GATE CSE NOTES

by

Joyoshish Saha

* **Signal**: A function that represents the variation of a physical quantity with respect to any parameter.

In Electronics, usually signal is variation of electrical quantity (current or voltage) with time.

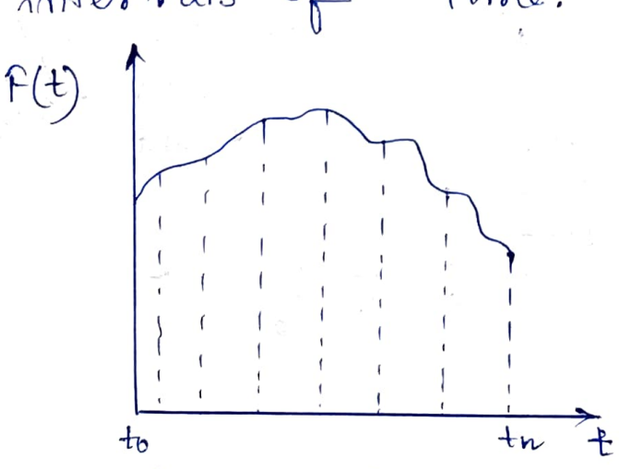
If $dI = 0$, then I is not a signal; it's DC.

* **Transducer**: Used to convert non-electrical signal to electrical signal.

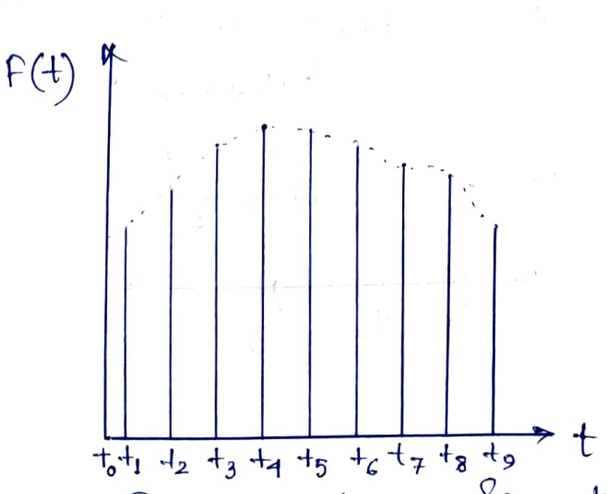
Reverse transducer: Used to convert electrical signal to non-electrical signal.

* **Analog Signal**: A continuous function that is existent for any value of independent variable.

* **Discrete Time Signal**: Signal defined for discrete intervals of time.



Analog Signal



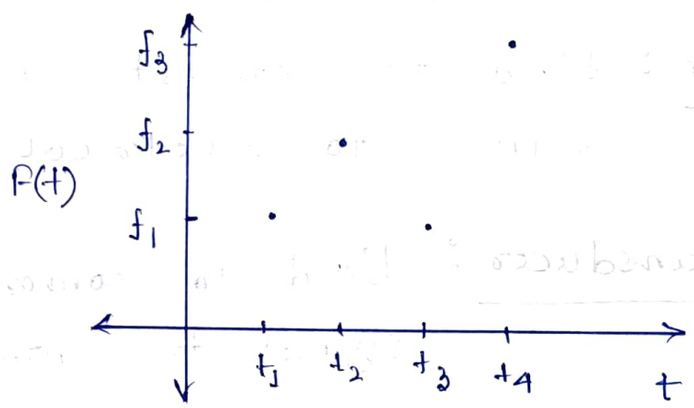
Discrete time Signal.

* Signal types :

i) Discrete time, continuous amplitude
ii) Discrete time, discrete amplitude
iii) Continuous time, continuous amplitude
iv) Continuous time, discrete amplitude.

* **Digital Signal :** Discrete time, discrete amplitude signal.

Signal can take certain values due to discretisation of amplitude.

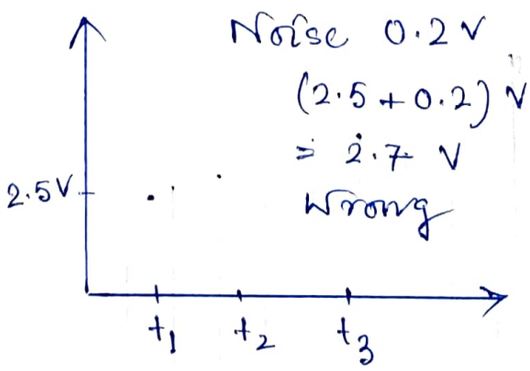


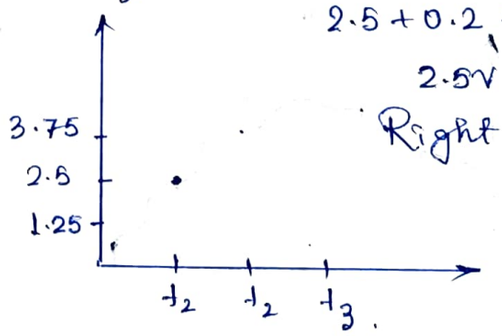
More the number of levels, lesser the error.

* **Need of Digital Signal :** Digital signal is used to minimise the effect of noise.

Transmitter ⟶ Receiver.

(Noise addition)





* Digital System → Sub System → Modules → Basic units (Gates) → Circuits (Transistor, circuitry)

* **Advantages of Digital System:**

i) Noise Immunity
ii) Uses less bandwidth
iii) Encryption.
iv) Efficiency for long distance transmission

* **Switch & Bits:**

1 switch 2 Level
2 switches 4 Levels

- m switches 2^m levels. $[l = 2^m]$.
- bits - 0 & 1. used to represent switch.

* **Boolean Algebra:** Set of rules, used to simplify the given logic expression without changing functionality.

Used when number of variables are less.

For more variables, we use K-Map.

The final simplified result in Boolean Algebra is not always same (drawback).

* Rules:

i) Complement : $(A')' = A$

ii) AND :
$$A \cdot A = A$$
$$A \cdot 0 = 0$$
$$A \cdot 1 = A$$
$$A \cdot A' = 0$$

iii) OR :
$$A + A = A$$
$$A + 0 = A$$
$$A + 1 = 1$$
$$A + A' = 1$$

iv) Distributive :
$$A \cdot (B + C) = A \cdot B + A \cdot C$$
$$* \; A + (B \cdot C) = (A + B) \cdot (A + C).$$

$$\begin{cases} A + A'B = A + B \\ \bar{A} + AB = \bar{A} + B \end{cases}$$

v) Commutative :
$$A + B = B + A$$
$$A \cdot B = B \cdot A$$

vi) Associative :
$$(A \cdot B) \cdot C = A \cdot (B \cdot C).$$

vii) Priority : NOT
AND
OR
↑ More

viii) De Morgan's Law :
$$\overline{(A + B)} = A' \cdot B'$$
$$\overline{(A \cdot B)} = A' + B'$$

Eg. 1. $Y = BAC' + B'AC' + BC'$

$= AC'(B + B') + BC'$

$= AC' \cdot 1 + BC'$

$= AC' + BC'$

$= (A + B)C'$

2. $AB + AB'$

$= A(B + B')$

$= A \cdot 1.$

$= A$

3. $AB + AB'C + AB'C'$

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

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

$= AB + AC + AB'C'$

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

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

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

$= A(B + 1)$

$= A \cdot 1 = A.$

4. $(A + B + C)(A + B' + C)(A + B + C').$

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

$= (A + B + 0)(A + B' + C)$

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

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

$= A + BB' + BC$

$= A + BC.$

5. $(A + B)(A + B')(A' + B)(A' + B')$

$= (A + B \cdot B')(A' + B \cdot B')$

$= A \cdot A'$

$= 0$

* Redundancy Theorem: Consensus theorem

1. Three Variables
2. Each variable repeated twice.
3. One variable is complemented.
4. Take the complemented Variable.

Eg. $Y = AB + A'C + \underline{BC}$ — redundant.

$= AB + A'C.$

Proof: $(AB + A'C + BC \cdot 1)$

$= AB + A'C + BC \cdot (A + A')$

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

$= AB(1 + C) + A'C(1 + B)$

$= AB + A'C.$

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

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

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

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

* Sum of Product:

Expression of product when the function is high (1).

Minterms:	A	B	C	F
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	1
m_3	0	1	1	0
m_4	1	0	0	1
m_5	1	0	1	1
m_6	1	1	0	1
m_7	1	1	1	1

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

[SOP Form].

Canonical/Standard SOP Form (derived from Truth table).

- Minterm – $A\bar{B}C$, $\bar{A}\bar{B}C$ etc.

$$F(A,B,C) = m_2 + m_4 + m_5 + m_6 + m_7$$

$$= \Sigma m\,(2,4,5,6,7)$$

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

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

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

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

$F = A + B\bar{C}$. [Minimal SOP Form].

- Canonical form : Each minterm is having all the variables in normal or complemented form.
- Minimal form : Each minterm does not have all the variables in normal or complemented form.

Eg.

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

minimize SOP expression.

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

$\boxed{Y = B}$

Eg. $Y(A,B) = \Sigma m\,(0,2,3)$.

$Y = m_0 + m_2 + m_3$

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

$= \bar{B} + AB$

$= A + \bar{B}$

* Product of Sum (POS Form):

- Used when the output is low (0).

maxterms	A	B	C	Y
M_0	0	0	0	0
M_1	0	0	1	0
M_2	0	1	0	1
M_3	0	1	1	0
M_4	1	0	0	1
M_5	1	0	1	1
M_6	1	1	0	1
M_7	1	1	1	1

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

$0 \rightarrow A$	$0 \rightarrow \bar{A}$
$1 \rightarrow \bar{A}$	$1 \rightarrow A$
POS	SOP

- maxterm – $A+B+C$, $A+B+\bar{C}$ etc.

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

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

By de Morgan's Law,

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

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

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

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

$= A + B\bar{C}$

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

Eg. Minimise POS expression.

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

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

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

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

$= \bar{B}.$

$Y = \Pi\ (M_1, M_2)$.

$= \Pi M(1,3)$.: Maxterm

$Y = \Sigma m\ (0,2)$ Minterm.

- SOP Form — AND

  POS Form — OR

Eg.

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

$Y(A,B,C) = \Sigma m(0,2,3,6,7)$ SOP

$Y(A,B,C) = \Pi M\ (1,4,5)$ POS.

SOP

$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$ canonical

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

POS

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

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

- Minimal to Canonical:

$Y = A + B'C$ SOP

$= A \cdot 1 \cdot 1 + B'C \cdot 1$

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

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

$= ABC + ABC' + AB'C + AB'C' + B'CA + B'CA'$.

Step 1: 3 Variables A, B, C.

Step 2: m_1 A ✓ B ✗ C ✗ m_2 A ✗ B ✓ C ✓

Step 3: $B + \bar{B} = 1$ replacement

$F = (A+B+C')(A'+C)$ **POS**

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

$= (A+B+C')$

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

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

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

i) 3 variables

ii)

Term 1	Term 2
A ✓	A ✓
B ✓	B ✗
C ✓	C ✓

iii) $CC' = 0$ replacement

Eg. $F = A + B'C$.

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

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

$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$.

- Number of logical expressions $= 2^{2^n}$

  $n \rightarrow$ no. of variables.

* Positive & Negative logic.

- **Positive logic.**

  Higher voltage corresponds to 1.

  Lower voltage corresponds to 0.

- **Negative logic.**

  Higher voltage corresponds to 0.

Eg. logic 0 $\rightarrow -5$ V

logic 1 $\rightarrow$ 0 V. } Positive logic.

* Dual form :

- Positive & Negative Logic AND Gate :

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

+ve logic

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

-ve logic

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

0 ⟶ Low
1 ⟶ High
} +ve logic.

1 ⟶ low
0 ⟶ high
} -ve logic

- Positive & Negative Logic OR Gate :

+ve logic.

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

-ve logic

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

{ +ve logic AND ≡ -ve logic OR
{ +ve logic OR ≡ -ve logic AND.

- Dual form is used to convert between +ve & -ve logic.

Eg. $A \cdot B \xrightarrow{\text{Dual}} A + B$.

$C + D \xrightarrow{\text{Dual}} C \cdot D$.

Acquire::{https, http, ftp}::proxy "http://<proxy>:<port>/";
/etc/apt/apt.conf

* Self Dual :

- for any logical expression, two times dual gives the same expression. In self dual expression, one time dual gives the same expression.

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

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

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

$f \equiv (f')'$

Eg. $G = A \cdot B + B \cdot C + A \cdot C$ Self dual expression.

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

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

$= AB + AAC + BC + ACC$.

$= AB + BC + AC$.

$G = G'$ (Self Dual)

- For n variables, there are $2^{2^{n-1}}$ number self duals.

* Complementing :

AND $\leftrightarrow$ OR

$\cdot \leftrightarrow +$

$0 \leftrightarrow 1$

Eg. $f = A\bar{B} + \bar{A}B$

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

} Using de Morgan's.

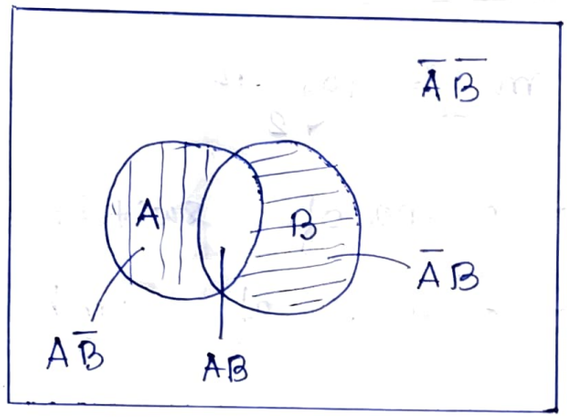
Eg. $G = ABC + \bar{A}BC + A\bar{B}C$.

$$\bar{G} = \overline{(ABC + \bar{A}BC + A\bar{B}C)}$$

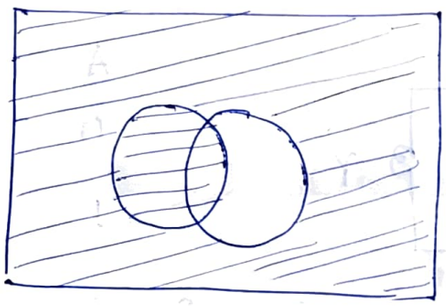
$$= \overline{(ABC + \bar{A}BC)} \cdot \overline{(A\bar{B}C)}$$

$$= (\overline{ABC}) \cdot (\overline{\bar{A}BC}) \cdot (\overline{A\bar{B}C})$$

* Venn Diagram.



Eg.



Minimise the SOP expression for shaded region.

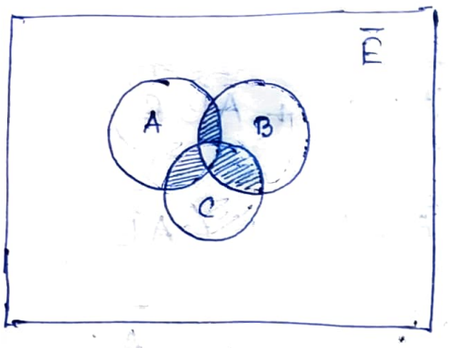
$$Y = \bar{A}\bar{B} + A\bar{B} + AB$$

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

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

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

Eg.



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

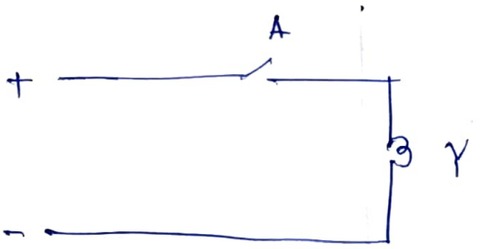
* Switching Circuits. :

- $m = \log_2 n$

$m \longrightarrow$ no. of switches required

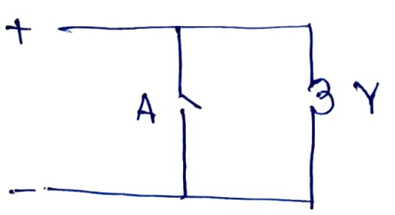
$n \longrightarrow$ no. of combinations.

- Series Switch. (A)



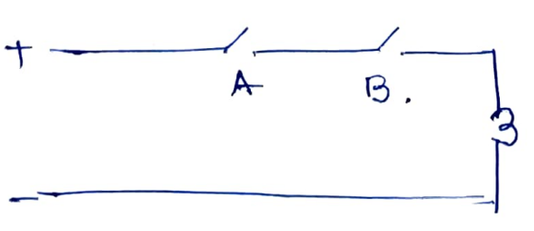
A	Y
0	0
1	1

- Parallel Switch. (NOT)

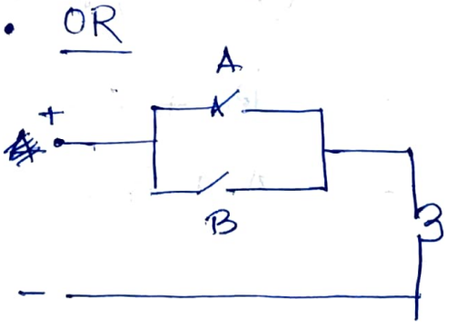


A	Y
0	1
1	0

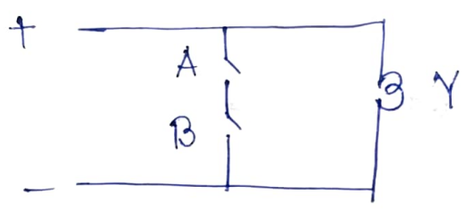
- AND



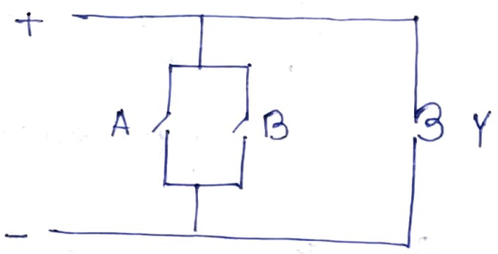
- OR



• NAND – $Y = \overline{AB}$



• NOR $Y = \overline{A+B}$

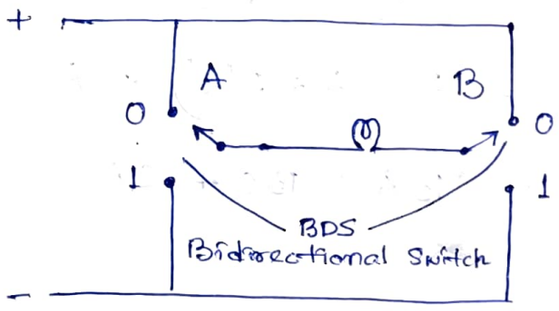


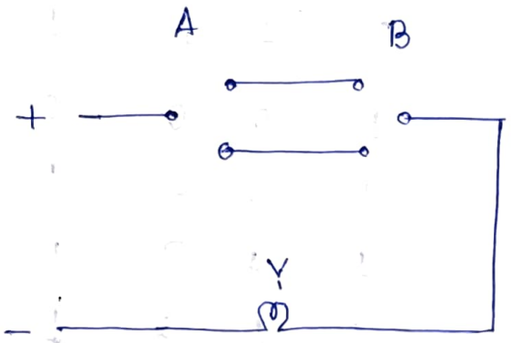
• XOR $Y = AB' + A'B$

$= A \oplus B$

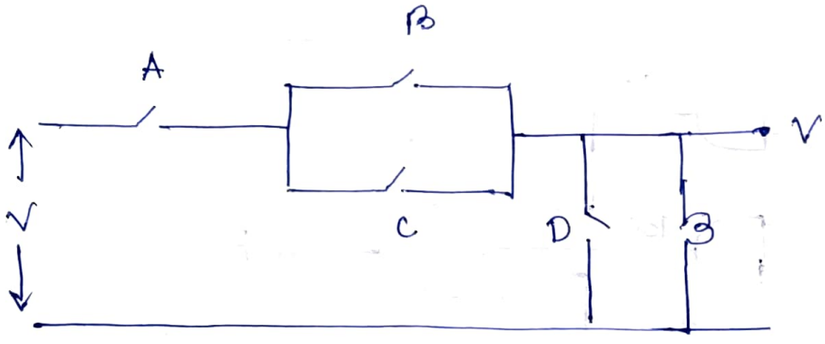
• XNOR $Y = AB + A'B'$

$= A \odot B$





Eg.



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

(Intuitional).

$Y = AB\bar{D} + AC\bar{D}$

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

Find ways to glow the bulb & OR them, then simplify.

* Statement Problems.

Eg. High when i) B, C true ii) A, C false iii) A, B, C true iv) A, B, C false.

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

$= BC + A'C'$

Eg. A, B, C i/p. O/p high when majority of i/p are 1.

Implement circuit.

→

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$F = A'BC + AB'C + ABC' + ABC$

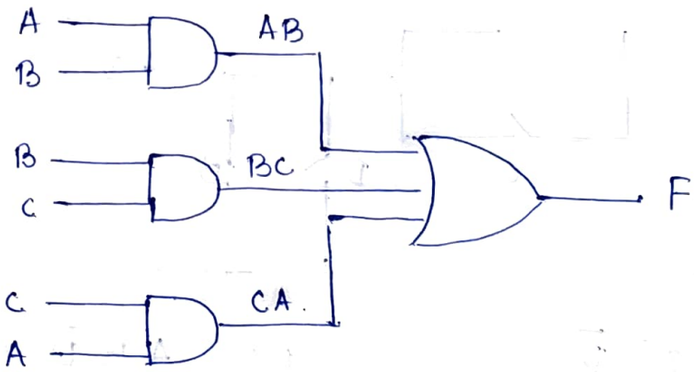
$= A'BC + AB'C + AB$

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

$= A'BC + AB + AC$

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

$= BA + BC + CA.$



* Number Systems

Defines a set of values used to represent quantity.

Name	Base/Radix (r)	(No. of distinct digits)
Binary	2	[0, 1] Bits
Octal	8	
Decimal	10	Digits —
Duodecimal	12	$(0, \ldots, r-1)$.
Hexadecimal	16	

* Binary Number System :

$(101)_2 = (5)_{10}$

- Bits — 0 & 1.
- Any number system — to decimal

ABCDE

$\rightsquigarrow A \times r^4 + B \times r^3 + C \times r^2 + D \times r^1 + E \times r$

AB.CD

$\rightsquigarrow A \times r^1 + B \times r^0 + C \times r^{-1} + D \times r^{-2}$

- MSB (Most Significant Bit)

10101

MSB

- LSB (Least Significant Bit)

10101 — LSB.

- 1 Nibble = 4 bits BCD — Hex

1 Byte = 8 bits

1 word = 16 bits = 2 bytes.

1 double word = 32 bits = 4 bytes

- Decimal → Binary.

Divide integer part by r, base & multiply fractional part by r.

Eg. $(13)_{10} \equiv (1101)_2$

Divider	Dividend	Rem.
2	13	1
2	6	0 ↑
2	3	1
2	1	1
	0	

Eg. $(25.625)_{10} \equiv 11001.101$

$0.625 \times 2 = 1.25$

$0.25 \times 2 = 0.50$

$0.50 \times 2 = 1.00$

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

* Decimal ⟶ Octal.

Eg. $(112)_{10} \longrightarrow (160)_8$

8	112	0
8	14	6
8	1	1
	0	

Eg. $(25.625)_{10} \longrightarrow (31.5)_8$

$0.625 \times 8 = 5.000$

8	25	1
8	3	3
	0	

* Decimal ⟶ Hexadecimal

Eg. $(254)_{10} \rightarrow (FE)_{16}$

16	254	14 E
16	15	15 F
	0	

Eg. $(25.625)_{10} \rightarrow 19.A$

$0.625 \times 16 = 10.00$

A

16	25	9
16	1	1
	0	

* Binary to Decimal.

Generally

$$(a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2})_r \longrightarrow$$

(r → base)

$$a_3 r^3 + a_2 r^2 + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1} + a_{-2} r^{-2}$$

Eg. $(10101.11)_2 \longrightarrow$

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

$$= 16 + 4 + 1 + 0.5 + 0.25$$

$$= (21.75)_{10}$$

* Octal ⟶ Decimal.

Eg. $(57.4)_8 \longrightarrow$

$$5 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1}$$

$$= 40 + 7 + 0.5$$

$$= (47.5)_{10}.$$

* Hexadecimal ⟶ Decimal.

Eg. $(BAD)_{16} \longrightarrow$

$$11 \times 16^2 + 10 \times 16^1 + 13 \times 16^0$$

$$= 2989.$$

* Octal NS. (3 bit bin)

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

Octal → Bin.

$(34.75)_8 \rightarrow$

$(011100.111101)_2$.

Bin → Octal

$(10110.11)_2 \rightarrow (010110.110)_2$

$\rightarrow (26.6)_8$.

Make group of 3 bits.

* Hexadecimal NS. (4 bit bin).

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

Hex → Bin

$(259A)_{16} \rightarrow$

$(0010010110011010)_2$

Bin → Hex

$(10001001.11)_2 \rightarrow (89.C)_{16}$

Make group of 4 bits.

* Hex → Oct
  Oct → Hex

Eg. $(CAD)_{16}$ → Bin

$(110010101101)_2$

→ Oct

$(6255)_8$.

Eg. $(652)_8$ → bin → $(110101010)_2$

→ hex → $(1AA)_{16}$

* Binary Addition.

Eg.
$$\begin{array}{r} 1\ 1\ 0 \\ +\ \ 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 1 \end{array}$$

	Sum	Carry
0+0	0	0
1+0	1	0
0+1	1	0
1+1	0	1

* Binary Subtraction.

borrow 2

Eg.
$$\begin{array}{r} 1\ 1\ 0\ 1\ 1 \\ -\ \ 1\ 0\ 1\ 1\ 0 \\ \hline 0\ 0\ 1\ 0\ 1 \end{array}$$

$$\begin{array}{r} 1\ 1\ 1\ 0 \\ -\ \ \ 1\ 1\ 1 \\ \hline 0\ 1\ 1\ 1 \end{array}$$

* Bin Multiplication.

	Product
0×0	0
1×0/0×1	0
1×1	1.

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ \times\ \ \ 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 0 \\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 0 \\ \hline 1\ 1\ 0\ 0\ 1\ 0 \end{array}$$

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

* Binary Division.

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

* Complements.

1. r's Complement (Radix Complement).

r's complement $\rightarrow$ $r^n - N$

r - base, n - no. of digits.

N - given number

Eg. N = 5690.

10's complement = $10^4 - 5690$

= 4310.

Eg. N = 1101

2's complement = $2^4 - 1101$

= $(16)_{10} - 1101$

= $10000 - 1101$

= 00011

2. $(r-1)$'s complement (Diminished radix complement).

$(r-1)$'s comp. $= r^n - N - 1$.

$= r$'s comp. $- 1$.

No borrow operation is involved.

Eg. 7's comp. of $(5674)_8$

$= 8^4 - 5674 - 1$.

$= (4096)_{10} - 5674 - 1$

$= 10000 - 5674 - 1$

$= 7777 - 5674$

$= 2103$

$$\begin{array}{r} 7777 \\ -\ 5674 \\ \hline 2103 \end{array}$$

Eg. 8's comp. of $(5674)_8$

$= 2103 + 1$

$= 2104$.

Eg. 1's complement of 1101.

$$\begin{array}{r} 1111 \\ -\ 1101 \\ \hline 0010 \end{array} \longrightarrow \text{Ans}$$

2's comp. $= 0010 + 1$

$= 0011$.

Eg. 1's complement of $(1010)_2$.

$$\begin{array}{r} 1111 \\ -\ 1010 \\ \hline 0101 \end{array} \text{ — Ans.}$$

- Shortcut for 2's complement:

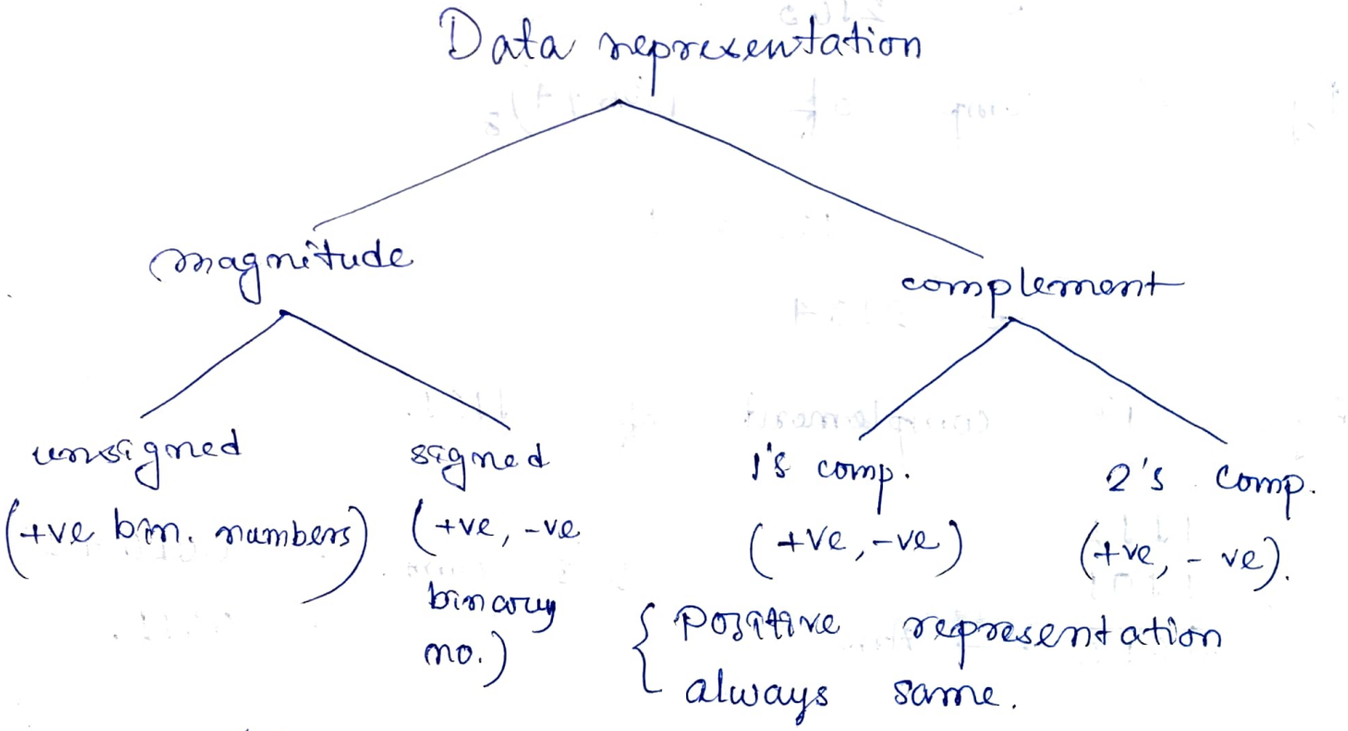
Write given number → starting from LSB copy all zeroes till first 1 → Copy first 1 → Complement all remaining bits.

Eg. 10111000 ← LSB — Step 1

01001000 — Step 2

result — Step 1.

* Data Representation using Signed Magnitude:



- Unsigned

110 → +6

-6 → can't represent

- Signed

+6 → 0110

-6 → 1110

Sign bit (MSB)

Sign bit	Number
0	Positive
1	Negative

} Signed Magnitude Representation.

- Range (Signed Magnitude) $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

$n \rightarrow$ no. of variables

- 1's complement.

$+6 \quad 0110$

$-6 \quad 1001$ ← 1's complement.

$+0 = 0000$ (+ve zero)

$-0 = 1111$ (-ve zero).

Range. $-(2^{n-1}-1)$ to $(2^{n-1}-1)$.

- 2's complement.

$+6 = 0110$

1's → 1001

$-6 = 1010$ ← +1 ← 2's

Range -2^{n-1} to $(2^{n-1}-1)$

- MSB indicates sign.

(*) Bin. Subtration using 1's comp.:

i) Convert no. to be subtracted to its 1's comp. form.

ii) Preform addition.

** iii) If final carry is 1, then add it to the result. If 0, result is negative & in the 1's complement form.

Sign bit	Number
0	Positive
1	Negative

} Signed Magnitude Representation.

- Range (Signed Magnitude) $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

$n \rightarrow$ no. of variables

- 1's complement.

$+6 = 0110$

$-6 = 1001$ ← 1's complement.

$+0 = 0000$ (+ve zero)

$-0 = 1111$ (-ve zero).

Range. $-(2^{n-1}-1)$ to $(2^{n-1}-1)$.

- 2's complement.

$+6 = 0110$ → (1's) 1001 → (+1, 2's) $-6 = 1010$

Range -2^{n-1} to $(2^{n-1}-1)$

- MSB indicates sign.

(*) Bin. Subtration using 1's comp.:

i) Convert no. to be subtracted to its 1's comp. form.

ii) Preform addition.

** iii) If final carry is 1, then add it to the result. If 0, result is negative & in the 1's complement form.

Eg. $(1100)_2 - (0101)_2$.

A B

$-B = 1010$.

$A + (-B) = 1100 + 1010 = 1\,0110$

$0110 + 1 = 0111$. (Ans).

Eg. $(0101)_2 - (1100)_2$

A B

$-B = 0011$.

$A + (-B) = 0101 + 0011 = 0\,1000$

-ve.
1's comp.

$A + (-B) = 0111$.

(*) Bm. Subtraction using 2's comp.:

i) Find 2's comp. of the number to be subtracted.

ii) Perform addition.

** iii) If final carry is generated then the result is +ve & in true form. If not generated, then is -ve & in 2's complement form.

Eg. $-(0100)_2 + (1001)_2$

B A

$-B = 1100$.

$1001 + 1100 = 1\,0101$

Result is 0101.

$$\begin{array}{r} 1011 \\ 1 \\ \hline 1100 \end{array}$$

Eg. $(0110)_2 - (1011)_2$

A B

$-B = 0100 + 1$

$= 0101.$

$0110 + 0101 = 1011$

1's ↓

$0101 \xleftarrow[+1]{2's} 0100$

* Condition for Overflow (2)

x & y are sign bits of two numbers

z is sign bit of result

$\bar{x}\bar{y}z + xy\bar{z} = 0$ (No overflow)

$= 1$ (Overflow).

* Codes. (group of symbols).

Classification

- Weighted (Binary, 2421, 8421)
- Non-weighted (XS-3, gray)
- Reflective code/ self-complementing (2421, XS-3)
- Sequential (8421, XS-3)
- Alpha-numeric (ASCII)
- Error detecting & correcting codes. (Hamming code).

* Binary Coded Decimal (BCD Code):

- Each decimal digit is represented by a 4-bit binary number.
- Positional weights — 8 4 2 1
- Also called 8421 code.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Decimal No. → BCD.

i) $(17)_{10} \longrightarrow (00010111)$

BCD → Decimal No.

i) $(10100) \longrightarrow (14)_{10}$

(00010100)

- Packed BCD – BCD representation for dec. No. > 9.
- BCD is less efficient than Binary as it uses more no. of bits.
- BCD Addition.

1. Sum ≤ 9, Final carry = 0 — Correct ans.
2. Sum ≤ 9, " " = 1 — Add (0110) to ans.
3. Sum > 9, " " = 0 — Add (0110) to ans.

Eg. $(2)_{10} + (6)_{10}$

0010 + 0110

= 1000 ✓ Sum < 9, FC = 0.

(Ans)

Eg. $(3)_{10} + (7)_{10}$

$\downarrow \quad \downarrow$

0011 0111

$=$ 1010 (Sum) > 9.

$\downarrow + 0110$

000 10000 BCD

$\downarrow$

$(10)_{10}$

Eg. $(8)_{10} + (9)_{10}$

$\downarrow \quad \downarrow$

1000 1001

$=$ 1 0001

FC | Sum < 9

$\downarrow + 0110$

10111 BCD

$\downarrow$

$(17)_{10}$

Eg. $(57)_{10} + (26)_{10}$

$\downarrow \quad \downarrow$

01010111 0010 0110

$=$ 0 1 1 1 1 0 1 1

7 $<$ 9 13 $>$ 9

0111 1101 ⟶ Check every 4 bits individually & check conditions

+ 0110 & perform :-

1000 0011 BCD

$\downarrow$

$(83)_{10}$.

* Shift ADD-3 Method. (Bin. ⟶ BCD).

$(15)_{10}$

Operations	Tens	Ones	Decimal
Original No.			1111
Shift		1	111
Shift		11	11
Shift		111	1
Add-3		+ 011	
		1010	1
Shift		10101	

BCD for 15.

* 2421 Code (BCD.) Self-Complementing

Decimal Digit	Set I 2*4 2 1	Set II 2*4 2 1 (Self-Complement)
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1
6	0 1 1 0	1 1 0 0
7	0 1 1 1	1 1 0 1
→ 8	1 1 1 0	1 1 1 0
→ 9	1 1 1 1	1 1 1 1

* Excess-3 Code. (XS3) Self-Complementing.

- Decimal → 8-4-2-1 (BCD) —Add 0011 (3)→ Excess-3

Eg. 5 → 0101 —+0011→ 1000

- Unweighted Code., 4-bit Code
- Decimal

	BCD		XS3
0	0000	+0011	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

Eg. $(24)_{10} \longrightarrow$ (BCD) (00100100) $\xrightarrow{\text{Add } 0011}$ $00100100 + 00110011$

01010111.

Ans.

Eg. $(658)_{10} \longrightarrow$ (BCD) $(0110\ 0101\ 1000)$

$\downarrow +0011$

$1001\ 1000\ 1011$

- XS-3 is only unweighted code that is self-complementing.

* Whether self complementing or not — [For weighted Codes]

Weights. — w_1, w_2, w_3, w_4.

$w_1 + w_2 + w_3 + w_4 = 9$ ~ Self Complementing

$\neq 9$ ~ Not "

* Excess-3 Code Addition:

Eg. $(2)_{10} + (5)_{10}$.

BCD: 0010 $\downarrow$ 0101.

XS3 $+0011$: 0101 $\downarrow$ 1000

0101 — XS3

$+\ 1000$ — XS3

1101 — XS6.

$-\ 0011$ — Subtract 3.

1010 — (Ans)

** Eg. $(27)_{10} + (39)_{10}$

BCD: $0010\ 0111$ $\downarrow 0011$ | $0011\ 1001$ $\downarrow$

XS3 $+0011$: $0101\ 1010$ | $0110\ 1100$

$0101\ 1010$

$+\ 0110\ 1100$

$1100\ 0110$

$-0011 + 0011$

$1001\ 1001$ Ans.

* { FC = 1, add 3; FC = 0, sub. 3

* Gray Code (Frank Gray, AT&T).

- Reflected binary code (RBC).
  Unweighted, cyclic code
- Unit distance code & minimum error code.
- Two successive values differ in only one bit
- Binary no. converted to GC to reduce switching operation.

Decimal	Binary	GC.
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000
16	10000	0000

- **Binary → GC.** (MSB remains same)

Step 1. Record MSB as it is.

Step 2. Add MSB to the next bit, record the sum, neglect carry (XOR)

Step 3. Repeat the process.

Eg. $(1011)_{Bin}$

1 0 1 1
↘ + ↘ + ↘
1 1 1 0 — Ans.

Eg. $(1110)_{Bin}$

1 1 1 0
1 0 0 1 Ans.

- **GC → Binary.**

Step 1. Record MSB as it is.

Step 2. Add MSB to next bit of Gray code, record sum & neglect carry (X-OR)

Step 3. Repeat.

Eg. $(1110)_{gc}$

1 1 1 0
↗ + ↗ + ↗
1 0 1 1 (Ans)

Eg. $(1001)_{gc}$

1 0 0 1
1 1 1 0 (Ans).

* Parity:

→ Single bit error is detected by it.

→ i) Even parity – No. of 1s even

ii) Odd parity – No. of 1s odd.

0100 : 1 — even

1100 : 0 — even

0100 : 0 — odd.

1100 : 1 — odd.

→ Transmit: 0100 : 1 (Even parity) ⟶ Due to noise ⟶ Receive: 0101 : 1 (Odd parity) ✗ Error detected.

* Hamming Code – Error Detection.

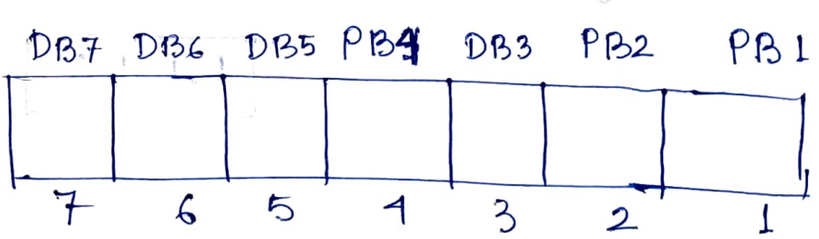
→ Given by R.W. Hamming

→ Easy to implement

→ 7-bit hamming code is used commonly.

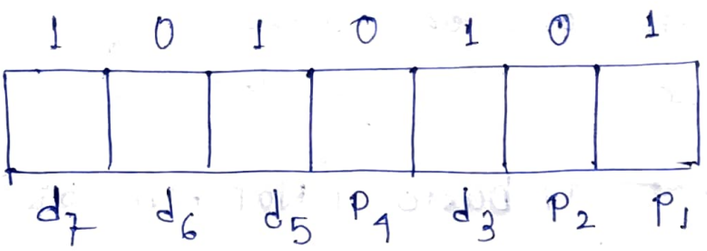
→ Data bits – 4
Parity bits – 3
} for 7-bit

→ Position of parity bits 2^n.



PB1	associated with		DB3, DB5, DB7
PB2	"	"	DB3, DB6, DB7
PB4	"	"	DB5, DB6, DB7

Eg. (1011) Transmitter side



$P_1 \rightarrow d_3, d_5, d_7 \rightarrow \boxed{1}\ 1\ 1\ 1$ even parity

$P_2 \rightarrow \boxed{0}\ 1\ 0\ 1$ $\quad P_4 \rightarrow \boxed{0}\ 1\ 0\ 1$

Error detection –

due to noise →

$d_7 d_6 d_5\ d_3$
1 1 1 0 1 0 1.
$P_4\ P_2\ P_1$

Check $P_1 \rightarrow d_3, d_5, d_7$
even parity ✓

Check $P_2 \rightarrow d_3, d_6, d_7$
✗ error

Check $P_4 \rightarrow d_5, d_6, d_7$
✗ error.

* Hamming Code – Error Correction

Eg. If the 7 bit hamming code word received by a receiver is 1011011, assuming the even parity state whether the received code is correct or wrong ? Locate error bit. [Correct – 1001011]

$d_7\ d_6\ d_5\ P_4\ d_3\ P_2\ P_1$

1	0	1	1	0	1	1

→ $P_1 \rightarrow 1\ (1\ 1\ 0) \times \quad P_1 = 1$

$P_2 \rightarrow 1\ (1\ 0\ 0)\ \checkmark \quad P_2 = 0$

$P_4 \rightarrow 1\ (1\ 0\ 1) \times \quad P_4 = 1$

$(P_4 P_2 P_1) \equiv (101)_2 \equiv (5)_{10}$ — Error

Error bit

Whenever parity unmatches
$P_i = 1$, otherwise
$P_i = 0$.

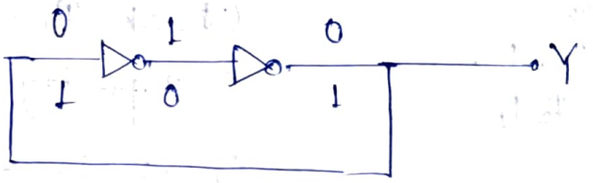
* Logic Gates.

- A physical device that performs logic operation on one or more logical inputs, and produces a single logical output.
- Gates — i) basic (NOT, AND, OR)
  ii) universal (NAND, NOR)
  iii) arithmetic (X-OR, X-NOR)

- **NOT Gate**
  IC 7404

A	Y
1	0
0	1

Eg. 8.



a) buffer
b) astable multivibrator
c) bistable multivibrator
d) square wave generator

buffer

i/p	o/p
0 →	0
1 →	1

Y = 1 if i/p = 1
Y = 0 if i/p = 0.

a) Not a buffer because of the feedback.

astable — not stable on either state, continually changes from one state to another.

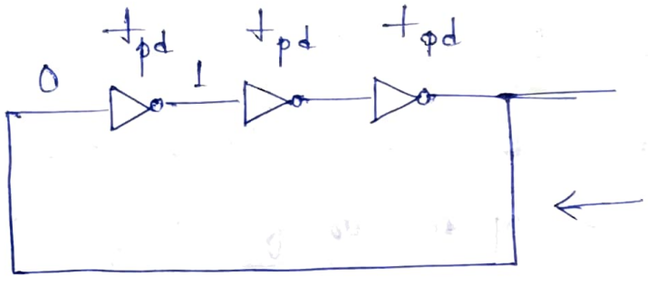
b) Not a stable.

c) bistable — stable on either state. Flipped by an external trigger pulse.

d) Not square wave generator.

Clock generator

Eg.

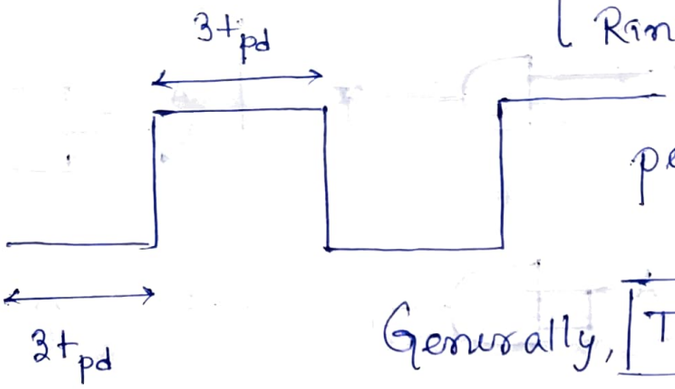


← Square wave generator

t_{pd} → propagation delay for each not gate

{ Astable MV.
{ Ring Oscillator

graph –



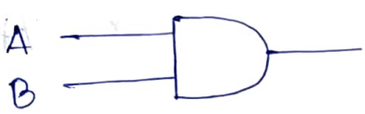
period, $T = 6t_{pd}$.

Generally, $\boxed{T = 2nt_{pd}}$

n → no. of NOT gates.

- **AND Gate** o/p high when all i/p high.

IC 7408



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

→ $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

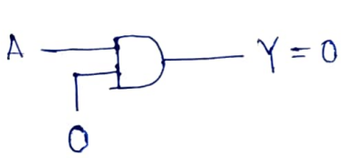
[Associativity]

→ $A \cdot B = B \cdot A$

[Commutativity]

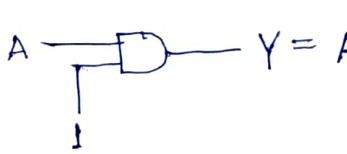
→ Enable & Disable.

→ When A = 0, Y = 0.



0 is disable for AND Gate.

→ When A = 1, Y = 0/1



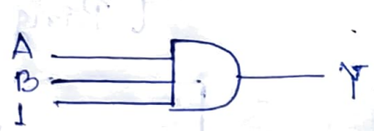
1 is enable (buffer).

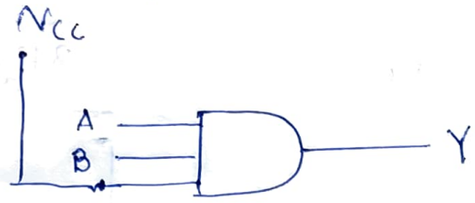
→ Unused I/P.

↝ In TTL logic (Transistor – Transistor) if any i/p is open or floating it will act as 1.

↝ In ECL logic if open, act as 0.

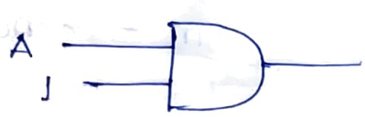
↝ i)





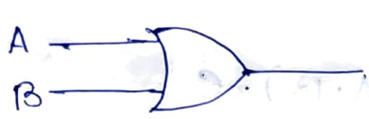
ii)

iii)



TTL logic

OR Gate IC 7432 One of all or all are high. ⟶ O/p high



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

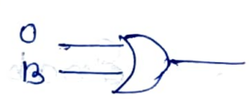
↝ (A+B)+C = A+(B+C)

A+B = B+A.

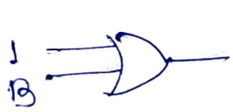
↝ Enable & Disable.

0 ⟶ Enable

1 ⟶ Disable.



B Buffer

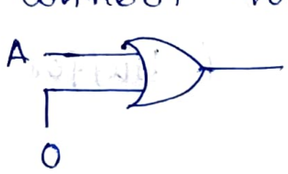


↝ Unused I/P

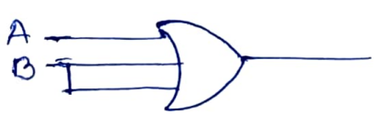
TTL – 1

ECL – 0.

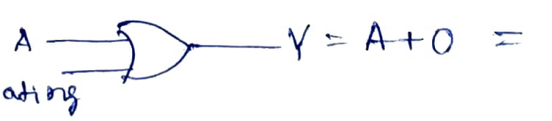
2) Connect to 0



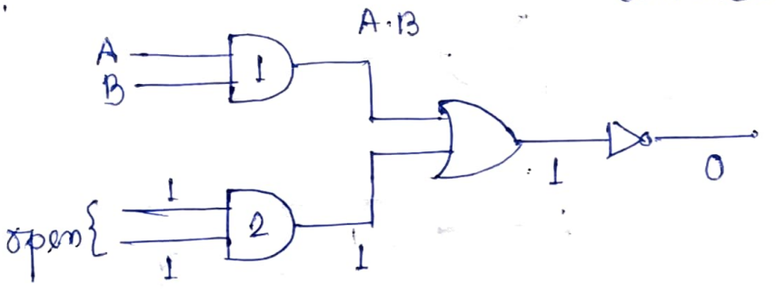
ii)



iii) ECL logic



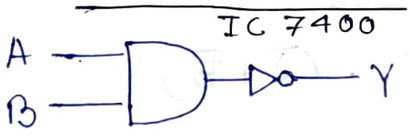
Eg. Consider TTL logic.

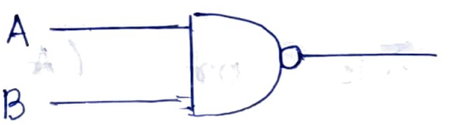


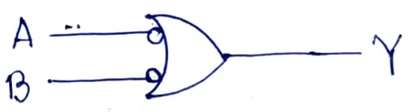
Y = 0. (Ans)

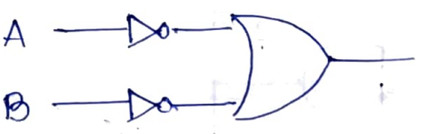
* NAND Gate. $(\overline{A}+\overline{B})$ Bubbled OR

IC 7400









0 → disable

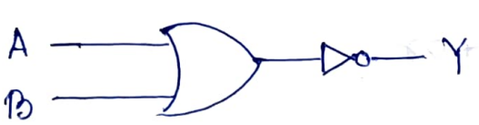
1 → enable

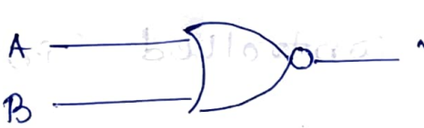
Unused i/p – same as AND.

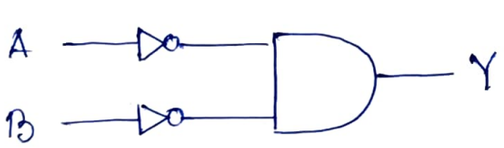
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

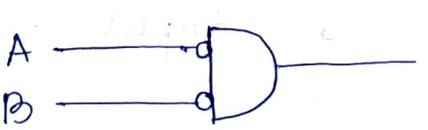
* NOR Gate $(\overline{A}\cdot\overline{B})$ Bubbled AND.

IC 7402









0 → enable

1 → disable.

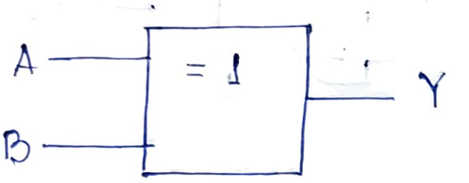
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Unused i/p – same as OR

* XOR Gate.

IC 7486

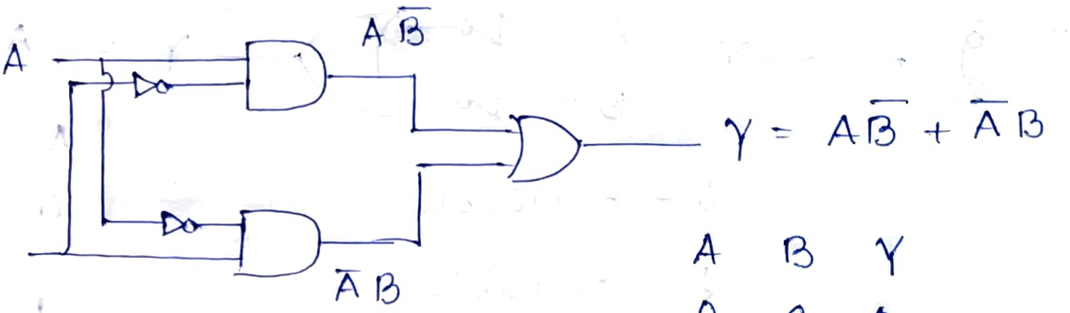
IEEE Symbol -



$Y = A \oplus B$

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

CKt

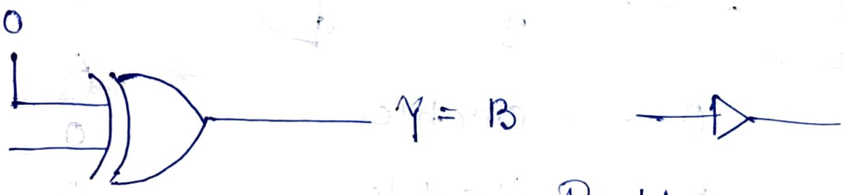


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Enable - 0 buffer
Enable - 1 inverter

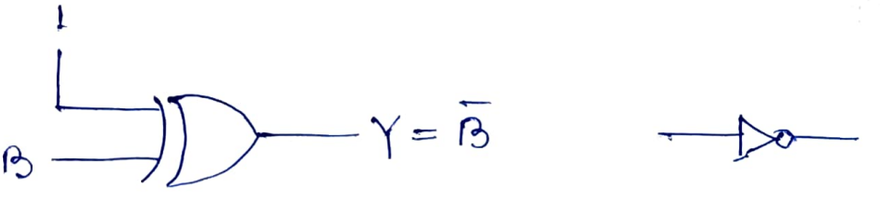
- XOR is odd one's detector.
- XOR - controlled inverter

We can make it buffer or inverter according to input.



Buffer

B	Y
0	0
1	1



Inverter.

B	Y
0	1
1	0

• Properties of XOR :

1. $A \oplus A = 0$

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

   $A \oplus 0 = A$

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

2. If $A \oplus B = C$ then

   $B \oplus C = A$

   $A \oplus C = B$

   $A \oplus B \oplus C = 0$.

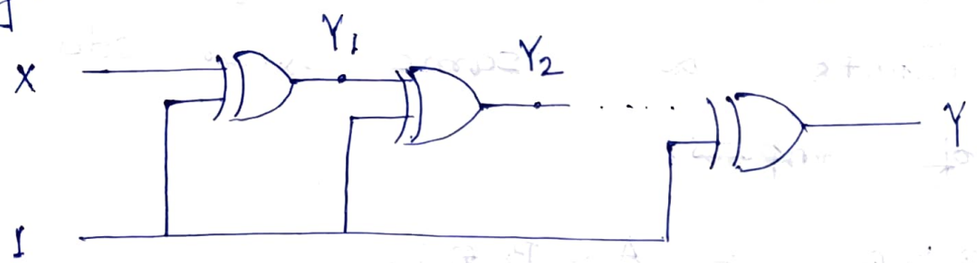
3. $A \oplus A \oplus A = A$

   $A \oplus A \oplus \cdots \oplus \cdots n$ times $=$

   A, $n \rightarrow$ odd

   0, $n \rightarrow$ even.

Eg.



Cascading of 20 XOR Gates.

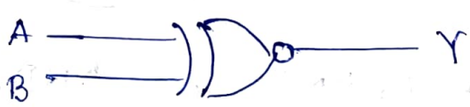
$Y_1 = X \oplus 1 = \bar{X}$

$Y_2 = \bar{X} \oplus 1 = X$

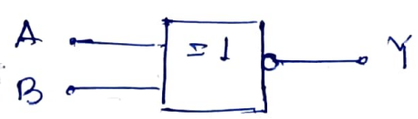
$Y_{20} \Rightarrow Y = X$.

* XNOR Gate. $(A \odot B)$

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



IEEE Symbol



- $A = B$ o/p $= 1$

  $A \neq B$ o/p $= 0$

- XNOR follows commutative, associative law.
- Enable – 0, inverter

  Enable – 1, buffer.

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

- Properties.

i) $A \odot A = 1$

$A \odot \bar{A} = 0$

$A \odot 0 = \bar{A}$

$A \odot 1 = A$.

ii) $A \odot A \odot A = A$.

$A \odot A \odot \ldots$ n times

$= A$, n is odd

$= 1$, n is even

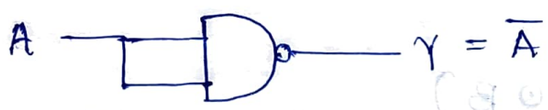
iii) XOR & XNOR are complementary when even inputs & same when odd no. of inputs.

Eg. $A \odot B \odot C = A \oplus B \oplus C$.

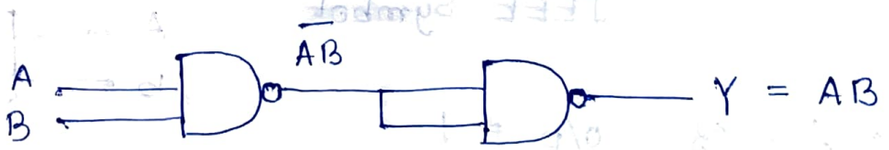
$A \odot B \odot C \odot D = \overline{A \oplus B \oplus C \oplus D}$

* NAND as Universal Gate.

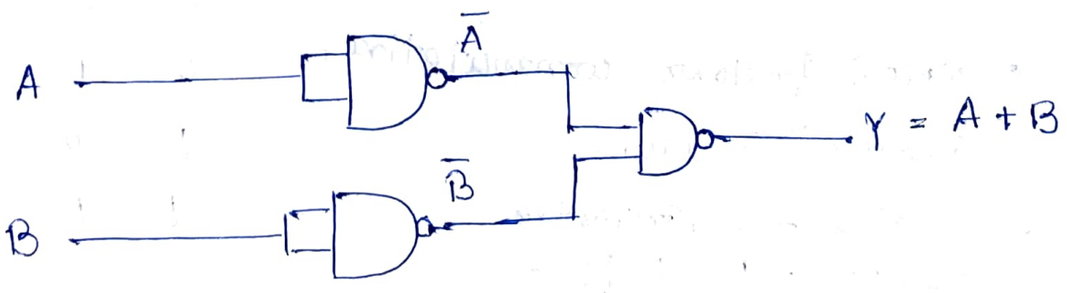
- NAND as NOT. $Y = \overline{AB}$, $Y = \overline{AA} = \bar{A}$



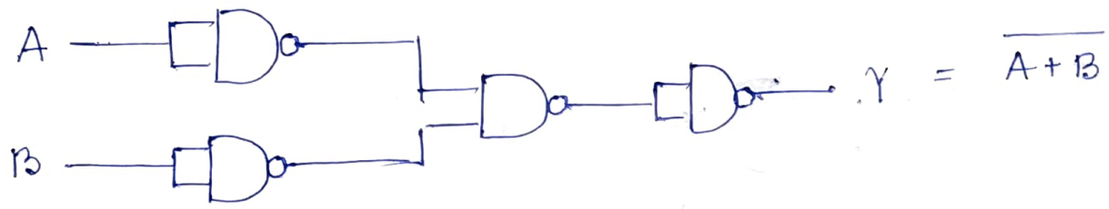
- NAND as AND.



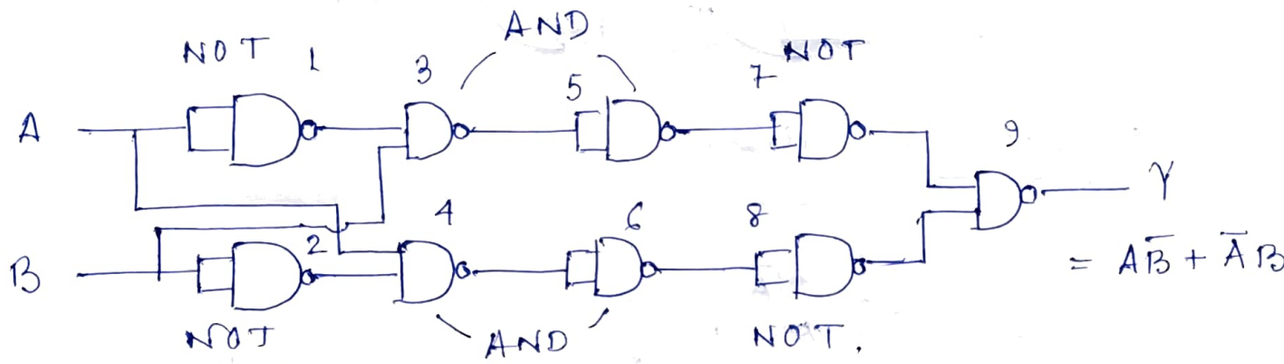
- NAND as OR. $Y = \overline{AB} = \bar{A} + \bar{B}$



• NAND as NOR.



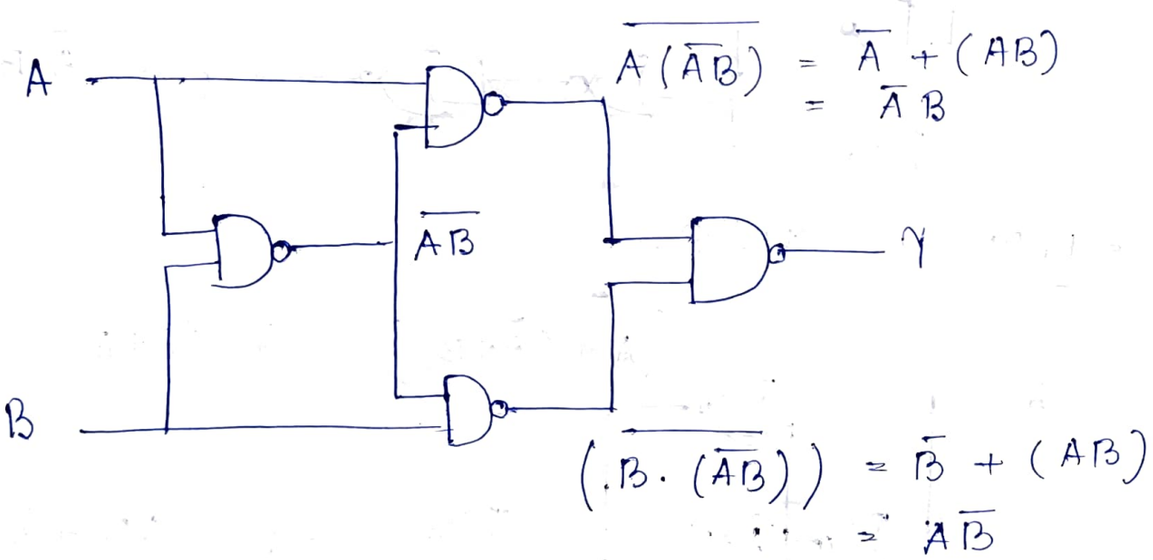
* *
• NAND as XOR. $Y = A\bar{B} + \bar{A}B$.



