→ Codes - It is representation of no / letters / words by a special grp of symols.
eg:- MORSE CODE - series of dots & dashes . . . - - -

→ BCD (Binary coded Decimal). (8421)
In this code, each decimal digit is represented by a 4-bit binary no.
Positional weights are 8 - 4 - 2 - 1

Decimal	BCD (8 4 2 1)
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

4 bits are necessary in order to cover 8 & 9.

as the bits are 4 ∴ combination are $2^4 = 16$

10	1010	decimal no. not decimal digits
11	1011	
12	1100	
13	1101	
⋮	⋮	

(Invalid / Forbidden Grps)

eg:- 0111 1010 1100 (1010 and 1100 marked ×)

- Conversion of decimal no to BCD.

(i) $(17)_{10}$ → 0001 0111

(ii) $(10)_{10}$ → 0001 0000

(iii) 1 0100 BCD → $(14)_{10}$

(iv) 1001001 BCD → $(49)_{10}$

Scanned with CamScanner

• Advantage

1. easy to code & decode.
2. 7 segment display.

• Disadvantage

1. code density is high.
2. To change a no. we have to change more no. of bits.

* To overcome this method, we have to go for grey code.

→ BCD Addition (simply Binary Addition)

1. Sum ≤ 9, Final carry = 0 ---- Ans is correct.
2. Sum ≤ 9, Final carry = 1, add 6 = 0110
3. Sum ≥ 9, Final carry = 0, add 6 = 0110.

	C
0 + 0 = 0	0
0 + 1 = 1	1
1 + 1 = 0	0

Q $2_{10} + 6_{10}$

```
  (1)(1)
   0010
  +0110
  ------
   1000   → 8₁₀
```

Q $3_{10} + 7_{10}$

```
  (1)(1)(1)
   0011
  +0111
  ------
   1010   > 9 ∴ not BCD
  +0110
  ------
  10000
```

0001 0000 = 10_{10}

Scanned with CamScanner

Q. $8_{10} + 9_{10}$

```
  1000
①1001
 10001
+ 0110
 10111
```

0001 0111
↳ 17

Q. $57_{10} + 26_{10}$

```
  0101  0111
  0010  0110
  0111  1101
      + 0111
      ①0100
```

```
  0111
+    1
  1000
```

→ 1000 0010
↳ 83

→ 2-4-2-1 code

Decimal digit	Set I 2-4-2-1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0 (0*4×1 + 2×1 + 1)
7	0 1 1 1
8	1 1 1 0
9	1 1 1 1

Scanned with CamScanner

→ Excess - 3 code (Decimal → 8-4-2-1 code (BCD) $\xrightarrow{\text{Add } 0011}$ Excess-3)

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

Q. xs-3 code for decimal no.

(ii) 24.

```
  (1)
  0010   0100
  0011   0011
  0101   0111   → 57
```

(iii) 658

```
  (1)(1)   (1)(1)(1)
  0110     0101     1000
  0011     0011     0011
  1001     1000     1011   → 98 (x) 11
```

→ Grey Code :- or Reflected Binary Code / Cyclic code.

- unweighted code
- unit distance code & minimum error code.
- 2 successive values differs in a single 1 bit
- Binary no. is converted to g-c to reduce switching operation 3↔4 3bit change.

Scanned with CamScanner

Decimal	Binary	grey-code
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	1000
15	1111	0000
16	10000	

- Applications of Grey Code -

Scanned with CamScanner

IC works on +5V

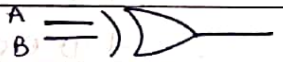
→ LOGIC GATES AND BOOLEAN ALGEBRA :-
These are basic building blocks of any digital circuit
An electronic device that gives logical output based on input.

& More than 1
- OR — $Y = A + B$ — IC = 7432
- AND — $Y = A \cdot B$ — IC = 7408

only 1.
- NOT — $Y = \overline{A}$ — IC = 7404.

- NOR — $Y = \overline{A+B}$ — IC = 7402 — acts as NOT Gate if B is short circuit
- NAND — $Y = \overline{A \cdot B}$ — IC = 7400

XOR (Ex-OR) exclusive OR. IC = 7486.

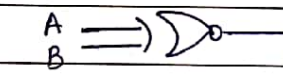


$Y = A \oplus B$

Can be more than 2.

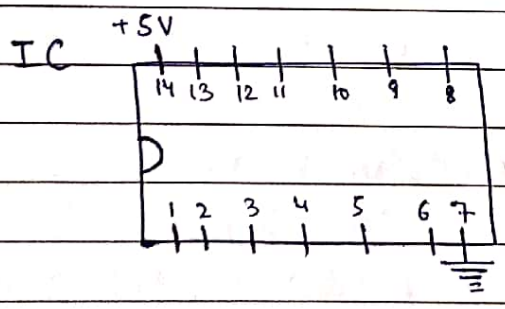
A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

XNOR (Symmetric Gate). IC = 740



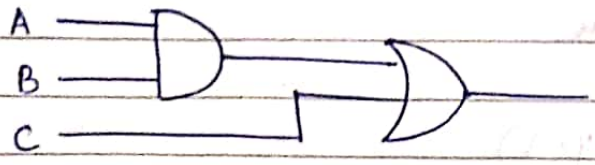
$Y = A \odot B$

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

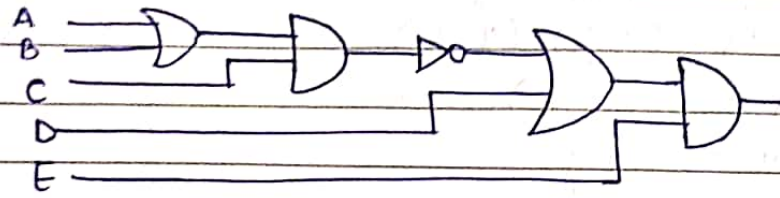


14 pins in IC.

Scanned with CamScanner

1. 

$\equiv (A \cdot B) + C$.

2. 

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

- **Boolean Algebra** - George Boole (1854).
  mathematics of digital logic is called boolean algebra.
  different from arithmetic algebra ∵ only 2 values — 0's 1's

- **Boolean Variable** - any value which can pass either 0 or 1.
  eg:- x, A, W.

- **literals** - compliment of boolean variable or without
  eg:- $x, \bar{x}, A, \bar{A}$.

- **Boolean Expression** - used to express relation b/w a logic visuals Input & outputs.
  eg:- $Y = \overline{A+B}$ $F = \overline{xyz} + y$.

- **Boolean Theorem**:-

1. Single Value Theorem (Intersection). + (Union).

(i) $x \cdot 0 = 0$
(ii) $x \cdot 1 = x$
(iii) $x \cdot x = x$
(iv) $x \cdot \bar{x} = 0$
(v) $x + \bar{x} = 1$
(vi) $x + 0 = x$
(vii) $x + 1 = 1$
(viii) $x + x = x$

Demorgan Law
$\overline{A \cdot B} = \bar{A} + \bar{B}$
$\overline{A+B} = \bar{A} \cdot \bar{B}$
$\overline{A \cdot B \cdot C} = \bar{A} + \bar{B} + \bar{C}$

Scanned with CamScanner

2. Multiple Variable Theorem.

- (i) $x+y = y+x$ — commutative
- (ii) $(x+y)+z = x+(y+z)$ — associative
  $(x+y)\cdot z = (x\cdot z)+(y\cdot z)$ — distributive
  $(x\cdot y)+z = (x+z)\cdot(y+z)$
  $x+xy = x(1+y) = x$.
  $x+\bar{x}y = x+y$
  $\bar{x}+xy = \bar{x}+y$.

Q. Simplify the expression using De morgan law.

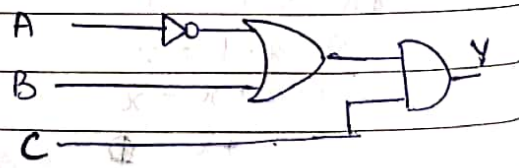
1. $Z = \overline{(\bar{A}+C)(B+\bar{D})}$
   $= \overline{(\bar{A}+C)} + \overline{(B+\bar{D})}$
   $= (A\cdot\bar{C}) + (\bar{B}\cdot D)$

2. $W = \overline{(A+\overline{BC})}$
   $= \bar{A}\cdot BC$

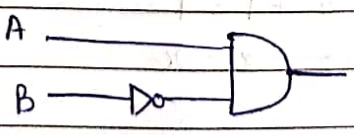
Q. Simplify the expression & implement using appropriate logic gate.

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

2. $Y = (\bar{A}+B)C + ABC$.
   $= \bar{A}C + BC + ABC$.
   $= \bar{A}C + BC(1+A)$.
   $= \bar{A}C + BC$
   $= C(\bar{A}+B)$



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



Scanned with CamScanner

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

5. $X = ACD + \bar{A}BCD$
$= ACD(A + B\bar{A})$.
$= CD(A + B)$.

6. $Y = \overline{A}BC\bar{D} + \overline{A}BCD$.
$= \overline{AB}D(C + \bar{C})$
$= \overline{AB}D$.

7. $Y = \bar{A}D + ABD$.
$= D(\bar{A} + AB)$.
$= D(A + B)$.

- Canonical or Standard form of boolean expression :-
$F_1 = xyz$
$F_2 = xy + y\bar{z}$ → boolean function.

Truth Table Representation.

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

Complement of function → $F \Rightarrow \overline{F}$
$F = \bar{x}y\bar{z} + \bar{x}yz \Rightarrow \overline{F} = (x + \bar{y} + z) \cdot (x + y + \bar{z})$

Dual of function $\cdot \longrightarrow +$ $+ \longrightarrow \cdot$

- Minterms
For a 2 variable x, y.
The AND outputs may be xy, $\bar{x}y$, $x\bar{y}$, $\bar{x}\bar{y}$

Scanned with CamScanner

- **Sum of Product Form. (SOP).**

$F_1 = xy + y\bar{z}$ $\qquad$ $F_2 = x + xy$

- **Standard SOP form.**

$F_1 = x + xy$ $\qquad$ × S.SOP.

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

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

$= x\bar{y} + xy$

$F_2 = xy + y\bar{z}$

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

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

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

- **Maxterms**

2 variables: x, y $\qquad$ OR ng the outputs give

$x+y$, $\bar{x}+\bar{y}$, $\bar{x}+y$, $x+\bar{y}$

- **Product of Sum (POS):-**

$F_1 = (x+y)(x+\bar{y})$.

$F_2 = x(\bar{x}+\bar{y})$.

- **Standard POS** $\qquad$ $F_2 = x(\bar{x}+\bar{y})$

$= (x+0)(\bar{x}+\bar{y})$

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

x	y	z	minterms.	maxterm	F_1	F_2
0	0	0	$\bar{x}\bar{y}\bar{z}$	$\bar{x}+\bar{y}+\bar{z}$	0	0
0	0	1	$\bar{x}\bar{y}z$	$\bar{x}+\bar{y}+z$	1	0
0	1	0	$\bar{x}y\bar{z}$	$\bar{x}+y+\bar{z}$	0	0
0	1	1	$\bar{x}yz$	$\bar{x}+y+z$	0	1
1	0	0	$x\bar{y}\bar{z}$	$x+\bar{y}+\bar{z}$	1	0
1	0	1	$x\bar{y}z$	$x+\bar{y}+z$	0	1
1	1	0	$xy\bar{z}$	$x+y+\bar{z}$	0	1
1	1	1	xyz	$x+y+z$	1	1

Scanned with CamScanner

- **Conversion of SOP to POS.**

$F_1 = \bar{x}\bar{y}\bar{z} + x\bar{y}z + xyz = m_1 + m_4 + m_7 = \sum(1, 4, 7)$.

$\bar{F}_1 = \overline{xyz} + \bar{x}y\bar{z} + \bar{x}yz + xyz + xy\bar{z}$

$\bar{\bar{F}}_1 = (x+y+z) \cdot (x+\bar{y}+z) \cdot (x+\bar{y}+\bar{z}) \cdot (\bar{x}+y+\bar{z}) \cdot (\bar{x}+\bar{y}+z)$.

Conversion b/w SOP & POS form.

SOP → $F_1 = \sum(0, 1, 2, 4, 5)$.

$\bar{F}_1 = \sum(3, 6, 7)$

POS. → $\bar{\bar{F}}_1 = \pi(3, 6, 7)$

$F_1 = \pi(3, 6, 7)$.

$(x+\bar{y}+\bar{z})(\bar{x}\bar{y}+z)(\bar{x}+\bar{y}+\bar{z})\bar{F}_1 = \pi(0, 1, 2, 4, 5)$.

$\bar{\bar{F}}_1 = \sum(0, 1, 2, 4, 5)$.

Q. Convert in another form

$F(x, y, z) = \sum(1, 3, 7)$ $\quad F(x, y, z) = \pi(0, 1, 2, 3, 4, 6, 12)$.

Q. Express the following in SOP & POS form.

$F(x, y, z, w) = \bar{y}w + \bar{x}w + yw$.

Q. Express in SPOS & SSOP.

$F_2 = xy + \bar{x}z$.

$F_3 = x + \bar{x}y + z$

$F_4 = y\bar{z} + xy\bar{z}$

Q. For the given truth table, minimize the SOP expression.

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

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

$= B$.

Simplify the expression for

$Y(A, B) = \sum_m(0, 2, 3)$.

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

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

$= \bar{B} + A$

Scanned with CamScanner

Q. For the given truth table, minimize the POS expression

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

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

Or $Y = \Sigma(M_0, M_2)$

$\bar{Y} = \Sigma(M_1, M_3)$

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

Q. Express the following in SOP & POS form.

$F(x,y,z,w) = \bar{y}w + \bar{x}w + yw$

$F(x,y,z) = (xy+z)(xz+y)$.

Sol:-

$F_1 = x+\bar{y}z$

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

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

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

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

$F_2 = xy + \bar{x}z$

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

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

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

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

$F_3 = x+\bar{x}y\cdot z$ } convert in standard POS → SOP

$F_4 = y\bar{z} + xyz$

Scanned with CamScanner

Binary system - 0, 1
trivial system - 0, 1, 2
quadraple system - 0, 1, 2, 3
octal system - 0, 1, 2, 3, 4, 5, 6, 7.
Hexadecimal system - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Decimal system - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

→ **Simplification of boolean function:-**

→ **K-Map method**

- provides a systematic method of boolean function.
- K-map is a map made of squared n-variable will have $\underbrace{2^n}_{\downarrow}$ minterms
  2^n no. of squares.

(i) **2-variable map**

$n = 2 \quad 2^n = 2^2 = 4 \longrightarrow$ 4 minterms 4 squares

x \ y	0	1
0	$\bar{x}\bar{y}$	$\bar{x}y$
1	$x\bar{y}$	xy

eg:

x \ y	0	1
0		
1	1	1

$F = x\bar{y} + xy.$

x \ y	0	1
0		1
1	1	1

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

Scanned with CamScanner

(iii) 3- Variable Maps:-

$n = 3$ $\quad 2^n = 2^3 = 8$ minterms 8 squares.

x \ yz	00	01	11	10
0	$\bar{x}\bar{y}\bar{z}$ [0]	$\bar{x}\bar{y}z$ [1]	$\bar{x}yz$ [3]	$\bar{x}y\bar{z}$ [2]
1	$x\bar{y}\bar{z}$ [4]	$x\bar{y}z$ [5]	xyz [7]	$xy\bar{z}$ [6]

eg:- map $F = \Sigma_1(0,1,2,3)$ on k-Map.

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

$= x$.

eg:- map $F = x + z$ on k-Map.

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

eg:- $F = \bar{y}z$.

x \ yz	00	01	11	10
0		1		
1		1		

eg:- $F = \bar{y}$

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

→ Grouping the 1's

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

$yz + x + x\bar{y}\bar{z}$

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

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

Scanned with CamScanner

Q. Simplify the boolean function :-

(i) $F = \Sigma(0, 2, 3, 4, 5)$

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

$\Rightarrow \bar{y}\bar{z} + \bar{x}y + x\bar{y}$

(ii) $F = \Sigma(0, 2, 4, 5, 6)$

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

$G_1 = (\bar{x}\bar{y}\bar{z}, x\bar{y}\bar{z}, \bar{x}y\bar{z}, xy\bar{z})$
↳ wrapping around.

$F = \bar{z} + x\bar{y}$

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

x \ yz				
	0	1	3	2
	4	5	7	6

$F = z + \bar{x}y$

Problems

$F(x, y, z) = \Sigma(3, 5, 6, 7)$

$F(x, y, z) = \Sigma(0, 1, 2, 5, 7)$

$F(x, y, z) = xy + \bar{x}yz + \bar{x}y\bar{z}$

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

(iii) **4 - variable Map :-**

$n = 4$, $2^n = 2^4 = 16$ => 16 minterms

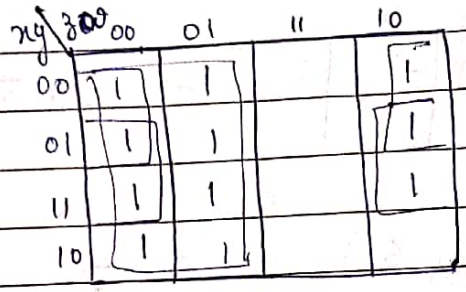
16 Squares

xy \ zw	00	01	11	10
00	0 $\bar{x}\bar{y}\bar{z}\bar{w}$	1 $\bar{x}\bar{y}\bar{z}w$	3 $\bar{x}\bar{y}zw$	2 $\bar{x}\bar{y}z\bar{w}$
01	4 $\bar{x}y\bar{z}\bar{w}$	5 $\bar{x}yzw$	7 $\bar{x}yzw$	6 $\bar{x}y\bar{z}w$
11	12 $xy\bar{z}\bar{w}$	13 $xy\bar{z}w$	15 $xyzw$	14 $xyz\bar{w}$
10	8 $x\bar{y}\bar{z}\bar{w}$	9 $x\bar{y}\bar{z}w$	11 $x\bar{y}zw$	10 $x\bar{y}z\bar{w}$

Scanned with CamScanner

Q. Simplify the boolean function

(ii) $F = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

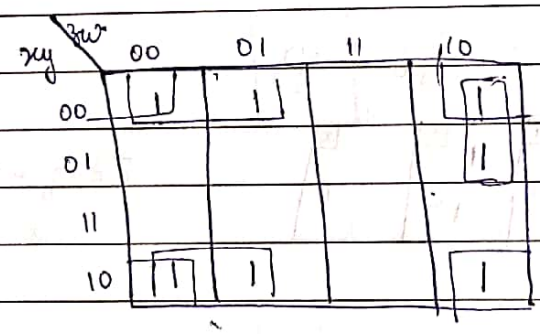


$F = \bar{z} + x\bar{w} + \bar{w}y$

(iii) $F = \overline{wxy} + \bar{x}y\bar{z} + \bar{w}xy\bar{z} + w\overline{xy}$

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

$= \overline{w\bar{x}yz} + \overline{wxyz} + w\bar{x}y\bar{z} + \overline{w\bar{x}y\bar{z}} + \bar{w}xy\bar{z} + w\bar{x}\bar{y}z + wxy\bar{z}$



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

$= \overline{wxy} + \bar{x}y\bar{z} + \bar{w}xy\bar{z} + w\overline{xy}$

$= (w+w)\overline{xy} + yz(\bar{x} + \bar{w}x)$

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

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

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

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

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

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

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

$F = \overline{wxyz} + w\bar{x}yz + w\overline{xy}z + \bar{w}yz + wx\bar{y}z$.

Scanned with CamScanner

(iv) **5 - variable Map :-**

ABCDE

n = 5 no. of minterms = $32 = 2^5$.

" " square = $2^5 = 32$.

