SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF Computer Science and Engineering

COURSE MATERIAL

Subject Name: Fundamentals of Digital Systems UNIT II Subject Code: SCSA1201

Axiomatic definitions of Boolean Algebra - Basic Theorems and Properties of Boolean Algebra - Boolean Functions- Canonical and Standard forms - Digital Logic Gates- Simplification of Boolean Expressions, The map method- SOP and POS - NAND and NOR implementation - Don't Cares - The Tabulation Method - Determination and Selection of Prime Implicants.

1. Axiomatic Definition of Boolean algebra

1. Closure
   a. Closure with respect to (wrt) OR (+)
   b. Closure with respect to AND (·)
2. Identity
   a. Identity element wrt to OR : 0
   b. Identity element wrt to AND : 1
3. Commutative Property
   a. Commutative Property wrt to OR : $x + y = y + x$
   b. Commutative Property wrt to AND : $x \cdot y = y \cdot x$
4. Distributive Property
   $X \cdot (y + z) = (x \cdot y) + (x \cdot z)$
   $x + (y \cdot z) = (x + y)(x + z)$
5. Existence of Complement
   $x + x' = 1$
   $x \cdot x' = 0$

Precedence:

(1) Parentheses (2) NOT (3) AND (4) OR

2.Basic Theorems and Properties of Boolean algebra

Operations with 0 and 1:

- $X + 0 = X$
- $X \bullet 1 = X$
- $X + 1 = 1$
- $X \bullet 0 = 0$

Idempotent laws

- $X + X = X$
- $X \bullet X = X$

Involution law:

- (X') ' = X

Laws of complementarity:

- X + X' = 1
- X • X' = 0

Commutative laws:

- X + Y = Y + X
- X • Y = Y • X

Associative laws:

- (X + Y) + Z = X + (Y + Z) = X + Y + Z
- (XY)Z = X(YZ) = XYZ

Distributive laws:

- X(Y + Z) = XY + XZ
- X + YZ = (X + Y) (X + Z)

Simplification theorems:

- X Y + X Y' = X
- (X + Y) (X + Y') = X
- X + XY = X
- X (X + Y) = X
- (X + Y') Y = XY
- XY' + Y = X + Y

DeMorgan's laws:

There are two "de Morgan´s" rules or theorems,

- Two separate terms NOR´ed together is the same as the two terms inverted (Complement) and AND´ed for example, (X+Y)' = X'.Y'.
- Two separate terms NAND´ed together is the same as the two terms inverted (Complement) and OR´ed for example, (X.Y)' =X' +Y'.

Duality:

"Every algebraic expression deducible from the postulates of Boolean Algebra remains valid if the operations and identity elements are interchanged."

- (X + Y + Z + …) D = X Y Z …
- (X Y Z…) D = X + Y + Z + …
- [f (X1, X2, … XN, 0, 1, +, •)] D = f (X1, X2, … XN, 1, 0, •, +)

3. Boolean Functions

A simple 2-input AND, OR and NOT Gates can be represented by 16 possible functions as shown in the following table.

3.1 Laws of Boolean Algebra

	Function	Description	Expression
1.	NULL	0	
2.	IDENTITY	1	
3.	Input	A	A
4.	Input	B	B
5.	NOT	A	A’
6.	NOT	B	B’
7.	A AND B (AND)	A . B	
8.	A AND NOT B	A . B’	
9.	NOT A AND B	A’ . B	
10.	NOT A AND NOT B (NAND)		A’ . B’
11.	A OR B (OR)	A + B	
12.	A OR NOT B	A + B’	
13.	NOT A OR B	A’ + B	
14.	NOT OR (NOR)	(A + B)’	
15.	Exclusive-OR	A.B’ + A’.B	
16.	Exclusive-NOR	A’.B’ + A.B	

Example

Using the above laws, simplify the following expression: (A + B)(A + C)

Q =	(A + B).(A + C)	
	A.A + A.C + A.B + B.C	– Distributive law
	A + A.C + A.B + B.C	– Idempotent AND law (A.A = A)
	A(1 + C) + A.B + B.C	– Distributive law
	A.1 + A.B + B.C	– Identity OR law (1 + C = 1)
	A(1 + B) + B.C	– Distributive law
	A.1 + B.C	– Identity OR law (1 + B = 1)
Q =	A + (B.C)	– Identity AND law (A.1 = A)

Then the expression: (A + B)(A + C) can be simplified to A + (B.C) as in the Distributive law.

4. Canonical and Standard Forms

Logical functions are generally expressed in terms of different combinations of logical variables with their true forms as well as the complement forms. Binary logic values obtained by the logical functions and logic variables are in binary form. An arbitrary logic function can be expressed in the following forms.

(i) Sum of the Products (SOP)
(ii) Product of the Sums (POS)

Product Term:

In Boolean algebra, the logical product of several variables on which a function depends is considered to be a product term. In other words, the AND function is referred to as a product term or standard product. The variables in a product term can be either in true form or in complemented form. For example, ABC′ is a product term.

Sum Term:

An OR function is referred to as a sum term. The logical sum of several variables on which a function depends is considered to be a sum term. Variables in a sum term can also be either in true form or in complemented form. For example, A + B + C′ is a sum term.

Sum of Products (SOP):

The logical sum of two or more logical product terms is referred to as a sum of products expression. It is basically an OR operation on AND operated variables. For example, Y = AB + BC + AC or Y = A′B + BC + AC′ are sum of products expressions.

Product of Sums (POS):

Similarly, the logical product of two or more logical sum terms is called a product of sums expression. It is an AND operation on OR operated variables. For example, Y = (A + B + C)(A + B′ + C)(A + B + C′) or Y = (A + B + C)(A′ + B′ + C′) are product of sums expressions.

