unit : I

Number system and digital Logic families

1. Review of number system

Inside today's computer, data is represented as 1's and 0's. These 1's and 0's might be stored magnetically on a disk.

Modern digital systems do not represent numeric values using the decimal system. Instead they use a binary or two's complement numbering system.

The various number systems are,

→ decimal number system (0-9) base 10

→ Binary number system (0-1) base 2

→ Octal Number system (0-7) base 8

→ Hexadecimal number system. (0-15) base 16

→ BCD (Binary coded decimal).

⇒ Decimal Number system

The decimal number is composed of 10 symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. By using these symbols we can write any quantity.

The decimal system is also called base 10 system as it has '10' digits.

Table : Representation of decimal number.

$10^3 = 1000$	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$	.	$10^{-1} = 0.1$	$10^{-2} = 0.01$	$10^{-3} = 0.001$
Most significant digit (MSD)				decimal Point			Least significant digit (LSD)

where,

MSB → Left most digit which has greatest weight called most significant Bit.

LSB → Right most digit which has least weight called least significant Bit.

for example,

decimal number are 6897 and 27.95. The decimal number $(6897)_{10}$ is written as.

$$(6897)_{10} = 6\times10^3 + 8\times10^2 + 9\times10^1 + 7\times10^0.$$

or.

$$[6000 + 800 + 90 + 7.]$$

The decimal number $(27.95)_{10}$ can be represented as,

$$(27.95)_{10} = 2\times10^1 + 7\times10^0 + 9\times10^{-1} + 5\times10^{-2}.$$

or

$$[20 + 7 + 0.9 + 0.05]$$

⇒ Binary Number system

In binary number system, the digit are '0' and '1' with the base 2.

$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	.	$2^{-1}=1/2$	$2^{-2}=1/4$	$2^{-3}=1/8$.
Most significant Bit (MSB)				Binary Point			Least significant Bit (LSB).

Table : Representation of binary number.

for example ,

Represent binary number 1010101011_2 in power of 2 and its decimal equivalent.

Soln,

$$N = 1\times2^9 + 0\times2^8 + 1\times2^7 + 0\times2^6 + 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$$

$$= 512 + 0 + 128 + 0 + 3 + 0 + 8 + 0 + 2 + 1$$

$$= (654)_{10}.$$

⇒ **Octal Number**

The Octal number system has a base of eight, meaning it has eight possible digits 0, 1, 2, 3, 4, 5, 6 and 7. The octal number system is shown in table

$8^3 = 512$	$8^2 = 64$	$8^1 = 8$	$8^0 = 1$	.	$8^{-1} = 1/8$	$8^{-2} = 1/64$	$8^{-3} = 1/512$
Most significant digit (MSD)				Octal point			Least Significant digit (LSD)

Example :-

Represent octal Number 567 in power of 8 and find its decimal equivalent.

$$N = 5\times8^2 + 6\times8^1 + 7\times8^0$$

$$= 5\times64 + 6\times8 + 7$$

$$= (375)_{10}.$$

⇒ Hexadecimal Number system

In hexadecimal system the base is 16. So this system has 16 possible digit symbol.

It uses the digits 0 through 9 and and the letters A, B, C, D, E and F as the 16 digit symbols.

The letter A to F are used for the value of 10-15

$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$	.	$16^{-1} = 1/16$	$16^{-2} = 1/256$	$16^{-3} = 1/4096$
Most significant digit (MSD)				Hexa decimal Point			Least significant digit (LSD)

Example

Represent hexadecimal number 3FD in power of 16 and find its decimal equivalent.

$$N = 3 \times 16^2 + F \times 16^1 + D \times 16^0$$

$$= 3 \times 256 + 15 \times 16 + 13 \times 1$$

$$= (1021)_{10}.$$

decimal	Binary	Hexadecimal	octal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0110	5	5
6	0111	6	6
7	1000	7	7
8	1001	8	
9	1010	9	
10	1011	A	
11	1011	B	
12	1100	C	
13	1101	D	
14	1110	E	
15	1111	F	

Number Base conversion

⇒ Binary to decimal conversion.

Any binary number can be converted in to its decimal equivalent simply by summing together the weight of the various position in a binary number.

Problem.

1. Convert the binary number $(1110011)_2$ in to decimal

Soln

$$1\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0$$

$$= 32+16+8+0+2+1$$

$$= 59_{10} \text{ (decimal)}$$

2. Convert the binary number $(110110101)_2$ in to decimal

Soln,

$$1\times2^8+1\times2^7+0\times2^6+1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0$$

$$= 256+128+0+32+16+0+4+0+1$$

$$= 437_{10} \text{ (decimal)}$$

3. Convert the following binary number in to decimal number a) 1110011 b) 1101.11

a) 1110011

$$1\times2^6+1\times2^5+1\times2^4+0\times2^3+0\times2^2+1\times2^1+1\times2^0$$

$$= 64+32+16+0+0+2+1$$

$$= (115)_{10}$$

b) $(1101.11)_2$

$1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2}$.

$= 8 + 4 + 1 + 0.5 + 0.25$

$= (13.75)_{10}$.

⇒ Decimal to Binary conversion

The decimal number is converted in to binary equivalent.

Problem.

1. convert the following decimal number to binary numbers.

   a) 11 b) 255

a) Conversion of $(11)_{10}$ in to binary.

2	11	
2	5	— 1
2	2	— 1
	1	— 0

Result $11_{10} = 1011_2$

b) conversion of $(255)_{10}$ in to binary.

2	255	
2	127	— 1
2	63	— 1
2	31	— 1
2	15	— 1
2	7	— 1
2	3	— 1
	1	— 1

Result $(255)_{10} = (11111111)_2$

3. Convert 25_{10} in to binary.

2 | 25		
2 | 12	— 1	(LSB)
2 | 6	— 0	
2 | 3	— 0	
1	— 1	

Result of $25_{10} = (1\,1\,0\,0\,1)_2$

4. Convert 600_{10} in to binary

2 | 600		
2 | 300	— 0	(LSB)
2 | 150	— 0	
2 | 75	— 0	
2 | 37	— 1	
2 | 18	— 1	
2 | 9	— 0	
2 | 4	— 1	
2 | 2	— 0	
1 (MSB)	— 0	

Result of $600_{10} = (1\,0\,0\,1\,0\,1\,1\,0\,0\,0)_2$

5. Conversion of $(0.72)_{10}$ in to binary.

Fraction No	Product		Integer Part	
0.72	× 2	⇒ 1.44	⇒ 1	(MSB)
0.44	× 2	⇒ 0.88	⇒ 0	
0.88	× 2	⇒ 1.76	⇒ 1	
0.76	× 2	⇒ 1.52	⇒ 1	
0.52	× 2	⇒ 1.04	⇒ 1	
0.04	× 2	⇒ 0.08	⇒ 0	
0.08	× 2	⇒ 0.16	⇒ 0	
0.16	× 2	⇒ 0.32	⇒ 0	
0.32	× 2	⇒ 0.64	⇒ 0	
0.64	× 2	⇒ 1.28	⇒ 1	
0.28	× 2	⇒ 0.56	⇒ 0	
0.56	× 2	⇒ 1.12	⇒ 1	(LSB)

$(0.72)_{10} = (0.1010000101)_2$

6. Conversion of $(24.625)_{10}$ in to binary.

$(24.625)_{10}$ in to binary.

2	24	
2	12	— 0
2	6	— 0
2	3	— 0
	1	— 1

Result $(24)_{10} = (1\ 1\ 0\ 0\ 0)_2$

Fraction Number	Product	Fraction part	Integer part
0.625	$0.625 \times 2 = 1.25$	.25	1 (MSB)
0.25	$0.25 \times 2 = 0.5$	.5	0.
0.5	$0.5 \times 2 = 1$	0	1 (LSB)

Result : $(24.625)_{10} = (11000.101)_2$

7. Conversion of $(.475)_{10}$ in to binary.

Fraction Number	Product	Fractionpart	Integer part
0.475	$0.475 \times 2 = 0.95$	.95	0
0.95	$0.95 \times 2 = 1.9$	.9	1
0.9	$0.9 \times 2 = 1.8$	.8	1
0.8	$0.8 \times 2 = 1.6$	.6	1
0.6	$0.6 \times 2 = 1.2$	.2	1
0.2	$0.2 \times 2 = 0.4$	.4	0
0.4	$0.4 \times 2 = 0.8$	.8	0

Result : $(0.475)_{10} = (.01111100)_2$

⇒ Decimal to octal conversion

1. Convert the following decimal number to octal numbers

a) 234_{10} b) 2988.6875_{10}

a) conversion of $(234)_{10}$ in to octal

8 | 234		
8 | 29	——	2
3	——	5

Result : $234_{10} = 352_8$

b) conversion of $(2988.6875)_{10}$ in to octal.

8 | 2988		
8 | 373	——	4
8 | 46	——	5
5	——	6

Result : $2988 = (5654)_8$

conversion of 0.6875

Fraction Number	Product	Fractional part	Integer Part
0.6875	$0.6875 \times 8 = 5.5$	5	5
0.5	$0.5 \times 8 = 4.0$	0	4

Result : $2988.6875 = (5654.54)_8$

2. Conversion of 178_{10} to octal

$8 \mid 178$

$8 \mid 22$ —— 2

2 —— 6

Result : $178_{10} = 268_8$

⇒ Octal to decimal conversion

1. Convert the following octal number to decimal number.

a) 416_8 b) 360.15_8

a) 416_8 in to decimal

$4 \times 8^2 + 1 \times 8^1 + 6 \times 8^0$

$= 256 + 8 + 6$

$= 270_{10}$

b). 360.15_8 in to decimal.

$3 \times 8^2 + 6 \times 8^1 + 0 \times 8^0 + 1 \times 8^{-1} + 5 \times 8^{-2}$

$= 192 + 48 + 0.125 + 0.078125$

$= 240.203_{10}$.

2. Convert the octal number $(24.6)_8$ in to decimal number.

24.6_8 in to decimal

$2 \times 8^1 + 4 \times 8^0 + 6 \times 8^{-1}$

$= 16 + 4 + 0.75$

$= 20.75_{10}$.

⇒ Binary - to - octal

Binary Equivalent of octal digit

Octal digit	Binary Equivalent (4 2 1)
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

1. convert the binary number $(1111001111010)_2$ in to octal.

⇒ $(111\ 100\ 111\ 010)_2$

⇒ $(7\ 4\ 7\ 2)_8$

Result : $(1111001111010)_2 = (7472)_8$

2. Convert the following binary number to octal number.

a) 1011110011110.001100 b) .111001111

a) $(1011110011110.001100)_2$ in to octal number.

⇒ $(101\ 111\ 001\ 110\ .001\ 100)_2$

⇒ $(5\ 7\ 1\ 6\ .\ 1\ 4)_8$

Result : $(5716.14)_8$

b) $(.11100111)_2$ in to octal

$\Rightarrow (.111\ 001\ 111)_2$

$\Rightarrow (.717)_8$

Result : $(.717)_8$.

⇒ Octal to binary conversion

1. Convert the following octal number in to binary number

a) 370.526_8 b) 2702_8 c) 24657_8

a) 370.526_8

$370.526_8 \Rightarrow (011\ 111\ 000\ .\ 101\ 010\ 110)_2$.

b) 2702_8

$2702_8 \Rightarrow (010\ 111\ 000\ 010)_2$

c) 24657_8

$24657_8 \Rightarrow (010\ 100\ 110\ 101\ 111)_2$.

⇒ Decimal to Hexadecimal conversion

1. Convert the following decimal number in to Hexadecimal number.

a) $(905)_{10}$ b) $(6786)_{10}$

a) $(905)_{10}$

$$\begin{array}{r|l} 16 & 905 \\ 16 & 56 \quad — 9 \\ & 3 \quad — 8 \end{array}$$

Result : $(905)_{10} = 389_{16}$

b) $(6786)_{10}$ to hexadecimal.

16	6786	
16	424	2
16	26	8
	1	10

Result : $(6786)_{10} = 1\,10\,8\,2.$

$(6786)_{10} = 1A82_{16}.$

2. Convert the following decimal number to hexadecimal number.

a) 0.625_{10} b) 2824.725.

a) 0.625_{10} in to hexadecimal.

Fraction Number	Product	Fractional part	Integer part
0.625	0.625x16 = 10	.0.	A

Result : $0.625_{10} = A_{16}.$

b) 2824.725

16	2824	
16.	176	8
	11	0.

Result of $2824_{10} = B08_{16}.$

Fraction Number	Product	Fractional part	Integer part
0.725	0.725x16 = 11.6	.6	B
0.6	0.6 x16 = 9.6	.6	9

$0.725 = .B9_{16}.$

$2824.725 = (B08.B9)_{16}$

⇒ hexadecimal to decimal Number

1. convert the following hexadecimal number to decimal numbers.

   a) $F2C_{16}$ b) $DF8.28_{16}$

a) $(F2C)_{16}$

$F \times 16^2 + 2 \times 16^1 + C \times 16^0$

$= 15 \times 16^2 + 2 \times 16 + 12 \times 1$

$= 3840 + 32 + 12$

$= 3884_{10}$

Result : $(F2C)_{16} = 3884_{10}$

b) $DF8.28$

$D \times 16^2 + F \times 16^1 + 8 \times 16^0 + 2 \times 16^{-1} + 8 \times 16^{-2}$

$= 13 \times 16^2 + 15 \times 16 + 8 \times 1 + 2/16 + 8/16^2$

$= 3328 + 240 + 8 + 0.125 + 0.03125$

$= 3576.1562_{10}$

Result : $(DF8.28)_{16} = (3576.1562)_{10}$.

Binary Equivalent (8 4 2 1)	Hexadecimal digit
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

1. Convert the binary number to hexadecimal number.

101111001011_2

$101111\ 001011_2 = (1011)(1100)(1011)_2$

$= B\ C\ D_{16}$

2. Convert the binary number to hexadecimal number.

1011001110.01101111_2

$(0010)(1100)(1110)\cdot(0110)(1110)(1000)_2 = (2CE\cdot 6E8)_{16}$.

⇒ Hexadecimal to binary number.

1. Convert the following hexadecimal number to binary numbers.

a) $A4C_{16}$ b) 3E7.DA

a) $A4C_{16}$

$A4C_{16} = (1010\ 0100\ 1100)_2$.

b) $3E7.DA_{16}$

$3E7.DA_{16} = (0011\ 1110\ 0111.1101\ 1010)_2$

⇒ Convert the following octal to Hexadecimal number.

1. convert the following octal to Hexadecimal number.

a) 744_8 b) 3472.56_8

a) 744_8.

→ first convert this octal number in to binary.

$744_8 = (111\ 100\ 100)_2$.

→ forming into group of 4 bits

$(111\ 100\ 100)_2 = (0001\ 1110\ 0100)_2$

$= (1E4)_{16}$.

b) 3472.56_8

→ first convert octal number in to binary.

$3472.56_8 = (011\ 100\ 111\ 010.101\ 110)_2$

→ Forming in to group of 4 bits

$(011100111010.101110)_2 = (0111\ 0011\ 1010.1011\ 1000)_2$

$= (73A.B8)_{16}$.

⇒ Hexadecimal to octal

1. Convert the following hexadecimal number to octal number.

a) B7A4 b) D43E.5A

a) Conversion of $(B7A4)_{16}$ to octal

$(B7A4)_{16} = (1011\ 0111\ 1010\ 0100)_2$

forming into groups of 3 bits

$= (001\ 011\ 011\ 110\ 100\ 100)_2$

$= (133644)_8$

b) Conversion of $(D43E.5A)_{16}$ to octal

$(D43E.5A)_{16} = (1101\ 0100\ 0011\ 1110\ .\ 0101\ 1010)_2$

forming into groups of 3 bits.

$= (001\ 101\ 010\ 000\ 111\ 110\ .\ 010\ 110\ 100)_2$

$= (152076.264)_8$

Converting any Radix to Decimal.

In general numbers can be represented as.

$$N = A_{n-1} r^{n-1} + A_{n-2} r^{n-2} \ldots + A_1 r^1 + A_0 r^0 + A_{-1} r^{-1} + A_{-2} r^{-2} + \ldots C_{-m} r^{-m}$$

where,

$N \rightarrow$ Number in decimal

$A \rightarrow$ digit

$r \rightarrow$ Radix (or) base of a number system

$n \rightarrow$ The number of digit in integer portion of number

$m \rightarrow$ The number of digit in fraction portion of number.

From this general equation we can convert number with any radix in to its decimal equivalent.

Problem.

1. Convert binary number 1101.1 to its decimal equivalent

Soln,

$(1101.1)_2$ Convert in to decimal.

$$(1101.1)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}$$

$$= 8 + 4 + 0 + 1 + 0.5$$

$$= (13.5)_{10}.$$

Result : $(1101.1)_2 = (13.5)_{10}$.

Conversion of decimal Number to any Radix Number.

There are two steps in converting decimal number in to any Radix number.

⊗Step 1: Convert integer part by successive division method

⊗Step 2: convert fractional part by successive multiplication Method.

⊗Successive Division for Integer part conversion

→ In this method we repeatedly divide the integer part of the decimal number by r (ie the new radix) until quotient zero.

→ The remainder are taken in the reverse order to form a new radix number. [(ie) from MSD to LSD where MSD → last remainder value, LSD → first remainder value.

Problem:-

1. Convert decimal number 37 to its binary equivalent

⊗Soln,

eg: $37_{10} \rightarrow (\ )_2$

Here $r \rightarrow 2$.

2	37	
2	18	— 1
2	9	— 0
2	4	— 1
2	2	— 0
	1	— 0

Result: $37_{10} = (100101)_2$

2. Convert decimal Number 214 to its octal equivalent.

Soln,

$(214)_{10} \rightarrow (\quad)_8$.

8	214	
8	26	— 6
	3	— 2

Result : $(214)_{10} = (326)_8$

3. Convert decimal Number 3509 to its hexadecimal equivalent.

Soln,

$(3509)_{10} \rightarrow (\quad)_{16}$

16	3509	
16	219	— 5
	13	— 11

Result : $(3509)_{10} = (DB5)_{16}$.

4. Convert $(54)_{10}$ to radix 4

$(54)_{10} \rightarrow (\quad)_4$

4	54	
4	13	— 2
	3	— 1

Result $(54)_{10} = (312)_4$.

Complements

→ In binary system the base value is 2 (ie) $r = 2$ so it is referred as 2's complement and 1's complement

→ In decimal system the base value is 10 (ie) $r = 10$; so it is referred as 10's complement and 9's complement.

1's complement Representation

The 1's complement of the binary number is the number that result when we change all 1's to zeros and the zeros to one.

Example :-

1. Find the 1's complement of $(1101)_2$

Soln,

1101 gn number.

[change 0 to 1 and 1 to 0]

Ans :- 1's complement is 0010.

2. Find 1's complement of 10111001_2

Soln,

10111001 gn number.

Ans : 1's complement is 01000110

2's complement Representation

The 2's complement is the binary number that result when we add 1 to the 1's complement. It is given as,

2's complement = 1's complement +1

Example :-

1. Find 2's complement of $(1001)_2$

eg: 1001_2

1's complement is 0110

2's complement is = 1's complement + 1

$$= \begin{array}{r} 0110 \\ \quad\quad 1 \\ \hline 0111 \end{array} +$$

Ans :- 2's complement of $1001_2 = (0111)_2$

2. Find 2's complement of $(10100011)_2$

eg, $(10100011)_2$.

2's complement is = 1's complement + 1

1's complement = $(01011100)_2$

$$\text{2's complement} = \begin{array}{r} 01011100 \\ \quad\quad\quad 1 \\ \hline 01011101 \end{array} +$$

Ans :- 2's complement = $(01011101)_2$

1's complement substraction

→ Substraction of binary number can be accomplished by the direct method by using the 1's complement method, which allow to perform substraction using only addition

→ For substraction of two number we have two cases,

→ Subtraction of smaller number from larger number
→ Subtraction of larger number from smaller number

Subtraction of smaller number from larger number Method

(i) determine the 1's complement of the smaller number
(ii) Add the 1's complement to the larger number
(iii) Remove the carry and add it to the result

This is called end around carry.

Example

1. Subtract $(101011)_2$ (smaller) from $(111001)_2$ (larger) using the 1's complement.

Soln,

eg: $(101011)_2$ from $(111001)_2$

(i) Determine 1's complement of smaller number.

$(101011)_2 \rightarrow (010100)_2$.

(ii) Add 1's complement to the larger number

```
  010100
  111001
 1001101
```

carry → 1

(iii) Remove carry and add it to the result

```
  00110 1
      + 1
  00111 0
```

Ans:
$(101011)_2$ from $(111001)_2 = (001110)_2$

Substraction of larger from smaller number :-

(i) determine 1's complement of the larger number
(ii) Add the 1's complement to the smaller number
(iii) Answer is in 1's complement form. To get answer in true form take the 1's complement and assign negative sign answer.

Example :-

1. Substract $(111001)_2$ from $(101011)_2$ using 1's complement form.

Soln,

$\therefore$ $(111001)_2$ from $(101011)_2$

(i) Determine 1's complement of larger no.

$$(111001)_2 \rightarrow (000110)_2$$

(ii) Add 1's complement to smaller no

$$\begin{array}{r} 000110 \\ +\ 101011 \\ \hline 110001 \end{array}$$

(iii) Answer is in 1's complement form & assume -ive sign.

ie) -001110

Ans :- $(111001)_2$ from $(101011)_2 = -(001110)_2$

2's complement substraction :-

Like 1's complement substraction, in 2's complement substraction, the substraction is accomplished by only addition.

(i) Substraction of smaller Number from larger Number

(i) determine 2's complement of smaller number.
(ii) Add 2's complement to larger no.
(iii) discard the carry.

Example :-

1. Substract $(101011)_2$ from $(111001)_2$ using the 2's complement method.

eg,
$(101011)_2$ from $(111001)_2$.

(i) determine 2's complement of smaller number.

2's complement = 1's complement + 1

1's complement of $(101011)_2 = (010100)_2$

2's complement = 010100 + 1 = 010101

(ii) Add the 2's complement of Larger No

```
  010101
  111001
 ------
1001110
```

discard carry → 1

(iii) discard carry.

Ans :- $(101011)_2$ from $(111001)_2 = (001110)_2$

Substraction of larger number from smaller number

(i) determine 2's complement of larger no

(ii) Add 2's complement to smaller no

(iii) Answer is in 2's complement form. To get answer in true form take 2's complement and assign negative sign to answer.

Example

1. Substract $(111001)_2$ from $(101011)_2$ using 2's complement method.

eg 1

Substract $(111001)_2$ from $(101011)_2$ using 2's complement Method.

(i) determine 2's complement of larger no

2's complement = 1's complement + 1

1's complement of $(111001)_2 = (000110)_2$

$$\begin{array}{r} 2\text{'s complement} = 000110 \\ \quad\quad\quad\quad\quad\quad + \\ \quad\quad\quad\quad\quad\quad 1 \\ \hline 000111 \end{array}$$

(ii) Add 2's complement to smaller no.

$$\begin{array}{r} 000111 \\ 101011 \\ \hline 110010 \end{array}$$

(iii) Answer is in 2's complement form and assume -ve sign to the answer.

2's complement = 1's complement + 1

1's complement of $(110010)_2 = (001101)_2$

$$\begin{array}{r} 2\text{'s complement} = 001101 \\ \quad\quad\quad\quad\quad\quad + \\ \quad\quad\quad\quad\quad\quad 1 \\ \hline -001110 \end{array}$$

Ans: $(111001)_2$ from $(101011)_2 = -(001110)_2$

Binary Arithmetic

Binary Addition

Binary addition consists of four possible operation as shown in below table.

S.No	Operation
1.	$0+0=0$
2	$0+1=1$
3	$1+0=1$
4	$1+1=10_2$

Problem

1. Perform addition of $(11001100)_2$ and $(11011010)_2$

Soln,

```
  1   1 1
  1 1 0 0 1 1 0 0
  1 1 0 1 1 0 1 0
-----------------
1 1 0 1 0 0 1 1 0
```

Result : 110100110_2.

2. perform addition of $(1001001)_2$ and $(0011001)_2$

Soln,

```
    1 1     1
  1 0 0 1 0 0 1
  0 0 1 1 0 0 1
---------------
  1 1 0 0 0 1 0
```

Result : 1100010_2.

3. Add 28_{10} and 15_{10} by converting them in to binary

4. Perform the following addition operation

a) $(275.75)_{10} + (37.875)_{10}$

b) $(AF1.B3)_{16} + (FFF.E)_{16}$

a) $(275.75)_{10} + (37.875)_{10}$

At first convert $(275.75)_{10}$ and $(37.875)_{10}$ in to its binary equivalent.

$(275.75)_{10} = 100010011.11_2$

$(37.875)_{10} = 100101.111_2$

$$\begin{array}{rl} & \overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1} \\ 275.75 = & 100010011.110 \\ 37.875 = & 000100101.111 \\ \hline 313.625_{10} = & 100111001.101 \end{array}$$

The decimal equivalent of $100111001.101 = 313.625_{10}$

b) $(AF1.B3)_{16} + (FFF.E)_{16}$.

Convert $(AF1.B3)_{16}$ and $(FFF.E)_{16}$ in to its binary equivalent.

$(AF1.B3)_{16} = (1010\ 1111\ 0001\ .\ 10110011)_2$

$(FFF.E)_{16} = (1111\ 1111\ 1111\ .\ 1110)_2$

$$\begin{array}{l} \overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1}\overset{1}{1} \\ 1010\ 1111\ 0001\ .\ 10110011 \\ 1111\ 1111\ 1111\ .\ 1110\ 0000 \\ \hline 11010111100001.10010011 \end{array}$$

Hexa decimal equivalent of $(11010111100001.10010011)_2$ is $(1AF1.93)_{16}$.

Binary substraction

The substraction consists of four possible elementary operations, as shown in table.

Sl.No	Operation
1.	0 - 0 = 0
2.	0 - 1 = 1
3.	1 - 0 = 1
4	1 - 1 = 0

Problem

1. perform $(1110\,1100)_2 - (0011\,0010)_2$

Soln,

$$\begin{array}{r} {}^{0}1\,{}^{10}1\,{}^{10}1\,{}^{0}0\,{}^{10}0 \\ 0\,0\,1\,1\,0\,0\,1\,0 \\ \hline 1\,0\,1\,1\,1\,0\,1\,0 \end{array}$$

Result : 10111010_2.

2. Substract $(1110.011)_2$ from $(11011.11)_2$

Soln,

The decimal equivalent of 1110.011_2 and 11011.11_2 are 14.375 and 27.75.

$$\begin{array}{r} {}^{0}1\,{}^{10}1\,{}^{10}0\,1\,1\,.\,{}^{0}1\,{}^{10}1\,{}^{10}0 \\ 0\,1\,1\,1\,0\,.\,0\,1\,1 \\ \hline 0\,1\,1\,0\,1\,.\,0\,1\,1 \end{array}$$

Result : 01101.011_2.

---

3. Substract 11001010 and -10011011

Ans: 101111

Binary Multiplication

Binary Multiplication is same as the decimal number. and binary multiplication is simple than decimal multiplication since it involves only 1s and 0s.

The table shows rules for binary multiplication.

S.No	Operation
1.	0 x 0 = 0
2.	0 x 1 = 0
3	1 x 0 = 0
4	1 x 1 = 1

Problem

1. Multiply $(1011010)_2$ and $(11011)_2$ using binary Multiplication method.

```
    1 0 1 1 0 1 0
  x   1 1 0 1 1
  ---------------
    1 0 1 1 0 1 0
  1 0 1 1 0 1 0
0 0 0 0 0 0 0
1 1 0 1 1 0 1 0
1 0 1 1 0 1 0
-----------------------
1 0 0 1 0 1 1 1 1 1 1 0
```

Result : 100101111110_2.

2. perform $(1100)_2 \times (1010)_2$ is

```
    1 1 0 0
    1 0 1 0
  ---------
    0 0 0 0
  1 1 0 0
0 0 0 0
1 1 0 0
-------------
1 1 1 1 0 0 0 0
```

Result : 11110000_2.

Binary Division

The division process for binary numbers is similar to the decimal numbers.

