Number System :

In digital electronics, the number system is used for representing the Information.

Types of Number Systems :-

1. Decimal Number system
2. Binary Number system
3. Octal Number system
4. Hexadecimal Number system.

1. Decimal Number System

- The number system is having digits 0,1,2,3,4,5, 6,7,8,9, this number system is known as "Decimal Number system".
- Base of decimal number system is 10 (0-9)

2. Binary Number System

- The number system which uses only two digits 0 and 1 is called "Binary Number system".
- Base of Binary number system is 2 (0,1).

3. Octal Number system

- The number system which is having digits 0,1,2,3,4,5,6,7 this number system is known as "Octal Number system".
- Base of the octal number is 8 (0-7).

4. Hexa Decimal Number System

- The number systems which consists the following sixteen number of digits 0,1,2,3,4,5,6,7,8,9,A,B,C,D,
- Base of Hexadecimal n.s is 16 (0-9, A-F)

Table :-

The Table is shown below the Decimal, Binary, Octal and Hexadecimal numbers from 0 to 15 & their equivalent binary number.

Decimal	Binary	octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Number Conversions

1. Decimal – Binary (2):

Repeatedly divide the number with 2 considering the remainder (Bottom up approach)

Ex:- $(141)_{10} = (\quad)_2$

2	141	
2	70	– 1 – LSB (Least Significant Bit)
2	35	– 0
2	17	– 1
2	8	– 1
2	4	– 0
2	2	– 0
2	1	– 0
	0	– 1 – MSB (Most significant Bit)

$\therefore (141)_{10} = (10001101)_2$

2. Decimal to Octal (8)

Repeatedly divide the number with 8, considering the remainder.

Ex: $(141)_{10} = (\quad)_8$

8	141	
8	17	– 5
8	2	– 1
	0	– 2

$\therefore (141)_{10} = (215)_8$

3. Decimal - Hexadecimal

Repeatedly divide the number with 16, considering the remainder.

Ex:- $(141)_{10} = (\quad)_{16}$

16	141	
16	8	-13 - D
	0	- 8

$\therefore (141)_{10} = (8\,D)_{16}$

1. Binary - Decimal

Starting from LSB multiply each positional value with 2^n where n is position, and add all the results.

Ex:- $(1011)_2 = (\quad)_{10}$

1	0	1	1
2^3	2^2	2^1	2^0

$(1\times2^3)+(0\times2^2)+(1\times2^1)+(1\times2^0) = 8+2+1 = (11)_{10}$

$\therefore (1011)_2 = (11)_{10}$

2.

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

$512+64+32+8+2 = (618)_{10}$

$\therefore (1001101010)_2 = (618)_{10}$

2. Binary - Octal

Starting from LSB Group 3 Bits, convert Decimal equivalent for each group of 3 bits.

Ex:- $(10110)_2 = (\quad)_8$

$\underset{2^2\,2^1\,2^0}{010}\ \underset{2^2\,2^1\,2^0}{110}$

$= 26$

$\therefore (10110)_2 = (26)_8$

2. $(10001110)_2 \longrightarrow (\quad)_8$

$\underset{2^2\,2^1\,2^0}{010}\ \underset{2^2\,2^1\,2^0}{001}\ \underset{2^2\,2^1\,2^0}{110}$

$= 216$

$\therefore (10001110)_2 = (216)_8$

3. Binary - Hexadecimal

Starting from LSB Group 4 bits, convert decimal equivalent for each group of 4 bits

Ex:- $(101011110)_2 = (\quad)_{16}$

$\underset{2^3\,2^2\,2^1\,2^0}{0001}\ \underset{2^3\,2^2\,2^1\,2^0}{0101}\ \underset{2^3\,2^2\,2^1\,2^0}{1110}$

1 5 D

$\therefore (101011110)_2 = (15D)_{16}$

1. Octal - Binary

Every position must be represented in 3 bit group

Ex:- $(123)_8 = (\quad)_2$

$\underset{001}{1} \quad \underset{010}{2} \quad \underset{011}{3}$

$\therefore (123)_8 = (001010011)_2$

2. Octal to Decimal

From LSB multiply positional value with 8^n, where n is the position and add all results.

Ex:- $(1\ 2\ 3)_8 \rightarrow (\quad)_{10}$

$\underset{8^2}{1} \quad \underset{8^1}{2} \quad \underset{8^0}{3}$ $= (8^2 \times 1) + (8^1 \times 2) + (8^0 \times 3)$

$= 64 + 16 + 3$

$= 83$

$\therefore (123)_8 = (83)_{10}$

3. Octal - Hexadecimal

Step-1 : Convert octal to Binary

Step-2 : Convert Binary to Hexadecimal

Ex:- $(1\ 2\ 3)_8 = (053)_{16}$

Step-1: $\underset{001}{1} \quad \underset{010}{2} \quad \underset{011}{3} \Rightarrow (001\ 010\ 011)_2$

Step-2 : $\underset{0}{0000} \quad \underset{5}{0101} \quad \underset{3}{0011} \Rightarrow (053)_{16}$

II - Method

Step-1: Convert Octal to Decimal

Step-2 : Convert Decimal to Binary

$(123)_8 = \underset{8^1}{1} \quad \underset{8^1}{2} \quad \underset{8^0}{3}$

$= 64 + 16 + 3$

$= (83)_{10}$

$(83)_{10} = (\quad)_{16}$

$16 \mid 83$

$16 \mid 5 - 3$

$0 - 5$

$= (053)_{16}$

$\therefore (123)_8 = (053)_{16}$

1. Hexadecimal - Binary

Every position value must be represented in 4 bit group

Ex:- $(123)_{16} = (\quad)_2$

$\underset{0001}{1} \quad \underset{0010}{2} \quad \underset{0011}{3}$

$\therefore (123)_{16} - (000100100011)_2$

Ex:- $(A5)_{16}$

$\underset{1010}{A} \quad \underset{0101}{5}$

$(A5)_{16} = (10100101)_2$

2. Hexadecimal to Decimal

Starting from LSB multiply each positional value with 16^n, where n is the position and add all the results.

Ex:- $(123)_{16}$

1 2 3

16^2 16^1 16^0

$256 + 32 + 3$

$= (291)_{10}$

$\therefore (123)_{16} = (291)_{10}$

3. Hexadecimal to Octal

Step-1: Convert Hexadecimal to binary

Step-2: Convert binary to Octal

Ex:- $(123)_{16} = (\quad)_8$

$\frac{1}{0001}$ $\frac{2}{0010}$ $\frac{3}{0011}$ $\Rightarrow (0001\ 0010\ 0011)_2$

$\frac{000}{2^2 2^1 2^0}$ $\frac{100}{2^2 2^1 2^0}$ $\frac{100}{2^2 2^1 2^0}$ $\frac{011}{2^2 2^1 2^0}$ $= (0443)_8$

$\therefore (123)_{16} = (0443)_8$

Conversion of Fractional Decimal to Binary

→ Divide Integral part and Fractional part.

→ Convert Integral part to binary

→ Fractional part must be multiplied with 2.

→ Divide Integral part & Fractional part

→ Multiply fractional part with 2.

→ Repeat the same process until fractional part become '0'.

Fractional Part → '0' → Termination.

Fractional part is same with previous values → Stop.

Ex:- $(34.25)_{10}$ — ($)_2$

Integral part — 34

Fractional part - 0.25

2	34
2	17 - 0
2	8 - 1
2	4 - 0
2	2 - 0
2	1 - 0
	0 - 1

$34 \Rightarrow 100010$

	Integral part	Fractional part
$0.25 \times 2 = 0.5$	0	0.5
$0.5 \times 2 = 1.0$	1	0
$0 \times 2 = 0$		

$\therefore 0.25 \Rightarrow 01$

$\therefore (34.25)_2 = 100010.01$

Ex:- 34.45

Integral part - 34

Fractional part - 0.45

2 | 34	
2 | 17	- 0
2 | 8	- 1
2 | 4	- 0
2 | 2	- 0
2 | 1	- 0
0	- 1

$34 = 100010$

		I.P	F.P
$0.45 \times 2 = 0.9$	→	0	0.9
$0.9 \times 2 = 1.8$	→	1	0.8
$0.8 \times 2 = 1.6$	→	1	0.6
$0.6 \times 2 = 1.2$	→	1	0.2
$0.2 \times 2 = 0.4$	→	0	0.4
$0.4 \times 2 = 0.8$	→	0	0.8
$0.8 \times 2 = 1.6$	→	1	0.6

Repition

So stop here

$(0.45) = 011100$

$\therefore (34.45) = (100010.011100)$

Conversion of Fractional Binary to Decimal

Ex:- $(101101.110)_2 = (\quad)_{10}$

$(2^5 \times 1) + (2^4 \times 0) + (2^3 \times 1) + (2^2 \times 1) + (2^1 \times 0) + (2^0 \times 1)$

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

$= 32 + 8 + 4 + 1 + \frac{1}{2} + \frac{1}{4} + 0$

$= 32 + 13 + 0.5 + 0.25$

$= 45.75$

$\therefore (101101.110)_2 = (45.75)_{10}$

Logic Gates :-

- Logic gates are digital circuits
- The building blocks of any digital circuits system.
- It is an electronic circuit having one (or) more inputs and only one output.

Types:-

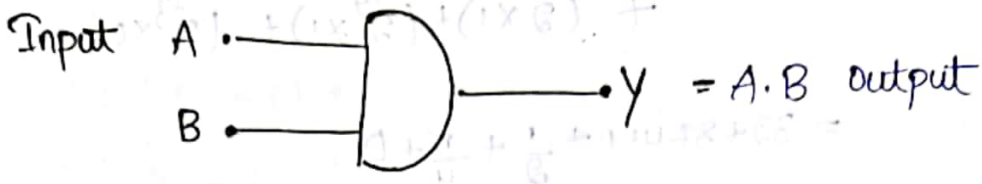
* Basic Gates
  1. AND Gates
  2. OR Gate
  3. NOT Gate

* Universal Gates:-
  4. NAND Gate
  5. NOR Gate

* Special Gates
  6. EXOR Gate 7. EXNOR Gate

i. AND Gate :-

- An AND Gate has two or more inputs and only one output which is equal to the product of all inputs.

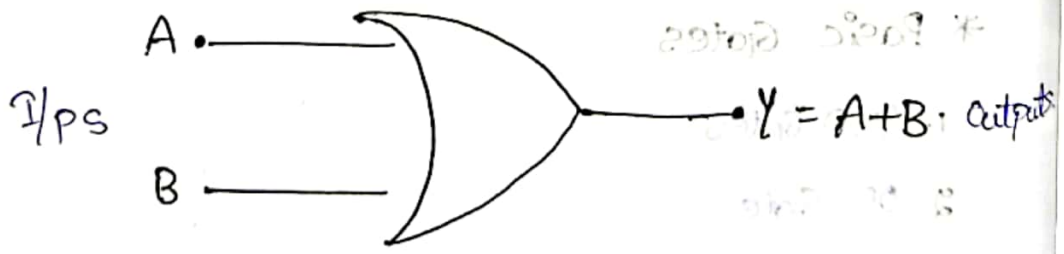


Truth Table :

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

ii. OR Gate :

- It has two or more inputs and only one output.
- The output is equal to the sum of all input.

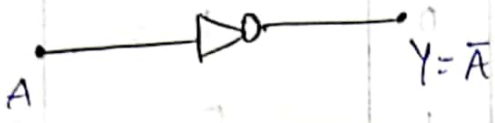


Truth Table:

A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

iii. NOT Gate (or) Inverter Gate

- Single input and single output.
- Output is the complement of the input logic.



Truth Table :-

Input (A)	Output ($\bar{A}$)
0	1
1	0

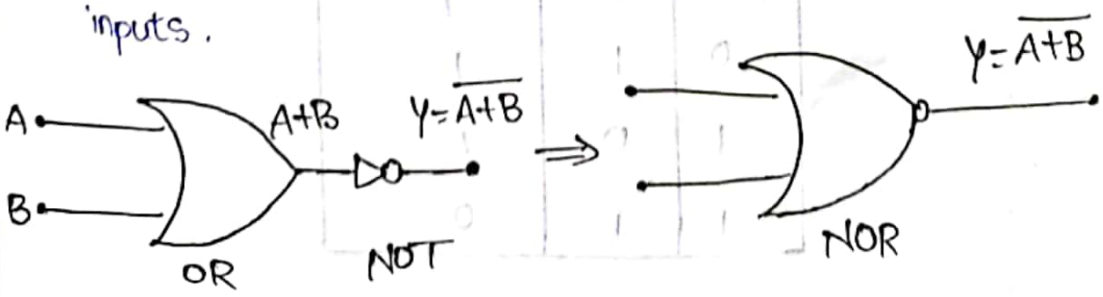
Universal Gates [NOR & NAND Gates]

* It is a gate which can be used to implement any other gates without using other types of gates.

- Ex:- NOR Gates
  NAND Gates

iv. NOR Gate :-

- NOT - OR
- It has two (or) more inputs but only one output.
- The output is the complement of the sum of the inputs.

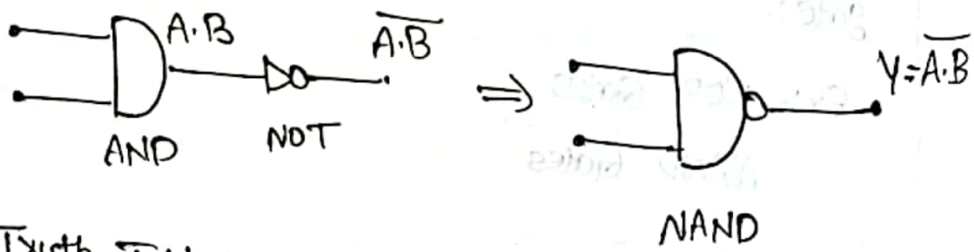


Truth Table :-

Inputs		output
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

V. NAND Gate :-

- (NOT-AND) (NAND Gate is the combination of NOT & AND).
- The output is the complement of the product of inputs.
- It has two or more inputs, but only one output.



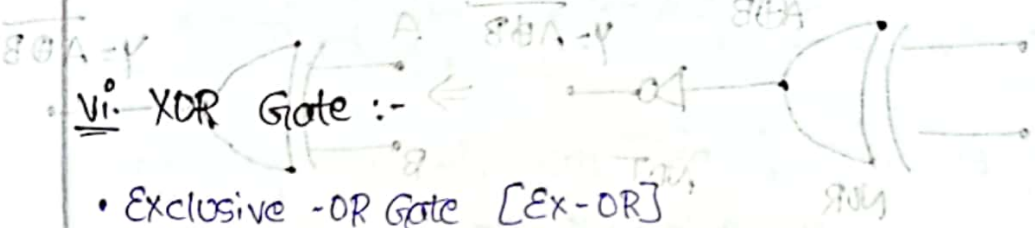
Truth Table :-

Inputs		output
A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Special Gates :-

* These are the exclusive gates used in the particular digital circuits.

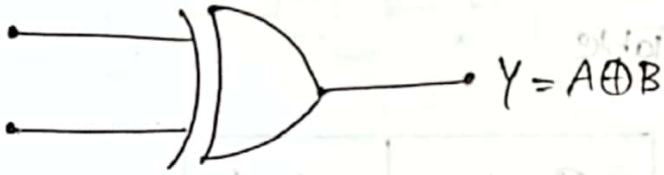
Ex:- XOR Gate
XNOR Gate



vi. XOR Gate :-

- Exclusive -OR Gate [Ex-OR]
- It has two or more inputs and only one output.
- Ex-OR operation can be represented with '⊕' symbol.

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



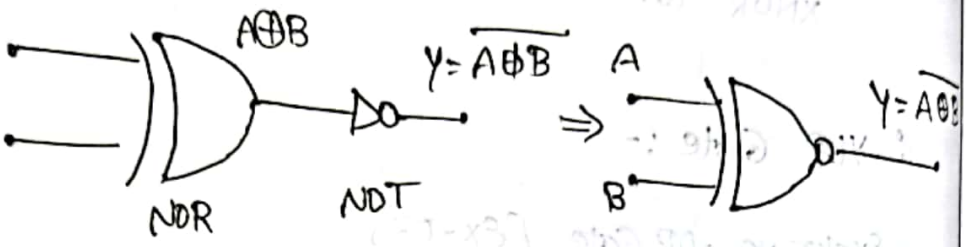
XOR GATE

Truth Table :-

Inputs		Output
A	B	Y = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

vii. XNOR Gate

- Exclusive - NOR Gate [Ex-NOR]
- It has two (or) more inputs and only one output
- The output is the complement of the XOR Gate



- So the operation of the XNOR Gate is given as

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

$$Y = A \odot B$$

Truth Table

Input		output
A	B	$Y = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

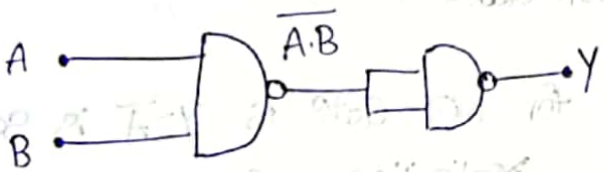
* Realization of Basic Gates (NOT, AND, OR) Using NAND Gate

1. NOT Gate

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

Logic equation for NOT Gate is $Y = \bar{A}$ is same as realisation eqn.

2. AND Gate



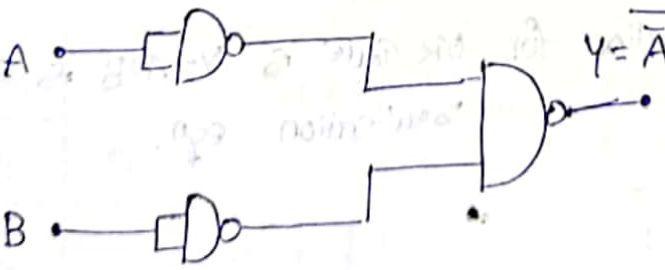
$= \overline{\overline{A \cdot B} \cdot \overline{A \cdot B}}$

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

$= A \cdot B$

Logic eqn for AND Gate is $Y = A \cdot B$ is same as realisation equation.

3. OR Gate



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

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

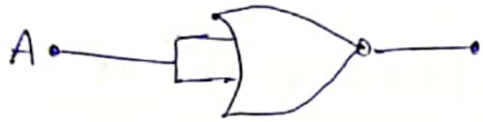
$Y = A + B$

Logic eqn for OR Gate is $Y = A + B$ is same as realisation eqn.

- For designing NOT Gate 1 NAND Gate is required.
- For designing AND Gate 2 NAND Gates are required.
- For designing OR Gate 3 NAND Gates are required.

