KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Module II

Boolean Algebra

Postulates of Boolean Algebra. Basic theorems and Properties of Boolean Algebra. Boolean Functions - Canonical and Standard forms. Simplification of Boolean Functions- Using Karnaugh- Map Method (upto five variables), Don't care conditions, Product of sums simplification, Tabulation Method. Digital Logic Gates- Implementation of Boolean functions using basic and universal gates.

Boolean Algebra

- Boolean algebra is an algebra for binary number system. It operates only on two numbers 0 & 1.
- It is a set of rules, laws and theorems by which logical operations can be mathematically expressed.
- In the Boolean algebra, Binary variables can take either of two values 0 and 1 (TRUE or FALSE),
- 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 Operators

1. **Dot sign(.):**
   - If the two variables are A and B, then their logical product is A × B = A.B
   - Dot operation is also called logical AND operation ⇒ A **AND** B

A	B	A **AND** B
0	0	0
0	1	0
1	0	0
1	1	1

2. **Plus sign(+):**
   - If the two variables are A and B, then their logical sum is A + B
   - Plus operation is also called logical OR operation⇒ A **OR** B

A	B	A **OR** B
0	0	0

0	1	1
1	0	1
1	1	1

3. **Complement**:
4.
   - (‘) or (¯). These signs indicates that the terms are complemented.
   - If the term is A then A’ is **NOT A**.

A	A’
0	1
1	0

Boolean Algebra is an algebraic structure defined on a set of elements K together with two binary operators + and . and provided the following (Huntington) postulates are satisfied:

Postulates/Axioms of Boolean Algebra

- **Postulate 1**: A Boolean algebra is a closed algebraic system containing a set K of two or more elements and the two operators · and + which refer to logical “AND” and logical “OR”.
- **Postulate 2:** Identity Elements
  There exist 0 and 1 elements in K, such that for every element x in K
  $x + 0 = A$
  $x \cdot 1 = A$
  0 is the identity element for + operation
  1 is the identity element for · operation
- **Postulate 3:** Commutativity
  Binary operators + and · are commutative.
  That is, for any elements x and y in K:
  $x + y = y + x$
  $x \cdot y = y \cdot x$
- **Postulate 4:** Distributivity
  Binary operator (+) is distributive over (·) and (·) is distributive over (+).
  That is, for any elements x, y and z in K:
  $x + (y \cdot z) = (x + y) \cdot (x + z)$
  $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

- **Postulate 5:** Complement
  A unary operation, complementation, exists for every element of K.
  That is, for any element x in K

  x.x' = 0
  x+ x' = 1
- **Postulate 6:**
  There exists at least two elements x, y in K such that x is not equal to y

Properties of Boolean Algebra

- Properties are stated as theorems
- Postulates are axioms and need no proof
- Theorems are proved from postulates

Theorems of Boolean Algebra

1. Idempotency Theorem

   A + A = A

   A . A = A

   Proof:

```
A + A = (A + A).1      (identity element)
= (A + A)(A + A`)      (complement)
= A + A A`             (distributivity)
= A + 0                (complement)
= A                    (complement)
By principle of duality A . A = A
```

2. Null Elements exist
   A + 1 = 1, for (+) operator
   A · 0 = 0, for (·) operator

```
Proof:
A + 1 = (A + 1).1        (identity element)
      = 1.(A + 1)        (commutativity)
      = (A + A`)(A + 1)  (complement)
```

```
= A + A`. 1        (distributivity)
= A + A`           (identity element)
= 1                (complement)
```

3. Involution Holds (Double Negation/Inversion Law/ Involution)
   (A`)` = A

   Proof:

   A + A` = 1 and A.A` = 0, (complements)

   or A` + A = 1 and A`.A = 0, (commutativity)

   i.e., A is complement of A`

   Therefore, A`` = A

4. Absorption Theorem

   A + A . B = A

   A .(A + B) = A

Proof:

```
A + A.B = A.1 + A.B  (identity element)
        = A.(1 + B)   (distributivity)
        = A.1        (Theorem 2: Null Elements exist)
        = A          (identity element)
```

5. **Associativity**

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

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

   AND laws

   A.0 = 0

   A.1 = A

   A.A = A

   A.A’ = 0

OR laws

A + 0 = A

A + 1 = 1

A + A = A

A + A' = 1

De Morgan's Theorems

I. Theorem 1: (A.B)' = A' + B'
Complement of product is equal to the sum of complements

Table showing verification of the De Morgan's first theorem −

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

II. Theorem 2: (A+B)' = A'.B'

Complement of sum is equal to the product of complements

Table showing verification of the De Morgan's second theorem −

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

Consensus Theorem

a) AB + A'C + BC = AB + A'C

BC: Consensus term

Proof:

AB + A'C + BC

=AB + A'C + BC.1

=AB + A'C + BC.(A + A') (A + A' =1)

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

=AB(1 + C) + A'C(1 + B)

=AB + A'C (1 + C =1)

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

(B+C): Consensus term

Duality Principle

This principle states that any algebraic equality derived from these axioms will still be valid whenever the OR and AND operators, and identity elements 0 and 1, have been interchanged. i.e. changing every OR into AND and vice versa, and every 0 into 1 and vice versa.

