UNIT-1

Review of Number systems & codes

There are four types of number systems in digital electronics

① Decimal number system (Base 10)
② Binary number system (Base 2)
③ Octal number system (Base 8)
④ Hexa-decimal number system (Base 16)

Base is also known as radix.

If the base of any number system describes the no. of distinct symbols and the highest digit/number in that number system.

Ex: In hexadecimal number system, there are 16 symbols they are

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

* List the first 20 numbers in hexadecimal number system

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13

* List the first 15 numbers in Base 12 system

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, 10, 11, 12

* List the first 10 numbers in octal number system

0, 1, 2, 3, 4, 5, 6, 7, 10, 11

Base Conversions:

To Convert $(\ )_{10}$ to $(\ )_r$ perform the successive division of the given decimal number with required base upto the coefficient is less than the base.

Ex: Convert $(48)_{10} = (?)_2$

2	48
	24 - 0
	12 - 0
	6 - 0
	3 - 0
	1 - 1

$\rightarrow (48)_{10} = (110000)_2$

*Convert $(71)_{10} = (?)_2$

2	71
	35 - 1
	17 - 1
	8 - 1
	4 - 0
	2 - 0
	1 - 0

$(71)_{10} = (1000111)_2$

*Convert $(524)_{10} = (?)_2$

2	524
	262 - 0
	131 - 0
	65 - 1
	32 - 1
	16 - 0
	8 - 0
	4 - 0
	2 - 0
	1 - 0

$(524)_{10} = (1000001100)_2$

*Convert $(461)_{10} = (?)_2$

461
230 - 1
115 - 0
57 - 1
28 - 1
14 - 0
7 - 0
3 - 1
1 - 1

$(461)_{10} = (111001101)_2$

*Convert $(10.125)_{10} = (?)_2$

2	10
2	5 - 0
2	2 - 1
	1 - 0

$.125 \times 2 = 0.250$
$.250 \times 2 = 0.500$
$.500 \times 2 = 1.000$

$(10.125)_{10} = (1010.001)_2$

Scanned with CamScanner

* Convert $(39.225)_{10} = (?)_2$

2 | 39	
2 | 19 - 1	
2 | 9 - 1	
2 | 4 - 1	
2 | 2 - 0	
1 - 0	

- $.225 \times 2 = 0.450$
- $.450 \times 2 = 0.900$
- $.900 \times 2 = 1.800$
- $.800 \times 2 = 1.600$

$(39.225)_{10} = (100111.0011)_2$

* Convert $(63.525)_{10} = (?)_2$

2 | 63	
2 | 31 - 1	
2 | 15 - 1	
2 | 7 - 1	
2 | 3 - 1	
1 - 1	

- $.525 \times 2 = 1.050$
- $.050 \times 2 = 0.100$
- $.100 \times 2 = 0.200$
- $.200 \times 2 = 0.400$
- $.400 \times 2 = 0.800$

$(63.525)_{10} = (111111.1000)_2$

⇒ Conversion of decimal to octal

* Convert $(25)_{10} = (?)_8$

8 | 25	
3 - 1	

$(25)_{10} = (31)_8$

* Convert $(125)_{10} = (?)_8$

8 | 125	
8 | 15 - 5	
1 - 7	

$(125)_{10} = (175)_8$

* Convert $(93)_{10} = (?)_8$

8 | 93	
8 | 11 - 5	
1 - 3	

$(93)_{10} = (135)_8$

* Convert $(154.125)_{10} = (?)_8$

8 | 154	- $.125 \times 8 = 1.000$
8 | 19 - 2	
2 - 3	

$(154.125)_{10} = (232.1)_8$

* Convert $(23.225)_{10} = (?)_8$

8 | 23	- $.225 \times 8 = 1.800$
2 - 7	- $.800 \times 8 = 6.400$
	- $.400 \times 8 = 3.200$
	- $.200 \times 8 = 1.600$

$(23.225)_{10} = (27.1681)_8$

Scanned with CamScanner

⇒ Conversion from decimal to hexa decimal

* Convert $(31)_{10} = (?)_{16}$

16 | 31	
1 - F	

$(31)_{10} = (1F)_{16}$

* convert $(49)_{10} = (?)_{16}$

16 | 49	
3 - 1	

$(49)_{10} = (31)_{16}$

* convert $(62)_{10} = (?)_{16}$

16 | 62	
3 - E	

$(62)_{10} = (3E)_{16}$

* Convert $(74)_{10} = (?)_{16}$

16 | 74	
4 - A	

$(74)_{10} = (4A)_{16}$

* Convert $(70.250)_{10} = (?)_{16}$

16 | 70	.250 × 16 = 4.000
4 - 6	

$(70.250)_{10} = (46.4)_{16}$

---

* $(92.150)_{10} = (?)_{16}$

16 | 92	.150 × 16 = 2.400
5 - 6	.400 × 16 = 6.400
	.400 × 16 = 6.400
	.400 × 16 = 6.400

$(92.150)_{10} = (5C.2666)_{16}$

---

* Convert $(10.125)_{10} = (?)_{4}$

4 | 10	.125 × 4 = 0.500
2 - 2	.500 × 4 = 2.000

$(10.125)_{10} = (22.02)_{4}$

r to 10 conversions

Binary to decimal conversion

→ $(110000)_2 = (?)_{10}$

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

$\Rightarrow 32+16$

→ $\boxed{(48)_{10}}$

→ $(1000001100)_2$

$1\times2^9+0+1\times2^3+1\times2^2$

$512+8+4$

$\boxed{(524)_{10}}$

→ $(1000111)_2$

$1\times2^6+0+1\times2^2+1\times2^1+1\times2^0$

$64+4+2+1$

$68+3$

$\boxed{(71)_{10}}$

→ $(111001101)_2$

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

$256+128+64+8+4+2+1$

$\boxed{(461)_{10}}$

→ $(10110.0001)_2 = (?)_{10}$

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

$8+2+\frac{1}{2^3} \Rightarrow 10+0.125$

$\boxed{(10.125)_{10}}$

→ $(100111.00011)_2 = (?)_{10}$

$1\times2^5+1\times2^2+1\times2^1+1\times2^0+1\times2^{-3}+1\times2^{-4}$

$\Rightarrow 32+4+2+1+\frac{1}{2^3}+\frac{1}{2^4}$

$\Rightarrow 39+0.125+0.0625 \Rightarrow (39.1875)_{10} \approx \boxed{(39.225)_{10}}$

→ $(111111.1000)_2 = (?)_{10}$

$1\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0+1\times2^{-1}$

$32+16+8+4+2+1+0.5 \Rightarrow \boxed{(63.5)_{10}}$

Octal to decimal conversion:

* $(31)_8 = (?)_{10}$

$8^1 \times 3 + 1 \times 8^0$

$24 + 1$

$\boxed{(25)_{10}}$

* $(175)_8 = (?)_{10}$

$1 \times 8^2 + 7 \times 8^1 + 5 \times 8^0$

$64 + 56 + 5$

$\boxed{(125)_{10}}$

* $(135)_8 = (?)_{10}$

$1 \times 8^2 + 3 \times 8^1 + 5 \times 8^0$

$64 + 24 + 5$

$\boxed{(93)_{10}}$

* $(27.1631)_8 = (?)_{10}$

$2 \times 8^1 + 7 \times 8^0 + 1 \times 8^{-1} + 6 \times 8^{-2} + 3 \times 8^{-3} + 8^{-4} \times 1$

$16 + 7 + 0.125 + 0.09375 + 0.00585 + 0.00024$

$\boxed{(23.2248)_{10}}$

* $(232.1)_8 = (?)_{10}$

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

$128 + 24 + 2 \times \frac{1}{8}$

$154 + 0.125$

$\boxed{154.125}$

* hexadecimal to decimal

→ $(1F)_{16} = (?)_{10}$

$1 \times 16^1 + 15 \times 16^0$

$16 + 15 \Rightarrow \boxed{(31)_{10}}$

→ $(3E)_{16} = (?)_{10}$

$3 \times 16^1 + E \times 16^0$

$48 + 14 \times 1$

$\boxed{(62)_{10}}$

→ $(4A)_{16} = (?)_{10}$

$4 \times 16^1 + A \times 16^0$

$64 + 10 \times 1$

$\boxed{(74)_{10}}$

→ $(31)_{16} = (?)_{10}$

$16^1 \times 3 + 16^0 \times 1$

$48 + 1 = \boxed{49}_{10}$

→ $(46.4)_{16}$

$4 \times 16^1 + 6 \times 16^0 + 4 \times \frac{1}{16}$

$64 + 6 + 0.250 \Rightarrow \boxed{70.250}_{10}$

* $(5C.2666)_{16} = (?)_{10}$

$5 \times 16^1 + 12 \times 16^0 + 2 \times 16^{-1} + 6 \times 16^{-2} + 6 \times 16^{-3} + 6 \times 16^{-4}$

$80 + 12 + 0.125 + 0.0234 + 0.00146 + 0.00009$

$\Rightarrow (92.1498)_{10} \approx \boxed{(92.150)_{10}}$

Scanned with CamScanner

Base 4 to Base 10

* $(22.02)_4$

$2\times4^1 + 2\times4^0 + 0 + 2\times4^{-2}$

$8+2+0+2\times\frac{1}{16}$

$10+0.125$

$(10.125)_{10}$

8-8 Conversions:

In binary to octal conversion segment the given binary number as group and each group should contain 3 bit. Replace each group with its octal equivalent.

3-bit binary code

2^2	2^1	2^0	
4	2	1	
0	0	0	- 0
0	0	1	- 1
0	1	0	- 2
0	1	1	- 3
1	0	0	- 4
1	0	1	- 5
1	1	0	- 6
1	1	1	- 7

$2^3 = 8$

* Convert $(1011010)_2$ to $(?)_8$

Sol

001 | 011 | 010

1 3 2

$(132)_8$

* $(1101011011101)_2$ to $(?)_8$

001 | 101 | 011 | 011 | 101

1 5 3 3 5

$(15335)_8$

Scanned with CamScanner

* $(1110111111110110.0110111)_2 = (?)_8$ (8)

111|011|111|110|110.011|011|100
7 3 7 6 6 . 3 3 4

$(73766.334)_8$

Binary to hexadecimal:-

4 bit binary code

2^3	2^2	2^1	2^0	
8	4	2	1	
0	0	0	0	-0
0	0	0	1	-1
0	0	1	0	-2
0	0	1	1	-3
0	1	0	0	-4
0	1	0	1	-5
0	1	1	0	-6
0	1	1	1	-7
1	0	0	0	-8
1	0	0	1	-9
1	0	1	0	-10-A
1	0	1	1	-11-B
1	1	0	0	-12-C
1	1	0	1	-13-D
1	1	1	0	-14-E
1	1	1	1	-15-F

$2^4 = 16$.

* $(11001010)_2 = (?)_{16}$

1100 | 1010
C | A

$(CA)_{16}$

* $(10101011010101011)_2 = (?)_{16}$

0010 | 1011 | 1010 | 1011
2 | B | A | B

$(2BAB)_{16}$

* $(101101010.10101011)_2 = (?)_{16}$

0001 | 0110 | 1010 . 1010 | 1011 | 1000
1 | 6 | A | A | B | 8

$(16A.ABB8)_{16}$

00010000 00010001

Octal to Binary conversion:

* In octal to Binary conversion each digit of octal number should be replaced with its 3-bit binary equivalent

* $(132)_8$

1 | 3 | 2
001 | 011 | 010

$(001011010)_2$.

* $(16555)_8$

1 | 6 | 5 | 5 | 5
001 | 110 | 101

* $(73766.334)_8$

7 | 3 | 7 | 6 | 6 . 3 | 3 | 4
111 | 011 | 111 | 110 | 110 . 011 | 011 | 100

$(111011111110110.011011)_2$.

Scanned with CamScanner

Hexadecimal to binary:

* $(1011)_{16} = (?)_2$

1	0	1	1
0001	0000	0001	0001

$(0001\,0000\,0001\,0001)_2$

* $(CA)_{16} = (?)_2$

C	A
1100	1010

$(11001010)_2$

Octal to hexadecimal Conversion:

* to convert Octal to hexadecimal convert the octal number to binary and then to hexadecimal.

→ Convert $(145)_8 = (?)_{16}$.

(a) $(145)_8 = (?)_2$

1	4	5
001	100	101

$(0011\,0010\,1)_2$.

(b) 0000 | 0110 | 0101

0 6 5

$(065)_{16} = (65)_{16}$

→ $(1372)_8 = (?)_{16}$

(a) $(1372)_8 = (?)_2$.

001 | 3 | 7 | 2
011 | 111 | 010

$(001011111010)_2$

(b) $(001011111010)_2 = (?)_{16}$.

0010 | 1111 | 1010
2 | F | A

$(2FA)_{16}$

Hexadecimal to octal

→ $(DAC)_{16} = (?)_8$

(a) $(DAC)_{16} = (?)_2$.

D | A | C
1101 | 1010 | 1100

$(110110101100)_2$.

(b) $(110 | 110 | 101 | 100)_2 = (?)_8$
6 | 6 | 5 | 4

$(6654)_8$

→ $(1A8)_{16} = (?)_8$

(a) $(1A8)_{16} = (?)_2$

1 | A | 8
0001 | 1010 | 1000.

$(000110101000)_2$

(b) 000 | 110 | 101 | 000
0 | 6 | 5 | 0

$(650)_8$

*Note

10 − r → Successive divisi
r − 10 → powers multiplicati
r − r → if powers relation exists then grouping and use bit code
else
use intermediate (decimal / binary) to convert

Scanned with CamScanner

Binary addition:-

→ 8 → 1 0 0 0
7 → 0 1 1 1
15 → 1 1 1 1

Here 2 → 1 0
3 → 0 0 1 1
4 → 0 1 0 0
5 → 0 1 0 1
6 → 0 1 1 0
7 → 0 1 1 1

→ 7 → 0 1 1 1
7 → 0 1 1 1
14 → 1 1 1 0

→ 15 → 1 1 1 1
15 → 1 1 1 1
1 1 1 1 0

→ 5 → 0 1 0 1
5 → 0 1 0 1
10 → 1 0 1 0

→ 10 10 1
1 0 1 0
1 1 1 1
0 1 1 1
1 0 0 0 0

→ Binary subtraction:-

8 → 1 0
-4 → 0 0
4 → 0 1 0 0

3 → 0 1 1
07 → 0 1 1 1

10 10
1 0 1 0
0 1 0 1
0 1 0 1

15
2 5
7
18

0 1 1 2
0 1 2 2
0 2
6 → 1 1 0
2 → 0 1 0
4 → 1 0 0

8 → 1 0 0 0
7 → 0 1 1 1
1 0 0 0 1

```
      0  1  1  2  2
      0  1  2  2  2
      0  2
*16   1  0  0  0  0
14    0  1  1  1  0
---------------------
2     0  0  0  1  0
```

```
      0  1  2  0  2
*18   1  0  0  1  0
13    0  1  1  0  1
---------------------
5     0  0  1  0  1
```

Complements:

For any base-'r' system there exists two compliments 1. r's compliment
2. (r-1)'s compliment

* To find r's Compliment the following formula is used i.e., $r^n - N$

where r – Base of number system
n – no. of integer digits in the given number
N – The number for which r's compliment to be calculated.

* To find (r-1)'s compliment the following formula is used i.e., $(r^n - 1) - N$

Compliments in binary number system.

1's compliment

→ Find 1's compliment of 101

Sol:

N = 101
n = 3, r = 2.

Formula: $(r^n - 1) - N$

$(2^3 - 1) - N$
$(8 - 1) - 101$,
$7(111) - 101$
$111 - 101$
010

```
 111
 101
-----
 010
```

1's compliment of 101 is 010

→ Find 1's compliment of 1010

1's compliment of 1010 is 0101

Simply replace 1's with 0 & 0's with 1.

→ 100011

1's compliment is 011100

→ 2's Compliment:

2's compliment can be calculated by adding 1 to 1's compliment of number.

* Find the 2's compliment of following number

→ 1101

1's Compliment = 0010

Add 1 = 1

2's compliment 0011

→ 1000

1s Compliment = 0111

Add 1 = 1

2's compliment 1000

→ 10000

1's compliment = 01111

Add 1 = 1

10000

→ 10110

1's compliment = 01001

01010

By directly 01010

→ 10000

1's compliment = 01111

10000

use not in semexams

* To find 2's compliment directly there are two cases.

(i) if units place have zero then from right to left do not change the number until 1's reached then next numbers are interchanged by 0's as 1 & 1's as 0

(ii) if units place have 1 then not exchange 1 kept it as 1 & then next numbers are interchanged by 0 as 1 & 1 as 0

(i) 1010

2's comp Ans 0110

(ii) 111101

2's comp 000011

Scanned with CamScanner

1's Complement Subtraction

If 'A-B' is the required operation take the 1's complement of B & add it to A. After adding there are two cases

case(i): Carry generated.

If carry generated after the addition of A and 1's complement of B End-around the carry i.e., add carry to the LSB (Least significant bit)

case(ii): Carry not generated

If carry not generated after the addition, take the 1's Complement of result again and put a '-' sign before the result.

Ex:

$9 \rightarrow 1001$

$(-)\ 4 \quad 0100$

$5 \quad 0101$

case(i)

$9 \rightarrow 1001$

1's comp of $4 \rightarrow 1011$

$(+)\ \ 11$

carry $\rightarrow$ 1 0100

$+\ \ 1$

$5 \rightarrow 0101$

case(ii)

$4 \quad 0100$

$9 \quad 0110$

$(+)$

1010

$-0101 \rightarrow$ 1's comp of

$* \ 8 \rightarrow 1000$

$3 \rightarrow 0011$

$(-)$

$5 \rightarrow 0101$

case(i):

$8 \quad 1000$

$-3 \ (+)\ 1100$

$5 \quad$ 1 0100

carry $\rightarrow +\ \ 1$

0101

case(ii):

$3 \quad 0011$

8(1's comp) $\quad 0111$

$1010 \rightarrow$ carry not generated

$-5 \quad -0101$

Scanned with CamScanner

Scanned with CamScanner

* 10 → 1 0 1 0
(-) 7 → 0 1 1 1
3 0 0 1 1

10 → 1 0 1 0
1's comp of 7 (+) 0 0 0
1 0 0 1 0
(+) 1
0 0 1 1

7 → 0 1 1 1
10 0 1 0 1
1 1
1 1 0 0
-0 0 1 1

* 15 → 1 1 1 1
(-) 11 → 1 0 1 1
4 0 1 0 0

15 1 1 1 1
1's comp of 11 (+) 0 1 0 0
1 0 0 1 1
4 +1
0 0 1 0 0

11 1 0 1 1
1's of 15 0 0 0 0
-4 1 0 1 1
-0 1 0 0

* 1 0 1 1 0 1
0 1 0 1 1 0

Case (i)

1 0 1 1 0 1
(+) 1 0 1 0 0 1
1 0 1 0 1 1 0
+1
0 1 0 1 1 1

0 1 0 1 1 0
(+) 0 1 0 0 1 0
1 0 1 0 0 0
(-0 1 0 1 1 1)

2's compliment subtraction:

If "A-B" is the required operation, takes the Two's Compliment of "B" and Add it to 'A'. After addition there are two cases.

Case(1):

Carry generated. After adding 'A' & 2's Comp-liment of 'B" if Carry is generated discard the Carry

Case(2):

Carry is not generated. After adding "A" & 2's compliment of 'B" if Carry is not generated take the 2's Compliment of result again and Put a (-) (minus) sign before it.

Ex:

* 8 → 1 0 0 0 → 1 0 0 0

(-) 5 → 0 1 0 1 → (+) 1 0 1 1

3

discard the carry → 1 0 0 1 1

0 0 1 1

Ans. 3

5 → 0 1 0 1 → 0 1 0 1

(-) 8 → 1 0 0 0 → (+) 1 0 0 0

-3

1 1 0 1

Ans 2's (-0 0 1 1) → -3.

* 9 → 1 0 0 1 → 1 0 0 1

(-) 4 → 0 1 0 0 → (+) 1 1 0 0

5.

discard → 1 0 1 0 1

0 1 0 1 → 5.

4 → 0 1 0 0 → 0 1 0 0

(-) 9 → 1 0 0 1 → 0 1 1 1

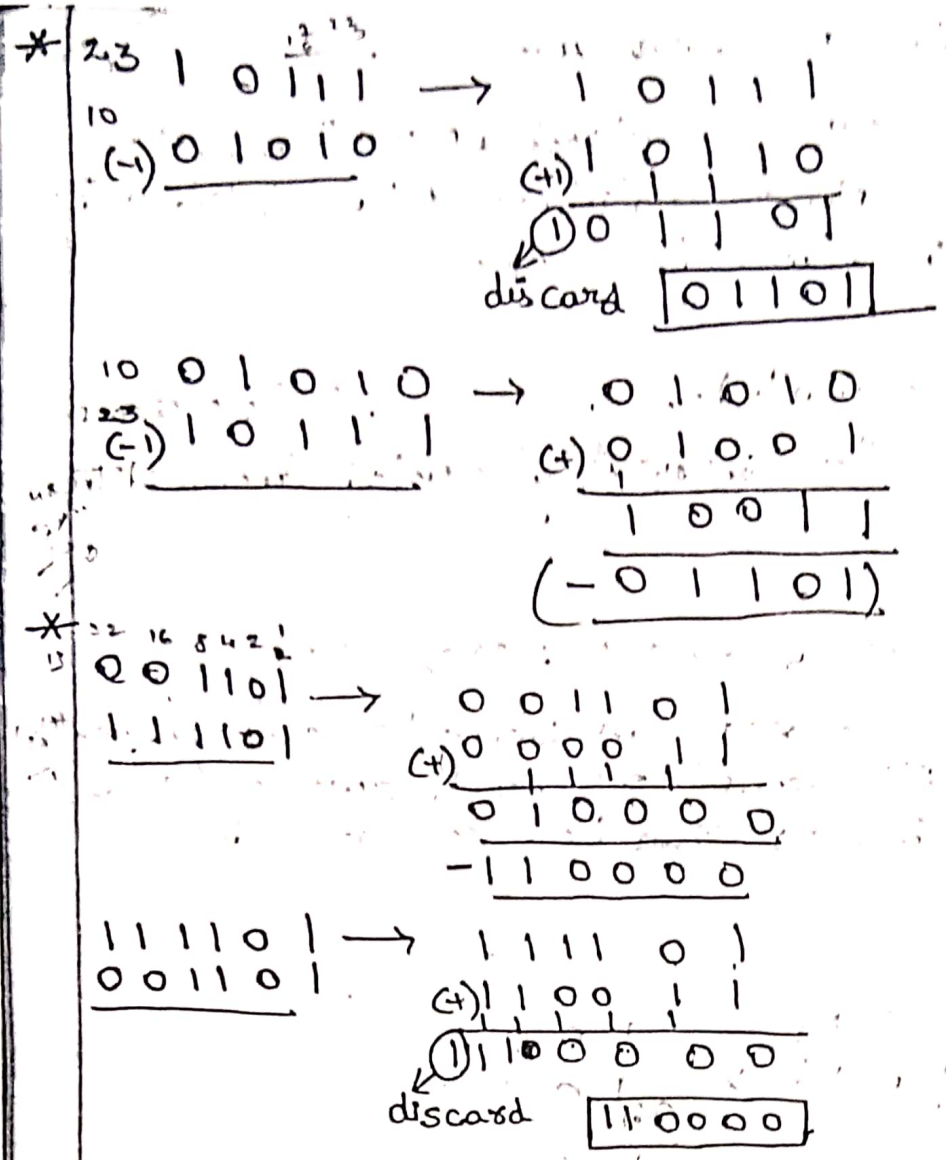
-5

1

1 0 1 1

2's (-0 1 0 1) → -5

Scanned with CamScanner



9's Complement Subtraction:-

If 'A-B' is the required operation take the 9's complement of B & add it to A. After adding there are two cases

Case1: Carry generated.

If Carry generated after the addition of A and 9's Complement of B. End around the carry i.e., add carry to LSB

Case2: Carry not generated

If Carry not generated after the addition take the 9's Compliment of result again and put a '-' sign before the result.

(19)

* 8 → 8 | 9 / −5 / 4
−5 9's comp → 4 / 12 / +1 → end around carry / 3

5 → 5
−8 9's comp → 1 / 6
9's of 6 → −3

9 / −8 / 1 — 9 / −6 / 3

* 9 → 9 | 9 / 4 / 5
−4 9's comp → (+)5 / 14 / +1 / 5

4 → 4
−9 9's → (+)0 / 4
9's of 4 → −5

9 / −4 / 5

99 / 19 / 80

* 25 → 25
−19 9's comp → (+)80 / 105 / 6

19 → 19
−25 9's comp → (+)74 / 93
→ −6

99 / 25 / 74 — 99 / −93 / 6

* 84 → 84 | 99 / 27 / 72
−27 9's comp → (+)72 / 156 / +1 / 57

27 → 27
(−)84 → (+)15 / 42
9's of 42 → −57

99 / 84 / 15 — 99 / 42 / 57

1c

* 459 → 459 | 999 / 362 / 637
(−)362 9's of → (+)637 / 1096 / +1 / 097

362 → 362 | 999 / 459 / 540
−(459) → (+)540 / 902
9's of 902 → −97

999 / 902 / 97

10's Complement Subtraction

If "A - B" is the required operation, takes 10's Complement of B and add it to A. After addition there are two cases:

Case(i):- Carry generated

If Carry generated after adding A & 10's Complement of B, if carry generated then discard the Carry

Case(ii):- Carry not generated.

After adding A & 10's Complement of B if Carry not generated take 10's Complement of result again and put a (-) minus sign before it.

* 9 − 3 → 9 + (10's of 3) = 9 + 7 = 16 → discard → 6

10's comp of 3: 9 − 3 + 1 = 7

* 3 − 9 → 3 + (10's of 9) = 3 + 1 = 4 → 10's of 4 = −6

10's comp of 9: 9 − 9 = 0, + 1 = 1

* 8 − 4 → 8 + (10's of 4) = 8 + 6 = 14 → discard → 4

10's comp of 4: 9 − 4 + 1 = 6

* 4 − 8 → 4 + (10's of 8) = 4 + 2 = 6 → 10's of 6 = −4

10's comp of 8: 9 − 8 = 1, + 1 = 2

* 56 − 43 → 56 + (10's of 43) = 56 + 57 = 113 → discard → 13

10's comp of 43: 99 − 43 = 56, + 1 = 57

43 − 56 → 43 + (10's of 56) = 43 + 44 = 87 → 10's of 87 = −13

10's comp of 56: 99 − 56 = 43, + 1 = 44

99 − 87 = 12, + 1 = 13

Scanned with CamScanner

* 546 → 546   999
− 307 → 10's of 307 → 692   −307 = 692, +1 → 693
546 + 693 = (1)239 → 239

307 → 307
−546 → 10's of 546 → 454
307 + 454 = 761 (no carry)
10's of 761 → −239

(−) 999 − 546 = 453, + 1 = 454
999 − 761 = 238, + 1 = 239

Binary codes:

Binary codes are classified into 6 types

1. Weighted Codes
2. Non-weighted Codes
3. Sequential codes
4. Reflective codes
5. Alphanumeric codes
6. Error Correcting & detecting codes

Weighted Codes:-

In weighted codes each place value is having a specific weight, Based on the weights, the codes are derived.

Ex: 8421, 2421, 5411 etc....

Non-weighted codes.

These are the codes derived from weighted codes and no specific weight for place value

Ex: Gray code, Parity, excess 3 code etc...

Sequential codes:

(4211) → not sequential

In sequential codes the weights are continously increasing from LSB to MSB

Ex: 8421.

Reflective codes.

In these codes zero is the 1's compliment of 9, 1 is the 1's compliment of 8

(P.T.O)

2 is the 1's compliment of 8 and so on.

Ex: 2421

→ Write the 2421 code

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

Alphanumeric codes:

In Alphanumeric codes, each alphabet (both uppercase/lowercase), numbers and symbols are having specific binary representation

Ex: ASCII code

American Standard Code for Information Interchange

Error Correcting & Error detecting Codes

By using some of the binary codes errors can be detected but not corrected they are called error detecting codes.

By using some of the binary codes errors can be detected & corrected they are called error correcting code.

Ex: Hamming code.

BCD Code: (Binary coded decimal)

In BCD code all the decimal digits from 0 to 9 are coded with binary numbers as shown below

	8	4	2	1
0 — 0	0	0	0	0
1 — 0	0	0	0	1
2 — 0	0	1	0	
3 — 0	0	1	1	
4 — 0	1	0	0	
5 — 0	1	0	1	
6 — 0	1	1	0	
7 — 0	1	1	1	
8 — 1	0	0	0	
9 — 1	0	0	1	

BCD addition

In BCD addition, after adding two BCD numbers, the result should be valid BCD number. If it is invalid add correction factor 6 (0110) to the invalid BCD number.

* 5 → 0 1 0 1
3 → (+) 0 0 1 1
8 → 1 0 0 0

* 5 → 0 1 0 1
5 → (+) 0 1 0 1
10 → 1 0 1 0 → Invalid BCD.
+6 → + 0 1 1 0
0001 0 0 0 0
1 0

* 8 → 1 0 0 0
+6 → + 0 1 1 0
14 → 1 1 1 0 → Invalid BCD
+6 → + 0 1 1 0
0001 0 1 0 0
1 4

* 15 → 0 0 0 1 0 1 0 1
12 → + 0 0 0 1 0 0 1 0
27 → 0 0 1 0 0 1 1 1
2 7

* 39 → 0 0 1 1 1 0 0 1
(+) 23 → + 0 0 1 0 0 0 1 1
62 → 0 1 0 1 1 1 0 0 → Invalid BCD.

0110 0010 +6 0 1 1 0
6 2
0 1 1 0 0 0 1 0
6 2

* 28 → 0010 1000
  26 → (+) 0010 0110
  54 → 0100 1110 → Invalid BCD

0101 0100 +6
5 4

0110
0101 0100
5 4

* 45 → 0100 0101
  38 → 0011 1000
  83 → 0111 1101 → Invalid BCD

1000 0011 +6
8 3

0110
1000 0011
8 3

* 157 → 0001 0101 0111
  138 → 0001 0011 1000
  295 → 0010 1000 1111

0010 1001 0101 +6
2 9 5

0110
0010 1001 0101
2 9 5

* 867 → 1000 0110 0111
  574 → 0101 0111 0100
  1441 → 1101 1101 1011

0001 0100 0100 0001 +6

0110
1101 1110 0001

+6

0110
1110 0100 0001

+6

0110
0001 0100 0100 0001
1 4 4 1

BCD Subtraction:-

If "(A-B)" is required operation, take the 9's Complement of "B" and add it to A. After adding there are two cases.

Case 1:- Invalid BCD

If the sum is invalid BCD number validate it by adding 6 (0110). After addition of 6, the carry has to be end arounded.

Case 2:- Valid BCD.

After addition, if the result is valid BCD, take the 9's compliment of result. once again & put (-)minus sign before it.

Ex:

① 8 → 1000

(-)4 → 9's of 4 → 0101

4

0100

(+)

1101 → Invalid BCD

+6 0110

1 0011

→]

+ 1 1

0100

4 → 0100

-8 → 9's of 8 → 0001

-4

-0100

0101 → valid BCD

9's of result → 0100

② 9 → 1001

-3 → 9's of 3 → 0110

6

0110

1111 → Invalid BCD

+6 0110

1 0101

→ 1

+ 1

0110

9 / -3 / 6

8

3 → 0011

-9 → 9's of 9 → 0000

-6

(+)

0011

9 / -3 / 6

9's of result is -0110

Scanned with CamScanner

*25 → 0010 0101

-13 9's of 13 → (+) 1000 0110

12 1010 1011

0001 0010 +6 0110

1011 0001

+6 0110

1001 0001

→ +1

0001 0010

99 − 13 = 86 (27)

99 − 87 = 12

99 − 25 = 74

13 → 0001 0011

-25 0010 0101 + 0111 0100

-12 1000 0111

9's of result → −0001 0010

*257 → 0010 0101 0111

-143 (+) 0001 0100 0011

114 1010

999 − 143 = 856

0001 0001 0100

0010 0101 0111

+ 1000 0101 0110

1010 1010 1101

+6 0110

1010 1011 0011

+6 0110

1011 0001 0011

+6 0110

1000 0001 0011

+1

0001 0001 0100

Scanned with CamScanner

143 → 0001 0100 0011

-257 → 0111 0100 0010

-114 → + 1 1 1 1 1 1

1000 1000 0101

999
257
742

999
885
114

9's of 885 is 0001 0001 0100

Gray code:

Gray code is a non weighted code, derived from binary code. In gray code there is only one bit change from one number to the immediate next number.

Q. Generate 4-bit gray code from single bit gray code.

Ans

Q - 0 → 0 Q
1 - 1 → 1 Q

0 - 0
1 - 1

0-00
1-01
2-11
3-10

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

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

Binary to Gray code:

To convert given 4-bit binary to its equivalent 4-bit gray code, the following formula is used.

Let the given 4 bit binary number is $(B_3 B_2 B_1 B_0)$ and its equivalent 4 bit gray code is $(G_3 G_2 G_1 G_0)$ then

$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

$G_1 = B_2 \oplus B_1$

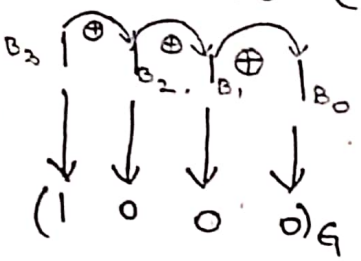
$G_0 = B_1 \oplus B_0$

EX-OR truth table

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

→ Convert $(111)_B = (?)_G$

Ans.



$(1\ 0\ 0\ 0)_G$

→ $(0111)_B = (?)_G$

Ans.

0 1 1 1

↓ ↓ ↓ ↓

$(0\ 1\ 0\ 0)_G$

→ $(1001)_B = (?)_G$

1 0 0 1

↓ ↓ ↓ ↓

$(1\ 1\ 0\ 1)_G$

→ $(1110)_B = (?)_G$

1 1 1 0

↓ ↓ ↓ ↓

$(1\ 0\ 0\ 1)_G$

→ $(101110)_B = (?)_G$

1 0 1 1 1 0

↓ ↓ ↓ ↓ ↓ ↓

1 1 1 0 0 1

$(111001)_G$

Gray code to Binary code Conversion:

Let the given 4-bit gray code is $(G_3G_2G_1G_0)$ and it's equivalent binary is $(B_3B_2B_1B_0)$ Then

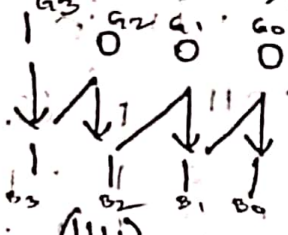
$$B_3 = G_3$$

$$B_2 = G_3 \oplus B_3 \oplus G_2$$

$$B_1 = B_2 \oplus G_1$$

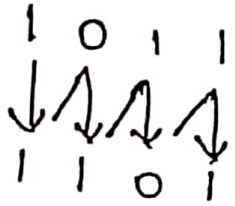
$$B_0 = B_1 \oplus G_0$$

→ Convert $(1000)_G = (?)_B$



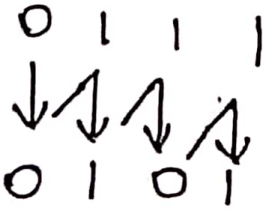
$(1111)_B$

→ $(1011)_G = (?)_B$



$(1101)_B$

→ $(0111)_G = (?)_B$



$(0101)_B$

Excess-3 Code:

Excess-3 code is a non weighted code derived from BCD code for every BCD number 3 is added to get excess 3-code

BCD	Excess-3
0 - 0000	→ 0011
1 - 0001	→ 0100
2 - 0010	→ 0101
3 - 0011	→ 0110
4 - 0100	→ 0111
5 - 0101	→ 1000
6 - 0110	→ 1001
7 - 0111	→ 1010
8 - 1000	→ 1011
9 - 1001	→ 1100

Excess-3 addition:

In excess-3 addition after adding two excess-3 numbers there are two cases

Case(i): No Carry

If there is no Carry after excess-3 addition subtract (0011)3 from the result To get final result

Case(ii): Carry generated:

After excess-3 addition if Carry generated add 3(0011) to both sum & carry.

Ex:

$$
\begin{array}{rl}
3 \rightarrow & 0110 \\
(+)\ 4 \rightarrow & (+)\ 0111 \\
\hline
7 & 1101 \\
(1010) & 0011\ \text{(sub-3)} \\
\hline
 & 1010
\end{array}
$$

2 → 0101
3 → 0110
5 → 1011
1000 0011 (Sub 3)
1000

6 → 1001
7 → 1010
13 → 0001 0011
0011 0011 Add 3
0100 0110
1 3

13 → 0100 0110
12 → 0100 0101
1000 1011
0101 1000
2 5
0011 Sub(3)
1000 1000
Sub 3
0011
0101 1000
2 5

38 → 0110 1011
25 → 0101 1000
63 → 1100 0011
1001 0110
0011 0011 Sub 3
1001 0110

45 → 0111 1000
36 → 0110 1001
81 → 1110 0001
1011 0100
8 1
0011 Add 3
1110 0100

Sub3 0011
1011 0100

93 → 1100 0110
48 → 0111 1001
141 → 0100 0111 0100
0011 0011 Add (3)
0100 0111 0100

Excess-3 Subtraction:

In excess-3 subtraction if A-B is the required operation then take the 9's (or 1's complement for excess 3 only) complement of B and add it to A. After addition there are two cases..

Case 1: Carry generated.

After addition if Carry generated, end-around the carry and add 3 (0011) to the sum to get the final answer.

Case 2: Carry not generated.

After addition if Carry is not generated subtract 3 from the sum and once again take the 9's complement (or 1's complement for excess 3 only) of the result and put (-) sign before the result.

* Excess-3 is a self complement code, Because 9's compliment of 1's complement is same.

Ex:

* 6 → 1001 — 9's of 3 → 6

−3 → 1001

3 → 1001 0

0110 ← (end-around) + 1

0011 0

0011 Add 3 (0011)

0110

3 → 0110

−6 → 9's of 6 → + 0110

−3 → 1100

−0110 → 0011 Sub 3 (0011)

9's of result → 1001 → −0110

*5

-4 9's → 1 0 0 0 (+) 1 0 0 0 → 1 0 0 0 0

9
4
5

1

0100 → end around carry.

0 0 0 1

0 0 1 1 | Add 3 (0011)

0 1 0 0

4 → 0 1 1 1

-5 → 0 1 1 1

9
5
4

-1

1 x_0 $x_{\phi 2}$ ϕ_2

0100 → 0 0 1 1 | Sub 3 (0011)

1 0 1 1

9's of result (0 1 0 0)

* 26 → 0101 1001 → 0101 1001

-14 9's → 0100 0111 1's comp → 1011 1000

(+) 1 1 1 1

0001 0001 0001

0100 0101

0001 0010

* 14 → 0100 0111

-26 9's → 1010 0110

(+) 1 1

1110 1101

0011 (sub 3)

1 x_1 $x_{\phi 2}$ 1010

0011

Sub 3

1 0 1 1 1 0 1 0

9's of result (-0100 0101)

1 2

99
26
73

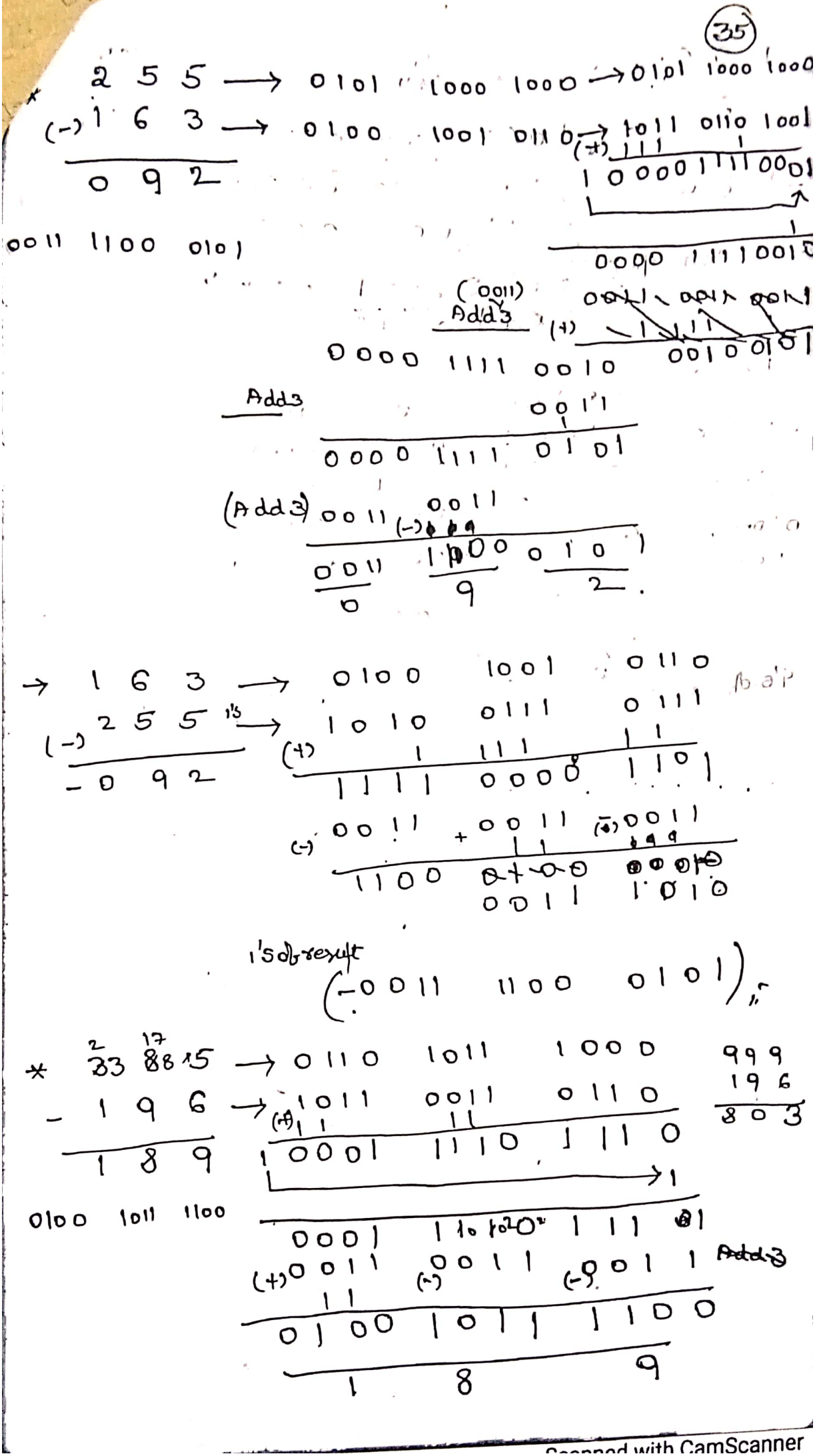
(+) 0 0 1 1 (0011) Add 3

0001 0101

0011 Add 3

0100 0101

1 2



196 → 0100 1100 1001

-385

-189

0100 1011

1100

1001 0100 0111

1 1 1 1 1

1110 0001 0000

(-) 0011 (+) 0011 (+) 0001

1011 0100 0011

0100 1011 1100

1 8 9

9's of result:

Signed Binary numbers:

In signed binary numbers, one sign bit is allocated for representation of +ve numbers and -ve numbers. For +ve numbers sign bit is 0. For -ve numbers the sign bit is 1.

For Ex:

sign bit

0 0001 = +1

1 0001 = -1

* The signed binary numbers can be represented in three ways.
  (1) sign magnitude representation
  (2) 1's compliment "
  (3) 2's Compliment "

Note:
***In all three representations, +ve numbers are having unique representation, where as -ve numbers having different representations

Ex:

	+5	-5
sign magnitude	0 0000101	1 0000101
1's compliment	0 0000101	1 1111010
2's compliment	0 0000101	1 1111011

Ex: using 2's Compliment

+7 = 00000111
+5 = 00000101
-5 = 11111011
-7 = 11111001
+2 = 00000010
-2 = 11111110
+12 = 00001100
-12 = 11110100

+7 → 00000111
+5 → 00000101 (+)
+12 00001100

+7 → 00000111
-5 → 11111011 (+)
+2 discard 1 00000010

-7 → 11111001
+5 → 00000101 (+)
-2 11111110

Scanned with CamScanner

-7 → 11111001
-5 → 11111011
-12 (+) 111 11
discard (1) 11110100

→ some above question using 1's compliment

* Using 1's Compliment.

+7 → 00000111
+5 → 00000101
-7 → 11111000
-5 → 11111010
+2 → 00000010
-2 → 11111101
+12 → 00001100
-12 → 11110011

+7 → 00000111
+5 → (+) 00000101
+12 00001100

+7 → 00000111
-5 → (+) 11111010
+2 100000001
+1
00000010

-7 → 11111000
+5 → (+) 00000101
-2 11111101

-7 → 11111000
-5 → 11111010
-12 11110010
+1
11110011

ERROR DETECTING CODES:

Parity bit is an extra bit which is added to the original bit. Parity is of Two types
1. Even Parity
2. odd Parity.

Even Parity: For even Parity the no. of 1's in the information should be even including Parity bit.

Odd Parity: For odd Parity the no. of 1's in information is odd including Parity bit.

Ex: Generate Even & odd Parity for given message

Ans → 0 0 1 0 1 1 0

0 0 1 0 1 1 0 (1) → even Parity
0 0 1 0 1 1 0 (0) → odd parity.

Block Parity: In block Parity the Parity is taken for both rows & columns for group of messages.

Ex:

Even parity	0 0 1 0 1 0	0
	0 0 1 0 1 0	0
	0 0 1 0 1 1	1
	1 0 1 0 1 0	1
	1 0 0 0 0 1	

Error detection & correction codes:

In the error detection & correction codes Hamming code is the best example. By using hamming code, error can be detected and it can be corrected.

Hamming code generation:

To generate hamming code for 'm' no. of message bits the following formula is used

$$2^P \geq m+p+1$$

where p = no. of parity bits

Generate Hamming code for 1011.

Ans:

1 0 1 1
m_1 m_2 m_3 m_4

$m = 4$

$2^P \geq m+p+1$

$2^P \geq 4+p+1$ [$\because m=4$]

if p = 0

$2^0 \geq 4+0+1$

$1 \geq 5$

if p = 1

$2^1 \geq 4+1+1$

$2 \geq 6$

if p = 2

$2^2 \geq 4+2+1$

$4 \geq 7$

if p = 3

$2^3 \geq 4+3+1$

$8 \geq 8$

$m = 4$

$p = 3$

Total = 7

(4)

1	2	3	4	5	6	7
P_1	P_2	m_1	P_3	m_2	m_3	m_4
		1		0	1	1

0 01-1
0 10-2
0 11-3
1 00-4
1 01-5
1 10-6
1 11-7

1,3,5,7 0 1 0 1

2,3,6,7 1 1 1 1

4,5,6,7 0 0 1 1

(0110011)

0010:

1	2	3	4	5	6	7
P_1	P_2	m_1	P_3	m_2	m_3	m_4
		0		0	1	0

0010
1001
1010
1110

1,3,5,7 0 0 0 0

2,3,6,7 1 0 1 0

4,5,6,7 1 0 1 0

(0101010)

1001:

1	2	3	4	5	6	7
P_1	P_2	m_1	P_3	m_2	m_3	m_4
		1		0	0	1

1,3,5,7 0 1 0 1

2,3,6,7 0 1 0 1

4,5,6,7 1 0 0 1

(0011001)

1010:

	1	2	3	4	5	6	7
	p_1	p_2	m_1	p_3	m_2	m_3	m_4
			1		0	1	0
1,3,5,7	1		1		0		0
2,3,6,7		0	1			1	0
4,5,6,7				1	0	1	0

(1011010)

1110:

	1	2	3	4	5	6	7
	p_1	p_2	m_1	p_3	m_2	m_3	m_4
			1		1	1	0
1,3,5,7	0		1		1		0
2,3,6,7		0	1			1	0
4,5,6,7				0	1	1	0

(0010110)

Error detection and correction:

→ original: 0101010 ⇒ 0100010

	1	2	3	4	5	6	7
	0	1	0	0	0	1	0
C_1 → 1,3,5,7	0		0		0		0 → 0
C_2 → 2,3,6,7		1	0				1 0 → 0
C_3 → 4,5,6,7				0	0	1	0 → 1

C_3 C_2 C_1

1 0 0 ⇒ 4

∴ there is an error in 4th position.

the correct information is 0101010

(4.3)

⇒ 0101010 $\overset{\text{error}}{\Longrightarrow}$ 0101000

	1	2	3	4	5	6	7	
	0	1	0	1	0	0	0	
$C_1 \rightarrow 1,3,5,7 \rightarrow$	0		0		0		0	$\rightarrow 0$
$C_2 \rightarrow 2,3,6,7 \rightarrow$		1	0			0	0	$\rightarrow 1$
$C_3 \rightarrow 4,5,6,7 \rightarrow$				1	0	0	0	$\rightarrow 1$

C_3 C_2 C_1

1 1 0 $\rightarrow 6$

∴ There is an error in 6th position.
The correct information is 0101010

⇒ 0101010 $\overset{\text{error}}{\Longrightarrow}$ 0001010

	1	2	3	4	5	6	7	
	0	0	0	1	0	1	0	
$C_1 \rightarrow 1,3,5,7 \rightarrow$	0		0		0		0	$\rightarrow 0$
$C_2 \rightarrow 2,3,6,7 \rightarrow$		0	0			1	0	$\rightarrow 1$
$C_3 \rightarrow 4,5,6,7 \rightarrow$				1	0	1	0	$\rightarrow 0$

C_3 C_2 C_1

0 1 0 $\rightarrow 2$

∴ There is an error in 2nd position, the Correct information is 0101010.

⇒ 0101010

	1	2	3	4	5	6	7	
	0	1	0	1	0	1	0	
$C_1 \rightarrow 1,3,5,7$	0		0		0		0	$\rightarrow 0$
$C_2 \rightarrow 2,3,6,7.$		1	0			1	0	$\rightarrow 0$
$C_3 \rightarrow 4,5,6,7.$				1	0	1	0	$\rightarrow 0$

C_3 C_2 C_1

0 0 0

No error.

Scanned with CamScanner

Logic Gates:

(1) Basic gates → AND, OR, NOT
(2) Universal gates → NAND, NOR
(3) Special gates → EX-OR, EX-NOR

AND Gate

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

$Y = A \cdot B$
$= AB$

OR Gate:

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

$Y = A + B$

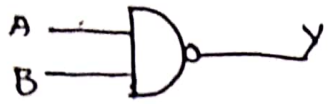
NOT Gate:

A	Y
0	1
1	0

$Y = \overline{A} = A'$

NAND Gate :

NAND Gate is AND gate followed by NOT Gate



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

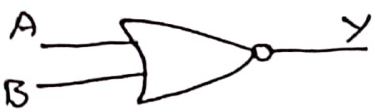
$= \overline{AB}$

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

Any boolean function can be purely implemented with this two gates (NAND, NOR)

NOR Gate :

NOR Gate is OR Gate followed by NOT gate.

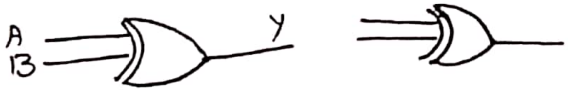


$Y = \overline{(A+B)}$

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

Special gates:

→ EX-OR: (inequality gate)



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

$Y = A \oplus B$

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

Important:

Property: $A \oplus 0 = A$

$A \oplus 1 = \bar{A}$

Ex-NOR (equality gate)

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

$$Y = A \odot B$$

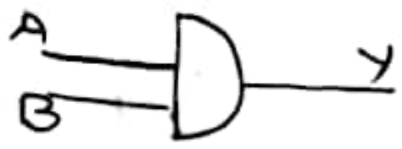
$$= A\,B + \overline{A}\,\overline{B}$$

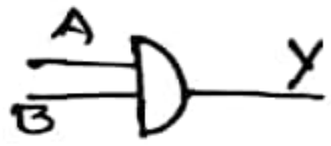
$$= AB + \overline{A}\,\overline{B}$$

Logic Gates:

(1) Basic gates → AND, OR, NOT
(2) Universal gates → NAND, NOR
(3) Special gates → EX-OR, EX-NOR

AND Gate





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

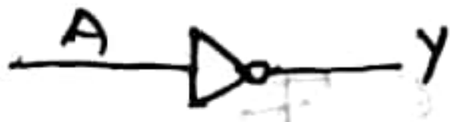
$Y = A \cdot B$
$= AB$

OR Gate:

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

$Y = A + B$

NOT Gate:

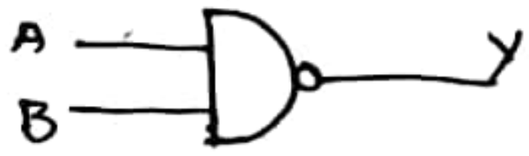


A	Y
0	1
1	0

$Y = \overline{A} = A'$

NAND Gate :

NAND Gate is • AND gate followed by NOT Gate



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

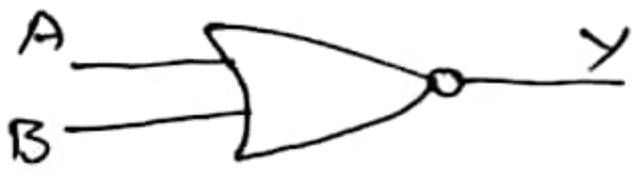
$= \overline{AB}$

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

Any boolean function can be purely implemented with this two gates (NAND, NOR)

NOR Gate :

NOR Gate is OR Gate followed by NOT gate.



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

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

Special gates:

→ EX-OR: (inequality gate)

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

$Y = A \oplus B$

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

Important:

Property: $A \oplus 0 = A$

$A \oplus 1 = \bar{A}$

Ex-NOR (equallity gate)

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

$Y = A \odot B$

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

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

MINIMIZATION TECHNIQUES

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

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

→

Boolean theorems :-

1. Complementation laws :-

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

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

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

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

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

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

2. AND laws :-

Law 1 = $A \cdot 0 = 0$

Law 2 = $A \cdot 1 = A$

Law 3 = $A \cdot A = A$

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

3. OR laws :-

Law 1 = $A + 1 = 1$

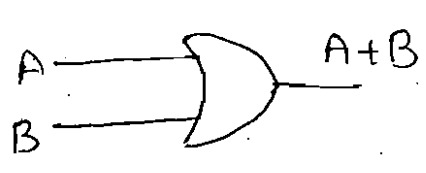
Law 2 = $A + 0 = A$

Law 3 : $A + A = A$

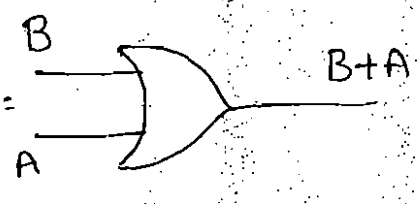
Law 4 : $A + \overline{A} = 1$

4. Commutative laws :-

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



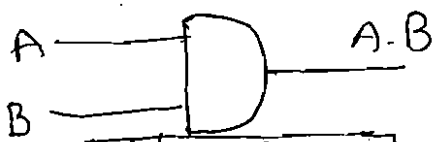
=



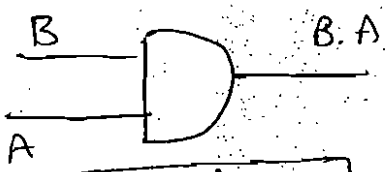
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

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

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



=

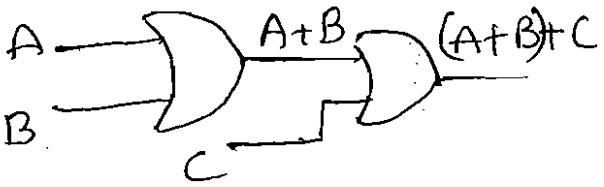


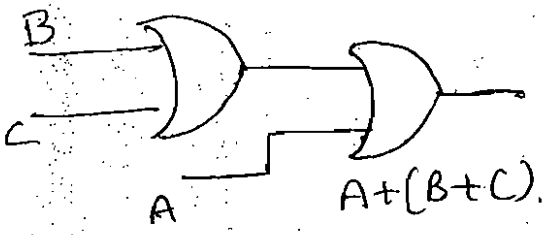
A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	B·A
0	0	0
0	1	0
1	0	0
1	1	1

5. Associate laws :-

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



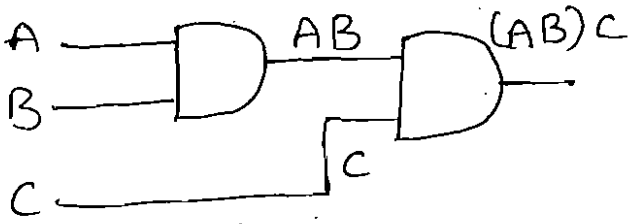


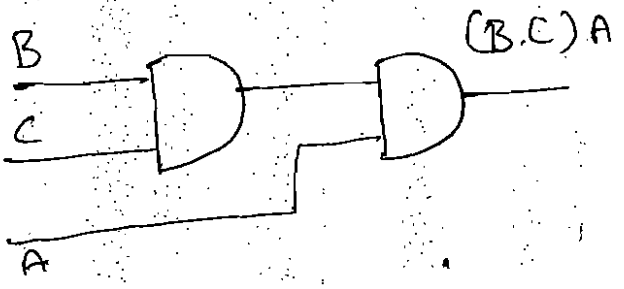
A B

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

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

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



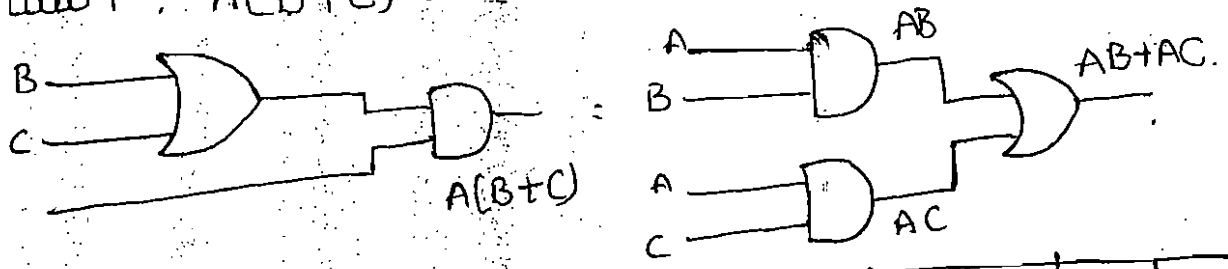


A	B	C	AB	(AB) C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

A	B	C	B.C	A(BC)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

6. Distributive Laws :-

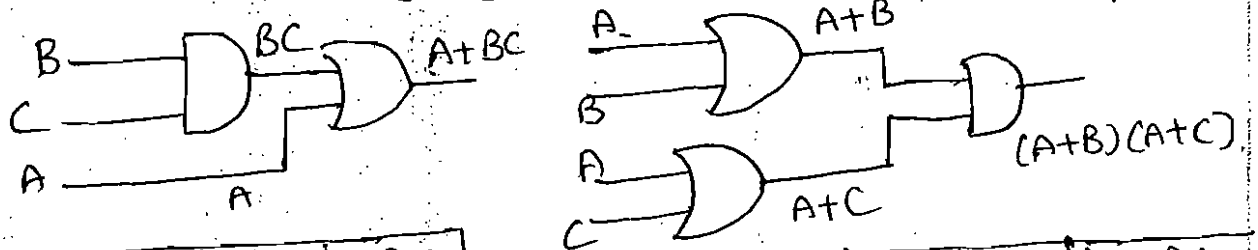
Law 1 : $A(B+C) = AB + AC$



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

A	B	C	AB	AC	AB+AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	1	1	1

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



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

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

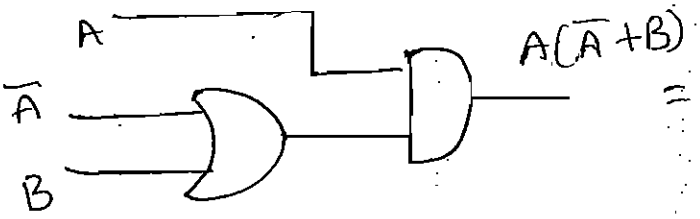
7. Redundant literal Rule :-

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

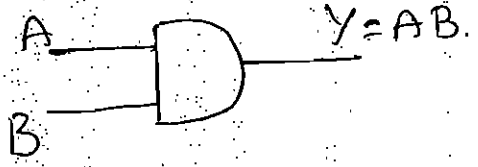
$= A(\bar{A}+B)$

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

$= AB$



=



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

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

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

$= A+\bar{A}B$

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

$= (A+B)$

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

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

8. Idempotence laws :-

Idempotence means the same value.