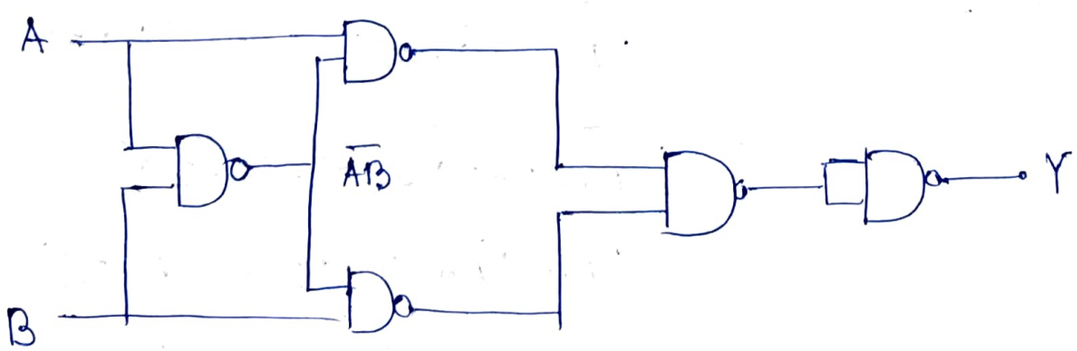
$Y = \left((\bar{A}B) \cdot (A\bar{B}) \right)$ | 5, 7 & 6, 8 not required

III

* * *

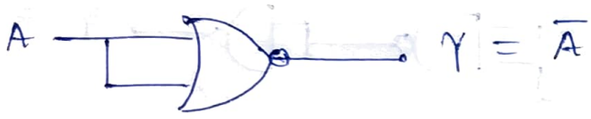


• NAND as XNOR.

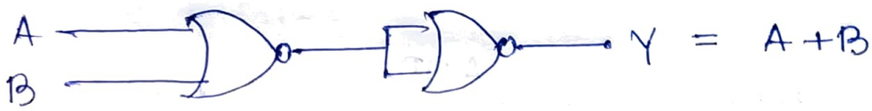


* NOR as Universal Gate.

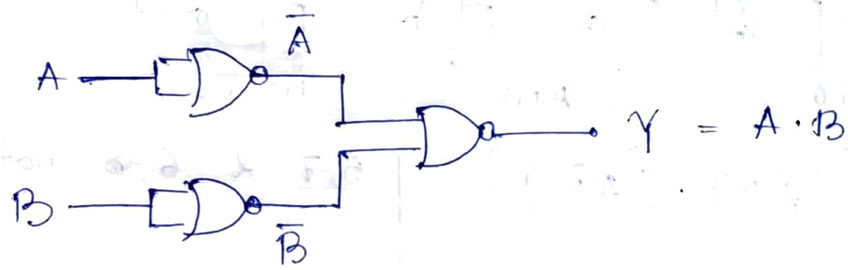
- **NOR as NOT.** $Y = \overline{A+B}$ $\quad Y = \overline{A+A} = \overline{A}$



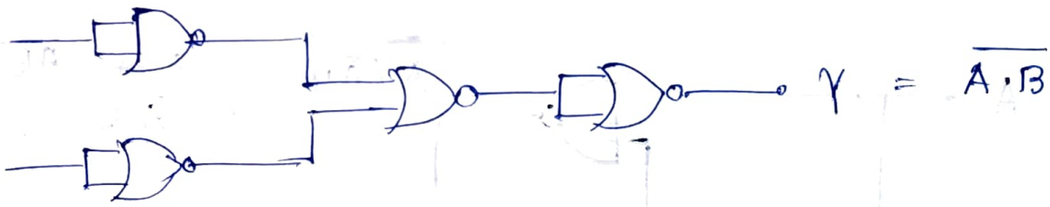
- **NOR as OR.**



- **NOR as AND.**

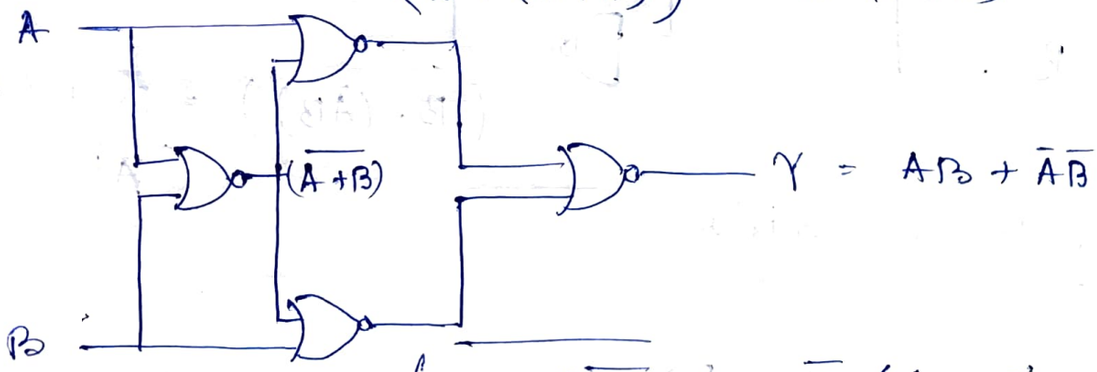


- **NOR as NAND.**



- **NOR as XNOR.**

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



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

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

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

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

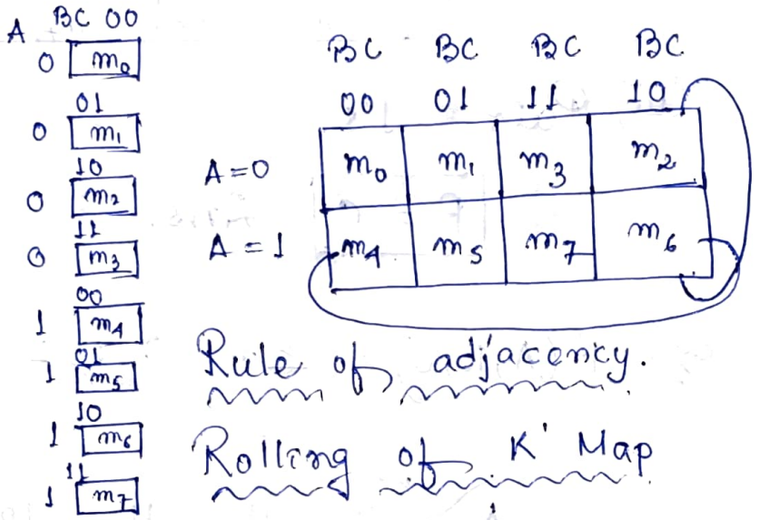
• NOR as XOR.

* Karnaugh Map [K' Map] :

Eg. $f = A + B\bar{C}$

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

In K' Map the adjacent cells are like only one variable should change.



Rule of adjacency.

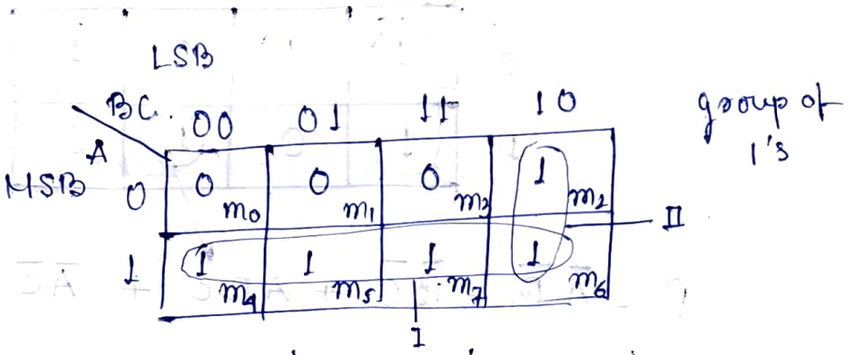
Rolling of K' Map

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

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

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

$= A + B\bar{C}$



• pairing 2 1's, 4 1's, 8 1's, 16 1's

(No diagonal pairing, or adjacency not follows).

Same 1 can be paired more than once.

* Inside

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

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

$= B\bar{C}.$

$f = I + II.$

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

$= A + B\bar{C}$

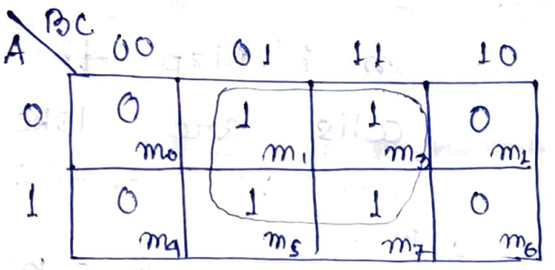
If variable changes → 1
If not → X/$\bar{X}$.

Eg. $f(A,B,C) = \Sigma m\,(1,3,5,7)$.

A → MSB, C → LSB, Σm → SOP.

(i) Find out no. of variables : 3.

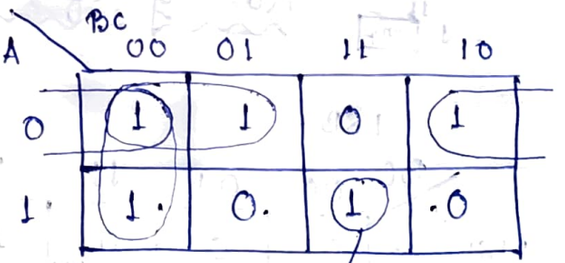
(ii) No. of cells in K' Map : $2^3 = 8$.



(iii) Pair 1's.

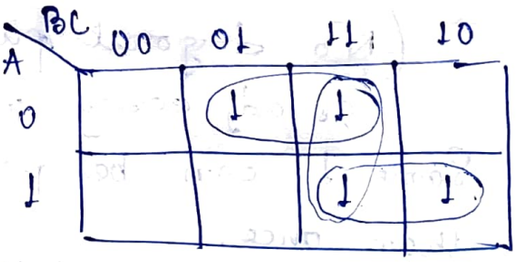
$\boxed{F = C}$ Ans.

Eg. $F(A,B,C) = \Sigma m\,(0,1,2,4,7)$.



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

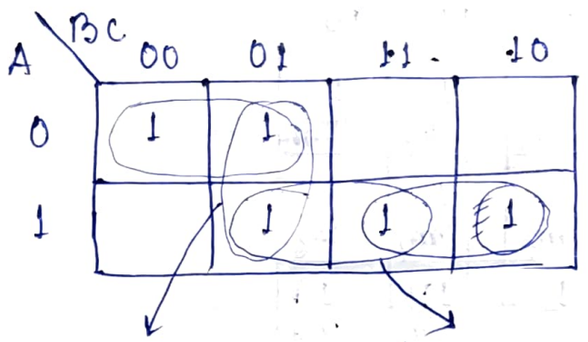
Eg. $F(A,B,C) = \Sigma m\,(1,3,6,7)$.



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

$= AB + \bar{A}C$ [by Redundancy Theorem]

Eg. $F(A, B, C) = \Sigma m(0, 1, 5, 6, 7)$.



$F = \bar{A}\bar{B} + \bar{B}C + AB$ | $F = \bar{A}\bar{B} + AC + AB$

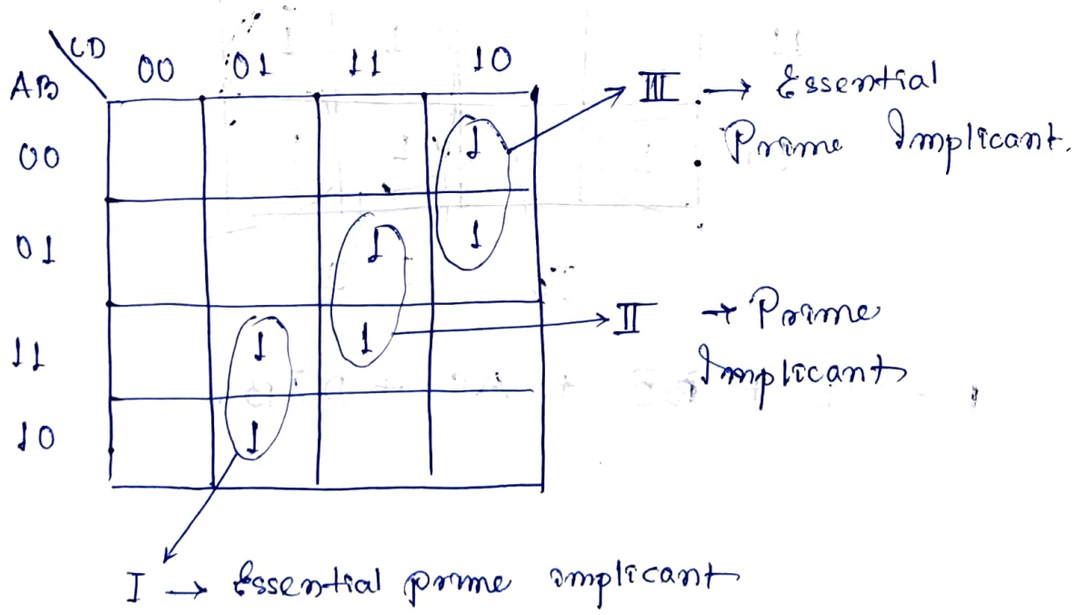
Result is minimum, but may not be unique.

- Implicants: Group of 1's in K' Map.

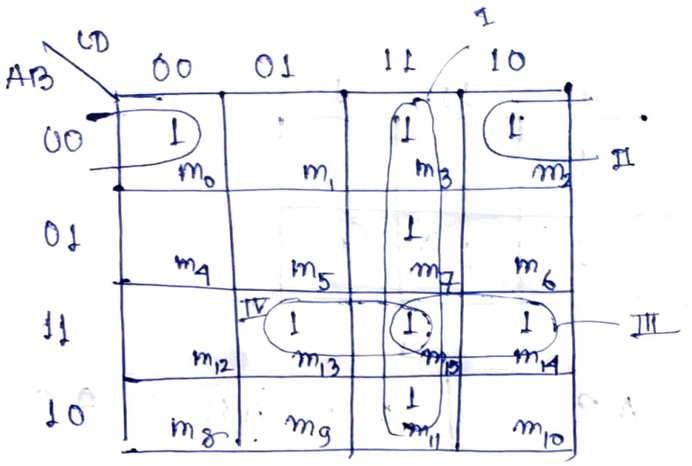
Prime Implicants: Largest possible group of 1's.

Essential Prime Implicants: At least, there is a single one(1) which can not be combined in any other way.

Eg.



Eg. $f(A,B,C,D) = \Sigma m\,(0,2,3,7,11,13,14,15)$.

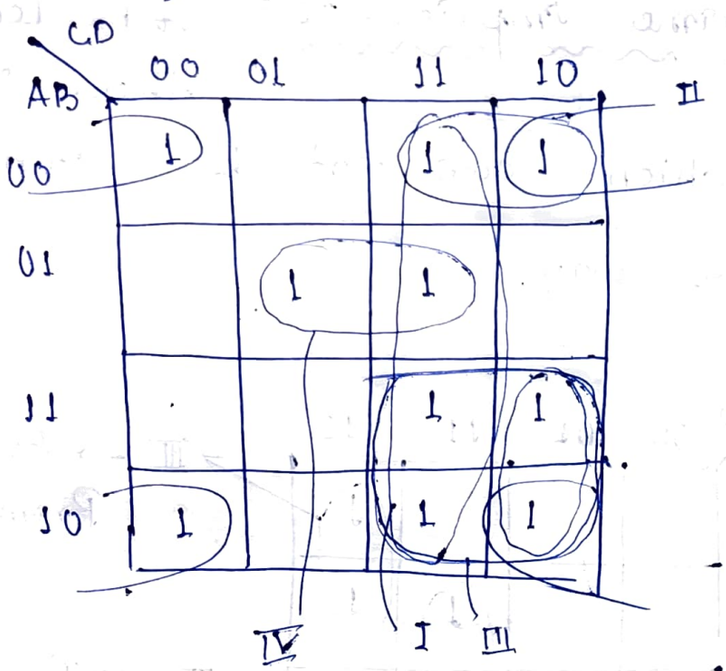


$$f = CD + \bar{A}\bar{B}\bar{D} + ABC + ABD$$

- NB – Combining 2 1's → 1 literal is reduced
  - " 4 1's → 2 " " "
  - " 8 1's → 3 " " "
  - " 16 1's → 4 " " ".

2^n n

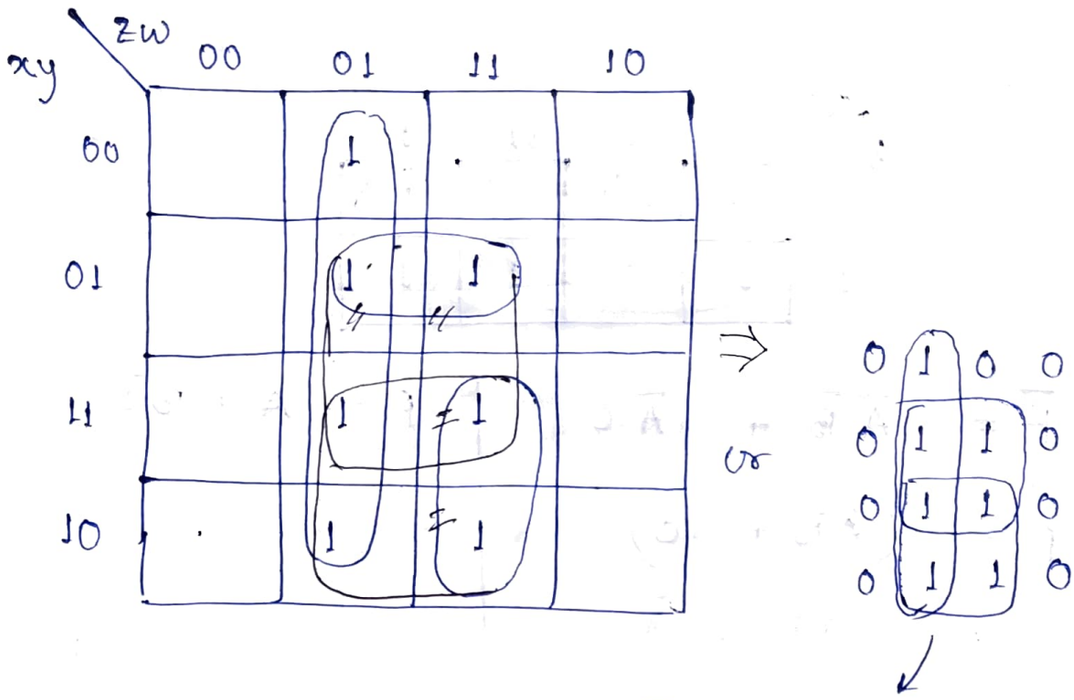
Eg. $F(A,B,C,D) = \Sigma m\,(0,2,3,5,7,8,10,11,14,15)$.



00	10
00	00
10	00
10	10

$$f = CD + \bar{B}\bar{D} + AC + D\bar{A}B.$$

Eg. $F(x,y,z,w) = \Sigma m\,(1,5,7,9,11,13,15)$.

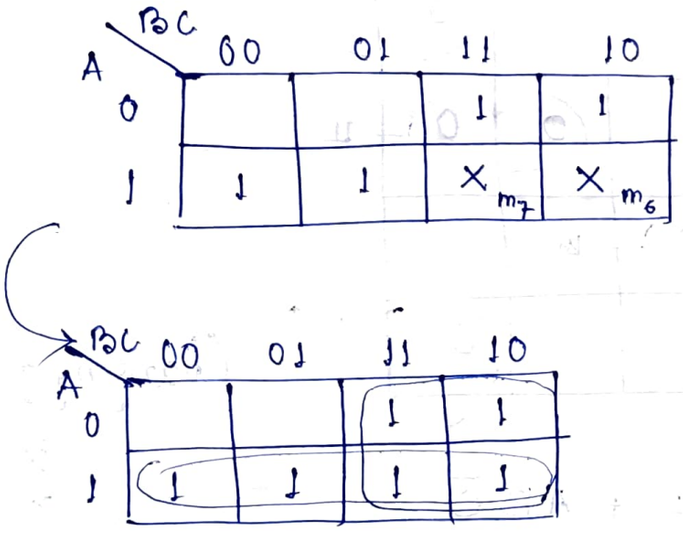


$$f = xw + yw + \bar{z}w.$$

* Don't Care in K' Map:

Eg. $F(A,B,C) = \Sigma m\,(2,3,4,5) + \Sigma d\,(6,7)$.

↳ don't cares



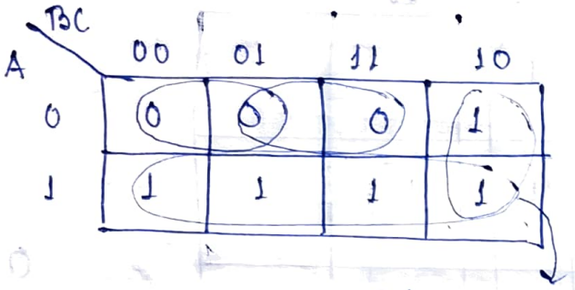
$f = A + B$

d → 0 or 1.

We take don't cares as 1, when reducing using minterms, & take as 0 when reducing using maxterms.

* K'Map using Maxterm.

Eg.



$\bar{F} = \bar{A}\bar{B} + \bar{A}C$ | $F = A + B\bar{C}$.

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

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

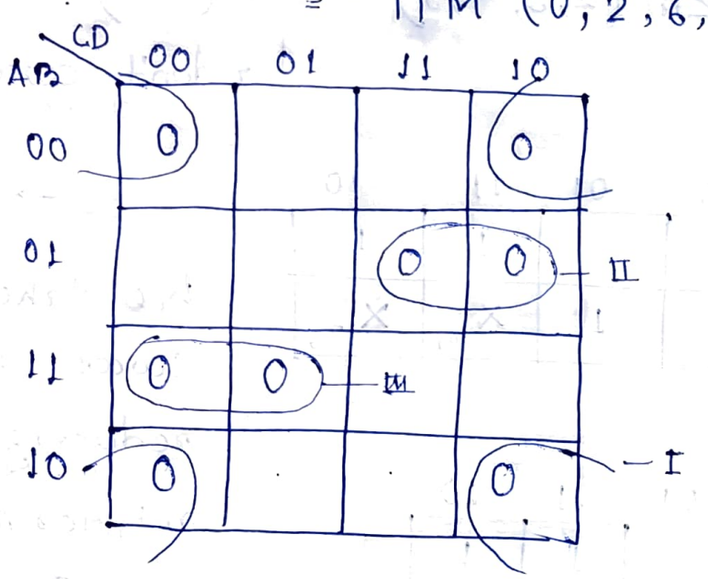
$= (A+B)\cdot(A+\bar{C})$. POS.

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

Complementing

Eg. $F(A,B,C,D) = \Sigma m\,(1,3,4,5,9,11,14,15)$.

$= \Pi M\,(0,2,6,7,8,10,12,13)$



Changing $+ \rightarrow \cdot$, $\cdot \rightarrow +$, $\bar{X} \rightarrow X$, $X \rightarrow \bar{X}$

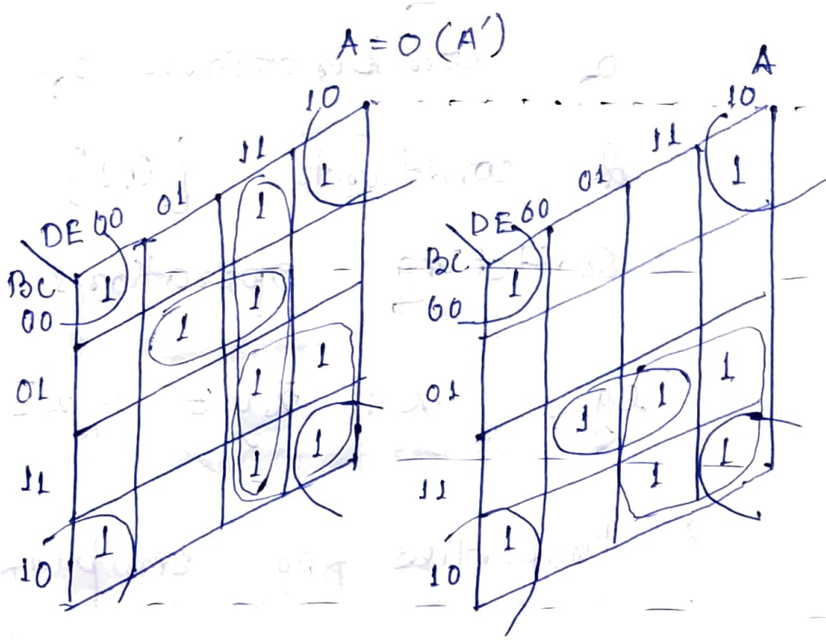
$\bar{F} = \bar{B}\bar{D} + \bar{A}BC + A\bar{B}\bar{C}$.

Complement

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

* K' Map with 5 variables:

	A	B	C	D	E
16	0	0	0	0	0
	0	0	0	0	1
	⋮				
	0	1	1	1	1
16	1	0	0	0	0
	1	0	0	0	1
	⋮				
	1	1	1	1	1



$$F = \bar{C}\bar{E} + BD + \bar{A}DE + \bar{A}BCE + ABCE$$

Switching Algebra.

* **Idempotency:**

$$x \cdot x = x$$
$$x + x = x.$$
$$x + 1 = 1 \qquad x \cdot 0 = 0$$
$$x + 0 = x \qquad x \cdot 1 = x.$$

* **Commutativity:**

$$x + y = y + x$$
$$xy = yx.$$

* **Associativity:**

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

* **Complementation:**

$$x + \bar{x} = 1$$
$$x \cdot \bar{x} = 0$$

* **Distributivity:**

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

* **Duality Principle**

$$\downarrow \quad + \quad 0 \quad 1$$
$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$
$$+ \quad \cdot \quad 1 \quad 0$$

* Switching expression is a finite number of combinations of switching variables & constants $\{0,1\}$ by means of switching operations $(+, \cdot, \sim)$.

eg. $x + \bar{x}yz + x\bar{z}$

* Properties for simplifying the SE:

1. Absorption: $x + x \cdot y = x$

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

2. Dual: $x \cdot (x' + y) = xy$

3. Consensus theorem:

$$xy + \bar{x}z + yz = xy + \bar{x}z.$$

Proof

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

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

$= xy(1 + z) + \bar{x}z(1 + y)$

$= xy + \bar{x}z.$

eg. $x'y'z + yz + xz.$

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

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

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

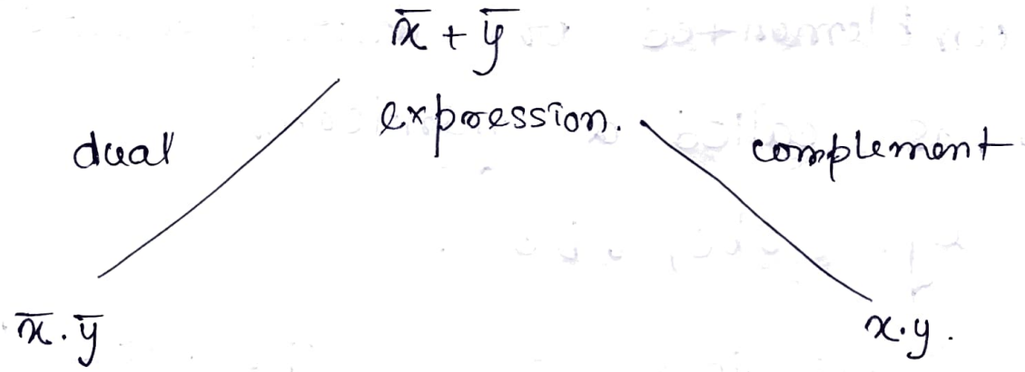
$= z.$

* De Morgan's Law :

$$\overline{xy} = \bar{x} + \bar{y}$$

$$\overline{x+y} = \bar{x} \cdot \bar{y}$$

*



* eg.

$$(x+y)\left[x'(y'+z')\right]' + x'\bar{y} + x'z'$$

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

$$= (x+y)\left[x + yz\right] + x'(y'+z').$$

$x + y \cdot yz$.

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

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

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

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

$$= x + z' + y' + z.$$

$$= 1.$$

* Canonical forms :

→ A product term which contains each of n variables as factors either in complemented or uncomplemented form, is called a minterm.

eg. abc, ab'c .

→ A minterm given the value 1 for exactly one combination of the variables.

→ The sum of all minterms of f for which f assumes 1 is called canonical sum of products or disjunctive normal form.

a + bc + ac SOP

abc + $\bar{a}$bc Canonical SOP.

→ To find canonical SOP, take the sum of product terms of literals when the function value is 1.

→ A sum term that contains each of n variables as factors either in complemented or uncomplemented form is called maxterm.

eg. $(A + \overline{B} + C)(A + B)$

→ A maxterm gives the value 0 for exactly one combination of the variable.

→ The product of all maxterms of f for which f assumes 0 is called canonical product of sums or conjunctive normal form.

$(A+B)(B+C)$. POS.

$(A+B+C)(A+\overline{B}+\overline{C})$ Canonical POS.

→ To find canonical POS form of f, take the maxterms of literals when value of f is 0.

* SOP minterm f is high (1) Σ

POS maxterm f is low (0) Π

eg.

$$f(x,y,z) = x'y + z' + xyz.$$

To find CSOP.

1. Draw truth table & find.
2. AND by 1 i.e. $z + \bar{z}$ (Expansion)

$$f(x,y,z) = x'y(z+z') + z'(x+x')(y+y') + xyz$$

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

$$= \Sigma(0,1,3,4,6,7).$$

$$= \Pi(1,5).$$

* Functional Properties :

→ The CSOP or CPOS form of a switching function is unique.

→ Two switching functions $f_1(x_1, \ldots, x_n)$ & $f_2(x_1, \ldots, x_n)$ are said to be logically equivalent iff both have same value for each & every condition of $(x_1, x_2, \ldots, x_n)$.

→ Two switching functions are equivalent if their CPOS & CSOP are identical.

* Number of functions:

• n - boolean variables.

Then no. of boolean expressions = 2^{2^n}.

• n - ternary variables.

Then no. of boolean functions = 3^{3^n}

• n - Kary variables.

Then no. of m-ary functions = m^{K^n}

* Eg. How many boolean functions are possible with 3 variables s.t. there are exactly 3 minterms? → $^8C_3$.

* How many boolean functions are possible with 3 variables st there are at most 3 minterms? → $^8C_0 + {}^8C_1 + {}^8C_2 + {}^8C_3$

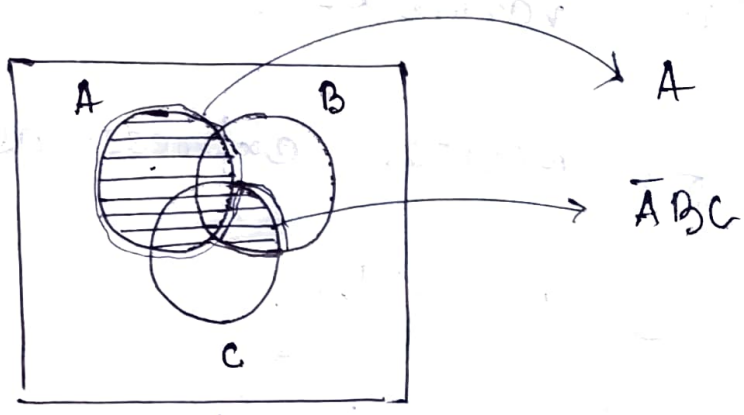
* Generally $^{2^K}C_m$. for k variables & m minterms.

* Neutral Functions: No. of minterms = No. of maxterms

Same no. of 0's & 1's for functional value.

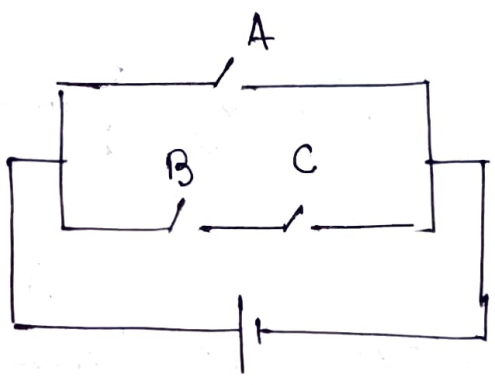
n variables, the no. of nf's = $^{2^n}C_{(2^n/2)} = {}^{2^n}C_{2^{n-1}}$.

Eg. Find the boolean function that the following Venn Diagram represents –



$\Rightarrow A + \bar{A}BC.$

* Contact Representation :



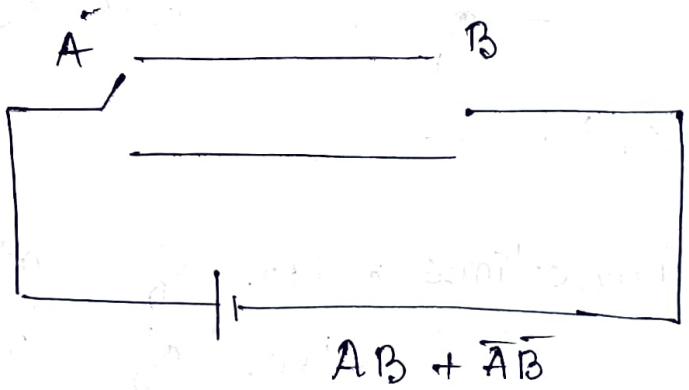
$A + BC.$

→ Every boolean function can be represented with the help of serial & parallel contact.

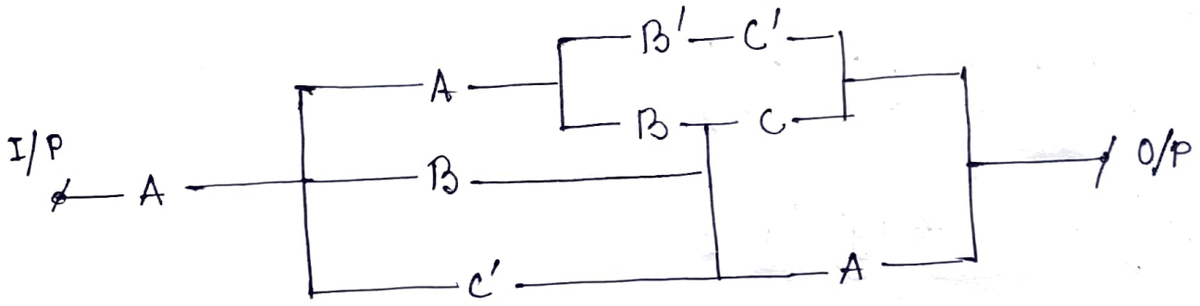
→ Serial contacts are performing AND operation.

→ Parallel contacts are performing OR.

Eg.



eg. Identify the boolean expression given by the circuit.



1. Find the valid forward path.
2. Perform OR among them.

Forward Path: Any path starting from I/P & ending at O/P without forming a cycle.

Validity: No path should contain a variable in both true & complemented form.

$$A\,A\,B'C' + A\,A\,B\,C + A\,A\,B\,A + A\,B\,C + A\,B\,A + A\,C\,C + A\,C'\,A$$

* Eg. In the following simultaneous boolean expressions, what are the values of w, x, y, z ?

$$x + y + z = 1$$
$$xy + w'z' = 0$$
$$xw' + yz' = 1.$$

	w	x	y	z
✗ a)	0	0	0	1
✗ b)	1	1	0	1
✓ c)	0	1	0	1
✗ d)	1	0	0	0

Substitute values.

* Nested function.

Q. $f(A,B) = A' + B$ then find

$$f\left(f(x+y, y), z\right).$$

$\Rightarrow$ $f(x+y, y) = (x+y)' + y$

$= x'y' + y$

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

$= x' + y$

$f\left(f(x+y, y), z\right) = xy' + z$

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

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

* NAND

1. Identity. $A \uparrow A \neq A$.

2. Commutativity. $A \uparrow B = B \uparrow A$

3. Associativity. $A \uparrow (B \uparrow C) \neq (A \uparrow B) \uparrow C$.

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

* Ex-OR

1. Idempotency. $A \oplus A \neq A$.

2. Commutativity $A \oplus B = B \oplus A$

3. Associativity.

$A \oplus (B \oplus C) = (A \oplus B) \oplus C$.

Imp.

** $B \oplus C = \overline{B} \oplus C = B \oplus \overline{C}$

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

* Ex-NOR.

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

1. Idempotency: $A \odot A \neq A$

2. Commutativity: $A \odot B = B \odot A$

3. Associativity: $A \odot (B \odot C) = (A \odot B) \odot C$

imp
- $\overline{A \odot B} = A' \odot B = A \odot B'$
- $\overline{A \odot B} = A \oplus B = A' \odot B = A \odot B'$
- $\overline{A \oplus B} = A \odot B = A' \oplus B = A \oplus B'$

* $\oplus \longrightarrow 1$, for odd no. of 1's

  $\odot \longrightarrow 1$, for even no. of 0's

* $A \oplus B = \overline{A \odot B}$

  $A \oplus B \oplus C = A \odot B \odot C$

  $A \oplus B \oplus C \oplus D = \overline{A \odot B \odot C \odot D}$

No. of inputs even → complementary

No. of inputs odd → identical

* K' Maps. for XOR XNOR

$A \oplus B$

B \ A	0	1
0	1	
1		1

$A \odot B$

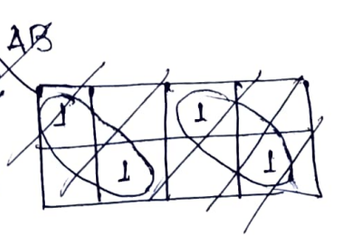
B \ A	0	1
0	1	
1		1

$A \oplus B \oplus C$.

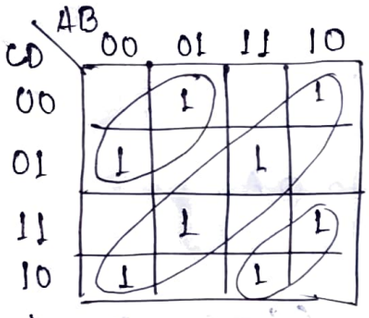
C \ AB	00	01	11	10
0		1		1
1	1		1	

$A \odot B \odot C$.

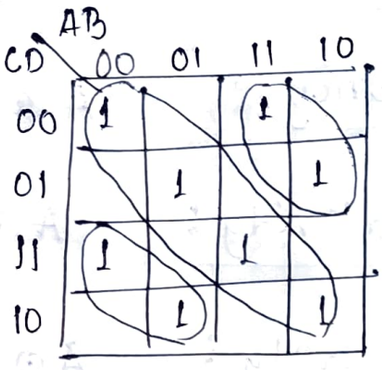
C \ AB	00	01	11	10
0		1		1
1	1		1	



$A \oplus B \oplus C \oplus D$.



$A \odot B \odot C \odot D$.



* Imp.

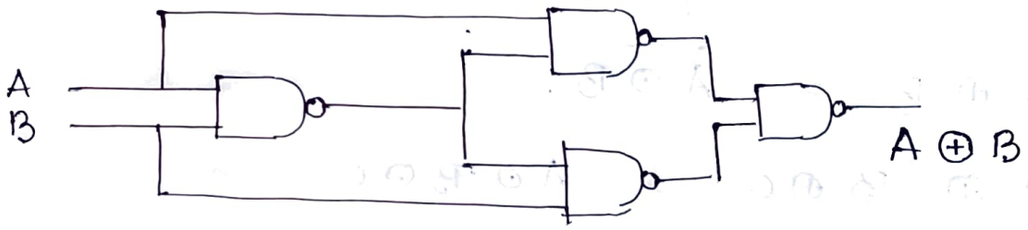
1. No. of minterms = No. of maxterms

2. No. of 0's even → XNOR

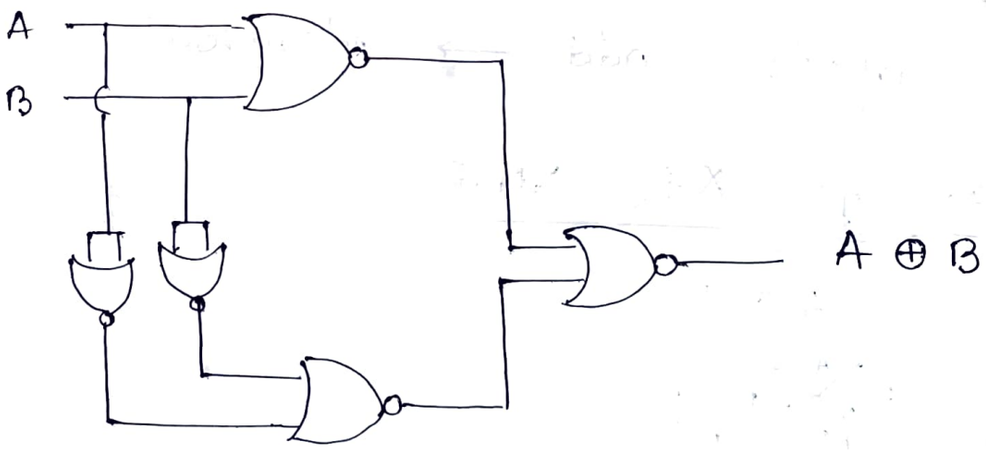
   No. of 1's odd → XOR.

* Ex-OR

1. Using NAND.

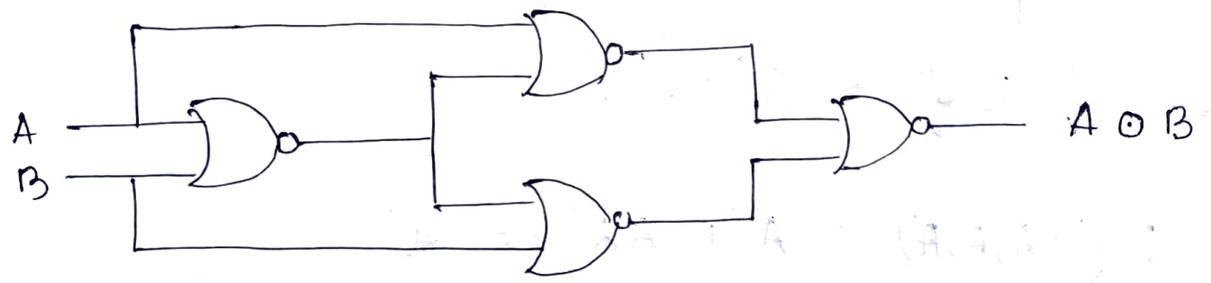


2. Using NOR.

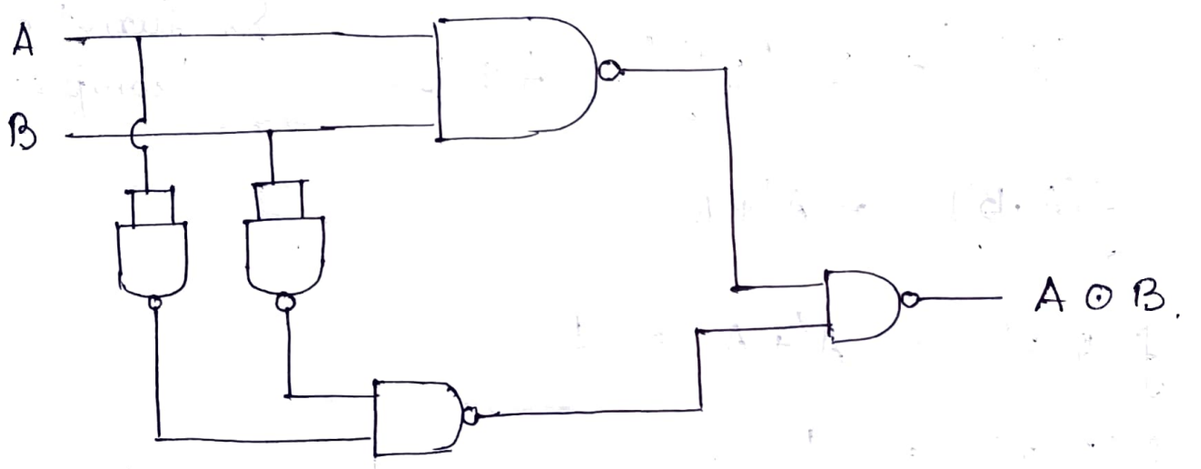


* Ex-NOR.

1. Using NOR.



2. Using NAND.



	NAND	NOR
Ex OR	4	5
EX NOR	5	4.

* Functionally Complete Operations.

A set of operations is said to be functionally complete or universal iff every switching function can be expressed by means of operations in it.

→ The set $\{+, \cdot, -\}$ is clearly functionally complete.

→ The set $\{+, -\}$ is said to be functionally complete.

→ The set $\{\cdot, -\}$ is also functionally complete.

* Examples.

1. $f(A,B,C) = A' + BC'$

↳ $f(A,A,A) = A' + AA' = A'.$

$$f\Big(\underbrace{f(A,A,A)}_{\downarrow A'}, B, \underbrace{f(B,B,B)}_{\downarrow B'} \Big)$$

$= (A')' + B(B')' = A + B.$ So, functionally complete.

2. $f(A,B) = A' + B.$

$f(A,A) = A' + A = 1$

$f(B,B) = 1.$

$f(A,0) = \bar{A} + 0 = A'$

$f(f(A,0), B) = (\bar{A}) + B = A + B.$

It is not functionally complete fully.
It's partially functionally complete.
It takes support from 0.

* Pun^n ⟵ FC $(+, \cdot, -)$
PFC $(1,0) \longrightarrow (+, \cdot, -)$
Not FC.

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

Partially functionally complete.

$f(A,A) = \bar{A}A = 0$

$f(B,B) = \bar{B}B = 0$.

$f(A,1) = \bar{A}\cdot 1 = \bar{A}$ NOT

$f(f(A,1),B) = (\bar{A})\cdot B$.

$= A\cdot B$. AND.

4. $f(A,B,C) = AB + BC + CA$.

No complement term in the expression.
So, it is not functionally complete.
(Complement can never be derived.
It can only be extracted.)

5. $f(x,y) = \bar{x}y + x\bar{y}$

$f(x,x) = \bar{x}x + x\bar{x} = 0$

$f(y,y) = 0$.

RHS.

$f(x,1) = \bar{x}\cdot 1 + x\cdot 0$

$= \bar{x}$. NOT

Not functionally complete.

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

$f(x,y') = \bar{x}\bar{y} + xy$ X (+, ·)

$f(\bar{x},\bar{y}) = x\bar{y} + \bar{x}y$.

* Self-Dual Functions.

Dual

+	·	1	0
↓	↓	↓	↓
·	+	0	1

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

$f_d(A,B,C)$ = dual of f

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

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

$= ABC + ABC' + A'BC + AB'C$.

implying no. of minterms = 4 i.e $\frac{8}{2} = \frac{2^3}{2}$

⇒ Neutral fun^n.

also, there is no mutually exclusive terms on SOP.

⇒ Conditions : A boolean function is self dual if,

(i) It is neutral. (no. of minterms = no. of maxterms = $\frac{2^n}{2}$, $n \rightarrow$ no. of variables).

(ii) The function doesn't contain two mutually exclusive terms.

$ABC \xrightarrow{\text{m.e.}} A'B'C'$

$AB'C \xrightarrow{\text{m.e.}} A'BC'$

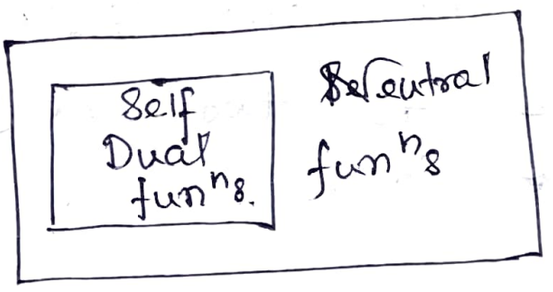
⇒ No. of Self dual functions.

No. of variables = 2^n.

Valid pairs = $\frac{2^n}{2} = 2^{n-1}$.

No. of functions possible =

$2 \times 2 \times \ldots\ldots \times 2^{n-1}$ times $= 2^{2^{n-1}}$



Every self dual f is neutral, but the reverse is not true.

Eg. Which of these is sd.

$f(A,B,C)$ = $\Sigma(0,2,3)$ ✗ Not neutral

$\Sigma(0,1,6,7)$ ✗ Not satisfying 2nd cond.

$\Sigma(0,1,2,4)$ ✓

$\Sigma(3,5,6,7)$ ✓

* If a self dual function is found, its complemented form of the literals is also a self dual function.

i.e. $f(A,B',C)$ is self dual,

$f(A',B,C)$ is also self dual.

So, self duality is closed under complementation.

---

* Electronic Gate Networks.

- Electronic gates generally receives voltages as inputs & produce voltages as outputs.
- The precise values of these voltages are not significant towards determination of logical operation of gates.

• The significant point is that voltages are restricted to two ranges of values, high & low.

• Thus two valued variables may be used to represent these voltages.

• If we associate constant 1 with high V & 0 with low V, it is called positive logic system.

Reverse is negative logic system.

Q. G'16 Let $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ where x_1, x_2, x_3, x_4 are boolean variables & $\oplus$ is XOR operator. Which of these must always be true?

a) $x_1 x_2 x_3 x_4 = 0$

b) $x_1 x_3 + x_2 = 0$

c) $\bar{x}_1 \oplus \bar{x}_3 = \bar{x}_2 \oplus \bar{x}_4$ ✓

d) $x_1 + x_2 + x_3 + x_4 = 0$

→ **XOR — modulo 2 addition.** Imp.

$\oplus$ mod 2 addition.

$\frac{A+B}{2} = 2k + R$

$0 \oplus 0 = \underline{0}$ $\quad \frac{0+0}{2} = 0 + \underline{0}$

$0 \oplus 1 = \underline{1}$ $\quad \frac{0+1}{2} = 0 + \underline{1}$

$1 \oplus 0 = \underline{1}$ $\quad \frac{1+0}{2} = 0 + \underline{1}$

$1 \oplus 1 = \underline{0}$ $\quad \frac{1+1}{2} = 1 \cdot 2 + \underline{0}$

Now, $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ ← { even no. of 1's.

$$\frac{x_1 + x_2 + x_3 + x_4}{4} = 4K + 0$$ ←

i) for 0 no. of 1's it's true.

ii) for 2 no. of 1's it's true

iii) for 4 no. of 1's it's true.

Option A. $x_1 x_2 x_3 x_4 = 0$

for 4 no. of 1's, it fails.

Option B $x_1 x_3 + x_2 = 0$.

for $x_1 = 1, x_3 = 1, x_2 = 0, x_4 = 0$, it fails.

Option D $x_1 + x_2 + x_3 + x_4 = 0$.

for $x_1 = 1 = x_2 = x_3 = x_4$; it fails.

Hence, C. ✓

Now, $\bar{x}_1 \oplus \bar{x}_3 = x_1 \oplus x_3$

$\bar{x}_2 \oplus \bar{x}_4 = x_2 \oplus x_4$ } Property.

for all (i), (ii), (iii) it's satisfying.

$x_1 \oplus x_3$		$x_2 \oplus x_4$
		0 0
1	1	1 1
0	0	1 0.
1	0	
⋮	⋮	⋮

for all.

Q. GT'06 Consider numbers represented in 4-bit gray code. Let $h_3h_2h_1h_0$ be the gray code representation of a number 'n' & let $g_3g_2g_1g_0$ be the gray code of $(n+1)$ modulo 16 value of the nember. Which one is correct?

a) $g_0(h_3h_2h_1h_0) = \Sigma(1,2,3,6,10,13,14,15)$

b) $g_1(h_3h_2h_1h_0) = \Sigma(4,9,10,11,12,13,14,15)$

c) $g_2(h_3h_2h_1h_0) = \Sigma(2,4,5,6,7,12,13,15)$

d) $g_3(h_3h_2h_1h_0) = \Sigma(0,1,6,7,10,11,12,13)$.

→ $(n+1)$ modulo 16 is nothing but the next number of n for 4-bit gray code, as after 15 we want to get back to 0. That's why we use modulo operation.

Now,

$h_3h_2h_1h_0$	gray code	$g_3g_2g_1g_0$
0		1
1		2
2		3
3		4
4		5
5		6
6		7
7		8
⋮		⋮
14		15
15		0

We need to make g_3 as a fun^n of $h_3h_2h_1h_0$ & so for g_2, g_1, g_0 also. Then pick right option.

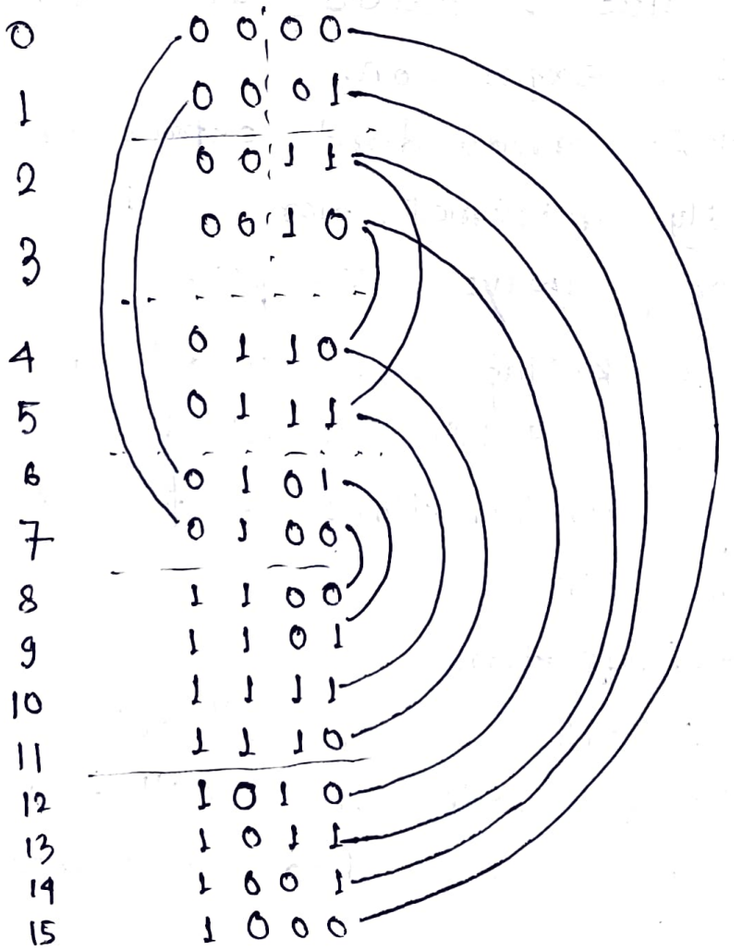
	h_3	h_2	h_1	h_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	1
3	0	0	1	0	0	0	1	0
4	0	1	1	0	0	1	1	0
5	0	1	1	1	0	1	1	1
6	0	1	0	1	0	1	0	1
7	0	1	0	0	0	1	0	0
8	1	1	0	0	1	1	0	0
9	1	1	0	1	1	1	0	1
10	1	1	1	1	1	1	1	1
11	1	1	1	0	1	1	1	0
12	1	0	1	0	1	0	1	0
13	1	0	1	1	1	0	1	1
14	1	0	0	1	1	0	0	1
15	1	0	0	0	1	0	0	0

m_4

m_{15}.

row numbers are not representing the minterm numbers. We have to derive them separately as it's not normal bin. code.

↖ Write gray codes using mirror concept.



Now, taking options one by one, & checking them,

$$g_2(h_3 h_2 h_1 h_0) = \Sigma (2,4,5,6,7,12,13,15)$$

Minimization.

* Criteria to determine minimal cost:

1. Minimal no. of appearances of literals.
2. Minimum no. of literals in SOP or POS expression.
3. Minimum no. of terms in SOP expression, provided there is no other such expression with the same number of terms & fewer literals.

* Irredundant Expression.

An SOP expression from which no term or literal can be deleted without altering its logical value is called an irredundant or irreducable expression.

→ An irredundant expression is not necessarily minimal, nor the minimal expression always unique.

* K-Map. A K-Map is a modified form of a truth table in which the arrangement of combinations is particularly convenient for minimization.

- Simplification.

1. A collection of 2^m cells, each adjacent to m cells of collection is called a subcube, & the subcube is said to cover these cells.

2. Each subcube can be expressed by a product containing n-m literals where n is no. of variables on which function depends.

Any cell may be included in as many subcubes as desired.

3. A function f can be expressed as sum of those product terms which corresponds to subcubes necessary to cover all its '1' cells.

4. The number of product terms in the expression for f is equal to the no. of subcubes, while the no. of literals in each term is determined by size of corresponding subcubes.

Therefore to obtain a minimal expression, we must cover all '1' cells with smallest no. of subcubes such that each subcube is as long as possible.

* Implicants: A switching function $f(x_1, x_2, \ldots, x_n)$ is said to cover $g(x_1, \ldots, x_n)$ denoted by f is superset of g if f assumes true value whenever g does.

$$f \supseteq g$$
$$\{0,1\} \supseteq \{1\}$$

	g	f	
0	0	1	g → f
1	1	1	
2	0	0	
3	0	0	

• If f covers g & g covers f, f & g are equivalent. If g has x minterms & g is a function of n variables, then no. of covering functions for g is $2^{(2^n - x)}$.

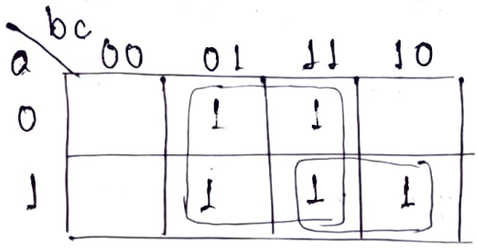
• If f covers g, then g is said to imply f. This is denoted by $g \rightarrow f$.

g is implicant of f.

eg. $f(a,b,c) = ab + c$

here ab is implicant of f

c is implicant of f.



Every subcube is an implicant.

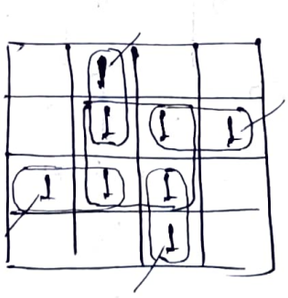
* Prime Implicant: An implicant P of a function f is said to be prime implicant if

i) P is a product term (i.e. subcube).

ii) Deletion of any literal from P results in a new product which is not covered by f i.e. a subcube cannot be part of any other subcube. It should be the longest possible subcube.

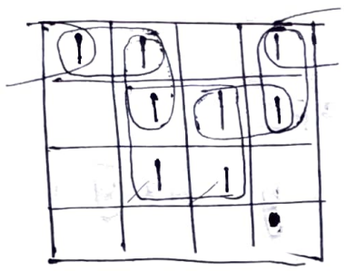
* Essential Prime Implicant: A prime implicant P of a function f is said to be an essential prime implicant if it covers at least one minterm of f that is not covered by any other prime implicants.

Eg.

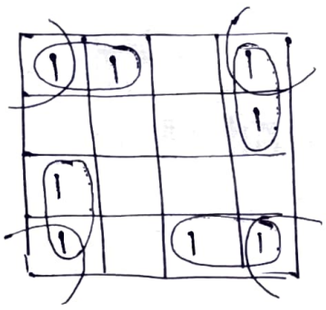


PI → 5

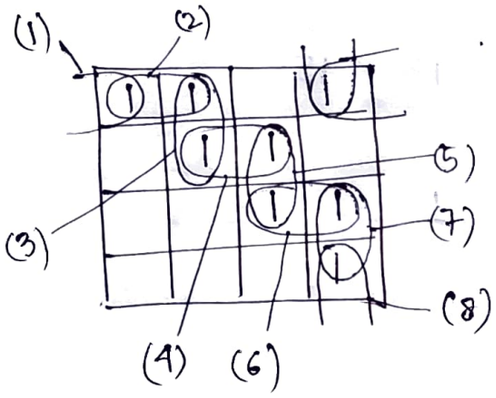
EPI → 4.



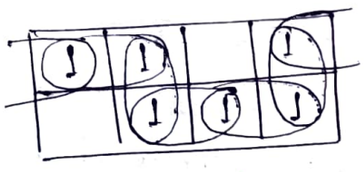
PI → 6
EPI → 1.



PI → 5
EPI → 1.



PI → 8
EPI → 0
cyclic.

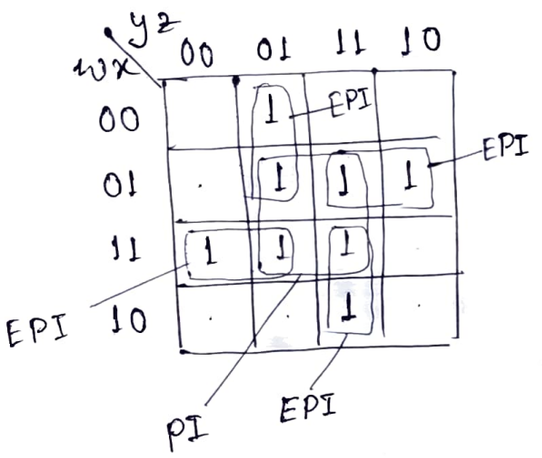


PI → 6.
EPI → 0. cyclic.

* Procedure for obtaining minimal SOP:

1. Determine all essential prime implicants & include them in the minimal SOP.
2. Remove from list of prime implicants all those which are covered by essential prime implicants.
3. If the set determined in step 1, covers all the minterms of f then it is unique minimal expression, otherwise select the additional prime implicants so that the function f is covered completely & the total number & size of prime implicants added are minimal

Eg. $\Sigma(1,5,6,7,11,12,13,15) = f(w,x,y,z)$



1. Take all EPI.

$f = \bar{w}\bar{y}z + \bar{w}xy + wx\bar{y} + wyz$

PI is redundant.

PI chart ***

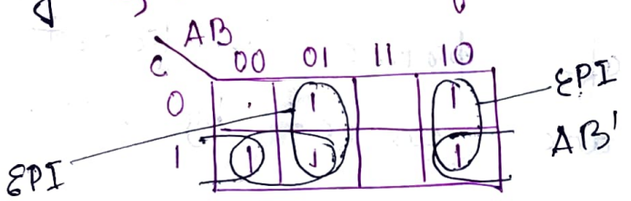
	1	5	6	7	11	12	13	15
$\bar{w}\bar{y}z$	1	1						
$wx\bar{y}$						1	1	
wyz					1			1
$\bar{w}xy$			1	1				
xz	✓	1	✓	1	✓	✓	1	1

No 1 on this column except for $\bar{w}\bar{y}z$

$\bar{w}xy$ wyz $wx\bar{y}$

All minterms covered, so, we need not to go on.

Eg. Q. Which of the PI s are essential?



a) $B'C$, $A'B$
b) $A'C$, $A'B$
c) $A'B$, AB' ✓
d) $A'B$, AB', $B'C$

~~A'B~~

* Determining minimal POS.

Similar -

$+ \longrightarrow \cdot$

$\cdot \longrightarrow +$

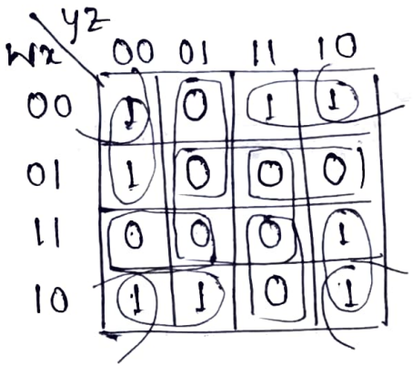
$1 \Rightarrow \bar{x} \longrightarrow x$

$0 \Rightarrow x \longrightarrow \bar{x}$

Q. Find the no. of literals in minimum POS & SOP for

$$f(w,x,y,z) = \Pi\ (1,5,6,7,11,12,13,15)$$

→



POS

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

SOP

$$w\bar{x}\bar{y} + wy\bar{z} + \bar{w}\bar{x}y + \bar{w}\bar{y}\bar{z}$$

12 literals. Ans.

* Don't Care Combinations.

→ A function is said to be completely specified if it is given 0 or 1 for every combination of i/p variables. There exists some functions which are not completely specified.

→ Combinations for which the value of a function is not specified are called don't care combinations.

→ The value of a function for it is denoted by 'ϕ' or 'd'.

→ Since each don't care combination represents two values {0,1}, an incompletely specified function containing K don't care combinations corresponds to a class of 2^K distinct functions.

→ Our task is to choose a function having minimal representation out of these 2^K functions.

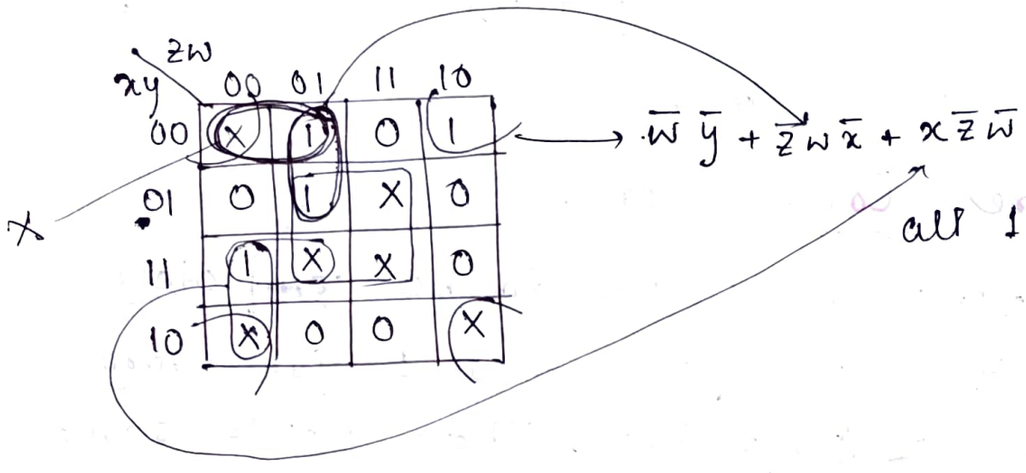
→ We could assign a 0 or 1 to a don't care combination on such a way to increase or decrease size of a subcube.