Boolean Expressions and Their Corresponding Duals:

Given Expression	Dual	Given Expression	Dual
0 = 1	1 = 0	A. (A+B) = A	A + A.B = A
0.1 = 0	1 + 0 = 1	AB = A + B	A+B = A.B
A.0 = 0	A + 1 = 1	(A+C) (A +B) = AB + AC	AC + AB = (A+B). (A+C)
A.B = B. A	A + B = B + A	A+B = AB + AB +AB	AB = (A+B).(A+B). (A+B)
A.A = 0	A + A = 1	AB + A + AB = 0	((A+B)).A.(A+B) = 1
A. (B.C) = (A.B). C	A+(B+C) = (A+B) + C		

Prove Transposition Theorem : AB + A'C = (A + C) (A' + B)

Proof:

RHS

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

= AA' + A'C + AB + CB

= 0 + A'C + AB + BC

= A'C + AB + BC(A + A')

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

= AB + A'C

= LHS

Prove Redundancy Theorem: AB + BC' + AC = AC + BC'

Proof

LHS

= AB + BC' + AC

= AB(C + C') + BC'(A + A') + AC(B + B')

= ABC + ABC' + ABC' + A'BC' + ABC + AB'c

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

= AC(B + B') + BC'(A + A')

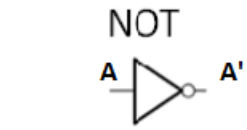
= AC + BC'

= RHS

Logic gates

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

1. NOT Gate (or Inverter)
   - It has one input A and one output Y.
   - Output of NOT gate is complement of input.
   - Logic diagram and Truth Table for NOT Gate

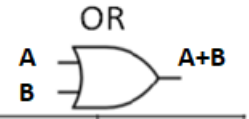


INPUT	OUTPUT
A	
0	1
1	0

2. AND Gate
   - It has n input (n >= 2) and one output.
   - Y = A.B
   - Output of AND gate is High (1) only if both inputs are High (1)
   - Logic diagram and Truth Table for AND Gate

INPUT		OUTPUT A.B
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

3. OR Gate
   - It has n input (n >= 2) and one output.
   - Y = A+ B
   - Output of OR gate is High (1) if atleast one input is High (1).
   - Logic diagram and Truth Table for OR Gate.

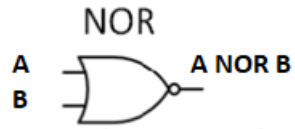


INPUT		OUTPUT A+B
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

4. NAND Gate
   - A NOT-AND operation is known as NAND operation.
   - It has n input (n >= 2) and one output.
   - Output of NAND gate is complement of AND gate. Output is low (0) only when both inputs are High (1).
   - Logic diagram and Truth Table for NAND Gate.

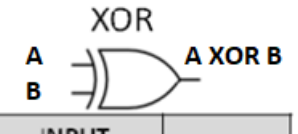
INPUT		OUTPUT A NAND B
A	B	
0	0	1
1	0	1
0	1	1
1	1	0

5. NOR Gate
   - A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.
   - Output of NOR gate is complement of OR gate. Output is low (0) when atleast one input is High(1).
   - Logic diagram and Truth Table for NOR Gate.



INPUT		OUTPUT A NOR B
A	B	
0	0	1
1	0	0
0	1	0
1	1	0

6. XOR Gate
   - XOR or Ex-OR gate is a special type of gate. It has n input (n >= 2) and one output.
   - Y = A ⊙ B
   - Output of 2-input XOR gate is high(1) when inputs are different, low (0) when inputs are same.
   - Output of 3-input XOR gate is high(1) when number of input that are high are odd, and low (0) when the number of inputs that are high are not odd.
   - Logic diagram and Truth Table for XOR Gate.



INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

7. XNOR Gate
   - XNOR or EX-NOR gate is a special type of gate. It has n input (n >= 2) and one output.
   - Y = A ⊙ B
   - Output of XNOR gate is high(1) when inputs are same, low (0) when inputs are different.
   - Logic diagram and Truth Table for XNOR Gate.

INPUT		OUTPUT A XNOR B
A	B	
0	0	1
1	0	0
0	1	0
1	1	1

Boolean Functions

- A **Boolean function** is described by an algebraic expression consisting of binary variables, the constants 0 and 1, and the logic operation symbols '.', '+', '''.
- For a given set of values of the binary variables involved, the boolean function can have a value of 0 or 1.
- Example.

$$F(A, B, C, D) = A + \overline{BC} + ADC \qquad \text{Equation No. 1}$$

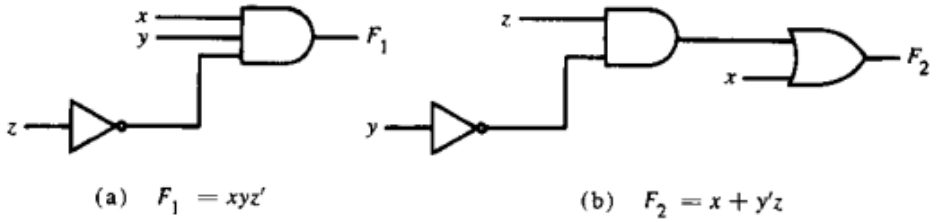
F (A, B, C, D) — Boolean Function; $A + \overline{BC} + ADC$ — Boolean Expression

Here the left side of the equation represents the output Y. So equation no. 1 can be written as

$$Y = A + \overline{BC} + ADC$$

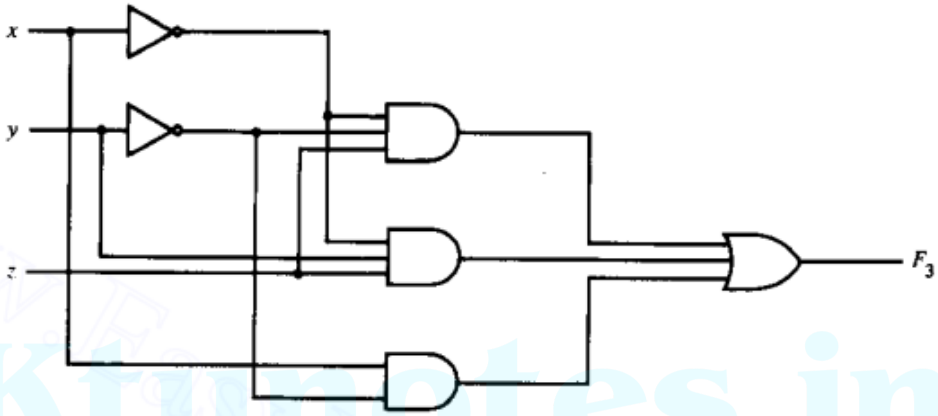
Implementation of Boolean functions with gates

Examples:



(a) $F_1 = xyz'$

(b) $F_2 = x + y'z$



(c) $F_3 = x'y'z + x'yz + xy'$

(d) $F_4 = xy' + x'z$

Canonical and Standard Forms

Any binary variable can take one of two forms, x or x'.