Standard form:

The standard form of the Boolean function is when it is expressed in sum of the products or product of the sums fashion. The examples stated above, like Y =AB + BC + AC or Y = (A + B + C)(A + B′ + C)(A + B + C′) are the standard forms. However, Boolean functions are also sometimes expressed in nonstandard forms like F = (AB + CD)(A′B′ + C′D′), which is neither a sum of products form nor a product of sums form. However, the same expression can be converted to a standard form with help of various Boolean properties, as:

F = (AB + CD)(A′B′ + C′D′) = A′B′CD + ABC′D′

4.1 Minterm

A product term containing all n variables of the function in either true or complemented form is called the minterm. Each minterm is obtained by an AND operation of the variables in their true form or complemented form. For a two-variable function, four different combinations are possible, such as, A′B′, A′B, AB′, and AB. These product terms are called the fundamental products or standard products or minterms. In the minterm, a variable will possess the value 1 if it is in true or uncomplemented form, whereas, it contains the value 0 if it is in complemented form. For three variables function, eight minterms are possible as listed in the following table

A	B	C	Minterm
0	0	0	A’B’C’
0	0	1	A’B’C

0	1	0	A’BC’
0	1	1	A’BC
1	0	0	AB’C’
1	0	1	AB’C
1	1	0	ABC’
1	1	1	ABC

So, if the number of variables is n, then the possible number of minterms is 2n. The main property of a minterm is that it possesses the value of 1 for only one combination of n input variables and the rest of the 2n – 1 combinations have the logic value of 0. This means, for the above three variables example, if A = 0, B = 1, C = 1 i.e., for input combination of 011, there is only one combination A′BC that has the value 1, the rest of the seven combinations have the value 0.

Canonical Sum of Product Expression:

When a Boolean function is expressed as the logical sum of all the minterms from the rows of a truth table, for which the value of the function is 1, it is referred to as the canonical sum of product expression. The same can be expressed in a compact form by listing the corresponding decimal-equivalent codes of the minterms containing a function value of 1.

For example, if the canonical sum of product form of a three-variable logic function F has the minterms A′BC, AB′C, and ABC′, this can be expressed as the sum of the decimal codes corresponding to these minterms as below.

F (A,B,C) = (3,5,6)

= m3 + m5 + m6

= A′BC + AB′C + ABC′

where Σ (3,5,6) represents the summation of minterms corresponding to decimal codes 3, 5, and 6. The canonical sum of products form of a logic function can be obtained by using the following procedure:

1. Check each term in the given logic function. Retain if it is a minterm, continue to examine the next term in the same manner.

2. Examine for the variables that are missing in each product which is not a minterm. If the missing variable in the minterm is X, multiply that minterm with (X+X′).

2. Multiply all the products and discard the redundant terms.

4.2 Maxterm

A sum term containing all n variables of the function in either true or complemented form is called the maxterm. Each maxterm is obtained by an OR operation of the variables in their true form or complemented form. Four different combinations are possible for a two-variable function, such as, A′ + B′, A′ + B, A + B′, and A + B. These sum terms are called the standard sums or maxterms. Note that, in the maxterm, a variable will possess the value 0, if it is in true or uncomplemented form, whereas, it contains the value 1, if it is in complemented form. Like minterms, for a three-variable function, eight maxterms are also possible as listed in the following table

A	B	C	Maxterm
0	0	0	A+B+C
0	0	1	A+B+C’
0	1	0	A+B’+C
0	1	1	A+B’+C’
1	0	0	A’+B+C
1	0	1	A’+B+C’
1	1	0	A’+B’+C
1	1	1	A’+B’+C’

So, if the number of variables is n, then the possible number of maxterms is 2n. The main property of a maxterm is that it possesses the value of 0 for only one combination of n input variables and the rest of the 2n –1 combinations have the logic value of 1. This means, for the above three variables example, if A = 1, B = 1, C = 0 i.e., for input combination of 110, there is only one combination A′ + B′ + C that has the value 0, the rest of the seven combinations have the value 1.

Canonical Product of Sum Expression:

When a Boolean function is expressed as the logical product of all the maxterms from the rows of a truth table, for which the value of the function is 0, it is referred to as the canonical product of sum expression. The same can be expressed in a compact form by listing the corresponding decimal equivalent codes of the maxterms containing a function value of 0. For example, if the canonical product of sums form of a three-variable logic function F has the maxterms A + B + C, A + B′ + C, and A′ + B + C′, this can be expressed as the product of the decimal codes corresponding to thesemaxterms as below,

$F(A,B,C) = \Pi(0,2,5)$

$= M0\ M2\ M5$

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

where Π (0,2,5) represents the product of maxterms corresponding to decimal codes 0, 2, and 5. The canonical product of sums form of a logic function can be obtained by using the following procedure.

1. Check each term in the given logic function. Retain it if it is a maxterm, continue to examine the next term in the same manner.

2. Examine for the variables that are missing in each sum term that is not a maxterm. If the missing variable in the maxterm is X, add that maxterm with (X.X′).

3. Expand the expression using the properties and postulates as described earlier and discard the redundant terms. Some examples are given here to explain the above procedure.

5. **Boolean Function**

Boolean algebra deals with binary variables and logic operation. A **Boolean Function** is described by an algebraic expression called **Boolean expression** which consists of binary variables, the constants 0 and 1, and the logic operation symbols. Consider the following example

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

5.1 Truth Table Formation

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.

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

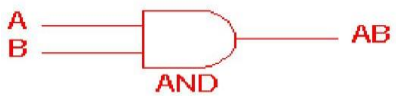
The output will be high (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is shown by Table (a). The number of rows in the truth table is 2^n where n is the number of input variables (n=3 for the given equation). Hence there are $2^3 = 8$ possible input combination of inputs.

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

6. DIGITAL LOGIC GATES

A large number of electronic circuits (in computers, control units, and so on) are made up of logic gates.Digital systems are said to be constructed by using logic gates. These process signals which represent true or false. The basic gates are the AND, OR, NOT gates. The most common symbols used to represent logic gates are shown below.

AND gate:



2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

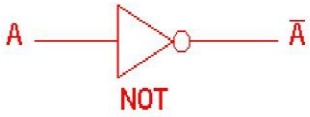
The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB.

OR gate:

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high. A plus (+) is used to show the OR operation.