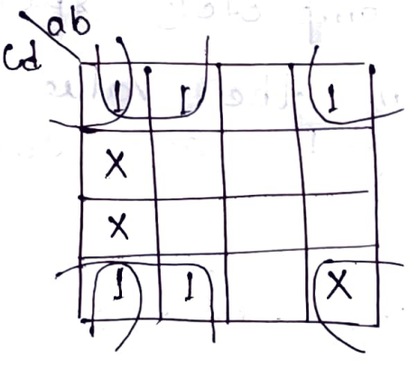
Eg.

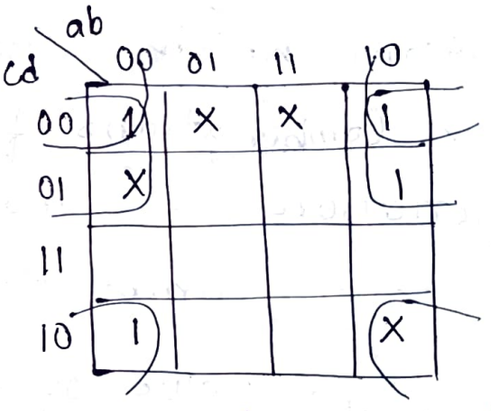
AB \ CD	00	01	11	10
00	0	0	1	0
01	X	X	1	X
11	0	1	1	0
10	0	1	1	0

CD + AD:

Minimal SOP.







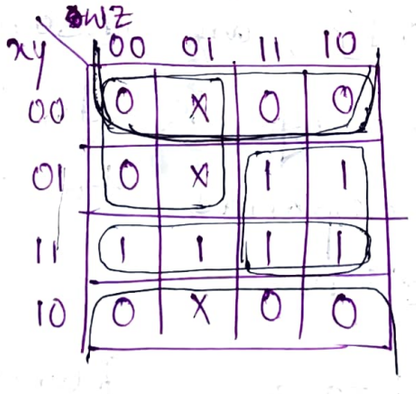
Q. Which function does not implement the K'Map given?

a) $(w+x)y$

b) $xy + yw$

c) $(w+x)(\bar{w}+y)(\bar{x}+y)$

d) None



→

a) $(w+x)y$ POS

covers all zeroes

b) $xy + yw$ SOP.

covers all 1's

$wy + xy$. SOP

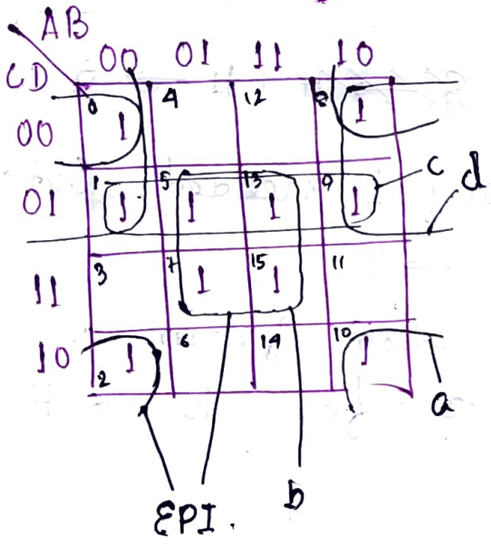
c. $(w+x)(\bar{w}+y)(\bar{x}+y)$ POS.

xy \ wz	00	01	11	10
00	0	X	0	0
01	0	X	1	1
11	1	1	1	1
10	0	X	0	0

All 0's covered.

d. None.

Q How many minimal expressions are possible?



PI – 4.

	0	1	2	5	7	8	9	10	13	15
a	1		1			1		1		
b				1	1				1	1
c		(1)		1			(1)		1	
d	1	(1)				1	(1)			

a → EPI. b → EPI

c and d same size.
They are equal candidates to be included in SOP.

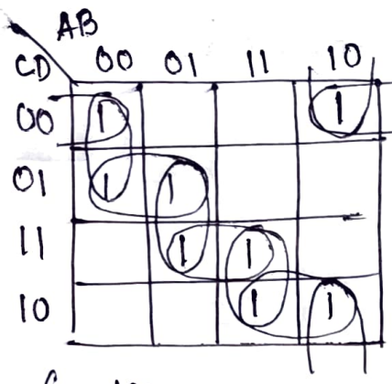
$a+b+c$ $a+b+d$

So, 2 minimal expressions are possible.

reduced PI chart

	1	9
c	1	1
d	1	1

* Branching method for Cyclic functions :



Cyclic.

$A + C + G + E$

A has been covered. So, delete rows & columns for A.

	0	1	5 ✓	7 ✓	8 ✓	10 ✓	14	15
A $w'x'y'$	×	×						
B $w'y'z'$		×	×					
C $w'xy$			(×)	×				
D xyz				×				×
E wxy							×	×
F wyz'						×	×	
G $wx'z'$					(×)	×		
H $x'y'z'$	×				×			

B → C ~~G~~ H → G

B & H cannot needed. Cut them.

F → E D → E.

This was done with initiation from A. We have to make this for B to H & find the minimal.

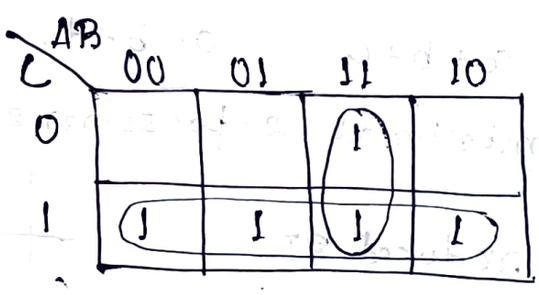
Q. Which of the following can be a prime implicant of $f(A,B,C)$?

a) $ab + c$

b) $Bc + a$

c) $ac + b$

d) ab



Q. Which cannot represent PI set of $f(A,B,C)$?

a) $\{ab, bc, ca\}$

b) $\{bc'a, ba, bc\}$ as $bc'a \rightarrow ba$.

c) $\{a', b, ac\}$

d) $\{a'b', ab\}$

Q. What minterms have to be added to make the following a self-dual? ⇒ 4.

→ $f(A,B,C,D) = A'BC + (AC+B)D$.

i) minterm no = maxterm no.

ii) me terms should not be present.

Pairable

(0,15) (1,14) (2,13) (3,12) (4,11) (5,10) (6,9) (7,8)

$f = A'BC(D+D') + A(B+B')CD + (A+A')B(C+C')D.$

$= A'BCD + A'BCD' + ABCD + AB'CD + AB'C'D +$
7 6 15 11 5

$AB\bar{C}'D$
13

2 terms missing

We can add (1,3) (14,12) (14,3) (1,12).

Q. How many functions does $f_1 \cdot f_2$ & $f_1 + f_2$ represent?

$f_1(a,b,c) = \Sigma(0,2,4) + \Sigma\phi(3,5,7)$

$f_2(a,b,c) = \Sigma(2,3) + \Sigma\phi(1,6,7)$.

→

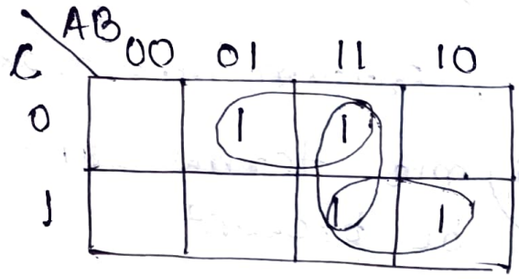
	f_1	f_2	$f_1 \cdot f_2$	$f_1 + f_2$
0	1	0	0	1
1	0	x	0	x
2	1	1	1	1
3	x	1	x	1
4	1	0	0	1
5	x	0	0	x
6	0	x	0	x
7	x	x	x	x
			2^2	2^4

Ans.

Q. The number of prime implicants, EPI & redundant PI for the function

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

→



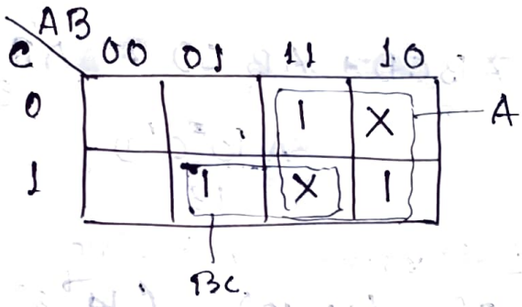
PI → 3

EPI → 2

RPI → 1.

Q. A function $f(A,B,C) = \Sigma(3,5,6)$ is minimised to $(A + BC)$, then what are the don't cares ?

→



$\Sigma_\phi(1,7)$.

Q. In a K-Map it was found that EPIs are covering all terms except 2 minterms. Those 2 minterms are in turn covered by 3 non-essential prime implicants each. What is the no. of minimal SOP expressions.?

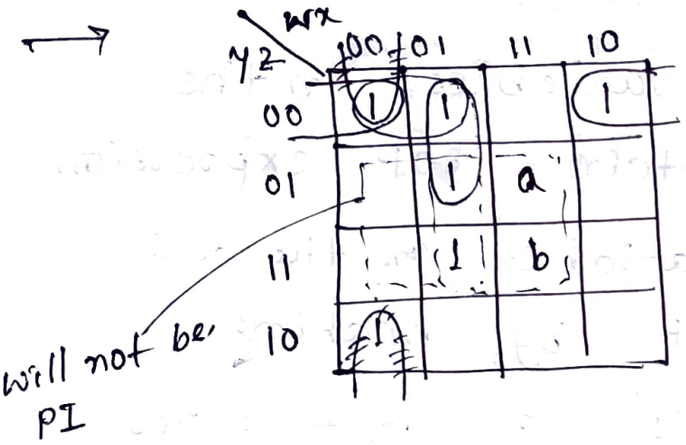
→

EPI + $\underbrace{\_\_}_{\text{2 minterms}}$ + $\underbrace{\_\_}_{\text{2 minterms}}$ 3 choices 3 choices

$3 \times 3 = 9$ minimal expressions are possible.

Q. In a PI chart representing a boolean expression $f(w,x,y,z)$ columns represent minterms & rows represent PI, identify P, Q, R, S, & a, b.

	0	4	5	7	8	a	b
P	✓	✓					
Q	✓				✓		
R		✓	✓				
S			✓	✓		✓	✓



P → $\bar{w}\bar{y}\bar{z}$

Q → ~~$\bar{w}\bar{x}\bar{z}$~~ $\bar{x}\bar{y}\bar{z}$

R → $\bar{w}x\bar{y}$

S → xz.

a → 13

b → 15

* Variable entrant Map (VEM)

$f(A,B,C) = \Sigma(1,3,5,7)$.

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

A	B	f
0	0	C
0	1	C
1	0	C
1	1	C.

$f(A,B,C) = \Sigma(1,2,5,6)$.

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	B	f
0	0	C
0	1	1
1	0	C
1	1	C'

B \ A	0	1
0	C	C
1	1	C'

* Minimisation using VEM:

1. Set all the variables in the cell as 0 & obtain SOP expression.

2. a. Make one variable in the cell as 1 & obtain SOP by making earlier minterms (1's) as don't cares.

2. b. Multiply the above SOP with the concerned variable.

3. Repeat step 2 until all the variables on the cell are covered.

4. SOP of VEM is obtained by ORing the previous SOP expressions.

Eg.

C \ AB	00	01	11	10
0	D	1	D'	D'
1	D	1	0	ϕ

consider D & D' as seperate variables.

Step 1.

C \ AB	00	01	11	10
0	0	1	0	0
1	0	1	0	ϕ

SOP = A'B

Step 2.

C \ AB	00	01	11	10
0	1	ϕ	0	0
1	1	ϕ	0	ϕ

A' → A'D

C \ AB	00	01	11	10
0	0	ϕ	1	1
1	0	ϕ	0	ϕ

AC' → AC'D'

Step 3

$A'B + A'D + AC'D'$ Ans.

Q. $f(A,B,C) = \Sigma(3,5,6,7)$ is realised by following VEM, then find P, Q, R, S.

b \ a	0	1
0	P	S
1	R	Q

a) P=0, R=S=C, Q=1

b) P=Q=0, R=C, S=1

c) P=0, Q=R=C, S=1

d) None.

→

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

VEM.

A	B	f	
0	0	0	— P
0	1	C	— R
1	0	C	— S
1	1	1	— Q

Q. How many variables are free in the expression denoted by the K-Map

yz \ wx	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

$(\bar{x}z + x\bar{z})$ SOP

$(x+z)(\bar{x}+\bar{z})$ POS

(w, y) Ans.

Independent of w & y, both pos & SOP.

Q. Let $f(w,x,y,z) = \Sigma(1,2,3,5,13) + \Sigma_\phi(6,7,8,9,11,15)$ & let $f(w,x,y,z)$ be minimal SOP & $g(w,x,y,z)$ be the minimal POS. Are f & g identical?

→

SOP

yz \ wx	00	01	11	10
00				Ø
01	1	1	1	Ø
11	1	Ø	Ø	Ø
10	1	Ø		

$\phi = 0$

POS

yz \ wx	00	01	11	10
00	0	0	0	Ø
01				Ø
11		Ø	Ø	Ø
10		Ø	0	0

$\phi = 1$

$f = \Sigma(1,2,3,5,6,7,9,11,13,15)$

f & g aren't identical. $g = \Sigma(1,2,3,5,9,13)$.

Q. Minimal expression represented by the map is free from

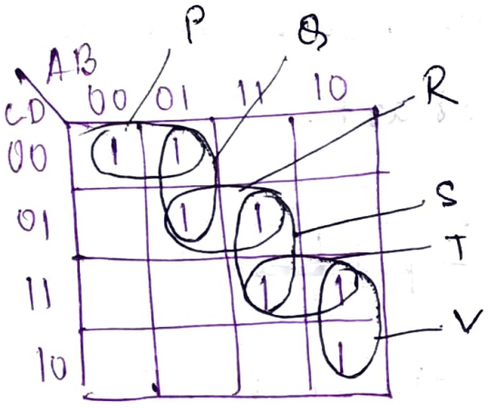
a) 1 variable .
b) 2 "
c) 3 "
d) dependent on all.

CD \ AB	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	1	1	1	1
10	1	1	1	1

→ * Both POS & SOP (minimal) will be independent of same variables if there are no don't cares.

$\bar{A}D + C + \bar{A}\bar{B}$ "independent of none dependent on all.

Q.



1) PI
2) EPI
3) RPI
4) minimal SOP, POS
5) how many minimal expressions.

→ PI → 6 EPI → 2 RPI → 6 - 2 = 4.

PI chart. —

	0	4	5	13	15	11	10
P = $A\bar{C}\bar{D}$	✓	✓					
Q = $\bar{A}B\bar{C}$		✓	✓				
R = $B\bar{C}D$			✓	✓			
S = ABD				✓	✓		
T = ACD					✓	✓	
V = $AB\bar{C}$						✓	✓

Q → R
T → S.

EPI $\bar{A}\bar{C}\bar{D}$

EPI $AB\bar{C}$.

P + V + R + S Ans (4)

(5)

5 13 15

$(Q + R) \cdot (R + S) \quad (S + T)$

$= QRS + QRT + QSS + QST + RRS + RRT + RSS + RST$

$= QRS + QRT + QS + QST + RT + RS + RST.$

redundant

3 minimal expressions.

* NB. Don't cares are never included in the prime implicant chart..

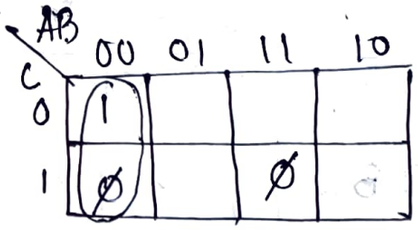
Q. $f_1(A,B,C) = \Sigma(0,7) + \Sigma\phi(1,2,5)$

$f_2(A,B,C) = \Sigma(0,1,3) + \Sigma\phi(4,7)$

find $f_1 \cdot f_2$ & $f_1 + f_2$ when minimised.

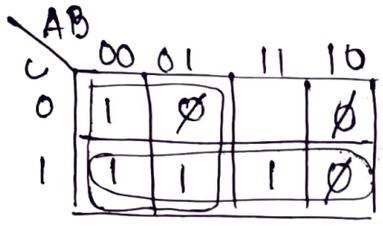
→

	f_1	f_2	$f_1 \cdot f_2$	$f_1 + f_2$
0	1	1	1	1
1	ø	1	ø	1
2	ø	0	0	ø
3	0	1	0	1
4	0	ø	0	ø
5	ø	0	0	ø
6	0	0	0	0
7	1	ø	ø	1



$f_1 \cdot f_2$

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



$f_1 + f_2$

$C + \bar{A}$

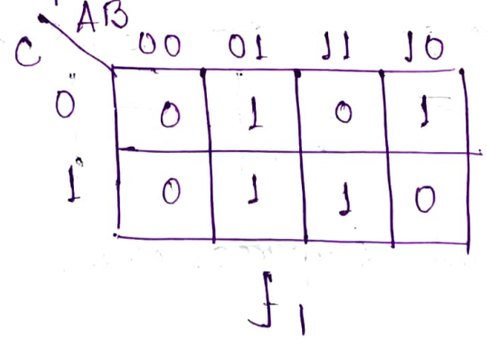
By inspection—

$f_1 \cdot f_2 = \Sigma(0) + \Sigma\phi(1,7)$

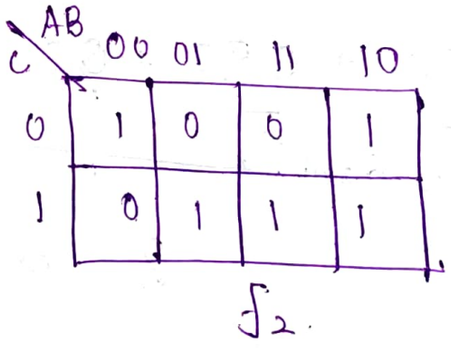
$f_1 + f_2 = \Sigma(0,1,3,7) + \Sigma\phi(2,5,4)$

Q. Consider a new boolean operation * defined as $A * B = A' + B$, then find

$f_1 * f_2$

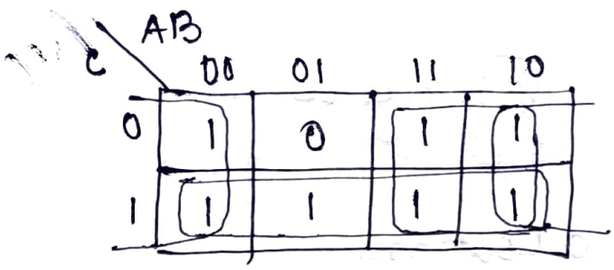


f_1



f_2

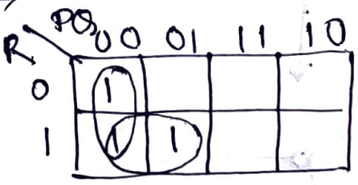
→ $f_1 * f_2 = f_1' + f_2$.



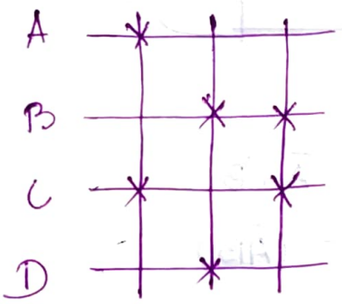
$\overline{C} + \overline{B} + \overline{A}$ Ans.

Q. Consider three three variable functions $f_1(P,Q,R) = \Sigma(0,1,3)$; $f_2(P,Q,R) = \Sigma(3,5,7)$ $f_3(P,Q,R) = \Sigma(1,3,7)$. These functions are sharing 4 prime implicants A, B, C, D (all of them are pairs) as shown

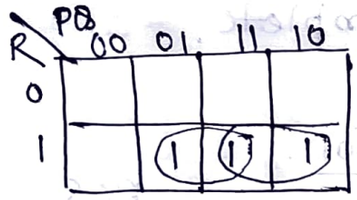
→ f_1 —



$P'Q' + P'R$.

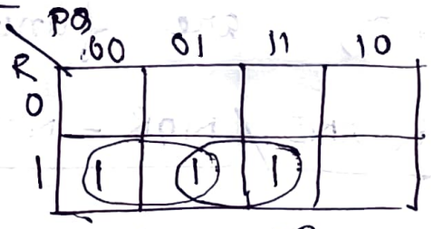


f_2 —



$QR + PR$

f_3 —



$P'R + QR$

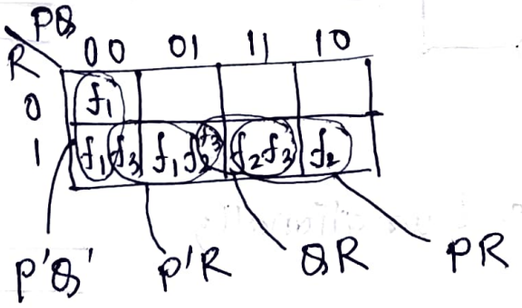
C — $P'R$

B — QR

D — PR

A — $P'Q'$

Method 2.



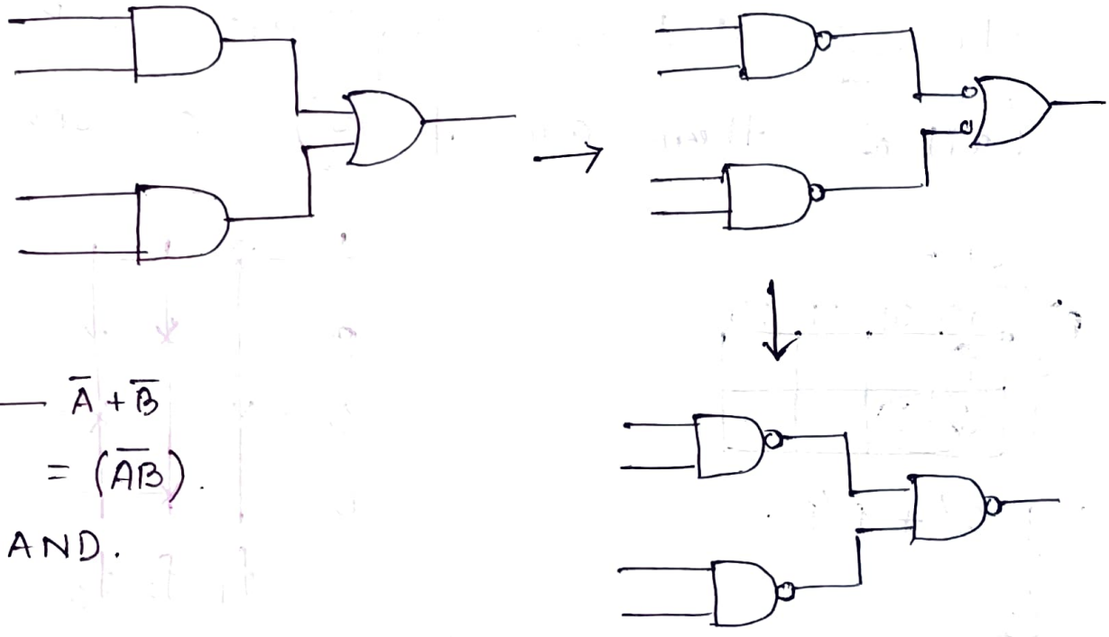
Design & Synthesis of Combinational Circuits.

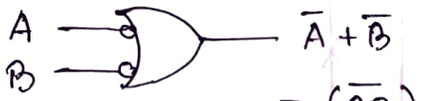
* SOP — AND-OR realisation.

  POS — OR-AND "

* AND-OR / NAND-NAND realisation

SOP





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

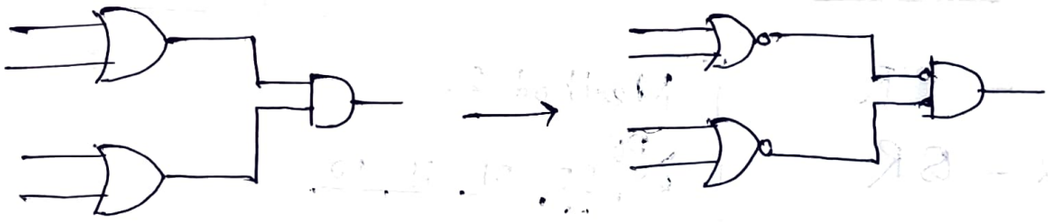
$= (\bar{A}\bar{B})$.

NAND.

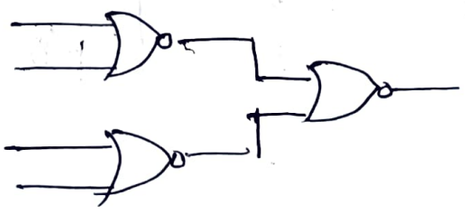
(·, ~) are functionally complete.

* OR-AND / NOR-NOR realisation.

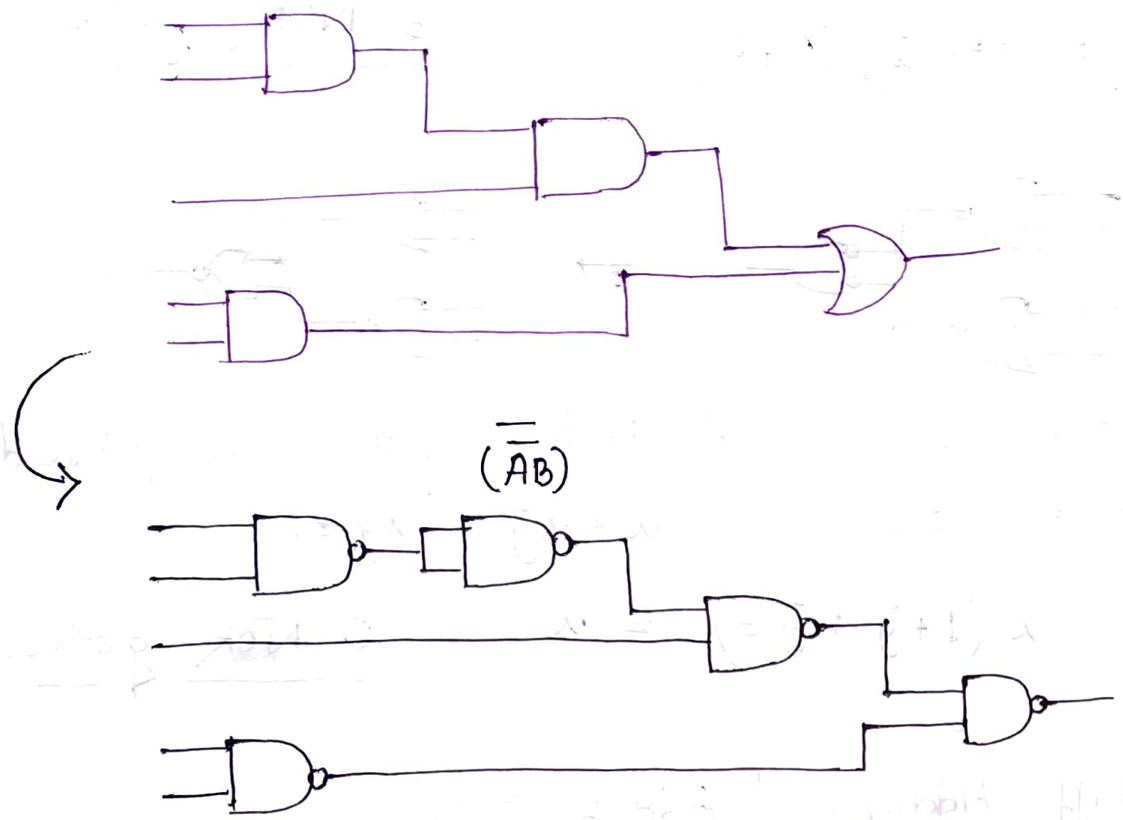
POS



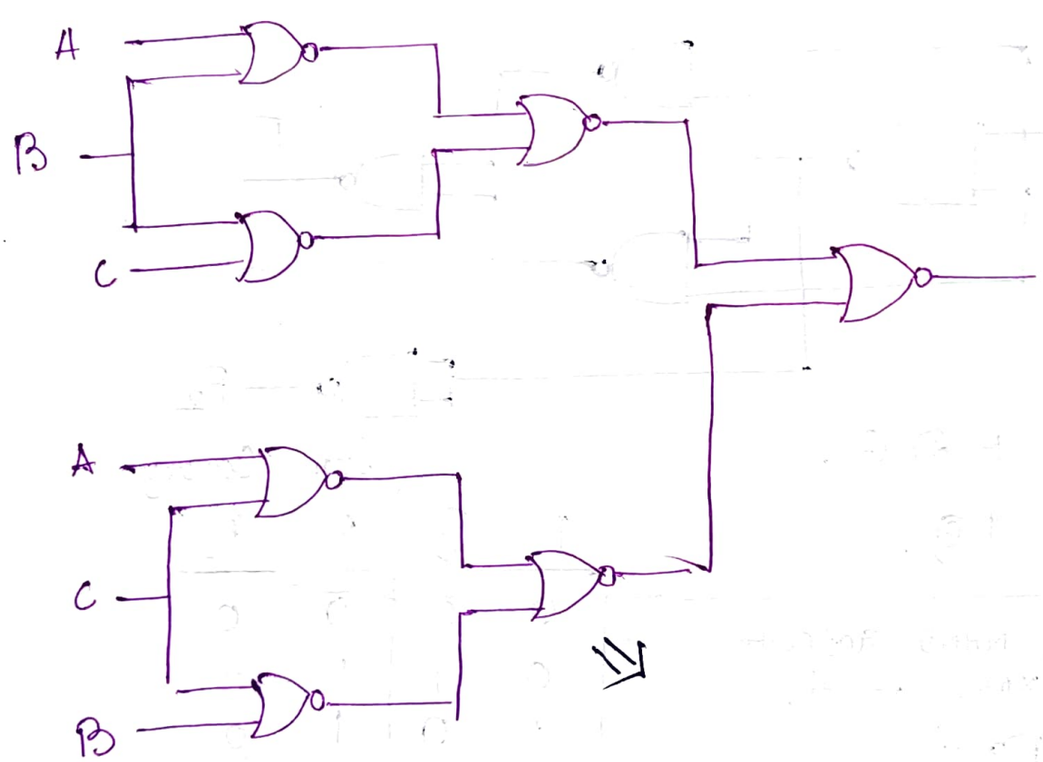
(+, ~) are functionally complete.



Q. Identify min no. of two i/p NAND gates required to represent the following.



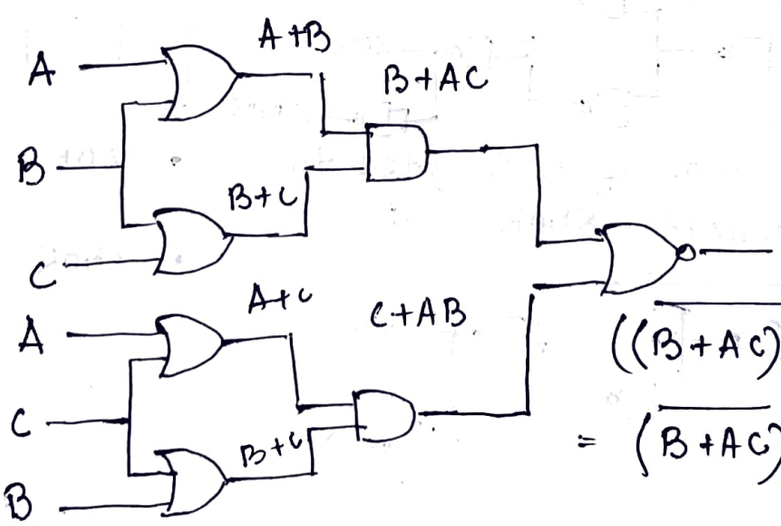
Q.



$\overline{(B+AC)}\ \overline{(C+AB)}$

$= \cancel{B\,\overline{AC}}\ \overline{B}\,\overline{C}$

(Ans)



$\overline{((B+AC)+(C+AB)}$

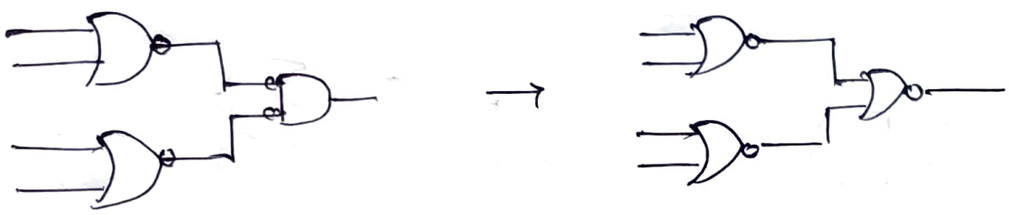
$= \overline{(B+AC)}\ \overline{(C+AB}$

* Find the no. of i/p NOR gates required to represent $f(P,Q,R) = P+QR$.

→ $P+QR$

$= (P+Q)(P+R)$

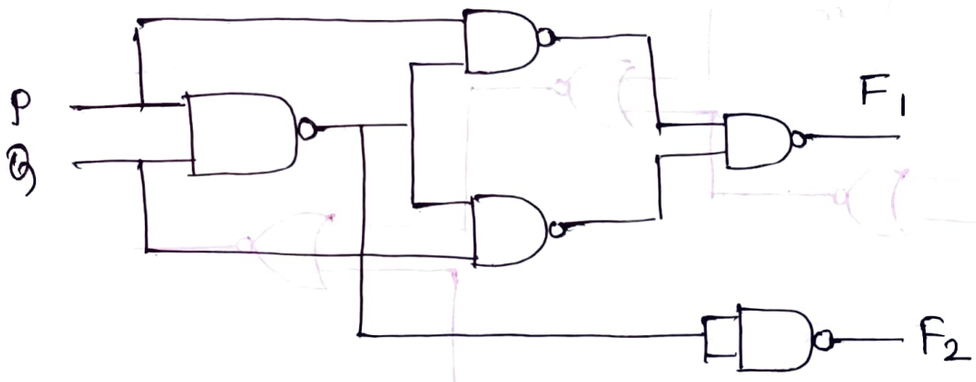
3 NOR gates.



* Minimum no. of NOR gates required to implement $x + x\bar{y} + x\bar{y}z$.

→ $x(1+\bar{y}+\bar{y}z) = x$. 0 NOR gates.

* Half Adder. [Add 2 bits]



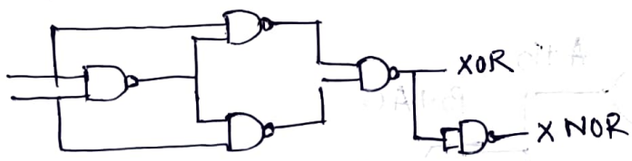
$F_1 = P \oplus Q$.

$F_2 = PQ$.

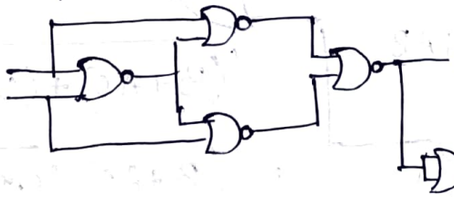
Sum Carry

P	Q	F_1	F_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Min. no. of NAND gates to implement XOR. – 4.

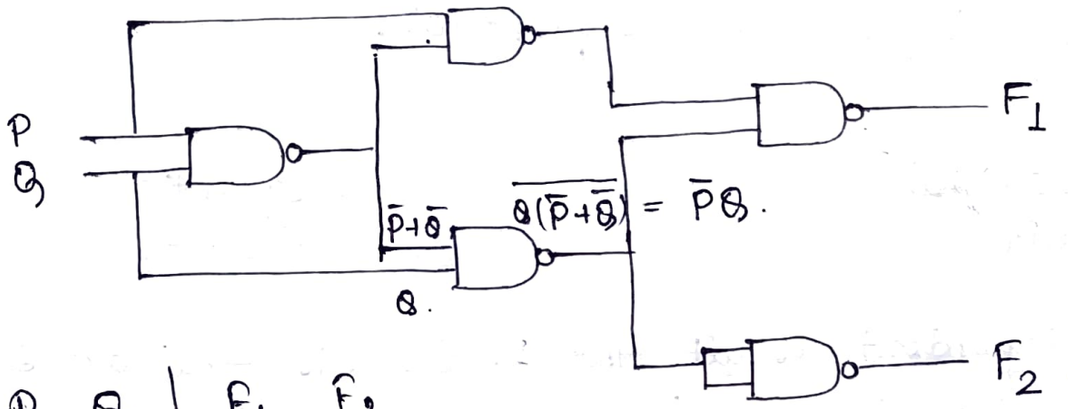


- Min. no. of NOR gtes to implement XNOR – 4



- XOR using NOR – 5 NOR gates
- XNOR using NAND – 5 NAND gates

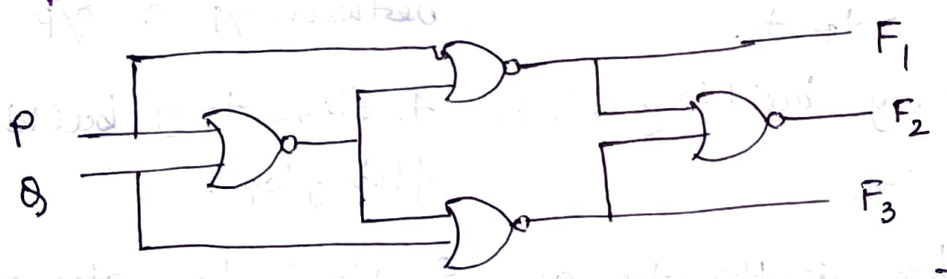
* Half Subtractor :



$P \oplus Q$.

P	Q	F_1	F_2
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

* Comparator :



$F_1 = \bar{P}Q$; $F_2 = \bar{P}\bar{Q} + PQ$; $F_3 = P\bar{Q}$.

		P=Q	P<Q	P>Q
P	Q	F_2	F_1	F_3
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

* Full Adder :

x	y	z	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	

S

z \ xy	00	01	11	10
0		1		1
1	1		1	

$x \oplus y \oplus z$

C

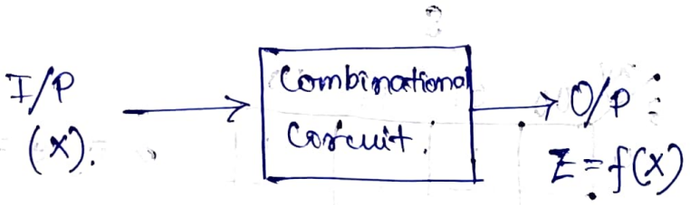
z \ xy	00	01	11	10
0			1	
1		1	1	1

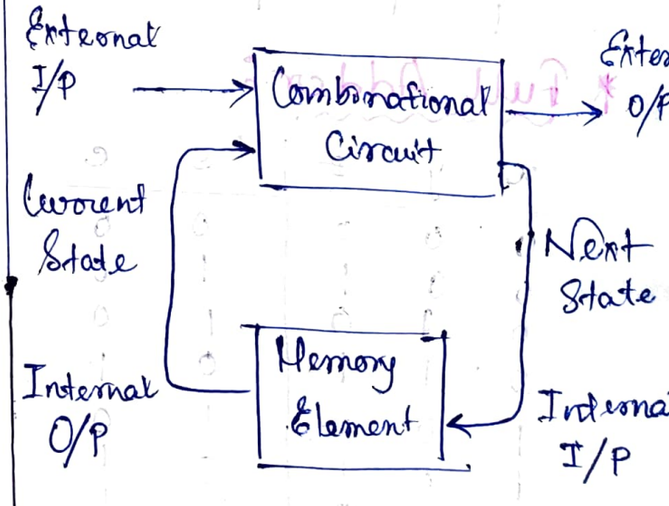
$xy + yz + zx$.

$= z(x \oplus y) + xy$.

* Logic circuits for digital systems may be combinational or sequential.

Combinational Circuit	Sequential Circuit.
1. Time independent circuits that does not depend upon previous inputs to generate any output.	1. Circuits that are dependent on clock cycles & depends on present as well as past inputs to generate any output.
2. Speed is fast.	2. Speed is slow.
3. No feedback between input & output.	3. There's a feedback path between i/p & o/p.
4. Elementary building blocks - logic gates.	4. Elementary building blocks flip-flops.
5. Used for arithmetic as well as boolean operations.	5. Used for storing data.
6. Don't have capability to store any state.	6. Have capability to store any state or to retain earlier state.
7. eg. Encoder, Decoder, MUX, DeMUX.	7. eg. flip-flops, counters.





* BCD to XS-3 Code Converter.

Input BCD				Output XS-3 Code.			
A	B	C	D	w	x	y	z
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

. . . . . Don't care's

W

AB \ CD	00	01	11	10
00				
01		1	1	1
11	X	X	X	X
10	1	1	X	X

$W = A + BC + BD$

$= A + B(C + D)$

x

AB \ CD	00	01	11	10
00		1	1	1
01	1			
11	X	X	X	X
10		1	X	X

$x = B'C + B'D + BC'D'$

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

y

AB \ CD	00	01	11	10
00	1		1	
01	1		1	
11	X	X	X	X
10	1		X	X

$y = CD + C'D'$

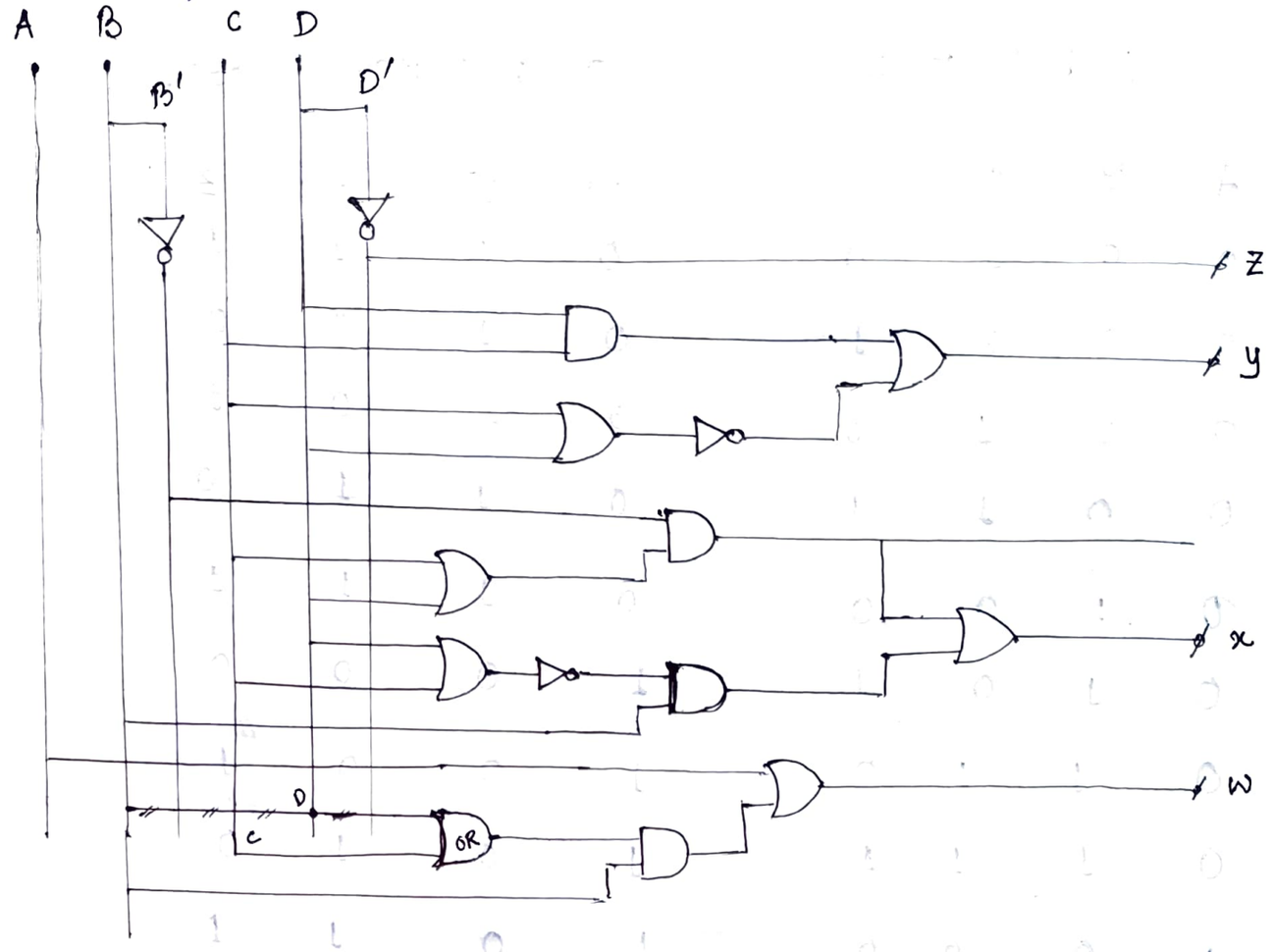
$= CD + (C+D)'$

z

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

$z = D'$

Logic diagram.



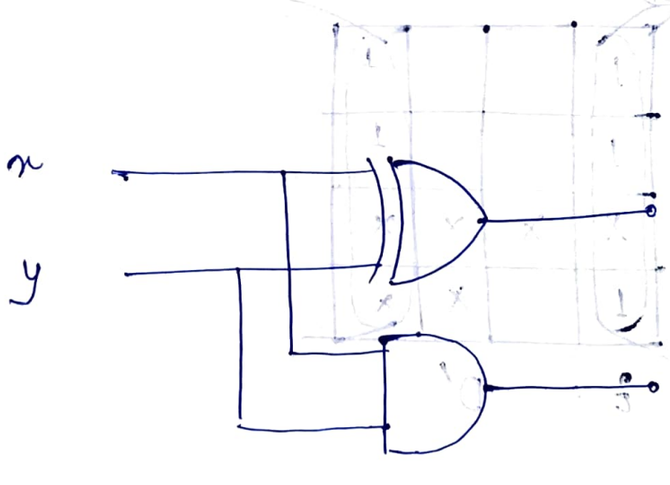
* Binary Adder / Subtractor :

☒ Half Adder : Performs addition of two bits.

Sum, $S = x'y + xy' = x \oplus y$

Carry, $C = xy$

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full Adder : Combinational circuit that forms the arithmetic sum of three bits.

Two of the input variables, denoted by x & y, represent the two significant bits to be added. Third input, z, represents the carry from the previous lower significant position.

x	y	z	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

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

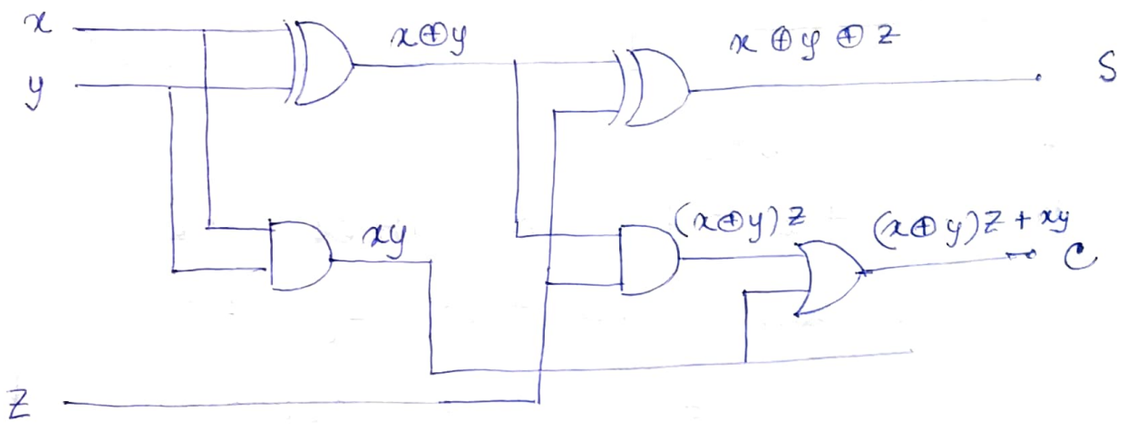
$$C = xy + xz + yz$$

by K'Map

$$\begin{cases} S = x \oplus y \oplus z. \\ C = xy + z(x \oplus y). \end{cases}$$

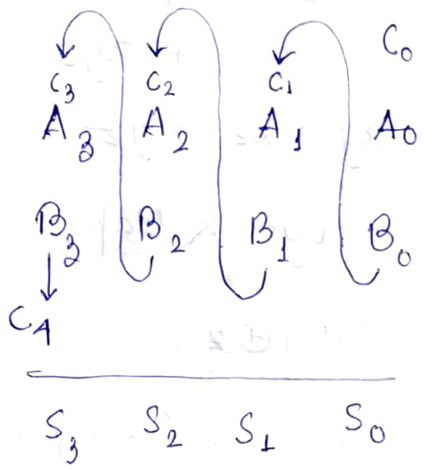
Binary Adder : Digital circuit that produces the arithmetic sum of two binary numbers.

It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder on the chain.

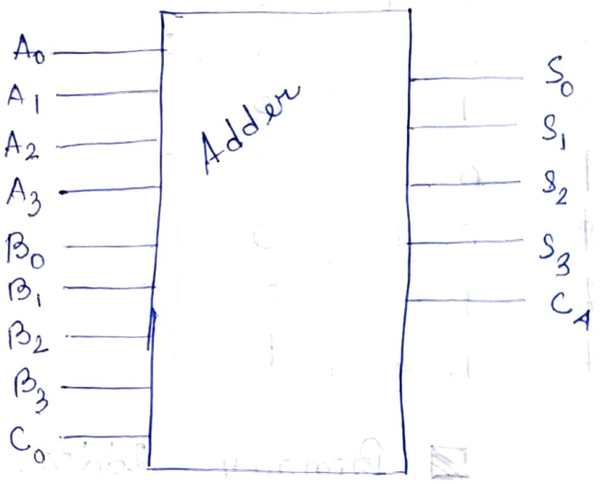


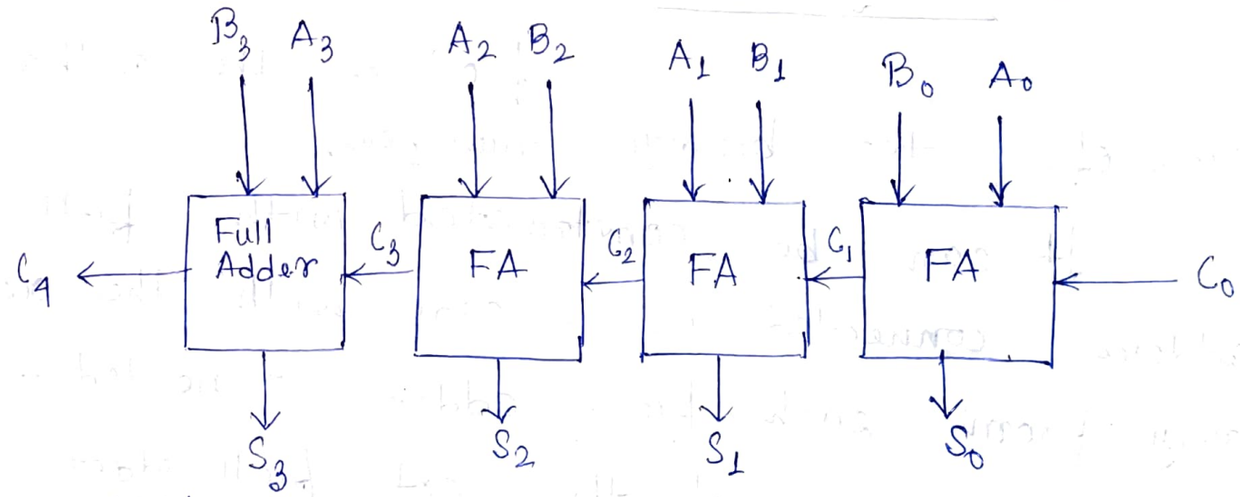
Full Adder using two half adders

→ Ripple Carry Adder :



4-bit adder





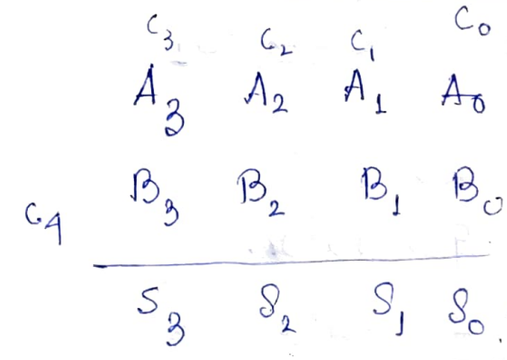
eg. 1011
+ 0011

	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output Carry	0	0	1	1	C_{i+1}

An n bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder.

Problem with ripple carry adder is every carry has to wait for its previous carry input, generating delay.

→ Carry Lookahead Adder :



$$C_1 = C_0 (A_0 \oplus B_0) + A_0 B_0$$

$$C_2 = C_1 (A_1 \oplus B_1) + A_1 B_1$$

$$C_3 = C_2 (A_2 \oplus B_2) + A_2 B_2$$

$$C_4 = C_3 (A_3 \oplus B_3) + A_3 B_3$$

Carry generate

$G_i = A_i B_i$

$P_i = A_i \oplus B_i$

Carry propagate.

$$C_1 = C_0 P_0 + G_0$$

$$C_2 = C_1 P_1 + G_1$$

$$C_3 = C_2 P_2 + G_2$$

$$C_4 = C_3 P_3 + G_3$$

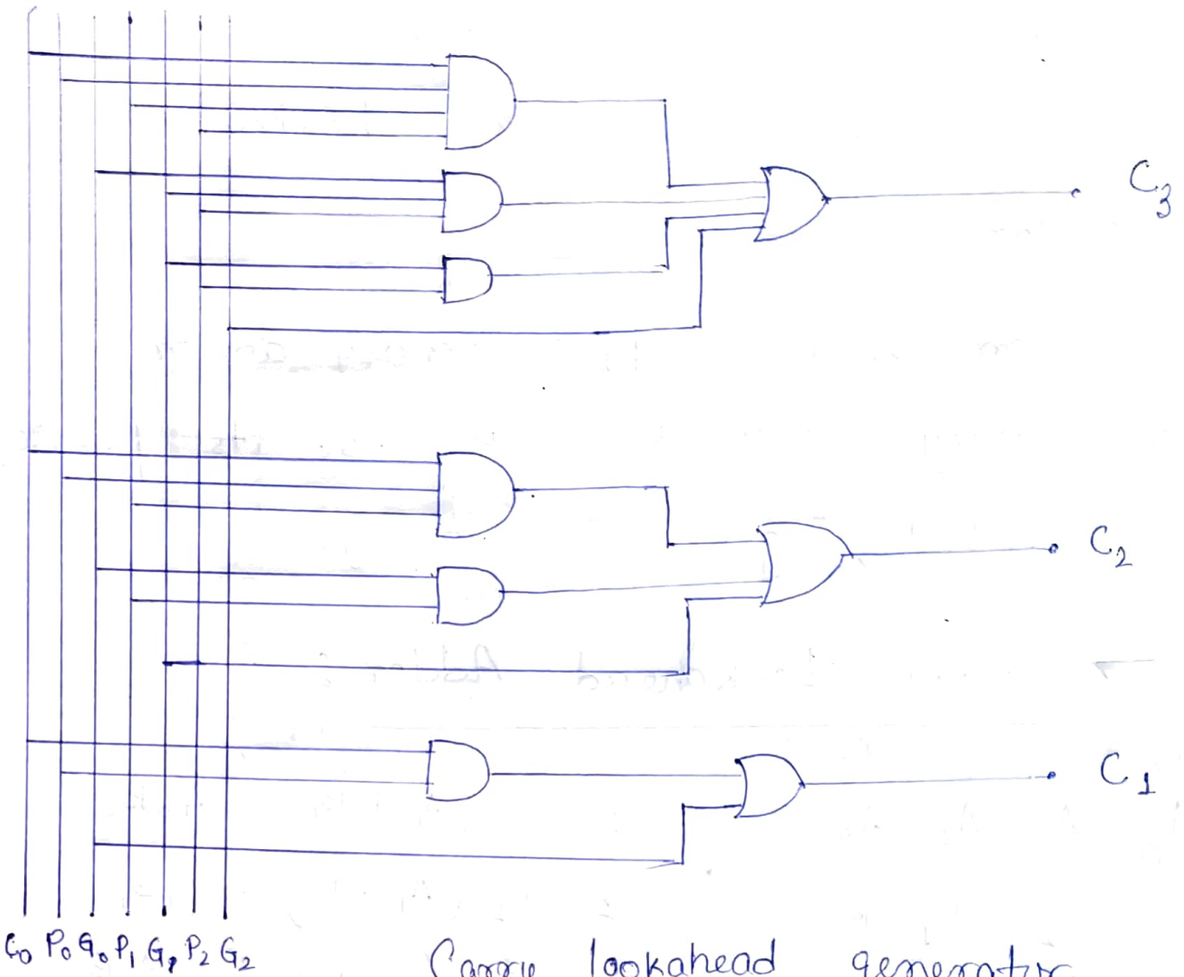
$$C_2 = C_1 P_1 + G_1 = (C_0 P_0 + G_0) P_1 + G_1$$

$$= C_0 P_0 P_1 + G_0 P_1 + G_1$$

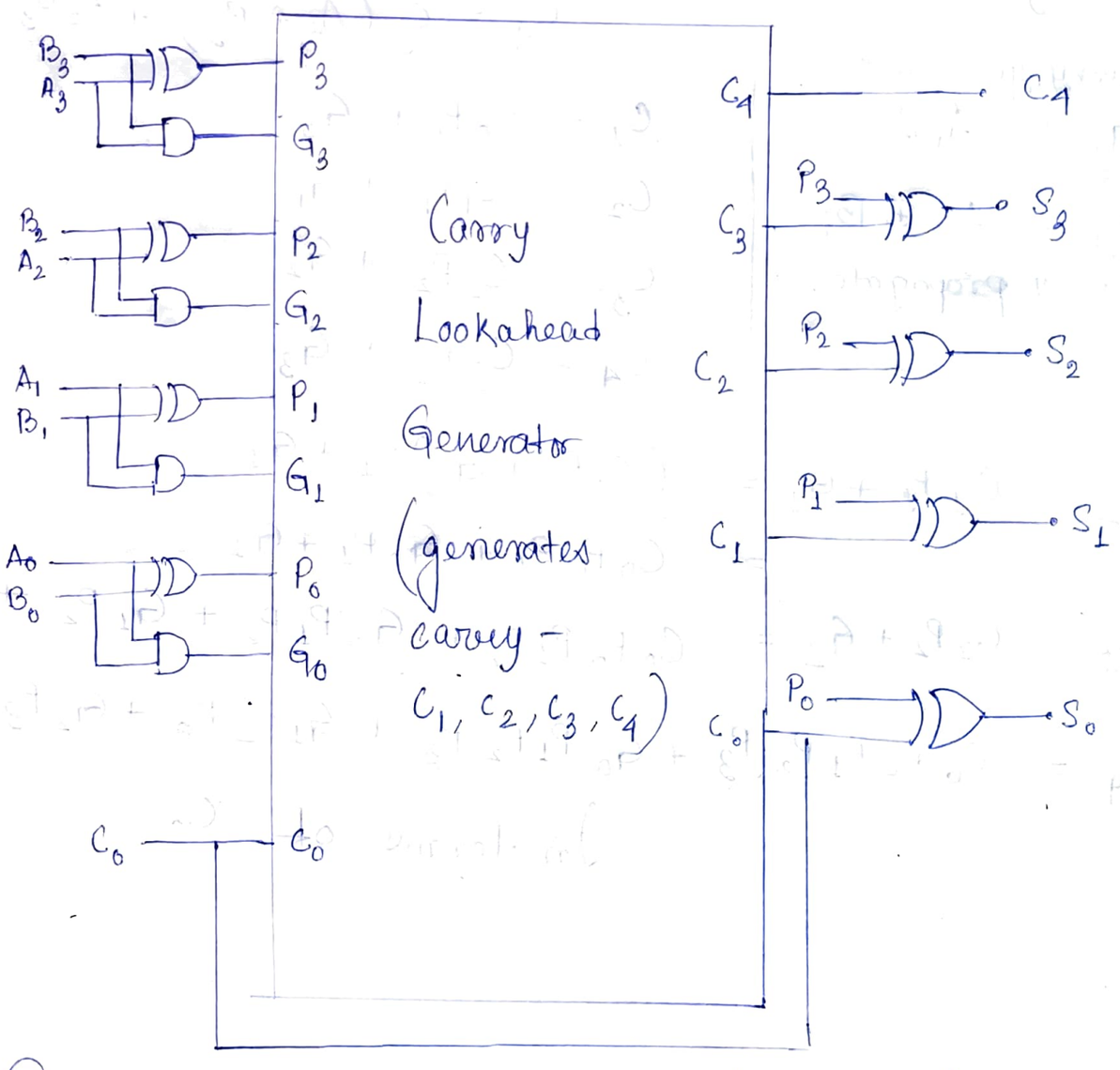
$$C_3 = C_2 P_2 + G_2 = C_0 P_0 P_1 P_2 + G_0 P_1 P_2 + G_1 P_2 + G_2$$

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

In terms of C_0



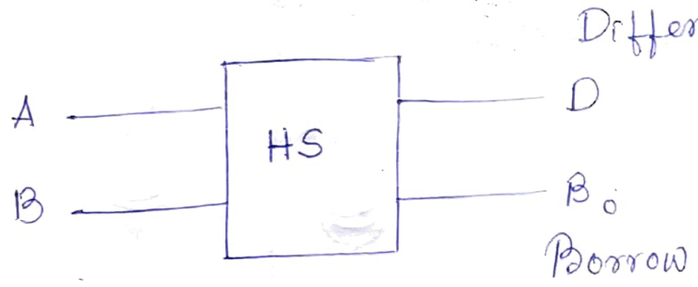
Carry lookahead generator.



Four bit adder with carry lookahead.

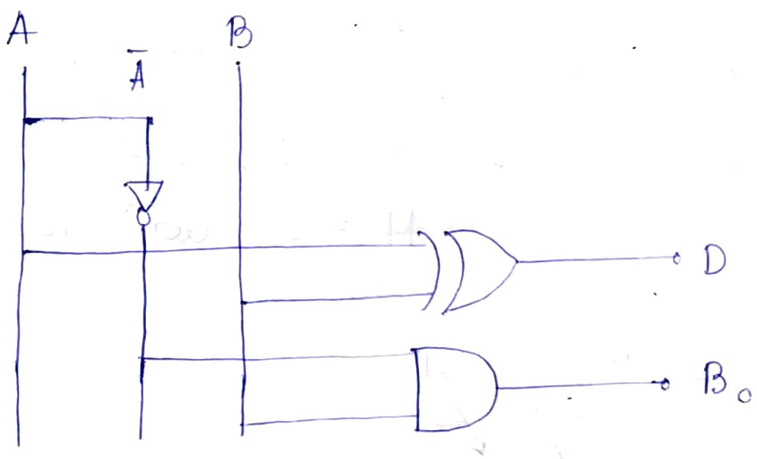
Half Subtractor.

A	B	D	B_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

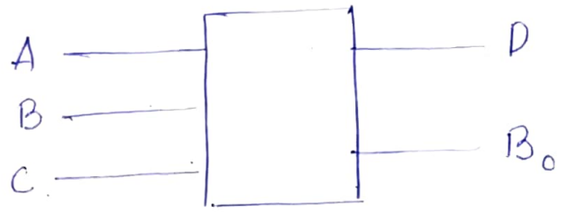


$D = A \oplus B$

$B_0 = \bar{A} B$

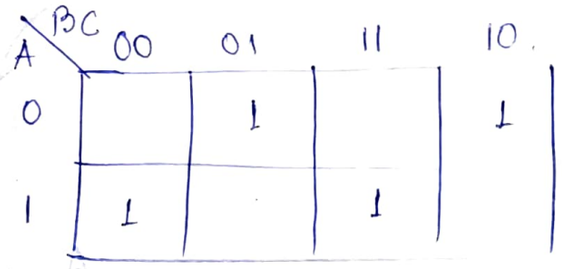


Full Subtractor.



