Module 5: Logic Gates and Boolean Algebra

! "
$ % & ' ! (
) * + , - & ' * ./0
1 ! ' 2 ! 3 24 2 ! ' ! 32 4 56- 0 0 7 3!0 1 % 4 & ' !
!
, 0 ! ! %

5.1 Logic Gates: AND, OR, NOT, NAND, NOR, XOR – Symbols and Truth Tables

Logic gates play an important role in circuit design and digital systems. It is a building block of a digital system and an electronic circuit that always have only one output. These gates can have one input or more than one input, but most of the gates have two inputs. On the basis of the relationship between the input and the output, these gates are named as AND gate, OR gate, NOT gate, etc.

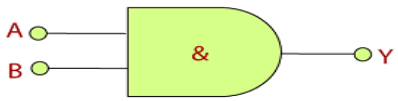
AND Gate

The AND gate plays an important role in the digital logic circuit. The output state of the AND gate will always be low when any of the inputs states is low.

Simply, if any input value in the AND gate is set to 0, then it will always return low output(0).

The logic or Boolean expression for the AND gate is the logical multiplication of inputs denoted by a full stop or a single dot as

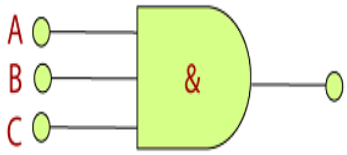
A.B=Y



2- Input AND Gate

Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

The 3-input AND Gate



3- Input AND Gate

Input			Output
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

AND Gate (·)

Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

This is basically AND Gate one of the interesting thing here to notice is that except for both Inputs to be 1 rest all outputs are 0

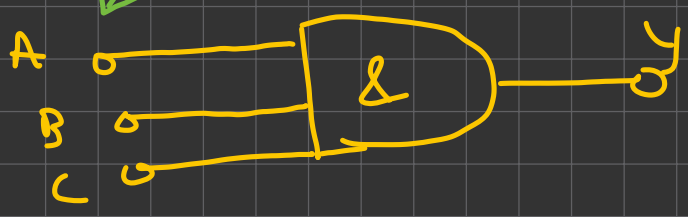
This is the diagram for AND gate

2 input AND gate

Input			Output
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Notice how its the same thing even for 3 I/p & gate

This is the diagram for 3 I/p & gate



$A \cdot B = Y$

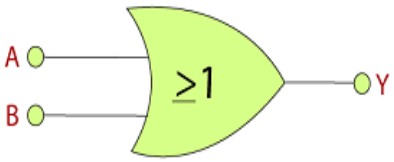
OR Gate

The OR gate is a mostly used digital logic circuit. The output state of the OR gate will always be low when both of the inputs states is low.

Simply, if any input value in the OR gate is set to 1, then it will always return high-level output(1).

The logic or Boolean expression for the OR gate is the logical addition of inputs denoted by plus sign(+) as

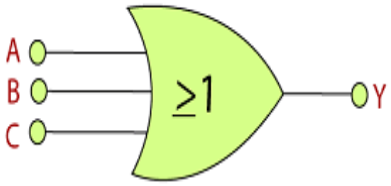
A+B=Y



2- Input OR Gate

Inputs		Output
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

The 3-input OR gate



3- Input OR Gate

Input			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

NOT Gate

The NOT gate is the most basic logic gate of all other logic gates. NOT gate is also known as an inverter or an inverting Buffer. NOT gate only has one input and one output. When the input signal is "Low", the output signal is "High" and when the input signal is "High", the output is "Low".

The Boolean expression for the NOT gate is as follows:

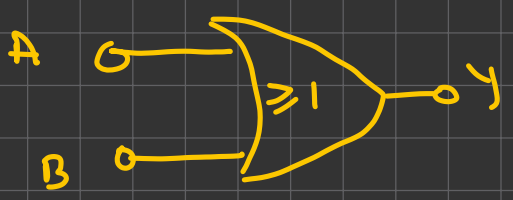
A'=Y

When A is not true, then Y is true

OR Gate (+)

Inputs		Output
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

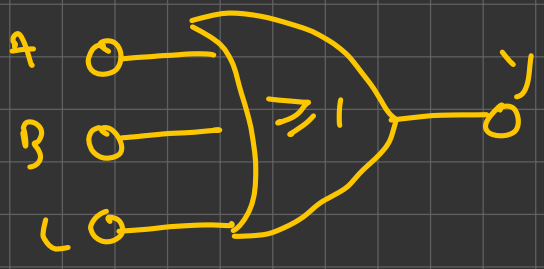
In the OR Gate the logic is basically that unless the value of both I/P is 0 O/P will always be (1)



This is the diagram of 2 I/P OR Gate. A+B = Y

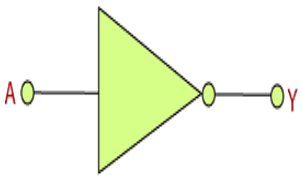
Input			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Again the same thing for the 3 I/P OR gate unless all A, B, C are 0, 0, 0 The o/p will always be (1)



This is the diagram for 3 I/P OR gate

The standard NOT gate is given a symbol that is shaped like a triangle with a circle at the end, pointing to the right. This circle is known as an "invert bubble" and is used to represent the logical operation of the NOT function in the NOT, NAND and NOR symbols in their output.



NOT Gate

Input	Output
A	B
0	1
1	0

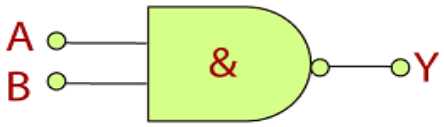
NAND Gate

The NAND gate is a special type of logic gate in the digital logic circuit. The NAND gate is the universal gate. It means all the basic gates such as AND, OR, and NOT gate can be constructed using a NAND gate. The NAND gate is the combination of the NOT-AND gate. The output state of the NAND gate will be low only when all the inputs are high.
Simply, this gate returns the complement result of the AND gate.

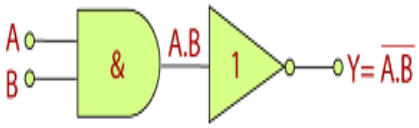
The logic or Boolean expression for the NAND gate is the complement of logical multiplication of inputs denoted by a full stop or a single dot as

(A.B)'=Y

The value of Y will be true when any one of the input is set to 0.



2- Input NAND Gate



2- Input "AND" gate plus a "NOT" gate

Inputs		Output
A	B	(AB)'
0	0	1
0	1	1
1	0	1
1	1	0

NOT Gate (')

Input	Output
A	B
0	1
1	0

Its job is to basically invert the Input whatever it's taking

$A' = Y$ ← Boolean expression

← Diagram for an NOT gate

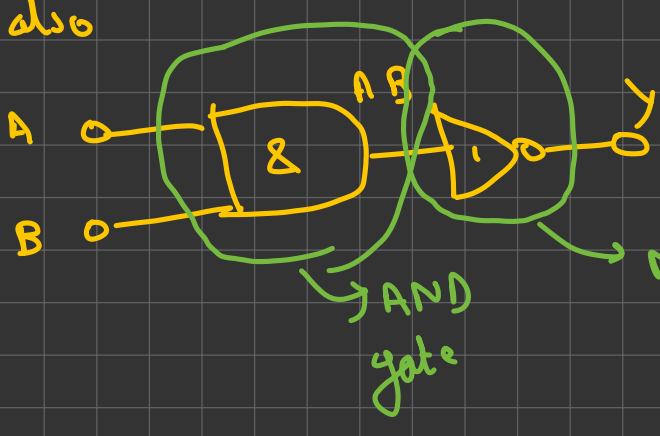
NAND Gate

Inputs		Output
A	B	(AB)'
0	0	1
0	1	1
1	0	1
1	1	0

Simply speaking this is not + AND gate, as clear from the table when both of the inputs are high it's o/p its low or 1 1 ⇒ 0

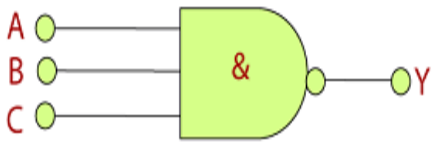
← This is the diagram of the nand gate

also



← 2 Input NAND gate

The 3-input NAND Gate



3- Input NAND Gate

Input			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

NOR Gate

The NOR gate is also a universal gate. So, we can also form all the basic gates using the NOR gate.

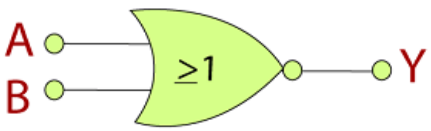
The NOR gate is the combination of the NOT-OR gate. The output state of the NOR gate will be high only when all of the inputs are low.

Simply, this gate returns the complement result of the OR gate.

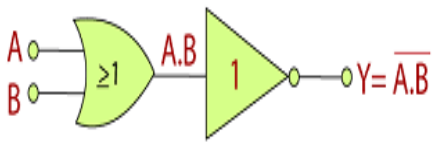
The logical or Boolean expression for the NOR gate is the complement of logical multiplication of inputs denoted by the plus sign as

(A+B)'=Y

The value of Y will be true when all of its inputs are set to 0.



2- Input NOR Gate



2- Input "AND" gate plus a "NOT" gate

Inputs		Output
A	B	(AB)'
0	0	1
0	1	0
1	0	0
1	1	0

Input			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

An 3 input NAND gate as evident from the table when all 3 are high (1) then the o/p given by the NAND gate is low (0)

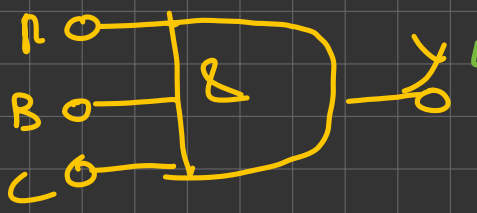


Diagram of an 3 input NAND gate

NOR Gate

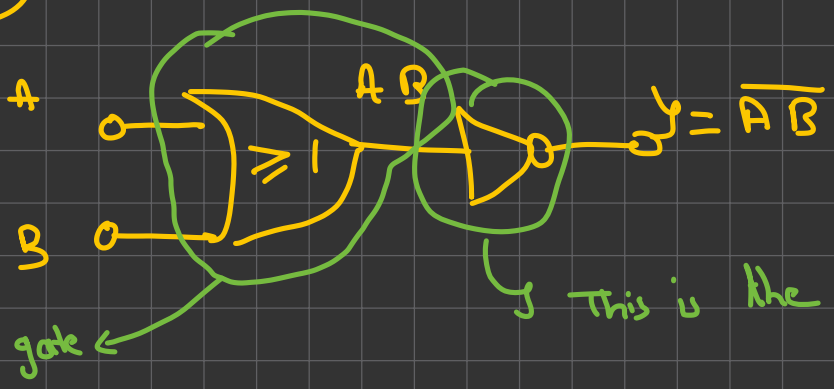
Inputs		Output
A	B	(AB)'
0	0	1
0	1	0
1	0	0
1	1	0

→ It's NOT + OR Gate

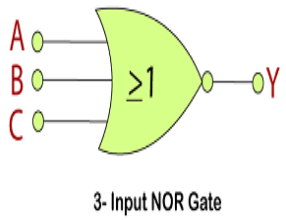
→ Boolean Exp $Y = \overline{A+B}$

→ As, visible when both of the inputs A & B are 0 then the o/p is the highest that is (1) while for the rest its (0)

Diagram of an 2 I/P NOR gate



The 3-input NOR gate



3- Input NOR Gate

Input			Output
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

XOR Gate

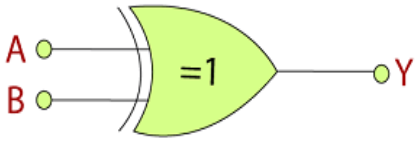
The XOR gate stands for the Exclusive-OR gate. Apart from the AND, OR, NOT, NAND, and NOR gate, there are two special gates, i.e., Ex-OR and Ex-NOR. These gates are not basic gates in their own and are constructed by combining with other logic gates. The XOR and XNOR gates are the hybrids gates.