* Realization of Basic gates (NOT, AND, OR) using NOR Gate

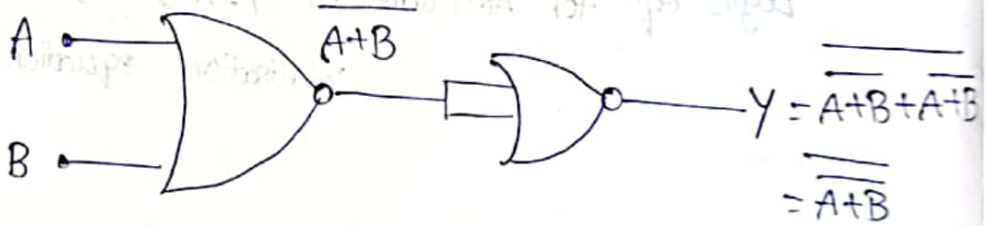
1. NOT Gate



NOR Gate

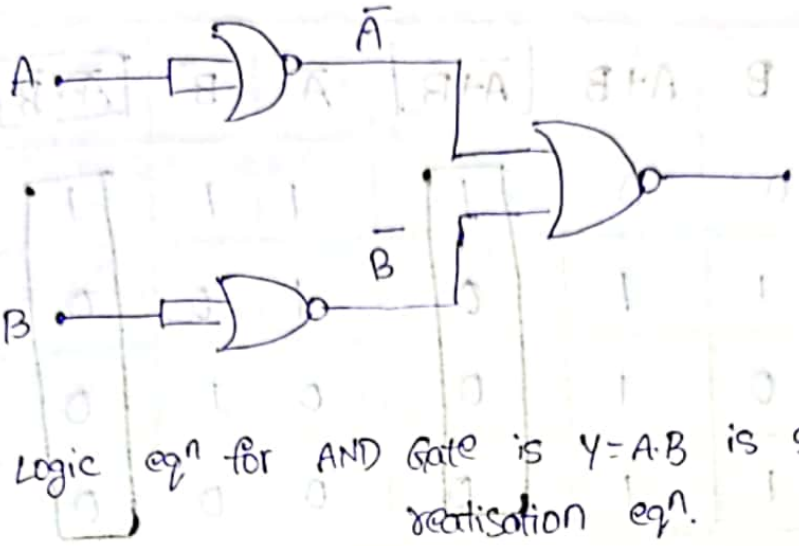
Logic equation for NOT Gate is $Y=\overline{A}$ is same as realisation eqn.

2. OR Gate



Logic equation for OR Gate is $Y=A+B$ is same as realisation eqn.

3. AND Gate :-



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

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

$= A \cdot B$

Logic eqn for AND Gate is $Y = A \cdot B$ is same as realisation eqn.

- For designing the NOT Gate 1 NOR Gate is required
- " " " OR " 2 NOR " are "
- " " " AND " 3 NOR " " "

De morgan Theoxem

Statement:- If you have two inputs, A & B, Then

1. $\overline{A \cdot B} = \bar{A} + \bar{B}$
2. $\overline{A + B} = \bar{A} \cdot \bar{B}$

* Proof of $\overline{A \cdot B} = \bar{A} + \bar{B}$

A	B	A·B	$\overline{A \cdot B}$	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

According to the Truth Table, we can verify that $\overline{A \cdot B} = \bar{A} + \bar{B}$

* Proof of $\overline{A+B} = \overline{A} \cdot \overline{B}$

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

According to the Truth table we can verify that

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

Boolean Laws

Identics :-

$A + \overline{A} = 1$

$A \cdot \overline{A} = 0$

$A + 1 = 1$

$A + A = A$

$A + 0 = A$

$A \cdot A = A$

$A \cdot 1 = A$

$A \cdot 0 = 0$

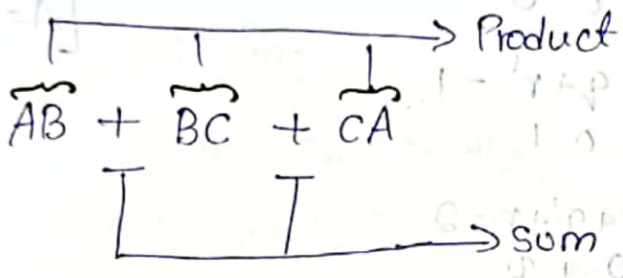
$\overline{\overline{A}} = A$

Boolean Equations

Boolean eqn's are are two Types

1. Sum of Product (SOP) expression
2. Product of sum (POS) expression

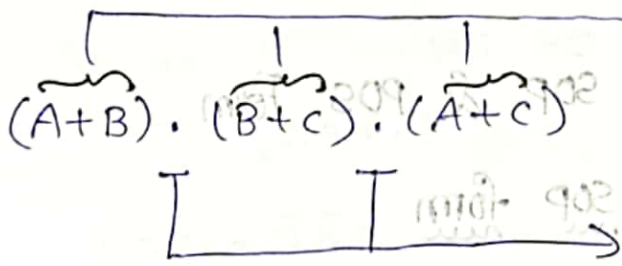
1. Sum of Product (SOP)

Ex:- 

[Format of SOP]

Each variable = literal

2. Product of Sum (POS)

Ex:- 

[Format of POS]

Canonical & Standard Forms

* Canonical [SOP and POS] Forms

Canonical SOP form

Canonical SOP form is the sum of min Terms in which, each product term contains all literals.

Ex:- $f = p'qr + pq'r + pqr' + pqr$

$0\,1\,1 \quad 1\,0\,1 \quad 1\,1\,0 \quad 1\,1\,1$

$3 \quad 5 \quad 6 \quad 7$

$f = \sum m(3,5,6,7)$ (Sum of minterms)

Canonical (POS) Form

Canonical POS form is the product of MAX terms, in which each sum term contain all literals.

Ex:-

$$f=(P+q+r)\cdot(P+q+r')\cdot(P+q'+r)\cdot(P'+q+r)$$

$P+q+r = 0$
$0\ 0\ 0$

$\begin{bmatrix} A'=1 \\ A=0 \end{bmatrix}$

$P+q+r' = 1$
$0\ 0\ 1$

$P+q'+r = 2$
$0\ 1\ 0$

$P'+q+r = 4$
$1\ 0\ 0$

$\bar{f} = \Pi M(0,1,2,4)$ Product of Max Terms.

Standard SOP & POS Form

- **Standard SOP form**

Standard SOP form is the standard sum of products form in which each product term need not contain all literals. So the product terms may or may not be the min terms

Ex = $f = pq+qr+pr$

Q. Convert the following boolean function into standard SOP form

$$f = p'qr + pq'r + pqr' + pqr$$

$$= pq(r'+r) + pr(q'+q) + qr(p+p')$$

$$\boxed{f = pq + pr + qr}$$

- **Standard POS form**

Standard POS form is the standard product of sums form. In which each sum term need not contain all literals. so the sum terms may or maynot be the Max terms.

Ex:- $f = (p+q).(q+r).(p+r)$

Q. Convert the sum of minterms into standard SOP form

$$f = \Sigma m(0,5,6,7)$$

$0 = 000 = \bar{A}\bar{B}\bar{C}$

$5 = 101 = A\bar{B}C$

$6 = 110 = AB\bar{C}$

$7 = 111 = ABC$

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

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

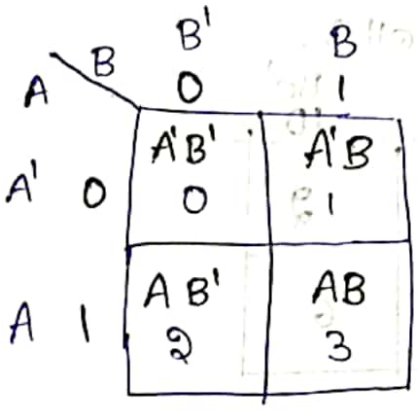
$$= AB + AC + \bar{A}\bar{B}\bar{C}$$

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

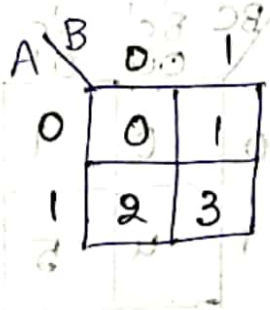
Karnaugh Map (K-Map)

- K-map is used for reducing boolean expressions.
- It is a graphical Method.
- It is made up of Rectangular boxes.
- Grey codes are used in this K-map.
- There are 2 variable 3 & 4 variables K-Map.

- Two Variable K-Map



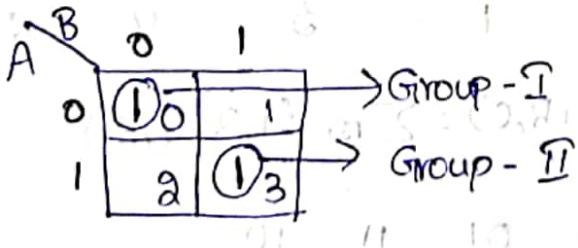
$2^n = 2^2 = 4$ cells



$A \rightarrow 1 \quad A' \rightarrow 0$

$B \rightarrow 1 \quad B' \rightarrow 0$

Ex-1 :- $F(A,B) = \sum m(0,3)$



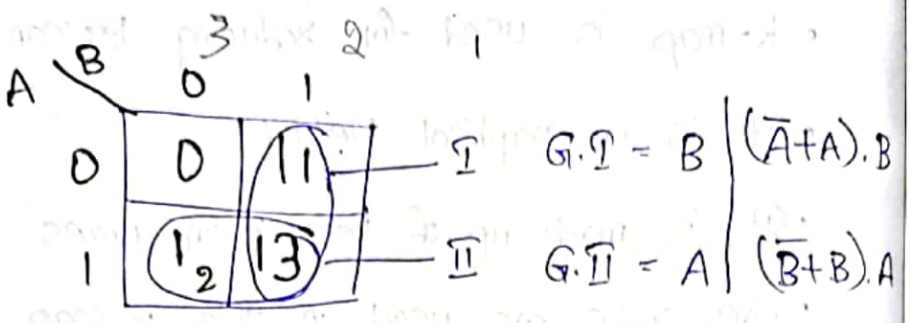
$G.I = \bar{A}\bar{B}$

$G.II = AB$

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

Ex-2 :- $F = A'B + A\bar{B} + \bar{A}'B$

1 1 1 0 0 1



- Three Variable K-Map

$2^n = 2^3 = 8$ cells.

A \ BC	B'C' 00	B'C 01	BC 11	BC' 10
A' 0	0	1	3	2
A 1	4	5	7	6

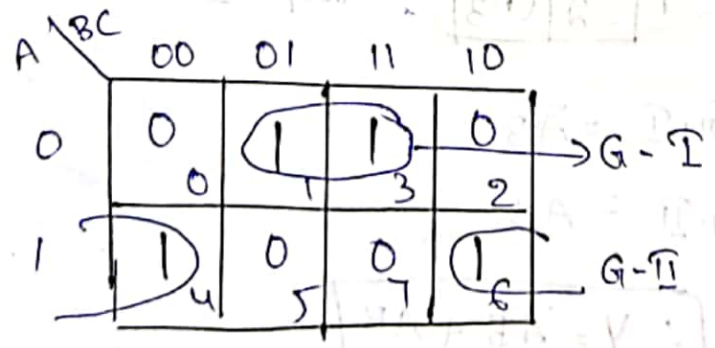
Ex-1 :- $f(A,B,C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C}$

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

0 0 1 0 1 1 1 0 0 1 1 0

1 3 4 6

$f(A,B,C) = \sum m\,(1,3,4,6)$



G-I - $\bar{A}C$

G-II - $A\bar{C}$

$\therefore f = \bar{A}C + A\bar{C}$

Ex-a:- $f = (A+B+C)\cdot(A+B+\bar{C})\cdot(A+\bar{B}+\bar{C})\cdot(\bar{A}+B+C)\cdot(\bar{A}+\bar{B}+C)\cdot(\bar{A}+\bar{B}+\bar{C})$ (POS form)

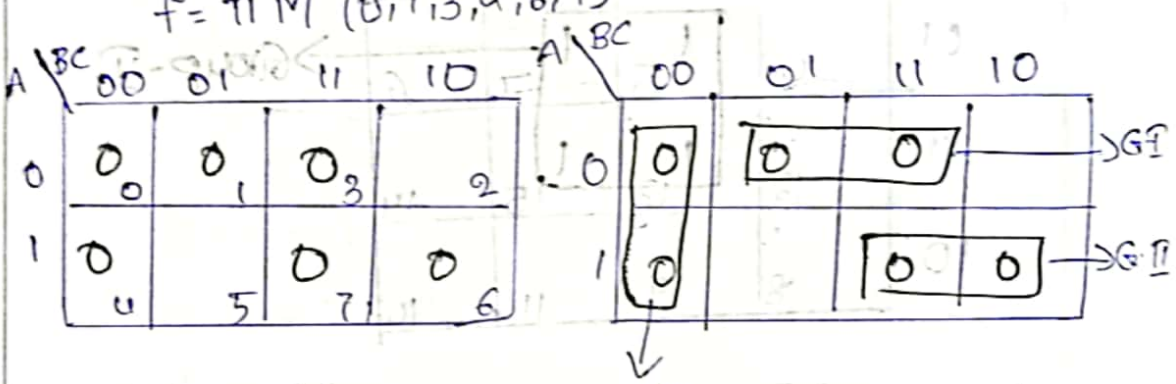
$f = (A+B+C)\cdot(A+B+\bar{C})\cdot(A+\bar{B}+\bar{C})\cdot(\bar{A}+B+C)\cdot(\bar{A}+\bar{B}+C)\cdot(\bar{A}+\bar{B}+\bar{C})$

$\frac{0\,0\,0}{0}$ $\frac{0\,0\,1}{1}$ $\frac{0\,1\,1}{3}$ $\frac{1\,0\,0}{4}$ $\frac{1\,1\,0}{6}$ $\frac{1\,1\,1}{7}$

$2^n = 2^3 = 8$ cells

n = no. of variables = 3.

$f = \Pi M\,(0,1,3,4,6,7)$.



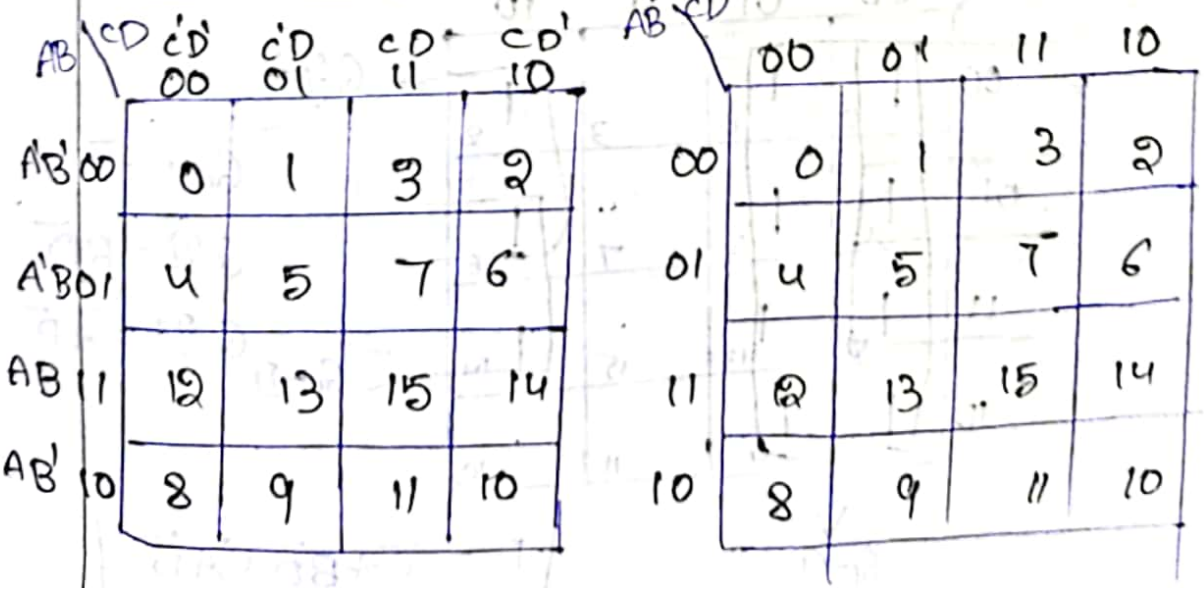
$G.I = A+\bar{C}$

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

$G.III = B+C$

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

- Four Variable K-Map

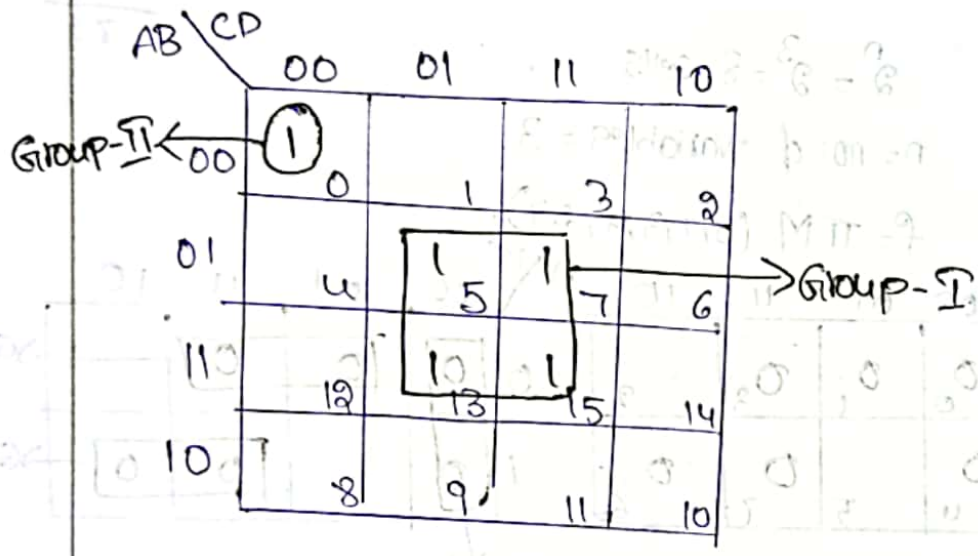


Ex:- $f = \bar{A}B\bar{C}D + AB\bar{C}D + \bar{A}BCD + ABCD + \bar{A}\bar{B}\bar{C}\bar{D}$ (SOP form)

$f = \bar{A}B\bar{C}D + AB\bar{C}D + \bar{A}BCD + ABCD + \bar{A}\bar{B}\bar{C}\bar{D}$