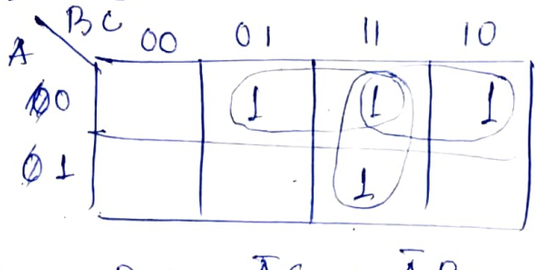
A	B	C	D	B_0
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

For D.



$D = A \oplus B \oplus C.$

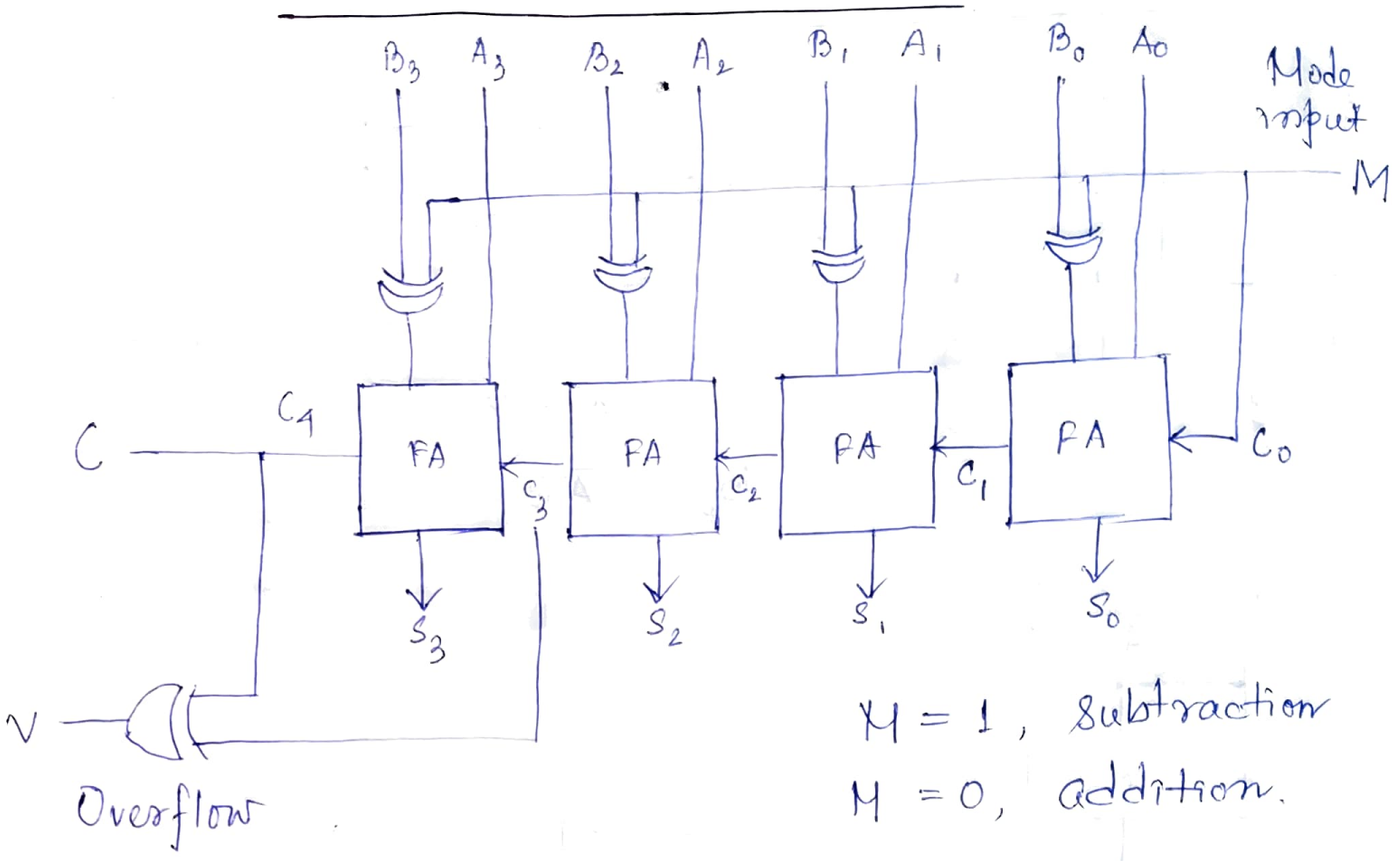
For B_0.



$B_0 = BC + \bar{A}C + \bar{A}B$

$B_0 = \bar{A}C + C(\overline{A \oplus B}).$

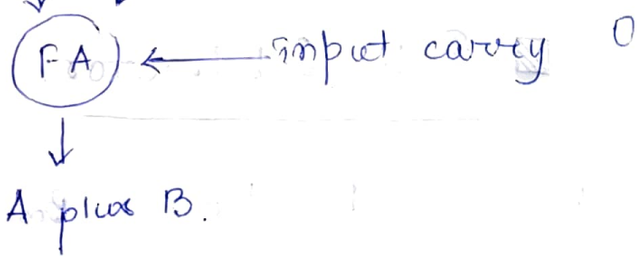
Four-bit adder-subtractor.



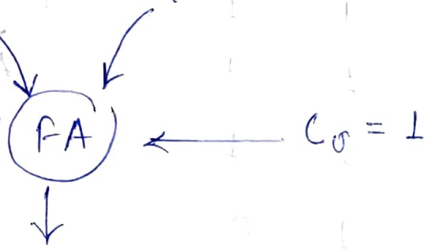
$M = 1$, Subtraction

$M = 0$, addition.

$M = 0$, $B_i \oplus 0 = B_i$ A_i



$M = 1$, $B_i \oplus 1 = B_i'$ A_i



A plus 2's complement of B.

* BCD Adder.

Decimal	Binary Sum C^*	S_3^*	S_2^*	S_1^*	S_0^*	BCD Sum C	S_3	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	0	Same as Binary Sum.
1	0	0	0	0	1	0	0	0	0	1	
2	0	0	0	1	0	0	0	0	1	0	
3	0	0	0	1	1	0	0	0	1	1	
4	0	0	1	0	0	0	0	1	0	0	
5	0	0	1	0	1	0	0	1	0	1	
6	0	0	1	1	0	0	0	1	1	0	
7	0	0	1	1	1	0	0	1	1	1	
8	0	1	0	0	0	0	1	0	0	0	
9	0	1	0	0	1	0	1	0	0	1	
10	0	1	0	1	0	1	0	0	0	0	$S_3^* \cdot S_1^* = 1$
11	0	1	0	1	1	1	0	0	0	1	
12	0	1	1	0	0	1	0	0	1	0	$S_3^*(S_2^* + S_1^*) = 1$
13	0	1	1	0	1	1	0	0	1	1	
14	0	1	1	1	0	1	0	1	0	0	
15	0	1	1	1	1	1	0	1	0	1	
16	1	0	0	0	0	1	0	1	1	0	$C^* = 1$
17	1	0	0	0	1	1	0	1	1	1	
18	1	0	0	1	0	1	1	0	0	0	
19	1	0	0	1	1	1	1	0	0	1	

Add 6 (0110) when —

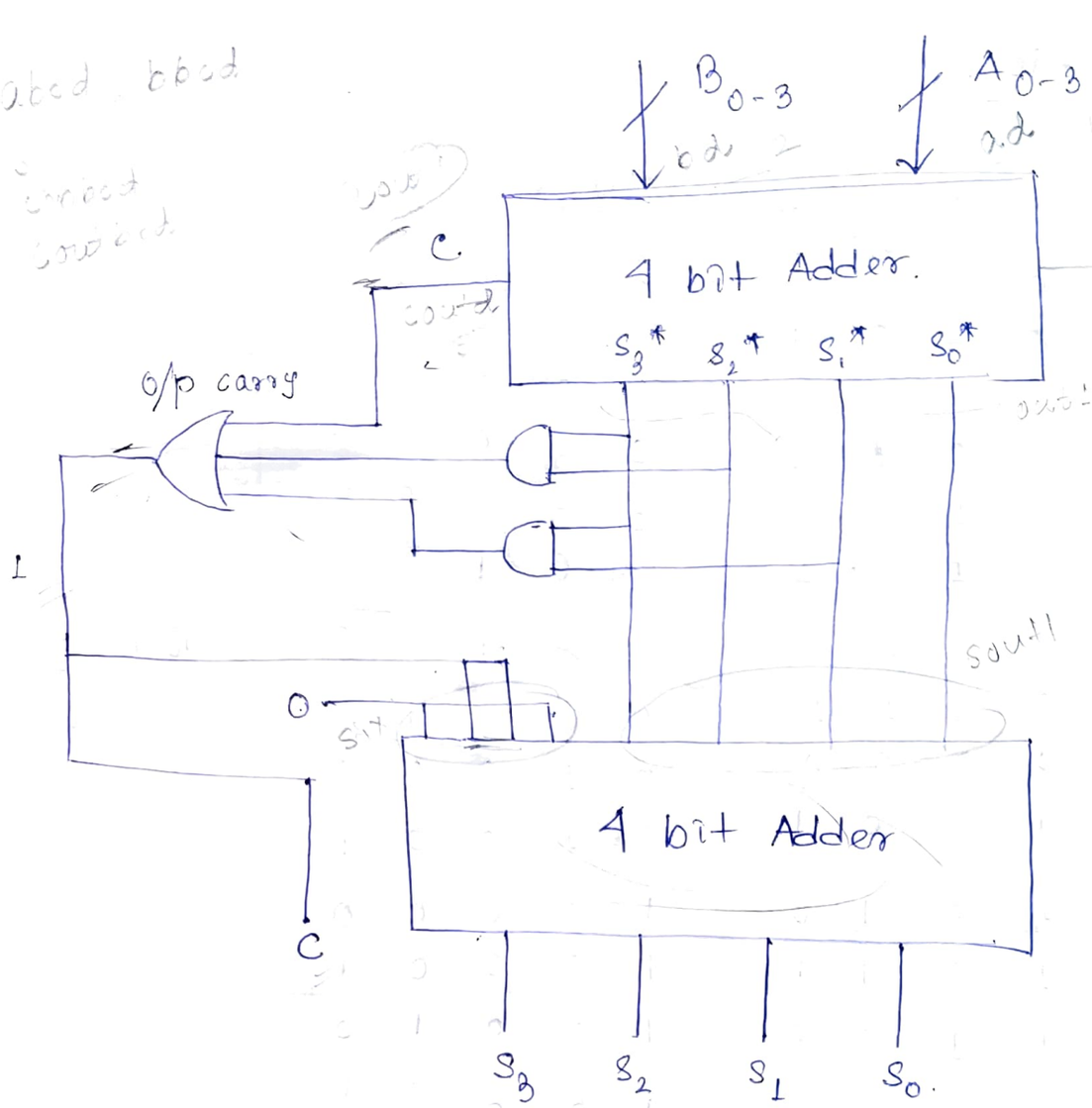
i) $C^* = 1$

ii) $S_3^* (S_2^* + S_1^*) = 1$

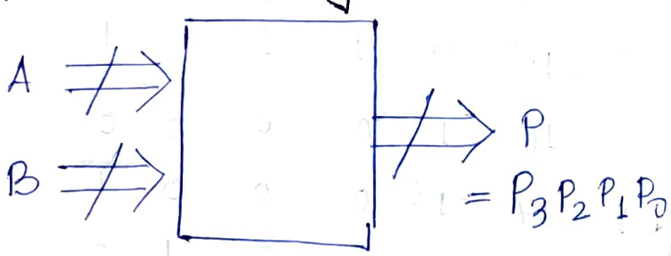
iii) $S_3^* \cdot S_1^* = 1$

Output carry expression =

$$C^* + S_3^* S_2^* + S_3^* S_1^*$$



* Binary Multiplier : (2 bit, using HA)



$A = A_1\ A_0$

$B = B_1\ B_0$

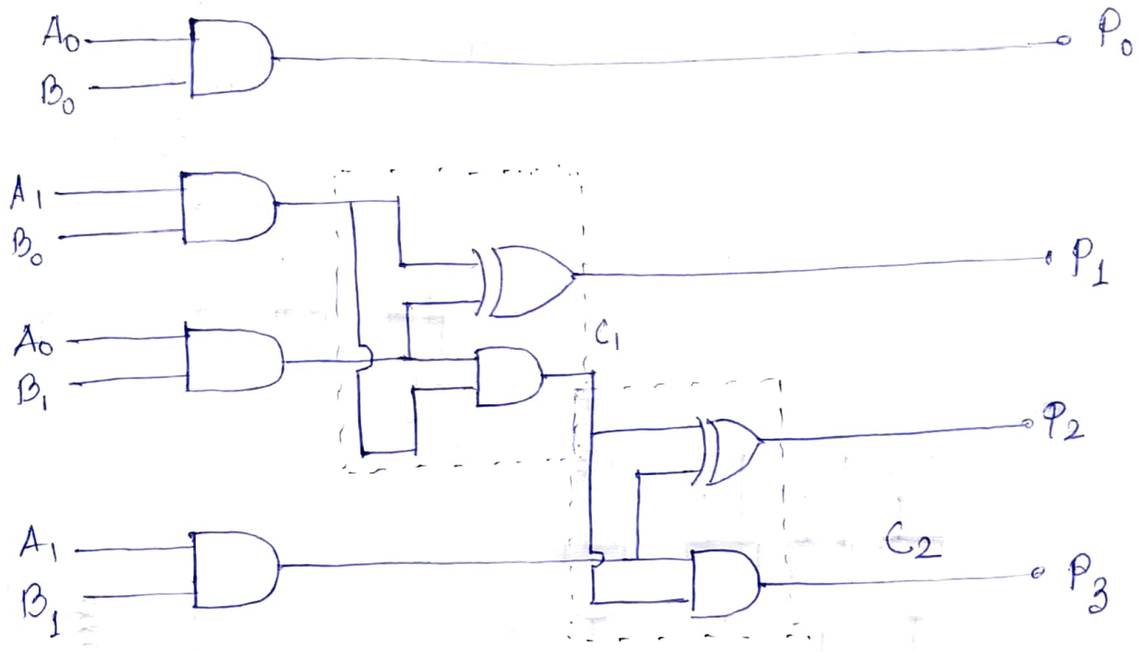
	A_1B_0	A_0B_0
C_2 ←	A_1B_1 (+ C_1) → C_2	A_0B_1 → C_1
	$(A_1B_1 + C_1)$	$(A_1B_0 + A_0B_1)$ A_0B_0

$P_0 = A_0B_0$

$P_1 = A_1B_0 + A_0B_1$ — addition (binary) Not OR.

$P_2 = A_1B_1 + C_1$

$P_3 = C_2$



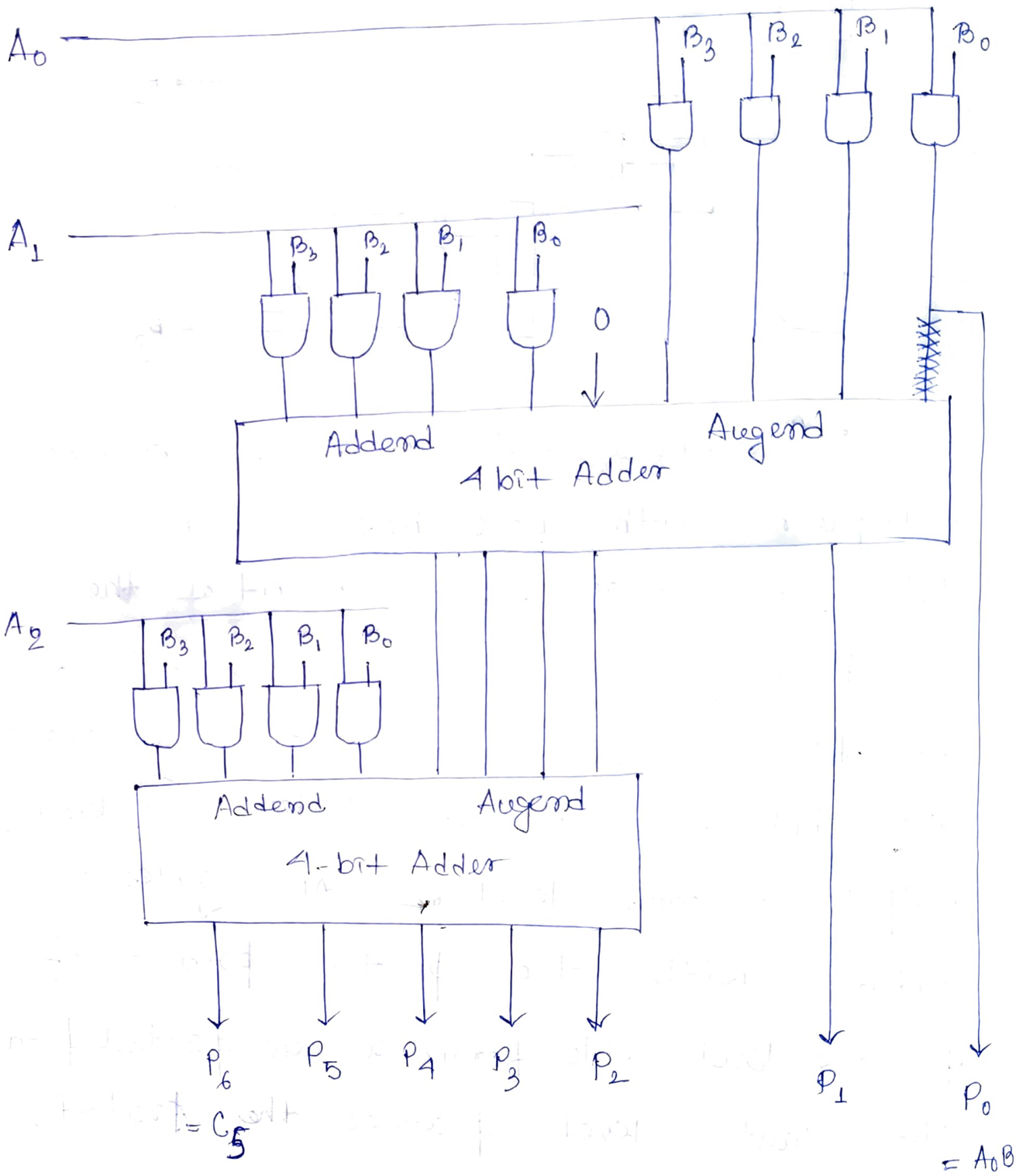
• A combinational circuit of binary multiplier with more bits can be constructed in a similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier. The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product. The last level produces the product.

• For J multiplier bits & K multiplicand bits, we need (J x K) AND gates & (J-1) K-bit adders to produce a product of (J+K) bits.

→ example

4×3 ~~3×4~~ bit multiplier.

multiplicand
$B_3 B_2 B_1 B_0$
multiplier
$A_2 A_1 A_0$.



* Magnitude Comparator.

$$A = A_3 A_2 A_1 A_0$$
$$B = B_3 B_2 B_1 B_0$$

Algorithm

i) Equal if each bit of the binary numbers is equal. The equality of each bit pair can be expressed with an XNOR function as

$$x_i = A_i B_i + A_i' B_i'$$

$x_i = 1$ only if A_i & B_i are equal.

$$(A = B) = x_3 x_2 x_1 x_0$$

$= 1$ only if x_i s are 1 i.e. all bit pairs are equal.

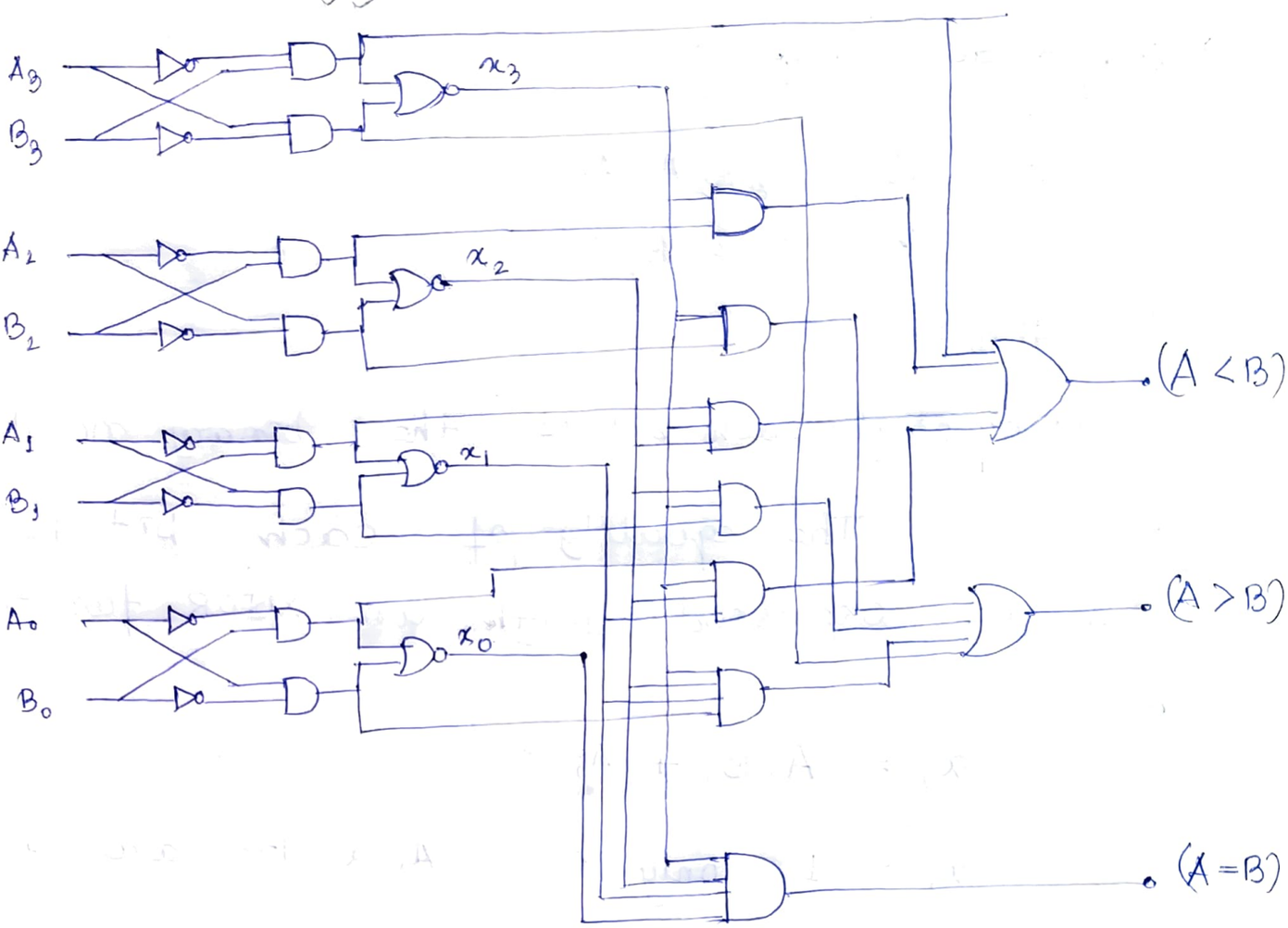
ii) To determine inequality start from the msb & look for inequality of bit pair. If bit pair has inequality —

a) $A_i = 1$, $B_i = 0$ } $A > B$

b) $A_i = 0$, $B_i = 1$ } $A < B$.

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0.$$



* Decoders.

→ A decoder has a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.

* Encoder.

→ Combinational circuit that performs reverse operation of decoders. It has max 2^n input lines & n output lines.

* Multiplexer. (MUX)

→ A multiplexer is an electronic circuit or switch that can connect one out of n inputs to output.

It has maximum 2^n data inputs, n selection lines & single output line. Since there are n selection lines, there will be 2^n possible combinations of zeros & ones.

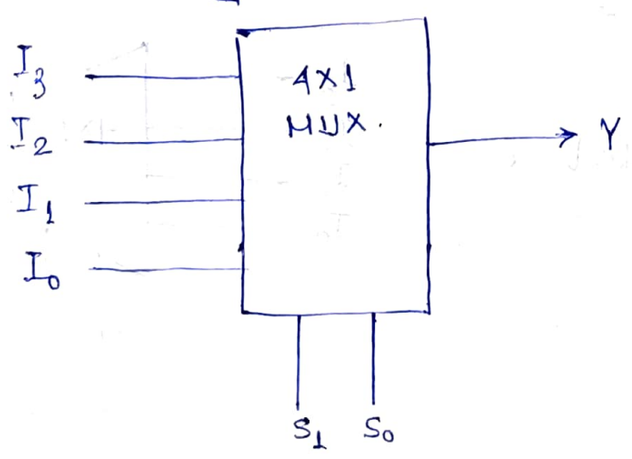
→ It cannot change the logical level of the input ; it only provides the connection between i/p & o/p.

→ MUX is functionally complete, i.e. all boolean functions can be realised using MUX without any other gates.

→ Selection lines are used to decide the input line.

- **4X1 MUX** : 4 data inputs - I_3, I_2, I_1, I_0; & selection lines - S_0, S_1 & 1 output line Y.

Block diagram —



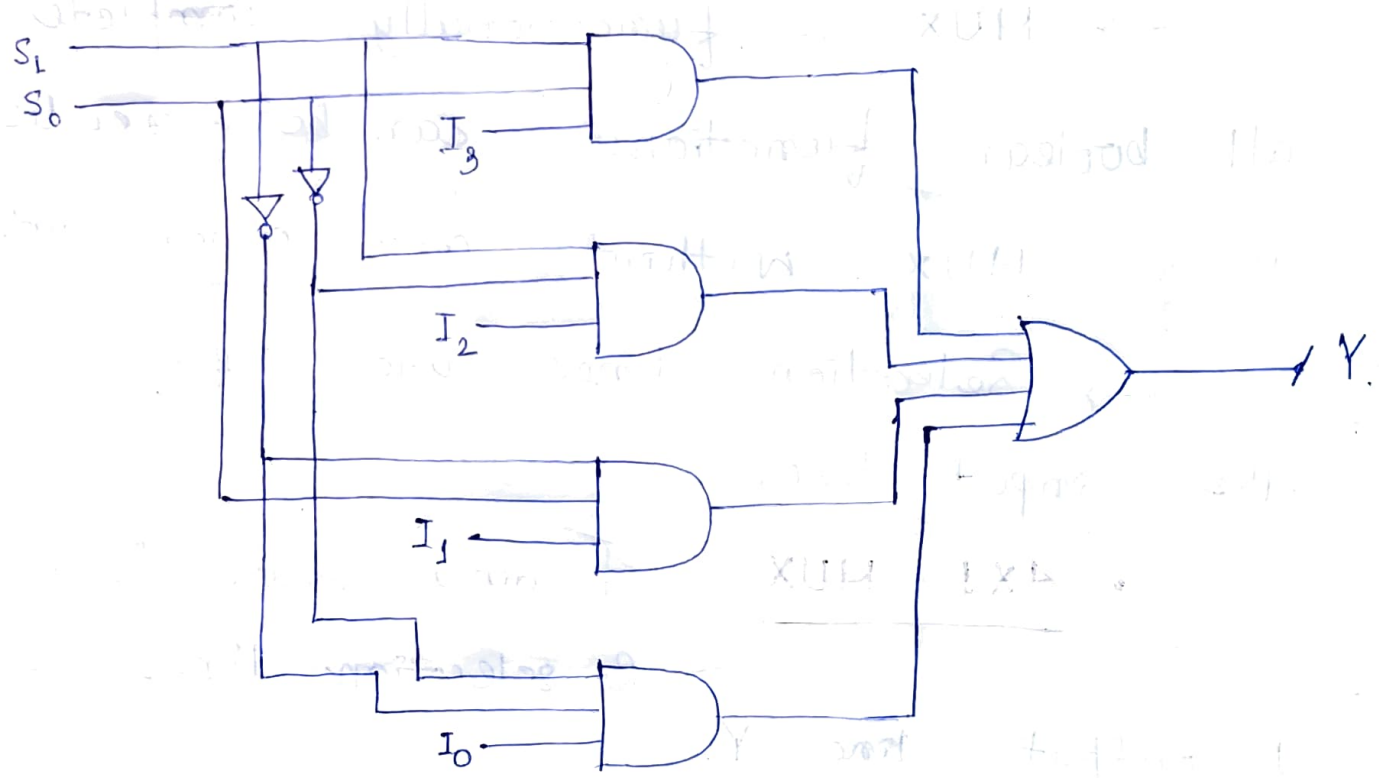
Characteristic table for 4×1 MUX. →

Selection lines.		Output.
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3.

Characteristic equation –

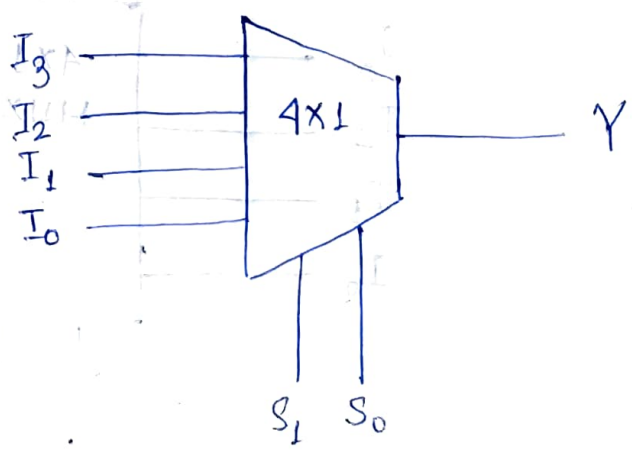
$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3.$$

Circuit Diagram.



4×1 MUX.

Symbol :



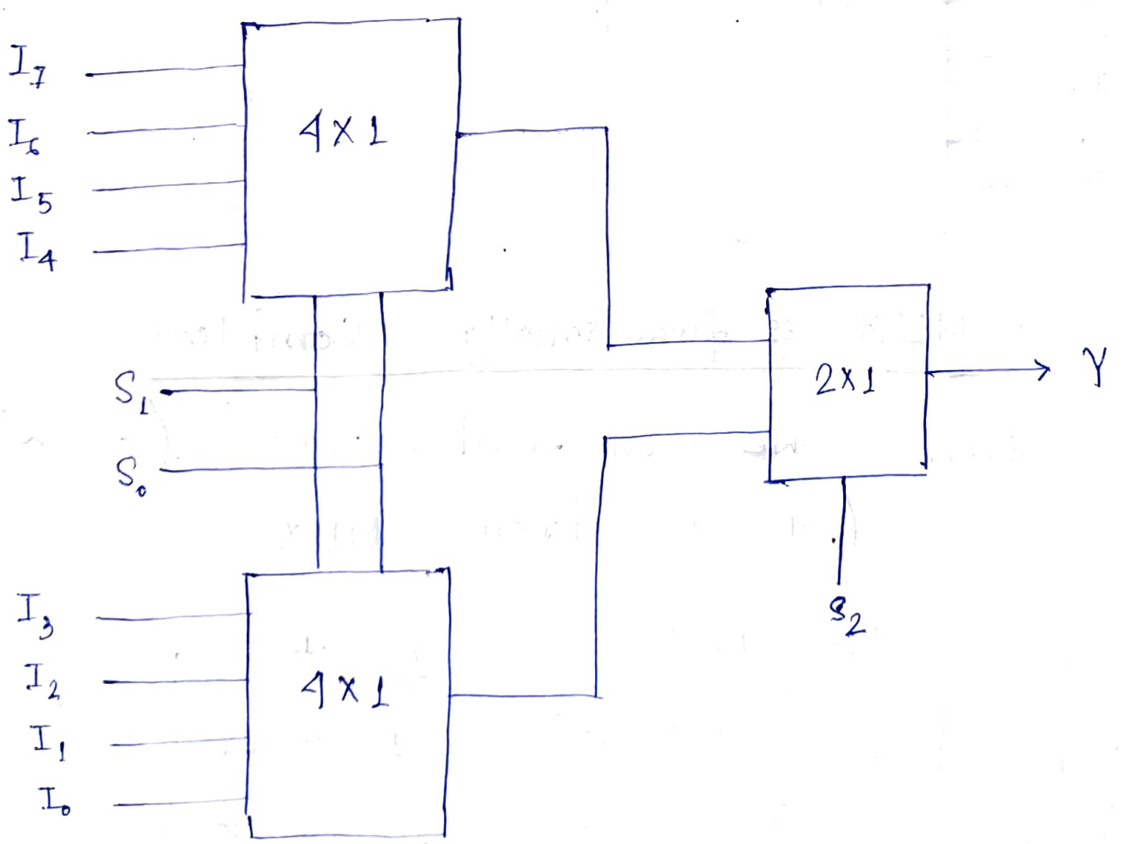
• 8×1 MUX :

We need two 4×1 & one 2×1 MUX to implement 8×1 MUX.

Characteristic Table :

S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

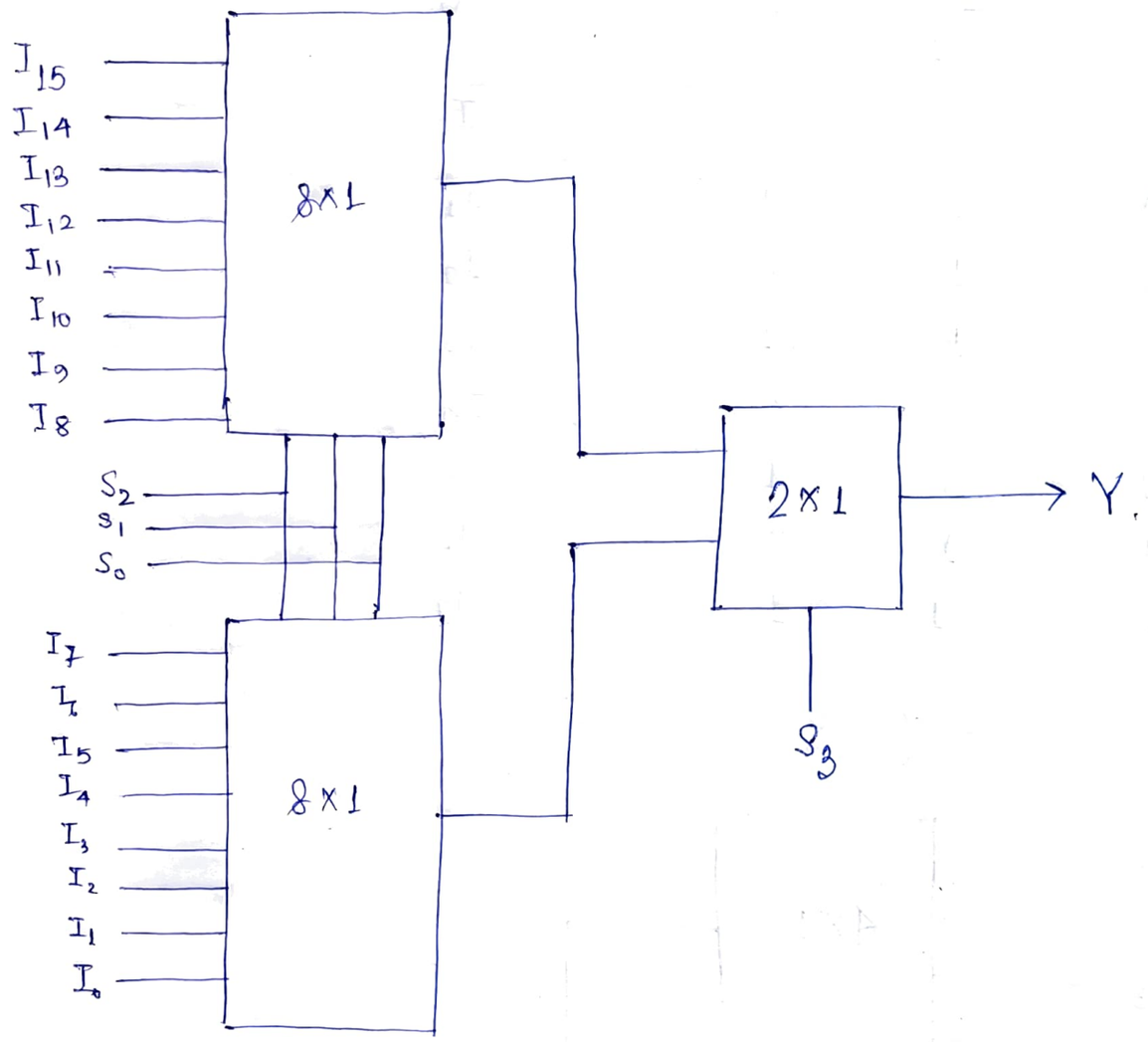
Block diagram : (using 4×1)



• 16 × 1 MUX : (using 8×1)

We require two 8×1 & one 2×1 MUX to implement 16×1 MUX.

Block diagram.



• MUX is functionally Complete

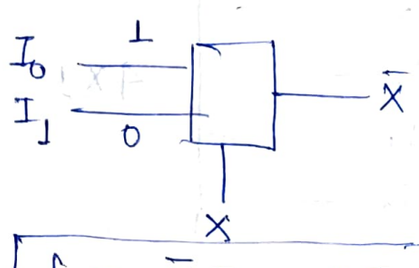
i.e. we can implement $(\cdot, \sim)$ or $(+, \sim)$ from MUX.

Implementing Not.

X	$\bar{X}$
0	1
1	0

$f = \bar{X} \cdot 1 + X \cdot 0$

$= \bar{X}.$

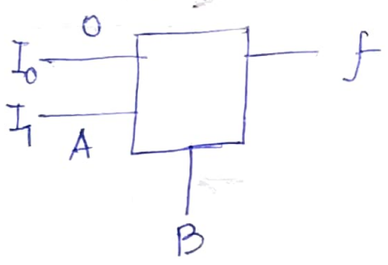


So, we can implement NOT.

$$f = \bar{X} I_0 + X I_1$$

Implementing AND

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



$$f = \bar{B} I_0 + B I_1$$

$$f = \bar{B}\,(0) + B \cdot A = AB$$

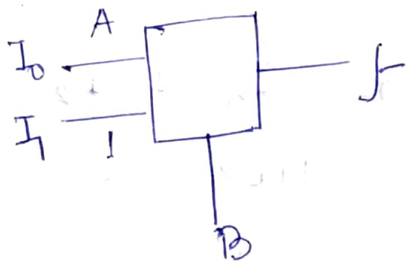
So, we can implement AND

$AB = BA$

$= \bar{B}(0) + B \cdot A.$

Implementing OR

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



$$f = \bar{B} I_0 + B I_1.$$

$$f = \bar{B}\,(A) + B \cdot (1) = \bar{B} \cdot A + B\,(A + \bar{A})$$

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

$$= A + B.$$

So, we can implement OR.

$A + B =$

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

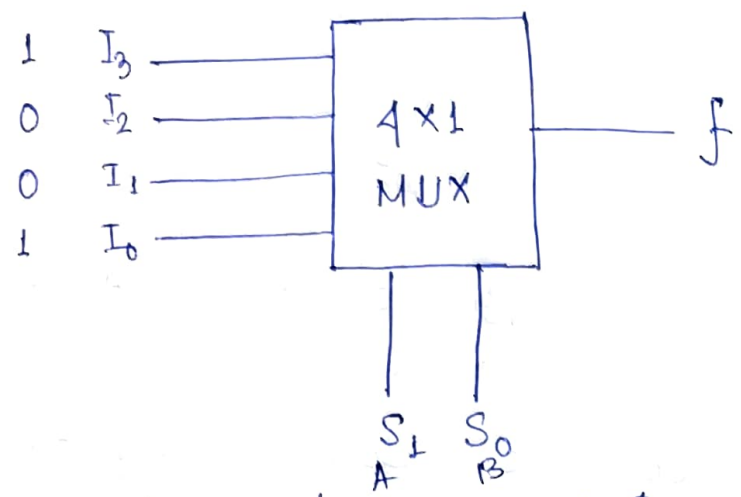
$\bar{B}A + B(A + \bar{A})$

So, MUX is functionally complete.

- # Implementing functions with MUX.

→ Implementing XNOR. (2 inputs).

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



$Y = A'B' + AB.$

$= A'B'(1) + A'B(0) +$

$AB'(0) + AB(1).$

$f = S_1'S_0'I_0 + S_1'S_0I_1 +$

$S_1S_0'I_2 + S_1S_0I_3$

When $I_3 = 1, I_2 = 0, I_1 = 0, I_0 = 1$, the 4x1 MUX will act like XNOR of S_1 and S

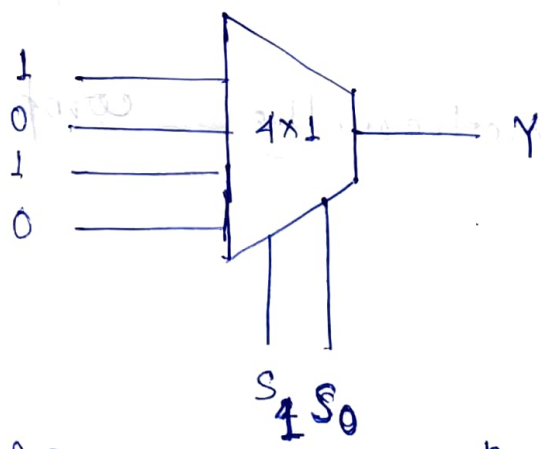
→ Implementing 2 i/p function.

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

$Y = A'B' + AB'.$

$= A'B'(1) + A'B(0) +$

$AB'(1) + AB(0).$



→ If we have $2^n \times 1$ MUX, we can implement n variable function (using only one $2^n \times 1$ MUX). We can implement ~~8x1~~ more than one of 3 variables by 4x1 MUXs.

→ Implementing 3 variable fun^n.

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

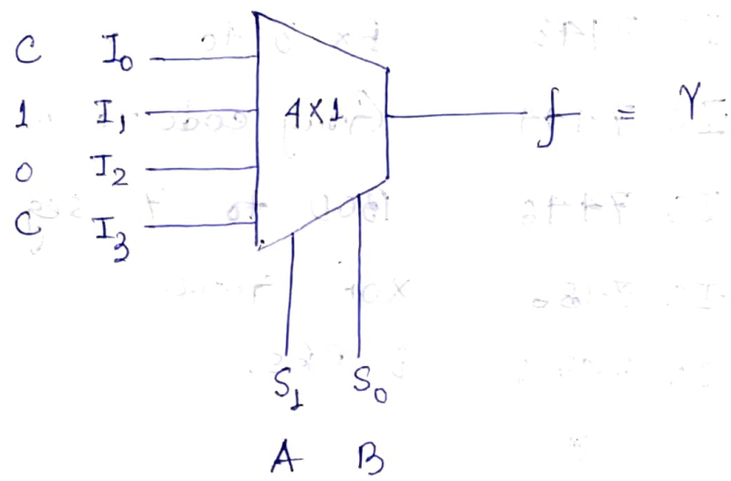
$Y = A'B'C + A'BC' + A'BC + ABC$

$= A'B'(C) + A'B(C' + C) + AB'(0) + AB(C)$

$(C' + C) = 1$

We need to get the input lines by analysing the Y as

$f = I_0 S_1' S_0' + I_1 S_1' S_0 + I_2 S_1 S_0' + I_3 S_1 S_0$

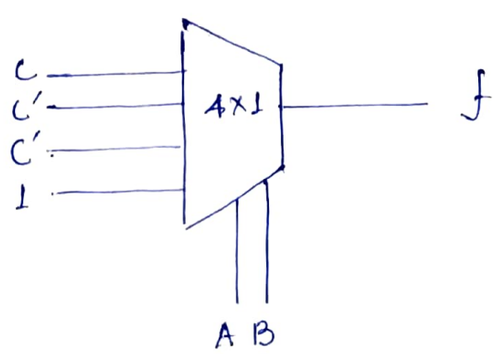


→ To implement higher variable functions we can use lower level MUX s in sequence. Then the input lines will come from MUX output lines.

→ $f(A,B,C) = \Sigma(1,2,4,6,7)$. Implement.

$f = A'B'C + A'BC' + AB'C' + ABC' + ABC.$

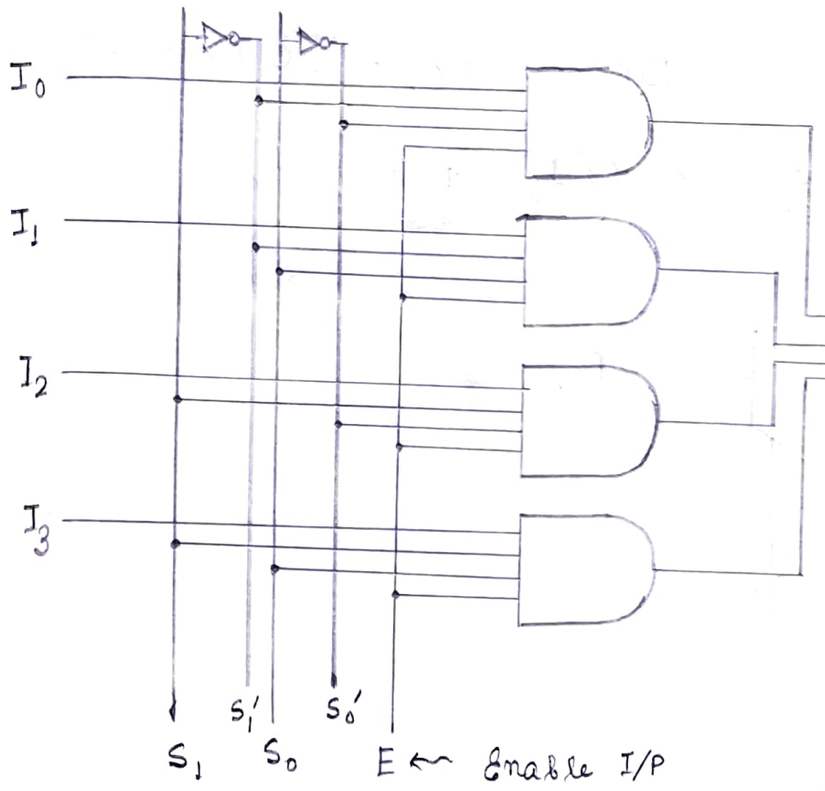
$= A'B'(C) + A'B(C') + AB'(C') + AB(1).$



Design and Synthesis of Combinational Circuits. (Contd.)

* Multiplexer (Contd.)

• MUX with enable input.

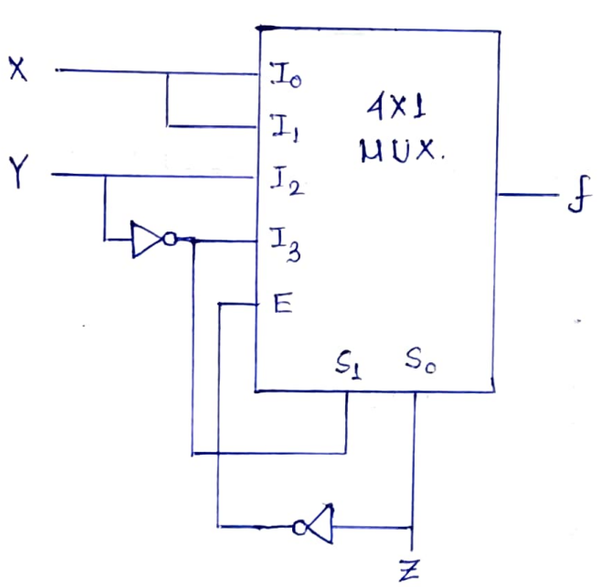


Enable is generally a on/off switch for MUX here.

$$f = (I_0 S_1' S_0' + I_1 S_1' S_0 + I_2 S_1 S_0' + I_3 S_1 S_0)\, E.$$

MUX works only when $E = 1$.

→ eg. Q. Minimise the function represented by following MUX.



$$f = E(I_0 S_1' S_0' + I_1 S_1' S_0 + I_2 S_1 S_0' + I_3 S_1 S_0).$$

$$= Z'(XYZ' + XYZ + YY'Z' + Y'Y'Z)$$

$$= Z'(XY + Y'Z).$$

$$= XYZ' \quad (Ans).$$

→ 8. Implement

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

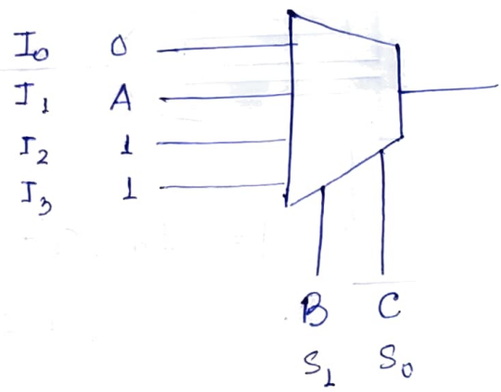
a) $S_1 = B$, $S_0 = C$

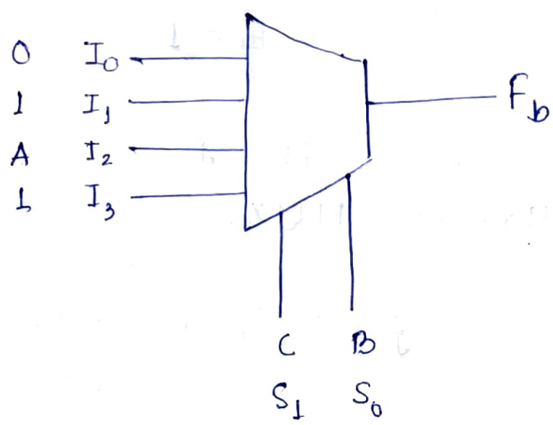
b) $S_1 = C$, $S_0 = B$.

$$f = A'BC' + A'BC + AB'C + ABC' + ABC$$

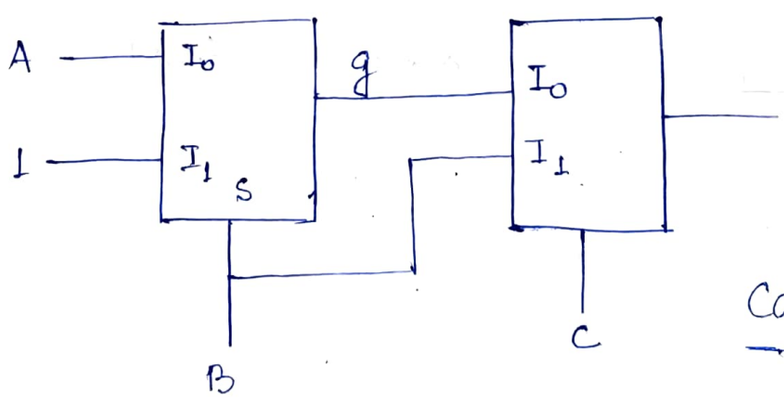
$$= B'C'(0) + B'C(A) + BC'(A'+A) + BC(A+A')$$

$$= B'C'(0) + B'C(A) + BC'(1) + BC(1).$$





→ 8. What is the function in canonical SOP?



Cascading MUX

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

$f = \bar{C} I_0 + C I_1$

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

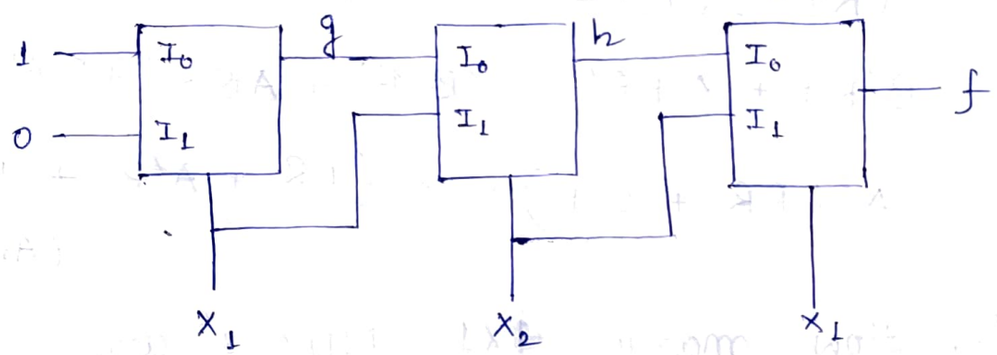
$= A\bar{B}\bar{C} + B\bar{C} + BC.$

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

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

$F(A, B, C) = \Sigma(2, 3, 4, 6, 7).$

→ Q.



$g = X_1'(1) + X_1(0) = X_1'$

$h = X_2'(I_0) + X_2 I_1 = X_2'X_1' + X_2 X_1.$

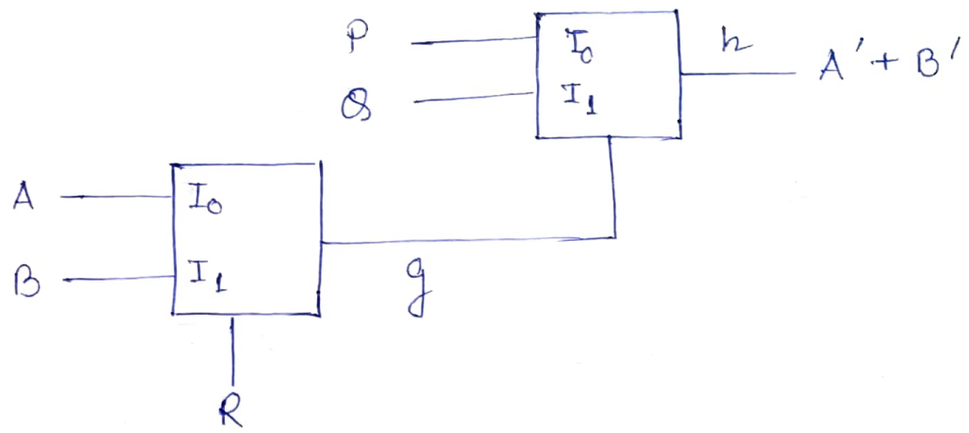
$f = X_1'(I_0) + X_1(I_1)$

$= X_1'(X_2'X_1' + X_2X_1) + X_1X_2$

$= X_1'X_2' + X_1X_2.$

$\therefore f = X_1 \odot X_2.$

→ Q.



$g = R'A + RB.$

$h = (R'A + RB)'P + (R'A + RB)Q.$

$= (R + A')(R' + B')P + AQR' + BQR$

$= B'RP + A'PR' + A'B'P' + AQR' + BQR.$

$= A'(PR' + B'P) + B'(PR + A'P) + BQR.$

(Ans)

→ Q. How many 4×1 MUX s are required to implement 32×1 MUX?

⇝ 4×1 MUX can cover 4 input lines.

To cover 1 line we need $\frac{1}{4}$ MUX.

To cover 32 input lines at level 1, we need $32 \times \frac{1}{4} = 8$ MUX. (1st level)

To cover 8 lines, $8 \times \frac{1}{4} = 2$ MUX (2nd level)

To cover 2 lines, $2 \times \frac{1}{4} = \frac{1}{2}$ MUX

$\equiv 1$ MUX

∴ We require $(8 + 2 + 1) = 11$ MUX (4×1).

→ Q. 64×1 MUX using 4×1 MUX.

64 lines — $64 \times \frac{1}{4} = 16$

16 lines — $16 \times \frac{1}{4} = 4$.

4 lines — $4 \times \frac{1}{4} = 1$

} = 21 MUX we need.

→ Generalisation for expansion of MUX.

M×1 MUX using N×1 MUX ←

There are M input lines.

N×1 MUX can cover N lines

1 line can be covered by $\frac{1}{N}$ MUX.

For first level,

M input lines can be covered by $\frac{M}{N}$ MUX(N×1)

These M/N MUXs (N×1) will create M/N o/p lines

For second level,

M/N i/p lines can be covered by M/N^2 MUX(N×1)

for third level,

M/N^2 i/p lines can be covered by M/N^3 MUX(N×1)

⋮

M/N^{K-1} i/p lines can be covered by M/N^K MUX(N×1)

Now, $M/N^K \leq 1 \Rightarrow \boxed{K \geq \log_N M}$

→ So, there are $\log_N M$ levels, or more.

$\therefore K = \lceil \log_N M \rceil$

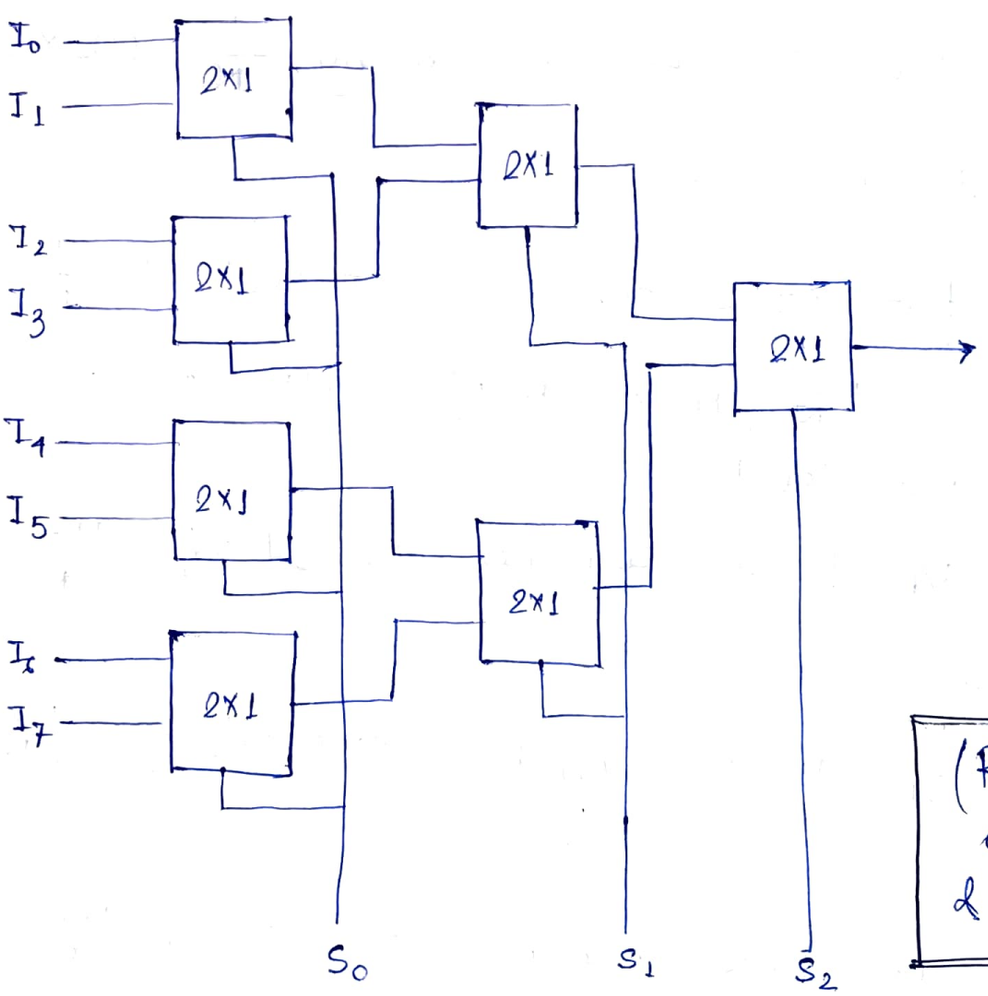
at least $\lceil\ \rceil$
at most $\lfloor\ \rfloor$

→ No. of MUXs needed =

$$\sum_{K=1}^{\lceil \log_N M \rceil} \left(M / N^K \right).$$

→ Assigning select lines while expanding MUX :

8 x 1 using 2 x 1.



(Remember the order of S & I)

example - $S_2 S_1 S_0 \equiv 1\,1\,0 \rightarrow I_6$

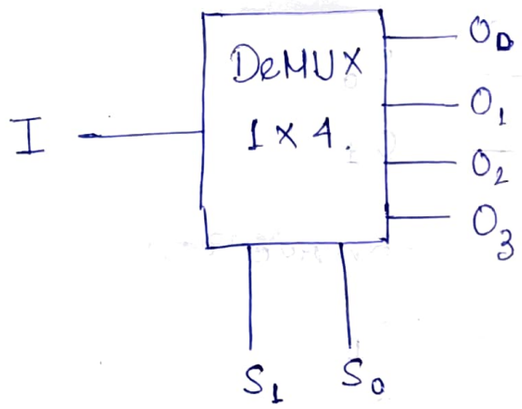
* Demultiplexer. (DeMUX).

→ It performs opposite operation of MUX. It has one input & 2^n outputs where n is no. of select lines.

→ It is derived from MUX by joining all the inputs together and removing OR gate.

Characteristic Table –

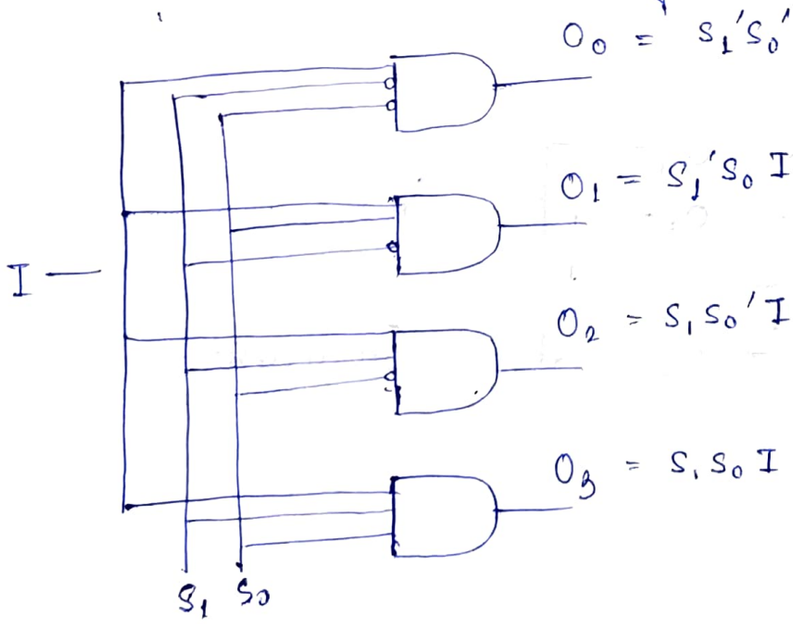
S_1	S_0	O_0	O_1	O_2	O_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I.



→ Switching by DeMUX.

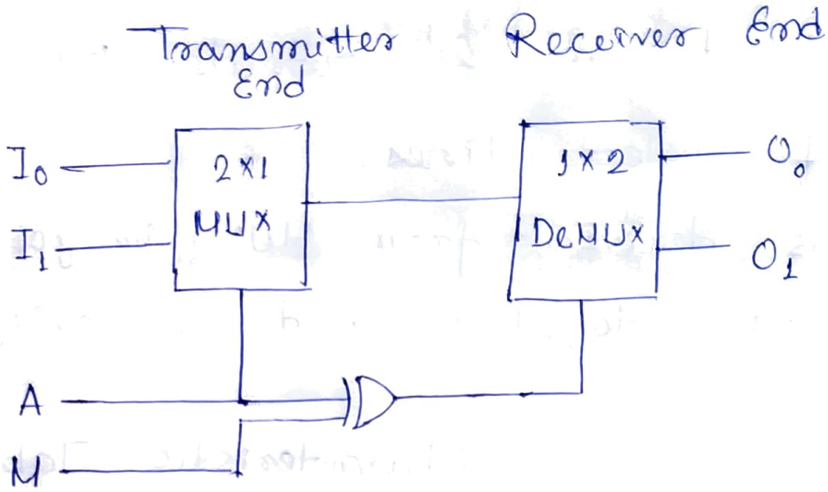
DeMUX is mainly used in construction of switches.

→ Circuit Diagram for DeMUX –



→ Switching operation.

(using 2×1 MUX & 1×2 DeMUX)



When M = 0 → Straight connection

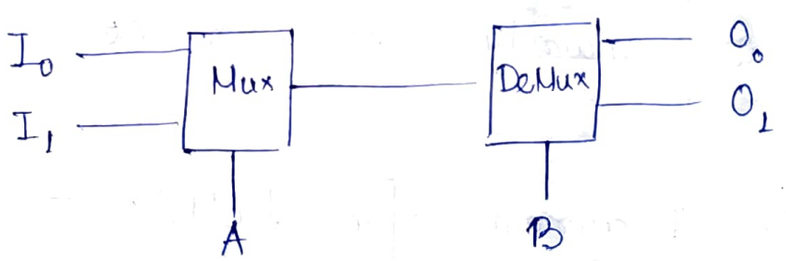
I_0 — O_0

I_1 — O_1

When M = 1 → Cross connection

I_0 → O_1

I_1 — O_0.



A	B
0	0
0	1
1	0
1	1

I_0 — O_0

I_0 — O_1 → Cross.