S.No	Operation
1.	$0 \div 1 = 0$
2.	$1 \div 1 = 1$

Problem

1. Divide 1101101_2 by 110_2.

```
        100100
    ____________
110 ) 1101101
      - 110
      -------
        00011
          110
          ---
          00011
```

2. Divide 100001_2 by 110_2

```
        101.1
    ____________
110 ) 100001
      - 110
      -------
         1001
          110
          ---
           110
           110
           ---
             0
```

Binary codes

* When numbers or words are represented by a specific group of symbols, we can say that they are encoded.
* The group of symbols used to encode them are called codes. The digital data is represented, stored and transmitted as group of binary digits (bits)
* The group of bits also known as binary codes.

Classification of Binary code

The codes are broadly classified as,

1. Weighted codes.
2. Non weighted codes
3. Reflective codes.
4. Sequential codes
5. Alphanumeric codes
6. Error detecting and correcting codes.

1. Weighted codes.

In weighted code, each digit position of the number represent a specific weight.

→ For example, in decimal code, if the number is 567 then weight of 5 is 100, weight of 6 is 10 and weight of 7 is 1.

→ In weighted binary code each bit has a weight 8, 4, 2 or 1 and each decimal digit is represented by a group of four bit. There also some weighted

code namely 4221, 2421 and 5211.

→ The weight code 8421 represent the weight of the bit position.

Decimal digit	weighted BCD codes			
	8421	4221	2421	5211
0	0000	0000	0000	0000
1	0001	0001	0001	0001
2	0010	0010	0010	0011
3	0011	0011	0011	0101
4	0100	1000	0100	0111
5	0101	1001	0101	1000
6	0110	1100	1100	1010
7	0111	1101	1101	1100
8	1000	1110	1110	1110
9	1001	1111	1111	1111

Table :- weighted BCD code

Non-weighted codes

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

→ Excess-3 and gray code are the non-weighted codes.

Reflective code

→ A code is said to be reflective when the code for 9 is the complement for 0, the code for 8 is complement for 1, 7 for 2, 6 for 3 and 5 for 4.

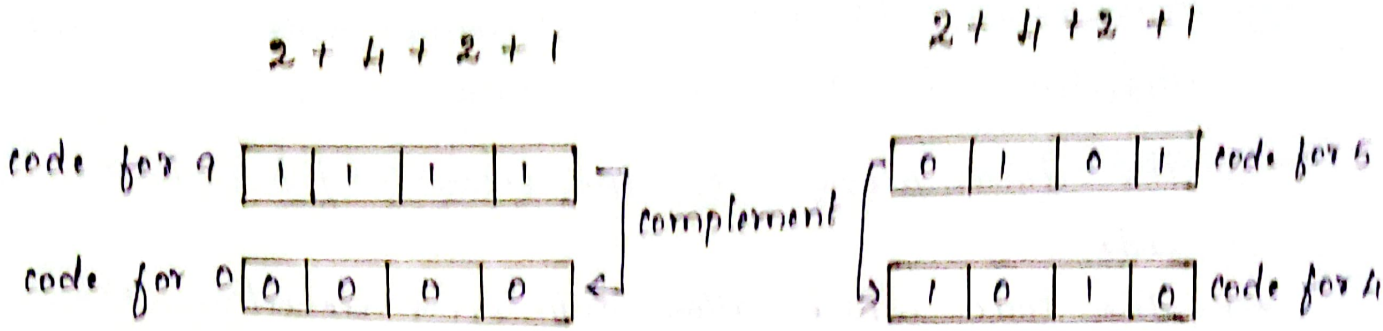


Fig: Reflective code.

Sequential code :-

→ In sequential codes each succeeding code is one binary number greater than its preceding code

→ The 8421 and excess-3 are sequential, whereas the 2421 and 5211 codes are not.

Alphanumeric code :-

→ The code which contain of both numbers and alphabetic characters are called alphanumeric code.

→ The most commonly used alphanumeric codes are :
ASCII (American standard code for information interchange)
EBCDIC (Extended Binary coded decimal Interchange code)

Error detecting and correcting code.

→ When the digital information in the binary form is transmitted from one circuit or system to another circuit or system an error may occur. This means a signal corresponding to 0 may change to 1 or vice versa due to presence of noise.

→ To maintain the data integrity between transmitter and receiver, extra bit or more than one bit are added in the data.

→ These extra bits allow the detection and sometimes correction of error in the data.

→ The data along with the extra bit / bits forms the code.

→ The code which allow only error detection are called error detecting code.

→ The code which allow error detection and correction are called error detecting and correcting code.

Binary coded Decimal code (BCD) code.

→ BCD is a numeric code in which each digit of a decimal number is represented by a separate group of 4 bit. The most common BCD code is 8 4 2 1 BCD

Advantage :-

1. Easy to convert between it and decimal.

Disadvantage :-

1. Arithmetic operation are more complex.

BCD Addition

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digit are added.

Case 1 : Sum equals 9 or less with carry 0

→ Consider addition of 3 and 6 in BCD

	1 1	
6	0 1 1 0	BCD for 6
+ 3	0 0 1 1	BCD for 3
9	1 0 0 1	BCD for 9

→ addition of 5 and 2 in BCD.

5	0 1 0 1	BCD for 5
+ 2	0 0 1 0	BCD for 2
7	0 1 1 1	BCD for 7

The addition is carried out as in normal binary addition and the sums are 1001 and 0111 which are valid BCD codes.

Case 2 :- Sum greater than 9 with carry 0

Let us consider addition of 6 and 8 in BCD

6	0 1 1 0	BCD for 6.
+ 8	1 0 0 0	BCD for 3.
14	1 1 1 0	Invalid BCD number (1110) > 9

The Sum 1110 is an invalid BCD number. This has occured because the sum of the two digits exceed 9. Whenever this occur the sum is corrected by the addition of six (0110) in the invalied BCD number as shown in follow.

```
   6      0 1 1 0    BCD for 6
 + 8      1 0 0 0    BCD for 8
 ----   ---------
  14      1 1 1 0    Invalied BCD number
          0 1 1 0    Add 6 for correction
        ---------
        1 0 1 0 0
```

```
0 0 0 1 | 0 1 0 0    BCD for 14
---------------------
    1       4        decimal.
```

Case(iii) Sum equal 9 or less with carry 1

Consider addition of 8 and 9 in BCD

```
   8      1 0 0 0    BCD for 8
 + 9      1 0 0 1    BCD for 9
 ----   ---------
  17    1  0 0 0 1
      carry
      -------------
      0 0 0 1 | 0 0 0 1
      -------------
        1         1
```

] incorrect BCD result.

→ In this case 0001 0001 is incorrect BCD number. To get correct BCD result correction factor of 6 has to be added to the least significant digit sum

```
   8      1 0 0 0    BCD for 8.
 + 9      1 0 0 1    BCD for 9
 ----   ---------
  17    1 0 0 0 1    Incorrect BCD result.
```

```
  0 0 0 1 0 0 0 1
  0 0 0 0 0 1 1 0    Add 6 for correction
-----------------
  0 0 0 1 0 0 0 1    BCD for 17.
```

Procedure of BCD Addition

BCD addition procedure are as follow.

1. Add two BCD number using ordinary binary addition
2. If four bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.
3. If the four bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalied.
4. To correct the invalid sum, add 6 (0110_2) to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

Problem

1. Perform each of the following decimal addition in 8421 BCD

a) $+\begin{array}{r}24\\18\end{array}$ b) $+\begin{array}{r}48\\58\end{array}$

a) $+\begin{array}{r}24\\18\end{array}$

$$\begin{array}{rll} +\begin{array}{r}24\\18\end{array} & 0010 \quad 0100 & \text{BCD for } 24 \\ \hline 42 & 0001 \quad 1000 & \text{BCD for } 18 \\ \hline & 0011 \quad 1100 & \text{Invalid BCD number} \\ + & \quad\quad 0110 & \text{Add 6 for correction} \\ \hline & 0100 \quad 0010 & \end{array}$$

4 2.

b) 48 +
58

+ 48	0100 1000	BCD for 48
58	0101 1000 +	BCD for 58
106	1010 0000	BCD invalied
	0110	Add 6 for correction
	1010 0110	
	106.	

Excess - 3 code (XS 3 code)

→ The Excess - 3 code (XS 3 code) is a non-weighted code and it is used with binary coded decimal (BCD) numbers.

→ This code is an important 4-bit code and each 4-bit code represent a specific decimal digit.

Table show the BCD code and XS 3 code for decimal digit from 0 through 9.

Decimal digit	BCD code	XS 3 code.
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

To convert any decimal number in to its XS3 form, we add 3 to the decimal number and then the sum is converted in to BCD number.

Example of conversion of decimal number 5 in to XS3 code.

{ Initially add 3 with the decimal number 5 }

Decimal number 5

Add 3

Sum. 8

Then we convert the sum to BCD form. So the decimal number 5 represent 1000 in Excess 3 code.

→ In excess-3 code we get 9's complement of a number by just complementing each bit. Due to this excess-3 code is called self-complementing code or reflecting code

Problem :-

1. Determine the XS3 equivalent of the following decimal numbers. a) 345 b) 698.

a) 345

The XS_3 equivalent of 345 is 0110 0111 1000

b) 698

The XS3 equivalent of 698 is 1001 1100 1011

2. Find the excess-3 code and its 9's complement for following decimal numbers.

a) 592 b) 403.

a) 592

The XS3 equivalent of 592 is 1000 1100 0101

9's complement of 592_{10} is 0111 0011 1010

b) 403

The XS3 equivalent of 403 is 0111 0011 0110

9's complement of 403 is 1000 1100 1001

Excess-3 Addition

To perform excess-3 addition we have to

→ Add two excess-3 number using binary addition.

→ If carry = 1 → add 3 $(0011)_2$ to the sum of two digits

0 → substract 3 $(0011)_2$ from the sum.

3. Perform the excess-3 addition of a) 8,6 b) 1, 2.

a) 8, 6

```
                  1 1
Excess-3 for 8 :  1 0 1 1
Excess-3 for 6 :  1 0 0 1
                ,'1|0 1 0 0     carry = 1 hence
                  0 0 1 1       Add 3
                ---------
                  0 1 0 0 | 0 1 1 1
```

1 4. Result in decimal.

b) 1, 2.

```
                  '
Excess-3 for 1 :  0 1 0 0.
Excess-3 for 2 :  0 1 0 1
                ---------
                0.|1 0 0 1      carry = 0 hence
                  0 0 1 1       Sub 3.
                ---------
                  0 1 1 0
                ---------
                  3             Result in decimal.
```

Excess-3 Substraction

To perform excess-3 Substraction we have to

→ complement the subtrahend.

→ Add complement the subtrahend to minuend.

* If carry = 1 Result is positive. Add 3 $(0011)_2$ and end-around carry.

* If carry = 0 Result is negative. Substract 3 $(0011)_2$.

4. Perform the excess-3 substraction of a) 8-5 b) 5-8

a) 8-5

		1	1	1	
Excess-3 for 8	-	1	0	1	1
complement of 5 in excess 3	-	0	1	1	1
	1	0	0	1	0
		0	0	1	1
					Add 3.
		0	1	0	1
				1	Add end-around carry.
		0	1	1	0
					Excess-3 for 3.

b) 5 - 8

Excess-3 for 5		1	0	0	0
complement of 8 in excess-3	+	0	1	0	0
	0	1	1	0	0
		0	0	1	1
					Substract 3.
		1	0	0	1
					→ Excess-3 for -3

Gray code (Reflected code)

Gray code is a non-weighted code and is a special case of unit-distance code

→ In unit-distance code, bit patterns for two consecutive numbers differ in only one bit position. These codes are also called cyclic code.

→ The table shows the bit pattern assigned for gray code from decimal 0 to decimal 15

Decimal code.	gray code
0	0000.
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Error Detection and correction codes

→ When the binary is transmitted from circuit or system to another circuit or system an error may occur.

(i) The binary digit 0 may change to 1 or vice-versa due to presence of noise

→ To maintain the data between transmitter and receiver extra bit or more than one bit are added in the data.

→ These extra bit allow the detection and sometimes correction of error in the data

→ The data along with extra bit forms the code.

→ code which allow only error detection are called error detecting code and code which allow error detection and correction are called error detecting and correcting code.

Parity Bit

→ during transmitting data from one system to another system an incorrect code will be received at receiving end. This data lost may be occur due to electrical disturbance during data transmission, unacceptable humidity level, and dust particle.

→ To overcome this problem an extra bit is added with data for detecting error. This extra bit is called parity bit.

→ There are two types of parity bit namely,

* Even parity
* odd parity

→ In even parity, the parity bit is selected in such a way that even numbers of one are in word.

→ In odd parity, the parity bit is selected so that total number of one is an odd number.

* The message, including the parity bit is transmitted and then checked at the receiving end for errors.
* An error is detected if the checked parity does not correspond with the one transmitted.

→ The circuit that generate the parity bit in the transmitter is called parity generator.

→ The circuit that check the parity in the receiver is called a parity checker.

Note:-
In even parity the added parity bit will make the total number of 1s an even amount. In odd parity the added parity bit will make the total number of 1s an odd number.

3-bit Message			Message with odd Parity		Message with Even Parity	
A	B	C	Message	Parity	Message	Parity
0	0	0	000	1	000	0
0	0	1	001	0	001	1
0	1	0	010	0	010	1
0	1	1	011	1	011	0
1	0	0	100	0	100	1
1	0	1	101	1	101	0
1	1	0	110	1	110	0
1	1	1	111	0	111	1

Example :

1. When Ascii code is 1001111, a parity bit 1 is added with data to make it even number of ones. Then the new code after addition of parity bit is 11001111

Ascii code : 1 0 0 1 1 1 1

Data : 1 1 0 0 1 1 1 1

↳ Parity bits

2. The Ascii code is 1000100 a parity bit 0 is added with data as even number of ones are already present in data. After addition of parity bit, the code has been changed to 01000100

Ascii code : 0 1 0 0 0 1 0 0

Data : 0 0 1 0 0 0 1 0 0

↳ Parity bit

Hamming code .

Binary data is transmitted through any communication medium such as optical fiber cable or radio waves. If any noise is introduced in a communication system there will be some error between the transmitted and received data.

Therefore an error detecting code should be used to detect error during data transmission.

→ Parity bit checking is commonly used to detect error, but it cannot correct the error.

→ The error can be corrected by error correcting code which is known as Hamming code

" The Hamming code is called detecting and correcting code "

The Hamming code was developed by R.W Hamming. In this code, one or more parity bit are added to the data in such a way that error can be detected and corrected.

Number of Parity Bits.

Number of Parity bit depends on the number of information bit.

The number of parity bit P is determine by the following relationship

$$2^P \geq x + P + 1 \quad \text{—— (1)}$$

x → Number of information bit

P → Number of Parity bit

{ P → value is found by trial and error method }

Example :-

Consider four information bit, ie $x = 4$, then P is found by trial and error method.

Let P = 2 then

$$2^P = 2^2 = 4$$

and

$$x + P + 1 = 4 + 2 + 1 = 7$$

Since 2^P must be equal or greater than $x + P + 1$ in eqn (1)

So, $4 \geq 7$ is not satisfied, Hence assume another value for P, Let P = 3.

$$2^P = 2^3 = 8$$

and

$$x + P + 1 = 4 + 3 + 1 = 8$$

Hence.

$8 = 8$, equation (1) is satisfied

→ Hence we can say that three parity bits are required to provide single error correction for four information bits.

Location of the parity Bits in the code

→ Now we know how to calculate the number of parity bit required to provide single error correction for given number of information bits.

→ Now we must know the position of the parity bit.

Example :-

We assume 4 information bit and 3 parity bits. Therefore the code is of seven bits.

Bit 7, Bit 6, Bit 5, Bit 4, Bit 3, Bit 2, Bit 1

The parity bits are located in the position corresponding to ascending power of two

Therefore for 7 bit code, location for parity bits and information bits are shown below.

D_7 , D_6 , D_5 , P_4 , D_3 , P_2 , P_1

where,

P_n → particular parity bit.

D_n → particular information bit.

Assigning values to parity Bit.

Let us see how to determine 1 or 0 value to each parity bit.

To do this, we must write the binary number for each decimal location number.

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Information bits (D_n)							
Parity bits (P_n)							

Assignment of P_1

Looking at the above table. We can see that the binary location number of parity bit P_1 has a 1 for its right-most digit. This parity bit checks all bit location, including itself that have 1s in the same location in the binary location numbers.

Therefore, parity bit P_1 checks bit locations 1, 3, 5 and 7 assign P_1 according to even or odd parity. For even parity Hamming code, it assign P_1 such that bit location 1, 3, 5 and 7 will have even parity.

Assignment of P_2:

Looking at the table. we can see that the binary location of parity bit P_2 has a 1 for its middle bit. This parity bit checks all the bit locations, including itself, that have 1s in the middle bit. Therefore parity bit P_2 checks bit location 2, 3, 6 and 7 and assigns P_2 according to even or odd parity.

Assignment of P_4

Looking at the table we can see that the binary location number of parity bit P_4 has a 1 for its left most digit. This parity bit checks all bit location, including itself, that have 1s in the left most bit. Therefore, parity bit P_4 check bit location 4, 5, 6 and 7 and assign P_4 according to even and odd parity.

Problem

1. Encode the binary word 1011 in to Seven bit even parity Hamming code.

Soln,

Step: 1 : Find the number of parity bit required.

Let P= 3 then.

$2^P : 2^3 : 8$

$x+P+1 : 4+3+1 : 8$

Three parity bit are sufficient

∴ Total code bits : 4+3 = 7.

Step : 2 :- Construct a bit location table.

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Information bits	1	0	1		1		
Parity bits							

Step 3 :- Determine the parity bits.

For P_1 : Bit location 3, 5 and 7 have three 1s and therefore to have an even parity P_1 must be 1

For P_2 : Bit location 3, 6 and 7 have two 1s and therefore to have an even parity P_2 must be 0

For P_3 : Bit location 5, 6 and 7 two 1s. there fore to have an even parity P_4 must be 0.

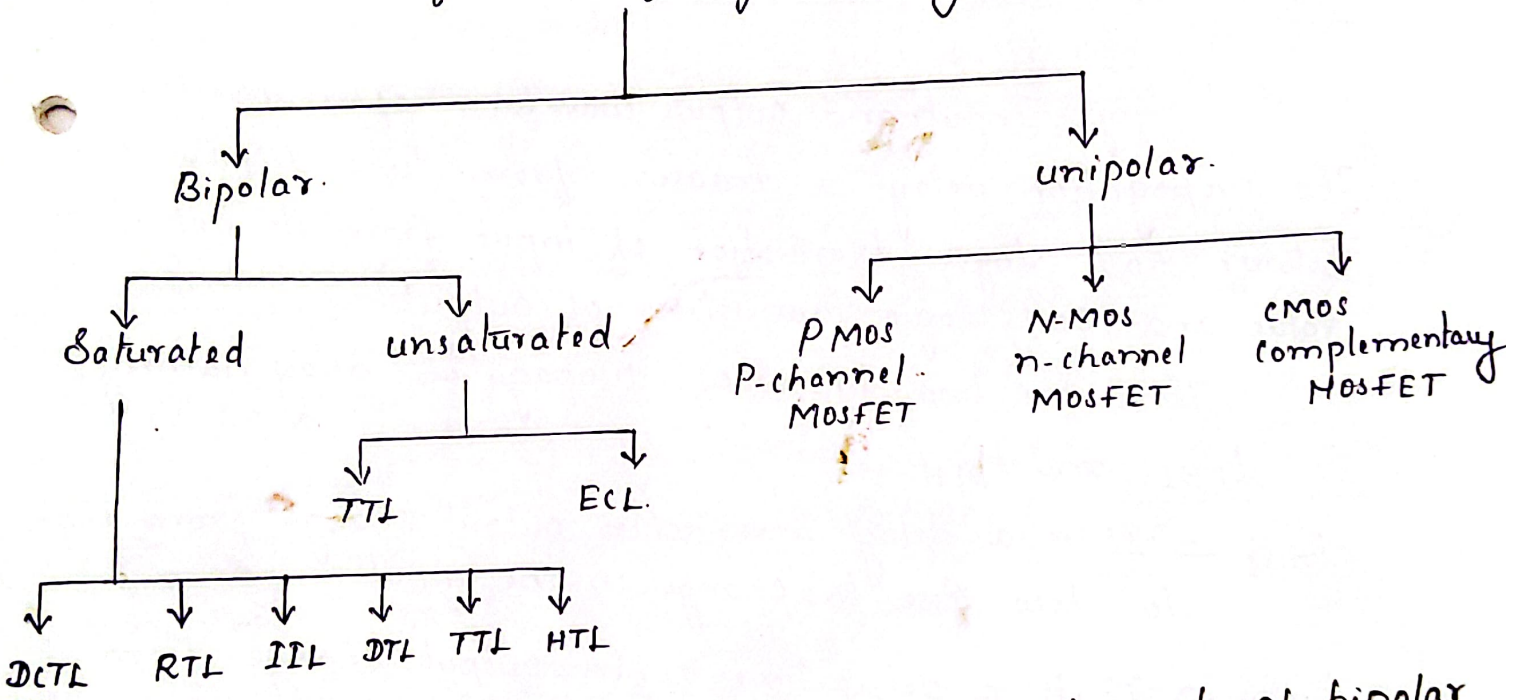
Step : 4 :- Enter the parity bits in to the table to form a Seven bit Hamming code = 1010101

digital Logic families

All modern system use digital Integrated circuit. Because they are reduction in weight and size. Most logic families are fabricated as an (Ic). The logic gates can be designed in different Method.

According to the component used in digital logic family it is classified in to two types they are.

Classification of Logic family



→ Transistor, diode, resistors are main element of bipolar logic family.

→ MOSFET is used in unipolar logic family.

Characteristic of digital Logic family

Most important performance parameter of digital logic families are given below.

1. Propagation delay

The speed of operation is specified in terms of propagation delay.

→ propagation delay is basically the time interval between the application of input pulse and the occurence of the resulting output pulse.

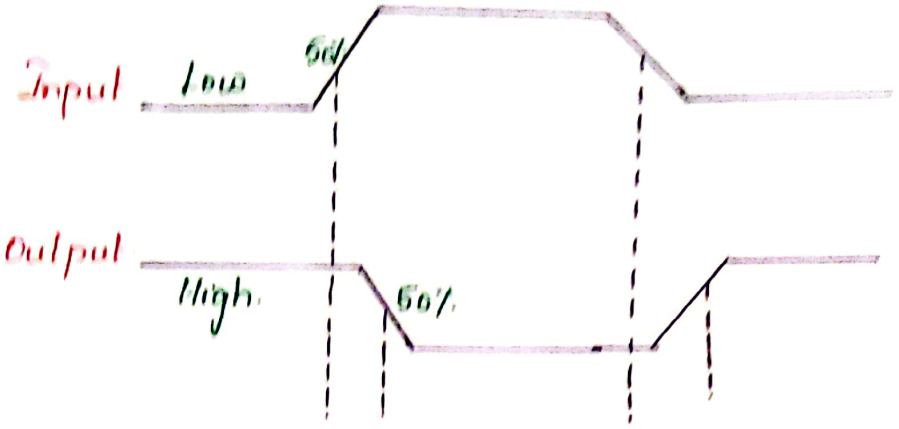


fig :- Input and output waveform of an inverter.

→ The propagation delay is measure from time difference between 50% logic transition of input from its initial value and 50% logic transition of output.

There are two types of propagation delay namely, t_{PHL} and t_{PLH}.

t_{PHL} → It is a delay time when output change from high to low due to change in the input.

t_{PLH} → It is a delay time when output change from low to high.

2. Power dissipation

The power dissipation is actually the power required to operate a gate.

→ Average power dissipated is a product of dc supply voltage and mean current.

$$P_D = I_{cc} \times V_{cc}.$$

I_{cc} → average supply current.

The power delivered to the gate from the power supply is expressed in milliwatt.

3. Voltage parameter

The following voltage are very important in designing digital system.

$V_{IH(min)}$ → High Level input Voltage :-

It is a minimum voltage level required for a logical 1 at a input. Any voltage below this level will not accepted as a High by a logic circuit. It is a minimum of 2 V.

$V_{IL(max)}$ → Low Level input Voltage

It is the maximum voltage level required for a logic 0 at an input. Any voltage above this level will not be accepted as a low by the logic circuit. It is a minimum of 0.8 V.

$V_{OH(min)}$ → High level output Voltage.

It is min voltage level that requires for a logical 1 as its output. It is a minimum of 2.4 V.

$V_{OL(max)}$ → Low Level output Voltage.

It is a maximum output voltage for logic 0. It is a minimum of 0.4.

4. Current parameter.

I_{IH} → High level input current

→ I_{IH} is the current flow in to an input when high level or logic 1 voltage is applied to that input.

I_{IL} → Low Level input current.

I_{IL} is the current flow in to an input when a low level (or) logic 0 voltage is applied to that input.

I_{OH} → High level output current.

I_{OH} is maximum current that flow from an output in the logic 1 state under specified loaded condition

I_{OL} → Low Level output terminal.

→ The current that flow from an output in the logic 0 state under specified load condition.

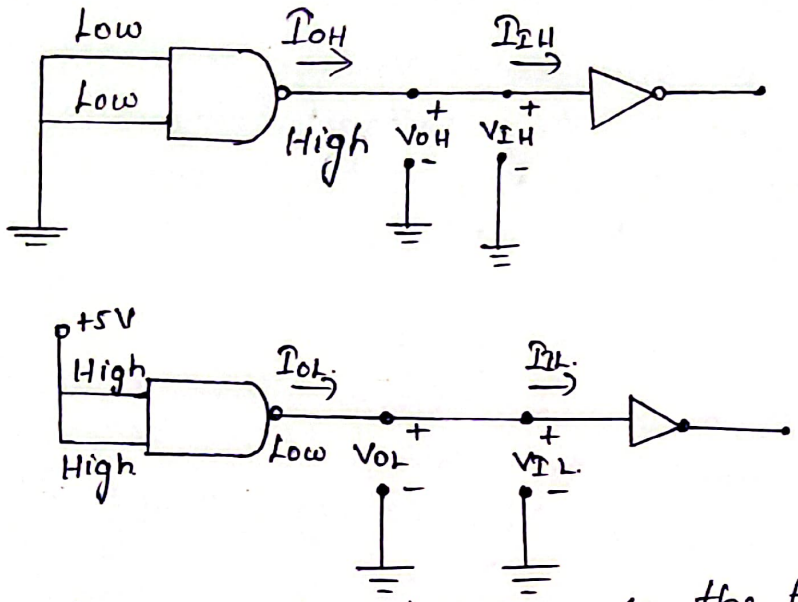


fig :- current and voltage in the two logic states.

5. Noise Margin

Noise is always present in electronic circuit due to stray electric and Magnetic field.

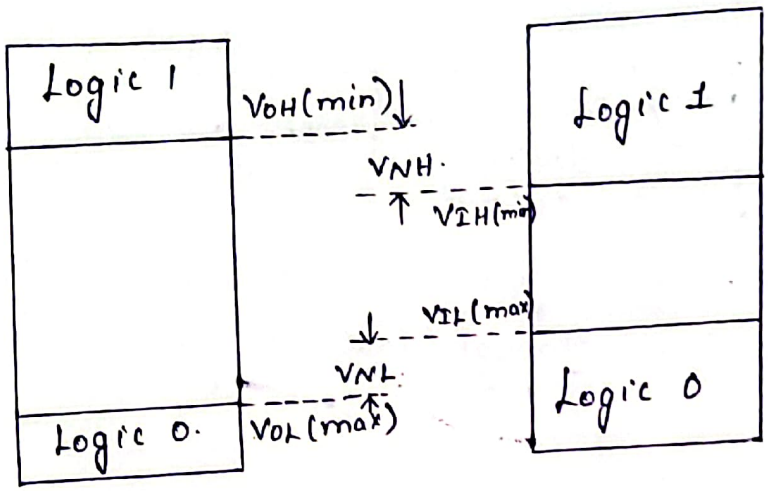
→ If noise in the circuit is high enough it can push a logic 0 up or drop a logic 1 down in to a illegal region.

"The ability of a circuit to tolerate the effect of noise is called noise immunity"

"The amount by which a circuit can tolerate the effect of noise is called noise Margin".

There are two types of Noise Margin they are,

1) Low Noise Margin
2) High Noise Margin



* Low Noise Margin

$V_{NL}(Low) = V_{IL}(max) - V_{NL}(max)$

* High Noise Margin.

$V_{NL}(High) = V_{OH}(min) - V_{IH}(min)$.

6. fan out.