The 2-input OR gate is also known as the Inclusive-OR gate because when both inputs A and B are set to 1, the output comes out 1(high). In the Ex-OR function, the logic output "1" is obtained only when either A="1" or B="1" but not both together at the same time.

Simply, the output of the XOR gate is high(1) only when both the inputs are different from each other. The plus(+) sign within the circle is used as the Boolean expression of the XOR gate. So, the symbol of the XOR gate is .

This Ex-OR symbol also defines the "direct sum of sub-objects" expression.

Y=(A B)
Y=(A' B+AB')

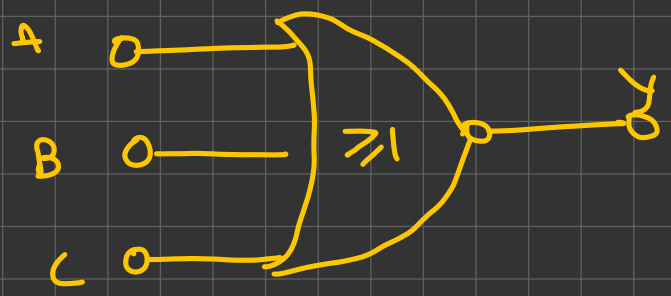


2-Input XOR Gate

Inputs		Output
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Input			Output
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

This one is for the 3 I/P NOR gate this one is the same as that of the previous one i.e it give o/p as 1 when all I/P are 0, 0, 0



← This is the diagram for 3 I/P NOR gates

XOR (Gate (⊕))

Inputs		Output
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Basic logic of the XOR gate, Simply put if either of the I/P A or B are different then the o/p is (1) else it's always equals to (0)

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

$$Y = A'B + B'A$$

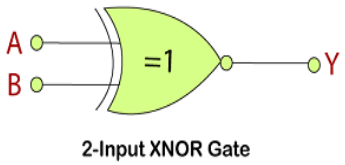
← Boolean Exp for XOR Gate

XNOR Gate

The XNOR gate is the complement of the XOR gate. It is a hybrid gate. Simply, it is the combination of the XOR gate and NOT gate. The output level of the XNOR gate is high only when both of its inputs are the same, either 0 or 1. The symbol of the XNOR gate is the same as XOR, only complement sign is added. Sometimes, the XNOR gate is also called the Equivalence gate.

Y=(A B)'

Y=((AB)'+AB)



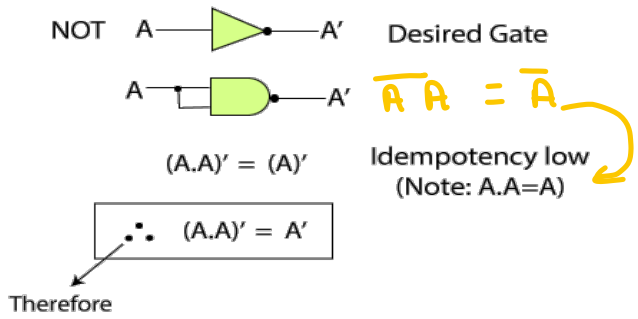
2-Input XNOR Gate

Inputs		Output
A	B	A⊖B
0	0	1
0	1	0
1	0	0
1	1	1

5.2 Universal Gates: Realization of basic gates using NAND/NOR

NAND gate and NOR gate are called ! % because any logic function (NOT, AND, OR, XOR, etc.) can be implemented using only NAND or only NOR.

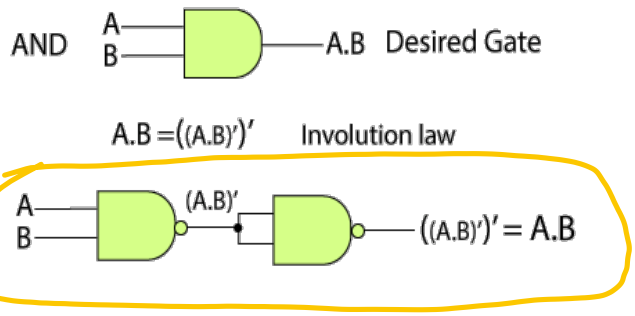
NOT gate using NAND gate

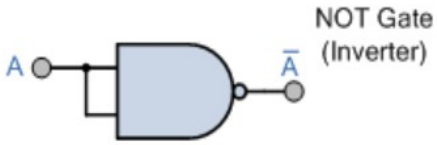


The Derivation of the NOT Gate

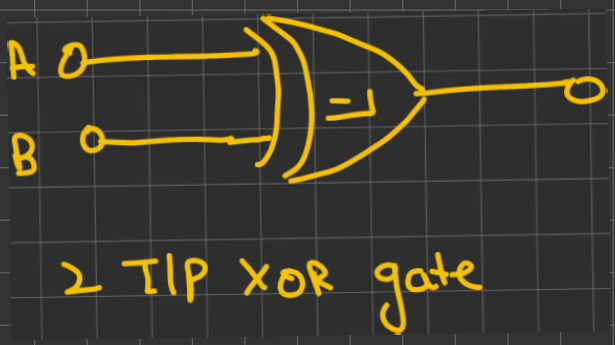
AND gate using NAND gate

$(AB)' = \overline{AB}$





The Derivation of the AND Gate



2 I/P XOR gate

XNOR Gate(⊙):-

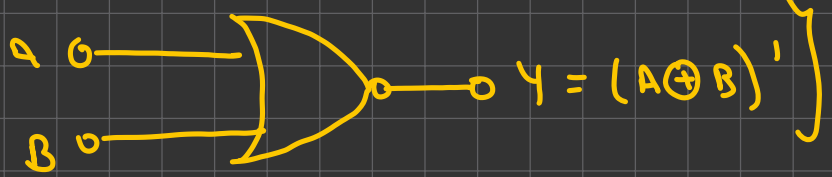
Inputs		Output
A	B	A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

It's basically the inverse to that of XOR gate i.e when both of the I/Ps are opposite then it gives the o/p as 0 else it is 1

$Y = (A \oplus B)'$

$Y = (AB)' + AB$

This is the expression for XNOR gate



2 I/P XNOR Gate

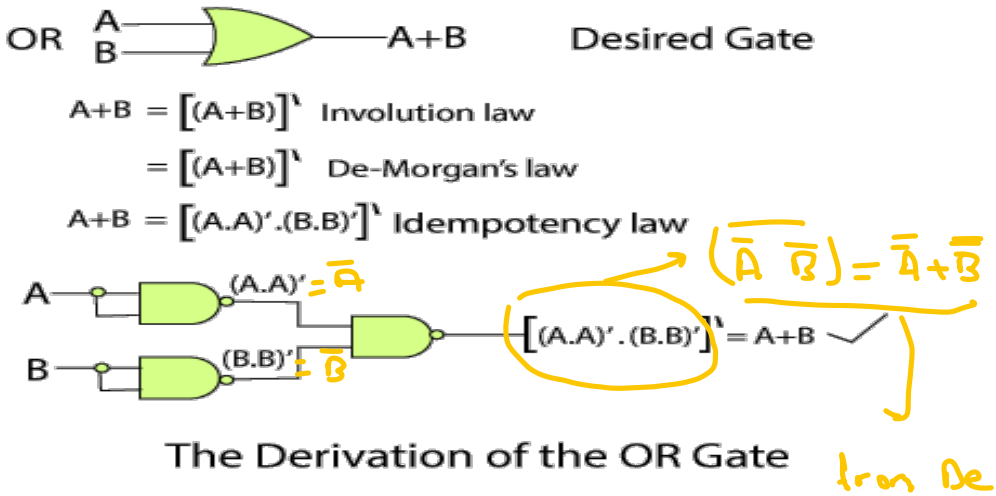
OR gate using NAND gate.

From

Idempotency law

B.B = B

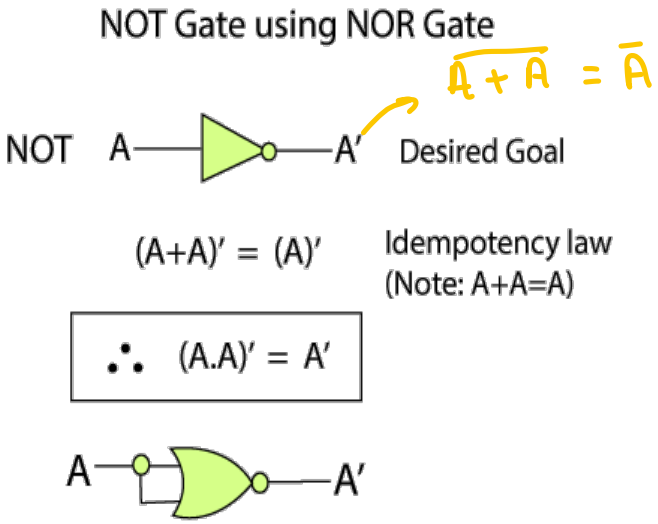
B+B = B



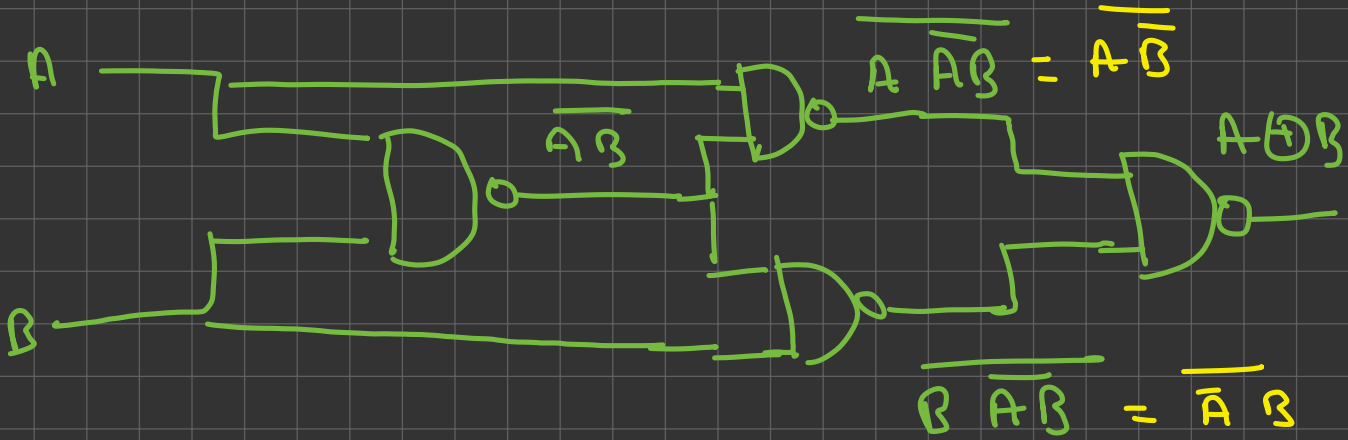
NOR Gate as universal gate

The NOR gate is also a universal gate. So, we can also form all the basic gates using the NOR gate.