NOT gate:



NOT gate	
A	$\overline{A}$
0	1
1	0

7. Simplification of Boolean Expressions

Minimization of Boolean functions is an approach where a given Boolean expression can be transformed from one form to another equivalent form by applying Boolean Theorems. By minimizing the expressions the individual components used in electrical circuits can be minimized or reduced. This allows designers to make use of fewer components, thus reducing the cost of a particular system. It should be noted that there are no fixed rules that can be used to minimize a given expression. It is left to an individual's ability to apply Boolean Theorems in order to minimize a function.

Examples:

Example 1:

Using Boolean algebra techniques, simplify the expression X . Y + X (Y + Z) + Y (Y + Z)

Solution:

Given: X . Y + X (Y + Z) + Y (Y + Z).
Applying distributive property, we get

X . Y + X (Y + Z) + Y (Y + Z) = X . Y + X . Y + X . Z + Y . Y + Y . Z

We know B . B = B
= X . Y + X . Y + X . Z + Y + Y . Z

We know A . B + A . B = A . B
= X . Y + X . Z + Y + Y . Z
= X . Y + X . Z + Y [We know (B + BC = B)]
= Y + XZ

Example 2:

Using Boolean algebra techniques, simplify this expression: AB + A(B + C) + B(B + C)

Solution
Apply the distributive law to the second and third terms in the expression, as follows:

AB + A(B + C) + B(B + C)=AB + AB + AC + BB + BC =AB + AB + AC + B + BC
[BB = B] =AB + AC + B + BC [AB + AB = AB] =AB + AC + B [B
+ BC = B] =B+AC
[AB + B = B]

Example 3:

Using Boolean algebra techniques, simplify this expression A.B' + A.B + B.C

Solution

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

Example 4:

Using Boolean algebra techniques, simplify this expression A'.B.C + A.B'.C + A.B.C' + A.B.C

Solution:

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

7.1 STANDARD FORMS OF BOOLEAN EXPRESSIONS

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of-sums form.

Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

7.1.1 The Sum-of-Products (SOP) Form

When two or more product terms are summed by Boolean addition, the resulting expression is a sum-of-products (SOP). Some examples are:

AB + ABC
ABC + C'DE + B'CD'
AB + BCD + AC

Also, an SOP expression can contain a single-variable term, as in

A + ABC' + BCD'.

In an SOP expression a single over bar cannot extend over more than one variable.

Example

Convert each of the following Boolean expressions to SOP form:
(a) AB + B(CD + EF)
(b) (A + B)(B + C + D)
(c) [(A + B)' + C+'

The Standard SOP Form

So far, you have seen SOP expressions in which some of the product terms do not contain all of the variables in the domain of the expression.

For example, the expression A'BC' + AB'D + ABC'D' has a domain made up of the variables A, B, C. and D. However, notice that the complete set of variables in the domain is not represented in the first two terms of the expression; that is, D or D' is missing from the first term and C or C' is missing from the second term.

A standard SOP expression is one in which all the variables in the domain appear in each product term in the expression. For example, A'BCD' + ABC'D+ AB'CD are a standard SOP expression.

Converting Product Terms to Standard SOP:

Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard SOP to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into standard form using Boolean algebra rule → (A + A' = 1) i.e., A variable added to its complement equals 1.

Step 1: Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms. As you know, you can multiply anything by 1 without changing its Value.

Step 2: Repeat Step 1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable.

Example
Convert the following Boolean expression into standard SOP form: AB'C + A'B' + ABC'D

Solution

The domain of this SOP expression A, B, C, D. Take one term at a time.

The first term, ABC, is missing variable D or D', so multiply the first term by(D + D) as follows: AB'C =
AB'C(D + D') = AB'CD + AB'CD'
In this case, two standard product terms are the result.

The second term, A'B'; is missing variables C or C' and D or D', so first multiply the second term by C + C' as follows:
A'B' = A'B'(C + C') = A'B'C + A'B'C'

The two resulting terms are missing variable D or D', so multiply both terms by (D + D) as follows
A'B'C(D + D') + A'B'C'(D + D')= A'B'CD + A'B'CD' + A'B'C'D + A'B'C'D'

In this case, four standard product terms are the result.

The third term, ABC'D, is already in standard form. The complete standard SOP form of the original expression is as follows:
AB'C + A'B' + ABC'D = AB'CD + AB'CD '+ A' B'CD + A'B'CD' + A'B'C'D +A'B'C'D' + ABC'D

7.1.2 The Product-of-Sums (POS) Form

A sum term was defined before as a term consisting of the sum (Boolean addition) of literals (variables or

their complements). When two or more sum terms are multiplied, the resulting expression is a product-of-sums (POS). Some examples are

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

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

A POS expression can contain a single-variable term, as inA(A + B + C)(B + C + D).

In a POS expression, a single over bar cannot extend over more than one variable; however, more than one variable in a term can have an over-bar. For example, a POS expression can have the term A' + B' + C' but not [A + B + C+'.

Implementation of a POS Expression simply requires ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation and the product of two or more sum terms is produced by an AND operation.

The Standard POS Form

So far, you have seen POS expressions in which some of the sum terms do not contain all of the variables in the domain of the expression.

For example, the expression(A' + B + C) (A + B + D') (A + B' + C' + D)has a domain made up of the variables A, B, C, and D. Notice that the complete set of variables in the domain is not represented in e first two terms of the expression; that is, D or D' is missing from the first term and C or C' is missing from the second term.

A standard POS expression is one in which all the variables in the domainappear in each sum term in the expression. For example,(A' + B' + C + D)(A + B' + C + D)(A + B + C + D)is a standard POS expression.

Converting a Sum Term to Standard POS

Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a Nonstandard POS expression is converted into standard form using Boolean algebra rule $\rightarrow$ (A'. A = 0) i.e., A variable multiplied to its complement equals 0.

Step 1. Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms. As you know, you can add 0 to anything without changing its value.

Step 2. Apply rule A + BC = (A + B)(A + C).

Step 3. Repeat Step 1 until all resulting sum terms contain all variables in the domain in either complemented or non-complemented form.

Example

Convert the following Boolean expression into standard POS form:(A' + B + C)(B' + C + D')(A + B' + C' + D)

Solution

The domain of this POS expression is A, B, C, D. Take one term at a time.