I_1 — O_0

I_1 — O_1 → Straight

* Decoder

→ Decoder is a combinational circuit that has n input lines and maximum of 2^n output lines.

One of the outputs will be active high based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the min terms of n input variables, when it is enabled.

✓→ A DeMUX can be converted to a decoder by setting I = 1 & making the select lines as inputs.

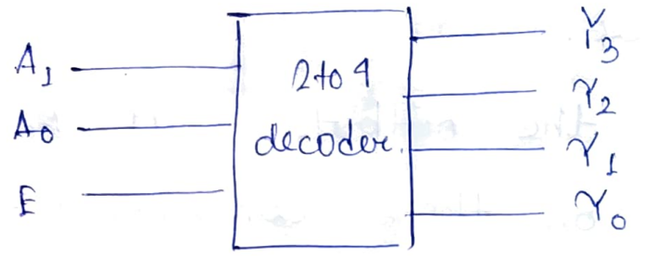
→ Since a decoder provides all the min-terms, we can implement any function in canonical SOP (AND-OR Realisation) using OR gate.

→ If all the AND gates of decoder are replaced with NAND gates, the decoder becomes active low. (AND-OR ↔ NAND-NAND)

→ 2 to 4 decoder.

Two inputs – A_1, A_0

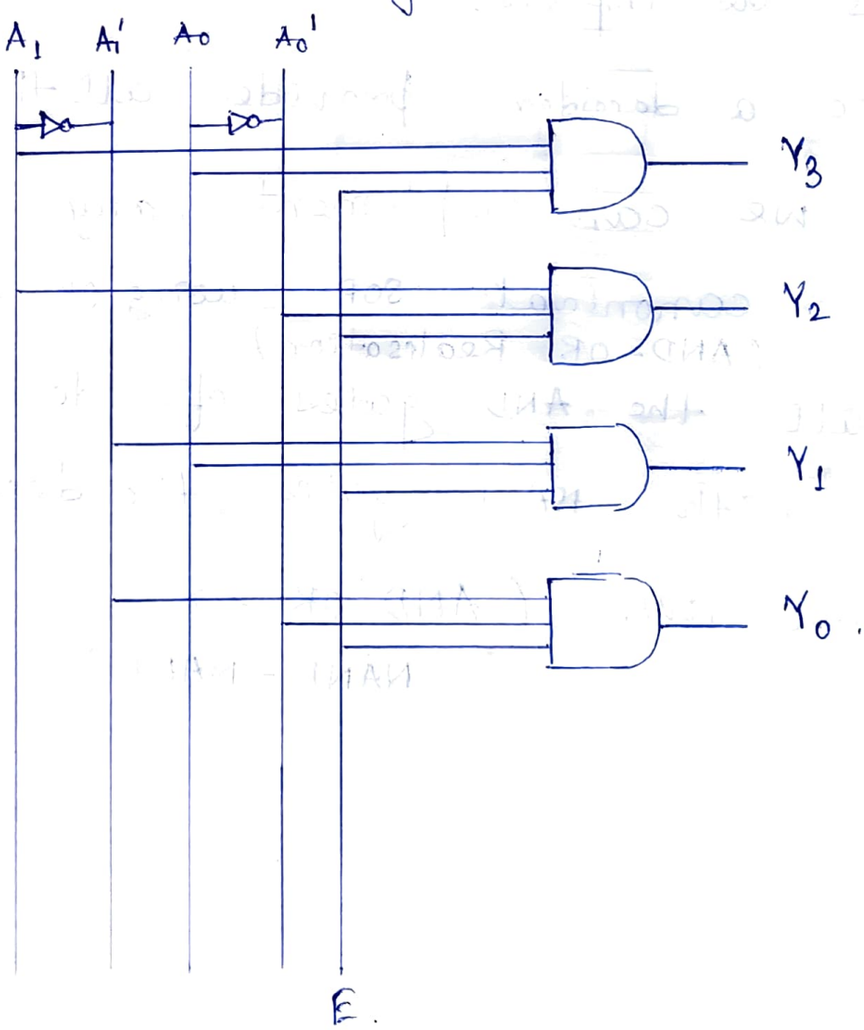
Four outputs – Y_3, Y_2, Y_1, Y_0



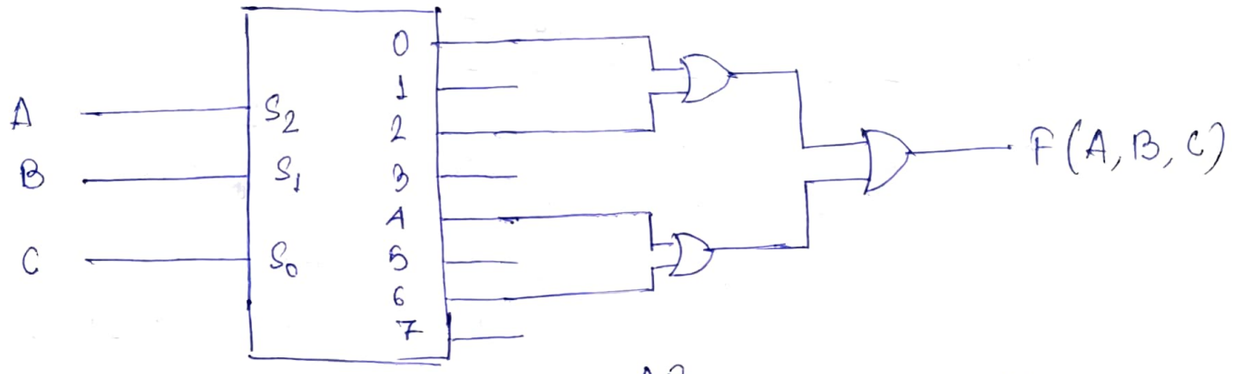
Enable	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$Y_3 = E.A_1.A_0 \mid Y_2 = E.A_1.A_0' \mid Y_1 = E.A_1'A_0 \mid Y_0 = EA_1'A_0'$

circuit diagram –



→ Q.

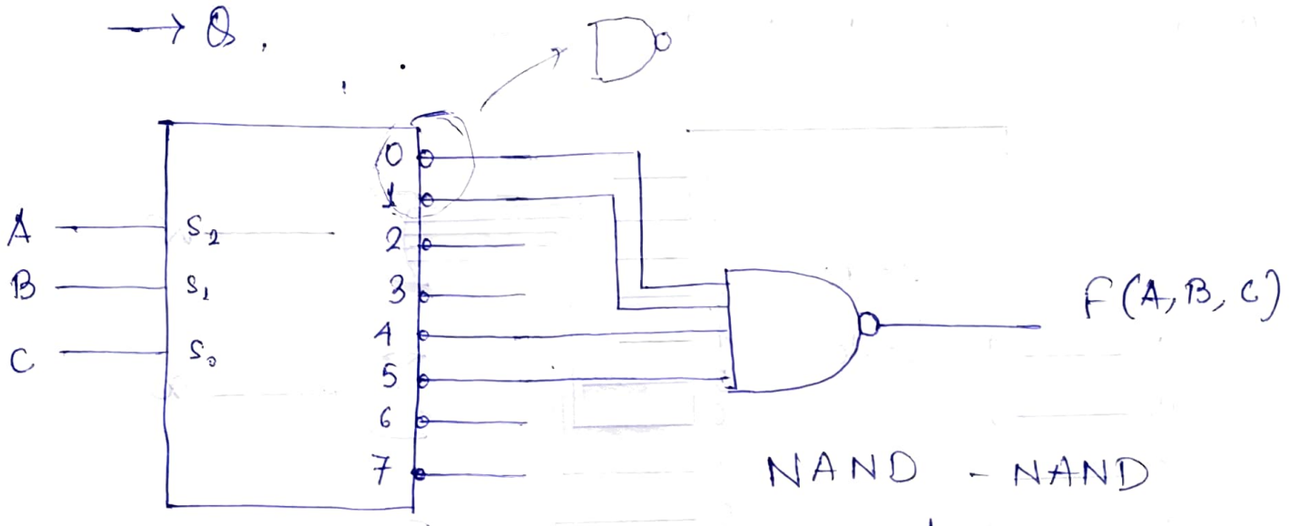


$F = \Sigma(0, 2, 4, 6)$

$F = C'$

C \ AB	00	01	11	10
0	1	1	1	1
1				

→ Q.



NAND - NAND

↓

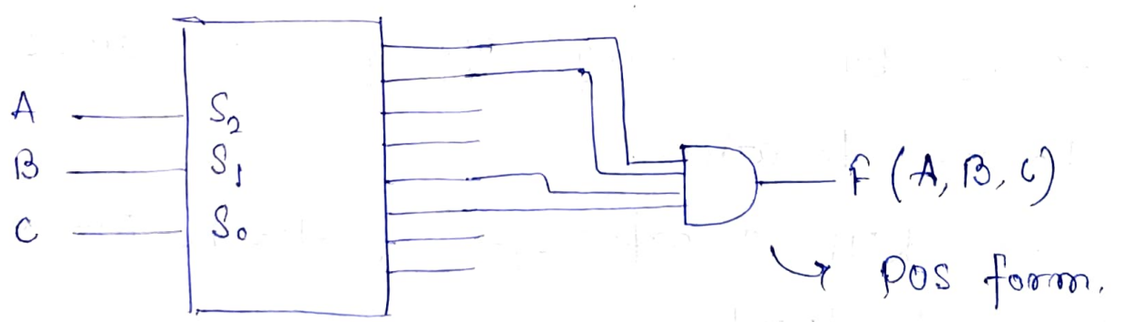
AND - OR.

$F = \Sigma(0, 1, 4, 5)$.

$F = B'$

C \ AB	00	01	11	10
0	1			1
1	1			1

→ Q.



↳ POS form.

$F = \Pi(0, 1, 4, 5) = \Sigma(2, 3, 6, 7)$

$F = B$.

→ 8-4-2-1 to 2-4-2-1 code using decoder.

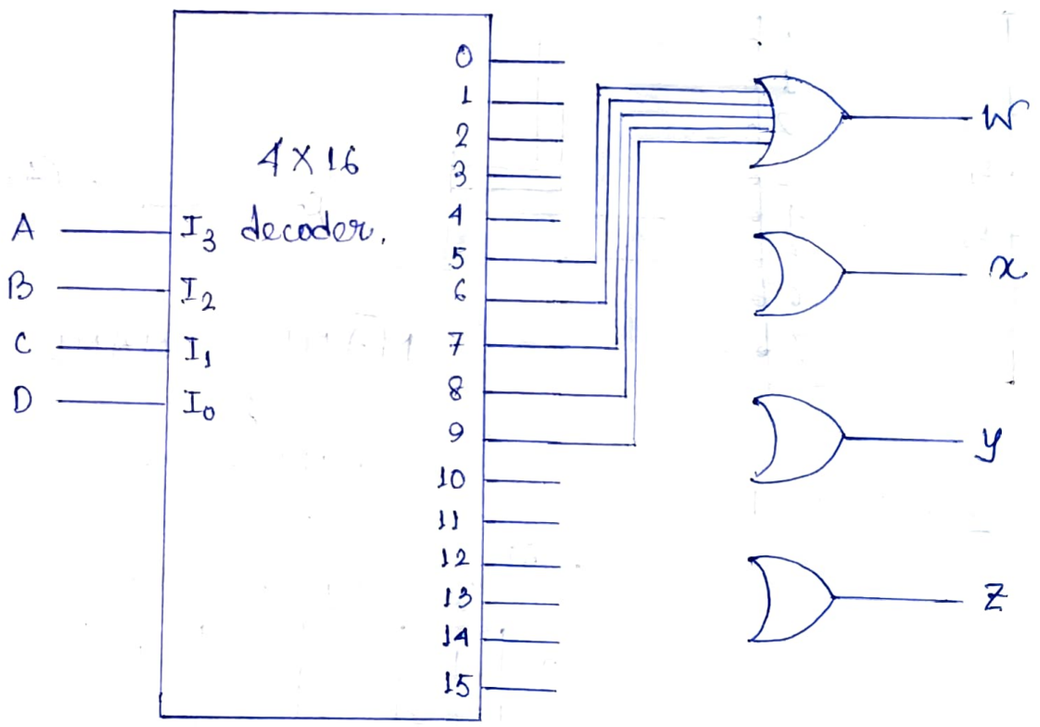
	A	B	C	D	W	x	y	z
	8	4	2	1	2	4	2	1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	1	0	0
7	0	1	1	1	1	1	0	1
8	1	0	0	0	1	1	1	0
9	1	0	0	1	1	1	1	1

$W = \Sigma(5, 6, 7, 8, 9) + \Sigma_\phi(10, 11, 12, 13, 14, 15)$

$x = \Sigma(4, 6, 7, 8, 9) + \Sigma_\phi(10, 11, 12, 13, 14, 15)$

$y = \Sigma(2, 3, 5, 8, 9) + \Sigma_\phi(10, 11, 12, 13, 14, 15)$

$z = \Sigma(1, 3, 5, 7, 9) + \Sigma_\phi(10, 11, 12, 13, 14, 15)$



→ ROM Implementation using decoder.

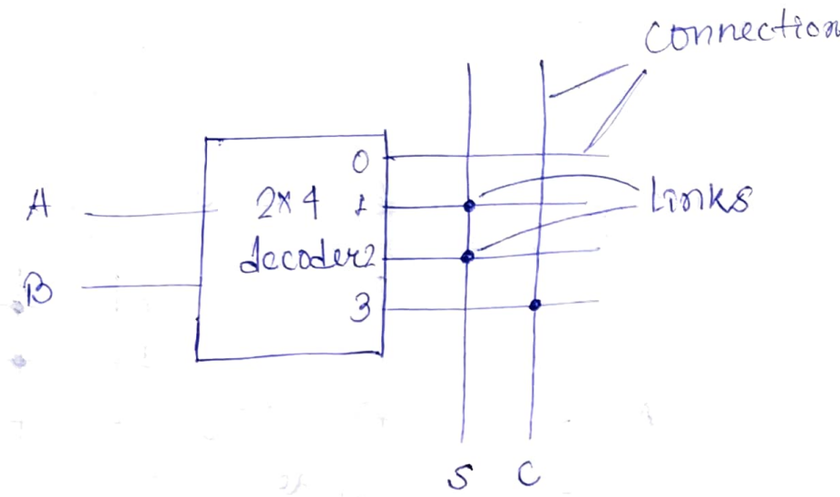
ROM can be used to realise combinational functions by storing appropriate values at appropriate locations.

Every ROM is expressed in terms of ROM matrix & decoder.

ROM matrix contains set of links & connections.

Half adder –

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$S = \Sigma(1,2)$

$C = \Sigma(3)$.

- Burning
- Fusing
- Memory (Not reprogrammable once burned).

→ Implementing functions using decoder.

To implement n variable function we need $n \times 2^n$ decoder.

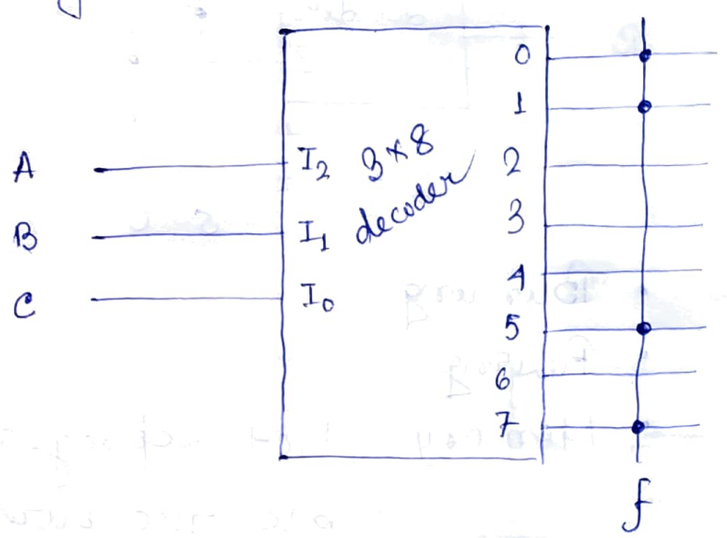
To implement m number of n var. functions there will be $(2^n + m)$ connections & $(2^n \times m)$ links.

→ Implementing functions using decoder & MUX.

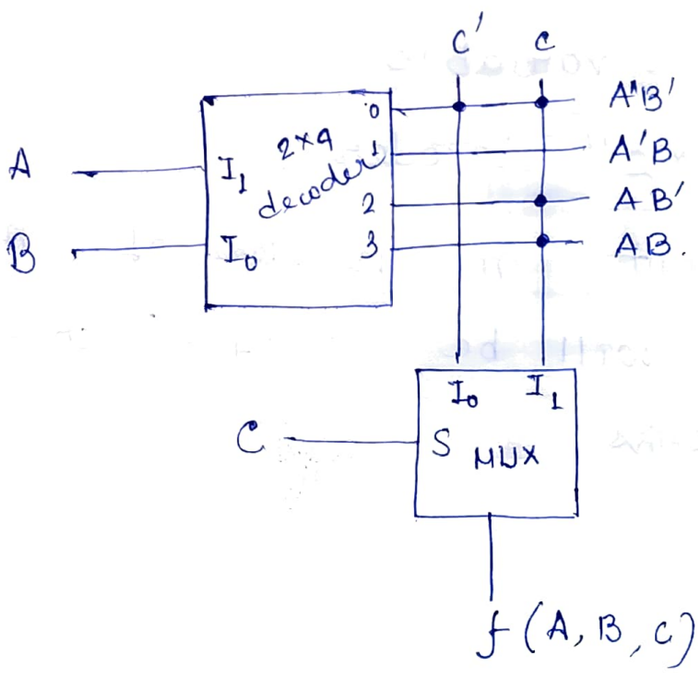
Taking example,

$f(A, B, C) = \Sigma(0, 1, 5, 7)$.

using 3×8 decoder,



To decrease number of connections, we can use MUX & a 2×4 decoder



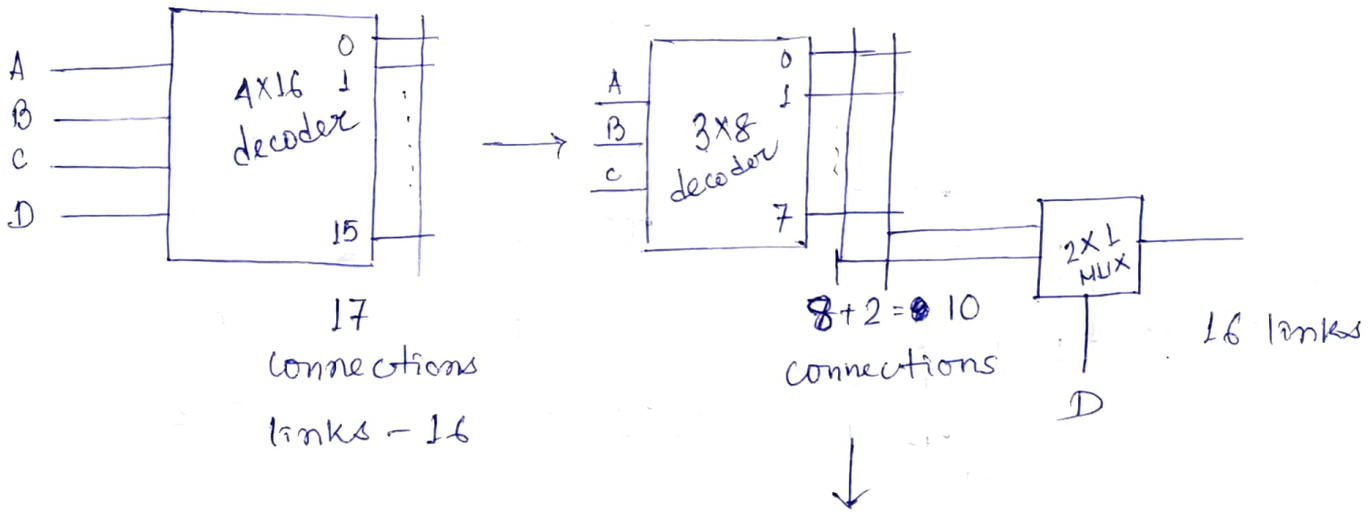
$f = A'B'C' + A'B'C + AB'C + ABC.$

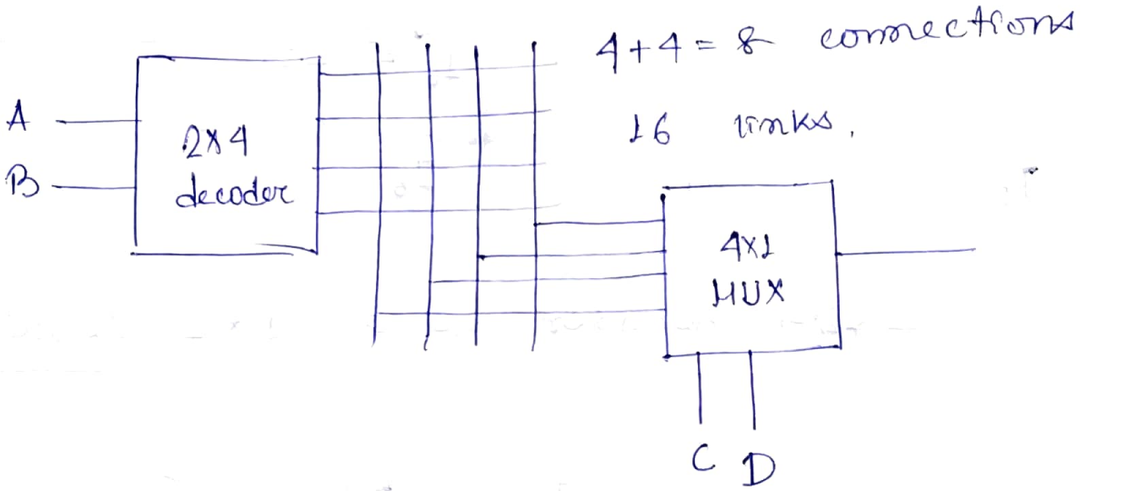
→ If there are n number of select lines to decoders & m number of MUXs, input lines to

no. of connections $= (2^m + 2^n)$.

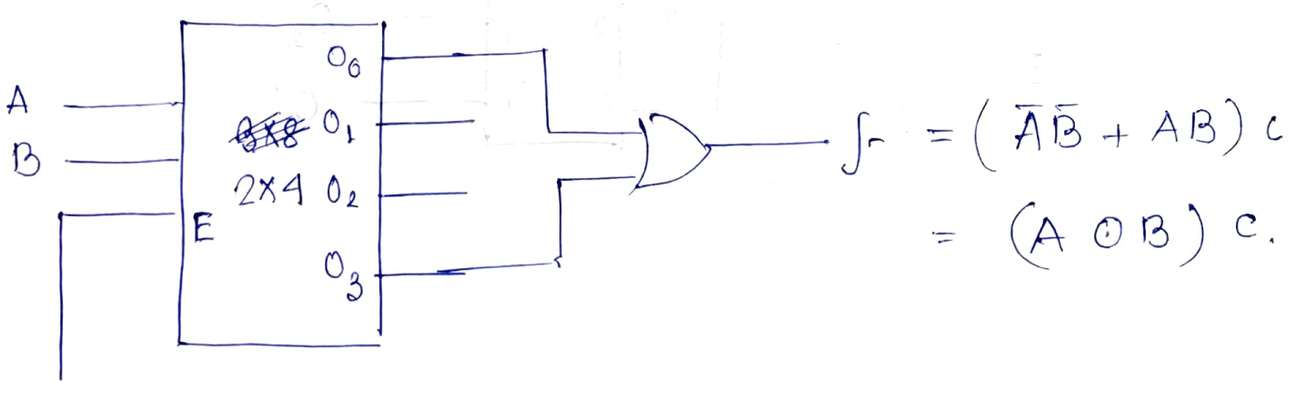
no. of links $= (2^m \times 2^n)$.

→ Using lesser inputs to decoders may decrease number of connections (not always).



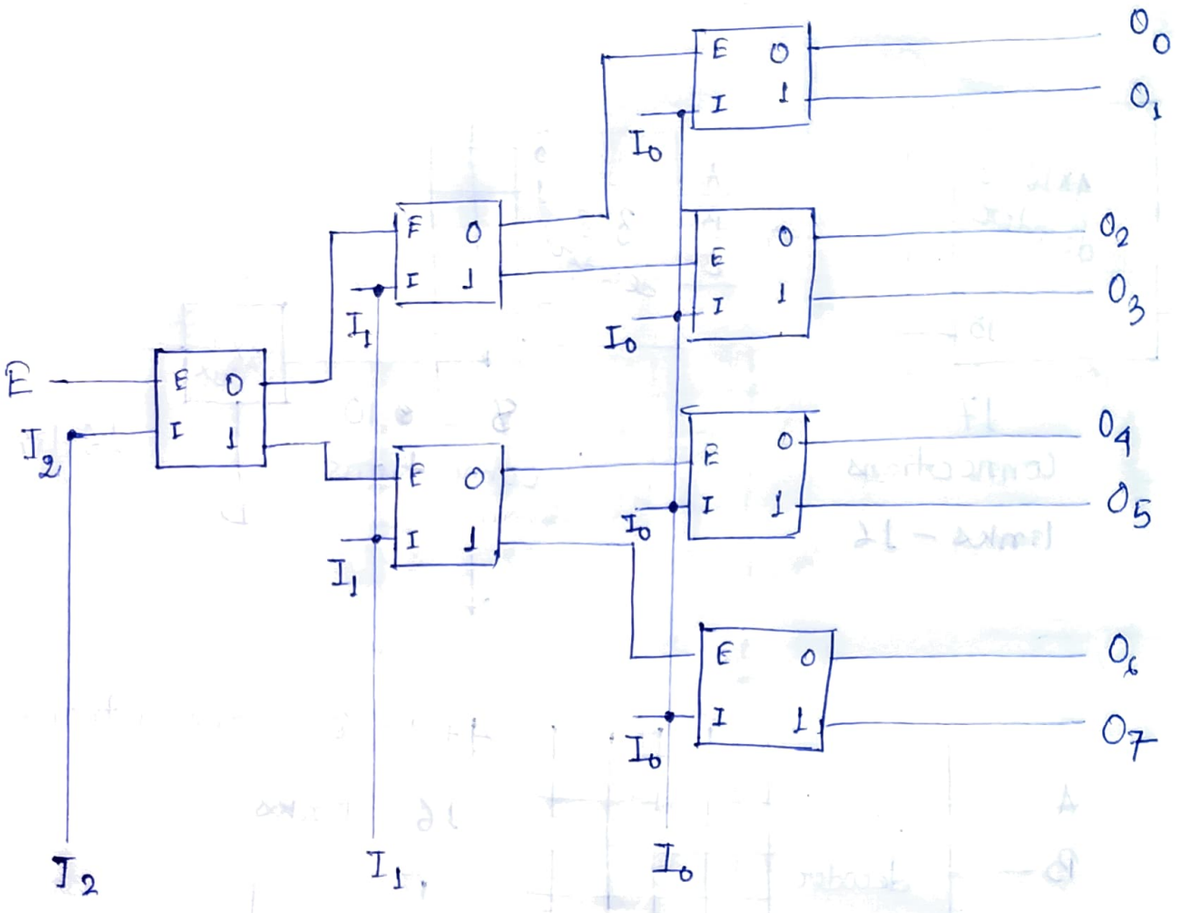


→ Implementing functions with decoder with enable input :

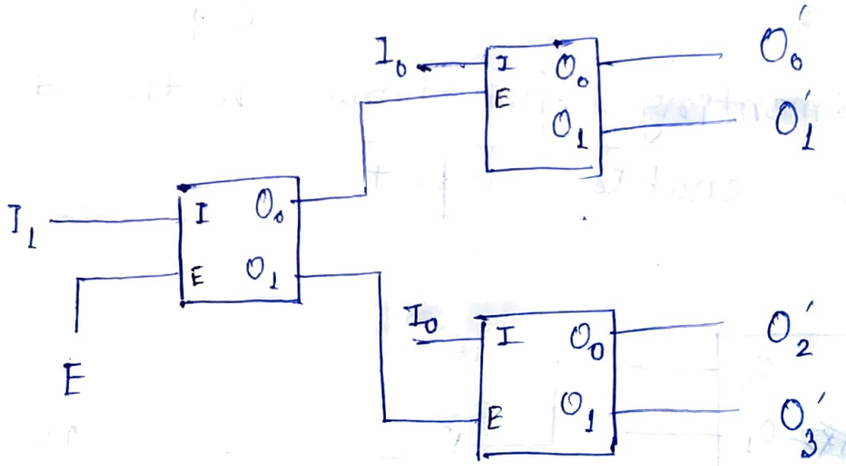


Enable input just gets multiplied with the function.

→ 3X8 decoder using 1 x 2 decoder.



→ 2x4 decoder using 1 x 2 decoder.



→ 6 X 64 decoder using 3 × 8 decoder.

1 3X8 decoder covers 8 o/p lines

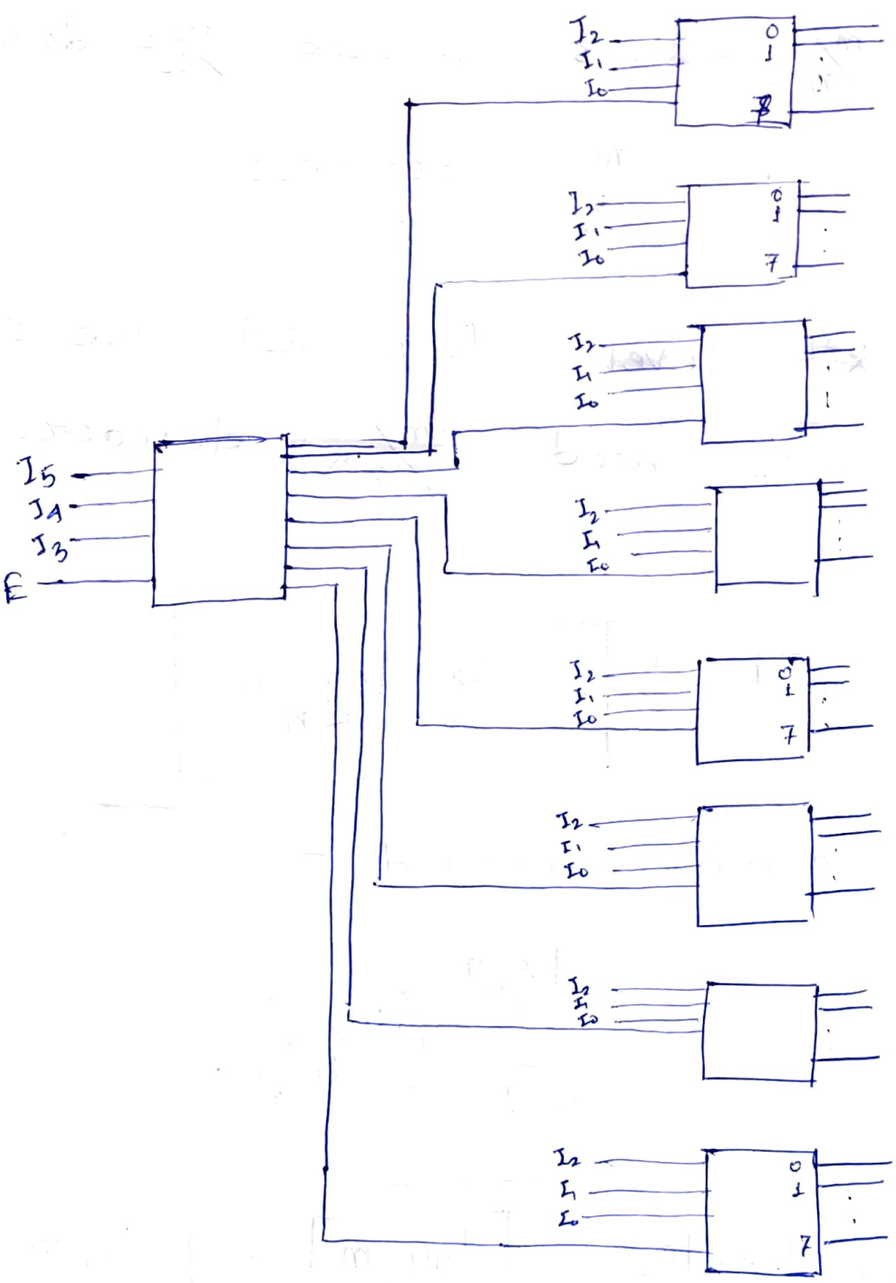
1 o/p line is covered by 1/8 of decoder.

At the last level we need $64 \times \frac{1}{8} = 8$ decoder.

These 8 decoders have 8 input enables.

To cover them we need $8 \times \frac{1}{8} = 1$ decoder

So, we need $(8+1) = 9$ 3×8 decoders.



→ Expansion of Decoder on general.

Constructing $(\log_2 m) \times m$ decoder using $(\log_2 n) \times n$ decoder.

1 $(\log_2 n) \times n$ decoder covers n lines (o/p)

1 o/p line can be covered by $\frac{1}{n}$ decoder.

To cover m o/p lines we need m/n decoders (last level).

These m/n decoders have m/n enable i/p s.

To cover m/n enables, we need m/n^2 decoders.

For next level, m/n^3 decoders.

Say, at k^{th} level from last we stop, where we need m/n^k decoders.

Now,

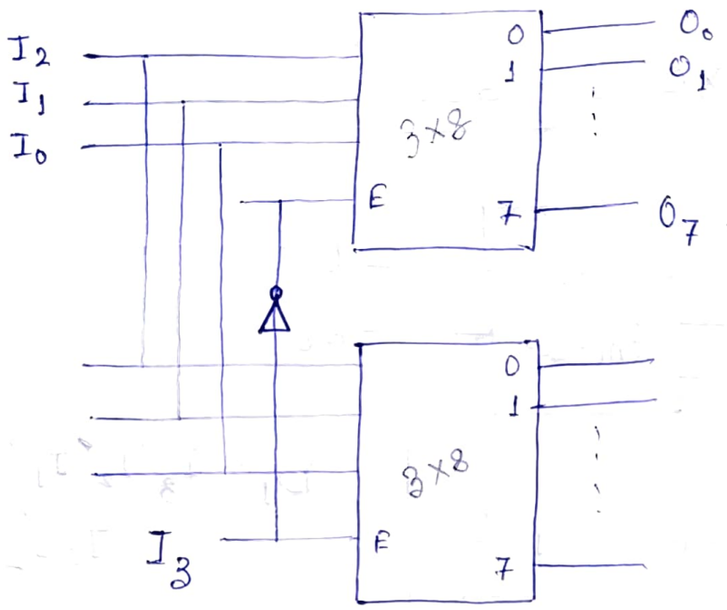
$$\frac{m}{n^k} \leq 1 \Rightarrow \boxed{k \geq \log_n m}$$

∴ No of decoders needed $= \sum_{k=1}^{\log_n m} \left(m/n^k\right)$.

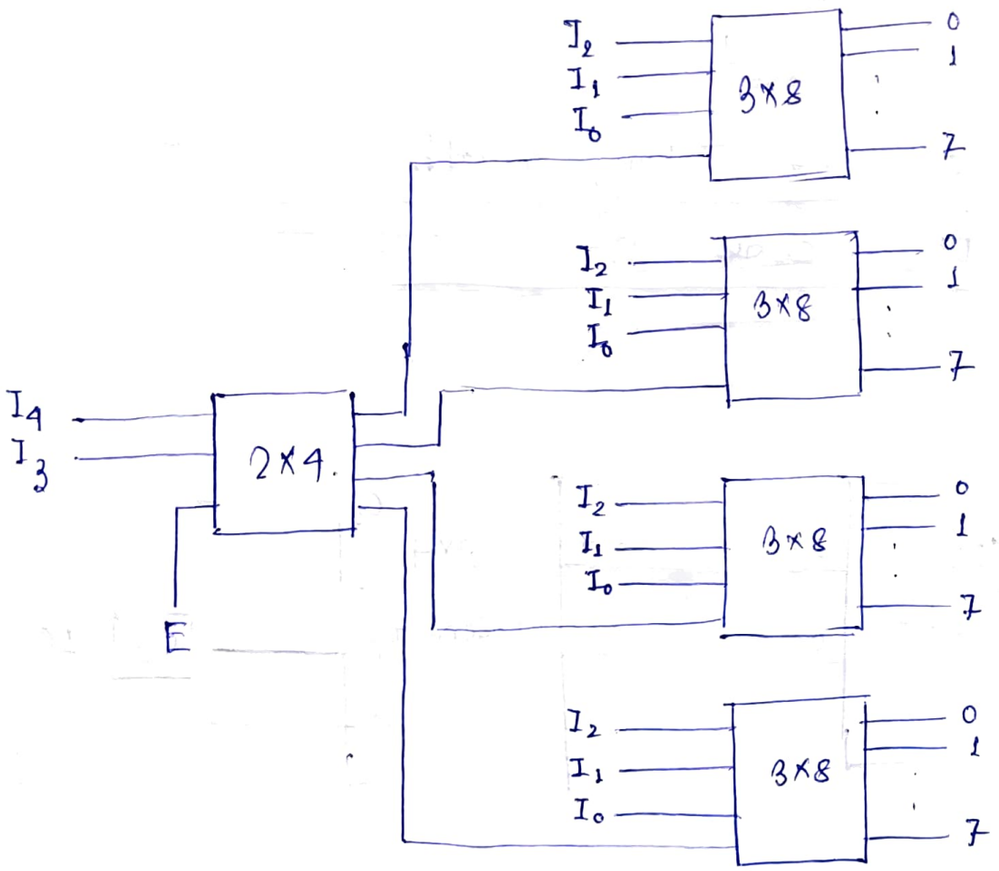
No. of levels $= \lceil \log_n m \rceil = \left\lceil \frac{\log_2 m}{\log_2 n} \right\rceil$

→ Expansion of Decoder by direct connection

4X16 using 3 x 8



5 x 32 using 3x8 & 2x4.



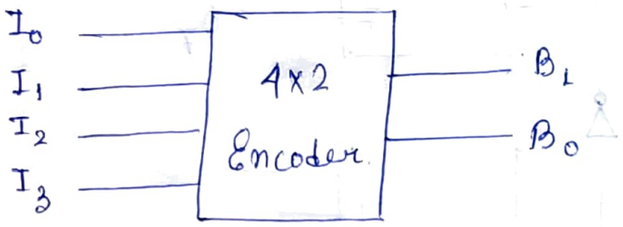
→ $m = 2^x$, $n = 2^y$ | $x \times 2^x$ decoder using $y \times 2^y$ decoder

$\lceil \frac{x}{y} \rceil$ levels

* Encoders.

→ Combinational circuit with 2^n input lines & n output lines.

→ 4X2 Encoder.

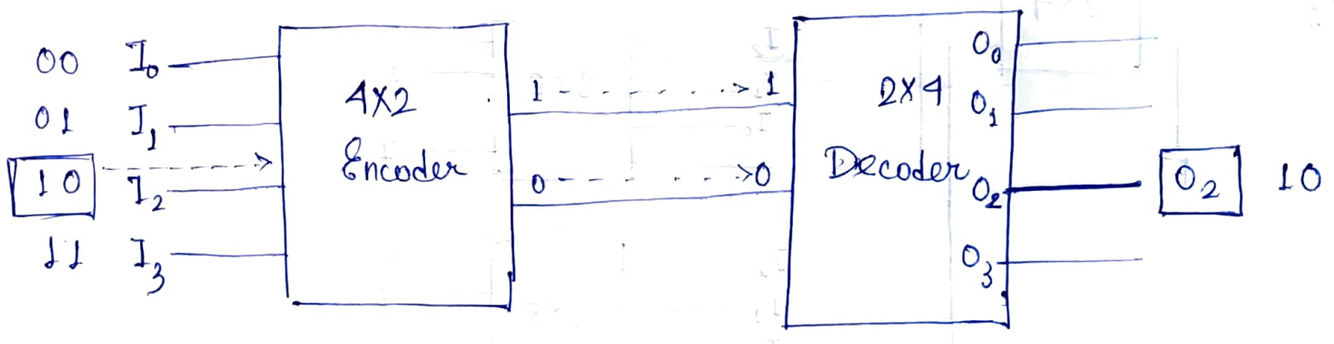


I_3	I_2	I_1	I_0	B_1	B_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$B_1 = I_3' I_2 I_1' I_0' + I_3 I_2' I_1' I_0'$$

$$B_0 = I_3' I_2' I_1 I_0' + I_3 I_2' I_1' I_0'$$

→ Encoder along with decoder to encode & decode -



→ Encoders perform lossless compression.

→ Types of Encoders –

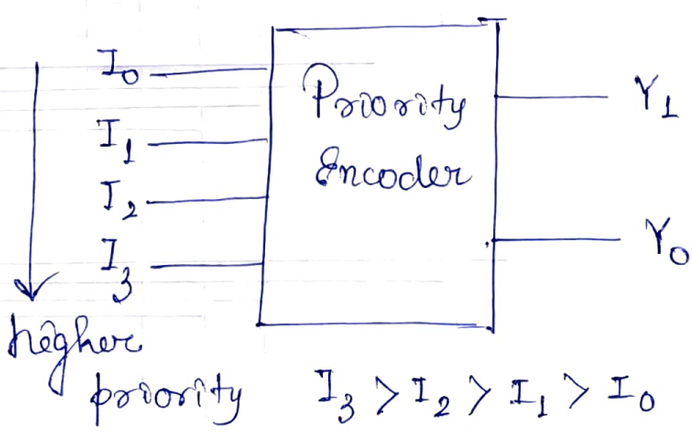
i) Non-priority (Doesn't support simultaneous i/p activation)

ii) Priority (Supports simultaneous i/p activation & used for interrupt servicing)

→ Due to static priorities, the lower priority i/p is exposed to starvation.

→ Priority Encoder. :

I_3	I_2	I_1	I_0	Y_1	Y_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1



$I_3 > I_2 > I_1 > I_0$

I_3I_2 \ I_1I_0	00	01	11	10
00	X	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$Y_1 = I_2 + I_3$

Or by inspecting TT,

$Y_1 = I_2 I_3' + I_3 = I_3 + I_2$

Similarly,

$Y_0 = I_3' I_2' I_1 + I_3$

$Y_0 = I_3 + I_2' I_1$

→ Decimal to BCD Encoder.

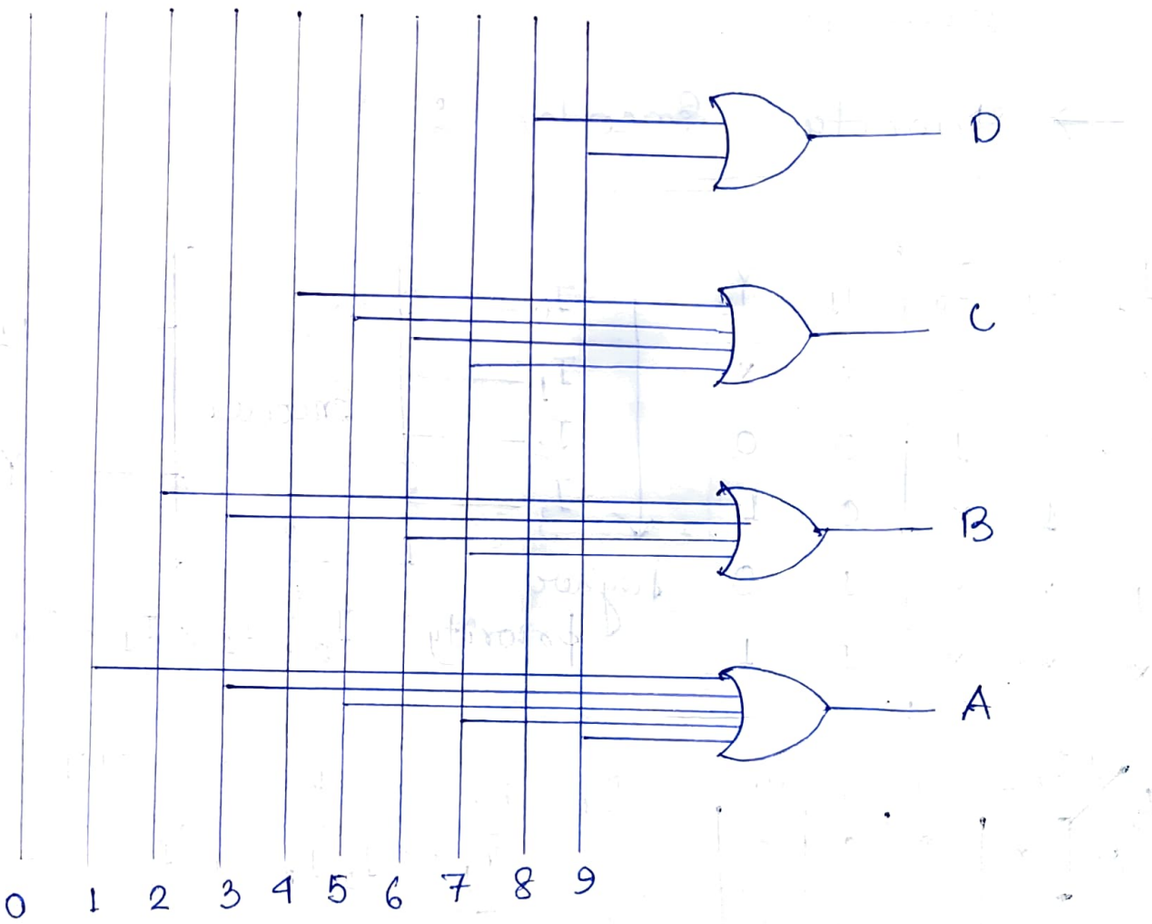
I/P	O/P			
	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

$D = 8 + 9$

$C = 4 + 5 + 6 + 7$

$B = 2 + 3 + 6 + 7$

$A = 1 + 3 + 5 + 7 + 9$



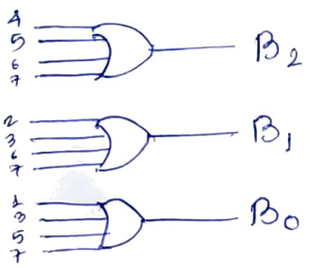
→ Octal to Binary Encoder.

I/P	B_2	B_1	B_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

$B_2 = 4 + 5 + 6 + 7$

$B_1 = 2 + 3 + 6 + 7$

$B_0 = 1 + 3 + 5 + 7.$



→ Hexadecimal to Binary Encoder.

I/P	B_3	B_2	B_1	B_0
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
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

$B_0 = 1 + 3 + 5 + 7 + 9 + B + D + F$

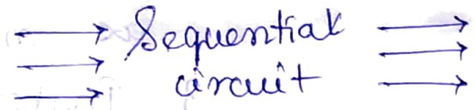
$B_1 = 2 + 3 + 6 + 7 + A + B + E + F$

$B_2 = 4 + 5 + 6 + 7 + C + D + E + F$

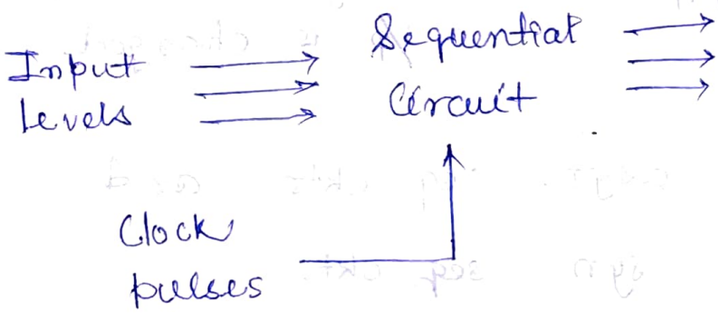
$B_3 = 8 + 9 + A + B + C + D + E + F.$

Sequential Circuits.

Asynchronous sequential circuits: Does not use clock signal but uses the pulses of the inputs. These are faster than synchronous sequential circuits. Difficult to design & o/p is uncertain.



Synchronous sequential circuits: Uses clock signal & level inputs. Since, they wait for the next clock pulse to arrive to perform the next operation, these circuits are bit slower.



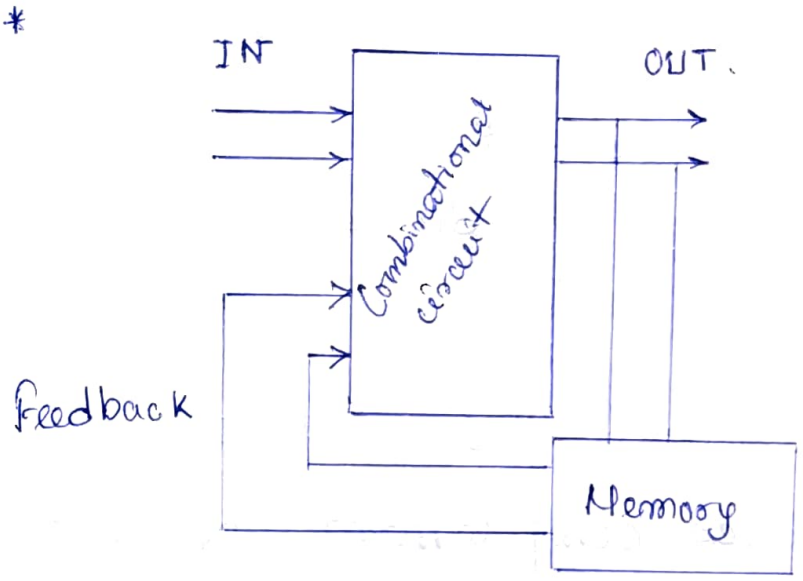
Syn. Seq. CKt.	Asyn. Seq. CKt.
1. Easy to design.	1. Difficult to design
2. Clocked FF acts as memory element.	2. Time delay elements or unclocked FF used as memory element.
3. Slower.	3. Faster as clock not present.
4. Status of memory element is affected only at the active edge of clock, if I/p is changed.	4. Status of memory element will change any time as soon as I/p is changed.

Latches are used in asyn. seq. ckts. and FFs are used in syn. seq. ckt.

Sequential Circuits.

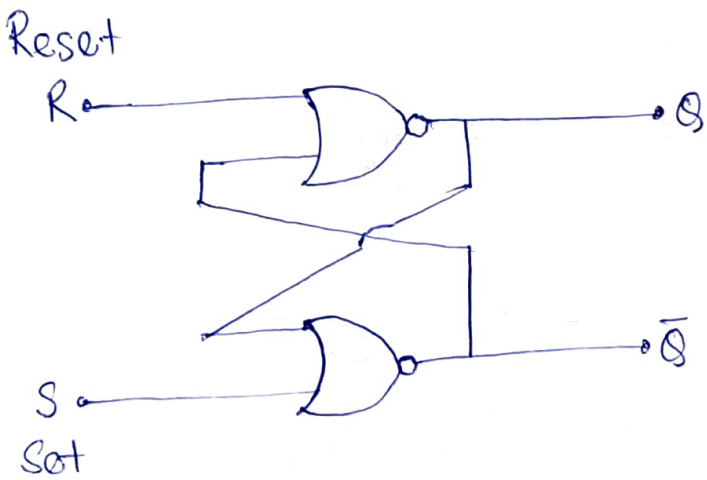
* In sequential circuits, the present output depends on the present input as well as past output(s)

*



* SR Latch : The basic storage element is called the latch. It latches 0 or 1.

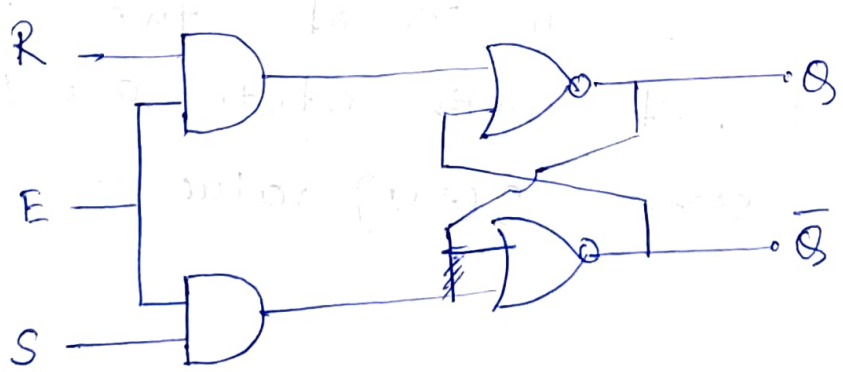
→ NOR SR latch and NAND SR latch are two types.

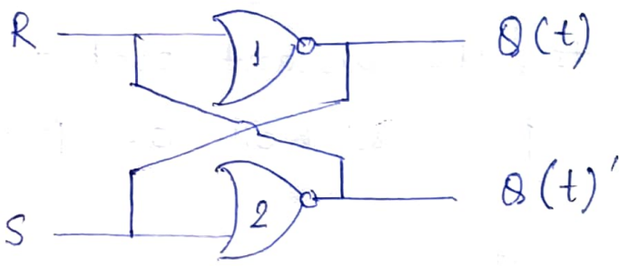


Reset ⇒ Q = 0
Set ⇒ Q = 1

NOR SR latch.

With enable,





1 NOR Gate has two inputs - R & complement of present state $Q(t)'$ & produces next state $Q(t+1)$.

2 NOR Gate has two inputs - S & present state, $Q(t)$ & produces complement of next state.

- If S = 1, next state $Q(t+1)$ will be equal to 1 irrespective of present state $Q(t)$.
- If R = 1, next state $Q(t+1)$ will be equal to 0 irrespective of present state $Q(t)$.

→ S = 0, R = 1, $Q = 0$, $\bar{Q} = 1$.

S = 0, R = 0, $Q = 0$, $\bar{Q} = 1$ - Memory.

→ S = 1, R = 0, $Q = 1$, $\bar{Q} = 0$.

S = 0, R = 0, $Q = 1$, $\bar{Q} = 0$ - Memory.

→ S = 1, R = 1, $Q = 0$, $\bar{Q} = 0$. (Not used)

S = 0, R = 0, $Q = 0$, $\bar{Q} = 1$ }
$Q = 1$, $\bar{Q} = 0$ }

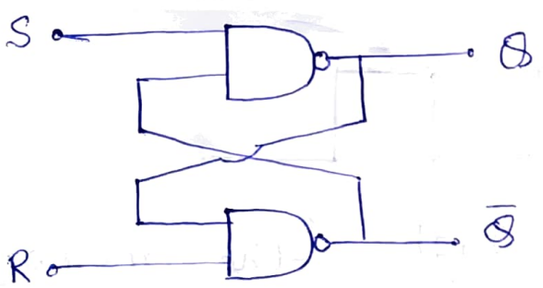
At any time, only one of two inputs should be 1. If both inputs are 1, then the next state $Q(t+1)$ value is undefined.

State Table | SR latch

S	R	Q(t+1)	Q(t+1)'
0	0	Q(t)	Q(t)'
0	1	0	1
1	0	1	0
1	1	– Not used.	

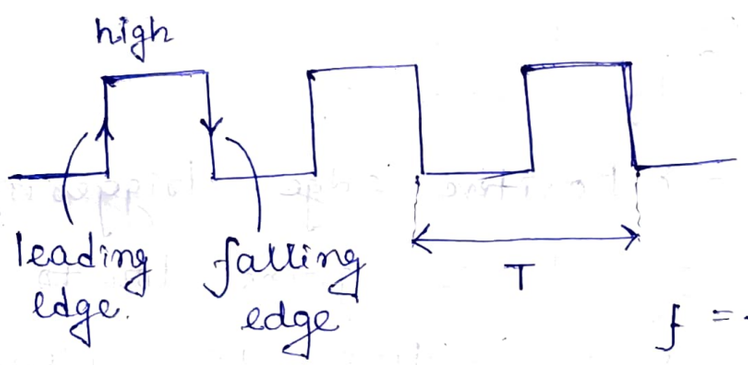
→ SR latch performs three types of functions such as hold (00), set (10), reset (01) based on input conditions.

→ NAND SR latch.



S	R	Q	Q'
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	as before memory.	

* Clock :



$f = \frac{1}{T}$

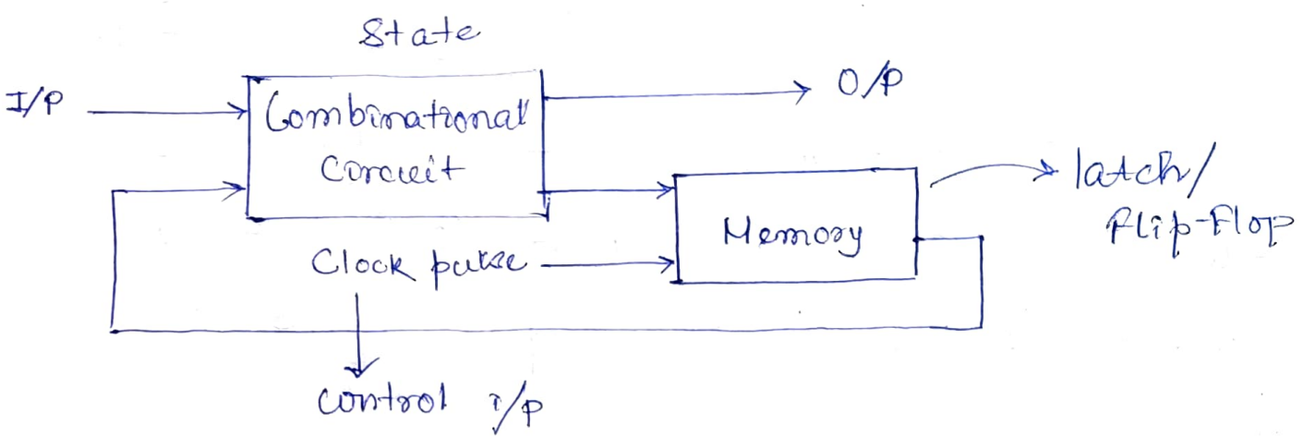
→ Duty cycle (DC) = ratio of ontime & total time period

$$= \frac{t_{ON}}{t_{ON} + t_{OFF}} \times 100\%.$$

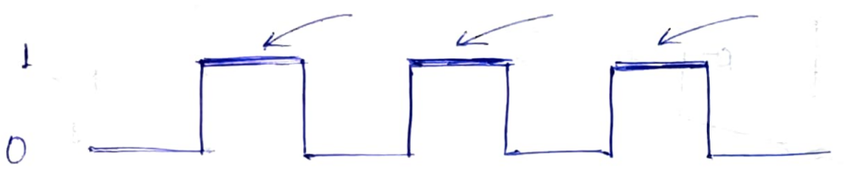
For $t_{ON} = T/2$, DC = 50 %.

Clock is nothing but a signal that controls when the inputs should be taken, when the outputs are needed to change.

* Triggering Methods.



i) Level triggering :

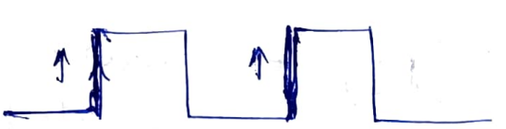


When the level is high, the memory is triggered (transition in memory)

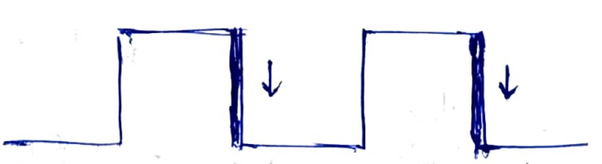
ii) Edge triggering :

→ Positive edge triggering.

When signal is going low to high, there is transition in memory.

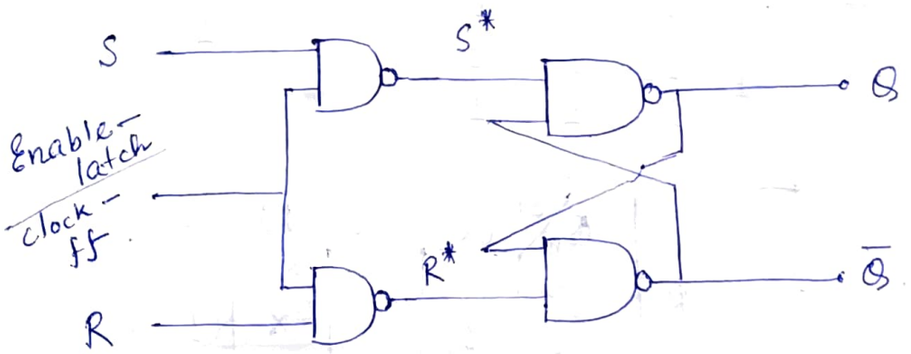


→ Negative edge triggering



* Flip-flop is a circuit that maintains a state until directed by input to change the state.

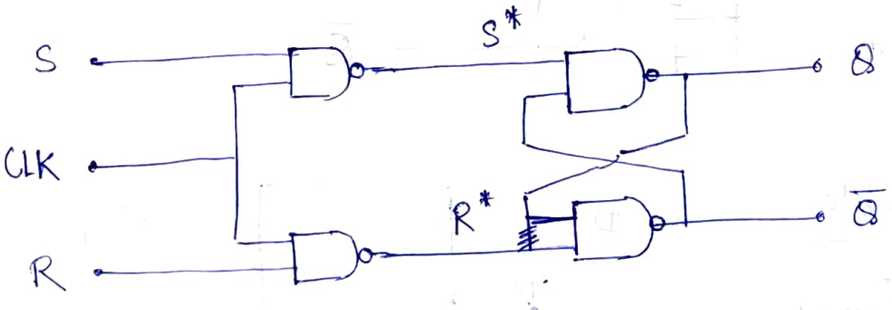
* Difference between latch & flip-flop:



Only difference lies on the control input. If it is only an enable input it acts as latch & if it is clock then it acts as flip-flop.

When there is enable, it is level triggered.

* S-R flip-flop.:



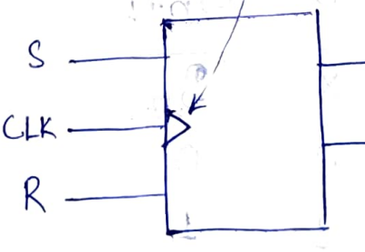
$S^* = \bar{S} + \overline{CLK}$

$R^* = \bar{R} + \overline{CLK}$

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

↑ Truth Table

represents edge-triggered.



S^*	R^*	Q	$\bar{Q}$
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	Memory.	

Truth Table for SR FF

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

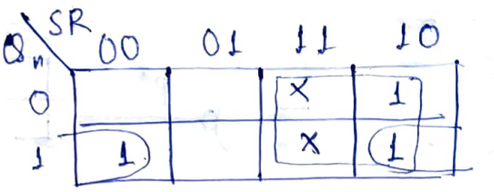
Characteristic Table –

CLK = 1

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

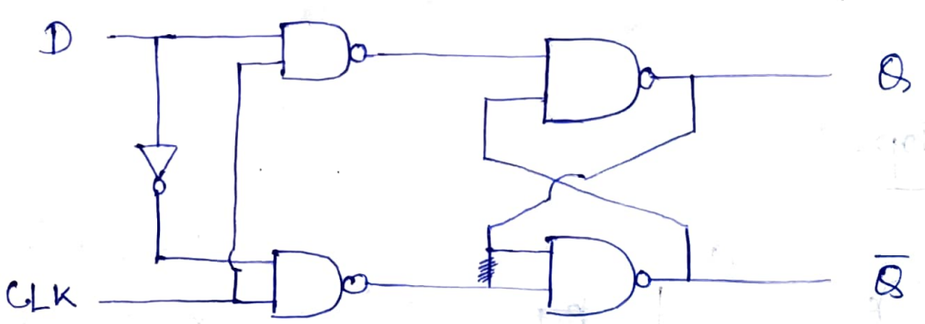
Excitation Table –

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



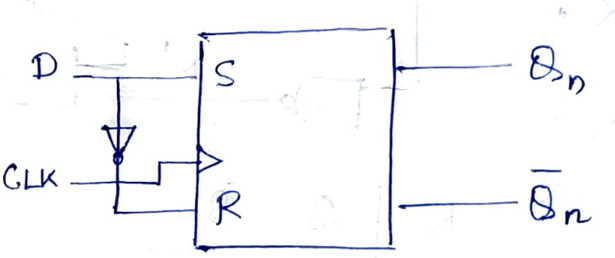
$Q_{n+1} = S + Q_n\overline{R}$

* D Flip-Flop. (D for data)



Truth Table

CLK	D	Q_{n+1}
0	X	Q_n
1	0	0
1	1	1



Characteristic Table :

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

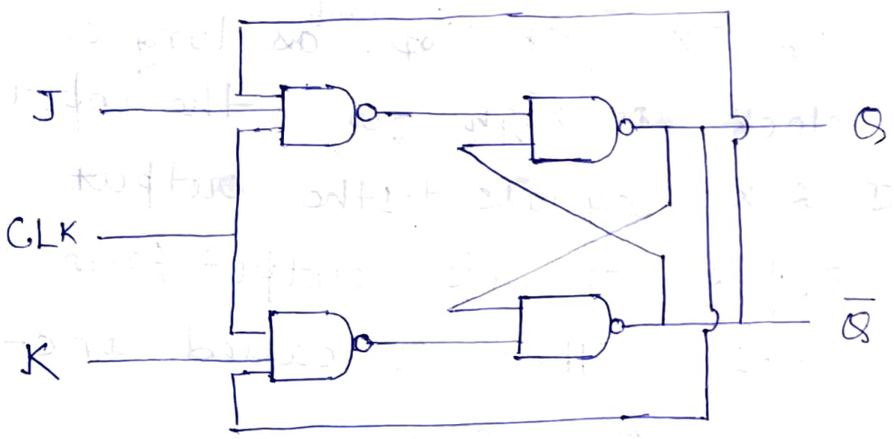
$$Q_{n+1} = D$$

Excitation Table

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

* JK Flip-flop (Jack Kilby)

→ In SR flip-flop we can't use $S=1, R=1$ that's why we need JK flip-flop.



