# Digital logic

## Advantages of digital logic :

- Digital data can be processed and transmitted more efficiently & reliably than analog data
- Effect of noise is less
- Data Compression is possible in digital
- Digital data can be Encrypted ie security is more
- Speed of operation is high
- Power Consumption is less

## Number system

**Binary $(\ )_2$**
- 0, 1
- Ex:- $(10.11)_2$
- $1*2^1+0*2^0+1*2^{-1}+1*2^{-2}$

**Octal $(\ )_8$**
- 0 to 7
- Ex:- $(3.71)_8$
- $3*8^0+7*8^{-1}+1*8^{-2}$

**Decimal $(\ )_{10}$**
- 0 to 9
- Ex:- $(22.3)_{10}$
- $2*10^1+2*10^0+3*10^{-1}$

**Hexadecimal $(\ )_{16}$**
- 0 to 9, A to F
- Ex:- $(A4.F)_{16}$
- $A*16^1+4*16^0+F*16^{-1}$

## Conversions :

① B → O : 3 places
B → D : $2^0, 2^1 \ldots$
B → H : 4 places

② O → B : replace 3 digit binary num
O → D : $8^0, 8^1 \ldots$
O → H : O → B → H (replace 4 places)

③ D → B : 2 ⌊
D → O : 8 ⌊
D → H : 16 ⌊
Bottom up approach

④ H → B : replace 4 bit bin num for each digit
H → O : H → B → O (rep, 3 places)
H → D : $16^0, 16^1 \ldots$

**Note :**
- Minimum digit in any number system is 0
- Maximum digit in any number system is $r-1$ ($r$→radix)

$(163.875)_{10} \rightarrow (\ )_2$

163 ⇒ 10100011

.875 × 2 ⇒ 1.75 → 1
.75 × 2 ⇒ 1.5 → 1
.5 × 2 → 1.0 → 1

$(10100011.111)_2$

$(673.23)_{10} \rightarrow (\ )_8$

8 | 673
8 | 84 − 1
8 | 10 − 4
1 − 2

1241

.23 * 8 ⇒ 1.84 → 1
.84 * 8 ⇒ 6.72 ⇒ 6
.72 * 8 ⇒ 5.76 → 5
.76 * 8 ⇒ 6.08 ⇒ 6
.08 * 8 ⇒ 0.64 → 0
.64 * 8 ⇒ 5.12 → 5

165605 ...

$\therefore (1241.165605\ldots)_8$

Num of bits reqd to represent 32 digit decimal num in binary is 107 bits.

32 digit max → $10^{32}-1$

$2^n >$ Number

$2^n > 10^{32}-1$

$n \log_{10}^2 > 32 \log_{10}^{10}$

$n > \frac{32}{0.3010} \Rightarrow n > 106.3$

$n = 107$ bit

Num of digits reqd to rep 126 bit bin num 38 digit

126 bits → $2^{126}-1$

$10^n - 1 \geq 2^{126} - 1$

$10^n > 2^{126}$

$n > 126 \log_{10}^2$

$n > 126(0.3010)$

$n > 37.93$

38 digit

$*(CE)_{16} + (A2)_{16}$

1
CE
A2
170

$(10)_{16}$

$16 \rightarrow 1*16^1+0*16^0$

$23 \rightarrow 1*16+7*15^0$

$(17)_{16}$

$(2BC)_{16} - (1DE)_{16}$

2 B C
1 D E
0 D E

(16) (16) 28
1 13 14
0 13 14
⇓ ⇓
b E

## Compliments :

- reduce hardware Complexity
- Subtraction can be done using adders
- It affects only negative numbers but not on +ve numbers

### Types of Compliments :

**Diminished / $r-1$'s Comp**
For : $r^n - r^{-m} - N$

**Radix / $r$'s comp**
For : $r^n - N$

N : Given num, n - num of non-fractional bits, m - num of fractional bits

*) $(Binary)_2$ → 1's Comp, 2's Comp | *) $(octal)_8$ → 7's Comp, 8's Comp | *) $(Decimal)_{10}$ → 9's Comp, 10's Comp | *) $(Hexa)_{16}$ → 15's Comp, 16's Comp

* **r's Comp = r-1's Comp + 1**

**Note : General method**

For r-1's Comp

| $r^{-1}$ | $r^{-1}$ | $r^{-1}$ | $r^{-1}$ |
|---|---|---|---|
| A | B | C | D = |
| ⊖ | | | |

For r's Comp drop

| $r^{-1}$ | $r^{-1}$ | $r^{-1}$ | r |
|---|---|---|---|
| A | B | C | (D=1) 0 |
| ⊖ | | | 0 |

9's Comp of $(30670)_{10}$

9 9 9 9 9
3 0 6 7 0
$(69329)_{10}$

9's Comp of $(3670)_9$

8 8 8 9
3 0 6 7 0
$(58220)_9$

## Signed numbers

### Sign-magnitude method:

+5 : Sign 5 → 0101
−5 : 1101 (5)

→ 2 rep for zero
∴ +0(0000), −0(1000)

### Signed 1's Comp method:

+5 : 0101
−5 : 1010 (1's Comp)

→ 2 rep's for zero
∴ +0(0000) −0(1111)

### Signed 2's Comp method:

+5 : 0101
−5 : 1011 (2's Comp)

→ 2 time's add^n not req^d
→ more preferable bcoz unique rep of zero ☺

**Note :**

| Method | Range |
|---|---|
| sign-magnitude | $-(2^{n-1}-1)$ to $2^{n-1}-1$ |
| 1's Comp | $-(2^{n-1}-1)$ to $2^{n-1}-1$ |
| 2's Comp | $-2^{n-1}$ to $2^{n-1}-1$ |

*) 2's Comp(x) = x → 1000 0000
1000 / 10000
10

*) Same value for 2's Comp & Sign & mag
1100 0000 / 1100 / 11000

### Sign Extension :

For +ve numbers placing and removing 0's at MSB doesn't affect similarly
For −ve numbers placing & removing is at MSB doesn't affect

Ex:- −16 → (2's comp) → 10000 , 11110000 (−128 64 32 16) → −128+64+32+16 → −128+112 ⇒ (−16)

Ex:- 2's Comp rep(P) → $[F87B]_{16}$ then 2's Comp of P →

1111 [1000 0111 1011] * 8 (1000) → 11000 0111 1011 1000
$[C\ 3\ D\ 8]_{16}$

## Binary Codes :

| Note : Binary Codes | Examples |
|---|---|
| Numeric | 8421, Ex-3, Gray, 2421, BCD |
| Alpha Numeric | ASCII, EBCDIC |
| Weighted | 8421, 2421, BCD |
| Non-weighted | Ex-3, Gray |
| (Sum=9) Self Compliment | 2421, 5211, 642-3, 8'4-2-1, Ex-3 |
| Not self | 5421, BCD |
| Cyclic Codes / Reflective | Gray, 2421 |

**Self Comp Code :**
In any code a number and it's 9's Comp are complement to each other then it is called self comp code.

**Excess-3 code :** BCD + 0011 → XS3
(Not self → self)

**Gray Code :**
- unit distance code.
- used in K-maps.

**Binary to Gray :**
1011 → B
1110 → G

**Gray to Binary :**
1 1 1 0 → G
1 0 0 1 → B

| XS-3 | Gray | 2421 | BCD |
|---|---|---|---|
| - Numeric | - Numeric | - Numeric | - Numeric |
| - self Comp | - Cyclic | - Cyclic | - Not self |
| - Non weighted | - Non weighted | - weighted | |
| | | → self Comp | |

Note :

| Decimal | binary | BCD | (self) XS-3 | (cyclic) Gray | (self reflective) 2421 |
|---|---|---|---|---|---|
| 0 | 0 | 0000 | 0011 | 0000 | 0000 |
| 1 | 1 | 0001 | 0100 | 0001 | 0001 |
| 2 | 10 | 0010 | 0101 | 0011 | 0010 |
| 3 | 11 | 0011 | 0110 | 0010 | 0011 |
| 4 | 100 | 0100 | 0111 | 0110 | 0100 |
| 5 | 101 | 0101 | 1000 | 0111 | 1011 |
| 6 | 110 | 0110 | 1001 | 0101 | 1100 |
| 7 | 111 | 0111 | 1010 | 0100 | 1101 |
| 8 | 1000 | 1000 | 1011 | 1100 | 1110 |
| 9 | 1001 | 1001 | 1100 | 1101 | 1111 |

**Points to remember :**

① Invalid states in XS-3 : 0000, 0001, 0010 : 1101, 1110, 1111

**Cyclic Codes :**
- Convinient for Error detection & correction
- Implemented using shift registers.

**Self Comp Codes :**
- Design of hlw becomes easy

**ASCII** (American standard Code for Info Interchange)
→ 7 bit Code, 1 bit parity
48-57 ⇒ 0-9 → 1100000 / 1001
65-90 ⇒ A-Z
97-122 ⇒ a-z

**EBCDIC** (Extended binary Coded decimal Interchange Code)
→ 8 bit Code

## Boolean Algebra :

**Purpose :** Complicated fns can be Converted into simple form

∵ Hardware Comp reduces
less num of gates reqd
less cost | More efficient
Small size | Power consumption is less

**Boolean laws :-**

① **Commutative law**
$A+B = B+A$
$AB = BA$

② **Associative law :**
$A+(B+C) = (A+B)+C$
$A\cdot(B\cdot C) = (A\cdot B)\cdot C$

③ **Distributive law :**
$A+(B\cdot C) = (A+B)\cdot(A+C)$
$A(B+C) = AB+AC$

④ **Demorgan's Law**
$(A+B)' = A'\cdot B'$
$(AB)' = A'+B'$

**Postulates of Boolean Algebra :**

① $A+0 = 0+A = A$
② $A+1 = 1+A = 1$
③ $A\cdot 0 = 0\cdot A = 0$
④ $A\cdot 1 = 1\cdot A = A$
⑤ $A+A = A$
⑥ $A\cdot A = A$
⑦ $A+A' = 1$
⑧ $A\cdot A' = 0$
⑨ $(A')' = A$
⑩ $A+AB = A$ (Reduct) ~~Dual~~
⑪ $A+\bar{A}B = A+B$ ~~Dual~~