The first term, A + B + C, is missing variable D or D', so add D'D and apply rule as follows: A' + B +
C = A' + B + C + D'D

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

The second term, B' + C + D', is missing variable A or A', so add A'A andapply rule as follows: B' + C +
D' = B' + C + D' + A'A

= (A’ + B’ + C + D’)(A + B’ + C + D’)

The third term, A + B’ + C’ + D, is already in standard form. The standardPOS form of the original expression is as follows:

(A’ + B + C)(B’ + C + D’)(A + B’ + C’ + D) = (A’ + B + C + D’)(A’ + B + C +D) (A’ + B’ + C + D’)(A + B’ + C + D’) (A + B’ + C’ + D)

7.2 CANONICAL FORMS OF BOOLEAN EXPRESSIONS

With one variable x & x.

With two variables x y, x y, x y and x y.

With three variables x’ y’ z’, x’ y’ z, x’ y z’, x’ y z, x y’ z’, x y’ z, x y z’ & x y z.

These eight AND terms are called Minterms.

X	Y	Z	MINTERM	DESIGNATION
0	0	0	X’Y’Z’	m0
0	0	1	X’Y’Z	m1
0	1	0	X’YZ’	m2
0	1	1	X’YZ	m3
1	0	0	XY’Z’	m4
1	0	1	XY’Z	m5
1	1	0	XYZ’	m6
1	1	1	XYZ	m7

Maxterm is the complement of its corresponding minterm and vice versa

X	Y	Z	MAXTERMS	DESIGNATION
0	0	0	X+Y+Z	M0
0	0	1	X+Y+Z’	M1
0	1	0	X+Y’+Z	M2
0	1	1	X+Y’+Z’	M3
1	0	0	X’+Y+Z	M4
1	0	1	X’+Y+Z’	M5
1	1	0	X’+Y’+Z	M6
1	1	1	X’+Y’+Z’	M7

For example the function F (for minterms)

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

F = x’ y’ z + x y’ z’ + x y z F = m1 + m4 + m7

Any Boolean function can be expressed as a sum of minterms (sum of products **SOP**) or product of maxterms (product of sums **POS**).

For example the function F (for maxterms)

F’ = x’ y’ z’ + x’ y z’ + x’ y z + x y’ z + x y z’

The complement of F’ = (F’)’= F

F = (x + y + z) (x + y’ + z) (x + y’ + z’) (x’ + y + z’) (x’ + y’ + z)
F = M0M2M3M5M6

Example 1
Express the Boolean function F = A + B’C in a sum of minterms (SOP).
Solution
The term A is missing two variables because the domain of F is (A, B, C)
A = A(B + B’) = AB + AB’ because B + B’ = 1
BC missing A, so
B’C(A + A’) = ABC + A’B’C
AB(C + C’) = ABC + ABC’
AB’(C + C’) = AB’C + AB’C’
F = ABC + ABC’ + AB’C + AB’C’ + ABC + A’B’C
Because A + A = A
F = ABC + ABC’ + AB’C + AB’C’ + A’B’C
F = m7 + m6 + m5 + m4 + m1
In short notation
F(A, B, C) = Σ(1, 4, 5, 6, 7)
F’ (A, B, C) = Σ(0, 2, 3)

- **The complement of a function expressed as the sum of minterms equalto the sum of minterms missing from the original function.**

Truth table for F = A + B’C

	A	B	C	B’	B’C	F
0	0	0	0	1	0	0
1	0	0	1	1	1	1
2	0	1	0	0	0	0
3	0	1	1	0	0	0
4	1	0	0	1	0	1
5	1	0	1	1	1	1
6	1	1	0	0	0	1
7	1	1	1	0	0	1

Example 2
Express F = xy + x’z in a product of maxterms form.

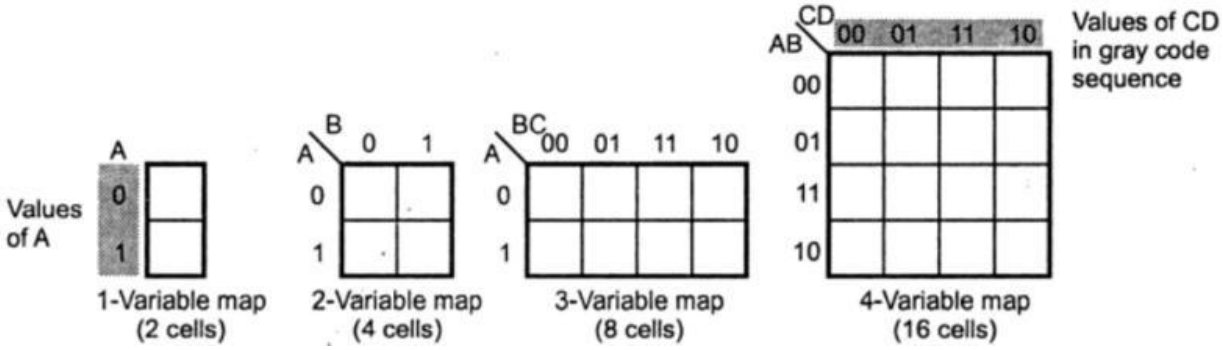
Solution
F = xy + x’z = (xy + x’)(xy + z) = (x + x’)(y + x’)(x + z)(y + z) remember x + x’ =
1
F = (y + x’)(x + z)(y + z)
F = (x’ + y + zz’)(x + yy’ + z)(xx’ + y + z)
F = (x ‘+ y + z)(x’ + y + z’)(x + y’ + z)(x + y +z)(x’ + y + z) F = (x’ + y + z)(x’ +
y + z’)(x + y + z)(x + y’ + z)
F = M4 M5M0M2 F(x, y, z) = Π(0, 2, 4, 5)
F(x, y, z) = Π(1, 3, 6, 7)
The complement of a function expressed as the product of maxterms equal to the product of maxterms missing from the original function.
To convert from one canonical form to another, interchange the symbols Σ,Π and list those numbers missing from the original form.

