REVIEW OF NUMBER SYSTEMS & CODES.

Number systems :-

→ Number systems are used to represent data in digital form

→ Number system is nothing more than a code that uses symbols to represent a number.

→ In general, in any number system, there is an ordered set of symbols known as digits.

→ A number is made up of a collection of digits and it has two parts.

- integer and fraction part.

→ Both are separated by a radix point (·). The number is represented as,

$$(\underbrace{d_{n-1}\, d_{n-2} \cdots d_1 d_0}_{\text{Integer part}} \cdot \underbrace{d_{-1}\, d_{-2} \cdots d_{-m}}_{\text{Fractional part}})_r \quad r \rightarrow \text{radix}$$

Radix point.

Number systems are classified as.

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

Decimal number system :-

→ The decimal number system is a radix 10. It has 10 different digits or symbols to represent a number.

→ These are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

→ The radix point is known as the decimal point.

→ The value of any mixed decimal number is

$$d_n . d_{n-1} d_{n-2} \cdots d_1 d_0 . d_{-1} d_{-2} d_{-3} \cdots d_{-m}.$$

is given by.

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + \cdots + (d_1 \times 10^1) + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \cdots$$

Consider the decimal number.

$$(135.56)_{10} = 1 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2}$$

$$= 100 + 30 + 5 + 0.5 + 0.06$$

$$= 135.56.$$

Binary number system :-

→ The Binary number system is a radix 2. It has Two independent ~~digits~~ Bits.

→ Binary numbers are represented in terms of '0' and '1'.

→ The radix point is known as the Binary point.

'0' or '1' is a bit.

Four number bits → Nibble

8 bits. → byte

Group of 16 bits → word.

→ The binary number system is used in digital computers because the switching circuits used in these computers use two-state devices such as transistors, diodes, etc.

Octal Number System :-

→ The octal number system is a radix 8.

→ It has 8 different digits or symbols to represent a number

→ They are 0, 1, 2, 3, 4, 5, 6, and 7.

→ The radix point is known as the octal point.

→ octal number system is more efficient for us to write the number in octal rather than binary.

→ Electronically, it is easier and cheaper.

Hexadecimal Number System :-

→ The Hexadecimal Number System is a radix 16.

→ It has 16 different digits or symbols to represent a number.

→ They are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

→ The radix point is known as the Hexadecimal point.

→ The decimal equivalent of A, B, C, D, E and F are 10, 11, 12, 13, 14 and 15.

→ In hexadecimal number system easier way of representing large binary numbers stored and processed inside the computer.

→ The contents of the memory when represented in hexadecimal form are very convenient to handle.

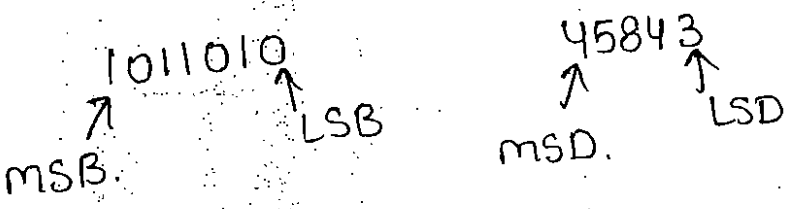
Radix - NO of symbols presented in a respective number system is called radix.

LSB - Least Significant bit.

The bit which having the least position weight bit is called LSB.

MSB - most significant bit.

The bit which having the most ~~significan~~ position weight is called MSB.



MSD - most Significant digit.

LSD - least Significant digit.

Conversion from one radix to another radix :-

In computer systems process binary data, but the information given by the user may be in the form of decimal number, hexadecimal number, & octal number.

→ Binary to decimal → octal to binary
→ octal to decimal → binary to octal
→ Hexadecimal to decimal → Hexadecimal to binary
→ decimal to binary → binary to Hexadecimal.
→ decimal to octal → octal to Hexadecimal
→ decimal to Hexadecimal → Hexadecimal to octal.

Binary to decimal :-

$(101101.0110)_2 = (\quad)_{10}$.

$$(\underset{5\,4\,3\,2\,1\,0\;-1\,-2\,-3\,-4}{101101.0110})_2 = (1\times2^5)+(0\times2^4)+(1\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)+(0\times2^{-1})+(1\times2^{-2})+(1\times2^{-3})+(0\times2^{-4})$$

$$= 32+0+8+4+0+1+0+0.25+0.125+0.$$

$$= (45.375)_{10}.$$

octal to decimal :-

→ $(4053.03)_8 = (2091\quad)_{10}$.

$$(\underset{3\,2\,1\,0\;-1\,-2}{4053.03})_8 = 4\times8^3+0\times8^2+5\times8^1+3\times8^0+0\times8^{-1}+3\times8^{-2}.$$

$$= 2048+0+40+3+0+$$

$$= 2091$$

$(532.78)_8 = (\quad)_{10}$.

In this given number system is octal but the value in the given problem 8 is not a octal number.

$$(\underset{2\,1\,0\;-1\,-2}{753.63})_8 = (491.79.)_{10}$$

$$= 7\times8^2+5\times8^1+3\times8^0+6\times8^{-1}+3\times8^{-2}.$$

$$= 448+40+3+0.75+0.046$$

$$= 491.79$$

Hexadecimal to decimal:-

→ $(5A3)_{16} = (1443.)_{10}$

$(5A3)_{16} = (5 \times 16^2) + (10 \times 16^1) + (3 \times 16^0)$

$= 1280 + 160 + 3$

$= (1443)_{10}$

→ $(64D)_{16} = (1613)_{10}$.

$(64D)_{16} = (6 \times 16^2) + (4 \times 16^1) + (13 \times 16^0)$.

$= 1,536 + 64 + 13$

$= (1613)_{10}$

Decimal to binary :-

→ To convert a decimal number to a binary number is known as repeated division and multiplication method

→ The integer and fractional parts of a decimal number are treated separately for the conversion and then the results are combined.

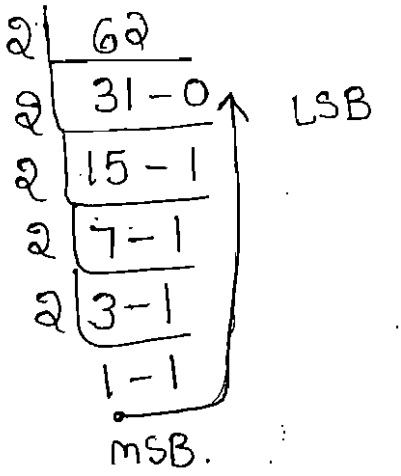
→ Integer part conversion.

To convert the integer part by using the repeated division method, the given decimal number is divided by 2

The remainder is found after each division until the quotient 0.

The remainder read from bottom to top give the equivalent binary integer number.

$(62)_{10} = (\quad)_2$



$(62)_{10} = (111110)_2$

Fractional part conversion.

To convert the fractional part by using the repeated multiplication method, the given decimal number is multiplied by 2.

The integer part of the multiplication is found after each multiplication operation until the fractional part of a decimal number becomes zero.

The integers read from top to bottom gives the equivalent Binary fraction.

$(0.625)_{10} = (\quad)_2$.

$0.625 \times 2 = 1.25 \quad 1$

$0.25 \times 2 = 0.5 \quad 0$

$0.5 \times 2 = 1.0 \quad 1$ ↓

$(0.625)_{10} = (101)_2$

→ $(105.15)_{10} = (\quad)_2$

2 | 105	
2 | 52 - 1	↑
2 | 26 - 0	
2 | 13 - 0	
2 | 6 - 1	
2 | 3 - 0	
1 - 1	

integer part.

$105_{10} = 1101001_2$

$0.15 \times 2 = 0.30$ 0
$0.30 \times 2 = 0.60$ 0
$0.60 \times 2 = 1.20$ 1
$0.20 \times 2 = 0.40$ 0
$0.40 \times 2 = 0.80$ 0
$0.80 \times 2 = 1.60$ 1 ↓

Fractional part

$0.15_{10} = 0.001001_2$.

$(105.15)_{10} = (1101001.001001)_2$.

Decimal to octal :-

Decimal conversion is same as decimal to binary except that in this case repeated multiplication and division are by 8 which is the base of octal number system.

$(183.62)_{10} = (\quad)_8 = (267.475)_8$.

8 | 183	
8 | 22 - 7	
2 - 6	

$0.62 \times 8 = 4.96$ 4
$0.96 \times 8 = 7.68$ 7
$0.68 \times 8 = 5.44$ 5.

$(145.93)_{10} = (\quad)_{8} = (221.734)_{8}$

8 | 145
8 | 18 - 1
8 | 2 - 2
0 - 2

$0.93 \times 8 = 7.44$
$0.44 \times 8 = 3.52$
$0.52 \times 8 = 4.16.$

Decimal to Hexadecimal :-

Decimal to Hexadecimal conversion is same as decimal to octal. except that in this case repeated multiplication and division are by 16 which is the base of Hexadecimal number system.

→ $(138.58)_{10} = (80A.947.)_{16}$

$16 \times 8 = 128$
$138 - 128 = 10$

16 | 138
16 | 128 - 10
8 - 0

$0.58 \times 16 = 9.28$
$0.28 \times 16 = 4.48$
$0.48 \times 16 = 7.68.$

16 | 138 (L)
16 | 8 - 10 (A)
0 - 8 (M)

→ $(256.26)_{10} = (100.428F)_{16}$

16 | 256
16 | 16 - 0
1 - 0

$0.26 \times 16 = 4.16$
$0.16 \times 16 = 2.56.$
$0.56 \times 16 = 8.96$
$0.96 \times 16 = 15.36.$

octal to binary conversion :-

To convert octal to binary. just replace each octal digit by its 3-bit binary equivalent.

→ $(563)_8 \rightarrow (101110011)_2$

→ $(725)_8 \rightarrow (111010101)_2$

→ $(326)_8 \rightarrow$ 3 2 6

011 010 110

$(011010110)_2$.

Binary to octal conversion :-

To convert binary to octal, just replace each group of 3 bits by equivalent octal number.

→ Splitting the integer and fractional parts into Groups of three bits. starting from the binary point on both sides.

→ In integer part to making group of 3-bits from right to left

→ In fractional part to making group of 3-bits from left to right.

→ If needed add 0's on the extreme left of the integer part and extreme right of the fractional part.

→ $(110110.01101)_2 \longrightarrow (155.32)_8$

001	101	101	.	011	010
1	5	5	.	3	2

→ $(101101010.1101010)_2 \longrightarrow (552.650)_8$

101	101	010	.	110	101	000
5	5	2		6	5	0

Hexadecimal to Binary conversion :-

To convert a hexadecimal number to binary, replace each digit by it's 4-bit binary equivalent.

Example :-

→ $(6A3)_{16} \longrightarrow (\quad)_2$

6	A	3
0110	1010	0011

$(6A3)_{16} \longrightarrow (011010100011)_2$

A - 10
B - 11
C - 12
D - 13
E - 14
F - 15

→ $(58C.26)_{16} \longrightarrow (\quad)_2$

5	8	C	.	2	6
0101	1000	1100		0010	0110

$(58C.26)_{16} \longrightarrow (010110001100.0010\ 0110)_2$

→ Binary to Hexadecimal conversion :-

To convert binary to Hexadecimal number. by splitting the integer and fractional parts into Groups of 4-bits. Replace each 4-bit binary group by the equivalent hexadecimal digit.

→ $(1001011010101)_2 \longrightarrow (\quad)_{16}$.

0001	0010	1101	0101
1	2	D	5.

$(1001011010101)_2 \rightarrow (12D5)$.

→ $(1101101010101.1010101)_2 \rightarrow (\quad)_{16}$.

0001	1011	0101	0101	.	1010	1010	1000
1	B.	5	5.	.	A	A	8.

$(1101101010101.10101010)_2 \rightarrow (1B55.AA8)_{16}$.

→ $(110101101.001101)_2$.

0011	0110	1101	.	0011	0100
3.	6	D	.	3.	4.

→ Octal to Hexadecimal conversion

To convert an octal number to hexadecimal, first convert the given octal number to binary and then the binary number to hexadecimal.

→ $(356.63)_8 \longrightarrow (\quad)_{16}$.

1. octal to binary
2. binary to hexadecimal.

3	5	6	.	6	3
011	101	110	.	110	011

$(011\,101\,110\,.\,110\,011)_2$

0000	1110	1110	.	1100	1100
0	E	E	.	C	C

$(356.63)_8 \longrightarrow (0EE.CC)_{16}$

→ Hexadecimal to octal conversion

To convert a hexadecimal number to octal, first convert the given Hexadecimal number to binary and then the binary number to octal.

→ $(B9F.AE)_{16} \longrightarrow (\quad)_8$

1. Hexadecimal to binary
2. binary to octal.

B	9	F	. A	E
1011	1001	1111	1010	1110

101	1001	1111	. 1010	1110 0
5	6	3	7	5	3	4

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

Conversion from any system to any system.:-

$(1321)_5 \longrightarrow (\quad)_{12}$.

To convert any system to any system, first convert the given number to decimal number and then the decimal number to any system.

Step1:- $(1321)_5 \longrightarrow (211)_{10}$.

$(1321)_5$ — 3 2 1 0

$= 1\times5^3 + 3\times5^2 + 2\times5^1 + 1\times5^0$

$= 125 + 75 + 10 + 1$

$= (211)_{10}$

Step2:- $(211)_{10} \longrightarrow (157)_{12}$.

12 | 211

12 | 17 - 7

12 | 1 - 5

0 - 1

$(1321)_5 \longrightarrow (157)_{12}$.

r-1's complements and r's compliments :-

Complements are used in digital systems to simplify the subtraction operation base (radix) r system there are two useful types of complements, the r's-complement (Radix complement) and the (r-1)'s-complement (Diminished Radix complement).

1's and 2's Complements :-

The 1's complement of a binary number is obtained by complementing all its bits, that is, by replacing all 0's by 1's and all 1's by 0's.

Example →	01101010	10101010 1100
1's complement →	10010101	01010101 0011

The 2's complement of a binary number is obtained by adding '1' to its 1's complement.

Example	01101010	10101101
1's complement	10010101	01010010100
	+1	+1
2's complement	10010110	01010010101

1's complement :-

In 1's complement subtraction, add the 1s complement of the subtrahend to the minuend. If there is a carry out, bring the carry around and add it to the LSB. This is called the end around carry. If the MSB is 0, the Result is positive and is in true binary. If the MSB is '1' The Result is negative and is in its 1s complement form.

Example :-

Subtract 20 from 36 using the 8-bit 1's complement for

36 - 20

36 - 00100100 ⟶ 00100100

20 - 00010100 ⟶ +11101011 (1's complement form

100001111

1 (Add the end arou carry)

00010000

MSB.

36 - 20 = 16

The MSB is 0. The result is positive.

In 1's Complement addition, add the 1's complement form of subtrahend and minuend. If there is a carry out add the carry in LSB position.

If the MSB of the result 0, then the answer is positive and MSB of the result 1, then the answer is negative.

Example 2

Add +36 to +20 using 8 bit 1's complement form.

$36 \rightarrow 00100100 \rightarrow 11011011$

$20 \rightarrow 00010100 \rightarrow 11101011$

$\boxed{1}\,11000110$

$\hookrightarrow \quad 1$ (add the end around carry).

11000111

MSB is 1, then the answer is negative

$00111000 = 56$

$36+20 = 56.$

Example 3

Add -36 to -20 using 8 bit 1's complement form.

$36 \rightarrow 00100100 \rightarrow 11011011$

$00010100 \rightarrow 11101011$

$\boxed{1}\,11000110$

$\quad\quad\quad\quad\quad\quad 1$

$11000111 \rightarrow$ negative.

$00111000 = 56.$

$-36-20 = -56\ (11000111)$

Example 4

Add ~~-25~~ -36 to +20 using 8-bit 1's complement form.

$+36 \rightarrow 00100100$

$-36 \rightarrow 11011011$

$+20 \quad 00010100$

$11101111 \longrightarrow -16$

00010000

$+20$

-36

-16

2's complement :-

In 2's complement subtraction, add the 2's complement of the subtrahend to the minuend. If there is a carry out, ~~bring~~ ignore it. If the msb is 0, the result is positive and is in true binary. If the msb is '1' the result is negative and is in its 2's complement form.

Example 1 :-

Subtract 20 from 36 using the 8-bit 2's complement form.

36 - 20.

→ Take subtrahend. 20.

20 → 00010100

1's complement → 11101011

2's complement → 11101100

36 → 00100100

20 → 11101100 (2's complement form of 20).

1]00010000 (ignore the carry).

Example 2 :-

Add -36 to +20 using the 8-bit 2's complement form.

+36 ⟶ 00100100

− 11011011 (1's complement).

+1

-36 ⟶ 11011100 (2's complement).

20 → 00010100

-36 → 11011100

-16 = 11110000

msb is 1, then result is negative.

11110000

00001111

+1

00010000 (+16)

Example 3 :-

Add -45.75 to +87.5 using the 12-bit 2's complement arithmetic.

+87.5 → 0101 0111 . 1000

+45.75 → 00101101 . 1100

1101 0010 . 0011 (1's complement form)

1 1 1

1101 0010 . 0100 (2's complement form)

128 64 32 16 8 4 2 1

0 0 1 0 1 0 0 1

+87.5 → 0101 0111 . 1000

1101 0010 . 0100

1 1 1 1 1 1 1 1

1] 0010 1001 . 1100 (ignore the carry.

↓ ↓

41 . 75

Example 4 :-

Add -45.75 to -87.5 using the 12-bit 2's complement form.

+87.5 → 0101 0111 . 1000

(1's comple) 1010 1000 . 0111

(add 1) 1 1 1 1

(2's comp). 1010 1000 . 1000

-87.5

+45.75 → 00101101 . 1100

(1's comp) 1101 0010 . 0011

1

1 1

(2's comp). 1101 0010 . 0100

-45.75

-87.5. 1010 1000 . 1000

-45.75 1101 0010 . 0100

1] 0111 1010 . 1100 (ignore the carry).

1000 0101 . 0011

1

1 1 1

1000 101 . 0100

9's Complement:-

In 9's complement subtraction.

→ find the 9's complement of the subtrahend.
→ Add 9's complement of subtrahend to the minuend.
→ if there is a carry it indicates that the answer is positive. Add the carry to the LSD of this result to get answer.
→ If there is no carry, it indicates that the answer is negative and the result obtained is its 9's complement.
→ Find the 9's complement of the following decimal numbers.

→ 4986
9999
4986
5013

→ 738.65
999.99
738.65
261.34

→ 4526.075
9999.999
4526.075
5473.924

9's complement method of subtraction:-

→ 745.86 - 436.62

Step 1:- 999.99
436.62
563.37

Step 2:-
745.86
563.37
1) 309.23
1
309.24

The carry indicates the answer is positive.

+309.24.

436.62 – 745.86.

Step1:- 999.99
745.86
254.13

Step2:- 436.62
254.13
690.75

Step3:- If there is no carry. the answer is negative.

Step4:- 999.99
690.75
309.24

answer is -309.24.

10's complement:-

The 10's complement of a decimal number is obtained by adding a 1 to its 9's complement.

Find the 10's complement of the following decimal number.

→ 4986
9999
4986
5013
1
5014

→ 738.65.
999.99
738.65
261.34
1
261.35

→ 4526.075
9999.999
4526.075
5473.924
1
5473.925

10's complement method of subtraction :-

→ 745.86 - 436.62

```
  999.99
  436.62
 -------
  563.37
       1
 -------
  563.38   (10's complement)
```

```
  745.86
  563.38
 -------
(1)309.24   (ignore the carry)
```

→ 436.62 - 745.86

```
  999.99
  745.86
 -------
  254.13
       1
 -------
  254.14
```

```
  436.62
  254.14
 -------
  690.76   (No carry, negative answer)
```

```
  999.99
  690.76
 -------
  309.23
       1
 -------
  309.24   - 309.24.
```

15's Complement methode :-

Find the 15's complement of the following decimal number.

→ 6A36

15	15	15	15
6	A	3	6
9	5	C	9

→ 9AD.3A

15	15	15	15	15
9	A	D	. 3	A
6	5	2	. C	5

15's complement methode of subtraction :-

69B - C14

→

15	15	15	.
C	1	4	
3	E	B	.

→

6	9	B
3	E	B
A	8	6

→ No carry, it indicates answer is negative.

15	15	15
A	8	6
-	5	79

$69B_{16} - C14_{16} = -579_{16}$

F - 15
10 - 16
11 - 17
12 - 18
13 - 19
14 - 20
15 - 21
16 - 22
17 - 23
18 - 1E
19 - 1F

20
21
22

16 | 22 | 6
1 | 6 ↑
16

16's complement methode :-

Find the 16's complement of the following decimal number.

→ A8C

```
  15  15  15
 - A   8   C
-------------
  5   7   3   — (15's complement)
  0   0   1   (add 1)
-------------
  5   7   4   (16's complement).
-------------
```

16's complement methode of subtraction :-

C9B − C14

```
  15  15  15
   C   1   4
-------------
   3   E   B   (15's complement).
           1
-------------
   3   E   C   (16's complement).
-------------

   1   1
   C   9   B
   3   E   C
-------------
1  0   8   7   (ignore it).
-------------
```

if carry present, the answer is positive.

C9B − C14 → $(087)_{16}$.

7's complement methode :-

Find the 7's complement of the following decimal number.

→ 536

777
536
241 (7's complement).

7's complement methode of subtraction :-

623 - 352.

777
352
425 (7's complement).

623 → 623
-352 → 425
1250
↳ 1 (add carry)
251

7 - 7
8 - 10
9 - 11
10 - 12
11 - 13
12 - 14
13 - 15

352 - 623

777
623
154.

352
154
526 (No carry, answer is negative

777
526
-251

8's complement methode

Find the 8's complement of the following decimal number

→ 536.

```
  777
  536   (
  241   (7's complement)
   +1   (add 1)
  242   (8's complement)
```

8's complement methode of subtraction :-

623 - 352

```
  777
  352
  425   (7's complement)
    1   (add 1)
  426   (8's complement)
```

```
  623
  426
  251   (ignore the carry).
```

623
−426
1049

if carry present, the answer is positive.

$623 - 352 = (251)_8$

Number Representation in binary :-

There two types of numbers

→ unsigned numbers

→ signed numbers.

The numbers without positive or negative signs are known as unsigned numbers. The unsigned numbers are always positive numbers. (considered).

In signed number system, the number may be positive or negative. Different formats are used for representation of signed binary numbers. They are.

→ sign - magnitude representation

→ 1's complement representation.

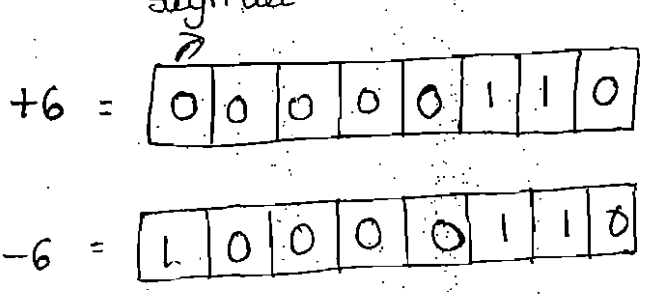
→ 2's complement representation.

Sign - magnitude Representation :-

In sign - magnitude representation the MSB represents the sign and the remaining bits represents the magnitude. The MSB bit is 1, it indicates the sign is negative, and MSB bit is 0, it indicates the sign is positive.

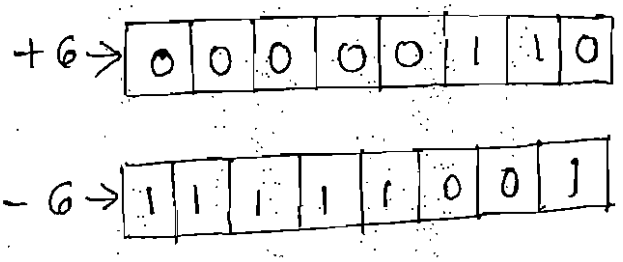
In eight bit representation, the MSB indicates the sign. and remaining seven bits represent the magnitude.

Consider the number +6 and -6 represented in binary.



1's Complement Representation :-

In the 1's complement representation, the positive numbers remain uncharged. 1's complement representation of negative number can be obtained by the 1's complement of the binary number.

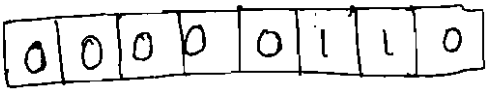


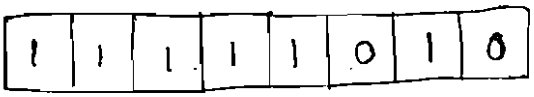
2's Complement Representation :-

In the 2's complement representation, the positive numbers remain uncharged, 2's complement representation of negative number can be obtained by

→ find the 1's complement of the number (by replacing 0 by 1 and 1 by 0)

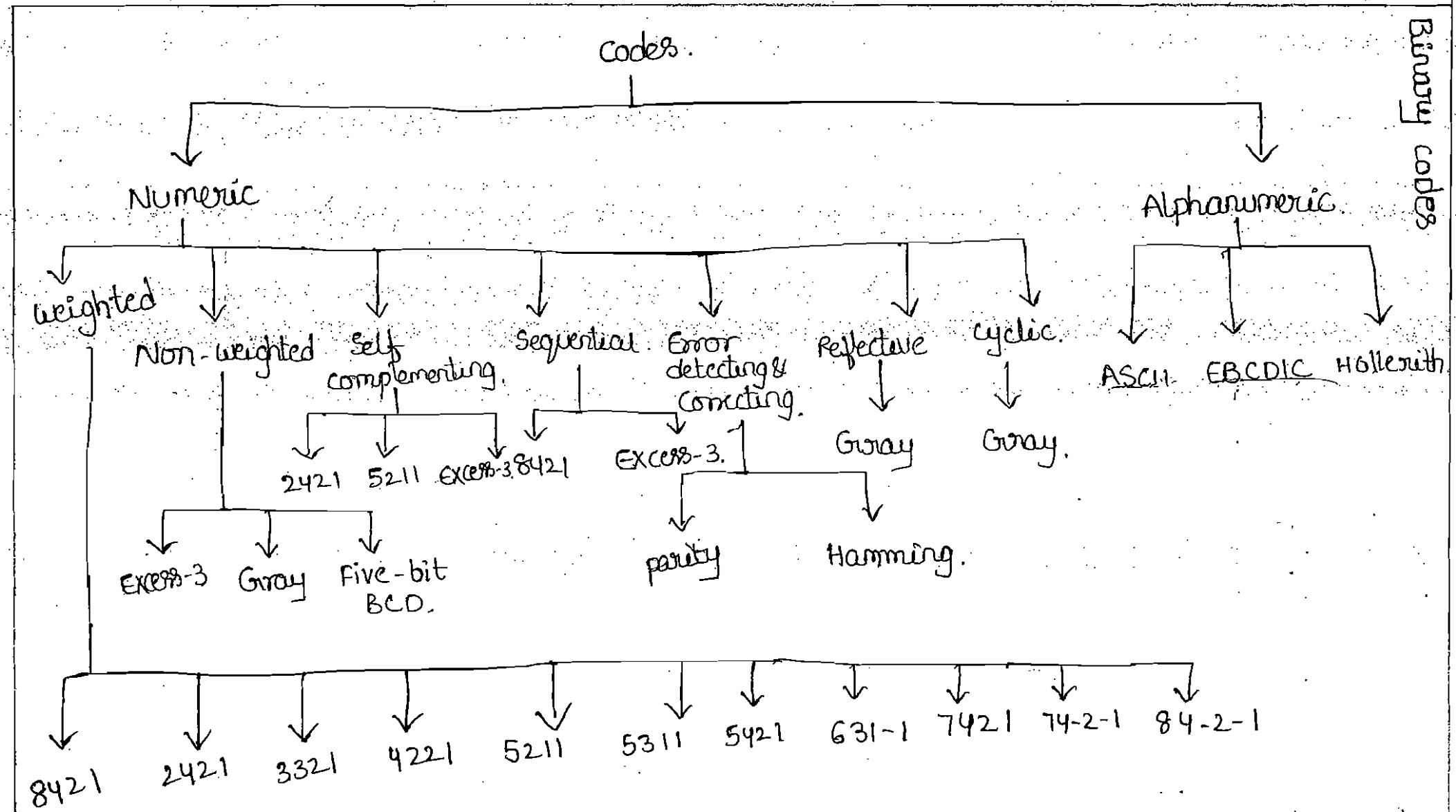
→ find the 2's complement of the number by adding 1 to 1's complement of the number.

+6 → 

-6 → 

Decimal	Signed magnitude	Signed - 1's Complement	Signed - 2's Complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	-
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	-	-	1000

1001
1110



→ Numeric codes

Numeric codes are codes which represent numeric information that is only numbers as a series of 0s and 1s. Numeric codes used to represent the decimal digits are called Binary Coded decimal (BCD).

8421, 2421, 5211 are BCD codes.

8421, XS-3, Gray code are numeric codes.

Alphanumeric codes

Alphanumeric codes are binary codes which represent alphanumeric data. This code includes alphanumeric data, letters of the Alphabet, numbers, mathematical symbols and punctuation marks. ASCII and EBCDIC are commonly used Alphanumeric codes.

ASCII (American Standard code for Information Interchange).

EBCDIC (Extended Binary coded decimal Interchange code).

Alphanumeric codes are used to interface Input - output devices such as keyboards, printers, VDUs.

→ weighted and non-weighted codes:-

The weighted codes are those which obey the position - weighting principle. Each position of the number represents a specific weight.

Ex:- 8421, 2421, 84-2-1.

In decimal code, if number is 637 then weight of 6 is 100, weight of 3 is 10, and weight of 7 is 1

Non-weighted codes are codes which are not assigned with any weight to each digit position.

EX:- XS-3 (Excess-3) and Gray code.

→ Error detecting and correcting codes:-

Codes which allows only error detection are called error detecting codes. Codes which allows error detection and correction are called error-detec error correction codes. Error detection and correction involves the addition of extra bits, called check bits, to the transmitted data. These extra bits allows the detection and some time correction of errors in data.

EX:- parity and Hamming codes.

→ Self-complementing codes:-

A code is said to be a self-complementing if the code word of the 9's complement of N, of 9-N can be obtained by the 1's complement of the word of N. Therefore in a self complementing code, the code word for 9 is the complement for the code 0, 8 for 1, 7 for 2, 6 for 3 5 for 4.

Excess-3 code is example of self-complementary code.

→ Sequential codes:-

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

EX:- 8421 and excess-3 codes.

→ Cyclic codes :-

Cyclic codes are also called unit distance codes. The name itself indicates that there is unit distance between two consecutive codes. The unit distance codes have special advantages in that they minimize transitional errors or flashing.

EX:- Gray code.

→ Reflective codes :-

A Reflective code is a binary code in which the n least significant bits for code words 2^n through $2^{n+1}-1$ are the mirror images of those for 0 through 2^n-1.

EX:- Gray code.

Binary coded decimal (BCD) code or 8421 code :-

In this code, each decimal digit, 0 through 9, is coded by a 4-bit binary number. it is also called the natural binary code. For example, the decimal number 15 can be represented as 1111 in binary but in BCD.

00010101

It is useful for mathematical operations. The main advantages of this code is its ease of conversion to and from decimal.

disadvantage of the BCD code is that, arithmetic operations are more complex and it requires more bits

BCD addition :-

→ Convert the decimal numbers into their equivalent BCD codes.

→ Add the two BCD numbers, using the basic rules for Binary addition.

→ Check the result If sum is equal to or less than 9, it is a valid BCD number.

→ If the result is greater than 9 or if a carry generated from the 4-bit sum, the sum is invalid.

→ To correct the sum, add (0110) 6 to the sum term of that Group.

→ If carry results when 6 is added, simply add the carry to the next 4-bit Group.

Example :-

23 + 13.

$$\begin{array}{rl} 23 \rightarrow & 0010\ 0011 \\ 13 \rightarrow & 0001\ 0011 \\ \hline 36 & 0011\ 0110 \end{array}$$

683.5 + 256.2.

$$\begin{array}{rl} 683.5 \rightarrow & 0110\ 1000\ 0011.0101 \\ 256.2 \rightarrow & 0010\ 0101\ 0110.0010 \\ \hline 939.7 & 1000\ 1101\ 1001.0111 \\ & \quad\ 0110 \\ \hline & 1000\ 0011\ 1001.0111 \\ & \quad\ \curvearrowright \\ \hline & 1001\ 0011\ 1001.0111 \end{array}$$

BCD Subtraction :-

→ Each 4-bit Group of subtrahend from the corresponding 4-bit Group of the minuend.

→ if there is no borrow from the next higher Group then no correction.

→ if there is a borrow from the next group, then 6(0110) is subtracted from the difference term of group.

Example :-

23 - 13

23 → 0010 0011

13 → 0001 0011

10 ← 0000 0000

1 0

236.8 - 142.5

236.8 → 0010 0011 0110. 1000

142.5 → 0001 0100 0010. 0101

94.3 0000 1111 0100.0011

0110

0000 1001 0100. 0011

0 9 4 . 3

BCD Subtraction using 9's complement :-

236.8 - 142.5

999.9 236.8 → 0010 0011 0110. 1000

142.5 857.4 → 1000 0101 0111. 0100

857.4 629.4

1010 1000 1101. 1100

0110 0110

1010 1001 0100. 0010

0110

1 0000 1001 0100. 0010

1

0000 1001 0100. 0011

0 9 4 . 3

BCD Subtraction using 10's Complement:-

1) 236.8 - 142.5

```
  999.9
  142.5
 -------
  857.4
      1
 -------
  857.5
```

```
236.8 → 0010 0011 0110. 1000
857.5 → 1000 0101 0111 . 0101
        ---- ---- ---- ----
        1010 1000 1101 . 1101
        0110      0110 . 0110
        ---- ---- ---- ----
   1 ]  0000 1001 0100 . 0011
        ---- ---- ---- ----
          0    9    4 . 3
```

In 10's complement ignore the carry.

```
  236.8
  142.5
 -------
  094.3
 -------
```

Excess Three Code (XS-3):-

The excess-3 code for a given decimal number is determined by adding 3 (0011) to each decimal digit in the Given number. It is a non-weighted BCD code, sequential and self-complementing code

It can be used for arithmetic operations.

XS-3 Addition:-

23+13

23 → 56 → 0101 0110

13 → 46 → 0100 0110

1001 1100

-0011 -0011

0110 1001 (answer in XS-3)

23	13
33	33
56	46

If carry generated add +0011

If carry not generated subtract -0011

→ 236.8 + 142.5

236.8 → 569.B → 0101 0110 1001 . 1011

142.5 → 475.8 → 0100 0111 0101 . 1000

1001 1101 1110 ① 0011

1001 1101 1111 0011

~~(answer in XS-3)~~

1001 1101 1111 0011

-0011 -0011 -0011 +0011

0110 1010 1100 . 0110 (answer in XS-3)

3 7 9 . 3

XS-3 Subtraction :-

→ if borrow generated subtract 3 (0011)

→ if borrow not generated add 3 (0011).

→ 25.3 - 16.5.

25.3 → 58.6 → 0101 1000. 0110

16.5 → 49.8 → 0100 1001 . 1000

0000	1110	1110
+0011	-0011	-0011
0011	1011	1011

Answer in XS-3.

3	B	B
-3	-3	-3
0	8 .	8

→ 267 - 175

267 → 5 9 10 → 0101 1001 1010

175 → 4 10 8 → 0100 1010 1000

0000	1111	0010
+0011	-0011	+0011
0011	1100.	0101

Answer in XS-3).

3	C .	5
-3	-3	-3
0	9 .	2

XS-3 Subtraction using 9's complement

→ 687 - 348

348 → 999
348
651 (9's complement form).
333
984 (XS-3 form).

687 → 1001 1011 1010 (XS-3 form of 687)
984 → 1001 1000 0100

I] 0000 0011 1110
↳ 1

I) 0001 0011 1110
↳ 1

0001 0011 1111
+ 0011 + 0011 - 0011

0110 0110 1100 (answer in XS-3).

6 6 12
-3 -3 -3

3 3 9.

If borrow generated subtract 3 (0011)
if borrow not generated add 3 (0011).

XS-3 Subtraction using 10's complement :-

→ 687 - 348

```
  999
  348
 ---
  651   (9's complement form)
    1   (+add'1)
 ---
  652   (10's complement form)
```

```
  652
 +333
 ---
  985   (XS-3 form).
```

687 → XS-3 → 1001 1011 1010

1001 1000 0101

1 0010 0011 1111

(1) (A)

1 0011 0011 1111

↳ (ignore the carry)

0011 0011 1111

+0011 +0011 -0011

1 1

0110 0110 1100

answer in XS-3.

-3 -3 -3

3 3 9.

Gray code :-

Gray code is a 4-bit numeric code. It is a unit distance code because successive code words in this code differ in one bit position only. It is also a reflective code, it is both reflective and unit distance.

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

Binary to Gray code conversion :-

→ Record the MSB of the binary as the MSB of the Gray code.
→ Add the MSB of the binary to the next bit in binary ignore the carry
→ Add the 2nd bit of binary to the 3rd bit of the binary, the 3rd bit to the 4th bit

Convert binary 0110 to the Gray code.

$$0 \oplus 1 \oplus 1 \oplus 0$$
$$0 \quad 1 \quad 0 \quad 1$$

Convert binary 1001 to the Gray code.

$$1 \oplus 0 \oplus 0 \oplus 1$$
$$1 \quad 1 \quad 0 \quad 1$$

Gray to Binary conversion :-

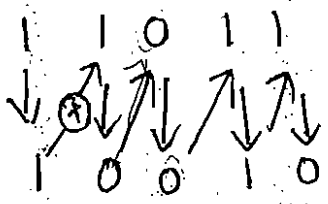
→ The MSB of the binary number is the same as the MSB of the Gray code

→ Add the MSB of the binary to the next significant bit of the Gray code, ignore the carry.

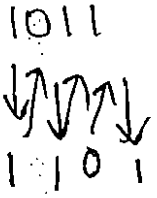
→ Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code.

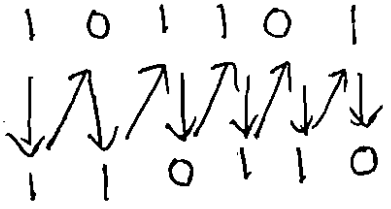
→ convert Gray to Binary.

11011



1011





Applications of Gray code :-

→ Gray code is used in the transmission of digital signals as it minimizes the occurrence of errors.

→ The Gray code is better than the binary code in angle measuring devices.

→ The Gray code is used for labelling the axes of Karnaugh maps.

→ The use of Gray codes to address program memory in computers minimizes power consumption.

→ Another application of Gray code is position indication in rotating disk.

Logic Gates:-

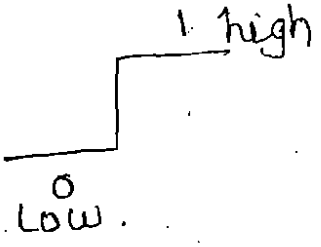
The logic Gates are the fundamental blocks of digital systems. The logic Gate is ability to make decisions (output). logic gate are electronic circuits because they are made up of a number of electronic devices and components.

Inputs and outputs of logic Gates can occur only in two levels.

Positive Logic:-

high → 1

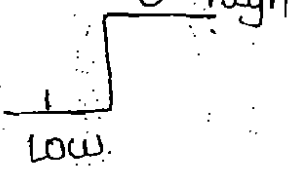
Low → 0.



negative Logic

Low → 1

high → 0



mixed Logic:-

mixed logics provides a simplified mechanism for the analysis and design of digital circuits. In mixed logic, the assignment of logical values to voltage values is not fixed, and it can be decided by the logic designers.

Logic design:-

The interconnection of gates to perform a variety of logical operations is called logic design.

Truth table:-

A table which lists all the possible combinations of input variables and the corresponding outputs is called a truth table.

Types of logic gates:-

1. AND Gate
2. OR Gate
3. NOT Gate

} Basic Gates.

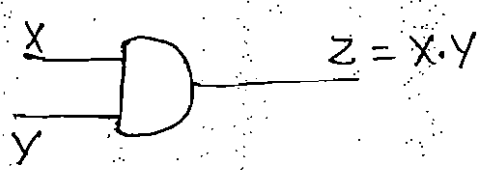
4. NAND Gate
5. NOR Gate

} universal Gates.

6. EXCLUSIVE OR Gate
7. EXCLUSIVE NOR Gate.

AND Gate:-

An AND gate is a logic circuit with two or more input and one output that performs ANDing operation. The output of an AND gate is high only when all of its inputs are in the high state. In all other conditions the output is low.



Logic symbol.

Truth table

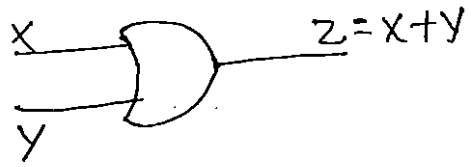
X	Y	Z=X.Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate:-

An OR gate is a logic circuit with two or more inputs and one output that performs ORing operation. The output of an OR gate is high when any one of the input is high state. In all other conditions the output is low.

~~NOT Gate~~ :-

Symbol



Truth table.

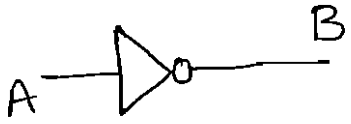
X	Y	Z=X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate :-

A NOT gate is also called an inverter is a single input, single output logic circuit whose output is always the complement of the input. That is a low input produces a high output, high input produces a low output.

Truth table.

A	B
0	1
1	0



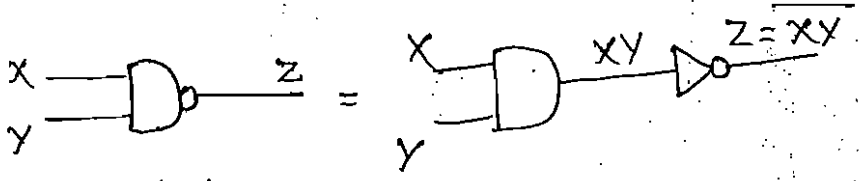
Symbol

NAND gate :-

A NAND gate is equivalent to AND gate followed by a NOT gate: The output of NAND gate is low when all inputs are in high state. In all other conditions the output is high.

Truth table.

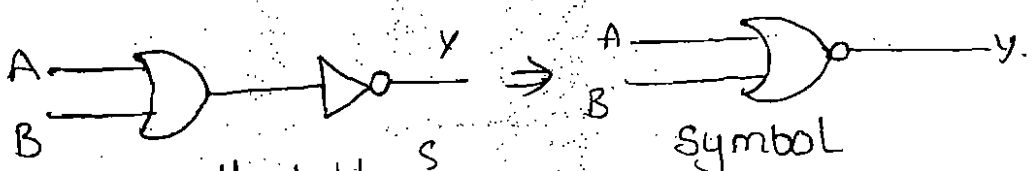
X	Y	$Z = \overline{XY}$
0	0	1
0	1	1
1	0	1
1	1	0



Symbol

NOR Gate :-

The term NOR implies OR Gate followed by a NOT Gate. The output of a NOR Gate is Logic 1 when all the inputs are logic '0'. remaining all other conditions the output is Logic '0'.



Truth table

Symbol

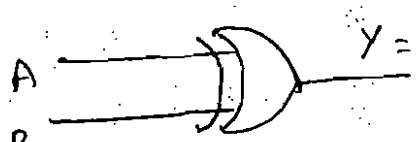
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

$Y = \overline{A+B}$

EX-OR Gate :- (Exclusive -OR Gate)

The output of an EX-OR Gate is a Logic 1 when the two inputs are different logic and Logic '0' when the two inputs are at the same logic.

Truth table $Y = \bar{A}B + A\bar{B}$



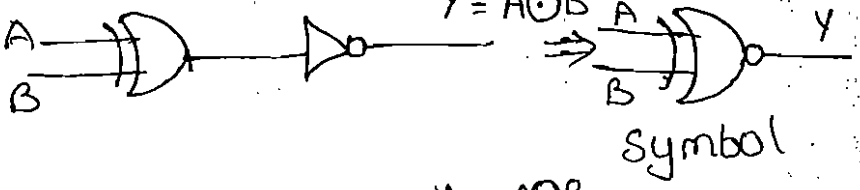
Symbol.

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

EX-NOR Gate :- (EXclusive -NOR)

The output of an EX-NOR Gate is a Logic 1 when the two inputs are same and Logic 0 when the two inputs are different. EX-NOR Gate is EX-OR

Gate followed by NOT Gate



Symbol

$Y = A \odot B$.

Truth table

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

Error-detecting codes :-

When the digital information in the binary form is transmitted from one system to another system an error may occur.

This means a signal '0' may change to '1' or '1' change to '0'.

Parity bit :-

To maintain data integrity between transmitter and receiver, extra bit or more than one bit are added in the data. These extra bits allow the detection and some times correction of error in the data. Those extra bits are called parity bits.

→ There are two types of parity - odd and even parity.

→ For odd parity, the parity bit is set to a 0 or 1 at the transmitter such that the total number of '1' bits in the word including the parity bit is an odd number.

→ For even parity, the parity bit is set to a 0 or 1 at the transmitter such that the total number of '1' bits in the word including the parity bit is an even number.

decimal.	Binary	odd parity	even parity.
0	0000	1	0
1	0001	0	1
2	0010	0	1
3	0011	1	0
4	0100	0	1
5	0101	1	0
6	0110	1	0
7	0111	0	1

⇒ In an even - parity scheme, which of the following words contain an error.

(a) 10110110
The no. of 1's in the word is odd (5). So, there is an error.

(b) 11011011
The no. of 1's in the word is even (6), so, there is no error.

(c) 10101000
The no. of 1's in the word is odd (3), so there is an error.

⇒ In an odd - parity scheme, which of the following words contain an error.

(a). 11011101
The no. of 1's in the word is even (6). So, there is an error.

(b) 11011010
The no. of 1's in the word is odd (5). So there is no error.

(c) 11011000
The no. of 1's in the word is even (4), so there is ~~no error~~ an error.

Hamming code :- (Error - correcting codes).

A code is said to be an error - correcting code, which allow error detection and correction are called error detecting and correcting codes.

→ Hamming code not only provides the detection of a bit error, but also identifies which bit is in error.

→ The code uses a number of parity bits located at certain positions in the code group.

1-bit Hamming code :-

To transmit four data bits, three parity bits located at positions 2^0, 2^1 and 2^2 from left are added to make a 7-bit code word which is then transmitted. The word format is.

P_1 P_2 D_3 P_4 D_5 D_6 D_7.

D for the data bits

P for the parity bits.

P_1 is the to be set to a '0' or a '1' so that it establishes even parity over bits 1,3,5 and 7 (P_1, D_3, D_5, D_7)

P_2 is to be set to a '0' or a '1' so that it establishes even parity over bits 2,3,6,7 (P_2, D_3, D_6, D_7)

P_4 is to be set to a '0' or a '1' so that it establishes even parity over bits 4,5,6,7 (P_4, D_5, D_6, D_7).

At the receiving end, the message received in the Hamming code is decoded to see if any errors have occured.

Bits 1,3,5,7, bits 2,3,6,7, and bits 4,5,6,7 are all checked for even parity.

→ If they all check out, there is no error.

→ if there is an error, the error bit can be located by forming a 3-bit binary number.

$$C_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7.$$

$$C_2 = P_2 \oplus D_3 \oplus D_5 \oplus D_7$$

$$C_3 = P_4 \oplus D_5 \oplus D_6 \oplus D_7.$$

EX:- ~~1010~~ Encode data bits 1010 into the 7-bit even-parity hamming code.

The bit pattern.

P_1	P_2	D_3	P_4	D_5	D_6	D_7
		1		0	1	0

Bits 1,3,5,7 (P_1 1 1 1) must have even parity. So P_1 must be a '1'

Bits 2,3,6,7 (P_2 1 1 0) must have even parity. So P_2 must be a '0'

Bits 4,5,6,7 (P_4 0 1 0) must have even parity. So P_4 must be a '1'.

The final code with parity bits is

P_1	P_2	D_3	P_4	D_5	D_6	D_7
1	0	1	1	0	1	0

→ This data transmitted through a noisy channel.

The code is

1 0 1 0 0 1 0 (error occured).

P_1 P_2 D_3 P_4 D_5 D_6 D_7

To correct the code by using.

$C_1 = 1 \oplus 1 \oplus 0 \oplus 0$ = 0 put a '0' in the 1's position.

$C_2 = 0 \oplus 1 \oplus 1 \oplus 0$ = 0 put a '0' in the 2's position.

$C_3 = 0 \oplus 0 \oplus 1 \oplus 0$ = 1 put a '1' in the 4's position.

error in the 4th position.

1011010

12-bit hamming code :-

To transmit eight data bits, four parity bits located at positions $2^0, 2^1, 2^2$ and 2^3 from left are added to make a 12-bit code word which is then transmitted.

$P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7\ P_8\ D_9\ D_{10}\ D_{11}\ D_{12}$

P_1 is set to a '0' or '1' ($P_1, D_3, D_5, D_7, D_9, D_{11}$).

Similarly. P_2 ($P_2, D_3, D_6, D_7, D_{10}, D_{11}$).

P_4 ($P_4, D_5, D_6, D_7, D_{12}$)

P_8 ($P_8, D_9, D_{10}, D_{11}, D_{12}$).

Example:- given - 01011010, generate the 12-bit code.

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}
		0		1	0	1		1	0	1	0

$P_1 (P_1, 0, 1, 1, 1, 1)$ even parity, so $= P_1 = 0$.

$P_2 (P_2, 0, 0, 1, 0, 1)$ even parity, so $= P_2 = 0$

$P_4 (P_4, 1, 0, 1, 0)$ even parity, so $= P_4 = 0$

$P_8 (P_8, 1, 0, 1, 0)$ even parity, so $= P_8 = 0$

000010101010 (final data).

15-bit hamming code :-

To transmit eleven data bits, four parity bits located at positions $2^0, 2^1, 2^2$ and 2^3 from left are added to make a 15-bit code word which is then transmitted.

$P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7\ P_8\ D_9\ D_{10}\ D_{11}\ D_{12}\ D_{13}\ D_{14}\ D_{15}$

P_1 is set to be '0' or '1' $(P_1, D_3, D_5, D_7, D_9, D_{11}, D_{13}, D_{15})$

P_2 is set to be '0' or '1' $(P_2, D_3, D_6, D_7, D_{10}, D_{11}, D_{14}, D_{15})$

P_4 is set to be '0' or '1' $(P_4, D_5, D_6, D_7, D_{12}, D_{13}, D_{14}, D_{15})$.

P_8 is set to be '0' or '1' $(P_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15})$

Example :-

11-bit group 01101110101 find out the final code.

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}
		0		1	1	0		1	1	1	0	1	0	1

$P_1 (P_1, 0, 1, 0, 1, 1, 1, 1)$ even parity $P_1 = 1$

$P_2 (P_2, 0, 1, 0, 1, 1, 0, 1)$ even parity $P_2 = 0$

$P_4 (P_4, 1, 1, 0, 0, 1, 0, 1)$ even parity $P_4 = 0$

$P_8 (P_8, 1, 1, 1, 0, 1, 0, 1)$ even parity $P_8 = 1$

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}
1	0	0	0	1	1	0	1	1	1	1	0	1	0	1

The final code is

100011011110101

Standard sop and pos :-

Sop (sum of products):-

product term → multiplying two or more variable is called product term. (EX:- ABC, $\bar{A}B\bar{C}$, AB, $AB\bar{C}\bar{D}E$)

sop → summiation of product term is called sop.

EX:- $AB+\bar{B}A+\bar{A}C$

It is also called the Disjunctive normal form (DNF).

Standard sop :-

It is also called Disjunctive canonical form (DCF) It is also called the Expanded sum of products form or canonical sum-of-products form. In this form, the function is the sum of a number of product terms where each product term contains all variables either in complemented or uncomplemented form.

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

Sop to Standard SOP :-

→ write down all the terms

→ If one or more variables are missing in any term, expand that term by multiplying it with the sum of each one of the missing variable and its complement.

→ drop out the redundant term.

→ Replace the variables by 1's & 0's.
complemented variables by 0's
non-complemented variables by 1's.

* Expand $f(A,B) = \bar{A}B + B$.

The given expression is a two-variable function. In second term, the variable A is missing. So multiply it by $[A+\bar{A}]$.

$$\bar{A}B + B = \bar{A}B + B(A+\bar{A})$$
$$= \bar{A}B + AB + \bar{A}B \rightarrow \text{Drop out redundant}$$
$$= \bar{A}B + AB.$$
$$= 01 + 11 \quad \rightarrow \text{non-complemented} \rightarrow '1'$$
$$= m_1 + m_3. \quad \rightarrow \text{complemented} \rightarrow '0'.$$
$$= \Sigma m(1,3).$$

* $f(A,B,C) = AB\bar{C} + A\bar{B}C + AB + BC$
$$= AB\bar{C} + A\bar{B}C + AB(C+\bar{C}) + BC(A+\bar{A})$$
$$= AB\bar{C} + A\bar{B}C + ABC + AB\bar{C} + ABC + \bar{A}BC$$
$$= \bar{A}BC + ABC + A\bar{B}C$$
$$= 011 + 111 + 101$$
$$= m_3 + m_7 + m_5$$
$$= \Sigma m(3,5,7)$$

* $f(A,B,C) = A + AB + BCA.$
$$= A(B+\bar{B})(C+\bar{C}) + AB(C+\bar{C}) + BCA$$
$$= AB + A\bar{B}(C+\bar{C}) + ABC + AB\bar{C} + ABC.$$
$$= ABC + A\bar{B}C + A\bar{B}\bar{C} + AB\bar{C} + ABC + AB\bar{C} + ABC$$
$$= ABC + A\bar{B}C + A\bar{B}\bar{C} + AB\bar{C}$$
$$= 111 + 101 + 100 + 110 = m_7 + m_5 + m_4 + m_6$$
$$= \Sigma m(4,5,6,7).$$

POS (product of sum) :-

Sum term :- Sum of two or more variable is called sum term. (Ex :- (A+B), (A+B+C).

POS → product of sum terms is called POS.

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

It is also called conjunctive normal form (CNF).

Standard POS :-

It is also called conjunctive canonical for (CCF). It is also called the expanded product of sum form or canonical product of sum form. In this form, the function is product of a number of sum terms. where each sum term contains all variables either in complemented or uncomplemented form.

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

POS to standard POS :-

→ write down all ther terms.

→ If one or more variables are missing in any term expand that term by adding the product of each of the missing variable and its complement.

→ drop out the redundant one.

→ Replace the complemented variables by 1's and the non-complemented variables by 0's

* Expand $f(AB) = (\bar{A}+B)(A)$.

The given expression is a two-variable function. In second term, the variable B is missing. So add it by $(B \cdot \bar{B})$.

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

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

$$= (10)(11)(01)$$

$$= M_1, M_2, M_3$$

$$= \Pi M(1,2,3).$$

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

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

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

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

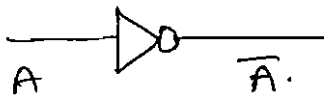
$$(000)\ (001)\ (010)\ (011)\ (100)\ (101)$$

$$= M_0, M_1, M_2, M_3, M_4, M_5.$$

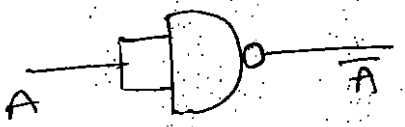
$$= \Pi M(0,1,2,3,4,5).$$

NAND - NAND Realization :-

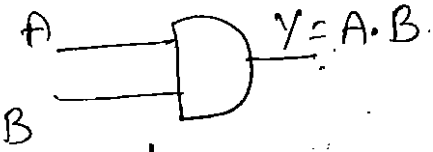
1) NOT Gate

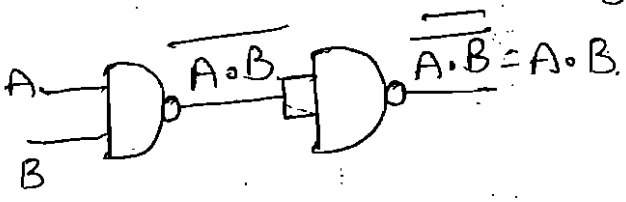


Using NAND gate.



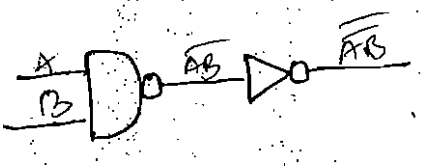
2) AND Gate.

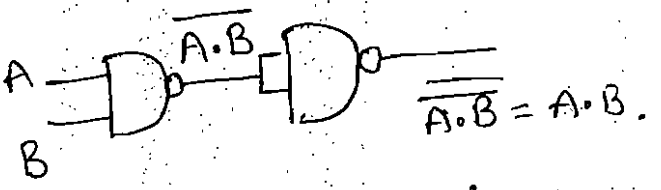




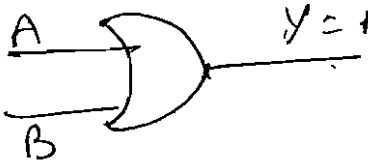
Step :- 1 add bubble on output

Step :- 2 add one NOT gate

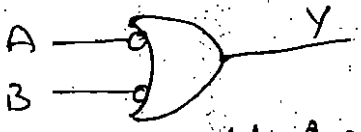




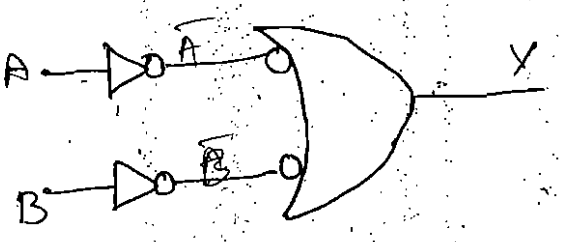
3) OR Gate

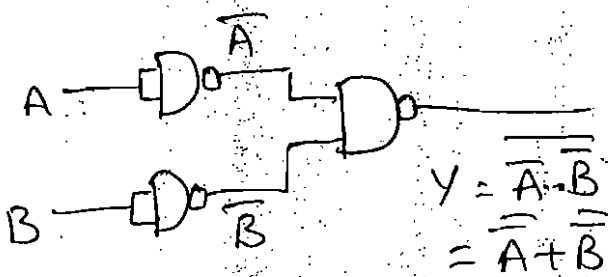


Step :- 1 add bubble on input



step 2 :- add not gate



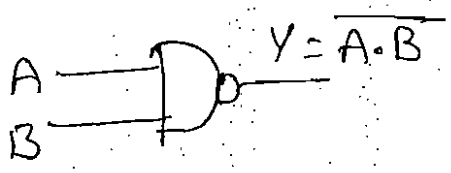


$Y = \overline{\overline{A} \cdot \overline{B}}$

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

$= A + B.$

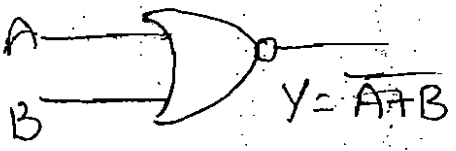
4) NAND gate

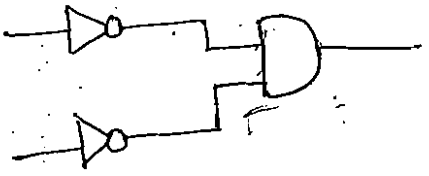


5) NOR Gate

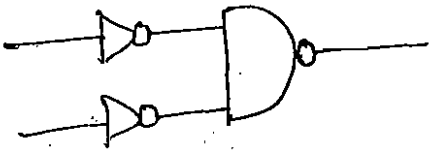
$Y = \overline{A+B}$.

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

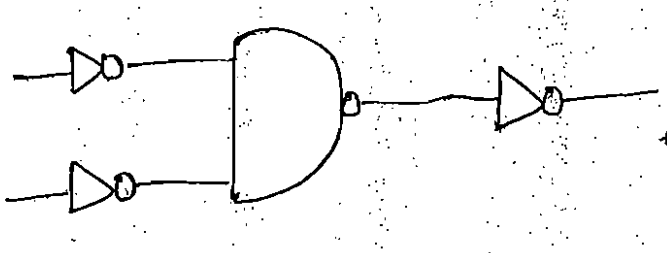




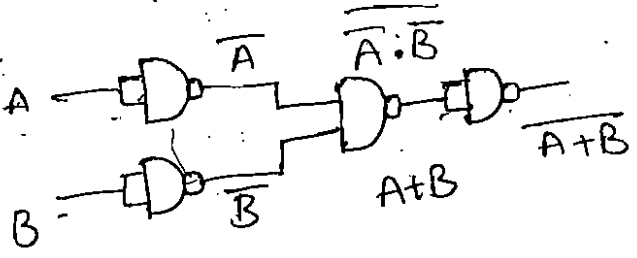
Step:-1



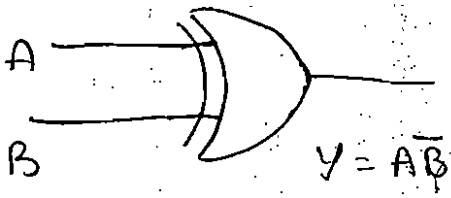
Step :- 2

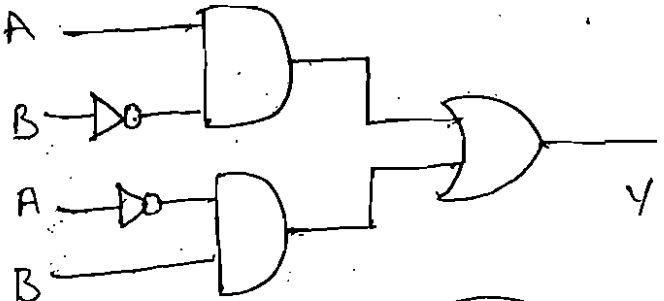


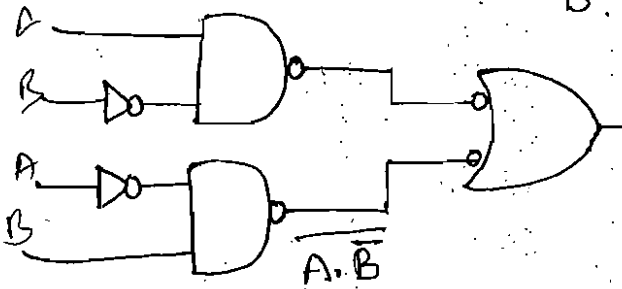
Step 3 :-

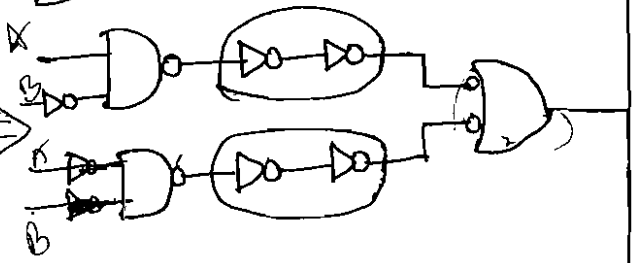


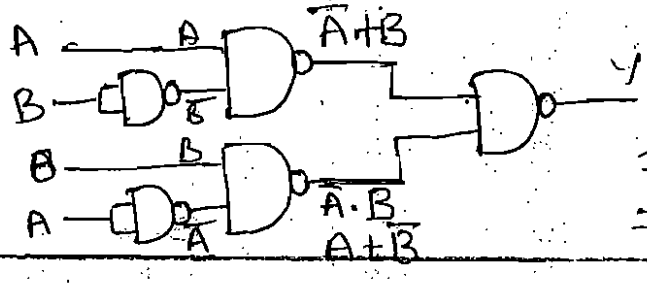
6) EX-OR Gate









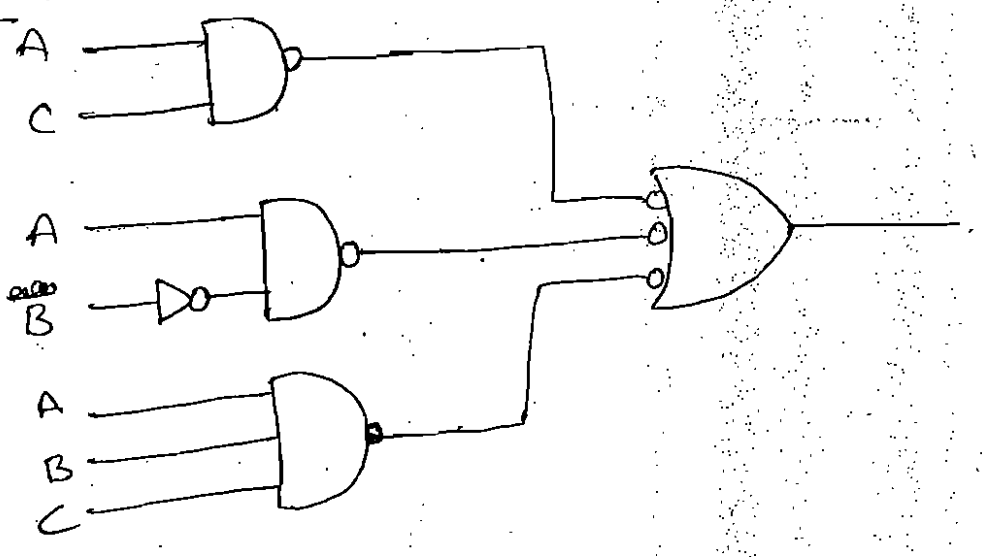


$Y = \overline{(\overline{A}+B)(A+\overline{B})}$

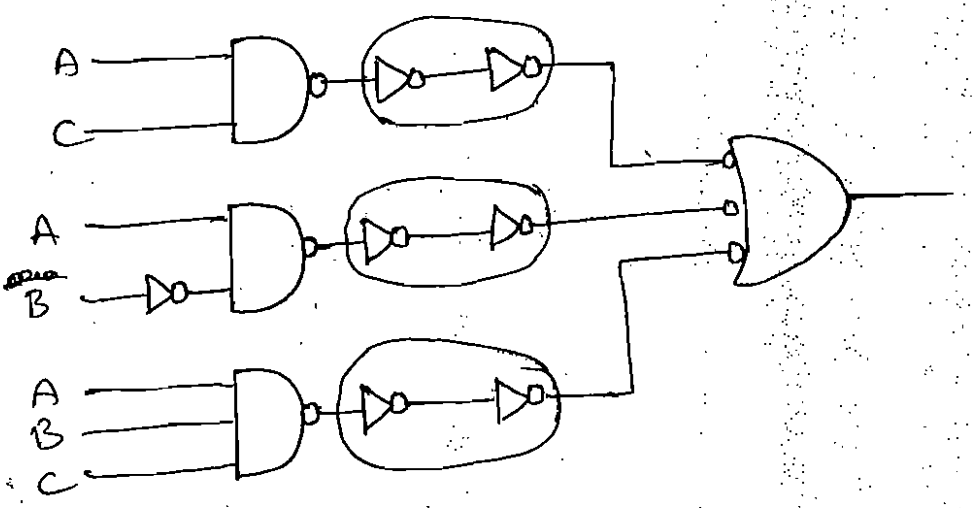
$= \overline{(\overline{A}+B)} + \overline{(A+\overline{B})}$

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

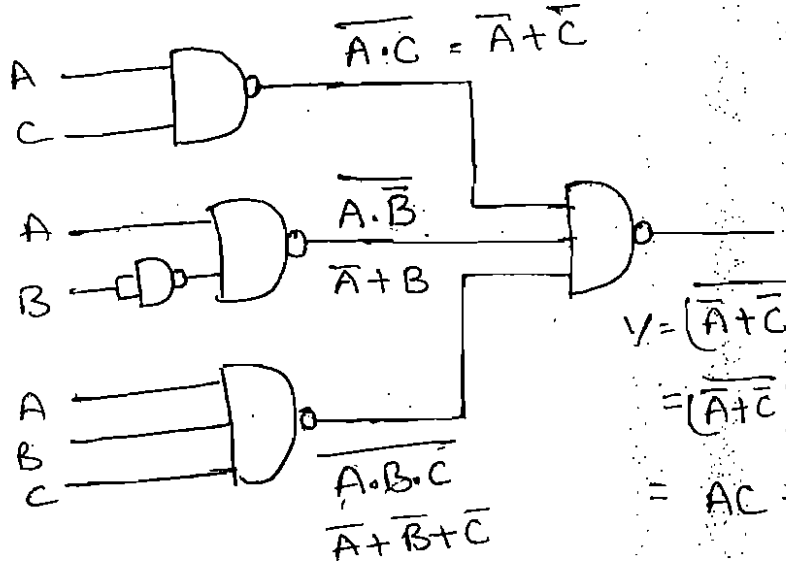
Step 1:-



Step 2:-



Step 3:-



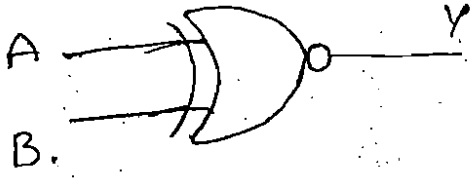
$$Y = \overline{(\overline{A}+\overline{C})(\overline{A}+B)(\overline{A}+\overline{B}+\overline{C})}$$

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

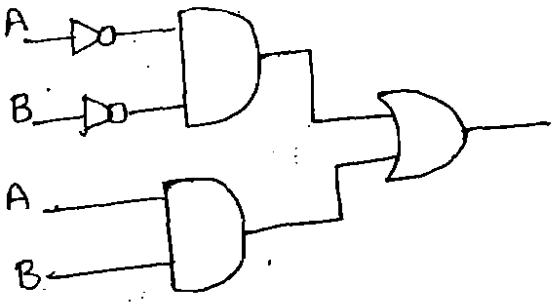
$$= AC + A\overline{B} + \overline{\overline{A}} \cdot \overline{\overline{B}} \cdot \overline{\overline{C}}$$

$$= AC + A\overline{B} + ABC.$$

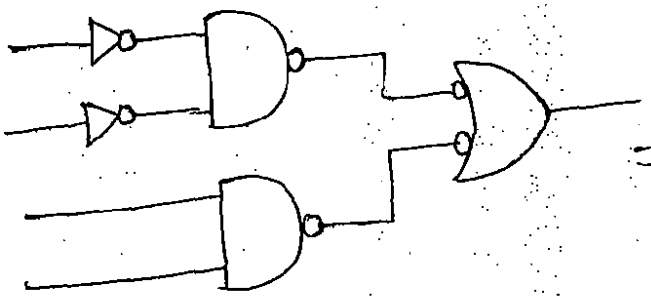
7) EX-NOR Gate :-

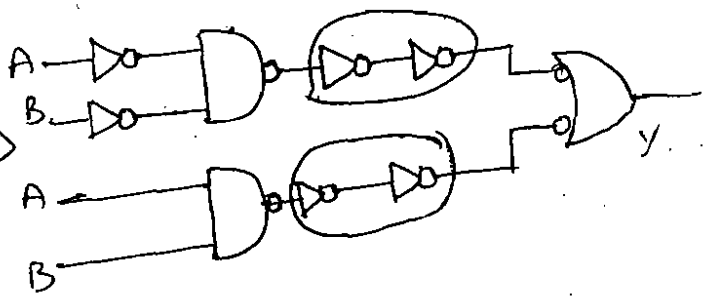


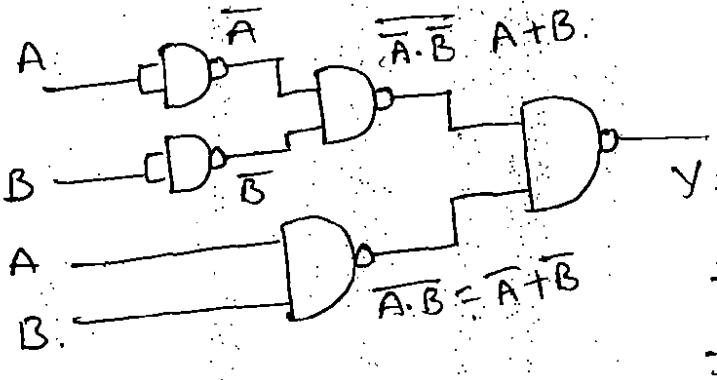
$Y = \overline{A}B + A\overline{B}$.



Step 1 :-







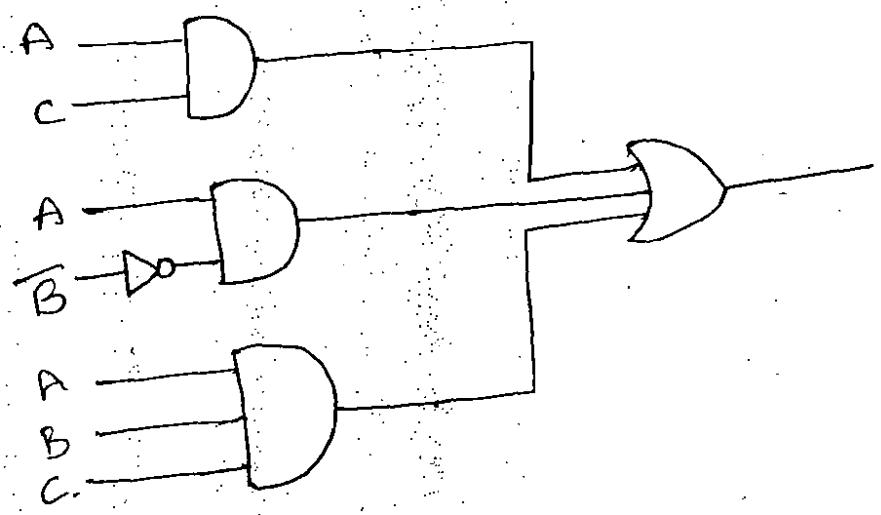
$Y = \overline{(A+B)(\overline{A}+\overline{B})}$

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

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

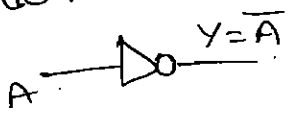
* Implement a given Boolean Expression by using NAND gate.

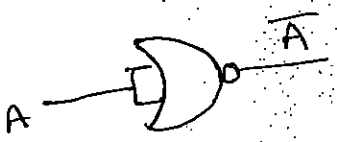
$Y = AC + A\overline{B} + ABC$.



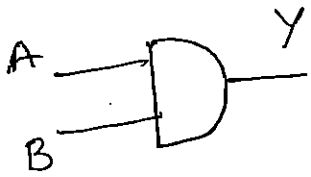
NOR - NOR Realization :-

1) NOT Gate

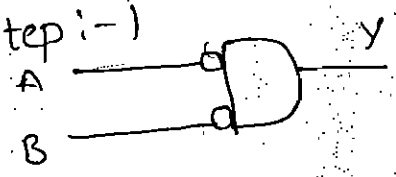




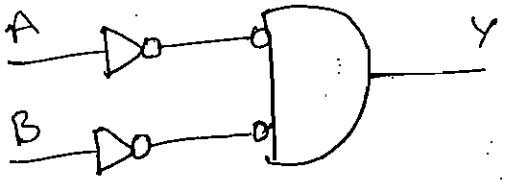
2) AND Gate.