0101 1101 0111 1111 0000

5 13 7 15 0

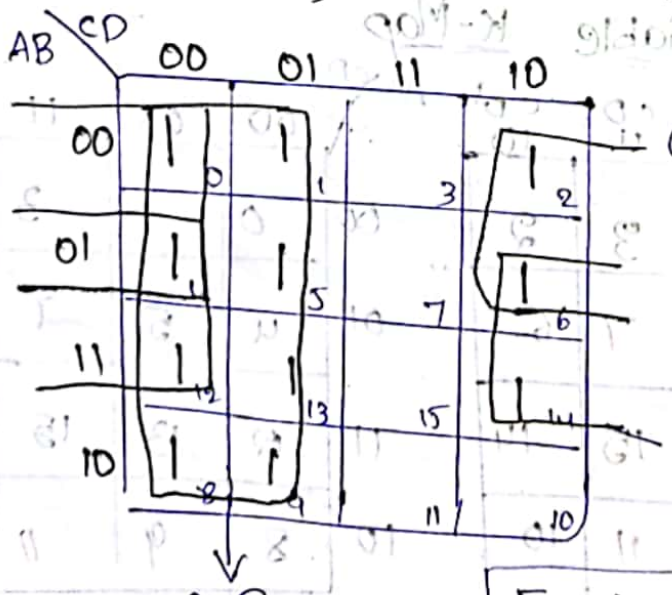


$G\cdot I = BD$

$G\cdot II = A'B'C'D'$

$f = A'B'C'D' + BD$

Ex:- $f(A,B,C,D) = \sum(0,1,2,4,5,6,8,9,12,13,14)$ SOP



$G_1\cdot I = \bar{C}$

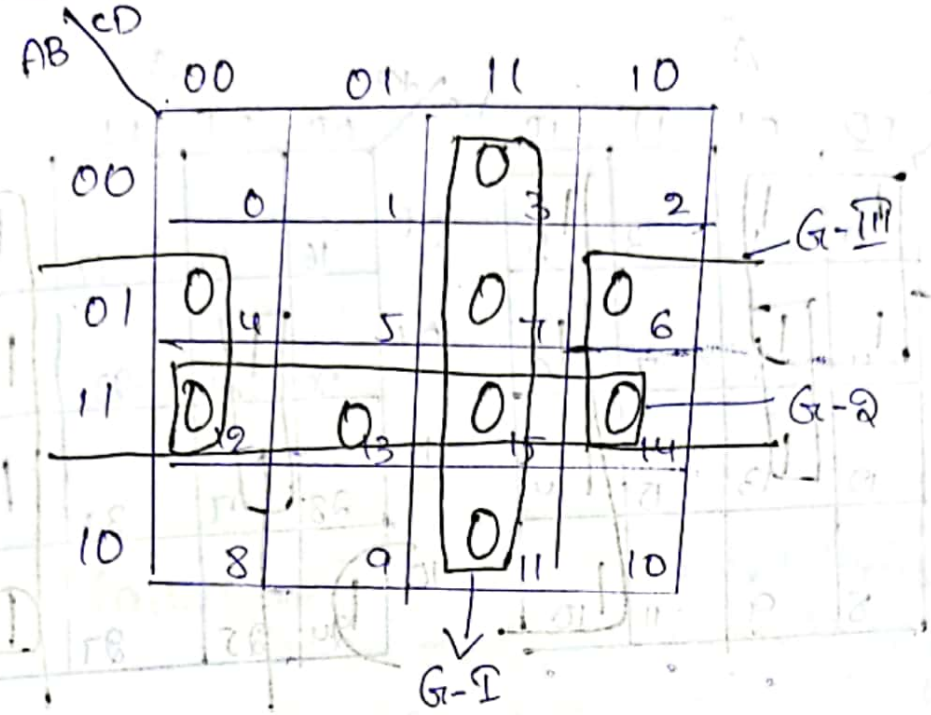
$G\cdot II = B\bar{D}$

$G\cdot III = \bar{A}\bar{D}$

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

Ex-3:- $f(A,B,C,D) = \Pi\ (3,4,6,7,11,10,13,14,15)$

POS.



$G-1 = \overline{C} + \overline{D}$

$G-2 = \overline{A} + \overline{B}$

$G-3 = \overline{B} + D$

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

*5 Variable K-Map

$2^n = 2^5 = 32$ squares.

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

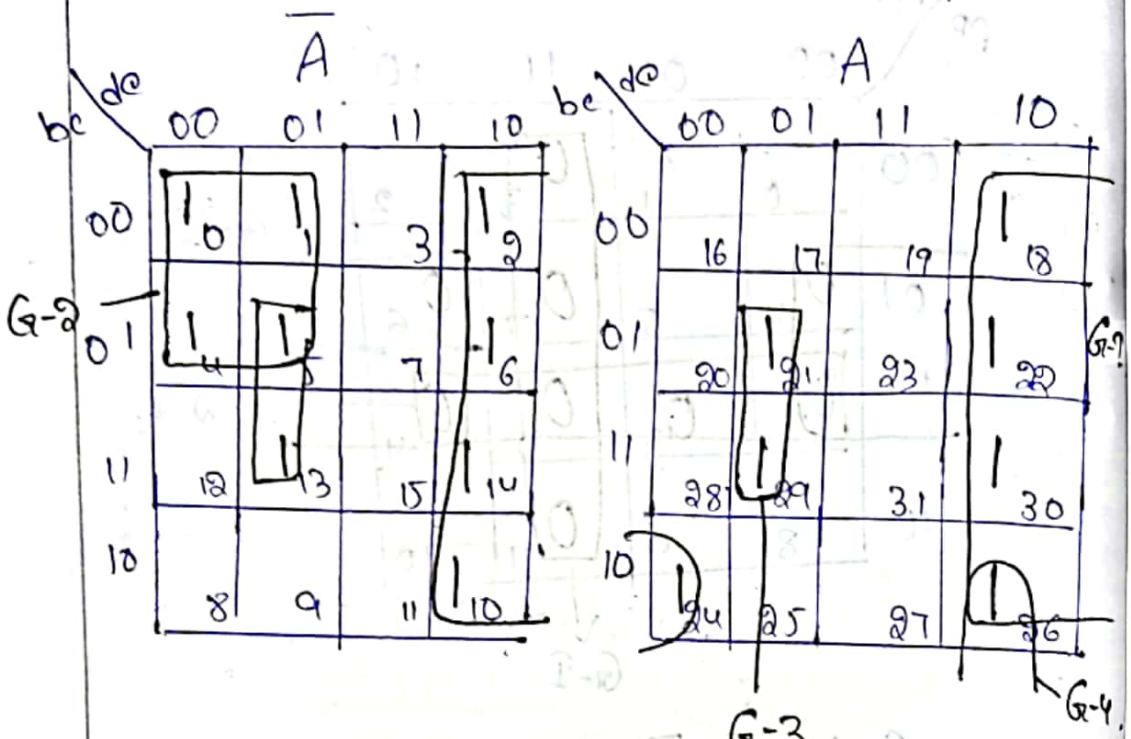
A = 1

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

Minterms 0 through 15 belongs with A=0

& 16 Through 31 belongs with A=1

Ex-1: $f(a,b,c,d,e) = \sum m(0,1,2,4,5,6,10,13,14,16,18,21,22,24,26,29,30)$



$G-1 = D\bar{E}$

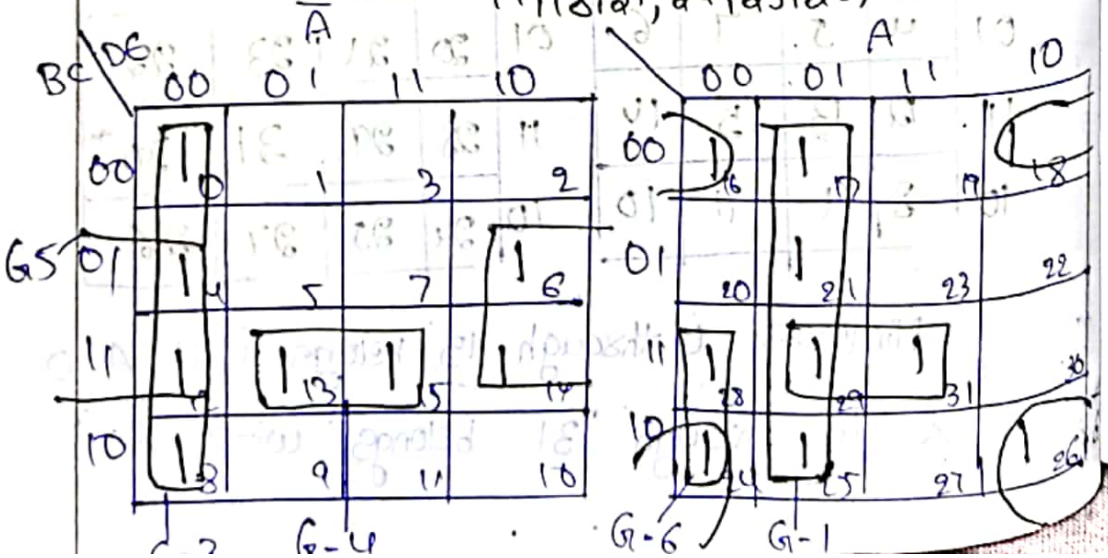
$G-2 = \bar{A}\bar{B}\bar{D}$

$G-3 = C\bar{D}E$

$G-4 = AB\bar{C}\bar{E}$

$\therefore f = D\bar{E} + \bar{A}B\bar{D} + C\bar{D}E + AB\bar{C}\bar{E}$

Ex-2: $f(a,b,c,d,e) = \sum m(0,4,6,8,12,13,14,15,16,17,18,21,24,25,26,28,29,31)$



G-1 : $A\bar{D}E$

G-2 : $A\bar{C}\bar{E}$

G-3 : $\bar{A}\bar{D}\bar{E}$

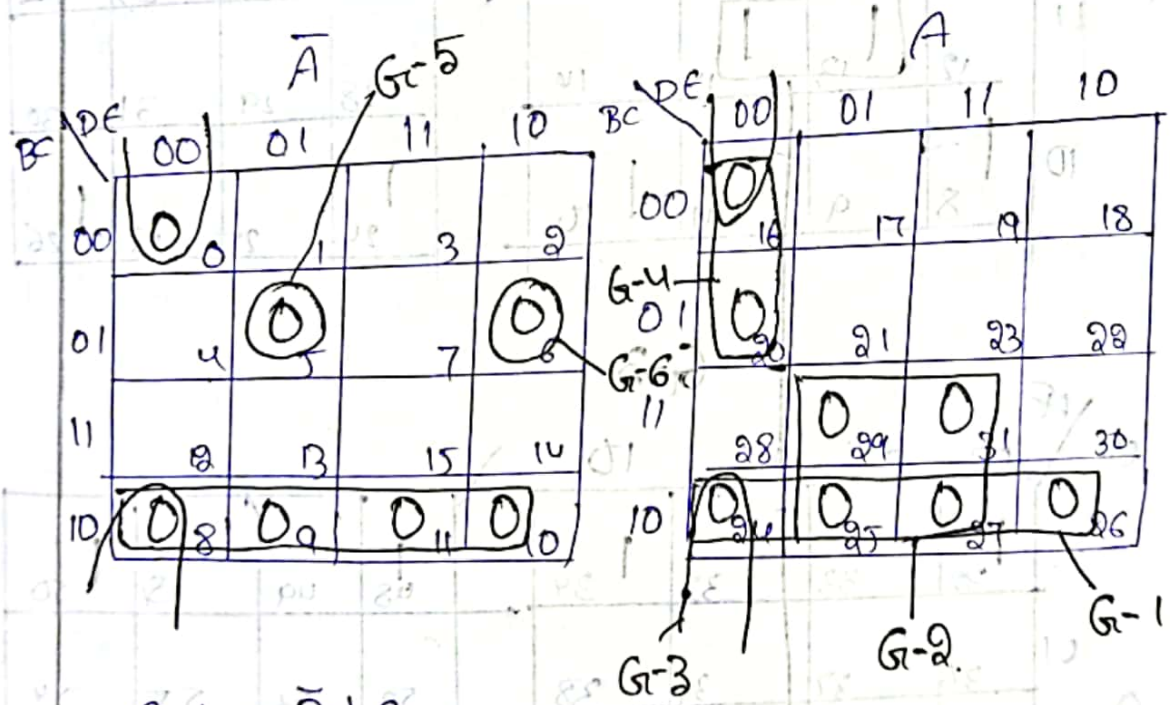
G-4 : BCE

G-5 : $\bar{A}C\bar{E}$

G-6 : $B\bar{D}\bar{E}$

$\therefore f = A\bar{D}E + A\bar{C}\bar{E} + \bar{A}\bar{D}\bar{E} + BCE + \bar{A}C\bar{E} + B\bar{D}\bar{E}$

Ex:- $f(a,b,c,d,e) = \Pi M(0,5,6,8,9,10,11,16,20,24,25,26,27,29,31)$, {POS}



G-1 = $\bar{B} + C$

G-2 = $\bar{A} + \bar{B} + \bar{E}$

G-3 = $C + D + \bar{E}$

G-4 = $\bar{A} + B + D + E$

G-5 : $A + B + \bar{C} + D + \bar{E}$

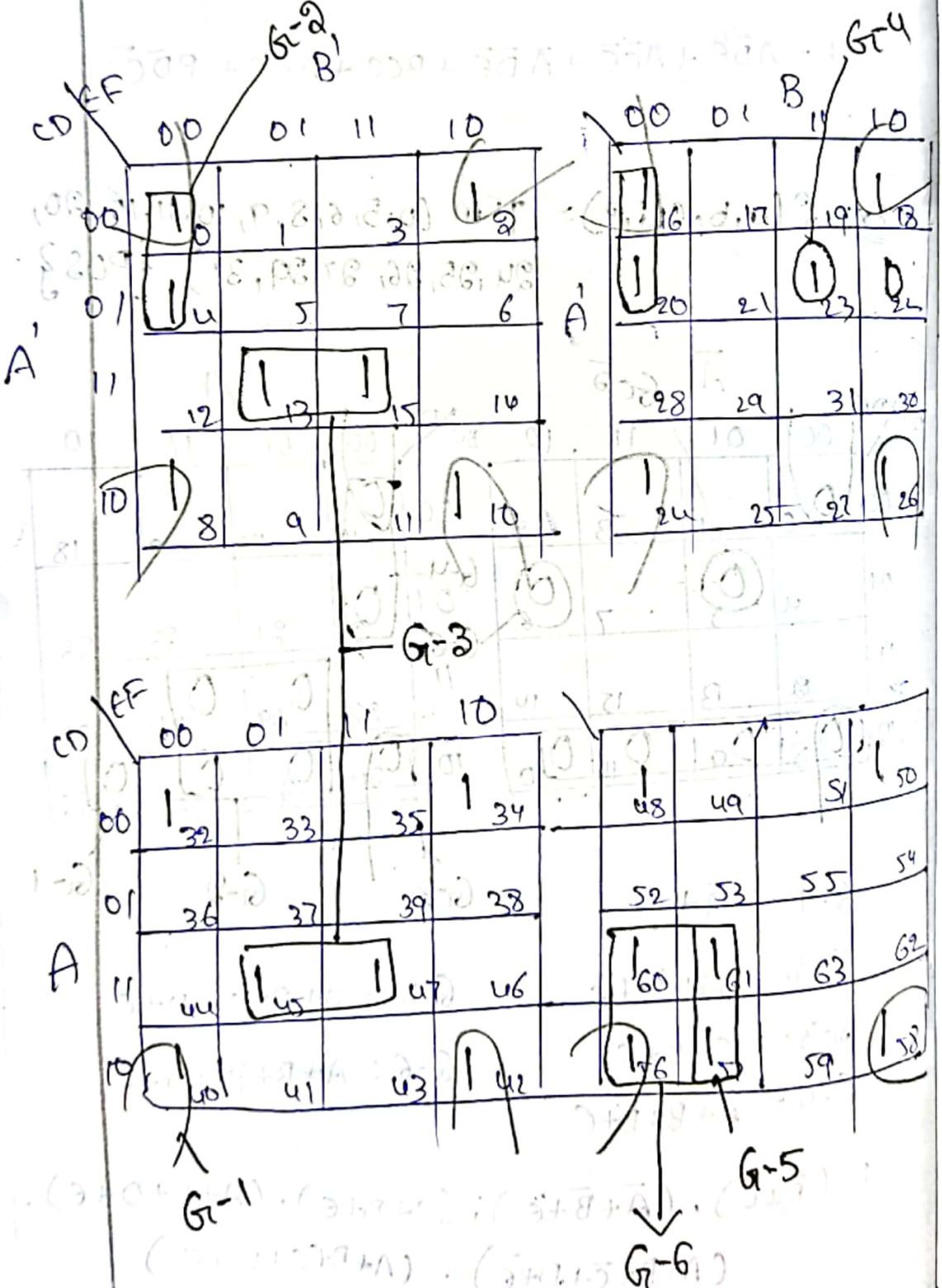
G-6 : $A + B + \bar{C} + \bar{D} + \bar{E}$

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

* Six - Variable K-Map (A, B, C, D, E, F).

$2^n = 2^6 = 64$ Squares

$f(A,B,C,D,E,F) = \sum (0,2,4,8,10,13,15,16,18,20,23,24,26,29,34,40,41,42,45,47,48,50,56,57,58,60,61)$.



G-1 = $\bar{D}\bar{F}$ G-4 = $\bar{A}B\bar{C}D\bar{E}F$

G-2 = $\bar{A}\bar{C}\bar{E}\bar{F}$ G-5 = $AC\bar{E}F$

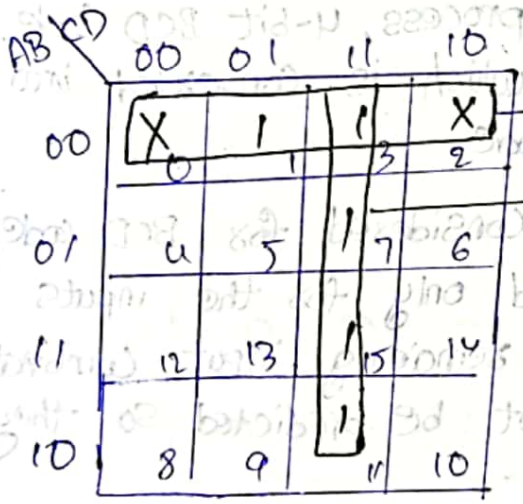
G-3 = $\bar{B}CDF$ G-6 = $ABC\bar{E}$

$\therefore F = \bar{D}\bar{F} + \bar{A}\bar{C}\bar{E}\bar{F} + \bar{B}CDF + \bar{A}B\bar{C}D\bar{E}F + AC\bar{E}F + ABC\bar{E}$

*Don't Care Condition in K-Map

- Don't care conditions are denoted by (X) and may be assumed "0" or "1" as per requirement for simplification.

Ex:- $Y = \Sigma m(1,3,7,11,15) + d(0,2,5)$

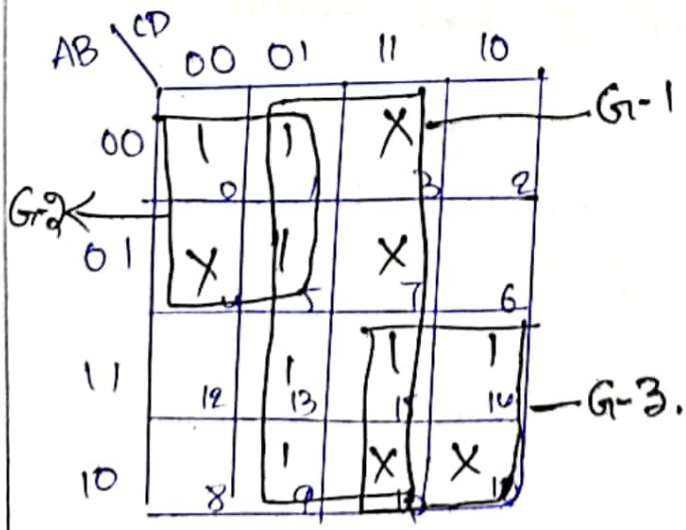


G-1 = CD

G-2 = $\bar{A}\bar{B}$

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

Ex:- $Y = \Sigma m(0,1,5,9,13,14,15) + d(3,4,7,10,11)$.