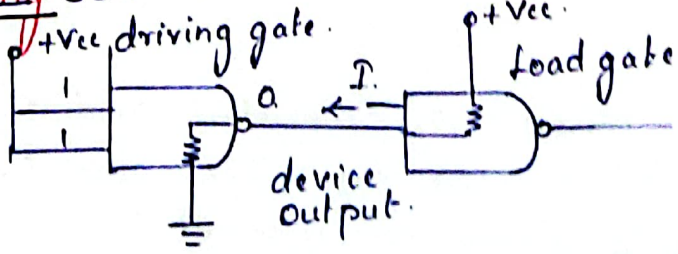
The Maximum number of input of several gate that can be driven by the output of a logic gate is decided by the parameter called fan out.

7. fan in

The fan in of a digital logic gate refer to a number of input.

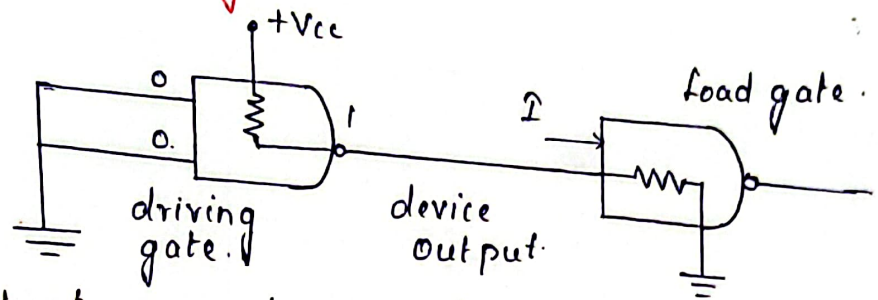
* 2 input for NOR gate has a fan in of 2.
* 4 input for Nand gate has a fan in of 4.

8. Current sinking source



The current flow from the power supply through the load and through device output to ground.

9. Current sinking



when current flow from the power supply out of the device output through load to ground.

Resistor - Transistor Logic (RTL)

Resistor Transistor Logic has become not in use now, but because of its simplicity and historical reason some attention is given to it.

Circuit Description

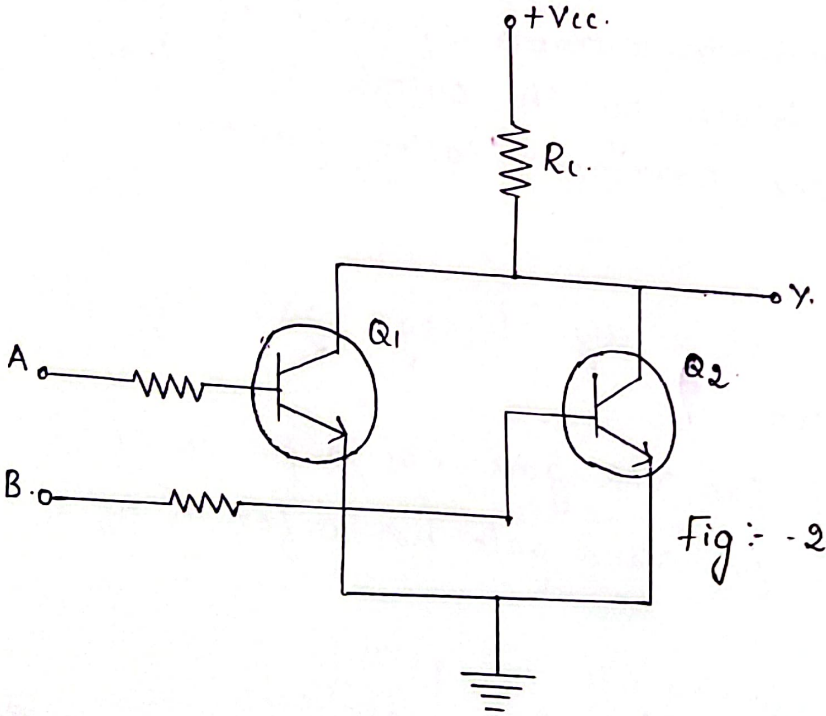


fig :- 2-input RTL NOR gate.

→ RTL circuit consists of resistors and transistor.
→ Emitter of both the transistor are tied together and connected to a common ground.
→ Collector of both transistors are tied through a common collector resistor Rc to supply voltage Vcc.
→ R_c is known as pull up resistor.

Circuit operation

→ A and B are the input terminal.

→ when input voltage are low level (Logic 0) it is low enough for the corresponding transistor to cut off.

→ when input voltage are high level (Logic 1) it is high enough for the corresponding transistor to saturation.

→ When both the input are low (Logic 0 for A and B) the transistor Q_1 and Q_2 are cut off and the output is high.

→ When both the input are high (Logic 1 for A and B) the transistor Q_1 and Q_2 leads to saturation causing the output go low.

A	B	Y (output)
0	0	1
0	1	0
1	0	0
1	1	0

fig :- Truth table for 2-input NOR gate.

Performance parameter for RTL.

1. Power Dissipation :- The average power dissipation is 30-100mw
2. propagation Delay :- The speed of operation is 12ns-25ns.
3. Noise Margin :- 0.2 volts.
4. Fan out :- 4.

Diode - Transistor Logic (DTL)

It is some what complex than RTL but because of its greater fan-out and improved noise Margin it has replaced RTL.

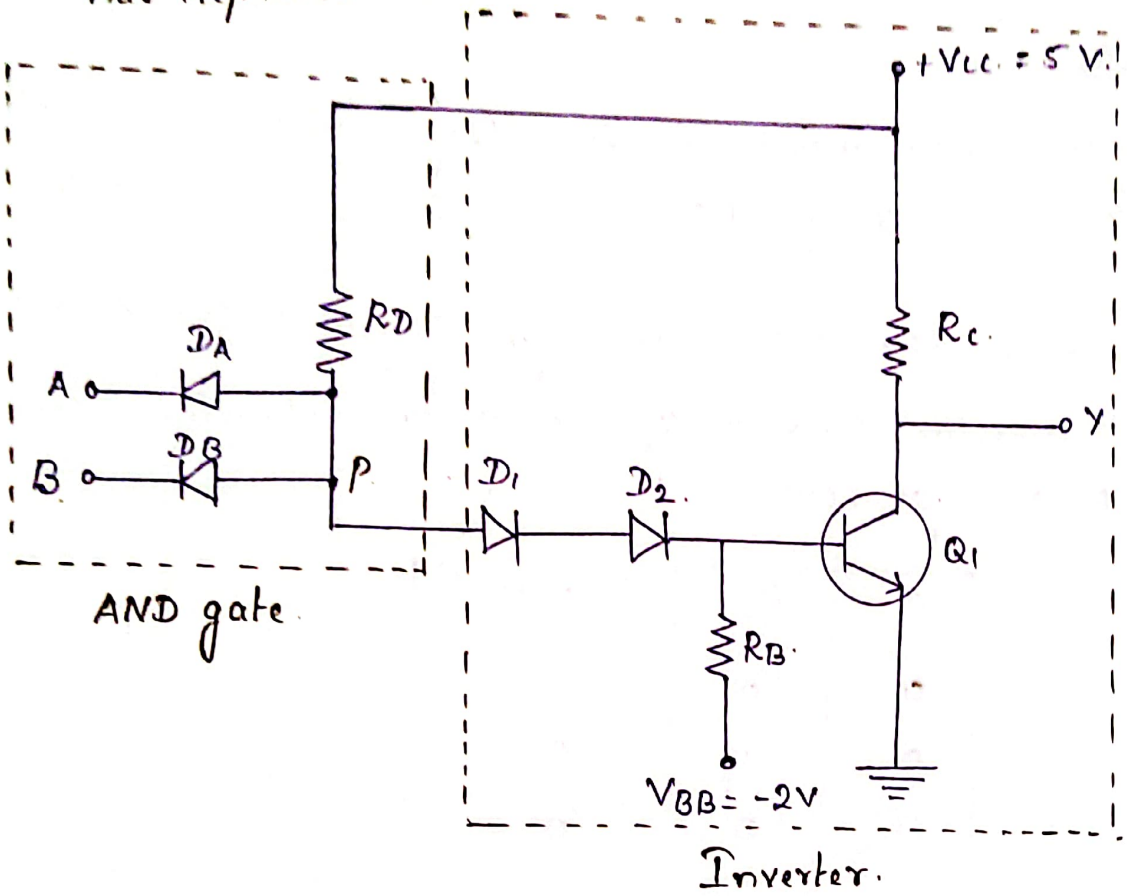


Fig :- 2-Input DTL NAND gate.

→ when both the input are low diode D_A and D_B are forward biased and diode conduct resulting 0.7 volts. This 0.7 voltage is not satisfied to drive the transistor Q_1. Therefore Q_1 is cut off giving output voltage $V_o = V_{cc}$ (ie) logic 1.

→ when both the input are high the diode D_A and D_B are reversed biased. This cause the base current of transistor Q_1 to flow through R_D, D_1, D_2 and the base of transistor, Q_1

Circuit Description

→ This drive the transistor Q_1 in saturation giving output voltage 0.2 V = logic 0.

→ For driving transistor Q_1 in to saturation was require more than 2.1 V, at point P.

→ When A and B input are High, transistor Q_1 is driven in to saturation and if the input goes low voltage at point P become 0.7V and transistor Q_1 will try to come out of saturation

→ To drive the transistor from saturation to cut-off it is necessary to discharge the stored charge.

→ The resistor R_B provide a discharge path for the charged stored in the transistor.

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Truth table for 2-input NAND gate.

Performance Parameter

→ Power Dissipation : 60mW.

→ propagation Delay : 30nsec.

→ Noise Margin : 0.7 Volts

→ Fan out : 8

Transistor Transistor Logic (TTL).

→ It is a logic family with bipolar process technology that combines NPN transistor, PN junction diode and resistor in a single structure to get a desired logic function.

→ TTL is name for its dependence on transistor alone.

→ The first version known as standard TTL was developed in 1965 and it is rarely used in today's system.

→ By improving its performance TTL is used.

2-Input TTL NAND gate.

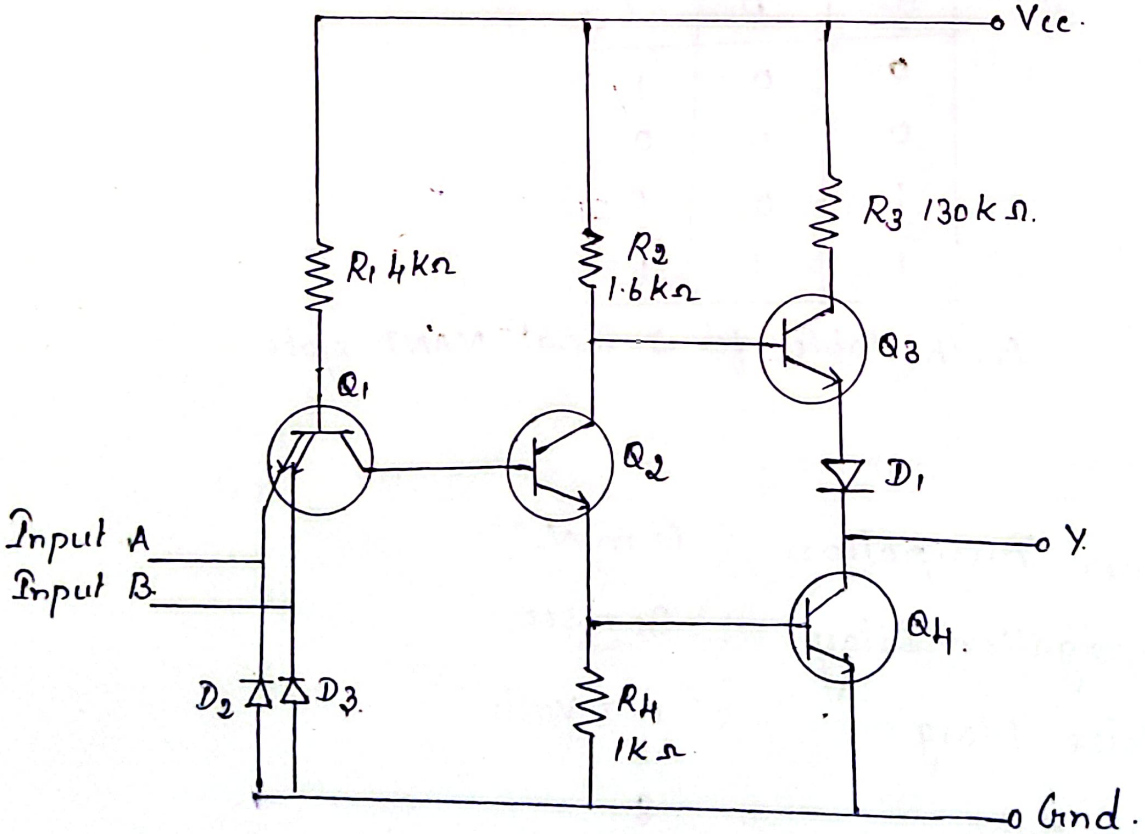


fig: Two input TTL NAND gate.

Circuit Description

Transistor Q_1 is a two emitter NPN transistor which is equivalent to two NPN transistor with base and emitter terminal tied together.

→ Two emitter are the two input of the NAND gate. Diode D_2 and D_3 are used to limit negative input voltage.

diode equivalent circuit of Q_1.

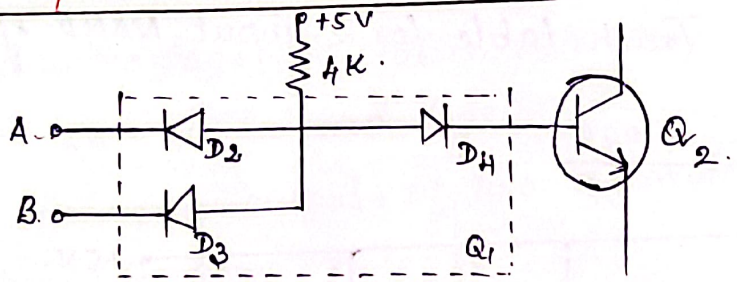


fig :- Diode equivalent for Q_1

→ When both the input is high (1,1) the current flow through the base - collector PN junction diode of transistor Q_1 in to the base of transistor Q_2. Q_2 is turned ON in to saturation with the result that transistor Q_3 is switched off and transistor Q_4 is switched ON.

→ This produce logic low (Logic 0) at the output with V_{OL} being 0.4 V

→ Diode D_1 is used to prevent Q_3 from conduction, during even a small amount of current flow when the output is low.

→ With out D_1 the transistor Q_3 conduct slightly.

→ when both the two input are in Logic Low (0,0) base - emitter region of Q_1 conduct current and Q_2 to cut off.

Q_3 is driven to conduction and Q_4 to cut off.

This produce a Logic high output with $V_{OH}(min) = 2.4\ V$.

A	B.	Y
0	0	1
0	1	1
1	0	1
1	1	0

Fig :- Truth table for 2-input NAND gate.

Totem - Pole output stage

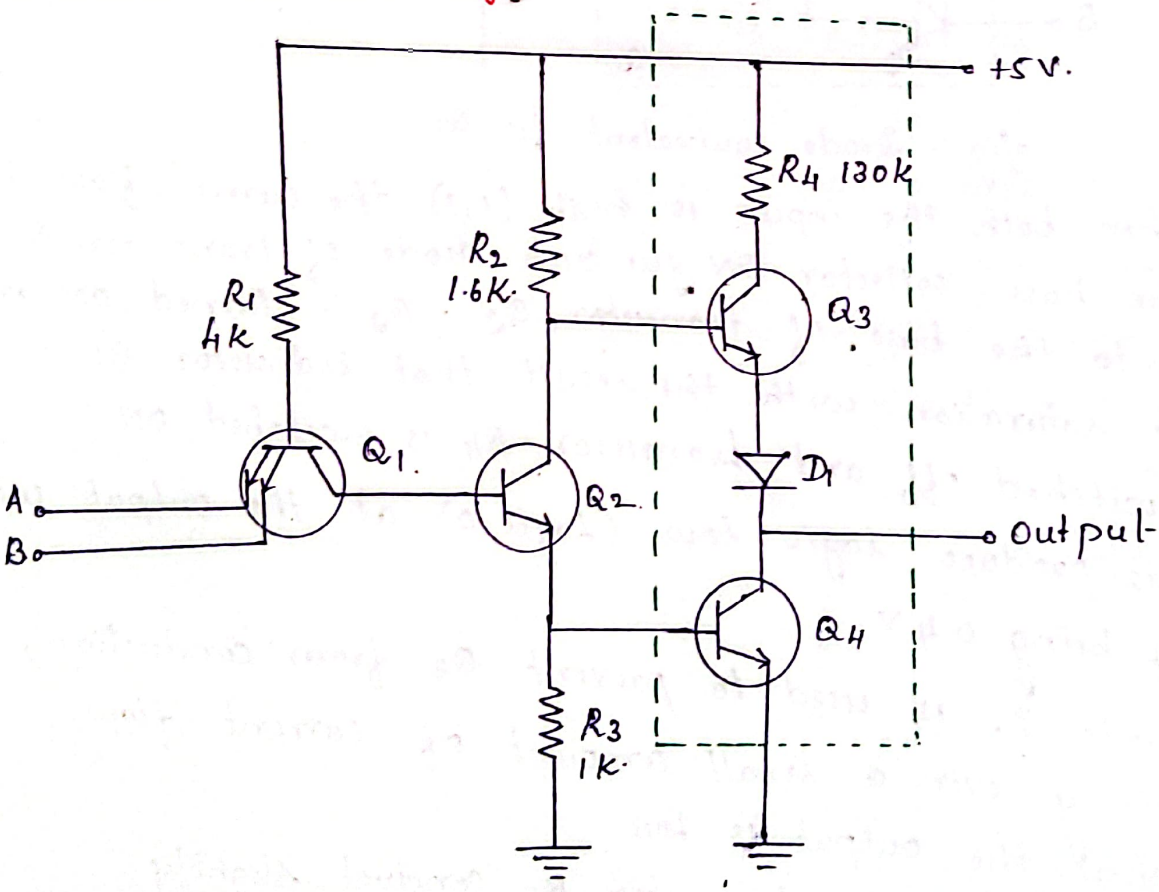


Fig :- Two input NAND gate with totem pole output.

→ Transistor Q_3 and Q_4 is a totem pole output configuration.

→ Totem pole transistor are used because they produce a low output impedance.

→ Q_3 act as a High output and Q_4 is saturated for Low output.

→ when Q_3 is conducting, the output impedance is approximately 70Ω, when Q_4 is saturated the output impedance is only 12 Ω. Either way, the output impedance is Low.

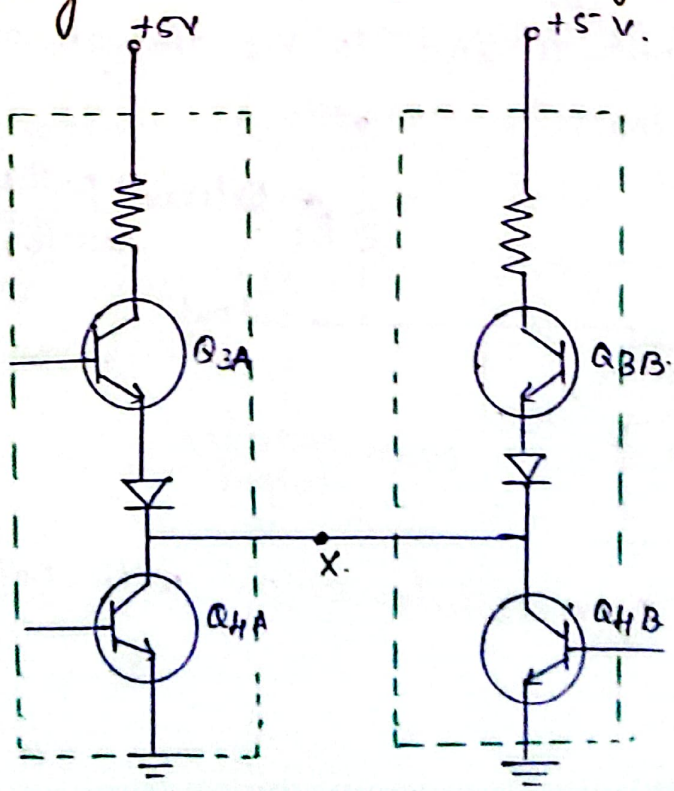
→ This means that the output Voltage can change quickly from one state to another.

→ The disadvantage of totem pole output is that two output cannot be tied together.

→ The totem pole output of two separate gate are connected together, at point X.

* Suppose the output of gate A is high (Q_{3A} ON and Q_{4A} OFF) and the output of gate B is Low (Q_{3B} OFF and Q_{4B} ON). So it draws a high current around 55mA.

* This current might not damage Q_{3A} or Q_{4B} immediately but over a period of time can cause overheating and device failure might occur.



→ More gate cannot tied together.

fig: Totem pole output tied together can produce harmful current.

Open - collector Output

TTL provide another type of open collector output. Fig shows a 2-input NAND gate with an open-collector output eliminate the pull up transistor. Q_3, D_1 and R_4. The output is taken from the open collector terminal of transistor Q_4.

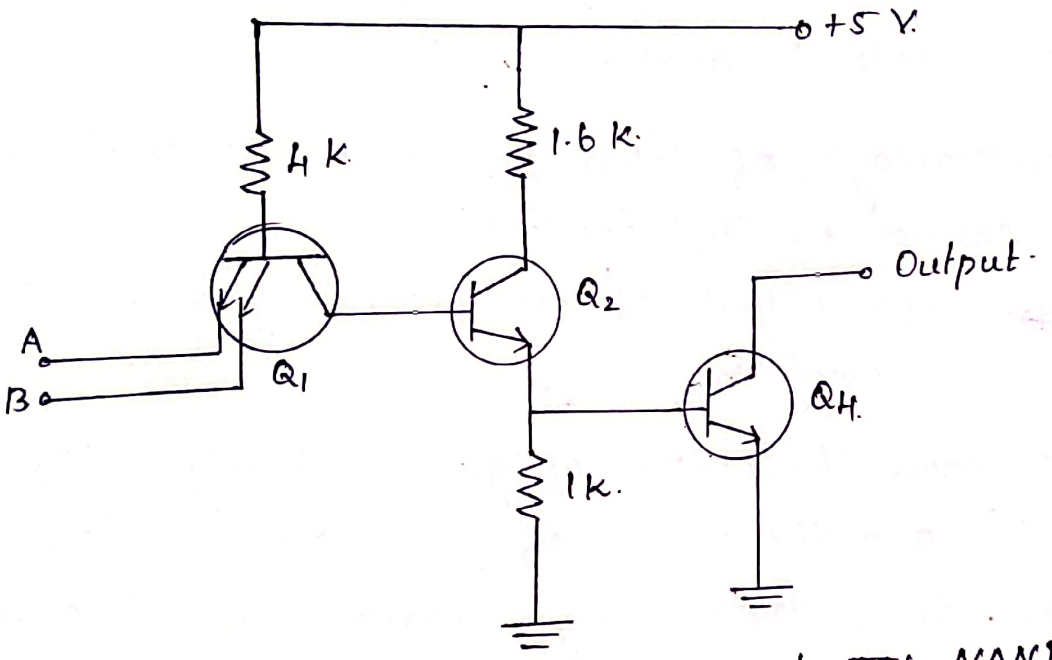


Fig :- Open collector 2 input TTL NAND gate.

→ when Q_4 is ~~[illegible]~~ ON output is Low

→ when Q_4 is off the circuit will not work properly

So an output is tied to Vcc through an external pull up resistor.

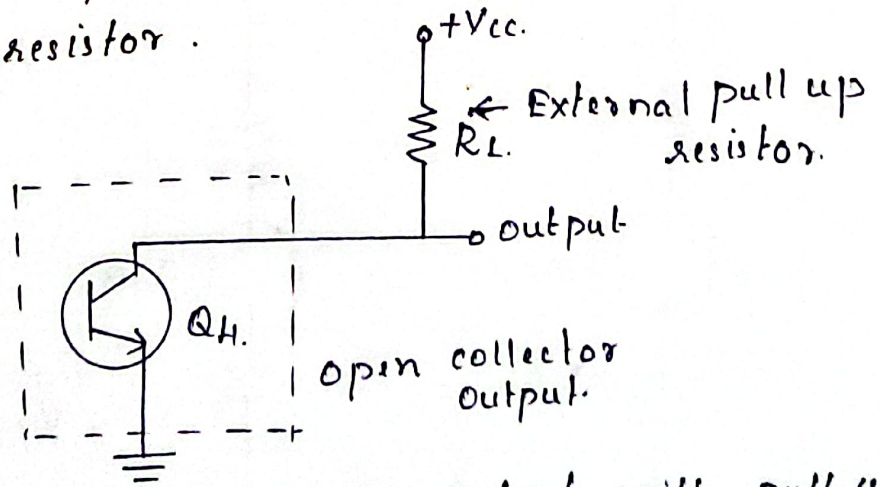
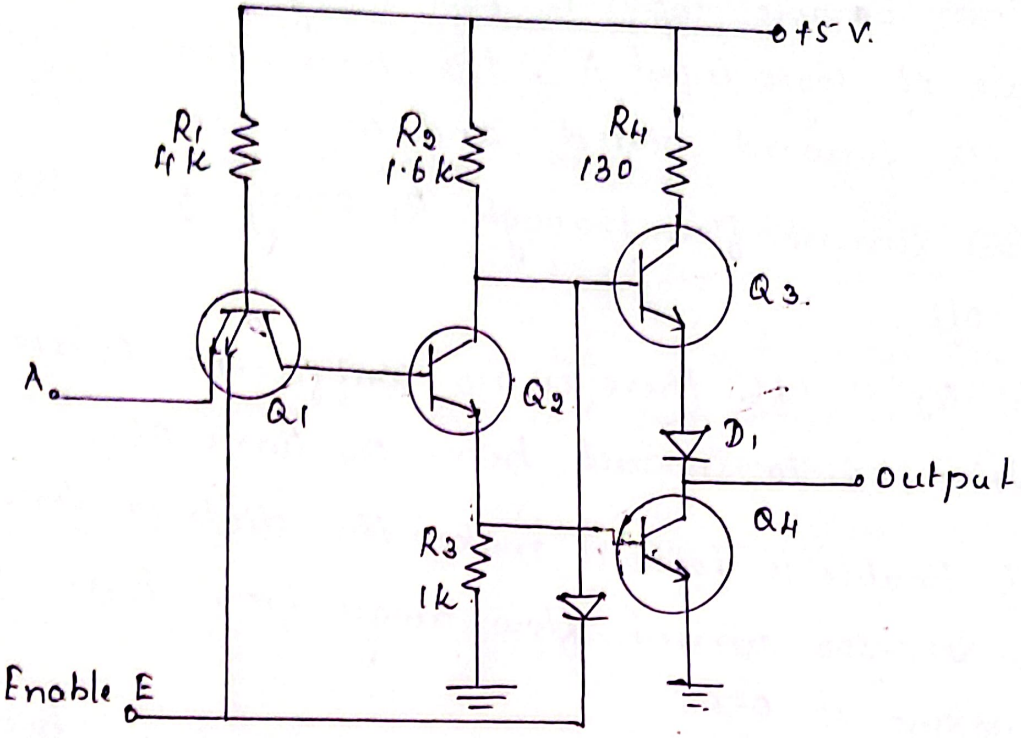


Fig :- Open collector output with pull up resistor.

→ In open collector output two or more gate can be connected together.

Tri state TTL inverter.



→ It is a third type of TTL configuration. It is called tristate TTL because it allow three possible output stages : High, Low and high impedance.

→ The transistor Q_3 is ON when output is High.

→ The transistor Q_4 is ON when output is Low.

→ In high impedance state both transistors, Q_3 and Q_4 are turned off. As the result the open is open or floating, it is neither Low or High.

→ A is a normal input and E is an Enable input.

→ when Enable input is High, the circuit work as a normal inverter.

→ When Enable input is High the state of the transistor Q_1 depends on the logic input A. The diode D_2 is reversed biased, when the enable is High.

→ When Enable input is Low, regardless of the state of logic input A, the base-emitter junction of Q_1 is forward biased and as a result it turns ON.

→ Thus current flow through R_1 away from Q_2 making it off.

→ As Q_2 is off there is no sufficient current to drive Q_4 to conduct hence Q_4 turns off.

→ If Enable is Low it make the diode to forward biases. So current flow away from base of Q_3 making it OFF.

→ In this way when Enable input is Low, both transistor are OFF and output is high impedance state.

Advantage of TTL.

1. High speed operation.
2. Moderate Power disipation
3. Low cost.
4. Available for wide range of function

Disadvantage of TTL.

1. High power disipation than CMOS
2. Lower Noise immunity
3. Less fan-out

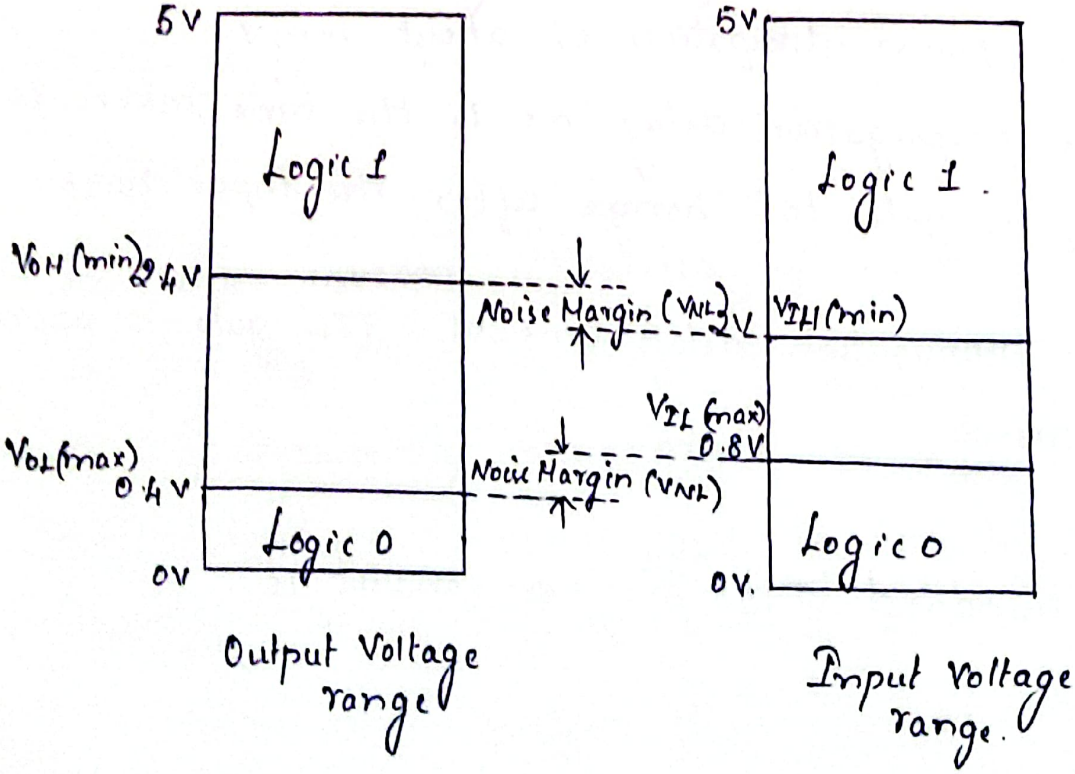
Comparison between Totem-pole and open collector output.

S. No	Totem pole	Open collector.
1.	Output stage consist of pull-up transistor (Q_3), diode resistor and pull-down transistor (Q_4).	Output stage consists of only pull down transistor.
2.	External pull up resistor is not required	External pull up resistor is required for proper operation of gate.
3.	Output of two gate cannot be tied together	Output of two gate can be tied together using wired AND technique.
4.	Operating speed is high.	Operating speed is Low.

Characteristic of TTL family.

Let us see the characteristic of TTL family.

1. Voltage Level and Noise Margin



V_{IL} (max) → Low level input Voltage

V_{IL} (max) = 0.8 V

V_{IH} (min) → High Level input Voltage.

V_{IH} = 2V

V_{OL} (max) → Low Level input Voltage.

V_{OL} – 0.4 V

V_{OH} (min) → High Level input Voltage.

V_{OH} = 2V

Noise Margin,

$$V_{NH} = V_{OH}(min) - V_{IH}(min)$$
$$= 2.4 - 2$$
$$= 0.4 V.$$

$$V_{NL} = V_{IL}(max) - V_{OL}(max)$$
$$= 0.8 - 0.4.$$
$$= 0.4 V$$

Power dissipation and propagation delay.

→ The power dissipation of about 10mV.

→ The propagation delay time is the time taken for the output of a gate to change after the input have changed

→ The propagation delay time of a TTL gate is approximately 10 nanoseconds.

Fan out.

The standard TTL has a fanout 10.

Advantage of TTL

1. High speed operation. propagation delay time is about 10ns.
2. Moderate power dissipation.
3. Low cost.
4. Available in commercial

Disadvantage.

1. Higher power dissipation than CMOS.
2. Lower Noise Margine than CMOS.
3. Less fan out.

Mos family.

MOSFET is grouped in to three categories.

1. PMOS uses only P-channel enhancement MOSFET
2. NMOS uses only N- channel enhancement MOSFET
3. CMOS (complementary Mos) uses both P and N- channel device.

→ NMOS and CMOS are widely used in digital Ics.

N- Mos Logic.

NMOS Inverter

→ In NMOS inverter it consists of two N- channel MOSFET

→ Q_1 and Q_2 MOSFET, where Q_1 is a Load Resistance.

→ Q_1 is permanently connected to V_{DD}, it is always ON, hence load resistance is equal to R_{ON} of the MOSFET.

→ R_{ON} of Q_1 is 100kΩ and R_{ON} of Q_2 is 1kΩ.

→ When high input is applied between gate and source Q_1 switched ON and it make the output low.

→ when input is Low Q_2 is switched off therefore output is high.

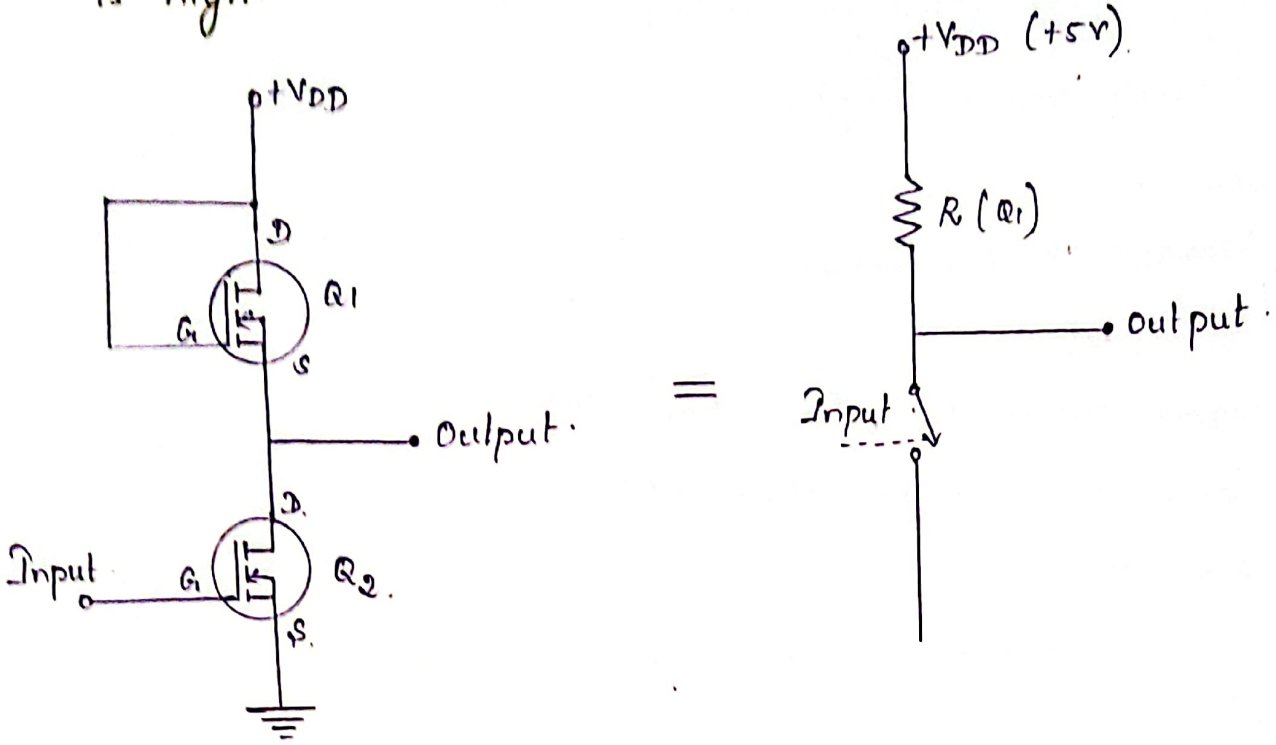


fig:- NMOS Inverter.

Vin	Q_2	$V_0 = \overline{Vin}$
0V (logic 0)	off	+5 (logic 1
5V (logic 1)	ON	0V (logic 0

NMOS NAND gate.

→ It consists of 2-input NMOS NAND gate. Q_1 as a load resistor.

→ Q_2 and Q_3 are the switching MOSFET controlled by the input A and B.

→ If both the input is Low (A and B → logic 0) corresponding MOSFET are off, ie the corresponding switches are open and output is high (logic 1).

→ If both the input are high (A and B → logic 1) the corresponding switches are closed and output is Low.

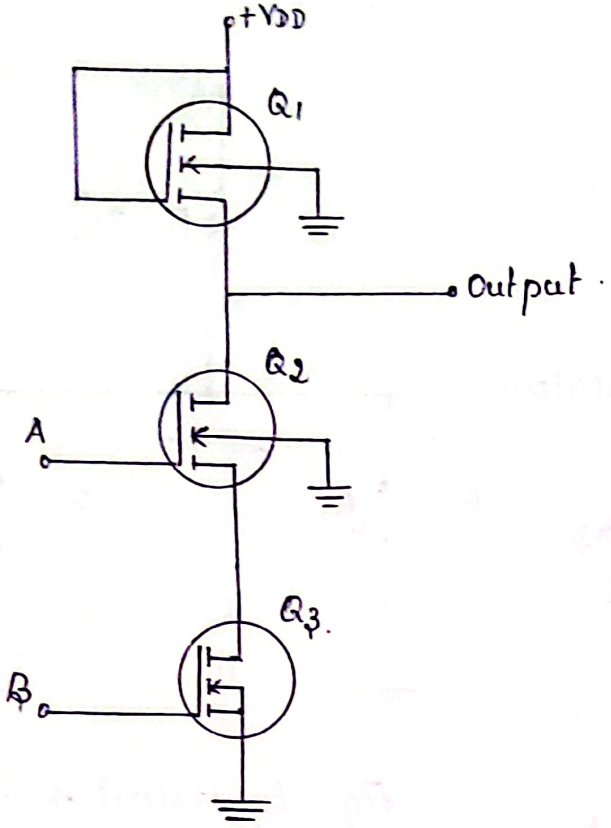


fig :- 2-input NMOS NAND gate.

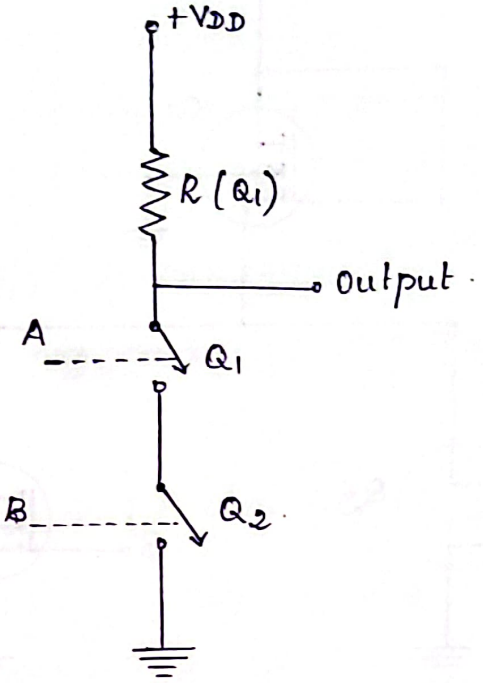


fig :- Equivalent switching circuit.

A	B	Q_2	Q_3	$V_o = \overline{AB}$
0	0	off	off	1
0	1	off	ON	1
1	0	ON	off	1
1	1	ON	ON	0.

0 0 → 0
0 1 → 0
1 0 → 0
1 1 → 1

NMOS NOR gate.

→ It consists of 2-input NMOS NOR gate

→ Q_1 as a load resistor, Q_2 and Q_3 are the switching circuit, connected in parallel.

→ When both the input are high (Logic 1) the corresponding MOSFET are ON (ie) corresponding switches are closed making the output low.

→ When both the input are Low (Logic 0) the corresponding MOSFET are off (ie) corresponding switches are open making the output high

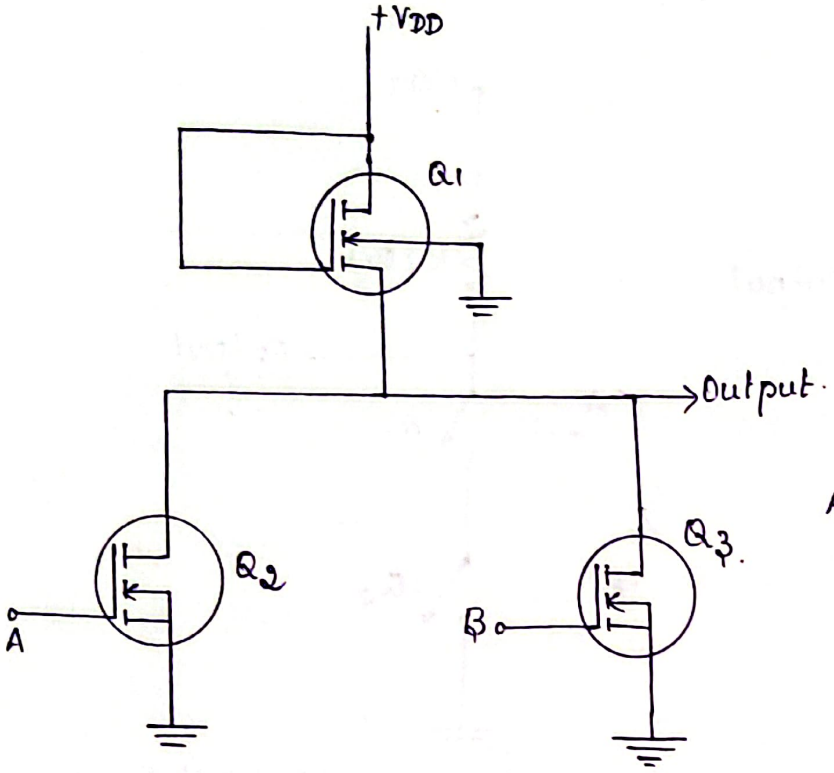


Fig: 2-input NMOS NOR gate.

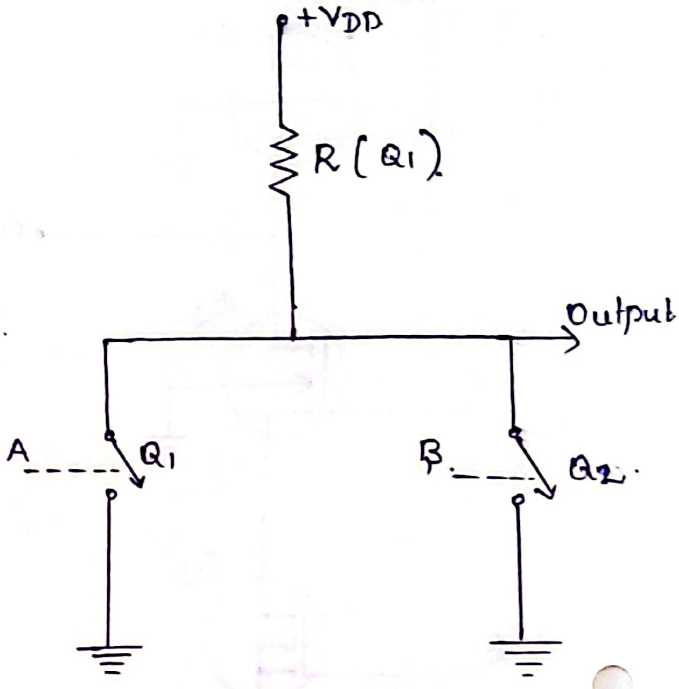


Fig:- Equivalent switching circuit.

A	B	Q_2	Q_3	$V_o = \overline{A+B}$
0	0	off	off	1
0	1	off	ON	0
1	0	ON	off	0
1	1	ON	ON	0

Characteristic of NMOS.

1. Low operating speed with propagation delay 50ns.
2. Noise Margin 1.5 V
3. Fan out typically 30

CMOS Inverter

→ It consists of two MOSFET in series.

→ P-channel device is connected to the source $+V_{DD}$ a positive voltage. and the n-channel device has its source connected to ground.

→ The gate of the two device are connected together as the common input and drain are connected together as the common output.

1. When input is High the gate of Q_1 (P-channel) is at 0V. Thus Q_1 is OFF. On the other hand gate of Q_2 is at $+V_{DD}$ (n-channel) thus Q_2 is ON. This will produce output Low i.e 0V.

2. When input is Low, the gate of Q_1 (P-channel) is at a negative potential relative to its source, Thus Q_1 is ON and V_{Q_2} is off. This will produce high output (Logic 1).

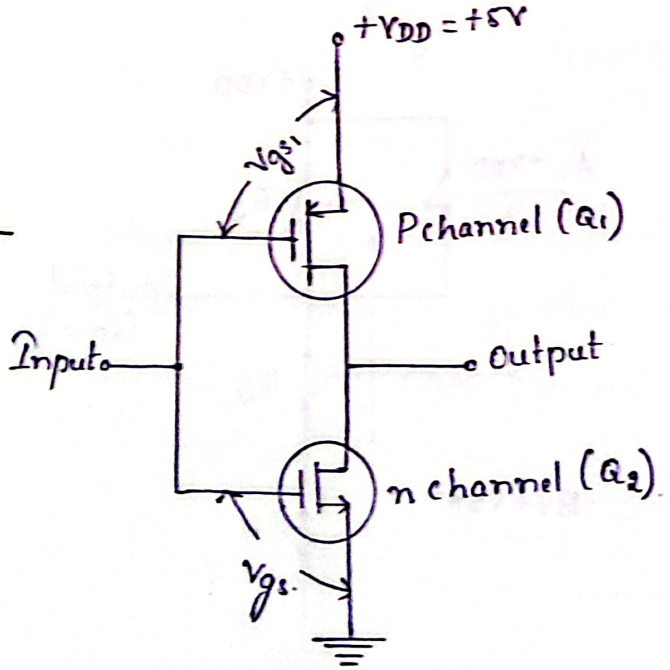


fig :- CMOS inverter circuit

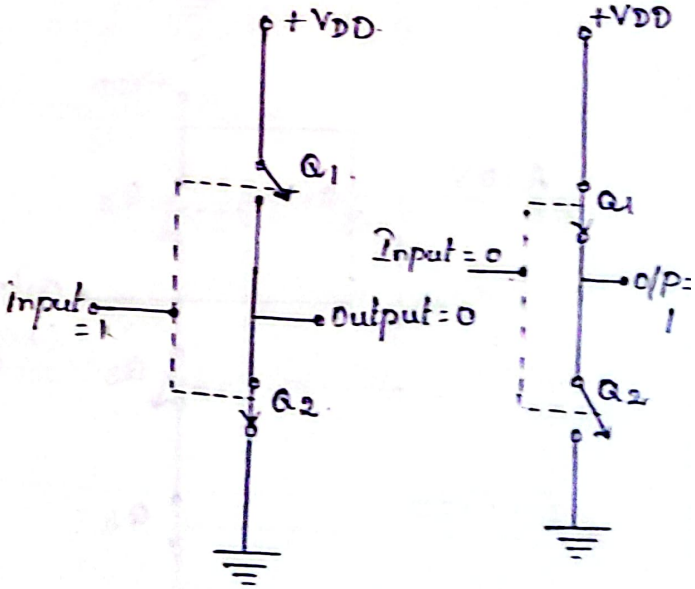


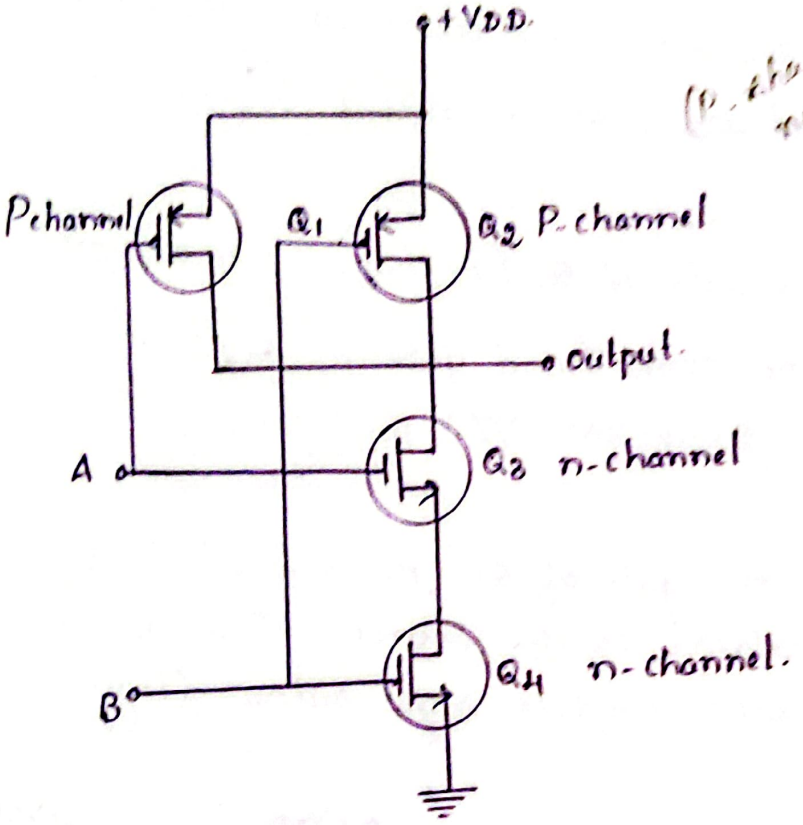
fig :- Operation of CMOS inverter for both input condition

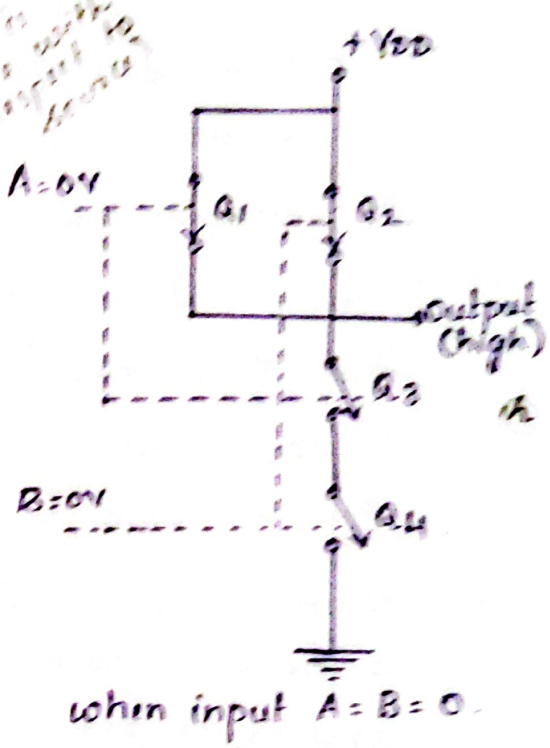
A	Q_1	Q_2	O/P
0	ON	OFF	1
1	OFF	ON	0

fig :- Truth table for inverter

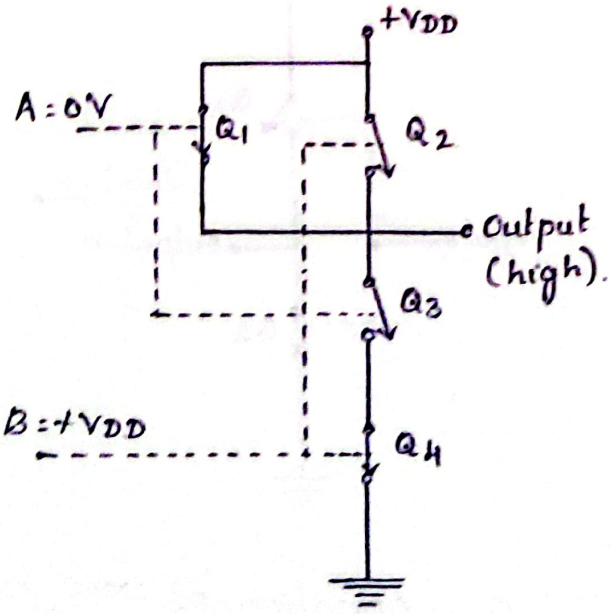
CMOS NAND gate

Fig shows CMOS 2-input NAND gate. It consists of two P-channel MOSFETs Q_1 and Q_2, connected in parallel and two n-channel MOSFET Q_3 and Q_4 connected in series.

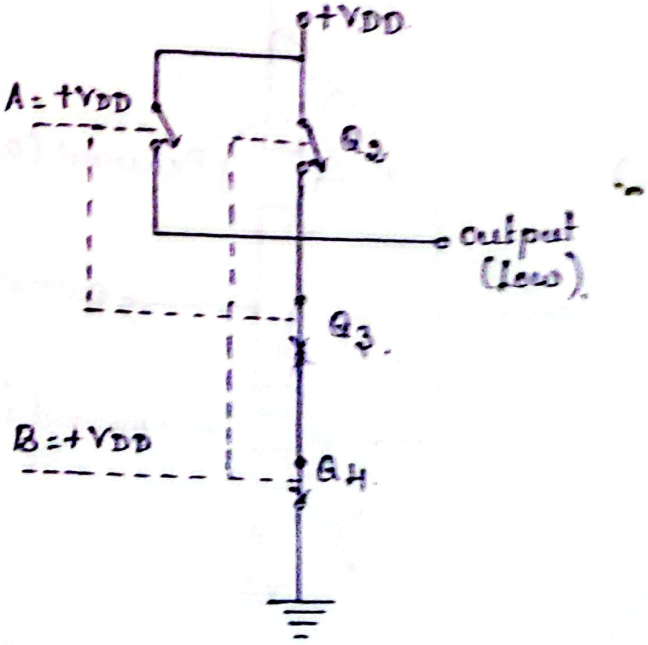




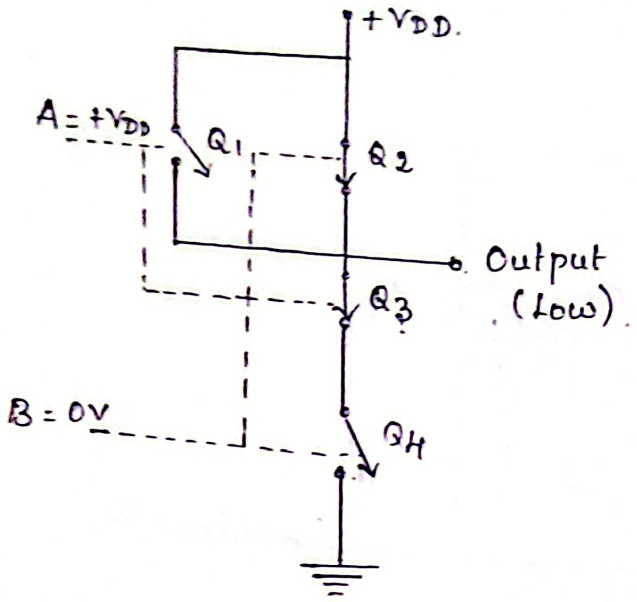
when input A = B = 0.



when input A=0, B=1



when A = 1, B = 1



When A=1 B=0.

→ when both the input are Low, the gate of both P-channel MOSFET are negative with respect to source. Thus Q_1 and Q_2 both are ON. Since the gate to source voltage of Q_3 and Q_4 (n-channel) are 0V so Q_3 and Q_4 are OFF. Therefore output is connected to $+V_{DD}$ (High) through Q_1 and Q_2.

→ when the input A = 0 and B = $+V_{DD}$ (1), Q_1 is ON because of $-V_{DD}$ and Q_4 is ON because of V_{DD}. MOSFET Q_2 and Q_3 are off. The output is connected to V_{DD} through Q_1

→ when the input A=1 and B=0, the MOSFET switch Q_2 and Q_4 is ON. MOSFET Q_1 and Q_4 are off. The output is connected to V_{DD} through Q_2.

→ when the input A=1 and B=1, MOSFET Q_3 and Q_4 is ON. MOSFET Q_1 and Q_2 are off. Thus the output is connected to the ground through Q_3 and Q_4. Hence the output is Low.

A	B	Q_1	Q_2	Q_3	Q_4	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

3. input CMOS NAND gate.

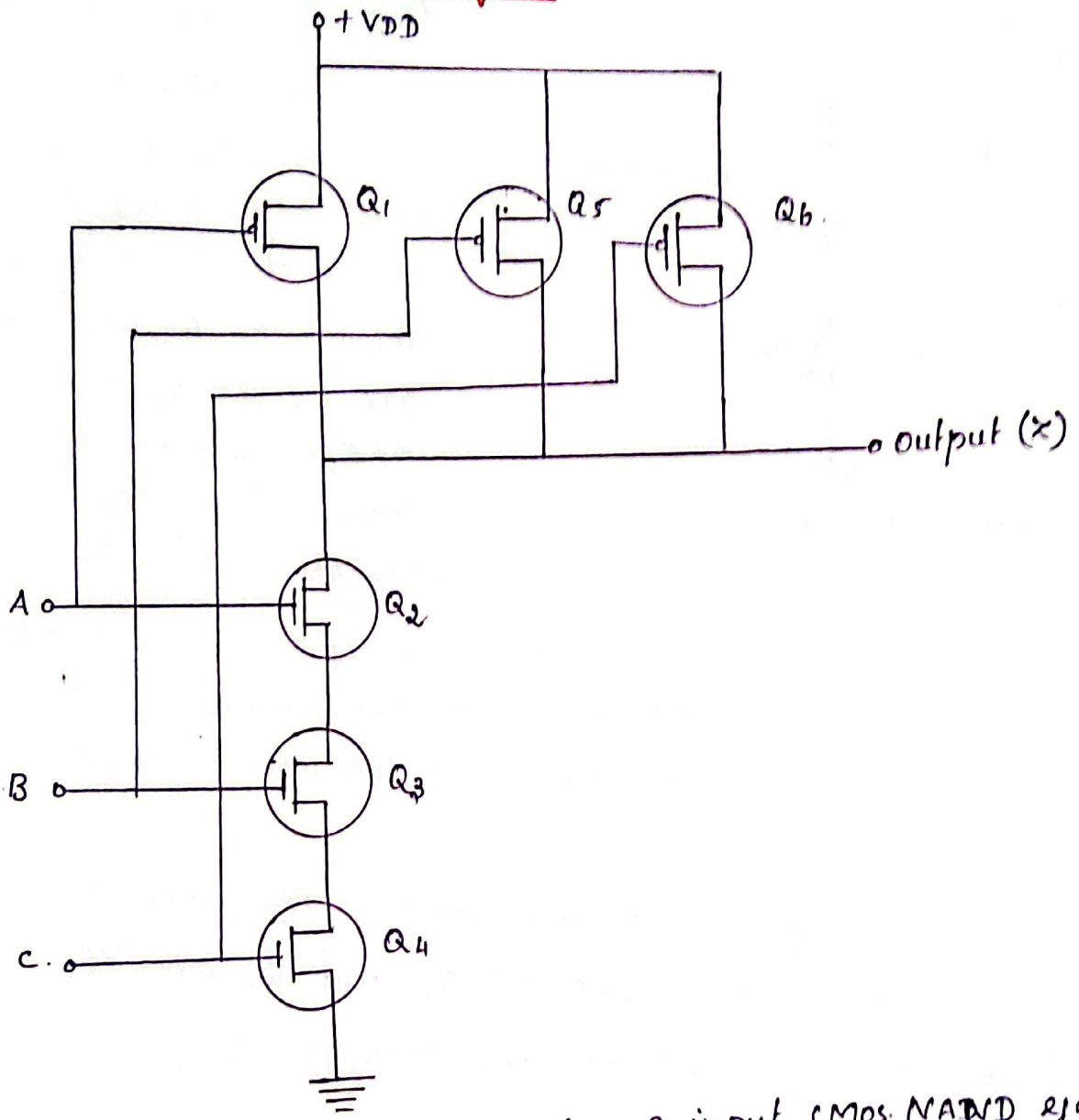


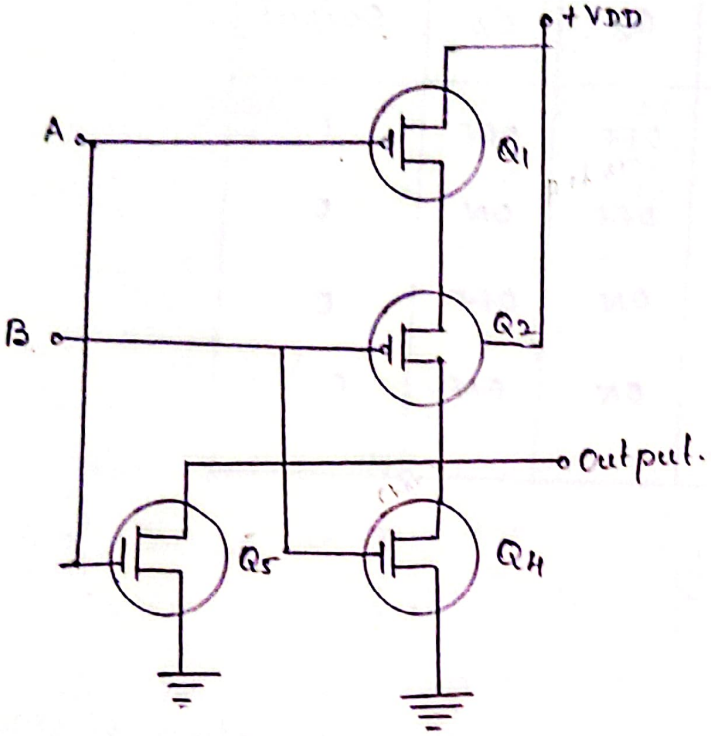
fig :- circuit diagram for 3-input CMOS NAND gate.

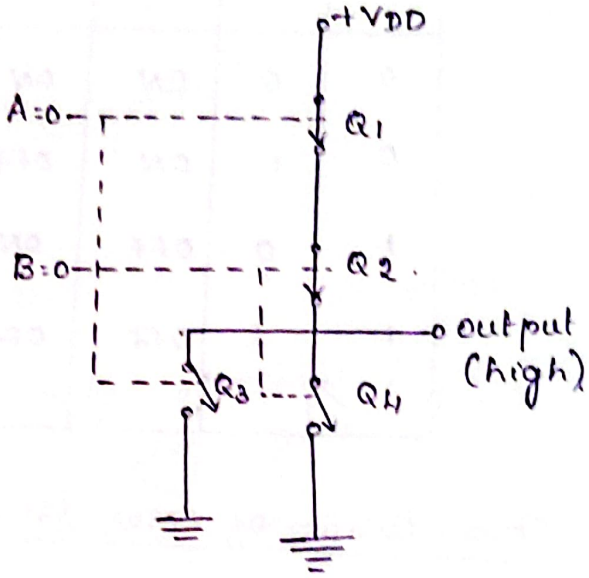
A	B	C	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Z
0	0	0	ON	OFF	OFF	OFF	ON	ON	1
0	0	1	ON	OFF	OFF	ON	ON	OFF	1
0	1	0	ON	OFF	ON	OFF	OFF	ON	1
0	1	1	ON	OFF	ON	ON	OFF	OFF	1
1	0	0	OFF	ON	OFF	OFF	ON	ON	1
1	0	1	OFF	ON	OFF	ON	ON	OFF	1
1	1	0	OFF	ON	ON	OFF	OFF	ON	1
1	1	1	OFF	ON	ON	ON	OFF	OFF	0

Truth table for 3 input CMOS NAND gate.

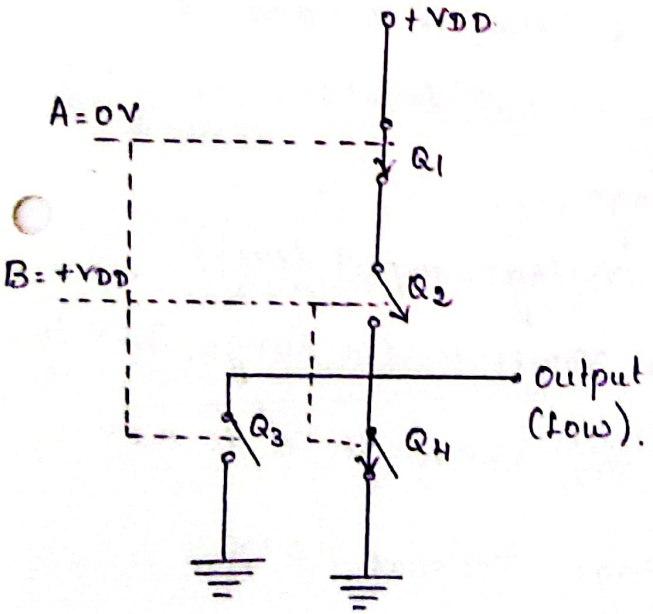
CMOS NOR gate.

fig shows 2-input CMOS NOR gate. Here P-channel MOSFET Q_1 and Q_2 are connected in series and n-channel MOSFET Q_3 and Q_4 are connected in parallel.

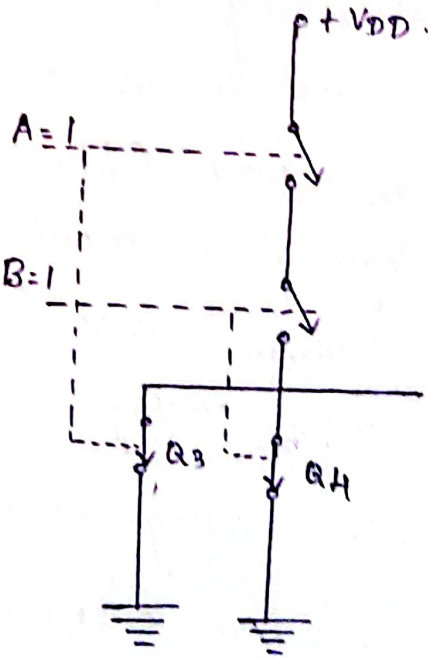




A = B = 0.



A = 0 B = 1.



A = B = 1.

→ when both the input A=0 and B = 0, the P channel MOSFET is ON (Q_1 and Q_2). MOSFET Q_3 and Q_4 are off. Thus output is connected to V_{DD} through Q_1 and Q_2.

→ when input A=0 and B=1 p-channel MOSFET Q_1 is ON and Q_2 is off and MOSFET Q_3 is off and Q_4 is ON Hence output is connected through ground.

A	B	Q_1	Q_2	Q_3	Q_4	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

Characteristic of CMOS family.

1. Voltage Level.

According to manufacturers specification of CMOS De

→ The Low input Voltage is Varied between 0V and 1.5 V.

V_{IL} = 1.5V (Low Voltage input level)

→ High logic level input Voltage,

V_{IHmin} = 3.5 V (High Voltage input level)

High input Voltage of CMOS Varies in the range 3.5V to 5V.

→ Low output Voltage of CMOS V_{OL} Varies in 0 to 0.1V

The Low Level output Voltage $V_{OL\,max}$ = 0.1V.

→ High output Voltage of CMOS V_{OH} Varies in between 4.9V and 5V.

High Level output Voltage $V_{OH\,min}$ = 4.9V.

2. Noise Margin

The Low Level Noise Margin,

$V_{NL} = V_{IL(max)} - V_{OL(max)}$

$= 1.5 - 0.1$

$= 1.4V.$

High Level Noise Margin

$V_{NH} = V_{OH(min)} - V_{IH(min)}$

$= 4.9 - 3.5$

$= 1.4V.$

3. Fan-out and Fan-In.

Fan-out is the maximum number of CMOS Logic gates that can be driven by a single CMOS Logic gate.

→ fan-out of CMOS varies in the range of 20 to 50.

Fan-in means maximum number of input for a CMOS gate.

→ The fan-in of CMOS lies between 2 to 8.

4. propagation delay.

The propagation delay of CMOS Ic generally varies in between about 20ns to 100ns.

5. Power dissipation

The power dissipation of a CMOS gate is about 10nw.

Advantage and disadvantage of CMOS family.

1. Advantage.

→ consume Less power.

→ can be operated at high voltage.

→ fan-out is more.

→ Better noise Margin.

disadvantage.

1. Switching speed low.
2. greater propagation delay.

ECL family.

The TTL family uses transistor operating in the saturation mode. As a result their switching speed is limited by the storage delay time associated with a transistor.

→ Hence another logic family have been developed thin prevent transistor saturation, there by increasing over all switching speed, by using a different circuit structure called current Mode logic (CML). This logic family is also called Emitter-coupled logic (ECL).

ECL OR/NOR gate.

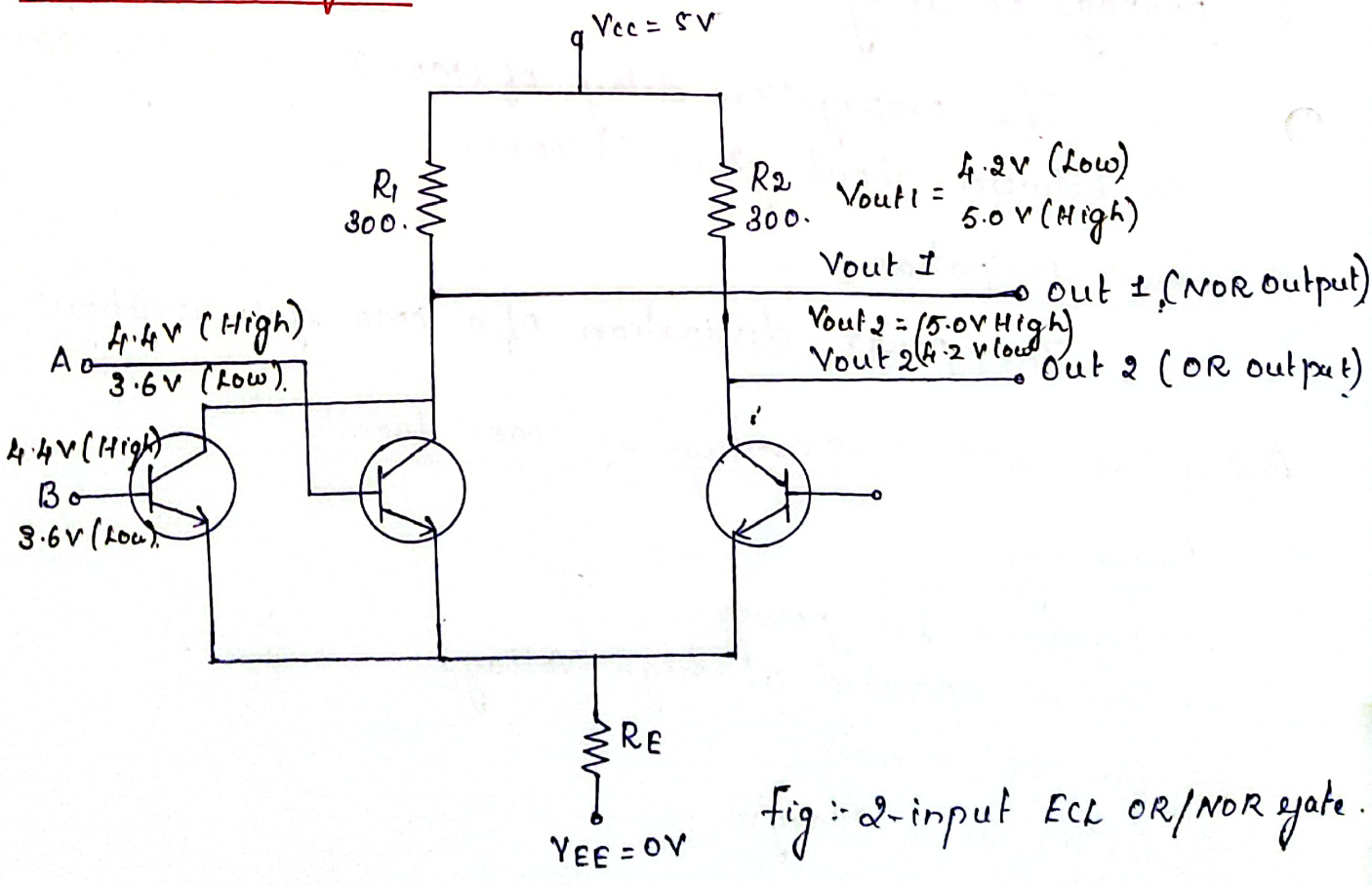


Fig :- 2-input ECL OR/NOR gate.

Fig shows - 2 input ECL OR/NOR gate, It has three transistor Q_1, Q_2 and Q_3.

→ Input is connected to Q_1 and Q_2 the input are A and B.

A	B	NOR	OR
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

→ The Emitter of all the transistor are coupled together.

→ At Q_3 constant voltage supply is given at the base.

→ If the input of A and B are low (ie) logic 0 the base emitter of Q_1 and Q_2 are less when comparing to Q_3. Hence Q_1 and Q_2 is off therefore output is high at Vout 1 and low at Vout 2 thus Q_3 is ON

→ If both the input A and B are high (ie) logic 1 the transistor Q_1 and Q_2 are ON, the voltage at Q_1 and Q_2 is high comparing to the voltage at Q_3 Hence Vout 1 is low and Vout 2 is 1.

→ we can observe that the input and output Low and High voltage level for basic ECL family are not same it has 0.6 v difference. This problem cannot be solved by connecting diode in series with output is lower its voltage by 0.6 volt because if we does this, it result poor fan out.

Comparision between Various digital Logic families.

Parameter	RTL	DTL	TTL	ECL	CMOS.
Component used	Resistance and transistor	Resistor, diode and transistor	Resistor, diode and transistor	Resistor and transistor	N-channel an P-channel MOSFET
circuit	Simple	Moderate	complex	complex	Moderate.
Noise Margin [Noise immunity]	Nominal	Good	Very good	good	Very good.
Fan out	Low (4)	Medium (8)	More (10)	High. (25)	(50)
Power dissipation in mw per gate	12	8-12	10	40-55	0.1

unit : III

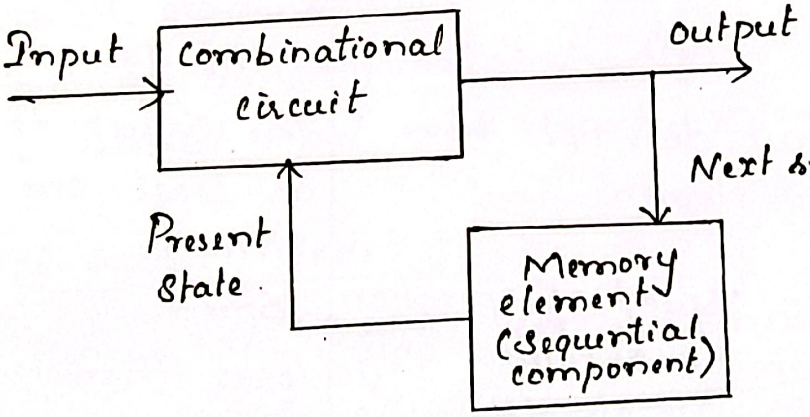
Synchronous Sequential circuit.

Introduction

Digital Logic circuit are classified in to two types combinational and sequential

→ combinational Logic circuit is one whose output depend only on its current inputs.

→ A Sequential logic circuit is one whose output depend not only on its current input, but also on the past sequence of input.



Shows the block diagram of Sequential circuit or finite state Machine (FSM).

→ The present state of the Sequential circuit is always stored in the memory element. Therefore memory element must be capable to store the information and to specify the state of the machine.

→ The next state of the finite state machine can be determined by the present state of the machine and by input.

→ The memory element used in the sequential circuit are flip flop which are capable of storing one bit binary information. Generally flip flop circuit has two output, one for normal value and other for the complemented value of the bit stored in it.

Comparision between combinational circuit and sequential circuit.

S: No.	Combinational circuit	Sequential circuit.
1.	A logic circuit whose output depends only on the present input	A circuit whose output depends not only on the present input, but also on the past history of the data.
2.	The circuit does not have memory element	The circuit that have at least one memory element.
3.	Faster in speed of operation delay can be obtained only due to propagation delay of gates.	Speed of operation is lower than the combinational circuit because due to presence of memory element.
4.	Combinational circuit are easy to design	Sequential circuit are harder to design.
5.	circuit is more expensive Example:- Adder, Subtractor	Cheaper circuit. Example: Serial adder circuit.

Analysis and design of synchronous sequential circuit.

The combinational logic circuit are a part of digital system and they have many application such as decoder, encoder, adder, subtracter, multiplexer, demultiplexer etc. But when the circuit output not only depends on the present state but also the previous state is known as sequential logic circuit.

Any sequential circuit consists of a combinational logic circuit and memory element. The output of combinational logic circuit is stored in the memory element. Memory element output feed back in to combinational logic circuit and used as a input variable. The output of the combinational logic circuit depends upon the external input and input from memory element.

Sequential circuit Model.

Shows the model for a general sequential circuit which consists of combinational logic circuit and memory element.

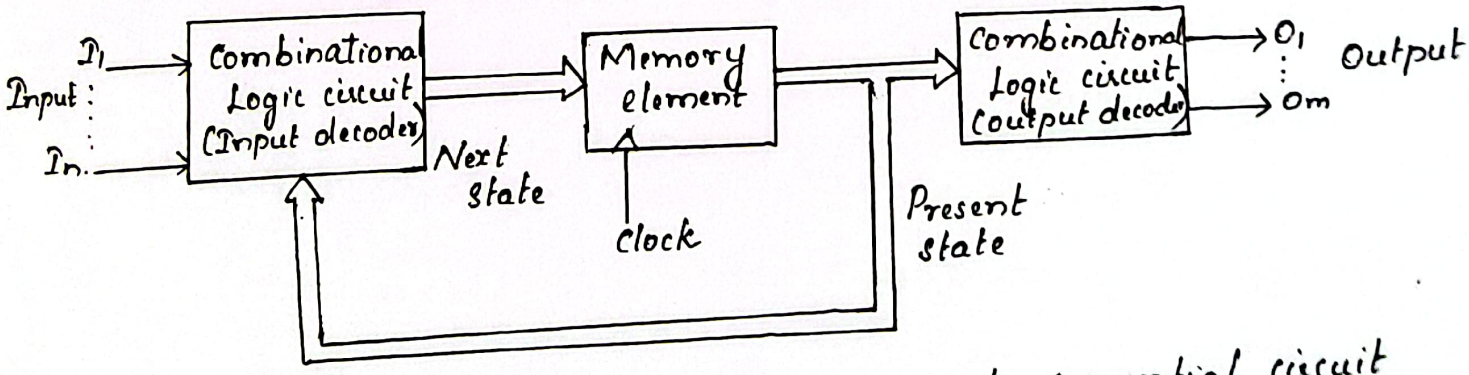


Fig:- Block diagram of a general sequential circuit

This Sequential circuit is driven by a clock signal and the output can be changed on either positive or negative edge of the clock pulse only.

The present state of the sequential circuit is always stored in the memory element.

The next state of finite state machine can be determined by the present state of the machine and by the input

Classification of Sequntial circuit

The Sequential circuit are classified depending upon the presence or absence of a clock signal such as,

→ Synchronous Sequential circuit.
→ Asynchronous sequential circuit.

* when a Sequntial circuit is driven by a clock signal. it is called synchronous sequential circuit.
* If a circuit perform operation with out a clock signal it is known as asynchronous sequential circuit.

Again the circuit can be classified depending on the effect of the present input on the present output. Such as Moore Machine and Melay Machine.

Moore Machine

In Moore Machine, the output depend directly only on the present state.

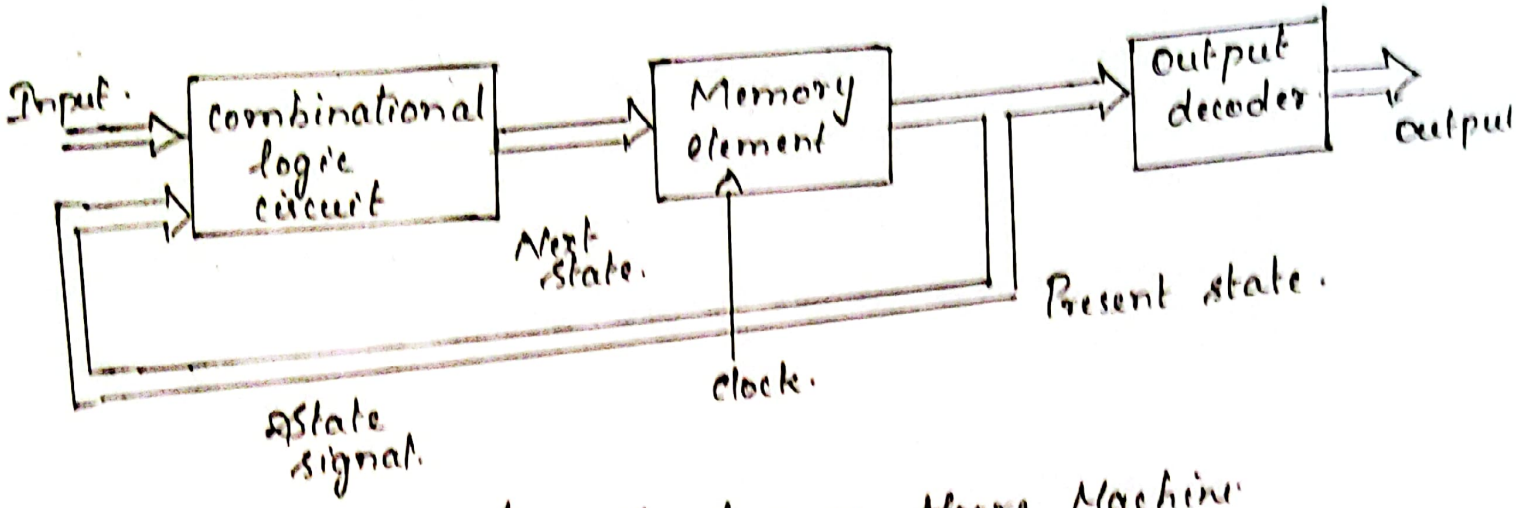


fig :- Synchronous Moore Machine

Melay Machine.

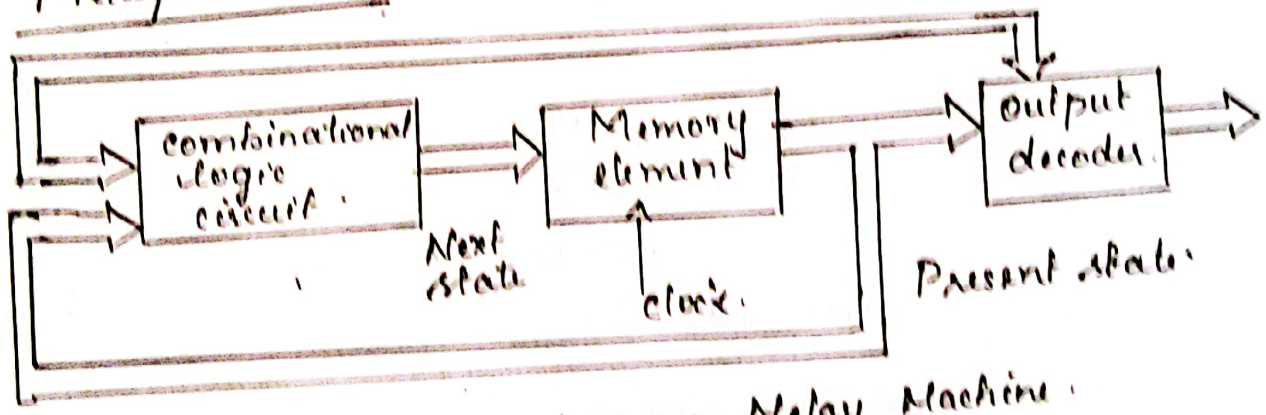


fig :- Synchronous Melay Machine.

The output of the sequential circuit depend on both the present state and input.

Moore Model	Melay Model
1. Its output is a function of present state	It output is a function of Present state as well as present input
2. Input change does not affect the output	Input change may affect the output of the circuit
3. Moore model require more number of state for implementing same function	It requires less number of state for implementing same function

Latch

A latch is an electronic sequential logic circuit used to store information in an asynchronous arrangement.

Latch is an asynchronous device and it has no clock input.

SR Latch using NOR gates

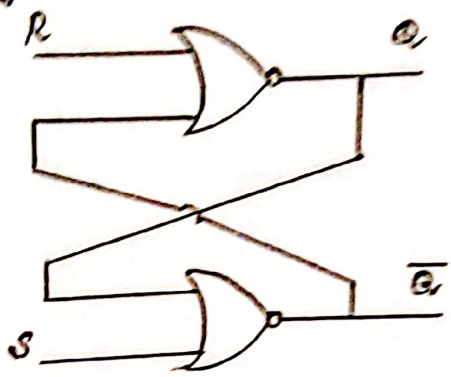


fig :- Cross coupled NOR gates

The fig shows the two cross coupled NOR gates. The NOR gate are connected in such a way the output of one feeds back to the input of another.

→ S and R are two input of S-R latch. S stand for set, it means that when S is 1, it stores 1. Similarly, R stands for Reset and if R=1, flip flop reset and its output will be 0.

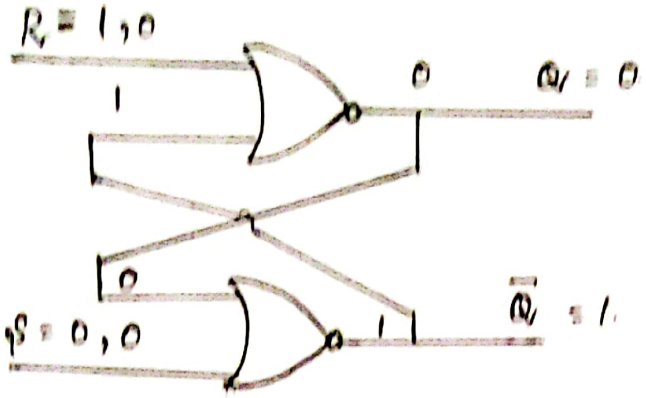
Truth table for NOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Case(i) when S=0, R=1, Q=0 and $\bar{Q}$=1

→ If we apply S=0 and R=1, the flip flop remain in reset mode. ii) Q=0 and $\bar{Q}$=1

→ It is to be concluded the flip flop remembers on to the previous output state when both the input go low [ii, S=0, R=0]

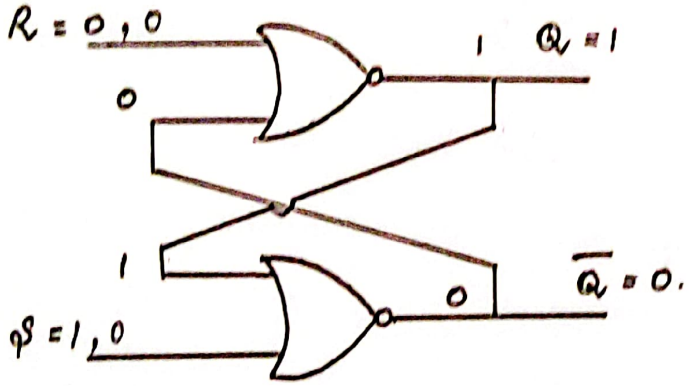


(a)

Case (ii) when S = 1 and R = 0 , Q = and $\bar{Q}$

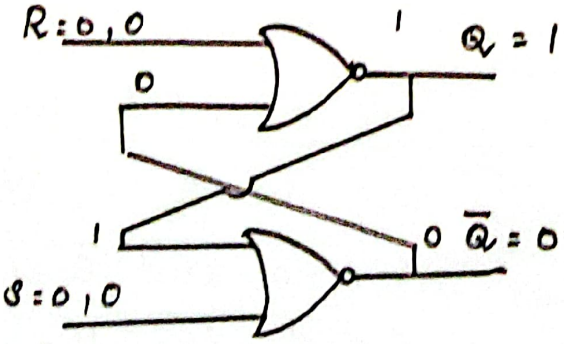
→ If S = 1 and R = 0 the flip flop would remain set and output $\bar{Q}$ = 0 and Q = 1.

→ And then by applying S = 0 , R = 0 the flip flop remain in set mode.



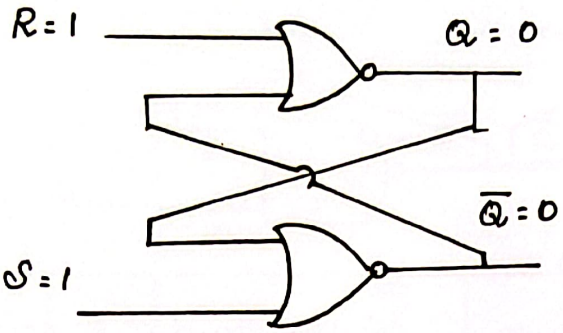
Case (iii) when S = 0 , R = 0

In this condition there is no effect on output consequently the state Q and $\bar{Q}$ will not be changed This is memory operation of SR latch.



Case (iv) $S=1$, $R=1$

If $S=1$, $R=1$, latch is set and reset at the same time. The output will be $Q=0$ and $\overline{Q}=0$. But practically both output zero have no use. Therefore this condition is invalid as the Q and $\overline{Q}$ output must be complement of each other.



The truth table of S-R latch using NOR gate.

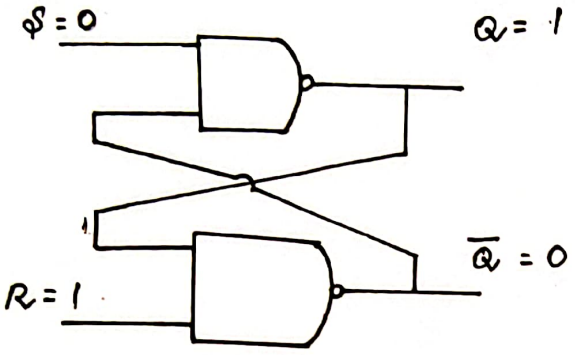
Input		Output		Comment.
S	R	Q	$\overline{Q}$	
0	0	Latch	Latch	Memory.
0	1	0	1	Reset.
1	0	1	0	Set
1	1	0	0	Not used (invalid)

SR Latch using NAND gate

The SR latch constructed from two NAND gate. The fig shows that two NAND gate are cross coupled. Output of one NAND gate is connected with the one input of the other NAND gate. The output of gates are Q and $\overline{Q}$. The output are complement of each other.

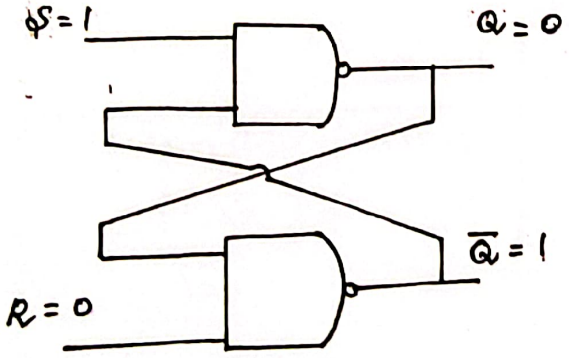
Case(i) when $S = 0$ and $R = 1$.

when $S = 0$ and $R = 1$ the output of latch will $Q = 1$ and $\bar{Q} = 0$, The flipflop is set. This is called set operation of SR latch.



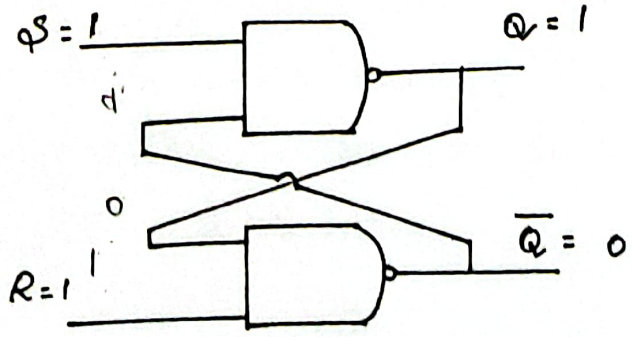
case(ii) when $S = 1$ and $R = 0$

when $S = 1$ and $R = 0$, the output will be $Q = 0$ and $\bar{Q} = 1$. This is known as reset operation of S-R latch.



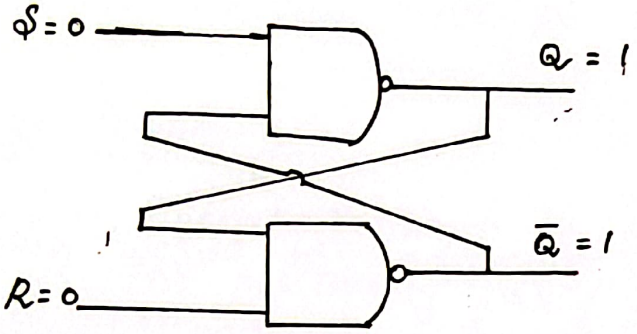
case(iii) when $S = 1$ and $R = 1$

when $S = 1$ and $R = 1$ there is no effect on output. Consequently, the state of Q and $\bar{Q}$ will not be change. This is the memory state of the latch.



Case iv : when $S=0$ and $R=0$.

when $S=0$ and $R=0$ the output of the latch will be $Q=1$ and $\bar{Q}=1$. But practically both output high will not be allowed, therefore this state is called invalid state of latch.



Input		output		comment
S	R	Q	$\bar{Q}$	
0	0	1	1	Not used.
0	1	0	1	Reset
1	0	1	0	set
1	1	latch	latch	Memory

Flip-Flops.

A flip flop is a device which changes its state at the time when a change is taking place in the clock signal. The flip flop is triggered by either positive edge or negative edge of the clock signal.

Clock

A clock signal is a particular type of signal that oscillate between a high and low state It is produced by a clock generator.

→ The most common clock signal is in the form of a square wave with a 50% duty cycle.

→ The time required to complete one cycle is called clock period or clock cycle.

Type of Triggering

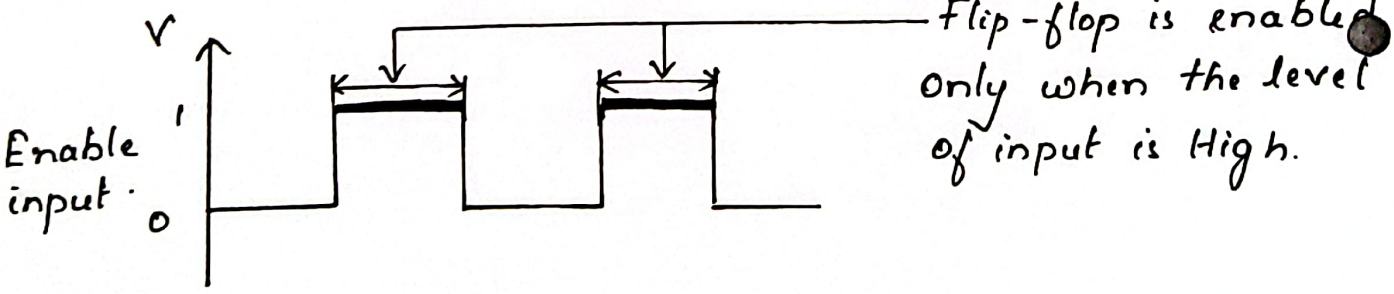
1. Level Triggering

In Level triggering the output state is allowed to change according to input(s). There are two types of level triggering

1. Positive level triggering.
2. Negative level triggering.

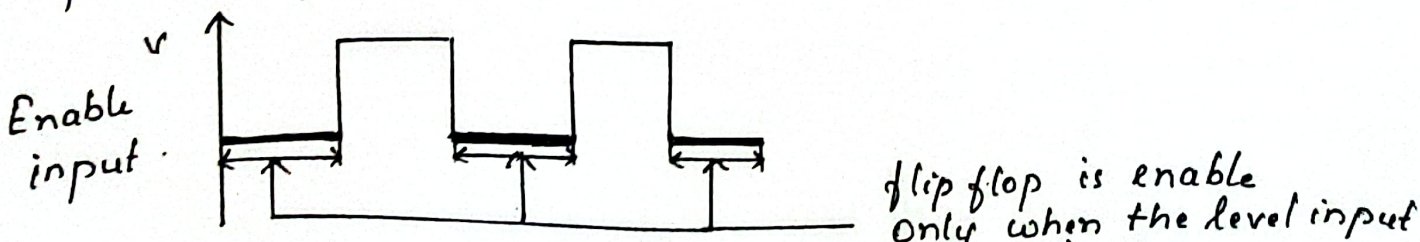
* Positive level triggering

The output of flip flop responds to the input changes only when its enable input is 1 (High).



* Negative level triggering

The output of the flip flop responds to the input changes only when its enable input is 0 (Low)

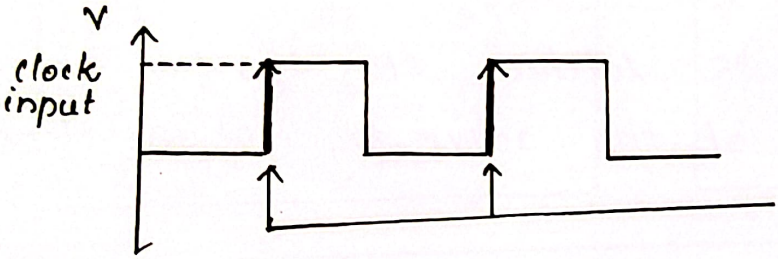


Edge triggering

In Edge triggering, the output responds to the change in the input only at the positive or negative edge of the clock pulse at the clock input. Edge triggered flip flop are designed to avoid race condition. There are two types of edge triggering.

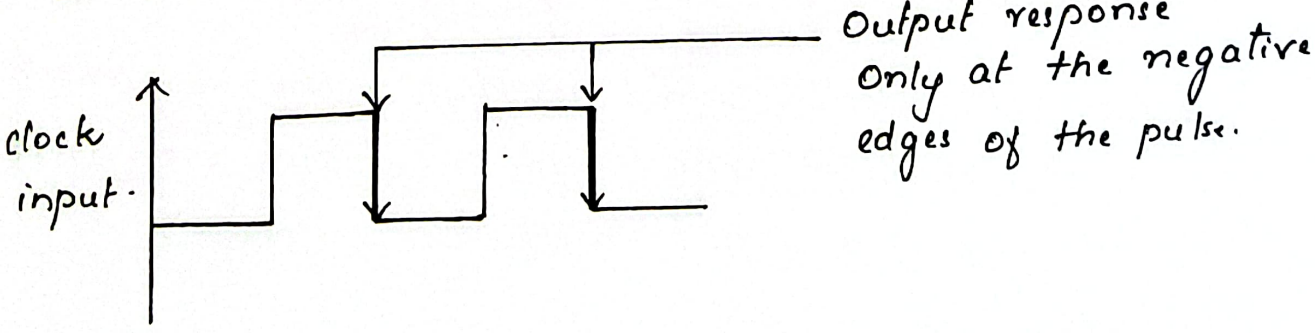
* Positive edge triggering

The output responds to the changes in the input only at the positive edge of the clock pulse at the clock input.



* Negative edge triggering

The output responds to the change in the input only at the negative edge of the clock pulse at the clock input.



Characteristic table and characteristic Equation of clocked flip flops.

There are basically four main types of clocked flip flops they are.

(i) SR flip flop.
(ii) D flip flop.
(iii) JK flip flop.
(iv) T flip flop.

(i) SR flip flop

Like SR latches, SR flip flop are useful in control application where we want to able to set or reset the data bit.

But unlike SR latches, SR flip flop changes their content only at the active edge of the clock signal.

→ In case of clocked Rs flip flop, the output changes state as per the input only on the occurence of a clock pulse.

→ Two NAND gate at the input have been used to coupled the R and S input to the flip flop input under the control of clock signal.

→ when the clock signal is High, the two NAND gate is enabled and the SR input are passed on to flip flop input with their status complemented.

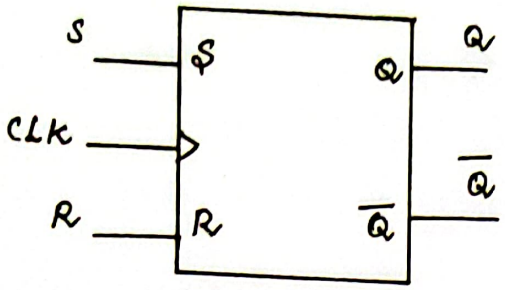


fig :- Logic symbol

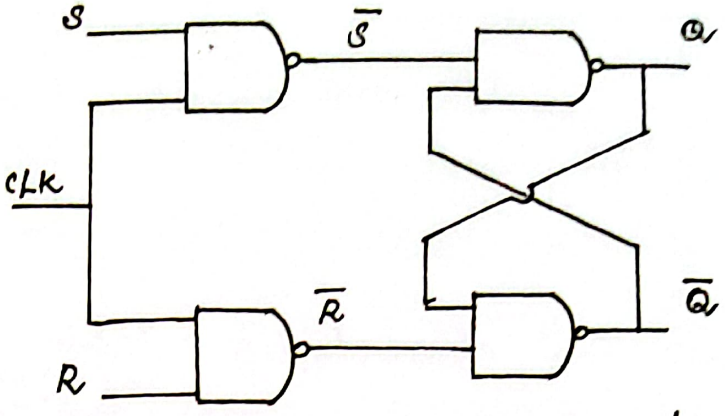


fig :- flip flop using NAND gate

Note :-

SR Latch truth table.		
$\overline{S}$	$\overline{R}$	Q $\overline{Q}$
0	0	Not used
0	1	1 0
1	0	0 1
1	1	Memory.

CLK	S	R	Q $\overline{Q}$
0	x	x	Memory.
1	0	0	Memory
1	0	1	0 1
1	1	0	1 0
1	1	1	Not used

Truth table for SR flip flop.

CLK	S	R	Qn+1
0	x	x	Qn.
1	0	0	Qn.
1	0	1	0
1	1	0	1
1	1	1	Invalid.

Characteristic table of SR flip flop :-

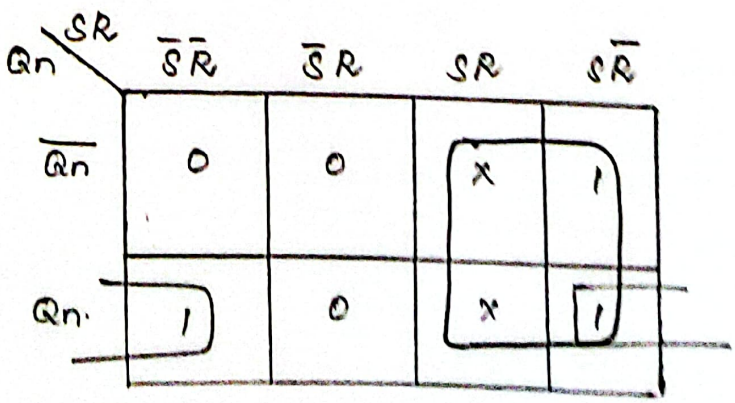
The characteristic table of SR flip flop is given below. This define the next state as a function of input S & R and present state Qn.

→ The characteristic table describe the operation of flip flop with the application of input and present state and specifies the next state.

clk	present state	data input		Next state	Comment.
	Qn	S	R	Qn+1.	
1	0	0	0	0	No change
1	0	0	1	0	Reset
1	0	1	0	1	Set
1	0	1	1	x	Indeterminate
1	1	0	0	1	No change
1	1	0	1	0	Reset
1	1	1	0	1	Set
1	1	1	1	x	indeterminate

Characteristic Equation

The characteristic Equation is the functional Boolean equation that is derived from the characteristic table.



$$Q_{n+1} = S + \bar{R}Q_n$$

Excitation table

The table that describe the require input of a flip flop with the application of present state and next state such form of table is known as excitation table

Present state	Next state	Input	
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

D - flip flop

A D flip flop is a single input device It is basically an SR flip flop where S is replaced with D and R is replaced with $\bar{D}$ (inverted D)

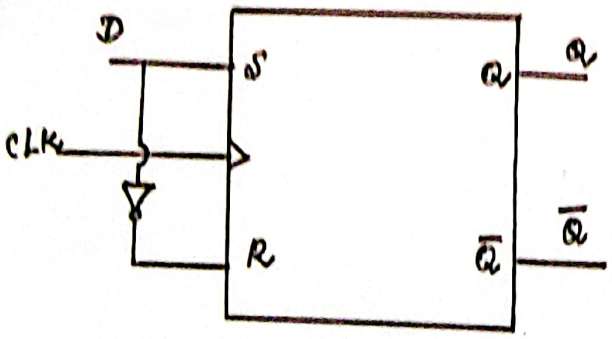


fig :- Logic symbol

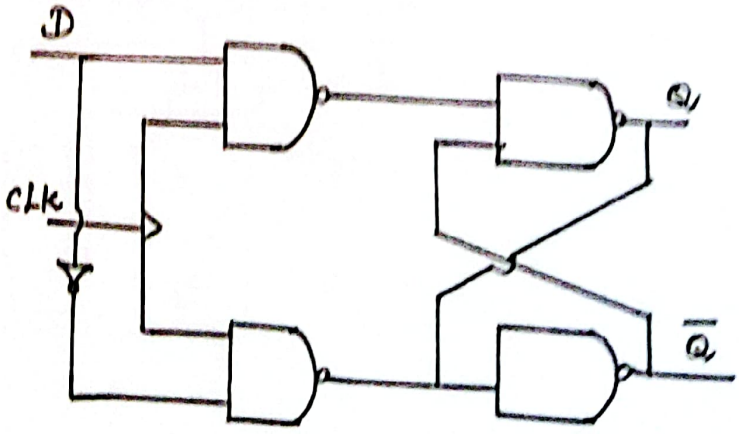


Fig :- D flip flop.

Note:

Truth table for SR flip flop

clk	S	R	Qn+1
0	X	X	Qn
1	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	invalid

Truth table for D flip flop

clk	D	Qn+1
0	X	Qn
1	0	0
1	1	1

characteristic table for D flip flop

Qn	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

The characteristic equation is derived from K-Map.

Qn \ D	$\overline{D}$	D
$\overline{Qn}$	0	1
Qn	0	1

$Qn+1 = D$

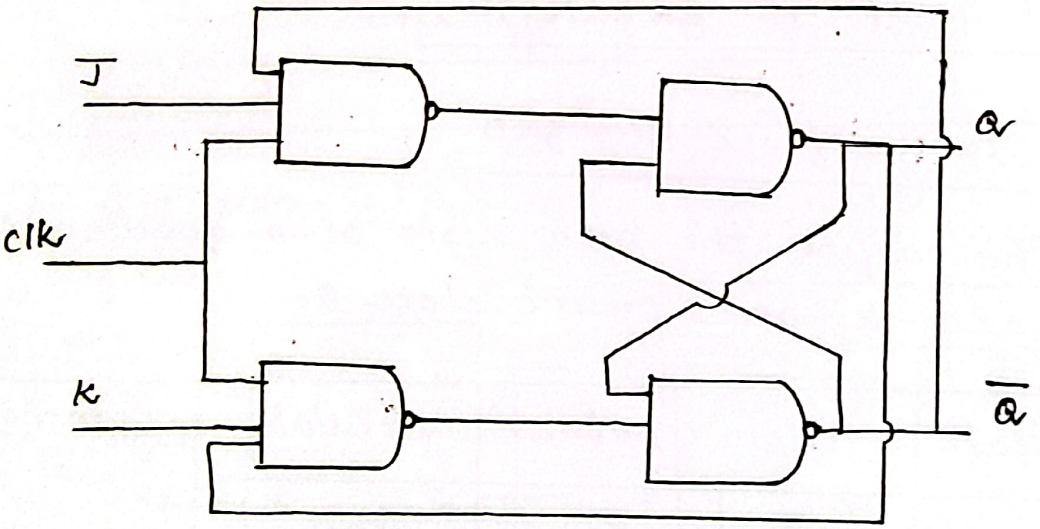
Excitation table

The table that list the required input for a given change of state is called an excitation table.

Present state	Next state	Input
Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

J-K flip flop

- when S=1, R=1 is not used so it is a disadvantage in SR flip flop. To overcome this problem J-K flip flop is used.



Case (i)

when clock = 0, the next state is memory.

case(ii)

when clk = 1, J = 1, k = 0, Q = 1 and $\overline{Q}$ = 0

case(iii)

when clk = 1, J = 0, k = 1, Q = 0 and $\overline{Q}$ = 1

case(iv)

when clk = 1, J = 1, k = 1

Assume , $Q=0$ & $\bar{Q}=1$

$Q = 0, 1, 0, 1 \ldots$

$\bar{Q} = 1, 0, 1, 0 \ldots$

The input $J = K = 1$, toggles the flip flop output.

Truth table for JK flip flop

Clk	J	K	Qn+1
0	x	x	Qn.
1	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	$\bar{Q}n$ (toggle)

Characteristic table of J-K flip flop.

This define the next state as a function of input J and K and present state Qn.

Present state	data input		Next state	comment
Qn	J	K	Qn+1.	
0	0	0	0	No change.
0	0	1	0	Reset
0	1	0	1	Set
0	1	1	1	Toggle
1	0	0	1	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	0	Toggle.

The characteristic equation of JK flip flop.

Qn \ JK	$J\bar{K}$	$\bar{J}\bar{K}$	JK	$J\bar{K}$
$\bar{Q_n}$	0	0	1	1
Q_n	1	0	0	1

$$Q_{n+1} = J\bar{Q_n} + \bar{K}Q_n.$$

Excitation table.

Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

T-flip flop.

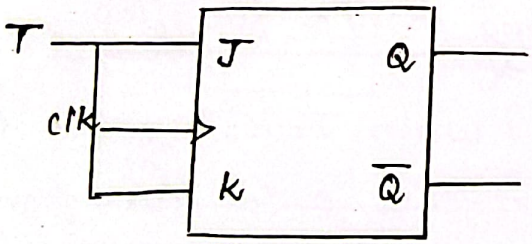
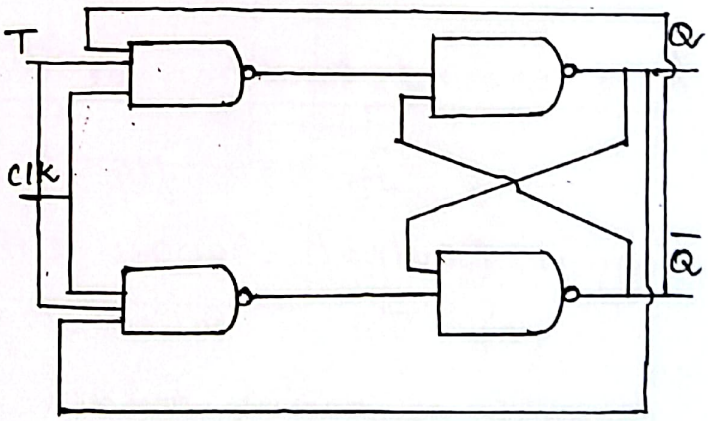


fig:- Logic symbol.



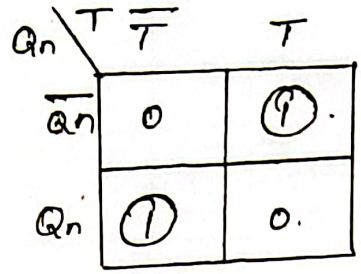
Truth table for T flip flop.

clk	T	Q_{n+1}
0	x	Q_n (memory)
1	0	Q_n (memory)
1	1	$\bar{Q_n}$ (toggling)

Characteristic table

Qn	T	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

Characteristic Equation



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

Excitation Table

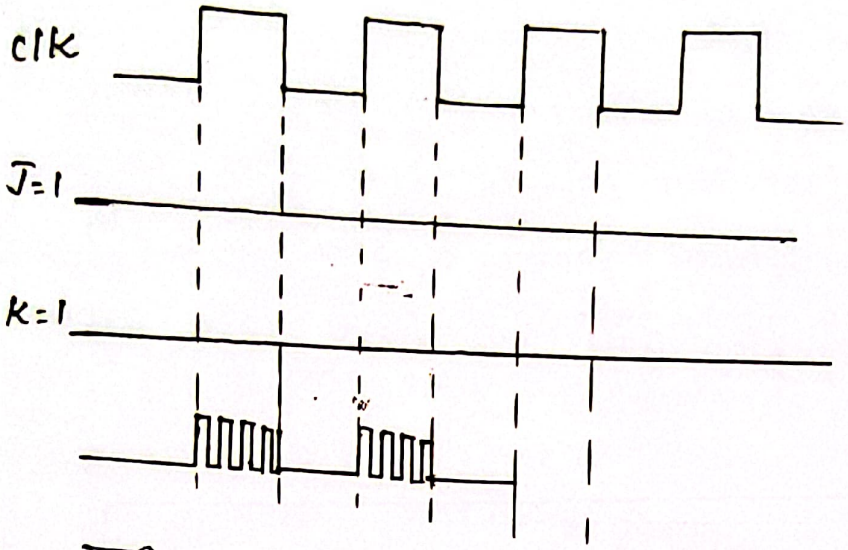
Present state	Next state	Input
Qn	Qn+1	T
0	0	0
0	1	1
1	0	1
1	1	0

Race around condition or Racing in JK flip flop

In JK flip flop, when J = K = 1, the output toggles (output changes either from 0 to 1 or from 1 to 0) This is called toggling output or uncontrolled changing or racing condition.

This is because the clock pulse duration is more than the propagation delay of flip flop, to avoid this we need to adjust the clock pulse duration or we need to put restriction on clock pulse width, as

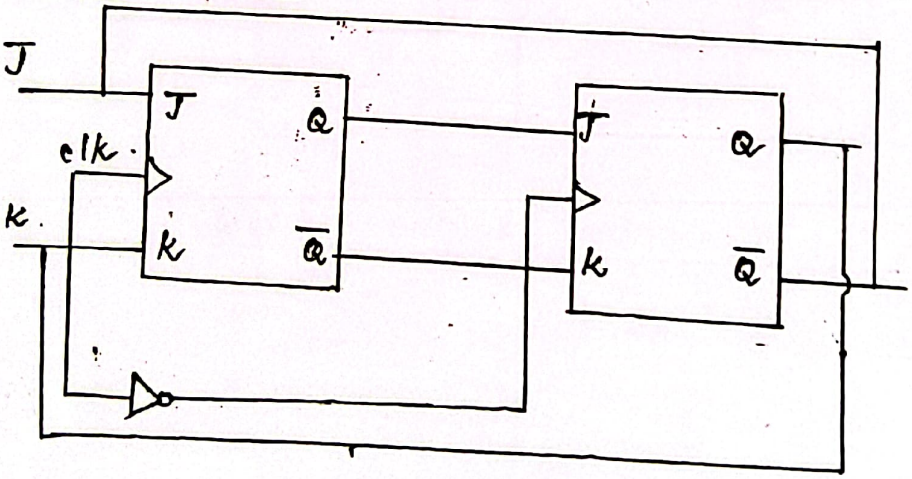
clock pulse (t) & propagation delay using Master slave configuration.



This condition is called as Race around condition.

Master-slave JK flip flop.

The block diagram of a master slave JK. is shown below.



It consists of clocked JK flip flop as a master and clocked SR flip flop as a slave. The clock signal is connected directly to the master flip flop but it is connected through inverter to the slave flip flop.

Set-24.

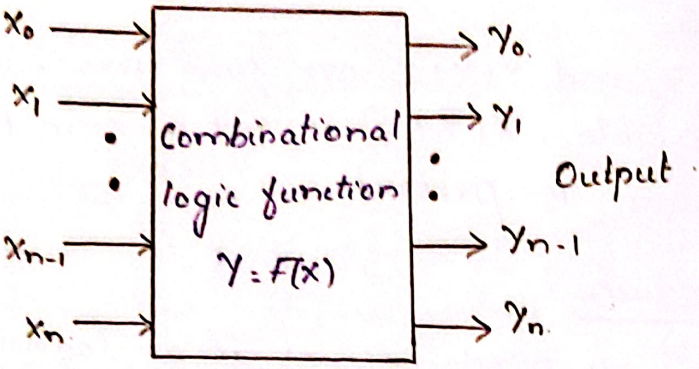
(432)

Unit-II

Combinational circuits

Combinational Logic.

A digital circuit is combination if its output is depending on input. The Combinational logic circuit is memory less. This logic circuit deals with the method of combining basic gates to get desired solution. combinational logic circuit can be constructed using logic gates and without feed back from output to input.



X is the set of input variable $x_0, x_1, x_2 \ldots$ to x_n and Y is the set of output variable $Y_0, Y_1, Y_2 \ldots Y_n$

→ The output Y_0 is a function of $x_0, x_1, x_2 \ldots$ to x_n and it is mathematically written as

$$Y_0 = F(x_0, x_1, x_2, \ldots \text{to } x_n)$$

Elements of combinational Logic.

Some of the basic definition are explained below

Literal :-

It is a Boolean Variable. It will be either primed or unprimed state in the logic expression

Example :-

X and $\bar{X}$ are both literal. Similarly, $ABC\bar{D}$ consists of four literals A, B, C and $\bar{D}$

Product Term :-

A product term is the logical product (AND) of literals.

Minterm

Example :-

$x, x\bar{y}, xyz$ are the product of terms when x, y, z are Boolean Variable.

But $x+y+z$ is not a product term due to presence of pluse (+) sign in the expression.

Sum Term :-

A sum term is sum of literals or the logical OR of literals.

Example :-

$x+\bar{y}$ and $x+y+z$ are sum term, when x, y, z are Boolean Variable. $x(\bar{y}+z)$ is not a sum term as the logical AND operation is present.

Sum of products :-

Sum of product (SOP) is a logical expression in which OR of multiple product term are present. Each product term is the logical AND of literals.

Example :-

SOP expression is $y + x\bar{y} + xyz$

Products of sums

Product of sum (POS) is the logical expression in which AND of multiple OR terms are present. Each sum term is the OR of literals

Example :-

$(x+x\bar{y})(xy+z)(\bar{y}+z)$

AND) Minterms

It is a special type of product (AND) term. It is a product term which contain all the input variable that make up a Boolean expression

Maxterm :-

A maxterm is a special type (OR) term. A maxterm is a sum term that contain all the input variable that make up a Boolean expression.

Canonical forms :-

Canonical is defined as "conforming to a general rule". The rule for boolean logic is that each term used in a boolean equation must contain all the variables.

Canonical sum of product

A canonical sum of product (SOP) is a complete set of minterms that defines when an output variable is a logical 1. Each miniterm corresponds to the row in the truth table when the output function is 1

Canonical product of sums

A canonical product of sums (POS) is a complete set of max-terms that defines when an output variable is a logical '0'. Each maxterm corresponds to the row in the truth table when the output function is 0.

Sum of Minterm

Sum of Minterm is the logical expression in which OR of multiple product terms are present. Each product term is the logical AND of literals.

Absorption property

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

DeMorgan's Theorems

DeMorgan suggested two theorems that form an important part of Boolean algebra.

In the equation form they are

1. $\overline{AB} = \overline{A} + \overline{B}$
2. $\overline{A+B} = \overline{A} \cdot \overline{B}$

1. $\overline{AB} = \overline{A} + \overline{B}$

The complement of a product is equal to the sum of the complements. This is illustrated by truth table

A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

2. $\overline{A+B} = \overline{A} \cdot \overline{B}$

The complement of a sum is equal to the product of the complements. The truth table is illustrated as

A	B	$\overline{A+B}$	$\overline{A}\,\overline{B}$
0	0	1	1
1	1	0	0
1	0	0	0
1	1	0	0

Representation of Logic function - SOP and POS forms

To implement a Boolean function with a less nu of gates we have to minimize literals and the number of terms. usually, literals (Boolean variable in complemented or uncomplemented form) and terms are arranged in one of the two standard form of switching equation.

→ Sum of product form (SOP)
→ Product of sum form (POS)

Sum of product form

The words sum and product are derived from the symbolic representation of the OR and AND functions by + and .

$$AB + Bc + A\bar{c}$$

Product of sum form

A product of Sum expression is a Sum term or several sum terms logically multiplied, An ANDing of ORed variable such as.

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

Standard SOP and standard POS forms.

We can realize that in the SOP form, all the individual terms do not involve all literals. For example, in expression $AB + AB\bar{C}$ the first product do not contain literal C.

If each term in SOP form contain all the literals then the SOP form is known as standard or canonical SOP form.

→ Each individual term in the standard SOP form is called minterm.

Example.

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

Each product term consists of all literals in either complemented form or uncomplemented form

→ If each term in POS form contain all the literals then the POS form is known as standard or canonical POS form.

* Each individual term in the standard POS form is called maxterm.

Converting Expression in standard SOP or POS form.

Sum of product form can be converted to standard Sum of products by ANDing the terms in the expression with terms formed by ORing the literal and its complement which are not present in that term.

→ For example for a three literal expression with literals A, B and C, if there is a term AB, where C is missing, then we form term $(C+\bar{C})$ and AND it with AB. Therefore we get $AB(C+\bar{C}) = ABC + AB\bar{C}$

Steps to convert SOP to standard SOP form.

Step 1: Find the missing literal in each product term if any.

Step 2: AND each product term having missing literal/s with term/s form by ORing the literals and its complement.

Step: 3 Expand the term by applying distributive law and reorder the literals in the product terms.

Step 4: Reduce the expression by omitting repeated product term if any Because $A+A=A$

Example :-

1. Convert the given expression in standard SOP form

$$f(A,B,C) = AC + AB + BC$$

Step 1: Find the missing literals in each product term

$$f(A,B,C) = AC + AB + BC$$

Literal A is missing

Literal C is missing

Literal B is missing

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

Original product terms

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

Missing literals and their complement.

Step 3: Expand the terms and reorder literals.

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

Reorder : $f(A,B,C) = ABC + A\bar{B}C + ABC + AB\bar{C} + ABC + \bar{A}BC.$

Step: 4 Omit repeated product terms.

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

Steps to convert POS to standard POS form

Step 1 . Find the missing literals in each sum term if any

Step 2 : OR each sum term having missing literals with terms form by ANDing the literals and its complement .

Step 3 : Expand the term by applying distributive law and reorder the literals in the sum term.

Step 4 : Reduce the expression by omitting repeated sum term if any . Because $A \cdot A = A$

Example :-

1. convert the given expression in standard POS form.

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

Step 1 : Find the missing literals in each sum term.

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

literal B is missing

literal A is missing

literal C is missing

Step 2 : OR sum term with (missing literal . its complement)

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

Missing literals and their complement .

Step 3 : Expand the terms and reorder literals.

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

Step 4 : Omit repeated sum term.

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

Karnaugh Map Minimization

→ The map method first proposed by Veitch and modified by Karnaugh hence it is known as the Veitch diagram or karnaugh map.

→ The basic of this method is a graphical chart known as karnaugh map (k-map).

→ It contains boxes called cells.

→ Each of the cell represent one of the 2^n possible product that can be formed from a variables.

→ Thus a 2-variable map contain, $2^2 = 4$ cells a 3 variable contain $2^3 = 8$ cell and so on.

Examples: 2 Variable k Map (A, B).

A \ B	$\bar{B}$ 0	B 1
$\bar{A}$ 0	00 m_0	01 m_1
A 1	10 m_2	11 m_3

→ It is important to note that when we move from one cell to next along any row (or) from one cell to the next along any column, one and only one variable in the product term change.

3. Variable Map.

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	000 m_0	001 m_1	011 m_3	010 m_2
A 1	100 m_4	101 m_5	111 m_7	110 m_6

AB \ C	$\bar{C}$	C
$\bar{A}\bar{B}$ 00	000 m_0	001 m_1
$\bar{A}B$ 01	010 m_2	011 m_3
AB 11	110 m_6	111 m_7
$A\bar{B}$ 10	100 m_4	101 m_5

4 Variable Map

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0000 m_0	0001 m_1	0011 m_3	0010 m_2
$\bar{A}B$ 01	0100 m_4	0101 m_5	0111 m_7	0110 m_6
AB 11	1100 m_{12}	1101 m_{13}	1111 m_{15}	1110 m_{14}
$A\bar{B}$ 10	1000 m_8	1001 m_9	1011 m_{11}	1010 m_{10}

Representing standard SOP in K-Map

→ A Boolean expression is the sum of product form can be plotted on the karnaugh map by placing a '1' in each cell corresponding to a term (minterm) in the sum of product expression. Remaining cell are filled with zero.

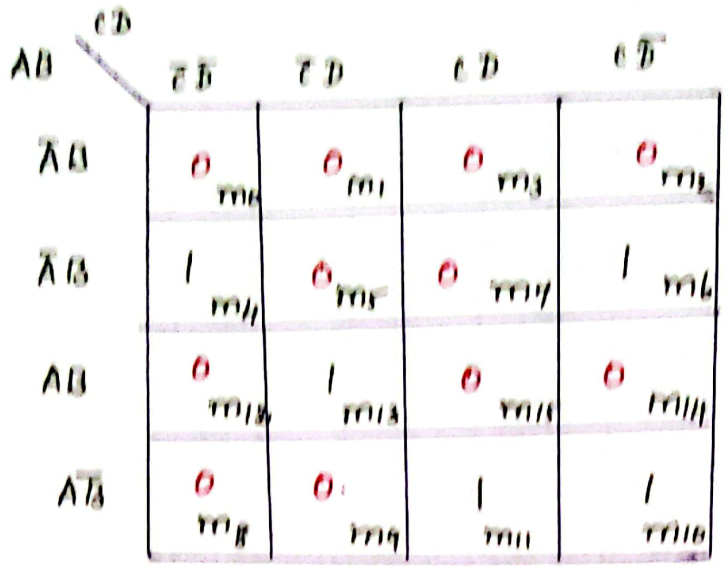
Example

1. plot Boolean expression $Y = AB\bar{C} + ABC + \bar{A}\bar{B}C$ on the karnaugh Map.

The expression has 3 variable and hence it can be plotted using 3 variable Map.

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0 m_0	1 m_1	0 m_3	0 m_2
A 1	0 m_4	0 m_5	1 m_7	1 m_6

2. Plot $Y = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + A\bar{B}C D + AB\bar{C}D$ on the karnaugh map

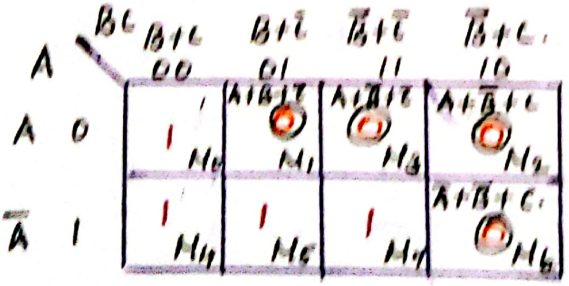


Representing standard POS on K-Map

→ A Boolean expression in the product of sums can be plotted on the karnaugh map by placing a '0' in each cell corresponding to the term. Remaining cells are filled ones.

Examples:

1. Plot Boolean Expression $Y = (A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)(A+B+\bar{C})$ on the karnaugh map.



2. Plot Boolean Expression

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

Soln,

$(A+B+C+\bar{D}) = M_1$

$(A+\bar{B}+\bar{C}+D) = M_6$

$(A + B + \bar{C} + \bar{D}) = M_3$

$(\bar{A} + \bar{B} + C + \bar{D}) = M_{13}$

$(\bar{A} + \bar{B} + \bar{C} + D) = M_{14}$

AB \ CD	C+D (00)	C+$\bar{D}$ (01)	$\bar{C}$+$\bar{D}$ (11)	$\bar{C}$+D (10)
A+B (00)	1 M_0	0 M_1	0 M_3	1 M_2
A+$\bar{B}$ (01)	1 M_4	1 M_5	1 M_7	0 M_6
$\bar{A}$+$\bar{B}$ (11)	1 M_{12}	0 M_{13}	1 M_{15}	0 M_{14}
$\bar{A}$+B (10)	1 M_8	1 M_9	1 M_{11}	1 M_{10}

Grouping cells for simplification

→ The grouping is nothing but combining terms in adjacent cells.

Grouping two adjacent ones (pair).

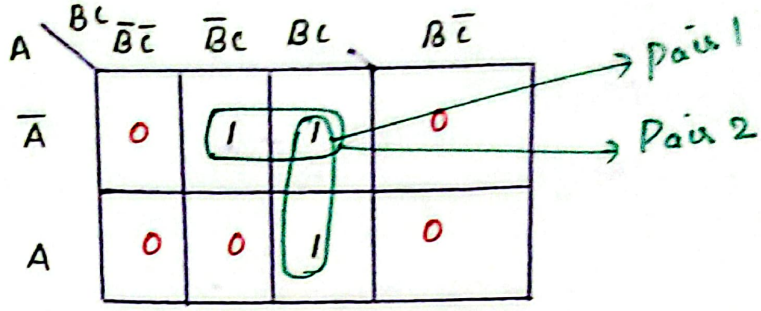
A \ Bc	$\bar{B}\bar{c}$	$\bar{B}c$	Bc	$B\bar{c}$
$\bar{A}$	0	1	1	0
A	0	0	0	0

Pair

$Y = \bar{A}c$.

A \ Bc	$\bar{B}\bar{c}$	$\bar{B}c$	Bc	$B\bar{c}$
$\bar{A}$	0	0	0	0
A	1	0	0	1

$Y = A\bar{c}$

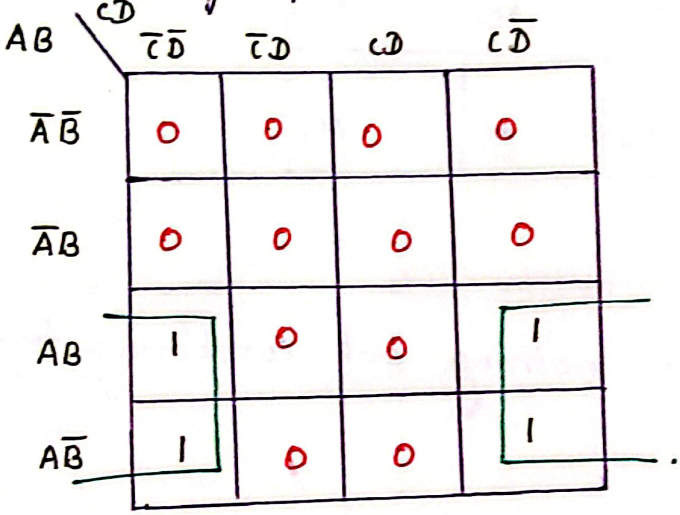


{Here overlap possible}

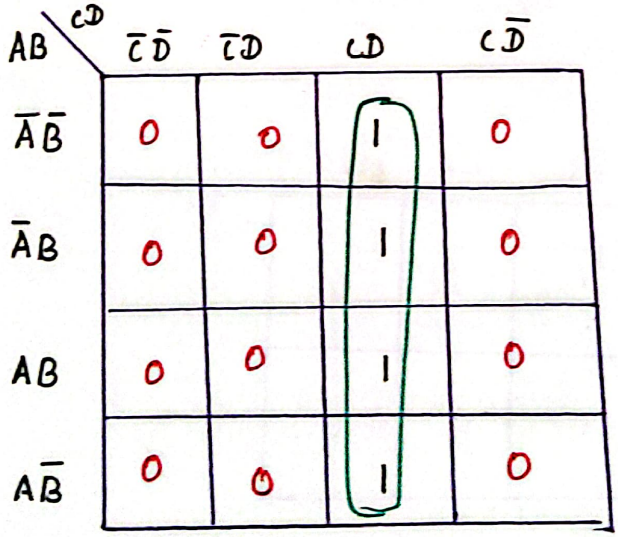
$$Y = \bar{A}C + BC$$

Grouping four adjacent ones (Quad)

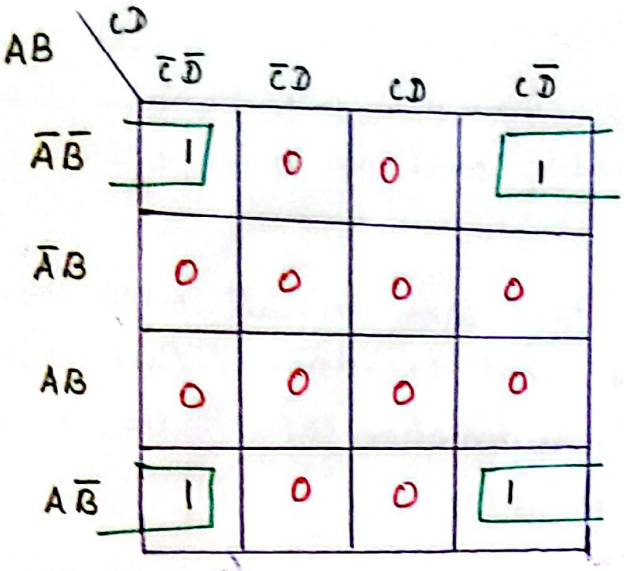
→ In a Karnaugh map we can group four adjacent ones. The resultant group is called Quad.



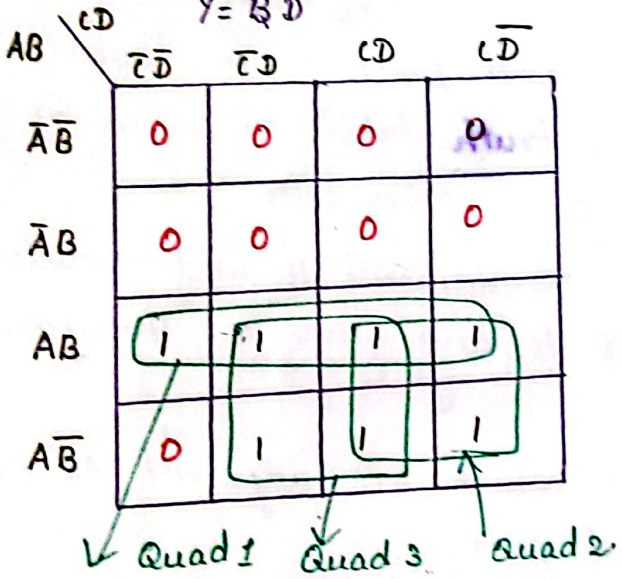
$$Y = A\bar{D}$$



$$Y = CD.$$



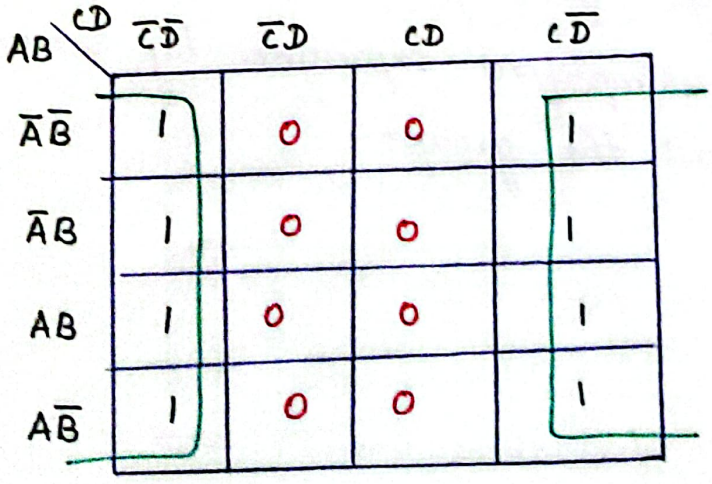
$Y = \bar{B}\bar{D}$



$Y = AB + A\bar{C} + A\bar{D}$.

Grouping Eight Adjacent Ones (Octet)

→ In a Karnaugh map we can group eight adjacent one. The resultant group is called as octet.



$Y = \bar{D}$

Minimization of SOP Expression

A pair of 1s eliminate one variable, a quad 4 1s eliminate two variable and an octet of 1s eliminate three variables

→ Variables that are same in all with the group must appear in the final expression. Each group gives us a product term and summation of all the product term gives us a Boolean expression.

Generalize procedure to simplify Boolean Expression

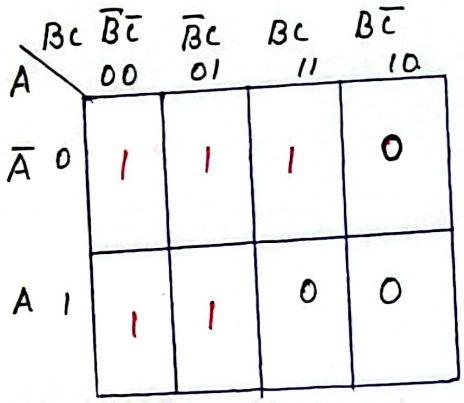
1. plot the k-map and place 1s in those cells corresponding to the 1s in the truth table or sum of product expression. Place 0s in other cells.
2. check the k-map for adjacent 1s and encircle those 1s which are not adjacent to any other 1s. These are called isolated 1s.
3. check for those 1s which are adjacent to only one other 1 and encircle such pair
4. check for quads and octets of adjacent 1s even if it contain some 1s that have already been encircled. While doing this make sure that there are minimum number of groups.
5. combine any pairs necessary to include any 1s that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

Example:

1. Minimize the expression $Y = A\bar{B}C + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}B\bar{C}$.

Soln,

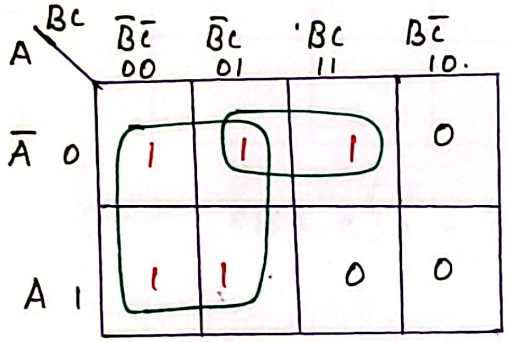
Step 1: k-map for three variable and it is plotted according to the given expression.



Step: 2 There is no isolated 1s.

step: 3 1 in the cell 3 is adjacent only to 1 in the cell 1. This pair is combined and referred to as group 1

step: 4. There is no octet, but there is a quad. cell 0 1, 4 an 5 form a quad. This quad is combined and referred to as group 2.



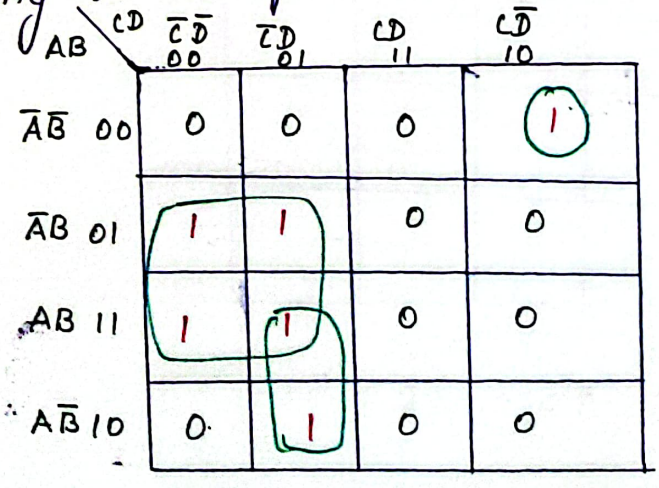
$Y = \bar{A}C + \bar{B}$

2. Minimize the expression.

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

Soln,

The kmap for four variables and it is plotted according to the given expression.



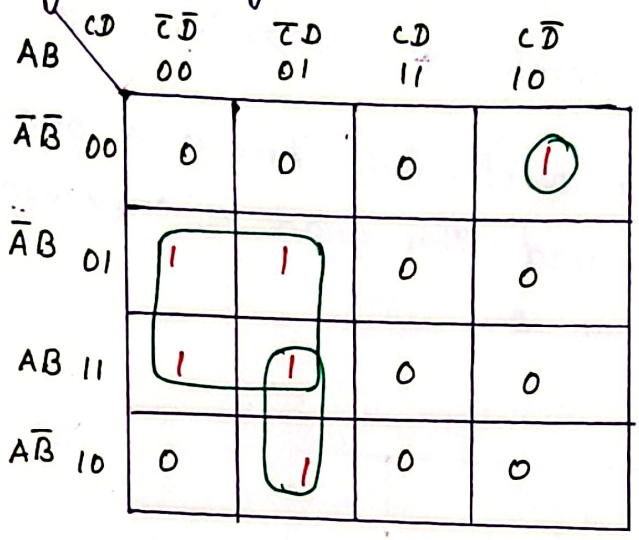
$Y = \bar{A}\bar{B}C\bar{D} + A\bar{C}D + B\bar{C}$

3. Reduce the following four variable function to its minimum sum of products forms.

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

Soln,

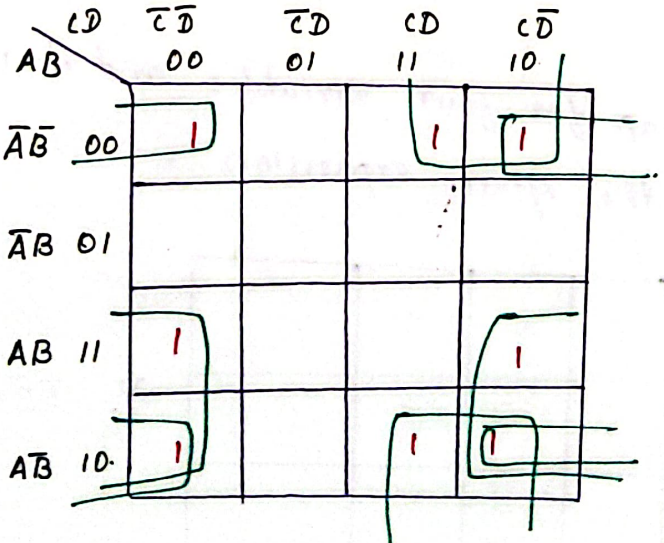
The K-map for four variables and it is plotted according to the given expression.