**Literals :** Var + Comp (repetitions allowed)
Var : A, B, C ---
Comp : $\bar{A}, \bar{B}, \bar{C}$ ---
0, 1 ⇒ Not literals

**Note :**
- Num of Boolean fns possible with boolean variables
  2 (var) → 4 possibilities → $2^4$ Boolean fns
  $n \longrightarrow 2^n \longrightarrow 2^{2^n}$ ⇒ Boolean fn with n-var boolean algebra

n-any fn for n-any algebra for n-var
$n^{(var)} \longrightarrow n^{n\,(pos)} \longrightarrow n^{n^n}$ fns.
$n \leftrightarrow$ var
$n^n \rightarrow$ Algebra
$n^{n^n} \rightarrow$ functions

- Num of self dual fns with n var Boolean Algebra : $2^{2^{n-1}}$

- Boolean function of n-variables with 'p' min terms : ${}^{2^n}C_p$
- Boolean function of n-variables with 'p' max terms : $2^n - {}^{2^n}C_p$
- Num of Boolean functions with 0, 1, 2 --- $2^n$ minterms : ${}^{2^n}C_0 + {}^{2^n}C_1 + {}^{2^n}C_2 + \cdots + {}^{2^n}C_{2^n}$

**Duality theorem :** ($x \rightarrow x$)
- It will not affect variables
- $0 \rightleftarrows 1$
- AND ⇄ OR (·) (+)
- XOR ⇄ XNOR
- NAND ⇄ NOR

**Note :** If $f_D = f$ then f is called self dual function
Ex :- $AB+BC+CA$
- Num of minterms = Num of max terms
- There should not be any mutually exclusive terms.
  $f(x_1, x_2, \cdots x_n) = f(x_1', x_2' \cdots x_n')$

## Mutually Exclusive terms

$f(x_1, x_2, x_3 \cdots x_n) = f(x_1', x_2', x_3' \cdots x_n')$

Self Dual fns

Ex:- n=2 ME terms : (0,3)(1,2) ⇒ 2 terms ⇒ No. $2^2 = 4$

n=3 ME terms : (0,7)(1,6)(2,5)(3,4) ⇒ 4 ⇒ $2^4 = 16$

**Note:**

- $\Sigma m(\_\_\_)$ (Missing terms) = Equal = $\Pi M(\_\_\_)$ (Missing terms)

$\Sigma m(\_\_\_)$ (Same terms) = Comp = $\Pi M(\_\_\_)$ (Same terms)

**Note:** if $i \in \Sigma m$ then $2^n - 1 - i \in \Pi M$ then function is called self dual function

**Note:** calculating Dual for $\Sigma m(0,2,5,7)$

$\Sigma m(0,2,5,7)$

$\Sigma m(1,3,4,6)$ ← Same symbol and absent terms

calculating Compliment for $\Sigma m(0,2,5,7)$

$\Sigma m(0,2,5,7)$

$\Pi M(0,2,5,7)$ ← different symbol & same terms

## Compliment of Boolean function:

$f'(x,y) = f_D(x', y')$

- It will affect variables also $(x \rightleftarrows x')$
- $0 \rightleftarrows 1$
- AND $\rightleftarrows$ OR
- XOR $\rightleftarrows$ XNOR
- NAND $\rightleftarrows$ NOR

**Note:** If $f' = f_d$ then f is called orthogonal fn

Ex:- XOR

**Note:**

$\Sigma m$ → Equal : $\Pi M$ (missing terms)

$\Sigma m$ → dual : $\Sigma m$ (missing terms)

$\Sigma m$ → Comp : $\Pi M$ (same terms)

(Self dual fn)

## Canonical and standard forms:

Standard / Canonical → Simplified form

SSOP —Simplify→ SOP

SPOS —Simplify→ POS

**Sum of product:** (A → 1, $\bar{A}$ → 0)

B/w terms : sum

B/w literals : product

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

$= \Sigma m(3,4,5,7)$

**Product of sum:** (A → 0, $\bar{A}$ → 1)

B/w terms : product

B/w literals : sum

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

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

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

## Logic gates:

① Buffer: ▷

Purpose : For Time delay

I/p: A —Vm— A : o/p

② XOR :

- odd bit detector
- parity checking

$\overline{XOR} = XNOR$

- Bubbled AND ≡ NOR gate
- Bubbled OR ≡ NAND gate

Universal gate : NOR, NAND, 2×1 mux

Basic gates : AND, OR, NOT

single i/p gate : NOT, Buffer

## Properties of XOR:

- $A \oplus B = A\bar{B} + \bar{A}B$
- $A \oplus 0 = A$
- $A \oplus 1 = \bar{A}$
- $A \oplus A = 0$
- $A \oplus \bar{A} = 1$
- $\overline{A \oplus B} = A \odot B = \bar{A}\bar{B} + AB$
- $A[B \oplus C] = AB \oplus AC$
- $A \oplus B \oplus C = A\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC = \Sigma m(1,2,4,7)$ ] Equal
- $A \odot B \odot C = A\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC = \Sigma m(1,2,4,7)$ ] Equal
- $A \odot B = A' \oplus B$, $A \oplus B'$,

**Note:**

- If num of variables are Even then $\overline{XOR} = XNOR$
- If num of variables are odd then $XOR = XNOR$.

*) $X \oplus X \oplus X \oplus \cdots$

If XOR's are even then o/p is X

If XOR's are odd then o/p is 0

## Properties of logic gates:

| | Identity law | Commutative | Associative | Symmetric |
|---|---|---|---|---|
| AND | ✓ | ✓ | ✓ | ✓ |
| OR | ✓ | ✓ | ✓ | ✓ |
| NAND | × | ✓ | × | ✓ |
| NOR | × | ✓ | × | ✓ |
| XOR | × | ✓ | ✓ | ✓ |
| XNOR | × | ✓ | ✓ | ✓ |
| Implication $A'+B$ | ✓ | × | × | × |
| Check | Same result every time | 'a' num must be in $^nC_a$ | changing brackets method | $f = f'$ & Commutative |

*) All symmetric functions are Commutative


**Symmetric function:**
- Compliment of sym fn is also sym fn.
- ANDing, ORing of sym fn will give sym fn only.

**Distributive law:**
- XOR is not distributive on any operation including itself
- AND operation is distributive on XOR
  $A \cdot (B \oplus C) = AB \oplus AC$
- OR oper$^n$ is distributive on XNOR
  $A + (B \odot C) = (A+B) \odot (A+C)$

How to Check?


**Step1**: Check symmetricity ⇒ If symmetric then Commutative bcoz

**Step2**: Terms with 'a' num must be in $^nC_a$ → If requirements met then Commutative
Else not commutative

Ex: $f(w,x,y,z) = \Sigma(0,1,2,4,15)$

Comp → 0 1 1 1 4 ← num of 1's.
(a) (a) (a) (a)
(1) (3) (1)

Now, n = 4

$^4C_0 = 1$ ✓ | $^4C_1 = 4$ × | $^4C_4 = 1$ ✓ → Not symmetric

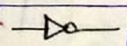
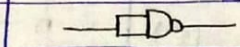
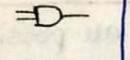
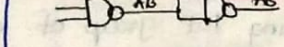
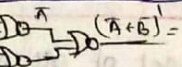
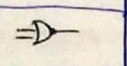
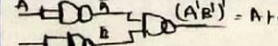
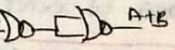
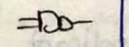
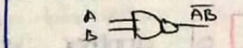
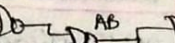
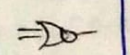
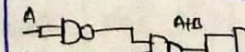
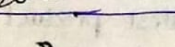
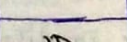
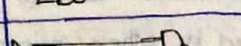
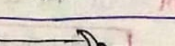
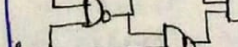
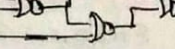
Add $\Sigma(8)$ to $\Sigma(0,1,2,4,15)$ → $\Sigma(0,1,2,4,8,15)$

1 1 1 1 1 1
0 1 1 1 1 4
$^4C_0$ $^4C_1$ $^4C_4$ ✓ → Symmetric
∴ Commutative.