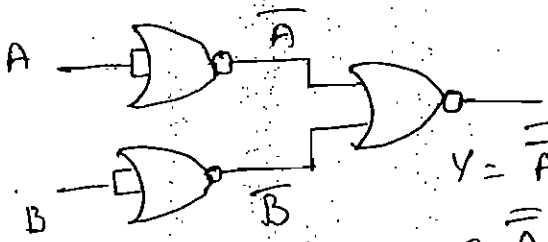
Step :-1



Step 2



Step 3

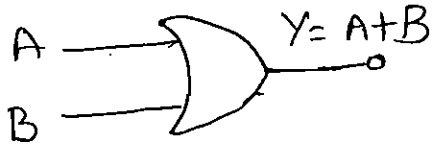


$Y = \overline{\bar{A} + \bar{B}}$

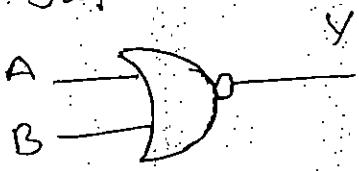
$= \bar{\bar{A}} \cdot \bar{\bar{B}}$

$= A \cdot B.$

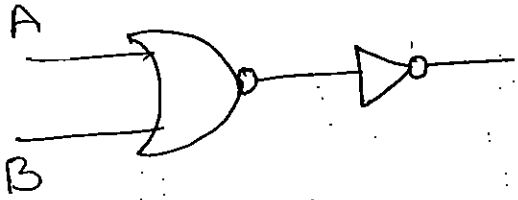
3) OR Gate

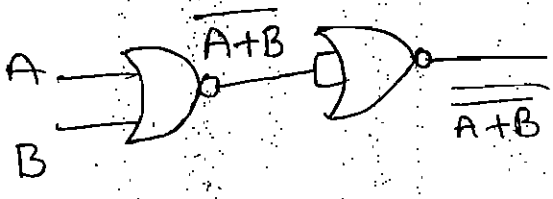


Step :-1



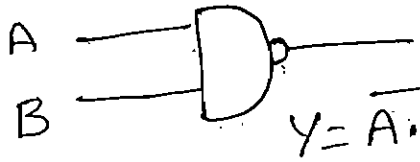
Step :- 2





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

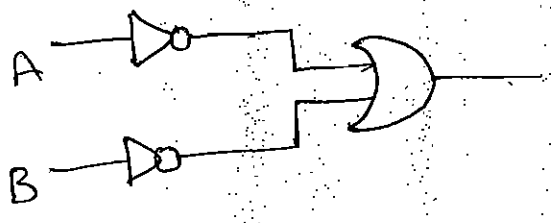
4) NAND Gate



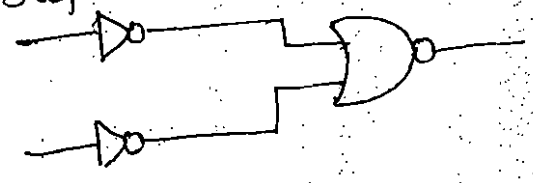
$Y = \overline{A \cdot B}$

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

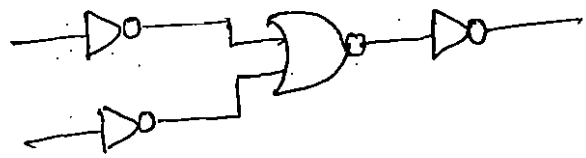
Step :- 1



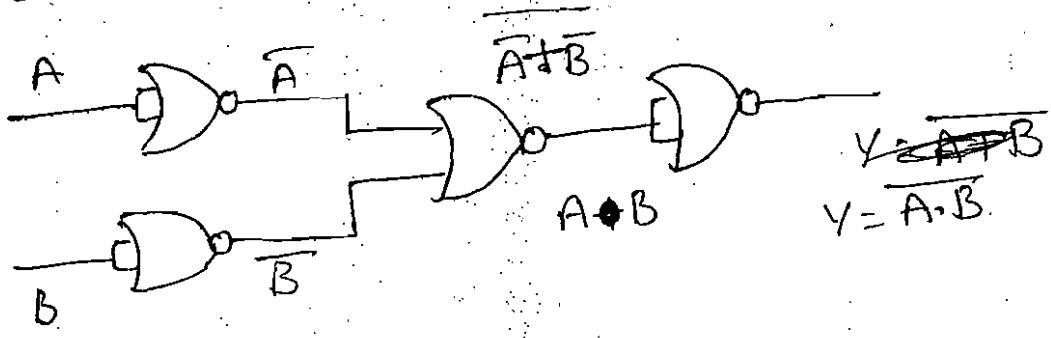
Step 2:-



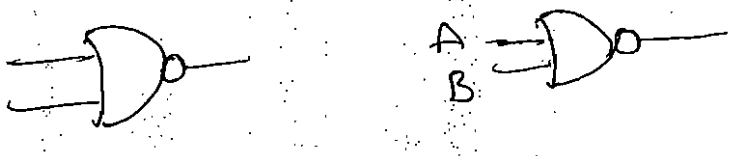
Step 3



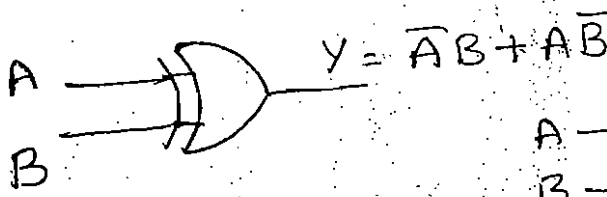
Step 4 :-



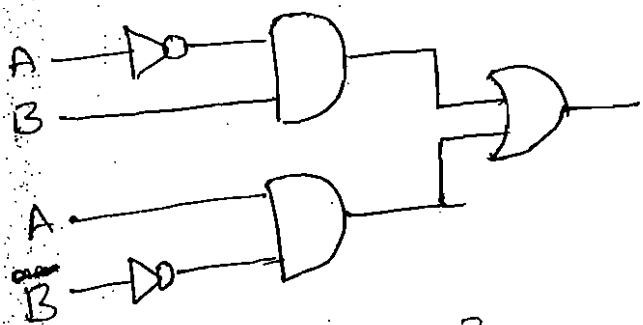
5) NOR Gate :-



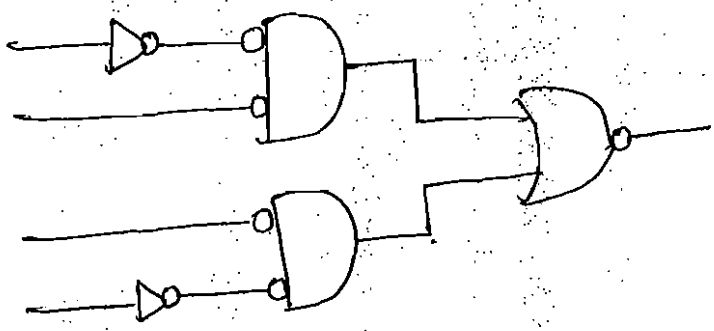
6) EX-OR Gate :-



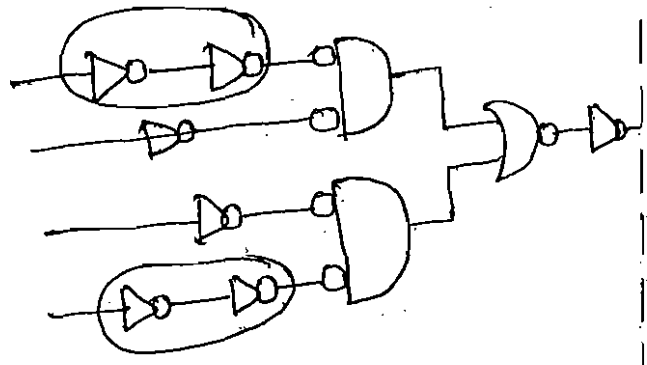
Step :- 1



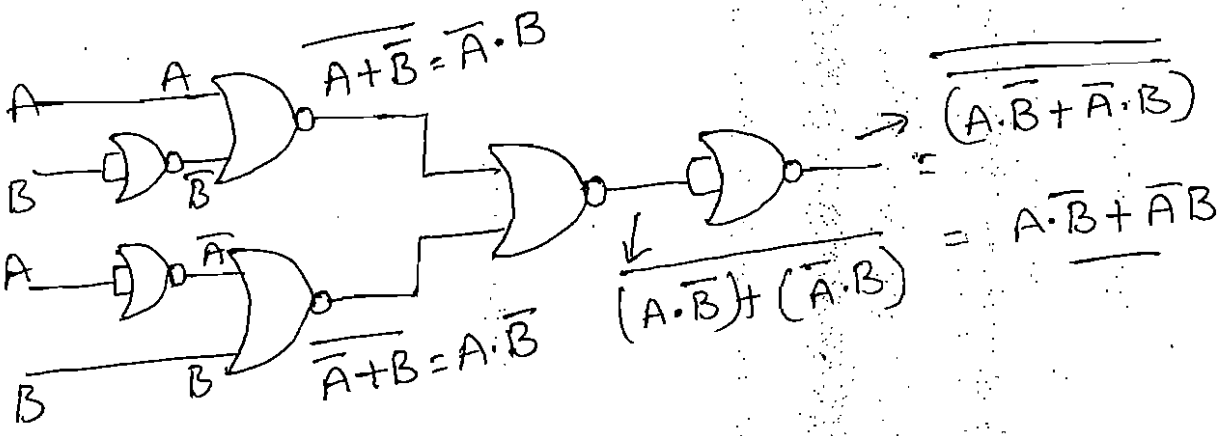
Step :- 2



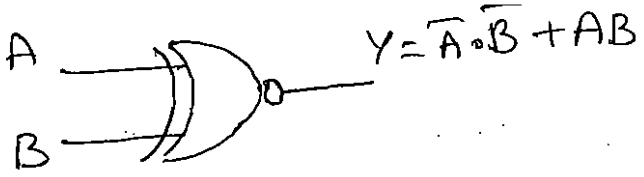
Step :- 3



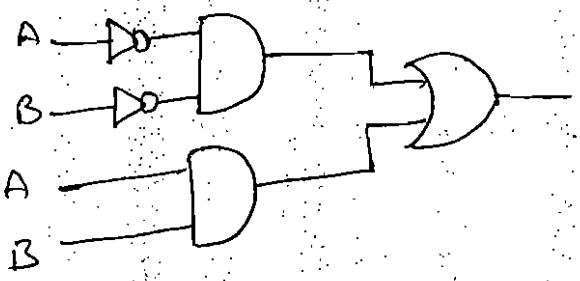
Step 4 :-



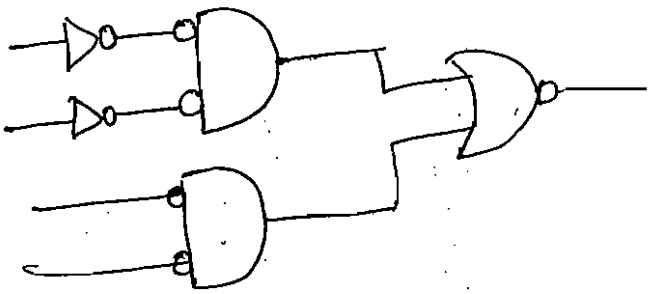
7) EX-NOR gate :-



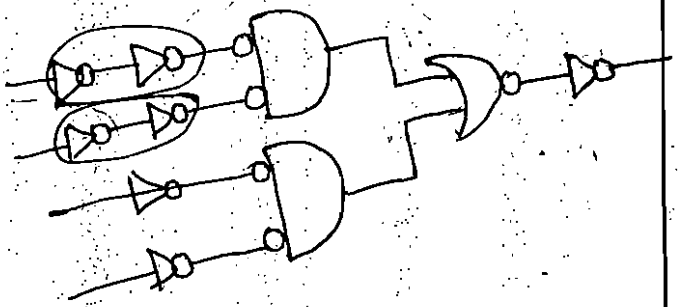
Step 1 :-



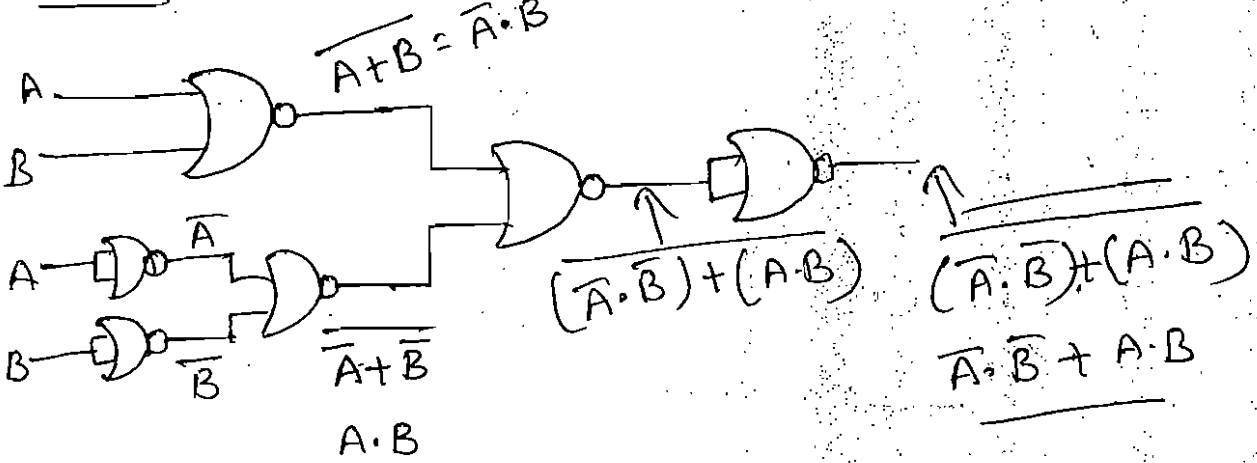
Step 2 :-



Step 3 :-

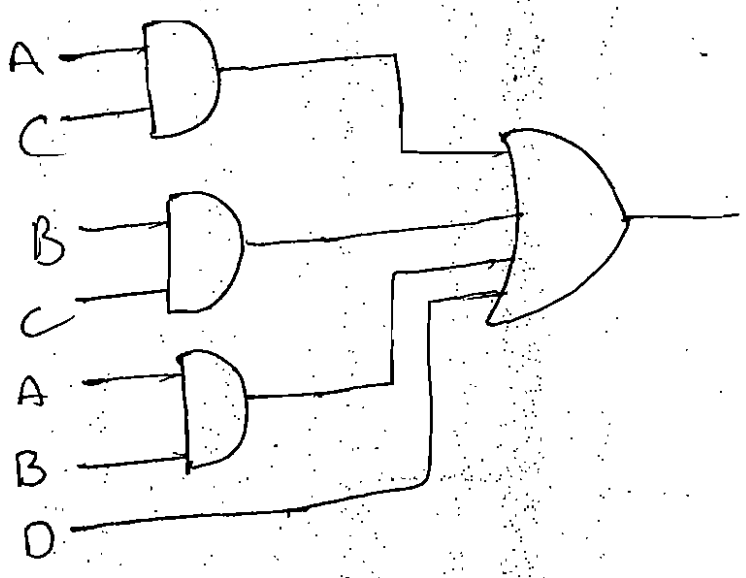


Step 4 :-

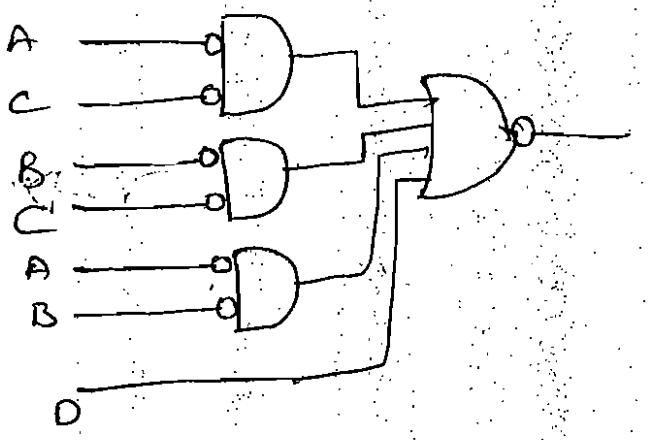


* $Y = AC + BC + AB + D$, implement Boolean Expression by using NOR Gate.

$Y = AC + BC + AB + D$,

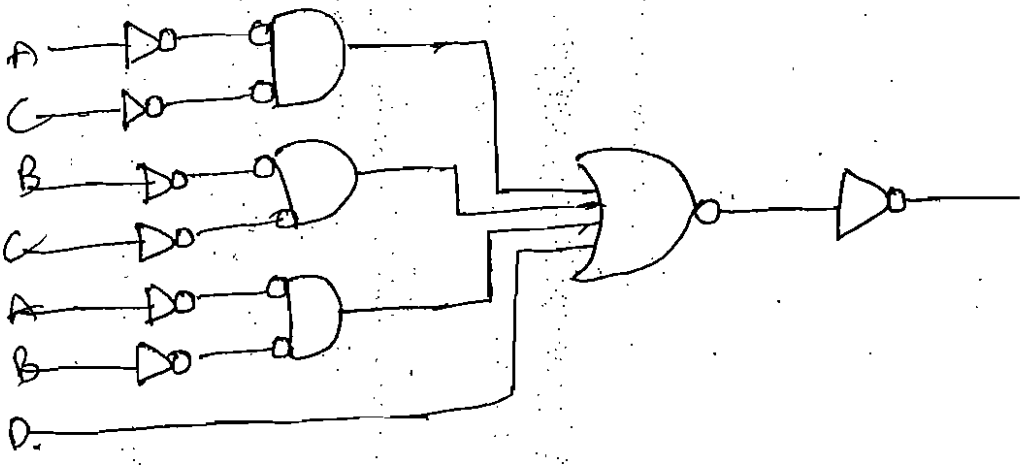


Step :- 1

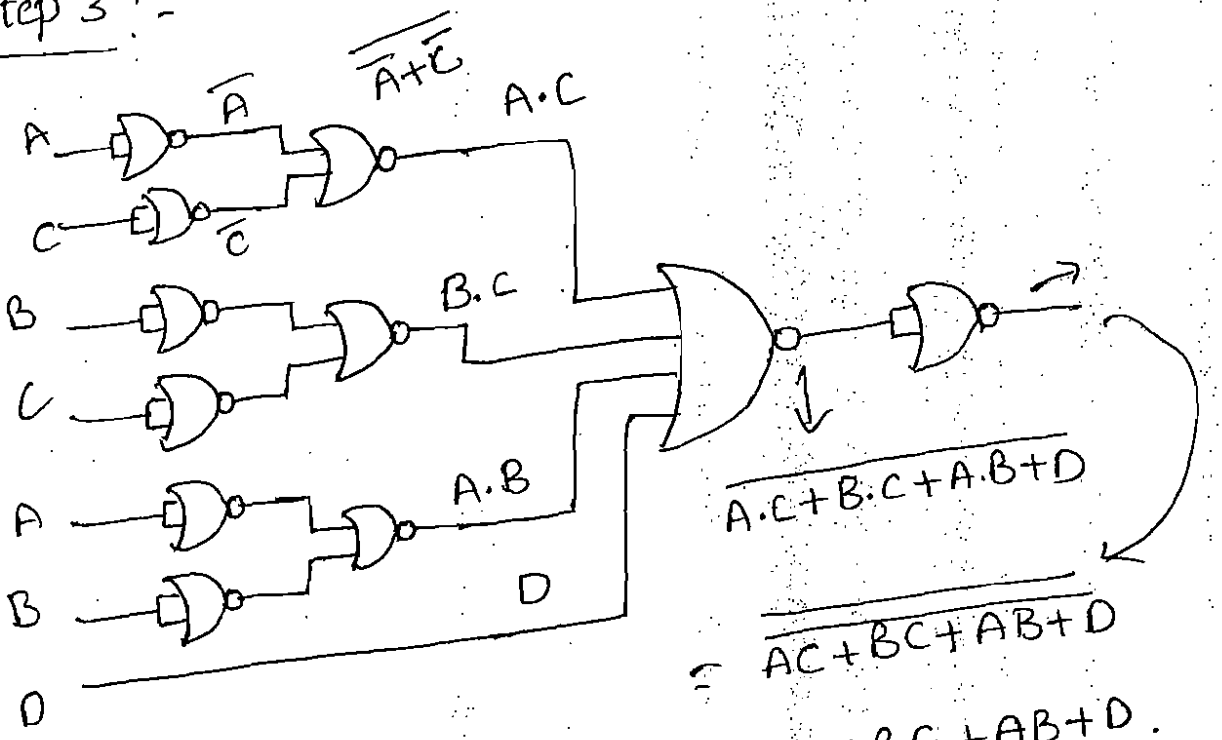


Step :- 2

Step :- 2



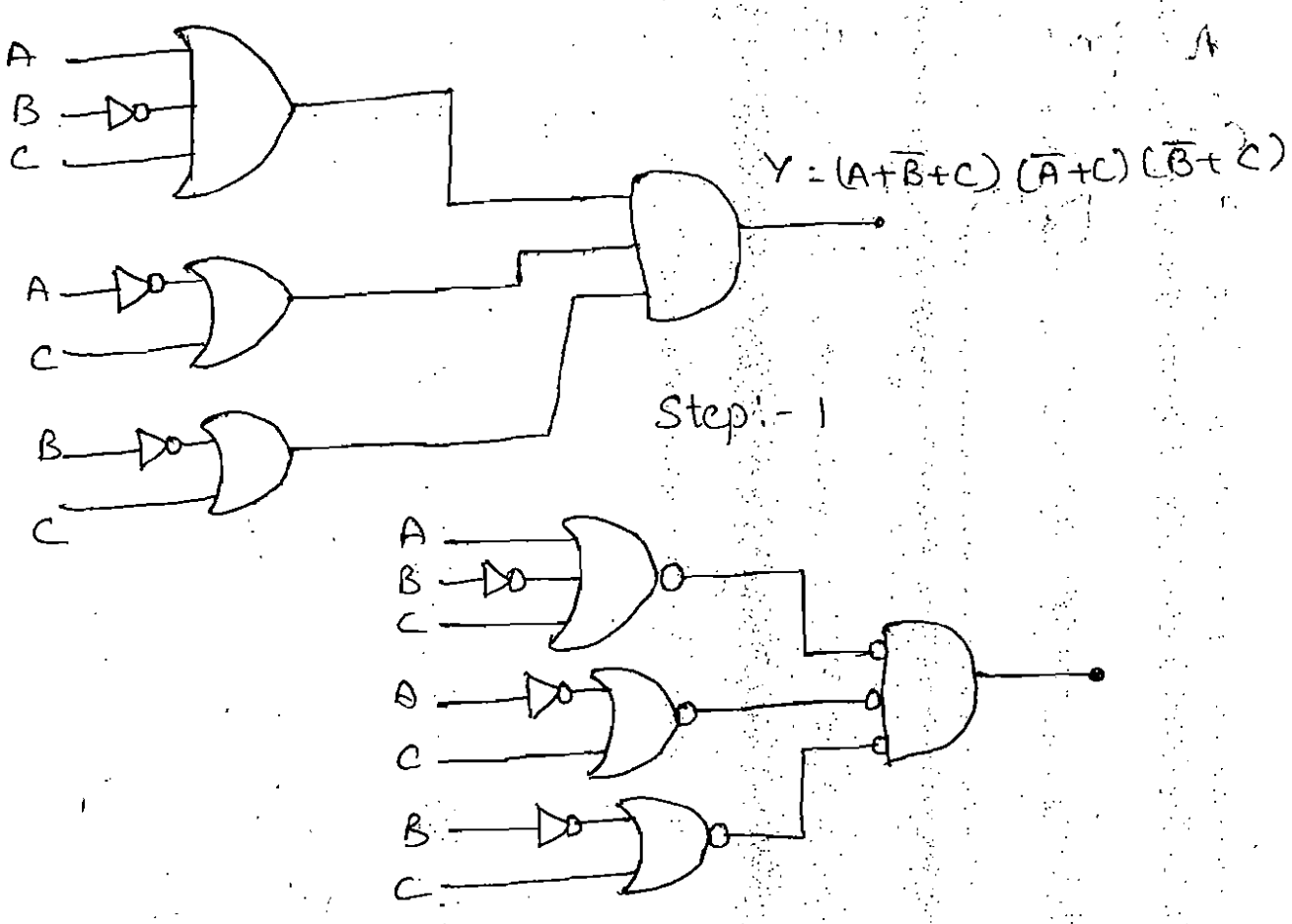
Step 3 :-



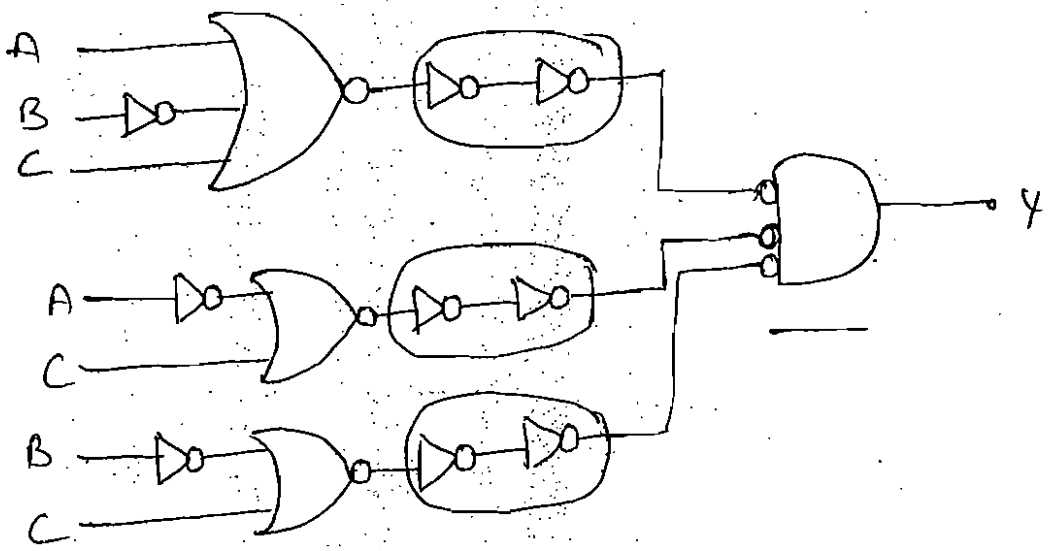
$= \overline{\overline{AC+BC+AB+D}}$

$= AC+BC+AB+D.$

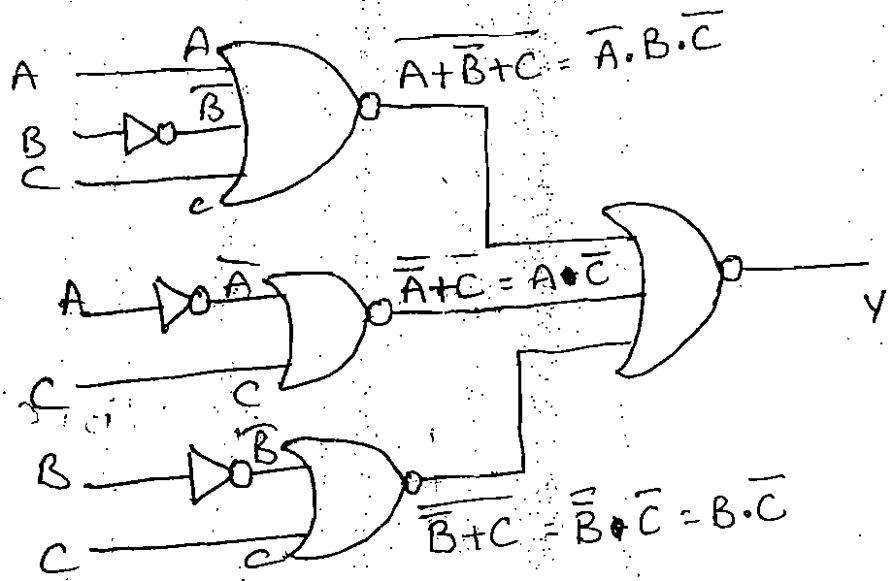
* implement the Boolean Expression by using NOR Gate.
$(A+\bar{B}+C)\ (\bar{A}+C)\ (\bar{B}+C)$.



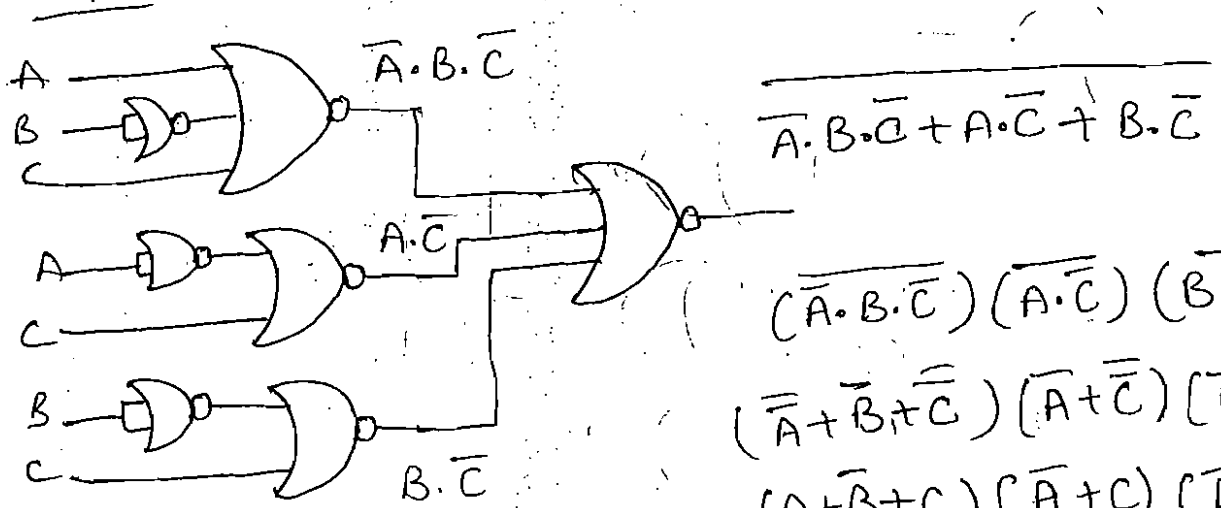
Step 2 :-



Step 3 :-



Step 4 :-



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

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

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

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

MINIMIZATION TECHNIQUES

→ Binary Logic is used in all of today's digital computers and devices, the cost of the circuits that implement it is an important factor.

→ Finding simpler and cheaper, but equivalent, realizations of a circuit must be good. To reducing the overall cost of the design.

→

Boolean theorems :-

1. Complementation laws :-

Complement means, to change 0's to 1's and 1's to 0's.

Law 1 = $\overline{0} = 1$.

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

Law 3 = $A = 0 \rightarrow \overline{A} = 1$

Law 4 = $A = 1 \rightarrow \overline{A} = 0$.

Law 5 = $\overline{\overline{A}} = A$ (Double complementation does not change the function).

2. AND laws :-

Law 1 = $A \cdot 0 = 0$

Law 2 = $A \cdot 1 = A$

Law 3 = $A \cdot A = A$

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

3. OR laws :-

Law 1 = $A + 1 = 1$

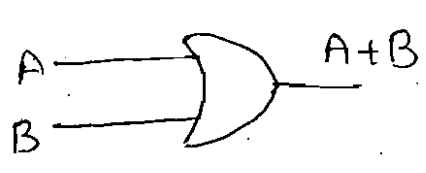
Law 2 = $A + 0 = A$

Law 3 : $A + A = A$

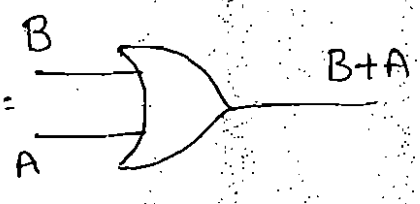
Law 4 : $A + \overline{A} = 1$

4. Commutative laws :-

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



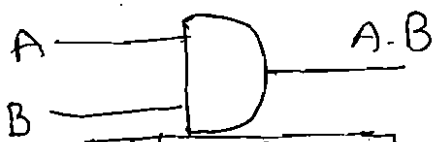
=



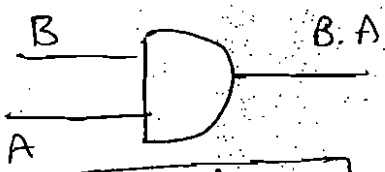
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

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

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



=

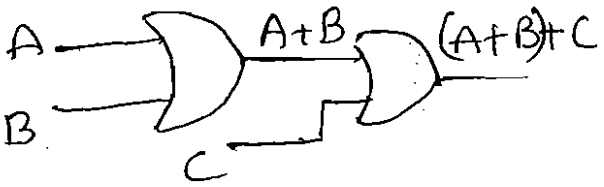


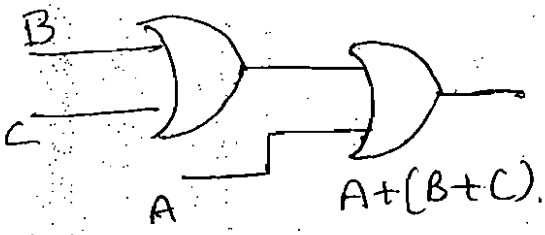
A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	B·A
0	0	0
0	1	0
1	0	0
1	1	1

5. Associate laws :-

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



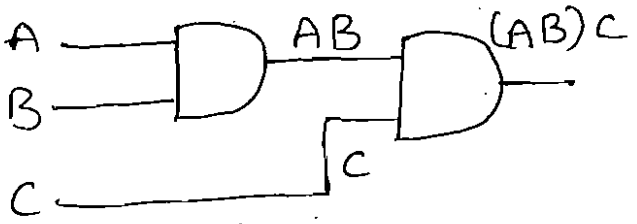


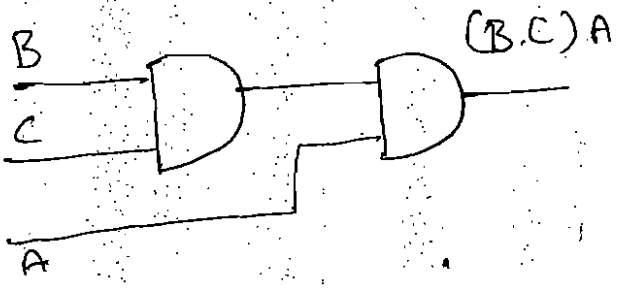
A B

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

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

Law 2 : $(A.B)C = A(B.C)$.



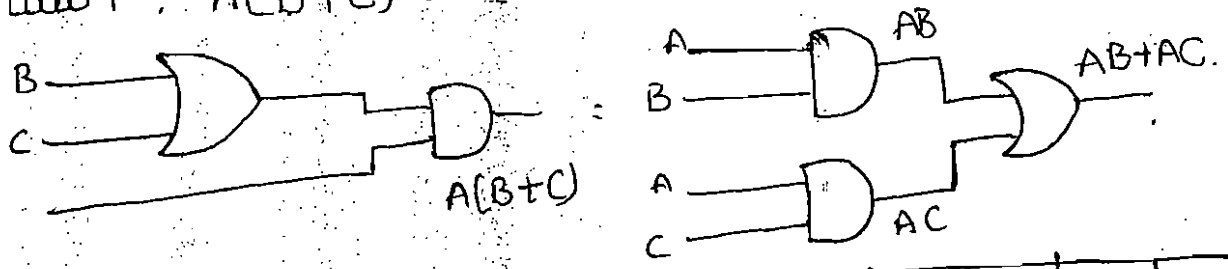


A	B	C	AB	(AB) C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

A	B	C	B.C	A(BC)
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

6. Distributive Laws :-

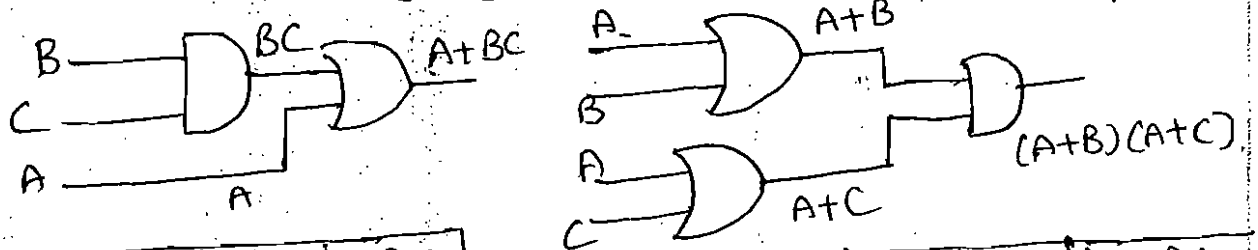
Law 1 : $A(B+C) = AB + AC$



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

A	B	C	AB	AC	AB+AC
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	1	1
1	1	1	1	1	1

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



A	B	C	BC	A+BC
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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

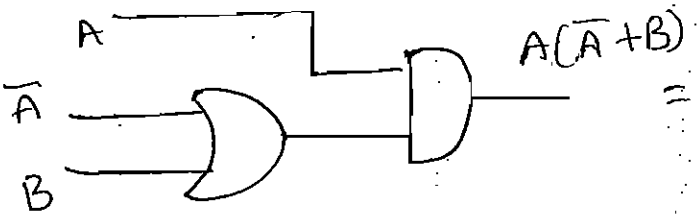
7. Redundant literal Rule :-

Law 1 : $A(\bar{A}+B) = AB$.

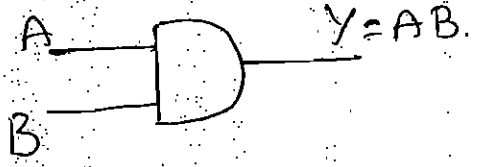
$= A(\bar{A}+B)$

$= A.\bar{A} + AB$

$= AB$



=



A	B	$\bar{A}+B$	$A(\bar{A}+B)$
0	0	1	0
0	1	1	0
1	0	0	0
1	1	1	1

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

Law 2 : $A+\bar{A}B = A+B$.

$= A+\bar{A}B$

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

$= (A+B)$

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

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

8. Idempotence laws :-

Idempotence means the same value.

Law 1 = $A \cdot A = A$

if A=0 then 0.0 = 0

if A=1 then 1.1 = 1

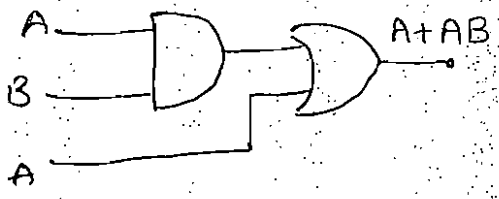
Law = $A+A = A$

if A=0 then 0+0 = 0

if A=1 then 1+1 = 1

9. Absorption Laws :-

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



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

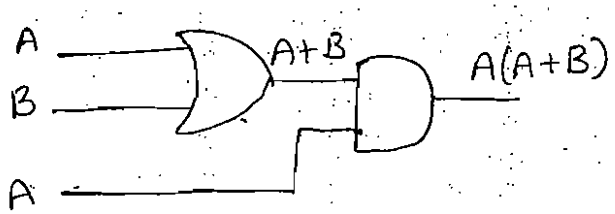
$= A+A\cdot B$

$= A(1+B)$ $\quad \because 1+B = 1$

$= A\cdot 1$

$= A.$

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



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

$= A\cdot A + AB$

$= A + AB$

$= A(1+B)$

$= A.$

$= A(A + \text{any term})$

$= A$

10. Consensus Theorem :-

Theorem 1 : $AB + \bar{A}C + BC = AB + \bar{A}C.$

L.H.S $\rightarrow AB + \bar{A}C + BC$

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

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

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

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

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

L.H.S $\rightarrow$ $AB + \bar{A}C + BCD$

$AB + \bar{A}C + BCD(A + \bar{A})$

$AB + \bar{A}C + ABCD + \bar{A}BCD$

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

$AB + \bar{A}C$.

Theorem 2 : $(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}+C)$.

L.H.S : $(A+B)(\bar{A}+C)(B+C)$

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

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

$ABC + BC + \bar{A}BC + AC + BC + \bar{A}B$

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

$= BC + \bar{A}B + AC(1 + B)$

$= AC + BC + \bar{A}B$.

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

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

$= \bar{A}B + AC + BC$.

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

If a sum of products comprises a term containing A and a term containing $\bar{A}$, and a third term containing the left out literals of the first two terms, then the third term is redundant.

$\rightarrow$ The function remains the same with & without the third term removed & retained.

Transposition Theorem :-

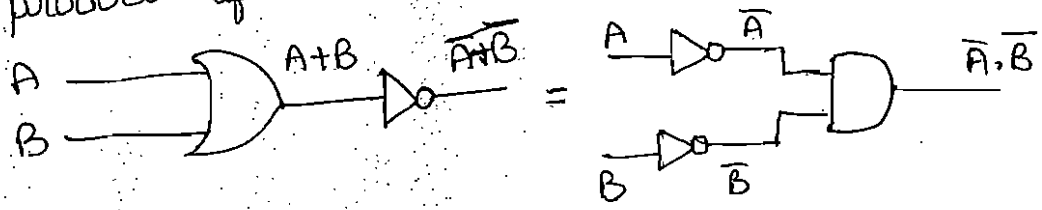
Theorem : $AB + \bar{A}C = (A+C)(\bar{A}+B)$

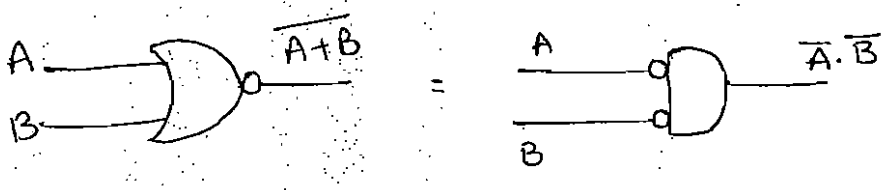
$$R.H.S = (A+C)(\bar{A}+B)$$
$$= A.\bar{A} + AB + \bar{A}C + BC$$
$$= AB + \bar{A}C + BC$$
$$= AB + \bar{A}C + BC(A+\bar{A})$$
$$= AB + \bar{A}C + ABC + \bar{A}BC$$
$$= AB(1+C) + \bar{A}C(1+B)$$
$$= AB + \bar{A}C$$

Demorgan's Theorem :-

Law I $\quad \overline{A+B} = \bar{A}.\bar{B}$

The complement of a sum of variables is equal to the product of their individual complements.





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

A	B	$\bar{A}$	$\bar{B}$	$\bar{A}.\bar{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

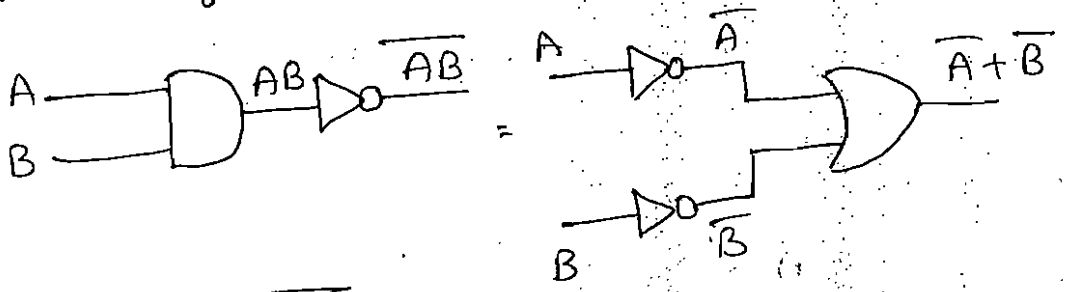
Similarly → $\overline{A+B+C+D+E} = \overline{A}\,\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}$

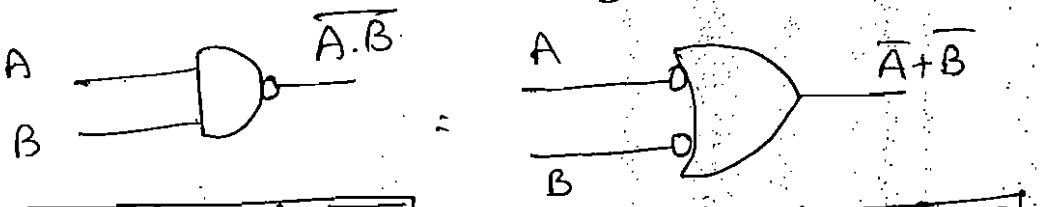
→ $\overline{\overline{A}B+C\overline{D}+\overline{E}\overline{D}} = (\overline{\overline{A}B})(\overline{C\overline{D}})(\overline{\overline{E}\overline{D}})$

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

Law 2 $\quad \overline{AB} = \overline{A}+\overline{B}$

The complement of the product of variables is equal to the sum of their individual complements.





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

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

Duality:-

When changing from one logic system to another, '0' becomes '1' and '1' becomes '0'. An AND gate becomes an OR gate and an OR gate becomes an AND gate. Given a Boolean identity, we can produce a dual identity by changing all '+' sign to '.' signs, all '.' signs to '+' signs. The variables are not complemented.

→ $A.(A+B) = A.B$ → $A+A+B = A+B$.

→ $\overline{AB}+\overline{A}+AB = 0$ → $\overline{A+B}.\overline{A}.(A+B) = 1$

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

Reducing Boolean expressions by using Boolean theorems :-

* $f = A[B+\bar{C}(\overline{AB+A\bar{C}})]$

$f = A[B+\bar{C}((\overline{AB})(\overline{A\bar{C}}))]$ (or)

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

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

$= A[B+\bar{C}[\bar{A}\bar{A}+\bar{A}C+\bar{A}\bar{B}+\bar{B}C]$ $\quad \bar{A}\cdot\bar{A}=\bar{A}$

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

$= A[B+\bar{A}\bar{C}+\bar{A}C\bar{C}+\bar{A}\bar{B}\bar{C}+\bar{B}C\bar{C}]$ $\quad C\cdot\bar{C}=0$

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

$= AB+A\bar{A}\bar{C}+A\bar{A}\bar{B}\bar{C}$ $\quad A\cdot\bar{A}=0.$

$= AB$

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

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

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

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

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

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

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

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

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

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

$= \bar{A} + \bar{B} + A + \bar{C} = A + \bar{A} + \bar{B} + \bar{C} = 1 + \bar{B} + \bar{C} = 1$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$* f = A \cdot \overline{[\overline{(A \oplus B) \oplus C}]}$

$\rightarrow f = A \cdot \overline{[\overline{(A \oplus B)} \oplus C]}$

$A \quad B$

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

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

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

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

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

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

$= A \overline{[AB\overline{A}B + AB\overline{B}A + ABC + \overline{A}\overline{B}\overline{A}B + \overline{A}\overline{B}\overline{B}A + \overline{A}\,\overline{B}C + \overline{A}BC + \overline{B}A\overline{C} + C \cdot \overline{C}]}$

$= A \overline{[ABC + \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C}]}$

$= A \cdot ABC + A \cdot \overline{A}\overline{B}C + A \cdot \overline{A}B\overline{C} + A \cdot A\overline{B}\overline{C}$

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

$= A[BC + \overline{B}\overline{C}]$

$= A[B \odot C]$

$* f = (B + BC)(B + \overline{B}C)(B + D)$

$f = (B \cdot B + B\overline{B}C + BC \cdot B + BC\overline{B}C)(B + D)$

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

$= B \cdot B + BD + B \cdot BC + BCD$

$= B + BD + BC + BCD$

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

$= B + BD \Rightarrow B(1 + D) = B$

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

Applying the consensus theorm to 2nd and 4th terms

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

3rd and 5th terms, the term $\bar{A}\bar{B}$ becomes redundant.

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

$f_{min} = \bar{A}\bar{D} + \bar{B}\bar{C} + CD$

K-map (Karnaugh-map) :-

The K-map method, on the other hand, is a systematic methode of (Simplification dep) simplifying the Boolean Expressions. The K-map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.

→ An n variable function can have 2^n possible combinations of product terms in sop form, or 2^n possible combinations of sum terms in pos form.

→ A Two-variable K-map will have $2^2 = 4$ cells or squares,

→ a three-variable K-map will have $2^3 = 8$ cells or squares

→ a four-variable K-map will have $2^4 = 16$ cells or squares.

→ Any boolean Expression can be expressed in a standard or expanded sum of products form or in a standard or canonical or expanded product of sums form.

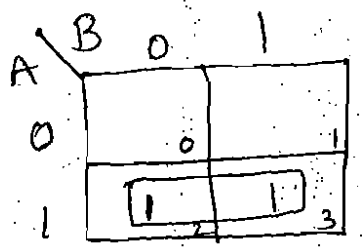
→ Each of the product terms in the standard sop form is called minterms and each of the sum terms in the standard pos form is called maxterms.

Two-Variable K-map :- (Mapping of SOP Expressions).

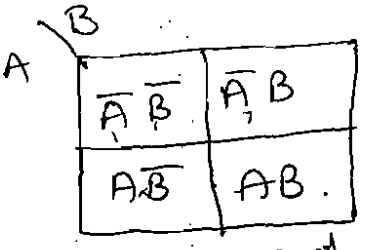
A two-variable K-map has $2^2 = 4$ cells or squares. 4 possible combinations of the input variables A and B. ($\bar{A}\bar{B}$, $\bar{A}B$, $A\bar{B}$, AB).

$m_0 = \bar{A}\bar{B}$, $m_1 = \bar{A}B$, $m_2 = A\bar{B}$, $m_3 = AB$.

* $f = A\bar{B} + AB$ simplify by using K-map.



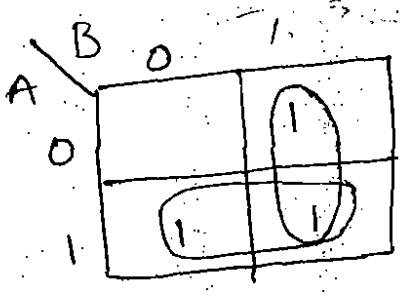
$f = B$.



The minterms of a two-variable K-map.

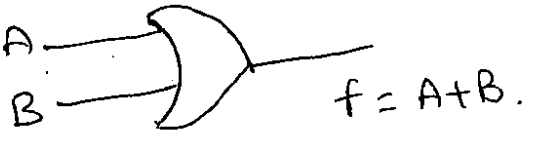
* Reduce the expression by using K-map.

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



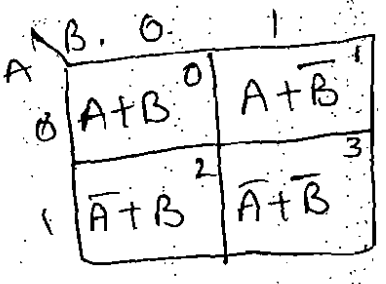
$f = A + B$

logic diagram



mapping of POS Expressions :-

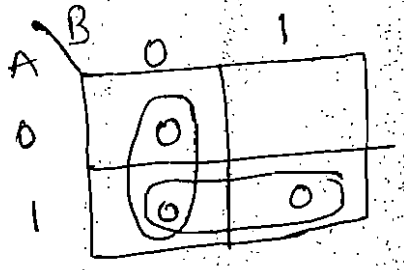
$M_0 = A + B$, $M_1 = A + \bar{B}$, $M_2 = \bar{A} + B$, $M_3 = \bar{A} + \bar{B}$

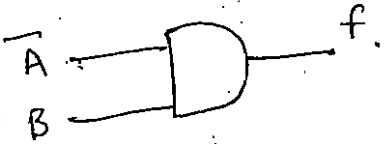


The maxterms of a two-variable K-map.

→ Reduce the expression by using k-map.

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





$f = \bar{A}.B$

Three- variable K-map :-

A function in three variable (A,B,C) Expressed in the standard Sop form can have $2^3 = 8$ possible combinations they are $\bar{A}\bar{B}\bar{C}$, $\bar{A}\bar{B}C$, $\bar{A}B\bar{C}$, $A\bar{B}\bar{C}$, $\bar{A}BC$, $A\bar{B}C$, $AB\bar{C}$, and ABC. in the standard pos form can have $2^3 = 8$ possible combinations. They are $A+B+C$, $A+B+\bar{C}$, $A+\bar{B}+C$, $A+\bar{B}+\bar{C}$, $\bar{A}+B+C$, $\bar{A}+B+\bar{C}$, $\bar{A}+\bar{B}+C$ and $\bar{A}+\bar{B}+\bar{C}$

A \ BC	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$ $(m_0)_0$	$\bar{A}\bar{B}C$ $(m_1)_1$	$\bar{A}BC$ $(m_3)_3$	$\bar{A}B\bar{C}$ $(m_2)_2$
1	$A\bar{B}\bar{C}$ $(m_4)_4$	$A\bar{B}C$ $(m_5)_5$	ABC $(m_7)_7$	$AB\bar{C}$ $(m_6)_6$

minterms

0 $A+B+C$ (M_0)	1 $A+B+\bar{C}$ (M_1)	3 $A+\bar{B}+\bar{C}$ (M_3)	2 $A+\bar{B}+C$ (M_2)
4 $\bar{A}+B+C$ (M_4)	5 $\bar{A}+B+\bar{C}$ (M_5)	7 $\bar{A}+\bar{B}+\bar{C}$ (M_7)	6 $\bar{A}+\bar{B}+C$ (M_6)

max terms.

* Reduce the expression $f = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC$.

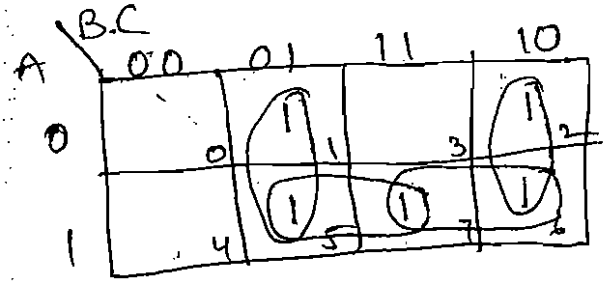
$\bar{A}\bar{B}C = 001 = m_1$,

$A\bar{B}C = 101 = m_5$

$\bar{A}B\bar{C} = 010 = m_2$

$AB\bar{C} = 110 = m_6$.

$ABC = 111 = m_7$



$f_1 = m_1 + m_5 = \bar{A}\bar{B}C + A\bar{B}C = \bar{B}C(A+\bar{A}) = \bar{B}C$.

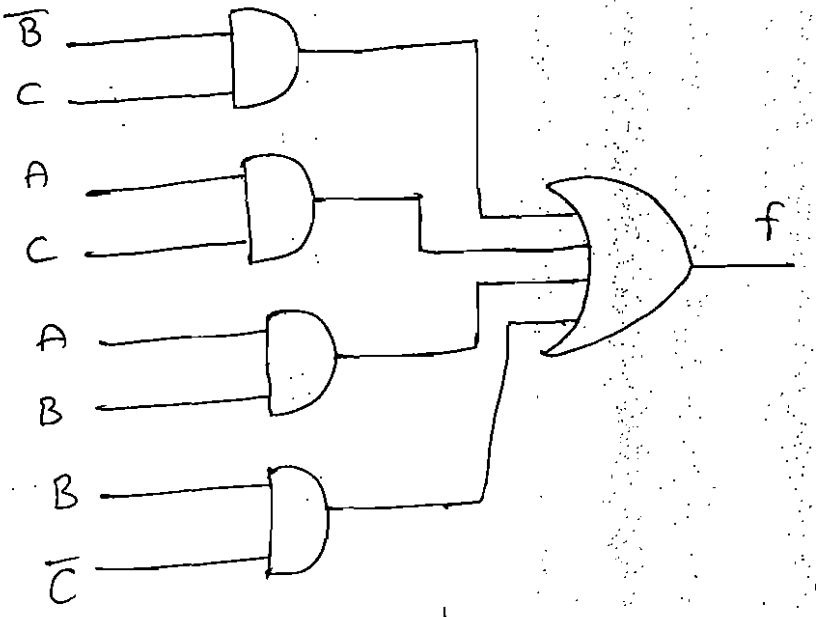
$f_2 = m_5 + m_7 = A\bar{B}C + ABC = AC(B+\bar{B}) = AC$

$f_3 = m_7 + m_6 = ABC + AB\bar{C} = AB(C+\bar{C}) = AB$

$f_4 = m_2 + m_6 = AB\bar{C} + \bar{A}B\bar{C} = B\bar{C}(A+\bar{A}) = B\bar{C}$

$f = f_1 + f_2 + f_3 + f_4$

$= \bar{B}C + AC + AB + B\bar{C}$



Logic diagram

* Reduce the Expression. $f = (A+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+\bar{C})$

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

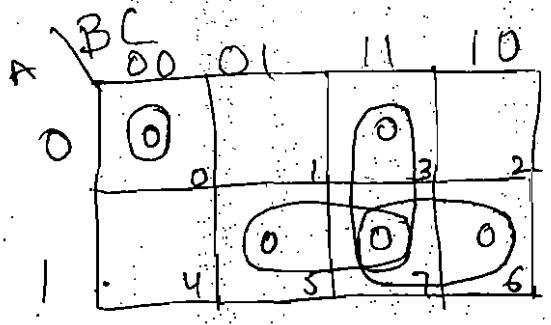
$A+B+C = 000 = M_0$

$\bar{A}+B+\bar{C} = 101 = M_5$

$\bar{A}+\bar{B}+\bar{C} = 111 = M_7$

$A+\bar{B}+\bar{C} = 011 = M_3$.

$\bar{A}+\bar{B}+C = 110 = M_6$.



$f_1 = M_0 = A+B+C$

$f_2 = M_5 \cdot M_7 = (\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+\bar{C})$

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

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

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

$f_2 = \bar{A} + \bar{C}$

$f_3 = M_3 \cdot M_7$

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

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

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

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

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

$f_4 = M_7 \cdot M_6$

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

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

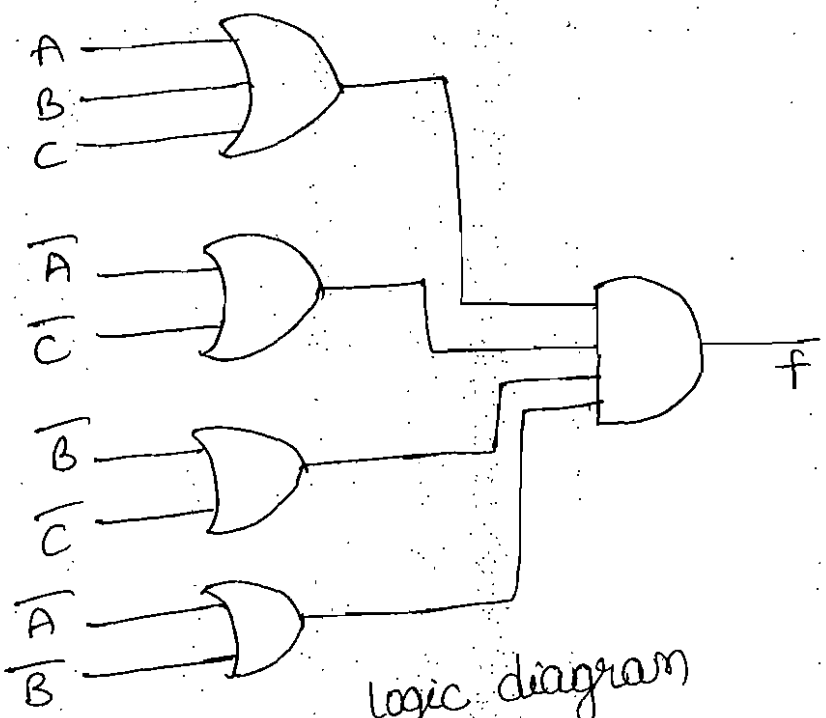
$= \bar{A} + \bar{A}\bar{B} + \bar{A}C + \bar{B} + \bar{B}C + \bar{A}\bar{C} + \bar{B}\bar{C} + 0$

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

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

$F = f_1 \cdot f_2 \cdot f_3 \cdot f_4$

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



Logic diagram

Four-variable K-map :-

A four-variable (A, B, C, D) expression can have $2^4 = 16$ possible combinations.

AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$ 0	$\bar{A}\bar{B}\bar{C}D$ 1	$\bar{A}\bar{B}CD$ 3	$\bar{A}\bar{B}C\bar{D}$ 2
01	$\bar{A}B\bar{C}\bar{D}$ 4	$\bar{A}B\bar{C}D$ 5	$\bar{A}BCD$ 7	$\bar{A}BC\bar{D}$ 6
11	$AB\bar{C}\bar{D}$ 12	$AB\bar{C}D$ 13	$ABCD$ 15	$ABC\bar{D}$ 14
10	$A\bar{B}\bar{C}\bar{D}$ 8	$A\bar{B}\bar{C}D$ 9	$A\bar{B}CD$ 11	$A\bar{B}C\bar{D}$ 10

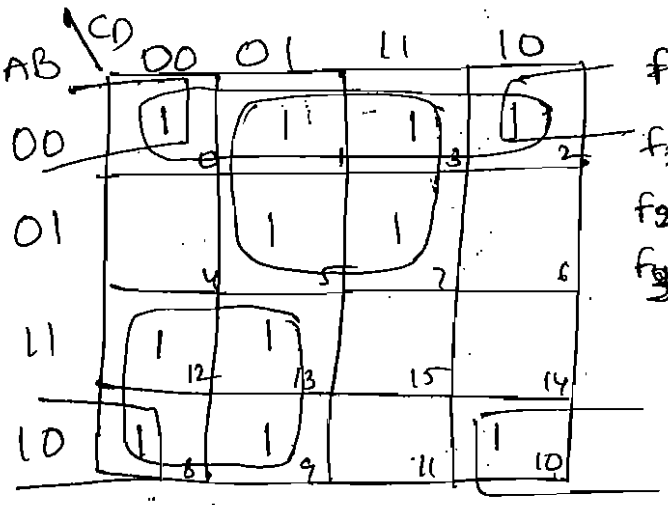
Sop form.

AB \ CD	00	01	11	10
00	$A+B+C+D$ 0	$A+B+C+\bar{D}$ 1	$A+B+\bar{C}+\bar{D}$ 3	$A+B+\bar{C}+D$ 2
01	$A+\bar{B}+C+D$ 4	$A+\bar{B}+C+\bar{D}$ 5	$A+\bar{B}+\bar{C}+\bar{D}$ 7	$A+\bar{B}+\bar{C}+D$ 6
11	$\bar{A}+\bar{B}+C+D$ 12	$\bar{A}+\bar{B}+C+\bar{D}$ 13	$\bar{A}+\bar{B}+\bar{C}+\bar{D}$ 15	$\bar{A}+\bar{B}+\bar{C}+D$ 14
10	$\bar{A}+B+C+D$ 8	$\bar{A}+B+C+\bar{D}$ 9	$\bar{A}+B+\bar{C}+\bar{D}$ 11	$\bar{A}+B+\bar{C}+D$ 10

pos form.

* Reduce the expression $f = \Sigma m(0,1,2,3,5,7,8,9,10,12,13)$ and implement the expression by using universal logic.

$f = \Sigma m(0,1,2,3,5,7,8,9,10,12,13)$



$f_1 = m_0 + m_1 + m_2 + m_3 = \overline{A}\,\overline{B}$ ~~$\overline{A}\,\overline{B}$~~

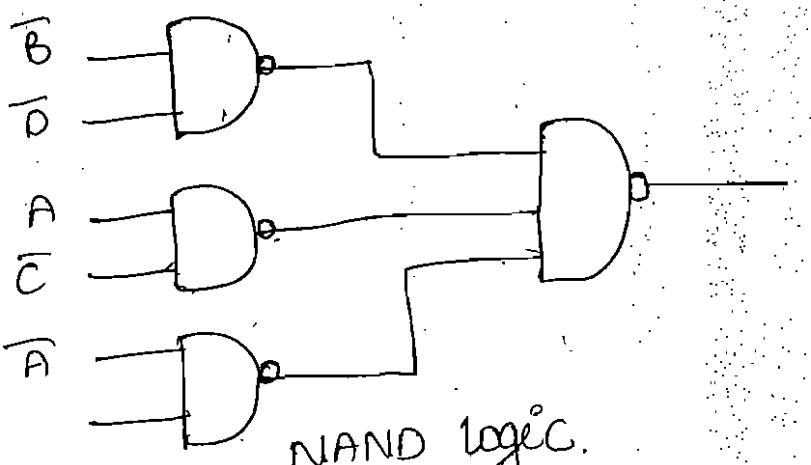
$f_2 = m_1 + m_3 + m_5 + m_7 = \overline{A}D$

$f_3 = m_{12} + m_{13} + m_8 + m_9 = A\overline{C}$

$f_4 = m_8 + m_{10} + m_0 + m_2 = \overline{B}\,\overline{D}$

$f_{min} = f_1 + f_2 + f_3$

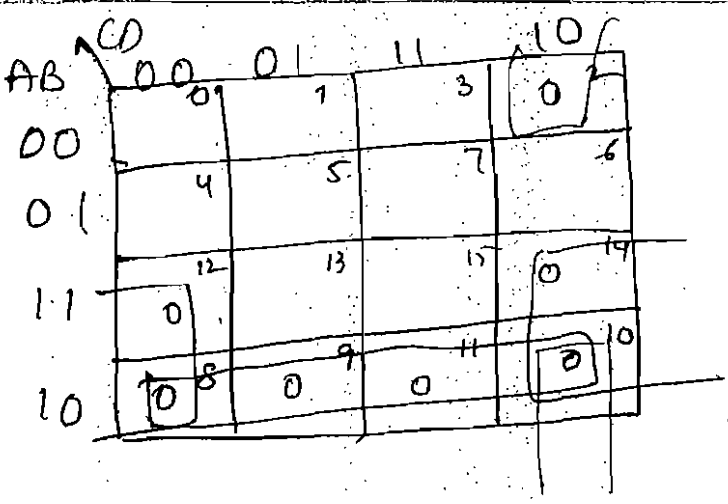
$= \overline{B}\,\overline{D} + A\overline{C} + \overline{A}D$



NAND logic.

* Reduce the expression $f = \Pi M(2,8,9,10,11,12,14)$ and implement the expression by using universal logic.

$f = \Pi M(2,8,9,10,11,12,14)$.



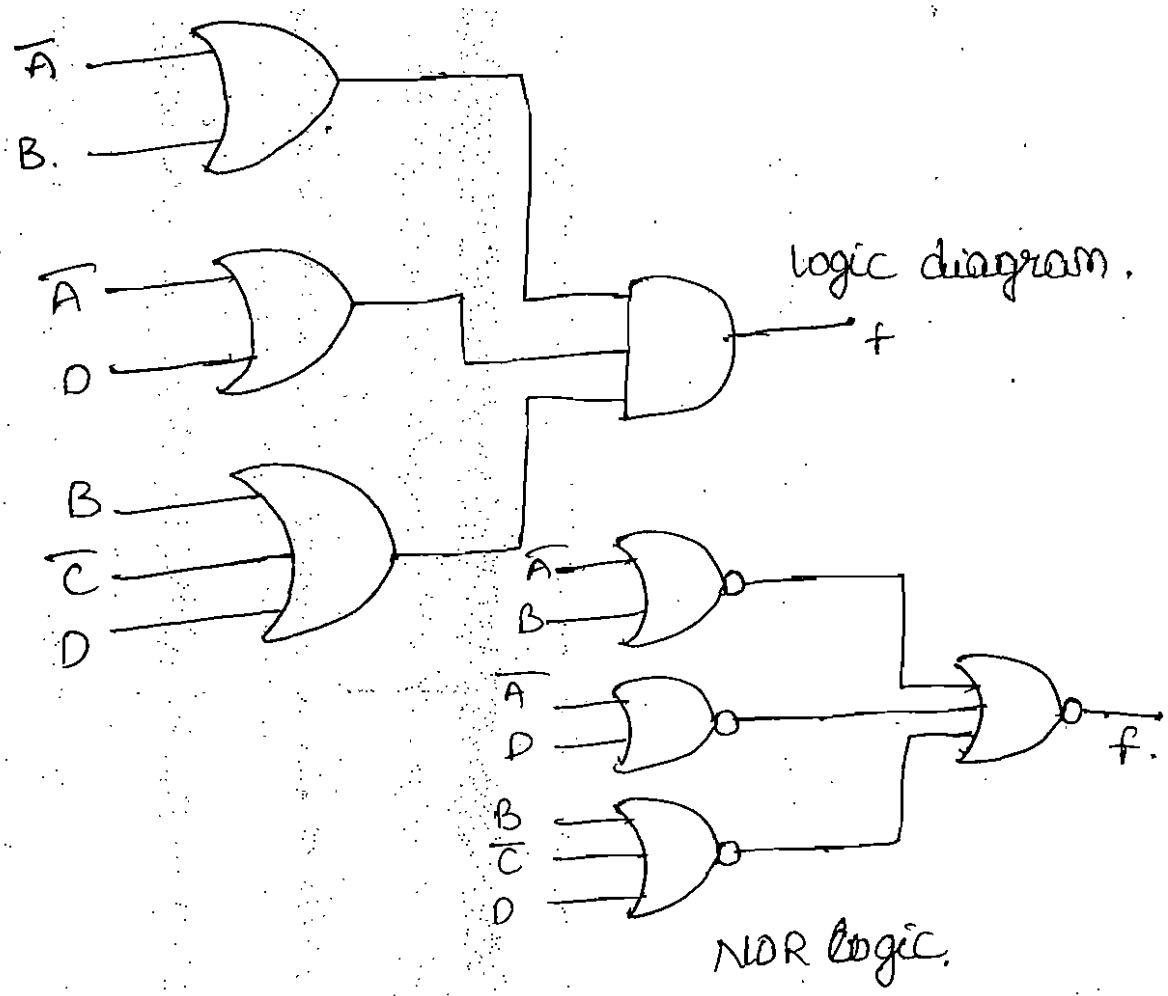
$f_1 = M_8 \cdot M_9 \cdot M_{11} \cdot M_{10}$ $f_2 = M_8 \cdot M_{12} \cdot M_{10} \cdot M_{14}$

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

$f_3 = M_2 \cdot M_{10}$

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

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



logic diagram.

NOR logic.

prime implicants :- [PI]

Each square or rectangle made up of the bunch of adjacent minterms is called a subcube. Each of these subcubes is called a prime implicants.

essential prime implicants:-

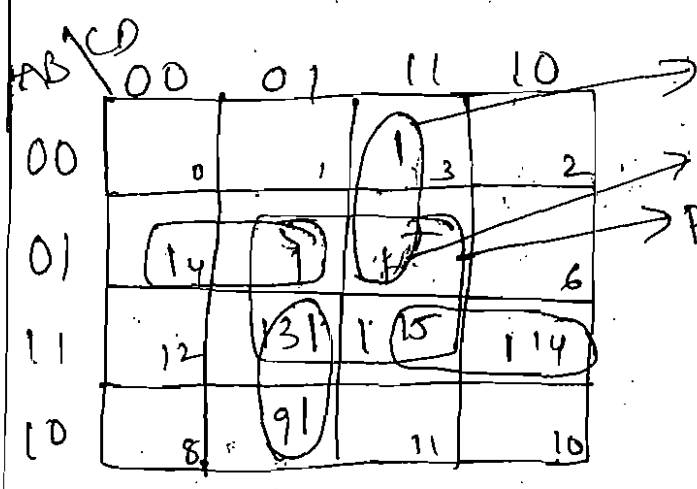
The prime implicants which contains at least one 1 which cannot be covered by any other prime implicant is called an essential prime implicant [EPI]

Redundant prime implicants:-

The prime implicant whose each 1 is covered at least by one EPI is called a redundant prime implicants (RPI).

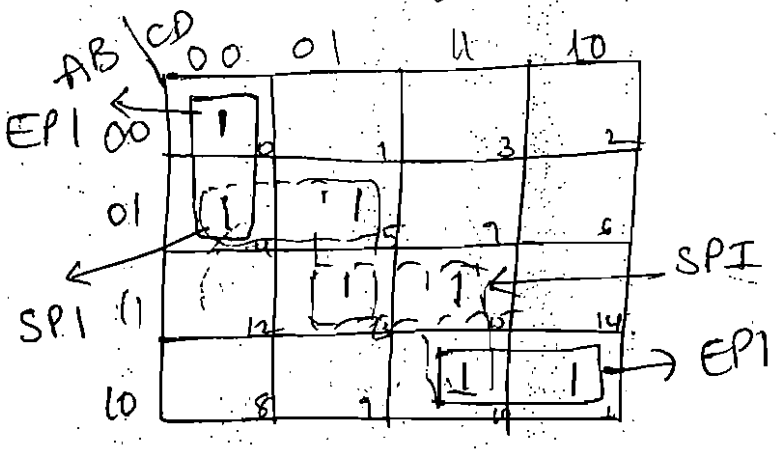
Selective prime implicants :-

A prime implicant which is neither an essential prime implicant nor a redundant prime implicant is called a selective prime implicant (SPI).



$f = \Sigma m(5, 7, 13, 15, 9, 14, 4)$.

$f(A,B,C,D) = \Sigma m(0,4,5,10,11,13,15)$.

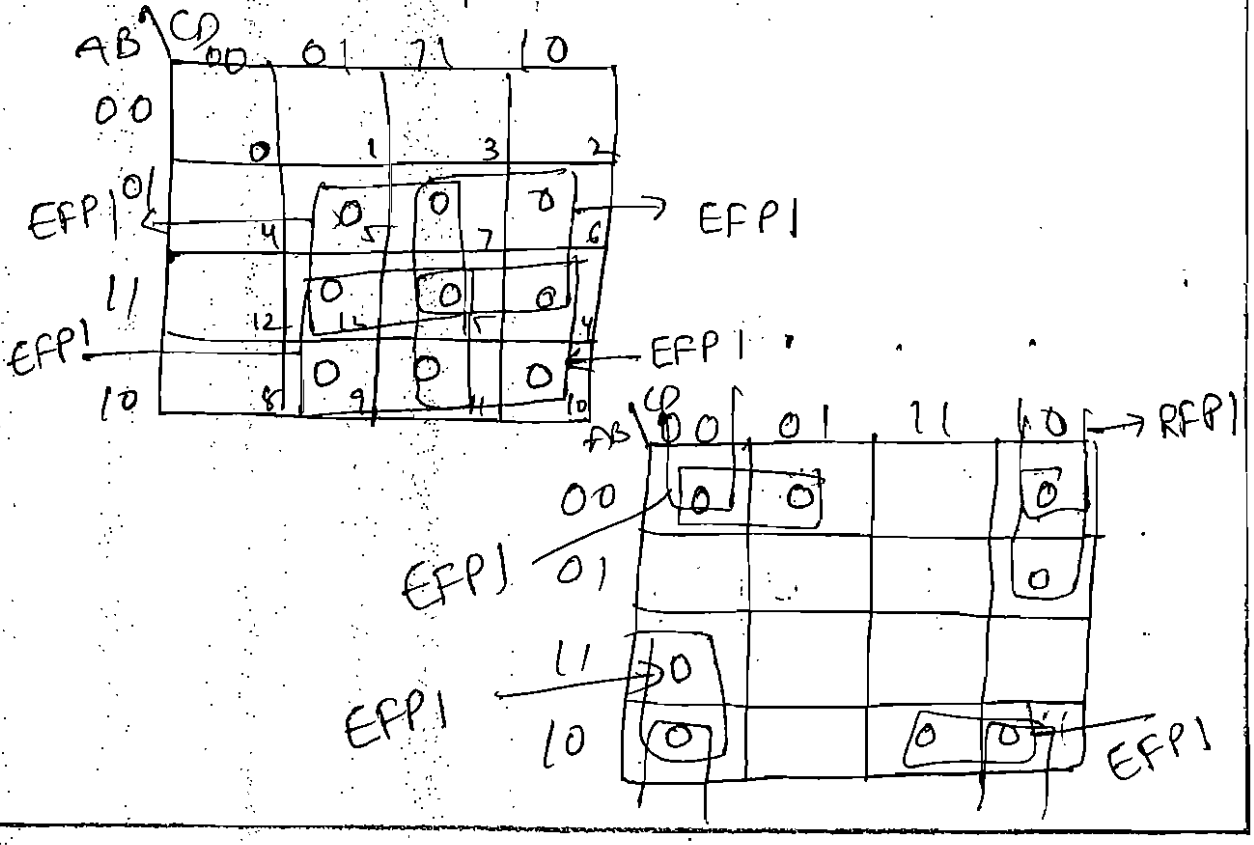


False prime implicants :- (FPI)

The prime implicants obtained by using the maxterms are called false prime implicants.

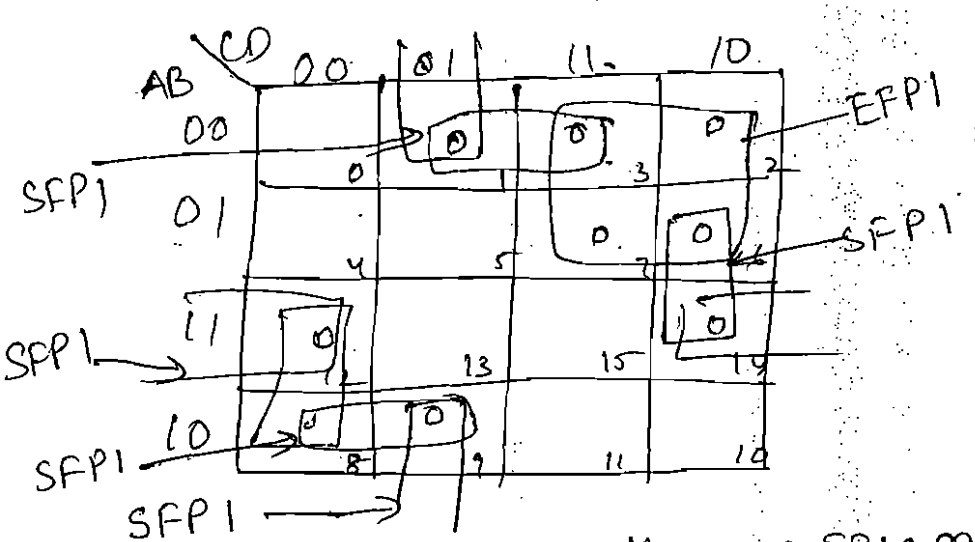
Essential false prime implicants :-

The FPI's which contains at least one '0' which cannot be coverd by any other FPI is called Essential False prime implicants (EFPI).



The four corner 0's from the largest cluster of adjacent 0's, which is an FPI whose 0s are covered by essential FPIs and hence is a redundant false prime implicant (RFPI.

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



The function has in all seven FPIs marked in figure. The FPI is an essential FPI as it contains 0s at locations 2 and 7 which cannot be covered by any other FPI. The remaing FPI are all SFPIs.

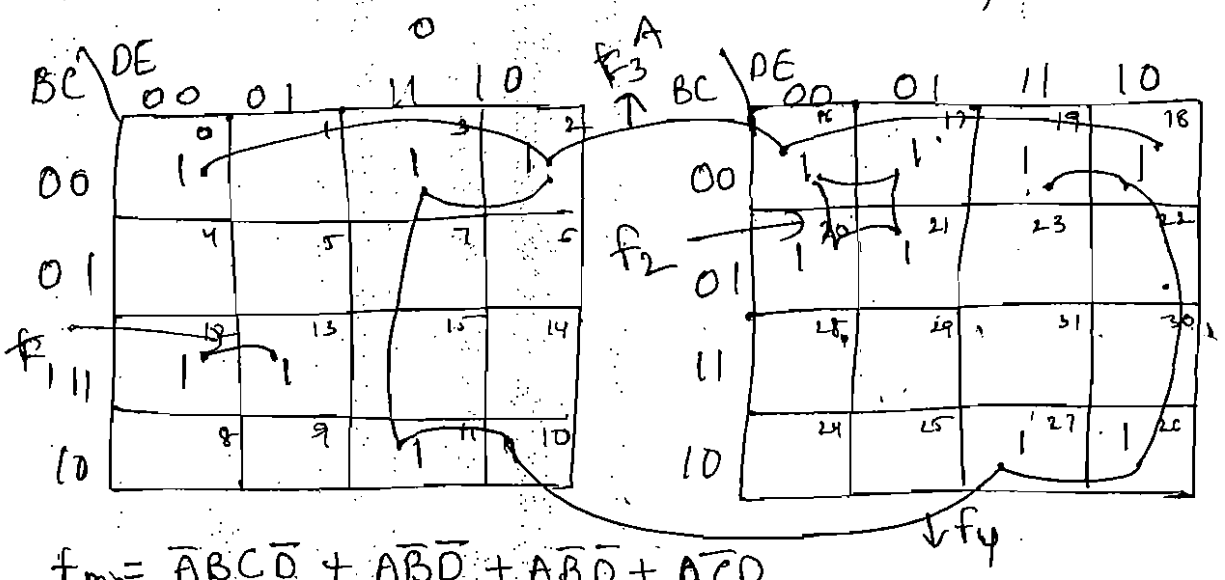
Five-variable K-map :-

A five-variable (A,B,C,D,E) expression can have $2^5 = 32$ possible combinations of input variables.

The 32 squares of the K-map are divided into 2 blocks of 16 squares each. one block taken as a A = '0'. and another one taken as a A = '1'.

* Reduce the following expression in sop and pos form.

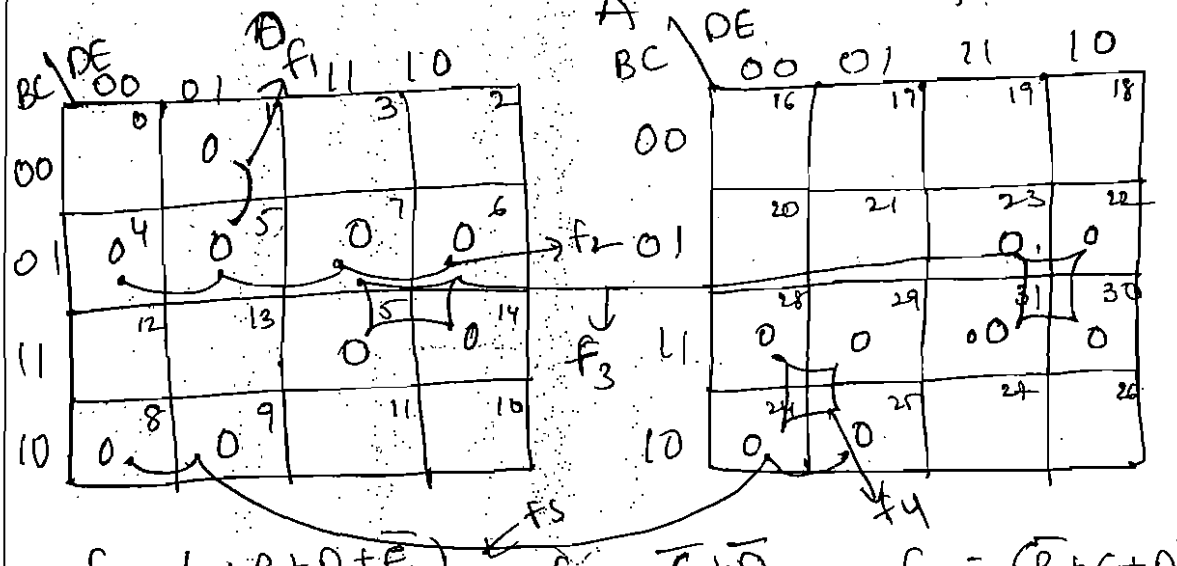
$f = \Sigma m(0,2,3,10,11,12,13,16,17,18,19,20,21,26,27)$



$f_{min} = \bar{A}BC\bar{D} + A\bar{B}\bar{D} + A\bar{B}\bar{D} + A\bar{C}D$

$f_1 \quad f_2 \quad f_3 \quad f_4$

$f = \Pi M(1,4,5,6,7,8,9,14,15,22,23,24,25,28,29,30,31)$



$f_1 = (A+B+D+\bar{E})$ $\quad f_3 = \bar{C}+\bar{D}$ $\quad f_5 = (\bar{B}+C+D)$

$f_2 = (A+B+\bar{C})$ $\quad f_4 = \bar{A}+\bar{B}+D$

$F_{min} = f_1 \cdot f_2 \cdot f_3 \cdot f_4 \cdot f_5$

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

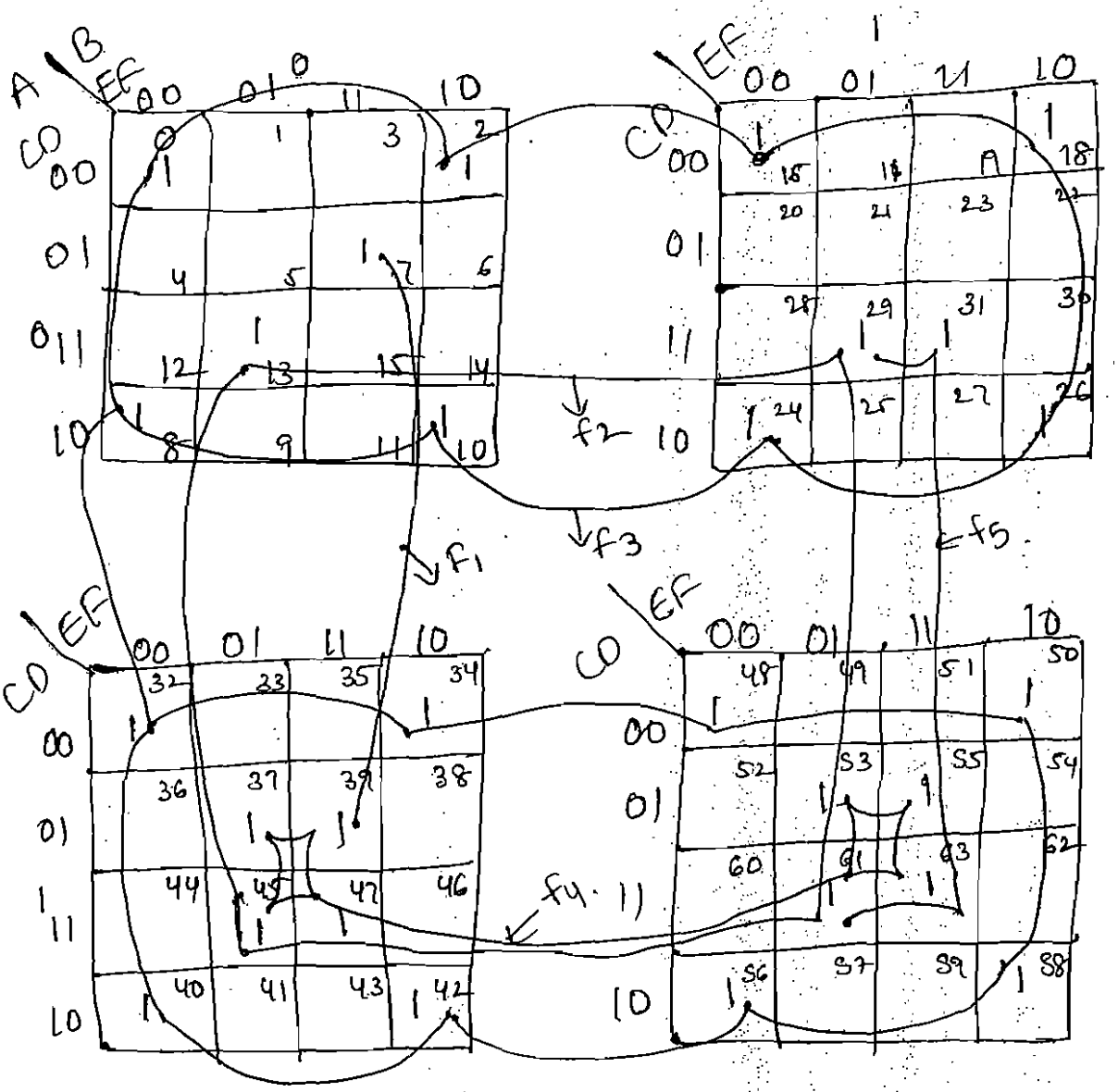
Six-Variable K-map :-

A Six-variable (A,B,C,D,E,F) expression can have $2^6 = 64$ possible combinations of input variables.

The 64 squares of the K-map are divided into 4 blocks of 16 squares each. one block taken as a AB = "00" and remaing are "01", "10", and "11".

* Reduce the expression

$f = \Sigma m(0,2,7,8,10,13,16,18,24,26,29,31,32,34,37,39,40,42,45,47,48,50,53,55,56,58,61,63)$.



$f_1 = \bar{B}\bar{C}DEF$

$f_2 = CD\bar{E}F$

$f_3 = \bar{D}\bar{F}$

$f_4 = ADF$

$f_5 = BCDF$

$F_{min} = f_1 + f_2 + f_3 + f_4 + f_5$

$F_{min} = \bar{B}\bar{C}DEF + CD\bar{E}\bar{F} + \bar{D}\bar{F} + ADF + BCDF$.

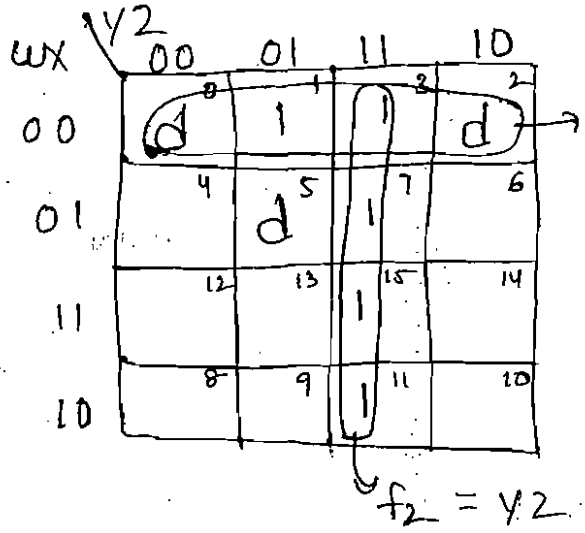
Don't care combinations :-

The combinations for which the values of the expression are not specified are called don't care combinations & optional combinations. and such expressions, therefore incompletely specified. The don't care terms are denoted by d, X & ϕ. During the process of design using an sop map each don't care is treated as a 1 if it is helpful in map reduction. During the process of design using an pos map each don't care is treated as a 0 if it is helpful in map reduction.

→ A standard sop expression with don't cares can be converted into a standard pos form by keeping the don't cares as they are, and writing the missing maxterms of the sop form as the maxterms of the pos form.

→ A standard pos expression with don't cares can be converted into a standard sop form by keeping the don't cares as they are and writing the missing maxterms of the pos form as the min terms of the sop form.