A = 0

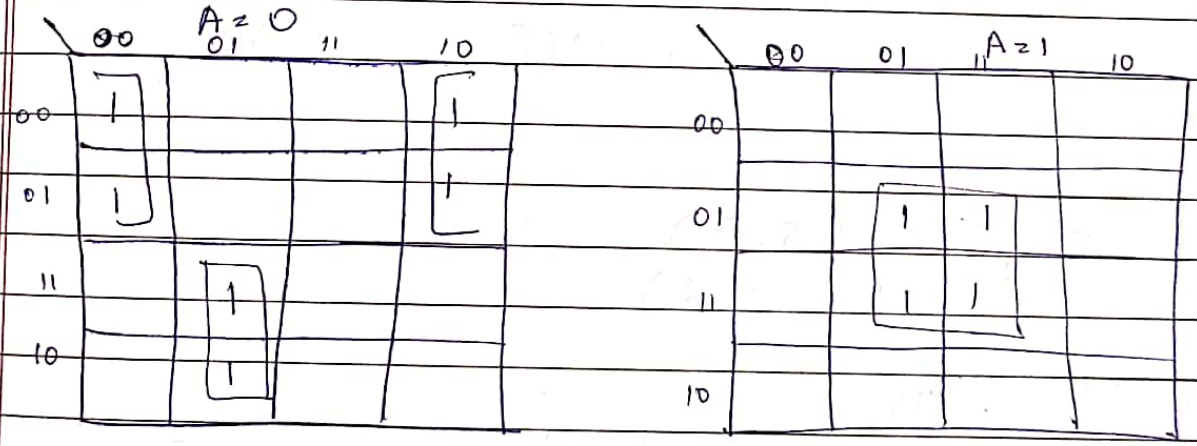
BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

A = 1.

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

Q. Simplify the function

$F(A,B,C,D,E) = \Sigma(0,2,4,6,9,13,21,23,29,31)$.



$F = \overline{ABE} + \bar{A}B\bar{D}E + ACE$

Scanned with CamScanner

⇒ POS Simplification using K-Map :-

F ⟶ 1's

$\bar{F}$ ⟶ 0's combine.
(using de morgan's)
$\bar{\bar{F}}$ ⟶ in POS form.

Q: Simplify the function in POS form :-

(i) $F(A, B, C, D) = \sum (0, 1, 2, 5, 8, 9, 10)$

AB \ CD	00	01	11	10
00	1 (0)	1 (1)	0 (3)	1 (2)
01	0 (4)	1 (5)	0 (7)	0 (6)
11	0 (12)	0 (13)	0 (15)	0 (14)
10	1 (8)	1 (9)	0 (11)	1 (10)

$\bar{F} = CD + AB + B\bar{D}$

~~POS form~~

~~If we take complement of POS we get SOP~~

$\bar{\bar{F}} = \overline{(CD + AB + B\bar{D})}$

$= (\bar{C}+\bar{D})(\bar{A}+\bar{B})(\bar{B}+D)$. ⟶ POS form.

(ii) $F = \prod (0, 2, 5, 7)$.
$\updownarrow$
$= \sum (1, 3, 4, 6)$.

SOP

x \ yz	00	01	11	10
0	0	1 (1)	1 (3)	2
1	1 (4)	5	7	1 (6)

$F \longrightarrow \bar{x}z + x\bar{z}$

POS

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

$\bar{F} = xz + \bar{x}\bar{z}$

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

↳ POS form

Scanned with CamScanner

Q. Use K-Map to minimize the following expressions.

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

→ **Don't Care in K-Maps :-**

↳ ϕ

eg:- $F(x,y,z) = \sum(0,1,2,6) + \phi(4,5)$

↳ don't care on 4, 5. i.e you can keep it as 1 or 0 acc to the need

x \ yz	00	01	11	10
0	1	1		1
1	X	X		1

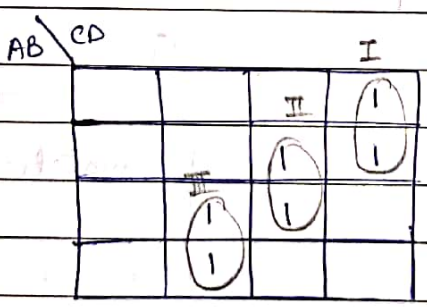
Here we will keep X X as 1.

SOP $\xrightarrow{\phi}$ 1

POS $\xrightarrow{\phi}$ 0

$$F = \bar{y} + \bar{z}$$

- **Implicants** :- The group of 1's like 2, 4, 8, - - -
- **Prime Implicant** :- The largest possible group of 1's.
- **Essential Prime Implicant** - The largest group of 1's in which there is a single 1 that can't be combined with any other 1's.



Implicant → I, II, III

P. I → I, II, III.

Essential P.I → I, III

∵ II can be grouped like

Scanned with CamScanner

=> **The QUINE - McCLUSKEY Method:-**

↳ useful for variable ≥ 5.

eg:- $F(w,x,y,z) = \Sigma(2,6,8,9,10,11,14,15)$.

	w x y z	F
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	1 1
3	0 0 1 1	0
4	0 1 0 0	0
5	0 1 0 1	0
6	0 1 1 0	1 2
7	0 1 1 1	0
8	1 0 0 0	1 1
9	1 0 0 1	1 2
10	1 0 1 0	1 2
11	1 0 1 1	1 3
12	1 1 0 0	0
13	1 1 0 1	0
14	1 1 1 0	1 3
15	1 1 1 1	1 4

Step-1 → Grouping of 1's in ascending order of No. of 1's.

Group	Minterms	w x y z
G_1	2	0 0 1 0
	8	1 0 0 0
G_2	6	0 1 1 0
	9	1 0 0 1
	10	1 0 1 0
G_3	11	1 0 1 1
	14	1 1 1 0
G_4	15	1 1 1 1

Step-2:- Merging of minterms if match.

Scanned with CamScanner

Group	Minterms Pairs.	w x y z
G_1	2, 6	0 _ 1 0
	2, 10	_ 0 1 0
	8, 9	1 0 0 _
	8, 10	1 0 _ 0
G_2	6, 14	_ 1 1 0
	9, 11	1 0 _ 1
	10, 11	1 0 1 _
	10, 14	1 _ 1 0
G_3	11, 15	1 _ 1 1
	14, 15	1 1 1 _

Step - 3 :- Merging of minterm pairs.

Group	Minterm pairs	w x y z
G_1	2, 6, 10, 14	_ _ 1 0
	2, 10, 6, 14	_ _ 1 0 × Redundant
	8, 9, 10, 11	1 0 _ _ groups
	8, 10, 9, 11	1 0 _ _ × can be removed
G_2	10, 11, 14, 15	1 _ 1 _
	10, 14, 11, 15	1 _ 1 _ ×

Since further no grouping is possible, we write are prime implicants as → yz , $w\bar{x}$, wy.

Essential prime implicants →

Scanned with CamScanner

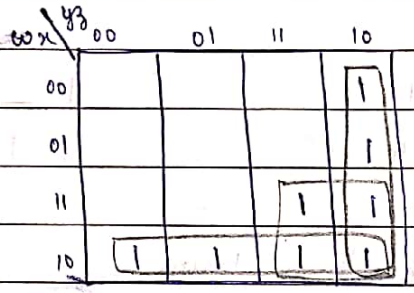
Essential Prime Implicant →

P.I	Minterms	2	6	8	9	10	11	14	15
yz'	2, 6, 10, 14	(1)	(1)			1		1	
wx'	8, 9, 10, 11			(1)	(1)	1	1		
wy	10, 11, 14, 15.					1	1	1	(1)

Since all the 3 are necessary to cover all P.I.

∴ essential implicants are → yz', wx', wy.

→ To verify with K-Map :-

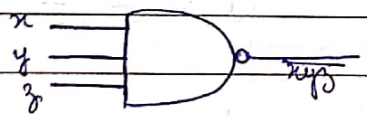


→ $wx' + yz' + wy$

⇒ Implementation of Boolean Algebra :-

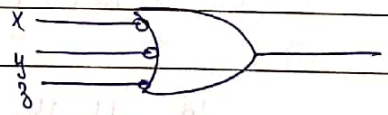
1. NOR / NAND Implementation :-

• NAND.



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

AND - Invert

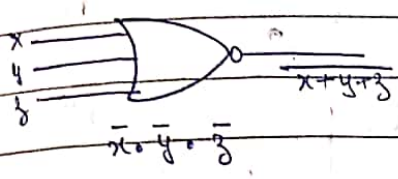


$\bar{x} + \bar{y} + \bar{z}$

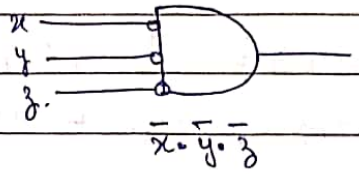
Invert - OR.

Scanned with CamScanner

- NOR

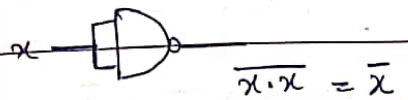


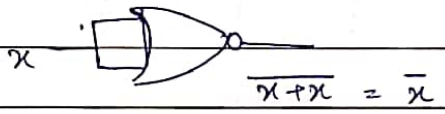
OR - Invert



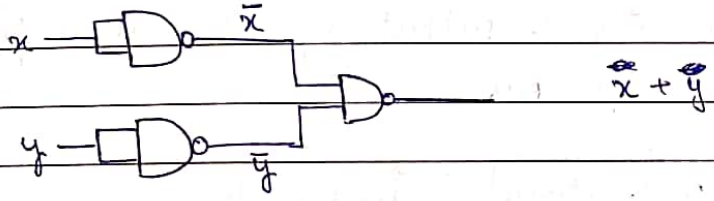
Invert - OR.

- NOT gate using NAND | NOR



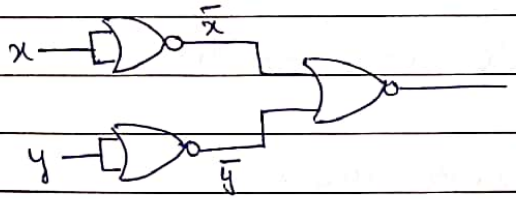


- OR gate using NAND

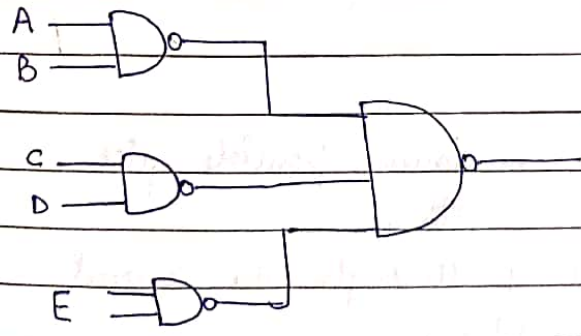


$F = \overline{\bar{x}\cdot\bar{y}} = x + y$

- AND gate using NOR.



Q. $F = AB + CD + E$ using only NAND GATE



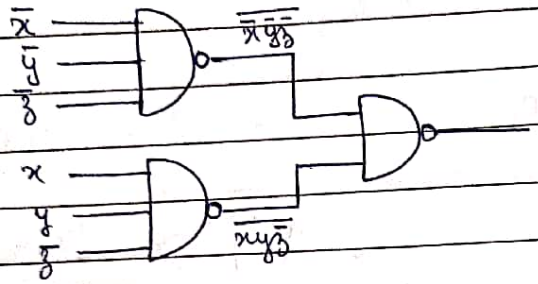
Scanned with CamScanner

Q. Implement using NAND/NOR.

(i) $F(x,y,z) = \Sigma(0,6)$

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

x \ yz	00	01	11	10
0	0	1	3	2
1	4	5	7	6



$$F = \overline{(\overline{\bar{x}\bar{y}\bar{z}} \cdot \overline{xy\bar{z}})}$$

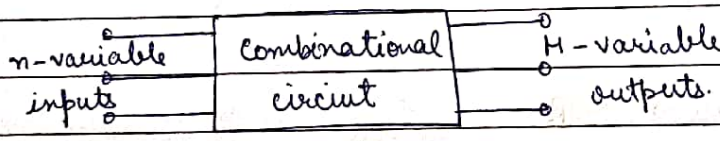
$$= (\overline{xyz} \cdot \overline{\bar{x}\bar{y}\bar{z}})$$

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

⇒ **Combinational Logic :-**

↳ depends only on the current input.

It consists of logic gates & output depends only on present input & not on the previous inputs.



→ **Design procedure for combination logic circuits :-**

Step 1 :- Problem Statement

Step 2 :- Input & output variable assignment

Step 3 :- Truth table describing the input/output relation.

Step 4 :- Simplification of boolean function

Step 5 :- logic implementation using logic gates.

- **Constraints :-**

1. Our design should have minimum possible gates.
2. minimum no. of input of gates.
3. minimum propagation delay through the circuit.
4. minimum no. of interconnections.

Scanned with CamScanner

(1)(1)		
1 1 0 0	12	
0 1 0 1	5	
(1) 0 0 0 1	17	

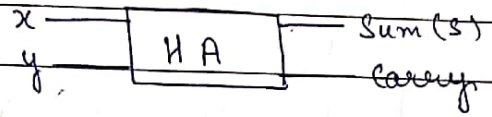
classmate
Date
Page

5. Limitation of driving capability of gate.

→ ADDER :-

ii) Half - Adder

can add 2 bits 0 & 1.

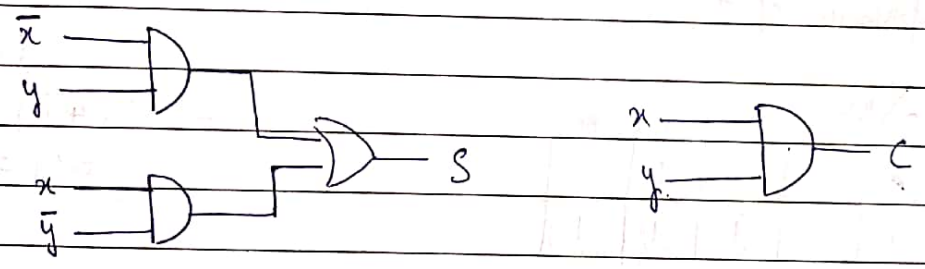


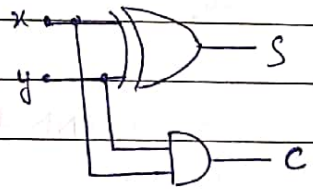
x	y	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

$S = x\bar{y} + \bar{x}y$

$= x \oplus y$ XOR Gate

$C = xy$ AND Gate

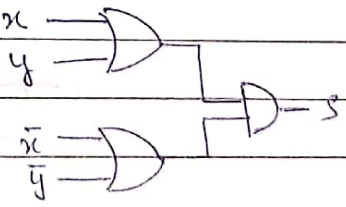


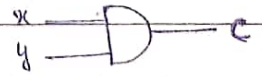


$S = \bar{x}y + x\bar{y} + x\bar{x} + y\bar{y}$

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

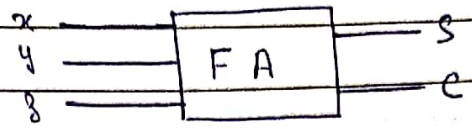
$= (x+y)(\bar{x}+\bar{y})$





Scanned with CamScanner

iii) Full - Adder ---- using 2 HA to make FA



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

expression of sum

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

$S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

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

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

$= z \oplus (x \oplus y)$

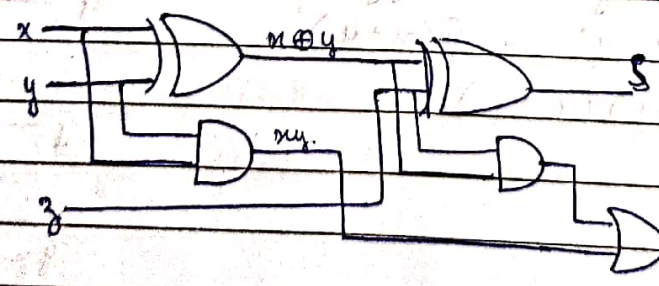
expression of C.

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

$C = yz + xz + xy$

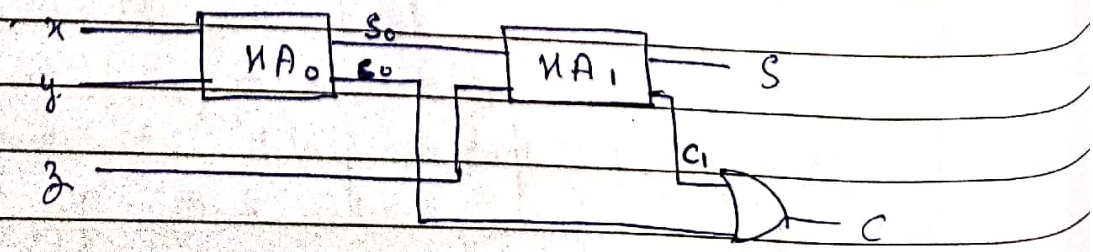
S can be done using 4 AND & 1 OR Gate

C " " " 3 " " " "



$S = \overset{1}{(x \oplus y)} \quad \overset{2}{z \oplus (x \oplus y)}$

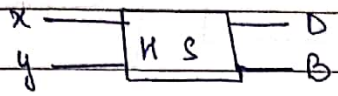
$C = \overset{1}{xy} \quad \overset{2}{z(x \oplus y)}$



Scanned with CamScanner

SUBTRACTER :-

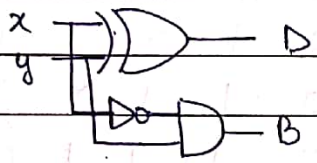
Half - subtracter -



x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$D = x\bar{y} + \bar{x}y = x \oplus y$

$B = \bar{x}y$



Full Subtracter -

x	y	z	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

using k - map

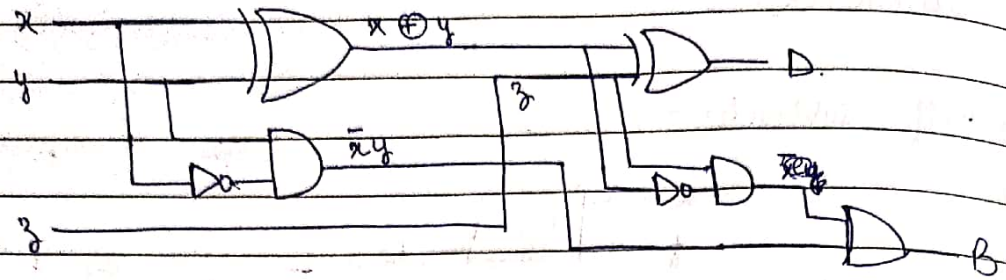
$D = (x \oplus y) \oplus z$

B =

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

$B = \bar{x}z + \bar{x}y + yz$

Scanned with CamScanner



→ **BCD to Ex-3 Code Conversion :-**

Input BCD				Output excess 3			
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
Invalid BCD Number	1	0	1	0	x	x	x	x
	1	0	1	1	x	x	x	x
	1	1	0	0	x	x	x	x
	1	1	0	1	x	x	x	x
	1	1	1	0	x	x	x	x
	1	1	1	1	x	x	x	x

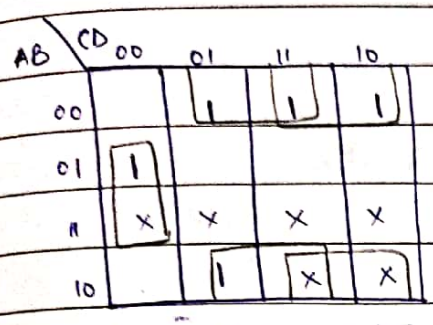
expression for w

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

$w = A + BD + BC$ ←

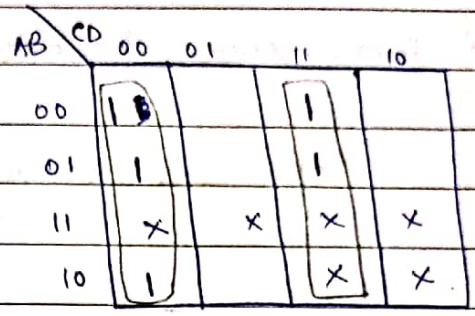
Scanned with CamScanner

expression for x



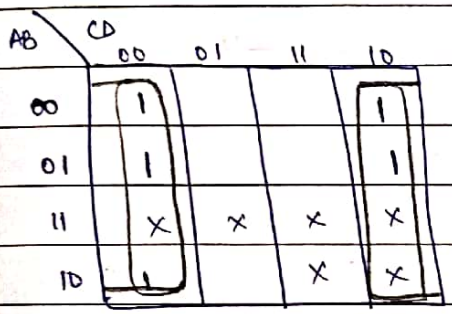
$x = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$

expression for y.



$y = \bar{C}\bar{D} + CD$

expression for z.



