UNIT - II

GATE LEVEL MINIMIZATION

Gate level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.

For simplification of boolean expressions by boolean algebra we need to understand of rules, theorems and boolean laws. For these we need to predict each successive step.

On the other hand the map method gives a systematic approach for simplifying a Boolean expression. The Map method first introduced by VEITCH & modified by KARNAUGH.

So it is known as VEITCH DIAGRAM OR KARNAUGH MA

MAP METHOD:-

The map method provides a simple, straightforward procedure for minimizing Boolean functions. This method is a pictorial form of a Truth table.

↳ The map method is also known as Karnaugh map or K-map.
Every 'n' variable map consists of 2^n cells representing all possible combi's of variables.

K-Map:-

K-Map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized.

The simplified expressions produced by k-map are always in one of the two standard forms

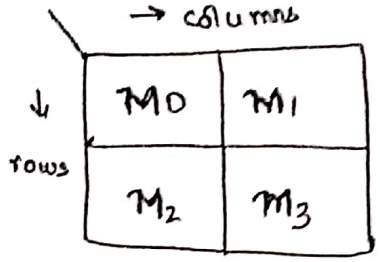
↳ Sum of Products

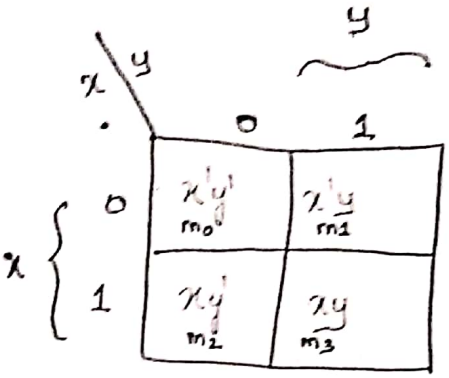
↳ Product of Sums.

It will be assumed that the simplest algebraic expression is an algebraic expression with a minimum number of terms and with the smallest possible number of literals in each term.
This expn produces a ckt diagram with a minimum number of gates and the minimum number of inputs to each gates.

$m_0 = 00$	$x'y'$
$m_1 = 01$	$x'y$
$m_2 = 10$	xy'
$m_3 = 11$	xy

TWO-VARIABLE k-map

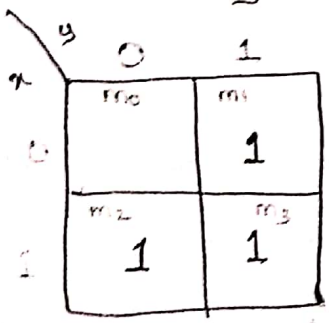




↳ There are 4 minterms $[m_0, m_1, m_2, m_3]$ for 2 variables
↳ 0 & 1 designate the values of variables
↳ Variable x appears printed in row 0 and unprimed in row 1.
↳ Variable y appears printed in column 0 and unprimed in column 1.

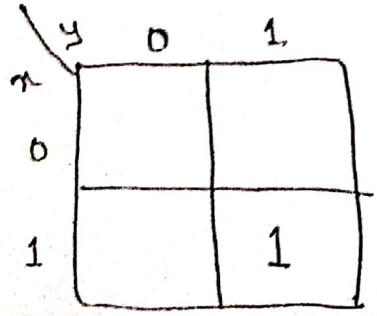
Ex:- $F = x + y$

$m_1 + m_2 + m_3 = x'y + xy' + xy$



$F = x \cdot y$

$m_3 = xy$.



* Any two adjacent cells in the map differ by only one variable, which appears complemented in one cell and uncomplemented in the other.

Ex:- m_0 is adjacent to m_1 & m_2
$m_0 = x_1'y_1'$ & $m_1 = x_1'y_1$
$m_2 = x_1y_1'$

* Grouping of (2Ring) of 1's allows simplfn.

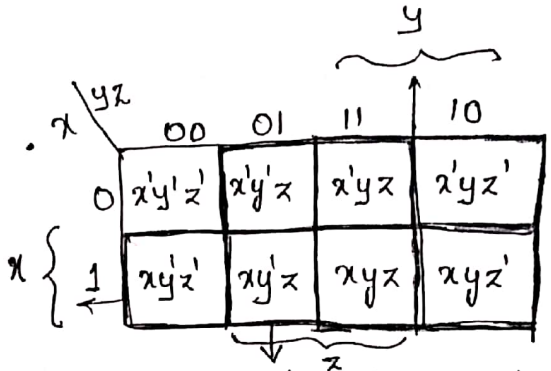
THREE VARIABLE K-MAP:-

There are eight minterms for three binary variables. So, the map consists of eight cells (squares).

** Minterms are arranged in Gray code sequence, but not in binary. Any two Adjacent squares in the map differ by only one variable.

"Only one bit changes in value from one adjacent column to the next".

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



* There are 4 squares in which each variable is equal to 1 & four in which each is equal to zero.

From the postulates of B.A, it follows that the sum of two minterms in adjacent squares can be simplified to a single product term consisting of only two literals.

Consider the sum of two adjacent squares m_5 & m_7.

$$m_5 + m_7 = xyz + xyz = xz(y' + y) = xz$$

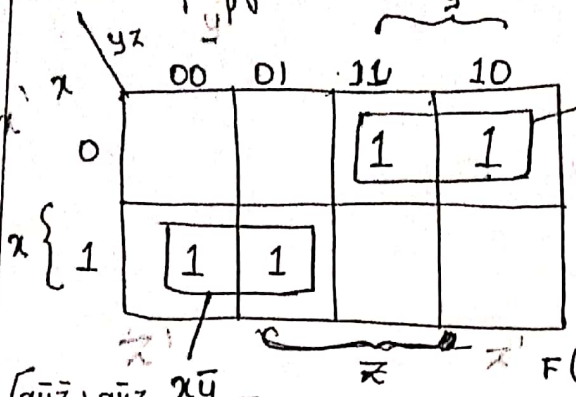
∴ So Any two minterms in adjacent squares (vertically or horizontally, but not diagonally adjacent) that are ORed together will cause a removal of dissimilar variable.

In certain cases, two squares in the map are considered to be adjacent even though they do not touch each other. m_0 is adjacent to m_2 & m_4 is adjacent to m_6 because minterms differ by only one variable.

$$m_0 + m_2 = x'y'z' + x'yz' = x'z'(y + y') = x'z'$$
$$m_4 + m_6 = xy'z' + xyz' = xz'(y' + y) = xz'$$

Problems On 3-Variable K-map:-

(1) ↳ Simplify the Boolean Function $F(x,y,z) = \Sigma(2,3,4,5)$



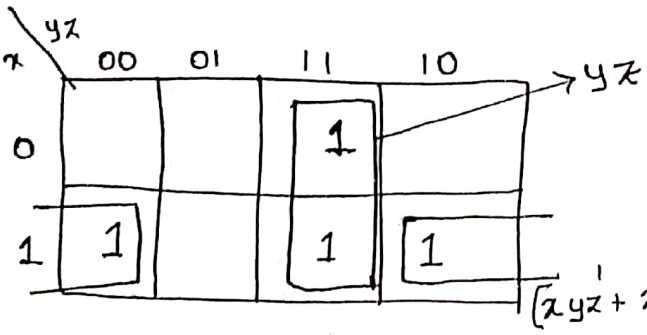
Step-1:- 1 is marked in each minterm squares that represents the function.

Step-2:- Find possible adjacent squares

$F(x,y,z) = \Sigma(2,3,4,5) = x'y + xy'$

From the map, the sum of 4 minterms can be replaced by a sum of only two product terms.