A boolean function can be expressed in terms of n binary variables. If all the binary variables are combined together using the AND operation, then there are a total of 2^n combinations since each variable can take two forms.

Minterm or **standard product** or product terms

Each of the combinations is called a minterm **or** standard product. A minterm is represented by m_i where i is the decimal equivalent of the binary number the minterm is designated.

Maxterm or standard sum or sum terms

If the variables are combined together with OR operation, then the term obtained is called a maxterm or standard sum. A maxterm is represented by M_i where i is the decimal equivalent of the binary number the maxterm is designated.

Minterms and Maxterms for function in 3 variables

Variables		Minterm		Maxterm	
x	y	Term	Designation	Term	Designation
0	0	x'y'	m_0	x+y	M0
0	1	x'y	m_1	x+y'	M1
1	0	xy'	m_2	x'+y	M2
1	1	xy	m_3	x'+y'	M3

Minterms and Maxterms for function in 3 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

Relation between Minterms and Maxterms

Each minterm is the complement of it's corresponding maxterm.

For example, for a boolean function in two variables –

$$m_0 = x'y'$$
$$(m_0)' = (x'y')'$$
$$(m_0)' = (x')' + (y')'$$
$$(m_0)' = x + y = M_0$$
In general $m_i = (M_i)'$ or $M_i = (m_i)'$

Constructing Boolean Functions

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 OR of all those terms.

Algebraic Procedure to Obtain Canonical SOP

a) Examine each term of a given sum-of-products expression; if it is not a minterm, go to the next step.
b) For all missing variable xi, multiply the term by (xi' + xi).
c) Multiply out all products and eliminate redundant product terms.

Example:

f (a, b, c)
= a.b' + b + a.b.c
= a.b' (c + c') + b (a + a')(c + 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' + a.b.c + a.b.c' + a'.b.c + a'.b.c'

Algebraic Procedure to Obtain Canonical POS

a) Examine each term of a given product-of-sums expression; if it is not a maxterm, go to the next step.
b) For all missing variable xi add the term xi' xi
c) Obtain the sum terms, and eliminate redundant terms.
Example:

f (a, b, c)
= a' (b' + c)
= (a' + bb' + cc') (b' + c + aa')
= (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)

Transforming One Form to Another

• Double complement the given function and apply De Morgan's theorem.
• Rule to be followed:
– Complement of a sum of true minterms is the same as the sum of the false minterms.

Example:
f (a, b, c) = a'.b'.c' + a'.b'.c + a'.b.c' + a.b.c' + a.b.c
f
= (f')'
= [(a'.b'.c' + a'.b'.c + a'.b.c' + a.b.c' + a.b.c)']'
= [a'.b.c + a.b.c' + a.b'.c]' (Consider remaining minterms)
= (a + b' + c')(a' + b' + c)(a' + b + c)

Example: Obtain the canonical sum of product form of the following function:

F (A, B) = A + B

Solution:

The given function contains two variables A and B. The variable B is missing from the first term of the expression and the variable A is missing from the second term of the expression. Therefore, the first term is to be multiplied by (B + B′) and the second term is to be multiplied by (A + A′) as demonstrated below.

F (A, B) = A + B

= A.1 + B.1

= A (B + B′) + B (A + A′)

= AB + AB′ + AB + A′B

= AB + AB′ + A′B (as AB + AB = AB)

Hence the canonical sum of the product expression of the given function is

F (A, B) = AB + AB′ + A′B.

Example: Obtain the canonical sum of product form of the following function.

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

Solution:

Here neither the first term nor the second term is minterm. The given function contains three variables A, B, and C. The variables B and C are missing from the first term of the expression and the variable A is missing from the second term of the expression. Therefore, the

first term is to be multiplied by (B + B′) and (C + C′). The second term is to be multiplied by (A + A′)..

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

= A (B + B′) (C + C′) + BC (A + A′)

= (AB + AB′) (C + C′) + ABC + A′BC

= ABC + AB′C + ABC′ + AB′C′ + ABC + A′BC

= ABC + AB′C + ABC′ + AB′C′ + A′BC (as ABC + ABC = ABC)

Hence the canonical sum of the product expression of the given function is

F (A, B, C) = ABC + AB′C + ABC′ + AB′C′ + A′BC.

Example : Obtain the canonical product of the sum form of the following function.

F (A, B, C) = (A + B′) (B + C) (A + C′)

Solution:

In the above three-variable expression, C is missing from the first term, A is missing from the second term, and B is missing from the third term. Therefore, CC′ is to be added with first term, AA′ is to be added with the second, and BB′ is to be added with the third term.

F (A, B, C) = (A + B′) (B + C) (A + C′)

= (A + B′ + 0) (B + C + 0) (A + C′ + 0)

= (A + B′ + CC′) (B + C + AA′) (A + C′ + BB′)

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

[using the distributive property, as X + YZ = (X + Y)(X + Z)]

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

[as (A + B′ + C′) (A + B′ + C′) = A + B′ + C′]

Hence the canonical product of the sum expression for the given function is

F (A, B, C) = (A + B′ + C) (A + B′ + C′) (A + B + C) (A′ + B + C) (A + B + C′)

Example: Obtain the canonical product of the sum form of the following function.

F (A, B, C) = A + B′C

Solution:

In the above three-variable expression, the function is given at sum of the product form. First, the function needs to be changed to product of the sum form by applying the distributive law as shown below.