$z = \bar{D}$

$z = \bar{D}$

$y = \bar{C}\bar{D} + CD$

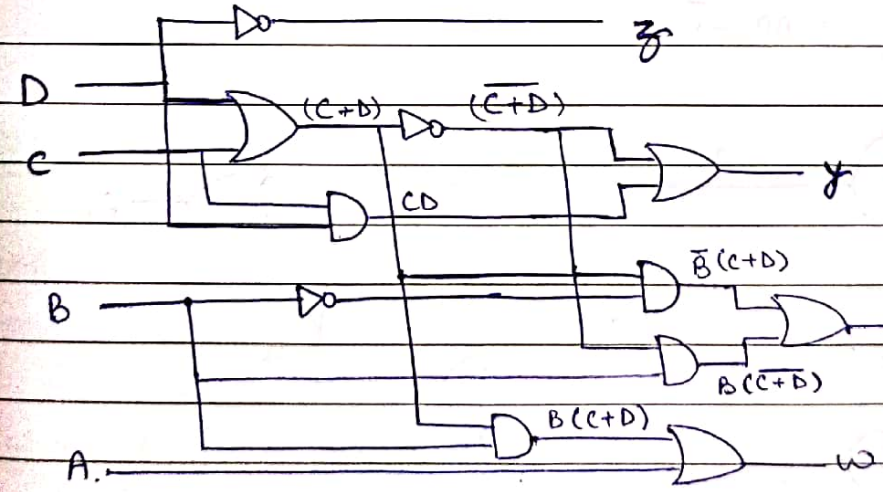
$= (\overline{C+D}) + CD$

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

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

$w = A + BD + BC$

$= A + B(C+D)$



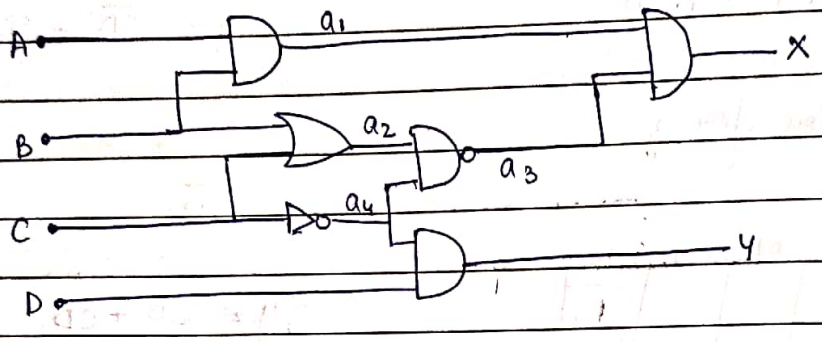
Q. Convert Excess-3 to BCD

Scanned with CamScanner

⇒ Analysis of Combination Circuit :-

→ Reverse process of logic design -

Logic circuit → [Analysis] → logic function. (O/P in terms of I/P)



$X = a_1 a_3$

$Y = D a_4$

$a_1 = AB$

$a_2 = B + C$

$a_3 = \overline{a_2 a_4}$

$a_4 = \bar{C}$

$X = (AB)(\overline{a_2 a_4})$

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

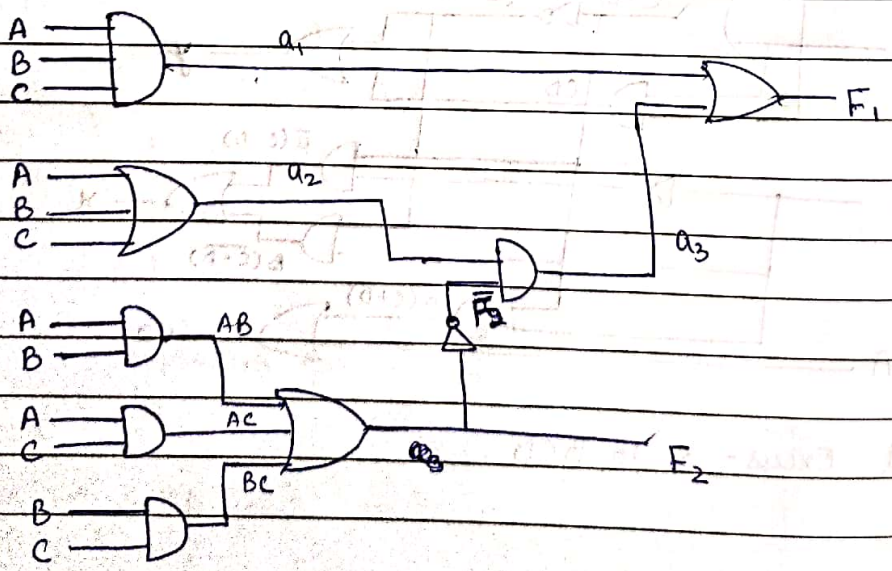
$= AB \; \overline{B\bar{C}}$

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

$Y = D\bar{a}_4$

$= D\bar{C}$

Q.



Scanned with CamScanner

$F_1 = a_1 + a_3$ $\quad a_1 = ABC$ $\quad a_2 = A+B+C$

$F_2 = AB + AC + BC$ $\quad a_3 = a_2 \bar{F}_2$

$F_1 = ABC + a_2 \bar{F}_2 = ABC + (A+B+C)(\overline{AB+AC+BC})$

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

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

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

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

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

A	B	C	F_2	a_1	a_2	a_3	F_1
0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1
0	1	0	0	0	1	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	1	1
1	0	1	1	0	1	0	0
1	1	0	1	0	1	0	0
1	1	1	1	1	1	0	1

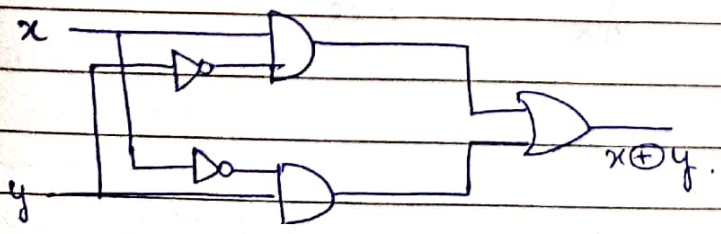
⇒ **Ex-OR Function :-**

XOR $\longrightarrow$ $x \oplus y = x\bar{y} + \bar{x}y$.

XNOR $\longrightarrow$ $\overline{x \oplus y} = xy + \bar{x}\bar{y}$

x \ y	0	1
0	0	1
1	1	0

⇒ **Implementation of Ex-OR :-**



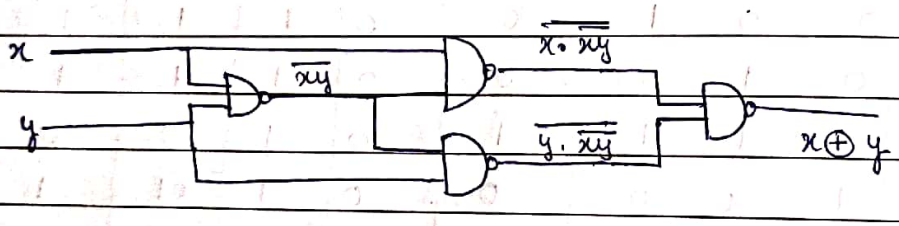
Scanned with CamScanner

→ NAND only Implementation

$$x \oplus y = x\bar{y} + \bar{x}y$$
$$= x\bar{y} + \bar{x}y + x\bar{x} + y\bar{y}$$
$$= x(\bar{x}+\bar{y}) + y(\bar{x}+\bar{y})$$
$$= (x+y)(\underbrace{\bar{x}+\bar{y}}_{\text{NAND}})$$
$$= (x+y)(\overline{xy})$$
$$= \overset{A}{(x \cdot \overline{xy})} + \overset{B}{(y \cdot \overline{xy})}$$
$$= \overline{\overline{x \cdot \overline{xy}} \cdot \overline{y \cdot \overline{xy}}}$$

NAND (of A and B) · NAND

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



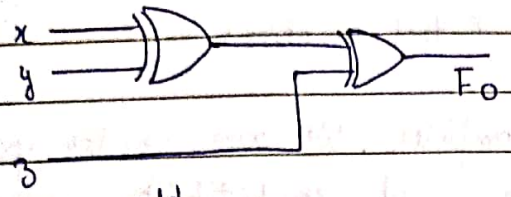
⇒ Odd / Even Function :-

If odd no. of 1's then odd function.
" even " " " " even " "

x	y	F_o	F_e
0	0	0 → even	→ 1
0	1	1 → odd	→ 0
1	0	1 → odd	→ 0
1	1	0 → even	→ 1
		↑ XOR	↑ XNOR

Scanned with CamScanner

→ 3 input odd function :-



odd.

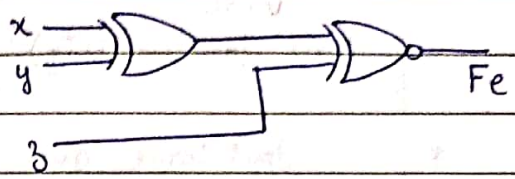
x \ yz	00	01	11	10
0		1		1
1	1		1	

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

~~skip 1 skip 1 ---~~

$F_o = \sum (1, 2, 4, 7)$

3 input even function :-



even

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

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

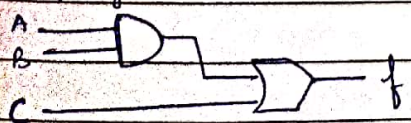
$F_e = \sum (0, 3, 5, 6)$

⇒ Hardware Descriptive Languages

↳ VHDL

→ Hardware Language (VHDL)

1. well defined structure
2. execution is concurrent
3. Case Insensitive
4. Synthesis & Emulate (simulation with hardware)
5. VHDL program ×



f = (A and B) or C

Scanned with CamScanner

Synthesis → VHDL (code functionality)

~~VHDL~~ Emulate → VHDL (~~test benches~~).

* we don't have any VHDL compiler. We have vendor specific Synthesizers that share some % of compatibility with VHDL.

⇒ VHDL

↳ VHSIC → Very high speed Integrated Circuit Hardware Descriptive Language.

- It is a language that allows to code the description of some hardware
- Design Automation.

Design Entry (VHDL)
↓
Synthesis
↓
Translate
↓
Mapping
↓
TapeOut (IC)

Automated

VHDL → Three fundamental Units

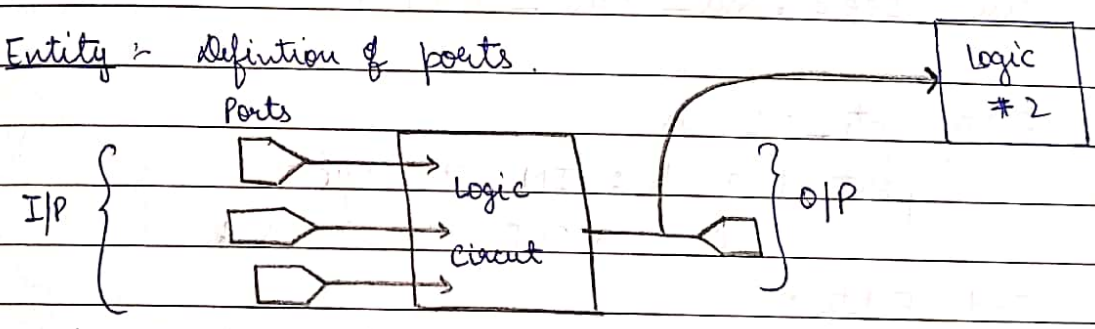
Library
Entity
Architecture

Scanned with CamScanner

1. Library :- It is a collection of commonly used peices of code.
- every library has few packages.
- each package supports some features of VHDL.

Library Ieee
01) std_logic_1164
- logical package (XOR, OR, NOR ----)
- Logical datatypes & operators.
- std logic [8-valued system]
02) std_logic_arith
- arithematic package
- arithematic datatypes & corresponding operators.
- signed & unsigned

2. Entity :- Difintion of ports.



- mode of the port (I/P or O/P) or Inout port.
- data type of port.

→ key word ←

Entity entity_name is
Port (port_name$_1$: Mode data-type ;
port_name$_2$: mode data-type);
End Entity ;

3. Architecture :- It is the part where we encode the functionality

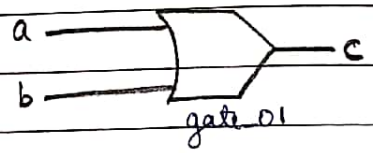
Scanned with CamScanner

Key word

Architecture behavior of entity-name is

Begin
    concurrent statement # 01 ;
    concurrent statement # 02 ;
    .
    .
    .
End Architecture ;

Q. Write a code for an OR Gate

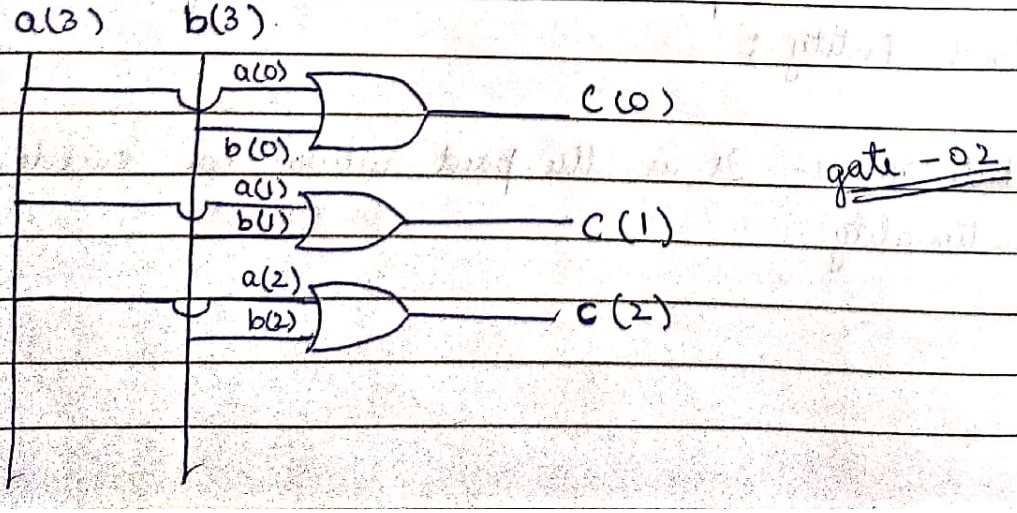


L
```
Library IEEE ;
Use Ieee . std_logic_1164 all ;
```

E
```
Entity gate_01 is
    port ( a , b : IN std_logic ;
           c     : OUT std_logic) ;
End Entity ;
```

A
```
Architecture behaviour of gate_01 is
Begin.                    assignment operator.
B.      c <= a OR b ;
End Architecture ;
```

Q.



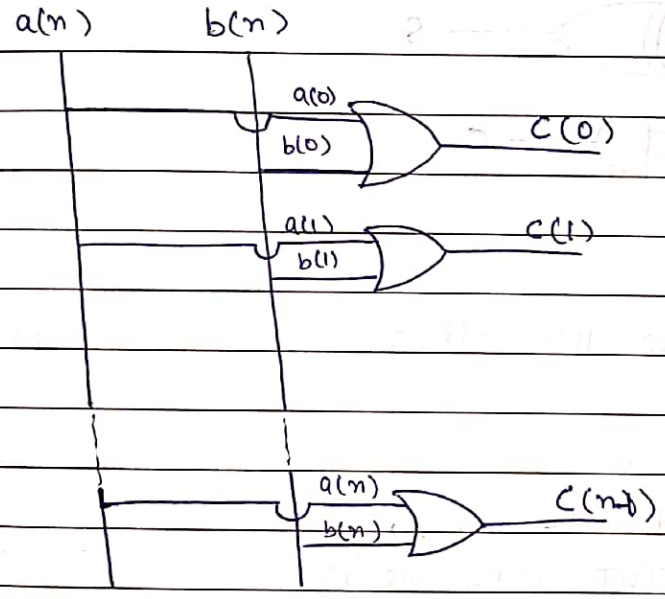
Scanned with CamScanner

→ Library IEEE;
Use IEEE Std_logic_1164.all;

Entity gate-02 is
port (a,b : IN Std_logic_vector (2, DOWNTO 0);
c : OUT Std_logic_vector (2, DOWN TO 0);
End gate-02

Architecture behaviour of gate-02 is
Begin
c <= a OR b;
End Architecture;

Q



→ LIBRARY IEEE;
Use IEEE Std_logic_1164 all;
~~End~~
Entity gate_02 is
(Generic n Integer := 3;
Port (a,b : IN Std_logic_vector (n-1 DOWNTO 0);
c : Out Std_logic_vector (n-1 DOWNTO 0);
End gate_02;

Scanned with CamScanner

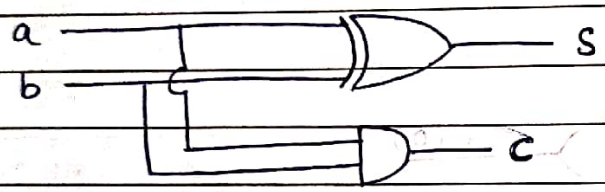
```
Architecture behaviour of gate_02 is
Begin
    c <= a OR b;
End Architecture;
```

Q. HA - addition of 2 bits.

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = \bar{a}b + \bar{b}a$

$C = ab$




```
Library IEEE
use IEE . std_logic_1164 all ;

Entity half-adder is
    port (a, b : IN std_logic );
          S, C : OUTPUT std_logic );
END half adder ;

BEGIN
        S <= a XOR b;
        C <= a AND b;
END Architecture ;
```

Scanned with CamScanner

Q. Full - Adder

```
Library IEEE;
Use IEEE std_logic_1164, all;

Entity full_adder is
    port ( a, b, cin : IN std_logic )
           S, Cout : OUT std_logic );
End full_adder ;

Architecture behaviour of full_adder is
    Signal d, e, f : std_logic;
Begin
    d <= a XOR b;
    s <= d XOR cin;
    e <= d AND cin;
    f <= a AND b;
    Cout <= e OR f;
End Architecture;
```

```
S <= a XOR b XOR cin;

Cout <= (cin AND (a XOR b))
        OR (a AND b);
```

Q. Binary to Grey Converter. (3 bit).

b_2	b_1	b_0	g_2	g_1	g_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

$g_2 = b_2 \cdot \bar{b}_1 \cdot \bar{b}_0 + b_2 \bar{b}_1 b_0 + b_2 b_1 \bar{b}_0 + b_2 \cdot b_1 \cdot b_0$

$g_1 = \bar{b}_2 b_1 + \bar{b}_0 + \bar{b}_2 b_1 b_0 + b_2 \bar{b}_1 b_0 + b_2 \bar{b}_1 b_0$

$g_0 = \bar{b}_2 \bar{b}_1 b_0 + \bar{b}_2 b_1 \bar{b}_0 + b_2 \bar{b}_1 b_0 + b_2 b_1 \bar{b}_0$

Scanned with CamScanner

Library IEEE;-
use Ieee std_logic_1164.all;

Entity bin_to_gray is
port (b_2, b_1, b_0 : IN std_logic;
g_2, g_1, g_0 : OUT std_logic;
End bin_to_gray;

Architecture behaviour of bin_to_gray is
Begin
$g_2 \Leftarrow$ (b_2 AND (NOT b_1) AND (NOT b_0)) OR (b_2 AND (NOT b_1) AND b_0) OR (b_2 AND b_1 AND (NOT b_0)) OR (b_2 AND b_1 AND b_0).

$g_1 \Leftarrow$

$g_0 \Leftarrow$

End Architecture;

⇒ When Statement

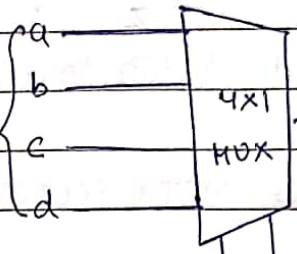
(i) When | Else
(ii) Selected When.