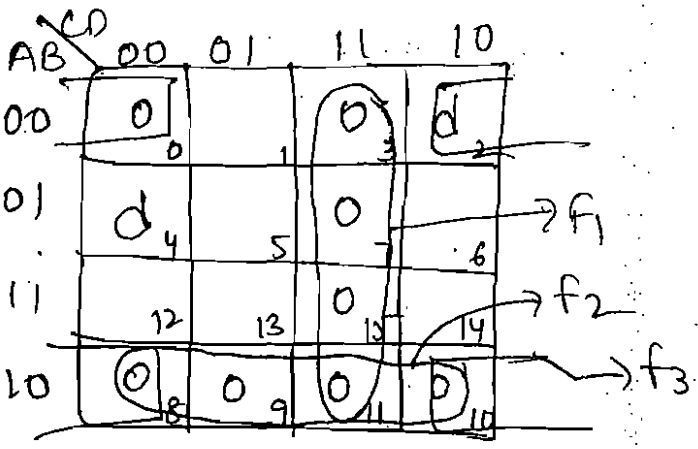
$F(w,x,y,z) = \Sigma m(1,3,7,11,15) + \Sigma d(0,2,5)$.



$F_{min} = f_1 + f_2$

$= \bar{w}\bar{x} + yz$.

$F(A,B,C,D) = \Pi M(0,3,7,8,9,10,11,15).\ \Pi d(2,4)$.



$f_1 = \bar{C} + \bar{D}$

$f_2 = \bar{A} + B$

$f_3 = (B+D)$.

$F_{min} = f_1 * f_2\ f_3 = (\bar{C}+\bar{D})(\bar{A}+B)(B+D)$.

Limitations of K-map :-

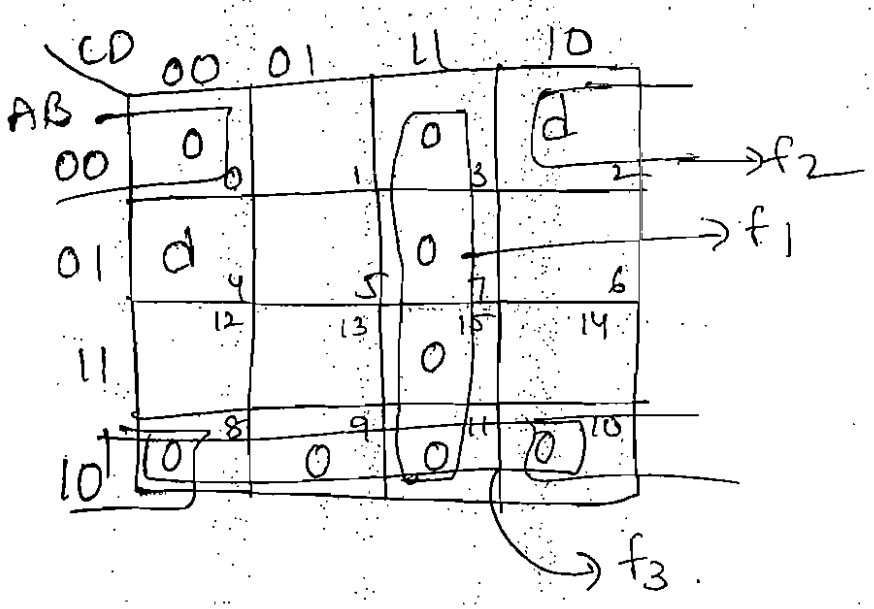
→ K-maps are not suitable when the number of variables involved exceed four. It may be used with difficulty up to five and six variable systems. But beyond 'six variable' K-maps cannot be physically visualized.

→ The K-map simplification is a manual technique and simplification process is heavily dependent on the abilities of the designer. It cannot be programmed.

→ Reduce the expression $f = \Sigma m(1,5,6,12,13,14) + d(2,4)$ by using k-map in POS form and implement by using universal logic

$f = \Sigma m(1,5,6,12,13,14) + d(2,4)$ → SOP form.

$f = \Pi M(0,3,7,8,9,10,11,15) \cdot \Pi d(2,4)$.

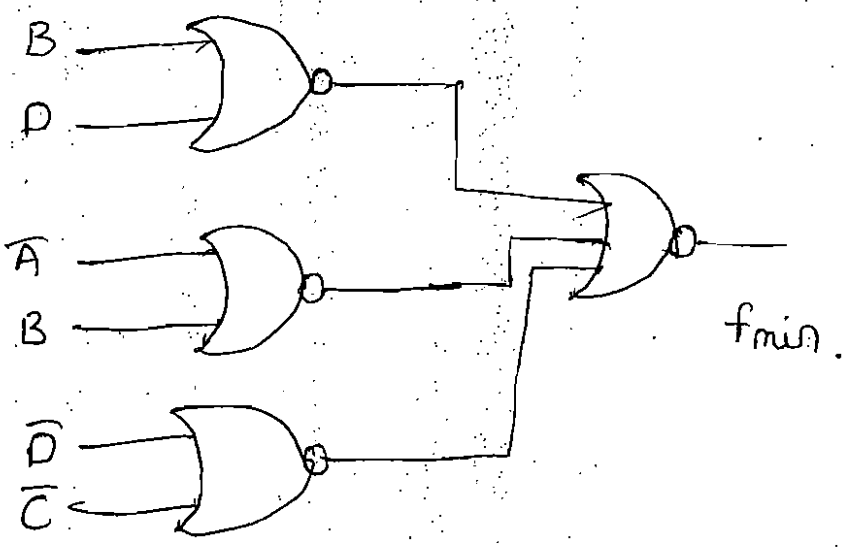


$f_1 = \bar{C} + \bar{D}$

$f_2 = (B + D)$

$f_3 = (\bar{A} + B)$

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



Quine Mccluskey or tabular method :-

W.V. Quine and E.J. Mccluskey developed an exact tabular method to simplify the Boolean Expression. This methode is called the Quine Mccluskey or tabular method.

The procedure for the minimization of a Boolean expression by the tabular methode

Step 1. List all the minterms.

Step 2: arrange all minterms in groups of the same number of 1's in their binary representation in column 1.

Step 3. compare each term of the lowest index group with every term in the succeeding group.

Step 4 :- compare the terms generated in step2 in the same fashion. combine two terms which differ by only a single 1 and whose dashes are in the same position to generate a new term.

Step 5 :- list all the prime implicants and draw the implicant chart. (The don't cares if any should not appear in the prime implicant chart).

Step 6 :- obtain essential prime implicants and write the minimal expression.

* obtain the minimal expression for $f = \Sigma m(1,2,3,5,6,7,8,9,12,13,15)$ using tabular methode.

Step 1		Step 2	Step 3.
index 1	1 0001 ✓	(1,3)(2) 00-1	(1,3,5,7)(2,4) 0--1 T
	2 0010 ✓	(1,5)(4) 0-01	(1,5,9,13)(4,8) --01 S
	8 1000 ✓	(1,9)(8) -001	(2,3,6,7)(1,4) 0-1- R
index 2	3 0011 ✓	(2,3)(1) 001-	(8,9,12,13)(1,4) 1-0- Q
	5 0101 ✓	(2,6)(4) 0-10	(5,7,13,15)(2,8) -1-1 P
	6 0110 ✓	(8,9)(1) 100-	
	9 1001 ✓	(8,12)(4) 1-00	
	12 1100 ✓	(3,7)(4) 0-11	
index 3	7 0111 ✓	(5,7)(2) 01-1	
	13 1101 ✓	(5,13)(8) -101	
		(6,7)(1) 011-	
index 4	15 1111 ✓	(9,13)(4) 1-01	
		(12,13)(1) 110-	
		(7,15)(8) -111	
		(13,15)(2) -11-1	

Prime implicant chart

	1	2	3	5	6	7	8	9	12	13	15
P → 5,7,13,15(2,8)				X		X				X	(X)
Q → 8,9,12,13							(X)	X	(X)	X	
R → 2,3,6,7		(X)	X		(X)	X					
S → 1,5,9,13	X			X				(X)		X	
T → 1,3,5,7	X		X	X		X					

P + Q + R + S → $BD + A\bar{C} + \bar{A}C + \bar{C}D$

P + Q + R + T → $BD + A\bar{C} + \bar{A}C + \bar{A}D$.

Simplify the following function using the branching method

$f(A,B,C,D,E) = \Sigma m(0,4,12,16,19,24,28,29,31)$

Index 0	0	00000	(0,4) (4)	00-00	W
			(0,16) (16)	-0000	V
index 1	4	00100	(4,12) (8)	0-100	U
	16	10000	(16,24) (8)	1-000	T
index 2	12	01100	(12,28) (16)	-1100	S
	24	11000	(24,28) (4)	11-00	R
index 3	19	10011	(28,29) (1)	1110-	Q
	28	11100			
index 4	29	11101	(29,31) (2)	111-1	P
index 5	31	11111			

Prime implicant chart

	0	4	12	16	19	24	28	29	31
* P(29,31)								X	X
Q(28,29)							X	X	
R (24,28)						X	X		
S (12,28)			X				X		
T (16,24)				X		X			
U (4,12)		X	X						
V (0,16)	X			X					
W (0,4)	X	X							
* X (19)					X				

In table x and p are essential prime implicants.

	0	4	12	16	24	28
w(0,4) (4)	X	X				
V(0,16) (16)	X			X		
U(4,12) (8)		X	X			
T(16,24) (8)				X	X	
S(12,28) (16)			X			X
R(24,28) (4)					X	X
Q(28,29) (1)						X

In the reduced PI chart, there are no essential PIs, dominated rows & dominating columns. in column 0. it is covered by rows w and V. First select row w.

	12	16	24	28
V(0,16)		X		
U(4,12)	X			
T(16,24)		X	X	
S(12,28)	X			X
R(24,28)			X	X
Q(28,29)				X

In this row V is dominated by row T, row U and row Q are dominated by row S. So rows V, U, Q can be reduced.

	12	16	24	28
* T(16,24)		X	X	
* S(12,28)	X			X
R(24,28)			X	X

In this T and S are the secondary essential prime implicants so R is Redundant.

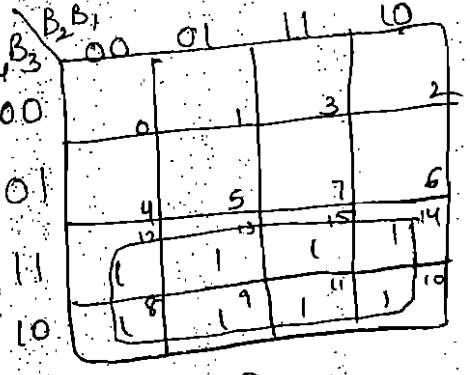
$$f_{min} = w + S + T + X + P$$
$$= \bar{A}\bar{B}\bar{D}\bar{E} + BC\bar{D}\bar{E} + A\bar{C}\bar{D}\bar{E} + A\bar{B}\bar{C}DE + ABCE$$

Code converters :-

Design a circuit 4-bit binary to Gray code converters

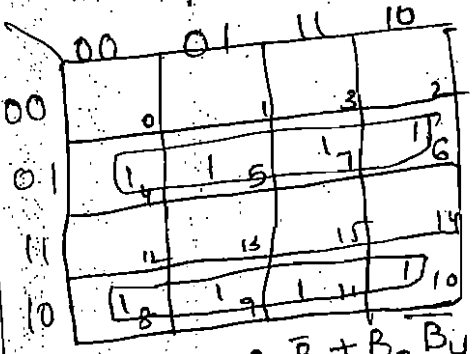
4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

K-map for G_4



$G_4 = B_4$

K-map for G_3



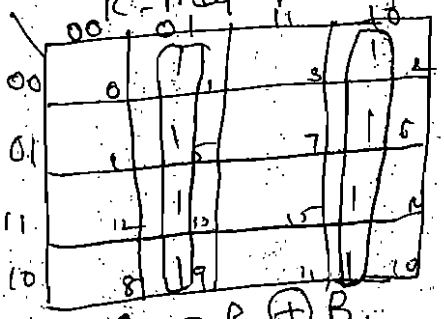
$G_3 = B_4\bar{B}_3 + B_3\bar{B}_4$

$= B_3 \oplus B_4$

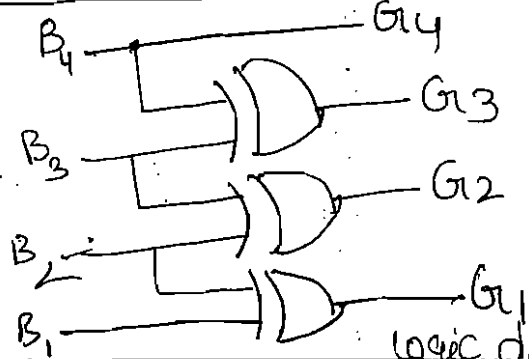
K-map for G_2

$G_2 = B_3 \oplus B_2$

K-map for G_1



$G_1 = B_2 \oplus B_1$

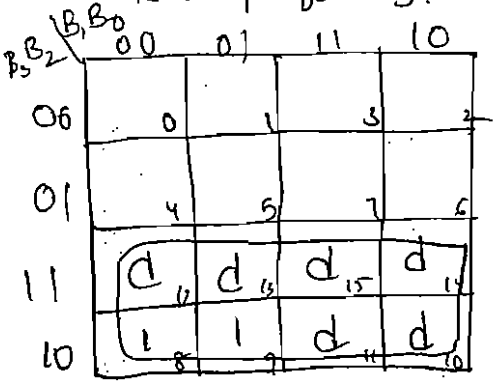


logic diagram

Design of a BCD to Gray code converter :-

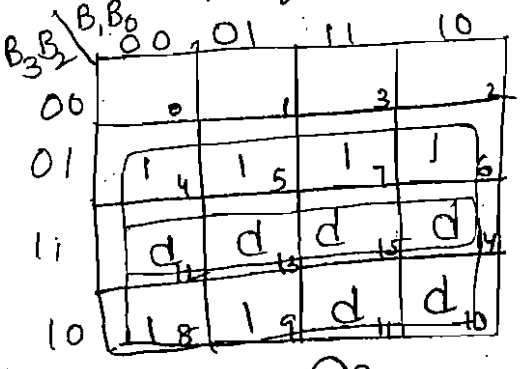
BCD code $B_3\ B_2\ B_1\ B_0$	Gray code $G_3\ G_2\ G_1\ G_0$
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1

K-map for G_3.



$G_3 = B_3$.

K-map for G_2



$G_2 = B_2 \oplus B_3$.

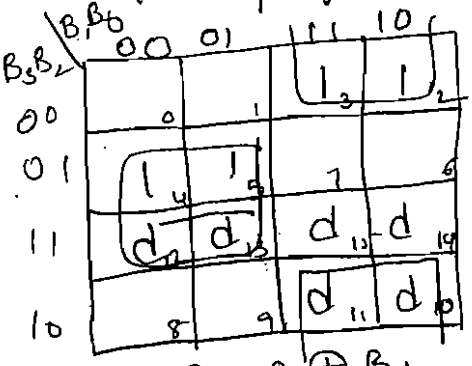
$G_3 = \Sigma m(8,9) + d(10,11,12,13,14,15)$.

$G_2 = \Sigma m(4,5,6,7,8,9) + d(10,11,12,13,14,15)$

$G_1 = \Sigma m(2,3,4,5) + d(10,11,12,13,14,15)$.

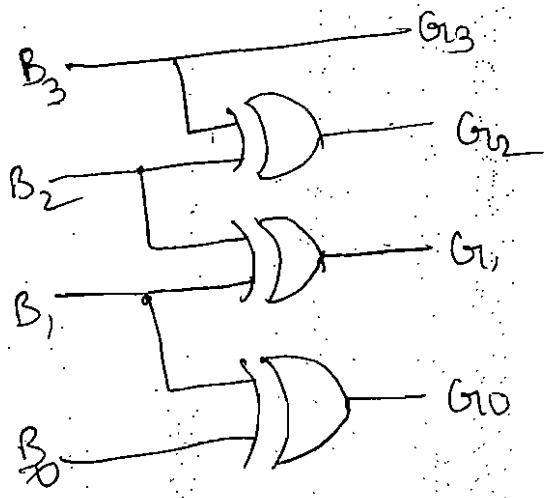
$G_0 = \Sigma m(1,2,5,6,9) + d(10,11,12,13,14,15)$.

K-map for G_1

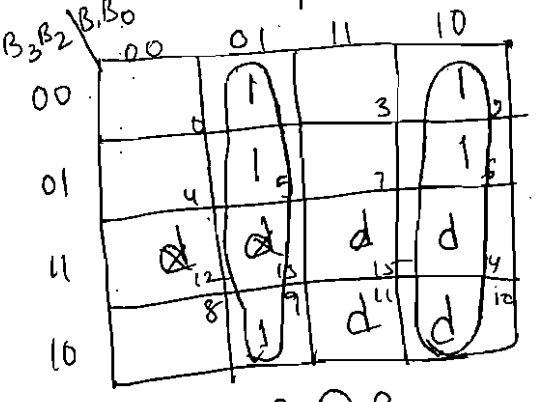


$G_1 = B_2 \oplus B_1$.

Logic diagram :-



K-map for G_0



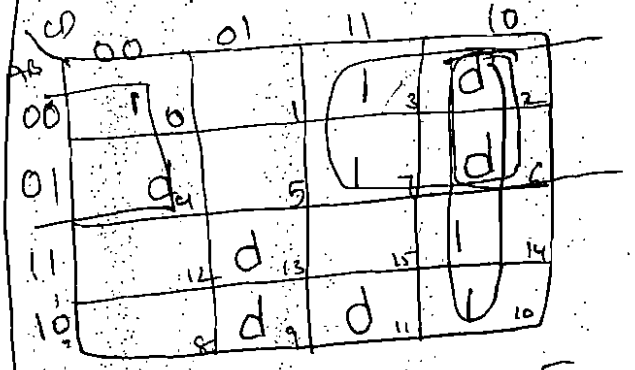
$G_0 = B_1 \oplus B_0$.

→ Design an SOP circuit to detect the decimal numbers 0, 2, 4, 6 and 8 in a 4-bit 5211 BCD code input.

Decimal number	5211 Code A B C D	output f
0	0 0 0 0	1
1	0 0 0 1	0
2	0 0 1 1	1
3	0 1 0 1	0
4	0 1 1 1	1
5	1 0 0 0	0
6	1 0 0 1	1
7	1 0 1 1	0
8 x	1 1 1 0	1
9	1 1 1 1	0

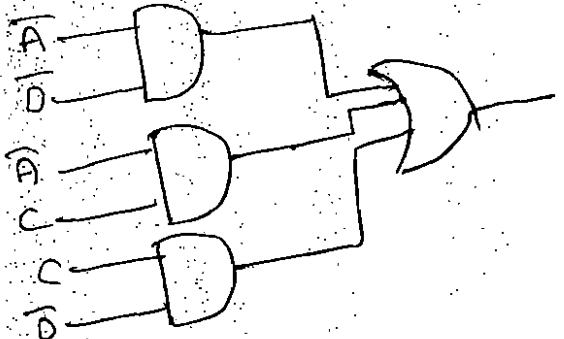
$f = \Sigma m(0,3,7,10,14) + d(2,4,6,9,11,13)$

K-map



$f_{min} = \bar{A}\bar{D} + \bar{A}C + C\bar{D}$

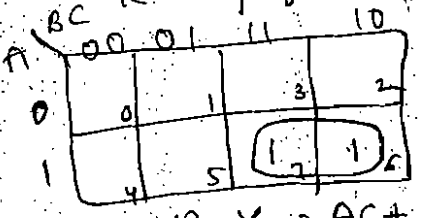
Logic diagram



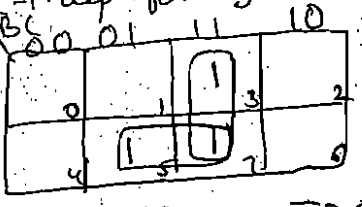
→ Design a combinational circuit that accepts a 3-bit BCD number and generates an output binary number equal to the square of the input number.

Inputs			outputs					
A	B	C	X_6	X_5	X_4	X_3	X_2	X_1
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

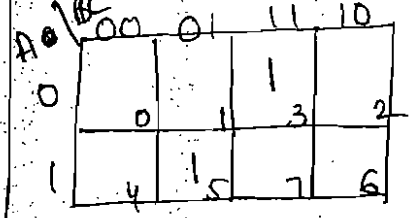
K-map for $X_6 = AB$



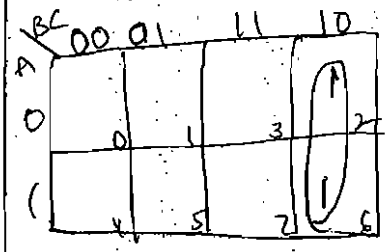
K-map for $X_5 = AC + BC$



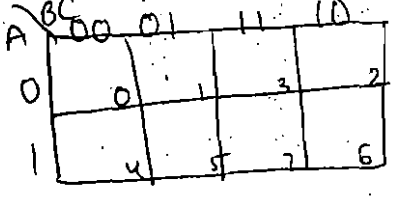
K-map for $X_4 = \bar{A}BC + A\bar{B}C$



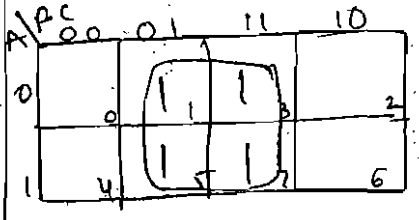
K-map for $X_3 = B\bar{C}$

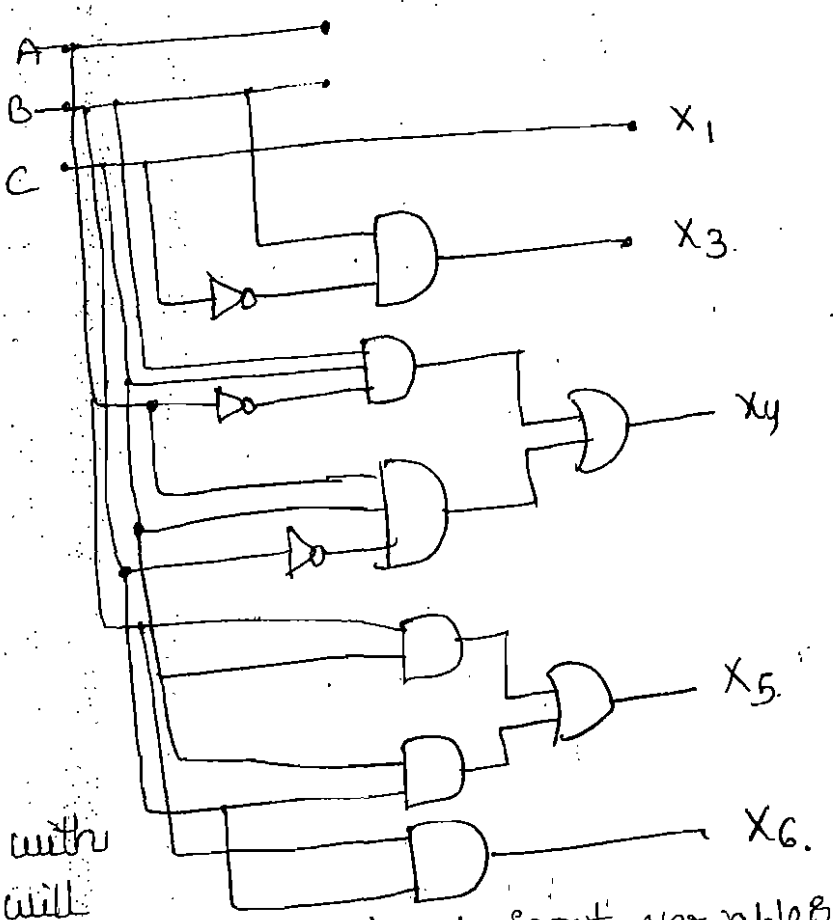


K-map for $X_2 = 0$



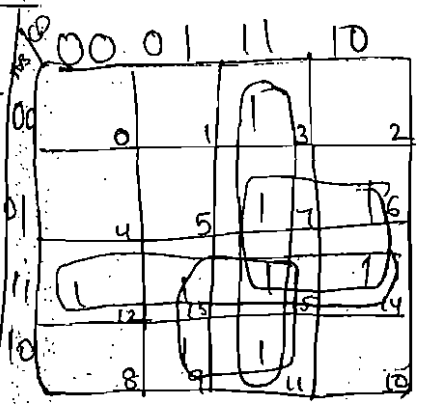
K-map for $X_1 = C$



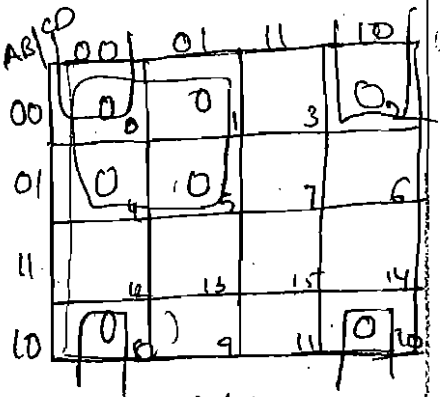


* Design a logic circuit with 4 inputs A, B, C, D that will produce output '1' only whenever two adjacent input variables are '1's. A and D are also to be treated as adjacent implement it using universal logic.

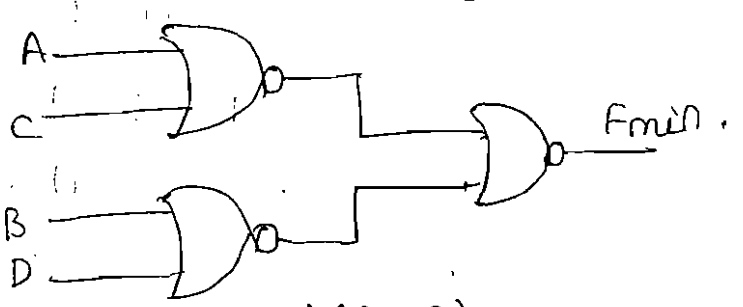
input A B C D	output f
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	0
0 1 1 0	1
0 1 1 1	1
1 0 0 0	0
1 0 0 1	1
1 0 1 0	0
1 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1



$F = AB + AD + BC + CD$



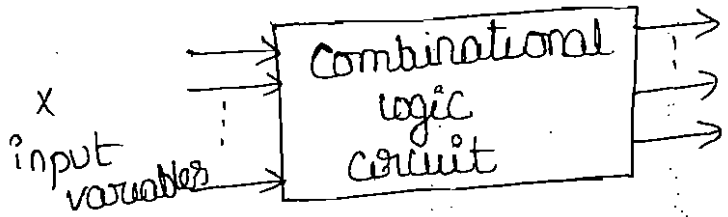
$f = (A+C)(B+D)$.



$F_{min} = (A+C)(B+D)$.

12.6 Combinational logic circuits design.

Logic circuits for digital systems may be combinational or sequential. In combinational circuits, the output variables at any instant of time are dependent only on the present input variables. In sequential circuits, the output variables at any instant of time are dependent on the present and past input variables.

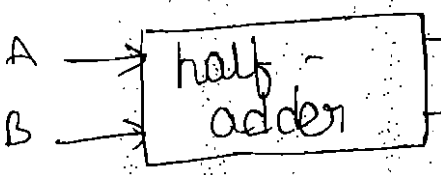


Adders :-

The most basic arithmetic operation is the addition of two binary digits. A combinational circuit that performs the addition of two bits is called a "half-adder" one that performs the addition of three bits (two bits and previous carry) is called a "full-adder".

Half-adder

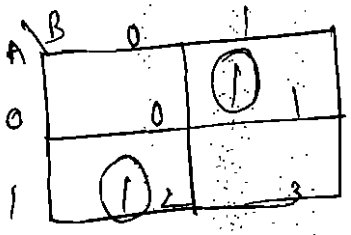
A half adder is a combinational circuit with two binary inputs. (augend and addend bits) and two outputs. Sum and carry.



Inputs		output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table.

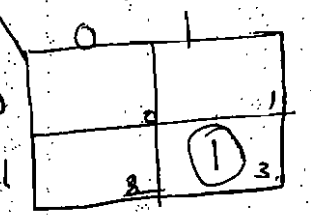
K-map for S.



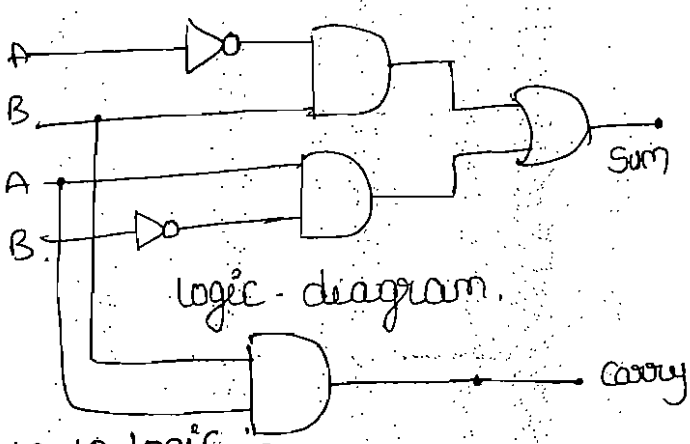
$S = A\bar{B} + \bar{A}B$

$= A \oplus B$

K-map for C

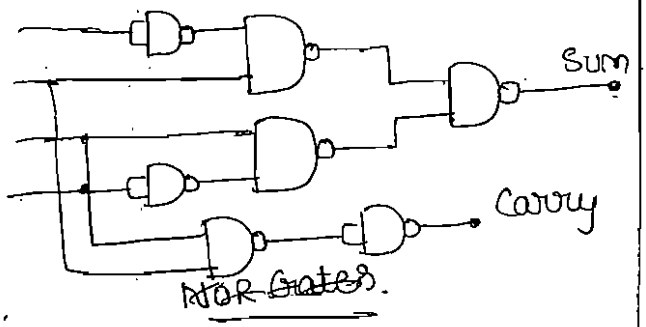


$C = A \cdot B$



Logic - diagram.

NAND Gates.



NOR Gates.

NAND logic :-

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

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

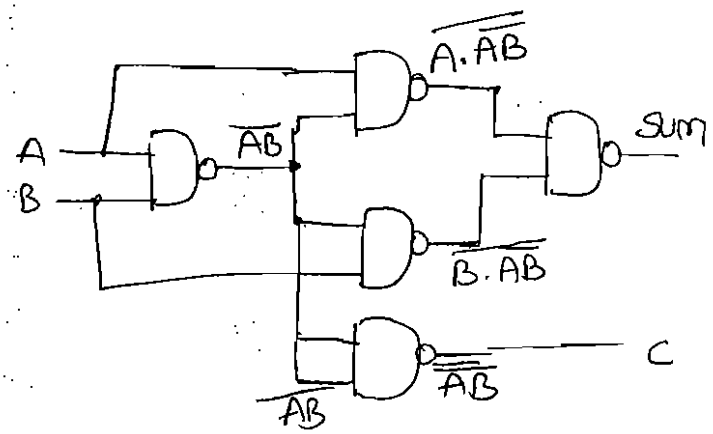
$= A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$

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

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

$= \overline{A\cdot\overline{AB}}\cdot\overline{B\cdot\overline{AB}}$

$C = AB = \overline{\overline{AB}}$



NOR logic

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

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

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

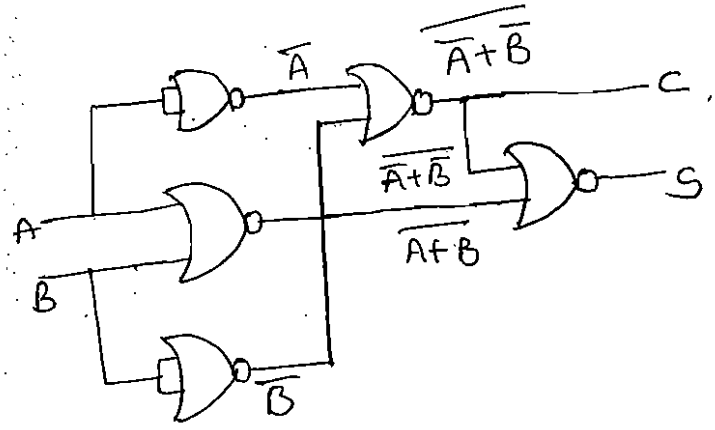
$= (A+B)(\bar{A}+\bar{B})$

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

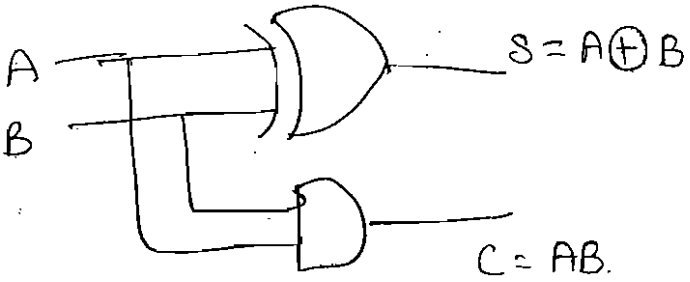
$= \overline{(\bar{A}+B)} + \overline{(\bar{A}+\bar{B})}$

$C = AB = \overline{\overline{AB}}$

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

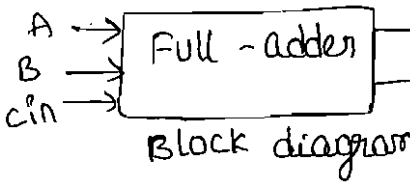


Simple logic diagram is.



Full adder :-

A full adder is a combinational circuit that adds two bits and a carry and outputs are sum and carry. The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in}.

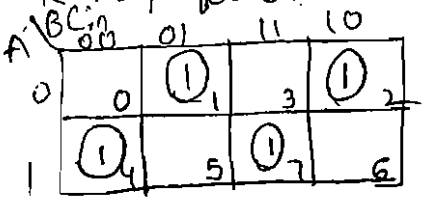


Block diagram.

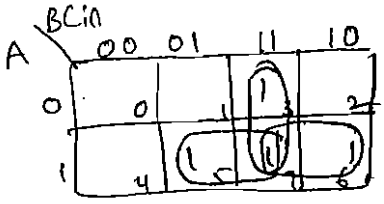
inputs			outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table

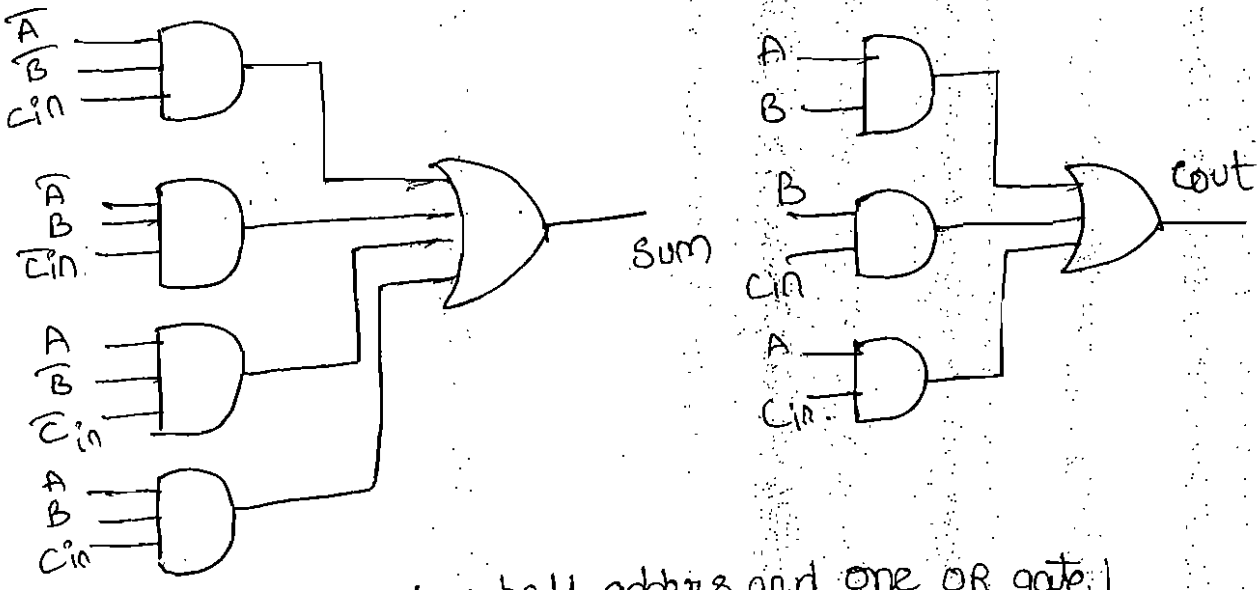
K-map for S.



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



$C_{out} = AB + BC_{in} + AC_{in}$.



Full adder (By using two half adders and one OR gate).

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

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

$= \overline{A \oplus B}\, C_{in} + \overline{C_{in}}(A \oplus B)'$

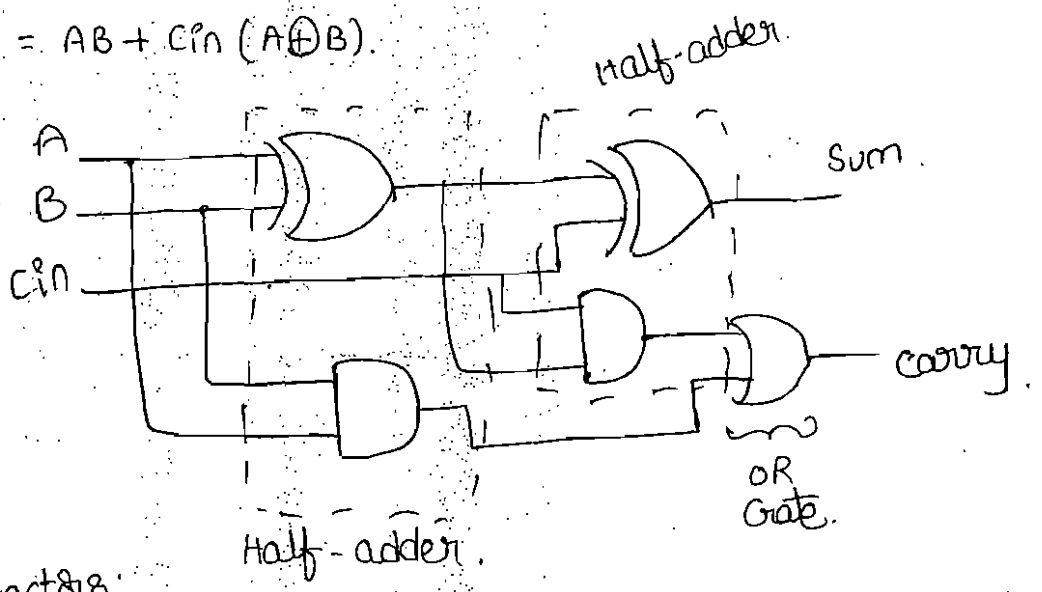
$= A \oplus B \oplus C_{in}$.

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

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

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

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

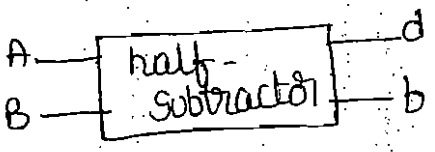


Subtractors:-

In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference position.

Half-subtractor:-

A half subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed.



inputs		outputs	
A	B	d	b
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

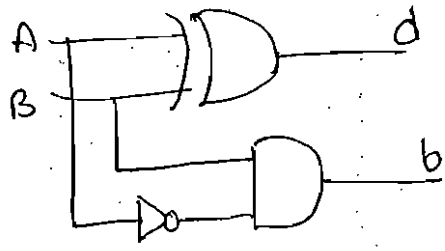
K-map for d

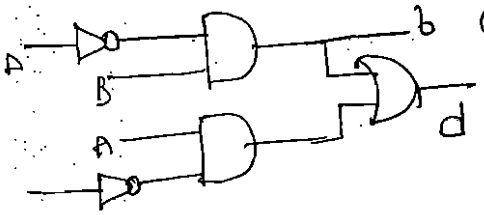
A \ B	0	1
0	0	(1) 1
1	(1) 2	3

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

$= A \oplus B$

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





A \ B	0	1
0	0	(1) 1
1	2	3

$b = \bar{A}B$

Logic diagrams of a half-subtractor.

NAND logic:-

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

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

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

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

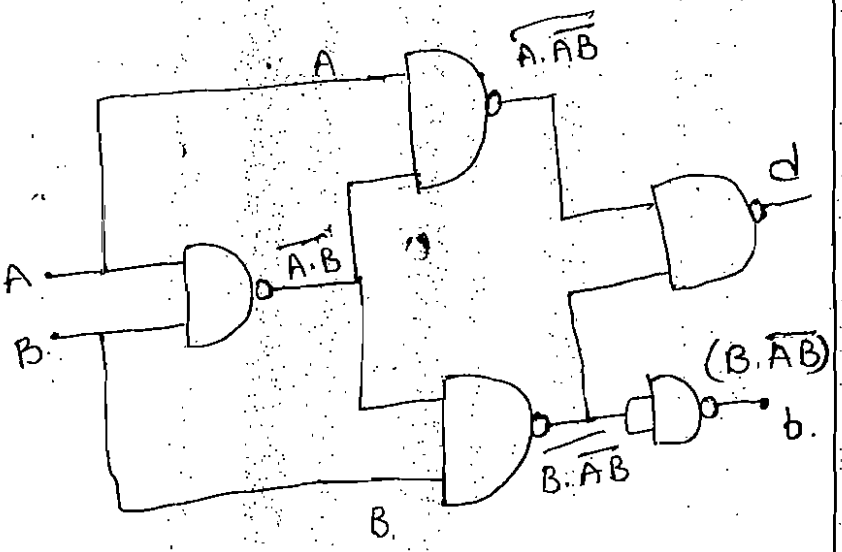
$= A.\overline{\bar{A}B}.\ B.\overline{\bar{A}B}$

$b = \bar{A}B$

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

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

$= B(\overline{AB})$.



NOR logic

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

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

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

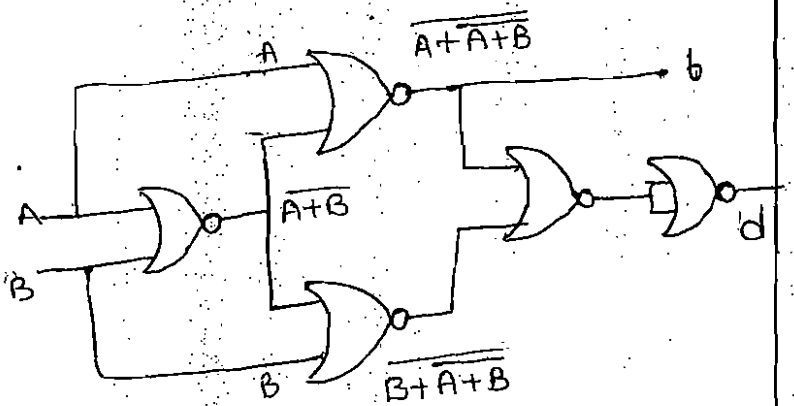
$= \overline{B+\bar{A}+B} + \overline{A+\bar{A}+B}$

$b = \bar{A}B$

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

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

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



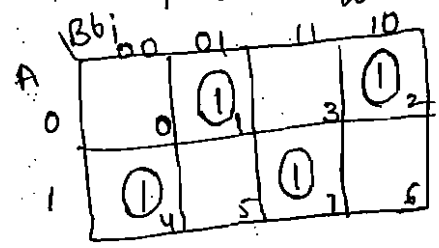
Full - Subtractor:-

The half - ~~adder~~ subtrator can be used only for LSB subtration. If there is a borrow during the subtration of the LSBs, it affects the subtraction in the next higher column. the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction. in the preceding column.

In full subtractor the inputs are A, B, borrow in b_i, and outputs are difference bit (d) and borrow (b).

inputs			outputs	
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map for difference



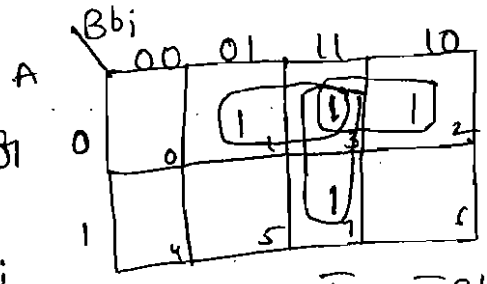
$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + A\bar{B}\bar{b_i} + AB b_i$$

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

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

$$= A \oplus B \oplus b_i$$

K-map for borrow.

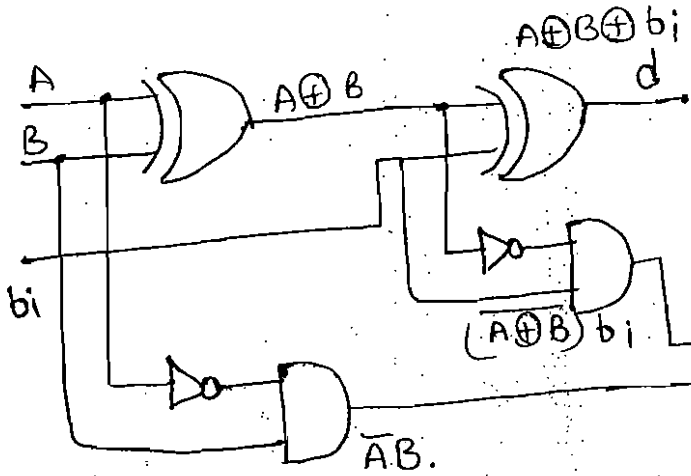


$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + \bar{A}Bb_i + AB b_i$$

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

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

Full-Subtractor by using two half-Subtractor



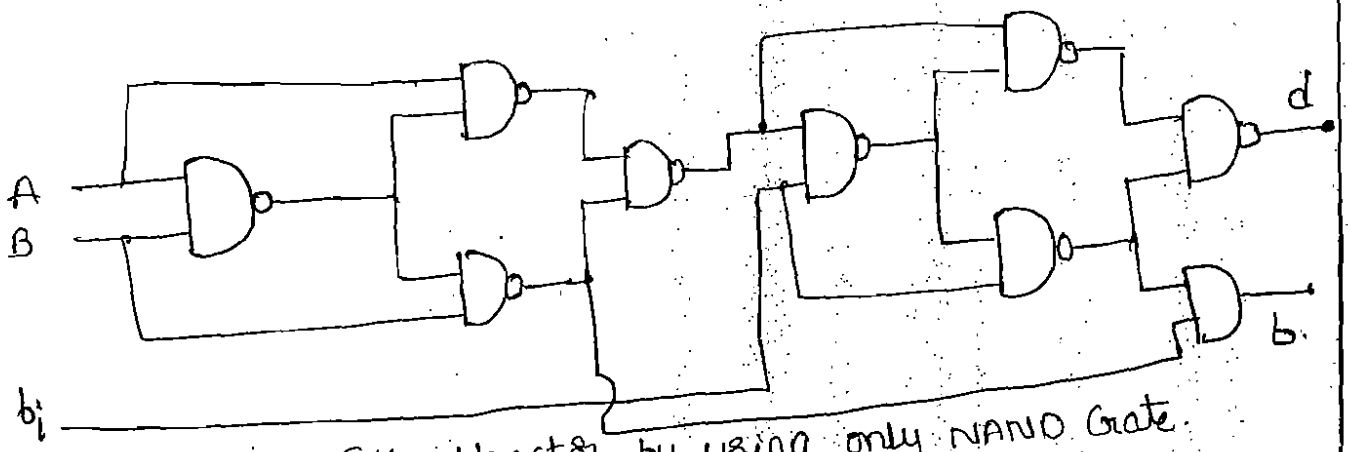
NAND logic:-

$$d = A \oplus B \oplus b_i = \overline{(A \oplus B) \oplus b_i} = \overline{(A \oplus B)(\overline{A \oplus B})b_i} \cdot \overline{b_i(A \oplus B)b_i}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B}) = \overline{\bar{A}B + b_i(\overline{A \oplus B})}$$

$$= \overline{\bar{A}B} \cdot \overline{b_i(\overline{A \oplus B})} = B(\bar{A} + \bar{B}) \cdot \overline{b_i[b_i + (\overline{A \oplus B})]}$$

$$= \overline{B \cdot \bar{A}B} \cdot \overline{b_i[\bar{b_i} \cdot (A \oplus B)]}$$



Full subtractor by using only NAND Gate.

NOR Logic.

$$d = \overline{\overline{A \oplus B \oplus b_i}}$$

$$= \overline{(A \oplus B)\, b_i + (\overline{A \oplus B})\, \overline{b_i}}$$

$$= \overline{[(A \oplus B) + (\overline{A \oplus B})\, \overline{b_i}]\, [b_i + (\overline{A \oplus B})\, \overline{b_i}]}$$

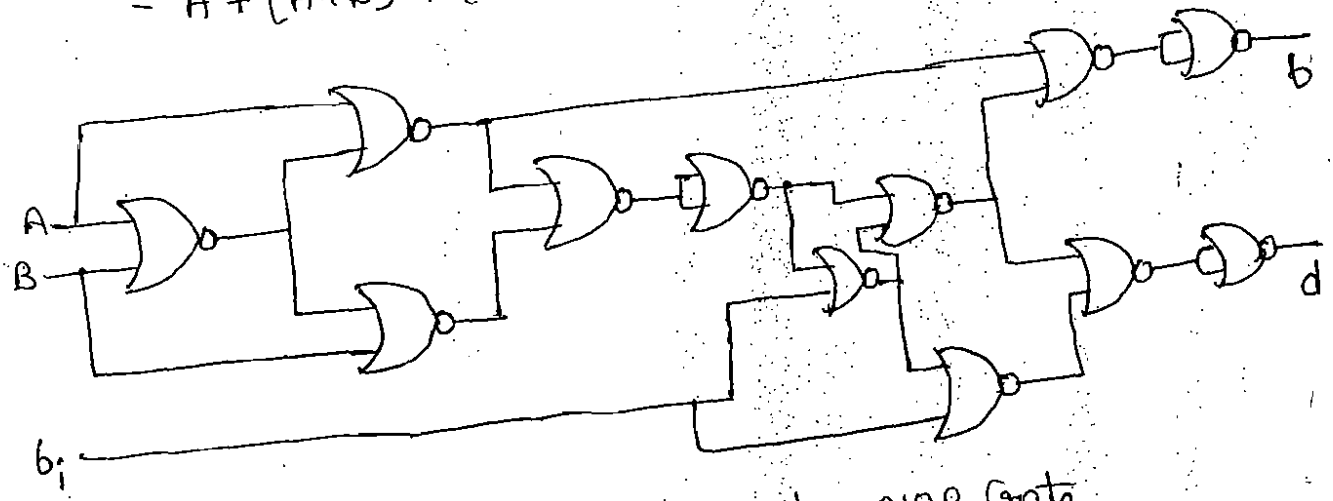
$$= \overline{(A \oplus B) + (\overline{A \oplus B}) + b_i} + \overline{b_i + (A \oplus B) + b_i}$$

$$= \overline{\overline{(A \oplus B) + (\overline{A \oplus B}) + b_i} + \overline{b_i + (\overline{A \oplus B}) + b_i}}$$

$$b = \overline{A}B + b_i(\overline{A \oplus B})$$

$$= \overline{A}(A + B) + (\overline{A \oplus B})[A \oplus B + b_i)$$

$$= \overline{\overline{A + (\overline{A+B})} + \overline{(A \oplus B) + (A \oplus B) + b_i}}$$

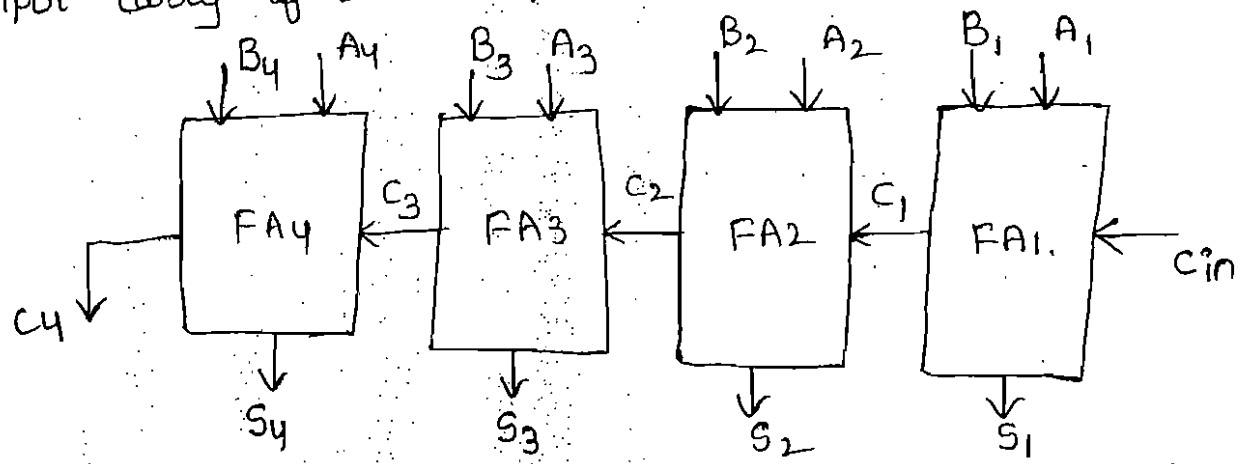


Full subtractor by using only NOR Gate

Applications of full adders :-

Binary parallel adder :-

A Binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.



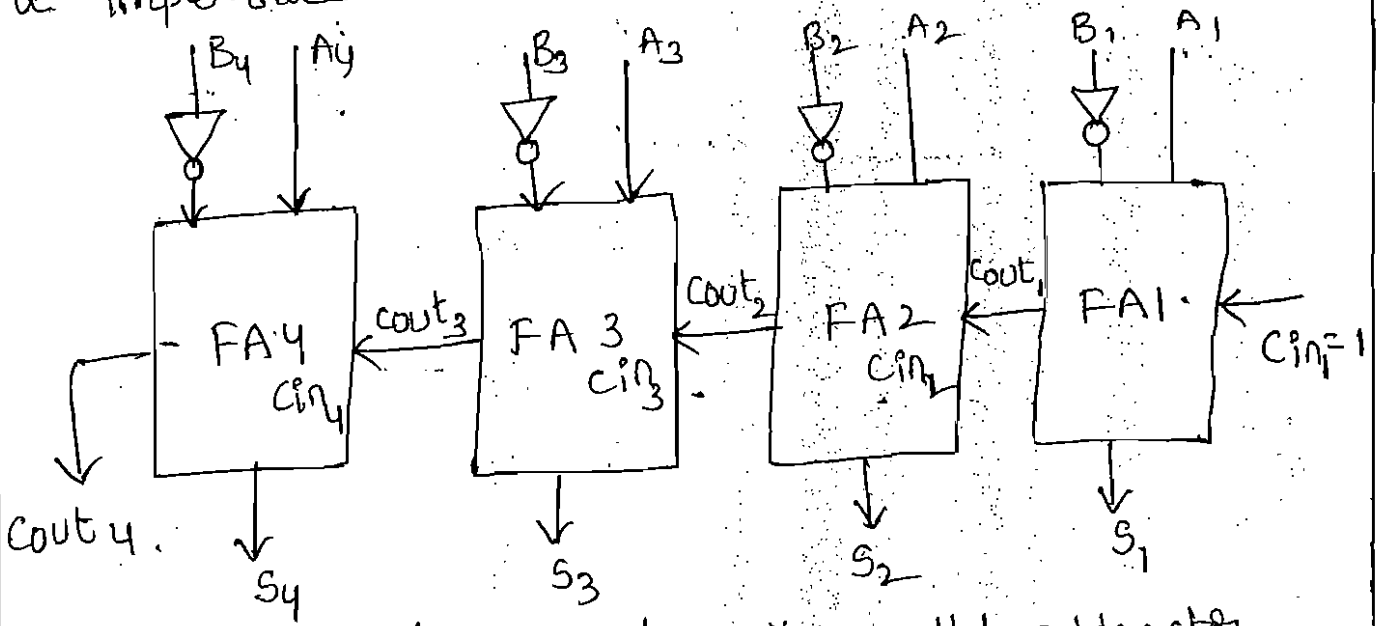
Logic diagram of 4-bit binary parallel adder.

The inter-connection of full-adder (FA) circuits to provide a 4-bit parallel adder. The augend bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower-order bit. The input carry to the adder is Cin and the output carry is C_4. The S outputs generate the required sum bits. When the 4-bit full adder circuit is enclosed within an IC package, it has four terminals for the augend bits, four terminals for the addend bits, four terminals for the sum bits and two input terminals for the input and output carries.

The parallel adder in which the carry-out of each full adder is the carry-in to the next most significant adder is called a ripple carry adder. In the parallel adder the carry-out of each stage is connected to the carry-in of the next stage. The Sum and carry out bits of any stage cannot be produced, until some time after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry, which lead to a time delay in the addition process.

4-bit parallel subtractor:-

The subtraction of binary numbers can be carried out most conveniently by means of complements. The subtraction A-B can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with not gate (inverters).



Logic diagram of 4-bit parallel subtractor.

Binary adder - Subtractor :-

The addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each Full adder. The M mode input controls the operation.

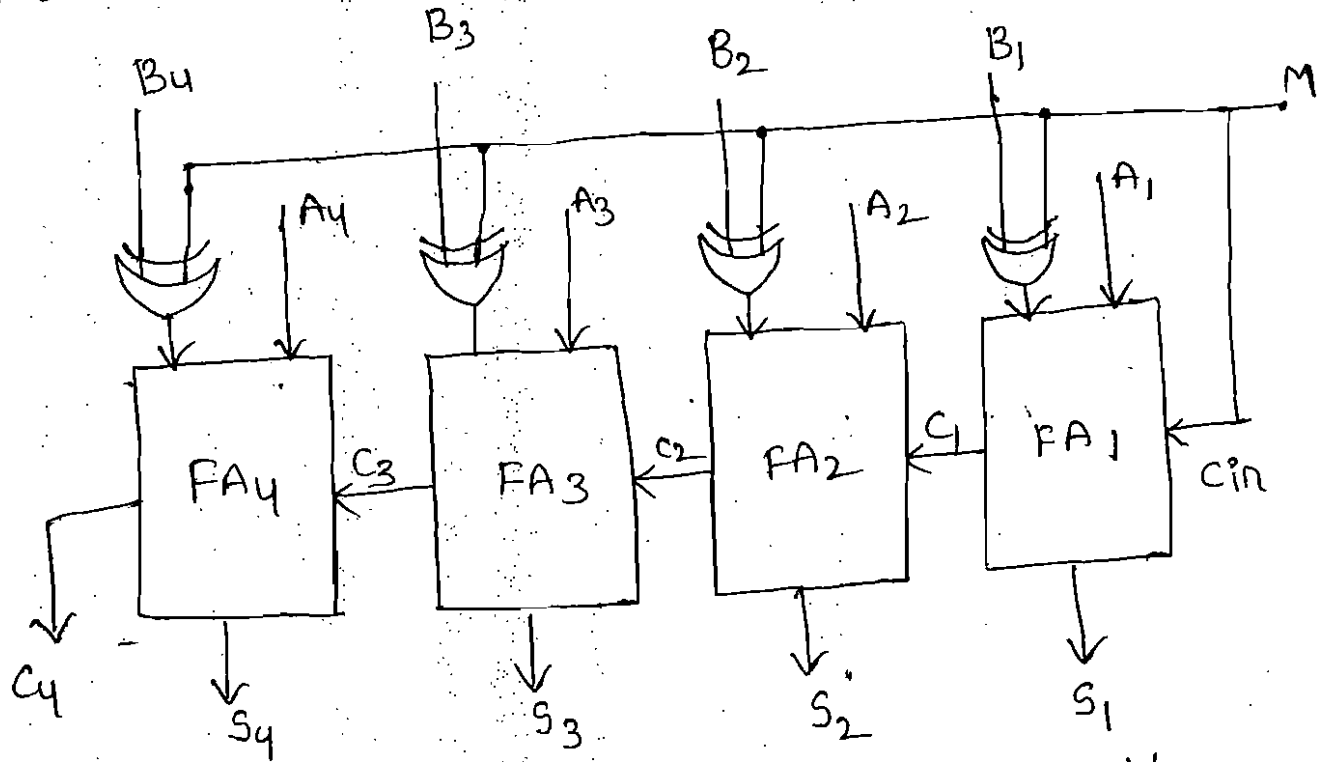
→ when M = 0, the circuit is an adder.

→ when M = 1, the circuit is an subtractor.

Each XOR gate receives input M and one of the inputs of B.

→ when M = 0, $B \oplus 0 = B$. The full adder receives the value of B, the input carry is '0' and the circuit performs A+B.

→ when M = 1, $B \oplus 1 = \overline{B}$, The full adder receives the value of $\overline{B}$, the input carry is '1' and the circuit performs A-B.

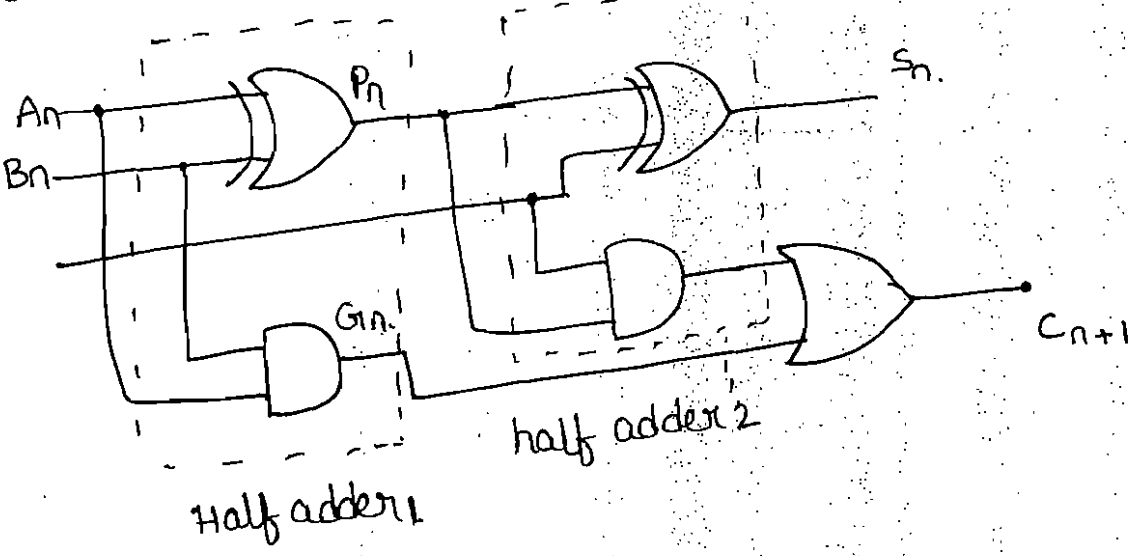


Logic diagram of a 4-bit binary - adder - subtractor.

LOOK-A-HEAD-CARRY ADDER :-

The parallel adder, the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder.

The look-ahead-carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously. The methode of speeding up the process is based on the two additional functions of the full adder, called the carry generate and carry propagate functions.



carry generate :-

consider one full adder stage, nth stage of a parallel adder. carry is generated only if both the input bits are 1, that is, if both the bits A and B are is, a carry has to be generated in this stage regardless of whether the input carry Cin is a 0 or a 1. If Gi is a carry-generation function.

$$G_i = A \cdot B.$$

The present bit as the n^{th} bit, then G_i rewrite as a

$$G_n = A_n \cdot B_n$$

Carry propagation :-

A carry is propagated if any one of the two input bits A & B are 0, a carry will never be propagated. on the other hand, if both A and B are 1, then it will not propagate the carry but will generate the carry.

If P is taken as a

$$P = A \oplus B.$$

The present bit as the n^{th} bit, then P rewrite as a

$$P_n = A_n \oplus B_n.$$

For the final sum and carry outputs of the n^{th} stage,

$$S_n = P_n \oplus C_n$$

$(\because P_n = A_n \oplus B_n)$

$$C_{on} = C_{n+1} = G_n + P_n C_n$$
$$= A_n \cdot B_n + P_n C_n$$
$$= A_n \cdot B_n + (A_n \oplus B_n) C_n.$$

Based on these, the expression for the carry-outs of various full-adders are.

$n = 1$, $C_1 = G_0 + P_0 C_0$

$$= G_0 + (A_0 \oplus B_0) C_0$$
$$= A_0 \cdot B_0 + (A_0 \oplus B_0) C_0.$$

$n = 2$

$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$

$n = 3$

$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 C_0$

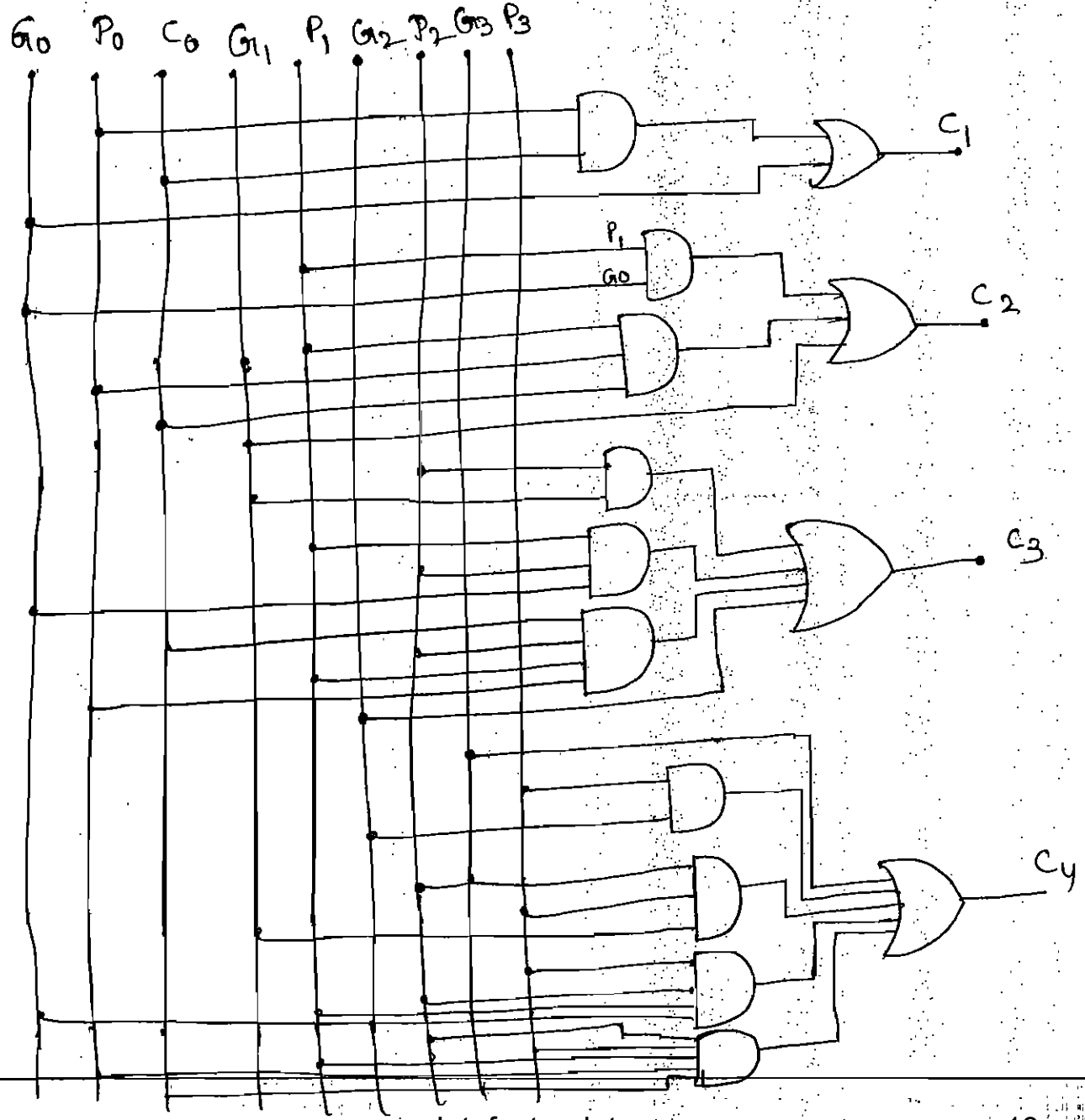
$n = 4$

$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 P_1 G_0 +$
$P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$

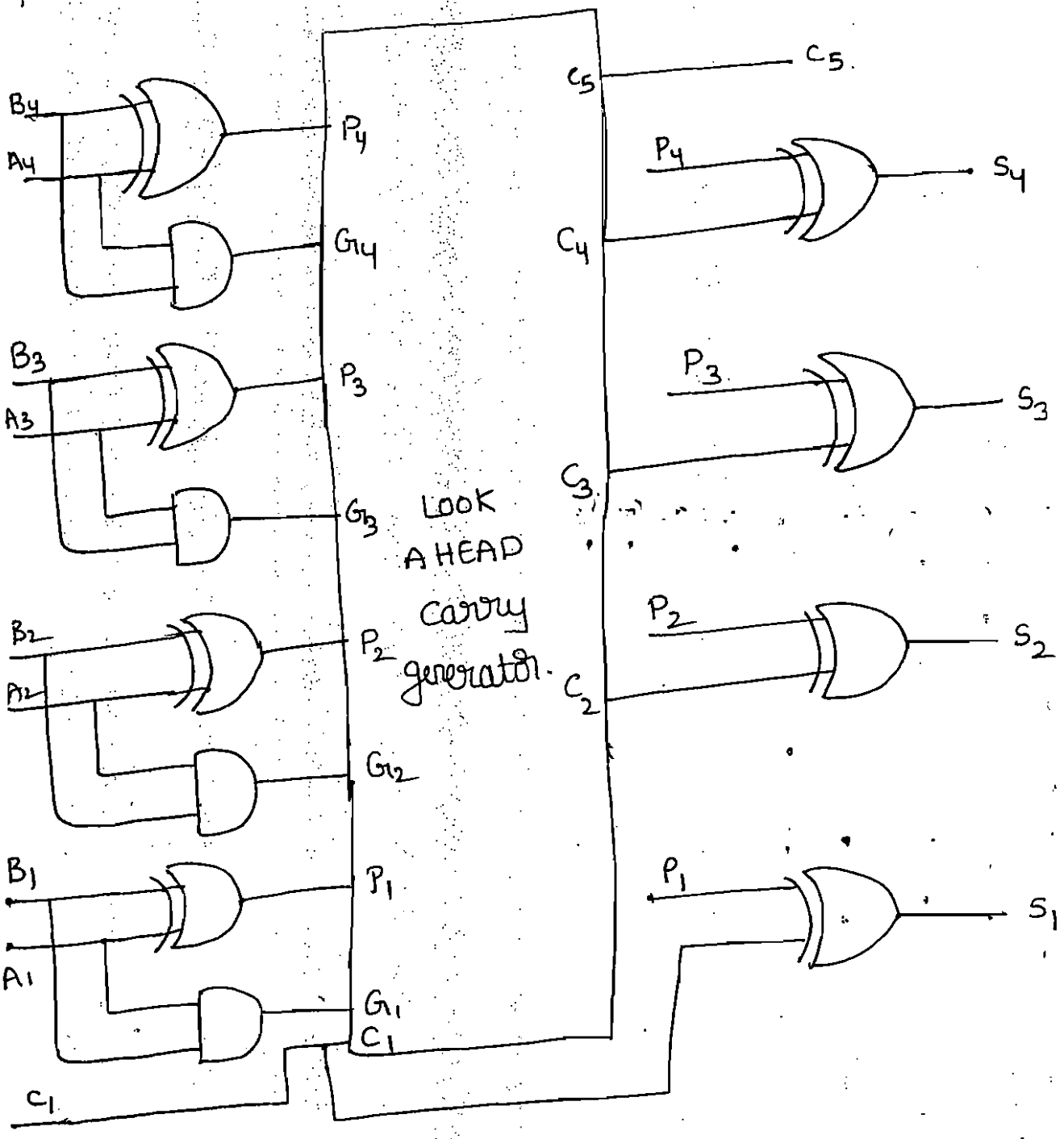
The general expression for n-stages.

$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1}$

$= G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \cdots + P_{n-1} \cdots P_0 \cdot C_0$

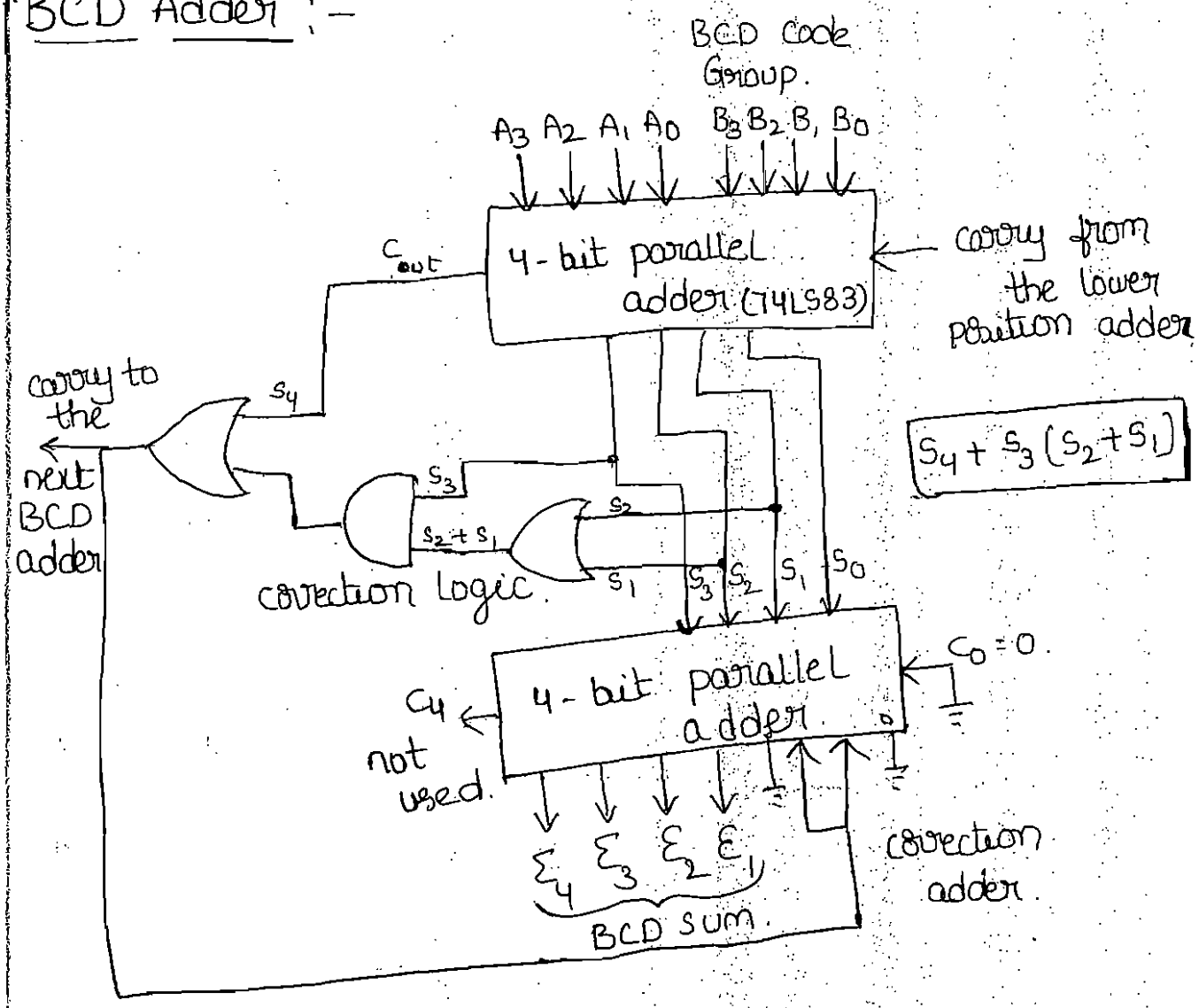


The block diagram of a 4-stage look-ahead-carry parallel adder is shown in the below figure.



Basic logic diagram of a 4-bit look-Ahead carry adder.

BCD Adder :-



→ In BCD adder, Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.

→ For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.

→ where the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result.

→ In above figure 4-bit parallel adder (using IC 74LS83). The two BCD groups A_3, A_2, A_1, A_0 and B_3, B_2, B_1, B_0 are applied to a 4-bit parallel adder.

The adder output will be C_4, S_3, S_2, S_1, S_0. where C_4 is taken as a S_4.

→ when both the inputs are 1001. The Sum output S_4, S_3, S_2, S_1, S_0 can range from 00000 to 10010.

→ The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001.

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

→ In above Table shows the cases for greater than 1001.

The Sum will be high → wherever $S_4 = 1$,

→ wherever $S_3 = 1$ and either S_2 & S_1 & both are 1.

Then $X = S_4 + S_3(S_2 + S_1)$.

wherever $X = 1$, it is necessary to add the 0110 to the Sum bits.

The circuit consists of three basic parts. The BCD code groups A_3, A_2, A_1, A_0 and B_3, B_2, B_1, B_0 are added together in upper 4-bit parallel adder to produce the Sum $S_4 S_3 S_2 S_1 S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the add correction 0110 to the Sum bits only when X = 1, producing final BCD Sum output represented by $\varepsilon_3 \varepsilon_2 \varepsilon_1 \varepsilon_0$.

when X = 0, there is no carry and no correction.

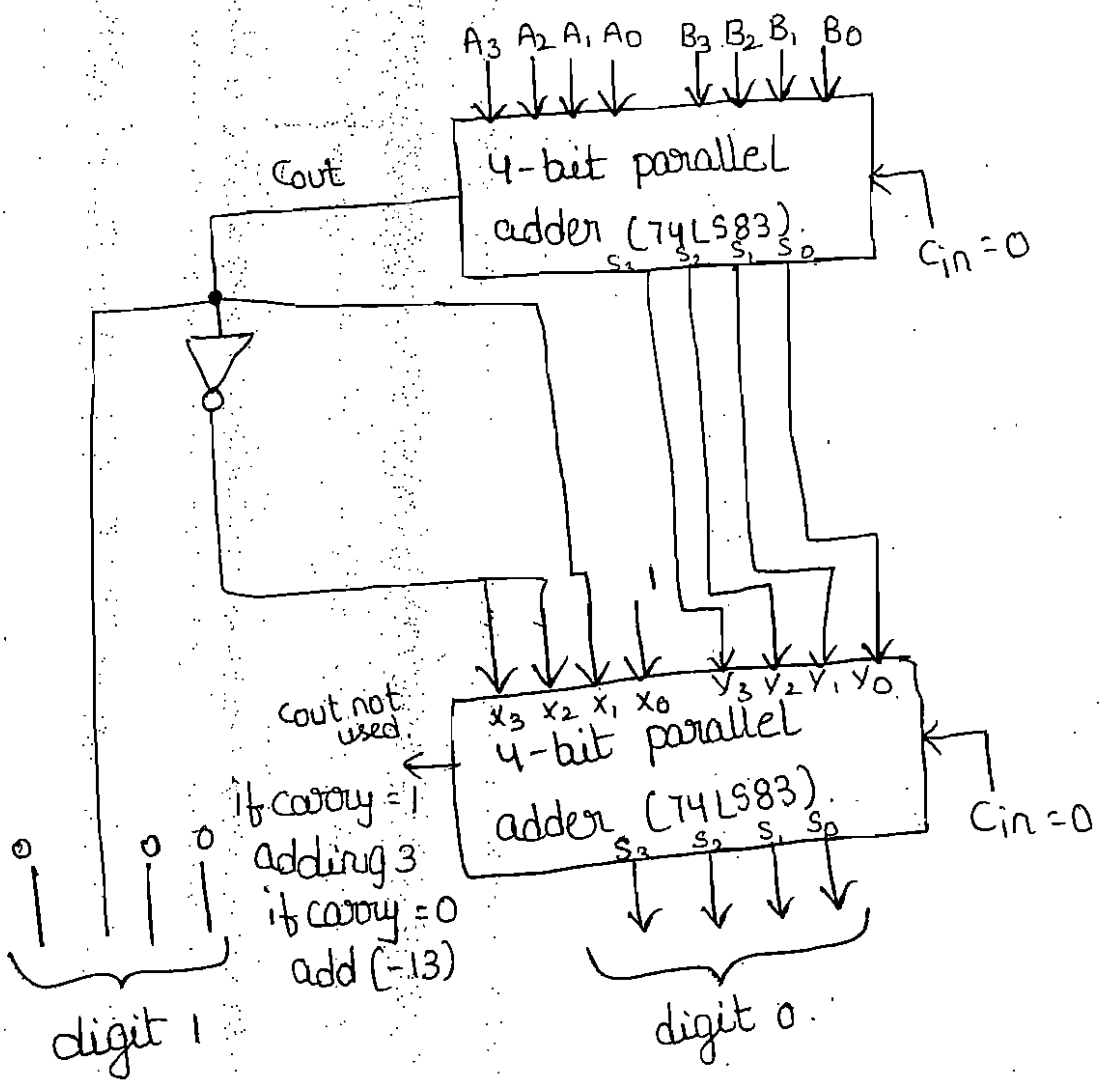
In such cases $\varepsilon_3 \varepsilon_2 \varepsilon_1 \varepsilon_0 = S_3 S_2 S_1 S_0$. Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder.

EXCess-3 Adder :-

→ In EXCess-3 addition.

1. Add two XS-3 code groups.
2. If carry = 1 add 0011
   if carry = 0 subtract 0011, or add 1101 (13 in decimal).

In figure The augend (A_3, A_2, A_1, A_0) and addend ($B_3 B_2 B_1 B_0$) in XS-3 added using the 4-bit parallel adder. If the carry is a 1, then 0011 is added to the Sum bits $S_3 S_2 S_1 S_0$ of the upper adder in the lower 4-bit parallel adder. If the carry is a '0' then 1101 is added to the Sum bits.



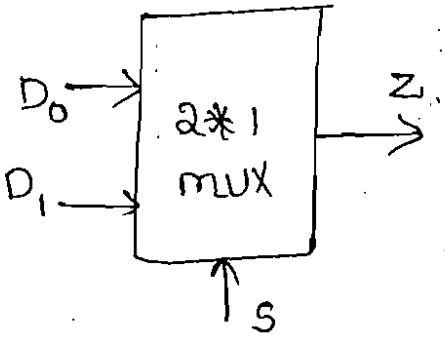
The final Answer in XS-3 form.

Multiplexers (data selectors).

multiplexing means sharing. A multiplexer or data selector is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the output. The routing of the desired data inputs to the output is controlled by SELECT inputs. Normally there are 2^n input lines and n select lines and one output.