Law 1 = $A \cdot A = A$

if A=0 then 0.0 = 0

if A=1 then 1.1 = 1

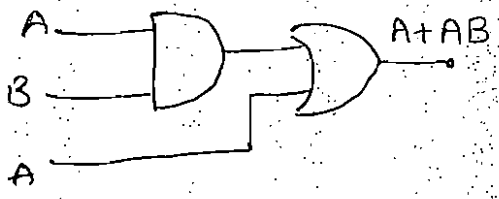
Law = $A+A = A$

if A=0 then 0+0 = 0

if A=1 then 1+1 = 1

9. Absorption Laws :-

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



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

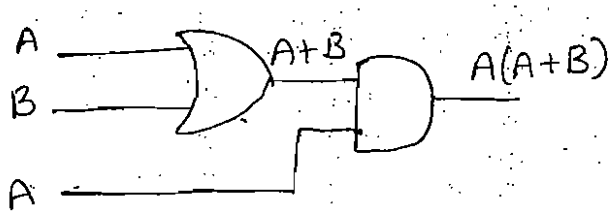
$= A+A\cdot B$

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

$= A\cdot 1$

$= A.$

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



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

$= A\cdot A + AB$

$= A + AB$

$= A(1+B)$

$= A.$

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

$= A$

10. Consensus Theorem :-

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

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

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

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

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

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

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

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

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

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

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

$AB + \bar{A}C$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Transposition Theorem :-

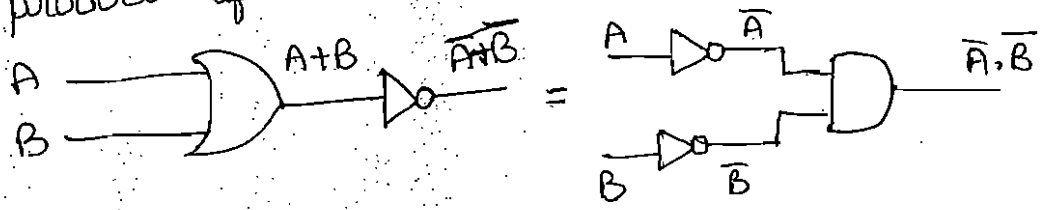
Theorem : $AB + \bar{A}C = (A+C)(\bar{A}+B)$

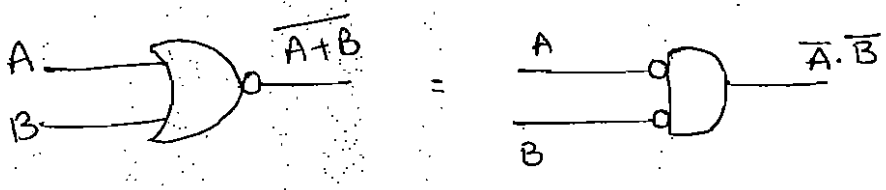
$$R.H.S = (A+C)(\bar{A}+B)$$
$$= A.\bar{A} + AB + \bar{A}C + BC$$
$$= AB + \bar{A}C + BC$$
$$= AB + \bar{A}C + BC(A+\bar{A})$$
$$= AB + \bar{A}C + ABC + \bar{A}BC$$
$$= AB(1+C) + \bar{A}C(1+B)$$
$$= AB + \bar{A}C$$

Demorgan's Theorem :-

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

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





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

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

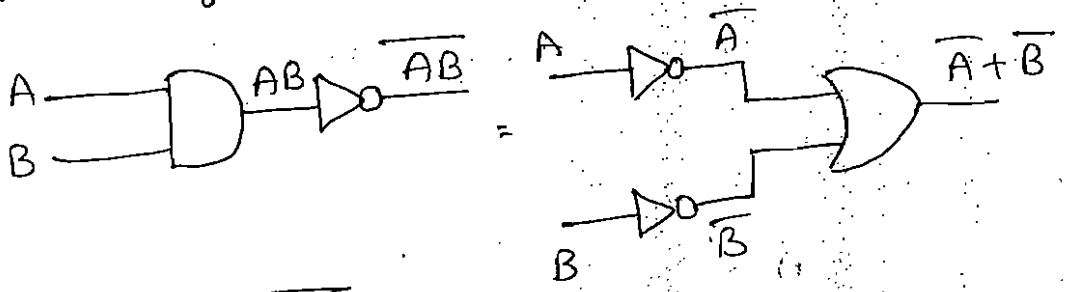
Similarly → $\overline{A+B+C+D+E} = \overline{A}\,\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}$

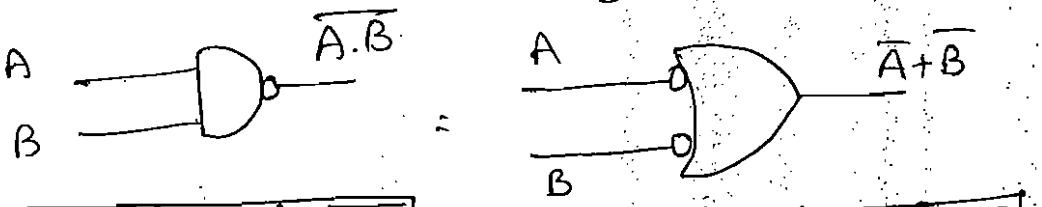
→ $\overline{\overline{A}B+C\overline{D}+\overline{E}\overline{D}} = (\overline{\overline{A}B})(\overline{C\overline{D}})(\overline{\overline{E}\overline{D}})$

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

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

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





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

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

Duality:-

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

→ $A.(A+B) = A.B$ → $A+A+B = A+B$.

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

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

Reducing Boolean expressions by using Boolean theorems :-

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

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

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

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

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

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

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

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

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

$= AB$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$A \quad B$

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

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

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

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

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

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

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

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

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

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

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

$= A[B \odot C]$

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

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

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

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

$= B + BD + BC + BCD$

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

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

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

Applying the consensus theorm to 2nd and 4th terms

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

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

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

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

K-map (Karnaugh-map) :-

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

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

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

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

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

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

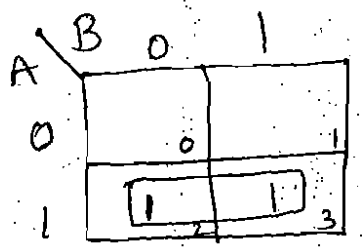
→ Each of the product terms in the standard sop form is called minterms and each of the sum terms in the standard pos form is called maxterms.

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

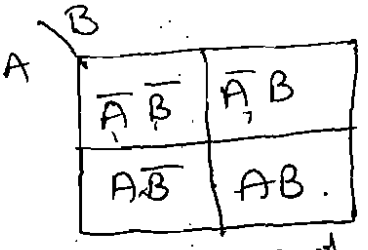
A two-variable K-map has $2^2 = 4$ cells or squares. 4 possible combinations of the input variables A and B. ($\bar{A}\bar{B}$, $\bar{A}B$, $A\bar{B}$, AB).

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

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



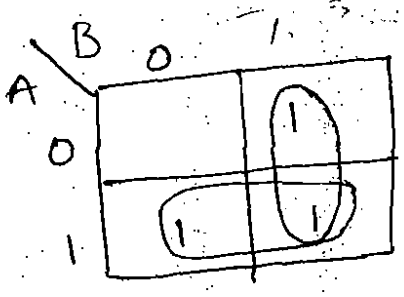
$f = B$.



The minterms of a two-variable K-map.

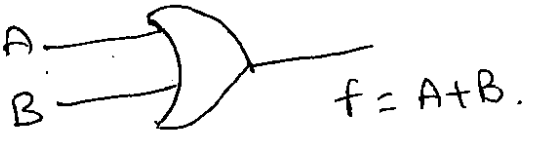
* Reduce the expression by using K-map.

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



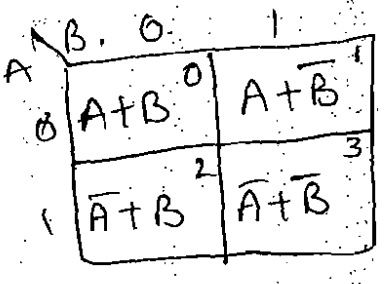
$f = A + B$

logic diagram



mapping of POS Expressions :-

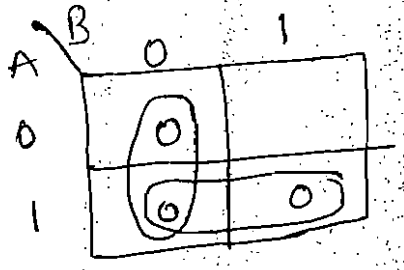
$M_0 = A + B$, $M_1 = A + \bar{B}$, $M_2 = \bar{A} + B$, $M_3 = \bar{A} + \bar{B}$

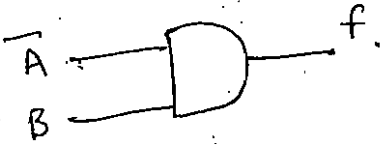


The maxterms of a two-variable K-map.

→ Reduce the expression by using k-map.

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





$f = \bar{A}.B$

Three- variable K-map :-

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

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

minterms

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

max terms.

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

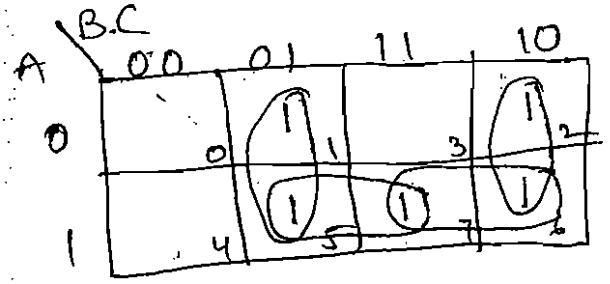
$\bar{A}\bar{B}C = 001 = m_1$,

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

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

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

$ABC = 111 = m_7$



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

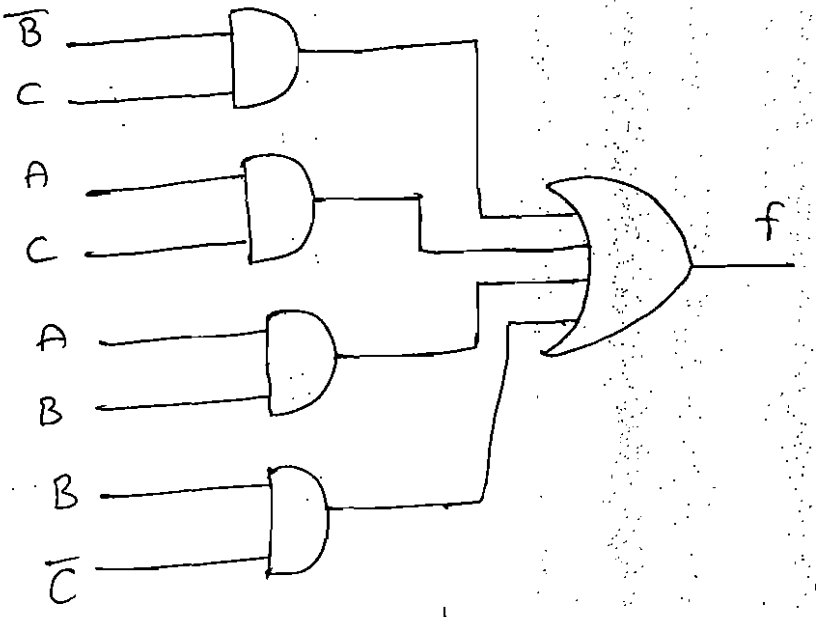
$f_2 = m_5 + m_7 = A\bar{B}C + ABC = AC(B+\bar{B}) = AC$

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

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

$f = f_1 + f_2 + f_3 + f_4$

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



Logic diagram

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

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

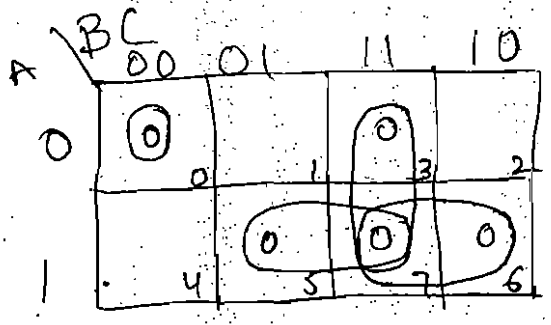
$A+B+C = 000 = M_0$

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

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

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

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



$f_1 = M_0 = A+B+C$

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

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

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

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

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

$f_3 = M_3 \cdot M_7$

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

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

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

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

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

$f_4 = M_7 \cdot M_6$

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

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

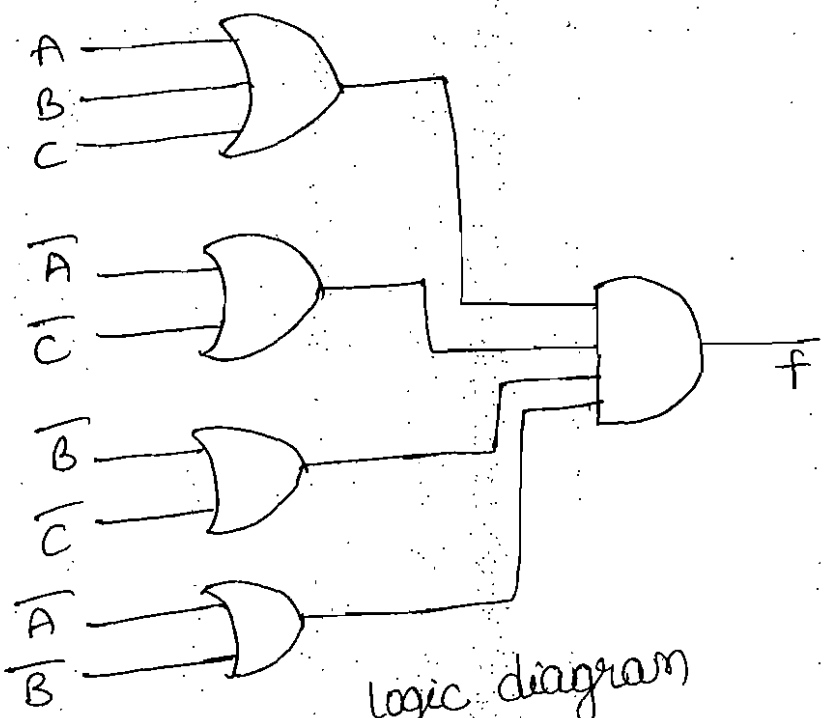
$= \bar{A} + \bar{A}\bar{B} + \bar{A}C + \bar{B} + \bar{B}C + \bar{A}\bar{C} + \bar{B}\bar{C} + 0$

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

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

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

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



Logic diagram

Four-variable K-map :-

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

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

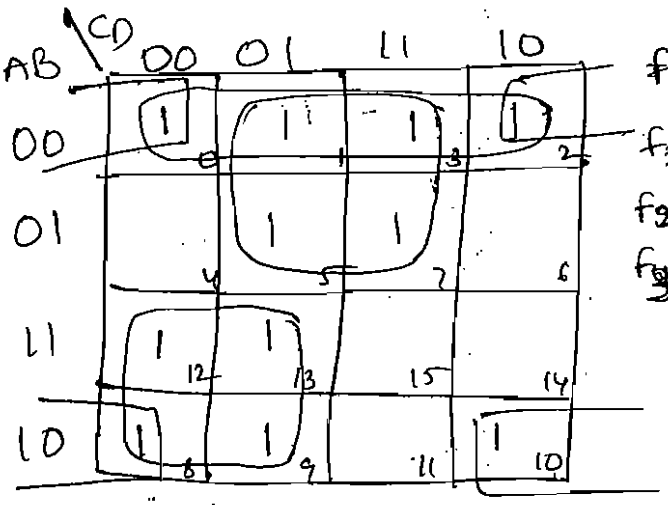
Sop form.

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

pos form.

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

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



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

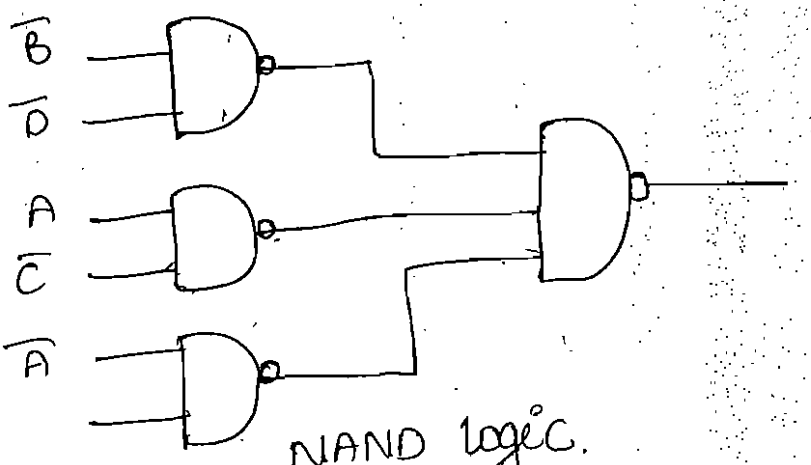
$f_2 = m_1 + m_3 + m_5 + m_7 = \overline{A}D$

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

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

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

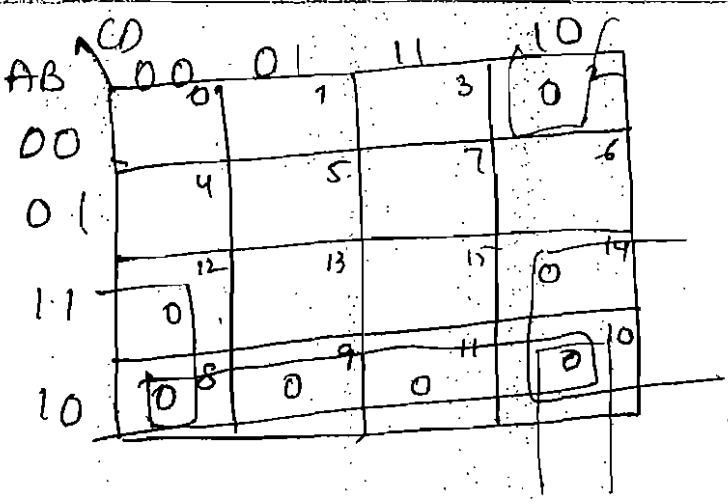
$= \overline{B}\,\overline{D} + A\overline{C} + \overline{A}D$



NAND logic.

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

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



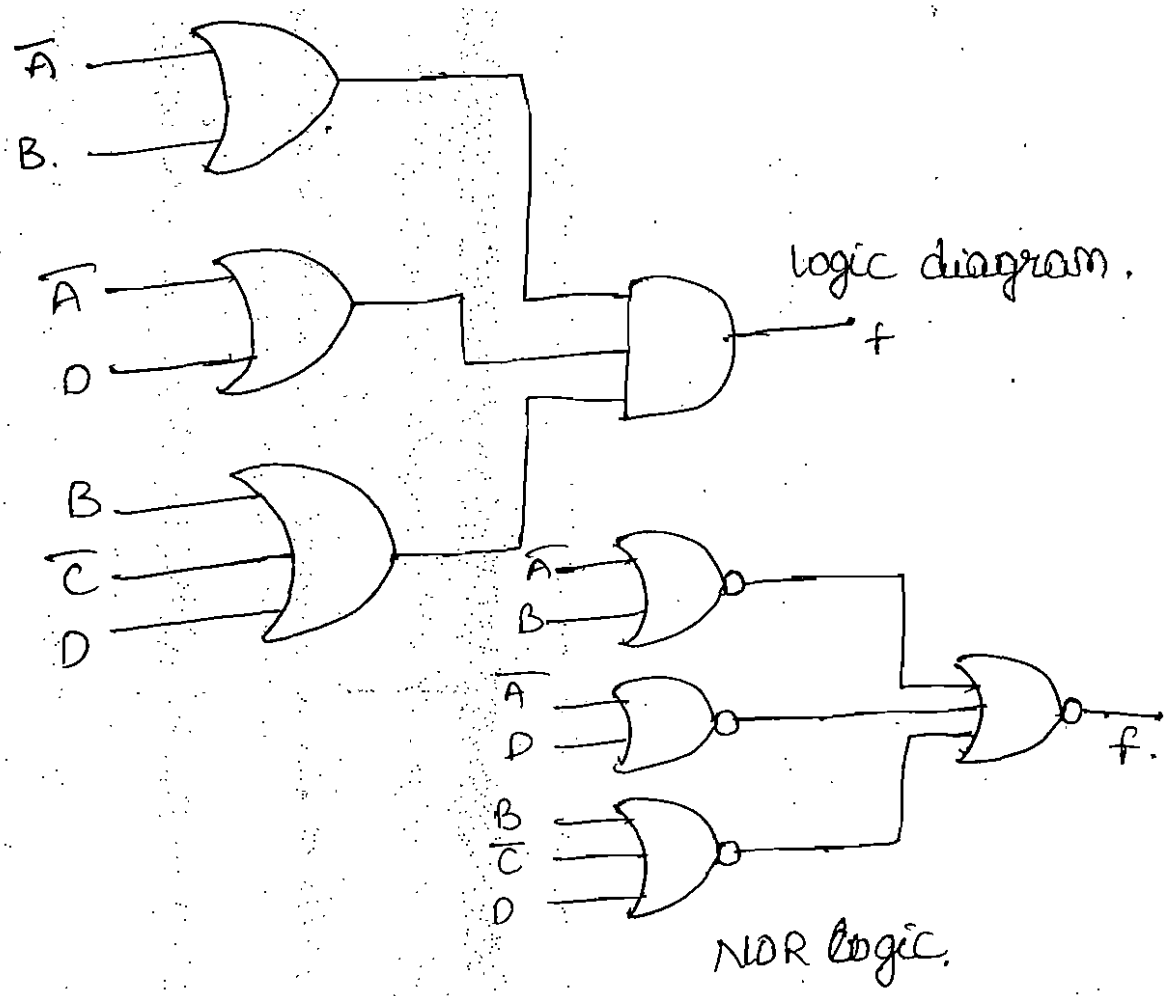
$f_1 = M_8 \cdot M_9 \cdot M_{11} \cdot M_{10}$ $f_2 = M_8 \cdot M_{12} \cdot M_{10} \cdot M_{14}$

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

$f_3 = M_2 \cdot M_{10}$

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

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



logic diagram.

NOR logic.

prime implicants :- [PI]

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

essential prime implicants:-

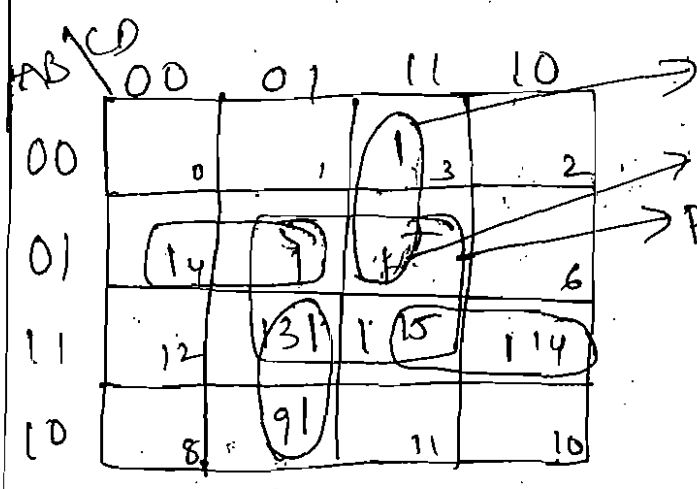
The prime implicants which contains at least one 1 which cannot be covered by any other prime implicant is called an essential prime implicant [EPI]

Redundant prime implicants:-

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

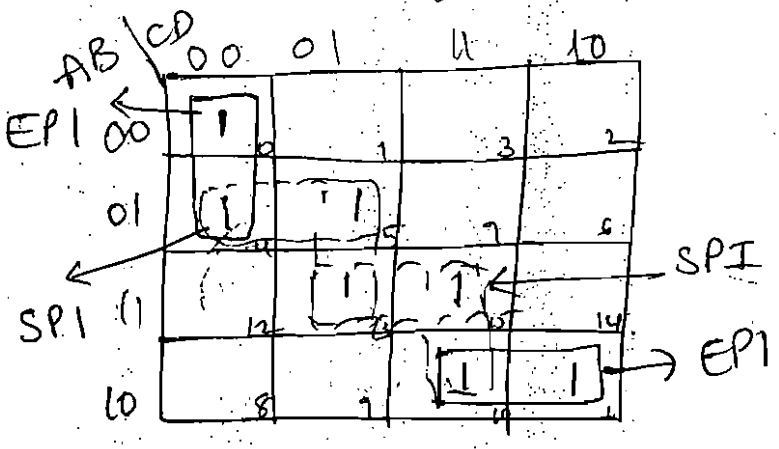
Selective prime implicants :-

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



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

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

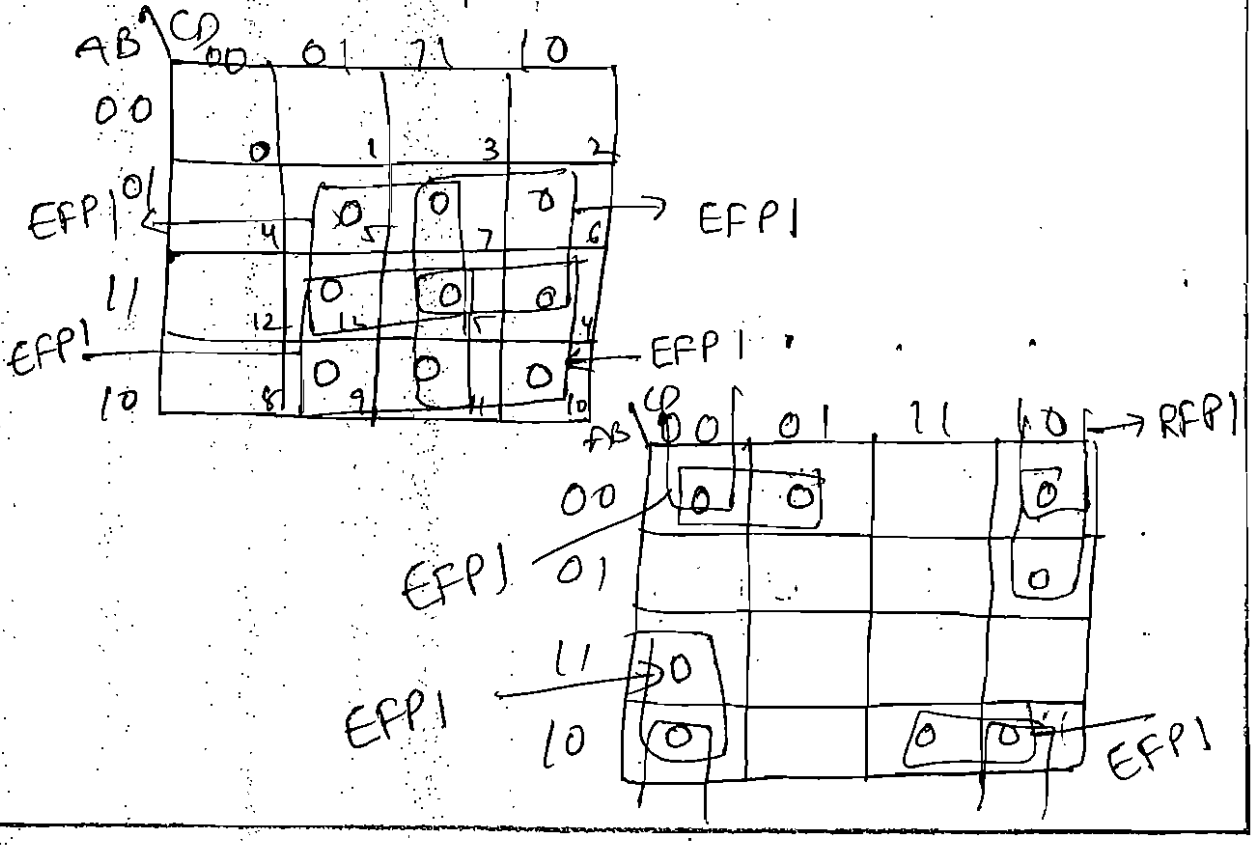


False prime implicants :- (FPI)

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

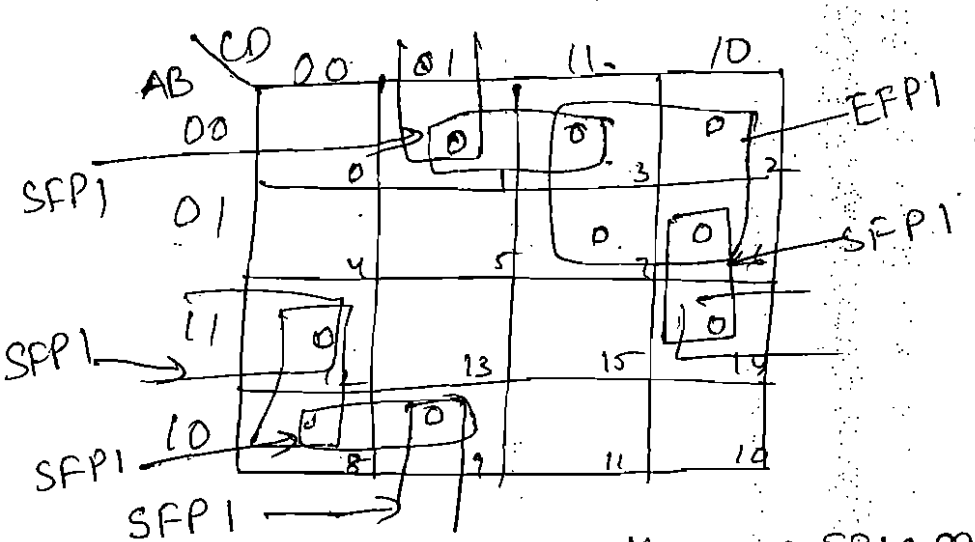
Essential false prime implicants :-

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



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

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



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

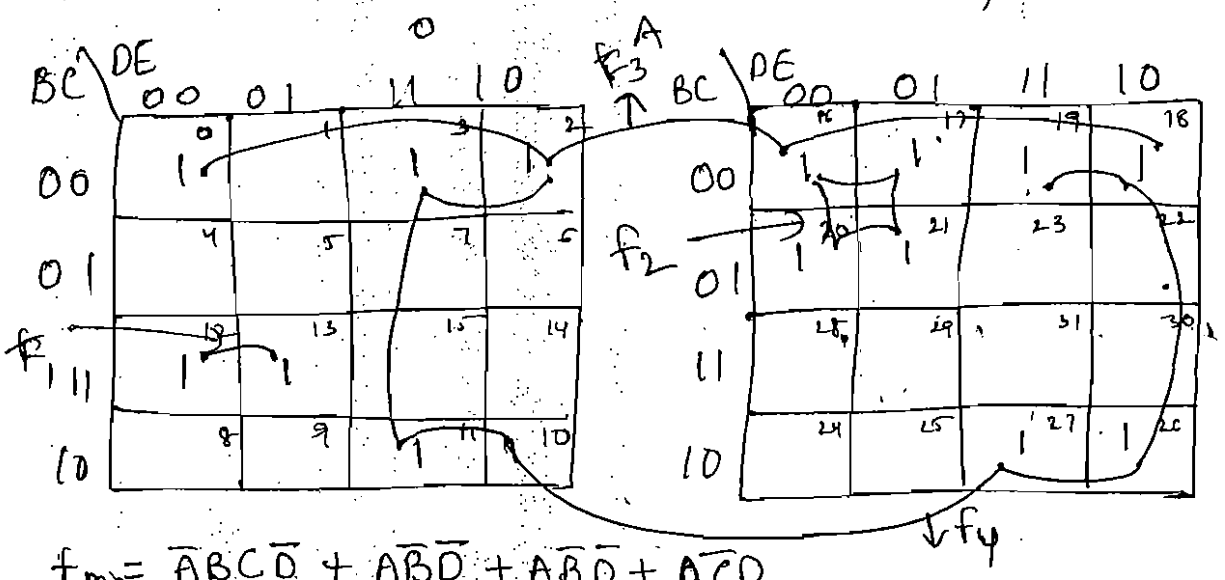
Five-variable K-map :-

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

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

* Reduce the following expression in sop and pos form.

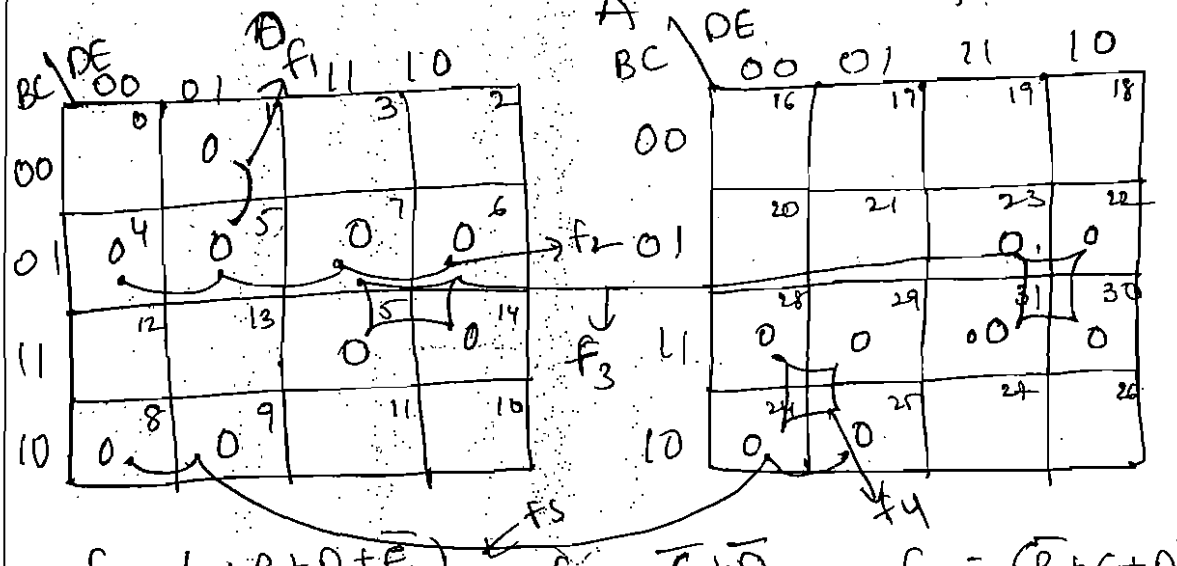
$f = \Sigma m(0,2,3,10,11,12,13,16,17,18,19,20,21,26,27)$



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

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

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



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

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

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

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

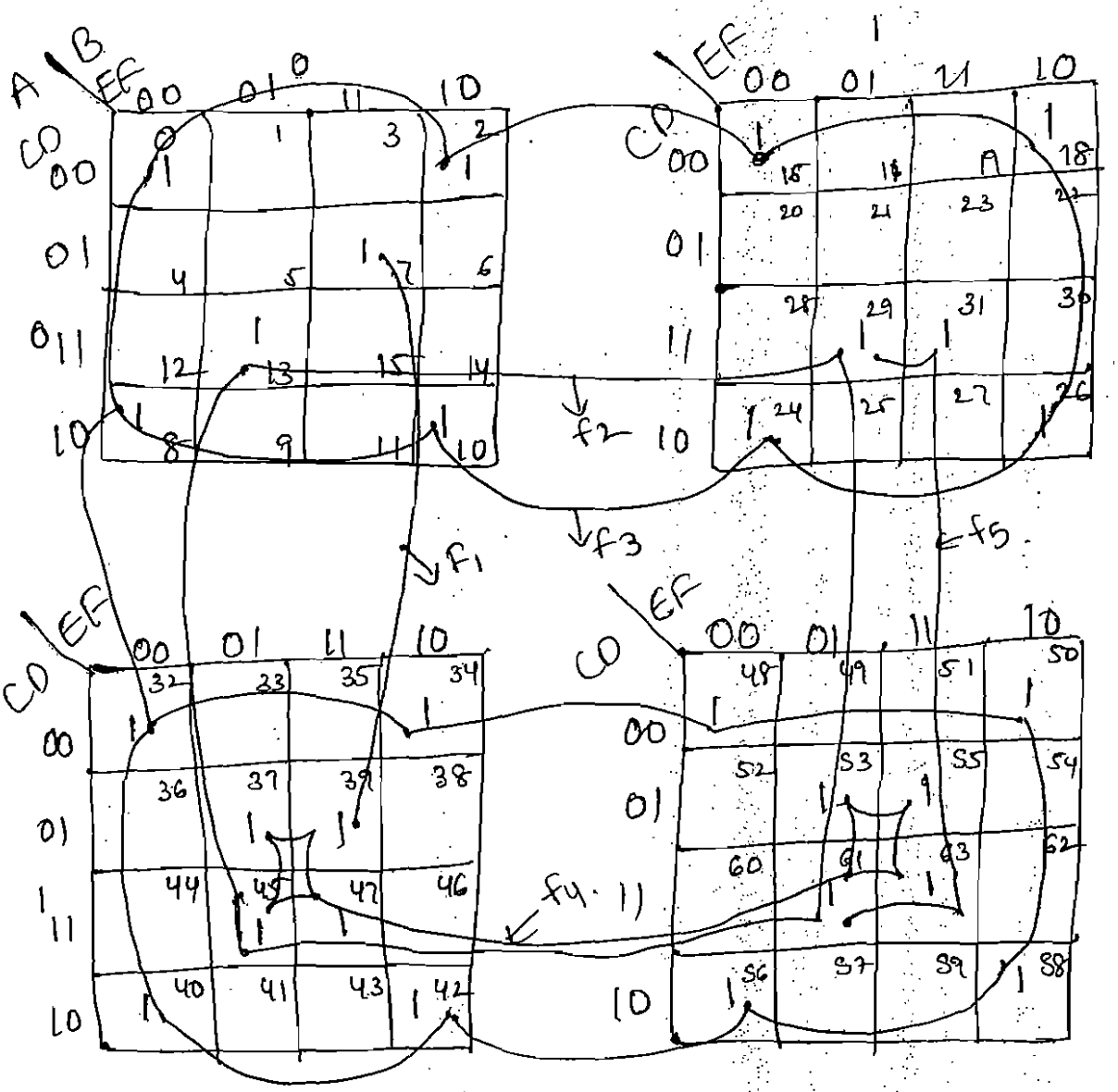
Six-Variable K-map :-

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

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

* Reduce the expression

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



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

$f_2 = CD\bar{E}F$

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

$f_4 = ADF$

$f_5 = BCDF$

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

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

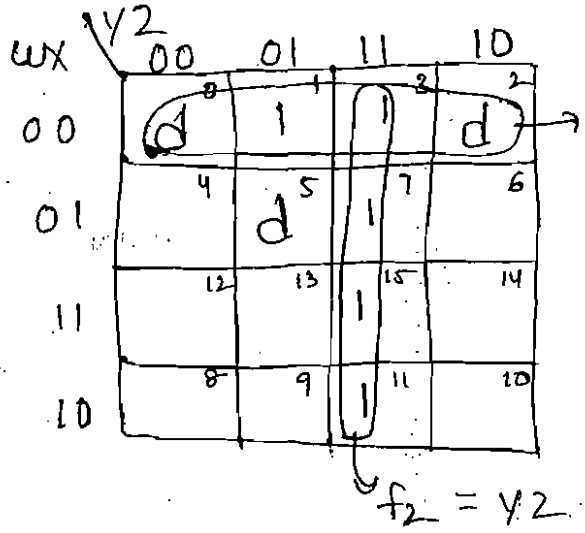
Don't care combinations :-

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

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

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

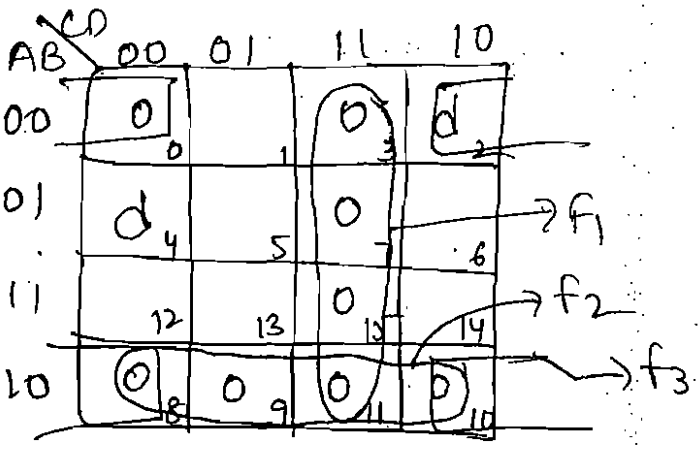
$F(w,x,y,z) = \Sigma m(1,3,7,11,15) + \Sigma d(0,2,5)$.



$F_{min} = f_1 + f_2$

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

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



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

$f_2 = \bar{A} + B$

$f_3 = (B+D)$.

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

Limitations of K-map :-

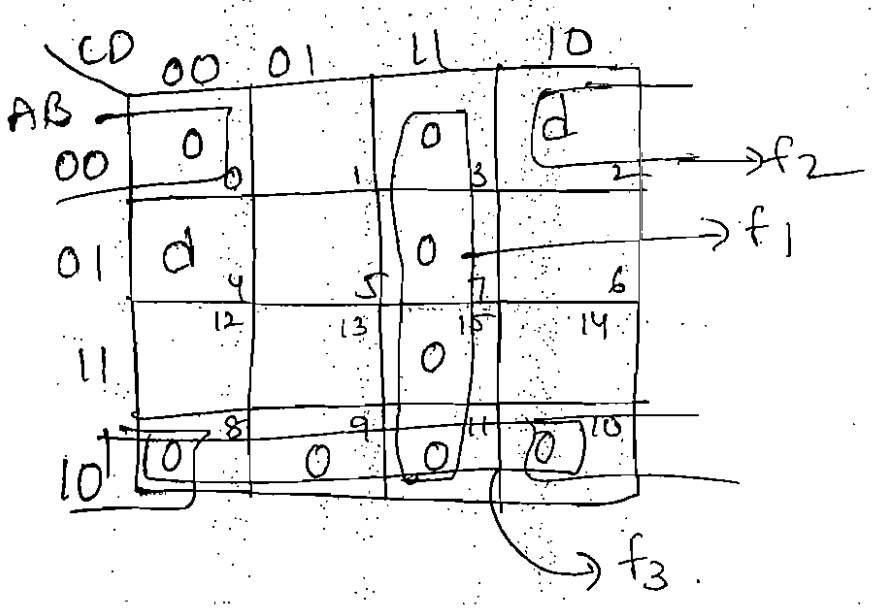
→ K-maps are not suitable when the number of variables involved exceed four. It may be used with difficulty up to five and six variable systems. But beyond 'six variable' K-maps cannot be physically visualized.

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

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

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

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

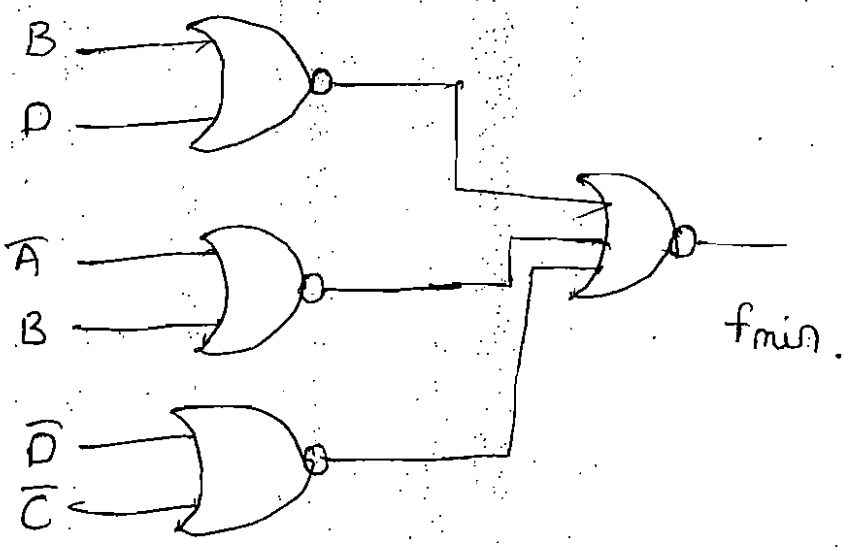


$f_1 = \bar{C} + \bar{D}$

$f_2 = (B + D)$

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

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



Quine Mccluskey or tabular method :-

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

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

Step 1. List all the minterms.

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

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

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

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

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

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

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

Prime implicant chart

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

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

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

Simplify the following function using the branching method

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

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

Prime implicant chart

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

In table x and p are essential prime implicants.

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

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

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

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

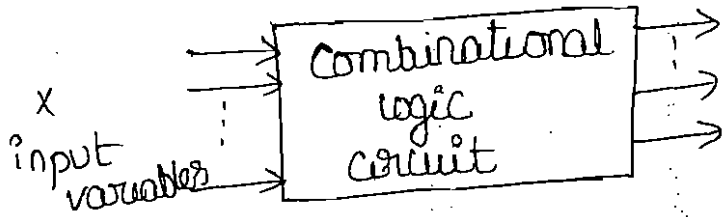
	12	16	24	28
* T(16,24)		X	X	
* S(12,28)	X			X
R(24,28)			X	X

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

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

12.6 Combinational logic circuits design.

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

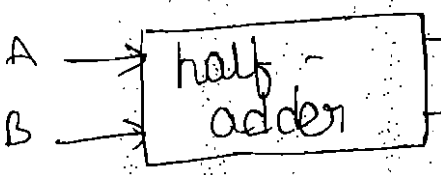


Adders :-

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

Half-adder

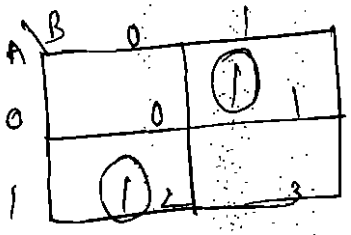
A half adder is a combinational circuit with two binary inputs. (augend and addend bits) and two outputs. Sum and carry.



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

(a) Truth table.

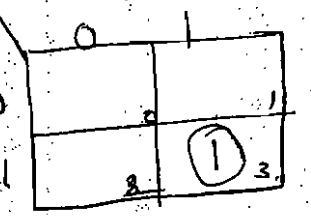
K-map for S.



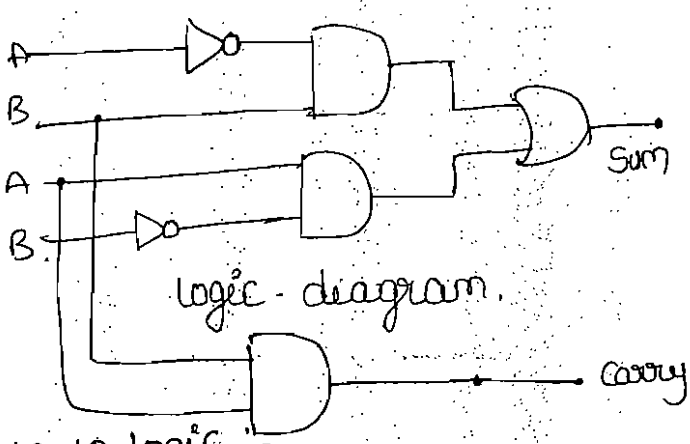
$S = A\bar{B} + \bar{A}B$

$= A \oplus B$

K-map for C

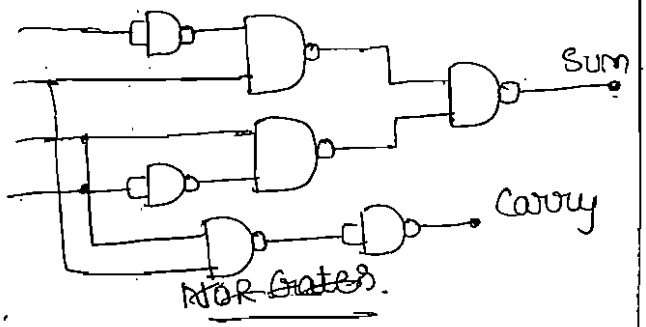


$C = A \cdot B$



Logic - diagram.

NAND Gates.



NOR Gates.

NAND logic :-

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

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

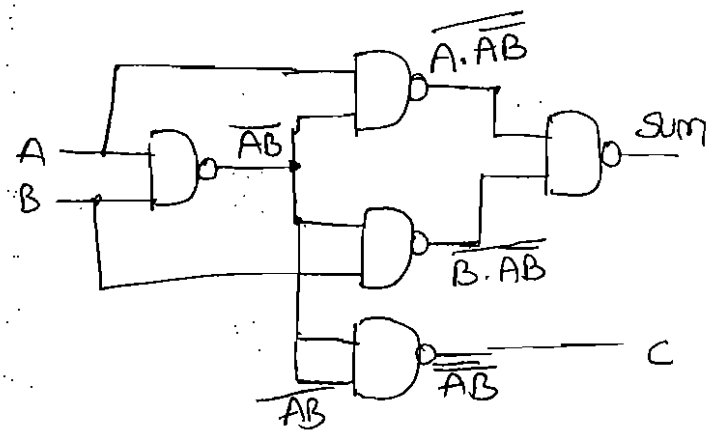
$= A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$

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

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

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

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



NOR logic

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

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

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

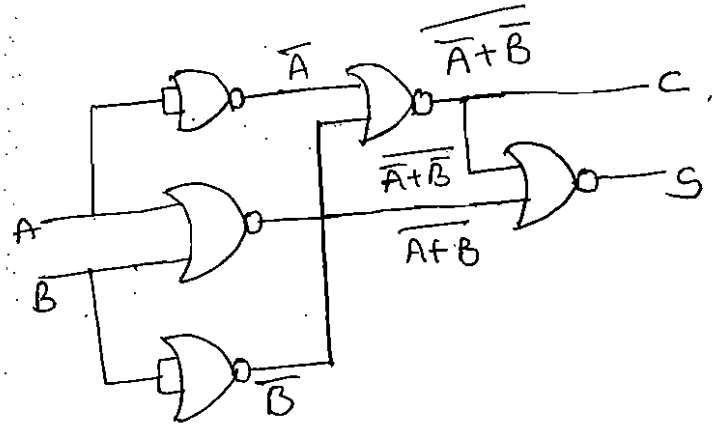
$= (A+B)(\bar{A}+\bar{B})$

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

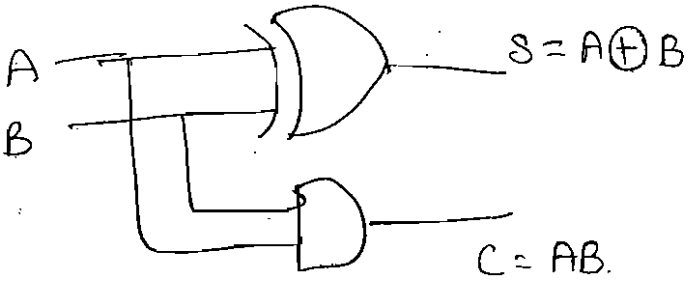
$= \overline{(\bar{A}+B)} + \overline{(\bar{A}+\bar{B})}$

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

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

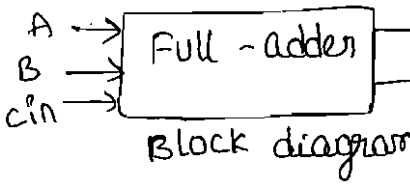


Simple logic diagram is.



Full adder :-

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

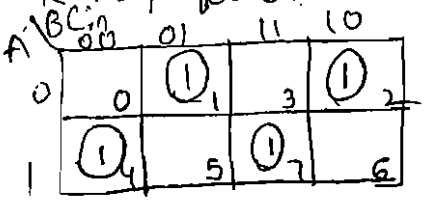


Block diagram.

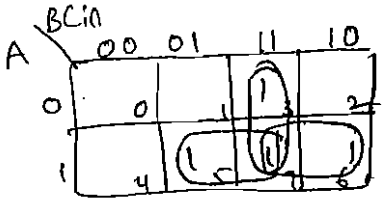
inputs			outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table

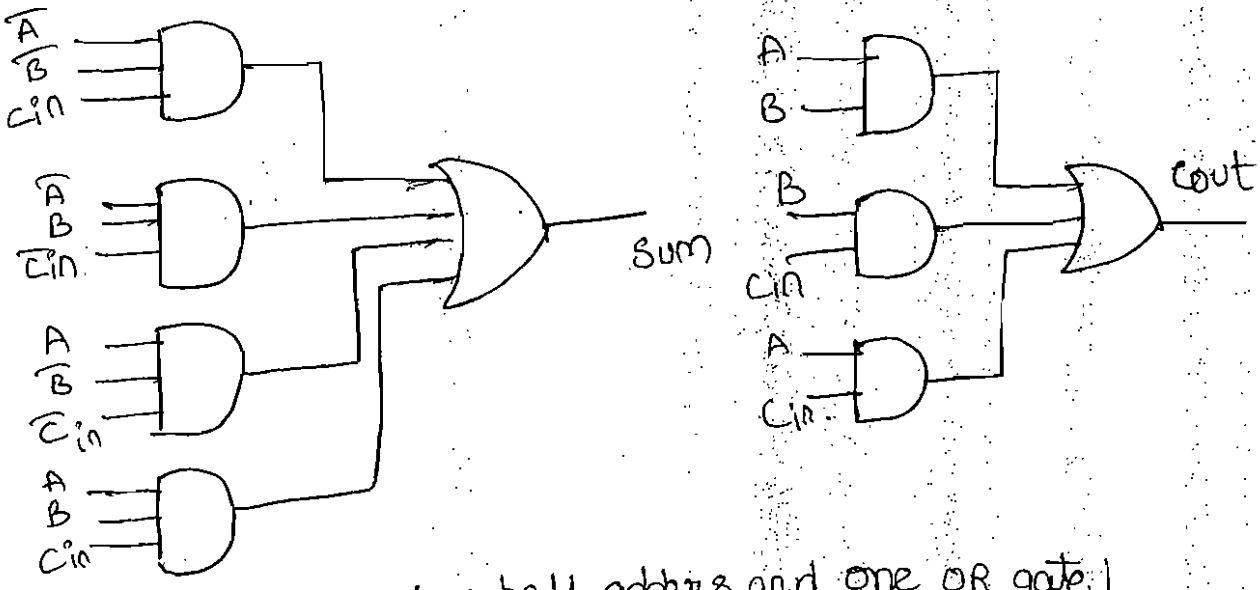
K-map for S.



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



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



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

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

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

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

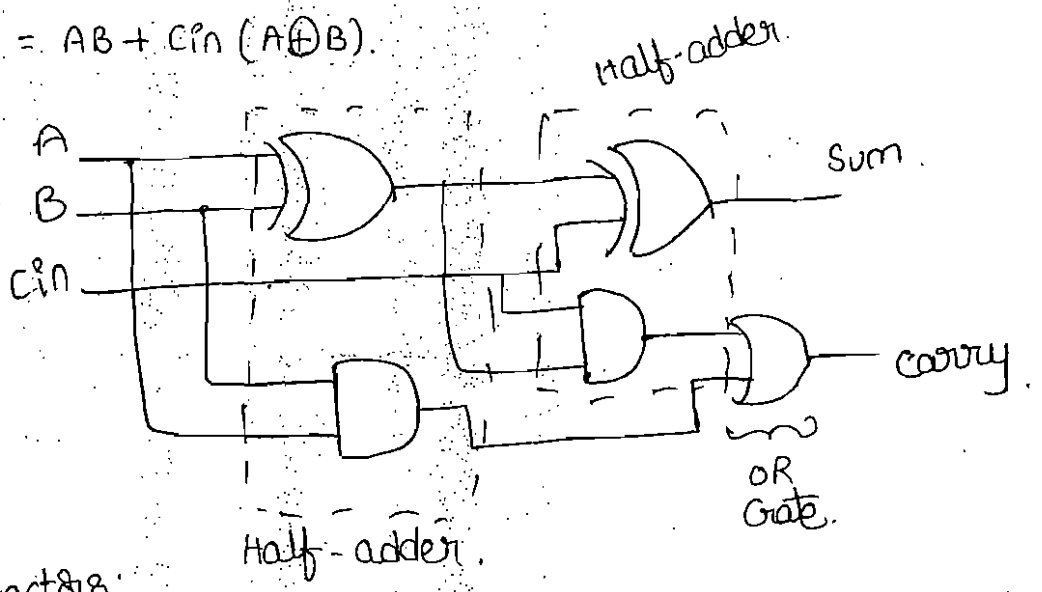
$= A \oplus B \oplus C_{in}$.