1. assignment WHEN condition ELSE
assignment WHEN condition ELSE
:

2. WITH identifier SELECT
assignment WHEN value;
assignment WHEN value;

Scanned with CamScanner

Multiplexer



→
```
Library IEEE;
Use IEEE.Std_logic_1164.all;

Entity mux is
    port(a,b,c,d,S1,S0: IN Std_logic;
         y: OUT Std_logic);
End mux;

Architecture behaviour of mux is
Begin
  y <= (a AND (NOT S1) AND (NOT S0)) OR
       (b AND (NOTS1) AND S0) OR
       (c AND S1 AND (NOT S0)) OR
       (d AND S1 AND S0);
End Architecture;
```

S_1	S_0	y
0	0	a
0	1	b
1	0	c
1	1	d

$y \Leftarrow a.\overline{S_1}.\overline{S_0} + b\,\overline{S_1}.S_0 + c.S_1.\overline{S_0} + d.S_1.S_0$

→ Using WHEN ELSE :-

```
Library IEEE;
Use IEEE.Std_logic_1164.all;

Entity mux is
    port(a,b,c,d: IN Std_logic;
         S: IN Std_logic_vector(1 DOWN TO 0));
End mux;

Architecture behaviour of mux is
Begin
  y <= a WHEN S = "00" ELSE
       b WHEN S = "01" ELSE
       c WHEN S = "10" ELSE
```

Scanned with CamScanner

```
    d WHEN S = "11" ELSE
    'Z' ;
End Architecture ;
```

→ Using WITH SELECT

```
Library Ieee ;
Use Ieee. std_logic_1164. all ;

Entity mux is
  port (a, b, c, d  : IN std_logic ;
        S : IN std_logic. vector (1 DOWNTO 0) ;
End mux ;

Architectural behaviour of mux is
Begin
    WITH  S  SELECT.
      y <=  a  WHEN "00",
            b  WHEN "01",
            c  WHEN "10",
            d  WHEN "11",
            'Z' WHEN OTHERS ;
End Architecture ;
```

Q.

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	a(7:0)
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	1	0	
0	0	0	0	0	1	0	0	
1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	
1	0	0	0	0	0	0	0	

Encoder

b_2	b_1	b_0	b(2:0)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	1	1	
1	1	1	

Scanned with CamScanner

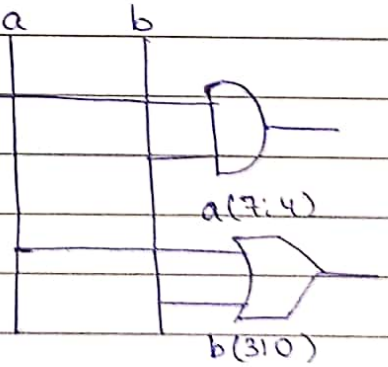
```
Library Ieee;
Use Ieee std_logic_1164.all;

Entity Encoder is
    port (a: IN std_logic_vector (7 down to 0);
          b: OUT std_logic_vector (2 down to 0);
End Encoder;

Architecture behaviour of Encoder is
Begin
  b <= "000" WHEN a = "00000001" ELSE
       "001" WHEN a = "00000010" ELSE
       "010" WHEN a = "00000100" ELSE
        :
        :
       "111" WHEN a = "1000 0000" ELSE
       "zzz";
End Architecture;
```

Q. For | Generate

```
c(0) <= a(3) AND b(7);
c(1) <= a(2) AND b(6);
c(2) <= a(1) AND b(5);
c(3) <= a(0) AND b(4);
```




```
LIBRARY Ieee;
Use Ieee: std_logic_1164.all;

Entity Gate_03 is
    port (a,b :IN std_logic_vector (7 down to 0);
          c,d : OUT std_logic_vector (3 down to 0);
End Gate_03;
```

Scanned with CamScanner

Architecture behaviour of Gate-03 is
Begin.
L1 : FOR i IN 0 TO 3 Generate
c(i) <= a(i) AND b(i+4);
d(i) <= b(i) OR a(i+4);
End Generate ;

LABEL : For identifier IN RANGE.
statements
End Generate ;

Scanned with CamScanner

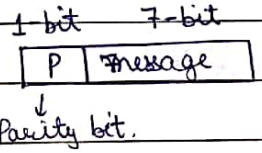
⇒ Parity Generation & Checking :-

There is a chance of error in transmitting the bits called parity.

eg:- send 10111 (even no of 1's) ⟶ reciever 10011 (odd no of 1's)

* Parity bit is used for the purpose of detecting errors during the transmission of bit sequence.

Parity bit → is an extra bit included with the sequence (message bit)



* Parity Generator → It generate parity bit of the message at the transmitter side

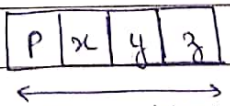
* Parity checker → It checks the parity bit at reciever side.

→ Even Parity Generator :-

↳ To make no. of ~~bits~~ 1's even.

x	y	z	P
0	0	0	0
0	0	1	1 → To make even no of 1's.
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

no of 1's
0
2
2
2
2
2
4



Scanned with CamScanner

If P is an odd function then it is an even parity generator.

=> To design a even parity generator :-

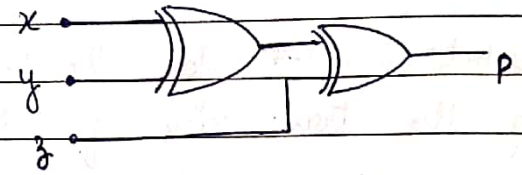
Step-1 K-map for P

	00	01	11	10
0		1		1
1	1		1	

∴ P is an odd function
∴ $P = x \oplus y \oplus z$.

Step-2

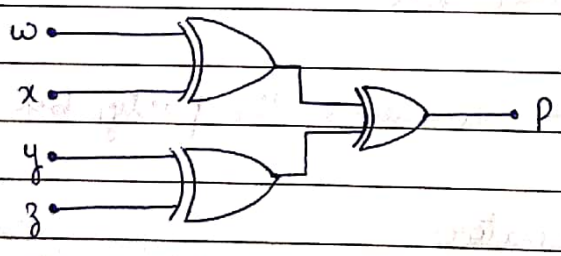
Circuit →



3-input parity generator.
(even parity)

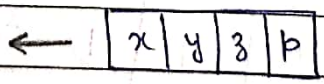
→ To design 4-input even parity generator :-

$P = x \oplus y \oplus z \oplus w$
or $(w \oplus x) \oplus (y \oplus z)$



→ Even Parity Checker :-

we can keep parity bit at the last or at the start depending upon user needs.



At the reciever side 4 bits will be recieved.
& the recieved bits must have even no. of 1's.

* error will occur if during transmission 4-bits recieved have odd no of 1's

Scanned with CamScanner

2's 110 → write till encounter first 1 & then flip rest bits
010

00111010 → 11000110

x	y	z	p	c
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

c = 0 if no error

c = 1 if error occurs.

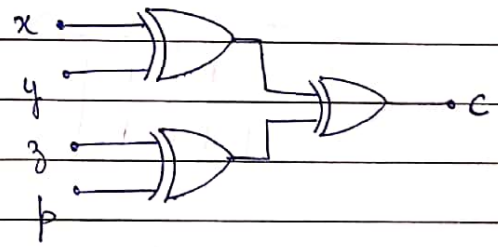
K map for c

xy \ zp	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

c is an odd function

c is output of 4-input XOR gate.

$c = (x \oplus y) \oplus (z \oplus p)$



Q. Design a circuit of a 3-bit parity generator & 4-bit parity checker, using odd parity bits.

Q. Design a circuit which generates 2's complement of the input binary number

(i) with 3 input & 3 output

(ii) with 4 input & 4 output.

110
001 → 1's
+ 1
010 → 2's

Scanned with CamScanner

$$\begin{array}{r} 110 \\ +\ 1 \\ \hline 111 \end{array} \qquad \begin{array}{r} 100 \\ 011 \\ +\ 11 \\ \hline 100. \end{array}$$

number	x y z	2's C A B C.
0	0 0 0	0 0 0
1	0 0 1	1 1 1
2	0 1 0	1 1 0
3	0 1 1	1 0 1
4	1 0 0	1 0 0
5	1 0 1	0 1 1
6	1 1 0	0 1 0
7	1 1 1	0 0 1

K-map for A

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

K-map for B

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

K-map for C

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

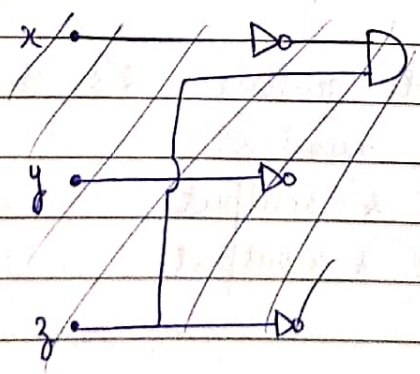
$A = \bar{x}z + \bar{x}y + x\bar{y}\bar{z}$

$B = \bar{y}z + \bar{z}y = y \oplus z$

$C = z$

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

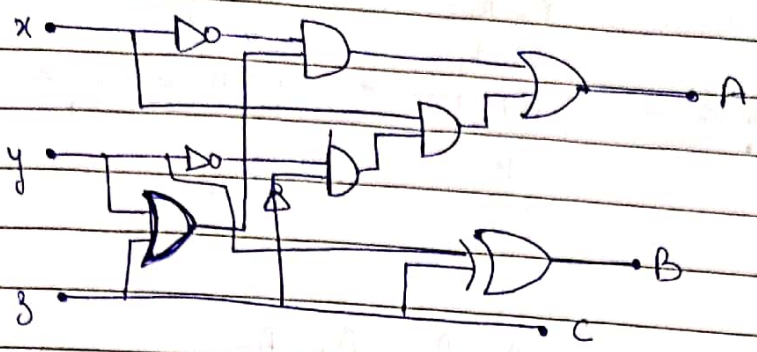
$= \bar{x}(z+y) + x\bar{y}\bar{z}$



Scanned with CamScanner

$A_3\ A_2\ A_1\ A_0$

A = 1 0 1 1



→ Combination logic with MSI & LSI :-

→ Level of Integration of IC :-

			no. of gates	
(i)	Small Scale Integration	(SSI)	< 10	eg- decoder, MUX, Adder/Subtractor
(ii)	Medium "	" (MSI)	10 - 100	
(iii)	Large "	" (LSI)	100 - 1000	small memory chips, processors.
(iv)	Very Large "	" (VLSI)	>1000	complex processors

→ Binary Adder / Subtractor

The n-bit binary adder / subtracter can be designed by utilising n number of full adders.

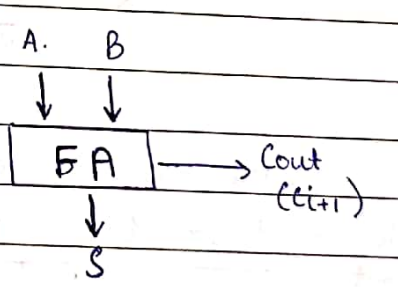
Output Carry, $i = 3, 2, 1, 0$ → C_{i+1} : 01 10

Augent → A_i : 10 11

Addent → + B_i : 0011

Sum → S : 1110

Input carry → C_i : 00 1 1



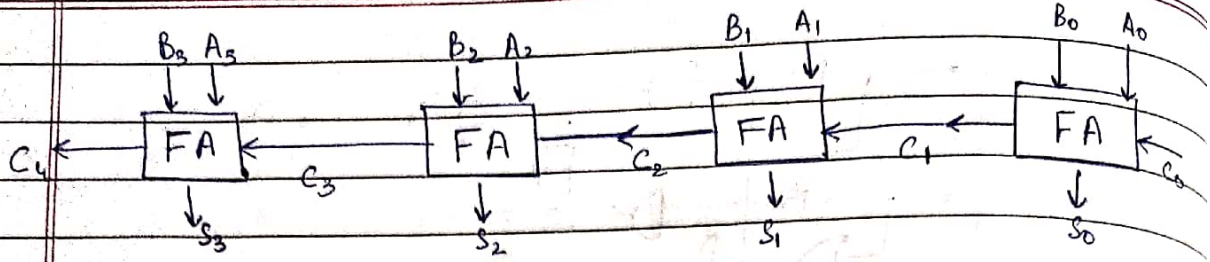
→ Binary Parallel Adder :-

- Use 4 FA connected in chain
- Output carry of FA is connected to input carry of the next stage full adder in the chain.

Scanned with CamScanner

Sub → A - B

= A + (2's C of B)



C_3 C_2 C_1 C_0

A_3 A_2 A_1 A_0

B_3 B_2 B_1 B_0

C_4 S_3 S_2 S_1 S_0

→ Binary Subtracter :-

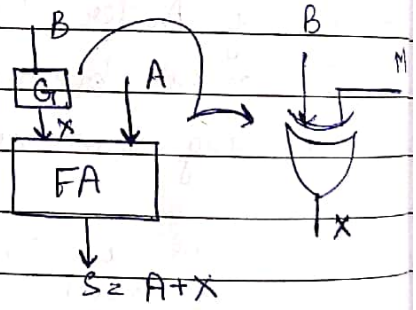
A - B = A + 2's Comp of B

= A + 1's Comp of B + 1

Subtracter can be re-utilised by using the same circuit as above except

(i) C_0 should be 1

(ii) B input should be inverted



$A + X$

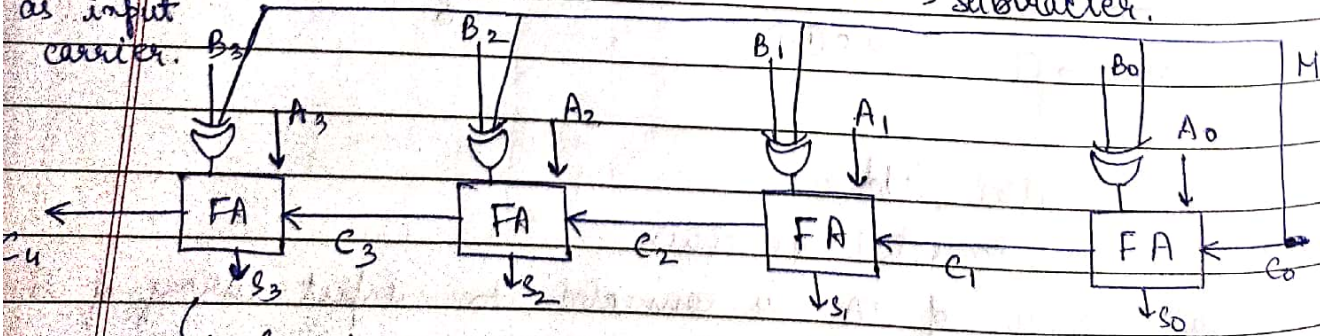
$X = B \oplus M$

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

$M = 0 \quad X = B$ → Adder

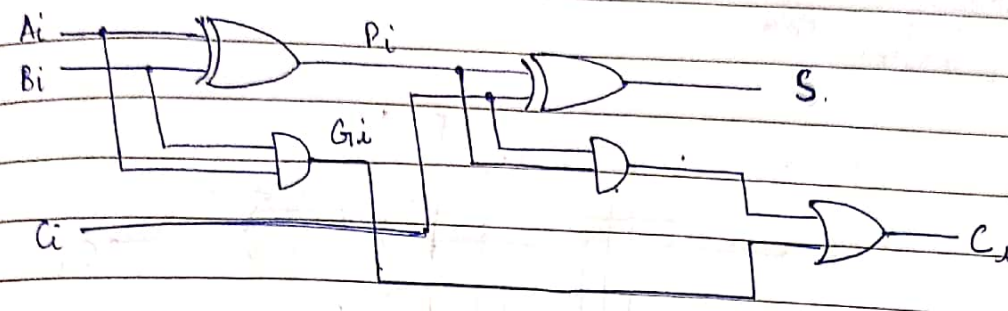
$M = 1 \quad X = \bar{B}$ → Subtracter.

Mean about M as input carrier.



↳ Can be used as both Adder & Subtracter

Scanned with CamScanner



In 1 bit FA

$C_0 - C_1$ => the carry will pass through 2 gates (AND, OR).

In n bit FA

$C_0 - C_n$ => the carry will pass through 2n gates

∴ as it passes through ~~a~~ large no. of gates ∴ will be a lot of delay in ~~communication~~ propagation.

↳ To make the addition ~~fast~~ / subtraction fast we need to reduce the carry propagation delay time.

↳ The solution is Look-ahead carry.

→ For the Full Adder:-

$P_i = A_i \oplus B_i$

$G_i = A_i B_i$

The output, $S = P_i \oplus C_i$

next carry ← $C_{i+1} = G_i + P_i C_i$

The expression of next carry can be written as.

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1$

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

$= G_1 + P_1 G_0 + P_1 P_0 C_0$

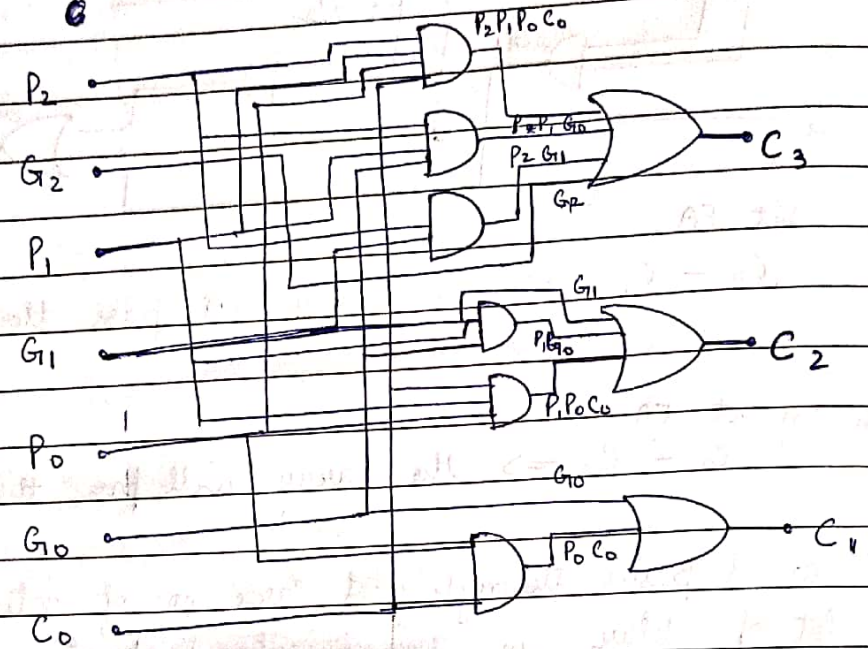
$C_3 = G_2 + P_2 C_2$

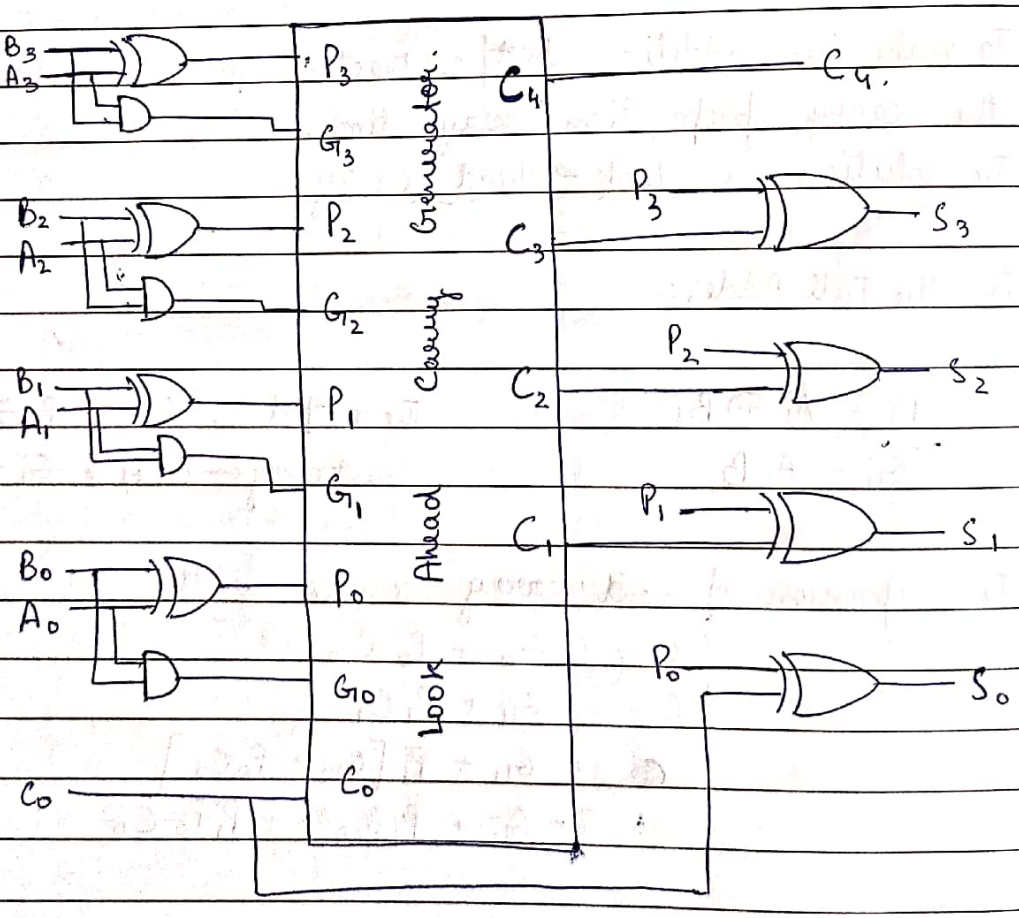
$= G_2 + P_2 [G_1 + P_1 G_0 + P_1 P_0 C_0]$