NOT gate using NOR gate



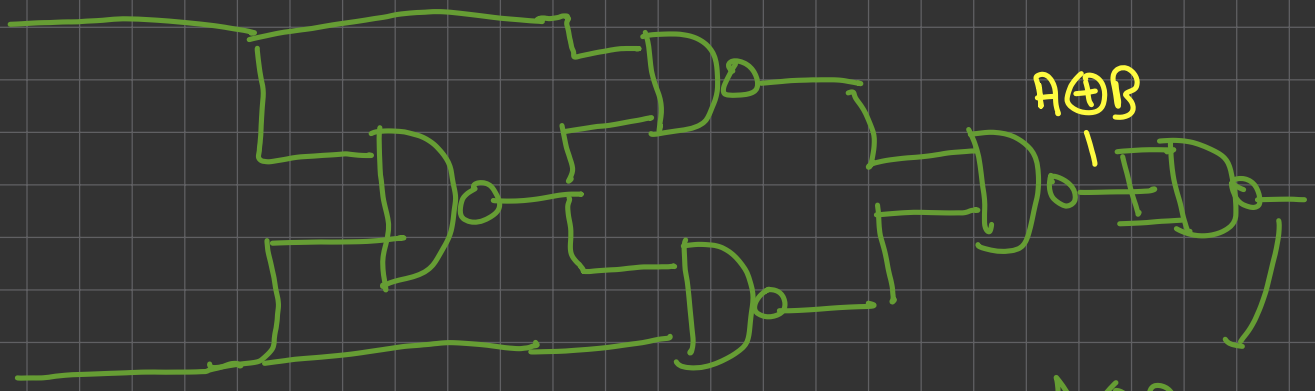
EXOR using NAND gates (Additional stuff)



$$A \cdot \overline{AB} = A[\bar{A} + \bar{B}]$$
$$= A\bar{A} + A\bar{B}$$
$$= A\bar{B}$$

$$\overline{A\bar{B}}\;\overline{\bar{A}B} \leftarrow A \oplus B$$
$$= \bar{A}\bar{\bar{B}} + \bar{\bar{A}}\bar{B}$$
$$= A\bar{B} + \bar{A}B$$
$$= A \oplus B$$

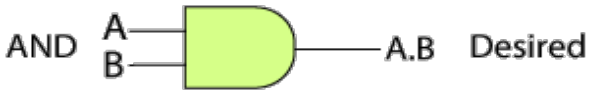
Ex NOR



$$A \oplus B = \overline{A\bar{B} + \bar{A}B} = A \odot B$$
$$= \overline{A\bar{B}}\;\overline{\bar{A}B}$$
$$= (\bar{A} + B)(A + \bar{B}) \Rightarrow \boxed{\bar{A}A + \bar{A}\bar{B} + AB + \bar{A}\bar{B}}$$

AND gate using NOR gate

AND Gate using NOR Gate



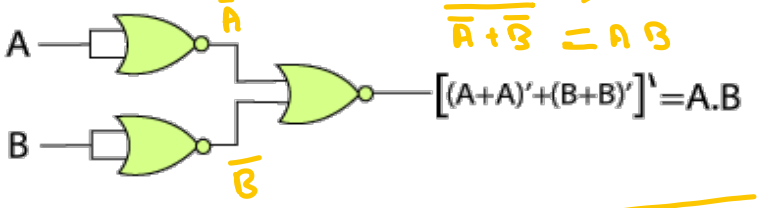
$A.B = [(A.B)']'$ Involution law

$= [A'+B]'$ Demorgan's law

$A.B = [(A+A)'+(B+B)']'$ Idempotency law

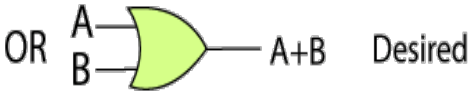
from De morgan

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



OR gate using NOR gate

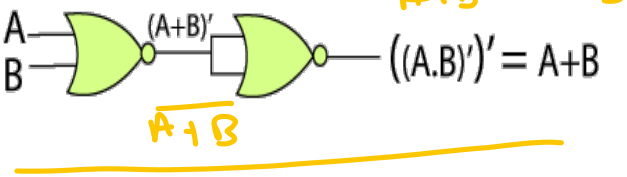
OR Gate using NOR Gate



2 OR gates

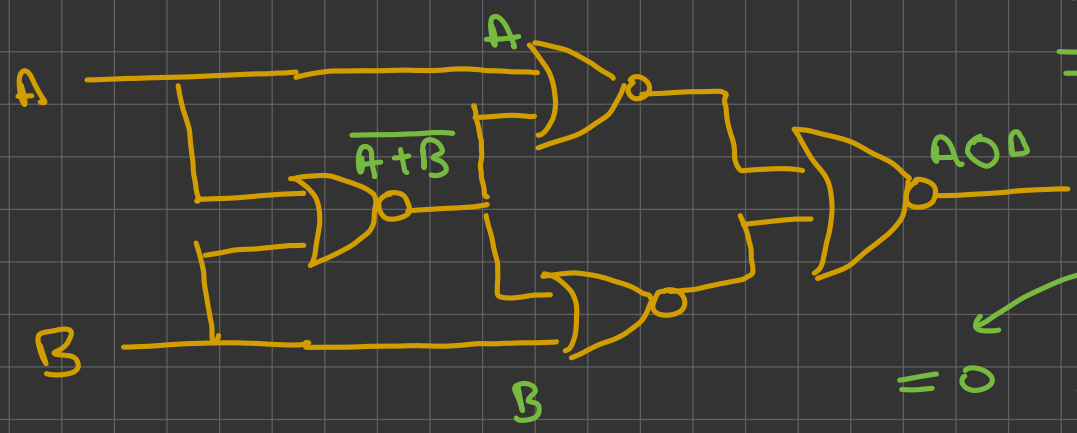
$A+B = ((A.B)')'$ Involution low

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



Implementation of logic gates using NOR gate

EXNOR



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

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

$= 0 + \bar{A}B$

$= \bar{A}B$

$O/P = \overline{\bar{A}B + A\bar{B}} = \overline{\bar{A}B}\;\overline{A\bar{B}}$

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

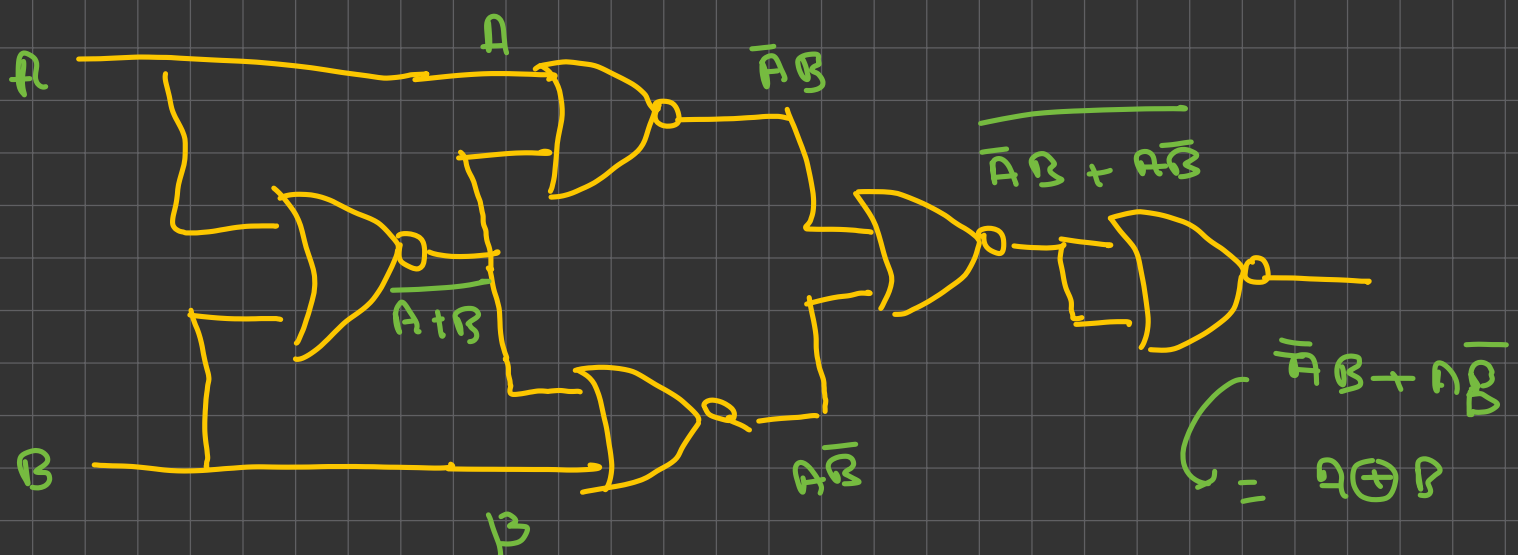
$= A\bar{A} + AB + \bar{A}\bar{B} + B\bar{B}$

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

$= A \odot B$

EXOR

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



5.3 Boolean Algebra – Laws, Identities, Minimization of Boolean Expressions.

Boolean algebra is a branch of Algebra (Mathematics) that deals with operations on logical values with Boolean variables, Boolean variables are represented as binary numbers which takes logic 1 and logic 0 values. Hence, the Boolean algebra is also called two-valued logic, Binary Algebra or Logical Algebra. The Boolean algebra was introduced by great mathematician George Boole in 1847. The Boolean algebra is a fundamental for the development of digital electronic systems, and is provided for in all programming languages. Set theory and statistics fields also uses Boolean algebra for the representation, simplification and analysis of mathematical quantities.

Rules in Boolean algebra

Only two values (1 for high and 0 for low) are possible for the variable used in Boolean algebra.

The overbar(-) is used for representing the complement variable. So, the complement of variable C is represented as C' .

The plus(+) operator is used to represent the ORing of the variables.

The dot(.) operator is used to represent the ANDing of the variables.

Properties of Boolean algebra

Annulment Law

When the variable is AND with 0, it will give the result 0, and when the variable is OR with 1, it will give the result 1, i.e.,

B.0 = 0

B+1 = 1

Identity Law

When the variable is AND with 1 and OR with 0, the variable remains the same, i.e.,

B.1 = B

B+0 = B

Idempotent Law

When the variable is AND and OR with itself, the variable remains same or unchanged, i.e.,

B.B = B

B+B = B

Complement Law

When the variable is AND and OR with its complement, it will give the result 0 and 1 respectively.

B.B' = 0

B+B' = 1

Double Negation Law

This law states that, when the variable comes with two negations, the symbol gets removed and the original variable is obtained.

((A)')' = A

Commutative Law

This law states that no matter in which order we use the variables.

It means that the order of variables doesn't matter in this law.

A.B = B.A

A+B = B+A

Associative Law

This law states that the operation can be performed in any order when the variables priority is of same as '*' and '/'.

(A.B).C = A.(B.C)

(A+B)+C = A+(B+C)

Distributive Law

This law allows us to open up of brackets.

Simply, we can open the brackets in the Boolean expressions.

A+(B.C) = (A+B).(A+C)

A.(B+C) = (A.B)+(A.C)

Absorption Law

This law allows us for absorbing the similar variables.

B+(B.A) = B

B.(B+A) = B

De Morgan Law

DeMorgan's Theorem is a powerful theorem in Boolean algebra which has a set of two rules or laws. These two laws were developed to show the relationship between two variable AND, OR, and NOT operations.

English mathematician Augustus DeMorganwho explained the NAND and NOR operations as NOT AND and NOT OR operations respectively. This explanation was named De Morgan's Theorem. These two rules enable the variables to be negated, i.e. opposite of their original form. Therefore, DeMorgan's theorem gives the dual of a logic function.

DeMorgan's First Theorem (Law 1)

DeMorgan's First Law states that the complement of a sum (ORing) of variables is equal to the product (ANDing) of their individual complements. In other words, the complement of two or more ORed variables is equivalent to the AND of the complements of each of the individual variables, i.e.

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

Or, it may also be represented as,

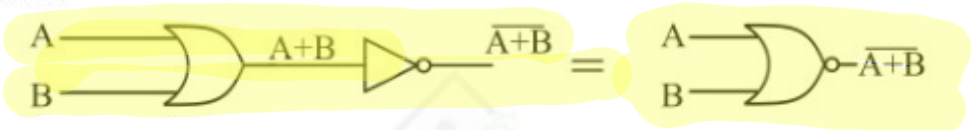
$$(A + B)' = A' \cdot B'$$

Proof of De Morgan's Law

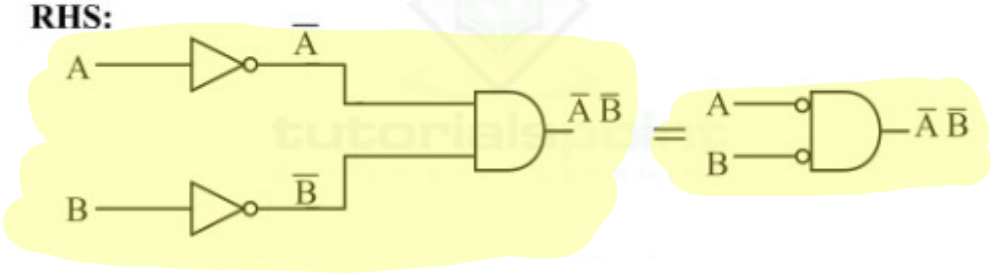
Left Side			Right Side		
A	**B**	**(A + B)'**	**A'**	**B'**	**A' · B'**
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Both output columns match in every row, confirming $(A+B)' \equiv A'\ B'$.