G-1 = D

G-2 = $\bar{A}\bar{C}$

G-3 = AC.

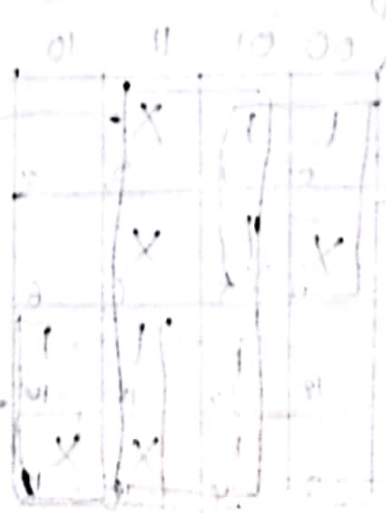
$Y = \bar{A}\bar{C} + AC + D$

Code Convertors

- The code convertors is used to convert one type of binary code to another.
- The code conversion is useful to provide security and to reduce the power dissipation in VLSI circuits.
- There are different types of Binary codes like BCD code, gray code, excess 3 code.
- BCD - Binary Coded Decimal.

BCD to Excess-3 code Convertor

- For this conversion process, 4-bit BCD code, is considered as input, which is converted into 4-bit excess 3-code.
- Since 4 bit is considered for BCD code, the output is produced only for the inputs from 0 to 9, for the remaining input combinations, the output cannot be predicted so they are don't care outputs.



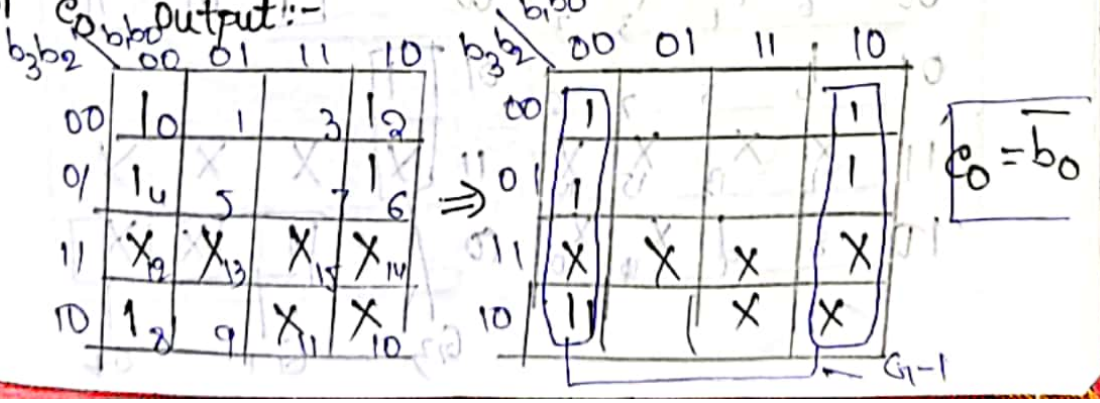
Truth Table :-

Decimal Number	BCD code Input				Excess				
	b_3	b_2	b_1	b_0	c_3	c_2	c_1	c_0	
0	0	0	0	0	0	0	1	1	3
1	0	0	0	1	0	1	0	0	4
2	0	0	1	0	0	1	0	1	5
3	0	0	1	1	0	1	1	0	6
4	0	1	0	0	0	1	1	1	7
5	0	1	0	1	1	0	0	0	8
6	0	1	1	0	1	0	0	1	9
7	0	1	1	1	1	0	1	0	10
8	1	0	0	0	1	0	1	1	11
9	1	0	0	1	1	1	0	0	12

From the Truth Table, the Minterms are obtained for each outputs (c_3, c_2, c_1, c_0)

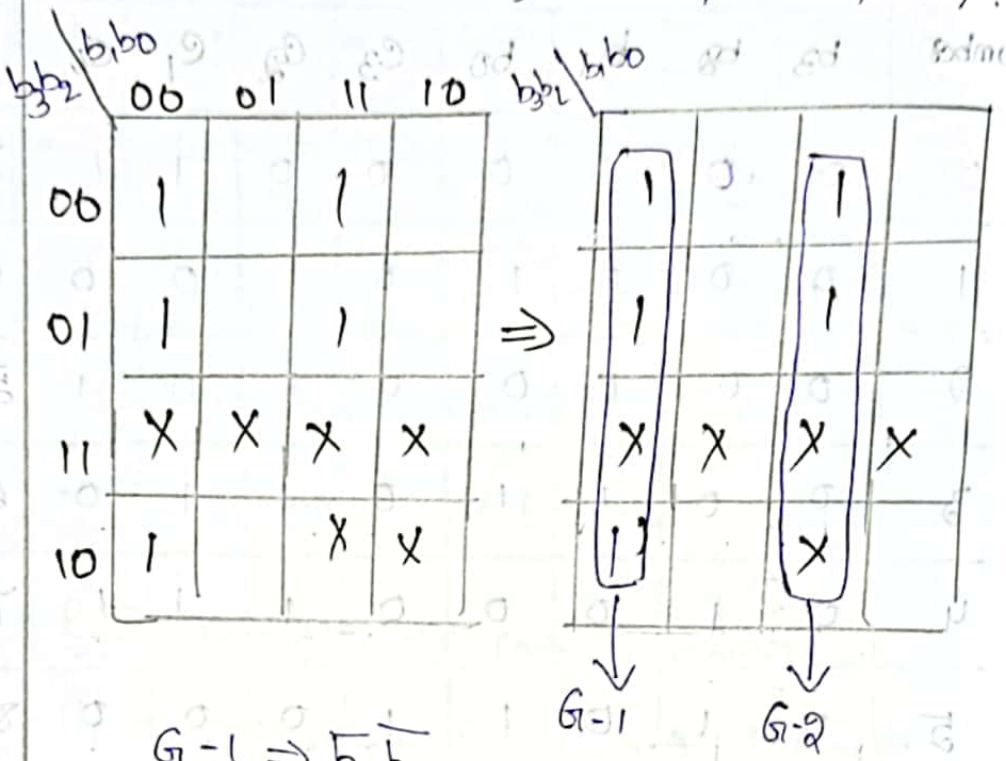
$$c_0 = \Sigma m(0,2,4,6,8) + D(10,11,12,13,14,15).$$

For c_0 output :-



For c_1 Output :-

$c_1 = \Sigma m(0,3,4,7,8) + D(10,11,12,13,14,15)$



$G-1 \Rightarrow \bar{b_1}\bar{b_0}$

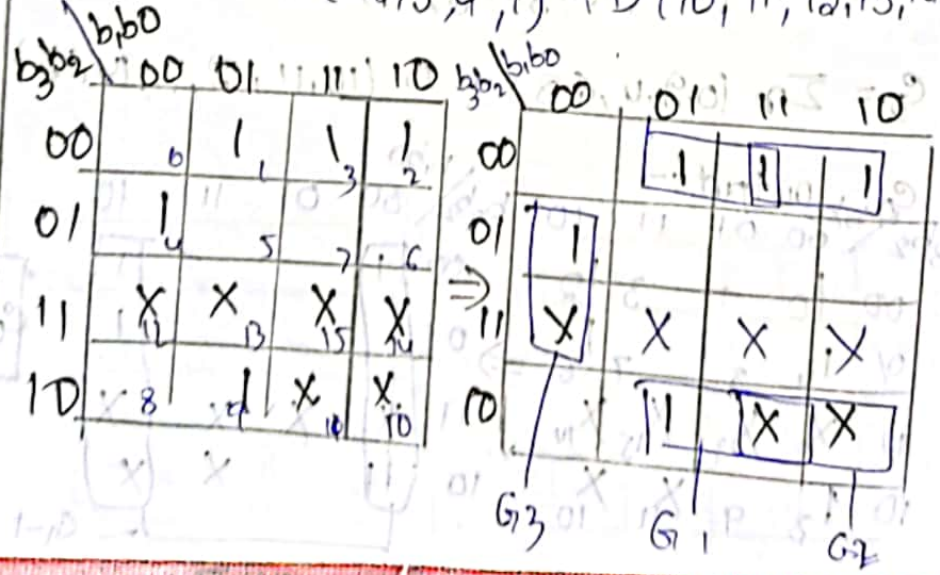
$G-2 \Rightarrow b_1 b_0$

$$c_1 = \bar{b_1}\bar{b_0} + b_1 b_0$$ (or)

$$c_1 = b_0 \odot b_1$$

For c_2 Output

$c_2 = \Sigma m(1,2,3,4,9) + D(10,11,12,13,14,15)$



$G\text{-}1 = \bar{b_2} b_0$

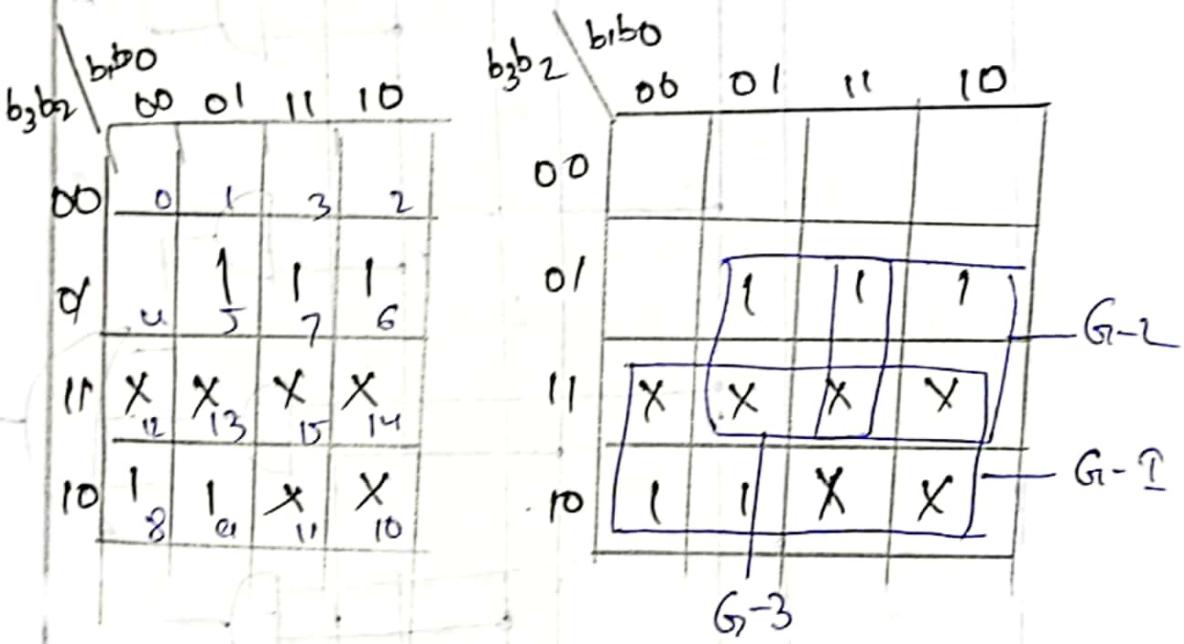
$G\text{-}2 = \bar{b_2} b_1$

$G\text{-}3 = b_2 \bar{b_1} \bar{b_0}$

$$C_2 = \bar{b_2} b_0 + \bar{b_2} b_1 + b_2 \bar{b_1} \bar{b_0}$$

For C_3 output

$$C_3 = \Sigma m\,(5,6,7,8,9) + D\,(10,11,12,13,14,15)$$



$G\text{-}1 = b_3$

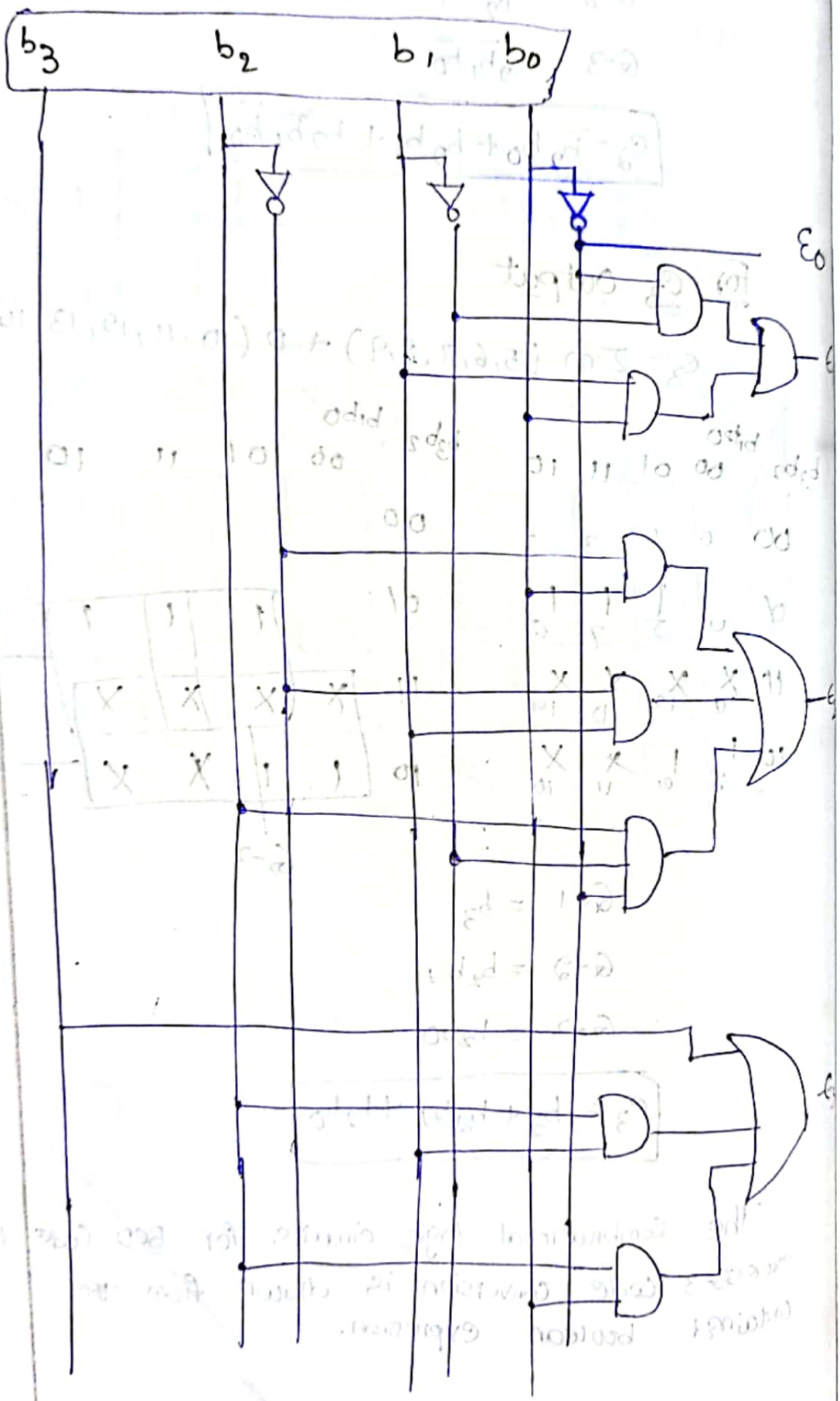
$G\text{-}2 = b_2 b_1$

$G\text{-}3 = b_2 b_0$

$$C_3 = b_3 + b_2 b_1 + b_2 b_0$$

The combinational logic circuits for BCD code to excess-3 code conversion is drown from the obtained booleon expression.

BCD Code (Input).



24/12/2021

Binary to Gray Code

MSB LSB → MSB LSB

$b_3 b_2 b_1 b_0 \longrightarrow g_3 g_2 g_1 g_0$

$g_3 = b_3$

$g_2 = b_2 \oplus b_3$

$g_1 = b_1 \oplus b_2$

$g_0 = b_0 \oplus b_2$

	0 0 0 0	0 0 0 0
	0 0 0 1	0 0 0 1
1	0 0 1 0	0 0 1 1
	0 0 1 1	0 0 1 0
2	0 1 0 0	0 1 1 0
	0 1 0 1	0 1 1 1
	0 1 1 0	0 1 0 1
	0 1 1 1	0 1 0 0

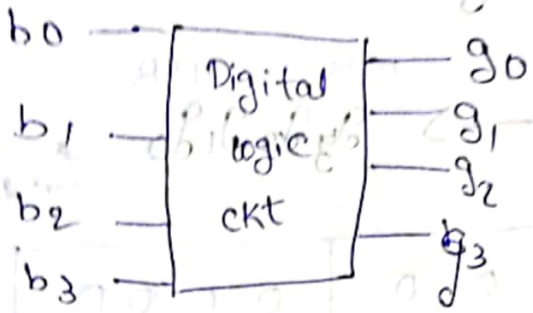
b_3	b_2	b_1	b_0	
0	0	0	1	} 2 Transition in binary
0	0	1	0	

g_3	b_2	g_1	g_0	
0	0	0	1	} 1 Transition.
0	0	1	1	

binary				
0	0	1	1	} 3
0	1	0	0	

Gray code.				
0	0	1	0	} 1
0	1	1	0	

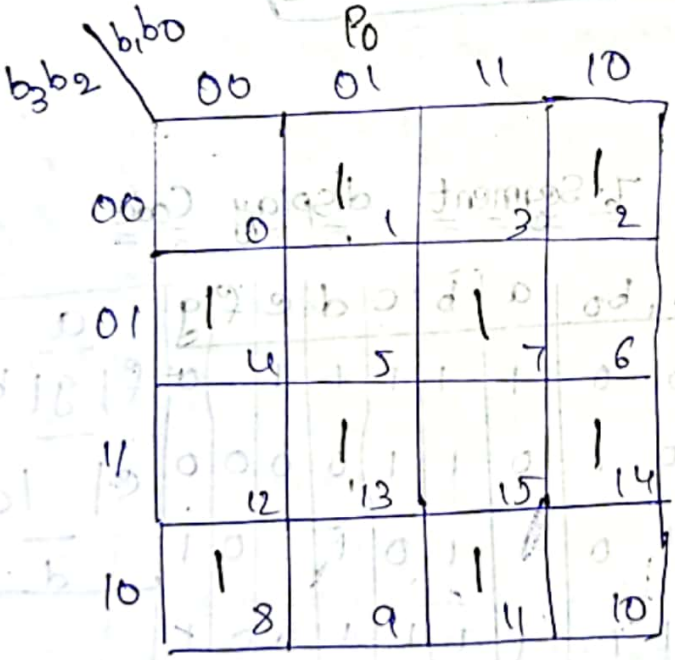
12/2021

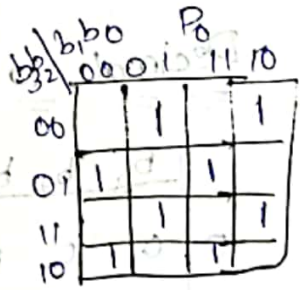


Parity bit generator

	b_3	b_2	b_1	b_0	P_0
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1

	b_3	b_2	b_1	b_0	P_0
(8)	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
(11)	1	0	1	1	1
(12)	1	1	0	0	0
(13)	1	1	0	1	1
(14)	1	1	1	0	1
15	1	1	1	1	0





$$P_0 = \bar{b_3}\bar{b_2}\bar{b_1}b_0 + \bar{b_3}\bar{b_2}b_1\bar{b_0} + \bar{b_3}b_2\bar{b_1}\bar{b_0} + \bar{b_3}b_2b_1b_0$$
$$+ b_3\bar{b_2}\bar{b_1}b_0 + b_3b_2\bar{b_1}\bar{b_0} + b_3\bar{b_2}b_1\bar{b_0} + b_3\bar{b_2}\bar{b_1}\bar{b_0}$$
$$= \bar{b_3}\bar{b_2}(\bar{b_1}b_0 + b_1\bar{b_0}) + \bar{b_3}b_2(\bar{b_1}\bar{b_0} + b_1b_0)$$
$$+ b_3b_2(\bar{b_1}b_0 + b_1\bar{b_0}) + b_3\bar{b_2}(\bar{b_1}\bar{b_0} + b_1b_0)$$