Basic 2-input multiplexer :-

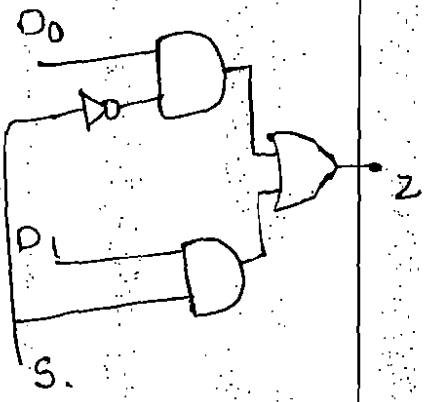
In 2-input multiplexer have 2-inputs they are D_0 and D_1. and one select line S. and output is Z.



Block diagram.

S	Z
0	D_0
1	D_1

Truth table.



$Z = \bar{S}D_0 + SD_1$

Logic diagram.

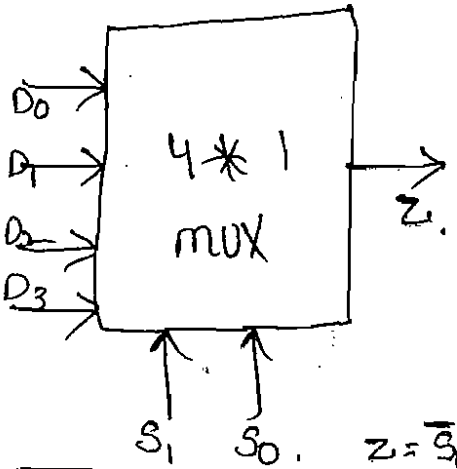
The logic levels applied to the S inputs determines which AND gate is enabled. So that its data input passes through the OR gate to the output.

When S = 0, AND gate 1 is enabled and AND gate 2 is disabled, S_0, $Z = D_0$

S = 1, AND gate 2 is Enabled and AND gate 1 is disabled, S_0, Z = D1.

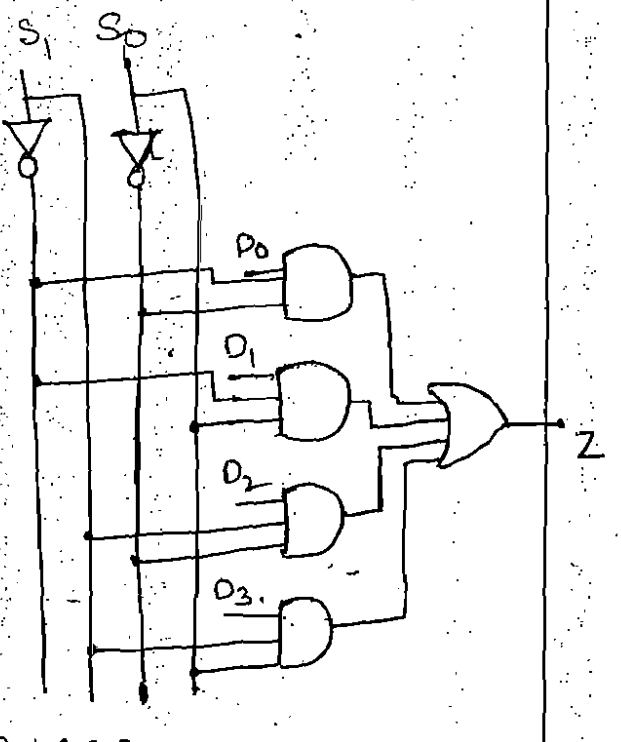
Basic 4-input multiplexer :-

Block diagram



Truth table

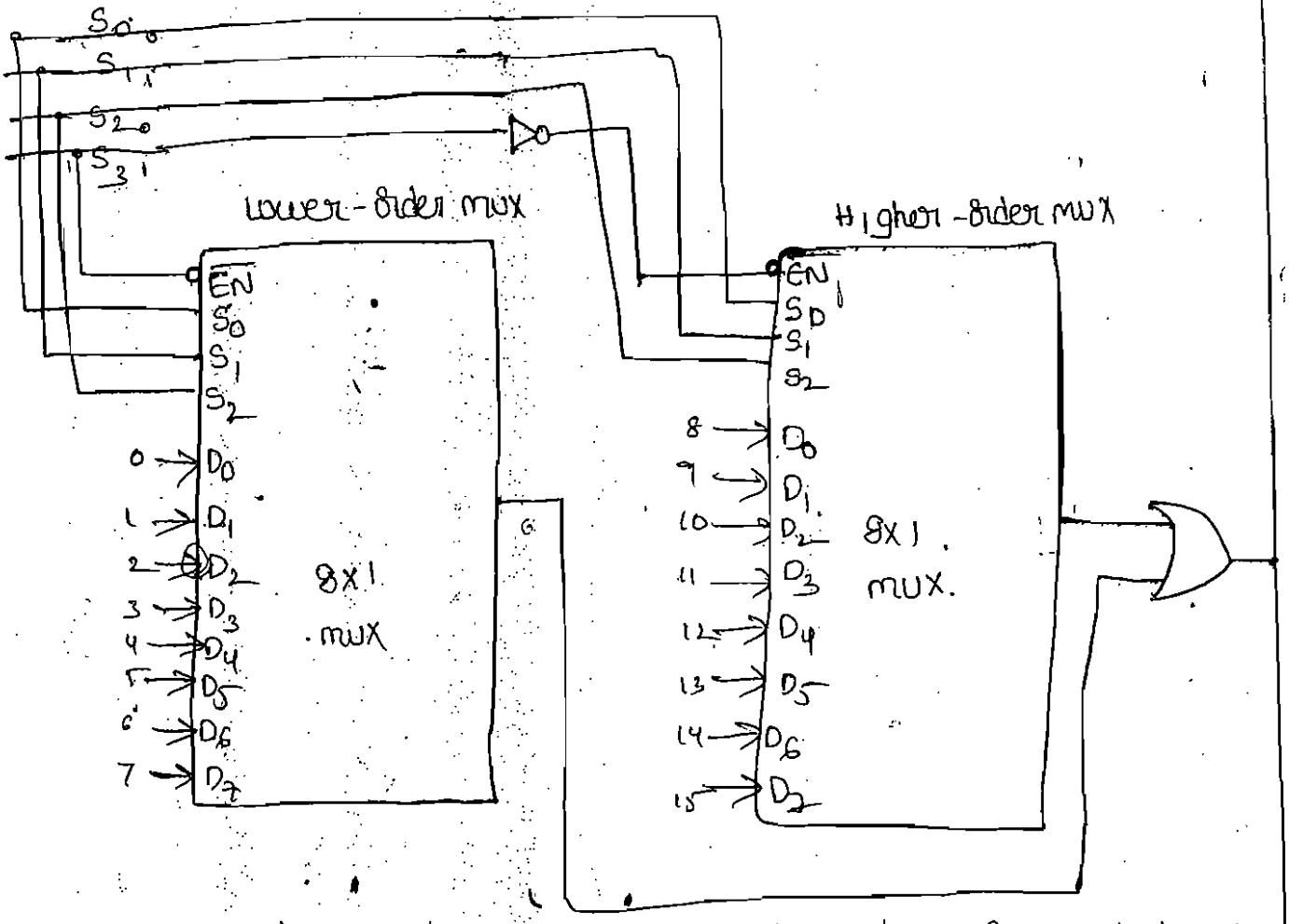
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



$Z = \bar{S}_0\bar{S}_1D_0 + S_0\bar{S}_1D_1 + \bar{S}_0S_1D_2 + S_0S_1D_3$

The 16-input multiplexers from Two 8-input multiplexers:-

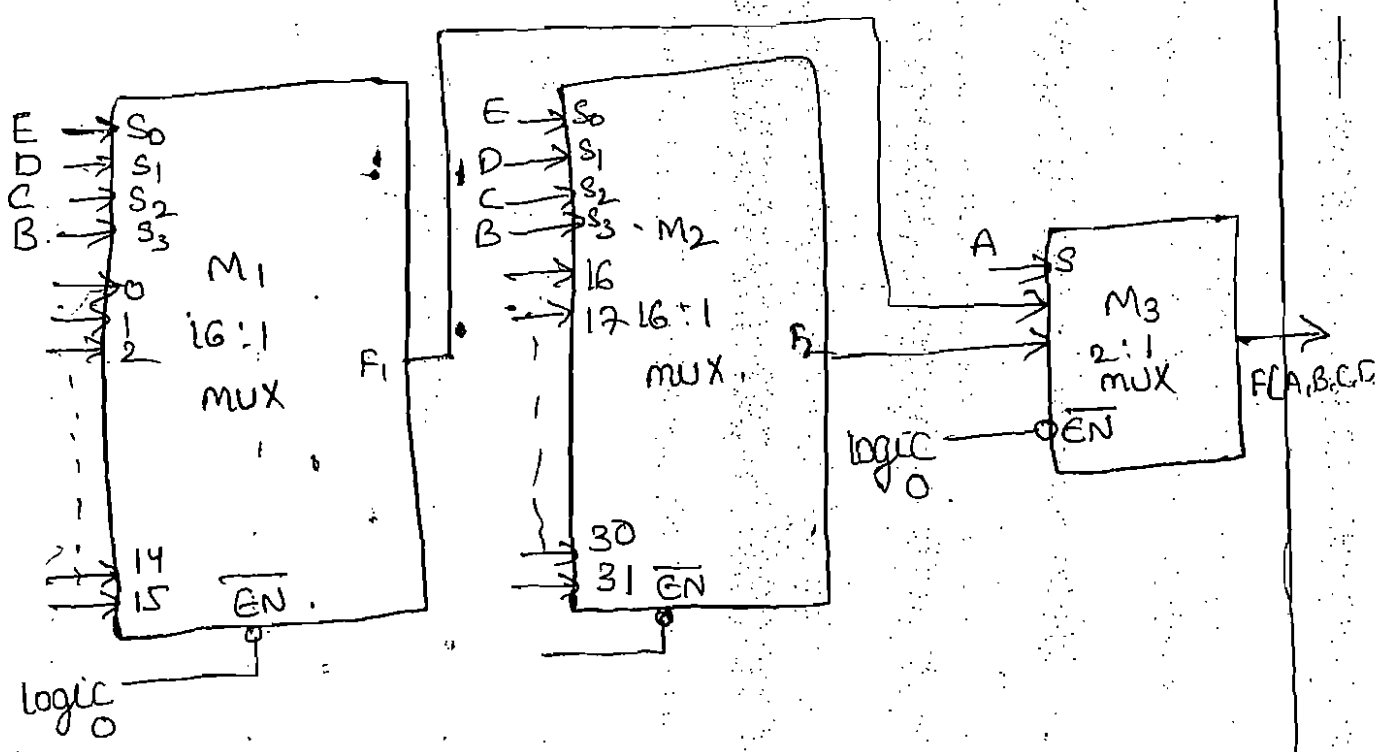
To use two 8-inputs multiplexers to get a 16-inputs multiplexers one OR gate and one inverter are also required. The four select inputs S_3, S_2, S_1 and S_0 and will select one of the 16 inputs to pass through to X. The S_3 input determines which multiplexer is enabled. When $S_3 = 0$, the left multiplexer is enabled and S_2, S_1 and S_0 inputs determine which of its data inputs will apper at its output and pass through the OR gate to X. When $S_3 = 1$, the right multiplexers is enabled and S_2, S_1 and S_0 inputs select one of its data inputs for passage to output X.



Logic diagram for cascading of two 8x1 mux to get 16x1

Design of a 32*1 mux using Two 16*1 muxs and one 2*1 mux :-

To obtain a 32*1 mux using two 16*1 muxes and one 2*1 mux. A 32*1 mux has 32 data inputs. so it requires five data select lines. since a 16*1 mux has only four data select lines, the inputs B, C, D, E are connected to the data select lines of the both 16*1 muxes and the most significant input A is connected to the single data select line of the 2*1 mux. For the values of BCDE = 0000 to 1111, inputs 0 to 15 will appear at the input terminals 0 of the 2:1 mux through the output F_1 of the First 16*1 mux and inputs 16 to 31 will apper at the input terminal 1 of the 2*1 mux through the output F_2 of the second 16*1 mux. For A = 0, output $F = F_1$, for A = 1, output $F = F_2$.

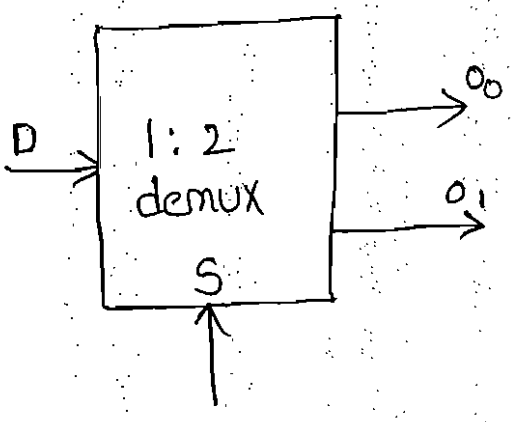


DEMULTIPLEXERS :-

A demultiplexer performs the reverse operation; it takes a single input and distributes it over several outputs. so a demultiplexer is also called as a "data distributors". Since it transmits the same data to different destinations. A demultiplexer is a 1-to-N device.

1-line to 2-line demultiplexer :-

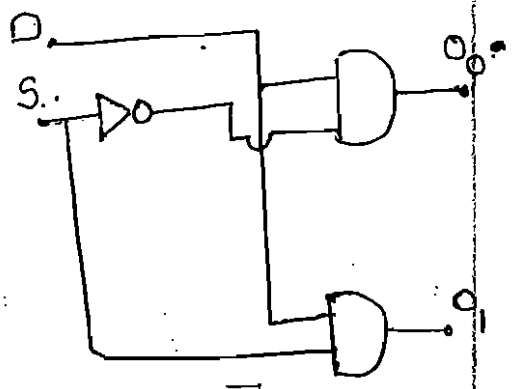
The input data line goes to all of the AND gates. The select line S enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output lines.



Block diagram

S	O_0	O_1
0	0	D
1	D	0

Truth table



$O_0 = D\bar{S}$

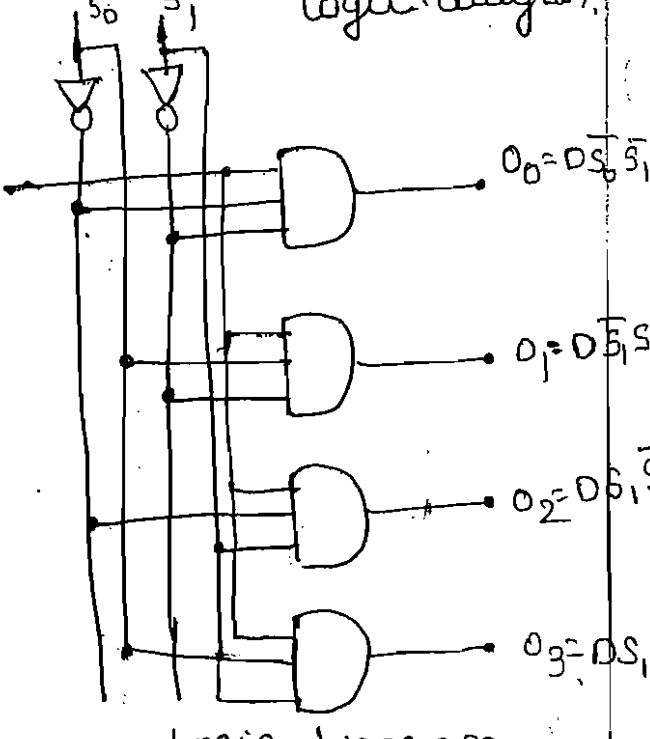
$D_1 = DS$

Logic diagram.

1-line to 4-line demultiplexer :-

S_1	S_0	D_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Truth table.

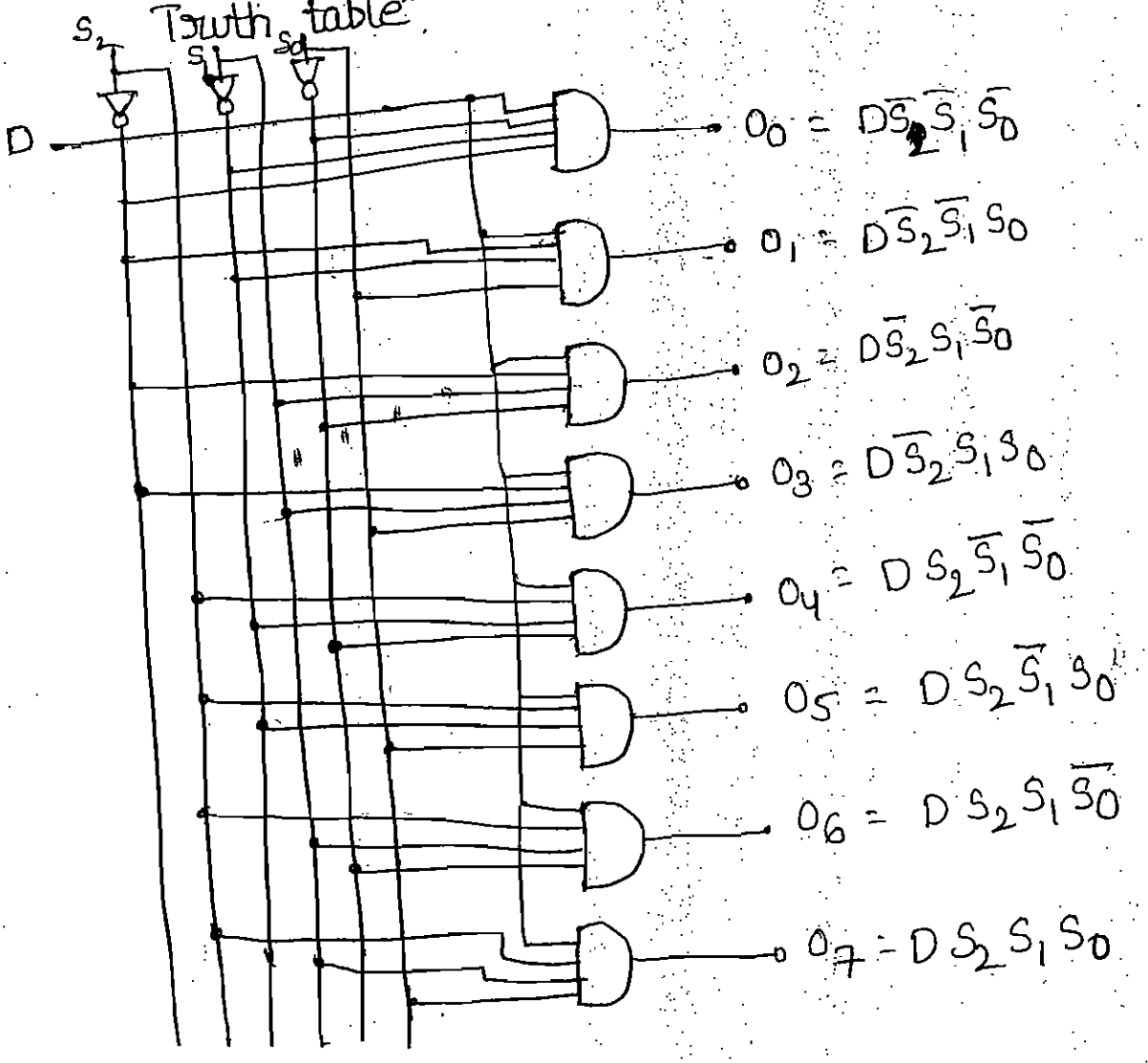


Logic diagram.

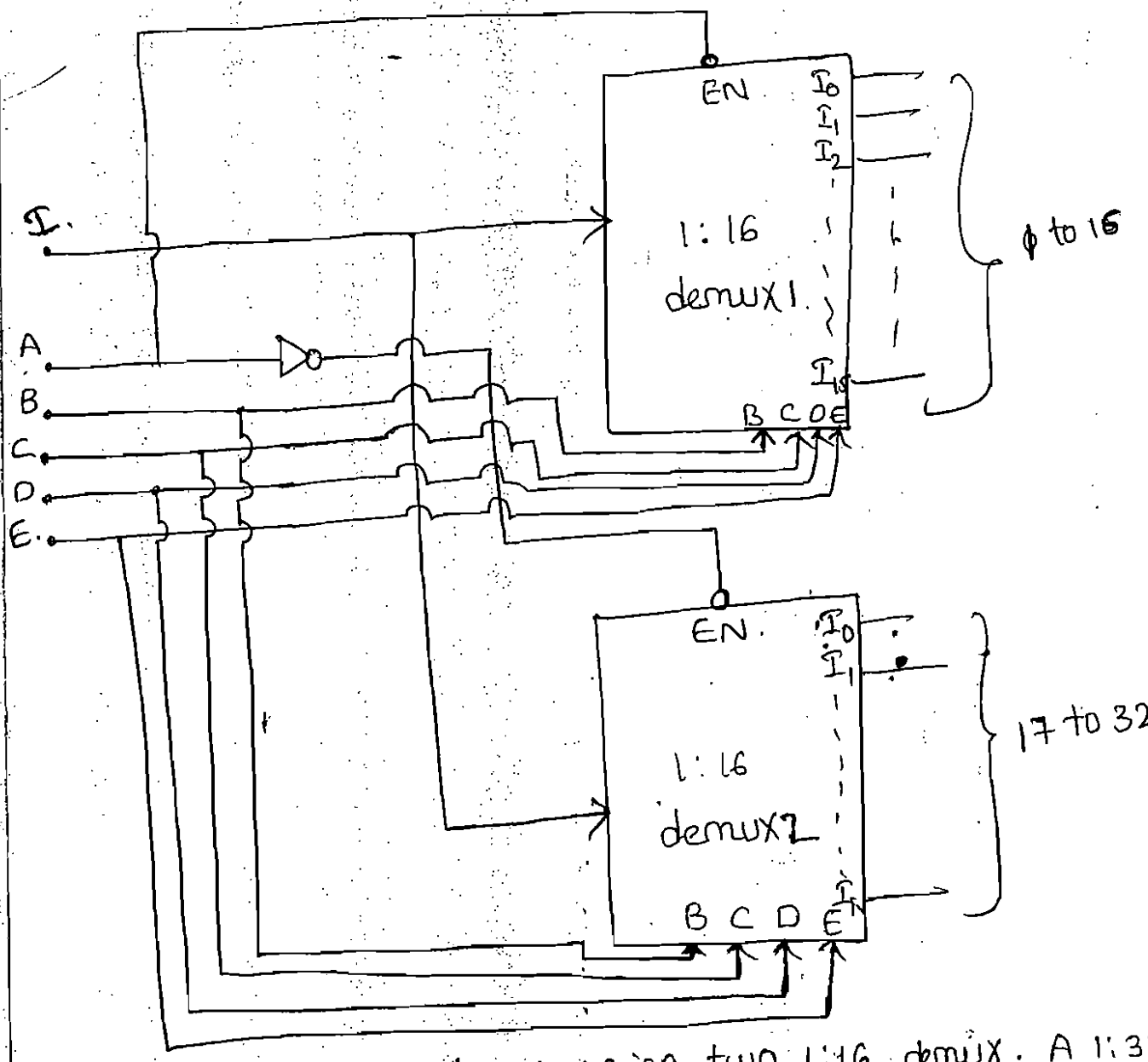
1-line to 8-line demultiplexer:-

S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0

Truth table.



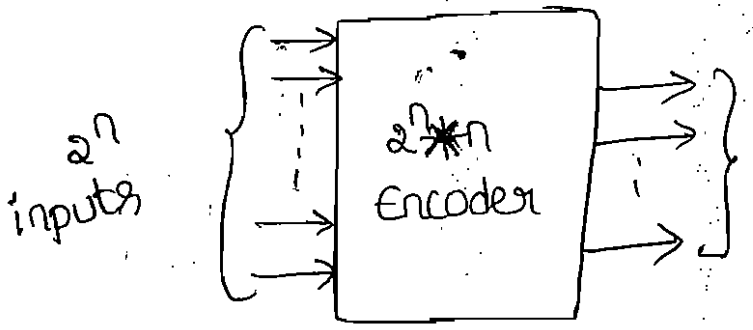
design of 1:32 demux using two 1:16 demux.



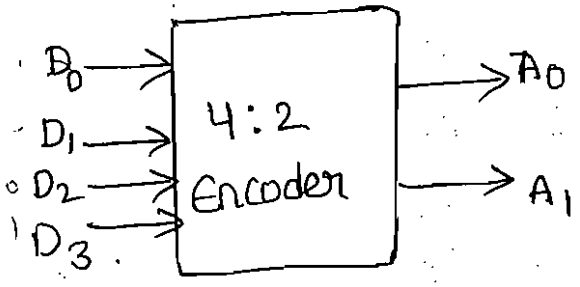
To obtain 1:32 demux using two 1:16 demux. A 1:32 demux has 32 data outputs. So it requires five data select lines. Since 1:16 demux has only four select inputs. the inputs B, C, D, E are connected to the data select lines of both the 1:16 demuxes and the most significant input A is connected to the single data select line of the both 1:16 demuxs $\overline{EN}$ input. 1 to 16 will apper in the first demux when (A=0). 17 to 32 will apper in the second demux when A=1.

Encoders :-

An encoder is a device whose inputs are decimal digits and/or alphabetic characters and whose outputs are the coded representation of those inputs, An encoder is a device which converts familiar numbers or symbols into coded format. The Encoder has 2^n inputs and n outputs.



4bit - Encoder :-



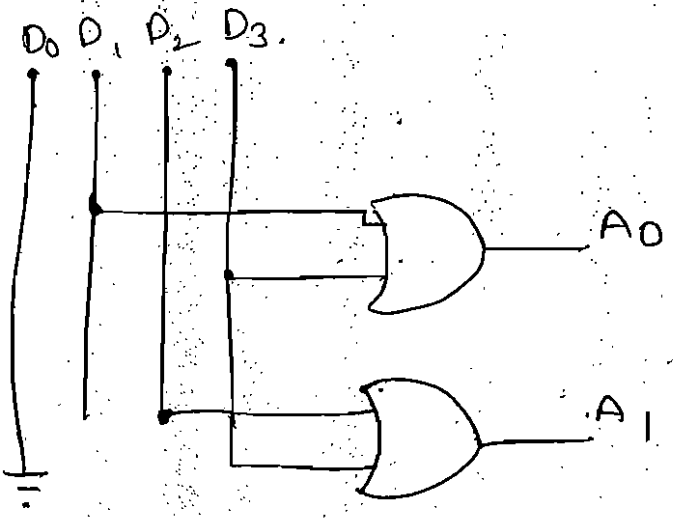
Block diagram.

inputs	outputs A_1 A_0
D_0	0 0
D_1	0 1
D_2	1 0
D_3	1 1

Truth table.

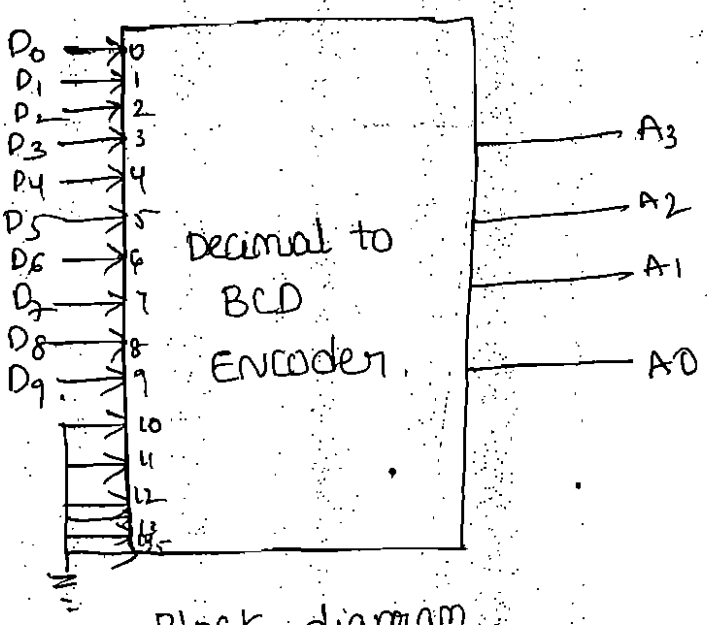
$A_1 = D_2 + D_3$

$A_0 = D_1 + D_3$.



Decimal to BCD Encoder :-

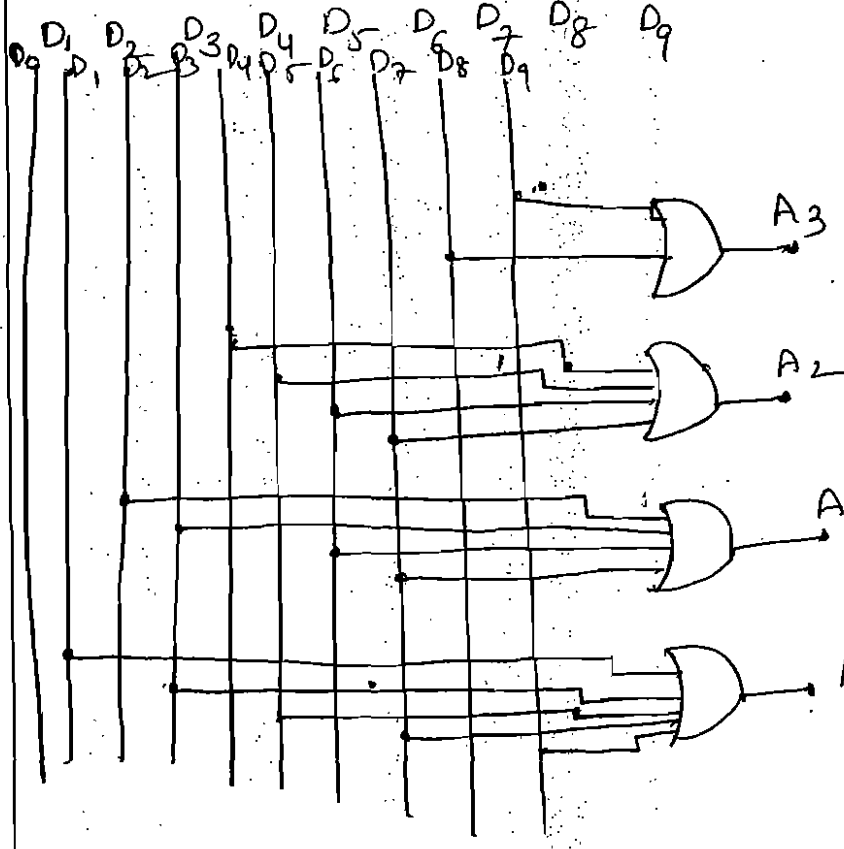
In this type of encoder has 10 inputs - one for each decimal digit, and 4 outputs corresponding to the BCD code.



Block diagram.

inputs. Decimal		Binary output A_3 A_2 A_1 A_0
D_0	0	0 0 0 0
D_1	1	0 0 0 1
D_2	2	0 0 1 0
D_3	3	0 0 1 1
D_4	4	0 1 0 0
D_5	5	0 1 0 1
D_6	6	0 1 1 0
D_7	7	0 1 1 1
D_8	8	1 0 0 0
D_9	9	1 0 0 1

Truth table.



$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$

$A_1 = D_2 + D_3 + D_6 + D_7$

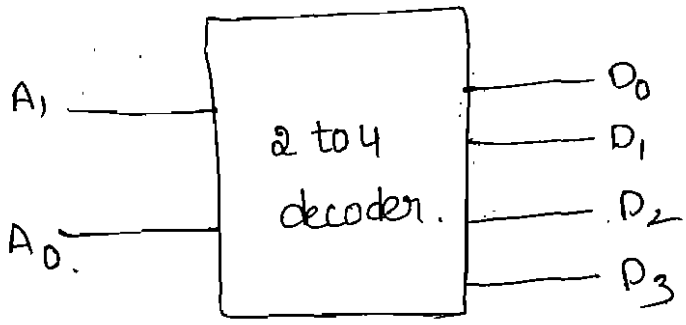
$A_2 = D_4 + D_5 + D_6 + D_7$

$A_3 = D_8 + D_9$.

Decoders :-

A decoder is a logic circuit that converts an N-bit binary input code into 2^n output lines such that only one output line is activated for each one of the possible combinations of inputs. For each of these input combinations, only one of the output will be activated (high), all the other outputs will remain inactive. (low). Some decoders are designed to produce active low output, while all the other outputs remain high.

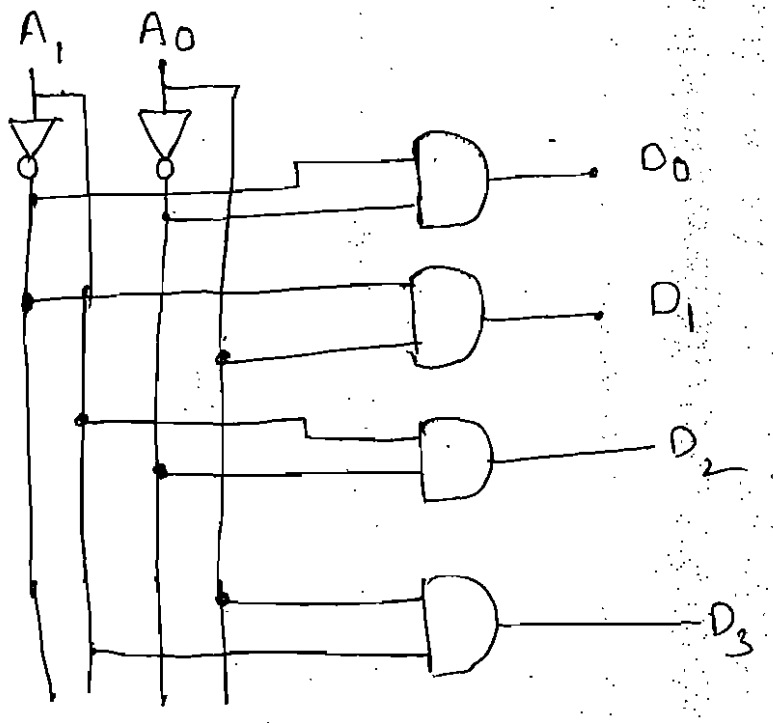
2-4 line decoder :-



Block diagram.

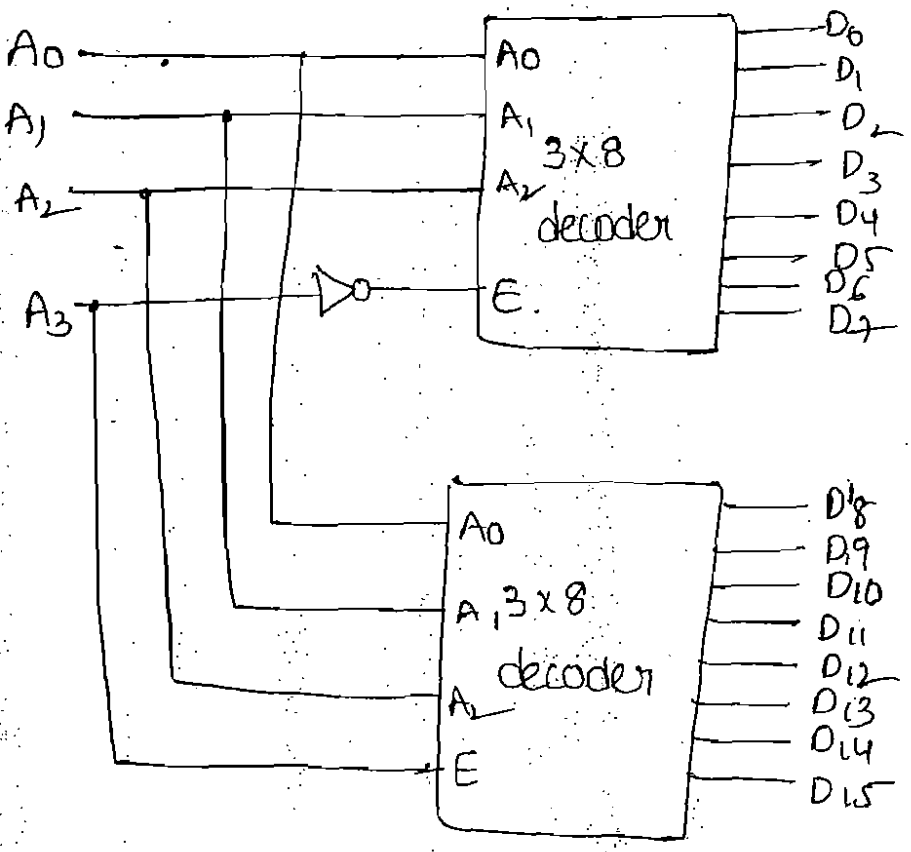
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Truth table.



4-to-16 decoder from two 3-to-8 decoders :-

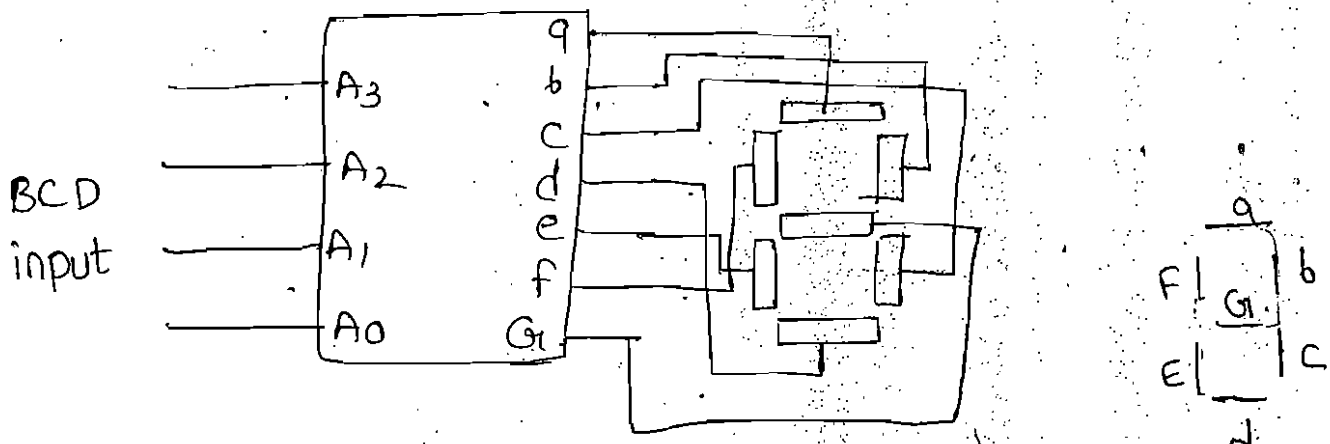
Decoders with enable inputs can be connected together to form a larger decoder. To obtain a 4-to-16 decoder it requires two 3-to-8 decoders. The most significant input bit A_3 is connected through an inverter to $\bar{E}$ on the upper decoder and directly to E on the lower decoder. Thus A_3 is low, the upper decoder is enabled and the lower decoder is disabled. The bottom decoder outputs all 0's, and top 8 outputs. generates minterms when A_3 is high, the lower decoder is enabled and the upper decoder is disabled. The bottom decoder outputs generates minterms 1000 to 1111 while the outputs of the top decoder are all 0's.



logic diagram

Seven Segment Decoders :-

This type of decoders accepts the BCD code and provides outputs to energize seven segment display devices in order to produce a decimal read out. Each segment is made up of a material that emits light when current is passed through it. The most commonly used materials include LEDs, incandescent filaments and LCDs.



Block diagram.

Decimal digit	BCD input A_3 A_2 A_1 A_0	Seven segment code a b c d e f G
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 0 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 0 1 1 1

$a = \Sigma m(0,2,3,5,6,7,8,9) + d(10,11,12,13,14,15)$.

$b = \Sigma m(0,1,2,3,4,7,8,9) + d(10,11,12,13,14,15)$

$c = \Sigma m(0,1,3,4,5,6,7,8,9) + d(10,11,12,13,14,15)$.

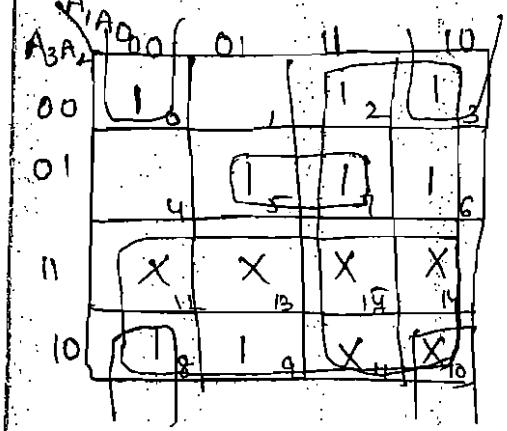
$d = \Sigma m(0,2,3,5,6,8,9) + d(10,11,12,13,14,15)$.

$e = \Sigma m(0,2,6,8) + d(10,11,12,13,14,15)$.

$f = \Sigma m(0,4,5,6,8,9) + d(10,11,12,13,14,15)$.

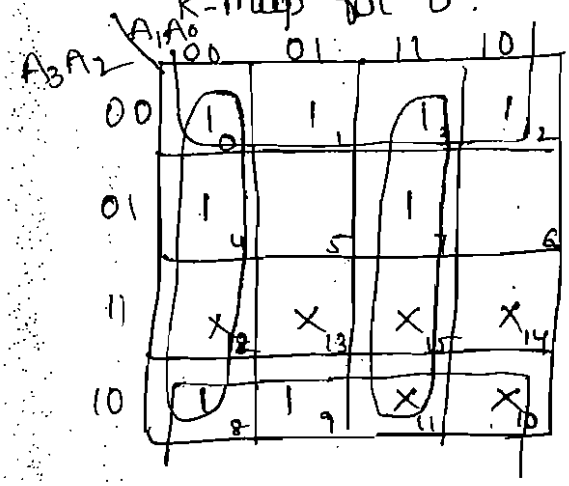
$g = \Sigma m(2,3,4,5,6,8,9) + d(10,11,12,13,14,15)$.

K-map for a.

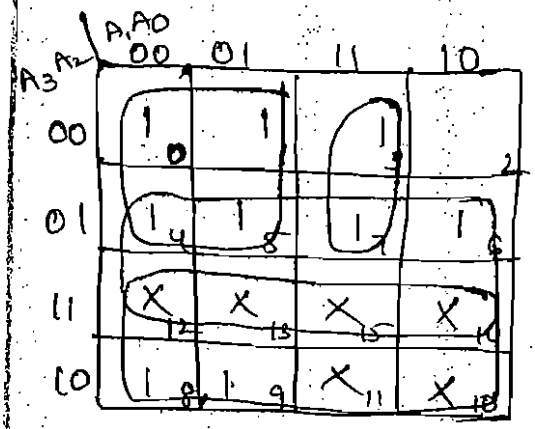


$a = A_1 + A_3 + \overline{A_2}\,\overline{A_0} + \overline{A_3}A_2A_0$

K-map for b.

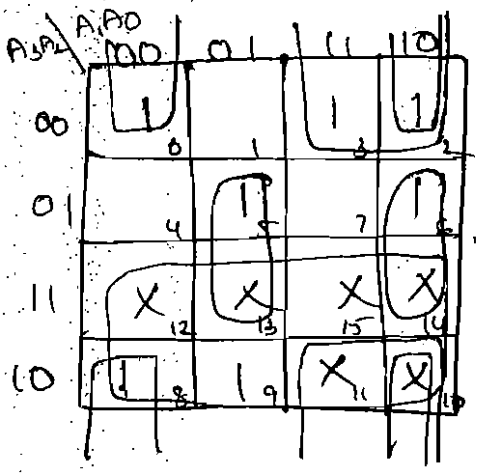


$b = \overline{A_2} + A_1\overline{A_0} + A_1A_0$



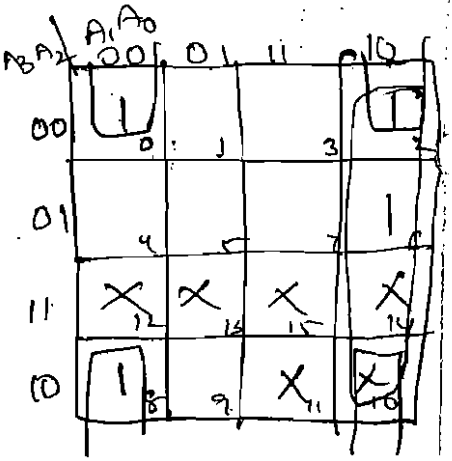
K-map for c

$c = A_2 + A_3 + \overline{A_3}\,\overline{A_1} + \overline{A_3}A_1A_0$



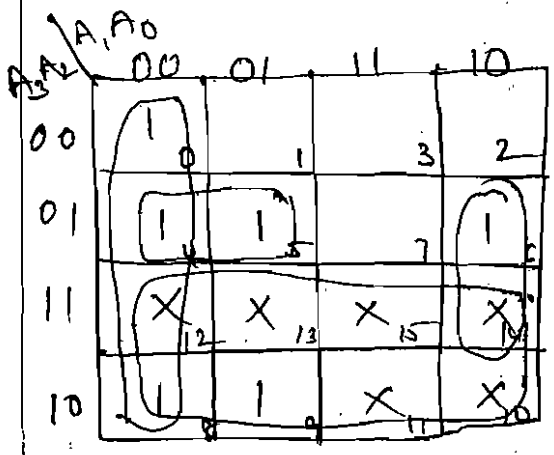
K-map for d

$d = A_3 + \overline{A_2}A_1 + A_2\overline{A_1}A_0 + A_2A_1\overline{A_0} + \overline{A_2}\,\overline{A_0}$

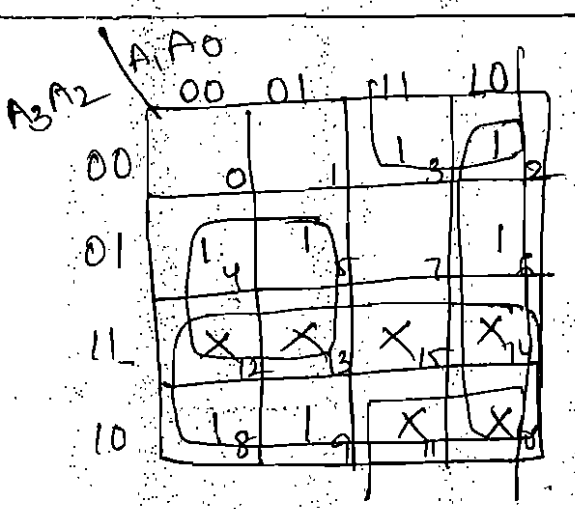


K-map for e

$e = A_1\overline{A_0} + \overline{A_2}\,\overline{A_0}$



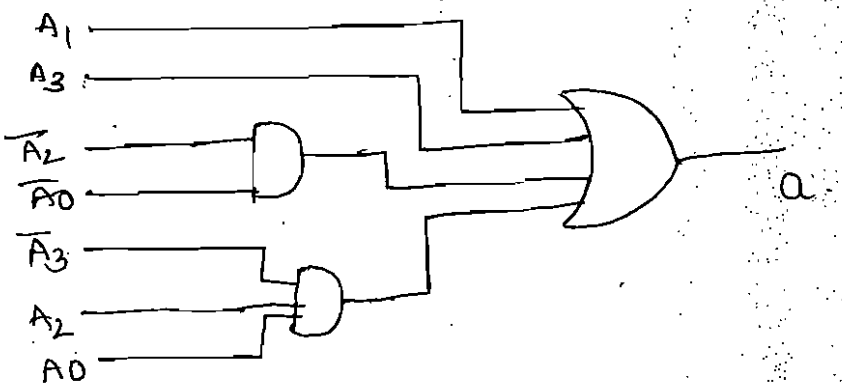
K-map for f.

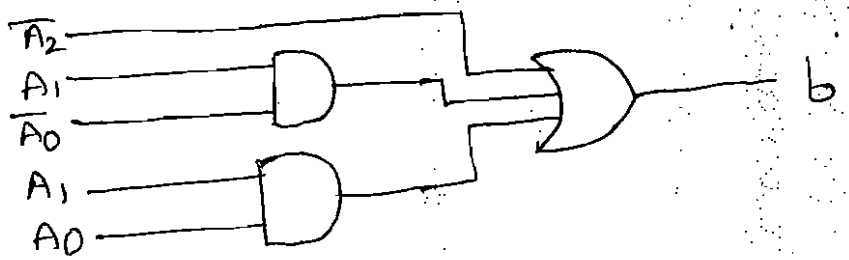


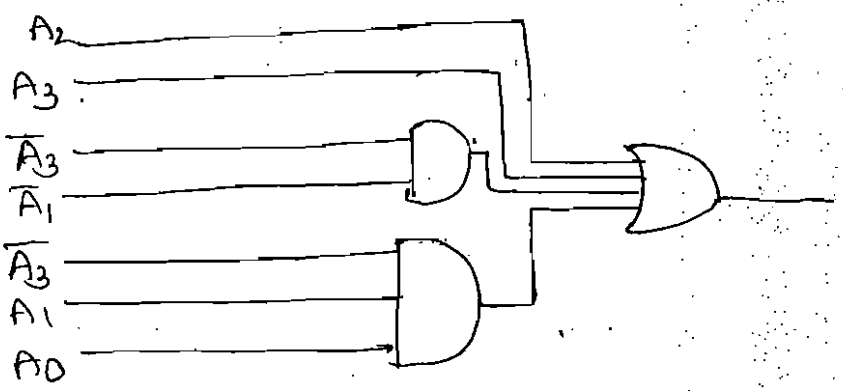
K-map for G1.

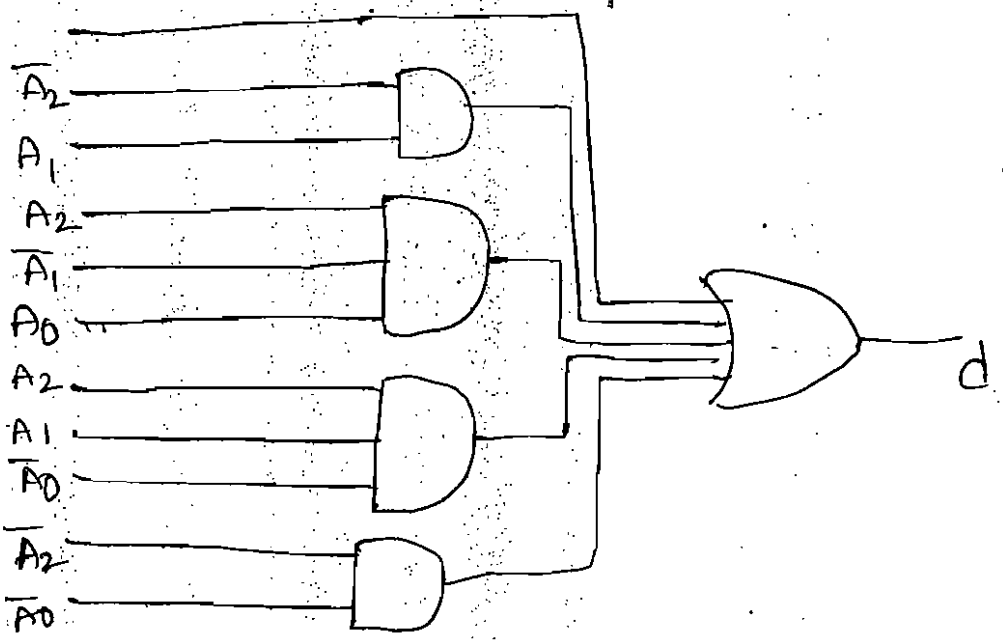
$f = \bar{A}_1\bar{A}_0 + A_3 + \bar{A}_3A_2\bar{A}_1 + A_2A_1\bar{A}_0$

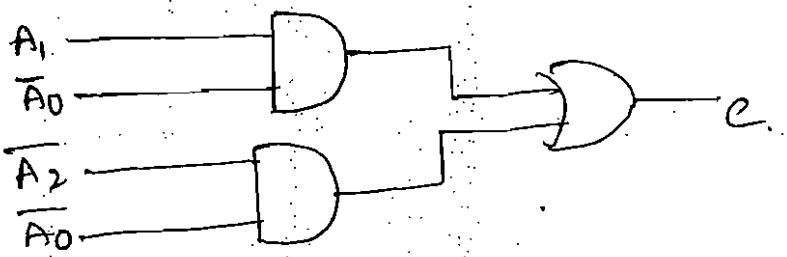
$G_1 = A_3 + A_1\bar{A}_0 + A_2\bar{A}_1 + \bar{A}_2A_1$

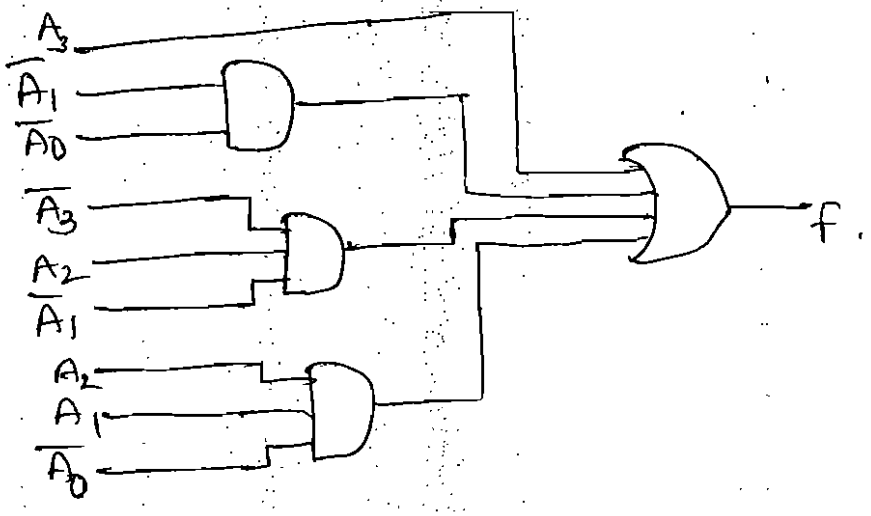


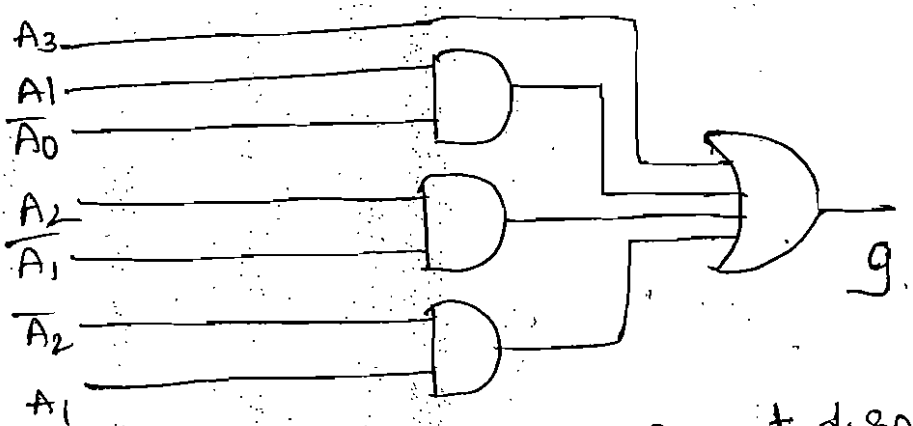








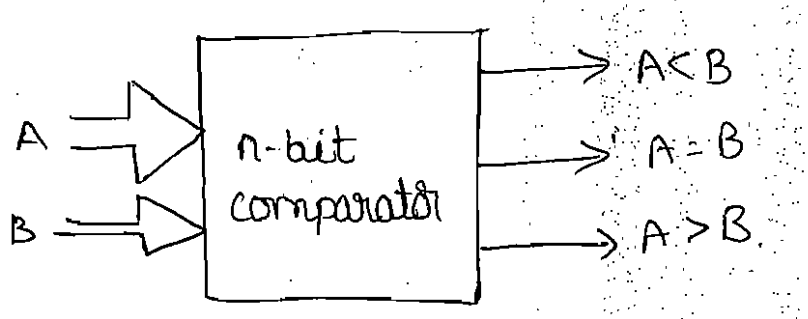




diagrams for seven segment display.

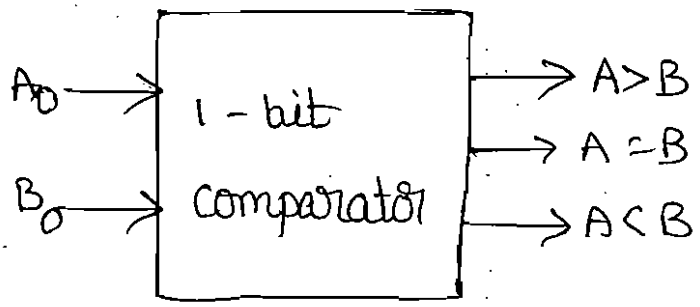
Comparator :-

The comparator is a combinational logic circuit It compares the magnitude of two n-bit numbers and provides the relative result as the output The block diagram of an n-bit digital comparator has 2 inputs and three outputs. A and B are the n-bit inputs. The comparator outputs are A>B, A=B and A<B. Depending upon the result of comparsion, one of these outputs will be high.



1-bit Comparator :-

The one bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely A<B, A=B, A>B.



Block diagram.

Inputs		outputs		
A_0	B_0	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

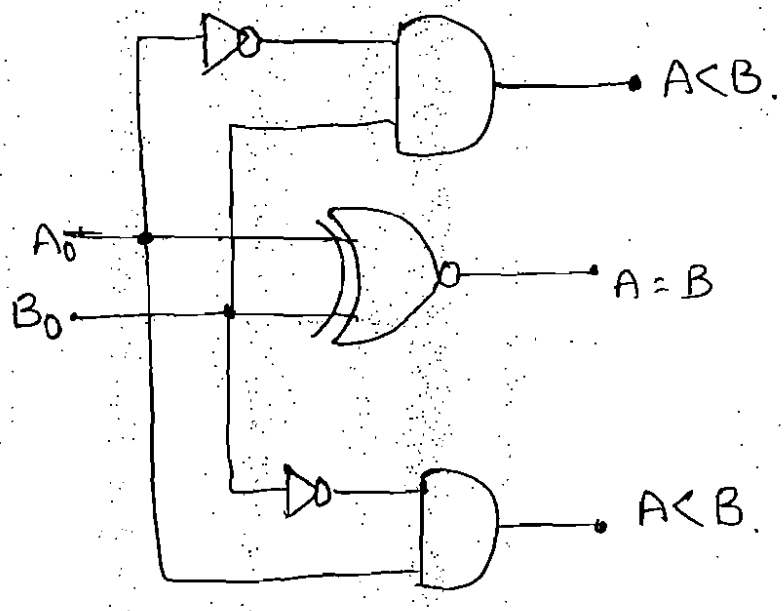
Truth table

In truth table

$A=B : \overline{A_0}\,\overline{B_0} + A_0 B_0 = A_0 \odot B_0$

$A<B : \overline{A_0} B_0$

$A>B : A_0 \overline{B_0}$



2-bit comparator :-

The logic for a 2-bit comparator. let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

→ if $A_1 = 1$ and $B_1 = 0$, then $A > B$ or

→ if A_1 and B_1 are equal and $A_0 = 1$ and $B_0 = 0$ then $A > B$.

$A>B : A_1 \overline{B_1} + (A_1 \odot B_1) A_0 \overline{B_0}$

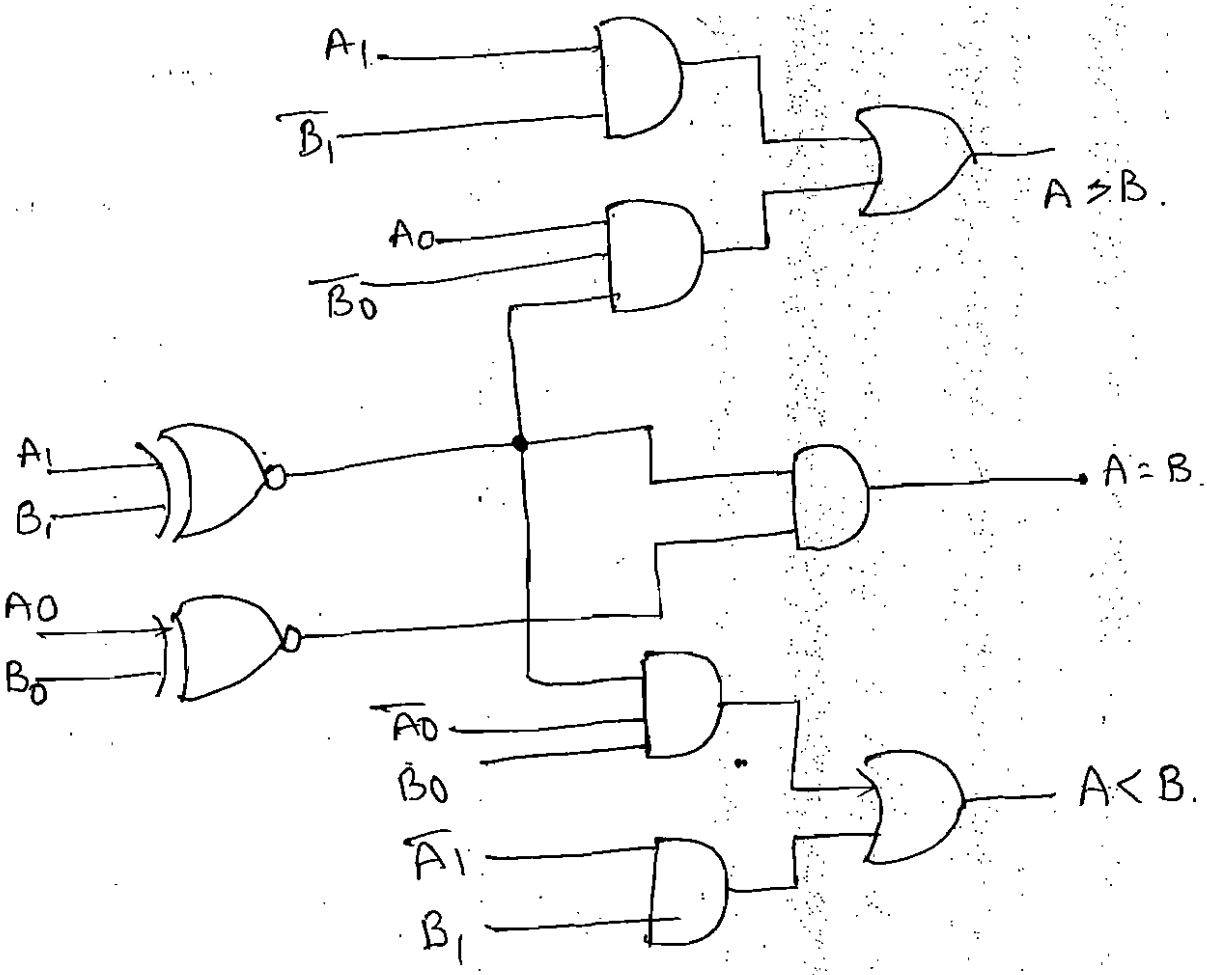
→ if $B_1 = 1$ and $A_1 = 0$ then $A < B$ or

→ if B_1 and A_1 are equal and $B_0 = 1$ and $A_0 = 0$ then $A < B$

$A<B : \overline{A_1} B_1 + (A_1 \odot B_1) \overline{A_0} B_0$

→ if A_1 and B_1 are equal and if A_0 and B_0 are equal then $A = B$

$A = B : (A_1 \odot B_1)(A_0 \odot B_0)$



4-bit comparator :-

The logic for a 4-bit comparator. let the four bit numbers will be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

→ if $A_3 = 1$ and $B_3 = 0$, then $A > B$ or

→ if A_3 and B_3 are equal, and if $A_2 = 1$ and $B_2 = 0$, or

→ if A_3 and B_3 are equal, A_2 and B_2 are equal, and if A_1 = 1 and $B_1 = 0$, or

→ if A_3 and B_3 are equal, and if A_2 and B_2 are equal, and if A_1 and B_1 are equal, and if $A_0 = 1$ and $B_0 = 0$.

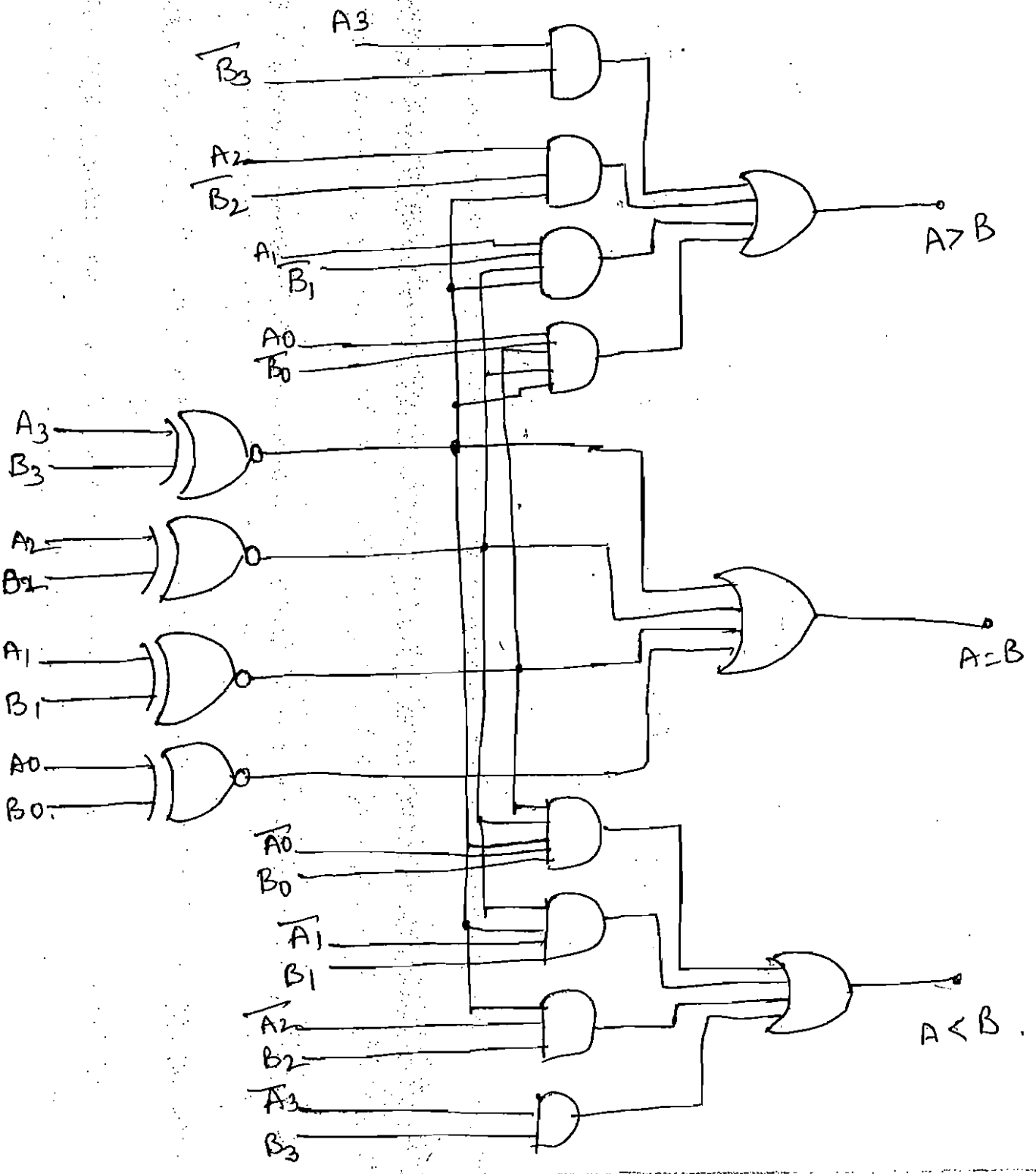
$(A>B) = A_3\overline{B_3} + (A_3 \odot B_3) A_2\overline{B_2} + (A_3 \odot B_3)(A_2 \odot B_2) A_1\overline{B_1}$

$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1) A_0\overline{B_0}$

$(A<B) = \overline{A_3} B_3 + (A_3 \odot B_3)\overline{A_2} B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\overline{A_1} B_1$

$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\overline{A_0} B_0$

$A = B : (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0).$



Logic diagram of 4-bit comparator.

priority Encoders :-

It is possible that two or more inputs are active at a time. To overcome this, priority encoders are used. A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously high. The most common priority system is based on the relative magnitudes of the inputs.

In some practical applications, priority encoders may have several inputs that are routinely high at the same time, and the principal function of the encoder in those cases is to select the input with the highest priority.

4-input priority Encoder :-

In 4-input priority Encoder in addition to the outputs A and B, the circuit has a third output designed by V. This is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't care conditions.

Truth table

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

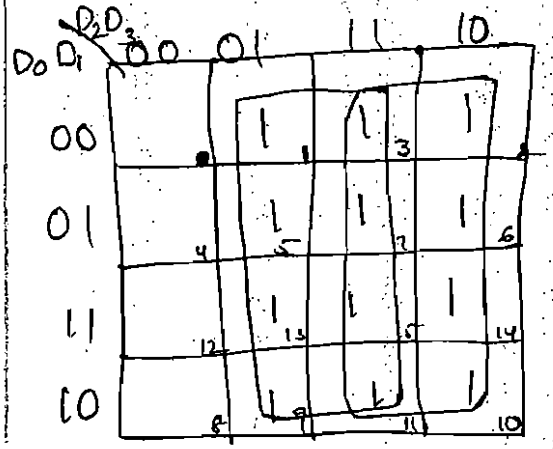
$V = D_0 + D_1 + D_2 + D_3$.

According to the truth table. the outputs A and B are.

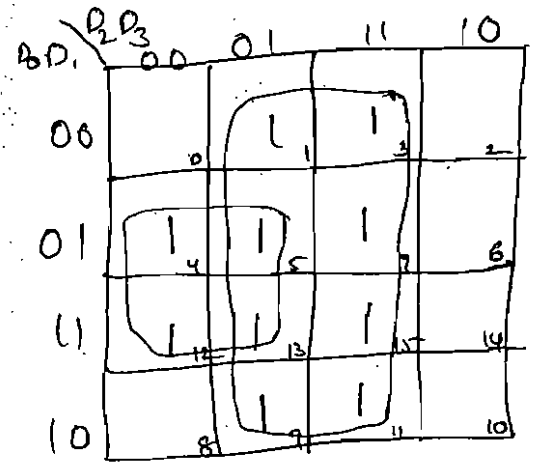
$$A = \Sigma m(1,2,3,5,6,7,9,10,11,13,14,15)$$

$$B = \Sigma m(1,3,4,5,7,9,11,12,13,15)$$

K-map for A

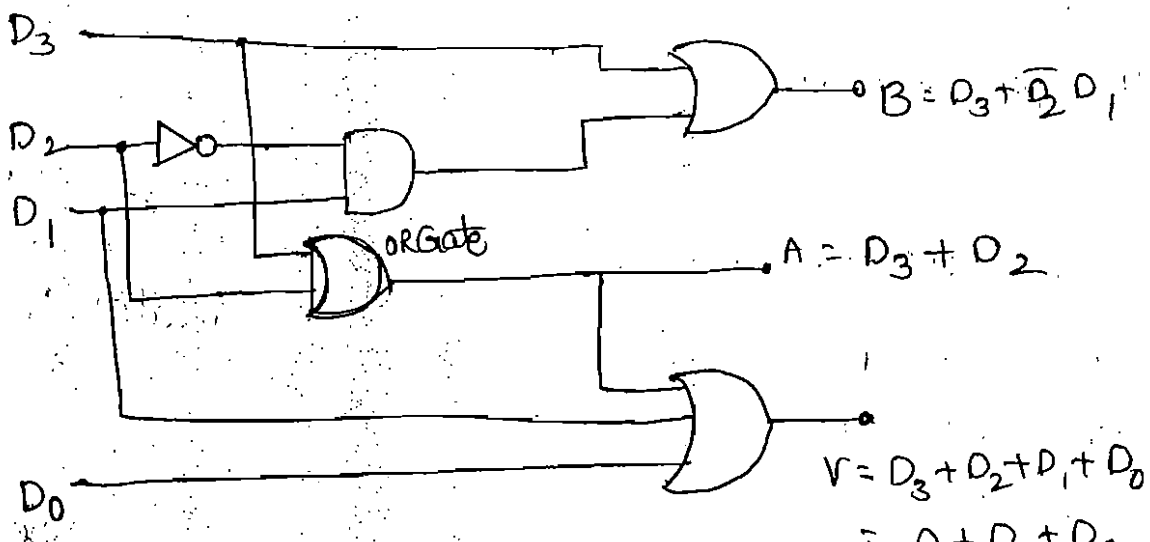


K-map for B.



$A = D_3 + D_2$

$B = D_3 + \overline{D_2} D_1$



Logic diagram for

4-bit priority Encoder.

PROGRAMMABLE LOGIC DEVICES

→ Logic designers have a wide range of standard IC's available to them with numerous logic functions and logic circuit arrangements on a chip. In addition, these ICs are available from many manufactures and at a reasonably low cost.

→ PLD is an IC that contains large number of gates, flip-flops and registers that are intercomected on chip. This IC is said to be programmable because the specific function IC is determined by Intercomecting required contacts.

Basically, there are three types of programmable device which are.

→ Read only memory. (ROM)

→ programmable logic array (PLA)

→ programmiable Array logic (PAL)

READ ONLY memory :-

→ The read only memory is a type of semiconductor memory that is designed to hold data that is either permanent or will not change frequently.

→ During operation, no new data can be written into a ROM, but data can be read from ROM. the process of Entering data is called programming or burung-in the ROM.

→ Some ROM's cannot have their data changes once they have been programmed others can be erased and reprogrammed as often as desired.

Types of ROM's :-

1. Masked memory ROM
2. programmable Read only memory (PROM)
3. Erasable programmable Read only memory (EPROM)
4. Electrically Erasable programmable Read only memory (EEPROM)

Masked memory (Rom) :-

→ cannot be reprogrammed
→ Nonvolatile, retain data even when power is turn off.
→ cheaper than programmable devices.
→ useful for fixed programme instructions.

PROM

→ programmed by blowing built-in fuses.
→ can not be reprogrammed
→ Non volatile.
→ useful for small volume data storing
→ user programmable

EPROM :-

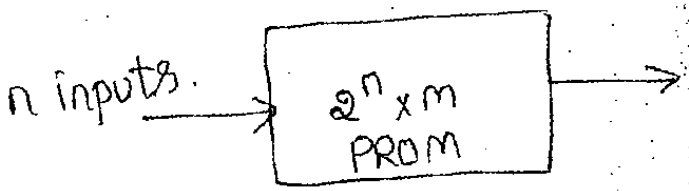
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.
→ NON volatile

Programmable ROM :-

→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination that comes out of the output lines is called a word.



Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array provide an AND-OR Sum of products implementati

EPROM :-

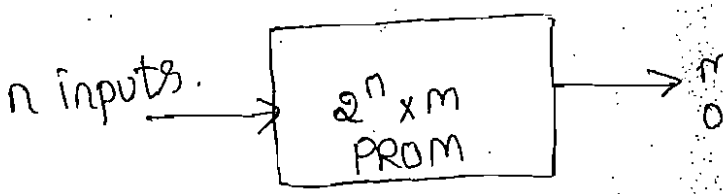
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.
→ NON volatile

Programmable ROM :-

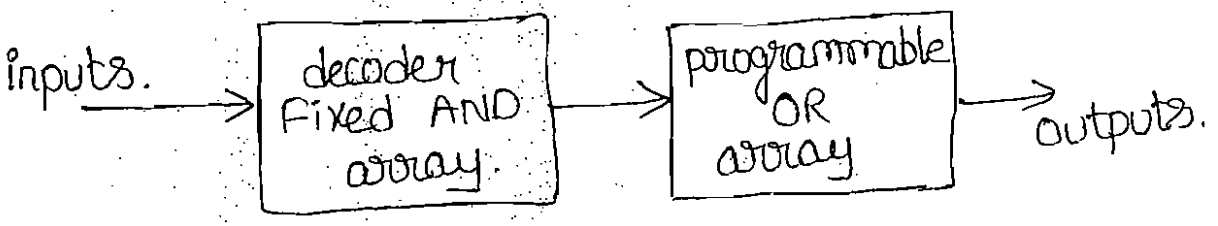
→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination that comes out of the output lines is called a word.

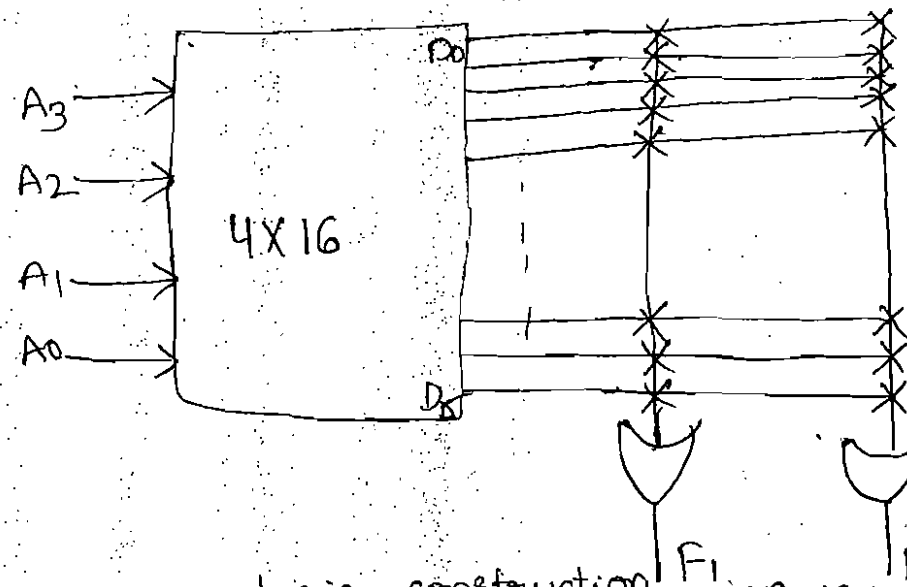


Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR Sum of products implementation.

The programmable read-only memory (PROM) has a fixed AND array constructed as a decoder and a programmable OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR Gate.





Logic construction using 16X2 ROM.

→ Implement full-adder using PROM.

The number of inputs variables of a full adder are 3. The possible number of combinations are 8. So we need 3X8 decoder. The number of outputs of full adder are 2. They are Sum and Carry.

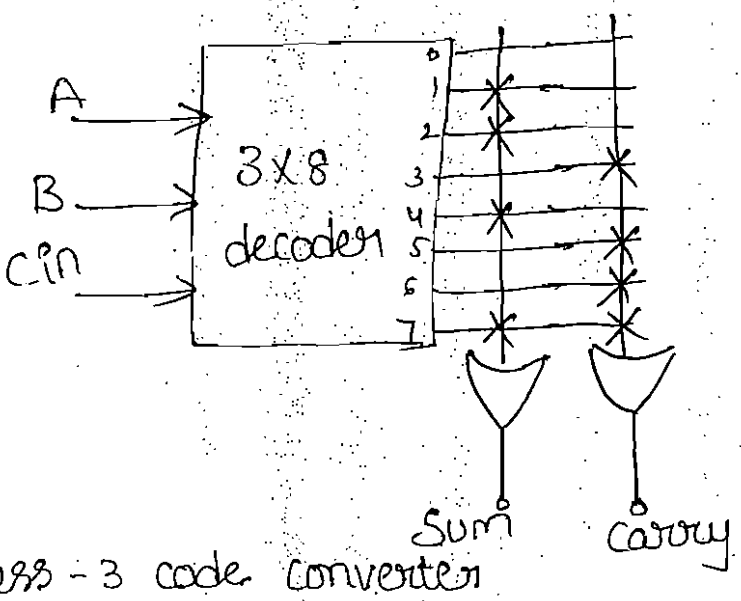
Truth table

A	B	Cin	S	C
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

output expression for Sum & carry is Sum = $\Sigma m(1,2,4,7)$

carry = $\Sigma m(3,5,6,7)$.

Logic diagram



→ Design BCD to Excess-3 code converter using PROM.

Step:- 1 - Truth table.

BCD				EXCESS-3			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1

Step 2 :-

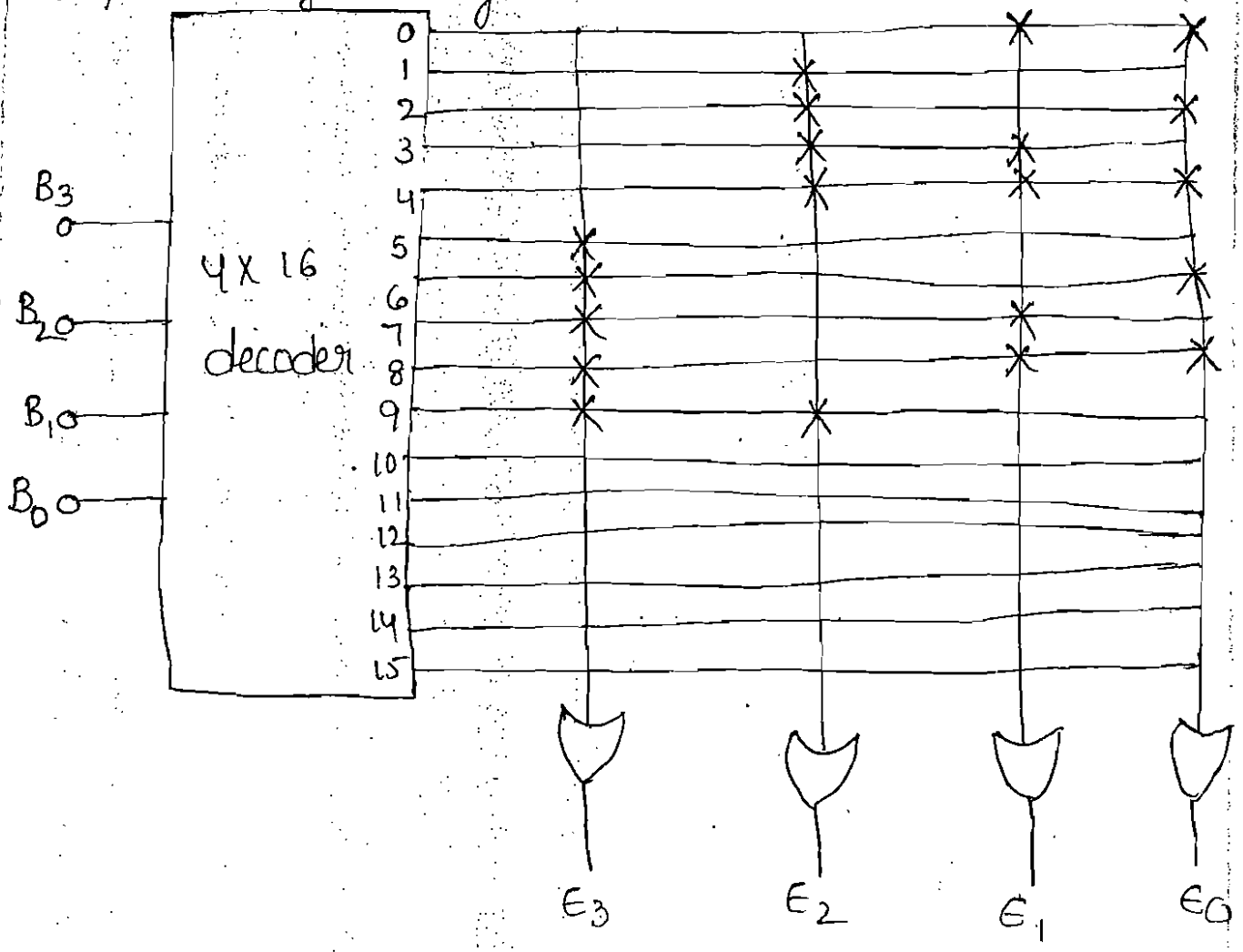
$E_0(B_3\ B_2\ B_1\ B_0) = \Sigma m(0,2,4,6,8)$

$E_1(B_3\ B_2\ B_1\ B_0) = \Sigma m(0,3,4,7,8)$

$E_2(B_3\ B_2\ B_1\ B_0) = \Sigma m(1,2,3,4,9)$

$E_3(B_3\ B_2\ B_1\ B_0) = \Sigma m(5,6,7,8,9)$

Step 3 :- logic diagram.