(2) Simplify the Boolean Function $F(x,y,z) = \Sigma(3,4,6,7)$



$\Sigma(3,4,6,7) = yz + xz'$

✱✱ Any combination of four adjacent squares in the three variable map, represents the logical sum of 4 minterms and results in an expression with only one literal.

$m_0+m_2+m_4+m_6 = x'y'z' + x'yz' + xy'z' + xyz'$

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

$= x'z' + xz'$

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

Rules for 3-variable K-map

✱ combine the squares in a power of 2, such as 1, 2, 4, 8.

→ One square represents one minterm, giving a term with three literals.

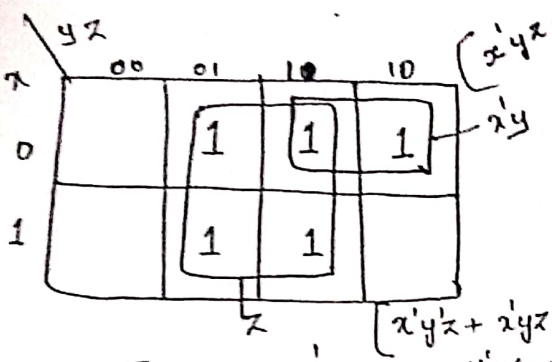
→ Two adjacent squares represent a term with two literals.

→ Four adjacent squares represent a term with one literal.

→ Eight adjacent squares encompass the entire map and produce a function that is equal to 1.

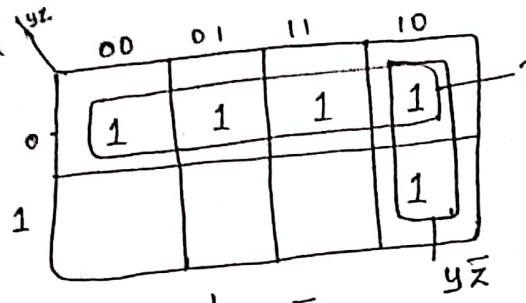
4,5) each

$F(x,y,z) = \Sigma(1,2,3,5,7)$



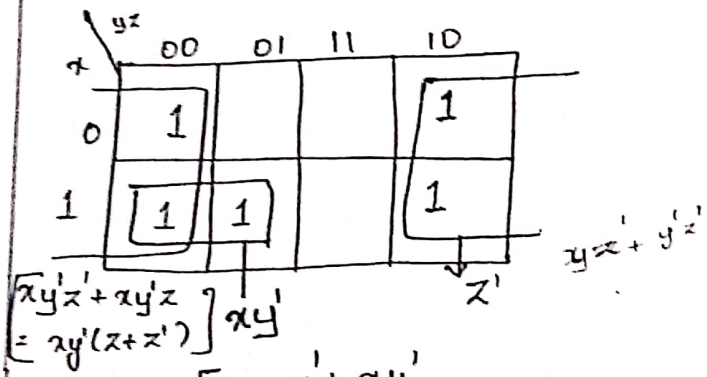
$F = z + x'y$ $= y'z(x+x') + yz(x+x')$ $= y'z + yz = z(y+y')$

(4) $F(x,y,z) = \Sigma(0,1,2,3,6)$



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

(5) $F(x,y,z) = \Sigma(0,2,4,5,6)$.



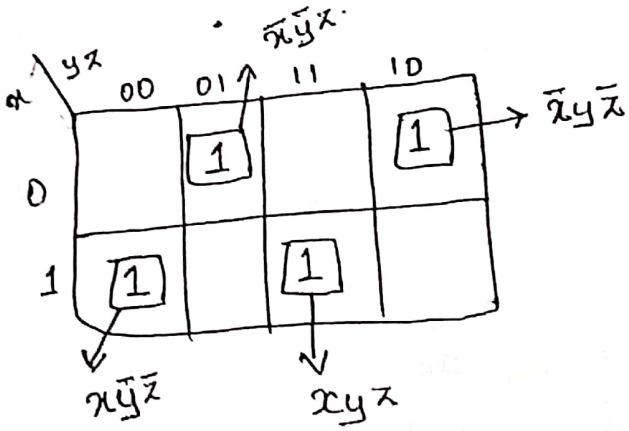
$F = z' + xy'$

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

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

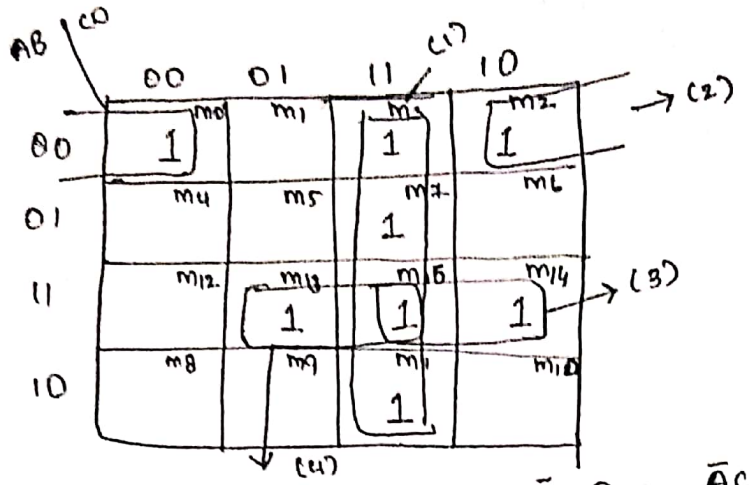
$= \bar{y}\bar{z} + y\bar{z} = \bar{z}(y+y') = \bar{z}$

(6) $F(x,y,z) = \Sigma(1,2,4,7)$



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

(3) $F(A,B,C,D) = \Sigma(0,2,3,7,11,13,14,15)$



(1) $\bar{A}\bar{B}CD + \bar{A}BCD + ABCD + A\bar{B}CD = \bar{A}CD[B+\bar{B}] + ACD[B+\bar{B}] = CD[A+$

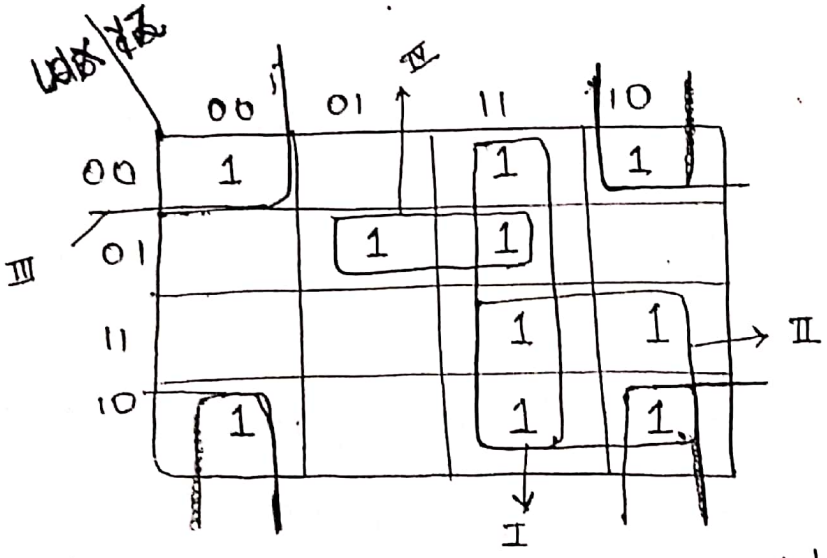
(2) $\bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} = \bar{A}\bar{D}$

