classmate
Date 09/07/25
Page

Digital Electronics :-

→ Digital electronics is the branch of electronics that utilizes digital circuit such as logic gates, flip-flops etc, to process and store the information in binary form.

Difference between Analog electronics and digital electronics:

Basis of diff	Analog	Digital
1. Definition	→ Analog electronics is the branch of electronics which deals with analog signals.	→ Digital electronics is the branch of electronics that deals with the study of systems with digital signals.
2. Types of signal used	→ Uses continuous time (analog) signals.	→ Uses discrete time signals or two state signals.
3. Components used	→ Analog electronics mostly uses passive circuit components like resistors, capacitors etc.	→ Digital electronics uses active elements only.
4. power consumption	→ consumes more power	→ consumes comparatively less power
5. Noise and distortion	→ High noise and distortion of signals.	→ very low noise and distortion of signals.
6. Used for	→ Mainly help in capturing data from a system.	→ Help in analyzing the data of a system.
7. Applications	→ Widely used in radio and audio devices such as FM radios, Tvs, Telephones etc.	→ Used in computers, data processing and storage, automation, digital watches and many other digital devices.

: Number System and Codes :

Number Systems :-

→ A number system is defined as the technique of writing to express numbers.

→ The number system has different bases and the most common of them are the decimal, binary, octal and hexadecimal.

→ The base or radix of the number system is the total number of the digit used in the number system.

Need of Number Systems :-

→ Computers - understand machine language. Every letter, symbol etc. that we write in the instructions given to computer, it gets converted into machine language.

→ The machine language comprises of numbers. In order to understand the language used by computers and other digital system, it is crucial to have a better understanding of number system

Types of Number System :-

→ There are various type of number system used for representing information

→ The number systems are of following types.

1. Binary Number System
2. Decimal Number System
3. Octal Number System
4. Hexadecimal Number System

1. Binary Number System :-

→ Generally, a binary number system is used in the digital computers.

→ In this number system, it carries only two digits either 0 or 1. So the base or radix of the number system is 2.

→ There are two types of electronics pulses present in a binary number system. The first one is the absences of an electronic pulse representing '0' and second one is the presence of electronic pulse representing '1'.

→ Each digit is known as bit

→ A four-bit collection (1101) is known as nibble

→ A collection of eight bits (11100111) is known as a byte

11/07/2025

Examples :- a) $(10101)_2$
b) $(0.101)_2$
c) $(11001.11)_2$

2. Decimal Number system :-

→ The decimal numbers are used in our day to day life.

→ The decimal number system contains ten digits and these digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

→ The base or radix of the number system is 10 because total 10 digits are available in the number system.

Examples :- a) $(2546)_{10}$
b) $(36.46)_{10}$
c) $(4.567)_{10}$

3. Octal Number System :-

→ The Octal Number System is a number system which uses eight digits to express any number.

→ The digits used are 0, 1, 2, 3, 4, 5, 6 and 7.

→ The base of the octal number system or radix is 8. This is because the total number of digits in the number system is 8.

→ Example :- a) $(1234)_8$

b) $(765)_8$

c) $(1024.06)_8$

4. Hexadecimal Number System :-

→ It is another technique to represent the number in the digital system called the hexadecimal number system.

→ The number system has base of 16 means there are total 16 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) used for representing a number.

→ The ~~signal~~ single-bit representation of decimal values 10, 11, 12, 13, 14 and 15 are represented by A, B, C, D, E and F.

→ Only 4 bits are required for representing a number in a hexadecimal number.

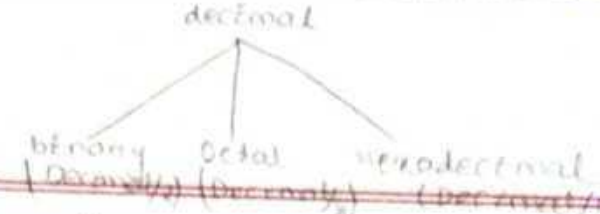
→ Example :- $(198A)_{16}$, $(67.89F)_{16}$

Decimal to binary conversion :-

Rules for converting integers

i) Calculate the reminders by dividing the decimal number by the binary number's base (2).

ii) Divide the result by 2 once again, note down the remainder, and repeat this process until the result is 0.



iii) The remainder, which is the binary counterpart of the provided decimal number, should now be written down in the opposite order.

Q:- Convert the decimal number 43 into its binary equivalent.

2	43	
2	21	1
2	10	1
2	5	0
2	2	1
	1	0

$(43)_{10} \Rightarrow (101011)_2$

Q:- Convert the decimal number 10.25 into its binary equivalent.

2	10	0
2	5	1
2	2	0
	1	

$0.25 \times 2 = 0.5$

$0.5 \times 2 = 1.0$

$(10.25)_{10} \Rightarrow (1010.01)_2$

Decimal to octal conversion:-

Q:- Convert $(158)_{10}$ to equivalent octal number.

8	158	
8	19	6
	2	3

$(158)_{10} \Rightarrow (236)_8$

Q:- Convert $(473)_{10}$ to equivalent octal number.

8	473	
8	59	1
	7	3

$(473)_{10} \Rightarrow (731)_8$

Q:- Convert $(0.513)_{10}$ to octal equivalent.

$0.513 \times 8 = 4.104$
$0.104 \times 8 = 0.832$
$0.832 \times 8 = 6.656$
$0.656 \times 8 = 5.248$
$0.248 \times 8 = 1.984$

$(0.513)_{10} \Rightarrow (0.406651)_8$

Decimal to Hexadecimal conversion:-

Q:- Convert $(423)_{10}$ to $(\ )_{16}$.

16 | 423
16 | 26 — 7
1 — 10

$(423)_{10} \Rightarrow (1A7)_{16}$

Q:- ~~Convert $(2598)_{10}$ to $(\ )_{16}$.~~
Convert $(2598.675)_{10}$ to $(\ )_{16}$.

16 | 2598
16 | 162 — 6
10 — 2

$0.675 \times 16 = 10.80$
$0.8 \times 16 = 12.8$

$(A26.ACD)_{16}$

Binary to Decimal conversion:-
Rules for converting integer
i) Each binary digit's positional value should be known. From right to left, multiply each digit by $2^0, 2^1, 2^2, 2^3, 2^4$ and so forth.
ii) Add together all of these numbers to get the corresponding decimal number.

Q:- Convert $(11010)_2$ to $(\ )_{10}$.

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

1 1 0 1 0

$= 16 + 8 + 0 + 2 + 0$

$= (26)_{10}$

Q:- Convert $(1010.01)_2$ to $(\ )_{10}$

1 0 1 0 . 0 1

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

$= 8 + 0 + 2 + 0 . 25$

$= 10.25$

Octal to Decimal conversion:-

Q:- Convert $(726)_8$ to $(\ )_{10}$.

$726 = 7 \times 8^2 + 2 \times 8^1 + 6 \times 8^0$

$= 448 + 16 + 6$

$= (470)_{10}$.

Q:- Convert $(12.2)_8$ to $(\ )_{10}$.

$12.2 = 1 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1}$

$= 8 + 2 + 0.25$

$= (10.25)_{10}$

Hexadecimal to Decimal Conversion:-

Q:- Convert $(27FA)_{16}$ to $(\ )_{10}$.

$27FA = 2 \times 16^3 + 7 \times 16^2 + 15 \times 16^1 + 10 \times 16^0$.

$= 8192 + 1792 + 240 + 10$

$= (10234)_{10}$.

Q:- Convert $(A0F9.0EB)_{16}$ to $(\ )_{10}$.

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

$= 40960 + 0 + 240 + 9 + 0 + 0.05 + 2.68 \times 10^{-2}$

$= (41209.0573)_{10}$

Date
Page

Binary to octal and vice versa :-

Q:- Convert $(1101010)_2$ to $(\ )_8$.

$001 = 0\times2^2 + 0\times2^1 + 1\times2^0 = 1$

$101 = 1\times2^2 + 0\times2^1 + 1\times2^0 = 4+1 = 5$

$010 = 0\times2^2 + 1\times2^1 + 0\times2^0 = 2$

$(152)_8$.

Q:- Convert $(100100101)_2$ to $(\ )_8$.

$100 = 1\times2^2 + 0\times2^1 + 1\times2^0 = 4$

$100 = 1\times2^2 + 0\times2^1 + 1\times2^0 = 4$

$101 = 1\times2^2 + 0\times2^1 + 1\times2^0 = 5$

$(445)_8$.

Q:- Convert $(562)_8$ to $(\ )_2$..

562

$(101110010)_2$

Octal Number	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Binary to Hexadecimal and viceversa

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

Each hexadecimal numbers is represented into 4 digits

Q:- Convert $(1111\,01)_2 = (\quad)_{16}$

0011 = 3

1101 = D

$(3D)_{16}$

Q:- Convert $(2AB)_{16} = (\quad)_2$

$(001010101011)_2$.

Octal to Hexadecimal :-

-> To convert an octal number to hexadecimal, the simplest way is to first convert to binary and then binary to hexadecimal.

Q: Convert $(756.603)_8$ to $(\quad)_{16}$.

$(756.603)_8 = (111101110.110\ 000\ 011)_2$

$= (1EF.C18)_{16}$.

16/07/2025

Hexadecimal to octal conversion:-

→ To convert a hexadecimal no. to octal, the simplest way is to first convert to binary and then binary to octal.

→ Example:-

Convert $(B9F.AE)_{16}$ to $(\quad)_8$.

$(B9F.AE)_{16}$

$= (101110011111.10101110)_2$

$= (5637.534)_8$.

Q: Convert $(A75C)_{16}$ to $(\quad)_8$.

$(A75C)_{16}$

$= (1010011101011100)_2$

$= (123534)_8$

Practice Questions :-

Q:- convert $(1011010)_2$ to $(\quad)_{10}$.

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

$1\ 0\ 1\ 1\ 0\ 1\ 0$

$= 64 + 0 + 16 + 8 + 0 + 2$

$= (90)_{10}$.

Q: Convert $(0.1101)_2$ to $(\quad)_{10}$.

$2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

$0.\ 1\ 1\ 0\ 1$

$= 0 + 0.5 + 0.25 + 0 + 0.0625$

$= (0.8125)_{10}$.

Q:- Convert $(1111111.0101)_2$ to $(\quad)_{10}$.

$6\ 5\ 4\ 3\ 2\ 1\ 0\ .\ -1\ -2\ -3\ -4$

$1\ 1\ 1\ 1\ 1\ 1\ 1\ .\ 0\ 1\ 0\ 1$

$= 64 + 32 + 16 + 8 + 4 + 2 + 1 + 0 + 0.25 + 0 + 0.0625$

$= (127.3125)_{10}$.

Practice Questions:-

1) Convert $(16512)_8$ to $(\quad)_{10}$.

$1\ 6\ 5\ 1\ 2$ (positions $4\ 3\ 2\ 1\ 0$)

$= 4096 + 3072 + 320 + 8 + 2$

$= (7498)_{10}$.

2) Convert $(0.145)_8$ to $(\quad)_{10}$.

$0.1\ 4\ 5$ (positions $-1\ -2\ -3$)

$= 0 + 0.125 + 0.0625 + 9.76 \times 10^{-3}$

$= 0.19726$

3) Convert $(13D1A)_{16}$ to $(\quad)_{10}$.

$1\ 3\ D\ 1\ A$ (positions $4\ 3\ 2\ 1\ 0$)

$= 65536 + 12288 + 3328 + 16 + 10$

$= 81178$.

4) Convert $(0.2A5)_{16}$ to $(\quad)_{10}$.

$0.2\ A\ 5$ (positions $-1\ -2\ -3$)

$= 0 + 0.125 + 0.0390 + 0.0012$

$= 0.1653$

Representation of Signed Binary Number:-

1's complement:-

→ The 1's complement of any binary number is performed by simply changing all 1's into 0's and 0's into 1's i.e. each bit is replaced by its complement.

Binary number	1's complement
1011	0100
010101	101010
1100.01	0011.10

2's Complement :-

→ The 2's complement of any binary number is determined by adding 1 to 1's complement of that number.

→ 2's complement of number = 1's complement of a no. + 1

Q Find out 1's and 2's complement of 101101

1's → 010010

2's → + 1

010011

Q:- 2's of $(11.01)_2$

11.01

1's :- 00.10

2's :- + 1

00.11.

Representation of Signed Number :-

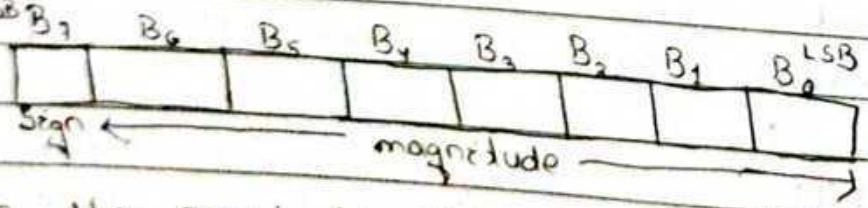
→ There are 2 techniques to represent the sign no.

i) Sign - magnitude form

ii) Complement form.

i) Sign-magnitude form :-

→ The sign-magnitude format for 8-bit signed number.



→ Here, the most significant bit (MSB) represents sign of the number.

→ If MSB is 1, number is -ve and if MSB is 0, number is +ve.

→ The remaining 7 bits represent magnitude of no.

Q: Represent +25 and -25 sign numbers in sign-magnitude form.

+25

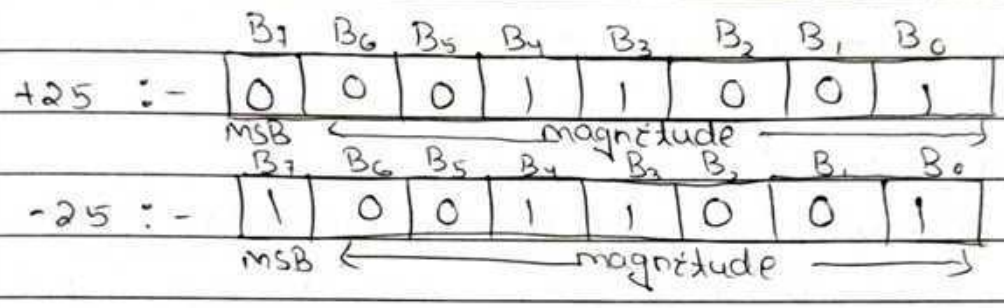
```
2 | 25
2 | 12  1
2 |  6  0
2 |  3  0
      1  1
```

11001

-25

```
2 | 25
2 | 12  1
2 |  6  0
2 |  3  0
      1  1
```

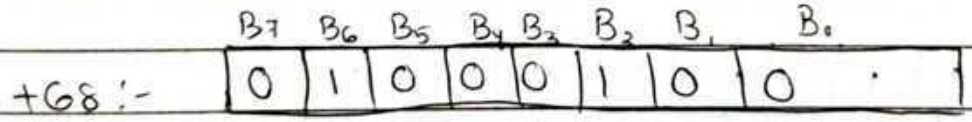
11001

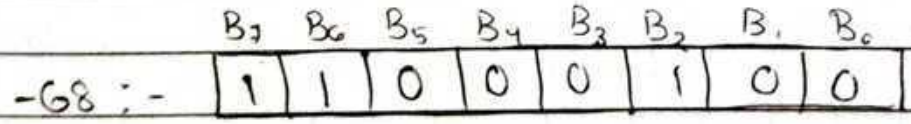


Q:- Represent the following sign-numbers in sign-magnitude form. i) +68
ii) -68

```
2 | 68
2 | 34  0
2 | 17  0
2 |  8  1
2 |  4  0
2 |  2  0
      1  0
```

1 0 0 0 1 0 0





classmate
Date
Page

ii) Note:-
The actual range of numbers we can represent in sign-magnitude format for n-bit would be:
$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

ii) Complement form:-
→ There are 2 complement forms:-
a) 1's complement form
b) 2's complement form

a) 1's complement Representation:-
→ If the number is +ve, the magnitude is represented in its true binary form and a sign bit 0 is placed in the MSB.
→ If the number is -ve, the magnitude is represented in its 1's complement form and a sign bit 1 is placed in the MSB.

Example:-
i) +25 :-

0	0	0	1	1	0	0	1

2 | 25
2 | 12 — 1
2 | 6 — 0
2 | 3 — 1
1 — 1

ii) -25:-

1	1	1	0	0	1	1	0

Q: Represent the following sign numbers in 1's complemented form
i) +68:-

0	1	0	0	0	1	0	0

ii) -68 :-

1	0	1	1	1	0	1	1

b) 2's complement Representation:-

-> If the number is +ve, the magnitude is represented in its true binary form and a sign bit 0 is placed in the MSB.

-> If the number is -ve, the magnitude is represented in its 2's complement form and a sign bit 1 is placed in the MSB.

Q: Represent the following sign numbers in 2's complement form:-

i) +25 ii) -25 iii) +68 iv) -68.

i) +25 :-

0	0	0	1	1	0	0	1

ii) -25 :-

1	1	1	0	0	1	1	1

1's: 1 1 0 0 1 1 0
2's: + 1
1 1 0 0 1 1 1

iii) +68 :-

0	1	0	0	0	1	0	0

iv) -68:-

1	0	1	1	1	1	0	0

1's: 0 1 1 1 0 1 1
2's: + 1
0 1 1 1 0 0

Q: Each of the following no. is a sign-binary number. Determine the decimal value in each case in sign-magnitude form, 1's complement form and 2's complement form if the numbers are in the above 3 formats.

i) 01101

ii) 010111

iii) 10111

iv) 1101010

i) $\overset{4}{0}\overset{3}{1}\overset{2}{1}\overset{1}{0}\overset{0}{1}$ (MSB)

Decimal :- 8 + 4 + 1

= +13

iii) $\overset{2}{0}\overset{1}{1}\overset{0}{1}$

1's :- 1000

Given number	Sign-magnitude form	2's	1's
i) 01101	+13	+13	+13
ii) 010111	+23	+23	+23
iii) 10111	-7	-9	-8
iv) 1101010	-42	-22	-21

iv) $\overset{5}{1}\overset{4}{1}\overset{3}{0}\overset{2}{1}\overset{1}{0}\overset{0}{1}\overset{}{0}$ (MSB)

→ Decimal :- $2^5 + 2^3 + 2^1 = 32 + 8 + 2 = 42$

→ 1's :- $0\overset{4}{1}\overset{3}{0}\overset{2}{1}\overset{1}{0}\overset{0}{1}$

Decimal :- $2^4 + 2^2 + 2^0 = 16 + 4 + 2 = 22$

→ 2's :- $0\overset{4}{1}\overset{3}{0}\overset{2}{1}\overset{1}{1}\overset{0}{0}$

Decimal :- $2^4 + 2^3 + 2^1 = 16 + 4 + 2 =$

1) Binary addition:

The rules of binary addition are the following:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

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

Q: Add the following binary numbers:-

i)
$$\begin{array}{r} 11010 \\ +\ 1101 \\ \hline 100111 \end{array}$$

ii)
$$\begin{array}{r} 01100\,1 \\ +\ 11011 \\ \hline 110100 \end{array}$$

Q:- Add
$$\begin{array}{r} 1101.101 \\ +\ 111.011 \\ \hline 10101.000 \end{array}$$

2) Binary Substraction:-

The rules of binary substraction are the following:

$0 - 0 = 0$

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

$1 - 0 = 1$

$1 - 1 = 0$

Q:- Substract 101 from 1011

$$\begin{array}{r} 1011 \\ -\ 101 \\ \hline 0110 \end{array}$$

Q:-

3) Binary multiplication:-

The rules of binary multiplication are:-

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Q:- Multiply

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

4) Binary Division:-

Q: Divide 11010 by 101

$$\begin{array}{r} 101 \\ 101\overline{)\,11010} \\ -101\downarrow \\ \hline 110 \\ -00 \\ \hline 110 \\ -101 \\ \hline 001 \end{array}$$

← Quotient

→ Remainder

$$\begin{array}{r} 5 \\ 5\overline{)\,26} \\ -25 \\ \hline 1 \end{array}$$

23/07/2025

- Binary Arithmetic using 1's and 2's complement :-

The Binary arithmetic operation can be written as :

$$A+B = B+A$$
$$A-B = A+(-B)$$
$$-A+B = (-A)+B$$
$$-A-B = (-A)+(-B)$$

- The negative binary number can be represented either 1's or 2's complement form.

Binary arithmetic using 1's complement

Case-1: Binary addition rules to add two positive numbers using 1's complement.

→ Convert the positive numbers into the 1's complement form.
→ Add two numbers using binary addition rules to obtain the final result.

Q: Add 14 to 25 using 1's complement method.

25	0 0 0 1 1 0 0 1	(1's complement)
+14	+ 0 0 0 0 1 1 1 0	(1's complement)
39	0 0 1 0 0 1 1 1	

00100111 = +39

Case-2:

→ To add a positive number with a negative number.
→ Write the 1's complement of negative number.
→ Add this number to positive number.
* If there is carry 1 add this to the LSB.

Q: Calculate $(25)_{10} - (14)_{10}$ using 1's complement method.

25	0 0 0 1 1 0 0 1	
-14	1 1 1 1 0 0 0 1	
11	(1) 0 0 0 0 1 0 1 0	
	carry 1	
	0 0 0 0 1 0 1 1	

The MSB is 0, so the result is positive.

Case 3:- To add a negative number with positive number.

→ Write the 1's complement of negative numbers

→ Add this number to positive number.

→ The result will be : (-1's complement of the addition).

Q: Add -25 to 14 using 1's complement method.

+14	0 0 0 0 1 1 1 0	
-25	1 1 1 0 0 1 1 0	(1's complement
-11	1 1 1 1 0 1 0 0	(no carry)

→ There is no carry. The MSB is on 1. So, the result is negative and is in its 1's complement form. The 1's complement of 11110100 is 00001011. The result is therefore, $(-11)_0$.

Case 4:- To add a negative number with a negative number

→ write the 1's complement at both the numbers.

→ Add both the numbers.

→ Add carry to the LSB and the result will be "(-1's complement of the addition)".

Q: Add -14 to -25 using 1's complement method.

-25	1 1 1 0 0 1 1 0	(1's of -25)
-14	1 1 1 1 0 0 0 1	(1's of -14)
-39	① 1 1 0 1 0 1 1 1	
	→ + 1	
	1 1 0 1 1 0 0 0	

$= -(00100111)$

$= -39$

Add the following numbers using 1's complement form.

① +22 & +11

② 22 & -11

③ -22 & +11

④ -22 & -11

① $+22$ → 0 0 0 1 0 1 1 0
$+11$ → 0 0 0 0 1 0 1 1
$+33$ → 0 0 1 0 0 0 0 1

25/07/2025

Binary Arithmetic using 2's complements :-

case-1 : To add a "positive number" with a "negative number".

→ Write the 2's complement of negative number.
→ Add this number to positive number.
→ If there is carry '1', discard it.

Q: Calculate $(46)_{10} - (14)_{10}$ using 2's complement method or substract 14 from 46 using 2's complement method.

14 :- 00001110 (Binary)
11110001 (1's)
-14 :- 11110010 (2's)

$2|14$
$2|7$ 0
$2|3$ 1
1 1

+46 :- 00101110 (Binary)
-14 :- 11110010
① 00100000 = $+(1 \times 2^5)$

$2|46$
$2|23$ 0
$2|11$ 1
$2|5$ 1
$2|2$ 1
1 0

∴ The result is +32.

classmate
Date
Page

case-2: To add a "negative number" with a "positive number"

→ Write the 2's complement of negative number.
→ Add this number to positive number.
→ The result will be "-(2's complement of the addition result)"

Q:- Add -75 with 26 using 2's complement form.

75 :- 01001011 (Binary)
10110100 (1's)
-75 :- 10110101 (2's)

2	26
2	13 0
2	6 1
2	3 0
	1 1

2	75
2	37 1
2	18 1
2	9 0
2	4 1
2	2 0
	1 0

```
        1   1
-75   1 0 1 1 0 1 0 1
+26   0 0 0 1 1 0 1 0
      -----------------
      1 1 0 0 1 1 1 1
```

1's :- 00110000
2's :- 00110001 = $\sum a_i 2^i$

$= 1\times2^5 + 1\times2^4 + 1\times2^0$
$= 32 + 16 + 1$
$= 49$

The result is -49.

There is no carry and the MSB bit is 1 so the result is negative.

case-3:- To add a "negative number" with a "negative nu

→ Write the 2's complement of both the numbers.
→ Add both the numbers.
→ The carry will be ignored and final result will be -("2's complement of addition result.

Q:- Add -14 to -25

14 :- 00001110 (Binary)

:- 11110001 (1's)

-14 :- 11110010 (2's)

25 :- 00011001 (Binary)

:- 11100110 (1's)

-25 :- 11100111

-25 :- 1 1 1 0 0 1 1 1

-14 :- 1 1 1 1 0 0 1 0

1 1 1 0 1 1 0 0 1

1's :- 0 0 0 1 0 0 1 1 0

2's :- 0 0 0 1 0 0 1 1 1

$= 1\times2^5 + 1\times2^2 + 1\times2^1 + 1\times2^0$

$= 32 + 4 + 2 + 1$

$= 39$

2|14
2|7 0
2|3 1
1 1

2|25
2|12 1
2|6 0
2|3 0
1 1

The result is -39.

Important Points (2's complement addition / Substraction) :-

One number is "positive" and another number is "negative":

- If carry is there, discard it
- If no carry, the result will be "-(2's complement of the result)"

Both numbers are negative -

- Discard carry and result will be "-(2's complement of the result)"

Floating - point Number Representation :-

→ Floating - Point is useful for representing a number in a wide range : very small to very large.

→ It is widely used in the Scientific world.

→ There are many ways to write a number in scientific notation, but there is always a unique normalized representation, with exactly one non-zero digit to the left of the point.

Example :-

$257 = 25.7 \times 10^1 = 2.57 \times 10^2$

$12315 = 1.2315 \times 10^4$

$1001 = 1.001 \times 2^3$

Q:- What's the normalized representation of

$00101101.101 = 1.01101101 \times 2^5$

Q:- What's the normalized representation of

$0.0001101001110 = 1.1010011 10 \times 2^{-4}$

→ We can represent floating-point numbers with three binary fields : i) a sign bit s,
ii) an exponent field E, and
iii) A Significand (mantissa) / fraction field F.

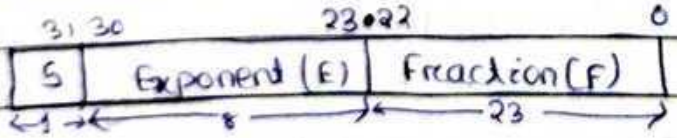
Sign	Exponent	Significand

→ The IEEE 754 (Institute of Electrical and Electronics Engineers) Standard defines how floating-point numbers are stored in binary format, including single-precision (32-bit), double-precision (64-bit) etc.

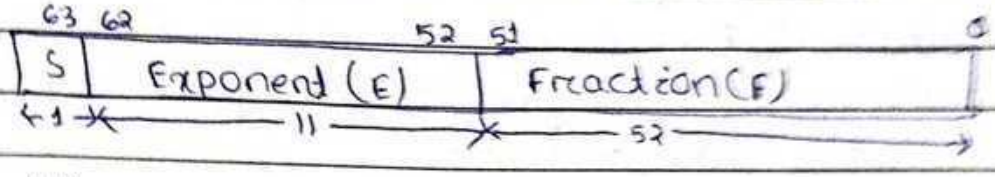
→ The IEEE 754 standard defines several different precisions.

- Single precision (32 bits) $[\overset{S}{1} + \overset{E}{8} + \overset{F}{23}]$
- Double precision (64 bits) $[\overset{S}{1} + \overset{E}{11} + \overset{F}{52}]$
- Extended precision (80 bits) $[\overset{S}{1} + \overset{E}{15} + \overset{F}{64}]$

- Single precision :- numbers include a 1-bit sign, an 8-bit exponent and a 23-bit fraction, for a total of 32 bits.



- Double precision:- numbers have a 1-bit sign, an 11-bit exponent and a 52 bit fraction, for a total of 64 bits.



Sign (S)

- The Sign bit is
  0 for positive numbers and
  1 for negative numbers.

Exponent (E):-

→ The E field represents the exponent as a biased number.

→ It contains the actual exponent +127 for single precision and for double precision is represented as actual exponent +1023.

→ For Single precision, final Exponent (E) = actual exponent + 127 (bias)

→ For double precision, final exponent (E) = actual exponent + 1023 (bias)

★

Single	Double
$2^{e-1}-1$	$2^{e-1}-1$
$= 2^{8-1}-1$	$= 2^{11-1}-1$
$= 2^{7}-1$	$= 2^{10}-1$
$= 128-1$	$= 1024-1$
$= 127$	$= 1023$

Fraction / significand :-

→ The field F contains a binary fraction.

Example:- What is the Single-precision representation of 347.625 ?

→ First convert the number to binary:

$347.625 = 101011011.101_2$

→ Normalize the number by shifting the binary point until there is a single 1 to the left:

$101011011.101 \times 2^0 = 1.01011011101 \times 2^8$

→ The bits to the right of the binary point comprise the fraction field F.

→ The number of times you shifted gives the exponent (E).

→ The field E should contain :

exponent +127 = 8 + 127 = 135 = 10000111.

→ Sign bit : 0 if +ve, 1 if -ve.

0 10000111 01011011101000000000000

Q:- Represent -3.75 in single precision format.

→ -3.75 = -11.11 (Binary)

0.75 x 2 = 1.5 2 | 3
0.5 x 2 = 1.0 1 1

→ Normalize :-

1.111×2^1

Exponent = 1 + 127 = 128 = 10000000

2 | 128
2 | 64 0
2 | 32 0
2 | 16 0
2 | 8 0
2 | 4 0
2 | 2 0
1 0

→ 1 10000000 11100000000000000000000

Q:- Fendout the floating point number from the following representation.

0 10000011 00111000000000000000000

Exponent, E = 10000011 = 131

$N = (-1)^S \times (1+F) \times 2^{E-bias}$

$= (-1)^0 \times (1.00111000000000000000000) \times 2^{131-127}$

$= 1 \times (1.00111000000000000000000) \times 2^4$

$= 1.00111000000000000000000 \times 2^4$

$= 10011.10$

$= 19.5$

30/07/2025

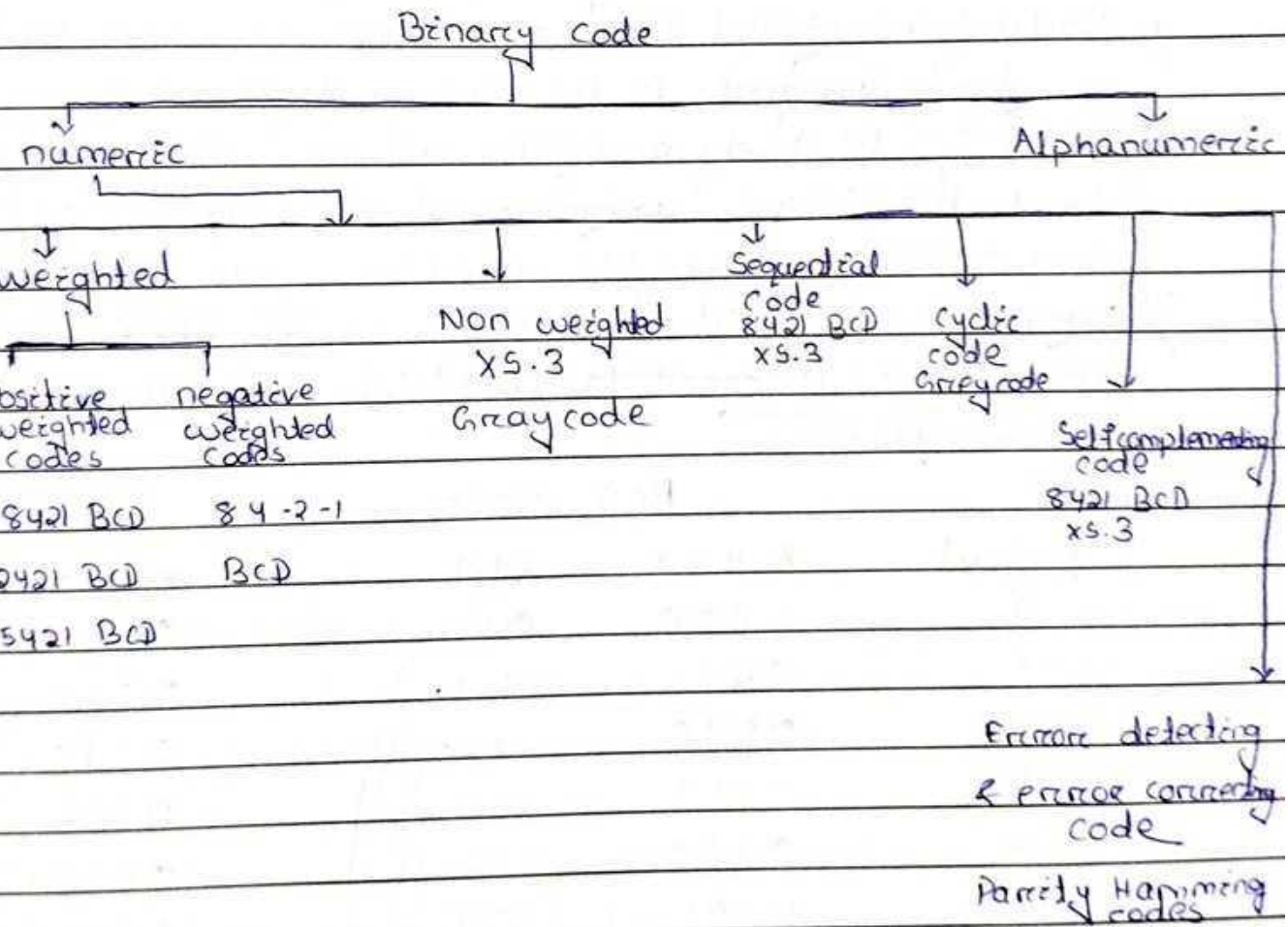
Binary Codes :-

→ In digital electronics, binary codes are the foundation for representing and processing any information.

→ Binary code is based on the binary number system that uses only two digits : 0 and 1.

→ Binary codes are the fundamental language of digital electronics, enabling the representation and processing of all types of information within digital systems.

Classification of Binary Codes :-



1. Numeric Codes :-

→ These are codes which represent numeric information, i.e only numbers as a series of 0s and 1s. It is divided into the following types.

a) Weighted Codes :-

→ In the weighted code, each bit in the number has a value depending upon its position or location within the no. and this value is called the weight.

→ The Decimal value of the no. will be equal to the sum of the product of the bits and their respective weights.

Types of weighted codes :-

It is divided into 2 type i.e

i) Positively - weighted codes are those in which all the weights assigned to the binary digits are positive.

→ In every positively weighted code, the first weight must be 1, the second weight must be either 1 or 2, and sum of all the weights must be equal to or greater than 9.

ii) Negatively - weighted codes are those in which some of the weights assigned to the binary digits are negative.

BCD Codes

Decimal	8 4 2 1	2 4 2 1	8 4 -2 -1
0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 1 1 1
2	0 0 1 0	0 0 1 0	0 1 1 0
3	0 0 1 1	0 0 1 1	0 1 0 1
4	0 1 0 0	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1	1 0 1 1
6	0 1 1 0	1 1 0 0	1 0 1 0
7	0 1 1 1	1 1 0 1	1 0 0 1
8	1 0 0 0	1 1 1 0	1 0 0 0
9	1 0 0 1	1 1 1 1	1 1 1 1

b) Non-Weighted Codes:-

→ These are binary codes where the pos° of each bit does not have a fixed numerical value or weight.

Examples: Gray code and Excess-3 code

i) Gray code:

A unit-distance code where only one bit changes bet° consecutive numbers

ii) Excess-3 Code:-

Excess-3 code is also called as XS-3 code. It is a type of non-weighted code used to express decimal numbers.

Decimal	BCD	Gray code	XS-3 code
0	0000	0000	0011
1	0001	0001	0100
2	0010	0011	0101
3	0011	0010	0110
4	0100	0110	0111
5	0101	0111	1000
6	0110	0101	1001
7	0111	0100	1010
8	1000	1100	1011
9	1001	1101	1100

c) Sequential Codes:-

→ Sequential code is one, in which each succeeding code word is one binary no. greater than its preceding code word.

The 8421 and Excess-3 codes are Sequential

The codes 5211, 2421 and 642-3 are not sequential.

iv) Self-complementing codes:-

→ Self-complementing codes in DF are binary codes where the 9's complement of a decimal no. can be obtained by simply inverting the bits of its corresponding code.

→ 9's complement:

→ In a decimal no. system, the 9's complement of a digit is found by substracting it from 9.

→ A key characteristic is that the code for 9 is the complement of the code for 0, the code for 8 is the complement of the code for 1 and so on

→ The 2421, 5211, 642-3, 84-2-1 and Excess-3 codes are Self complementing codes.

→ The 8421 and 5421 codes are not self-complementing code

v) Error Detecting and Error correcting codes:-

It is used to detect the error and correct the error

Eg:- Shift counter code, 2-out-of-5, 63210 codes are error detecting codes.

Eg:- The Hamming code is a error correcting code.

vi) Cyclic codes:-

→ Cyclic codes are those in which each successive code word differs from the preceding one in only one bit position.

→ They are also called unit distance codes.

→ The Gray code is a cyclic code.

Alphanumeric codes:-

→ These codes in DF are binary codes used to represent both letters and numbers (and sometimes other symbols) in a format that computers can understand and process.

→ They are essential for interfacing input/output devices like keyboards and displays with computers, allowing for the representation of text alongside numerical data.

→ Example :-
ASCII (American
EBCDIC

The ASCII code :-

→ The ASCII is a widely used alphanumeric code.

→ This is basically a 7-bit code. Since the number of different bit patterns that can be created with 7 bits is $2^7 = 128$, the ASCII can be used to encode both lowercase and uppercase characters of the alphabet (52 symbols) and some special symbols as well, in addition to the 10 decimal digits.

→ It is used extensively for printers and terminals that interface with small computer systems.

→ Eg :- EBCDIC
Another character encoding scheme, primarily used by IBM mainframe systems.

Logic levels and pulse waveform

Logic Levels and Pulse Waveforms :

→ Digital systems use the binary number system. Therefore, two-state devices are used to represent the two binary bits 1 and 0 by two different voltage levels, called HIGH and LOW.

→ If the HIGH voltage level is used to represent 1 and the LOW voltage level to represent 0, the system is called the positive logic system.

→ If the HIGH voltage level is used

→ Normally, the binary 0 and 1 are represented by the logic voltage levels 0v and +5V.

→ In a positive logic system, 1 is represented by +5V (HIGH) and 0 is represented by 0v (LOW).

→ In a negative logic system, 0 is represented by +5V (HIGH) and 1 is represented by 0v (LOW).