### Compensation theorem:

D
- $AP_1 + A'P_2 + P_1P_2 = AP_1 + A'P_2$
- $(A+P_1)(A'+P_2)(P_1+P_2) = (A+P_1)(A'+P_2)$

### Conversions using NAND and NOR:

| Basic gate | Symbol | Using NAND gate | Min num of NAND gates | Using NOR gate | Min num of NOR gates |
|---|---|---|---|---|---|
| Not |  |  | 1 |  | 1 |
| AND |  |  | 2 |  | 3 |
| OR |  |  | 3 |  | 2 |
| NAND |  |  | 1 |  | 4 |
| NOR |  |  | 4 |  | 1 |
| XOR |  |  | 4 |  | 5 |
| XNOR |  |  | 5 |  | 4 |

**Note:**
- To design n i/p NAND gate, min num of 2 i/p NAND gates reqd is $2n-3$
- To design n i/p NOR gate, min num of 2 i/p NOR gates reqd is also $2n-3$

**Designing of Boolean fn using NAND:**
- Step①: Given fn should be in sop
- Step②: Design AND-OR circuit
- Step③: Replace AND with NAND and OR with bubbled OR
- Step④: Replace Bubbled OR with NAND gate

**Designing of Boolean fn using NOR:**
- Step①: Given function should be in pos
- Step②: Design OR-AND circuit
- Step③: Replace OR with NOR and AND with Bubbled AND
- Step④: Replace Bubbled AND with NOR

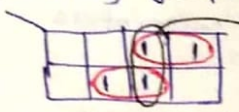
**K-maps:** (Karnaugh maps)

**Purpose:** ~~Num of terms~~ Complicated functions can be simplified
- If num of terms are less we use boolean algebra.
  If num of terms are more we use K-maps.
- K-map is a systematic graphical method which consists of $2^n$ cells for n-var K map

i/p (SSop / Spos) → K-map → (Sop / Pos) o/p

**Grouping:**
32 16 8 4 2 1 → priority

- Group of 2 terms – pair
  4 terms – Quad
  8 terms – octet
- while forming groups atleast one term should be independent otherwise it is redundant




**Types of K-maps**

**Completely specified fn**
Ex:- $F(A,B,C) = \Sigma(3,5,6)$

**Incompletely specified function**
Ex:- $F(A,B,C) = \Sigma(3,5,6) + \Sigma_{\phi}(1,2)$

**Variable Entrant K-map**
- used to represent higher values into lower maps

**Note:**
- Grouping of all minterms / max terms are compulsory
- Grouping of all don't cares are not compulsory

**Note:**
$xy + \bar{x}z + yzPqrst\text{----} = xy + \bar{x}z$

| Note: | var Eliminated | remaining |
|---|---|---|
| Pair ($2^1$) | 1 | n-1 |
| Quad ($2^2$) | 2 | n-2 |
| octet ($2^3$) | 3 | n-3 |
| ⋮ | ⋮ | ⋮ |
| ($2^n$) | n | 0 |

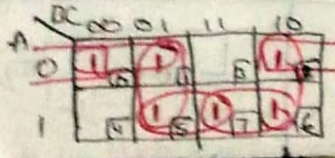
**Implicant (I):** It is a set of all adjacent minterms nothing but group of all possible combinations of octet, Quad, Pair ---etc
- It also considers redundant terms.

**Prime Implicant (PI):** It is an Implicant which is not a subset of another Implicant [Smallest product term]

**Essential prime Implicant (EPI):** It is a Prime Implicant which contains atleast one minterm independent [which cannot be covered by other prime Implicant]

EPI ⊆ PI ⊆ I

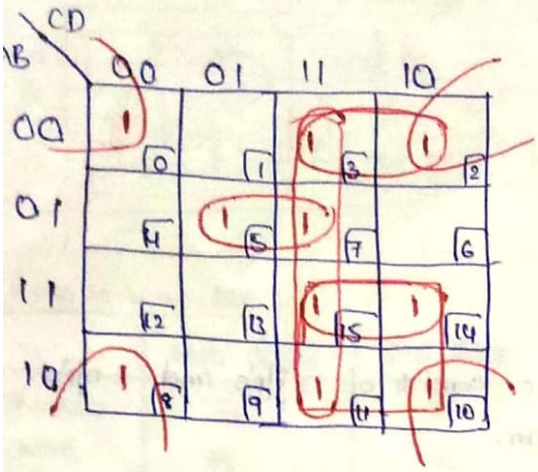
Ex:- $\Sigma m(0,1,2,5,6,7)$




PI: (6)

EPI: None

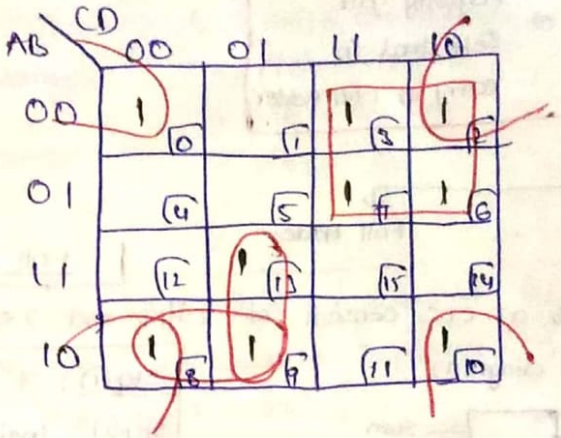
Ex:- $\Sigma m(0,2,3,5,7,8,10,11,14,15)$




PI : 5

EPI : 3

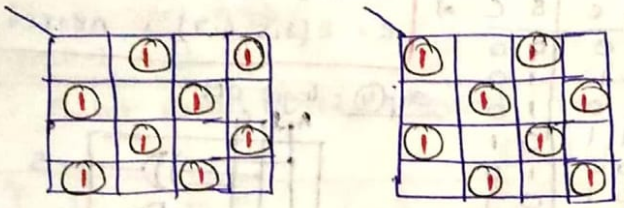
Ex:- $\Sigma m(0,2,3,6,7,8,9,10,13)$




PI : 4

EPI : 3

**Note :** For a n-variable boolean function max num of PI's are $2^{n-1}$



$PI = EPI = 2^{n-1} \rightarrow 8$

**Note :**

- For XOR, If num of is { odd → o/p is 1; Even → o/p is 0 } → used to design odd functions.
- For XNOR If num of 1's { odd → o/p is 0; Even → o/p is 1 } → used to design Even functions.

Ex:- $\Sigma m(1,2,4,7,8,11,13,14)$

num of 1's 1 1 1 3 1 3 3 3 → odd.

o/p : $A \oplus B \oplus C \oplus D$

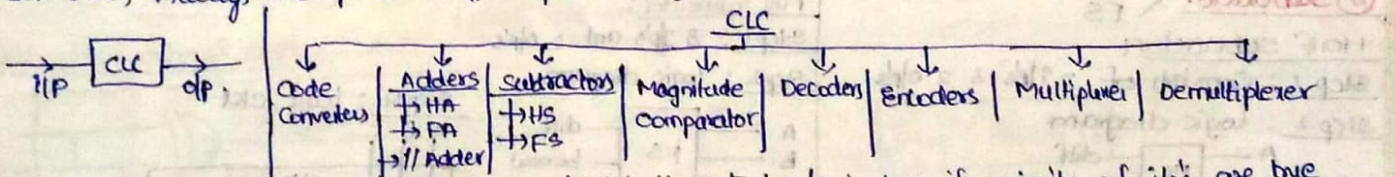
$\Sigma m(0,3,5,6,9,10,12,15)$

4 2 2 2 2 2 2 0 → num of 1's are even

o/p : $A \odot B \odot C \odot D$

**Combinational logic circuits : (CLC)**

In CLC, Always the present o/p depends only on present i/p but not on prev data




**① Code Converters :**

② i/p's = 3 o/p = 1

The o/p is 1 when bin value less than 3 otherwise zero.

sol ① i/ps : 3 [A,B,C] | o/p's : 1 [Y]

② Truthtable :

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

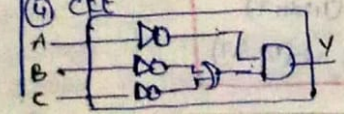
③ Simplification

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

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

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

④ ckt




**Majority gate :** o/p is true if majority of i/p's are true.

① i/p : 3 [A,B,C] o/p : 1 [Y]

② Truth table

| A | B | C | Y |
|---|---|---|---|
| 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 |

③ Simplification

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

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

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

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

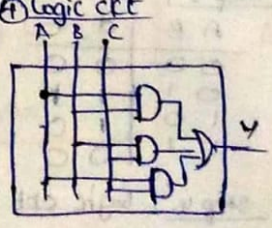
$= AB + AC + \bar{A}BC$

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

$= AB + C(A + B)$

$Y = AB + BC + CA$

④ Logic ckt



$AB+BC+CA \Rightarrow \Sigma(3,5,6,7)$ → Majority gate, self dual fn, carry in Full Adder

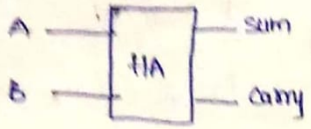
## ③ Adders :

Half Adder, Full Adder

### Half Adder :

Step ① : It is a CLC consists of 2 i/p's and 2 o/p's

Step ② : Logic diagram




Step ③ : Truth table

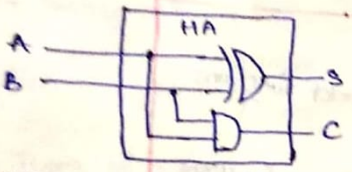
| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

$S = A'B + AB'$

$S = A \oplus B$

$C = AB$

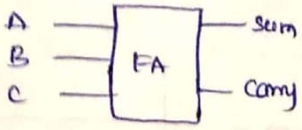
Step ④ : Logic ckt




### Full Adder :-

Step ① : It is a CLC consists of 3 i/p's and 2 o/p's.