LHS:



RHS:



Thus we may conclude that the DeMorgan's First Law converts an expression from a sum form under a NOT sign to a product form.

DeMorgan's Second Theorem (Law 2)

DeMorgan's second law states that the complement of the product (ANDing) of variables is equivalent to the sum (ORing) of their individual complements.

In other words, the complement of two or more ANDed variables is equal to the sum of the complement of each of the individual variables, i.e.,

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

basically Complement of AND is equal to the individual Complement of OR

It may also be represented as,

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

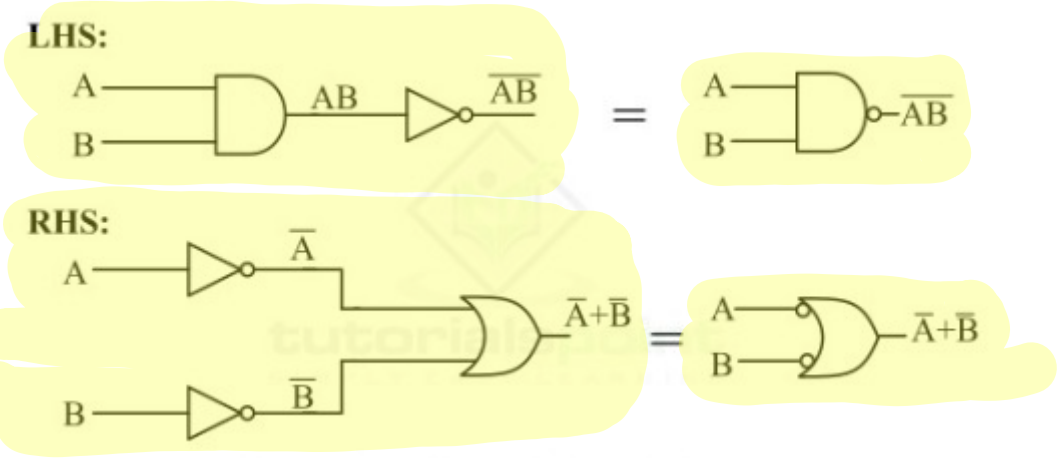
+ → OR
. → AND

Proof of DeMorgans law

Consider the truth table to prove this law

Left Side				Right Side		
A	**B**	**AB**	**(AB)'**	**A'**	**B'**	**A' + 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

from the thruth table,both outputs match, proving that (AB)′=A′+B′



DeMorgan's second law proves that the NAND gate is equivalent to a bubbled OR gate.

Simplify the following expressions using Boolean laws

1. $\mathbf{Y = BC + B\bar{C} + BA}$

solution:

$Y = BC + B\bar{C} + BA$

$Y = B(C + \bar{C}) + BA$

$Y = B + BA \qquad (\because C + \bar{C} = 1)$

$Y = B(1 + A) \qquad (\because 1 + A = 1)$

$Y = B.1$

$\mathbf{Y = B}$

2. $\mathbf{Y = A + \bar{A}B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}E}$

Solution:

$Y = A + \bar{A}B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}E$

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

$Y = A + (B + \bar{B}C + \bar{B}\bar{C}D + \bar{B}\bar{C}\bar{D}E) \qquad (\because A + \bar{A}B = A + B)$

$Y = A + (B + \bar{B}(C + \bar{C}D + \bar{C}\bar{D}E)) \qquad (\because A + \bar{A}B = A + B)$

$Y = A + (B + (C + \bar{C}D + \bar{C}\bar{D}E)) \qquad (\because A + \bar{A}B = A + B)$

$Y = A + \left(B + \left(C + \bar{C}(D + \bar{D}E)\right)\right) \qquad (\because A + \bar{A}B = A + B)$

$Y = A + (B + (C + (D + \bar{D}E))) \qquad (\because A + \bar{A}B = A + B)$

$\mathbf{Y = A + B + C + D + E}$

3. $\mathbf{Y = C + \bar{B}\bar{C}}$

Solution:

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

$Y = (C + \bar{C}) + \bar{B}$

$Y = 1 + \bar{B}$

$\mathbf{Y = 1}$

4. $\mathbf{Y = \bar{A}\bar{B}(\bar{A} + B)(\bar{B} + B)}$

Solution:

$Y = (\bar{A} + \bar{B})(\bar{A} + B)(1)$

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

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

$Y = \bar{A}(1)$

$\mathbf{Y = \bar{A}}$

5. $\mathbf{Y = (A + C)(AD + A\bar{D}) + AC + C}$

Solution:

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

$Y = (A + C)(A)(1) + AC + C$

$Y = AA + AC + AC + C$

$Y = A + AC + C$

$Y = A(1 + C) + C$

$\mathbf{Y = A + C}$

1. $Y = BC + B\bar{C} + BA$

solution:

To simplify by using boolean expression

$Y = B(C + \bar{C}) + BA$ $\quad$ $C + \bar{C} = 1$

$Y = B + BA$

$Y = B(1 + A)$ $\quad$ $1 + A = 1$ → from boolean algebra

$Y = B$

2. $Y = A + \bar{A}B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}E$

solution:

$Y = A + \bar{A}B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}E$

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

$A + \bar{A}B = A + B$ → Yaad rakhna

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

$Y = A + B + \bar{B}(C + \bar{C}D + \bar{C}\bar{D}E)$ } by using $A + \bar{A}B$ or in this case $B + \bar{B}A$

$Y = A + B + C + \bar{C}D + \bar{C}\bar{D}E$

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

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

$Y = A + B + C + D + E$

3. $Y = C + \overline{BC}$

from demorgan's 1st law we know that $\overline{A \cdot B} = \bar{A} + \bar{B}$

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

$Y = C + \bar{C} + \bar{B}$ $\quad C + \bar{C} = 1$

$Y = 1 + \bar{B}$ - - $1 + B$ or $1 + \bar{B} = 1$

$Y = 1$

4. $Y = \overline{AB}(\bar{A} + B)(\bar{B} + B)$

$Y = \bar{A}\bar{B}(\bar{A} + B)(1) \cdot (\bar{B} + B = 1)$

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

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

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

$Y = \bar{A}(1)$

$Y = \bar{A}$

It is infact very much possible to do this in the way you'd normally do maths

$\bar{B}\,B = 0$ (from complement law)

$\bar{A}\,\bar{A} = 0$

5. $Y = (A + C)(AD + A\bar{D}) + AC + C$

$Y = (A + C)A(D + \bar{D}) + C(A + 1)$

$Y = (A + C)A + C$

$Y = AA + AC + 1$

$Y = A + AC + C$

$Y = A(1) + C$

$y = A + C$

6. $\mathbf{Y = \overline{A}(A + B) + (B + AA)(A + \overline{B})}$

Solution:

$Y = \overline{A}A + \overline{A}B + BA + B\overline{B} + AAA + AA\overline{B}$

$Y = 0 + \overline{A}B + AB + 0 + A + A\overline{B}$

$Y = B(\overline{A} + A) + A + A\overline{B}$

$Y = B + A(1 + \overline{B})$

$\mathbf{Y = A + B}$

7. = + (+) + (+)

Solution:

Y = AB + AB + AC + BB + BC

Y = AB + AC + B + BC

Y = AB + AC + B(1 + C)

Y = AB + AC + B

Y = B(A + 1) + AC

= +

Practice Problems:

Using Boolean algebra techniques, simplify this expression:

1. AB + A(B + C) + B(B + C)
2. Simplify F = x'yz + x'yz' + xz.
3. Prove x'y'z' + x'yz' + xyz' = x'z' + yz'
4. AB'+AB+A'B
5. AB+A'C+BC
6. XY+XY'Z+X'Y
7. (AB)′+A
8. Simplify F=ABC'+AB'C'+A'BC+ABC
9. Prove: AB+AB'+A'B=A+B

Answers

1. B+AC
2.x'y+xz
4.A+B
5.AB+A'C
6. X+Y
7. 1
8.AC'+BC

6. $Y = \bar{A}(A + B) + (B + AA)(A + \bar{B})$

Solution:

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

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

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

$Y = B(\bar{A} + A) + A(1 + \bar{B})$

$Y = B(1) + A(1)$

$Y = A + B$

7. $Y = AB + A(B + C) + B(B + C)$

$Y = AB + AB + AC + B\bar{B} + BC$ ($B\bar{B} = 0$)

$Y = AB + AC + BC + B$

$Y = AB + AC + B(C + 1)$

$Y = AB + AC + B$

$Y = B(A + 1) + AC$

$Y = B + AC$

5.4 Sum of Product (SOP) and Product of Sum(POS), K-Map Simplification (up to 4 variables), realization of circuits using logic gates.

There are two ways in which we can put the Boolean function. These ways are the minterm canonical form and maxterm canonical form.

A Literal signifies the Boolean variables including their complements. Such as B is a boolean variable and its complements are ~B or B', which are the literals.

Complements are literals

Minterm

The product of all literals, either with complement or without complement, is known as minterm.

Example

The minterm for the Boolean variables A and B is:

A. B

A. $\bar{}$

$\bar{}$ .B

The complement variables $\bar{}$ and $\bar{}$ can also be written as A' and B' respectively. Thus, we can write the minterm as:

A.B'

A'.B

Minterm from values

Using variable values, we can write the minterms as:

If the variable value is 1, we will take the variable without its complement.

If the variable value is 0, take its complement.

Example

Let's assume that we have three Boolean variables A, B, and C having values

A=1

B=0

C=0

Now, we will take the complement of the variables B and C because these values are 0 and will take A without complement. So, the minterm will be:

Minterm=A.B'C'

Let's take another example in which we have two variables B and C having the value

B = 0

C = 1

Minterm=B'C

Shorthand notation for minterm

We know that, when Boolean variables are in the form of minterm, the variables will appear in the product. There are the following steps for getting the shorthand notation for minterm.

In the first step, we will write the term consisting of all the variables

Next, we will write 0 in place of all the complement variables such as $\bar{}$ or A'.

We will write 1 in place of all the non-complement variables such as A or b.

Now, we will find the decimal number of the binary formed from the above steps.

In the end, we will write the decimal number as a subscript of letter**m**(minterm). Let's take some example to understand the theory of shorthand notation

Example 1: Minterm = AB'

First, we will write the minterm:

Minterm = AB'

Now, we will write 0 in place of complement variable B'.

Minterm = A0

8 + + 0 ' 6 0 %

Minterm = 10

" ! ' " *9 :

The decimal point number of $(10)_2$ is 2.

So, the shorthand notation of AB' is

Example 2: Minterm = AB'C'

First, we will write the minterm:

Minterm = AB'C'

Now, we will write 0 in place of complement variables B' and C'.

Minterm = A00

We will write 1 in place of non-complement variable A.

Minterm = 100

The binary number of the minterm AB'C' is 100.

The decimal point number of $(100)_2$ is 4. So, the shorthand notation of AB'C' is **Minterm = m_4**

Maxterm

The sum of all literals, either with complement or without complement, is known as maxterm.

Example:

The maxterm for the Boolean variables A and B will be:
A+B

A+ $\bar{}$

$\bar{}$ +B

We know that the complement variables $\bar{}$ and $\bar{}$ can be written as A' and B' respectively. So, the above maxterm can be written as
A+B'
A'+B

Maxterm from values

Using the given variable values, we can write the maxterm as:

If the variable value is 1, then we will take the variable without a complement.
If the variable value is 0, take the complement of the variable.