(3) $ABCD + ABC\bar{D} = ABC$

(4) $AB\bar{C}D + ABCD = ABD$

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

(4) $F(w,x,y,z) = \Sigma(0,2,3,5,7,8,10,11,14,15)$



I. $w'x'yz + w'xyz + wxyz + wx'yz \Rightarrow xyz(w'+w) + x'yz(w+w') \Rightarrow \underline{yz}(x+x')$

II. $wxyz + wxyz' + wx'yz + wx'yz' \Rightarrow wxy(z+z') + wx'y(z+z') \Rightarrow \underline{wy}(x+x')$

III. $w'x'y'z' + w'x'yz' + wx'y'z' + wx'yz' \Rightarrow w'x'z'(y+y') + wx'z'(y+y') \Rightarrow \underline{x'z'}(w+w'$

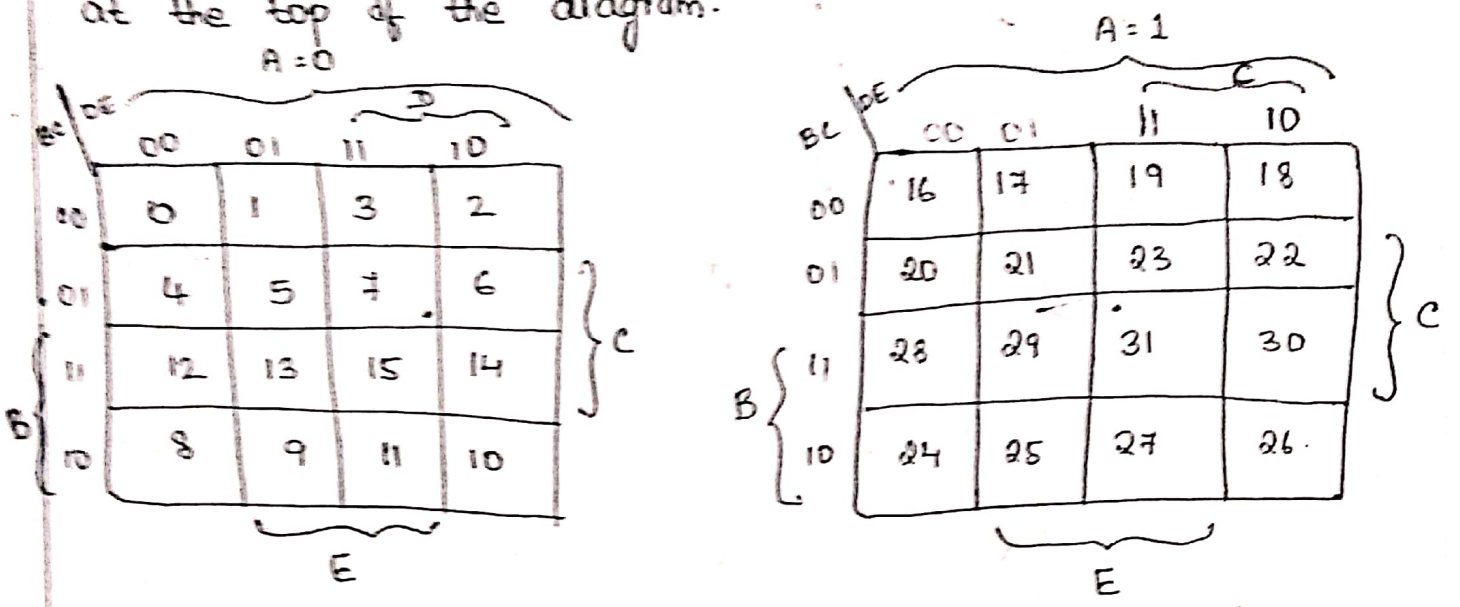
IV. $w'xy'z + w'xyz \Rightarrow \underline{w'xz}(y+y')$

$F = \boxed{yz + wy + x'z' + w'xz}$

FIVE VARIABLE K-MAP.

The five variable K-map consists of 2 four-variable maps with variables A, B, C, D & E.

Variable A distinguishes between the two maps, as indicated at the top of the diagram.



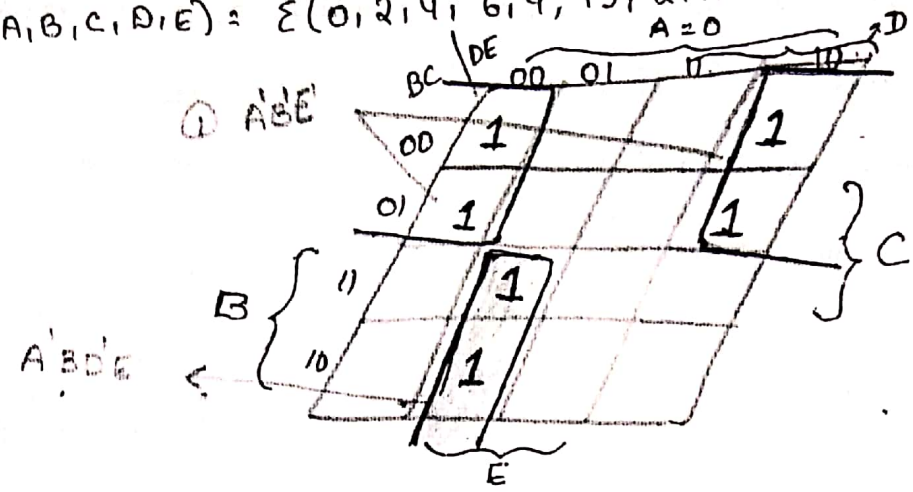
Each square in the A=0 map is adjacent to the corresponding square in the A=1 map.

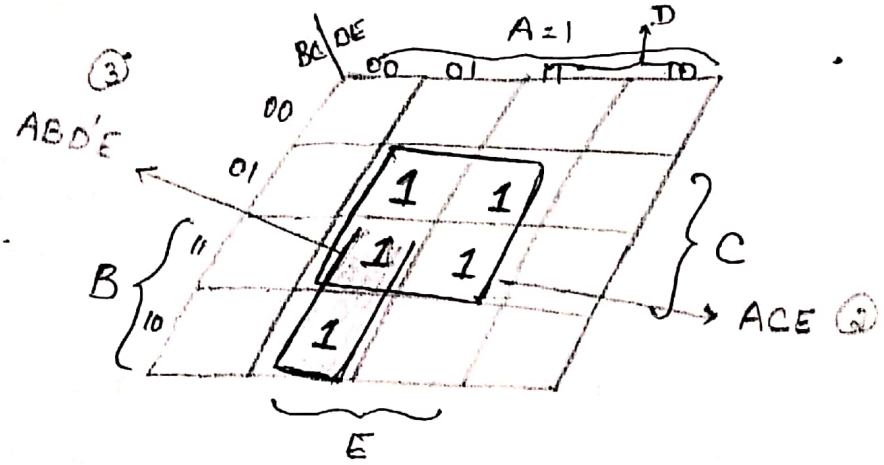
For ex., minterm 4 is adjacent to minterm 20
" 7 is adjacent to " 23
15 is adjacent to " 31

Relationship Between the Number of Adjacent Squares and the Number of literals in the Term.