Step ② : Logic diagram



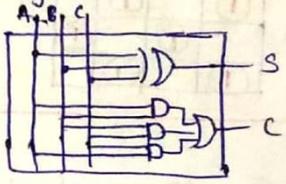

Step ③ : Truth table

| A | B | C | 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 |

$S = \Sigma(1,2,4,7) = A \oplus B \oplus C$

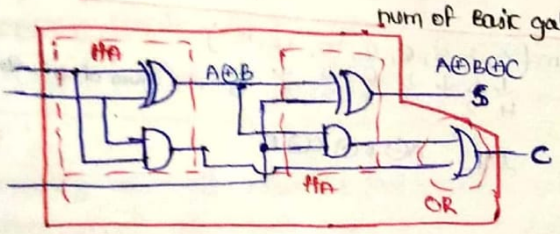
$C = \Sigma(3,5,6,7) = AB+BC+CA$

Step ④ : Logic ckt




Q To design FA using HA, num of HA reqd 2

num of Basic gates reqd 1 (OR gate)




$C = (A \oplus B)C + AB$

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

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

$= C(B+A) + BA$

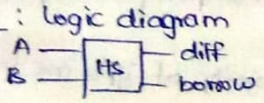
$C = AB+BC+CA$

## ③ Subtractor (HS, FS)

### Half Subtractor :

Step 1 : Consists of 2 i/p's & 2 o/p's

Step 2 : Logic diagram



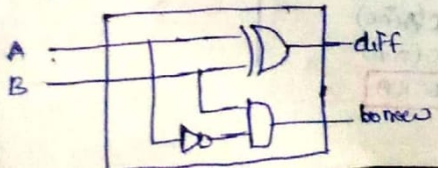

Step 3 : Truth table

| A | B | d | b |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

$d = A'B + AB'$

$b = A'B$

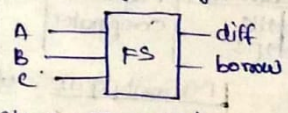
Step 4 : Logic ckt




### Full subtractor :

Step 1 : 3 i/p's and 2 o/p's

Step 2 : Logic diagram



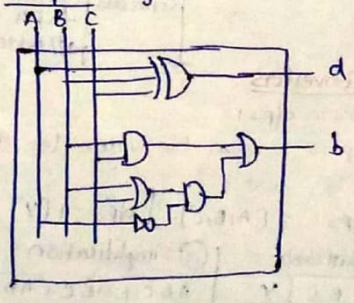

Step 3 : Truth table

| A | B | C | d | b |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

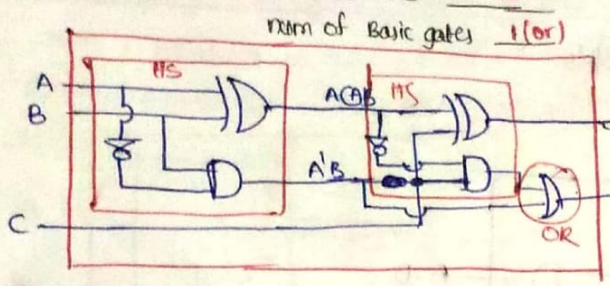
$d = A \oplus B \oplus C = \Sigma(1,2,4,7)$

$b = A'B + BC + A'C = \Sigma(1,2,3,7)$

Step 4 : Logic ckt



b) To design FS, num of HS 2
num of Basic gates 1(or)




$b = (\overline{A \oplus B})C + A'B$
$= (A'B' + AB)C + A'B$
$= A'B'C + ABC + A'B$
$= A'B + BC + A'C$

**Points to remember :**

| | Half adder | Full Adder |
|---|---|---|
| NAND | 5 | 9 |
| NOR | 5 | 9 |

*) 1 Full Adder = 2 Half Adder + 1 OR gate.
1 Full Adder = 1 Full subtractor + 1 not gate

HA
*) Num of MUX(2×1) reqd to Implement HA is 3
*) Min num of logic gates to Implement HA is 3 (Except XOR & XNOR)

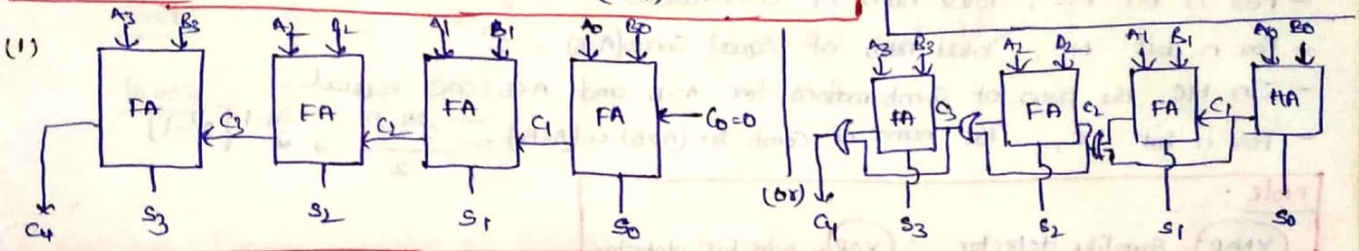
In sum view
Adders → Same → subtractor
In carry view
Adders → A's Inverted → subtractor

### ③ Binary or parallel Adder :

4 bit parallel Adder :

**Note :** n-bit parallel Adder requires n FA or (n-1) FA and 1 HA

n-bit // Adder = (n-1) FA + 1 HA
= (n-1) [2HA + 1OR] + 1 HA
n-bit // Adder = (2n-1) HA + (n-1) OR




**Note :** Time reqd for n-bit parallel Adder is

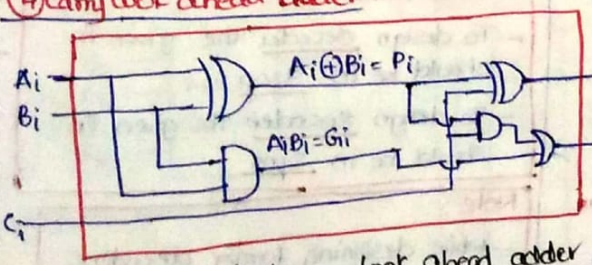
$$\text{Time} = (n-1)T_c + \text{Max}[T_s, T_c]$$

$T_c$ = Time reqd to produce carry
$T_s$ = Time reqd to produce sum

∴ parallel Adder :
→ In parallel Adder carry at any stage depends on all previous carry. so, time reqd for Add$^n$ is more. Hence parallel Adder is a slow Adder

↓sol
carry look ahead adder

### ④ Carry look ahead adder :




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

For a 4 bit carry look ahead adder.

case1 : i=0 ; $C_1 = G_0 + P_0 C_0$
case2 : i=1 ; $C_2 = G_1 + P_1[G_0 + P_0 C_0]$
case3 : i=2 ; $C_3 = G_2 + P_2[G_1 + P_1 G_0 + P_1 P_0 C_0]$
case4 : i=3 ; $C_4 = G_3 + P_3[G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0]$

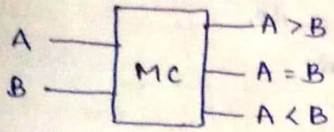
**Note :**
n-bit carry look ahead adder
- AND gates → $\frac{n(n+1)}{2}$
- OR gates → n

∴ carry look ahead adder :
→ carry at any stage depends only on Initial carry but not on prev carry Hence carry look ahead adder is the fastest adder

## ⑤ Magnitude Comparator : (MC)

**Step 1** : It is a CLC consists of 2 i/p's and 3 o/p's

**Step 2** : logic diagram




**Step 3** : Truth table

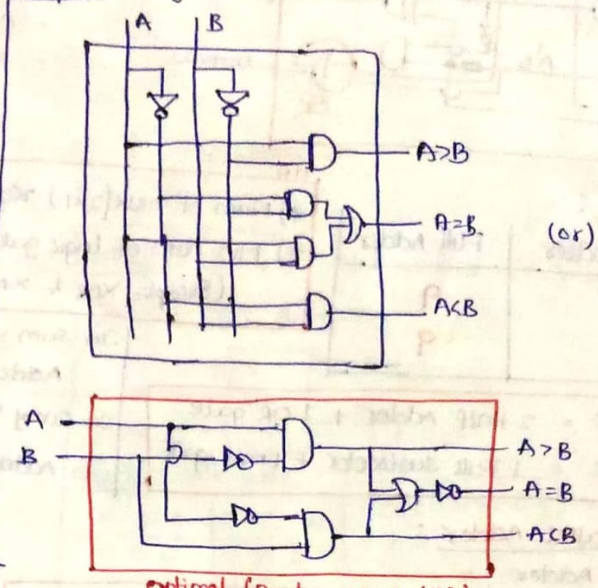
| A | B | A>B | A=B | A<B |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |

$A>B \Rightarrow AB'$

$A=B \rightarrow A \odot B$

$A<B \Rightarrow A'B$

**Step 4** : Logic ckt




optimal (3 not, 2 AND, 1 OR)

**Points to remember :**

- For n bit MC, Total num of combinations - $2^{2n}$
- For n bit MC, Total num of Equal Comb(A=B) - $2^n$
- In MC, the num of combinations for A>B and A<B are equal
- For n bit MC, Tot num of comb for (A>B) or (A<B) - $\frac{2^{2n}-2^n}{2} \Rightarrow 2^{n-1}[2^n-1]$

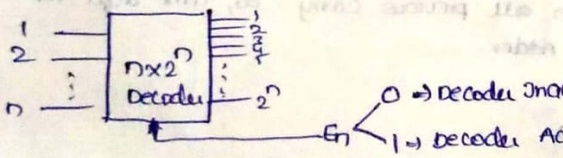
**Note :**

(XNOR) - Equality detector ; (XOR) - odd bit detector

## ⑥ Decoders :

**Step 1** ; It is a CLC which consists of n i/p's and $2^n$ o/p's.

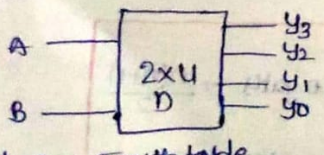
**Step 2** : Logic diagram




**2×4 Decoder :**

**Step 1** : A B → i/p's $y_0\ y_1\ y_2\ y_3$ → o/p's

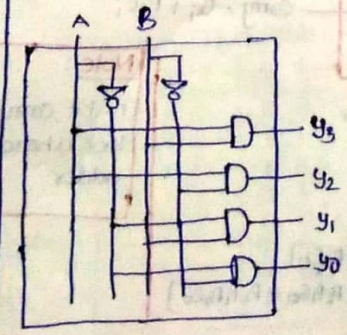
**Step 2** : Logic diagram




**Step 3** : Truth table

| En | A | B | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
|---|---|---|---|---|---|---|
| 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 |
| 0 | x | x | 0 | 0 | 0 | 0 |

**Step 4** : Logic ckt




**Note :-**

- Internal ckt of decoder consists of AND gates
- Internal ckt of encoder consists of OR gates
- To design decoder the given fn should be in SSOP
- To design encoder the given fn should be in SPOS.