Truth Table.

CLK	J	K	Q_{n+1}
0	x	x	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$\bar{Q}_n$ (toggle)

When $J=1, K=1, CK=1$,

$Q_{n+1} = 0 - 1 - 0 - \cdots$

Called racing.

Characteristic Table

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

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

Excitation Table

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

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

→ Race Around Condition.

Racing – In JK flip-flop as long as clock is high for the input conditions J & K equals to 1 the output changes or complements its output from $1 \rightarrow 0$ & $0 \rightarrow 1$. This is called uncontrolled changing or racing.

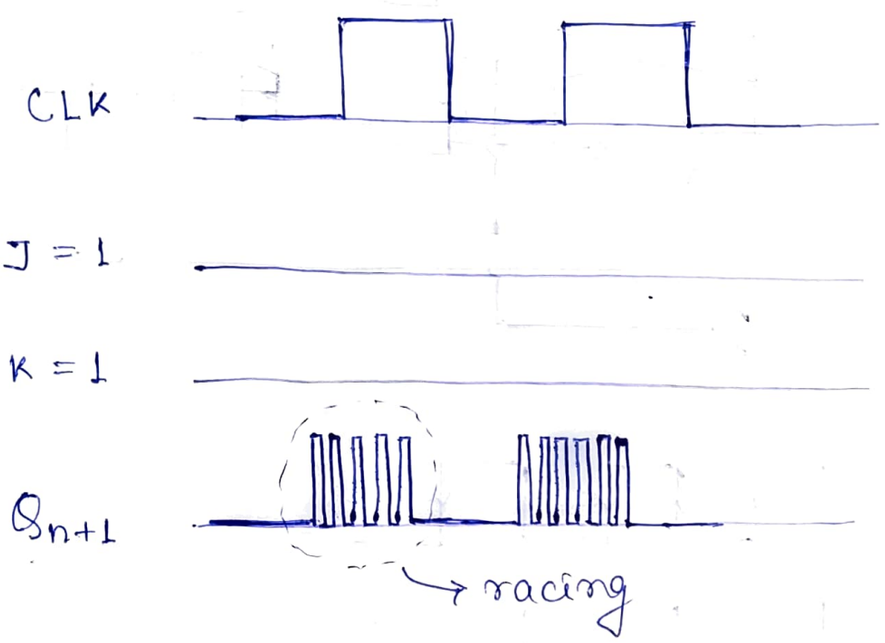
Steps to avoid racing condition on JK flip-flop –

i) If the clock ON or high time is less than the propagation delay of the flip-flop then racing can be avoided. This is done by using edge triggering rather than level triggering.

$T/2$ < propagation delay of FF

[Duty cycle 50%]

ii) If the flip-flop is made to toggle over one clock period then racing can be avoided. (This introduced the concept of Master Slave JK FF).



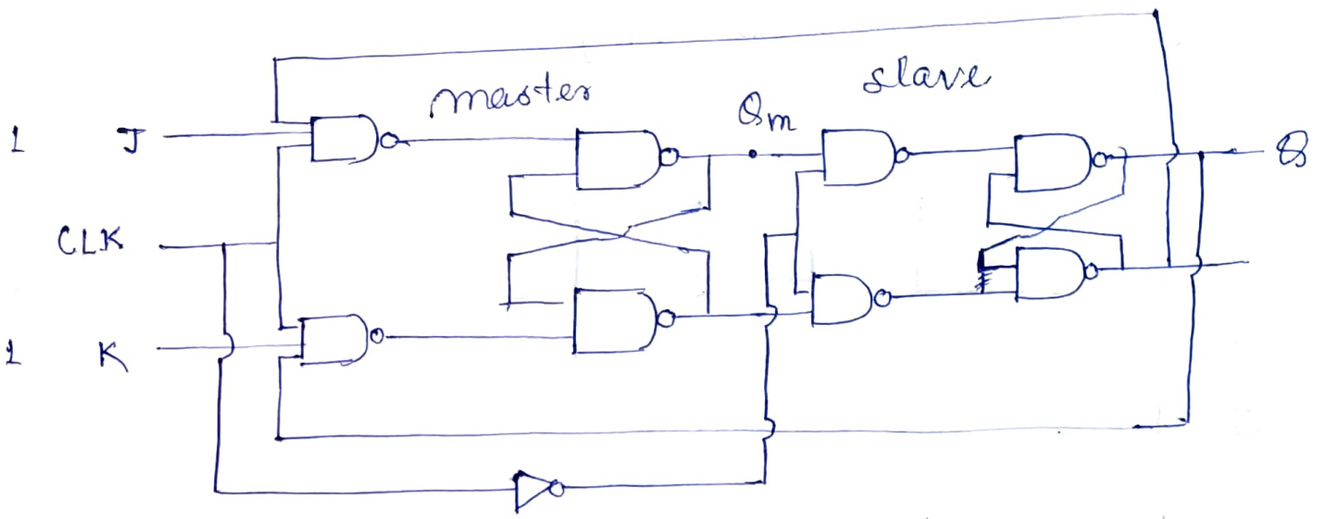
→ Toggling : Without the edge-triggering of the clock input, the circuit would continuously toggle between its two output states when both J & K were held high making of an astable device.

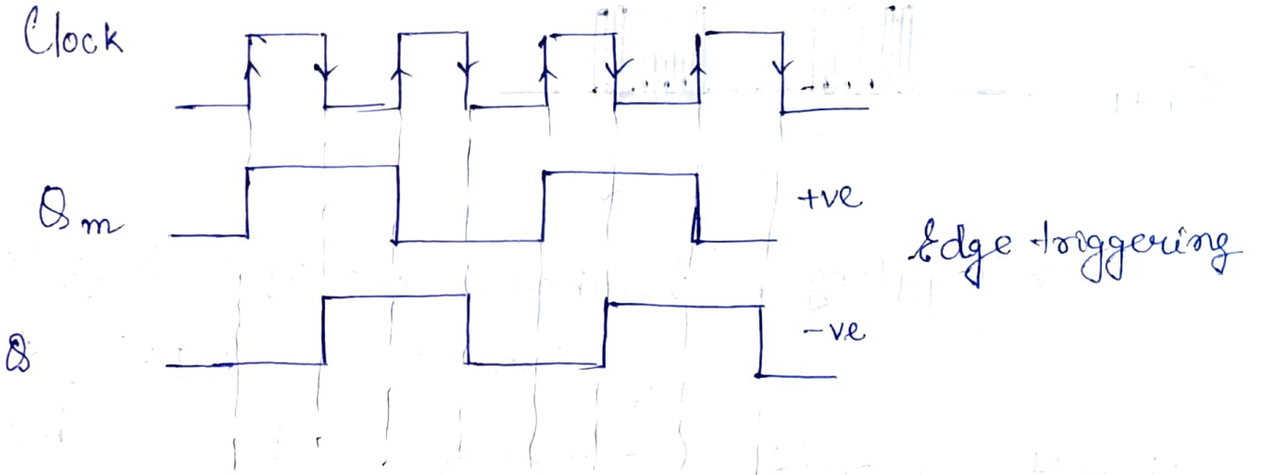
Toggling means the transition of output from 0 → 1 or 1 → 0.

Racing is the fact that in a clock high time how many times the toggling happens.

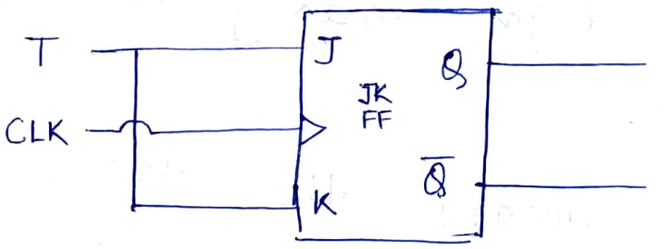
→ Master Slave JK FF.

Master Slave operation is same as negative edge triggered JK FF.





* T Flip-Flop. (T for toggling)



Truth Table

CLK	T	Q_{n+1}
0	X	Q_n
1	0	Q_n
1	1	$\overline{Q}_n$ (toggling)

Characteristic Table

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

Excitation Table

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

$Q_{n+1} = Q_n \oplus T$. (odd one's detector).

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

* Flip Flop Conversion.

→ Steps :

i) Identify available & required FF.

ii) Make characteristic table for required FF.

iii) Make excitation table for available FF.

iv) Write boolean expression for available FF.

v) Draw the circuit.

→ eg. JK to D FF.

available - JK FF | Required - D FF

characteristics table of D FF.

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

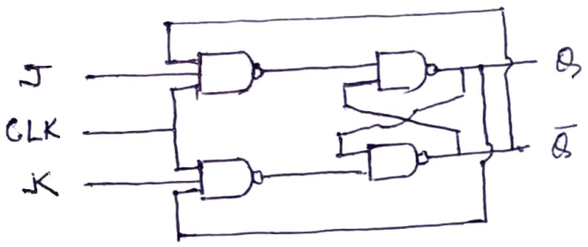
Excitation table of JK FF

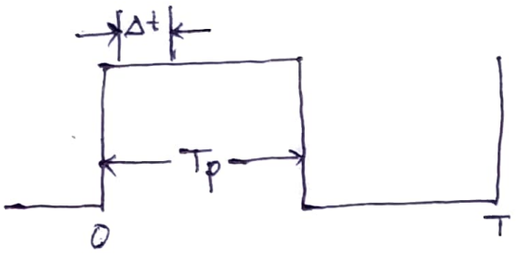
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

$J = D$

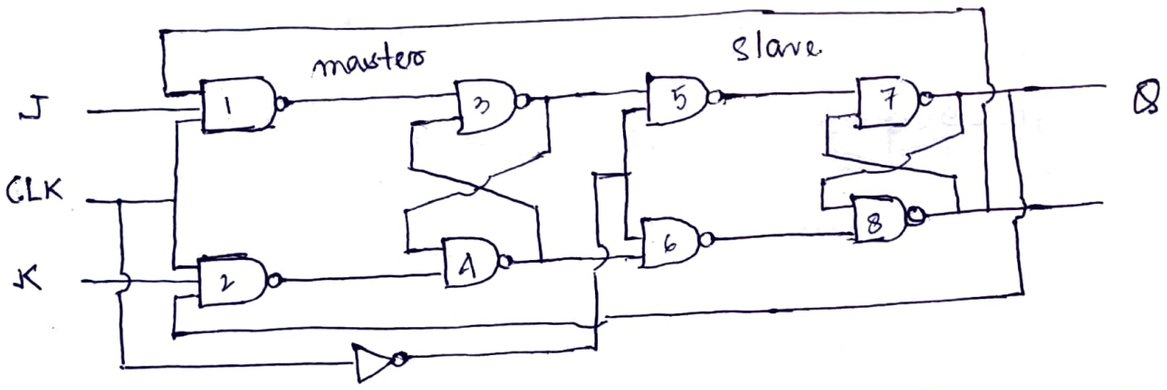
$K = D'$

Advantage of Master-Slave JK FF over JK FF:

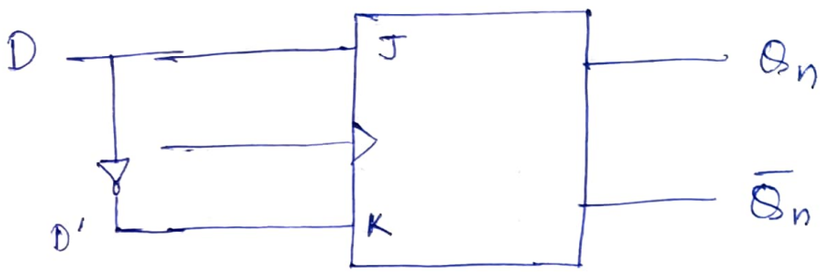




For the J-K flip-flop when CLK is 1 and J = K = 1 and Q = 1, after a time interval Δt (propagation delay of FF) the output will change to Q = 0. Now, we have J = 1 = K and Q = 0. So, if $\Delta t < T_p$ [T_p → time of clock pulse when high] the output will change again to Q = 1. Hence, for the time duration of T_p of the clock pulse, the output will oscillate between 0 and 1. This uncontrolled changing of output is called racing and this is not wanted as the o/p is uncertain. Generally, the propagation delay of TTL gates is in the order of nanoseconds. So, if the clock pulse is of the order microseconds, output will change thousands of times.



For master-slave JK FF, master FF responds to the data input when the clock pulse is high whereas the slave flip-flop responds to the output of the master FF when clock is low. At any time, only one of the master and slave FFs works and slave always follow master FF. Thus, the final output changes only when the clock is low, when the data inputs are not effective (inputs to 5 & 6 don't change at the time of application of clock pulse). This way race-around condition is avoided & this is the advantage of m-s JK FF over JK FF.



JK to D FF conversion.

→ T FF to D FF conversion.

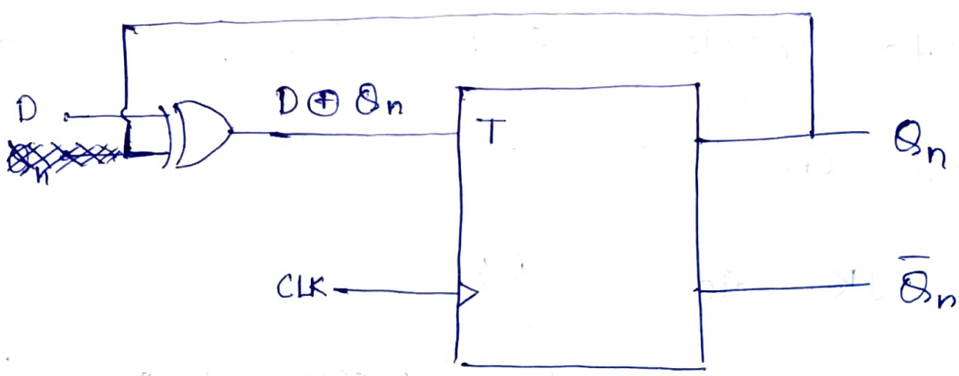
Ch. Table
D FF

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

Exci. Table
T FF

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

$T = Q_n \oplus D$



→ SR FF to JK FF conversion.

Ch. Table of JK FF.

Q_n	J	K	Q_{n+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

Exc. Table
SR FF

S	R
0	x
0	x
1	0
1	0
x	0
0	1
x	0
0	1

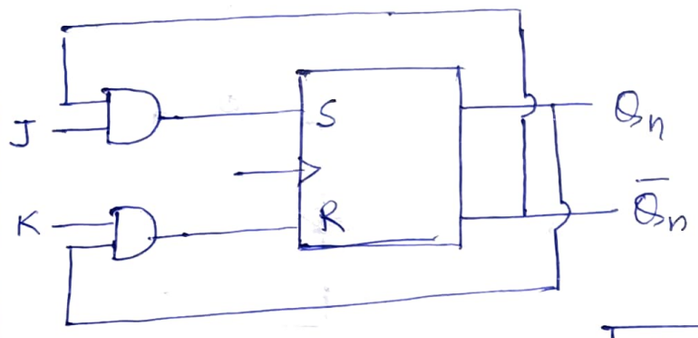
$S = \bar{Q}_n J$

$R = Q_n K$

[by K' Map]

Excl. Table SR FF

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



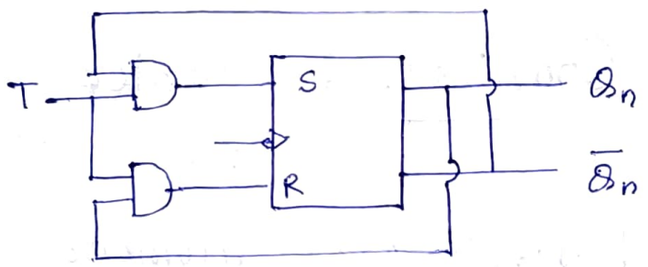
JK → SR
T → SR

→ SR FF to T FF conversion.

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

$S = \bar{Q}_n T$

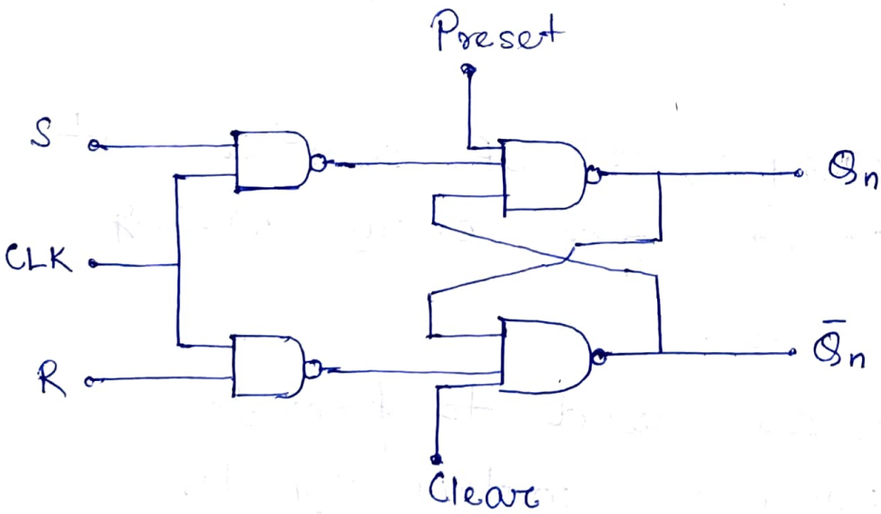
$R = Q_n T$



* Preset & Clear Inputs.

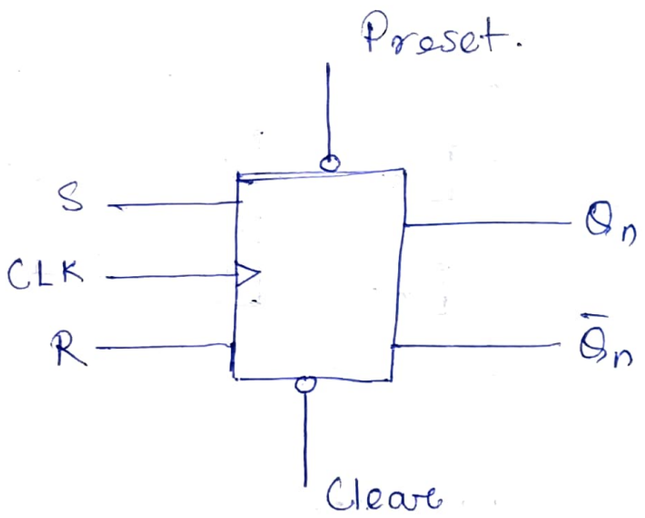
They are the direct inputs or overriding inputs or asynchronous inputs.

The synchronous inputs are S, R, J, K, D, T.



Preset-0, $Q_n = 1$
Clear-0, $\bar{Q}_n = 1$
$\Rightarrow Q_n = 0$

Whatever be the value of CLK & synchronous i/p s.



Pr.	Cle.	Q_n
0	0	Not used
0	1	1
1	0	0
1	1	FF performs normally.

* Counters. : Counter is a sequential circuit that stores (and often displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

Counters are used in digital electronics for counting purpose, they can count specific event processing in the circuit.

For example, in UP counter, it increases count for every rising edge of clock.

Counters are used to provide accurate timing & control signals.

→ Counters are of two types –

Synchronous – all the flipflops respond to the same clock intervals.

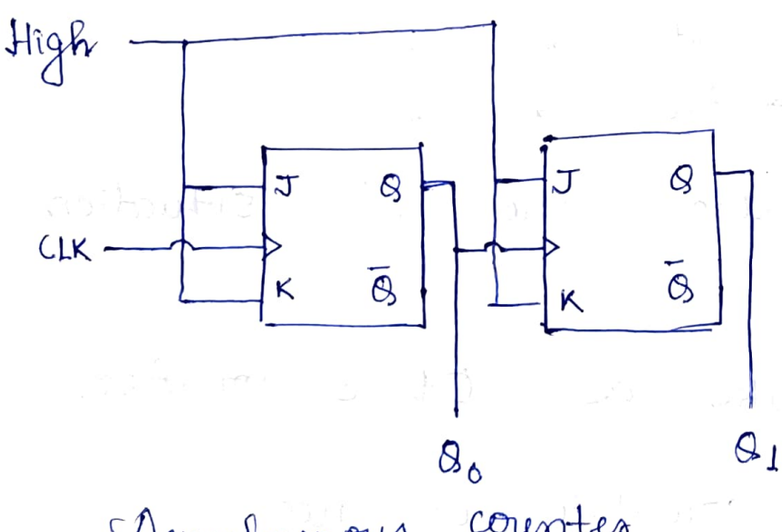
asynchronous (ripple) – the output of one FF drives the clock of another FF.

→ Synchronous counters are faster.

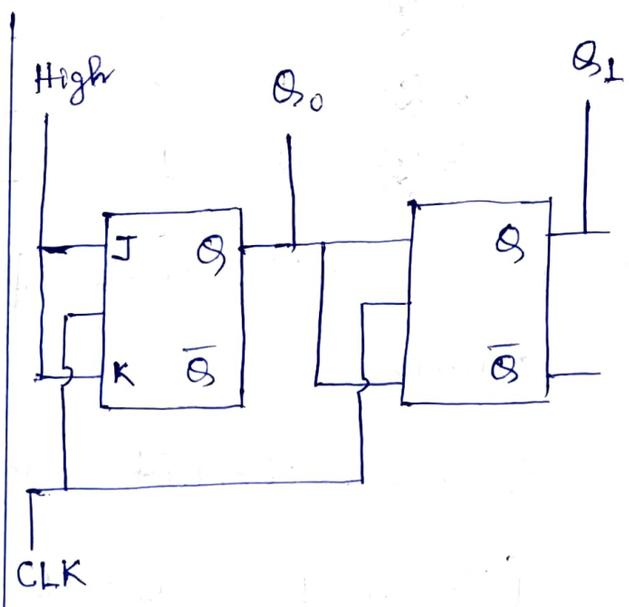
→ Due to simplicity of design, asynchronous counters are used in IC fabrication.

→ Simplified version of synchronous counter is called shift counters. Basic element in shift counter is D FF.

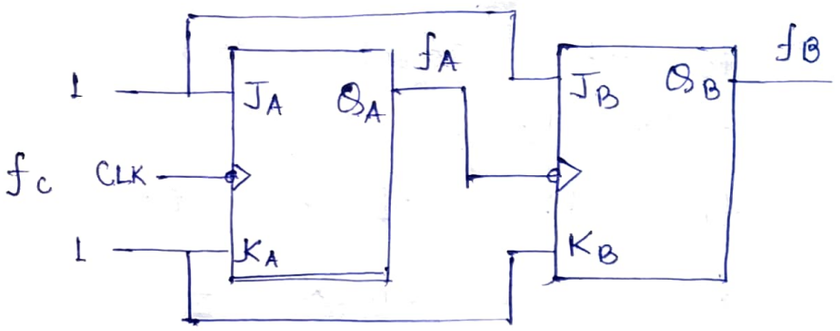
Ring counter & Johnson counter are further simplified version of shift counter.



Asynchronous counter



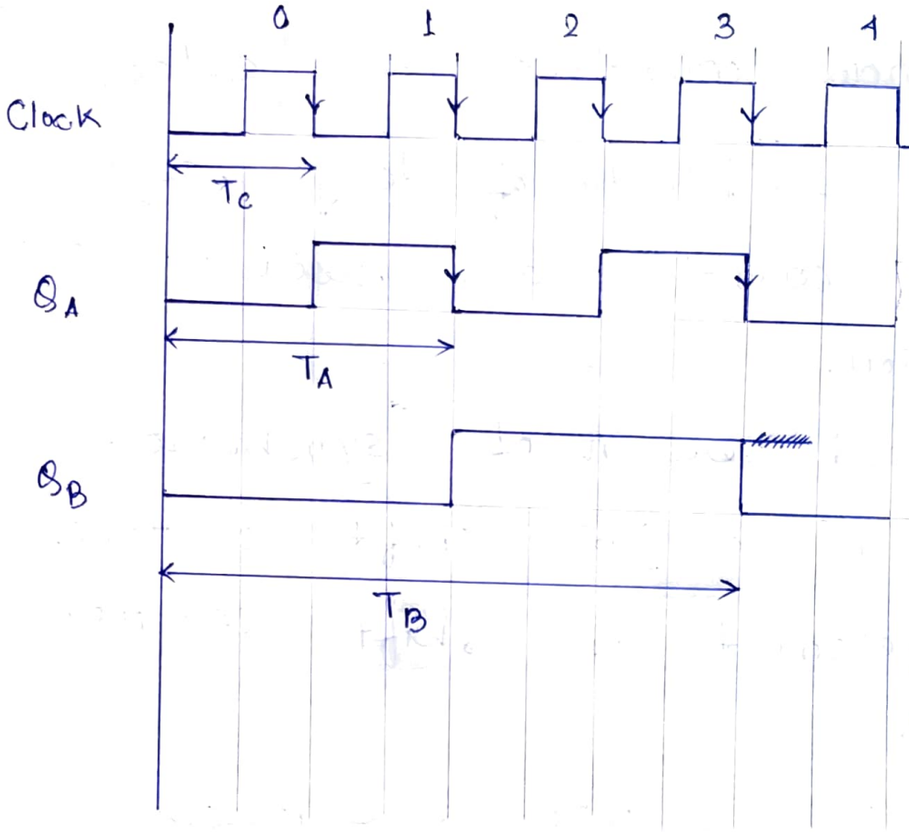
→ FF as divide-by-2 circuit.



f → frequency of signal.

(0 to 3 Counter)

asynchronous.



$T_A = 2T_C$

$T_B = 2T_A = 4T_C$.

$f_A = \frac{f_c}{2}$

$f_B = \frac{f_c}{4}$.

If there are p no. of flip-flops final frequency will be $f_c/2^p$

CLK	Q_B	Q_A	
0 (↓)	0	0	← 0th clock pulse
1 (↓)	0	1	← 1st " "
2 (↓)	1	0	← 2nd " "
3 (↓)	1	1	→ 3rd " "

For 4th clock pulse the 0th situation again.

So, the circuit works as 0 to 3 counter.

(To make 0-15 counter we need 4 flop-flops)

→ Difference between ripple & synchronous counters :

Ripple / asynchronous.	Synchronous.
1. FFs are connected in such a way that the o/p of first FF drives the clock of next FF.	1. There is no connection between o/p of first FF & clock of next FF.
2. FFs are not clocked simultaneously.	2. FFs are clocked simultaneously.
3. Circuit is simple for more number of states.	3. Circuit becomes complex as number of states increases.
4. Speed is slow as clock is propagated through number of stages.	4. Speed is high as clock is given at a same time.

→ UP counter — counts to up 0-1-2-

DOWN " — counts to down 3-2-1-

* Set, Reset, Preset, Clear.

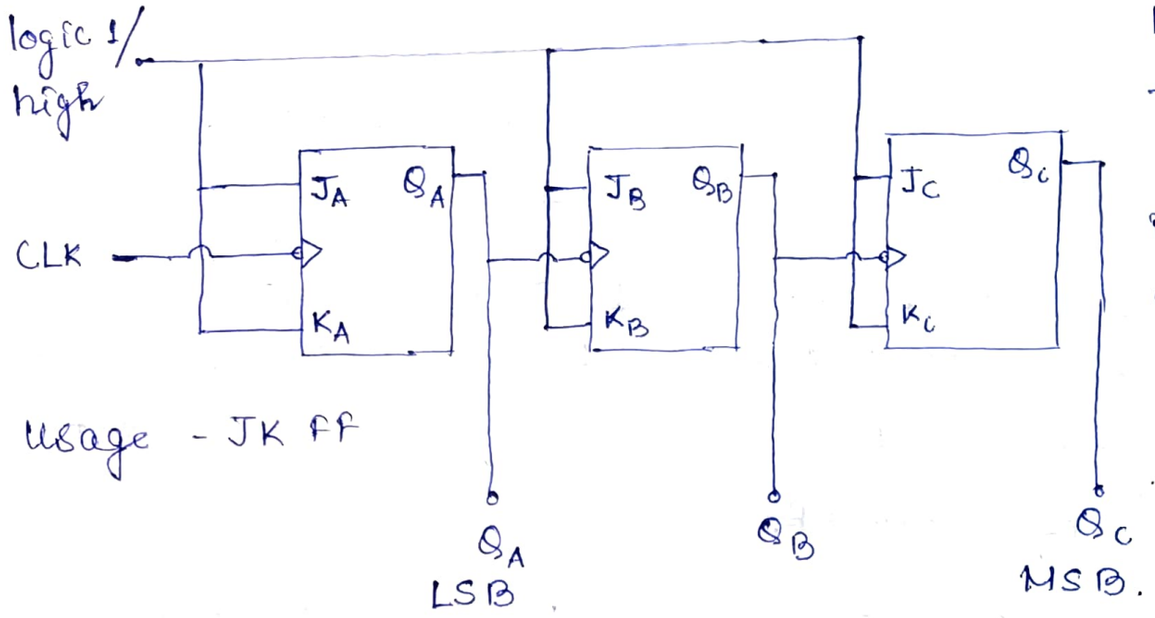
Set — $J=1, K=0 \Rightarrow Q=1, Q'=0$

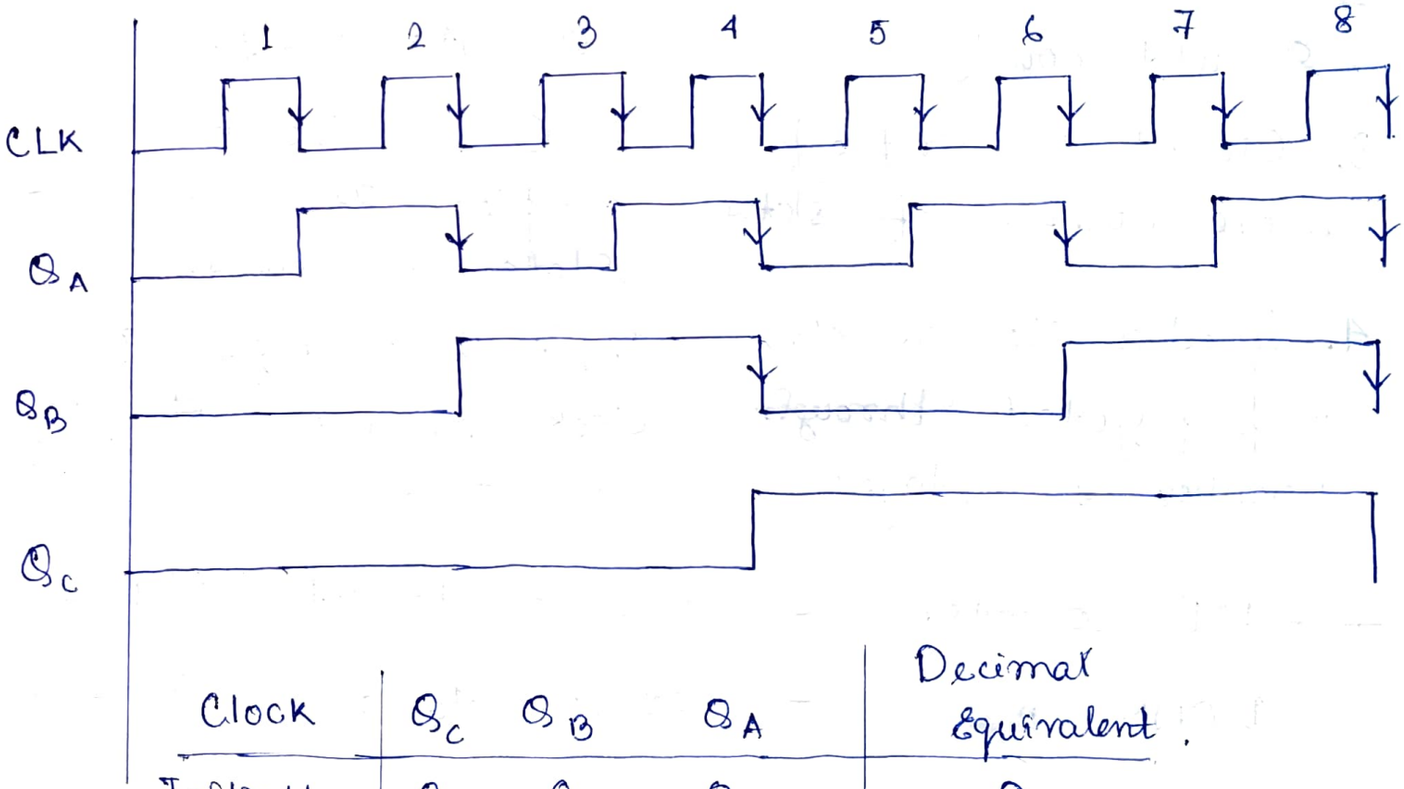
Reset — $J=0, K=1 \Rightarrow Q=0, Q'=1$

Preset — When 1, output 0

Clear — When 0, output will be 0

→ 3 bit Asynchronous UP counter.





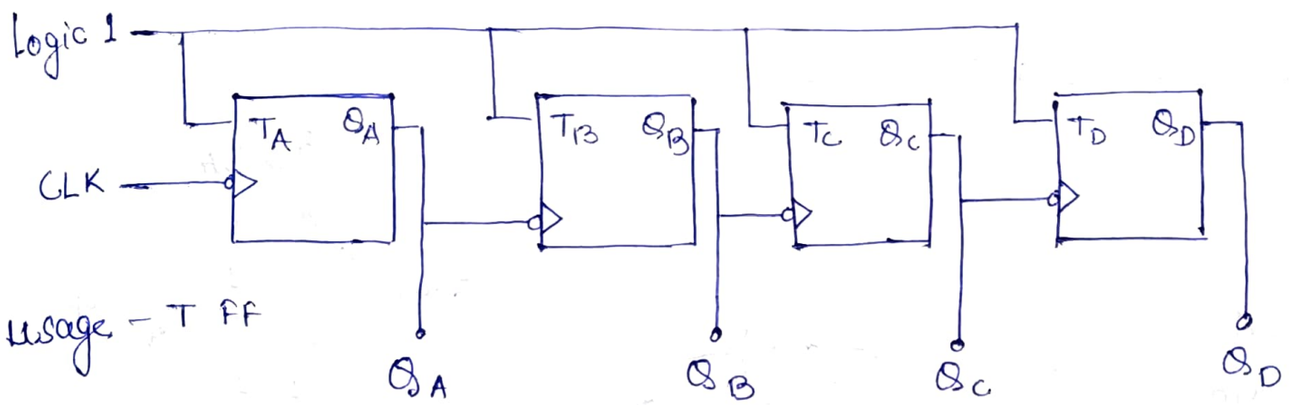
Clock	Q_C	Q_B	Q_A	Decimal Equivalent.
Initially	0	0	0	0
1st (↓)	0	0	1	1
2nd (↓)	0	1	0	2
3rd (↓)	0	1	1	3
4th (↓)	1	0	0	4
5th (↓)	1	0	1	5
6th (↓)	1	1	0	6
7th (↓)	1	1	1	7
8th (↓)	0	0	0	0.

8 states

$2^n = 2^3 = 8$

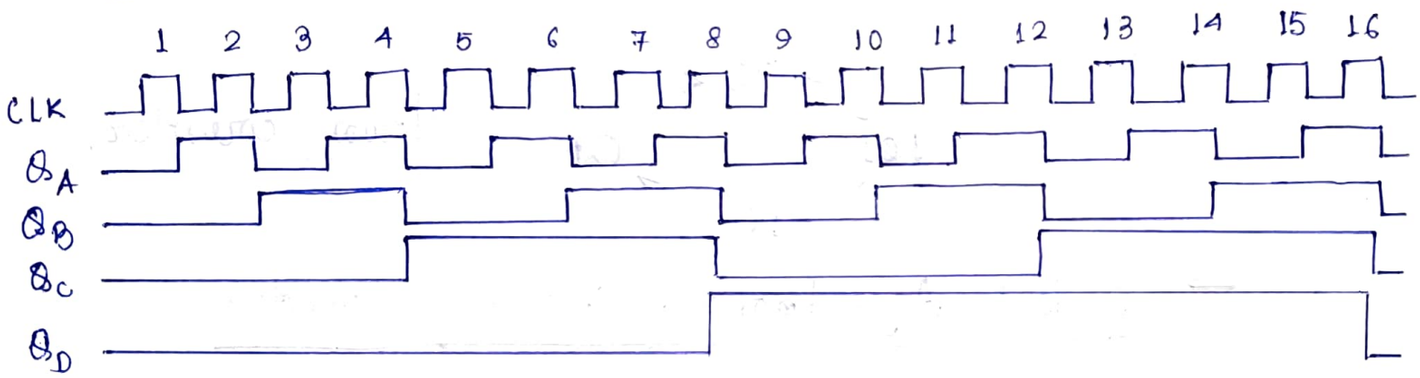
Maximum count = $2^n - 1$

→ 4 bit Asynchronous UP counter.



$2^4 = 16$ states

$2^4 - 1 = 15$ maximum count.



Clock	Q_D	Q_C	Q_B	Q_A	Decimal Equivalent.
Initially	0	0	0	0	0
1st (↓)	0	0	0	1	1
2nd (↓)	0	0	1	0	2.
3rd (↓).	0	0	1	1	3.
⋮					⋮
9th (↓)	1	0	0	1	9
⋮					⋮
15th (↓)	1	1	1	1	15.

→ State diagram of a Counter.

2 bit up counter –

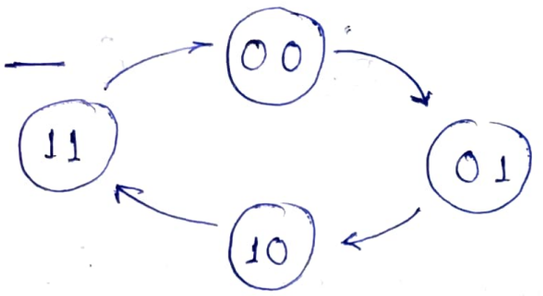
Maximum count = $2^2 - 1 = 3$. (11)

No. of states = $2^2 = 4$.

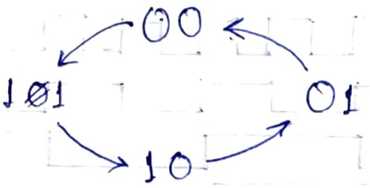
Q_B	Q_A
0	0
0	1
1	0
1	1

down ↑ ↓ up.

State diagram –



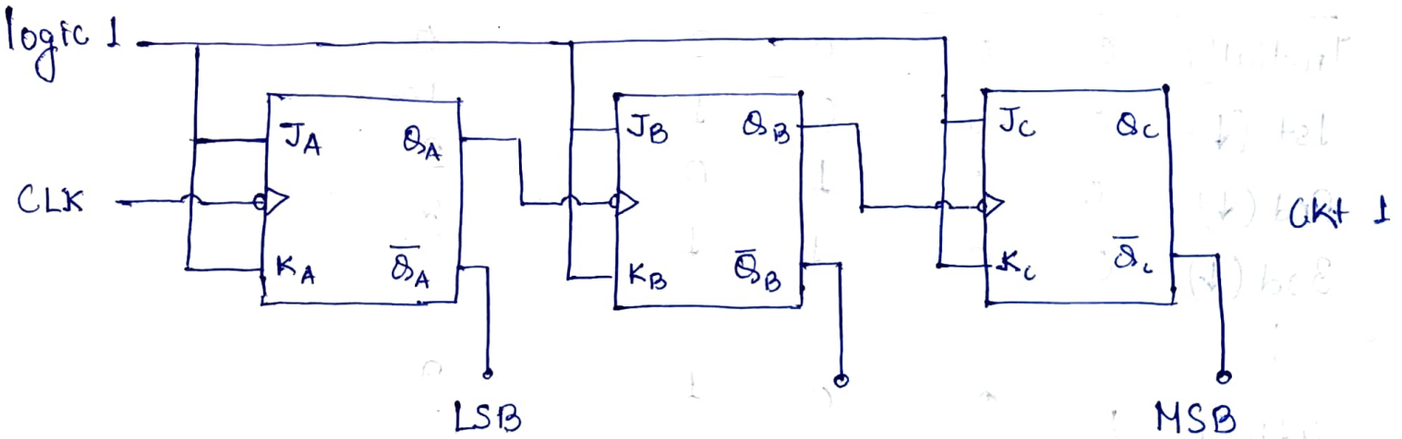
up counter.



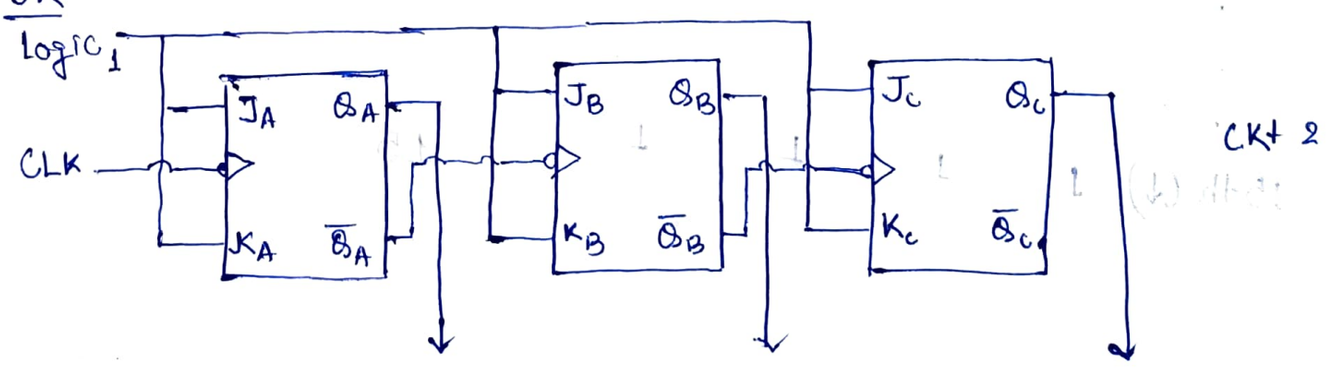
down counter.

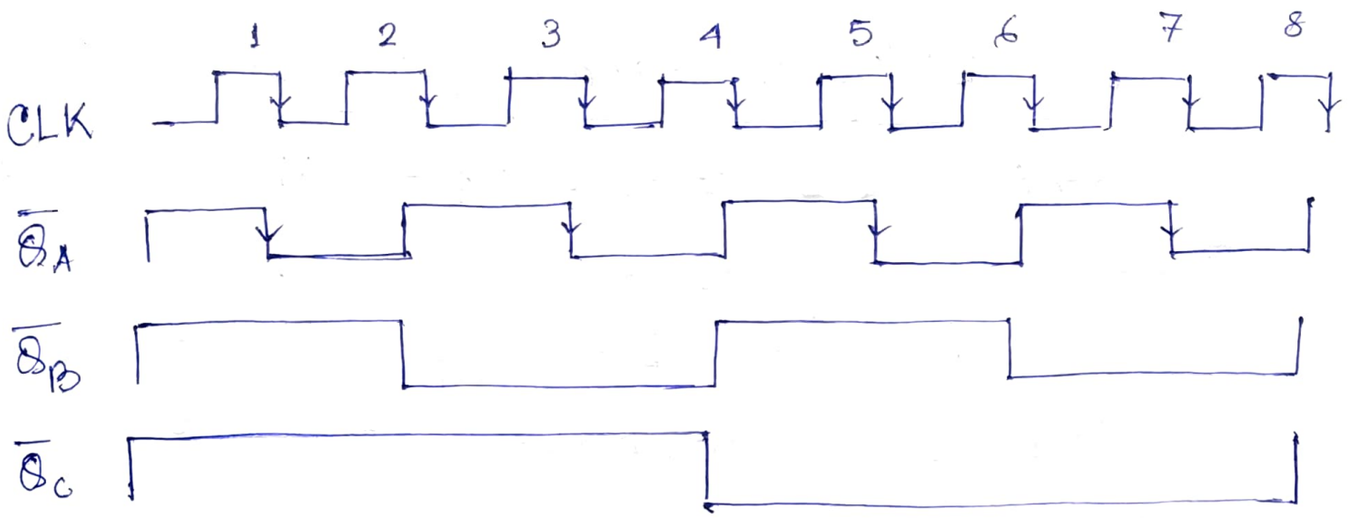
→ 3 bit Asynchronous Down counter.

Counts 7, 6, 5, 4, 3, 2, 1, 0.



OR –

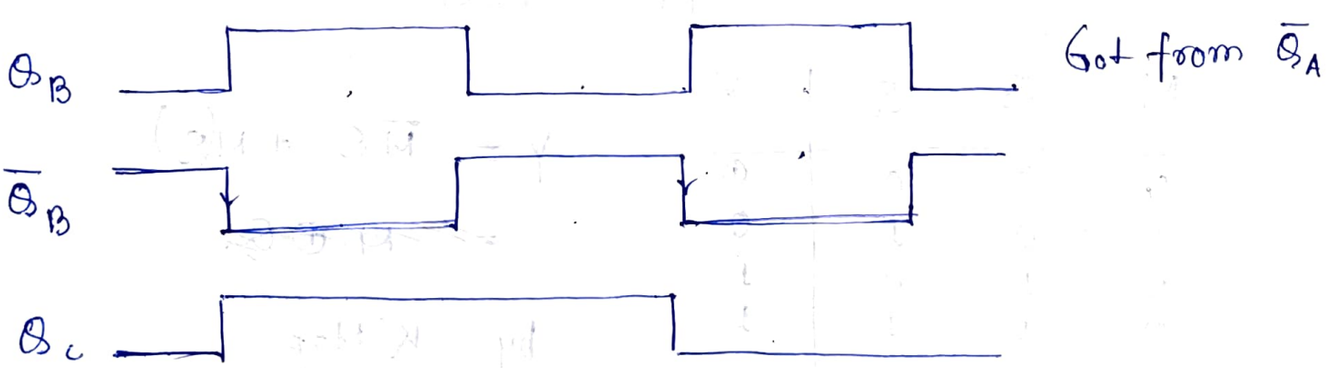




Got by reversing low ↔ high of Q_A, Q_B, Q_C signal.

	UP			DOWN.			
CLK	Q_C	Q_B	Q_A	$\bar{Q}_C$	$\bar{Q}_B$	$\bar{Q}_A$	Decimal.
Initially.	0	0	0	1	1	1	7
1st (↓)	0	0	1	1	1	0	6
2nd (↓)	0	1	0	1	0	1	5
3rd (↓)	0	1	1	1	0	0	4
4th (↓)	1	0	0	0	1	1	3
5th (↓)	1	0	1	0	1	0	2
6th (↓)	1	1	0	0	0	1	1
7th (↓)	1	1	1	0	0	0	0.

For circuit 2., Q_A —



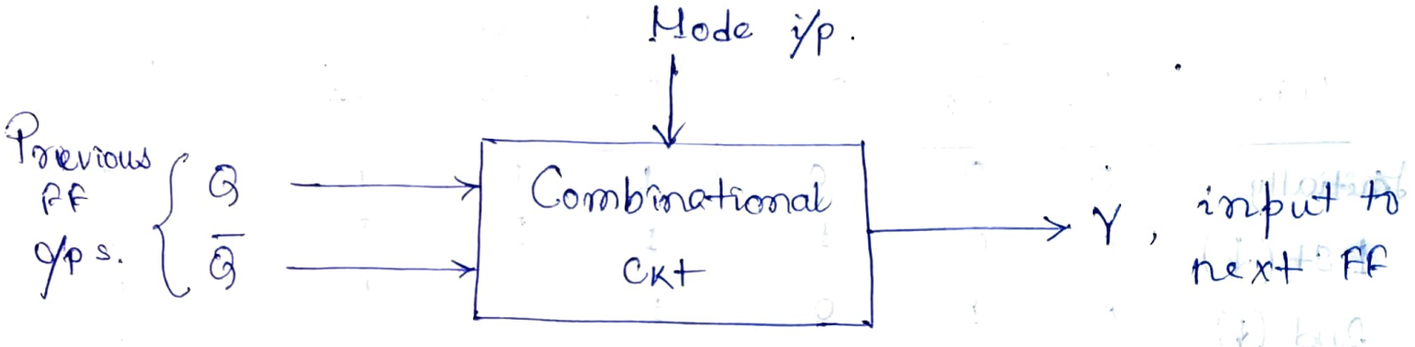
Got from $\bar{Q}_A$

* 3 bit UP/DOWN Ripple Counter :

→ A mode control input (M) is used to select either up or down mode.

In practice both up and down counters are combined into one.

→ A combinational circuit is required between each pair of FFs.



We will set the circuit as when the M = 1, it will process down counting and when M = 0, it will process up counting.

So, M = 0 → Q connected to next FF's clock

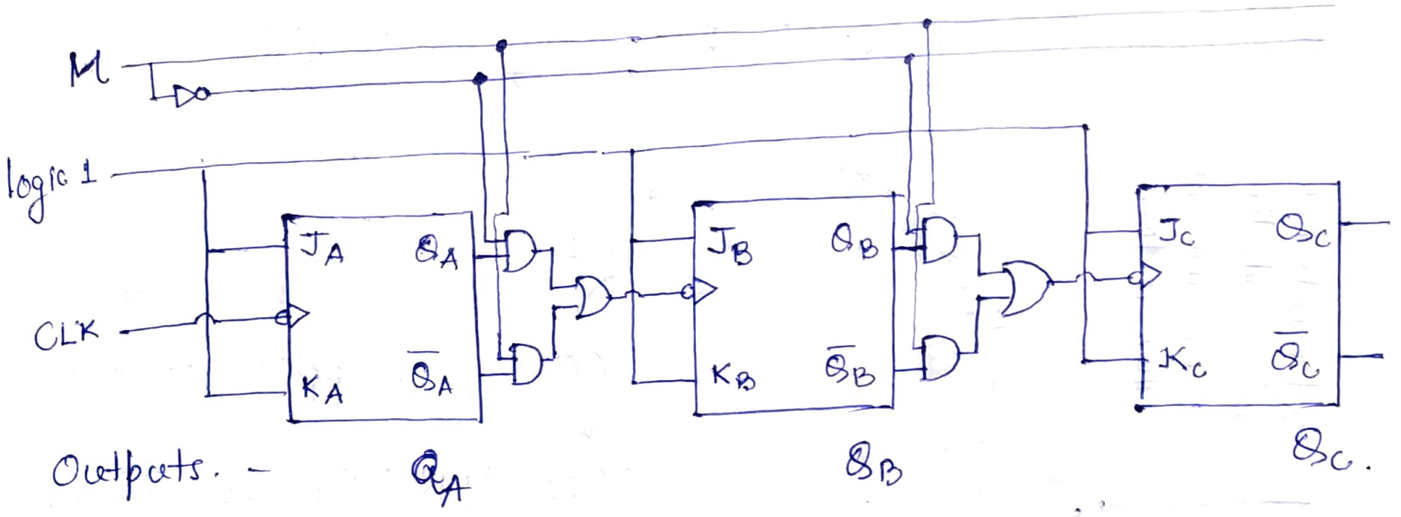
M = 1 → $\overline{Q}$ connected to next FF's clock.

M	Q	$\overline{Q}$	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$Y = \overline{M}Q + M(\overline{Q})$

~~$M \oplus Q$~~

by K' Map



→ Decade (BCD) Ripple Counter.

Important points

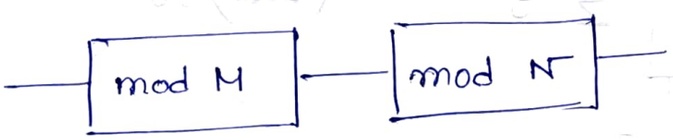
1. Negative Edge Triggered

Q as CLK	$\bar{Q}$ as CLK
↓	↓
UP counter	DOWN counter

Positive Edge Triggered.

Q as CLK	$\bar{Q}$ as CLK.
↓	↓
DOWN counter	UP counter.

2. Cascading of counters.



III

mod MN.

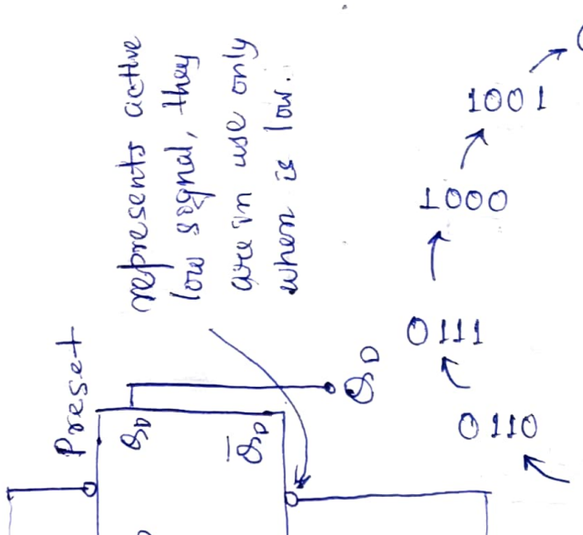
- No of states in BCD counter = 10.

Maximum count = 10 – 1 = 9.

State diagram. MOD-10 using MOD-16

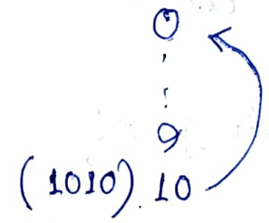
up counting

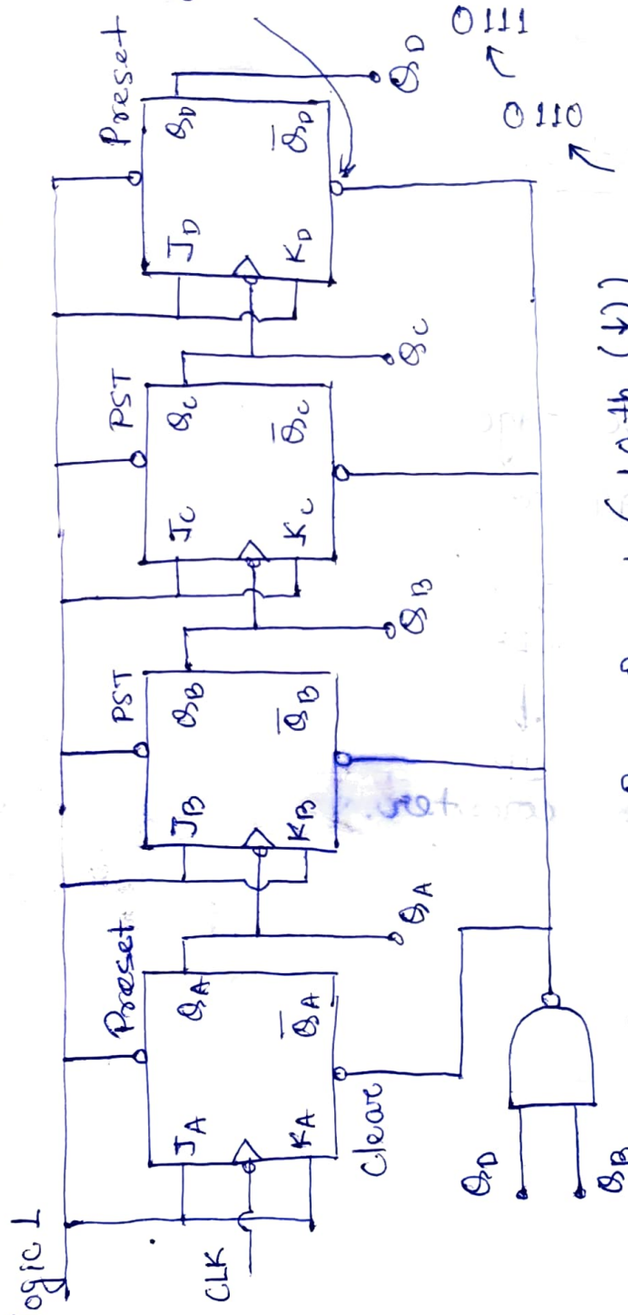
We need 4-bit ripple counter.



CLK	Q_D	Q_C	Q_B	Q_A
Initially	0	0	0	0
1st (↓)	0	0	0	1
2nd (↓)	0	0	1	0
3rd (↓)	0	0	1	1
4th (↓)	0	1	0	0
5th (↓)	0	1	0	1
6th (↓)	0	1	1	0
7th (↓)	0	1	1	1
8th (↓)	1	0	0	0
9th (↓)	1	0	0	1
	1	0	1	0

Decade counter.





→ Modulus of the Counter and Counting up to particular value:

- 2-bit ripple counter is called MOD-4 or modulus-4 counter.
- 3-bit ripple counter is called MOD-8 counter.

$n \rightarrow$ no. of bits $\Rightarrow$ mod no. $= 2^n$. (same as no. of states)

• MOD-6 counter using MOD-8 counter.

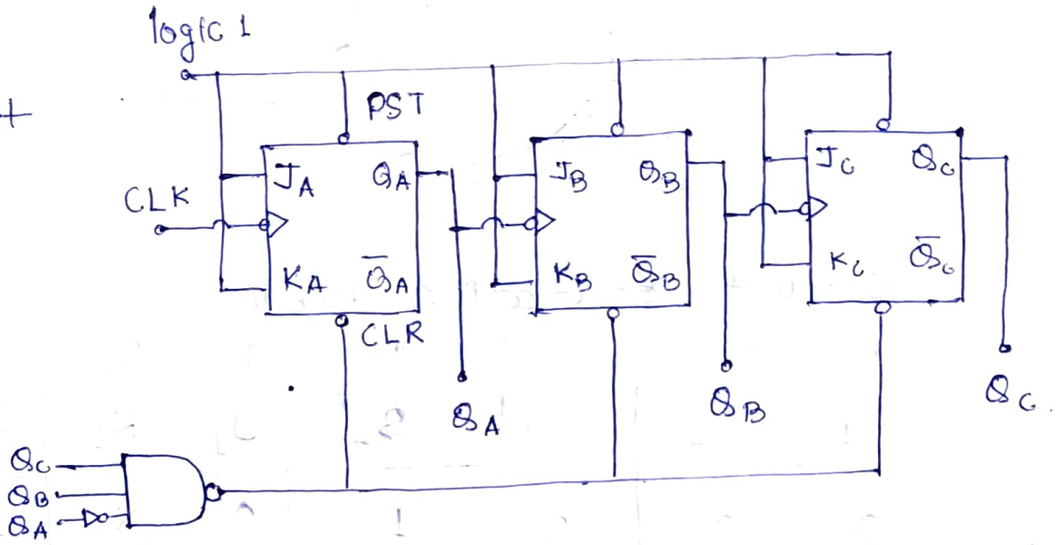
MOD-6

States 6

Max. count = 5.

000	0
001	1
010	2
011	3
100	4
101	5.
110	6

Clear.



↑ This NAND gate is going to reset the FFs as soon as 6 is arrived.

→ Designing Synchronous Counters.

Steps :

1. Decide the number of FFs.
2. Excitation table of FF.
3. State diagram & circuit excitation table.
4. Obtain simplified equation using K' Map.
5. Draw the logic diagram.

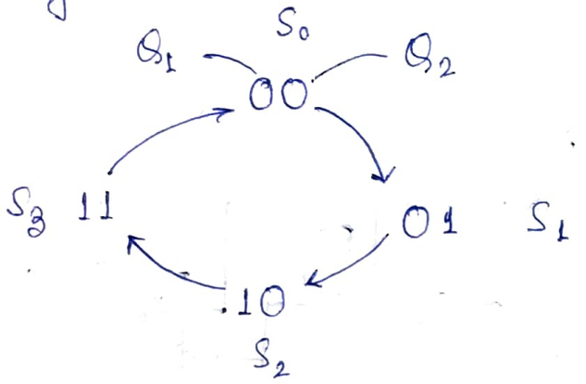
• 2 bit Synchronous up counter.

We need 2 flip-flops.

Excitation table for JK ff :

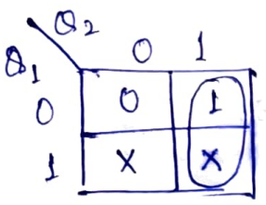
Q_n	Q_{n+1}	J	K.
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0.

State diagram :

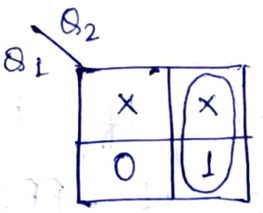


Circuit Excitation table :

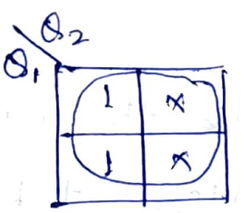
Present St.		Next St.		For Q_1		For Q_2	
Q_1	Q_2	Q_1^*	Q_2^*	J_1	K_1	J_2	K_2
0	0	0	1	0	x	1	x
0	1	1	0	1	x	x	1
1	0	1	1	x	0	1	x
1	1	0	0	x	1	x	1



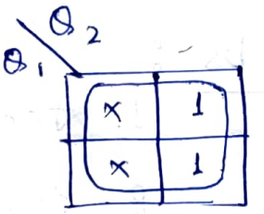
$J_1 = Q_2$



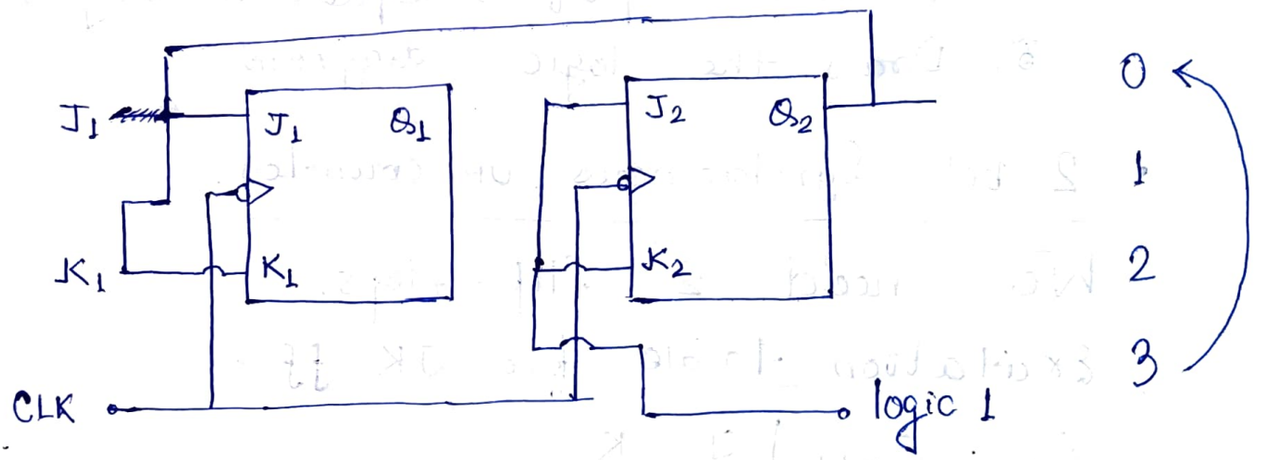
$K_1 = Q_2$



$J_2 = 1$



$K_2 = 1$



Circuit diagram for 2 bit synchronous up counter.