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

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

Now, in the above expression, C is missing from the first term and B is missing from the second term. Hence CC' is to be added with the first term and BB' is to be added with the second term as shown below.

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

$= (A + B' + CC')(A + C + BB')$

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

[using the distributive property, as $X + YZ = (X + Y)(X + Z)$]

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

[as $(A + B' + C)(A + B' + C) = A + B' + C$]

Hence the canonical product of the sum expression for the given function is

$F(A, B, C) = (A + B' + C)(A + B' + C')(A + B + C)$.

Deriving a Sum of Products (SOP) Expression from a Truth Table

The sum of products (SOP) expression of a Boolean function can be obtained from its truth table summing or performing OR operation of the product terms corresponding to the combinations containing a function value of 1. In the product terms the input variables appear either in true (uncomplemented) form if it contains the value 1, or in complemented form if it possesses the value 0.

Now, consider the following truth table, for a three-input function Y. Here the output Y value is 1 for the input conditions of 010, 100, 101, and 110, and their corresponding product terms are A′BC′, AB′C′, AB′C, and ABC′ respectively.

Inputs			*Output*	*Product terms*	*Sum terms*
A	*B*	*C*	*Y*		
0	0	0	0		A + B + C
0	0	1	0		A + B + C′
0	1	0	1	A′BC′	
0	1	1	0		A + B′ + C′
1	0	0	1	AB′C′	
1	0	1	1	AB′C	
1	1	0	1	ABC′	
1	1	1	0		A′ + B′ + C′

The final sum of products expression (SOP) for the output Y is derived by summing or performing an OR operation of the four product terms as shown below.

Y = A′BC′ + AB′C′ + AB′C + ABC′

In general, the procedure of deriving the output expression in SOP form from a truth table can be summarized as below.

1. Form a product term for each input combination in the table, containing an output value of 1.

2. Each product term consists of its input variables in either true form or complemented form. If the input variable is 0, it appears in complemented form and if the input variable is 1, it appears in true form.

3. To obtain the final SOP expression of the output, all the product terms are OR operated.

Deriving a Product of Sums (POS) Expression from a Truth Table

The product of sums (POS) expression of a Boolean function can also be obtained from its truth table by a similar procedure. Here, an AND operation is performed on the sum terms corresponding to the combinations containing a function value of 0. In the sum terms the input variables appear either in true (uncomplemented) form if it contains the value 0, or in complemented form if it possesses the value 1.

Now, consider the same truth table, for a three-input function Y. Here the output Y value is 0 for the input conditions of 000, 001, 011, and 111, and their corresponding product terms are A + B + C, A + B + C′, A + B′ + C′, and A′ + B′ + C′ respectively.

So now, the final product of sums expression (POS) for the output Y is derived by performing an AND operation of the four sum terms as shown below.

$Y = (A + B + C)\,(A + B + C')\,(A + B' + C')\,(A' + B' + C')$

In general, the procedure of deriving the output expression in POS form from a truth table can be summarized as below.

1. Form a sum term for each input combination in the table, containing an output value of 0.
2. Each product term consists of its input variables in either true form or complemented form. If the input variable is 1, it appears in complemented form and if the input variable is 0, it appears in true form.
5. To obtain the final POS expression of the output, all the sum terms are AND operated.

Conversion between Canonical Forms

Complement of a function expressed as the sum of products (SOP) equals to the sum of products or sum of the minterms which are missing from the original function. This is because the original function is expressed by those minterms that make the function equal to 1, while its complement is 1 for those minterms whose values are 0.

According to the truth table given

$F(A,B,C) = \Sigma(2,4,5,6)$

$= m2 + m4 + m5 + m6$

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

This has the complement that can be expressed as

$F'(A,B,C) = (0,1,3,7)$

$= m0 + m1 + m3 + m7$

Now, if we take complement of F′ by DeMorgan's theorem, we obtain F as

$F(A,B,C) = (m0 + m1 + m3 + m7)'$

$= m0'.m1'.m3'.m'7$

$= M0.M1.M3.M7$

$= \Pi(0,1,3,7)$

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

the maxterm with subscript j is a complement of the minterm with the same subscript j, and vice versa, $m'_j = M_j$.

In general, to convert from one canonical form to other canonical form, it is required to interchange the symbols Σ and π, and list the numbers which are missing from the original form.

To find the missing terms, the total 2^n number of minterms or maxterms must be realized, where n is the number of variables in the function.

Karnaugh Map(K-Map) method

The **K-map** is a systematic way of simplifying Boolean expressions.

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.

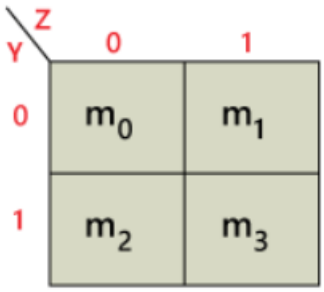
Drawback: –
Since it is a pictorial approach, it is difficult to visualize functions with more than 5 variables

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 24. 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, 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

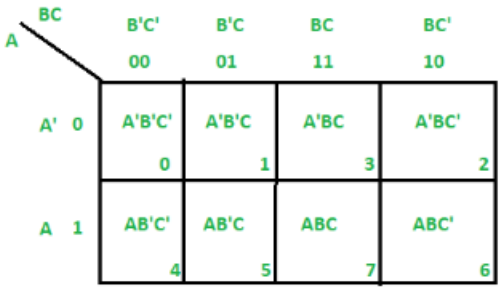
There is a total of 4 variables in a 2-variable K-map. There are two variables in the 2-variable K-map. The following figure shows the structure of the 2-variable K-map:



- 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

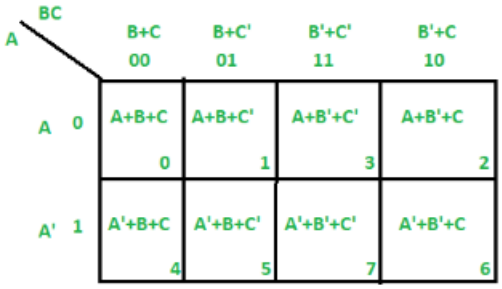
The 3-variable K-map is represented as an array of $2^3 = 8$ cells.