→ In reality, because of circuit variations, the 0 and 1 would be represented by voltage ranges instead of particular voltage levels.

→ Logic-0 → 0v to 0.8v
Logic-1 → 2v to 5v
0.8 to 2v → indeterminate range.

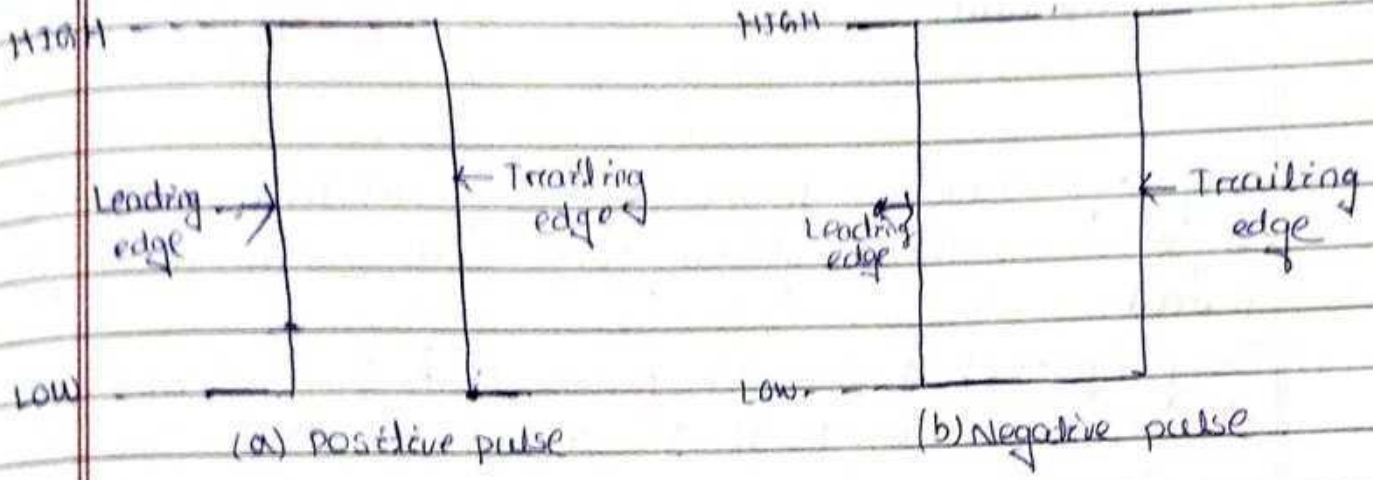
→ If the signal falls betⁿ 0.8v and 2v, the response is not predictable.

Pulse:-

→ A pulse may be a positive pulse or a negative pulse.

→ A single positive pulse is generated when a normally low voltage goes to its HIGH level and then returns to its normal LOW level.

→ A single Negative pulse is generated when a normally HIGH voltage goes to its LOW level and then returns to its normal HIGH level.



(a) Positive pulse (b) Negative pulse

→ A pulse has 2 edges. i) Leading edge
ii) Trailing edge.

→ In the case of a positive pulse, the edge going from low logic level to high logic level is called the leading edge, and the edge going from high logic level to low logic level is called the trailing edge.

→ In the case of a negative pulse, the edge going from high logical level to low logic level is called the leading edge, whereas the edge going from low logic level to high logic level is called the trailing edge.

→ The pulse has 2 different times.
a) Rise time of the pulse
b) Fall time "

→ The rise time is defined as the time taken by the pulse to rise from 10% to 90% of the pulse amplitude.

→ The fall time is defined as the time taken by the pulse to fall from 90% to 10% of the pulse amplitude.

→ The duration of the pulse is usually indicated by pulse width (t_w) which refers to the duration of a digital pulse. Specifically the time bet^n its leading (rising) and trailing (falling) edges.

→ The reciprocal

→ Duty cycle refers to the percentage of time a digital signal is in the "high" or "on" state within a single period.

→ Duty cycle of a periodic pulse waveform is the ratio of the pulse width t_w to the period of the pulse waveform (T).

$$\text{duty cycle} = \frac{t_w}{T} \times 100\%$$

Q:- A periodic digital waveform has a pulse width of 25 μsec and a time period of 150 μsec. Determine the frequency and dutycycle.

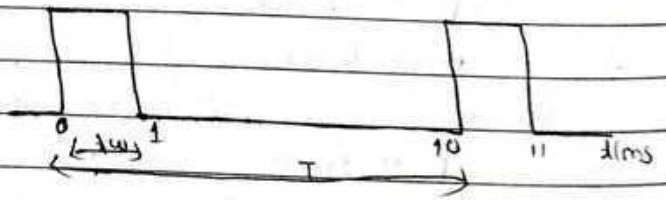
$$\text{frequency } f = \frac{1}{T} = \frac{1}{150 \times 10^{-6}} = 6666.667 \text{ Hz}$$

$$= 6.67 \text{ KHz}$$

$$\text{Duty cycle} = \frac{25 \times 10^{-6}}{150 \times 10^{-6}} \times 100$$

$$= 16.7\%$$

Q:- A portion of a periodic digital waveform is shown in Fig. The measurements are in milliseconds. Determine
a) period
b) frequency and
c) duty cycle.



Sol" Here $t_w = 1$ ms

$T = 10$ ms

a) Period = 10 ms

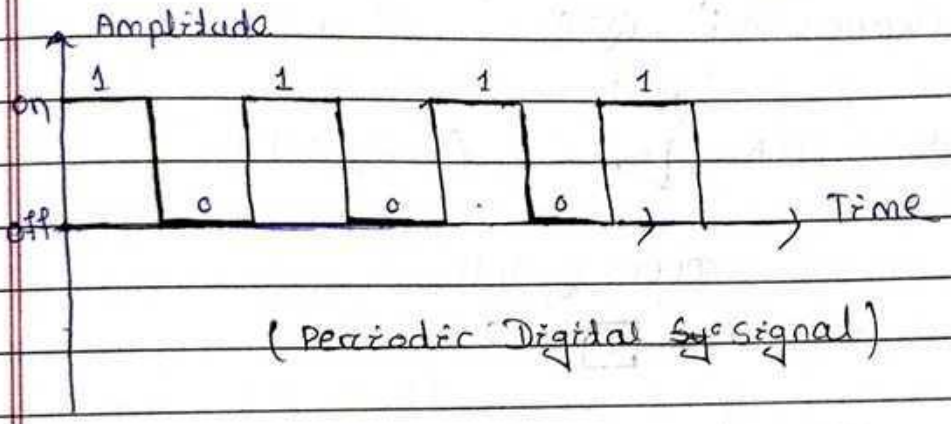
b) frequency $f = \frac{1}{T} = \frac{1}{10 \text{ ms}} = 100$ Hz

c) duty cycle $= \frac{t_w}{T} \times 100 = \frac{1}{10} \times 100 = 10\%$

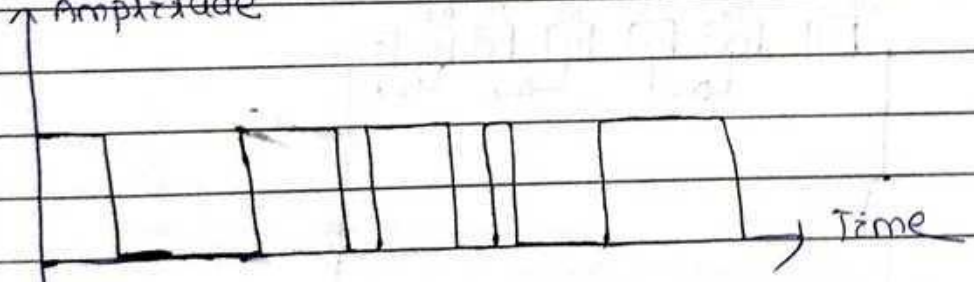
→ In digital systems the waveforms are composed of a series of pulses and can be classified as periodic waveforms and non-periodic waveforms.

→ A periodic waveform is one which repeats itself at regular intervals of time called the period, T

→ A non-periodic waveform, of course, does not repeat itself at regular intervals and may be composed of pulses of diff. widths and/or differing time intervals bet^n the pulses.



(Periodic Digital Sig signal)



(Aperiodic Digital Signal)

Logic gates :-

Buffer :-

→ A buffer is a basic logic gate that passes its input, unchanged to its output.

→ The main purpose of a buffer is to regenerate the input, usually using a strong high and a strong low.

→ A buffer has one input and one output; it's output always equals its input.

→ Its symbol is simply a triangle.

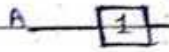
→ Boolean Expression : Q=A

Truth table :-

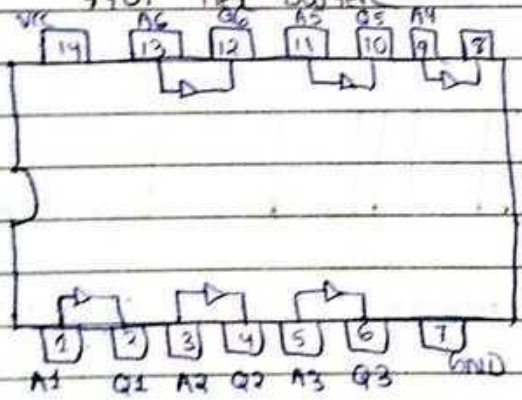
Input	output
A	Q
0	0
1	1

Symbol

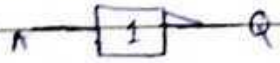
IEEE Symbol



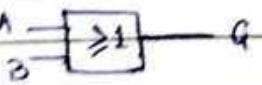
7407 Hex Buffers



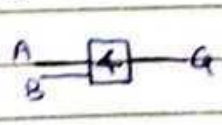
NOT gate :-

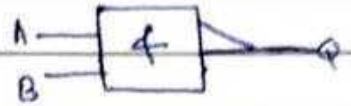


OR gate :-

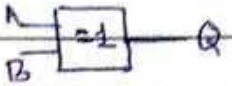


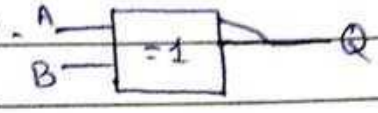
AND gate :-



NAND gate :- 

NOR gate :- 

Ex-OR gate :- 

Ex-NOR gate :- 

3- input Ex-OR gate :-

The logic fun^n implemented by a 2- input Ex-OR is given as either : " A OR B but NOT both" will give an output of Q. In general, an Ex-OR gate will give an output value of logic "1" only when there are an ODD no. of 1's on the inputs to the gate. Then an Ex-OR fun^n with more than two inputs is called an "odd fun^n".

Truth table

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

Axioms and Laws of boolean algebra :-

Boolean algebra :-

→ Boolean algebra is used to analyze and simplify the digital (logic) circuit.

→ It uses only the binary number 0 & 1.

→ It is also called as Binary algebra or logical Algebra.

→ Algebra consists of symbolic representation of a statement (generally mathematical statements)

→ Similarly, there are expressions, equations and functions in boolean algebra as well.

Axioms or Postulates of Boolean algebra :-

→ Axioms or postulates of boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorums.

→ Actually axioms are nothing more than the definitions of three basic logic operations that we have already discussed: AND, OR, NOT.

Basic Laws of Boolean Algebra :-

In all the cases given below A can be either be 0 & 1.

1. $A \cdot 0 = 0$
2. $A \cdot 1 = A$
3. $A \cdot A = A$
4. $A \cdot \bar{A} = 0$
5. $A + 0 = A$
6. $A + 1 = 1$
7. $A + \bar{A} = 1$
8. $A + A = A$

Let :- 4. $A = 1$, Let $A = 0$.

4. $A \cdot \bar{A} = 0 \neq 0 \cdot 1 = 0$.

$\Rightarrow 1 \cdot 0 = 0$ (proved)

Let, $A = 1$, $A \neq 0$

8. $1 + 1 = 1$, $A + A = A$

$= 0 + 0 = 0$ (proved)

Some basic Boolean algebra laws that are used to simplify Boolean expressions are :

1. Idempotent law :
   $A * A = A$
   $A + A = A$
2. Associative law
   $(A * B) * C = A * (B * C)$
   $(A + B) + C = A + (B + C)$
3. Commutative law
   $A * B = B * A$
   $A + B = B + A$
4. Distributive law
   $A * (B + C) = A * B + A * C$
   $A + (B * C) = (A + B) * (A + C)$
5. Identity law
   $A * 0 = 0$, $A * 1 = A$
   $A + 1 = 1$, $A + 0 = A$
6. Inversion law :
   $\bar{\bar{A}} = A$
7. Complement law
   $A * \bar{A} = 0$
   $A + \bar{A} = 1$
8. Absorption law :-
   Absorption law, $A(A+B) = A$, $A + AB = A$.

Proof

1. $A + A = A$

L.H.S

$A + A$

$\therefore (A + A) \cdot 1$

$\therefore (A + A) \cdot (A + \bar{A})$ $\quad (A + (B * C) = (A + B) * (A + C))$

$\therefore A + A \cdot A'$ $\quad (\because A \cdot \bar{A} = 0) \quad = A = \underline{R.H.S}$

2. $A \cdot A = A$

L.H.S

$A \cdot A$

$= A \cdot A + 0$

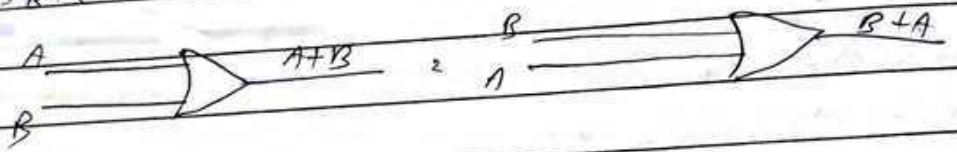
$= A \cdot A + A \cdot A'$ [∵ distributive law]

$= A(A + A')$ (∵ $A + A' = 1$)

$= A$ = R.H.S proved

Commutative law

(a) $A + B = B + A$

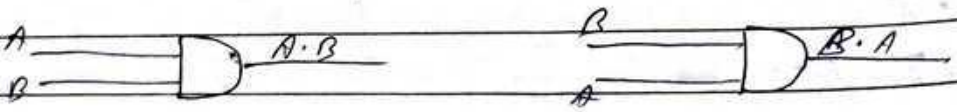


A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

=

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

(b) $A \cdot B = B \cdot A$



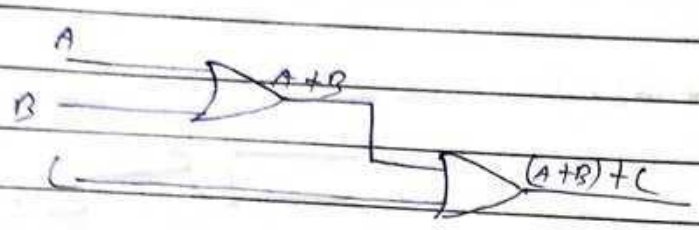
A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

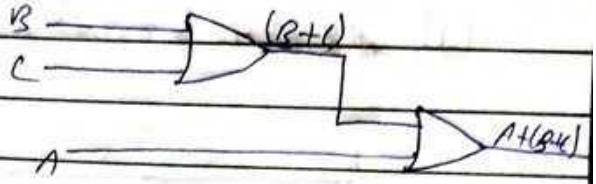
=

B	A	B·A
0	0	0
0	1	0
1	0	0
1	1	1

Associative law :-

(a) $(A + B) + C = A + (B + C)$



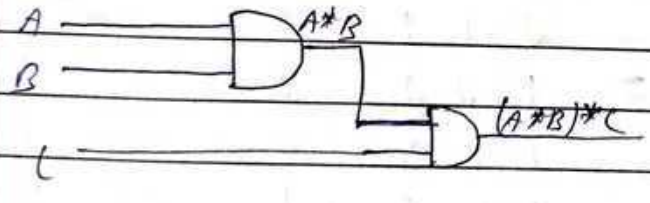


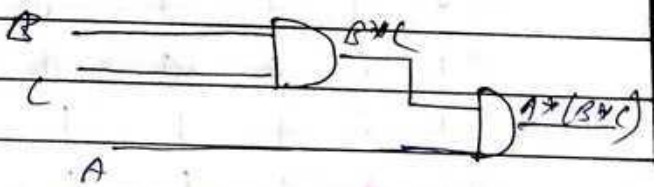
A	B	C	(A+B) ①	① + C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

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

(b) $(A * B) * C = A * (B * C)$





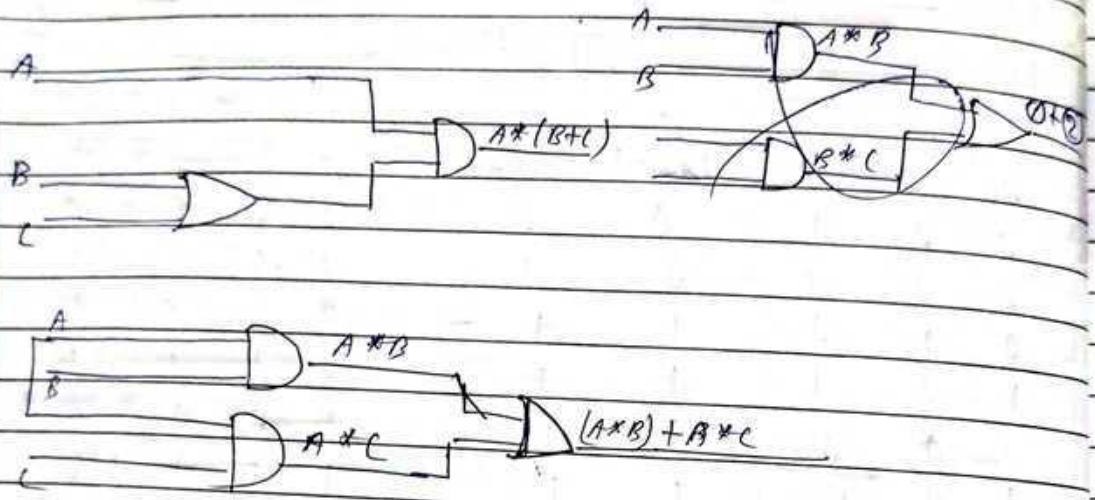
A	B	C	A*B ①	① * C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

=

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

Distributive law :-

(a) $A * (B+C) = (A*B) + (A*C)$

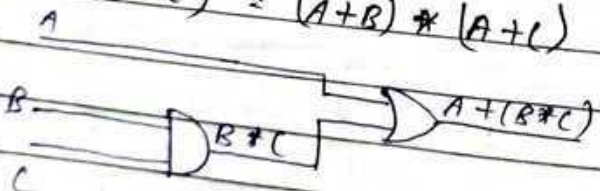


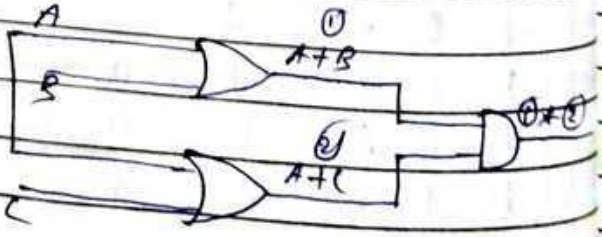
A	B	C	(B+C)	A*①
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

A	B	C	A*B	A*C	①+②
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(b) $A + (B*C) = (A+B) * (A+C)$

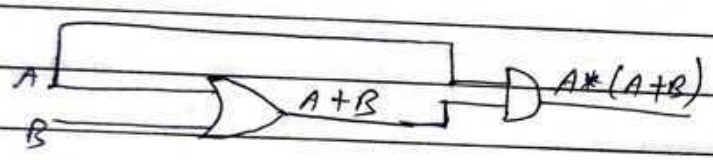




A	B	C	① R*C	A+①	=	A	B	C	① A+B	② A+C	①*②
0	0	0	0	0		0	0	0	0	0	0
0	0	1	0	0		0	0	1	0	1	0
0	1	0	0	0		0	1	0	1	0	0
0	1	1	1	1		0	1	1	1	1	1
1	0	0	0	1		1	0	0	1	1	1
1	0	1	0	1		1	0	1	1	1	1
1	1	0	0	1		1	1	0	1	1	1
1	1	1	1	1		1	1	1	1	1	1

Absorption law:-

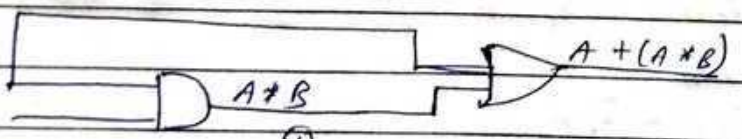
① $A(A+B) = A$



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

∵ $A(A+B) = A$ proved

② $A + AB = A$



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

∵ $A + AB = A$ proved

Consensus Theorum :-

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

L.H.S

$AB + \bar{A}C + BC$

$= AB + \bar{A}C + BC(A + \bar{A})$

$= AB + \bar{A}C + BCA + BC\bar{A}$

$= AB(1 + C) + \bar{A}C(1 + B)$

$= AB \cdot 1 + \bar{A}C \cdot 1$

$= AB + \bar{A}C$ = R.H.S (proved)

Transposition theorum :-

Proof :- $AB + \bar{A}C = (A + C)(\bar{A} + B)$

L.H.S

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

$A\bar{A} + AB + C\bar{A} + BC$

$= 0 + \bar{A}C + AB + BC$

$= \bar{A}C + AB + BC(A + \bar{A})$

$= AB + ABC + \bar{A}C + \bar{A}BC \Rightarrow AB(1 + C) + \bar{A}C(1 + B)$

$= AB + \bar{A}C$ = R.H.S proved

2

De-Morgan's Theorum :-

→ De-Morgan's law states that :-

→ 1st law :-

The complement of a sum equals to the product of the complements.

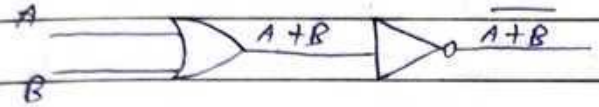
$\overline{A + B} = \bar{A} \cdot \bar{B}$

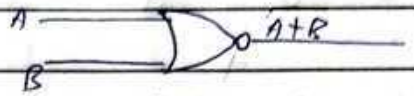
→ 2nd law :-

The complement of a product is equal to the sum of the complement.

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

① L·H·S

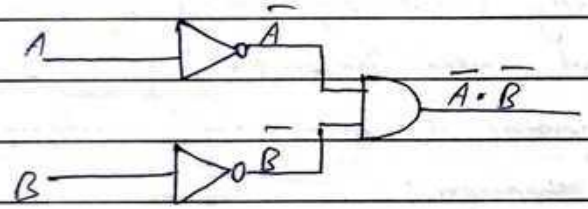


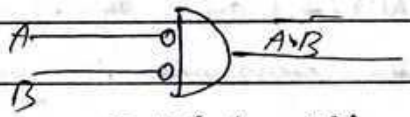


NOR gate.

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

R·H·S





Bubbled AND gate.

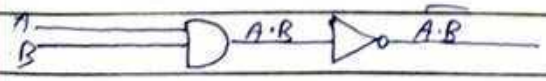
A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

L·H·S = R·H·S proved ∵ [first law]

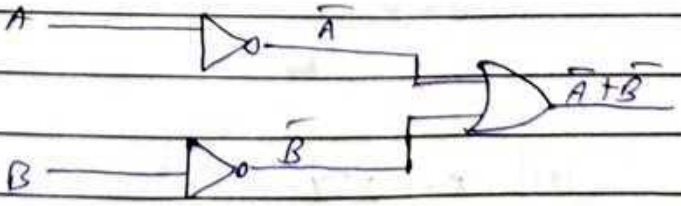
② $\overline{A \cdot B} = \overline{A} + \overline{B}$

2·H·S

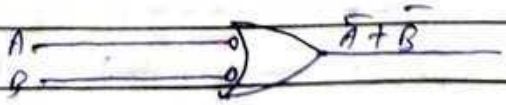
Bubbled NAND gate.



R·H·S



Bubbled OR gate →



A	B	$\bar{A} \cdot B$	$A \cdot \bar{B}$		A	B	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$
0	0	0	1		0	0	1	1	1
0	1	0	1	=	0	1	1	0	1
1	0	0	1		1	0	0	1	1
1	1	1	0		1	1	1	0	0

L.H.S = R.H.S proved

Duality :-

→ In the Boolean algebra, the principle of duality states, that if you have a valid boolean expression, you can create a new, equally valid expression by interchanging AND (·) and OR (+), and also interchanging 0 & 1.

→ This principle is fundamental to simplifying Boolean expressions and proving theorems.

13/08/25

Reducing boolean expressions :-

Simplify the following boolean function to the minimum number of literals.

(1) $x(x'+y)$

(2) $(x + x'y)$

(3) $(x+y)(x+y')$

(4) $xy + x'z + yz$

(1) $x(x'+y)$

$= x \cdot x' + x \cdot y$

$= 0 + xy$

$= xy$ Ans

(2) $x + x'y$

$(x + x')(x + y)$

$= 1 \cdot (x+y) = (x+y)$ Ans

2) $x + x'y$

$= (x + x') \cdot (x + y)$

$= 1 \cdot (x + y)$

$= (x + y)$

(3) $(x + y)(x + y')$

$= x + (y \cdot y')$

$= x$

$x \cdot x + xy' + xy + y \cdot y'$ (0)

$= x(x + y' + y) = x(x + 1) = x \cdot x + x$

$= x + x = x$

(4) $xy + x'z + yz$

$= xy + x'z + yz \cdot (x + x')$

$= xy + x'z + yzx + yzx'$

$= x(y + yz) + x'(z + yz)$

$= x[(y + x)(y + z)] + x'[(z + y) \cdot (z + z)]$

$= x[y(y + z)] + x'[(z + y) \cdot z]$

$= x[yy + yz] + x'[z \cdot z + yz] = x[yz] + x'[yz]$

4) $xy + x'z + yz$

$= xy + x'z + yz \cdot 1$

$= xy + x'z + yz(x + x')$

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

$= xy(1 + z) + x'z(1 + y)$

$= xy \cdot 1 + x'z \cdot 1$

$= xy + x'z$

Q: Reduce the boolean expression :-

$\overline{(a + \bar{b} + c) + (b + \bar{c})}$

$= \overline{(a + \bar{b} + c)} \cdot \overline{(b + \bar{c})}$

$= (a\bar{b}c) \cdot (b\bar{c}) = (\bar{a}b\bar{c}) \cdot (\bar{b}c) = \bar{a} \cdot 0 = 0.$

6

Q: Reduce the expression :-

$(B + BC)(B + \bar{B}C)(B + D)$

$= (BB + B\bar{B}C + BBC + \bar{B}\bar{B}CC)(B + D)$

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

$= (BB + BBC)(\bar{B} + D)$

$= (B + BC)(B + D)$

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

$= B(1)(1) = B.$

$B + (BC \cdot \bar{B}C \cdot D)$

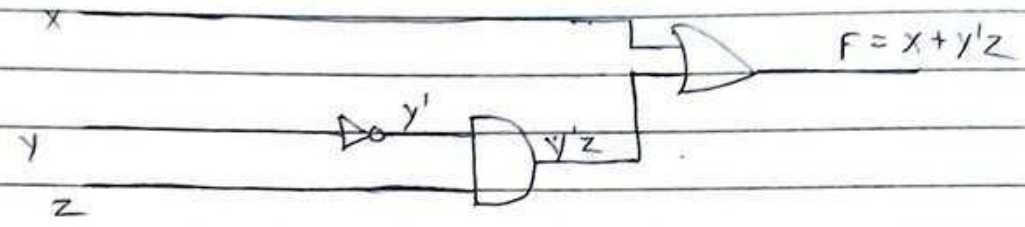
$= B + (0 \times C \times D)$

$= B + (0 \times D)$

$= B$

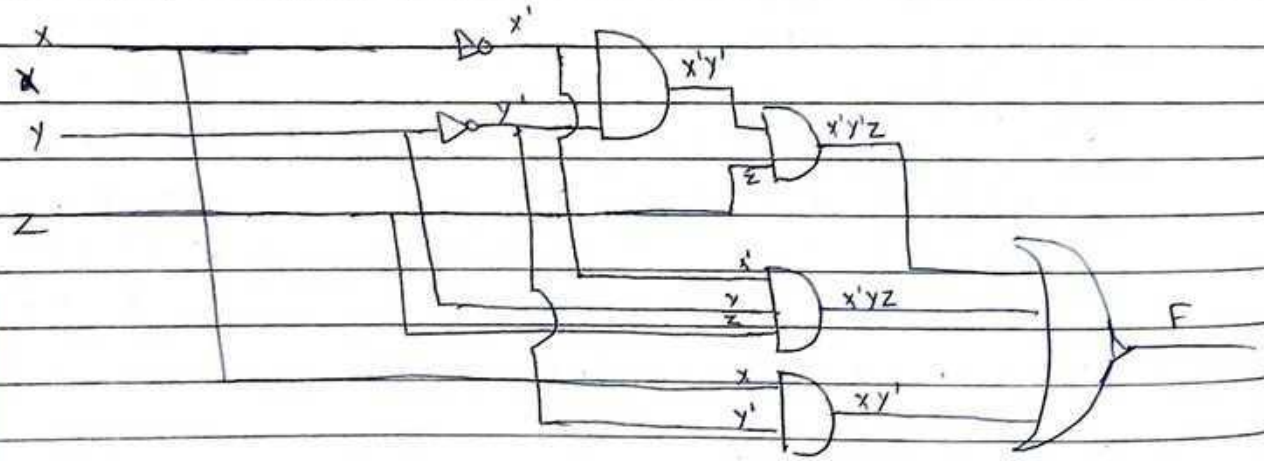
Draw the logic diagram to implement the following Boolean expression.

$F = x + y'z$



2) Draw the logic diagram to implement the following Boolean expression.

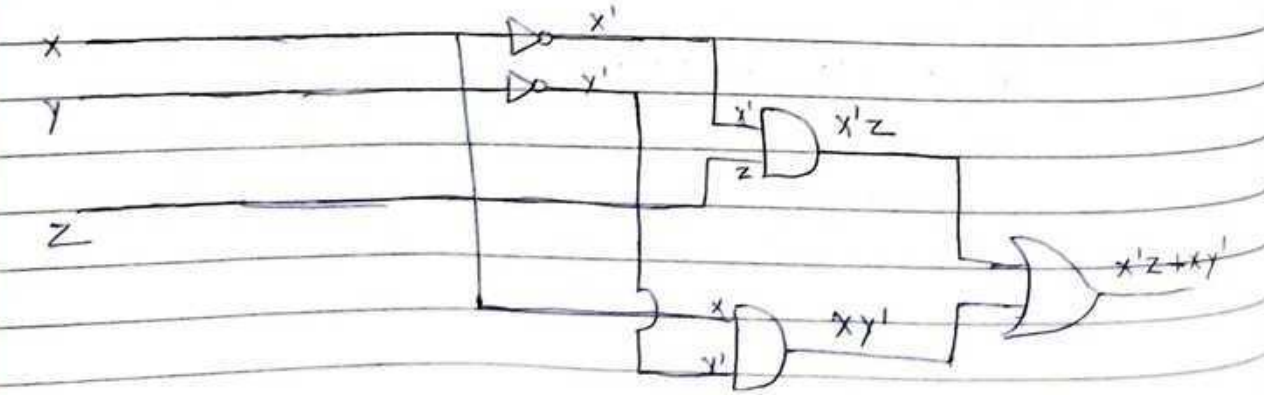
$F = x'y'z + x'yz + xy'$



$F = x'y'z + x'yz + xy'$ $\qquad x'z + xy'$

$= x'z(y' + y) + xy'$

$= x'z + xy'$



18/8/25

Boolean Function:

→ A Boolean function refers to a function having n number of variables or inputs, so it has 2^n number of possible combination of the given variables or input.

→ For a given value of the binary variables on inputs, the function can be equal to either 0 or 1 as its output

→ Example

$F1 = x + y'z$

$F2 = x'y'z + x'yz + xy'$

→ A Boolean function can be represented in a truth table

→ The number of rows in the truth table is 2^n, where n is the number of variables or inputs in the function.

→ The binary combinations for the truth table are obtained from the binary numbers by counting from 0 to 2^n-1

$F_1 = x + y'z$

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$F_2 = x'y'z + x'yz + xy'$

x	y	z	f_2
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Minterms and Maxterms :-

-> A Boolean function can be expressed algebraically from a given truth table by forming a :

-> Minterm for each combination of the variables that produces a 1 in the function and then taking the 'OR' of all those terms.

-> Maxterm for each combination of the variables that produces a 0 in the function and then taking the 'AND of those terms.

-> A symbol for each minterm is of the form m_j, where the subscript j denotes the equivalent of the binary number of the minterm designated.

-> for maxterm M_j.

-> Truth tales.

			Minterm		Maxterm	
			Product terms		Sum terms	
x	y	z				
0	0	0	$m_0 = \bar{x}\cdot\bar{y}\cdot\bar{z} = \min(\bar{x},\bar{y},\bar{z})$		$M_0 = x+y+z = \max(x,y,z)$	
0	0	1	$m_1 = \bar{x}\cdot\bar{y}\cdot z = \min(\bar{x},\bar{y},z)$		$M_1 = x+y+\bar{z} = \max(x,y,\bar{z})$	
0	1	0	$m_2 = \bar{x}\cdot y\cdot\bar{z} = \min(\bar{x},y,\bar{z})$		$M_2 = x+\bar{y}+z = \max(x,\bar{y},z)$	
0	1	1	$m_3 = \bar{x}\cdot y\cdot z = \min(\bar{x},y,z)$		$M_3 = x+\bar{y}+\bar{z} = \max(x,\bar{y},\bar{z})$	
1	0	0	$m_4 = x\cdot\bar{y}\cdot\bar{z} = \min(x,\bar{y},\bar{z})$		$M_4 = \bar{x}+y+z = \max(\bar{x},y,z)$	
1	0	1	$m_5 = x\cdot\bar{y}\cdot z = \min(x,\bar{y},z)$		$M_5 = \bar{x}+y+\bar{z} = \max(\bar{x},y,\bar{z})$	
1	1	0	$m_6 = x\cdot y\cdot\bar{z} = \min(x,y,\bar{z})$		$M_6 = \bar{x}+\bar{y}+z = \max(\bar{x},\bar{y},z)$	
1	1	1	$m_7 = x\cdot y\cdot z = \min(x,y,z)$		$M_7 = \bar{x}+\bar{y}+\bar{z} = \max(\bar{x},\bar{y},\bar{z})$	

→ In minterm $\bar{x} = 0$, $x = 1$

→ In maxterm $\bar{x} = 1$, $x = 0$

→ From the above table it is clear that minterm is expressed as product format and maxterm is expressed as sum format.

Ex-1 Express the truth table in minterm and maxterm.

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A. Minterm (1) (SOP)

$F_1 = x'y'z + xy'z' + x'yz$

$= m_0 + m_4 + m_7$

$= \Sigma m(1, 4, 7)$

Maxterm (0) (POS)

~~$F_1 = (x+y+z')$~~

$F_1 = (x+y+z) \cdot (x+y'+z) \cdot (x+y'+z') \cdot (\bar{x}+y+\bar{z}) \cdot (\bar{x}+\bar{y}+z)$

$= M_0 + M_2 + M_3 + M_5 + M_6$

$= \pi M(0, 2, 3, 5, 6)$

Ex-2

x	y	z	f_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A. ~~min~~ max ~~term~~ term (0) ~~(SOP)~~ (POS)

$f = \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z}$

$= M_0 + M_1 + M_2 + m_4$

$= \sum m(0, 1, 2, 4)$

~~max~~ term (0)

min term (0) (SOP)

$f_1 = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

$= m_3 + m_5 + m_6 + m_7$

$= \sum m(3, 5, 6, 7)$

max term (1) (POS)

$f_1 = (x+y+z) \cdot (x+y+\bar{z}) \cdot (x+\bar{y}+z) \cdot (\bar{x}+y+z)$

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

$= \pi M(0, 1, 2, 4)$

Canonical Forms

-> Boolean functions expressed as a sum of minterms or product of maxterms are said to be canonical form.

-> Each output variable can be expressed in following two ways.

-> Canonical SOP form:- -> Canonical sop form means canonical sum of product form.

-> In this form each product term contains all the variables.

-> It is also called as sum of min terms form.

-> Canonical POS form:- -> Canonical pos form means canonical Product of sum form.

-> In this form each sum term contains all the variables

-> It is also called as product of max term form.

Sum of minterms:-

-> The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.

-> If the function is not in this form, it can be made so by first expanding the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variables.

-> If it misses one or more variables, It is ANDed with an expression such as $x + x'$, where x is one of the missing variable

Ex- 1

Express

A- The fo

A = A (B

B'C = (

Combining

F = A +

= AB

= A

= A

F(A, B, C

Product

-> The maxte which

-> To expre must fins done by

-> Then any $X X'$.

Ex-1

Express the Boolean function $F = A + B'C$ as a sum of minterms

A. The function has three variable A, B, C

$$A = A(B+B') = AB + AB' = AB(C+C') + AB'(C+C')$$
$$= ABC + ABC' + AB'C + AB'C'$$

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

Combining all terms we have

$$F = A + B'C$$
$$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$$
$$= ABC + ABC' + AB'C + AB'C' + A'B'C$$
$$= m_7 + m_6 + m_5 + m_4 + m_1$$

$$F(A,B,C) = \sum m(1,4,5,6,7)$$

Product of Maxterms:

→ The maxterms whose product defines the Boolean function are those which give the 0's of the function in a truth table.

→ To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms. This may be done by using the distributive law, $x + yz = (x+y)(x+z)$.

→ Then any missing variable x in each OR term is ORed with xx'.

Example :-2 ✎

Express the Boolean function $F = xy + x'z$ as a product of maxterms.

A. First convert into OR terms.

$F = xy + x'z$

$= (xy + x')(xy + z)$

$= (x + x')(y + x')(x + z)(y + z)$

$= 1 \cdot (y + x')(x + z)(y + z)$

$= (y + x')(x + z)(y + z)$

The function has three variables x, y, z.

$y + x' = x' + y + zz' = (x' + y + z)(x + y + z')$

$x + z = x + z + yy' = (x + z + y)(x + z + y')$

$y + z = y + z + xx' = (x + y + z)(x' + y + z)$

Combining all the ten.

$F = (x' + y + z)(x + y + z')(x + y + z)(x + y' + z)(x + y + z)(x' + y + z)$

$= (x' + y + z)(x + y + z')(x + y + z)(x + y' + z)$

~~$= M_4 \cdot M_1 \cdot M_7 \cdot M_2$~~

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

$F(x, y, z) = \Pi M(0, 2, 4, 5)$.

2018/2/5

Conversion between Canonical forms:-

→ The complement of a function expressed as the sum of minterms is equal to the sum of minterms missing from the original function.

→ This is because the original function is expressed by those minterms which make the function equal to 1, whereas the its complement is a 1 for those minterms for which the function is 0.