	Number of Adjacent Squares	Number of literals in a Term in an n-variable Map			
k	2^k	n = 2	n = 3	n = 4	n = 5
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0

(1) $F(A,B,C,D,E) = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$





$$F = A'B'E' + BD'E + ACE$$

① $\bar{A}B\bar{C}\bar{D}\bar{E} + \bar{A}BC\bar{D}\bar{E} + \bar{A}B\bar{C}D\bar{E} + \bar{A}BCD\bar{E}$

$\Rightarrow \bar{A}B\bar{D}\bar{E} + \bar{A}BD\bar{E}$

$\Rightarrow \boxed{\bar{A}B\bar{E}}$

② $A\bar{B}C\bar{D}E + A\bar{B}CDE + ABC\bar{D}E + ABCDE$

$\Rightarrow A\bar{B}CE(D+\bar{D}) + ABCE(D+\bar{D})$

$\Rightarrow ACE(B+\bar{B})$

$\Rightarrow \boxed{ACE}$

③ $\bar{A}BC\bar{D}E + ABC\bar{D}E + \bar{A}B\bar{C}\bar{D}E + AB\bar{C}\bar{D}E$

$\Rightarrow BC\bar{D}E(A+\bar{A}) + (\bar{A}+A)B\bar{C}\bar{D}E$

$\Rightarrow B\bar{D}E$

→ One square/cell gives a term of 5 literals

→ Two adjacent squares gives a term of 4 literals.

→ Four adjacent squares gives a term of 3 literals.

→ Eight adjacent squares gives a term of 2 literals.

→ Sixteen adjacent squares gives a term of 1 literal.

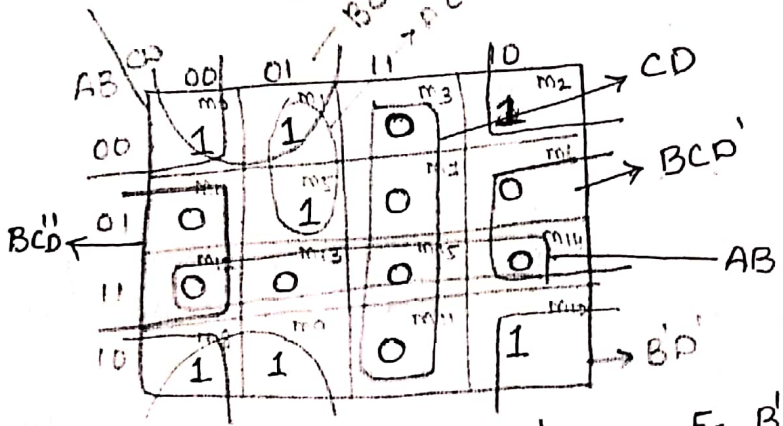
↳ 32 adjacent squares gives 1 Boolean function.

PRODUCT OF SUMS SIMPLIFICATION:-

The procedure for obtaining a minimized function in product-of-sums form follows from the basic properties of Boolean function. The 1's placed in the squares of the map represent the minterms of the function.

The minterms not included in the standard sum of products form a function denote the complement of function. These are called maxterms represented by 0's and called as Product of sums.

Simplify the following Boolean function into
(a) Sum of Products form $F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$
(b) Product of sums form.



$BC'D' + BCD' = BD'$

$F' = AB + CD + BD'$

↓

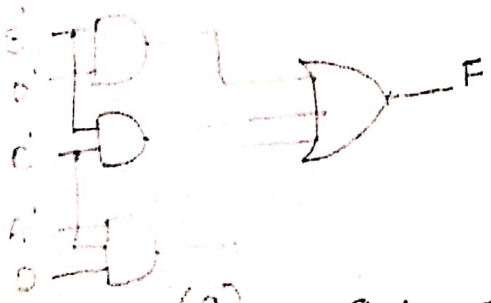
By Applying De-morgans theorem, we obtain the simplified f in product-of-sums form.

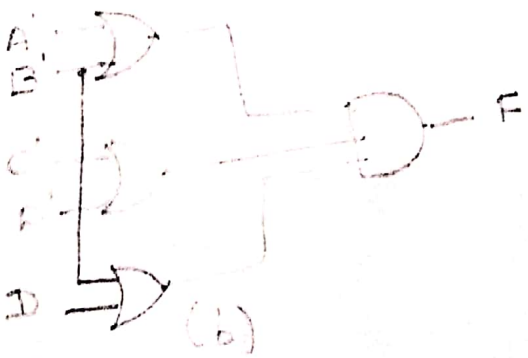
(a)

$F = B'D' + B'C' + A'C'D$ [SOP]

(b)

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





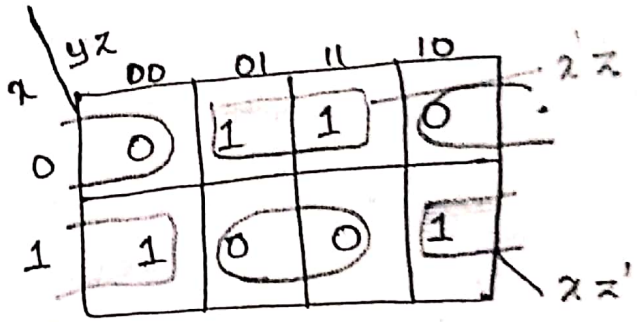
Gate Implementation

* The procedure is also valid when the function originally expressed in the product of maxterms canonical form.

Consider, the Truth table that defines Function.

$F(x,y,z) = \Pi(0,2,5,7)$ in maxterms form.

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



$F = x'z + xz'$

$F' = xz + x'z'$

$\downarrow$

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

So $\Sigma(1,3,4,6)$ are minterms or SOP.

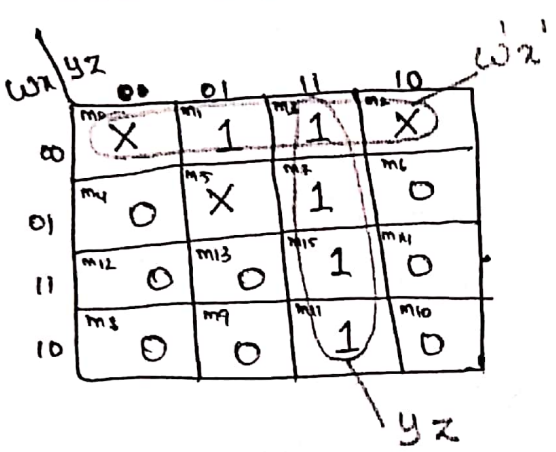
DONT CARE CONDITIONS:-

Don't care conditions can be used on a map to provide further simplification of the Boolean expression. These are represented by 'x'.

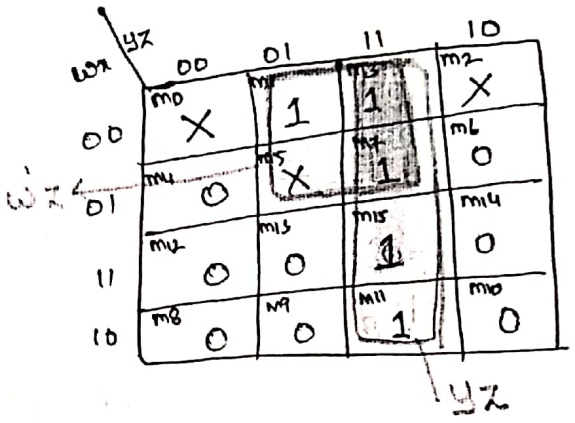
Don't cares in a K-map or truth table may be either 1's or 0's.

Simplify the Boolean Function $F(w,x,y,z) = \Sigma(1,3,7,11,15)$ which has dont care conditions $d(w,x,y,z) = \Sigma(0,2,5)$.

The minterms of d are dont care conditions or dont care minterms that may be assigned either 0 or 1.



$F = yz + w'x'$



$F = yz + w'z$.

