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.

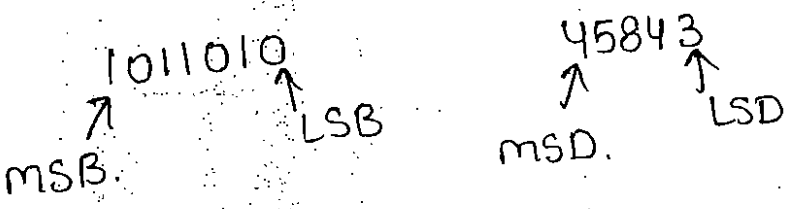
Radix - NO of symbols presented in a respective number system is called radix.

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

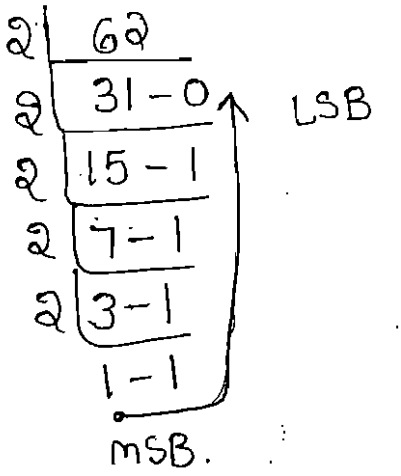
→ The integer and fractional parts of a decimal number are treated separately for the conversion and then the results are combined.

→ 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.

$105_{10} = 1101001_2$

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

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	

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

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

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 ↑
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.

→ $(563)_8 \longrightarrow (101110011)_2$

→ $(725)_8 \longrightarrow (111010101)_2$

→ $(326)_8 \longrightarrow$ 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.

→ $(1101101.01101)_2 \longrightarrow (155.32.)_8$

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.

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

→ $(1101101010101.10101010)_2 \rightarrow (\quad)_{16}$.

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	1011	1111	. 1010	1110
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:- $(1321)_5 \longrightarrow (211)_{10}$.

(3 2 1 0 — 5)

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

$= 125 + 75 + 10 + 1$

$= (211)_{10}$

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	01010010
	+1	+1
2's complement	10010110	01010010

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 → 00100100 → 11011011
20 → 00010100 → 11101011

```
  11011011
  11101011
 ---------
1 11000110
  ↳       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 → 00100100 → 11011011
00010100 → 11101011

```
  11011011
  11101011
 ---------
1 11000110
        1
 ---------
  11000111  → negative.
```

00111000 = 56.

-36-20 = -56 (11000111)

Example 4

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

+36 → 00100100
-36 → 11011011
+20 00010100

```
  11011011
  00010100
 ---------
  11101111  ⟶  -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, 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	7 9

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

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
 ----
 1251  (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.

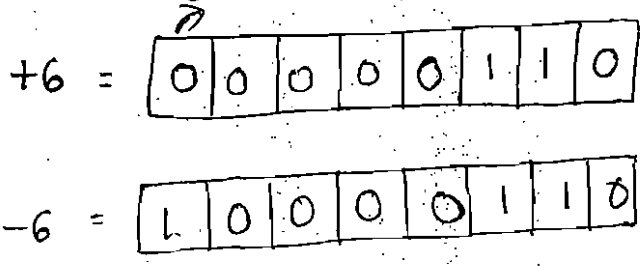
→ 2's complement representation.

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.

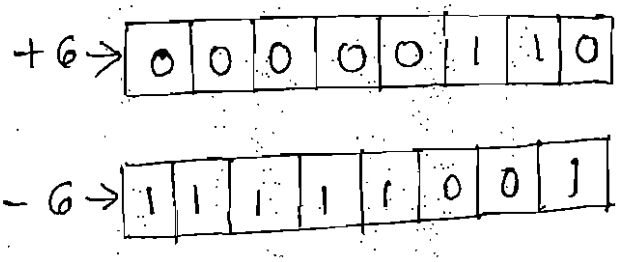
In eight bit representation, the MSB indicates the sign and remaining seven bits represent the magnitude.

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.

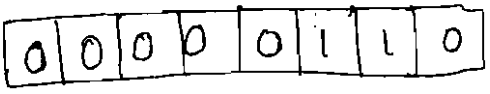


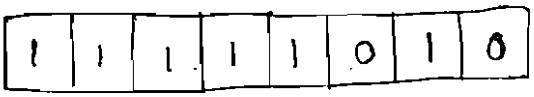
2's Complement Representation :-

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)

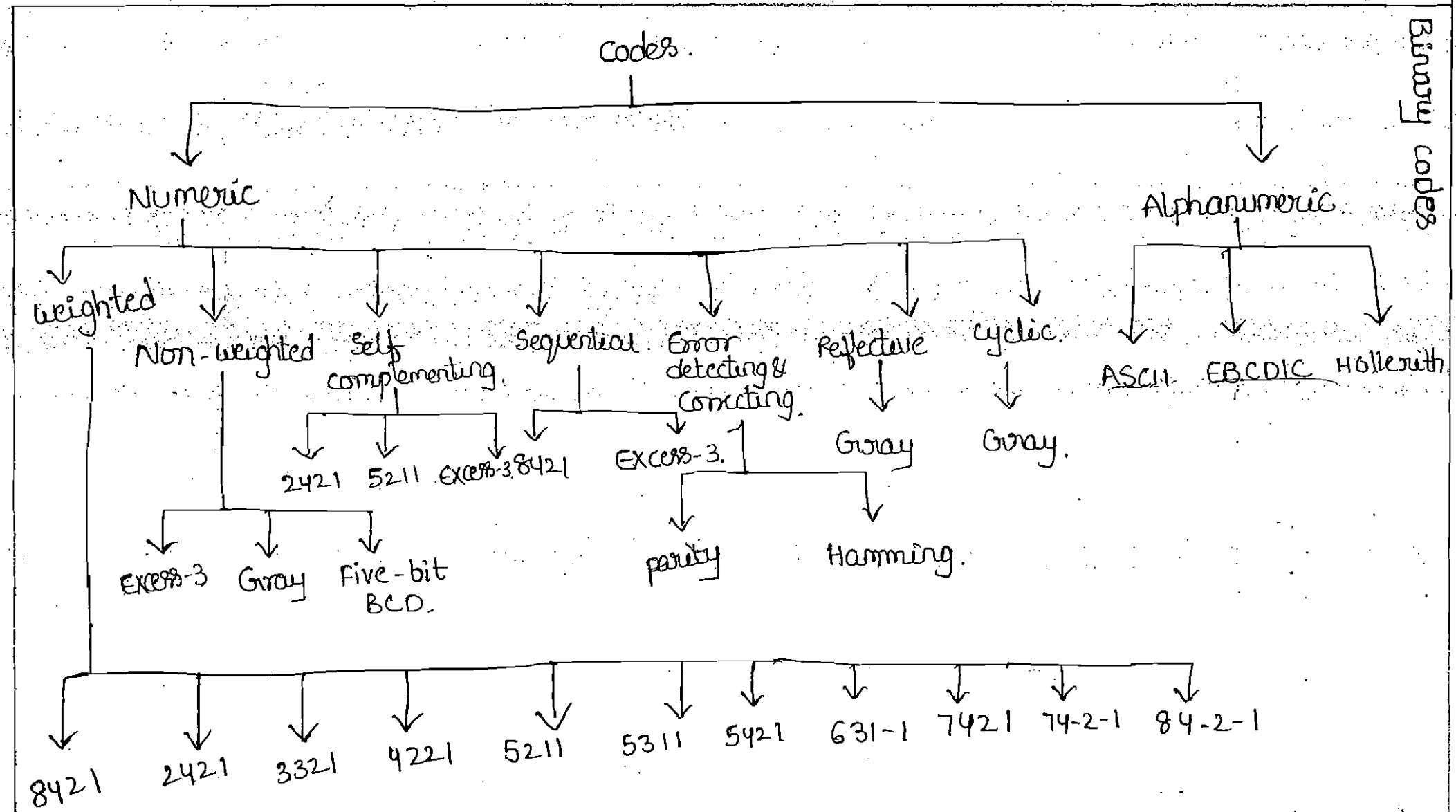
→ find the 2's complement of the number by adding 1 to 1's complement of the number.

+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

1000


Codes.
Numeric
Alphanumeric.
Weighted
Non-weighted
Self complementing.
Sequential.
Error detecting & Correcting.
Reflective
Cyclic.
ASCII
EBCDIC
Hollerith
2421
5211
Excess-3 8421
Excess-3.
Gray
Gray.
Excess-3
Gray
Five-bit BCD.
parity
Hamming.
8421
2421
3321
4221
5211
5311
542-1
631-1
7421
74-2-1
84-2-1

→ 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, $211 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.

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

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 - 142.5 = 857.4

857.4 - 629.4

236.8 → 0010 0011 0110. 1000

857.4 → 1000 0101 0111. 0100

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

23	13
33	33
56	46

1001 1100
-0011 -0011
0110 1001 (answer in XS-3)

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↷

II) 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

(1A)

1 0011 0011 1111

↳ (ignore the carry)

0011 0011 1111

+0011 +0011 -0011

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 ⊕ 1 ⊕ 1 ⊕ 0
  1   1   1   1
0 1 0 1
```

Convert binary 1001 to the Gray code.