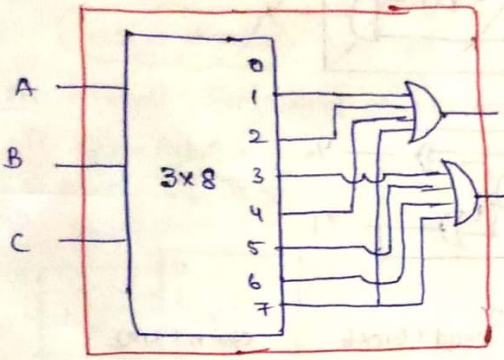
**Note :**

- While designing larger decoders with smaller decoders always the num of decoders are odd

## Implementing Full Adder using decoder:

Sum = $A \oplus B \oplus C = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC = \Sigma(1,2,4,7)$

Carry = $AB + BC + CA = ABC + AB\bar{C} + \bar{A}BC + A\bar{B}C = \Sigma(3,5,6,7)$




→ 1 (3×8 decoder)
2 (4 I/p OR gates.)

### Design 10×1024 D using 3×8 D:

**Note:** Decoders at each level = Reqd o/p / Given o/p

Now,
level 1 : 1024/8 → 128
level 2 : 128/8 → 16
level 3 : 16/8 → 2
level 4 : 2/8 → ⌈0.25⌉ → 1

147
odd

### Design 5×32 D using 2×4 & 3×8

Reqd : 5×32 Avail : 2×4, 3×8

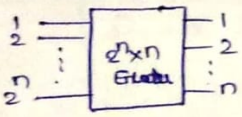
level 1 : 32/8 → 4
level 2 : 4/4 → 1

5
odd

∴ 4 (3×8 D) and 1 (2×4 D)

## (7) Encoders: (Reverse operation of decoder)

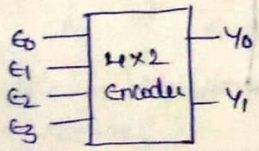
- It is a CLC which consists of $2^n$ i/p's and n o/p's
- logic diagram




### 4×2 Encoder:

step 1: $E_0\ E_1\ E_2\ E_3$ → i/p's, $Y_0\ Y_1$ → o/p's.

step 2: Logic diagram



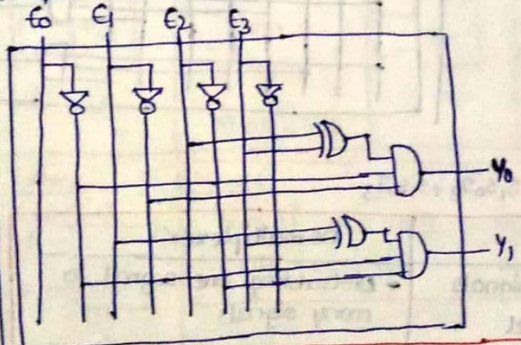

step 3: Truth table

| $E_0$ | $E_1$ | $E_2$ | $E_3$ | $Y_0$ | $Y_1$ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |

$Y_0 = \bar{E_0}\bar{E_1}[E_2 \oplus E_3]$

$Y_1 = \bar{E_0}\bar{E_2}[E_1 \oplus E_3]$

step 4: logic ckt




**Note:** Encoders work properly if only 1 i/p is active high, If more than one i/p is active high then encoders give invalid o/p

↓ sol

Priority encoders

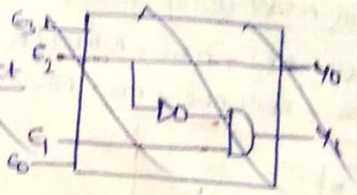
## Priority Encoder :

| $E_0$ | $E_1$ | $E_2$ | $E_3$ | $Y_0$ | $Y_1$ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| x | 1 | 0 | 0 | 0 | 1 |
| x | x | 1 | 0 | 1 | 0 |
| x | x | x | 1 | 1 | 1 |

$Y_0 = E_2$

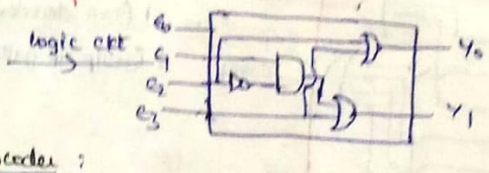
$Y_1 = E_1 E_2'$

Incorrect o/p set




$Y_0 = E_2 E_3' + E_3 \rightarrow E_3 + E_2$

$Y_1 = E_1 \bar{E_2} \bar{E_3} + E_3 \rightarrow E_3 + E_1 \bar{E_2}$




## Design 64x6 Encoder using 4x2 Encoder :

**Note :** Num of Encoders reqd at each level = $\frac{\text{reqd o/p}}{\text{Given o/p}}$

Now,

reqd : 64x6 Given : 4x2

Level 1 : $\frac{64}{4} = 16$

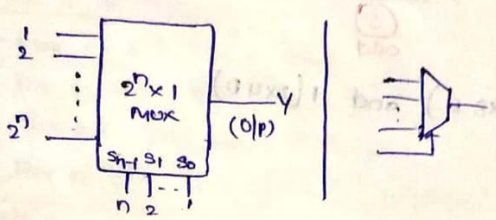
Level 2 : $\frac{16}{4} = 4$

Level 3 : $\frac{4}{4} = 1$

21

## (8) Multiplexer : (Mux)

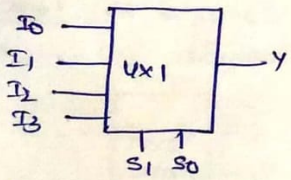
- It is a CLC which consists of $2^n$ inputs and a single o/p with 'n' selection lines.
- Logic diagram




## Design 4x1 Mux :

**Step 1 :** I/P's :- $I_0, I_1, I_2, I_3$ ; o/p's : Y ; selection lines : $S_0, S_1$

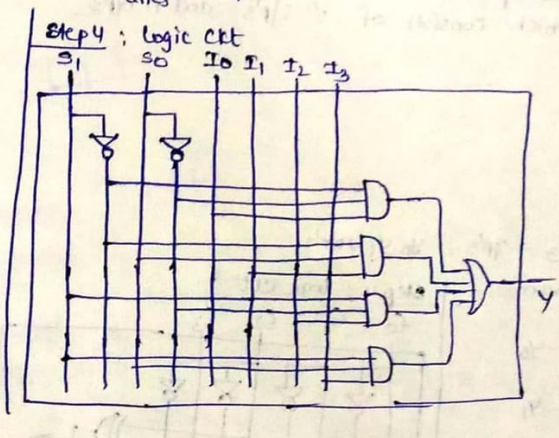
**Step 2 :** Logic diagram




**Step 3 :** Truth table

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

**Step 4 :** Logic ckt




$Y(S_1, S_0) = S_1' S_0' I_0 + S_1' S_0 I_1 + S_1 S_0' I_2 + S_1 S_0 I_3$

| **Note :** Multiplexer | Demultiplexer |
|---|---|
| - sharing of many signals through single channel<br>- Many to one<br>- Data selector<br>- Converts parallel data to serial data | - Distributing one signal to many signals<br>- One to many<br>- Data distributor<br>- Converts serial data to parallel data. |

# Multiplexer design models

## Fn i/p's = selection i/p's

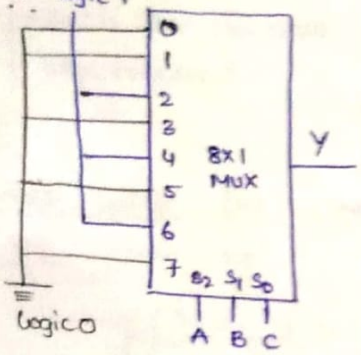
Design directly

Ex:- $F(A,B,C) = Em(2,4,6)$ using 8x1 mux

Fn i/p's = A,B,C = 3

selection = 3

∴ Logic 1




- No extra H/W

## Fn i/p's > selection i/p's

Implementation table method

Ex:- $F(A,B,C) = Em(2,4,6)$ using 4x1 mux

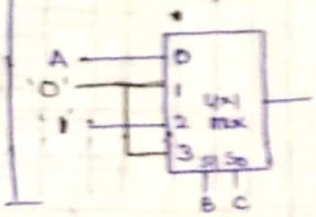
FI = 3

SI = 2

FI > SI

Implementation table

| | $\bar{B}\bar{C}$ $I_0$ | $\bar{B}C$ $I_1$ | $B\bar{C}$ $I_2$ | $BC$ $I_3$ |
|---|---|---|---|---|
| $\bar{A}$ | 0 | 1 | (2) | 3 |
| $A$ | (4) | 5 | (6) | 7 |
| | A | 0 | 1 | 0 |

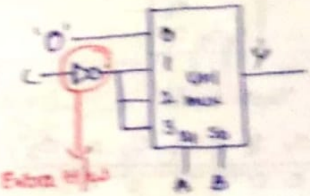



- requires H/W

(ex)

| | $\bar{A}\bar{B}$ $S_0$ | $\bar{A}B$ $S_1$ | $A\bar{B}$ $S_2$ | $AB$ $I_3$ |
|---|---|---|---|---|
| $\bar{C}$ | 0 | (2) | (4) | (6) |
| $C$ | 1 | 3 | 5 | 7 |
| | 0 | $\bar{C}$ | $\bar{C}$ | $\bar{C}$ |

(ex)




## Fn i/p's < selection i/p's

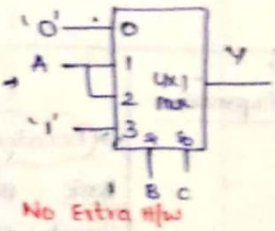
Practically not feasible

## Full Adder carry using 4x1 mux :

Carry = $AB+BC+CA = \Sigma(3,5,6,7)$

FI = 3, SI = 2 FI > SI

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| $\bar{A}$ | 0 | 1 | 2 | (3) |
| $A$ | 4 | (5) | (6) | (7) |
| | 0 | A | A | 1 |