Either one of the preceding two expresion satisfies the conditions.

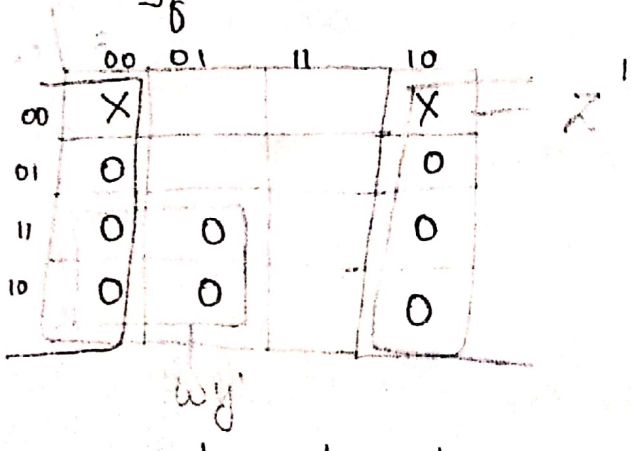
$\therefore F(w,x,y,z) = yz + w'x' = \Sigma(0,1,2,3,7,11,15)$

$\therefore F(w,x,y,z) = yz + w'z = \Sigma(1,3,5,7,11,15)$.

This is for simplified sum of products.

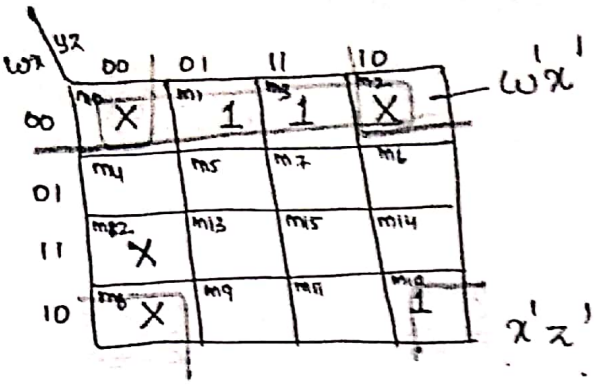
It is also possible to obtain a ~~Sum of products~~ product of sums expression for the given function.

If we combine maxterms



$\therefore F' = z' + wy' = \Sigma(w'+y')$ =

② Simplify the Boolean ∃ⁿ $F(w,x,y,z) = \Sigma(1,3,10) + \Sigma_d$ 2,8,11,12



③ Simplify the Boolean ∃ⁿ. $F(A,B,C,D) = \Sigma_m(0,2,6,10,11,12,13) +$ $\Sigma d(3,4,5,1$ 15)

NAND AND NOR IMPLEMENTATION:-

These are also called as "UNIVERSAL GATES"

Digital circuits are frequently constructed with NAND OR NOR gates rather than with AND, OR. Nand and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.

NAND circuits:-

The ~~NAND~~ NAND gate is be universal gate because any logic circuit can be implemented with it.

** "A convenient way to implement a Boolean function with NAND gates is to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND logic.

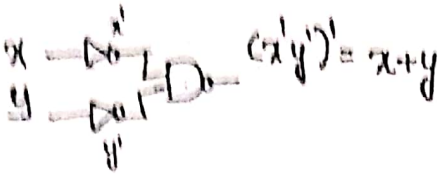
NOR circuits:-

NOR oper^n is the dual of NAND oper^n. This is another universal gate to implement any Boolean ∃ⁿ.

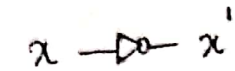
Logic operations with NAND GATES

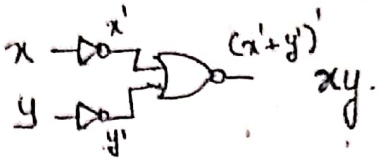
Inverter 

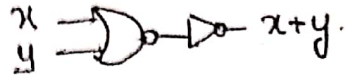
AND 

OR 

Logic open's with NOR GATES

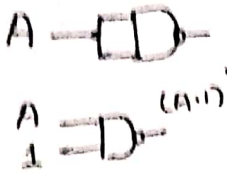
Inverter 

AND 

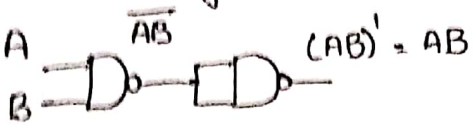
OR 

USING ONLY NAND

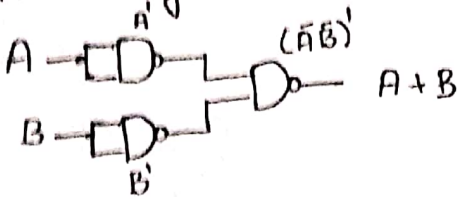
(1) Inverter using NAND



(2) AND using NAND.

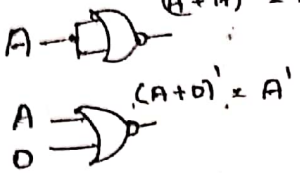


(3) OR using NAND

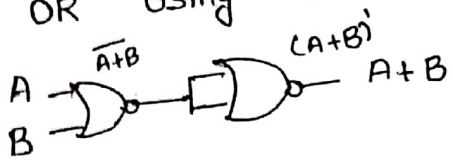


USING ONLY NOR.

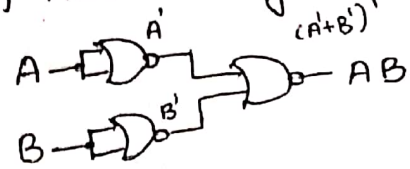
(1) Inverter using NOR.



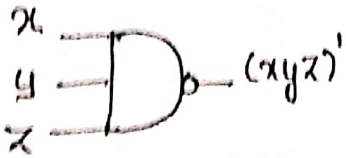
(2) OR using NOR



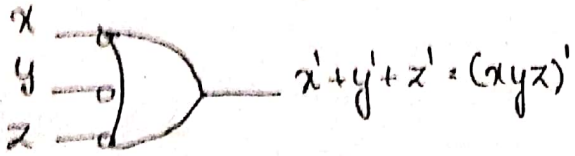
(3) AND using NOR



AND-Invert is same as Invert OR

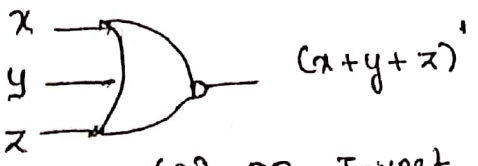


(a) AND- Invert

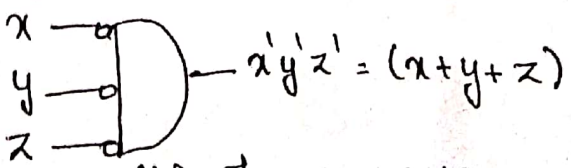


(b) Invert - OR

OR Invert is same as Invert AND.



(a) OR- Invert

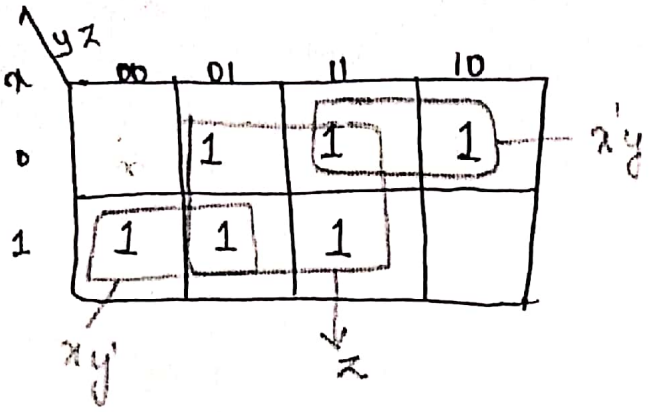


(b) Invert - AND.