```
1 ⊕ 0 ⊕ 0 ⊕ 1
  1   1   1   1
1 1 0 1
```

Gray to Binary conversion :-

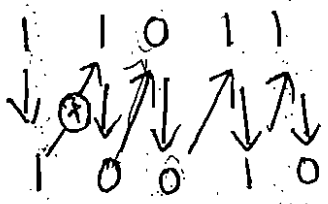
→ The MSB of the binary number is the same as the MSB of the Gray code

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

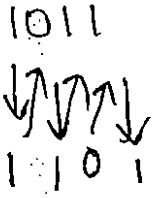
→ Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code.

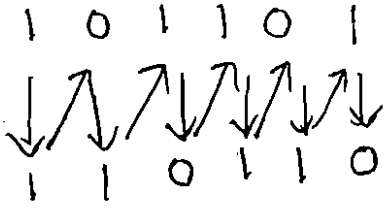
→ convert Gray to Binary.

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 :-

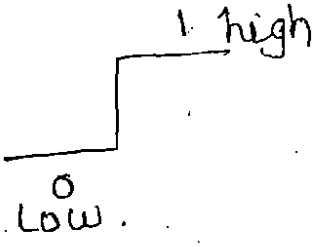
The logic Gates are the fundamental blocks of digital systems. The logic Gate is ability to make decisions (output). logic gate are electronic circuits because they are made up of a number of electronic devices and components.

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

Positive Logic :-

high → 1

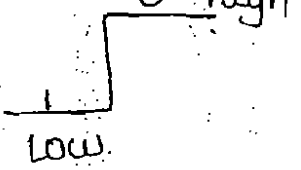
Low → 0.



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.

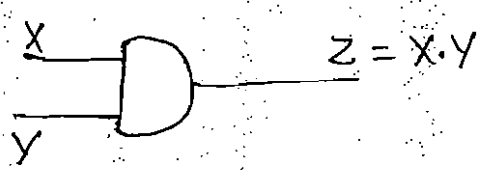
4. NAND Gate
5. NOR Gate

} 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

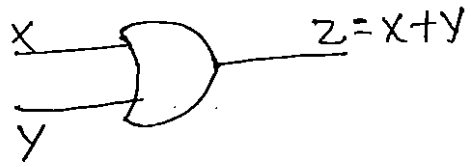
X	Y	Z=X.Y
0	0	0
0	1	0
1	0	0
1	1	1

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.

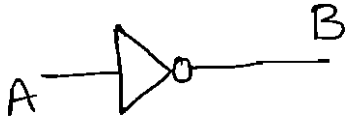
X	Y	Z=X+Y
0	0	0
0	1	1
1	0	1
1	1	1

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



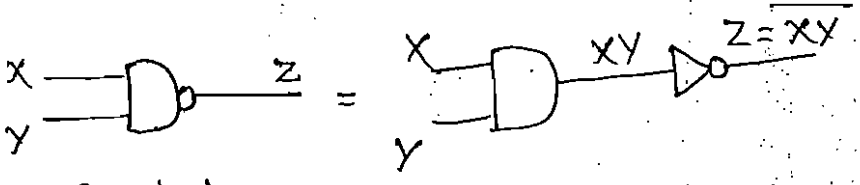
Symbol

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.

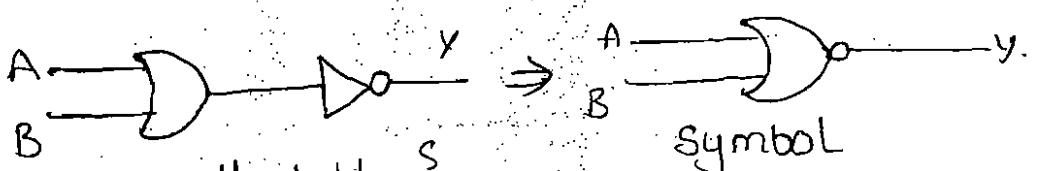
X	Y	$Z = \overline{XY}$
0	0	1
0	1	1
1	0	1
1	1	0



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

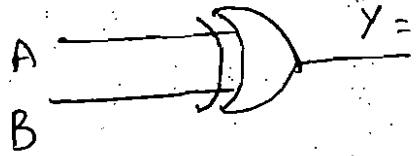
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

$Y = \overline{A+B}$

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



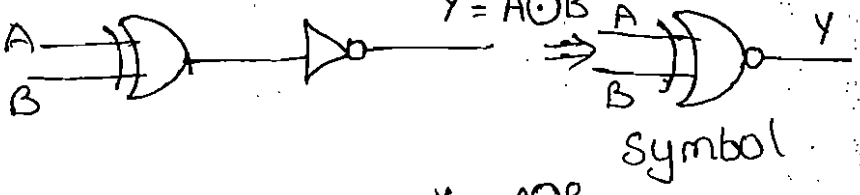
Symbol.

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 → summation 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 $(B \cdot \bar{B})$.

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

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

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

$$= M_1, M_2, M_3$$

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

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

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

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

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

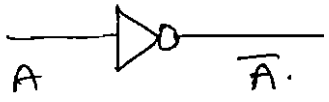
$$(000)\ (001)\ (010)\ (011)\ (100)\ (101)$$

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

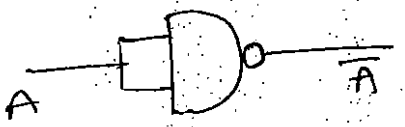
$$= \Pi M(0,1,2,3,4,5).$$

NAND - NAND Realization :-

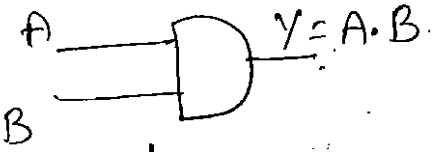
1) NOT Gate

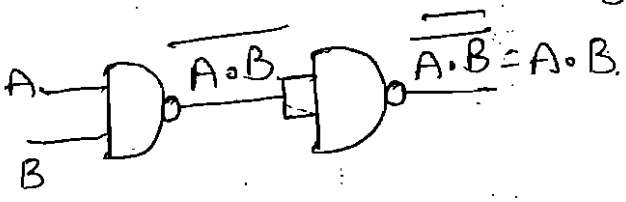


Using NAND gate.



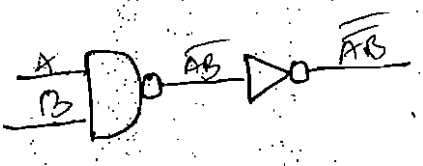
2) AND Gate.

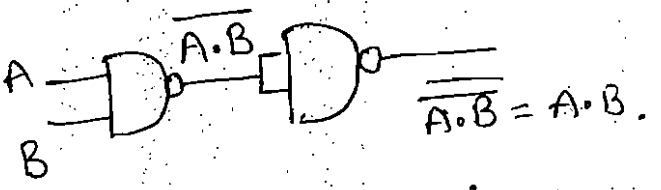




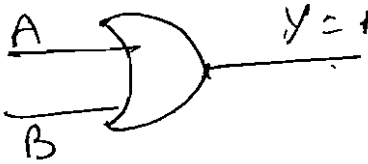
Step :- 1 add bubble on output

Step :- 2 add one NOT gate

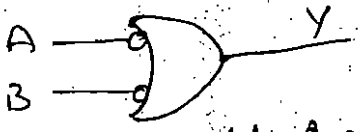




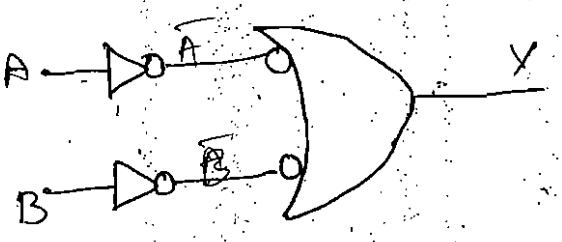
3) OR Gate

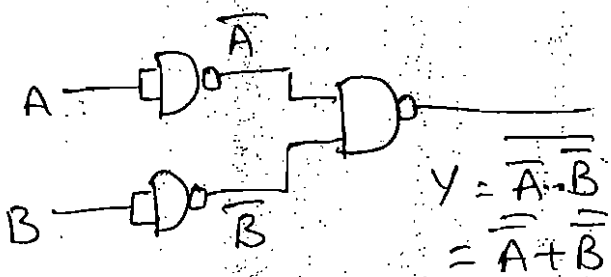


Step :- 1 add bubble on input



step 2 :- add not gate



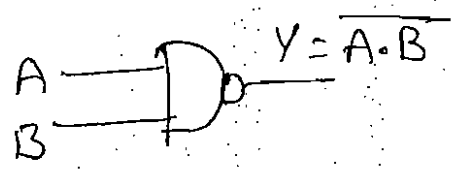


$Y = \overline{\overline{A} \cdot \overline{B}}$

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

$= A + B.$

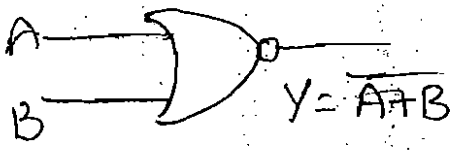
4) NAND gate

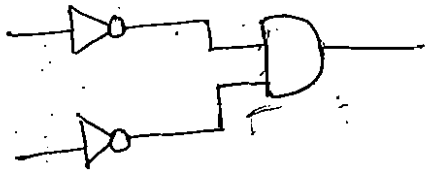


5) NOR Gate

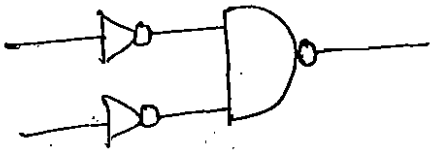
$Y = \overline{A+B}$.

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

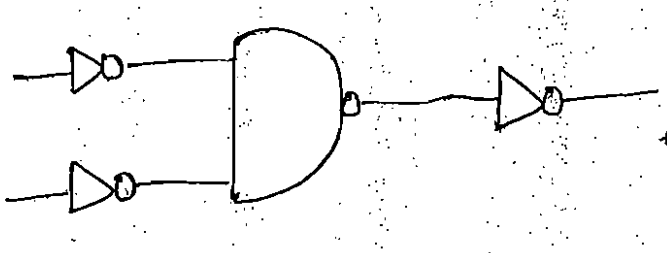




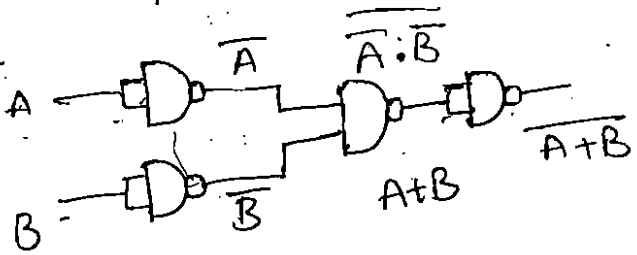
Step:-1



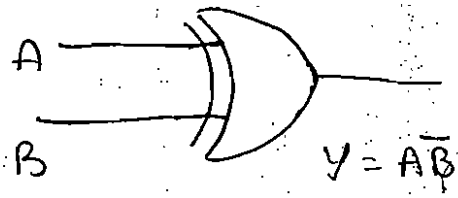
Step :- 2

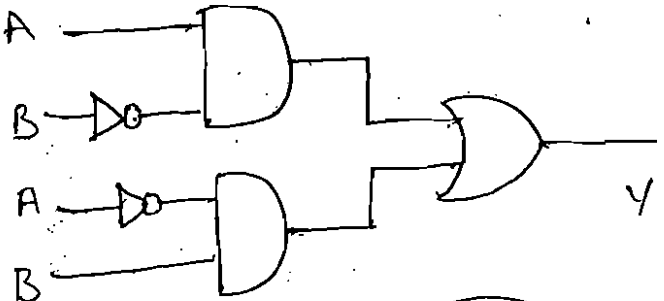


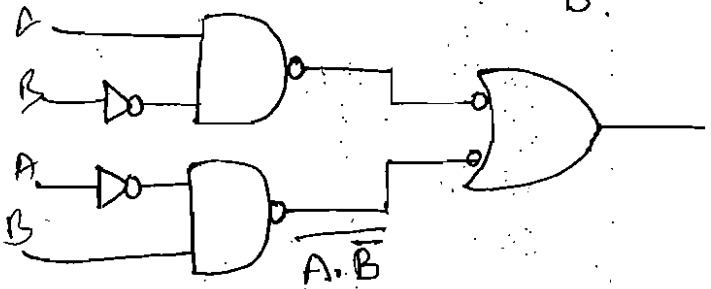
Step 3 :-

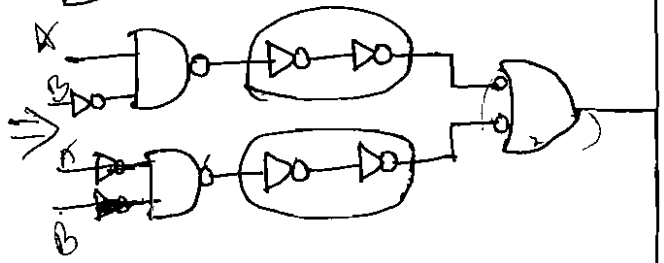


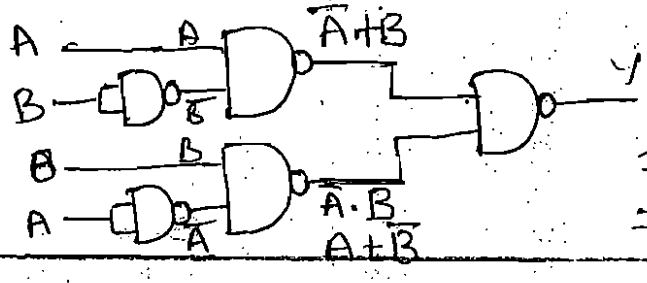
6) EX-OR Gate









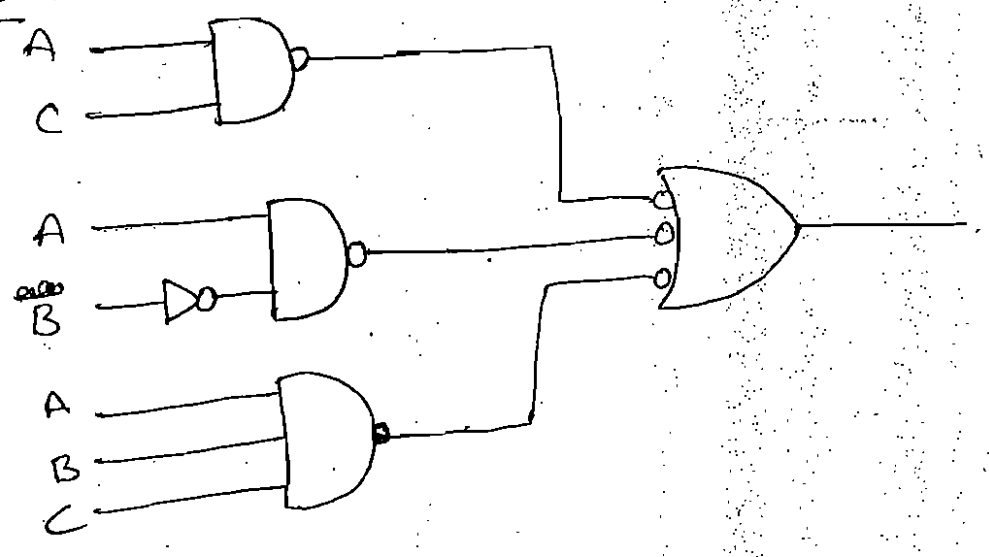


$Y = (\overline{A}+B)(A+\overline{B})$

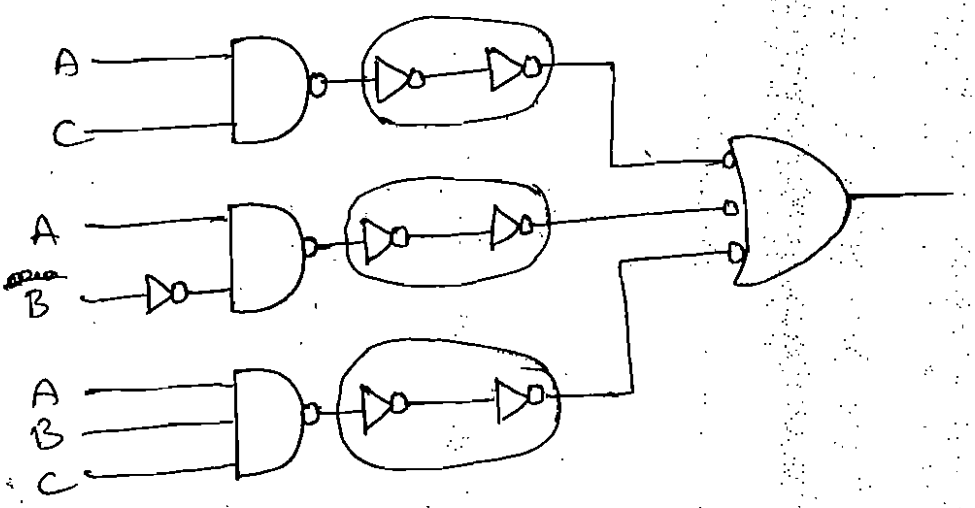
$= \overline{(\overline{A}+B)} + \overline{(A+\overline{B})}$

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

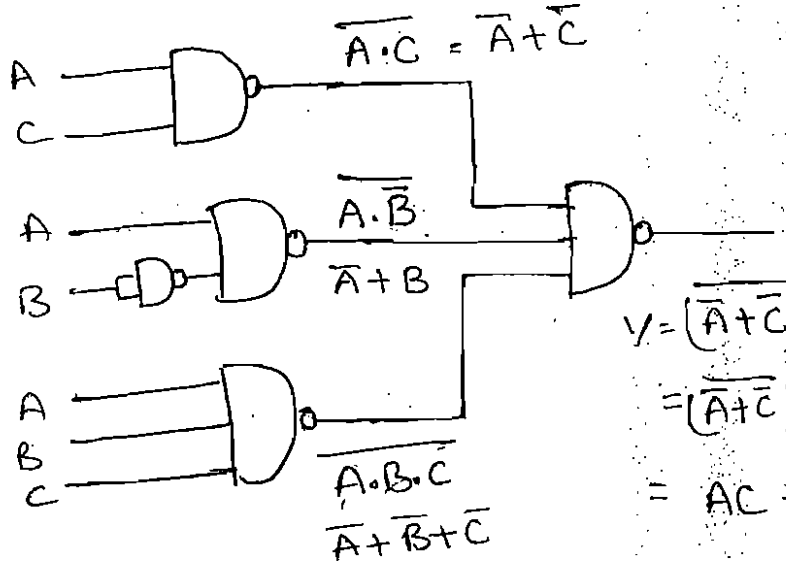
Step 1 :-



Step 2 :-



Step 3 :-



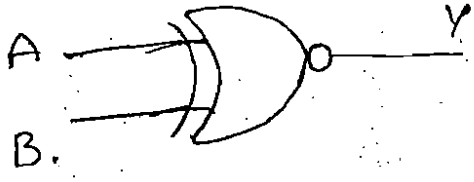
$$Y = \overline{(\overline{A}+\overline{C})(\overline{A}+B)(\overline{A}+\overline{B}+\overline{C})}$$

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

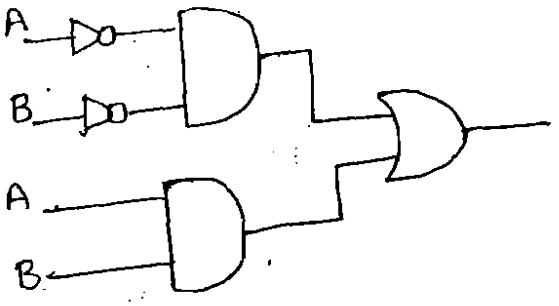
$$= AC + A\overline{B} + \overline{\overline{A}} \cdot \overline{\overline{B}} \cdot \overline{\overline{C}}$$

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

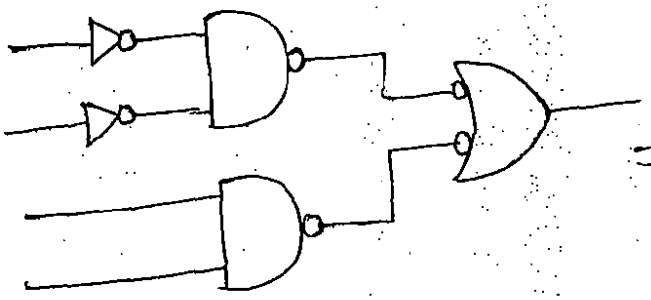
7) EX-NOR Gate :-

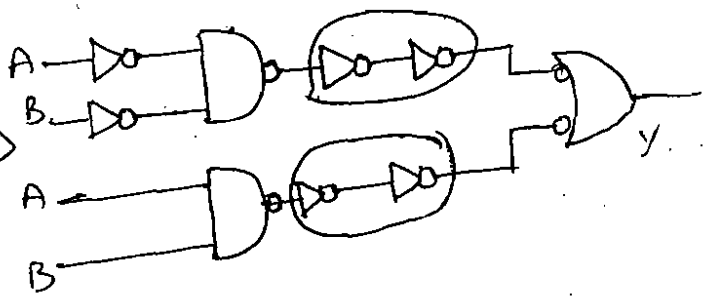


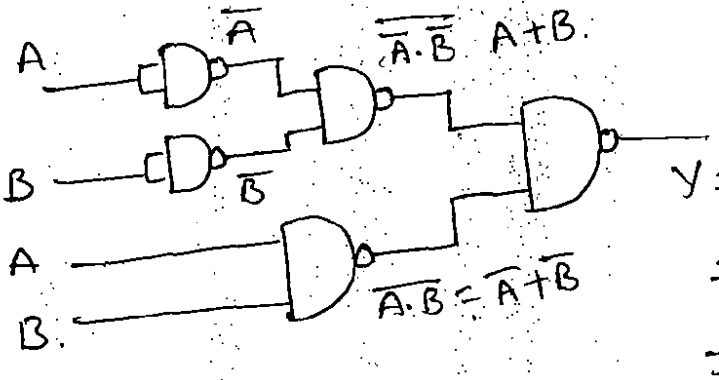
$Y = \overline{A}B + A\overline{B}$.



Step 1 :-







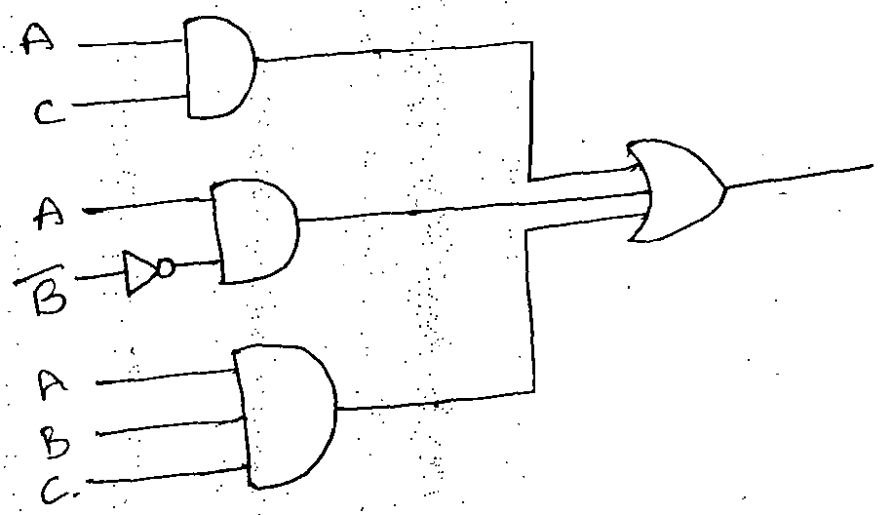
$Y = \overline{(A+B)(\overline{A}+\overline{B})}$

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

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

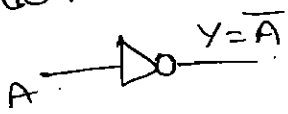
* Implement a given Boolean Expression by using NAND gate.

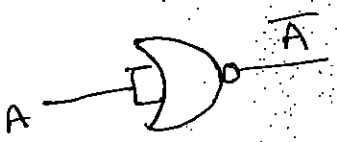
$Y = AC + A\overline{B} + ABC$.



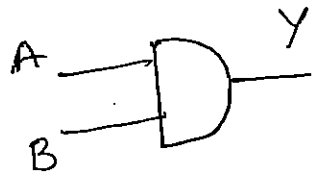
NOR - NOR Realization :-

1) NOT Gate

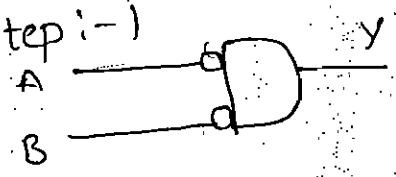




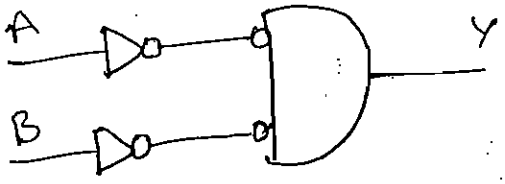
2) AND Gate.



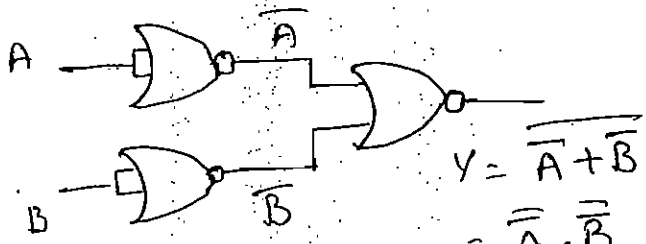
Step :-1



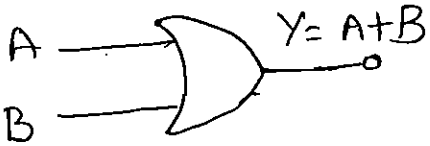
Step 2



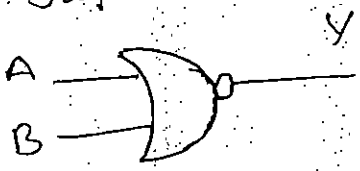
Step :3



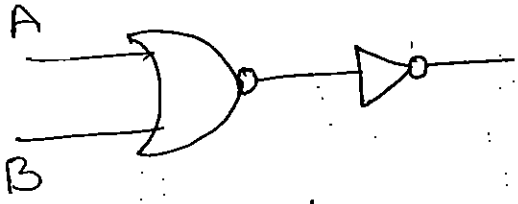
3) OR Gate

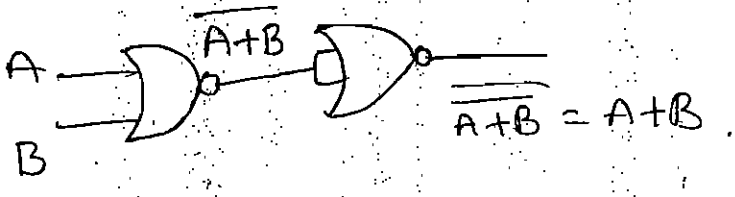


Step :-1



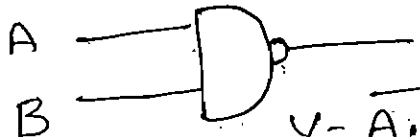
Step :- 2





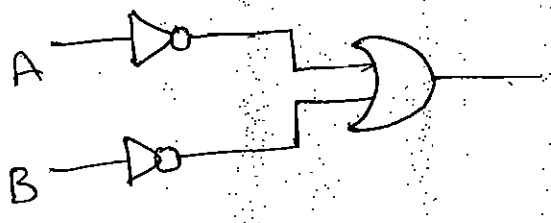
4) NAND Gate

Step :- 1

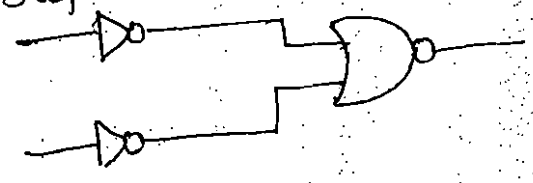


$Y = \overline{A \cdot B}$

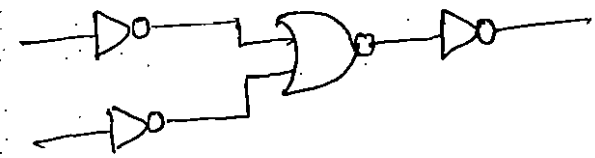
$= \bar{A} + \bar{B}$



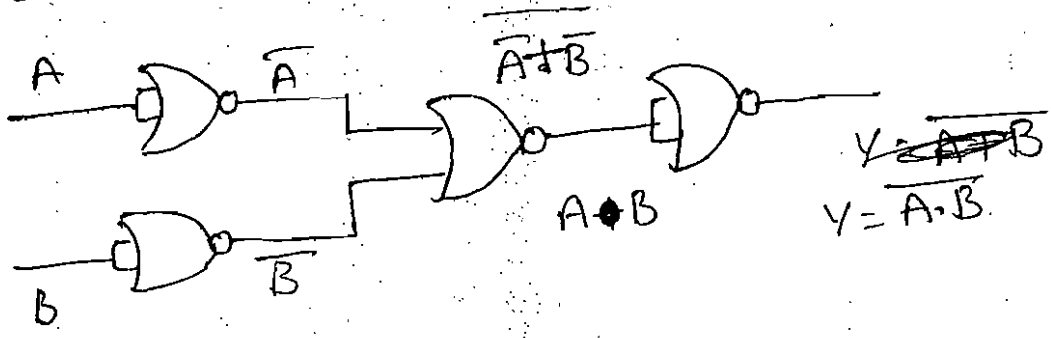
Step 2:-



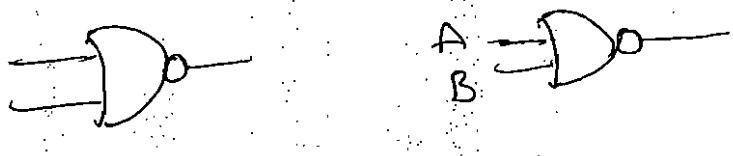
Step 3



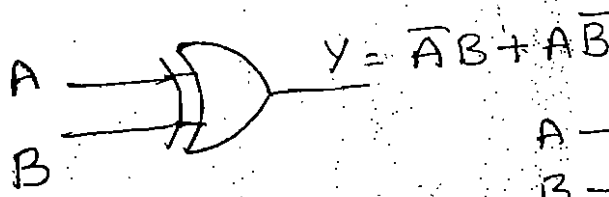
Step 4 :-



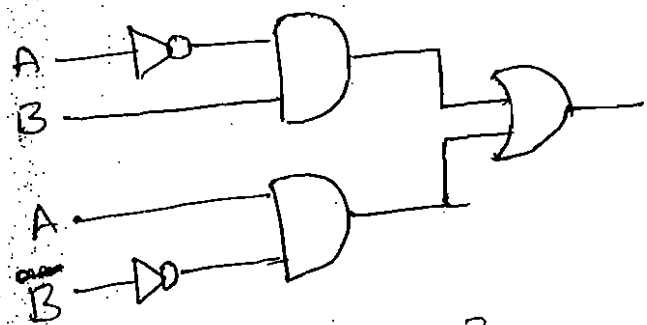
5) NOR Gate :-



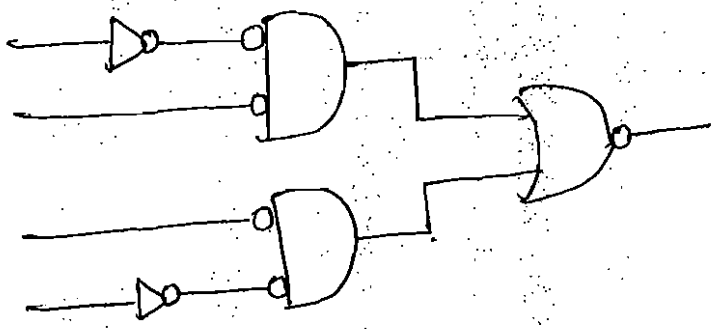
6) EX-OR Gate :-



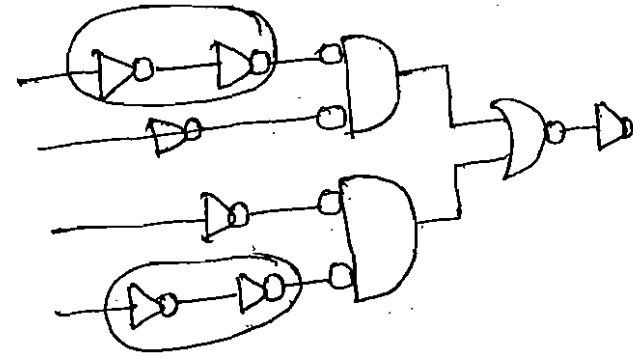
Step :- 1



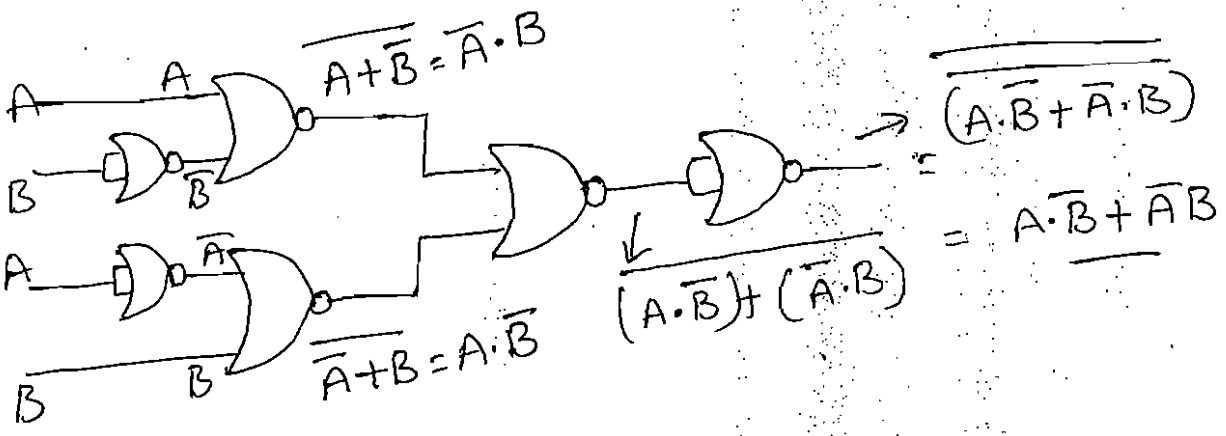
Step :- 2



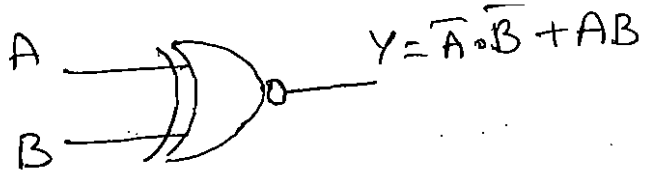
Step :- 3



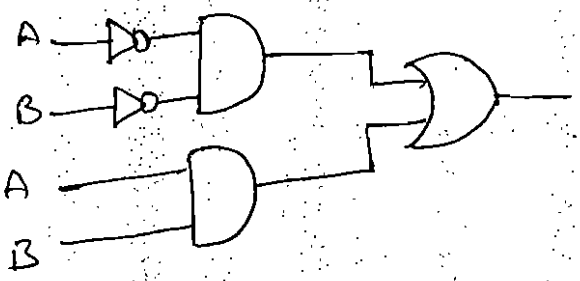
Step 4 :-



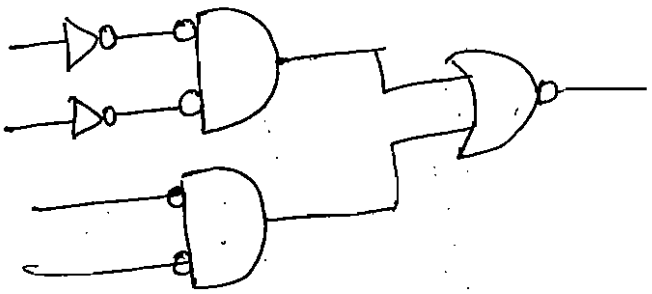
7) EX-NOR gate :-



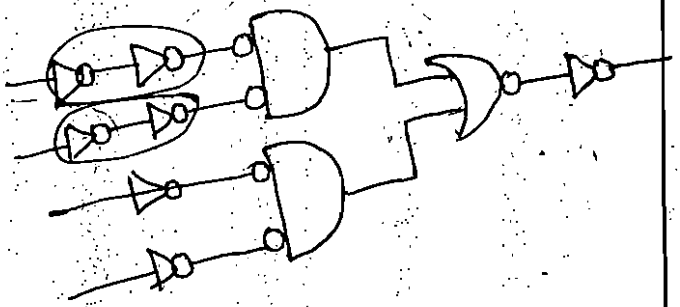
Step 1 :-



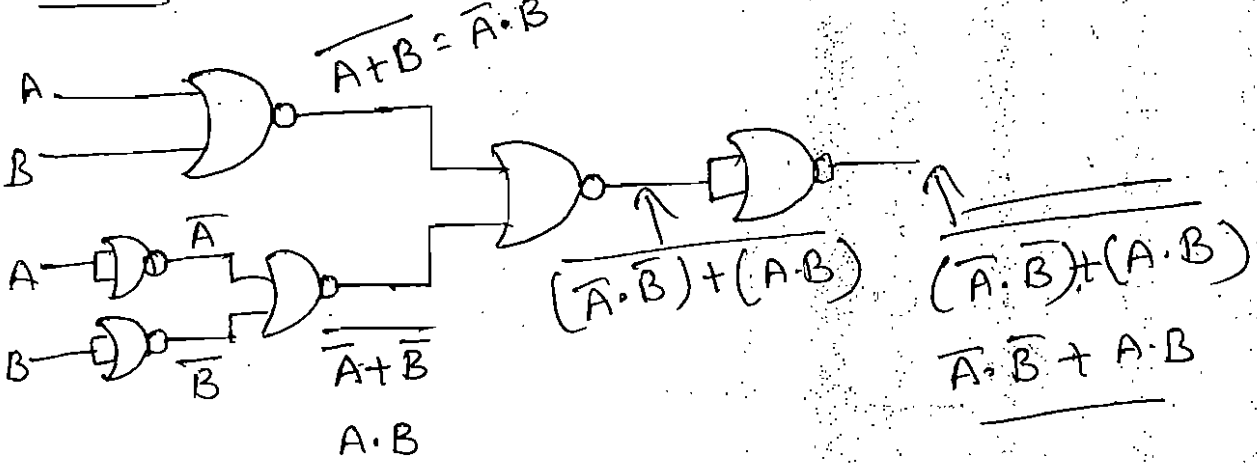
Step 2 :-



Step 3 :-

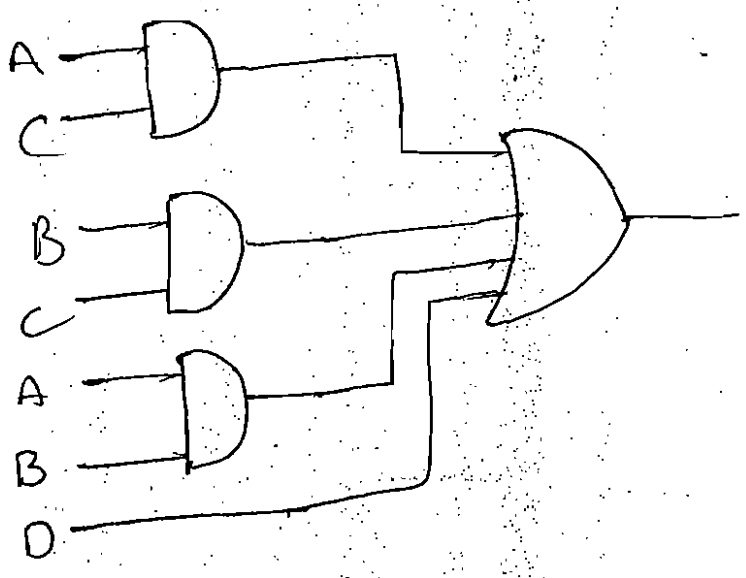


Step 4 :-

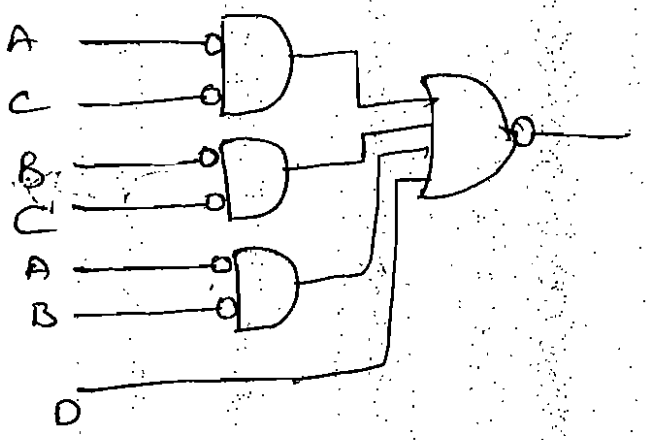


* $Y = AC + BC + AB + D$, implement Boolean Expression by using NOR Gate.

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

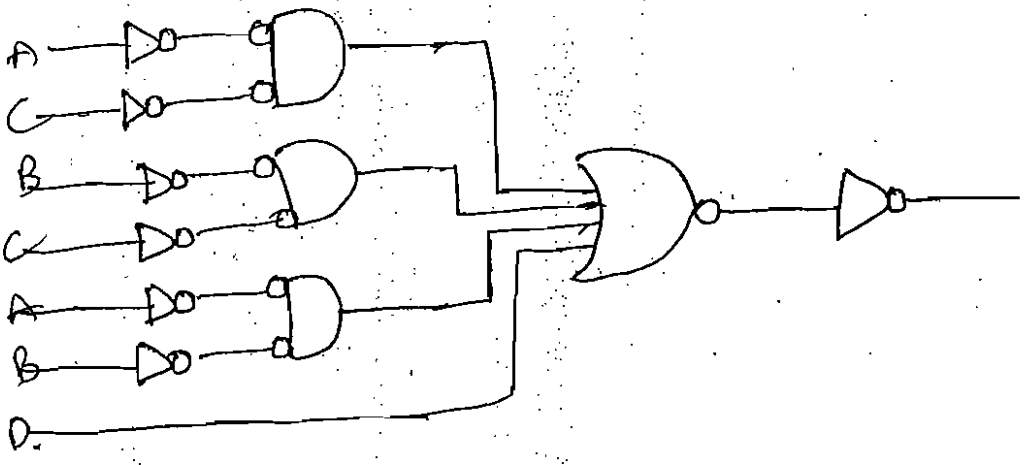


Step :- 1

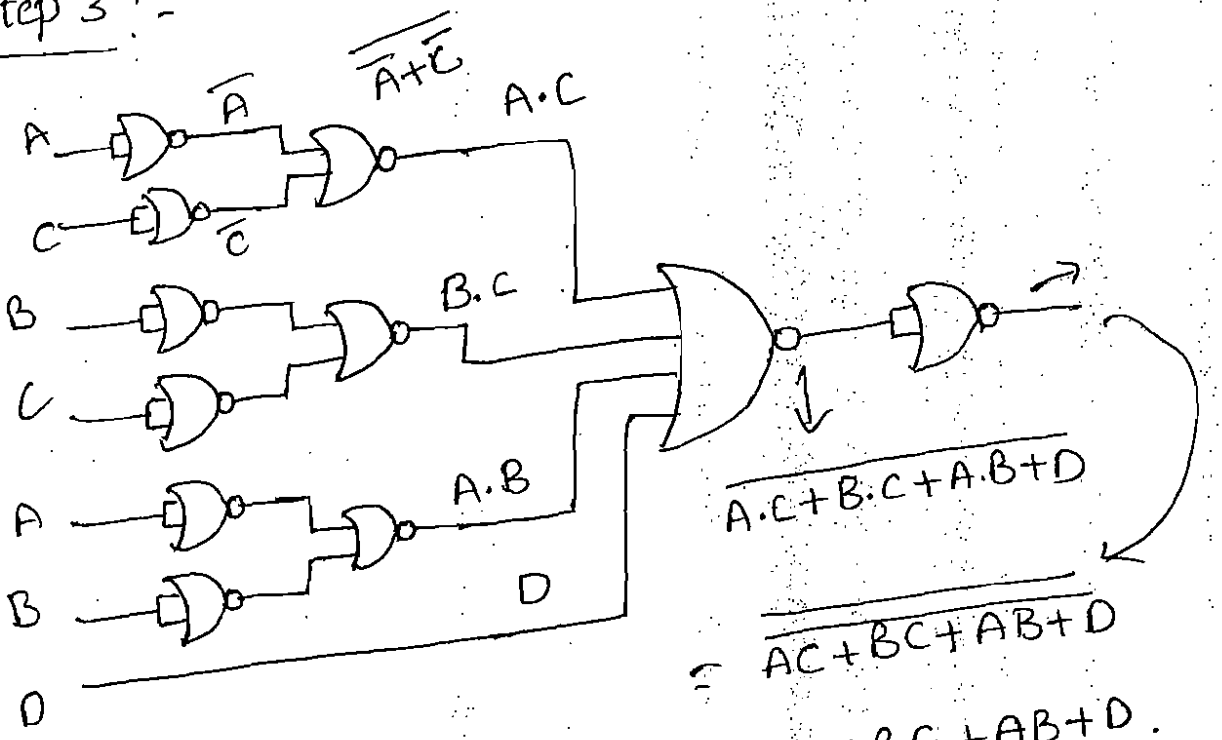


Step :- 2

Step :- 2



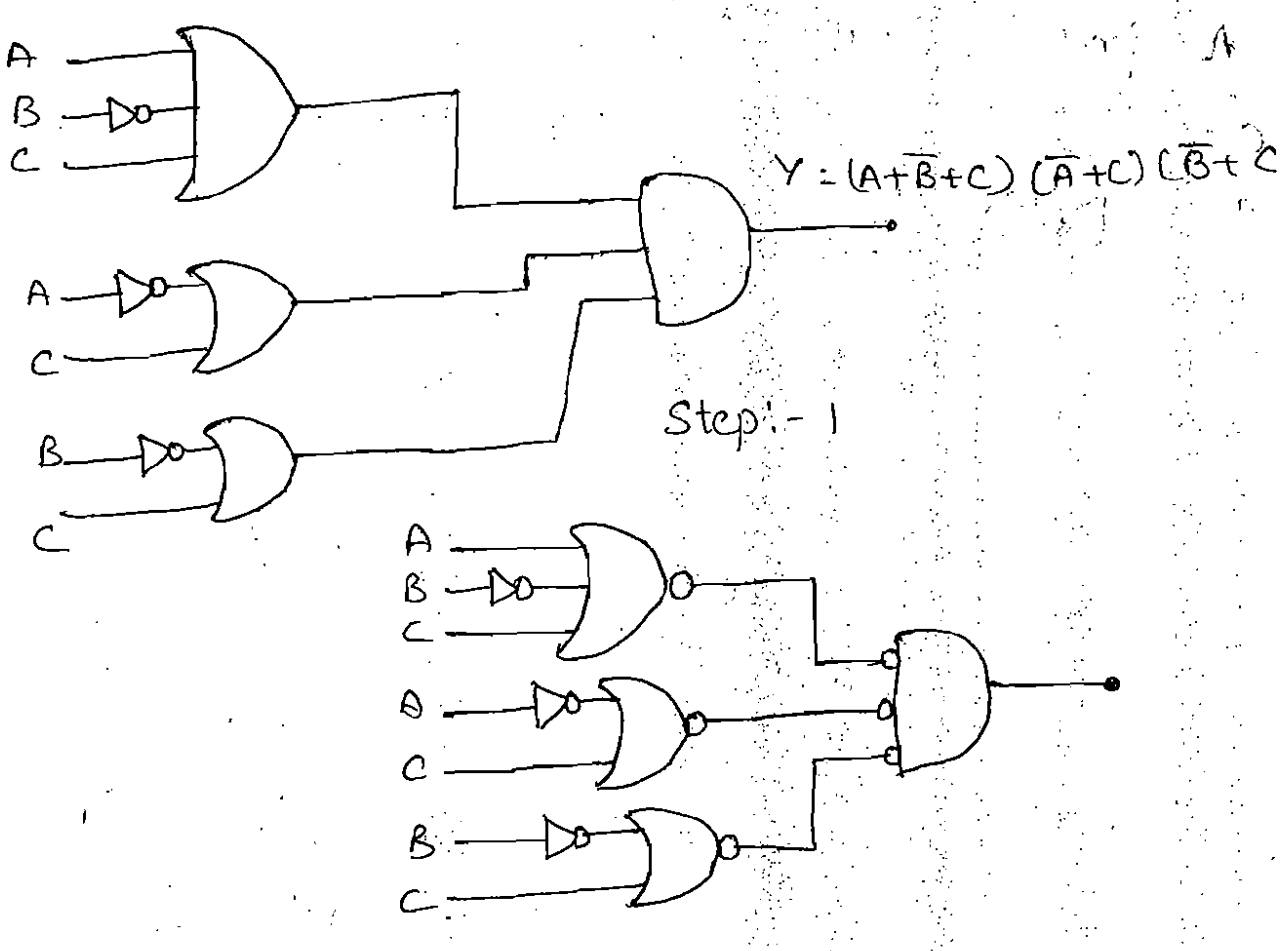
Step 3 :-



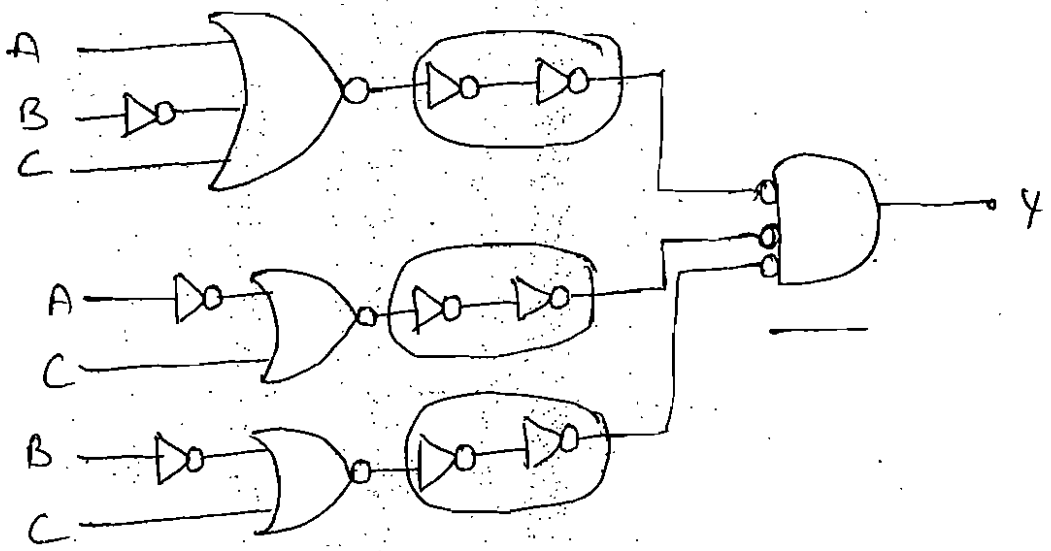
$= \overline{AC+BC+AB+D}$

$= AC+BC+AB+D.$

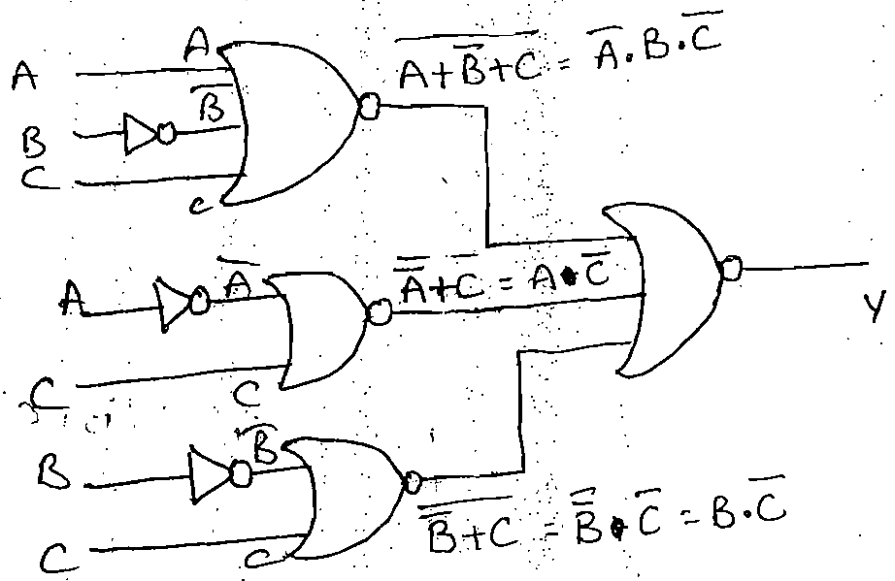
* implement the Boolean Expression by using NOR Gate.
$(A+\bar{B}+C)\ (\bar{A}+C)\ (\bar{B}+C)$.



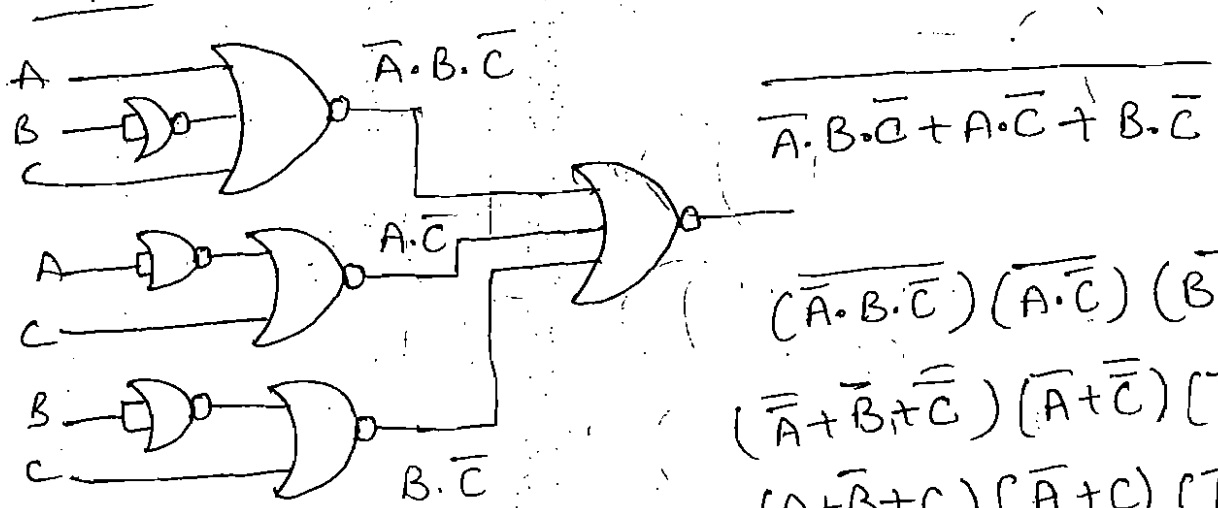
Step 2 :-



Step 3 :-



Step 4 :-



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

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

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

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

① $F(A,B,C,D) = \Sigma m(0, 2, 3, 6, 7, 8, 10, 12, 13)$ ①

→ list all minterms in binary form

minterm	Binary representation
m_0	0000
m_2	0010
m_3	0011
m_6	0110
m_7	0111
m_8	1000
m_{10}	1010
m_{12}	1100
m_{13}	1101

→ Arrange the minterms according to number of 1's

minterm	Binary representation
m_0	0000
m_2	0010
m_8	1000
m_3	0011
m_6	0110
m_{10}	1010
m_{12}	1100
m_7	0111
m_{13}	1101

→ Compare each binary number with every term in the adjacent next higher category & if they differ only by one position put a check mark & copy the term in the next column with '-' in the position that they differed.

minterms	Binary representation	minterms	Binary representation
0, 2	0 0 - 0 ✓		
0, 8	- 0 0 0		
2, 3	0 0 1 - ○		
2, 6	0 - 1 0		
2, 10	- 0 1 0		
8, 10	1 0 - 0 ✓		
8, 12	1 - 0 0		
3, 7	0 - 1 1		
6, 7	0 1 1 - ○		
12, 13	1 1 0 -		

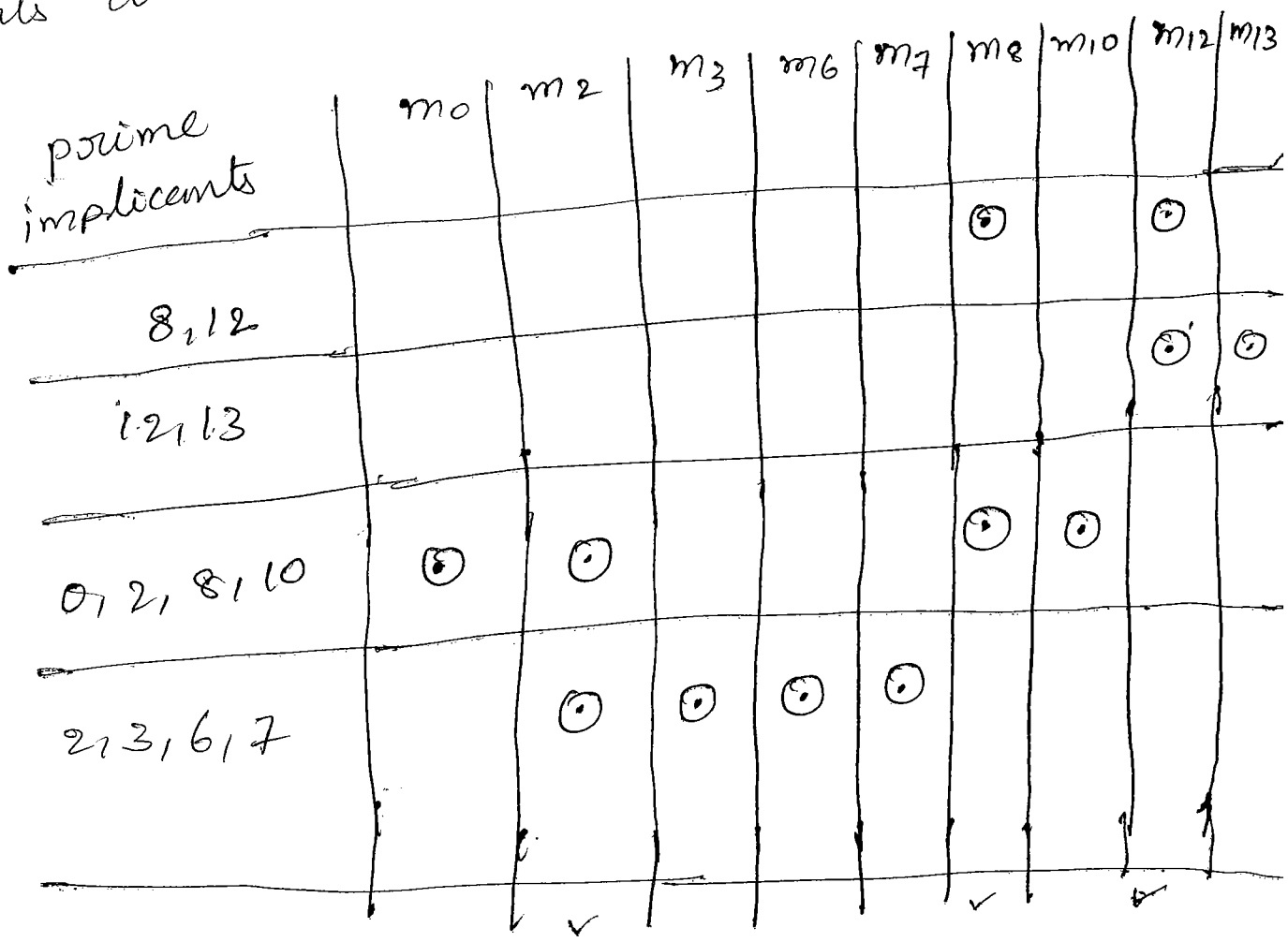
List the prime implicants

0, 2, 8, 10 - 0 - 0

2, 3, 6, 7 0 - 1 -

Prime implicants	Binary representation ②
2, 3, 6, 7	0 − 1 − → $\bar{A}C$
0, 2, 8, 10	− 0 − 0 → $\bar{B}\bar{D}$
12, 13	1 1 0 − → $AB\bar{C}$
8, 12	1 − 0 0 → $A\bar{C}\bar{D}$

Select the minimum number of prime implicants which must cover all the minterms.



$$F(A,B,C,D) = (0-1-) + (-0-0) + (110-)$$

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

(3)

(8) $Y(W,X,Y,Z) = \Sigma m(1,2,3,5,9,12,14,15) + \Sigma d(4,8,11)$

minterm	Binary representation
m_1	0 0 0 1
m_2	0 0 1 0
m_4	0 1 0 0
m_8	1 0 0 0
m_3	0 0 1 1
m_5	0 1 0 1
m_9	1 0 0 1
m_{12}	1 1 0 0
m_{11}	1 0 1 1
m_{14}	1 1 1 0
m_{15}	1 1 1 1

Compare each binary number with every term in the adjacent next higher category

minterm	Binary representation
1, 3	0 0 - 1 ✓
1, 5	0 - 0 1
1, 9	- 0 0 1
2, 3	0 0 1 -

1, 3, 9, 11	- 0 - 1

4,12	- 1 0 0
8,9	1 0 0 -
8,12	1 - 0 0
3,11	- 0 1 1

Prime implicants.

1,5 ⟶ 0 - 0 1

2,3 ⟶ 0 0 1 -

4,5 ⟶ 0 1 0 -

4,12 ⟶ - 1 0 0

8,9 ⟶ 1 0 0 -

8,12 ⟶ 1 - 0 0

12,14 ⟶ 1 1 - 0

11,15 ⟶ 1 - 1 1

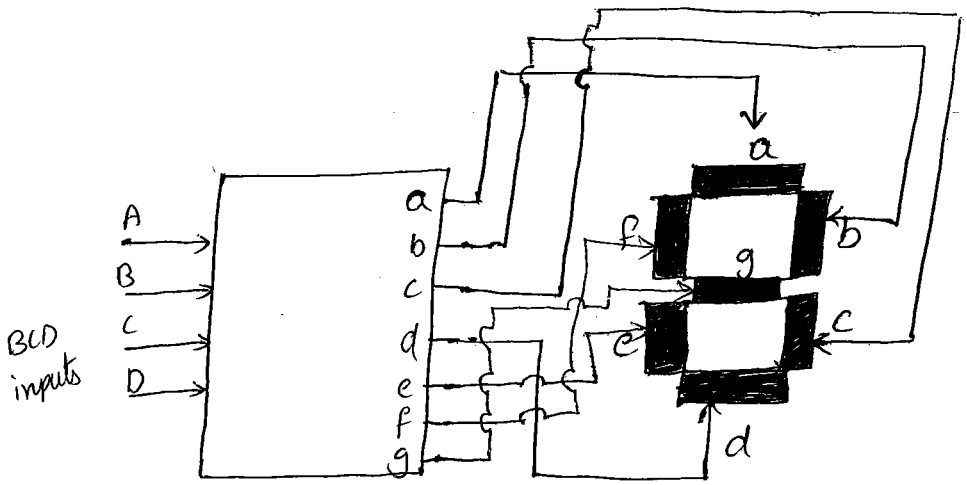
14,15 ⟶ 1 1 1 -

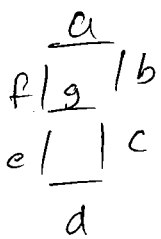
1,3,9,11 ⟶ - 0 - 1

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

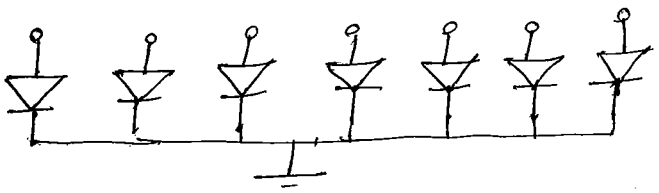
Seven segment decoder :- This type of decoders accepts the BCD code and provides outputs to energize seven segment display devices in order to produce a decimal read out.

→ Each segment is made up of a material that emits light when current is passed through it. the most commonly used materials include LED's, incandescent filaments and LSD's.





Block diagram



A common cathode LED display

Decimal digit	BCD input A B C D	seven segment outputs 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 1 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 1 0 1 1

$a = \Sigma m(0, 2, 3, 5, 6, 7, 8, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$

$b = \Sigma m(0, 1, 2, 3, 4, 7, 8, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$

$c = \Sigma m(0, 1, 3, 4, 5, 6, 7, 8, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$

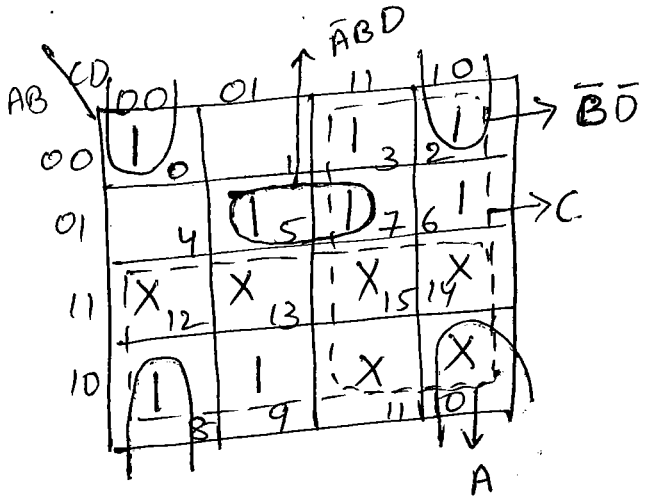
$d = \Sigma m(0, 2, 3, 5, 6, 8, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$

$e = \Sigma m(0, 2, 6, 8) + \Sigma d(10, 11, 12, 13, 14, 15)$

$f = \Sigma m(0, 4, 5, 6, 8, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$

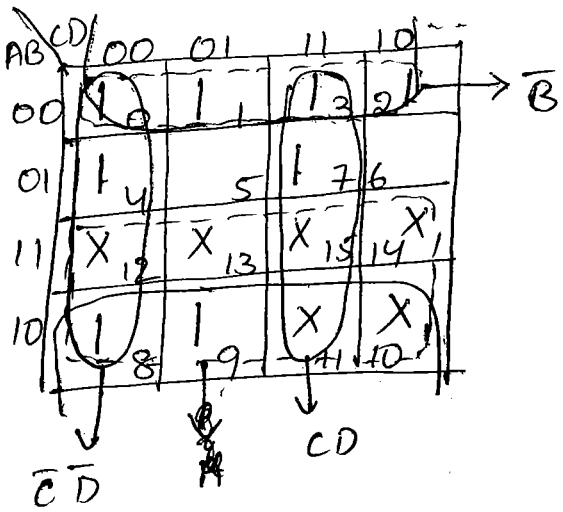
$g = \Sigma m(2, 3, 4, 5, 6, 8, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$

k-map for a



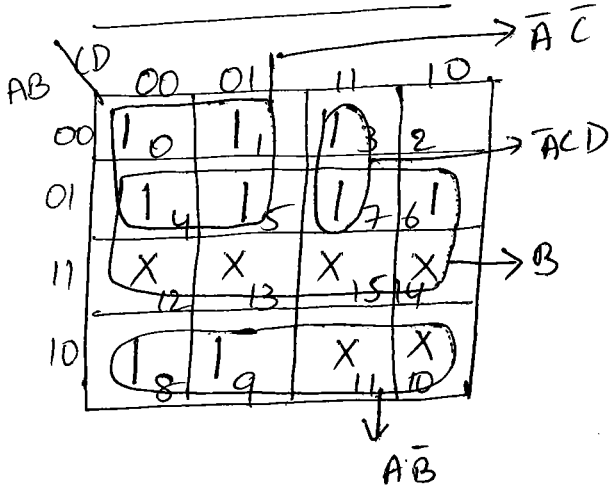
$a = A + C + \bar{B}\bar{D} + \bar{A}BD$

k-map for b



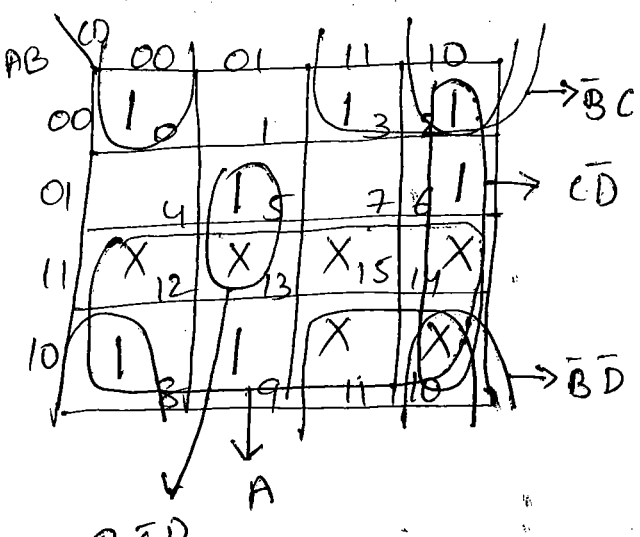
$b = \bar{B} + \bar{C}\bar{D} + CD$

k-map for c



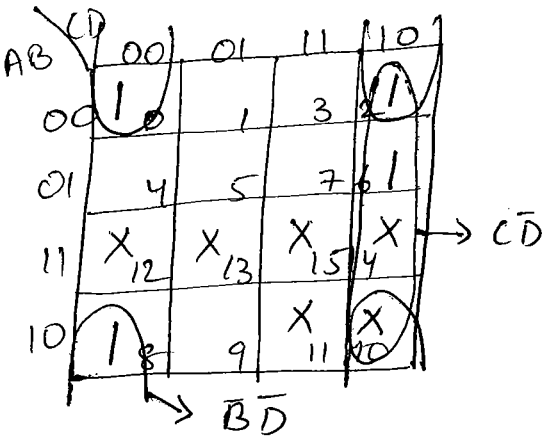
$c = A\bar{B} + B + \bar{A}\bar{C} + \bar{A}CD$

k-map for d



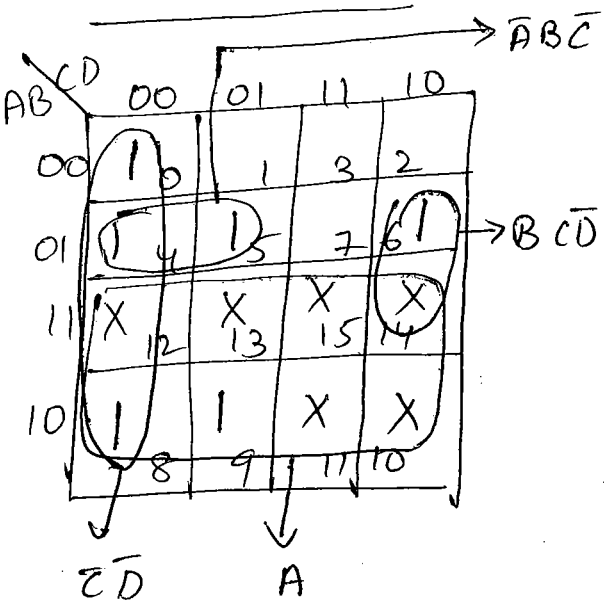
$d = A + \bar{B}C + \bar{B}D + B\bar{C}D + C\bar{D}$

K-map for e



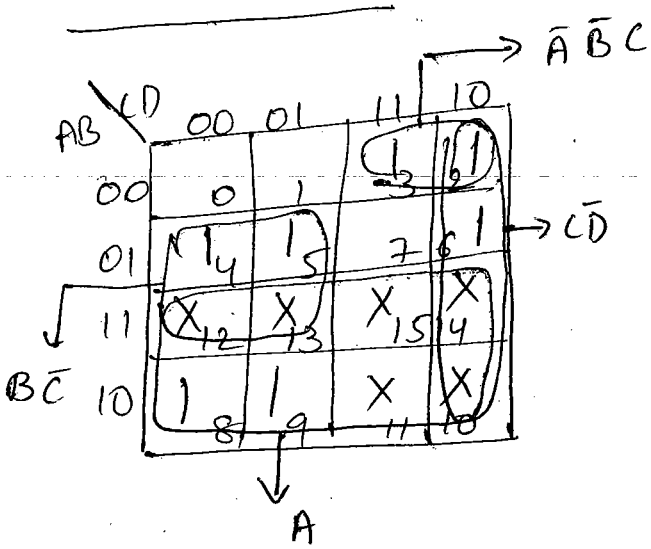
$e = \bar{B}\bar{D} + C\bar{D}$

K-map for f



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

K-map for g



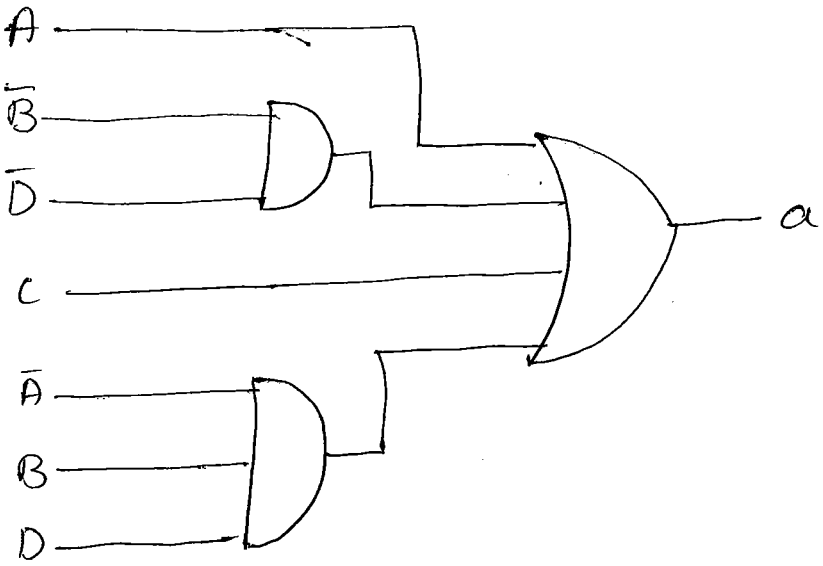
$g = A + B\bar{C} + \bar{A}\bar{B}C + C\bar{D}$

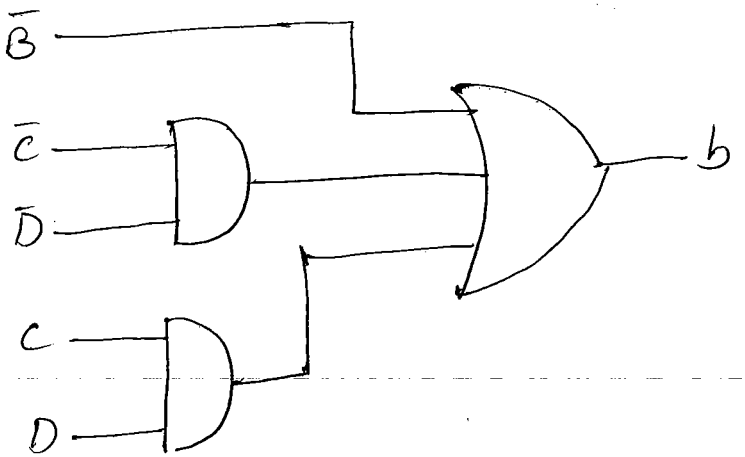
$\therefore \quad a = A + C + \bar{B}\bar{D} + \bar{A}BD \qquad e = \bar{B}\bar{D} + C\bar{D}$

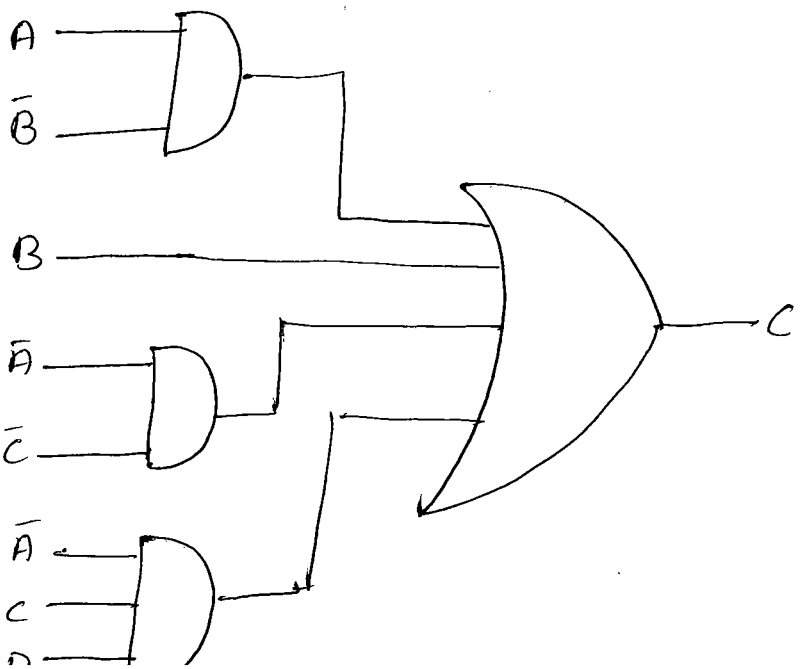
$b = \bar{B} + \bar{C}\bar{D} + CD \qquad f = A + \bar{C}\bar{D} + BC\bar{D} + \bar{A}B\bar{C}$

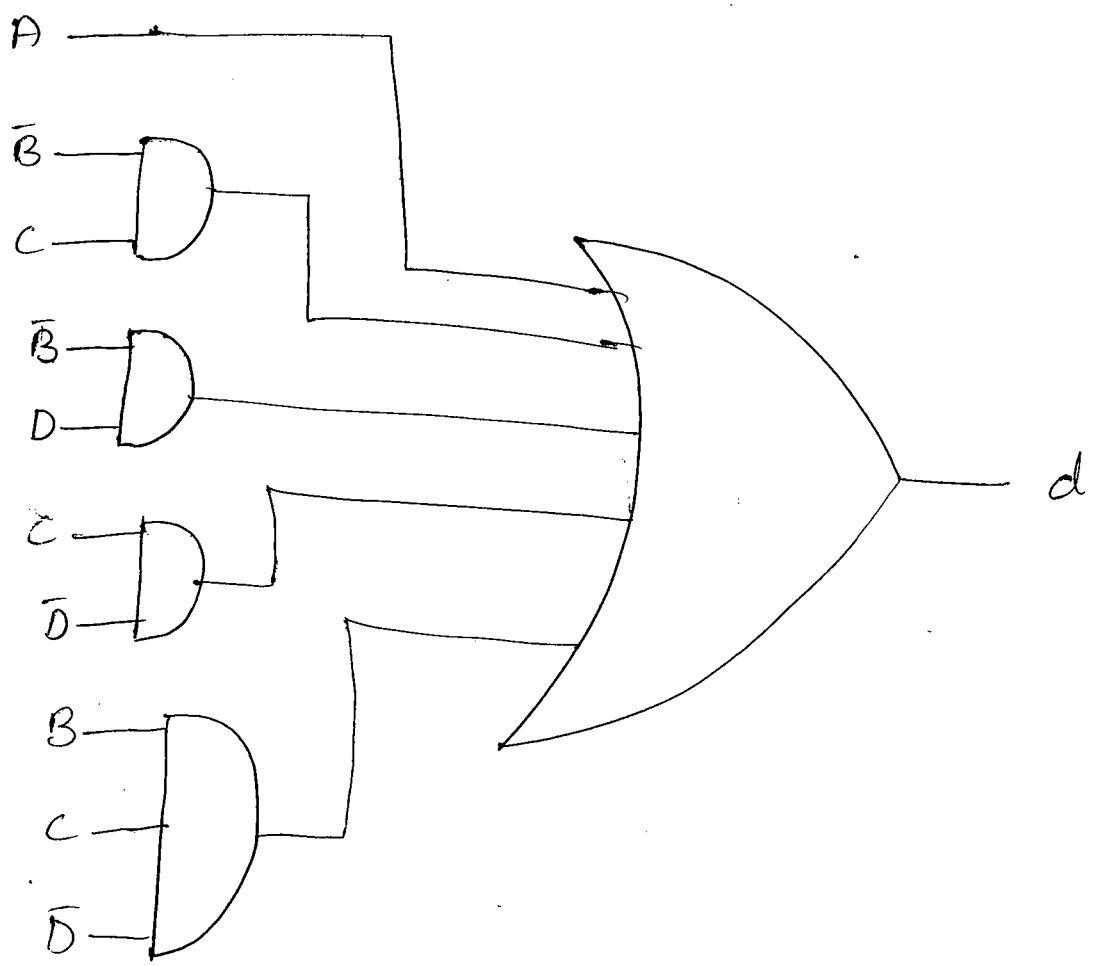
$c = A\bar{B} + B + \bar{A}\bar{C} + \bar{A}CD \qquad g = A + B\bar{C} + \bar{A}\bar{B}C + C\bar{D}$

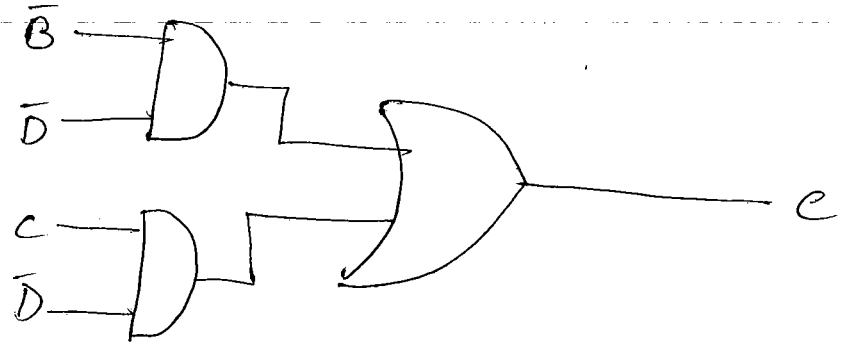
$d = A + \bar{B}C + \bar{B}D + C\bar{D} + BC\bar{D}$

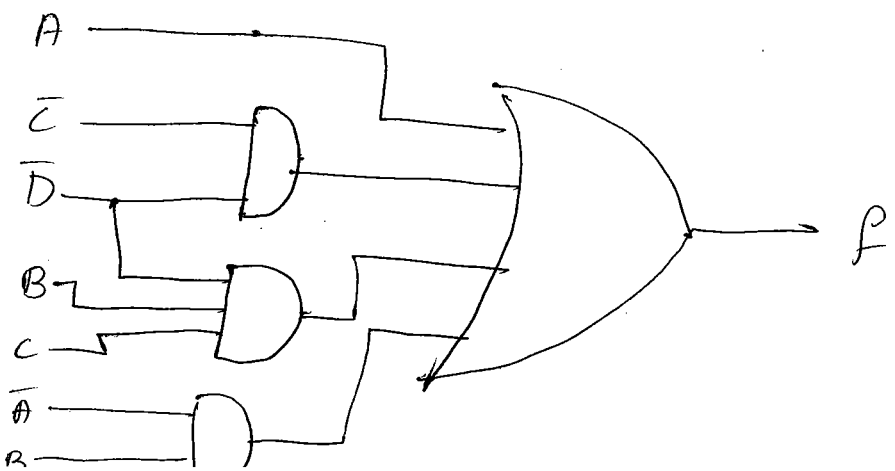

A
$\bar{B}$
$\bar{D}$
C
$\bar{A}$
B
D
a


$\bar{B}$
$\bar{C}$
$\bar{D}$
C
D
b


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


A
$\bar{B}$
C
$\bar{B}$
D
$\bar{C}$
$\bar{D}$
B
C
$\bar{D}$
d


$\bar{B}$
$\bar{D}$
C
$\bar{D}$
e


A
$\bar{C}$
$\bar{D}$
B
C
$\bar{A}$
B
f

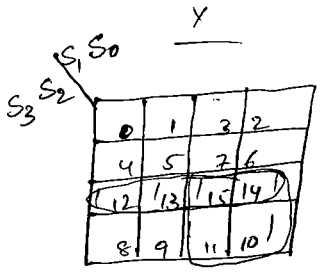
→ In BCD adder, 4-bit parallel adder (using IC 74LS83). two BCD groups A_3, A_2, A_1, A_0 and B_3, B_2, B_1, B_0 are applied to a 4-bit parallel adder

→ 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 outputs S_4, S_3, S_2, S_1, can range from 00000 to 10010

→ The circuitry for BCD adder must include the logic needed to detect whenever the sum is greater than 1001

inputs				output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
⋮	⋮	⋮	⋮	⋮
1	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



$Y = S_3 S_2 + S_3 S_1$

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 → 6 should be added to that sum, to produce the proper BCD result.

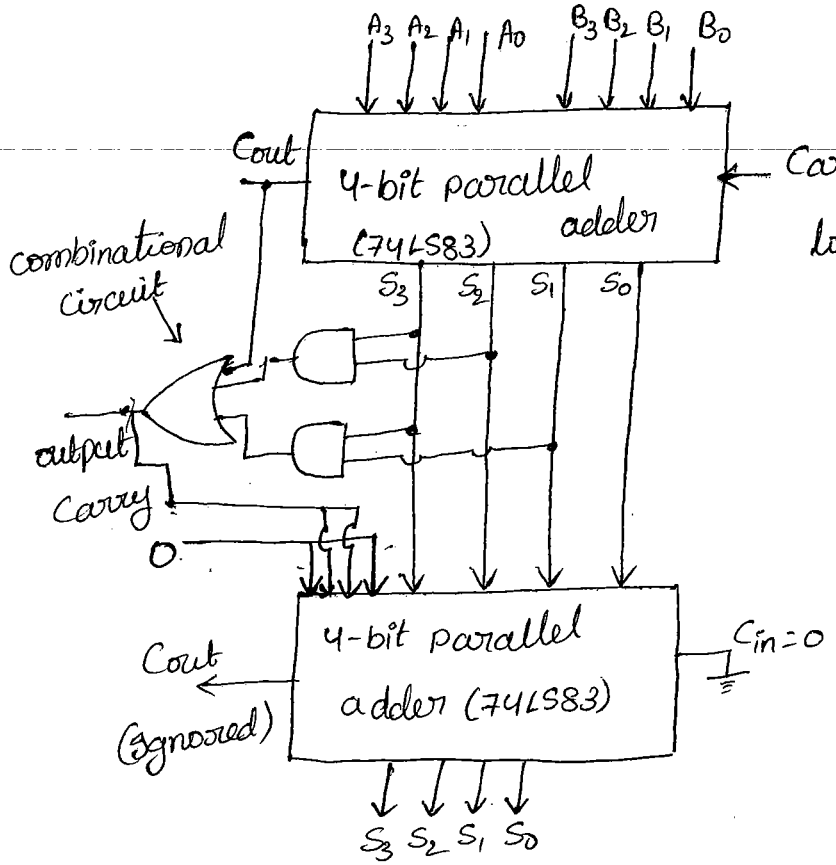


Figure:- Block diagram of BCD adder

(8)

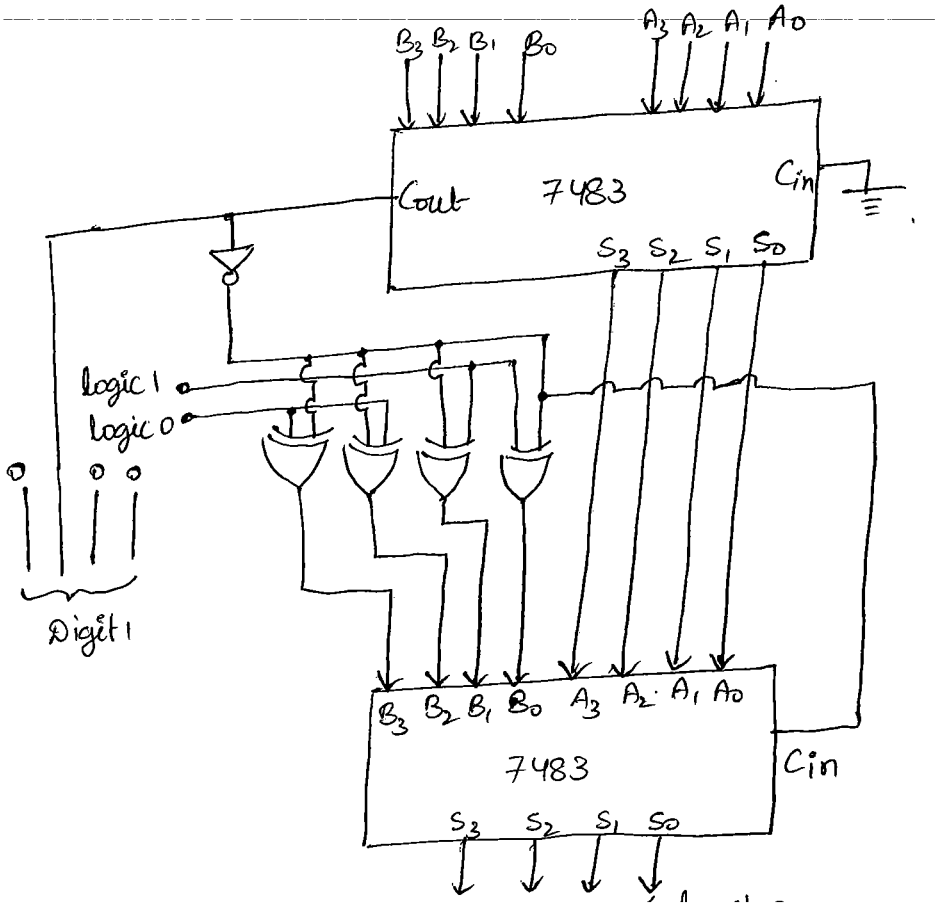
Excess-3 Adder

* To perform excess-3 addition of two numbers

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

carry = 0 → subtract $3(0011)_2$ from the sum.

* when carry = 1, B_3, B_2, B_1, B_0 of second 7483 should be 0011 & when carry = 0, $B_3 B_2 B_1 B_0$ of the second should be 1101 (2's complement of 3). in below figure.

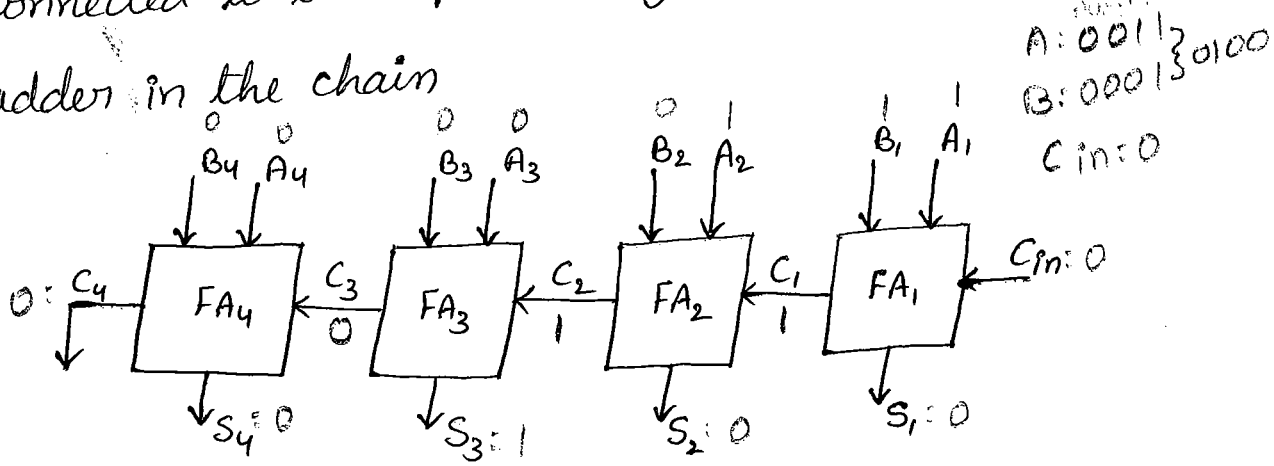


when $C_{out} = 0$, $B_3\ B_2\ B_1\ B_0$ get the 1's complement of 3 as an input and $C_{in} = 1$ with these inputs we get addition of resulted number with 2's complement of 3 i.e Final result = result - 3.

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.
* There are 2^n input lines and n select lines and one output.

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 a 4-bit binary parallel adder.

The interconnection of four 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 carries are connected in a chain through the full-adders. The input carry to the adder is C_{in} 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 terminals for the input and output Carries. An n-bit parallel adder requires n-full adders. it can be constructed from 4-bit, 2-bit, and 1-bit full adder IC's by cascading several packages. the output carry from one package must be connected to the input carry of the one with the next higher - order bits.

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. if two numbers are added such that no carries occur between stages.

4-Bit parallel subtractor: The subtraction of binary numbers can be carried out most conveniently by means of complements as 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 of B and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters

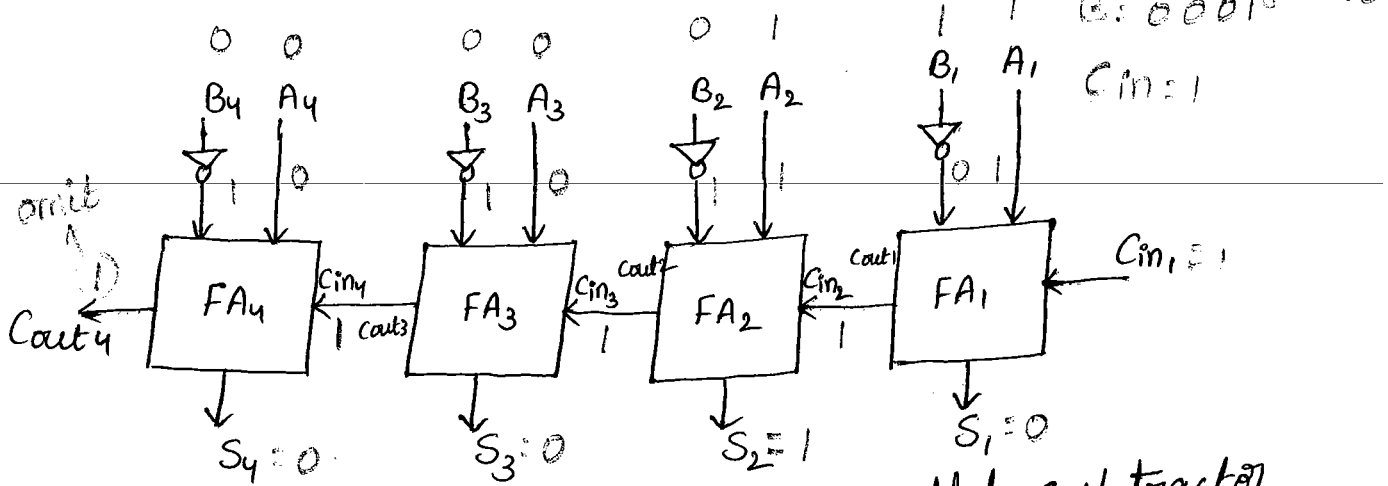


Fig: Logic diagram of a 4-bit parallel subtractor

FA_1: A_1: 1, B_1: 0, C_{in1}: 1 → 1 0 (Carry)

FA_2: A_2: 1, B_2: 1, C_{in_2}: 1 → 1 1 (Carry)

FA_3: A_3: 0, B_3: 1, C_{in_3}: 1 → 1 0 (Carry)

FA_4: A_4: 0, B_4: 1, C_{in_4}: 1 → 1 0 (Carry)

Binary Adder- Subtractor :- A 4-bit adder-subtractor circuit. Here 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 mode input M controls the operation.

→ when M=0 the circuit is an adder

→ when M=1 the circuit is an subtractor

→ when M=0, B⊕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⊕1 = $\bar{B}$, the full adder receives the value of $\bar{B}$, the input carry is '1' and the circuit performs A-B

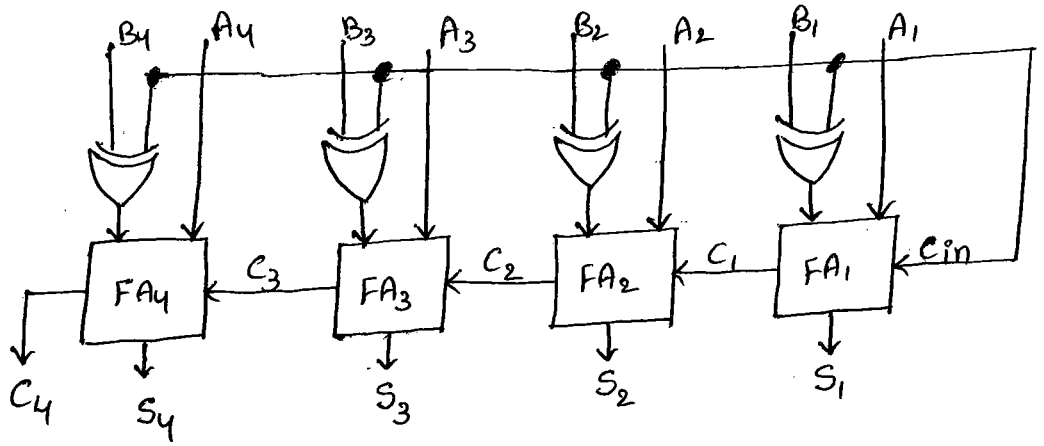
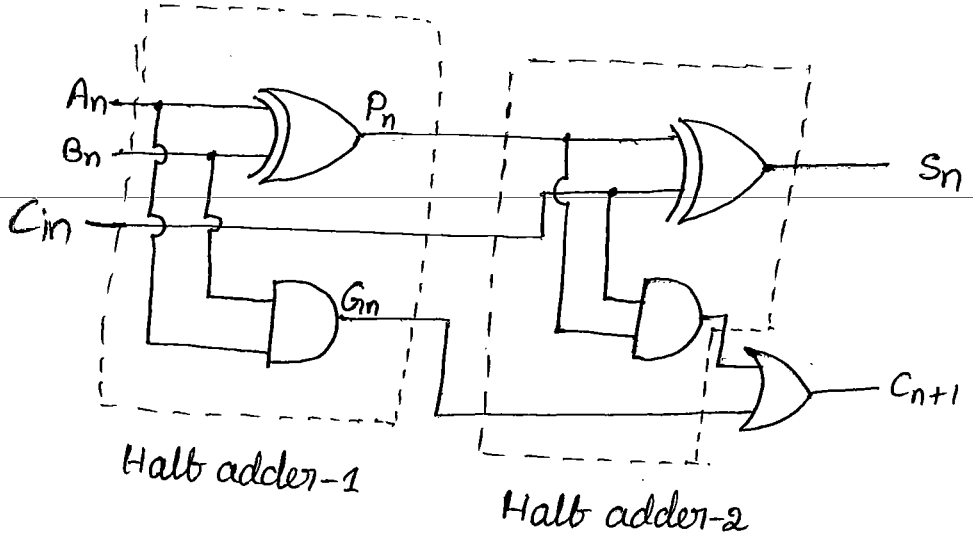


fig :- Logic diagram of a 4-bit binary adder-subtractor.

THE Look-Ahead-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 and also generates the carry-in bits for all the stages simultaneously.



Carry generate:- Consider one full adder stage. n^{th} 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 regadless of whether the input

Carry C_{in} is '0' or '1'. if G_i 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 or 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

$$S_n = P_n \oplus C_n \quad (\because P_n = A_n \oplus B_n)$$

$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

$$= A_n . B_n + P_n C_n$$

$$= A_n . B_n + (A_n \oplus B_n) C_n$$

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

$$n=1 \rightarrow C_1 = G_0 + P_0 . C_0 = G_0 + (A_0 \oplus B_0) C_0$$

$$= A_0 B_0 + (A_0 \oplus B_0) C_0$$

$$n=2 \rightarrow C_2 = G_1 + P_1 . C_1 = G_1 + P_1 . G_0 + P_1 . P_0 . C_0$$

$$n=3 \rightarrow C_3 = G_2 + P_2 . C_2 = G_2 + P_2 . G_1 + P_2 . P_1 . G_0 + P_2 . P_1 . P_0 C_0$$

$$n=4 \rightarrow C_4 = G_3 + P_3 . C_3$$

$$= G_3 + P_3 . G_2 + P_3 . P_2 + P_3 . P_2 . G_1 + P_3 . P_2 P_1 G_0 +$$

$$P_3 . P_2 . P_1 . P_0 . C_0$$

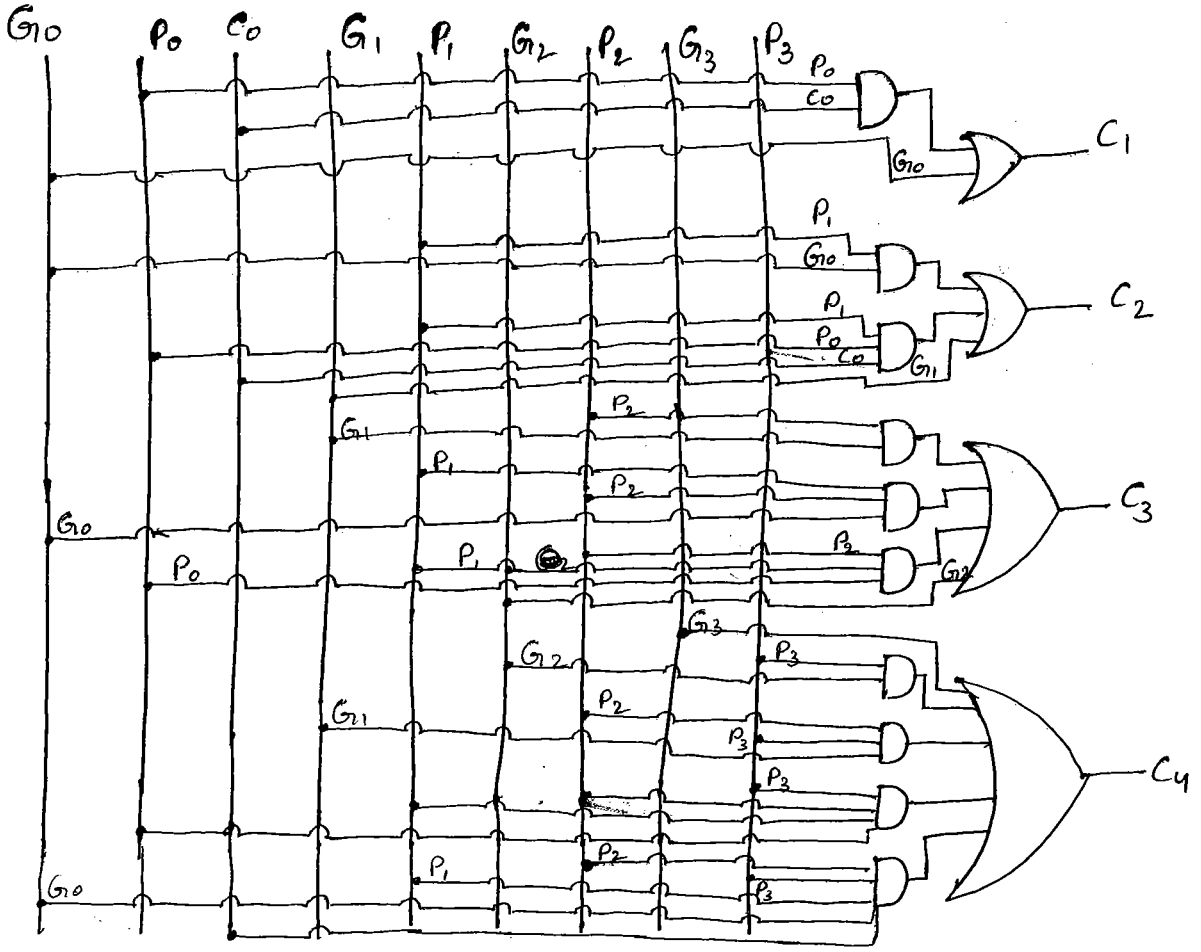
The general expression for n-stages

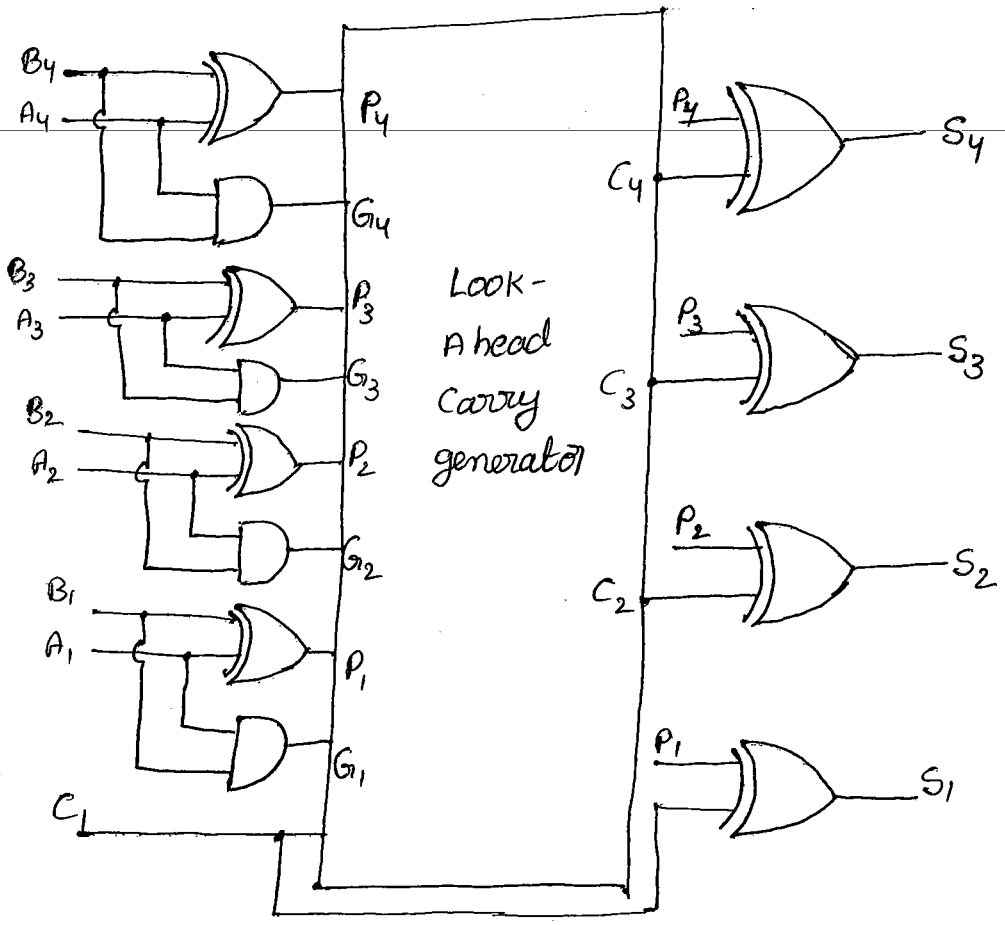
$$C_n = G_{n-1} + P_{n-1} . C_{n-1}$$

$$= G_{n-1} + P_{n-1} . G_{n-2} + P_{n-1} . P_{n-2} . G_{n-3} + \text{------}$$

$$\text{-----} + P_{n-1} . \text{---} P_0 . C_0$$

The final output carry is expressed as a function of the input variables in sop form, which is a two-level AND-OR or equivalent NAND-NAND form.





Logic diagram of a 4-bit look-ahead carry adder

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 = $A + 1 = 1$

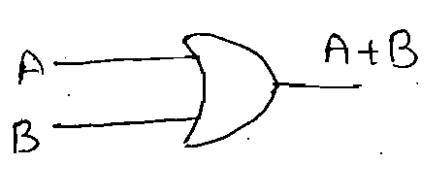
Law 2 = $A + 0 = A$

Law 3 : $A + A = A$

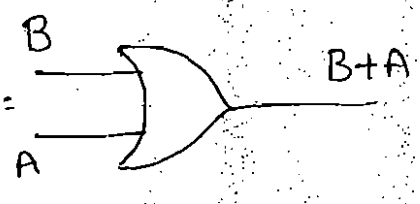
Law 4 : $A + \overline{A} = 1$

4. Commutative laws :-

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



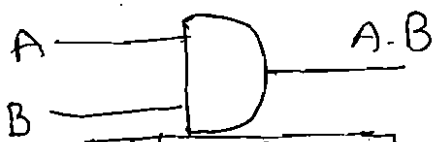
=



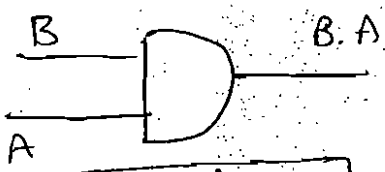
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

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

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



=

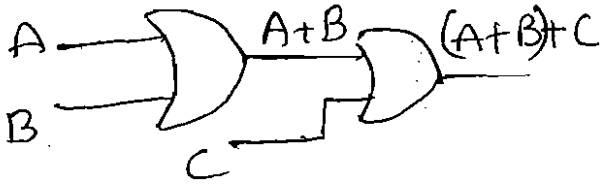


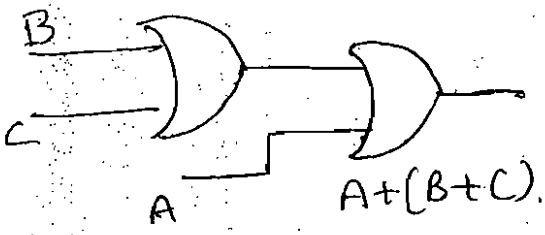
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

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

5. Associate laws :-

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



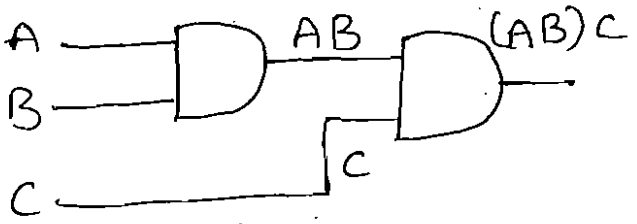


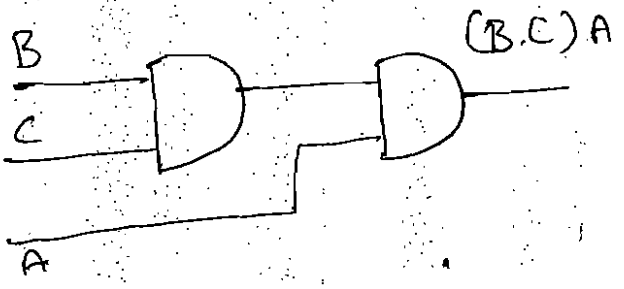
A B

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



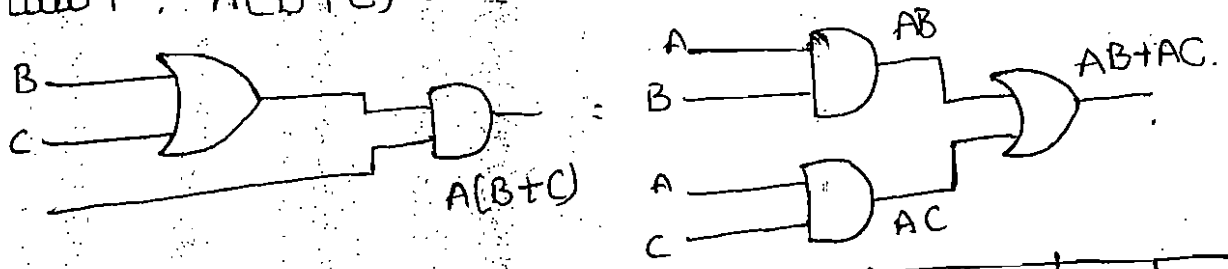


A	B	C	AB	(AB) C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

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 :-

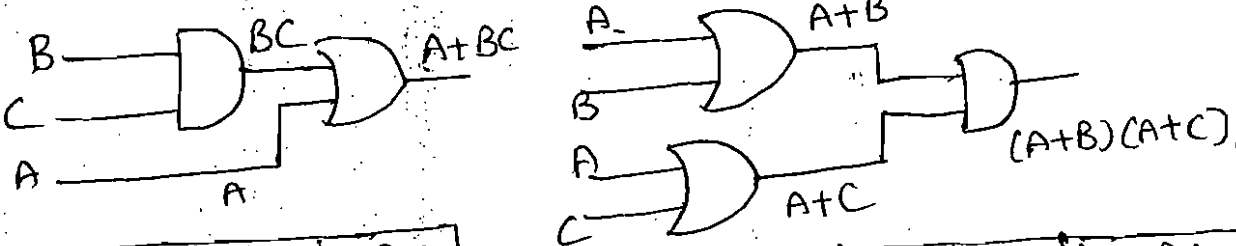
Law I : $A(B+C) = AB + AC$



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 II : $A + BC = (A+B)(A+C)$



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

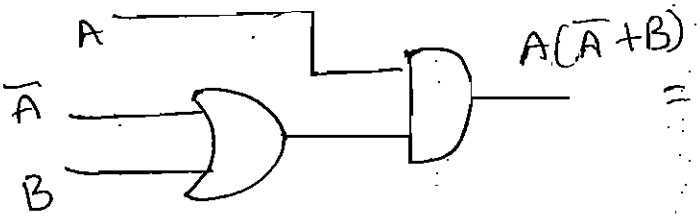
7. Redundant literal Rule :-

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

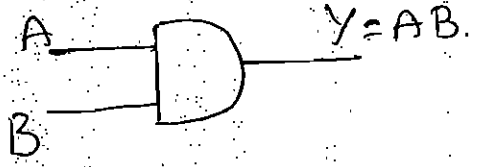
$= A(\bar{A}+B)$

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

$= AB$



=



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

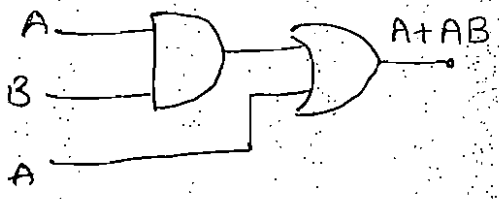
Law = $A+A = A$

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

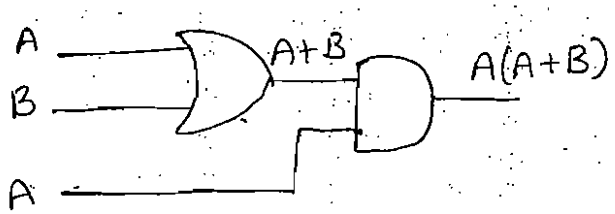
$= A+A\cdot B$

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

$= A\cdot 1$

$= A.$

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 or without the third term removed or retained.

Transposition Theorem :-

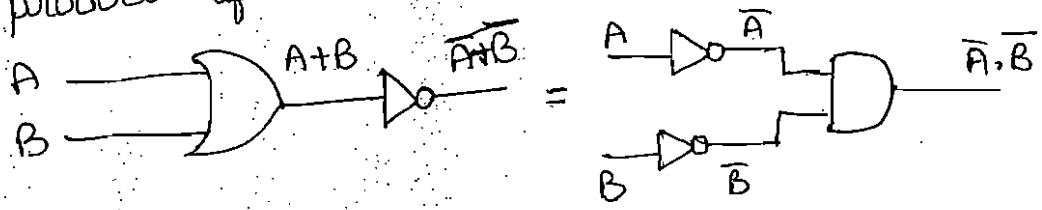
Theorem : $AB + \bar{A}C = (A+C)(\bar{A}+B)$

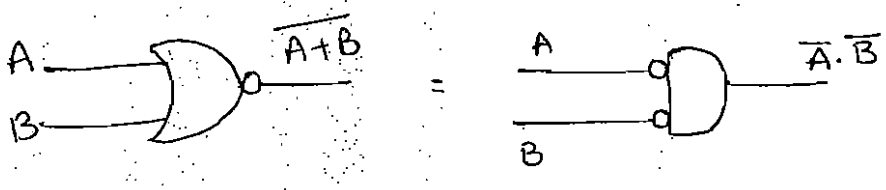
$$\begin{aligned} R.H.S &= (A+C)(\bar{A}+B) \\ &= A\cdot\bar{A} + AB + \bar{A}C + BC \\ &= AB + \bar{A}C + BC \\ &= AB + \bar{A}C + BC(A+\bar{A}) \\ &= AB + \bar{A}C + ABC + \bar{A}BC \\ &= AB(1+C) + \bar{A}C(1+B) \\ &= AB + \bar{A}C \end{aligned}$$

Demorgan's Theorem :-

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

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

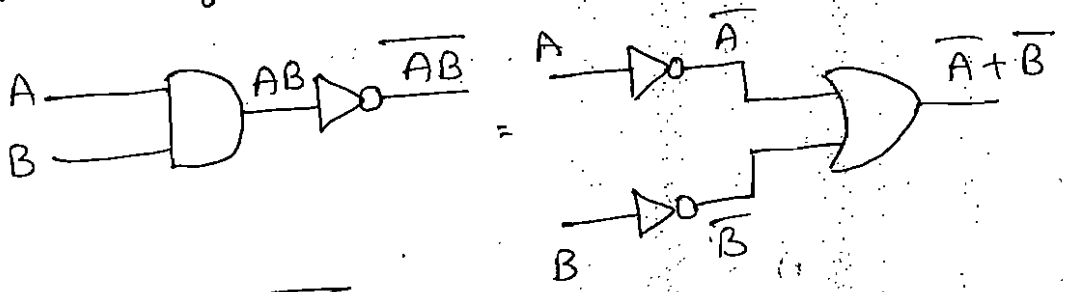
Similarly → $\overline{A+B+C+D+E} = \overline{A}\,\overline{B}\cdot\overline{C}\cdot\overline{D}\,\overline{E}$

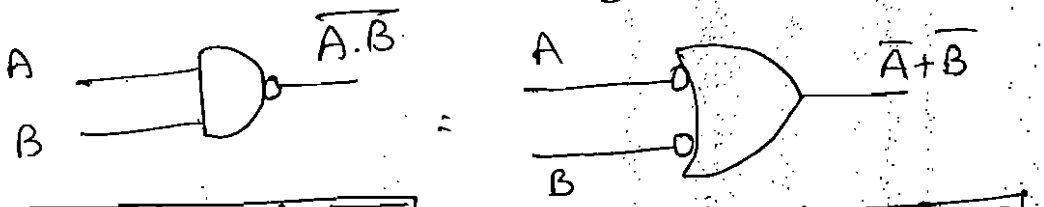
→ $\overline{\overline{A}B+C\overline{D}+\overline{E}\overline{D}} = (\overline{\overline{A}B})(\overline{C\overline{D}})(\overline{\overline{E}\overline{D}})$

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

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 :-

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

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

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

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

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

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

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

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

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

$= AB$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$A \quad B$

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

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

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

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