F = M4 M5M0M2 = m1 + m3 + m6 + m7
F(x, y, z) = Π(0, 2, 4, 5) = Σ(1, 3, 6, 7)

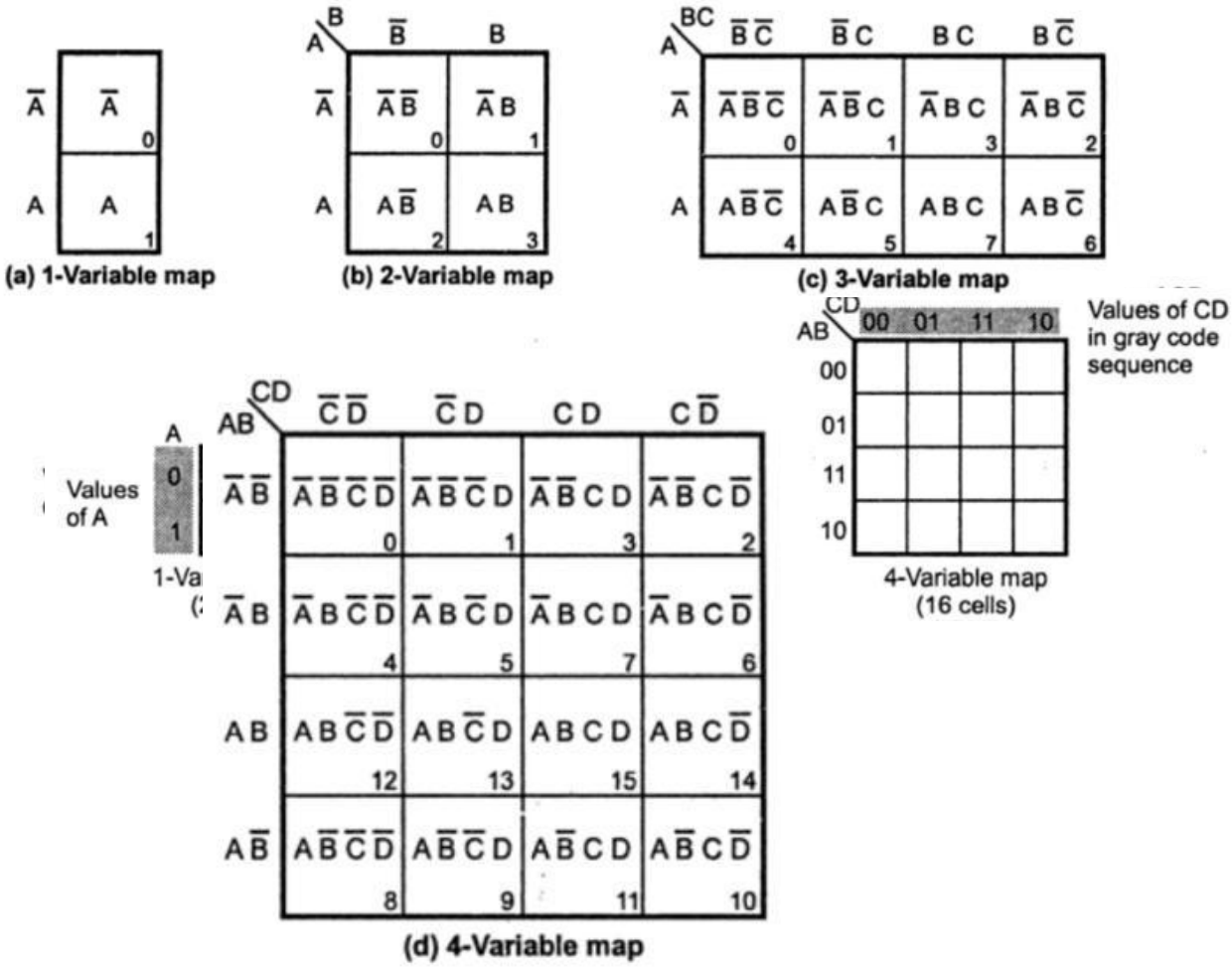
8. Karnaugh Map

Karnaugh map method gives us a systematic approach for simplifying a Boolean expression. Karnaugh map method was first proposed by Veitch and modified by Karnaugh, hence it is known as Karnaugh Map or K-map.

K-map contains boxes called cells. Each of the cell represents one of the 2^n possible products that can be formed from n variables. A two variable mzp contains 2^2 =4 cells, a three variable contains 2^3 =8 cells and four variable contains 2^4 =16 cells. The following figure shows the outline of 1, 2, 3 and 4 variable
maps.

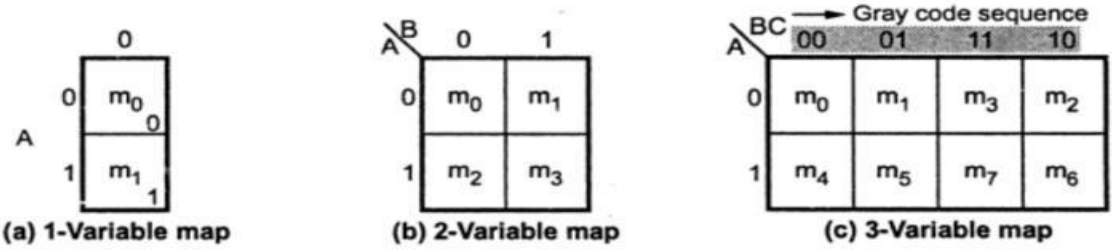


The product term(minterm) assigned to the cells of K-map by labelling each row and column is shown in 1, 2, 3 and 4 variable map and the product term(minterm) corresponding to each cell is shown in the below figure (a),(b),(c) and (d).

