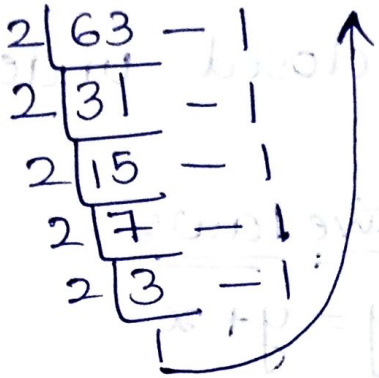
* BOOLEAN ALGEBRA:

⇒ Conversions:

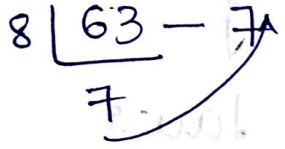
1) Decimal to Binary:

$(63)_{10} = (111111)_2$



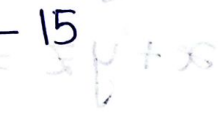
2) Decimal to Octal:

$(63)_{10} = (77)_8$



3) Decimal to Hexadecimal:

$(63)_{10} = (3F)_{16}$



4) Binary to Decimal:

$(11010)_2 = (16+8+0+2)_{10} = (26)_{10}$

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

5) Binary to Octal:

~~$(11010)_8 = 4096 + 8 + 0 + 8 + 0 = 4112$~~

~~$8^4\ 8^3\ 8^2\ 8^1\ 8^0$~~

6) Octal to Decimal:

$(765)_8 = (64 \times 7 + 8 \times 6 + 5)_{10} = (501)_{10}$

$8^2\ 8^1\ 8^0$

7) Hexadecimal to Decimal:

$(1B2)_{16} = 256 + 16 \times 11 + 2 \times 1 = (434)_{10}$

$16^2\ 16^1\ 16^0$

* Properties of Boolean Algebra:

1) Closure:

$B = \{0, 1\}$ $\quad x, y, z \in B$

x, y is closed under : + (or)
: · (and)

2) Commutative Law:

$x + y = y + x$

$x \cdot y = y \cdot x$

3) Distributive Law:

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

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

4) Unary Operator:

$x \in B$

$\exists\, x' \in B$

5) $x + x' = 1 \in B$
$x \cdot x' = 0 \in B$

6) $x, y \in B$ then
$x \neq y$

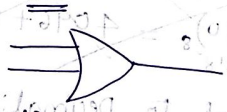
7) Associative Law:

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

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

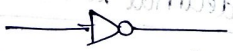
* $x + x = x$
$x \cdot x = x$
$x + xy = x$
$x \cdot 1 = x$
$xy + yx = xy$

* OR



AND

NOT



* De-Morgan's Law:

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

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

* Consensus Theorem:

$xy + x'z + yz = xy + x'z$

* Duality Principle:

The dual of the algebraic expression is given by interchanging OR and AND operators and replace 1's by 0's and 0's by 1's.

Precedence: NOT → AND → OR

BOOLEAN FUNCTION → POS (~~AND gates~~)
→ SOP (~~OR gates~~)

MINTERM: Output of a single cell = 1

MAXTERM: Output of a single cell = 0

→ K-MAP ⇒ 1 is mapped

→ K-MAP ⇒ 0 is mapped.

All the variables appear.

SOP → sum of products : 2 or more product terms are summed up.

POS → product of sums : 2 or more sum terms are multiplied.

* **Canonical SOP**: Sum of minterms.

Canonical POS: Product of maxterms.

* → Conversion of SOP form to standard form: SOP:

1) By multiplying each product term with sum of missing variable and its complement due to which the no of terms gets double.

Ex: $F = xy + yz' + xz$

$F = xy(z+z') + yz(x+x') + xz(y+y')$

$$F = xyz + xyz' + x'yz + xy'z$$

* → Conversion of POS form to standard POS,

By adding each sum term with the product of missing variable & its complement and using $A+BC = (A+B)*(A+C)$

Ex: $F = (A'+B+C)*(B+C+D')*(A+B'+C'+D)$

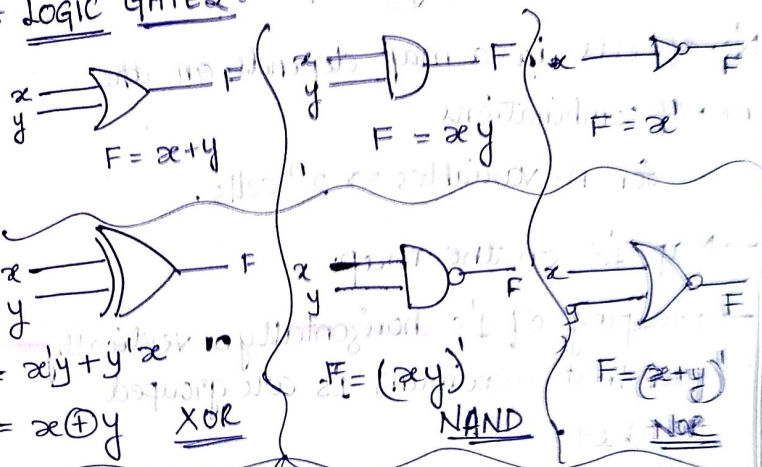
$F = (A'+B+C+DD')*(A\cdot A'+B+C+D')*(A+B'+C'+D)$