→ Implement the following Boolean expression using PROM.

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

first given expression converted into a standard sop form.

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

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

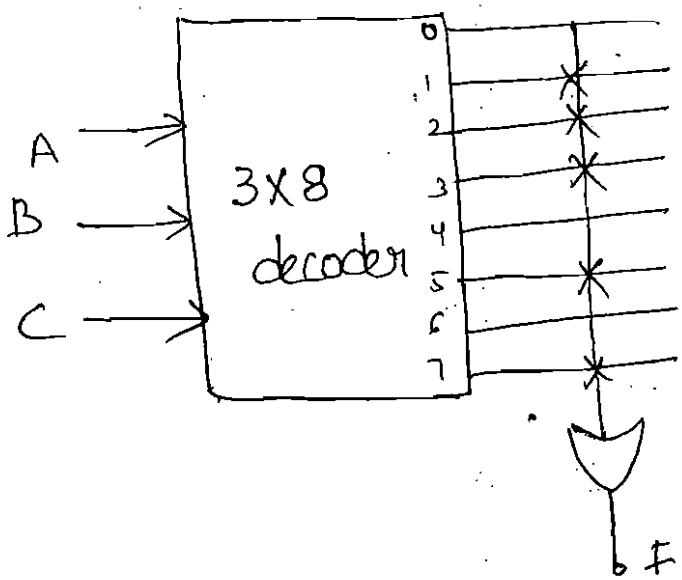
$$= \underset{1}{\bar{A}BC} + \bar{A}B\bar{C} + \underset{2}{ABC} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \underset{2}{AB\bar{C}} + \underset{1}{\bar{A}BC}$$

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

$011, 010, 111, 101, 011, 001$

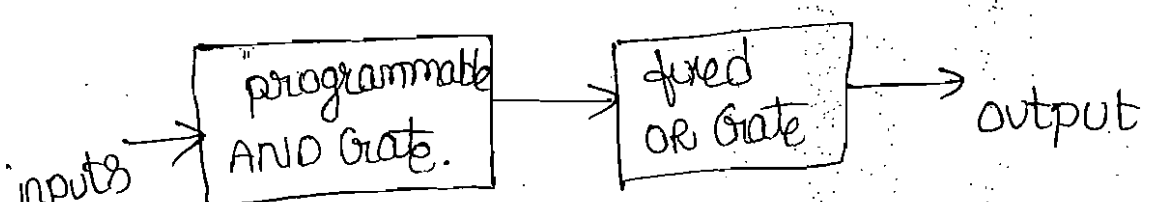
$f(A,B,C) = \Sigma m(1,2,3,5,7)$

Logic diagram :-



PAL :- programmable Array logic :-

The programmable Array logic is a programmable device with a fixed OR array and a programmable AND array because, only the AND gates are programmble.

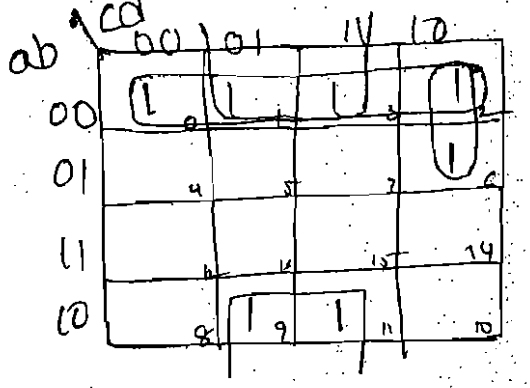


→ Implement the following functions using PAL.

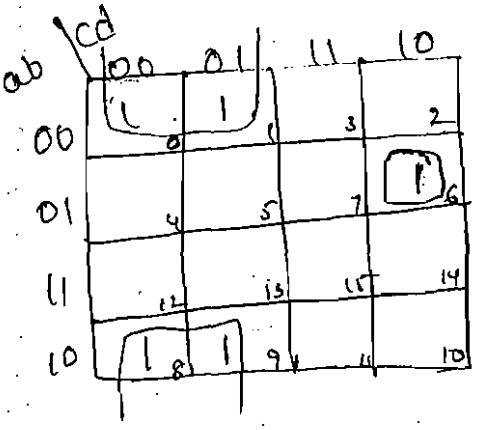
$F_1(a,b,c,d) = \Sigma m(0,1,2,3,6,9,11)$

$F_2(a,b,c,d) = \Sigma m(0,1,6,8,9)$

Step 1:- K-map simplification for F_1 and F_2



$\bar{a}\bar{b} + \bar{a}c\bar{d} + \bar{b}d$

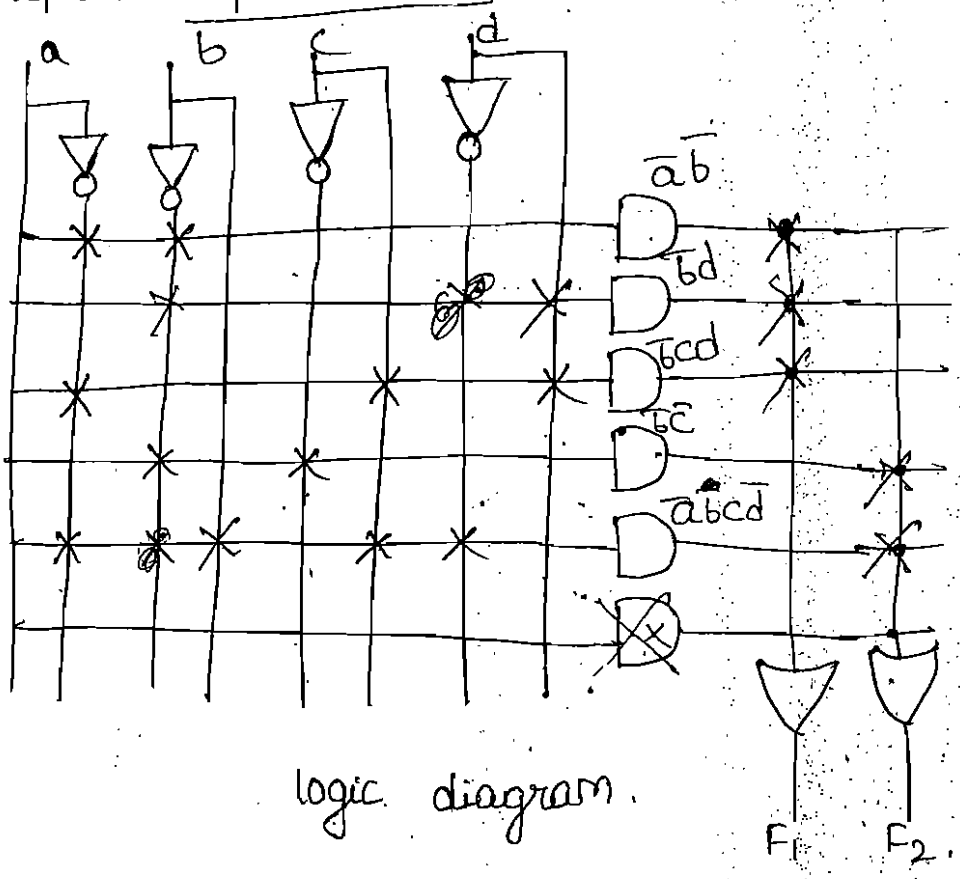


$\bar{b}\bar{c} + \bar{a}bc\bar{d}$

Step 2 :- PAL programming table.

product terms	AND gate inputs a	b	c	d	outputs
1	0	0	–	–	$F_1 = \bar{a}\bar{b} + \bar{a}cd + \bar{b}d$
2	0	–	1	1	
3	–	0	–	1	
4	–	0	0	–	$F_2 = \bar{b}\bar{c} + \bar{a}bc\bar{d}$
5	0	1	1	0	
6	–	–	–	–	

Step 3:- implementation:-



logic diagram.

→ Implement 4-bit BCD to XS-3 code conversion using PAL.

Step:-1

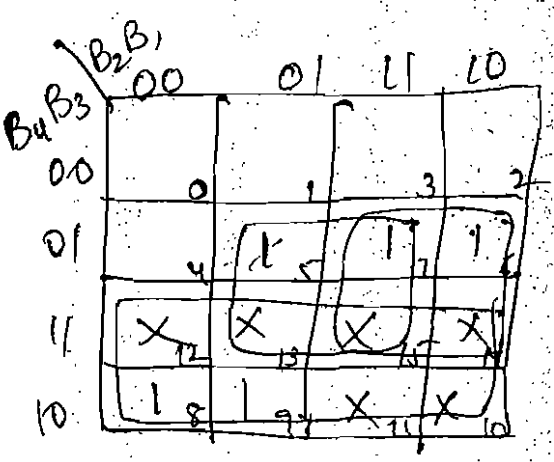
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

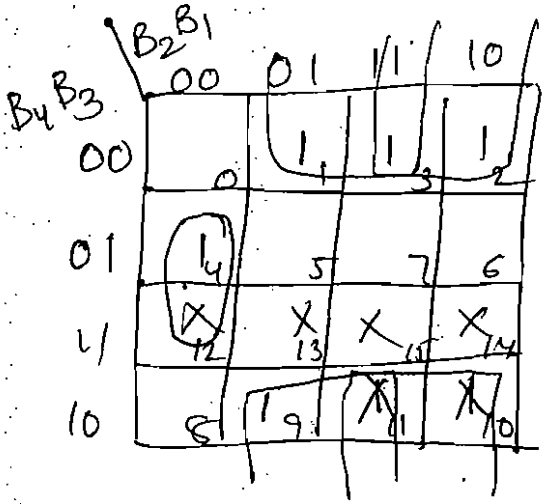
$X_4 = \Sigma m(5,6,7,8,9) + d(10,11,12,13,14,15)$

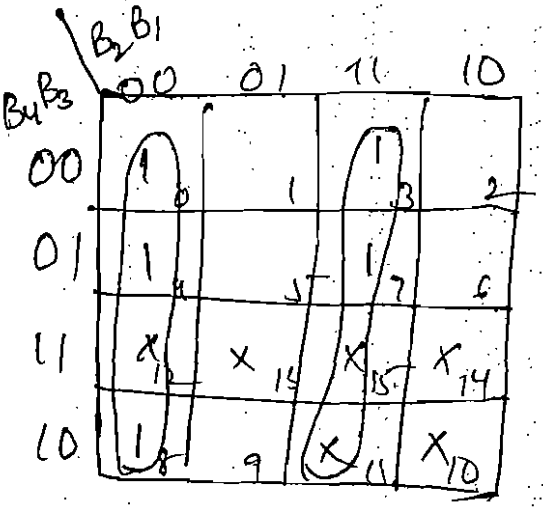
$X_3 = \Sigma m(1,2,3,4,9) + d(10,11,12,13,14,15)$

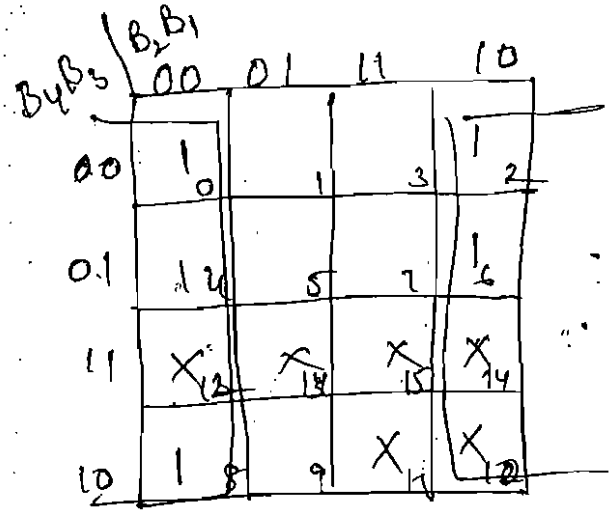
$X_2 = \Sigma m(0,3,4,7,8) + d(10,11,12,13,14,15)$

$X_1 = \Sigma m(0,2,4,6,8) + d(10,11,12,13,14,15)$









$X_4 = B_4 + B_3B_2 + B_3B_1$

$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$

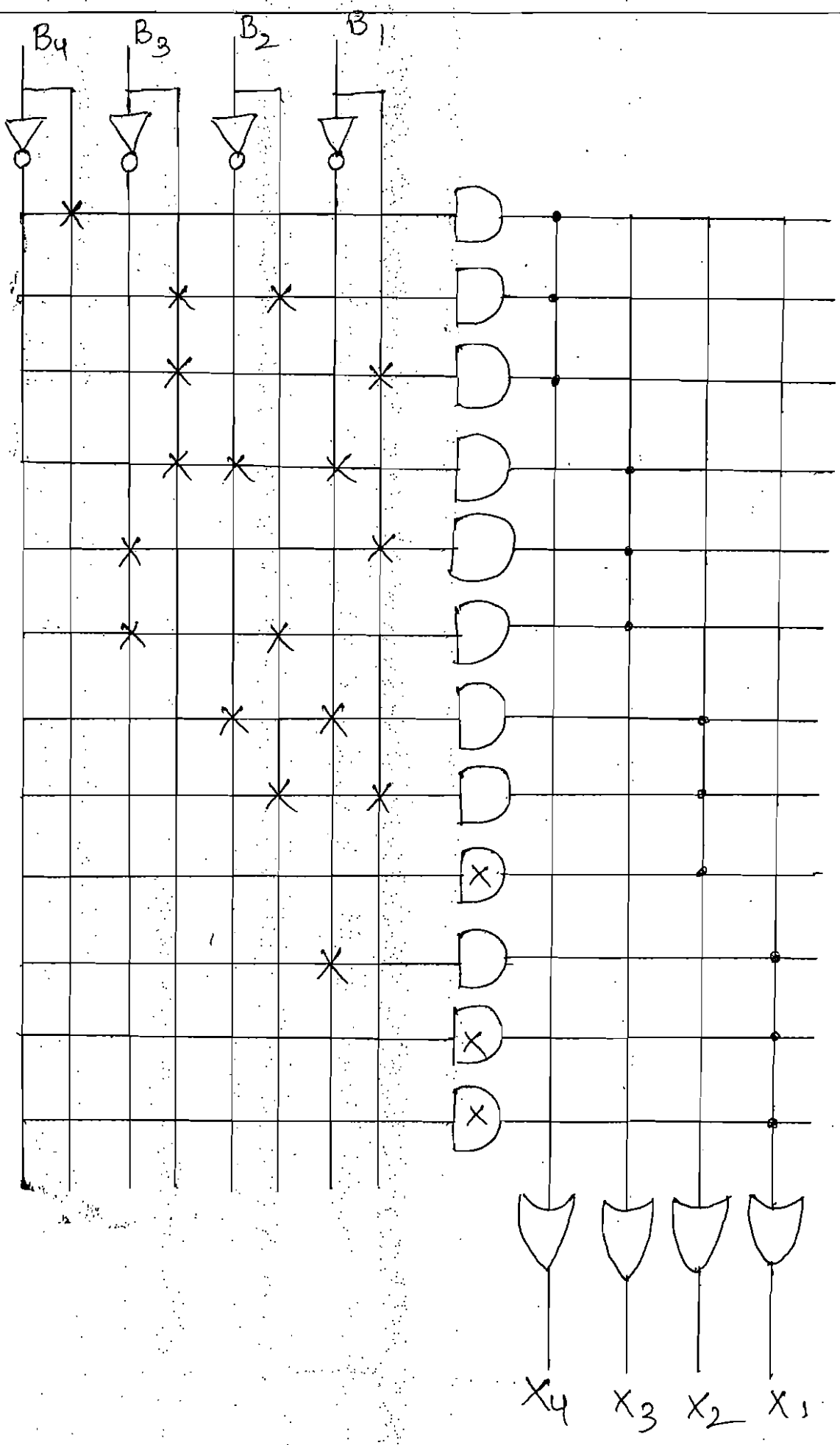
$X_2 = \overline{B_2\overline{B_1} + B_2B_1}$

$X_1 = \overline{B_1}$

Step 2 :- programming table.

product terms.	AND Gate inputs. B_4	B_3	B_2	B_1	Outputs.
1	1	—	—	—	$X_4 = B_4 + B_3B_2 + B_3B_1$
2	—	1	1	—	
3	—	1	—	1	
4	—	1	0	0	$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$
5	—	0	—	1	
6	—	0	1	—	
7	—	—	0	0	$X_2 = \overline{B_2}\,\overline{B_1} + B_2B_1$
8	—	—	1	1	
9	—	—	—	—	
10	—	—	—	0	$X_1 = \overline{B_1}$
11	—	—	—	—	
12	—	—	—	—	

Step 3 :- logic diagram.



logic diagram.

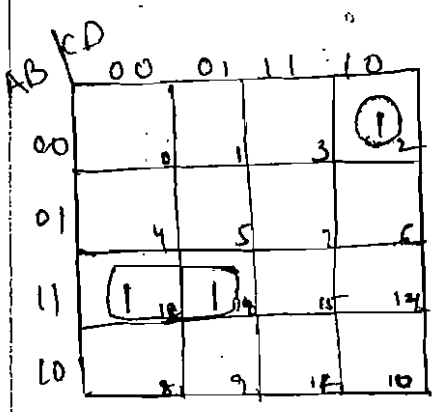
Implement the following boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

$F_1(A,B,C,D) = \Sigma m(2,12,13)$

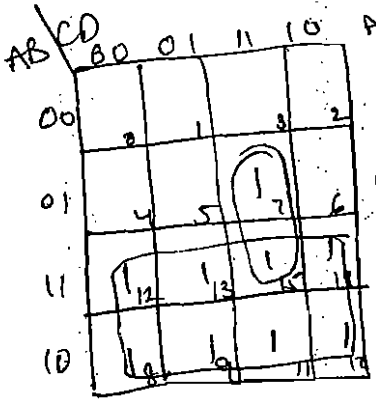
$F_2(A,B,C,D) = \Sigma m(7,8,9,10,11,12,13,14,15)$

$F_3(A,B,C,D) = \Sigma m(0,2,3,4,5,6,7,8,10,11,15)$

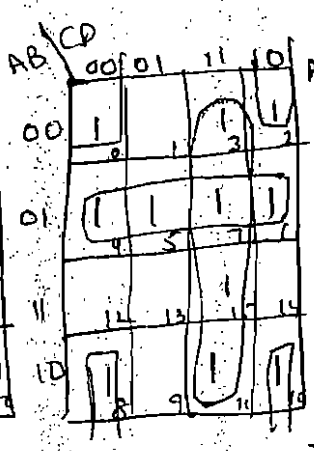
$F_4(A,B,C,D) = \Sigma m(1,2,8,12,13)$



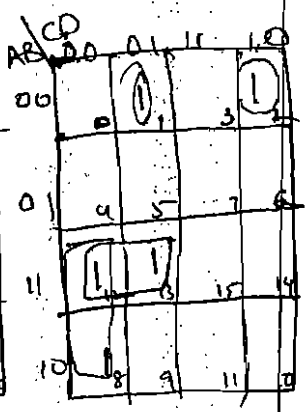
$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$



$F_2 = A + BCD$



$F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$



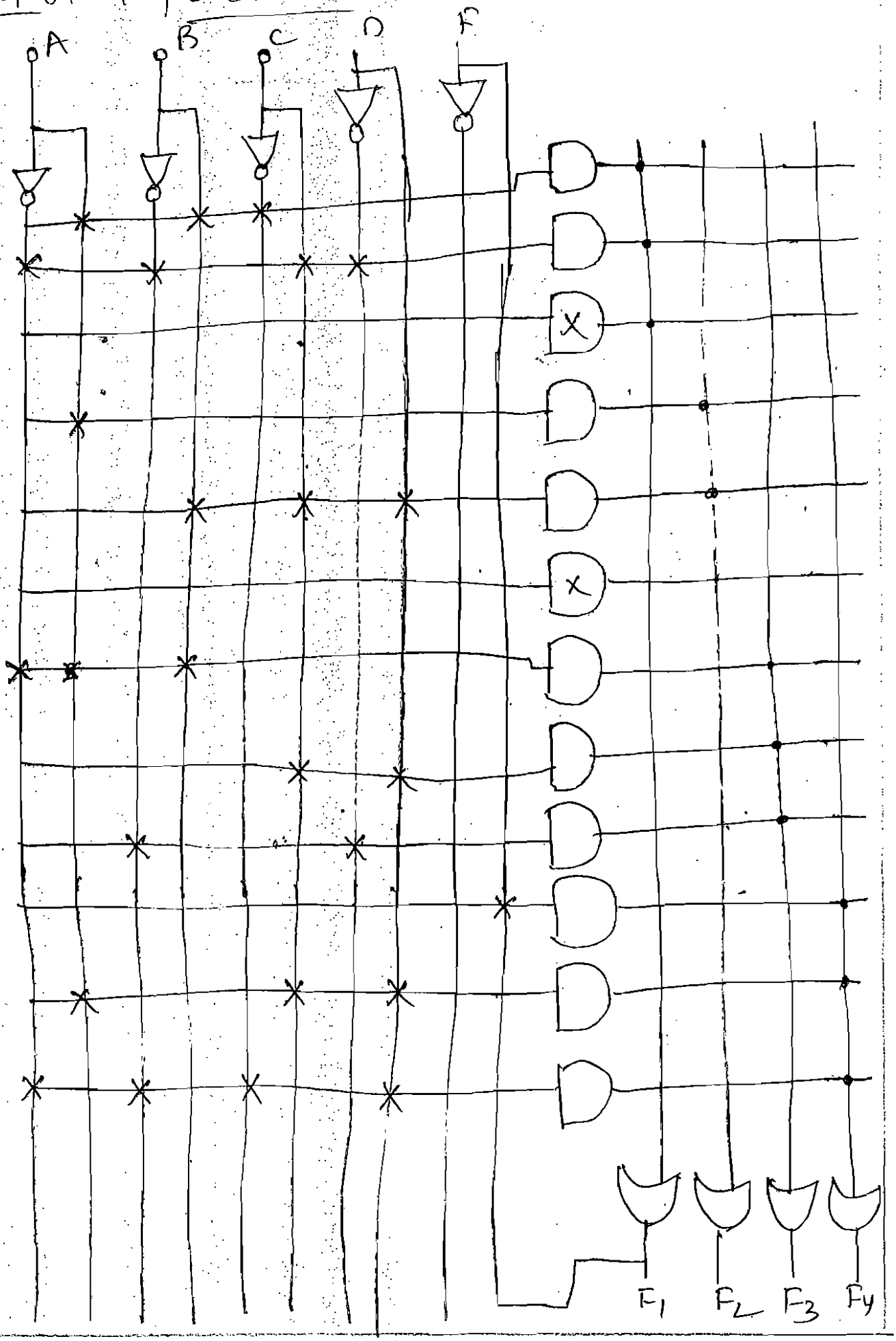
$F_4 = AB\bar{C} + A\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}\bar{B}C\bar{D}$

$= F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$

Step 2:- programming table

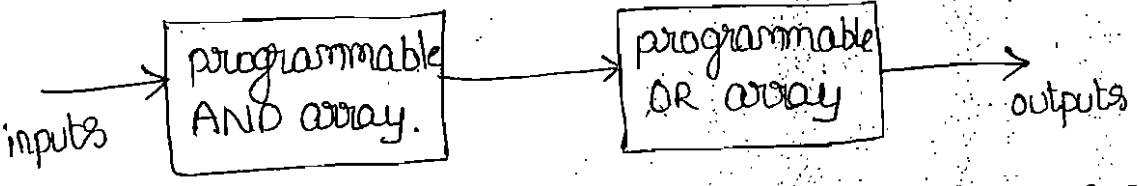
product term	AND inputs A B C D	F_1	outputs
1	1 1 0 –		$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$
2	0 0 1 0		
3	– – – –		– – –
4	1 – – –		$F_2 = A + BCD$
5	– 1 1 1		
6	– – – –		– – –
7	0 1 – –		$F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$
8	– – 1 1		
9	– 0 – 0		– – –
10	– – – –	1	
11	1 – 0 0		$F_4 = F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$
12	0 0 0 1		

Step 3 :- Implementation :-



PLA :- programmable logic Array

In programmable logic array where both the AND and OR arrays can be programmed. The product terms in the AND array may be shared by any OR gate to provide the required sum of products.

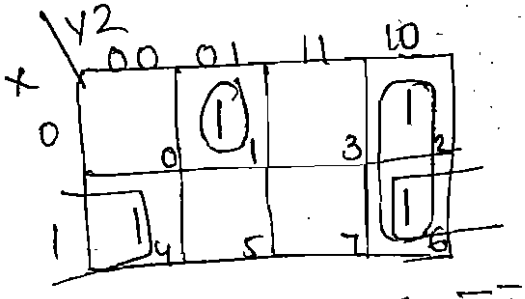


→ Implement the given boolean functions by using PLA.

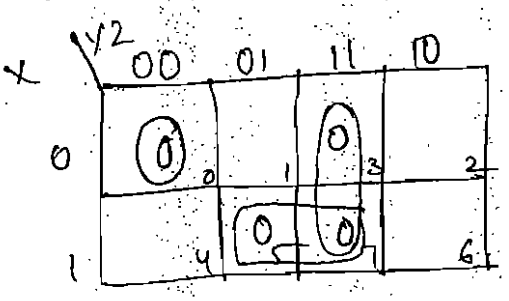
$A(X,Y,Z) = \Sigma m(1,2,4,6)$

$B(X,Y,Z) = \Sigma m(1,2,3,5,7)$

Step 1 :- The K-maps for the functions A, B, their minimization, and the minimal expressions for both the true and complement of those in sum of products.



$A(T) = X\bar{Z} + Y\bar{Z} + \bar{X}\bar{Y}Z$



$\bar{A} = XZ + YZ + \bar{X}\bar{Y}\bar{Z}$

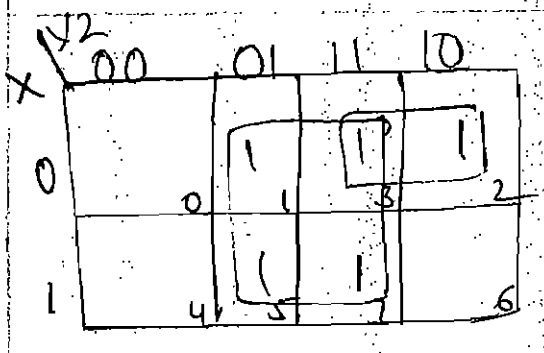
$A(C) = \overline{XZ + YZ + \bar{X}\bar{Y}\bar{Z}}$

K-map for A.

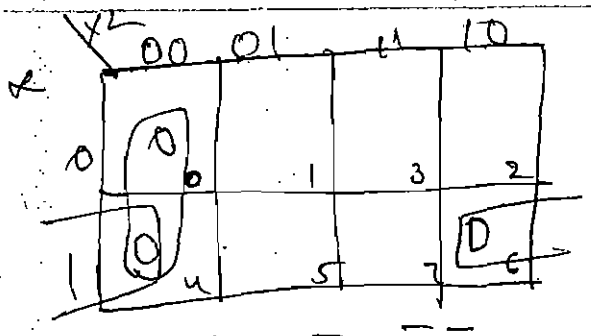
Simply $A(C) = \overline{(\bar{X}+\bar{Z})\cdot(\bar{Y}+\bar{Z})\cdot(X+Y+Z)}$

$= \overline{(\bar{X}+\bar{Z})} + \overline{(\bar{Y}+\bar{Z})} + \overline{(X+Y+Z)}$

$= XZ + YZ + \bar{X}\bar{Y}\bar{Z}$



$B(T) = \bar{X}\bar{Y} + Z$.



$\bar{B} = X\bar{Z} + \bar{Y}\bar{Z}$

$B(C) = \overline{X\bar{Z} + \bar{Y}\bar{Z}}$

Simply $B(C) = \overline{(\bar{Y}+Z)(\bar{X}+Z)}$

$= \overline{(\bar{Y}+Z)} + \overline{(\bar{X}+Z)}$

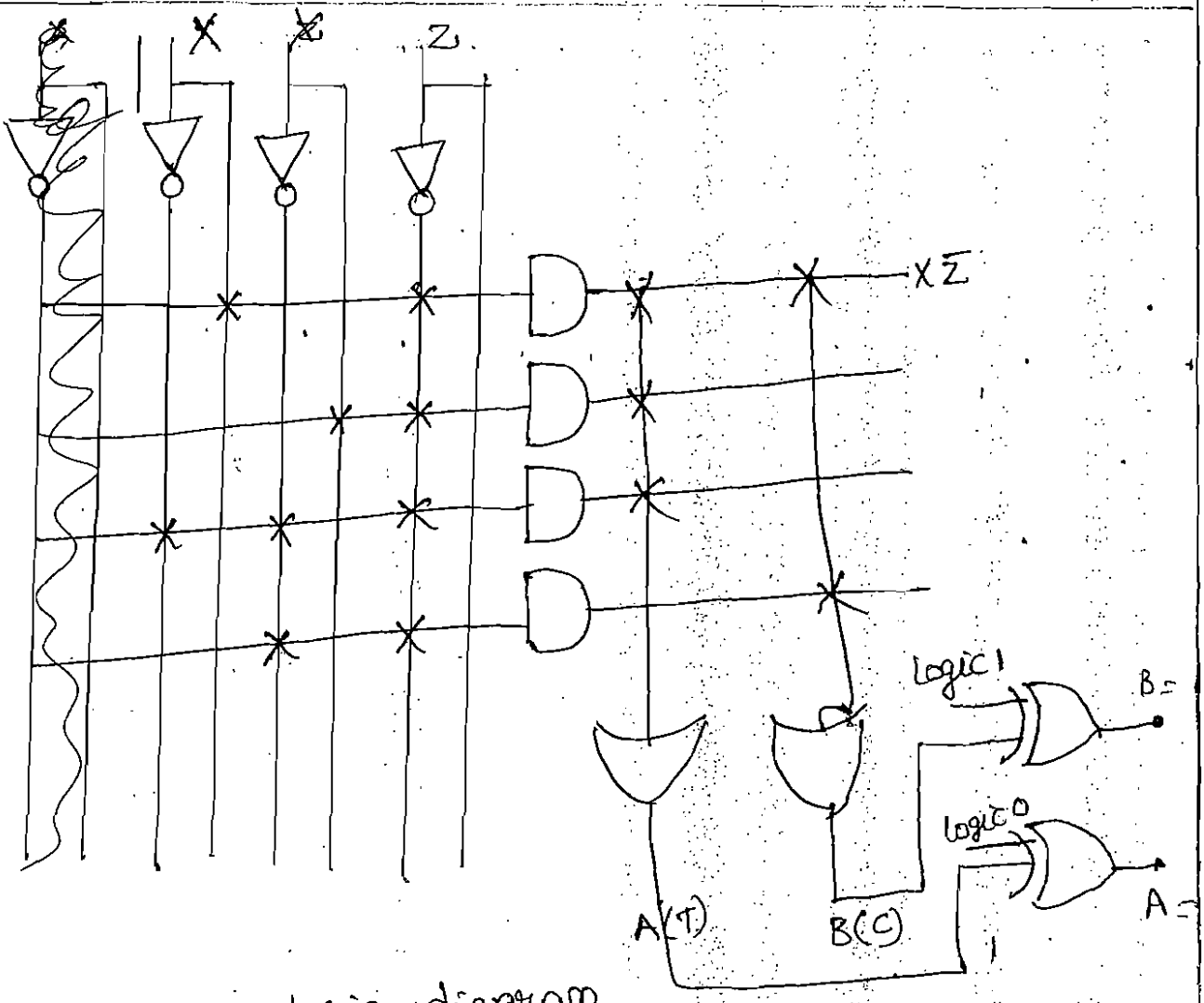
$= \bar{Y}\bar{Z} + X\bar{Z}$

$A(T) = X\bar{Z} + Y\bar{Z} + \bar{X}\bar{Y}Z$, $B(T) = \bar{X}Y + Z$

$A(C) = XZ + YZ + \overline{XYZ}$ $B(C) = \bar{Y}\bar{Z} + X\bar{Z}$

Step 2 :- programming table.

product term	inputs X	Y	Z	outputs A(T)	A(C)	B(T)	B(C)
$X\bar{Z}$	1	–	0	1	–	–	1
$Y\bar{Z}$	–	1	0	1	–	–	–
$\bar{X}\bar{Y}Z$	0	0	1	1	–	–	–
XZ	1	–	1	–	1	–	–
YZ	–	1	1	–	1	–	–
$\overline{XYZ}$	0	0	0	–	1	–	–
$\bar{X}Y$	0	1	–	–	–	1	–
Z	–	–	1	–	–	1	1
$\bar{Y}\bar{Z}$	–	0	0	–	–	–	1
$X\bar{Z}$	1	–	0	–	–	–	1



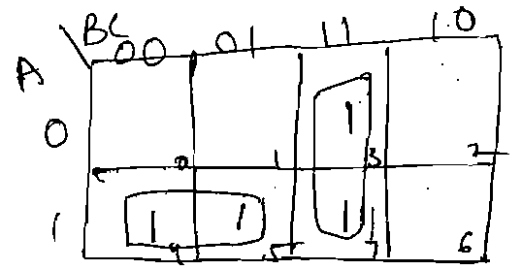
Logic diagram.

→ Implement the following boolean functions F_1 & F_2 of a combinational logic circuit using PLA.

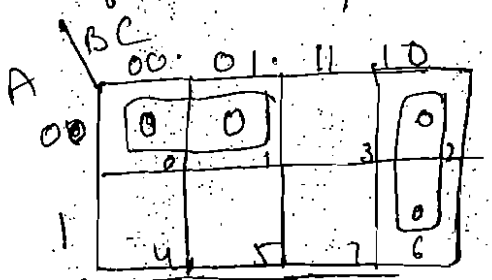
$F_1(A,B,C) = \Sigma m(3,4,5,7)$

$F_2(A,B,C) = \Sigma m(1,4,6)$.

Step 1:- K-map for F_1 (True form) Complement form



$F_1 = A\bar{B} + BC$



$\bar{F_1} = (A+B)(\bar{B}+C)$

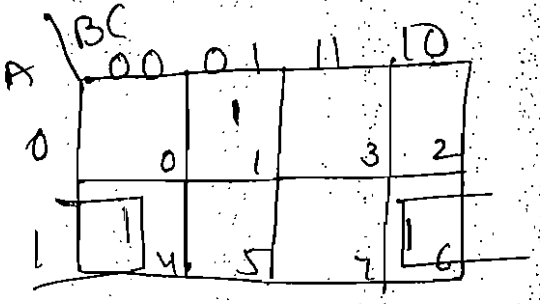
$= \overline{(A+B)} + \overline{(\bar{B}+C)}$

$F_1(T) = A\bar{B} + BC$

$F_1(C) = (\bar{A}.B) + \bar{B}.\bar{C}$

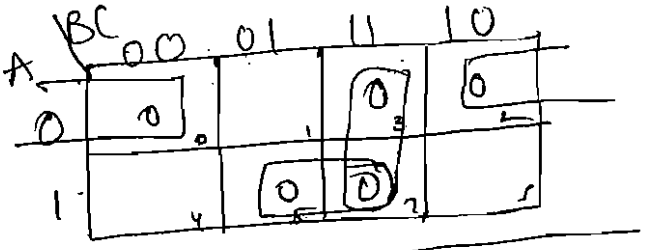
$= \bar{A}.\bar{B} + B.\bar{C}$

K-map for F_2 (true form)



$F_2 = \bar{A}\bar{B}C + A\bar{C}$

K-map for F_2 (complement form)



$\bar{F}_2 = (\bar{A}+C).(\bar{B}+\bar{C})(\bar{A}+\bar{C})$

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

$= \bar{A}.\bar{C} + BC + AC.$

$F_1(T) = A\bar{B} + BC$

$F_2(T) = \bar{A}\bar{B}C + A\bar{C}$

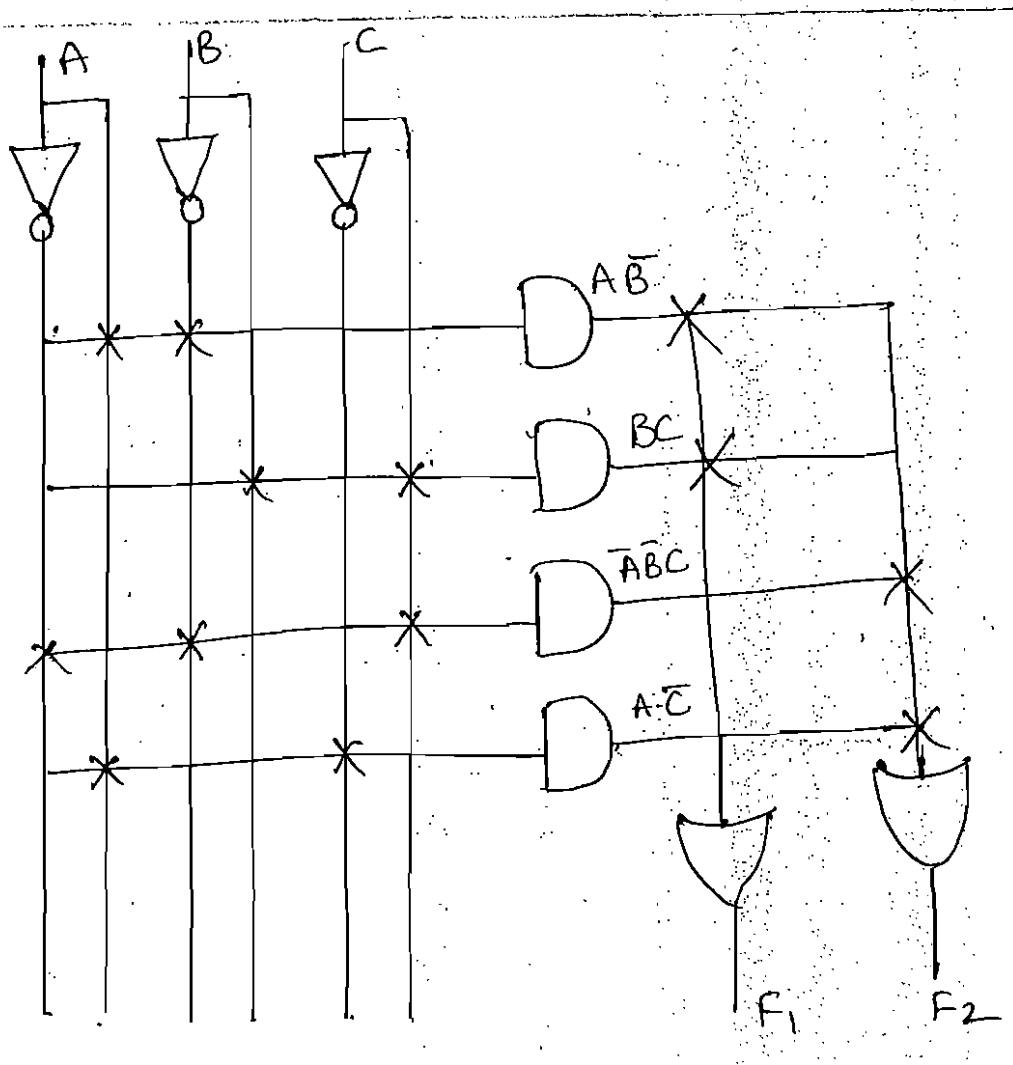
$F_1(C) = \bar{A}\bar{B} + B\bar{C}$

$F_2(C) = \bar{A}\bar{C} + BC + AC$

When we take $F_1(T)$ & $F_2(T)$ get 4 product terms and also having same number of product terms while taking $F_1(T)$ & $F_2(C)$, $F_1(C)$ & $F_2(T)$. So we have to consider $F_1(T)$ & $F_2(T)$.

Step 2 :- programming table

product terms	inputs A	B	C	outputs $F_1(T)$	$F_2(C)$
$A\bar{B}$	1	0	—	1	—
BC	—	1	1	1	1
$\bar{A}\bar{B}\bar{C}$	0	—	0	1	1
$A\bar{C}$	1	—	1	—	1



logic diagram.

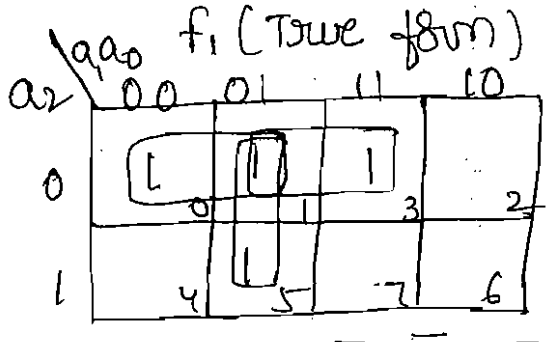
→ Implement the following multi boolean function using (3X4X2) PLA.

$F_1(a_2, a_1, a_0) = \Sigma m(0,1,3,5)$

$F_2(a_2, a_1, a_0) = \Sigma m(3,5,7)$

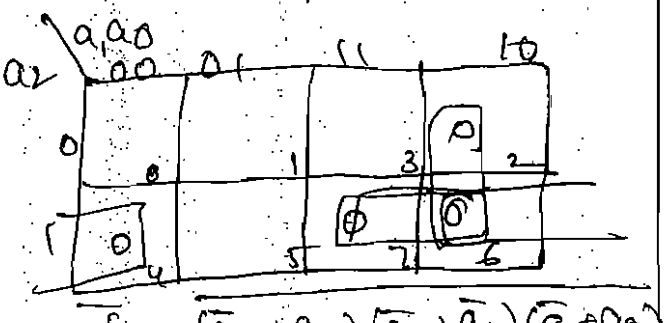
step 1:- K-maps.

f_1 (True form)



$f_1(T) = \bar{a}_1a_0 + \bar{a}_2\bar{a}_1 + \bar{a}_2a_0$

f_1 (complement form)

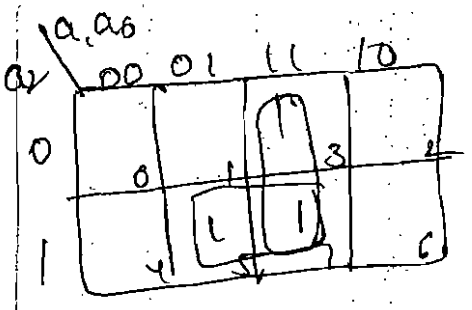


$\bar{f}_1 = (\bar{a}_2 + a_0)(\bar{a}_2 + \bar{a}_1)(\bar{a}_1 + a_0)$

$f_1(C) = \overline{\overline{a}_2} \cdot a_0 + \overline{\overline{a}_2} \cdot \overline{\overline{a}_1} + \overline{\overline{a}_1} \cdot \overline{a}_0$

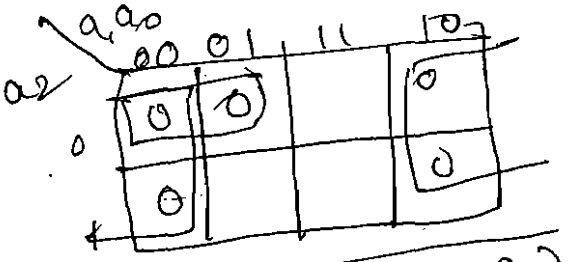
$= a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

f_2(True form)



$f_2(T) = a_2 a_0 + a_1 a_0$

F_2 (complement form..



$\overline{F}_2 = (\overline{a}_0) \cdot (a_2 + a_1)$

$F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

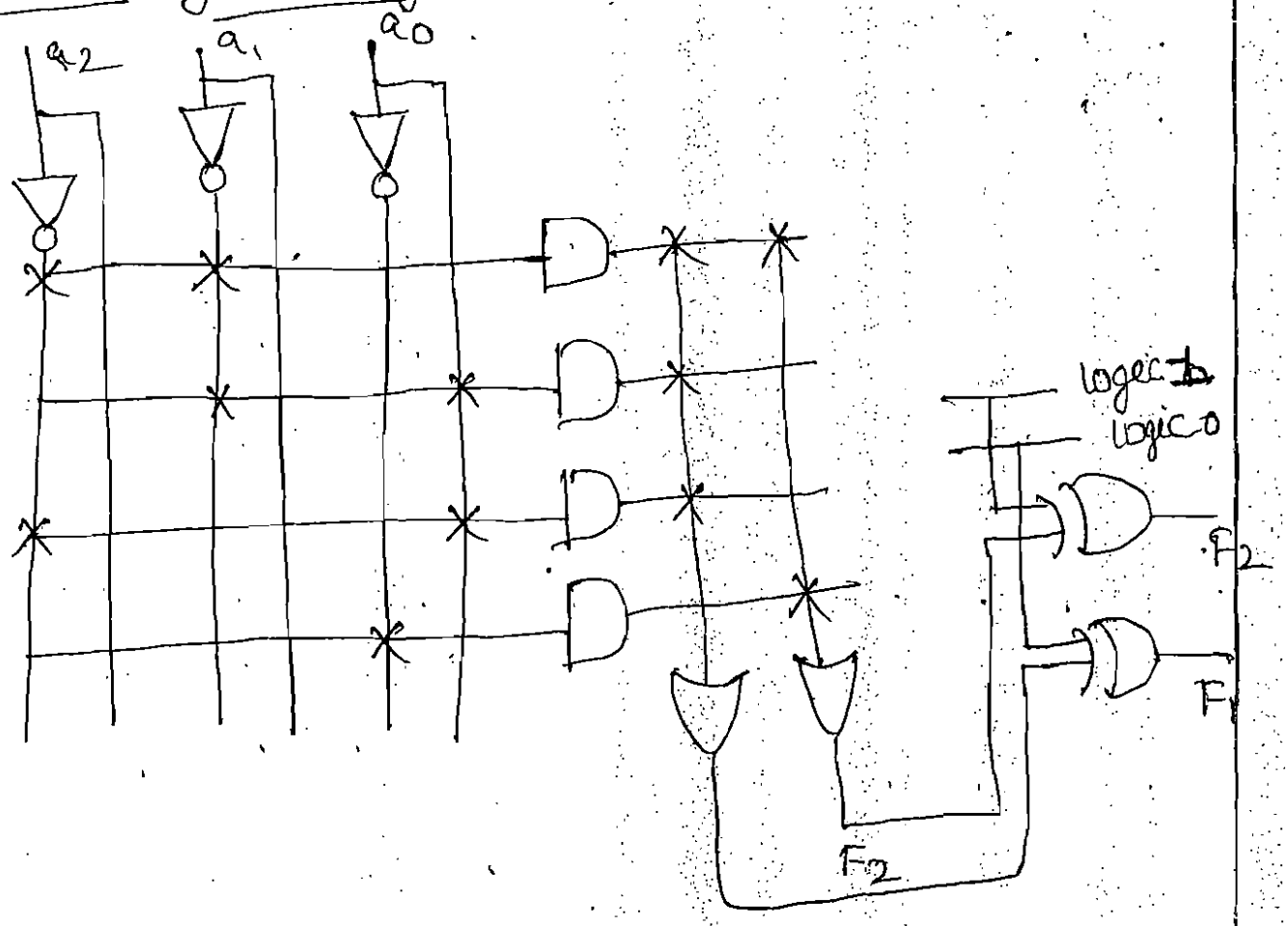
$F_1(T) = \overline{a}_1 a_0 + \overline{a}_2 \overline{a}_1 + \overline{a}_2 a_0$ $F_1(C) = a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

$F_2(T) = a_2 a_0 + a_1 a_0$ $F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

step 2:- PLA programming table.

product terms	Inputs a_2	a_1	a_0	outputs $F_1(T)$	$F_2(C)$
$\overline{a}_1 a_0$	—	0	1	1	—
$\overline{a}_2 \overline{a}_1$	0	0	—	1	1
$\overline{a}_2 a_0$	0	—	1	1	—
$\overline{a}_0$	—	—	0	—	1

Step 3 :- logic diagram :-



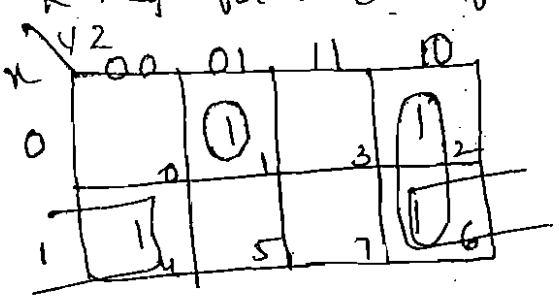
→ Implement the following using PLA

$A(x,y,z) = \Sigma m(1,2,4,6)$, $B(x,y,z) = \Sigma m(0,1,6,7)$

$C(x,y,z) = \Sigma m(2,6)$

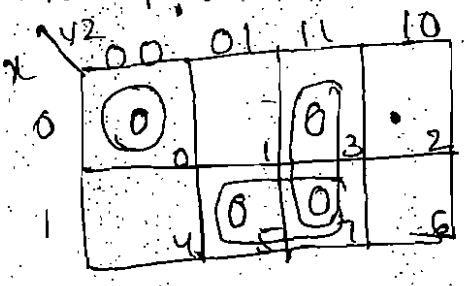
Step 1 :- K-map.

K-map for A (True form).



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

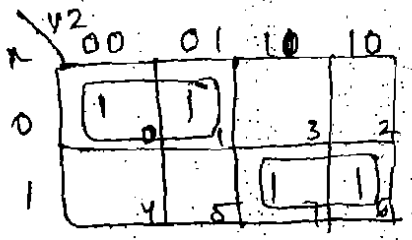
K-map for A (complement form).



$A(C) = \overline{(x+y+z)\cdot(\bar{y}+\bar{z})(\bar{x}+\bar{z})}$

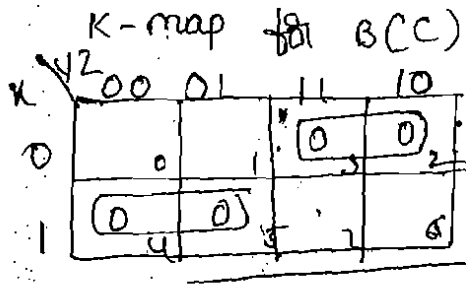
$= \bar{x}\bar{y}\bar{z} + yz + xz.$

K-map for B(T)



$B(T) = \bar{x}y + x\bar{y}$.

K-map for B(C)

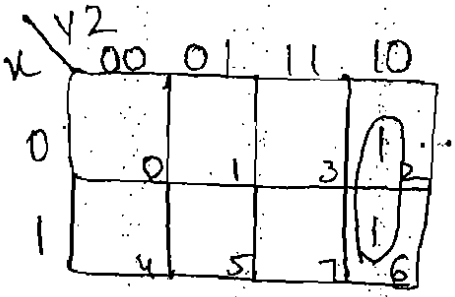


$B(C) = \overline{(\bar{x}+y)(x+\bar{y})}$

$= \bar{x}\cdot\bar{y} + \bar{x}\bar{\bar{y}}$

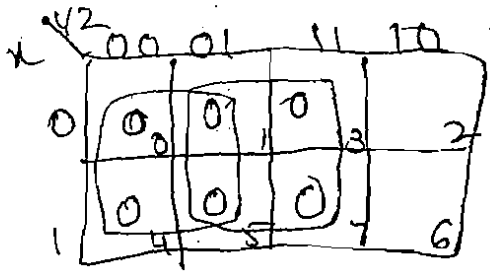
$= x\bar{y} + \bar{x}y$.

K-map for C(T)



$C(T) = y\bar{z}$

K-map for C(C)



$C(C) = \overline{\bar{y}\cdot\bar{z}}$

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

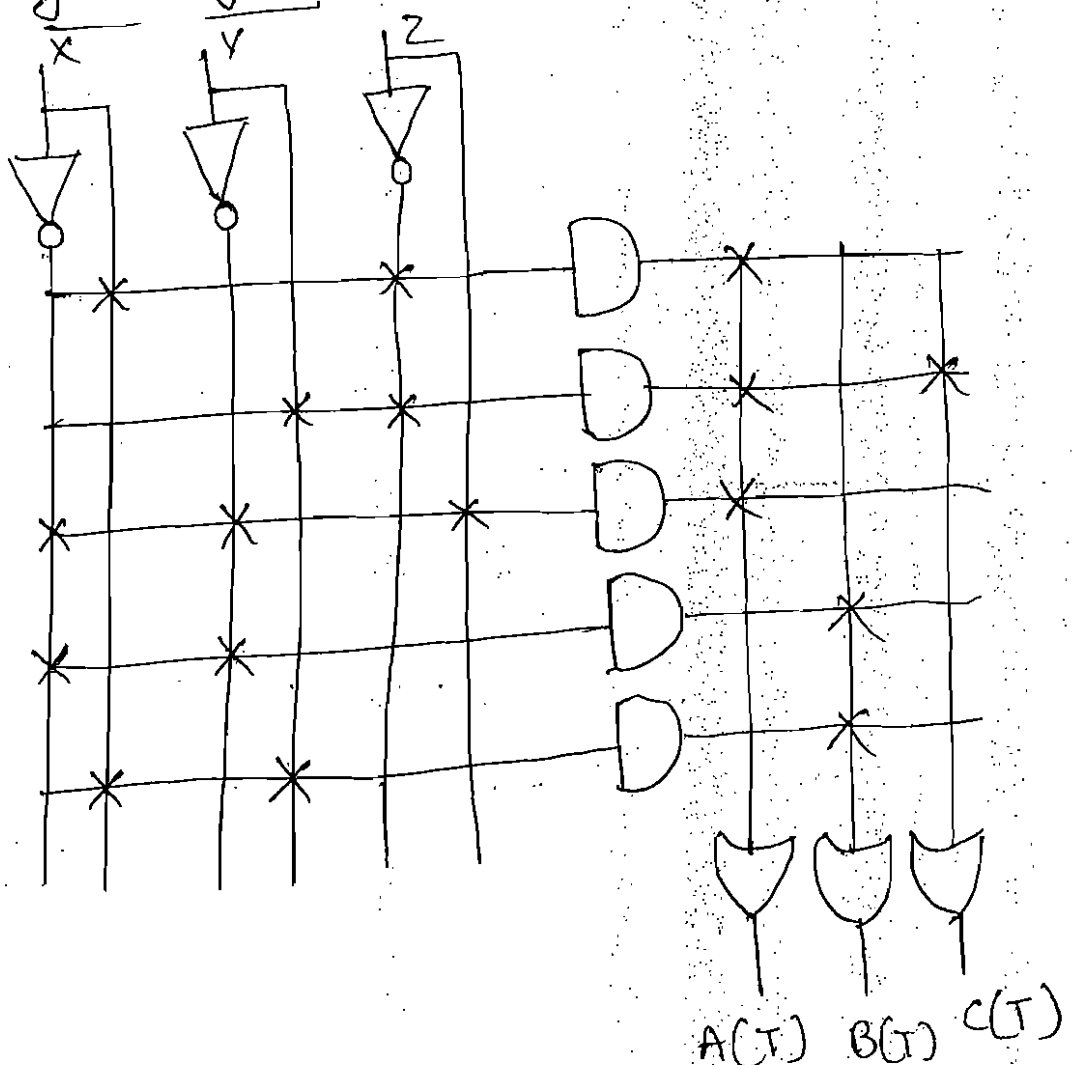
$= \bar{y} + z$.

Step 2 :- programming table.

product term	inputs x	y	z.	outputs A(T)	B(T)	C(T)
$x\bar{z}$	1	–	0	1	–	–
$y\bar{z}$	–	1	0	1	–	1
$\bar{x}\bar{y}z$	0	0	1	1	–	–
$\bar{x}\bar{y}$	0	0	x	–	1	–
xy	1	1	–	–	1	–

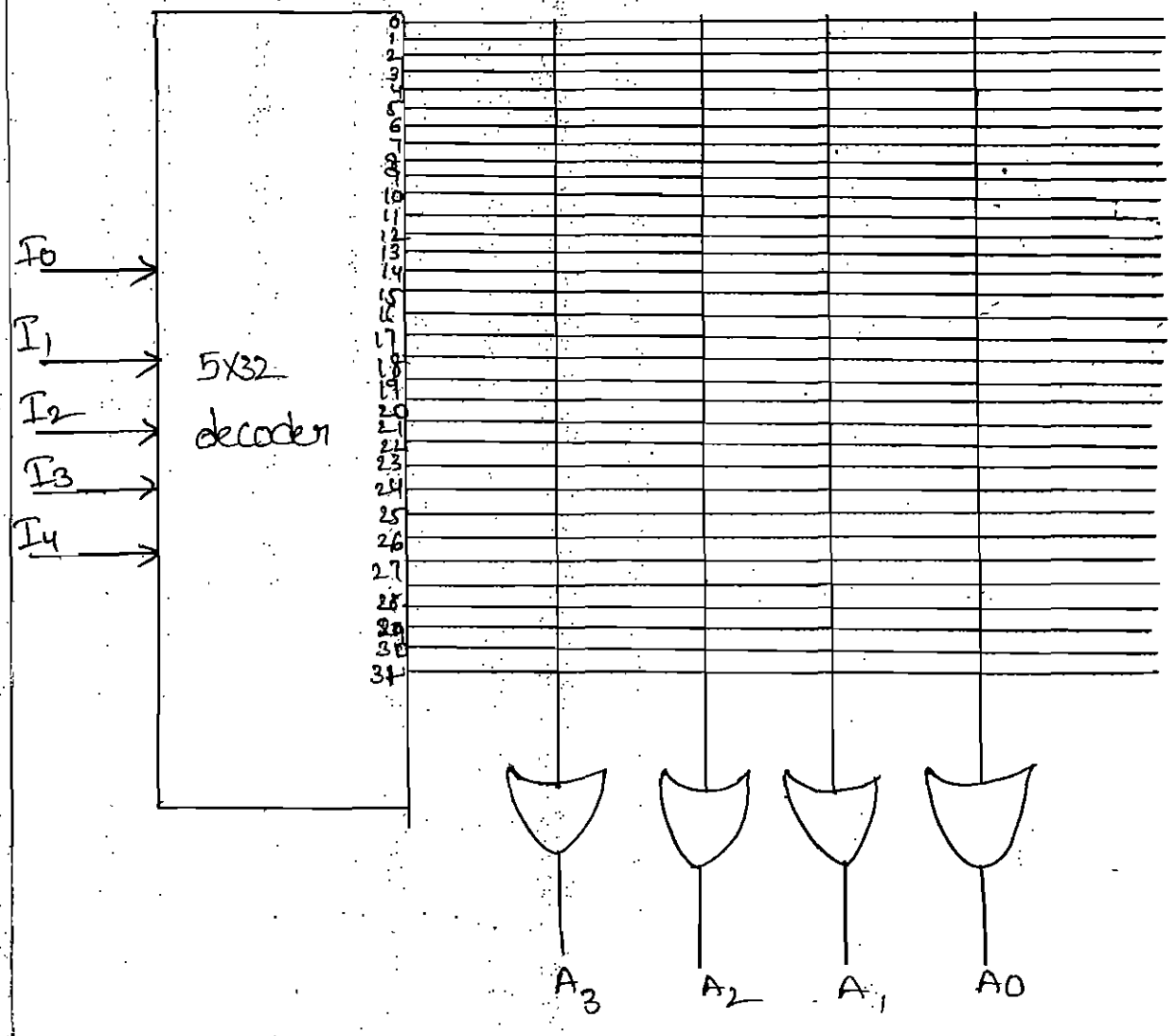
Step - 3

Logic diagram :-



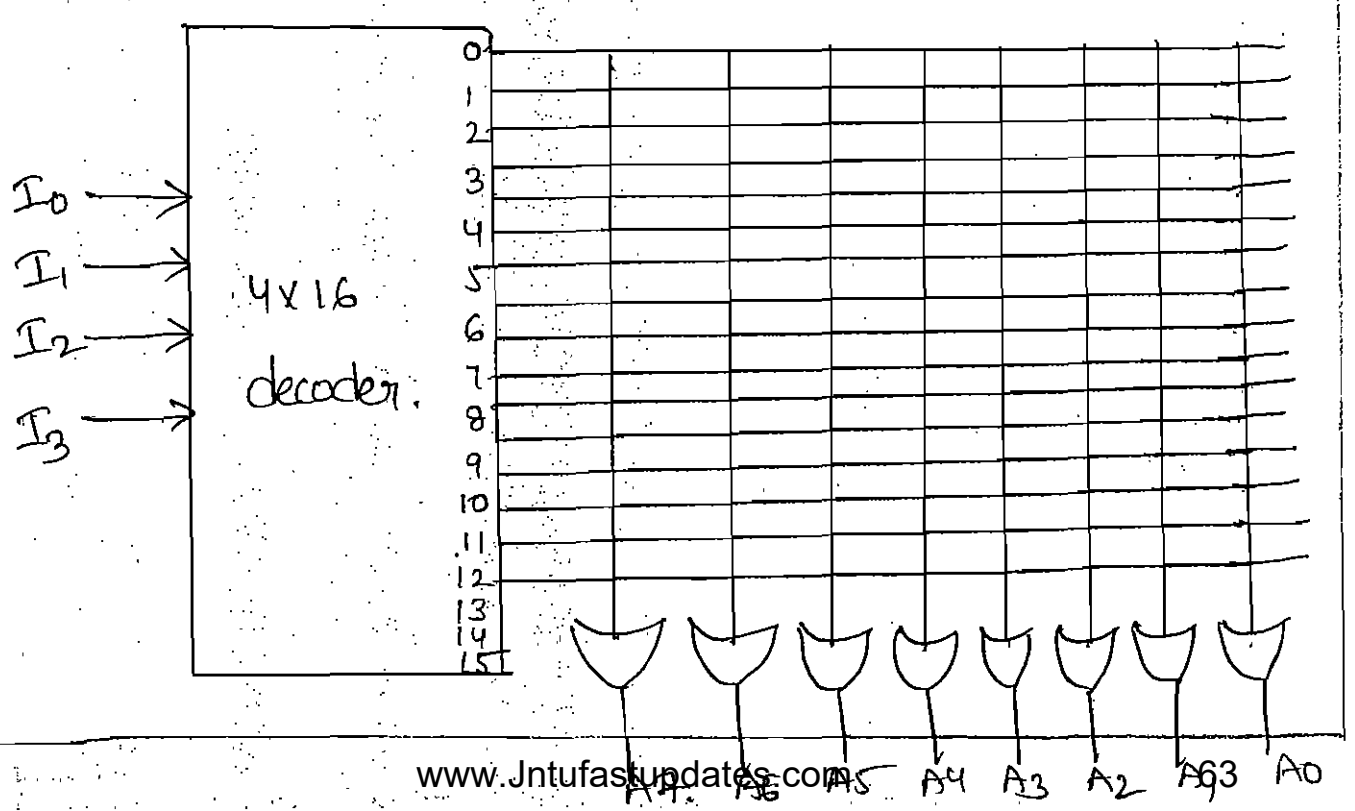
1) Give the logic implementation of a 32x4 bit ROM using a decoder of a suitable size.

A 32x4 bit ROM is to be implemented. it consists of 32 words of four bits each. There must be five input lines that form the binary numbers from 0 through 31 for the address. The five inputs are decoded into 32 distinct outputs by means of a 5x32 decoder. Each output of the decoder represent a memory address. The 32 outputs of the decoder are connected to each of the four OR gates.



32 X 4 bit ROM.

→ 16 X 8 ROM.



Comparison between PROM, PLA and PAL.

PROM	PLA	PAL
1. AND array is fixed and OR array is programmable.	1. Both AND and OR arrays are programmable.	1. OR array is fixed and AND array is programmable.
2. cheaper and simple to use	2. costliest and more complex than PAL and PROM.	2. Cheaper and simpler.
3. ALL minterms are decoded.	3. AND array can be programmed to get desired minterms.	3. AND array can be programmed to get desired minterms.
4. only Boolean functions in standard sop form can be implemented using PROM.	4. Any Boolean function in sop form can be implemented using PROM.	4. Any Boolean function in SOP form can be implemented using PAL.

→ Implement a Binary to BCD code converter by using PAL

→ Iam taking 3-bit Binary P,

Binary B_3 B_2 B_1	BCD code C_4 C_3 C_2 C_1
0 0 0	0 0 0 0
0 0 1	0 0 0 1
0 1 0	0 0 1 0
0 1 1	0 0 1 1
1 0 0	0 1 0 0
1 0 1	0 1 0 1
1 1 0	0 1 1 0
1 1 1	0 1 1 1

→ Ram taking 4-binary.

B_4 B_3 B_2 B_1	C_8 C_7 C_6 C_5	C_4 C_3 C_2 X_1
0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 0	0 0 0 1
0 0 1 0	0 0 0 0	0 0 1 0
0 0 1 1	0 0 0 0	0 0 1 1
0 1 0 0	0 0 0 0	0 1 0 0
0 1 0 1	0 0 0 0	0 1 0 1
0 1 1 0	0 0 0 0	0 1 1 0
0 1 1 1	0 0 0 0	0 1 1 1
1 0 0 0	0 0 0 0	1 0 0 0
1 0 0 1	0 0 0 0	1 0 0 1
1 0 1 0	0 0 0 1	0 0 0 0
1 0 1 1	0 0 0 1	0 0 0 1
1 1 0 0	0 0 0 1	0 0 1 0
1 1 0 1	0 0 0 1	0 0 1 1
1 1 1 0	0 0 0 1	0 1 0 0
1 1 1 1	0 0 0 1	0 1 0 1

$C_5 = Em(10,11,12,13,14,15)$.

$C_4 = Em(8,9)$

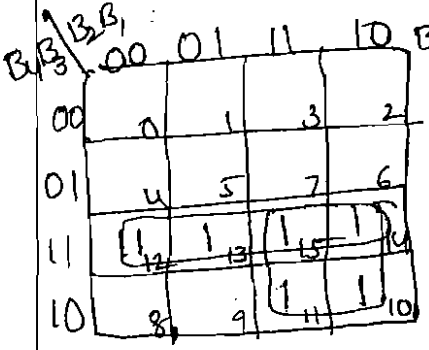
$C_3 = Em(4,5,6,7,14,15)$

$C_2 = Em(2,3,6,7,12,13)$

$C_1 = Em(1,3,5,7,9,11,13,15)$.

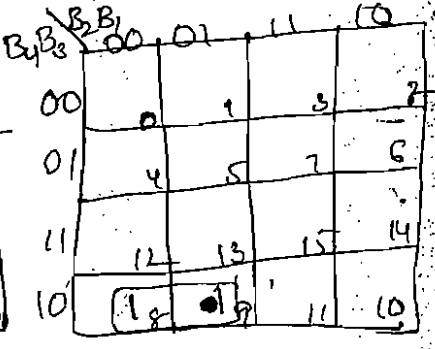
Step 1:- K-map.

K-map for C_5.



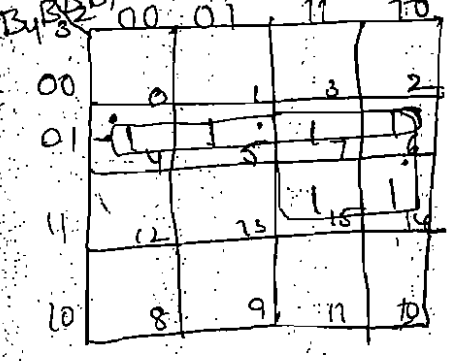
$B_4B_3 + B_4B_2$.

K-map for C_4



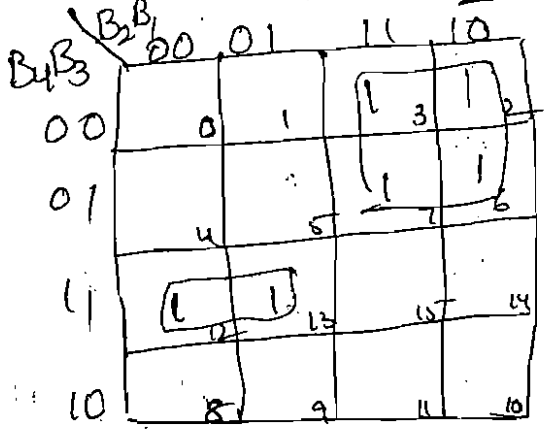
$B_4\overline{B_3}\,\overline{B_2}$

K-map for C_3



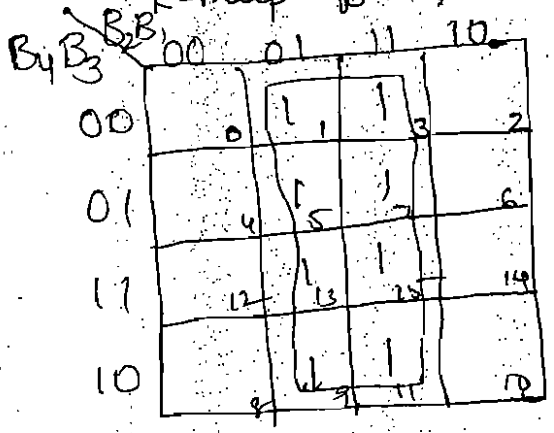
$\overline{B_4}B_3 + B_3B_2$

K-map for C_2



$B_4B_3\overline{B_2} + \overline{B_4}B_2$

K-map for C_1

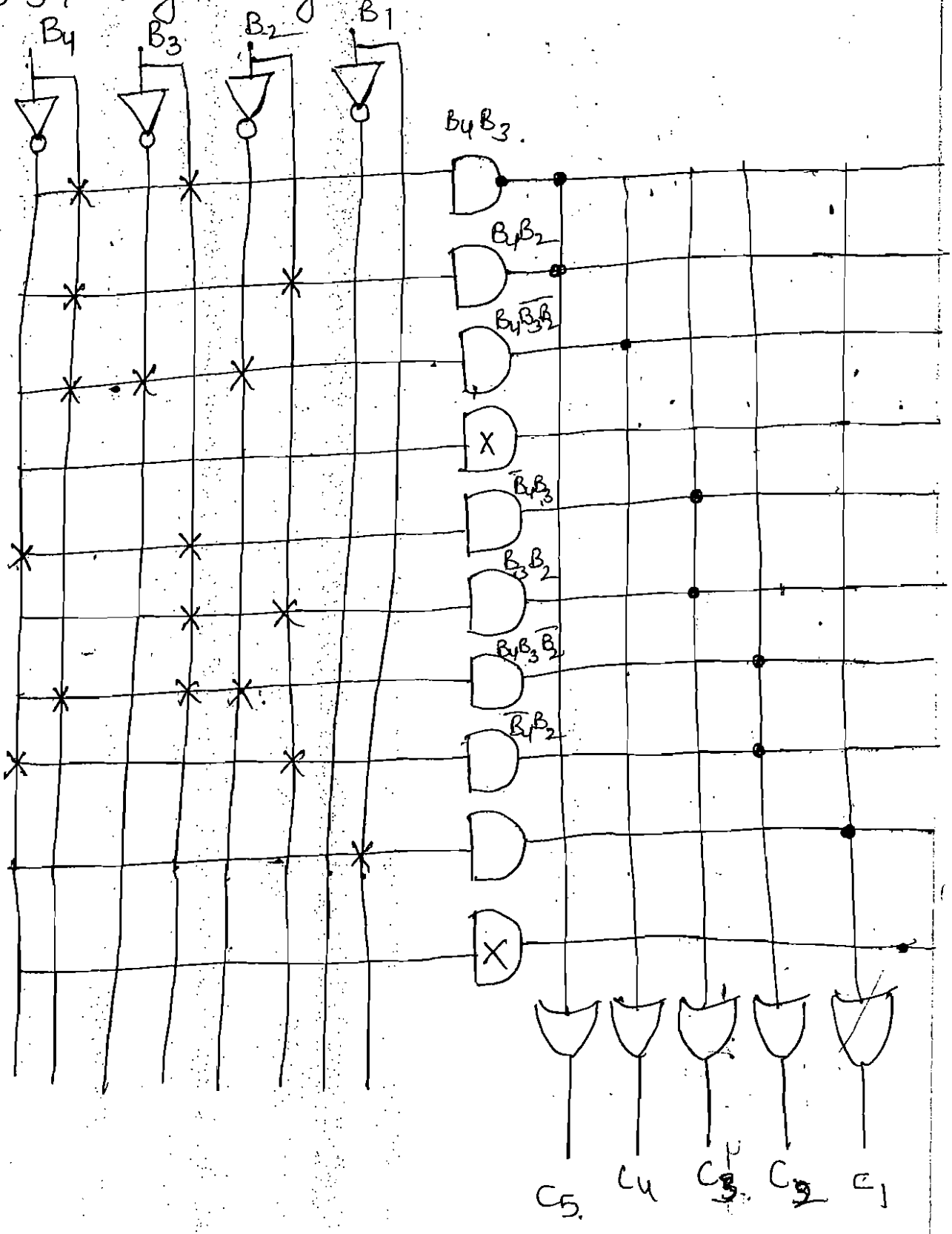


B_1

step 2:- programming table

product terms	inputs B_4 B_3 B_2 B_1	output
1	1 1 – –	$C_5 = B_4B_3 + B_4B_2$
2	1 – 1 –	
3	1 0 0 –	$C_4 = B_4\overline{B_3}\,\overline{B_2}$
4	– – – –	
5	0 1 – –	$C_3 = \overline{B_4}B_3 + B_3B_2$
6	– 1 1 –	
7	1 1 0 –	$C_2 = B_4B_3\overline{B_2} + \overline{B_4}B_2$
8	0 – 1 –	
9	– – – 1	$C_1 = B_1$
10	– – – –	

Step 3 :- logic diagram.



PROGRAMMABLE LOGIC DEVICES

→ Logic designers have a wide range of standard IC's available to them with numerous logic functions and logic circuit arrangements on a chip. In addition, these ICs are available from many manufactures and at a reasonably low cost.

→ PLD is an IC that contains large number of gates, flip-flops and registers that are intercometed on chip. This IC is said to be programmable because the specific function IC is determined by Intercomecting required contacts.

Basically, there are three types of programmable device which are.

→ Read only memory. (ROM)

→ programmable logic array (PLA)

→ programmable Array logic (PAL).

READ ONLY memory :-

→ The read only memory is a type of semiconductor memory that is designed to hold data that is either permanent or will not change frequently.

→ During operation, no new data can be written into a ROM, but data can be read from ROM. the process of Entering data is called programming or burning-in the ROM.

→ Some ROM's cannot have their data changes once they have been programmed others can be erased and reprogrammed as often as desired.

Types of ROM's :-

1. Masked memory ROM
2. programmable read only memory (PROM)
3. Erasable programmable Read only memory (EPROM)
4. Electrically Erasable programmable Read only memory (EEPROM)

Masked memory (ROM) :-

→ cannot be reprogrammed
→ Nonvolatile, retain data even when power is turn off.
→ cheaper than programmable devices.
→ useful for fixed programme instructions.

PROM

→ programmed by blowing built-in fuses.
→ can not be reprogrammed
→ Non volatile.
→ useful for small volume data storing
→ user programmable

EPROM :-

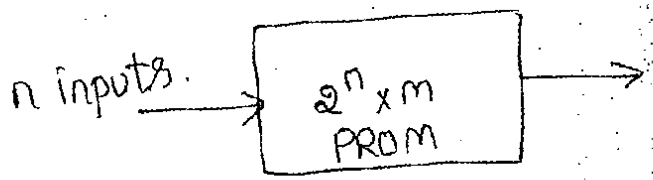
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.
→ NON volatile

Programmable ROM :-

→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination that comes out of the output lines is called a word.



Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array provide an AND-OR Sum of products implementati

EPROM :-

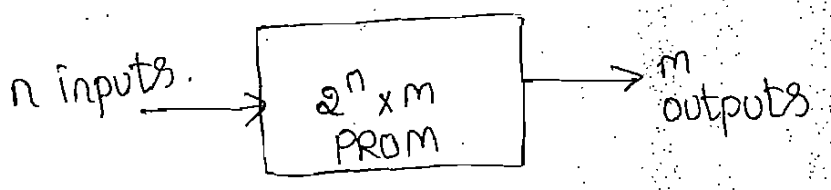
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.
→ NON volatile

Programmable ROM :-

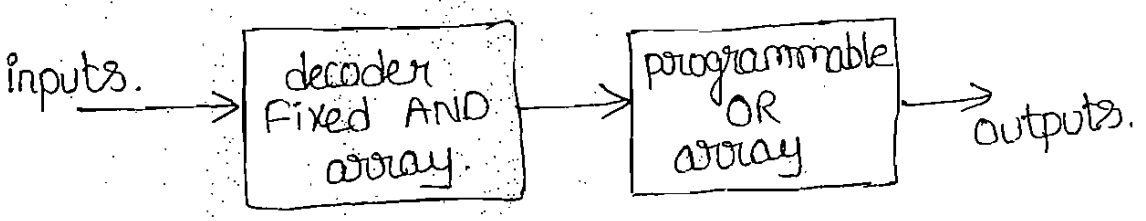
→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination that comes out of the output lines is called a word.

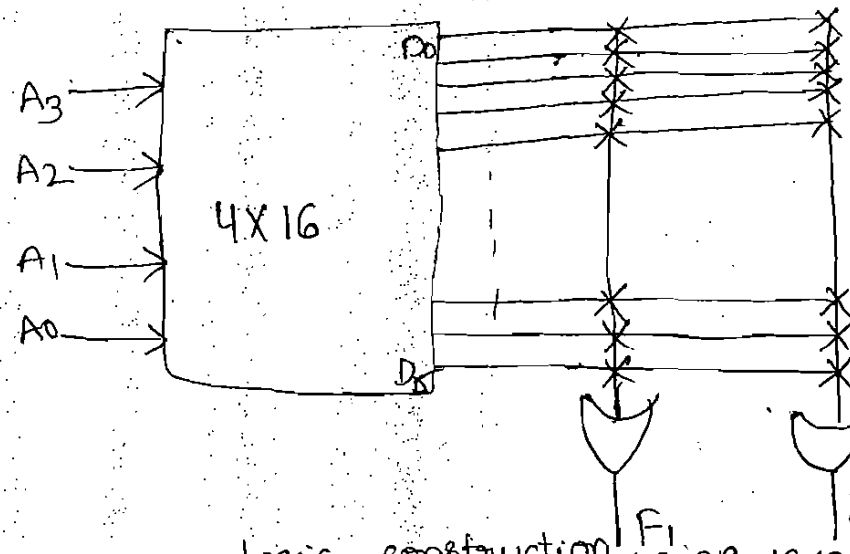


Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR Sum of products implementation.

The programmable read-only memory (PROM) has a fixed AND array constructed as a decoder and a programmable OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR Gate.





Logic construction using 16X2 ROM.

→ Implement full-adder using PROM.

The number of inputs variables of a full adder are 3. The possible number of combinations are 8. So we need 3X8 decoder. The number of outputs of full adder are 2. They are Sum and carry.

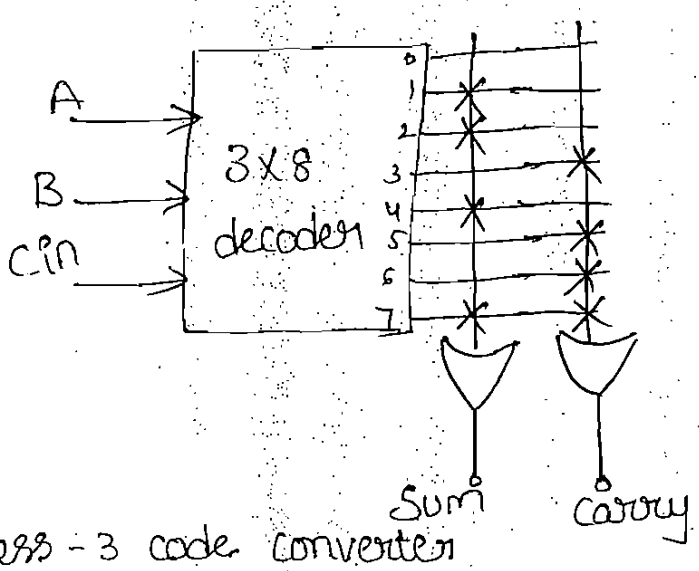
Truth table

A	B	C_{in}	S	C
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

output expression for sum & carry is Sum = $\Sigma m(1,2,4,7)$

carry = $\Sigma m(3,5,6,7)$

Logic diagram



→ Design BCD to Excess-3 code converter using PROM.

Step:- 1 – Truth table.