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

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

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

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

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

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

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

$= A [B \odot C]$

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

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

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

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

$= B + BD + BC + BCD$

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

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

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

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.

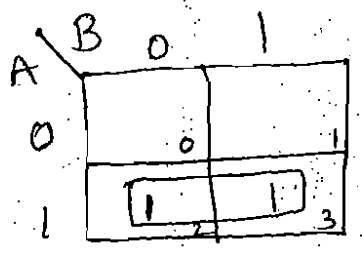
→ Each of the product terms in the standard sop form is called minterms and each of the sum terms in the standard pos form is called maxterms.

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

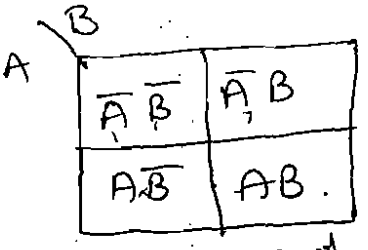
A two-variable K-map has $2^2 = 4$ cells or squares. 4 possible combinations of the input variables A and B. ($\bar{A}\bar{B}$, $\bar{A}B$, $A\bar{B}$, AB).

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

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



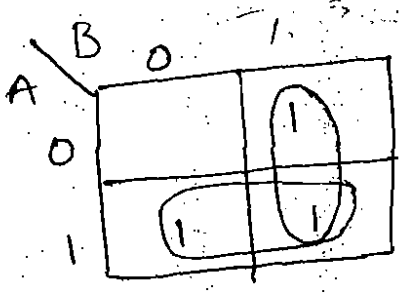
$f = B$.



The minterms of a two-variable K-map.

* Reduce the expression by using K-map.

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

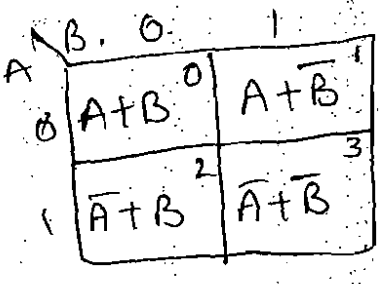


$f = A + B$

logic diagram

mapping of POS Expressions :-

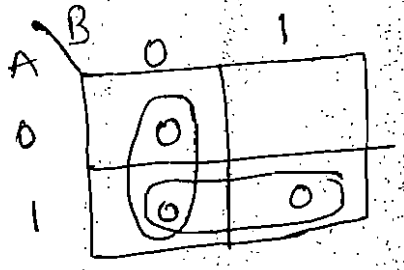
$M_0 = A + B$, $M_1 = A + \bar{B}$, $M_2 = \bar{A} + B$, $M_3 = \bar{A} + \bar{B}$

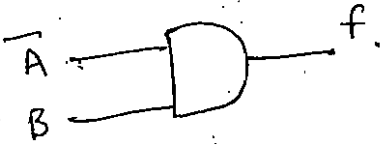


The maxterms of a two-variable K-map.

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

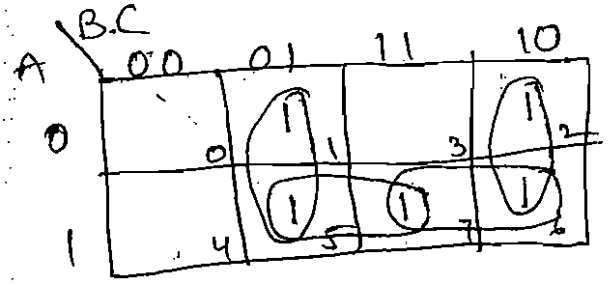
$\bar{A}\bar{B}C = 001 = m_1$,

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

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

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

$ABC = 111 = m_7$



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

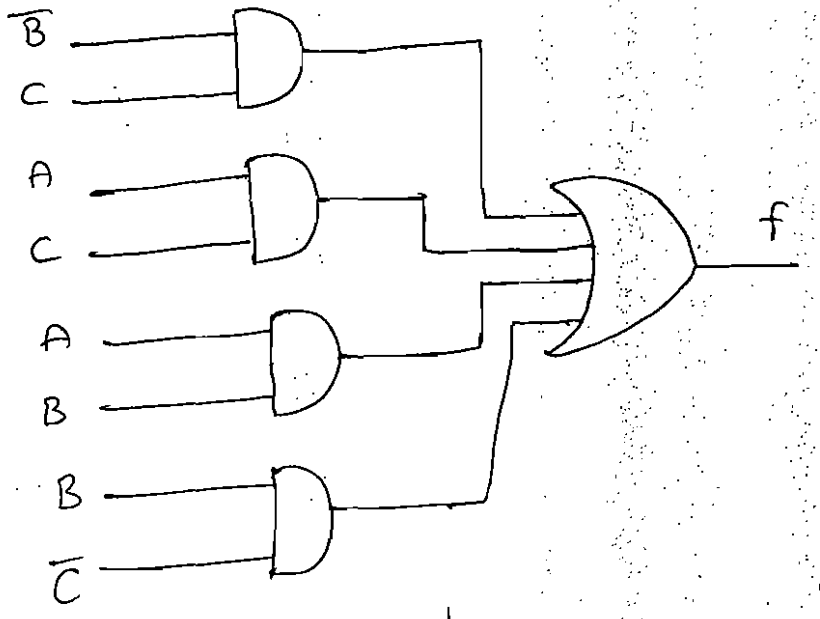
$f_2 = m_5 + m_7 = A\bar{B}C + ABC = AC(B+\bar{B}) = AC$

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

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

$f = f_1 + f_2 + f_3 + f_4$

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



logic diagram

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

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

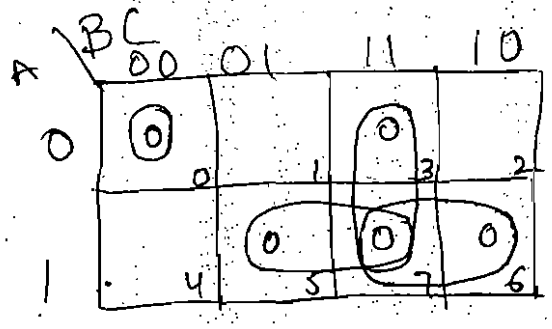
$A+B+C = 000 = M_0$

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

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

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

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



$f_1 = M_0 = A+B+C$

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

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

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

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

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

$f_3 = M_3 \cdot M_7$

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

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

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

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

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

$f_4 = M_7 \cdot M_6$

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

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

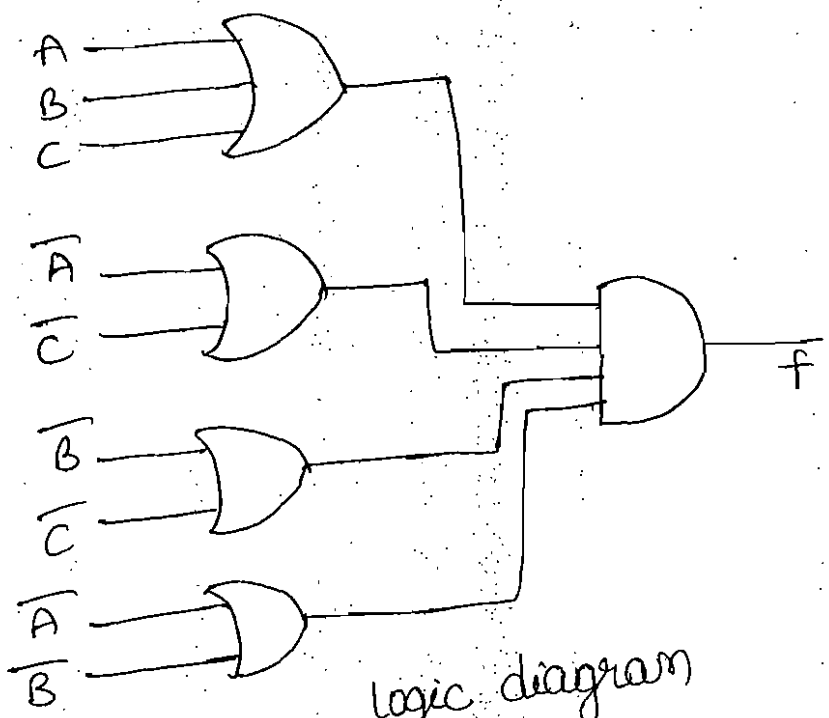
$= \bar{A} + \bar{A}\bar{B} + \bar{A}C + \bar{B} + \bar{B}C + \bar{A}\bar{C} + \bar{B}\bar{C} + 0$

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

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

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

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



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

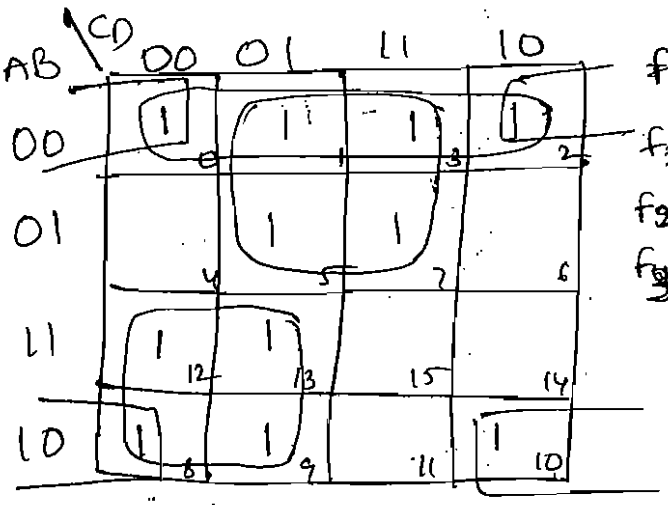
Sop form.

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.

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



$f_1 = m_8 + m_9 + m_2 + m_3 = \overline{A}B + \overline{A}\overline{B}$

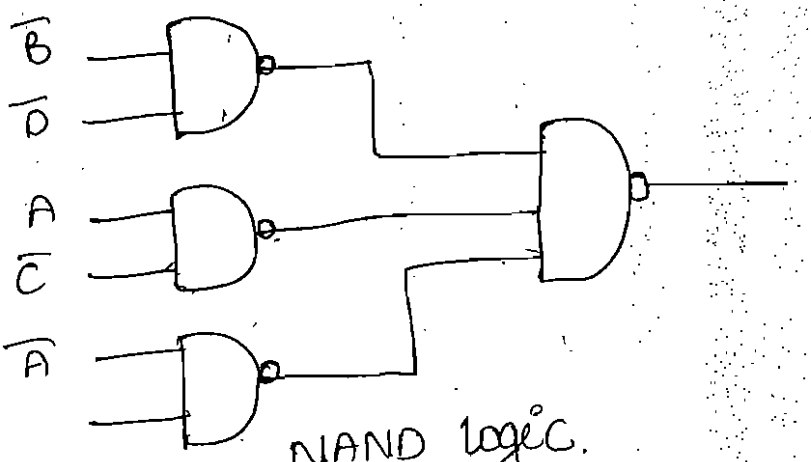
$f_2 = m_1 + m_3 + m_5 + m_7 = \overline{A}D$

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

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

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

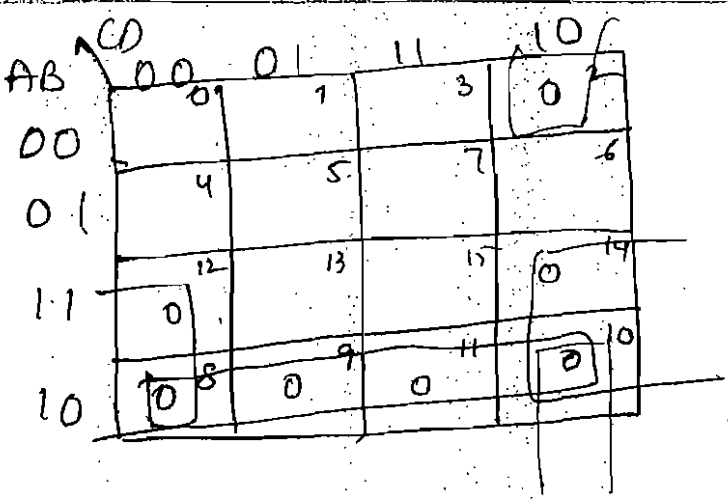
$= \overline{B}\overline{D} + A\overline{C} + \overline{A}D$



NAND logic.

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

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



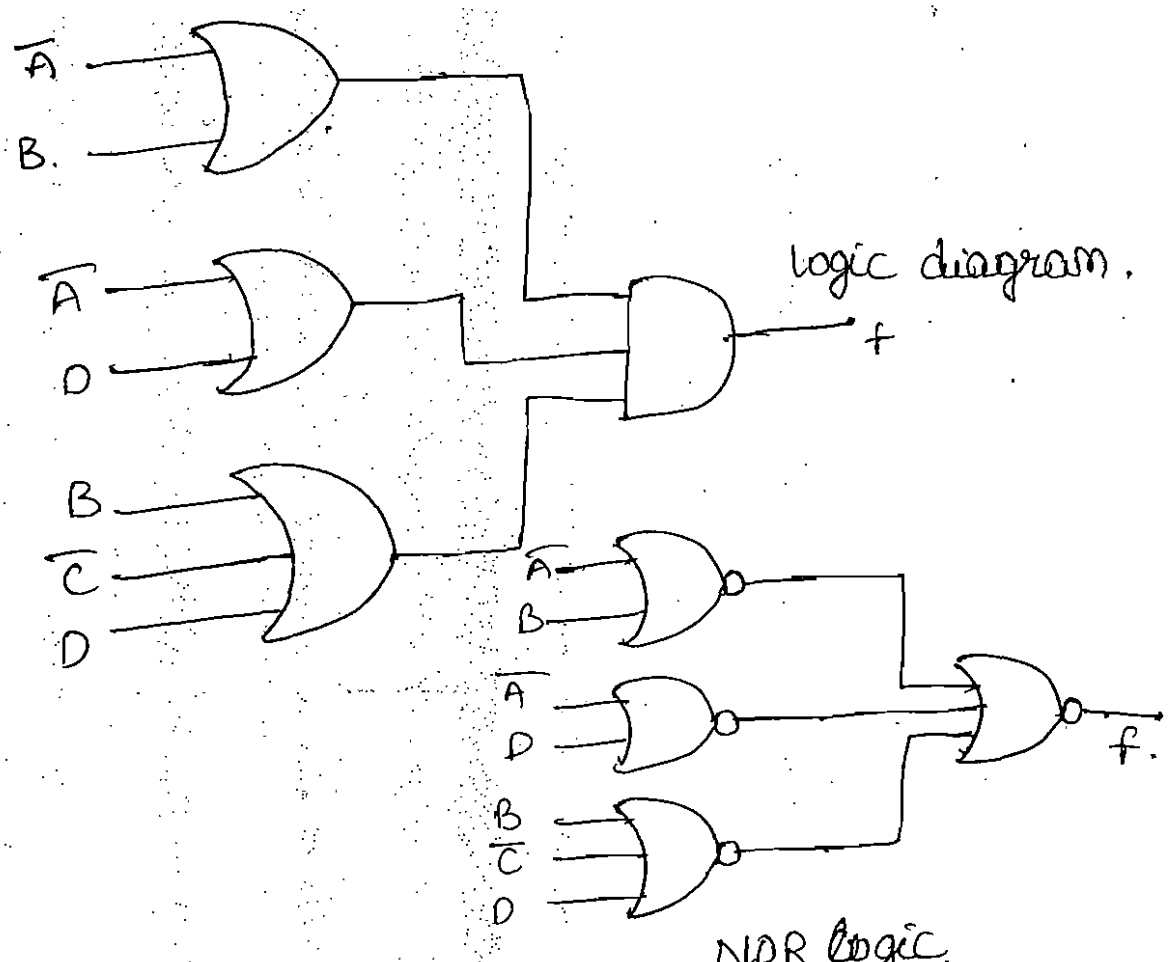
$f_1 = M_8 \cdot M_9 \cdot M_{11} \cdot M_{10}$ $f_2 = M_8 \cdot M_{12} \cdot M_{10} \cdot M_{14}$

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

$f_3 = M_2 \cdot M_{10}$

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

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



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:-

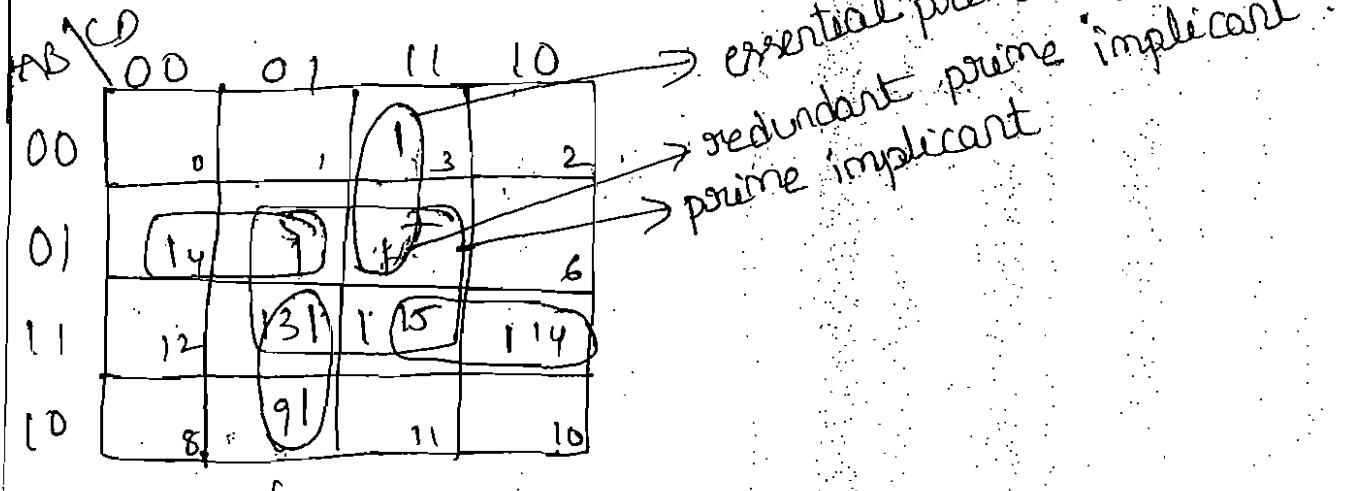
The prime implicants which contains at least one 1 which cannot be covered by any other prime implicant is called an essential prime implicant [EPI]

Redundant prime implicants:-

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

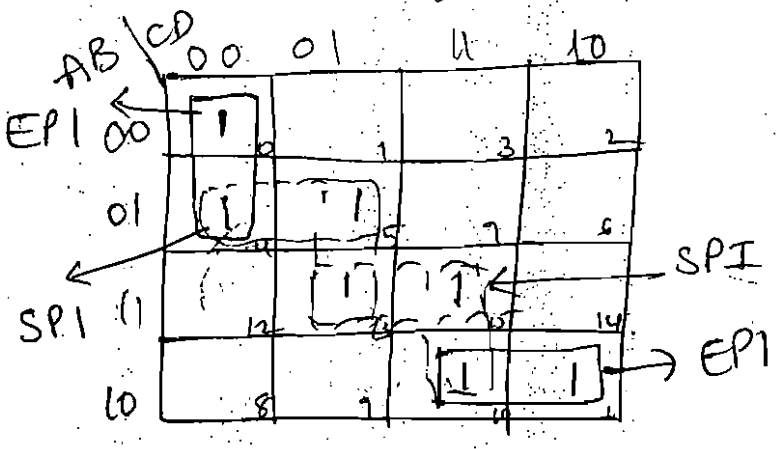
Selective prime implicants :-

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



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

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

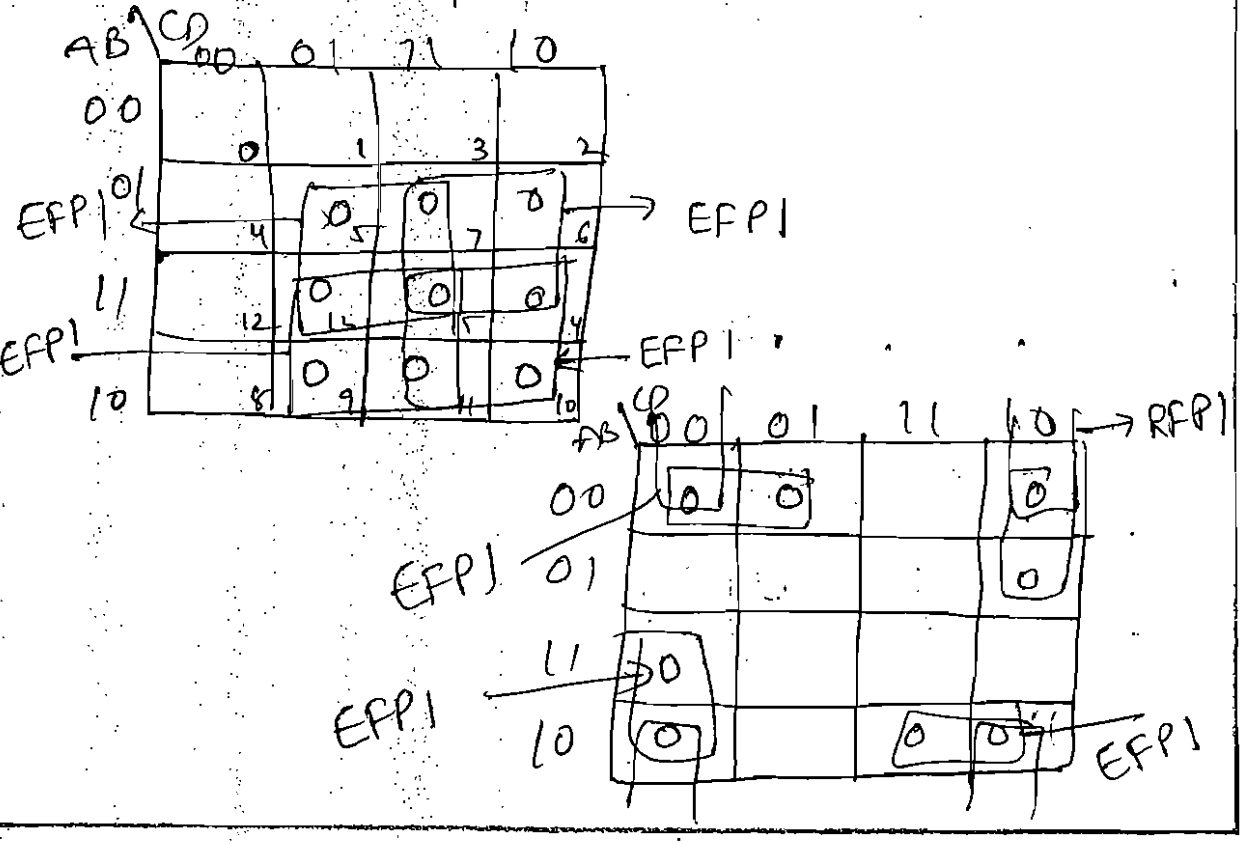


False prime implicants :- (FPI)

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

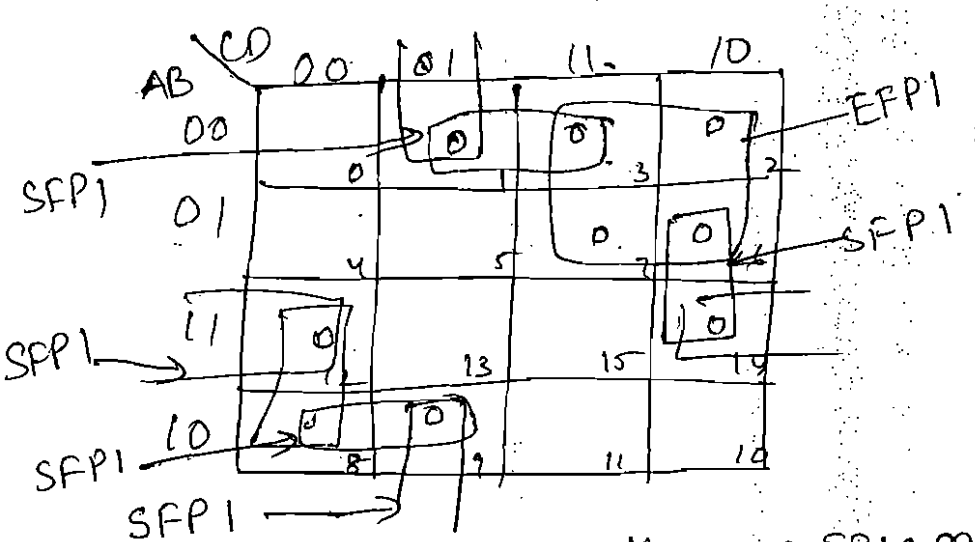
Essential false prime implicants :-

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.

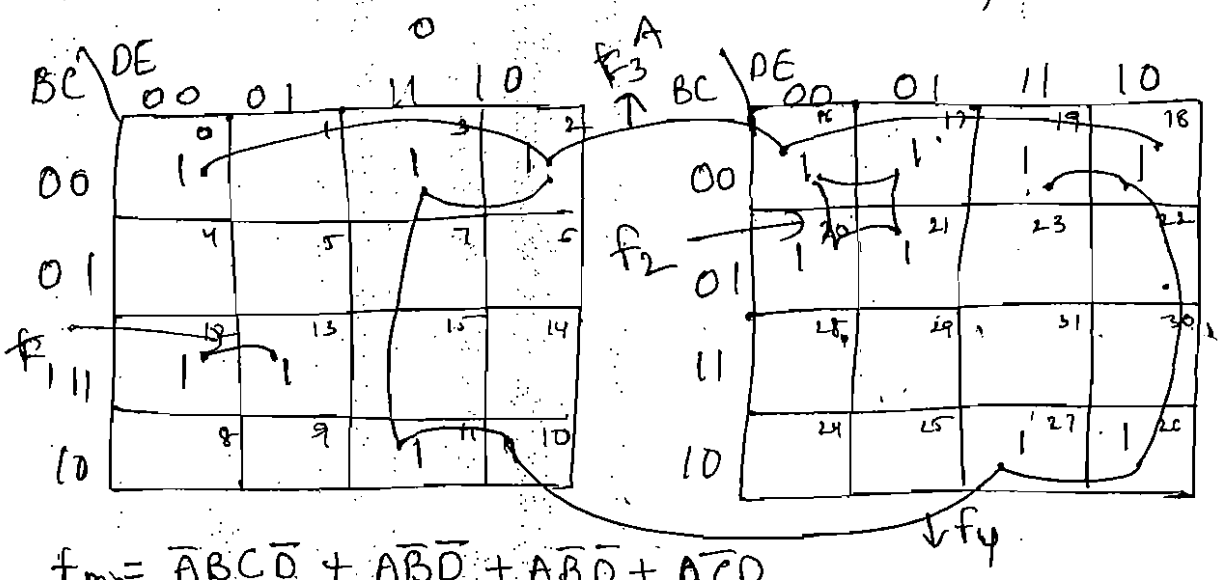
Five-variable K-map :-

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.

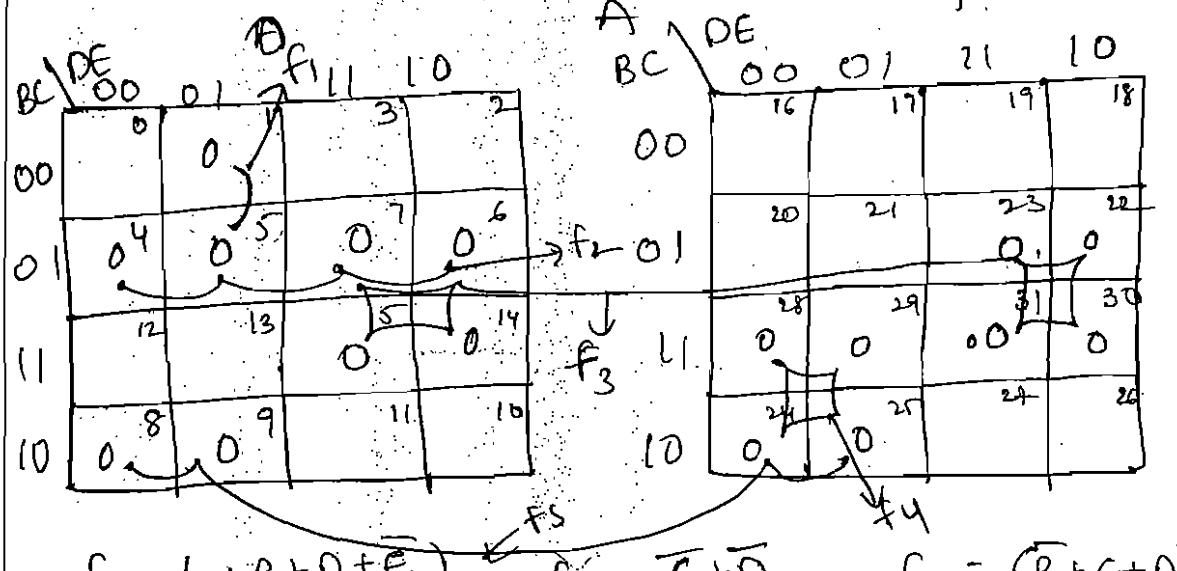
$f = \Sigma m(0,2,3,10,11,12,13,16,17,18,19,20,21,26,27)$



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

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

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



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

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

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

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

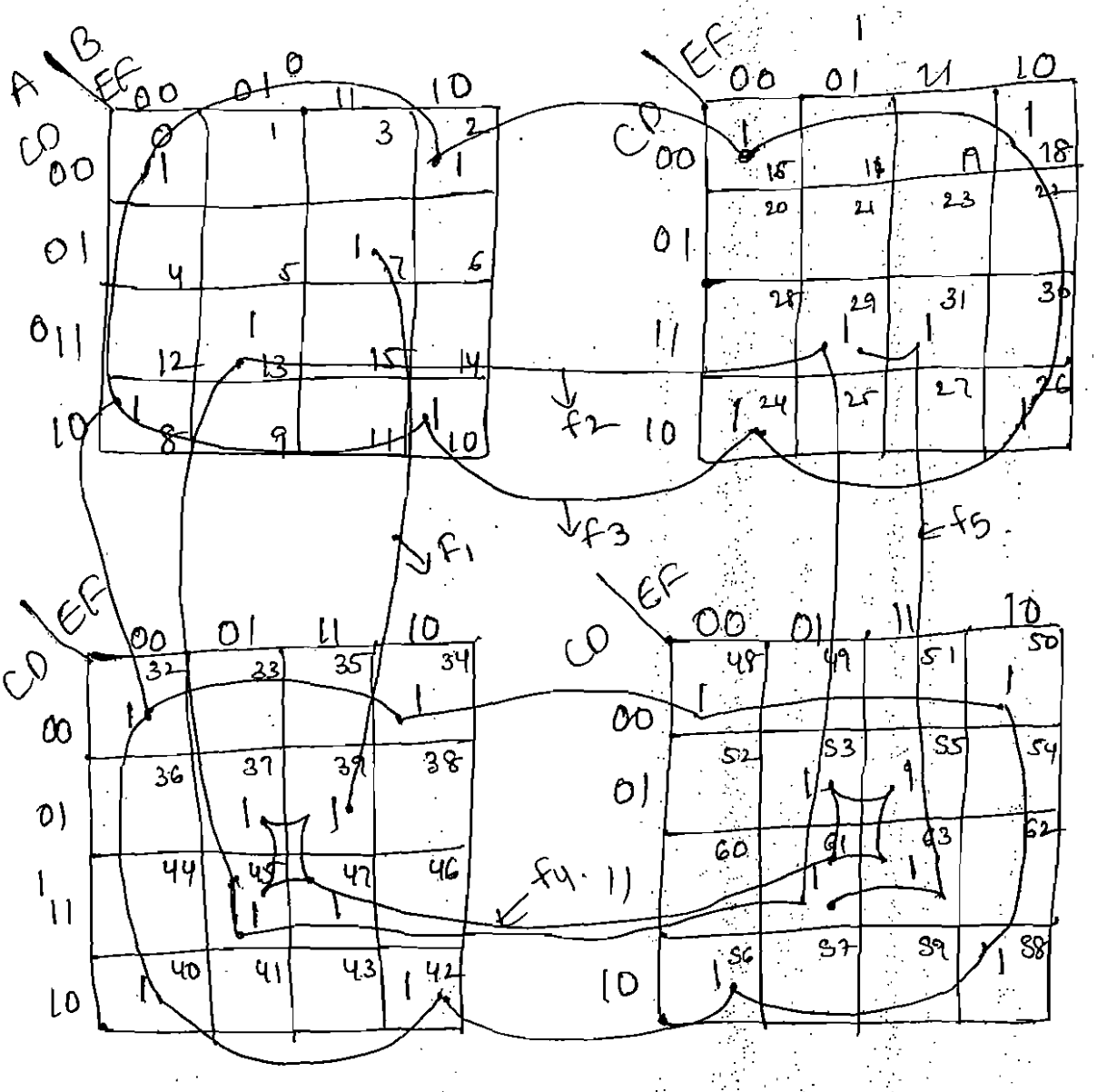
Six-Variable K-map :-

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

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



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

$f_2 = CD\bar{E}F$

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

$f_4 = ADF$

$f_5 = BCDF$

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

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

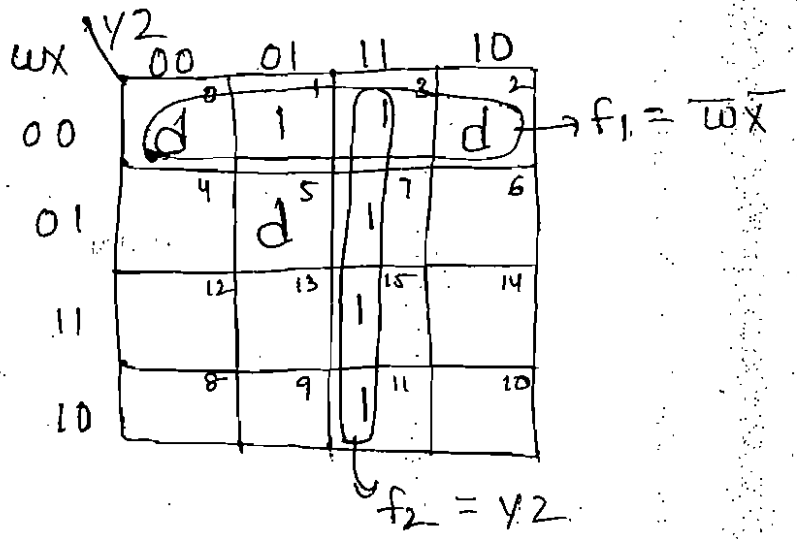
Don't care combinations :-

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.

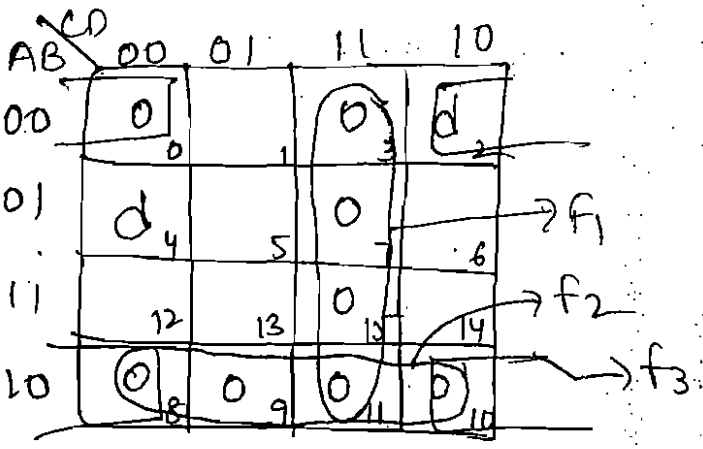
$F(w,x,y,z) = \Sigma m(1,3,7,11,15) + \Sigma d(0,2,5)$.



$F_{min} = f_1 + f_2$

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

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



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

$f_2 = \bar{A} + B$

$f_3 = (B+D)$.

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

Limitations of K-map :-

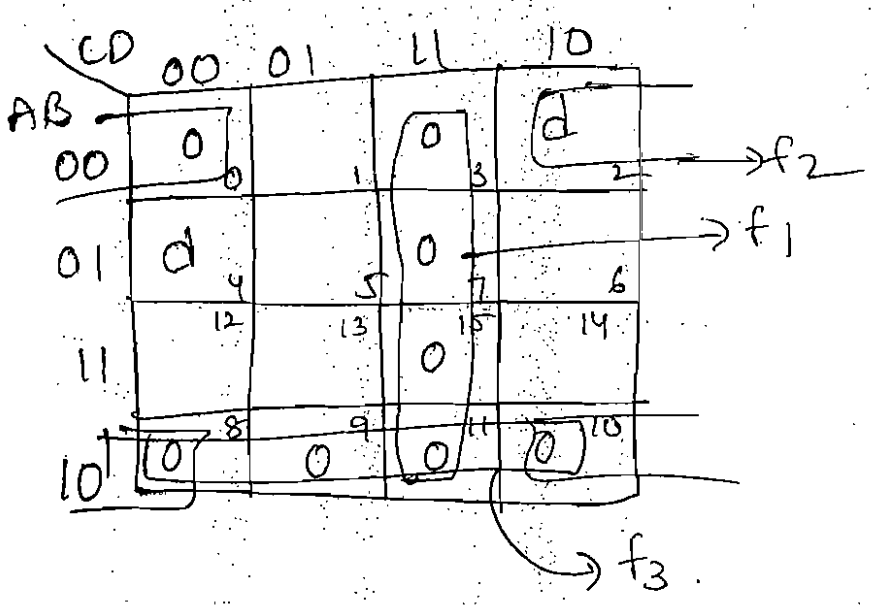
→ K-maps are not suitable when the number of variables involved exceed four. It may be used with difficulty up to five and six variable systems. But beyond six variable K-maps cannot be physically visualized.

→ 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.

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

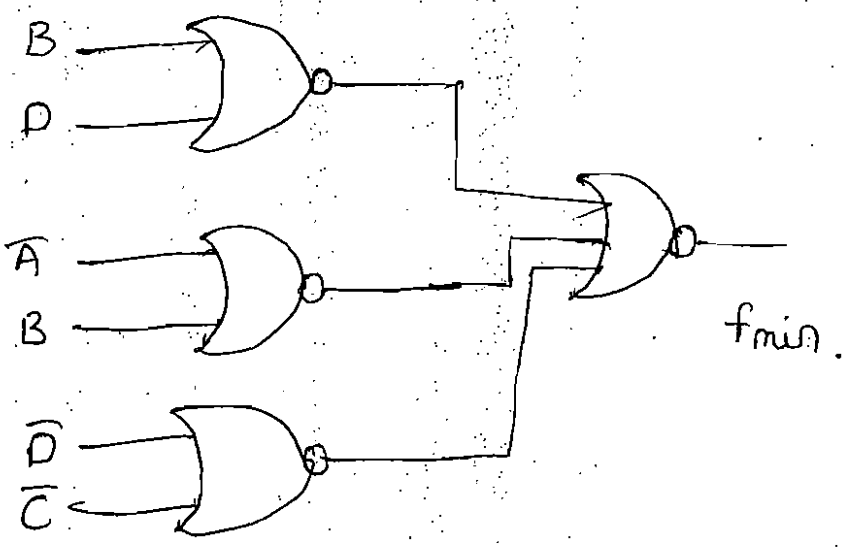


$f_1 = \bar{C} + \bar{D}$

$f_2 = (B + D)$

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

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



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.

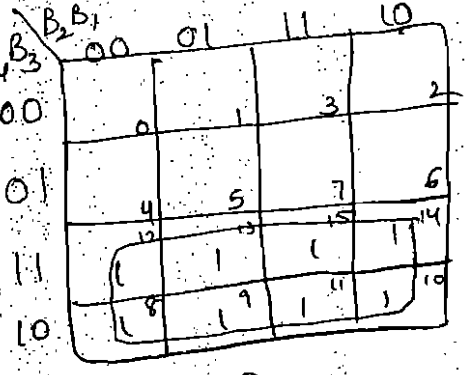
$$f_{min} = w + S + T + X + P$$
$$= \bar{A}\bar{B}\bar{D}\bar{E} + BC\bar{D}\bar{E} + A\bar{C}\bar{D}\bar{E} + A\bar{B}\bar{C}DE + ABCE$$

Code converters :-

Design a circuit 4-bit binary to Gray code converters

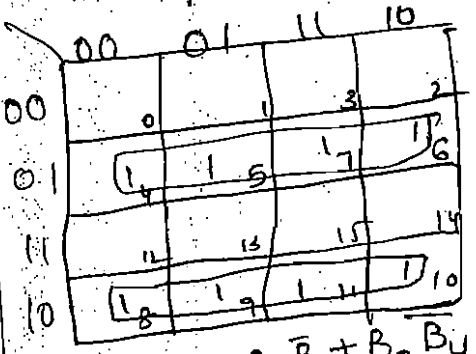
4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1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

K-map for G_4



$G_4 = B_4$

K-map for G_3



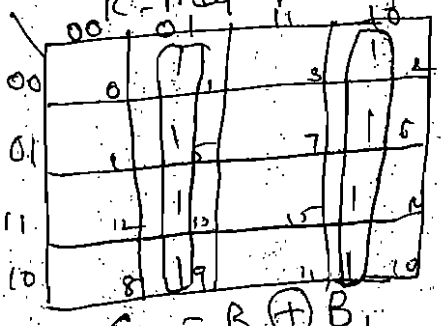
$G_3 = B_4\bar{B}_3 + B_3\bar{B}_4$

$= B_3 \oplus B_4$

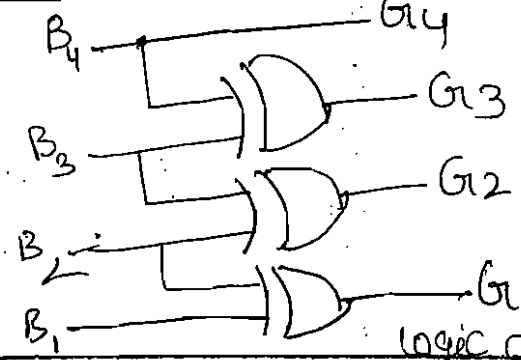
K-map for G_2

$G_2 = B_3 \oplus B_2$

K-map for G_1



$G_1 = B_2 \oplus B_1$

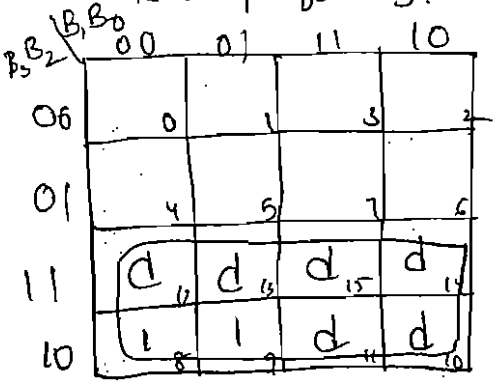


logic diagram

Design of a BCD to Gray code converter :-

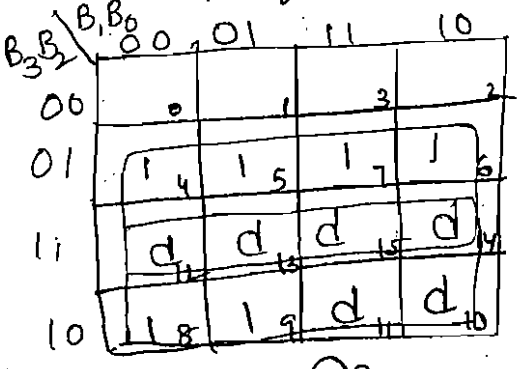
BCD code $B_3\ B_2\ B_1\ B_0$	Gray code $G_3\ G_2\ G_1\ G_0$
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1

K-map for G_3.



$G_3 = B_3$.

K-map for G_2



$G_2 = B_2 \oplus B_3$.

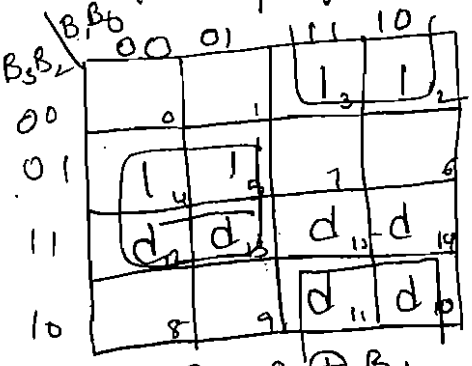
$G_3 = \Sigma m(8,9) + d(10,11,12,13,14,15)$.

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

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

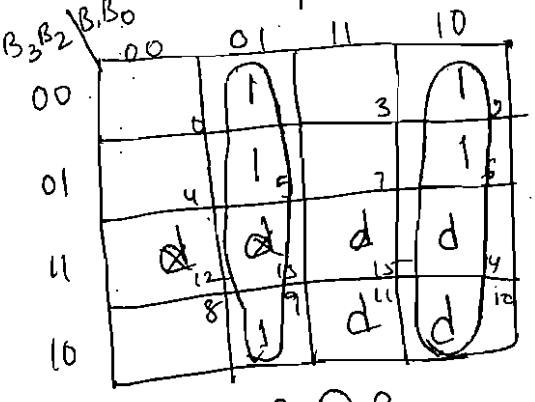
$G_0 = \Sigma m(1,2,5,6,9) + d(10,11,12,13,14,15)$.

K-map for G_1



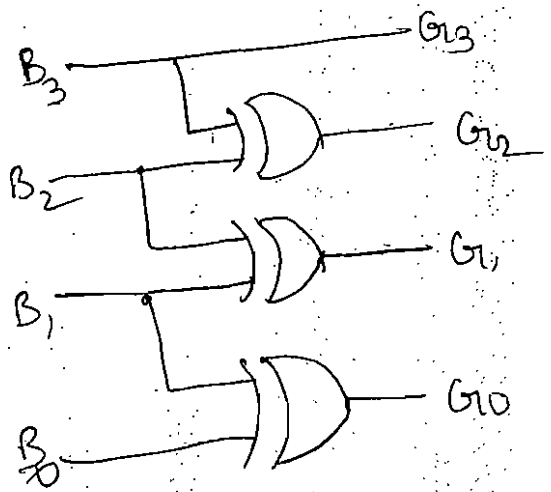
$G_1 = B_2 \oplus B_1$.

K-map for G_0



$G_0 = B_1 \oplus B_0$.

Logic diagram :-

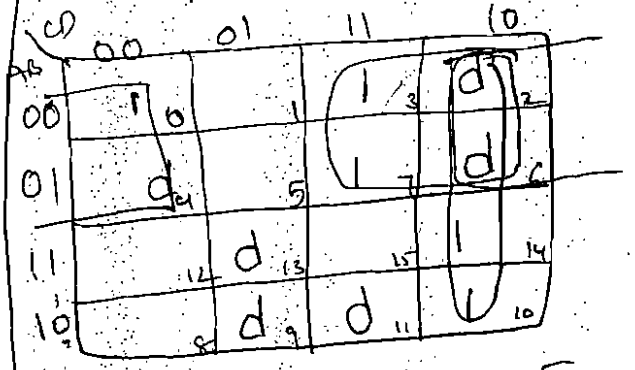


→ Design an SOP circuit to detect the decimal numbers 0, 2, 4, 6 and 8 in a 4-bit 5211 BCD code input.

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

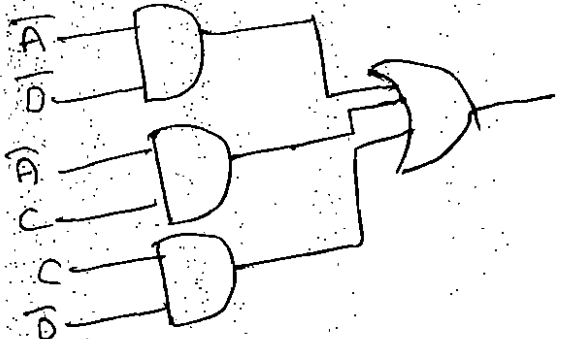
$f = \Sigma m(0,3,7,10,14) + d(2,4,6,9,11,13)$

K-map



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

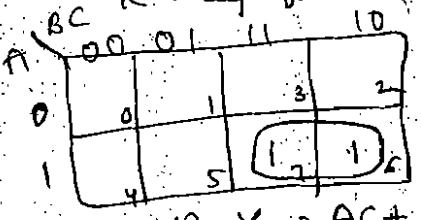
Logic diagram



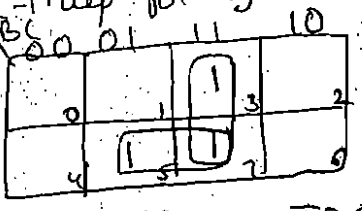
→ Design a combinational circuit that accepts a 3-bit BCD number and generates an output binary number equal to the square of the input number.

Inputs A	B	C	outputs 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	0	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

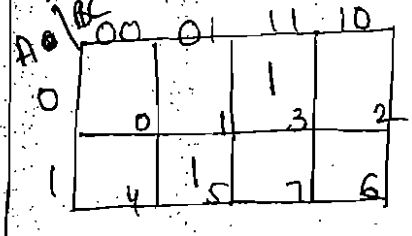
K-map for $X_6 = AB$



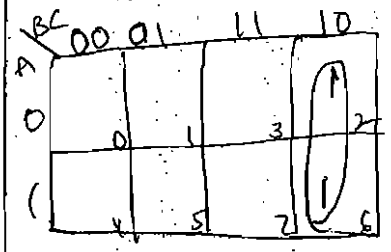
K-map for $X_5 = AC + BC$



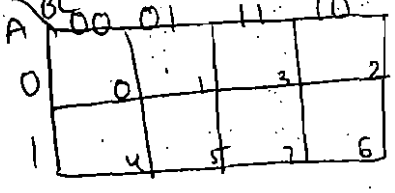
K-map for $X_4 = \bar{A}BC + A\bar{B}C$



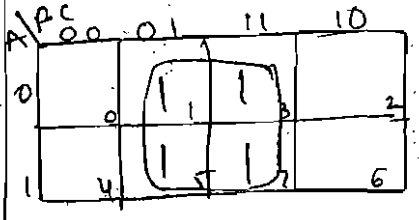
K-map for $X_3 = B\bar{C}$

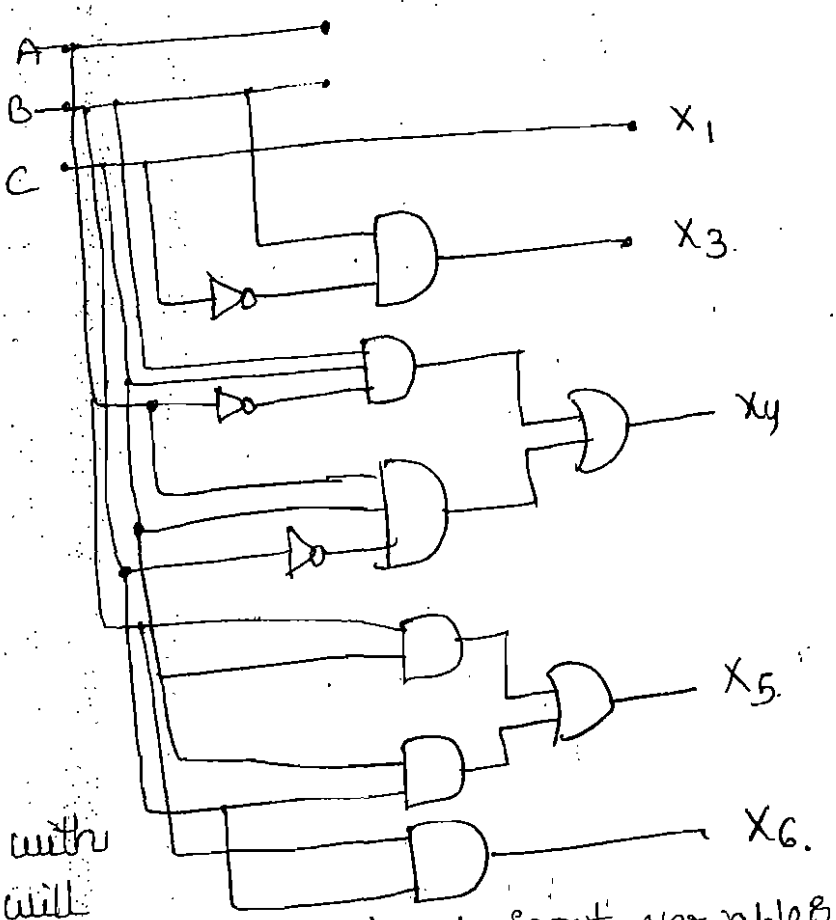


K-map for $X_2 = 0$



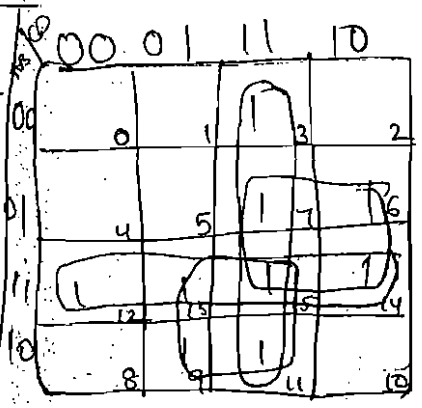
K-map for $X_1 = C$



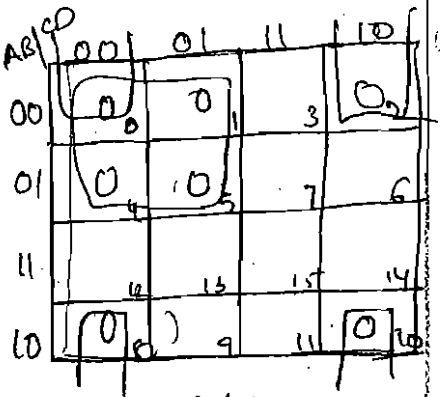


* Design a logic circuit with 4 inputs A, B, C, D that will produce output '1' only whenever two adjacent input variables are '1's. A and D are also to be treated as adjacent implement it using universal logic.

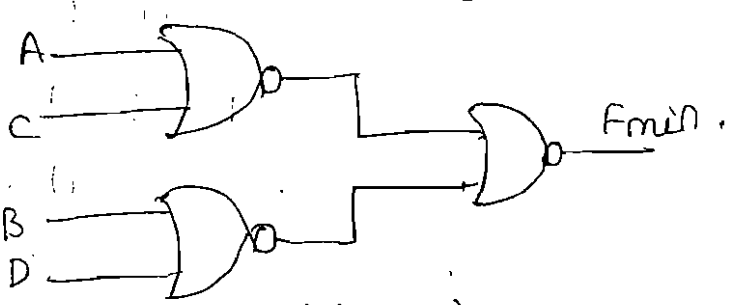
input A B C D	output f
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	0
0 1 1 0	1
0 1 1 1	1
1 0 0 0	0
1 0 0 1	1
1 0 1 0	0
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	



$F = AB + AD + BC + CD$



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

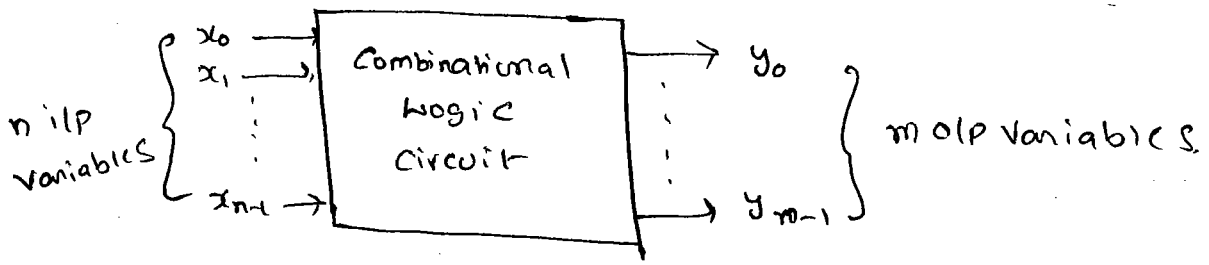


$F_{min} = (A+C)(B+D)$.

COMBINATIONAL LOGIC CIRCUITS

→ when logic gates are connected to Produce a specified o/
for certain specified combination of i/p's, no storage is invo
the resulting CKt. is called Combinational logic.

→ In this logic, o/p variables are all times depends on th
i/p variables.



Ex:- Adders, Subtractors, multiplexers, encoders, decoder's etc.

Adders:- Digital Computer Performs various arthematic operations. The most basic operation is addition.

Addition of two bits i.e $0+0=0$
$0+1=1$
$1+0=1$ → sum
$1+1=0,1$ → carry

In the above addition $1+1=10$ leads to the operation of Halfadder.

Similarly. The CKt. which Performs addition of 3-bits [Two significant bits & a Previous carry] is called a Full Adder.

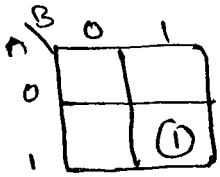
HALF ADDER :-

It has two inputs : Augend & addend bits and two outputs: Such as Sum & Carry.

Truth table :-

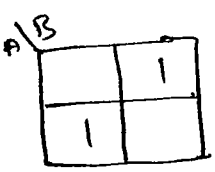
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-map for Carry



Carry = AB

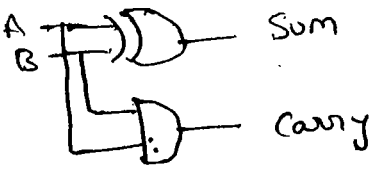
K-map for Sum



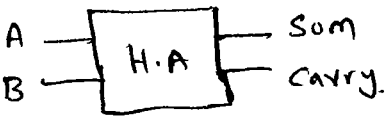
Sum = $A\bar{B} + \bar{A}B$

$= A \oplus B$

logic diagram



Block diagram:



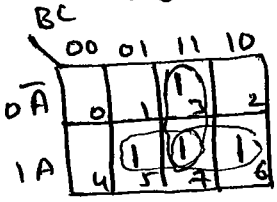
FULL ADDER :-

It is a combinational logic CKt which consists of 3 i/ps & 2 o/ps. inputs are A, B, Cin and outputs are carry, Sum.

Truth table :-

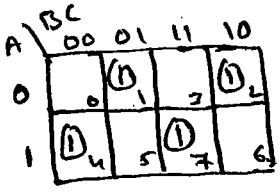
A	B	Cin	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-map for Carry



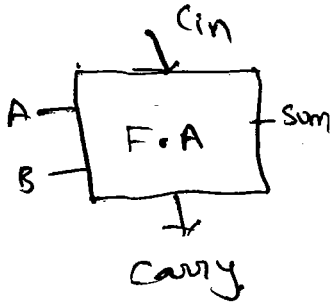
Carry = $AB + ACin + BCin$.

K-map for Sum

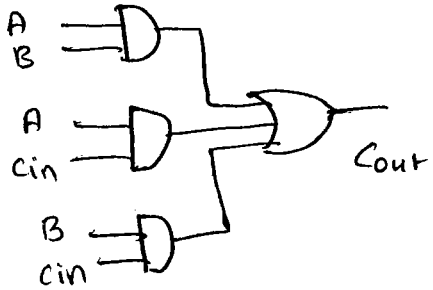


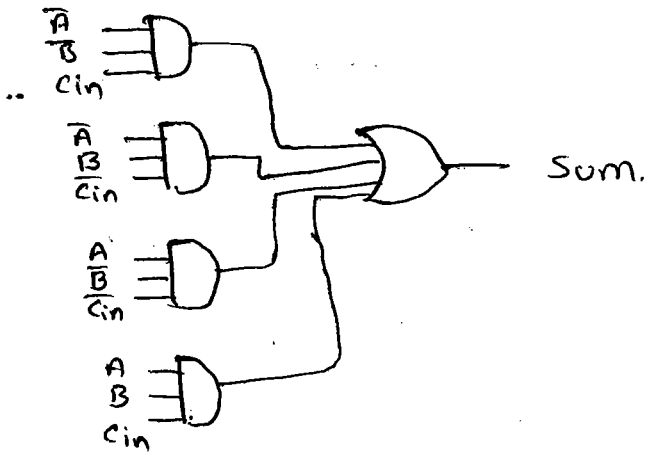
Sum = $\bar{A}\bar{B}Cin + \bar{A}B\bar{C}in + A\bar{B}\bar{C}in + ABCin$

logic diagram



logic CKt diagram :-





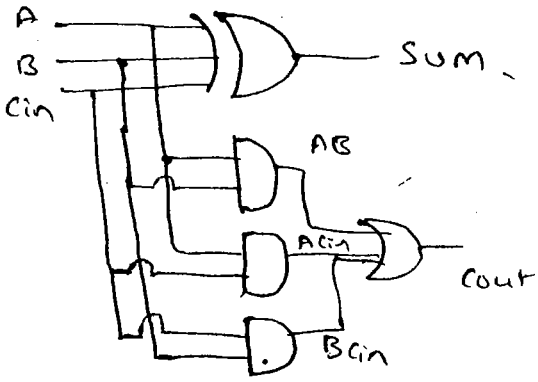
Sum = $\bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

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

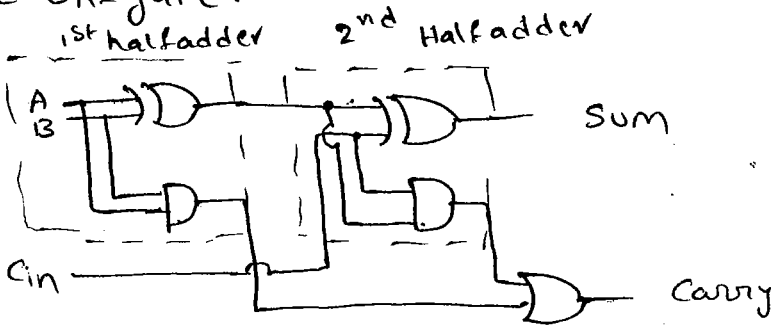
$= C_{in}(A \odot B) + \bar{C}_{in}(A \oplus B)$

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

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



* Full-adder can be implemented by 2 half adders and one OR-gate.



Subtractor :- These are two types. (1) Half Subtractor.
(2) Full Subtractor.

Half Subtractor :- It has two inputs "minuend" and "subtrahend". It performs A-B where A is minuend, B is subtrahend.

Truth table :-

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

K-map for diff

A \ B	0	1
0		1
1	1	

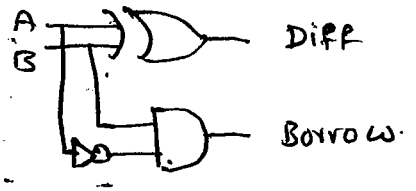
K-map for Borrow

A \ B	0	1
0		1
1		

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

Borrow = $\bar{A}B$

Logic diagram



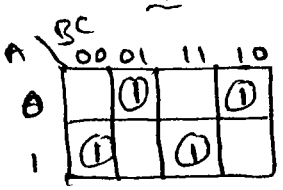
~~Full Adder~~ :

Full Subtractor :- It is a combinational logic CKt. which has 3 inputs and two outputs.

Truth table :-

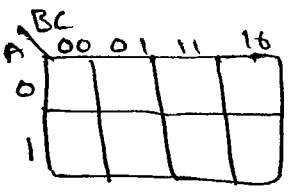
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map for D



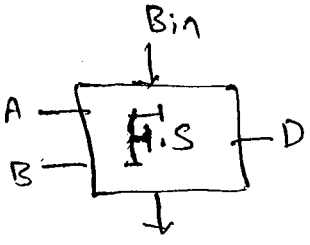
$D = \bar{A}\bar{B}Bin + \bar{A}B\bar{Bin} + A\bar{B}\bar{Bin} + ABBin$

K-map for Borrow

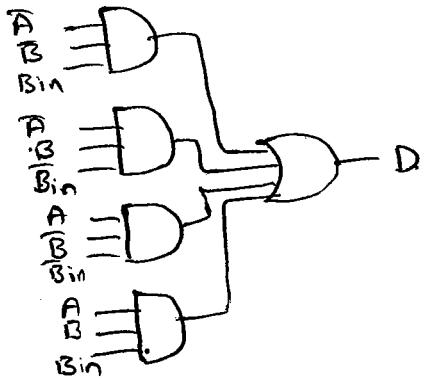


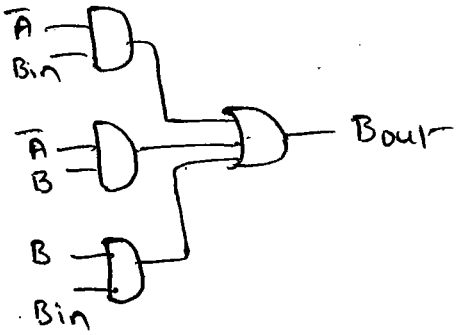
$Bout = \bar{A}Bin + \bar{A}B + BBin$

Block diagram



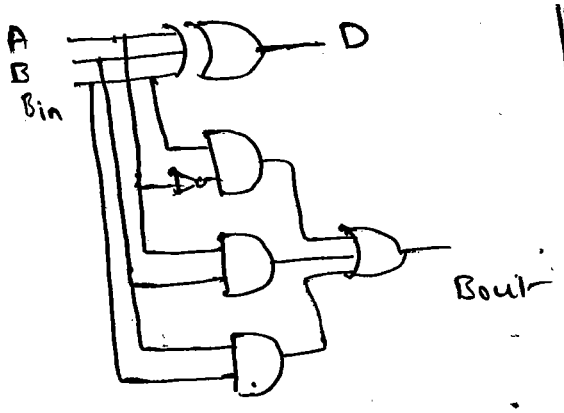
Logic diagram :-

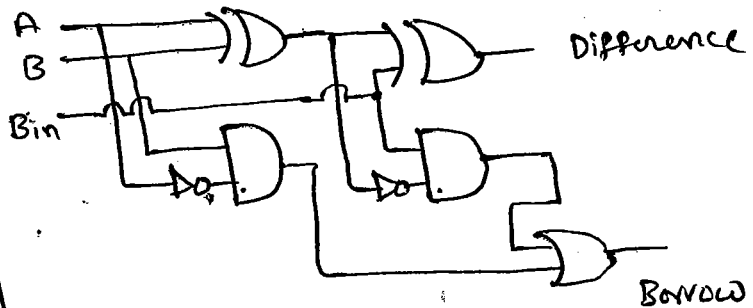




The boolean function can be simplified as

Full Subtractor using two half Subtractors





$D = \bar{A}\bar{B}Bin + \bar{A}B\bar{Bin} + A\bar{B}\bar{Bin} + ABBin$

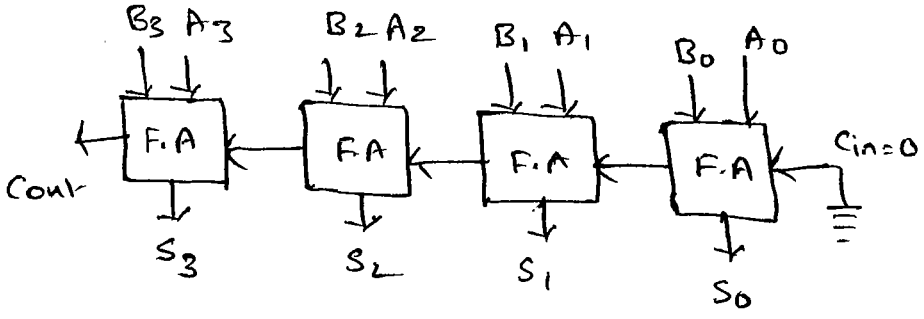
$= Bin[\bar{A}\bar{B} + AB] + \bar{Bin}[\bar{A}B + A\bar{B}]$

$= Bin[A \overline{\oplus} B] + \bar{Bin}[A \oplus B]$

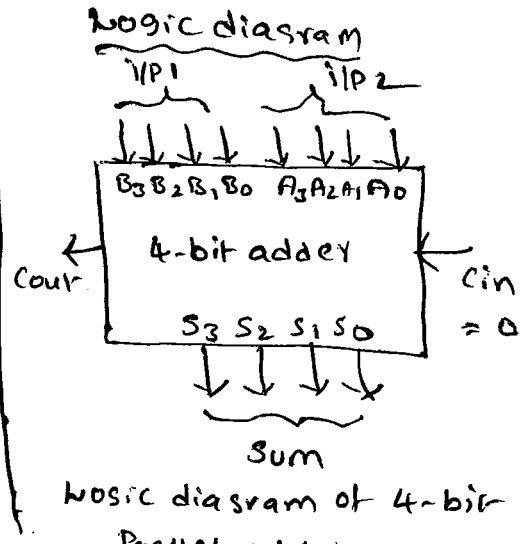
$= Bin \overline{\oplus} [A \oplus B]$

4-BIT BINARY ADDER:-

It is capable of adding two 1-bit no's & i/p carry. Inorder to add more than '1' bit additional full adders must be employed. In order to get this the full adder CKts are connected in cascade i.e; the carry o/p of each adder is connected to the carry i/p of the next higher order



Block diagram diagram of 4-bit fulladder

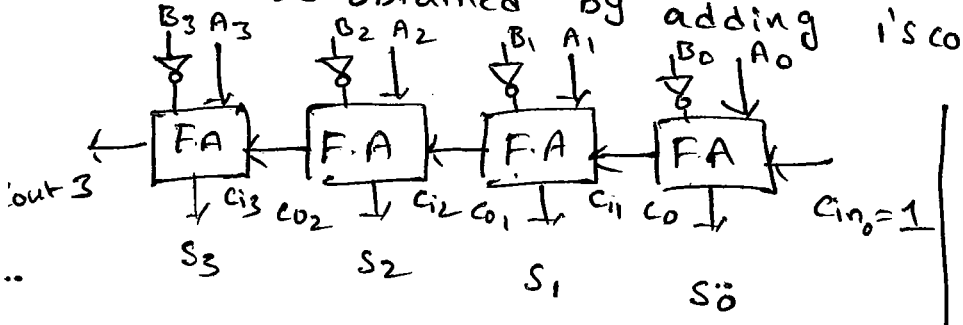


Logic diasram of 4-bit Parllel adder

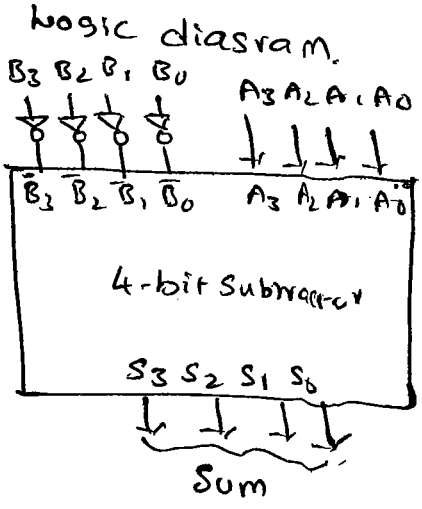
4-BIT Binary Subtractor :-

→ The subtractor of binary no's. can be done most conveniently by means of complements. General subtraction can be Adding with 2's complement of B that means A-B = A+(2'scomp of B)

→ It can be obtained by adding 1's comp to '1' to LSB of bits



4-bit Parllel Subtractor



rallel Adder / Subtractor :-

→ The addition & Subtraction operations can be combined into on one CKt with one common binary adder. This is done by including an exclusive-OR gate with each full adder.

→ If mode 'm' controls the operation when m=0, the CKt is an adder. When m=1, the CKt becomes Subtractor CKt.

→ When m=0, we have B⊕0 = B. The full adders receive the value of 'B', & the i/p carry (Cin=0). and the CKt Performs A+B.

→ When m=1, we have B⊕1 = B̄ & Cin=1. The 'B' i/ps are all complemented & a '1' is added through the i/p carry.

∴ The CKt Performs A+2's comp of B i.e A−B.

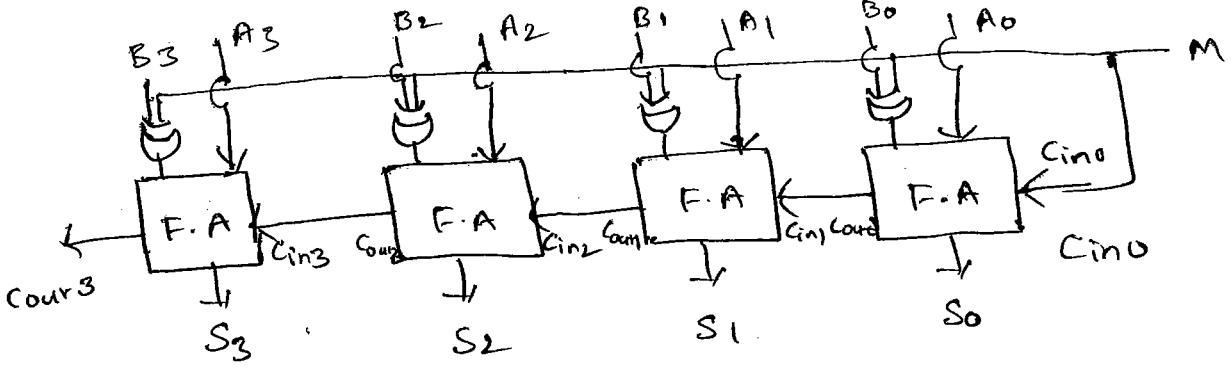


fig. 4-bit adder/subtractor CKt.

Decimal BCD ADDER :-

→ A BCD adder is a CKt that adds two BCD digits & Produces a Sum digit also in BCD.

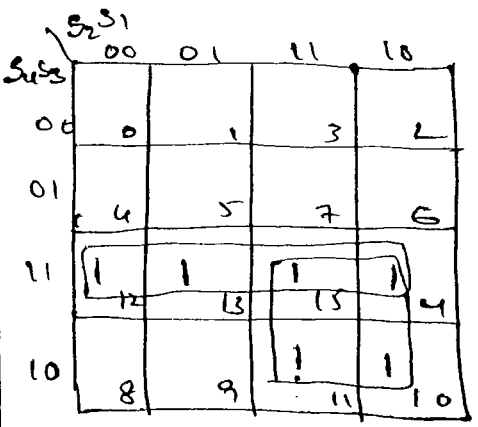
→ BCD no's use 10 digits, 0 to 9 which are represented in the Binary form 0000 to 1001, i.e each BCD digit is represented as a 4-bit binary no.

* When Sum ≤ 9 and Cout = 0 nothing (zero) is added to the binary Sum.
* When Sum > 9 and Cout = 1 binary 0110 is added to the binary Sum through the bottom 4-bit binary adder, the o/p generated is ignored.

Inputs	o/ps
1010	1
1011	1
1100	1
1101	1
1110	1
1111	1

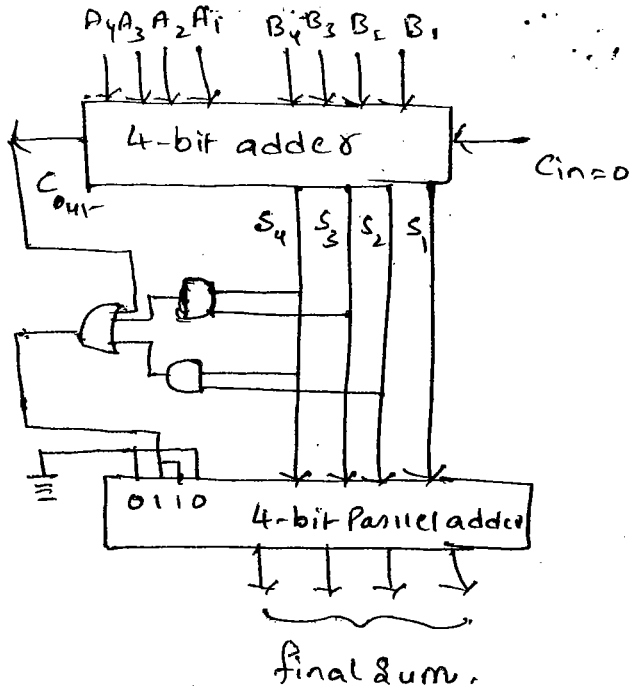
Remaining all inputs the o/p should be zero

K-map :-



$Y = S_4S_3 + S_4S_2$ [>9, Cout = 1]

CKT diagram :-



Excess-3 adder CKT :-

In excess-3 addition

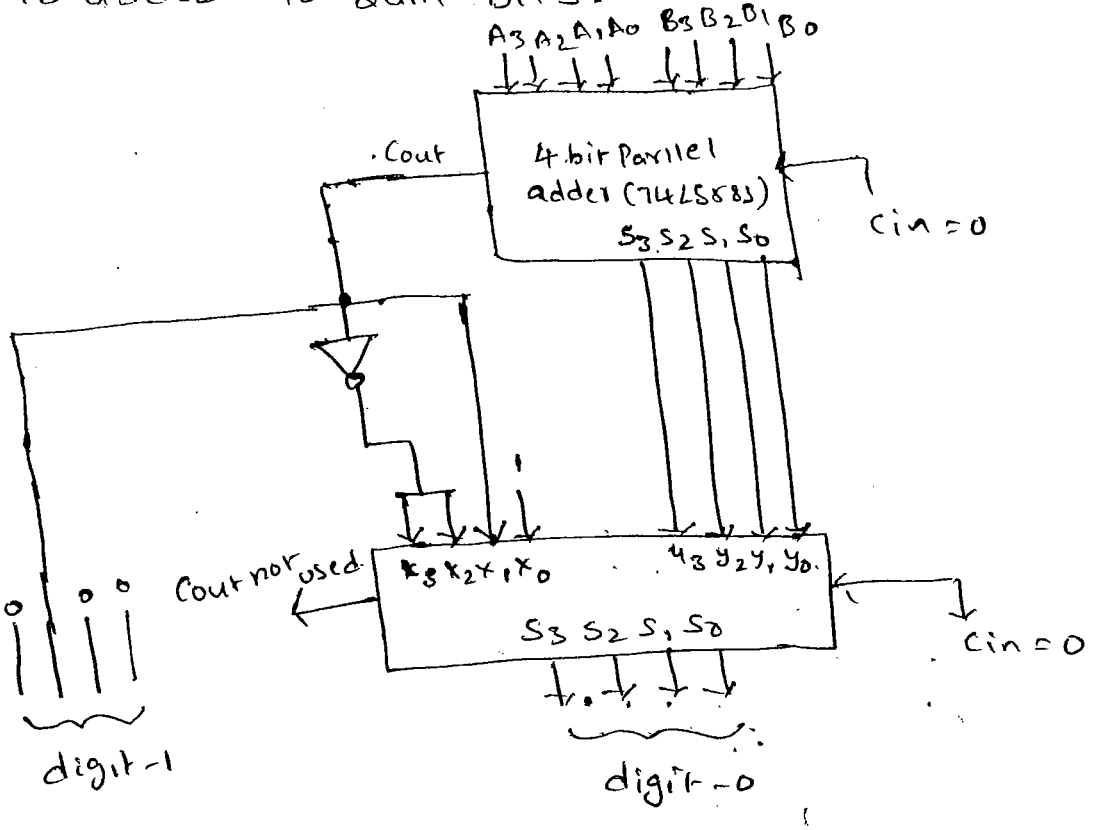
① Add two xs-3 code groups.

② If carry = 1 add 0011

if carry = 0 Subtract 0011, or add 1101 (13 in decimal)

In the following figure shows 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 Carry is 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 '0' then '01' is added to Sum bits.



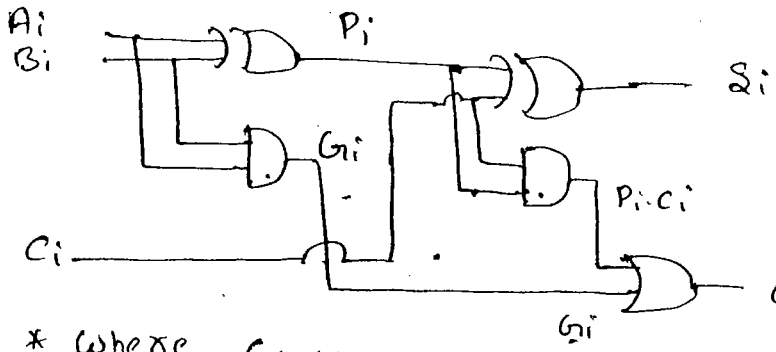
Carry Look Ahead Adder:-

* The Purpose of Carry Look Ahead Adder is to reduce the higher order delay. This can be achieved to represend each higher order delay can be represented in terms of lower order delay. In Parallel Adder CKt each full adder Cascaded Sections Produce delay this delay is known as Propagation delay. one method of Speeding UP the Process by eliminating interstage Carry delay is called "Look ahead Carry addition".

It uses 2 functions:- Carry generate & Carry-Propagate

Consider the full adder CKt.

Full adder CKt:-



$P_i = A_i \oplus B_i$

$G_i = A_i \cdot B_i$

OIP Sum $S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

* where G_i is called Carry generate & it Produces when A_i & B_i are 1! P_i is called Carry Propagate because it is the term associated with Propagation of the Carry from C_i to C_{i+1}

→ from the Boolean function

$$C_2 = G_1 + P_1 C_1$$

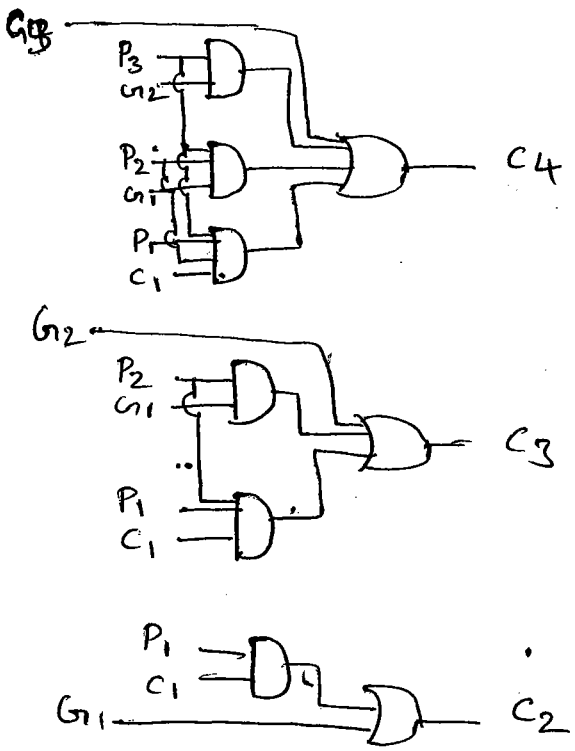
$$C_3 = G_2 + P_2 C_2$$

$$= G_2 + P_2 [G_1 + P_1 C_1] = G_2 + P_2 G_1 + P_1 P_2 C_1$$

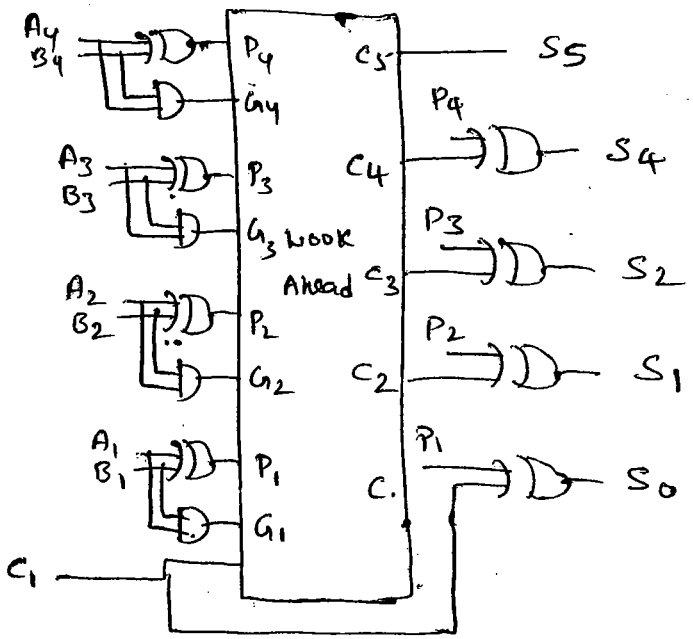
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 [G_2 + P_2 G_1 + P_1 P_2 C_1]$$

$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_1 P_2 P_3 C_1$$

CKt diagram:-

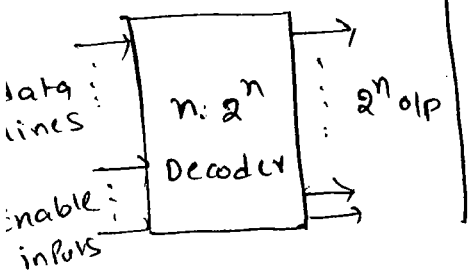


Logic IC diagram



Decoder:- It is a multiple input, multiple output logic ckt which converts coded inputs into coded o/ps, where the i/ps and o/ps are different

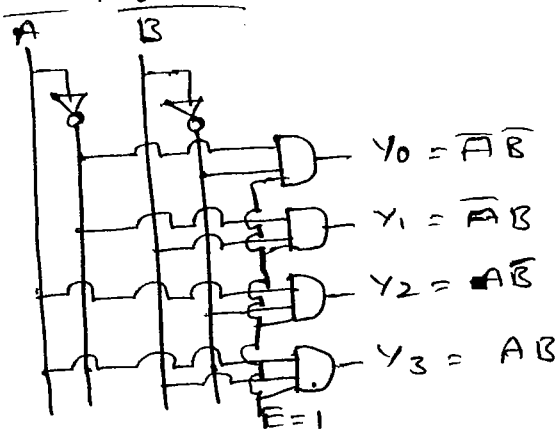
General Structure



2x4 Decoder:-

EN	A	B	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

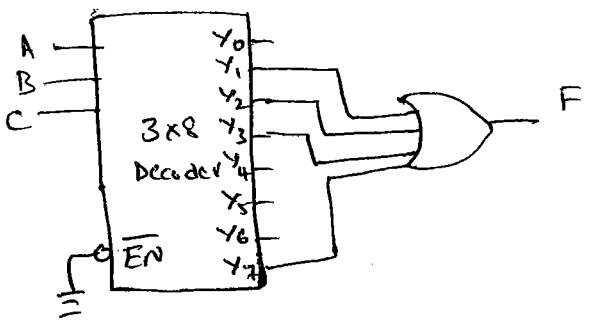
ckt diagram:-



A Decoder which has an n-bit binary input and a one activated output out of 2^n output code is called binary decoder

Implement the Boolean function $F = \Sigma m(1,2,3,7)$ using 3:8 Decoder

Sol:

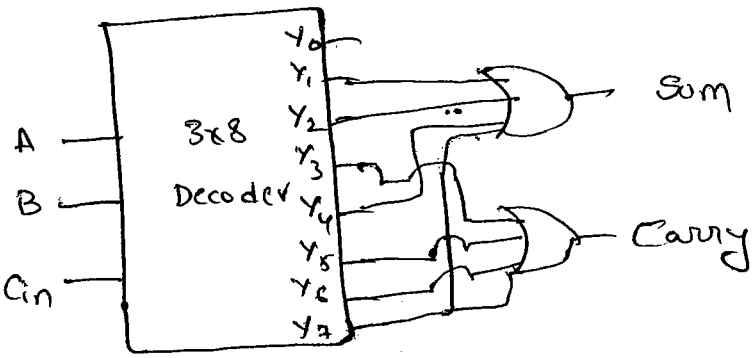


* Design and implement a full adder ckt using 3x8 decoder.

Sol:

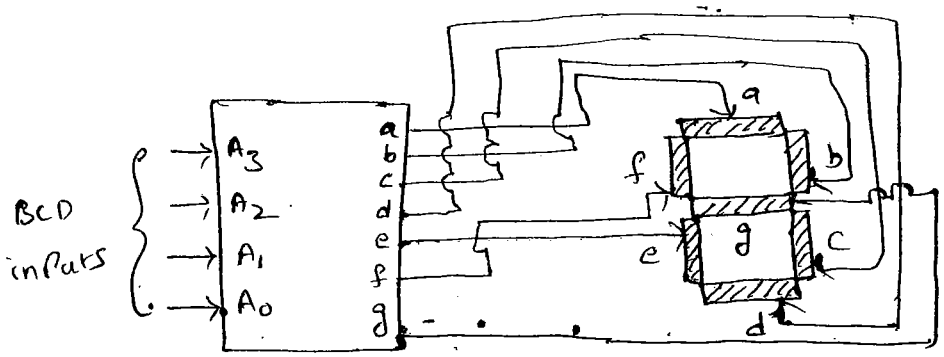
Inputs			outputs	
A	B	Cin	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Carry = $\Sigma(3,5,6,7)$, Sum = $\Sigma(1,2,4,7)$



* BCD to Seven Segment :-

This type of decoders accepts the BCD code and Provides outputs of Energise 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 materially include LED'S, incandesent filaments and LCD'S.



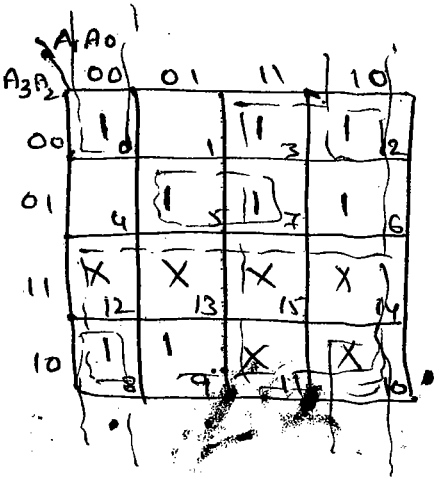
Block diagram:

Truth table :-

Decimal No:	Inputs BCD i/ps $A_3A_2A_1A_0$	outputs Segment o/ps. 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 1 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 1 0 1 1

K-map for a :-

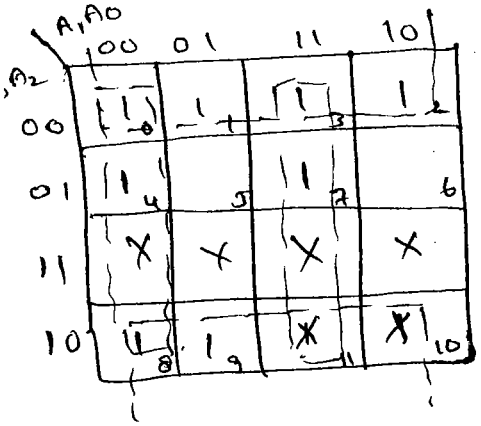
$a = \Sigma m(0,2,3,5,6,7,8,9) + d(10,11,12,13,14,15)$



$a = A_1 + A_3 + \bar{A}_2\bar{A}_0 + \bar{A}_3A_2A_0$

K-map for b:-

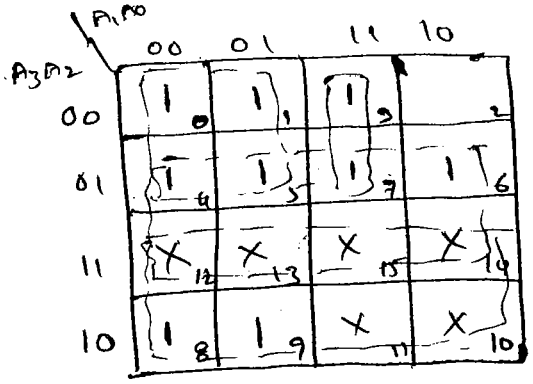
$b = \Sigma m(0,1,2,3,4,7,8,9) + d(10,11,12,13,14,15)$



$b = \bar{A}_2 + A_1\bar{A}_0 + A_1A_0$

K-map for C:-

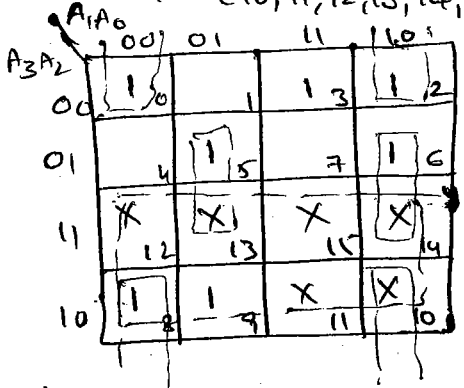
$C = \Sigma m(0,1,3,4,5,6,7,8,9) + d(10,11,12,13,14,15)$



$C = A_2 + A_3 + \bar{A}_3\bar{A}_1 + \bar{A}_3A_1A_0$

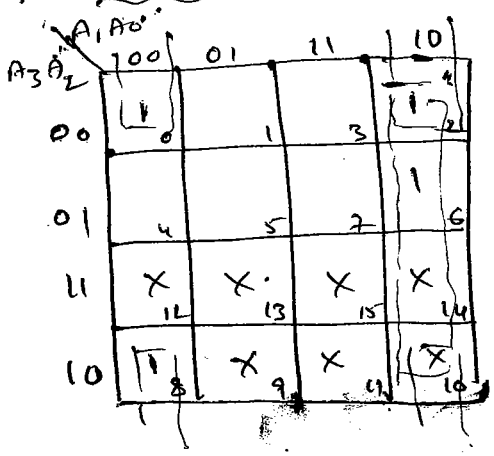
K-map for d

$d = \Sigma m(0,2,3,5,6,8,9) + d(10,11,12,13,14,15)$



$d = A_3 + \bar{A}_2A_1 + A_2\bar{A}_1A_0 + A_2A_1\bar{A}_0 + \bar{A}_2\bar{A}_0$

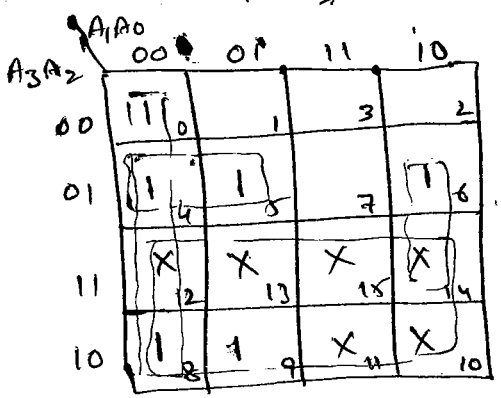
K-map for e: $e = \Sigma m(0,2,6,8) + d(10,11,12,13,14,15)$



$e = A_1\bar{A}_0 + \bar{A}_2\bar{A}_0$

K-map for f:-

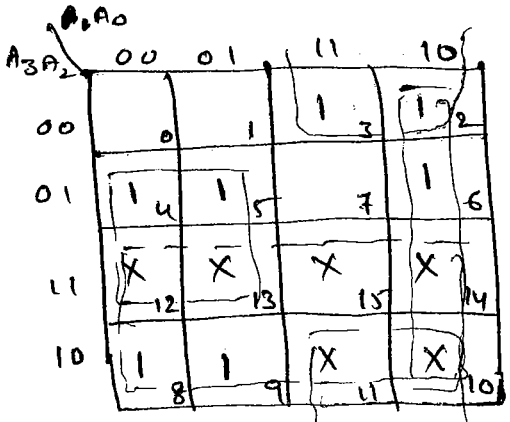