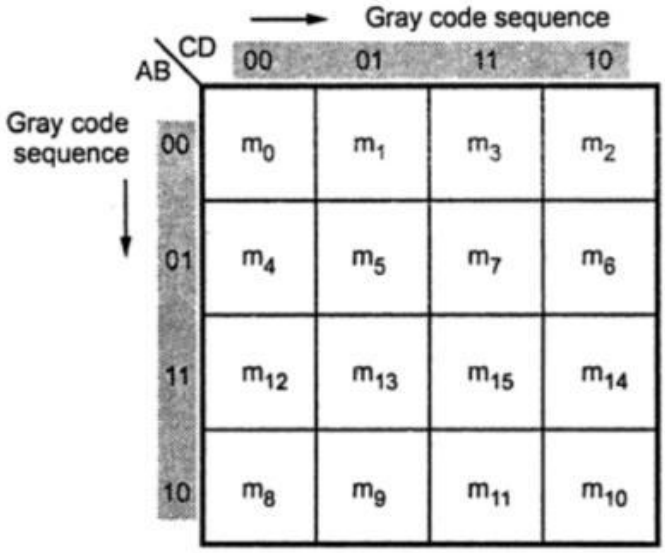


The labelling of the rows and columns of a 1, 2, 3 and 4 variable K-map using Gray code and the

product terms(minterm) corresponding to each cell is shown in the figure(a) (b) (c) and (d).

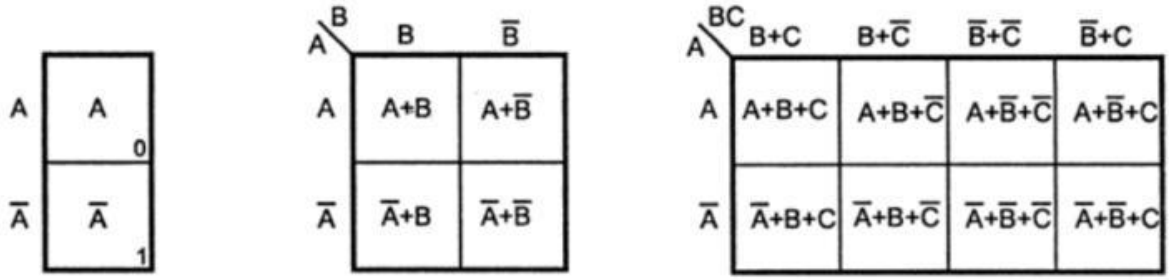


(a) 1-Variable map (b) 2-Variable map (c) 3-Variable map

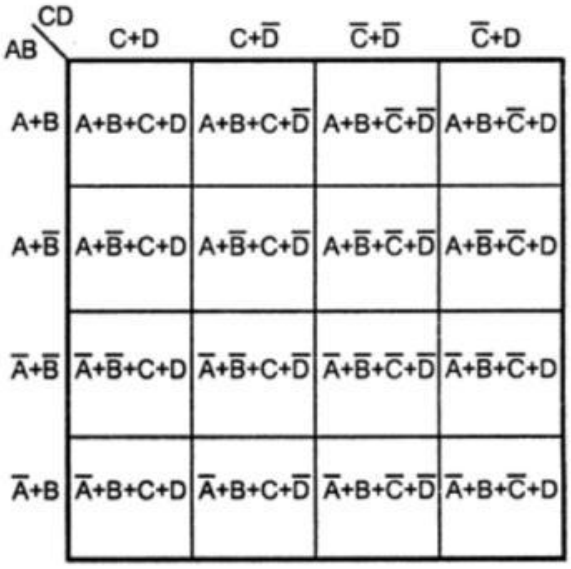


(d) 4-Variable map

The sum term(maxterm) assigned to the cells of K-map by labelling each row and column is shown in 1, 2, 3 and 4 variable map and the sum term(maxterm) corresponding to each cell is shown in the below figure (a),(b),(c) and (d).

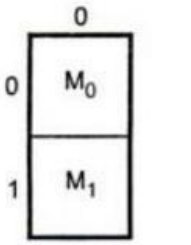


(a) 1-Variable map (b) 2-Variable map (c) 3-Variable map

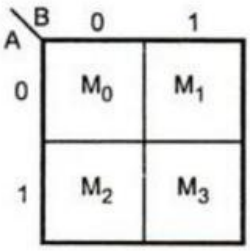


(d) 4-Variable map

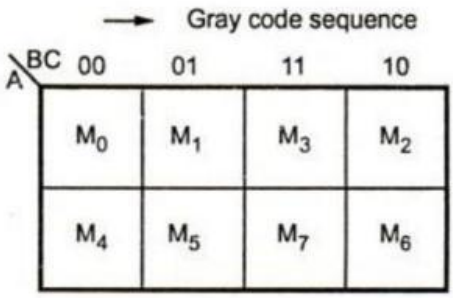
The labelling of the rows and columns of a 1, 2, 3 and 4 variable K-map using Gray code and the sum terms(maxterm) corresponding to each cell is shown in the figure(a) (b) (c) and (d)