SOP(MINTERMS)

8 Blocks = 1
4 Blocks = 1 variable term
2 Blocks = 2 variable term
1 Block = 3 variable term

K-map SOP form for 3 variables



POS (MAXTERMS)

8 Blocks = 0
4 Blocks = 1 variable term
2 Blocks = 2 variable term
1 Block = 3 variable term

K-map 3 variable POS form

4-Variable Karnaugh Map

The 4-variable K-map is represented as an array of 2^4= 16 cells.

CD / AB	C'D' (00)	C'D (01)	CD (11)	CD' (10)
A'B' (00)	A'B'C'D' 0	A'B'C'D 1	A'B'CD 3	A'B'CD' 2
A'B (01)	A'BC'D' 4	A'BC'D 5	A'BCD 7	A'BCD' 6
AB (11)	ABC'D' 12	ABC'D 13	ABCD 15	ABCD' 14
AB' (10)	AB'C'D' 8	AB'C'D 9	AB'CD 11	AB'CD' 10

SOP(MINTERMS)

16 Blocks = 1
8 Blocks = 1 variable term
4 Blocks = 2 variable term
2 Blocks = 3 variable term
1 Block = 4 variable term

K-map 4 variable SOP form

CD / AB	C+D (00)	C+D' (01)	C'+D' (11)	C'+D (10)
A + B (00)	A+B+C+D 0	A+B+C+D' 1	A+B+C'+D' 3	A+B+C'+D 2
A + B' (01)	A+B'+C+D 4	A+B'+C+D' 5	A+B'+C'+D' 7	A+B'+C'+D 6
A'+B' (11)	A'+B'+C+D 12	A'+B'+C+D' 13	A'+B'+C'+D' 15	A'+B'+C'+D 14
A'+B (10)	A'+B+C+D 8	A'+B+C+D' 9	A'+B+C'+D' 11	A'+B+C'+D 10

POS(MAXTERMS)

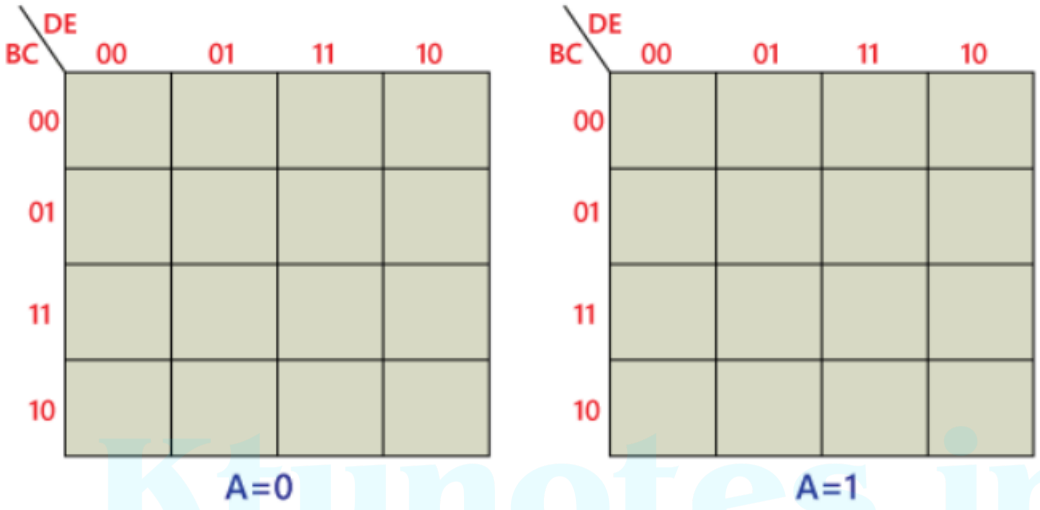
16 Blocks = 0
8 Blocks = 1 variable term
4 Blocks = 2 variable term
2 Blocks = 3 variable term
1 Block = 4 variable term

K-map 4 variable POS form

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).

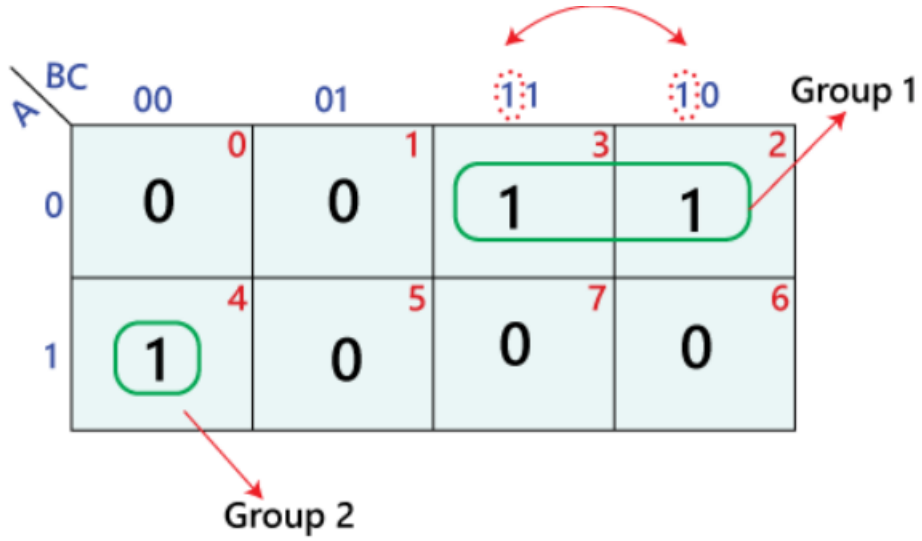


Simplification of boolean expressions using Karnaugh Map

- **Step 1:** First define the given expression in its canonical form
- **Step 2:** 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:** form the groups by considering each one in the K-map (refer K-map grouping rules)
- **Step 4:** find the boolean expression for each group. By looking at the common variables in cell-labeling, define the groups in terms of input variables.
- **Step 5:** find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, combine the product-terms of all individual groups.