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

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

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

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

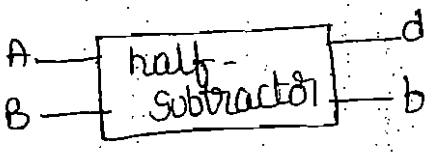


Subtractors:-

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

Half-subtractor:-

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



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

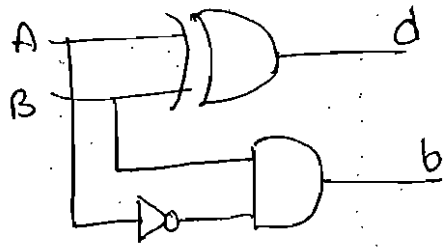
K-map for d

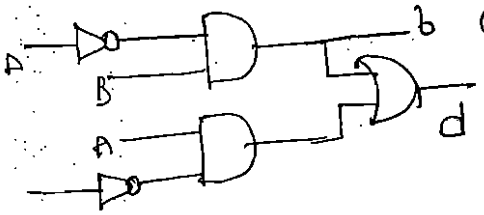
A \ B	0	1
0	0	(1) 1
1	(1) 2	3

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

$= A \oplus B$

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





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

$b = \bar{A}B$

Logic diagrams of a half-subtractor.

NAND logic:-

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

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

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

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

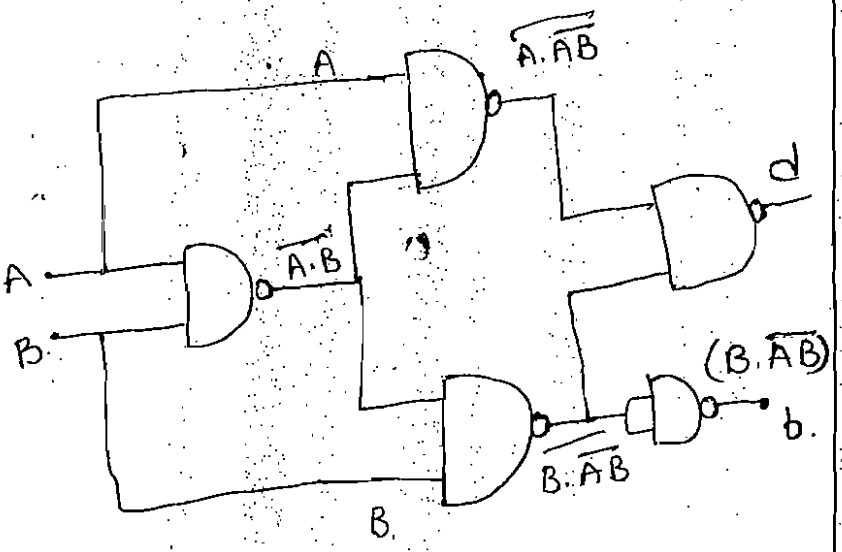
$= A.\overline{\bar{A}B}.\ B.\overline{\bar{A}B}$

$b = \bar{A}B$

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

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

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



NOR logic

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

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

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

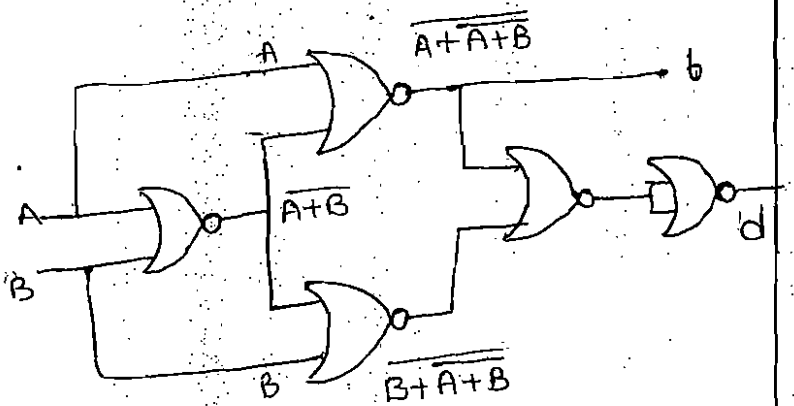
$= \overline{B+\bar{A}+B} + \overline{A+\bar{A}+B}$

$b = \bar{A}B$

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

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

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



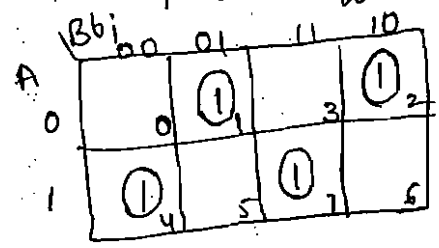
Full - Subtractor:-

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

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

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

K-map for difference



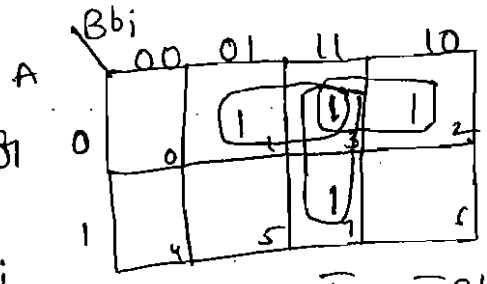
$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + A\bar{B}\bar{b_i} + AB b_i$$

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

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

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

K-map for borrow.

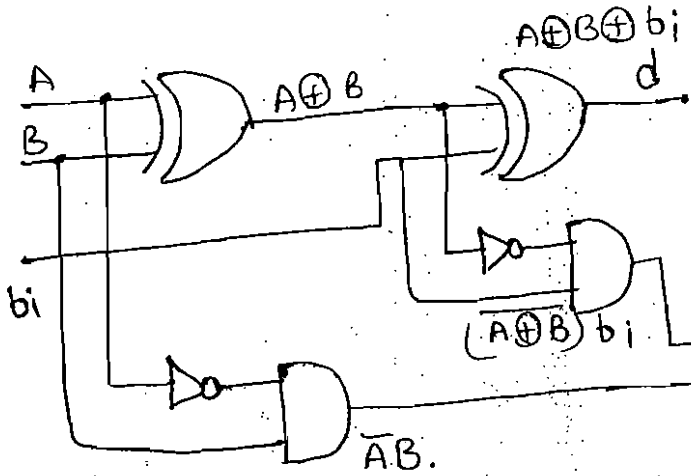


$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + \bar{A}Bb_i + AB b_i$$

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

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

Full-Subtractor by using two half-Subtractor



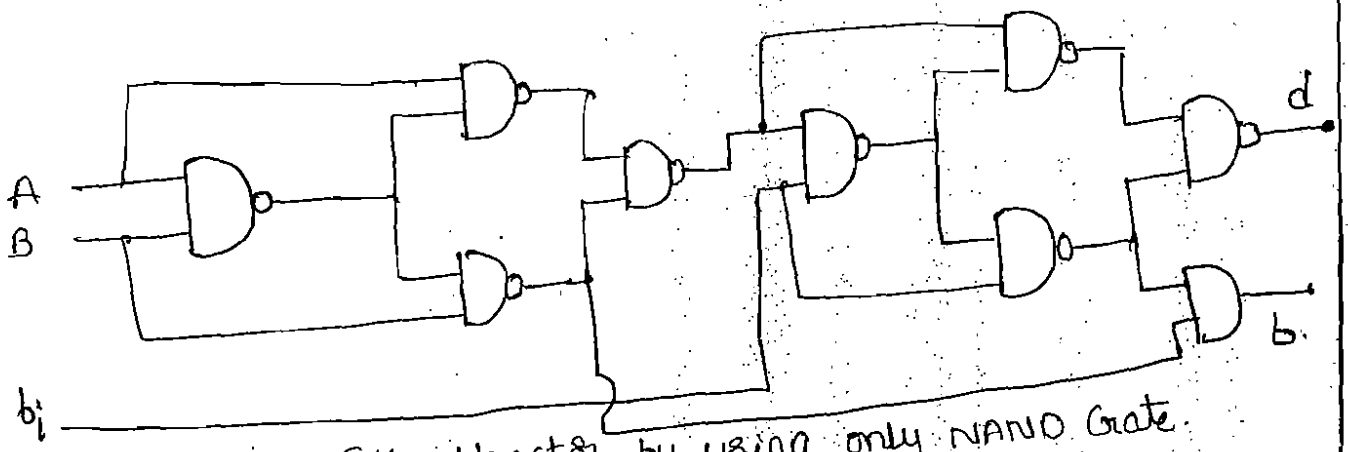
NAND logic:-

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

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

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

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



Full subtractor by using only NAND Gate.

NOR Logic.

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

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

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

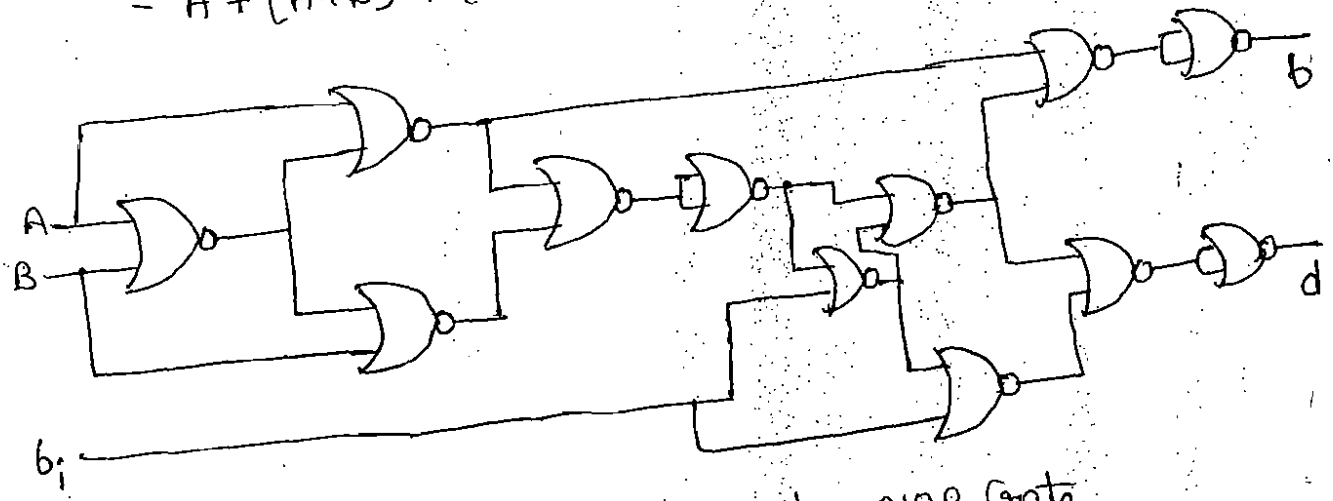
$$= \overline{(A \oplus B) + (\overline{A \oplus B}) + b_i} + \overline{b_i + (A \oplus B) + b_i}$$

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

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

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

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

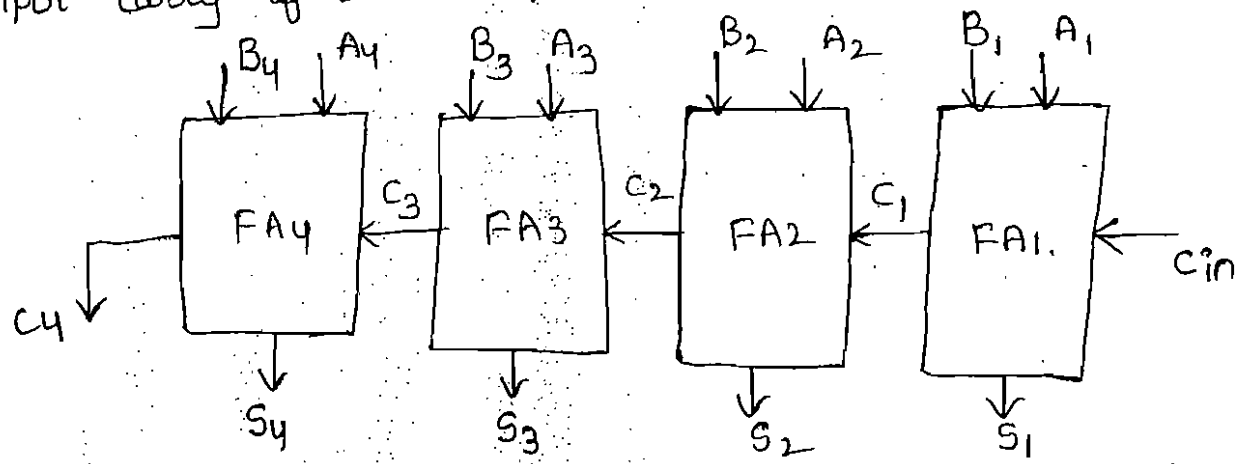


Full subtractor by using only NOR Gate

Applications of full adders :-

Binary parallel adder :-

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



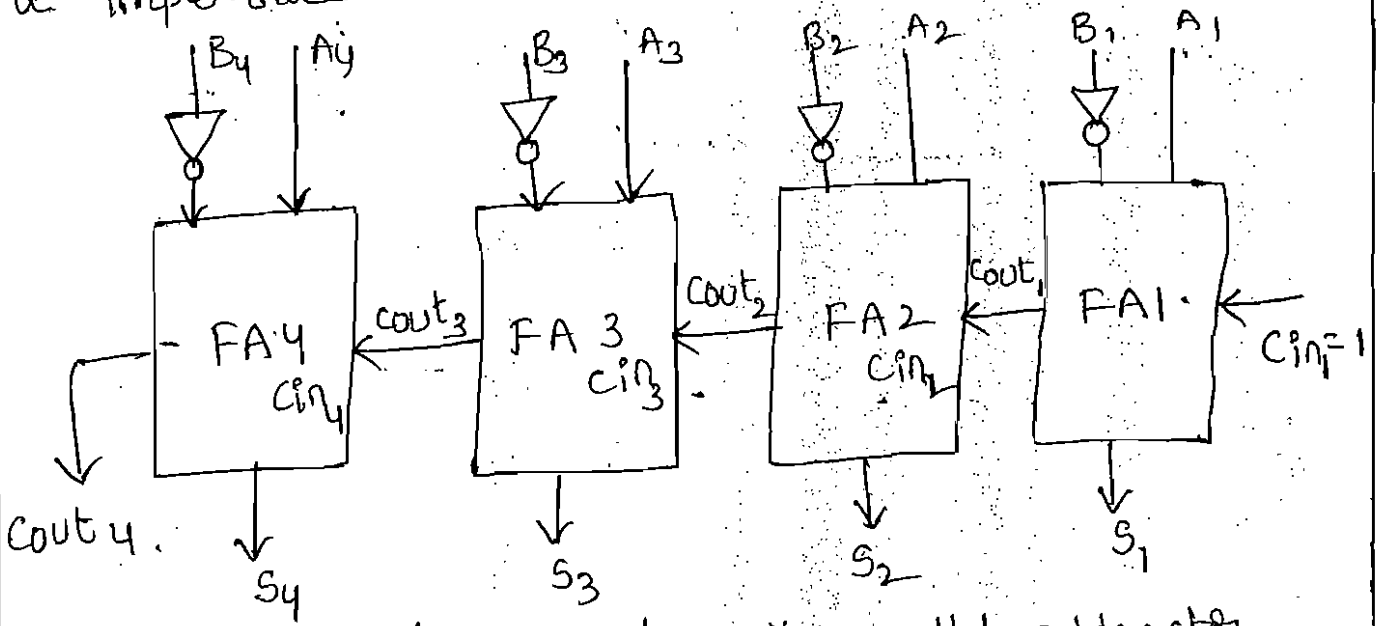
Logic diagram of 4-bit binary parallel adder.

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

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

4-bit parallel subtractor:-

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



Logic diagram of 4-bit parallel subtractor.

Binary adder - Subtractor :-

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

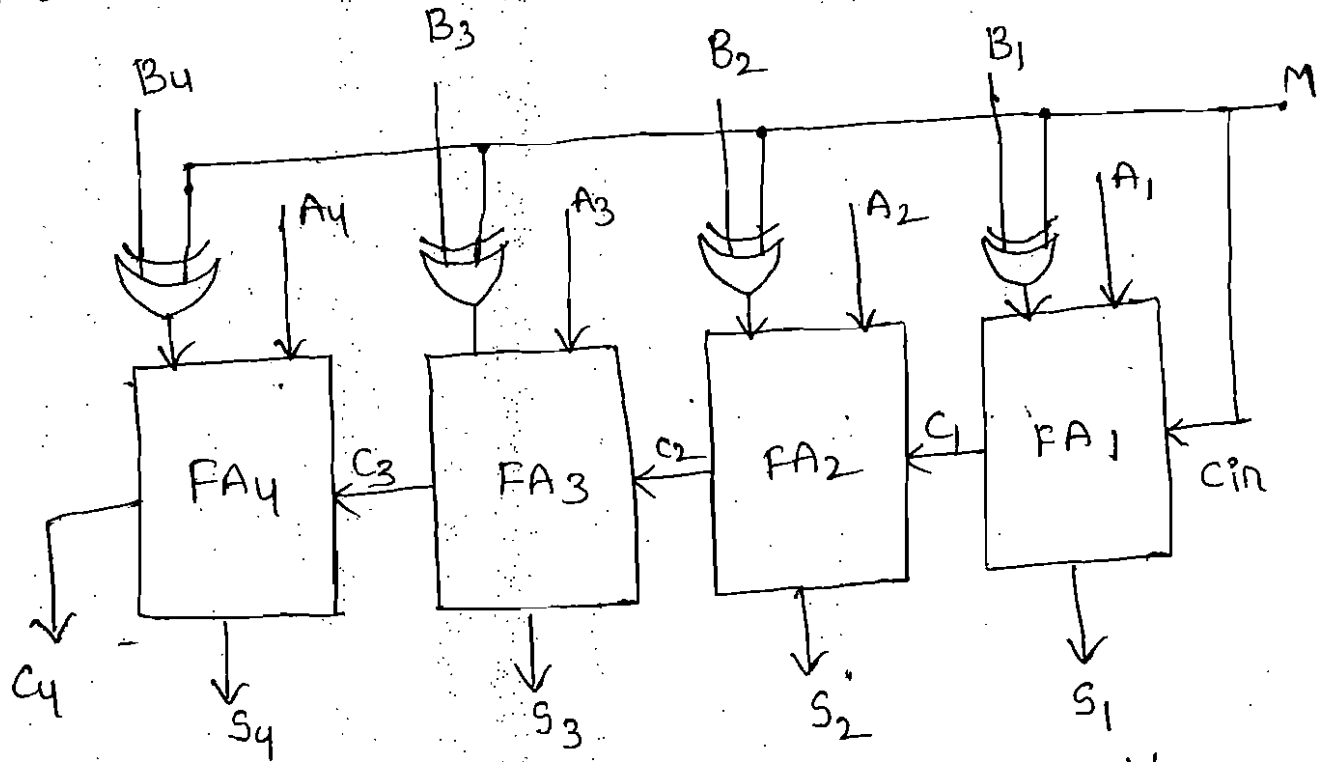
→ when M = 0, the circuit is an adder.

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

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

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

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

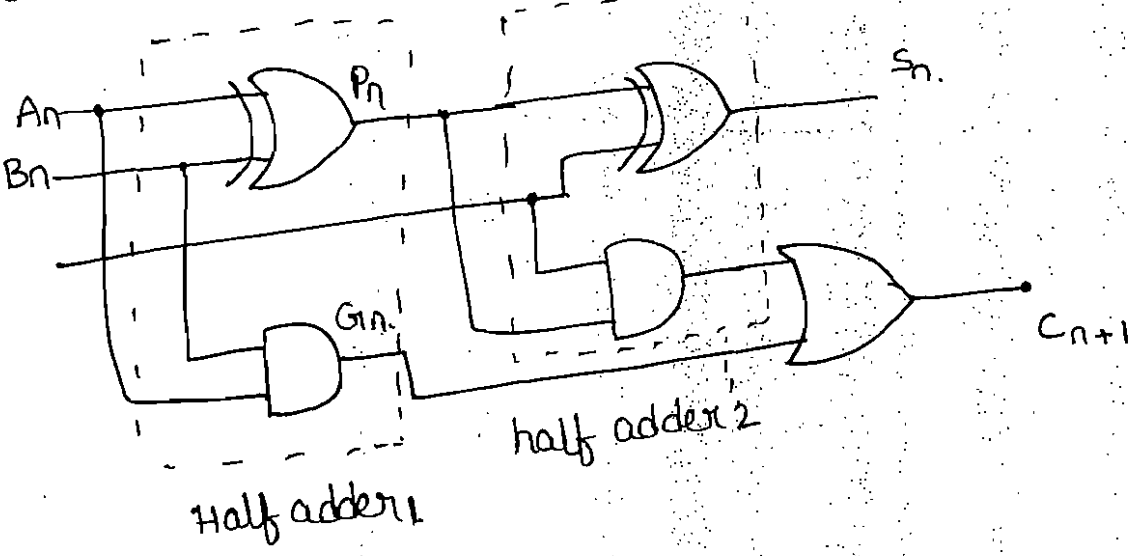


Logic diagram of a 4-bit binary - adder - subtractor.

LOOK-A-HEAD-CARRY ADDER :-

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

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



carry generate :-

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

$$G_i = A \cdot B.$$

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

$$G_n = A_n \cdot B_n$$

Carry propagation :-

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

If P is taken as a

$$P = A \oplus B.$$

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

$$P_n = A_n \oplus B_n.$$

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

$$S_n = P_n \oplus C_n$$

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

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

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

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

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

$n = 2$

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

$n = 3$

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

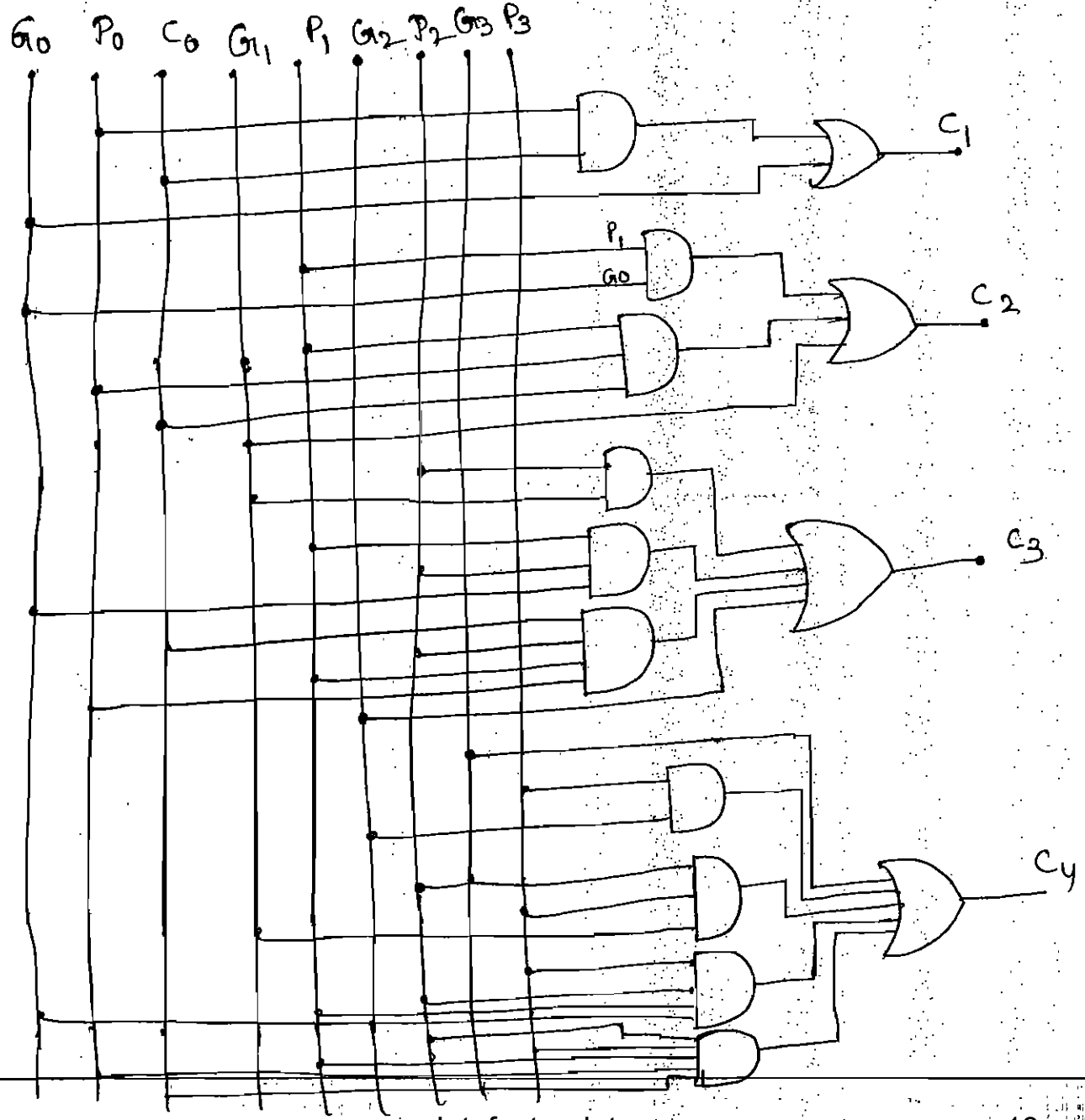
$n = 4$

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

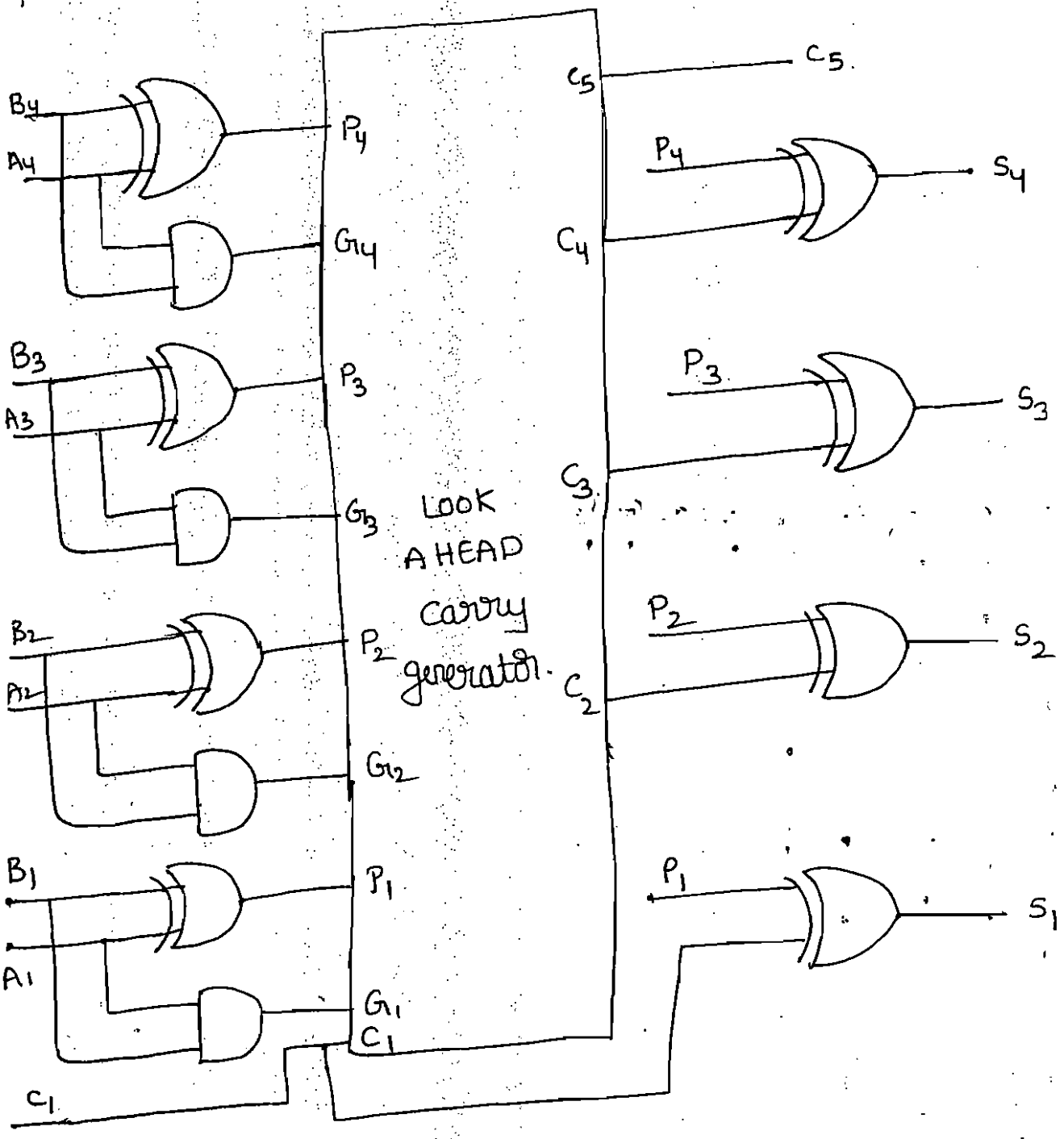
The general expression for n-stages.

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

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

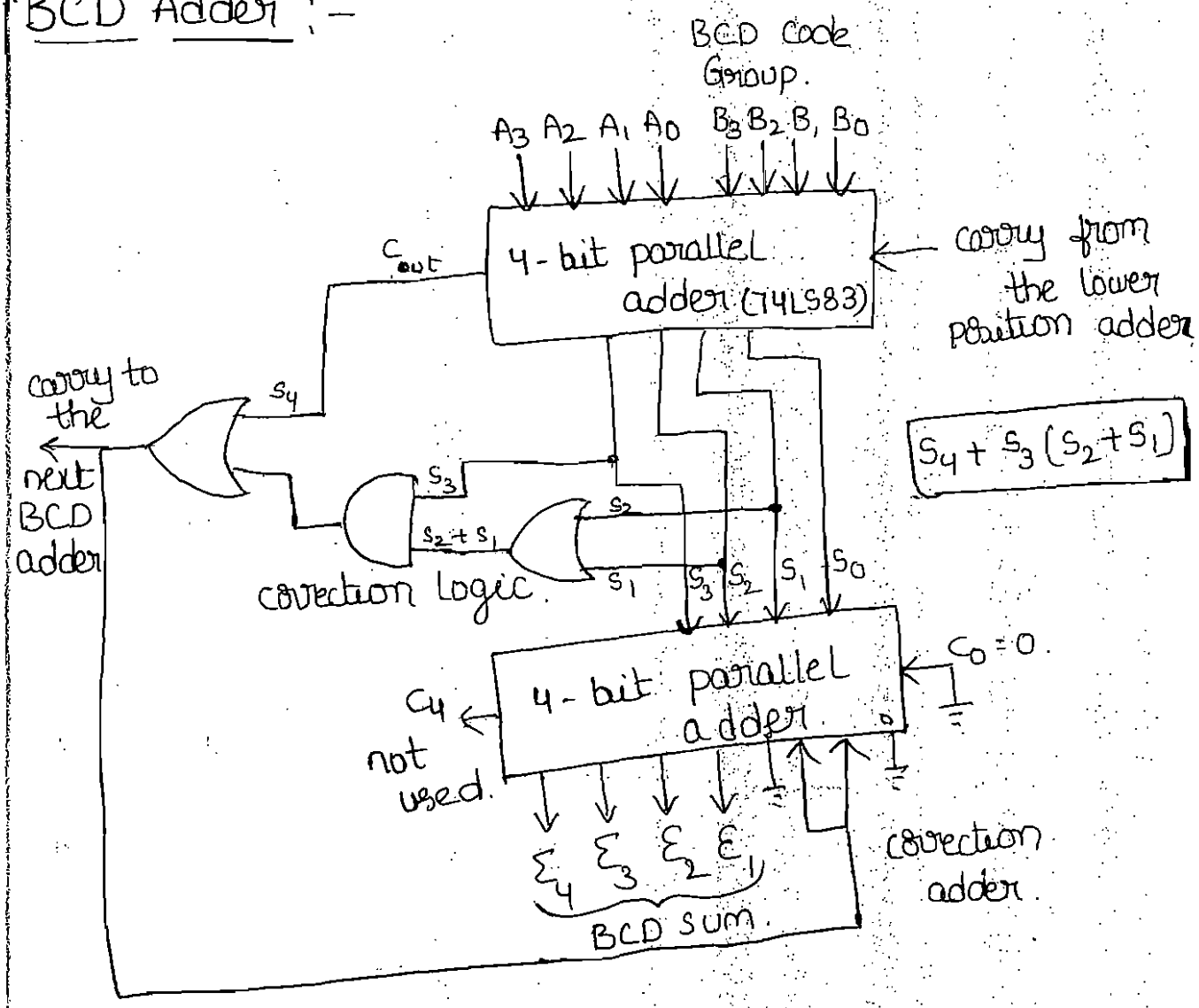


The block diagram of a 4-stage look-ahead-carry parallel adder is shown in the below figure.



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

BCD Adder :-



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

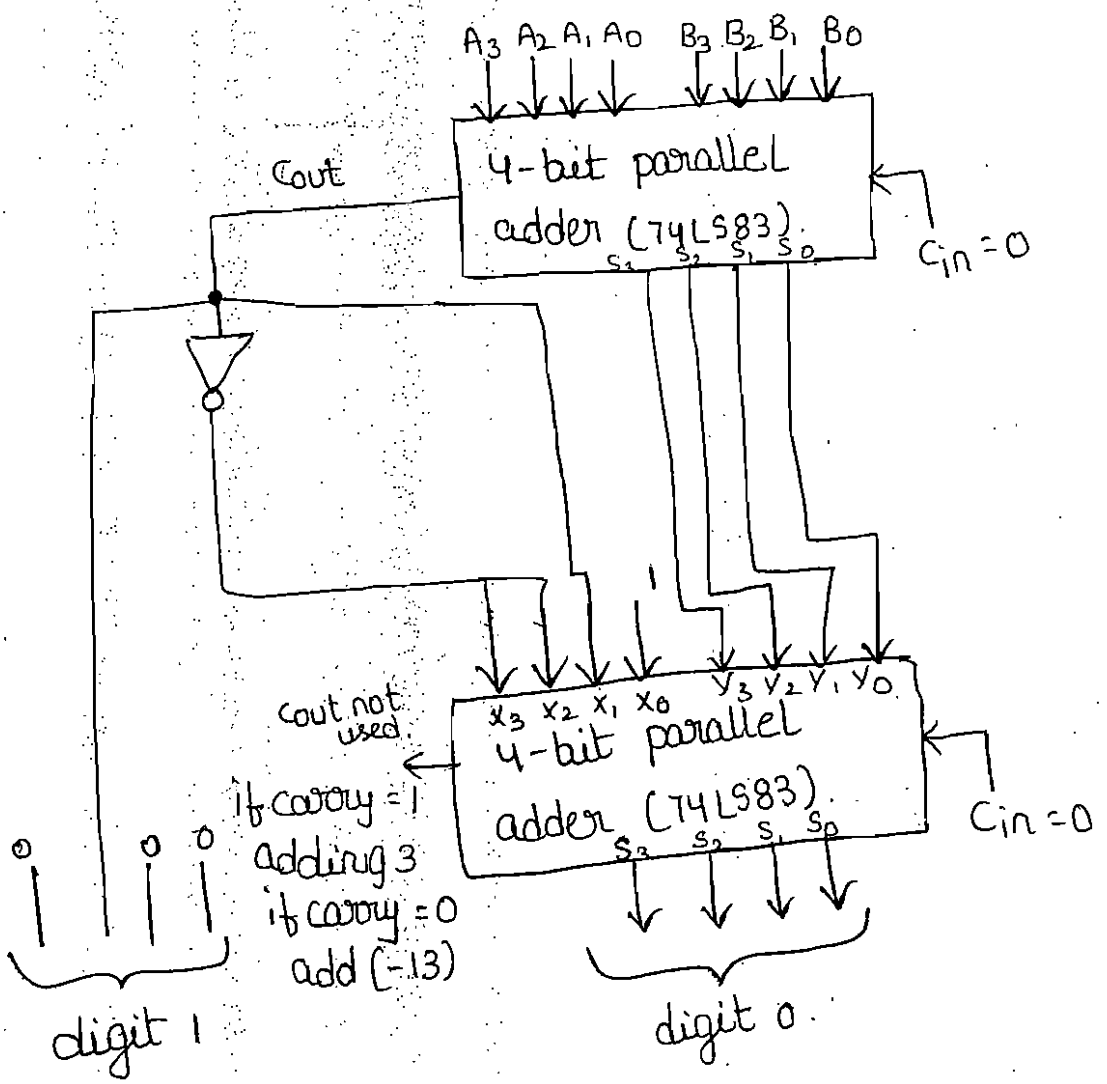
In such cases $\varepsilon_3 \varepsilon_2 \varepsilon_1 \varepsilon_0 = S_3 S_2 S_1 S_0$. Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder.

EXCess-3 Adder :-

→ In EXCess-3 addition.

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

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



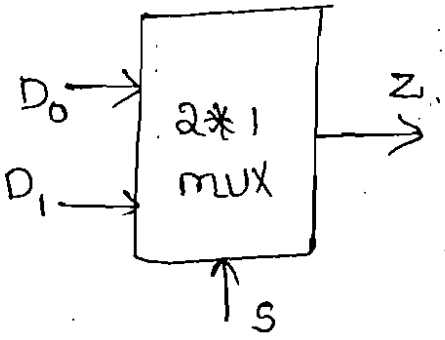
The final Answer in XS-3 form.

Multiplexers (data selectors).

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

Basic 2-input multiplexer :-

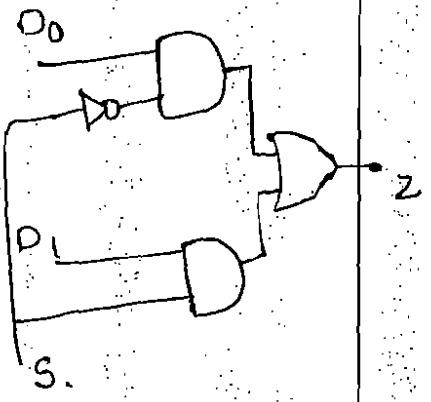
In 2-input multiplexer have 2-inputs they are D_0 and D_1. and one select line S. and output is Z.



Block diagram.

S	Z
0	D_0
1	D_1

Truth table.



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

Logic diagram.

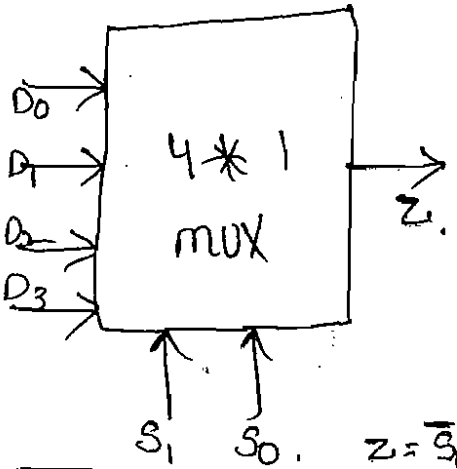
The logic levels applied to the S inputs determines which AND gate is enabled. So that its data input passes through the OR gate to the output.

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

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

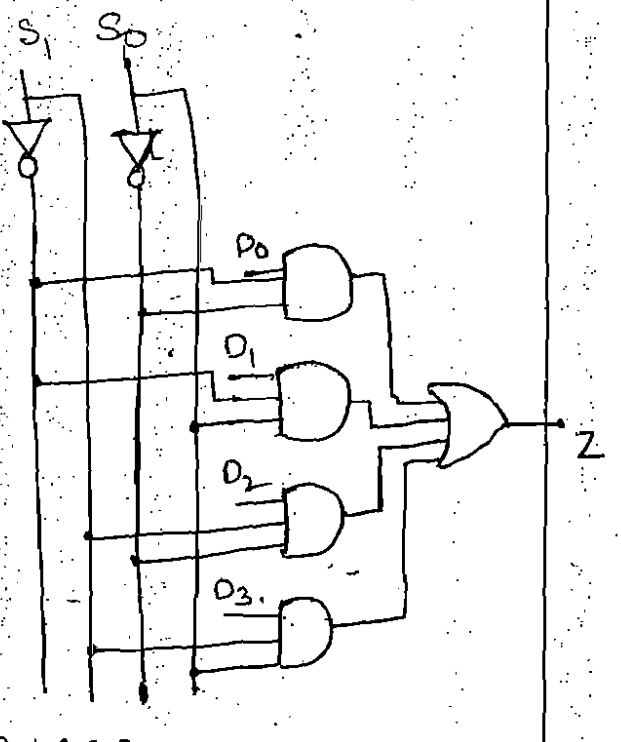
Basic 4-input multiplexer :-

Block diagram



Truth table

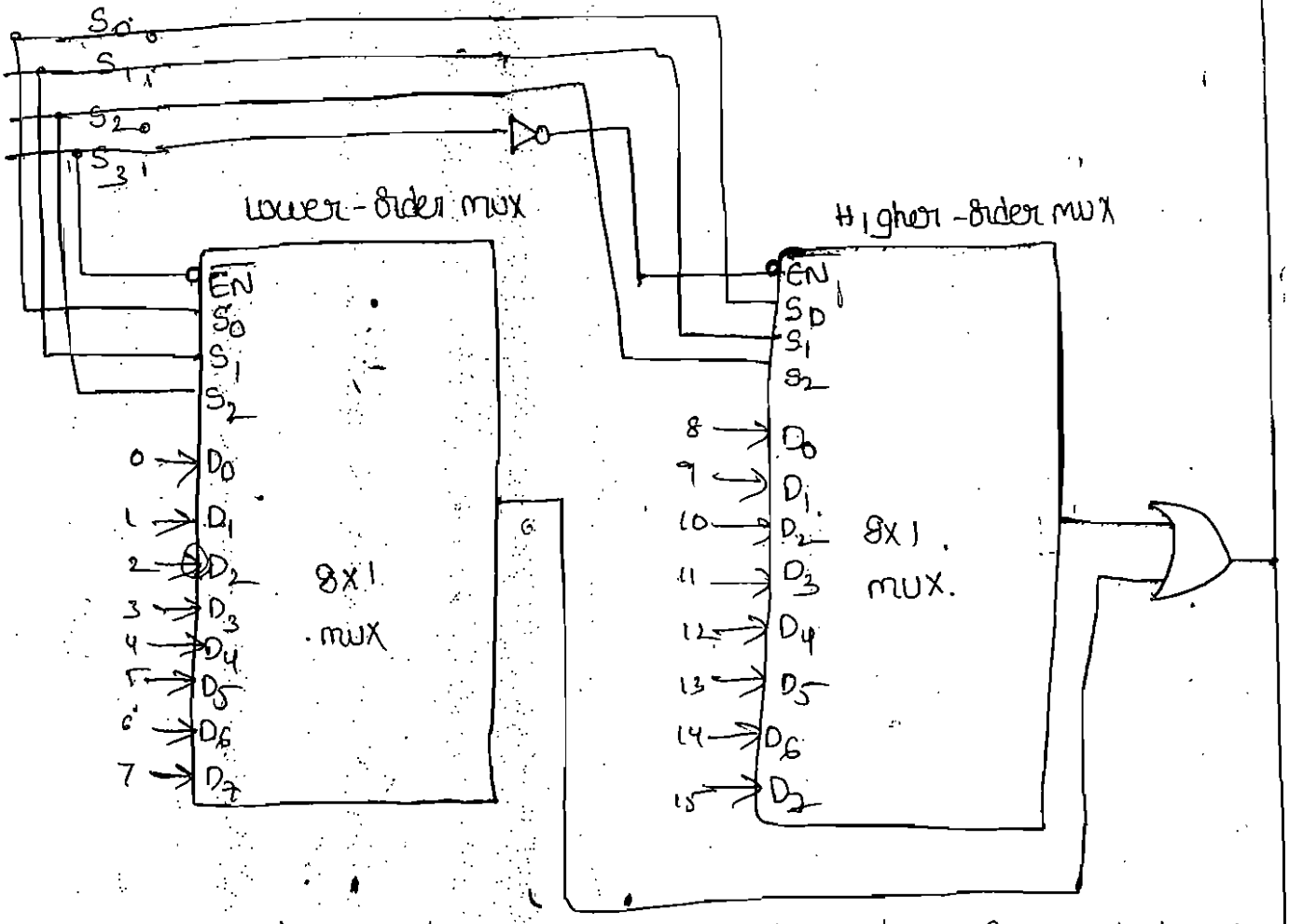
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



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

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

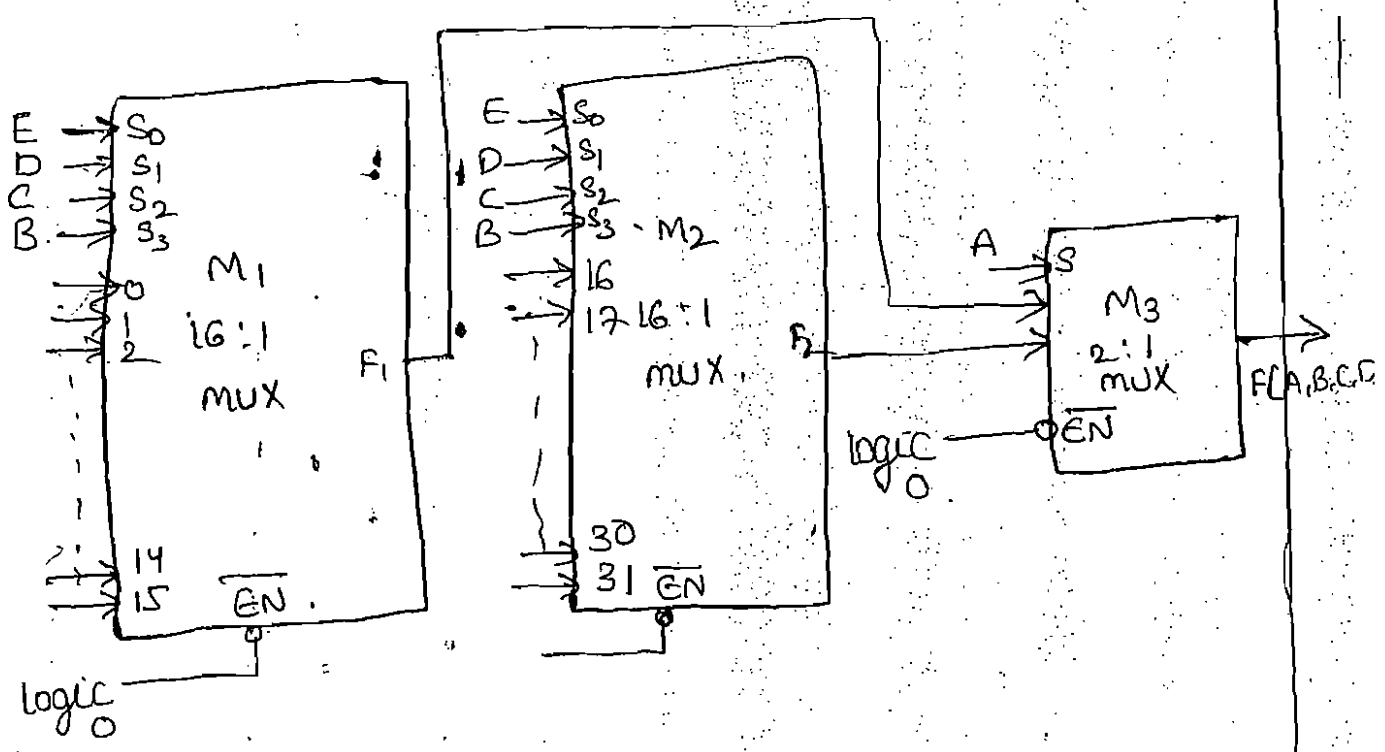
To use two 8-inputs multiplexers to get a 16-inputs multiplexers one OR gate and one inverter are also required. The four select inputs S_3, S_2, S_1 and S_0 and will select one of the 16 inputs to pass through to X. The S_3 input determines which multiplexer is enabled. When $S_3 = 0$, the left multiplexer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



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

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

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

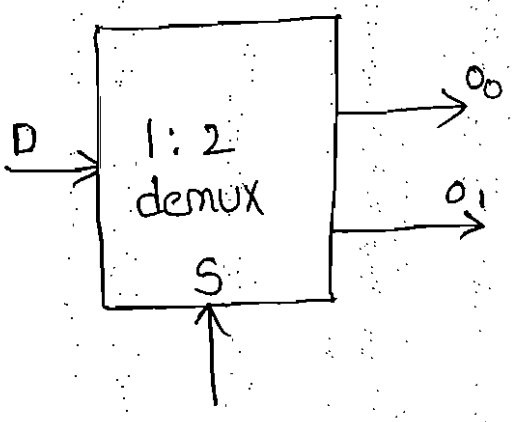


DEMULTIPLEXERS :-

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

1-line to 2-line demultiplexer :-

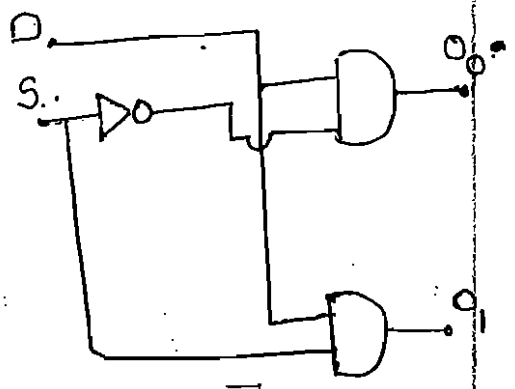
The input data line goes to all of the AND gates. The select line S enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output lines.



Block diagram

S	O_0	O_1
0	0	D
1	D	0

Truth table



$O_0 = D\bar{S}$

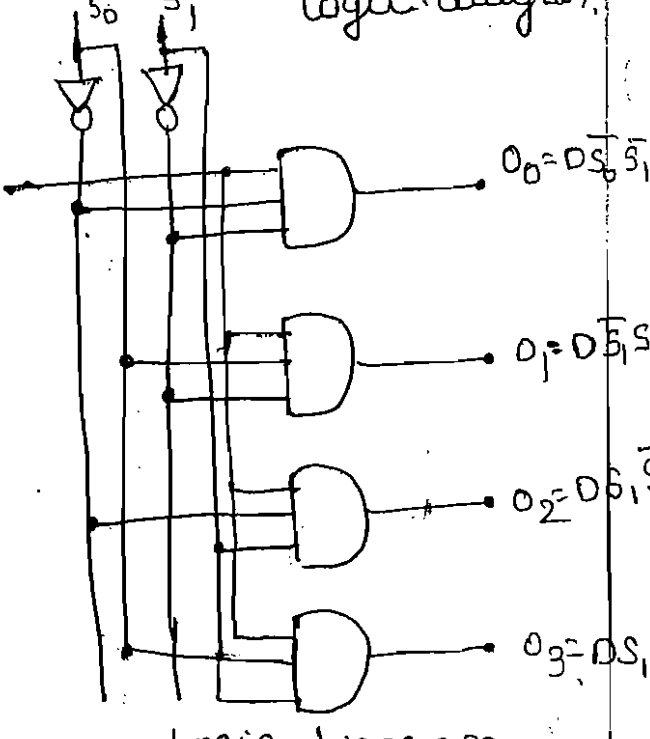
$D_1 = DS$

Logic diagram.

1-line to 4-line demultiplexer :-

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

Truth table.

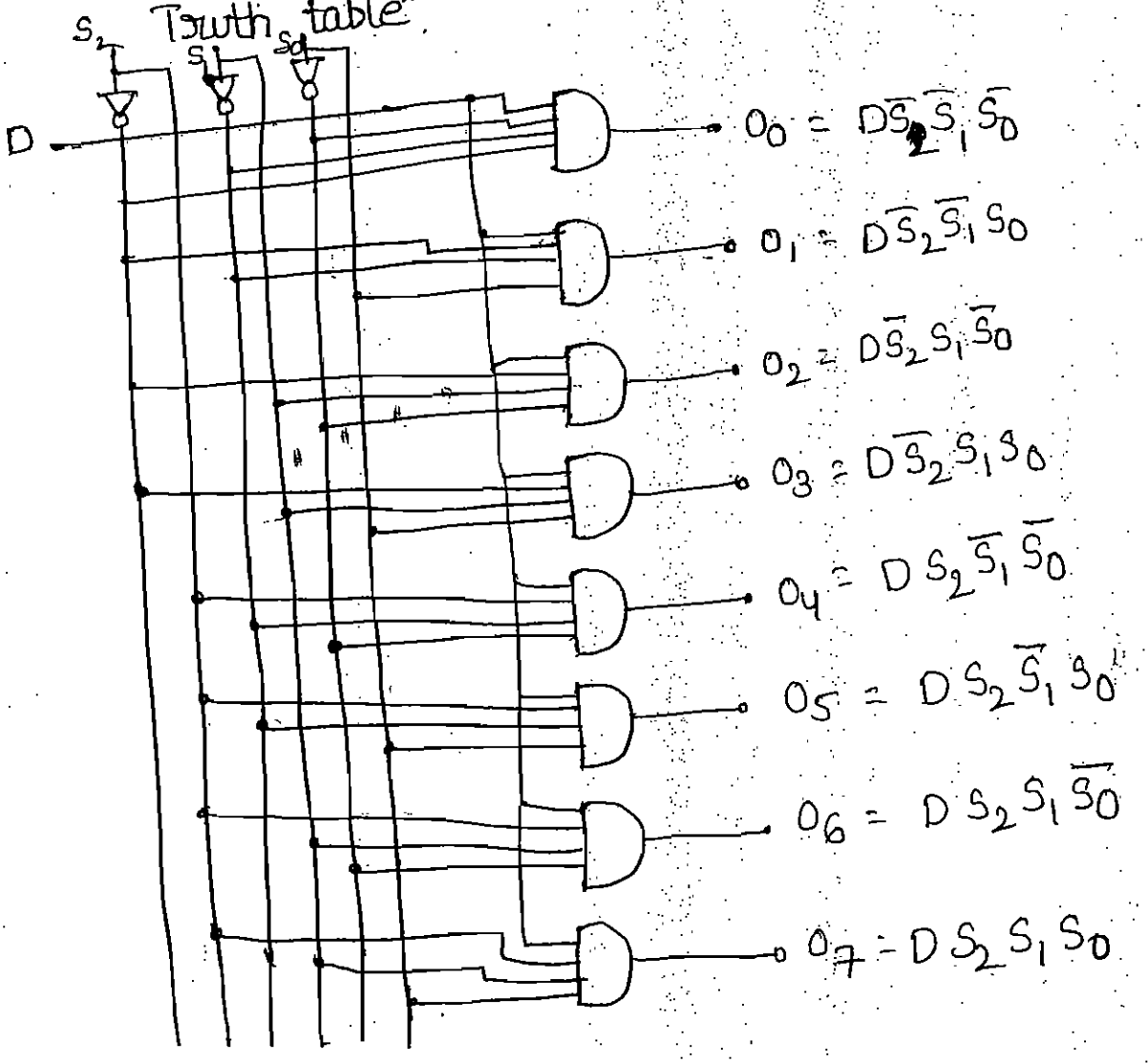


Logic diagram.

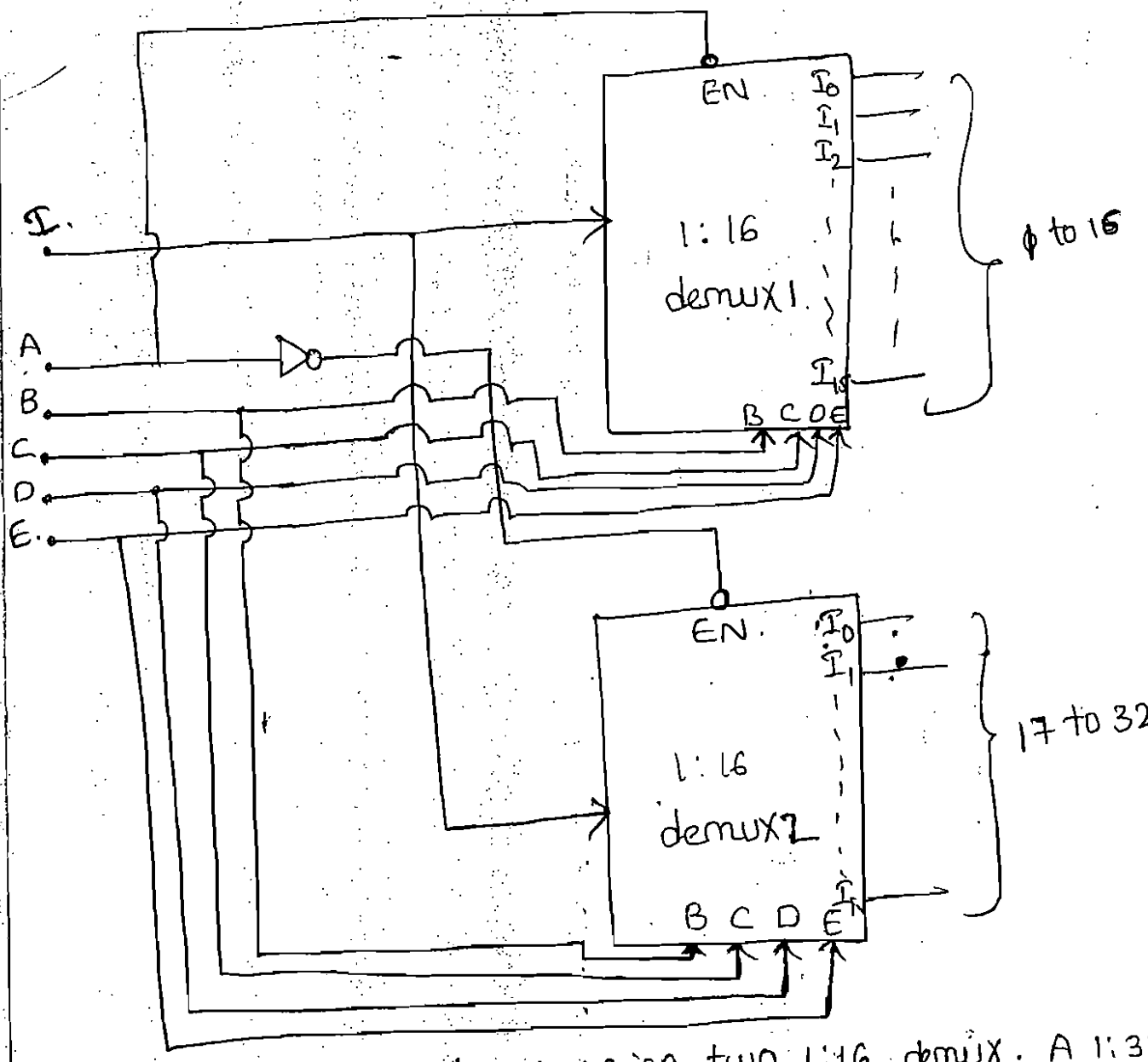
1-line to 8-line demultiplexer:-

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

Truth table.