$f = \Sigma m(0,4,5,6,8,9) + d(10,11,12,13,14,15)$



$f = \bar{A}_1\bar{A}_0 + A_3 + \bar{A}_3A_2\bar{A}_1 + A_2A_1\bar{A}_0$

K-map for G

$g = \Sigma m(2,3,4,5,6,8,9) + d(10,11,12,13,14,15)$



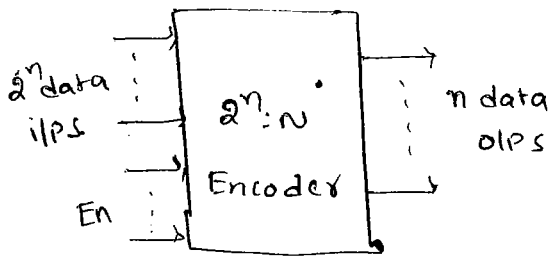
$G = A_3 + A_1\bar{A}_0 + A_2\bar{A}_1 + \bar{A}_2A_1$

Applications of Decoder:-

- It is used as code converters.
- Implementation of combinational circuits.
- Address decoding.
- BCD to 7-segment decoder.

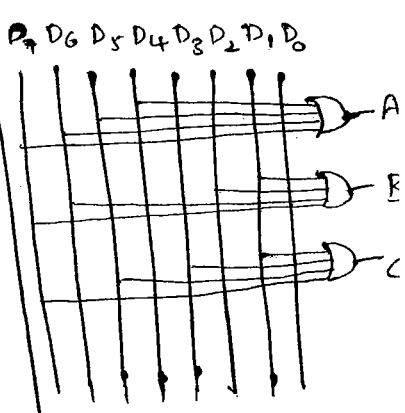
Encoder:- It performs inverse operation of decoder. It has 2^N inputs and N outputs.

Block diagram:-



Octal to Binary Encoder

ilps								olps		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1



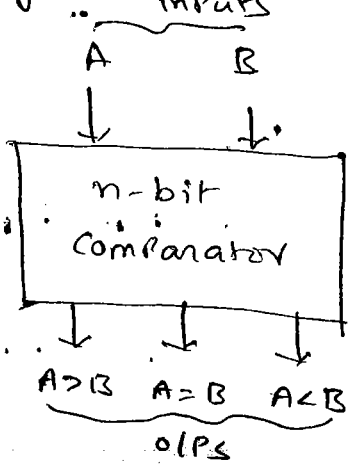
$A = D_4 + D_5 + D_6 + D_7$

$B = D_2 + D_3 + D_6 + D_7$

$C = D_1 + D_3 + D_5 + D_7$

Comparator:-

A comparator is a special combinational CKT designed primarily to compare the relative magnitudes of two binary numbers. Following is the block diagram of comparator. It receives two n-bit numbers A and B as inputs and the outputs are A>B, A=B and A<B depending on Relative magnitudes of the two numbers, one of the outputs will be high.

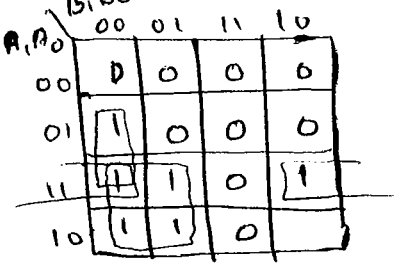


Design 2-bit comparator using gates :-

inputs				O/Ps		
A_1	A_0	B_1	B_0	$A>B$	$A=B$	$A<B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

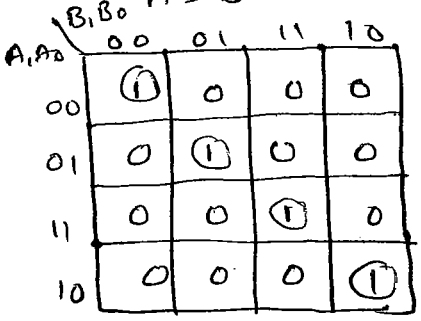
K-maps

$A > B$



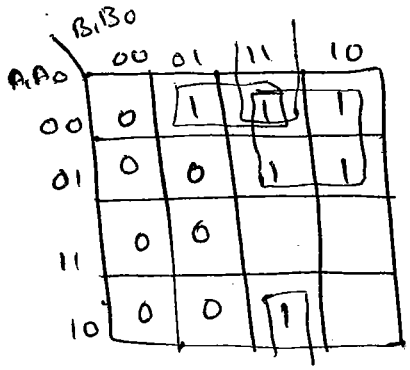
$Y(A>B) = A_0\bar{B}_1\bar{B}_0 + A_1\bar{B}_1 + A_1A_0\bar{B}_0$

$A = B$



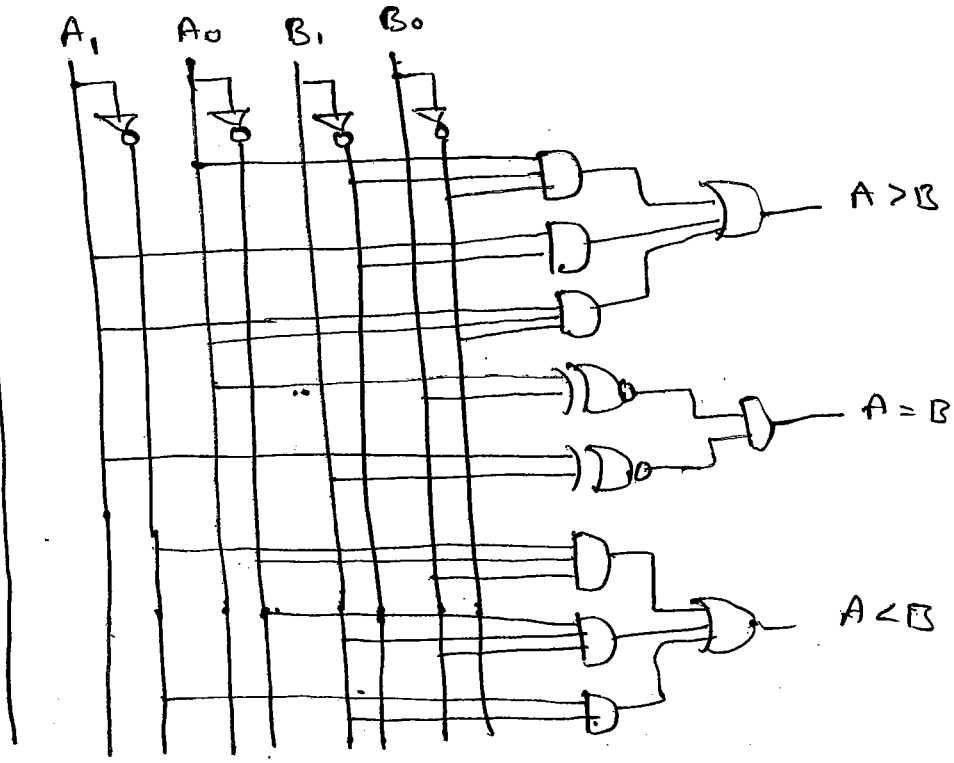
$Y(A=B) = \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0)$

$= (A_0 \odot B_0)(A_1 \odot B_1)$



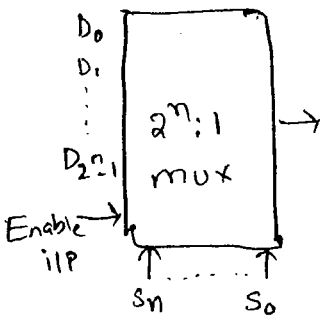
$Y(A<B) = \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0 + \bar{A}_1B_1$

Logic diagram :-



(8) **Multiplexer:-** It is a digital switch which allows the digital information through several inputs and single o/p.

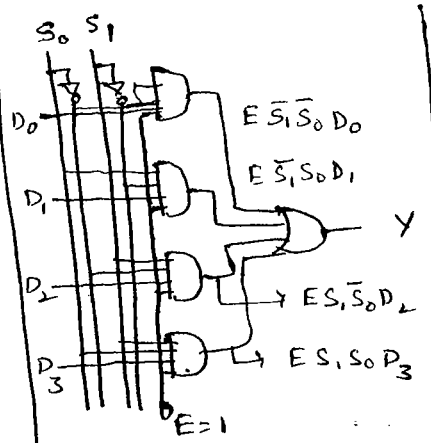
Block diagram:-



4×1 mux:-

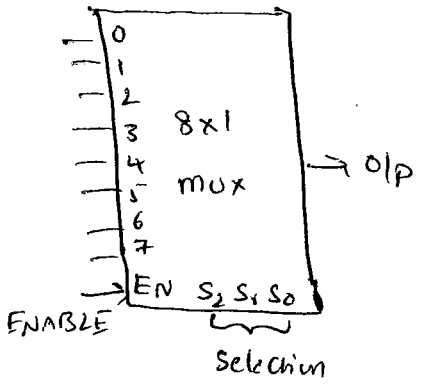
E	S_1	S_0	Y
0	x	x	0
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3

$Y = E\bar{S}_1\bar{S}_0D_0 + E\bar{S}_1S_0D_1 + ES_1\bar{S}_0D_2 + ES_1S_0D_3$.



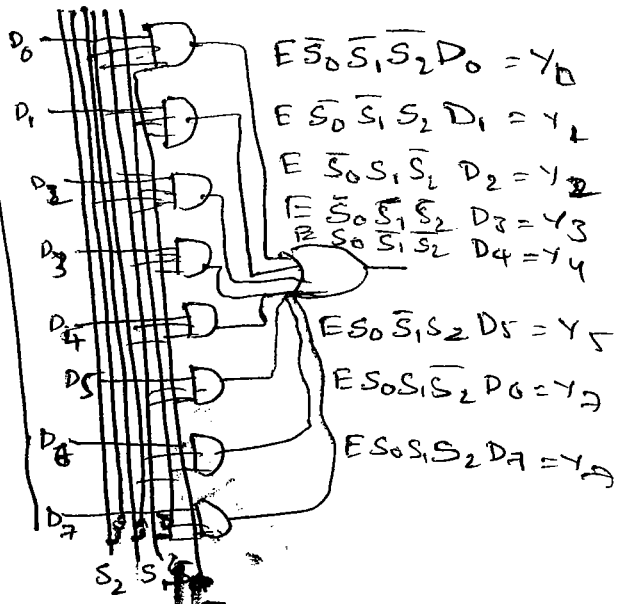
8×1 multiplexer:-

S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7



Logic symbol

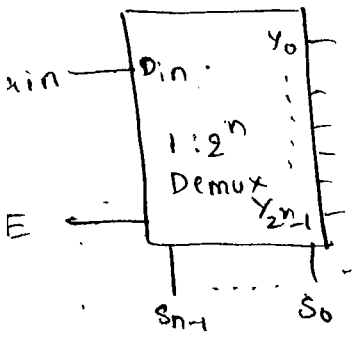
Logic diagram:-



Demultiplexers:-

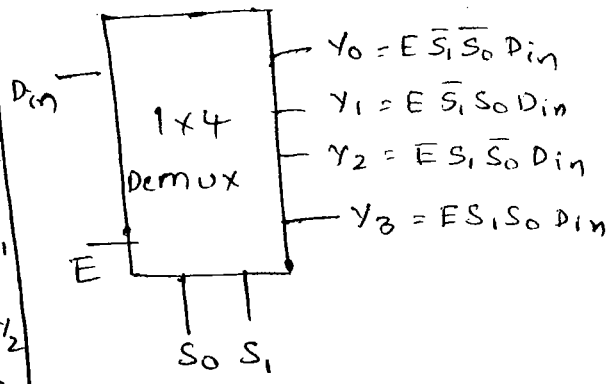
A Demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2ⁿ possible o/p lines. The selection of specific outputline is controlled by the values of n selection lines.

Block diagram :-

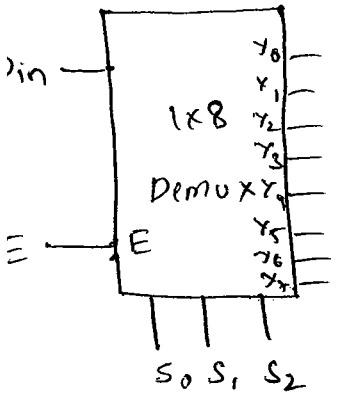


1:4 Demux :-

E	S_1	S_0	Din	Y_0	Y_1	Y_2	Y_3
0	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1



1:8 Demux



E	Select i/Ps			outputs								
E	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	
0	X	X	X	0	0	0	0	0	0	0	0	
1	0	0	0	Din	0	0	0	0	0	0	0	$= E\bar{S}_2\bar{S}_1\bar{S}_0 Din$
1	0	0	1	0	Din	0	0	0	0	0	0	$= E\bar{S}_2\bar{S}_1 S_0 Din$
1	0	1	0	0	0	Din	0	0	0	0	0	
1	0	1	1	0	0	0	Din	0	0	0	0	
1	1	0	0	0	0	0	0	Din	0	0	0	
1	1	0	1	0	0	0	0	0	Din	0	0	
1	1	1	0	0	0	0	0	0	0	Din	0	
1	1	1	1	0	0	0	0	0	0	0	Din	$= E S_2 S_1 S_0 Din$

Differentiate between mux and Demultiplexer :-

Parameters	multiplexers	Demultiplexers
1) Definition	It is a digital switch which allows digital information from several sources to be routed onto a single output line.	Demultiplexer is a CKT that receives information on a single line and transmits this information on one of 2^n possible output lines.
2) No. of Data inputs	2^n	1
3) No of Data outputs	1	2^n
4) Relation	many to one	one to many
5) Application	• used as Data selector • Time Division multiplexing at transmitting End	• used as Data distributor • Time division multiplexing at Receive end

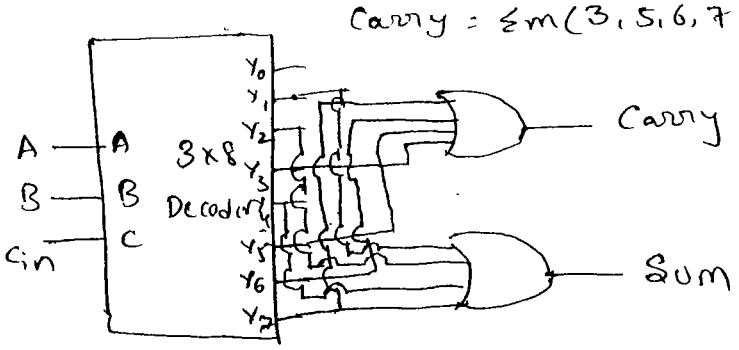
(1) Design and implement of full adder CKT using 3×8 decoder

Sol:-

IIPS			OIPS	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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

$Carry = \Sigma m(3,5,6,7)$



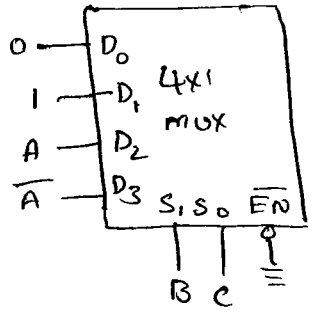
(2) Implement the following using 4:1 multiplexer

Sol:- $F(A,B,C) = \Sigma m(1,3,5,6)$

	D_0	D_1	D_2	D_3	
$\bar{A}$	0	(1)	2	(3)	row 1
A	4	(5)	(6)	7	row 2
	0	1	A	$\bar{A}$	

most significant variable

(a) implementaion table



(b) multiplexer implementation.

(3) Implement the following Boolean function using 8×1 multiplexer

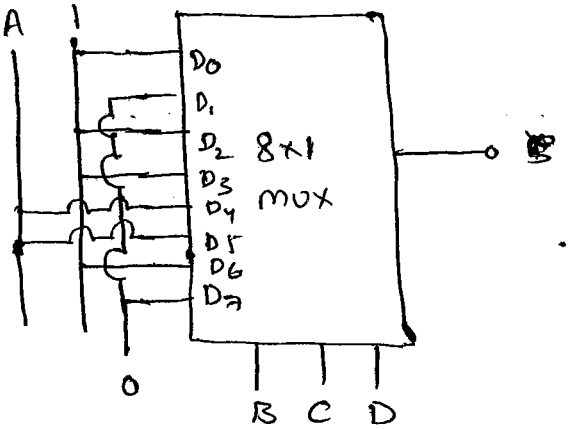
$F(A,B,C,D) = \Sigma m(0,2,6,10,11,12,13) + d(3,8,14)$

Sol:-

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	(0)	1	(2)	(3)	4	5	(6)	7
A	(8)	9	(10)	(11)	(12)	(13)	(14)	15
	1	0	1	1	A	A	1	0

Here don't cares are treated as 1's

(a) implementation Table.

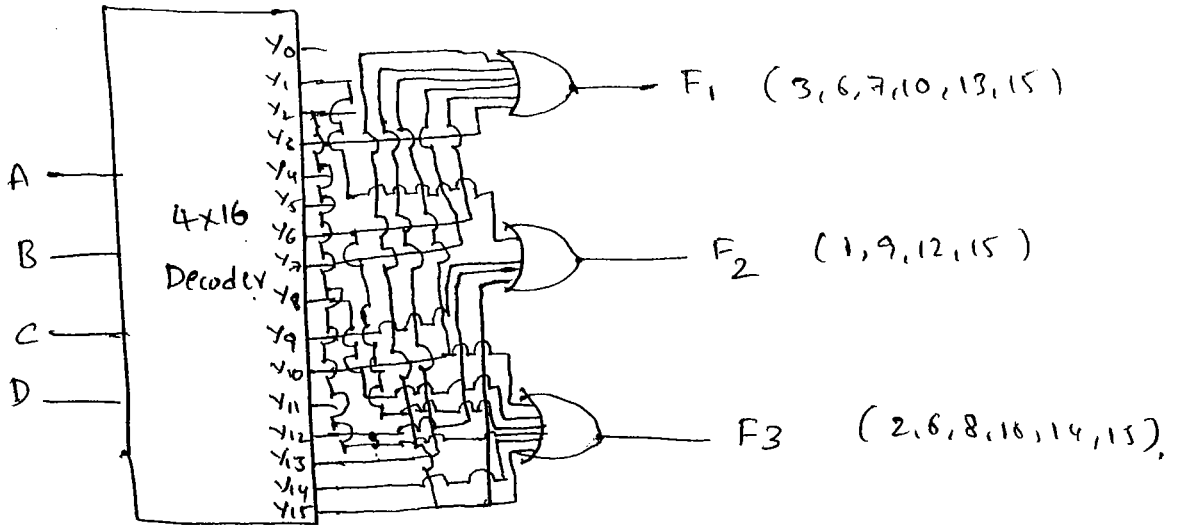


(b) implementation.

3)(f) Implement the following Boolean functions with Decoder

(i) $F_1 = \Sigma(3,6,7,10,13,15)$ (ii) $F_2 = \Sigma(1,9,12,15)$ (iii) $F_3 = \Sigma(2,6,8,10,14,15)$

Soln:-



* Application of Decoder:-

* The uses of Decoders are:
  - Code Converters
  - Implementation of Combinational CKT's
  - Address decoding
  - BCD to 7 segment decoder

* Applications of Demultiplexers:-

* It can be used as decoder.
* It can be used as data distributer
* It can be used as time division - multiplexing at the receiving end as a data seperator
* Used to implement Boolean expressions.

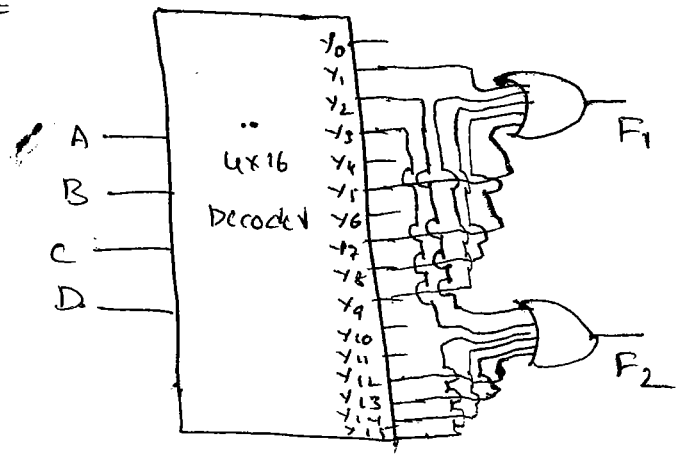
Applications of multiplexer:-

* They are used as a data selector to select one out of many data inputs.
* used implement combinational logic CKt.
* Used as Time Division, frequency Division multiplexing systems.
* Used in data acquisition systems

*Realize the following Boolean function using Decoder and logic OR gate

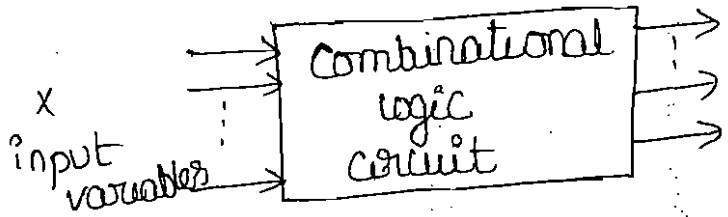
$f_1 = \Sigma(1,5,7,8,9)$, $f_2 = \Sigma(2,3,12,13,14,15)$

Soln:-



126 Combinational logic circuits design.

Logic circuits for digital systems may be combinational & sequential. In combinational circuits, the output variable 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.

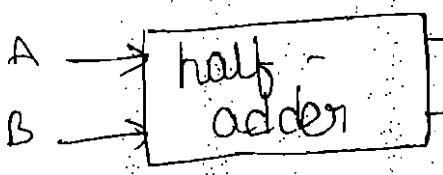


Adders :-

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

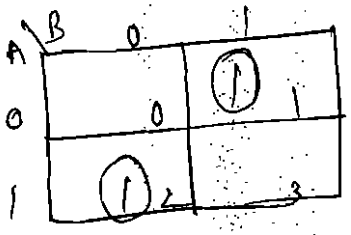
A half adder is a combinational circuit with two binary inputs. (augend and addend bits) and two outputs. Sum and carry.



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.

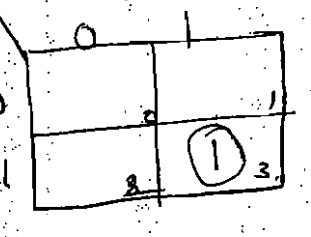
K-map for S.



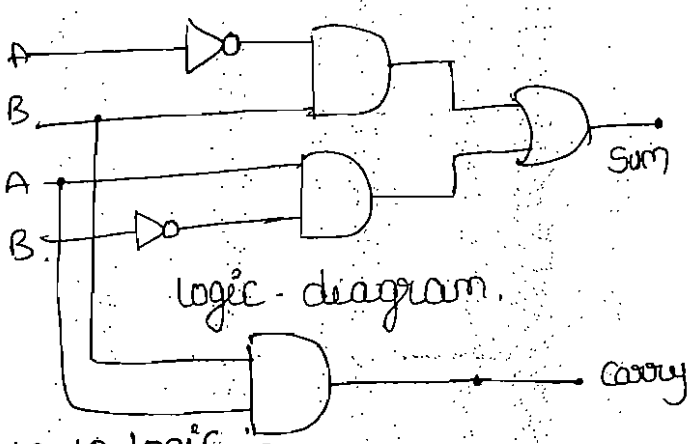
$S = A\bar{B} + \bar{A}B$

$= A \oplus B$

K-map for C

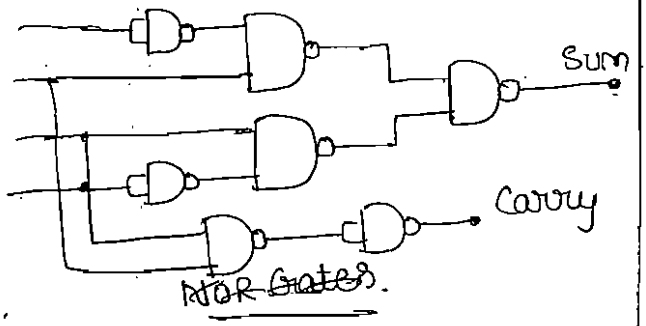


$C = A \cdot B$



Logic - diagram.

NAND Gates.



NOR Gates.

NAND logic :-

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

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

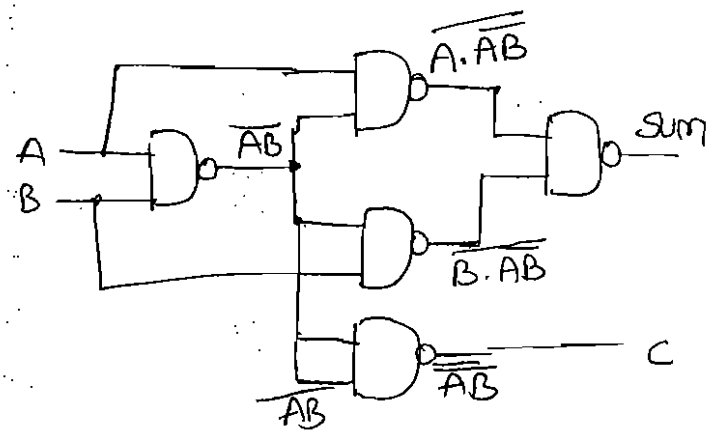
$= A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$

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

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

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

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



NOR logic

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

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

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

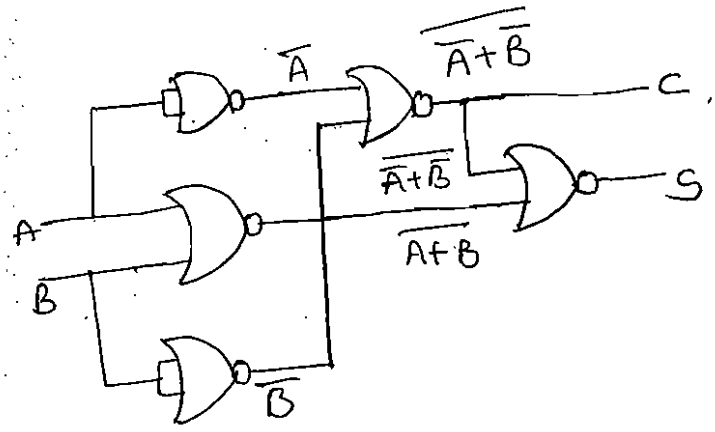
$= (A+B)(\bar{A}+\bar{B})$

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

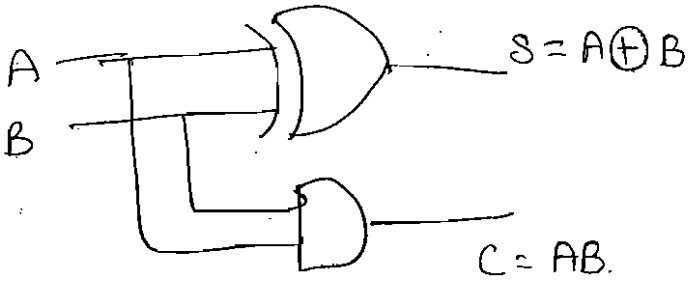
$= \overline{(\bar{A}+B)} + \overline{(\bar{A}+\bar{B})}$

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

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

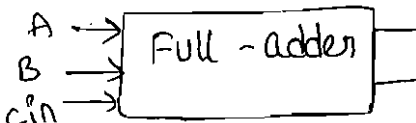


Simple logic diagram is.



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}.

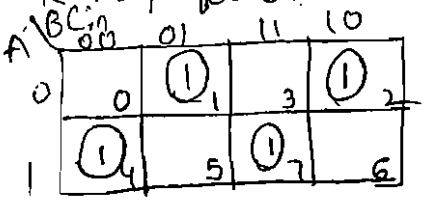


Block diagram.

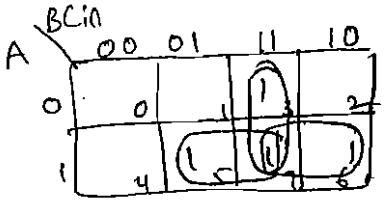
inputs			outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table

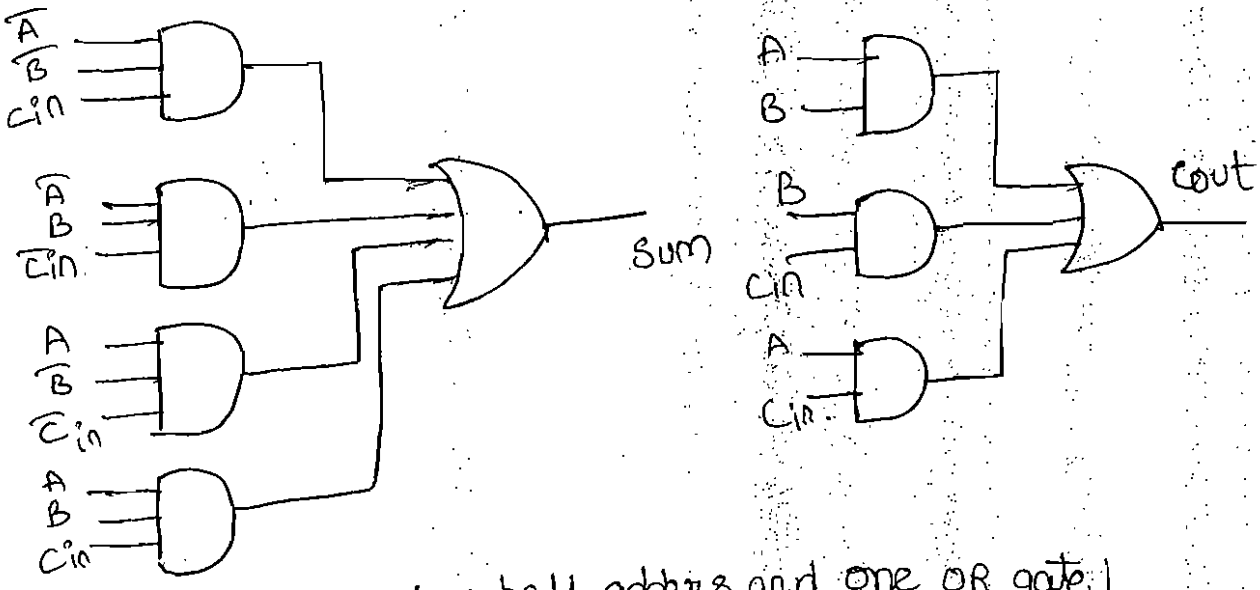
K-map for S.



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



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



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

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

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

$= \overline{A \oplus B}\, C_{in} + \bar{C}_{in}(A \oplus B)$

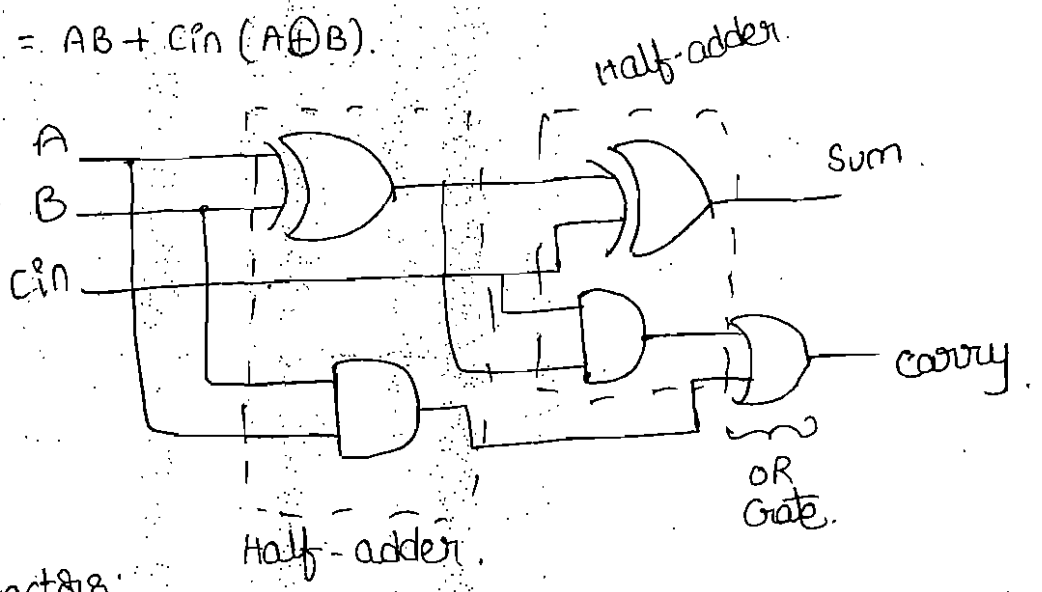
$= A \oplus B \oplus C_{in}$.

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

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

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

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

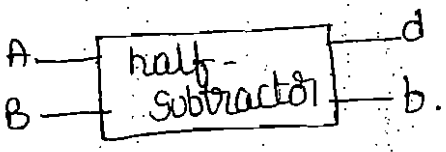


Subtractors:-

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

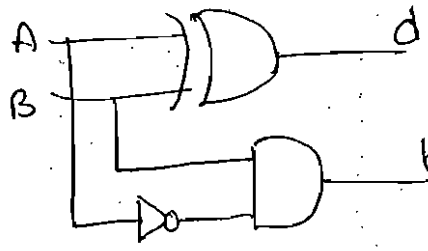
$= A\bar{B} + \bar{A}B$

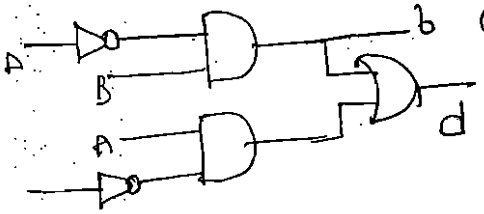
$= A \oplus B$

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

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

$b = \bar{A}B$





Logic diagrams of a half-subtractor.

NAND logic:-

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

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

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

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

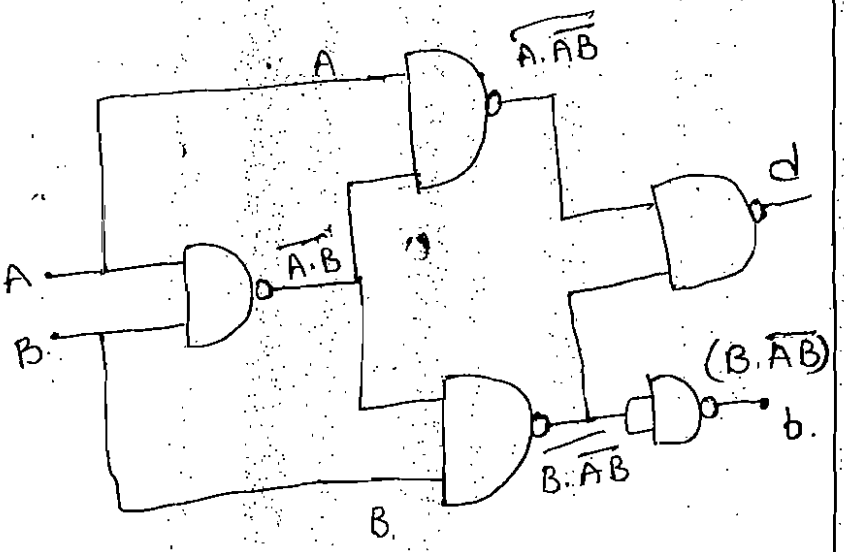
$= A.\overline{\bar{A}B}.\ B.\overline{\bar{A}B}$

$b = \bar{A}B$

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

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

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



NOR logic

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

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

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

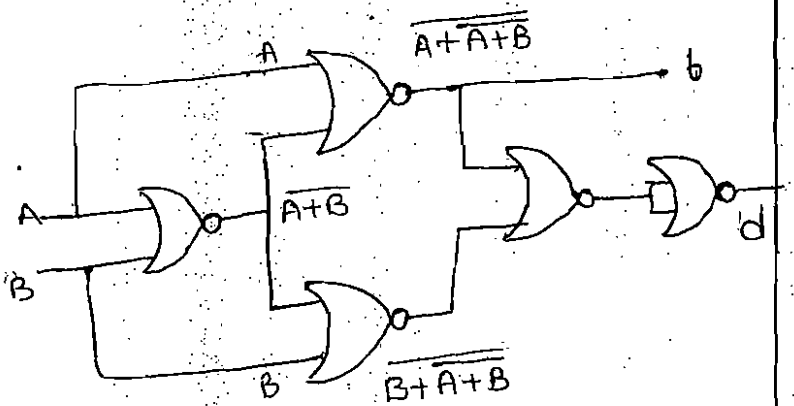
$= \overline{B+\bar{A}+B} + \overline{A+\bar{A}+B}$

$b = \bar{A}B$

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

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

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



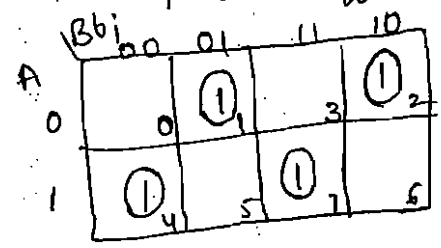
Full - Subtractor:-

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).

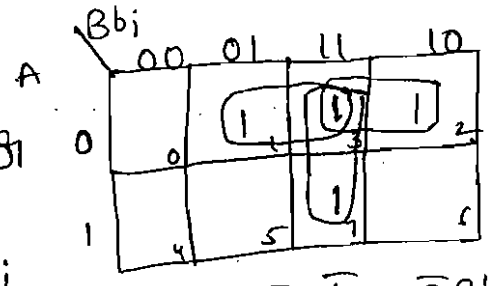
inputs			outputs	
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map for difference



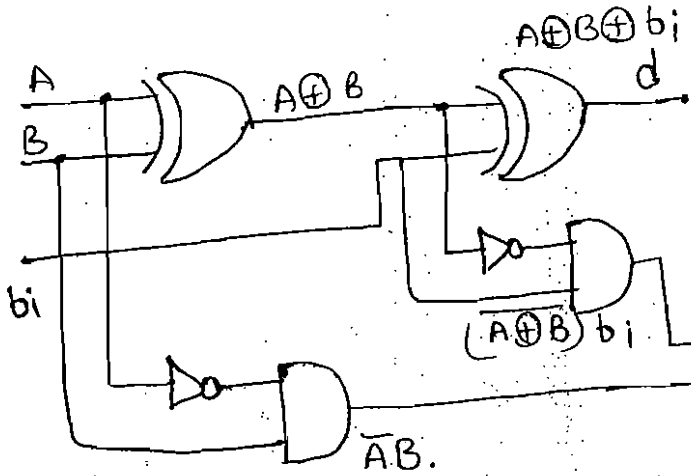
$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + A\bar{B}\bar{b_i} + AB b_i$$
$$= b_i(AB + \bar{A}\bar{B}) + \bar{b_i}(\bar{A}B + A\bar{B})$$
$$= b_i(\overline{A \oplus B}) + \bar{b_i}(A \oplus B)$$
$$= A \oplus B \oplus b_i$$

K-map for borrow.



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

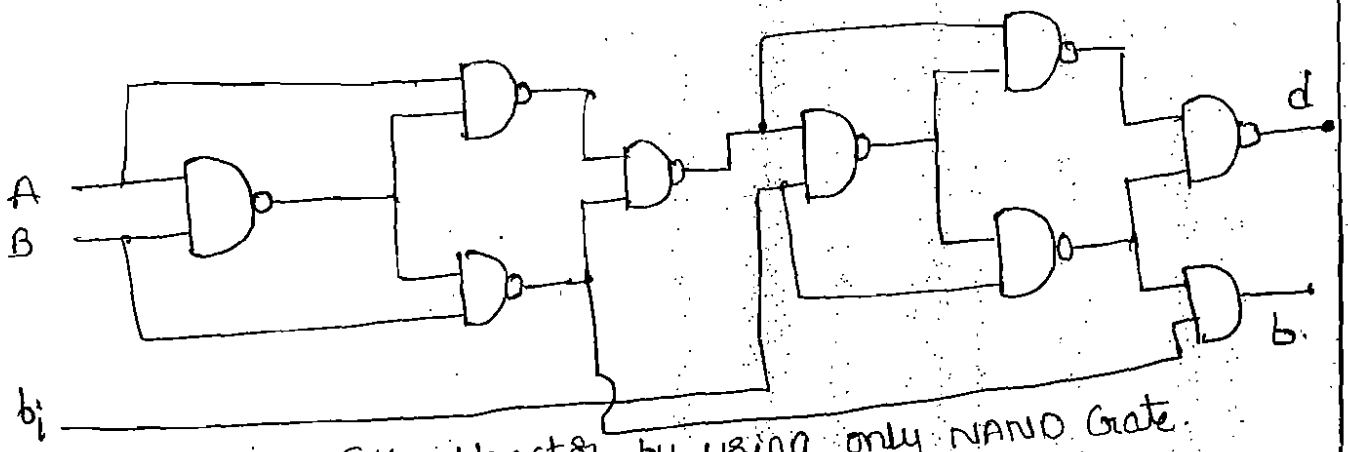
Full-Subtractor by using two half-Subtractor



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 subbractor by using only NAND Gate.

NOR Logic.

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

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

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

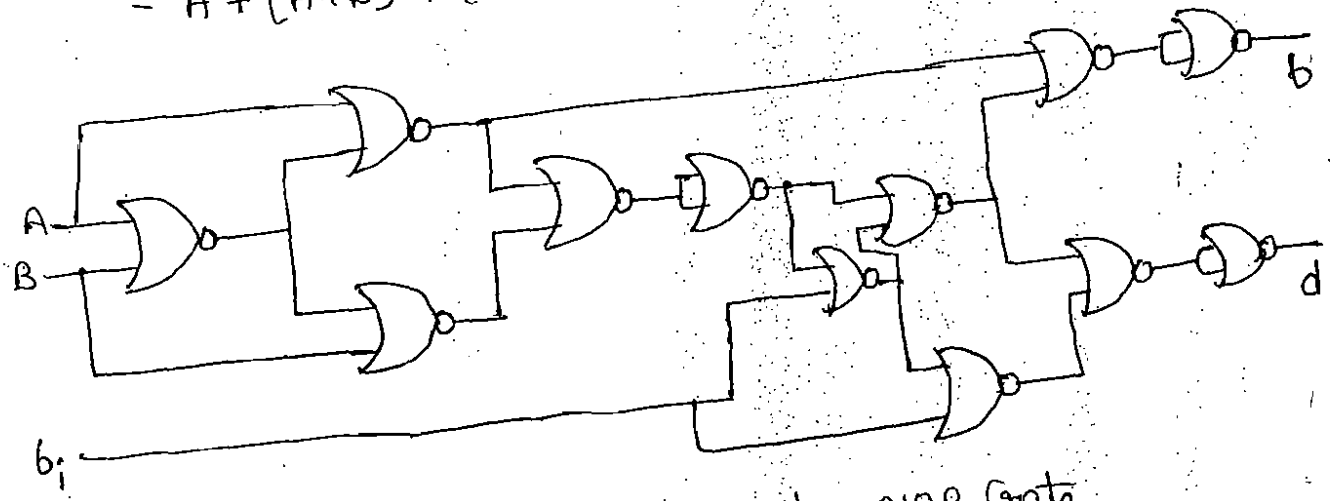
$$= \overline{(A \oplus B) + (\overline{A \oplus B}) + b_i + b_i + (A \oplus B) + b_i}$$

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

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

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

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

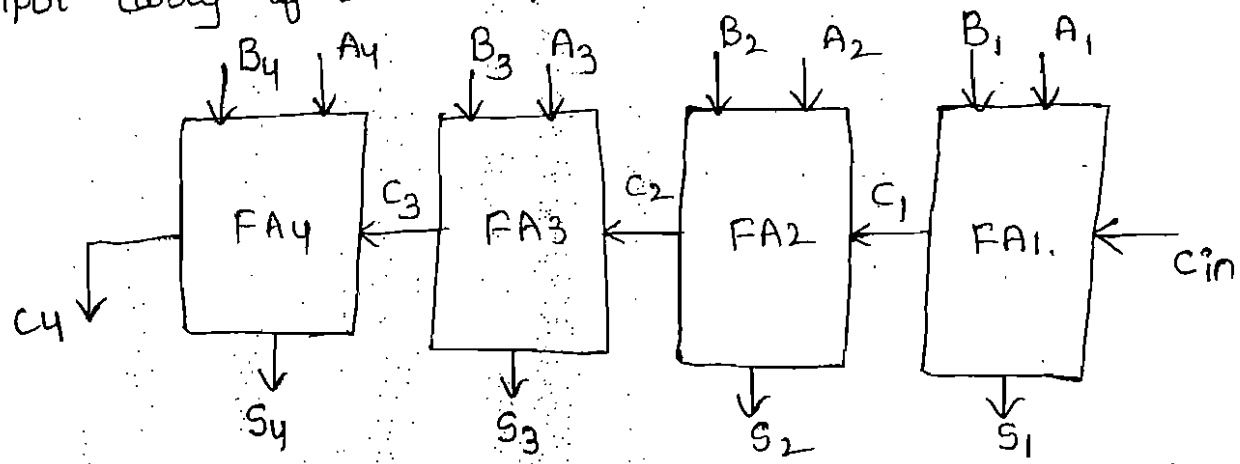


Full Subtractor by using only NOR Gate

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.



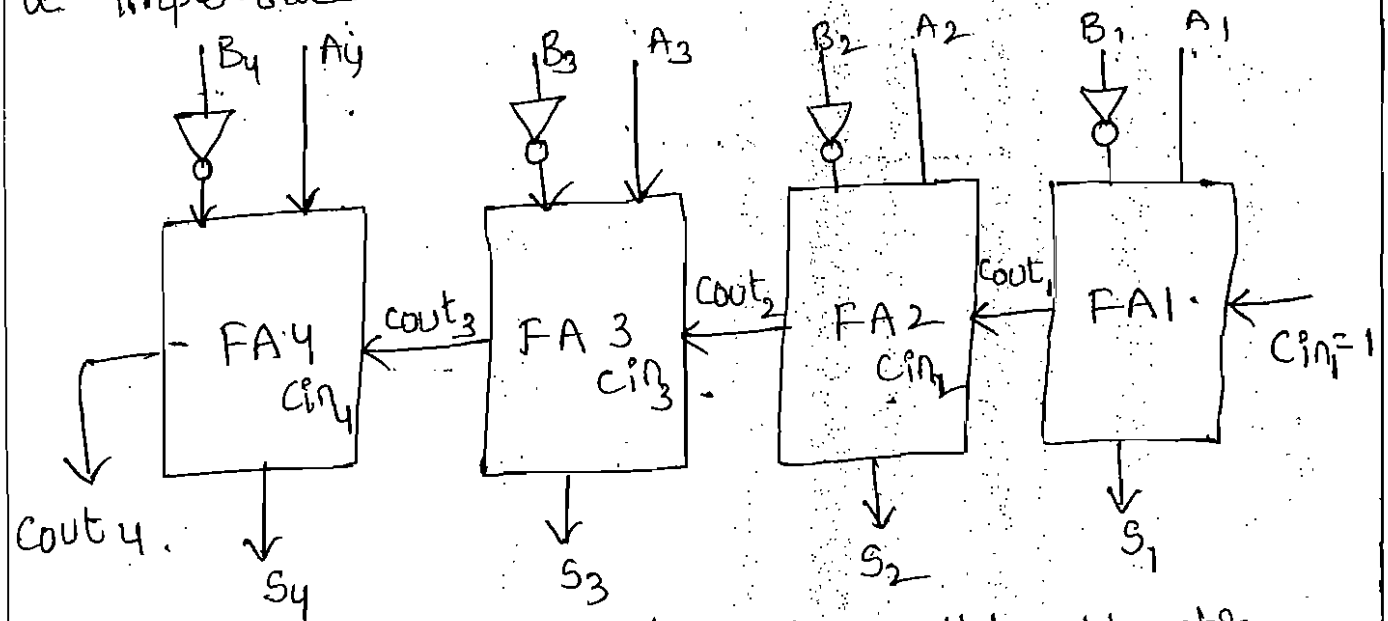
Logic diagram of 4-bit binary parallel adder.

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.

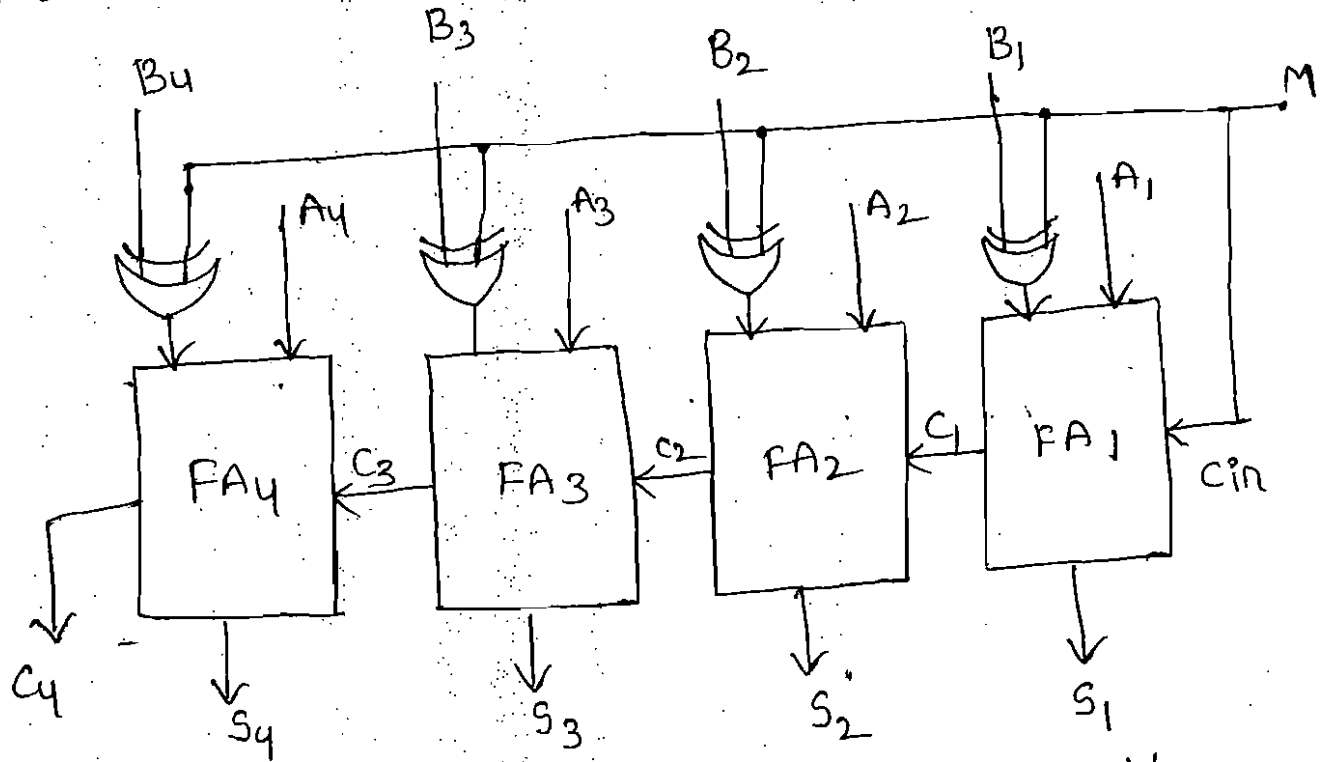
→ when M = 0, the circuit is an adder.

→ 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.

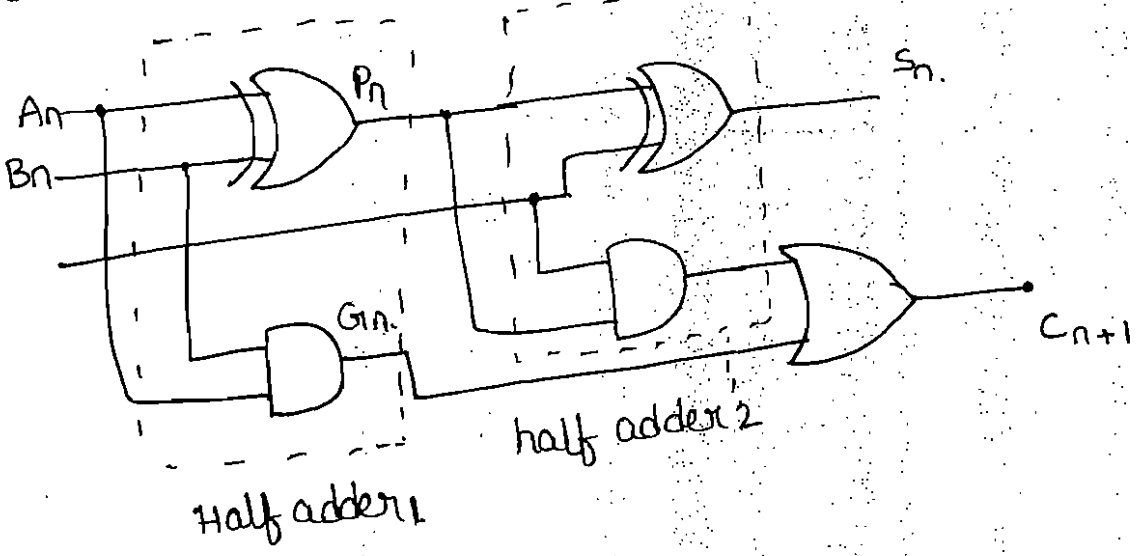


Logic diagram of a 4-bit binary - adder - subtractor.

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.

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

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

$n = 2$

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

$n = 3$

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

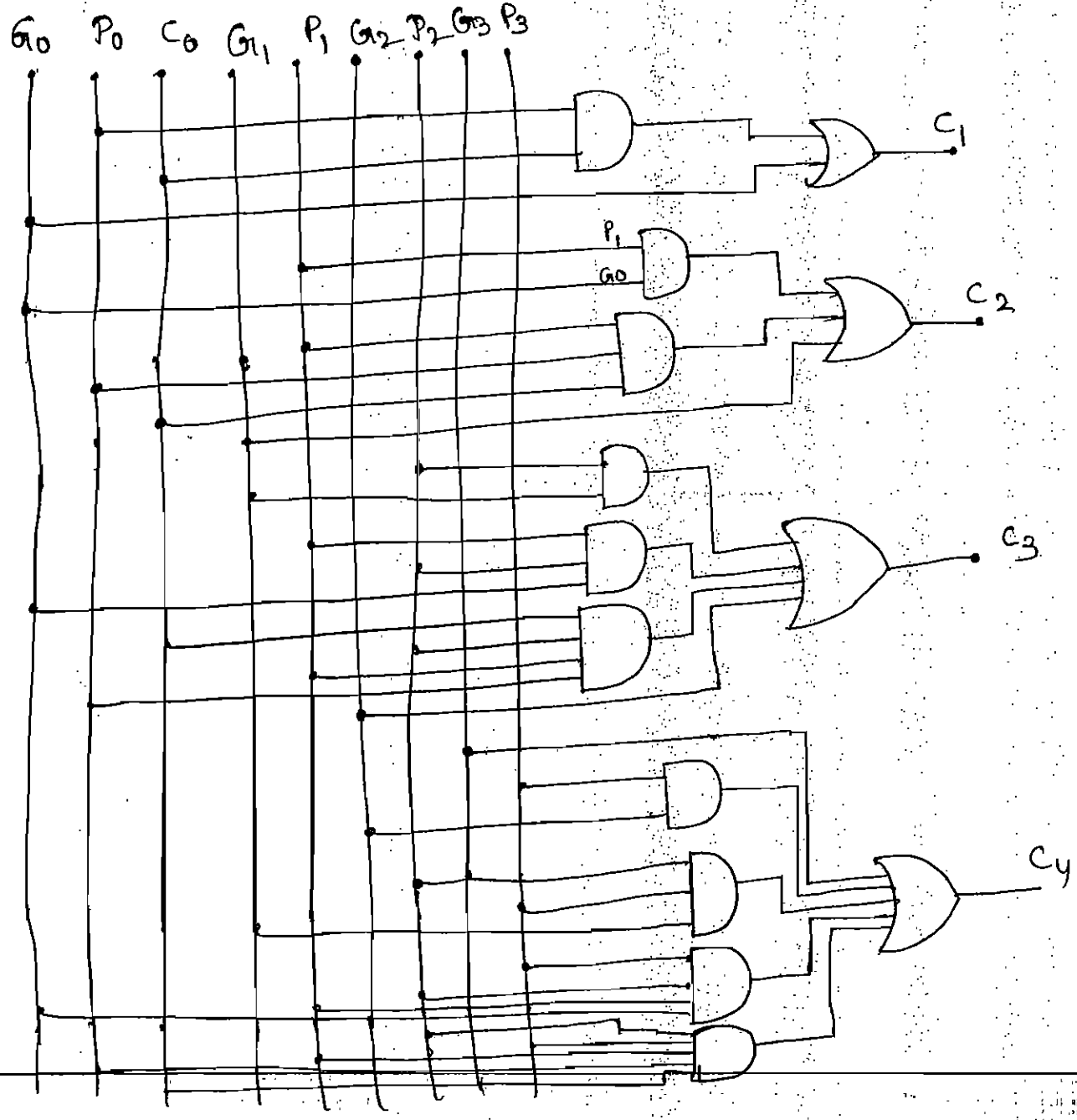
$n = 4$

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

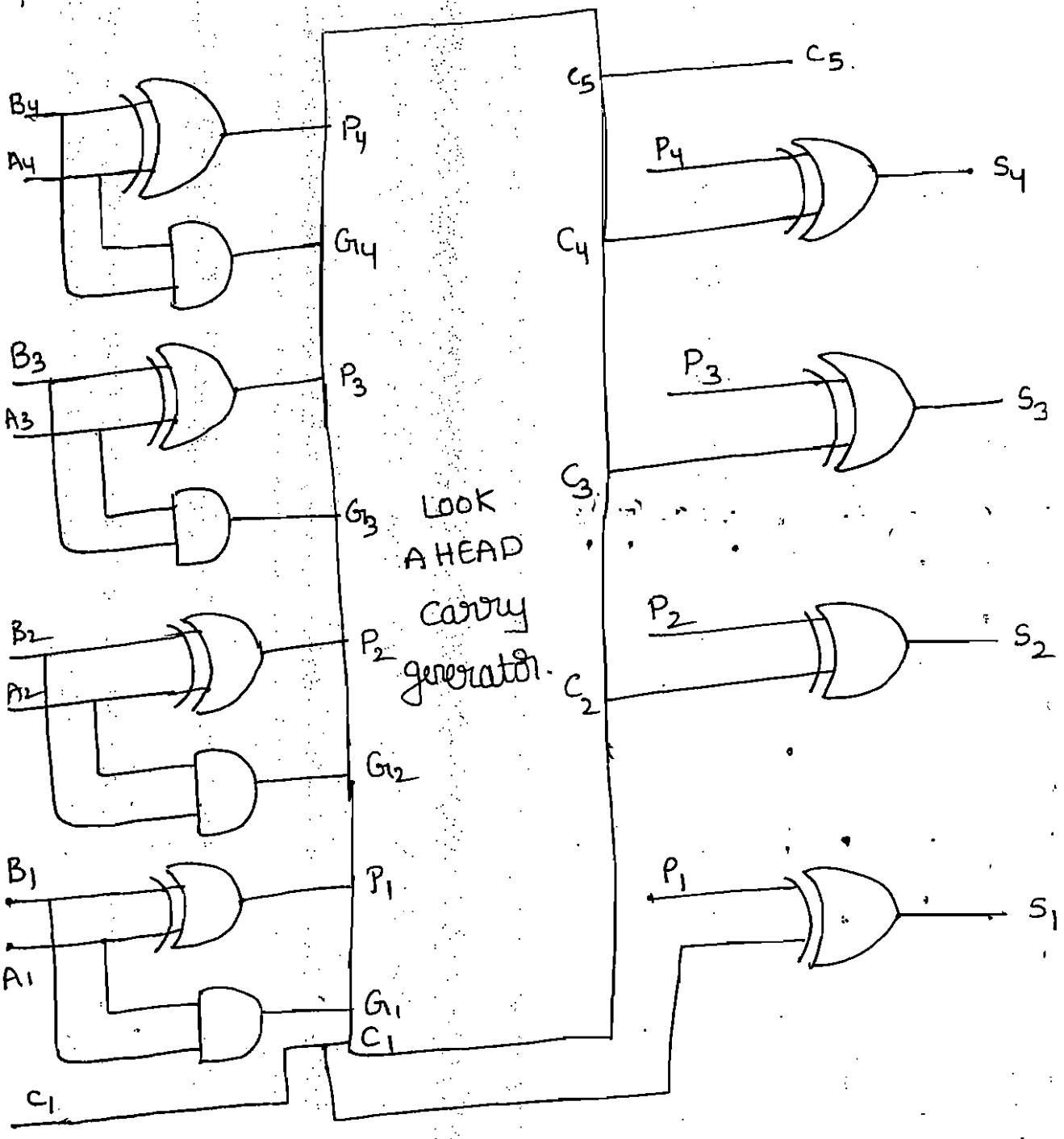
The general expression for n-stages.

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

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

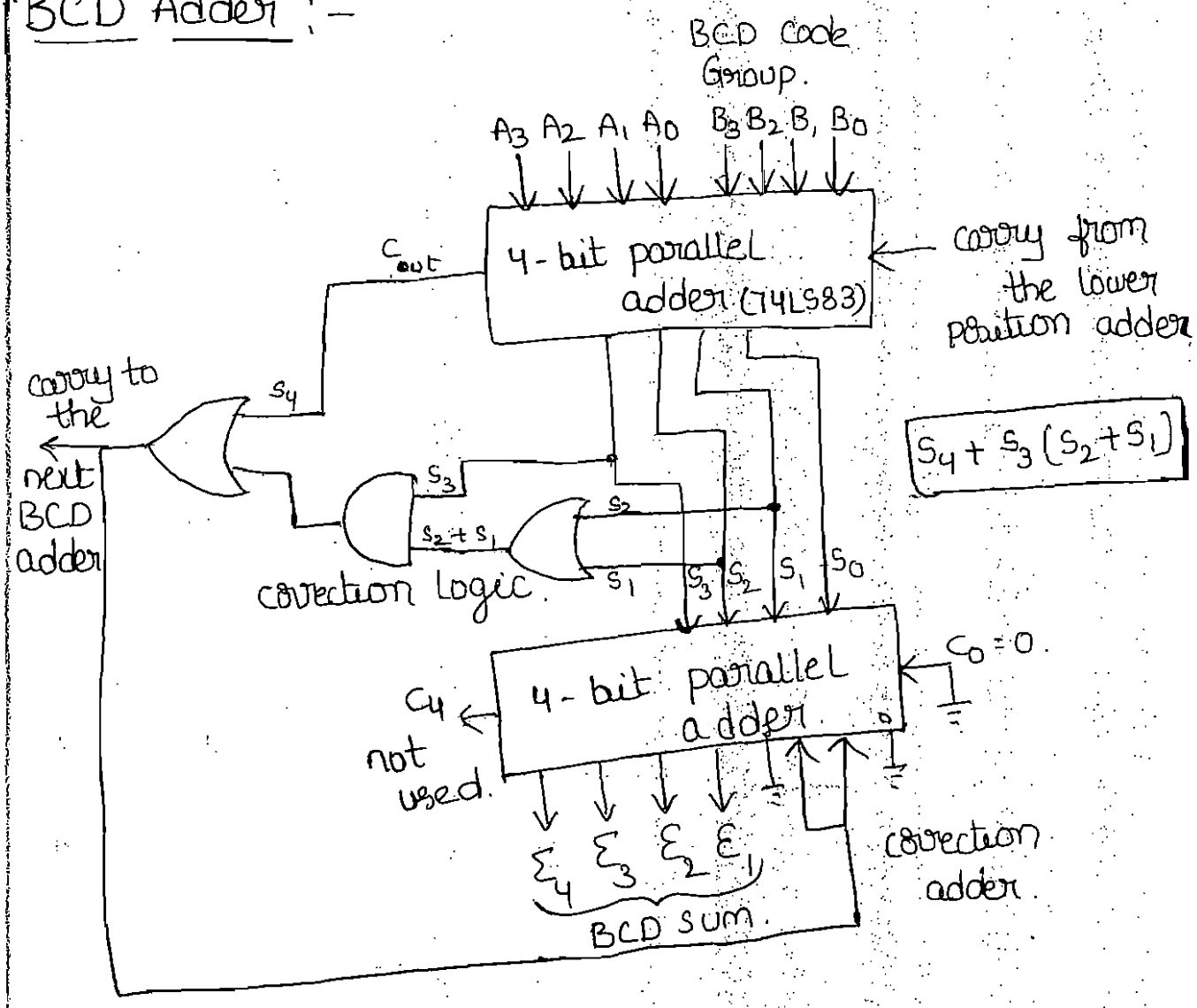


The block diagram of a 4-stage look-ahead-carry parallel adder is shown in the below figure.



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 whenver 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 → whereever $S_4 = 1$,

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

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

whereever $X = 1$, 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.

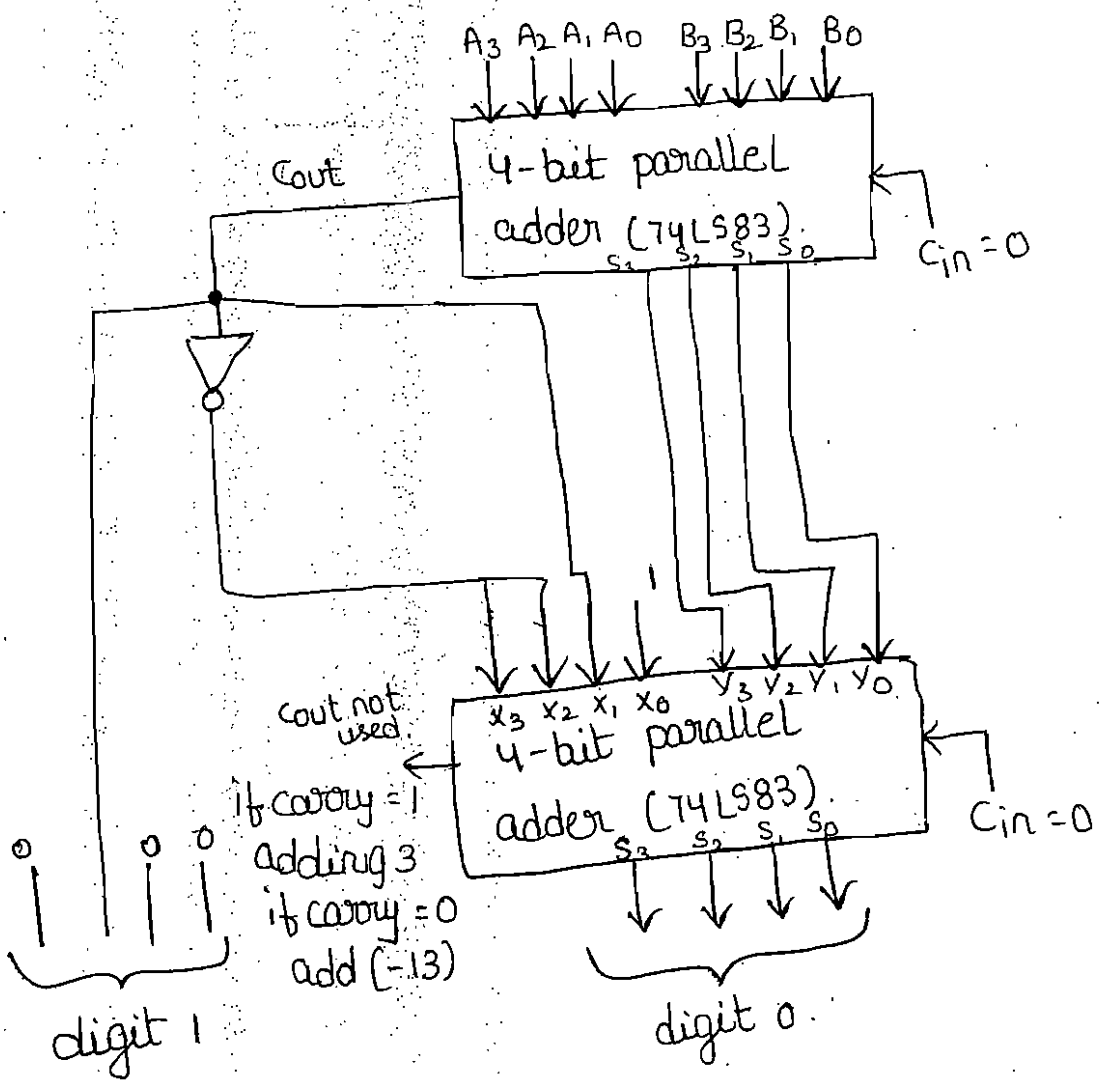
In such cases $\varepsilon_3 \varepsilon_2 \varepsilon_1 \varepsilon_0 = S_3 S_2 S_1 S_0$. Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder.

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.



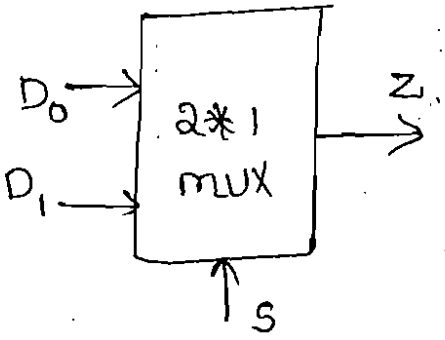
The final Answer in XS-3 form.

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 :-

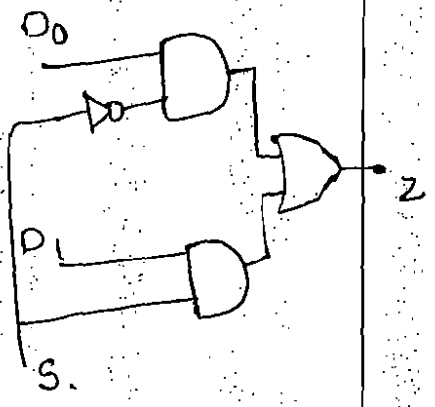
In 2-input multiplexer have 2-inputs they are D_0 and D_1. and one select line S. and output is Z.



Block diagram.

S	Z
0	D_0
1	D_1

Truth table.



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

Logic diagram

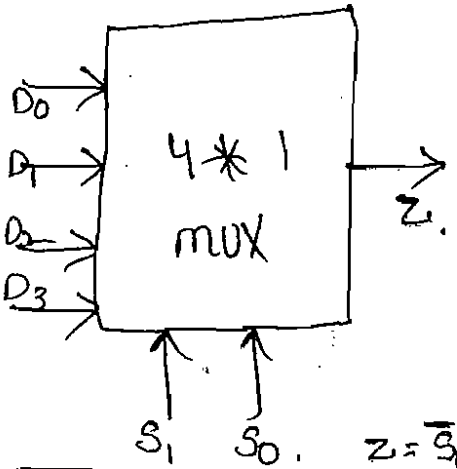
The logic levels applied to the S inputs determines which AND gate is enabled. So that its data input passes through the OR gate to the output.

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.

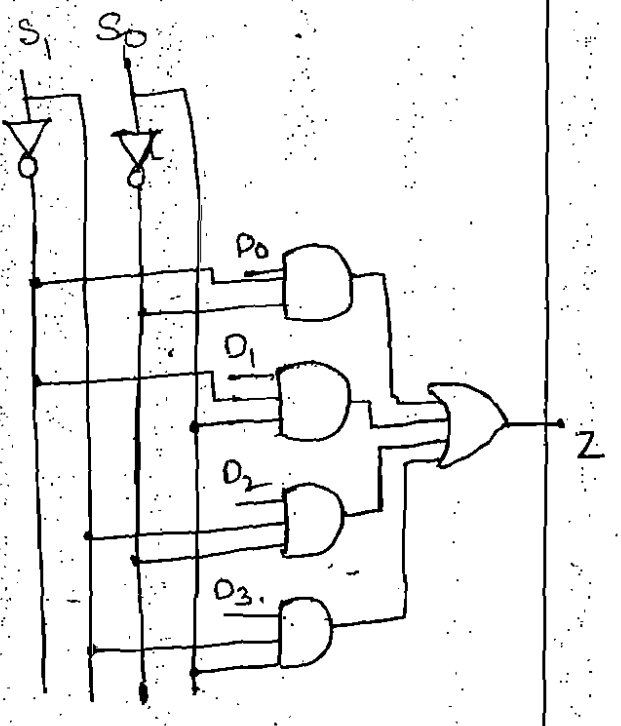
Basic 4-input multiplexer :-

Block diagram



Truth table

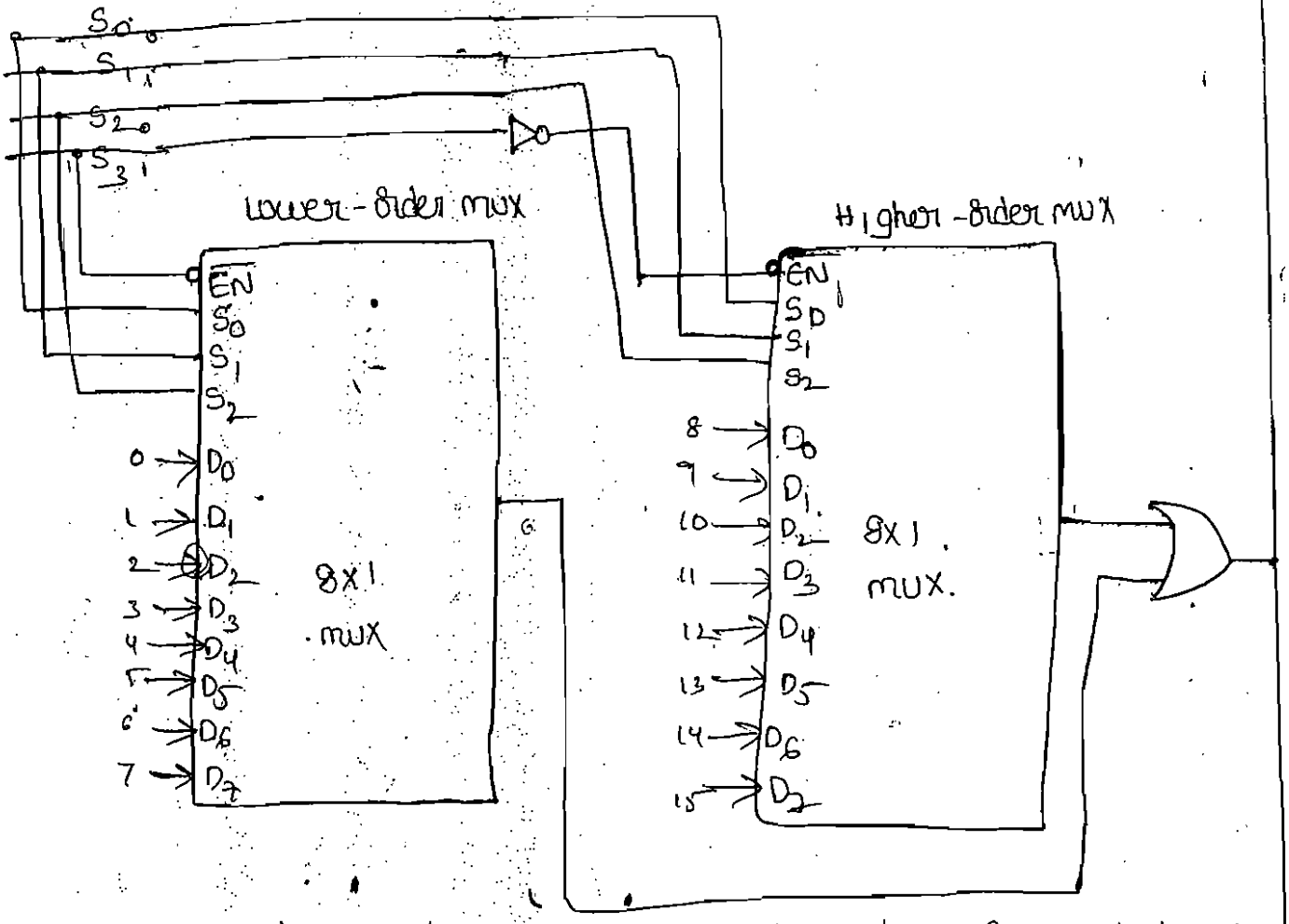
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



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

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

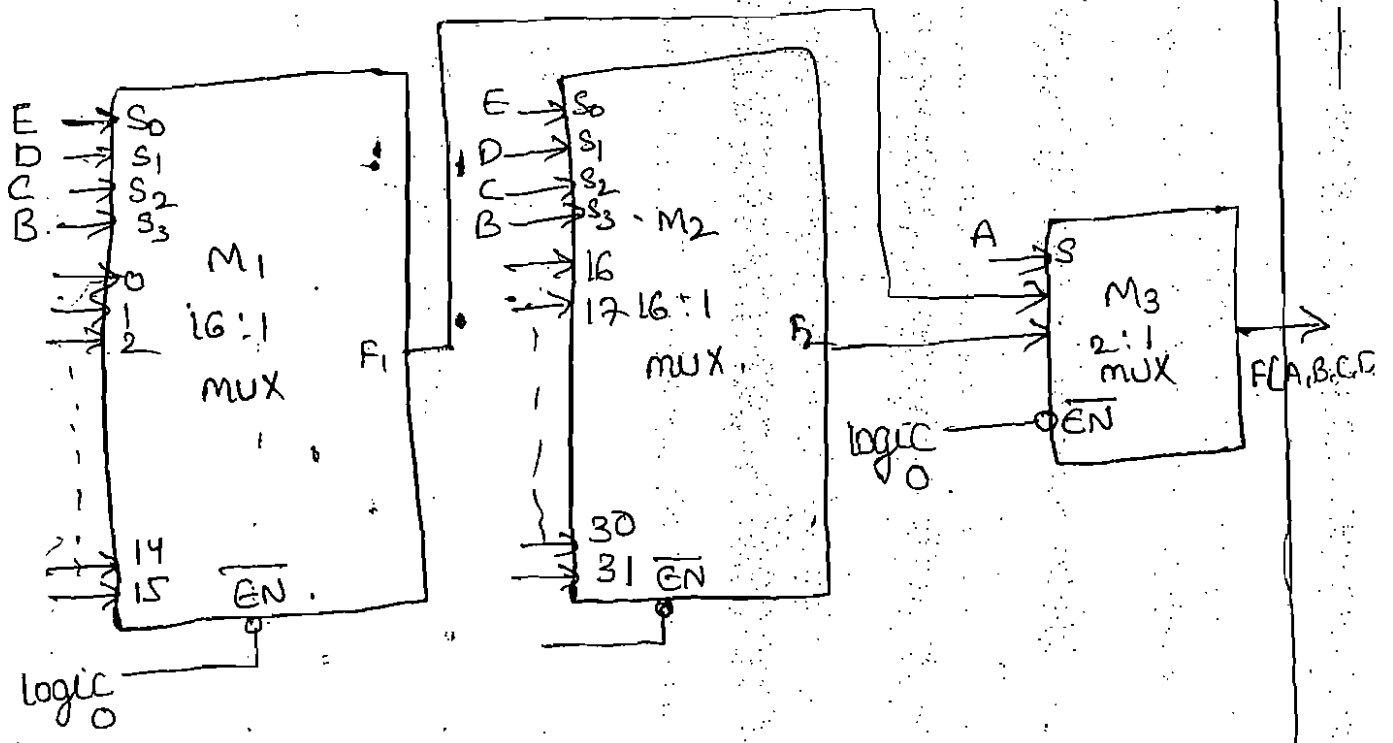
To use two 8-inputs multiplexers to get a 16-inputs multiplexers. one OR gate and one inverter are also required. The four select inputs S_3, S_2, S_1 and S_0 and will select one of the 16 inputs to pass through to X. The S_3 input determines which multiplexer is enabled. when $S_3 = 0$, the left multiplexer 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 16x

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 $F = F_2$.

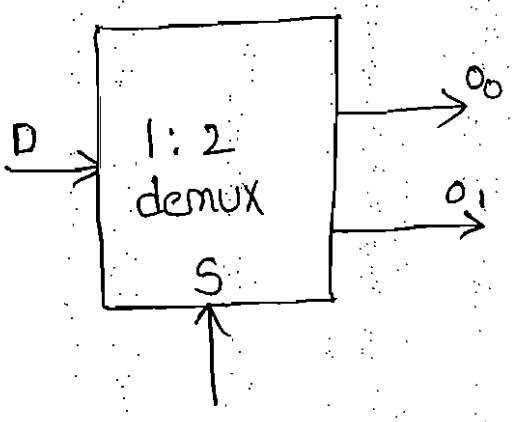


DEMULTIPLEXERS :-

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 :-

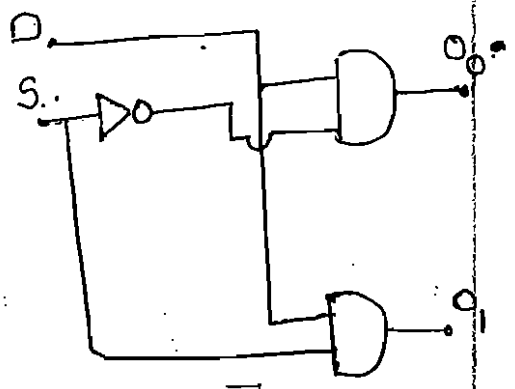
The input data line goes to all of the AND gates. The select line S enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output lines.



Block diagram.

S	O_0	O_1
0	D	0
1	0	D

Truth table.



$O_0 = D\bar{S}$

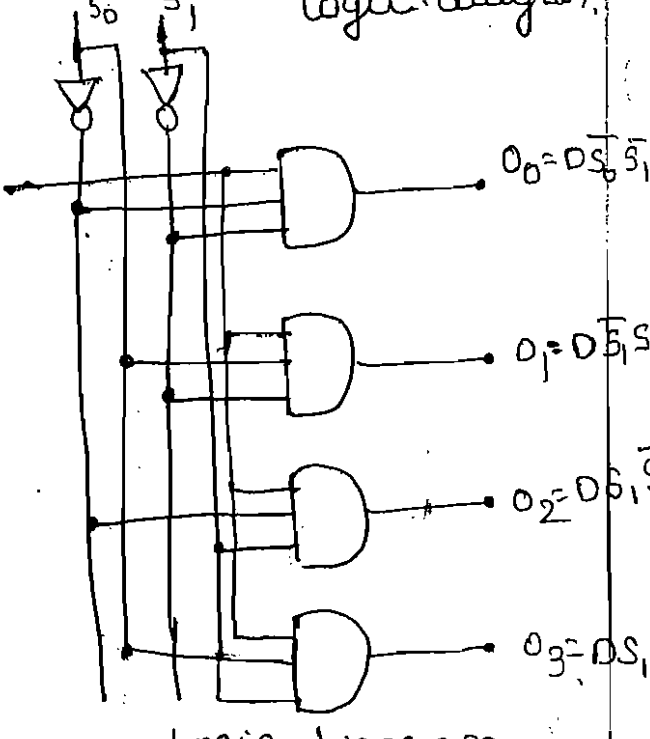
$D_1 = DS$

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.

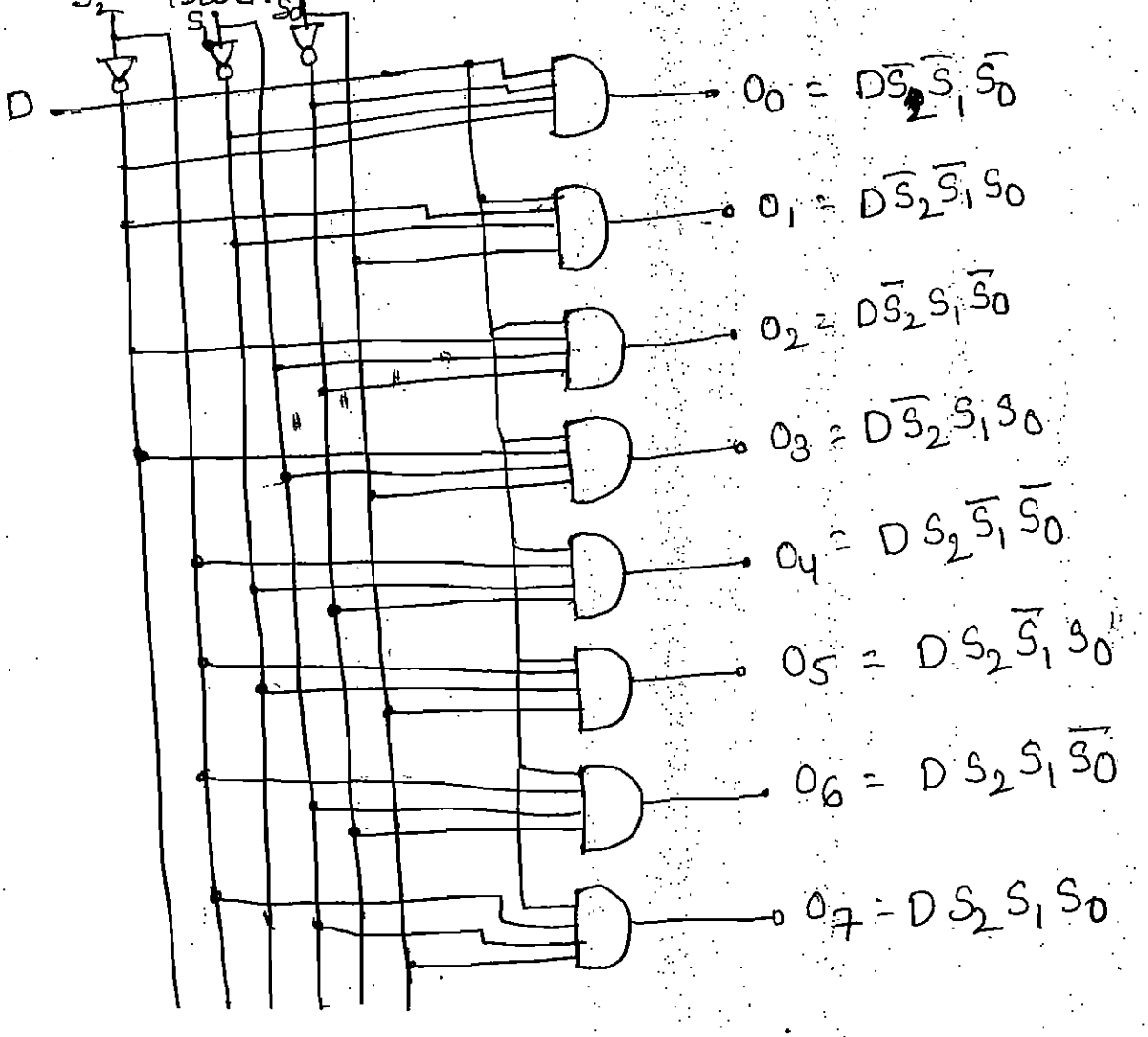


Logic diagram.

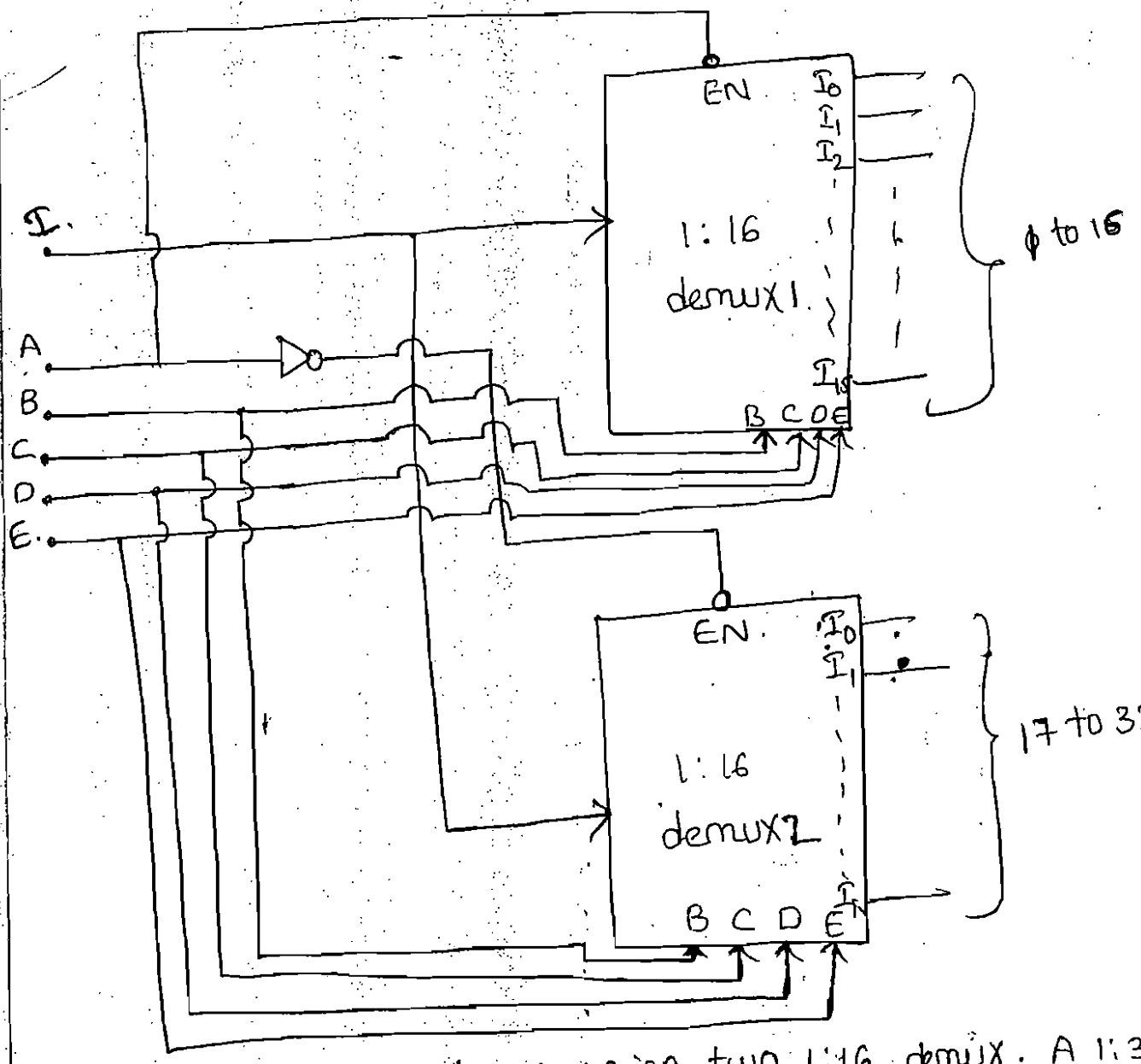
1-line to 8-line demultiplexer:-

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.



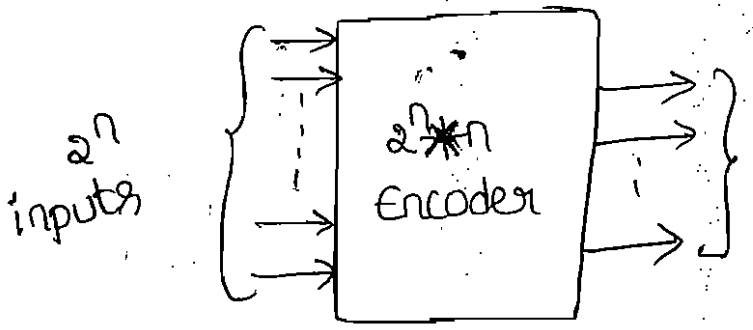
design of 1:32 demux using two 1:16 demux.



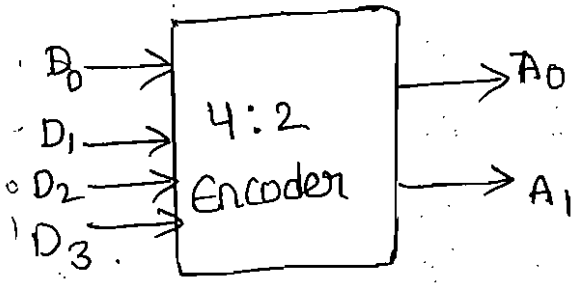
To obtain 1:32 demux using two 1:16 demux. A 1:32 demux has 32 data outputs. So it requires five data select lines. Since 1:16 demux has only four select inputs. the inputs B, C, D, E are connected to the data select lines of both the 1:16 demuxes and the most significant input A is connected to the 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 :-



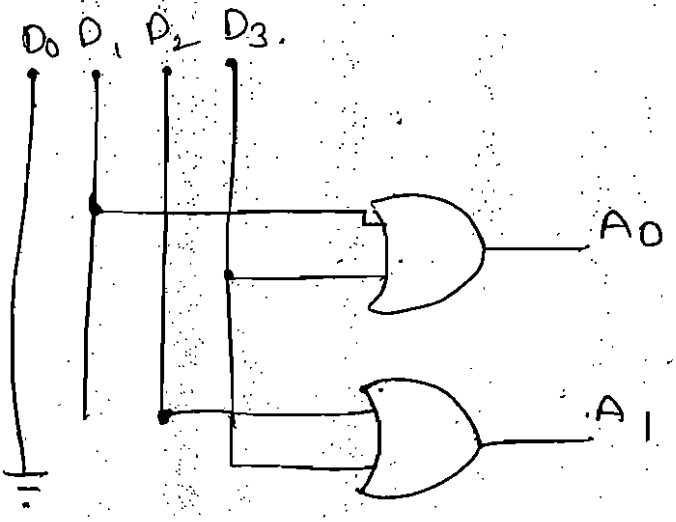
Block diagram.

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

Truth table.

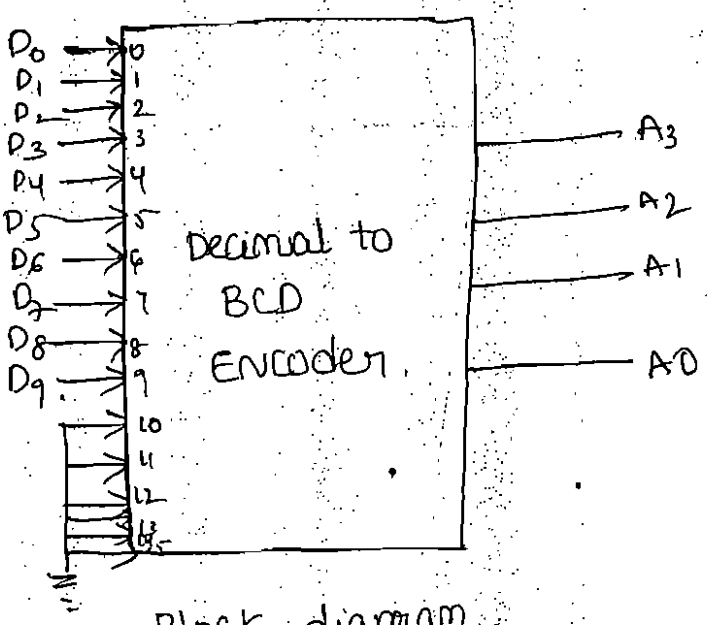
$$A_1 = D_2 + D_3$$

$$A_0 = D_1 + D_3.$$



Decimal to BCD Encoder :-

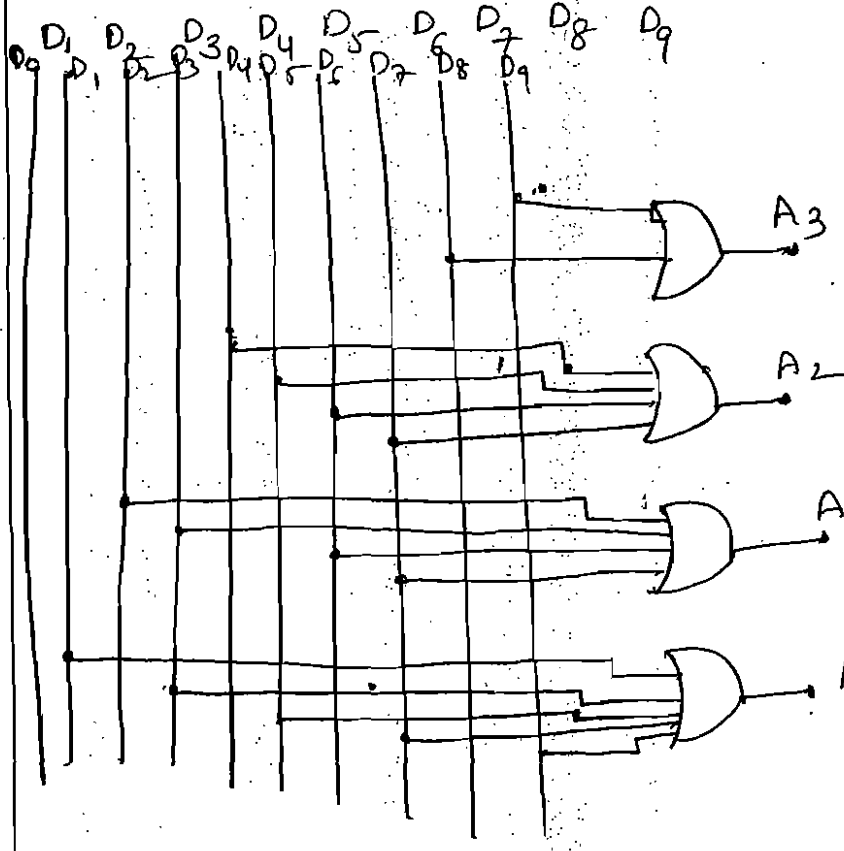
In this type of encoder has 10 inputs - one for each decimal digit, and 4 outputs corresponding to the BCD code.



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.



$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$

$A_1 = D_2 + D_3 + D_6 + D_7$

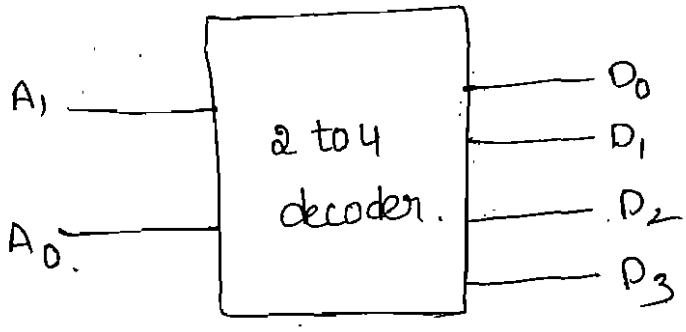
$A_2 = D_4 + D_5 + D_6 + D_7$

$A_3 = D_8 + D_9$.

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.

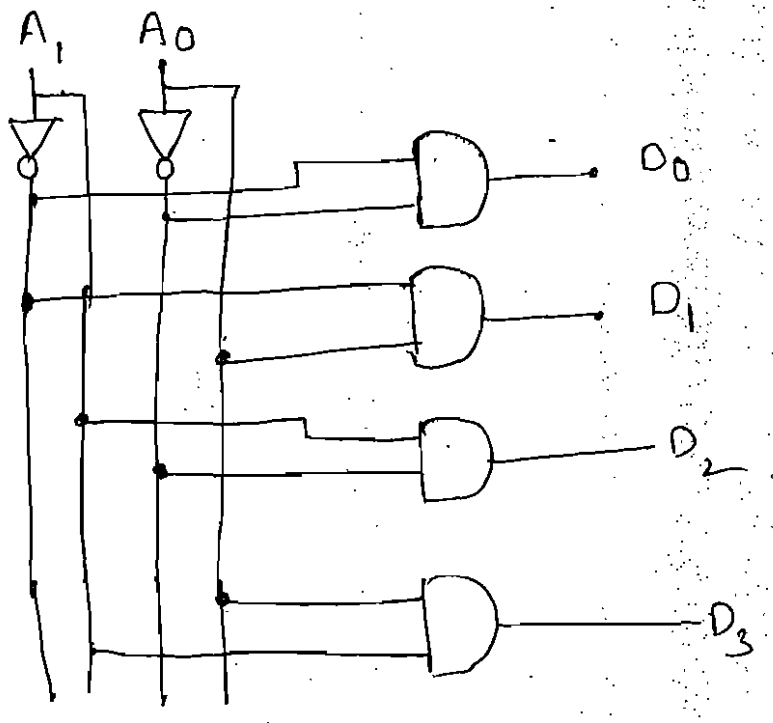
2-4 line decoder :-



Block diagram.

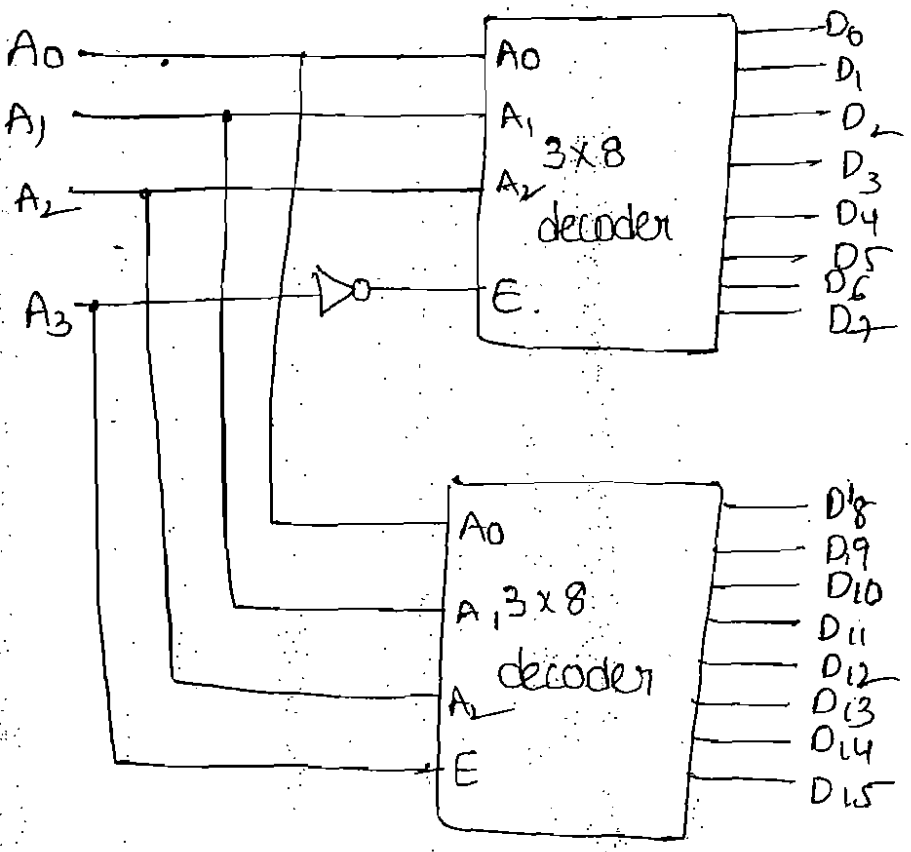
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Truth table.



4-to-16 decoder from two 3-to-8 decoders :-

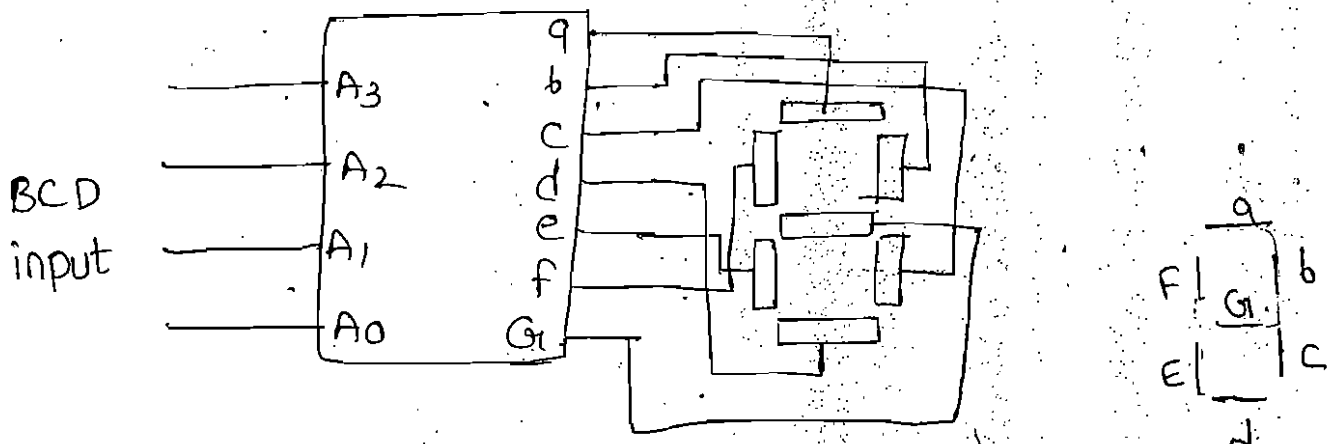
Decoders with enable inputs can be connected together to form a larger decoder. To obtain a 4-to-16 decoder it requires two 3-to-8 decoders. The most significant input bit A_3 is connected through an inverter to $\bar{E}$ on the upper decoder and directly to E on the lower decoder. Thus A_3 is low, the 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

$a = \Sigma m(0,2,3,5,6,7,8,9) + d(10,11,12,13,14,15)$.

$b = \Sigma m(0,1,2,3,4,7,8,9) + d(10,11,12,13,14,15)$

$c = \Sigma m(0,1,3,4,5,6,7,8,9) + d(10,11,12,13,14,15)$.

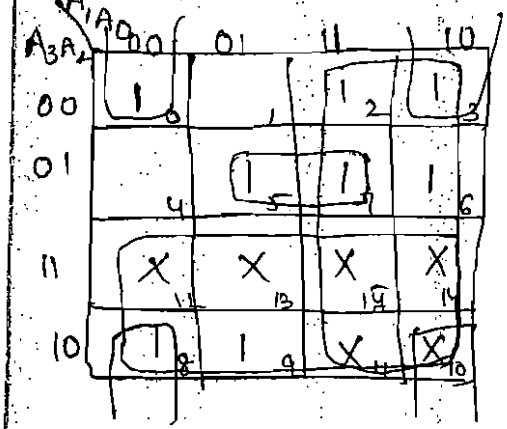
$d = \Sigma m(0,2,3,5,6,8,9) + d(10,11,12,13,14,15)$.

$e = \Sigma m(0,2,6,8) + d(10,11,12,13,14,15)$.

$f = \Sigma m(0,4,5,6,8,9) + d(10,11,12,13,14,15)$.

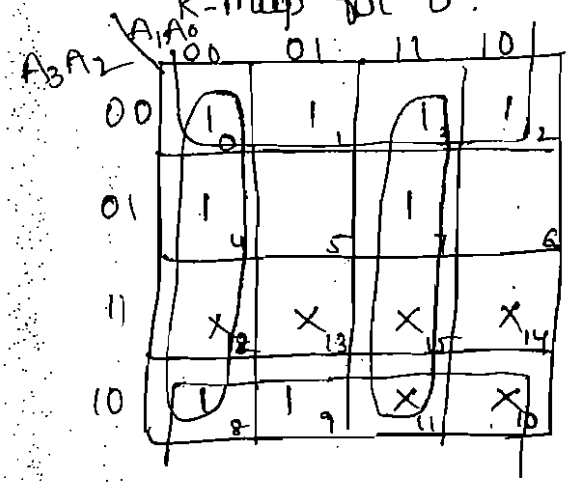
$g = \Sigma m(2,3,4,5,6,8,9) + d(10,11,12,13,14,15)$.

K-map for a.

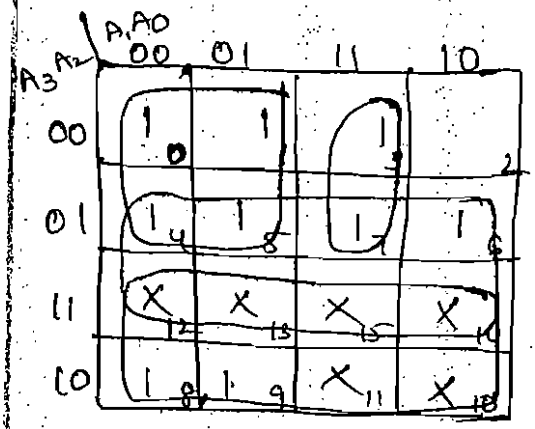


$a = A_1 + A_3 + \overline{A_2}\,\overline{A_0} + \overline{A_3}A_2A_0$

K-map for b.

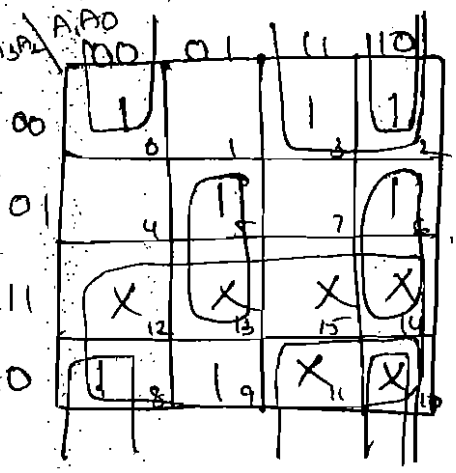


$b = \overline{A_2} + A_1\overline{A_0} + A_1A_0$



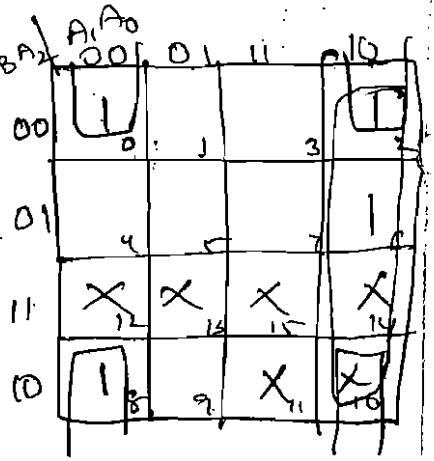
K-map for c

$c = A_2 + A_3 + \overline{A_3}\,\overline{A_1} + \overline{A_3}A_1A_0$



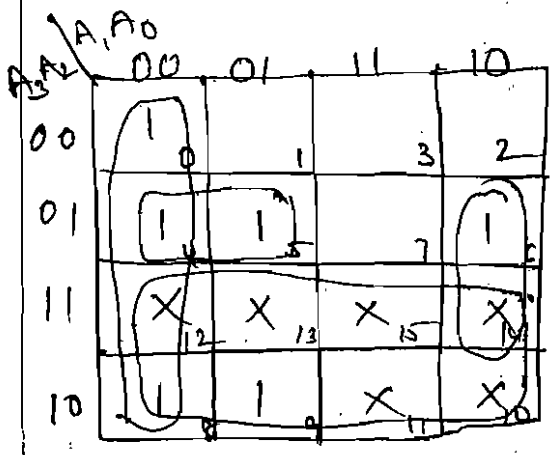
K-map for d

$d = A_3 + \overline{A_2}A_1 + A_2\overline{A_1}A_0 + A_2A_1\overline{A_0} + \overline{A_2}\,\overline{A_0}$

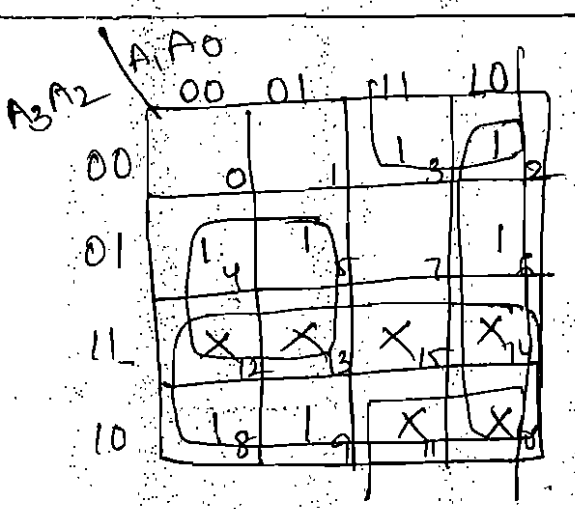


K-map for e

$e = A_1\overline{A_0} + \overline{A_2}\,\overline{A_0}$



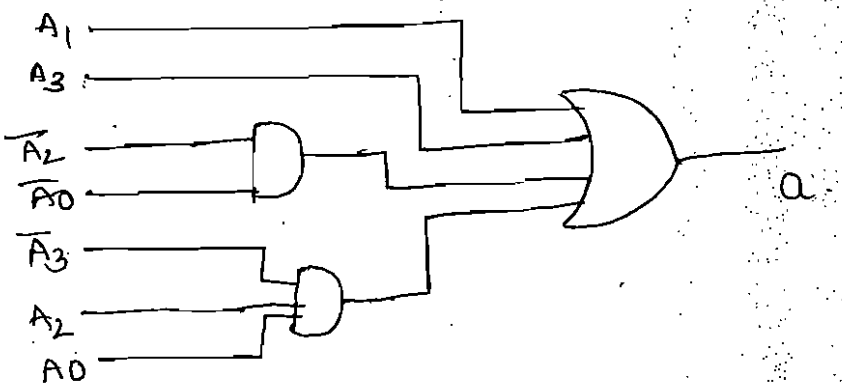
K-map for f.

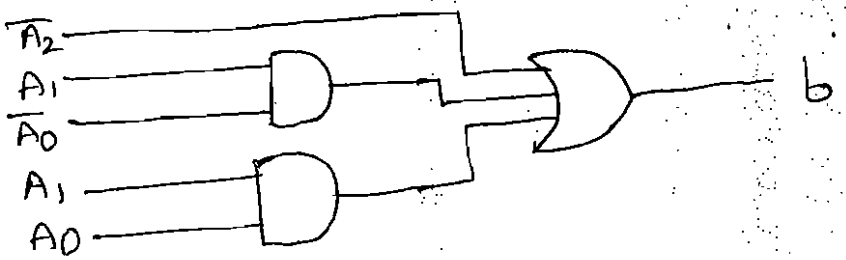


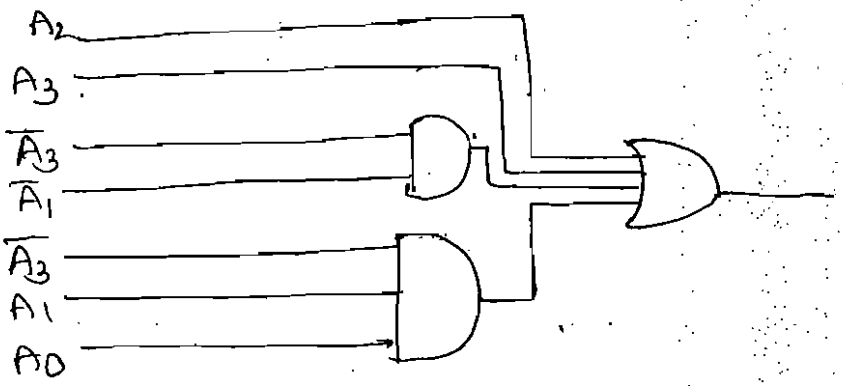
K-map for G1.

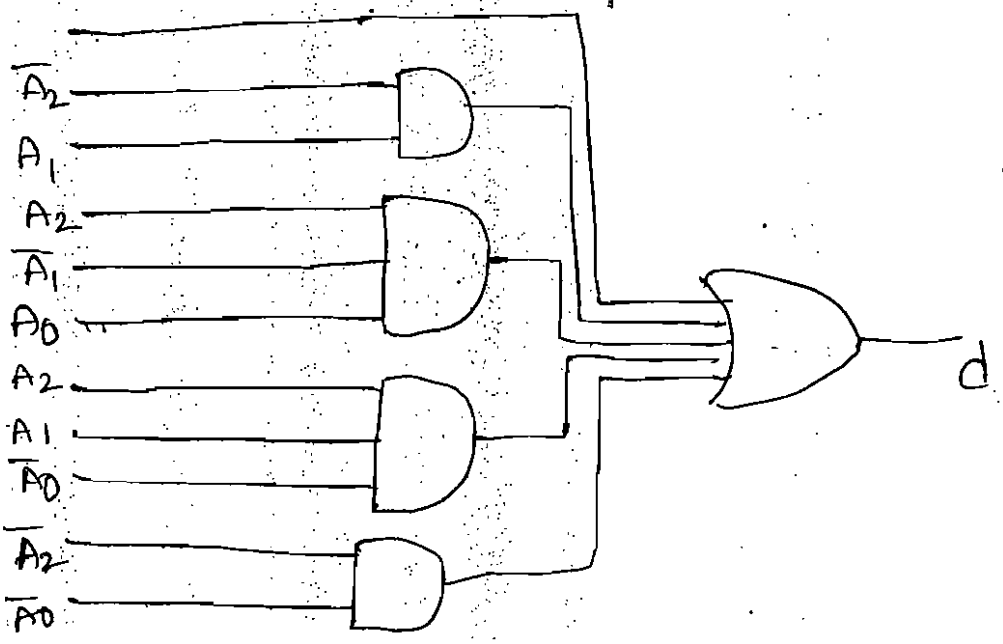
$f = \bar{A}_1\bar{A}_0 + A_3 + \bar{A}_3A_2\bar{A}_1 + A_2A_1\bar{A}_0$

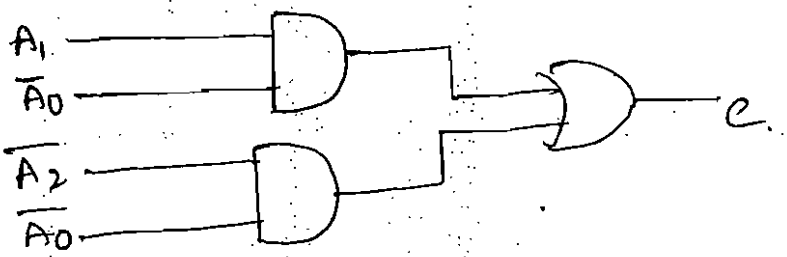
$G_1 = A_3 + A_1\bar{A}_0 + A_2\bar{A}_1 + \bar{A}_2A_1$

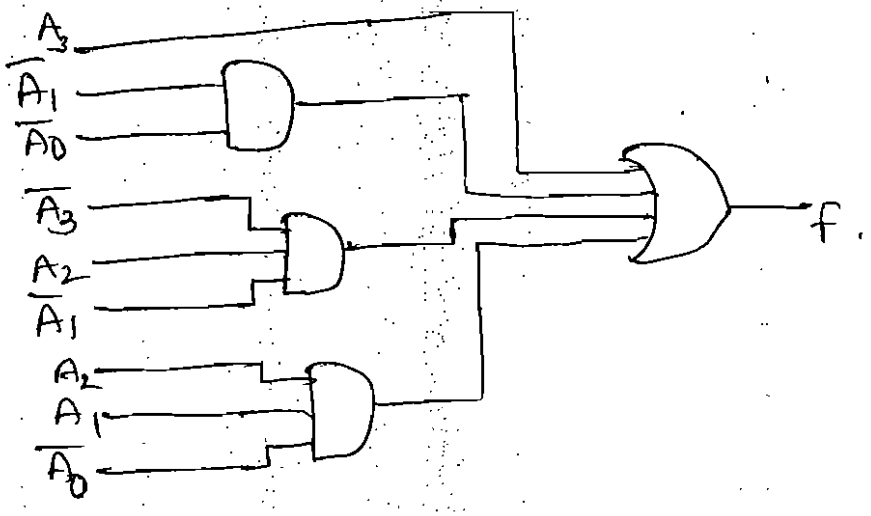


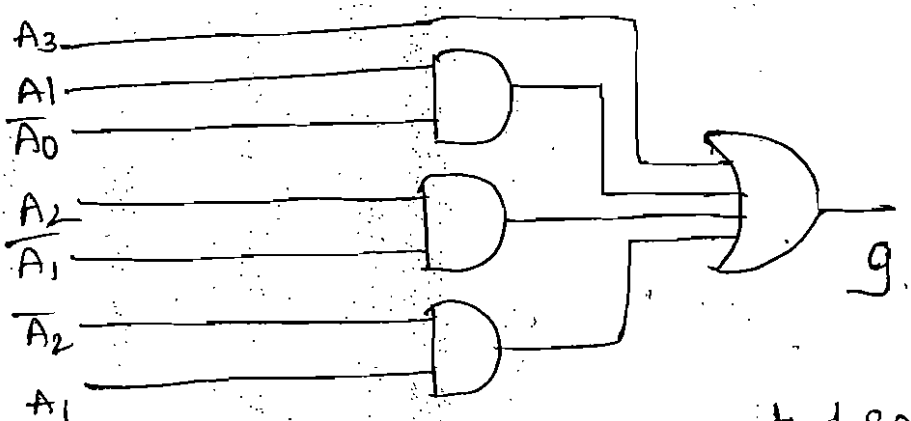








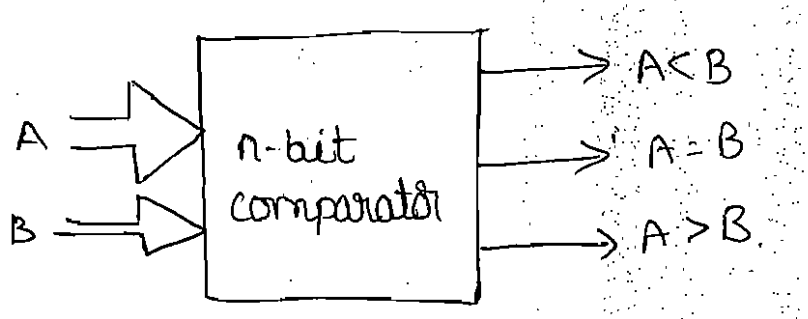




diagrams for seven segment display.

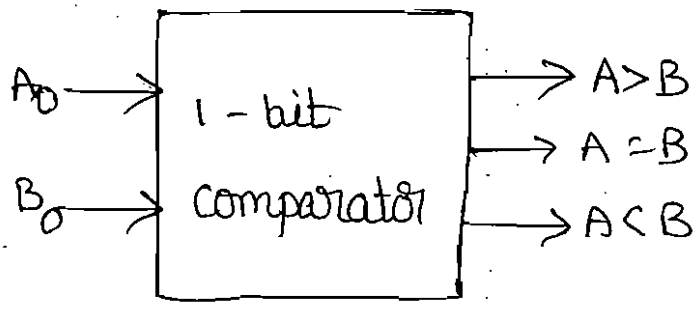
Comparator :-

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 comparision, 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

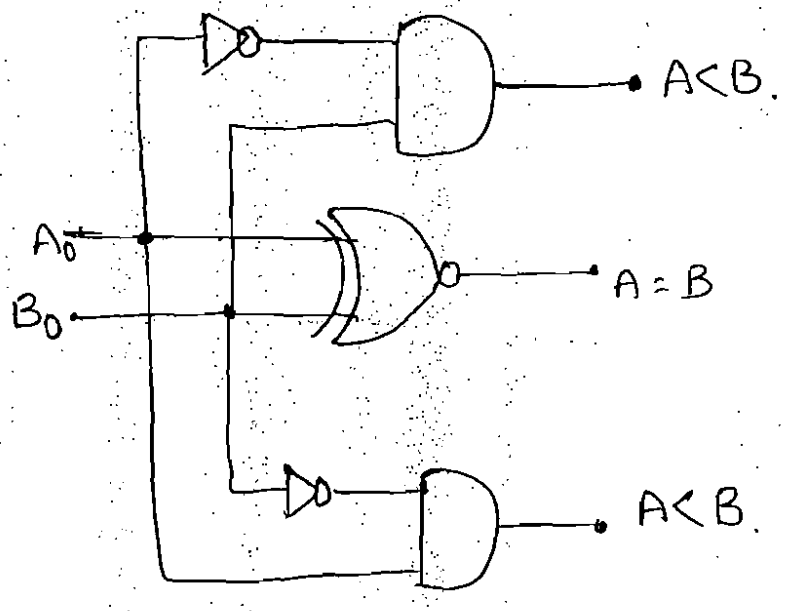
Truth table

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 $A > B$.

$A>B : A_1 \overline{B_1} + (A_1 \odot B_1) A_0 \overline{B_0}$

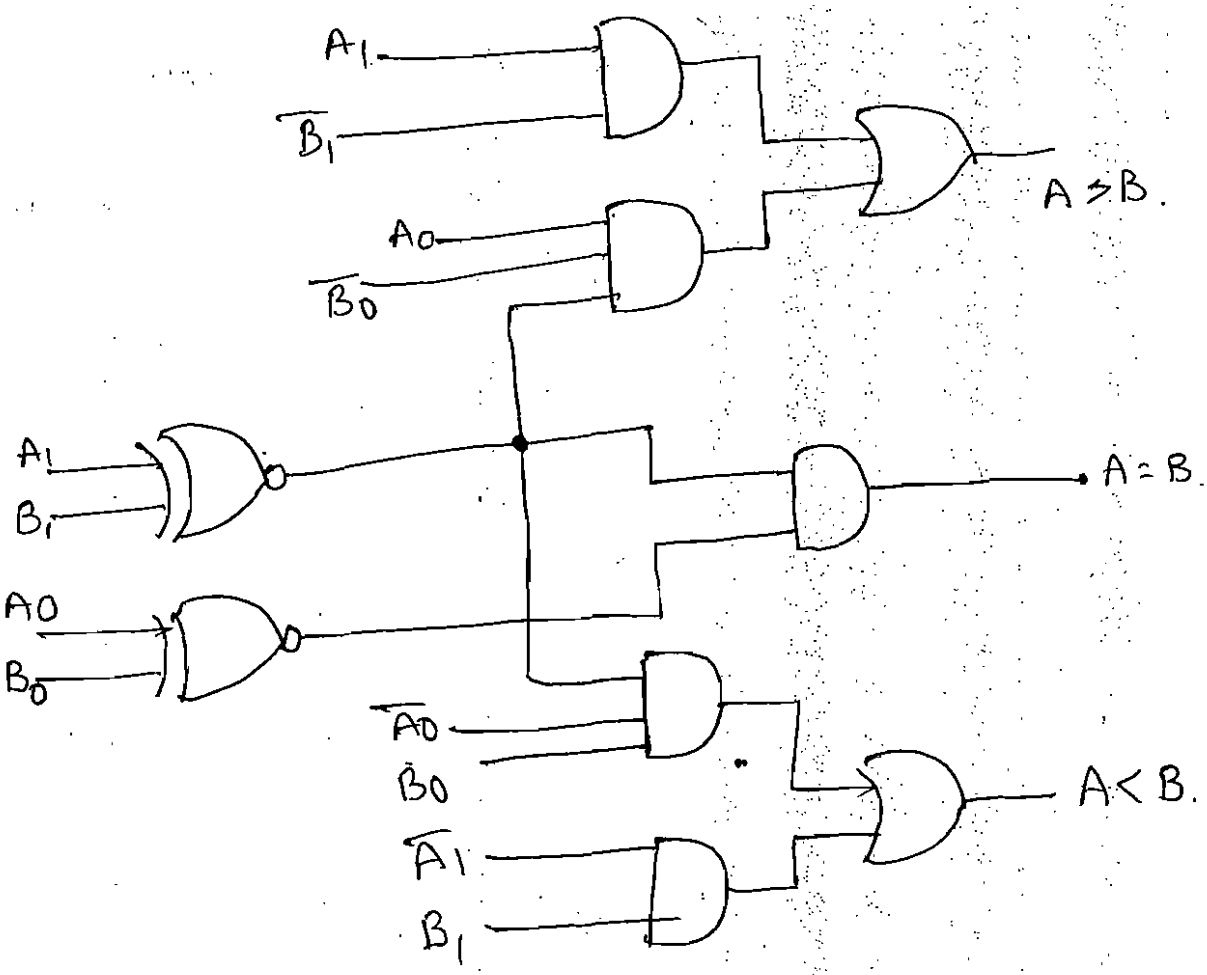
→ if $B_1 = 1$ and $A_1 = 0$ then $A < B$ or

→ 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

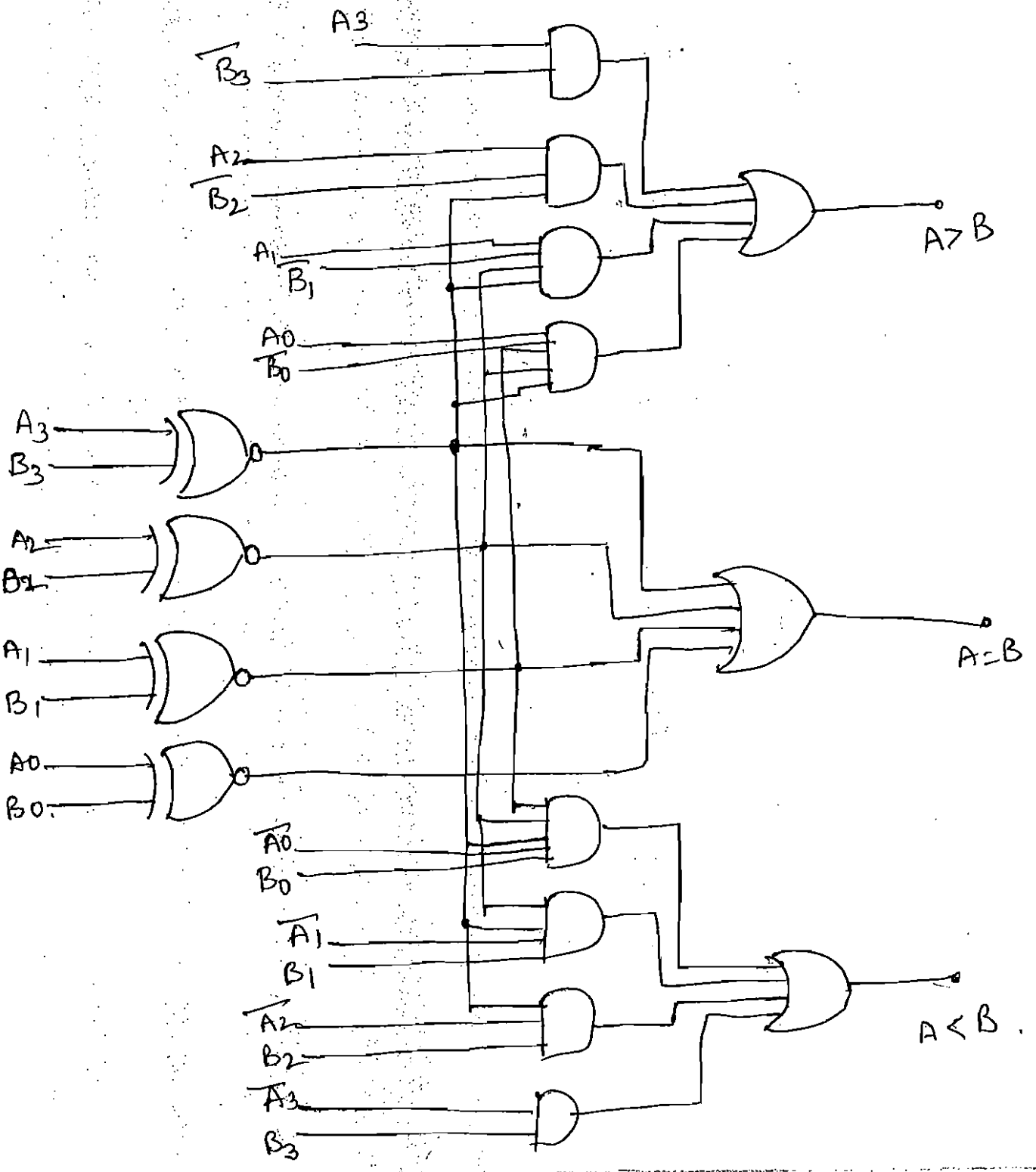
→ if A_3 and B_3 are equal, A_2 and B_2 are equal, and if A_1 = 1 and $B_1 = 0$, or

→ 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

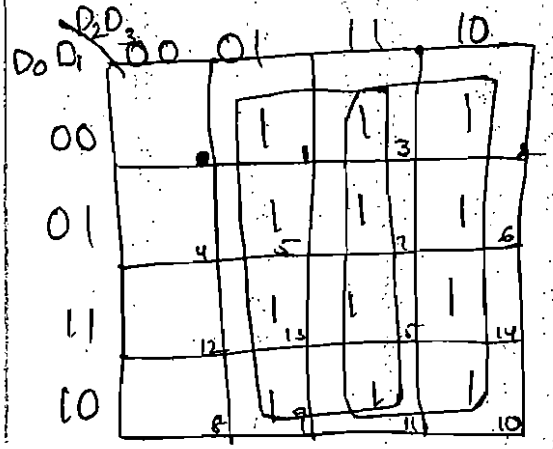
$V = D_0 + D_1 + D_2 + D_3$.

According to the truth table. the outputs A and B are.

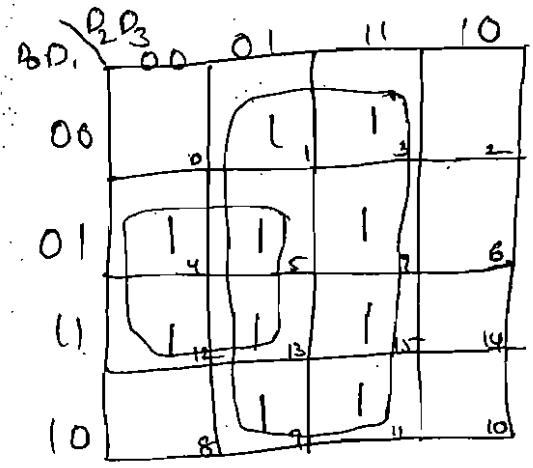
$$A = \Sigma m(1,2,3,5,6,7,9,10,11,13,14,15)$$

$$B = \Sigma m(1,3,4,5,7,9,11,12,13,15)$$

K-map for A

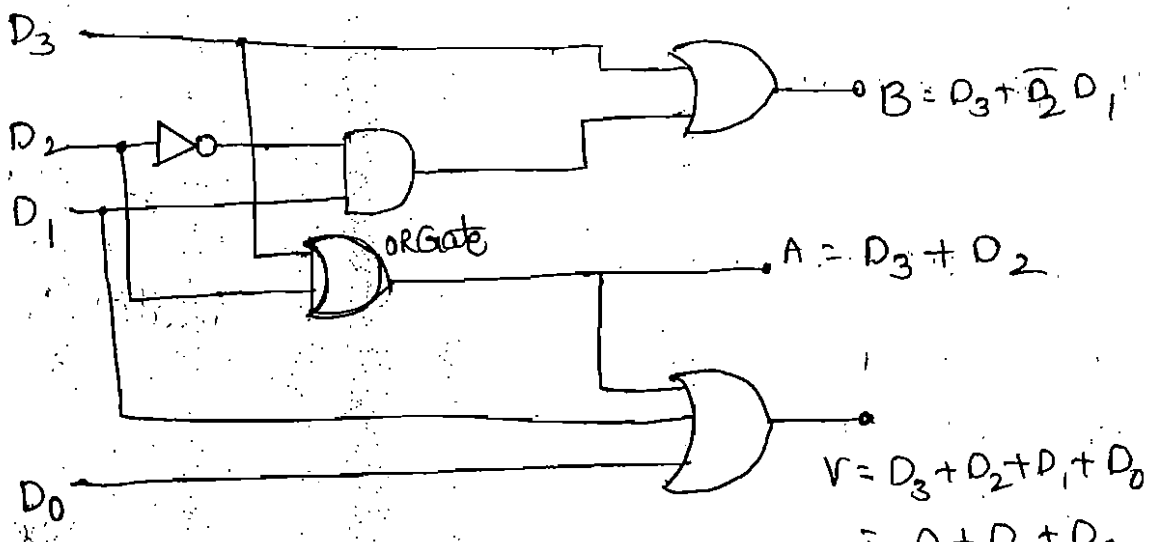


K-map for B.



$A = D_3 + D_2$

$B = D_3 + \overline{D_2} D_1$



logic diagram for

4-bit priority Encoder.

* Introduction of PLD's :-

→ In Digital IC's to Perform basic Digital Operations and the other functions, such as adders, comparators, multiplexers demultiplexers, code converter's etc. These are known as fixed function Ic's. So the manufacturers Produce larger quantity. So There is another two approaches such as ASICS (Application Specific integrated circuits) and PLD's.

* Following table comparision between these 3 design's.

Parameter :-	Fixed function approach	ASIC Approach	PLD Approach
① Cost	Low	High	Low
② Spacer required	Large	minimumum	Less
③ Power required	Large	Less	Less.
④ Reliability	Low	Highest	high.
⑤ CKt testing	Easy	Testing method's increasing, cost is high	Easy
⑥ flexibility	less	No	more.
⑦ Design Security	No Security	High	High
⑧ Design time.	less	more	Less.

* From the above table we can noted that PLD Approach is the best one when compared to other approches.

→ PLD's is an Ic that contains large no. of gates, flip-flops and registers that are interconnected on chip. This IC is said to be Programmable because the specific function IC is determined by

Interconnecting required components.

* There are 3-types of Programmable devices
  - → Read only memory (ROM)
  - → Programmable logic Array (PLA)
  - → Programmable Array Logic (PAL)