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

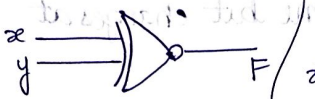
$(A+B'+C'+D)$

~~* KARNAUGH MAPS:~~

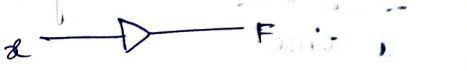
* LOGIC GATES:



XNOR:



Buffer:



SOP → NAND gates } Universal Gates

POS → NOR gates }

XOR → Odd function (odd no of 1's)

~~XNOR~~ → Even function (even no of 1's)

1 → High level → +ve logic system

0 → Low level → -ve logic system

* Gate-Level Minimisation:

K-Map (or) Karnaugh Map:

No. of cells in a map depends on the no. of combinations

for n-variables ⇒ 2^n cells.

→ Map 1's on the map

→ Grouping of 1's horizontally or vertically such that maximum 1's are grouped together.

→ Group must be of powers of 2 (2, 4, 8, ...)

*→ Gorey code: which only one bit changes at a time!

*→ Implicant: '1' that is mapped on K-Map.

*→ Prime Implicant: A prime implicant is a product term obtained by combining maximum possible no: of adjacent squares on map.

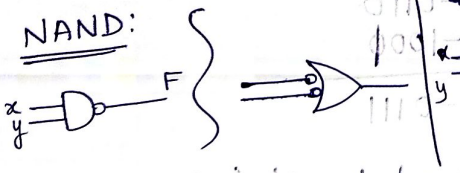
*→ Essential Prime Implicant:

If a minterm in a square is covered by only one prime implicant, then that prime implicant is said to be essential.

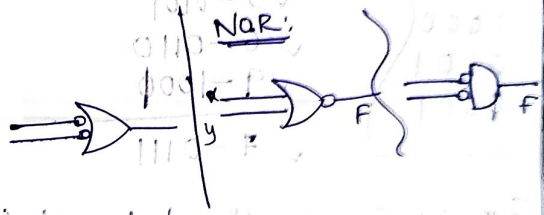
for SOP terms: we map 1 on K-Map and for POS terms: we map 0 on K-Map.

*→ Combination of inputs which give unspecified output are said to be don't care conditions.

NAND:



NOR:



① Expressing SOP using NAND gates:

1) Simplify function & express using SOP form

2) Draw simple OR & AND gates and convert them to NAND by adding appropriate nots.

② Expressing POS using NOR gates:

1) Simplify function & express using POS form.

2) Draw simple OR & AND gates and convert to NOR gates.

* TABULATION METHOD:

→ Ex:

$\Sigma m(0,1,3,6,7) + d(4,5,8,9)$

Step-1: Group according to no. of 1's

0000	✓ 0 – 0000
0001	✓ 1 – 0001
0011	✓ 4 – 0100
0110	✓ 8 – 1000
0111	✓ 3 – 0011
0100	✓ 5 – 0101
1000	✓ 6 – 0110
0101	✓ 9 – 1001
1001	✓ 7 – 0111

Step-2: Group adjacent elements (only one bit change group).

✓ 0,1 – 000–	✓ 3,7 – 0–11
✓ 0,4 – 0–00	✓ 5,7 – 01–1
✓ 0,8 – –000	6,7 – 011–
✓ 1,3 – 00–1	
✓ 1,5 – 0–01	
✓ 1,9 – –001	
✓ 4,5 – 010–	
✓ 4,6 – 01–0	
✓ 8,9 – 100–	

Step-3: Group further if possible having same blank and one bit change.

0,1,4,5 : 0 – 0 – ✓

0,1,8,9 : – 00 – ✓

0,4,1,5 : 0 – 0 – ✓

0,8,1,9 : – 00 – ✓

1,5,3,7 : 0 – – 1 ✓

1,3,5,7 : 0 – – 1 ✓

4,6,5,7 : 01 – –

8 9 10 11 :

If possible group further else.

Step-4: Prime Implicant Chart:

	0	1	3	6	7	4
a'bc				×	×	
a'c'	×	×				
b'c'	×	×			×	
a'd		×	×			
a'b				×	×	

P.I = column with only 1 'x'

$F = a'd + a'b + a'c' = a'd + a'b + b'c$

$= a'd + a'bc + b'c = a'd + a'bc + a'c'$

* The amount of time taken to give the output for given inputs is called propagation delay.

*→ If gates are running parallely the propagation delay will remain same.

*→ Parity Checker & Generator:

- XOR gates are used for error detection and correction codes.

→ parity bit ⇒ used for error detection
- Included with binary message to make the no of 1's either even or odd.

- Circuit generating parity bit : parity generator
- Circuit checking parity bit : parity checker.

*→ Even parity Generator Truth Table:

3-bit:

x	y	z	P_g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

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

*→ Even parity checker for above generator:

x	y	z	P_g	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

$P_c = \Sigma m(1,2,4,7,8,11,13,14)$.

If even parity checker & generator no. of 1's is even and odd parity checker & generator no. of 1's is odd.

Ex: $F = ab + a'b'$

$F^D = (a+b)\cdot(a'+b') \Rightarrow F' = (a'+b')(a+b)$

$F'' = \left((a+b)(a'+b')\right)' = (a+b)' + (a'+b')'$

$= a'b' + ab$