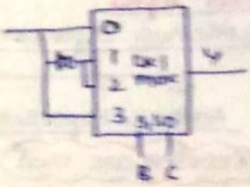

No Extra H/W

## Full Adder Sum using 4x1 Mux :

Sum = $A \oplus B \oplus C = Em(1,2,4,7)$

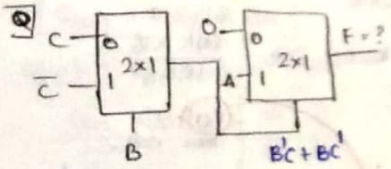
FI = 3, SI = 2 FI > SI

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| $\bar{A}$ | 0 | (1) | (2) | 3 |
| $A$ | (4) | 5 | 6 | (7) |
| | A | $\bar{A}$ | $\bar{A}$ | A |




(1 Not gate)
Extra H/W

Q



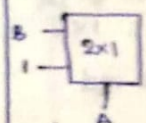

$F = (B'C+BC')'(0) + (B'C+BC')A$

$= AB'C+ABC'$

Note: 2x1 mux acts as a universal logic gate

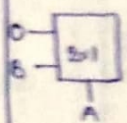
NOT :


OR

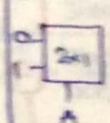



$AB+A \text{ or } A+B$

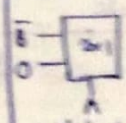
AND




Buffer

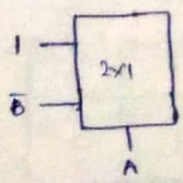



NOR :-




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

NAND

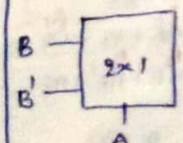



$A'+AB'$

$A'+B'$

$(AB)'$

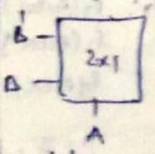
XOR




$A'B+AB'$

$A \oplus B$

XNOR




$A'B'+AB$

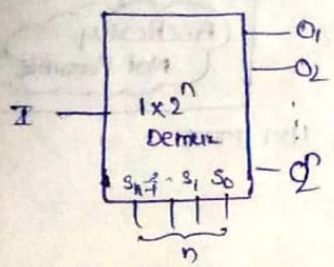
$A \odot B$

**Note :**

Num of mux req at each level = $\frac{\text{Req i/p}}{\text{Given i/p}}$

## ④ Demultiplexer : (Reverse operation of mux)

- It is a CLC which consists of single i/p and $2^n$ o/p's with n selection lines.
- logic diagram



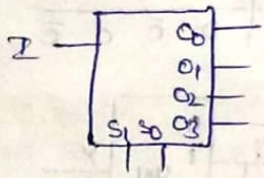

**Note :**

Num of Demux reqd at each level = Reqd o/p / Given o/p

Construct 1x4 Demux :

Step 1 : i/p (I) ; o/p's :- $O_0\ O_1\ O_2\ O_3$

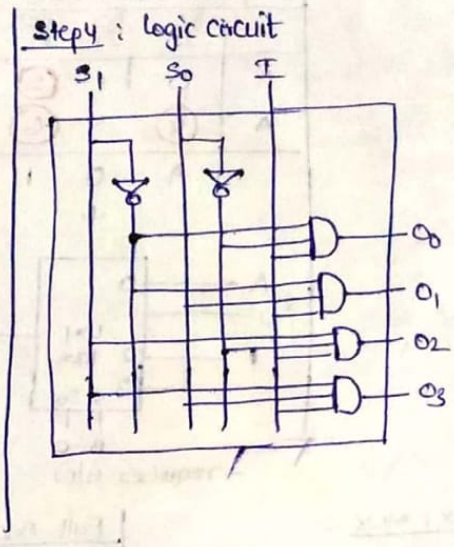
Step 2 : logic diagram




Step 3 : Truth 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 |

Step 4 : logic circuit




$O_0 = S_1' S_0' I$ ; $O_1 = S_1' S_0 I$ ; $O_2 = S_1 S_0' I$ ; $O_3 = S_1 S_0 I$

Other topics :

① Memory Expansion :

### Address Expansion

Decoders are used

Basic : 1K x 8

Target : 4K x 8

Num of rows & cols = Target / Basic

= 4K x 8 / 1K x 8 = 4 x 1 (rows, col)

4 chips reqd

2 x 4 Decoder reqd

### Data Expansion

Basic : 1K x 8

Target : 1K x 16

Num of rows & cols = 1K x 16 / 1K x 8

= 1 x 2 (mxn for)

2 chips reqd

### Address & Data Expansion

Decoders are used

Basic : 1K x 8

Target : 64K x 16

Num of rows & cols = 64K x 16 / 1K x 8

= 64K x 2 rows cols.

= 128 chips reqd

6 x 64 Decoder reqd

② PLA : (programmable logic array) → contains → AND ROM, OR ROM

- Supports minimization
- Size of PLA is measured in terms of links ie

Num of links = (2n + q) p

n → num of var's in fns

p → Independent products

q → Total num of fns

Ex :- $f_1(A,B,C) = A + BC$ (P1, P2)

$f_2(A,B,C) = AB + BC + CA$ (P3, P4)

$f_3(A,B,C) = A'B + AB' + AC$ (P5, P6)

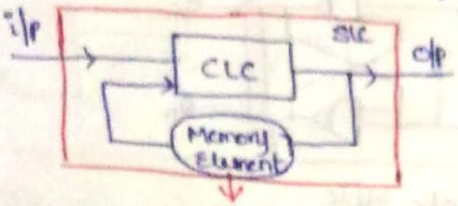
n = 3, p = 6, q = 3

Num of links = (2x3+3)x6

⇒ 54 links

## Sequential circuits :

- A circuit in which present o/p not only depends on present i/p but also on prev o/p is called sequential logic circuit




SLC = CLC + Memory Element

| *) | CLC | SLC |
|---|---|---|
| | - Can't store data | - can store data |
| | - Faster | - slower |
| | - parallel Adder, MUX, Demux ... etc | - Serial Adder, registers, Counters etc |

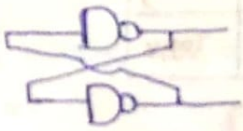
### 1) Flipflop :

- used to store 1bit data
- Edge triggered

- SR FF } 2 i/p
- JK FF }
- D FF } 1 i/p
- T FF

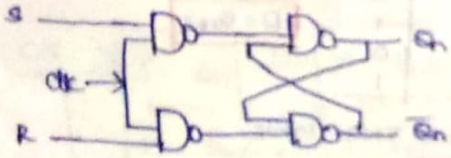
2 o/p's ($Q_n$, $\bar{Q}_n$)

Flipflop ckt :



### a) SR Flipflop : (set = 1, reset = 0)

step1 : Logic ckt




step2 : characteristic table / Truth table

| S | R | $Q_n$ | $Q_{n+1}$ | State |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Memory |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | Reset |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | Set |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | x | Invalid |
| 1 | 1 | 1 | x | |

o/p = $\Sigma(1,4,5) + \Sigma(6,7)$ (d)

$Q_{n+1} = S + R'Q_n$

step3 : 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 |

$S = Q_{n+1}$

$R = \bar{Q}_{n+1}$

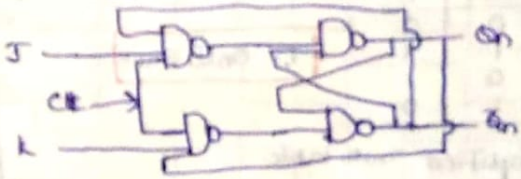
step4 : Simplified Truth table.

| S | R | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | x |

∴) when S=1, R=1 next state value is not determined → JK Flipflop

### b) JK Flipflop

step1 : Logic ckt




step2 : characteristic table

| J | k | $Q_n$ | $Q_{n+1}$ | State |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Memory |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | Reset |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | Set |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 1 | Toggle |
| 1 | 1 | 1 | 0 | |

$Q_{n+1} = \Sigma(1,4,5,6)$

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

step3 : 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 |

$J = Q_{n+1}$

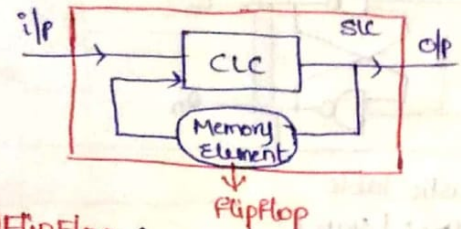
$K = \bar{Q}_{n+1}$

step4 : simplified Truth table

| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\bar{Q}_n$ |

## Sequential circuits :

- A circuit in which present o/p not only depends on present i/p but also on prev o/p is called sequential logic circuit




SLC = CLC + Memory Element

| *) CLC | SLC |
|---|---|
| - Can't store data | - can store data |
| - Faster | - slower |
| - parallel adder, MUX, Demux etc | - serial adder, registers, counters etc |

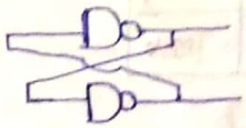
### i) Flipflop :

- used to store 1bit data
- Edge triggered

- SR FF } 2 i/p
- JK FF }
- D FF } 1 i/p
- T FF }

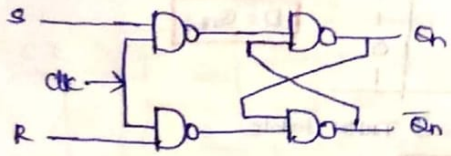
2 o/p's ($Q_n$, $\bar{Q}_n$)

Flipflop ckt :



### a) SR Flipflop : (set = 1, reset = 0)

step 1 : logic ckt




step 2 : characteristic table / Truth table

| S | R | $Q_n$ | $Q_{n+1}$ | State |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Memory |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | Reset |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | Set |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | x | Invalid |
| 1 | 1 | 1 | x | |

o/p = $\Sigma(1,4,5) + \Sigma_\phi(6,7)$

$Q_{n+1} = S + R'Q_n$

step 3 : 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 |

$S = Q_{n+1}$

$R = \bar{Q}_{n+1}$

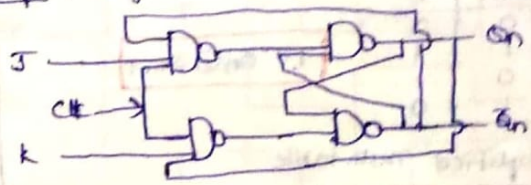
step 4 : Simplified Truth table.