XOR XNOR XOR XNOR

$$P_0 = \bar{b_3}\bar{b_2}(b_0 \oplus b_1) + \bar{b_3}b_2(\overline{b_0 \oplus b_1}) + b_3b_2(b_0 \oplus b_1) + b_3\bar{b_2}(\overline{b_0 \oplus b_1})$$

$$P_0 = b_0 \oplus b_1(\bar{b_3}\bar{b_2} + b_3b_2) + \overline{b_0 \oplus b_1}(\bar{b_3}b_2 + b_3\bar{b_2})$$

XNOR XOR

$$P_0 = (b_0 \oplus b_1)(\overline{b_2 \oplus b_3}) + \overline{b_0 \oplus b_1}(b_3 \oplus b_2)$$

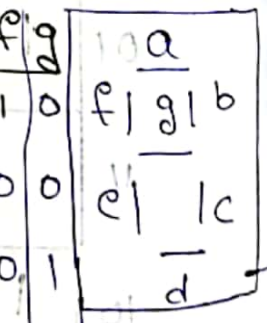
X $\bar{Y}$ $\bar{X}$ Y

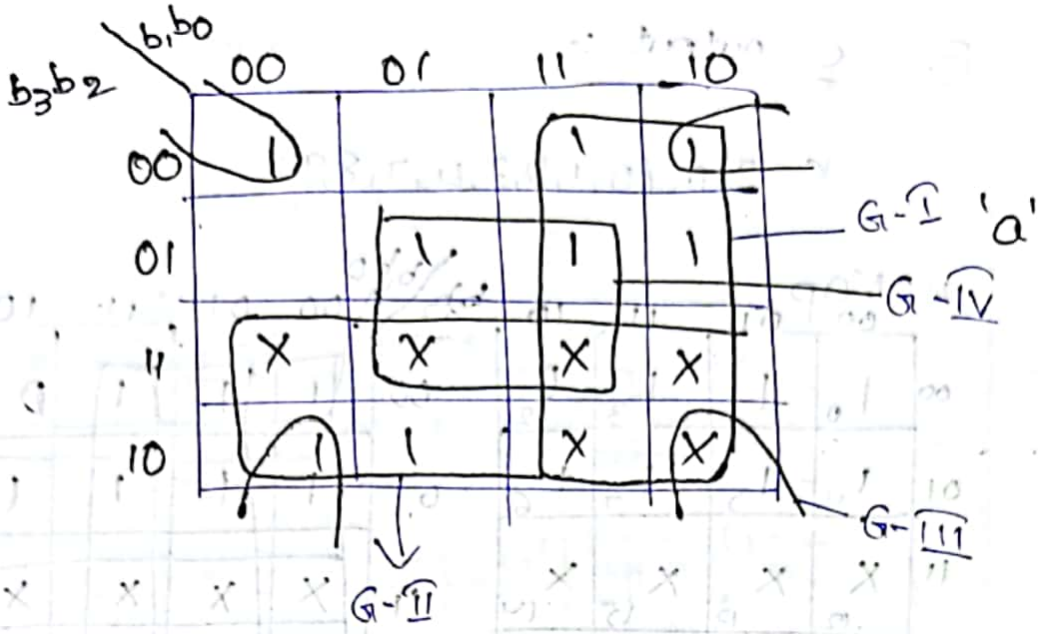
$$P_0 = (b_0 \oplus b_1) \oplus (b_2 \oplus b_3)$$

$$\boxed{P_0 = b_0 \oplus b_1 \oplus b_2 \oplus b_3}$$

Decimal to 7-Segment display Code

-	b_3	b_2	b_1	b_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1





G-I :- b_1

G-II :- b_3

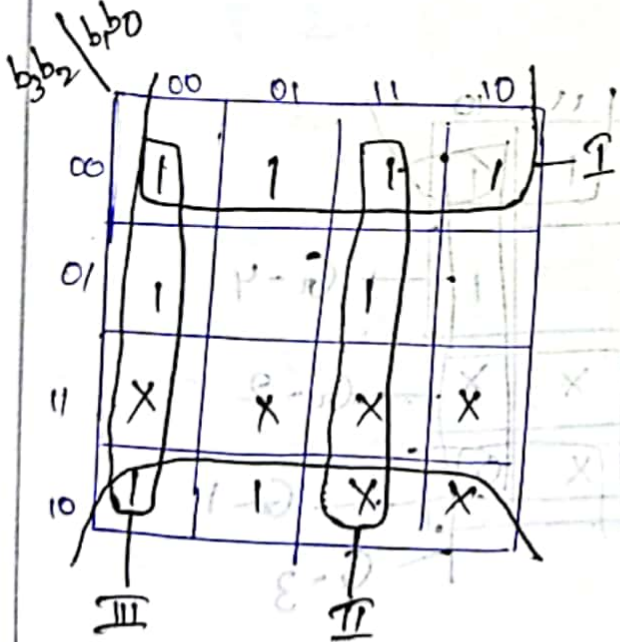
G-III :- $\overline{b_2} \cdot \overline{b_0}$

G-IV :- $b_2 \cdot b_0$

SOP form.

$$a = b_1 + b_3 + \overline{b_2}\,\overline{b_0} + b_2 b_0$$

$$\boxed{a = b_1 + b_3 + b_0 \odot b_2}$$



$I = \overline{b_2}$

$II = b_1 \cdot b_0$

$III = \overline{b_1} \cdot \overline{b_0}$

$$b = \overline{b_2} + b_1 b_0 + \overline{b_1}\,\overline{b_0}$$

$$= \overline{b_2} + b_1 \odot b_0$$

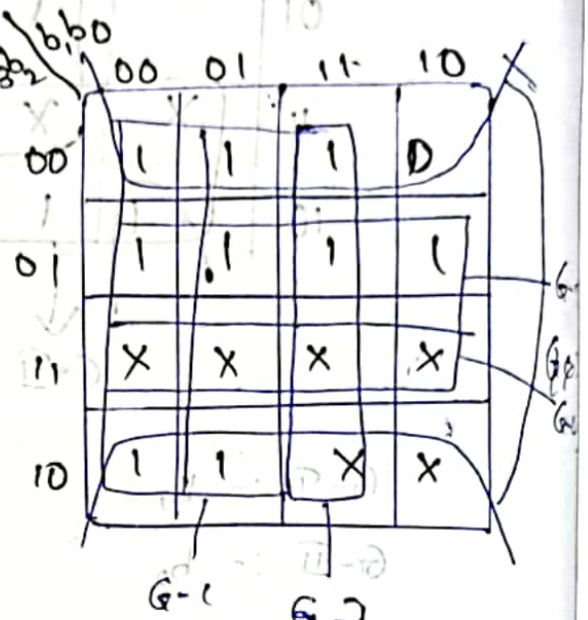
$$\boxed{b = \overline{b_2} + b_1 \odot b_0}$$

For c output :-

$c = \sum m(0,1,2,3,4,7,8,9)$

K-Map

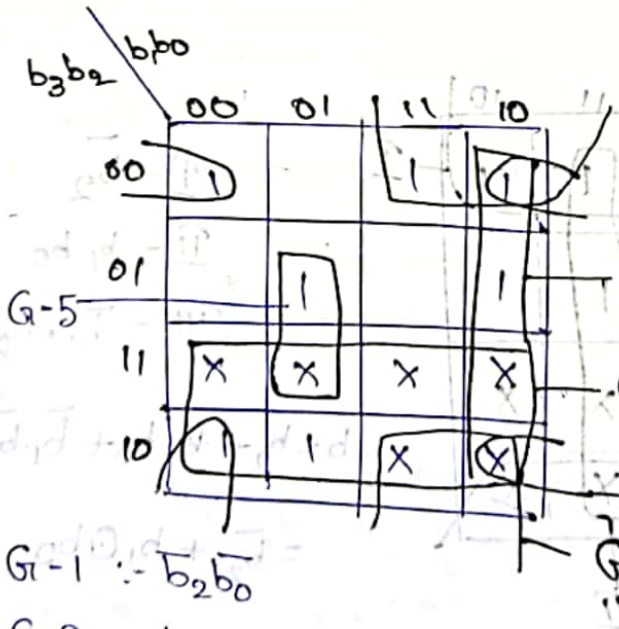
b_3b_2 \ b_1b_0	00	01	11	10
00	1 (0)	1 (1)	1 (3)	0 (2)
01	1 (4)	1 (5)	1 (7)	1 (6)
11	X (12)	X (13)	X (15)	X (14)
10	1 (8)	1 (9)	X (11)	X (10)



$$c = \bar{b}_1 + b_2 + b_0$$

For d output :-

$d = \sum m(0,2,3,5,6,8,9)$



G-1 :- $\bar{b}_2\bar{b}_0$

G-2 : b_3

G-3 : $\bar{b}_2 b_1$

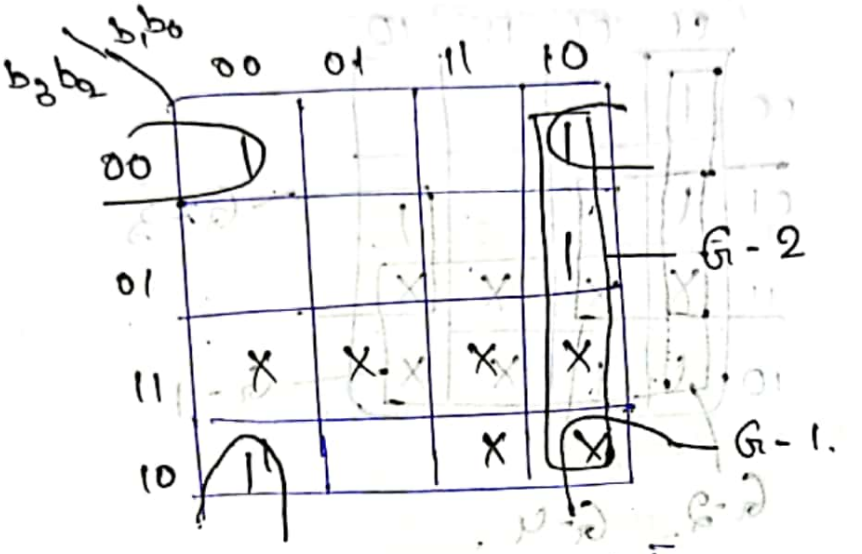
G-4 :- $b_1\bar{b}_0$

G-5 :- $b_2 b_0 \bar{b}_1$

$$d = \bar{b}_2\bar{b}_3 + \bar{b}_3 + \bar{b}_2 b_1 + b_1\bar{b}_0 + b_2 b_0 \bar{b}_1$$

For e Output :-

$e = \Sigma m (0, 2, 6, 8)$

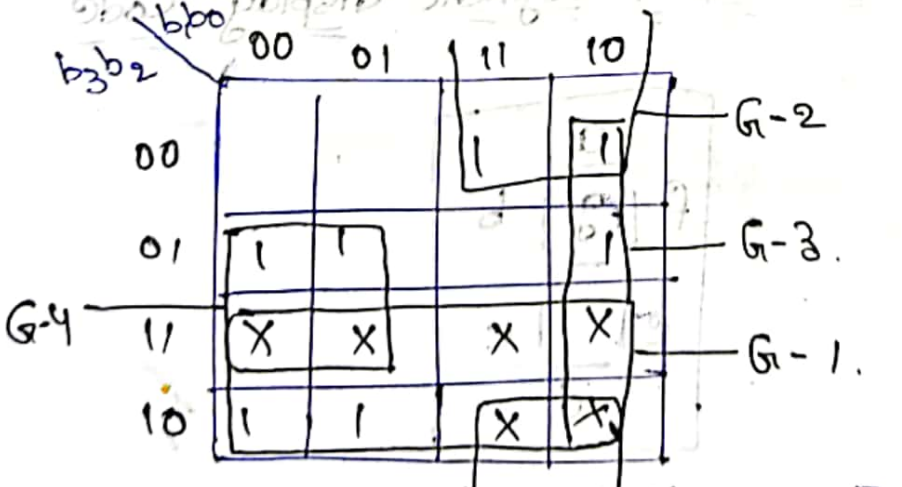


G-1 :- $\bar{b_2}\bar{b_0}$ G-2 : $b_1\bar{b_0}$

$$e = \bar{b_2}\bar{b_0} + b_1\bar{b_0}$$

For output g :-

$g = \Sigma m (0, 2, 3, 5, 6, 8, 9)$



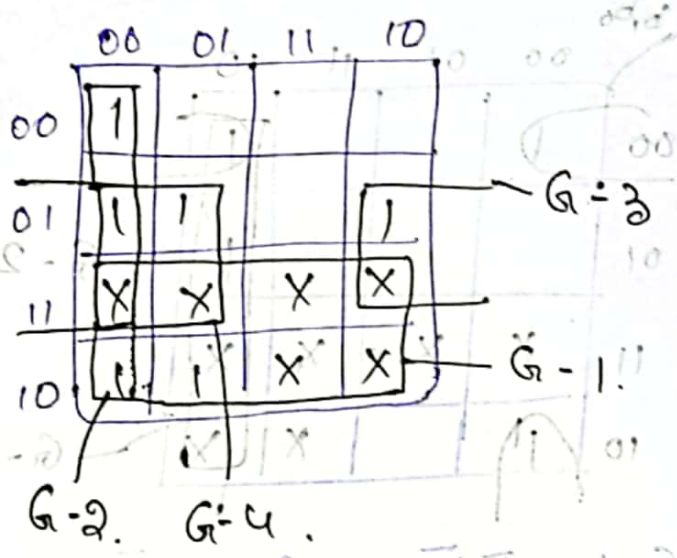
G-1 :- b_3 G-2 :- $\bar{b_2}b_1$ G-3 : $b_1\bar{b_0}$

G-4 :- $b_2\bar{b_1}$

$$g = b_3 + \bar{b_2}b_1 + b_1\bar{b_0} + b_2\bar{b_1}$$

For f output :-

$$f = \Sigma m(0,4,5,6,8,9)$$



G-1 :- b_3

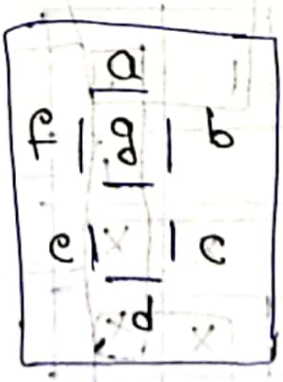
G-2 : $\bar{b}_1\bar{b}_0$

G-3 :- $b_2\bar{b}_0$

G-4 :- $b_2\bar{b}_1$

$$\therefore f = b_3 + \bar{b}_1\bar{b}_0 + b_2\bar{b}_0 + b_2\bar{b}_1$$

Decimal to 7 segment display code

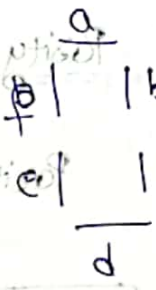


7-segment LED Display

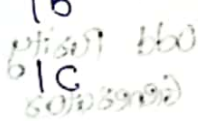
- The logic circuit for the decimal to 7-segment display code is designed with 4 inputs and 7 outputs.
- Truth Table is constructed with the combination of Inputs for each decimal number.
- Truth Table is constructed by listing 7 display i/p signals, decimal number & corresponding 4 digit binary numbers.

Decimal – 7 segment display

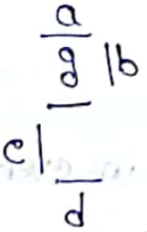
0



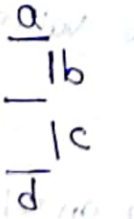
1 –



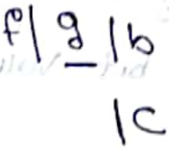
2.



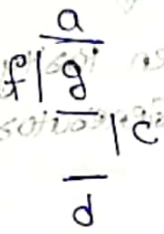
3 –



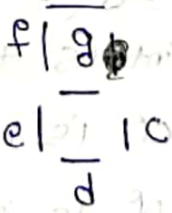
4.



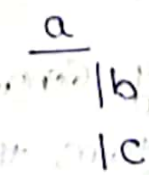
5.



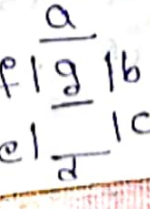
6.



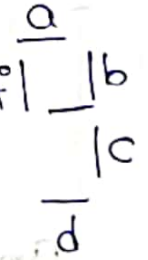
7.



8.



9.

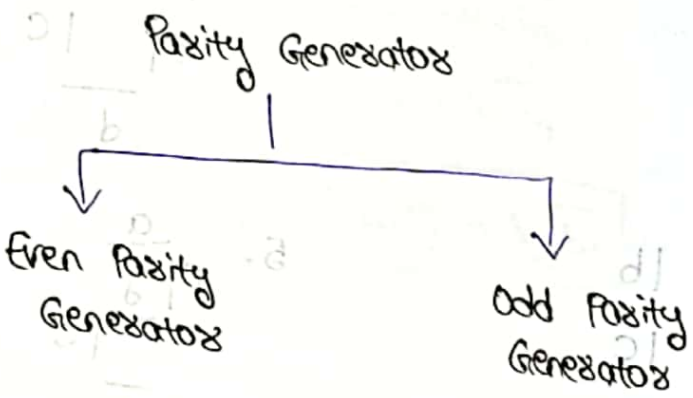


* Parity bit Generator :-

The Parity bit (or) check bit are the bits added to the binary code to check whether the particular code is in even parity or odd parity by number of 1's.