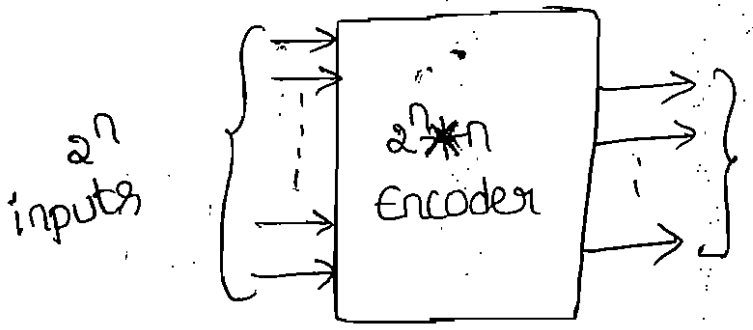
design of 1:32 demux using two 1:16 demux.



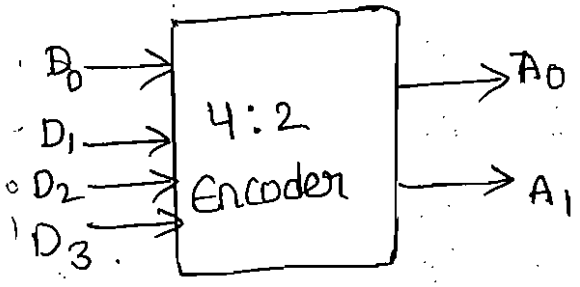
To obtain 1:32 demux using two 1:16 demux. A 1:32 demux has 32 data outputs. So it requires five data select lines. Since 1:16 demux has only four select inputs. the inputs B, C, D, E are connected to the data select lines of both the 1:16 demuxes and the most significant input A is connected to the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

Encoders :-

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



4bit - Encoder :-



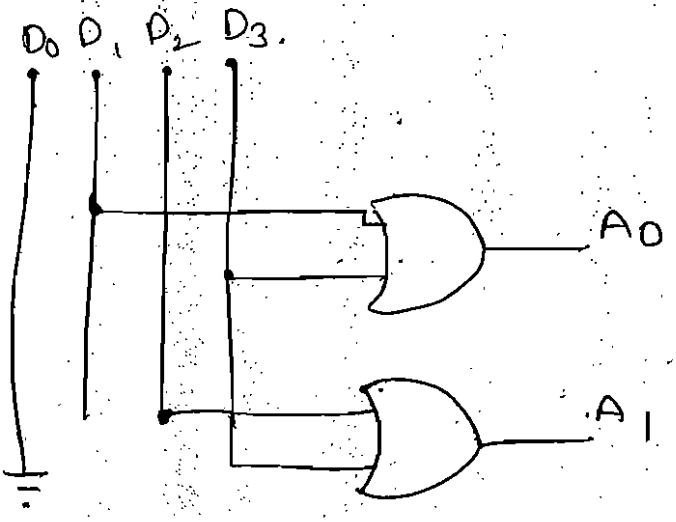
Block diagram.

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

Truth table.

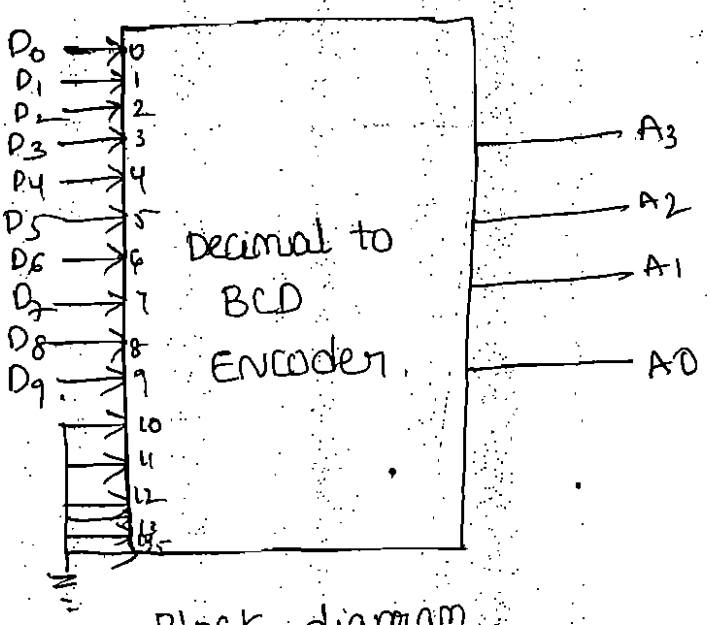
$A_1 = D_2 + D_3$

$A_0 = D_1 + D_3$.



Decimal to BCD Encoder :-

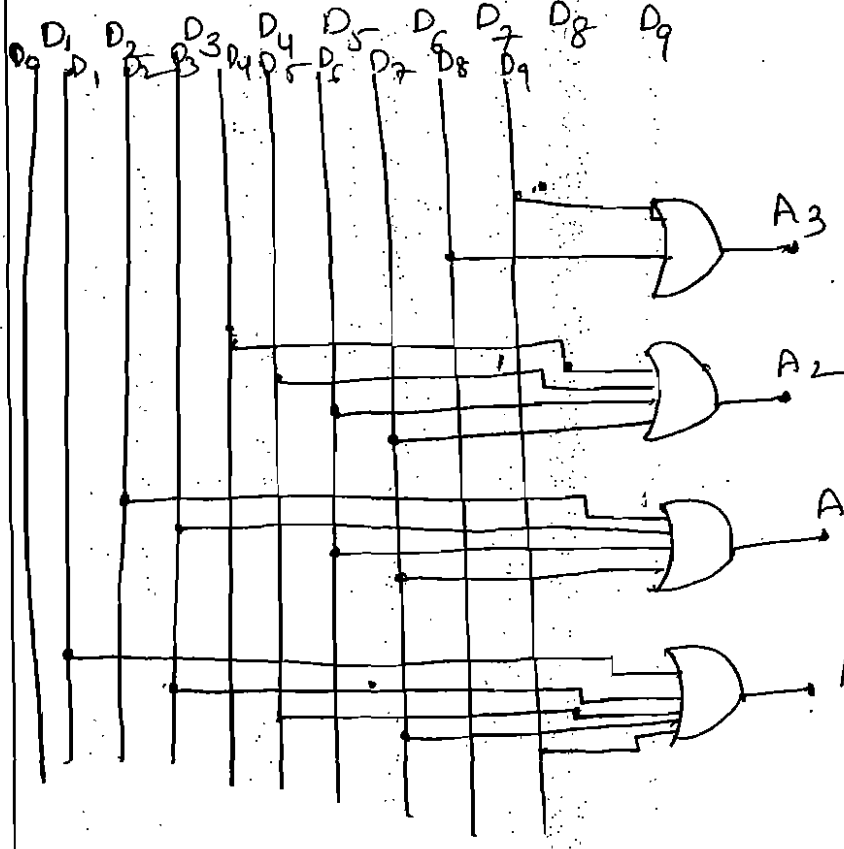
In this type of encoder has 10 inputs - one for each decimal digit, and 4 outputs corresponding to the BCD code.



Block diagram.

inputs. Decimal		Binary output A_3 A_2 A_1 A_0
D_0	0	0 0 0 0
D_1	1	0 0 0 1
D_2	2	0 0 1 0
D_3	3	0 0 1 1
D_4	4	0 1 0 0
D_5	5	0 1 0 1
D_6	6	0 1 1 0
D_7	7	0 1 1 1
D_8	8	1 0 0 0
D_9	9	1 0 0 1

Truth table.



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

$A_1 = D_2 + D_3 + D_6 + D_7$

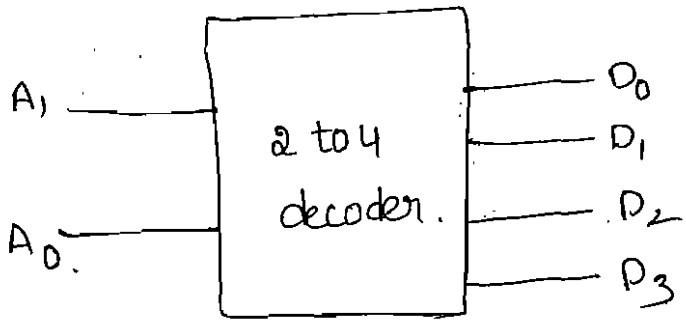
$A_2 = D_4 + D_5 + D_6 + D_7$

$A_3 = D_8 + D_9$.

Decoders :-

A decoder is a logic circuit that converts an N-bit binary input code into 2^n output lines such that only one output line is activated for each one of the possible combinations of inputs. For each of these input combinations, only one of the output will be activated (high), all the other outputs will remain inactive. (low). Some decoders are designed to produce active low output, while all the other outputs remain high.

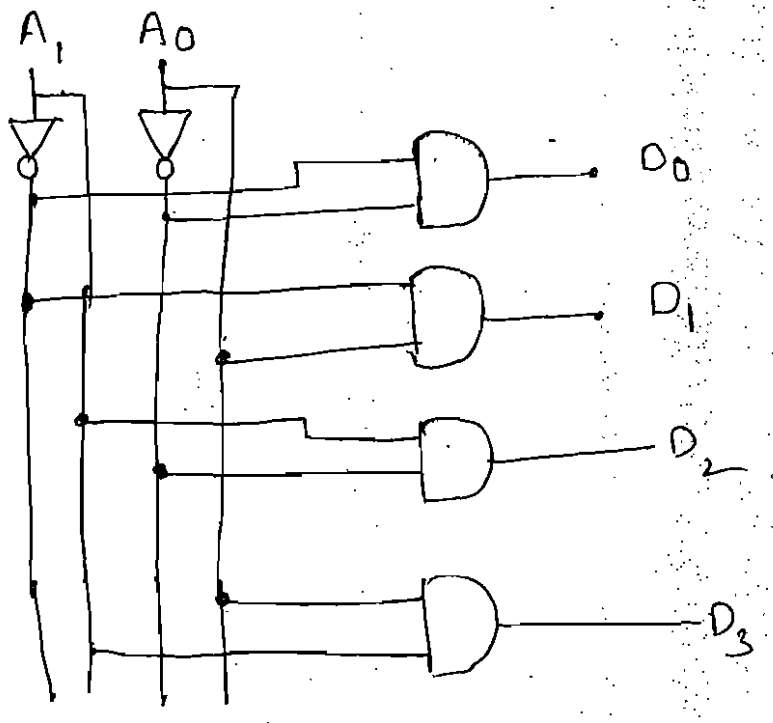
2-4 line decoder :-



Block diagram.

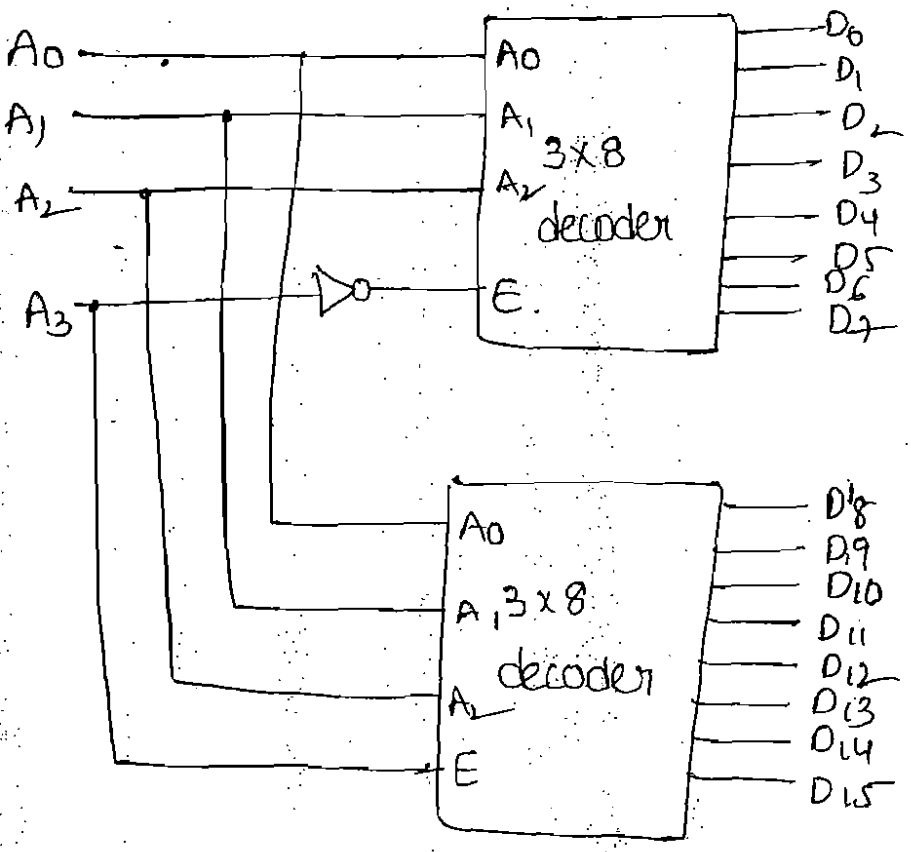
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Truth table.



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

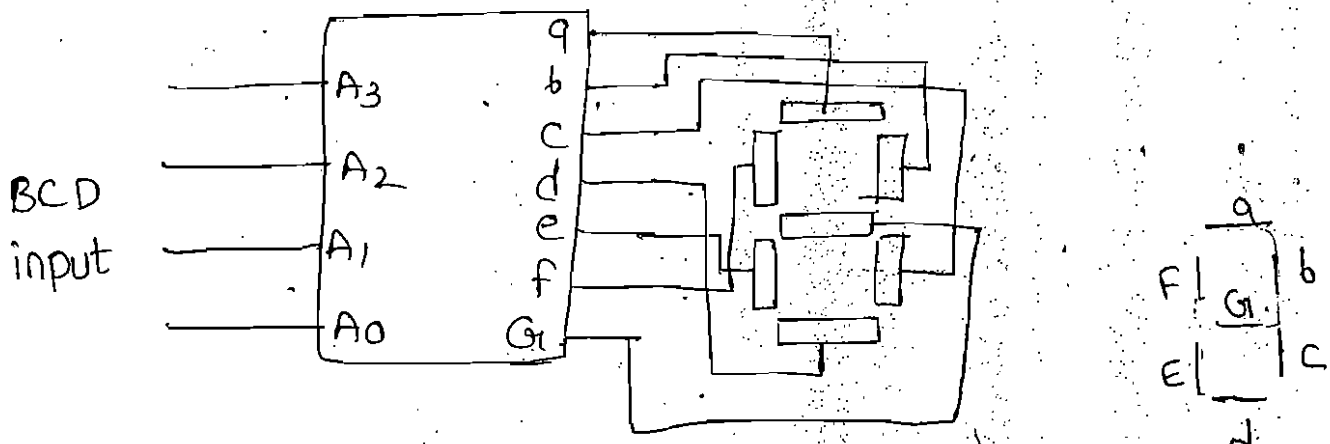
Decoders with enable inputs can be connected together to form a larger decoder. To obtain a 4-to-16 decoder it requires two 3-to-8 decoders. The most significant input bit A_3 is connected through an inverter to $\bar{E}$ on the upper decoder and directly to E on the lower decoder. Thus A_3 is low, the upper decoder is enabled and the lower decoder is disabled. The bottom decoder outputs all 0's, and top 8 outputs. generates minterms when A_3 is high, the lower decoder is enabled and the upper decoder is disabled. The bottom decoder outputs generates minterms 1000 to 1111 while the outputs of the top decoder are all 0's.



logic diagram

Seven Segment Decoders :-

This type of decoders accepts the BCD code and provides outputs to energize seven segment display devices in order to produce a decimal read out. Each segment is made up of a material that emits light when current is passed through it. The most commonly used materials include LEDs, incandescent filaments and LCDs.



Block diagram.

Decimal digit	BCD input A_3 A_2 A_1 A_0	Seven segment code a b c d e f G
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 0 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 0 1 1 1

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

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

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

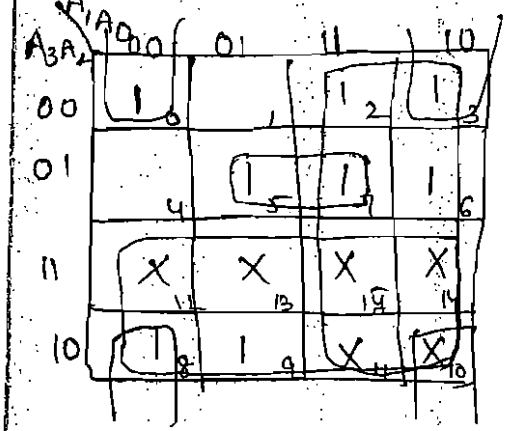
$d = \Sigma m(0,2,3,5,6,8,9) + d(10,11,12,13,14,15)$.

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

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

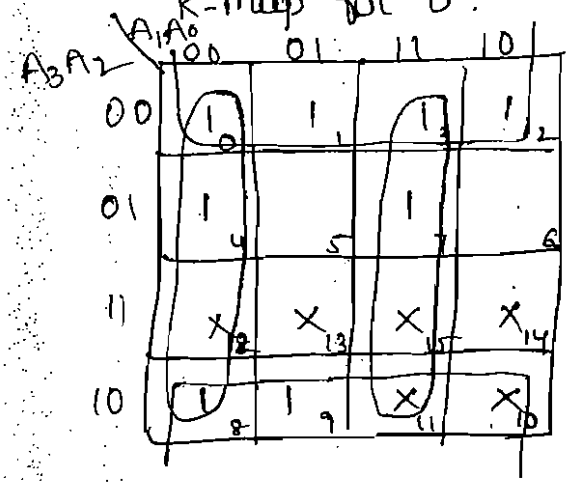
$g = \Sigma m(2,3,4,5,6,8,9) + d(10,11,12,13,14,15)$.

K-map for a.

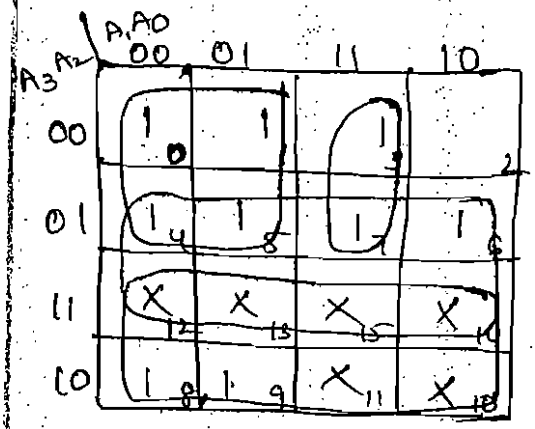


$a = A_1 + A_3 + \overline{A_2}\,\overline{A_0} + \overline{A_3}A_2A_0$

K-map for b.

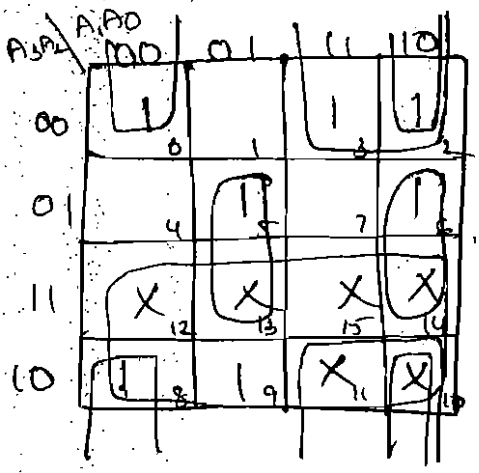


$b = \overline{A_2} + A_1\overline{A_0} + A_1A_0$



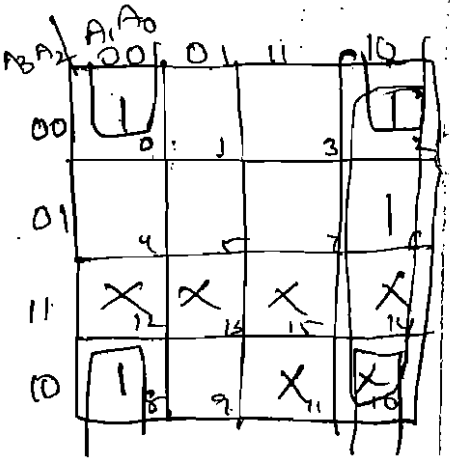
K-map for c

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



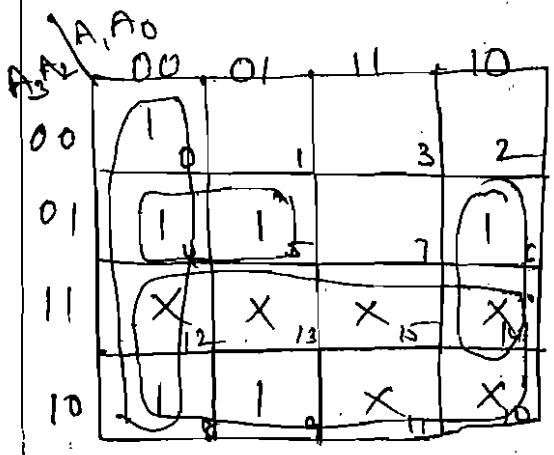
K-map for d

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

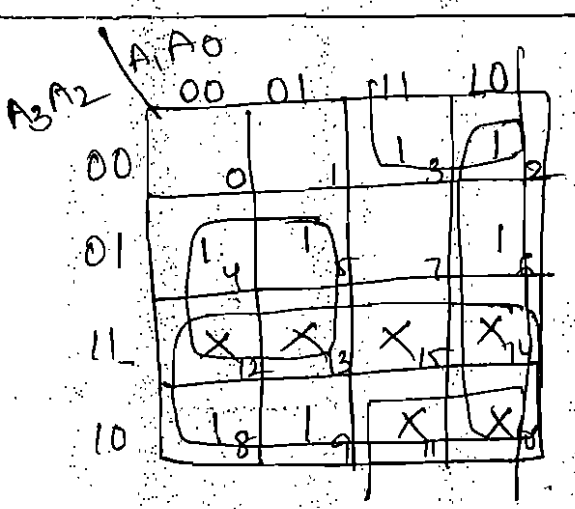


K-map for e

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



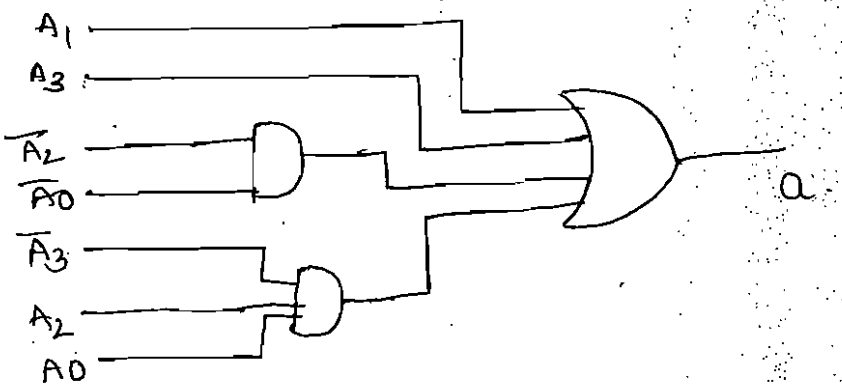
K-map for f.

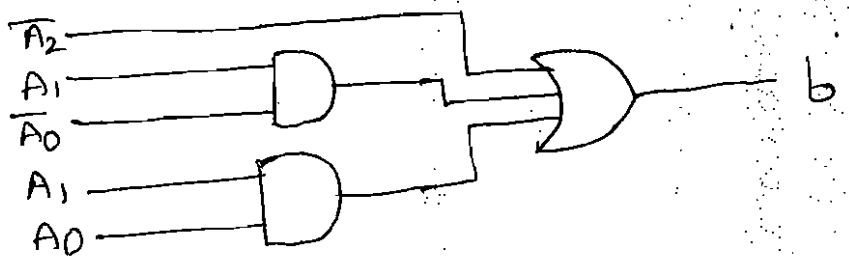


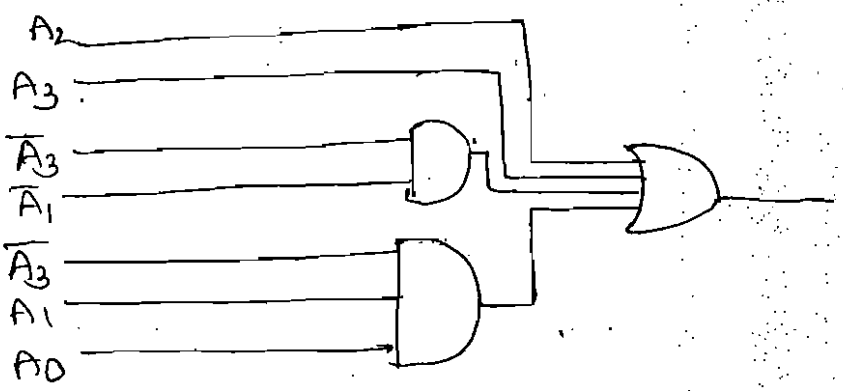
K-map for G1.

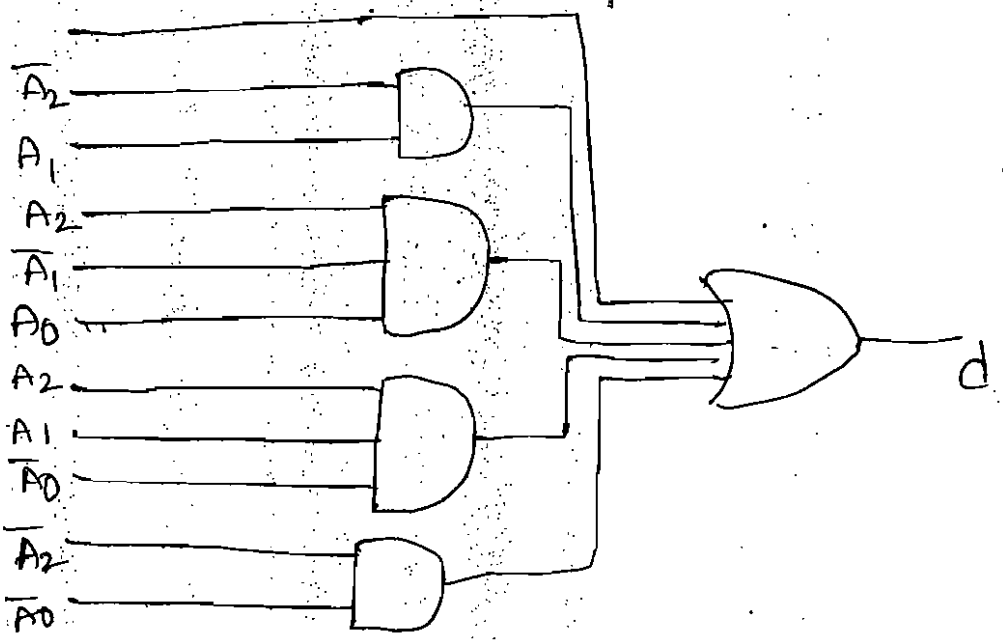
$f = \bar{A}_1\bar{A}_0 + A_3 + \bar{A}_3A_2\bar{A}_1 + A_2A_1\bar{A}_0$

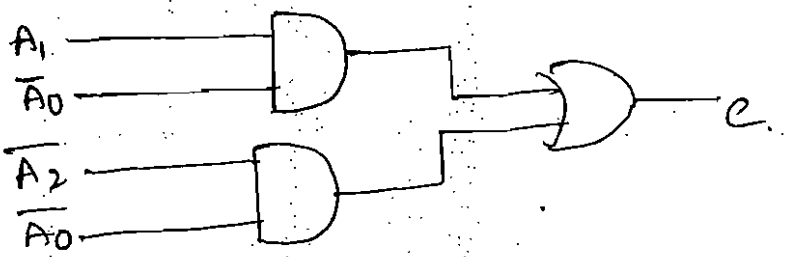
$G_1 = A_3 + A_1\bar{A}_0 + A_2\bar{A}_1 + \bar{A}_2A_1$

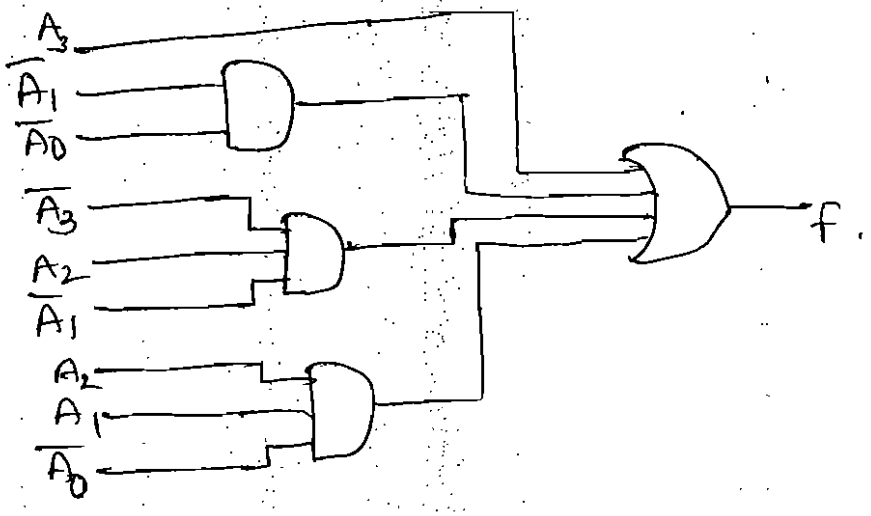


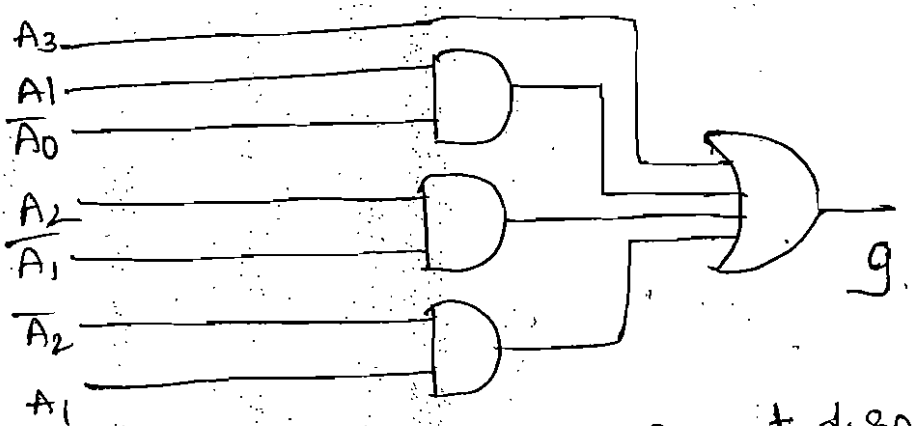








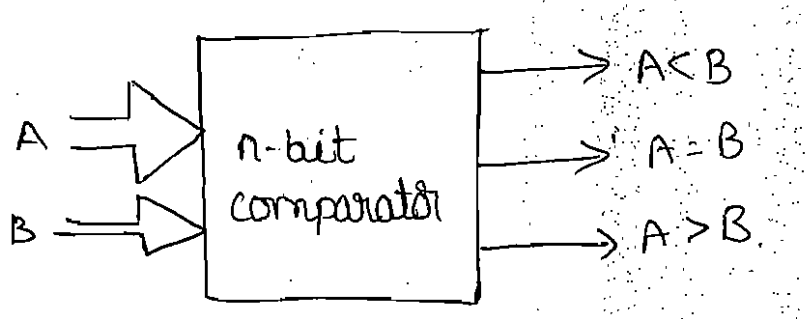




diagrams for seven segment display.

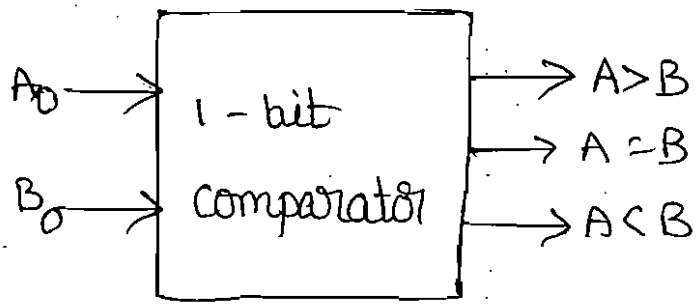
Comparator :-

The comparator is a combinational logic circuit It compares the magnitude of two n-bit numbers and provides the relative result as the output The block diagram of an n-bit digital comparator has 2 inputs and three outputs. A and B are the n-bit inputs. The comparator outputs are A>B, A=B and A<B. Depending upon the result of comparsion, one of these outputs will be high.



1-bit Comparator :-

The one bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely A<B, A=B, A>B.



Block diagram.

Inputs		outputs		
A_0	B_0	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

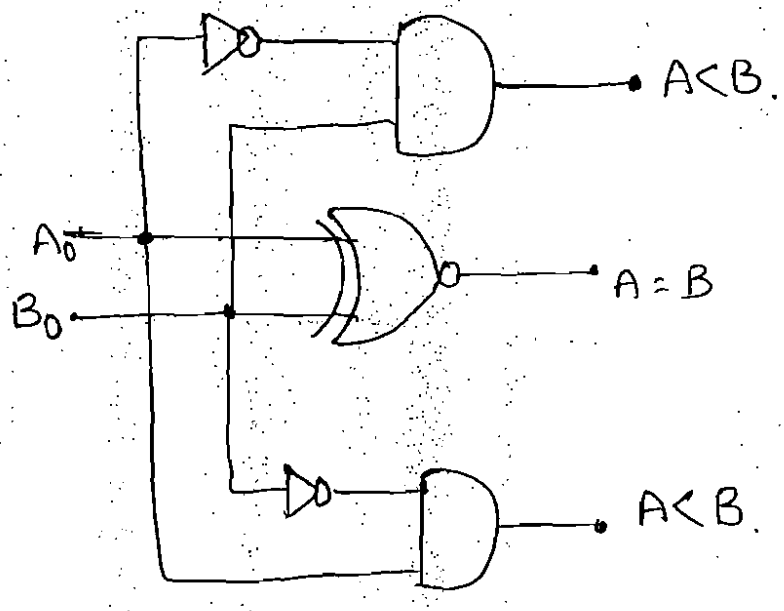
Truth table

In truth table

$A=B : \overline{A_0}\,\overline{B_0} + A_0 B_0 = A_0 \odot B_0$

$A<B : \overline{A_0} B_0$

$A>B : A_0 \overline{B_0}$



2-bit comparator :-

The logic for a 2-bit comparator. let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

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

→ if A_1 and B_1 are equal and $A_0 = 1$ and $B_0 = 0$ then $A > B$.

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

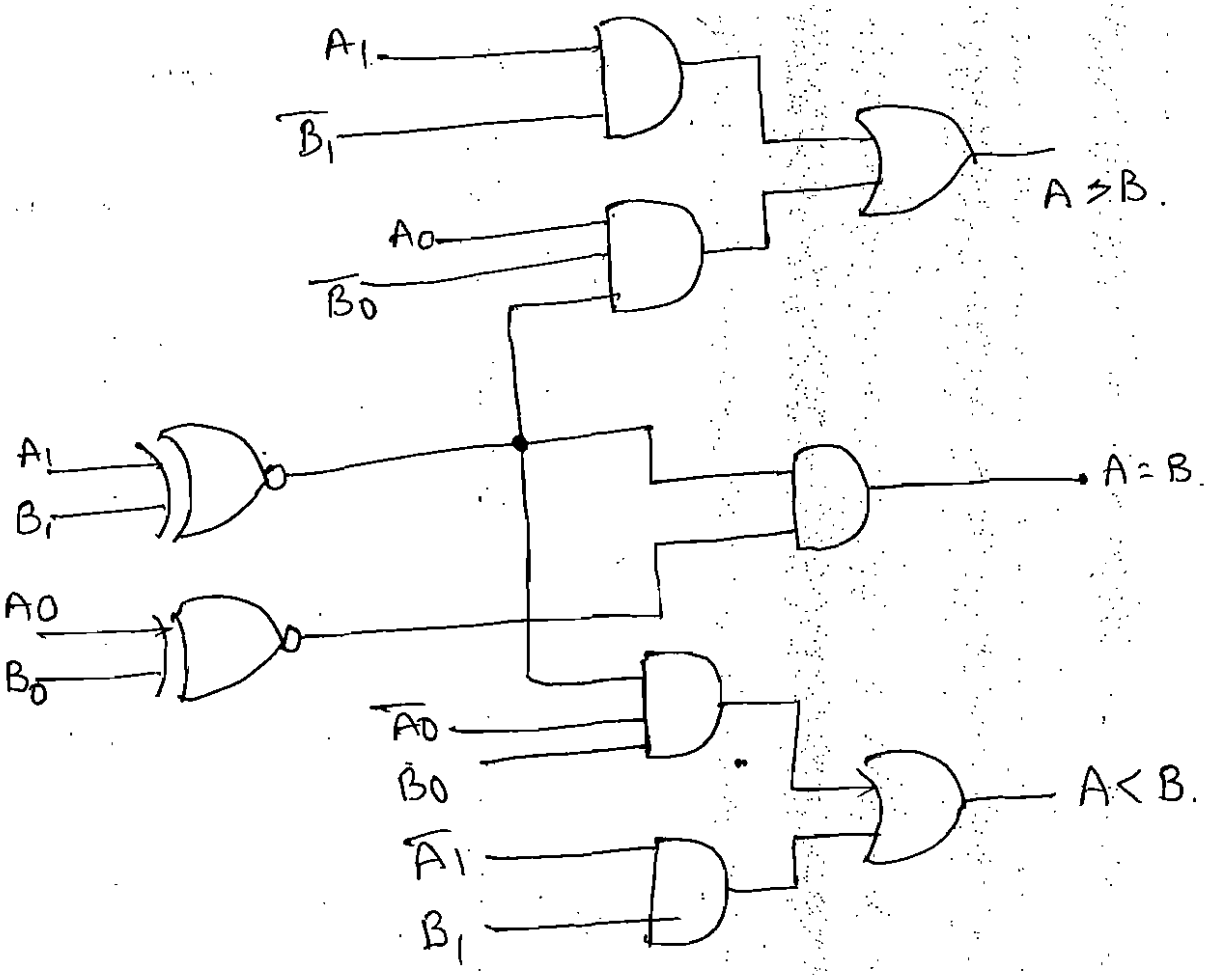
→ if $B_1 = 1$ and $A_1 = 0$ then $A < B$ or

→ if B_1 and A_1 are equal and $B_0 = 1$ and $A_0 = 0$ then $A < B$

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

→ if A_1 and B_1 are equal and if A_0 and B_0 are equal then $A = B$

$A = B : (A_1 \odot B_1)(A_0 \odot B_0)$



4-bit comparator :-

The logic for a 4-bit comparator. let the four bit numbers will be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

→ if $A_3 = 1$ and $B_3 = 0$, then $A > B$ or

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

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

→ if A_3 and B_3 are equal, and if A_2 and B_2 are equal, and if A_1 and B_1 are equal, and if $A_0 = 1$ and $B_0 = 0$.

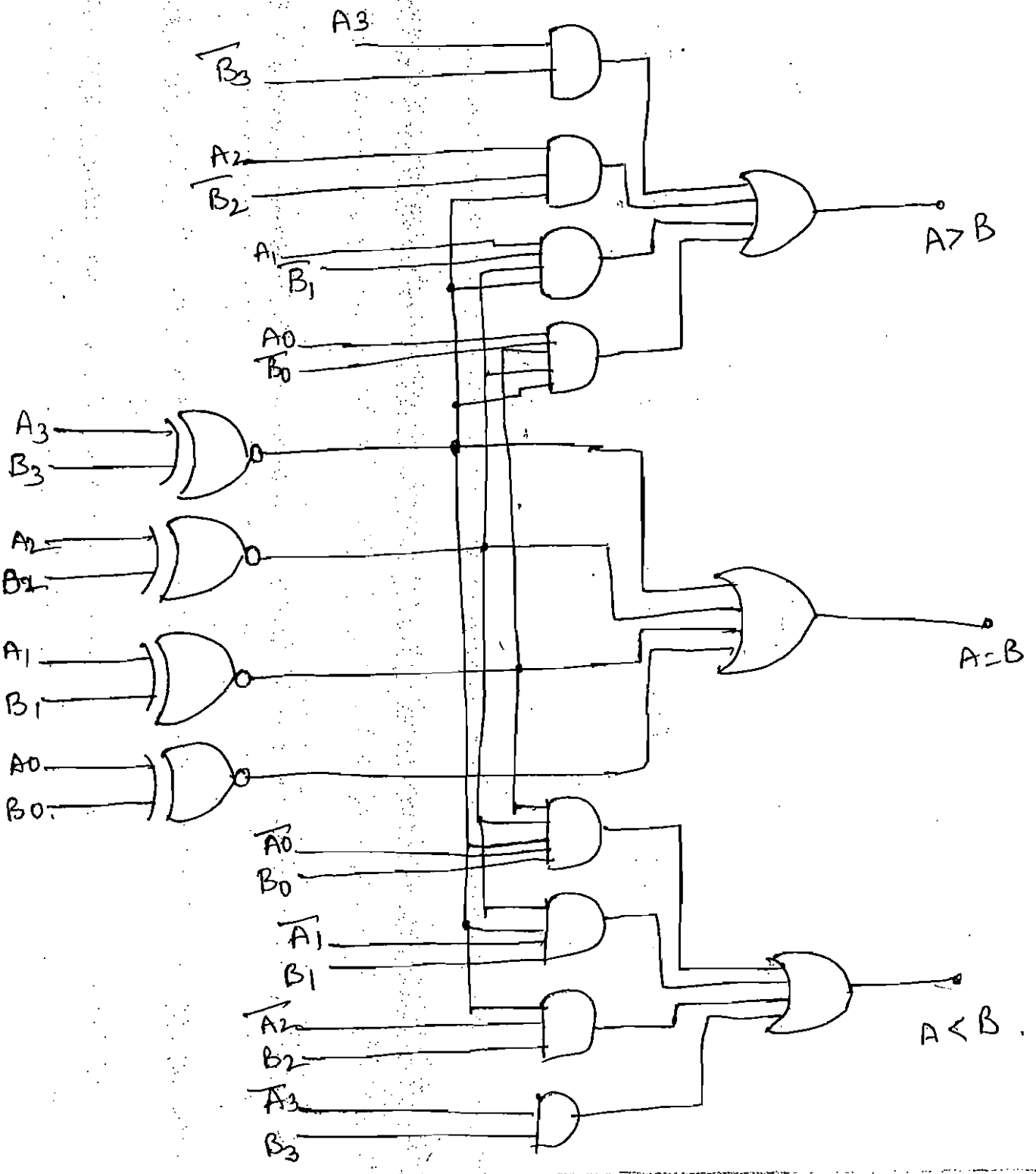
$(A>B) = A_3\overline{B_3} + (A_3 \odot B_3) A_2\overline{B_2} + (A_3 \odot B_3)(A_2 \odot B_2) A_1\overline{B_1}$

$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1) A_0\overline{B_0}$

$(A<B) = \overline{A_3} B_3 + (A_3 \odot B_3)\overline{A_2} B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\overline{A_1} B_1$

$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\overline{A_0} B_0$

$A = B : (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0).$



Logic diagram of 4-bit comparator.

priority Encoders :-

It is possible that two or more inputs are active at a time. To overcome this, priority encoders are used. A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously high. The most common priority system is based on the relative magnitudes of the inputs.

In some practical applications, priority encoders may have several inputs that are routinely high at the same time, and the principal function of the encoder in those cases is to select the input with the highest priority.

4-input priority Encoder :-

In 4-input priority Encoder in addition to the outputs A and B, the circuit has a third output designed by V. This is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't care conditions.

Truth table

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

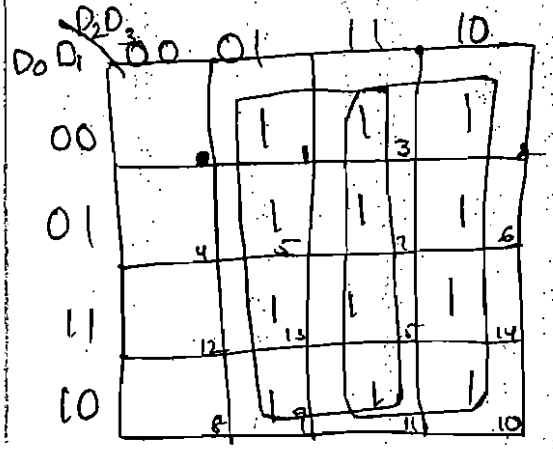
$V = D_0 + D_1 + D_2 + D_3$.

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

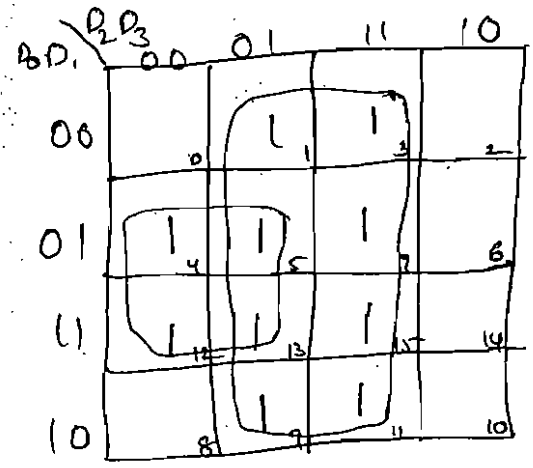
$$A = \Sigma m(1,2,3,5,6,7,9,10,11,13,14,15)$$

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

K-map for A

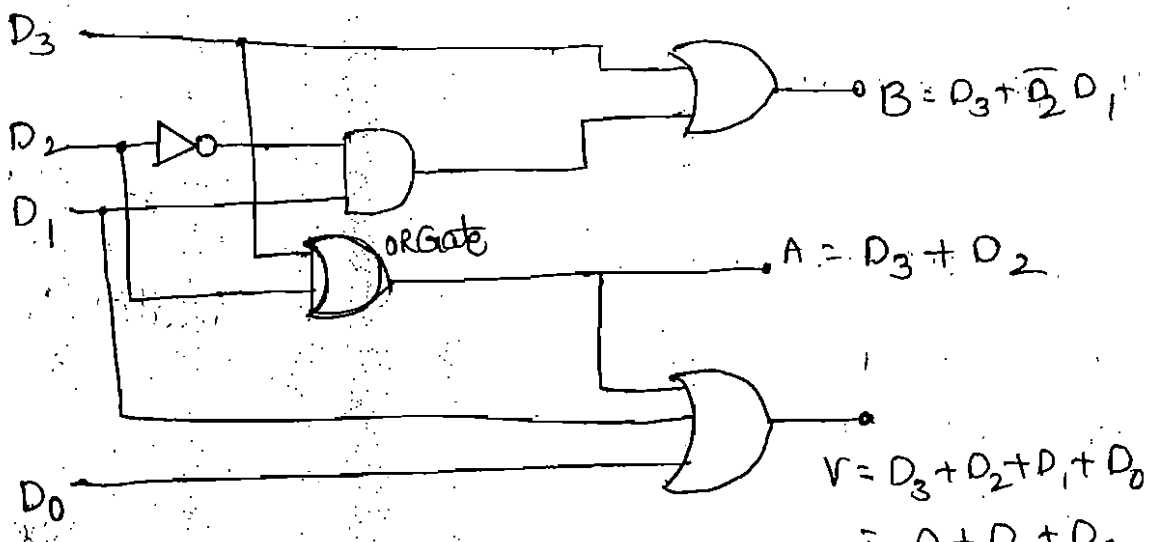


K-map for B.



$A = D_3 + D_2$

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



Logic diagram for

4-bit priority Encoder.

PROGRAMMABLE LOGIC DEVICES

→ Logic designers have a wide range of standard IC's available to them with numerous logic functions and logic circuit arrangements on a chip. In addition, these ICs are available from many manufactures and at a reasonably low cost.

→ PLD is an IC that contains large number of gates, flip-flops and registers that are intercomected on chip. This IC is said to be programmable because the specific function IC is determined by Intercomecting required contacts.

Basically, there are three types of programmable device which are.

→ Read only memory. (ROM)

→ programmable logic array (PLA)

→ programmiable Array logic (PAL)

READ ONLY memory :-

→ The read only memory is a type of semiconductor memory that is designed to hold data that is either permanent or will not change frequently.

→ During operation, no new data can be written into a ROM, but data can be read from ROM. the process of Entering data is called programming or burung-in the ROM.

→ Some ROM's cannot have their data changes once they have been programmed others can be erased and reprogrammed as often as desired.

Types of ROM's :-

1. Masked memory ROM
2. programmable Read only memory (PROM)
3. Erasable programmable Read only memory (EPROM)
4. Electrically Erasable programmable Read only memory (EEPROM)

Masked memory (Rom) :-

→ cannot be reprogrammed
→ Nonvolatile, retain data even when power is turn off.
→ cheaper than programmable devices.
→ useful for fixed programme instructions.

PROM

→ programmed by blowing built-in fuses.
→ can not be reprogrammed
→ Non volatile.
→ useful for small volume data storing
→ user programmable

EPROM :-

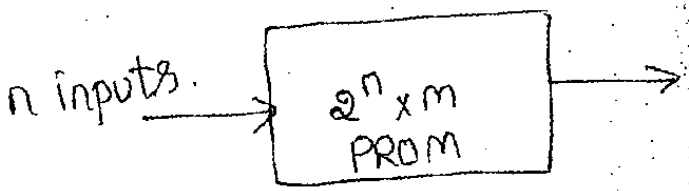
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.
→ NON volatile

Programmable ROM :-

→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination that comes out of the output lines is called a word.



Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array provide an AND-OR Sum of products implementati

EPROM :-

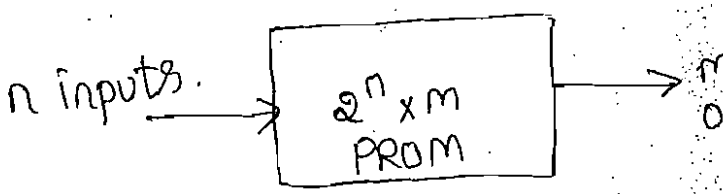
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.
→ NON volatile

Programmable ROM :-

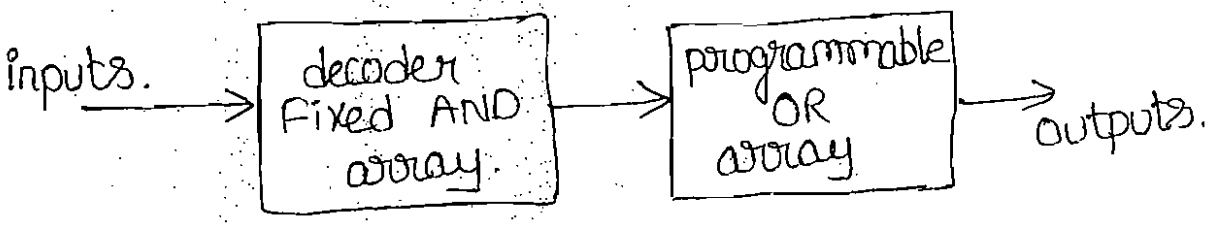
→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination that comes out of the output lines is called a word.

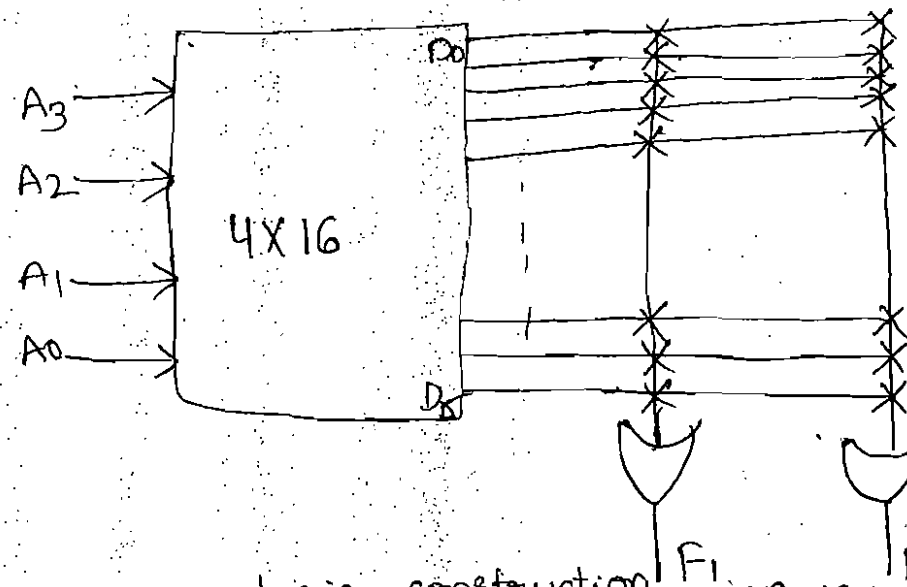


Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR Sum of products implementation.

The programmable read-only memory (PROM) has a fixed AND array constructed as a decoder and a programmable OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR Gate.





Logic construction using 16X2 ROM.

→ Implement full-adder using PROM.

The number of inputs variables of a full adder are 3. The possible number of combinations are 8. So we need 3X8 decoder. The number of outputs of full adder are 2. They are Sum and Carry.

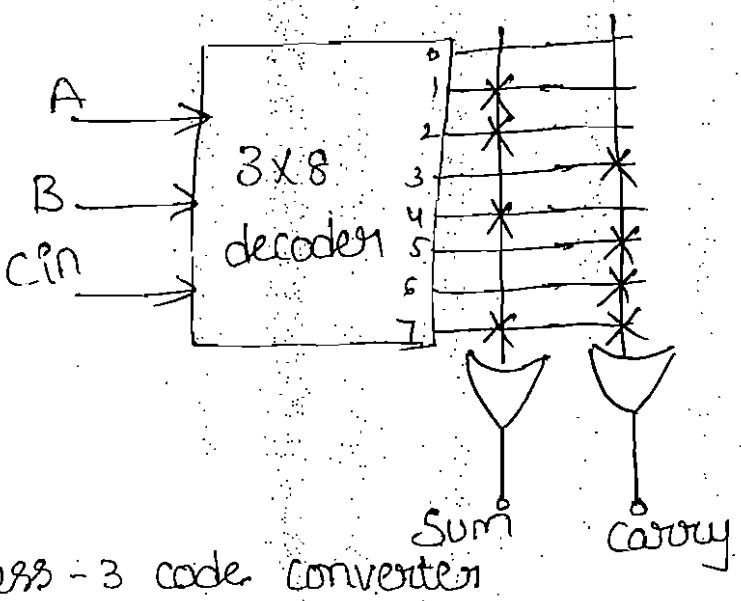
Truth table

A	B	Cin	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

output expression for Sum & carry is Sum = $\Sigma m(1,2,4,7)$

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

Logic diagram



→ Design BCD to Excess-3 code converter using PROM.

Step:- 1 - Truth table.