Example
Let's assume that we have three Boolean variables A, B., and C having values

A=1
B=0
C=0
Now, we will take the complement of the variables B and C because these values are 0 and will take A without complement. So, the maxterm will be:
Maxterm=A+B'+C'

Let's take another example in which we have two variables B and C having the value
B = 0
C = 1
Maxterm=B'+C

Shorthand notation for maxterm

We know that, when Boolean variables are in the form of maxterm, the variables will appear in sum. The steps for the maxterm are same as minterm:

In the first step, we will write the term consisting of all the variables

Next, we will write 0 in place of all the complement variables such as $\bar{}$ or A'.

We will write 1 in place of all the non-complement variables such as A or b.

Now, we will find the decimal number of the binary formed from the above steps.

In the end, we will write the decimal number as a subscript of letter Here, M denotes maxterm.

Let's take some example to understand the theory of shorthand notation

Example 1: Maxterm = A+B'

First, we will write the minterm:
Maxterm = A+B'
Now, we will write 0 in place of complement variable B'.
We will write 1 in place of non-complement variable A.
The binary number of the maxterm A+B' is 10.
The decimal point number of $(10)_2$ is 2.

So, the shorthand notation of A+B' is

Maxterm = M_2

Example 2: Maxterm = A+B'+C'

First, we will write the maxterm:
Maxterm = A+B'+C'
Now, we will write 0 in place of complement variables B' and C'.
We will write 1 in place of non-complement variable A.
The binary number of the maxterm A+B'+C' is 100.
The decimal point number of $(100)_2$ is 4.
So, the maxterm of A+B'+C' is M_4.

Sum of product(SOP)

A canonical sum of products is a boolean expression that entirely consists of minterms. The Boolean function F is defined on two variables X and Y. The X and Y are the inputs of the boolean function F whose output is true when any one of the inputs is set to true. The truth table for Boolean expression F is as shown below. The complement of the variables is taken whose value is 0, and the variables whose value is 1 will remain the same.

Inputs		Output	Minterms
X	**Y**	**F = X + Y**	**M**
0	0	0	X'Y'
0	1	1	X'Y
1	0	1	XY'
1	1	1	XY

The minterm quite literally means to take the "product" of A & B or A, B & C & it can be taken with or without compliments.
The example for the minterm be $A \cdot B$, $A \cdot \bar{B}$, $\bar{A} \cdot B$
take for eg $A = 1, B = 0, C = 0$; its min term = $A \cdot B' \cdot C'$

Here's how to write the shorthand notation of the minterm :-
first of all from the alphabets convert to 0's & 1's this converted version is in binary form then convert it to the decimal form then $m_{decimal}$ voila!

Maxterm :- It's literally the sum of the I/Ps taken with or without the compliment.
for example. $A+B$, $A+\bar{B}$, $\bar{A}+B$
Shorthand notation for the maxterm :- It's literally the same as that of the minterm no change nothing
for eg:- max term of $A+B' = (10)_2 = 2$ maxterm = M_2

Sum of the products :-

It's table is nearly identical to that of the fucking OR operation in this the product of the terms (it's the minterms) which gets added hence the name as such.

Canonical form which essentially mean that all of the terms are present in the SOP / POS terms as in ABC & its compliments included.

Inputs		Output	Minterms
X	Y	F = X + Y	M
0	0	0	X'Y'
0	1	1	X'Y
1	0	1	XY'
1	1	1	XY

this is the perfect rep. of WTF I'm saying

In this case the SOP would be

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

Also in this SOP there's this procedure of converting things to shorthand notation
Eg :) $F = x'y + xy' + xy$
we need to find the shorthand notations of the minterms

$x'y = 01 = (01)_2 = m_1$

Now, we will add all the minterms for which the output is true to find the desired canonical SOP(Sum of Product) expression.

F=X' Y+XY'+XY

Converting Sum of Products (SOP) to shorthand notation

The process of converting SOP form to shorthand notation is the same as the process of finding shorthand notation for minterms. There are the following steps to find the shorthand notation of the given SOP expression.

Write the given SOP expression.

Find the shorthand notation of all the minterms.

Replace the minterms with their shorthand notations in the given expression.

Example: F = X'Y+XY'+XY

1. Firstly, we write the SOP expression:

F = X'Y+XY'+XY

2. Now, we find the shorthand notations of the minterms X'Y, XY', and XY.

$X'Y = (01)_2 = m_1$
$XY' = (10)_2 = m_2$
$XY = (11)_2 = m_3$

3. In the end, we replace all the minterms with their shorthand notations:

$F=m_1+m_2+m_3$

Converting shorthand notation to SOP expression

The process of converting shorthand notation to SOP is the reverse process of converting SOP expression to shorthand notation. Let's see an example to understand this conversion.

Example:

Let us assume that we have a boolean function F, which defined on two variables X and Y. The minterms for the function F are expressed as shorthand notation is as follows:

$F=\sum(1,2,3)$

Now, from this expression, we will find the SOP expression. The Boolean function F has two input variables X and Y and the output of F=1 for m1, m2, and m3, i.e., 1st , 2nd , and 3rd combinations. So,

$F=\sum(1,2,3)$
$F= m_1 + m_2 + m_3$
F= 01 + 10 + 11

Now, we replace zeros with either X' or Y' and ones with either X or Y. Simply, the complement variable is used when the variable value is 1 otherwise the non-complement variable is used.

F = ∑(1,2,3)
F=01+10+11
F= A'B + AB' + AB

Product of Sum (POS)

A canonical product of sum is a boolean expression that entirely consists of maxterms. The Boolean function F is defined on two variables X and Y. The X and Y are the inputs of the boolean function F whose output is true when only one of the inputs is set to true. The truth table for Boolean expression F is as follows. A column will be added for the maxterm in the above table. The complement of the variables is taken whose value is 0, and the variables whose value is 1 will remain the same.

Inputs		Output	Minterms
X	**Y**	**F**	**M**
0	0	0	X'+Y'
0	1	1	X'+Y
1	0	1	X+Y'
1	1	0	X+Y

Now, we will multiply all the minterms for which the output is false to find the desired canonical POS(Product of sum) expression.
F=(X'+Y').(X+Y)

Converting Product of Sum (POS) to shorthand notation

The process of converting POS form to shorthand notation is the same as the process of finding shorthand notation for maxterms. There are the following steps used to find the shorthand notation of the given POS expression.

Write the given POS expression.

Find the shorthand notation of all the maxterms.

Replace the minterms with their shorthand notations in the given expression.

Example: F = (X'+Y').(X+Y)

1. Firstly, we will write the POS expression:

F = (X'+Y').(X+Y)

2. Now, we will find the shorthand notations of the maxterms X'+Y' and X+Y.

$x\bar{y}' = 10 = (10)_2 = m_2$ $\therefore F = m_1 + m_2 + m_3$

$xy = 11 = (11)_2 = m_3$

Converting Shorthand notation to the SOP form.

It's essentially an reverse process to Converting that SOP → Shorthand form we've to assume 2 literals in this case x & y

Here the Shorthand is represented as follows $F = \Sigma(1,2,3)$

now write that exactly as would have written in the vice versa case.

$F = m_1, m_2, m_3$

$F = 01, 10, 11$

now replace this 0's & is with A & B $(A'B + AB' + AB)$

(*) Keep in mind the keeping the order L → R A,B is absolutely necessary.

Now to Convert POS to Shorthand notation format

The process is very similar to that of Converting POS to Shorthand formation

for example $F = (x'+y')(x+y)$

now find Shorthand notations of the maxterms $(x'+y')(x+y)$

$x'+y' = (00)_2 = M_0$

$x+y = (11)_2 = M_3$

now replace the min terms with short hand of their expression, $F = M_0 \cdot M_3$

Now to Convert Shorthand notations into POS formal.

This is the boolean expression & they been expressed with 2 terms x & y

$F = \pi(1,2,3)$

$F = \pi(M_1, M_2, M_3)$

$F = \pi(01, 10, 11)$

now to Convert this into A & B's) In this case we've taken 1 to be Complements

$F = (A+B') \cdot (A'+B) \cdot (A'+B')$

$X'+Y' = (00)_2 = M_0$
$X+Y = (11)_2 = M_3$

3. In the end, we will replace all the minterms with their shorthand notations:

$F=M_0.M_3$

Converting shorthand notation to POS expression

The process of converting shorthand notation to POS is the reverse process of converting POS expression to shorthand notation. Let's see an example to understand this conversion.

Example:

Let us assume that we have a boolean function F, defined on two variables X and Y. The maxterms for the function F are expressed as shorthand notation is as follows:

$F=\prod(1,2,3)$

Now, from this expression, we find the POS expression. The Boolean function F has two input variables X and Y and the output of F=0 for M1, M2, and M3, i.e., 1st , 2nd , and 3rd combinations. So,

$F=\prod(1,2,3)$
F= M1.M2.M3
F= 01.10.11

Next, we replace zeros with either X or Y and ones with either X' or Y'. Simply, if the value of the variable is 1, then we take the complement of that variable, and if the value of the variable is 0, then we take the variable "as is".

$F = \sum(1,2,3)$
F=01.10.11
F=(A+B').(A'+B).(A'+B')

Conversion between Canonical Forms

In our previous section, we learned about SOP(sum of product) and POS(product of sum) expressions and calculated POS and SOP forms for different Boolean functions. In this section, we will learn about how we can represent the POS form in the SOP form and SOP form in the POS form. For converting the canonical expressions, we have to change the symbols $\prod$, $\sum$.

These symbols are changed when we list out the index numbers of the equations. From the original form of the equation, these indices numbers are excluded. The SOP and POS forms of the boolean function are duals to each other.

There are the following steps using which we can easily convert the canonical forms of the equations:

Change the operational symbols used in the equation, such as $\sum$, $\prod$.

Comparison / Conversions of Canonical forms :-

Use the Duality's De-Morgan's principal to write the indexes of the terms that are not presented in the given form of an equation or the index numbers of the Boolean function.

Conversion of POS to SOP form

For getting the SOP form from the POS form, we have to change the symbol ∏ to ∑. After that, we write the numeric indexes of missing variables of the given Boolean function.

There are the following steps to convert the **POS function F = Π x, y, z (2, 3, 5) = x y' z' + x y' z + x y z' into SOP form:**

In the first step, we change the operational sign to Σ.

Next, we **find the missing indexes of the terms, 000, 110, 001, 100, and 111**.

Finally, we write the product form of the noted terms.

000 = x' * y' * z'

001 = x' * y' * z

100 = x * y' * z'

110 = x * y* z'

111 = x * y * z

So the SOP form is:

F = Σ x, y, z (0, 1, 4, 6, 7) = (x' * y' * z') + (x' * y' * z) + (x * y' * z') + (x * y* z') + (x * y * z)

Conversion of SOP form to POS form

For getting the POS form of the given SOP form expression, we will change the symbol ∏ to ∑. After that, we will write the numeric indexes of the variables which are missing in the boolean function.

There are the following steps used to convert the SOP function **F = ∑ x, y, z (0, 2, 3, 5, 7) = x' y' z' + z y' z' + x y' z + xyz' + xyz into POS:**

In the first step, we change the operational sign to ∏.

We find the missing indexes of the terms, 001, 110, and 100.

We write the sum form of the noted terms.

001 = (x + y + z)

100 = (x + y' + z')

110 = (x + y' + z')

So, the POS form is:

F = Π x, y, z (1, 4, 6) = (x + y + z) * (x + y' + z') * (x + y' + z')

Conversion of SOP form to standard SOP form or Canonical SOP form

For getting the standard SOP form of the given non-standard SOP form, we will add all the variables in each product term which do not have all the variables. By using the Boolean algebraic law, (x + x' = 0) and by following the below steps we can easily convert the normal SOP function into standard SOP form.

Multiply each non-standard product term by the sum of its missing variable and its complement.

Repeat step 1, until all resulting product terms contain all variables

For each missing variable in the function, the number of product terms doubles.