| S | R | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | x |

∴) when S = 1, R = 1 next state value is not determined → JK Flipflop

### b) JK Flipflop

step 1 : Logic ckt




step 2 : characteristic table

| J | k | $Q_n$ | $Q_{n+1}$ | State |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Memory |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | Reset |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | Set |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 1 | Toggle |
| 1 | 1 | 1 | 0 | |

$Q_{n+1} = \Sigma(1,4,5,6)$

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

step 3 : 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 |

$J = Q_{n+1}$

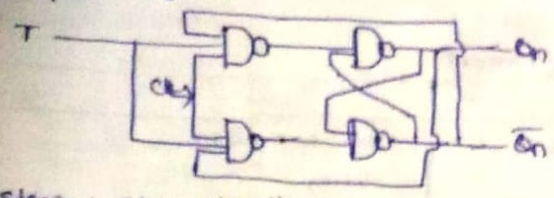
$K = \bar{Q}_{n+1}$

step 4 : simplified Truth table

| J | k | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\bar{Q}_n$ |

## c) T Flipflop :

step 1 : logic ckt




step2 : characteristic table

| T | Qn | Qn+1 | state |
|---|---|---|---|
| 0 | 0 | 0 | Memory |
| 0 | 1 | 1 | |
| 1 | 0 | 1 | Toggle |
| 1 | 1 | 0 | |

$Q_{n+1} = T \oplus Q_n$

step3 : Excitation table

| Qn | Qn+1 | T |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

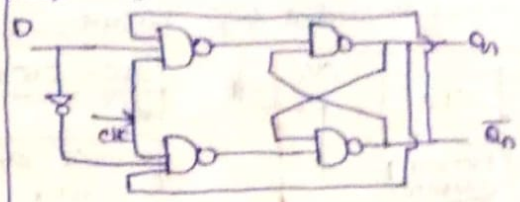
$T = Q_n \oplus Q_{n+1}$

step4 : simplified Truth table

| T | Qn+1 |
|---|---|
| 0 | Qn |
| 1 | $\overline{Q_n}$ |

## d) D Flipflop :

step1 : logic ckt




step2 : characteristic table

| D | Qn | Qn+1 | state |
|---|---|---|---|
| 0 | 0 | 0 | Reset |
| 0 | 1 | 0 | |
| 1 | 0 | 1 | set |
| 1 | 1 | 1 | |

$Q_{n+1} = D$

step3 : Excitation table

| Qn | Qn+1 | D |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

$D = Q_{n+1}$

step4 : simplified Truth table

| D | Qn+1 |
|---|---|
| 0 | 0 |
| 1 | 1 |

**Note :**

| FF's | characteristic Expr | Excitation Expr | states |
|---|---|---|---|
| SR | $Q_{n+1} = S + R'Q_n$ | $S = Q_{n+1}, R = \overline{Q_{n+1}}$ | set, reset, memory |
| JK | $Q_{n+1} = JQ_n' + K'Q_n$ | $J = Q_{n+1}, R = \overline{Q_{n+1}}$ | set, reset, memory, Toggle |
| T | $Q_{n+1} = T \oplus Q_n$ | $T = Q_n \oplus Q_{n+1}$ | memory, Toggle |
| D | $Q_{n+1} = D$ | $D = Q_{n+1}$ | set, reset |

*) D FF is used to introduce time delays
↓
used as buffer gate

*) D FF
↓ delay
↓ data

## Flip Flop Conversions :

step1 : Get characteristic table of Target flipflop

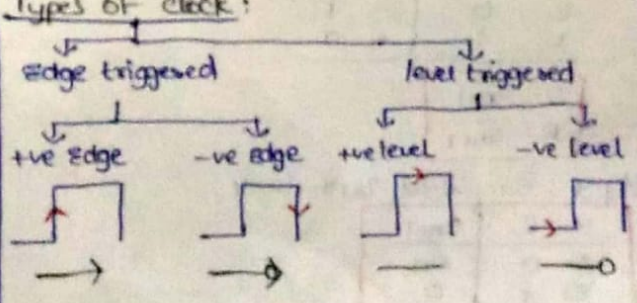
step2 : Replace next state (output) using Excitation table of given FF

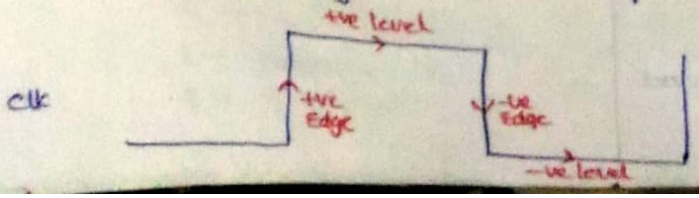
step3 : Get Expr for i/p's of given FF and realize them

## Clock :

- clock is used to change the o/p of any circuit at a specified point.
- clock is a particular type of signal that oscillate between high and low state
- clock signal is produced by clock generator
- clock is in the form of square wave with a fixed constant frequency.

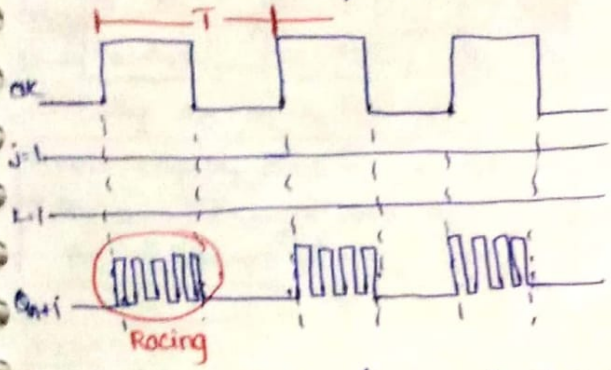
Types of clock :



## Race around problem:

In JK FF when J=1 and K=1, o/p continuously changes from 0 to 1 and 1 to 0. This problem is called race around problem.




Race around condition
$T/2 > T_{pd}$

T → Time period of clk
$T_{pd}$ → prop delay

### Conditions to overcome racing:

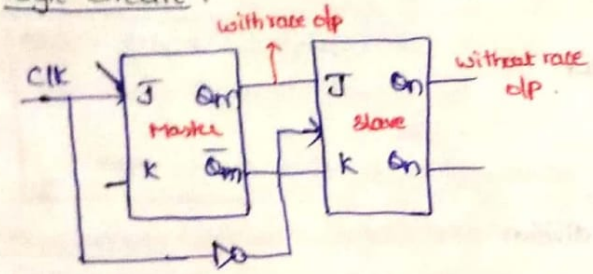
(i) $T/2 < T_{pd}$ (Practically not possible)
(ii) Edge triggering
(iii) Master slave Flip flop
} Feasible

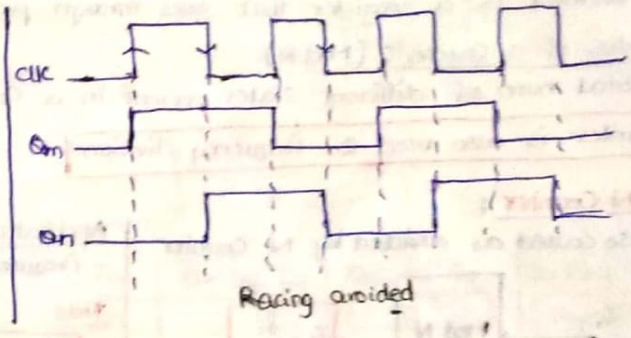
### Master slave Flipflop: (constructed using 2 FF's but used to store only one bit)

**Note:** Race around problem occurs only on level triggered but not on edge triggered

Master slave FF is same as -ve edge triggering

### logic circuit:




## 2) Registers:

- Group of flipflops is called register
- For n-bit register we require n flipflops
- Generally D Flipflops are used in registers.

### Shift register:

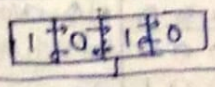
- A register in which shifting of data is possible in one or both directions is called shift reg.
- shift registers are capable of doing (i) serial loading
  (ii) parallel loading
  (iii) left shift opern (multiply by 2)
  (iv) Right shift opern (divide by 2)

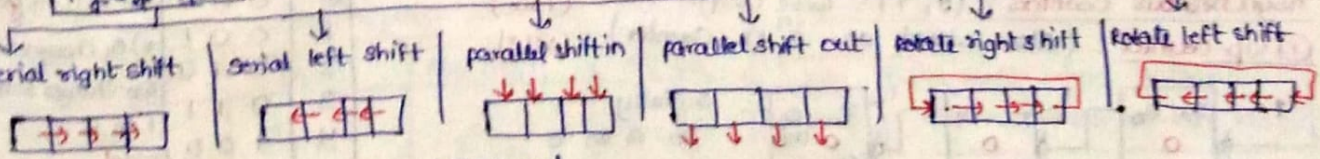
JK all functionality present → Universal shift reg

### Types

- SISO (slowest)
- SIPO
- PISO
} → used in Transmitting & recieving section in computer networks
- PIPO (fastest)

### operations on shift registers:

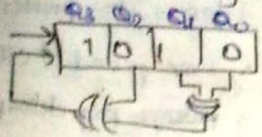



**Note:**

| Reg | Loading | Reading | Total clks req | |
|---|---|---|---|---|
| SISO | n | n-1 | 2n-1 | → slowest |
| SIPO | n | 0 | n | |
| PISO | 1 | n | n+1 | |
| PIPO | 1 | 0 | 1 | → fastest |