BCD				EXCESS-3			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1

Step 2 :-

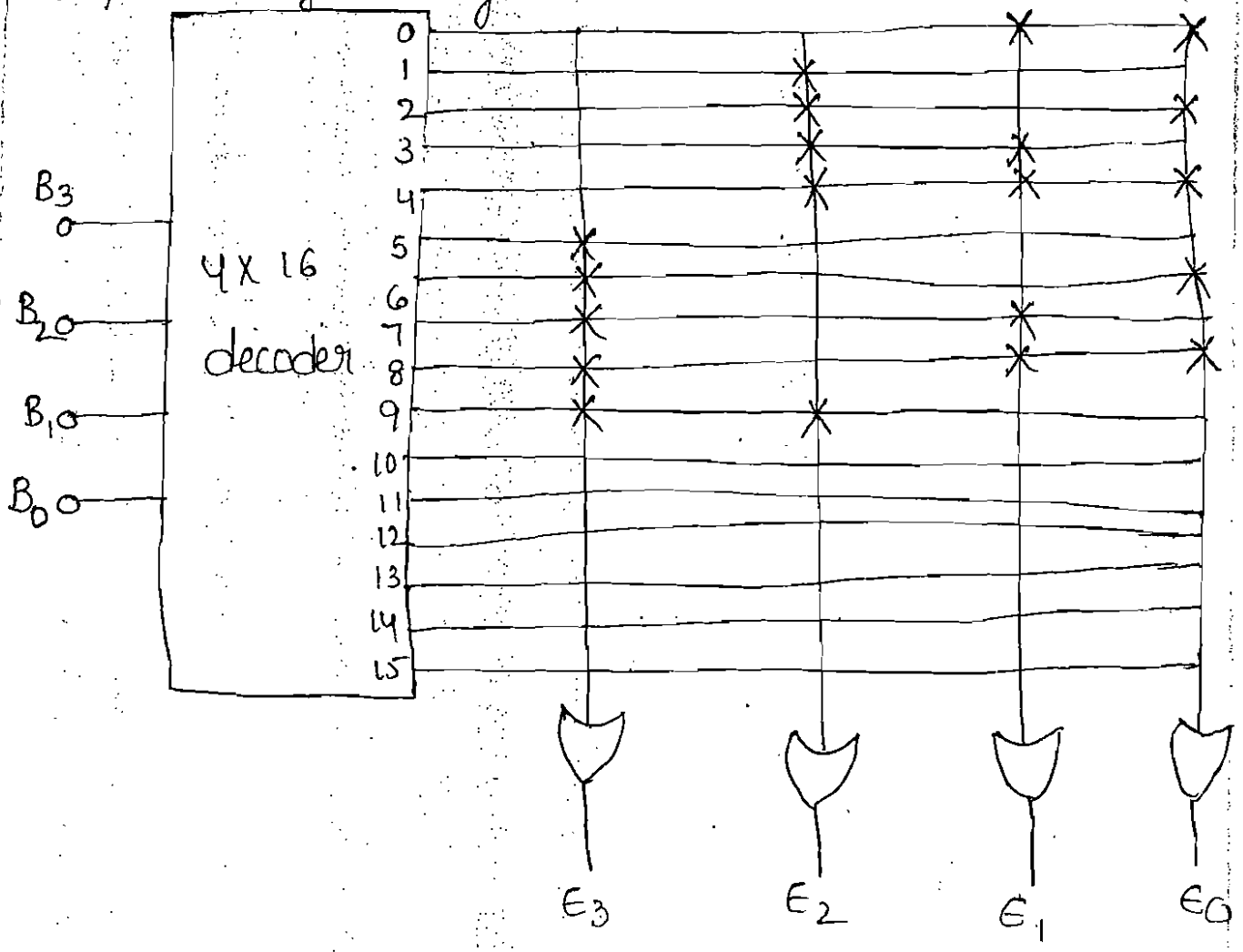
$E_0(B_3\ B_2\ B_1\ B_0) = \Sigma m(0,2,4,6,8)$

$E_1(B_3\ B_2\ B_1\ B_0) = \Sigma m(0,3,4,7,8)$

$E_2(B_3\ B_2\ B_1\ B_0) = \Sigma m(1,2,3,4,9)$

$E_3(B_3\ B_2\ B_1\ B_0) = \Sigma m(5,6,7,8,9)$

Step 3 :- logic diagram.



→ Implement the following Boolean expression using PROM.

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

first given expression converted into a standard sop form.

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

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

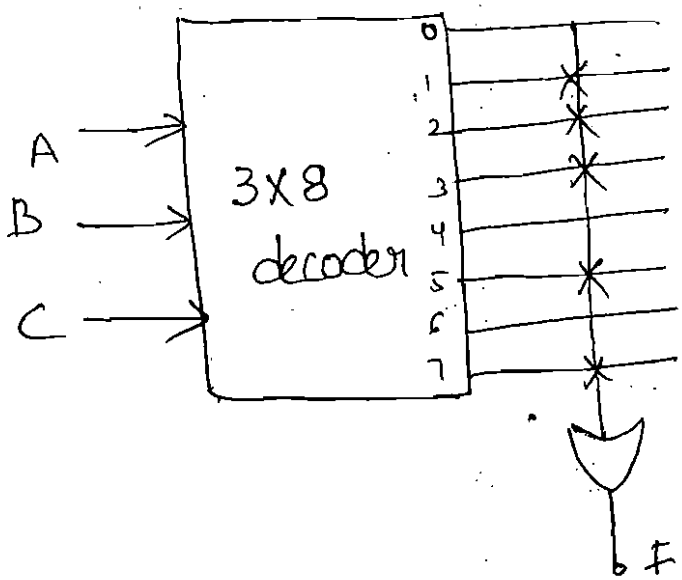
$$= \underset{1}{\bar{A}BC} + \bar{A}B\bar{C} + \underset{2}{ABC} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \underset{2}{AB\bar{C}} + \underset{1}{\bar{A}BC}$$

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

$011, 010, 111, 101, 011, 001$

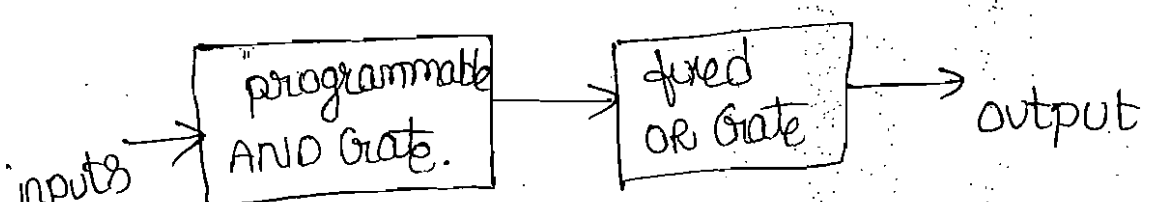
$f(A,B,C) = \Sigma m(1,2,3,5,7)$

Logic diagram :-



PAL :- programmable Array logic :-

The programmable Array logic is a programmable device with a fixed OR array and a programmable AND array because, only the AND gates are programmble.

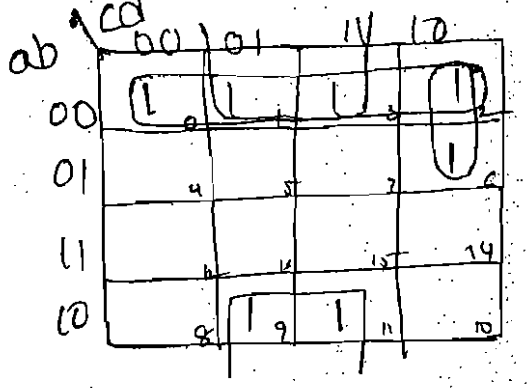


→ Implement the following functions using PAL.

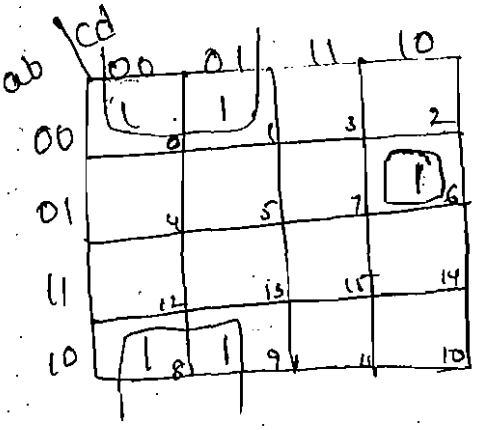
$F_1(a,b,c,d) = \Sigma m(0,1,2,3,6,9,11)$

$F_2(a,b,c,d) = \Sigma m(0,1,6,8,9)$

Step 1:- K-map simplification for F_1 and F_2



$\bar{a}\bar{b} + \bar{a}c\bar{d} + \bar{b}d$

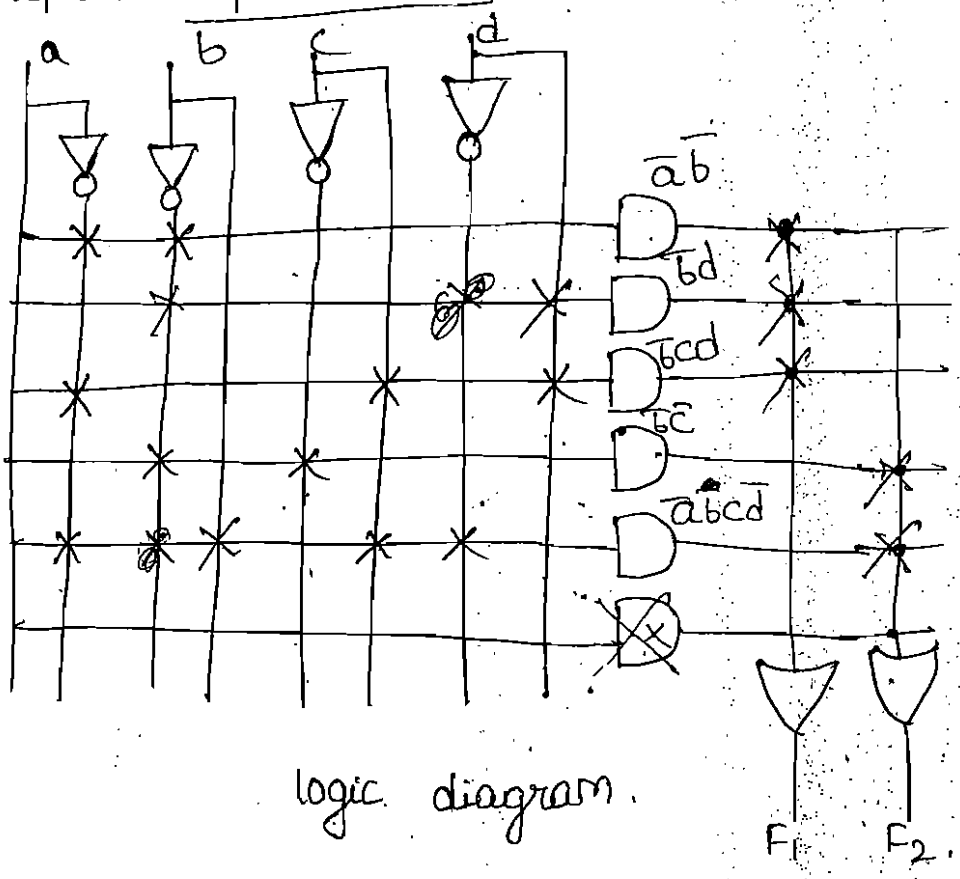


$\bar{b}\bar{c} + \bar{a}bc\bar{d}$

Step 2 :- PAL programming table.

product terms	AND gate inputs a	b	c	d	outputs
1	0	0	–	–	$F_1 = \bar{a}\bar{b} + \bar{a}cd + \bar{b}d$
2	0	–	1	1	
3	–	0	–	1	
4	–	0	0	–	$F_2 = \bar{b}\bar{c} + \bar{a}bc\bar{d}$
5	0	1	1	0	
6	–	–	–	–	

Step 3:- implementation:-



logic diagram.

→ Implement 4-bit BCD to XS-3 code conversion using PAL.

Step:-1

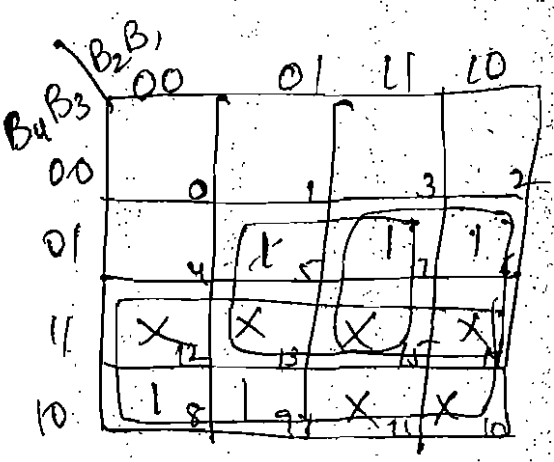
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

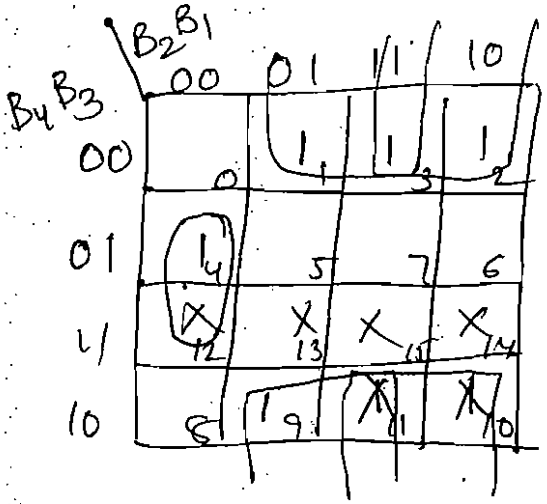
$X_4 = \Sigma m(5,6,7,8,9) + d(10,11,12,13,14,15)$

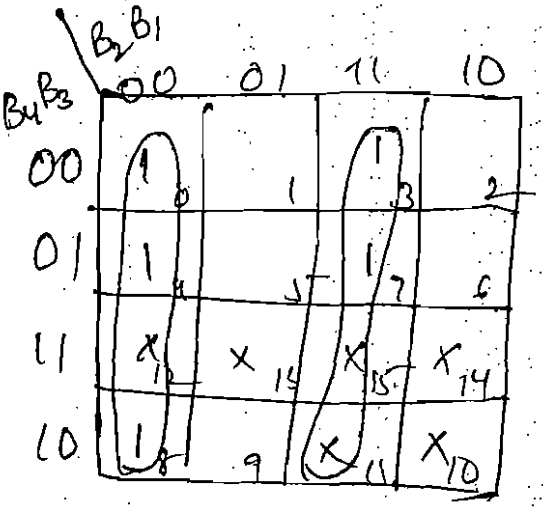
$X_3 = \Sigma m(1,2,3,4,9) + d(10,11,12,13,14,15)$

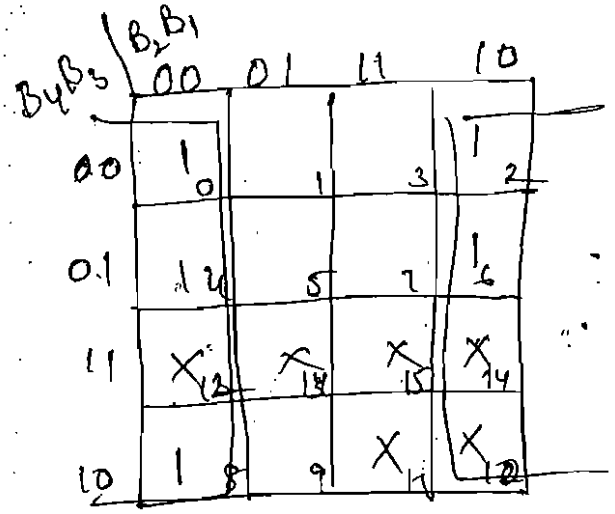
$X_2 = \Sigma m(0,3,4,7,8) + d(10,11,12,13,14,15)$

$X_1 = \Sigma m(0,2,4,6,8) + d(10,11,12,13,14,15)$









$X_4 = B_4 + B_3B_2 + B_3B_1$

$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$

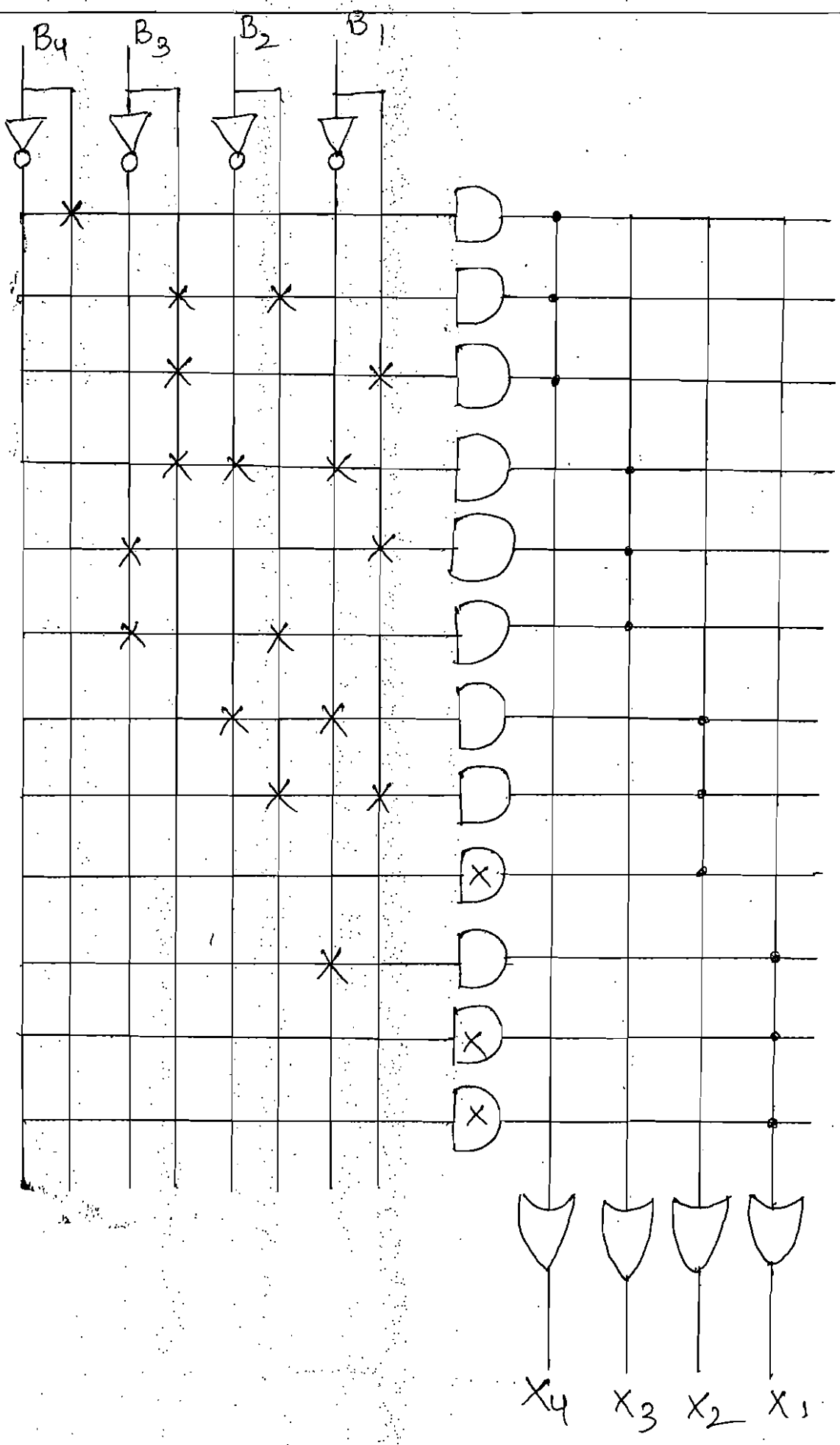
$X_2 = \overline{B_2\overline{B_1} + B_2B_1}$

$X_1 = \overline{B_1}$

Step 2 :- programming table.

product terms.	AND Gate inputs. B_4	B_3	B_2	B_1	Outputs.
1	1	—	—	—	$X_4 = B_4 + B_3B_2 + B_3B_1$
2	—	1	1	—	
3	—	1	—	1	
4	—	1	0	0	$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$
5	—	0	—	1	
6	—	0	1	—	
7	—	—	0	0	$X_2 = \overline{B_2}\,\overline{B_1} + B_2B_1$
8	—	—	1	1	
9	—	—	—	—	
10	—	—	—	0	$X_1 = \overline{B_1}$
11	—	—	—	—	
12	—	—	—	—	

Step 3 :- logic diagram.



logic diagram.

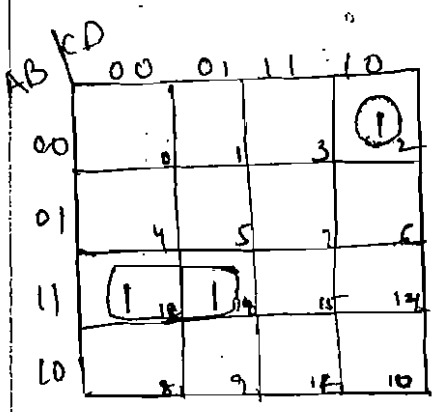
Implement the following boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

$F_1(A,B,C,D) = \Sigma m(2,12,13)$

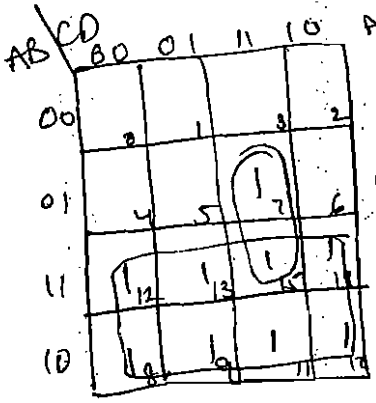
$F_2(A,B,C,D) = \Sigma m(7,8,9,10,11,12,13,14,15)$

$F_3(A,B,C,D) = \Sigma m(0,2,3,4,5,6,7,8,10,11,15)$

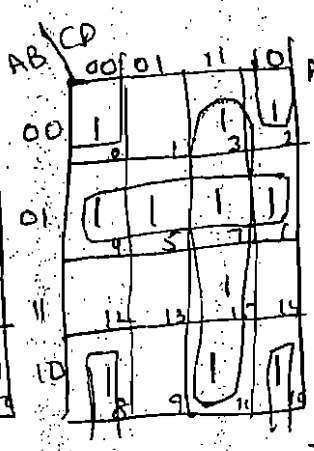
$F_4(A,B,C,D) = \Sigma m(1,2,8,12,13)$



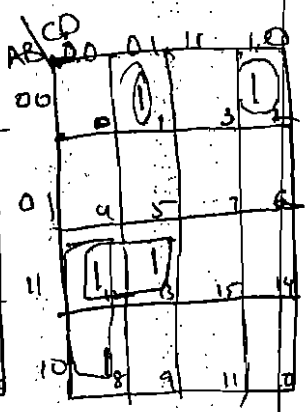
$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$



$F_2 = A + BCD$



$F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$



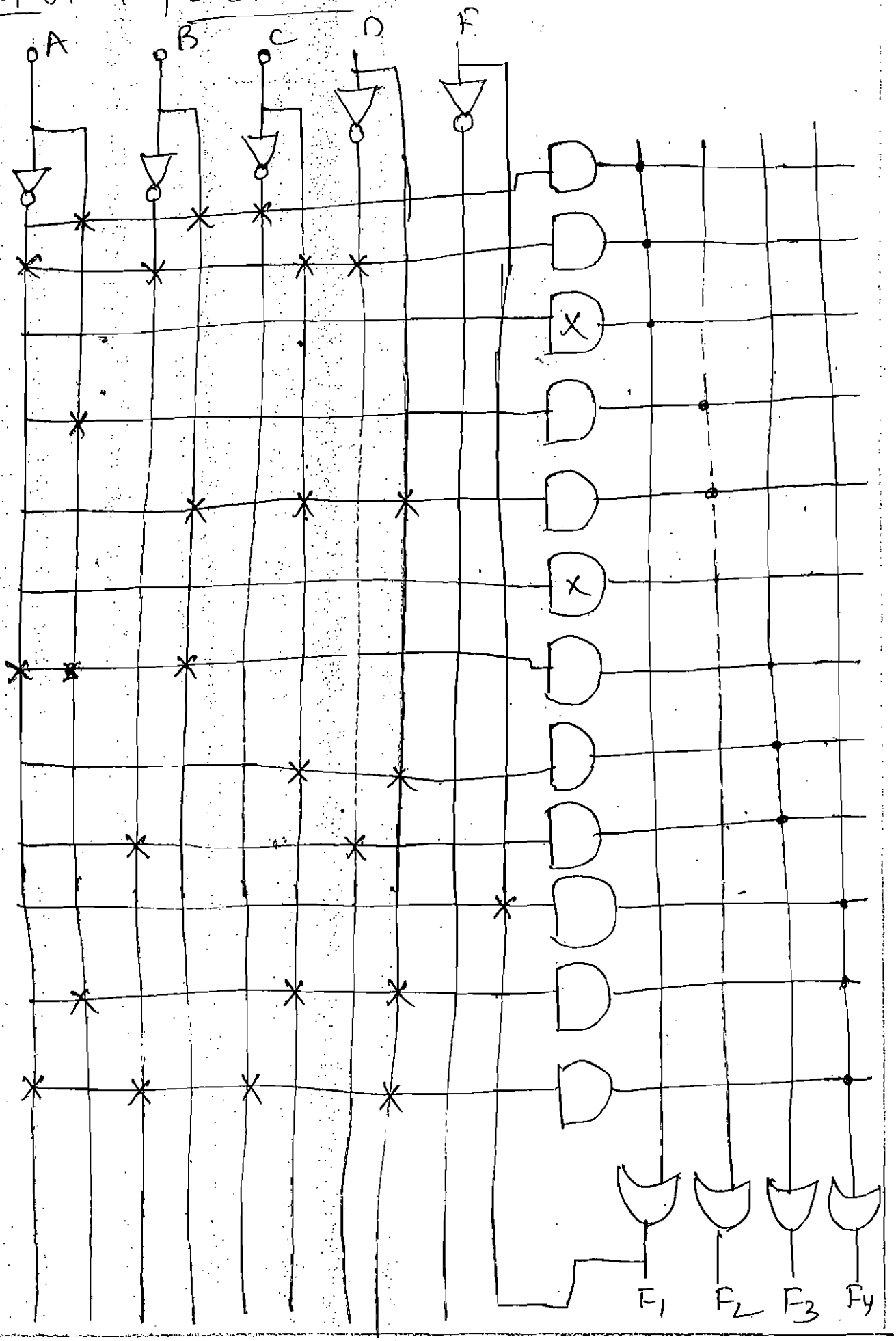
$F_4 = AB\bar{C} + A\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}\bar{B}C\bar{D}$

$= F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$

Step 2:- programming table

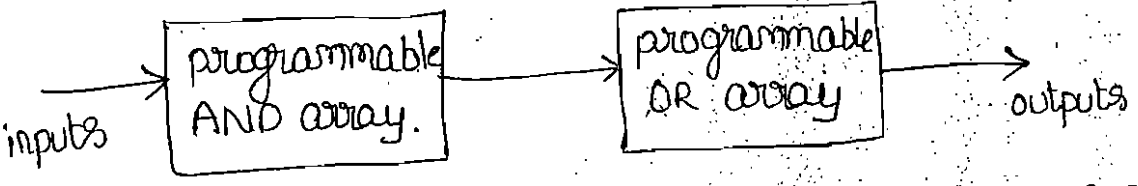
product term	AND inputs A B C D	F_1	outputs
1	1 1 0 –		$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$
2	0 0 1 0		
3	– – – –		– – –
4	1 – – –		$F_2 = A + BCD$
5	– 1 1 1		
6	– – – –		– – –
7	0 1 – –		$F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$
8	– – 1 1		
9	– 0 – 0		– – –
10	– – – –	1	
11	1 – 0 0		$F_4 = F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$
12	0 0 0 1		

Step 3 :- Implementation :-



PLA :- programmable logic Array

In programmable logic array where both the AND and OR arrays can be programmed. The product terms in the AND array may be shared by any OR gate to provide the required sum of products.

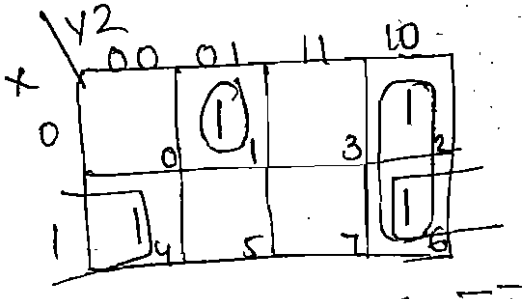


→ Implement the given boolean functions by using PLA.

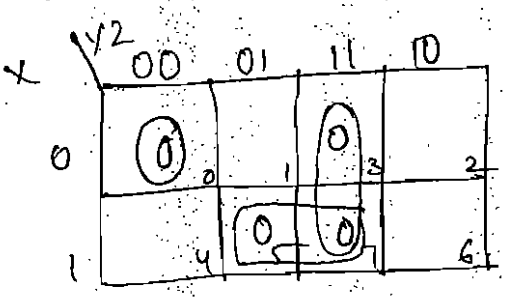
$A(X,Y,Z) = \Sigma m(1,2,4,6)$

$B(X,Y,Z) = \Sigma m(1,2,3,5,7)$

Step 1 :- The K-maps for the functions A, B, their minimization, and the minimal expressions for both the true and complement of those in sum of products.



$A(T) = X\bar{Z} + Y\bar{Z} + \bar{X}\bar{Y}Z$



$\bar{A} = XZ + YZ + \bar{X}\bar{Y}\bar{Z}$

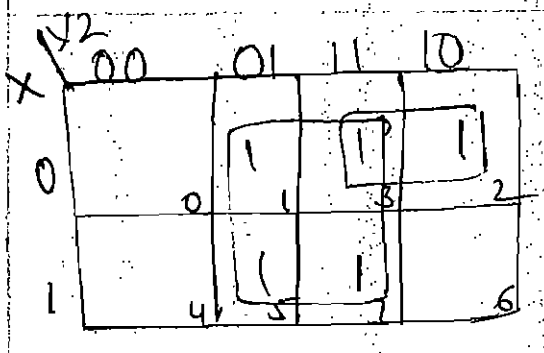
$A(C) = \overline{XZ + YZ + \bar{X}\bar{Y}\bar{Z}}$

K-map for A.

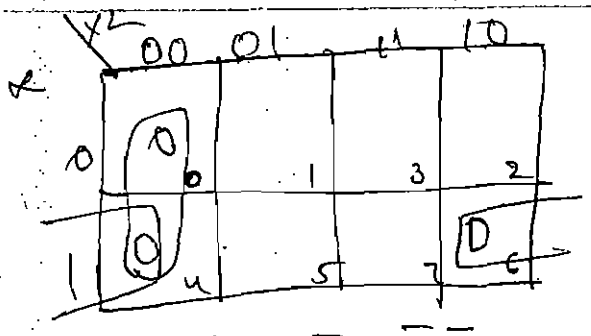
Simply $A(C) = \overline{(\bar{X}+\bar{Z})\cdot(\bar{Y}+\bar{Z})\cdot(X+Y+Z)}$

$= \overline{(\bar{X}+\bar{Z})} + \overline{(\bar{Y}+\bar{Z})} + \overline{(X+Y+Z)}$

$= XZ + YZ + \bar{X}\bar{Y}\bar{Z}$



$B(T) = \bar{X}\bar{Y} + Z$.



$\bar{B} = X\bar{Z} + \bar{Y}\bar{Z}$

$B(C) = \overline{X\bar{Z} + \bar{Y}\bar{Z}}$

Simply $B(C) = \overline{(\bar{Y}+Z)(\bar{X}+Z)}$

$= \overline{(\bar{Y}+Z)} + \overline{(\bar{X}+Z)}$

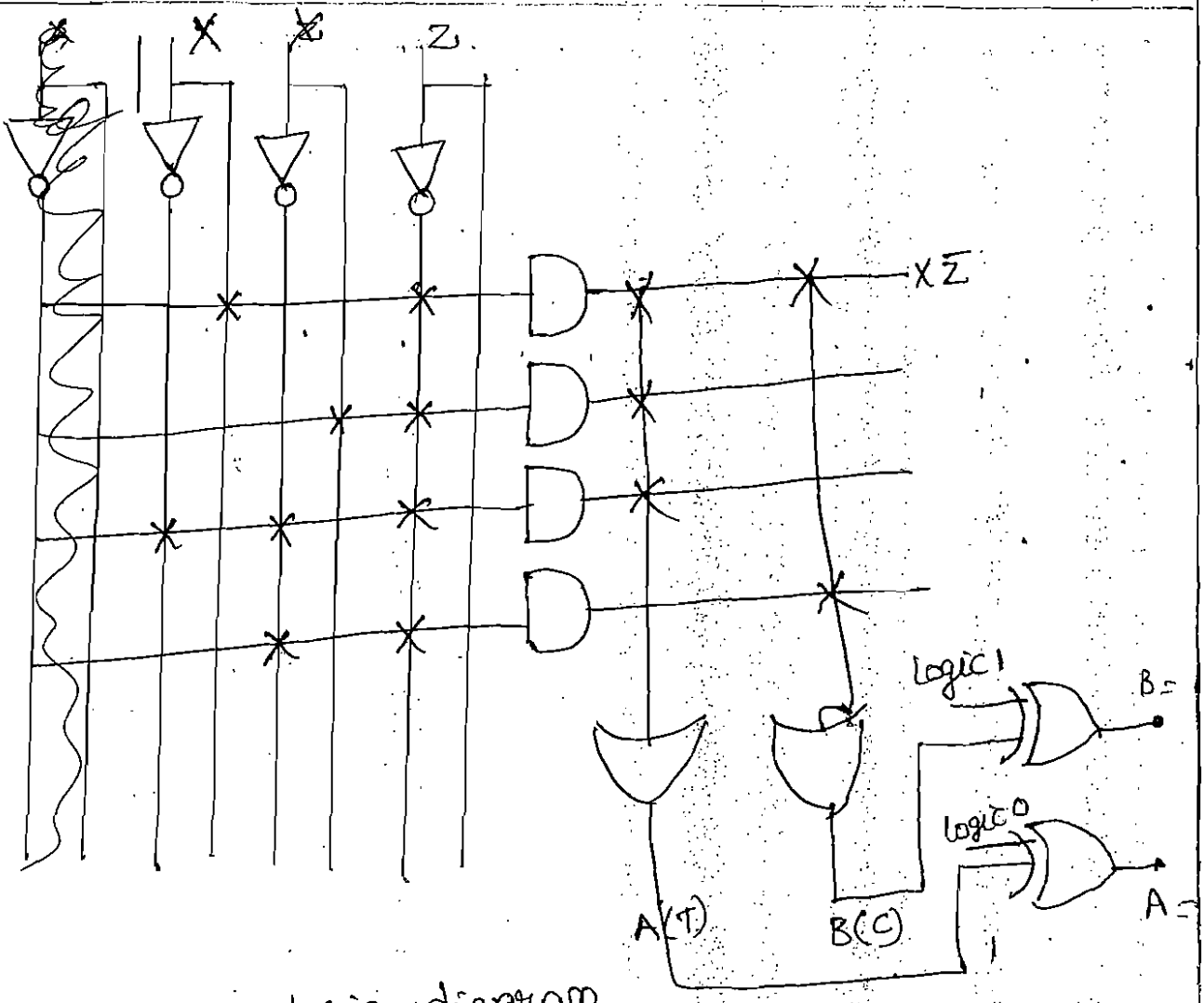
$= \bar{Y}\bar{Z} + X\bar{Z}$

$A(T) = X\bar{Z} + Y\bar{Z} + \bar{X}\bar{Y}Z$, $B(T) = \bar{X}Y + Z$

$A(C) = XZ + YZ + \overline{XYZ}$ $B(C) = \bar{Y}\bar{Z} + X\bar{Z}$

Step 2 :- programming table.

product term	inputs X	Y	Z	outputs A(T)	A(C)	B(T)	B(C)
$X\bar{Z}$	1	–	0	1	–	–	1
$Y\bar{Z}$	–	1	0	1	–	–	–
$\bar{X}\bar{Y}Z$	0	0	1	1	–	–	–
XZ	1	–	1	–	1	–	–
YZ	–	1	1	–	1	–	–
$\overline{XYZ}$	0	0	0	–	1	–	–
$\bar{X}Y$	0	1	–	–	–	1	–
Z	–	–	1	–	–	1	1
$\bar{Y}\bar{Z}$	–	0	0	–	–	–	1
$X\bar{Z}$	1	–	0	–	–	–	1



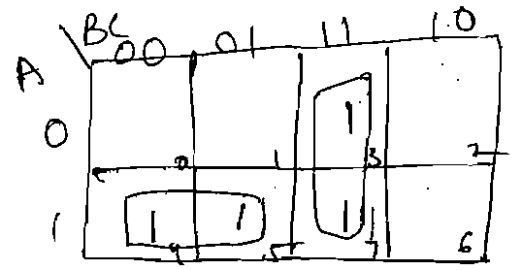
Logic diagram.

→ Implement the following boolean functions F_1 & F_2 of a combinational logic circuit using PLA.

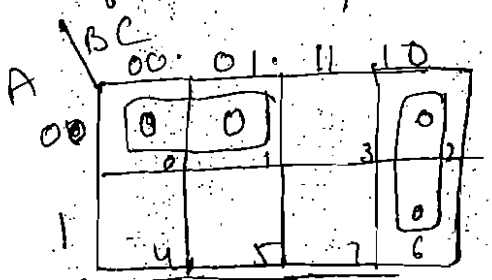
$F_1(A,B,C) = \Sigma m(3,4,5,7)$

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

Step 1:- K-map for F_1 (True form) Complement form



$F_1 = A\bar{B} + BC$



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

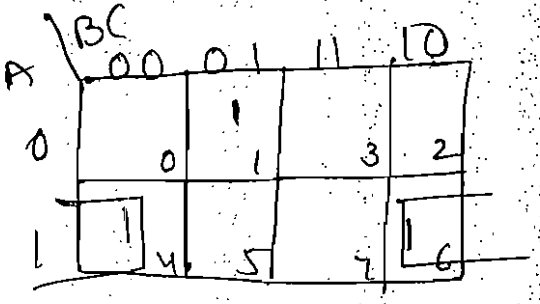
$= \overline{(A+B)} + \overline{(\bar{B}+C)}$

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

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

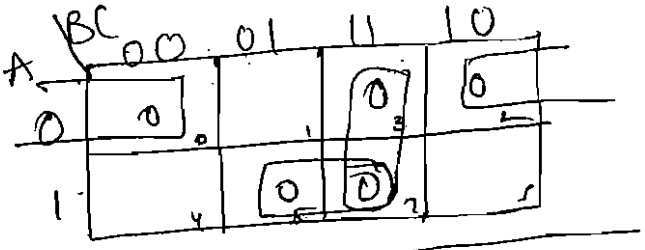
$= \bar{A}.\bar{B} + B.\bar{C}$

K-map for F_2 (true form)



$F_2 = \bar{A}\bar{B}C + A\bar{C}$

K-map for F_2 (complement form)



$\bar{F}_2 = (\bar{A}+C).(\bar{B}+\bar{C})(\bar{A}+\bar{C})$

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

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

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

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

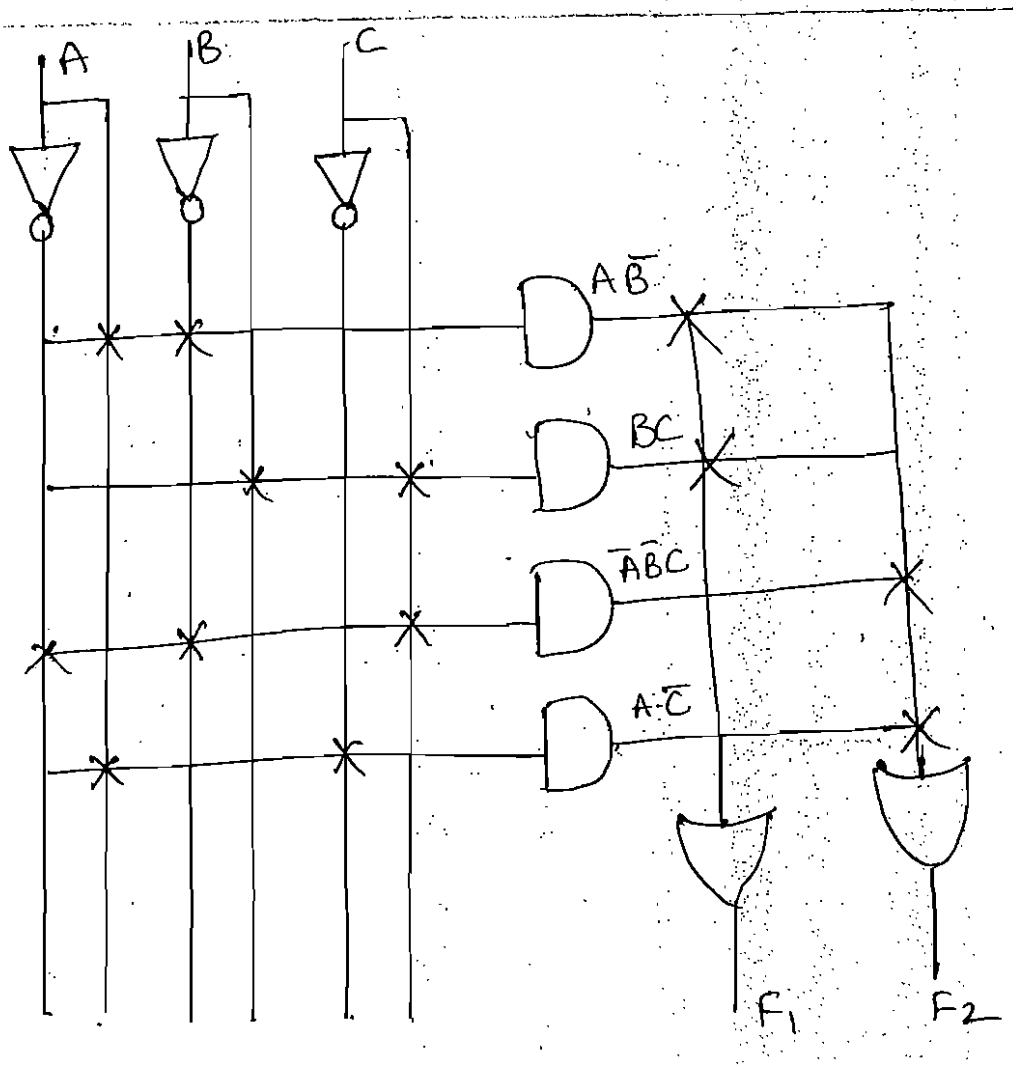
$F_1(C) = \bar{A}\bar{B} + B\bar{C}$

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

When we take $F_1(T)$ & $F_2(T)$ get 4 product terms and also having same number of product terms while taking $F_1(T)$ & $F_2(C)$, $F_1(C)$ & $F_2(T)$. So we have to consider $F_1(T)$ & $F_2(T)$.

Step 2 :- programming table

product terms	inputs A	B	C	outputs $F_1(T)$	$F_2(C)$
$A\bar{B}$	1	0	—	1	—
BC	—	1	1	1	1
$\bar{A}\bar{B}\bar{C}$	0	—	0	1	1
$A\bar{C}$	1	—	1	—	1



logic diagram.

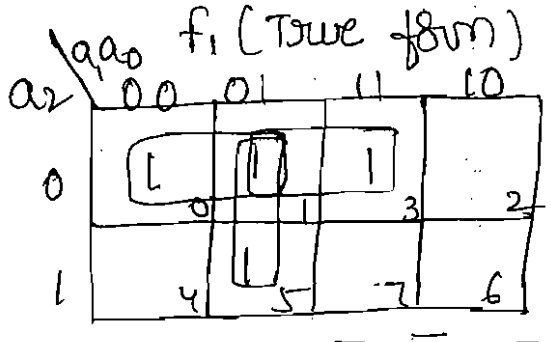
→ Implement the following multi boolean function using (3X4X2) PLA.

$F_1(a_2, a_1, a_0) = \Sigma m(0,1,3,5)$

$F_2(a_2, a_1, a_0) = \Sigma m(3,5,7)$

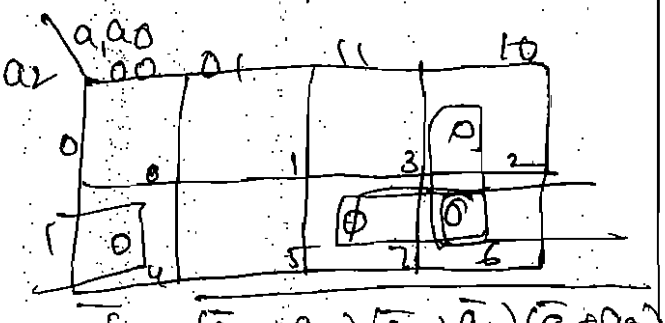
step 1:- K-maps.

f_1 (True form)



$f_1(T) = \bar{a}_1a_0 + \bar{a}_2\bar{a}_1 + \bar{a}_2a_0$

f_1 (complement form)

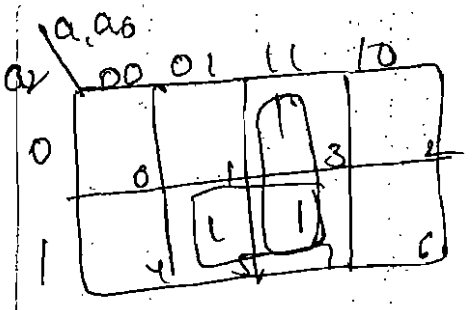


$\bar{f}_1 = (\bar{a}_2 + a_0)(\bar{a}_2 + \bar{a}_1)(\bar{a}_1 + a_0)$

$f_1(C) = \overline{\overline{a}_2} \cdot a_0 + \overline{\overline{a}_2} \cdot \overline{\overline{a}_1} + \overline{\overline{a}_1} \cdot \overline{a}_0$

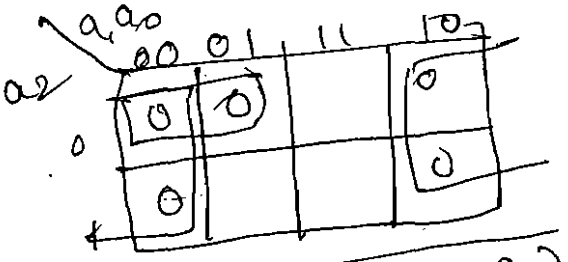
$= a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

f_2(True form)



$f_2(T) = a_2 a_0 + a_1 a_0$

F_2 (complement form..



$\overline{F}_2 = (\overline{a}_0) \cdot (a_2 + a_1)$

$F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

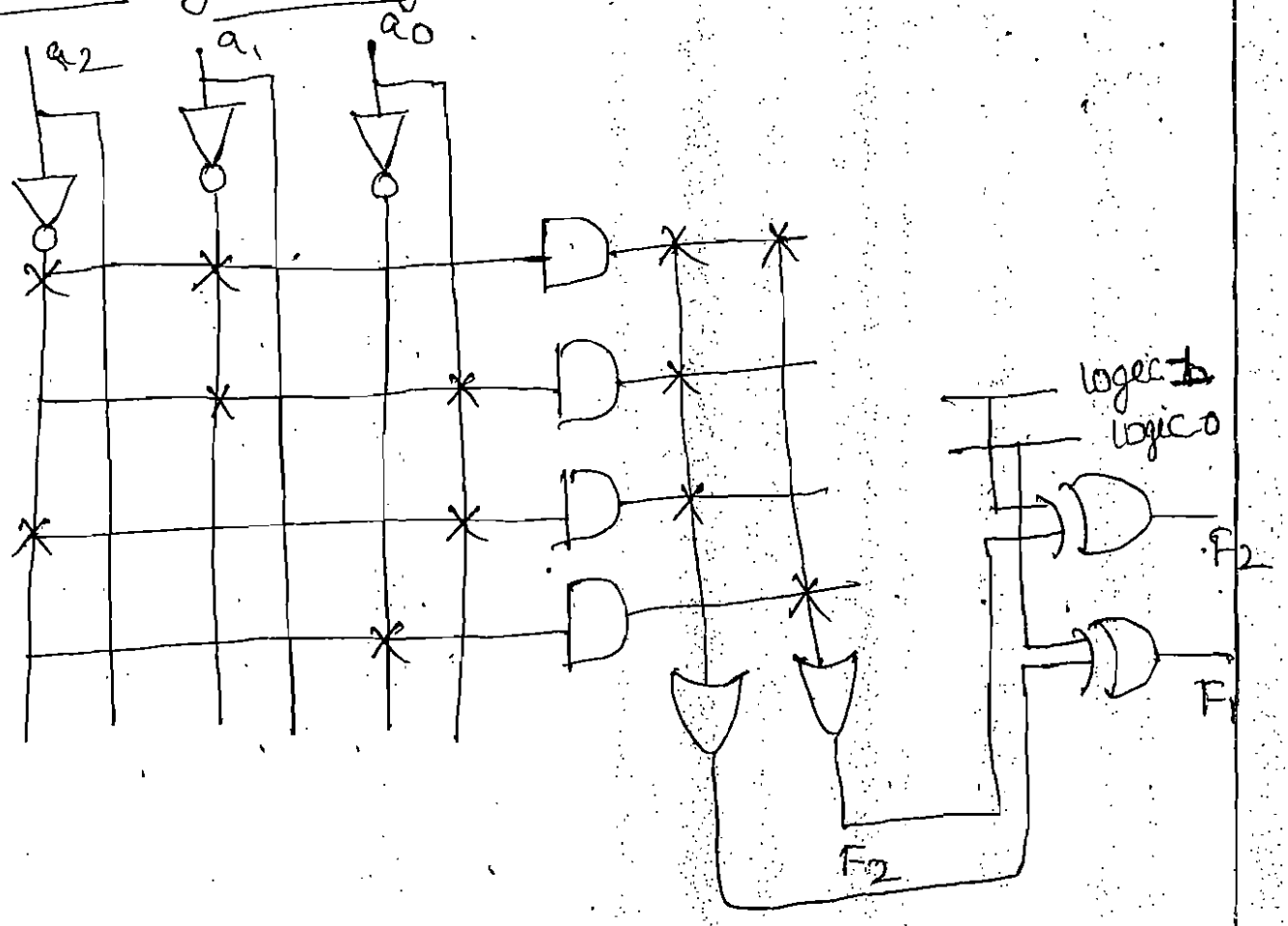
$F_1(T) = \overline{a}_1 a_0 + \overline{a}_2 \overline{a}_1 + \overline{a}_2 a_0$ $F_1(C) = a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

$F_2(T) = a_2 a_0 + a_1 a_0$ $F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

step 2:- PLA programming table.

product terms	Inputs a_2	a_1	a_0	outputs $F_1(T)$	$F_2(C)$
$\overline{a}_1 a_0$	—	0	1	1	—
$\overline{a}_2 \overline{a}_1$	0	0	—	1	1
$\overline{a}_2 a_0$	0	—	1	1	—
$\overline{a}_0$	—	—	0	—	1

Step 3 :- logic diagram :-



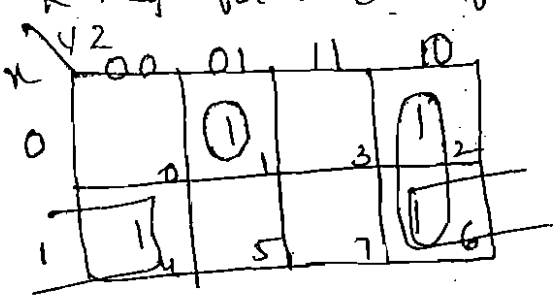
→ Implement the following using PLA

$A(x,y,z) = \Sigma m(1,2,4,6)$, $B(x,y,z) = \Sigma m(0,1,6,7)$

$C(x,y,z) = \Sigma m(2,6)$

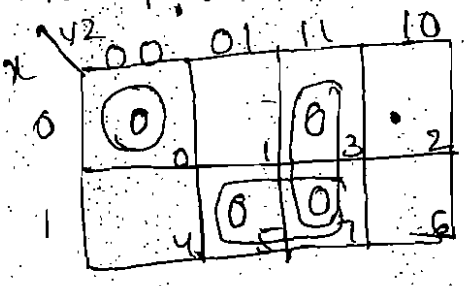
Step 1 :- K-map.

K-map for A (True form).



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

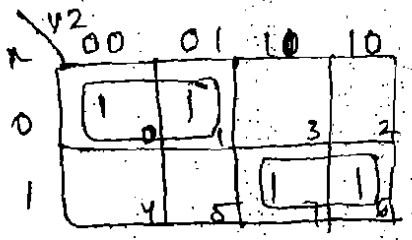
K-map for A (complement form).



$A(C) = \overline{(x+y+z)\cdot(\bar{y}+\bar{z})(\bar{x}+\bar{z})}$

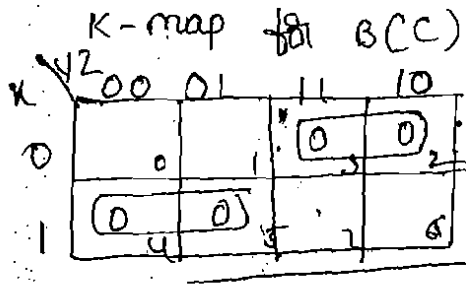
$= \bar{x}\bar{y}\bar{z} + yz + xz.$

K-map for B(T)



$B(T) = \bar{x}y + x\bar{y}$.

K-map for B(C)

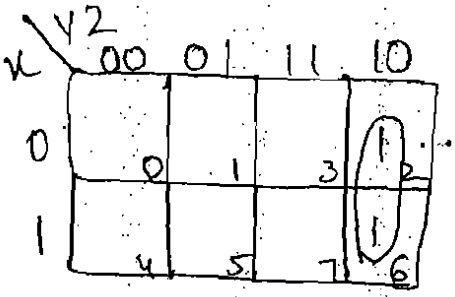


$B(C) = \overline{(\bar{x}+y)(x+\bar{y})}$

$= \bar{x}\cdot\bar{y} + \bar{x}\bar{\bar{y}}$

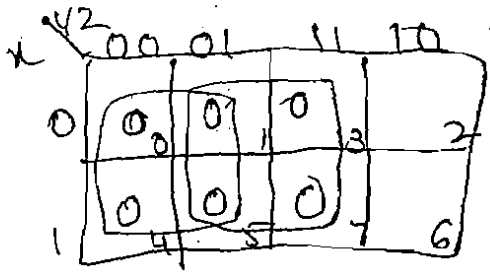
$= x\bar{y} + \bar{x}y$.

K-map for C(T)



$C(T) = y\bar{z}$

K-map for C(C)



$C(C) = \overline{\bar{y}\cdot\bar{z}}$

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

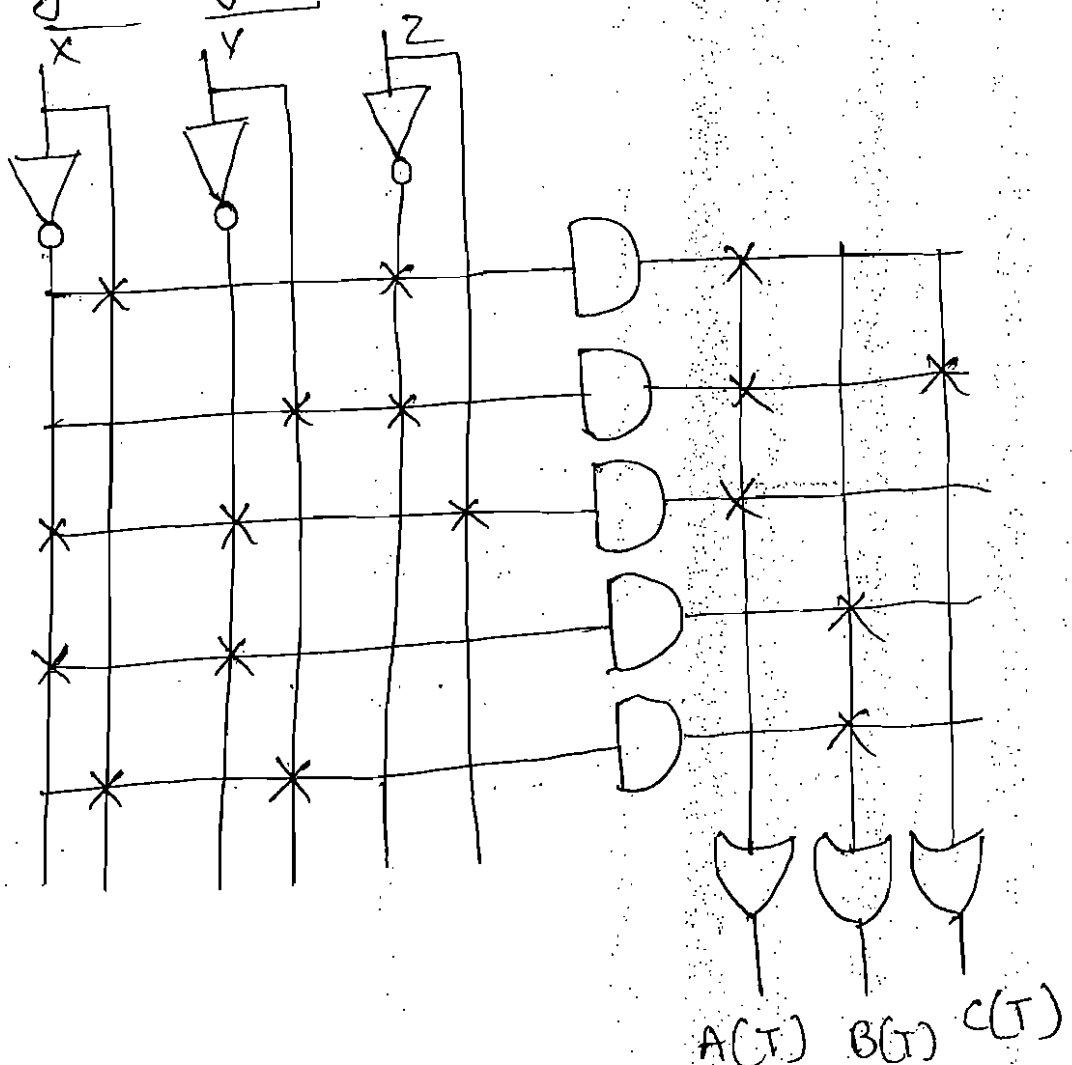
$= \bar{y} + z$.