Illustration of Step 4

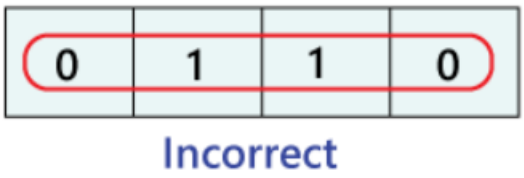
- In the below example, there is a total of two groups, i.e., group 1 and group 2, with two and one number of 'ones'.



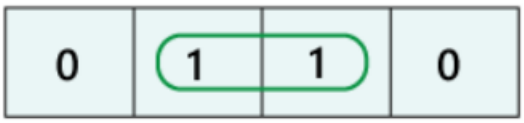
- In the first group, the ones are present in the row for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In these columns, only B term in common is the product term corresponding to the group as A'B. Just like group 1, 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'.

K-Map Grouping Rules

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

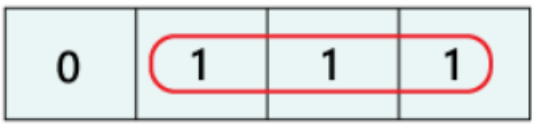


Incorrect



Correct

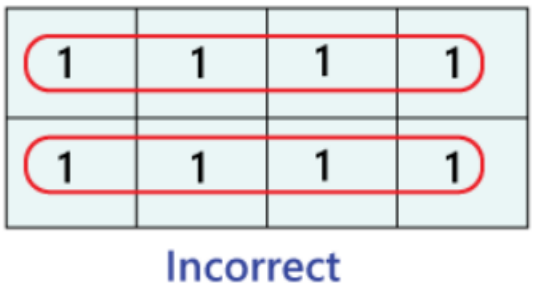
2. In a group, there should be a total of 2^n number of ones. Here, n=0, 1, 2, …n.



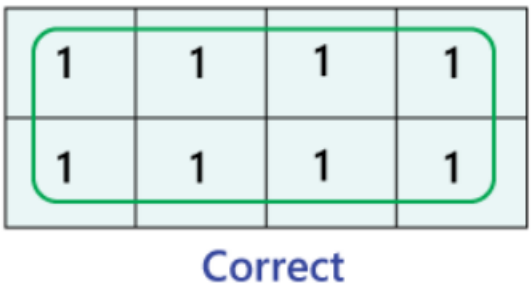
Incorrect

Correct

3. 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.

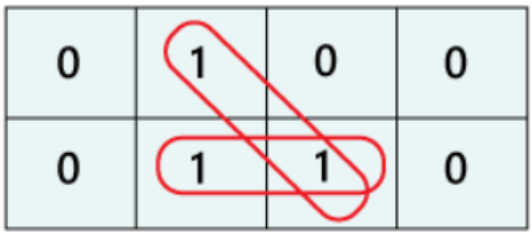


Incorrect

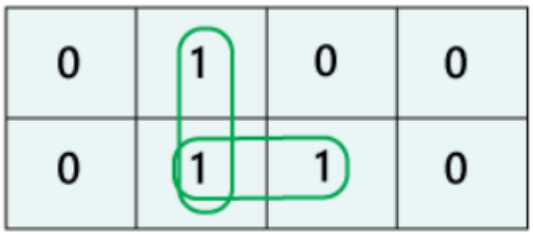


Correct

4. 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.