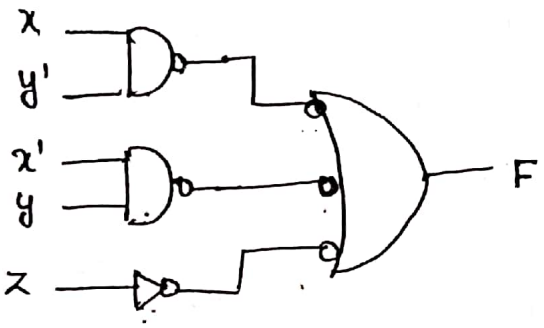
TWO LEVEL IMPLEMENTATIONS:-

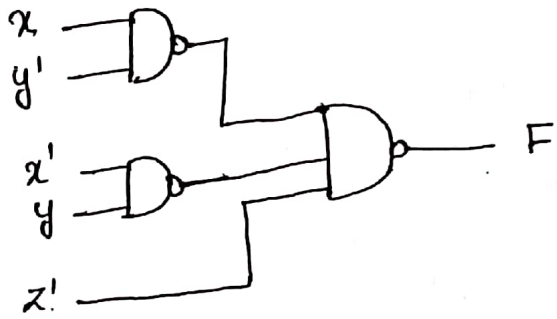
(1) Implement the following Boolean ∃ⁿ with NAND gates.

$F(x, y, z) = (1, 2, 3, 4, 5, 7)$.



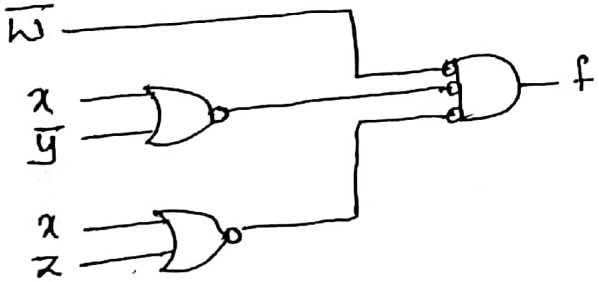
$F = xy' + x'y + z$

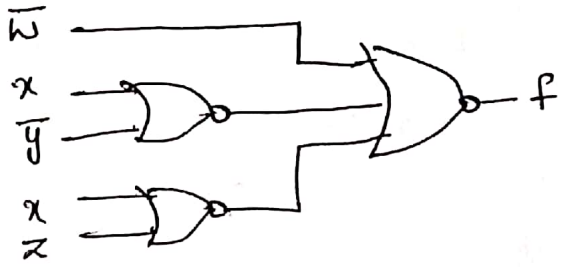




(2) Implement the following ∃ⁿ with NOR Gates

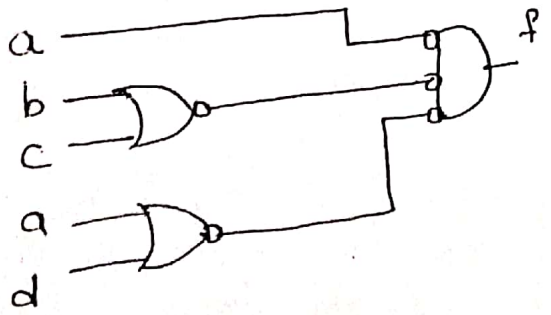
$F = w(x+y)(x+z)$

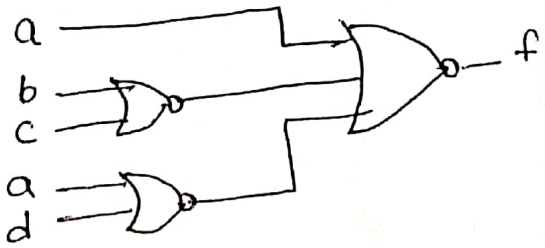




(3) Implement the following ∃ⁿ with NOR Gates.

$f = \bar{a}(b+c)(a+d)$





OTHER TWO-LEVEL IMPLEMENTATIONS:-

Consider four Types of Gates AND, OR, NAND, NOR

If we assign one Type of gate for first level and one Type for second level, there are 16 possible combinations of Two-level forms.

↳ Degenerate Forms:-

Same Type of gates in First level and Second level which degenerates to a single oper^n. There are eight.

Ex:- NAND - NAND
AND - AND
OR - OR
Nand - OR
NOR - AND
OR - NOR
Nand - NOR
NOR - Nand
etc.,

↳ Non Degenerate forms:-

The remaining eight non-degenerate forms produce an Implementation in

↳ Sum of products
(or)
↳ Product of Sum.

The Eight Non Degenerate forms are

↳ AND - OR
↳ NAND - NAND
↳ NOR - OR
↳ OR - NAND
↳ OR - AND
↳ NOR - NOR
↳ NAND - AND
↳ AND - NOR

The AND-OR and OR-AND are the basic Two-level implementations, NAND-NAND & NOR-NOR were also discussed

The remaining Four Forms

↳ NAND-AND, AND-NOR, NOR-OR, OR-NAND.

AND-OR-INVERT IMPLEMENTATION:-

NAND-AND } are equivalent and
AND-NOR } can be treated together.
Both perform "AND-OR-INVERT" Function.

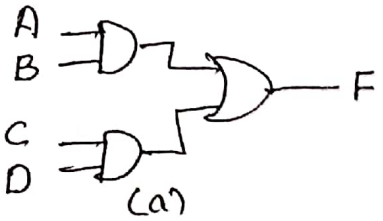
The AND-NOR form resembles the AND-OR form, but with an inversion done by the bubble in output of NOR gate.

$F = (AB + CD + E)'$

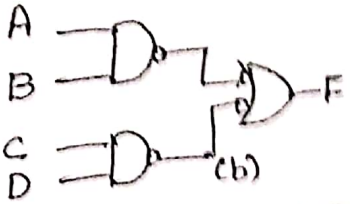
TWO LEVEL IMPLEMENTATION:-

The impln of Boolean functions with NAND gates requires that the functions be in "Sum of products" form.

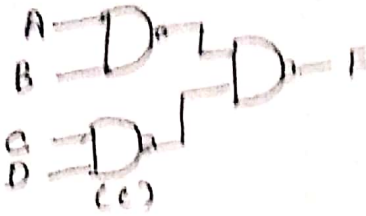
$F = AB + CD$



(a)



(b)



(c)

Three ways to implement $F = AB + CD$.

In (c) $F = ((AB')(CD)')' = AB + CD$.

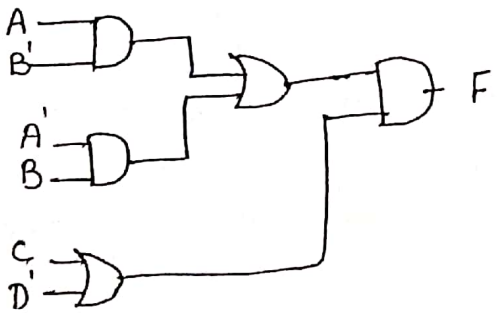
NOTE

** A logic diagram with a pattern of alternate levels of AND & OR gates can be easily converted into a NAND circuit with use of mixed Notation.