Example:

Consider the function

$$F(A,B,C) = \Sigma m(1,4,5,6,7)$$

The function has a complement that can be

$$F'(A,B,C) = \Sigma m(0,2,3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form.

$$F = (m_0 + m_2 + m_3)'$$

$$= m_0' \cdot m_2' \cdot m_3'$$

$$= M_0 \cdot M_2 \cdot M_3$$

$$= \Pi M(0,2,3)$$

$$F(A,B,C) = \Sigma m(0,1,4,5,6,7) = \Pi M(0,2,3).$$

Relationship between min and maxterm

$$\boxed{\overline{m_j} = M_j} \quad \text{or} \quad \boxed{\overline{M_j} = m_j}$$

Ex- convert each of the following to the other canonical form

1) $F(x,y,z) = \Sigma(1,3,5)$

A $F'(x,y,z) = \Sigma(0,2,4,6,7) =$

$F(x,y,z) = \Pi M(0,2,4,6,7)$

2) $F(x,y,z) = \Pi(0,3,6,7)$

A- $F(x,y,z) = \Sigma m(1,2,4,5)$

3) $F(A,B,C,D) = \Sigma(0,2,6,11,13,14)$

A- $F(A,B,C,D) = \Pi M(1,3,4,5,7,8,9,10,12,15)$

4) $f(A,B,C,D) = \Pi(0,1,3,4,6,8,11,12)$.

A- $F(A,B,C,D) = \Sigma m(2,5,7,9,10,13,14,15)$

Ex- Express the complement of the following function in sum-of-minterms form:

1) $F(x,y,z) = \Sigma(0,3,6,7)$

A- $F'(x,y,z) = \Sigma m(1,2,4,5)$

2) $F(A,B,C,D) = \Sigma(2,4,7,10,12,14)$

A- $F'(A,B,C,D) = \Sigma m(0,1,3,5,6,8,9,11,13,15)$

3) $F(A,B,C) = \Pi(3,5,7)$

A- ~~$F(A,B,C) = \Pi M(0,1,2,4,6)'$~~

$\Rightarrow F'(A,B,C) = \Sigma m(3,5,7)$

4) $F(w,x,y,z) = \Pi(0,1,3,4,6,8,11,12)$

A- $F'(w,x,y,z) = \Sigma m(0,1,3,4,6,8,11,12)$.

Standard forms

→ In canonical form, each term (minterm or maxterm) includes all the variable of the function.

→ But, in the standard form do not have this restriction.

→ Standard form is the another way to express Boolean functions.

→ It is the simplified versions of the canonical forms.

→ In this configuration, the terms that form the function may contain one, two or any number of literals or variable.

→ There are two types of standard forms:

the sum of product (SOP)

and the product of sums (POS)

→ The sum of products (SOP) is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms.

→ An example of a function expressed as a sum of products is $F_1 = y' + xy + x'yz'$

→ The expression has three product term, with one, two, and three variables, Their sum is, in effect, an OR operation.

→ The product of sums (POS) is a Boolean expression containing OR terms, called sum terms. Each term may have any number of variables. The product denotes the ANDing of these terms.

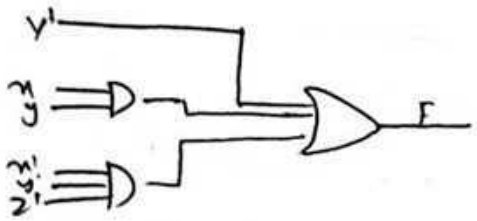
→ An example of a function expressed as a product of sums is $F_2 = x(y'+z)(x'+y+z')$

→ The expression has three terms, with one, two, and three literals. Their Product is an AND operation.

17/09/2025

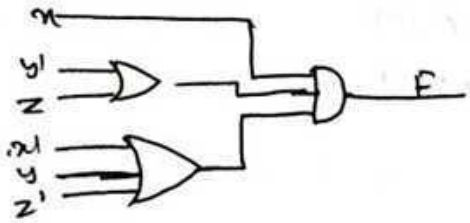
Sum of products

$F = y' + xy + x'yz'$



Product of sums

$F = x(y'+z)(x'+y+z')$



(1) Convert each of the expression in SOP and POS.

(a) $(AB+C)(B+C'D)$

SOP

$(AB+C)(B+C'D)$

$= ABB + ABC'D + CB + CC'D$

$= AB + ABC'D + CB$

$= AB(1 + C'D) + CB = AB + BC$

POS

$(AB+C)(B+C'D)$

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

$= (A+C)\,B + (C+C'+D) = (A+C)\,B + (1)\,(B+D)$

$= (A+C)\,B + (1+D) = (A+C)\cdot B\,(B+D)$

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

$= A+C$

② $x' + x(x+y')(y+z')$

SOP

$x' + (xx + xy')(y+z')$

$= x' + (x + xy')(y+z')$

$= x' + xy + xy'y + xz' + xy'z'$

$= x' + xy + xz' + xy'$

$= x' + xy + xy' + xz'$

$= x' + x(y+y') + xz'$

$= x' + x + xz'$

$= x' + x(1+z')$

$= x' + x$

$= 1$

POS

$x' + x(x+y')(y+z')$

$= (x'+x)(x'+x+y')(x'+y+z')$

$= 1 \cdot (x+y')(x'+y+z')$

$= 1 \cdot 1 \cdot (x'+y+z')$

$= (x'+y+z')$

(∵ $A + (BCD) = (A+B)(A+C)(A+D)$

① Express each function in sum-of-minterm and product of maxterm form.

① $(xy+z)(y+xz)$

$= xyy + xyxz + zy + xzz$

$= xy + xyz + yz + xz$

$= xy(z+z') + xyz + (x+x')yz + xz(y+y')$

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

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

$= m_7 + m_6 + m_3 + m_5$

$= \Sigma m(3,5,6,7) = \pi M(0,1,2,4)$.

② $(A'+B)(B'+C)$ $\Sigma(0,1,3,7)$

A. $A'B' + A'C + BB' + BC$

$= A'B' + A'C + BC$

$= A'B'(C+C') + A'C(B+B') + BC(A+A')$

$= A'B'C + A'B'C' + A'BC + A'B'C + ABC + A'BC$

$= A'B'C + A'B'C' + A'BC + ABC$

$= m_1 + m_0 + m_3 + m_7$

$= \Sigma m(0,1,3,7)$

$= \pi M(2,4,5,6)$.

29/8/25 UNIT-2

③ NAND and NOR Implementation:-

Universal gates:-

→ A universal gate is gate that can implement any Boolean function without the need to use any other gate type.

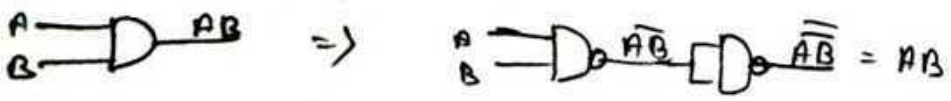
→ The NAND and NOR gates are universal gate.

→ The NAND and NOR gates are said to be universal gates because any logic circuit can be implemented with it.

Logic gates using only NAND gates:-

NOT gate

AND gate:-



29/08/2025

Unit-2

NAND and NOR implementation :-

: Universal Gates :

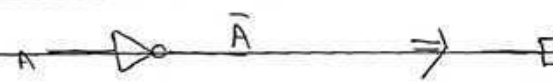
→ A universal gate is a gate that can implement any Boolean fun^n without the need to use any other gate type.

→ The NAND and NOR gates are universal gates.

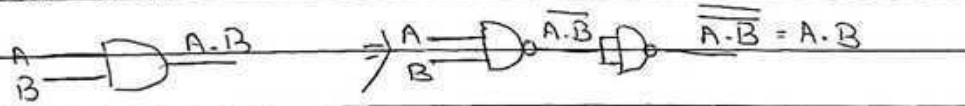
→ The NAND and NOR gates are said to be universal gates because any logic circuit can be implemented with it.

Logic Gates using only NAND gates :-

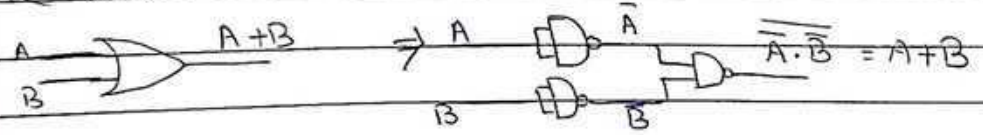
- NOT Gate :-



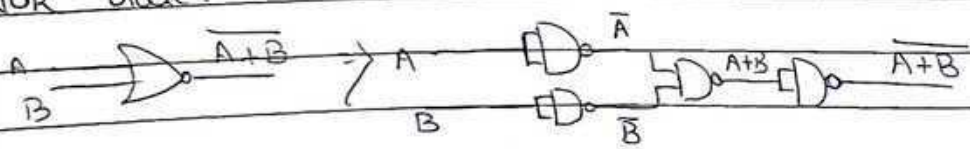
- AND Gate :-



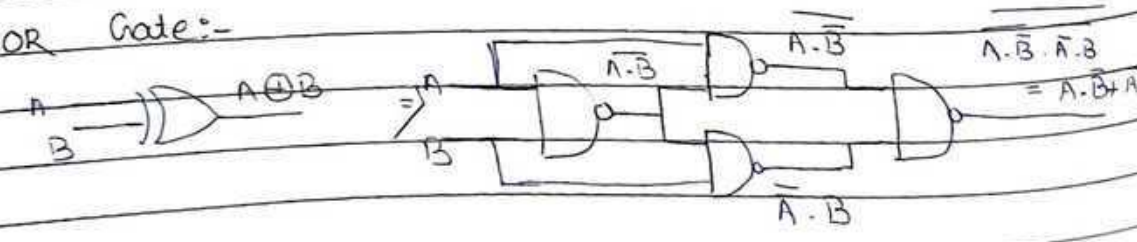
- OR Gate :-



- NOR Gate :-



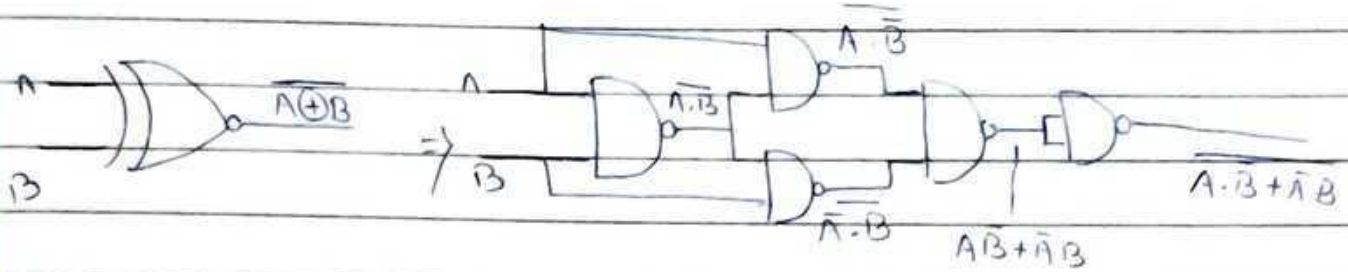
- Ex-OR Gate :-



$\overline{A(\overline{A.\bar{B}})} : \overline{A(\bar{A}+\bar{B})} = \overline{A.\bar{A} + A\bar{B}} = \overline{A\bar{B}} = \bar{A}B$

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

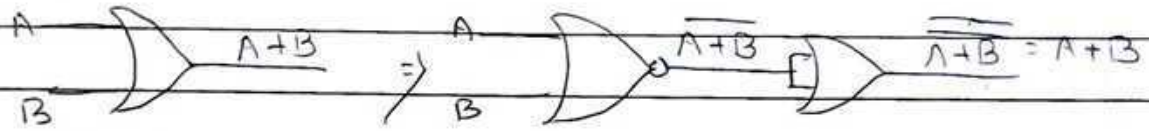
Ex-NOR Gate:-



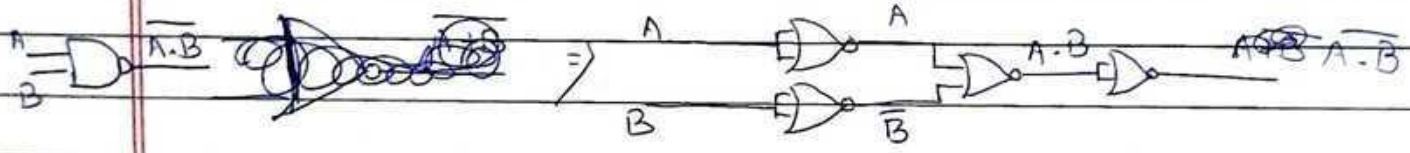
Logic Gates using only NOR Gates :-

- NOT Gate :-

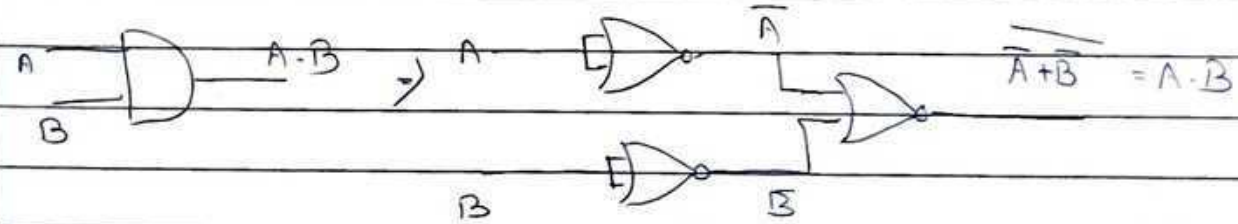
- OR gate :-



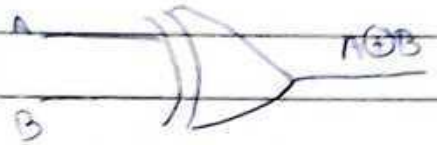
- NAND ~~NOR~~ Gate :-



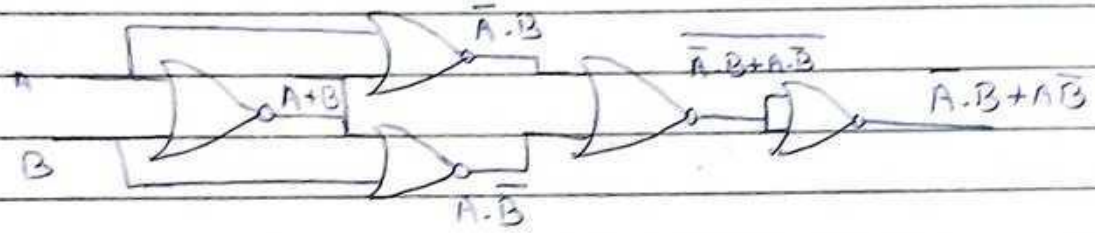
AND Gate :-



Ex-OR Gate:-



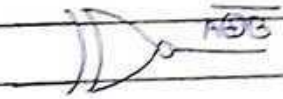
=>



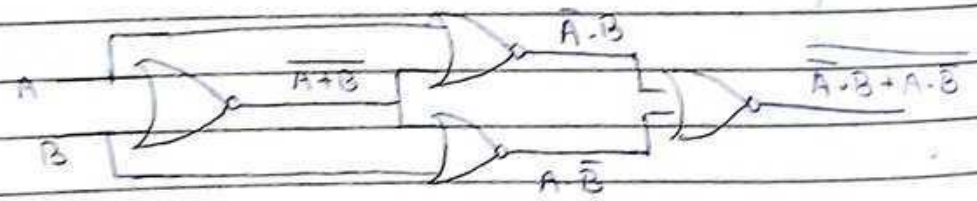
$A+(\overline{A+B}) = \overline{A(\bar{A}.\bar{B})}$

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

Ex-NOR Gate:-



=>

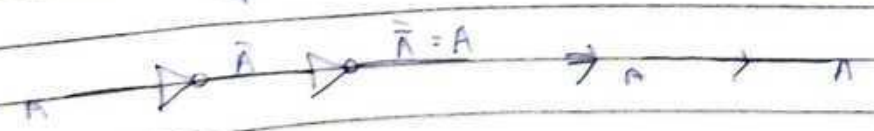


$\overline{A+(\overline{A+B})} = \overline{A+(\bar{A}.\bar{B})} = \overline{A+\bar{A}} \cdot \overline{A+\bar{B}} = \overline{A\bar{B}} = \bar{A}.\bar{B}$

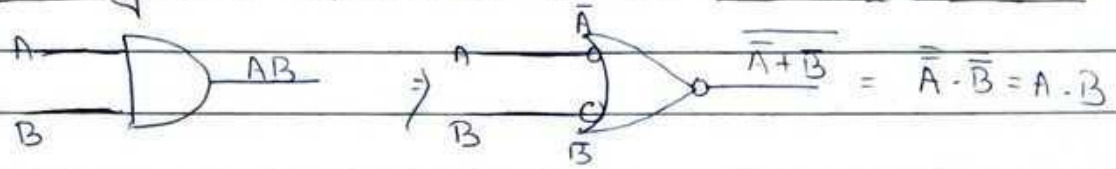
Equivalent Gates:-

=> Note that a bubble denotes complementation / inverter and two bubbles along the same line represent double complementation, so both can be removed.

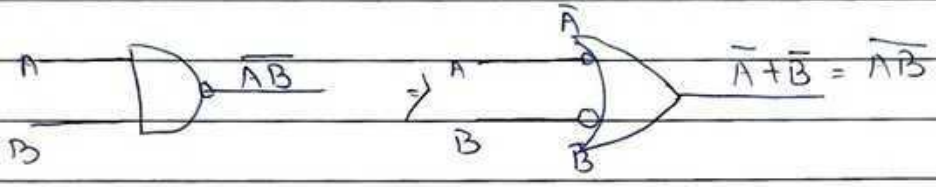
=> Two NOT gate in series are same as a buffer because they cancel each other as $\bar{\bar{A}} = A$



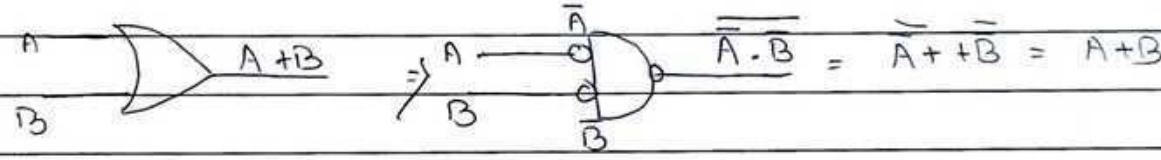
- An AND gate is equivalent to an inverted-input NOR gate



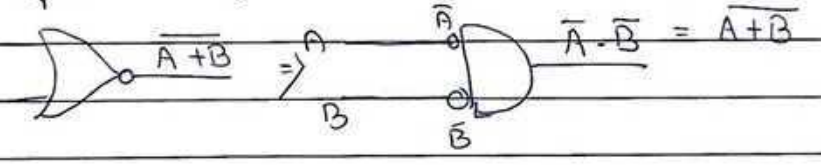
- A NAND gate is equivalent to an inverted-input OR gate



→ An OR gate is equivalent to an inverted input NAND gate



→ A NOR gate is equivalent to an inverted input AND gate



Unit-2

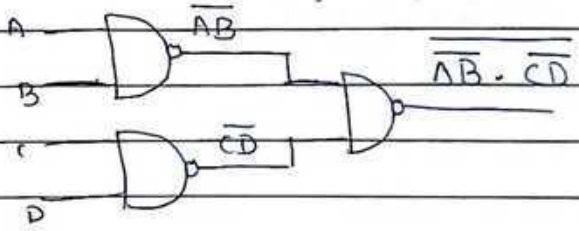
NAND implementation :-

→ The implementation of boolean fun^n^s with NAND gates requires that the fun^n^s be in sum of products (SoP) form.

① $F = AB + CD$

$\Rightarrow F = \overline{\overline{AB} + \overline{CD}}$

$= \overline{\overline{AB} \cdot \overline{CD}}$



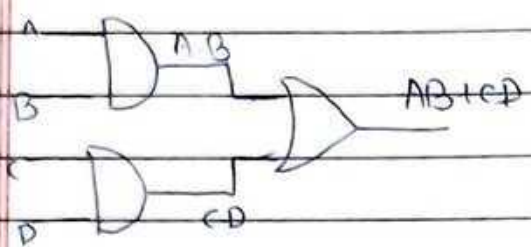
→ The general procedure for implementation of a Boolean fun^n^ with NAND gates using mixed notation is as follows :

1) Draw the Boolean fun^n^ by basic gates such as AND, OR and NOT gates.
2) Convert all AND gates to NAND gates.
3) Convert all OR gates to inverted OR symbols (to be called as NAND gate)
4) Check all the bubbles in the diagram.

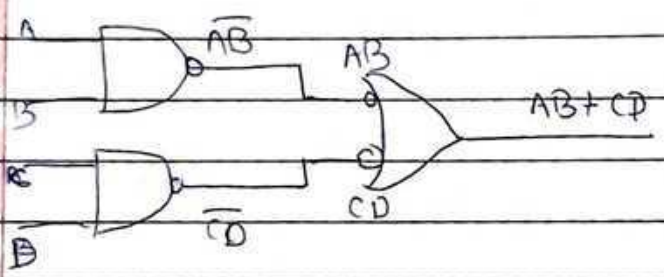
→ By using the above step ; the fun^ $F = AB + CD$ can be designed as follows :-

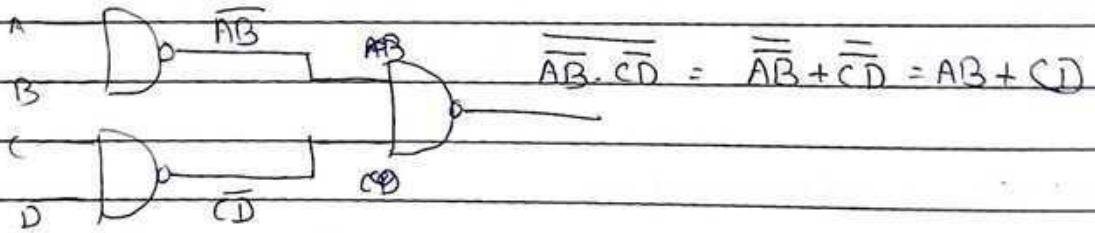
$F = AB + CD$

step (1)



step (2)





(2) $F = xy' + x'y + z$

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

classmate
Date
Page

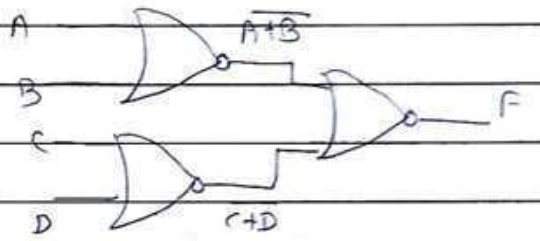
NOR implementation :-

→ The implementation of Boolean fun's with NOR gates requires that the fun's be in product-of-sums form (PoS).

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

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

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

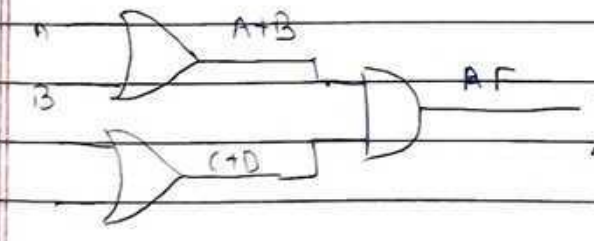


→ The general procedure for implementation of a Boolean fun^n with NOR gates using mixed notation is as follows :-

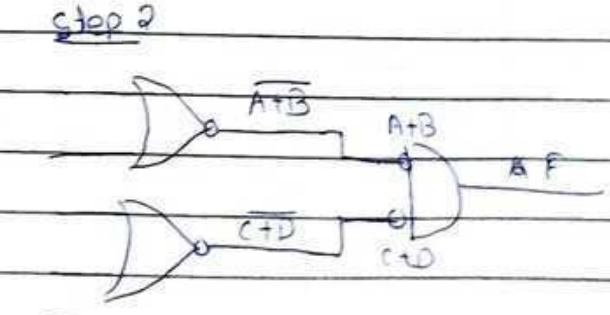
1) Draw the Boolean fun^n by basic gates such as AND, OR, and NOT gates.
2) Convert all OR gates to NOR gates.
3) Convert all AND gates to inverted-AND graphic symbols (i.e called as NOR gate).
4) Check all the bubbles in the diagram.

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

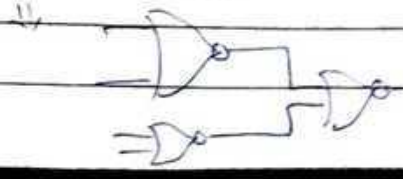
Step 1



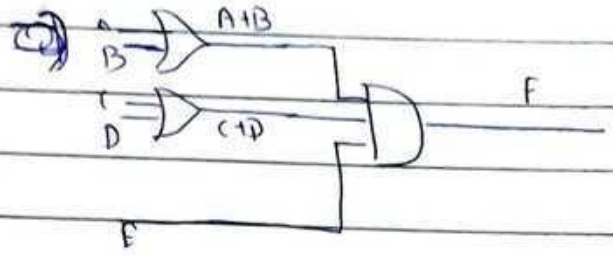
Step 2

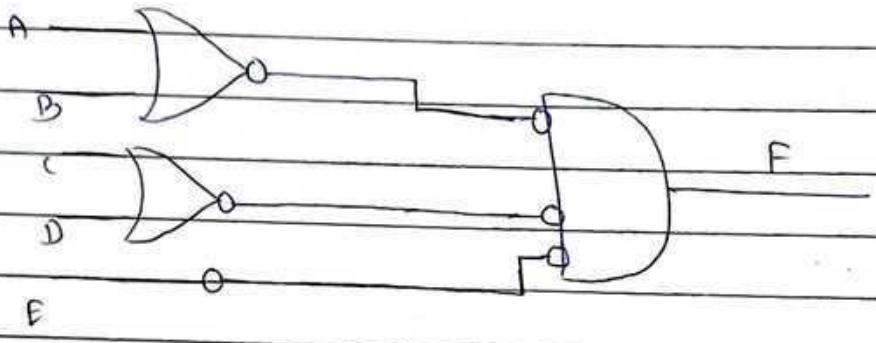


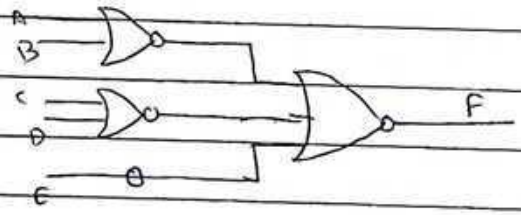
4)



$F = (A+B)(C+D)E$

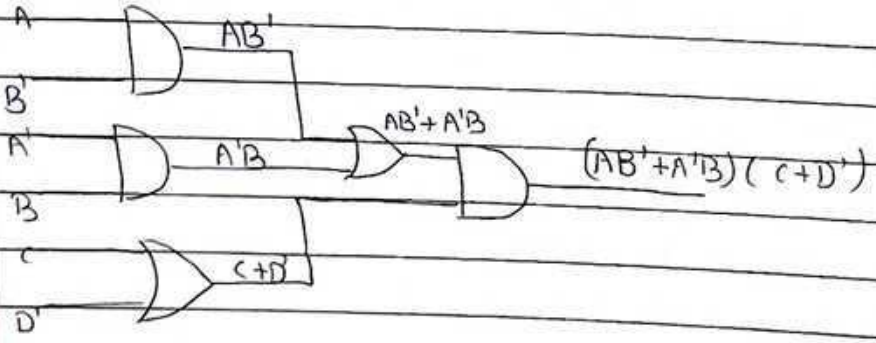




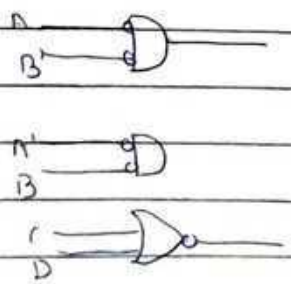


(2) $F = (AB' + A'B)(C+D')$

Step 1



Step 2



01/09/2025

K-maps - Two, three and four variable K-maps, Don't care conditions :-

The map method :-

→ The map method provides a simple, straightforward procedure for minimizing Boolean fun's.

→ This method may be regarded as a pictorial form of a truth table.

→ The map method is also known as the Karnaugh map or K-map

→ The simplified expressions produced by the map are always in one of the two standard forms : Sum of products or product of sums.

Two-Variable K-map :-

→ The two-variable K-map is shown in Fig.

→ There are 4 minterms for 2 variables ; hence, the map consists of 4 squares, one for each minterm.

m_0	m_1	
m_2	m_3	

(a)

x \ y	0	1
0	m_0 $x'y'$	m_1 $x'y$
1	m_2 xy'	m_3 xy

(b)

→ The map is redrawn in (b) to show the relationship bet^n the squares and the two variables x and y.

→ The 0 and 1 marked in each row and column designate the values of variables.

The rules of K-map simplification are :-

→ Groupings can contain only 1s ; no 0s.

→ Groups can be formed only at right angles (horizontal or vertical) ; diagonal groups are not allowed.

→ The no. of 1's in a group must be a power of 2.

→ The groups must be made as large as possible.

$2^1 = 2$, $2^2 = 4$, $2^3 = 8$,
↑ pair ↑ Quad ↑ octal

→ Each cell containing a one must be in at least one group
→ Groups can overlap.
→ Groups can wrap around the sides of the k-map.
→ The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.

Example :- Simplify the Boolean functions

1) $F(x,y) = \Sigma(0,1)$

x \ y	0	1
0	1 (00)	1 (01)
1	0 (10)	0 (11)

$F(x,y) = x'$

2) $F(x,y) = \Sigma(2,3)$

x \ y	0	1
0	0	0
1	1	1

$F(x,y) = x$

3) $F(x,y) = \Sigma(0,2)$

x \ y	0	1
0	1	0
1	1	0

$F(x,y) = y'$

4) $F(x,y) = \Sigma(1,3)$

	0	1
0	0	1
1	0	1

$F(x,y) = y$

5) $F(x, y) = \Sigma(1, 2, 3)$

x \ y	0	1
0	0 (00)	1 (01)
1	1 (10)	1 (11)

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

6) $F(x, y) = \Sigma(0, 2, 3)$

x \ y	0	1
0	1	0
1	1	1

$F(x, y) = x + y'$

7) $F(x, y) = \Sigma(0, 1, 3)$

x \ y	0	1
0	1	1
1	0	1

$F(x, y) = x' + y$

8) $F(x, y) = \Sigma(0, 1, 2)$

x \ y	0	1
0	1	1
1	1	0

$F(x, y) = x' + y'$

Three-Variable k-Map:-

- A 3-variable k-map is shown in Fig.
- There are 8 minterms for 3 binary variables; therefore, the map consists of 8 squares.
- Note that the minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code
- The characteristic of this sequence is that only one bit changes in value from one adjacent column to the next.

* The map drawn in part (b) is marked with numbers in each row and each column to show the relationship between the squares and the 3 variables.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

A \ BC	00	01	11	10
0	000 $\bar{A}\bar{B}\bar{C}$ 0	001 $\bar{A}\bar{B}C$ 1	011 $\bar{A}BC$ 3	010 $\bar{A}B\bar{C}$ 2
1	100 $A\bar{B}\bar{C}$ 4	101 $A\bar{B}C$ 5	111 ABC 7	110 $AB\bar{C}$ 6

Example: Simplify the Boolean funcs:-

1) $F(A, B, C) = \Sigma(0, 4)$

A \ BC	00	01	11	10
0	1	0	0	0
1	1	0	0	0

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

2) $F(A,B,C) = \Sigma(1,3)$

A \ BC	00	01	11	10
0	0	1	1	0
1	0	0	0	0

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

3) $F(A,B,C) = \Sigma(4,5)$

A \ BC	00	01	11	10
0	0	0	0	0
1	1	1	0	0

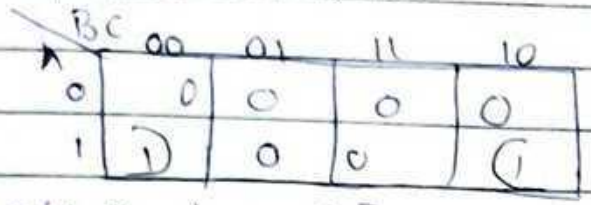
$F(A,B,C) = A\bar{B}$

4) $F(A,B,C) = \Sigma(0,2)$

A \ BC	00	01	11	10
0	1	0	0	1
1	0	0	0	0

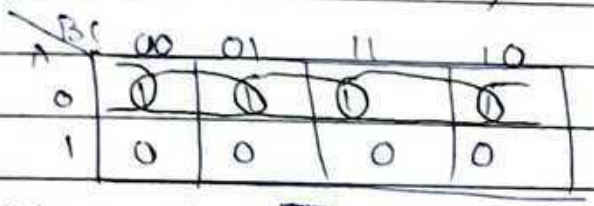
$F(A,B,C) = \bar{A}\bar{C}$

5) $F(A,B,C) = \Sigma(4,6)$



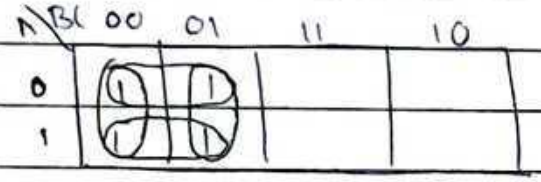
$F(A,B,C) = A\bar{C}$

7) $F(A,B,C) = \Sigma(0,1,2,3)$



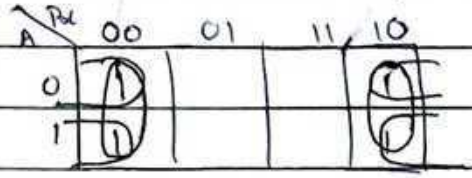
$F(A,B,C) = \bar{A}$

8) $F(A,B,C) = \Sigma(0,1,4,5)$



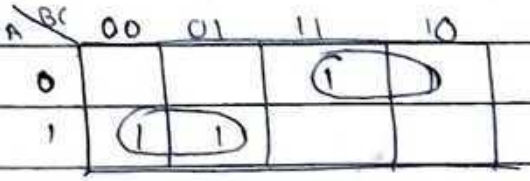
$F(A,B,C) = \bar{B}$

9) $F(A,B,C) = \Sigma(0,2,4,6)$



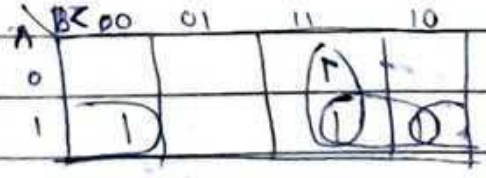
$F(A,B,C) = \bar{C}$

10) $F(A,B,C) = \Sigma(2,3,4,5)$



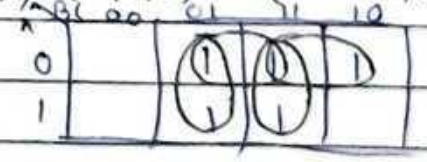
$F(A,B,C) = A\bar{B} + \bar{A}B$

11) $F(A,B,C) = \Sigma(3,4,6,7)$



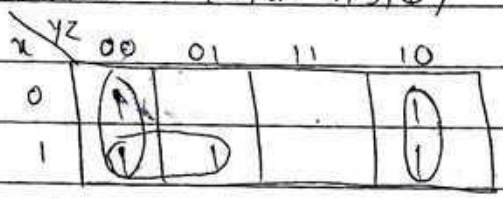
$F(x,y,z) = BC + AB + A\bar{C}$

12) $F(A,B,C) = \Sigma(1,2,3,5,7)$



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

12) $F(x,y,z) = \Sigma(0,2,4,5,6)$



$F(x,y,z) = xy' + x'z'$

Four - Variable K-map

classmate

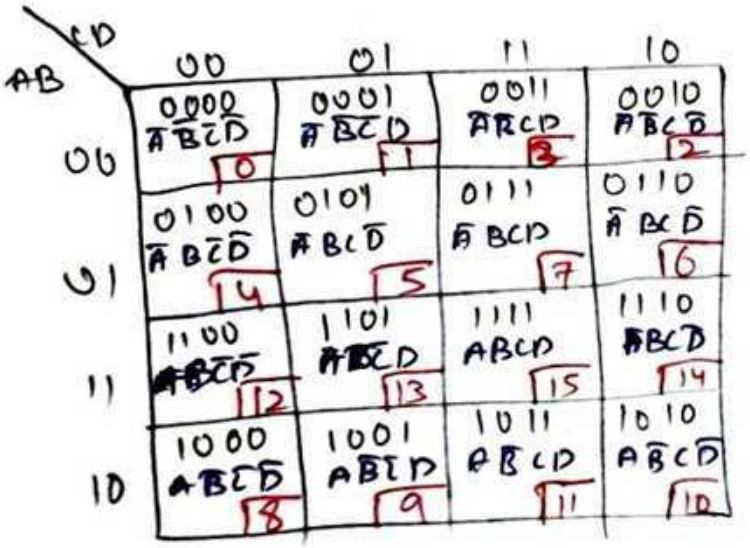
-> The map for Boolean function of four binary variable (A, B, C, D) is shown in fig.

-> In fig. listed the 16 minterms and the sequence assigned to each.

-> In fig, the map is redrawn to show the relationship between th squares and the four variables.

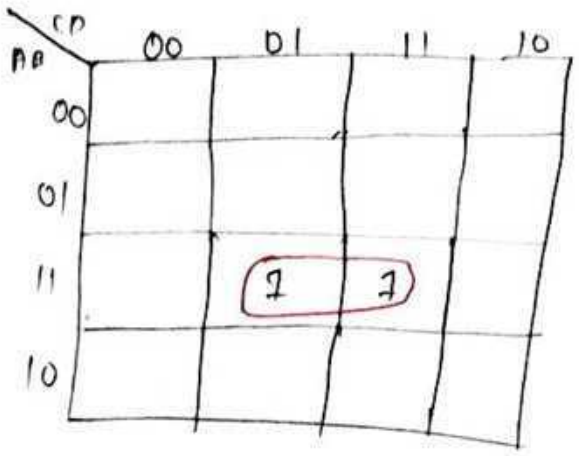
-> The rows and columns are number is a gray code sequence with only one digit changing the value between two adjacent row on columns.

-> The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number



→ Example · Simplify the Boolean function.

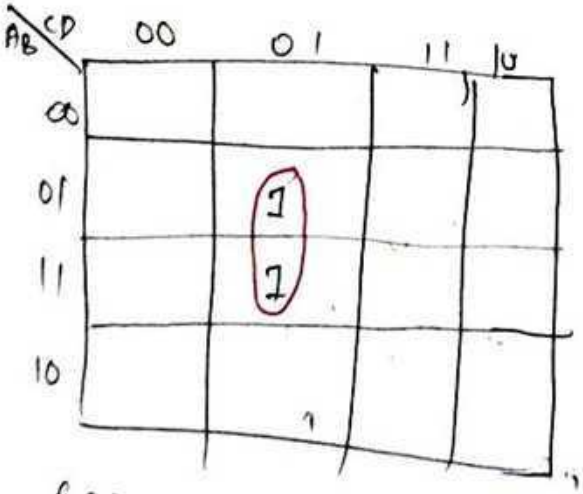
① $F(A,B,C,D) = \Sigma(13,15)$.