Parity Generator :- The parity generator is the combinational ckt at the transmitter. It takes an original message as i/p and generates the parity bit for that message.

Types of Parity Generator.



Even Parity Generator :-

1. It maintains the binary data in even no. of 1s.
2. If no. of 1's is even, parity bit value is 0.
3. If no of 1s is odd, parity bit value is 1.

Odd Parity Generator :-

1. It maintains the binary data in an odd no. of 1s.
2. If no. of 1's is odd, parity bit value is 0.
3. If no. of 1s is even, parity bit value is 1.

* BCD to Gray Code Converter

The logical circuit which converts the binary code to equivalent gray code is known as "Binary to Gray code converter."

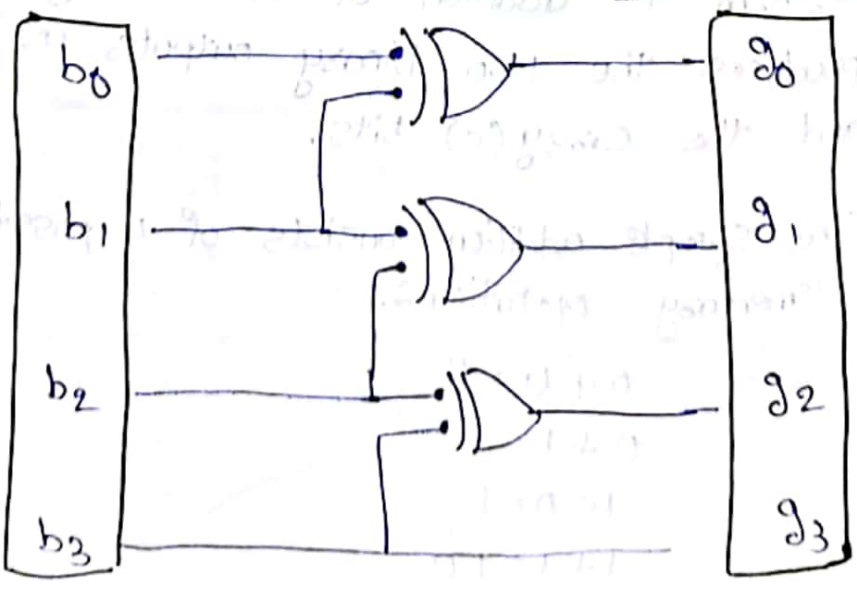
Gray Code :- It is a binary number system in which every successive pair of numbers differs in only one bit.

* A binary number is converted to gray code to reduce the switching operation.

* In this conversion process, 4 bit BCD code is considered as Input which is converted into 4-bit gray code.

* For this bcd to gray code conversion.

1. The MSB is kept the same. $g_3 = b_3$
2. Next, take XOR of 1st & 2nd binary bit $g_2 = b_2 \oplus b_3$
3. Next, take XOR of 2nd & 3rd binary bit $g_1 = b_1 \oplus b_2$
4. Lastly take XOR of 3rd & 4th binary bit $\therefore g_0 = b_0 \oplus b_2$



Arithmetic Circuits :-

Asithmetic circuits axe nothing but the logic circuits, which is used to complete some Asithmetic functions like Addition, Multiplication, Substraction, Division etc.

* Asithmetic circuits axe :
  - Addexs
  - Subtxactoxs
  - Multipliexs, etc....

Adders :- An addex is a device that will add togethex two bits and xesult as the output.

Thexe axe two kinds of addexs
1. Half Addex.
2. Full Addex.

1. Half Addex

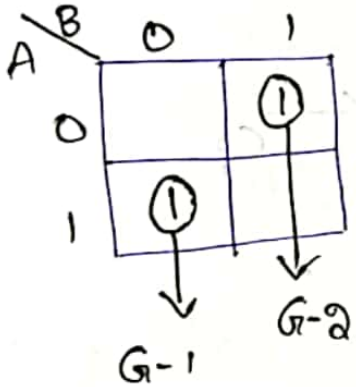
- Half Addex is a combinational cixcuit used to pexfoxm the addition of two binaxy inputs & pxoduces the two binaxy outputs as sum(S) and the caxxy(c) bits.
- The sample addition consists of 4 possible elementaxy opexations.

$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=10$$

Truth Table for Half Adder :-

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-Map for Carry (C)



Boolean expression for Sum

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

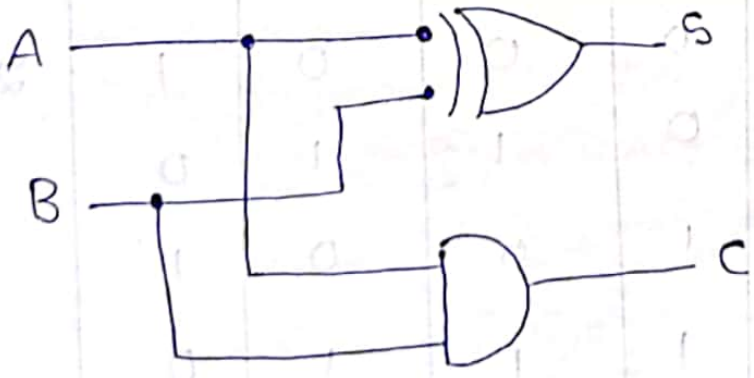
(or)

$S = A \oplus B$

G-1 :- $A\bar{B}$

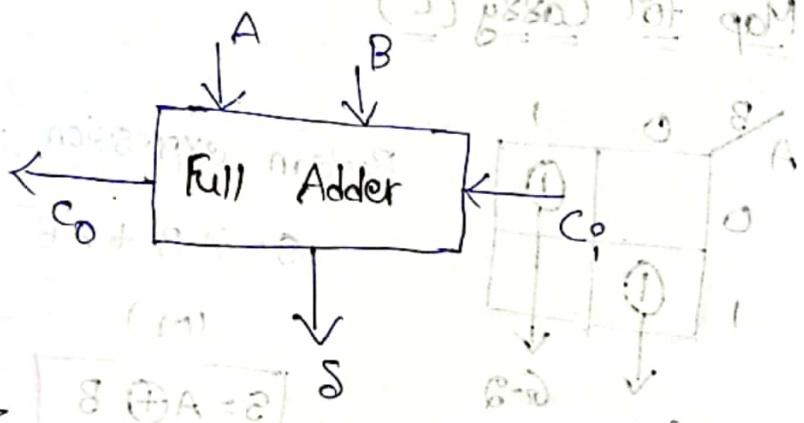
G-2 :- $\bar{A}B$

Logic circuit for Half Adder :-



2. Full Adder :-

- Full Adder is a combinational circuit used to perform the addition of 3 no. of Inputs and produces two outputs.
- The Full Adder adds the bits A & B and the carry from the previous column called the "carry-in (C_{input})" and gives output as the "Sum bit (S)" and the carry bit called the "carryout (C_{out})".



Truth Table for Full Adder :-

	A	B	C_i	C_o	S
0	0	0	0	0	0
1	0	0	1	0	1
2	1	0	0	0	1
3	1	0	1	1	0
4	0	1	0	0	1
5	0	1	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

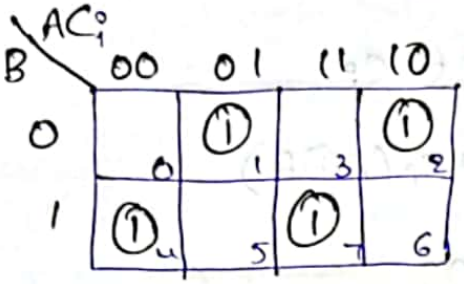
Ex:

$0 \rightarrow \begin{matrix} C & S \\ 0 & 0 \end{matrix}$

$2 \rightarrow \begin{matrix} C & S \\ 1 & 0 \end{matrix}$

K-Map for Sum :-

$$S = \sum m\,(1,2,4,7)$$



Boolean expression for Sum

$$S = \bar{B}\bar{A}C_i + \bar{B}A\bar{C}_i + B\bar{A}\bar{C}_i + BAC_i$$

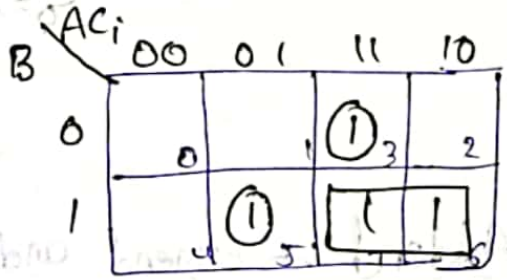
$$= \bar{B}(\bar{A}C_i + A\bar{C}_i) + B(\bar{A}\bar{C}_i + AC_i)$$

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

$$\boxed{S = (A \oplus B) \oplus C_i}$$

K-Map for Carryout (C_0)

$$C_0 = \sum m\,(3,5,6,7)$$



Boolean expression for Carryout

$$C_0 = BA + B\bar{A}C_i + \bar{B}AC_i$$

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

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

$$\boxed{C_0 = AB + C_i(A \oplus B)}$$

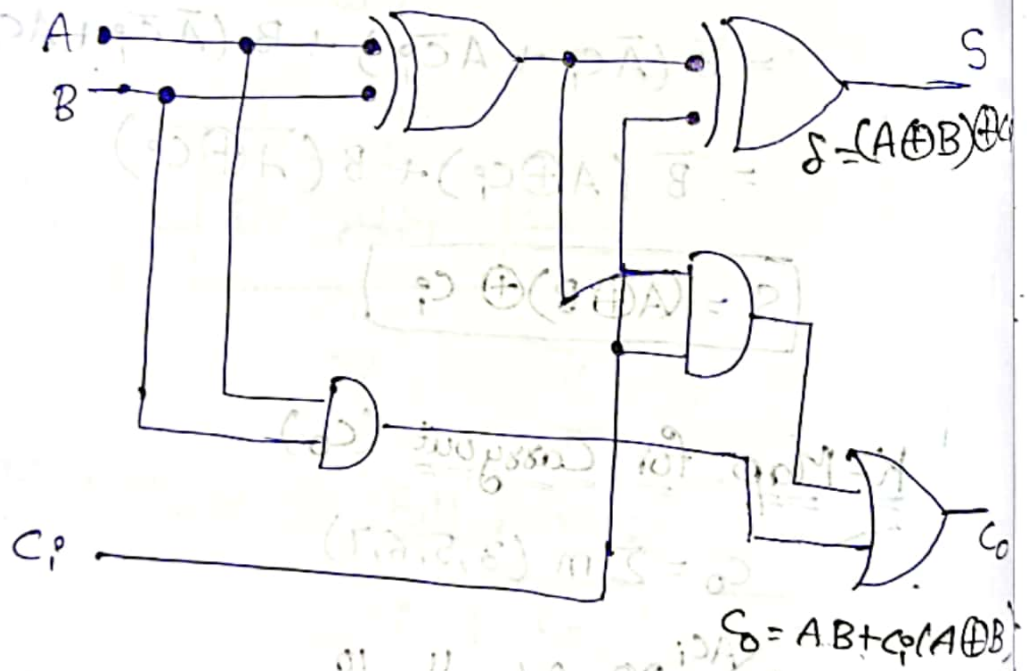
Logical Circuit for the Full Ladder :-

Logical circuit for the full ladder from the obtained boolean expression is.

$$S = (A \oplus B) \oplus C_i$$

$$C_o = AB + C_i(A \oplus B)$$

Here, 2-XOR Gates,
2-AND Gates,
1-OR Gate required.



Augend and Addend :- Augend and addend are added or summed up to find a sum or total.

3. Binary Adder (Ripple Carry Type adder)

- Multiple bits binary adder is obtained by using no. of full adders connected in cascade.
- For the 1st full adder. LSB of Augent & Addend along with carry in (C_{in}) are inputs.
- Its sum and carry out are outputs. The carry out is used as carry in for next full adder. Other i/ps for the next full adder are the next significant bits.
- It again generates the sum & carry.
- The process is repeated till the last MSB.

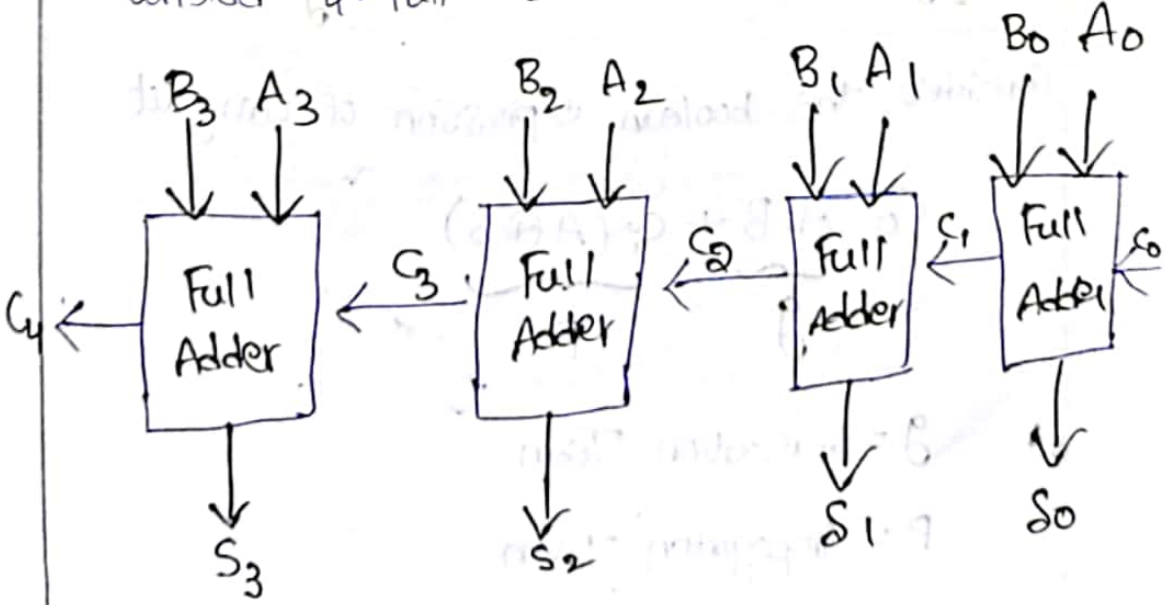
- **4-bit Binary Adder :-**

Consider A & B contain 4 bits,

$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$

Consider 4-full adders



Carry - Lookahead Adder (CLA)

- A carry-lookhead Adder (CLA) or fast adder improves speed by reducing the amount of time required to determine carry bits.
- The CLA adder calculates one or more carry bits before the sum. It reduces the wait time to calculate the result of larger-value bits of the adder.

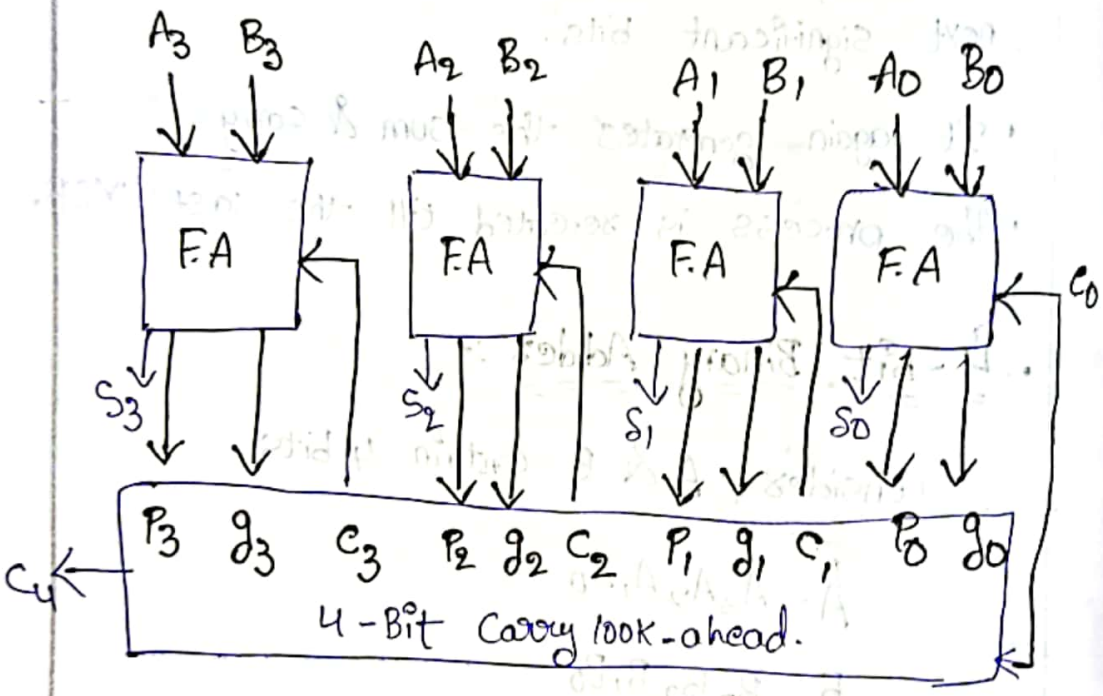


Fig:- 4-bit Adder with Carry Lookahead.

Consider the boolean expression of carry out

$$C_0 = \underbrace{A \cdot B}_{g} + \underbrace{C_i (A \oplus B)}_{P}$$

g = Generation Term

P = Propogation Term

The General expression for Carry.

$$C_i = g_i + P_i C_{i-1}$$

$$C_2 = g_2 + P_2 C_{2-1} \Rightarrow C_2 = g_2 + P_2 C_1$$

$$\therefore \boxed{C_0 = g_0 + P_0 C_{-1}} \longrightarrow (1)$$

$$C_1 = g_1 + P_1 C_0$$

$$= g_1 + P_1 (g_0 + P_0 C_{-1}) \quad [\text{from eq (1)}]$$

$$\boxed{C_1 = g_1 + P_1 g_0 + P_1 P_0 C_{-1}} \longrightarrow (2)$$

$$C_2 = g_2 + P_2 C_1 \quad [\text{from eq(2)}]$$

$$= g_2 + P_2 (g_1 + P_1 g_0 + P_1 P_0 C_{-1})$$

$$\boxed{C_2 = g_2 + P_2 g_1 + P_1 P_2 g_0 + P_1 P_2 P_0 C_{-1}} \longrightarrow (3)$$

$$C_3 = g_3 + P_3 C_2 \quad [\text{from eq(3)}]$$

$$\boxed{C_3 = g_3 + P_3 g_2 + P_3 P_2 g_1 + P_3 P_1 P_2 g_0 + P_3 P_1 P_2 P_0 C_{-1}}$$

$$\longrightarrow (4)$$

Here

$$g_3 = A_3 \cdot B_3$$

$$P_2 = A_2 \oplus B_2$$

Complements :-

Complements are used in digital circuits, because it is faster to subtract by adding complements than by performing true subtraction.

- Complements are two types.

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

* r's Complements

Ex:1 : Complement of $(7)_{10}$

Formula : $r^n - N$

r = radix / base

n = Total no. of digits

N = Given Number

$(10)^1 - 7 = 3$

∴ Complement of $(7)_{10} = 3$.