Step 2 :- programming table.

product term	inputs x	y	z.	outputs A(T)	B(T)	C(T)
$x\bar{z}$	1	–	0	1	–	–
$y\bar{z}$	–	1	0	1	–	1
$\bar{x}\bar{y}z$	0	0	1	1	–	–
$\bar{x}\bar{y}$	0	0	x	–	1	–
xy	1	1	–	–	1	–

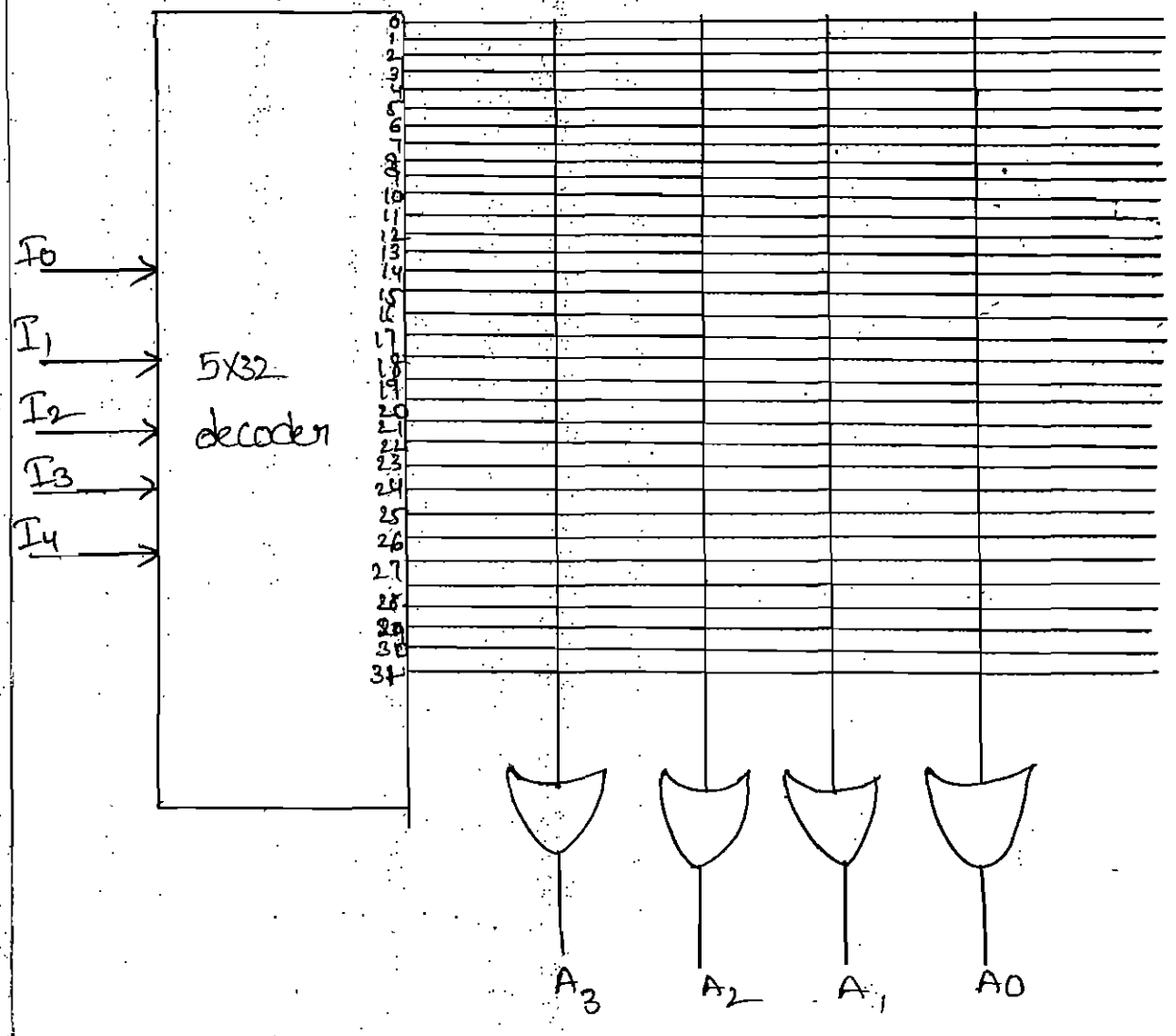
Step - 3

Logic diagram :-



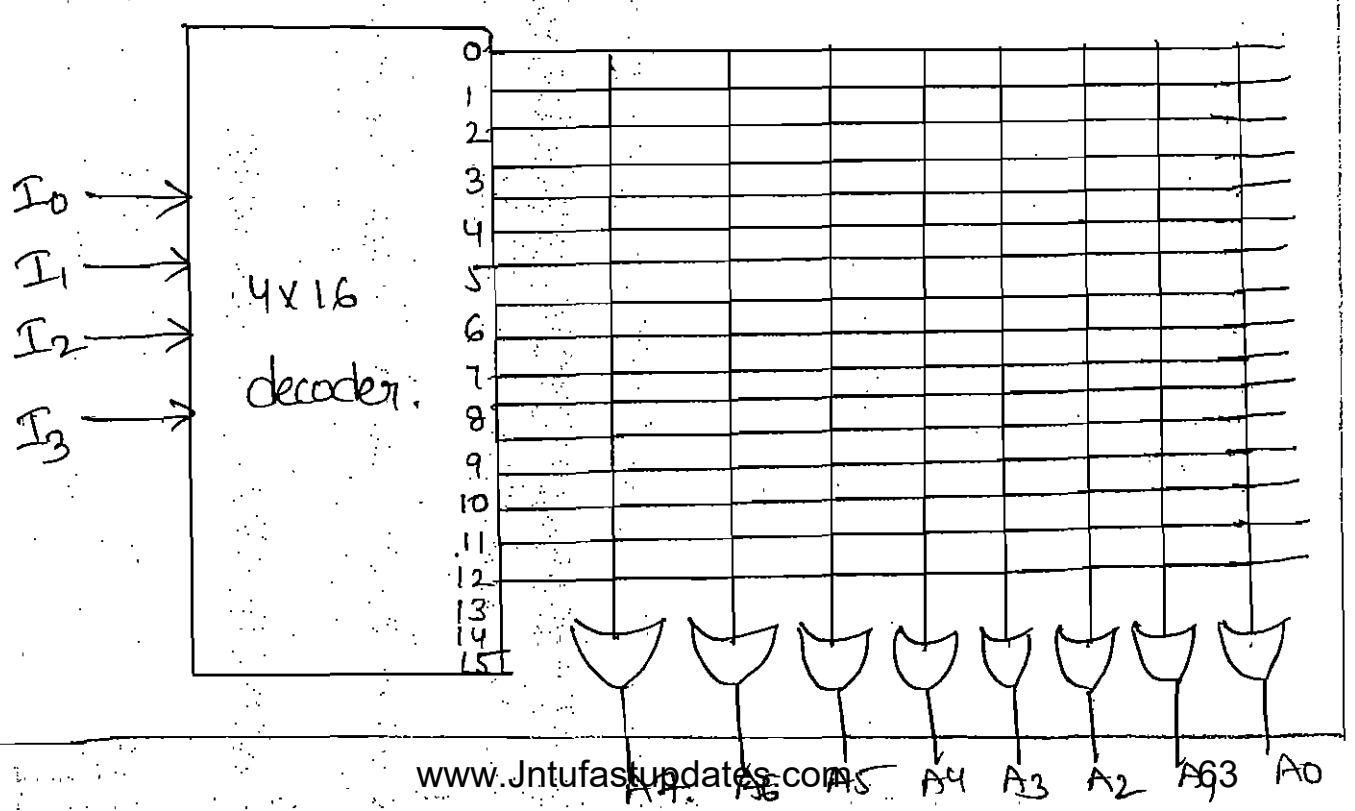
1) Give the logic implementation of a 32x4 bit ROM using a decoder of a suitable size.

A 32x4 bit ROM is to be implemented. it consists of 32 words of four bits each. There must be five input lines that form the binary numbers from 0 through 31 for the address. The five inputs are decoded into 32 distinct outputs by means of a 5x32 decoder. Each output of the decoder represent a memory address. The 32 outputs of the decoder are connected to each of the four OR gates.



32 X 4 bit ROM.

→ 16 X 8 ROM.



Comparison between PROM, PLA and PAL.

PROM	PLA	PAL
1. AND array is fixed and OR array is programmable.	1. Both AND and OR arrays are programmable.	1. OR array is fixed and AND array is programmable.
2. cheaper and simple to use	2. costliest and more complex than PAL and PROM.	2. Cheaper and simpler.
3. ALL minterms are decoded.	3. AND array can be programmed to get desired minterms.	3. AND array can be programmed to get desired minterms.
4. only Boolean functions in standard sop form can be implemented using PROM.	4. Any Boolean function in sop form can be implemented using PROM.	4. Any Boolean function in SOP form can be implemented using PAL.

→ Implement a Binary to BCD code converter by using PAL

→ Iam taking 3-bit Binary P,

Binary B_3 B_2 B_1	BCD code C_4 C_3 C_2 C_1
0 0 0	0 0 0 0
0 0 1	0 0 0 1
0 1 0	0 0 1 0
0 1 1	0 0 1 1
1 0 0	0 1 0 0
1 0 1	0 1 0 1
1 1 0	0 1 1 0
1 1 1	0 1 1 1

→ Ram taking 4-binary.

B_4 B_3 B_2 B_1	C_8 C_7 C_6 C_5	C_4 C_3 C_2 X_1
0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 0	0 0 0 1
0 0 1 0	0 0 0 0	0 0 1 0
0 0 1 1	0 0 0 0	0 0 1 1
0 1 0 0	0 0 0 0	0 1 0 0
0 1 0 1	0 0 0 0	0 1 0 1
0 1 1 0	0 0 0 0	0 1 1 0
0 1 1 1	0 0 0 0	0 1 1 1
1 0 0 0	0 0 0 0	1 0 0 0
1 0 0 1	0 0 0 0	1 0 0 1
1 0 1 0	0 0 0 1	0 0 0 0
1 0 1 1	0 0 0 1	0 0 0 1
1 1 0 0	0 0 0 1	0 0 1 0
1 1 0 1	0 0 0 1	0 0 1 1
1 1 1 0	0 0 0 1	0 1 0 0
1 1 1 1	0 0 0 1	0 1 0 1

$C_5 = Em(10,11,12,13,14,15)$.

$C_4 = Em(8,9)$

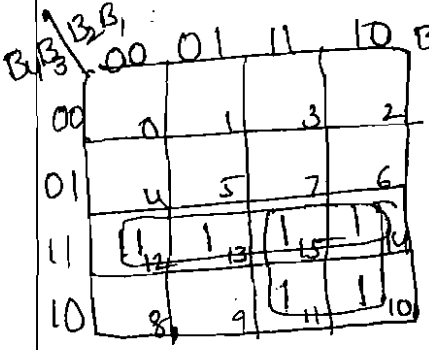
$C_3 = Em(4,5,6,7,14,15)$

$C_2 = Em(2,3,6,7,12,13)$

$C_1 = Em(1,3,5,7,9,11,13,15)$.

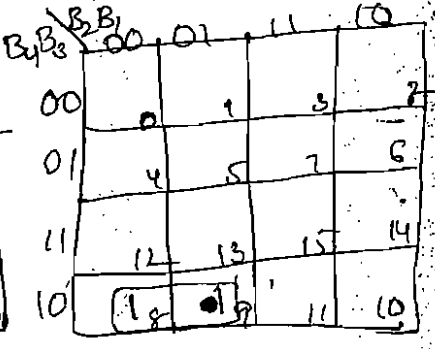
Step 1:- K-map.

K-map for C_5.



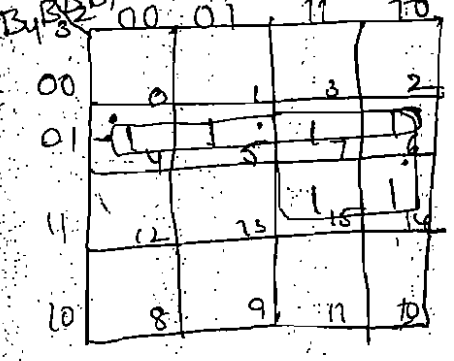
$B_4B_3 + B_4B_2$.

K-map for C_4



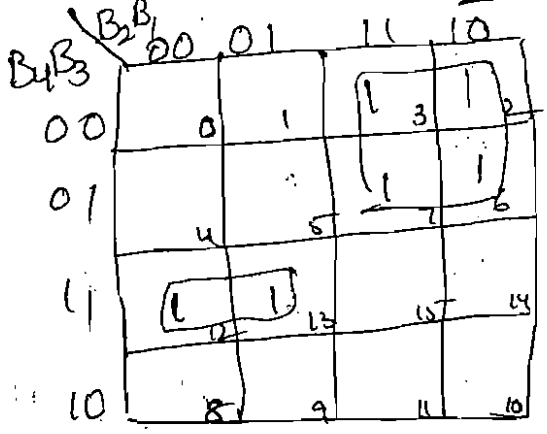
$B_4\overline{B_3}\,\overline{B_2}$

K-map for C_3



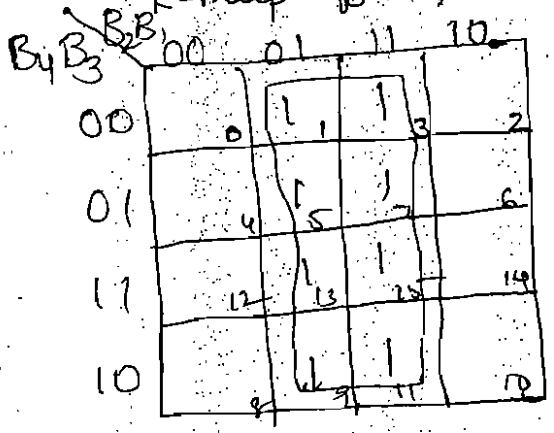
$\overline{B_4}B_3 + B_3B_2$

K-map for C_2



$B_4B_3\overline{B_2} + \overline{B_4}B_2$

K-map for C_1

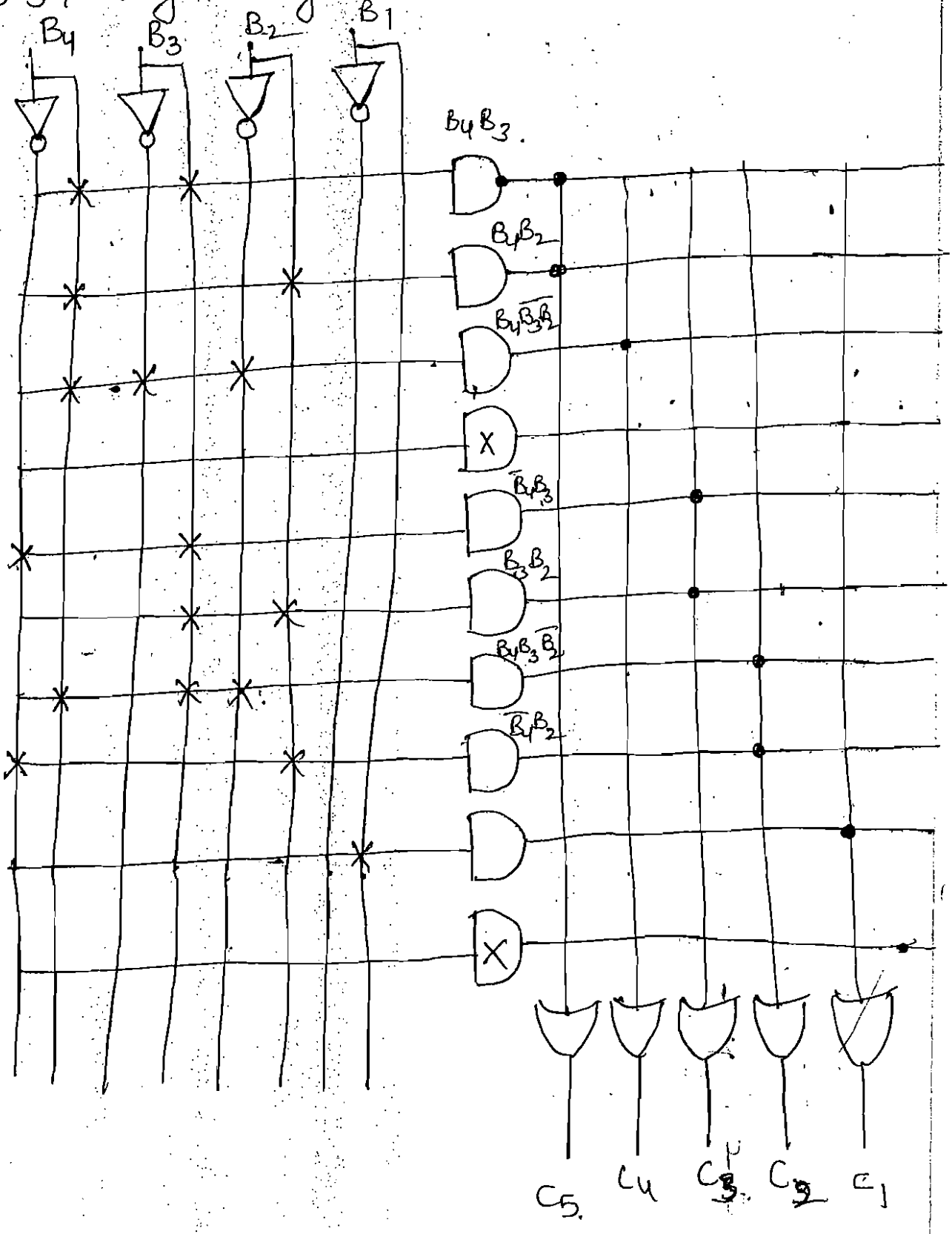


B_1

step 2:- programming table

product terms	inputs B_4 B_3 B_2 B_1	output
1	1 1 – –	$C_5 = B_4B_3 + B_4B_2$
2	1 – 1 –	
3	1 0 0 –	$C_4 = B_4\overline{B_3}\,\overline{B_2}$
4	– – – –	
5	0 1 – –	$C_3 = \overline{B_4}B_3 + B_3B_2$
6	– 1 1 –	
7	1 1 0 –	$C_2 = B_4B_3\overline{B_2} + \overline{B_4}B_2$
8	0 – 1 –	
9	– – – 1	$C_1 = B_1$
10	– – – –	

Step 3 :- logic diagram.



UNIT 5

Sequential Circuits

Combinational Circuit and Sequential Circuits:

Logic Circuits can be divided into :

1. Combinational Logic Circuit
2. Sequential Logic Circuit

Combinational Logic Circuit :

Combinational Logic circuit contains logic gates where its output is determined by the combination of the current input, regardless of the output or the prior combination of input. Basically, combinational circuit can be depicted by FIG 3.1 below:

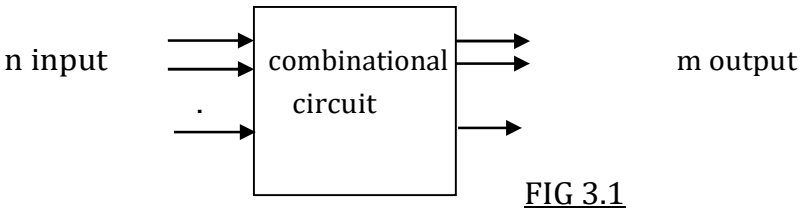


FIG 3.1

Examples of Combinational circuits in the computer system are decoder, parallel adder, and multiplexer.

Sequential Logic Circuit :

Sequential Logic Circuit contains logic gates arranged in parallel and its output is not only determined by the combination of the current input, but also the prior output. The circuit also contains memory elements that enable it to store the information of the prior output. Generally, sequential circuits can be depicted by FIG 3.2 below:

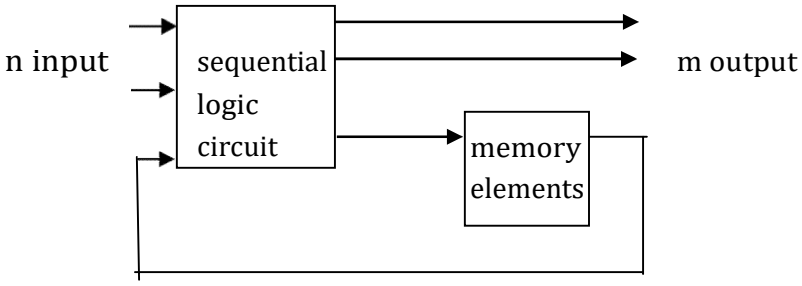


FIG 3.2

Examples of sequential circuits are like FLIP FLOPS,registers, counters and serial adders

- Unlike combinational logic circuits, the output of a *sequential* logic circuit depends not only on the current inputs but also on the current state of *memory elements* in the circuit.
- The binary information stored in the memory elements determines the *state* of the circuit at any given time.
- A typical sequential circuit consists of some combinational logic circuitry combined with memory elements to store the state of the circuit.
- The current state of the circuit is fed back to the combinational logic and can be considered as additional inputs to the circuit. This is called *feedback*.

Sequential circuits can be divided into

1. Synchronous
2. Asynchronous

In synchronous sequential circuit, all flip-flops are moved by the same clock pulse so that all flip-flops involved change simultaneously.
In asynchronous circuit, the change of flip-flop condition depends on the change that occurs on the input and the late time that is in the circuit. The memory elements in clocked sequential circuits are called *flip-flops*.
difference between a latch and a flip-flop. Now, let's make the answer easy to understand by tabulating some simple & important differences between latch and Flip-Flop.

Flip-Flop	Latch
The state of circuit changes only when the control signal goes from high to low or low to high (i.e. sensitive to signal change only).	Changes state as soon as the input signals change (of course there is some propagation delay).
Synchronous	Asynchronous
Edge triggered which means that the output of the circuit changes when there is a change in clock pulse (it may be a positive or negative edge of the clock pulse).	Level triggered which means that the output of the circuit depends on the level of the enable signal (either 1 or 0).
Built from latches (a FF is like a clocked latch). FF is used as a register.	Built from logic gates.
Works on clock pulses	Works on enable function input
Has a clock signal	Does not have a clock signal
Edge sensitive i.e. content of FF change only either at the rising or falling edge of the enable signal (usually the controlling clock signal). After the rising or falling edge of the clock signal, the content of FF remains constant even if the input changes.	Level sensitive i.e. content of latch changes immediately when there is a change in its inputs. Output is sensitive for the duration of enable signal active pulse and thus, can transmit or receive data during the active pulse only.
Sampling of the inputs is done only at a clock event for example, rising edge, falling edge.	Sampling of the inputs is done continuously only when the enable signal is on.

Flip-Flops

1. Introduction

The circuits stored information about the previous history of inputs are called storage or memory elements. A primitive storage element can be constructed from a small number of gates connecting the outputs back as inputs. These circuits are binary cells capable of storing one bit of information. They have two outputs, one for the normal value and one for the complement value of bit stored in it. Primitive memory elements actually fall into two board classes : *latches* and *flip-flop*.

If a latch has only data inputs, it is called an *unlocked latch* (or only latch). Level-sensitive latches have an additional enable input, sometimes called the *clock*. Level-sensitive latches continuously sample their inputs when they are enabled.

Any change in the level of the input is propagated through to the output. When the enable signal is unasserted, the last value of the inputs is determines the state held by the latch.

Flip-flops differ from latches in that their output change only with repeat to the clock, whereas latches change output when their inputs change. Flip-flops are characterized on the basis of the clock transition that cause the output change : there are *positive edge-triggered*, *negative edge-triggered*, and *master/slave* flip-flops.

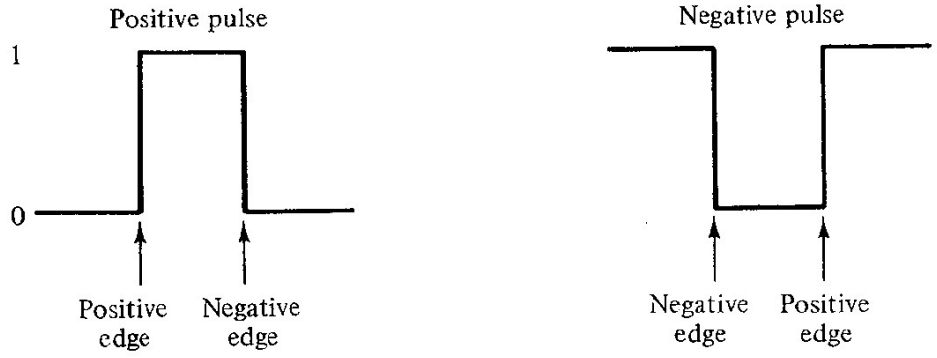
A positive edge-triggered flip-flop samples its inputs on the low-to-high clock transition. A negative edge-triggered flip-flop works in a similar fashion, with the input sampled on the high-to-low clock transition.

A master-slave flip-flop is constructed from two stage separate flip-flops. The first stage (first flip-flop) samples the inputs on the rising edge of a clock signal. The second stage transfer them to the output on the falling edge of the clock signal.

These circuits have two additional control inputs. These are Preset and Clear, which force the output of the flip-flop or latch to the logic-1 or logic-0 state, respectively, independent of the flip-flop or latch inputs .

Edge-Triggered Flip-Flop

- Edge triggered flip-flops respond only to a clock pulse edge and not to the level of the of the clock pulse.
- Edge-triggered flip-flops can be either positive or negative edge sensitive.

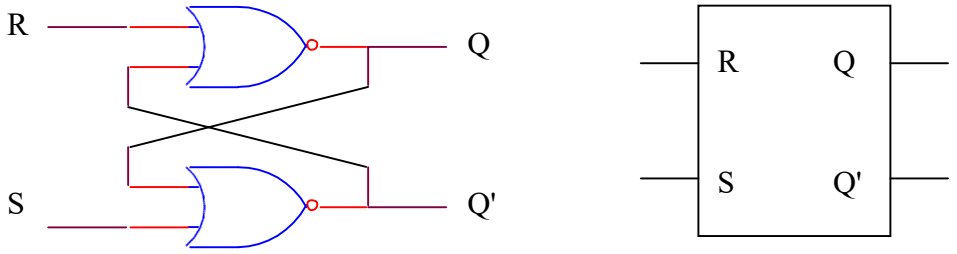


Definition of clock-pulse transition

S-R Latch:

- A S-R (Set-Reset) latch is the simplest possible memory element.
- It is constructed by feeding the outputs of two NOR gates back to the other NOR gates input.
- The inputs R and S are referred to as the Reset and Set inputs, respectively.
- To understand the operation of the S-R latch consider the following scenarios :
  - **S=1 and R=0:** The output of the bottom NOR gate is equal to zero, $Q' = 0$.
  - Hence both inputs to the top NOR gate are equal to zero, thus, $Q = 1$.

- Hence, the input combination S=1 and R=0 leads to the latch being set to $Q = 1$.
- **S=0 and R=1:** Similar to the arguments above, the outputs become $Q' = 1$ and $Q = 0$.
- We say that the latch is reset.
- **S=0 and R=0:** Assume the latch is set ($Q' = 0$ and $Q = 1$), then the output of the top NOR gate remains at $Q = 1$ and the bottom NOR gate stays at $Q' = 0$.
- Similary, when the latch is in a reset state ($Q' = 1$ and $Q = 0$), it will remain there with this input combination.
- Therefore, with inputs S=0 and R=0, the latch remains in its state.
- **S=1 and R=1:** This input combination must be avoided
- The logic diagram and graphic symbol are shown in Figure.3.3.
- The following truth table can be summarized the operation of the S-R latch.



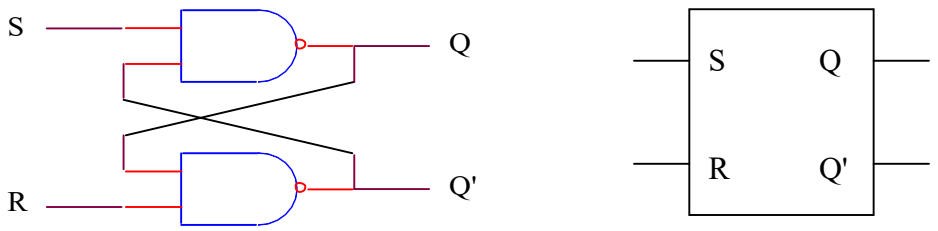
(a) Logic Diagram (b)Graphic Symbol

S	R	Q	Q'	Comment
0	0	Q	Q'	Hold State
0	1	0	1	Reset
1	0	1	0	Set
1	1	-	-	Forbidden

(c) Truth table

Figure.3.3 S-R latch with NOR gates.

- A S-R latch can also be constructed from NAND gates. The graphic symbol, logic diagram, and truth table of the latch are shown in Figure.3.4.



(a) Logic Diagram (b)Graphic Symbol

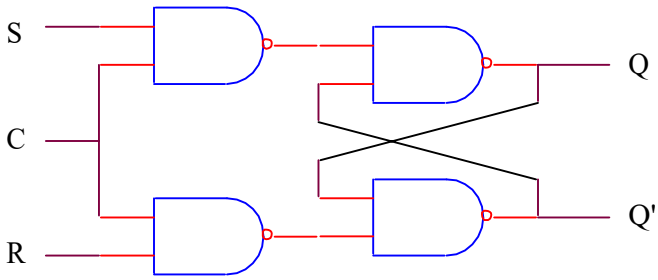
S	R	Q	Q'	Comment
1	1	Q	Q'	Hold State
0	1	1	0	Set
1	0	0	1	Reset
0	0	-	-	Forbidden

(c) Truth table

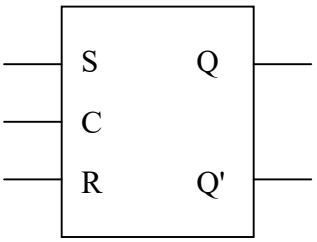
Figure.3.4 S-R latch with NAND gates.

Level Sensitive (Clock) S-R Latch:

The operation of the S-R latch can be modified by providing an additional control input that determines when the state of the circuit is to be changed. The logic diagram, graphic symbol, and thruth table of level sensitive S-R latch are shown in Figure.3.5 .



(a) Logic Diagram



(b)Graphic Symbol

S	R	C	Q	Q'	Comment
0	0	1	Q	Q'	Hold State
0	1	1	0	1	Reset
1	0	1	1	0	Set
1	1	1	-	-	Forbidden
x	x	0	Q	Q'	Hold State

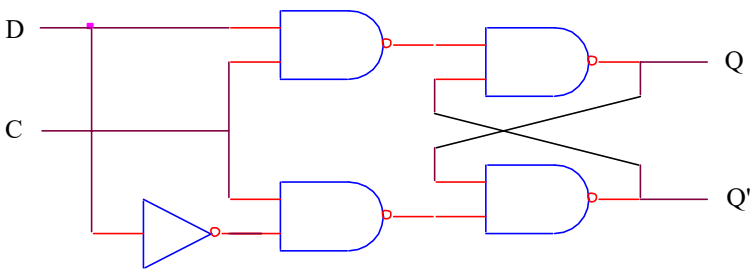
(c) Truth table

Figure.3.5 Level Sensitive S-R latch with NAND gates.

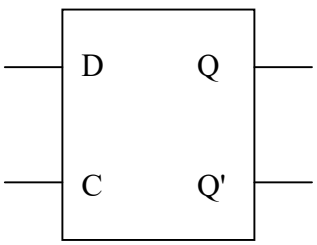
Level Sensitive (Clock) D (Delay) Latch :

One way to eliminate the undesirable condition of the indeterminate state in the S-R latch is to ensure that inputs S and R are never equal to 1 at the state time. This is done level sensitive D latch shown in Figure.3.6. The latch has only two inputs: D and C. The D input connect directly to the S input and its complement is applied to the R input. The D input is sampled when C is equal to 1. If D is equal to 1, the Q output goes to 1. If D is equal to 0, the Q output goes to 0. If C is equal to 0, the Q output remains in its previous state .

(a) Logic Diagram



(b)Graphic Symbol



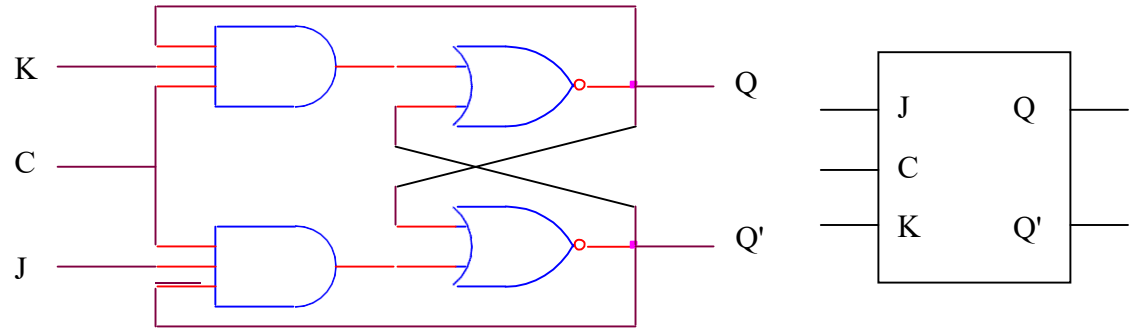
C	D	Q	Q'
1	0	0	1
1	1	1	0
0	x	Q	Q'

(c) Truth table

Figure.3.6 Level Sensitive D latch with NAND gates.

Level Sensitive (Clock) J-K Latch:

A level sensitive J-K latch shown in Figure.3.7 is a refinement of the S-R latch in that the indeterminate state of the S-R type is defined in the J-K type. Inputs J and K behave like inputs S and R to set and clear the latch, respectively. The input marked J is for set and the input marked K is for reset. When the both inputs J and K are equal to 1, the latch switches to its complement state, that is , if Q=1, it switches to Q=0, and vice versa. If the C is equal to 0, The output of the latch remains in its previous state .



(a) Logic Diagram (b)Graphic Symbol

C	J	K	Q	Q'	Comment
1	0	0	Q	Q'	Hold
1	0	1	0	1	Reset
1	1	0	1	0'	Set
1	1	1	Q'	Q	Toggle
0	x	x	Q	Q'	Hold

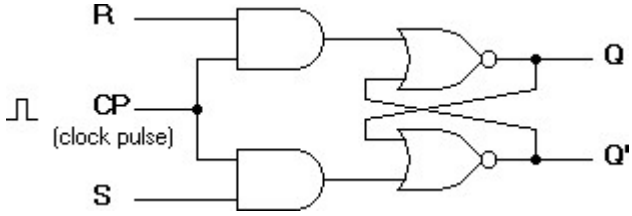
(c) Truth table

Figure.3.7 Level Sensitive J-K latch

S-R Flip-flop

The clocked SR flip-flop shown in Figure.3.8 consists of a basic NOR flip-flop and two AND gates. The outputs of the two AND gates remain at 0 as long as the clock pulse (or CP) is 0, regardless of the S and R input values. When the clock pulse goes to 1, information from the S and R inputs passes through to the basic flip-flop. With both S=1 and R=1, the occurrence of a clock pulse causes both outputs to momentarily go to 0. When the pulse is removed, the state of the flip-flop is indeterminate, ie., either state may result, depending on whether the set or reset input of the flip-flop remains a 1 longer than the transition to 0 at the end of the pulse..

1. If the value of both S and R are 0, the flip-flop will remain in its present condition (either 0 or 1).
2. If S = 0 and R = 1 *(reset),* then the flip-flop condition will change to 0 (its output, Q = 0).
3. If S = 1 (set) and R = 0, then the flip-flop condition will change to 1 (output, Q = 1).
4. This circuit does not allow combinational input of input S = 1 and R = 1



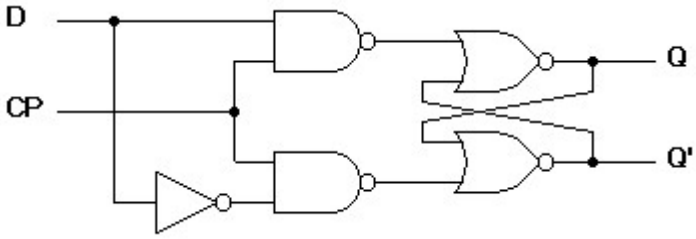
(a) Logic diagram

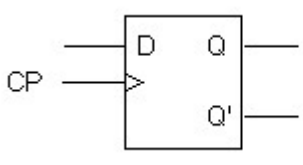
Q S R	Q(t+1)
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	indeterminate
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	indeterminate

Figure.3.8 SR flip-flop

D Flip-Flop:

D flip flop is actually a slight modification of the above explained clocked SR flip-flop. From the figure you can see that the D input is connected to the S input and the complement of the D input is connected to the R input. The D input is passed on to the flip flop when the value of CP is '1'. When CP is HIGH, the flip flop moves to the SET state. If it is '0', the flip flop switches to the CLEAR state.





Q D	Q(t+1)
0 0	0
0 1	1
1 0	0
1 1	1

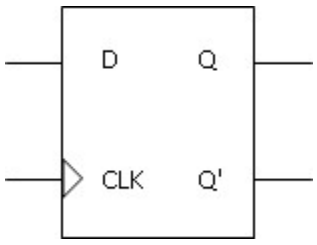
(a) Logic diagram with NAND gates **(b)** Graphical symbol **(c)** Transition table

Figure.3.9. Clocked D flip-flop

Positive-Edge Triggered:

CLK	D	Q	Q'
↑	0	0	1
↑	1	1	0
0	x	Q	Q'
1	x	Q	Q'

a) Truth table

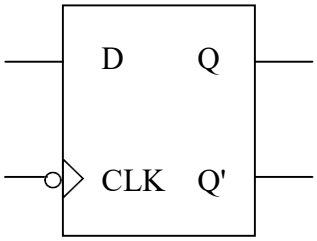


(b) Graphic Symbol

Figure.3.10. Positive edge-triggered D flip-flop.

Negative-Edge Triggered:

CLK	D	Q	Q'
↓	0	0	1
↓	1	1	0
0	x	Q	Q'
1	x	Q	Q'



(a) Truth table (b) Graphic Symbol

Figure.3.11. Positive edge-triggered D flip-flop.

J-K Flip-Flop:

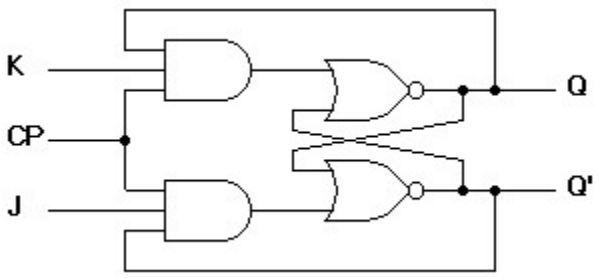
A J-K flip flop can also be defined as a modification of the S-R flip flop. The only difference is that the intermediate state is more refined and precise than that of a S-R flip flop.

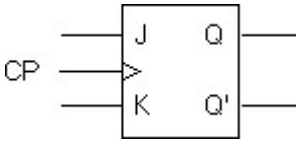
The behavior of inputs J and K is same as the S and R inputs of the S-R flip flop. The letter J stands for SET and the letter K stands for CLEAR.

When both the inputs J and K have a HIGH state, the flip-flop switch to the complement state. So, for a value of Q = 1, it switches to Q=0 and for a value of Q = 0, it switches to Q=1.

The circuit includes two 3-input AND gates. The output Q of the flip flop is returned back as a feedback to the input of the AND along with other inputs like K and clock pulse [CP]. So, if the value of CP is '1', the flip flop gets a CLEAR signal and with the condition that the value of Q was earlier 1. Similarly output Q' of the flip flop is given as a feedback to the input of the AND along with other inputs like J and clock pulse [CP]. So the output becomes SET when the value of CP is 1 only if the value of Q' was earlier 1.

The output may be repeated in transitions once they have been complimented for J=K=1 because of the feedback connection in the JK flip-flop. This can be avoided by setting a time duration lesser than the propagation delay through the flip-flop. The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction.





Q J K	Q(t+1)
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	0

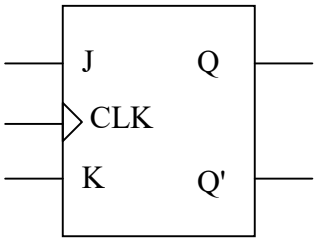
(a) Logic diagram (b) Graphical symbol (c) Transition table

Figure.3.12. Clocked JK flip-flop

Positive-Edge Triggered:

CLK	J	K	Q	Q'
↑	0	0	Q	Q'
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	Q'	Q
0	x	x	Q	Q'
1	x	x	Q	Q'

(a) Truth table



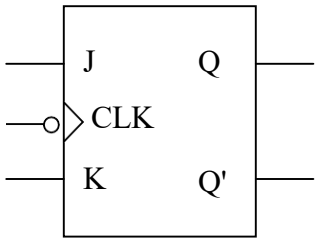
(b) Graphic Symbol

Figure.3.13. Positive edge-triggered J-K flip-flop.

Negative-Edge Triggered:

CLK	J	K	Q	Q'
↓	0	0	Q	Q'
↓	0	1	0	1
↑	1	0	1	0
↓	1	1	Q'	Q
0	x	x	Q	Q'
1	x	x	Q	Q'

(a) Truth table



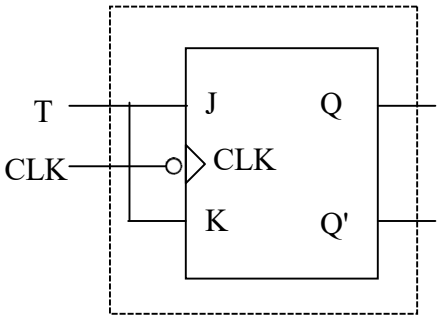
(b) Graphic Symbol

Figure.3.14. Negative edge-triggered J-K flip-flop.

T Flip-Flop:

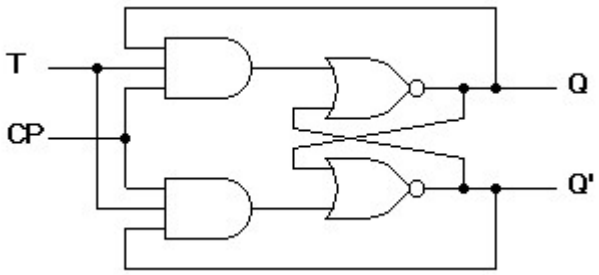
The T flip-flop is a single-input version of the J-K flip-flop. As shown in Figure.3.15, the T flip-flop is obtained from the J-k flip-flop when both inputs are tied together. The designation T comes from the ability of the flip-flop to *toggle*, or *complement,* its state. While input T is 1, The flip-flop complements its output when the clock pulse occurs. While T is 0, The output of the flip-flop remains in its previous state .

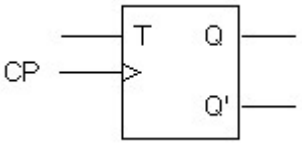
CLK	T	Q	Q'
↓	0	Q	Q'
↓	1	Q'	Q
0	x	Q	Q'
1	x	Q	Q'



(a) Truth table (b) Graphic Symbol

Figure.3.15. Negative Edge-Triggered T flip-flop.





Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(a) Logic diagram **(b)** Graphical symbol **(c)** Transition table

Figure.3.16. Clocked T flip-flop

Introduction - Triggering of Flip-flops

The state of a flip-flop is changed by a momentary change in the input signal. This change is called a trigger and the transition it causes is said to trigger the flip-flop. The basic circuits require an input trigger defined by a change in signal level. This level must be returned to its initial level before a second trigger is applied. Clocked flip-flops are triggered by pulses.

The feedback path between the combinational circuit and memory elements can produce instability if the outputs of the memory elements (flip-flops) are changing while the outputs of the combinational circuit that go to the flip-flop inputs are being sampled by the clock pulse. A way to solve the feedback timing problem is to make the flip-flop sensitive to the pulse transition rather than the pulse duration.

The clock pulse goes through two signal transitions: from 0 to 1 and the return from 1 to 0. As shown in Figure.3.17 the positive transition is defined as the positive edge and the negative transition as the negative edge.

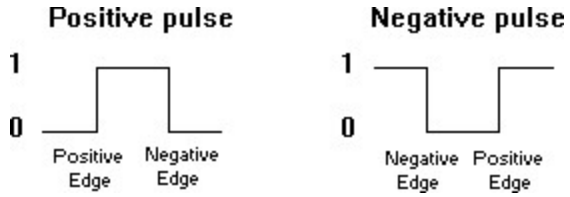


Figure.3.17. Definition of clock pulse transition

The clocked flip-flops already introduced are triggered during the positive edge of the pulse, and the state transition starts as soon as the pulse reaches the logic-1 level. If the

other inputs change while the clock is still 1, a new output state may occur. If the flip-flop is made to respond to the positive (or negative) edge transition only, instead of the entire pulse duration, then the multiple-transition problem can be eliminated.

Introduction - Master-Slave Flip-Flop:

A master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave. The logic diagram of an SR flip-flop is shown in Figure.3.18. The master flip-flop is enabled on the positive edge of the clock pulse CP and the slave flip-flop is disabled by the inverter. The information at the external R and S inputs is transmitted to the master flip-flop. When the pulse returns to 0, the master flip-flop is disabled and the slave flip-flop is enabled. The slave flip-flop then goes to the same state as the master flip-flop.

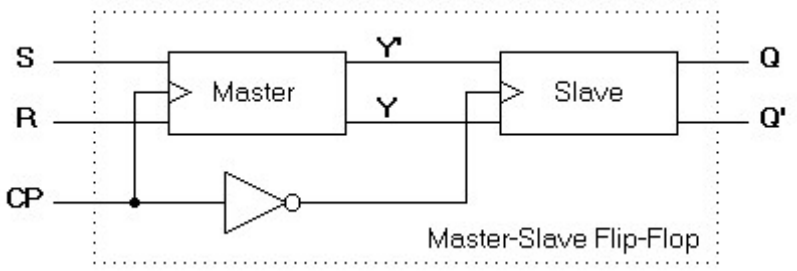


Figure.3.18. Logic diagram of a master-slave flip-flop

The timing relationship is shown in Figure 10 and is assumed that the flip-flop is in the clear state prior to the occurrence of the clock pulse. The output state of the master-slave flip-flop occurs on the negative transition of the clock pulse. Some master-slave flip-flops change output state on the positive transition of the clock pulse by having an additional inverter between the CP terminal and the input of the master.

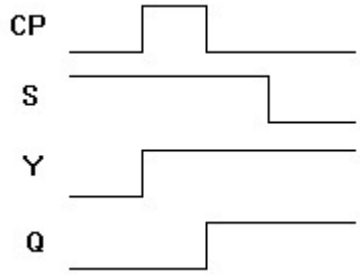


Figure.3.19. Timing relationship in a master slave flip-flop

The Master-Slave JK Flip-flop

The **Master-Slave Flip-Flop** is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and Q from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip flop being connected to the two inputs of the "Slave" flip flop. This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip flop as shown below.

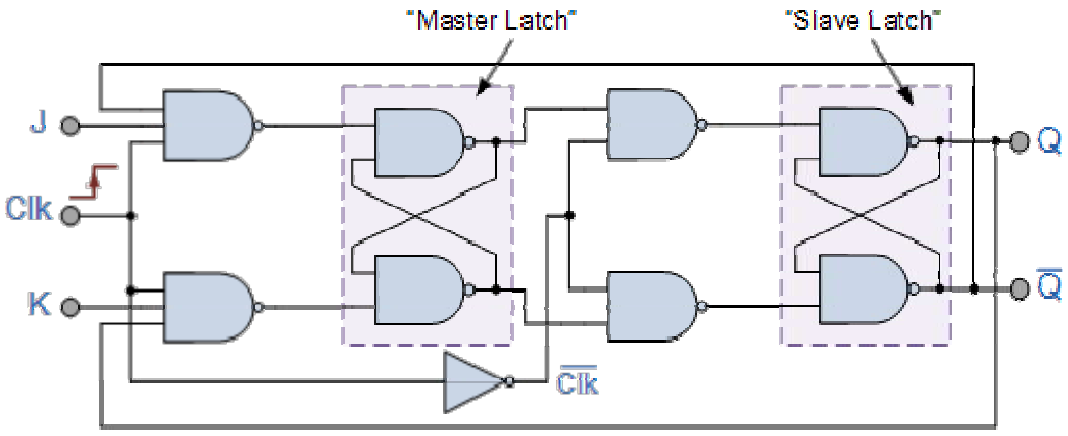


Figure.3.20

The input signals J and K are connected to the gated "master" SR flip flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip flop is the inverse (complement) of the "master" clock input, the "slave" SR flip flop does not toggle. The outputs from the "master" flip flop are only "seen" by the gated "slave" flip flop when the clock input goes "LOW" to logic level "0".

When the clock is "LOW", the outputs from the "master" flip flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip flop now responds to the state of its inputs passed over by the "master" section.

Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip flop are fed through to the gated inputs of the "slave" flip flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip flop edge or pulse-triggered.

Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal. In other words, the **Master-Slave JK Flip flop** is a "Synchronous" device as it only passes data with the timing of the clock signal.

Clock :

Latches are known as **level-sensitive** because their outputs are affected by their inputs as long as they are enabled. Their memory state can change during this entire time when the enable signal is asserted. In a computer circuit, however, we do not want the memory state to change at various times when the enable signal is asserted. Instead, we like to synchronize all of the state changes to happen at precisely the same moment and at regular intervals. In order to achieve this, two things are needed: (1) a synchronizing signal, and (2) a memory circuit that is not level-sensitive. The synchronizing signal, of course, is the **clock**, and the non-level-sensitive memory circuit is the flip-flop.

The clock is simply a very regular square wave signal, as shown in Figure 6.8. We call the edge of the clock signal when it changes from 0 to 1 the **rising edge**. Conversely, the **falling edge** of the clock is the edge when the signal changes from 1 to 0. We will use the symbol Ç to denote the rising edge and È for the falling edge.

In a computer circuit, either the rising edge or the falling edge of the clock can be used as the synchronizing signal for writing data into a memory element.

This edge signal is referred to as the **active edge** of the clock. In all of our examples, we will use the rising clock edge as the active edge. Therefore, at every rising edge, data will be clocked or stored into the memory element.

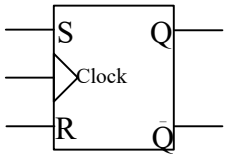
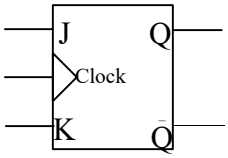
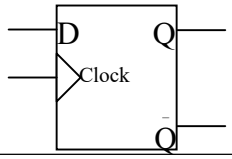
<table>
<tr><th>Name</th><th>Graphical Symbol</th><th>Feature Table</th></tr>
<tr><td>S-R</td><td> 
</td><td>
<table>
<tr><th>S</th><th>R</th><th>Q_{n+1}</th></tr>
<tr><td>0</td><td>0</td><td>Q_n</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>-</td></tr>
</table>
</td></tr>
<tr><td>J-K</td><td> 
</td><td>
<table>
<tr><th>J</th><th>K</th><th>Q_{n+1}</th></tr>
<tr><td>0</td><td>0</td><td>Q_n</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>Change condition</td></tr>
</table>
</td></tr>
<tr><td>D</td><td> 
</td><td>
<table>
<tr><th>D</th><th>Q_{n+1}</th></tr>
<tr><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td></tr>
</table>
</td></tr>
</table>

Table 1: A few basic Flip-flops

A **clock cycle** is the time from one rising edge to the next rising edge or from one falling edge to the next falling edge. The speed of the clock, measured in hertz (Hz), is the number of cycles per second. Typically, the clock speed for a microprocessor in an embedded system runs around 20 MHz, while the microprocessor in a personal computer runs upwards of 2 GHz and higher. A clock **period** is the time for one clock cycle (seconds per cycle), so it is just the inverse of the clock speed.

FF timing parameters

Propagation delay: propagation delay for a FF is the amount of time it takes for the output of the FF to change its state from a clock trigger or asynchronous set or rest. It is defined from the 50% point of the input pulse to the 50% point of the output pulse. Propagation delay is specified as t_{PHL} - the propagation time from a HIGH to a LOW, and as t_{PLH} - the propagation time from a LOW to a HIGH.

Output transition time: the output transition time is defined as the rise time or fall time of the output. The t_{TLH} is the 10% to 90% time, or LOW to HIGH transition time. The t_{THL} is the 90% to 10% time, or the HIGH to LOW transition time.

Setup time: the setup time is defined as the interval immediately preceding the active transition of the clock pulse during which the control or data inputs must be stable (at a valid logic level). Its parameter symbol is t_{su}.

Hold time: the hold time is the amount of time that the control or data inputs must be stable after the clock trigger occurs. The t_H is the parameter symbol for hold time.

Minimum pulse width: the minimum pulse width is required to guarantee a correct state change for the flip-flop. It is usually denoted by t_w.

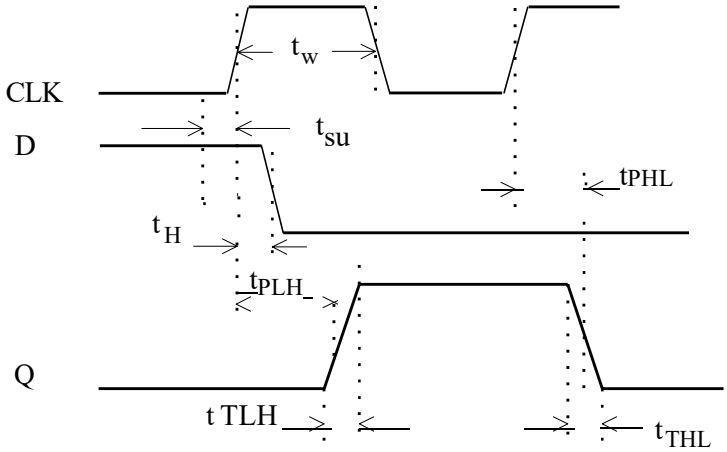
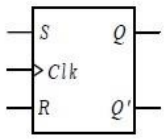
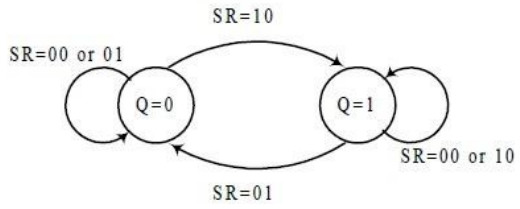


Figure.3.21

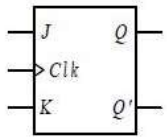
The ***characteristic equation*** is simply the Boolean equation that is derived directly from the characteristic table. Like the characteristic table, the characteristic equation specifies the flip-flop's next state, *Qnext*, as a function of its current state, *Q*, and input signals. The D flip-flop characteristic table has only one 1-minterm, which results in the simple characteristic equation for each flip-flop shown below .

What's an Excitation Table?

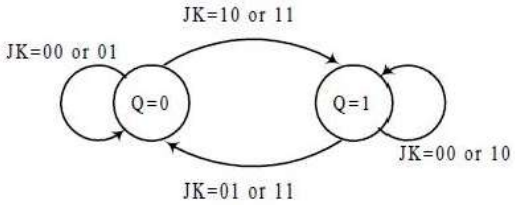
The **excitation table** is like the mirror image of the characteristic table by exchanging the input signal column(s) with the output (*Q*next) column. The excitation table shows what the flip-flop's inputs should be in order to change from the flip-flop's current state to the next state desired. In other words, the excitation table answers the question of what the flip-flop's inputs should be when given the current state that the flip-flop is in and the next state that we want the flip-flop to go to.

Name / Symbol	Characteristic (Truth) Table	State Diagram / Characteristic Equations	Excitation Table
SR  	*S R Q Qnext* 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 1 1 0 × 1 1 1 ×	 $Q_{next} = S + R'Q$ $SR = 0$	*Q Qnext S R* 0 0 0 × 0 1 1 0 1 0 0 1 1 1 × 0

JK



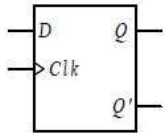
J	K	Q	Qnext
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



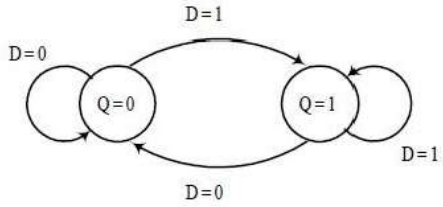
$$Q_{next} = J'K'Q + JK' + JKQ'$$
$$= J'K'Q + JK'Q + JK'Q' + JKQ'$$
$$= K'Q(J'+J) + JQ'(K'+K)$$
$$= K'Q + JQ'$$

Q	Qnext	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

D



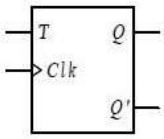
D	Q	Qnext
0	×	0
1	×	1



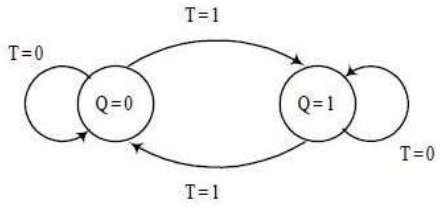
$$Q_{next} = D$$

Q	Qnext	D
0	0	0
0	1	1
1	0	0
1	1	1

T



T	Q	Qnext
0	0	0
0	1	1
1	0	1
1	1	0



$$Q_{next} = TQ' + T'Q = T \oplus Q$$

Q	Qnext	T
0	0	0
0	1	1
1	0	1
1	1	0

Conversion of Flip Flop :

For the conversion of one type of flip flop into another type of flip flop, we are required to design a combinational circuit. Steps to be followed :

step-1: Write the truth table of the Desired flip-flop

Step 2: Obtain the Excitation Table for the given Flip-Flop from its Truth Table

Step 3: Obtain Conversion Table

Step 4: Simplify the Expressions for the Inputs of the given Flip-Flop

Step 5: Design the Necessary Circuit and make the Connections accordingly

Conversion of SR flip flop:

1) SR flip flop to JK flip flop:

In this case we are required to convert SR flip flop into JK flip flop. Therefore we have to first design and connect the combinational circuit to the input of SR flip flop so that it will produce same outputs as that of JK flip flop. Here the external inputs are J and K. S and R will be the outputs of designed combinational circuit which are inputs of actual flip flop. We write a truth table with J, K, Qn, Qn+1, S and R. where Qn is the present state of the flip flop and Qn+1 will be the next state obtained when the particular J and K inputs are applied.

J, K and Qn can have eight combinations. For each combination of J, K and Qn find the corresponding Qn+1, i.e. determine to which next state the JK flip flop will go from the present state Qn if the present inputs J and K are applied. Now complete the table by writing the values of S and R required to get each Qn+1 from the corresponding Qn. i.e. write what values of S and R are required to change the state of the flip flop from Qn to Qn+1

Truth Table

Inputs		Outputs	
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Excitation Table

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

Conversion 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

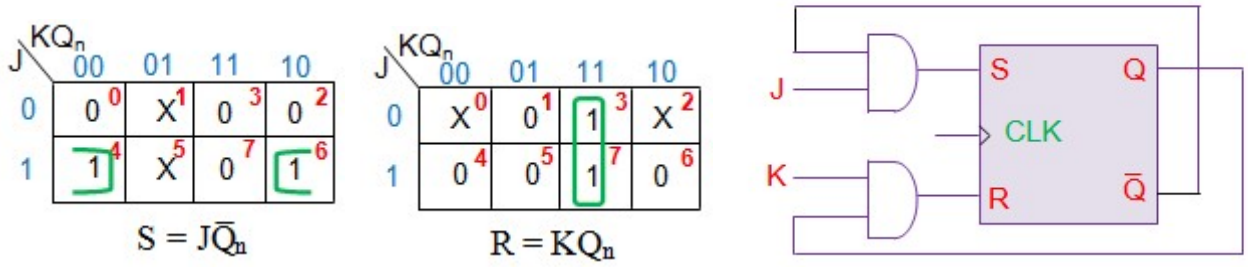


Figure.3.22

2) SR flip flop to D flip flop:

Similarly for the conversion of SR flip flop into the D flip flop we have connect combinational circuit to the inputs of the SR flip flop. In this case D the external input of the circuit. The output of the combinational circuit is connected to the inputs of the actual flip flop i.e. SR flip flop. Then the output of this flip flop will be the same as D flip flop.