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

Scanned with CamScanner

Look Ahead carry Realisation



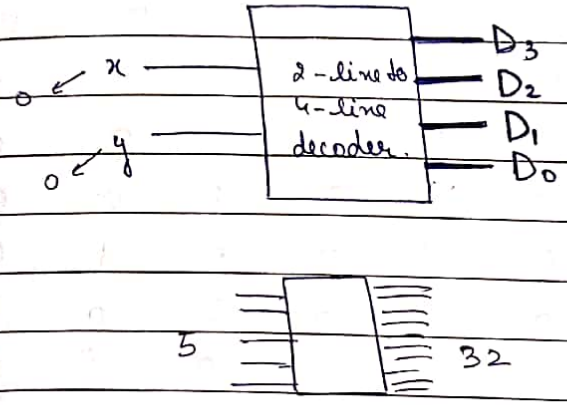


Scanned with CamScanner

classmate
Date
Page

Decoder & Encoder :-

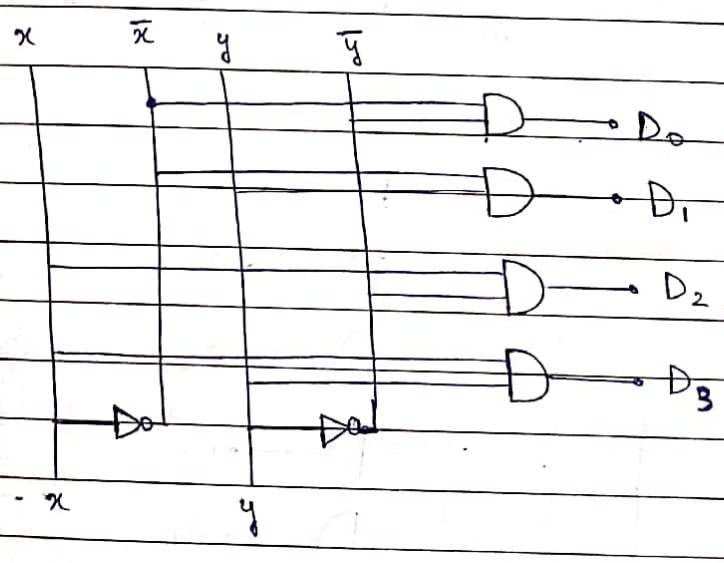
decoder is a combinational circuit that converts binary information from n-input lines to a max of 2^n unique output lines ⇒ n-input ⟶ 2^n out max lines.



Truth Table

x	y	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

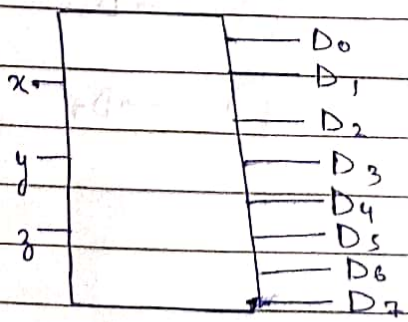
$D_0 = \bar{x}\bar{y}$ $D_1 = \bar{x}y$ $D_2 = x\bar{y}$ $D_3 = xy$



logic implementation.

2×4 decoder.

Ques- Make a 3×8 decoder.



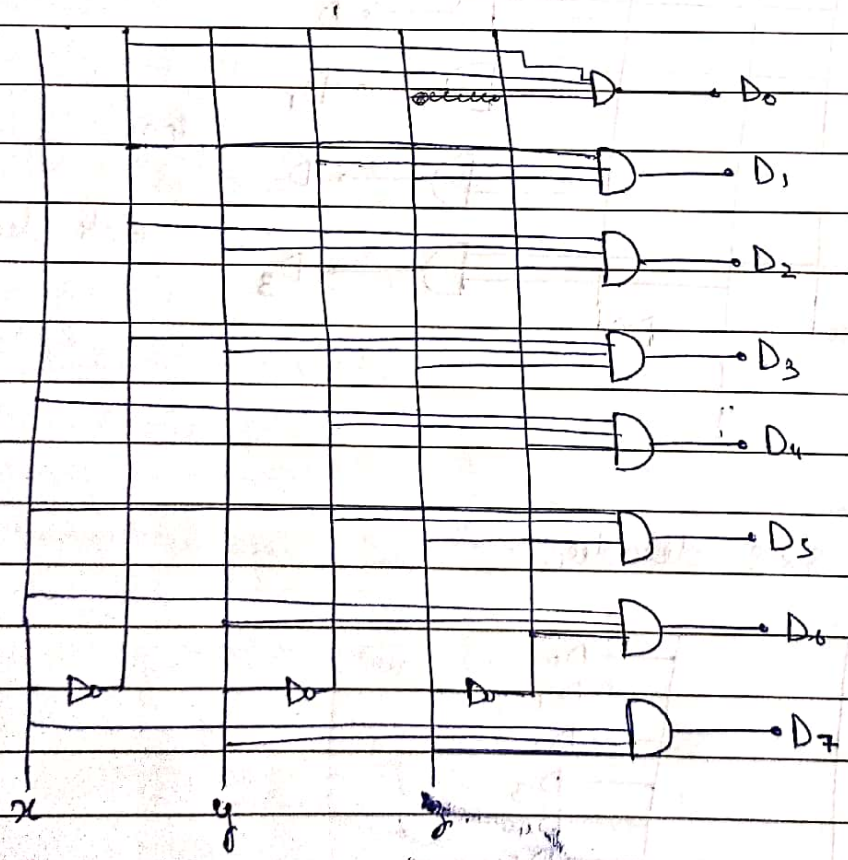
Scanned with CamScanner

→ 3 × 8 decoder

x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$D_0 = \bar{x}\bar{y}\bar{z}$ $D_1 = \bar{x}\bar{y}z$ $D_2 = \bar{x}y\bar{z}$ $D_3 = \bar{x}yz$

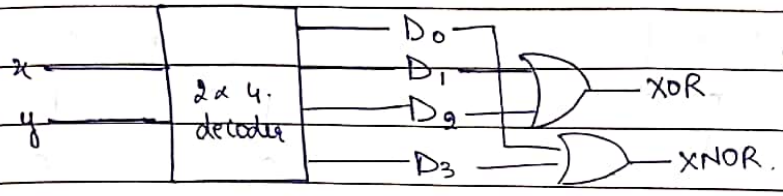
$D_4 = x\bar{y}\bar{z}$ $D_5 = x\bar{y}z$ $D_6 = xy\bar{z}$ $D_7 = xyz$



Scanned with CamScanner

→ Logic Implemented using decoder:
Function to be implemented ~~using~~ should be expressed in SOP

Q. Implement 2-input XNOR & XOR using decoders.



$x \oplus y = x\bar{y} + \bar{x}y$
$= D_1 + D_2$.

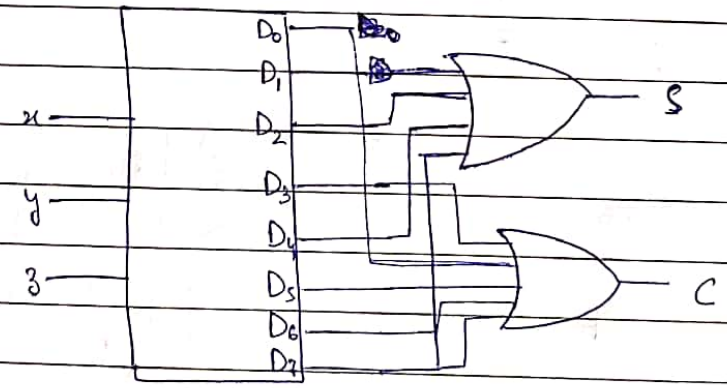
$\overline{x \oplus y} = \bar{x}\bar{y} + xy$.
$= D_0 + D_3$

$D_1 + D_2 = \overline{D_0 + D_3}$

Q. Implement FA using decoder.

Since FA has 3 input we use 3×8 decoder.

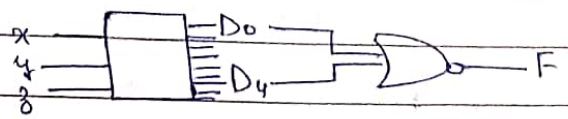
$S = \sum(1, 2, 4, 7)$
$C = \sum(3, 5, 6, 7)$



Q. $F = \sum(1, 2, 3, 5, 6, 7)$

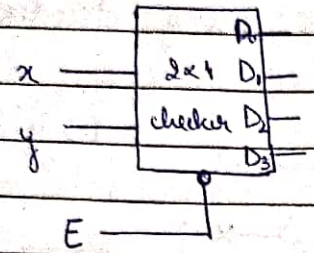
↳ It will take a 6 input OR GATE but it is not a good way.

$\bar{F} = \sum(0, 4)$ → It will take a NOR gate with 2 input.



Scanned with CamScanner

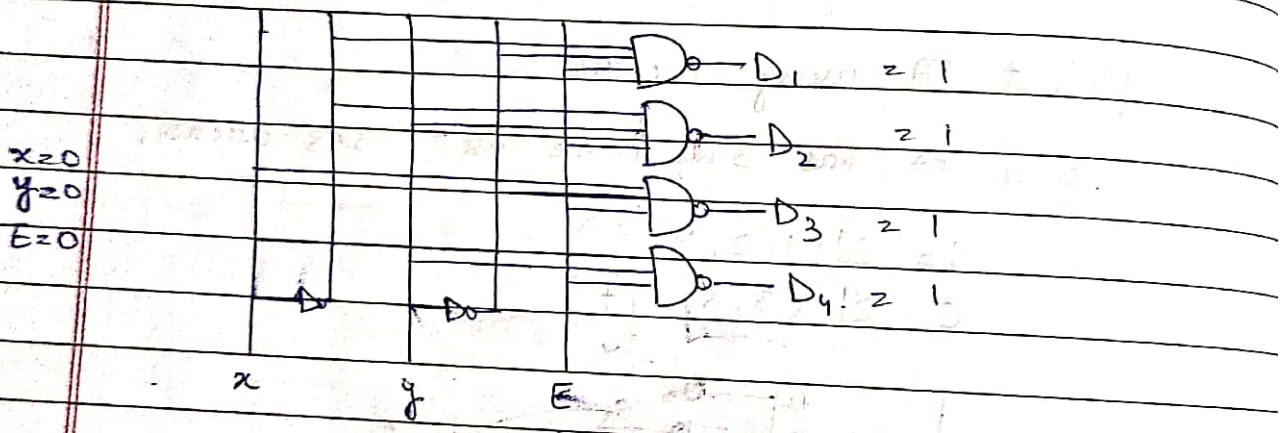
⇒ Decoder ⁻ Enable Input :-



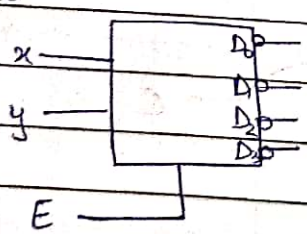
E	x	y	D_0	D_1	D_2	D_3
1	x	x	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

E = 0 → Decoder is Enabled

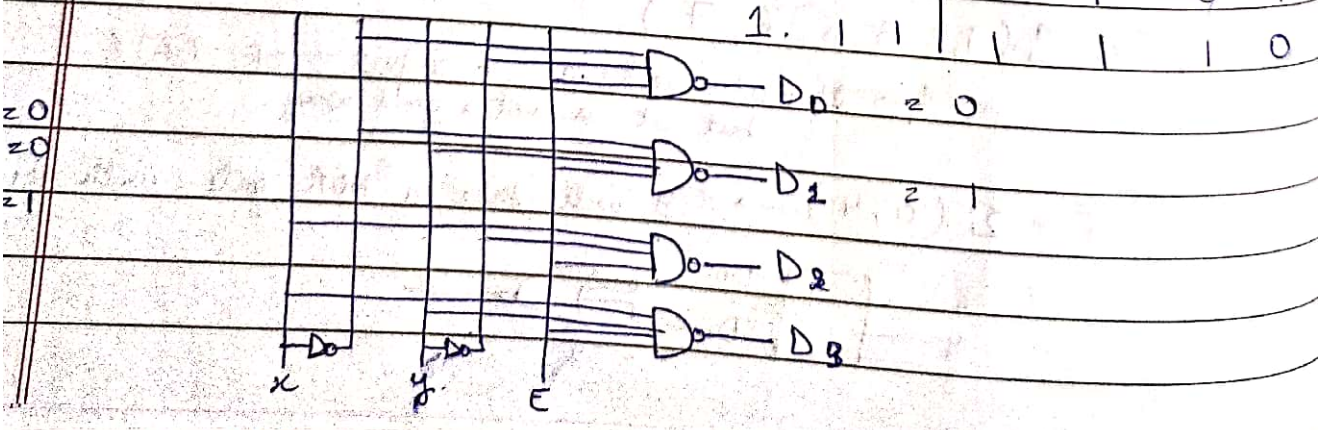
E = 1 → " " Disabled.



Now



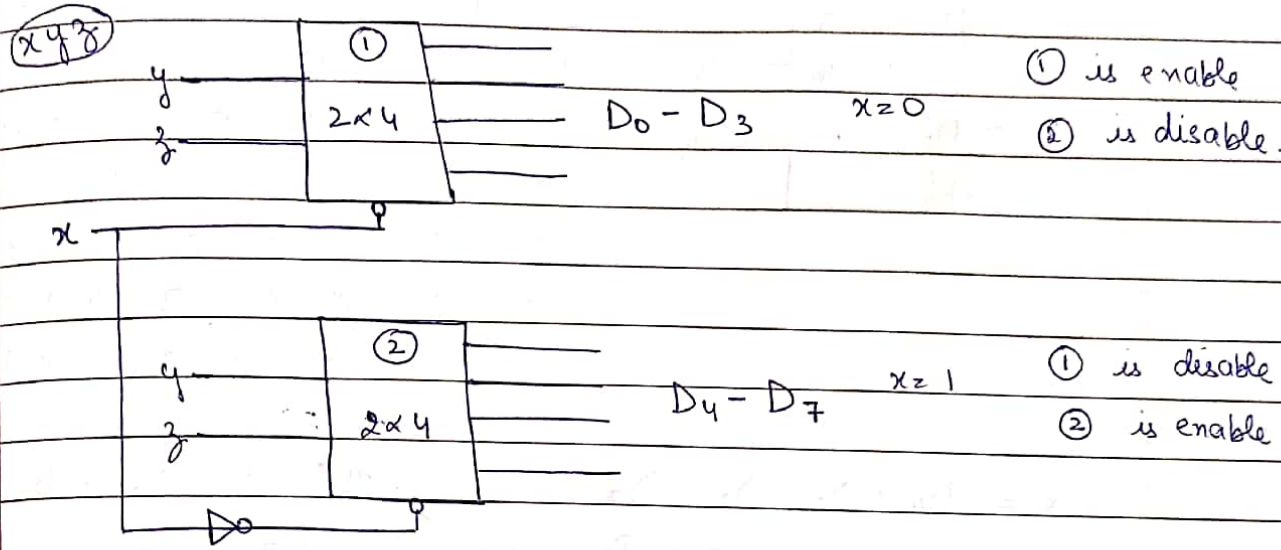
E	x	y	D_0	D_1	D_2	D_3
0	x	x	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0



Scanned with CamScanner

⇒ Larger Decoder using smaller Decoder:

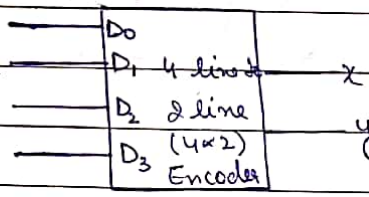
3×8 decoder



Que - Design a 4×16 decoder using a) 2 → 3×8 decoders
b) 4 → 2×4 decoders

⇒ Encoder :-

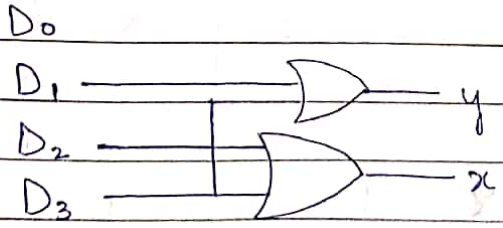
A digital circuit that performs reverse operation of decoder.



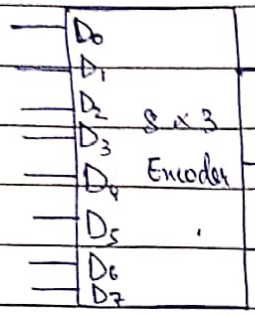
D_0	D_1	D_2	D_3	x	y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

$x = D_2 + D_3$

$y = D_1 + D_3$



Scanned with CamScanner



D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$x = D_4 + D_5 + D_6 + D_7$ $\quad y = D_2 + D_3 + D_6 + D_7$

$z = D_1 + D_3 + D_5 + D_7$

→ Limitations:-

1. If more than 2 - inputs are high $D_2 = D_3 = 1 \rightarrow xy = 11$
   but $x = 1$ & $y = 1$ for $D_2 = 0$, $D_3 = 1$.
   ∴ It is not desired. *[can be solved by setting high priority]

2. If all the inputs are 0 $x, y \rightarrow 0, 0$
   $D_0 = 1$ others 0 $\rightarrow x, y \rightarrow 00$
   } solved using additional output for all zero case.

⇒ Priority Encoder :-

It is an encoder circuit which include the priority function.

D_0 (low priority)	D_1	D_2	D_3 (High)	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
~~X~~ ← D_0 can be either 0 or 1	1	0	0	0	1	1
~~X~~ but we give	~~X~~	1	0	1	0	1
priority to D_1 0	0	0	1	1	1	1

Scanned with CamScanner

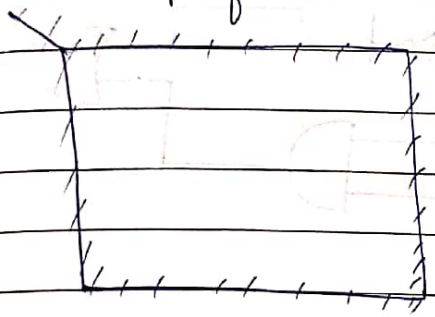
K-map for x

D_0D_1 \ D_2D_3	00	01	11	10
00	X	1	1	1
01		1	1	1
11		1	1	1
10		1	1	1

K-map for y

D_0D_1 \ D_2D_3	00	01	11	10
00	X	1	1	
01	1	1	1	
11	1	1	1	
10		1	1	

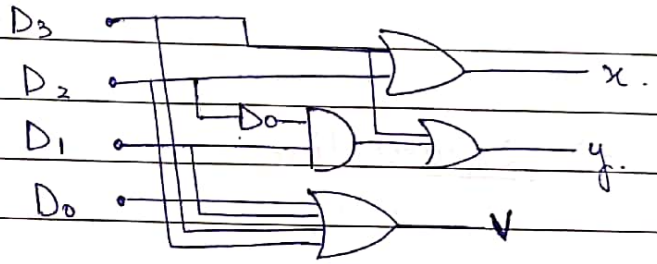
K-map for V



$x = D_2 + D_3$

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

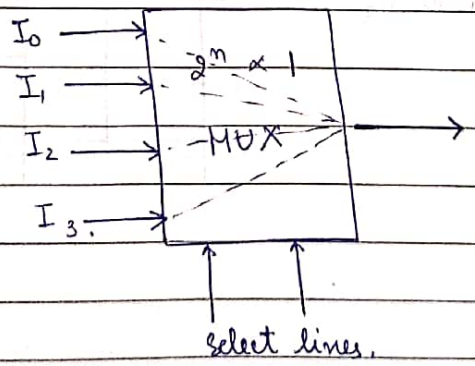
$V = D_0 + D_1 + D_2 + D_3$



⇒ **Multiplexer :-**

It is a combination circuit that selects binary information from one of the many input lines and direct it to a single output line.