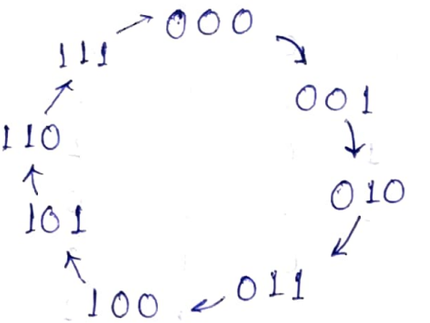
→ 3-bit Synchronous UP Counter :

Number of ffs - 3 Usage - T ff

Excitation Table
T FF

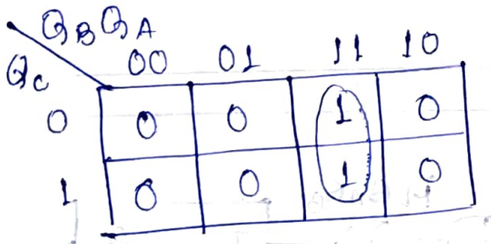
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

State diagram -



Circuit excitation table -

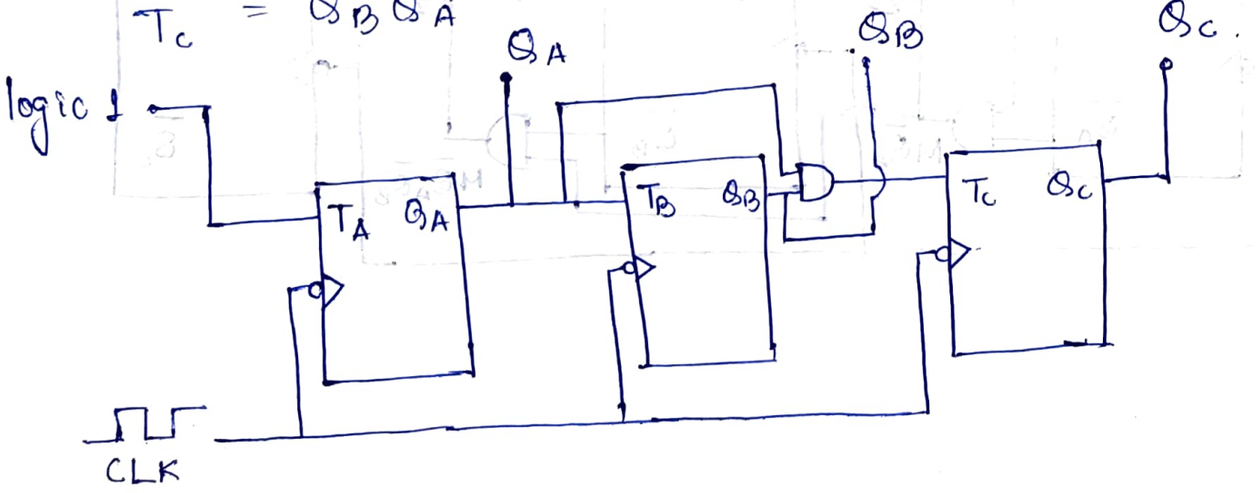
Present St.			Next St.			T_C	T_B	T_A
Q_C	Q_B	Q_A	Q_C^*	Q_B^*	Q_A^*			
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



$T_C = Q_B Q_A$

Similarly, $T_B = Q_A$.

$T_A = 1$



→ 3 bit UP/DOWN Synchronous Counter.

$\begin{cases} M = 0 \rightarrow \text{UP} \\ M = 1 \rightarrow \text{DOWN} \end{cases}$ 8 states.

M	Q_C	Q_B	Q_A	Q_C^*	Q_B^*	Q_A^*	T_C	T_B	T_A
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

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

T_C

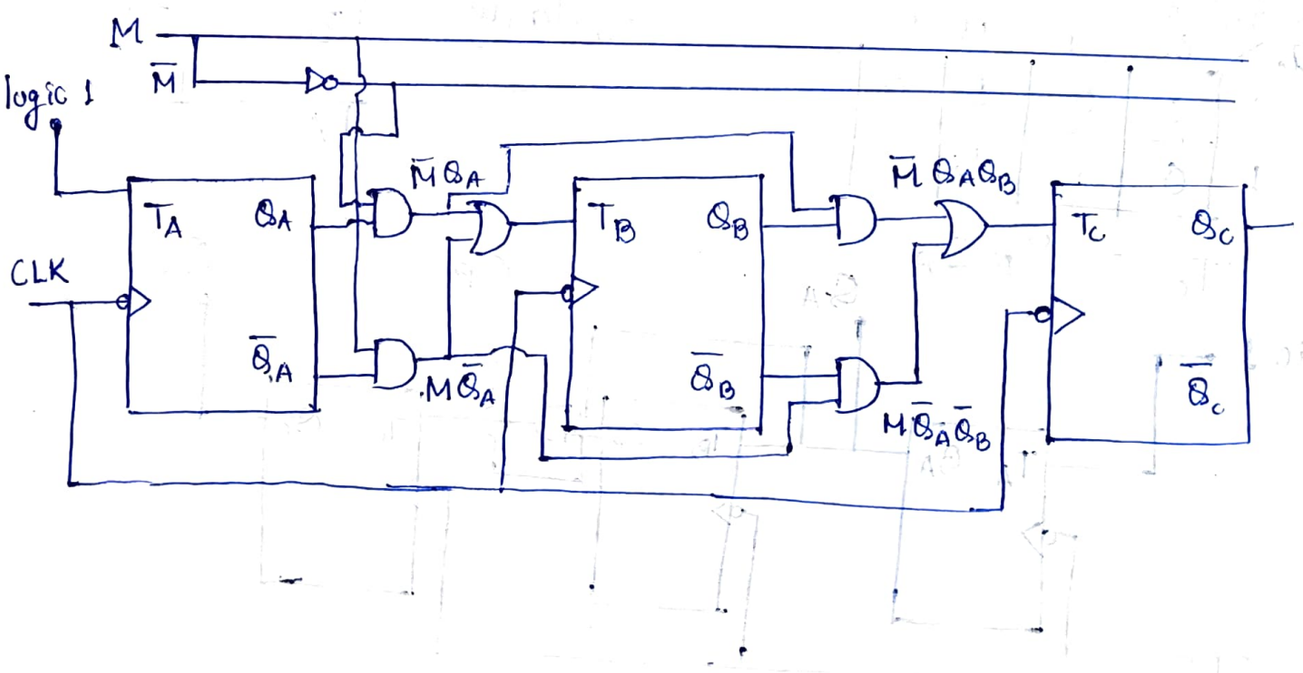
MQ_C \ Q_BQ_A	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	0	0	0
10	1	0	0	0

$$T_C = \bar{M} Q_B Q_A + M \bar{Q}_B \bar{Q}_A$$

Similarly,

$$T_B = M \oplus Q_A = M\bar{Q}_A + Q_A \bar{M}$$

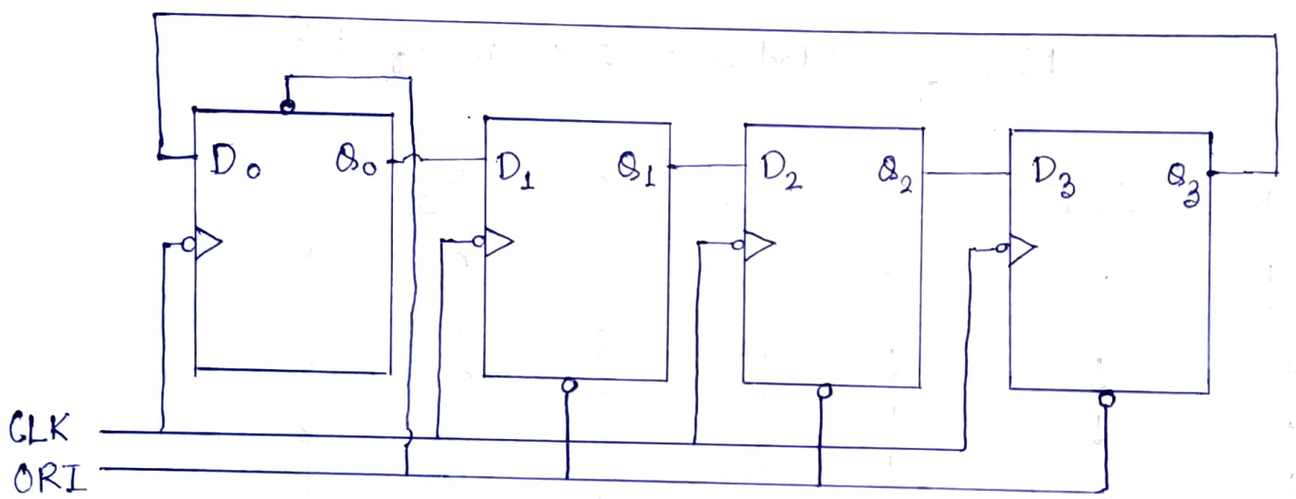
$$T_A = 1.$$



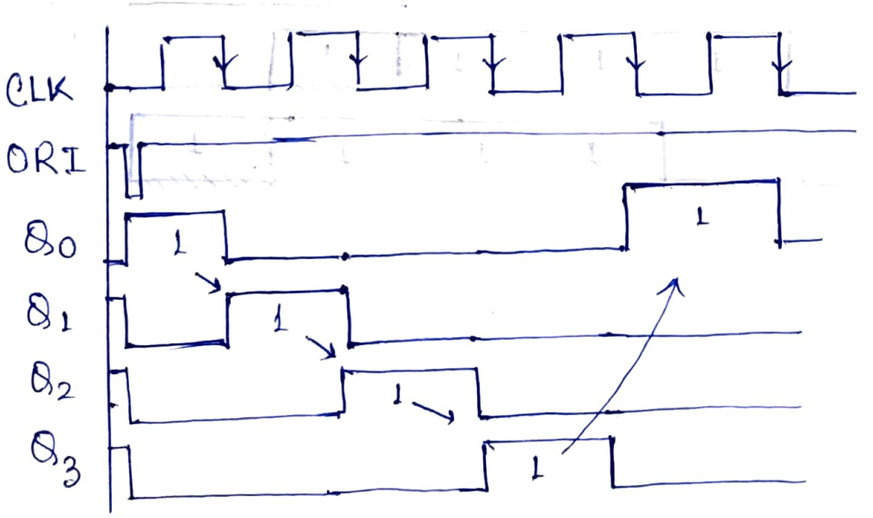
→ Ring Counter.

Typical application of shift register. Only change in ring counter is the output of last FF is connected to the input of first FF.

Number of states for ring counter is equal to the number of FFs used.

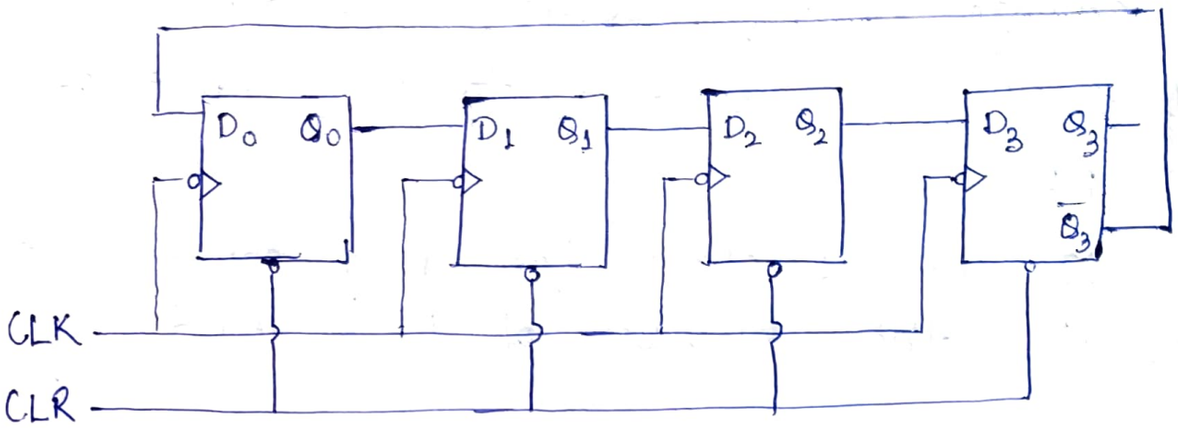


ORI	CLK	Q_0	Q_1	Q_2	Q_3
0	X	1	0	0	0
1	1st ↓	0	1	0	0
1	2nd ↓	0	0	1	0
1	3rd ↓	0	0	0	1
1	4th ↓	1	0	0	0



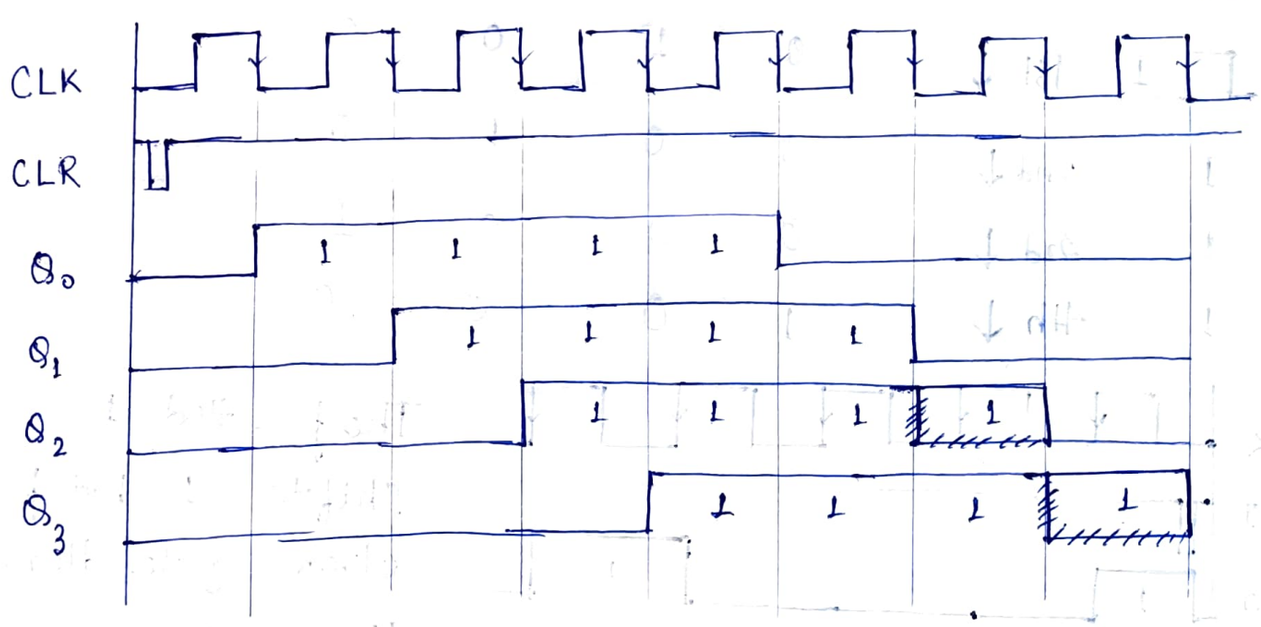
The preset 1 shifts 1 bit per clock cycle. Hence, the name ring counter.

→ Johnson's Counter (Twisted / Switch Tail Ring Counter). Moebius counter



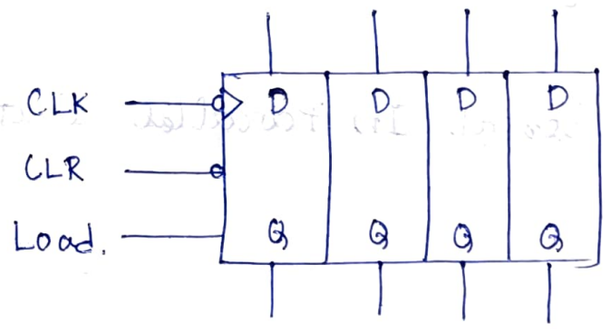
No. of States = 2 × no. of ff.

CLR	CLK	Q_0	Q_1	Q_2	Q_3	
⊓	×	0	0	0	0	1
1	↓	0	0	0	0	2
1	↓	1	0	0	0	3
1	↓	1	1	0	0	4
1	↓	1	1	1	0	5
1	↓	1	1	1	1	6
1	↓	0	1	1	1	7
1	↓	0	0	1	1	8
1	↓	0	0	0	1	
1	↓	0	0	0	0.	



* Registers.

→ Flip-flop is 1 bit memory cell. To increase the storage capacity, we have to use group of ffs. This group of ffs is known as Register. The n-bit register consists of n number of ffs & is capable of storing n-bit word.



We use load line to prevent the change of data.

→ Load is of two types –

i) Synchronous – The ff is operational when the clock is high & load is high.

ii) Asynchronous – The ff is operational only when load is high.

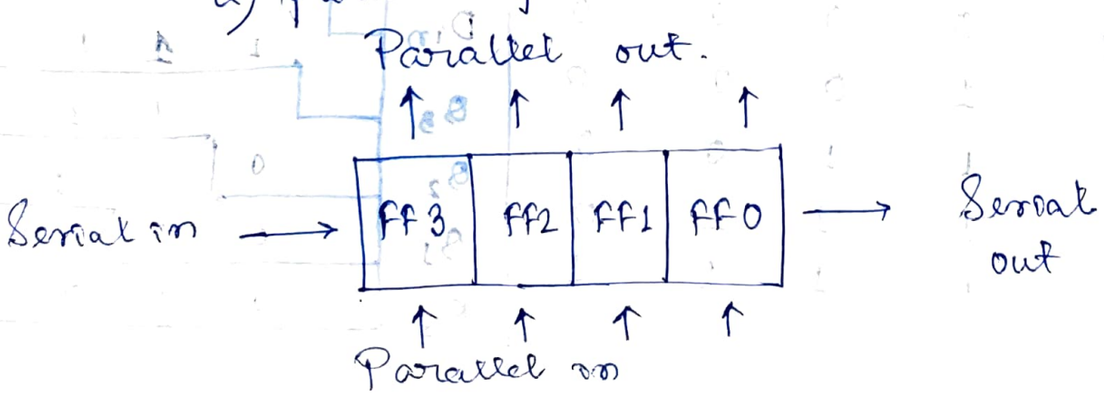
When the load is low there will be no change of data.

→ Data formats.

Data can be entered in

i) Serial form – one bit at a time

ii) parallel form – all bits at a time



→ Serial form — Temporal code

Parallel form — Special code

→ Classification of registers.

i) Depending on I/P & O/P.

a) SISO - Serial In Serial Out.

We enter the data in serial form & extract in serial form

b) SIPO - Serial In Parallel Out

c) PISO.

d) PIPO.

ii) Depending on application

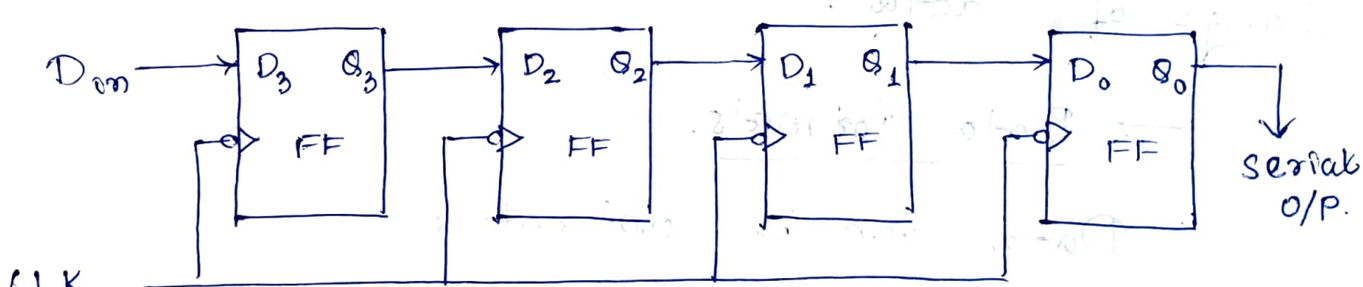
a) Shift Register

b) Storage Register (PIPO).

→ Shift Register.

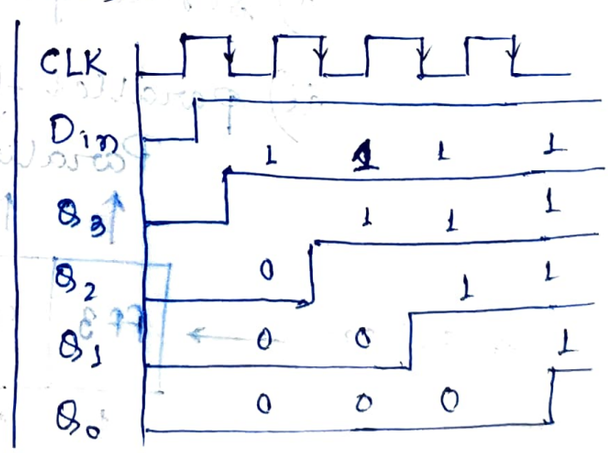
a) SISO

Shift Right Mode (Input is provided from left and stored value shifted to right)



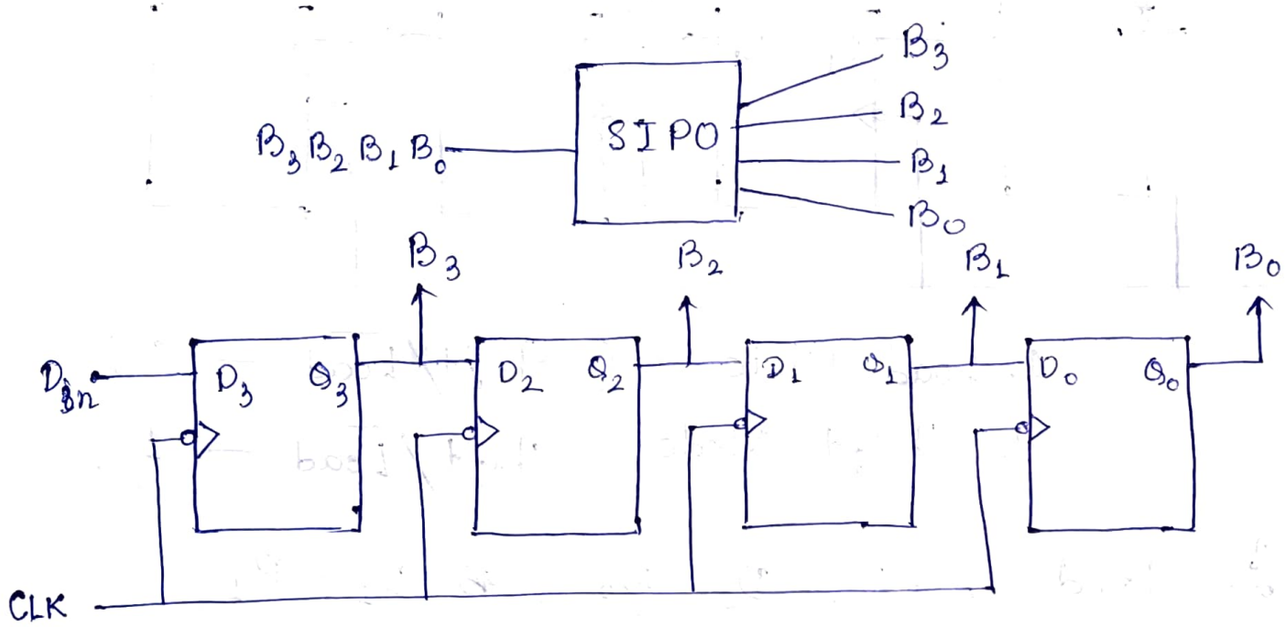
We want to store (1111).

CLK	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1st ↓	1	0	0	0
2nd ↓	1	1	0	0
3rd ↓	1	1	1	0
4th ↓	1	1	1	1



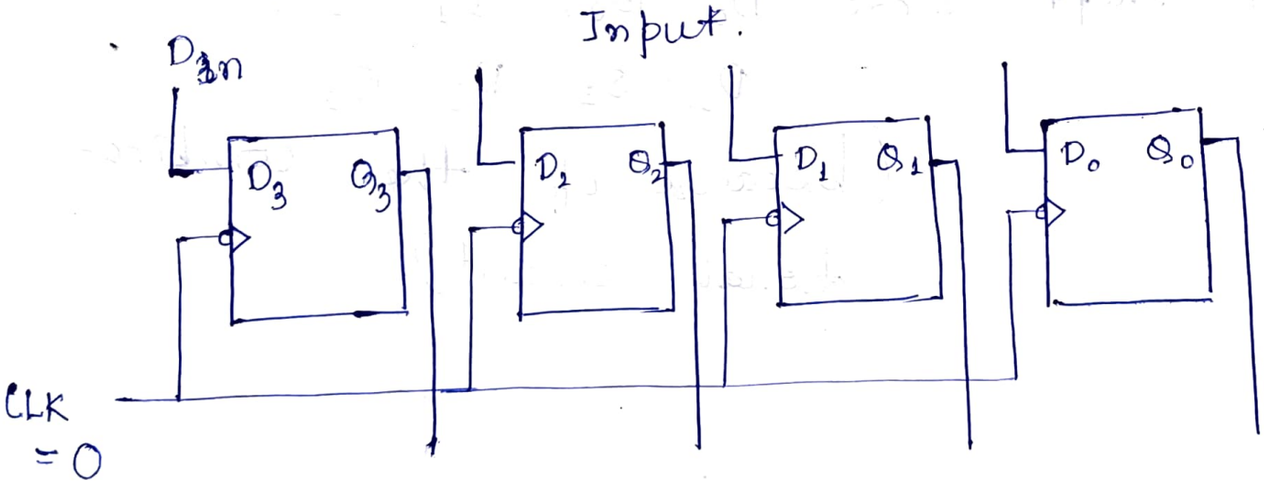
We need more than 4 clock pulses to have the data out.

b) SIPO Advantage of SIPO is we don't need more than 4 clock pulses to obtain the data and the data is out once the last clock pulse is reached



c) PIPO (Storage register / buffer register).

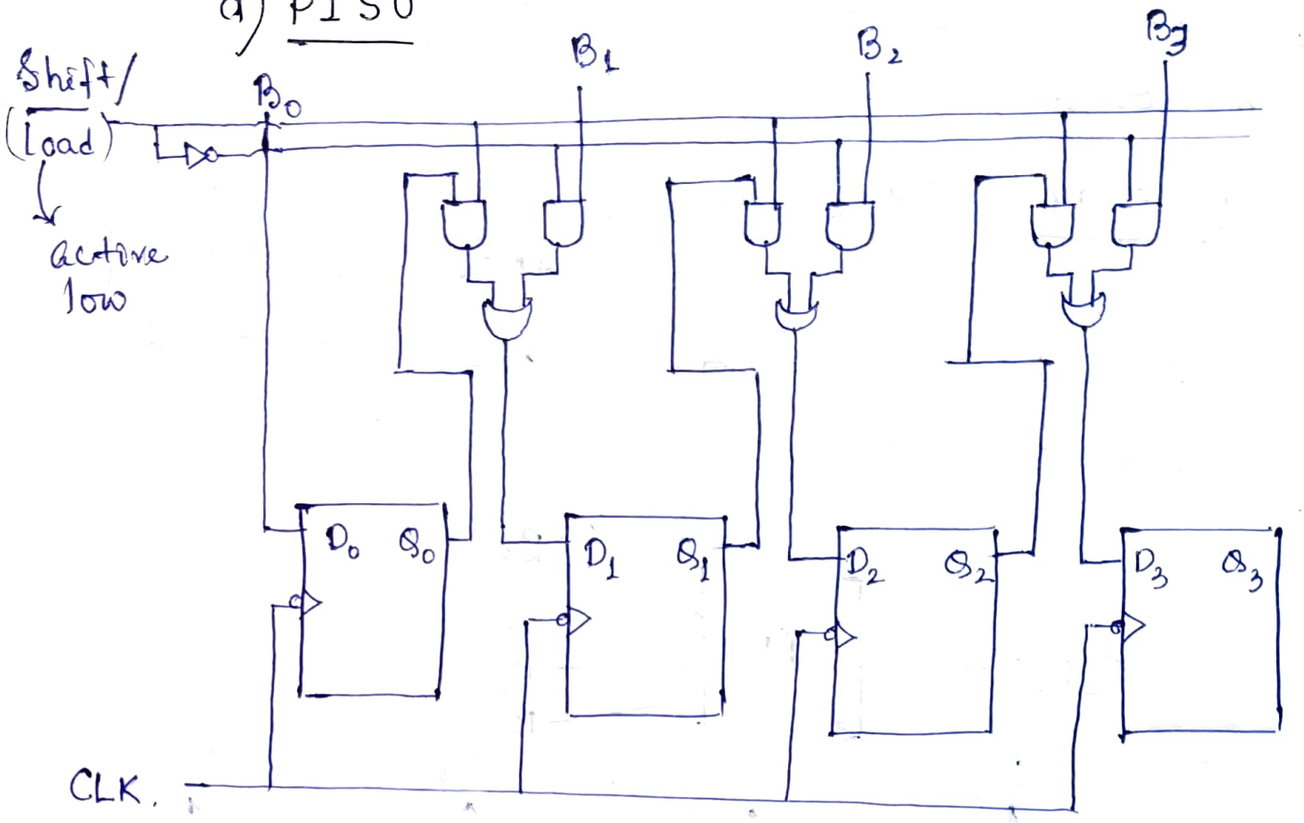
Input.



output.

We require 1 clock pulse to store the data.

d) PISO



1. Load Mode : (Shift/$\overline{Load}$ — 0).
2. Shift Mode. (Shift/$\overline{Load}$ — 1).

In load mode, $D_0 = B_0$, $D_1 = B_1$,

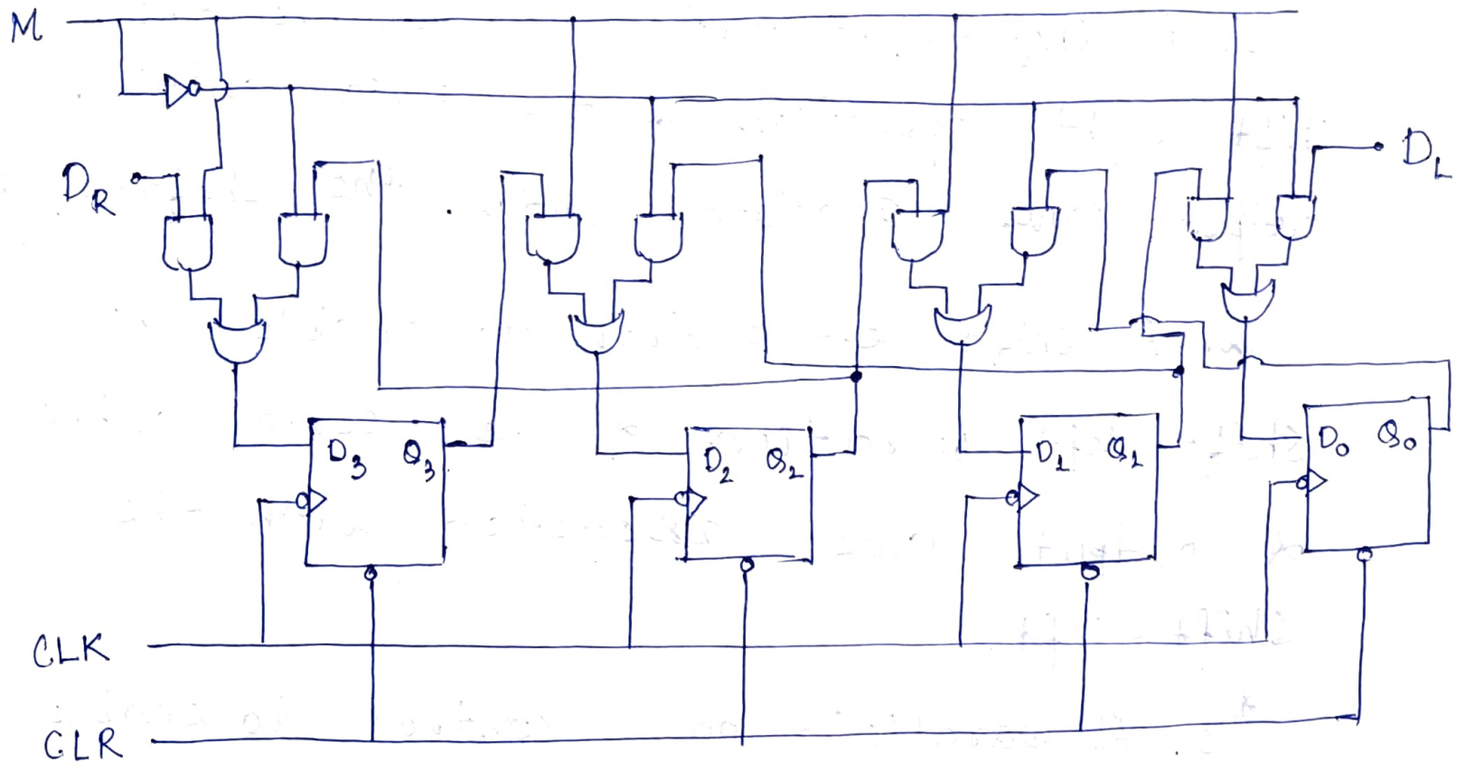
$D_2 = B_2$, $D_3 = B_3$.

In shift mode, $D_0 = B_0$, $D_1 = Q_0$,

$D_2 = Q_1$, $D_3 = Q_2$.

(because of the combinational circuit).

→ Bodirectional Shift Register.



When M = 1, shift right operation

When M = 0, shift left operation

D_R → Serial shift right input

D_L → Serial shift left input.

We can perform multiplication or division by this register.

→ Universal Shift Register.

If register has both the shift-right & shift-left capabilities, along with the necessary input and output terminals for parallel transfer, then it is called shift register with parallel load or Universal Shift Register.

The most general shift register has all these capabilities :

1. A shift-right control to enable the shift-right operation & the serial input & output lines associated with the shift-right.

2. A shift-left control to enable the shift-left operation & the serial input & output lines associated with the shift-left.

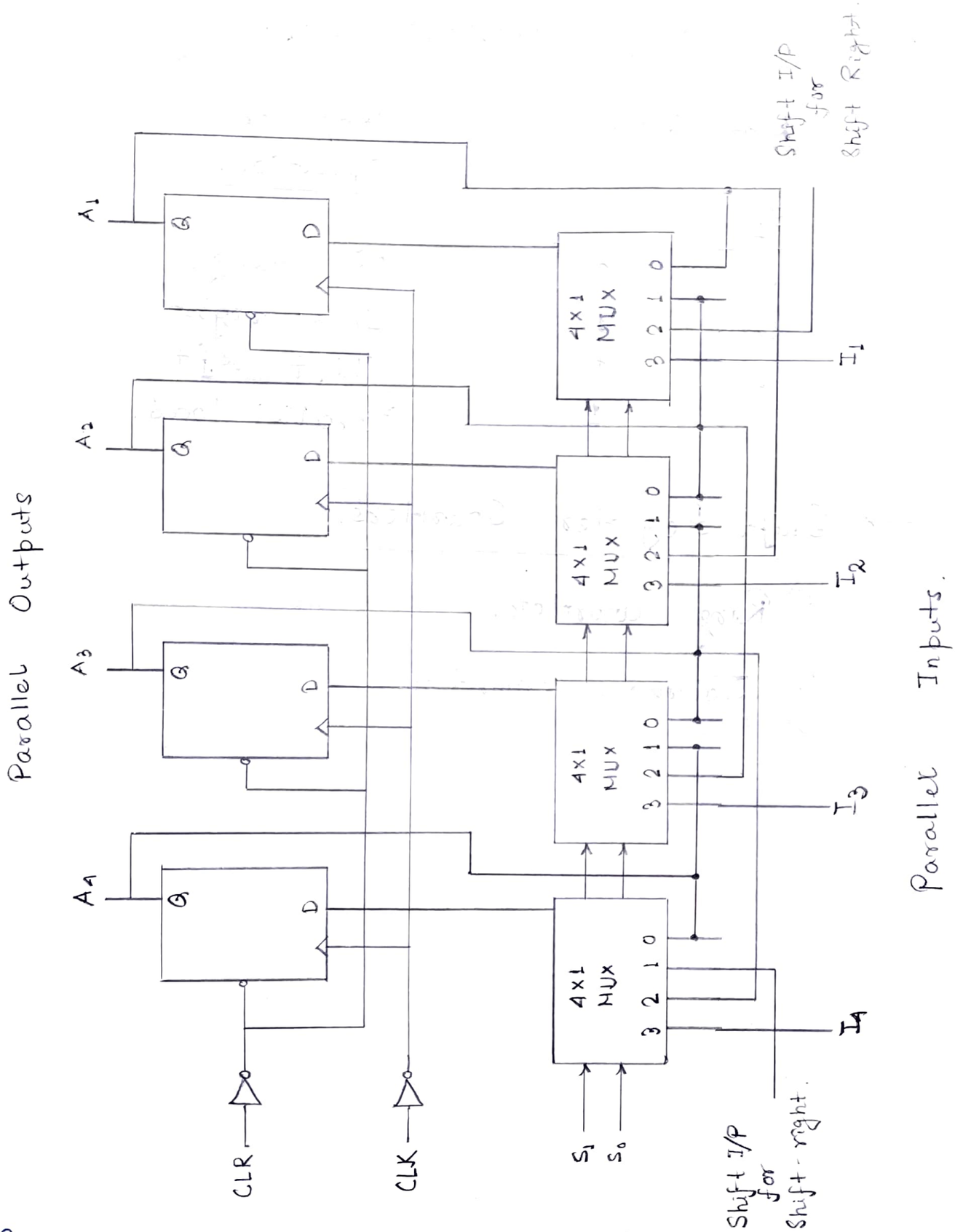
*3. A parallel-load control to enable a parallel transfer & the n input lines associated with the parallel transfer.

4. n parallel output lines.

5. A clear control to clear the register to 0.

6. A CLK input for clock pulses to synchronise all operations.

7. A control state that leaves the information in the register unchanged even though clock pulses are continuously applied.



Basic connections :

1. Zeroth pin of MUX is connected to the output pin of the corresponding FF.
2. 1st pin of MUX is connected to the o/p of the very-previous FF which facilitates the right shift.
3. 2nd pin of MUX is connected to the o/p of the very-next FF which facilitates the left shift.
4. 3rd pin of MUX is connected to the individual bits of the input data which facilitates parallel loading.
5. 4 MUXs have common selection lines.

Function table for universal shift register.

Mode Control		Register Operation.
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load.

→ Shift Register Counters.

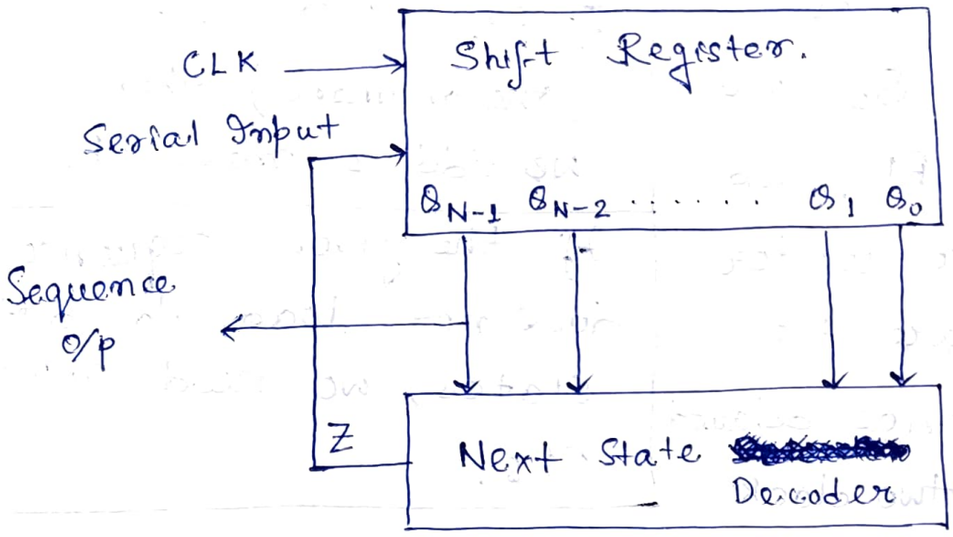
i) Ring counter.

ii) Johnson counter.

* Sequence Generator.

→ A sequence generator is a circuit that generates a desired sequence of bits in synchronisation with a clock.

A sequence generator can be used as a random bit generator, code generator & prescribed period generator.



Output of the next state decoder (Z) is a function of $Q_{N-1}, Q_{N-2}, \ldots, Q_1, Q_0$. This sequence generator is similar to a ring counter or a Johnson counter.

→ Design of a 4-bit Sequence Generator.

Consider design of a sequence generator to generate 1001.

① To design, we first calculate number of FFs we need by this [illegible] –

$$N \leq 2^n - 1.$$

$n \rightarrow$ no. of FFs. (min.)

$N \rightarrow$ Length of sequence.

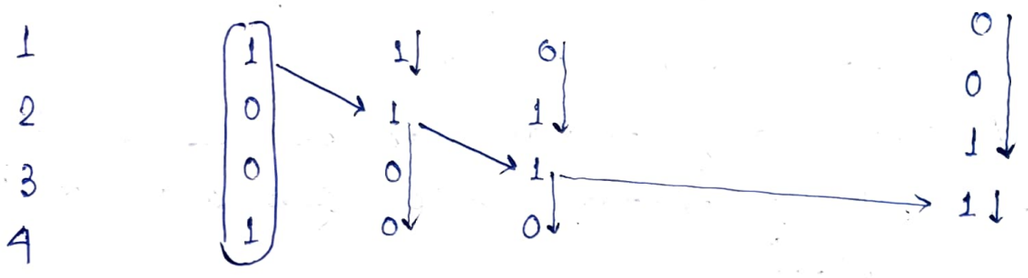
For 1001,

$4 \leq 2^n - 1$

$n = 3$

State table building –

② CLK | FF Outputs Q_2 Q_1 Q_0 | Serial Input (Feedback o/p) D_2 = Z



Given sequence is listed under Q_2, i.e. Q of MSB FF, and the sequence under Q_1 & Q_0 are the same sequence delayed by one & two clock pulses respectively.

If any state repeats (one or more) then we add 1 more FF.

If the given sequence does not lead to distinct states, we need more FFs.

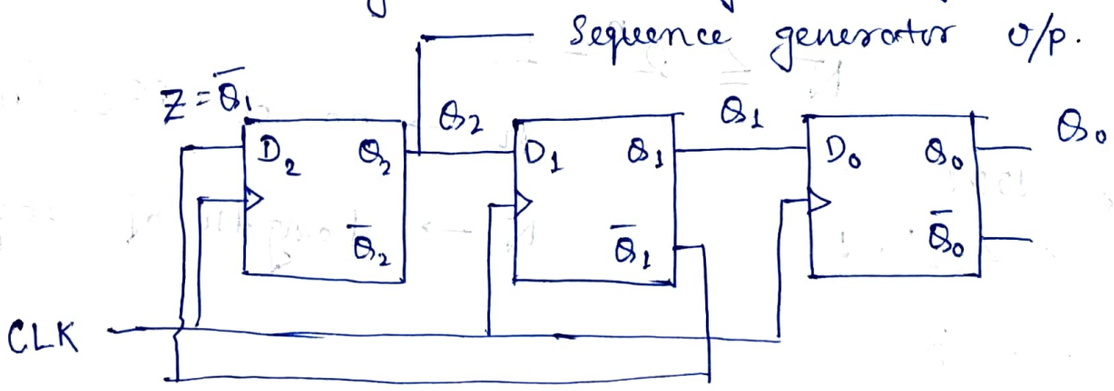
Last column (Z) gives the serial input required at the shift register.

③ K' Map for serial Input. –

Q_2 \ Q_1Q_0	00	01	11	10
0	X	1	0	X
1	1	X	X	0

$Z = \bar{Q}_1$

④ Now, building the logic diagram –



→ Design of a 5-bit Sequence Generator.

Sequence – 10011.

1. No. of FFs needed.

$$5 \leq 2^n - 1$$

$$\Rightarrow n >= 3$$

2. State table –

CLK	Q_2	Q_1	Q_0	Z
1	1	1	1	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1
5	1	1	0	1

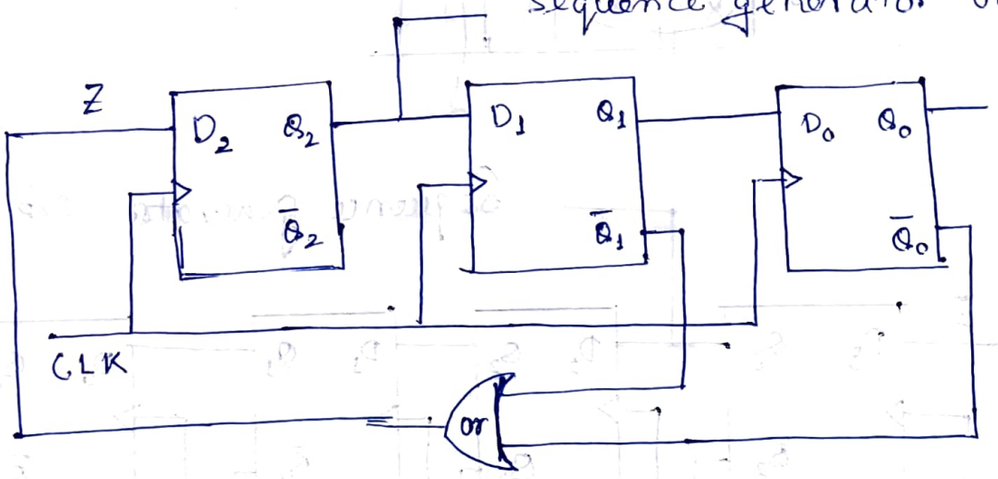
all distinct states → No more FFs needed

3.

Q_2 \ Q_1Q_0	00	01	11	10
0	X	1	0	X
1	1	X	0	1

$$Z = \bar{Q}_1 + \bar{Q}_0$$

4. Sequence generator output.



→ Design of a 6-bit Sequence Generator.

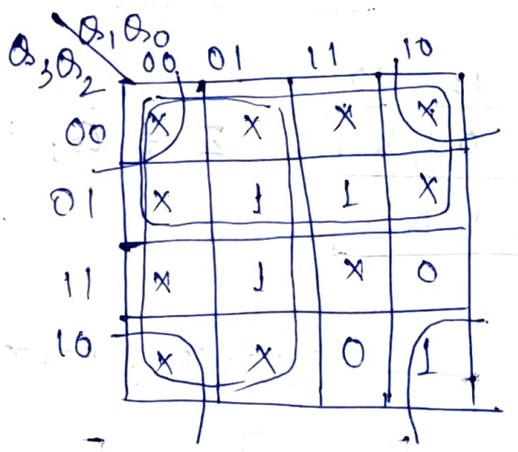
(110101)

1. $6 \leq 2^n - 1 \Rightarrow n \geq 3$.

2. State Table

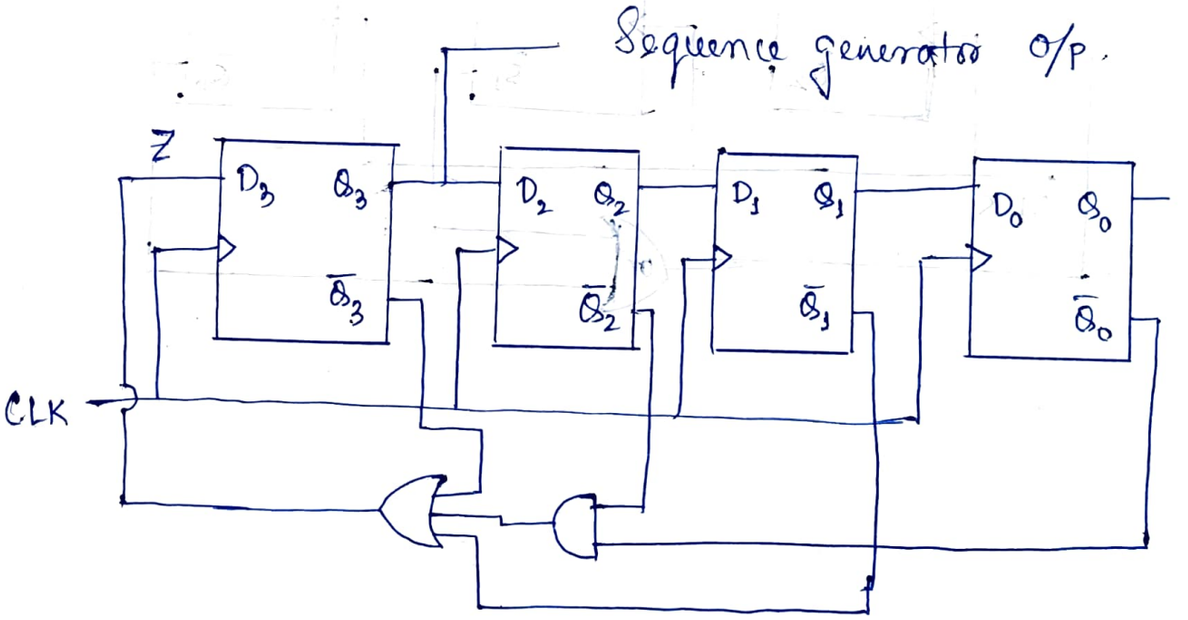
CLK	Q_3	Q_2	Q_1	Q_2^*	Q_1^*	Q_0^*	Q_0	Z
1	1	1	0	1	1	0	1	1
2	1	1	1	1	1	1	0	0
3	0	1	1	1	1	1	1	1
4	1	0	1	1	0	1	0	0
5	0	1	0	1	1	0	1	1
6	1	0	1	0	1	0	1	1

3.



$$Z = \bar{Q}_3 + \bar{Q}_1 + \bar{Q}_2\bar{Q}_0$$

4.



→ Sequence generator. $0 \to 3 \to 5 \to 6 \to 0$.

1. Excitation Table –

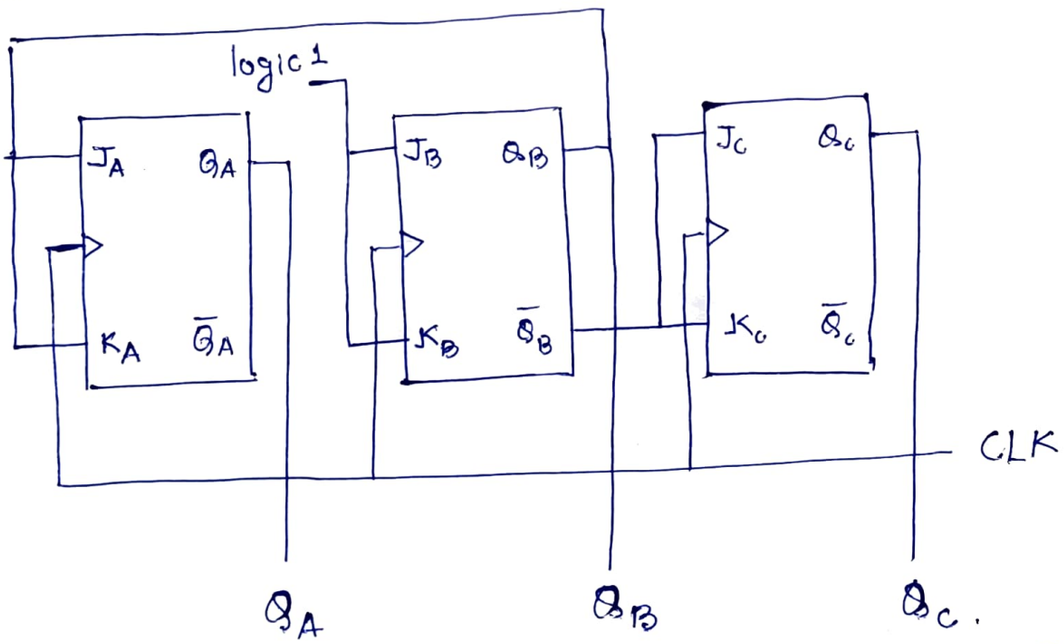
	Present St.			Next St.			FF I/Ps.					
	Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	1	0	X	1	X	1	X
3	0	1	1	1	0	1	1	X	X	1	X	0
5	1	0	1	1	1	0	X	0	1	X	X	1
6	1	1	0	0	0	0	X	1	X	1	0	X

2. by K' Map.

$J_A = Q_B$ $\quad J_B = 1$ $\quad J_C = \bar{Q}_B$

$K_A = Q_B$ $\quad K_B = 1$ $\quad K_C = \bar{Q}_B$

3. Logic Diagram.



Lock out condition : The logic states 001, 010, 100, 111 are not used here. If by chance, the counter happens to find itself in any of the unused states, its next state would be unknown. It may just be possible that the counter might go from one unused state to another & never arrive at a used state. A counter whose unused states have this feature is said to suffer from LOCK OUT.

To avoid Lock out & make sure that at the starting point the counter is in its initial state or it comes to its initial state with few clock cycles, external logic circuitry is to be provided & so we design the counter assuming the next state to be the initial state from each unused states.

Memory and Programmable Logic.

* A memory unit is a device to which binary information is transferred for storage and from which information is retrieved when needed for processing.

It is a collection of cells capable of storing a large quantity of binary information.

* Types of memories –

i) Random access memory (RAM): RAM stores new information for later use. RAM can perform both write and read operations. (Process of storing new information into memory is referred to as a memory write operation) (Process of transferring the stored information out of memory is referred to as a memory read operation).

ii) Read only Memory (ROM): Suitable binary information is already stored inside memory & can be retrieved or read at any time. That information can't be altered by writing.

* Programmable Logic Devices (PLD).

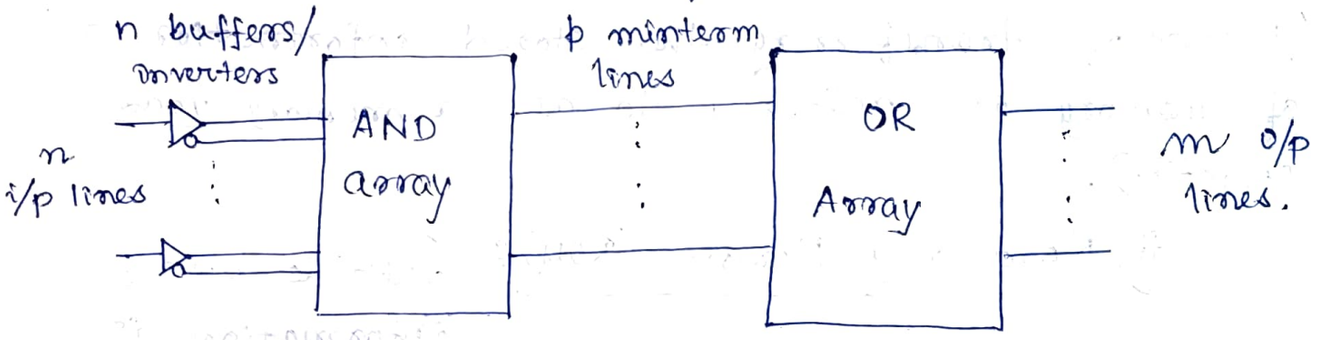
An electronic component made of array of AND gates, array of OR gates that is used to build reconfigurable logic devices or digital circuits. Unlike a logic gate, that has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed, that is, reconfigured.

Process of entering the information into these devices is known as programming (hardware programming - not software programming).

→ Advantages of using PLD are –

1. Reduced space requirements.
2. Reduced power requirements.
3. Design security
4. Compact circuitary
5. Short design cycle
6. Low development cost
7. Higher switching speed
8. low production cost for large-quantity production.

→ General structure of PLD –



→ Classification :

1. PLA (Programmable Logic Array)
2. PAL (Programmable array logic)
3. PROM (Programmable ROM)
4. GAL (Generic Array Logic)
5. CPLD (Complex PLD)
6. FPGA (Field Programmable Gate Array)

→

Device type	AND array	OR array.
ROM	Fixed	Programmable
PLA	Programmable	Programmable
PAL	Programmable	Fixed.

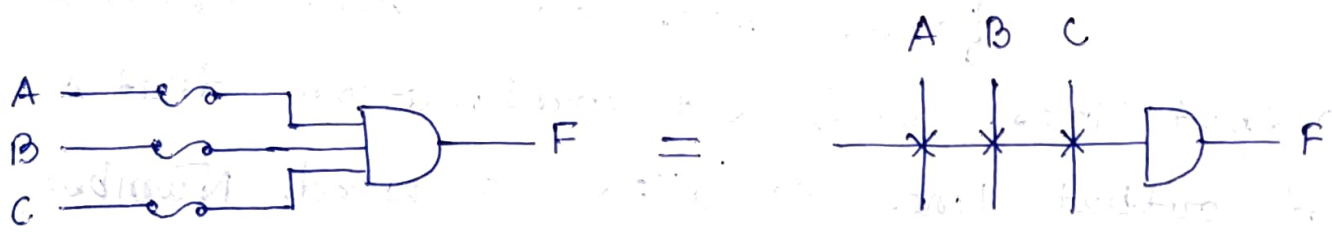
→ 

Conventional symbol

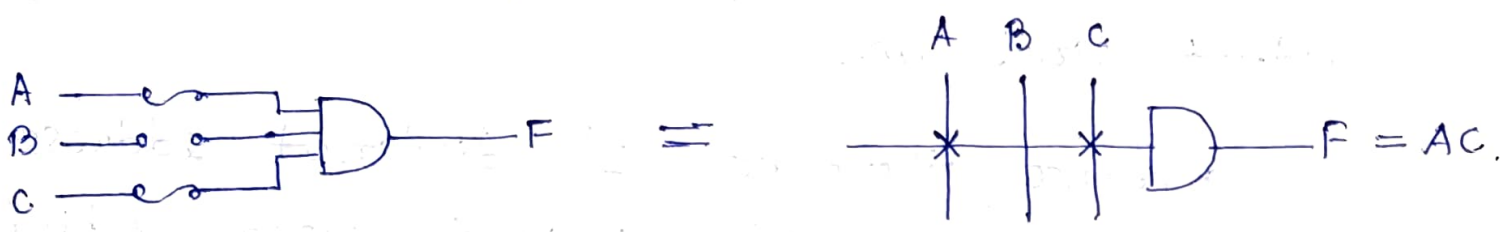
Array logic symbol

→ **PLD Notation :**

Cross-marked junctions represent the fusible joints while junctions with dots indicate permanent junctions that are not fusible.



All fuses are intact.

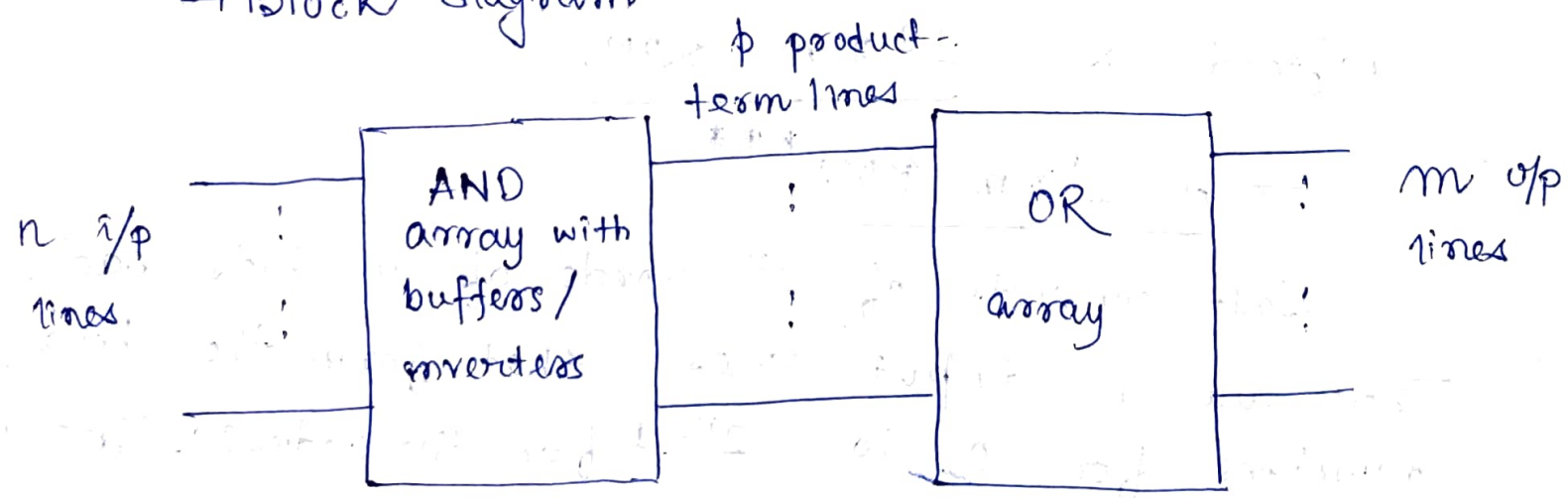


Fuse B is blown to obtain F = AC.

* **PROM :** If ROM has programmable feature, then it is called PROM.

The user has the flexibility to program the binary information electrically once by using PROM programmer. Once a pattern is established for a ROM, it remains fixed even if the power supply to the circuit is switched off & then switched on again.

→ Block diagram –



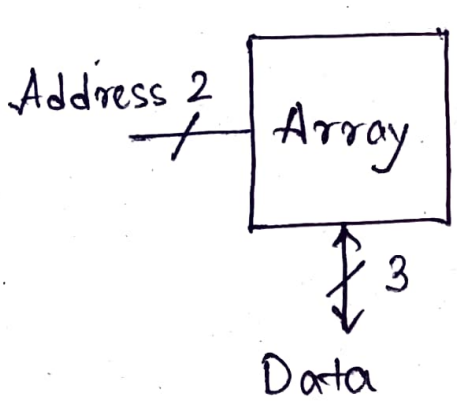
* Memory Arrays.

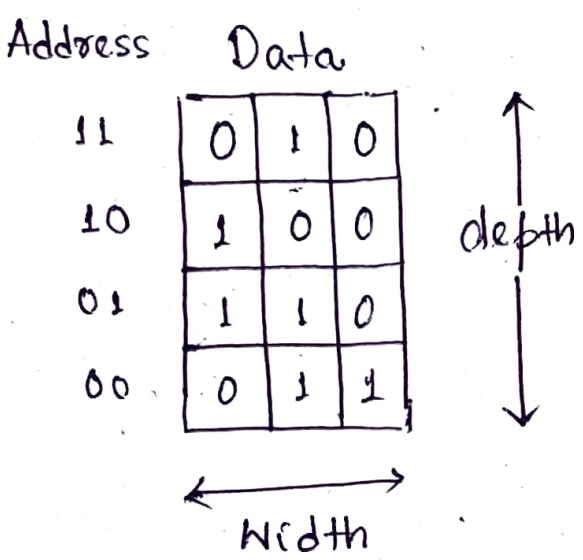
→ Types - DRAM, SRAM, ROM.

→ Two dimensional array of bit cells, where each bit cell stores one bit.

→ An array with N address bits & M data bits
- 2^N rows and M columns
- array size : $2^N \times M$ (depth x width).

→ Example - $2^2 \times 3$ bit-array.





Number of words = $2^2 = 4$.

Word size = 3 bits.

3-bit word stored at address 10 is 100.

→ No. of words = 2^N.

Word size = M.

Address lines = N

→ Address : Each bit combination of input variables is called an address. Address is essentially a binary number that denotes one of the minterms of n variables.

It is possible to generate $p = 2^n$ number of distinct addresses from n number of input variables.

Word : Each bit combination of input variables forms respective bit combination at output lines. Each bit combination that is formed at output lines is called a word. Number of bits per word is equal to the number of output lines m.

→ Since there are 2^n distinct addresses in a ROM, there are 2^n distinct words which are stored in the device & an output word can be selected by a unique address.