1101
1111

$F(A,B,C,D) = ABD$

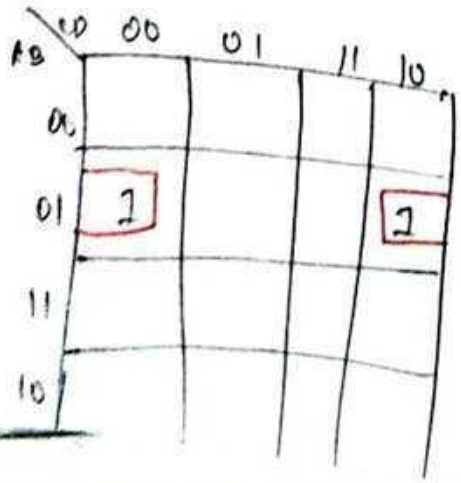
② $F(A,B,C,D) = \Sigma(5,13)$



0101
1101

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

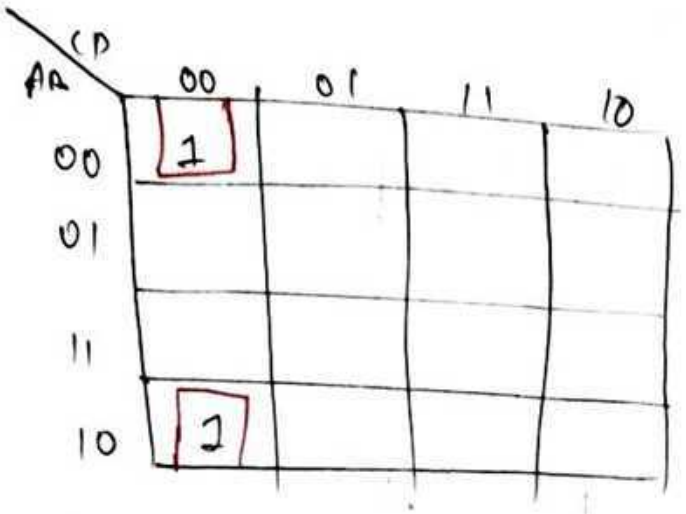
③ $f(A,B,C,D) = \Sigma(4,6)$.



0100
0110

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

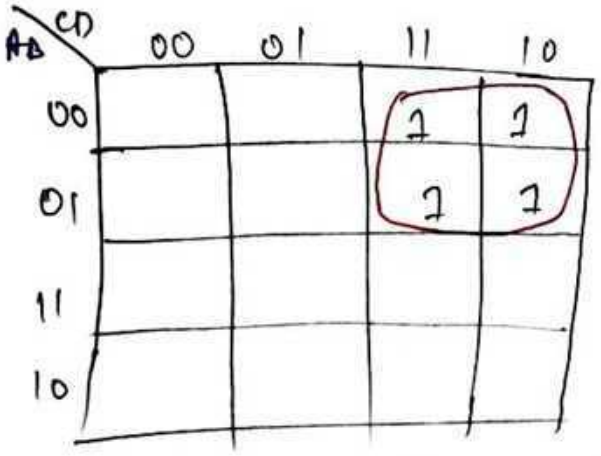
$= \Sigma(0,8)$.



0000
1000

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

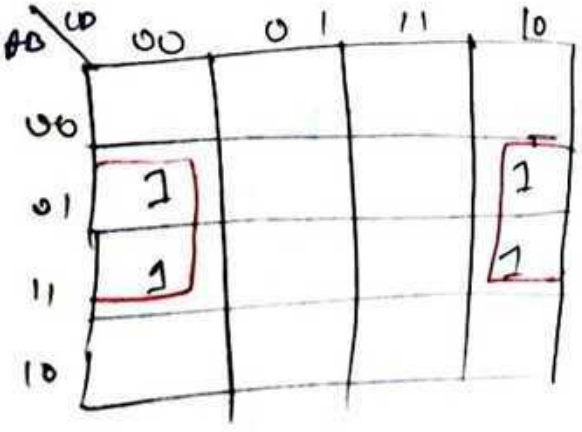
(5) $f(A,B,C,D) = \Sigma(2,3,6,7)$



0011
0010
0111
0110

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

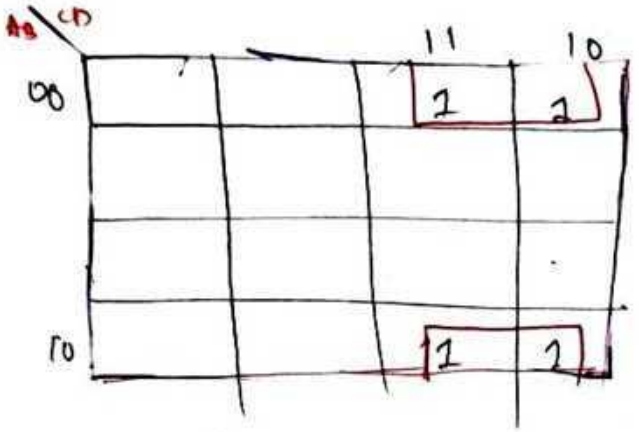
(6) $f(A,B,C,D) = \Sigma(4,6,12,14)$



0100
0110
1100
1110

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

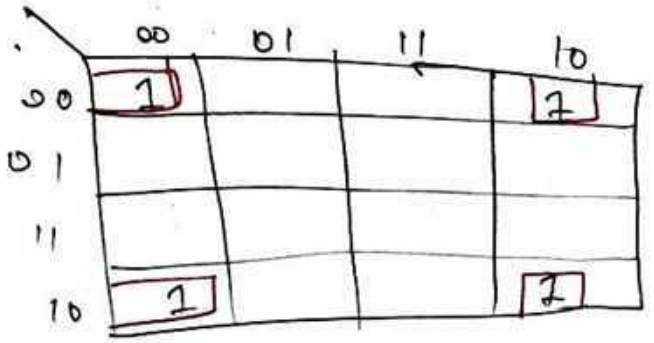
⑦ $f(A,B,C,D) = \Sigma(2,3,10,11)$.



0011
0010
1011
1010

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

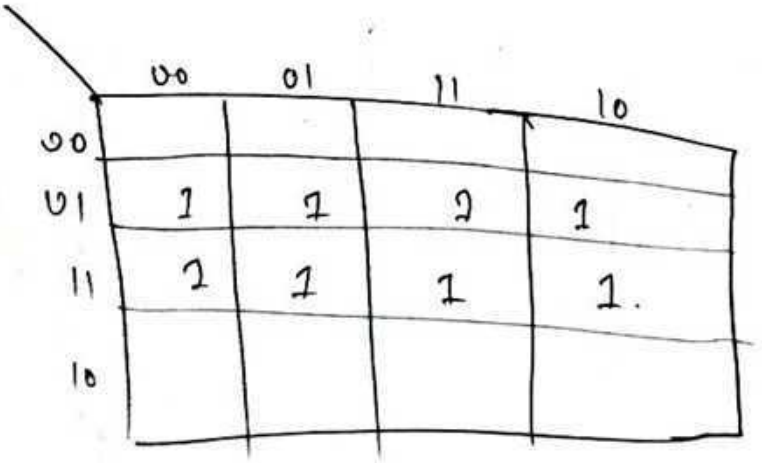
⑧ $f(A,B,C,D) = \Sigma(0,2,8,10)$



0000
0010
1000
1010

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

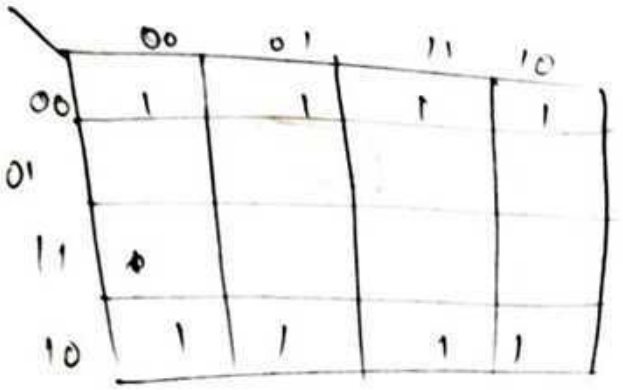
⑨ $f(A,B,C,D) = \Sigma(4,5,6,7,12,13,14,15)$



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

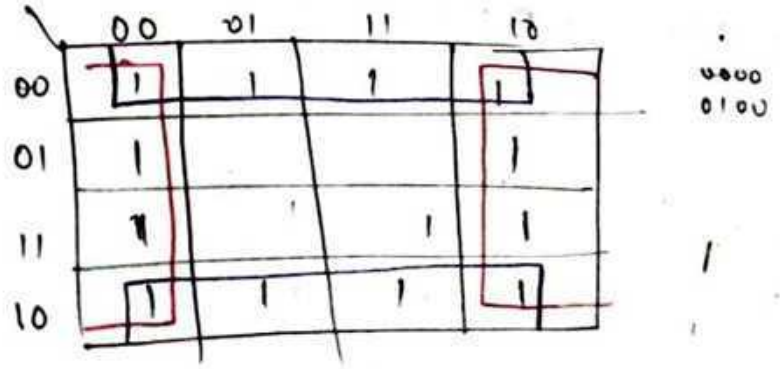
classmate

⑩ $f(A,B,C,D) = \Sigma(0,1,2,3,8,9,10,11)$



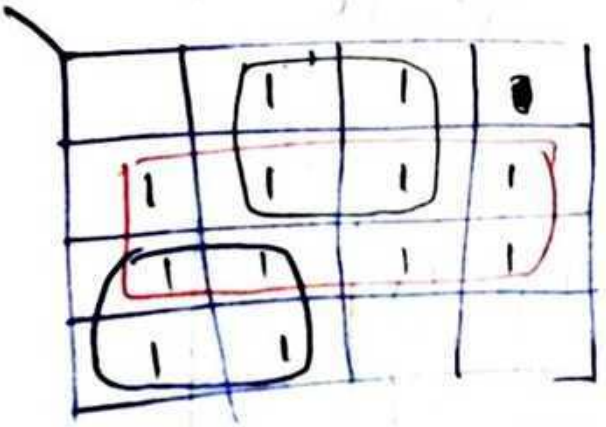
$f(A,B,C,D) = \overline{B}$

⑪ $f(A,B,C,D) = \Sigma(0,1,2,3,4,5,8,9,10,11,12,14)$



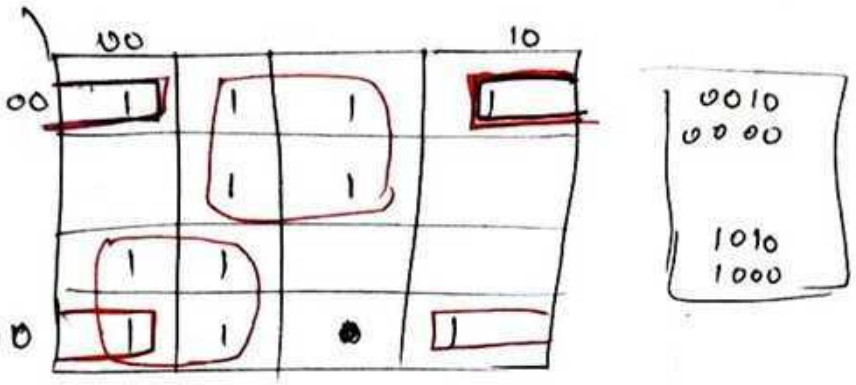
$f(A,B,C,D) = \overline{B} + \overline{D}$

⑫ $f(A,B,C,D) = \Sigma(1,3,4,5,6,7,8,9,12,13,14,15)$



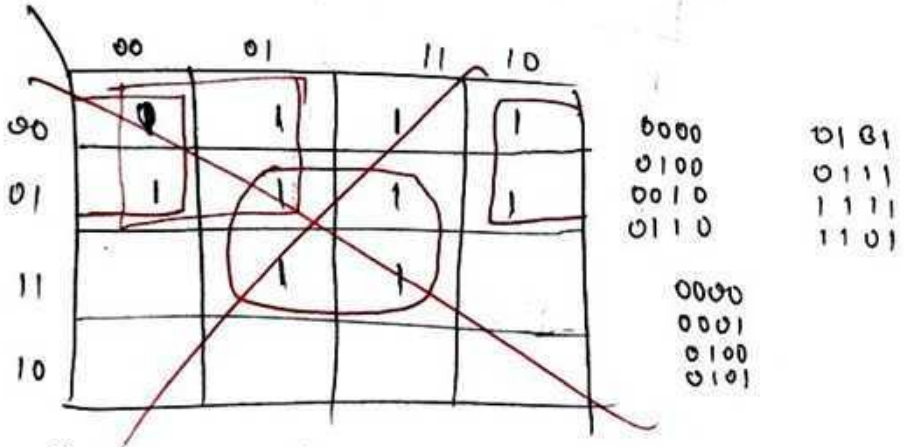
$f(A,B,C,D) =$ ~~B + D~~ $B + A'D + AC'$

(13) $f(A,B,C,D) = \Sigma(0,1,2,3,5,7,8,9,10,12,13)$

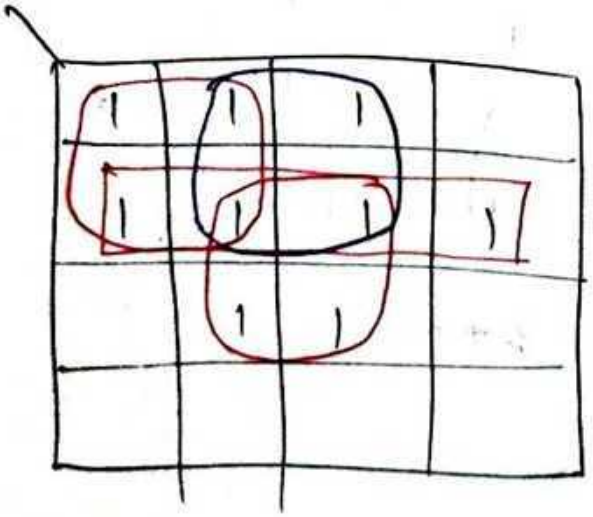


$f(A,B,C,D) = AD + AC' + \cancel{A\bar{B}\bar{C}\bar{D}} B'D'$

(14) $f(A,B,C,D) = \Sigma(0,1,3,4,5,6,7,13,15)$



$f(A,B,C,D) = P'B' + QS + P'R'$



$f(A,B,C,D) = QS + P'R' + P'S + P'Q.$

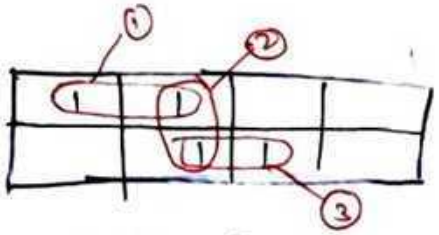
Prime Implicants :-

→ In a Karnaugh map, adjacent squares represent minterms that differ by only one variable.

→ A group of squares or rectangle made up of a ~~bunch~~ bunch of adjacent minterms which is allowed be the definition of k-map are called prime implicants (PI).

→ All possible groups formed in k-maps are called as PI.

→ Ex -

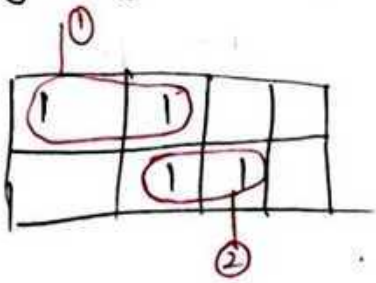


no. of PI = 3.

Essential Prime Implicants (EPI) :-

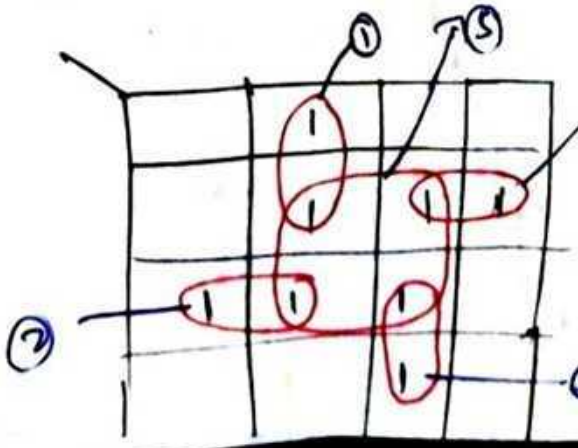
→ These are those group that covers at least one minterm that can't be covered by any other prime implicant (PI).

→ Ex.



no. of EPI = 2

① $f = \Sigma(1,5,6,7,11,12,13,15)$ find no. of PI, and EPIs.



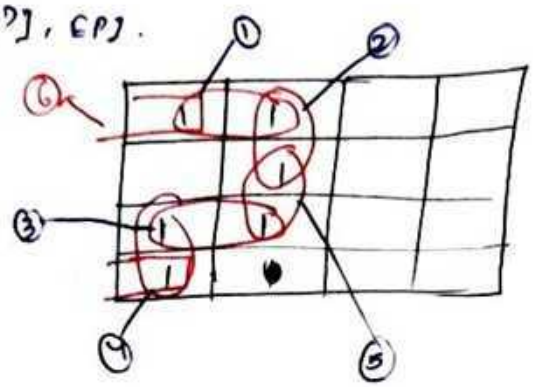
PI = 5 {1, 2, 3, 4, 5}

EPI = 4 {1, 2, 3, 4}.

Exm: $BD + A'C'D + A'BC + ACD + ABC'$

② $F = \Sigma(0, 1, 5, 9, 12, 13)$

PI, EPI.

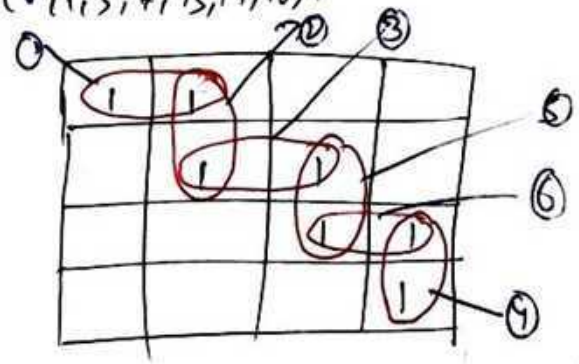


$PI = 5\{1, 2, 3, 4, 5, 6\}$.

$EPI = 0$

Exp :

③ $F = \Sigma(0, 1, 5, 7, 15, 14, 10)$.



$PI = 6\{1, 2, 3, 4, 5, 6\}$.

$EPI = 2\{1, 4\}$.

Donot ← Care Condition :-

→ A don't care minterms is a combination of variable whose logical value is not specified.

→ To distinguish the don't care condition from 1's and 0's, an X is used.

→ Thus an X inside a square in the map indicates that we don't care whether value 0 or 1 is assigned to F for the particular minterm.

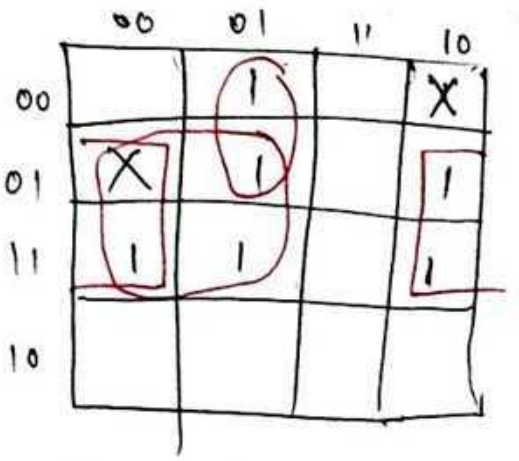
→ Those don't care condition can be used on a map to provide further simplification of the boolean expression.

classmate

→ Ex - Simplify the following Boolean function, together with the don't care conditions d.

① $f(A,B,C,D) = \Sigma(1,5,6,12,13,14)$

$d(A,B,C,D) = \Sigma(2,4)$.

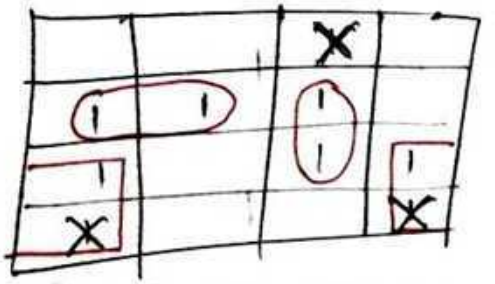


0100
1100
0110
1110

$f(A,B,C,D) = BC' + BD' + A'C'D$

② $f(A,B,C,D) = \Sigma(4,5,7,12,14,15)$

$d(A,B,C,D) = \Sigma(3,8,10)$.

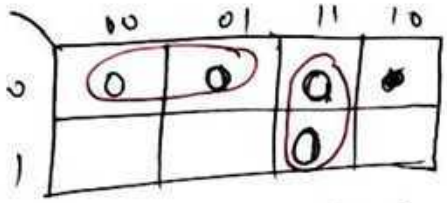


$F(A,B,C,D) = AD' + A'BC' + BCD$.

Product of Sums Simplification

Simplify the following function in POS form.

① $F(x,y,z) = \Pi(0,1,3,7)$



000
001
011
111

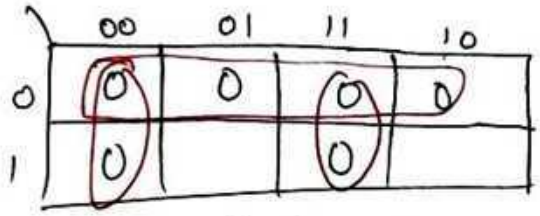
$f'(x,y,z) = \bar{x}\bar{y} + yz$

$F(x,y,z) = \overline{(\bar{x}\bar{y} + yz)}$

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

$= (x+y)(y+z)$.

② $f(A,B,C) = \Pi(0,1,2,3,4,7)$



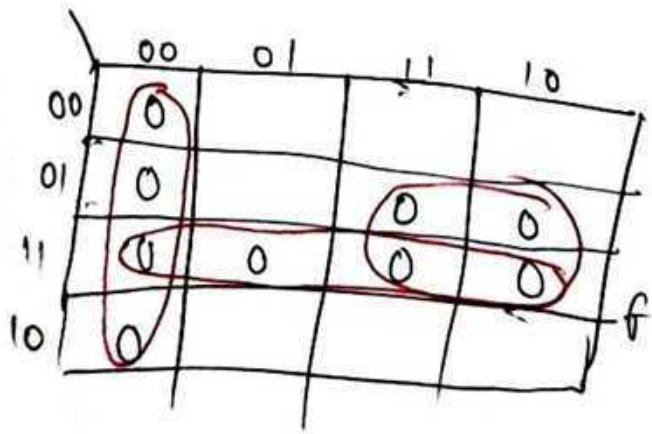
000
100
011
111

$f'(x,y,z) = A' + B'C' + BC$

$f(x,y,z) = \overline{A' + B'C' + BC}$

$= (A)(B+C)(B'+C')$

③ $f(A,B,C,D) = \Pi(0,4,6,7,8,12,13,14,15)$



0111
0110
1111
1110

$f'(x,y,z) = BC'D' + AB$

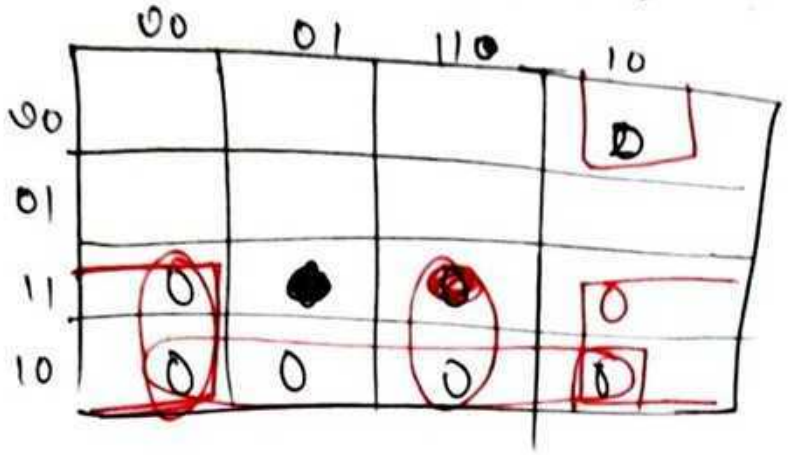
BC

$f(x,y,z) = (C+D)(A+B$

$(B'+C')$

④ $f(A,B,C,D) = \pi(2,9,8,10,11,12,14)$

classmate



0010
1010

$f'(x,y,z) = AB' + \ldots D' + \ldots + B'C D'$

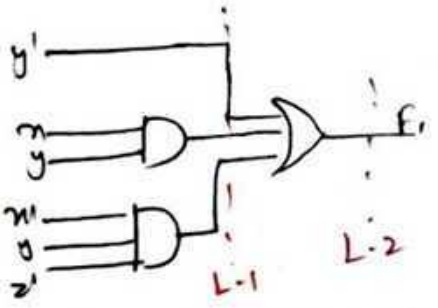
Two-level logic :-

→ The maximum numbers of levels that are present between inputs and output is two in two logic level.

→ Here, the outputs of first level logic gates connected as input of second level logic gate (s).

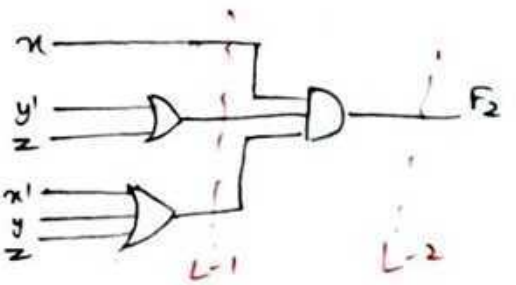
Two-level implementation

$F_1 = y' + xy + x'yz'$



(Sum of products)

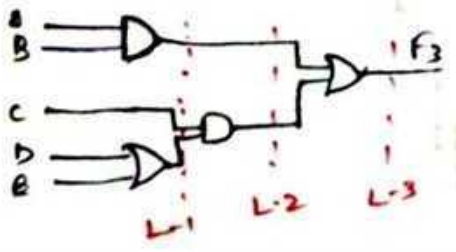
$f_2 = x(y'+z)(x'+y+z)$



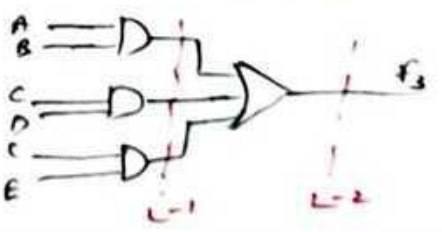
(Product of sums)

Multi-level- implementation

$F_2 = AB + C(D+E)$



$f_3 = A+B+(D+E)$
$= AB + CD + CE$



05/09/2025

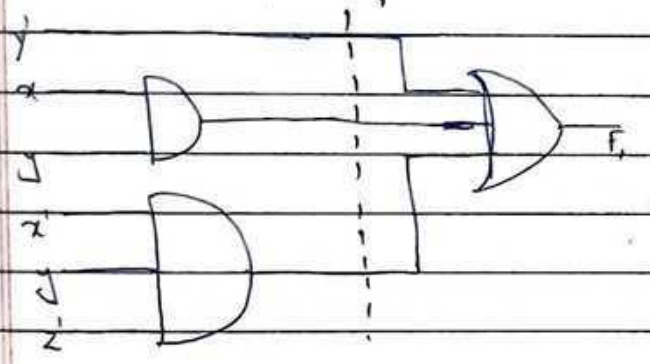
Two-level logic :-

→ The max^m no. of levels that are present between inputs and output is two in two level logic.

→ Here, the outputs of first level logic gates are connected as inputs of second level logic gate(s).

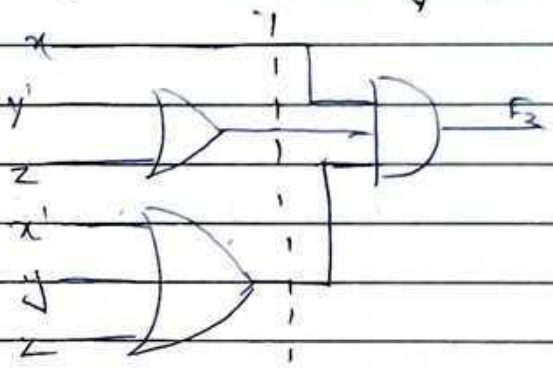
Two-level implementation :-

$F_1 = y' + xy + x'yz'$



(SOP)

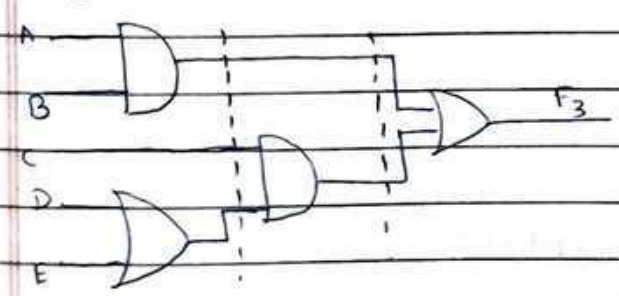
$F_2 = x(y'+z)(x'+y+z)$



(POS)

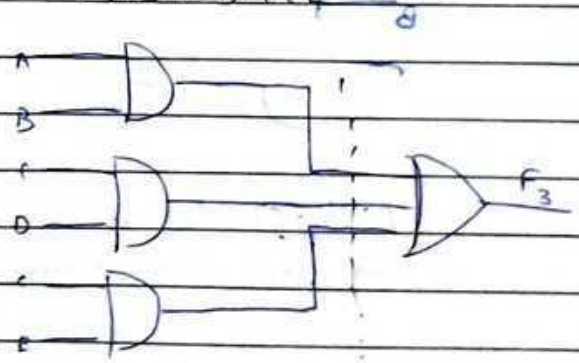
Multi-level implementation :-

$F_3 = AB + C(D+E)$



$F_3 = AB + C(D+E)$

$= AB + CD + CE$



→ The standard type (SOP/POS) circuit configuration is referred to as a two-level implementation.

→ In general, a two-level implementation is preferred because it produces the least amount of delay through the gates when the signal propagates from the inputs to the output.

Other Two-level implementations :-

→ For two-level logic implementation, we consider four types of gates : AND, OR, NAND, and NOR.

→ If we assign one type of gate for the first level and one type for the second level, we find that there are 16 possible combinations of two-level forms.

→ Combinations are :-

AND-AND	OR-OR	NAND-NAND	NOR-NOR
AND-OR	OR-AND	NAND-AND	NOR-AND
AND-NAND	OR-NAND	NAND-OR	NOR-OR
AND-NOR	OR-NOR	NAND-NOR	NOR-NAND

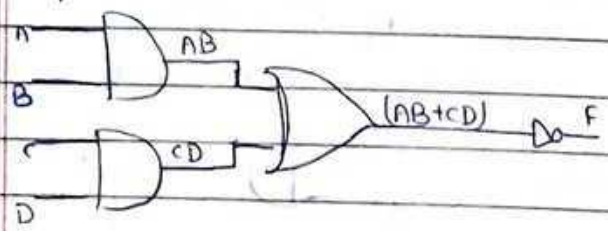
AOI Logic and OAI Logic :-

→ The logic fun^n $F = (AB+CD)'$ is called an AND-OR-INVERT fun^n.

→ The logic fun^n $F = [(A+B)(C+D)]'$ is called an OR-AND-INVERT fun^n

AOI fun^n

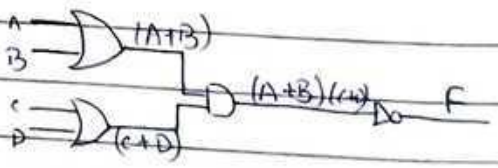
$F = (AB+CD)'$



OAI fun^n

~~$F = (AB+CD)'$~~

$F = [(A+B)(C+D)]'$



AND-OR-INVERT implementation :-

→ The AND-NOR form resembles the AND-OR form, but with an inversion done by the bubble in the output of the NOR gate.

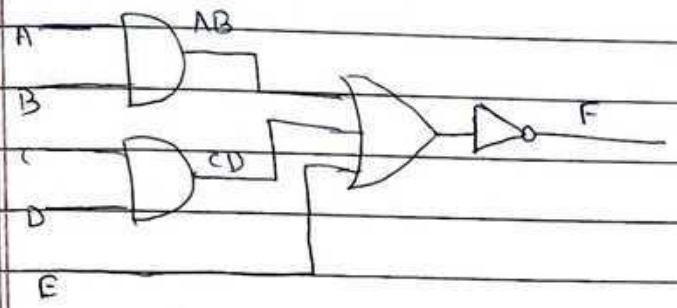
→ It implements the fun^n $F = (AB+CD+E)'$

→ An AND-OR implementation requires an expression in SOP form.

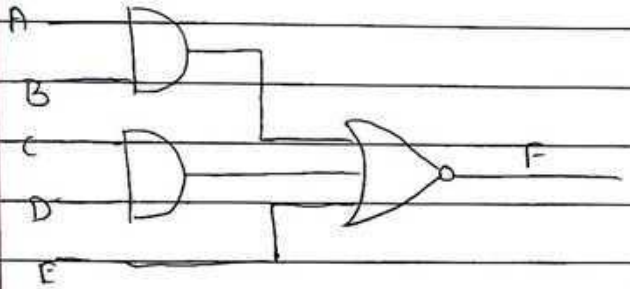
→ The two forms, NAND-AND and AND-NOR, are equivalent and can be treated together. Both perform the AOI fun^n

AOI

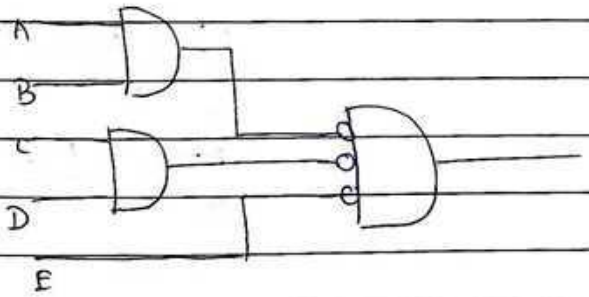
$F = \overline{(AB + CD + E)}'$



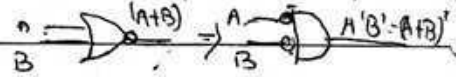
AND - NOR



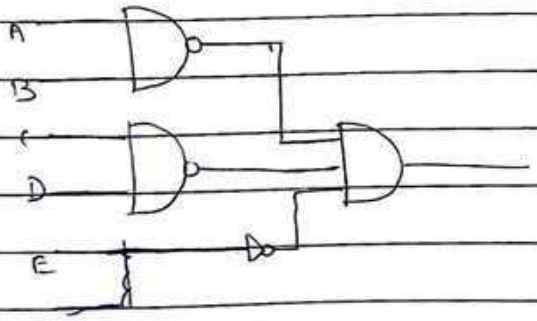
NAND - AND - NOR



NOR ⇒ Inverted input AND



NAND - AND



OR-AND-INVERT implementation :-

→ The OR-NAND form resembles the OR-AND form, except for the inversion done by the bubble in the NAND gate.

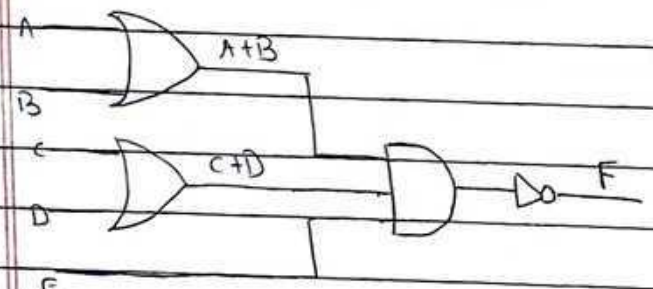
→ It implements the fun^n $F = [(A+B)(C+D)E]'$

→ The OR-AND-INVERT implementation requires an expression in POS form.

→ The OR-NAND and NOR-OR forms perform the OR-AND-INVERT fun^n.

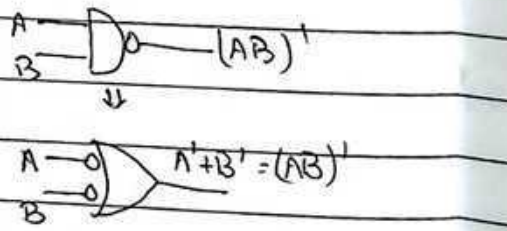
OAI :-

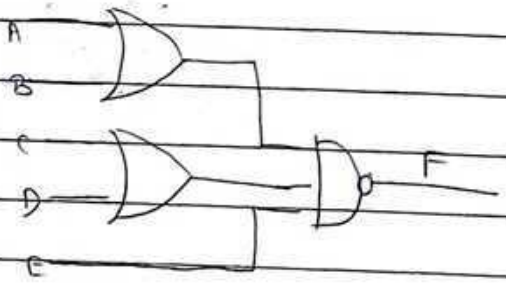
$F = [(A+B)(C+D)E]'$



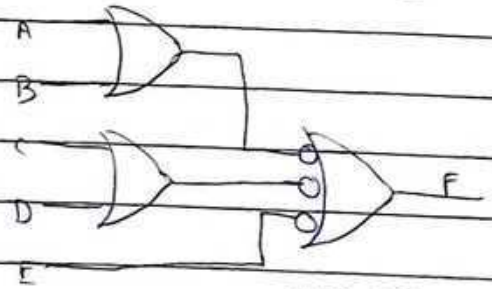
OR-NAND

NAND = inverted input OR

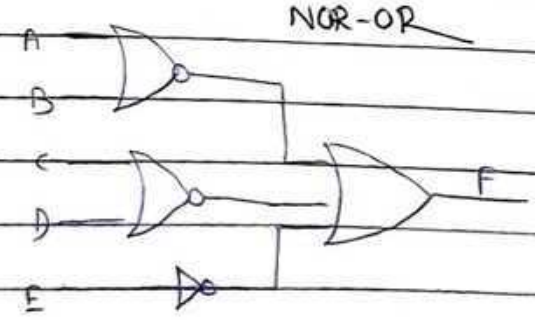




OR NAND



NOR-OR



Example:

Implement the fun $F = x'y'z' + xyz'$ with the four 2-level forms AND-NOR, NAND-AND, OR-NAND and NOR-OR.