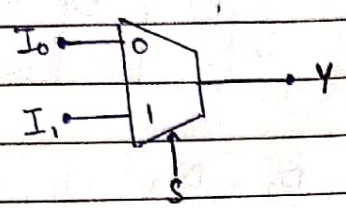
It is also known as data selector.



Scanned with CamScanner

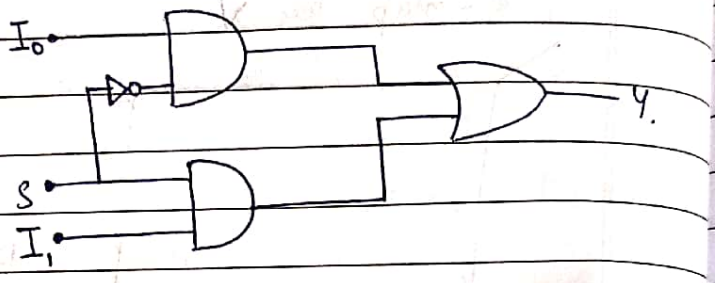
→ 1 Socket line :-
(2¹ × 1)

2 × 1 MUX

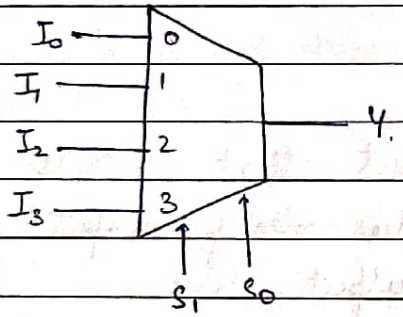


S	Y
0	I_0
1	I_1

$Y = \bar{S} I_0 + S I_1$

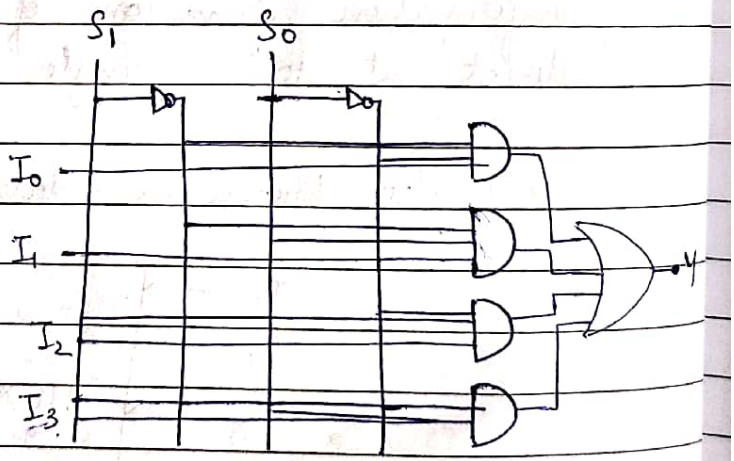


=> 2 Socket line :-
($2^2 \times 1$)



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

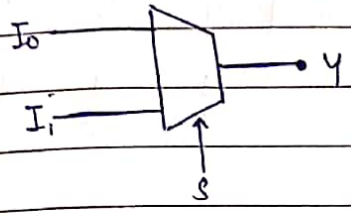
$Y = \bar{S_1}\bar{S_0} I_0 + \bar{S_1} S_0 I_1 + S_1 \bar{S_0} I_2 + S_1 S_0 I_3$



Scanned with CamScanner

→ Function Implementation using MUX :-

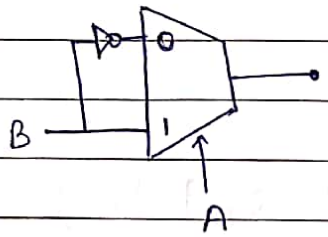
Ex 1 Implement ~~using~~ 2-input XOR using MUX.



$$Y = \bar{S} I_0 + S I_1$$

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

$$S = A, \quad \bar{B} = I_0, \quad B = I_1$$



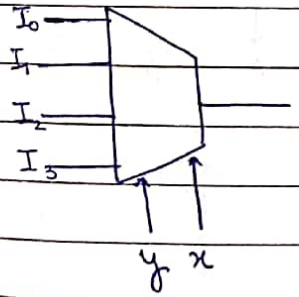
Ex 2 Implement $F = \sum (1, 2, 4, 6)$ using MUX

↳ $\bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$ --- [we can relate $\bar{c}$: 4×1 MUX formula ~~as done above~~]

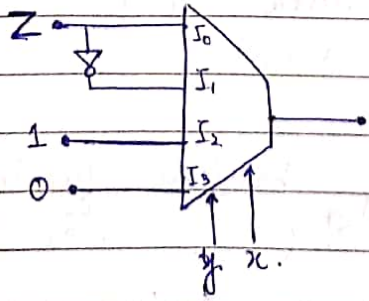
Method 1 → 2

x, y, z

$S_1 \rightarrow y$ $\quad$ $S_0 \rightarrow x$.



	I_0	I_1	I_2	I_3
$\bar{z}$	0	1	2	3
z	4	5	6	7
	↓	↓	↓	↓
	z	$\bar{z}$	$z+\bar{z}=1$	0

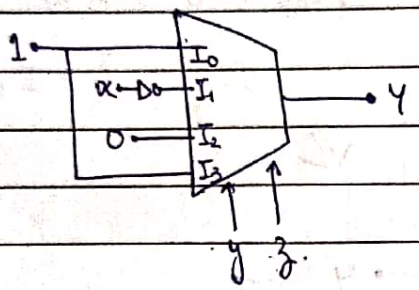


Scanned with CamScanner

Eg-3 $F = \Sigma(0, 1, 3, 4, 7)$ using MUX.

* y, z → selector
  x → input.

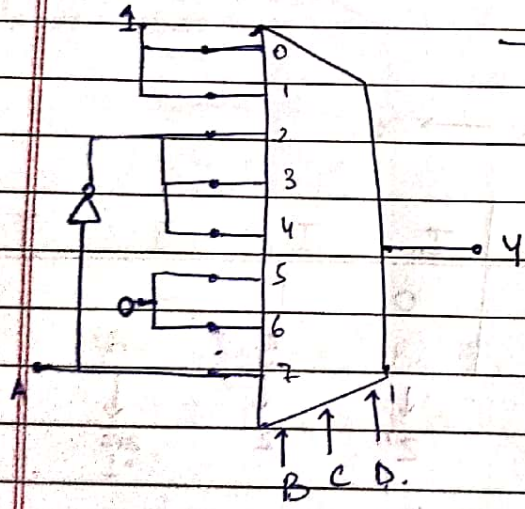
	I_0	I_1	I_2	I_3
$\bar{x}$	0	1	2	3
x	4	5	6	7
	↓	↓	↓	↓
	1	$\bar{x}$	0	1



Eg-3 $F = \Sigma(0, 1, 2, 3, 4, 8, 9, 15)$
A B C D

use 8×1 MUX

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	↑	↑	↓	↓	↓			↓
	1	1	$\bar{A}$	$\bar{A}$	$\bar{A}$			A



Ques 1 Implement a full-adder using 4×1 MUX
→ we need 2 MUX → S, C

2 Implement following boolean function with 8×1 MUX
$F(ABCD) = \Sigma(0, 3, 5, 6, 7, 11, 13, 14)$

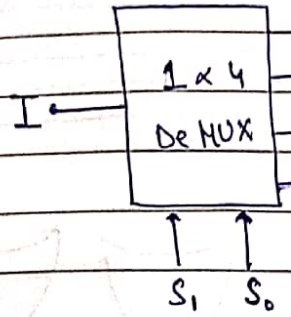
Scanned with CamScanner

ROM → To permanently store binary info.

⇒ De Multiplexer (De MUX) :-

↳ performs reverse operation of a MUX.

↳ It transfers information from one input line to one of the 2^n possible output line [controlled by n-select lines].

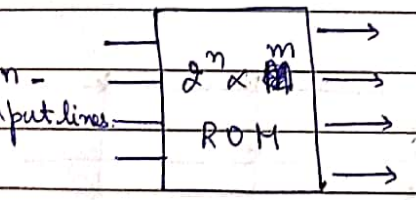


S_1	S_0	D_0	D_1	D_2	D_3
0	0	I	1	1	1
0	1	1	I	1	1
1	0	1	1	I	1
1	1	1	1	1	I.

↳ can be zero or 1 depending on the design.

⇒ Read Only Memory (ROM) :-

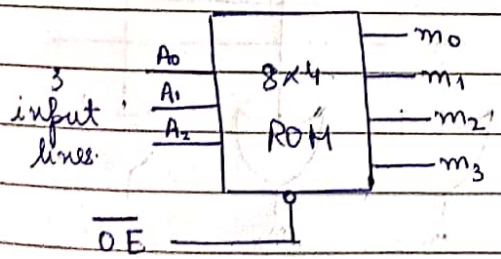
↳ It is a memory device which store binary information permanently.



n - address input lines

No. of words = 2^n.

word size / no. of bits per word = m



word no	A_0	A_1	A_2	m_0	m_1	m_2	m_3
0	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
2	0	1	0	D	D	D	D
3	0	1	1	D	D	0	D
4	1	0	0	0	0	D	D
5	1	0	1	0	0	1	1
6	1	1	0	1	0	0	0
7	1	1	1	1	1	0	0

↳ random no's.

If we find word $A_0 A_1 A_2$ = 100

↳ output = 0000

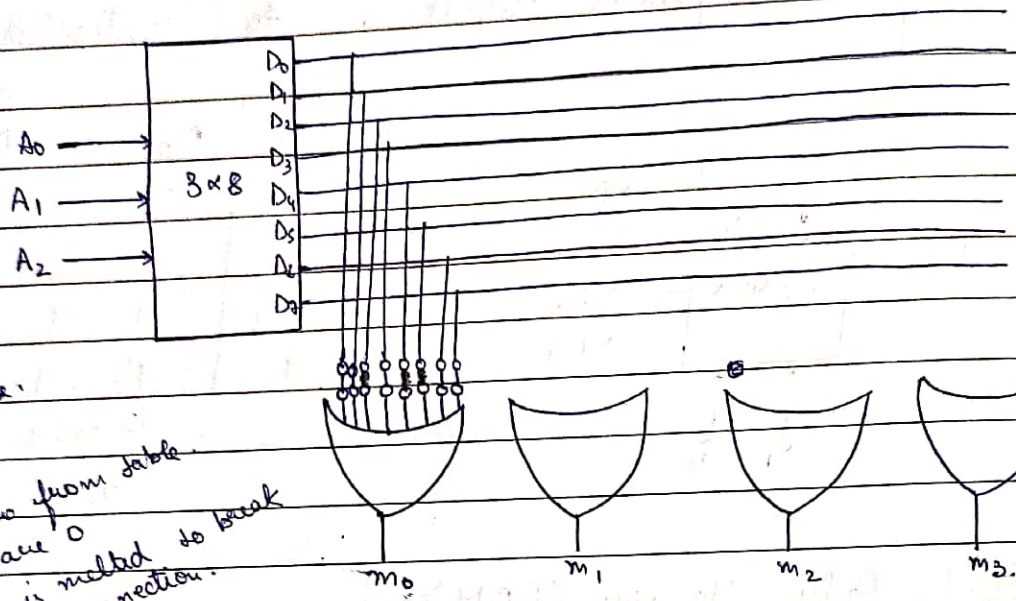
OE = 1 output = Hi - Z

OE = 0 output = 0000

Scanned with CamScanner

→ Logic implementation of 8×4 ROM :-

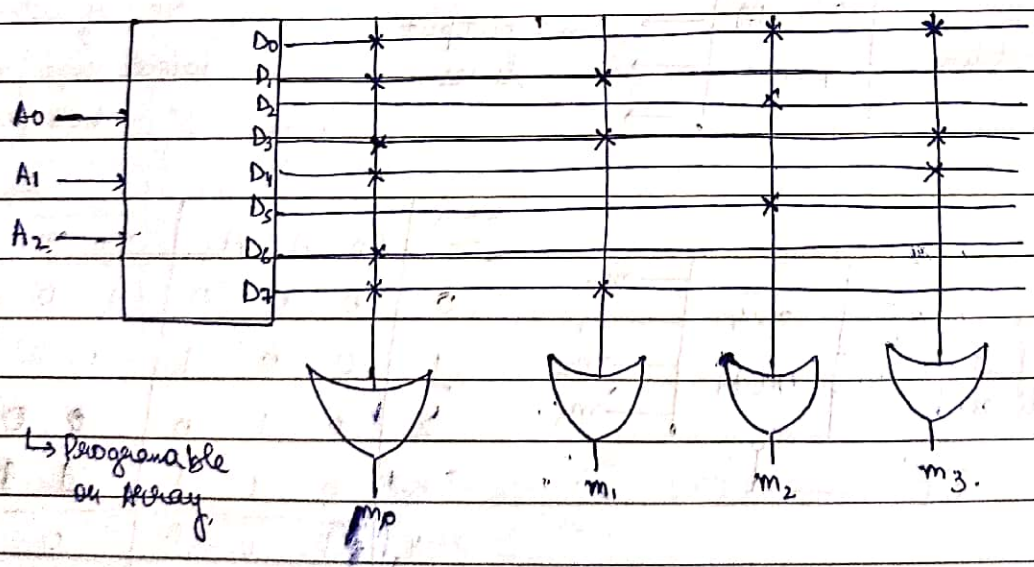
No. of words = 8 ⇒ decoder - size = 3×8



0 → melt fuse
1 → keep fuse.

Now for m0 from table.
D_2, D_1, D_5 are 0
∴ fuse is melted to break the connection.

* To simplify it we use notations :↓



↳ Programmable OR array.

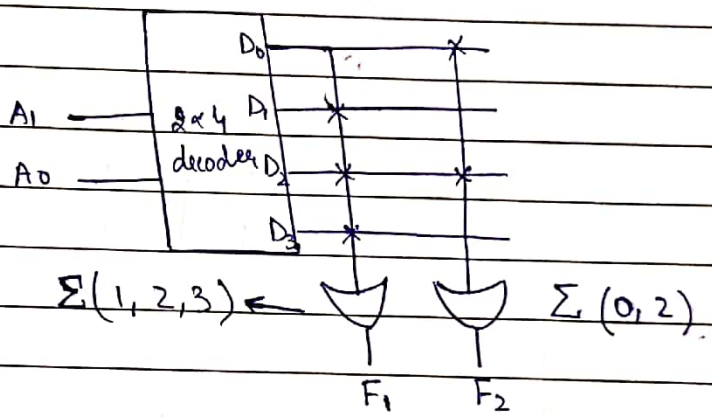
Scanned with CamScanner

⇒ Combination logic implementation using ROM:-

Q.1 Implement following functions ~~~~ shown in the table

A_0	A_1	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

Decoder size = 2×4.
NO. of OR gate = 2.



Q. Implement the following function in a ROM.

$F_1 = \sum (0, 1, 2, 5, 7)$

$F_2 = \sum (0, 2, 7)$ =>

$F_3 = \sum (1, 2, 6, 7)$

$F_4 = \sum (1, 2, 5, 6)$.

⇒ Programmable Logic Array (PLA):-

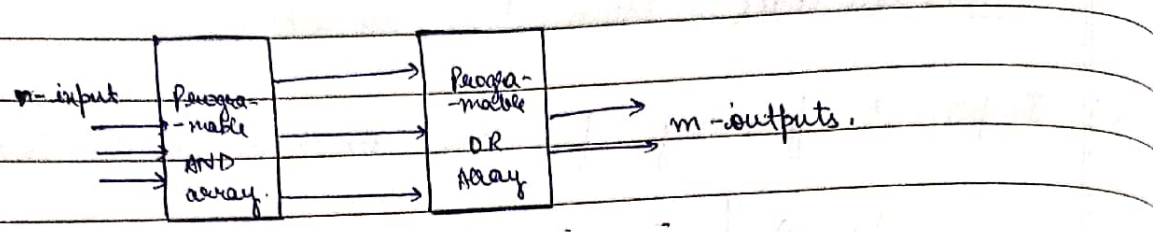
- ROM generates all minterms but in PLA we generate all programmable minterms.

ROM → AND array is fixed.
OR " " programmable.

PLA → both AND & OR array are programable.

Function Implementation:
- AND array → product terms
- OR array → adding the product terms

Scanned with CamScanner

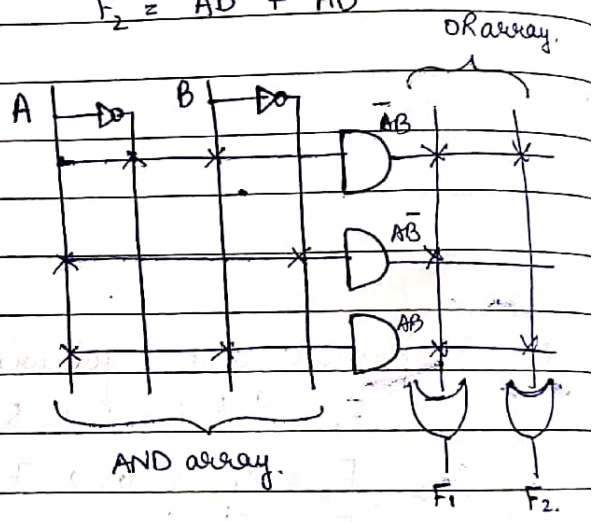


Q. Implement using PLA logic.

A	B	F_1	F_2
0	0	0	0
0	1	1	1
1	0	1	0
1	1	1	1

$F_1 = \bar{A}B + A\bar{B} + AB$

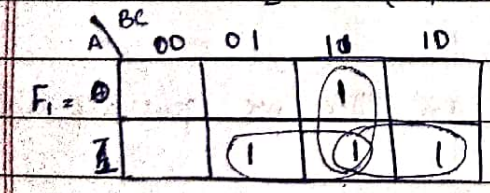
$F_2 = \bar{A}B + AB$



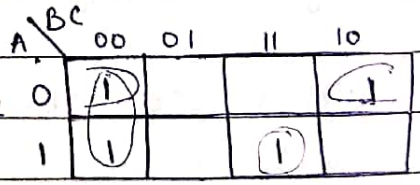
Q. Implement the following function using PLA logic.

$F_1 = \Sigma(3, 5, 6, 7)$

$F_2 = \Sigma(0, 2, 4, 7)$



$F_1 = ABC + AB + AC$

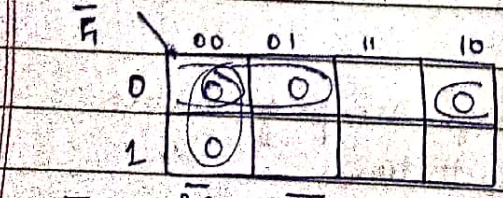


$F_2 = \bar{B}\bar{C} + \bar{A}\bar{C} + ABC$

Here we need 6 AND gates

further minimize

$\bar{F}_1 = \Sigma(0, 1, 2, 4)$



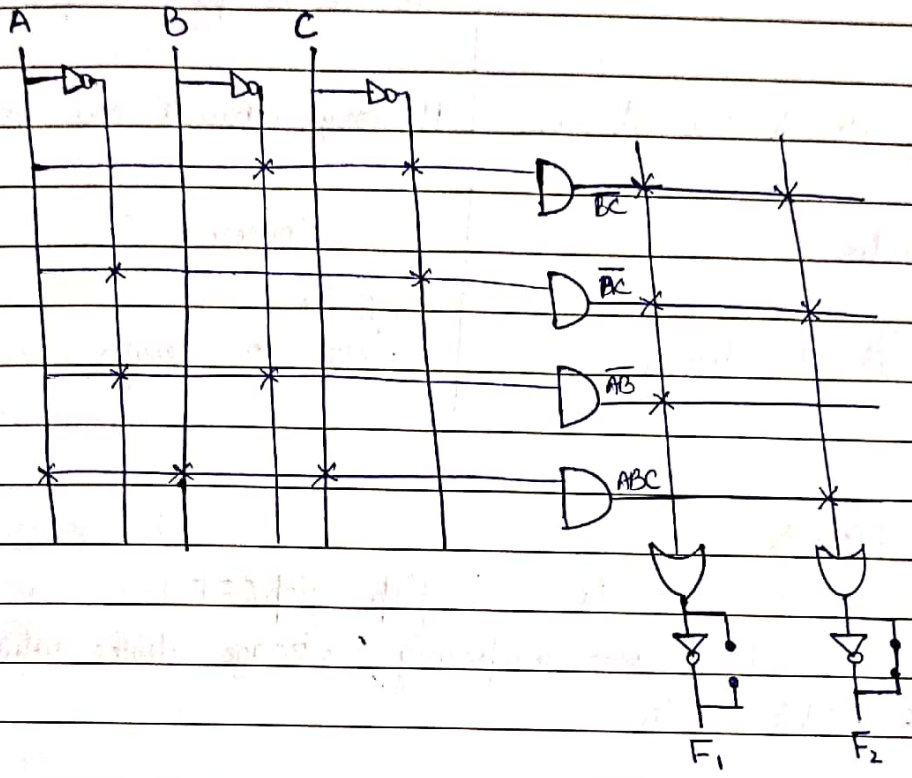
$\bar{F}_1 = \bar{B}\bar{C} + \bar{A}\bar{C} + \bar{A}B$ Now we need 4 AND gates

Scanned with CamScanner

★ PLA has addition not gate with a fuse, which melts if we want complemented value.