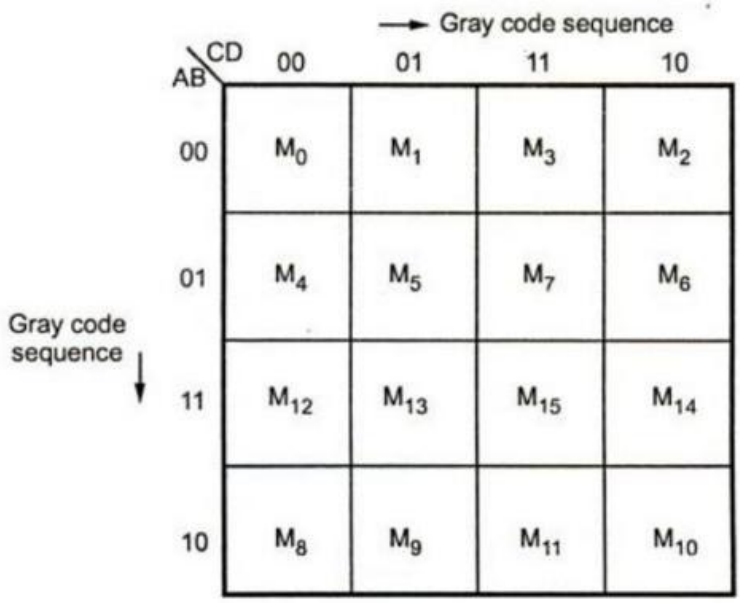
(a) 1-Variable map



(b) 2-Variable map



(c) 3-Variable map

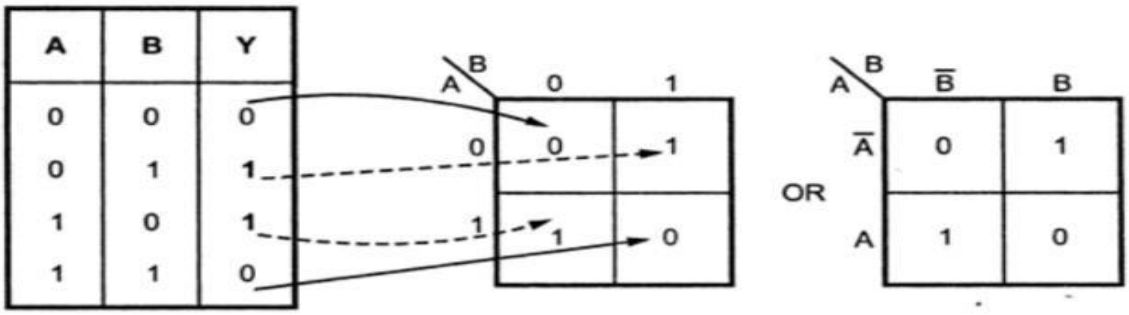


(d) 4-Variable map

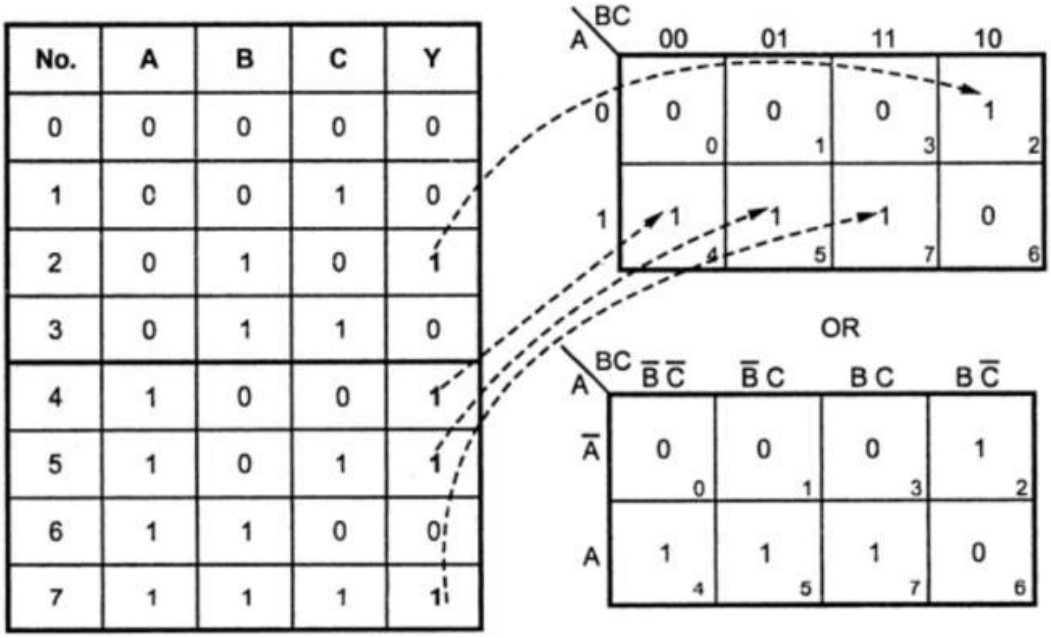
8.1 Plotting a Karnaugh Map

Representation of truth table on K-map

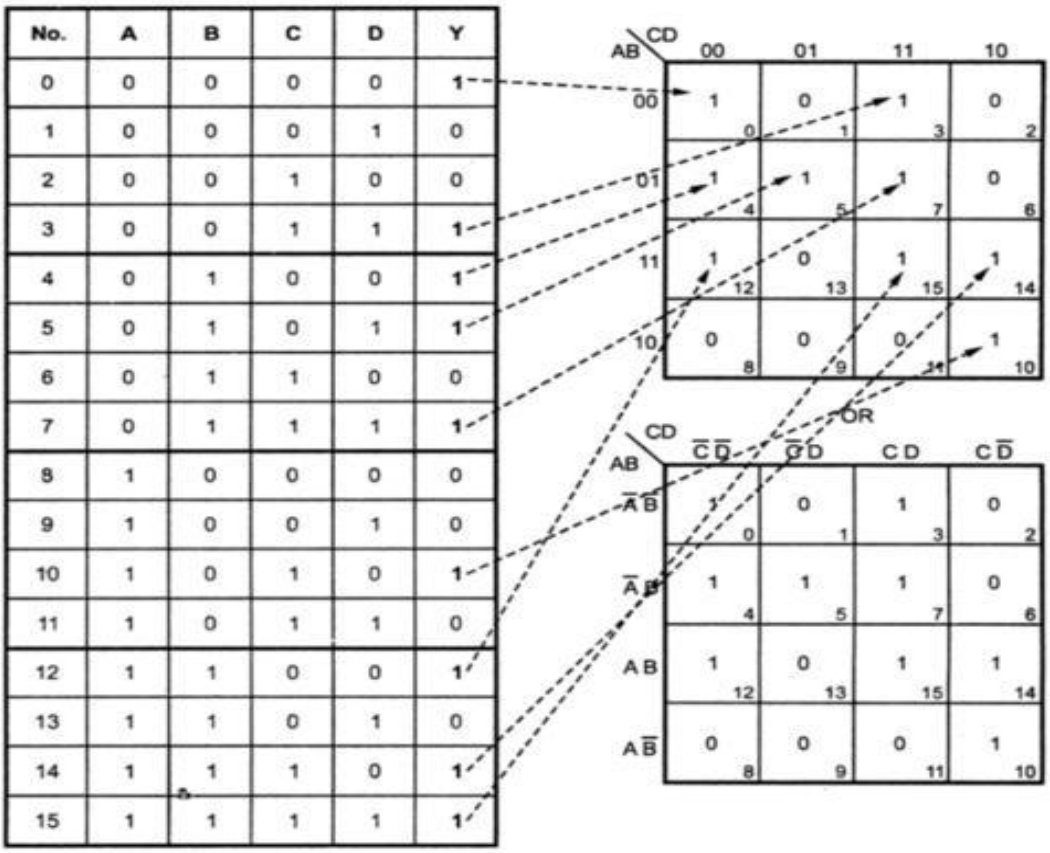
The representation of a two variable truth table on a Karnaugh map is shown below.