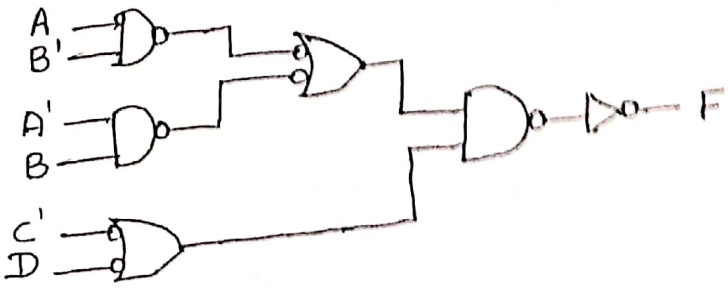
** The procedure is to change every AND gate to AND-invert graphic symbol and every OR gate to an Invert-OR graphic Symbol.

* Example for Multilevel Boolean function

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

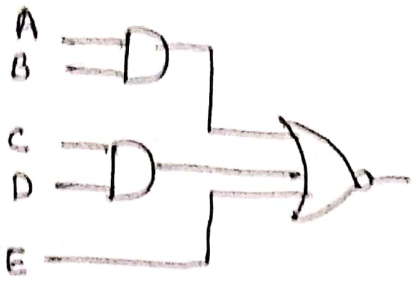


(a) AND-OR Gates

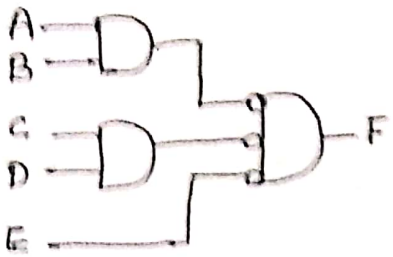


(b) NAND gates

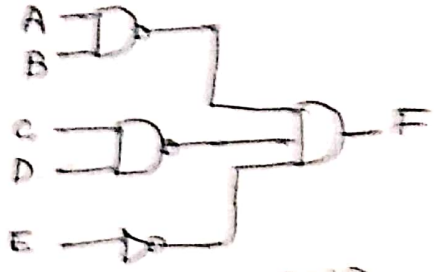
$F = (AB + CD + E)'$



(a) AND - NOR



(b) AND - NOR



(c) NAND - AND

"AND - OR - INVERT" circuits

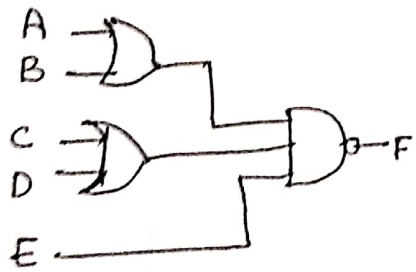
* OR - AND - INVERT IMPLEMENTATION:-

OR - NAND } Performs the "OR - AND - INVERT" fn.
NOR - OR }

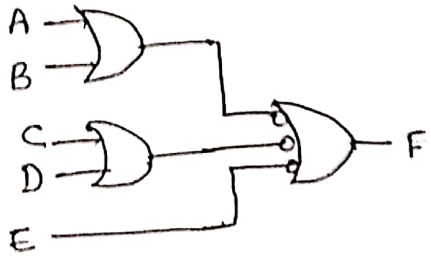
The OR - NAND form resembles the OR - AND form except for the inversion done by bubble in the NAND gate.

It Implements the function

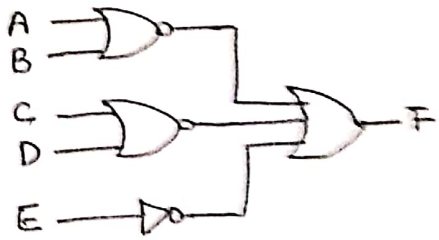
$F = [(A+B)(C+D)E]'$



(a) OR-NAND



(b) OR - NAND



(c) NOR - OR

* OR - AND - INVERT impl^n requires an expr^n to be in POS Form
* AND - OR - INVERT impl^n requires an expr^n to be in SOP Form

Implement the F with the four 2-level forms:

The complement of F is simplified into sum-of products form by combining 0's in the map.

$F' = x'y + xy' + z$ So, $F = (x'y + xy' + z')'$

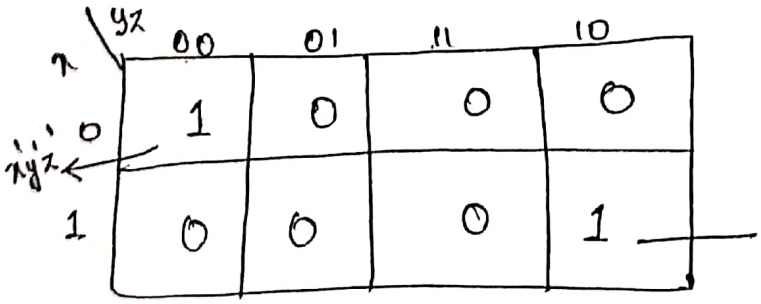
↓ AND-OR-Invert Form.

$F = x'y'z' + xyz'$ From this again complement.

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

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

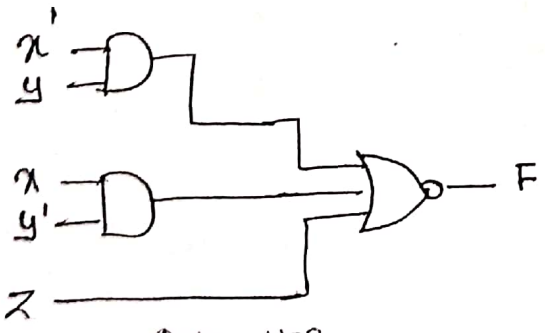
Which is OR-AND invert form. From this we can Implement the function in OR-NAND & NOR-OR



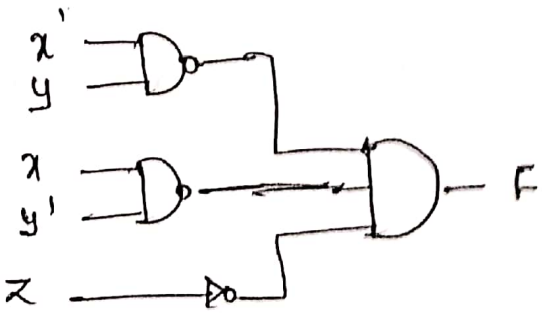
$F = x'y'z' + xyz'$

$F' = x'y + xy' + z$

Map Simplifn in Sum of Products.

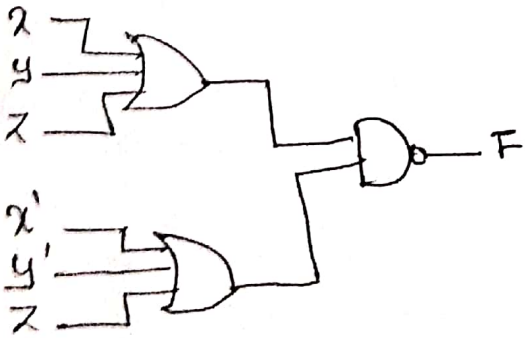


AND-NOR

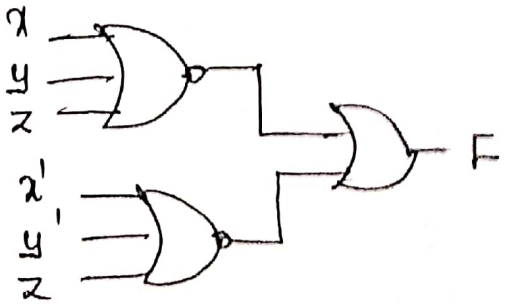


NAND-AND.