Solution:

AND-OR-INVERT form:

$$F = x'y'z' + xyz'$$

$$F' = (x'y'z' + xyz')'$$

$$= (x'y'z')'(xyz')'$$

$$= (x+y+z)(x'+y'+z)$$

$$= xx + xy' + xz + x'y + yy' + yz + x'z + y'z + zz$$

$$= xy' + xz + x'y + yz + x'z + y'z + z$$

$$= xy' + x'y + xz + yz + x'z + y'z + z$$

$$= xy' + x'y + xz + x'z + yz + y'z + z$$

$$= xy' + x'y + z(x+x') + z(y+y') + z$$

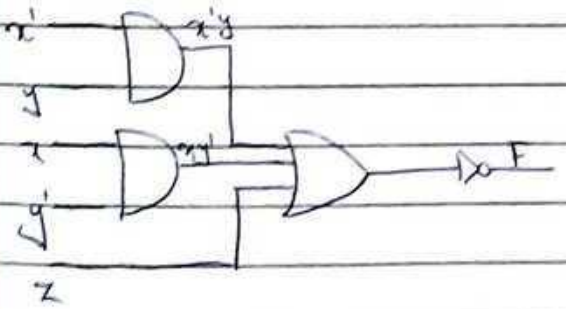
$$= xy' + x'y + z + z + z$$

$$= x'y + xy' + z$$

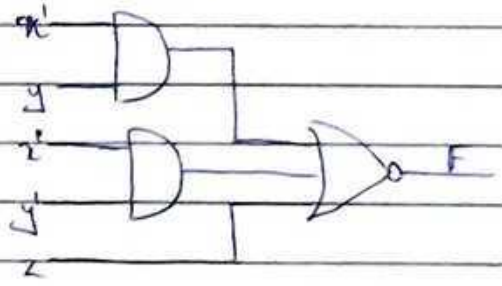
$$F' = x'y + xy' + z$$

$$F = (x'y + xy' + z)'$$

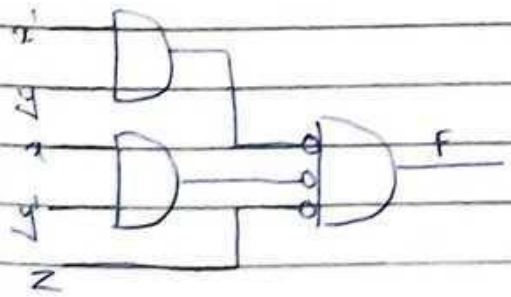
AOI :-



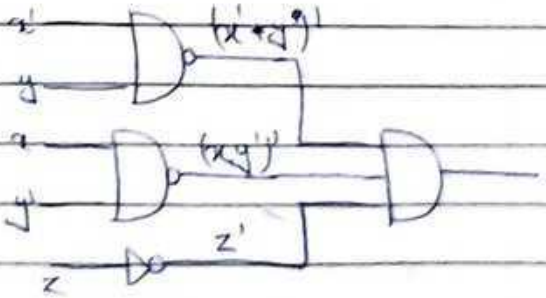
AND-NOR



AND-NOR



NAND-AND



OR-AND-INVERT :-

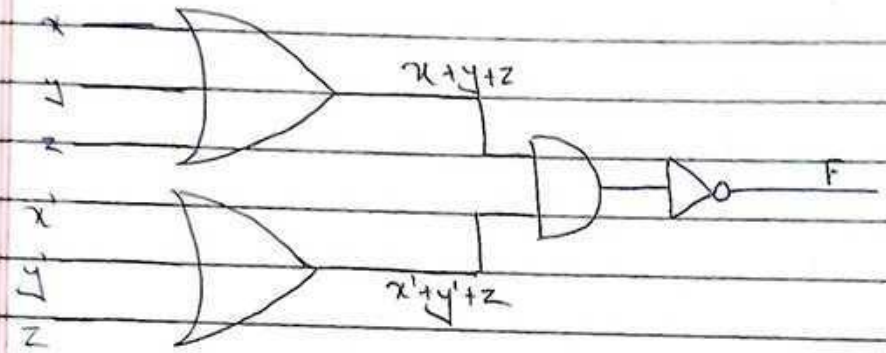
$F = x'y'z' + xyz'$

$F' = (x'y'z')'(xyz')'$

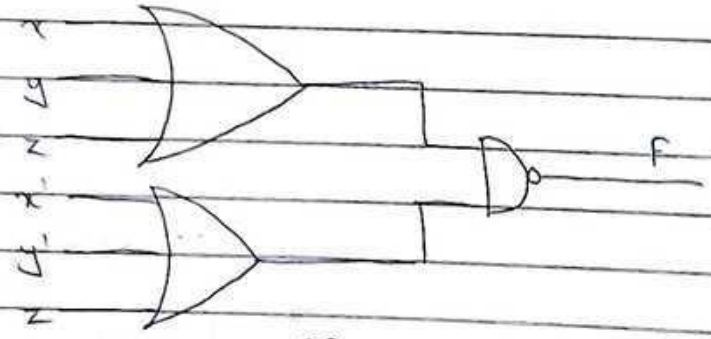
$= (x+y+z)(x'+y'+z)$

$F = [(x+y+z)(x'+y'+z)]'$

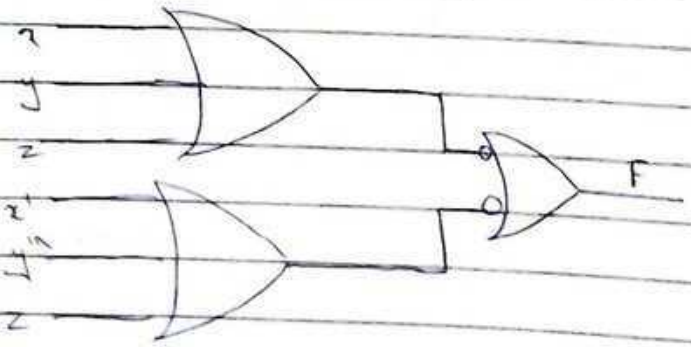
OAI :-



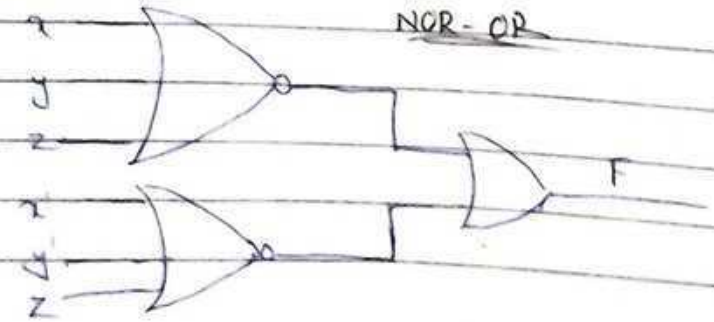
OR-NAND :-



OR-NAND



NOR-OR



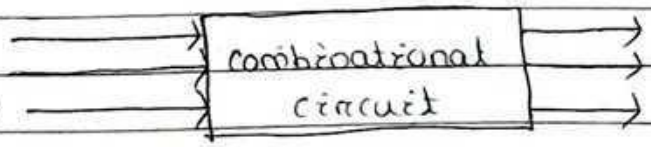
Unit-2 :- chapter-2:-

Combinational Logic Circuit :-

→ Logic circuits for digital System may be combinational or Sequential.
→ A combinational circuit consist of logic gates whose outputs at anytime are determined from only the present combination of inputs.
→ A combinational circuit performs an operation that can be Specified logically by a set of Boolean fun's.
→ In contrast, Sequential circuit employ Storage elements in addition to logic gates.
→ Their outputs are a fun° of the inputs and the State of the Storage elements.
→ Because the State of the storage elements is a fun° of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the ckt behaviour must be specified by a time · Sequence of inputs and internal states.

Combinational Circuits :-

→ A combinational ckt consists of an interconnection of logic gates.
→ Combinational logic gates react to the values of the Signals at their inputs and produce the value of the o/p signal, transforming binary info from the given i/p data to a required o/p data.



→ The n i/p binary variables come from an external Source; the m o/p variables are produced by the internal combinational logic circuit & go to an external destination.
→ Each i/p & o/p variable exists physically as an analogy Signal whose values are interpreted to be a binary Signal that represents logic 1 & logic 0.

→ For n i/p variables, there are 2^n possible combinations of the binary inputs.
→ For each possible i/p combination, there is one possible value for each o/p variable.
→ Thus, a combinational ckt can be specified with a truth table that lists the o/p values for each combination of i/p variables.
→ A combinational ckt also can be described by m Boolean fun's, one for each o/p variable. Each o/p fun? is expressed in terms of the n. i/p variables

Analysis Procedure :-
→ The analysis of a combinational ckt requires that we determine the fun^n that the ckt implements.
→ This task starts with a given logic diagram & culminates with a set of Boolean fun^n, a truth table or possibly, an explanation of the ckt operation

Design Procedure :-
→ The procedure involves the following steps :
1. From the specifications of the ckt, determine the required no. of i/ps and o/ps and design a symbol to each.
2. Derive the truth table that defines the required relation bet^n i/ps and outputs.
3. Obtain the simplified Boolean fun^n for each o/p as a fun^n of the i/p variables.
4. Draw the logic diagram and verify the correctness of the design (manually by on simulation).

Adders :-

→ The most basic arithmatic operation is the addition of 2 binary bits.

→ This simple addition consist of 4 possible operations, namely

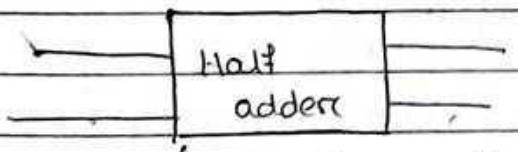
```
  0   0   1   1
  0   1   0   1
                (carry)
  0   1   1   0
```

Half adder :-

→ A combinational ckt that performs the addition of 2 bits is called a half adder.

→ A half adder is a combinational ckt with 2 binary i/Ps (augned and addend bits) and 2 binary o/Ps (sum & carry bits).

→ It adds the 2 i/Ps (A & B) and produces 2 outputs sum(s) and carry (c).



(Block diagram)

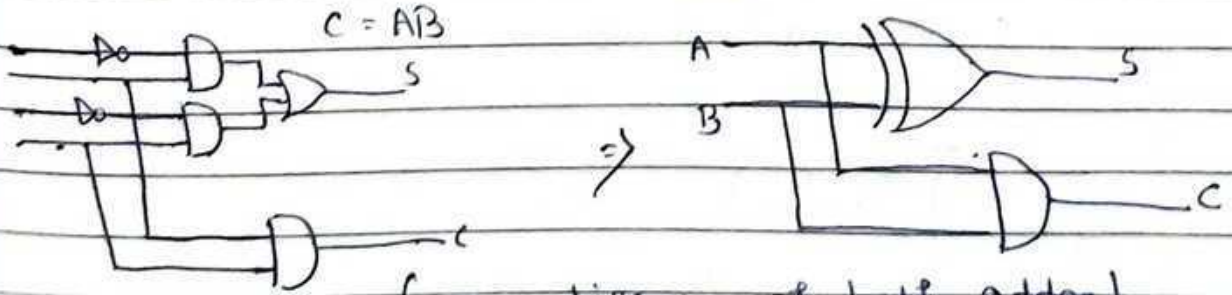
i/P		o/P	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(Truth table)

→ The Simplified Boolean fun^n for the 2 o/Ps can be obtained directly from the truth table. The simplified sum of products expression are

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

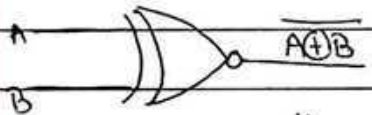
$$C = AB$$

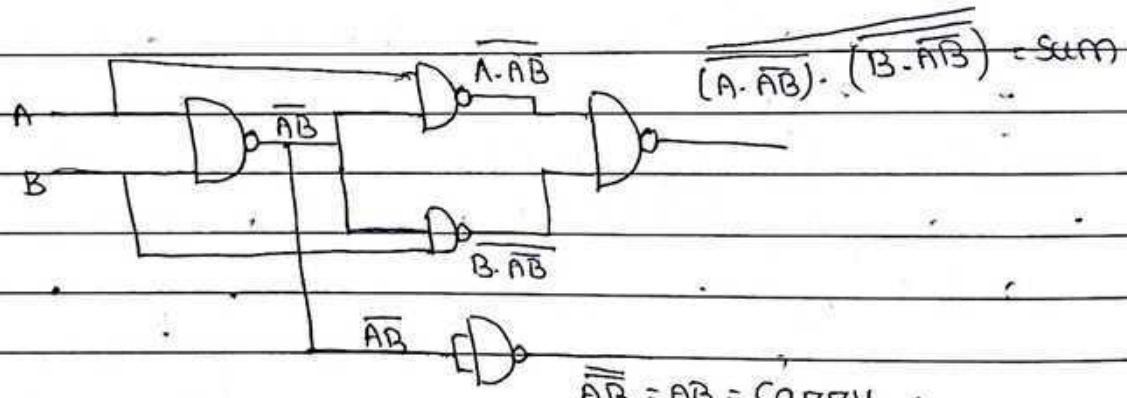


(Logic diagram of half adder)

Implementation half adder by using only NAND gates :-

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

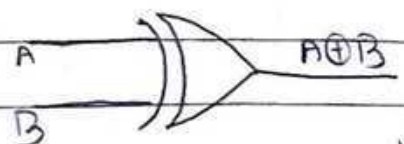


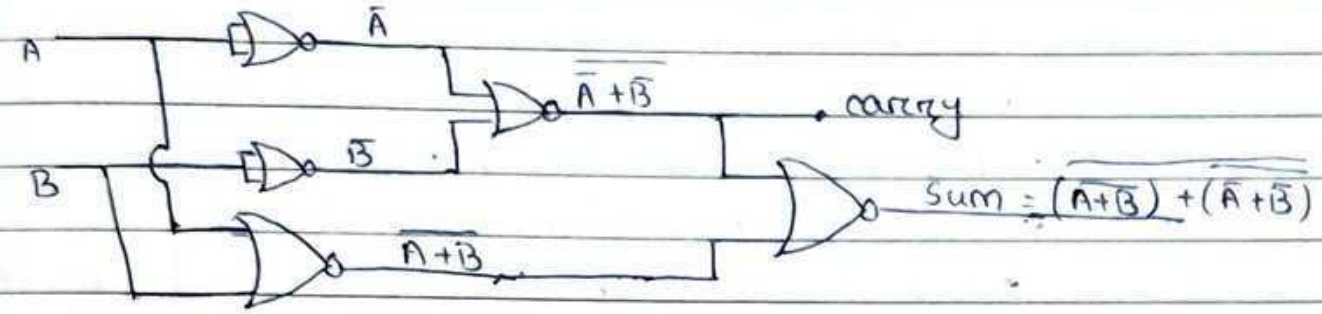


(Logic diagram of half adder using only NAND gate).

Implementation half adder by using only NOR gates :-

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





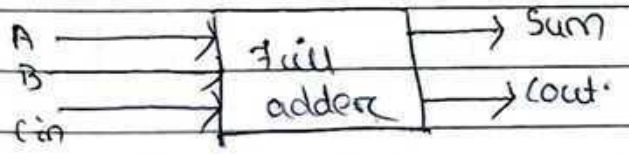
Full adder :-

→ It is a combinational ckt which is used to add 3 binary digits or bits.

→ The A & B are the present i/Ps and the 3rd i/P is coming from the previous addition, which is called as carry and it is denoted as C_{in}.

→ It has 2 o/Ps i.e Sum and Carry (C_{out}).

→ The block diagram and truth table of full adder is given below :

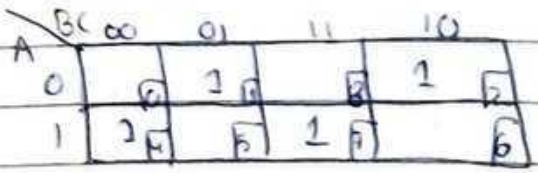


(fig (a) Block diagram of full adder)

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(fig (b) truth table of full adder)

→ The expression for the sum can be expressed by using the K-map as



Sum = $\Sigma(1,2,4,7)$

$= \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

$= \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + \bar{A}\bar{B}C_{in} + ABC_{in}$

$= \bar{C}_{in}(\bar{A}B + A\bar{B}) + C_{in}(\bar{A}\bar{B} + AB)$

$= \bar{C}_{in}(A \oplus B) + C_{in}(\overline{A \oplus B})$ $\quad [\because \bar{A}B + A\bar{B}]$

$= A \oplus B \oplus C_{in}$

→ The expression for Cout by using K-map can be represented as.

A \ BC	00	01	11	10
0			1	
1		1	1	1

011 101 111 .
111 111 110

$C_{out} = AC_{in} + \bar{B}C_{in} + AB$

(minimum expression using K-map)

→ The Cout without min^m expression can be represented as minterm of

$C_{out} = \Sigma(3,5,6,7)$

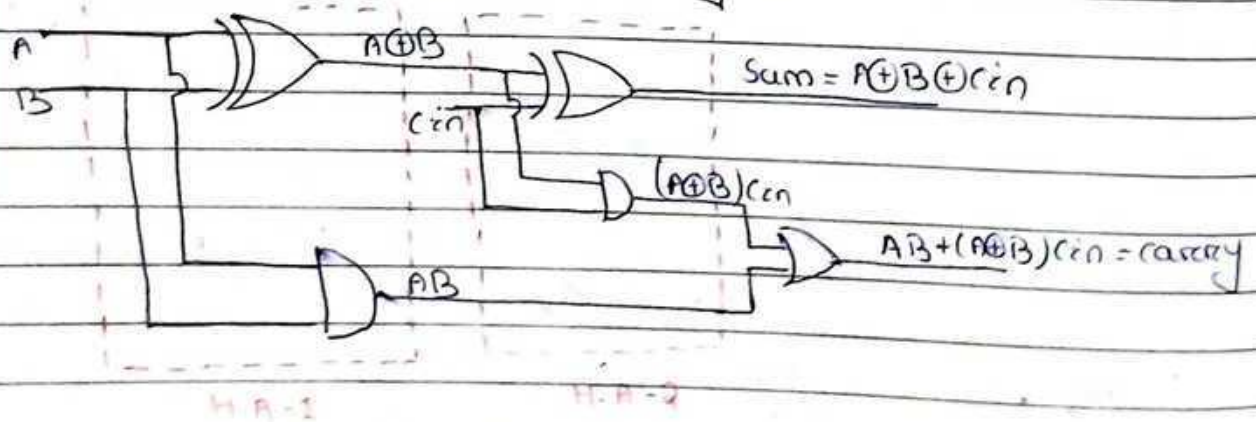
$= \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$

$= C_{in}(\bar{A}B + A\bar{B}) + AB(\bar{C}_{in} + C_{in})$

$= AB + (A \oplus B)C_{in}$.

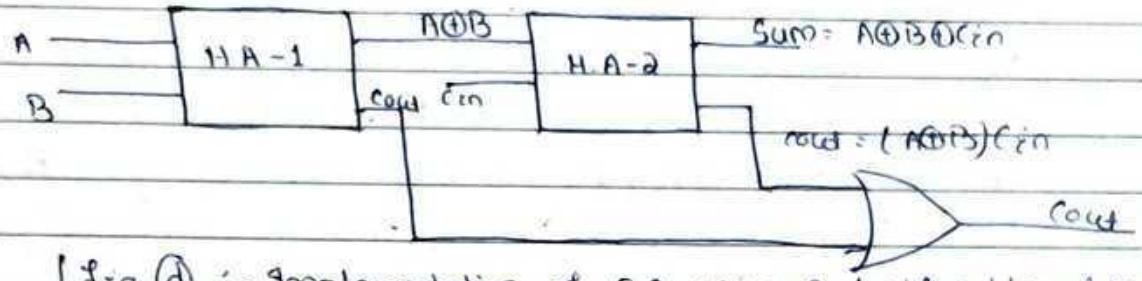
Design the logic circuit of the full adder for the sum and carry :-

Sum = $A \oplus B \oplus C_{in}$, Carry = $AB + (A \oplus B)C_{in}$



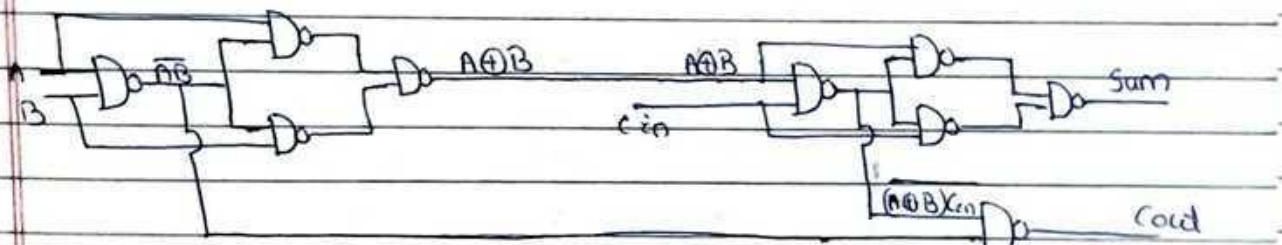
(fig ① - logic circuit of F.A)

→ From the above logic ckt it is analyze that 2 half adders & a OR gate is used to implement the full adder ckt.



[fig (a) :- Implementation of F.A using 2 half adder & OR gate]

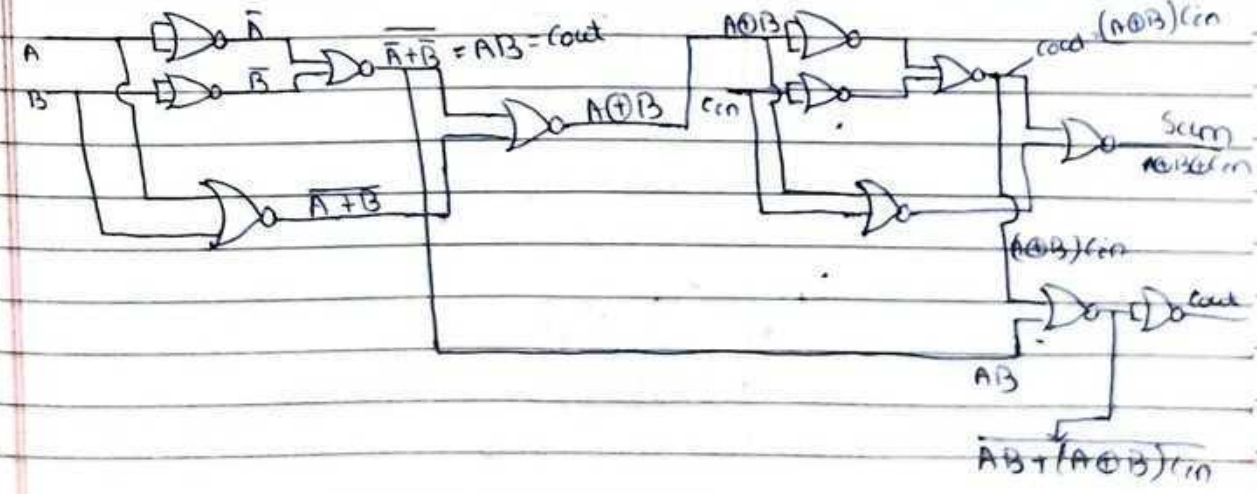
Implementation of full adder by using NAND gate :-



$$\text{Cout} = AB + (A \oplus B)C_{in}$$
$$= \overline{\overline{AB} + (A \oplus B)C_{in}}$$
$$= \overline{\overline{AB} \cdot \overline{(A \oplus B)C_{in}}}$$

$$\text{Cout} = \overline{\overline{AB} \cdot \overline{(A \oplus B)C_{in}}}$$
$$= AB + (A \oplus B)C_{in}$$

Implementation of full adder by using NOR gates :-



Half Subtractor :-

→ It is a combinational ckt which is used to subtract 2 binary digits ea bits.

→ It has 2 i/ps and 2 o/ps i.e difference and borrow.

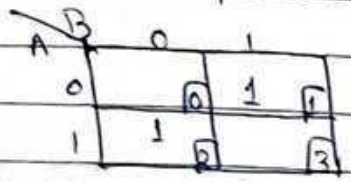
→ As it has 2 i/ps, so it has 4 possible combination.

→ The block diagram, truth table and logic ckt of half subtractor is shown in the below fig.

(fig @. block diagram of H.S)

A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

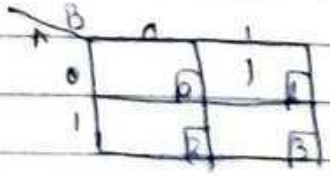
→ The expression for the Difference, Borrow using k-map can be represented as :-



$Diff = \Sigma(1, 2)$

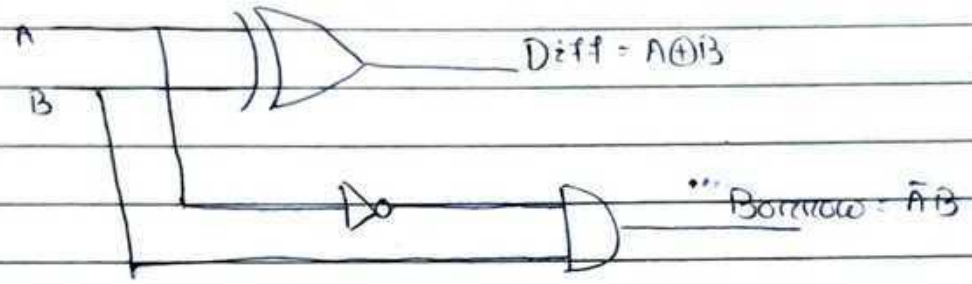
$= \bar{A}B + A\bar{B}$

$= A \oplus B$



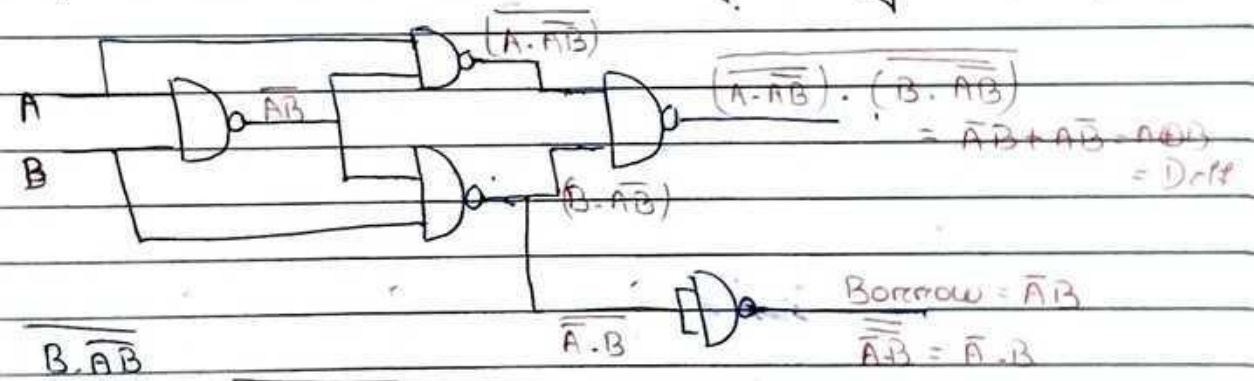
Borrow = $\bar{A}B$

Logic Circuit :-



(fig ① H.S logic circuit)

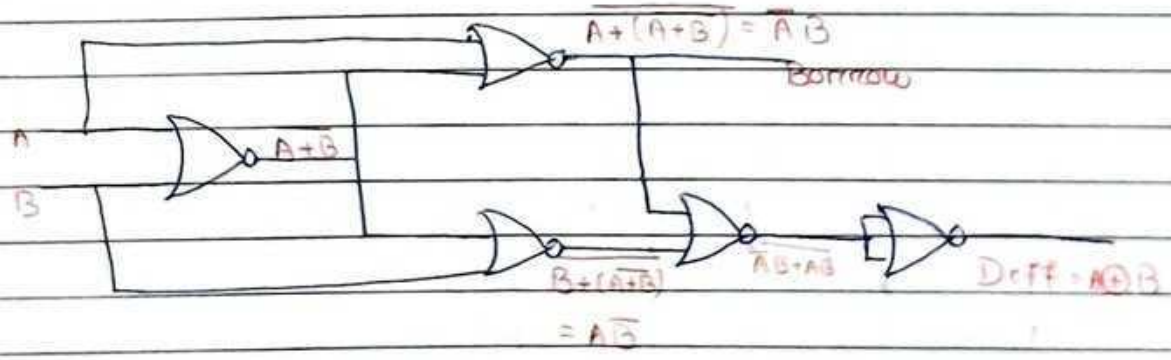
Implementation of Half Subtractor using NAND gate :-



$\overline{B.\overline{AB}}$

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

Implementation of Half Subtractor using NOR gate :-



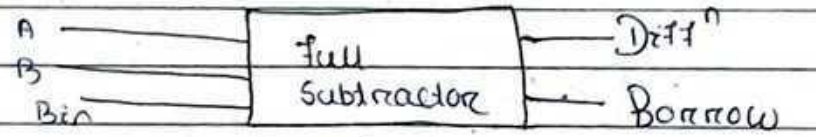
Full Subtractor :-

-> It is a combinational ckt which is used to subtract the 2 binary digits or bits and a borrow which is generated from the previous bit.

-> It has 3 i/p & 2 o/p where the o/p s. are represented as Difference and Borrow out (Bout).

-> As it has 3 i/p the possible combination are 8.

-> The truth table block diagram & logic ckt of full Subtractor is given below.



(Fig@ Block diagram of F.S)

A	B	Bin	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(Fig(b) Truth table of F.S)

-> By using the truth table the expression for the diff & borrow can be represented as;

$$Diff = \Sigma(1,2,4,7)$$

$$= \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$$

$$= \bar{A}B\bar{B}_{in} + \bar{A}\bar{B}B_{in} + AB\bar{B}_{in} + A\bar{B}B_{in}$$

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

$$= (A \oplus B)\bar{B}_{in} + (\overline{A \oplus B})B_{in}$$

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

$B_{out} = \Sigma(1,2,3,7)$

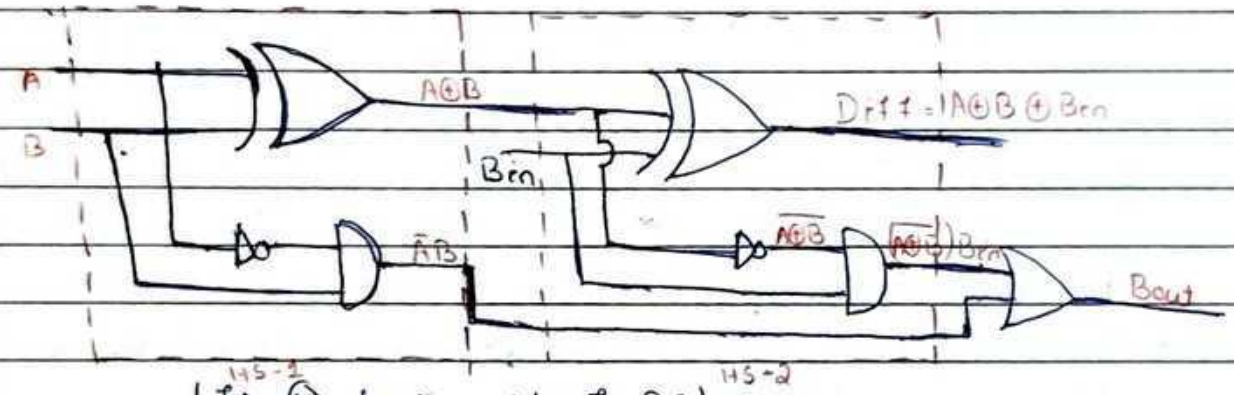
$= \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + \bar{A}BB_{in} + ABB_{in}$

$= \bar{A}\bar{B}B_{in} + ABB_{in} + \bar{A}B\bar{B}_{in} + \bar{A}BB_{in}$

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

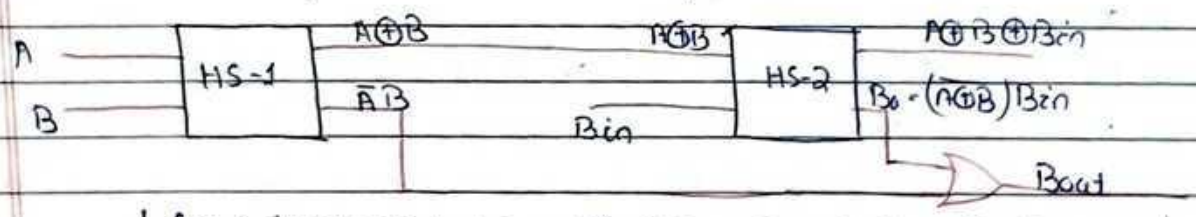
$= B_{in}(\overline{A \oplus B}) + \bar{A}B$

Logic circuit for F.S :-



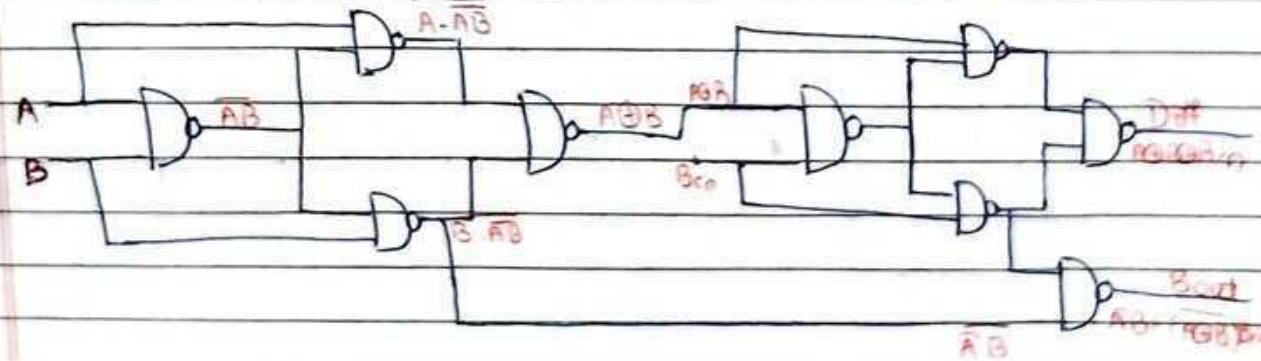
(Fig (b) logic ckt of F.S)

→ From the above ckt it is observed that a full subtractor can be implemented by using 2 half subtractor & a OR gate & the block diagram of this is given below,



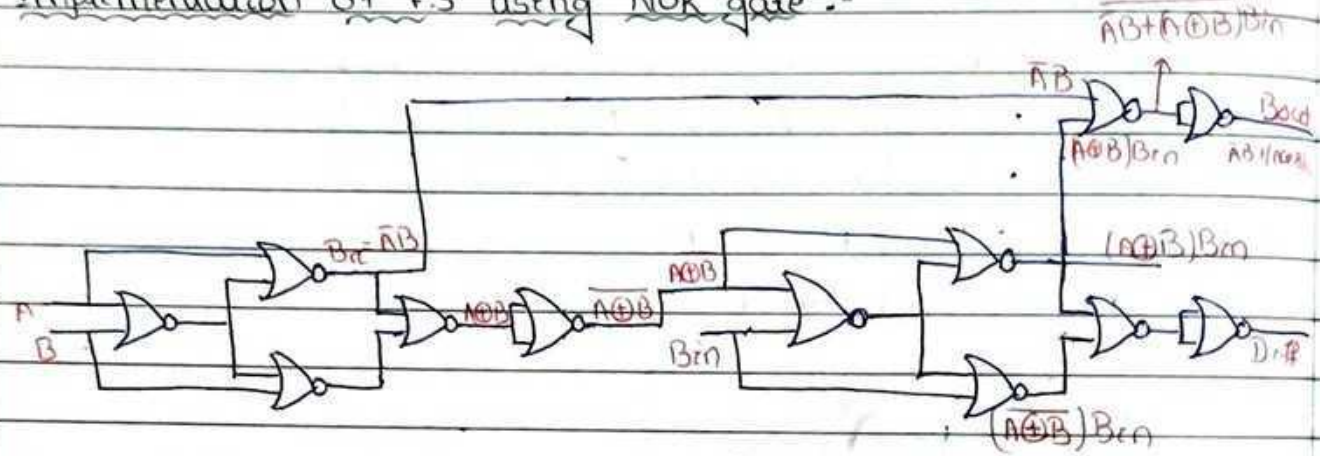
(Fig: Implementation of F.S using 2 H.S & OR gate)

Implementation of F.S using NAND gates :-



(Fig Implementation of F.S by using NAND gate)

Implementation of F.S using NOR gate :-



(fig: Implementation of F.S using NOR gate)

Parallel Binary adder :-

-> It is a combinational ckt which is used to add the 2 binary no.s (4bit, 8bit etc), simultaneously or parallely.

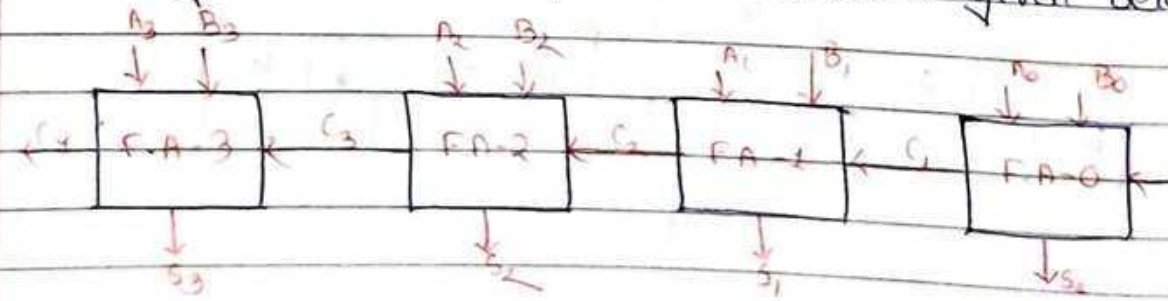
-> To design a 4bit parallel adder 4 full adder ckts are required & they are connected in series.

-> Initially, the carry is assumed as zero, during the addition of LSB.

-> As the F.A ckt are used to design the parallel added, it will add the 3 bits where the 2 bits are the i/p s and the 3rd bit is the carry generated from the previous addition.

-> To design N-bit parallel adder ; 'N' no. of full adder ckt are require, and they are connected in series or cascade.

-> The diagram for 4 bit parallel adder is given below.



(fig: 4-bit parallel Binary adder ckt)

→ It is also called as ripel carry adder, as the adder ckt depends on previous bit to add the present bit.

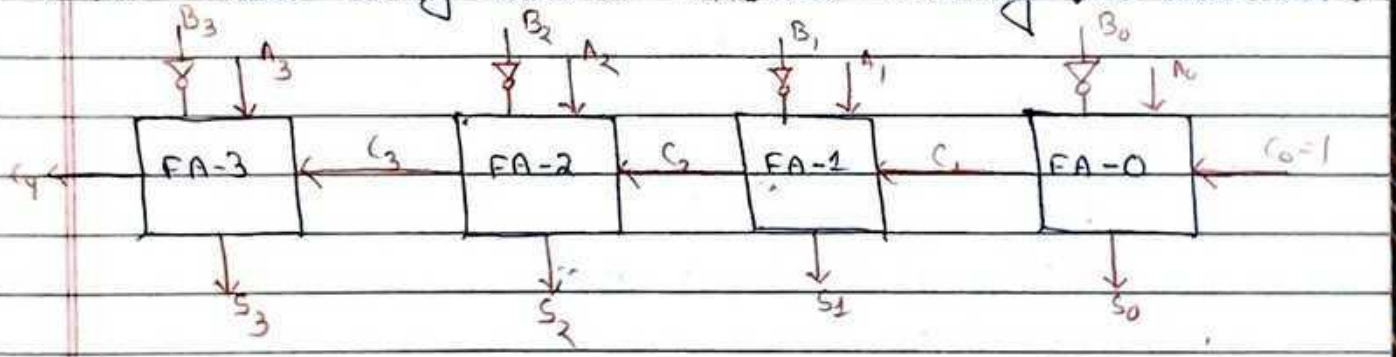
→ The parallel Subtract ckt can be design by N-no. of F.S. But the F.A and the separate F.S ckts which are used for the addition, the Subtraction increases the completion, requires no. of logic gates and also increase the cost also.

→ So, to overcome these limitations, instead of a F.S, the same adder ckt is used to do the Subtraction.

→ The -ve nos are represented in the 2's complement form and after that it will be added with the next no. to get the Subtraction result.

→ For the 2's complement at first the no. should be represented in the 1's complement at then add by 1.

→ The final carry will be involved during the addition.

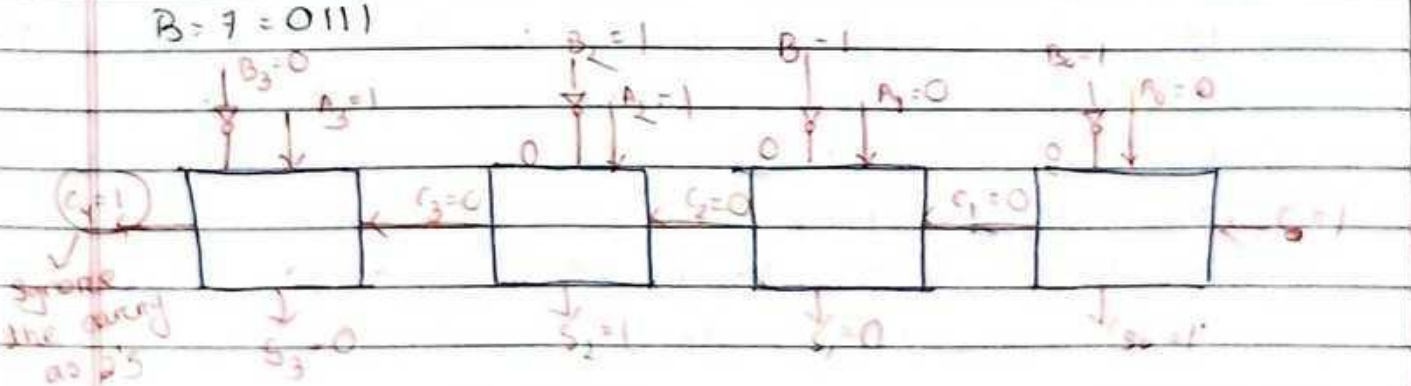


[Fig-4 bit parallel Subtractor or Substraction using adder ckt]

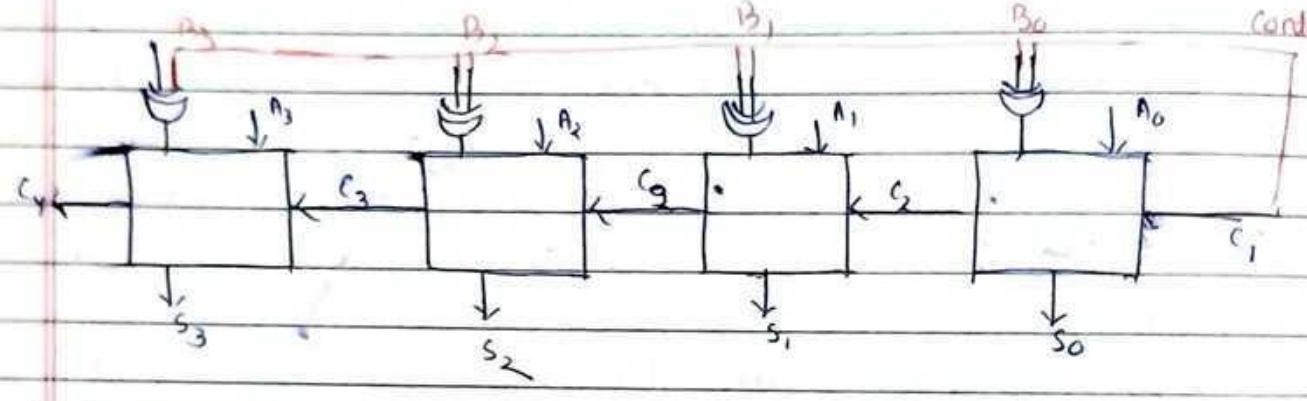
(1) 12 - 7 = 12 + (-7)

A = 12 = 1100

B = 7 = 0111



4 bit adder / Subtractor circuit :-



→ The control ckt is used to identify the adder and subtractor ckt, when the control line = 0, the ckt act as added ckt, when the control line = 1, then ckt is used as a subtractor. So only one ckt is used for additional and subtractor.

12/09/2025

Comparator :-

→ It is a combinational ckt which is used to compare the magnitude of 2 binary numbers.

→ It is divided into 2 types :-

i) Equality comparator

ii) Magnitude comparator.

i) Equality comparator :-

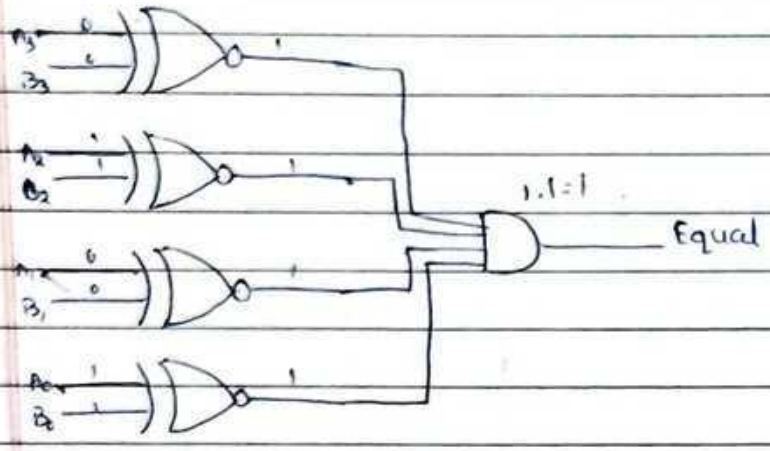
→ It compares the 2 no.s, provides the o/p whether they are equal or not.

→ Let us consider the 4 bit binary no.s are present where it is represented as $A(A_3\ A_2\ A_1\ A_0)$ and $B(B_3\ B_2\ B_1\ B_0)$.

→ When $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, $A_0 = B_0$ then the o/p of the comparator will be $A = B$. Otherwise $A \neq B$.

→ From the above analysis, it is observed if the bits are coincide then the o/p will $A = B$.

→ The logic circuit of the equality comparator for the 4 bit no. is given below.



Ex-NOR

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

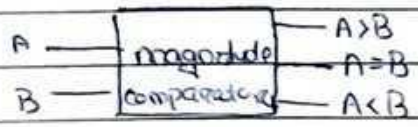
[fig-4 bit equality comparator logic ckt]

→ The logic ckt consist of Ex-NOR gates and a AND gate if the O/P of the AND gate is 1, then it specifice A=B.
→ But when the O/P of AND gate is 0, then it specifice A≠B.
→ The logical expression of equality comparator represented as

$$\text{Equality} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

Magnitude comparator :-

→ It compares the magnitude of 2 binary nos (A & B) & provide the O/P A>B, A<B or A=B.
→ The Block diagram of magnitude comparator is given below



[fig:- B.D of magnitude comparator]

→ It is 2 types.
i) 1-bit Comparator
ii) 2-bit Comparator

classmate
Date
Page

:) 1-bit comparator :-

> It compares 2 1-bit no., can 2 i/ps, we can having 4 possible combination.

> The truth table is given below.

A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

$T_1 = 1$
$F = 0$

-> The logical expression for A=B, A<B and A>B can be derived using the k-map.

For A=B :-

A \ B	0	1
0	1	
1		1

$A=B, \quad \bar{A}\bar{B} + AB = A \odot B = \overline{A \oplus B}$

For A>B :-

A \ B	0	1
0		
1	1	

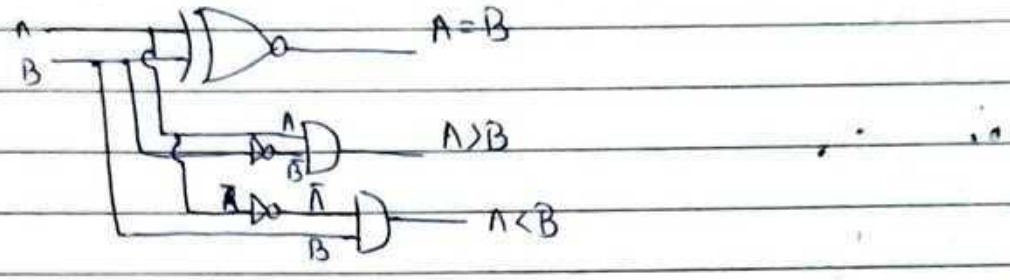
$A>B = A\bar{B}$

For A<B :-

A \ B	0	1
0		1
1		

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

The logic ckt of the 1-bit comparator can be designed as



(Fig: logic ckt of 1-bit comparator)

ii) 2-bit Comparator:-

-> It is used to compare the 2 bits where each no. consist of 2 bits

-> Let us consider the 2 bit nos are considered as A(A_1, A_0) & B(B_1, B_0)

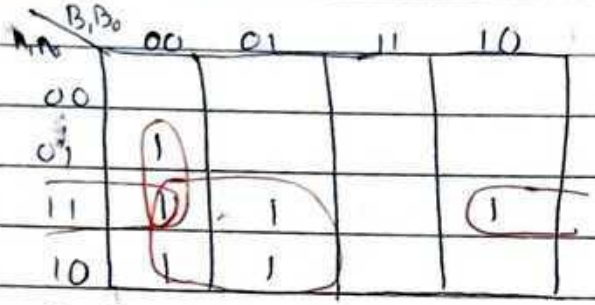
-> For 2 bit no we have 4 diff. bits and total 16 possible combination.

-> The truth table for 2 bit comparator is given below.

A		B		A>B	A=B	A<B
A_1	A_0	B_1	B_0			
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

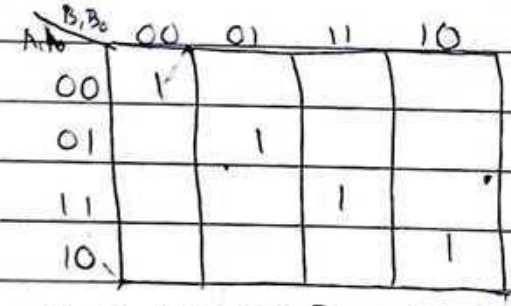
→ The logical expression by using k-map for A=B, A<B, A>B can be derived as

For A>B:-



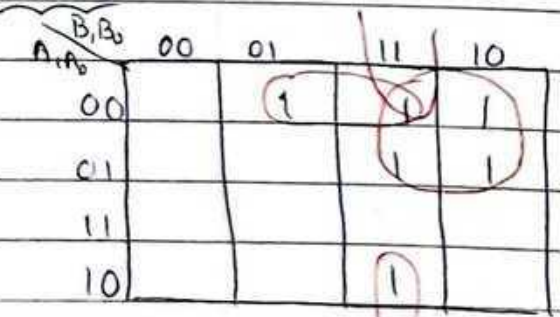
$A>B = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$

For A=B:-



$A=B \Rightarrow \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$

For A<B:-



$A>B = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$

→ The logic ckt of 2-bit comparator given below :-

17/09/2025

Multiplexer :-

→ It is a combinational ckt which is used to convert many i/ps into a single o/p. It is called as MUX. Gr

→ Generally it is represented as $2^n \times 1$ where 2^n = No. of input line and 1 = No of o/p lines and n = No. of Selection lines.

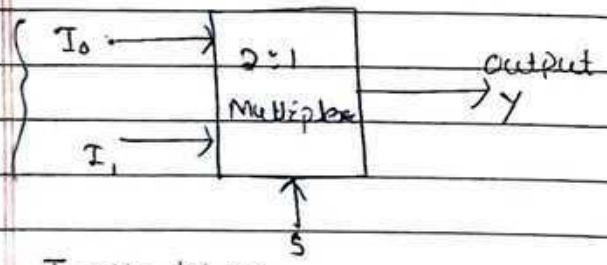
Types of MUX :-

→ It is divided into the following type i.e

i) 2x1 MUX
ii) 4x1 MUX
iii) 8x1 MUX
iv) 16x1 MUX and so on

i) 2x1 MUX :-

→ It has 2 i/p lines and one o/p line It requires only one Selection line to select the i/p as its o/p. The block diagram, Truthtable, logical expression and the logic circuit of 2x1 MUX is given below



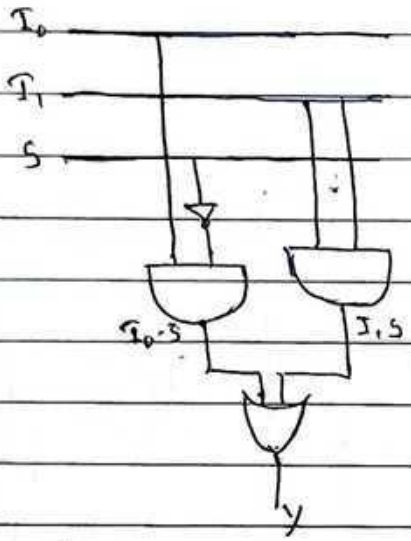
Truth table :-

S	y (o/p)
0	I_0
1	I_1

→ The logical expression from the truth table can be derived as;

$$Y = I_0\bar{S} + I_1S$$

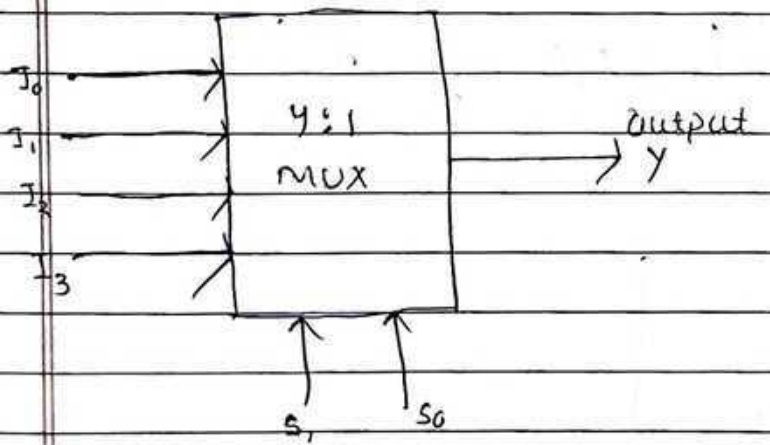
→ The logic ckt of 2x1 mux can be design;



ii) 4x1 MUX :-

-> It has 4 i/p lines and 1 o/p line.

-> To select the 4 i/p lines, it requires 2 selection lines. The block diagram, truth table, logical expression and the logic ckt of 4x1 MUX is given below.



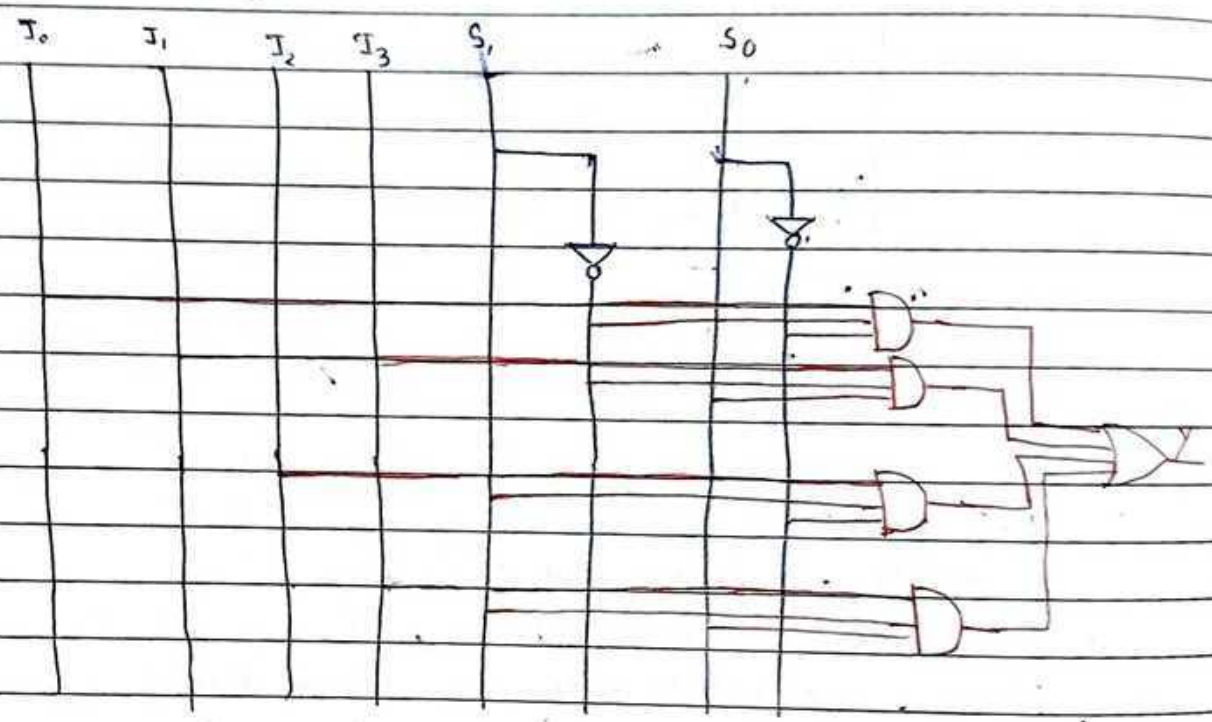
Truth table :-

S_1	S_0	Y(O/P)
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

-> The logical expression from the TT can be derived as;

$$Y = I_0 \cdot \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0$$

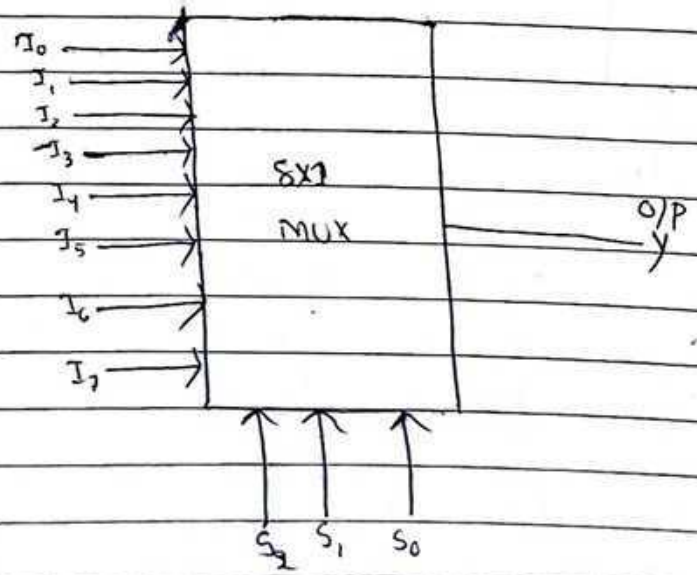
The logic ckt of 4x1 MUX can be designed as;



iii) 8x1 MUX:-

→ It has 8 i/p lines and 1 o/p line as its. It requires 3 selection line to select the i/p & o/p. (S_2 S_1 S_0)

Block diagram



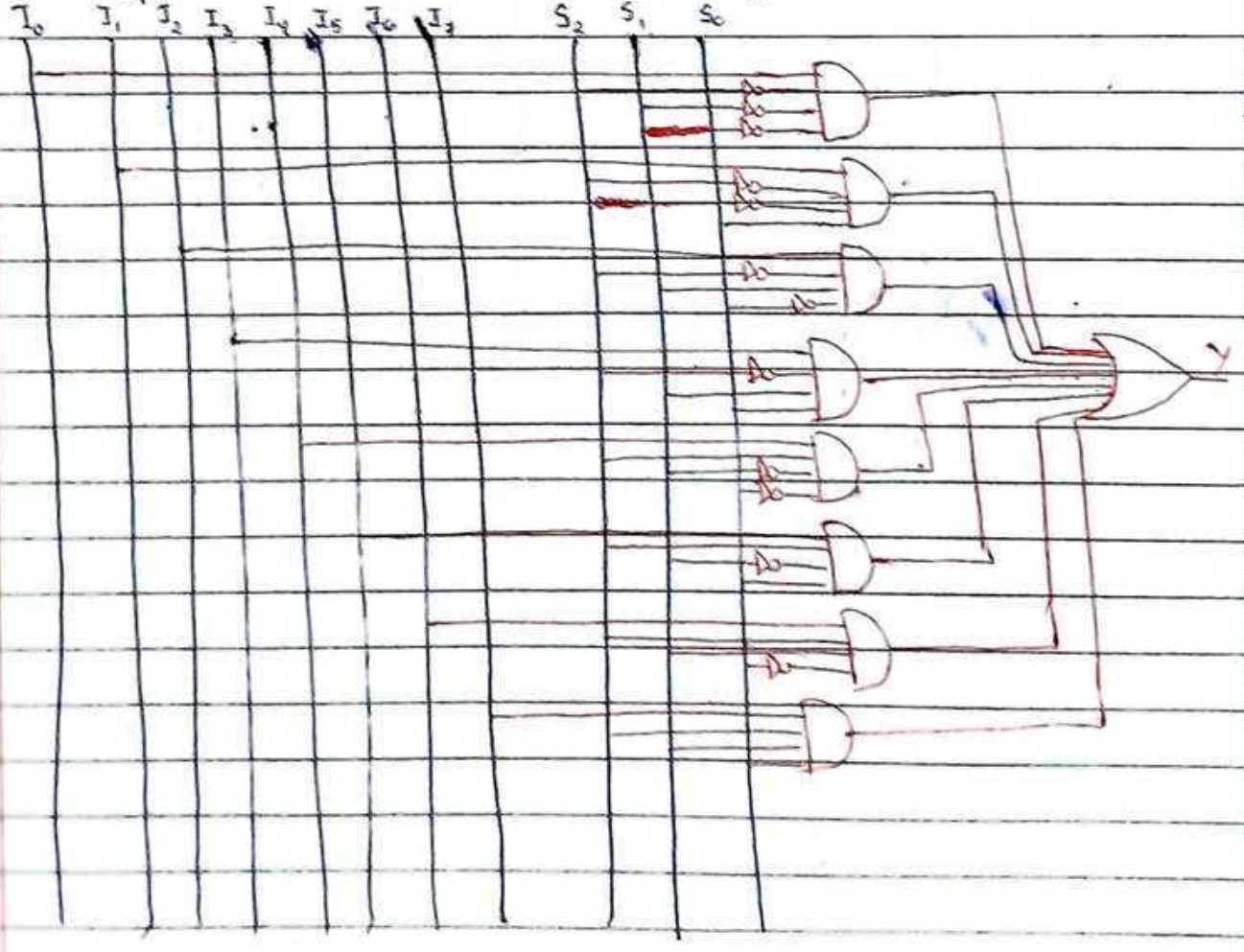
Truth table :-

S_2	S_1	S_0	Y (O/P)
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

The logical expression from the TT can be derived as;

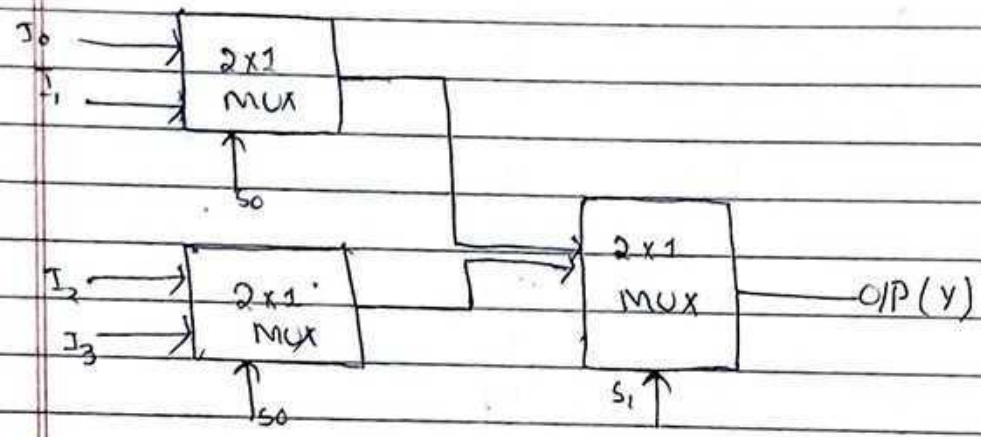
$$Y = I_0\bar{S}_2\bar{S}_1\bar{S}_0 + I_1\bar{S}_2\bar{S}_1S_0 + I_2\bar{S}_2S_1\bar{S}_0 + I_3\bar{S}_2S_1S_0 + I_4S_2\bar{S}_1\bar{S}_0 + I_5S_2\bar{S}_1S_0 + I_6S_2S_1\bar{S}_0 + I_7S_2S_1S_0$$

The logic ckt of 8x1 MUX can be designed as;

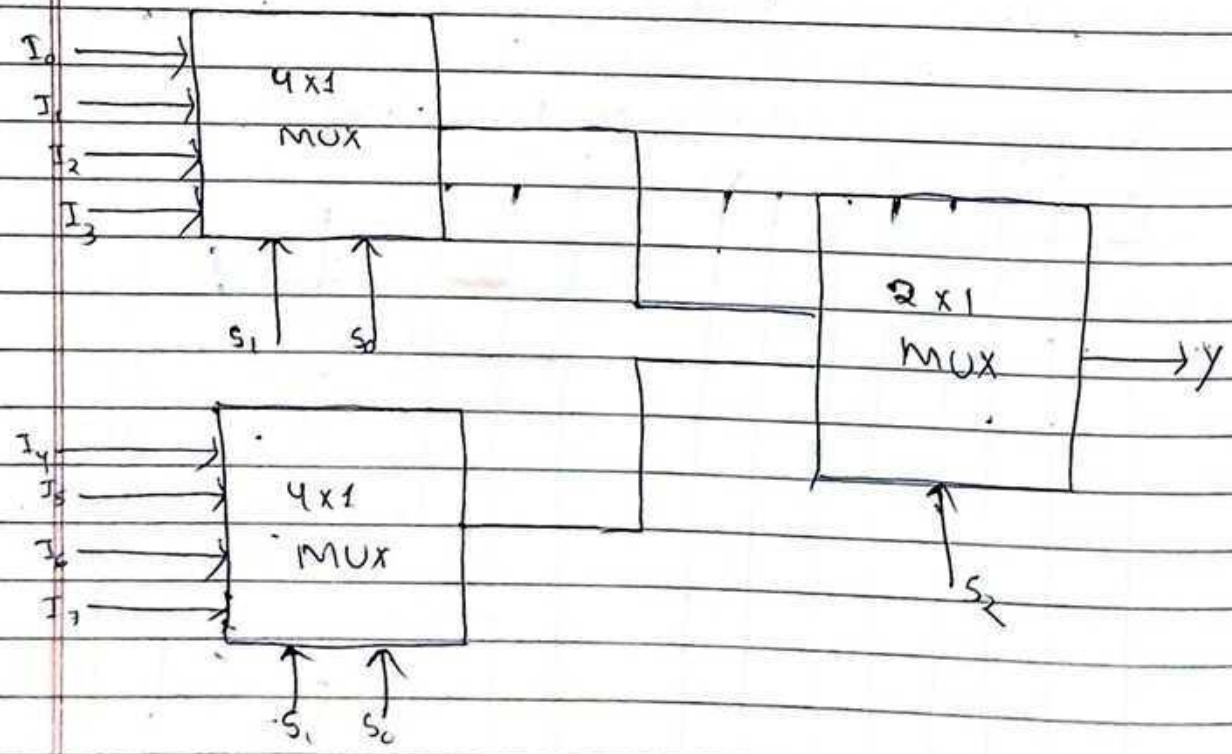


Implementation of Higher order MUX using Lower Order MUX.

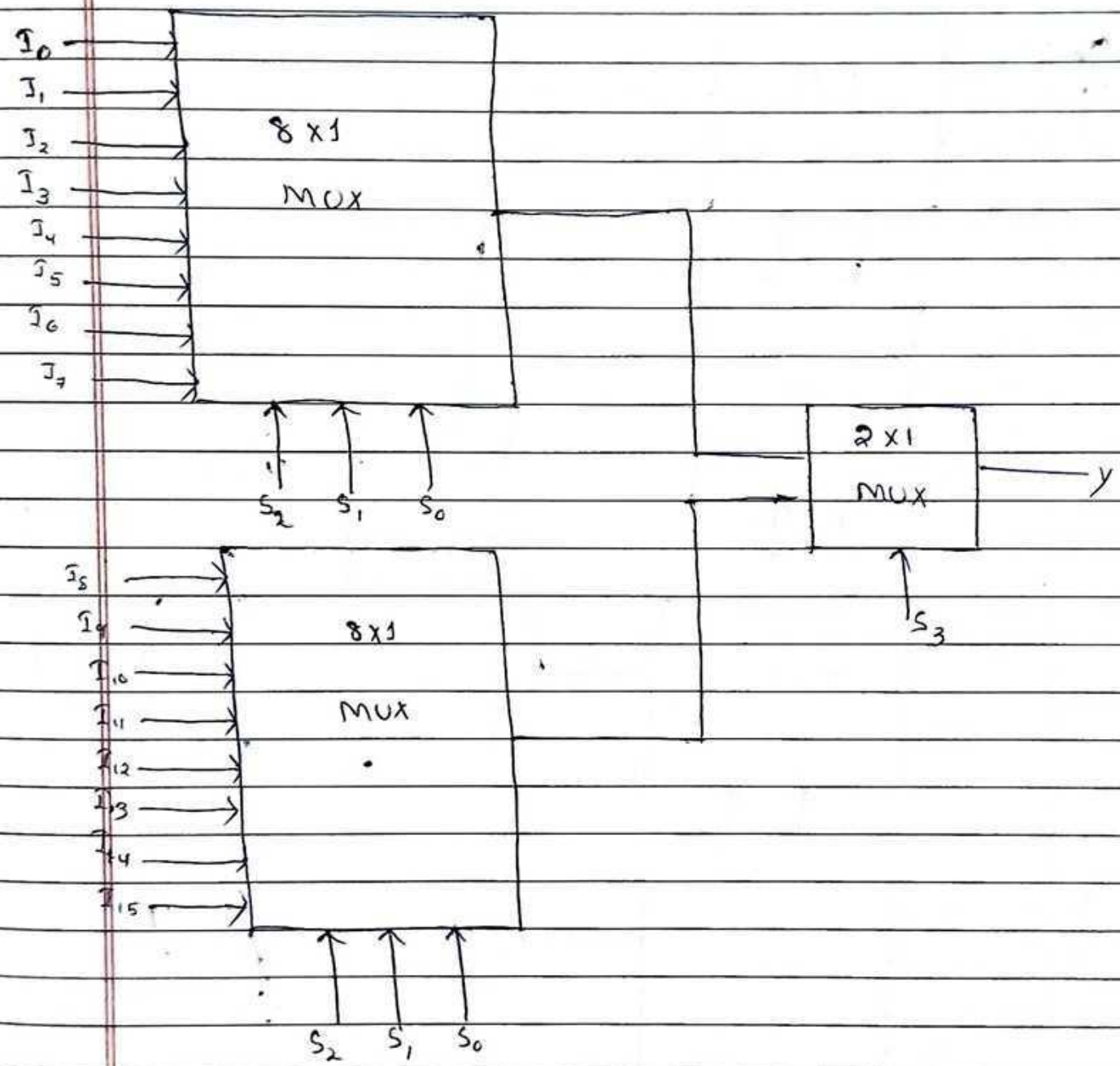
i) 4x1 MUX using 2:1 MUX :-



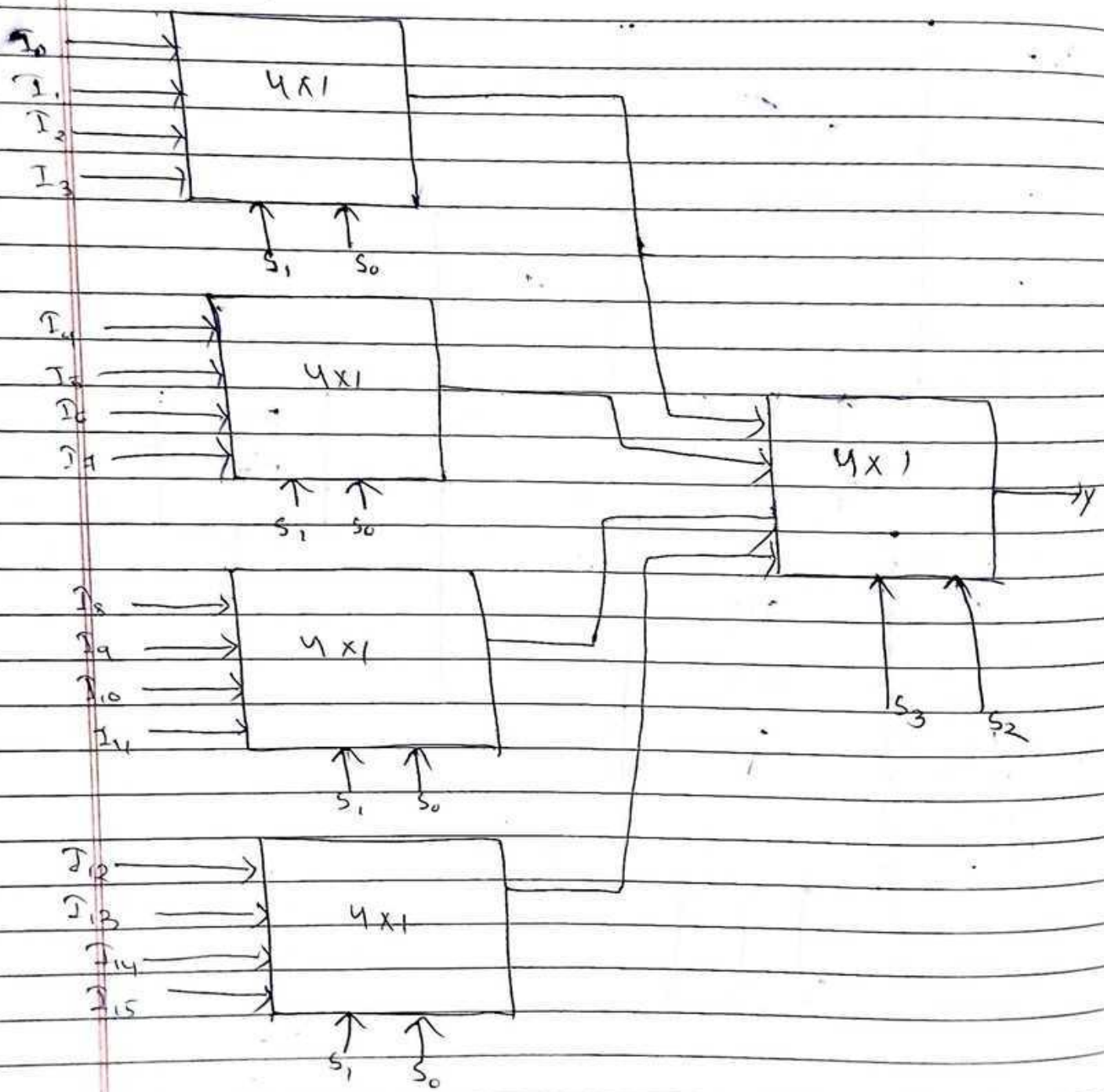
ii) 8x1 MUX using Lower order MUX :-



iii) 16X1 MUX using low order MUX (2 (8x1) and 1 (2x1))



iv) 16x1 using five 4x1 mux :-



Demultiplexers :-

→ It is a combinational ckt which converts single i/p into many outputs.

→ It performs the reverse operation of MUX.

→ It is represented as 1×2^n where:

1 : I/P line

2^n : No. of O/P lines

n : No. of selection lines

Types of DEMUX :-

→ It is divided into the following types :-

i) 1x2 DEMUX

ii) 1x4 DEMUX

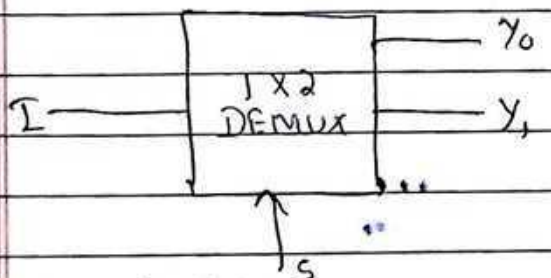
iii) 1x8 DEMUX

iv) 1x16 DEMUX and so on

i) 1x2 DEMUX :-

→ It has 1 i/P line and 2 O/P lines. It requires only 1 selection line to seled the i/P as its O/P.

Block Diagram :-



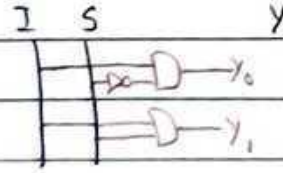
Truth table :-

S	Y_1	Y_0
0	0	I
1	I	0

The logical expression from TT can be derived as

$Y_0 = \overline{S} \cdot I$

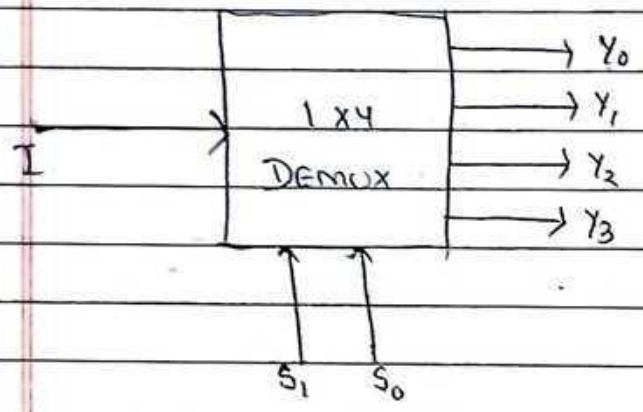
$Y_1 = S \cdot I$



ii) 1 x 4 DEMUX :-

→ It has only 1 i/p line & 4 o/p lines. It requires 2 selection lines.

Block Diagram :-



Truth table :-

S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

Logical expression :-

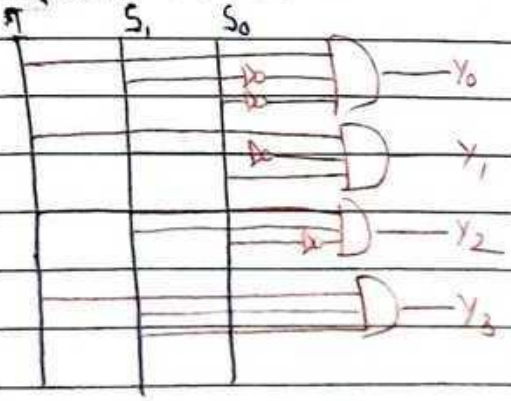
$Y_0 = I . \bar{S}_1 \bar{S}_0$

$Y_1 = I . \bar{S}_1 S_0$

$Y_2 = I . S_1 \bar{S}_0$

$Y_3 = I . S_1 S_0$

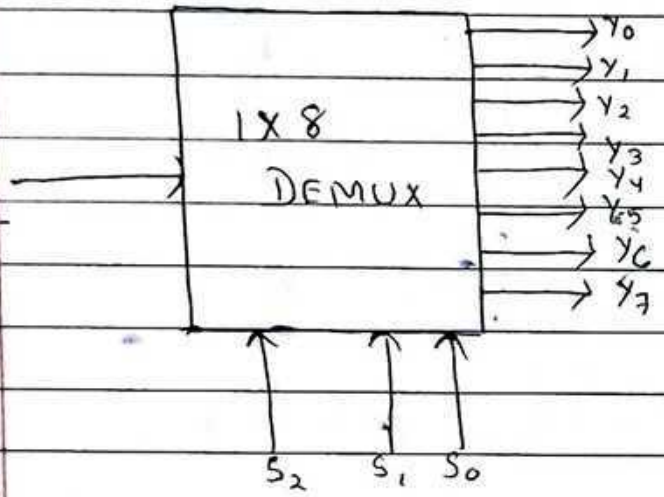
Logical Circuit :-



iii) 1x8 DEMUX :-

→ It has only 1 i/p line and 8 o/p lines. It requires 3 selection lines.

Block Diagram :-



Truth table :-

S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	I	0	0	0	0	0	0	0
0	0	1	0	I	0	0	0	0	0	0
0	1	0	0	0	I	0	0	0	0	0
0	1	1	0	0	0	I	0	0	0	0
1	0	0	0	0	0	0	I	0	0	0
1	0	1	0	0	0	0	0	I	0	0
1	1	0	0	0	0	0	0	0	I	0
1	1	1	0	0	0	0	0	0	0	I

Logical expression :-

$Y_0 = I\bar{S}_2\bar{S}_1\bar{S}_0$

$Y_1 = I\bar{S}_2\bar{S}_1 S_0$

$Y_2 = I\bar{S}_2 S_1\bar{S}_0$

$Y_3 = I\bar{S}_2 S_1 S_0$

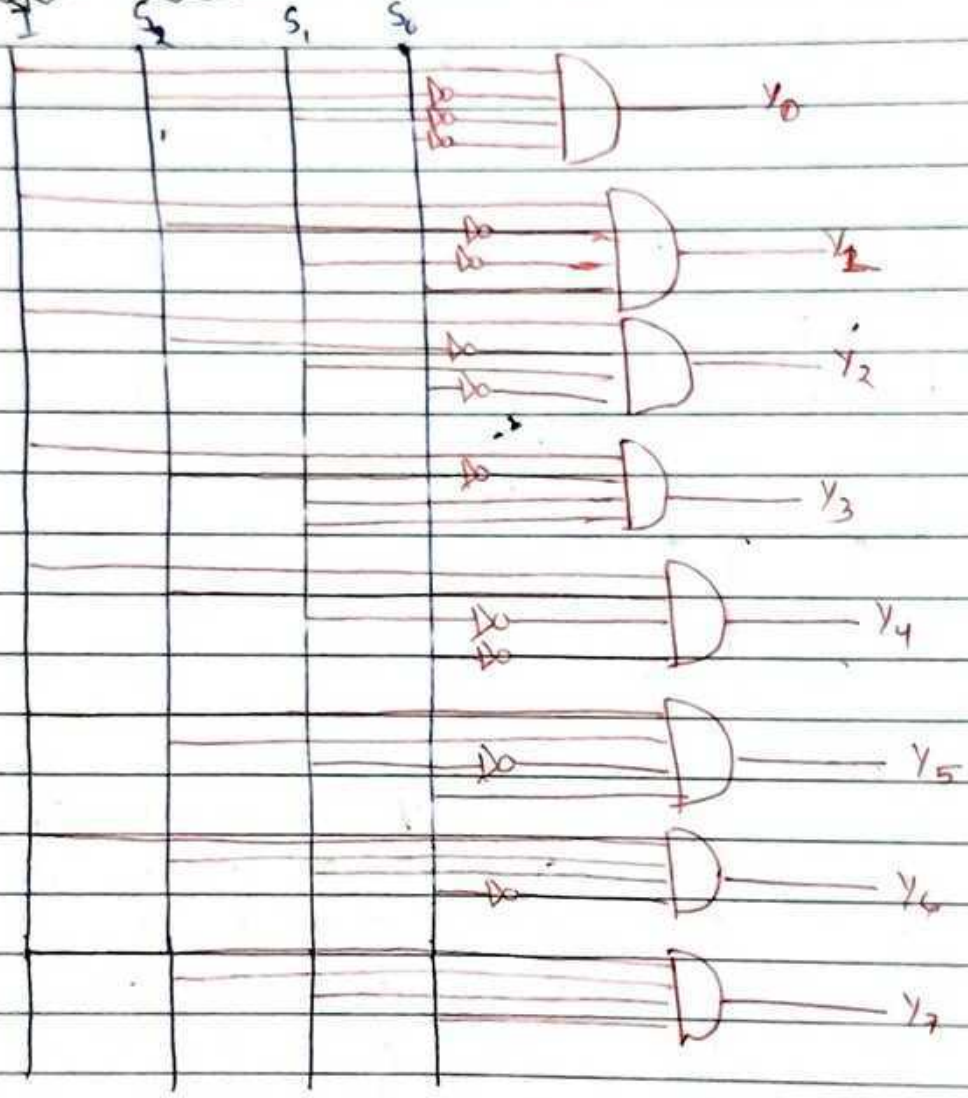
$Y_4 = I S_2\bar{S}_1\bar{S}_0$

$Y_5 = I S_2\bar{S}_1 S_0$

$Y_6 = I S_2 S_1\bar{S}_0$

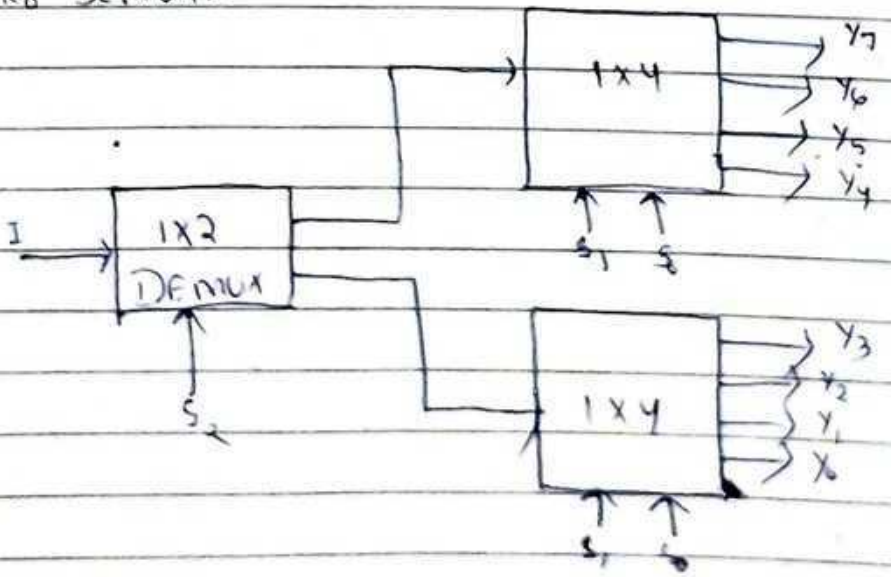
$Y_7 = I S_2 S_1 S_0$

Logical circuit:-

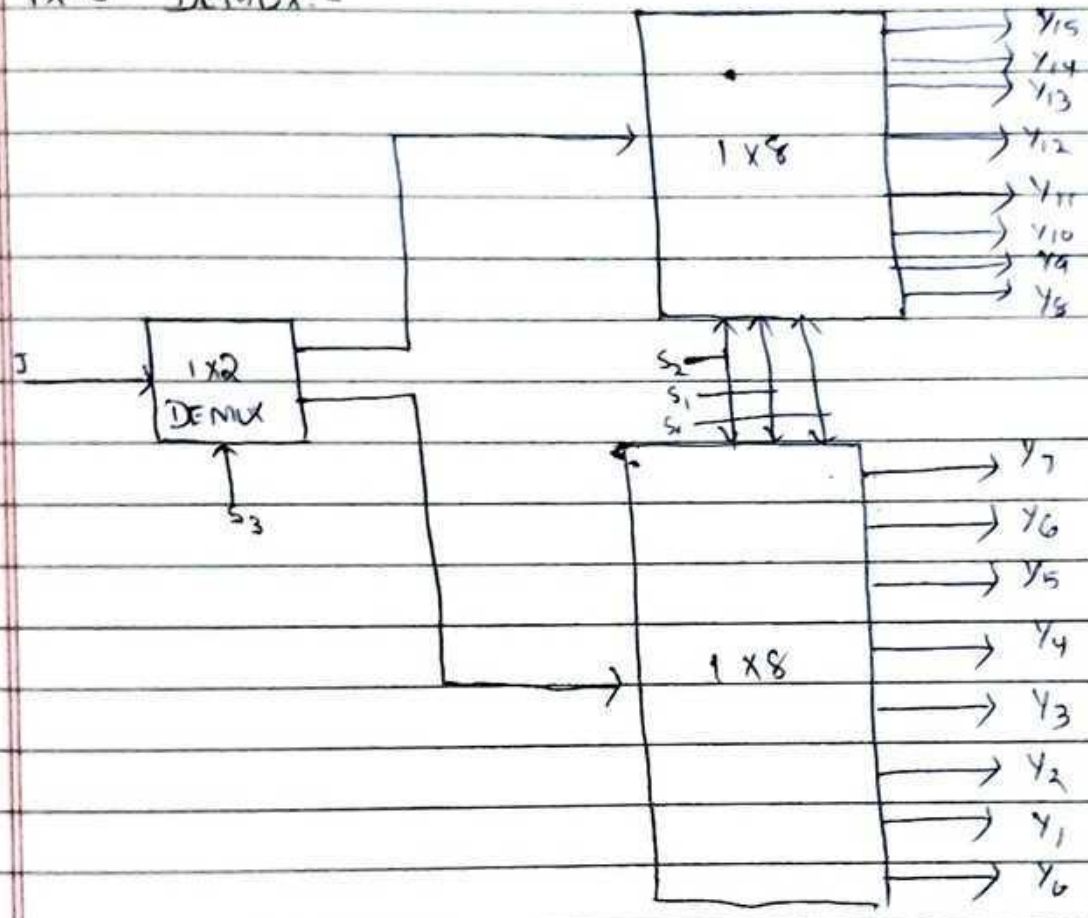


Implementation of Higher-order Demultiplexer:-

1x8 DEMUX:-



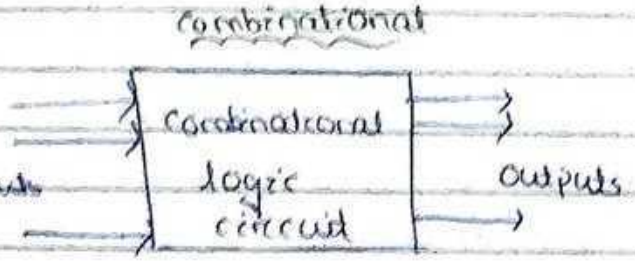
1x16 DEMUX:-

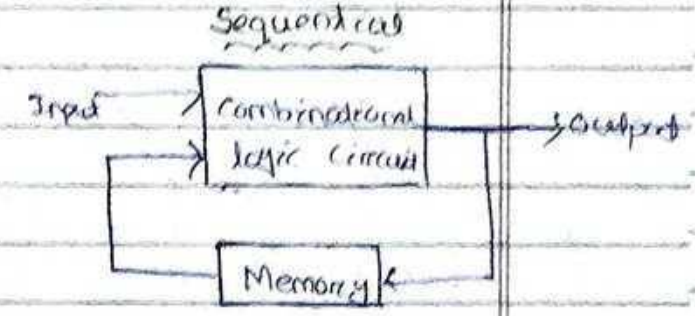

1x8
1x2
DEMUX
1x8
I
S_3
S_2
S_1
S_0
Y_{15}
Y_{14}
Y_{13}
Y_{12}
Y_{11}
Y_{10}
Y_9
Y_8
Y_7
Y_6
Y_5
Y_4
Y_3
Y_2
Y_1
Y_0

Unit-03

: Sequential Circuits :

→ Digital circuits are classified into 2 major categories namely, combinational circuits and sequential circuits.



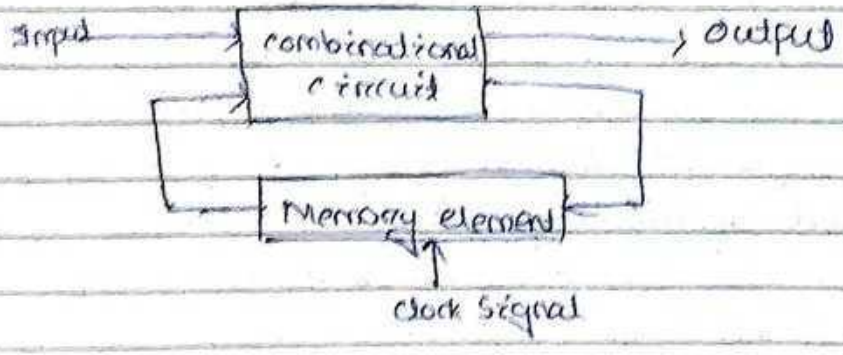


Types of Sequential :-

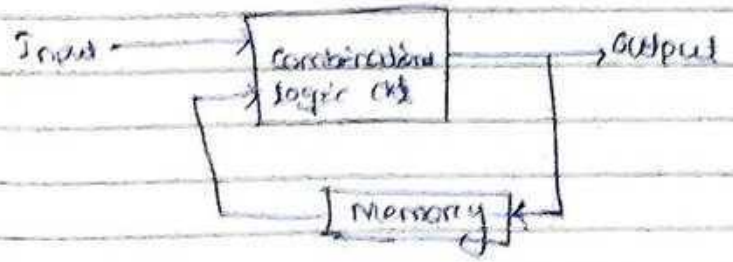
i) Synchronous Sequential ckt

ii) Asynchronous Sequential ckt

i) It requires the clock signal along with i/p signal.



ii) It doesn't depend on the clock signal.



Storage elements :-

→ Storage elements in digital electronics are circuits, primarily latches and flip-flops, that maintain a binary state (0 or 1) until a new input or clock signal directs them to change.

→ Latches are level-sensitive, changing their state when the i/p level meets certain criteria.

→ Flip-flops are edge-triggered means flip flop's state changes only at the exact moment a clock pulse transitions (eg :- from low to high).

What is a Latch ?

→ A latch is an asynchronous sequential ckt whose o/p changes immediately with the change in the applied i/p.

→ It can store 1-bit info.

→ It is used as the fundamental memory element in digital circuits.

→ A latch can have 2 stable states namely, set and reset.

→ The set state is denoted by the logic 1 and the reset state is represented by the logic 0.

→ Due to these 2 stable states, a latch is also known as a bistable-multivibrator.

Types :-

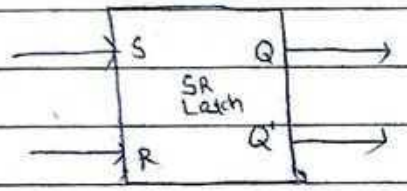
1) SR Latch
2) JK Latch
3) D Latch
4) T Latch

1) SR Latch :-

→ It is a type of latch which has 2 i/p lines designated as S & R

→ where, S represents the Set i/p and R respresents the Reset i/p. Thus, it is also known as Set-Reset Latch.

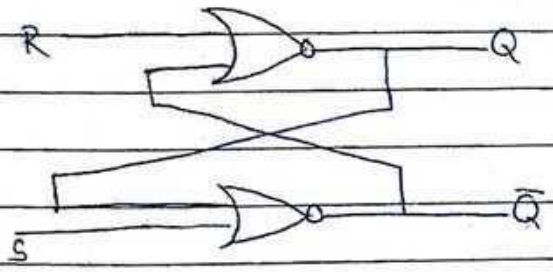
→ The SR latch has 2 stable states namely Set state(S) and Reset state(R).

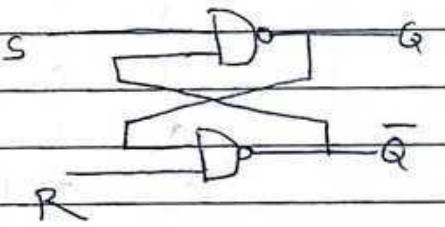


The S i/p sets the o/p(Q) to 1, while the R i/p resets the o/p (Q) to 0.

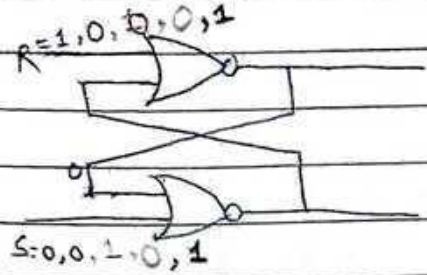
when S=1, Q=1, Set

when R=1, Q=0, Reset





SR Latch using NOR Gates :-



I/P		O/P	
S	R	Q	$\bar{Q}$
0	0	0	1 (Previous stage)
		1	0 (Previous stage)
0	1	0	1 (Reset)
1	0	1	0 (Reset)
1	1	0	0 (Invalid stage or Intermediate State or undefined State)

I/P		output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

If any one of the i/p is H o/p is L.

case-1: S=0, R=1, Q=0, $\bar{Q}$=1

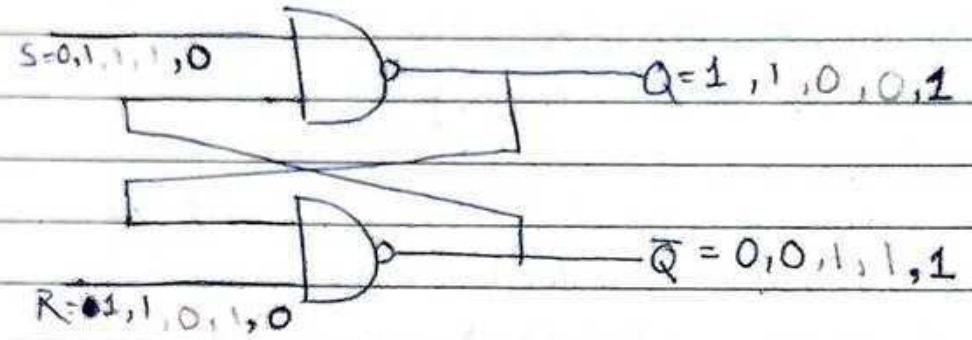
case 2: S=0, R=0, Q=0, $\bar{Q}$=1

case 3: S=1, R=0, Q=1, $\bar{Q}$=0

case 4: S=0, R=0, Q=1, $\bar{Q}$=0

case 5: S=1, R=1, Q=0, $\bar{Q}$=0

SR latch using NAND gates :-



NAND

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

S	R	Q	$\bar{Q}$	
0	0	1	1	(Invalid)
0	1	1	0	(Set)
1	0	0	1	(Reset)
1	1	1	0	(no change)
		0	1	(no change)

Case 1 : S=0, R=1, Q=1, $\bar{Q}$=0

Case 2 : S=1, R=1, Q=1, $\bar{Q}$=0

Case 3 : S=1, R=0, Q=0, $\bar{Q}$=1

Case 4 : S=1, R=1, Q=0, $\bar{Q}$=1

Case 5 : S=0, R=0, Q=1, $\bar{Q}$=1

→ The NAND latch requires a 0 signal to change its state, it is sometimes referred to as an S'R' latch.

24/09/25

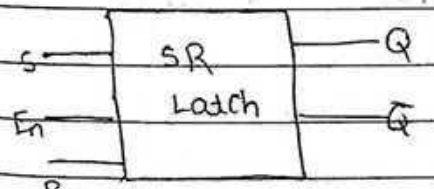
SR Latch with Control Input Signal :-

→ In the basic SR Latch there is a problem of transparency. It means the O/P changes immediately when the I/P changes, and there is no control to change the status of the O/P.

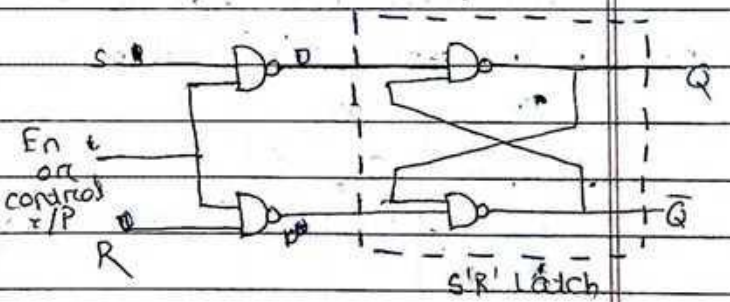
→ So to overcome this drawback or limitation, an extra I/P Signal is required called as control signal or enable signal.

→ When the enable signal is equal to 0 then the latch will maintain the previous stage as its O/P for any I/P variables.

→ But when the enable signal = 1, then it will change its state according to the I/P values. The Block diagram and the logic circuit diagram of SR latch with control I/P signal is given below.



(a) Block diagram of SR Latch with enable Signal)



(b) Logic circuit of SR Latch with En Signal)

→ The truth table of SR Latch with control I/P signal is given below.

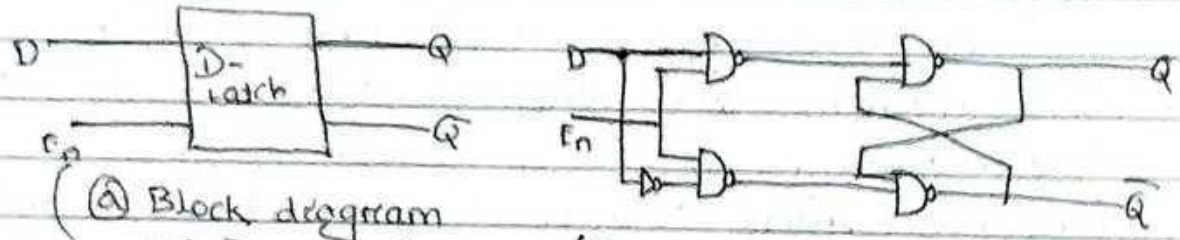
I/PS			O/PS		
E_n	S	R	Q	$\bar{Q}$	
0	X	X	Q	$\bar{Q}$	(Previous stage)
1	0	0	Q	$\bar{Q}$	(Previous stage)
1	0	1	0	1	(Reset)
1	1	0	1	0	(set)
1	1	1	Invalid		

S'	R'	Q $\bar{Q}$
0	0	Invalid
0	1	1 0 (Set)
1	0	0 1 (Reset)
1	1	Q $\bar{Q}$

case-1 :- $E_n=1, S=0, R=0, Q=Q, \bar{Q}=\bar{Q}$
case-2:- $E_n=1, S=0, R=1, Q=0, \bar{Q}=1$
case-3 :- $E_n=1, S=1, R=0, Q=1, \bar{Q}=0$
case-4:- $E_n=1, S=1, R=1$, Invalid

D Latch :-

→ In the SR latch it is observed that when S=1 & R=1, the status is invalid. So to eliminate this status or state, D latch is used. It has 2 states i.e set and reset. There is no invalid state in the D latch. The block diagram, logic ckt and the truth table of D latch is given below.



(a) Block diagram of D-latch) (b) logic ckt of D latch)

→ When $E_n = 0$, the latch is maintaining the previous state and no change in state but when $E_n = 1$, the o/p is same as the i/p in D-latch.

→ The truth table of D-latch is given below.

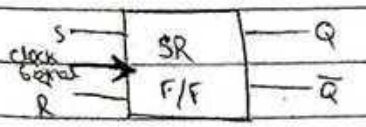
I/P		O/P		
E_n	D	Q	$\bar{Q}$	
0	X	Q	$\bar{Q}$	(Previous stage)
•	•			
1	0	0	1	
1	1	1	0	

Flip-flops :- (F/Fs)

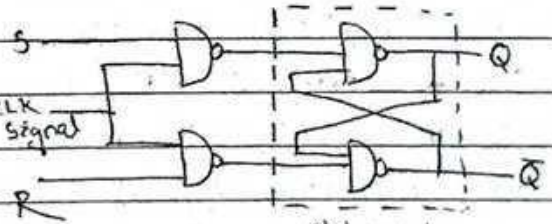
→ In the latch the clock signal is not used whereas in Flip-flop the clock signal is used to change the status of the flip-flop acc to the inputs. It is divided into the following types i.e i) SR F/F
ii) D F/F
iii) JD F/F
iv) T F/F
v) Master slave F/F

i) SR F/F : Using NAND Gate :-

→ It has 2 i/Ps i.e S and R and 2 o/Ps i.e Q and $\bar{Q}$.
→ When Q = 1, it means value 1 will store in the flip-flop and when Q = 0, value 0 will store in the F/F.
→ The Q and $\bar{Q}$ is also represented as Q_n & $\bar{Q}_n$. where
Q_n → Present state
$\bar{Q}_n$ → complement of present state
Q_{n+1} → Next state.



(a) Block diagram of SR F/F)

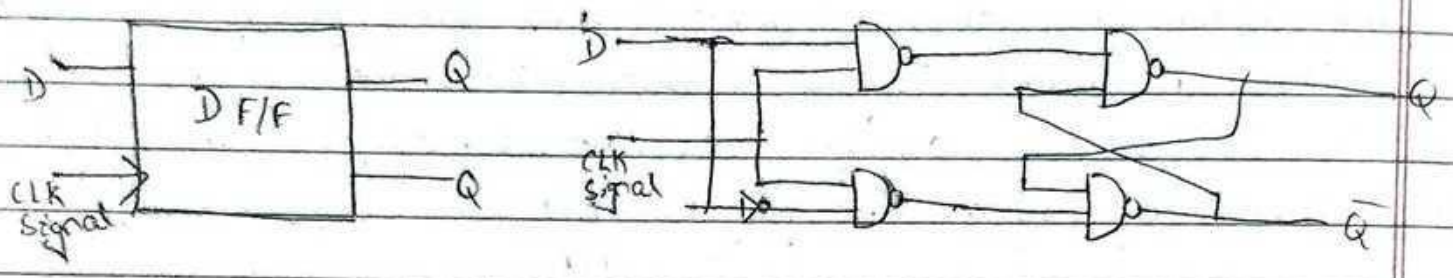


'SR' Latch
(b) Logic ckt of SR F/F)

→ When CLK signal = 0 there will be no change in the flip-flop for any value of S and R but when clock signal = 1, the state will change acc to the values of S & R. The truth table for SR F/F is given below

CLK	S	R	Q	$\bar{Q}$	
0	x	x	Q	$\bar{Q}$	(Previous State)
1	0	0	Q	$\bar{Q}$	"
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	Invalid		

ii) D F/F:-



→ It has 1 i/P i.e D and 2 O/P Q and $\bar{Q}$ and it needs a clk signal to change its state.

Truth table:-

I/P		O/P	
Clk	D	Q	$\bar{Q}$
0	x	Q	$\bar{Q}$ (Previous State)
1	0	0	1 (Reset)
1	1	1	0 (Set)

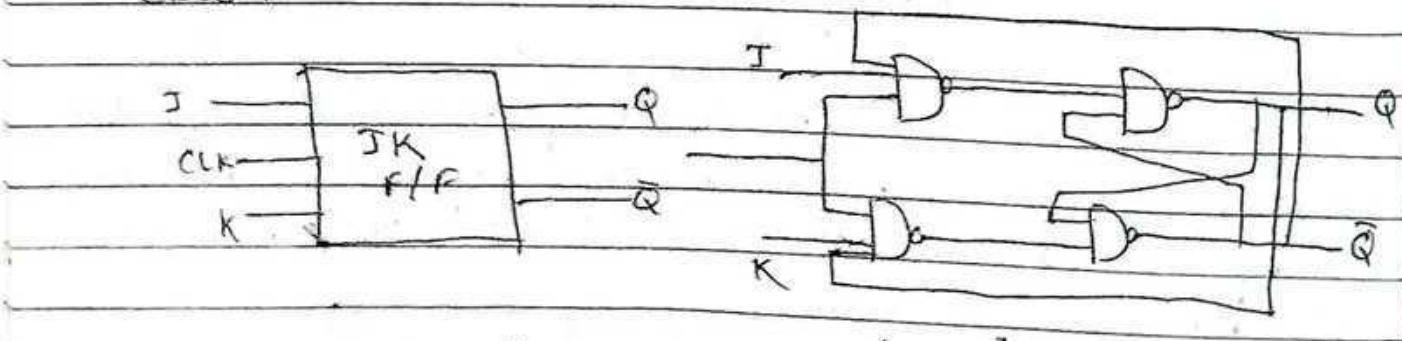
iii) JK F/F:-

→ It has 3 states ie Set, Reset and complement state.

→ The invalid state of SR F/F is overcome in JK F/F.

→ It has 2 i/Ps & 2 O/Ps. It needs the clk signal to change its state according to the i/P values.

→ The block diagram, logic ckt & the truth table is given below.



(J-K F/F)

I/P			O/P		
CLK	J	K	Q	$\bar{Q}$	
0	X	x	Q	$\bar{Q}$	(Previous state)
1	0	0	Q	$\bar{Q}$	"
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	$\bar{Q}$	Q	Complement form

Same as SR

10/10/2025

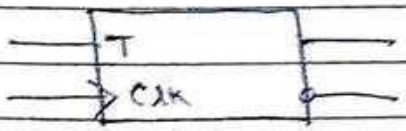
T Flip-Flop :-

→ The T (toggle) F/F is a complementing F/F and can be obtained from a JK F/F when i/ps J and K are tied together.

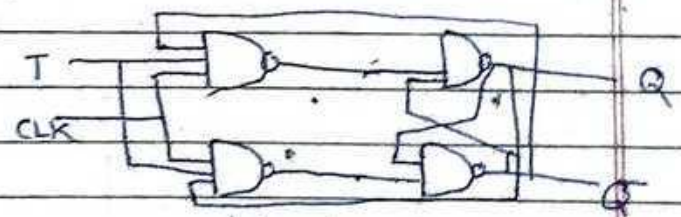
→ When T=0 (J=K=0), a clock edge does not change the O/P.

→ When T=1 (J=K=1), a clock edge complements the o/p.

→ The complementing F/F is useful for designing binary counters.



(a) Graphic Symbol



(logic diagram)

CLK	T	Q	$\bar{Q}$	
0	X	Q	Q'	No change
1	0	Q	Q'	"
1	1	Q'	Q	Complement

Q or Qn → Present state

F/F Analysis :-

→ To analyze the basic operation of any F/F we have to consider the following tables, diagram and eqⁿ.

i) Functional Table
ii) characteristic Table
iii) State Table.
iv) State Diagram
v) characteristic Eqⁿ.
vi) Excitation Table.

SR F/F :-

i) FT of SR F/F :-

CLK	S	R	Q_n	$\bar{Q}_n$	State / operation
↓	*	*			
1	0	0	Q_n	$\bar{Q}_n$	No change State or previous state
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	X	X	Invalid State

ii) CT of SR F/F :-

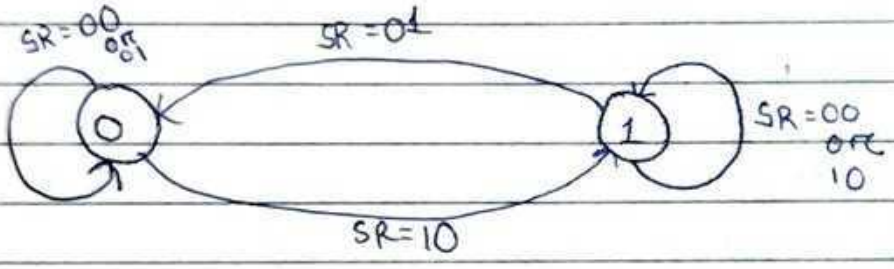
S	R	Q_{n+1}	State or operation
0	0	Q_n	Previous st
0	1	0	Reset
1	0	1	Set
1	1	X	Invalid st

iii) ST of SR F/F :-

I/Ps		Present state	Next state
S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

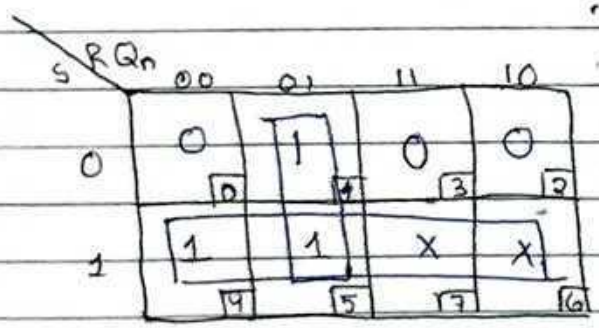
iv) State Diagram :-

→ The state diagram will be drawn by considering the present state (Q_n) and the next state (Q_n) values from the state table.



v) characteristic eqⁿ :-

→ The characteristic eqⁿ of SR F/F can be derived by using the K-map with the variables of state table.



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

vi) Excitation table :-

→ In the excitation table from the present state (Q_n) and the next state (Q_{n+1}) we will derive the expected i/ps of the SR F/F

PS	NS	I/PS	
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

1) F/F :-

i) FT :-

I/P		O/P		State
CLK	D	Q	$\bar{Q}$	
↓	*	Q	$\bar{Q}$	No change
1	0	0	1	Reset
1	1	1	0	Set

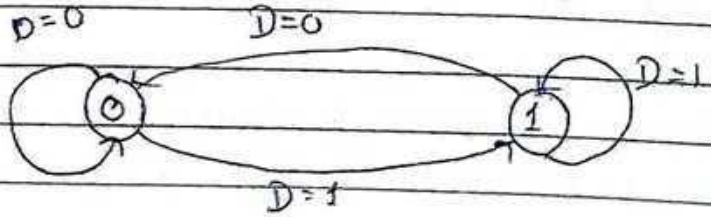
ii) CT :-

D	Q_{n+1}	State
0	0	reset
1	1	set

iii) ST :-

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

iv) SD :-



v) CE :-

Q_n / D	0	1
0	0	0
1	1	1

$Q_{n+1} = D$

vi) ET :-

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

JK F/F :-

i) FT :-

I/P			O/P		
Clk	J	K	Q	$\bar{Q}$	State
1	0	0	Q	$\bar{Q}$	Previous state
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	$\bar{Q}$	Q	complement form

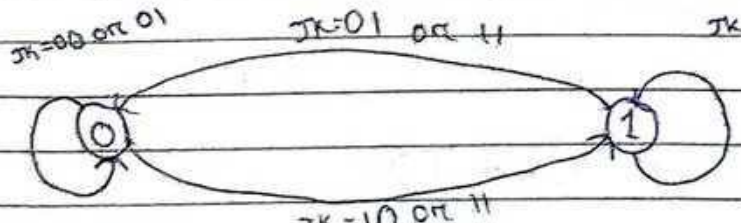
ii) CT :-

J	K	Q_{n+1}	State
0	0	Q	Previous
0	1	0	Reset
1	0	1	Set
1	1	$\bar{Q}$	complem

iii) ST :-

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

iv) SD :-



v)

KQ_n / J	00	01	11	10
0	0	1	0	0
1	1	1	0	1

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

vi) ET:

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

T F/F:-

i) FT:-

Clk	T	Q	$\bar{Q}$	State
1	0	Q	Q'	No change
1	1	Q'	Q	complement

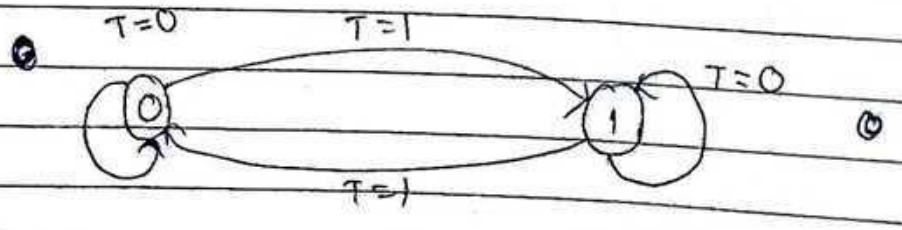
ii) CT:-

T	Q_{n+1}	State
0	Q_n	No change
1	$\bar{Q}_n$	complement

iii) SCT :-

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

iv) SD:-



v) CE:-

TQ_n	0	1
0	0	1
1	1	0

$Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n$

vii) ET:-

Q	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Conversion of F/F :-

→ Convert a source F/F or a given F/F to destination F/F or required F/F and the following steps are required.

1) Write the state table of destination or required F/F.
2) Write the excitation table of the source or given F/F.
3) Re-write the state table of the destination or required F/F acc to the excitation table of the source or given F/F.
4) Draw the k-map and find the expression to convert a source F/F to a destination F/F.
5) Draw the logic diagram acc to the ~~given~~ expression.

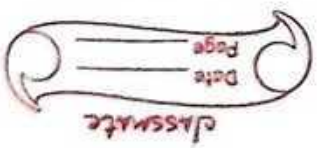
Conversion of SR F/F to JK F/F :-

Step 1: Write the state table of JK F/F

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

Step 2: Write the ~~state~~ excitation table of SR F/F.

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0



Step 3: Rewrite the state table of JK F/F acc. to the excitation table of the SR F/F.

J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Step 4: Draw the K map and find the expression.

Kmap for S:-

J \ KQ_n	00	01	11	10
0	0	X	0	0
1	1	X	0	1

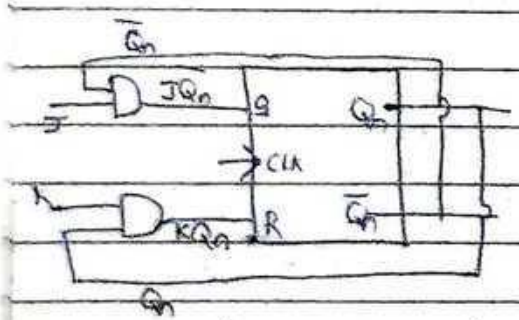
$S = J\overline{Q_n}$

K-map for R:-

J \ KQ_n	00	01	11	10
0	X	0	1	X
1	0	0	1	0

$R = KQ_n$

Step 5 Draw the logic diagram acc to the expression.



(conversion from SR F/F to JK F/F)

Conversion of D F/F to SR F/F :

1) Step 1 : Write the state table of SR F/F .

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

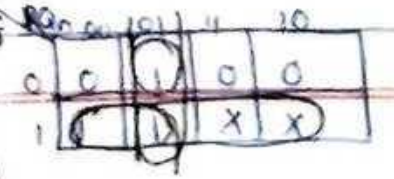
2) Step 2 : Write the excitation table of D F/F :-

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

3) Step 3 : Rewrite the state table of SR F/F arr to the excitation table of the D F/F .

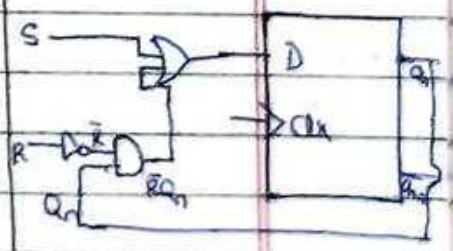
S	R	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	x	x
1	1	1	x	x

4) Draw the kmap and find the expression .



$D = S + \bar{R}Q_n$

Step 5 :

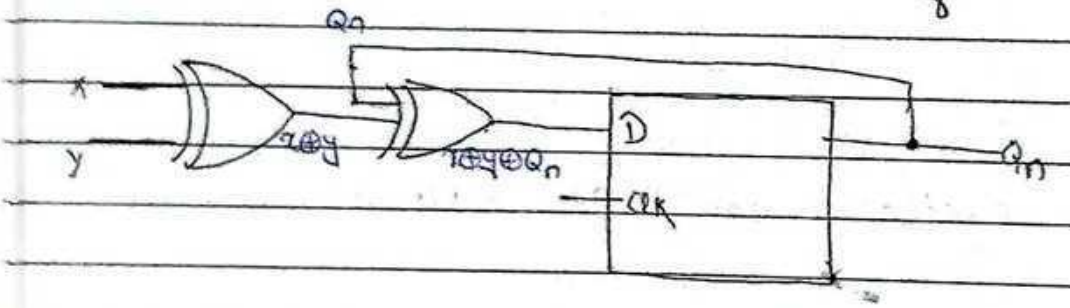


Analysis of clocked sequential circuits :-

For analysis of clocked sequential circuits the following steps are required.

i) Circuit Diagram
ii) State Equations
iii) State table
iv) state Diagram

1) Derive the state table and the state Diagram of the sequential circuit shown in the figure.



→ The circuit diagram has only one F/F i.e D F/F. Acc. to the diagram the i/p to the D F/F can be derived as

$$\boxed{D = x \oplus y \oplus Q_n}$$

→ Earlier we have derive the characteristic eq^n of D F/F which is represented as $Q_{n+1} = D$

where : Q_n → Present State

Q_{n+1} → Next State

• Now Q_{n+1} i.e next state can be expressed as

$Q_{n+1} = D$

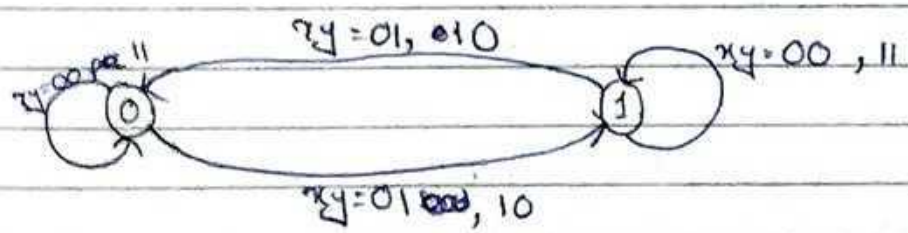
$$\Rightarrow \boxed{Q_{n+1} = x \oplus y \oplus Q_n}$$

State table :-

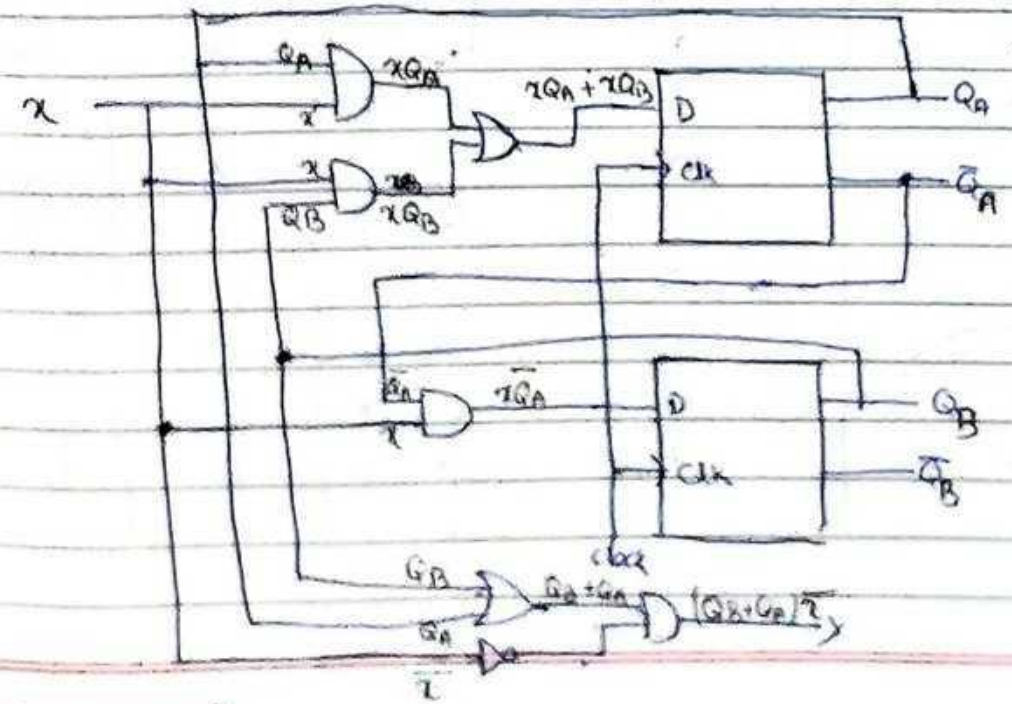
Present state (Q_n)	I/Ps x	I/Ps y	Next state Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$x \oplus y \oplus Q_n$
Same : 0
diff : 1

State Diagram :-



a) Derive the state table and the state diagram of the sequential circuit shown in the figure.



-> In the given ckt there are 2 D F/F ~~are~~ i.e, D_A and D_B. There is an input i.e x and one output i.e y.

-> Acc. to the ckt diagram the i/ps to the F/Fs can be represented as

$$D_A = Q_A \cdot X + Q_B \cdot \bar{X}$$

$$D_B = \bar{Q}_A \cdot X$$

The o/p of y is represented as;

$$y = (Q_A + Q_B) \cdot \bar{x}$$

As we know that the Eq^n of D F/F is represented as

$$\boxed{Q_{n+1} = D}$$

Now the next state values for both the F/Fs can be represented as

$$Q_{A+1} = D \Rightarrow Q_{A+1} = Q_A \cdot X + Q_B \cdot X$$

$$Q_{B+1} = D \Rightarrow Q_{B+1} = \bar{Q}_A \cdot X$$

where

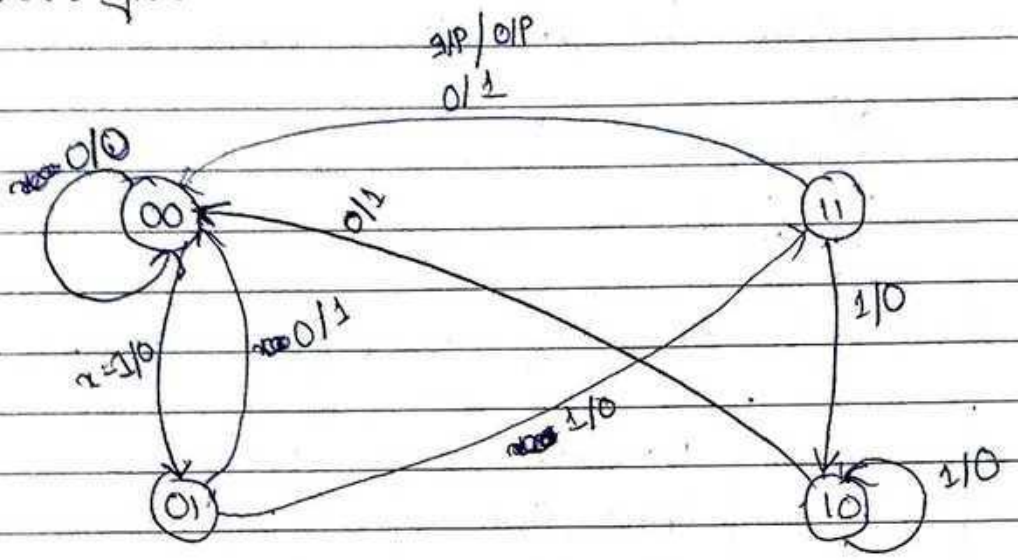
Q_{A+1} = next state of D_A F/F,

Q_{B+1} = next state of D_B F/F,

State table:-

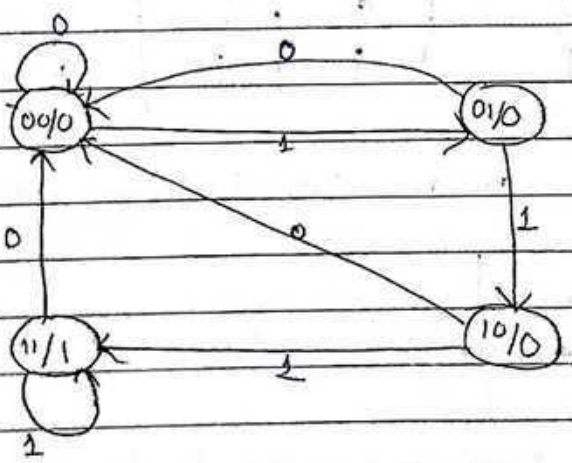
Present state		I/Ps	Next state		O/P
Q_A	Q_B	X	Q_{A+1}	Q_{B+1}	y
0	0	0	0+0 = 0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0+1 1	1	0
1	0	0	0+0 0	0	1
1	0	1	0+1 1	0	0
1	1	0	1+0 0	0	1
1	1	1	0+1 1	0	0

State Diagram :-



Mealy and Moore Models of Finite State Machines:-

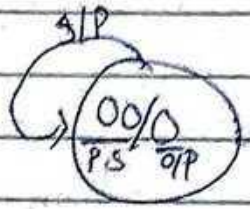
Design the sequential ckt specified by the state diagram of Figure using D F/Fs.



Sol^n :-

Acc. to the state diagram there are 2 D F/Fs are required to design the sequential ckt. Let the D F/Fs are represented as D_A and D_B. The present states are represented as Q_A and Q_B. The next states are represented as Q_{A+1} and Q_{B+1}.

After analyzing the State diagram again it is observed that there is 1 i/p and 1 o/p. Let the i/p is derived as x and o/p is y.



→ The State table of the diagram is given below.

Present state		I/P	Next State		output	
Q_A	Q_B	x	Q_{A+1}	Q_{B+1}	y	
0	0	0	0	0	0	
0	0	1	0	1	0	
0	1	0	0	0	0	(P.S/O.P)
0	1	1	1	0	0	
1	0	0	0	0	0	
1	0	1	1	1	0	
1	1	0	0	0	1	
1	1	1	1	1	1	

→ The i/p of the D F/Fs can be derived from the excitation table of the D F/F.

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Now rewrite the above state table acc to the excitation table to findout the i/ps of the D F/F

Q_A	Q_B	X	Q_{A+1}	Q_{B+1}	Y	D_A	D_B
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1

(kmap for D_A)

Q_A \ $Q_B x$	00	01	11	10
0	0	0	1	0
1	0	1	1	0

$D_A = Q_A x + Q_B x$

(kmap for D_B)

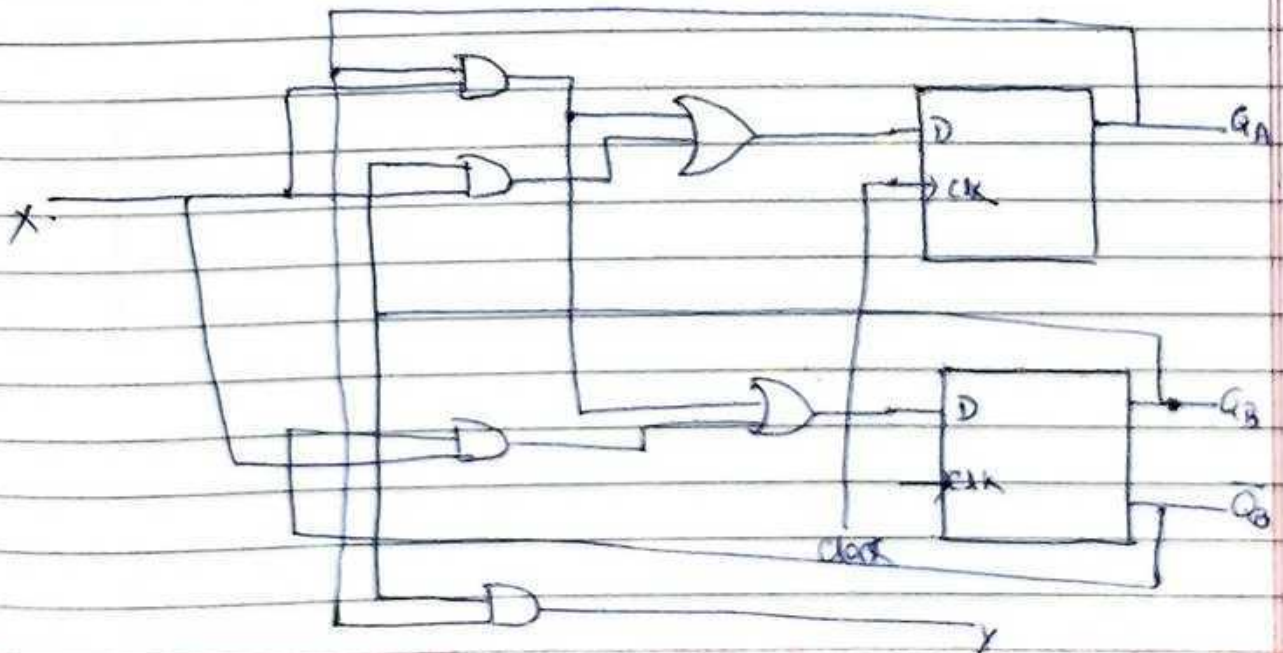
Q_A \ $Q_B x$	00	01	11	10
0	0	1	0	0
1	0	1	1	0

$D_B = Q_A \bar{Q}_B x$

Q_A \ $Q_B x$	00	01	11	10
0	0	0	0	0
1	0	0	1	1

$Y = Q_A Q_B$

(kmap for X)



classmate
Date
Page

17/10/2025

Unit-04

Registers:-

1) If the content of a 8-bit register is 10111010 then findout i) SHR 2
ii) SHL 1
iii) ROR 3.
iv) ROL 2

i) SHR 2:-

SHR1	0	1	0	1	1	1	0	1
SHR2	0	0	1	0	1	1	1	0

ii) SHL 1 :-

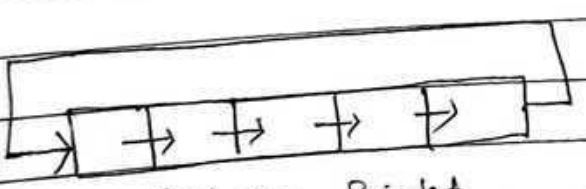
0	1	1	1	0	1	0	0

iii) ROR 3:-

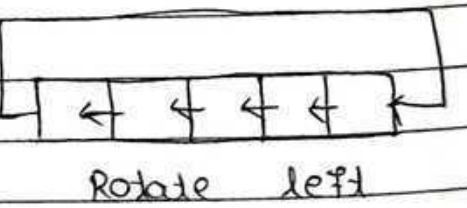
ROR 1 :-	0	1	0	1	1	1	0	1
ROR2:-	1	0	1	0	1	1	1	0
ROR3:-	0	1	0	1	0	1	1	1

iv) ROL 2:-

ROL 1 :-	0	1	1	1	0	1	0	1
ROL 2:-	1	1	1	0	1	0	1	0

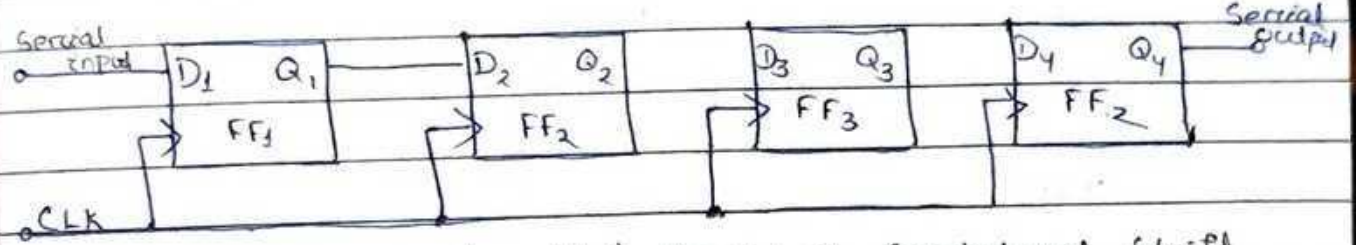


Rotate Right



Rotate left

Serial In- Serial- out (SISO) Shift Register :-



Logic diagram of a 4bit Serial-in, Serial out, Shift right, shift register.

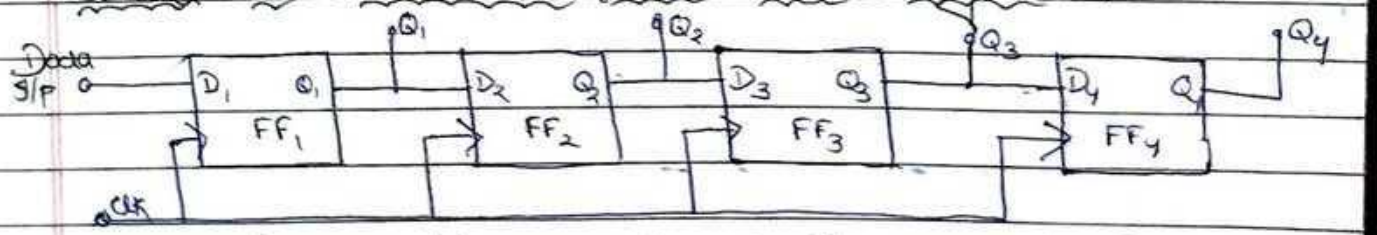
→ Let we have a 4 bit data i.e 1010 if we want to load and read the data in SISO Shift register then based on the following table the data will be loaded into the register and read from the register.

→ During SISO the LSB of the data will be entered first and the rest of the bits sequentially will be entered into the SISO register.

clock pulse	Data 1010	D_1	D_2	D_3	D_4	Q_1	Q_2	Q_3	Q_4
1st		LSB 0				0	0	0	0
2nd		1	0			1	0	0	0
3rd		0	1	0		0	1	0	0
4th		1	0	1	0	1	0	1	0
5th			1	0	1		1	0	1
6th				1	0			1	0
7th					1				1

LSB → Data out → MSB

Serial - in Parallel-out (SIPO) Shift Register :-

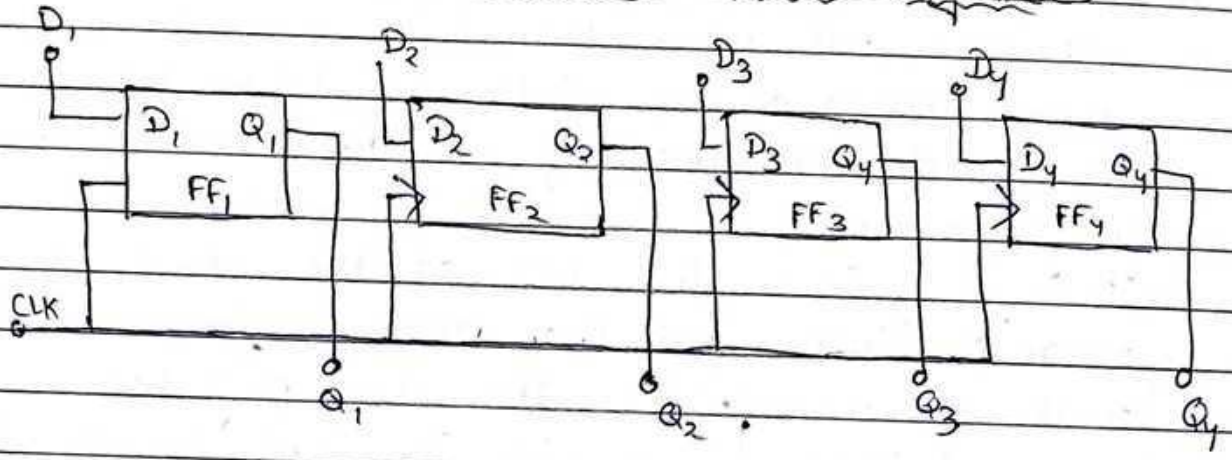


Logic diagram of a 4-bit Serial in Parallel out, Shift register

clock Pulse	Data 1010	D_1	D_2	D_3	D_4	Q_1	Q_2	Q_3	Q_4
1st		0				0			
2nd		1	0			1	0		
3rd		0	1	0		0	1	0	
4th		1	0	1	0	1	0	1	0

LSB, MSB, Data in, MSB, LSB, Data out

Parallel - In Parallel - Out (PIPO) Shift Register :-

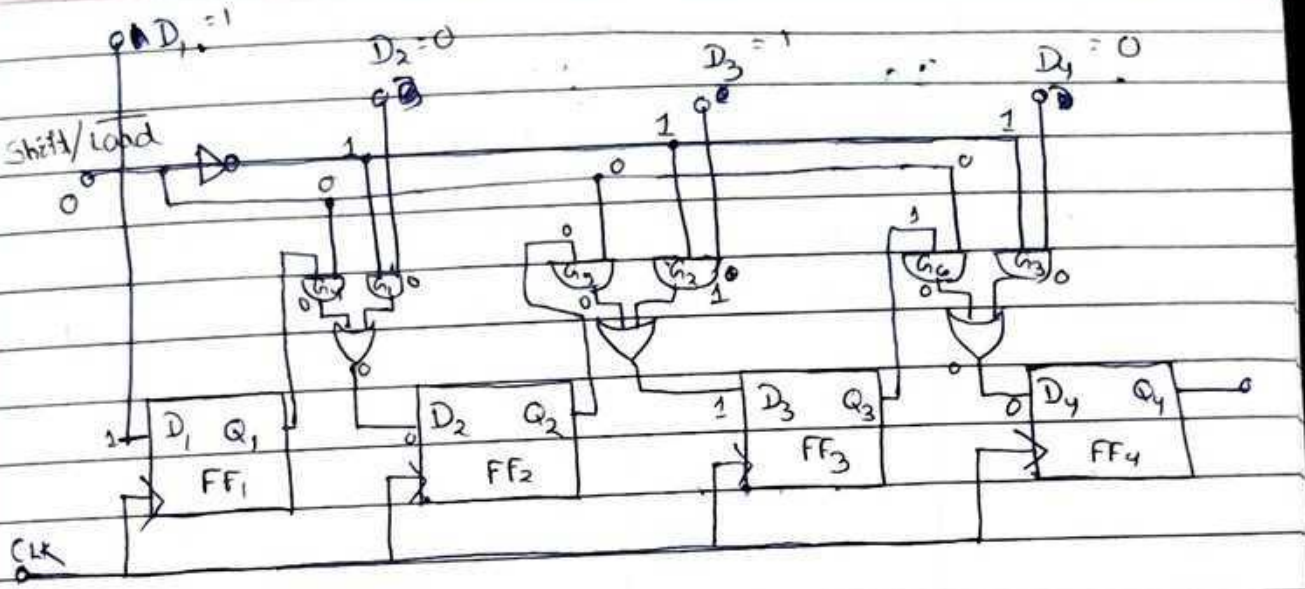


clock Pulse	Data 1010	D_1	D_2	D_3	D_4	Q_1	Q_2	Q_3	Q_4
1st		1	0	1	0	1	0	1	0

MSB, LSB, Data in, MSB, LSB, Data out

Parallel - In Serial - Out (PISO) . Shift Register :-

1010



Clock Pulse	Shift / $\overline{Load}$	Data	D_1	D_2	D_3	D_4	Q_1	Q_2	Q_3	Q_4
		1010								
1st	0		1	0	1	0	1	0	1	0
2nd	1		MSB	Data in		LSB		1	0	1
3rd	1								1	0
4th	1									1

LSB, Data out, MSB

-> In PISO the data will be loaded parallely and read from the register serially so to identify the loading and shifting operation shift/$\overline{Load}$ line is used. When shift/$\overline{Load}$ line = 0, Loaded
Shift / $\overline{Load}$ = 1, Shifting

-> After the analysis of SISO, SIPO, PISO and PIPO the following table is observed

Shift Reg	Clock Pulses Required: Data Loading	Data Reading	Total No. of CLK Pulses
SISO	n	n-1	2n-1
SIPO	n	0	n
PIPO	1	0	1
PISO	1	n-1	n

Counters :-

→ A register that goes through a prescribed sequence of states according to the application of input pulses is called as a counter.

→ It is divided into 2 types :-

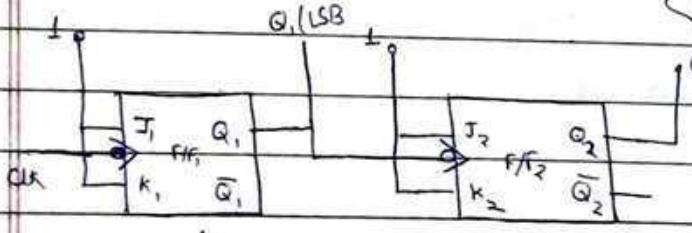
SQ i) Asynchronous or Ripple counter

LQ ii) Synchronous counter.

i) Asynchronous Counter :-

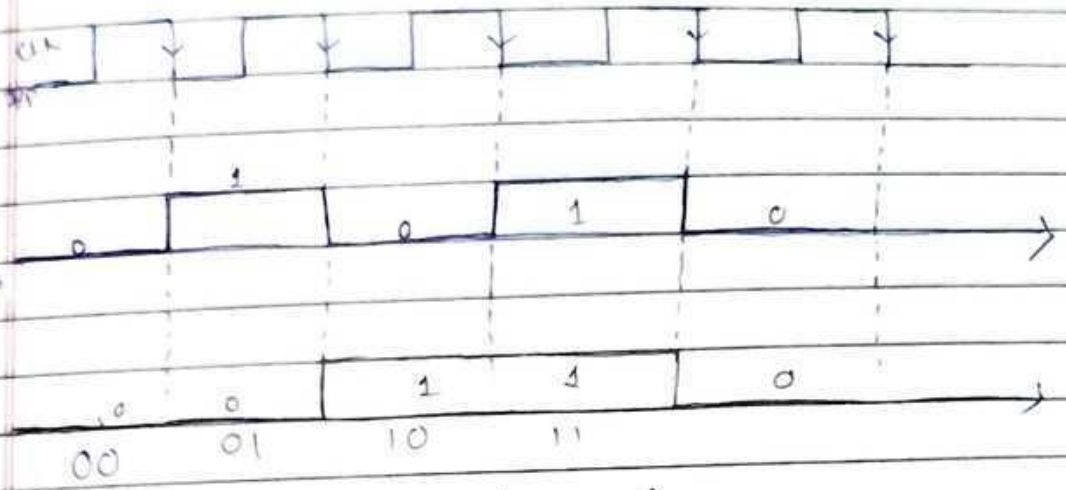
→ A binary ripple counter consists of a series connection of complementing F/Fs, with the o/p of each F/F connected to the clk i/p of the next higher order F/F.

Q:- 2-bit Ripple up-counter using Negative Edge-triggered F/Fs:-

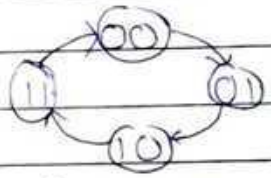


(a) logic diagram

MSB LSB
0 0
0 1
1 0
1 1

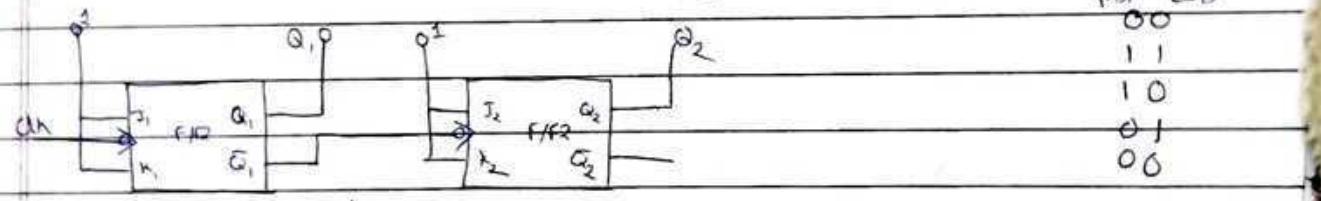


(Timing diagram)

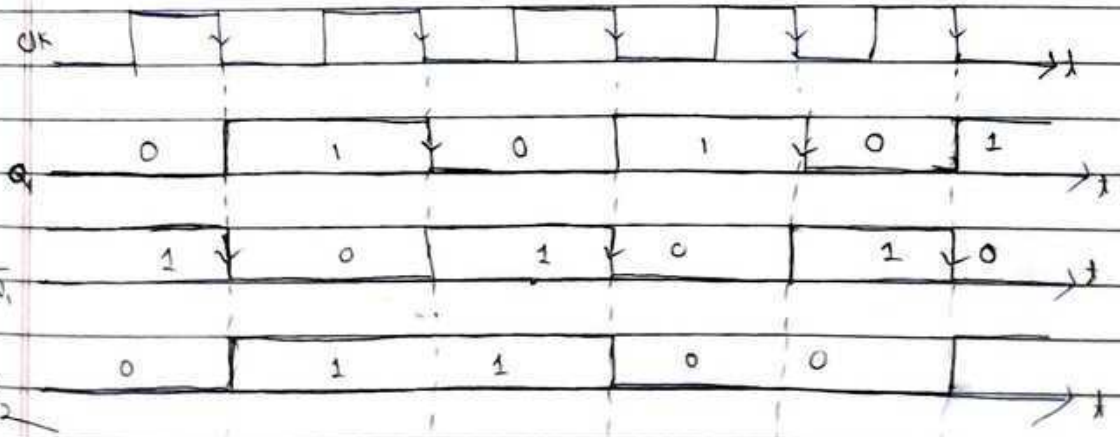


(State diagram)

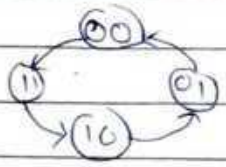
2-bit Ripple Down-counter using Negative Edge-triggered F/Fs:



(Logic diagram)



(Timing diagram)



(State diagram)

2-bit Ripple Up-down counter using Negative Edge-triggered FFs

VVIMP ii) Synchronous Counters :-

→ Step1 : Number of F/Fs : Based on the description of the problem, determine the required no. n of the FFs - the smallest value of n is such that the no. of states $N \leq 2^n$ - and the desired counting sequence.

→ Step2 : State diagram : Draw the state diagram showing all the possible states

→ Step3 : Select the F/F and excitation table

Step4 : Minimal expressions for the excitations by using K map

Step 5 : Logic diagram acc to eqn.

Q: Design a Synchronous 3-bit up counter using JK FFs :-

Soln:-

1) No. of F/Fs required to design :-

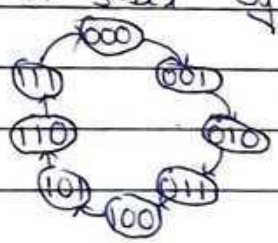
No. of F/Fs. required i.e $n=3$.

No. of States = $2^n = 2^3 = 8$

Counting values are = 0 to $2^n - 1$

= 0 to 7

2) State diagram of 3-bit Syn. upcounter :-



0	0	0 x
0	1	1 x
1	0	x 1
1	1	x 0

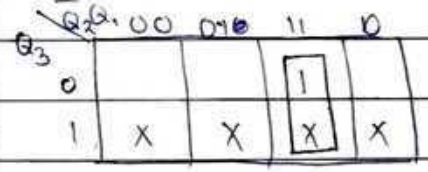
3) Select the F/F & excitation table by considering the PS & NS :-
As per the question the JK F/F will be used to design the counter.

Excitation table

Present State			Next State			Required excitations					
Q_3	Q_2	Q_1	Q_3^+	Q_2^+	Q_1^+	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	0	0	1	x	x	1	x	1
1	0	0	1	0	1	x	0	0	x	1	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	1	1	1	x	0	x	0	1	x
1	1	1	0	0	0	x	1	x	1	x	1

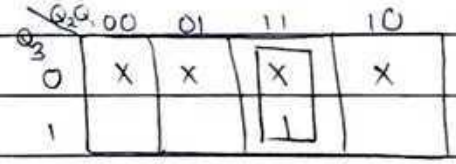
4) Using kmap derive expressions

i) J_3 :-



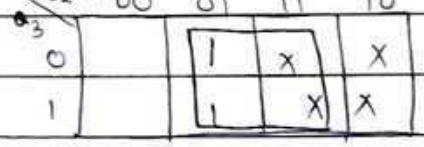
$J_3 = Q_2Q_1$

ii) K_3 :-



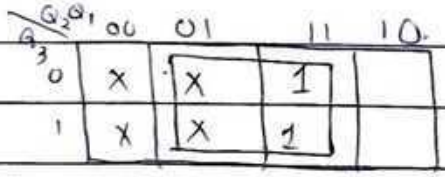
$K_3 = Q_2Q_1$

iii) J_2 :-



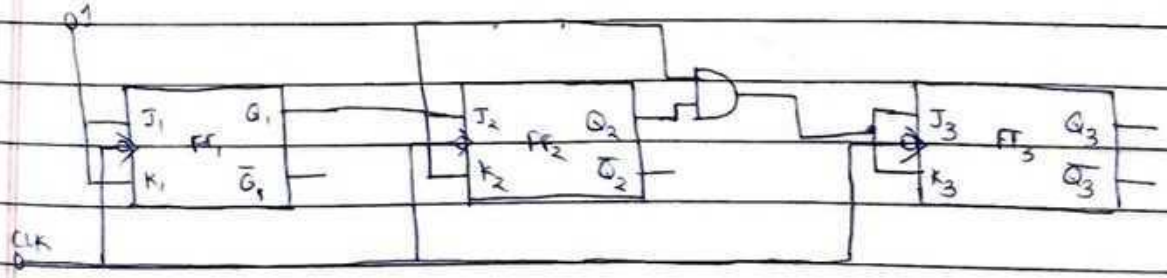
$J_2 = Q_1$

iv) K_2 :-



$K_2 = Q_1$

5)



0	0	X
0	1	1 X
1	0	X 1
1	1	X 0

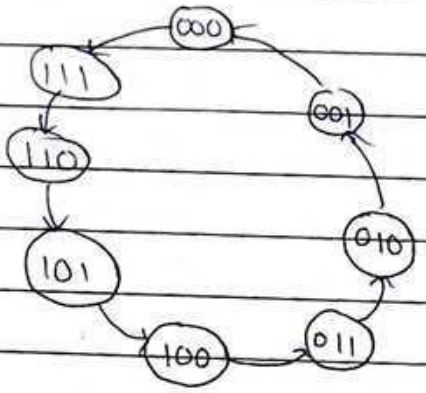
Design a Synchronous 3-bit Down Counter using JK FFs :-

1) No of F/Fs required : 3

No of States : 8

Counting value :- 0 to 7

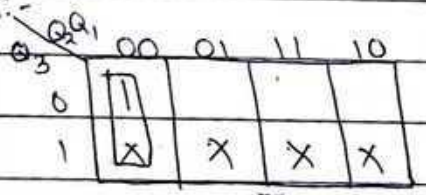
2)



3)

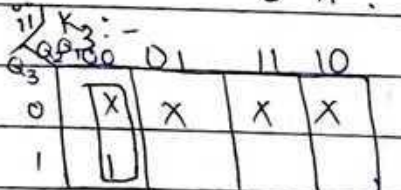
Present State			Next State			Required excitations					
Q_3	Q_2	Q_1	Q_3^+	Q_2^+	Q_1^+	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	1	1	1	1	X	1	X	1	X
1	1	1	1	1	0	X	0	X	0	X	1
1	1	0	1	0	1	X	0	X	1	1	X
1	0	1	1	0	0	X	0	0	X	X	1
1	0	0	0	1	1	X	1	1	X	1	X
0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	0	0	1	0	X	X	1	1	X
0	0	1	0	0	0	0	X	0	X	X	1

4) i) J_3 :-



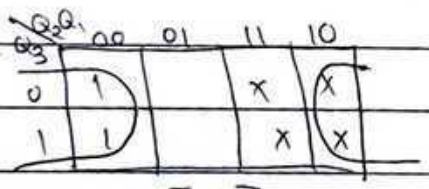
$J_3 = \overline{Q_2}\,\overline{Q_1}$

ii) K_3 :-



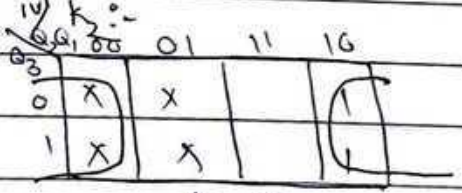
$K_3 = \overline{Q_2}\,\overline{Q_1}$

iii) J_2 :-

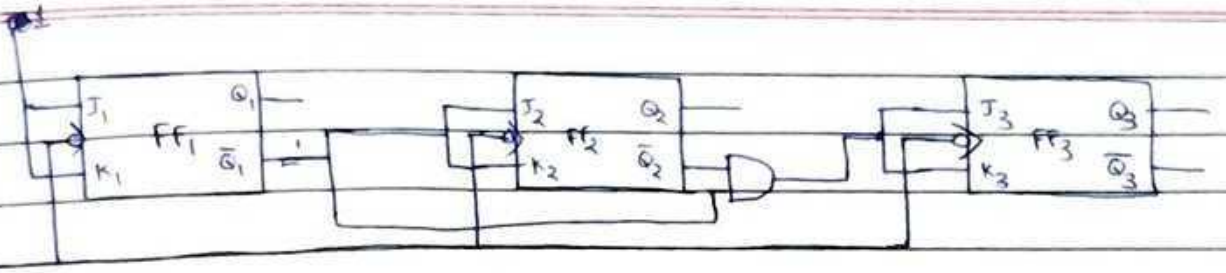


$J_2 = \overline{Q_1}$

iv) K_2 :-



$K_2 = \overline{Q_1}$



CLK