* Rom:- It is a memory to design to hold data that is either permanent or will n't change frequenty. only we read the data, No more modifications takes place. During the operation no data is written in to this memory. writing the data in to. memory is called Programming.

Types of Roms:- 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)

① masked memory Rom:-

1. Cannot be reProgrammed.
2. Non-volatile, retain the data Power is OFF
3. Cheaper than Programmable devices.
4. Useful for fixed Programme instructions

② PRom:-

- → Programmed by blowing built-in fuses.
- → Cannot be reProgrammed
- → NON volatile
- → Useful for

③ EPRom:-

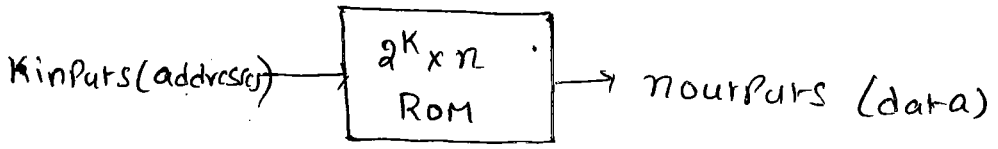
- → Erasable, Programmable Rom
- → Programmed by storing change on insulated gates.
- → Erasable with Ultraviolet light
- → Non volatile

④ EEPRom:-

- → Programmed by using storing Changes on insulated gates.
- → Non volatile in Nature.

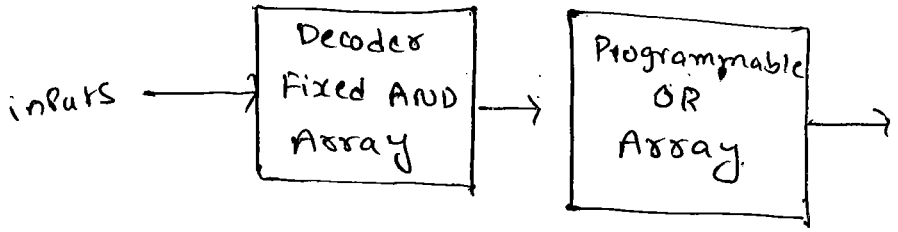
① PROGRAMMABLE ROM (PROM) :-

* It has K inputs and n outputs. The input provides the address for memory and the outputs give the data bits of the stored word which is selected by the address.

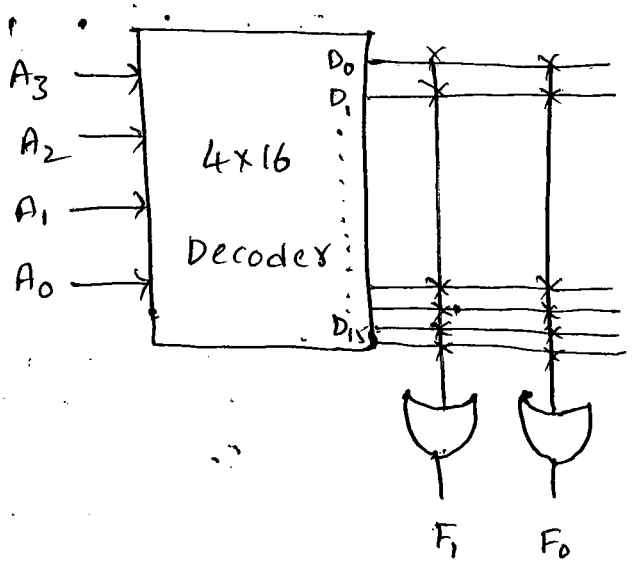


Rom Block diagram

* A Programmable Read only memory has a fixed AND Array with decoder. and 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.



Ex:- Construct 16x2 Rom using Decoder with Fixed AND Array and Programmed OR Array ?



In this 16x2 Rom, we have 16 input address lines corresponding to the inputs $A_3 A_2 A_1 A_0$ and the O/P of these Address lines hold the data inputs and the O/Ps are F_1, F_0, then total no. of connections for 16x2 Rom has 16x2 = 32 Connections.

Implement full adder using PROM:-

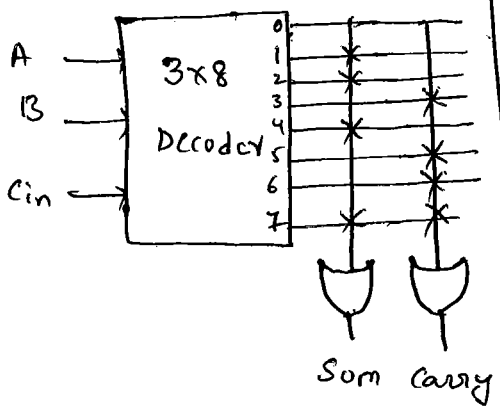
* The no. of inputs of full adder is 3. Therefore we need 3x8 decoder, the no. of o/ps of full adder 2 which is Sum and carry.. So we use the 8x2 PROM to implement this.

Truth table

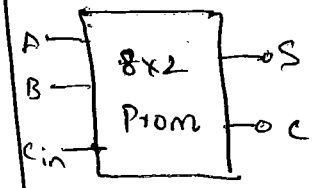
A	B	C_{in}	S	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$Sum = \Sigma m(1,2,4,7)$; $Carry = \Sigma m(3,5,6,7)$

logic Diagram:-



Block diagram of 8x2 Prom is



* Implement BCD to Excess-3 using Rom:-

Step (1):- Draw the table for the BCD and given Excess-3 table

BCD

B_3	B_2	B_1	B_0	
0	0	0	0	→0
0	0	0	1	→1
0	0	1	0	→2
0	0	1	1	→3
0	1	0	0	→4
0	1	0	1	→5
0	1	1	0	→6
0	1	1	1	→7
1	0	0	0	→8
1	0	0	1	→9

Excess-3

E_3	E_2	E_1	E_0	
0	0	1	1	→3
0	1	0	0	→4
0	1	0	1	→5
0	1	1	0	→6
0	1	1	1	→7
1	0	0	0	→8
1	0	0	1	→9
1	0	1	0	→10
1	0	1	1	→11
1	1	0	0	→12

Step (2):-

$E_0(B_3B_2B_1B_0) = \Sigma m(0,2,4,6,8)$

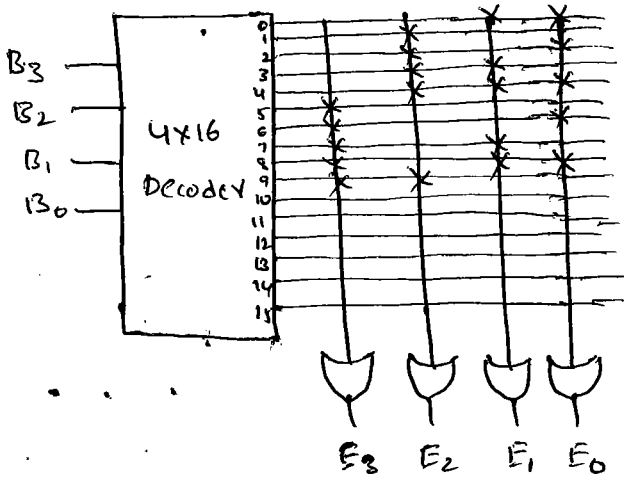
$E_1(B_3B_2B_1B_0) = \Sigma m(0,3,4,7,8)$

$E_2(B_3B_2B_1B_0) = \Sigma m(1,2,3,4,9)$

$E_3(B_3B_2B_1B_0) = \Sigma m(5,6,7,8,9)$

Step (3):-

16x4 ROM



Note:- 16x4 Rom contains 4x16 Decoder and 4 Programmable 'OR' gates.

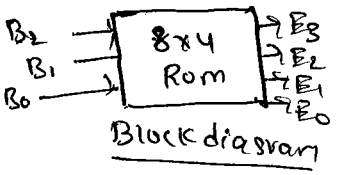
Binary to EXCESS-3 :-

* It has 3 bit binary and 4-bit excess-3 . So Rom has 3-inputs and 4-olps.

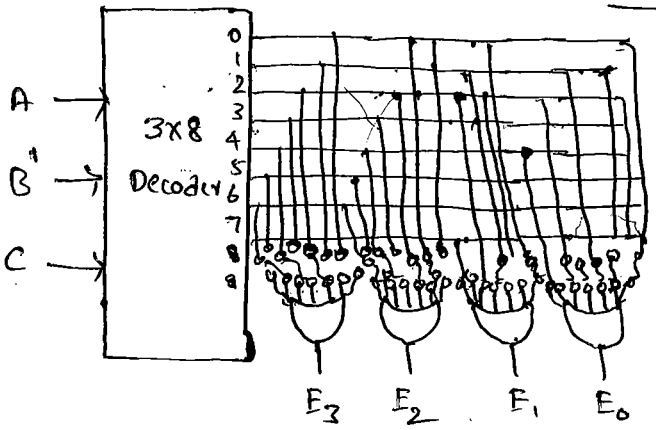
∴ Required Rom size is $2^3 \times 4 = 8 \times 4$ Rom.

Inputs			OIPS			
A	B	C	E_3	E_2	E_1	E_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

Logic Diagram :-



Block diagram



* $E_0(A,B,C) = \Sigma m(0,2,4,6)$; $E_1(A,B,C) = \Sigma m(0,3,4,7)$

$E_2(A,B,C) = \Sigma m(1,2,3,4)$; $E_3(A,B,C) = \Sigma m(5,6,7)$

* Pb:- Design a combination CKt using a Rom . The CKt accepts a 3-bit number and generates an OIP binary number equal to the square of the input number :- ?

Sol:- Truth table

Decimal	Inputs			OIPS						Decimal
Dec	A	B	C	B_5	B_4	B_3	B_2	B_1	B_0	Square
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	1
2	0	1	0	0	0	0	1	0	0	4
3	0	1	1	0	0	1	0	0	1	9
4	1	0	0	0	1	0	0	0	0	16
5	1	0	1	0	1	1	0	0	1	25
6	1	1	0	1	0	0	1	0	0	36
7	1	1	1	1	1	0	0	0	1	49

from the table we have the following equations.

$B_0 = C$

$B_1 = 0$

$B_2 = \bar{A}B\bar{C} + AB\bar{C} = B\bar{C}$

$B_3 = \bar{A}BC + A\bar{B}C = C(A \oplus B)$

$B_4 = A\bar{B}\bar{C} + A\bar{B}C + ABC =$

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

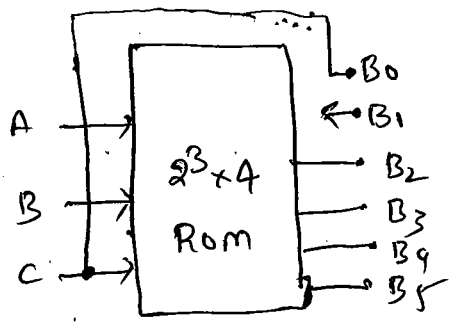
$= \bar{A}(\bar{B} + C)$

$B_5 = AB$

$\therefore B_0 = \Sigma m(1,3,5,7)$; $B_1 = 0$; $B_2 = \Sigma m(3,5)$; $B_4 = \Sigma m(4,5,7)$

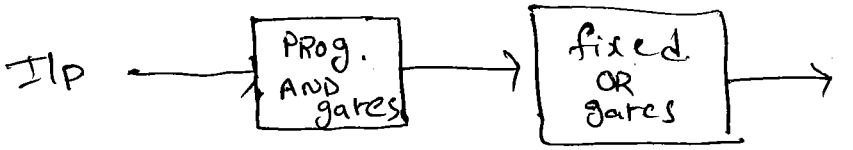
$B_5 = \Sigma m(6,7)$

To implement the Rom we need capacity $2^3 \times 6$. The below figure shows the block diagram.



Programmable Array Logic (PAL):-

PAL has Programmable AND Array and a fixed OR Array. Here only AND gates are Programmable and OR array is fixed.



Ex:- Implement 4 bit BCD to XS-3 code converter using PAL. ?

Sol:- Step (1):- To construct 4-bit BCD to Excess-3 Code. we have truth table.

Inputs				outputs				
B4	B3	B2	B1	X4	X3	X2	X1	
0	0	0	0	0	0	1	1	(3)
0	0	0	1	0	1	0	0	(4)
0	0	1	0	0	1	0	1	(5)
0	0	1	1	0	1	1	0	(6)
0	1	0	0	0	1	1	1	(7)
0	1	0	1	1	0	0	0	(8)
0	1	1	0	1	0	0	1	(9)
0	1	1	1	1	0	1	0	(10)
1	0	0	0	1	0	1	1	(11)
1	0	0	1	1	1	0	0	(12)

Step (2):- K-map simplifications for X_4, X_3, X_2, X_1 are

$X_4(B_4, B_3, B_2, B_1) = \Sigma m(5,6,7,8,9)$

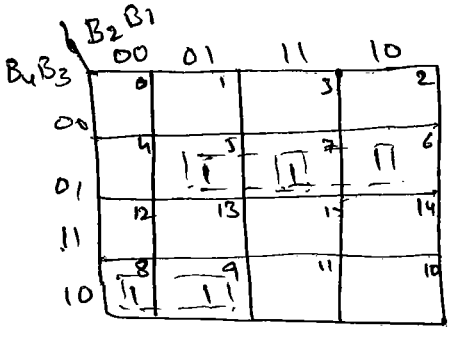
$X_3(B_4, B_3, B_2, B_1) = \Sigma m(1,2,3,4,9)$

$X_2(\quad,\quad,\quad,\quad) = \Sigma m(0,3,4,7,8)$

$X_1(\quad,\quad,\quad,\quad) = \Sigma m(0,2,4,6,8)$

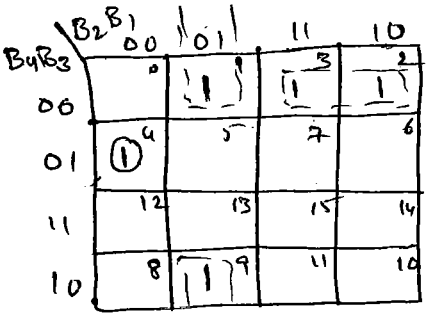
Step (3):- Get the K-map eq^{n}s for X_4, X_3, X_2, X_1.

$X_4 = \Sigma m(5,6,7,8,9)$



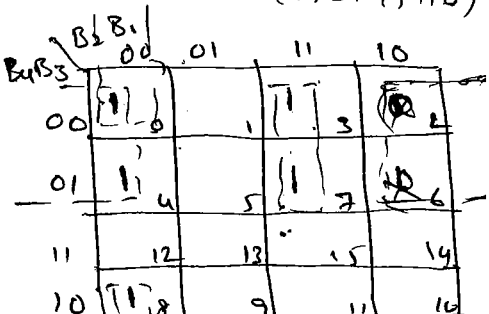
$X_4 = B_4\bar{B}_3\bar{B}_2 + \bar{B}_4 B_3 B_1 + \bar{B}_4 B_3 B_2$

$X_3 = \Sigma m(1,2,3,4,9)$

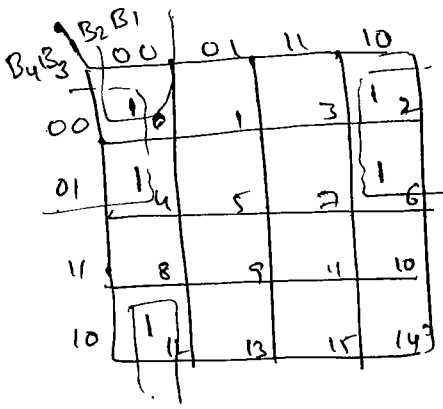


$X_3 = \bar{B}_4\bar{B}_3 B_2 + \bar{B}_3\bar{B}_2 B_1 + \bar{B}_4 B_3 \bar{B}_2 \bar{B}_1$

$X_2 = \Sigma m(0,3,4,7,8)$



$X_2 = \bar{B}_4 B_2 B_1 + \bar{B}_4 \bar{B}_2 \bar{B}_1 + \bar{B}_3 \bar{B}_2 \bar{B}_1$

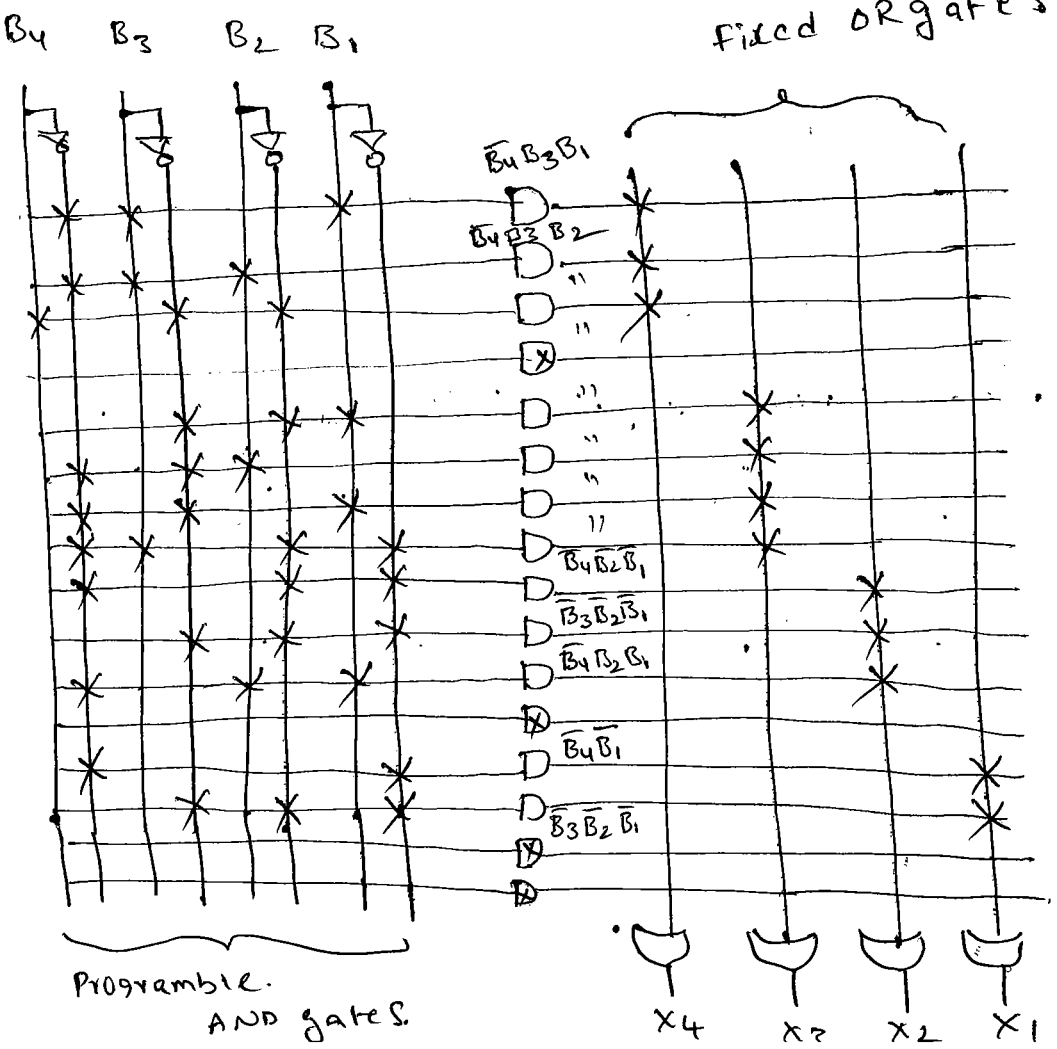


$X_1 = \bar{B}_4 \bar{B}_1 + \bar{B}_3 \bar{B}_2 \bar{B}_1$

PAL TABLE :-

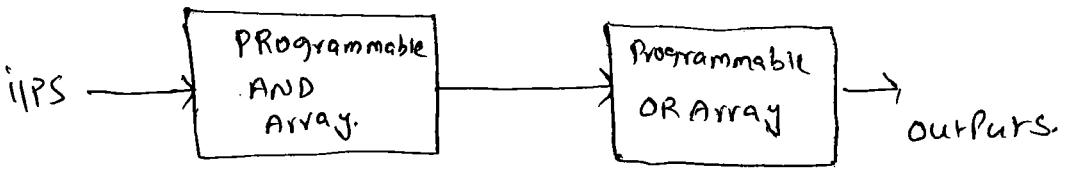
Product terms	AND gate i/ps B_4	B_3	B_2	B_1	outputs
1					x
2	0	1	–	1	
3	0	1	1	–	$X_4 = \bar{B}_4 B_3 B_1 + \bar{B}_4 B_3 B_2$
4	1	0	0	–	$+ B_4 \bar{B}_3 \bar{B}_2$
	–	–	–	–	
5	–	0	0	–	$X_3 = \bar{B}_2 B_1 \bar{B}_3$
6					
7	0	0	1	–	$+ B_2 \bar{B}_4 \bar{B}_3 + B_1 \bar{B}_4 \bar{B}_3$
8	0	0	–	1	$+ \bar{B}_4 B_3 \bar{B}_2 \bar{B}_1$
	0	1	0	0	
9	0	–	0	0	$X_2 = \bar{B}_4 \bar{B}_2 \bar{B}_1$
10	–	0	0	0	
11	0	–	1	1	$+ \bar{B}_3 \bar{B}_2 \bar{B}_1 +$
12	–	–	–	–	$\bar{B}_4 B_2 B_1$
13	0	–	–	0	
14	0	0	0	–	$X_1 = \bar{B}_4 \bar{B}_1 +$
15	–	–	–	–	$\bar{B}_2 \bar{B}_1 \bar{B}_3$
16	–	–	–	–	

Step (4) :- Design and implementation of PAL CKT using the above PAL Table.



PLA (Programmable Logic Array) :-

→ In Programmable logic array where both the AND and OR arrays can be Programmed. The Product terms in the AND and OR array may be Shared by any an OR gate to Provide the required Sum of Products.



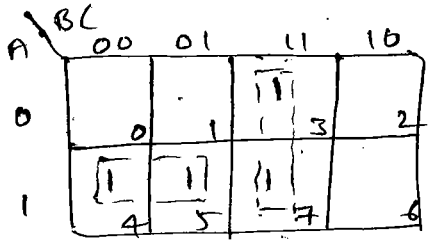
Eg①:- Implement the following boolean function F_1, F_2, Combinational logic circuit boolean function using PLA.

$F_1(A,B,C) = \Sigma m(3,4,5,7)$

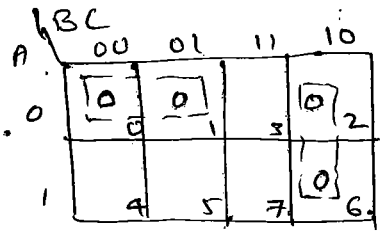
$F_2(A,B,C) = \Sigma m(1,4,6)$

Sol^n:- Step ①:- Implement the K-maps for given Boolean functions.

$F_1(A,B,C) = \Sigma m(3,4,5,7)$

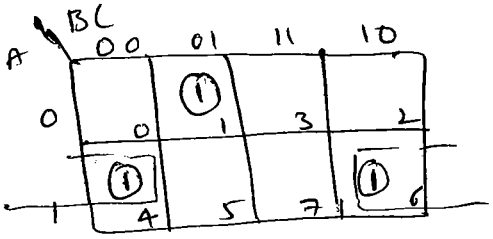


$F_1(T) = A\bar{B} + BC$

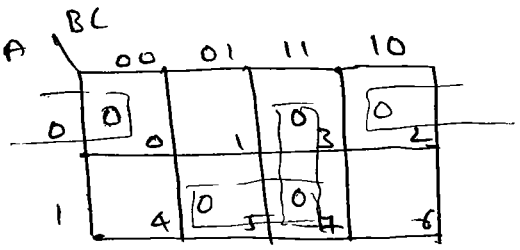


$F_1(C) = \bar{A}\bar{B} + B\bar{C}$

$F_2(A,B,C) = \Sigma m(1,4,6)$



$F_2(T) = A\bar{C} + \bar{A}\bar{B}C$



$F_2(C) = \bar{A}\bar{C} + BC + AC$

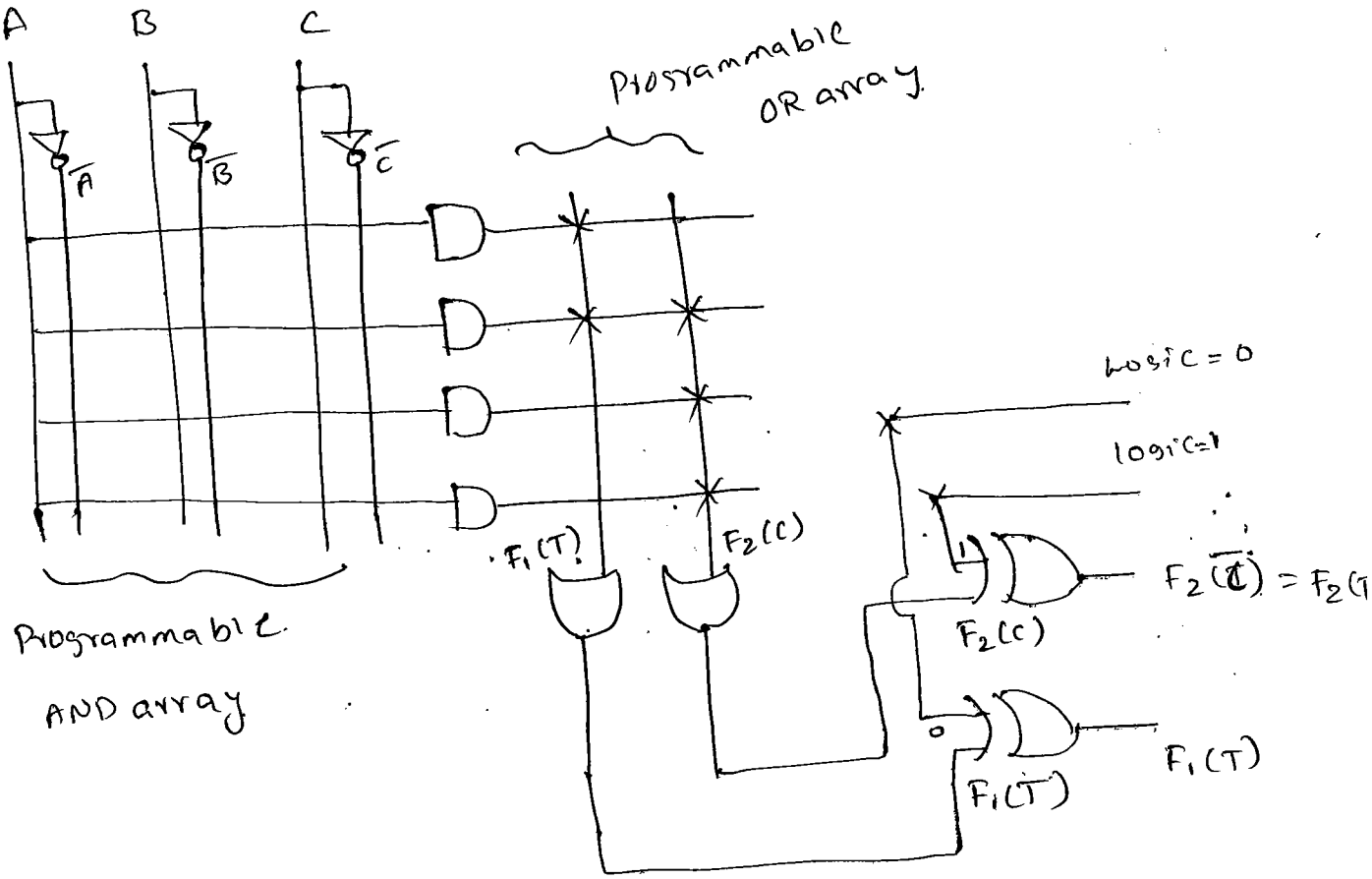
$F_1(T) = A\bar{B} + BC$

$F_2(C) = \bar{A}\bar{C} + BC + AC$

Step(2):- Design PLA Programming table.

Product term	Inputs			Outputs	
	A	B	C	$F_1(T)$	$F_2(C)$
$A\bar{B}$	1	0	–	1	–
BC	–	1	1	1	1
$\bar{A}\bar{C}$	0	–	0	–	1
AC	1	–	1	–	1

Step(3):- Design a PLA logic circuit-

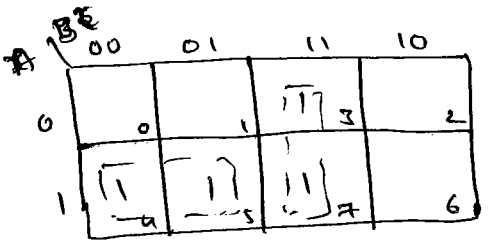


Pb(2) Implement the following boolean function F_1, F_2 combinational logic circuit boolean function using PLA

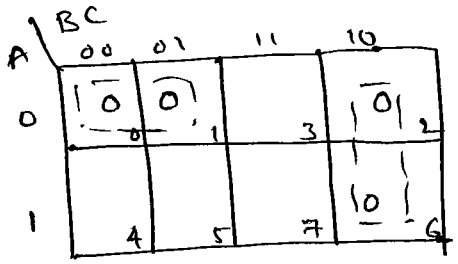
$F_1(A,B,C) = \Sigma m(3,4,5,7)$ $F_2(A,B,C) = \Sigma m(1,4,6)$

Soln:- Step(i):-

$F_1(A,B,C) = \Sigma m(3,4,5,7)$

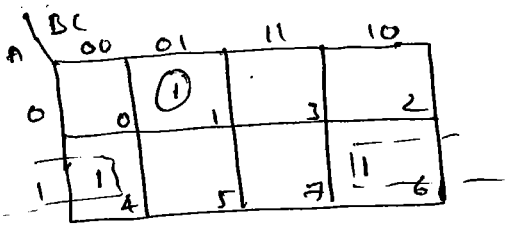


$F_1(T) = A\bar{B} + BC$

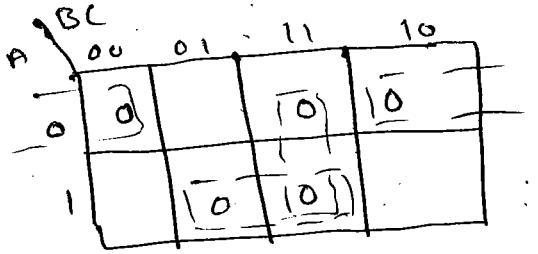


$F_1(C) = \bar{A}\bar{B} + B\bar{C}$

$F_2(A,B,C) = \Sigma m(1,4,6)$



$F_2(T) = A\bar{C} + \bar{A}\bar{B}C$



$F_2(C) = \bar{A}\bar{C} + BC + AC.$

Step(2):- From the above 4 equations we select the terms which is equal when compared to all.

$F_1(T) = A\bar{B} + \underline{BC}$

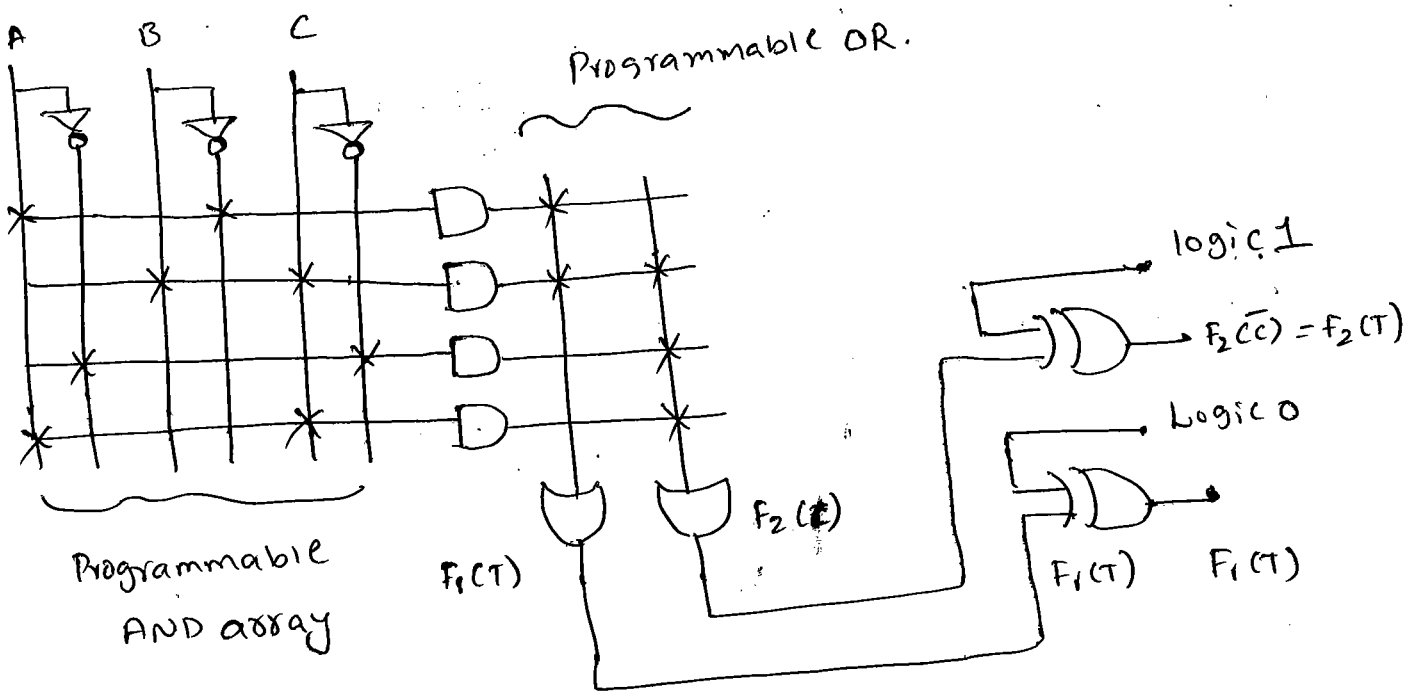
$F_2(C) = \bar{A}\bar{C} + \underline{BC} + AC.$

In the above 2 equations 'BC' is a similar term. when compared to other terms all are different so select the above 2 functions for the PLA Programming table.

Step(3):- Design of PLA Table from the above two eq'ns is as follows.

Product term	Inputs			outputs.	
	A	B	C	$F_1(T)$	$F_2(T)$
$A\bar{B}$	1	0	–	1	–
BC	–	1	1	1	1
$\bar{A}\bar{C}$	0	–	0	–	1
AC	1	–	1	–	1

Step(4):- Design of PLA logic CKT.



Comparision of PROM, PAL, PLA :-

PROM	PAL	PLA.
1) AND array is fixed and OR array is Programmed.	1) AND array is Programmed and OR array is fixed	1) Both AND and OR are Programmable
2) very cheapest and simple to use	2) Cheaper and simple construction.	2) Complex than PAL and PROM and more costlier
3) All minterms are decoded	3) AND array is Programmed to get desired minterms	3) AND array is desired to get minterms.
4) Standard SOP implemented using PROM	4) SOP implemented using PAL	4) SOP implemented using PLA.

BCD to Excess-3 Code for PLA:-

Step(i):- Draw a Tables for Both BCD no's such as. (B_3, B_2, B_1, B_0) and excess-3 (E_3, E_2, E_1, E_0). and. Consider don't care conditions. Since we will get 4 variable K-map.

Step (ii):- from the above tables we will get the minterms. as follows.

$E_0 = \Sigma m(0,2,4,6,8)$, $E_1 = \Sigma m(0,3,4,7,8)$

$E_2 = \Sigma m(1,2,3,4,9)$, $E_3 = \Sigma m(5,6,7,8,9)$

and add don't cares $d = \Sigma d(10,11,12,13,14,15)$.

Step (iii):- Draw the K-map's for the above minterms for Suitable Equations. we will get the following functions.

$E_0(T) = \overline{B_0}$, $E_0(C) = \overline{B_0}$

$E_1(T) = \overline{B_1}\,\overline{B_0} + B_1 B_0$, $E_1(C) = \overline{\overline{B_1} B_0 + B_1 \overline{B_0}}$

$E_2(T) = B_2\overline{B_1}\,\overline{B_0} + \overline{B_2} B_0 + \overline{B_2} B_1$, $E_2(C) = \overline{\overline{B_2}\,\overline{B_1}\,\overline{B_0} + B_2 B_0 + B_2 B_1}$

$E_3(T) = B_3 + B_2 B_0 + B_2 B_1$, $E_3(C) = \overline{\overline{B_3}\,\overline{B_2} + \overline{B_3}\,\overline{B_1}\,\overline{B_0}}$

PLA Table:-

Product terms	Inputs				outputs			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
					$(\overline{T})$	$(\overline{C})$	$(\overline{T})$	$(\overline{T})$
1. B_3	1	–	–	–	1	–	–	–
2. B_2B_0	–	1	–	1	1	1	–	–
3 B_2B_1	–	1	1	–	1	1	–	–
4. $\overline{B_2}\,\overline{B_1}\,\overline{B_0}$	–	0	0	0	–	1	–	–
5. $\overline{B_1}\,\overline{B_0}$	–	–	0	0	–	–	1	–
6. B_1B_0	–	–	1	1	–	–	1	–
7. $\overline{B_0}$	–	–	–	0	–	–	–	1.

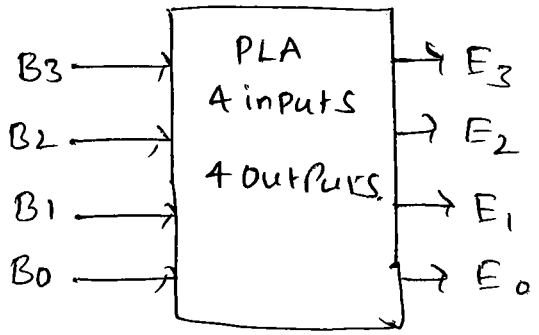
Note:- In the above Eq's $E_3(T) = B_3 + B_2B_0 + B_2B_1$,

$E_2(C) = \overline{B_2}\,\overline{B_1}\,\overline{B_0} + B_2B_0 + B_2B_1$

(T)&(C)

Compare both has similar terms such as B_2B_0, B_2B_1 so consider these two terms along with $E_0(T)$ and $E_1(T)$.

Block Diagram of PLA using 4 inputs & 4 o/ps



Important Questions:-

1. Brief notes on Architecture of PLD's. (7M)
2. Implement $f(A,B,C,D) = \Sigma m(0,1,3,5,6,8,9,11,12,13)$ using PAL and explain it's Procedure. (7M)
3. Write merits and Demerits of PROM. (7M)
4. Design and implement of full adder with PLA (7M)
5. Design BCD to excess-3 code Converter using PLA (7M)
6. Design BCD to excess-3 code using
7. Discuss how PROM, EPROM and EEPROM technologies differ from each other. (6M)
8. Draw Structure of 8x1 PROM and explain it's operation. (7M)
9. Design a Combinational Logic CKT using PROM, the CKT accepts a 3-bit number and generates an o/p binary number equal to Square of input number? (7M)
10. Realize the following four Boolean functions using PAL. (8M)

$F_1(w,x,y,z) = \Sigma m(1,2,3,7,9,11)$

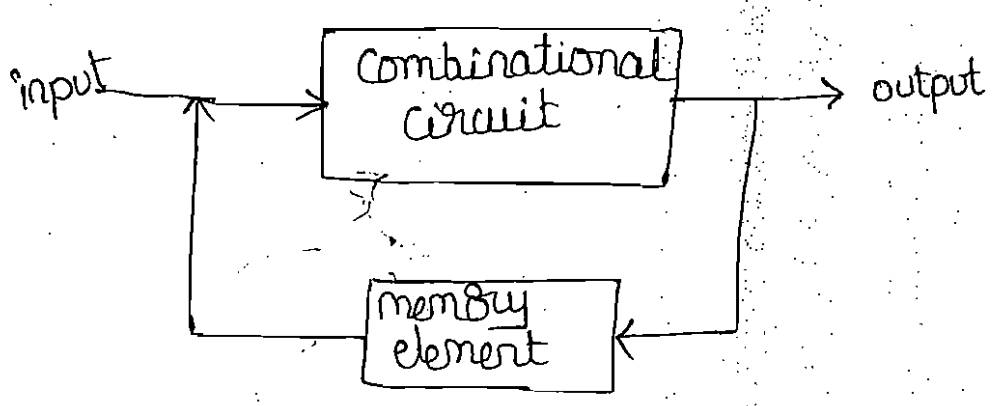
$F_2(w,x,y,z) = \Sigma m(0,1,2,3,10,12,14)$

$F_3(w,x,y,z) = \Sigma m(4,5,6,7,9,15)$

$F_4(w,x,y,z) = \Sigma m(1,2,3,10,13,15)$

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.

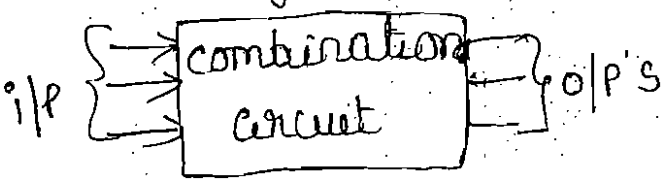
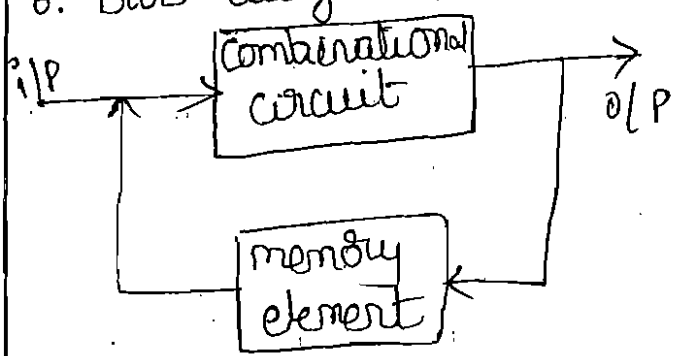
→ latch is a sequential device that checks all its inputs continuously and changes its outputs accordingly at any time independent of a clock signal.

→ 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.

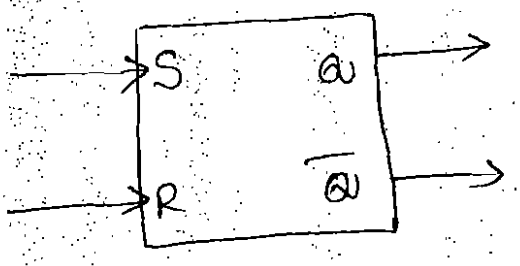
Latch	Flip-flop.
1. A latch is an electronic sequential logic circuit used to store information in an asynchronous arrangement.	1. A flip flop is an electronic sequential logic circuit used to store information in an asynchronous arrangement.
2. one latch can store one bit information, but output state 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 synchronised 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.



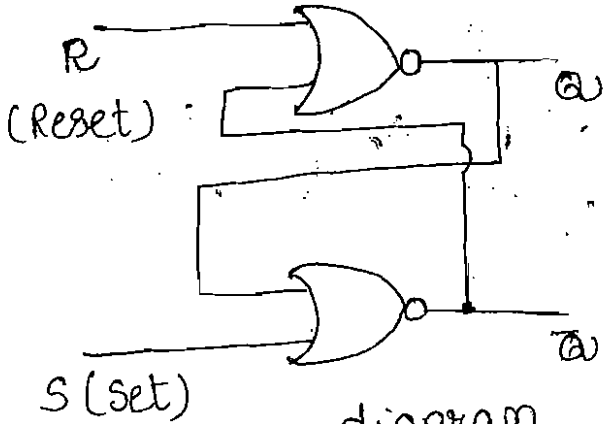
Logic symbol of S-R latch

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.

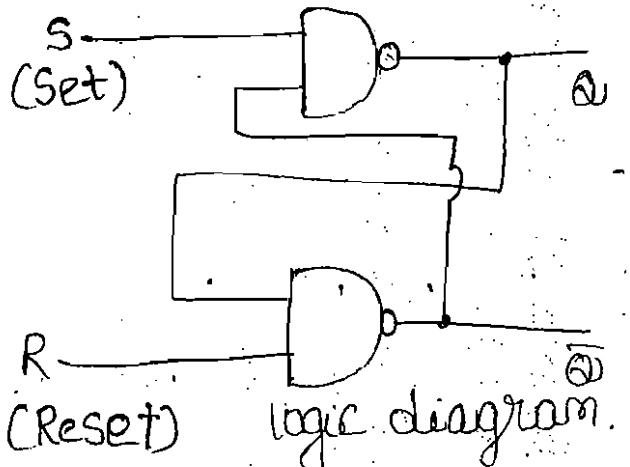
Simplified truth table.

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 occurrence 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

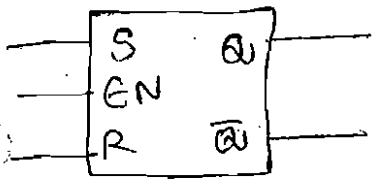
← Simplified truth table.

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	

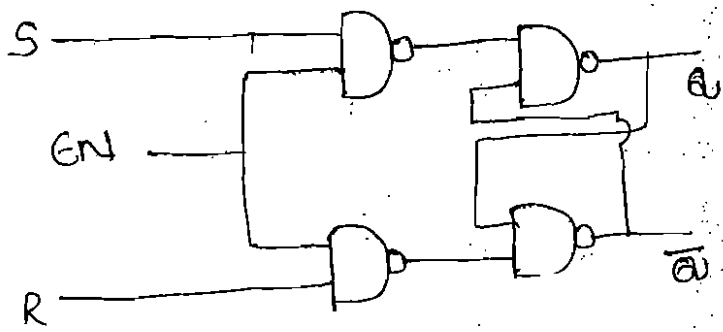
Truth table.

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.

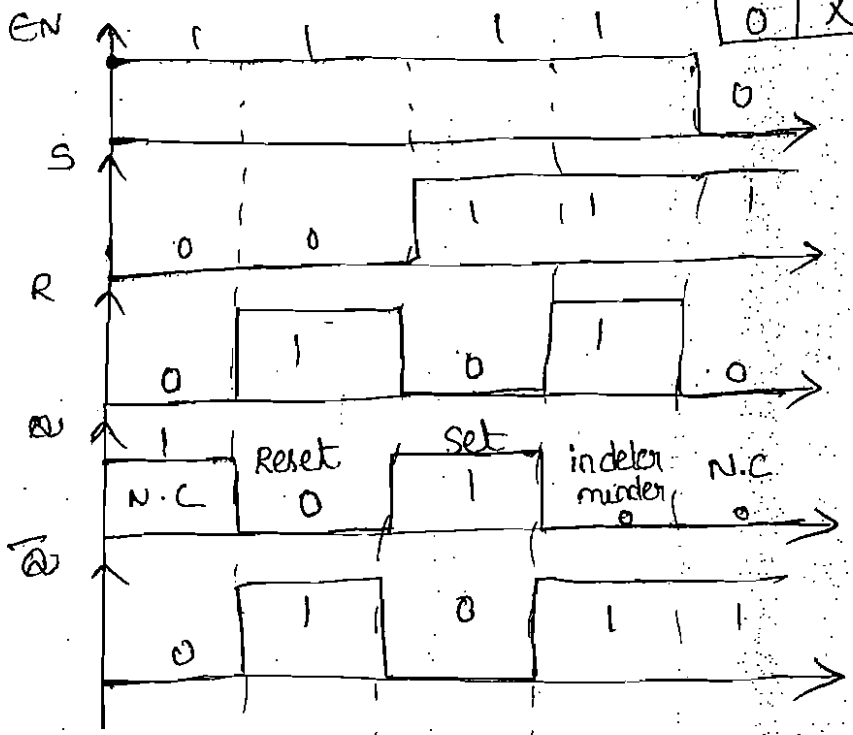


Logic symbol.



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.

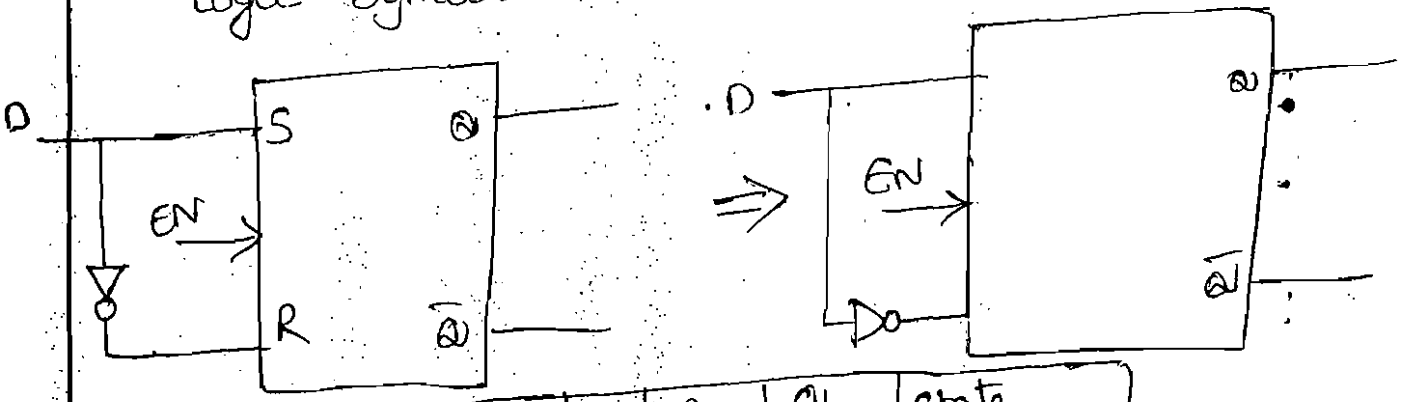
The Gated D-latch; - In many applications, it is not necessary to have separate S and R inputs to a latch. If the input combinations S=R=0 and S=R=1 are never needed, the S and R are always the complement of each other. So, to construct a latch with a single input (S) and obtain the R input by inverting 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.

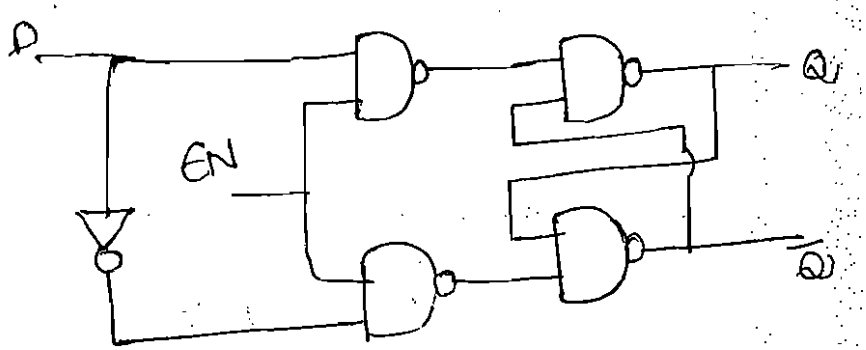
→ The output Q follows the D-input when En is high. So this latch is said to be transparent.

Logic Symbol

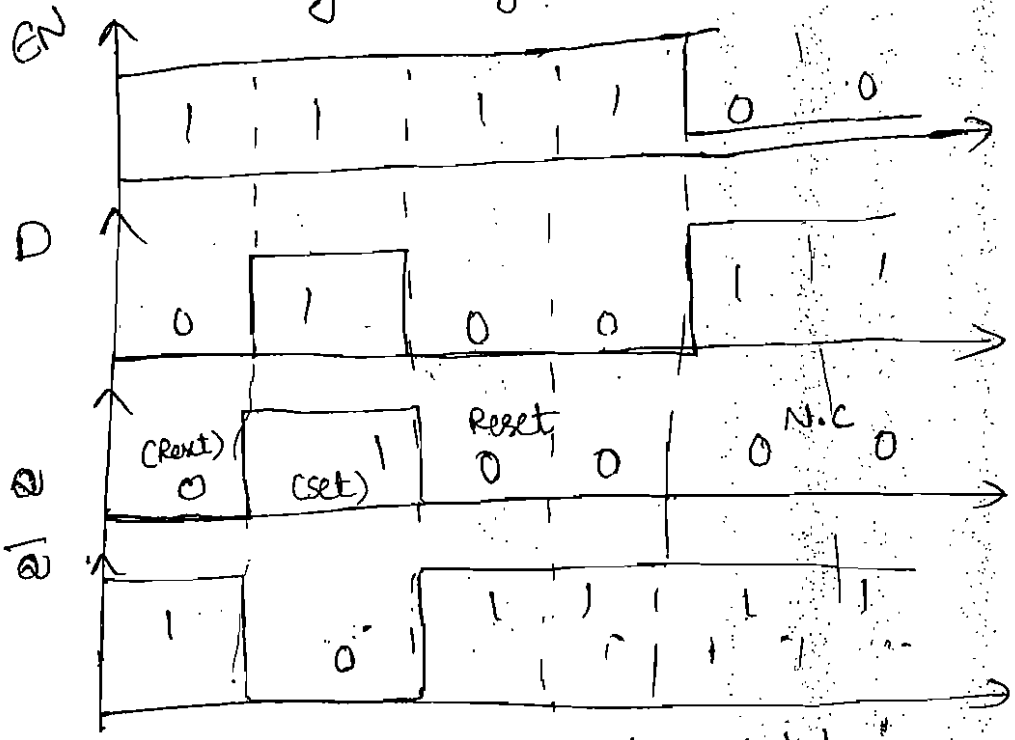


EN	D	Q_n	Q_{n+1}	State
1	0	0	0	Reset
1	0	1	0	
1	1	0	1	Set
1	1	1	1	
0	X	0	0	NO change (NC)
0	X	1	1	

Truth table



Logic diagram.



waveforms for Gated D-latch.

Flip-flops :-

Types of flip-flops :-

→ S-R flip-flop

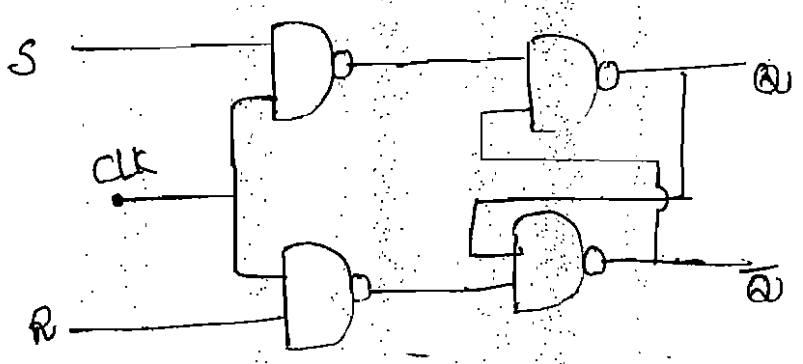
→ D flip-flop

→ J-K flip-flop

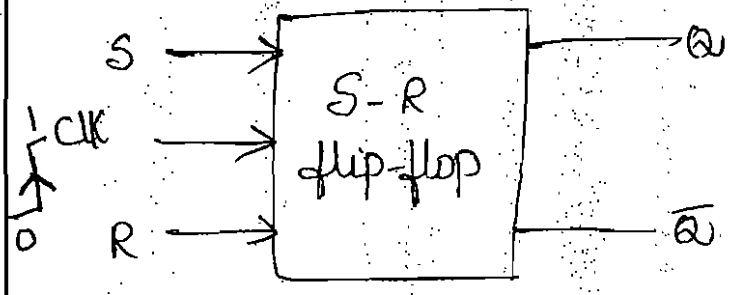
→ T flip-flop.

S-R flip-flop :-

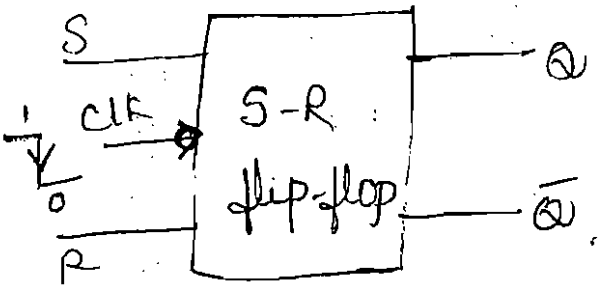
Logic diagram



Symbol of +ve edge trigger.



Symbol of -ve edge trigger

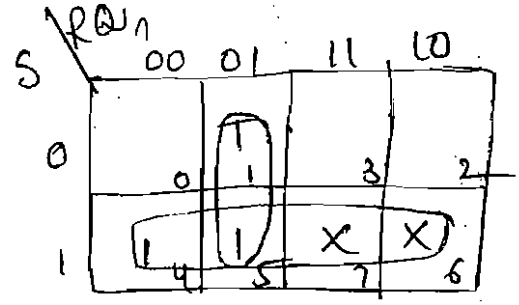


Truth table :-

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:-

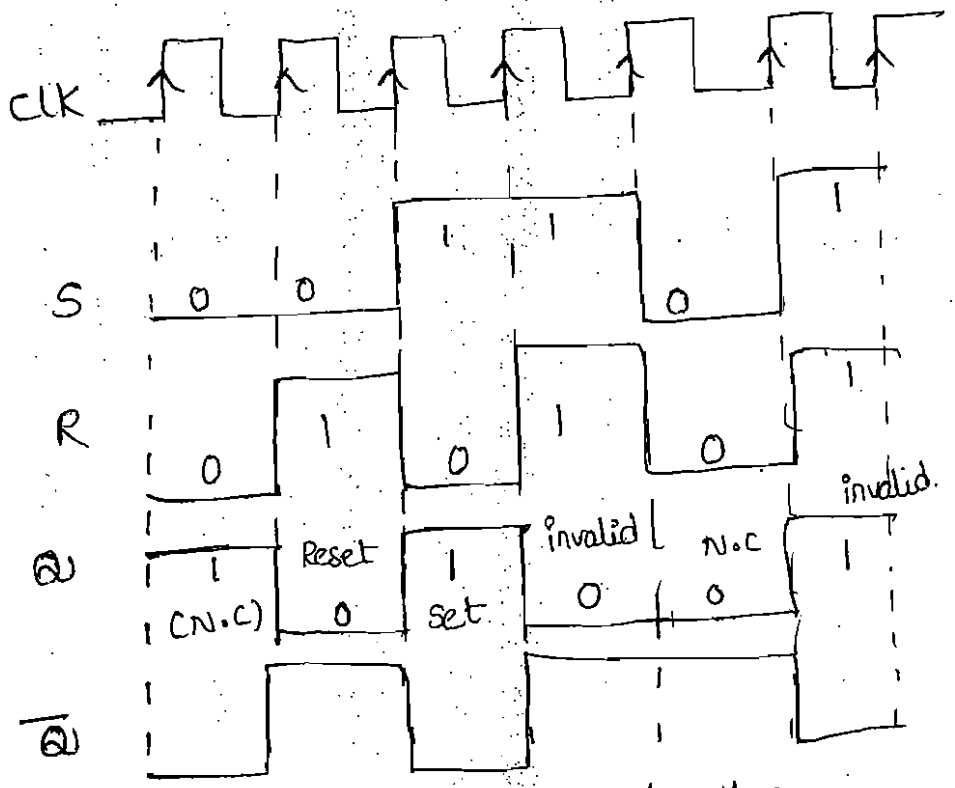
S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	X

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

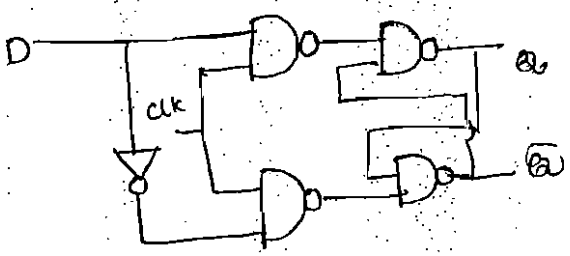
Timing diagram :-



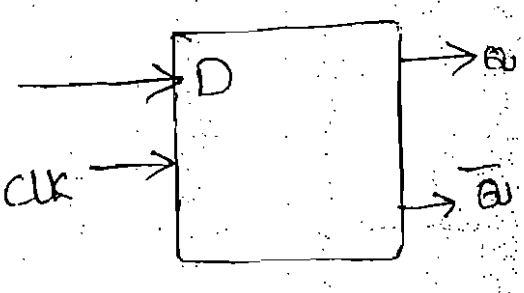
waveforms for S-R flip-flop.

D-flip-flop :-

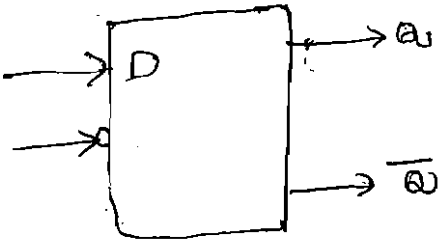
logic diagram :-



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 :-

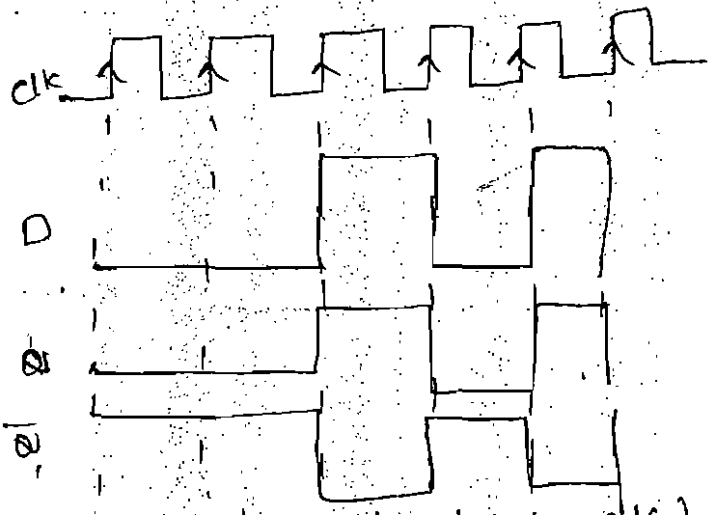
Q_n / D	0	1
0	0	1
1	1 (2)	1 (3)

$Q_{n+1} = D$.

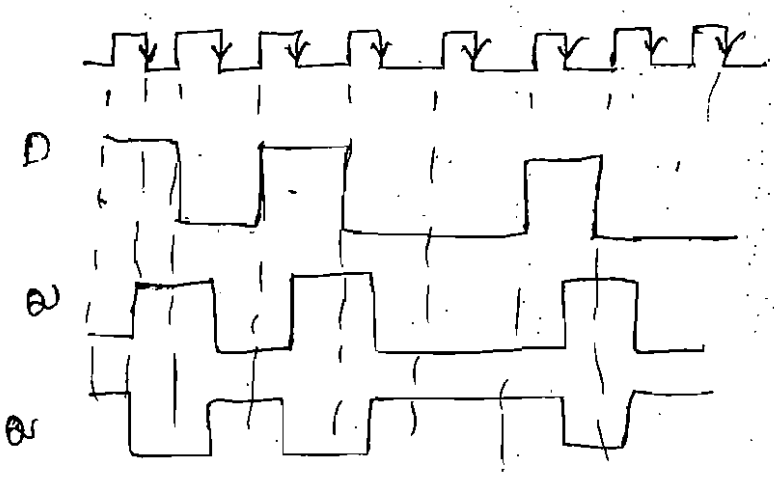
Excitation table :-

Present state Q_n	Next state Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

waveforms



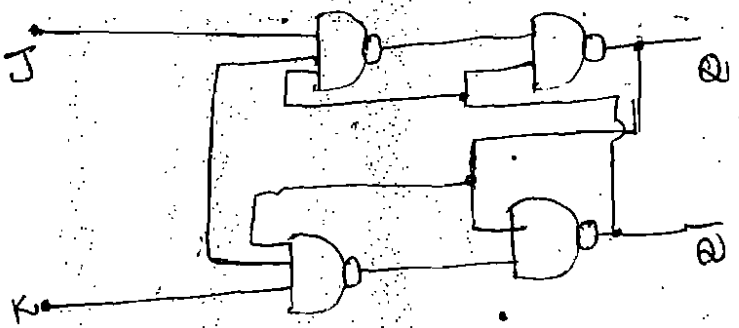
(Positive - edge trigger clk).



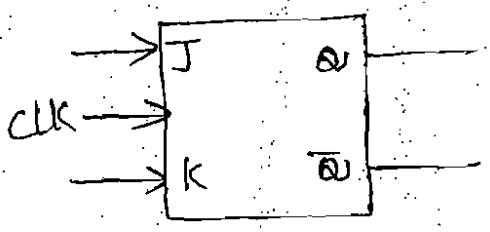
(negative - edge trigger clk).

(J-K - flip flops).

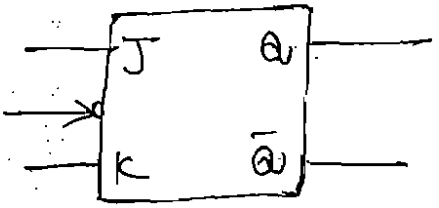
logic diagram



logic symbol.



(positive-edge)



(negative-edge)

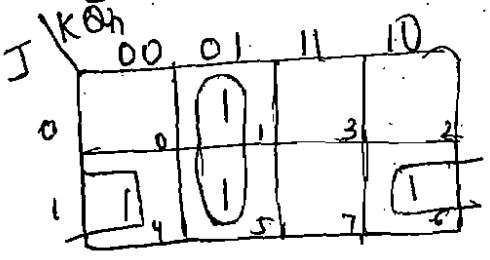
Truth table:-

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

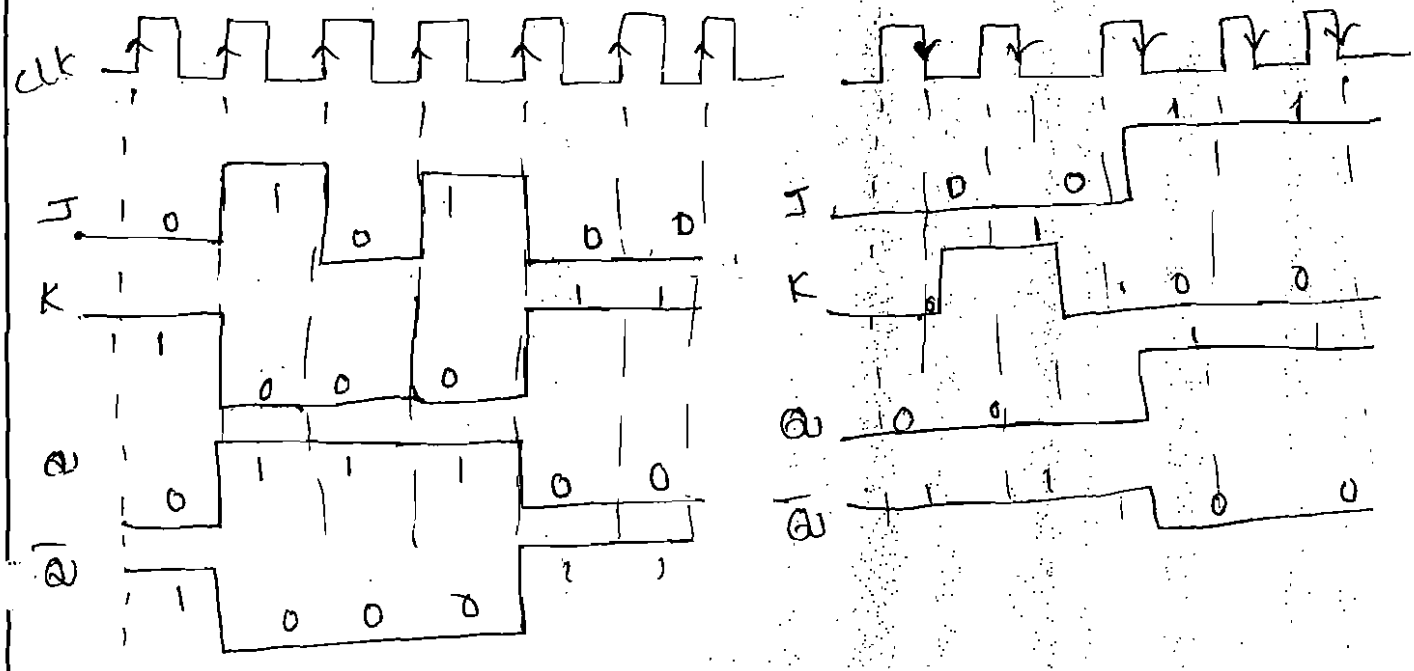


$$Q_{n+1} = \bar{K}Q_n + J\bar{Q}_n$$

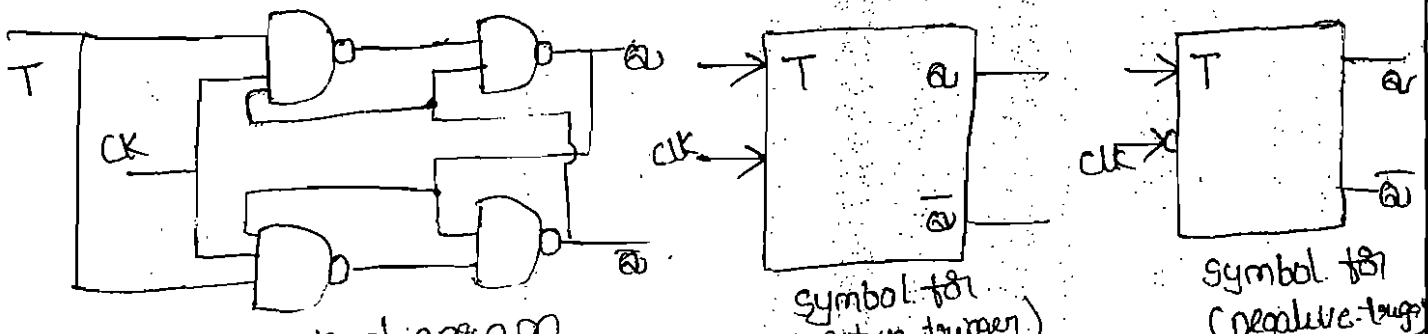
excitation table

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

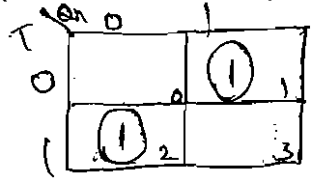
Truth table

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	
~~↑~~	x	0	0	NO change
~~↓~~	x	1	1	

Characteristic equation

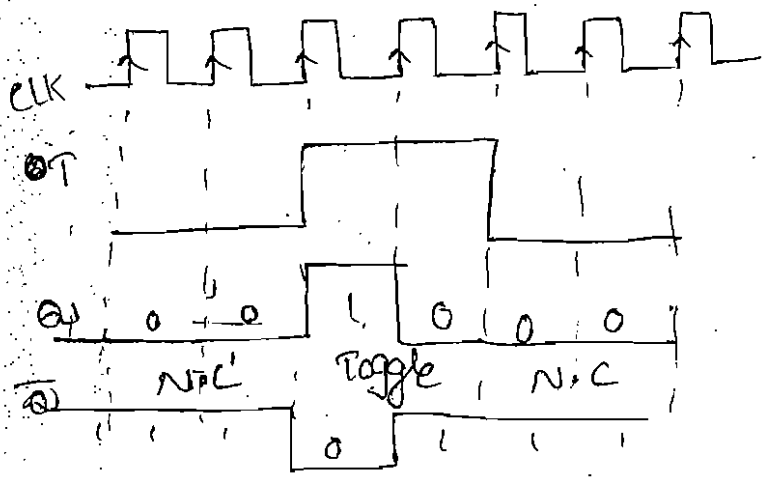


$Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n$.

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.

$$t_p >> t$$

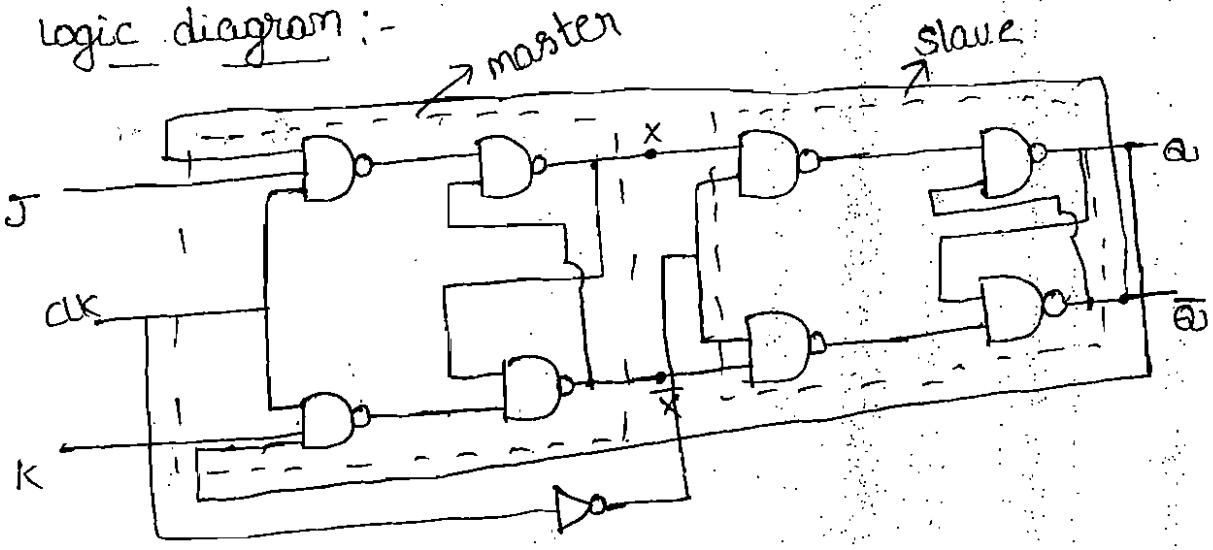
$t_p \rightarrow$ pulse width

$t \rightarrow$ propagation delay.

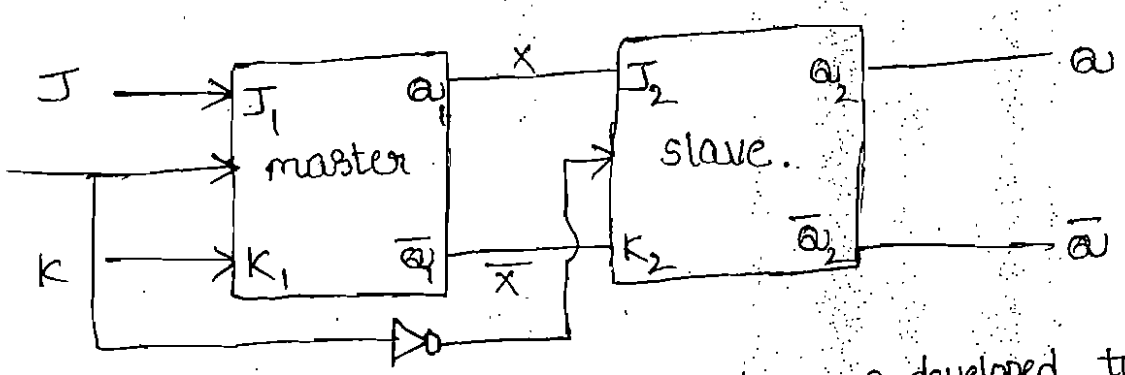
The clock pulse width should be such as to allow only one change to complement the state and not too long to allow many changes resulting in uncertainty about the final state. This is a stringent requirement which cannot be ensured in practice. This problem is eliminated using master-slave flip-flop or edge trigged flip-flop.

master-slave J-K flip-flop :-

Logic diagram :-



Symbol ;-



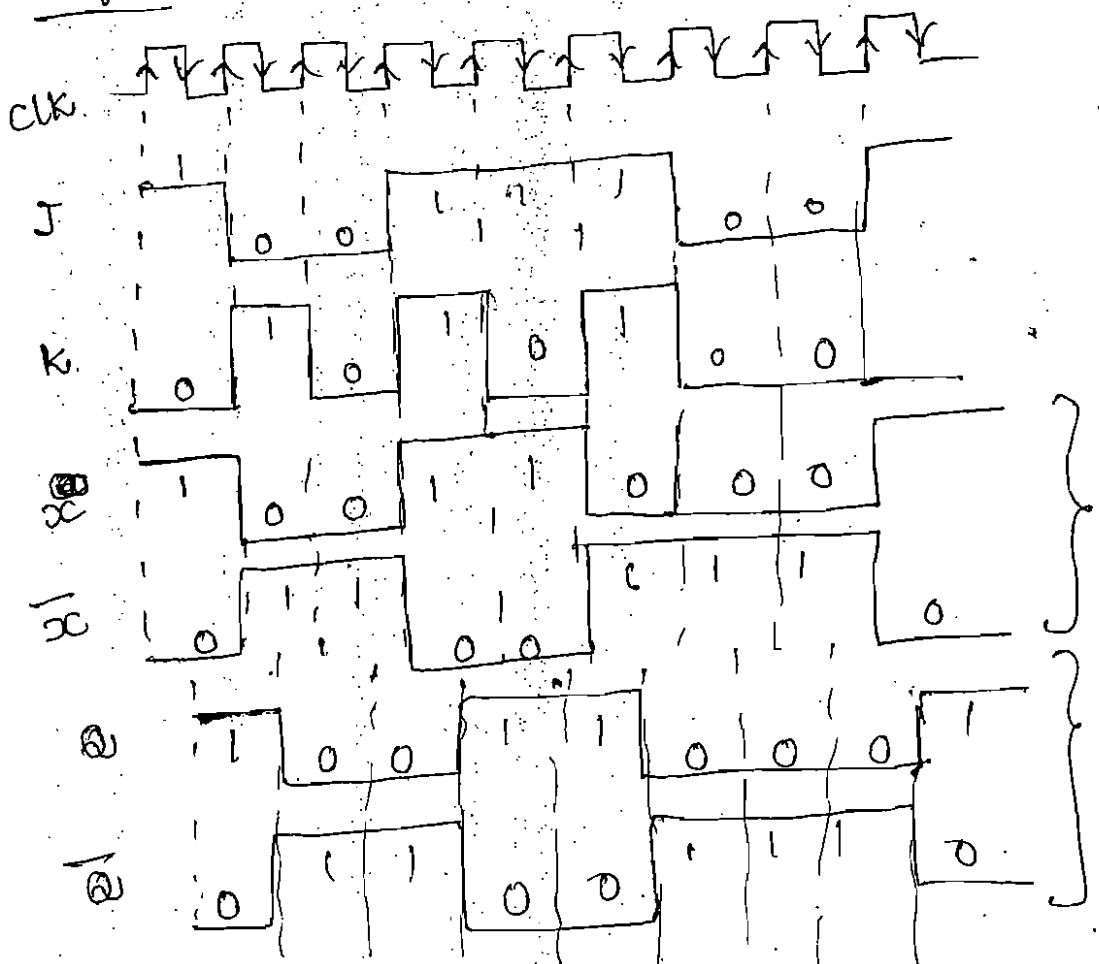
The master-slave flip flop was developed to make the synchronous operation more predictable, that is, to avoid the problems of logic race in clocked flip-flops. This improvement is achieved by introducing a known time delay between the time that the flip-flop responds to a clock pulse and the time the 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.

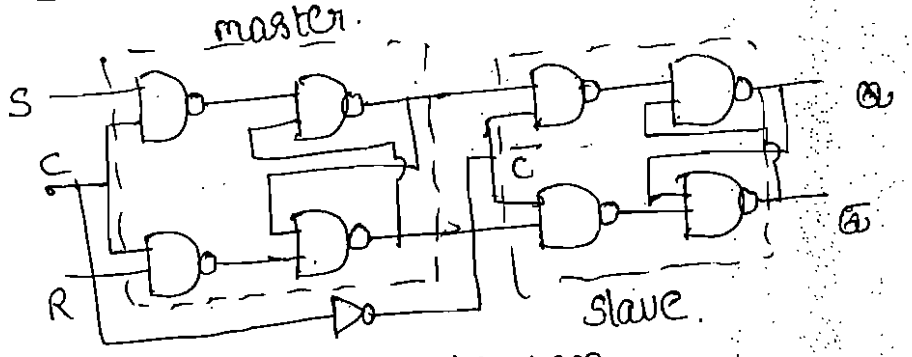
The master-slave flip-flops function very much like the negative-edge triggered flip-flops except for one more disadvantage The control inputs must be held stable while clk is high, otherwise an unpredictable operation may occur. This problem with the master-slave flip-flop is overcome with an improved 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 :-

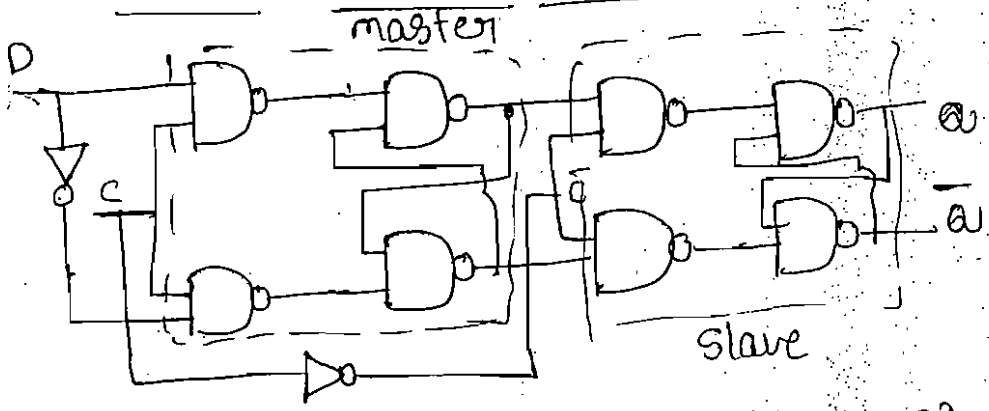


Logic diagram

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

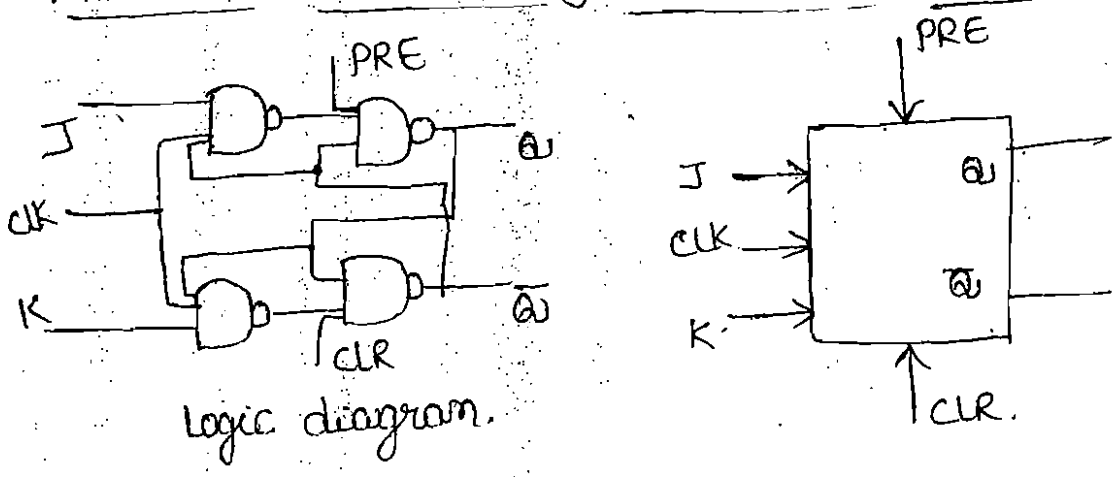
D	clk	Q	State
0	⎍	0	Reset
1	⎍	1	set

→ 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)x

→ 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'.

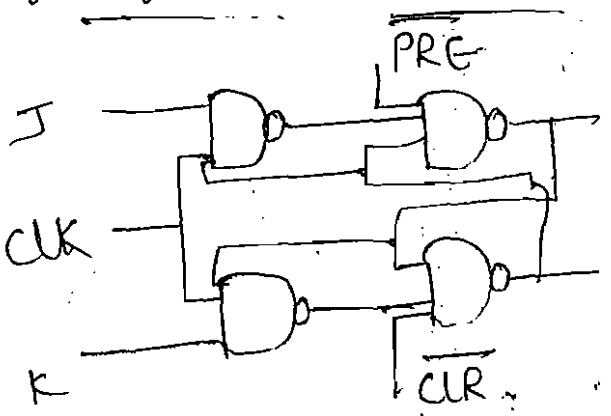
→ when PRE = 1, CLR = 0 then Asynchronous input PRESET is active the flip flop output is '1'.

→ when PRE = '1', CLR = '1'. This condition should not be used since it can result in an invalid state.

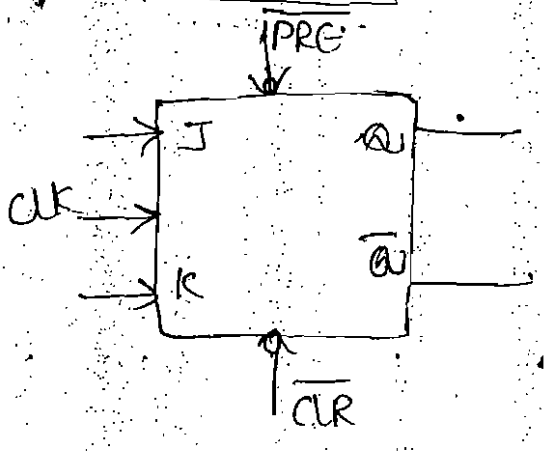
DC SET (PRE)	DC RESET (CLR)	FF response
0	0	clock operation
0	1	Q = 0
1	0	Q = 1
1	1	Not used.

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

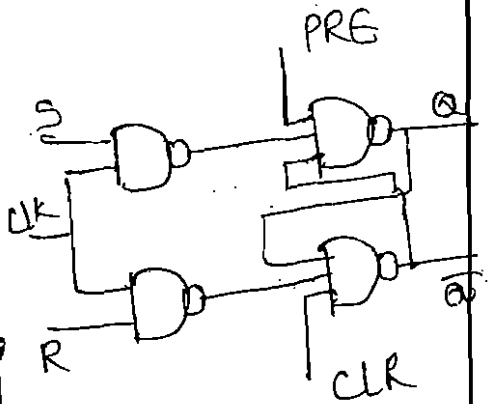
Truth table for J-K flip-flop (both Asynchronous & Synchronous Inputs).

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.	

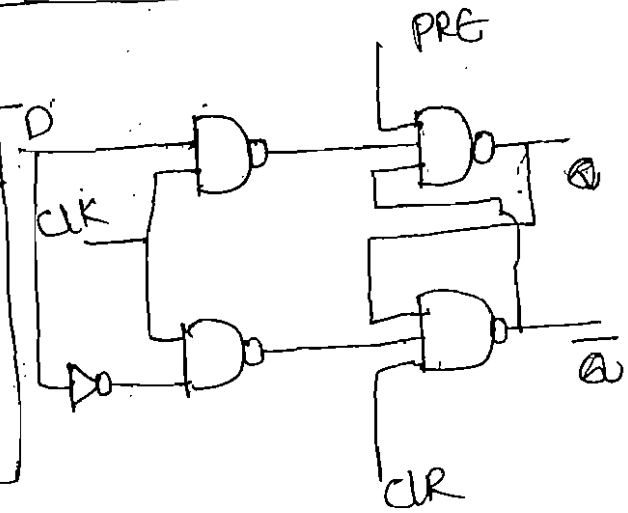


logic diagram

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)

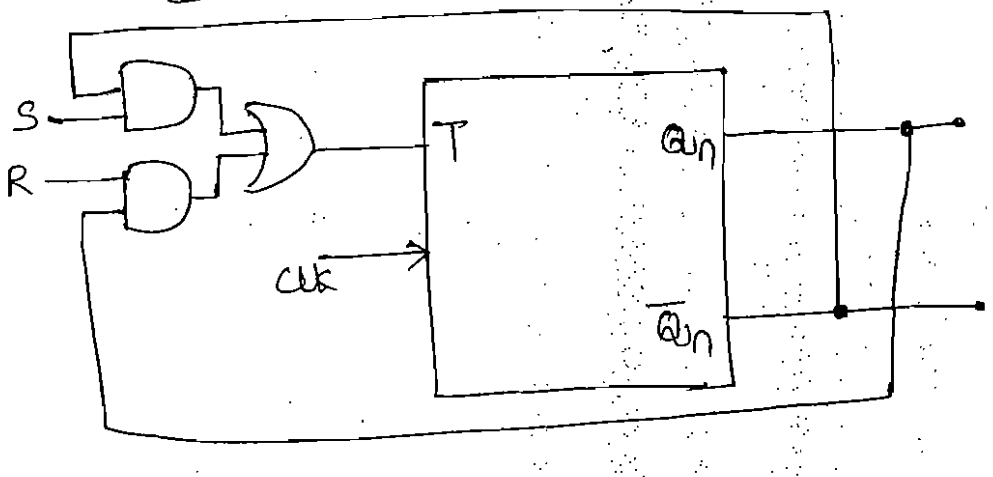
S	R	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	X	X
1	1	1	X	X

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



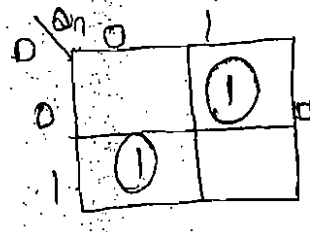
→ Conversion of T-flip-flop to D-flip-flop.

Available → T-flipflop → excitation table

Required → D-flipflop → characteristic table.

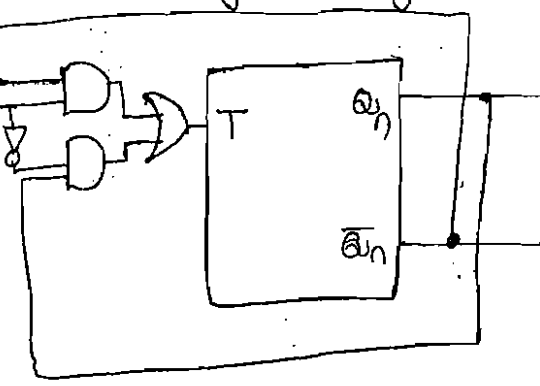
D	Q_n	Q_{n+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

K-map for T



$T = D\cdot\overline{Q_n} + \overline{D}Q_n$

Logic diagram



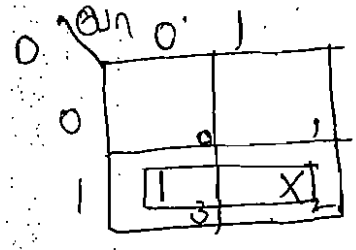
→ Construct a D-flip flop ~~to~~ using S-R flip flop

Available → S-R flipflop → excitation table.

Required → D flip flop → characteristic table.

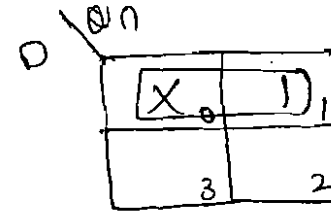
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-map for S



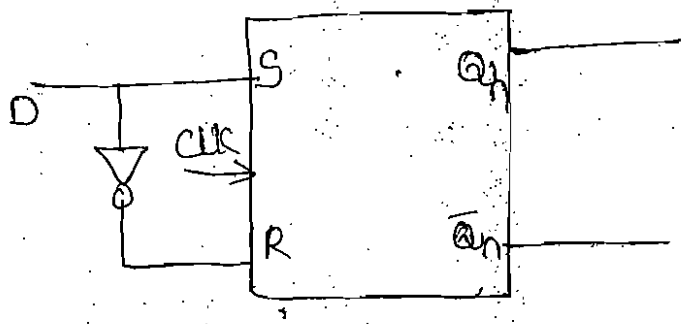
$S = D$

K-map for R



$R = \overline{D}$

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 \ $K Q_n$	00	01	11	10
0	0	X	0	0
1	1	X	0	1

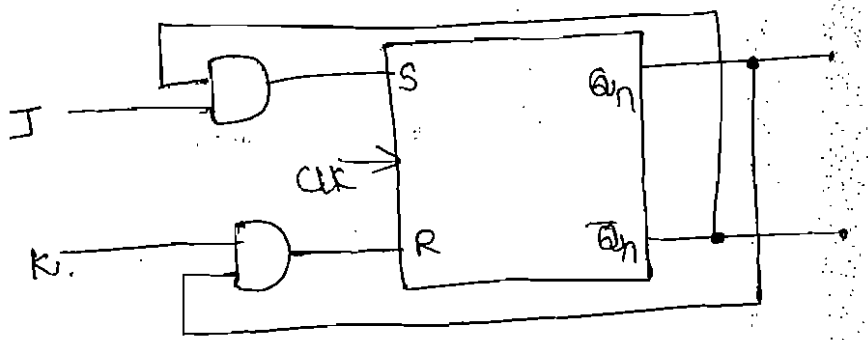
$S = J\bar{Q}_n$

K-map for R

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

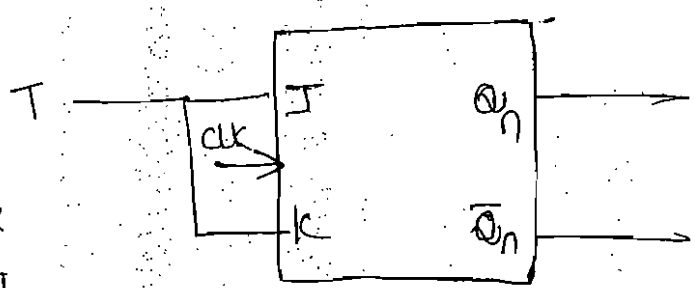
$J = T$

K-map for K

T \ Q_n	0	1
0	X	0
1	X	1

$K = T$

Logic-diagram

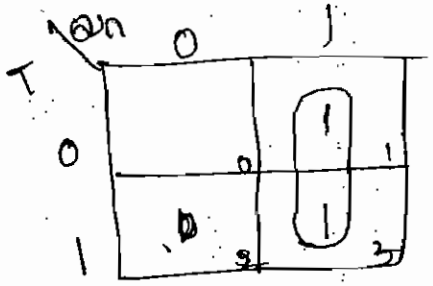


→ conversion of D-flip flop to T flip-flop.

* Available → D-flip flop → excitation table
* Available → T-flip flop → characteristic table.

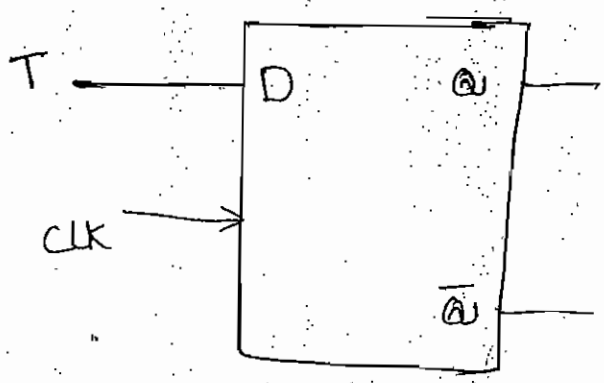
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

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.

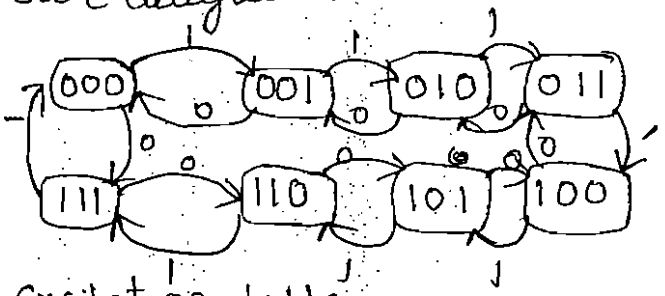
Design steps of Synchronous counters :-

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



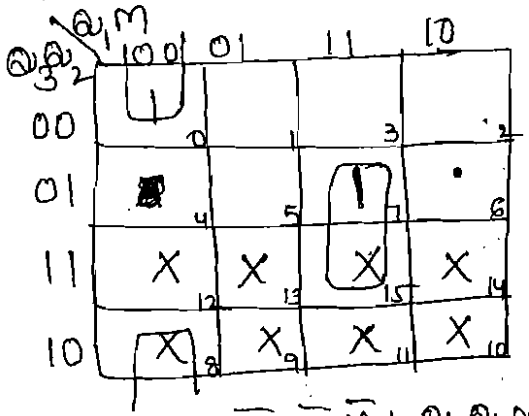
m=0 down counting
m=1 up counting

Step 3 :- Excitation table

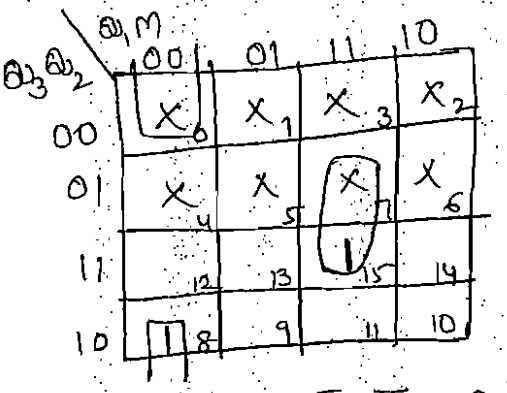
mode (m)	Present State Q_3 Q_2 Q_1	Next state Q_3 Q_2 Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0 0 0	1 1 1	1	X	1	X	1	X
0	0 0 1	0 0 0	0	X	0	X	0	1
0	0 1 0	0 0 1	0	X	X	1	1	X
0	0 1 1	0 1 0	0	X	X	0	X	1
0	1 0 0	0 1 1	X	1	1	X	1	X
0	1 0 1	1 0 0	X	0	0	X	X	1
0	1 1 0	1 0 1	X	0	X	1	1	X
0	1 1 1	1 1 0	X	0	X	0	X	1

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

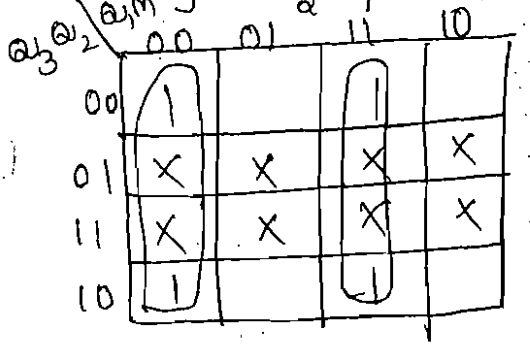
Step 4:- obtain the minimal expression.



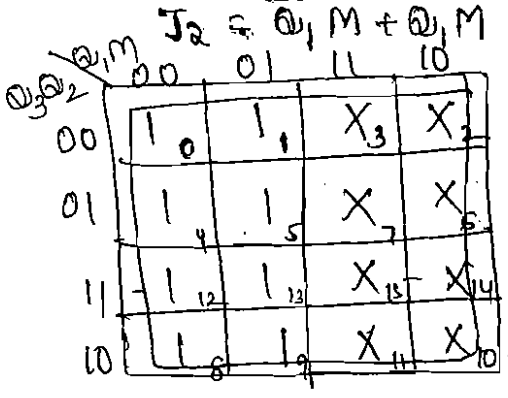
$J_3 = \bar{Q}_2\bar{Q}_1\bar{M} + Q_2Q_1M$