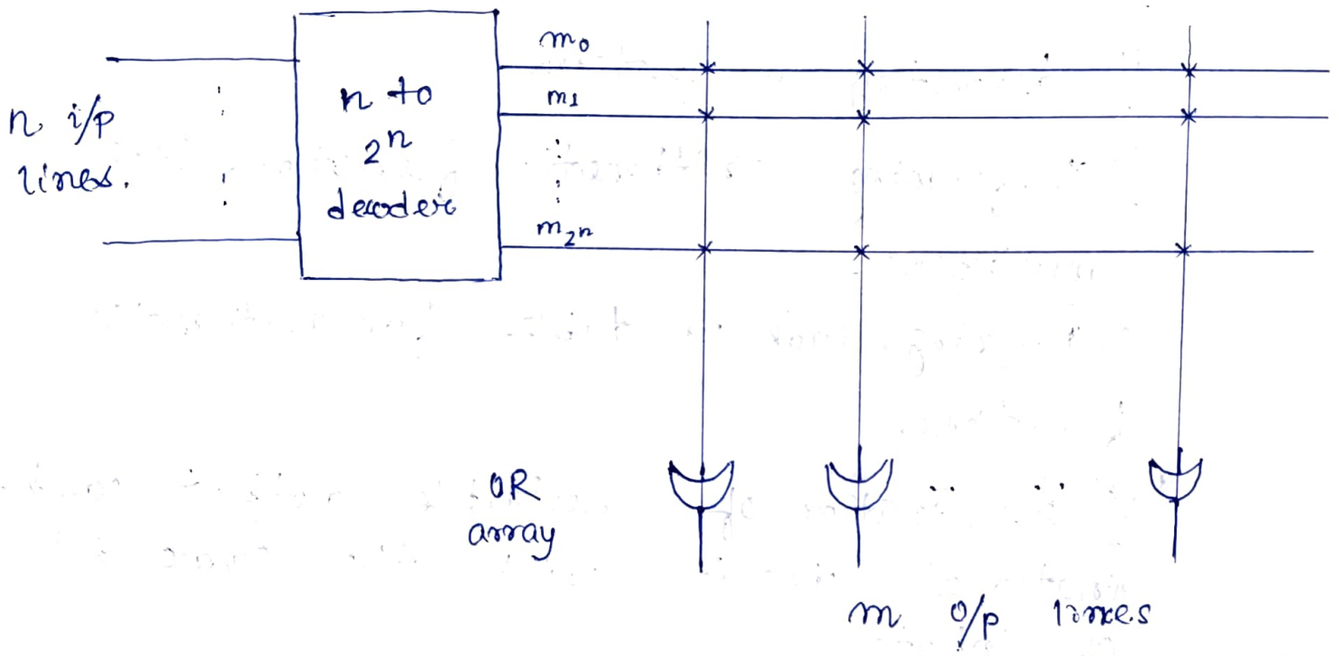
The address value applied to the input lines specifies the word at output lines at any given time.

→ A ROM is characterised by the number of words 2^n & number of bits per word m & defined as $2^n \times m$ ROM.

→ A $2^n \times m$ ROM. ***

Unit consists of 2^n words of m bits each. There are n input lines that form the binary numbers for 0 to $2^n - 1$ for the address. The n i/p s are decoded into 2^n distinct o/p s by means of a $n \times 2^n$ decoder. Each output of the decoder ~~is connected to one of the~~ represents a

memory address. The 2^n outputs are connected to each of the m OR gates. Each OR gate must be considered as having 2^n inputs. Since each OR gate has 2^n input connections & there are m OR gates, the PROM contains $2^n \times m$ internal connections which are programmable.



→ A programmable connection between two lines is logically equivalent to a switch that can be either open or closed. Programmable intersection is called a crosspoint. Various physical devices are used to implement crosspoint switches. One technology employs a fuse that normally connects the two points, but is opened or 'blown' by the application of a high-voltage pulse into the fuse.

→ The hardware procedure that programs the ROM blows fuse links in accordance with a given truth table consisting of inputs and output addresses.

I_{n-1} I_{n-2} ... I_0	O_{m-1} O_{m-2} ... O_1 O_0
0 0 0	1 0 ... 1 0
0 0 1	0 1 ... 0 1
⋮	⋮

Every 0 listed in the truth table specifies the absence of a connection, and every 1 listed specifies a path that is obtained by a connection.

→ ROM applications:

i) Realization of complex combinational circuits

ii) Code conversion

iii) Generating bit patterns

iv) Performing arithmetic functions like multipliers

v) Forming look-up tables for arithmetic functions.

* vi) Realization of multiple output combinational circuits with the same set of inputs.

→ Implementation of Combinational Logic Circuits.

Example - 1. Develop the boolean functions using ROM.

$$F_1(A, B, C) = \sum_m(0, 1, 2, 5, 7)$$

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

① Since there are 3 input variables, a ROM containing 3 to 8 decoder is needed. Since, there are two o/p functions, the OR array must contain at least 2 OR gates. That means, a $2^3 \times 2$ ROM is needed.

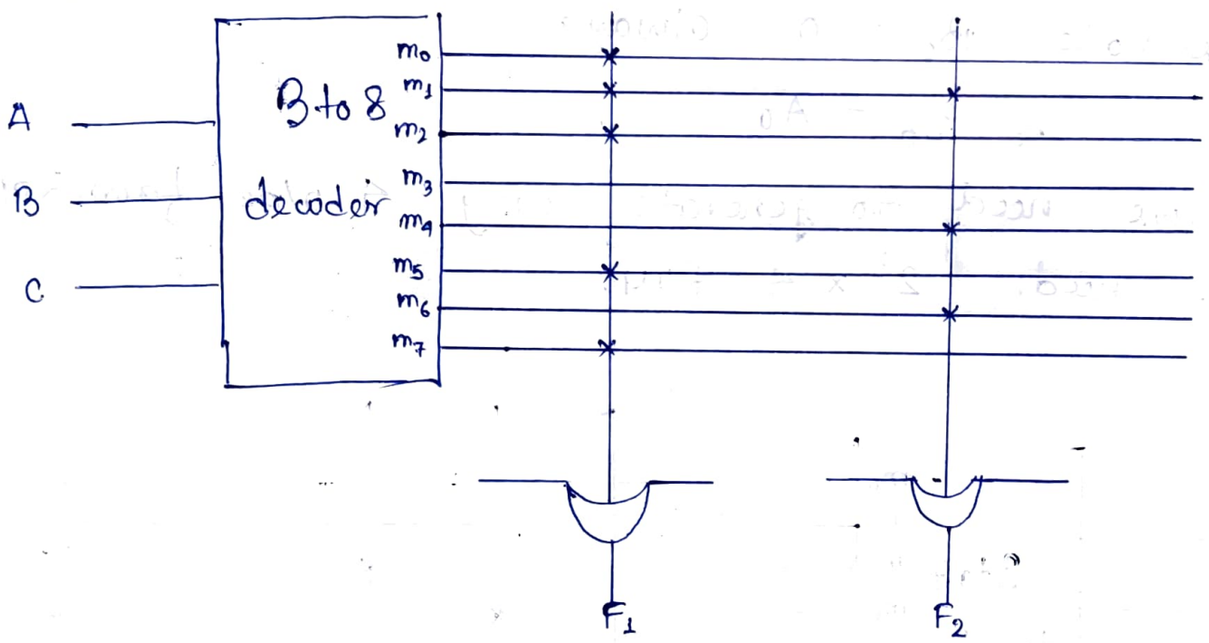
(2) Truth Table.

Decimal Equivalent	I/P variables A	B	C	Outputs F_1	F_2
0	0	0	0	1	0
1	0	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	0	1
7	1	1	1	1	0.

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

$F_2 = \bar{A}\bar{B}C + A\bar{B}\bar{C} + AB\bar{C}$

(4)



Example 2. Design a combinational circuit using ROM and the circuit accepts a 3-bit number & outputs a binary number equal to the square of the input number.

Truth Table –

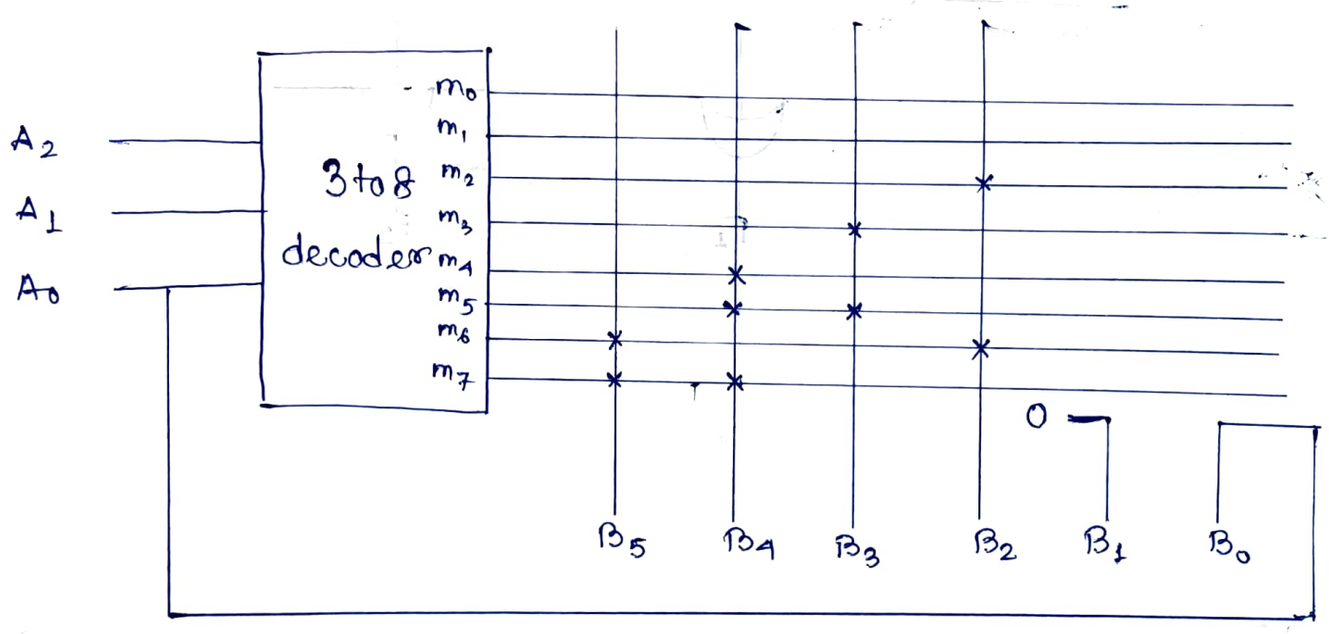
	Inputs			Outputs						Decimal
	A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	1
2	0	1	0	0	0	0	1	0	0	4
3	0	1	1	0	0	1	0	0	1	9
4	1	0	0	0	1	0	0	0	0	16
5	1	0	1	0	1	1	0	0	1	25
6	1	1	0	1	0	0	1	0	0	36
7	1	1	1	1	1	0	0	0	1	49

We note, $B_2 = 0$ always

& $B_0 = A_0$.

So, we need to generate only 4 o/ps from ROM.

We need. $2^3 \times 4$ ROM.



→ Types of ROM.

1. PROM – Manufacturer provides the PROM with all 0s or all 1s in every bit of stored words. Required links are broken by application of current pulses.

2. Erasable PROM – Can reconstruct the initial bit patterns of all 0s or all 1s, though it is already programmed for some bit configuration.

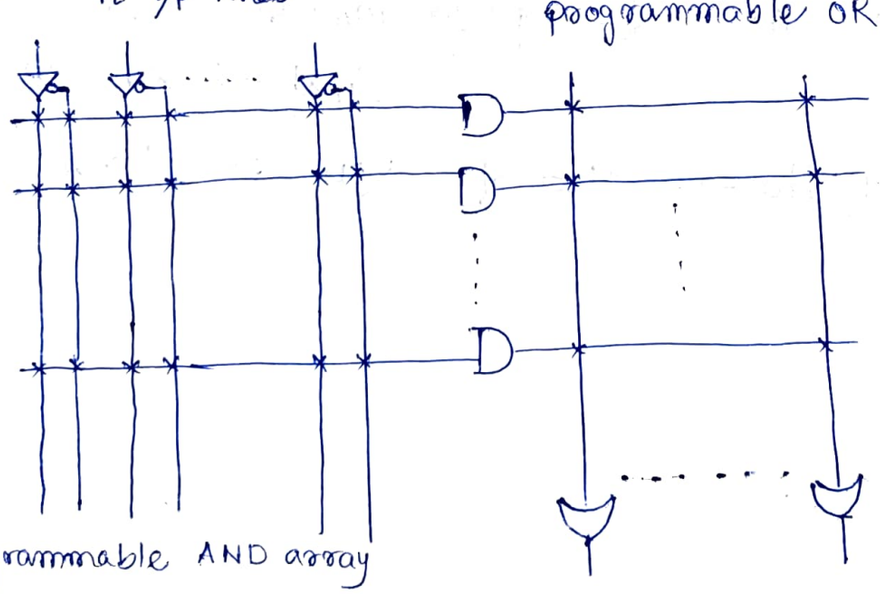
3. Electrically Erasable PROM (EEPROM).

Bit patterns are reset to their original state of all 0s or all 1s by applying a special electrical signal.

* PLA : PLA is a fixed architecture logic device with programmable AND gates followed by programmable OR gates.

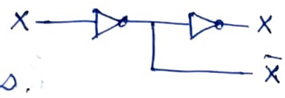
Advantage of PLA over ROM is, it does not provide all minterms, thus wastage of memory due to don't care addresses does not occur.

Disadvantage of PLA architecture has two sets of programmable fuses due to which PLA devices are difficult to manufacture, program & test.



→ Implementing combinational logic circuits with PLA

1. Calculate no. of i/p buffers ( ) from the number of variables.



No. of i/p buffers = no. of variables.

2. Find out the minimal SOP form of function from the truth table.

3. No. of programmable AND gates is equal to the number of unique minterms from the SOP expressions of the functions.

4. No. of programmable OR gates is equal to the number of function outputs.

→ Size of a PLA is specified by the number of inputs, the number of product terms & the number of outputs.

No. of i/p lines = n

No. of minterms = p [product terms]

No. of o/p lines = m.

Then the no. of programmable links =

$$2n \times p + p \times m + m.$$

(Whereas that of ROM is $2^n \times m$

→ **Example 1.** Implement combinational logic circuits for the following expressions —

$F_1(A,B,C) = \Sigma(0,1,3,4)$ and

$F_2(A,B,C) = \Sigma(1,2,3,4,5)$.

(1)

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

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

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

$F_2 = \bar{A}B + A\bar{B} + \bar{A}C$.

There are 4 distinct product terms.
$(\bar{B}\bar{C}, \bar{A}C, \bar{A}B, A\bar{B})$

(2) In the PLA device we need 3 i/p lines as there are 3 variables.

(3) We need 4 programmable AND gates.

(4) As there are 2 output functions, we need 2 o/p lines.

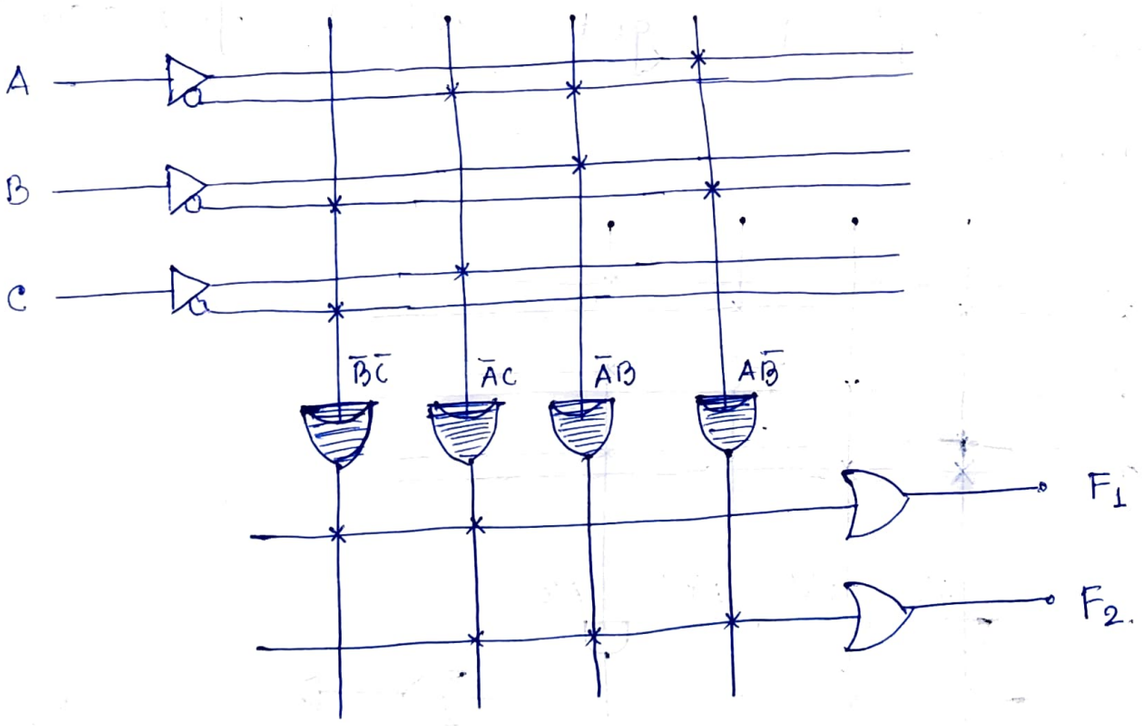


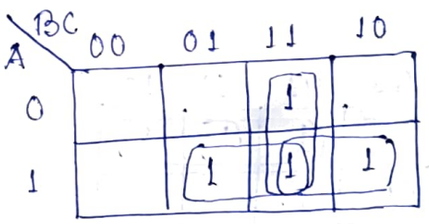
Diagram.

Example 2. Implement the boolean functions using PLA (3X4X

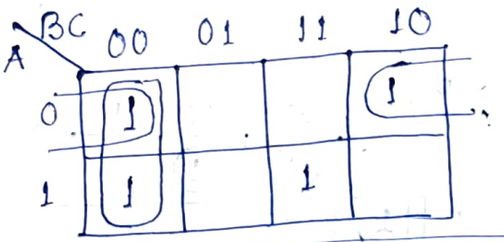
$F_1(A, B, C) = \Sigma(3, 5, 6, 7)$

$F_2(A, B, C) = \Sigma(0, 2, 4, 7)$.

(1)



$F_1 = AC + BC + AB$



$F_2 = B'C' + A'C' + ABC$

6 distinct minterms. So, we need 6 programmable AND gates.

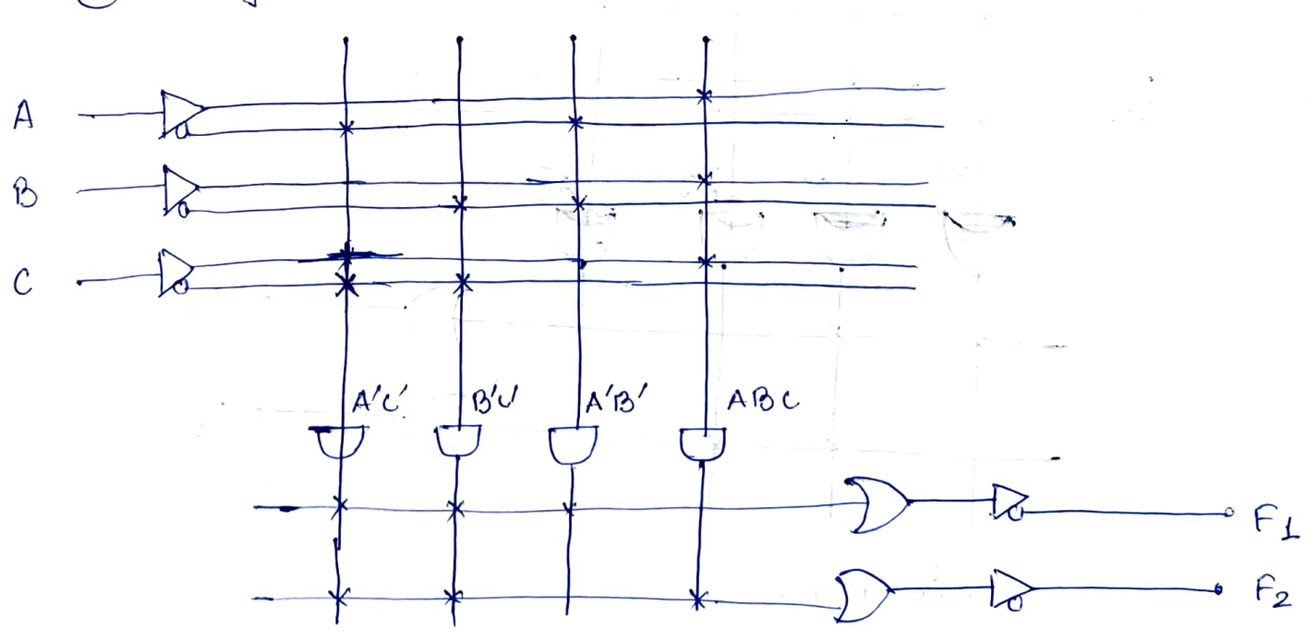
Now,

$F_1' = B'C' + A'C' + A'B'$ | $F_2' = A'C + B'C + ABC'$

If we select F_1' & F_2 as implementable SOP, we have 4 unique minterms.

So, we should select F_1' & F_2 for implementation by 3X4X2 PLA. We can complement F_1' to get actual function F_1.

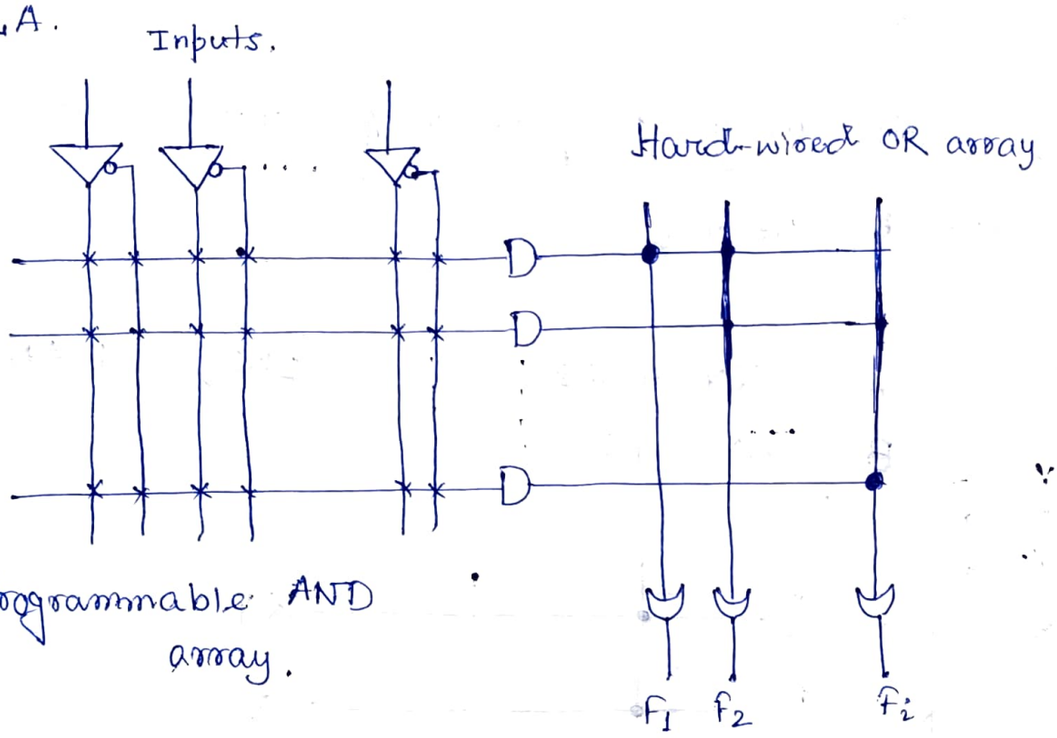
(2) Diagram —



* **PAL** : PAL is, a programmable logic device that has that has programmable AND array & fixed OR Array.

Advantage of PAL is that we can generate only the required product terms of Boolean function instead of generating all the minterms by using programmable AND gates.

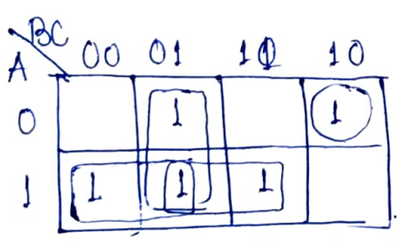
Disadvantage of PAL is that only the AND array is programmable & thus not flexible as compared to PLA.



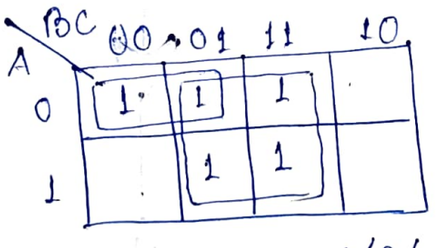
Example 1. Realize following functions using PAL

$F_1(A, B, C) = \Sigma(1, 2, 4, 5, 7)$

$F_2(A, B, C) = \Sigma(0, 1, 3, 5, 7)$

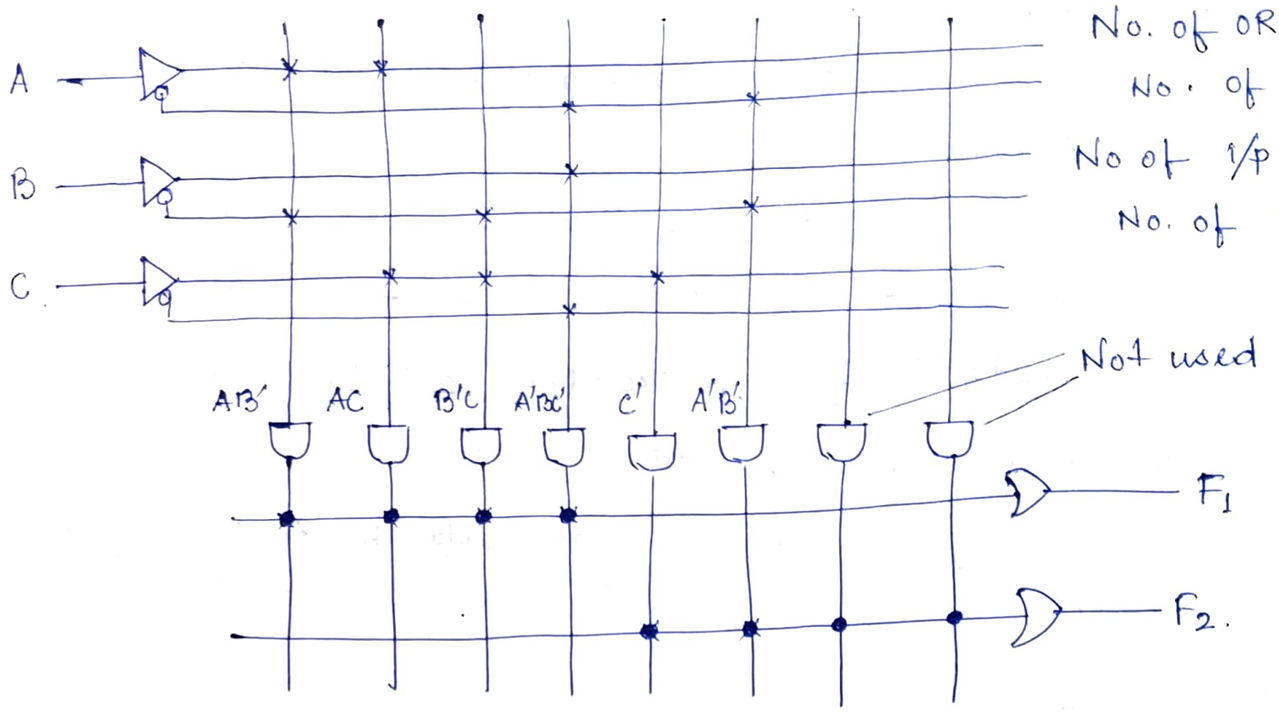


$F_1 = AB' + AC + B'C + A'BC'$



$F_2 = C + A'B'$

We will need 4X2 AND gates (4 for max no. of minterms in any function output & 2 for no. of functions).



No. of OR gates =
No. of fun^{n}s
No of i/p lines =
No. of variables

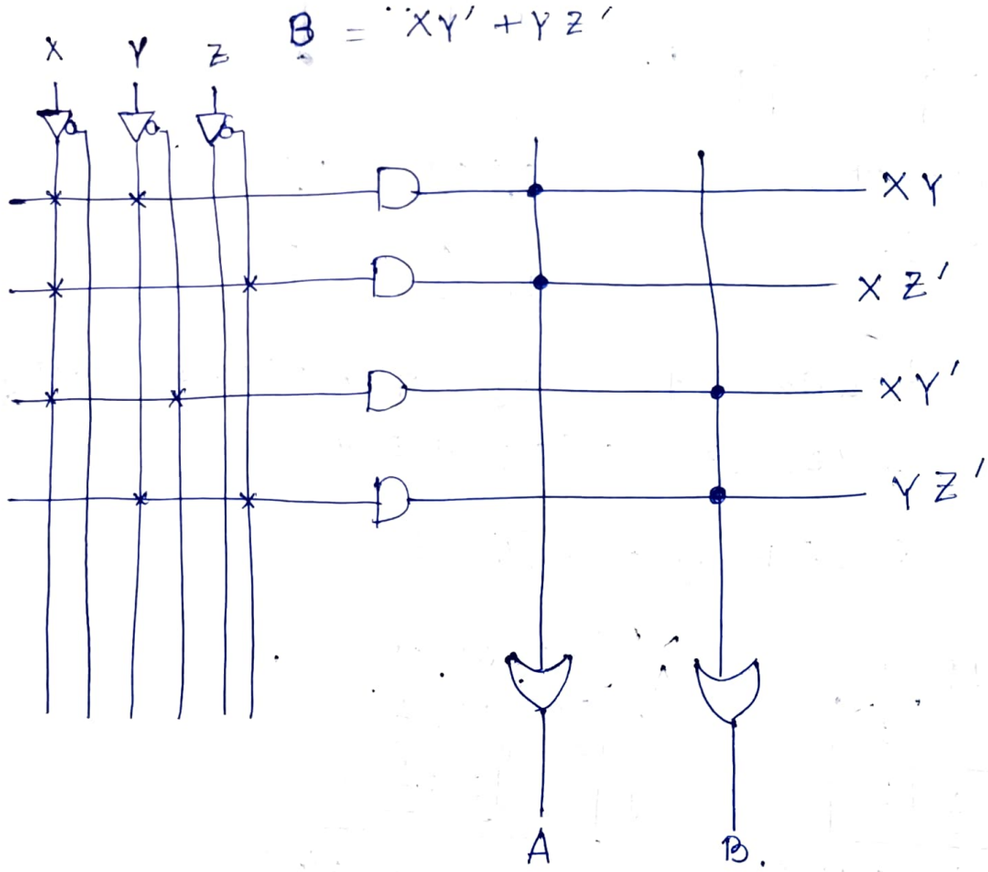
PAL generate only required product terms as needed by user.

Example 2.

$A = XY + XZ'$

$B = XY' + YZ'$

$(2 \times 2) = 4$ AND gates



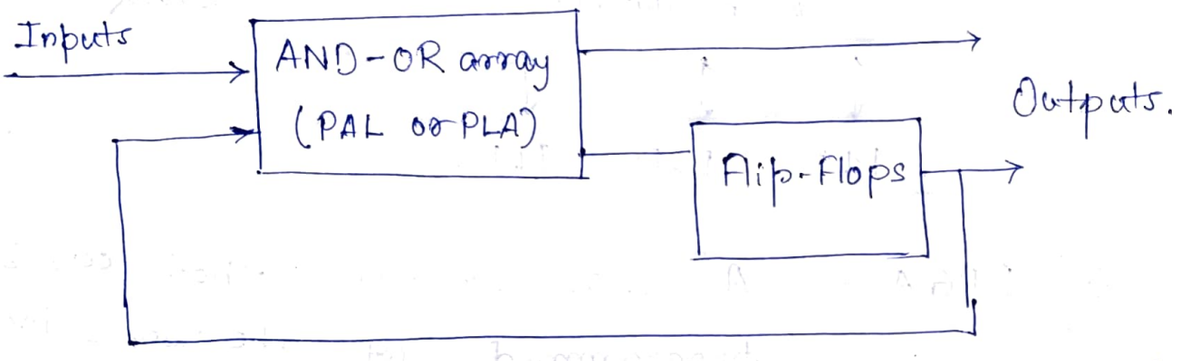
* Sequential Programmable Devices.

Include both gates and flip-flops and they can be programmed to perform a variety of sequential-circuit functions.

→ Major Types –

i) Sequential (or Simple) Programmable Logic Device (SPLD).

Includes flip-flops, in addition to AND-OR array, within the integrated circuit chip.



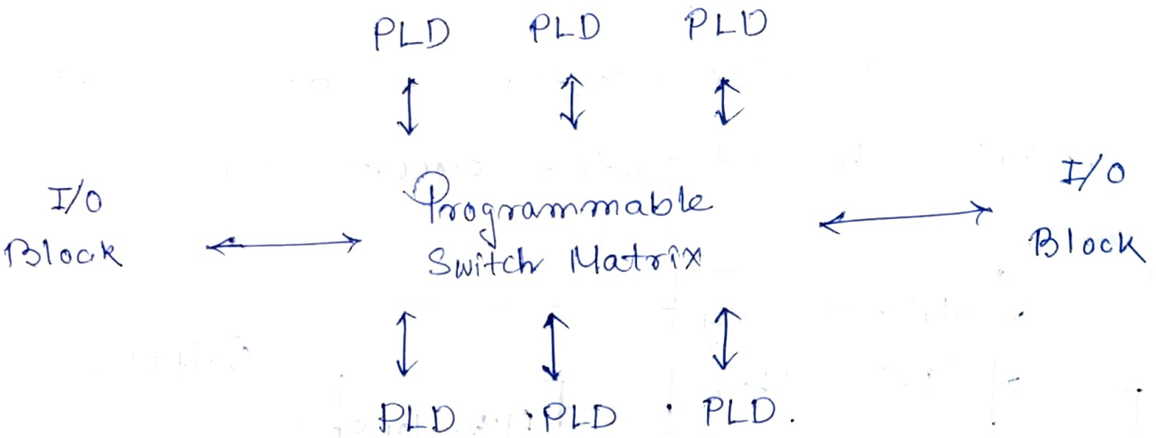
The configuration mostly used in an SPLD is the combinational PAL together with D FFs.

A PAL that includes FFs is referred to as a registered PAL.

Each section of an SPLD is called a macrocell, which is a circuit that contains a sum-of-products combinational logic function & an optional FF.

ii) Complex Programmable Logic Devices (CPLD)

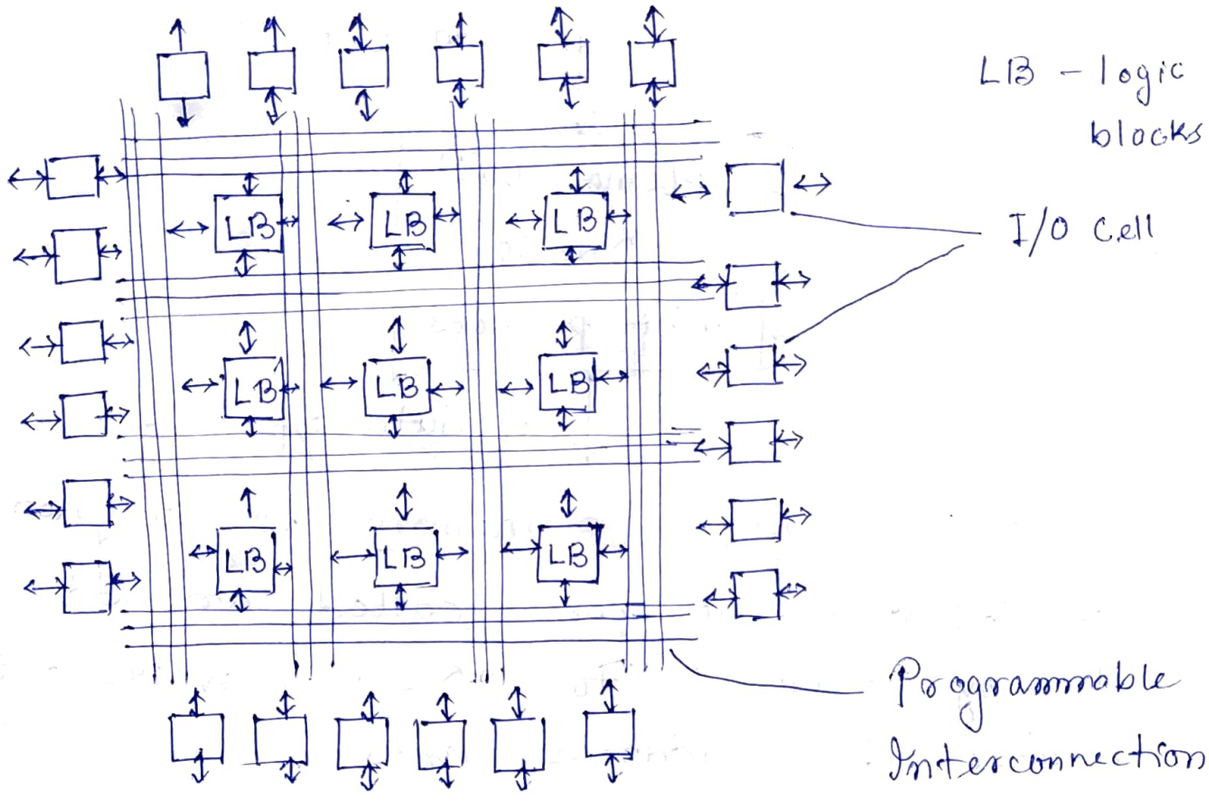
– Collection of individual PLDs on a single integrated circuit. The device consists of multiple PLDs interconnected through a programmable switch matrix. The switch matrix receives input from the I/O block & directs them to the individual macrocells.



iii) FPGA – A VLSI circuit that can be programmed at the user's location

A typical FPGA consists of an array of millions of logic blocks, surrounded by programmable i/o blocks & connected together via programmable interconnections. A typical FPGA logic block consists lookup tables, multiplexers, gates & FFs. A lookup table is a truth table stored in an SRAM & provides the combinational circuit functions for the logic block. The combinational logic section, along with a programmable multiplexers, is used to configure the input equations for the FF & the output of the logic block. The program can be downloaded either from a host computer

or from an onboard PROM. The program remains in SRAM until the FPGA is reprogrammed or the power is turned off. The device must be reprogrammed every time power is turned on.



FPGA Design.

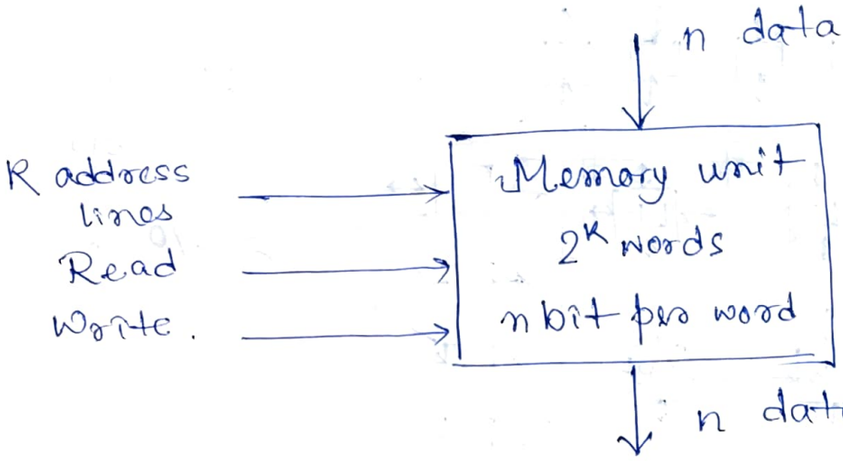
* Random Access Memory (RAM).

→ Internal memory of the CPU for storing data, program & program result. It's read/write memory which stores data until the machine is working. As soon as the machine is switched off, data is erased.

→ Access time in RAM is independent of the address. RAM is volatile.

→ A memory unit stores binary information in groups of bits called words. A memory word is a group of 1's & 0's & may represent a number, an instruction, one or more alphanumeric characters or any other binary coded information.

→ The communication between a memory & its environment is achieved through data input & output lines, address selection lines, & control lines that specify the direction of transfer.



→ Each word in a memory is assigned an identification number, called an address, starting from 0 to $2^K - 1$, where K is the number of address lines.

→ Number of words in a memory with one of the letters —

$K = 2^{10}$	$64K = 2^{16}$
$M = 2^{20}$	$2M = 2^{21}$
$G = 2^{30}$	$4G = 2^{32}$

→ Transferring a new word to be stored onto memory (write) —

1. Apply the binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the data input lines.
3. Activate the write input.

→ Transferring a stored word out of memory (read):

1. Apply the binary address of the desired word to the address lines.
2. Activate the read input.

→ Two types of RAM —

1. SRAM - Static RAM. - The word static indicates that the memory retains its contents as long as power is being supplied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6 transistors & no capacitors. Transistors do not require power to prevent leakage, so SRAM need not be refreshed on a regular basis. SRAM uses more chips than DRAM for same amount of storage, making manufacturing costs higher. SRAM is needed as cache memory & has very fast access.

SRAM is easier to use & has shorter read & write cycles.

Low density, low capacity, high cost, high speed, high power consumption.

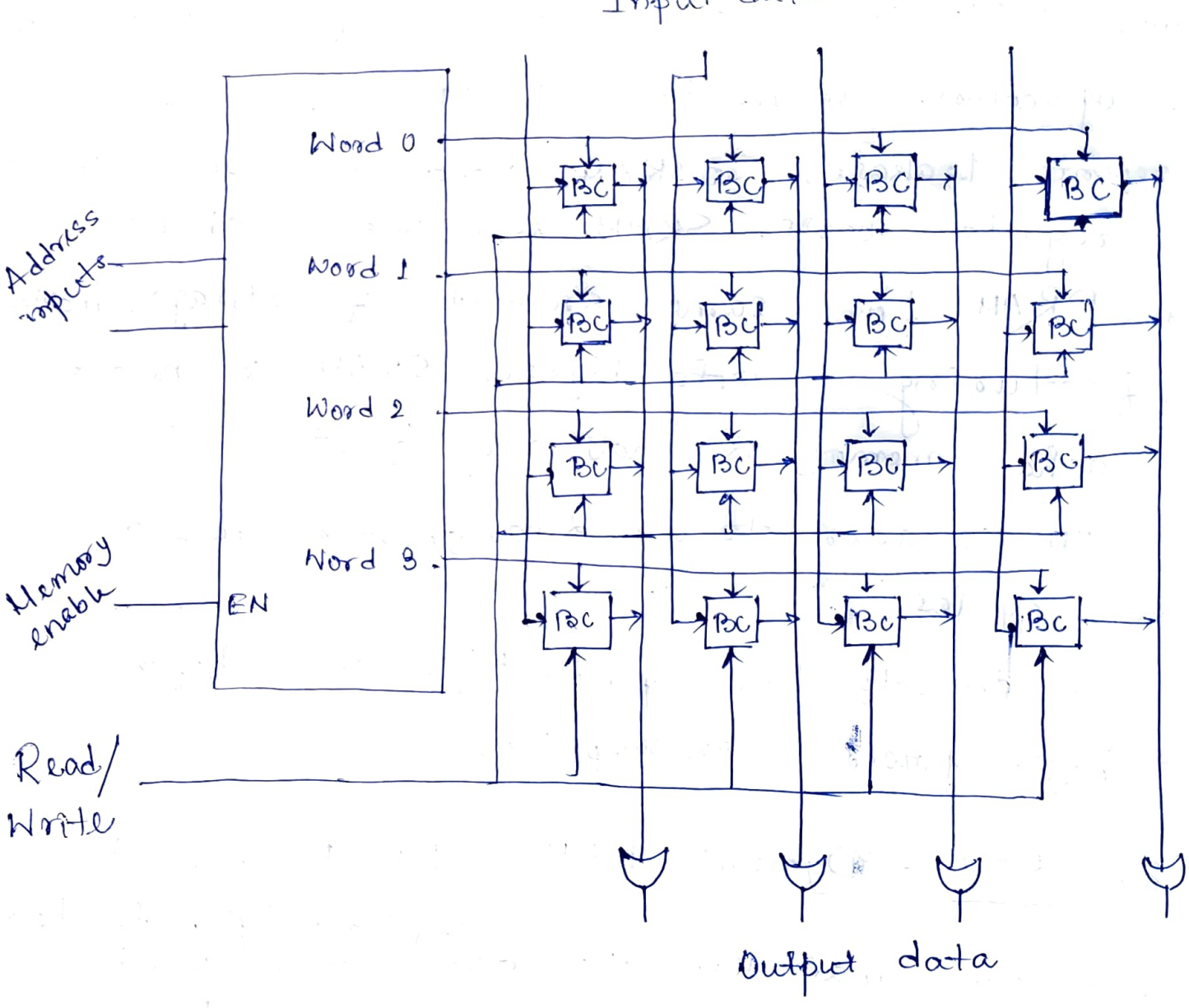
2. DRAM - Dynamic RAM - DRAM must be continually refreshed in order to maintain the data. It is used for most system memory as it is cheap & small. All DRAMs are made up of memory cells that are composed of one capacitor & one transistor. DRAM stores the binary information in the form of electric charges on capacitors. The capacitors tend to discharge with time &

must be periodically recharged by refreshing the dynamic memory.

High density, high capacity, low cost, low speed, low power consumption.

→ Diagram of a 4×4 RAM –

A memory with 2^k words of n bits per word requires k address lines that go into k × 2^k decoder.



Data Converters.

TP

* The system that realises the conversion of analog signal to digital signal is referred to as an analog to digital converter or A/D converter or ADC

→ Classification -

i) Counter type ADC ii) Successive approximation type iii) Flash or Parallel iv) Dual Slope or Integrating type v) Sigma-Delta ADC (Σ-Δ ADC)

* The system used for the digital to analog conversion of signal is called digital to analog converter or DAC or D/A converter

→ Classification:

i) Binary Weighted Resistor DAC.

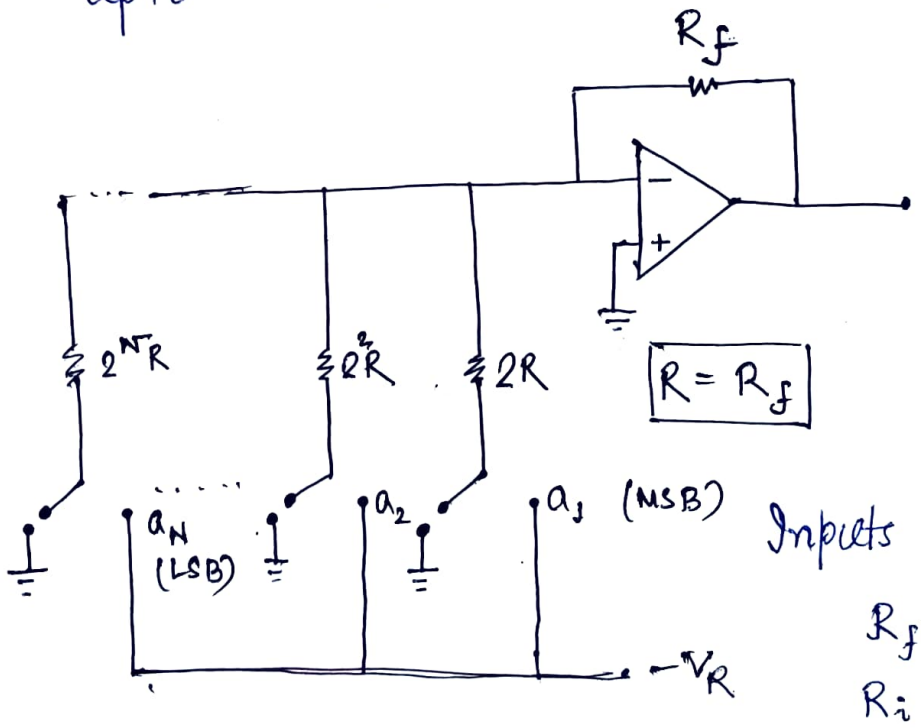
ii) R-2R Ladder DAC.

* DAC.

- Binary Weighted Resistor DAC.

To convert n bit data we need upto $2^n R$.

Binary weights ... 2^2 2^1 2^0



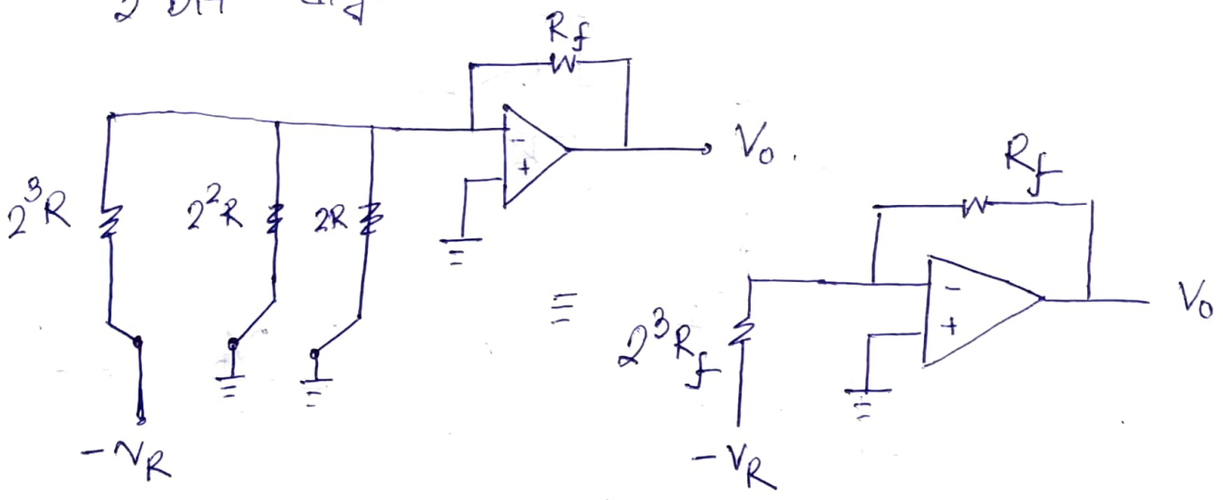
Inputs (dig.)

R_f → feedback resistance

R_i → i/p resistance.

Disadvantage - No. of resistances (high)

eg. 3 bit digital data – 001



$$V_o = -\frac{R_f}{R_i} V_i = -\frac{R_f}{2^3 R_f}(-V_R) = \frac{V_R}{2^3}$$

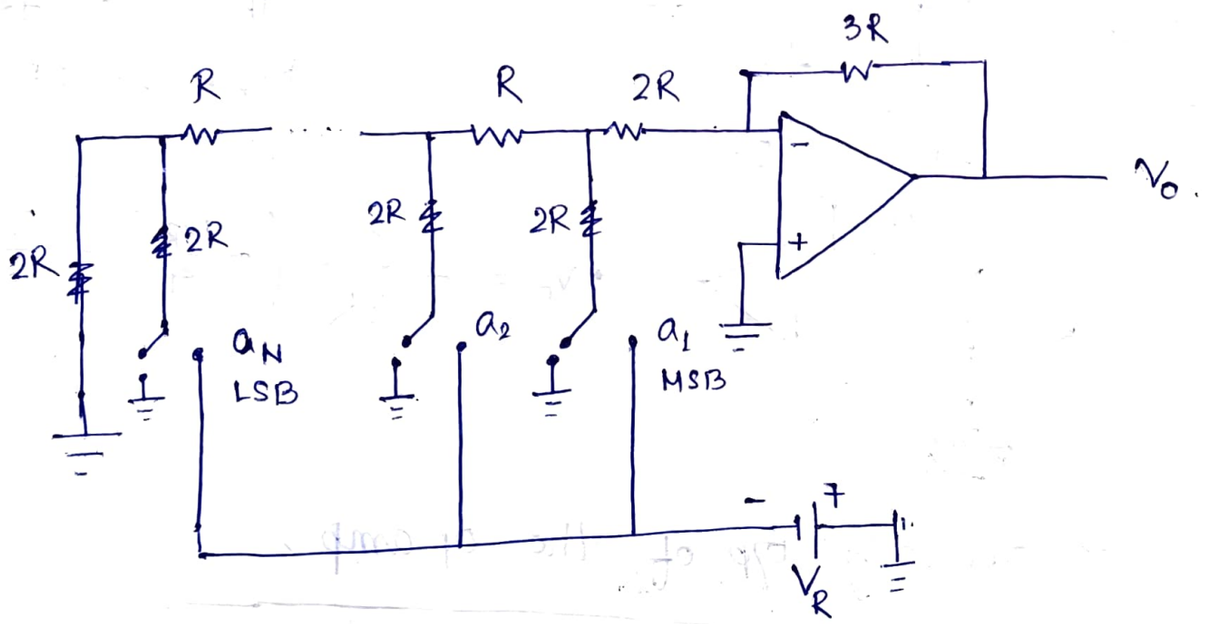
→ Generally, o/p of the OP Amp, V_o =

$$V_o = \left[\frac{a_1}{2} + \frac{a_2}{2^2} + \cdots + \frac{a_N}{2^N}\right] V_R$$

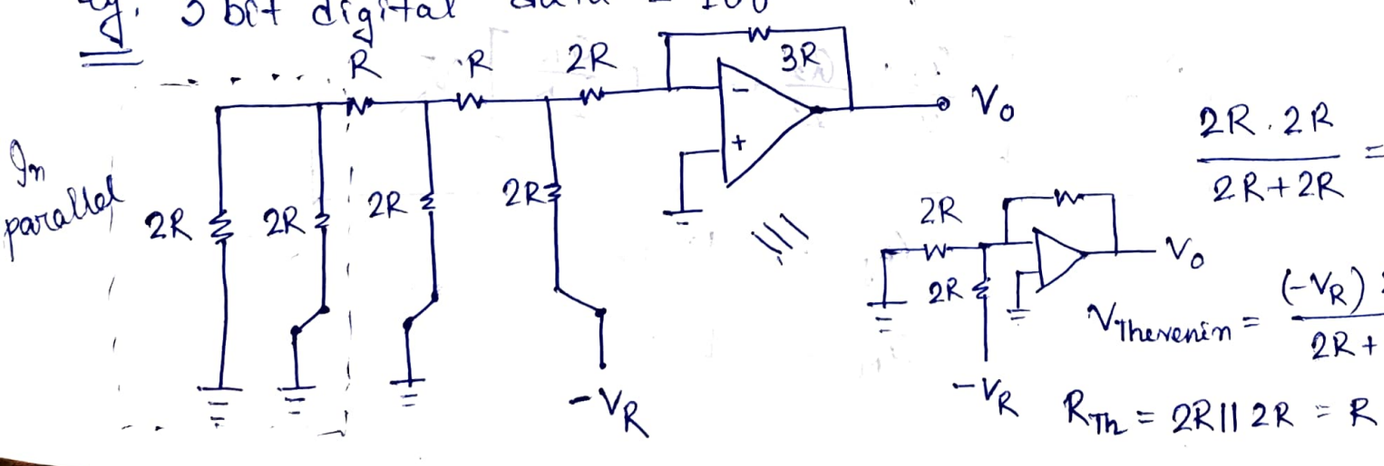
(MSB: a_1, LSB: a_N)

where, a_1 is MSB and a_N LSB.

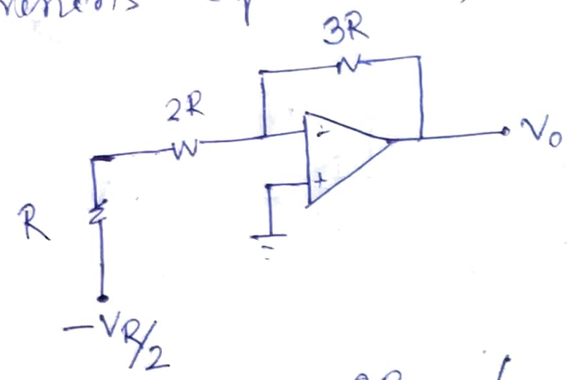
- R-2R Ladder DAC. (Voltage Switched).



eg. 3 bit digital data – 100

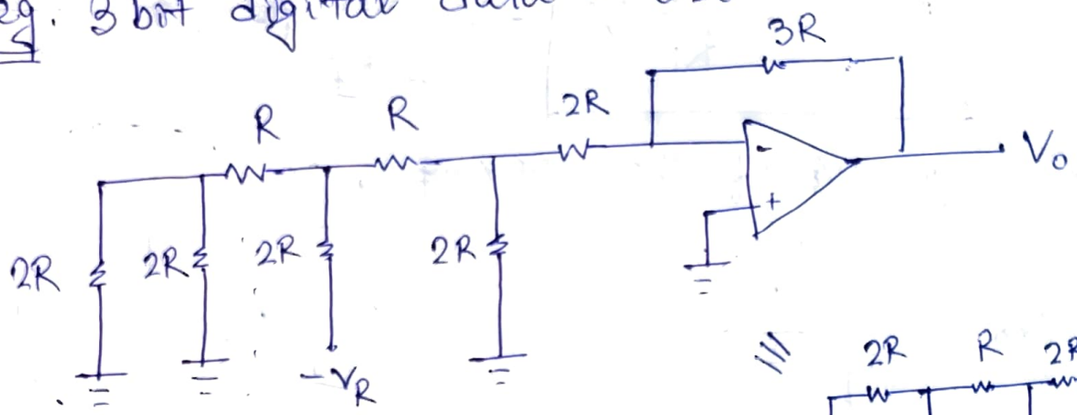


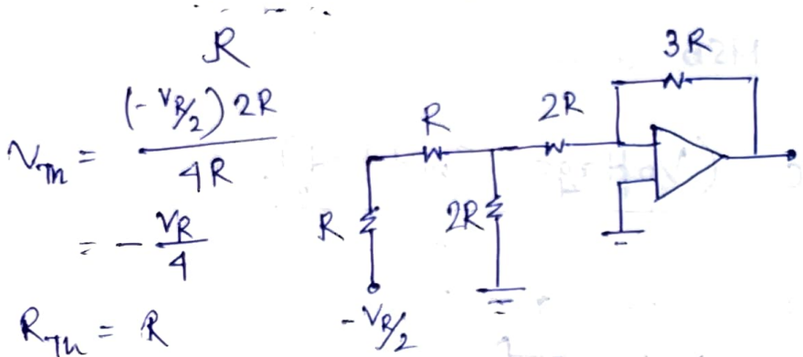
Considering Thevenin's Equivalents,



$$V_o = -\frac{R_f}{R_i} V_i = -\frac{3R}{3R}\left(-\frac{V_R}{2}\right) = \frac{V_R}{2} \quad \text{(Ans)}$$

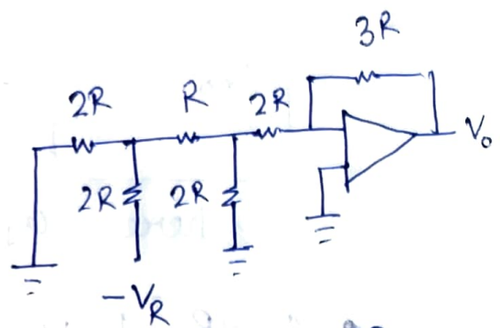
eg. 3 bit digital data - 010





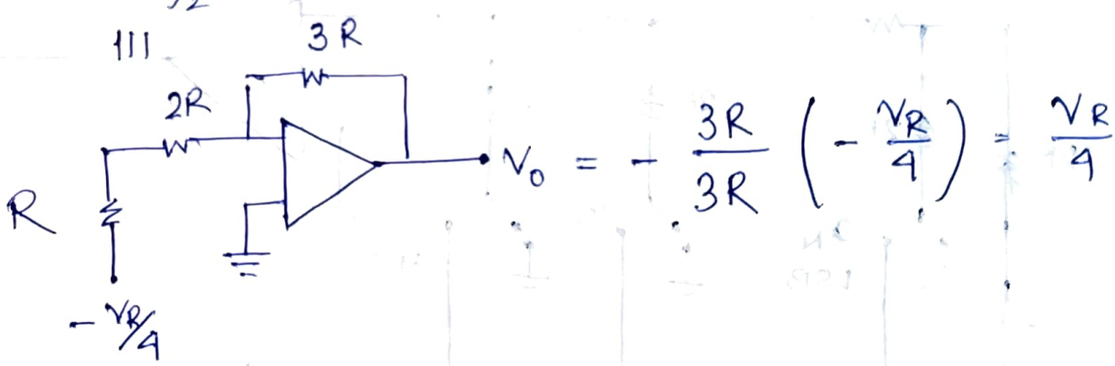
$$V_{Th} = \frac{(-V_R/2)2R}{4R} = -\frac{V_R}{4}$$

$$R_{Th} = R$$



$$V_{Th} = (-V_R)\frac{2R}{4R} = -\frac{V_R}{2}$$

$$R_{Th} = 2R \| 2R = R$$



$$V_o = -\frac{3R}{3R}\left(-\frac{V_R}{4}\right) = \frac{V_R}{4}$$

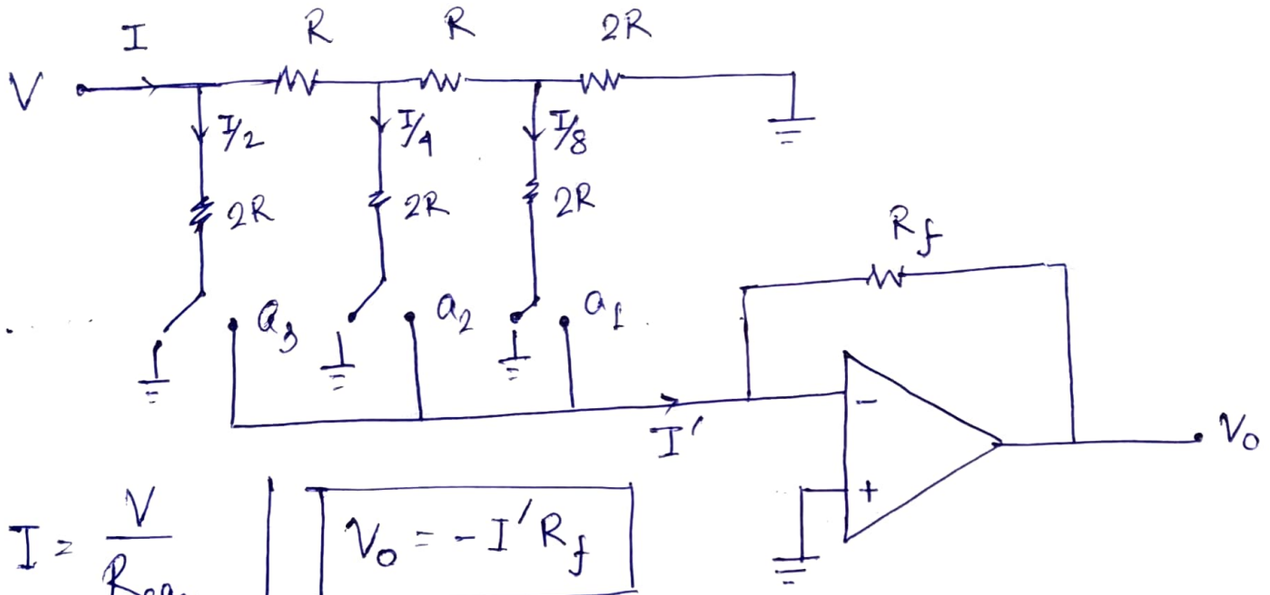
→ Generally o/p of the op amp,

$$V_o = \left[\frac{a_1}{2} + \frac{a_2}{2^2} + \dots + \frac{a_N}{2^N}\right] V_R$$

$a_1 \rightarrow$ MSB

$a_N \rightarrow$ LSB.

• R-2R Ladder DAC (Current Switched).



$$I = \frac{V}{R_{eq.}}$$

$$V_0 = -I' R_f$$

$$I' = \left[\frac{I}{2} + \frac{I}{4} + \frac{I}{8} \right]$$

For n-bit,

$$V_0 = -\left[\frac{a_N}{2} + \frac{a_{N-1}}{4} + \dots + \frac{a_1}{2^N} \right] I R_f .$$

Eg. $V = 5\,V$, $R = 2\,k\Omega$, $R_f = 2\,k\Omega$, $a_1 = 1$, $a_2 = a_3 = 0$.

$$I = \frac{V}{R} = \frac{5\,V}{2\,k\Omega} = 2.5\,mA$$

$$I' = \frac{1}{2^3} 2.5\,mA = \frac{2.5}{8}\,mA$$

$$V_0 = -\left(\frac{2.5}{8}\,mA\right) \cdot 2\,k\Omega = -0.5125\,V. \quad (Ans.)$$

* Different families of logic gates -

i) Diode logic (DL) - uses diode to AND, OR.

ii) Resistor - transistor logic (RTL)

iii) Diode - transistor logic (DTL)

iv) Transistor - transistor logic (TTL)

v) Emitted - Coupled logic (ECL) / Current Mode logic (CML)

vi) Complementary Metal Oxide Semiconductor (CMOS) logic.

vii) NMOS (N-type Metal oxide semiconductor logic)

* Fan In - The number of standard loads drawn by an input to ensure reliable operation.

Fan Out - The number of standard loads that can be reliably driven by an output, without causing the o/p voltage to shift out of its legal range of values.