BCD				EXCESS-3			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1

Step 2:-

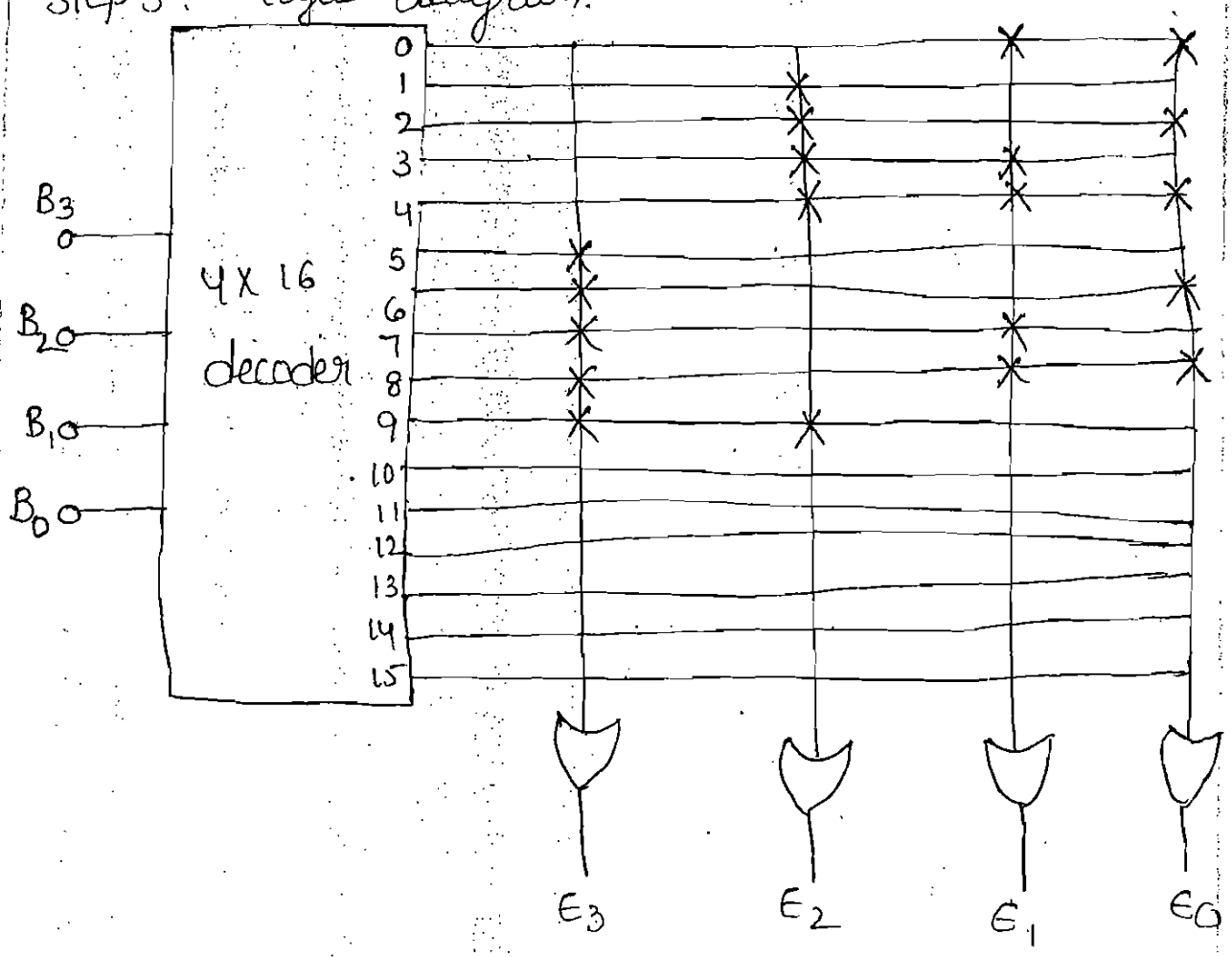
$E_0(B_3\ B_2\ B_1\ B_0) = Em(0,2,4,6,8)$

$E_1(B_3\ B_2\ B_1\ B_0) = Em(0,3,4,7,8)$

$E_2(B_3\ B_2\ B_1\ B_0) = Em(1,2,3,4,9)$

$E_3(B_3\ B_2\ B_1\ B_0) = Em(5,6,7,8,9)$

Step 3:- logic diagram.



→ Implement the following Boolean expression using PROM.

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

first given expression converted into a standard sop form.

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

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

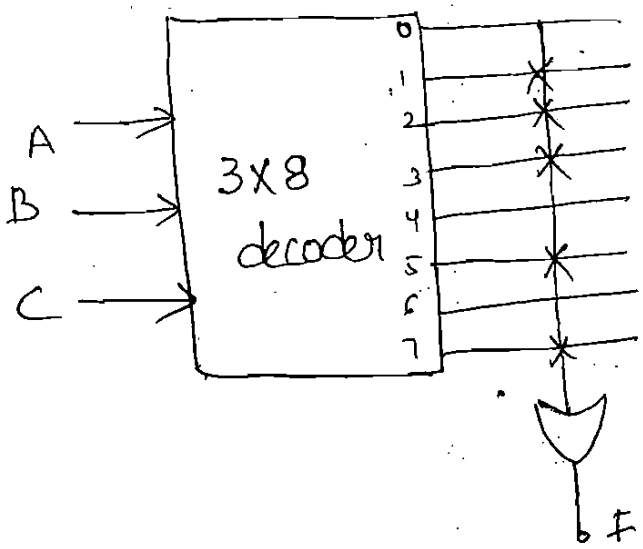
$$= \underset{1}{\bar{A}BC} + \bar{A}B\bar{C} + \underset{2}{ABC} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \underset{2}{AB\bar{C}} + \underset{1}{\bar{A}\bar{B}C}$$

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

$$011,\ 010,\ 111,\ 101,\ 011,\ 001$$

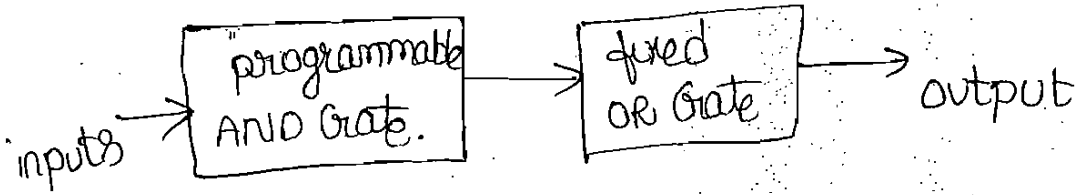
$$f(A,B,C) = \Sigma m(1,2,3,5,7)$$

Logic diagram :-



PAL :- programmable Array logic :-

The programmable Array logic is a programmable device with a fixed OR array and a programmable AND array because, only the AND gates are programmble.

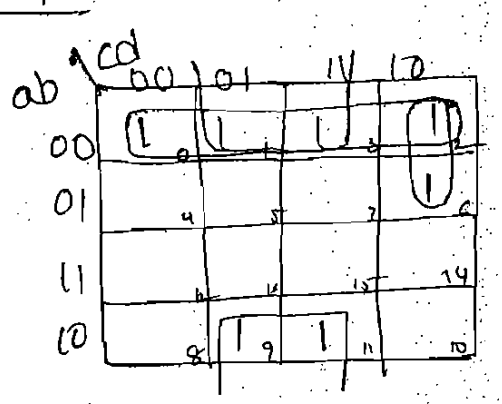


→ Implement the following functions using PAL.

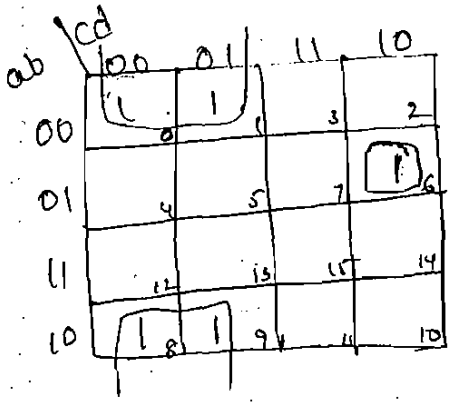
$F_1(a,b,c,d) = \Sigma m(0,1,2,3,6,9,11)$

$F_2(a,b,c,d) = \Sigma m(0,1,6,8,9)$

Step 1:- K-map simplification for F_1 and F_2



$\bar{a}\bar{b} + \bar{a}c\bar{d} + \bar{b}d$

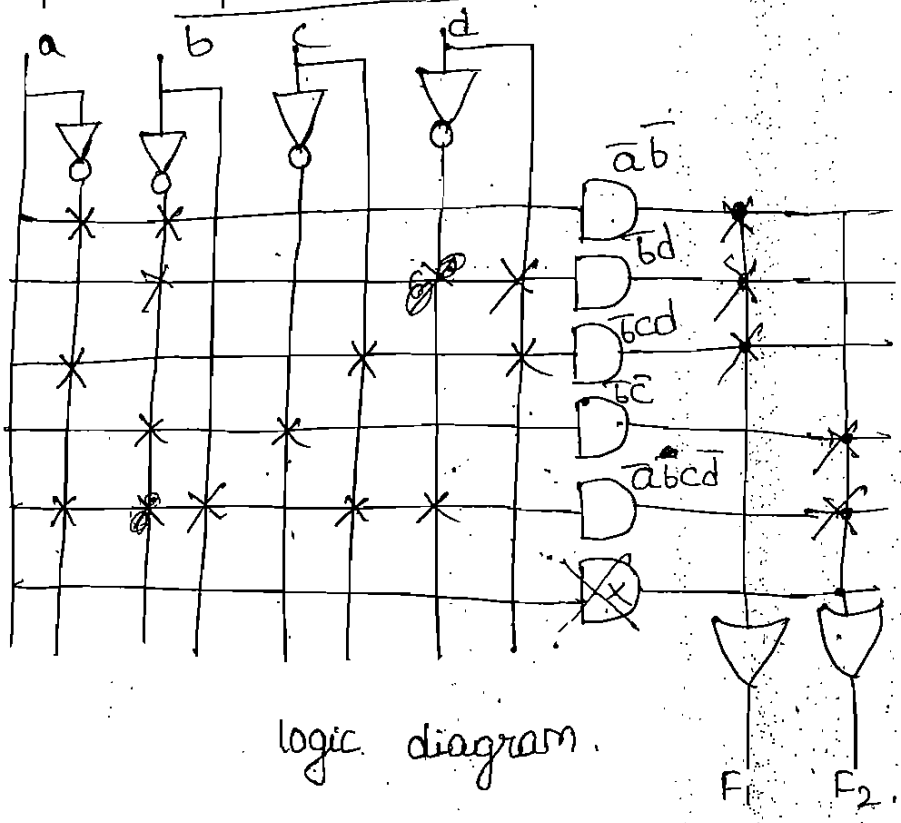


$\bar{b}\bar{c} + \bar{a}bc\bar{d}$

Step 2 :- PAL programming table.

product terms	AND gate inputs a b c d	outputs
1	0 0 – –	$F_1 = \bar{a}\bar{b} + \bar{a}cd + \bar{b}d$
2	0 – 1 1	
3	– 0 – 1	
4	– 0 0 –	$F_2 = \bar{b}\bar{c} + \bar{a}bc\bar{d}$
5	0 1 1 0	
6	– – – –	

Step 3:- implementation:-



logic diagram.

→ Implement 4-bit BCD to XS-3 code conversion using PAL.

Step:-1

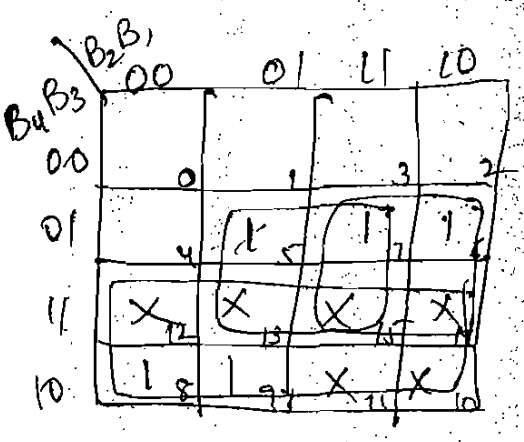
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

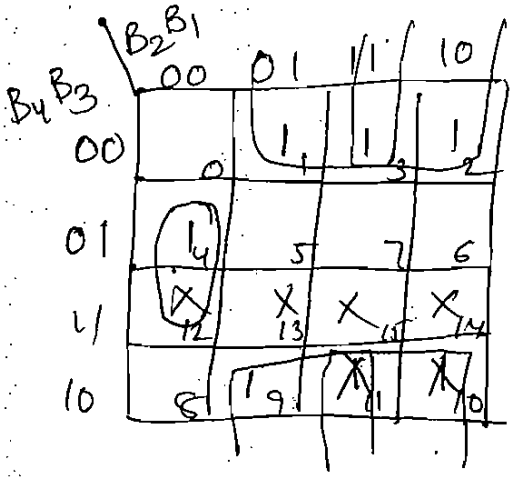
$X_4 = \Sigma m(5,6,7,8,9) + d(10,11,12,13,14,15)$

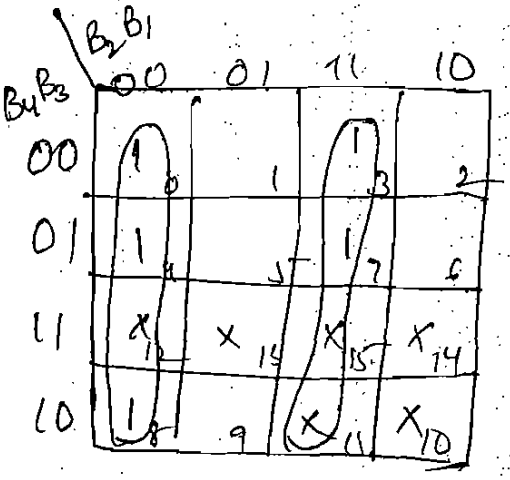
$X_3 = \Sigma m(1,2,3,4,9) + d(10,11,12,13,14,15)$

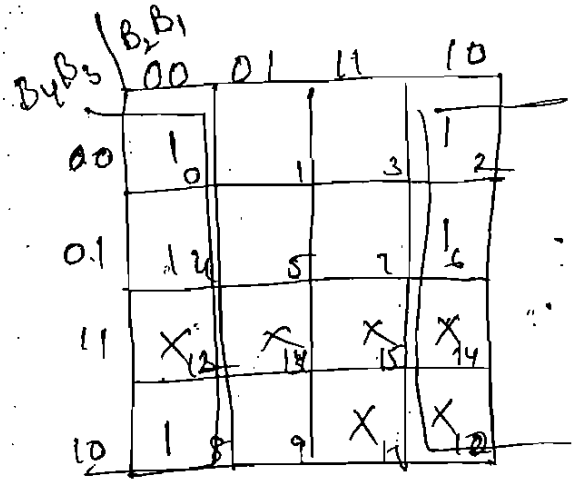
$X_2 = \Sigma m(0,3,4,7,8) + d(10,11,12,13,14,15)$

$X_1 = \Sigma m(0,2,4,6,8) + d(10,11,12,13,14,15)$









$X_4 = B_4 + B_3B_2 + B_3B_1$

$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$

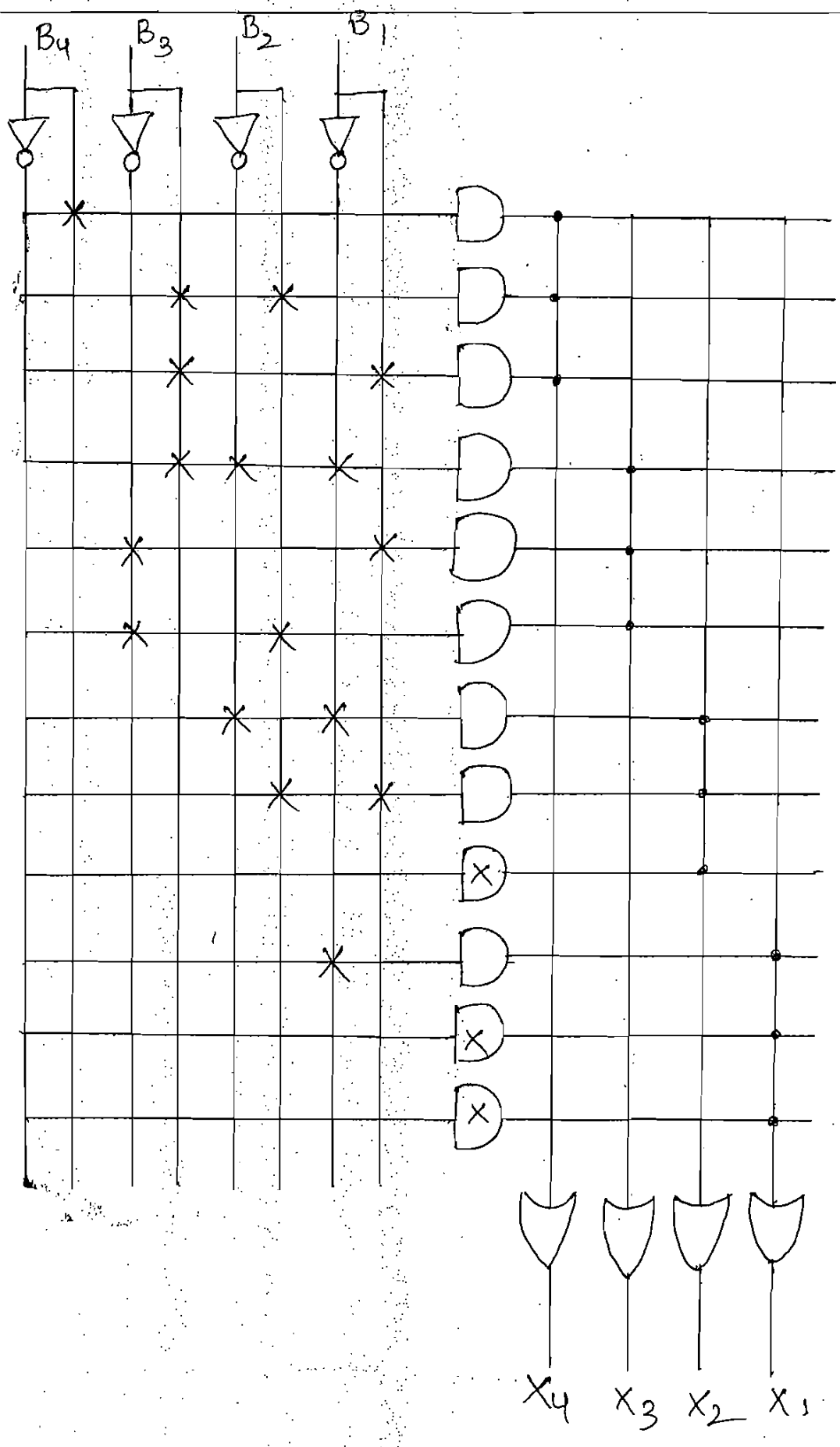
$X_2 = \overline{B_2\overline{B_1} + B_2B_1}$

$X_1 = \overline{B_1}$

step 2 :- programming table.

product terms.	AND Gate inputs. B_4	B_3	B_2	B_1	Outputs.
1	1	–	–	–	$X_4 = B_4 + B_3B_2 + B_3B_1$
2	–	1	1	–	
3	–	1	–	1	
4	–	1	0	0	$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$
5	–	0	–	1	
6	–	0	1	–	
7	–	–	0	0	$X_2 = \overline{B_2}\,\overline{B_1} + B_2B_1$
8	–	–	1	1	
9	–	–	–	–	
10	–	–	–	0	$X_1 = \overline{B_1}$
11	–	–	–	–	
12	–	–	–	–	

step 3 :- logic diagram.



logic diagram.

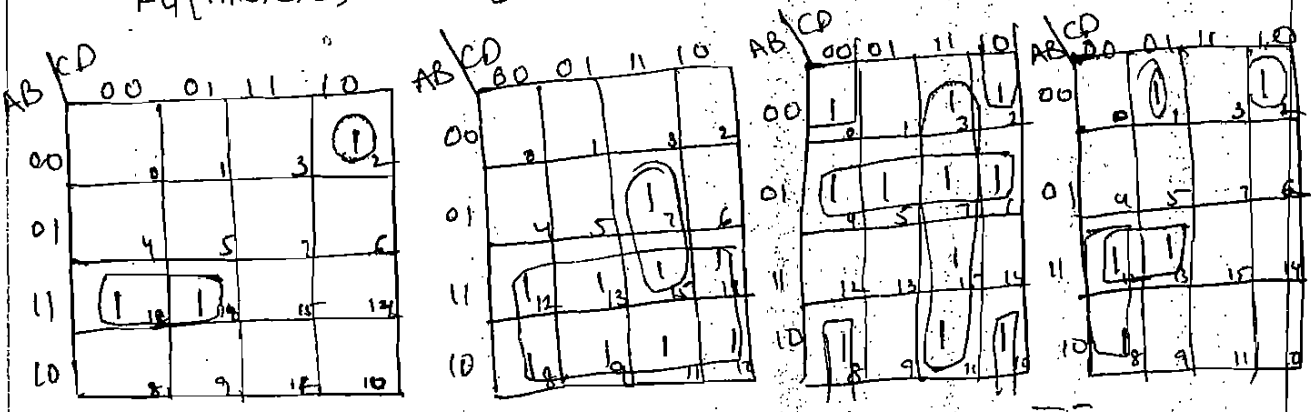
Implement the following boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

$F_1(A,B,C,D) = \Sigma m(2,12,13)$

$F_2(A,B,C,D) = \Sigma m(7,8,9,10,11,12,13,14,15)$

$F_3(A,B,C,D) = \Sigma m(0,2,3,4,5,6,7,8,10,11,15)$

$F_4(A,B,C,D) = \Sigma m(1,2,8,12,13)$



$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$ $F_2 = A + BCD$ $F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$

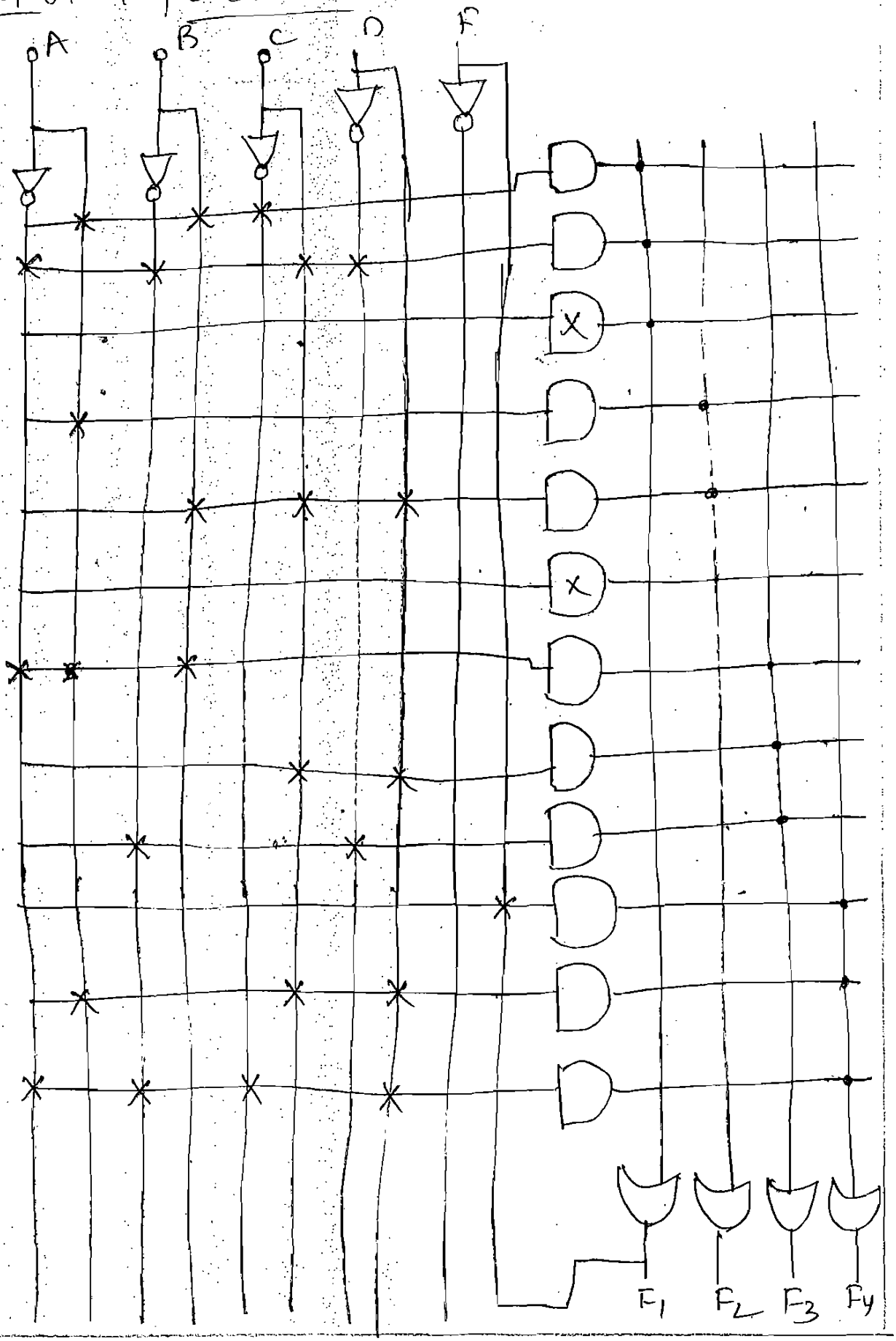
$F_4 = AB\bar{C} + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D}$

$= F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$

Step 2:- programming table

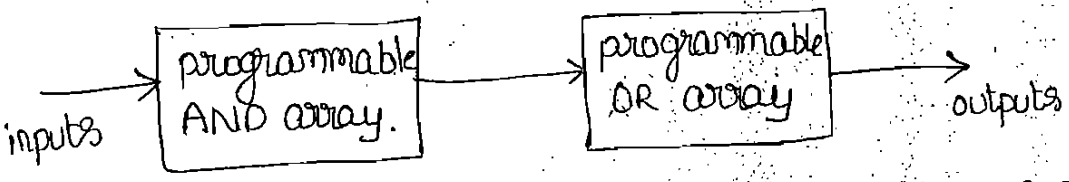
product term	AND inputs A B C D	F_1	outputs
1	1 1 0 –		$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$
2	0 0 1 0		
3	– – – –		– – –
4	1 – – –		$F_2 = A + BCD$
5	– 1 1 1		
6	– – – –		– – –
7	0 1 – –		$F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$
8	– – 1 1		
9	– 0 – 0		– – –
10	– – – –	1	
11	1 – 0 0		$F_4 = F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$
12	0 0 0 1		

Step 3 :- Implementation :-



PLA :- programmable logic Array

In programmable logic array where both the AND and OR arrays can be programmed. The product terms in the AND array may be shared by any OR gate to provide the required sum of products.

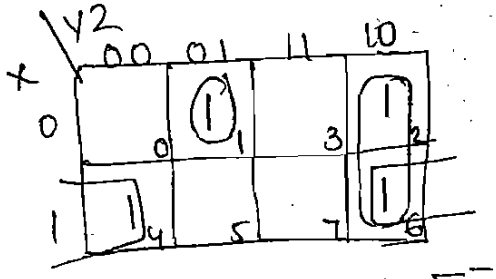


→ implement the given boolean functions by using PLA.

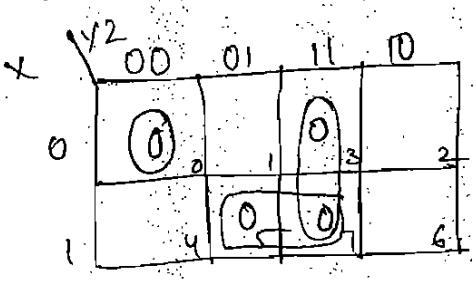
$A(X,Y,Z) = \Sigma m(1,2,4,6)$

$B(X,Y,Z) = \Sigma m(1,2,3,5,7)$

Step 1 :- The K-maps for the functions A, B, their minimized on, and the minimal expressions for both the true and complement of those in sum of products.



$A(T) = X\bar{Z} + Y\bar{Z} + \bar{X}\bar{Y}Z.$



$\bar{A} = XZ + YZ + \bar{X}\bar{Y}\bar{Z}$

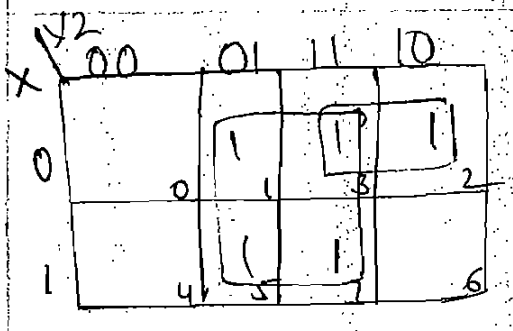
$A(C) = \overline{XZ + YZ + \bar{X}\bar{Y}\bar{Z}}$

K-map for A.

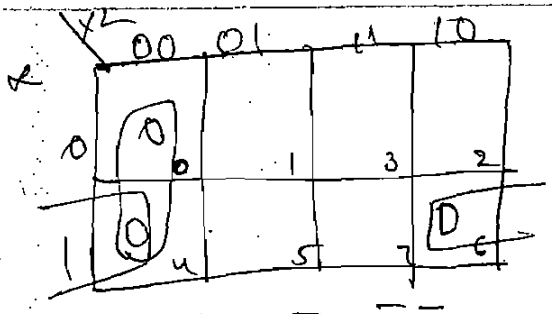
Simply $A(C) = \overline{(\bar{X}+\bar{Z})\cdot(\bar{Y}+\bar{Z})\cdot(X+Y+Z)}$

$= \overline{(\bar{X}+\bar{Z})} + \overline{(\bar{Y}+\bar{Z})} + \overline{(X+Y+Z)}$

$= XZ + YZ + \bar{X}\bar{Y}\bar{Z}$



$B(T) = \overline{X}\overline{Y} + Z$.



$\overline{B} = X\overline{Z} + \overline{Y}\,\overline{Z}$

$B(C) = \overline{X\overline{Z} + \overline{Y}\overline{Z}}$

Simply $B(C) = \overline{(\overline{Y}+Z)(\overline{X}+Z)}$

$= \overline{(\overline{Y}+Z)} + \overline{(\overline{X}+Z)}$

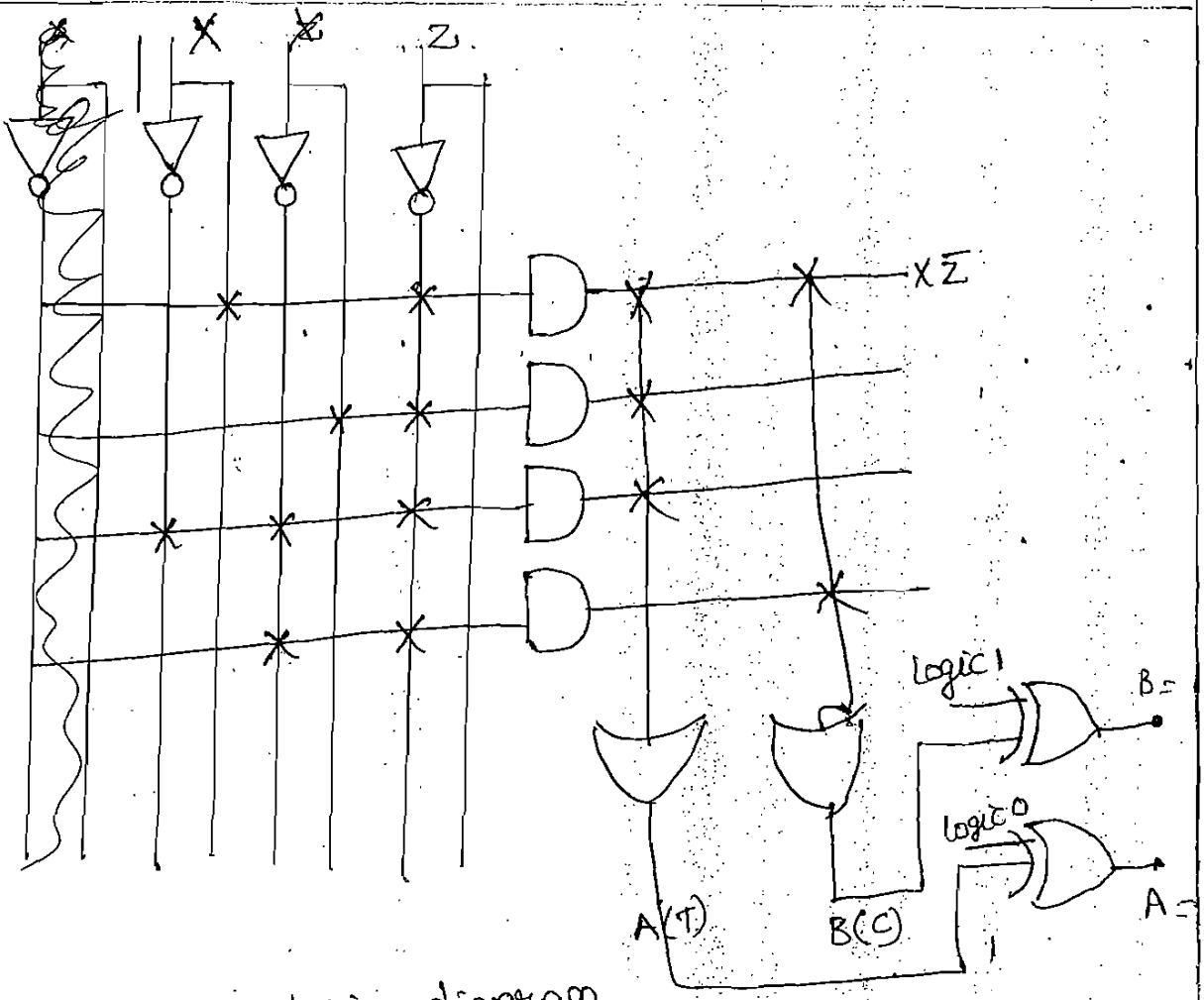
$= \overline{Y}\overline{Z} + X\overline{Z}$

$A(T) = X\overline{Z} + Y\overline{Z} + \overline{X}\,\overline{Y}Z$ $\quad B(T) = \overline{X}Y + Z$

$A(C) = XZ + YZ + \overline{XYZ}$ $\quad B(C) = \overline{Y}\overline{Z} + X\overline{Z}$

Step 2:- programming table.

product term	inputs X	Y	Z	outputs A(T)	A(C)	B(T)	B(C)
$X\overline{Z}$	1	–	0	1	–	–	1
$Y\overline{Z}$	–	1	0	1	–	–	–
$\overline{X}\,\overline{Y}Z$	0	0	1	1	–	–	–
XZ	1	–	1	–	1	–	–
YZ	–	1	1	–	1	–	–
$\overline{XYZ}$	0	0	0	–	1	–	–
$\overline{X}Y$	0	1	–	–	–	1	–
Z	–	–	1	–	–	1	–
$\overline{Y}\,\overline{Z}$	–	0	0	–	–	–	1
$X\overline{Z}$	1	–	0	–	–	–	1



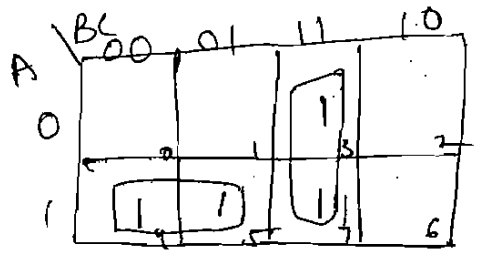
logic diagram.

→ Implement the following boolean functions F_1 & F_2 of a combinational logic circuit using PLA.

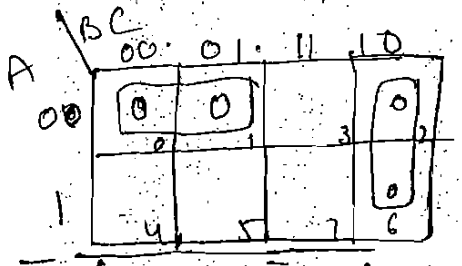
$F_1(A,B,C) = \Sigma m(3,4,5,7)$

$F_2(A,B,C) = \Sigma m(1,4,6)$.

Step 1;- K-map for F_1 (True form) Complement form



$F_1 = A\bar{B} + BC$



$\bar{F_1} = (A+B)(\bar{B}+C)$

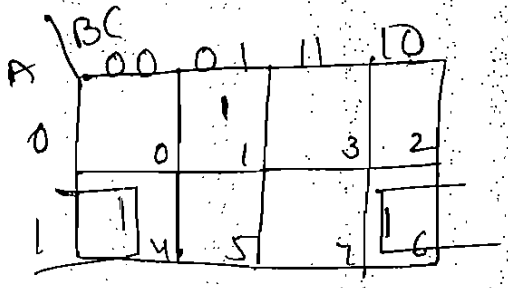
$= \overline{(A+B)} + \overline{(\bar{B}+C)}$

$F_1(T) = A\bar{B} + BC$

$F_1(C) = \overline{(\bar{A}.B) + \bar{B}.\bar{C}}$

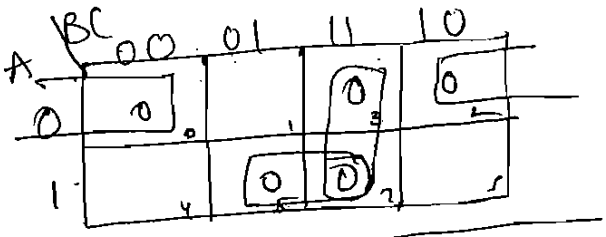
$= \bar{A}.\bar{B} + B.\bar{C}$

K-map for F_2 (true form)



$F_2 = \bar{A}\bar{B}C + A\bar{C}$

K-map for F_2 (complement form)



$\bar{F}_2 = \overline{(A+C).(\bar{B}+\bar{C})(\bar{A}+\bar{C})}$

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

$= \bar{A}.\bar{C} + BC + AC.$

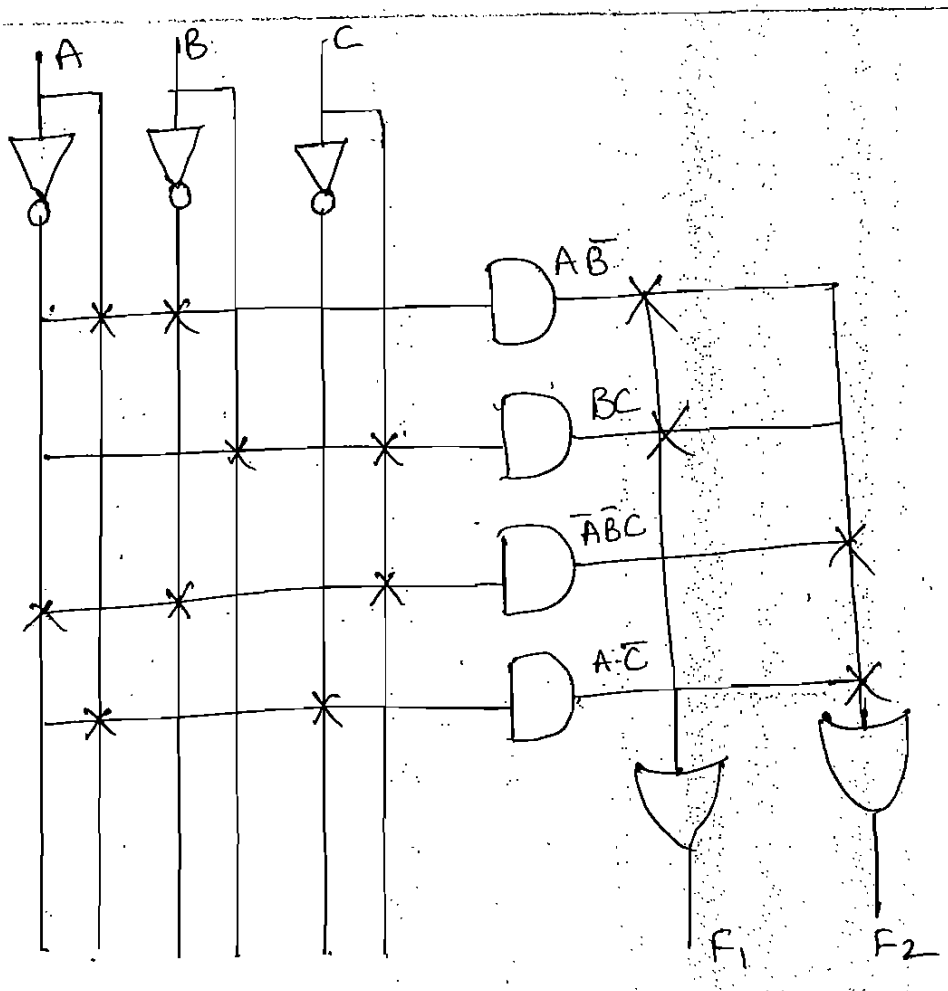
$F_1(T) = A\bar{B} + BC$ $\quad F_1(C) = \bar{A}\bar{B} + B\bar{C}$

$F_2(T) = \bar{A}\bar{B}C + A\bar{C}$ $\quad F_2(C) = \bar{A}\bar{C} + BC + AC$

When we take $F_1(T)$ & $F_2(T)$ get 4 product terms and also having same number of product terms while taking $F_1(T)$ & $F_2(C)$, $F_1(C)$ & $F_2(T)$. So we have to consider $F_1(T)$ & $F_2(T)$.

Step 2 :- programming table

product terms	inputs A	B	C	Outputs $F_1(T)$	$F_2(C)$
$A\bar{B}$	1	0	–	1	–
BC	–	1	1	1	1
$\bar{A}\bar{B}C$	0	–	0	1	1
$A\bar{C}$	1	–	1	–	1



logic diagram.

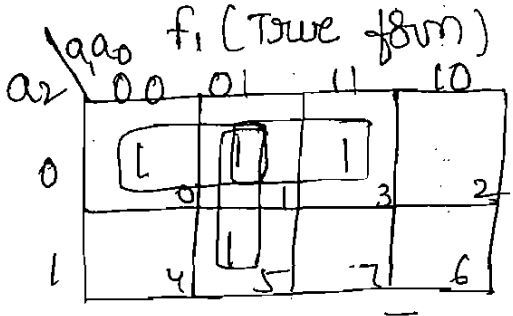
→ Implement the following multi boolean function using (3x4x2) PLA.

$F_1(a_2, a_1, a_0) = \Sigma m(0,1,3,5)$

$F_2(a_2, a_1, a_0) = \Sigma m(3,5,7)$

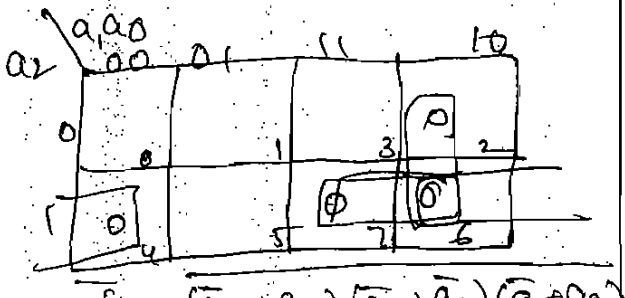
step 1:- K-maps.

f_1 (True form)



$f_1(T) = \bar{a}_1a_0 + \bar{a}_2\bar{a}_1 + \bar{a}_2a_0$

f_1 (complement form)

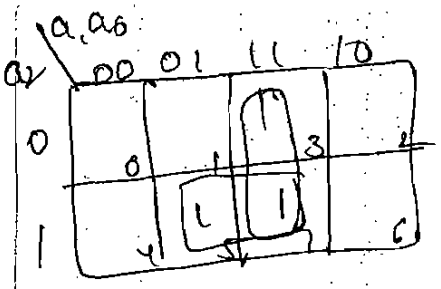


$\bar{f}_1 = (\bar{a}_2 + a_0)(\bar{a}_2 + \bar{a}_1)(\bar{a}_1 + a_0)$

$f_1(C) = \overline{\overline{a}_2} \cdot a_0 + \overline{\overline{a}_2} \cdot \overline{\overline{a}_1} + \overline{\overline{a}_1} \cdot \overline{a}_0$

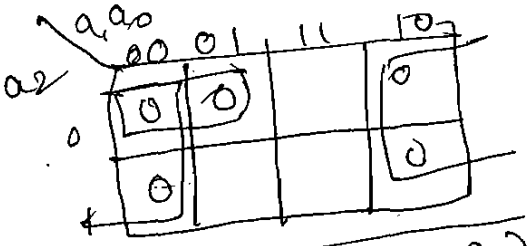
$= a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

f_2(True form)



$f_2(T) = a_2 a_0 + a_1 a_0$

F_2 (complement form..



$\overline{F}_2 = (\overline{a}_0) \cdot (a_2 + a_1)$

$F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

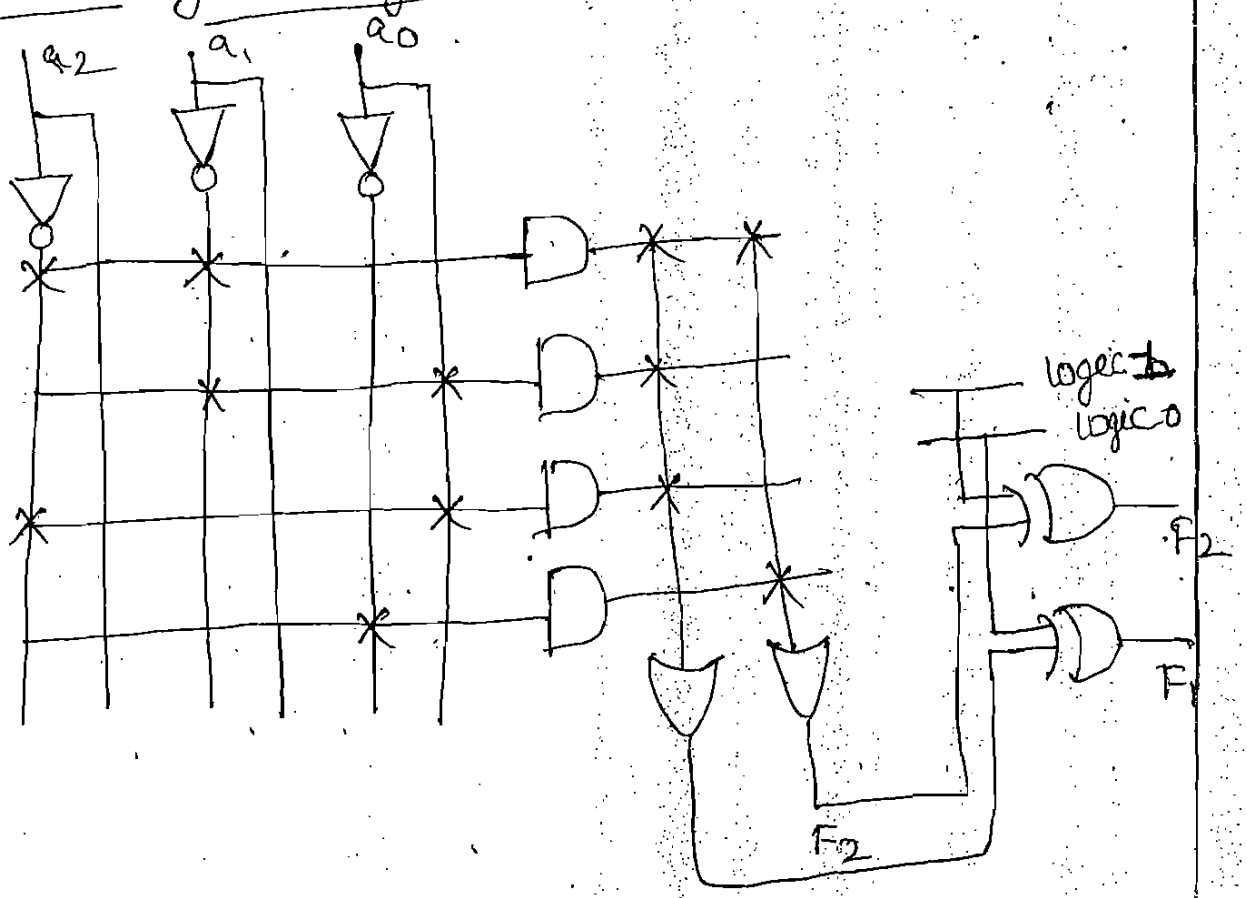
$F_1(T) = \overline{a}_1 a_0 + \overline{a}_2 \overline{a}_1 + \overline{a}_2 a_0$ $\quad F_1(C) = a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

$F_2(T) = a_2 a_0 + a_1 a_0$ $\quad F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

step 2:- PLA programming table.

product terms	Inputs a_2	a_1	a_0	outputs $F_1(T)$	$F_2(C)$
$\overline{a}_1 a_0$	—	0	1	1	—
$\overline{a}_2 \overline{a}_1$	0	0	—	1	1
$\overline{a}_2 a_0$	0	—	1	1	—
$\overline{a}_0$	—	—	0	—	1

Step 3:- logic diagram:-



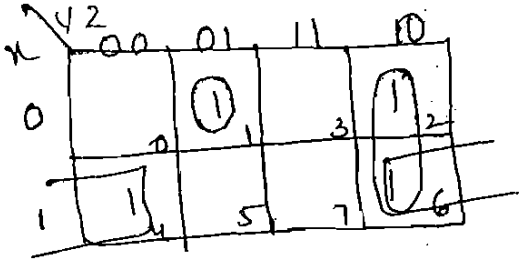
→ Implement the following using PLA

$A(x,y,z) = \Sigma m(1,2,4,6)$, $B(x,y,z) = \Sigma m(0,1,6,7)$

$C(x,y,z) = \Sigma m(2,6)$.

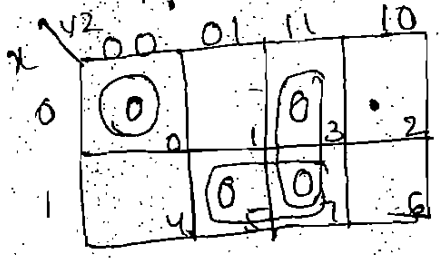
Step 1:- K-map.

K-map for A (True form).



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

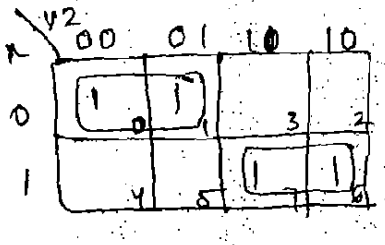
K-map for A (complement form).



$A(C) = \overline{(x+y+z)\cdot(\bar{y}+\bar{z})(\bar{x}+\bar{z})}$

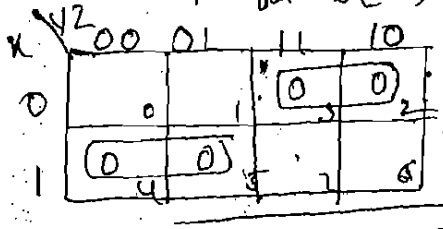
$= \bar{x}\bar{y}\bar{z} + yz + xz$.

K-map for B(T)



$B(T) = \bar{x}y + x\bar{y}$.

K-map for B(C)

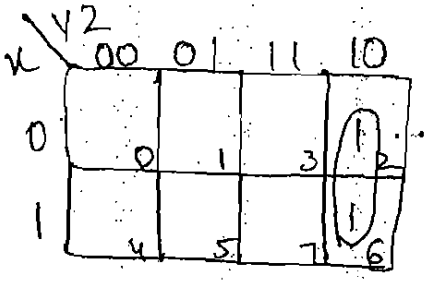


$B(C) = \overline{(\bar{x}+y)(x+\bar{y})}$

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

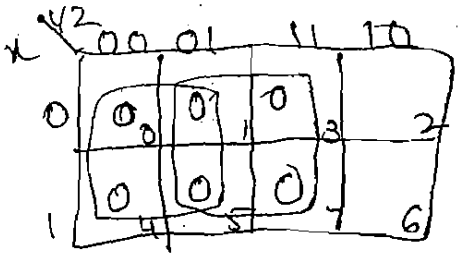
$= x\bar{y} + \bar{x}y$.

K-map for C(T)



$C(T) = y\bar{z}$

K-map for C(C)



$C(C) = \bar{y} \overline{(y \cdot \bar{z})}$

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

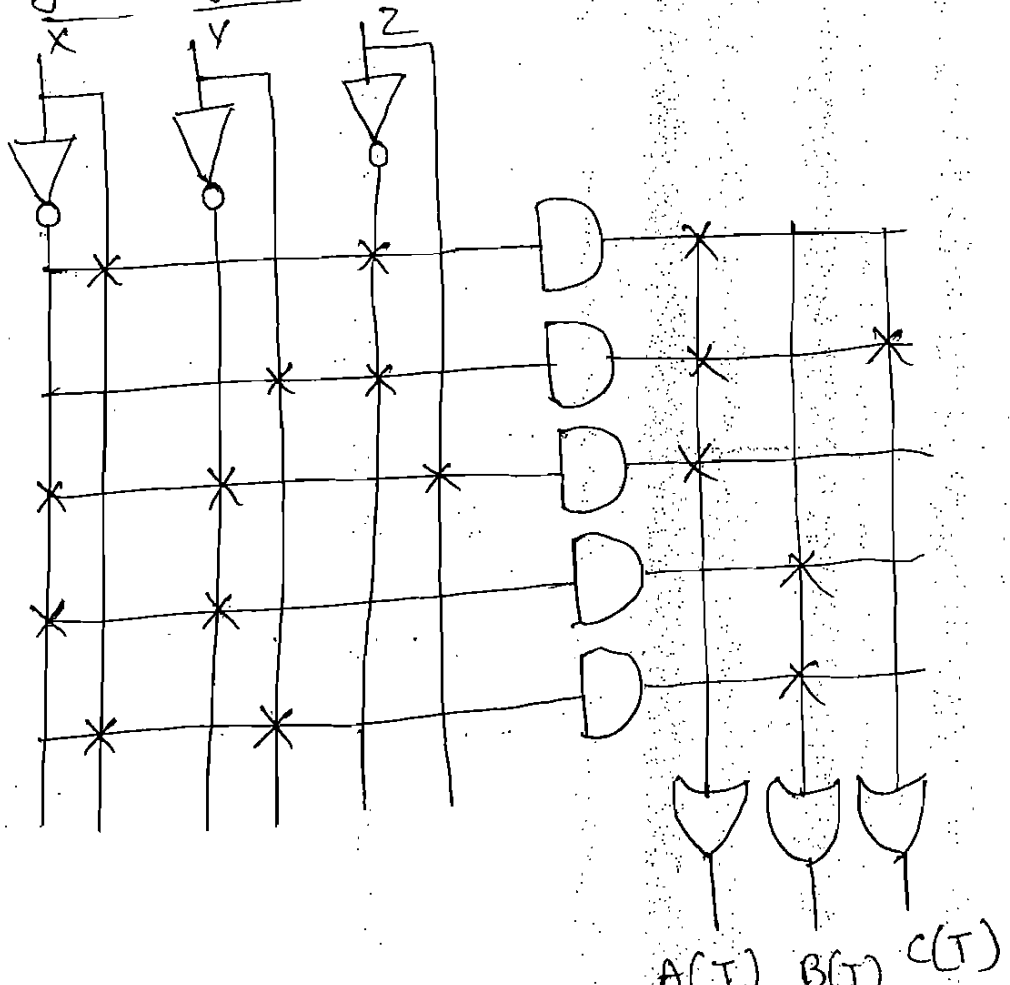
$= \bar{y} + z$.

Step 2 :- programming table.

product term	inputs x	y	z	outputs A(T)	B(T)	C(T)
$x\bar{z}$	1	–	0	1	–	–
$y\bar{z}$	–	1	0	1	–	1
$\bar{x}\bar{y}z$	0	0	1	1	–	–
$\bar{x}\bar{y}$	0	0	–	–	1	–
xy	1	1	–	–	1	–

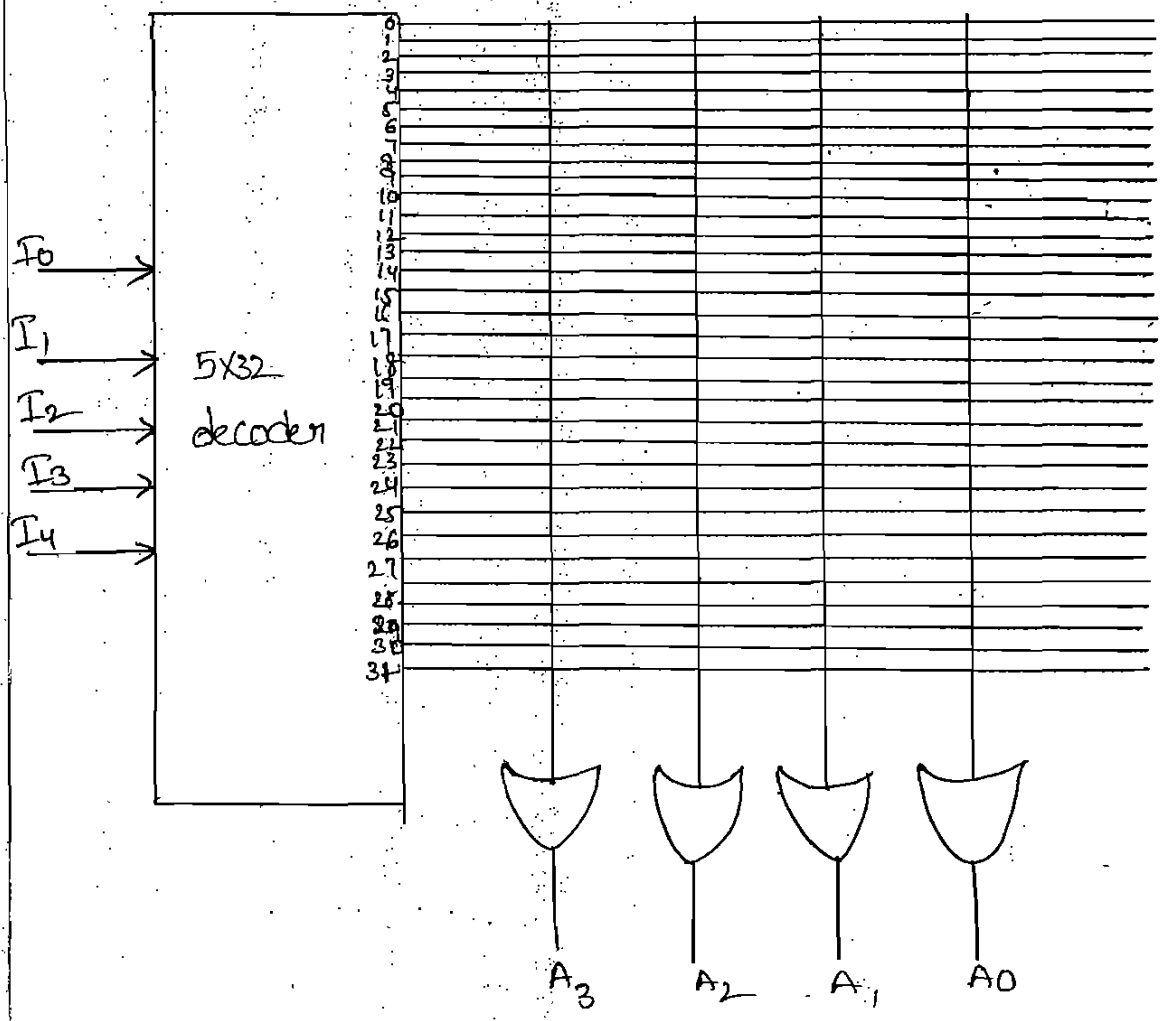
Step - 3

Logic diagram :-



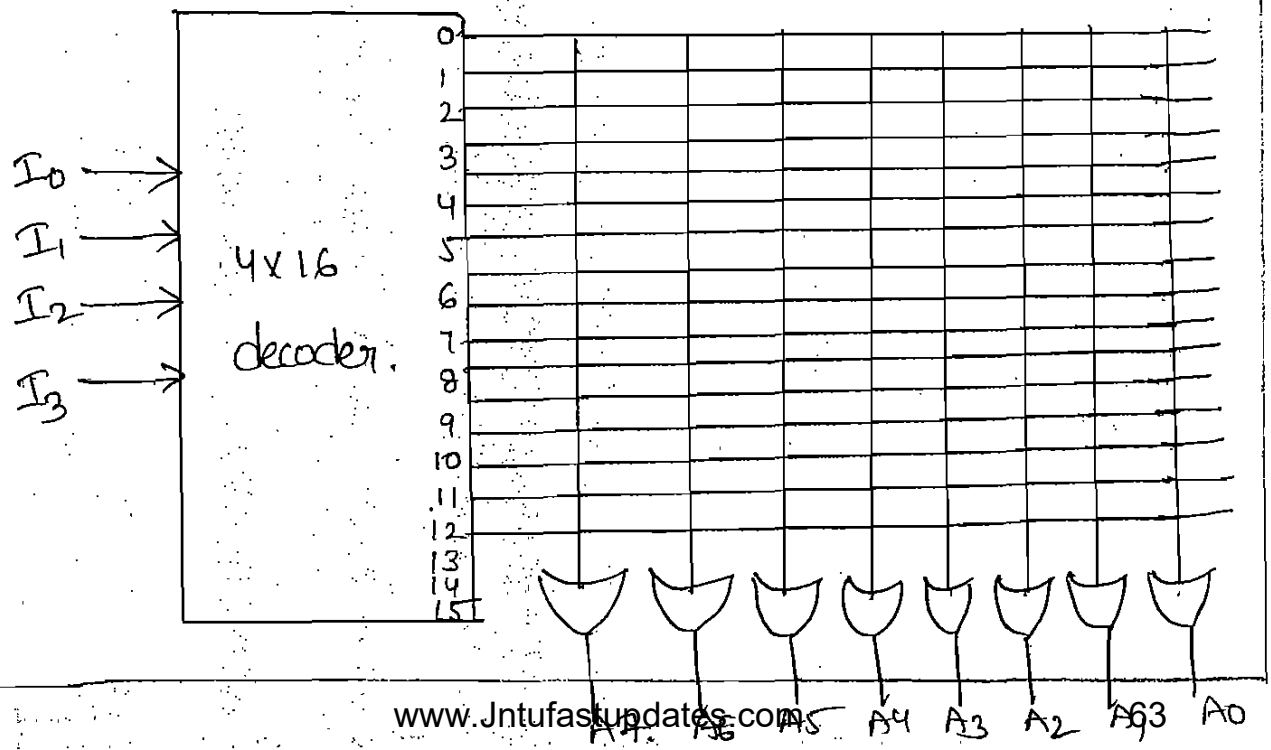
1. Give the logic implementation of a 32x4 bit ROM using a decoder of a suitable size.

A 32x4 bit ROM is to be implemented. it consists of 32 words of four bits each. There must be five input lines that form the binary numbers from 0 through 31 for the address. The five inputs are decoded into 32 distinct outputs by means of a 5x32 decoder. Each output of the decoder represent a memory address. The 32 outputs of the decoder are connected to each of the four OR gates.



32 x 4 bit ROM.

→ 16 x 8 ROM.



Comparison between PROM, PLA and PAL.

	PROM	PLA	PAL
1.	AND array is fixed and OR array is programmable.	1. Both AND and OR arrays are programmable.	1. OR array is fixed and AND array is programmable.
2.	cheaper and simple to use	2. costliest and more complex than PAL and PROM.	2. cheaper and simpler.
3.	ALL minterms are decoded.	3. AND array can be programmed to get desired minterms.	3. AND array can be programmed to get desired minterms.
4.	only Boolean functions in Standard sop form can be implemented using PROM.	4. Any Boolean function in sop form can be implemented using PROM.	4. Any Boolean function in SOP form can be implemented using PAL.

→ Implement a Binary to BCD code converter by using PAL

→ Iam taking 3-bit Binary P.

Binary B_3 B_2 B_1	BCD code C_4 C_3 C_2 C_1
0 0 0	0 0 0 0
0 0 1	0 0 0 1
0 1 0	0 0 1 0
0 1 1	0 0 1 1
1 0 0	0 1 0 0
1 0 1	0 1 0 1
1 1 0	0 1 1 0
1 1 1	0 1 1 1

→ Ram taking 4-binary.

B_4 B_3 B_2 B_1	C_8 C_7 C_6 C_5	C_4 C_3 C_2 X_1
0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 0	0 0 0 1
0 0 1 0	0 0 0 0	0 0 1 0
0 0 1 1	0 0 0 0	0 0 1 1
0 1 0 0	0 0 0 0	0 1 0 0
0 1 0 1	0 0 0 0	0 1 0 1
0 1 1 0	0 0 0 0	0 1 1 0
0 1 1 1	0 0 0 0	0 1 1 1
1 0 0 0	0 0 0 0	1 0 0 0
1 0 0 1	0 0 0 0	1 0 0 1
1 0 1 0	0 0 0 1	0 0 0 0
1 0 1 1	0 0 0 1	0 0 0 1
1 1 0 0	0 0 0 1	0 0 1 0
1 1 0 1	0 0 0 1	0 0 1 1
1 1 1 0	0 0 0 1	0 1 0 0
1 1 1 1	0 0 0 1	0 1 0 1

$C_5 = \Sigma m(10,11,12,13,14,15)$.

$C_4 = \Sigma m(8,9)$

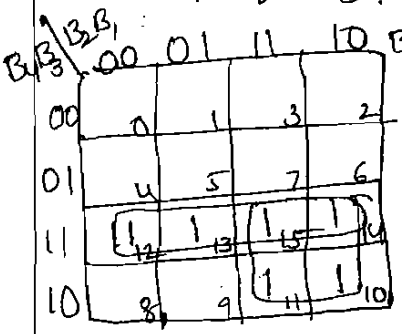
$C_3 = \Sigma m(4,5,6,7,14,15)$

$C_2 = \Sigma m(2,3,6,7,12,13)$

$C_1 = \Sigma m(1,3,5,7,9,11,13,15)$.

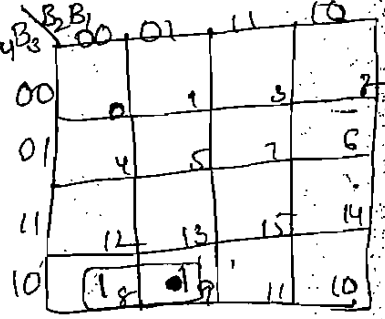
Step 1:- K-map.

K-map for C_5.



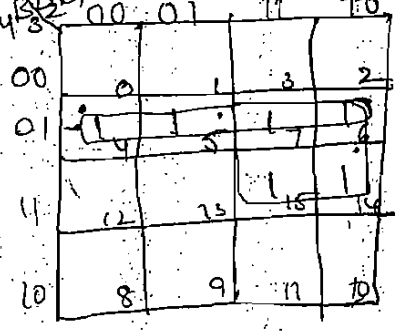
$B_4B_3 + B_4B_2$.

K-map for C_4



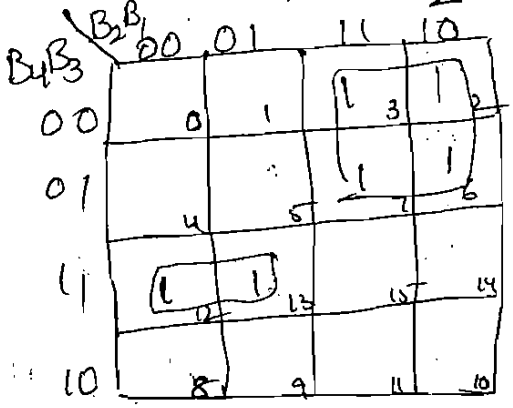
$B_4\overline{B_3}\,\overline{B_2}$

K-map for C_3



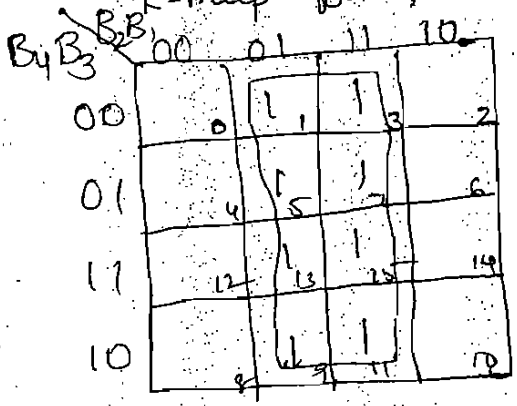
$\overline{B_4}B_3 + B_3B_2$

K-map for C_2



$B_4B_3\overline{B_2} + \overline{B_4}B_2$

K-map for C_1

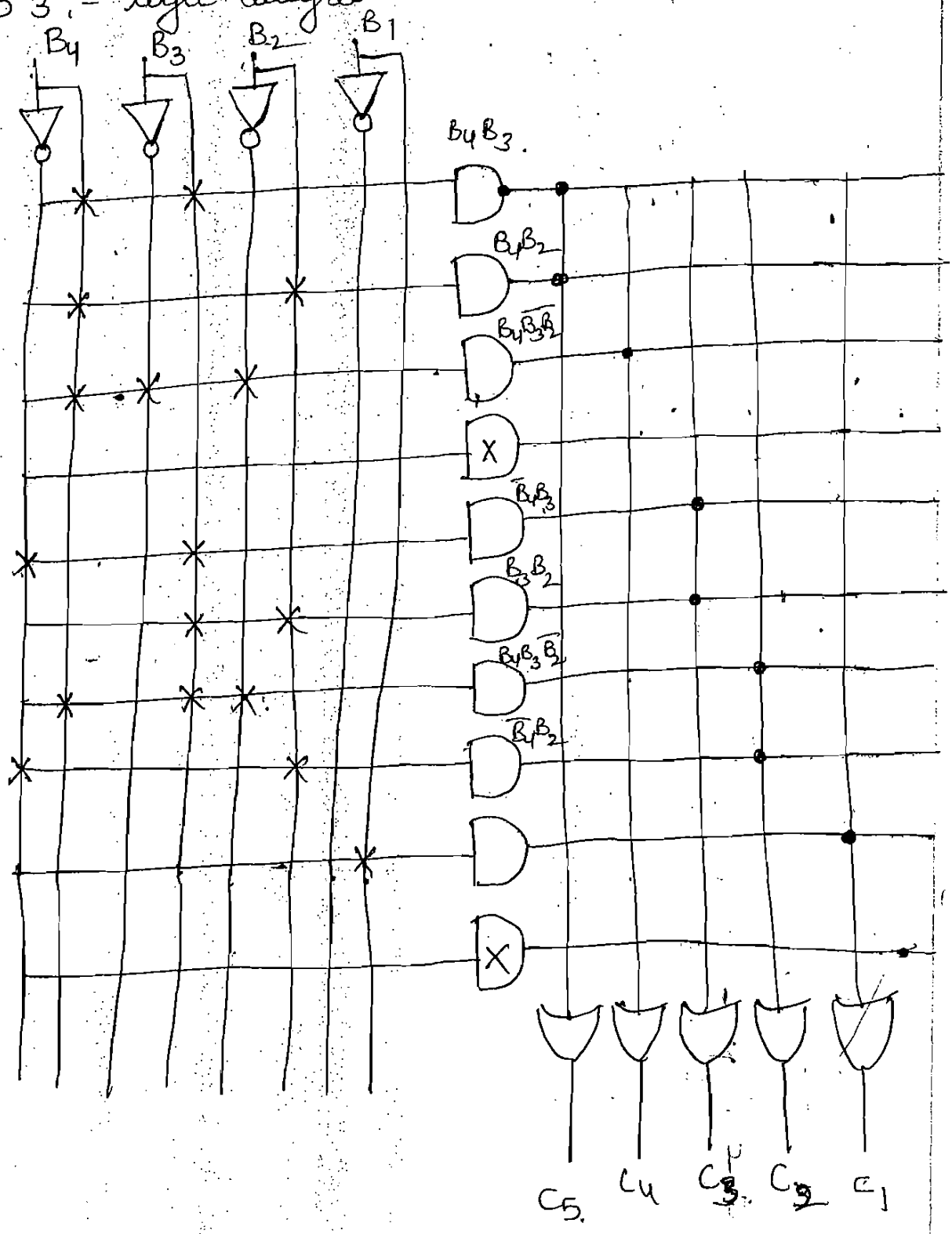


B_1

step 2:- programming table

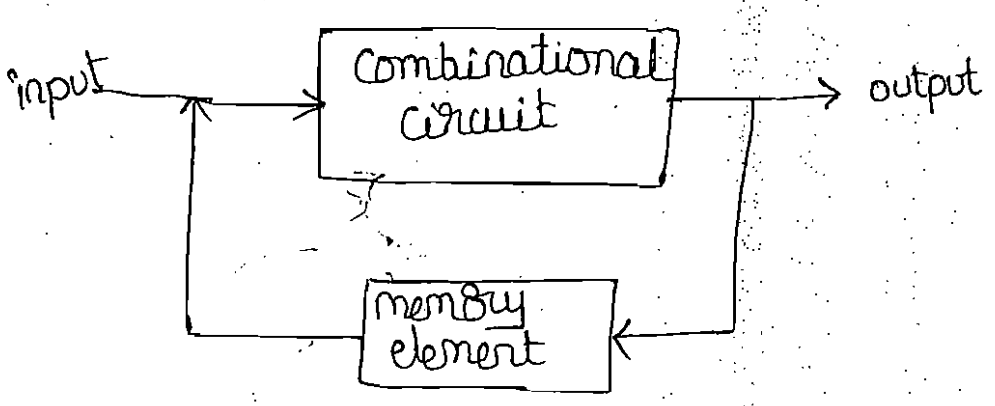
product terms	inputs B_4 B_3 B_2 B_1	output
1	1 1 – –	$C_5 = B_4B_3 + B_4B_2$
2	1 – 1 –	
3	1 0 0 –	$C_4 = B_4\overline{B_3}\,\overline{B_2}$
4	– – – –	
5	0 1 – –	$C_3 = \overline{B_4}B_3 + B_3B_2$
6	– 1 1 –	
7	1 1 0 –	$C_2 = B_4B_3\overline{B_2} + \overline{B_4}B_2$
8	0 – 1 –	
9	– – – 1	$C_1 = B_1$
10	– – – –	

Step 3 :- logic diagram.



Sequential circuits I.

In sequential logic circuits, the output is a function of the present inputs as well as the past inputs and outputs. sequential circuit include memory element to store the past data. The flip-flop is a basic element of sequential logic circuits.



Block diagram of sequential circuit.

There are two types of sequential circuits.

→ Synchronous circuit :- The sequential circuits which are controlled by a clock are called Synchronous sequential circuits. These circuits get actuated only clock signal is present.

→ A synchronous circuit :- The sequential circuits which are not controlled by a clock are called a synchronous sequential circuits, that is the sequential circuits in which events can take place any time the inputs are applied are called A synchronous sequential circuits.

Comparison between Synchronous & Asynchronous sequential ckt

Synchronous Sequential circuit	Asynchronous sequential circuit.
1. In synchronous circuits, the change in input signals can affect memory elements upon activation of clock signal.	1. In asynchronous circuits, change in input signals can affect memory elements at any instant of time.
2. In synchronous circuits, memory elements are clocked FF's.	2. In asynchronous circuits, memory elements are either unclocked FFs or time delay elements.
3. The maximum operating speed of clock depends on time delays involved.	3. Since the clock is not present, asynchronous circuits can operate faster than synchronous circuits.
4. They are easier to design.	4. more difficult to design.

→ Latches & Flip flops :-

→ The most important memory element is the flip-flop which is made up of an assembly of logic gates.

→ even though a logic gate by itself has no storage capability, several logic gates can be connected together in ways that permit information to be stored.

→ Flip-flops are the basic building blocks of most sequential circuits. Actually, flip-flop is an one-bit

memory device and it can store either 1 & 0.

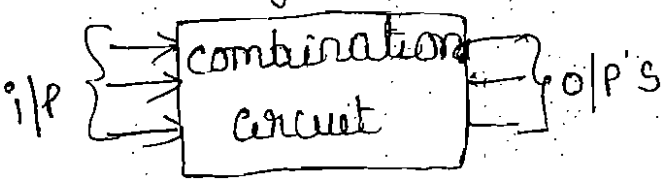
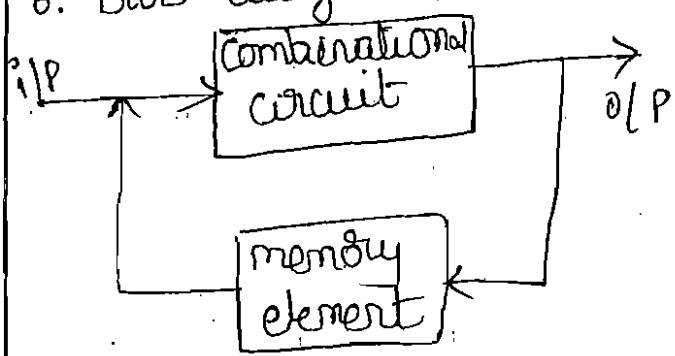
→ Latch is a sequential device that checks all its inputs continuously and changes its outputs accordingly at any time independent of a clock signal.

→ It refers to non-clocked flip-flops, because these flip flops 'latch on' to a 1 & a 0 immediately upon receiving the input pulse.

→ Difference between Latches & flip flops.

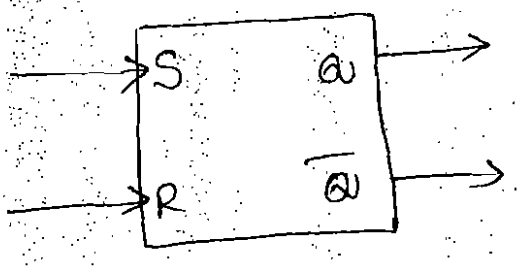
Latch	Flip-flop.
1. A latch is an electronic sequential logic circuit used to store information in an asynchronous arrangement.	1. A flip flop is an electronic sequential logic circuit used to store information in an asynchronous arrangement.
2. one latch can store one bit information, but output state changes only in response to data input.	2. one flip-flop can store one bit-data, but output state changes with clock pulse only.
3. latch is an asynchronous device and it has no clock input.	3. Flip-flop has clock input and its output is synchronized with clock pulse.
4. Latch holds a bit value and it remains constant until new inputs force it to change.	4. Flip flops holds a bit value and it remains constant until a clock pulse is received.
5. Latches are level-sensitive and the output tracks the input when the level is high. Therefore as long as the level is logic level 1, the output can change if the input changes.	5. Flip flops are edge-sensitive They can store the input only when there is either a rising & falling edge of the clock.

Difference between combinational, sequential circuits.

Combinational circuit	Sequential circuits.
1. The digital logic circuit whose outputs can be determined using the logic function of current state input are combinational logic circuits.	1. The digital logic circuits whose outputs can be determined using the logic function of current state inputs and past state inputs are called as sequential logic circuits.
2. It contains no memory element	2. It contains memory elements.
3. It's behaviour is described by the set of output functions.	3. It's behaviour is described by the set of next state functions and the set of output functions.
4. The combinational logic circuits are independent of the clocks.	4. The maximum sequential logic circuits are uses a clock for triggering the flip-flop operation
5. The combinational digital logic circuit don't require any feed back.	5. The sequential digital logic circuits utilize the feedbacks from outputs to inputs.
6. combinational circuits are easy to design.	6. Sequential circuits are complex to design.
7. combinational circuits are faster because the delay between the input and the output is due to propagation delay of gates only.	7. Sequential circuits are slower than combinational circuits.
8. Block diagram 	8. Block diagram. 

S-R latch :-

The S-R latch has two inputs, namely SET(S) and RESET(R), and two outputs Q and $\bar{Q}$, where $\bar{Q}$ is the complement of Q.



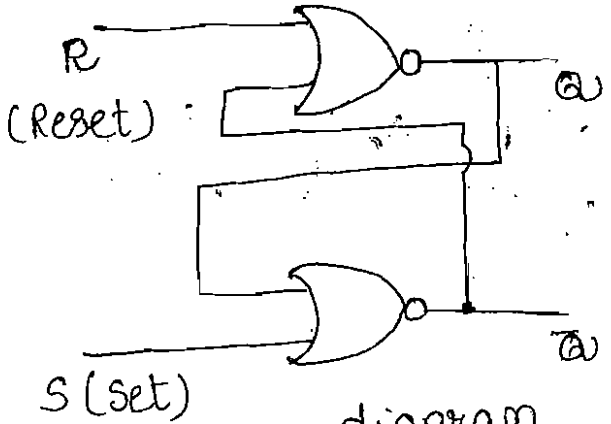
Logic symbol of S-R latch

S-R latch using NOR Gates :- [active-high S-R latch].

The logic diagram of the S-R latch composed of two cross-coupled NOR gates. S and R are two inputs of S-R latch.

→ S stands for set, it means that when S is 1, it stores 1.

→ R stands for Reset, and if R=1, latch Reset and it's output will be 0. This circuit is called as NOR gate latch or S-R latch.



logic diagram.

S	R	Q_n	Q_{n+1}	State
0	0	0	0	NO change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	indetermined (invalid).
1	1	1	X	

Truth table.

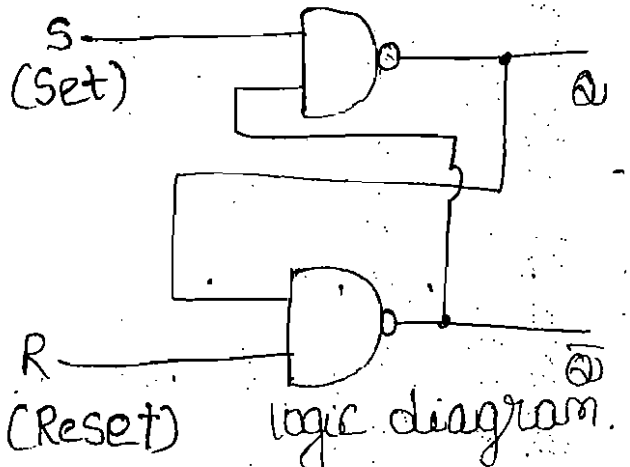
Simplified truth table.

S	R	Q_{n+1}	State
0	0	Q_n	NO change
0	1	0	Reset
1	0	1	Set
1	1	X	invalid state

1. SET = 0, RESET = 0 : This is the normal resting state of the NOR latch and it has no effect on the output state. Q and $\bar{Q}$ will remain in whatever state they were prior to the occurence of this input condition.
2. SET = 0, RESET = 1, This will always reset Q = 0, where it will remain even after RESET returns to 0.
3. SET = 1, RESET = 0, This will always set Q = 1, where it will remain even after SET returns to 0.
4. SET = 1, RESET = 1. This condition tries to SET and Reset the latch at the same time, and it produces $Q = \bar{Q} = 0$. if the inputs are returned to zero simultaneously, the resulting output state is erratic and unpredictable. This input condition should do not be used. it is indetermined state or invalid state.

S-R latch using NAND Gates :- (active low S-R latch)

→ The logic diagram of the S-R latch composed of two cross-coupled NAND gates.



Logic diagram.

S	R	Q_{n+1}	State
0	0	X	invaild
0	1	1	Set
1	0	0	Reset
1	1	Q_n	N.C

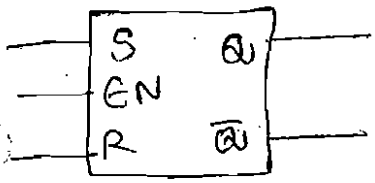
← Simplified truth table.

S	R	Q_n	Q_{n+1}	State
0	0	0	X	indetermined (invaild)
0	0	1	X	
0	1	0	1	Set
0	1	1	1	
1	0	0	0	Reset
1	0	1	0	
1	1	0	0	NO change
1	1	1	1	

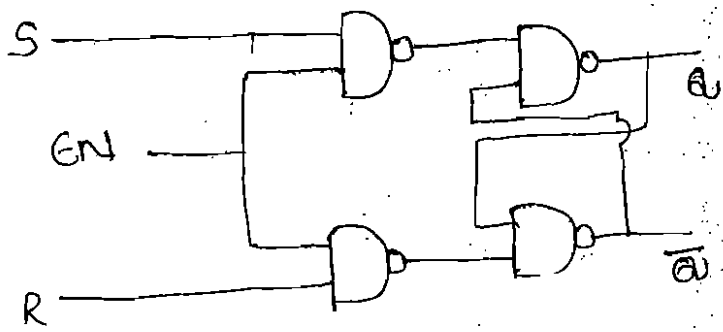
Truth table.