Example:

Convert the non standard SOP function F = AB + A C + B C

Sol:

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

So, the standard SOP form of non-standard form is F = A B C + A B C' + A B' C + A' B C

Conversion of POS form to standard POS form or Canonical POS form

For getting the standard POS form of the given non-standard POS form, we will add all the variables in each product term that do not have all the variables. By using the Boolean algebraic law **(x * x' = 0)** and by following the below steps, we can easily convert the normal POS function into a standard POS form.

By adding each non-standard sum term to the product of its missing variable and its complement, which results in 2 sum terms.

Applying Boolean algebraic law, x + y z = (x + y) * (x + z)

By repeating step 1, until all resulting sum terms contain all variables

By these three steps, we can convert the POS function into a standard POS function.

Example:

F = (p' + q + r) * (q' + r + s') * (p + q' + r' + s)

1. Term (p' + q + r)

As we can see that the variable s or s' is missing in this term. So we add s*s' = 1 in this term.

(p' + q + r + s*s') = (p' + q + r + s) * (p' + q + r + s')

2. Term (q' + r + s')

Similarly, we add p*p' = 1 in this term for getting the term containing all the variables.

(q' + r + s' + p*p') = (p + q' + r + s') * (p' + q' + r + s')

3. Term (q' + r + s')

Now, there is no need to add anything because all the variables are contained in this term.

So, the standard POS form equation of the function is

F = (p' + q + r + s)* (p' + q + r + s')* (p + q' + r + s')* (p' + q' + r + s') * (p + q' + r' + s)

Karnaugh Map(K-Map) method K-map (ompulsary [10M]

The **K-map** is a systematic way of simplifying Boolean expressions. With the help of the K-map method, we can find the simplest POS and SOP expression, which is known as the minimum expression. The K-map provides a cookbook for simplification.

Just like the truth table, a K-map contains all the possible values of input variables and their corresponding output values. However, in K-map, the values are stored in cells of the array. In each cell, a binary value of each input variable is stored.

The K-map method is used for expressions containing 2, 3, 4, and 5 variables. For a higher number of variables, there is another method used for simplification called the Quine-McClusky method.

In K-map, the number of cells is similar to the total number of variable input combinations. For example, if the number of variables is three, the number of cells is 2^3=8, and if the number of variables is four, the number of cells is 2^4. The K-map takes the SOP and POS forms. The K-map grid is filled using 0's and 1's. The K-map is solved by making groups.

There are the following steps used to solve the expressions using K-map:

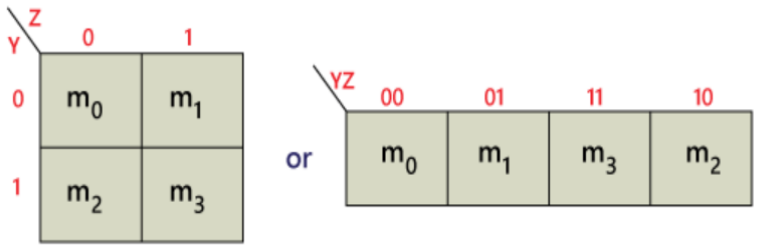
First, we find the K-map as per the number of variables. Find the maxterm and minterm in the given expression. Fill cells of K-map for SOP with 1 respective to the minterms. Fill cells of the block for POS with 0 respective to the maxterm.

Next, we create rectangular groups that contain total terms in the power of two like 2, 4, 8, … and try to cover as many elements as we can in one group.

With the help of these groups, we find the product terms and sum them up for the SOP form.

2 Variable K-map

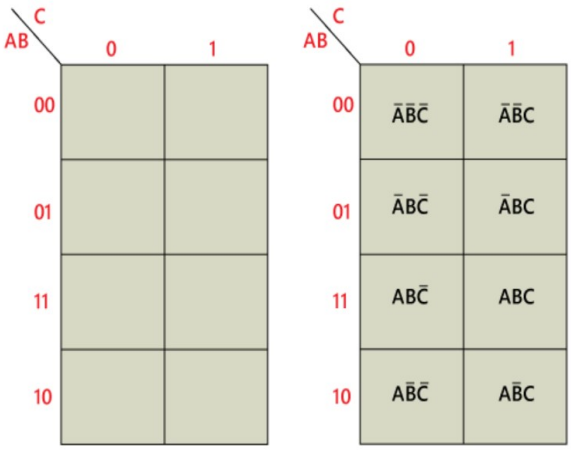
There is a total of 4 variables in a 2-variable K-map. There are two variables in the 2-variable K-map. The following figure shows the structure of the 2-variable K-map:



- In the above figure, there is only one possibility of grouping four adjacent minterms.
- The possible combinations of grouping 2 adjacent minterms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3-variable K-map

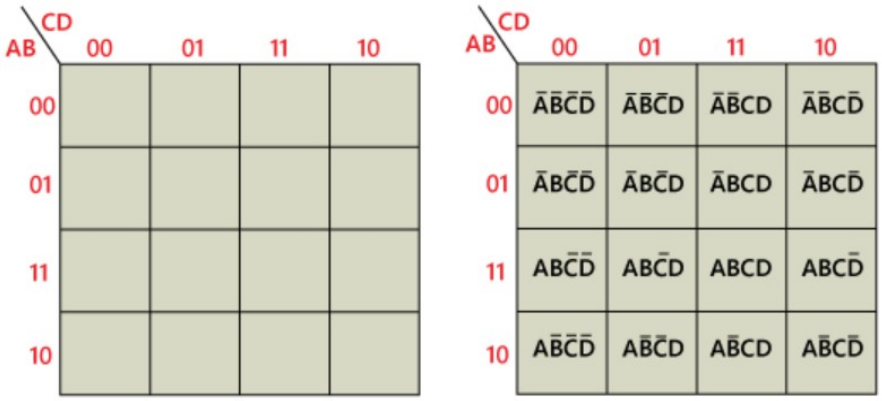
The 3-variable K-map is represented as an array of eight cells. In this case, we used A, B, and C for the variable. We can use any letter for the names of the variables. The binary values of variables A and B are along the left side, and the values of C are across the top. The value of the given cell is the binary values of A and B at left side in the same row combined with the value of C at the top in the same column. For example, the cell in the upper left corner has a binary value of 000, and the cell in the lower right corner has a binary value of 101.



The 4-Variable Karnaugh Map

The 4-variable K-map is represented as an array of 16 cells. Binary values of A and B are along the left side, and the values of C and D are across the top. The value of the given cell is the binary values of A and B at left side in the same row combined with the binary values of C and D at the top

in the same column. For example, the cell in the upper right corner has a binary value of 0010, and the cell in the lower right corner has a binary value of 1010



Simplification of boolean expressions using Karnaugh Map

As we know that K-map takes both SOP and POS forms. So, there are two possible solutions for K-map, i.e., minterm and maxterm solution. Let's start and learn about how we can find the minterm and maxterm solution of K-map.

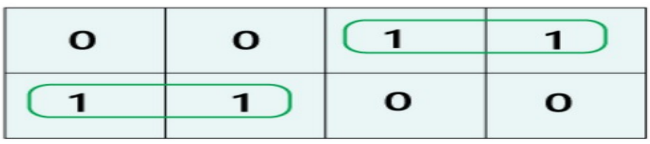
Minterm Solution of K Map

There are the following steps to find the minterm solution or K-map:

Step 1: Firstly, we define the given expression in its canonical form.

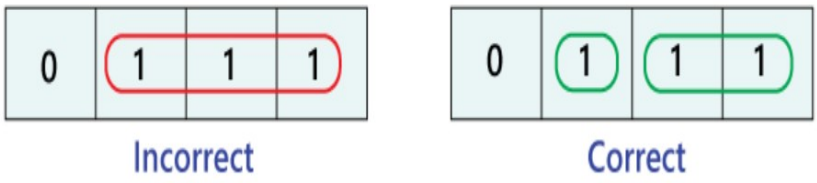
Step 2: Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.

Step 3: Next, we form the groups by considering each one in the K-map.

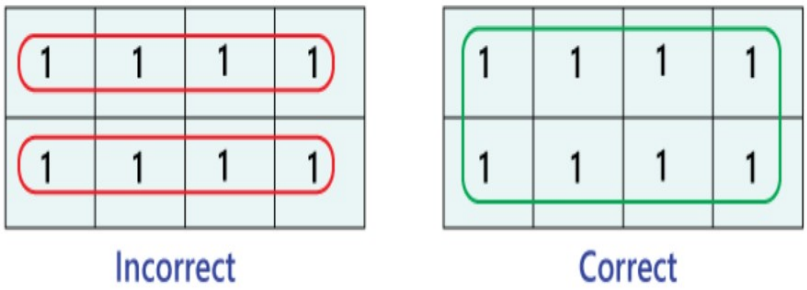


Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.

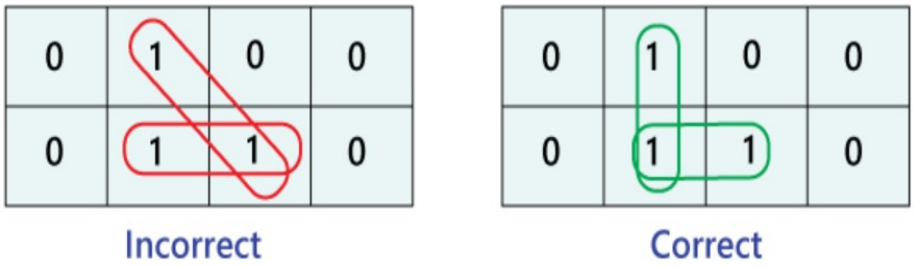
In a group, there is a total of 2^n number of ones. Here, n=0, 1, 2, …n.



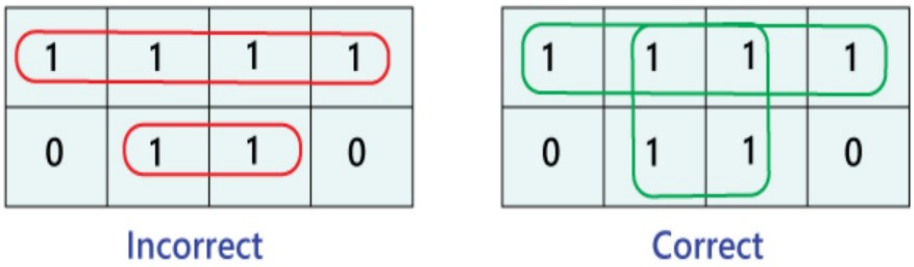
We group the number of ones in the decreasing order. First, we have to try to make the group of eight, then for four, after that two and lastly for 1.



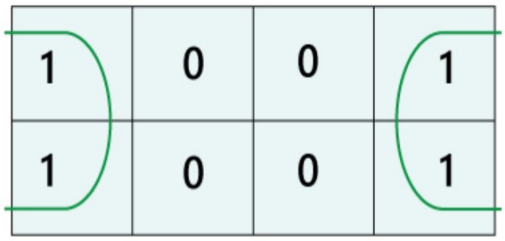
In horizontally or vertically manner, the groups of ones are formed in shape of rectangle and square. We cannot perform the diagonal grouping in K-map.



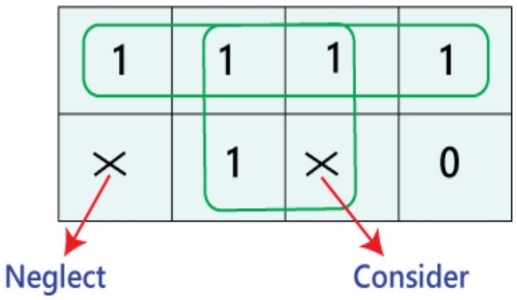
The elements in one group can also be used in different groups only when the size of the group is increased.



The elements located at the edges of the table are considered to be adjacent. So, we can group these elements.