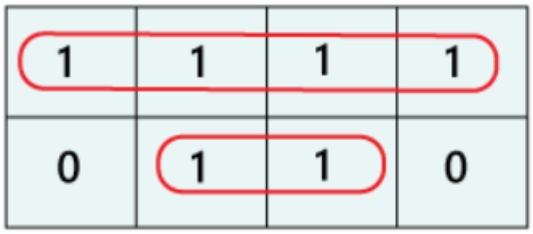


Incorrect

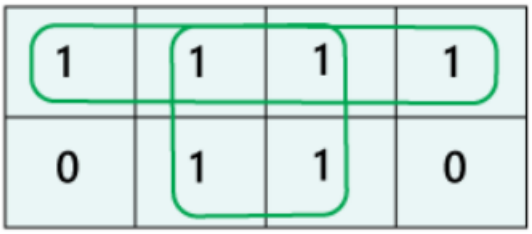


Correct

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

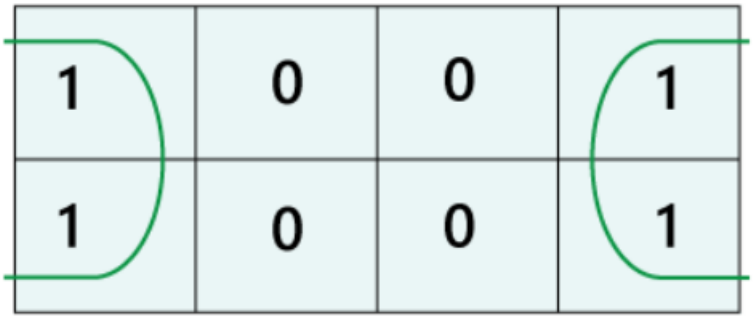


Incorrect

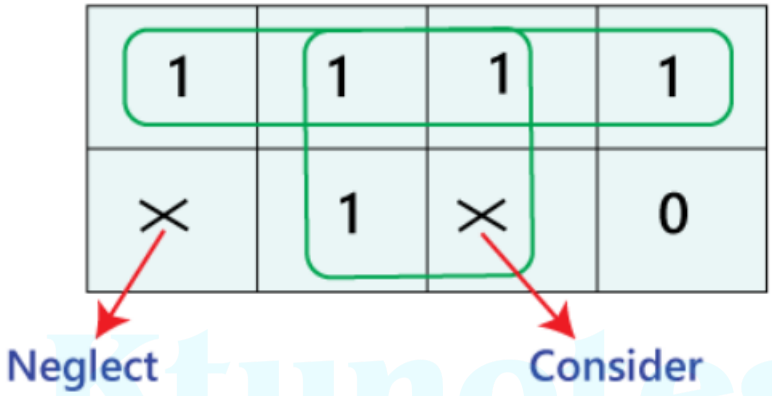


Correct

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



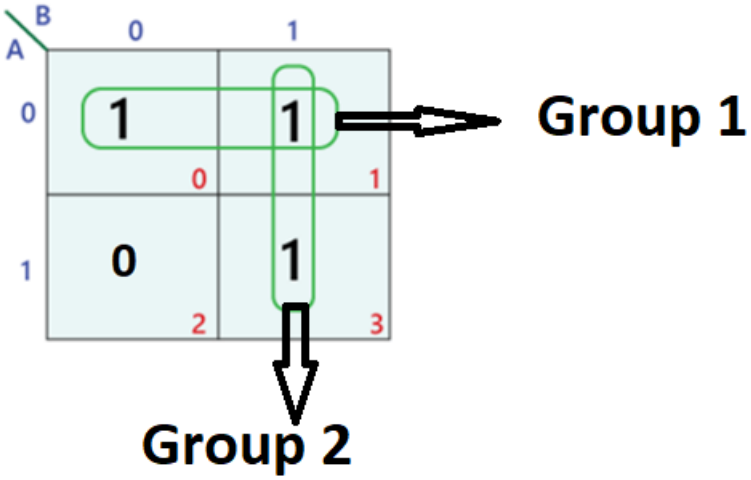
7. consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded



Example 1: Simplify using Kmap Y=A'B' + A'B+ AB

Solution:

- Given expression is in canonical form, as every term contains all the variables A and B.
- Expression contains 2 variables A and B, hence 2 variable K map with 4 cells required.
- enter 1 to each product-term into the K-map cell and fill the remaining cells with zeros.
- form the groups by considering ones in the K-map (refer K-map grouping rules)



- find the boolean expression for each group by looking at the common variables in cell-labeling

  Group 1 term= common variables in group 1 = A'

  Group 2 term= common variables in group 2 = B

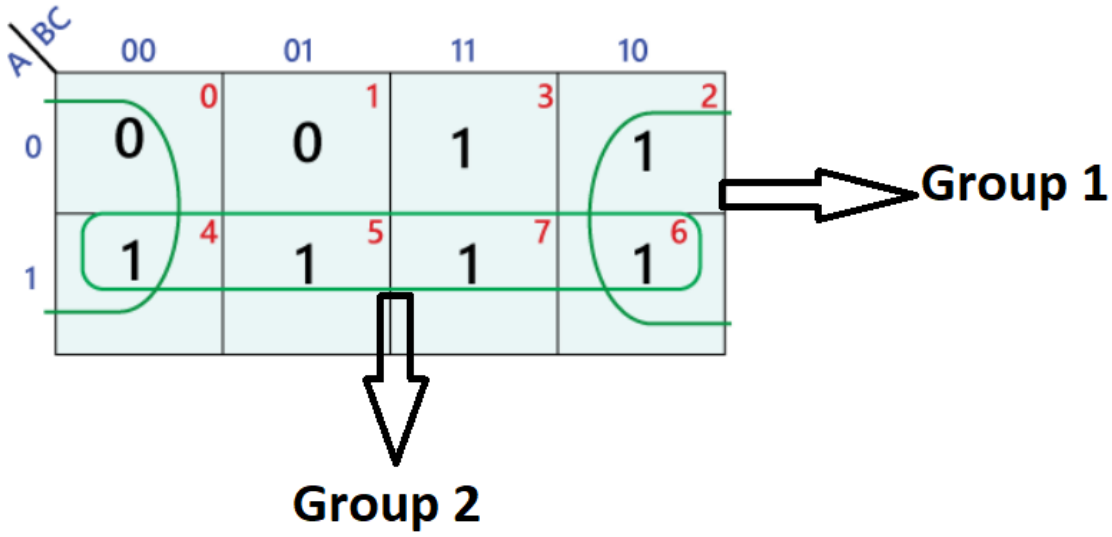
- simplified boolean expression in the SOP form is obtained by combining the product-terms of all individual groups.

Ans: Y=A'+B

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

Solution:

- Given expression is in canonical form, as every term contains all the variables A, B and C.
- Expression contains 3 variables A, B and C, hence 3 variable K map with 8 cells required.