The conversion table, K-maps, and Logic diagram for the conversion of SR flip flop to D flip flop.

1. Truth Table for T flip-flop

Input	Outputs	
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

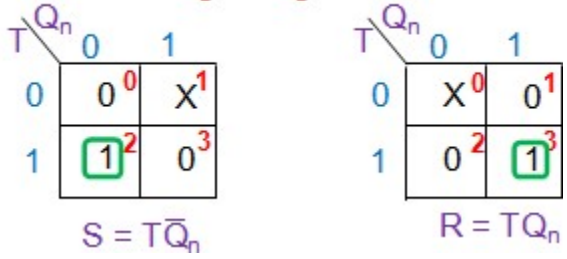
2. Excitation Table for SR flip-flop

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

3. Conversion Table

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

4. K-map Simplification



5. Circuit Design

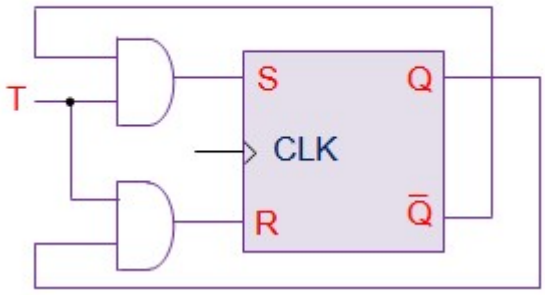


Figure.3.23

3. Conversion of SR Flip Flop to T Flip Flop:

1. Truth Table for T flip-flop

Input	Outputs	
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

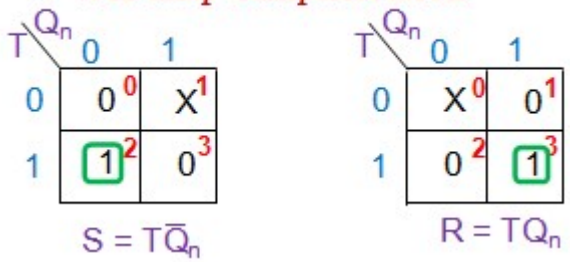
2. Excitation Table for SR flip-flop

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

3. Conversion Table

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

4. K-map Simplification



5. Circuit Design

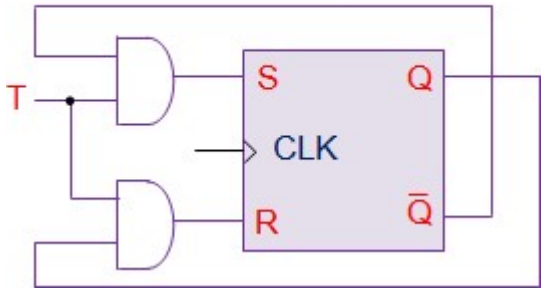


Figure.3.24

Conversion of D flip flop :

In the last article we have discussed about "how to convert JK flip flop into SR, D and T type of flip flop". Today we are going to convert D flip flop. We can convert D flip flop into SR and JK flip flop by using suitable combinational circuit. Combinational circuits for this purpose can be designed easily by using conversion tables and K-Maps. Let us see how to convert D flip flop into different types of flip flop.

1. Conversion of D Flip Flop to JK Flip Flop:

In case of conversion of D flip flop to JK flip flop we have to use J and K as the external inputs and D as the input of actual flip flop. J,K and Q_n makes eight possible combinations. Express D in terms of J, K and Q_n.

The conversion table, K-Maps and logic diagram for the conversion of D flip flop into JK flip flop is shown below:

1. Truth Table for JK flip-flop

Inputs		Outputs	
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

2. Excitation Table for D flip-flop

Outputs		Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

3. Conversion Table

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

4. K-map Simplification

J \ KQ_n	00	01	11	10
0	0 [0]	1 [1]	0 [3]	0 [2]
1	1 [4]	1 [5]	0 [7]	1 [6]

$$D = J\bar{Q}_n + \bar{K}Q_n$$

5. Circuit Design

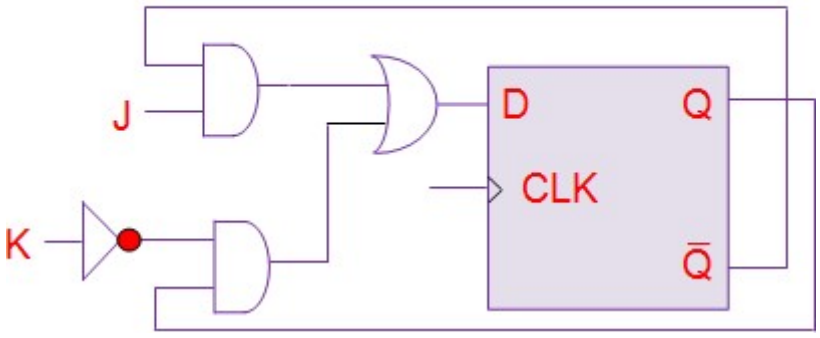


Figure.3.24

2. Conversion of D Flip Flop to SR Flip Flop : For converting D flip flop to SR flip flop, we use S and R as external inputs and D is the actual input to the flip flop. S, R and Q_n makes eight possible combinations, but S=R=1 is an invalid combination. So, the corresponding entries for Q_{n+1} and D are don't cares. Then we have to express D in terms of S, R and Q_n for the design of required flip flop.The conversion table, K-Maps and logic diagram for the conversion of D flip flop into SR flip flop is shown below:

1. Truth Table for SR flip-flop

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

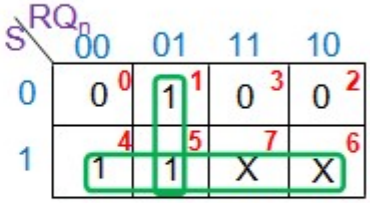
2. Excitation Table for D flip-flop

Outputs		Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

3. Conversion Table

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

4. K-map Simplification



$$D = S + \bar{R}Q_n$$

5. Circuit Design

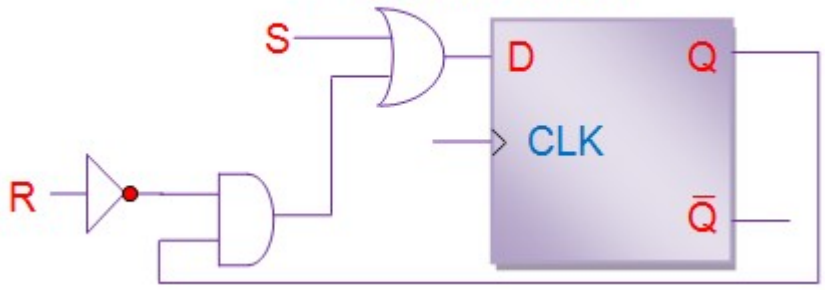


Figure.3.25

3.Conversion of D Flip Flop to T Flip Flop:

1. Truth Table for T Flip Flop

Input	Outputs	
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

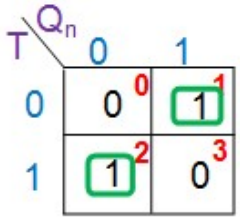
2. Excitation Table for D Flip Flop

Outputs		Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

3. Conversion Table

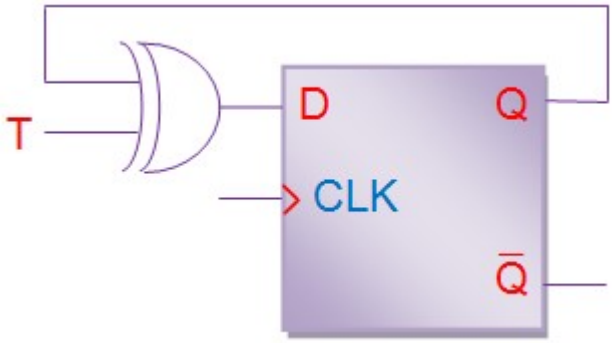
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

4. K-map Simplification



$D = T\overline{Q}_n + \overline{T}Q_n$

$= T \oplus Q_n$



5. Circuit Design

Figure.3.26

4.Conversion of T Flip Flop to JK Flip Flop :

1. Truth Table for JK Flip Flop

Inputs		Outputs	
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

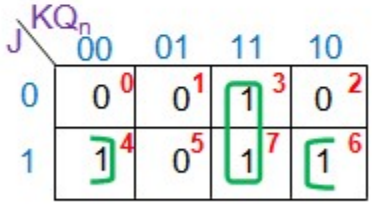
2. Excitation Table for T Flip Flop

Outputs		Input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

3. Conversion Table

J	K	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	1	1
1	1	1	0	1

4. K-map Simplification



$T = J\overline{Q}_n + KQ_n$

5. Circuit Design

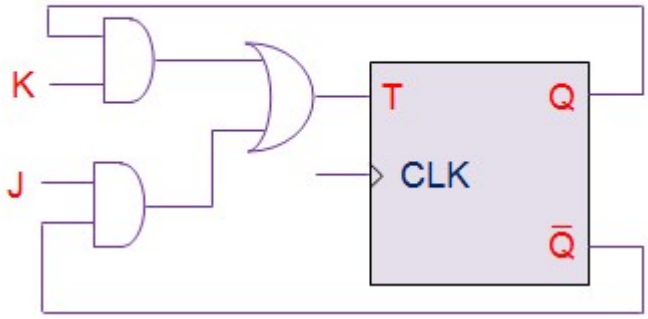


Figure.3.27

Conversion of JK flip flop : If we want to convert one type of flip flop into another type of flip flop, first we have to design a combinational circuit and after that connect it to the inputs of actual flip flop. So that the outputs of combinational circuit will be the inputs of actual flip flop and then it will produce same output as that of required flip flop.

We can convert JK flip flop into SR flip flop, T flip flop and D type of flip flop.

1) Conversion of JK flip flop to SR flip flop:

Step 1: Write the Truth Table of the Desired Flip-Flop, Here SR flip-flop is to be designed using JK flip-flop. Thus one needs to write the truth table for SR flip-flop.

Inputs		Outputs	
S	R	Present State, Q_n	Next State, Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	invalid	
1	1	invalid	

Truth Table for JK Flip-Flop

Inputs		Outputs	
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Step 2: Obtain the Excitation Table for the given Flip-Flop from its Truth Table Excitation tables provide the details regarding the inputs which must be provided to the flip-flop to obtain a definite next state (Q_{n+1}) from the known current state (Q_n).

From the truth table of JK flip-flop one can see that Q_{n+1} will become 0 from Q_n = 0 for both (i) J = K = 0 and (ii) J = 0 and K =1 (blue entries in first and third rows of the truth table).

This means that to obtain the next state, Q_{n+1} as 0 from the current state Q_n = 0, J must be made zero while K can be either 0 or 1.

This is indicated by the first row of the excitation table (blue entries in the first row of excitation table) where the value of K is expressed as 'X' indicating don't care condition. Similarly to obtain the next state as 1 from the current state 0, one has to have J equal to 1 while K can be either 0 or 1 (indicated by green entries of the truth table). This leads to the second row of excitation table (green entries) to be filled with values $Q_n = 0$, $Q_{n+1} = 1$, $J = 1$ and $K = X$. On the same grounds, entire excitation table needs to be filled (entries in pink and dark red colors).

Step 3: Append the Excitation Table of the given Flip-Flop to the Truth Table of the Desired Flip-Flop Appropriately to obtain Conversion Table, Here the conversion table is obtained by filling-up the values of the J and K inputs for the given Q_n and Q_{n+1}, by referring to the excitation table.

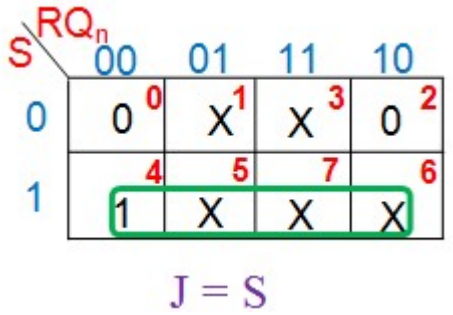
Excitation Table for JK Flip-Flop

Outputs		Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

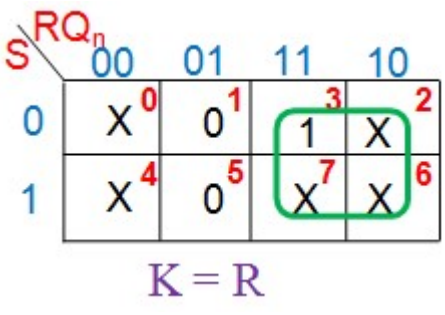
Conversion Table

S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	invalid		X	X
1	1	invalid		X	X

K-Maps



J = S



K = R

Step 4: Simplify the Expressions for the Inputs of the given Flip-Flop.In this case, one needs to arrive at the logical expressions for the inputs J and K in terms of S, R and Qn using suitable simplification technique like K-map.

Step 5: Design the Necessary Circuit and make the Connections accordingly. Here neither additional circuit nor new connections are necessary.

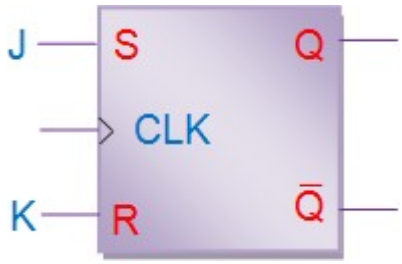


Figure.3.28

On the same grounds, one can convert the given flip-flop to any other type of flip-flop as shown below.

2) Conversion of JK flip flop to T flip flop:

For the conversion of JK flip flop to T type of flip flop, T will be the external input (input of combinational circuit) and the output of this combinational circuit is connected to the inputs of actual flip flop (J and K).

Then we prepare conversion table and using this table express J and K in terms of T and Q_n.The conversion table, K-Maps and logic diagram for the conversion of JK flip flop to T type of flip flop is shown below:

1. Truth Table for T Flip-Flop

Input	Outputs	
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

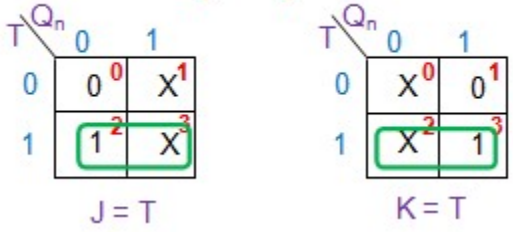
2. Excitation Table for JK Flip-Flop

Outputs		Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

3. Conversion Table

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

4. K-map Simplification



5. Circuit Design

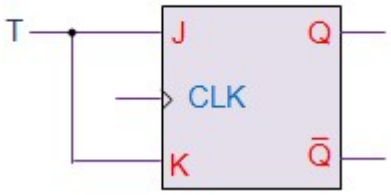


Figure.3.29

3) Conversion of JK flip flop to D flip flop:

In case of converting JK flip flop into D flip flop, D is the external input of combinational circuit, whereas J and K are the inputs of actual flip flop.

D and Q_n make four combinations. So, prepare a conversion table and using this table express J and K in terms of D and Q_n.

The conversion table, K-map and logic diagram for the conversion of JK flip flop to D flip flop is shown below:

1. Truth Table for D Flip-Flop

Input	Outputs	
D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

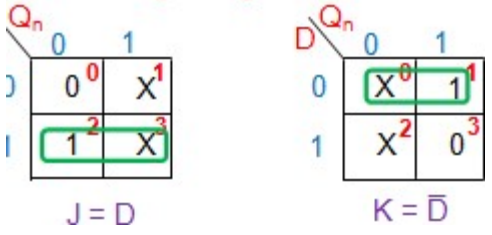
2. Excitation Table for JK Flip-Flop

Outputs		Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

3. Conversion Table

D	Q_n	Q_{n+1}	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

4. K-map Simplification



5. Circuit Design

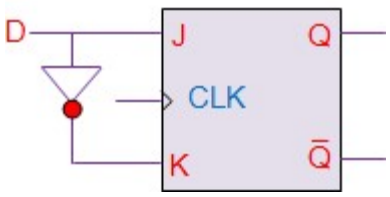


Figure.3.30

Applications of Flip-Flops:

Some of the common uses of the Flip-Flops are as follows:

1) Bounce elimination switch
2) Latch
3) Registers
4) Counters
5) Memory, etc.

The Problem of Clock Skew:

The clock doesn't arrive at all FF's at the same time. Skew is the difference between two clock edges, Correct behavior assumes next state of all storage elements determined by all storage elements *at the same time,* not possible in real systems!

- Logical clock driven from more than one physical circuit with timing behavior
- Different wire delay to different points in the circuit

Effect of Skew on Cascaded Flip-flops :

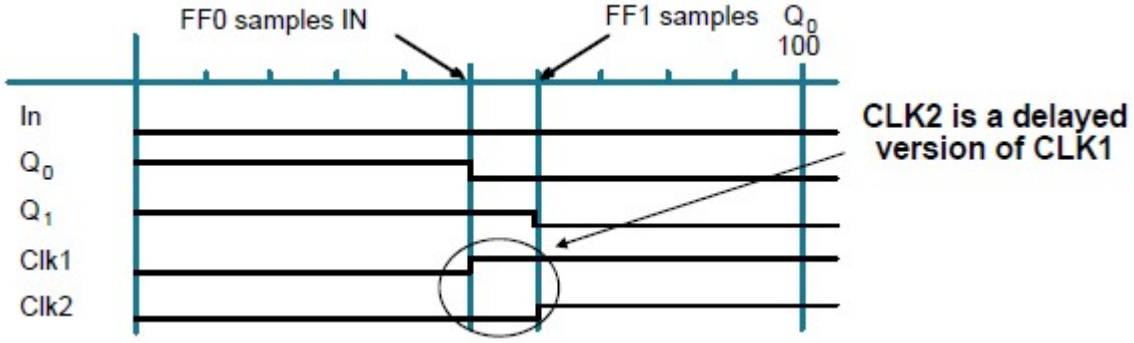


Figure.3.31

Original State: Q0 = 1, Q1 = 1, In = 0

Because of skew, next state becomes: Q0 = 0, Q1 = 0, not Q0 = 0, Q1 = 1

Design Strategies for Minimizing Clock Skew:

- distribute clock signals in general direction of data flows
- wire carrying the clock between two communicating components should be as short as possible
- for multiphase clocked systems, distribute all clocks in similar wire paths; this minimizes the possibility of overlap
- for the non-overlap clock generate, use the phase feedback signals from the furthest point in the circuit to which the clock is distributed; this guarantees that the phase is seen as low everywhere before it allows the next phase to go high

Shift registers:

In digital circuits, a **shift register** is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, *shifting in* the data present at its input and *shifting out* the last bit in the array, at each transition of the clock input. More generally, a **shift register** may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as **serial-in, parallel-out** (SIPO) or as **parallel-in, serial-out** (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also **bi-directional** shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a **circular shift register**

Buffer register:

The buffer register is the simple set of registers. It is simply stores the binary word. The buffer may be controlled buffer. Most of the buffer registers used D Flip-flops.

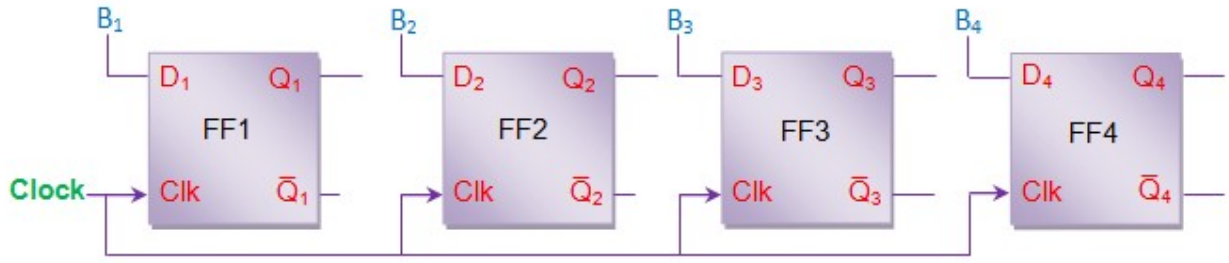


Figure 3.32: logic diagram of 4-bit buffer register

The figure shows a 4-bit buffer register. 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 terminals. i.e., the input word is loaded into the register by the application of clock pulse.

When the positive clock edge arrives, the stored word becomes: Q4Q3Q2Q1=X4X3X2X1;Q=X

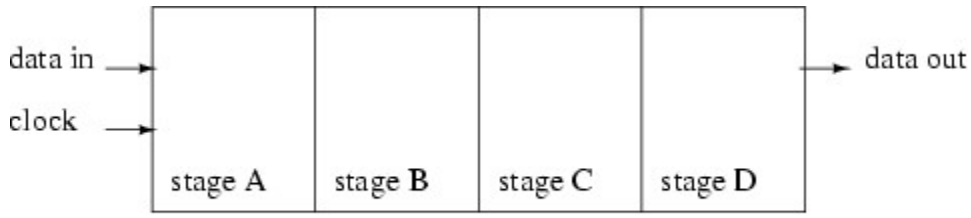
Controlled buffer register:

If goes LOW, all the FFs are RESET and the output becomes, Q=0000.

When is 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. Q4Q3Q2Q1=X4X3X2X1; Q=X

When load is low, the X bits cannot reach the FF‘s.

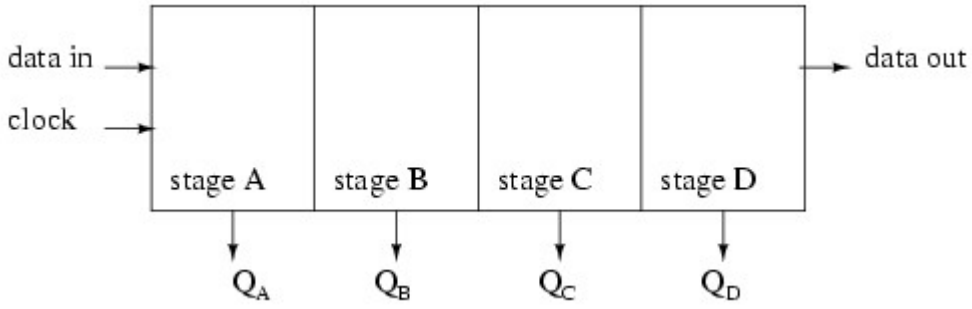
Data transmission in shift registers:



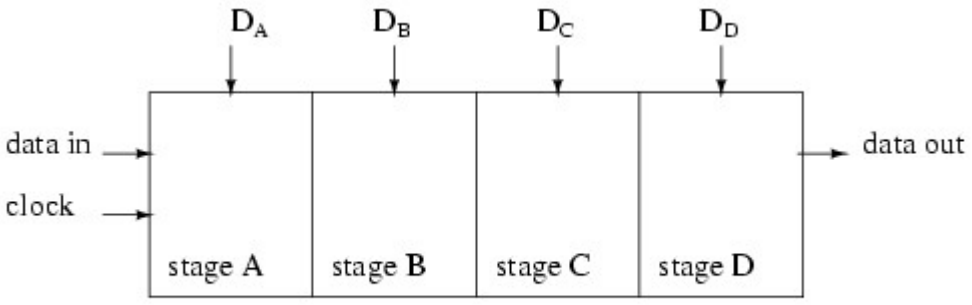
Serial-in, serial-out shift register with 4-stages

A number of ff's connected together such that data may be shifted into and shifted out of them is called shift register. data may be shifted into or out of the register in serial form or in parallel form. There are four basic types of shift registers.

1. Serial in, serial out, shift right, shift registers
2. Serial in, serial out, shift left, shift registers
3. Parallel in, serial out shift registers
4. Parallel in, parallel out shift registers



Serial-in, parallel-out shift register with 4-stages



Parallel-in, serial-out shift register with 4-stages

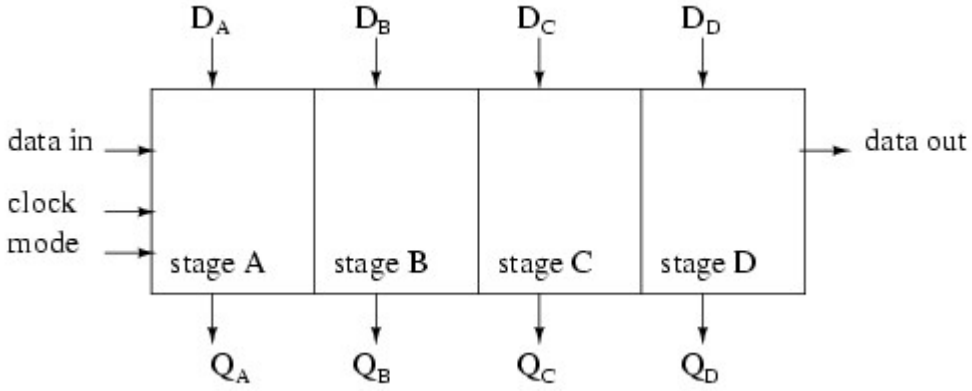


Figure.3.33 Parallel-in, parallel-out shift register with 4-stages

Serial IN, serial OUT, shift right, shift left register:

The logic diagram of 4-bit serial in serial out, right shift register with four stages. The register can store four bits of data. Serial data is applied at the input D of the first FF. the Q output of the first FF is connected to the D input of another FF. the data is outputted from the Q terminal of the last FF.

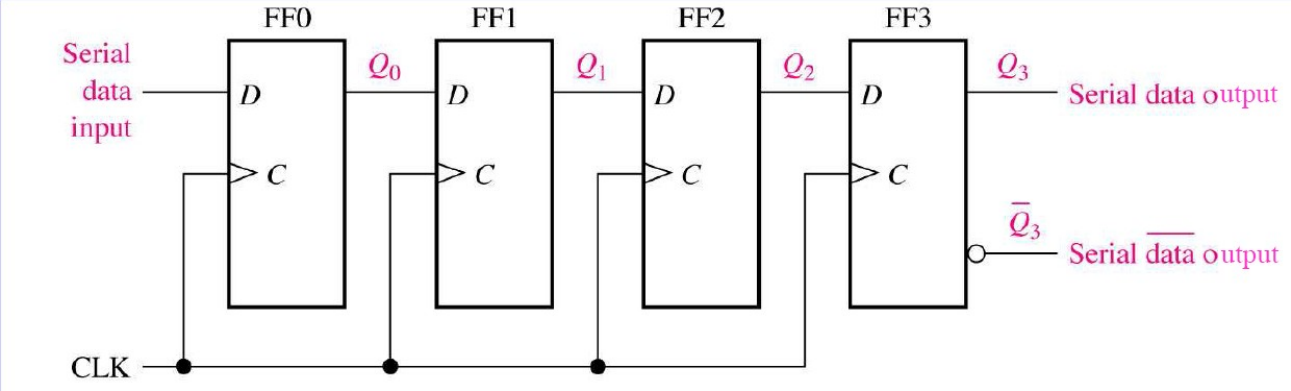
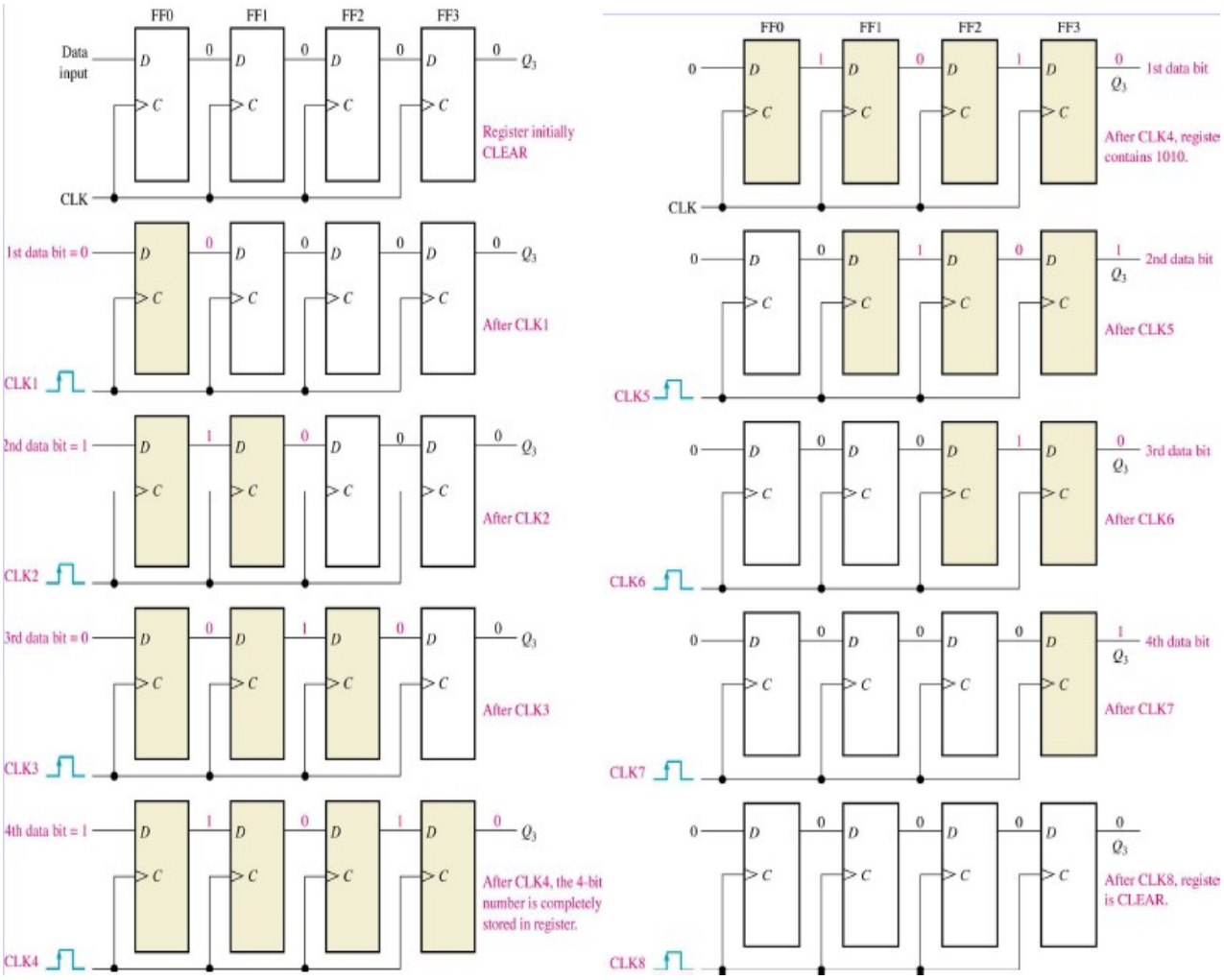


Figure.3.34

When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.

Figure.3.35



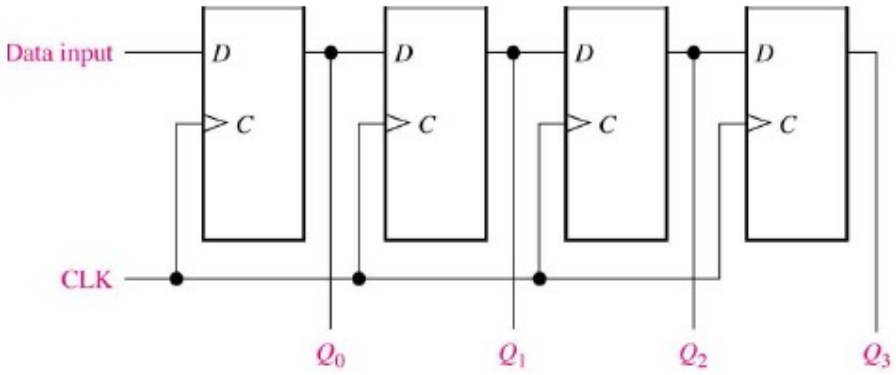
	CLK	Serial Input D_{in}	Q_3	Q_2	Q_1	Q_0
Initially			0	0	0	0
1st	↓	1	1	0	0	0
2nd	↓	1	1	1	0	0
3rd	↓	1	1	1	1	0
4th	↓	1	1	1	1	1

Direction of data travel →

Serial-in, parallel-out, shift register:

In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis with the serial output. The serial-in, parallel out, shift register can be used as serial-in, serial out, shift register if the output is taken from the Q terminal of the last FF.



Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

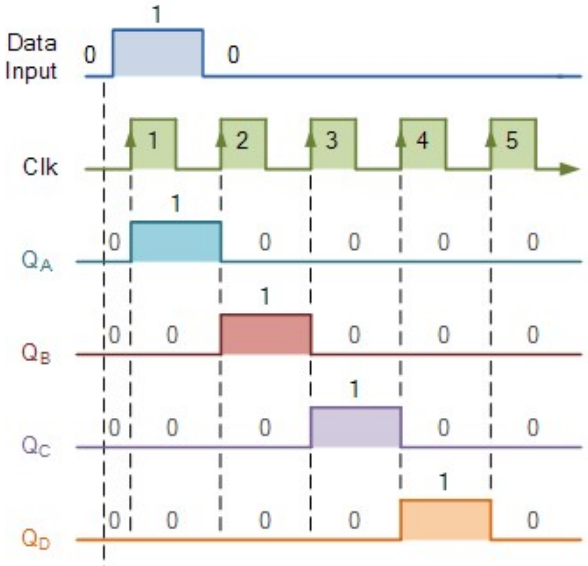


Figure.3.36

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 on one line as with serial data bits are transferred out of the register serially. On a bit-by-bit basis over a single line.

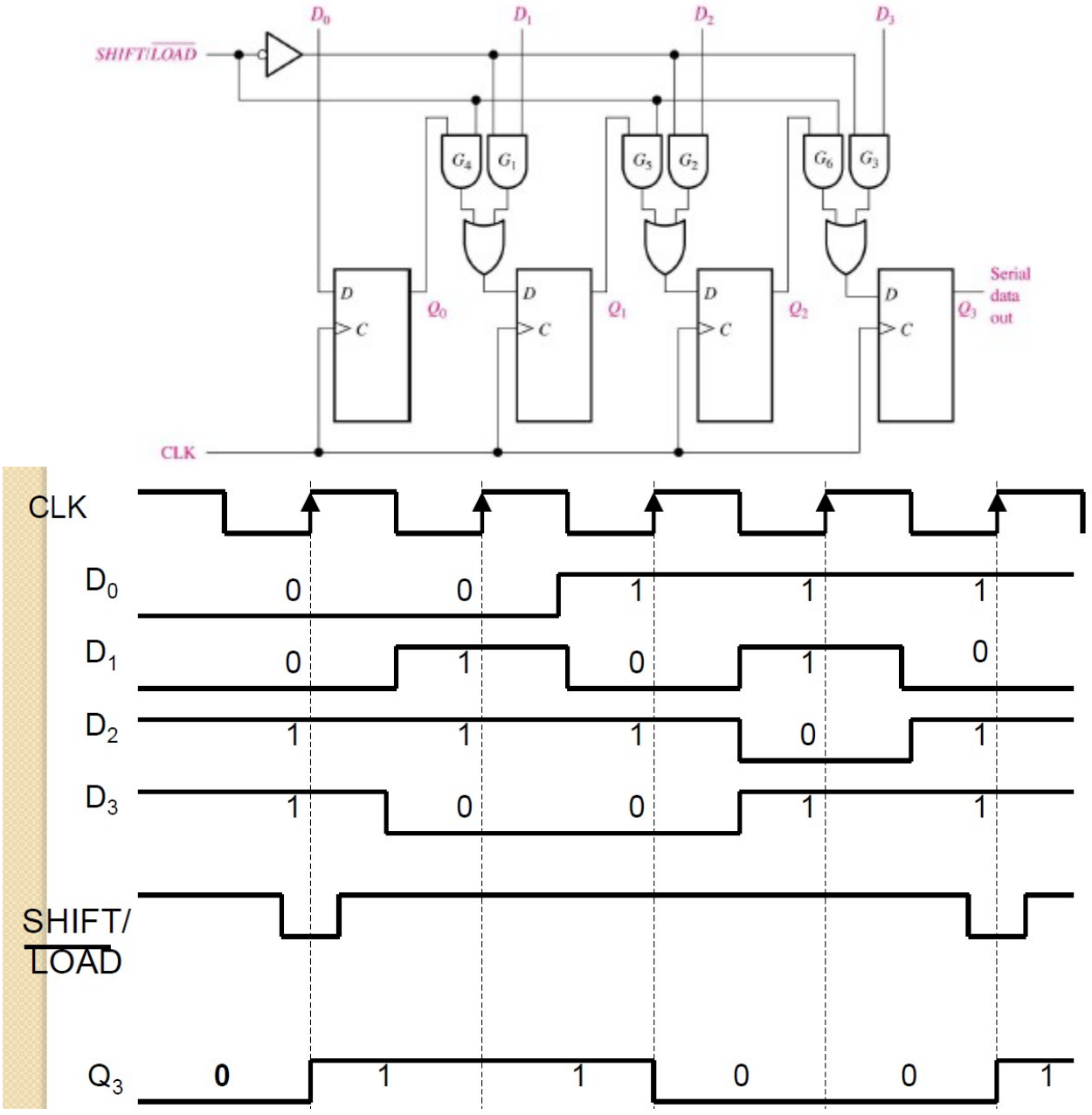


Figure.3.37

There are four data lines A, B, C, D through which the data is entered into the register in parallel form. The signal shift/ load allows the data to be entered in parallel form into the register and the data is shifted out serially from terminalQ4.

Parallel-in, parallel-out, shift register

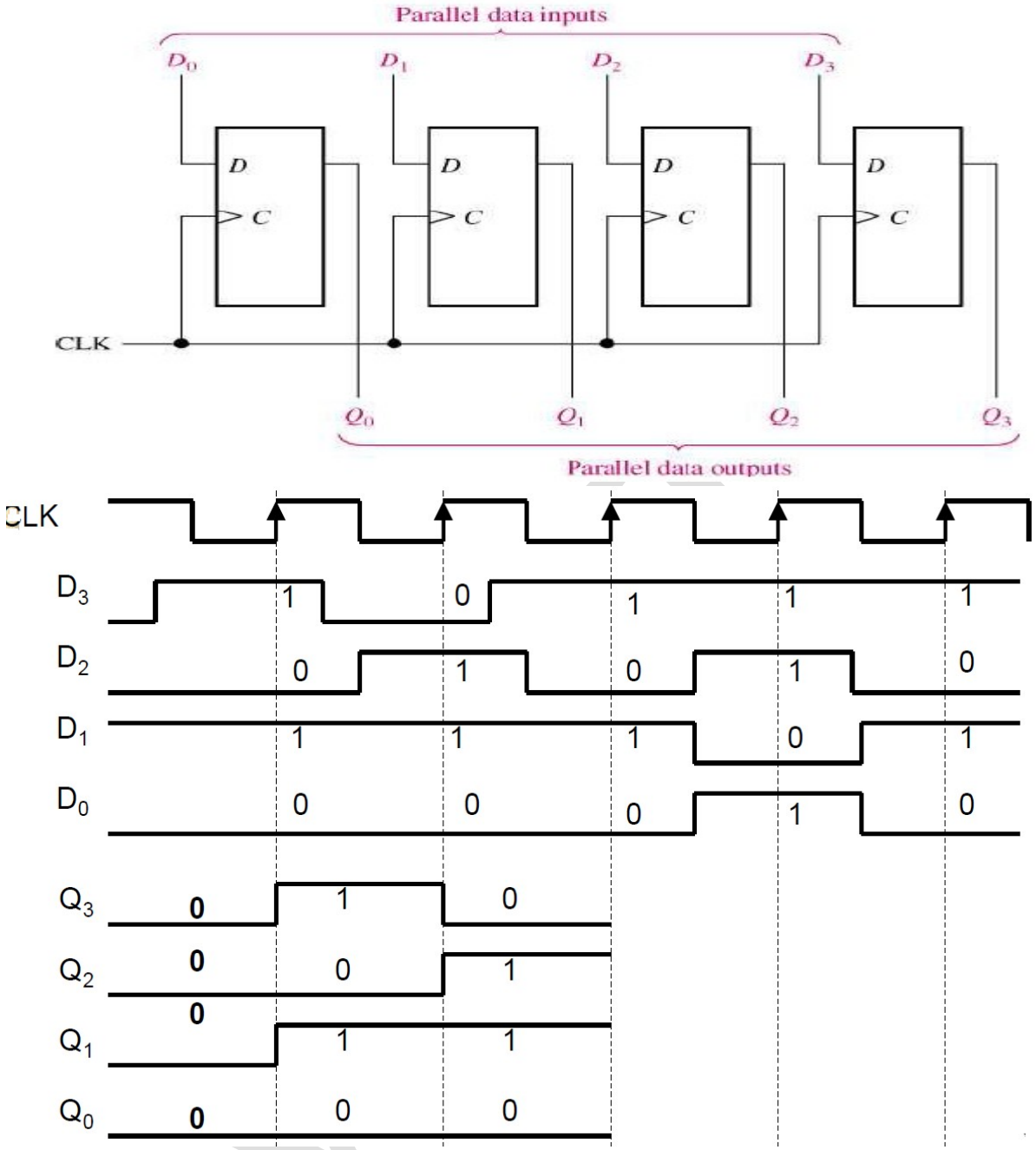


Figure.3.38

In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Data is applied to the D input terminals of the FF's. When a clock pulse is applied, at the positive going edge of the pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

Bidirectional shift register:

A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A fig shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, when right /left is a 1, the logic circuit works as a shift-register.the bidirectional operation is achieved by using the mode signal and two NAND gates and one OR gate for each stage.

A HIGH on the right/left control input enables the AND gates G1, G2, G3 and G4 and disables the AND gates G5,G6,G7 and G8, and the state of Q output of each FF is passed through the gate to the D input of the following FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the right/left control inputs enables the AND gates G5, G6, G7 and G8 and disables the And gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register.

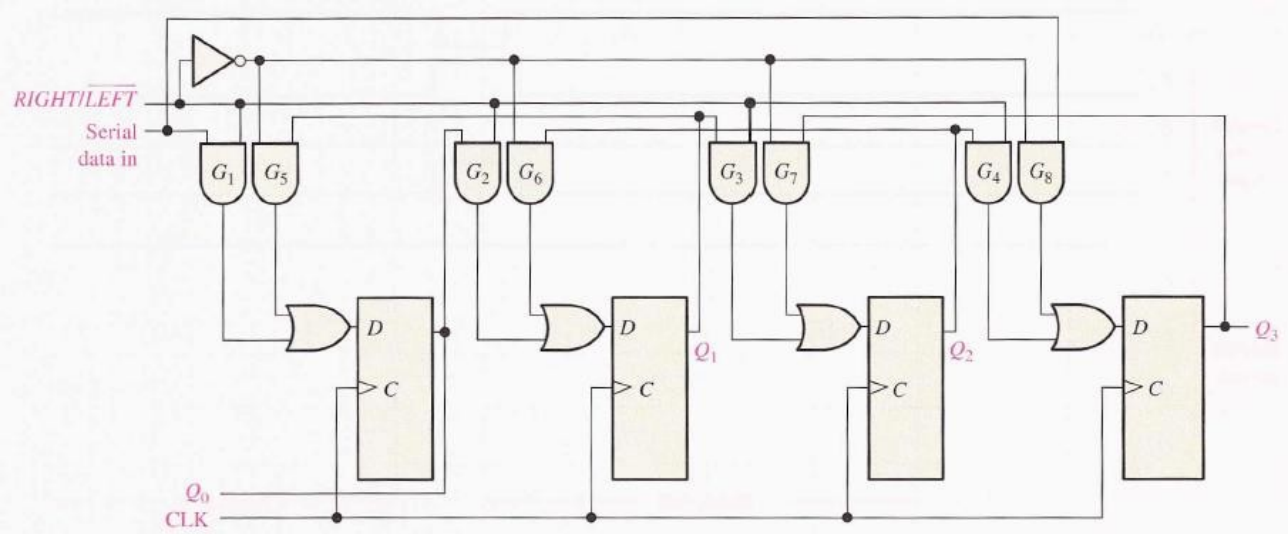
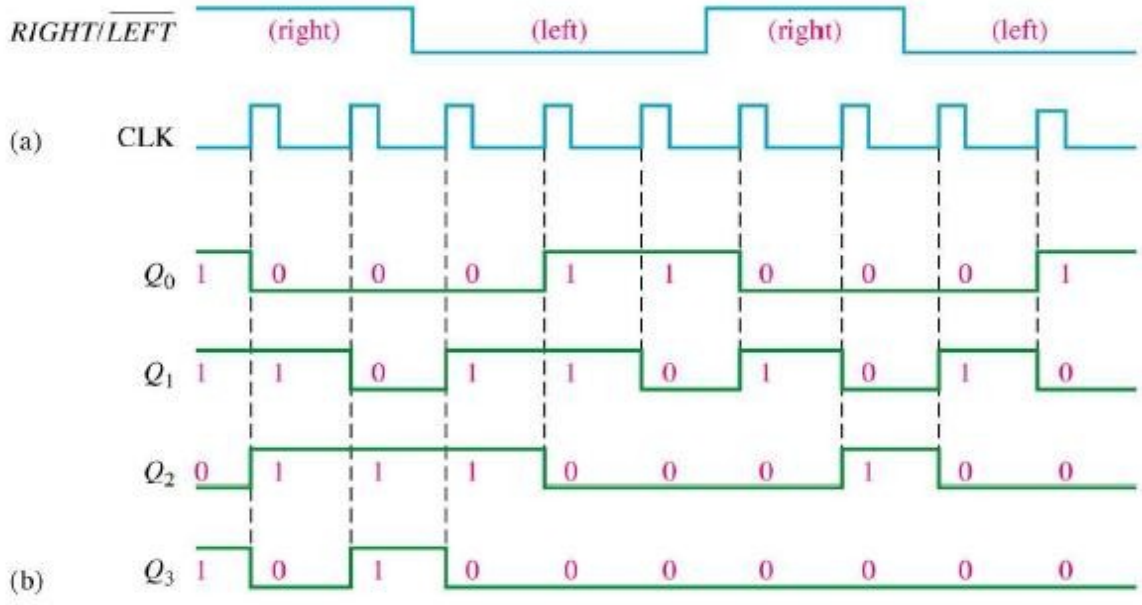


Figure 3.39: logic diagram of a 4-bit bidirectional shift register



Universal shift register:

A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift both directions is a bidirectional shift register. If the register has both shifts and parallel load capabilities, it is referred to as a universal shift registers. Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or I parallel form.

The most general shift register has the following capabilities.

1. A clear control to clear the register to 0
2. A clock input to synchronize the operations
3. A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right.
4. A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left
5. A parallel loads control to enable a parallel transfer and the n input lines associated with the parallel transfer N parallel output lines
6. A control state that leaves the information in the register unchanged in the presence of the clock.

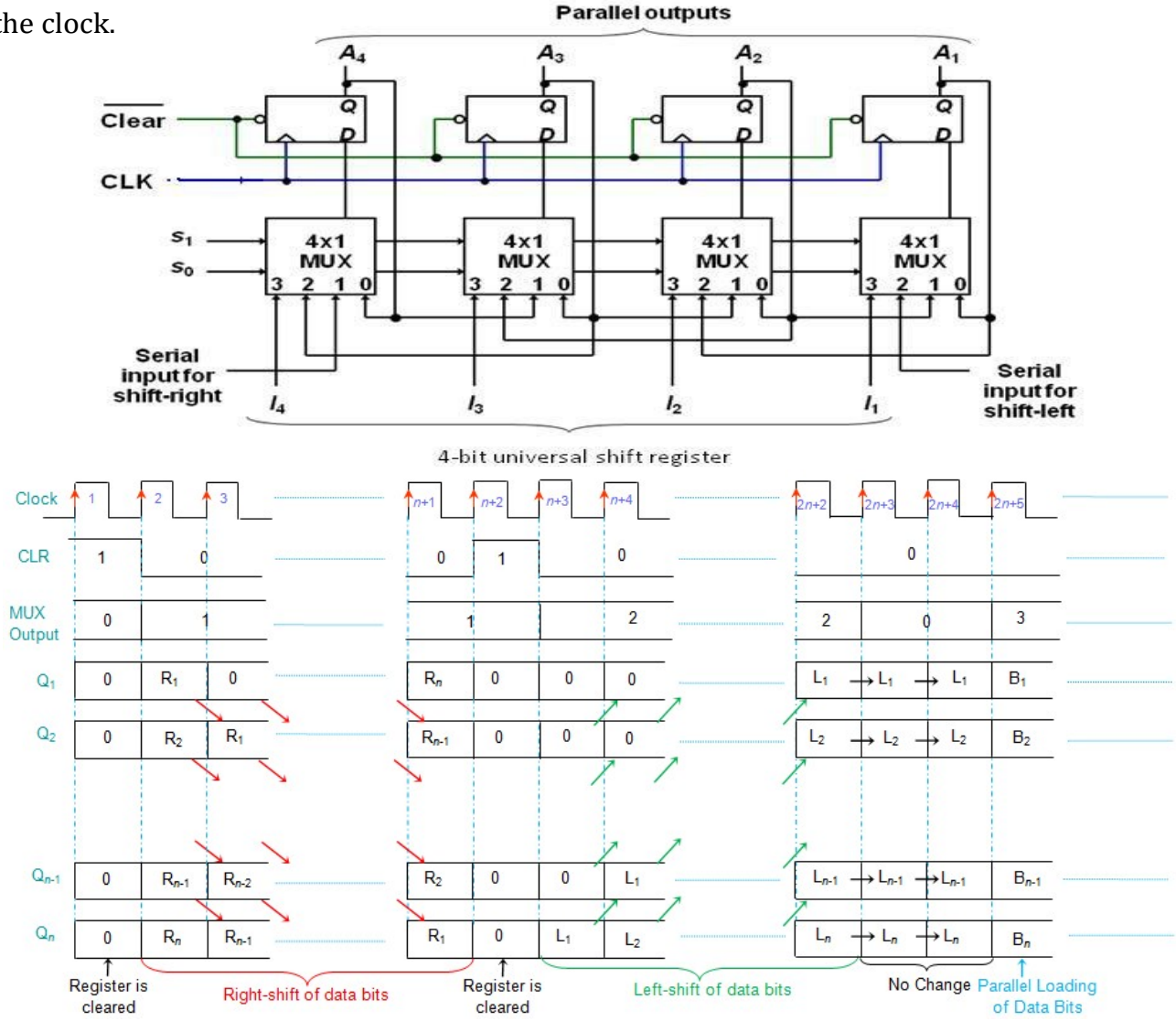


Figure.3.40

A universal shift register can be realized using multiplexers. It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s1 and s0. Input 0 in each multiplexer is selected when S1S0=00, input 1 is selected when S1S0=01 and input 2 is selected when S1S0=10 and input 4 is selected when S1S0=11. The selection inputs control the mode of operation of the register according to the functions entries. When S1S0=0, the present value of the register is applied to the D inputs of flip-flops. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When S1S0=01, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes a shift-right operation, with serial input transferred into flip-flopA4. When S1S0=10, a shift left operation results with the other serial input going into flip-flop A1. Finally when S1S0=11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock cycle.

Counters:

A counter is a sequential machine that produces a specified count sequence. The count changes whenever the input clock is asserted.

There is a great variety of counter based on its construction.

1. Clock: Synchronous or Asynchronous
2. Clock Trigger: Positive edged or Negative edged
3. Counts: Binary, Decade
4. Count Direction: Up, Down, or Up/Down
5. Flip-flops: JK or T or D

Counter that follows the binary number sequence is called a *binary counter*. An n-bit binary counter that cycles through all 2^n states in ascending (or descending) order. It consists of n flip-flops and can count in binary from 0 through $2^n - 1$. A counter may also follow any other sequence of states.

A modulo- N counter (also known as a divide-by- N counter) follows the sequence of N states repeatedly.

There are **two** categories of counters: **asynchronous** counters and **synchronous** counters.

In an **asynchronous** counter, all the bits in the count change synchronously with the assertion of the clock. With an asynchronous circuit, all the bits in the count do not all change at the same time.

Asynchronous counters are also called **ripple counters** because of the way the clock pulse ripples it way through the flip-flops.

In a **synchronous counter**, all flip-flops receive the common clock. Synchronous counters are faster than asynchronous counter because in synchronous counter all flip flops are clocked simultaneously.

3-bit Up Counter	3-bit Down Counter
000	000
001	111
010	110
011	101
100	100
101	011
110	010
111	001

Binary Ripple Counter (asynchronous UP Counter)**:**

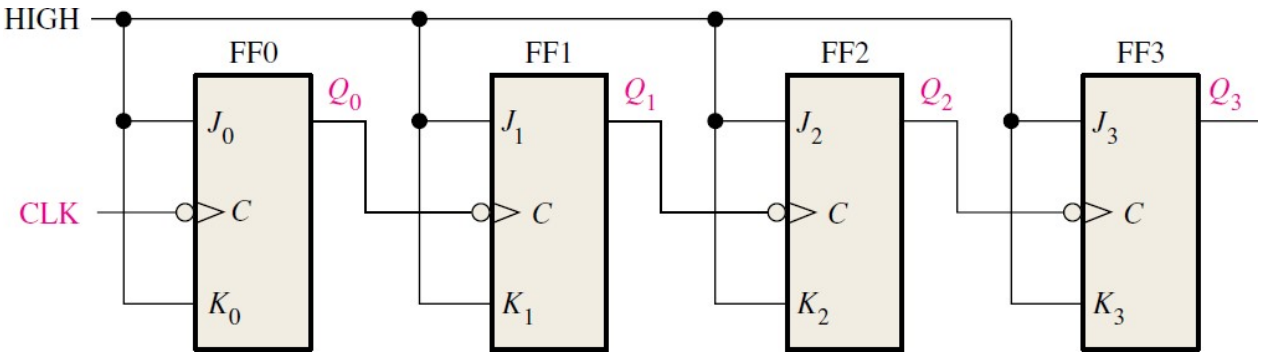
A binary ripple counter is an asynchronous counter where only the first flip-flop is clocked by an external clock. Each subsequent flip-flop is triggered by the transition occurring in the preceding flip-flop.

The main characteristic of an asynchronous counter is each flip-flop derives its own clock from other flip-flops and is therefore independent of the input clock. Consequently, the output of each flip-flop may change at different time, hence the term asynchronous. From the asynchronous counter diagram above, we observed that the output of the first flip-flop becomes the clock input for the second flip-flop, and the output of the second flip-flop becomes the clock input for the third flip-flop etc.

For the first flip-flop, the output changes whenever there is a negative transition in the clock input. This means that the output of the first flipflop produces a series of square waves that is half the frequency of the clock input. Since the output of the first flip-flop becomes the clock of the second flip-flop, the output of the second flip-flop is half the frequency of its clock, i.e. the output of the first flip-flop that in turn is half the frequency of the clock input. This behavior, in essence is captured by the binary bit pattern in the counting sequence

A counter may count up or count down or count up and down depending on the input control. Going through a count sequence from 0000,0001,0010, and so on, we note that the least significant bit, Q_0, toggles once with each count pulse input. Q_1 is toggled every time that Q_0 changes from 1 to 0. Q_2 is toggled every time that Q_1 changes from 1 to 0 and so on for any other higher order bits of a ripple counter. A 4-bit binary ripple counter constructed using T flip-flops and D flip-flops is shown in Fig.

In a count-down counter, LSB is toggled with every count pulse. Any higher-order bit is toggled if the preceding bit has a transition from 0 to 1. In this counter, the complement terminals Q' of all flip-flops give the outputs. Therefore, the diagram of a binary countdown counter looks the same as the binary ripple counter in Fig. , provided that all flip-flops trigger on the positive edge of the clock. (The bubble in the C inputs must be removed.)