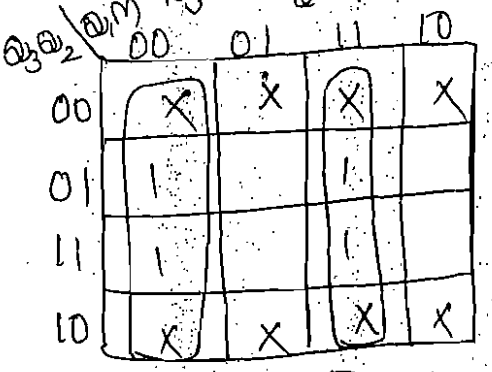
$K_3 = \bar{Q}_2\bar{Q}_1\bar{M} + Q_2Q_1M$



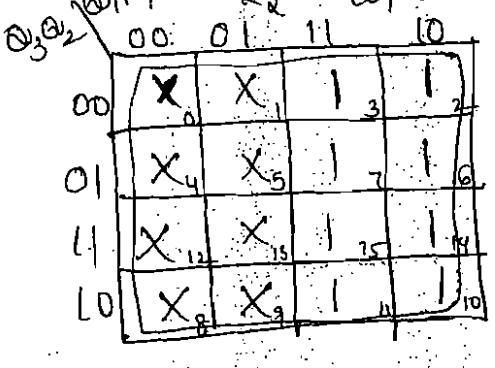
$J_2 = \bar{Q}_1\bar{M} + Q_1M$



$J_1 = 1$

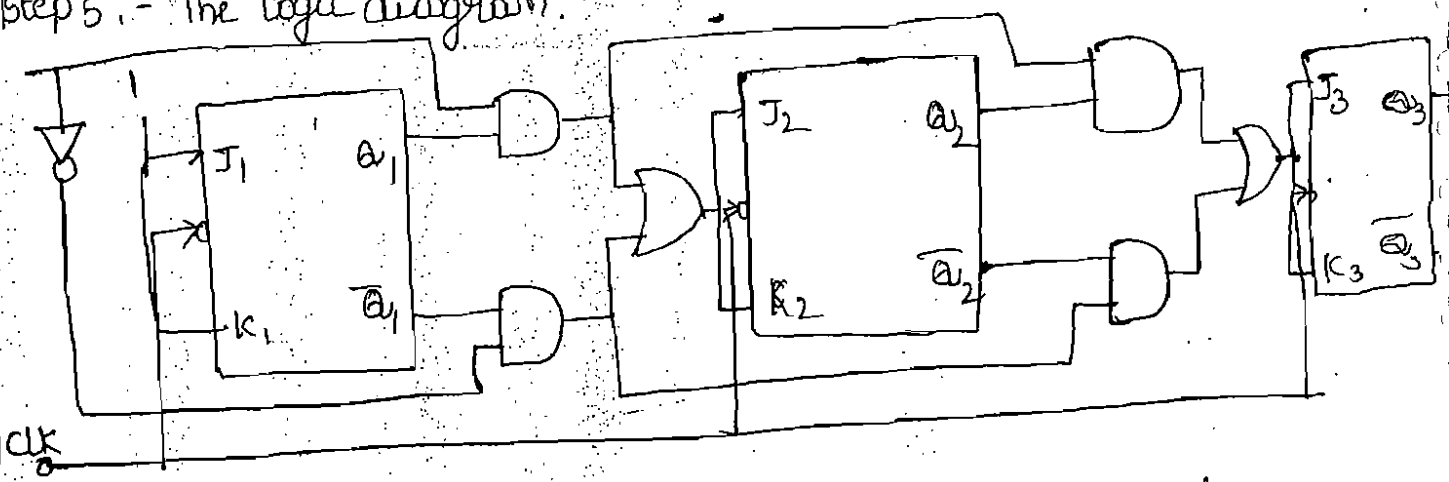


$K_2 = \bar{Q}_1\bar{M} + Q_1M$



$K_1 = 1$

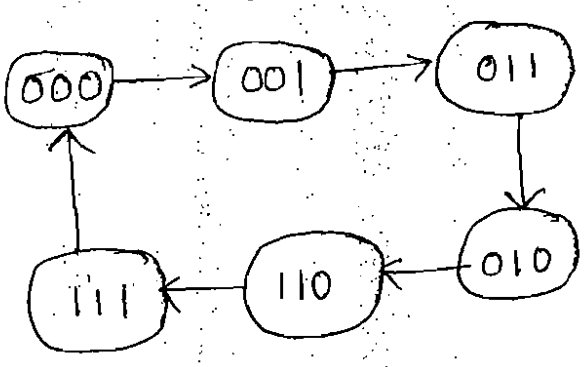
Step 5 :- The logic diagram.



→ Design of a synchronous modulo-6 Gray code counter.

Step 1 :- number of flip flops - 3 FF'S

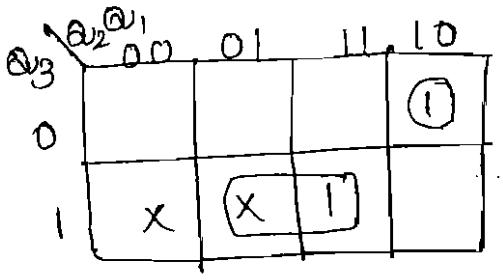
Step 2 :- state diagram



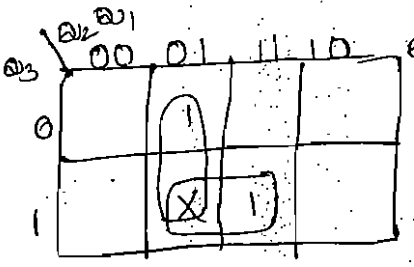
Step 3 :-

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	Q_1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

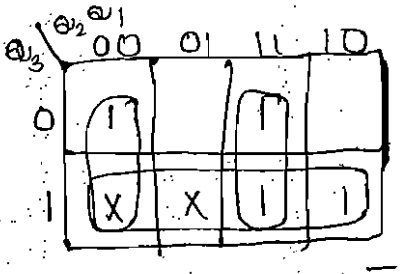
Step 4 :- minimal expressions :-



$T_3 = Q_3Q_1 + \bar{Q}_3Q_2\bar{Q}_1$

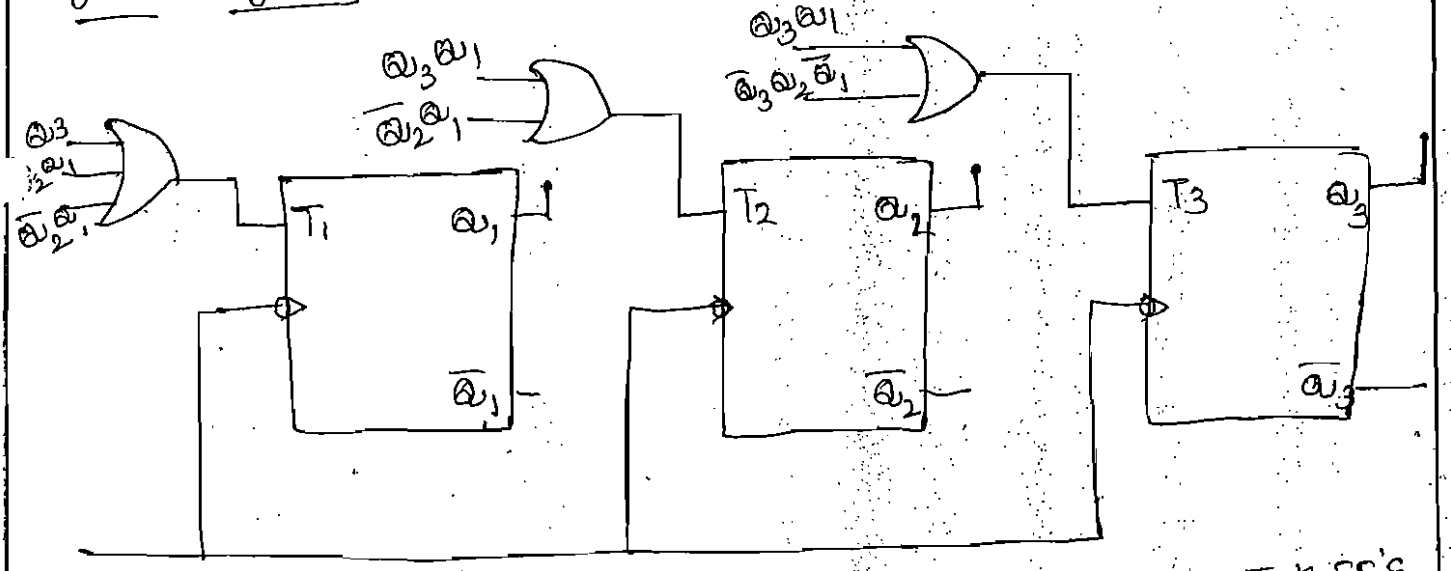


$T_2 = Q_3Q_1 + \bar{Q}_2Q_1$



$T_1 = Q_3 + Q_2Q_1 + \bar{Q}_2\bar{Q}_1$

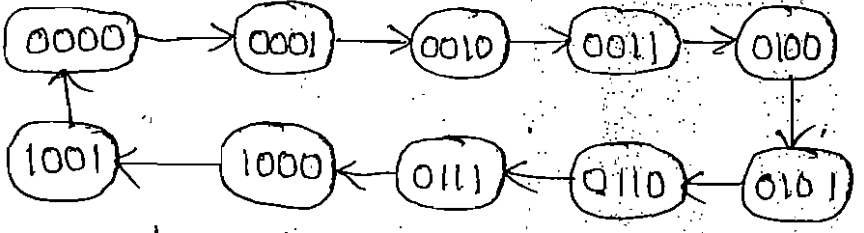
Logic diagram :-



→ Design a synchronous (mod-9) BCD counter using J-K FF's.

Step 1 :- 4 FF's

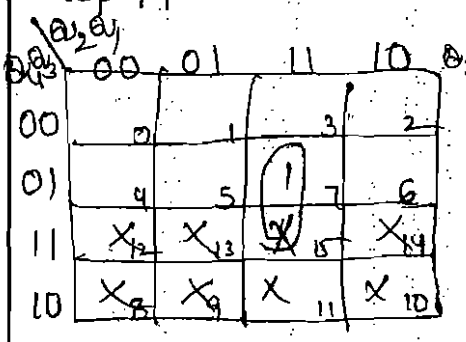
Step 2 :- State diagram.



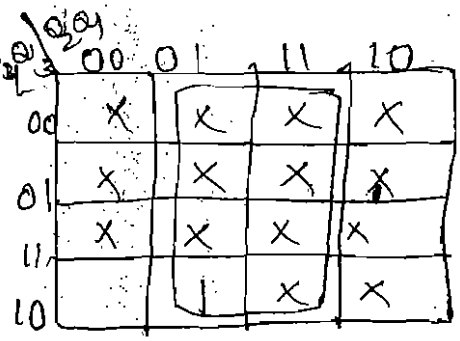
Step 3 :- excitation table.

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	X	1
1 0 0 1	0 0 0 0	X	1	0	X	0	X	X	1

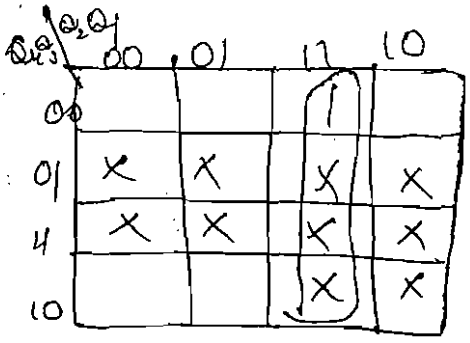
Step 4 :-



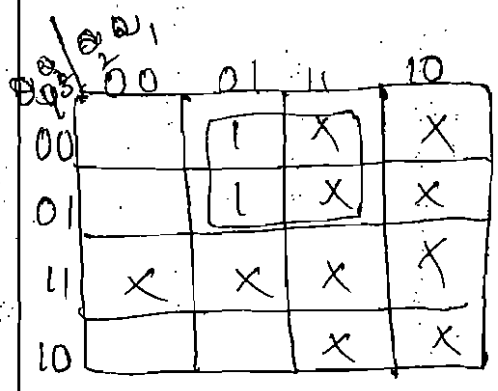
$J_4 = Q_3 Q_2 Q_1$



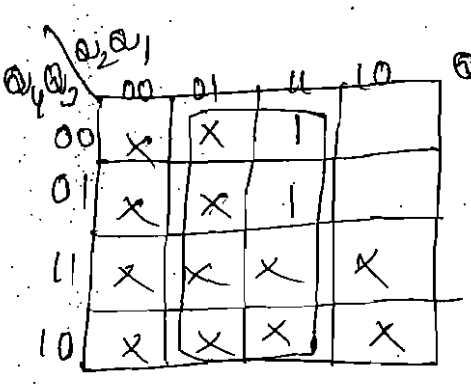
$K_4 = Q_1$



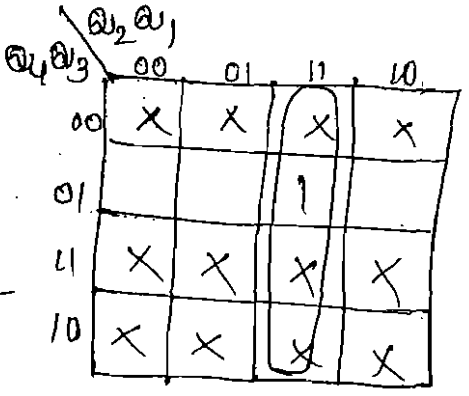
$J_3 = Q_2 Q_1$



$J_2 = \overline{Q_4} Q_1$

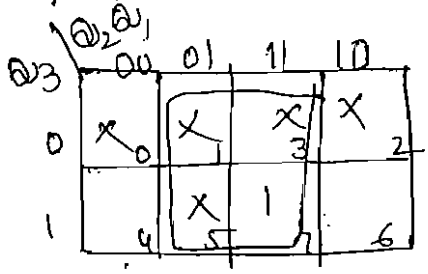


$K_2 = Q_1$

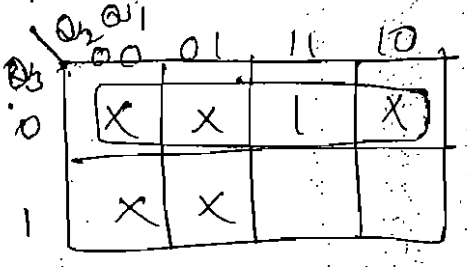


$K_3 = Q_2 Q_1$

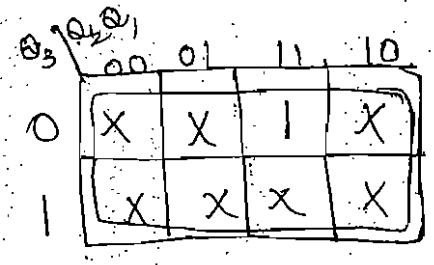
Step 4 :- The minimal expression.



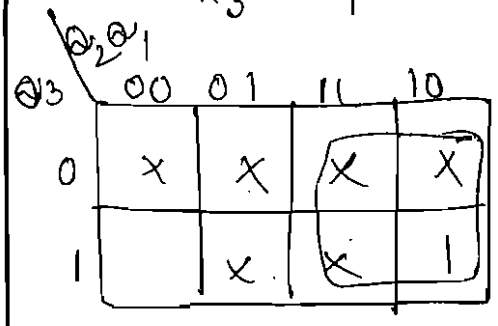
$K_3 = Q_1$



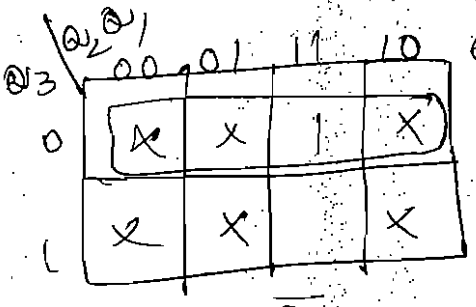
$K_2 = \overline{Q_3}$



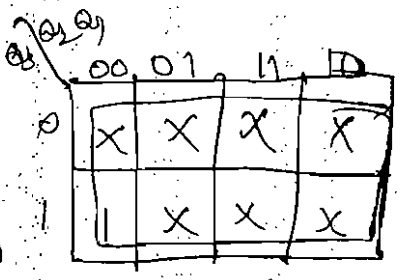
$J_3 = 1$



$J_1 = Q_2$

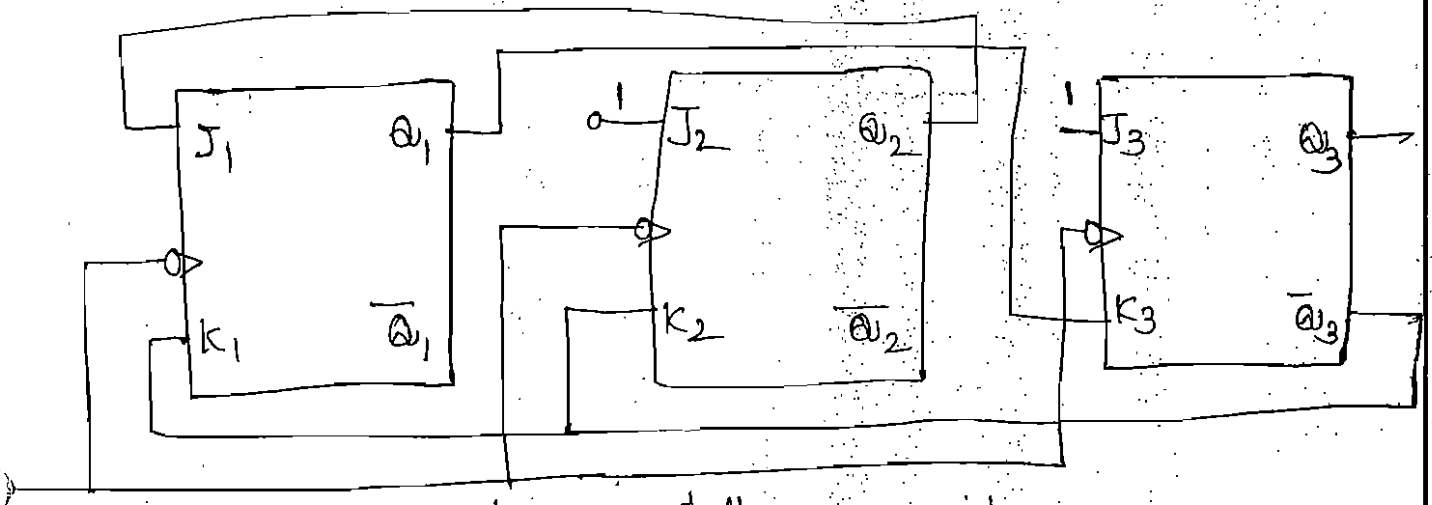


$K_1 = \overline{Q_3}$

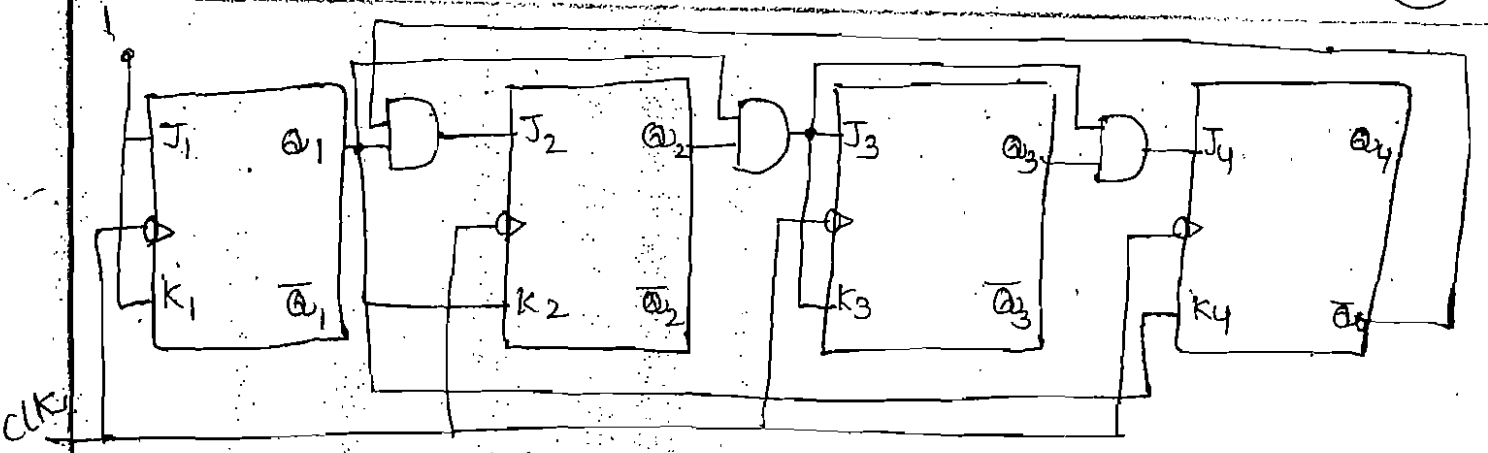


$J_2 = 1$

step 5 :- logic diagram



logic diagram of the J-K counter.

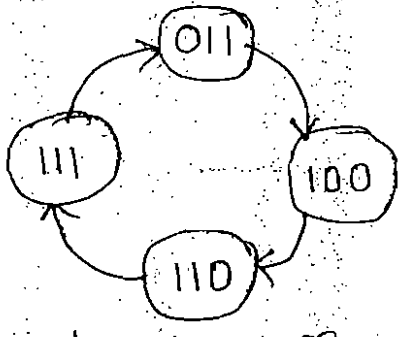


Logic diagram

→ 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.

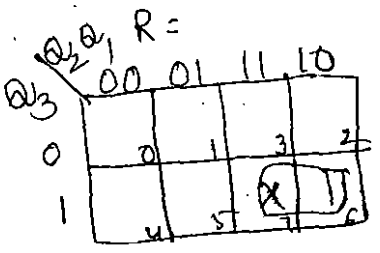
Step 2:- State diagram.

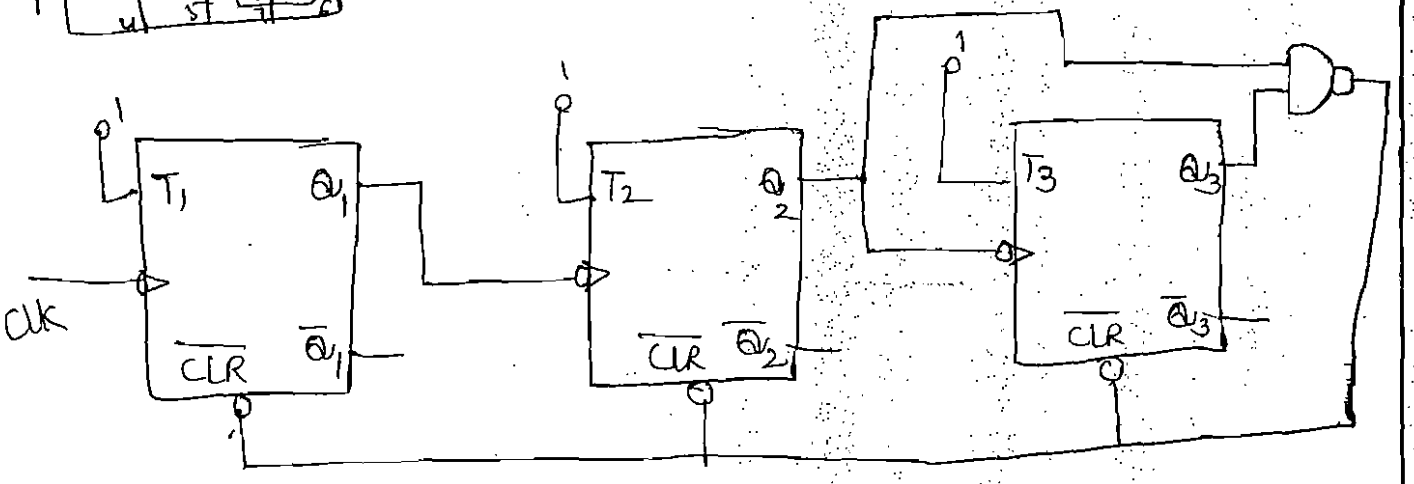


State-diagram

Step 3:- excitation table

Present state			next state			Required inputs					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	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

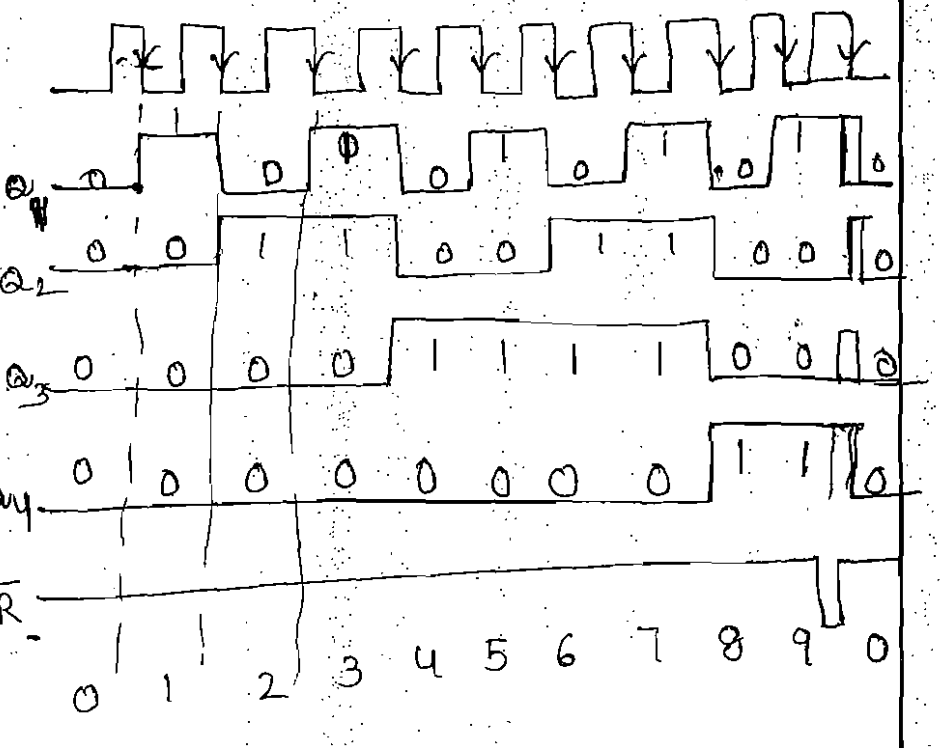




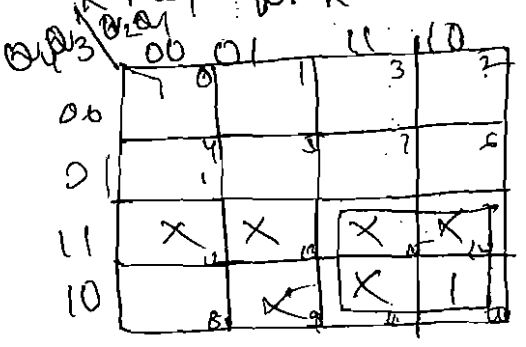
logic diagram.

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.

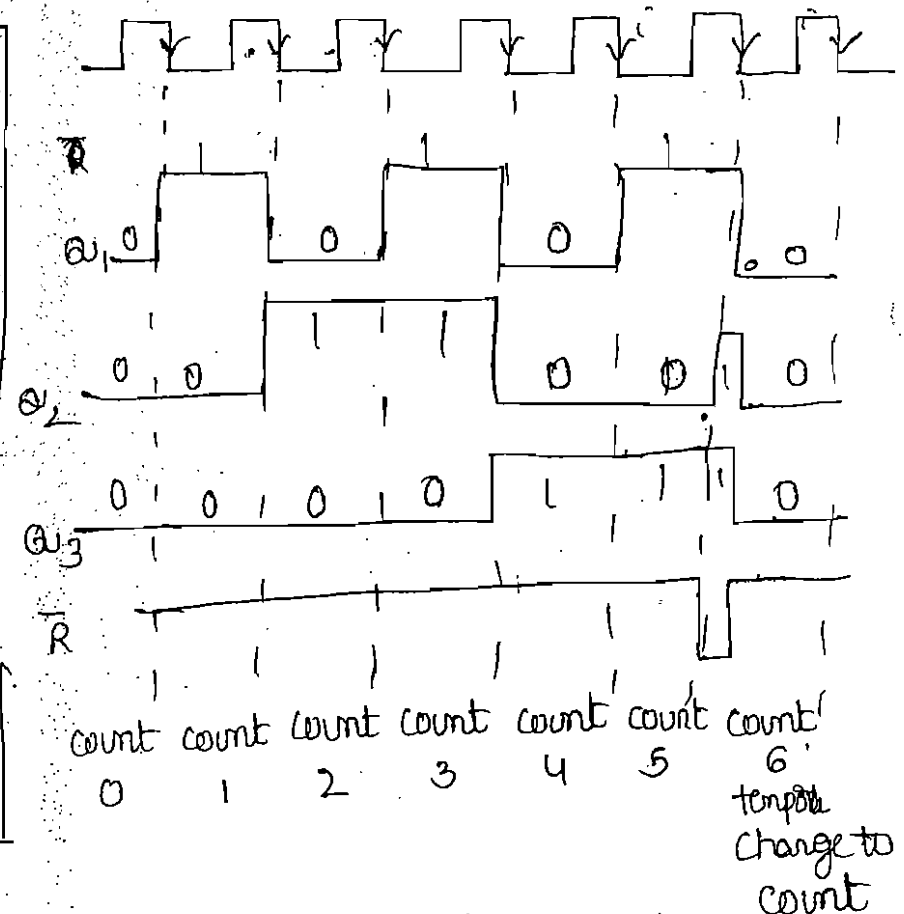
→ For the design, To write a truth table with the present state outputs Q_3, Q_2 and Q_1 as the variables, and reset R as the output.

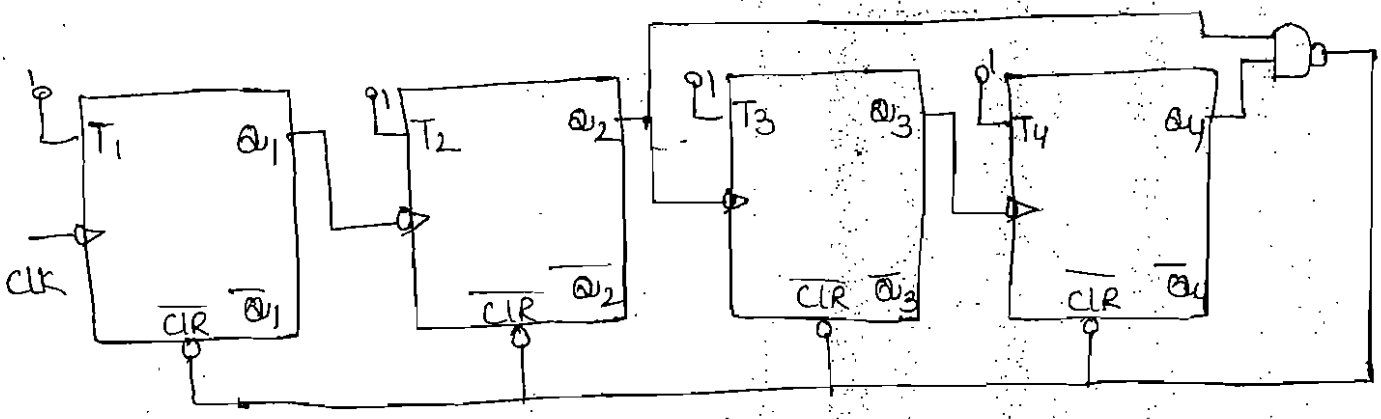
R = 0 for 000 to 101, R = 1 for 110, and R = X for 111.

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.

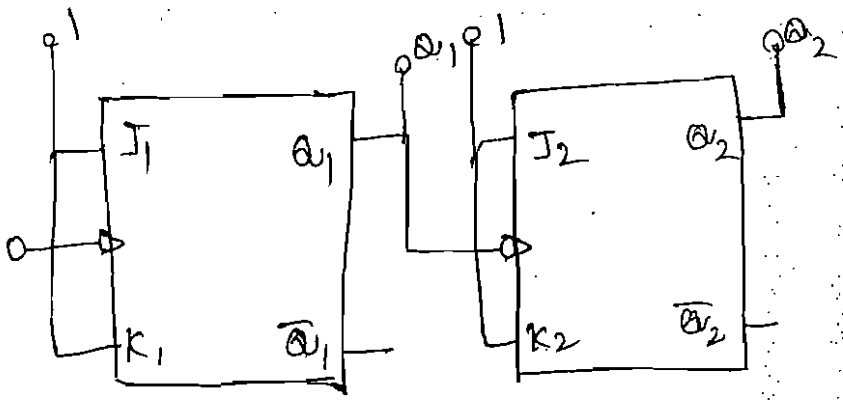
→ Two - bit ripple up- counter using negative edge - triggered flip-flops:

→ 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.

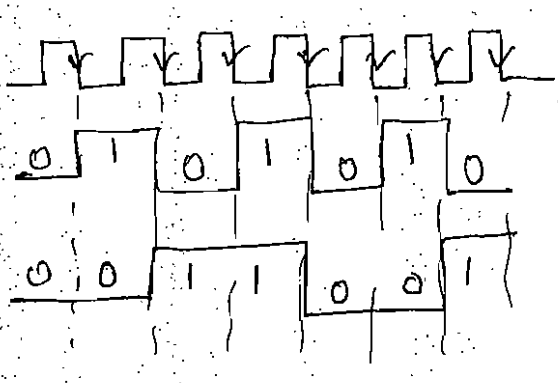
→ So FF_2 is not affected, and hence, the state of the counter after one clock pulse is '$Q_1 = 1$ and $Q_2 = 0$.

→ 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 $Q_1 = 1$, $Q_2 = 1$.

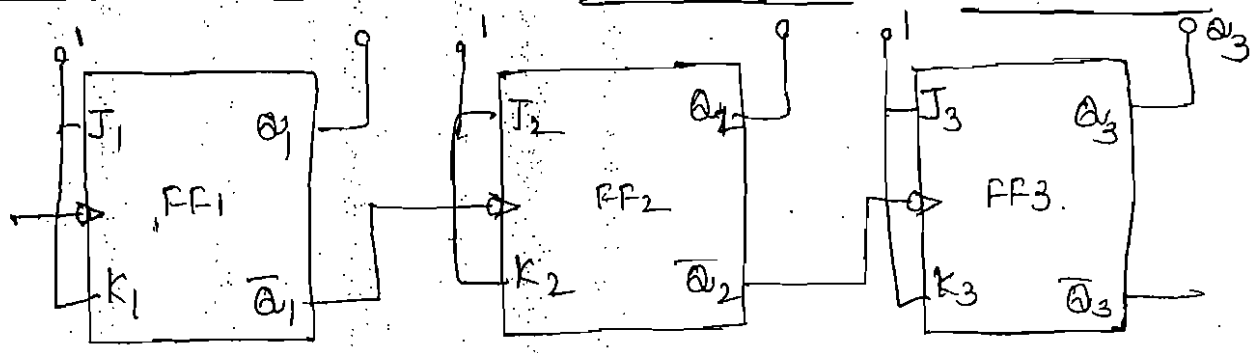


Logic diagram.



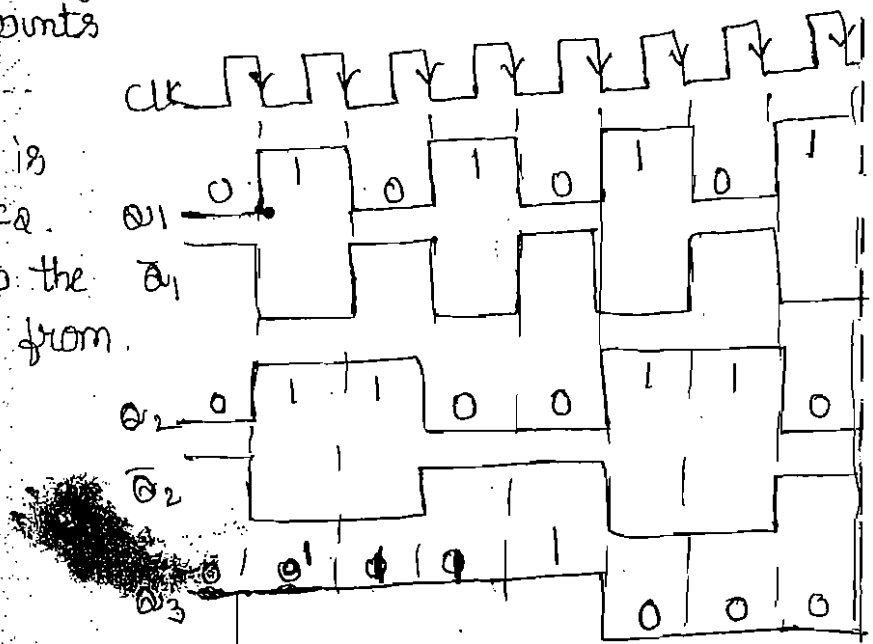
Timing diagram.

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.



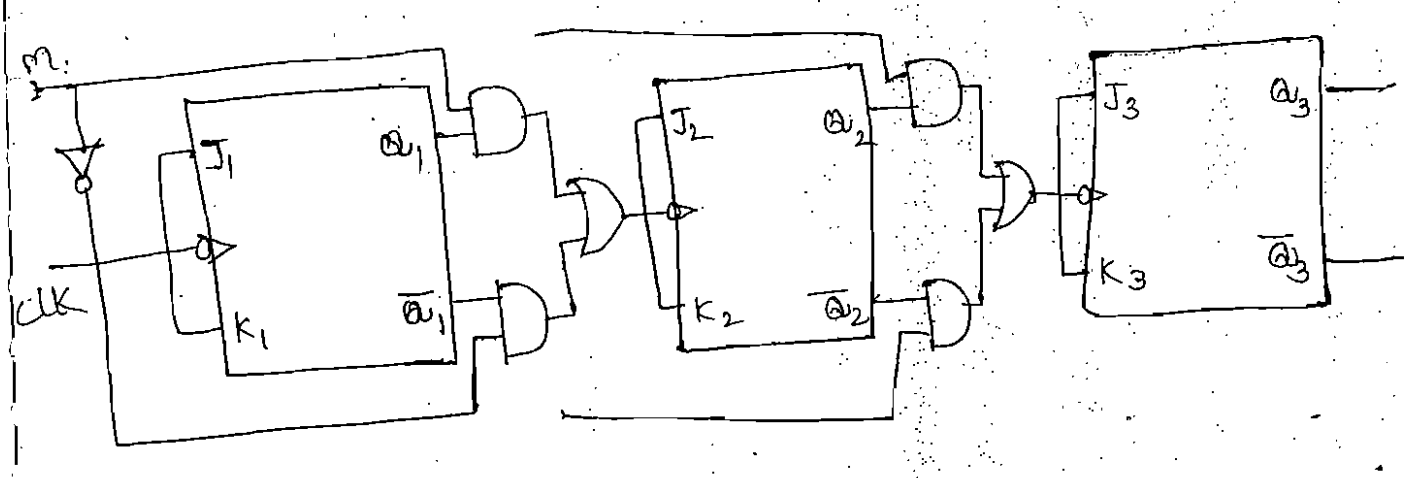
3-bit updown counter using negative edge-triggered flip-flops;-

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.

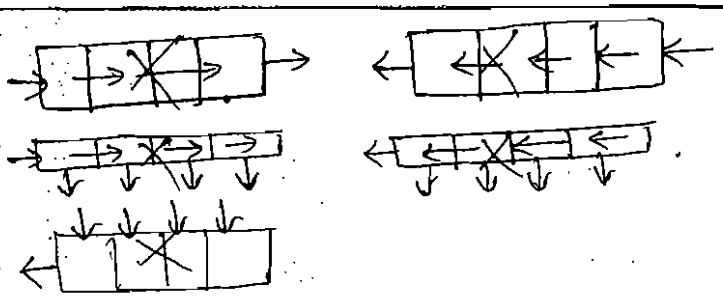
→ where as a counter has a specified sequence of states.

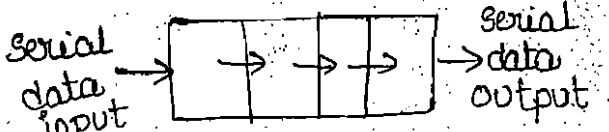
Data-Transmission in shift registers :-

→ 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.

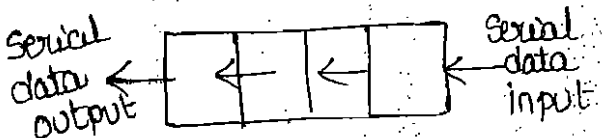
→ Data may be shifted into & out of the register either in serial form & in parallel form. So, there are four types of shift-registers. They are.

1. Serial - in, Serial - out
2. Serial - in, parallel - out
3. parallel - in, serial - out
4. parallel - in, parallel - out



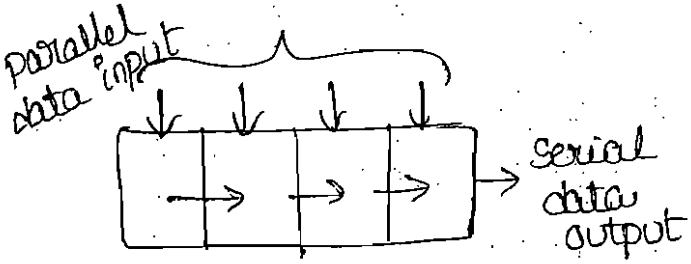


a) Shift-right

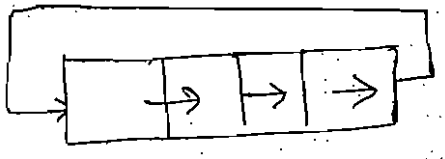


b) Shift - left.

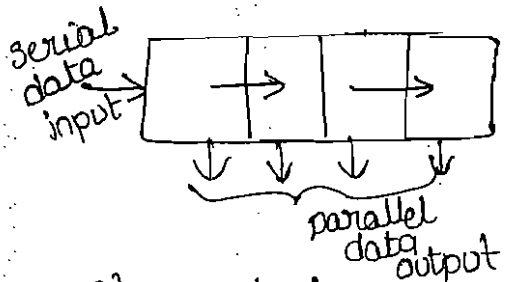
Serial - in, Serial - out



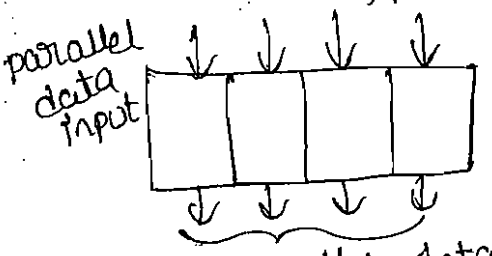
c) parallel - in, Serial output.



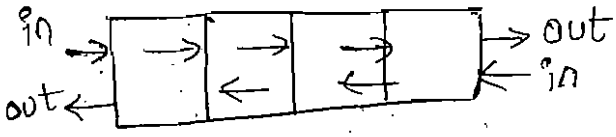
g) Rotate - right Shift register



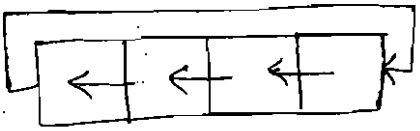
c) Serial - in, parallel out Shift reg



d) parallel - in, parallel - out, shift regis

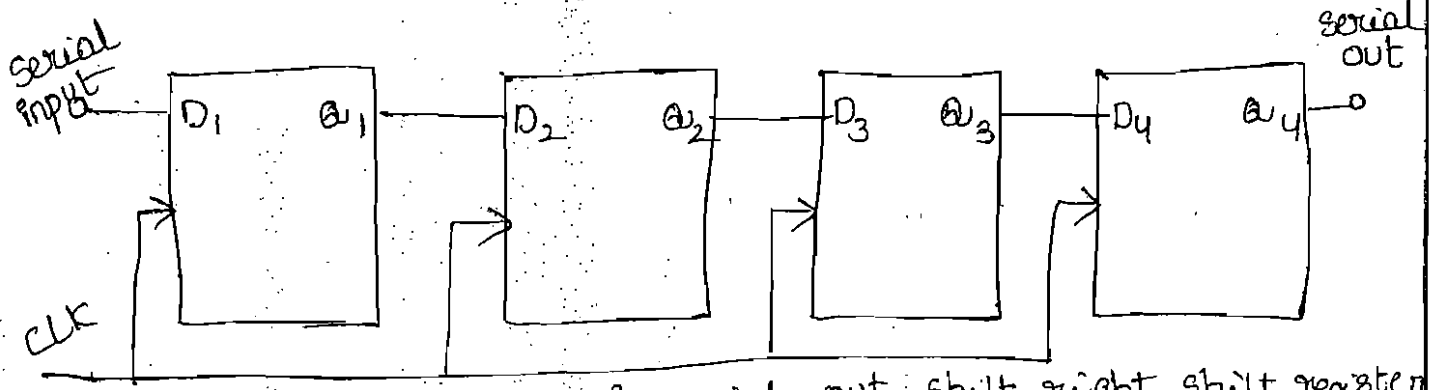


f) bi-directional shift register.

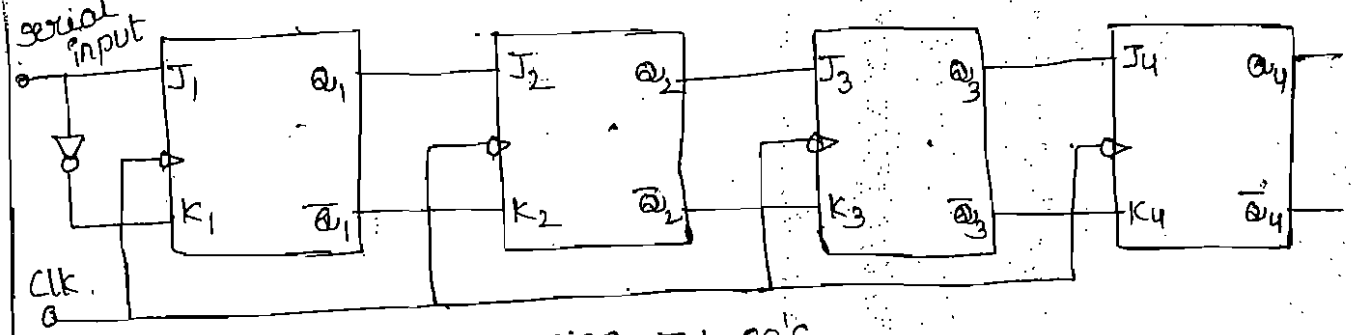


h) Rotate - left Shift register.

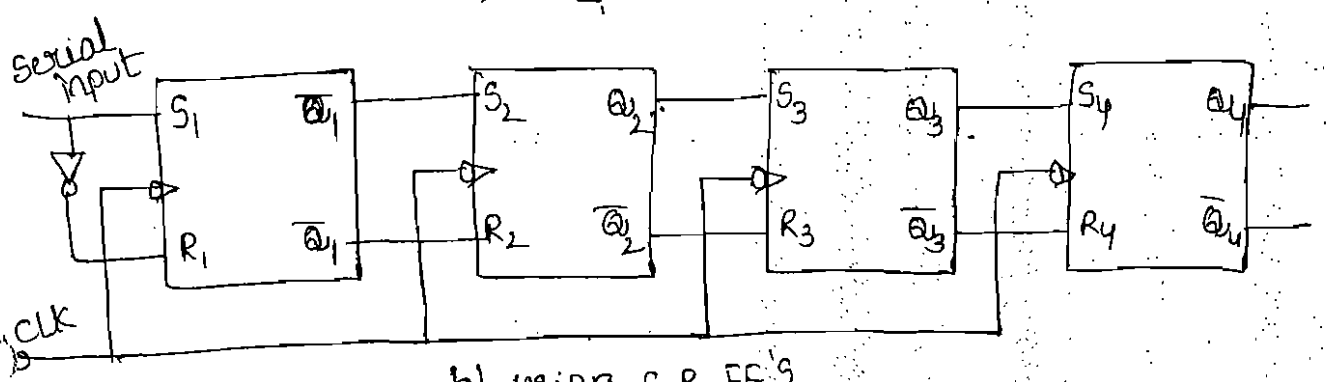
→ Serial - in, Serial - out Shift register :-



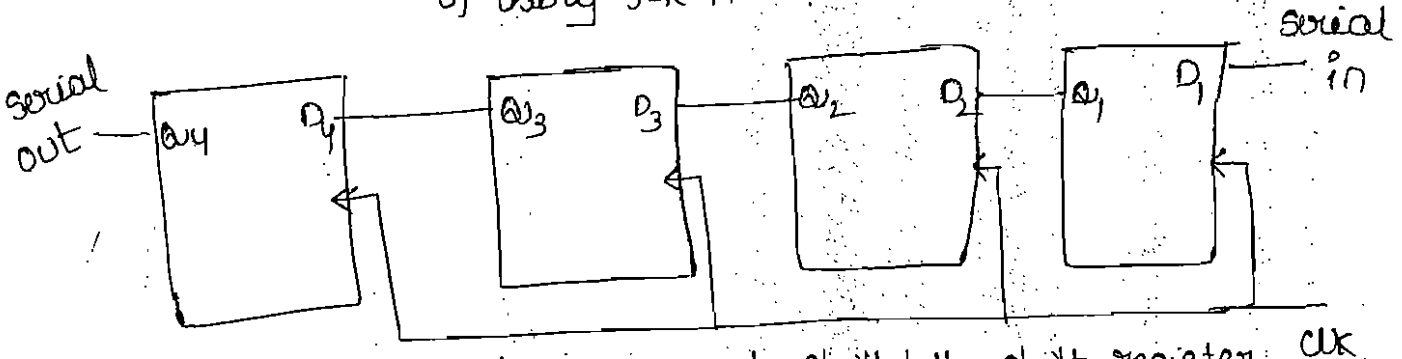
4-bit Serial - in, Serial - out, Shift right Shift register



a) using J-K FF'S.

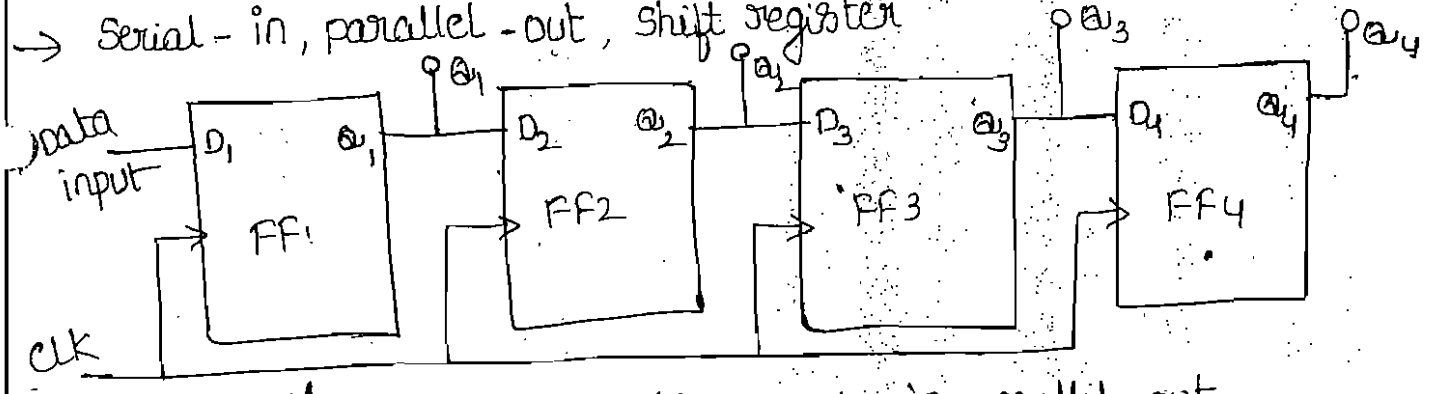


b) using S-R FF'S.

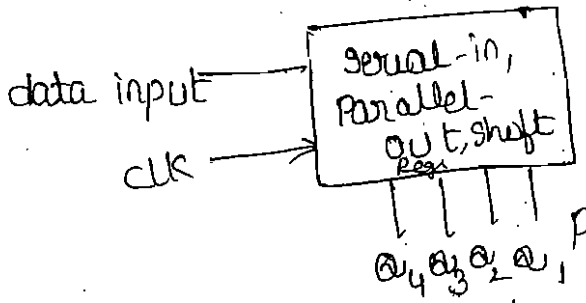


A 4-bit serial-in, serial-out, shift left shift register.

→ Serial-in, parallel-out, shift register

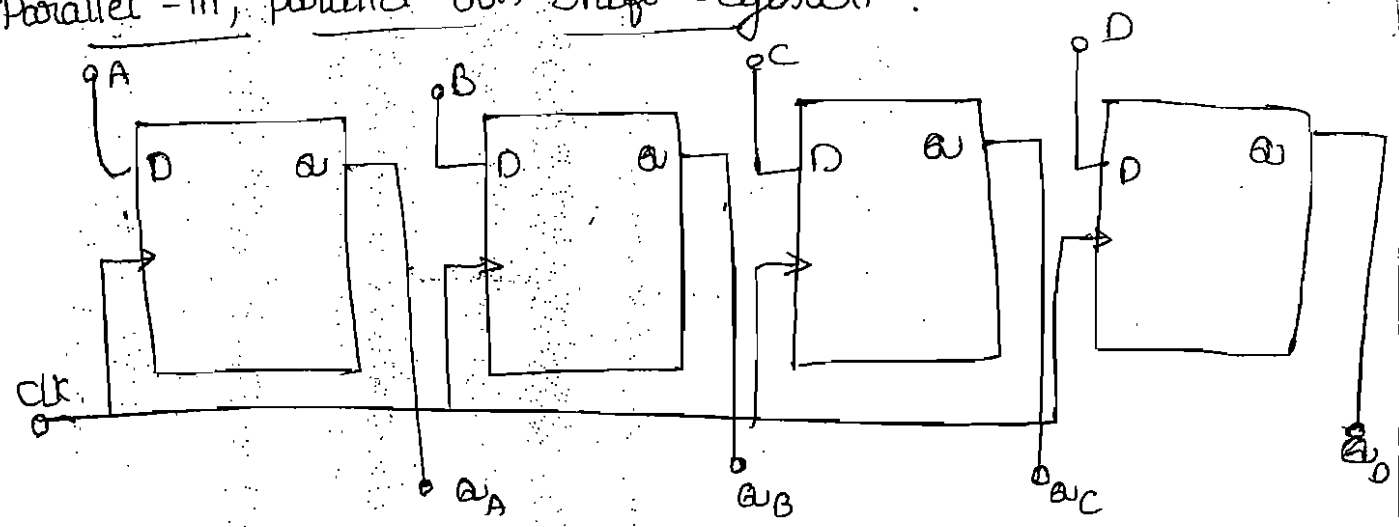


Logic diagram for serial-in, parallel-out



logic symbol.

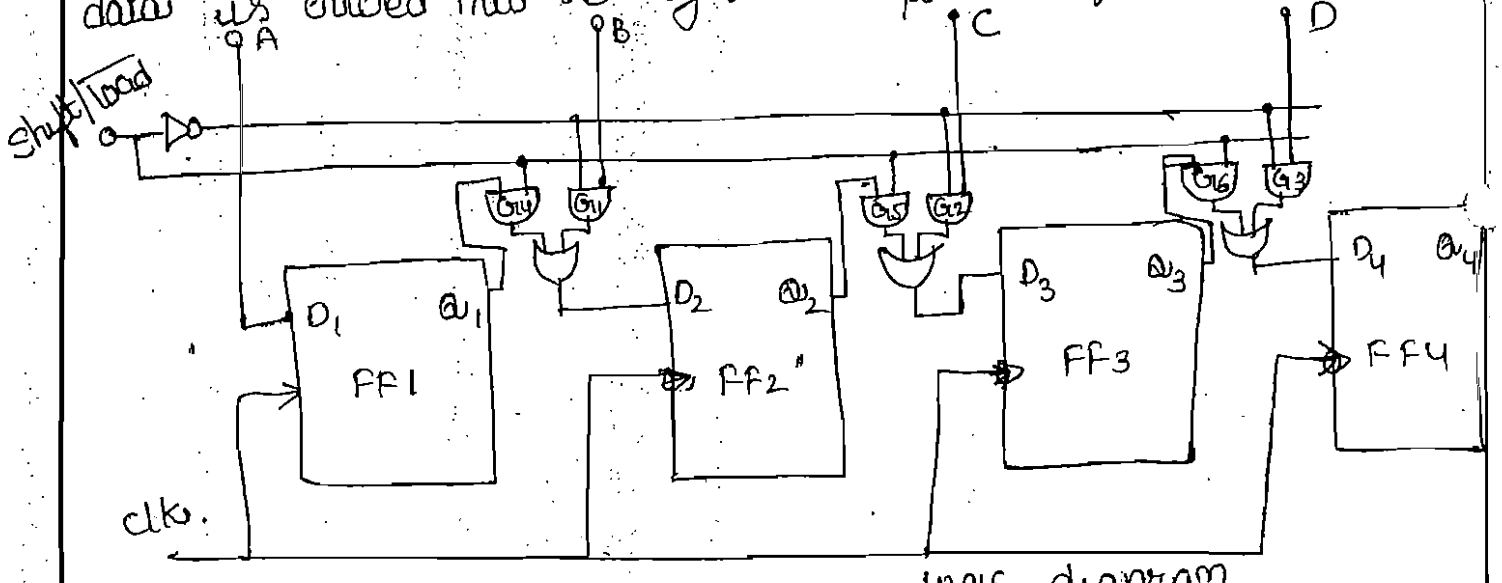
→ Parallel - in, parallel - out, shift - register :-



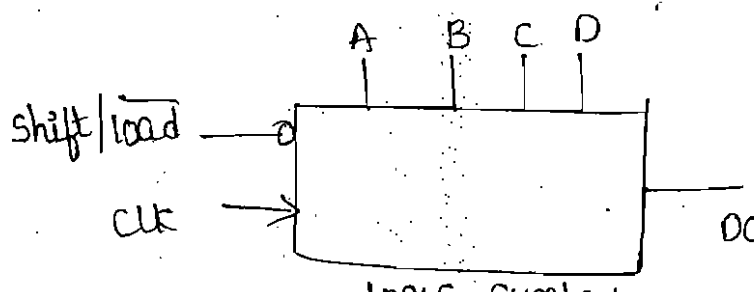
Logic diagram of a 4-bit parallel- in, parallel -out

→ 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.

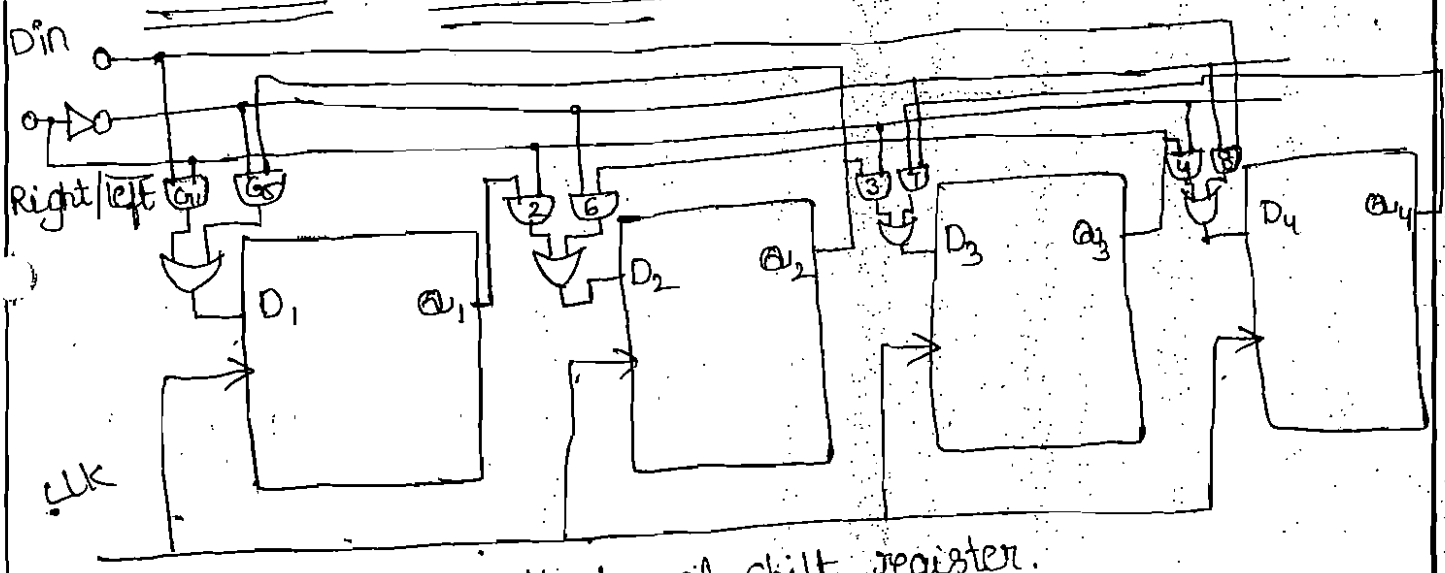
→ when Shift / $\overline{Load}$ line is high, gates G_1, G_2 and G_3 are disabled, but G_4, G_5, G_6 are enabled allowing the data bits to shift-right from one-stage to the next.

→ 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.

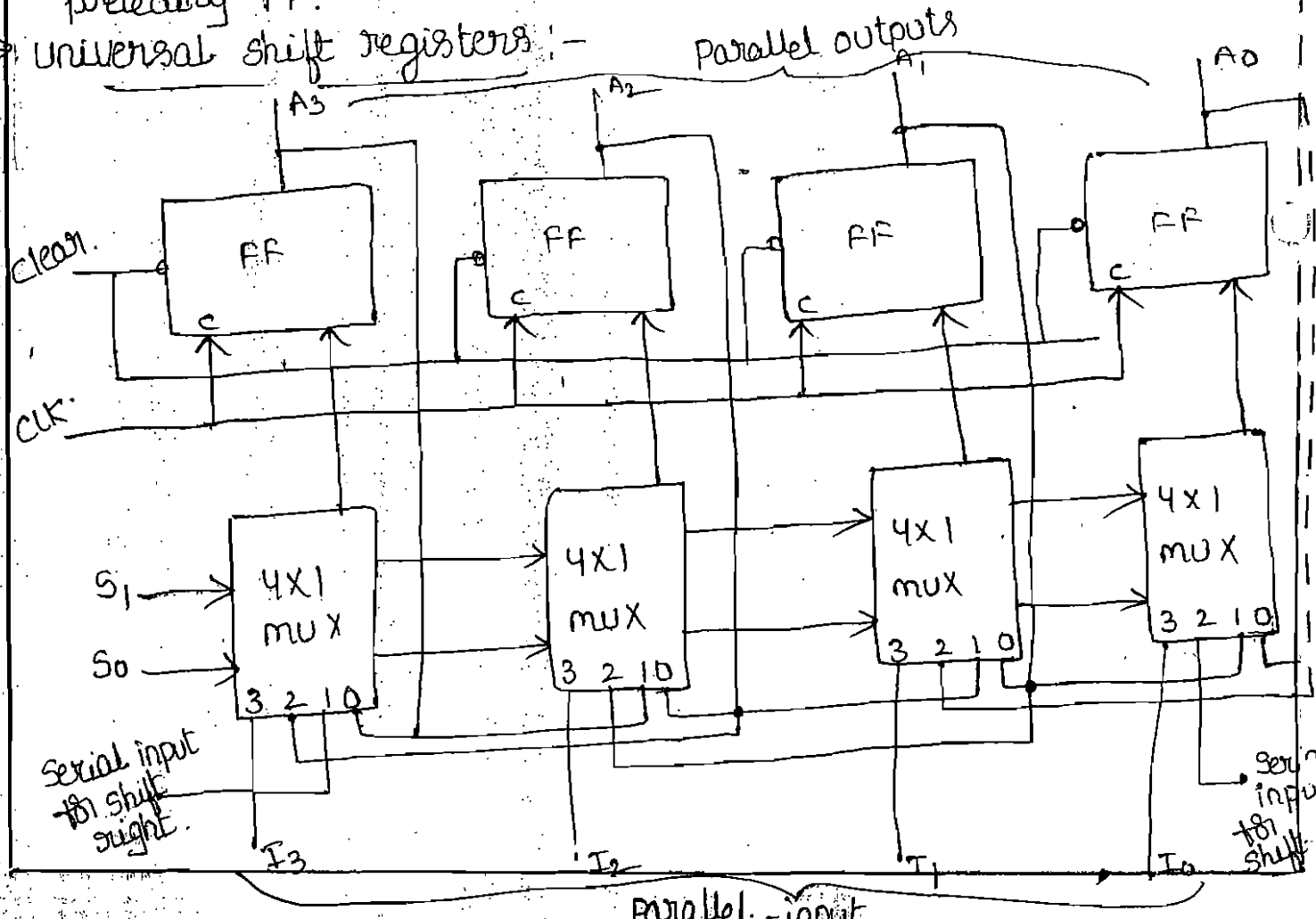
→ when Right / $\overline{\text{left}}$ is a 0, the logic circuit works as a shift-left shift register.

→ 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.

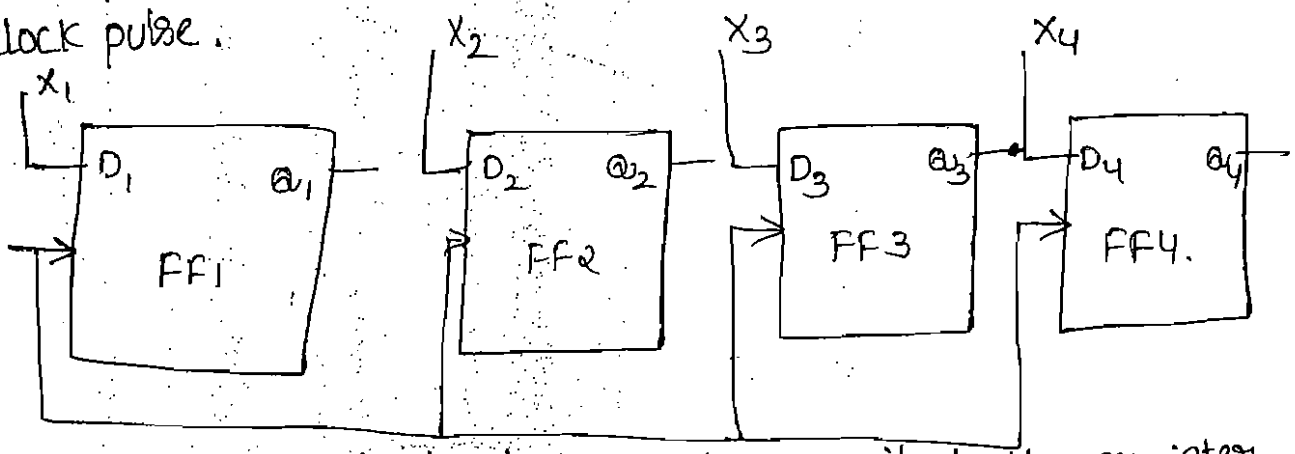
→ Finally $S_1S_0=11$ the binary information on the parallel

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.

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

$$Q = X.$$

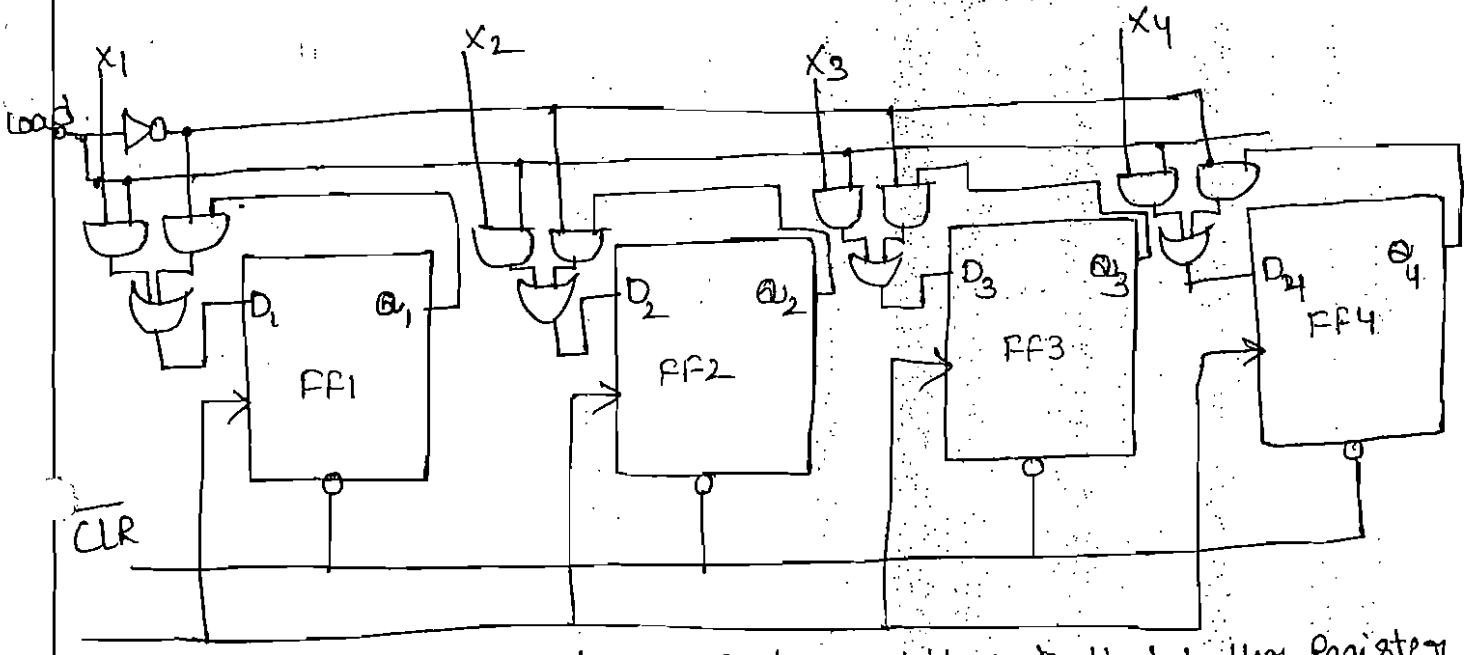
Controlled buffer Register :-

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.

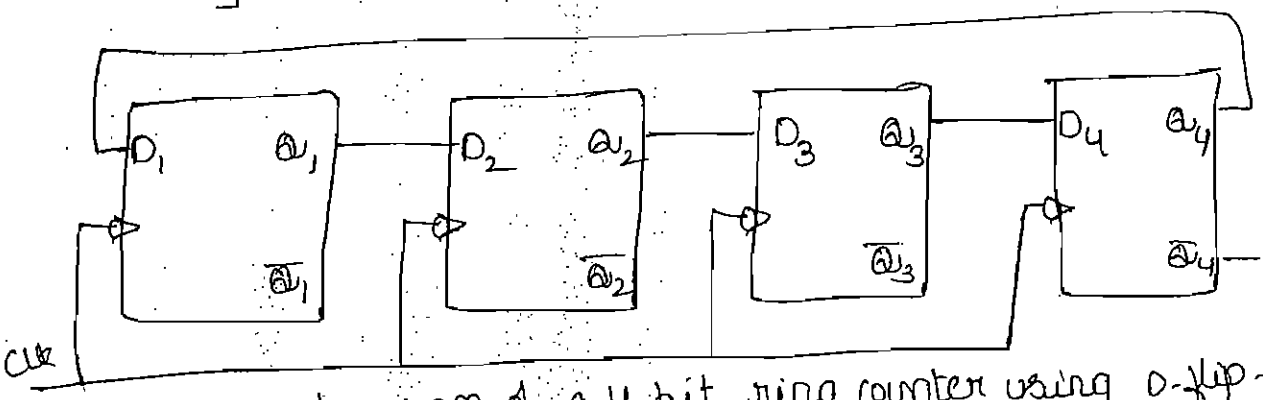
→ Shift Register counters :-

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

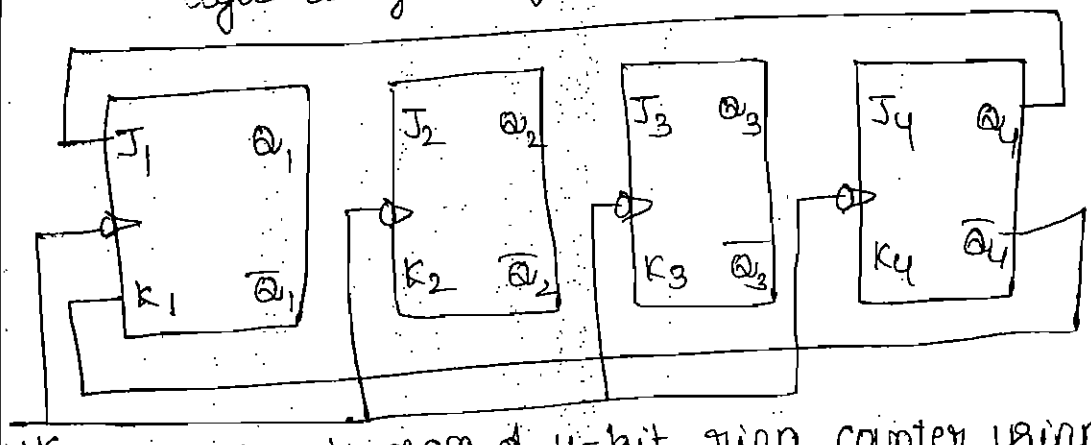
counter is the ring counter [simple ring counter)
twisted ring counter (Johnson counter & the switch-tail

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.

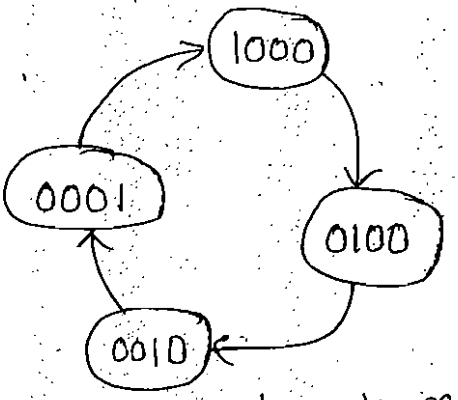


Logic-diagram of 4-bit ring counter using J-K flip-flops

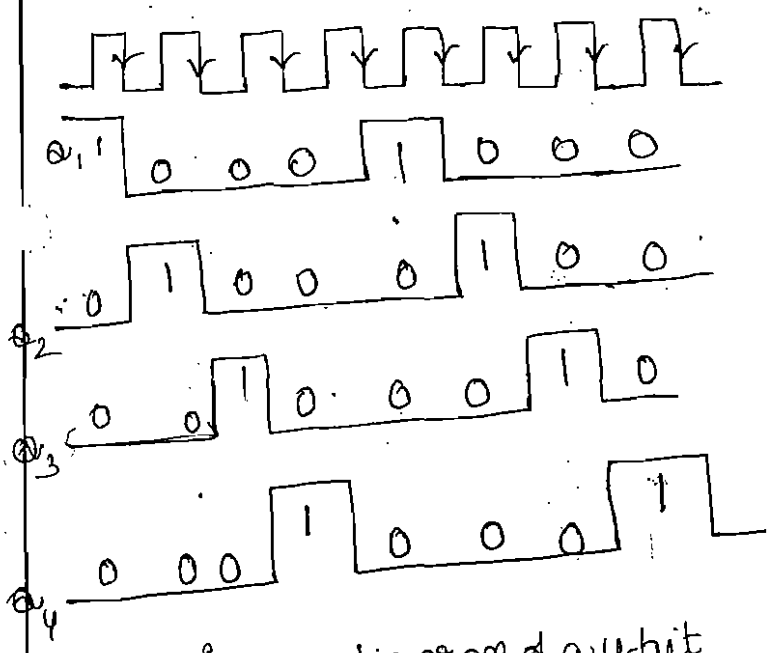
In most instances, only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially, the first FF is present to a 1. So, the initial state is 1000, that is $Q_1 = 1, Q_2 = 0, Q_3 = 0$

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.



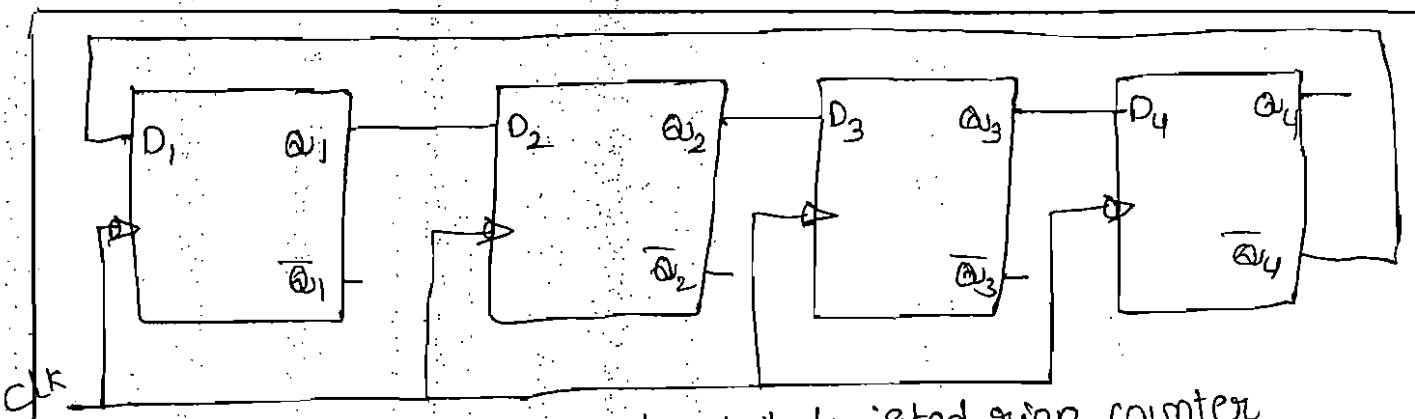
Timing diagram of a 4-bit ring counter.

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.

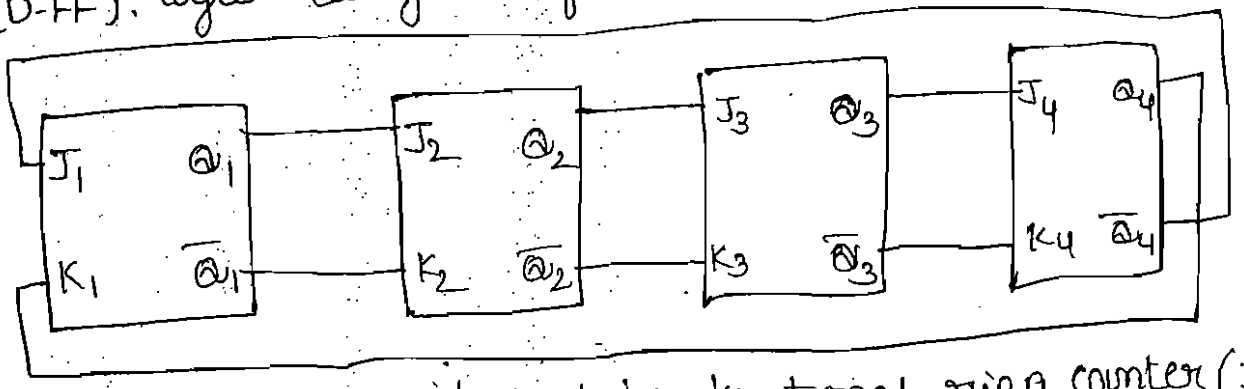
Sequence table.

Twisted Ring Counter :-

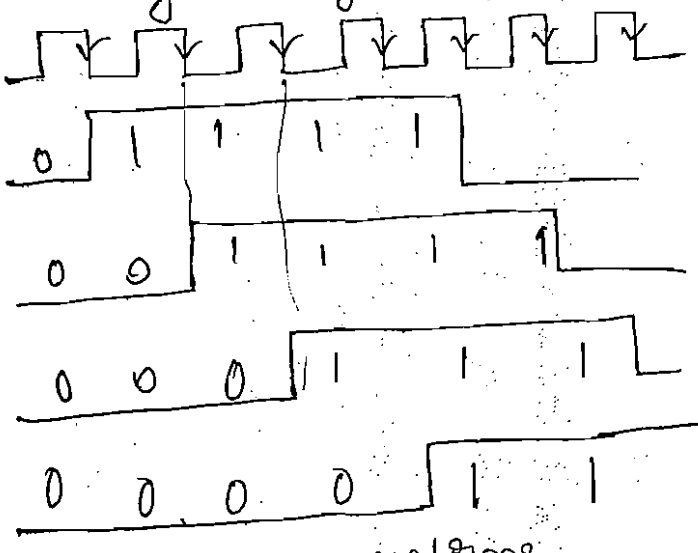
The counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. The Q output of each stage is connected to the D input of the next stage, but the $\bar{Q}$ output of the last stage is connected to the D input of first stage, therefore, the name twisted ring counter.



(D-FF). logic - diagram of 4-bit twisted ring counter



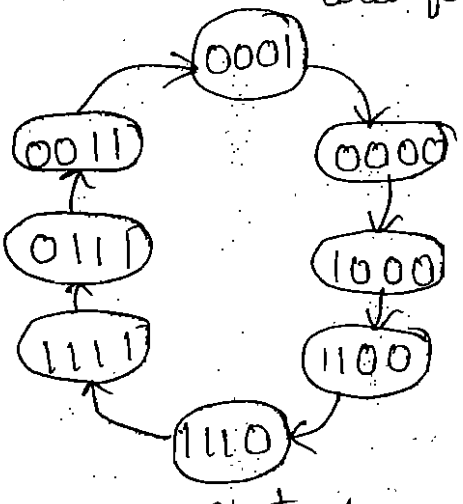
logic - diagram of 4-bit bi-directional ring counter (J-K FF)



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.

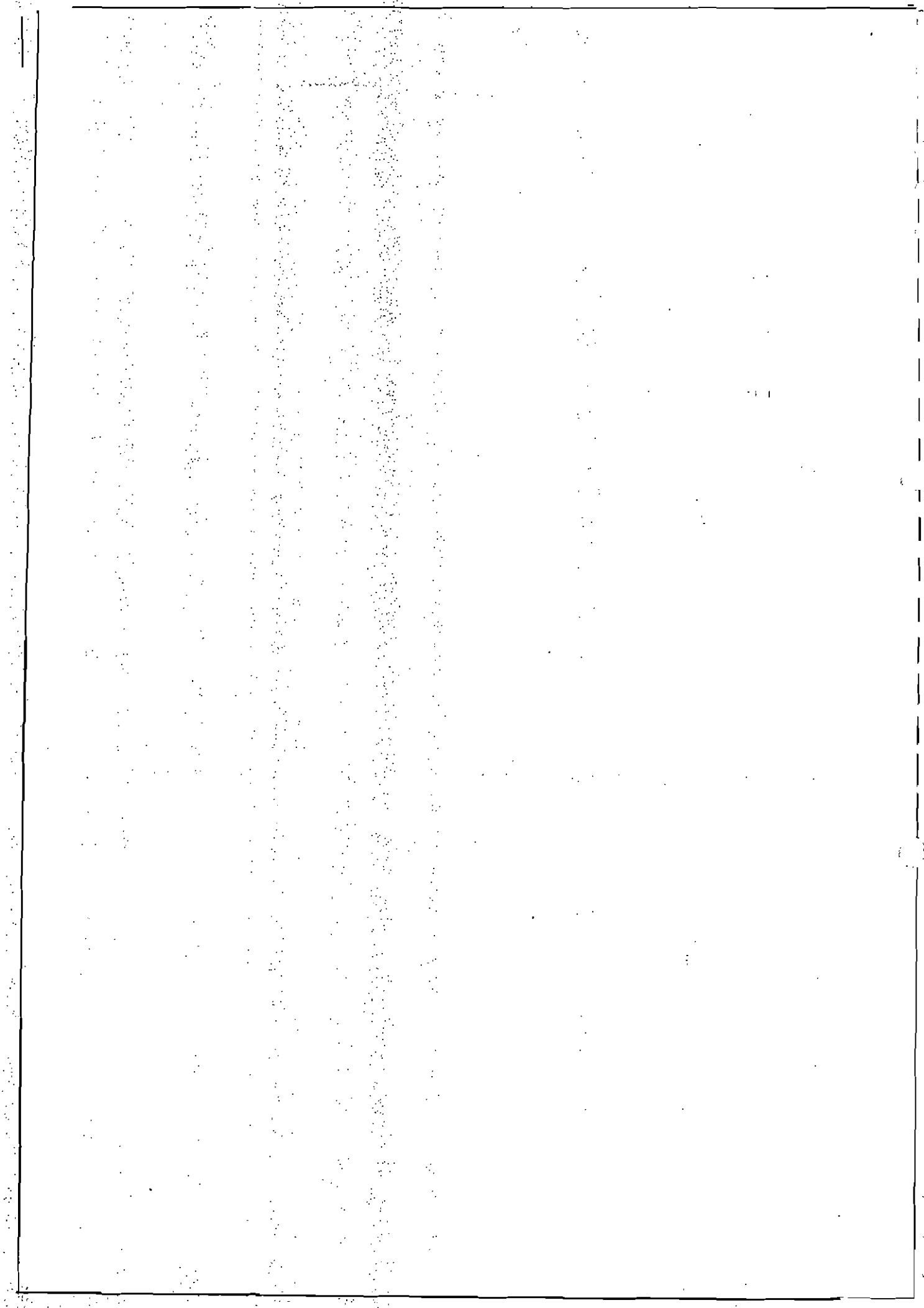
→ An n FF Johnson counter can have 2n unique states and can count up to 2n pulses. So, it is a mod-2n counter.

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.

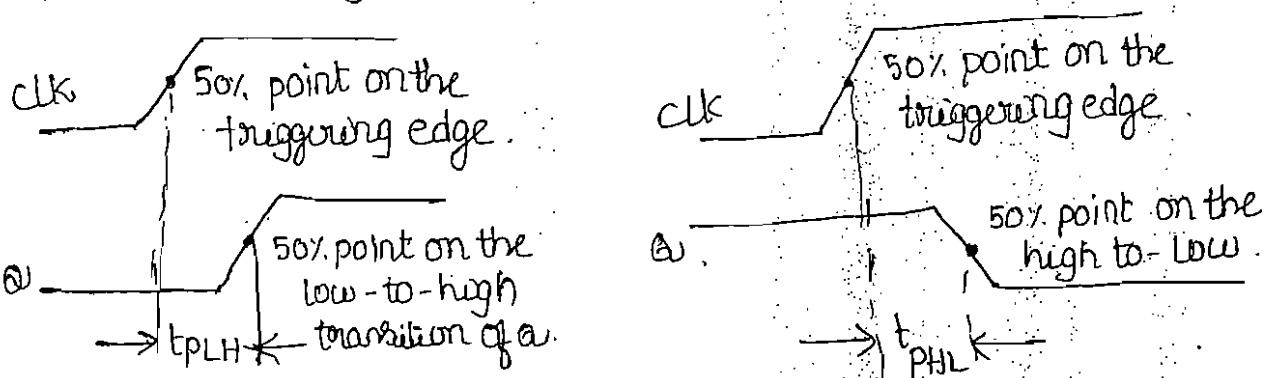


Flip-flop operating characteristics :-

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.

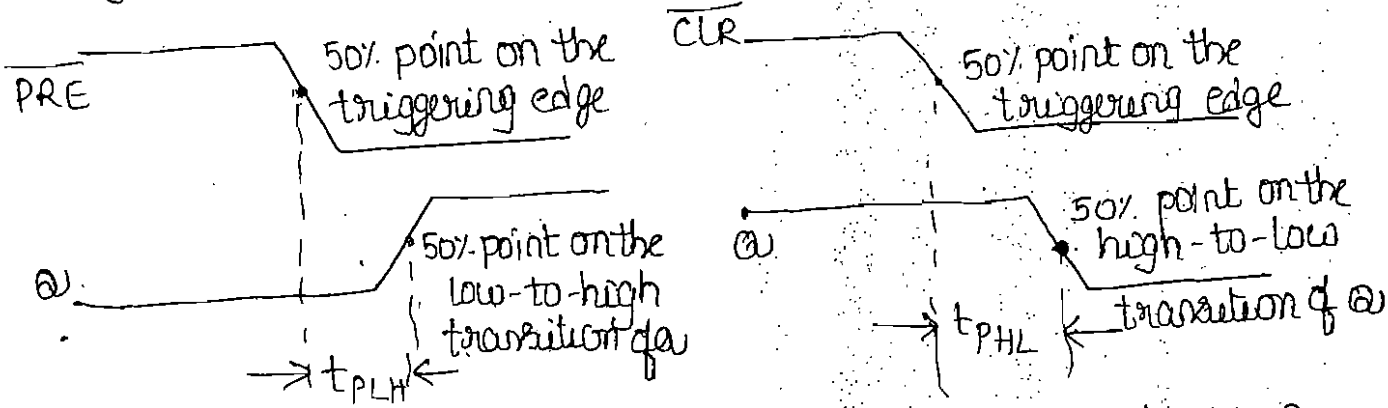
→ propagation delay t_{PHL} measured from the triggering of the clock pulse to the high-to-low transition of the output.



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



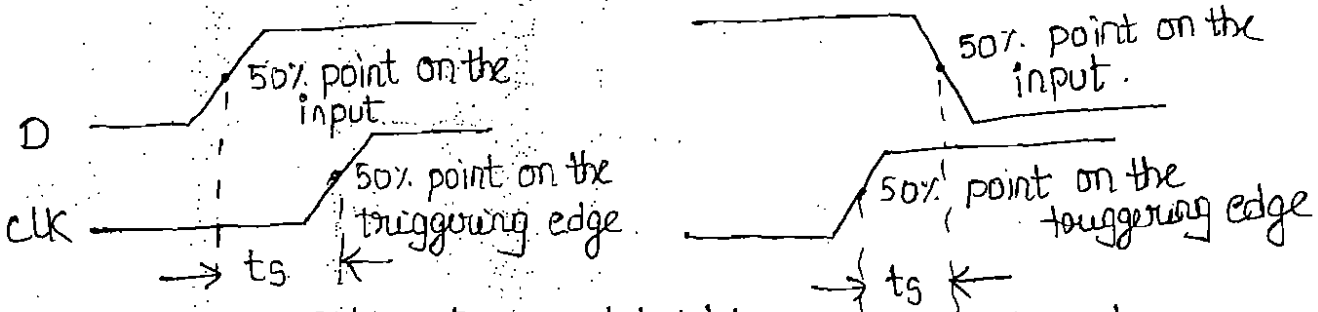
propagation delays t_{PLH}, t_{PHL} w.r.t. PRESET and CLEAR

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

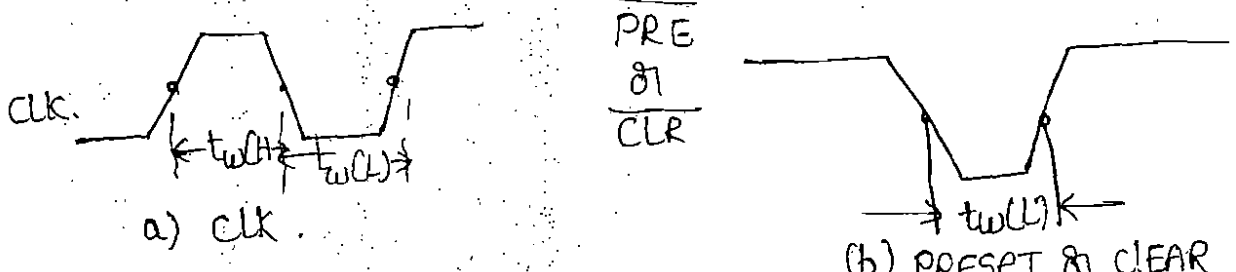
hold time :- The hold time (t_h) is the minimum time for which the control signals need to be maintained constant at the input terminals of the flip flop, after the arrival of the triggering edge of the clock pulse.

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

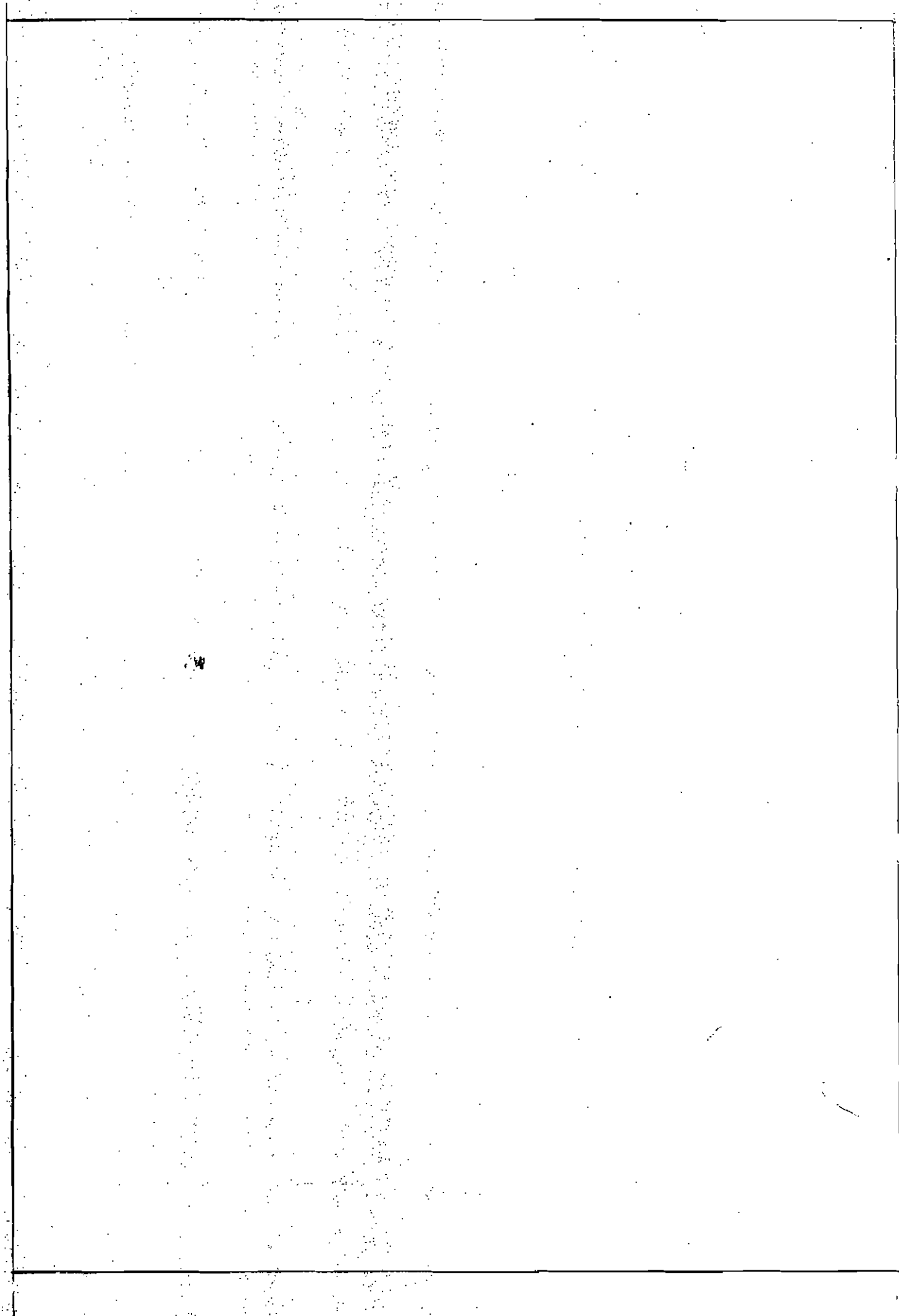
I_{CC} = current

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.

$$P_{TOT} = N \cdot V_{CC} \cdot I_{CC}$$

$$= (N \cdot P) \text{ mw}$$



14. Tabulate the PLA programmable table for the four Boolean functions listed below.

$$A(x,y,z) = \sum m(0,1,2,4,6)$$
$$B(x,y,z) = \sum m(0,2,6,7)$$
$$C(x,y,z) = \sum m(3,6)$$
$$D(x,y,z) = \sum m(1,3,5,7)$$

15. Tabulate the PLA programming table for the four Boolean functions listed below. Minimize the number of product terms.

$$A(x,y,z) = \sum(1,2,4,6)$$
$$B(x,y,z) = \sum(0,1,6,7)$$
$$C(x,y,z) = \sum(2,6)$$
$$D(x,y,z) = \sum(1,2,3,5,7)$$

16. Derive the PLA programming table for the combinational circuit that squares a 3-bit number. Minimize the number of product terms.
17. List the PLA programming table for the BCD to excess 3 code converter whose Boolean functions are simplified.
18. List the PAL programming table for the BCD to excess 3 code converter whose Boolean functions are simplified.
19. The following is a truth table of a 3-input, 4-output combinational circuit. Tabulate the PAL programming table for the circuit and mark the fuse map in a PAL diagram.

Inputs			Outputs			
x	y	z	A	B	C	D
0	0	0	0	1	0	0
0	0	1	1	1	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1

Chapter 6

Synchronous Sequential Logic Circuit

6.1 Introduction

A block diagram of a sequential circuit is shown in Fig.6.1. It consists of a combinational circuit to which storage elements are connected to form a feedback path. The storage elements are devices capable of storing binary information. This binary information stored in these elements at any given time define the state of the sequential circuit at that time.

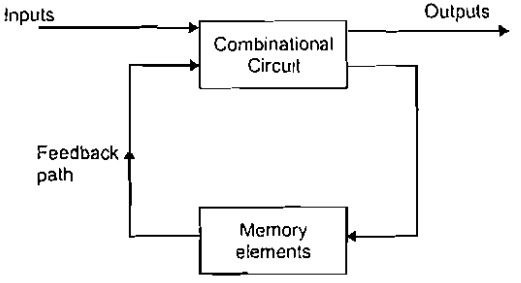


Fig 6.1 Block diagram of sequential circuits

The sequential circuit receives the binary information from external inputs. These inputs and the present state of memory elements determine the binary value of the outputs of the circuit. They also determine the condition for changing the state in memory elements. Thus, the next state of the memory elements is also a function of the external inputs and the present state.

6.1.1 Mealy Model Sequential Circuit

Fig 6.2 shows the clocked synchronous sequential Mealy machine. The output of mealy machine is the function of present inputs and present state (Flip flop outputs). If X is input, Q_n is the present state and the next state is $Q_{(n+1)}$, the output of Mealy function (Z) is given below.

$$Z = f(X, Q_n)$$

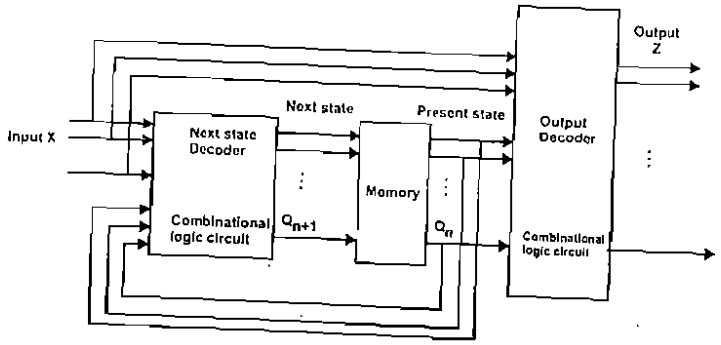


Fig 6.2 Mealy model sequential circuit

The output of memory element is conneeted to the input of output deeoder and next state decoder circuit. The output of memory element is considered as present state.

6.1.2 Moore Model Sequential Circuit

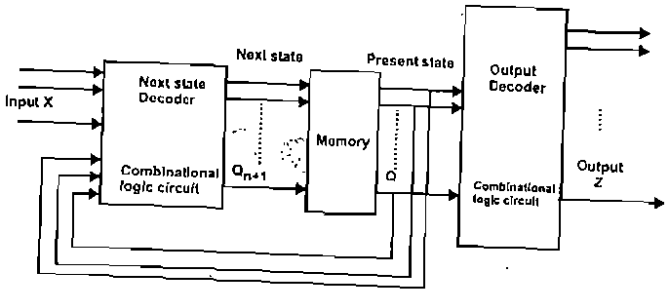


Fig 6.3 Moore model sequential circuit

Fig.6.3 shows the block diagram of a Moore machine. The output of Moore machine depends only on the present state. So the output of Moore machine is a function of its present state (Q_n). If the input is X, the next state is $Q_{(n+1)}$ and the present state is Q_n. The output of Moore machine is represented mathematically by

$$Z = f(Q_n)$$

The differences between the Moore machine and Mealy machine are tabulated as follows

S.No.	Moore machine	Mealy machine
1.	The output of this machine is the function of the present state only.	Its output is function of present input as well as present state
2.	Input changes do not affect the output	Input changes may affect the output of the circuit
3.	It requires more number of states for implementing same function	It requires less number of states for implementing same function

6.2 Analysis and Synthesis of Synchronous Sequential Circuits

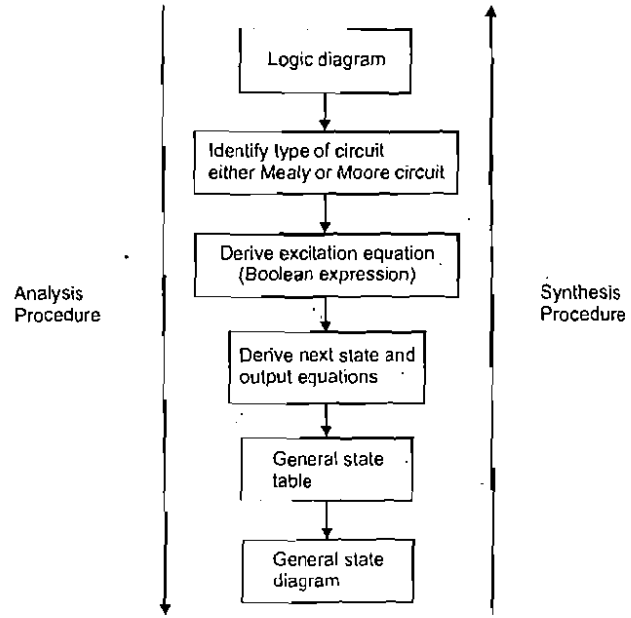


Fig 6.4 Flow chart

The behaviour of sequential circuit can be determined from the inputs, outputs and state of its flip flops. The outputs and next state are both a function of its inputs and the present state. The analysis of a sequential circuit consists of obtaining a state table or state diagram for the time sequence of inputs, outputs and internal states. The analysis of the clocked sequential circuits can be done by following the procedure as shown in Fig.6.4. The reverse process of analysis is known as synthesis of clocked sequential logic circuit.

For the analysis of sequential circuit, we start with the logic diagram. The excitation equation or Boolean expression of each flip-flop is derived from this logic diagram. Then, to obtain the next state equation, we insert the excitation equations into the characteristic equations. The output equations can be derived from the schematic. We can generate the state table using output and next state equations.

6.2.1 Analysis of Example Sequential Logic Circuit

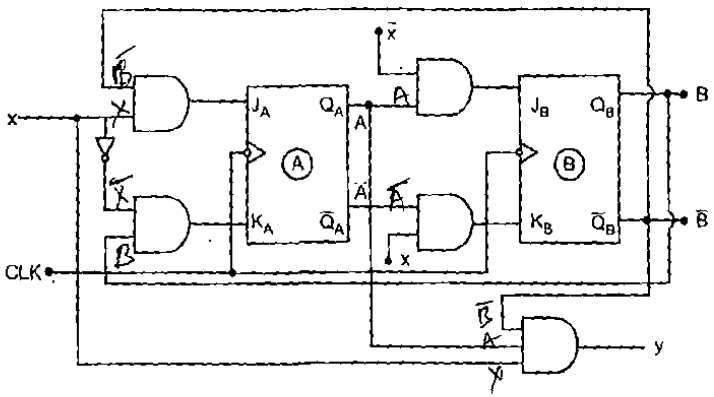


Fig 6.5 Example of sequential logic circuit

Fig. 6.5 shows a clocked sequential circuit. It has one input variable x, one output variable y and two clocked JK flip flops. The flip flops are labelled as A and B and their outputs are labelled as A and $\bar{A}$. B and $\bar{B}$ respectively.

Step 1 : Type of circuit

The output(y) of given logic circuit (Fig.6.5) depends on present input and also on present state (Flip flop outputs) of flip flops, so that the given sequential logic circuit is Mealy sequential machine.

Step 2 : Excitation equations

The excitation equations or Boolean expressions of flip flops A and B are obtained. The equations will be in the form of present states A and B and external

input x, since here are two JK flip flops which have output A and B. Therefore the excitation equation (equation formed for flip flop input)

For Flip flop - A $\quad J_A = x\overline{B}$, $\quad K_A = \overline{x}B$

For flip flop - B $\quad J_B = \overline{x}A$, $\quad K_B = x\overline{A}$

Step 3 : Next state equations The state equations can be derived directly from the logic diagram. Looking at Fig.6.5 we can see that the signal for J input of the flip flop A is generated by the function $\overline{B}x$ and the signal for input K by the function $B\overline{x}$. Substituting $J = \overline{B}x$ and $K = B\overline{x}$ into a JK flip flop characteristic equation given by

$$Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n$$

State equation for flip flop A

$$\begin{aligned}
A_{n+1} &= (\overline{B}x)\overline{Q}_n + (B\overline{x})Q_n \qquad \text{where } Q_n = A \\
&= \overline{B}x\overline{A} + B\overline{x}A \\
&= \overline{A}\,\overline{B}x + A(\overline{B}\overline{x}) \\
&= \overline{A}\,\overline{B}x + A(\overline{B} + x) \\
&= \overline{A}\,\overline{B}x + A\overline{B} + Ax \\
&= \overline{\overline{A}\,\overline{B} + x(A + \overline{A}\overline{B})} \\
&= AB + x(A + \overline{B}) \qquad \because \ (A + \overline{A}\,\overline{B} = A + \overline{B}) \\
A_{n+1} &= A\overline{B} + Ax + \overline{B}x
\end{aligned}$$

State equation for flip flop B

Similarly, we can find the state equation for flip flop B. $J = \overline{A}x$ and $K = \overline{A}x$. Therefore the state equation of flip flop B is given as

$$\begin{aligned}
B_{n+1} &= A\overline{x}\,\overline{B} + (\overline{\overline{A}\,x})B \\
&= A\overline{x}\overline{B} + (A + \overline{x})B \\
&= A\overline{x}\overline{B} + AB + B\overline{x} \\
&= \overline{x}(A\overline{B} + B) + AB \\
&= \overline{x}(A + B) + AB \\
B_{n+1} &= A\overline{x} + B\overline{x} + AB
\end{aligned}$$

Output equation

The given sequential circuit has output y. The output equation can be found from the Fig.6.5 which is derived using three input AND gate

$$y = A\bar{B}x$$

Step 4: State table

Table 6.1 is the state table for the given sequential logic circuit. It represents the relationship between input, output and flip flop states. It consists of three columns: present state, next state and output

Present state: It specifies the state of the flip flop before occurrence of a clock pulse.

Next state: It is the state of flip flop after the application of a clock pulse.

Output: This section gives the value of the output variables during the present state. Both next state and output section have two columns representing two possible input conditions $x = 0$ and $x = 1$.

Table 6.1

Present state	Next state		Output	
	AB	AB	y	
AB	$x=0$	$x=1$	$x=0$	$x=1$
00	00	10	0	0
01	01	00	0	0
10	11	10	0	1
11	01	11	0	1

We can derive the state table as follows

(i) If present state $AB = 00, x = 0$

When a present state is 00 i.e. $A = 0$ and $B = 0$ and input $x = 0$, the next state is obtained by using next state equation

Next state for flip flop A

$$\begin{aligned} A_{n+1} &= A\bar{B} + Ax + \bar{B}x \\ &= 01 + 0.0 + 1.0 \\ &= 0 \end{aligned}$$

Next state for flip flop B

$$\begin{aligned} B_{n+1} &= A\bar{x} + B\bar{x} + AB \\ &= 0.1 + 0.1 + 0.0 \\ &= 0 \end{aligned}$$

Next state for this case $AB = 00$

(ii) If present state $AB = 00, x = 1$

Next state for flip-flop A

$$\begin{aligned} A_{n+1} &= A\bar{B} + Ax + \bar{B}x \\ &= 0.1 + 0.1 + 1.1 \\ &= 1 \end{aligned}$$

Next state flip flop B

$$\begin{aligned} B_{n+1} &= A\bar{x} + B\bar{x} + AB \\ &= 0.0 + 0.0 + 0.0 \\ &= 0 \end{aligned}$$

Next state for this case $AB = 10$

Similarly we can obtain next state for all these different cases as shown in the table.

(iii) Determine the entries in the output section. For this, we have to examine AND gate for all possible present states and input.

(a) If a present state $AB = 00, \quad x = 0$

$$\begin{aligned} \text{output } y &= A\bar{B}x \\ &= 0.10 \end{aligned}$$

$$y = 0$$

(b) If a present state $AB = 00, \quad x = 1$

$$\begin{aligned} \text{output } y &= A\bar{B}x \\ &= 0.11 \end{aligned}$$

$$y = 0$$

Thus, the state table of any sequential circuit can be obtained by the same procedure used in the above example. This example contains 2 flip flops and one input, and one output, producing four rows, two columns in the next state and output sections. In general, a sequential circuit with m flip-flops and n-input variables produces 2^m rows and one for each state and 2^n columns, one for each input combination in the next state and output sections of the state table.

Step 5 State diagram

State diagram is a graphical representation of a state table. Fig.6.6 shows the state diagram for sequential circuit. Here each state is represented by a circle, and transition between states is indicated by directed lines connecting the circles. The binary number inside each circle identifies the state represented by the circle. The directed lines are labelled with two binary numbers separated by a symbol '/' (slash). The input value that causes the state transition is labelled first and output value is next.

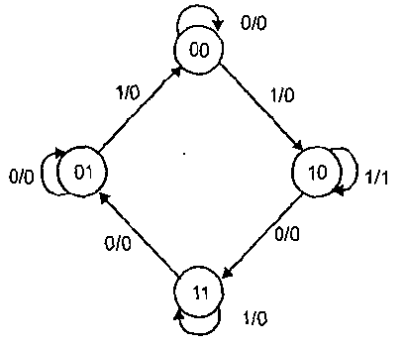


Fig 6.6 State diagram of Fig.6.5

Example 6.1 Derive the state table and state diagram for the sequential circuit shown in Fig.6.7(a).

□ **Solution**

Step 1: Type of circuit

The output y of given sequential circuit (Fig.6.7) depends on the present input and also present state (flip flop output) of flip flops, so the given sequential logic circuit is Mealy sequential machine.

Step 2: Excitation equation

For flip flop A

$$J_A = x + B$$
$$K_A = A$$

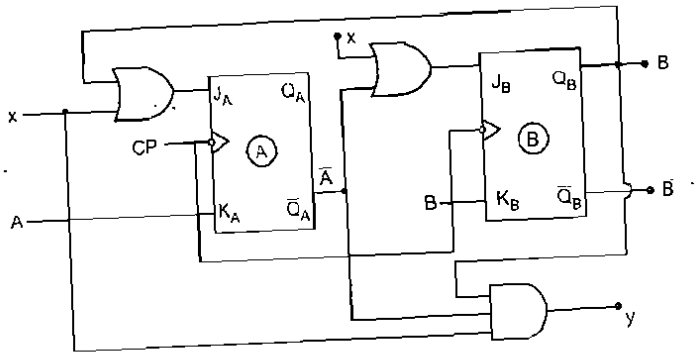


Fig 6.7

For flip flop B

$$J_B = \overline{A} + x$$
$$K_B = B$$

Step 3

We know that characteristic equation of JK flip flop

$$Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n$$

State equation for flip flop A

$$A_{n+1} = (x + B)\overline{Q}_n + AQ_n \qquad \text{(where } Q_n = A \text{ for flip flop } A\text{)}$$
$$= (x + B)\overline{A} + \overline{A}A$$
$$= \overline{A}x + \overline{A}B + 0$$
$$A_{n+1} = \overline{A}x + \overline{A}B$$

State equation for flip flop B

$$B_{n+1} = (\overline{A} + x)\overline{Q}_n + \overline{B}Q_n \qquad \text{(where } Q_n = B \text{ for flip flop } B\text{)}$$
$$= (\overline{A} + x)\overline{B} + \overline{B}B$$
$$B_{n+1} = \overline{A}\,\overline{B} + \overline{B}x$$

Output equation

$$y = \overline{A}Bx$$

Step 4 : State table

Present state	Next state		Output	
	AB	$\bar{A}B$	y	