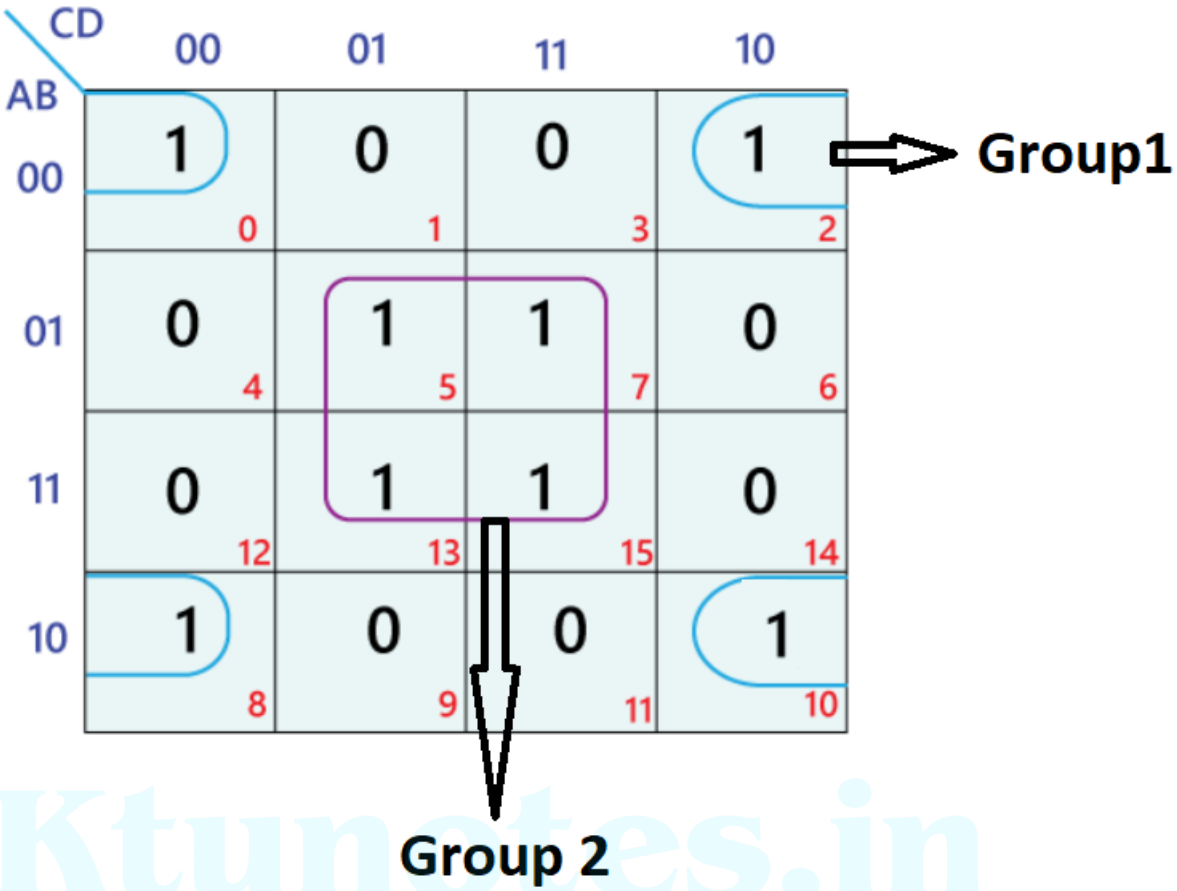
Group 1 : C′

Group 2 : A

Simplified expression: Y=A+C'

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

Or Simplify using Kmap F (A, B, C, D) = ∑(0, 2, 5, 7, 8, 10, 13, 15)



Group 1: B'D'

Group 2: BD

Simplified expression: Y=BD+B'D'

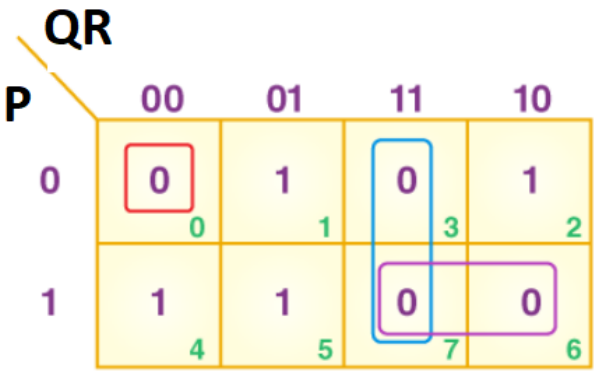
Maxterm Solution of K-Map

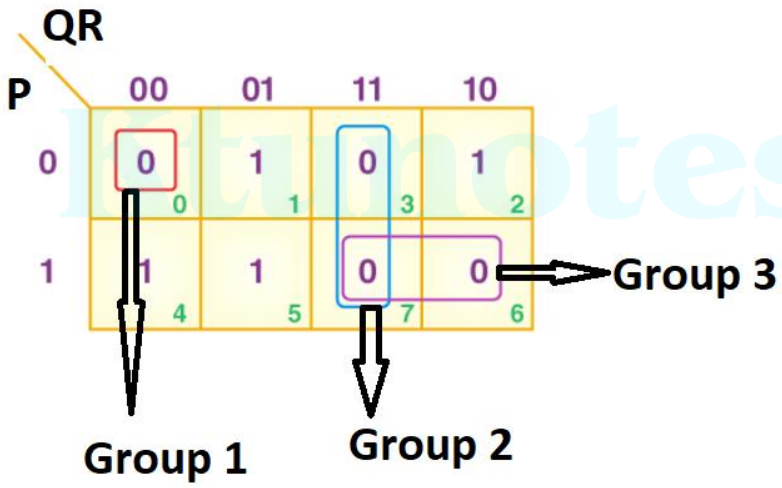
To find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:

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.

Example: Simplify using Kmap F (P, Q, R) = π(0,3,6,7)





Group 1 : P + Q + R

Group 2 : Q'+R'

Group 3 : P' + Q'

Simplified expression: F (P, Q, R) = (P' + Q') (Q' + R') (P + Q + R)

Handling Don't Care inputs

There exists functions for which some of the inputs are treated as don't cares and the corresponding output values do not matter. Such inputs will never appear.

The don't care minters are treated as 'X in the Kmap

When creating the groups, we can include the cells marked as 'X' along with those marked as '1' to make the group bigger. It is not necessary to include all cells marked as 'X'.

$$F(A, B, C, D) = \sum m(1,5,9,11,12,13) + \sum d(3,10,14,15)$$

AB \ CD	00	01	11	10
00		1	X	
01		1		
11	1	1	X	X
10		1	1	X