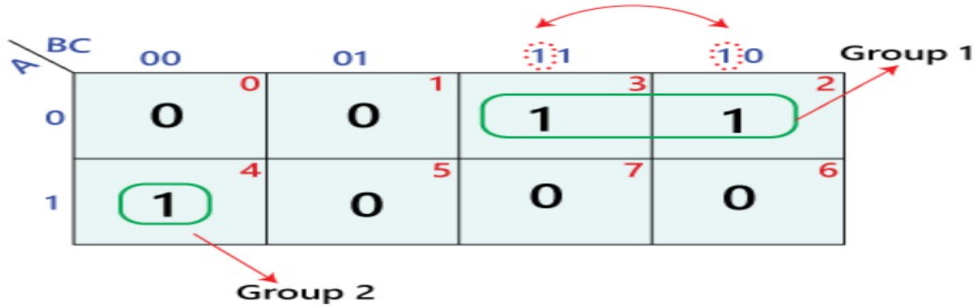


We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.



Step 4: In the next step, we find the boolean expression for each group. By looking at the common variables in cell-labeling, we define the groups in terms of input variables. In the below example, there is a total of two groups, i.e., group 1 and group 2, with two and one number of 'ones'.

In the first group, the ones are present in the row for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In these columns, only B term in common is the product term corresponding to the group as A'B. Just like group 1, in group 2, the one's are present in a row for which the value of A is 1. So, the corresponding variables of this column are B'C'. The overall product term of this group is AB'C'.

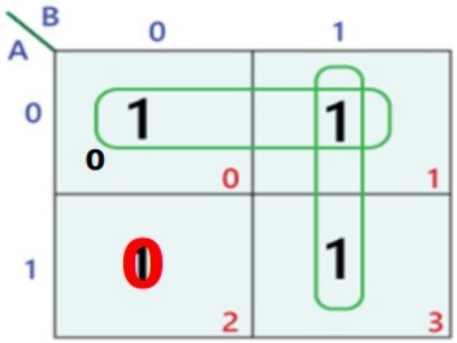


Step 5: Lastly, we find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, we combine the product-terms of all individual groups. So the simplified expression of the above k-map is as follows:

A’B+AB'C'

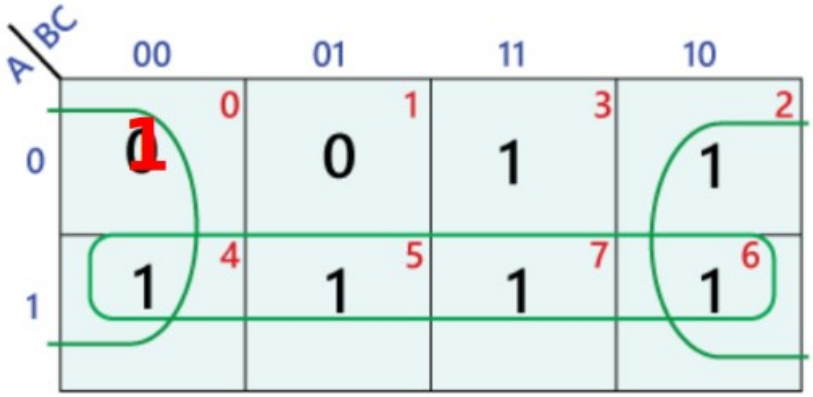
Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.

Example 1: Y=A'B' + A'B+AB



Simplified expression: Y=A'+B

Example 2: Solve using K map, Y=A'B'C'+A' BC'+AB' C'+AB' C+ABC'+ABC

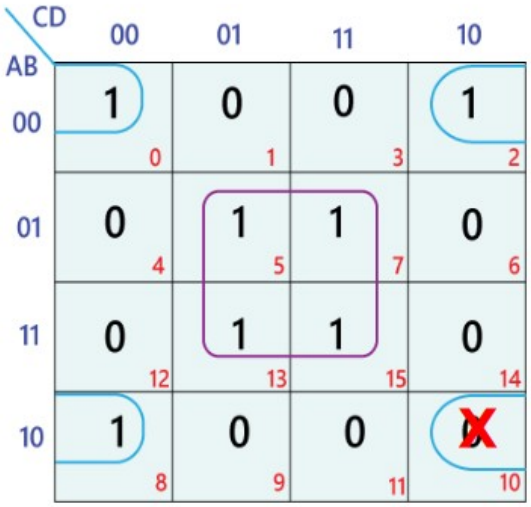


Simplified expression: Y=A+C'

Example 3: Solve Y=A'B'C'D'+A'B'CD'+A'BC'D+A'BCD+AB' C' D'+ABC'D+ABCD using K map

(0, 2, 5, 7, 8, 13,15)

Solution



Maxterm Solution of K-Map

To find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:

We will populate the K-map by entering the value of 0 to each sum-term into the K-map cell and fill the remaining cells with one's.

We will make the groups of 'zeros' not for 'ones'.

Now, we will define the boolean expressions for each group as sum-terms.

At last, to find the simplified boolean expression in the POS form, we will combine the sum-terms of all individual groups.

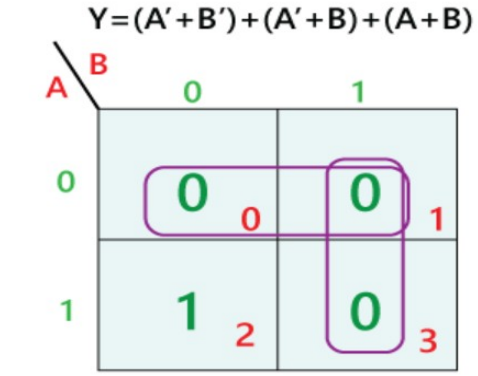
Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examples

Example 1: Solve using max term Y=(A'+B')+(A+B')+(A+B)

Solution : (A'+B') 11 3

(A+B') 01 1

(A+B) 00 0



Simplified expression: A'B

Example 2: Solve using maxterm

Y=(A + B + C') + (A + B' + C') + (A' + B' + C) + (A' + B' + C')

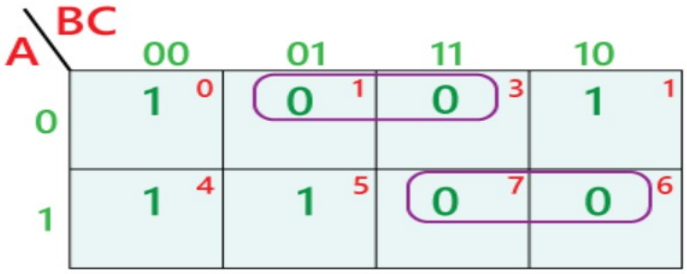
(A + B + C') 001 1

(A + B' + C') 011 3

(A' + B' + C) 110 6

(A' + B' + C') 111 7

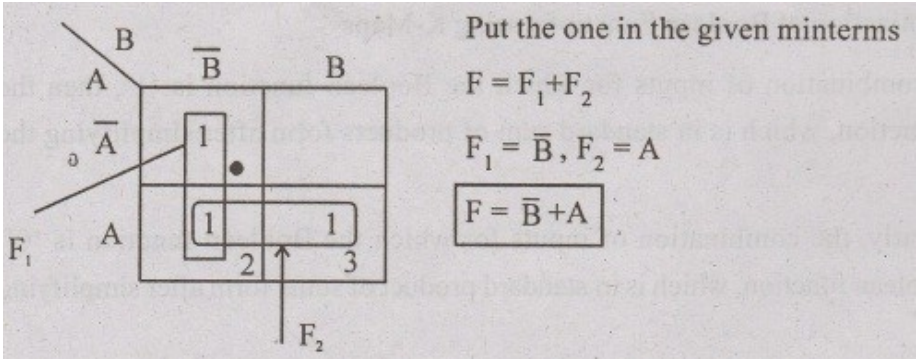
Solution:



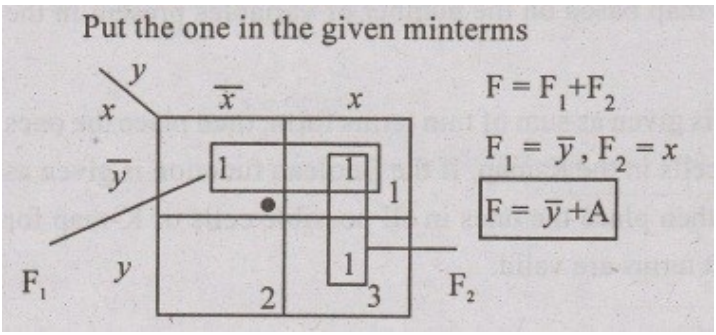
Simplified expression: Y=(A + C') .(A' + B')

More practice Problem using K map

1. Reduce the function using K-map F(A,b) = Σ(0,2,3)

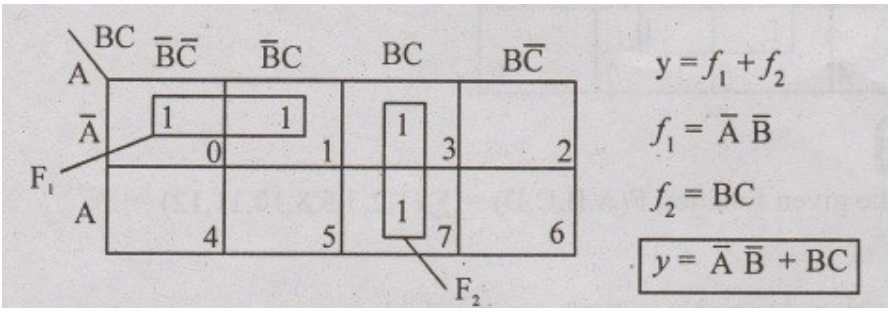


2. Simplify the function using Karnaugh Map F(x,y)= Σ(0,1,3)

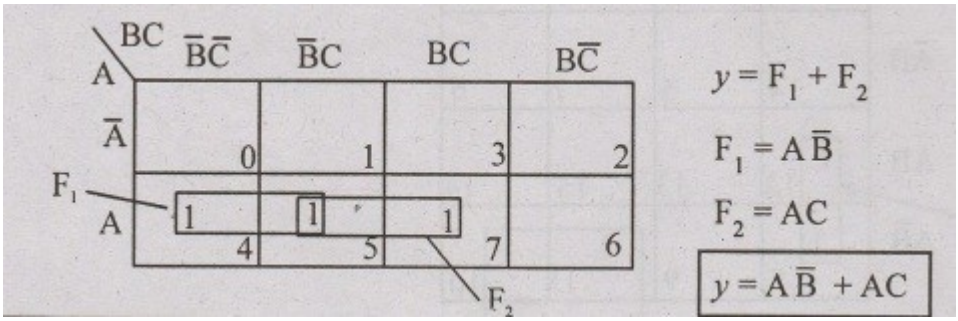


3. Reduce the function y(A,B,C) = Σ (0,1,3,6,7)

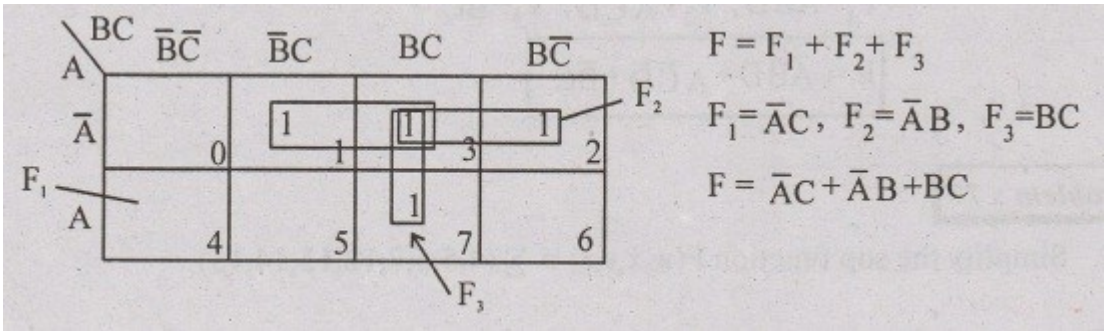
Here the 3 variable as in the function so we draw 3 variable k-map and fill ones in the given minterms



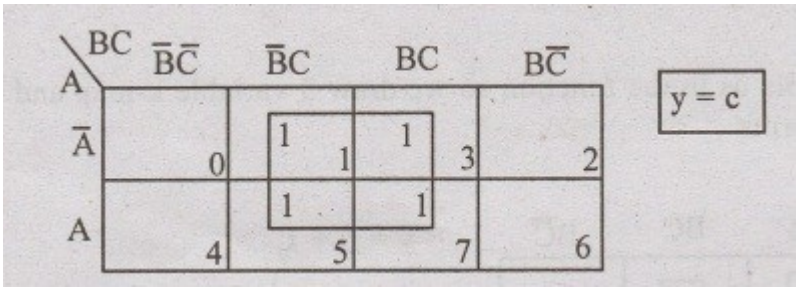
4. **Reduce the given function y = Σ (4,5,7)**



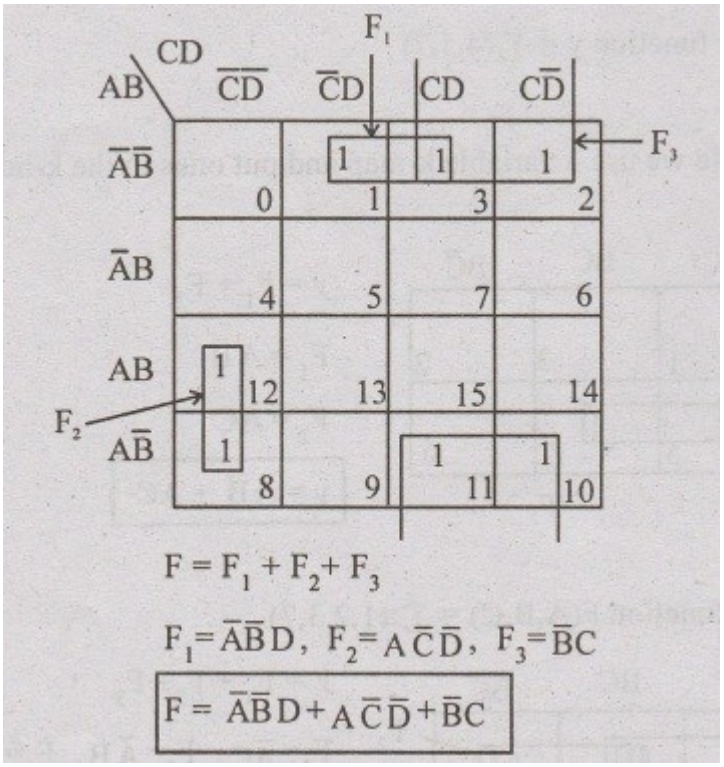
5. Simplify the sop function F(A,B,C) = Σ (1,2,3,7)



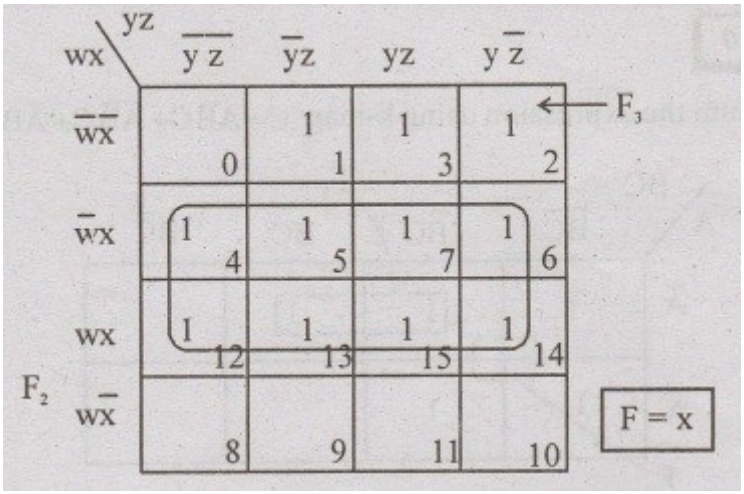
6. Simplify the sop function F(A,B,C) = Σ (1,3,5,7)



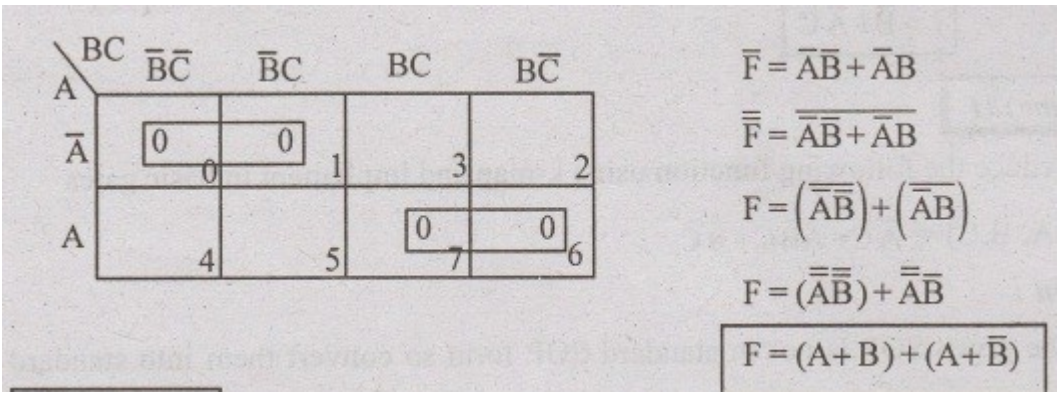
7. Reduce the given function F(A,B,C,D) = Σ (1,2,3,5,8,10,11,12)



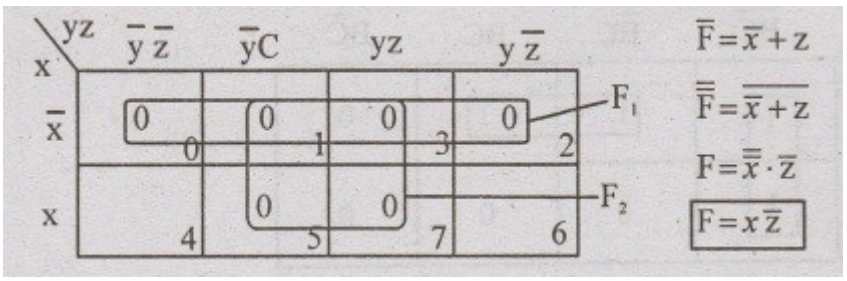
8. Simplify the sop function F(w,x,y,z) = Σ (4,5,6,7,12,13,14,15)



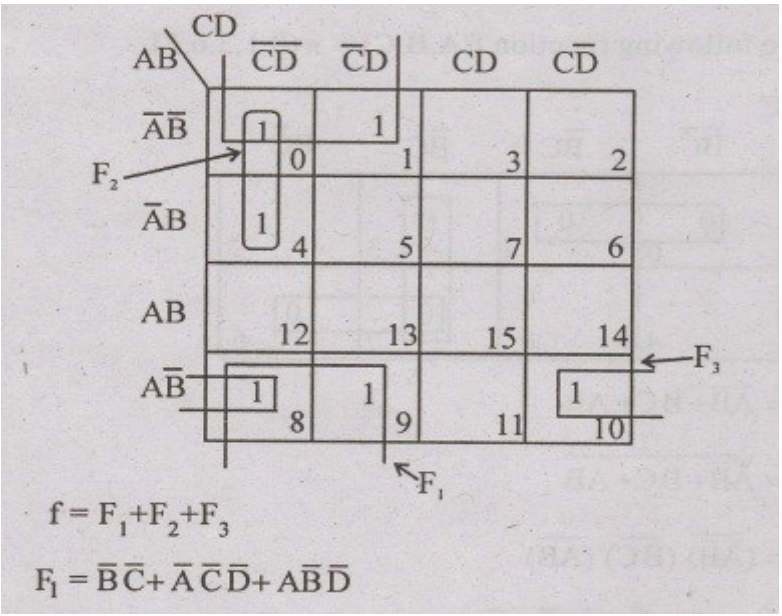
9. Simplify the sop function F(A,B,C) = π (0,1,6,7)



10. Simplify the sop function F(x,y,z) = π (0,1,2,3,5,7)



11. Reduce the following using karnaugh map f (A, B, C, D) = Σm (0,1,4,8,9,10)



Dont Care Condition

In the K-map method, there is a useful condition namely, Dont Care Condition, which helps in simplifying a Boolean function. The dont care condition makes the grouping of variables in K-map easy.

Sometimes, in a Boolean expression for certain input combinations, the value of the output is not specified either due to invalid input combinations or due to the precise value of output is of no importance. These input combinations for which the values of the Boolean function are not specified are called dont care combinations. Dont Care Combinations are also referred to as optional combinations. In K-map, dont care combinations are represented by "X" or "d" or "".

For example, in 8421 binary code, the binary combination 1010, 1011, 1100, 1101, 1110, and 1111 are the invalid terms and their corresponding outputs are dont care combinations. Similarly, in Excess-3 code, the combinations 0000, 0001, 0010, 1101, 1110, and 1111 do not occur, hence these are called dont care combinations.

When we deal with SOP (Sum of Products) K-map, each dont care term is treated as a 1, if it helps in reduction of the expression, otherwise it is considered as a 0 and left alone.On the other hand, when we are using POS (Product of Sums) K-map, each dont care term is considered as a 0, if it is helpful in reduction of the expression, otherwise it is treated as a 1 and left alone.

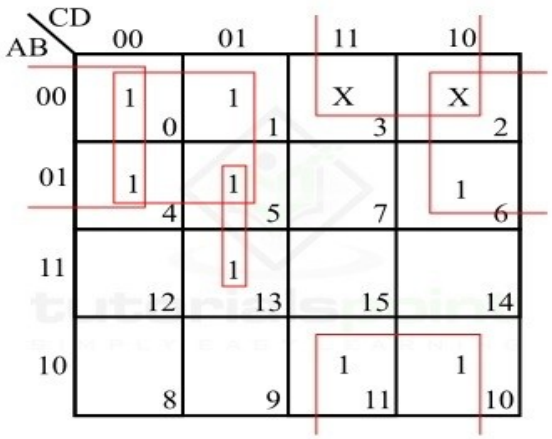
Also, we can convert a Standard Sum of Products (SSOP) expression with dont care terms into a Standard Product of Sums (SPOS) expression by keeping the dont care terms as they are, and writing the missing

minterms of the SSOP form as the maxterms of the SPOS form.Similarly, we can convert a standard product of sums (SPOS) expression with dont care terms into a standard sum of products (SSOP) expression by keeping the dont care terms of the SPOS expression as they are, and writing the missing maxterms of the SPOS expression as the minterms of the SSOP expression.

Example: Minimize the following 4-variable Boolean expression in SOP form using K-map.

$$f\ A,B,C,D = \sum m\ 0,1,4,5,6,10,13 + d\ 2,3$$

Solution: The SOP K-map representation of the given Boolean function is shown in Figure



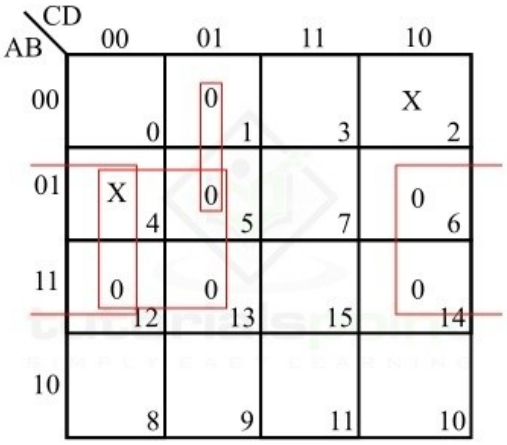
Therefore, the minimal Boolean expression is,

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

Example: Minimize the following 4-variable Boolean expression in POS form using K map.

$$f\ A,B,C,D = \prod M\ 1,5,6,12,13,14 + d\ 2,4$$

Solution: The POS K-map representation of the given Boolean function is shown in Figure



Therefore, the minimal Boolean expression is,

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