$A\bar{B}$	$x=0$	$x=1$	$x=0$	$x=1$
00	00	11	0	0
01	10	10	0	1
10	01	00	0	0
11	00	00	0	0

Step 5 : State diagram

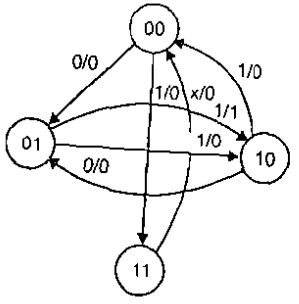


Fig 6.8

Example 6.2 Derive the state table and state diagram for the sequential circuit shown in Fig 6.9

□ **Solution**

Step 1 : Type of circuit

The output of a given circuit(See Fig.6.9) depends on present input and also on present states. so the given sequential logic circuit is Mealy machines

Step 2: Excitation Equation

For flip flop A $\quad D_A = Ax + B\bar{x}$

For Flip flip B $\quad D_B = \bar{A}x$

For Output $\quad y = AB + \bar{x}$

Step 3:

We know that characteristic equation of D flip flop (next state depends on input D)

$$A_{n+1} = D_A$$

$$\boxed{A_{n+1} = Ax + B\bar{x}}$$

$$B_{n+1} = D_B$$

$$\boxed{B_{n+1} = \bar{A}x}$$

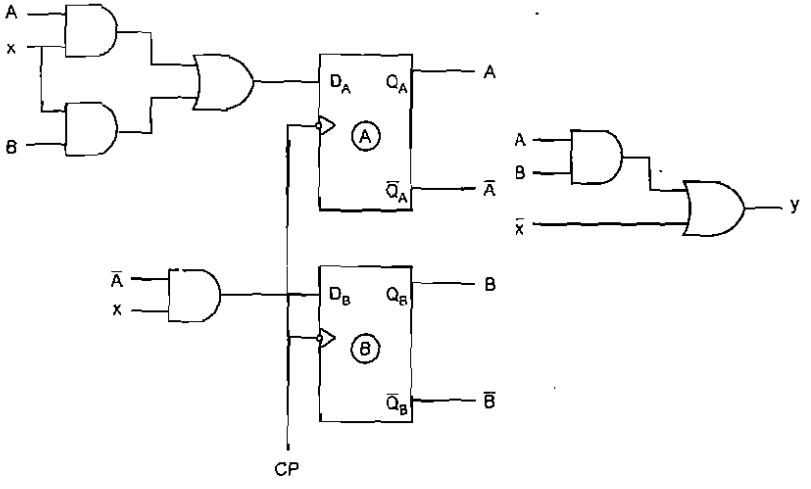


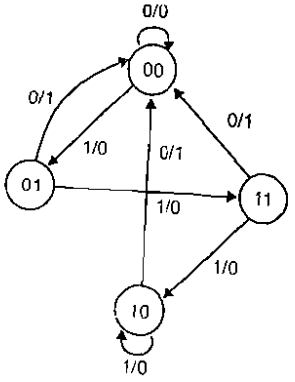
Fig 6.9

Step 4: State table

The state table contains four rows and three columns. The next state and output have two sub columns.

Present state	Next state		Output	
	AB	$\bar{A}B$	y	
$A\bar{B}$	$x=0$	$x=1$	$x=0$	$x=1$
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

Step 5: State Diagram



Example 6.3 Derive the state table and state diagram for sequential circuit shown in Fig.6.10

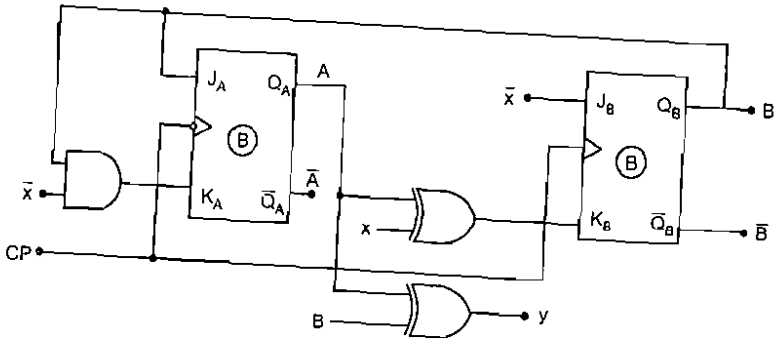


Fig 6.10

□ **Solution**

***Step 1* : Type of circuit**

The output y of the sequential circuit depends on present state only, so the given logic circuit is the Moore type circuit.

Step 2: Excitation equations

For flip flop A

$$J_A = B$$
$$K_A = B\bar{x}$$

For flip flop B

$$J_B = \bar{x}$$
$$K_B = A \oplus x$$
$$y = A \oplus B$$

Step 3 :

We know that characteristics equation of JK flip flop

$$A_{n+1} = J\overline{Q}_n + \overline{K}Q_n$$

State equation for flip flop A : $A_{n+1} = B\overline{A} + (\overline{B\bar{x}})\,A$ $\quad (\because \quad Q_n = A)$

$$= B\overline{A} + A(\overline{B} + x)$$
$$= B\overline{A} + A\overline{B} + x$$
$$A_{n+1} = (A \oplus B) + x$$

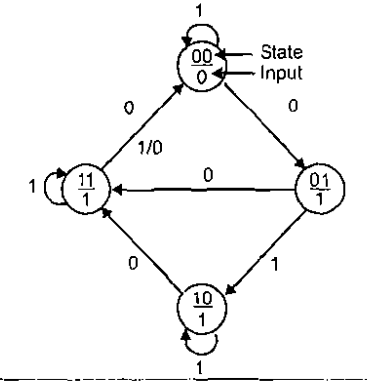
State equation for flip flop B : $B_{n+1} = \bar{x}\overline{B} + (\overline{A \oplus \bar{x}})B$

$$= \bar{x}\,\overline{B} + (Ax + \overline{A}\,\bar{x})B$$
$$B_{n+1} = \bar{x}\overline{B} + AxB + \overline{A}\bar{x}B$$

Step 4: State table

Present state	Next state		Output
	$x = 0$	$x = 1$	y
AB	AB	AB	
00	01	00	0
01	11	10	1
10	11	10	1
11	00	11	0

State diagram



Note: The state diagram for Moore machine is different from Mealy machine. Here each circle is coded with state binary number/output.

6.3 State Reduction

Any logic design process *must* consider the problem of minimizing the cost of the final circuit. One way to reduce the cost is by reducing the number of flip flops, *i.e.* by reducing the number of states. The state reduction technique basically avoids the introduction of redundant equivalent states. The reduction of redundant states reduces the number of flip flops and logic gates required, thus reducing the cost of the final circuit. Two states are said to be redundant or equivalent, if every possible set of inputs generate exactly the same outputs and the same next states. When two states are equivalent, one of them can be removed without altering input-output relationship. Let us consider the state diagram shown in Fig 6.11. The states are denoted by letter symbols instead of their binary values because in state reduction technique internal states are also important, but input output sequences are more important. The procedure contains two steps.

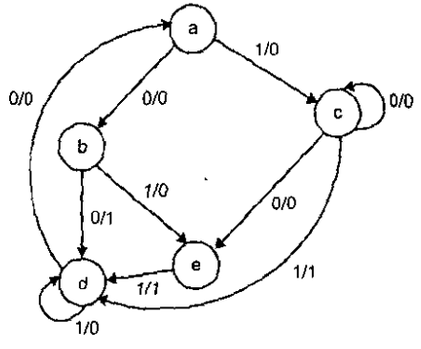


Fig 6.11

Step 1: Finding the state table for the given state diagram

First the given state diagram is converted into a state table. Fig.6.11 shows the example of state diagram.

Present state	Next state		Output	
	x=0	x=1	x=0	x=1
a	b	c	0	0
b	d	e	1	0
c	c	d	0	1
d	a	d	0	0
e	c	d	0	1

Both are equivalenl slates because of state c and e having same next state and same output

Step 2: Finding equivalent states

The two present states go to the same next state and have the same output for both the input combinations. We can easily find this from the state table. states

c and e are equivalent. This is because both c and e states go to states c and d outputs of 0 and 1 for $x = 0$, $x = 1$ respectively. Therefore, the state e can be removed and replaced by c. The final reduced table and state diagram are given in the table 6.2 and Fig.6.12. The second row have e state for the input $x = 1$, it replaced by c because the states c and e are equivalent.

Table 6.2 Reduced state table

Present state	Next state		Output y	
	AB	AB		
AB	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	b	c	0	0
b	d	c	1	0
c	c	d	0	1
d	a	d	0	0

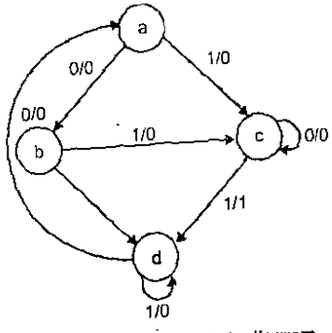


Fig. 6.12 Reduced state diagram

Example 6.4 Obtain the reduced state table and reduced state diagram for a sequential circuit whose state diagram is shown in Fig.6.13.

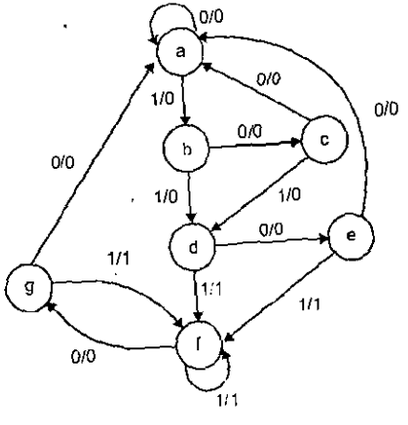


Fig 6.13

☐ Solution

The given diagram has seven states, one input and one output. As per the step 1, the given state diagram is converted to a state table.

State table

Table 6.4(a)

Present state	Next state		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Both are equivalent states because of state e and g having same next state and same output

From the above state table, it is clear that states e and g are equivalent. So the state g is replaced by state e. The reduced state table is shown in Ex.6.4

Reduced state Table

Table 6.4(b)

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

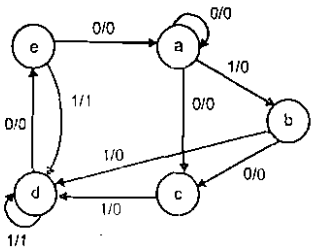
Both are equivalent states

From the above reduced table, states d and f are equivalent, hence 'f' can be replaced by d and it can be removed. Then finally the reduced state table is shown in Table 6.4(c)

Final reduced table

The state diagram of the reduced state Table is shown in Fig.6.4(b).

Present state	Next state		Output	
	AB	AB		
AB	$x=0$	$x=1$	$x=0$	$x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1



6.4 State Assignment

In sequential circuits we know that the behaviour of the circuit is defined in terms of its inputs, present state, next state and outputs. To generate the desired next state at particular present state and inputs, it is necessary to have specific flip flop inputs. These flip flop inputs are described by a set of Boolean functions called flip flop input functions. To determine the flip flop input functions, it is necessary to represent states in the state diagram using binary values instead of alphabets. This procedure is known as state assignment. The following rules are used in state assignment.

Rule 1. States having the same next states for a given input condition should have assignments which can be grouped into logically adjacent cells in a K-map. (Fig.6.15)

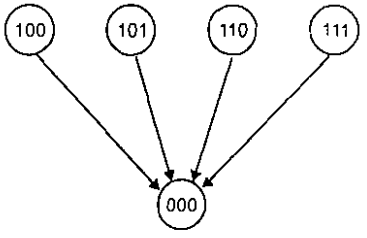


Fig.6.15

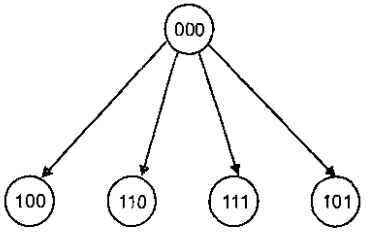


Fig.6.16

Rule 2. States having different next states should have assignment which can be grouped into logically adjacent cells in K-map. (Fig.6.16)

Example 6.5 Design a sequential circuit using D flip flop for a state diagram given below. Use state assignment rules for assigning states and compare the required combinational circuit with random state

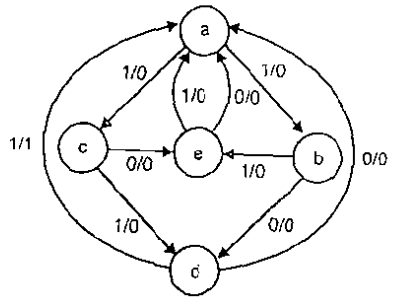


Fig 6.17

❑ Solution

The states are a, b, c, d and e. Each state is randomly assigned.

$a = 000$, $b = 001$, $c = 010$, $d = 011$, $e = 100$. The remaining combinations are considered as don't care conditions.

Excitation table

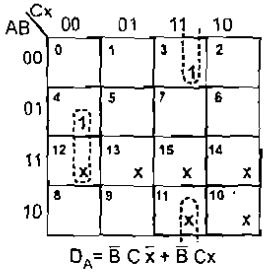
Present state			Input	Next State			Output
A	B	C	X	A_{n+1}	B_{n+1}	C_{n+1}	Z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	0
0	1	0	1	0	1	1	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0

Present state			Input	Next State			Output
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

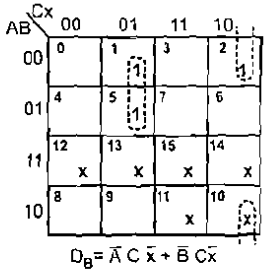
K-map simplification

The D flip flop input is equal to next state and the flip flop expression is obtained directly.

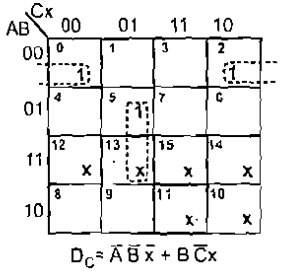
Expression for flip flop input D_A



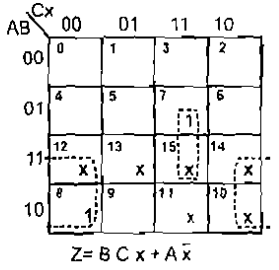
Expression for flip flop input D_B



Expression for flip flop input D_C



Expression for flip flop input D_D



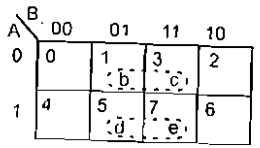
The random assignment requires

7 three input AND gates
1 two input AND gates
4 two input OR gates

Total 12 gates with 31 inputs and 3 flip flops are required to construct the sequential logic circuit. Now we apply state assignment rules, then follow the above steps.

From Rule 1, The states e and d must be adjacent

From Rule 2, states *b* and *c* must be adjacent. We form the adjacent cells in the 3 variable K-map

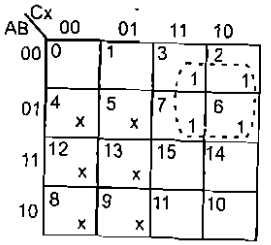


Excitation table

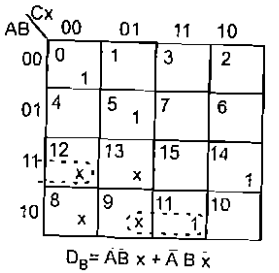
Present state			Input	Next State			Output
A	*B*	*C*	*X*	A_{n+1}	B_{n+1}	C_{n+1}	*Z*
0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0
0	0	1	1	1	1	1	0
0	1	0	0	*X*	*X*	*X*	*X*
0	1	0	1	*X*	*X*	*X*	*X*
0	1	1	0	1	1	1	0
0	1	1	1	1	0	1	0
1	0	0	0	*X*	*X*	*X*	*X*
1	0	0	1	*X*	*X*	*X*	*X*
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1
1	1	0	0	*X*	*X*	*X*	*X*
1	1	0	1	*X*	*X*	*X*	*X*
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

***K*- Map simplification**

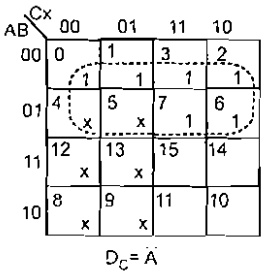
Expression for flip flop input D_A



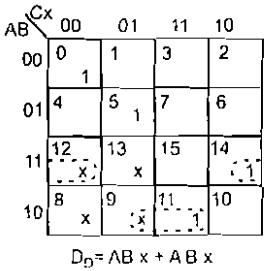
Expression for flip flop input D_B



Expression for flip flop input D_C



Expression for flip flop input D_D



Under the state assignment rules, we require

4 three input AND gates

1 two input AND gate

2 two input OR gates

A total of 7 gates with 18 inputs and 3 flip flops are required to construct the sequential logic circuit based on the state assignment rules.

6.5 Design Procedure

The following steps are followed to design the clocked sequential logic circuit.

1. Obtain the state table from the given circuit information such as a state diagram, a timing diagram or description.
2. The number of states may be reduced by state reduction technique.
3. Assign binary values to each state in the state table.
4. Determine the number of flip flops required and assign a letter symbol to each flip flop.
5. Choose the flip flop type to be used according to the application.
6. Derive the excitation table from the reduced state table.
7. Derive the expression for flip flop inputs and outputs using k-map simplification (The present state and inputs are considered for k-map simplification) and draw logic circuit using flip flops and gates

6.6 Synthesis of Clocked Sequential Logic Circuits

Synthesis means that, it is the reverse process of analysing a sequential logic circuit. In this synthesis, we get a logic circuit from the information of state diagram, word description etc. The detailed steps are given in the example. Now we will see the detailed description of each step.

A state diagram is obtained from the word description, timing diagram or other pertinent information. From this state diagram, we can form a state table.

The reduction of number of states and binary value assignment to each state gives the reduction in combinational circuit requirement. The number of flip flops required to design any sequential logic circuit depends on the number of states.

Example 6.6 A sequential circuit has one input and one output and its state diagram is shown in Fig.6.18(a).Design the sequential circuit using *D* flip flop

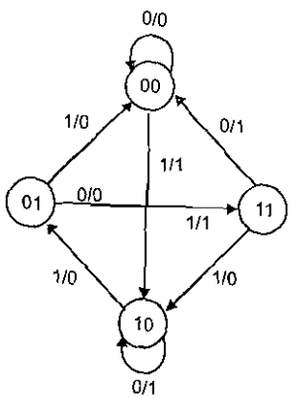


Fig 6.18(a)

☐ Solution

The given state diagram consists of four states. It hat one input (x) and one output (y). The state table for the given state diagram is shown in Table.6.6(a). It is clear that there are no equivalent states. Therefore, there is no reduction in the state diagram. As the state diagram contains 4-states, it requires 2 flip-flops which are named as A and B.

Table 6.6 (a)

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
AB	$A\bar{B}$	AB	y	y
00	00	10	0	1
01	11	00	0	0
10	10	01	1	0
11	00	10	1	0

Design using *D*-flip flop

For the design of circuit using D flip flop (or any flip flop), we need the excitation table. Table Ex 6.6(b) shows the excitation table of D flip flop from which we can develop excitation table for the required circuit as shown in table 6.6(c).

Table 6.6 (b) Excitation table for *D*-flip flop

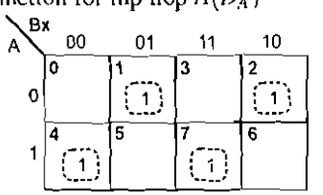
Present state	Next state	Flip flop input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Excitation table

Present state		Input	Next state		Flip flop input		Output
A	B	x	A	B	D_A	D_B	y
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1
0	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
1	0	0	1	0	1	0	1
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	1
1	1	1	1	0	1	0	0

The flip-flop input function and the circuit output function are obtained by using K-map simplification.

Input equation (or) function for flip flop $A(D_A)$



$$D_A = \bar{A}\,\bar{B}\,x + \bar{A}\,B\bar{x} + A\bar{B}\,\bar{x} + ABx$$

$$= \bar{A}(\bar{B}x + B\bar{x}) + A(\bar{B}\,\bar{x} + Bx)$$

Let us consider $z = \bar{B}x + B\bar{x}$. then $\bar{B}\,\bar{x} + Bx = \bar{z}$. Simplify the above equation

$$DA = \bar{A}z + A\bar{z}$$

$$= A \oplus z$$

Substitute $z = \bar{B}x + B\bar{x} = B \oplus x$ in the above equation

$$\boxed{D_A = A \oplus B \oplus x}$$

Input equation for flip flop $B(D_B)$

A \ Bx	00	01	11	10
0	0	1	3	2 1
1	4	5 1	7	6

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

Output function y

A \ Bx	00	01	11	10
0	0	1 1	3	2 1
1	4 1	5	7	6 1

$$y = A\bar{x} + \bar{A}\,\bar{B}x$$

The input equation for flip and output equation are simulated as follows

$$DA = A \oplus B \oplus x$$

$$DB = \bar{A}\,B\bar{x} + A\bar{B}x$$

$$y = A\bar{x} + \bar{A}\,\bar{B}x$$

A sequential circuit using D flip flop is obtained by using the above equations as shown in fig 6.18(b).

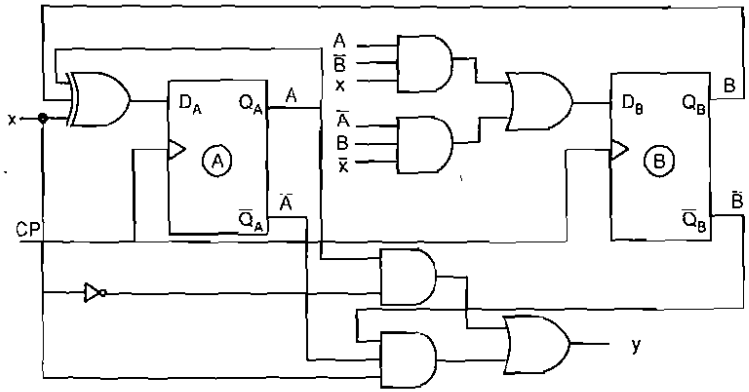


Fig 6.18(b)

6.7 Sequence Generator

A sequential circuit which generates a prescribed sequence of bits, synchronous with the clock, is referred to as a sequence generator. We can construct sequence generators by two ways

1. Sequence generators using counters
2. Sequence generators using shift registers

6.7.1 Sequence Generator using Counters

Fig.6.19 shows the block diagram of a sequence generator using counters. It contains two stages

1. counter, and
2. next state decoder.

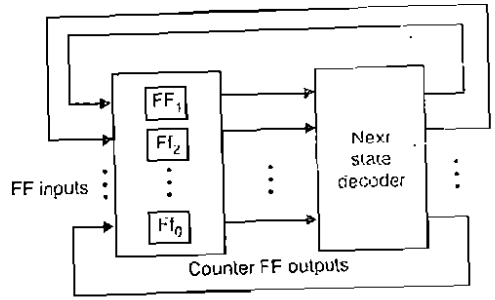


Fig 6.19

Design Procedure

Step 1 : Determine the number of flip-flops required

The number of flip-flops required to generate a particular sequence can be determined as follows.

(a) Find the number of 1's in the sequence.

(b) Find the number of 0's in the sequence.

(c) Take the maximum value from both. If 'n' is the required number of flip-flops, choose minimum value of 'n' to satisfy the following condition.

$$\max(0's, 1's) \leq 2^{n-1}$$

Step 2 : State assignment

Once the number of flip-flops is decided, we have to assign unique states corresponding to each bit in the given sequence such that the flip-flop representing least significant bit generates the given sequence (the output of the flip-flop which represents the least significant bit is used to represent the given sequence)

Step 3 : Draw the state diagram from the above state assignment and obtain the excitation table from the state diagram.

Step 4 : Find the Boolean expression for each flip-flop input by using k-map and draw the logic diagram for this Boolean expression

Example 6.7 Find the number of flip-flops required to generate the sequence 10110110.

Solution

In the given sequence, the number of 0's are 3 and number of 1's are 5.

$$\max(3,5) \leq 2^{n-1}$$

$$5 \leq 2^{n-1}$$

$$\boxed{n = 4}$$

■

Example 6.8 Design a sequence generator using *JK* flip-flop to generate the sequence 1101011.

Solution

Step 1: Number of flip-flops required

Number of 0's in the sequence = 2

Number of 1's in the sequence = 5

Hence $\max(2,5) \leq 2^{n-1}$

$$5 \leq 2^{n-1}$$

$$\boxed{n = 4}$$

We need four flip-flops named as A, B, C and D. The desired sequence is generated by the D flip-flops output

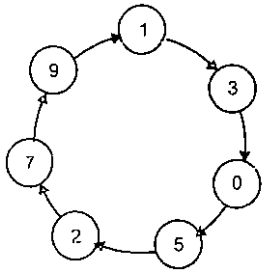
Step 2: State assignment

Decimal equivalent	*A*	*B*	*C*	*D*
1	0	0	0	1
3	0	0	1	1
0	0	0	0	0
5	0	1	0	1
2	0	0	1	0
7	0	1	1	1
9	1	0	0	1

← Given sequence, first enter this column

Assign binary value based on non-repeated states

Step 3: State diagram

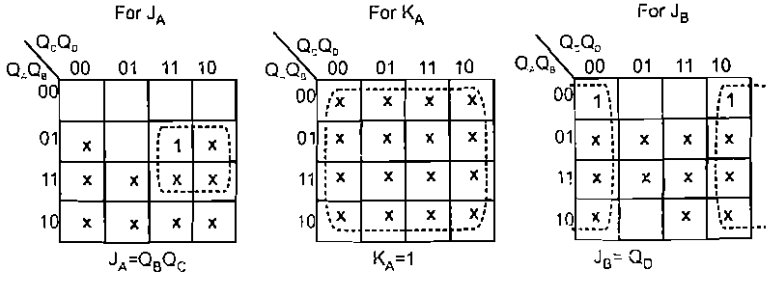


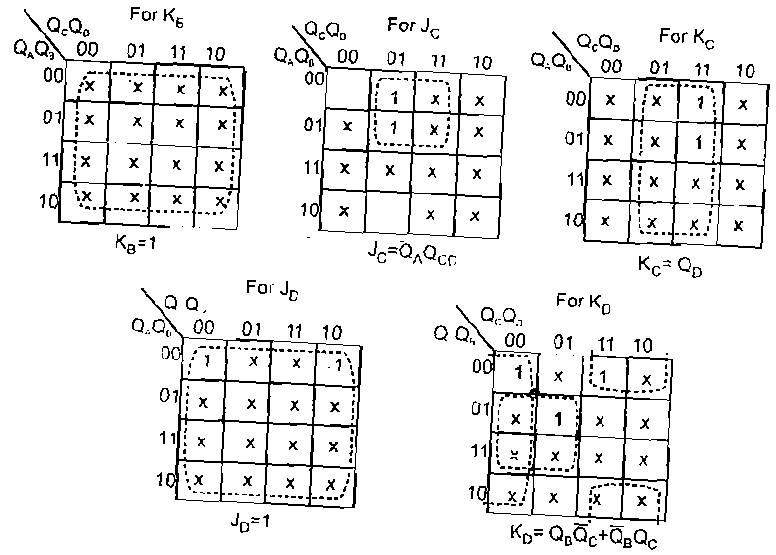
Excitation table

Present state				Next state				Flip-flop inputs							
Q_A	Q_B	Q_C	Q_D	Q_A	Q_B	Q_C	Q_D	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0	0	0	0	1	0	1	0	X	1	X	X	1	1	X
0	0	0	1	0	0	1	1	0	X	0	X	1	X	X	0
0	0	1	0	0	1	1	1	0	X	1	X	X	0	1	X
0	0	1	1	0	0	0	0	0	X	0	X	X	1	X	1
0	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
0	1	0	1	0	0	1	0	0	X	X	1	1	X	X	1
0	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
0	1	1	1	1	0	0	1	1	X	X	1	X	1	X	0
1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	0	1	0	0	0	1	X	1	0	X	0	X	X	0
1	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Note : The unused states 4, 6, 8, 10, 11, 12, 13, 14, and 15 are considered as *X*

***K*-map simplification**





Logic diagram

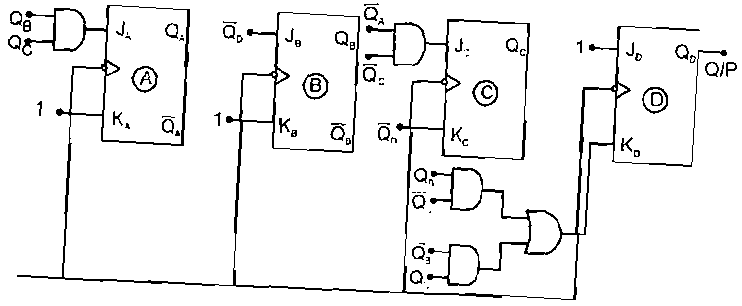
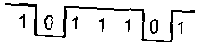


Fig 6.20

Example 6.9 Design a pulse train generator for the waveform shown below.



Solution

Step 1 : The pulse is repeated for every 4-bit sequence 0111. Therefore the required number of flip flop is determined as follows.

(i) number of 0's = 1
(ii) number of 1's = 3

Hence $\max(1,3) \le 2^{n-1}$

$3 \le 2^{n-1}$

$\boxed{n = 3}$

We need three flip-flop named as A, B and C. The desired sequence is generated by the C flip-flop.

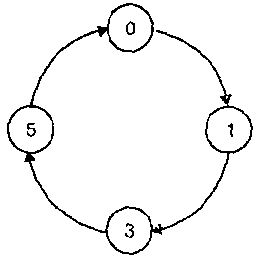
Step 2 : State assignment

Decimal equivalent	A	B	C
0	0	0	0
1	0	0	1
3	0	1	1
5	1	0	1

Given sequence

Note : The unused states are 2,4,6 and 7. Consider the don't care (X) for these states in the K-map simplification.

Step 3 : State diagram

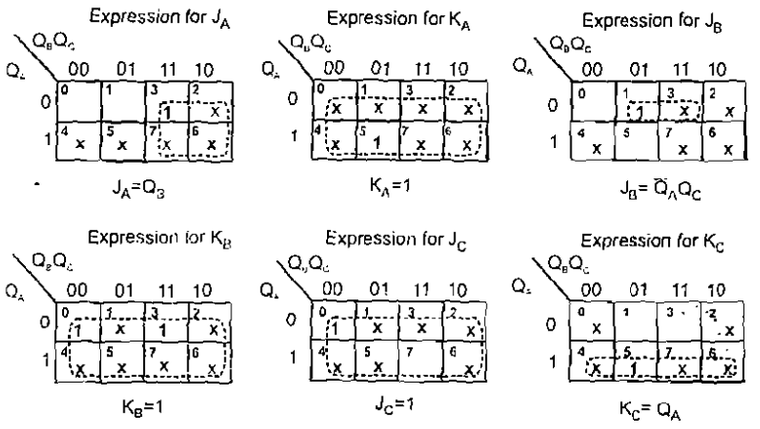


Excitation table

Present state			Next state			Flip-flop inputs					
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	1	1	0	1	1	X	X	1	X	0
1	0	1	0	0	0	X	1	0	X	X	1

Note : Unused states 2, 4, 6 and 7 are considered as X

K-map simplification



This minimal expression form is *K*-map simplification

$$J_A = Q_B \qquad J_B = \bar{Q}_A Q_C \qquad J_C = 1$$
$$K_A = 1 \qquad K_B = 1 \qquad K_C = Q_A$$

Logic diagram

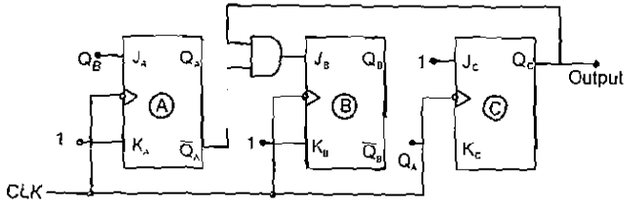


Fig 6.21

6.7.2 Sequence Generator using Shift Registers

This is another method for designing sequence generator. In this method shift registers with next state decoder logic are used. Fig.6.22 shows the block diagram of sequence generator using shift registers.

From this Fig.6.22.we see that the output of next state decoder is a function of Q_A. Q_B. ... Q_n. The next state decoder is a logic circuit which decodes the output of shift register and generates (input to get desired sequence from flip flop *A* (least significant bit).

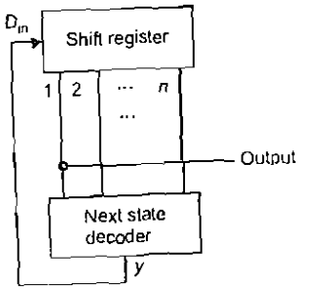


Fig 6.22 Block diagram of a sequence generator using shift registers

Example 6.10 Design a sequence generator to generate the sequence 1101011 by shift register method.

In this approach, the minimum number of flip-flops *n*, required to generate a sequence of length N is given by

$$N \leq 2^{n-1}$$

In this example $N = 7$ and therefore, the minimum value of *n*, which may generate the sequence is

$$7 \leq 2^n - 1$$
$$\boxed{n = 3}$$

With the three flip-flops. the sequence generation is shown in Table.6.7 The state diagram is shown In Fig.6.23

Table 6.7 State assignment

Flip-flop outputs			D_{in}	States
Q_A	Q_B	Q_C		
1	0	0	1	4
1	0	1	0	5
0	0	0	1	0
1	1	0	0	6
0	1	0	1	2
1	1	1	1	7

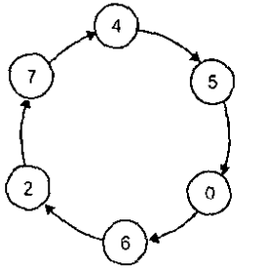
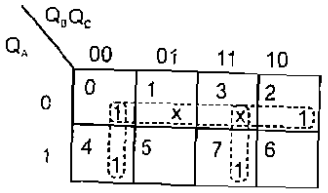


Fig.6.23

***K*-map simplification for D_{in}**



$$D_{in} = \bar{Q}_A + \bar{Q}_B\bar{Q}_C + Q_BQ_C$$
$$= \bar{Q}_A + Q_B \odot Q_C$$

The logic diagram is shown in Fig.6.24.The initial state is 100. So the input for D_A, D_B and D_C are 100 respectively.

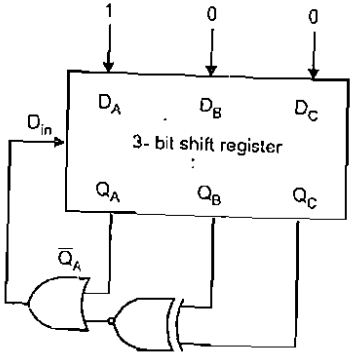


Fig 6.24

6.8 Sequence Detector

A sequence detector is a sequential logic circuit that can be used to detect whether a given sequence of bits has been received or not. We can draw the state diagram when we know the sequence and then follow the steps to design sequential logic circuit to obtain the sequence detector sequential logic circuit.

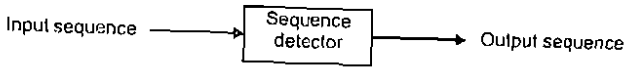


Fig 6.25

Generally sequence detector produces an output = 1, whenever it detects the desired input sequence and '0' for other cases. There are two types of detectors:

1. A detector which detects overlapping input sequence, and
2. A detector which detects non-overlapping input sequence.

Example 6.11 Design a sequence detector which detects the sequence 100011.

Solution

In general, the number of states in the state diagram is equal to the number of bits in the sequence. Once the number of states is known, one has to draw the directed lines with inputs and outputs as weighed between the two states. Let us start to draw state diagram, assuming initial state is A.