T → keep connection

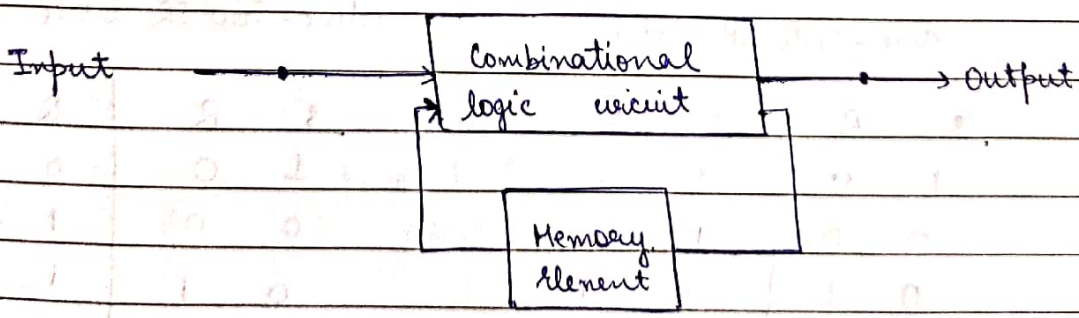
F → melt " "



Q. Realize full adder circuits using RLA logic.

Q. Realize BCD to excess-3 code converter using PLA logic.

⇒ Sequential Logic Current :-



Sequential circuit is formed by combinational logic with memory element in feed back path

Scanned with CamScanner

state → operation condition of

Combinational logic	Sequential Logic
1. Output depends on present input.	Output depends on present input as well as past output.
2. Memory elements are not req.	Memory elements are required.
3. Faster	slower
4. ~~difficult~~ easy to design	difficult ~~easy~~ to design.

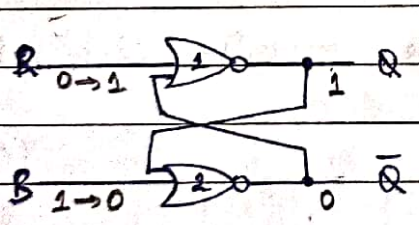
⇒ Memory Elements:-

↳ ~~non~~ memory elements are flip flop (FF).

↳ a type of ~~memory~~ temporary storage device which has two stable states

1. Set - Reset (SR) latch :-

~~or direct coupled SR FF~~



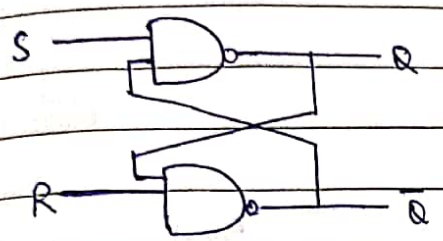
Active - high SR latch. Active - low SR latch.

NOR	S	R	Q	Q̄		S	R	Q	Q̄
1	1	0	1	0	→ set	avoid 1	0	0	1
0	0	0	1	0	→ no change	0	0	1	1
0	0	1	0	1	→ reset	0	1	1	0
Avoid this state	1	1	0	0	→ invalid	1	1	1	0

Scanned with CamScanner

classmate
Date
Page

⇒ SR Latch or Direct coupled SR FF :-



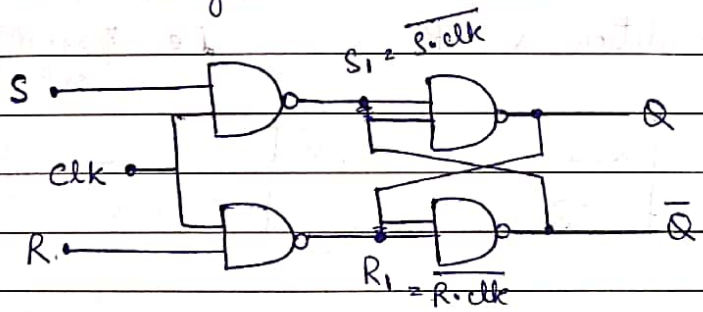
S	R	Q	$\bar{Q}$	
1	0	0	1	Reset
0	0	1	1	Invalid
0	1	1	0	Set
1	1	1	0	No change

→ SR Flip Flop :-

The latch can be modified by giving an additional input which determine when the state of FF is to be changed.

clocked FF.

The clock signal (clk) will determine when the state is to be changed.



when clk = 0

$S_1 = R_1 = 1$

No change in O/P

when clk = 1

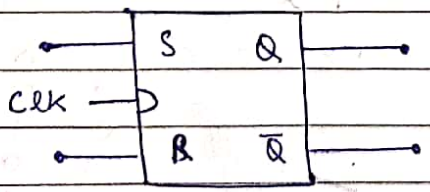
$S_1 = \bar{S}$, $R_1 = \bar{R}$

excitation table

clk = 1

S	R	Q_{n+1}	
0	0	Q_n	no change ①
0	1	0	Reset ② [R is high]
1	0	1	set ③ [S is high]
1	1	Invalid state ④	

» I calculate value of S_1 & R_1 & then from previous table see Q_{n+1}



Scanned with CamScanner

characteristic table

Q	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Invalid(x)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Invalid.(x)

Q_n \ SR	00	01	11	10
0			X	1
1	1		X	1

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

Characteristic eqn

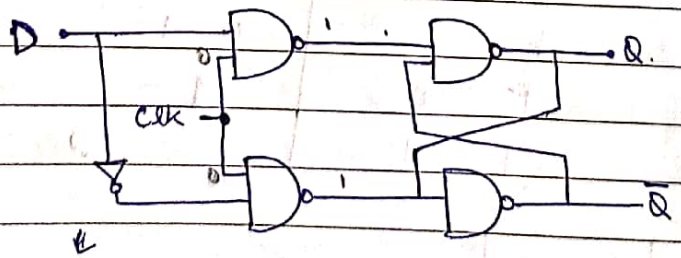
But we must add something in it such that we can represent the invalid condition & avoid S=1, R=1.

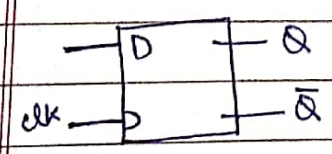
∴ The condition is $SR = 0$ i.e if S=1 R=0 or R=1 S=0

$$\therefore \boxed{\begin{array}{l} Q_{n+1} = S + Q_n \overline{R} \\ SR = 0 \end{array}}$$

⇒ **D - Flip Flop :-**

to ensure that both S & R don't have '1' value we use a not gate & short circuit the inputs.





clk = 0

$Q_{n+1} = Q_n$

↳ no change.

clk = 1

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

∴ it doesn't depend on Q_n.

Scanned with CamScanner

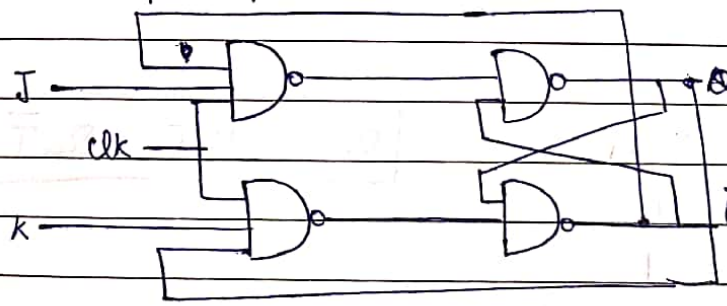
characteristic eqn → $Q_{n+1} = D$

∴ The input is directed to the output

Q. Determine Q & $\bar{Q}$ waveform for a D-FF.

The drawback is that we using only 2 states (0,1) & (1,0).

→ JK - Flip Flop :-



excitation table

clk = 1

	clk	J	K	Q_n	
①	↑	0	0	Q_n	No change
②	↑	0	1	0	reset
③	↑	1	0	1	set
④	↑	1	1	$\bar{Q}_n$	↳ Toggle

characteristic table

Q_n	J	K		Q_{n+1}
0	0	0	①	0
0	0	1	②	0
0	1	0	③	1
0	1	1	④	1
1	0	0	①	1
1	0	1	②	0
1	1	0	③	1
1	1	1	④	0

Q_n \ JK	00	01	11	10
0			1	1
1	1			1

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

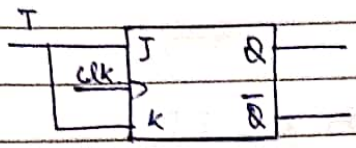
$J = K = 0 \quad Q_n = 0$

$\therefore Q_{n+1} = 1 \cdot 0 + 0 \cdot 1$

$= 0$

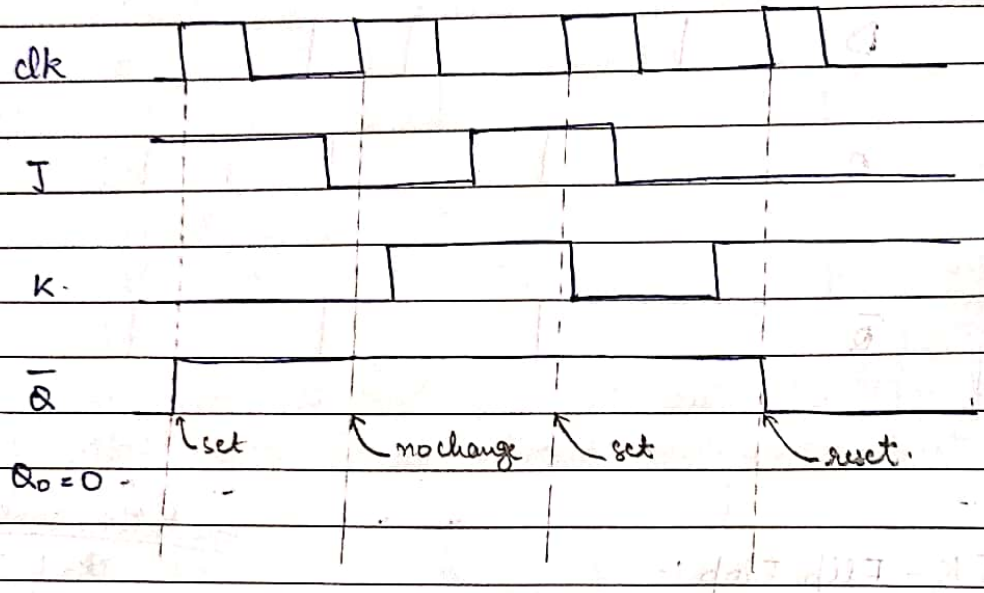
Scanned with CamScanner

→ Toggle Flip-Flop :-



clk	T	Q_{n+1}	
↑	0	Q_n	no change
↑	1	$\bar{Q}_n$	toggle.

Q. Determine the waveform of Q & $\bar{Q}$



characteristic table.

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

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

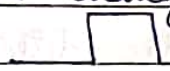
→ Triggering of FF :-

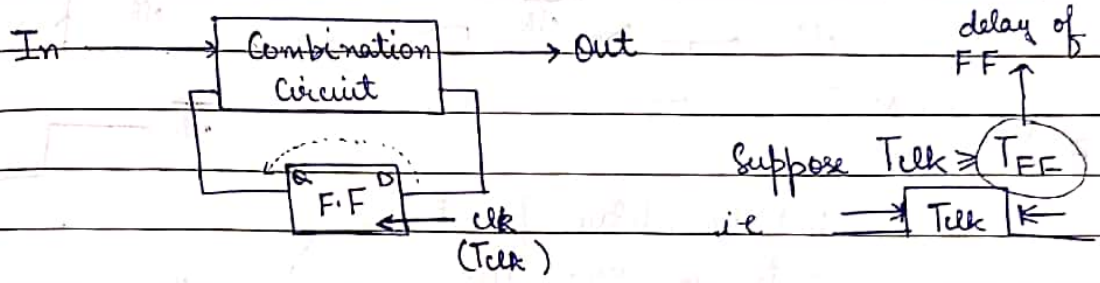
The momentary change in the input that switches the state of a FF is known as trigger

Latch of direct coupled FF → Trigger is change in input signal.

Scanned with CamScanner

FF → Trigger is change in clock pulse

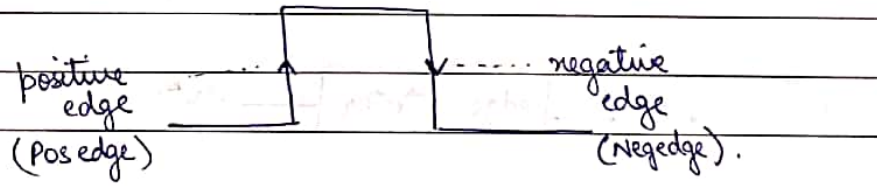




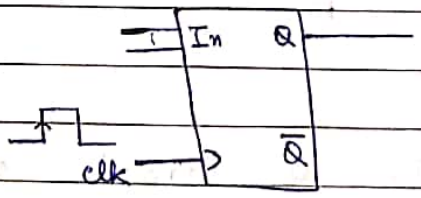
If $T_{clk} > T_{FF}$ instability may occur & to avoid this we should use Tclk smaller than the delay of FF

i.e $\boxed{T_{clk} \leq T_{FF}}$

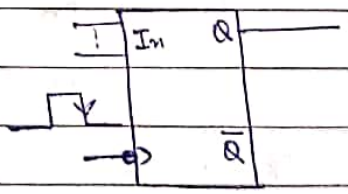
* Another way to avoid this feedback timing problem is to make FF sensitive to the pulse transition rather than pulse duration.



→ Making the Flip Flop edge triggered.



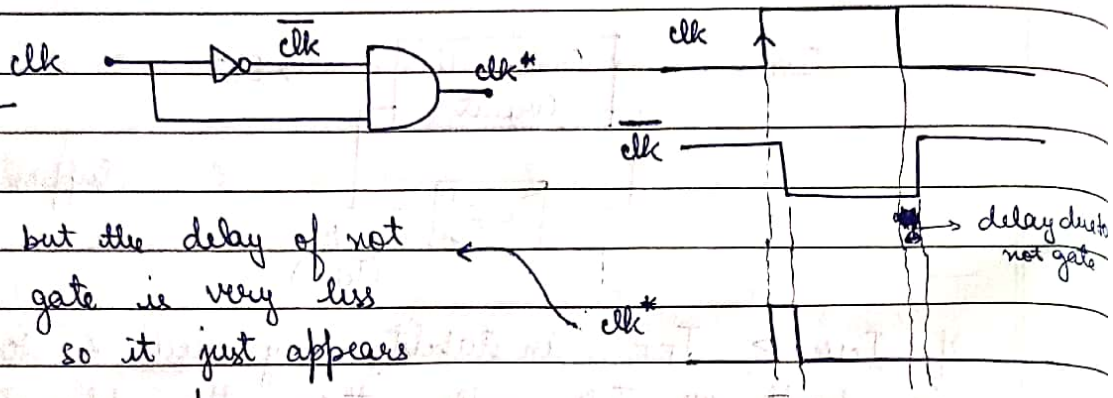
The output will change only at posedge of the clock

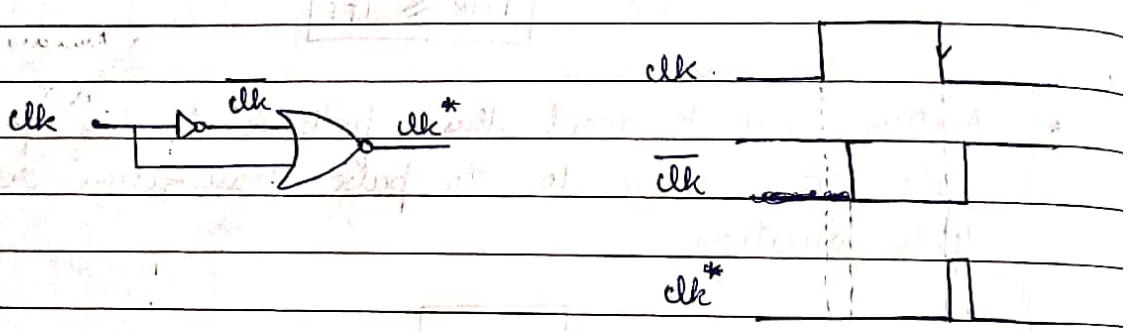


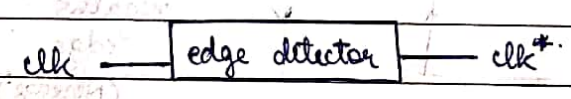
The output will change only negedge of the clock.

Scanned with CamScanner

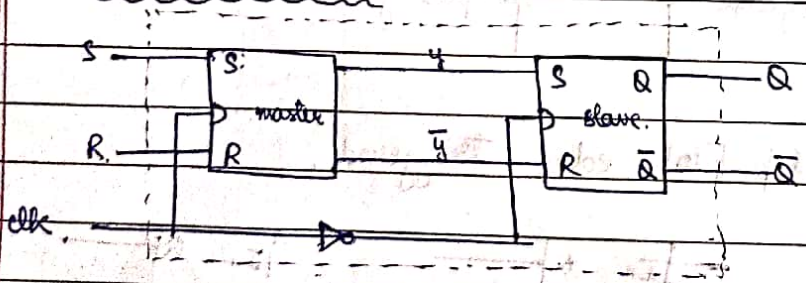
⇒ Implementation of Edge-detector:-







* Another way of having edge triggering is use of Master-slave FF.



M-S FF.

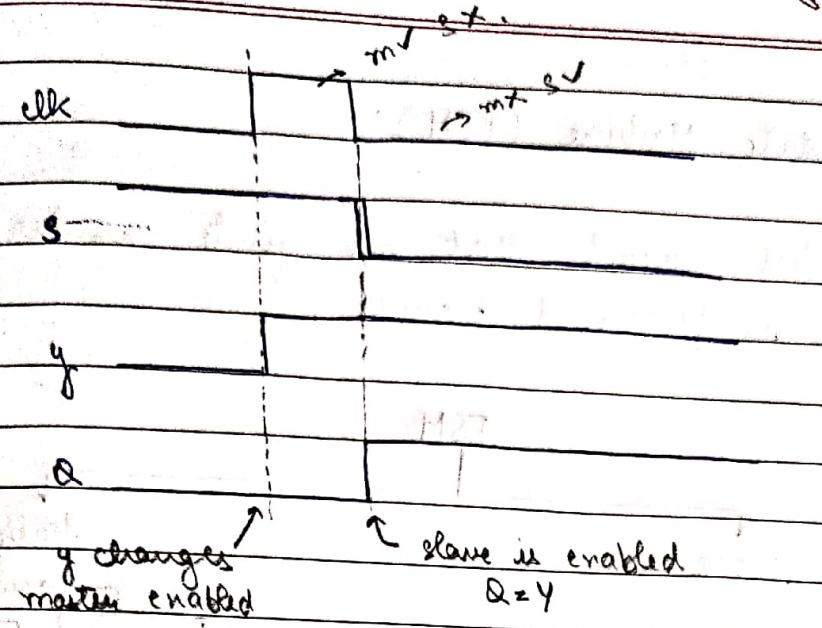
clk = 0 : master is disabled, slave is enabled.