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

4. Reduce the following four variable function to its minimum sum of products form :-

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



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

5. Reduce the following function to its minimum sum of products form.

$Y = \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + ABCD + A\bar{B}CD$

Soln:

K map for four variable and it is plotted according to the given expression

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	1	0	0
$\bar{A}B$ 01	0	1	1	1
AB 11	1	1	1	0
$A\bar{B}$ 10	0	0	1	0

$Y = \bar{A}\bar{C}D + \bar{A}BC + AB\bar{C} + ACD$

6. Reduce the following function using karnaugh map technique and implement using basic gates.

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

Soln:

The given function is not in the standard sum of product form. It is converted in to standard sop form as given below.

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

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

$= \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D + AB\bar{C}\bar{D} + \bar{A}BCD + \bar{A}B\bar{C}D + ABC\bar{D}$

The K.map for four variable and it is plotted according to expression in standard sop form.

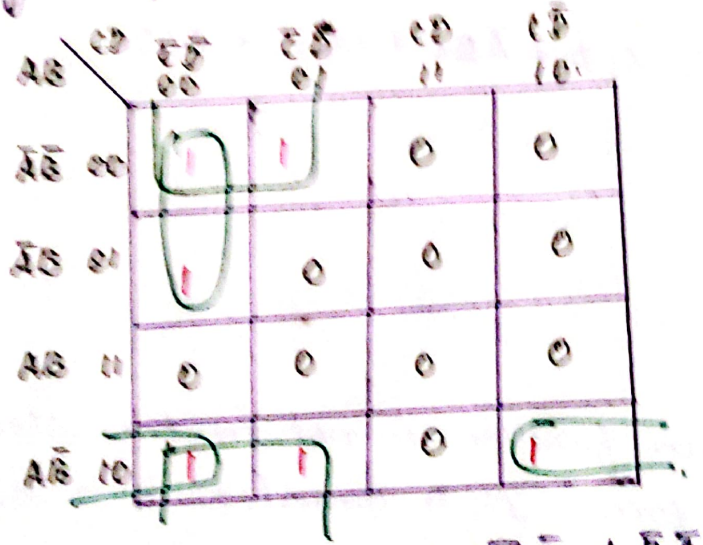
$Y = A\bar{B}\bar{C} + \bar{A}\bar{D}$

* Reduce the following function using K-map technique.

$f(A,B,C,D) = \Sigma m(0,1,4,5,9,13)$

Soln:

The K-map for four variables and it is plotted according to given minterm



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

Incompletely specified function (Don't care terms).

In some logic circuit, certain input condition never occur, therefore corresponding output never appear. In such case the output level is not defined, it can be either HIGH or LOW.

→ These output level are indicated by 'x' or 'd' in the truth table and are called don't care outputs or don't care condition or incompletely specified function.

1. Find the reduced SOP form of the following function.

$$f(A,B,C,D) = \Sigma m(1,3,7,11,15) + \Sigma d(0,2,4)$$

Soln,

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	X	1	1	X
$\bar{A}B$ 01	X	0	1	0
AB 11	0	0	1	0
$A\bar{B}$ 10	0	0	1	

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
00	1	1	1	1
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

2. Reduce the following function using karnaugh map technique.

$$f(A,B,C) = \Sigma m(0,1,3,7) + \Sigma d(2,5)$$

Soln,

Sept-18 (BSE) (4 degrees) (2014)

Unit - V

VHDL

Introduction

We are familiar with a design of digital system The basic steps involved in this process are,

a. specify the desired behaviour of the circuit.
b. Synthesize the circuit.
c. Implement the circuit
d. Test the circuit to check whether the desired specification meet.

⇒ But as the size and complexity of digital system increases, they cannot be designed manually, their design become highly complex.

⇒ So computed Aided Design (CAD) tools are used in the design of such system. One such a tool is a Hardware Description Language (HDL).

Hardware Description Language (HDL)

The Hardware Description Language (HDL) is a computer programming Language used for formal description of digital Logic circuit.

⇒ They can describe the circuit's operation, its design and organisation, and test to verify its operation by Simulation.

⇒ HDL are standard text-based expression of the structure and behaviour of digital electronic system.

⇒ One of the popular hardward description language is Register Transfer Language (RTL)

Very High Speed Integrated Circuit Hardware Description Language (VHDL)

VHDL is the VHSIC Hardware Description Language VHSIC is an abbreviation for Very High speed Integrated circuit. It can describe the behavior and structure of digital Logic circuit.

⇒ VHDL is used for modeling digital circuits from simple gates to complex systems.

⇒ Digital system design using VHDL consists of two design unit, namely, primary design unit and secondary design unit.

⇒ The entity declaration, the package declaration and configuration declaration are the primary design unit.

⇒ The secondary design unit is the architecture body and package body which are related with primary design unit.

Structure of VHDL Module.

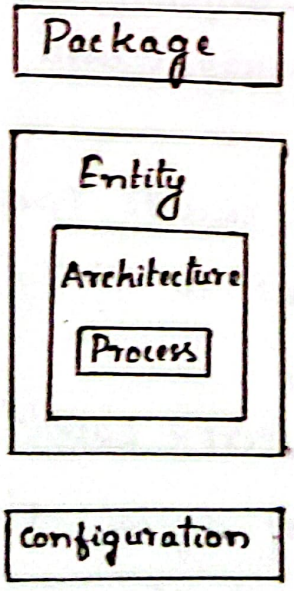


Fig:- Structure of VHDL module.

The main component of a VHDL description are.

* Package (optional)
* Entity
* Architecture
* configuration (optional)

→ A design may include any number of package, entity, architecture and configuration declaration. It is important to note that the entity and architecture blocks are compulsorily required, however, the package and configuration blocks are optional.

1. Entity {What is entity (2 Mark)}

All digital design must be expressed in terms of ENTITY. The entity in VHDL specifies the name of the entity and it also refer to any digital device namely AND, NAND, OR, NOR, XOR, XNOR and NOT gate, flipflops, ALU etc.

⇒ "The keyword ENTITY signifies the starting of an ENTITY statement. Each entity is uniquely assigned a name and its input and output signals through PORT

⇒ Each PORT is associated with the two keywords IN and OUT to represent input signal and output signals respectively.

⇒ The following keywords ENTITY IS PORT IN, OUT, INOUT and END are used in VHDL code for any design ENTITY.

The syntax of an ENTITY is as follows;

```
ENTITY entity-name IS
PORT (list of input port names, list of output port names
                                and their types);

END entity-name;
```

and R = 1. latch will

The example of an ENTITY and-gate is given below,

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY and_gate IS
PORT (
    a: IN std_logic;
    b: IN std_logic;
    c: IN std_logic;
    d: OUT std_logic);
END and_gate;
```

⇒ Here, the name of ENTITY is and_gate. The entity has four ports in the PORT.

* Three ports are for IN mode and one port is of OUT mode.

⇒ The three data input ports (a, b, c) of and_gate are std_logic type and the data output port (d) is also std_logic type.

⇒ The keyword word END signifies the end of the ENTITY declaration.

⇒ The symbol colon (:) is used for separator

⇒ The symbol semicolon (;) is used as terminator.

⇒ std_logic_vector (represent array of bit)

⇒ std_logic include value that allow us to accurately simulate the circuit.

2. ARCHITECTURE

An ARCHITECTURE specifies the internal operation of the module.

⇒ The ARCHITECTURE describe the actual implementation of the functionality of the ENTITY. It contain the statement, interconnected component to represent the behaviour of the ENTITY and the structure of the ENTITY.

The syntax for the architecture varies, depending on the model. generally,

* data flow model
* behavioral model
* structural model Architecture are used

in VHDL to design digital system.

⇒ An ARCHITECTURE body has declarative part and statement part.

The keywords ARCHITECTURE, OF, IS, BEGIN and END are used for any Architecture body.

The syntax of ARCHITECTURE body is given below,

```
ARCHITECTURE architecture_name OF entity_name IS
declarations;
BEGIN;
statements;
END architecture_name;
```

Data Flow Model.

In data flow modeling of ARCHITECTURE, concurrent statement are used and also executed concurrently.

⇒ The concurrent statement are expressed using concurrent signal assignment, conditional signal assignment and selected signal assignment.

The syntax of data flow model is,

```
ARCHITECTURE architecture-name OF entity-name IS
signal - declaration;
BEGIN
concurrent - statement
END architecture-name.
```

Concurrent signal Assignment

The concurrent signal assignment statement assign a numeric value. A signal assignment can be identified by the symbol <= .

The syntax of concurrent signal assignment is.

```
Signal x = expression
```

The expression may be any arithmetical expression and logical expression.

```
a <= '0';
c <= a xNOR (NOT b);
```

Conditional signal Assignment.

The conditional signal assignment statement select one of the different value to assign to a signal conditionally.

The Syntax of conditional signal assignment is,

```
Signal x = Value 1 WHEN Condition Else.
           Value 2 WHEN Condition Else.
           .....
           .....
```

The example of this statement is given below,

```
SELECT x = input 0 WHEN  S0 = '0' AND S1 = '0' Else.
           input 1 WHEN  S0 = '1' AND S1 = '0' Else
           input 2 WHEN  S0 = '0' AND S1 = '1' Else.
           input 3.
```

→ When S0 = 0 and S1 = 1 the SELECT value will be the input 2. Therefore, the signal assignment statement is sensitive with S0 and S1

Selected signal Assignment

In selected signal assignment statement, signal will get any one of several different values assigned to it based on the value of a select expression.

⇒ Here keyword OTHERS is used to denote all remaining choices.

The syntax of selected signal assignment statement is

```
WITH expression SELECT
    Signal x = input_1 WHEN Value 1,
               input-2 WHEN Value 2
               . . . . . .
               . . . . . .
               input_n WHEN OTHERS;
```

When the output of expression is equal to Value 1, input 1 is assigned to signal. If the output of expression is equal to Value 2, input 2 can be assigned to signal. When OTHERS values are generated from expression, input_n will be assigned to signal.

Structural Model.

(Explain the concept of structural Model in VHDL.)

In structural modeling, several components are interconnected with input signals. The component statement is used to declare each component, which are used in the netlist. Then the declared components are represented with the actual component in the digital circuit using the PORT MAP statement. After that, signals are used to connect the component together according to the netlist.

The syntax of structure modeling is,

```
ARCHITECTURE architecture - name of entity - name is
component - declaration ;
signal - declaration ;
BEGIN
  PORT MAP - statement
  concurrent - statement.
END architecture - name;
```

Component Declaration

Each COMPONENT declaration statement must be with in a ENTITY and an ARCHITECTURE.

⇒ The component declaration statement declares the name of a component and component interfacing with in netlist of digital circuit.

⇒ The keywords COMPONENT, IS, PORT, END COMPONENT are used in COMPONENT declaration,

The syntax of COMPONENT is,

```
COMPONENT component - name Is
PORT (list of input ports , list of output ports and
                                    their types);
END COMPONENT;
```

Eg,

```
COMPONENT and Is
PORT (a,b : IN bit;
      c   : OUT bit);
END COMPONENT;
```

Behavioral Model.

(Explain the concept of Behavioral Model)

In behavioral Model, statement are executed just like high level computer programming language.

⇒ The PROCESS statement is the main body of this model.

⇒ In this model, sequential statement, Variable declaration, if-then-else statement, case, loop, for loop and while loop statement are generally used to model the behavior of digital circuit.

Keyword: ARCHITECTURE, OF, IS, BEGIN, PROCESS, and END are incorporated in this model.

The syntax of behavioral model is as follow,

```
ARCHITECTURE architecture_name OF entity_name IS
signal_declarations;
function_definitions;
procedure definition;
BEGIN
 PROCESS_block;
 concurrent statements;
 END architecture_name
```

Process.

The process block contain concurrent statement. When multiple PROCESS block exist in an architecture, all PROCESS blocks will be executed simultaneously. Any PROCESS block also contain sequential statement which are executed sequentially.

The syntax of PROCESS block is,

```
PROCESS (sensitivity - list)
Variable declarations;
BEGIN
 Sequential - statement.
END PROCESS.
```

The example of the PROCESS is given below.

```
PROCESS (a,b,c)
BEGIN
 e<= a OR b OR c;
END PROCESS.
```

Variable Declarations

Variable are declared with in a PROCESS.

The syntax of variable assignment is,

```
Signal := expression;
```

Example is,

```
c := a+b;
```

Sequential - statement.

In the Sequential assignment statement, a value to be assigned to a signal. This statement is expressed just like concurrent statement but this statement will be executed sequential.

The syntax for Sequential - statement.

```
Signal <= expression;
```

Example,

```
d<= a AND (b AND c)
```

If then Else statement.

Example,

```
IF (a==b) THEN
  c:=d;
END IF
```

The above statement start with IF. Then the condition (a==b) is followed by the keyword THEN. When the condition is true, the assignment statement (c:=d) will be executed. If the condition is false, no statement are executed.

The Syntax of IF THEN ELSE statement are,

```
IF condition THEN.
  Sequential_statements_1;
Else
  Sequential_statements_2;
END If;
```

3. PACKAGE.

A PACKAGE provide a mechanism to hold data to be shared among several entities. A package consists of two parts,

* Package declaration section
* Package body section

⇒ generally, the package declaration and body are stored together in a separate file. The file name must be same as the Package name.

The Syntax of PACKAGE declaration and PACKAGE BODY are given below.

```
PACKAGE declaration

  PACKAGE package_name IS
  type_declarations ;
  signal_declarations ;
  variable_declarations ;
  component_declarations ;
  function_declarations ;
  procedure_declarations ;
  END package_name

PACKAGE BODY

  PACKAGE BODY package_name IS
  function_declarations ;
  procedure_declarations ;
  END package_name
```

RTL Design.

An RTL (Register Transfer Level) description describes a circuit's registers and the sequence of transfer between these registers but does not describe the hardware used to carry out these operations

The steps in RTL design are

→ determine the number and size of registers needed to hold the data used by the device

→ determine the logic and arithmetic operations that need to be performed on these register contents

VHDL for Combinational Logic

1. Data flow Modeling

A data flow model specifies the functionality of the entity. This functionality is expressed using concurrent signal assignment statements and block statement.

1. OR gate

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
ENTITY OR is
PORT (
a : IN std_logic;
b : IN std_logic;
c : OUT std_logic
);
END OR;
ARCHITECTURE dataflow_OR of OR is
begin
  c <= a or b;
end dataflow_OR
```

The other gate like AND, XOR, NOR, NAND and XNOR gate can be similarly modeled by replacing keyword or in the above program with keywords AND, XOR, NOR, NAND and XNOR. The entity name and architecture name can be changed according.

2. Half adder.

```
LIBRARY ieee;
USE ieee.std_logic_1164.All;
entity half adder is
PORT (
  a: in std_Logic;
  b: in std_Logic;
SUM: out std_Logic;
Carry: out std_Logic
  );
end Half_Adder;
ARCHITECTURE dataflow of half_adder is
 begin
  Sum <= a XOR b;
  Carry <= a and b;
end data flow_Half adder.
```

for carry : A B

Sum : A ⊕ B

3. Full Adder.

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.All;
entity Full_Adder_Design is
Port (
  a: in std_Logic;
  b: in std_Logic;
  c: in std_Logic;
Sum: out std_Logic;
carry: out std_Logic;
);
end Full_Adder_Design;
architecture data flow of full adder is.
begin
```

```
Sum <= a xor b xor c;
Carry <= (a and b) or
         (b and c) or
         (c and a);
end data flow full_adder;
```

Sum : $cin \oplus (A \oplus B)$

Cout : $AB + A\,cin + B\,cin$

A. 4:1 Multiplexer

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity multiplexer_4_1 is
 Port (
    a: in std_logic;
    b: in std_logic;
    c: in std_logic;
    d: in std_logic;
    x: in std_logic;
    y: in std_logic;
  dout: out std_logic
 );
end multiplexer 4_1;
architecture dataflow of multiplexer 4_1 is
begin
 dout <= ((not x) and (not y) and a) or
         ((not x) and y and b) or
         (x and (not y) and c) or
         (x and y and d);
end dataflow multiplexer 4_1
```

a, b, c, d → input

x & y → selected i/p

o/p

$Y: \bar{S}_1 \bar{S}_0 D_0 + \bar{S}_1 S_0 D_1 + S_1 \bar{S}_0 D_2 + S_1 S_0 D_3$

5) 1:4 Demultiplexer using with select vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
entity demultiplexer 1-4 is
PORT (
    din : in STD_Logic;
    sel : in STD_Logic_Vector (1 down 0);
    dout : out STD_Logic_Vector (3 down 0)
    );
end demultiplexer 1-4;
architecture data flow of demultiplexer 1-4 is
begin
 with sel select
 dout <= (din & "000") when "00",
         ('0' & din & "00") when "01",
         ("00" & din & '0') when "10",
         ("000" & din ) when others.
end dataflow demultiplexer 1-4 ;
```

Select input		output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	Din	0	0	0
0	1	0	Din	0	0
1	0	0	0	Din	0
1	1	0	0	0	Din

6. 3 to 8 decoder

```
Library ieee;
use ieee.std_logic_1164.all
entity decoder 3_8 is
port (
    input : in std_logic_vector (2 down to 0);
    output : out std_logic_vector (7 down to 0)
    );
end decoder 3_8;
```

```
architecture data flow of decoder 3-8 is
begin
output (0) <= (not input (2)) and (not input (1)) and (not input(0));
output (1) <= (not input (2)) and (not input (1)) and input (0);
output (2) <= (not input (2)) and input (1) and (not input(0));
output (3) <= (not input (2)) and input (1) and input (0);
output (4) <= input (2) and (not input (1)) and (not input (0));
output (5) <= input (2) and (not input (1) and (not input (0));
output (6) <= input (2) and input (1) and (not input(0));
output (7) <= input (2) and input (1) and input (0);
end data flow decoder 3-8;
```

1. **8 to 3 Encoder**

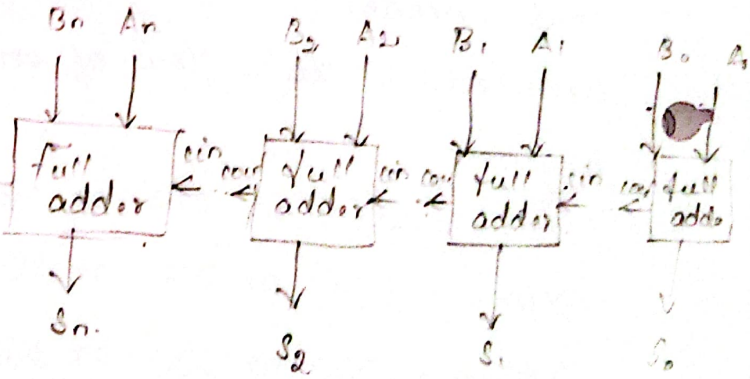
```
Library ieee;
use ieee. std_logic_1164. all
entity encoder is
Port
    (D : in std_logic_vector (7 down to 0);
    Y : out std_logic_vector (2 down to 0);
end encoder;
architecture data flow of encoder is
begin
  Y(0) <= D(1) or D(3) or D(5) or D(7);
  Y(1) <= D(2) or D(3) or D(6) or D(7);
  Y(2) <= D(4) or D(5) or D(6) or D(7);
end data flow.
```

Structural Modeling

In this model component and port map declaration are used to implement the structural modeling.

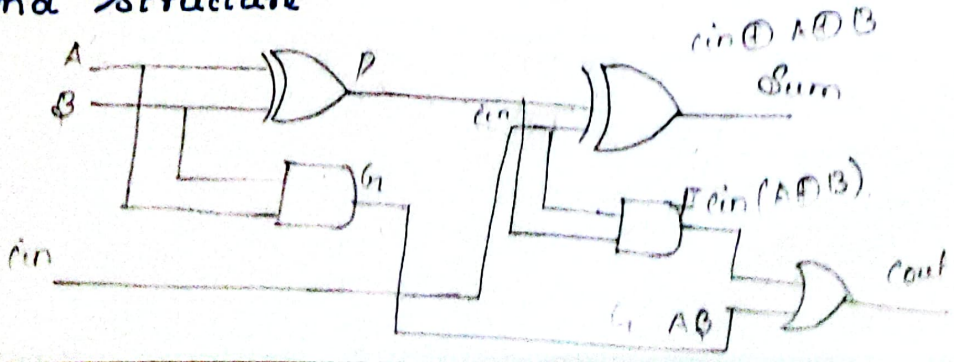
1. Four bit adder.

```
library ieee;
use ieee.std_logic_1164.all;
entity adder_4 is
Port (
      a : in std_Logic_Vector (3 down 0);
      b : in std_Logic_Vector (3 down 0);
      carry : out std_Logic;
      Sum : out std_Logic_Vector (3 down to 0)
     );
end adder_4;
architecture structure of adder 4 is
component full adder is
  Port (a : in std_logic;
        b : in std_logic;
        c : in std_logic;
      Sum : out std_Logic;
      carry : out std_Logic;
      );
  end component;
signal s : std_Logic_Vector (2 down to 0);
begin
 u0 : full adder port map (a(0), b(0), '0', sum(0), s(0));
 u1 : full adder Port map (a(1), b(1), s(0), sum(1), s(1));
 u2 : full adder port map (a(2), b(2), s(1), sum(2), s(2));
 u3 : full adder port map (a(3), b(3), s(2), Sum(3), carry);
end structure adder_4;
```



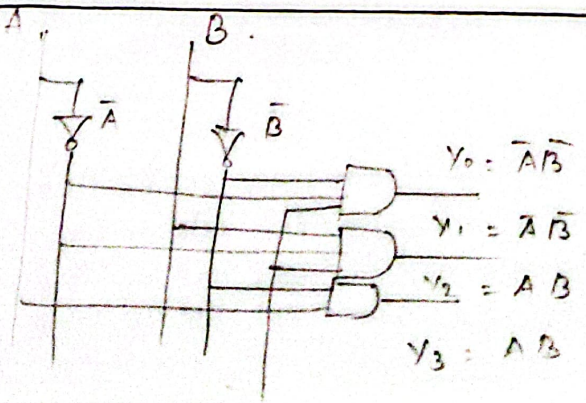
full adder

```
library ieee;
use ieee.std_logic_1164.all;
entity full adder is
Port
    (A, B, cin : in std_logic;
     Sum, carry out : out std_logic);
end full adder;
architecture structure of full adder is
 component xor2
 Port (x, y : in std_logic; z : out std_logic);
 end component
 component and2
 Port (a, b : in std_logic; c : out std_logic);
 end component;
 component or2
  Port (q, r : in std_logic; s : out std_logic);
 end component;
 signal P, G1, F : std_logic;
 begin
  g1 : xor2 Port map (A, B, P);
  g2 : xor2 Port map (P, cin, Sum);
  g3 : and2 port map (A, B, G1);
  g4 : and2 port map (P, cin, F);
  g5 : or2 port map (F, G1, carry out);
  end structure
```



2 to 4 decoder

```
library ieee;
use ieee std_logic_1164.all;
entity decoder is,
Port (x,y : in std_logic;
      D : out std_logic_vector (3 down to 0);
end decoder;
architecture structure of decoder is
component not1
Port (a: in std_logic; b : out std_logic);
end component;
component and2
Port (x,y : in std_logic; z : out std_logic);
end component;
signal xbar, ybar;
begin
  g1 : not1 port map (x, x bar);
  g2 : not1 port map (y, Y bar);
  g3 : and2 port map (xbar, Ybar, D(0));
  g4 : and2 port map (xbar, Y, D(1));
  g5 : and2 port map (x, Ybar, D(2));
  g6 : and2 port map (x, y, D(3));
end structure;
```



Behavioural Modeling

In this modeling, the behaviour of the entity is expressed using sequentially executed, procedural type code.

AND gate:

```
library ieee;
use ieee.std_logic_1164.all;
Entity AND is
Port (
    a : in std_logic;
    b : in std_logic;
    c : out std_logic
    );
End AND;
Architecture behaviour of and is
Begin
   Process (a,b)
Begin
   if ((a='1') and (b='1')) then
   c <= '1';
Else
   c <= '0';
End if;
End process;
end behaviour;
```

(Similarly other gate can be written using their corresponding truth table).

2) 4 : 1 Multiplexer using if - else statement.

```
Library ieee;
use ieee.std_logic_1164.All;
Entity mux 4_1 is
Port ( din : in STD_Logic_VECTOR (3 down to 0)
       Sel : in STD_Logic_Vector (1 down to 0)
       dout : out STD_Logic
     );
end multiplexer 4-1;
architecture behaviour of mux 4_1 is
begin
mux : process (din, sel) is
begin
  if (sel = "00") then
  dout <= din (3);
 else if (sel = "01") then
  dout <= din (2);
 else if (sel = "10") then
 dout <= din (1);
 else
 dout <= din (0);
end if;
end process mux;
end behaviour of Mox 4-1;
```

4:1 Multiplexer using case statement

```
library IEEE
use IEEE.std_logic_1164.all
entity multiplexer_case is
 Port (
   din : in std_logic_vector (3 down to 0);
   sel : in std_logic_vector (1 down to 0);
   dout : out std_logic
 );
end multiplexer_case.
architecture behaviour of multiplexer case is
 begin
   mux : Process (din, sel) is
 begin
   case sel is
    when "00" => dout <= din(3);
    when "01" => dout <= din (2);
    when "10" => dout <= din (1);
    when others => dout <= din (0);
 end case;
 end process mux;
 end behaviour of multiplex case;
```

3. Design of 2 to 4 Decoder using If-Else statement.

```
library IEEE;
use IEEE.std_Logic_1164.all;
entity decoder 2_4 is
```

```
Port (
    din : in std_logic_vector (1 downto 0);
    dout : out std_logic_vector (3 downto 0)
    );
end decoder 2-4;
architecture behaviour of decoder 2-4 is
begin
  encoder : Process (din) is
  begin
    if (din = "00") then
     dout <= "1000";
   else if (din = "01") then
     dout <= "0100";
   else if (din = "10") then
    dout <= "0010";
    else
     dout <= "0001";
   end if
  end process encoder.
  end behavioural of encoder 2.4.
```

design of 2 to 4 decoder using case statement.

```
library IEEE ;
use IEEE . STD_Logic_1164. all ;
entity decoder_case is
Port (
     din : in std_logic_vector (1 down to 0)
     dout : out std_logic_vector (3 down to 0)
    );
end decoder_case;
architecture behaviour of decoder_case is
```

```
begin
  decoder : Process (din) is
begin
  case din is
    when "00" => dout <= "1000";
    when "01" => dout <= "0100";
    when "10" => dout <= "0010";
    when other => dout <= "0010";
  end case ;
end process decoder ;
end behavioural of decoder_case.
```

4. Design of 8:3 priority Encoder using If-else statement.

```
library IEEE;
use IEEE.STD_Logic_1164.all;
entity priority_encoder 8_3 is
Port (
   din : in std_logic_vector (7 down 0);
   dout : out std_logic_vector (2 down 0)
   );
end priority_encoder 8_3;
architecture behaviour of priority_encoder 8_3 is
begin
   Pri_enc : process (din) is
  begin
    if (din(7) = '1') then
       dout <= "000";
    else if (din(6) = '1') then
       dout <= "001";
    else if (din(5) = '1') then
       dout < "010";
```