State A: In this state, the detector may receive either an input 0 or 1. Based on these inputs, a sequence detector is either in same state or move on to the next state as shown in Fig.6.26.

When input is 1, we have detected first bit in the sequence, hence we have to go to next state (B) to detect the next bit in the sequence

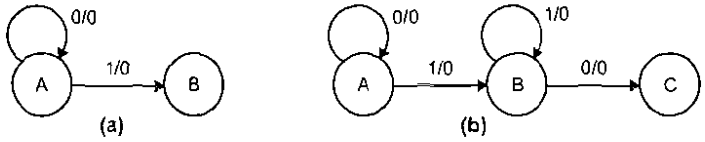


Fig 6.26

When input is 0, we have to remain in state A, because bit '0' is not the first bit in the sequence

State B: When input is 0, we have detected the second bit in the sequence. Hence we have to go to Next state (C) to detect the next bit in the sequence [See Fig.6.26(b)]

When input is 1, we have to remain in the state B, because 1 which we have detected may start the sequence output which is still zero for both cases

State C: When input is 0, we have detected the third bit in the sequence, hence we have to go next state (D) to detect the next bit in the sequence

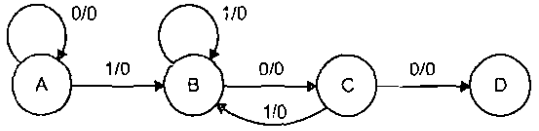


Fig 6.26(c)

When input is 1, we have to go to state B, because 1 which we have detected may not be in the sequence to be detected but it may be the start bit of the sequence, hence we can move to state B. The output is still zero (See Fig.6.26(c))

State D to State F: As explained for state A, state B and state C, if the desired bit is detected, we have to go for the next state otherwise we have to go to the previous state from where we can continue the desired sequence. When complete sequence is detected, we have to make output 1 and go to the initial state. The complete state diagram is shown in Fig.6.26(d).

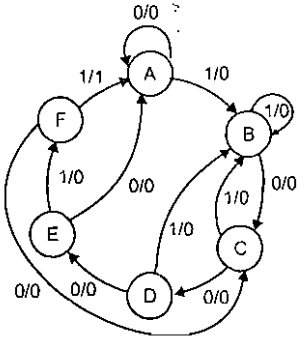


Fig 6.26(d)

State assignment

Assume that for state assignment, we need 3 flip flops to construct the sequence detector circuit. The sequence detector has a total of 6 states. Two flip flops are enough for less than or equal to 4 states. Hence $3(2^3 = 8)$ flip flops are required to construct the sequence detector. We choose JK flip flop and the flip flops are labeled as A, B and C, assuming state assignments as $A = 000, B = 001, C = 010, D = 011, E = 100$ and $F = 101$.

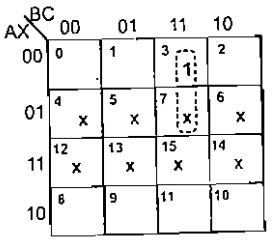
Excitation table

We can easily write the excitation table from the state diagram.

Input	Present state			Next state			Output	Flip flop Inputs					
X	A	B	C	A_{n+1}	B_{n+1}	C_{n+1}	Y	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	0	0	0	0	X	0	X	0	X
0	0	0	1	0	1	0	0	0	X	1	X	X	1
0	0	1	0	0	1	1	0	0	X	X	0	1	X
0	0	1	1	1	0	0	0	1	X	X	1	X	1
0	1	0	0	0	0	0	0	X	1	0	X	0	X
0	1	0	1	1	1	0	0	X	1	1	X	X	1
1	0	0	0	0	0	1	0	0	X	0	X	1	X
1	0	0	1	0	0	1	0	0	X	0	X	X	0
1	0	1	0	0	0	1	0	0	X	X	1	1	X
1	0	1	1	0	0	1	0	0	X	X	1	X	0
1	1	0	0	1	0	1	0	X	0	0	X	1	X
1	1	0	1	0	0	1	1	X	1	0	X	X	1

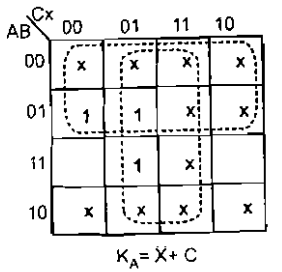
***K*-map simplification**

Expression for flip flop input J_A



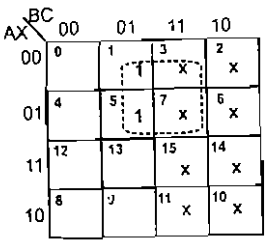
$J_A = \overline{X}\,BC$

Expression for flip flop input K_A



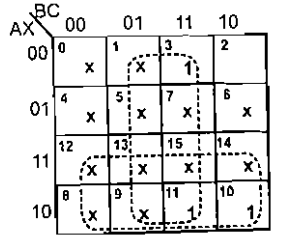
$K_A = \overline{X} + C$

Expression for flip flop input J_B



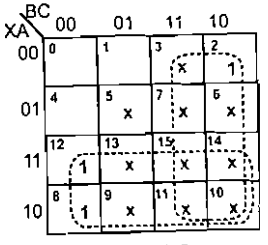
$J_B = \overline{X}\,C$

Expression for flip flop input K_B



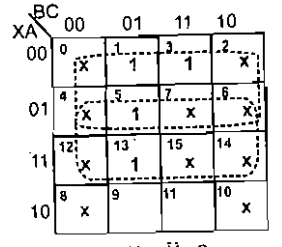
$K_B = \overline{X} + C$

Expression for flip flop input J_C



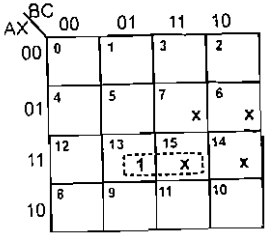
$J_C = X + B$

Expression for flip flop input K_C



$K_C = \overline{X} + C$

Expression for Output Y



$Y = XAC$

Logic diagram for sequence detector

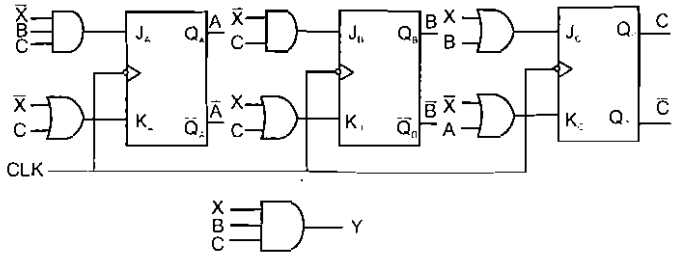


Fig 6.27 Logic diagram for sequence detector

Example 6.12 Design a sequence detector to detect the sequence 101 from 10101.

□ **Solution**

We have only 3 states because we have to detect the sequence 101 from the given number 10101. This circuit is allowed repetition.

Initially, we assume that the circuit is in its reset state, state *a*. With a 1 coming in as first bit in the valid sequence, it will go from state *a* to state *b* with an output as 0 because we have not yet detected all the bits in the sequence. When input is 0, we detect second valid bit in the sequence so it will go from state *b* to state *c*, otherwise state *b* remains same. When the input is 1, we detect third bit in the sequence, which will go from state *c* to state *b* with the output as 1 because we are yet to detect all bits in the sequence. If the input is 0, it will go from state *c* to state a with output 0.

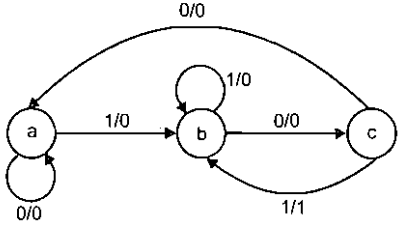


Fig 6.28 State diagram

The binary values are assigned to state *a*, *b* and *c*. Only two flip flops are enough ($2^2 = 4$)to design the sequence detector sequential logic circuit, by using *T* flip flops.

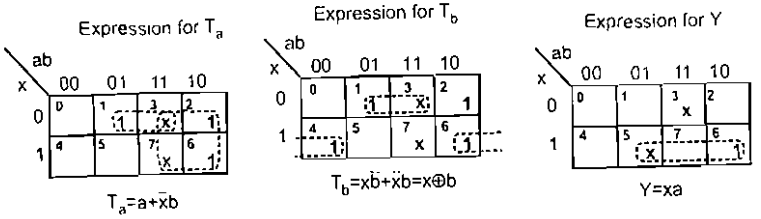
Assume the state assignment

$$a = 00, \qquad b = 01, \qquad c = 10$$

Consider the input is *X* and the output is *Y*.

Input	Present state		Next state		Flip flop Inputs		Output
X	*a*	*b*	a_{n+1}	b_{n+1}	T_a	T_b	*Y*
0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	0
0	1	1	X	X	X	X	X
1	0	0	0	1	0	1	0
1	0	1	0	1	0	0	0
1	1	0	0	1	1	1	1
1	1	1	X	X	X	X	X

***K*-map simplification**



Logic diagram for sequence detector

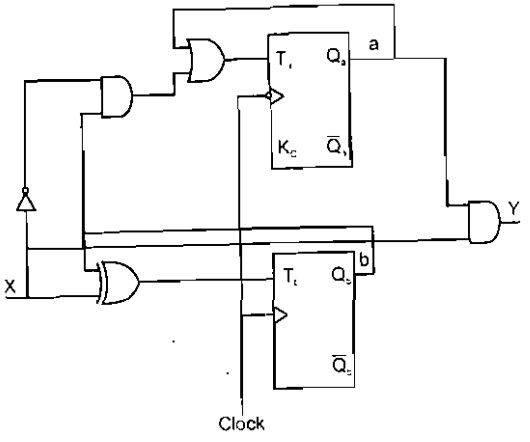


Fig 6.29 Logic diagram for sequence detector

Example 6.13 A sequential circuit with two *D* flip flops *A* and *B* and input *X* and output *Y* is specified by the following next state and output equations

$$A(t+1) = AX + BX$$
$$B(t+1) = A'X$$
$$Y = (A+B)X'$$

(a) Draw the logic diagram of the circuit

(b) Derive the state table

(c) Derive the state diagram

□ **Solution**

(a)

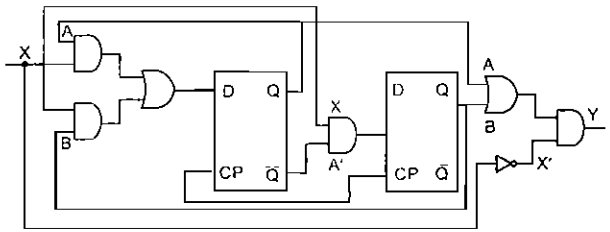


Fig 6.30

(b) **State Table**

Present state		Input	Next state		Flip flop Inputs		Output
A	B	X	A^+	B^+	D_A	D_B	Y
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	1	1	1	1	1	0
1	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0

(c) **State diagram**

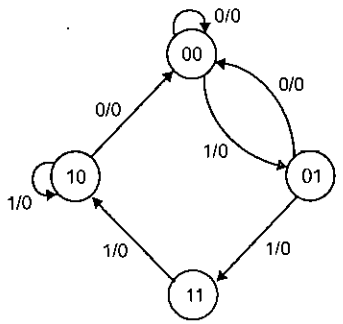


Fig 6.31

Example 6.14 Reduce the number of states in the following state table and tabulate the reduced state table.

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

□ **Solution**

Two states are said to be equivalent if their next states and outputs are equal.

Compare state 'a' with other states. The next state of 'a' and 'f' are equal for the inputs 0 and I but their outputs are not equal. Hence $a \neq f$. Similarly comparing other states it is found that $b \equiv e$ as their next state and outputs are equal and $d = h$. Hence the state table can be reduced as shown. The row e is removed from the table and if any previous row containing e, it is replaced by b and the rows are compared. If any state has the same next state and outputs replace one row, continue this process to obtain reduced state table.

[The strike out lines are kept to illustrate reduction process]

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	f	b	0	0
b	d	~~e~~a	0	0
~~c~~	~~f~~	~~b~~	~~0~~	~~0~~
d	g	a	1	0
~~e~~	~~d~~	~~c~~	~~0~~	~~0~~
f	f	b	1	1
g	g	~~h~~d	0	1
~~h~~	~~g~~	~~a~~	~~1~~	~~0~~

After the first comparison.we find that $a \equiv c$ hence the reduced state table is as shown below.

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

■

Example 6.15 Design a sequential circuit with two D flip flops, A and B, and one input x. When $x = 0$, the state of the circuit remains the same. When $x = 1$, the circuit passes through the state transitions from 00 to 01 to 11 to 10 and back to 00 and repeats.

□ **Solution**

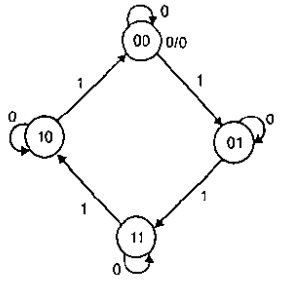


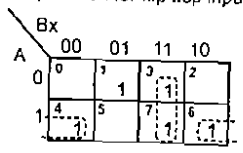
Fig 6.32

State table

Present state		Input	Next state		Output	
A	B	X	A^+	B^+	D_A	D_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	1	1	1
1	0	0	1	0	1	0
1	0	1	0	0	0	0
1	1	0	1	1	1	1
1	1	1	1	0	1	0

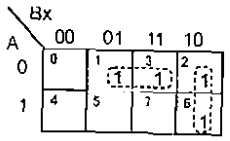
***K*-map simplification**

Expression for flip flop input D_A



$D_A = A\bar{x} + Bx$

Expression for flip flop input D_B



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

Circuit diagram

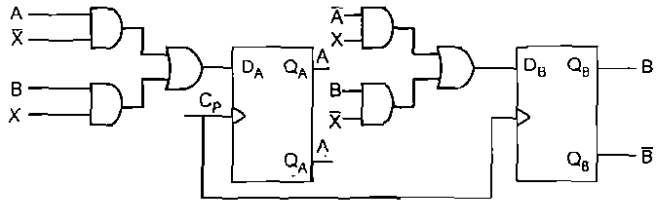


Fig 6.33

■

Example 6.16 Design a sequential circuit that three flip flops A, B, C, one input x and one output y. The state diagram is shown in the fig 6.34. The circuit is to be designed by treating the unused states as don't care conditions. Use JK flip flops in the design.

■ **Solution**

State diagram

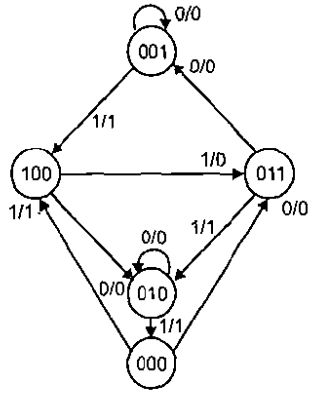


Fig 6.34

State table for the circuit

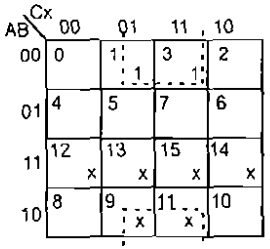
Present State			Next state						Output y	
			$x=0$			$x=1$				
A	B	C	A^+	B^+	C^+	A^+	B^+	C^+	$x=0$	$x=1$
0	0	1	0	0	1	1	0	0	0	1
1	0	0	0	1	0	0	1	1	0	0
0	1	0	0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	1	0	0	1
0	0	0	0	1	1	1	0	0	0	1

Excitation table for the circuit

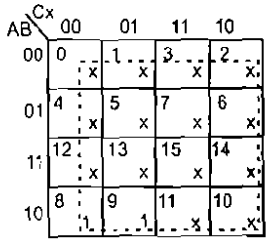
Present state			Input	Next state			Flip flop Inputs						Output
A	B	C	x	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C	y
0	0	0	0	0	1	1	0	X	1	X	1	X	0
0	0	0	1	1	0	0	1	X	0	X	0	X	1
0	0	1	0	0	0	1	0	X	0	X	X	0	0
0	0	1	1	1	0	0	1	X	0	X	X	1	1
0	1	0	0	0	1	0	0	X	X	0	0	X	0
0	1	0	1	0	0	0	0	X	X	1	0	X	1
0	1	1	0	0	0	1	0	X	X	1	X	0	0
0	1	1	1	0	1	0	0	X	X	0	X	1	1
1	0	0	0	0	1	0	X	1	1	X	0	X	0
1	0	0	1	0	1	1	X	1	1	X	1	X	0

***K*-map simplification**

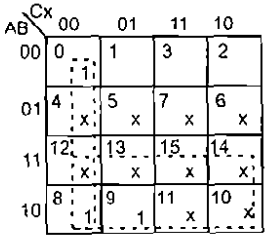
Note : minterms 10, 11, 12, 13, 14 and 15 are don't care



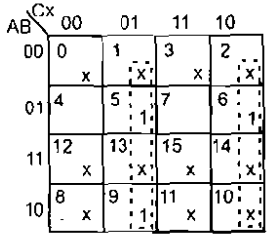
$J_A = \bar{B}x$



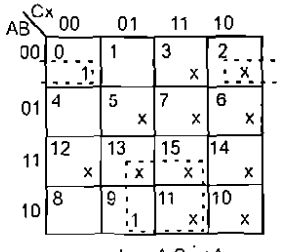
K_A=1(since all the cells are grouped)



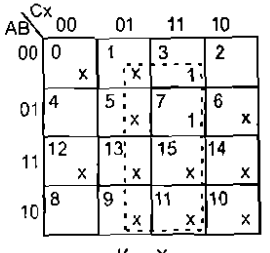
$J_B = \bar{C}\,\bar{X} + A$



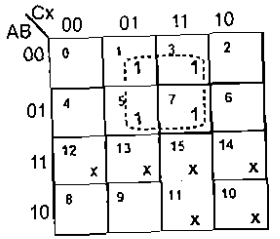
$K_B = \bar{C}x + C\bar{x} = C \oplus x$



$J_C = A\,B\,\bar{x} + Ax$



$K_C = X$



Y= Ax

Circuit Diagram

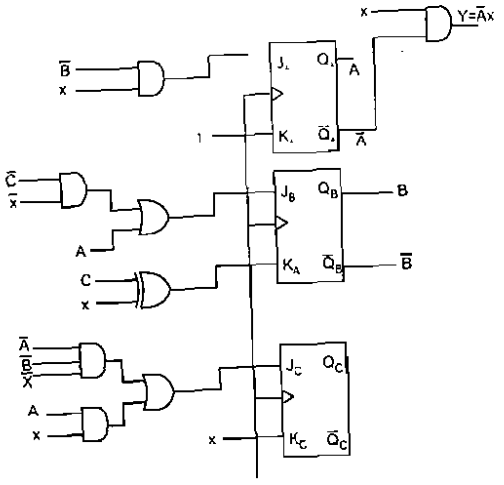


Fig 6.35

■

Example 6.17 Derive the state table and state diagram for the sequential circuit shown Fig 6.36.

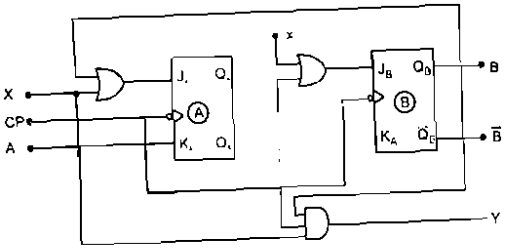


Fig 6.36

☐ Solution

The given circuit has two flip flops, A and B, one input X and one input Y. So, it produces a state table with 4 rows as shown below

Present State		Next state		Output y	
		$x=0$	$x=1$	$x=0$	$x=1$
A	B	$A\bar{B}$	AB	y	y
0	0	01	11	0	1
0	1	10	10	0	1
1	0	01	00	0	0
1	1	00	00	0	0

The state diagram for the sequential circuit is shown in Fig 6.37.

State diagram

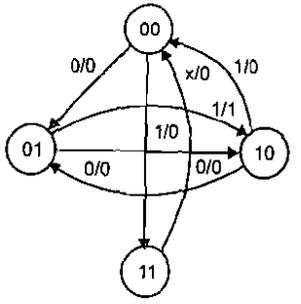


Fig 6.37

■

Example 6.18 A sequential circuit with two D flip flops, A and B; two inputs x and y; and one output z, is specified by the following next state and output equations:

$$A(t+1) = \bar{x}y + xA$$

$$B(t+1) = \bar{x}\bar{B} + xA$$

$$z = B$$

(a) Draw the logic diagram of the circuit

(b) List the state table for sequential circuit

(c) Draw the corresponding state diagram

☐ Solution

(a) Logic diagram of the sequential logic circuit

A sequential logic circuit is drawn using given equations as shown in Fig.6.38.

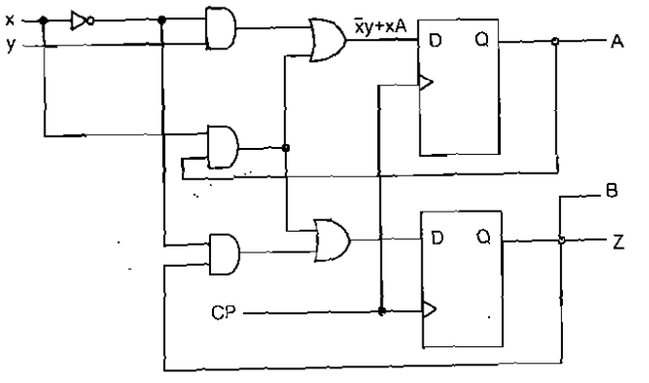


Fig 6.38

(b) State table

Present State		Inputs		Next state		Output
A	B	x	y	A	B	z
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	1	1	1
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	1	1	0
1	0	1	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

(c) State diagram

The state diagram is drawn as shown in Fig.6.39 with the help of the above table

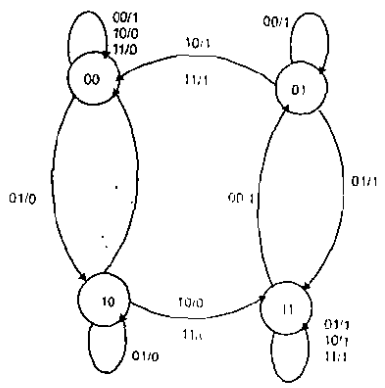


Fig 6.39

Example 6.19 Design a sequential circuit with two flip-flops A and B, and one input x. When $x = 0$, the state of the circuit remains same. When $x = 0$, the circuit goes through the state transitions tram 00 to 01 to 11 to 1 0 back to 00, and repeats.

☐ **Solution**

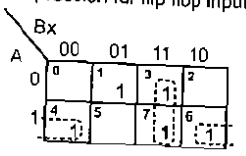
State table

Present state		Input	Next state		FF inputs	
A	B	x	A	B	D_A	D_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

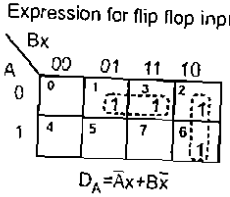
Note : the FF inputs of the D FF is same as the next state

K-map simplification

Expression for flip flop input D_A



Expression for flip flop input D_B



Logic Diagram

By using the above Boolean Expression, the diagram is constructed as shown in Fig.6.40.

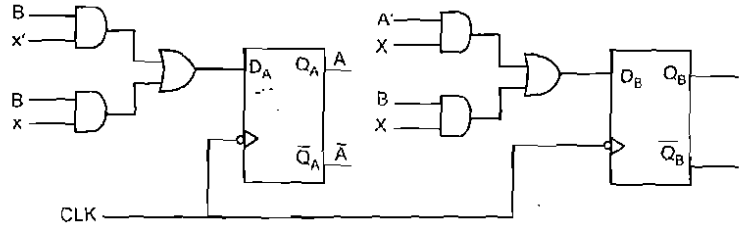


Fig 6.40

Example 6.20 Design a sequential circuit with two JK flip-flops A and B and two inputs E and x. If $E = 0$, the circuit remains in the same state regardless of the value of x. When $E = 1$ and $x = 1$, the circuit goes through the state transitions from 00 to 0 1 to 10 to 11 back to 00, and repeats. When $E = 1$ and $x = 0$, the circuit goes through the state transitions from 00 to 11 to 10 to 0 1 back to 00, and repeats.

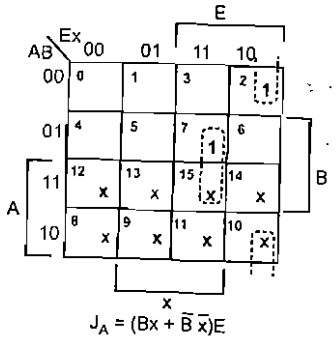
☐ **Solution**

The excitation table is derived directly from the given specification.

Present state		Input		Next state		Flip flop Inputs			
A	B	E	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	X	0	X
0	0	0	1	0	0	0	X	0	X
0	0	1	0	1	1	1	X	1	X
0	0	1	1	0	1	0	X	1	X
0	1	0	0	0	1	0	X	X	0
0	1	0	1	0	1	0	X	X	0
0	1	1	0	0	0	0	X	X	1
0	1	1	1	1	0	1	X	X	1
1	0	0	0	1	0	X	0	0	X
1	0	0	1	1	0	X	0	0	X
1	0	1	0	0	1	X	1	1	X
1	0	1	1	1	1	X	0	1	X
1	1	0	0	1	0	X	0	X	0
1	1	0	1	1	1	X	0	X	0
1	1	1	0	1	0	X	0	X	1
1	1	1	1	0	0	X	1	X	1

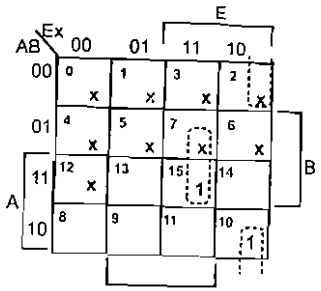
K-map simplification

Expression for J_A



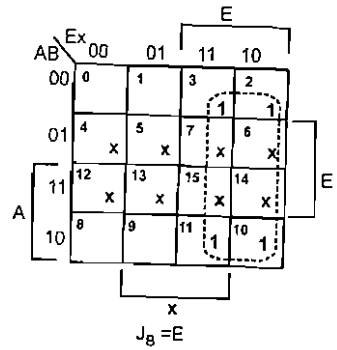
$J_A = (Bx + \bar{B}\,\bar{x})E$

Expression for K_A



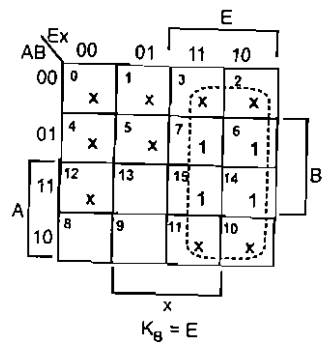
$K_A = BEx + \bar{B}\,\bar{E}\bar{x} = E(Bx + \bar{B}\bar{x})$

Expression for J_B



$J_B = E$

Expression for K_B



$K_B = E$

Logic diagram

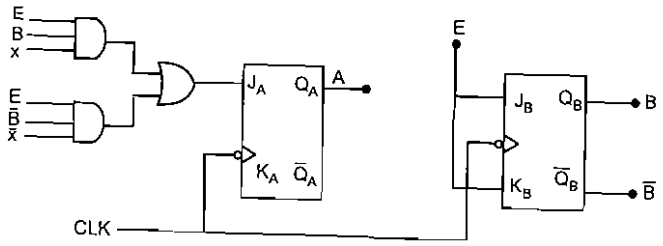


Fig 6.41

Short Questions and Answer

1. ***Mention the steps involved in the analysis of a sequential circuit.***

   The analysis of a sequential circuit consists of:

   (a) Obtaining a table or a diagram for the time sequence of inputs, outputs and internal states.
   (b) Writing Boolean expressions, which include the necessary time sequence, either directly or indirectly.

2. ***What is a state-equation?***

   A state-equation (also called transition equation) specifies the next state as a function of the present state and inputs.

3. ***What is a state-table?***

   A state-table contains data such as the time sequence of inputs, outputs and flip-flop states. The table consists of four sections labelled present state, input, next state and output state.

4. ***What is a state diagram?***

   A state diagram is a graphical representation of the information available in a state table. In the diagram, a state is represented by a circle and the transitions between states are indicated by directed lines connecting the circles.

5. ***What are the informations obtained from a state diagram?***

   The informations obtained are as follows :

   The state of the flip-flops are identified by the binary number inside the circle.

   A directed line connecting a circle which indicates that change of state occurs.

6. ***What are input and output equations?***

   The part of the circuit that generates the inputs to flipflops is described algebraically by a set of Boolean functions called input equations.

7. ***How are the next-state values of a sequential circuit that uses JK or T type flip-flops derived?***

   The next-state values of a sequential circuit that uses JK or T type flip-flops are derived using the following procedure:

   (a) Determine the flip flop input equations in terms of the present state and input variables.

(b) List the binary values of each input equation.
(c) Use the corresponding flip flop characteristic table to determine the next state value in the state table.

8. ***How are the next-state values obtained from the characteristic equation?***

The next state values can be obtained by evaluating the state equations from the characteristic equation. This is done by the following procedure:

(a) Determine the flip-flop input equations in terms of the present state and input variables.
(b) Substitute the input equations into the flip-flop characteristic equation to obtain the state equations.
(c) Use the corresponding state equations to determine the next state value in the state table.

9. ***What are the two models of sequential circuits?***

The two models of sequential circuits are:

(a) Mealy model, and
(b) Moore model.

10. ***Compare Mealy and Moore machine.***

Mealy machine	Moore machine
i. The output is a funetion of both the present state and input.	i. The output is a function of the present state only.
ii. The outputs may change if the inputs change during the clock cycle.	ii. The outputs are synchronized with the clock, because they depend only on flip-flop outputs that are synchronized with the clock.

11. ***What is the use of initial statement?***

The initial statement is used for generating input signals to simulate a design. In simulating a sequential circuit, it is necessary to generate a clock source for triggering the flip-flops.

12. ***How can the always statement be controlled?***

The always statement could be controlled by delays that wait for a certain time or by certain conditions to become true or by events to occur.

13. ***What is a sensitivity list?***

A sensitivity list specifies the events that must occur to initiate the execution of the procedural statements in the always block. Statements within the block execute sequentially and the execution suspends after the last statement has been executed.

14. ***What are the two kinds of procedural assignments?***

(a) The two kinds of procedural assignments are:

Blocking assignments

Blocking assignment statements are executed sequentially in the order, they are listed in a sequential block.

Blocking assignments use the symbol (=) as the assignment operator.

Example for procedural blocking assignments:

$$B = A$$
$$C = B + 1$$

(b) Non-blocking assignments:

It evaluates the expressions on the right hand side, but do not make the assignment to the left hand side, until all expressions are evaluated.

It uses the (<=) as the operator.

Example for non-blocking assignments: $B \Leftarrow A$

$$C \Leftarrow B + 1$$

15. ***How can we determine the behavior of a clocked sequential circuit?***

The behavior of a clocked sequential circuit could be determined from the inputs, outputs and the state of its flip-flops.

16. ***What are clocked sequential circuits?***

Synchronous sequential circuits that use clock pulses in the inputs of storage elements are called clocked sequential circuits.

17. ***How can we describe the structure of a sequential circuit?***

The sequential circuit is made up of flip-flops and gates and so its structure can be described by a combination of data flow and behavioral statements.

Short Answer Questions

1. Differentiate synchronous and asynchronous sequential logic circuits.
2. What are the classification of sequential machine ?
3. Define Mealy and Moore Machines.
4. Define state assignment.
5. When are two states said to be equivalent states ?

Review Questions

1. What are the steps for the design of an asynchronous sequential circuit ?
2. What is the significance of state assignment ?
3. List the different techniques used for state assignment.
4. Write a short note on

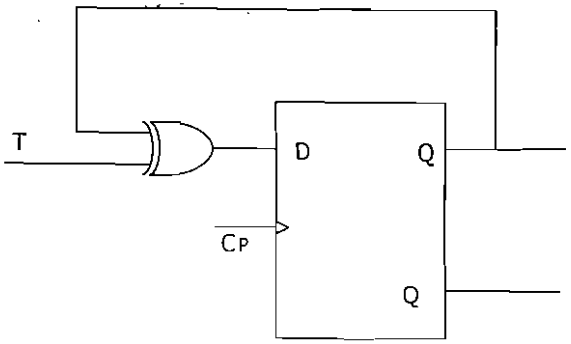
   (a) Shared row state assignment.
   (b) One hot state assignment.

Exercises

1. A sequential circuit with two D flip flops, A and B, two inputs x and y, and one input z is specified by the following next state and output equations.

$$A(t+1) = x'y + xA$$
$$B(t+1) = x'B + xA$$
$$Z = B$$

   (a) Draw the logic diagram of the circuit
   (b) Derive the state table.



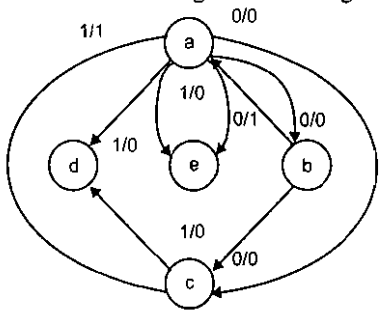
2. A sequential circuit has one flip flop Q, two inputs x and y: and one output S. It consists of a full-adder circuit connected to a D flip flop as shown in following fig.1. Derive the state table and state diagram of the sequential circuit.

3. Analyze the circuit in the following Fig.2 and prove that it is equivalent to a T flip flop.

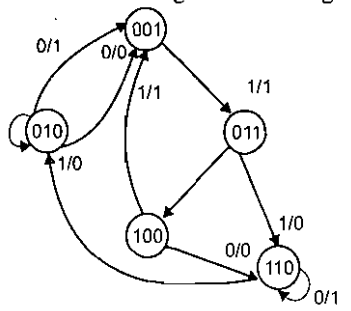
4. A sequential circuit has two JK flip flops, one input x and one output y. Following is the logic diagram of the circuit. Derive the state table and state diagram 7. Design a sequential circuit for the given state diagram by using JK flip flop of the circuit.

5. Design a sequential circuit with two JK flip flops, A and B and two inputs E and X. If E = 0, the circuit remains in the same state regardless of the value of x. When E = 1 and X = 1, the circuit goes through the state transitions from 00 to 01 to 10 to 11 back to 00 and repeats. When E = 1 and X = 0, the circuit goes through the state transitions from 00 to 11 to 10 to 01 back to 00 and repeats.

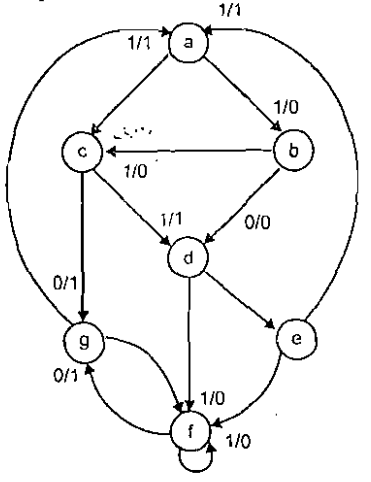
6. Design a sequential circuit for the given state diagram.



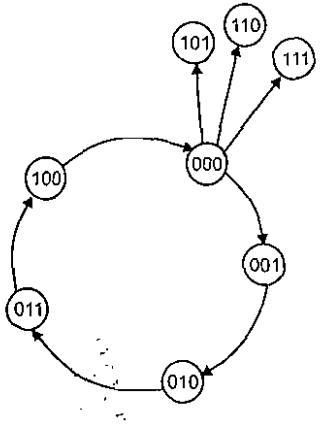
7. Design a sequential circuit for the given state diagram in Fig. P5.



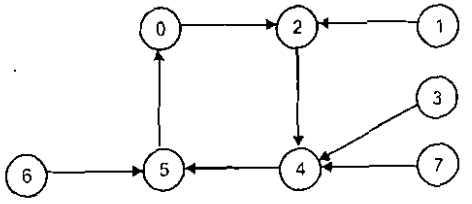
8. Design a clocked sequential circuit for the given state diagram in Fig. P6.



9. Design a sequential circuit for the following state diagram.



10. Design a sequential circuit for the given state diagram using JK flip flop.



11. Obtain the state table for the given state diagram and reduce it.