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



OR-NAND



NOR-OR

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

EXCLUSIVE OR FUNCTION:-

$x \oplus y = xy' + x'y$

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

[$\because (xy' + x'y)'$
$= (x'+y)(x+y')$
$= xy + x'y'$]

↳ PROPERTIES OF XOR

$\therefore x \oplus 0 = x$

$x \oplus 1 = x'$

$x \oplus x = 0$

$x \oplus x' = 1$

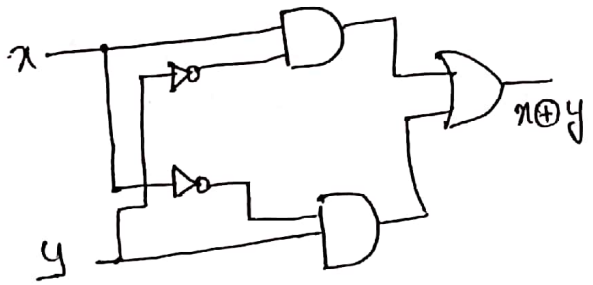
$x \oplus y' = x' \oplus y = (x \oplus y)'$

$\therefore$ EX-OR Oper^n is both commutative & Associative.

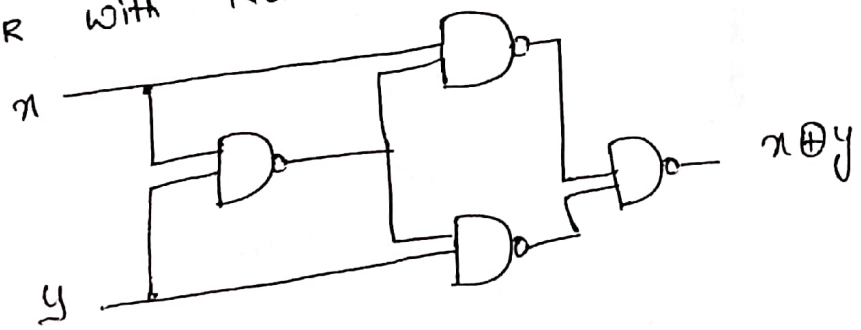
$A \oplus B = B \oplus A$

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

EX-OR with And-OR-NOT gates:-



EX-OR with Nand gates:-



* ODD FUNCTION:-

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

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

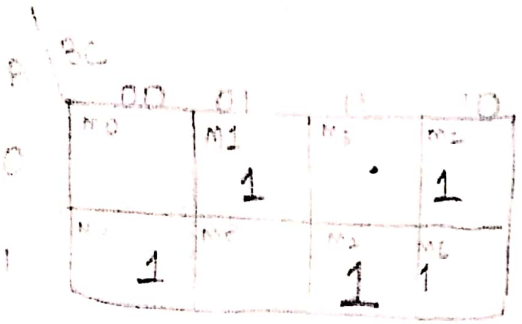
$= \Sigma(1, 2, 4, 7)$

Three Variable X-OR f^n is equal to 1 if only one Variable is equal to 1 or if all three variables are equal to 1.

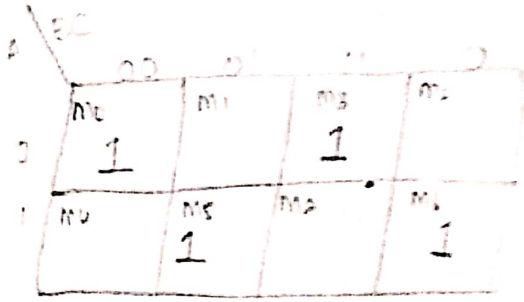
∴ The Multiple Variable exclusive - OR oper^n is defined as an "ODD FUNCTION".

The remaining four minterms not included in the f^n are 000, 011, 101 & 110 [0, 3, 5, 6].

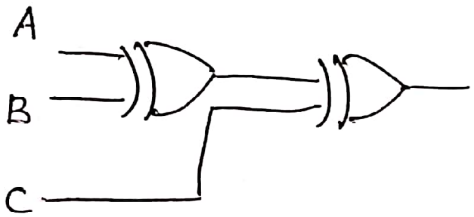
∴ So, the complement of Odd F^n is an Even F^n.

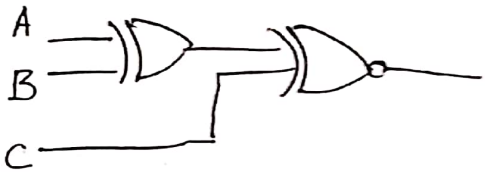


(a) Odd F^n F = A⊕B⊕C



(b) Even F^n F = (A⊕B⊕C)'





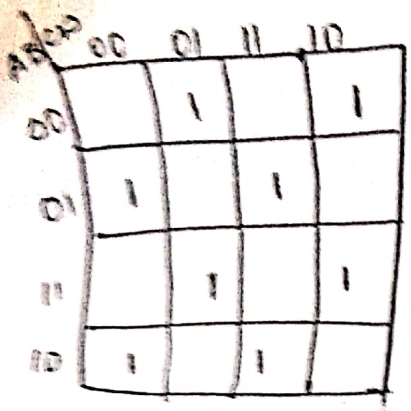
PARITY GENERATION & CHECKING:-

EX-OR F^n's are very useful in systems requiring Error Detection & Correction.

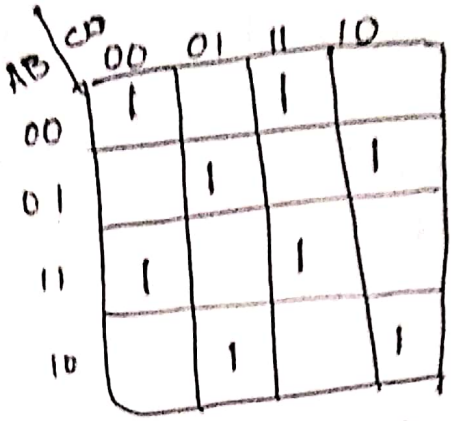
PARITY:- A Parity bit is used for the purpose of detecting errors during the transmission of binary information.

A parity bit is an extra bit is an extra bit included with a binary message to make the number of 1's either odd or even.

↳ The circuit that generates the parity bit in the Transmitted is called "Parity Generator." The circuit that checks parity in receiver is called "Parity checker."



(a) odd f^n $F = A \oplus B \oplus C \oplus D$

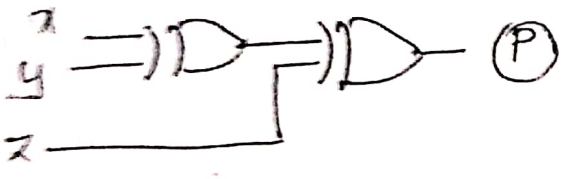


(b) Even f^n $F = (A \oplus B \oplus C \oplus D)'$

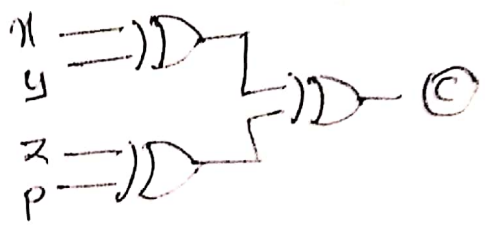
$$P = x \oplus y \oplus z$$

Even Parity Generator Truth Table

Three Bit Message			Parity Bit
x	y	z	P
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



3-bit even parity Generator



4-bit Even Parity checker

$$C = x \oplus y \oplus z \oplus P$$

Even Parity checker Truth Table

Four Bits Received				Parity Error check
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	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	