Q) A right shift reg




After how many clk pulses o/p = 1010 — 7 clks.

| clk | $Q_0 \oplus Q_1 \oplus Q_2 \oplus Q_3$ / $J_0$ | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|---|
| 0 | – | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 2 | 0 | 0 | 1 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 1 |
| 5 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 | 0 |
| (7) | 1 | 1 | 0 | 1 | 0 |

## (3) Counters :

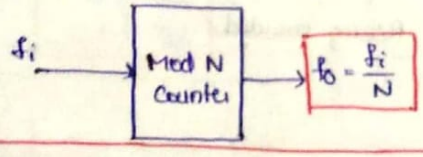
- A counter is a register that goes through pre defined sequence of states

**Modulus of a Counter : (Mod N)**

- Total num of different states present in a Counter
- Counter is also used for frequency division

**Mod N Counter :**

- Also called as divided by N counter



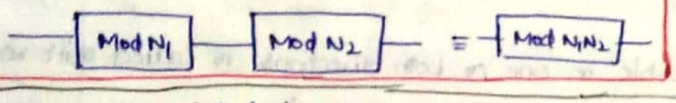

**Applications :**

- Frequency division

**Note :**

Num of states $(N) \leq \lceil 2^n \rceil$ ; n - num of FF's

**Note :**




**Note :**

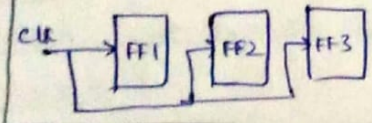
$Freq_0(f) = \frac{1}{n * tpd}$ ; tpd = prop delay

Time period $(T) = \frac{1}{f} = n * tpd$

**Note : sequential logic ckts**

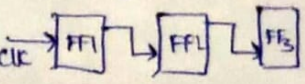
**Synchronous SLC**

- same clock pulse is applied for all flip flops
- o/p changes at the same time




**Asynchronous SLC**

- only first FF is connected by clock and rem clk signals are o/p of prev FF
- o/p changes at diff time

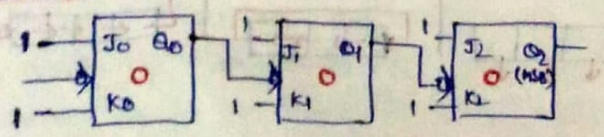



**Types of Counters**

- Async Counters (Ripple)
  - Up Counter
  - Down Counter
- sync Counters
  - Ring Counter
  - Johnson Counter (Twisted ring Counter)

**Asynchronous Counters : (JK FF's were used)** (Free) (self)

**(1) Up Counter : (Also called as $2^n$ : 1 Counter)**




- next Flipflop clock is given from Q
- For a n bit Async up Counter output repeats after $2^n$ clocks

| clk | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|
| 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 |
| 8 | 0 | 0 | 0 |

*) $Q_0$ changes for each clk pulse

*) $Q_1$ changes when $Q_0$ changes from 1 to 0

*) $Q_2$ changes when $Q_1$ changes from 1 to 0

## ① Down Counter : (Also called as, $2^n$:1 Counter)


Initially $Q_0 = Q_1 = Q_2 = 0$

| clk | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 0 |
| 3 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 0 | 1 | 1 |
| 6 | 0 | 1 | 0 |
| 7 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 |

*) $Q_0$ changes for each clk pulse
*) $Q_1$ changes when $Q_0$ changes from 0 to 1
*) $Q_2$ changes when $Q_1$ changes from 0 to 1

→ Next Flipflop clock is given from $\overline{Q_i}$
→ For a n-bit Async down counter the o/p changes after $2^n$ clocks

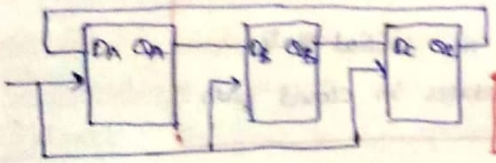
**Note :**
In Down Counter if -ve Edge clk is replaced with +ve Edge clock then it acts as up Counter
In up Counter if -ve Edge clk is replaced with +ve Edge clock then it acts as down Counter

**Note :** For n-bit Counter if up Counting value is P then down Counting value is '$2^n - P$'
(or)
Apply 2's comp to up Counting value.

| Clock | Next FF Clocks | Counter |
|---|---|---|
| -ve edge | $Q_i$ | Up |
| -ve edge | $\overline{Q_i}$ | Down |
| +ve Edge | $Q_i$ | Down |
| +ve edge | $\overline{Q_i}$ | Up |

## Synchronous Counter : (D FF's were used)

### ① Ring Counter : (Also called as n:1 Counter)




$Q_A = Q_B = Q_C = 0$
Not self
Not free

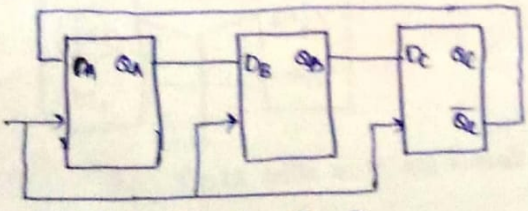
Initially $Q_A = 1, Q_B = 0, Q_C = 0$

| clk | $D_A$ | $D_B$ | $D_C$ | $Q_A$ | $Q_B$ | $Q_C$ |
|---|---|---|---|---|---|---|
| 0 | – | – | – | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 | 1 | 0 | 0 |

In Ring Counter
$D_A = Q_C$
$D_B = Q_A$
$D_C = Q_B$

n bit sync ring counter repeats after n clocks

### ② Johnson Counter ($2n$:1 Counter) (Twisted ring counter)




Initially $Q_A = Q_B = Q_C = 0$

Self
Not free

| clk | $D_A$ | $D_B$ | $D_C$ | $Q_A$ | $Q_B$ | $Q_C$ |
|---|---|---|---|---|---|---|
| 0 | – | – | – | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 1 | 1 | 0 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 1 | 0 | 1 | 1 |
| 5 | 0 | 0 | 1 | 0 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |

$D_A = \overline{Q_C}$ ; $D_B = Q_A$ ; $D_C = Q_B$

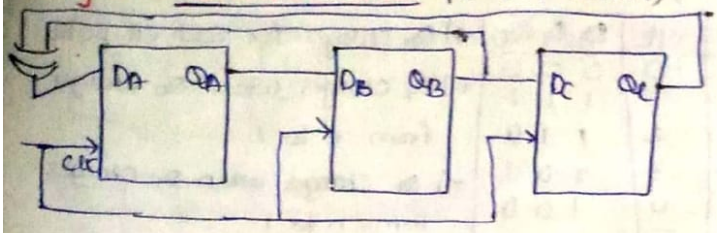
n bit johnson counter repeats after 2n clock pulses

| Note : Counter | MOD | Also called | o/p freq |
|---|---|---|---|
| Async (Ripple) [n bit up/down] | $2^n = N$ | $2^n$:1 | $f_0 = \frac{f_i}{2^n}$ |
| Ring Counter | $n = N$ | n:1 | $f_0 = \frac{f_i}{n}$ |
| Johnson Counter (twisted ring) | $2n = N$ | 2n:1 | $f_0 = \frac{f_i}{2n}$ |

**Note :** By default when Counter Name is not mentioned we take $N \leq 2^n$

x—The end—x

## ③ Shift Counter ($2^n - 1$ : 1 Counter)




Initially $Q_A = 1$ $Q_B = 0$ $Q_C = 1$

n-bit shift counter repeates after $2^n - 1$ CLKs

$D_A = Q_B \oplus Q_C$ ; $D_B = Q_A$ ; $D_C = Q_B$

| CLK | $D_A$ | $D_B$ | $D_C$ | $Q_A$ | $Q_B$ | $Q_C$ |
|---|---|---|---|---|---|---|
| 0 | - | - | - | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 1 | 1 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 0 | 0 | 1 | 0 |
| 7 | 1 | 0 | 1 | 1 | 0 | 1 |

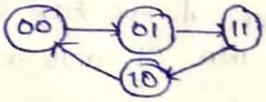
$Q_A = Q_B = Q_C = 0$

Not self
Not free

$f_0 = \frac{f_i}{2^n - 1}$

### Design of sync counter :

- get the state diagram of counting sequence
- identify num of FF's and type of FF's
- Get state table from state diagram and replace next state using excitation table of flipflops
- Get the i/p expr and realize them

Ex :- 1) 

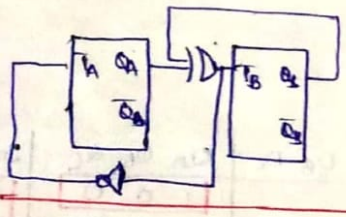

2) FF's = 2 ; Type = T

3)

| PS $Q_A$ $Q_B$ | NS $Q_{AN}$ $Q_{BN}$ | $T_A$ | $T_B$ |
|---|---|---|---|
| 0 0 | 0 1 | 0 | 1 |
| 0 1 | 1 1 | 1 | 0 |
| 1 1 | 1 0 | 0 | 1 |
| 1 0 | 0 0 | 1 | 0 |

$T_A = Q_A' Q_B + Q_A Q_B' = \overline{Q_A \oplus Q_B}$

$T_B = Q_A' Q_B' + Q_A Q_B = \overline{Q_A \oplus Q_B}$

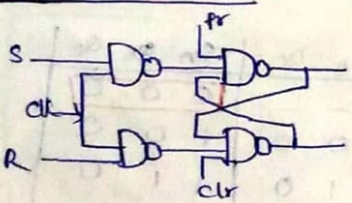



**Note :**

Self starting counter : Ensures counting whatever may be the initial state

Free running counter : If counter contains all possible states in closed path.

### Preset and clear :




| Pr | clr | Action |
|---|---|---|
| 0 | 0 | Not valid |
| 0 | 1 | set |
| 1 | 0 | reset |
| 1 | 1 | No effect |