2. Complement of $(5690)_{10}$

$r = 10$, $N = 5690$, $n = 4$

$r^n - N = (10)^4 - 5690$

$= 10000 - 5690$

$= 4310$

∴ Complement of $(5690)_{10} = 4310$

3. Complement of $(1101)_2$

$r = 2$

$N = 1101$

$n = 4$

$\therefore r^n - N = (2)^4 - 1101$

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

$= 10000 - 01101$

↓ complement

$10010 + 1$

10011

$\Rightarrow 10000$

10011

100011

$\therefore$ Complement of $(1101)_2 = 00011$

*$(r-1)$'s Complement

Here,

If

$r = 10$

$r = 2$

$r = 8$

$r = 16$

$(r-1)$'s complement

9's

is

7's

F's complement

Formula

$(r-1)^{'s}$ Complement $= r^n - N - 1$

* $(r-1)$'s Comp = r's Comp - 1

* $(r-1)$'s Comp + 1 = r's Comp

Ex:- Find 7's complement of octal no $(5674)_8$

Sol $(r-1)$'s Complement = $r^n - N - 1$

$r = 8$

$n = 4$

$N = 5674$

$\therefore r^n - N - 1 = 8^4 - 5674 - 1$

$= 7777 - 5674$

$= 2103$

$\therefore (r-1)$'s complement of $(5674)_8 = 2103$.

*10's Complement

1. 5896

$r^n - N = 10^4 - 5896$

$= 4104$

Subtracting Using 10's Complement

1. Find 9-4

$$\begin{array}{r} 10 \\ -4 \\ \hline 6 \end{array} \Rightarrow \begin{array}{r} 9 \\ +6 \\ \hline \cancel{1}5 \end{array}$$

Carry = 1 (1 ⇒ +ve)

9-4 = 5

$9-4=5 \qquad 10^n - N$

$10+9+(-4) = 5+10$

$9+(10-4) = 15$

$9+6 = 15$

$15 = 15.$

2. Find 88-32.

$$\begin{array}{r} 100 \\ 32 \\ \hline 68 \end{array} \Rightarrow \begin{array}{r} 88 \\ 68 \\ \hline 156 \end{array}$$

$100+88-32 = 56+100$

$88+(100-32) = 156.$

$156 = 156.$

$\therefore 88-32 = 56$

3. Find 72532 - 3250

Step-1:- $100000 - 03250 = 96750$

Step-2:- $72532 + 96750 = 169282$

$72532 - 3250 = 69282$

* 9's Complement

1. $(2457)_{10}$

$r^n - N - 1 = ((10)^4 - 1) - 2457$

$= 9999 - 2457$

$= 7542.$

2. $(89031)_{10}$

$r^n - N - 1 = 10^5 - 1 - 89031$

$= 99999 - 89031$

$= 10968.$

3. $(2974)_{10}$

$r^n - N - 1 = 10^4 - 1 - 2974$

$= 9999 - 2974$

$= 7025$

Subtraction using 9's Complement

1. Find 1234 − 1000

Minuend
↑
A − B − Subtrahend.

Step-1:
```
  9999
- 1000
------
  8999
```

1. 9's Complement of B.
2. Add it to A.
3. Add carry to Above.

Step-2:
```
  1234
+ 8999
-------
 10233
```

Step-3:- 0233 + 1 = 234

∴ 1234 − 1000 = 234

81.2 from 98.3.

Sol 1.
```
  999
 81.2
------
 18.7
```
— 9's

2.

2. Find 1234 - 2000

Step-1 :
```
  9999
- 2000
------
  7999
```

Step-2 :
```
  1234
+ 7999
------
  9233
```
No carry

Step-3 :
```
  9999
- 9233
------
  0766
```
9's complement

1234 - 2000 = -766

*1's Complement

① 1's complement of binary number 01001011

Given 0 1 0 0 1 0 1 1 → 1's Complement → 1 0 1 1 0 1 0 0

01001011 is → 10110100

Subtraction using 1's Complement

1. $(28)_{10} - (19)_{10}$

Step-1:- $(28)_{10} = (011100)_2$

$(19)_{10} = (010011)_2$

Step-2:- $(19)_{10} = 010011$

↓ 1's

101100

Step-3:- Add $(28)_{10}$ & 1's of 19

$$\begin{array}{r} 011100 \\ +\ 101100 \\ \hline (1)\,001000 \end{array}$$

Carry-1 neglect & add.

Step-4:-

$$\begin{array}{r} 001000 \\ +\ \quad\quad 1 \\ \hline 001001 \end{array} \rightarrow (9)_{10}$$

$\therefore (28)_{10} - (19)_{10} = (9)_{10}$

1. 1's complement to subtrahend
2. Add to minuend
3. Carry generated and add to result
4. m>n carry is generated
5. m<n no. carry then again do complement to result.

2. $(15)_{10} - (28)_{10}$

Step-1:- $(15)_{10} = (01111)_2$

$(28)_{10} = (11100)_2$

Step-2:- $(28)_{10} =$ 11100

↓ is comp.

00011

Step-3:- Adding

$$\begin{array}{r} 01111 \\ 00011 \\ \hline 10010 \end{array} = -13$$

Step-4:- Taking is complement of 10010

10010

↓ 1's

$\boxed{01101} = (13)_{10}$

3. $(0111)_2 - (0011)_2$ (or) $(7-3)$

Both the numbers have equal length

Step-1:- $(0111)_2 = (0011)_2$.

1's Complement of $(0011)_2 = (1100)_2$

Step-2:-

$$\begin{array}{r} 0111 \\ +1100 \\ \hline (1)0011 \end{array}$$

Step-3:-

$$\begin{array}{r} 0011 \\ +\quad 1 \\ \hline 0100 \end{array} \Rightarrow (4)_{10}$$

$\boxed{7-3=4}$

4. 3-7.

Step-1:- $(3)_{10} = (0011)_2$

$(7)_{10} = (0111)_2$

Step-2:- Take 1's complement of $(7)_{10}$

0 1 1 1

↓ is

1 0 0 0

Step-3:-

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

Carry is not generated is (-ve)

Step-4:- Taking 1's Comp..

1 0 1 1

↓

0 1 0 0

$(0100)_2 = (4)_{10}$

*2's Complement

Find 2's complement of 00000101

0 0 0 0 0 1 0 1

↓ 1's

1 1 1 1 1 0 1 0

\+ 1

1 1 1 1 1 0 1 1

Subtraction using 2's Complement

1. 12 - 18

Step-1:- $(12)_{10} = (01100)_2$

$(18)_{10} = (10010)_2$.

Step-2:- Taking 2's complement of $(18)_{10}$

```
  10010
    ↓ is
  01101
+     1
-------
  01110
```

Step-3:- Adding

```
  01100
+ 01110
-------
  11010
```

No carry is generated so the result is (-ve)

Step-4:- Taking 2's complement of $(11010)_2$

```
  11010
    ↓ is comp
  00101
+     1
-------
  00110   = (6)10
```

1. 2nd Complement to Subtrahend
2. Add to minued.
3. Carry generate neglect it) is not generated again 2's complement

2. 7 - 2.

Step-1:- $(7)_{10} = (111)_2$

$(2)_{10} = (010)_2$

Step-2:- Taking 2's comple of $(2)_{10}$.

$$\begin{array}{r} 010 \\ \downarrow \text{is} \\ 101 \\ +\quad 1 \\ \hline 110 \end{array}$$

Step-3:- Adding

$$\begin{array}{r} 111 \\ 110 \\ \hline (1)101 \end{array} \quad +ve$$

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

$\boxed{7-2=5}$

1's Complement

'1' by '0' & '0' by '1'

3 = 0 0 11

-3 = 1 1 0 0

Two's Complement

Step-1:- 0 0 1 1

↓ i's Complement

1 1 0 0

Step-2:- 1 1 0 0
+ 1
1 1 0 1

9's Complement

1234 $r^n - N$ $n=4$

$N=1234$.

9's Complement: 9999
- 1234
8765

10's Complement

8634

9's complement 9999
- 8634
1365

10's Complement = 1365 + 1

= 1366

Array Multiplier :-

An Array multiplier is a digital combinational circuit used for multiplying two binary numbers by employing an array of full adders and half adders.

Multiplication Schemes,

Two Types of Multiplication Schemes.

Serial Multiplication (shift - Add) :- The serial multiplication operation can be solved by finding partial products & then adding partial product together. The implementations are primitive with simple architecture.

Parallel Multiplication : The parallel product are generated simultaneously in parallel multiplication and a high-performance machine Parallel implementation are applied, latency is minimized.

* Multiplication process has three main steps :
  - Partial product generation
  - Partial product reduction
  - Final addition

* **Multiplicand** : a quantity which is to be multiplied by another (the multiplier)

* **Multiplier** : a quantity by which the multiplicand is to be multiplied.

4X4 Multiplication.

Multiplicand :- $A = a_3\ a_2\ a_1\ a_0$

Multiplier :- $B = b_3\ b_2\ b_1\ b_0$

$a_3\ a_2\ a_1\ a_0$

$b_3\ b_2\ b_1\ b_0$

$a_3b_0\ \ a_2b_0\ \ a_1b_0\ \ a_0b_0$

$a_3b_1\ \ a_2b_1\ \ a_1b_1\ \ a_0b_1$

$a_3b_2\ \ a_2b_2\ \ a_1b_2\ \ a_0b_2$

$a_3b_3\ \ a_2b_3\ \ a_1b_3\ \ a_0b_3$

} Partial Product

$P_6\ \ P_5\ \ P_4\ \ P_3\ \ P_2\ \ P_1\ \ P_0$ — Product

Ex:-

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

m = no. of bits in multiplicand

= no. of bits in multiplier

No. of AND gates = m x n

no. of Half Adders = n

no. of Adders = (m-1)n

no. of Full. Adders = (m-1)- n = (m-2) n

Steps in Multiplication

The following are the steps for any multiplication

- If LSB of multiplier is 1, then add the multiplicand into an accumulator multiplier bit is shifted one bit to the right and multiplicand bit is shifted one bit to the left.
- Stop when all the bits of the multiplier is zero.
- Less hardware is used if partial products are added serially. We can add all PP by a parallel multiplier. However, it is possible to use compressor technique the number of partial products can be reduced before addition, is performed.

Construction & Working of a 4X4 Array Multiplier :-

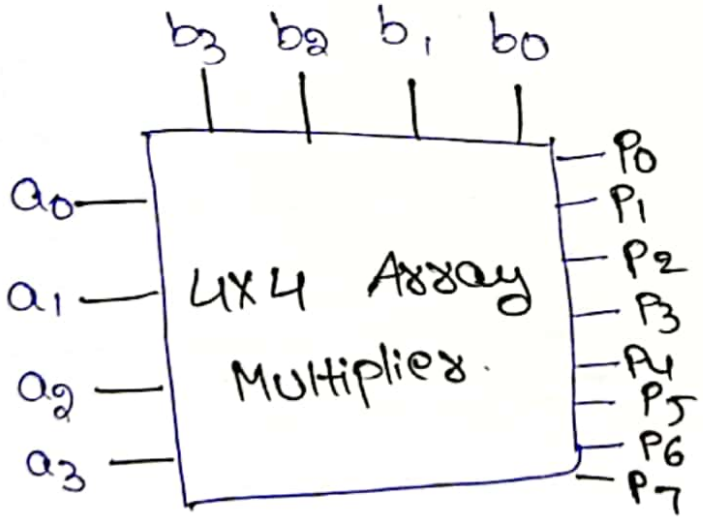
The design structure of the array multiplier is regular, it is based on the add shift algorithm principle

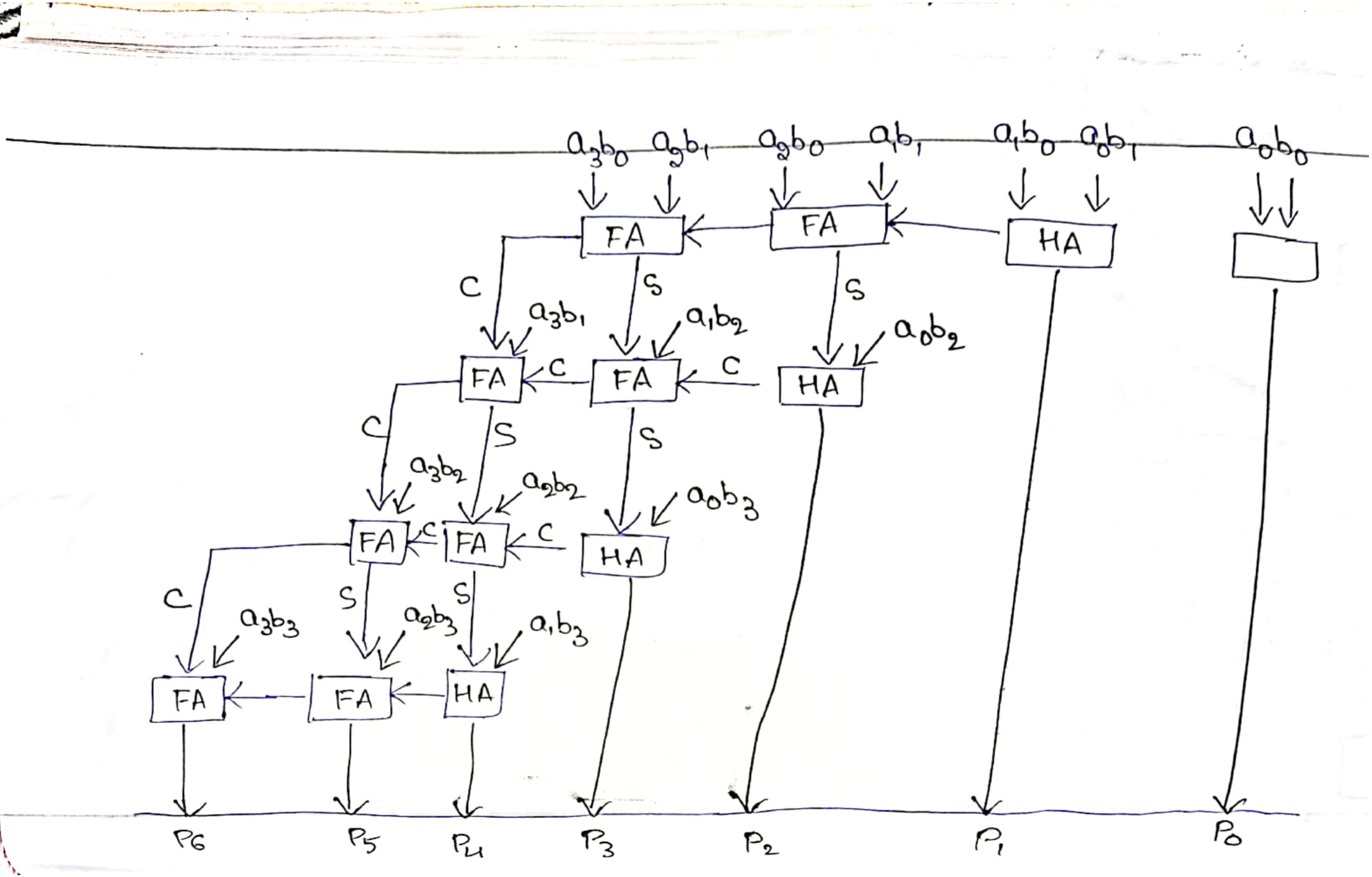
Partial Product = Multiplicand * Multiplier

AND gates are used for the product, the summation is done using full Adders and Half Adders where the partial product is shifted according to their bit orders.

In an $n*n$ array multiplier, $n*n$ AND gates compute the partial products and the addition of partial products can be performed by using $n*(n-2)$ Full adders and n Half adders.

The 4X4 array multiplier shown has 8 inputs & 8 outputs.





BCD Addition

1. $9+9=18$

$(9)_{10}=(1001)_2$

$(9)_{10}=(1001)_2$

```
  1001 H
  1001 H
 -------
  10010
 -------

  1    0010
  1      2
```

```
  1 2 H   [Need to add 6]
    6
 -------
   18
```

2. $5+6=11$

```
  0101 H
  0110 H
 -------
  1011
 -------

   B H
    6
 -------
   17
 -------
```

3. $7+5=12$

```
  0111
  0101
 -------
  1100
 -------

  12
   6
 -------
  18
 -------
```

4. $3+2=5$

```
  0011
  0010
 -------
  0101
 -------
   5H.
```

Decimal 18 equivalent
Hexa decimal is 12H

```
16 | 18
   | 1-2
```

Note :-

In this BCD addition, if the addition result is more than 9, need to add '6'.

otherwise we need to add 0.

BCD Addition :-

Step-1 :- At First the given numbers are to be added using the rule of binary

Ex :- Case-1 :-

$$\begin{array}{r} 1010 \\ +0101 \\ \hline 1111 \end{array}$$

Case-2 :-

$$\begin{array}{r} 0001 \\ +0101 \\ \hline 0110 \end{array}$$

Step-2 :- In case-1, the result of addition of two binary numbers is greater than 9, which is not valid for BCD numbers.

- In case-2, the result of addition is less than 9 which is valid for BCD numbers.

Step-3 :- If the 4 bit result of addition is greater than 9 and if a carry bit is present in the result then it is invalid and we have to add 6 (0110) to the result of addition.

Step-4 :- Then the resultant that we would get will be a valid binary coded number.

Ex:-

```
  1 0  1 0
+ 0 1  0 1
---------
  1 1  1 1   → Invalid BCD number
+ 0 1  1 0   → Add 6
---------
0 0 0 1 0 1 0 1   → Valid BCD number.
```

Ex:- Let, 0101 is added with 0110

```
  0 1 0 1
  0 1 1 0
---------
  1 0 1 1   → Invalid BCD number
  0 1 1 0   → Add 6
---------
0 0 0 1 0 0 0 1   → Valid BCD number.
```

Check:-

$(0101)_2 = (5)_{10}$

$(0110)_2 = (6)_{10}$

$(5)_{10} + (6)_{10} = (11)_{10}$

Ex:- Now, let 00010001 is added to 00100110

```
  0001 0001
  0010 0110
-----------
  0011 0111   → Valid BCD number
```

Check:-

$(0001\ 0001)_{BCD} \rightarrow (11)_{10}$

$(00100110)_{BCD} \rightarrow (26)_{10}$

$(00110111)_{BCD} \rightarrow (37)_{10}$

So no need to add 6 as because both

$(0011)_2 = (3)_{10}$

$(0111)_2 = (7)_{10}$

are less than $(9)_{10}$.

This is the process of BCD Addition.

BCD Definition: Binary coded decimal is the storage of number in which each decimal digit is converted into a binary number and stored in a single 8-bit byte.