Gated latches :-

The gated S-R latch :- The output can change state any time the input conditions are changed, so they are called Asynchronous latches. A gated S-R latch requires an Enable (EN) input. Its S and R inputs will control the state of latch only when the Enable is high. When the Enable is low, the inputs become ineffective and no change of state can take place.

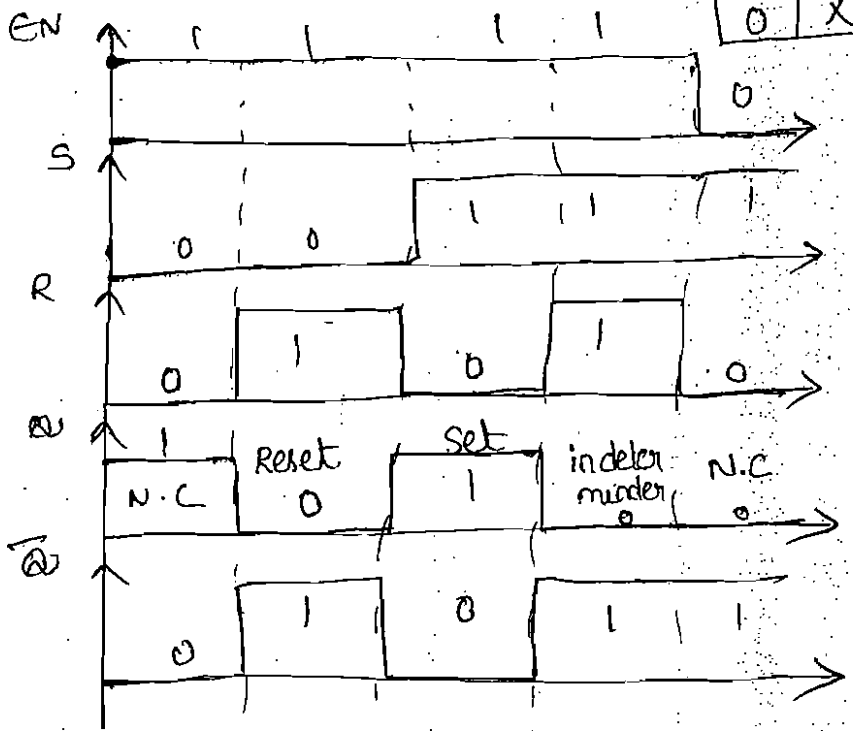


Logic symbol.



Logic diagram.

EN	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	NO change
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	indeterminate (invalid)
1	1	1	1	X	
0	X	X	0	0	NO change.
0	X	X	1	1	



Waveforms for Gated S-R latch.

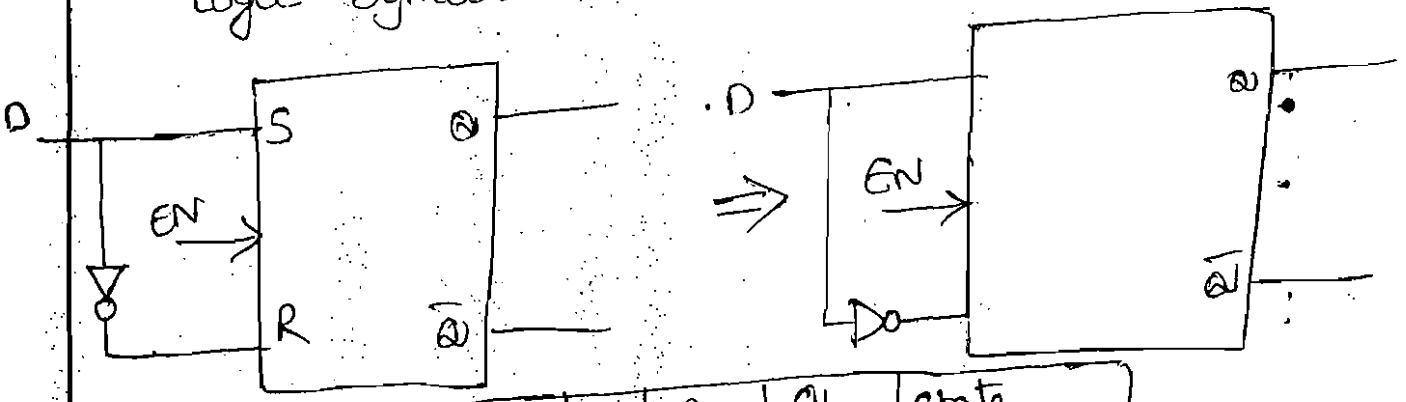
The Gated D-latch; - In many applications, it is not necessary to have separate S and R inputs to a latch. If the input combinations S=R=0 and S=R=1 are never needed, the S and R are always the complement of each other. So, to construct a latch with a single input (S) and obtain the R input by inverting it. This single input is labelled D (for data). and the device is called a D-latch.

→ when D=1, S=1 and R=0, causing the latch to set when Enabled. when D=0, S=0 and R=1, causing the latch to Reset when enabled. when En is low, the latch is ineffective, and any change in the value of D input does not affect the output at all.

→ when, En is high, a low D-input makes Q low, i.e resets the latch and high D input makes a Q high, that is sets the latch.

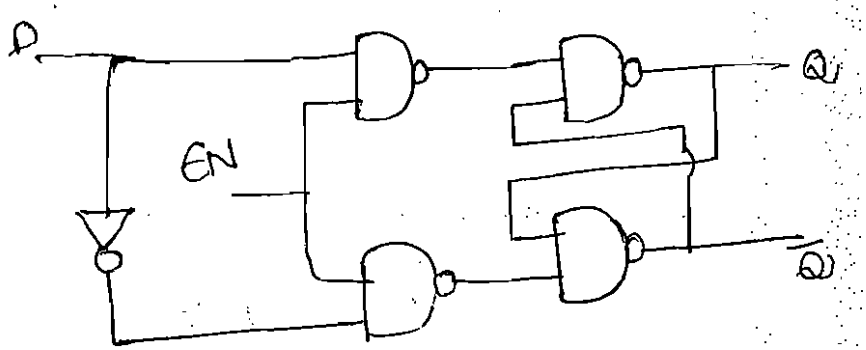
→ The output Q follows the D-input when En is high. So this latch is said to be transparent.

Logic Symbol

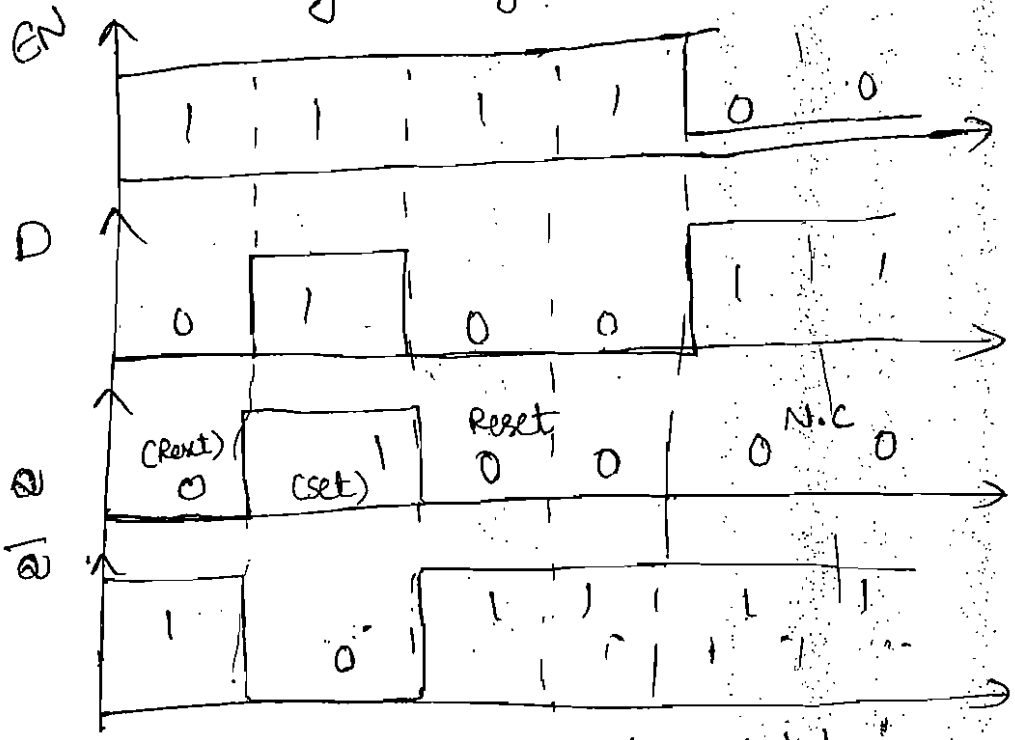


EN	D	Q_n	Q_{n+1}	State
1	0	0	0	Reset
1	0	1	0	
1	1	0	1	Set
1	1	1	1	
0	X	0	0	NO change (NC)
0	X	1	1	

Truth table



Logic diagram.



waveforms for Gated D-latch.

Flip-flops :-

Types of flip-flops :-

→ S-R flip-flop

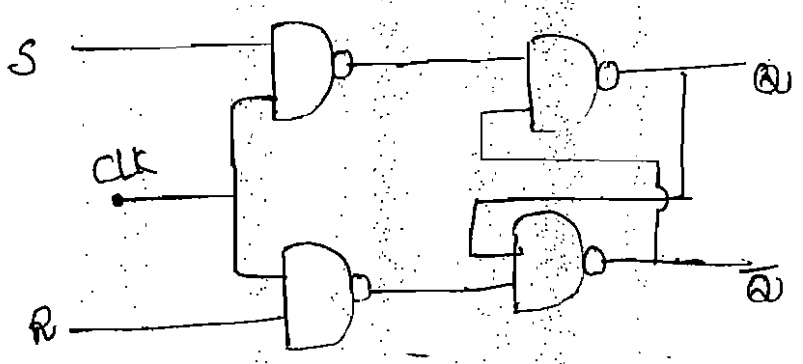
→ D flip-flop

→ J-K flip-flop

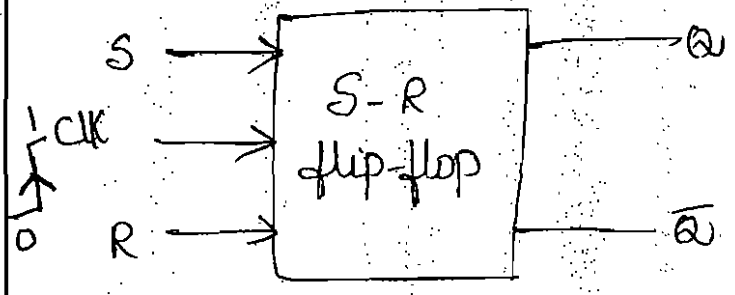
→ T flip-flop.

S-R flip-flop :-

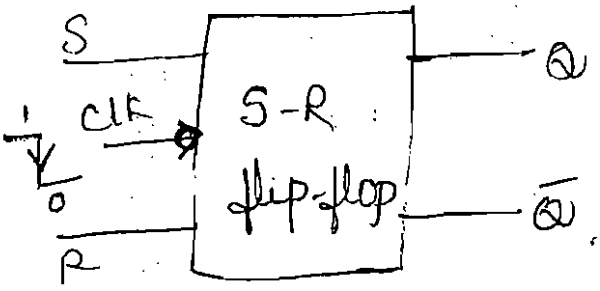
Logic diagram



Symbol of +ve edge trigger.



Symbol of -ve edge trigger

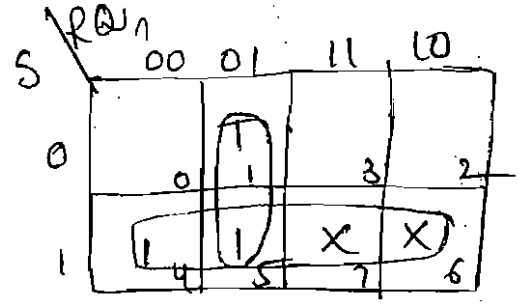


Truth table :-

Clk	S	R	Q_n	Q_{n+1}	State
↑	0	0	0	0	No change
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	set
↑	1	0	1	1	
↑	1	1	0	X	invalid State
↑	1	1	1	X	
0	X	X	0	0	No change
0	X	X	1	1	

Characteristic equation :- The characteristic equation of a flip-flop is the equation expressing the next state of a flip-flop in terms of its present state and present excitations. To obtain the characteristic equation of a.

flip-flop write the excitation requirements of the flip-flop, draw a K-map for the next state of the flip-flop in terms of its present state and inputs and simplify it.



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

Truth table:-

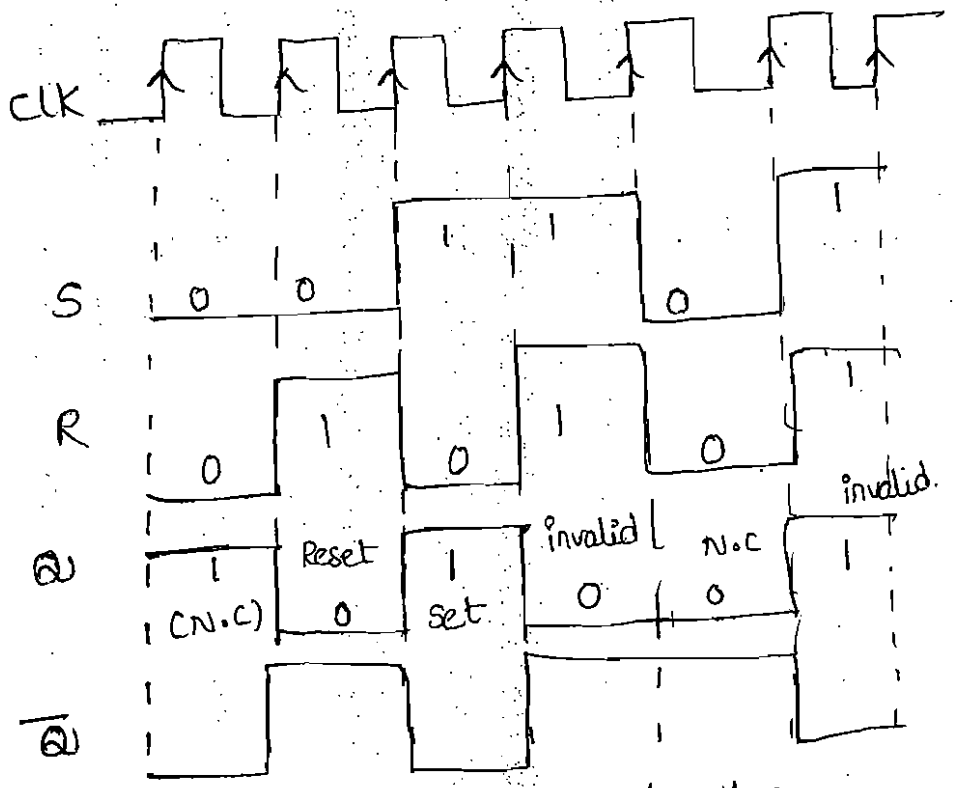
S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	X

Excitation table:- A table which lists the present state, the next state and the excitations of a flip-flop is called the excitation table.

→ A table which indicates the excitations required to take the flip-flop from the present state to the next state.

Present State Q_n	next state Q_{n+1}	Required S	Required R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

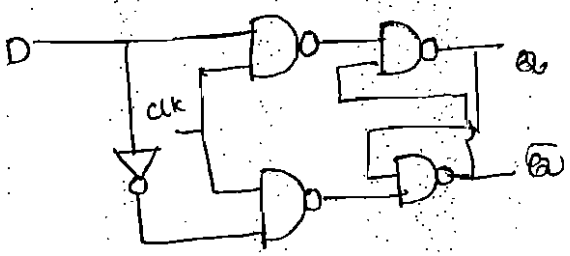
Timing diagram :-



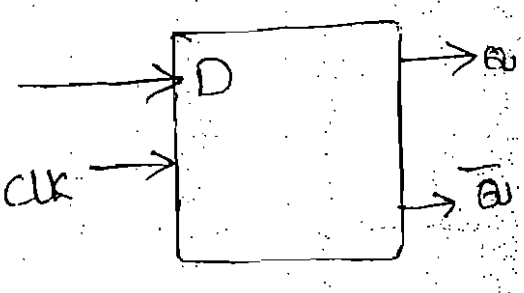
waveforms for S-R flip-flop.

D- flip-flop :-

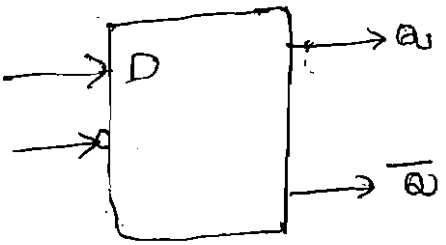
logic diagram :-



Symbol of +ve edge trigger



Symbol of -ve trigger



Truth table :-

D	Q_{n+1}
0	0
1	1

characteristic Table.

Clk	D	Q_n	Q_{n+1}	State
↑	0	0	0	Reset
↑	0	1	0	
↑	1	0	1	Set
↑	1	1	1	
↑	X	0	0	NO change
↑	X	1	1	

characteristic equation :-

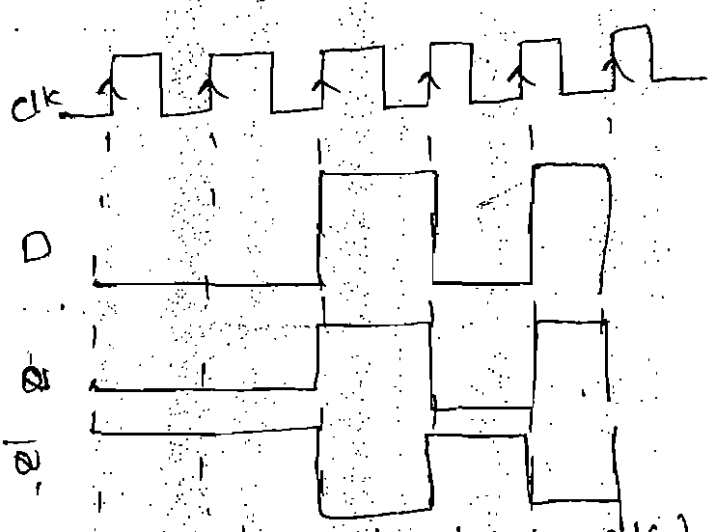
D \ Q_n	0	1
0	0	1
1	1 2	1 3

$Q_{n+1} = D$.

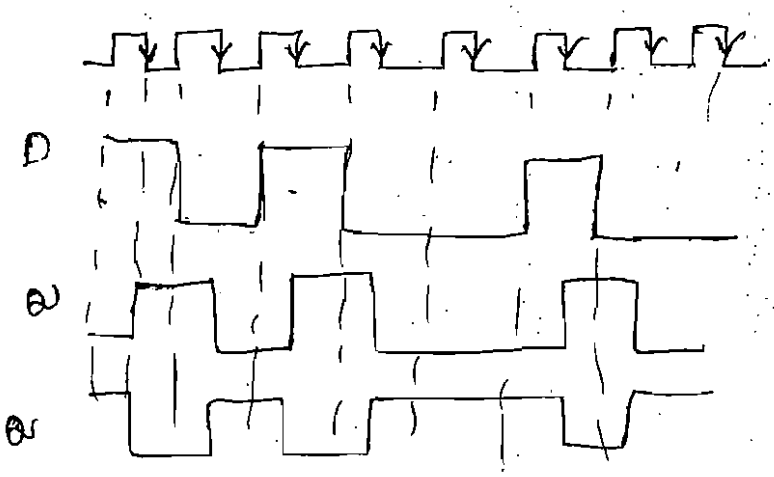
Excitation table :-

Present state Q_n	Next state Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

waveforms



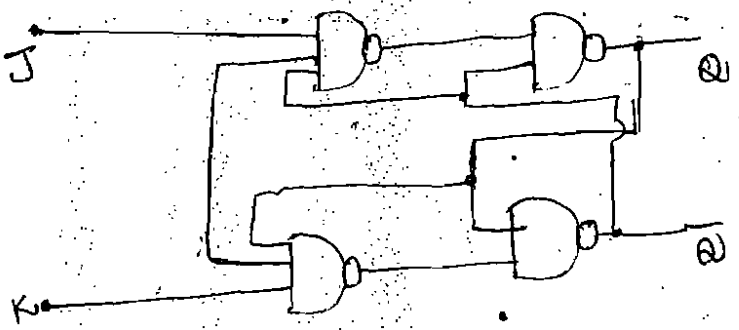
(Positive - edge trigger clk).



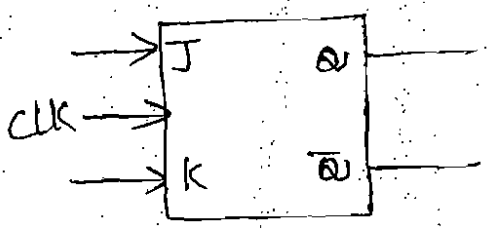
(negative - edge trigger clk).

(J-K - flip flops).

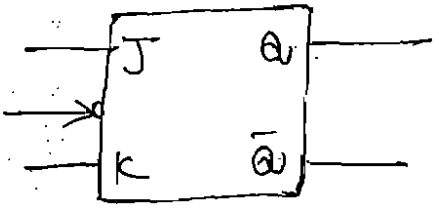
logic diagram



logic symbol.



(positive-edge)



(negative-edge)

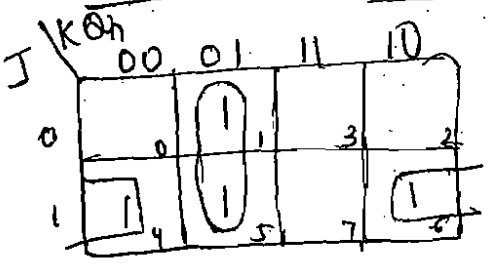
Truth table:-

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

characteristic table

Clk	J	K	Q_n	Q_{n+1}	State
↓	0	0	0	0	No change
↓	0	0	1	1	
↓	0	1	0	0	Reset
↓	0	1	1	0	
↓	1	0	0	1	Set
↓	1	0	1	1	
↓	1	1	0	1	Toggle
↓	1	1	1	0	
0	X	X	0	0	No change
0	X	X	1	1	

Characteristic equation

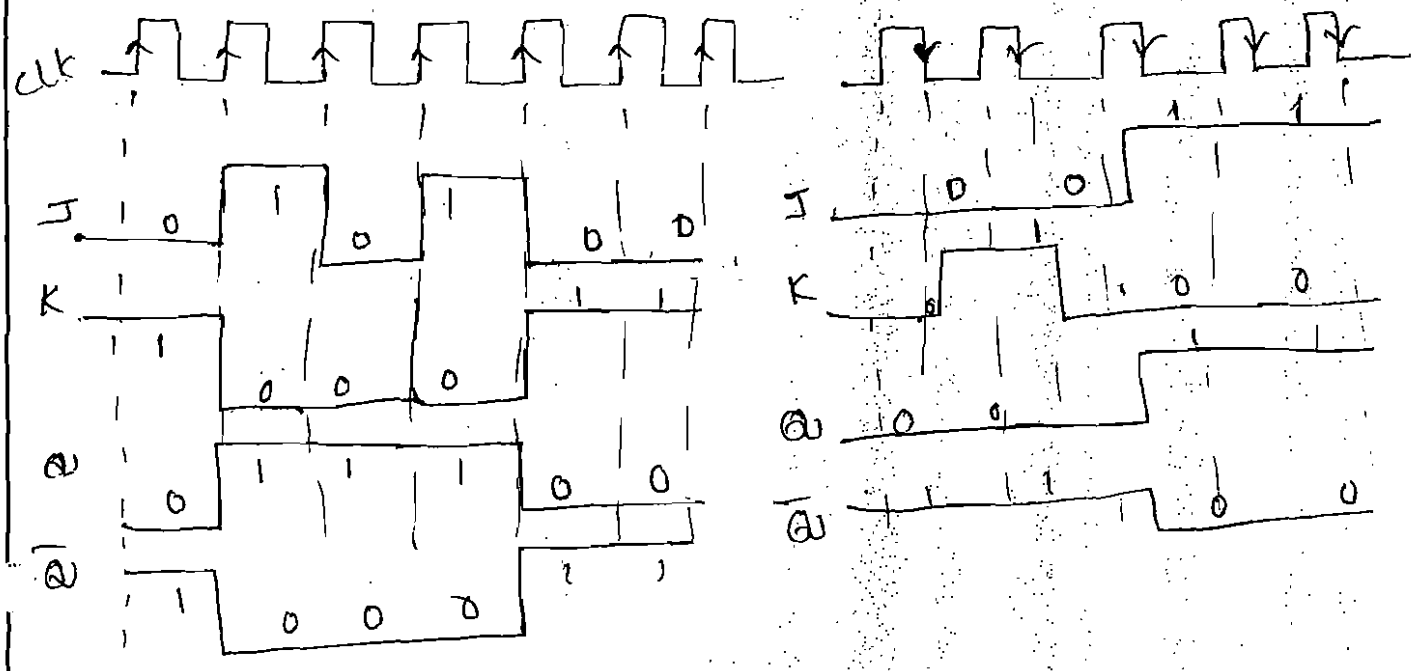


$Q_{n+1} = \bar{K}Q_n + J\bar{Q}_n$

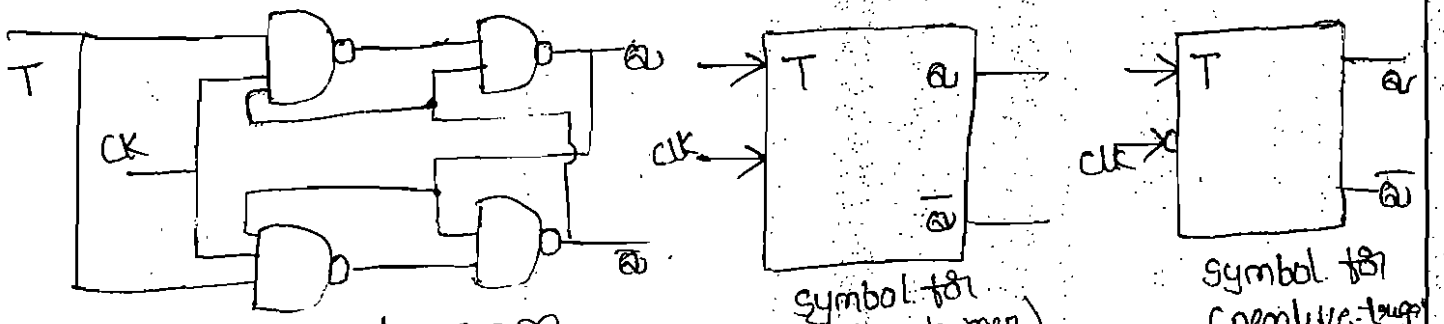
excitation table

Present state	next state	inputs J	inputs K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Wave-forms for Positive-edge triggering | negative-edge triggering



T-flip flop :-



logic diagram. Symbol for (Positive-trigger) Symbol for (negative-trigg

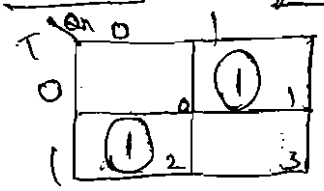
Truth table

T	Q_{n+1}
0	Q_n
1	0

Characteristic table

clk	T	Q_n	Q_{n+1}	State
↑	0	0	0	NO change
↓	0	1	1	
↑	1	0	1	Toggle
↓	1	1	0	
0	x	0	0	NO change
0	x	1	1	

Characteristic equation

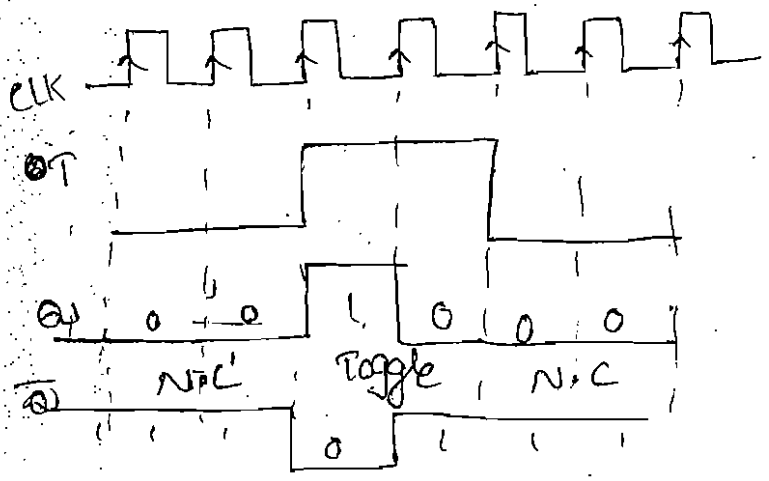


$Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n$.

Excitation table :-

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

waveforms.



Race - Around Condition :-

If the width of the clock pulse tp is too long, the state of the flip-flop will keep on changing from 0 to 1, 1 to 0, 0 to 1 and so on, and at the end of the clock pulse, its state will be uncertain. This phenomenon is called the race around condition. The outputs Q and $\bar{Q}$ will change on their own if the clock pulse width tp is too long compared with the propagation delay t of each NAND Gate.

$$t_p >> t$$

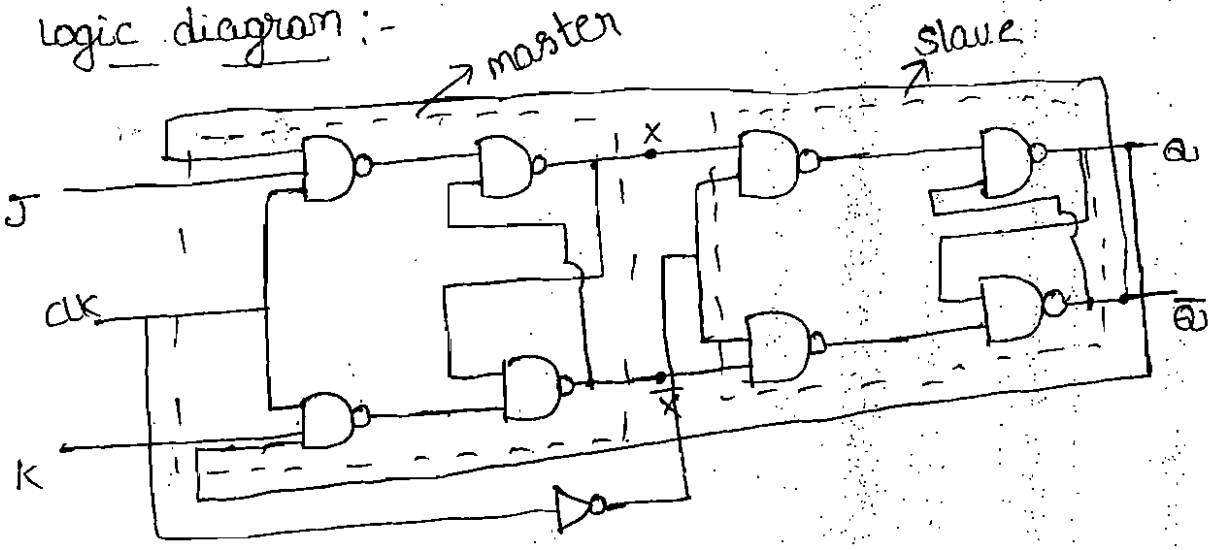
$t_p \rightarrow$ pulse width

$t \rightarrow$ propagation delay.

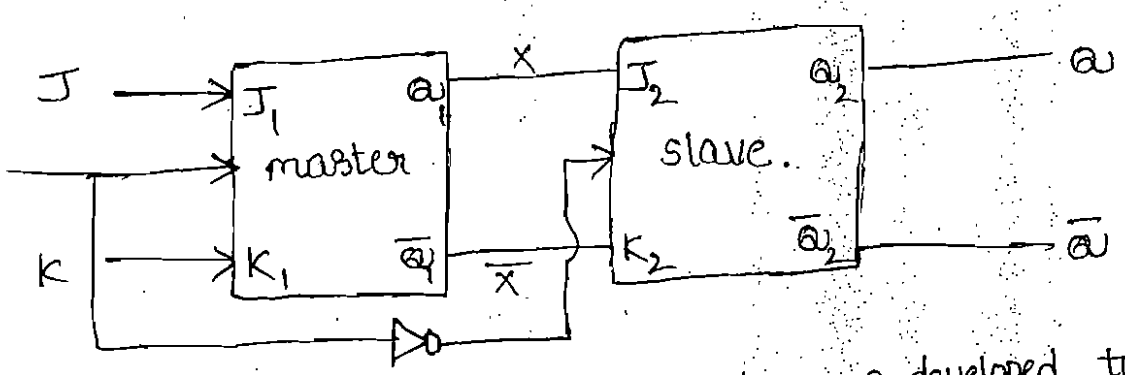
The clock pulse width should be such as to allow only one change to complement the state and not too long to allow many changes resulting in uncertainty about the final state. This is a stringent requirement which cannot be ensured in practice. This problem is eliminated using master-slave flip-flop or edge triggered flip-flop.

master-slave J-K flip-flop :-

Logic diagram :-



Symbol ;-



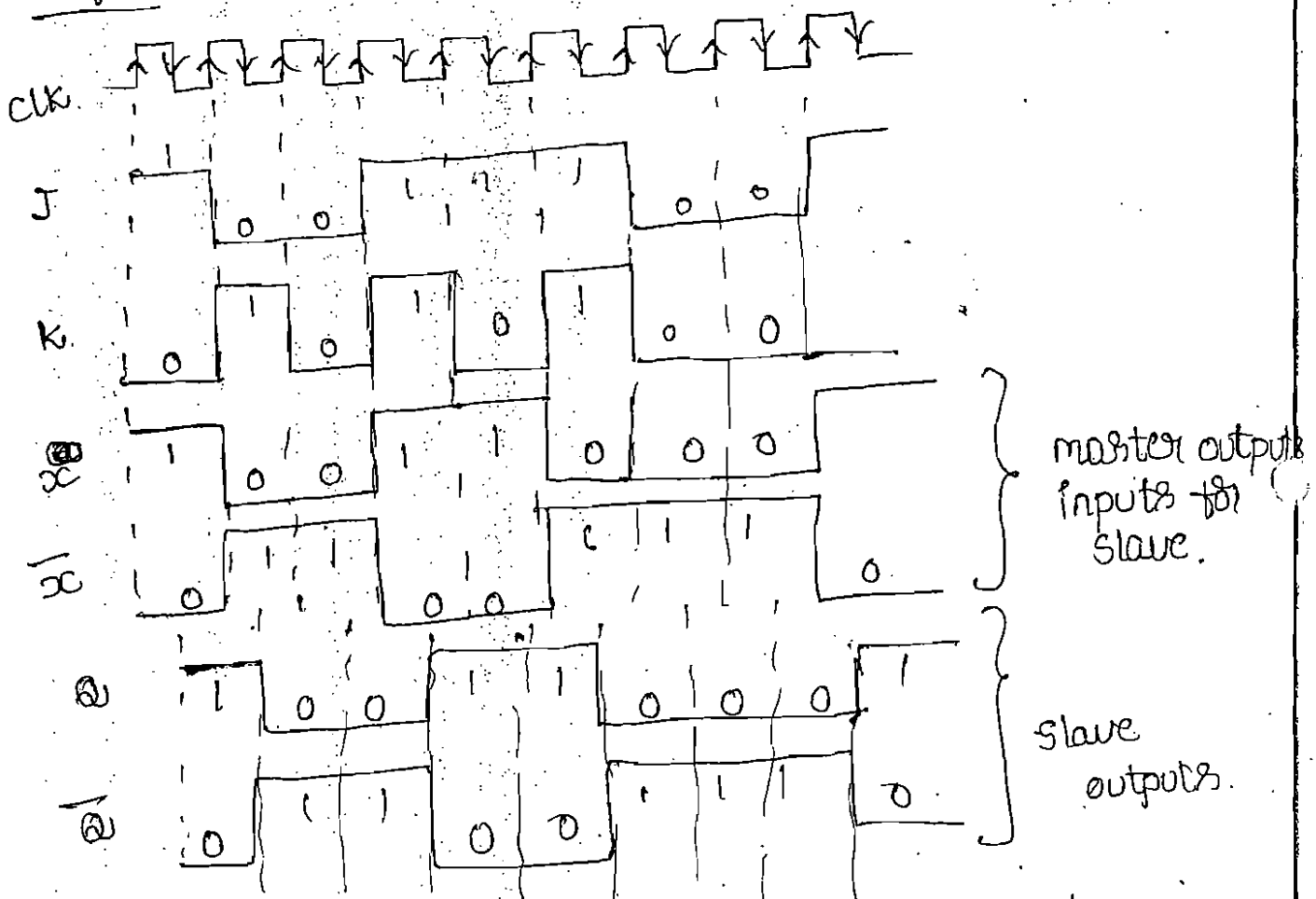
The master-slave flip flop was developed to make the synchronous operation more predictable, that is, to avoid the problems of logic race in clocked flip-flops. This improvement is achieved by introducing a known time delay between the time that the flip-flop responds to a clock pulse and the time the response appears at its output. A master-slave flip-flop is also called a pulse-triggered flip-flop because the length of the time required for its output to change state equals the width of one clock pulse.

In master-slave J-K flip-flop actually contains two flip-flops - a master flip-flop and a slave flip-flop. The control inputs are applied to the master flip-flop and master output is given as input to the

slave flip-flop. on the rising edge of the clock pulse, the levels on the control inputs are used to determine the output of the master. on the falling edge of the clock pulse, the state of the master is transferred to the slave, whose outputs are guarded

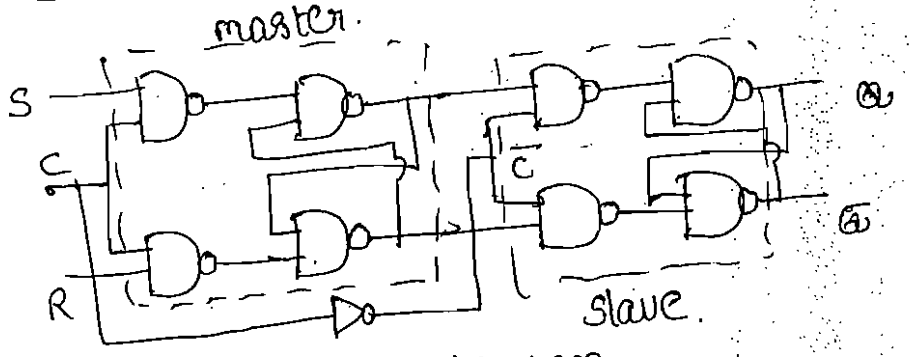
The master-slave flip-flops function very much like the negative-edge triggered flip-flops except for one more disadvantage The control inputs must be held stable while CLK is high, otherwise an unpredictable operation may occur. This problem with the master-slave flip-flop is overcome with an improved master-slave version called the master-slave with data lock-out

waveforms :-



in slave accept the negative edge triggered signal only. master accept the positive-edge triggered signal only.

master-slave S-R flip-flop:-

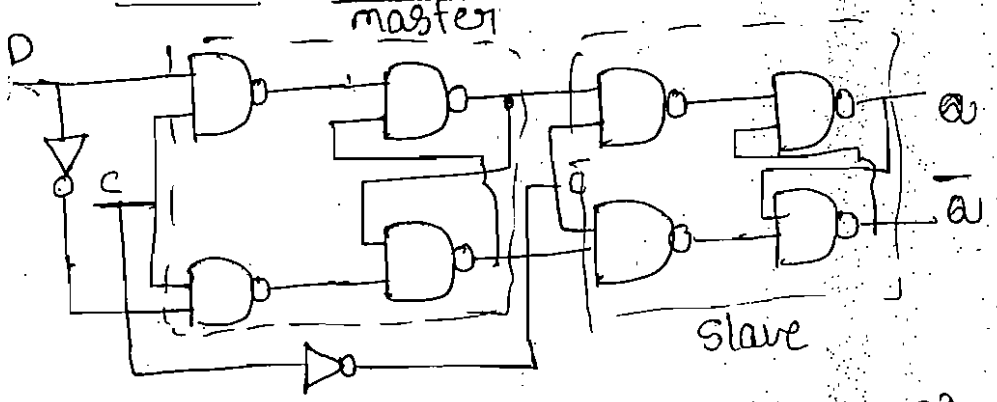


Logic diagram

S	R	clk	Q	State
0	0	⎍	Q_0	N.C
0	1	⎍	0	Reset
1	0	⎍	1	set
1	1	⎍	?	invalid

Truth table

master-slave D flip flop:-



Truth table

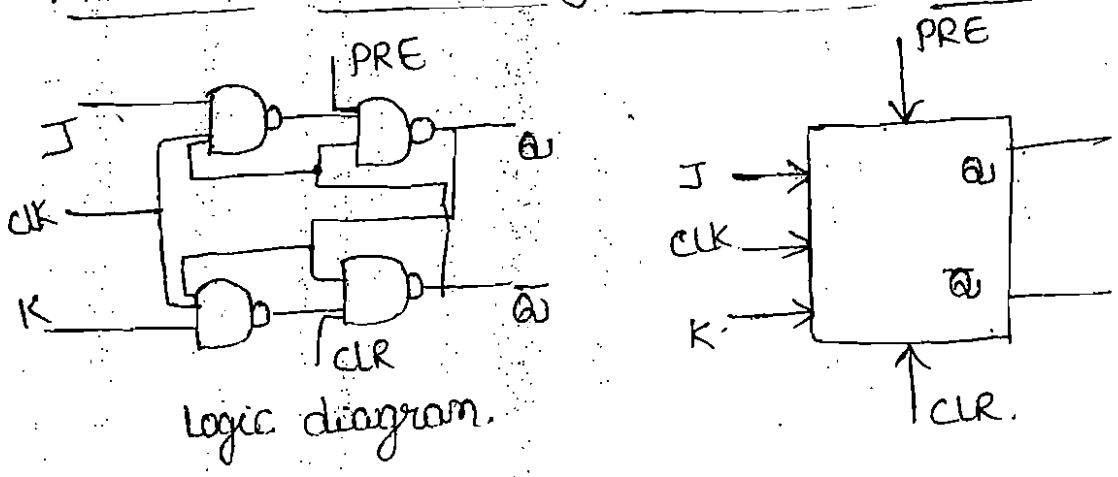
D	clk	Q	State
0	⎍	0	Reset
1	⎍	1	set

→ Asynchronous inputs (PRESET and CLEAR).

The S-R, D, and J-K inputs are called synchronous inputs, because their effect on the flip-flop output is synchronized with the clock input.

most IC flip-flops also have one or more asynchronous inputs. These Asynchronous inputs affect the flip-flop output independently of the Synchronous inputs and the clock input These as Synchronous inputs can be used to SET the flip-flop to the 1 State or RESET the flip-flop to the 0 State at any time regardless of the conditions at the other inputs. They are normally labelled PRESET or direct SET or DC SET, and CLEAR or direct RESET or DC CLEAR.

J-K flip-flop with Active-high Asynchronous inputs :-



Logic diagram.

→ When PRE = 0, CLR = 0 then DC SET = 1 and DC CLEAR = 1, The Asynchronous inputs are inactive and the flip-flop responds freely to J, K and CLK inputs in the normal way. In other words, the clocked operation can takes place.

→ when PRE = 0, CLR = 1 then DC SET = 0 and DC CLEAR = 1 The Jx

→ when PRE = 0, CLR = 0 then Asynchronous inputs are inactive and the flip-flop responds freely to J, K and CLK inputs.

→ when PRE = 0, CLR = 1 then Asynchronous input clear is active then flip flop output is '0'.

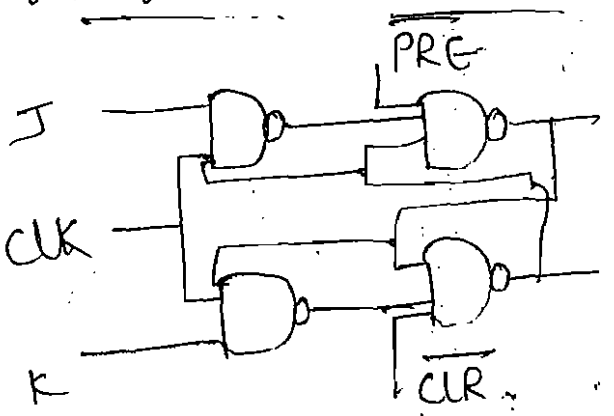
→ when PRE = 1, CLR = 0 then Asynchronous input PRESET is active the flip flop output is '1'.

→ when PRE = 1, CLR = '1'. This condition should not be used since it can result in an invalid state.

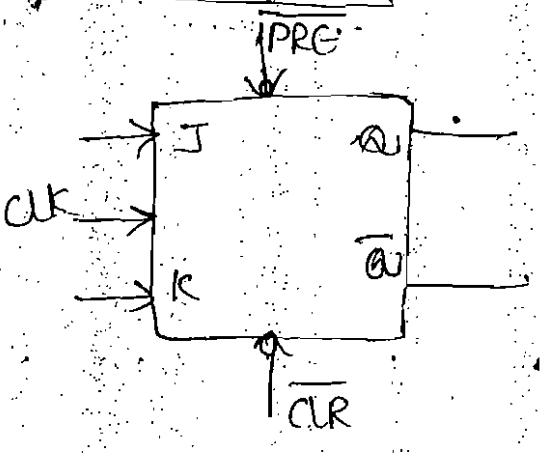
DC SET (PRE)	DC RESET (CLR)	FF response
0	0	clock operation
0	1	Q = 0
1	0	Q = 1
1	1	Not used.

Truth table

J-K flip-flop with Active-Low Asynchronous inputs



Logic diagram



→ when $\overline{PRE} = 0$ and $\overline{CLR} = 0$, this condition should not be used. since it can result in an invalid state.

→ when $\overline{PRE} = 0$ and $\overline{CLR} = 1$, then Asynchronous input $\overline{PRESET}$ is active then output is '1'.

→ when $\overline{PRE} = 1$ and $\overline{CLR} = 0$, then Asynchronous input $\overline{CLEAR}$ is active then output is '0'.

→ when $\overline{PRE} = 1$ and $\overline{CLR} = 1$, Then A synchronous inputs are inactive and the flip responds freely to J, K and CLK inputs.

DC SET ($\overline{PRE}$)	DC RESET ($\overline{CLR}$)	FF response
0	0	not used
0	1	Q = 1
1	0	Q = 0
1	1	clock operation

Truth table

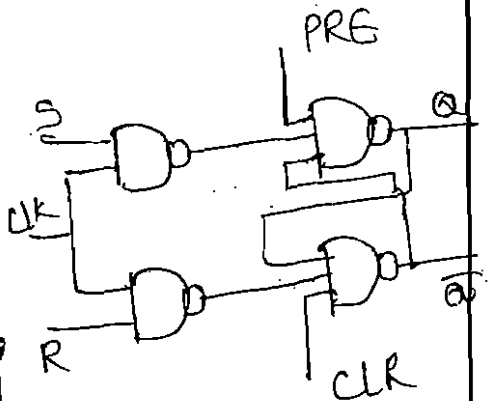
Truth table for J-K flip-flop (both Asynchronous & Synchronous inputs).

PRE	CLR	CLK	J	K	Q	$\bar{Q}$
1	1	X	X	X	invalid.	
1	0	X	X	X	1	0
0	1	X	X	X	0	1
0	0	↓	0	0	0	1
0	0	↓	0	1	0	1
0	0	↓	1	0	1	0
0	0	↓	1	1	0	1

X - don't care means either '0' or '1'.

Similarly S-R flip-flop

PRE	CLR	CLK	S	R	Q	$\bar{Q}$
1	1	X	X	X	invalid	
1	0	X	X	X	1	0
0	1	X	X	X	0	1
0	0	↓	0	0	0	1
0	0	↓	0	1	0	1
0	0	↓	1	0	1	0
0	0	↓	1	1	invalid	
0	0	0	X	X	N.C.	

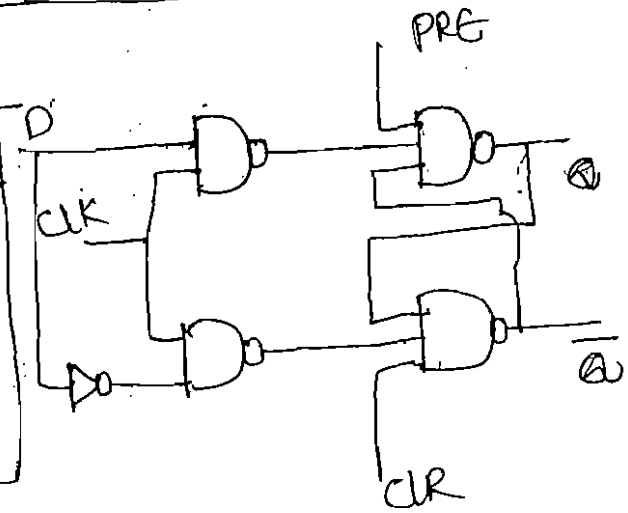


logic diagram

Similarly for D-flip-flop

PRE	CLR	CLK	D	Q	$\bar{Q}$
1	1	X	X	invalid	
1	0	X	X	1	0
0	1	X	X	0	1
0	0	↓	0	0	1
0	0	↓	1	1	0

Truth table



logic diagram

Flip-flop conversions:-

Step 1;- obtain the characteristic table of required flip-flop.

Step 2;- And also obtain the Excitation table of available flip-flop.

Step 3:- obtain the expressions for the inputs of the existing flip-flop in terms of the inputs of the required flip-flop and the present state variables of the existing flip-flop and implement them.

Conversion of T-flip flop to S-R flip flop

Available flip-flop → T-flip flop (Excitation table)

Required flip-flop → S-R flip flop (Characteristic table)

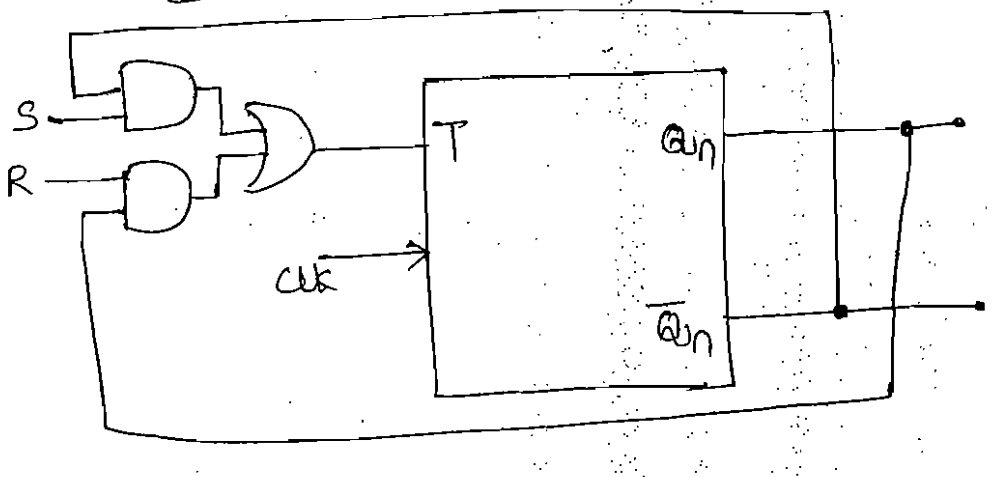
S	R	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	X	X
1	1	1	X	X

K-map for T

S \ RQ_n	00	01	11	10
0	0 (0)	0 (1)	0 (3)	0 (2)
1	1 (4)	0 (5)	X (7)	X (6)

$T = S.\bar{Q}_n + R Q_n$

Logic diagram



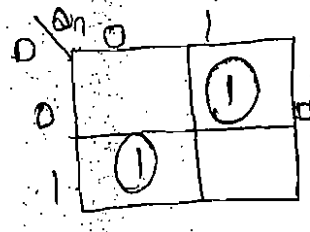
→ Conversion of T-flip-flop to D-flip-flop.

Available → T-flipflop → excitation table

Required → D-flipflop → characteristic table.

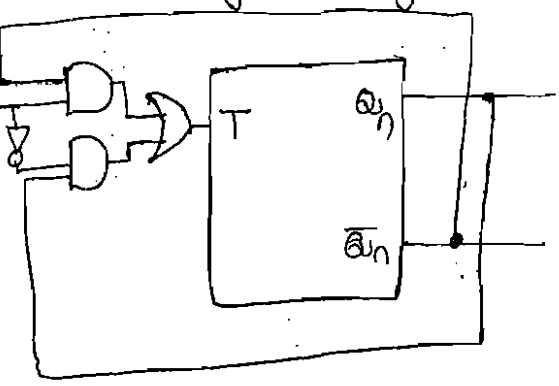
D	Q_n	Q_{n+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

K-map for T



$T = D\bar{Q}_n + \bar{D}Q_n$

Logic diagram



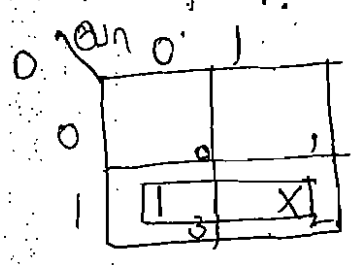
→ Construct a D-flip flop ~~to~~ using S-R flip flop

Available → S-R flipflop → excitation table.

Required → D flip flop → characteristic table.

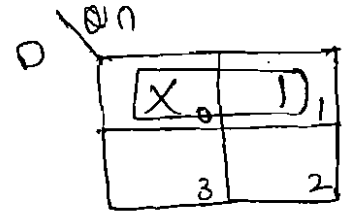
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-map for S



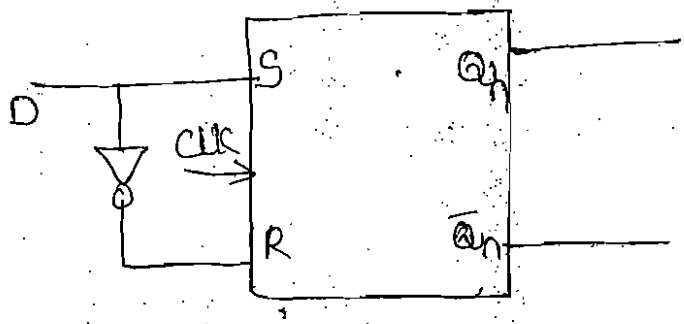
$S = D$

K-map for R



$R = \bar{D}$

Logic diagram.



→ Relize the S-R flip-flop by using J-K flip flop.

Available → S-R → excitation table

Required → J-K → characteristic table

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

K-map for S

J \ Q_n	00	01	11	10
0	0	X	0	0
1	1	X	0	1

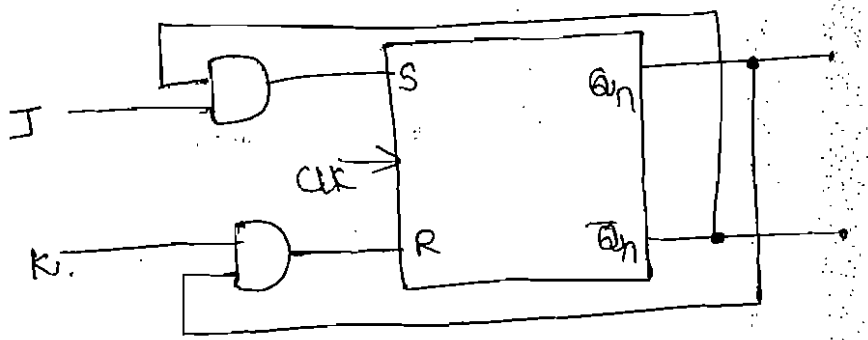
$S = J\bar{Q}_n$

K-map for R

K \ Q_n	0	1	X
X	0	1	X
0	0	1	0

$R = KQ_n$

Logic diagram:-



→ Convert J-K flip-flop to T flip-flop

T	Q_n	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-map for J

T \ Q_n	0	1
0	0	X
1	1	X

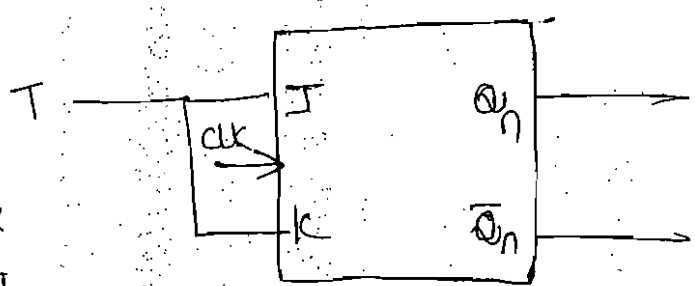
$J = T$

K-map for K

T \ Q_n	0	1
0	X	0
1	X	1

$K = T$

Logic-diagram

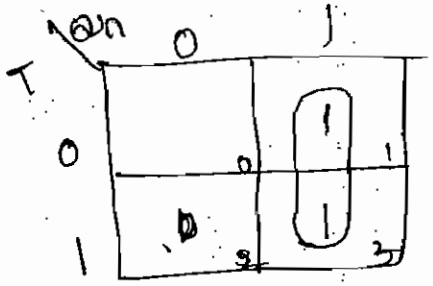


→ conversion of D-flip flop to T flip-flop.

* Available → D-flip flop → excitation table
* Available → T-flip flop → characteristic table.

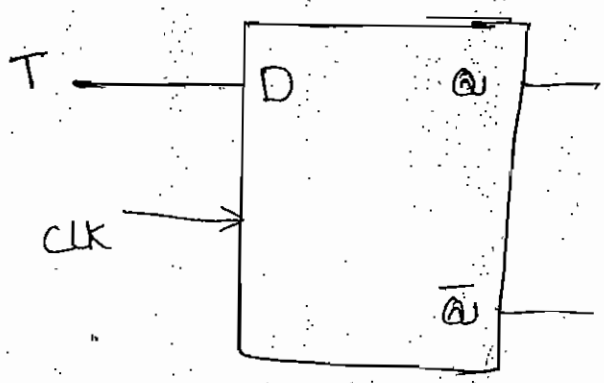
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

K-map for D



D = T.

logic diagram



Counters :- A digital counter is a set of flip-flops whose states change in response to pulses applied at the input to the counter.

→ The name itself it indicates, a counter is used to count pulses.

→ Counters may be asynchronous counters & synchronous counters. Asynchronous counters are also called ripple counters.

→ In asynchronous counters Flipflops are not triggered simultaneously. The clock does not directly control the time at which every stage changes state.

→ In synchronous counters are clocked such that each FF in the counter is triggered at the same time.

Comparison of synchronous and Asynchronous counters

Asynchronous Counters	Synchronous counters.
1. In this type of counter FF's are connected in such a way that the output of first FF drives the clock for the second FF, the output of the second FF to the third FF.	1. In this type of counter there is no connection between the output of first FF and clock input of next FF and so on.
2. All the FF's are not clocked simultaneously.	2. All the FFs are clocked simultaneously.
3. Design and implement is very simple even for more number of states.	3. Design and implementation becomes tedious and complex as the number of states increases.
4. main drawback of these counters is their low speed as the clock is propagated through a number of FFs before it reaches the last FF.	4. since clock is applied to all the FF's simultaneously the total propagation delay is equal to the propagation delay of only one FF. Hence they are faster.

Synchronous counters :-

→ Synchronous counters have the advantages of high speed and less severe decoding problems but the disadvantage of having more circuitry than that of Asynchronous counters.

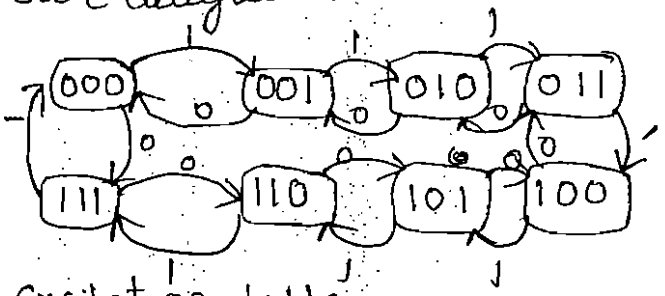
Design steps of Synchronous counters :-

1. Number of flip-flops
2. State diagram
3. Choice of flip-flops and excitation table.
4. minimal expression for excitations
5. Logic diagram.

→ Design of a synchronous 3-bit up-down counter using J-K FF's

Step 1 :- A 3-bit counter requires 3-FFs. It has 8 states. (000 ---- 111) and all the states are valid.

Step 2 :- State diagram



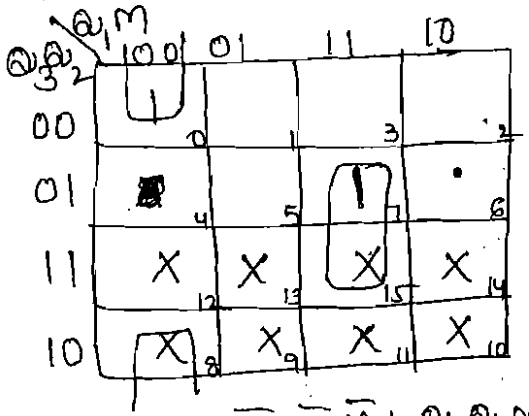
m=0 down counting
m=1 up counting

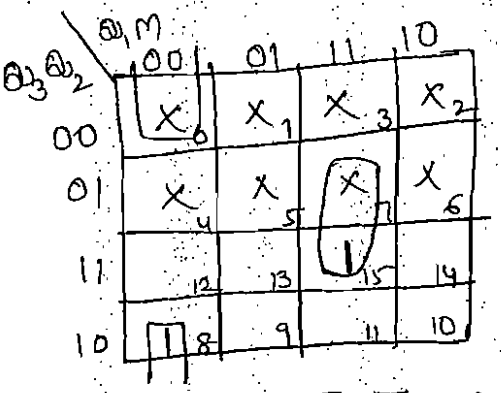
Step 3 :- Excitation table

mode (m)	Present State Q_3 Q_2 Q_1	Next state Q_3 Q_2 Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0 0 0	1 1 1	1	X	1	X	1	X
0	0 0 1	0 0 0	0	X	0	X	0	1
0	0 1 0	0 0 1	0	X	X	1	1	X
0	0 1 1	0 1 0	0	X	X	0	X	1
0	1 0 0	0 1 1	X	1	1	X	1	X
0	1 0 1	1 0 0	X	0	0	X	X	1
0	1 1 0	1 0 1	X	0	X	1	1	X
0	1 1 1	1 1 0	X	0	X	0	X	1

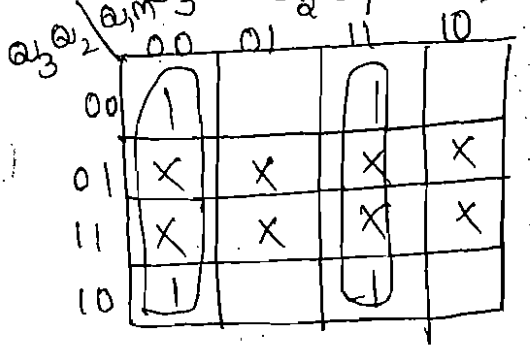
mode(m)	PS $Q_3Q_2Q_1$	N.S $Q_3\ Q_2\ Q_1$	J_3	K_3	J_2	K_2	J_1	K_1
1	0 0 0	0 0 1	0	X	0	X	1	X
1	0 0 1	0 1 0	0	X	1	X	X	1
1	0 1 0	0 1 1	0	X	X	0	1	X
1	0 1 1	1 0 0	1	X	X	1	X	1
1	1 0 0	1 0 1	X	0	0	X	1	X
1	1 0 1	1 1 0	X	0	1	X	X	1
1	1 1 0	1 1 1	X	0	X	0	1	X
1	1 1 1	0 0 0	X	1	X	1	X	1

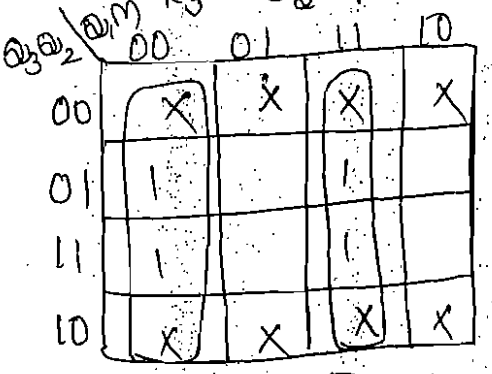
Step 4:- obtain the minimal expression.



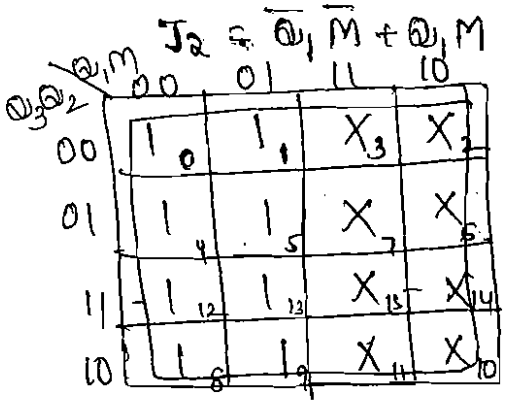


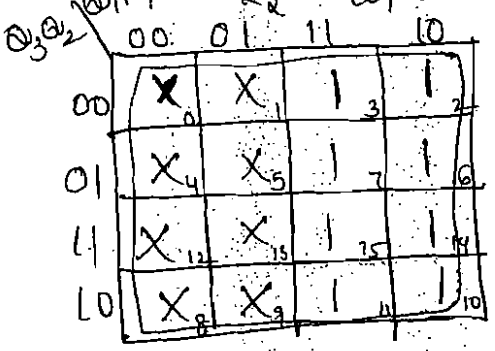
$K_3 = \bar{Q}_2\bar{Q}_1\bar{M} + Q_2Q_1M$





$K_2 = \bar{Q}_1\bar{M} + Q_1M$

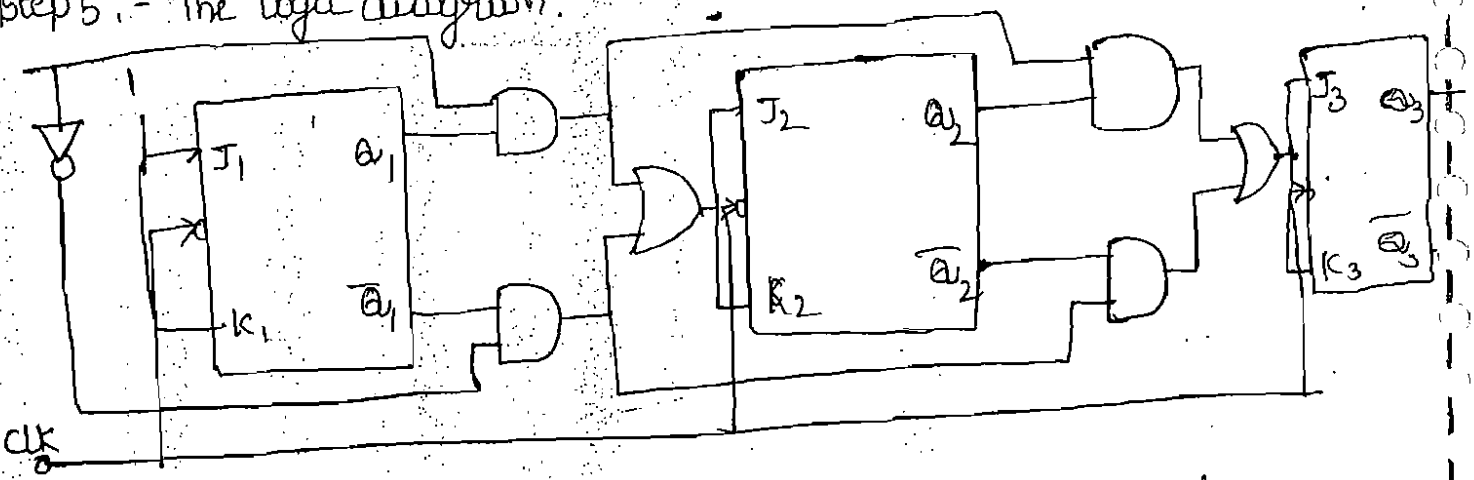




$J_1 = 1$

$K_1 = 1$.

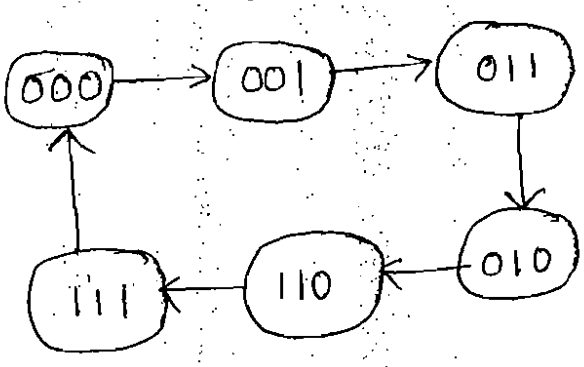
Step 5 :- The logic diagram.



→ Design of a synchronous modulo-6 Gray code counter.

Step 1 :- number of flip flops - 3 FF'S

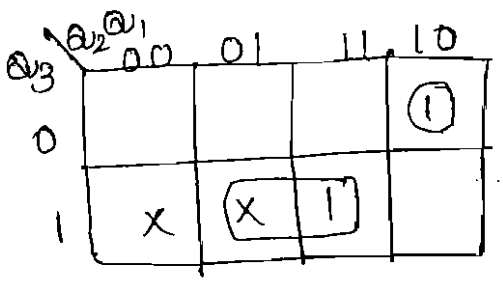
Step 2 :- state diagram



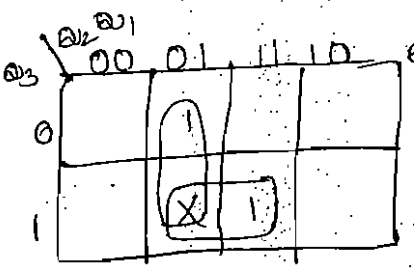
Step 3 :-

Present state			next state			Required excitations		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	T_3	T_2	T_1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

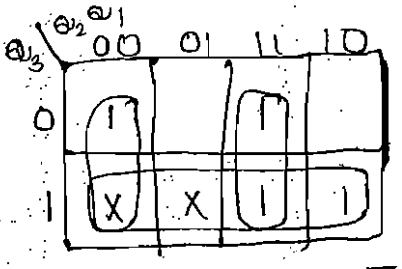
Step 4 :- minimal expressions :-



$T_3 = Q_3Q_1 + \bar{Q}_3Q_2\bar{Q}_1$

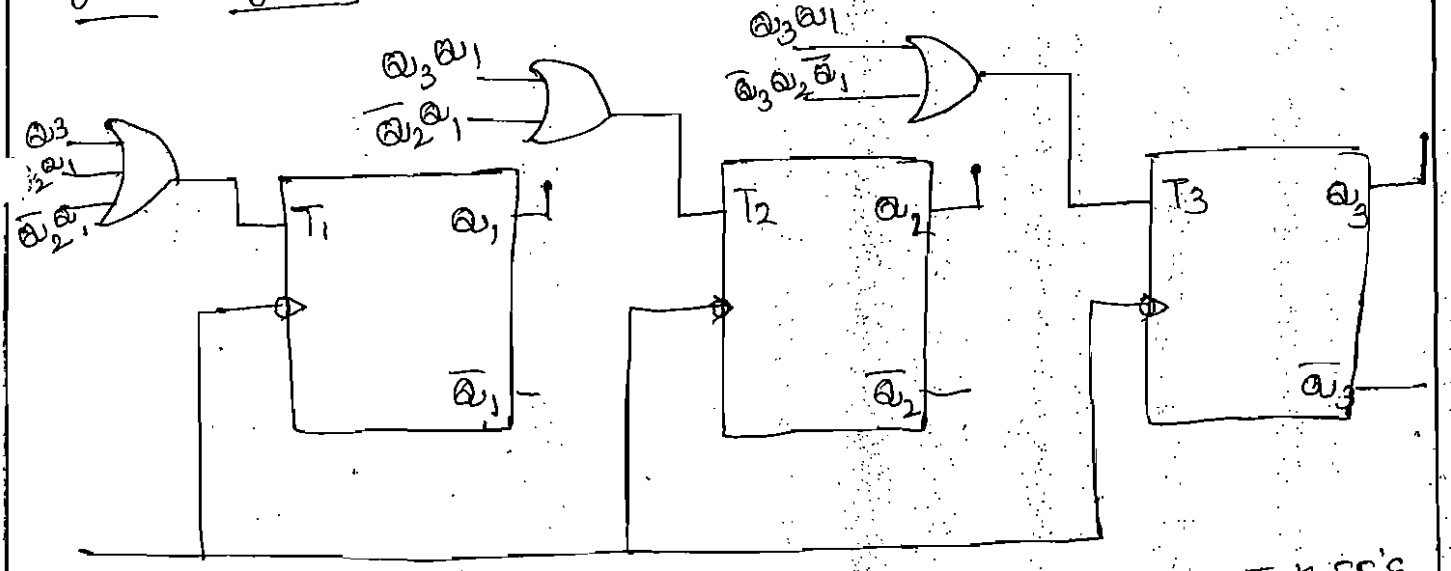


$T_2 = Q_3Q_1 + \bar{Q}_2Q_1$



$T_1 = Q_3 + Q_2Q_1 + \bar{Q}_2\bar{Q}_1$

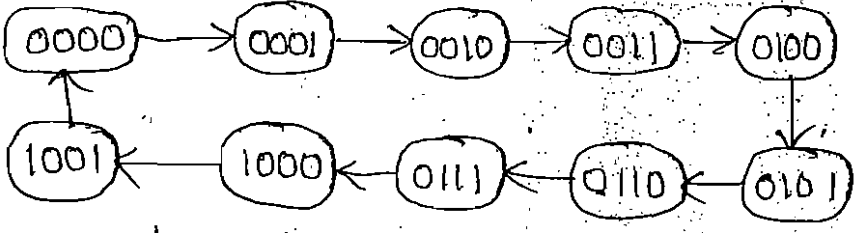
Logic diagram :-



→ Design a synchronous (mod-9) BCD counter using J-K FF's.

Step 1 :- 4 FF's

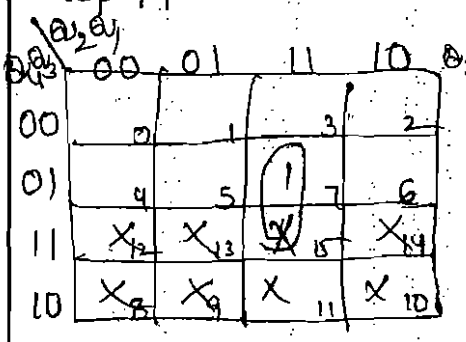
Step 2 :- State diagram.



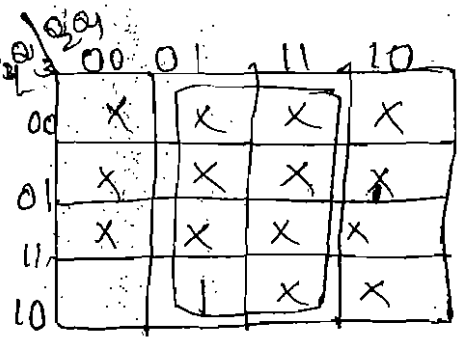
Step 3 :- excitation table.

Present state $Q_4\ Q_3\ Q_2\ Q_1$	next state $Q_4\ Q_3\ Q_2\ Q_1$	J_4 K_4	J_3 K_3	J_2 K_2	J_1 K_1
0 0 0 0	0 0 0 1	0 X	0 X	0 X	1 X
0 0 0 1	0 0 1 0	0 X	0 X	1 X	X 1
0 0 1 0	0 0 1 1	0 X	0 X	X 0	1 X
0 0 1 1	0 1 0 0	0 X	1 X	X 1	X 1
0 1 0 0	0 1 0 1	0 X	X 0	0 X	1 X
0 1 0 1	0 1 1 0	0 X	X 0	1 X	X 1
0 1 1 0	0 1 1 1	0 X	X 0	X 0	1 X
0 1 1 1	1 0 0 0	1 X	X 1	X 1	X 1
1 0 0 0	1 0 0 1	X 0	0 X	0 X	1 X
1 0 0 1	0 0 0 0	X 1	0 X	0 X	X 1

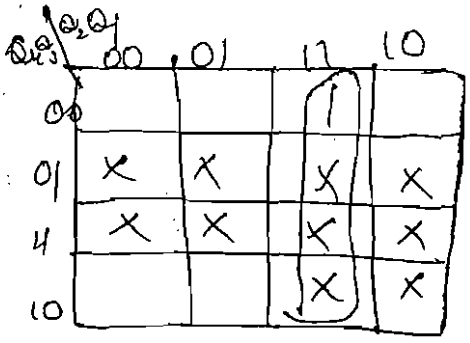
Step 4 :-



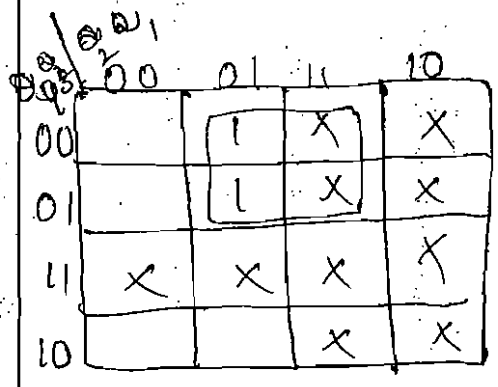
$J_4 = Q_3 Q_2 Q_1$



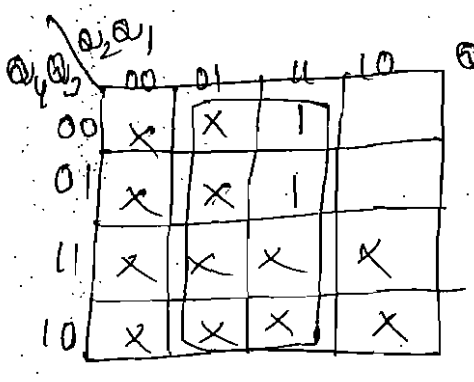
$K_4 = Q_1$



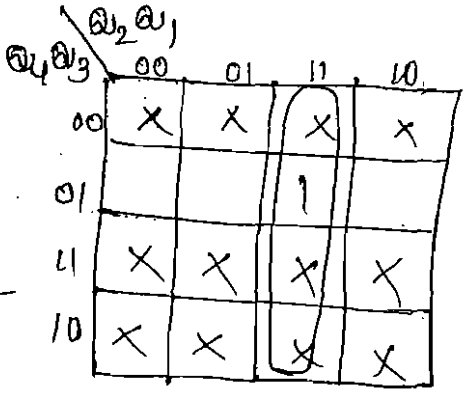
$J_3 = Q_2 Q_1$



$J_2 = \overline{Q_4} Q_1$

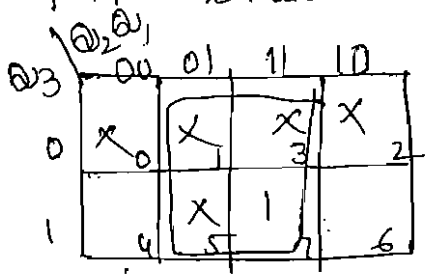


$K_2 = Q_1$

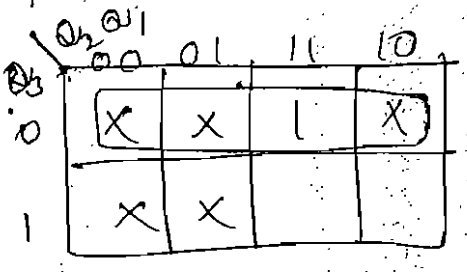


$K_3 = Q_2 Q_1$

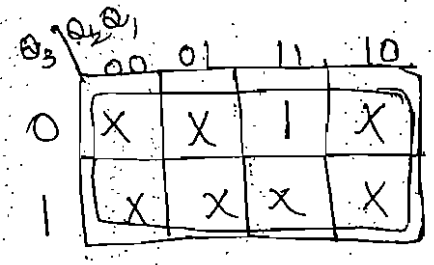
Step 4 :- The minimal expression.