$Q = y$, $\bar{Q} = \bar{y}$

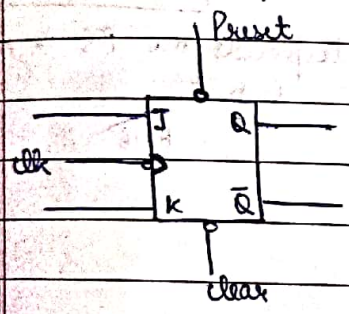
clk = 1 : master is enabled, slave is disabled

$y = S$, $\bar{y} = R$

Scanned with CamScanner



=> FF with special inputs :-



If clear = 0 $Q_{n+1} = 0$

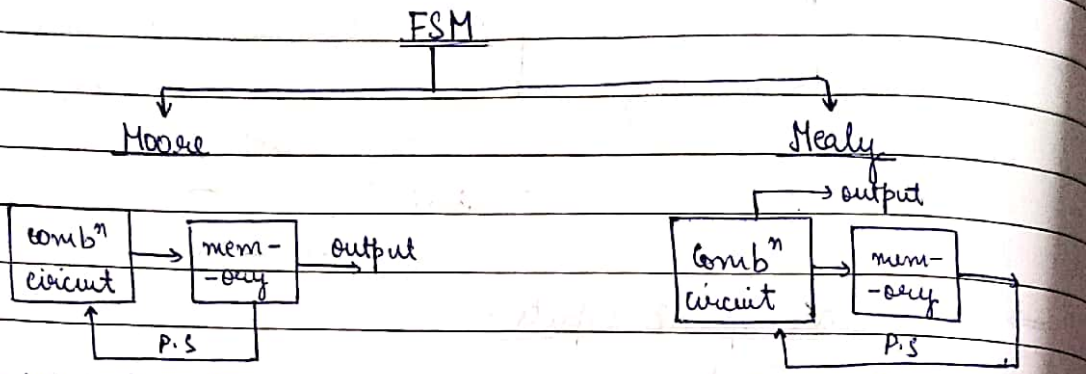
If preset = 0 change Q_{n+1} i.e on [↓]

	clear	Preset	clk	J	k	Q_{n+1}	$\bar{Q}_{n+1}$
	0	1	×	×	×	0	1
	1	0	×	×	×	1	0
Invalid →	0	0				?	?
no change	1	1	↓	0	0	Q_n	$\bar{Q}_n$
reset	1	1	↓	0	1	0	1
set	1	1	↓	1	0	1	0
Toggle	1	1	↓	1	1	$\bar{Q}_n$	Q_n
no change	1	1	↑/×	×	×	Q_n	$\bar{Q}_n$

Scanned with CamScanner

⇒ Finite State Machine (FSM) :-

A sequential circuit that has finite number of states occuring in prescribed manner.



O/P depends only on present state of FF.

eg:- 6.19

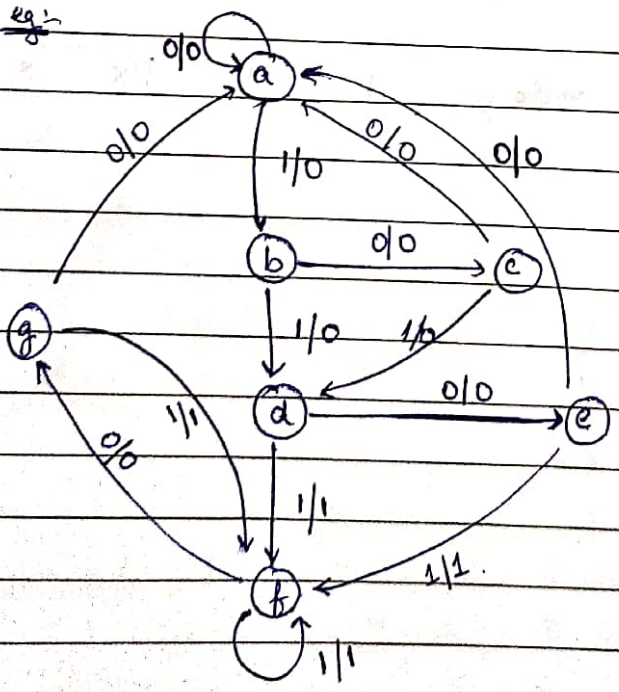
O/P depends on PS + external input.

eg:- 6.16

⇒ State-Reduction :-

It is ~~the~~ to find ways to reduce the number of state in a sequential circuit w/o altering the input/output relation.

eg:-



state diagram

Scanned with CamScanner

State Table

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g e	f	0	1
g	a	f	0	1

→ State Reduction Algorithm :-

The 2 states are said to be equivalent if for each no. of the set of inputs they give exactly the same output & send the circuit either to same state of equivalent state.

When 2 states are equivalent, one of them can be removed without altering the input output relation.

For x = 0 a, c, e, g ⟶ a at output = 0

For x = 1 a, c, e, g ⟶ f at output = 1

b d f f

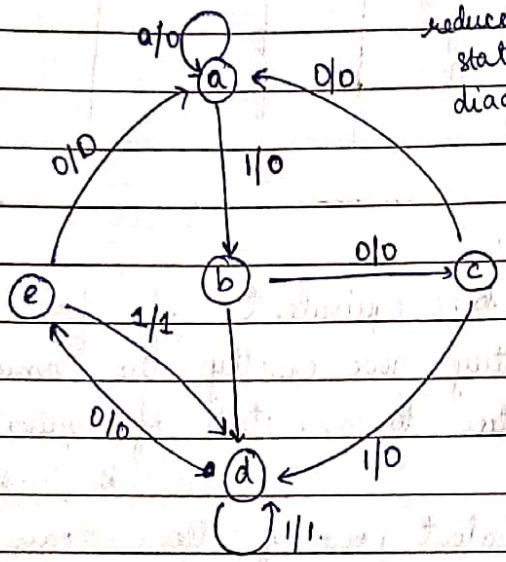
→ ∴ e & g are equivalent state as they are producing the same output & sending the circuit to same state for all the condition of input. Remove (g) from down to up & if we encounter g we replace it with e.

→ d & f are equivalent state.

Scanned with CamScanner

→ Reduced state table

PS	NS		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1



reduced state diagram.

reduced state from 7 to 5

$2^3 = 8$ ← 3 FF → 1 unused state

$2^3 = 8$ ← 3 FF → 3 unused state

∴ It is likely to reduce no. of gates in comb" part.

→ State Assignment :-
giving some binary number to the states.

State	Assignment 1	Assignment 2	Assignment 3
a	001	000	000
b	010	001	010
c	011	010	011
d	100	101	110
e	101	111	111.

Scanned with CamScanner

The best assignment is the one which results in min number of gates required.

eg:- Assignment 1 ✓

State table :-

P.S	N.S		Output	
	x=0	x=1	x=0	x=1
001	001	010	0	0
010	011	100	0	0
011	001	100	0	0
100	101	100	0	1
101	001	100	0	1

⇒ FF excitation table :-

RS

	Qn	Qn+1	S	R
Reset ↳01 / No change ↳00	← 0	0	0	X → don't care [can either be 0 or 1]
Set	← 0	1	1	0
Reset	← 1	0	0	1
Set ↳10 / No change ↳00	← 1	1	X	0

JK

	Qn	Qn+1	S	R
Reset 01 / No change 00	← 0	0	0	X
Set 10 / Toggle 11	← 0	1	1	X
Reset 01 / Toggle 11	← 1	0	X	1
Set 10 / No change 00	← 1	1	X	0

D

Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

Scanned with CamScanner

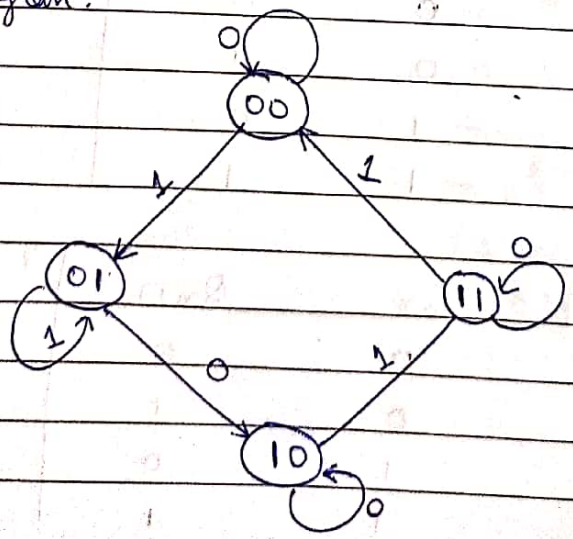
T	Q_n	Q_{n+1}	T
No change ←	0	0	0
Toggle ←	0	1	1
Toggle ←	1	0	1
No change ←	1	1	0

⇒ **Design Procedure of Sequential Circuit :-**

Step -1 :- Description of circuit behaviour
eg :- state diagram.

Step -2 :- obtaining the state table.
Step -3 :- state reduction (if possible)
Step -4 :- state assignment (if required)
~~Step 5~~ (determining the number of FF).
Step -5 :- Determining the numbe of FF → m FF can represent upto 2^m states.
Step -6 :- choosing a proper FF.
Step -7 :- determining the circuit output function & FF input function. (use K-Maps)
Step -8 :- Drawing the logic diagram.

Ex- Using JK Flip Flop to implement the following state diagram.



Scanned with CamScanner

Present state		Input	Next state		Flip Flop input Func.			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	1	0	1	x	x	1
0	1	1	0	1	0	x	x	0
1	0	0	1	0	x	0	0	x
1	0	1	1	1	x	0	1	x
1	1	0	1	1	x	0	x	0
1	1	1	0	0	x	1	x	1

Required no. of FF = 2.

J_A

A \ Bx	00	01	11	10
0				1
1	x	x	x	x

$J_A = B\bar{x}$

K_A

A \ Bx	00	01	11	10
0	x	x	x̄	x
1			1	

$K_A = Bx$.

J_B

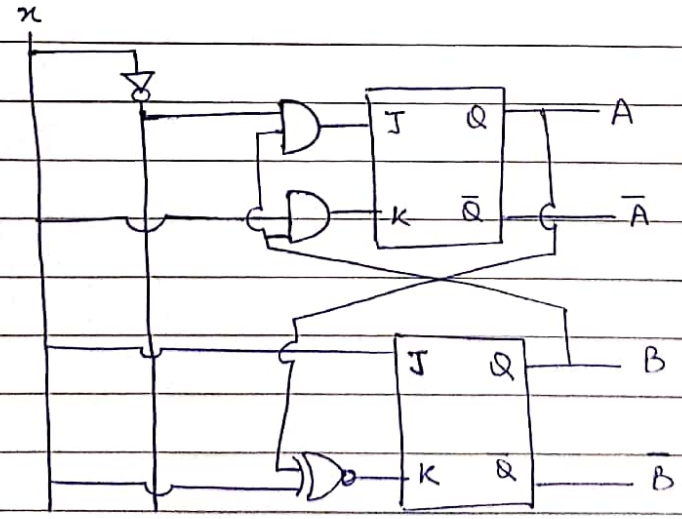
A \ Bx	00	01	11	10
0		1	x	x
1		1	x	x

$J_B = x$

K_B

A \ Bx	00	01	11	10
0	x	x		1
1	x	x	1	

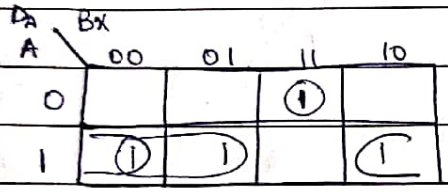
$K_B = \bar{A}x + Ax$.



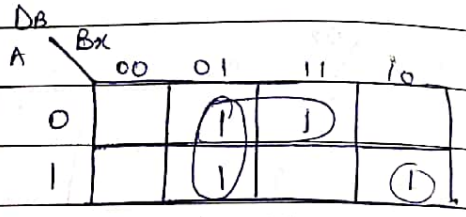
Scanned with CamScanner

→ D - Flip Flop :-

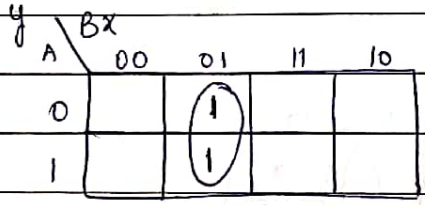
Present state		input	Next State		Output		
A	B	x	A	B	y	D_A	D_B
0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1
0	1	0	1	0	0	1	0
0	1	1	0	1	0	0	1
1	0	0	1	0	0	1	0
1	0	1	1	1	1	1	1
1	1	0	1	1	0	1	1
1	1	1	0	0	0	0	0



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



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



$y = \bar{B}x$

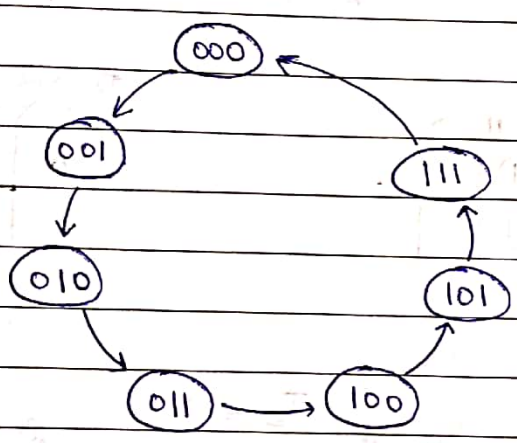
Scanned with CamScanner

→ **Design of Synchronous Counter :-**

A sequential circuit that goes through prescribed states in repeated fashion upon application of trigger input, known as clock pulse.

eg:- **Binary Counter :-**

- n input binary counter starts counting from 0 to $2^n - 1$
- 3 bit binary counter starts counting from 000 to 111 [0 to $\frac{7}{2^3-1}$]



- **designing 3-bit binary counter**

→ No. of FF = 3
→ Using T Flip Flop

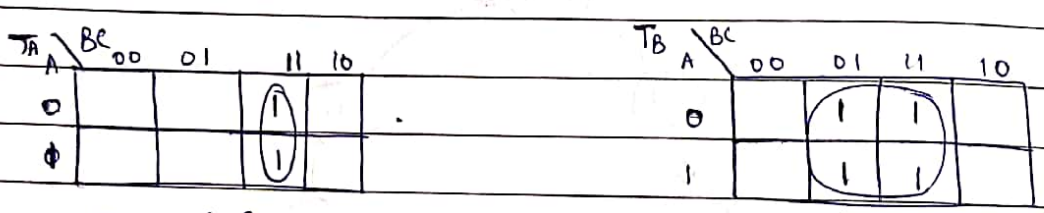
excitation table of T-FF :-

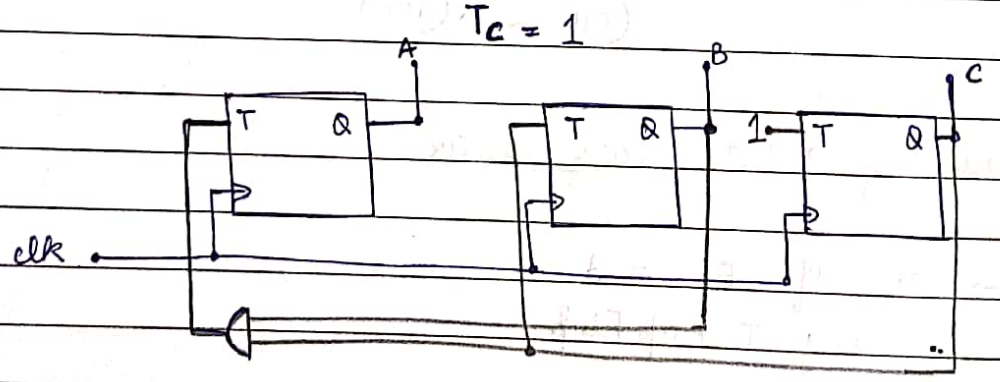
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Scanned with CamScanner

excitation table for counter

P.S			N.S			FF inputs		
A	B	C	A	B	C	T_A	T_B	T_C
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





3-bit binary counter

Q. Design a 3-bit binary counter using JK Flip Flop.

Scanned with CamScanner

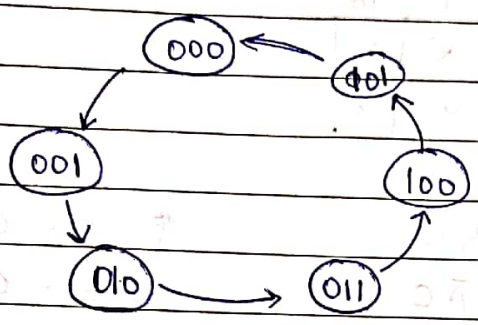
→ **Counter with non-binary sequence :-**

- The counter may have less than 2^{n-1} state

eg :-

3-bit binary counter.

000 to 101 to 000



- Using JK-FF :-

No. of FF = 3

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

excitation Table of Counter :-

P.S A	B	C	N.S A	B	C	FF inputs J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	0	0	1	x	x	1	x	1
1	0	0	1	0	1	x	0	0	x	1	x
1	0	1	0	0	0	x	1	0	x	x	1
1	1	0									
1	1	1									

Scanned with CamScanner

MID SEM → Till Multiplexer excluding VHDL ×

∴ $J_C = 1$ $K_C = 1$.

J_A

A \ BC	00	01	11	10
0	0	0	1	0
1	x	x	x	x

K_A

A \ BC	00	01	11	10
	x	x	x	x
	0	1	x	x

J_B

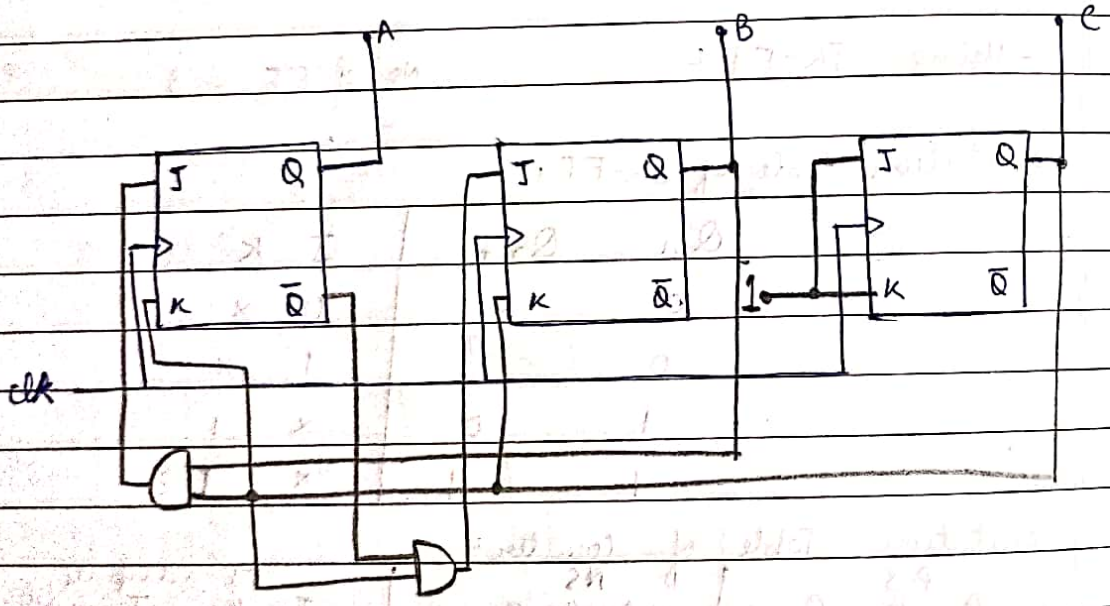
A \ BC	00	01	11	10
0	0	1	x	x
1	0	0	x	x

K_B

A \ BC	00	01	11	10
0	x	x	1	0
1	x	x	x	x

$J_A = BC$ $K_A = C$

$J_B = \bar{A}C$ $K_B = C$



Q. Design a BCD (0 to 9) counter [4 bit] using Toggle F.F

Scanned with CamScanner