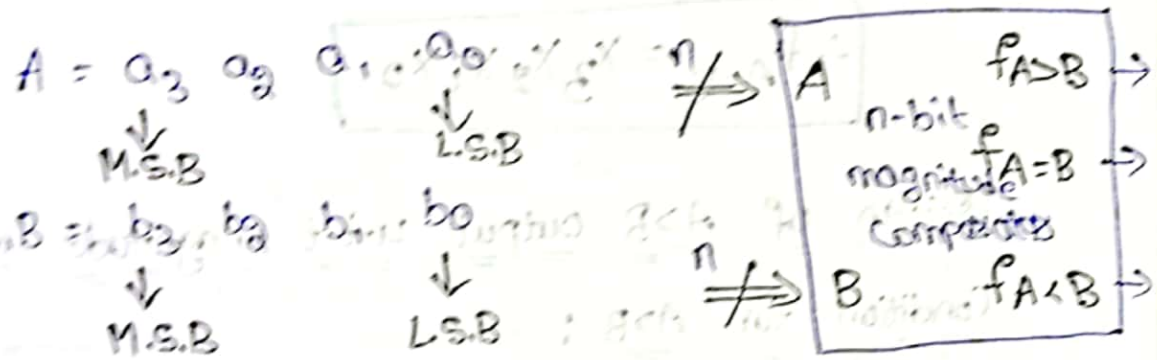
Decimal	Binary number	BCD [Binary Coded Decimal]
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	00010000
11	1011	00010001
12	1100	00010010
13	1101	00010011
14	1110	00010100
15	1111	00010101

Magnitude Comparator

A magnitude comparator is a combinational circuit that compares two numbers 'A & B' to determine whether:

- A > B (or)
- A = B (or)
- A < B

Let us consider two numbers A & B



Inputs :-

First n-bit number A

Second n-bit number B

Outputs :-

3 output signals $f_{A=B}$

$f_{A>B}$

$f_{A<B}$

Design of output (A=B) in 4-bit Magnitude Comparator.

Condition for A=B :

(A=B) Iff

$A_3 = B_3 \longrightarrow (x_3 = 1)$ and

$A_2 = B_2 \longrightarrow (x_2 = 1)$ and

$A_1 = B_1 \longrightarrow (x_1 = 1)$ and

$A_0 = B_0 \longrightarrow (x_0 = 1)$

$\therefore f_{A=B} = x_3 x_2 x_1 x_0$

Design of A>B output 4-bit magnitude comparator

Condition for A>B :

If $a_3 > b_3 \longrightarrow a_3 = 1$ and $b_3 = 0 \Rightarrow a_3 \bar{b_3}$ (or)

If $(a_3 = b_3)$ and $(a_2 > b_2) \Rightarrow x_3 a_2 \bar{b_2}$ (or)

If $(a_3 = b_3)$ and $(a_2 = b_2)$ and $(a_1 > b_1) \Rightarrow x_3 x_2 a_1 \bar{b_1}$

If $(a_3 = b_3)$ and $(a_2 = b_2)$ & $(a_1 = b_1)$ & $(a_0 > b_0) \Rightarrow x_3 x_2 x_1 a_0 \bar{b_0}$

To summarize, (A>B) Iff :

1. $a_3 \bar{b_3} = 1$ or
2. $x_3 a_2 \bar{b_1} = 1$ or
3. $x_3 x_2 a_1 \bar{b_1} = 1$ or
4. $x_3 x_2 x_1 a_0 \bar{b_0} = 1$.

$\therefore f_{A>B} = a_3 \bar{b_3} + x_3 a_2 \bar{b_2} + x_3 x_2 a_1 \bar{b_1} + x_3 x_2 x_1 a_0 \bar{b_0}$

Design of output (A<B) 4 bit magnitude comparator

Condition for A<B :

If $a_3 < b_3 \rightarrow a_3 = 0$ and $b_3 = 1 \Rightarrow \bar{a_3} \cdot b_3$ (or)

If $(a_3 = b_3)$ and $(a_2 < b_2) \Rightarrow x_3 \bar{a_2} b_2$ (or)

If $(a_3 = b_3)$ & $(a_2 = b_2)$ & $(a_1 < b_1) \Rightarrow x_3 x_2 \bar{a_1} b_1$ (or)

If $(a_3 = b_3)$ & $(a_2 = b_2)$ & $(a_1 = b_1)$ & $(a_0 < b_0) \Rightarrow x_3 x_2 x_1 \bar{a_0} b_0$

To summarize (A<B) Iff:-

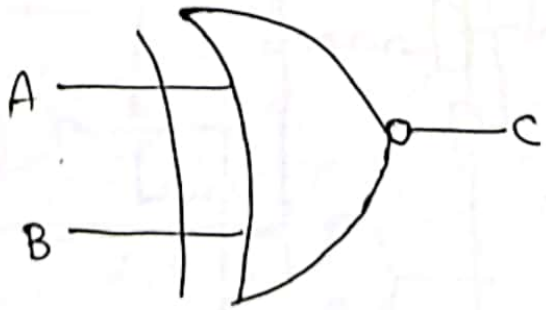
1. $\bar{a_3} \cdot b_3 = 1$ or
2. $x_3 \bar{a_2} b_2 = 1$ or
3. $x_3 x_2 \bar{a_1} b_1 = 1$ or
4. $x_3 x_2 x_1 \bar{a_0} b_0 = 1$

$$\therefore f_{A<B} = \bar{a_3} \cdot b_3 + x_3 \bar{a_2} \cdot b_2 + x_3 x_2 \bar{a_1} \cdot b_1 + x_3 x_2 x_1 \bar{a_0} \cdot b_0$$

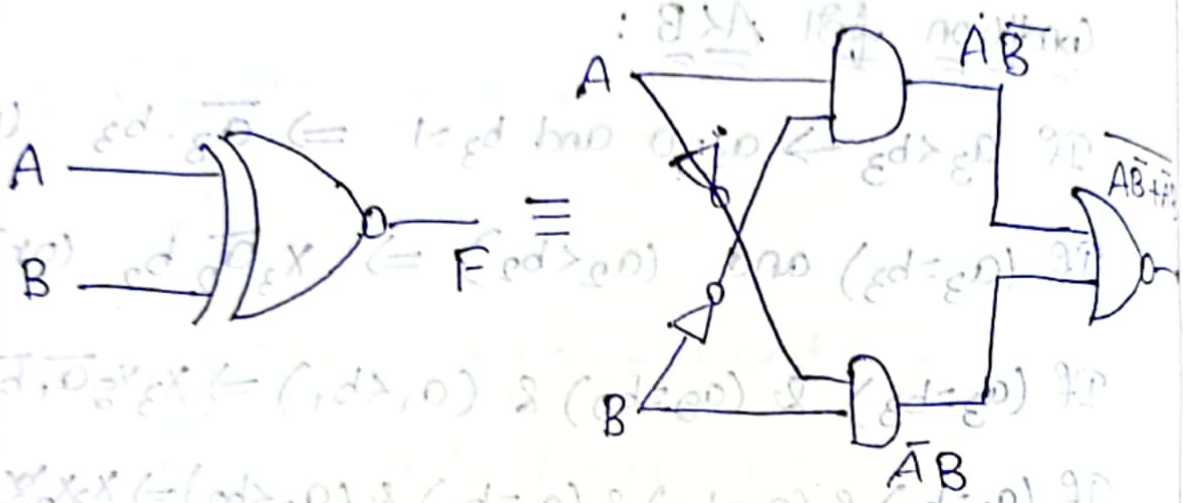
- A exclusive NOR gate is easiest way to compare the equality of the bits.

X-NOR

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



EX-NOR $= \overline{\text{EX-OR}} = \overline{\bar{A}B + A\bar{B}}$



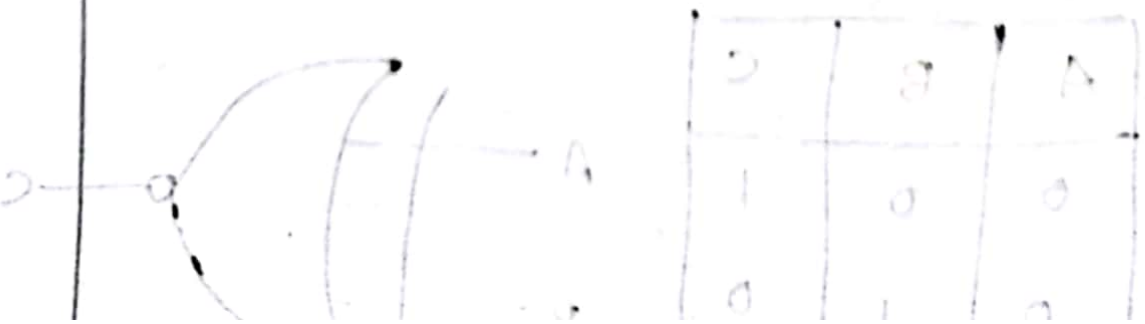
Instead of XNOR circuit we are going to use this circuit to check the given binary numbers are equal or not.

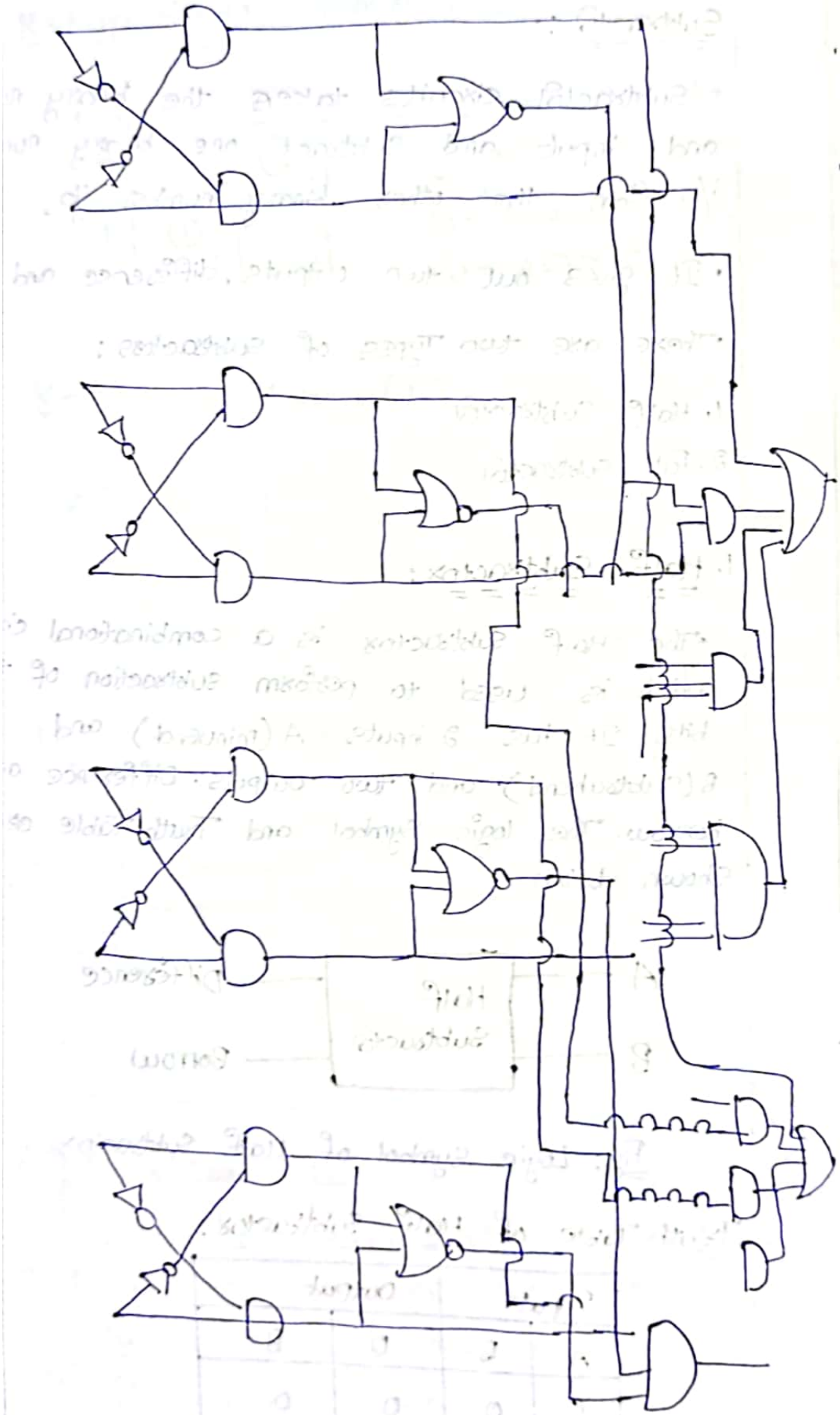
The gate implementation of the three outputs $(A<B)$, $A=B$, $A>B$, using the obtained boolean expressions.

$$f_{A=B} = x_3 x_2 x_1 x_0$$

$$f_{A>B} = a_3\bar{b_3} + x_3 a_2 \bar{b_2} + x_3 x_2 a_1 \bar{b_1} + x_3 x_2 x_1 a_0 \bar{b_0}$$

$$f_{A<B} = \bar{a_3} b_3 + x_3 \bar{a_2} b_2 + x_3 x_2 \bar{a_1} b_1 + x_3 x_2 x_1 \bar{a_0} b_0$$





Logic circuit for magnitude comparator.

Subtractor :-

Subtractor circuits takes the binary numbers and input and subtract one binary number i/p from the other binary numbers i/p.

- It gives out two outputs, difference and borrow.
- There are two Types of subtractors :

1. Half Subtractor
2. Full Subtractor

1. Half Subtractor :

- The Half Subtractor is a combinational circuit which is used to perform subtraction of two bits. It has 2 inputs A (minuend) and B (subtrahend) and two outputs. Difference and borrow. The logic symbol and Truth Table are shown below.

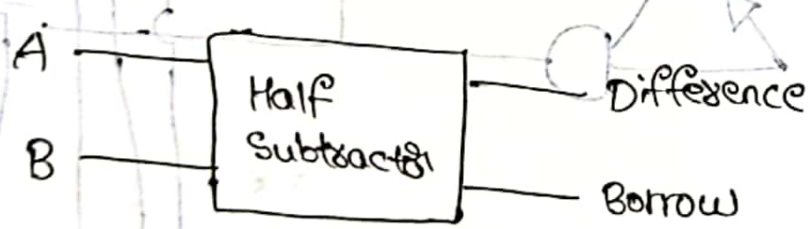
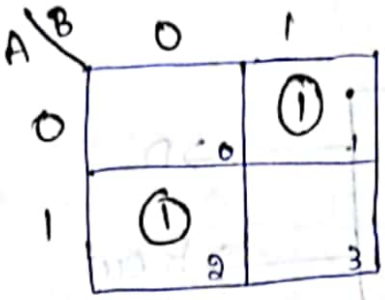


Fig: Logic symbol of Half Subtractor.

Truth Table of Half Subtractor :

Input		Output	
A	B	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-Map for Difference (D)

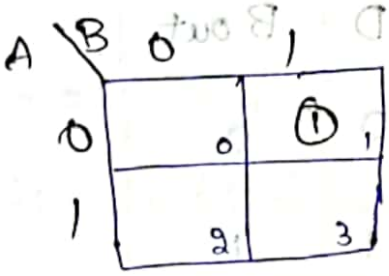


Boolean expression for difference

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

$D = A \oplus B$

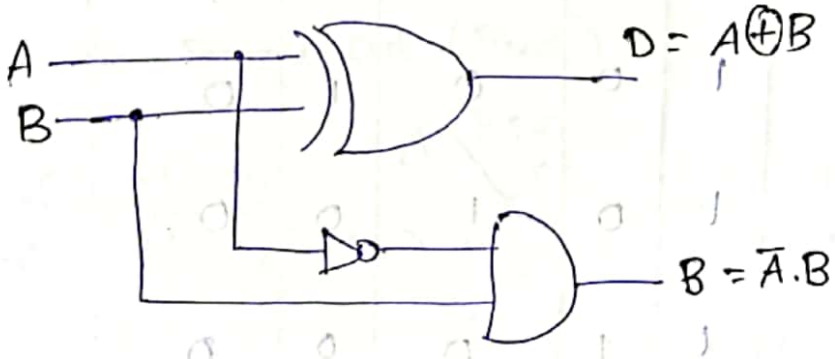
K-Map for Borrow (B)



Boolean expression for borrow

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

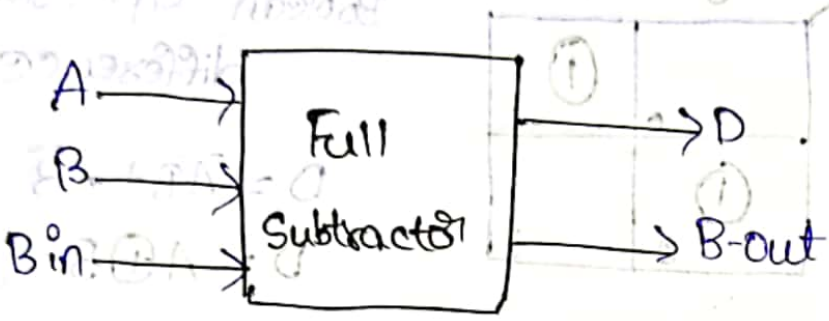
Logic circuit for Half Subtractor :



2. Full Subtractor :

A Full subtractor is a Combinational circuit that perfoms subtraction involving three bits, namely A (minuend), B (subtrahend) and Bin (Borrow-in) as inputs and it produces two outputs : Difference (D) and borrow out (B out).

Logic Symbol for Full Subtractor:

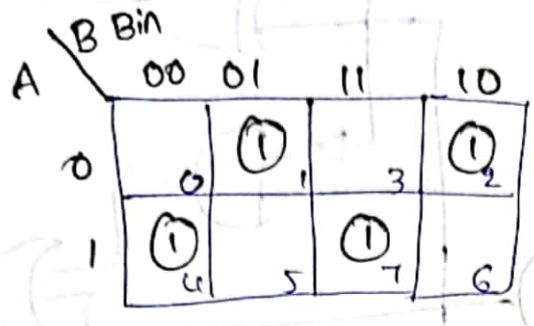


Truth Table for Full Subtractor:

A	B	Bin	D	B out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for Difference (D)

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



$$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$$

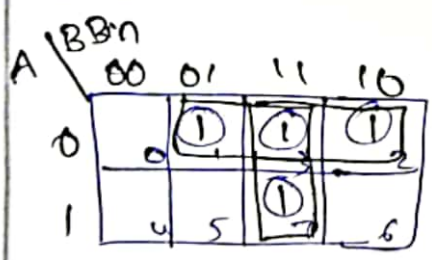
$$= \bar{A}(\bar{B}B_{in} + B\bar{B}_{in}) + A(\bar{B}\bar{B}_{in} + BB_{in})$$

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

$$= A \oplus B \oplus B_{in}$$

$$\therefore D = A \oplus B \oplus B_{in}$$

K-Map for Borrow-out (Bout)



(or)

Boolean expression for Borrow out (B-out):-

$$B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$$

(or)

$$B_{out} = \bar{A}\bar{B}B_{in} + \bar{A}B + ABB_{in}$$

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

$$B_{out} = B_{in}(\overline{A \oplus B}) + \bar{A}B$$

Logic circuit for Full Subtractor: