UNIT 2

BOOLEAN ALGEBRA

Boolean Algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as Binary Algebra or logical Algebra. Boolean algebra was invented by George Boole in 1854.

The most obvious way to simplify Boolean expressions is to manipulate them in the same way as normal algebraic expressions are manipulated. With regards to logic relations in digital forms, a set of rules for symbolic manipulation is needed in order to solve for the unknowns.

A set of rules formulated by t he English mathematician George Boole describe certain propositions whose outcome would be either true or false. With regard to digital logic, these rules are used to describe circuits whose state can be either, 1 (true) or 0 (false).

Basic Theorems And Properties Of Boolean **Algebra**

Laws of Boolean Algebra

There are six types of Boolean algebra laws. They are:

1. Commutative law
2. Associative law
3. Distributive law
4. AND law
5. OR law
6. Inversion law

1.Commutative Law

Any binary operation which satisfies the following expression is referred to as a commutative operation. Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

$A. B = B. A$

$A + B = B + A$

2.Associative Law

It states that the order in which the logic operations are performed is irrelevant as their effect is the same.

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

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

3.Distributive Law

Distributive law states the following conditions:

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

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

4.AND Law

These laws use the AND operation. Therefore they are called AND laws.

A .0 = 0

A . 1 = A

A. A = A

$A.\bar{A}=0$

5.OR Law

These laws use the OR operation. Therefore they are called OR laws.

A + 0 = A

A + 1 = 1

A + A = A

$A+\bar{A}=1$

6.Inversion Law

This law uses the NOT operation. The inversion law states that double inversion of variable results in the original variable itself.

$A+\bar{\bar{A}}=1$

DUALITY PRINCIPLE

According to this principle, if we have postulates or theorems of Boolean Algebra for one type of operation then that operation can be converted into another type of operation (i.e., **AND can be converted to OR and vice-versa**) just by interchanging **'0 with 1'**, **'1 with 0'**, **'(+) sign with (.) sign'** and **'(.) sign with (+) sign'**. This principle ensures if a theorem is proved using postulates of Boolean algebra, then the dual of this theorem automatically holds and we need not prove it again separately.

e.g. The dual of 1.01.0 which is 0+10+1,

is obtained by interchanging AND(.) to OR(+) and 1's to 0's and vice versa.

DE-MORGAN'S THEOREM

De-Morgan's theorem can be stated as follows:-

Theorem 1:

The compliment of the product of two variables is equal to the sum of the compliment of each variable. Thus according to **De-Morgan's laws** or De-Morgan's theorem if A and B are the two variables or Boolean numbers. Then accordingly

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

Theorem 2:

The compliment of the sum of two variables is equal to the product of the compliment of each variable.Thus according to De Morgan's theorem if A and B are the two variables then.

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

Simplification of Boolean Expression

1. X+XY

= X(1+Y)

= X

2. X+X'Y

= (X+X') (X+Y)

= 1. (X+Y)

= X+Y

3. X(X'+Y)

= XX'+XY

= 0 + XY

= XY

4. X'Y'Z+X'YZ+XY'

= X'Z(Y+Y')

= X'Z

5. C + (BC)'

= C (B' + C') DeMorgan's Law.

= (C + C') + B Commutative, Associative Laws.

= 1+ B Complement Law.

= 1 Identity Law.

6. $A(A+\overline{A})+B = AA+A\overline{A}+B$ by the distributive law

$= A+0+B$ because $AA = A$ and $A\overline{A} = 0$

$= A+B$ because $A + 0 = A$

7. $(A+B)(\overline{A}+B)\overline{B} = (A+B)(\overline{A}\ \overline{B}+B\overline{B})$ by the distributive law

$= (A+B)(\overline{A}\ \overline{B}+0)$ because $B\overline{B} = 0$

$= (A+B)(\overline{A}\ \overline{B})$ because $\overline{A}\ \overline{B}+0 = \overline{A}\ \overline{B}$

$= A\overline{A}\ \overline{B}+B\overline{A}\ \overline{B}$ by the distributive law

$= A0 + \overline{A}\,0$ because $A\overline{A} = 0$ and $B\overline{B} = 0$

$= 0$ because 0 ANDed with anything is 0

8. AB + A(B + C) + B(B + C)

=AB + AB + AC + BB + BC

=AB + AB + AC + B + BC

=AB + AC + B + BC

=AB + AC + B

=B+AC

9. Simplify: C + (BC)'

=C + (B' + C') DeMorgan's Law.

=(C + C') + B Commutative, Associative Laws.

=B

10. AB(A + B)(B + B)

=AB(A + B) Complement law, Identity law.

=(A + B)(A + B) DeMorgan's Law

=A + BB Distributive law.

=A Complement, Identity.

11. Simplify: (A + C)(AD + AD) + AC + C:

=(A + C)A(D + D) + AC + C	Distributive.
=(A + C)A + AC + C	Complement, Identity.
=A((A + C) + C) + C	Commutative, Distributive.
=A(A + C) + C	Associative, Idempotent.
=AA + AC + C	Distributive.
=A + (A + 1)C	Idempotent, Identity, Distributive.
=A + C	Identity, twice.

12. (AB)‘(A‘ + B)(B‘ + B)

= (AB)’(A’ + B)	Complement law, Identity law.
= (A’ + B’)(A’ + B)	DeMorgan's Law
= A’A’ + A’B + A’B’ + BB’	
= A’ + A’(B+B’) + BB’	
= A’ + A’.1 + BB’	
= A’ + BB’	Distributive law.
= A’	Complement, Identity.

BOOLEAN FUNCTIONS:

The binary variables and logic operations are used in Boolean algebra. The algebraic expression is known as Boolean Expression, is used to describe the Boolean Function. The Boolean expression consists of the constant value 1 and 0, logical operation symbols, and binary variables.

Example 1: F= xy' z+p

We defined the Boolean function F=xy' z+p in terms of four binary variables x, y, z, and p. This function will be equal to 1 when x=1, y=0, z=1 or p=1.

Example 2: F= A+B′C′+ACD

We defined the Boolean function F= A+B′C′+ACD in terms of four binary variables A,B,C and D. This function will be equal to 1 when A=1 or B=C=0 or A=C=D=0.

TRUTH TABLE

A truth table represents a table having all combinations of inputs and their corresponding result. It is possible to convert the switching equation into a truth table. For example, consider the following switching equation.

$$F(A, B, C) = A + BC$$

The output will be high (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is shown by table below. The number of rows in the truth table is 2^n where n is the number of input variables (n=3 for the given equation). Hence there are $2^3 = 8$ possible input combination of inputs.

Inputs			Output
A	**B**	**C**	**F**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table for the Boolean function F(A,B,C,D)=A+BC'+D

The output will be high when A=1 or BC'=1 or D=1 or all are set to 1. The truth table of the above example is given below. The 2^n is the number of rows in the truth table. The n defines the number of input variables. So the possible input combinations are 2^3=8.

Inputs				Output
A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

CANONICAL AND STANDARD FORM

Canonical Form – In Boolean algebra, Boolean function can be expressed as Canonical Disjunctive Normal Form known as **minterm** and some are expressed as Canonical Conjunctive Normal Form known as **maxterm**.

In Minterm, we look for the functions where the output results in "1" while in Maxterm we look for function where the output results in "0".

We perform **Sum of minterm** also known as Sum of products (SOP) and **Product of Maxterm** also known as Product of sum (POS).

Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

Standard Form – A Boolean variable can be expressed in either true form or complemented form. **In standard form, Boolean function will contain all the variables in either true form or complemented form while in canonical number of variables depends on the output of SOP or POS.**

A Boolean function can be expressed algebraically from a given truth table by forming a:

- minterm for each combination of the variables that produces a 1 in the function and then taking the AND of all those terms.
- maxterm for each combination of the variables that produces a 0 in the function and then taking the OR of all those terms.

MINTERM AND MAXTERM

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

Literal

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.

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:

1. A.B
2. A.~B or A.B′
3. ~A.B or A′.B
4. ~A. ~B or A′.B′

Minterm from values

Using variable values, we can write the minterms as:

1. If the variable value is 1, we will take the variable without its complement.
2. If the variable value is 0, take its complement.

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 ~A 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
- We will write 1 in place of non-complement variable A.
  Minterm = 10
- The binary number of the minterm AB' is 10. The decimal point number of $(10)_2$ is 2. So, the shorthand notation of AB' is
  Minterm = m_2

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:

1. A+B
2. A+~B or A+B'
3. ~A+B or A'+B
4. ~A+~B or A'+B’

Maxterm from values

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

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

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 1 in place of all the complement variables such as ~A or A'.
- We will write 0 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 1 in place of complement variable B'.
- We will write 0 in place of non-complement variable A.
- The binary number of the maxterm A+B' is 10. The decimal point number of $(01)_2$ is 1. So, the shorthand notation of A+B' is
  Maxterm = M_1

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

- First, we will write the maxterm:
  Maxterm = A+B'+C'

- Now, we will write 1 in place of complement variables B' and C'.
- We will write 0 in place of non-complement variable A.
- The binary number of the maxterm A+B'+C' is 100. The decimal point number of $(011)_2$ is 3. So, the maxterm of A+B'+C' is m_3.

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Example 1. Express the Boolean function F = x + y z as a sum of minterms.

Solution: F = x + y z = x + (y z)

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

=x(yz+yz'+y'z+y'z')+xyz+x'yz

= x y z + x y z' + x y' z + x y' z' + x y z + x' y z

= m7 + m6 + m5 + m4 + m3

= Σ(3, 4, 5, 6, 7)

Example 2. Express the Boolean function F = x + y z as a product of maxterms.

Solution: F = x + y z = x + (y z)

= (x + y) (x + z) use distributive law to change to product of OR terms

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

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

= M0 • M1 • M2

= Π(0, 1, 2)

Example 3. Express F ' = (x + y z)' as a sum of minterms.

Solution: F' = (x + y z)' = (x + (y z))'

= x' (y' + z')

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

= x' y' z + x' y' z' + x' y z' + x' y' z'

= m1 + m0 + m2

= Σ(0, 1, 2)

Example 4. Express F ' = (x + y z)' as a product of maxterms.

Solution: F' = (x + y z)' = (x + (y z))'

= x' (y' + z')

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

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

= M4 • M6 • M5 • M7 • M3

= ∏(3,4,5,6,7)

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 follows:

Inputs		**Output**
X	**Y**	**F**
0	0	0
0	1	1
1	0	1
1	1	1

In our previous section, we learned about how we can form the minterm from the variable's value. Now, a column will be added for the minterm 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	Minterm
X	**Y**	**F**	**M**
0	0	0	X'Y'
0	1	1	X'Y
1	0	1	XY'
1	1	1	XY

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: Converting Sum of Products (SOP) to shorthand notation **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=m1+m2+m3

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

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:

Inputs		Output
X	**Y**	**F**
0	0	0
0	1	1
1	0	1
1	1	0

In our minterm and maxterm section, we learned about how we can form the maxterm from the variable's value. A column will be added for the maxterm in the above table. The complement of the variables is taken whose value is 1, and the ordinary variables whose value is 0.

Inputs		Output	Maxterm
X	**Y**	**F**	**M**
0	0	0	X+Y
0	1	1	X+Y’
1	0	1	X’+Y
1	1	1	X’+Y’

Now, we will multiply all the maxterms 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.

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

$X+Y = (00)_2 = M_0$

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

$F=M_0.M_3$

$F=\prod(0,3)$

CONVERSION BETWEEN CANONICAL FORMS

For converting the canonical expressions, we have to change the symbols ∏, ∑. 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:

1. Change the operational symbols used in the equation, such as ∑, ∏.
2. 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:

1. In the first step, we change the operational sign to Σ.
2. Next, we find the missing indexes of the terms, 000, 001, 100, 110 and 111.
3. 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' + x' y z' + x' y z + xy'z + xyz into POS:

- In the first step, we change the operational sign to ∏.
- We find the missing indexes of the terms, 001, 100, and 110.
- 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 to standard SOP

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

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 roadmap for simplification.

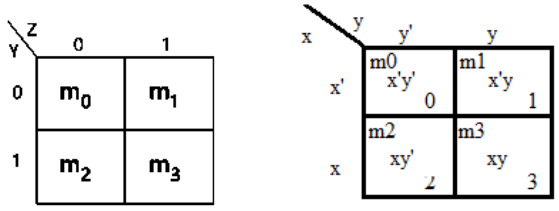
Just like the truth table, a K-map contains all the possible values of input variables and their corresponding output values. However, in K-map, the values are stored in cells of the array. In each cell, a binary value of each input variable is stored.

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:

1. First, we find the K-map as per the number of variables.
2. Find the maxterm and minterm in the given expression.
3. Fill cells of K-map for SOP with 1 respective to the minterms.
4. Fill cells of the block for POS with 0 respective to the maxterm.
5. 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.
6. 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:

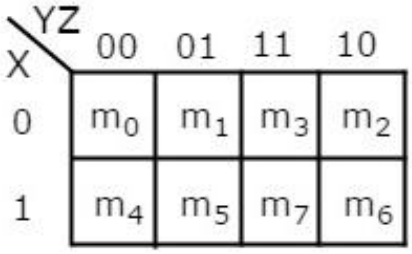


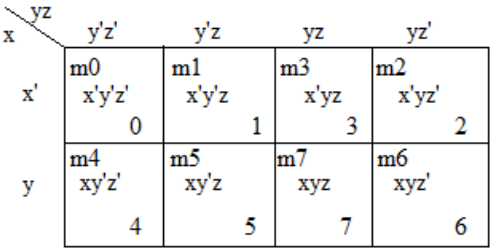
- In the above figure, there is only one possibility of grouping four adjacent minterms.

- 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

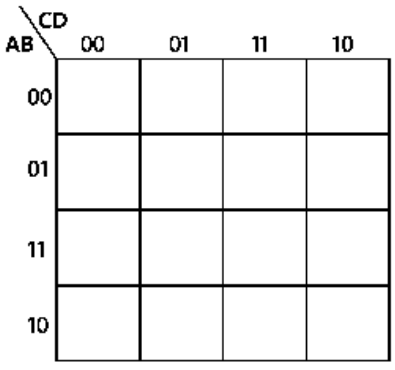
The 3-variable K-map is represented as an array of eight cells. In this case, we used A, B, and C for the variable. We can use any letter for the names of the variables. The binary values of variables A and B are along the left side, and the values of C are across the top. The value of the given cell 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.

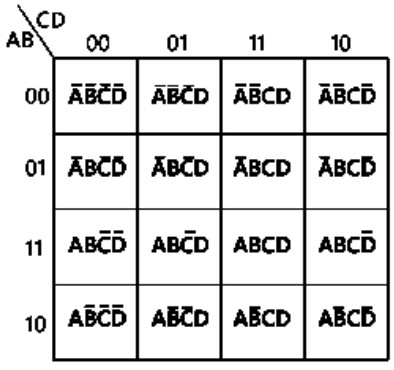




4-Variable Karnaugh Map

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

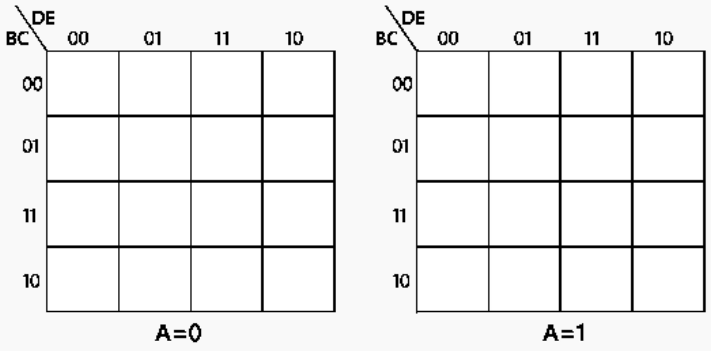




5-variable K-map

With the help of the 32- cell K-map, the boolean expression with 5 variables can be simplified. For constructing a 5-variable K-map, we use two 4-variable K-maps. The cell adjacencies within each of the 4- variable maps for the 5-variable map are similar to the 4- variable map.

A K-map for five variables (PQRST) can be constructed using two 4-variable maps. Each map contains 16 cells with all combinations of variables Q, R, S, and T. One map is for P = 0, and the other is for P = 1).



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:

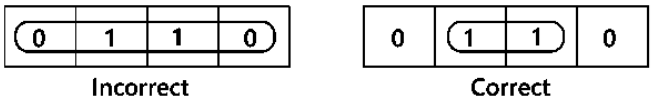
Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.

Step 3:

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

0	0	1	1
1	1	0	0

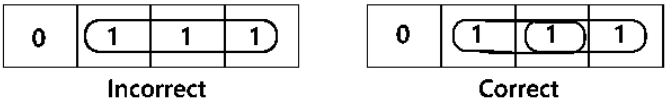
Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.



Incorrect Correct

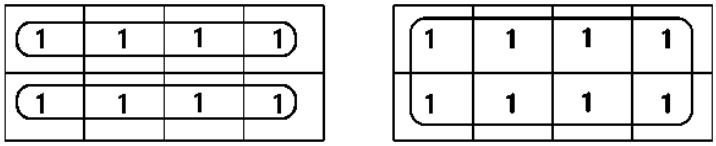
In a group, there is a total of 2^n number of ones. Here, n=0, 1, 2, …n.

Example: 2^0=1, 2^1=2, 2^2=4, 2^3=8, or 2^4=16.



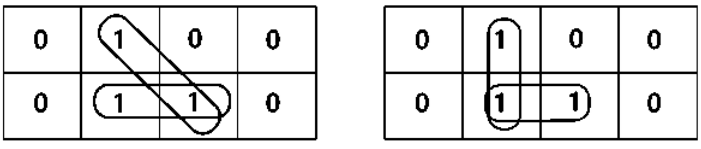
Incorrect Correct

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.



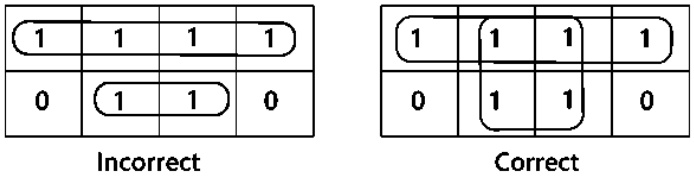
Incorrect Correct

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.



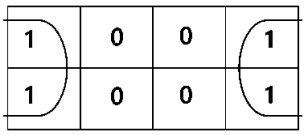
Incorrect Correct

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



Incorrect Correct

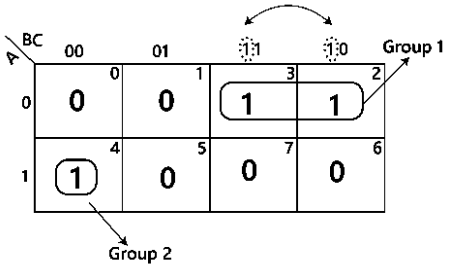
The elements located at the edges of the table are considered to be adjacent. So, we can group these elements.



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'.



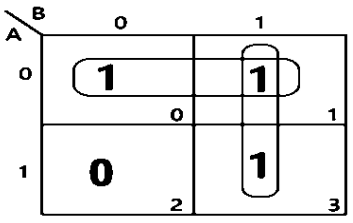
Step 5:

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'+AB'C'

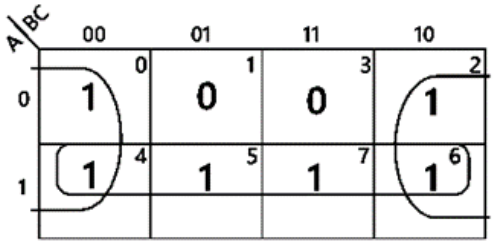
Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.

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



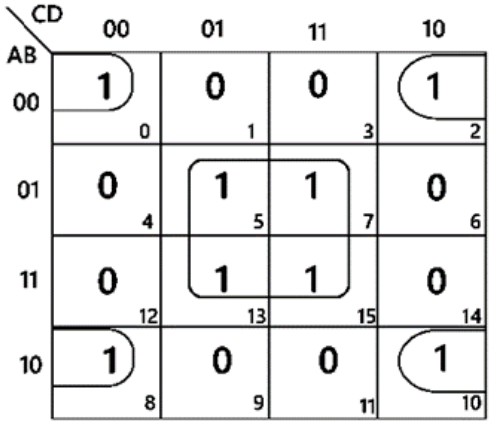
Simplified expression: Y=A'+B

Example 2: Y=A'B'C'+A' BC'+AB' C'+AB' C+ABC'+ABC



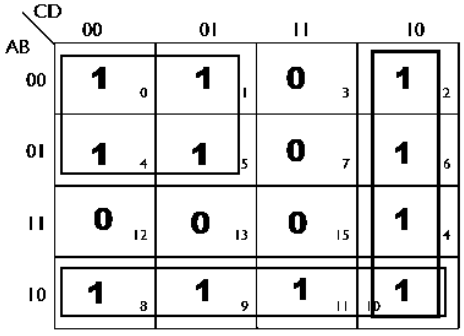
Simplified expression: Y=A+C'

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



Simplified expression: Y=BD+B'D'

Example 4: Obtain a simplified form using K Map: F(a, b, c, d) = ∑ (0, 1, 2, 4, 5, 6, 8, 9, 10, 11, 14).



F= A'C'+AB'+CD'

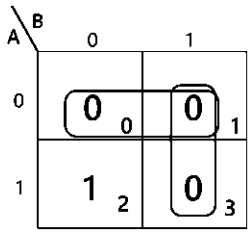
Maxterm Solution of K-Map

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:

1. 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.
2. We will make the groups of 'zeros' not for 'ones'.
3. Now, we will define the boolean expressions for each group as sum-terms.
4. At last, to find the simplified boolean expression in the POS form, we will combine the sum-terms of all individual groups.

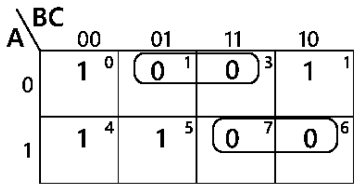
Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examples

Example 1: Y=(A'+B')(A'+B)(A+B)



Simplified expression: F'=(A'+B); F=AB'

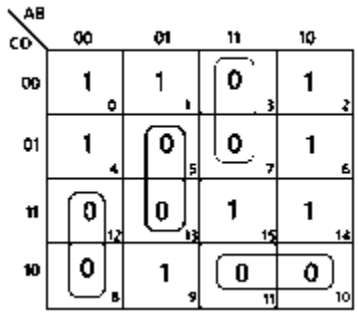
Example 2: Y=(A + B + C') (A + B' + C') (A' + B' + C) (A' + B' + C')



Simplified expression: Y'=(A'C+AB)'

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

Example 3: F(A,B,C,D)=π(3,5,7,8,10,11,12,13)

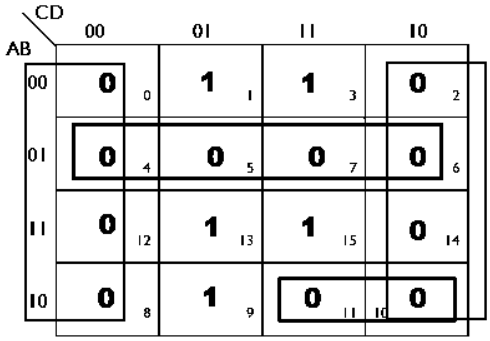


Simplified expression: Y'=(A'B'C+A'BD+ABC'+ACD')'

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

Example 4: Obtain a simplified form using K Map:

F(P,Q,R,S) = π (0,2,4,5,6,7,8,10,11,12,14)



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

Don't Care Condition

The "Don't care" condition says that we can use the blank cells of a K-map to make a group of the variables. To make a group of cells, we can use the "don't care" cells as either 0 or 1, and if required, we can also ignore that cell. We mainly use the "don't care" cell to make a large group of cells.

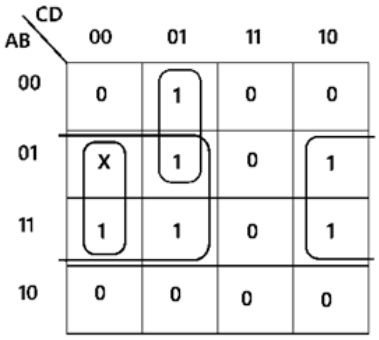
The cross(×) symbol is used to represent the "don't care" cell in K-map. This cross symbol represents an invalid combination. The "don't care" in excess-3 code are 0000, 0001, 0010, 1101, 1110, and 1111 because they are invalid combinations.

We can change the standard SOP function into a POS expression by making the "don't care" terms the same as they are. The missing minterms of the POS form are written as maxterms of the POS

form. In the same way, we can change the standard POS function into an SOP expression by making the "don't care" terms the same as they are. The missing maxterms of the SOP form are written as minterm of the SOP form.

Example 1: Minimize f = m(1,5,6,12,13,14) + d(4) in SOP minimal form

Solution:



So, the minimized SOP form of the function is:

f = BC' + BD' + A'C'D

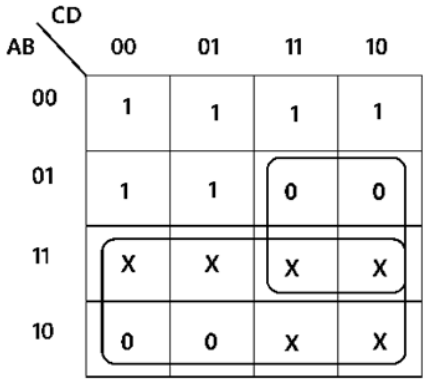
Example 2: Minimize F(A,B,C,D) = m(0,1,2,3,4,5) + d(10,11,12,13,14,15) in SOP minimal form

Solution:

The POS form of the given function is:

F(A,B,C,D) = M(6,7,8,9) + d(10,11,12,13,14,15)

The POS K-map for the given expression is:



So, the minimized POS form of the function is:

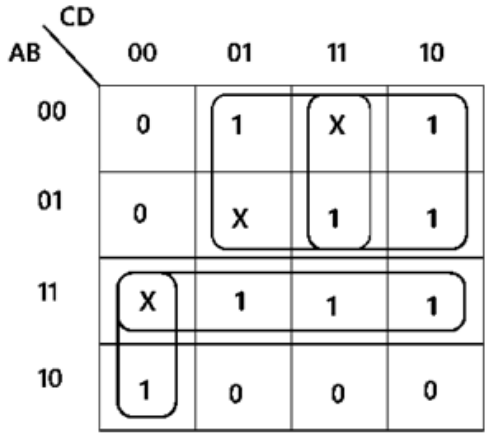
F = A'(B' + C')

Example-3:

Minimize the following function in SOP minimal form using K-Maps: F(A, B, C, D) = m(1, 2, 6, 7, 8, 13, 14, 15) + d(3, 5, 12)

Explanation:

The SOP K-map for the given expression is:



Therefore,

f = AC'D' + A'D + A'C + AB

Significance of "Don't Care" Conditions:

Don't Care conditions has the following significance with respect to the digital circuit design:

Simplification:

These conditions denote the set of inputs that never occurs for given digital circuits. Therefore, to simplify the boolean output expressions, the 'don't care' are used.