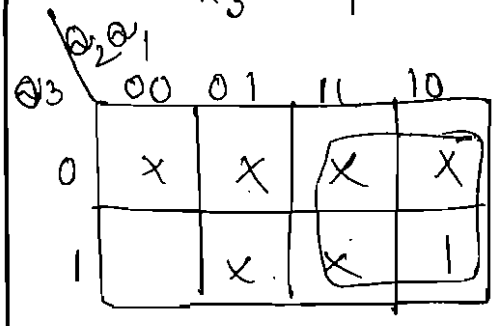
$K_3 = Q_1$



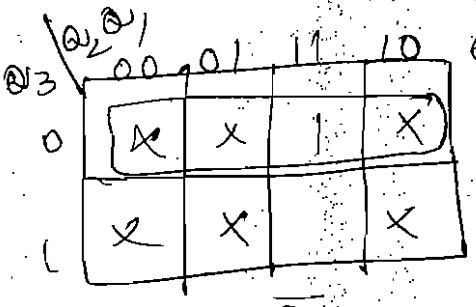
$K_2 = \overline{Q_3}$



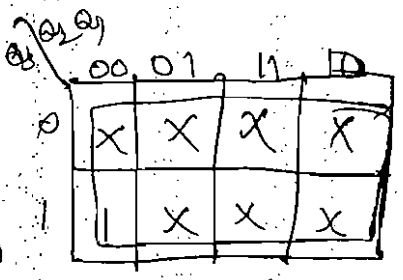
$J_3 = 1$



$J_1 = Q_2$

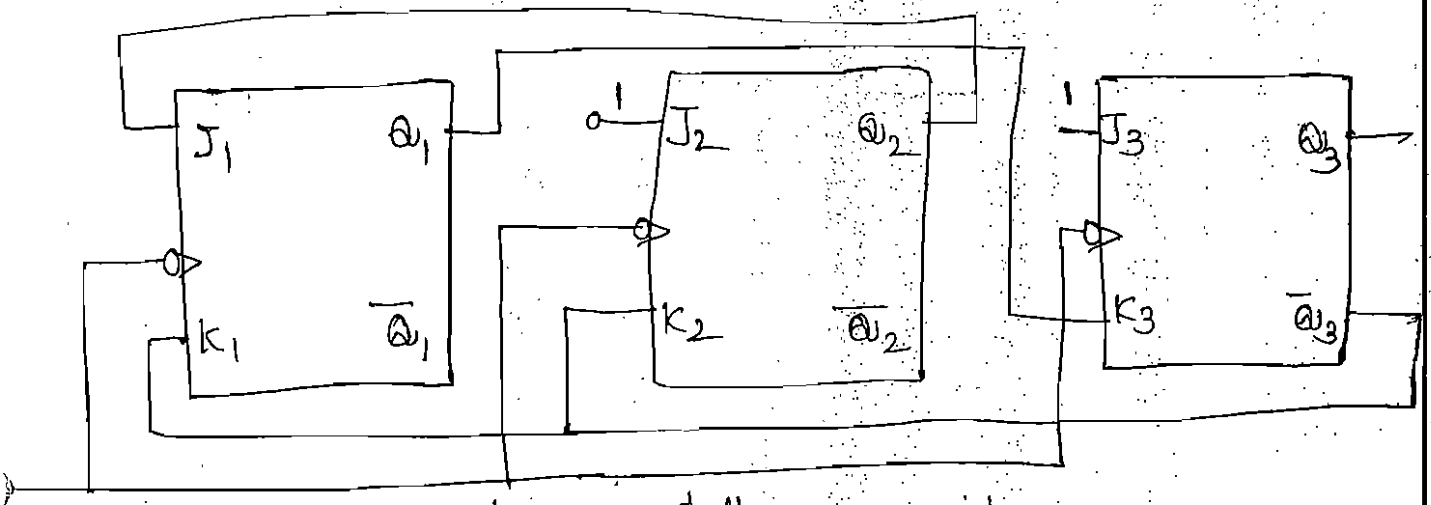


$K_1 = \overline{Q_3}$

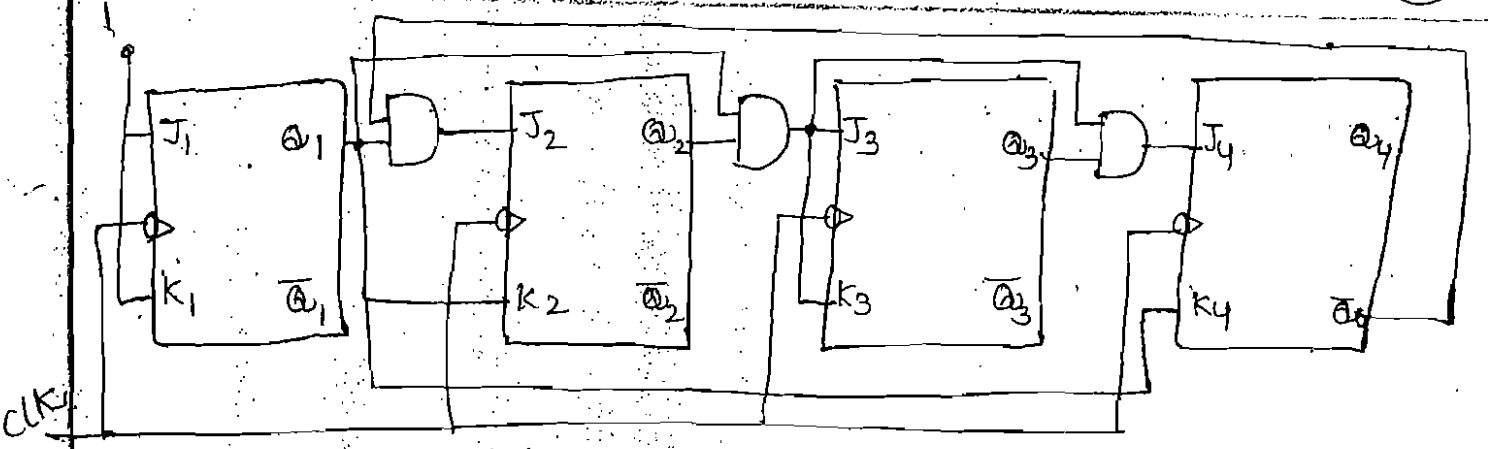


$J_2 = 1$

Step 5 :- logic diagram



logic diagram of the J-K counter.

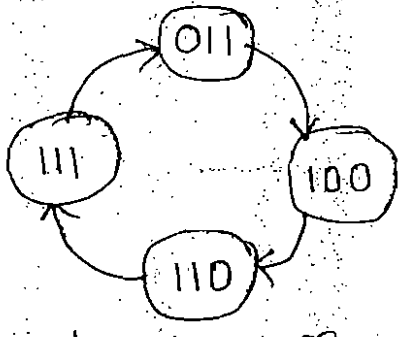


Logic diagram

→ Design a J-K counter that goes through states 3, 4, 6, 7 and 3... is the counter self-starting (take a remaining states are invalid)

Step 1:- number of flip-flops - 3 flip-flops.

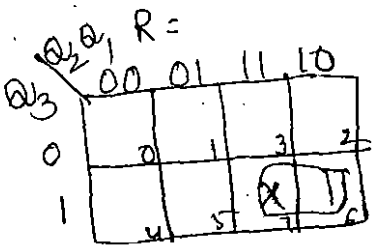
Step 2:- State diagram.

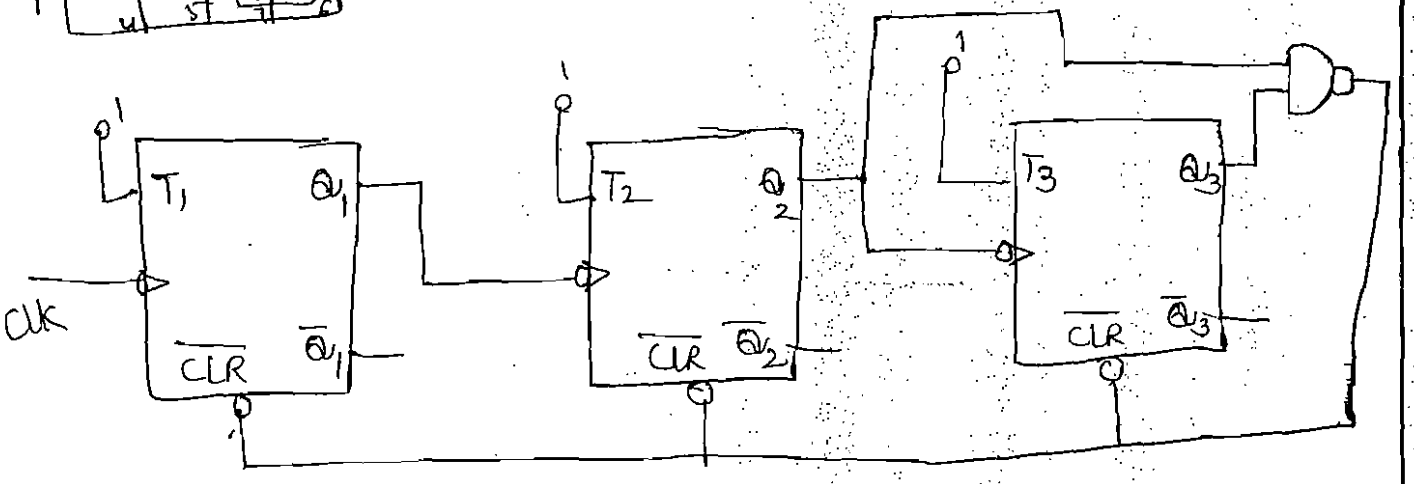


State-diagram

Step 3:- excitation table

Present state Q_3 Q_2 Q_1	next state Q_3 Q_2 Q_1	Required inputs J_3 K_3	J_2 K_2	J_1 K_1
0 1 1	1 0 0	1 X	X 1	X 1
1 0 0	1 1 0	X 0	1 X	0 X
1 1 0	1 1 1	X 0	X 0	1 X
1 1 1	0 1 1	X 1	X 0	X 0

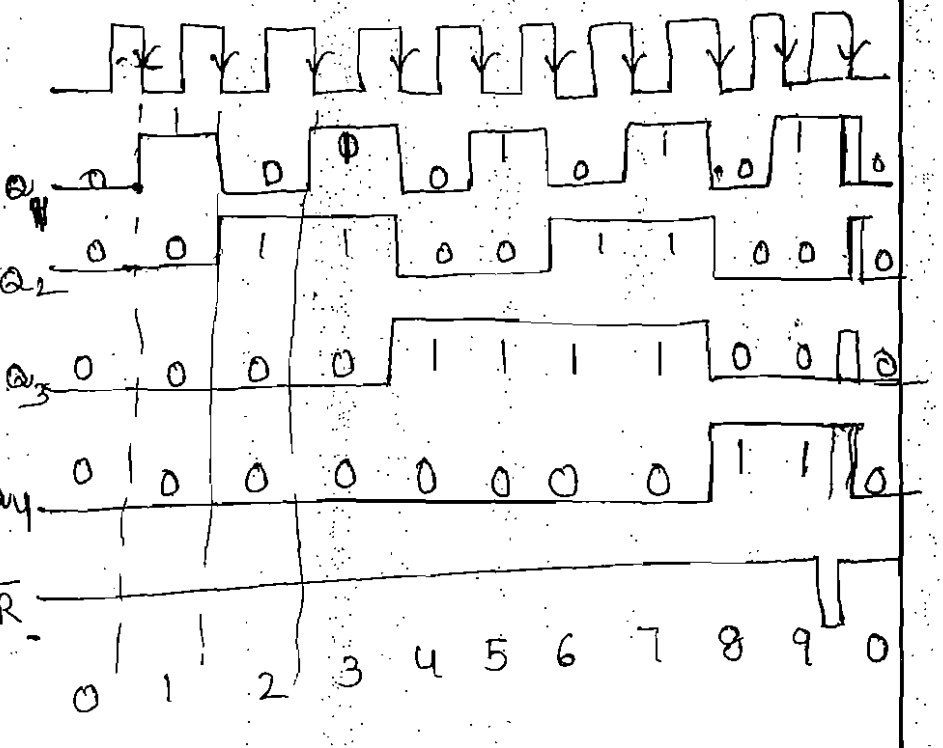




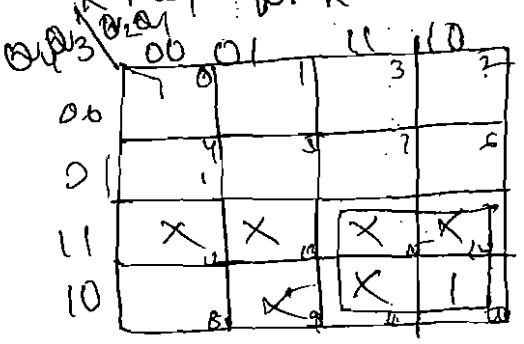
logic diagram.

mod -10 Asynchronous counter :-

R	After pulses	Count Q_4 Q_3 Q_2 Q_1
0	0	0 0 0 0
0	1	0 0 0 1
0	2	0 0 1 0
0	3	0 0 1 1
0	4	0 1 0 0
0	5	0 1 0 1
0	6	0 1 1 0
0	7	0 1 1 1
0	8	1 0 0 0
0	9	1 0 0 1
1	10	0 0 0 0



K-map for R



R = 0 for 0000 to 1001
R = 1 for 1010
R = X for 1011 to 1111

$R = Q_4Q_2$

A synchronous counters :-

→ Design a mod-6 Asynchronous counter using T FF's

→ A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When six pulse is applied, the counter temporarily goes to 110 state but immediately reset to 000.

→ It requires three FF's, because the smallest value of n satisfying the condition $N \leq 2^n$ is $n = 3$: three FFs can have eight possible states, out of which only six are utilized and remaining two states 110 and 111.

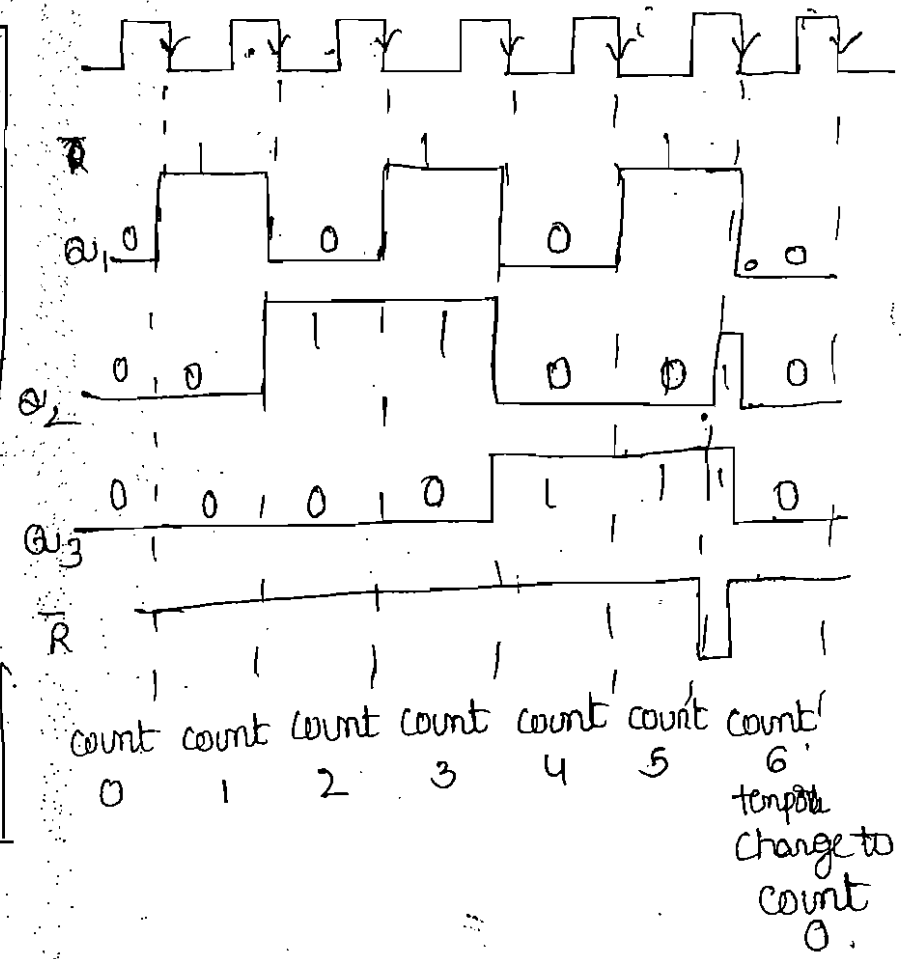
→ For the design, To write a truth table with the present state outputs Q_3, Q_2 and Q_1 as the variables, and reset R as the output.

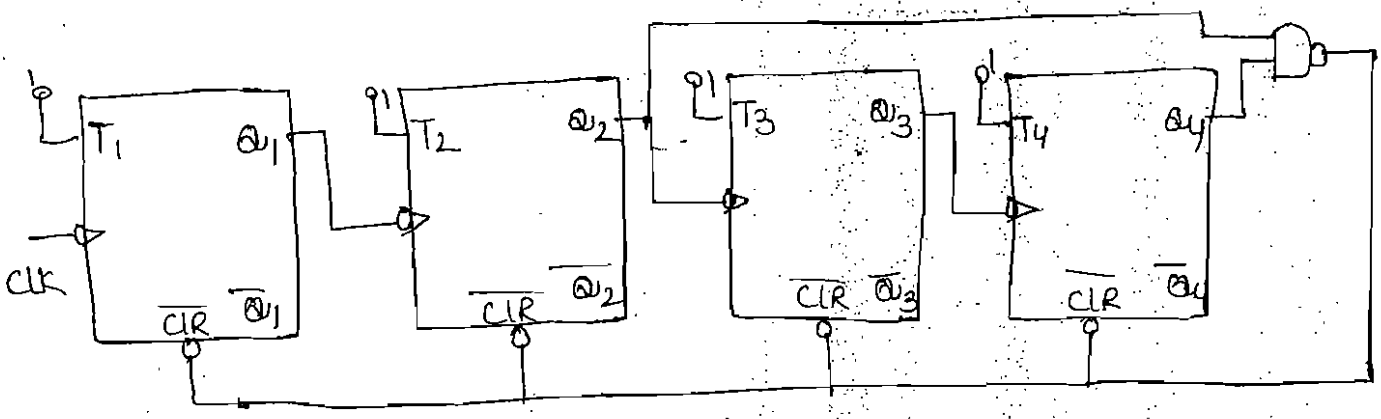
R = 0 for 000 to 101, R = 1 for 110, and R = X for 111.

Table for R

After pulses	State Q_3	Q_2	Q_1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	
	1	1	1	X
7	0	0	1	1

Timing diagram.





logic - diagram.

Asynchronous mod - 10 counter using T Flip-flops.

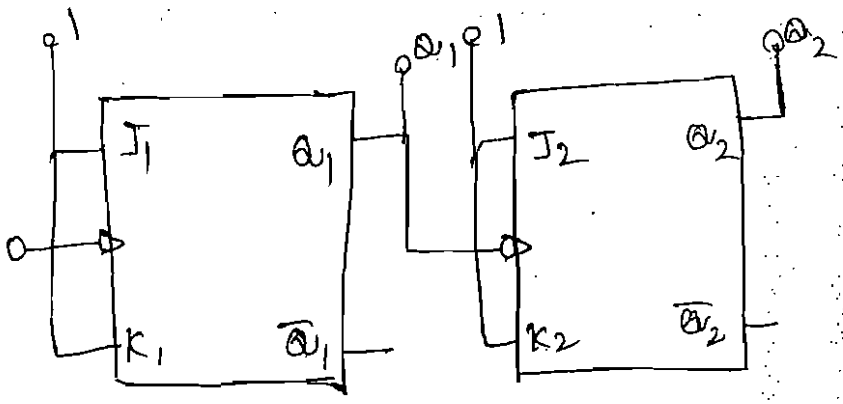
→ Two - bit ripple up- counter using negative edge - triggered flip-flops:

→ The 2-bit up - counter counts in the order 0,1,2,3,0,1... i.e 00, 01, 10, 11, 00, 01.. etc. A 2-bit ripple up-counter, using negative edge - triggered J-K FFs, The counter initially reset to 00 when first clock pulse is applied, FF_1 toggles at the negative-going edge of this pulse, therefore, Q_1 goes from low to high. This becomes a positive - going signal at the clock input of FF_2.

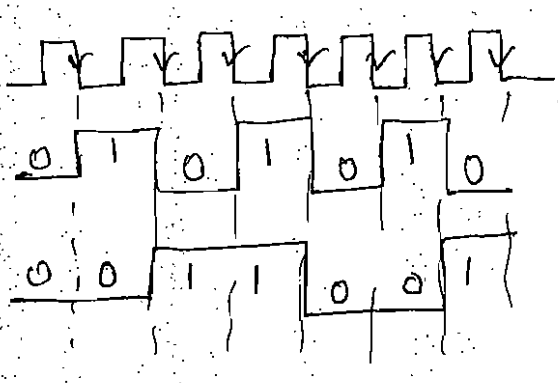
→ So FF_2 is not affected, and hence, the state of the counter after one clock pulse is $Q_1 = 1$ and $Q_2 = 0$.

→ So next clock pulse. FF_1 is change to 1 to 0. then negative going edge of this pulse FF_2 is change to 0 to 1. $Q_1 = 0$ and $Q_2 = 1$

→ So next clock pulse FF_1 is change to 0 to 1. then positive going edge of this pulse FF_2 is no change $Q_1 = 1$, $Q_2 = 1$.

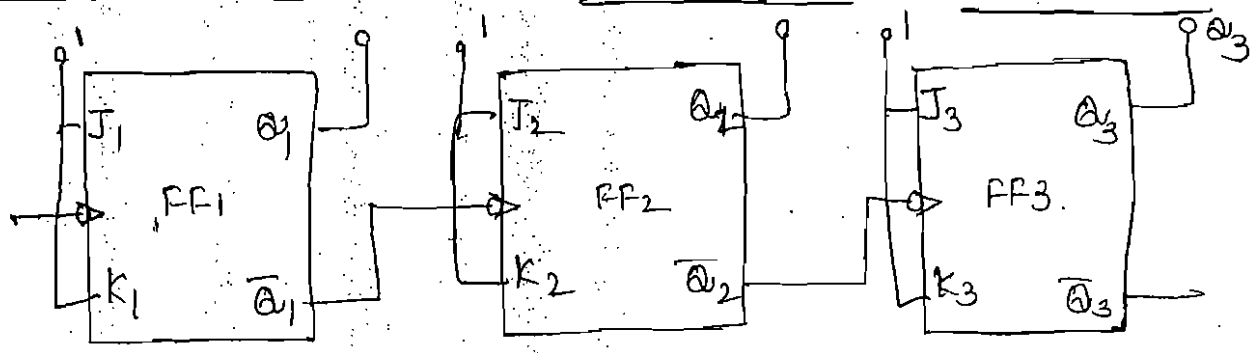


Logic diagram.



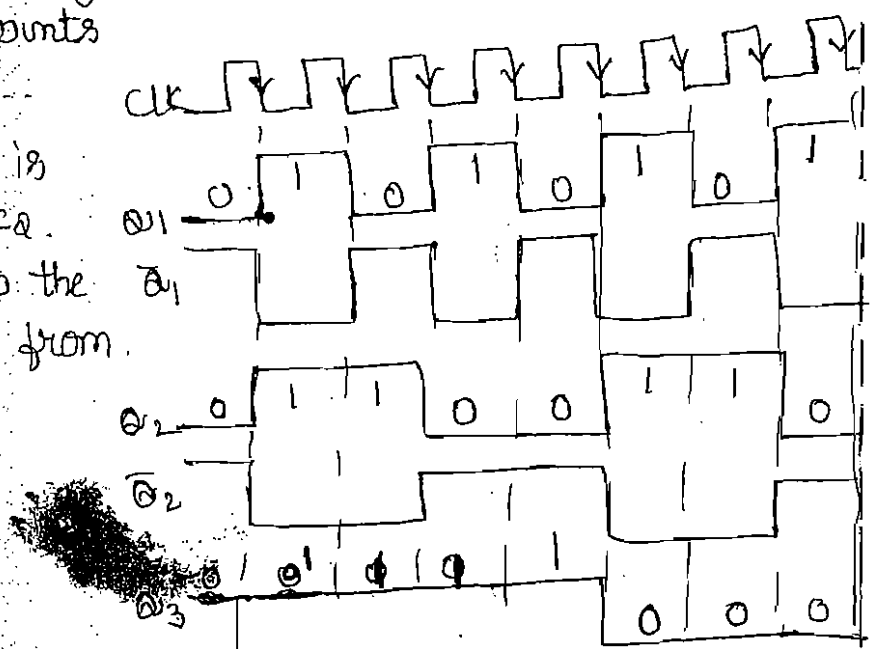
Timing diagram.

3-bit down-counter using negative-edge trigged J-K flip-flops.



Logic diagram.

A 3-bit down counter counts in order 0,7,6,5,4,3,2,1,0,1 ... For down counting $\bar{Q}_1$ to FF1 is connected to the clock of FF2. the $\bar{Q}_2$ to FF2 is connected to the clock of FF3. output taken from Q_1, Q_2 and Q_3.



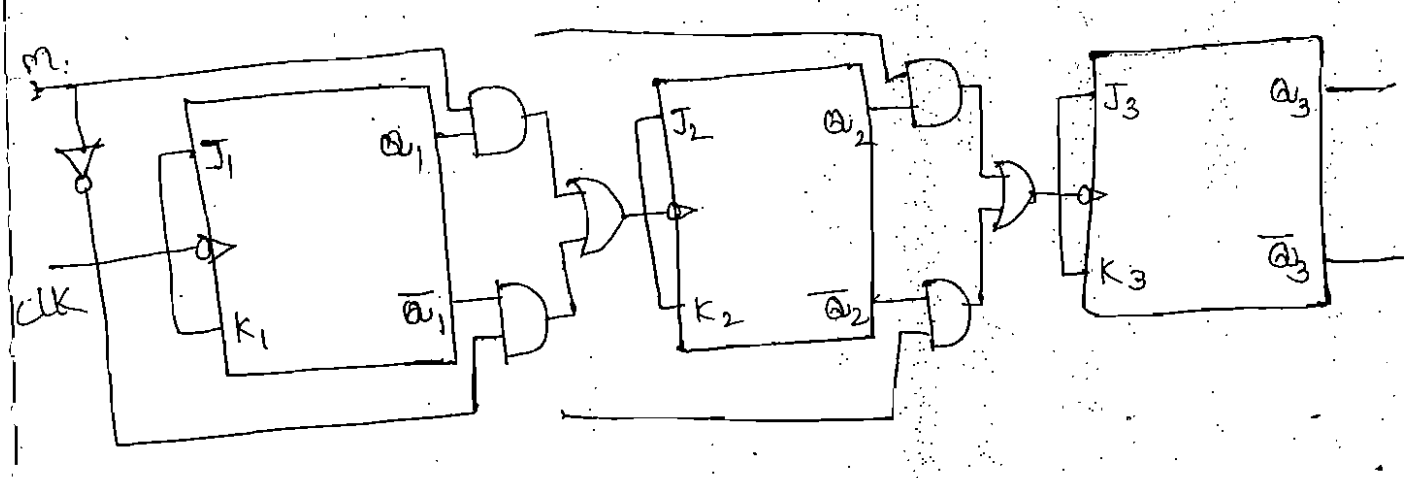
3-bit updown counter using negative edge-triggered flip-flops :-

As the name indicates an up-down counter is a counter which can count both in upward and downward directions An up-down counter is also called a forward / backward counter or a bi-directional counter.

→ So control signal M or mode signal is required to choose the direction of count.

→ when m=1 for up counting, m=0 for down counting for up-counting Q_1 is transmitted to clock FF2 and for down-counting $\bar{Q}_1$ is transmitted to clock FF2.

→ This is achieved by using two AND gates and one OR Gate.



Shift registers :-

→ As a flip-flop can store only one bit of data, a '0' or a '1', it is referred to as a single-bit register. When more bits of data are to be stored, a number of a number of FF's are used.

→ A register is a set of FF's used to store binary data. The storage capacity of a register is the number of bits of digital data it can retain

→ Shift-register are a type of logic circuits closely related to counters.

→ They are used basically for the storage and transfer of digital data. The basic difference between a shift register and counter is that a shift register has no specified sequence of states except in certain very specialized applications.

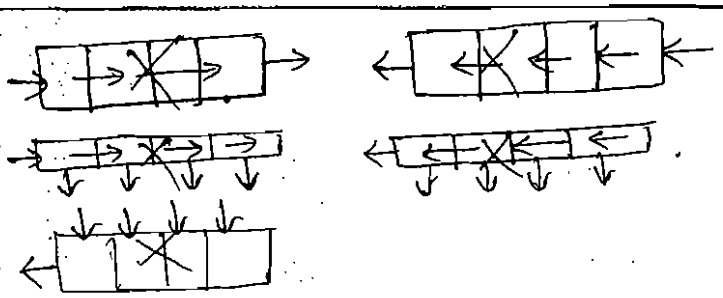
→ where as a counter has a specified sequence of states.

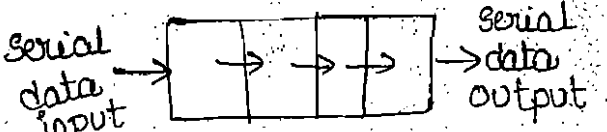
Data-Transmission in shift registers :-

→ A number of flip-flop's connected together such that data may be shifted into and shifted out of them is called a shift-register.

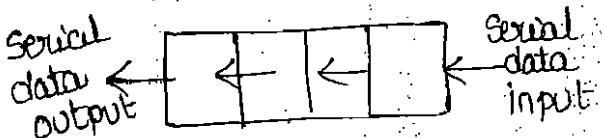
→ Data may be shifted into & out of the register either in serial form & in parallel form. So, there are four types of shift-registers. They are.

1. Serial - in, Serial - out
2. Serial - in, parallel - out
3. parallel - in, serial - out
4. parallel - in, parallel - out



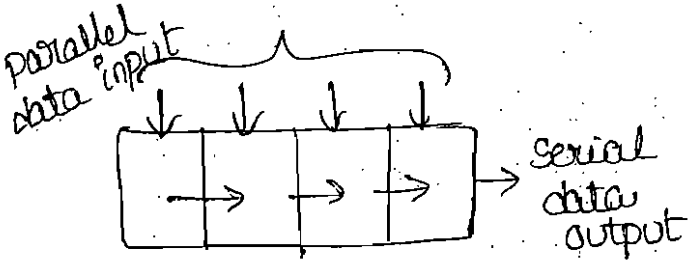


a) Shift-right

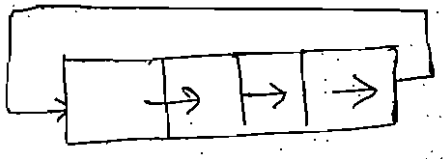


b) Shift - left.

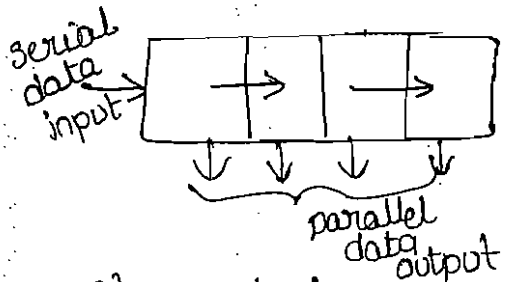
Serial - in, Serial - out



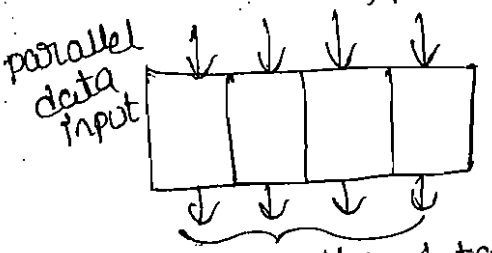
c) parallel - in, Serial output.



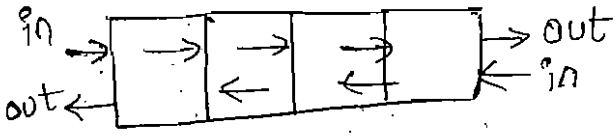
g) Rotate - right Shift register



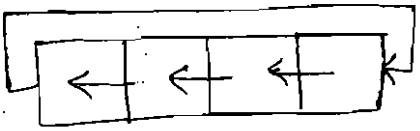
c) Serial - in, parallel out Shift reg



d) parallel - in, parallel - out, shift regis

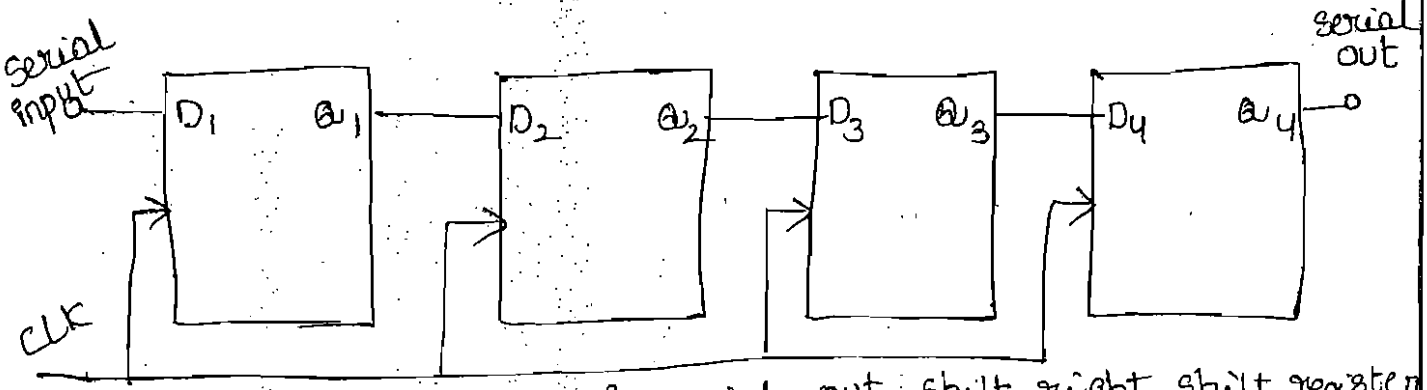


f) bi-directional shift register.

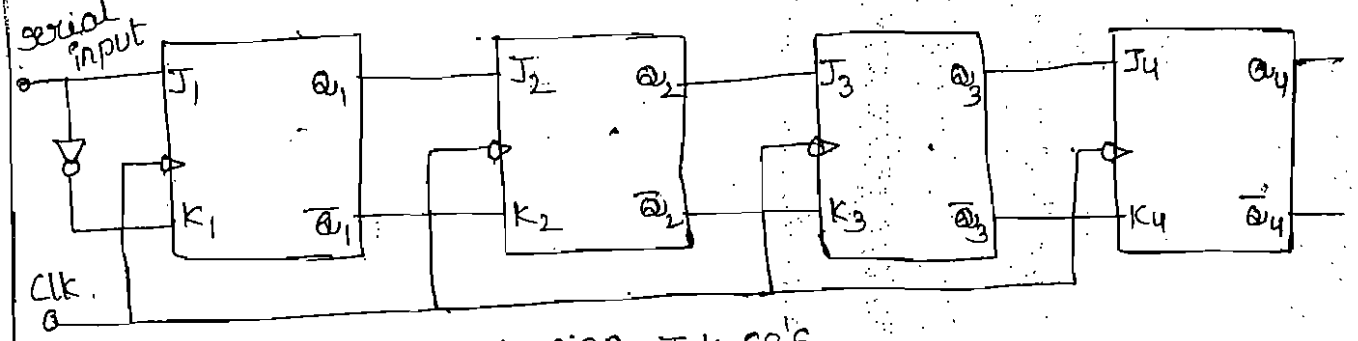


h) Rotate - left Shift register.

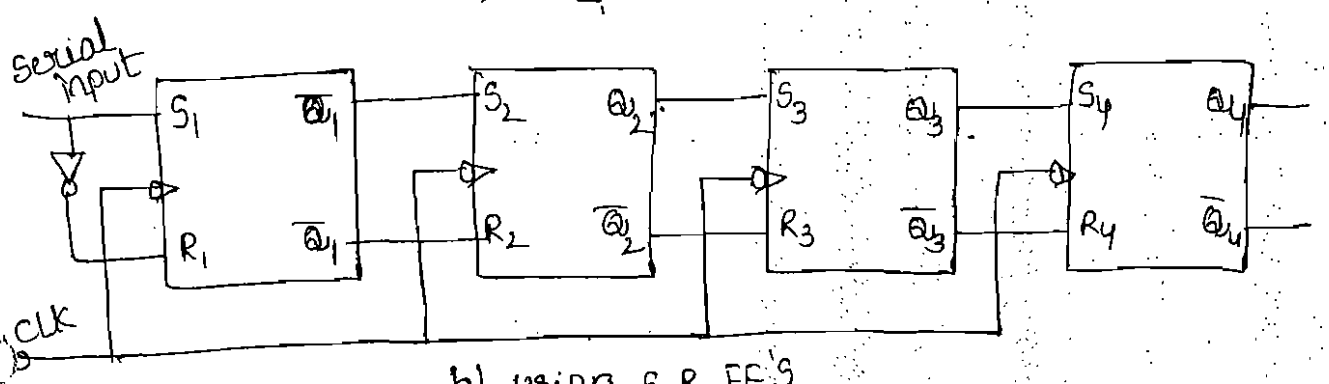
→ Serial - in, Serial - out Shift register :-



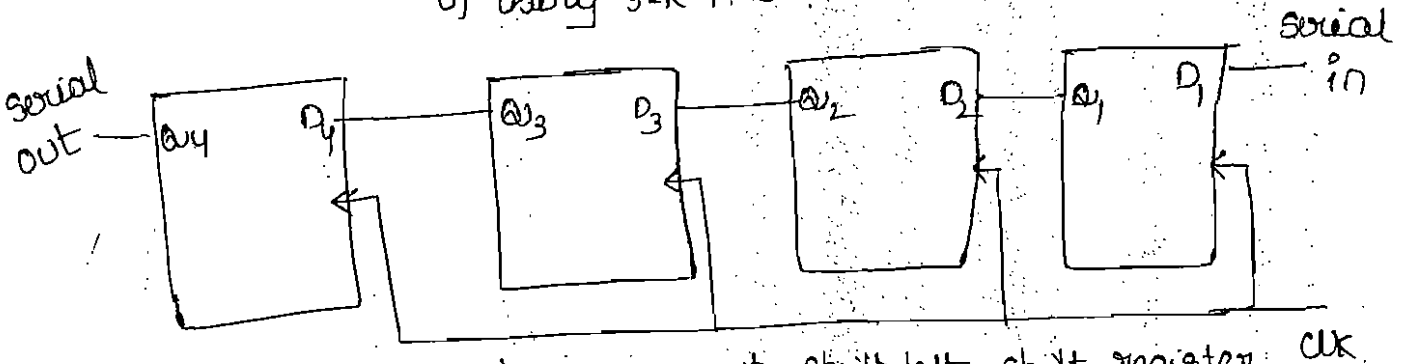
4-bit Serial - in, Serial - out, Shift right Shift register



a) using J-K FF'S.

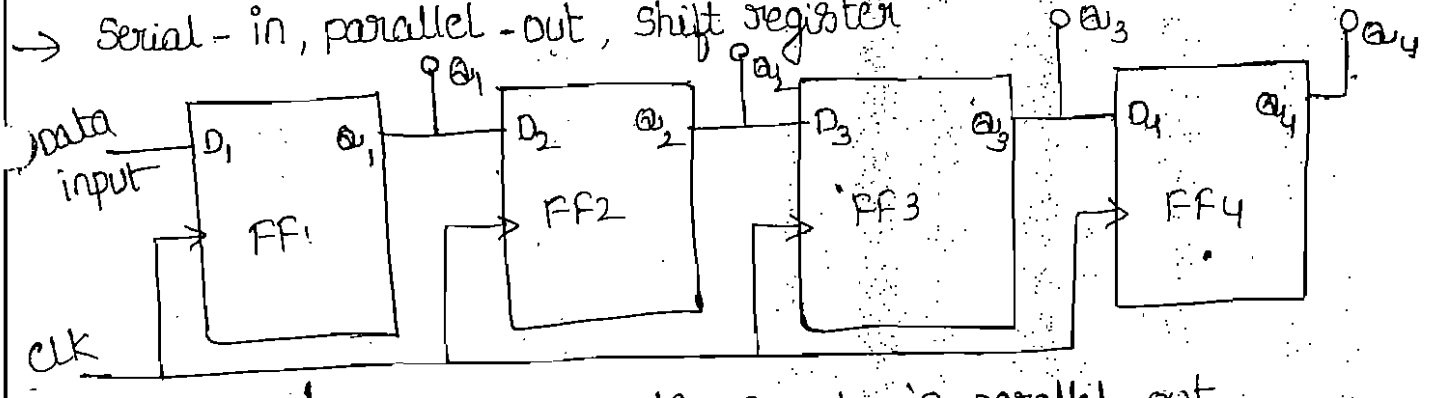


b) using S-R FF'S.

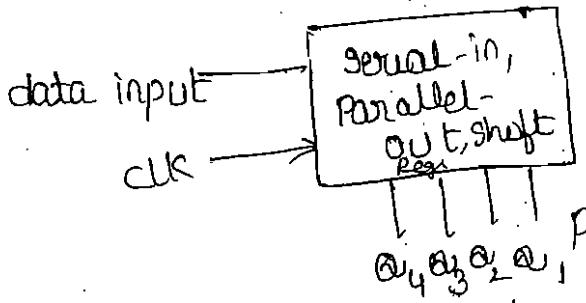


A 4-bit serial-in, serial-out, shift left shift register.

→ Serial-in, parallel-out, shift register

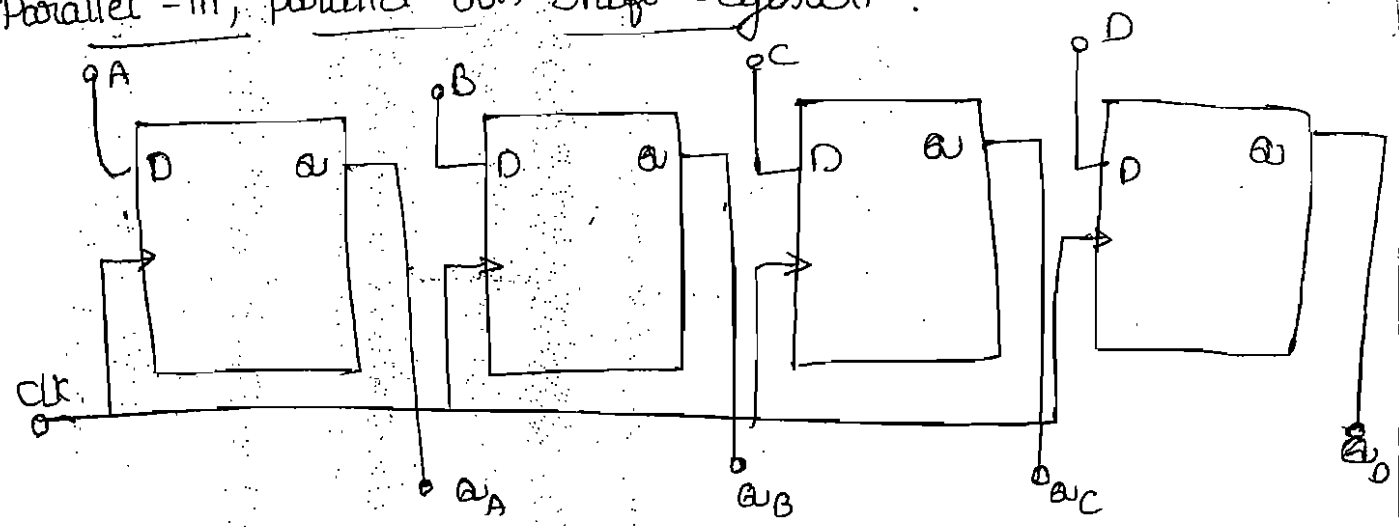


Logic diagram for serial-in, parallel-out



logic symbol.

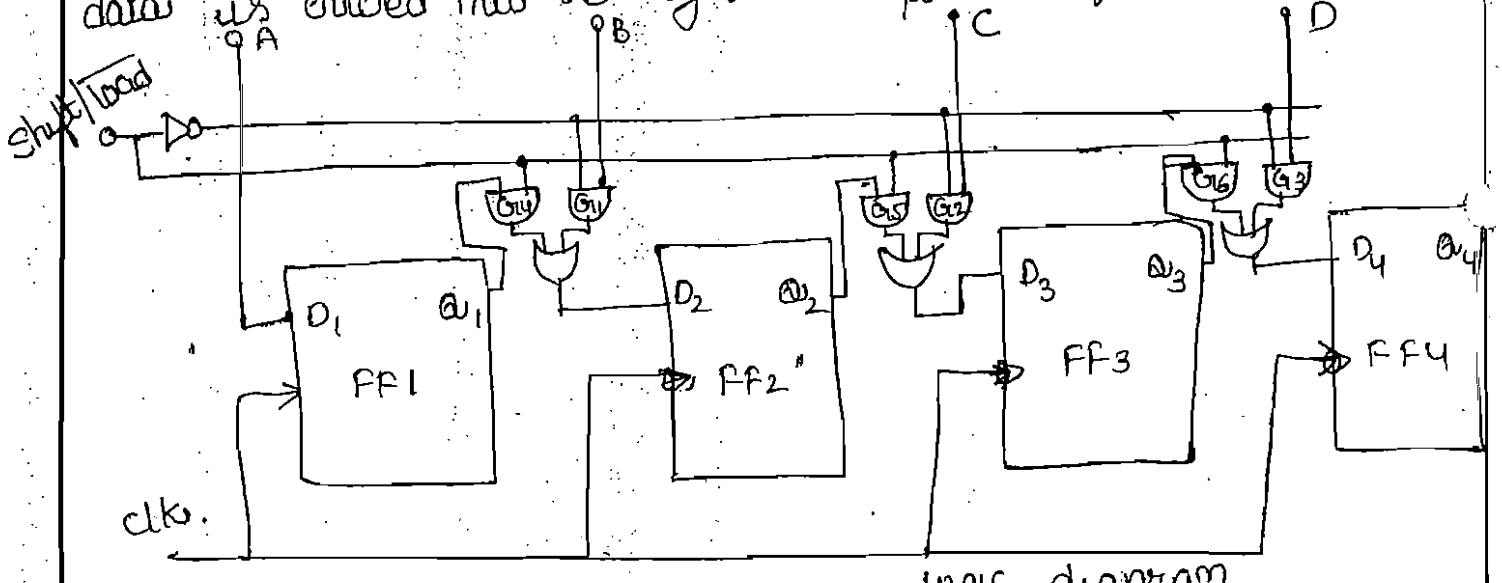
→ Parallel - in, parallel - out, shift - register :-



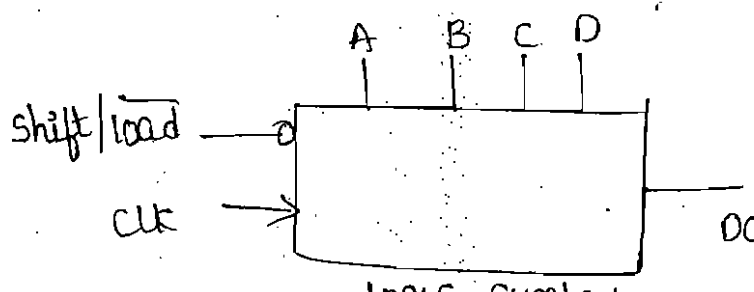
Logic diagram of a 4-bit parallel- in, parallel -out

→ Parallel - in, Serial - out Shift - register :-

For a parallel -in, serial- out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit- by- bit basis over a single line. a 4-bit parallel -in, serial -out, shift register using D-FFs. There are four data lines A, B, C and D through which the data is entered into the register in parallel form.



logic diagram



logic symbol

The signal Shift / $\overline{Load}$ allows (a) the data to be entered into the register in parallel form. (b) the data to be shifted out serially from terminal Q_4.

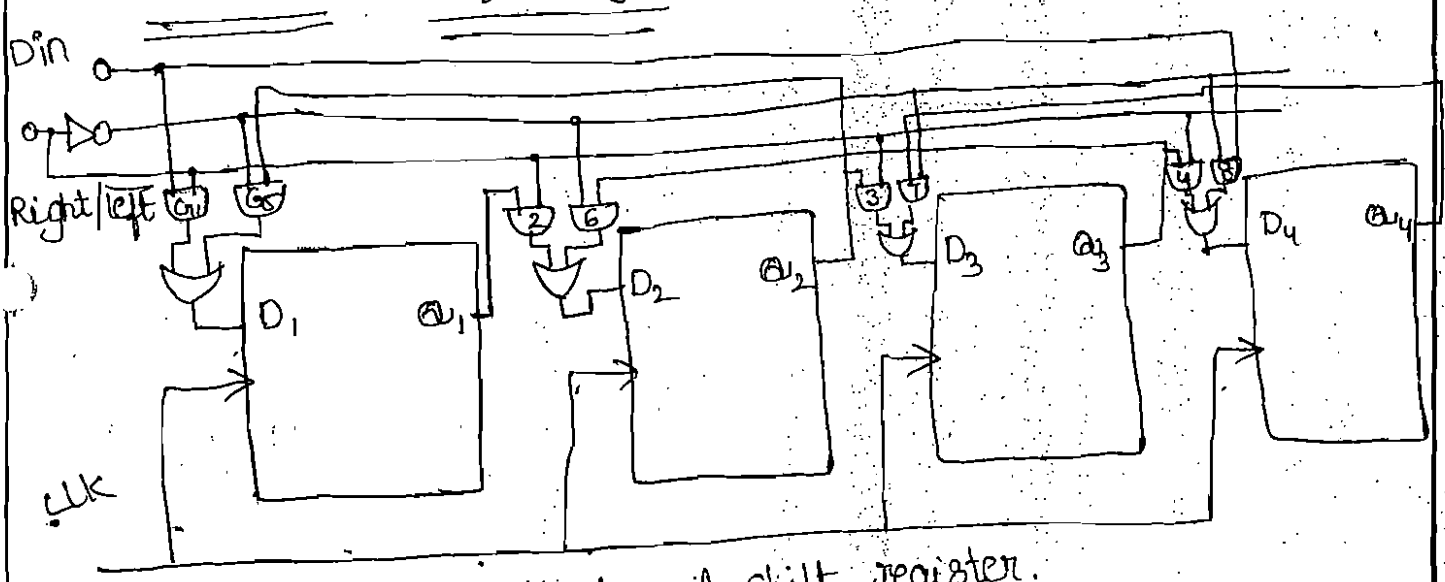
→ when Shift / $\overline{Load}$ line is high, gates G_1, G_2 and G_3 are disabled, but G_4, G_5, G_6 are enabled allowing the data bits to shift-right from one-stage to the next.

→ when Shift / $\overline{Load}$ line is low, gates G_4, G_5 and G_6 are disabled where as gates G_1, G_2 and G_3 are enabled allowing the data input to appear at the D inputs of the respective FFs.

→ when clock pulse is applied, these data bits are shifted to the Q output terminals of the FFs and, therefore, data is inputted in one step.

→ The OR Gate allows either the normal shifting operation or the parallel data entry depending on which AND gates are enabled by the level on the shift / $\overline{Load}$ input.

→ Bi-directional shift register :-



4-bit bi-directional shift register.

→ A bi-directional shift register is one in which the data bits can be shifted from left to Right or from Right to left.

→ in above figure 4-bit serial-in, serial-out, bidirectional (shift left, shift right) shift reighter.

→ when Right / $\overline{\text{left}}$ is a 1, the logic circuit works as a shift-right shift register.

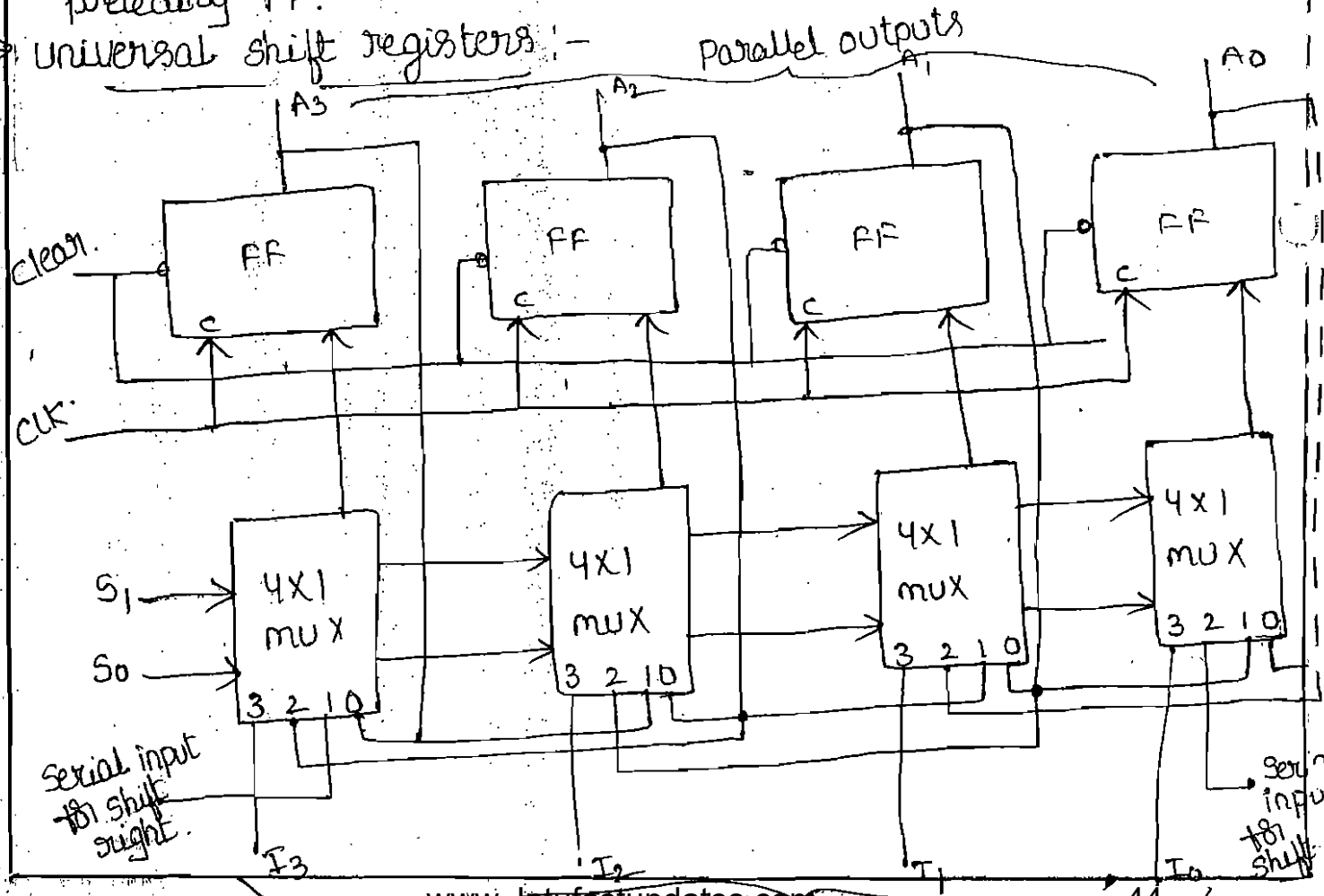
→ when Right / $\overline{\text{left}}$ is a 0, the logic circuit works as a shift-left shift register.

→ The bi-directional operation is achieved by using the mode signal and two AND gates and one OR Gate for each stage.

→ when mode signal Right / $\overline{\text{left}}$ is a '1' the 4 AND Gates. G_1, G_2, G_3 and G_4 are enabled and disables the AND Gates G_5, G_6, G_7 and G_8 and the state of a output of each FF is passed through the gate to the D input of the following FF.

→ when mode signal Right / $\overline{\text{left}}$ is a '0' the AND Gates. G_1, G_2, G_3 and G_4 are disabled, and enabled the AND gates G_5, G_6, G_7 and G_8 and the state of a output of each FF is passed through the gate to the D input of the preceding FF.

→ Universal shift registers :-



→ A register capable of shifting in one direction only is a unidirectional shift register. one that can shift in both directional is a bi-directional shift register.

→ If the register has both shifts and parallel load capabilities, it is referred to as a universal shift register.

→ A universal shift-register has both shifts as it means whose input can be either in serial form or in parallel form and whose output also can be either in serial form or in parallel form.

→ A universal shift register can be realized using multiplexers. it consists of four D-flip flops and four multiplexers.

→ The four multiplexers have two common selection inputs S_1 and S_0. Input 0 in each multiplexer is selected when $S_1S_0=00$ input 1 is selected when $S_1S_0=01$, and input 2 is selected when $S_1S_0=10$ and input 3 is selected when $S_1S_0=11$.

→ when $S_1S_0=0$, the present value of the register is applied to the D-input of flip-flops. This condition forms a path from the output of each flip-flop into the input of the same flip-flop

→ when $S_1S_0=01$, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flops. This causes a shift-right operation, with the serial input transferred into flip-flop A4.

mode control		Register operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	parallel load.

→ when $S_1S_0=10$, a shift-left operation results with the other serial input going into flip-flop A1.

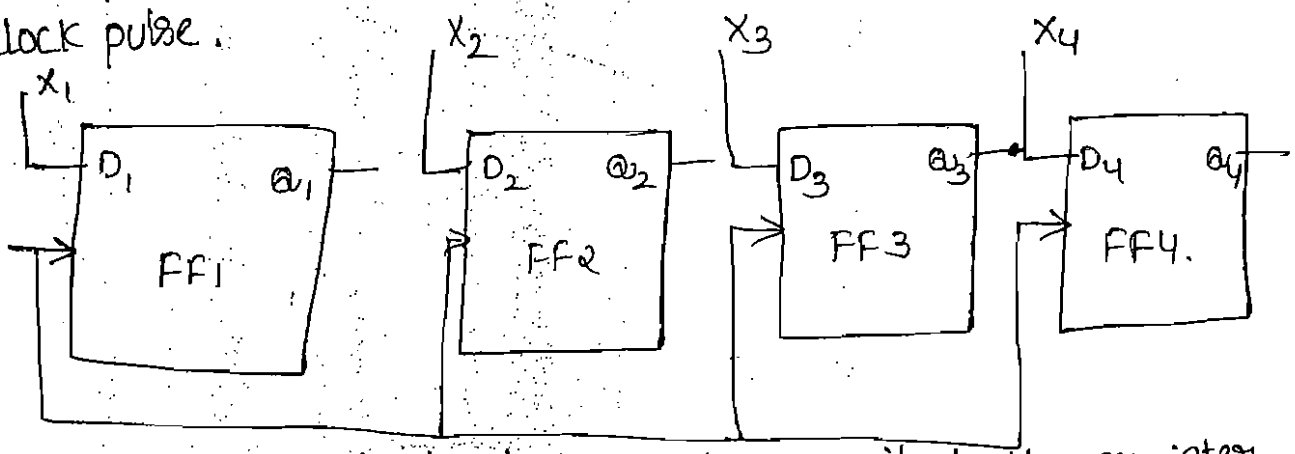
→ Finally $S_1S_0=11$ the binary information on the parallel

input lines is transferred into the register simultaneously during the next clock edge.

Buffer Register :-

Some registers do nothing more than storing a binary word. The buffer register is the simplest of registers. It simply stores the binary word. The buffer may be a controlled buffer. Most of the buffer registers use D-flip flops.

The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the input terminals. The input word is loaded into the register by the application of clock pulse.



Logic diagram of a 4-bit buffer register.

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

$$Q = X.$$

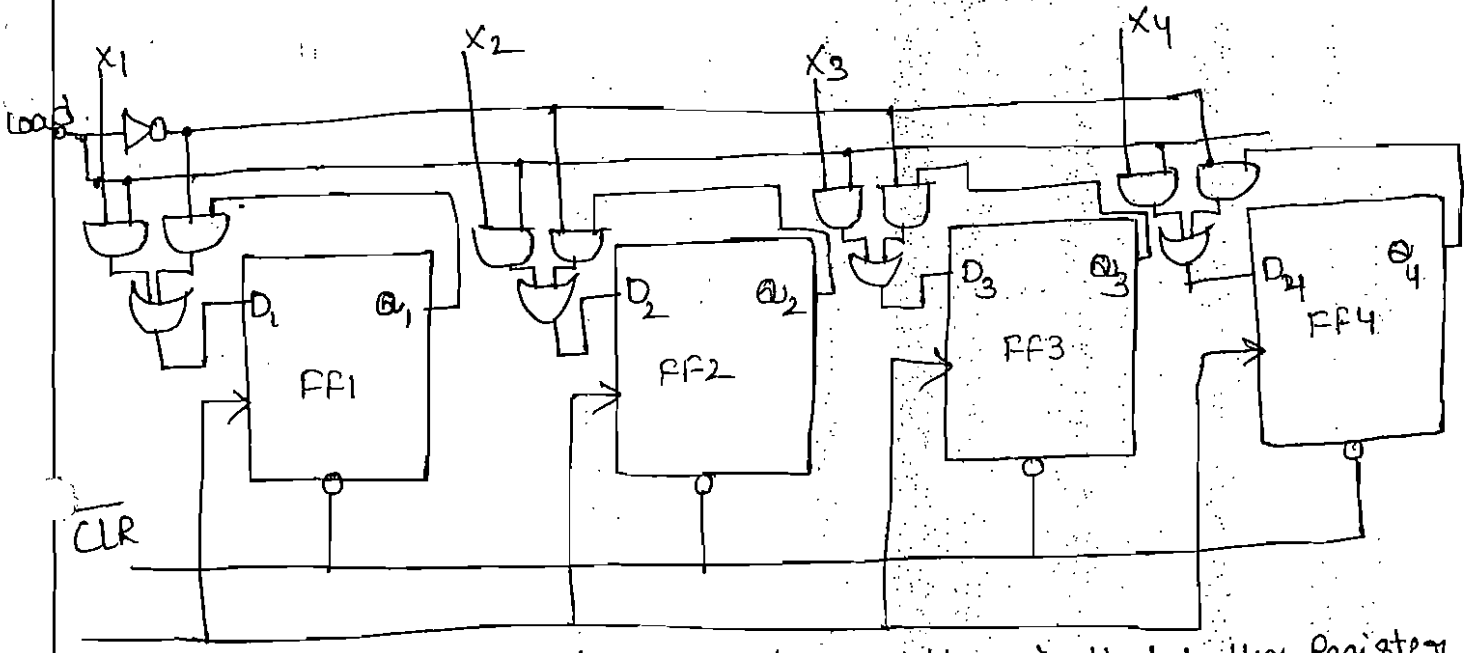
Controlled buffer Register :-

In Buffer register is too primitive to be of any use. it needs some control over the X bits, that is some way of holding them off until we are ready to store them.

→ If $\overline{CLR}$ goes low, all the FFs are RESET and the output becomes, Q = 0000.

→ when $\overline{CLR}$ goes high, the register is ready for action.

Load is the control input. When Load is high, the data bits X can reach the D inputs of FF's. At the positive-going edge of the next clock pulse, the register is loaded.



Logic diagram of a 4-bit controlled buffer Register.

When Load is low, the X bits cannot reach the FF's. At the same time, the inverted signal $\overline{Load}$ is high. This forces each flip-flop output to feed back to its data input. Therefore, data is circulated or retained on each clock pulse arrives. In other words, the contents of the register remain unchanged in spite of the clock pulses.

→ Longer buffer registers can be built by adding more FFS.

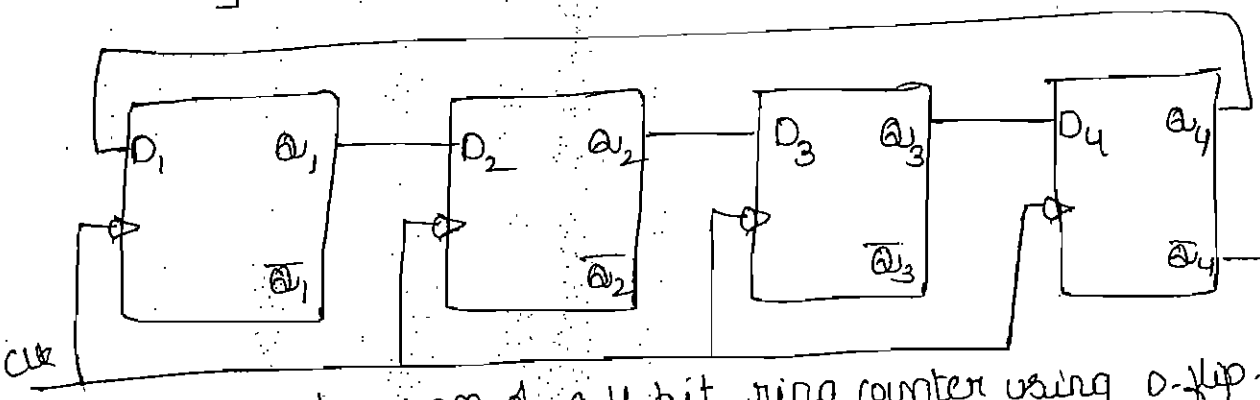
→ Shift Register counters :-

Shift register counters are obtained from serial-in Serial-out Shift-registers by providing feedback from the output of the last FF to the input of the first FF. These devices are called counters because they exhibit a specified sequence of states. The most widely used shift register

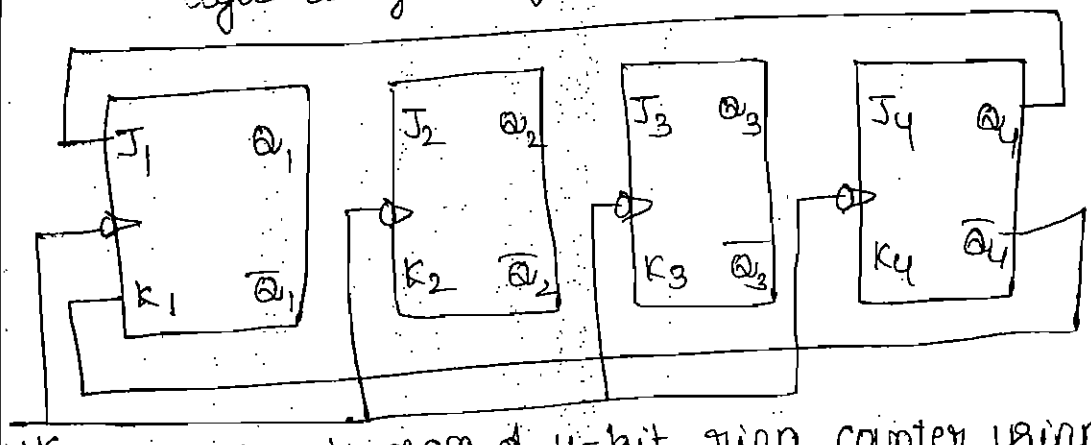
counter is the ring counter [simple ring counter)
twisted ring counter (Johnson counter & the switch-tail

Ring counter :-

This is the simplest shift register counter. The basic ring counter using D-FFs. The realization of this counter using J-K FFs is shown below figure. Its flip-flops are arranged as in a normal shift register, that is the Q output of each state is connected to the D input of the next stage, but the Q output of the last FF is connected back to the D-input of the first FF such that the array of FFs is arranged in rind and, therefore, the name ring counter.



Logic diagram of a 4-bit ring counter using D-flip-flops.

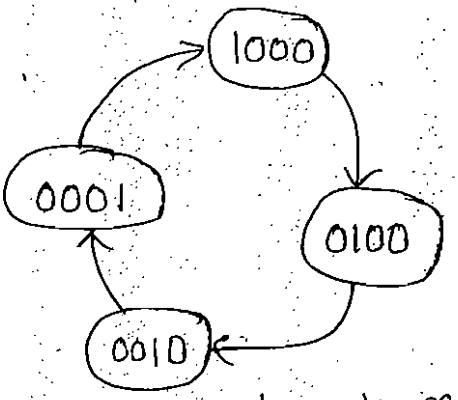


Logic-diagram of 4-bit ring counter using J-K flip-flops

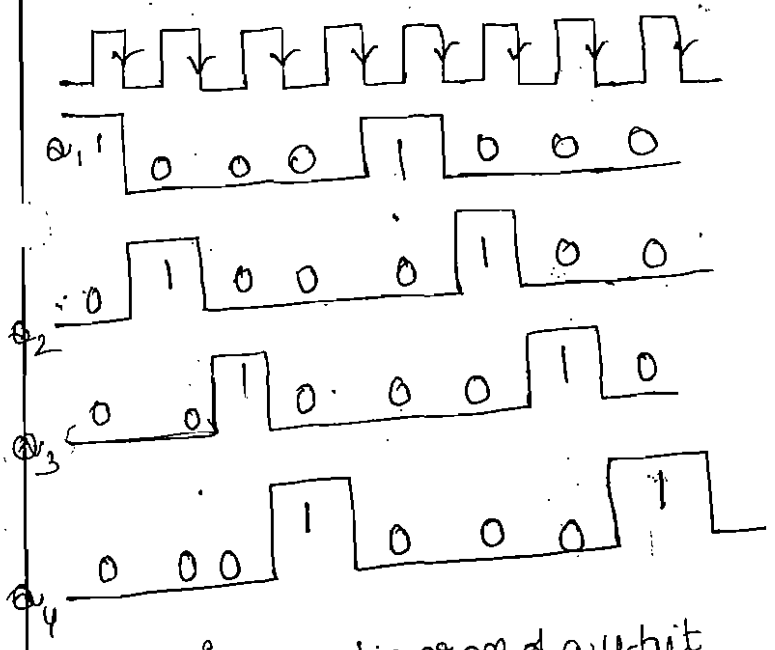
In most instances, only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially, the first FF is present to a 1. So, the initial state is 1000, that is $Q_1 = 1, Q_2 = 0, Q_3 = 0$

and $Q_4 = 0$. After each clock pulse, the contents of the register are shifted to the right by one bit and Q_4 is shifted back to Q_1. The sequence repeats after four clock pulses. The number of distinct states in the ring counter.

Twisted Ring Counter :-



State - diagram.



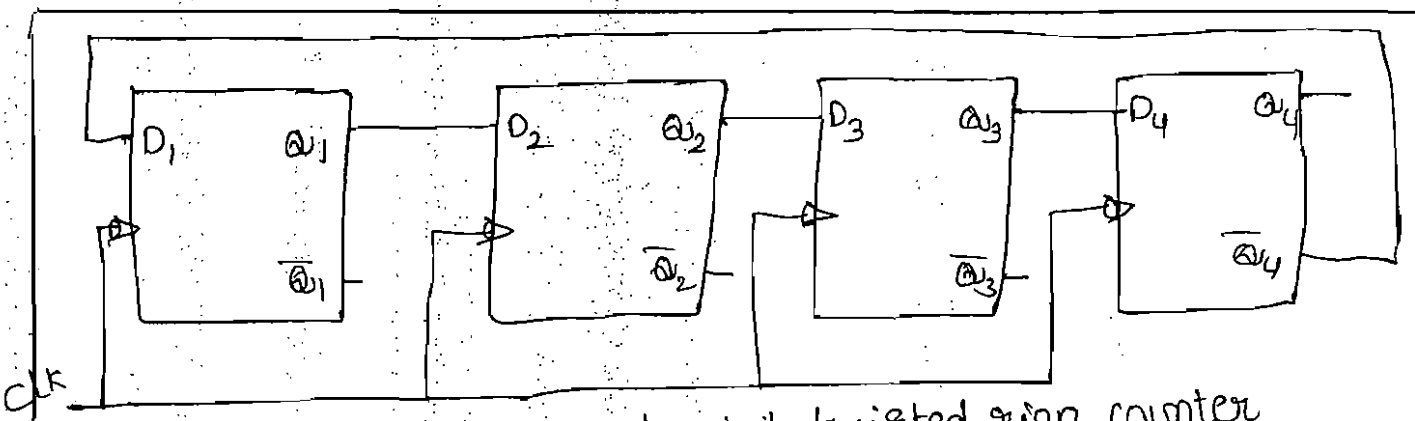
Timing diagram of a 4-bit ring counter.

Q_1	Q_2	Q_3	Q_4	After clk pul
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7.

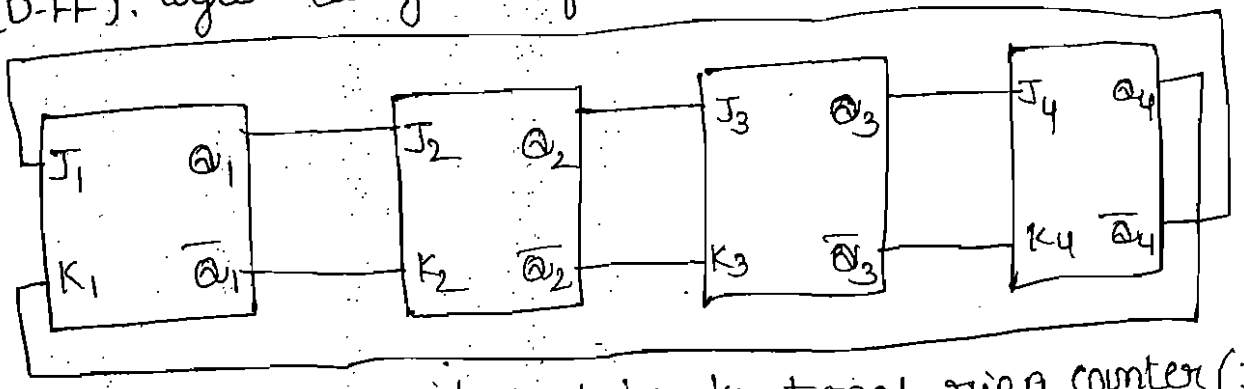
Sequence table.

Twisted Ring Counter :-

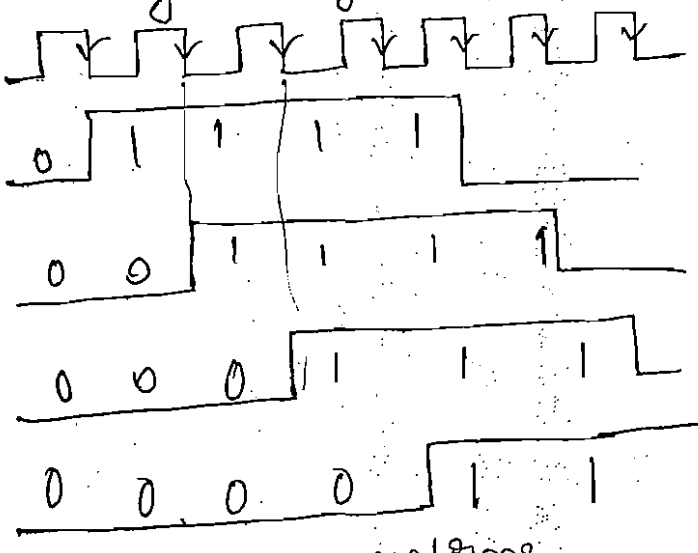
The counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. The Q output of each stage is connected to the D input of the next stage, but the $\bar{Q}$ output of the last stage is connected to the D input of first stage, therefore, the name twisted ring counter.



(D-FF). logic - diagram of 4-bit twisted ring counter



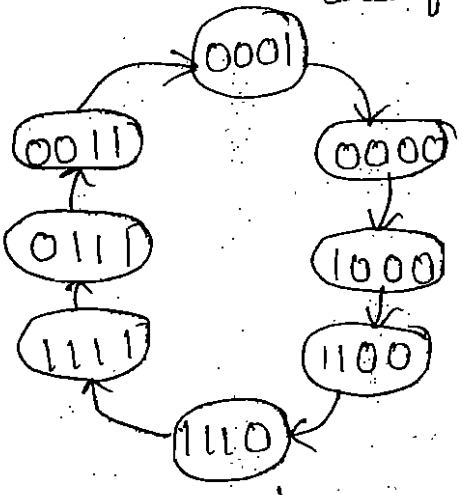
logic - diagram of 4-bit bi-directional ring counter (J-K FF)



waveforms

Q_1	Q_2	Q_3	Q_4	After clk pulse
0	0	0	0	0
1	0	0	0	1
1	1	0	0	2
1	1	1	0	3
1	1	1	1	4
0	1	1	1	5
0	0	1	1	6
0	0	0	1	7
0	0	0	0	8
1	0	0	0	9

Sequence table.



State diagram

let initally all the FFS be reset, that is the state of the counter be 0000. After each clock pulse, the level of Q_1 is shifted to Q_2, the level of Q_2 to Q_3, Q_3 to Q_4 and the level of $\overline{Q_4}$ to Q_1 and the sequence is repeated after every eight clock pulses.

→ An n FF Johnson counter can have 2n unique states and can count up to 2n pulses. So, it is a mod-2n counter.

Applications of flip flop :-

1. parallel data storage
2. serial data storage
3. Transfer of data.
4. serial - to - parallel conversion
5. parallel - to - serial conversion
6. counting
7. frequency division.

Applications of shift register :-

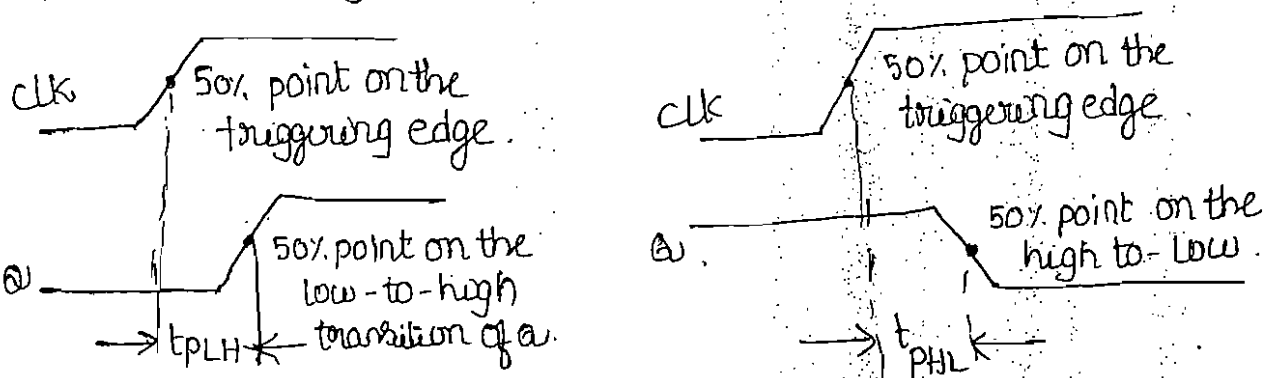
1. Time delays
2. serial / parallel data conversion
3. Ring counters
4. universal Asynchronous receiver transmitter.

Flip-flop operating characteristics :-

propagation delay time :- The output of a flip-flop will not change state immediately after the application of the clock signal or a synchronous inputs. The time interval between the time of application of the triggering edge or Asynchronous inputs and the time at which the output actually makes a transition is called the "propagation delay" time of the flip-flop. It is usually in the range of a few ns to 1μs.

→ propagation delay t_{PLH} measured from the triggering of the clock pulse to the low-to-high transition of the output.

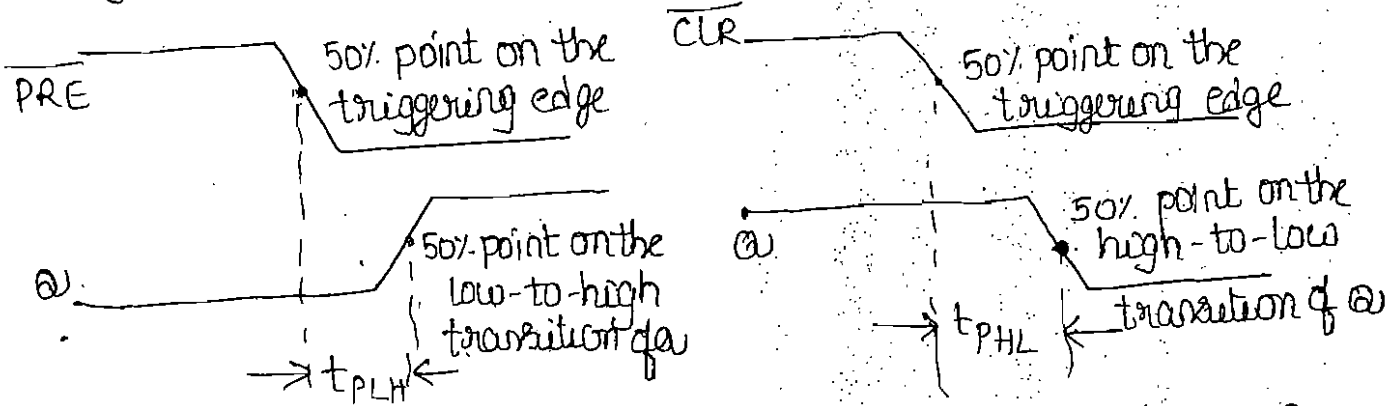
→ propagation delay t_{PHL} measured from the triggering of the clock pulse to the high-to-low transition of the output.



propagation delays t_{PLH} and t_{PHL} w.r.t clk.

→ Propagation delay t_{PLH} measured from the PRESET input to the low-to-high transition of the output

→ propagation delay t_{PHL} measured from the CLEAR input to the high-to-low transition of the output



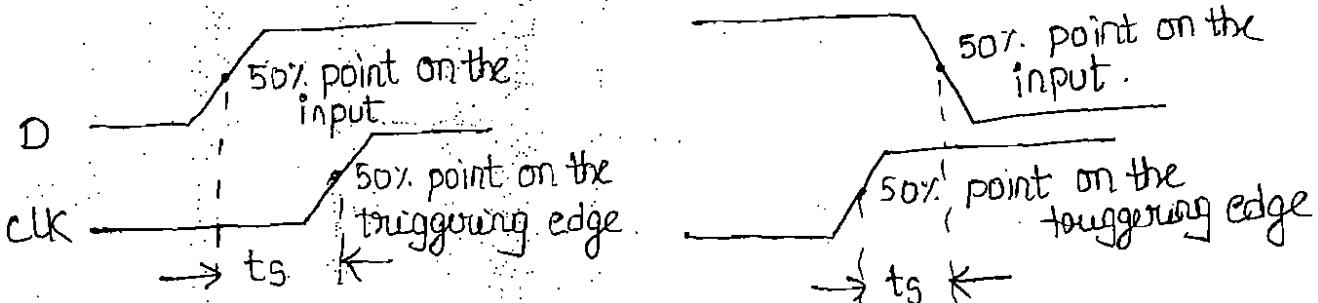
propagation delays t_{PLH}, t_{PHL} w.r.t. PRESET and CLEAR

Set-up-time :- The set-up-time (t_s) is the minimum time for which the control levels need to be maintained constant on the input terminals of the flip-flop, prior to the arrival of the triggering edge of the clock pulse

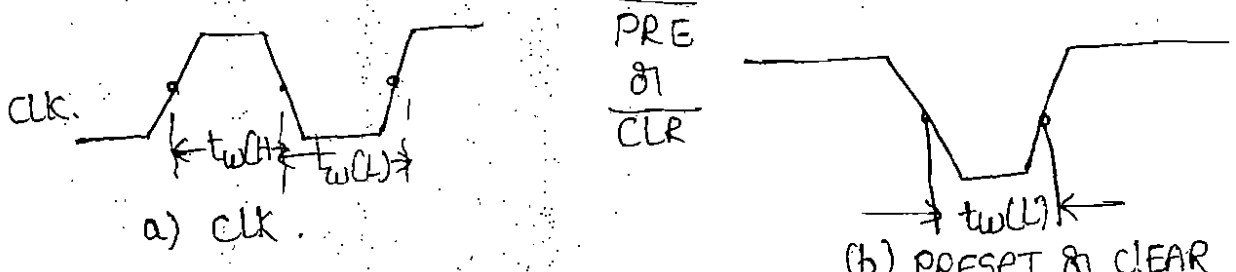
hold time :- The hold time (t_h) is the minimum time for which the control signals need to be maintained constant at the input terminals of the flip flop, after the arrival of the triggering edge of the clock pulse.

maximum clock frequency :- The maximum clock frequency (f_{max}) is the highest frequency at which a flip flop can be reliably triggered. If the clock frequency is above this maximum, the flip-flop would be unable to respond quickly enough and its operation will be unreliable. The f_{max} limit will vary from one flip flop to another.

Pulse widths :- The manufacturer usually specifies the minimum pulse widths for the clock and a synchronous inputs. For the clock signal, the minimum High time $t_{w(H)}$ and the minimum low time $t_{w(L)}$ are specified and for asynchronous inputs.



Set-up time and hold time for a D-flip-flop.



minimum pulse widths.

Clock transition times:- For reliable triggering, the clock waveform transition times (rise and fall times) should be kept very short. If the clock signal takes too long to make the transitions from one level to other, the flip-flop may either trigger erratically or not trigger at all.

power dissipation:- The power dissipation of a flip-flop is the total power consumption of the device. It is

$$P = V_{CC} \cdot I_{CC}$$

V_{CC} = Supply voltage

I_{CC} = current

The power dissipation of a flip-flop is usually in mw.

If a digital system has N-flip flops and if each flip-flop dissipates P mw of power, the total power requirements.

$$P_{TOT} = N \cdot V_{CC} \cdot I_{CC}$$

$$= (N \cdot P) \text{ mw}$$