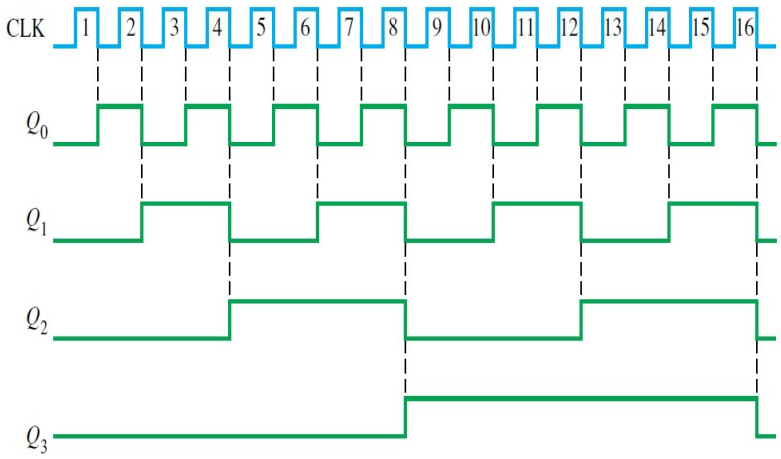


Figure.3.41

CLK PLUSE	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16 REPEAT	0	0	0	0

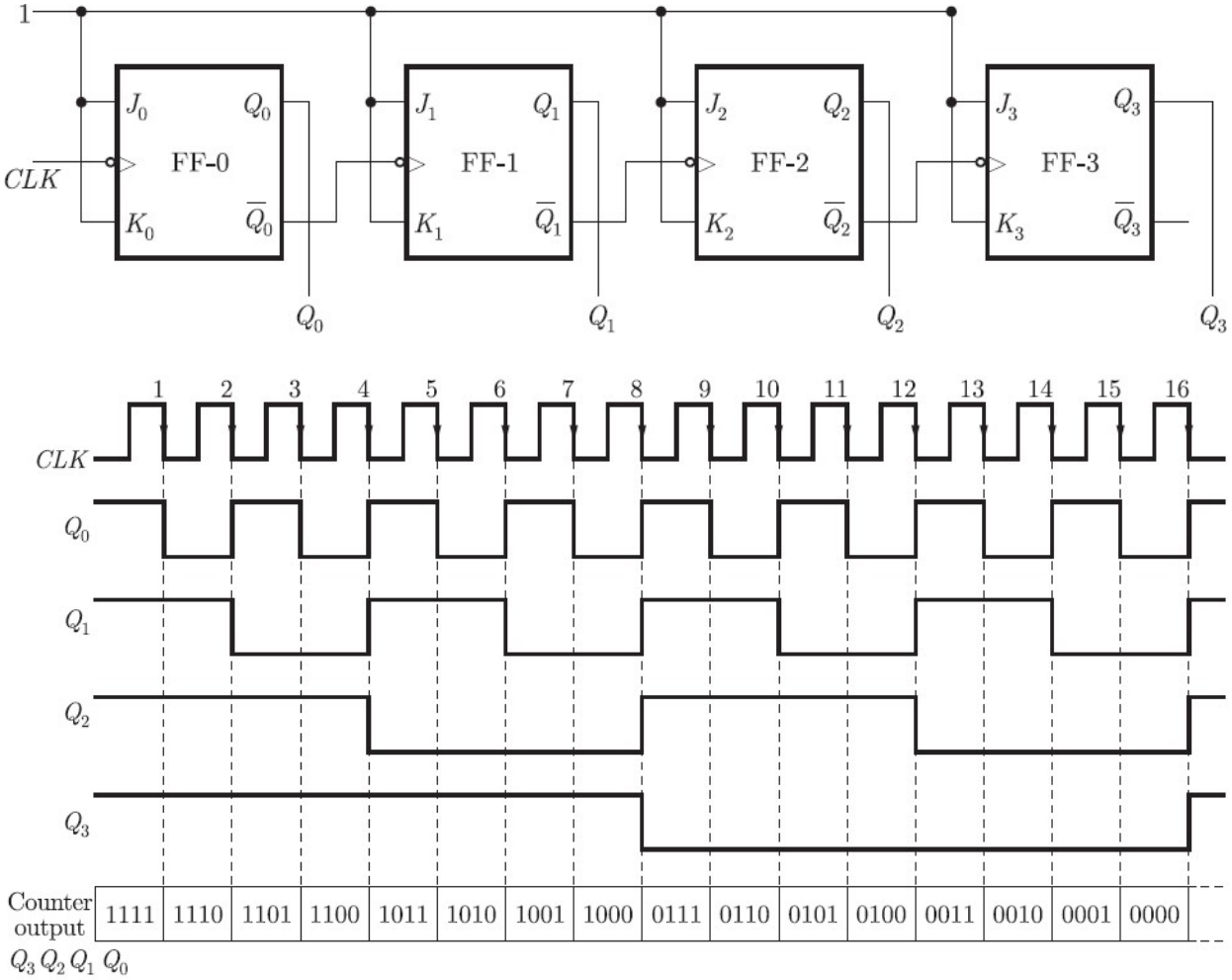
4-bit Ripple up counter:

A 4-bit up-counter counts from 0 to 15 or in binary, from 0000 to 1111. A 4-bit ripple up-counter, using negative edge-triggered J –K FFs is shown in Figure 8.4.9. The inputs J and K of flip-flops are connected to logic 1. Each flip-flop acts as a toggle switch.

The output of the first flip-flop feeds the clock input of the second, and the output of the second flip-flop feeds the clock input of the third, the output of which in turn feeds the clock input of the fourth flip-flop. The outputs of the four flip-flops are designated as $Q0$ (LSB), $Q1$, $Q2$ and $Q3$ (MSB). Figure 8.4.10 shows the waveforms appearing at $Q0$, $Q1$, $Q2$ and $Q3$ outputs as the clock signal goes through successive cycles of trigger pulses.

If you compare this counter with the up-counter in Figure 8.4.9 the only difference you will notice is that, in the down counter in Figure 8.4.11, the complement output instead of the normal output, is connected to the clock input of the next flip-flop. However, the counter output in both up and down counters, is the normal output, Q, of the flip-flops.

The timing diagram is shown in Figure 8.4.12. It shows the waveforms appearing at $Q0$, $Q1$, $Q2$ and $Q3$ outputs as the clock signal goes through successive cycles of trigger pulses.



Clock Pulse	Q_3	Q_2	Q_1	Q_0
0	1	1	1	1
1	1	1	1	0
2	1	1	0	1
3	1	1	0	0
4	1	0	1	1
5	1	0	1	0
6	1	0	0	1
7	1	0	0	0
8	0	1	1	1
9	0	1	1	0
10	0	1	0	1
11	0	1	0	0
12	0	0	1	1
13	0	0	1	0
14	0	1	0	1
15	0	0	0	0

Figure.3.42

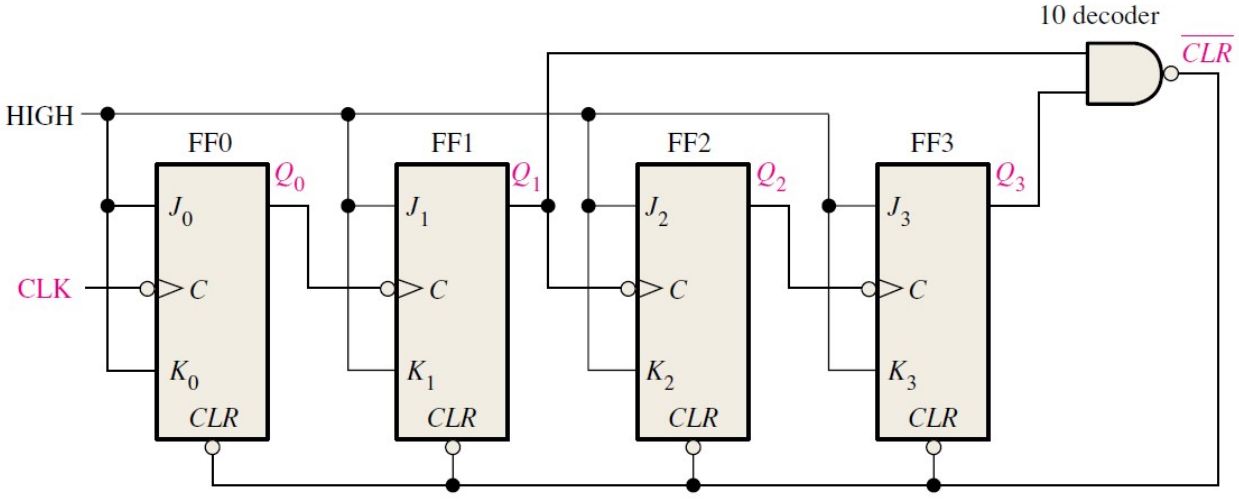
Asynchronous Decade Counters:

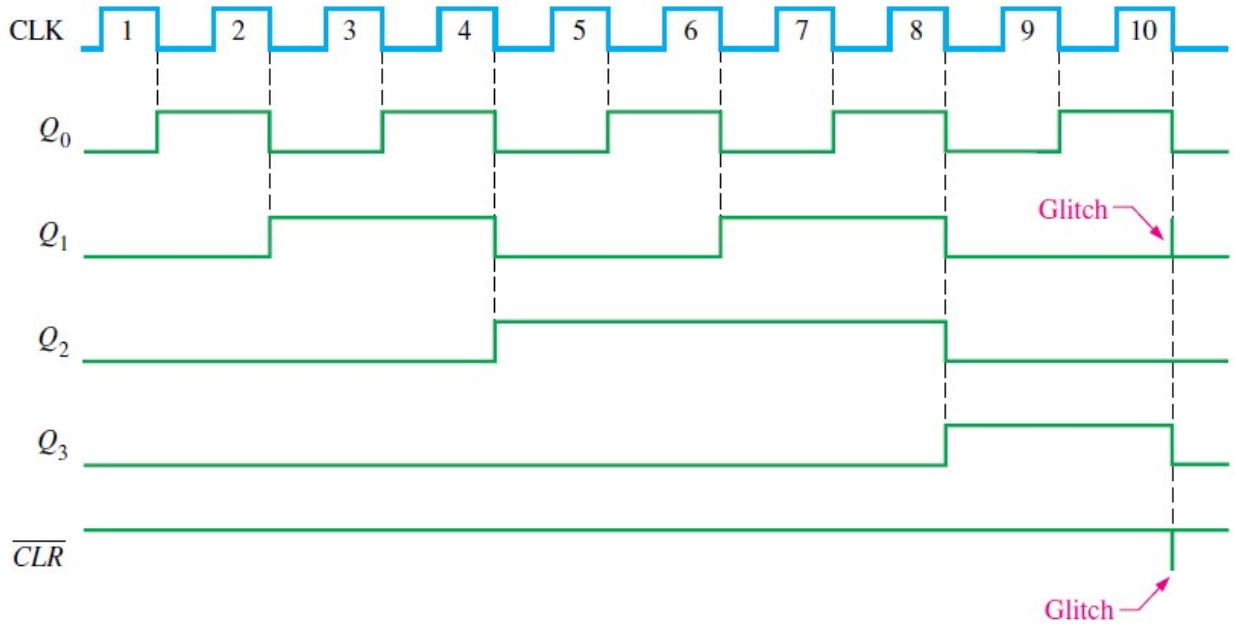
The modulus of a counter is the number of unique states through which the counter will sequence. The maximum possible number of states (maximum modulus) of a counter is $2n$, where n is the number of flip-flops in the counter. Counters can be designed to have a number of states in their sequence that is less than the maximum of $2n$. This type of sequence is called a *truncated sequence.* One common modulus for counters with truncated sequences is ten (called MOD10).

Counters with ten states in their sequence are called decade counters. A decade counter with a count sequence of zero (0000) through nine (1001) is a BCD decade counter because its ten-state sequence produces the BCD code. This type of counter is useful in display applications in which BCD is required for conversion to a decimal readout. To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states. For example, the BCD decade counter must recycle back to the 0000 state after the 1001 state. A decade counter requires four flip-flops (three flip-flops are insufficient because 23 = 8).

Let's use a 4-bit asynchronous counter such as the one in Example 1 and modify its sequence to illustrate the principle of truncated counters. one way to make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output of the NAND gate to the clear (*CLR*) inputs of the flip-flops, as shown in Figure 12(a).

Partial Decoding Notice in Figure 12(a) that only $Q1$ and $Q3$ are connected to the NAND gate inputs. This arrangement is an example of *partial decoding,* in which the two unique states ($Q1$ = 1 and $Q3$ = 1) are sufficient to decode the count of ten because none of the other states (zero through nine) have both $Q1$ and $Q3$ HIGH at the same time. When the counter goes into count ten (1010), the decoding gate output goes LoW and asynchronously resets all the flip-flops.





Input Pulses	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
0	0	0	0	0 (resets)

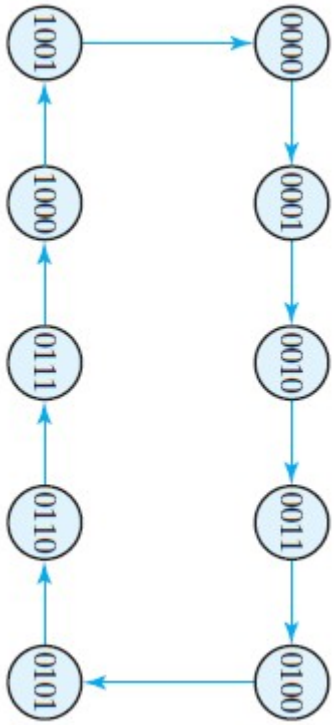


Figure.3.43

The resulting timing diagram is shown in Figure 12(b). Notice that there is a glitch on the $Q1$ waveform. The reason for this glitch is that $Q1$ must first go HIGH before the count of ten can be decoded. Not until several nanoseconds after the counter goes to the count of ten does the output of the decoding gate go LoW (both inputs are HIGH). Thus, the counter is in the 1010 state for a short time before it is reset to 0000, thus producing the glitch on $Q1$ and the resulting glitch on the *CLR* line that resets the counter.

BINARY RIPPLE COUNTER WITH A MODULUS OF LESS THAN 2^N :

The ripple counters discussed so far are full modulus counters. That is if n is the number of flip-flops or bits, then it will have MOD number equal to $2n$. This is the maximum MOD-number that can be obtained using n flip-flops.

The basic n-flip-flop binary ripple counter can be modified to have a modulus less than $2n$, with the help of simple externally connected combinational logic. This type of counter does not utilize all the possible states. Some of the states will be skipped. We will illustrate this simple concept with the help of an example.

M E T H O D O L O G Y :

1. Determine the minimum number of flip-flops n so that $N < 2^n$ and connect these flip-flops as a binary ripple counter

2. Find the binary number for N.

3. Identify the flip-flops for which output Q = 1, when the count is N. Choose a NAND gate with the number of inputs equal to the number of flip-flops for which output Q = 1. As an example, if the objective were to design an MOD-11 counter, then, in the corresponding count, that is, 1011, three flip-flops output are 1. The desired NAND gate would therefore be a three-input gate.

4. Connect the Q outputs of the identified flip-flops to the inputs of the NAND gate and the NAND gate output to asynchronous clear inputs of all flip-flops.

Design of MOD-6 Ripple Counter:

The steps are as follows:

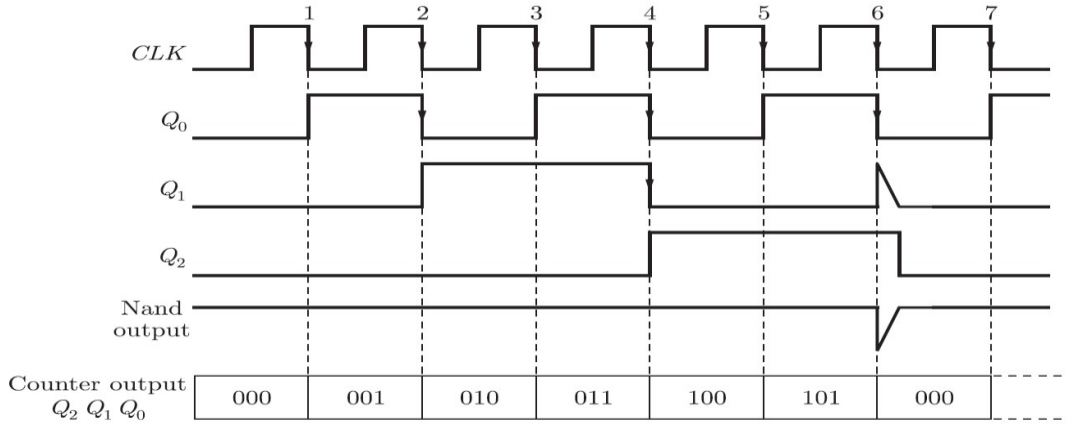
1. Here N = 6, so the minimum number of flip-flops such that $6 < 2^3$, will be 3. Now, connect three-flops as a binary ripple counter shown in Figure.

2. The binary number for N is 110.

3. The flip-flops for which output will be 1 at the count 110 are FF2 and FF1. So we choose a 2-input NAND gate.

4. Connect the Q outputs of the FF2 and FF1 flip-flops that is $Q2$ and $Q1$ to the inputs of the NAND gate and the NAND gate output to asynchronous clear inputs of all flip-flops.

The arrangement is shown in Figure, and the timing diagram is shown below.



Timing Diagram of MOD-6 ripple counter

The counter counts from 000 to 101 until NAND gate output is HIGH. But, when it goes to state 110, all NAND gate inputs becomes HIGH and its output will be LOW. Therefore, all the flip-flops will be cleared and it resets back to state 000. From timing diagram we can see that the frequency of the $Q2$ output is one-sixth of the input clock frequency. In other words, this MOD-6 counter has divided the input frequency by 6.

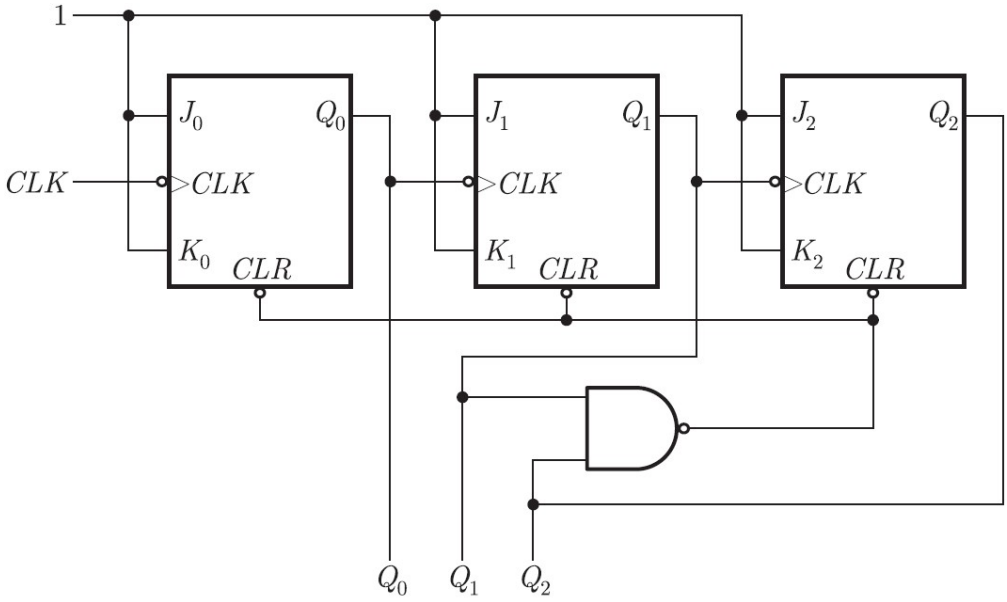


Figure.3.44:Logic Diagram of MOD-6 Ripple Counter

Synchronous Counters:

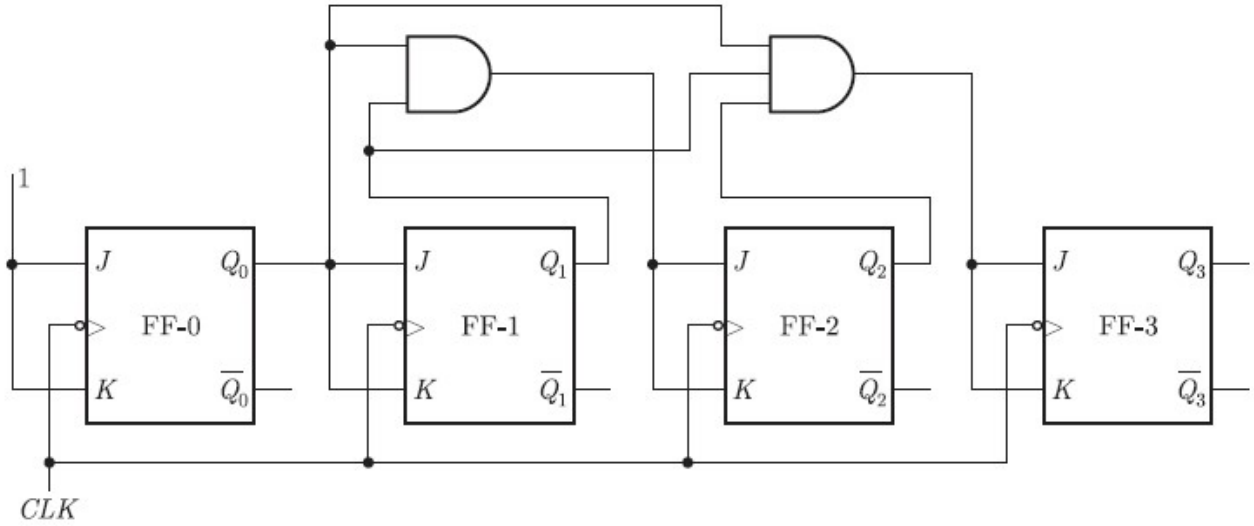
In synchronous counters, all flip-flops are triggered by common clock pulse input. This is different from ripple counter where flip-flops are triggered one at a time in succession.

In this counter, the flip-flop corresponding to LSB is toggled with every clock pulse. A flip-flop in any other position is toggled on the next clock pulse provided all the preceding (lower-order) bits are equal to 1.Note that the polarity of the clock is important in the ripple counter and not here.

Synchronous *Binary Up Counters :*

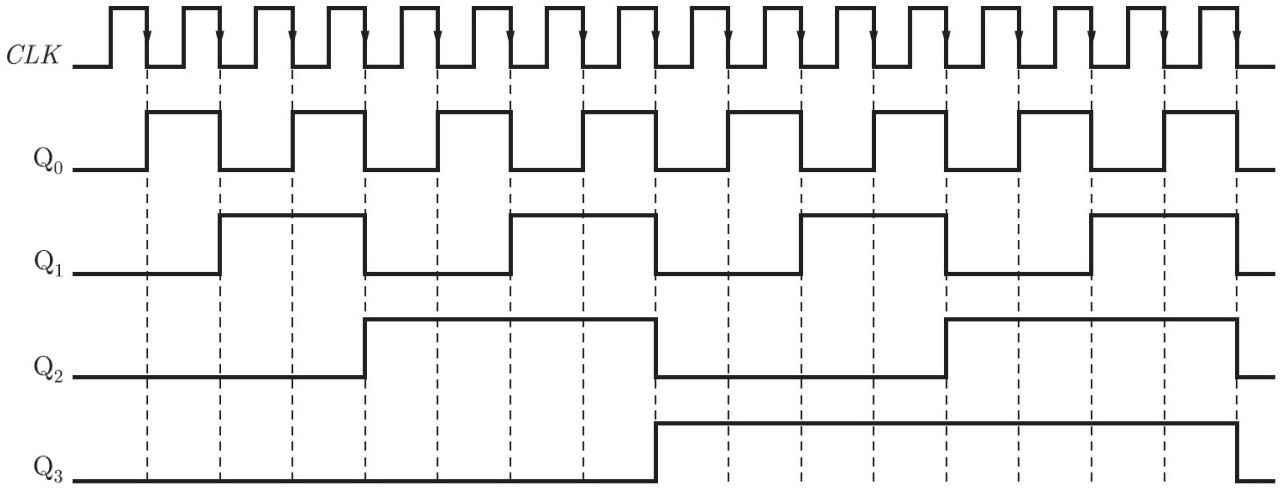
A synchronous binary counter counts from 0 to 2^N-1, where N is the number of bits/flip-flops in the counter. Each flip-flop is used to represent one bit. The flip-flop in the lowest-order position is complemented/toggled with every clock pulse and a flip-flop in any other position is complemented on the next clock pulse provided all the bits in the lower-order positions are equal to 1.

For example, consider the counting sequence of a four bit binary count shown in Table. We note that flip-flop FF-0 toggles with every clock pulse, flip-flop FF-1 toggles only when the output of FF0 is in the '1' state, flip-flop FF2 toggles only with those clock pulses when the outputs of FF0 and FF1 are both in the logic '1' state and flip-flop FF3 toggles only with those clock pulses when $Q0$, $Q1$ and $Q2$ are all in the logic '1' state. Such logic can be easily implemented with AND gates as shown in Figure 8.10.1. This is a 4-bit synchronous counter using J -K flip-flop. The timing waveforms are shown in Figure 3.45 which is self-explanatory.



Count sequence of a four bit binary counter:

Count	Q_3	Q_2	Q_1	Q_0	Count	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1



Timing diagram of a 4-bit synchronous up-counter

Figure.3.45 : ***4-bit Synchronous Binary Up-Counter***

From the diagram above, we can see that although the counter is synchronous and is supposed to change simultaneously, we have a propagation delay through the AND gates which add up to give an overall propagation delay which is proportional to the number of bits of the counter. To overcome this problem, we can feed the outputs from the flip-flops directly to a many-input AND gate as follows:

There are many variations to the basic binary counter. The one described above is the binary up counter (counts upwards). Besides the up counter, there is the binary down counter, the binary up/down counter, binary-coded-decimal (BCD) counter etc. Any counter that counts in binary is called a binary counter.

Synchronous *Binary down Counters:*

In a binary up counter, a particular bit, except for the first bit, toggles if all the lower-order bits are 1's. The opposite is true for binary down counters. That is, a particular bit toggles if all the lower-order bits are 0's and the first bit toggles on every pulse.

Taking an example, Q_3 Q2 Q1 Q0 = 0100. On the next count, Q3 Q2 Q1 Q0 = 0011. Q0, the lowest-order bit, is always complemented. Q1 is complemented because all the lower-order positions (Q0 only in this case) are 0's. Q2 is also complemented because all the lower-order positions, Q1 and Q0 are 0's. But Q3 is not complemented the lower-order positions, Q2 Q1 Q0 = 011, do not give an all 0 condition.

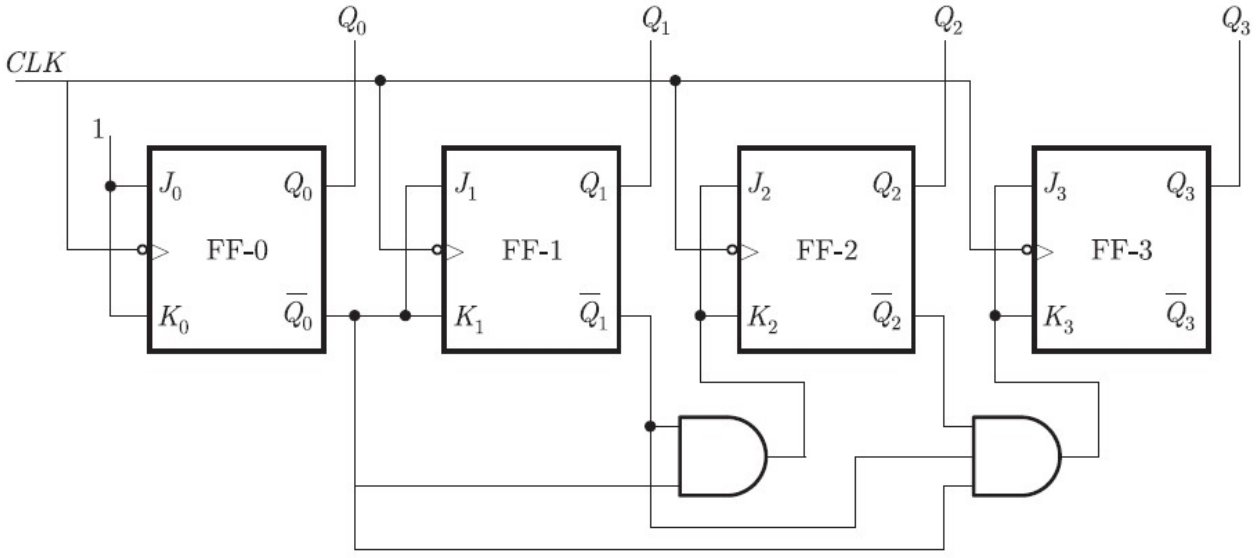


Figure.3.46 : ***4-bit Synchronous Binary Down Counter***

The implementation of a synchronous binary down counter is exactly the same as that of asynchronous binary up counter except that the inverted output from each flip-flop is used. All the methods used improve a binary up counter can be similarly applied here.

Synchronous *Binary Up/Down Counters:*

The similarities between the implementation of a binary up counter and a binary down counter leads to the possibility of a binary up/down counter, which is a binary up counter and a binary down counter combined into one. Since the difference is only in which output of the flip-flop to use, the normal output or the inverted one, we use two AND gates for each flip-flop to "choose" which of the output to use.

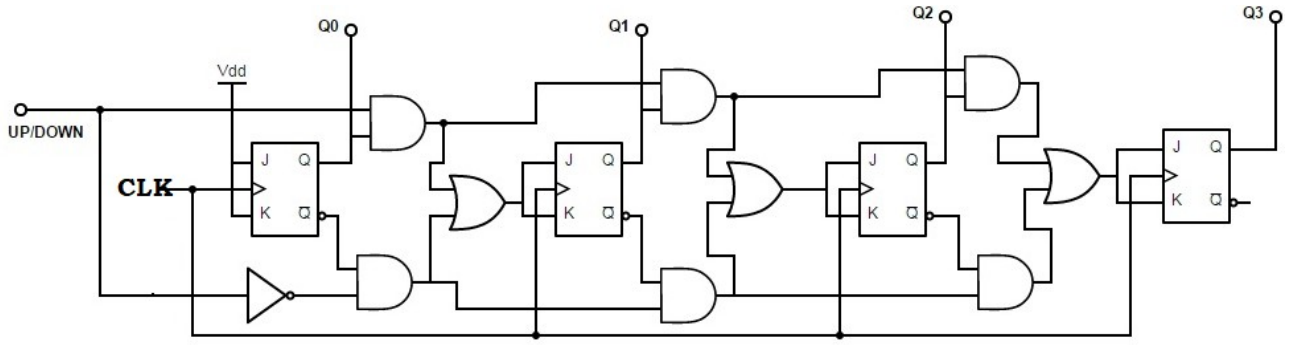


Figure.3.47 : 4-*bit Synchronous Binary Up/Down Counter*

From the diagram, we can see that COUNT-UP and COUNT-DOWN are used as control inputs to determine whether the normal flip-flop outputs or the inverted ones are fed into the J-K inputs of the following flip-flops. If neither is at logic level 1, the counter doesn't count and if both are at logic level 1, all the bits of the counter toggle at every clock pulse. The OR gate allows either of the two outputs which have been enabled to be fed into the next flip-flop. As with the binary up and binary down counter, the speed up techniques apply.

DESIGN OF SYNCHRONOUS COUNTERS:

In this section, we will discuss a commonly used technique to design synchronous counter using *J -K* flip-flop or *D* flip-flop or *T* flip-flop. The design of synchronous counters basically involves designing a suitable combinational logic circuit that takes its inputs from the normal and complemented outputs of the flip-flops used and decodes the different states of the counter to generate the correct logic states for the inputs of the flip-flops such as *J, K, D*, etc.

Design Procedure

The design procedure of synchronous counter is as follows:

1. Number of Flip-flops

Find the number of flip-flops required. For a Mod-*M* counter, the minimum number of flip-flops required is *n*, such that $M \# 2^n$.

2. State Transition Diagram

Draw the state transition diagram showing all possible states. Note that we can also include invalid state in the state transition diagram. If the next state to invalid state is not mentioned, we take it 000.

3. Choice of Flip-flops and Excitation Table

Select the type of the flip-flops to be used and write the excitation table for the counter. An excitation table is a table that lists the present state (PS), the next state (NS) and the required excitations of the flip-flops inputs.

Note that entries for excitations corresponding to invalid states are taken as don't care.

4. Minimal Expression for Flip-flop Inputs

Prepare K-map for each input of flip-flops in terms of the present states of flip-flops. Obtain the minimal expressions for the excitations of the FFs using the K-maps

5. Logic Diagram: Connect the inputs of the flip-flops as per the simplified Boolean equations

Design of a Synchronous MOD-10 (BCD or Decade) Counter:

A BCD or Decade (MOD-10) counter has ten states i.e., 0 to 9. We consider the design of MOD-10 counter using *J* -*K* flip-flops. The steps of design are as follows:

Step 1: Number of Flip-flops

Here, modulus *M* = 10, so 10 # 2^4 and therefore, the number of flipflops required will be 4.

Step 2: State Transition Diagram

The state diagram for the BCD counter is drawn as shown in Figure.

Step 3: Choice of Flip-flops and Excitation Table

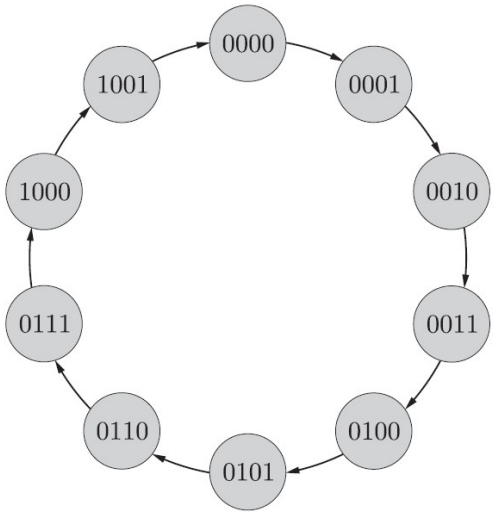
J -*K* flip-flops are selected and the excitation table of a mod-10 counter using *J* -*K* flip-flops is drawn as in Table 8.11.4. Note that six states 1010, 1011, 1100, 1101, 1110 and 1111 are invalid, so the excitation inputs corresponding to these states are taken as don't care and hence these states are not included in state diagram as well as in excitation table.

Excitation table for MOD-10 counter

Present state				Next state				Required excitations							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
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	1	0	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

Step 4: Minimal Expression for Flip-flop Inputs

The K-maps for excitations of flip-flops *J*3 , *K*3 , *J*2, *K*2, *J*1, *K*1, *J*0 and *K*0 in terms of the present state of flip-flops *Q*3 , *Q*2, *Q*1 and *Q*0 can be drawn as shown below.



State diagram of synchronous MOD-10 (BCD) counter

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	0	X	X
01	0	0	X	X
11	0	1	X	X
10	0	0	X	X

$J_3 = Q_2Q_1Q_0$

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	X	X	0
01	X	X	X	1
11	X	X	X	X
10	X	X	X	X

$K_3 = Q_0$

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	X	X	0
01	0	X	X	0
11	1	X	X	X
10	0	X	X	X

$J_2 = Q_1Q_0$

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	0	X	X
01	X	0	X	X
11	X	1	X	X
10	X	0	X	X

$K_2 = Q_1Q_0$

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	1	X	X
10	0	0	X	X

$K_1 = Q_0$

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	0	X	0
01	1	1	X	0
11	X	X	X	
10	X	X	X	X

$J_1 = \overline{Q}_3Q_0$

Step 5: Logic Diagram: Using the above minimized expressions, the logic diagram for the MOD-3 counter can be drawn as shown in Figure.

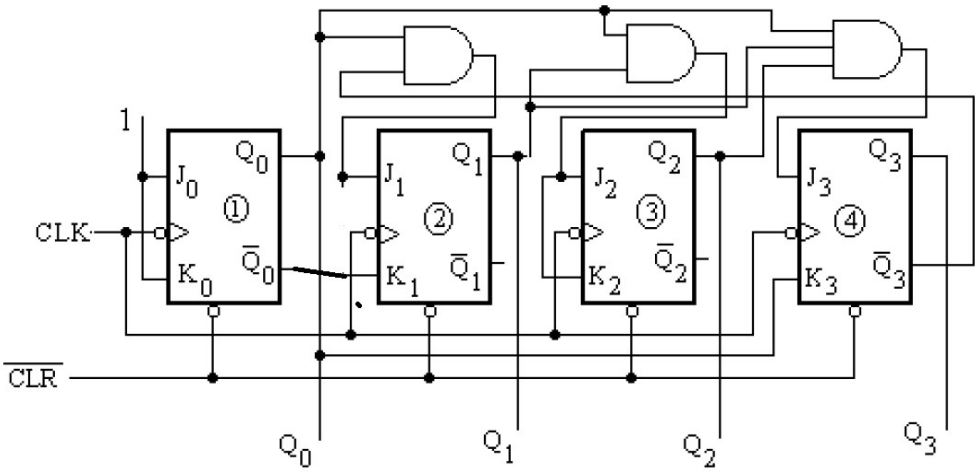


Figure.3.48

Synchronous BCD Counter using T FF:

A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000. Because of the return to 0 after a count of 9, a BCD counter does not have a regular pattern, unlike a straight binary count. The state table of a BCD counter is listed in Table. The input conditions for the T flip-flops are obtained from the present- and next-state conditions. Also shown in the table is an output y, which is equal to 1 when the present state is 1001. In this way, y can enable the count of the next-higher significant decade while the same pulse switches the present decade from 1001 to 0000.

The flip-flop input equations can be simplified by means of maps. The unused states for minterms 10 to 15 are taken as don't-care terms. The simplified functions are

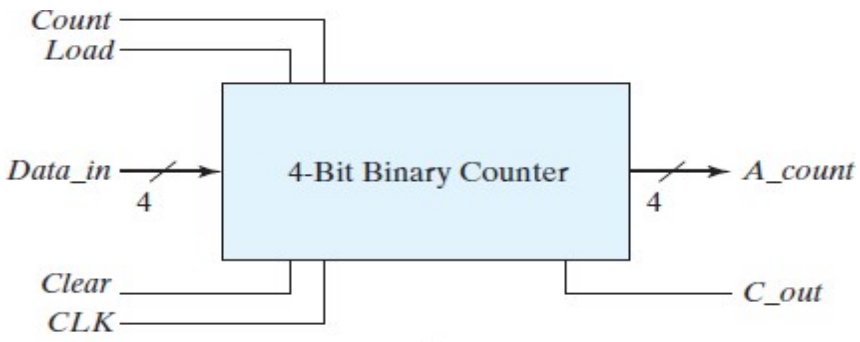
$$TQ1 = 1; \quad TQ2 = Q8Q1; \quad TQ4 = Q2Q1$$
$$TQ8 = Q8Q1 + Q4Q2Q1; \quad y = Q8Q1$$

Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

Binary Counter with Parallel Load:

Counters employed in digital systems quite often require a parallel-load capability for transferring an initial binary number into the counter prior to the count operation. Figure 6.14 shows the top-level block diagram symbol and the logic diagram of a four-bit register that has a parallel load capability and can operate as a counter. When equal to 1, the input load control disables the count operation and causes a transfer of data from the four data inputs into the four flip-flops. If both control inputs are 0, clock pulses do not change the state of the register.

Clear	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	Load inputs
1	↑	0	1	Count next binary state
1	↑	0	0	No change



Ring Counters:

Ring counters are implemented using shift registers. It is essentially a circulating shift register connected so that the last flip-flop shifts its value into the first flip-flop. There is usually only a single 1 circulating in the register, as long as clock pulses are applied.

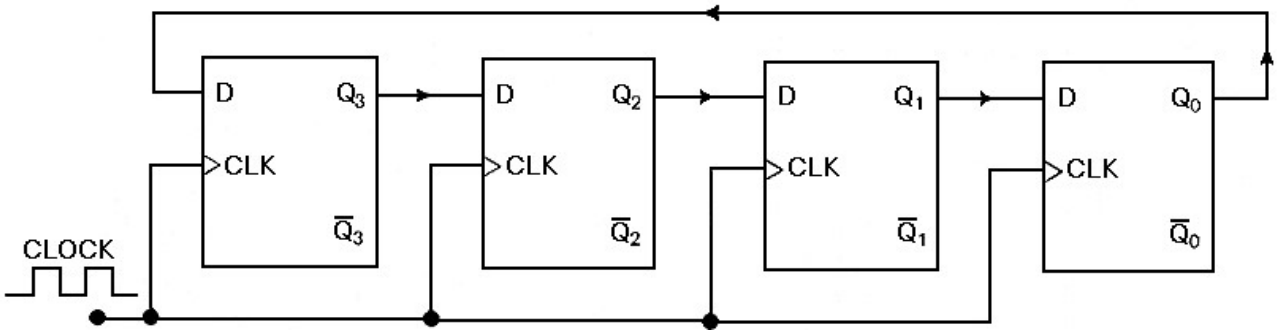
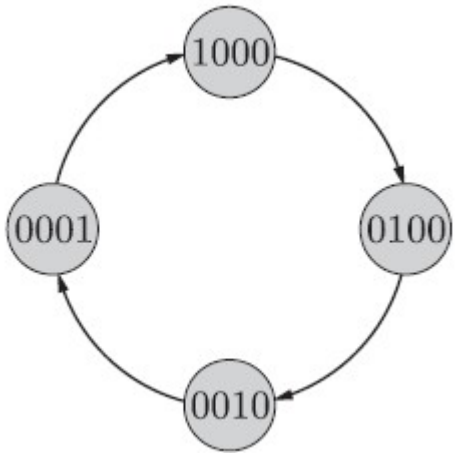
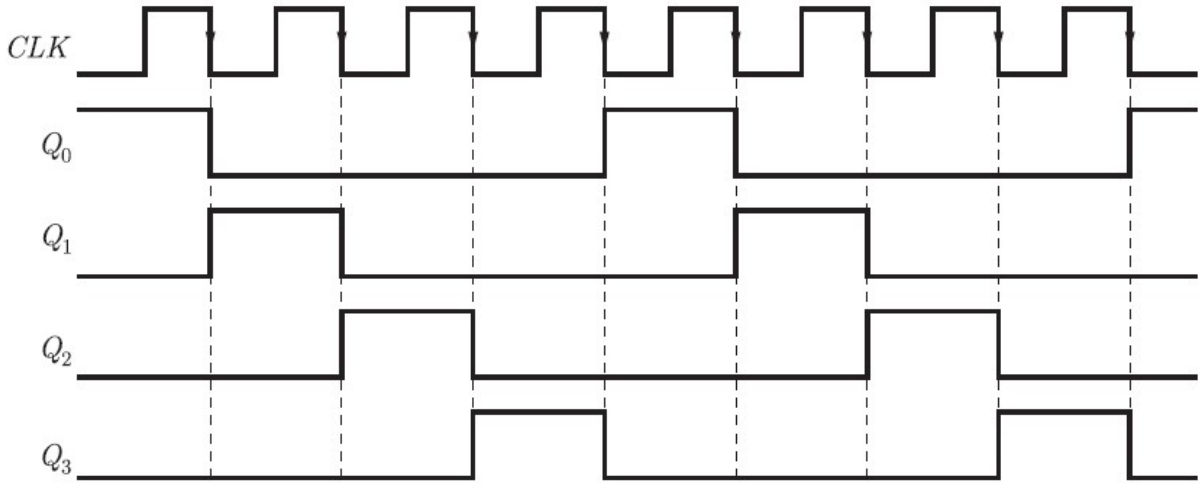


Figure.3.49 :*4-bit Synchronous Ring Counter*

Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	1





Let us assume that flip-flop FF0 is initially set to a '1' and all other flip-flops are reset to '0'. The counter output is therefore 1000. With the first clock pulse, this '1' gets shifted to the second flip-flop output and the counter output becomes 0100. Similarly, with the second and third clock pulses, the counter output will become 0010 and 0001. With the fourth clock pulse, the counter output will again become 1000. The sequence repeats after four clock pulses. From the above discussion we note the following point.

The ring counter above functions as a MOD-4 counter since it has four distinct states and each flip-flop output waveform has a frequency equal to one-fourth of the clock frequency. A ring counter can be constructed for any MOD number. A MOD-N ring counter will require N flip-flops connected in the arrangement as the diagram above.

A ring counter requires more flip-flops than a binary counter for the same MOD number. For example, a MOD-8 ring counter requires 8 flip-flops while a MOD-8 binary counter only requires 3 ($2^3 = 8$).

Johnson/Twisted-Ring Counters:

The Johnson counter, also known as the twisted-ring counter, is exactly the same as the ring counter except that the inverted output of the last flip-flop is connected to the input of the first flip-flop.

The Johnson counter on the other hand is constructed by providing feedback from the inverted output of the last flip-flop to the *D* input of the first flip-flop. The *Q* output of each stage is connected to the *D* input of the next stage, but the *Q* output of the last stage is connected to the *D* input of first stage, therefore, the name twisted ring counter. This type of feedback produces a unique sequence of states. Figure 8.13.4 shows the logic diagram of a basic four-bit Johnson counter using *D* flip-flops.

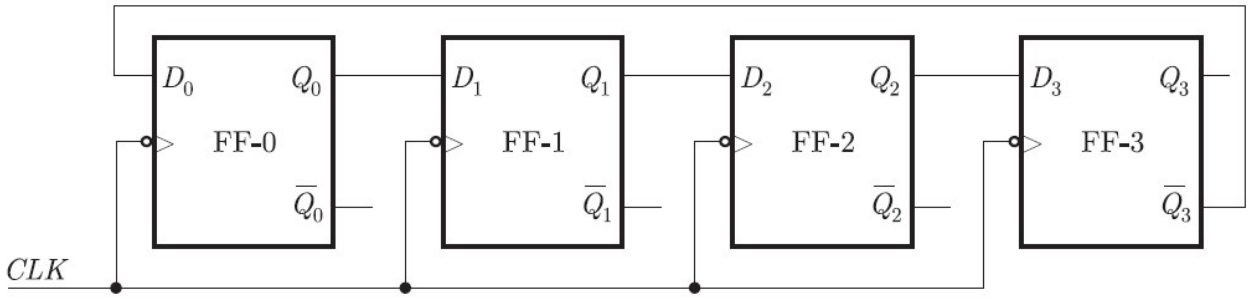
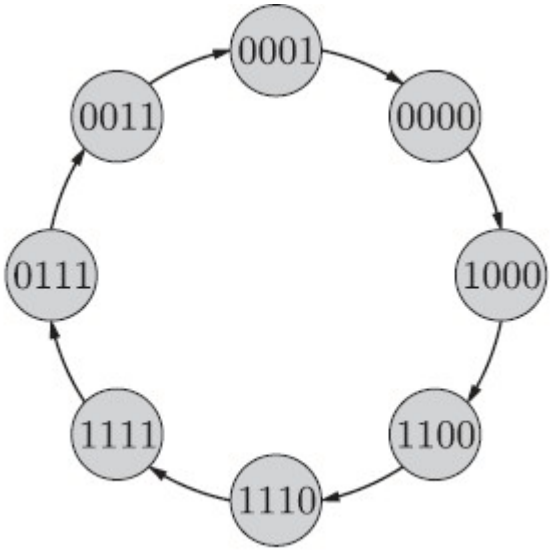


Figure.3.50 :*4-bit Synchronous Johnson Counter*

Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0
9	1	0	0	0

Count Sequence of a 4-bit Johnson Counter



State diagram of Johnson counter

The MOD number of a Johnson counter is twice the number of flip-flops. In the example above, three flip-flops were used to create the MOD-6 Johnson counter. So for a given MOD number, a Johnson counter requires only half the number of flip-flops needed for a ring counter. However, a Johnson counter requires decoding gates whereas a ring counter doesn't. As with the binary counter, one logic gate (AND gate) is required to decode each state, but with the Johnson counter, each gate requires only two inputs, regardless of the number of flip-flops in the counter. Note that we are comparing with the binary counter using the speed up technique discussed above. The reason for this is that for each state, two of the N flip-flops used will be in a unique combination of states. In the example above, the combination $Q_2 = Q_1 = 0$ occurs only once in the counting sequence, at the count of 0. The state 010 does not occur. Thus, an AND gate with inputs (not Q_2) and (not Q_2) can be used to decode for this state. The same characteristic is shared by all the other states in the sequence.

A Johnson counters represent a middle ground between ring counters and binary counters. A Johnson counter requires fewer flip-flops than a ring counter but generally more than a binary counter; it has more decoding circuitry than a ring counter but less than a binary counter. Thus, it sometimes represents a logical choice for certain applications.

Example : Design a synchronous Mod-10 counter to count in the sequence 0, 2, 4,5, 6, 8, 9, 3, 1, 7, 0. Use J K flip-flops to design the counter.

Solution: For the design of this decade counter, four J K flip-flops are required. The state diagram showing the required states in the counter in sequence wise is given in figure.

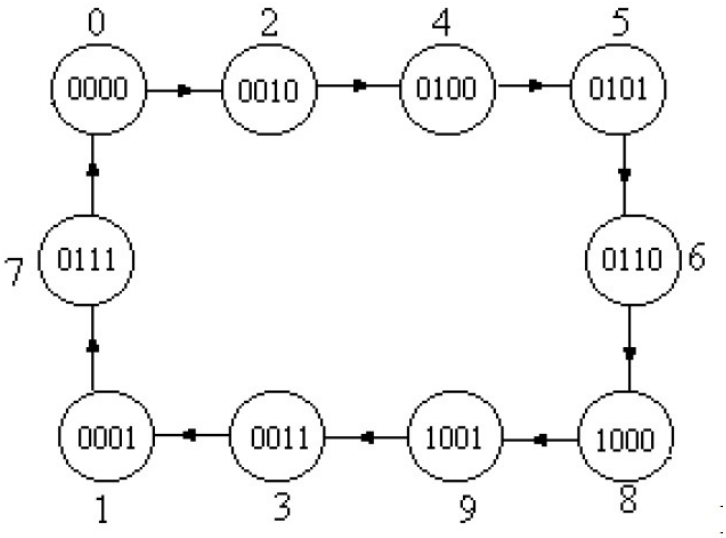
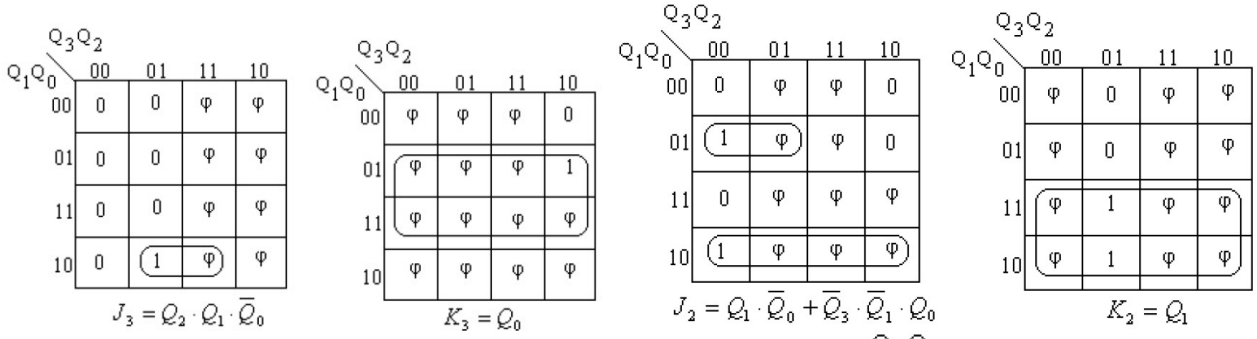
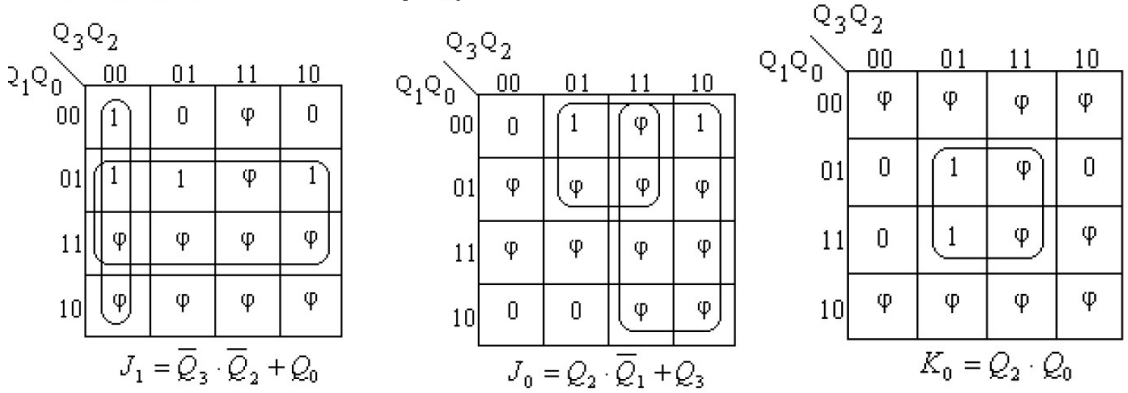


Table shows present states of the counting sequence and next states after the clock pulse and input values of the flip-flops.

The K-maps for all inputs of the flip-flops are drawn as shown in figures through (g) and the expressions for these inputs are given as:

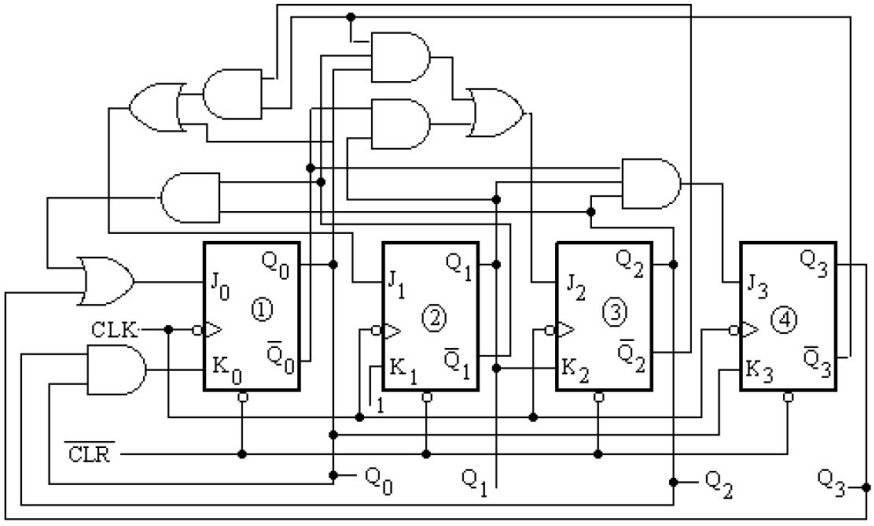
Presnt States				Next States											
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	0	1	0	0	φ	0	φ	1	φ	0	φ
0	0	0	1	0	1	1	1	0	φ	1	φ	1	φ	φ	0
0	0	1	0	0	1	0	0	0	φ	1	φ	φ	1	0	φ
0	0	1	1	0	0	0	1	0	φ	0	φ	φ	1	φ	0
0	1	0	0	0	1	0	1	0	φ	φ	0	0	φ	1	φ
0	1	0	1	0	1	1	0	0	φ	φ	0	1	φ	φ	1
0	1	1	0	1	0	0	0	1	φ	φ	1	φ	1	0	φ
0	1	1	1	0	0	0	0	0	φ	φ	1	φ	1	φ	1
1	0	0	0	1	0	0	1	φ	0	0	φ	0	φ	1	φ
1	0	0	1	0	0	1	1	φ	1	0	φ	1	φ	φ	0





$J_3 = Q_2 \cdot Q_1 \cdot \overline{Q}_0$, $J_1 = \overline{Q}_3 \cdot \overline{Q}_2 + Q_0$, $K_3 = Q_0$ $K_1 = 1$ (directly from the table)

$J_2 = Q_1 \cdot \overline{Q}_0 + \overline{Q}_3 \cdot \overline{Q}_1 \cdot Q_0$, $J_0 = Q_2 \cdot \overline{Q}_1 + Q_3$, $K_2 = Q_1$ $K_0 = Q_2 \cdot Q_0$



The logic circuit diagram of this synchronous counter, whose outputs will be in the given sequence, is shown in figure.

SUPPLEMENT READING

What makes propagation delay?

[Ans] All *real* devices have some delay associated with transferring an input change to the output. Although a *ideal* gate doesn't have such a delay, any implementation in the real world takes time to do its job. This delay that is due to the signal propagation through the device is called the *propagation delay*.

What is Setup time?

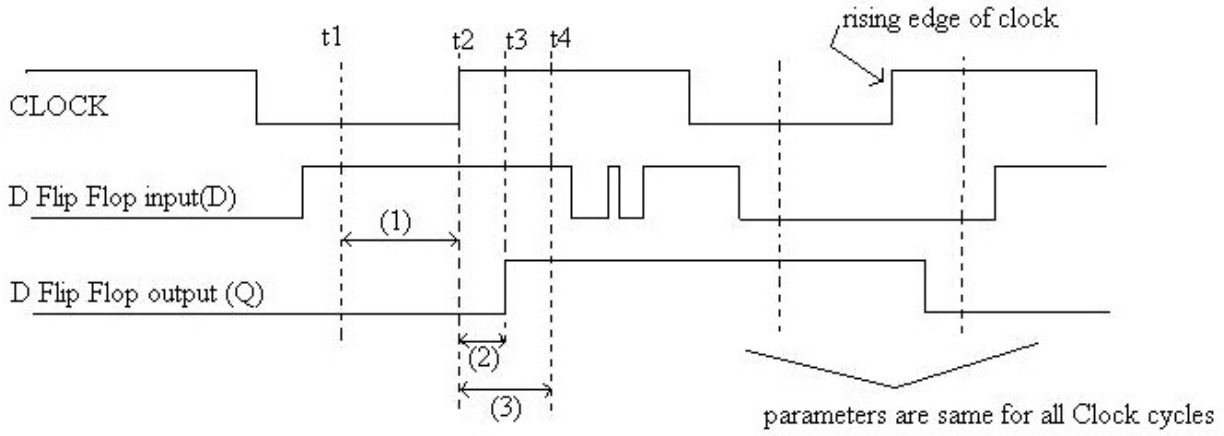
[Ans] Setup time is a timing parameter associated with Sequential Devices (for simplicity henceforth I will be only referring to the Flip Flop). The Setup time is used to meet the minimum pulse width requirement for the first (Master) latch makes up a flip flop is. More simply, the *setup time* is the amount of time that an input signal (to the device) must be stable (unchanging) before the clock ticks in order to guarantee minimum pulse width and thus avoid possible metastability---

What is Hold time?

[Ans] Hold time is also a timing parameter associated with Flip Flops and all other sequential devices. The Hold time is used to further satisfy the minimum pulse width requirement for the first (Master) latch that makes up a flip flop. The input must not change until enough time has passed after the clock tick to guarantee the master latch is fully disabled. More simply, *hold time* is the amount of time that an input signal (to a sequential device) must be stable (unchanging) after the clock tick in order to guarantee minimum pulse width and thus avoid possible metastability.

What do Setup and Hold time look like on a timing diagram?

[Ans] Observe the waveform below:



The timing diagram above illustrates three signals: the Clock, the Flip Flop Input (D) and the Flip Flop output (Q).

(1) is the Setup Time [t2 - t1]: the minimum amount of time Input must be held constant BEFORE the clock tick. Note that D is actually held constant for somewhat longer than the minimum amount. The extra "constant" time is sometimes called the setup margin.

(2) is the Propagation delay of the Flip Flop [t3 - t2]: this is the time that it takes for the new input to be to propagate and influence the output.

(3) is the Hold time [t4 - t2]: the minimum amount of time the Input is held constant AFTER the clock tick. Note that Q is actually held constant for somewhat longer than the minimum amount. The extra "constant" time is sometimes called the hold margin.

(The above timing diagram has 2 clock cycles; the timing parameters for the second cycle will also be similar to that of the first cycle).