The representation of a three variable truth table on a Karnaugh map is shown below



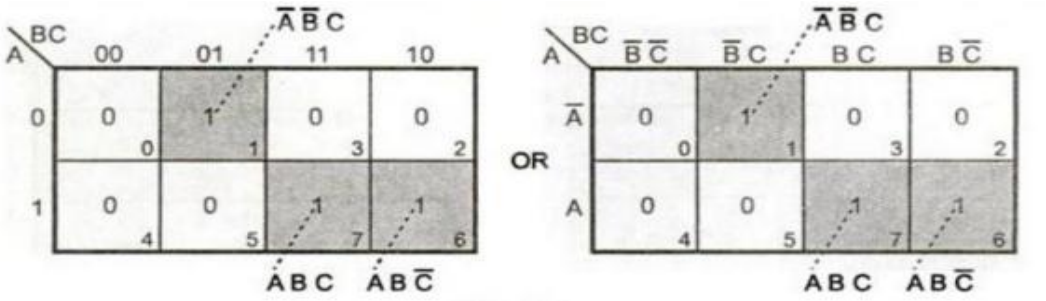
The representation of a four variable truth table on a Karnaugh map is shown below



Representation standard SOP on K-map

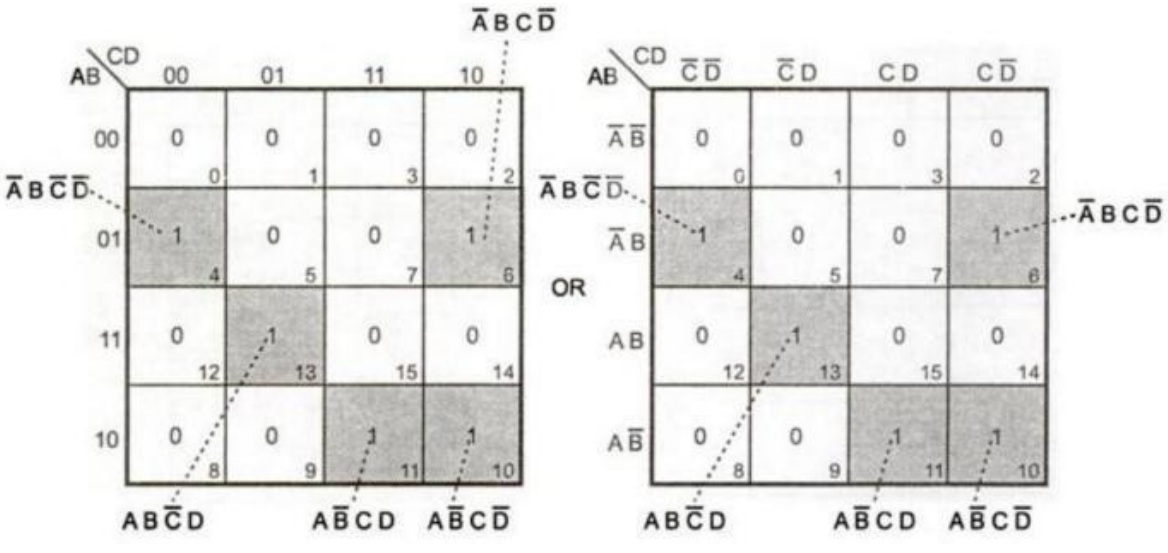
Example 1:

Plot Boolean expression Y=ABC' +ABC+A'B'C on the Karnaugh map



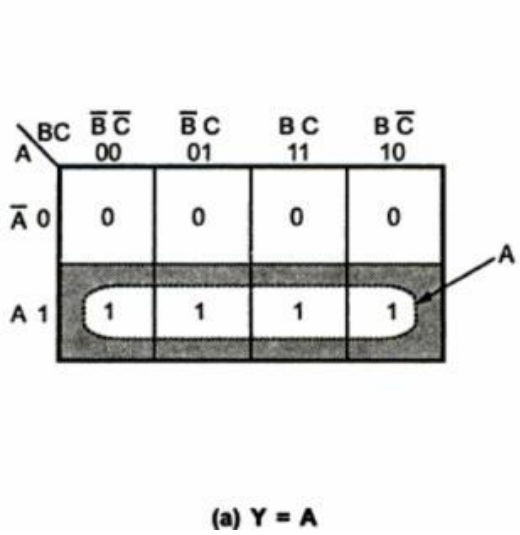
Example 2:

Plot Boolean expression Y=A'BC'D' + AB'CD +A'BCD +AB'CD+ABC'D on the karnaugh map.

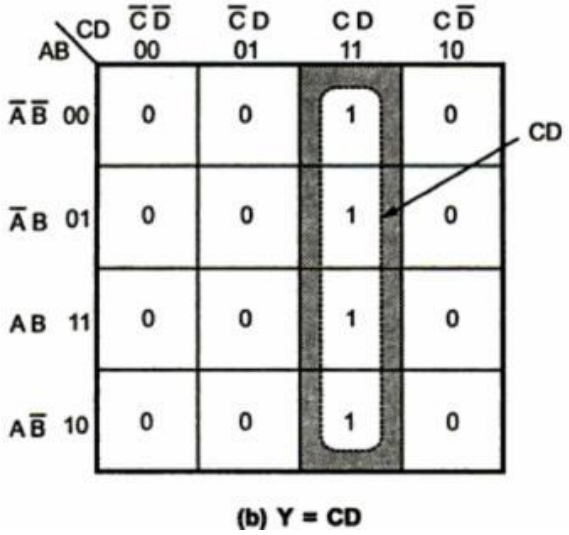


Grouping Cells for Simplification

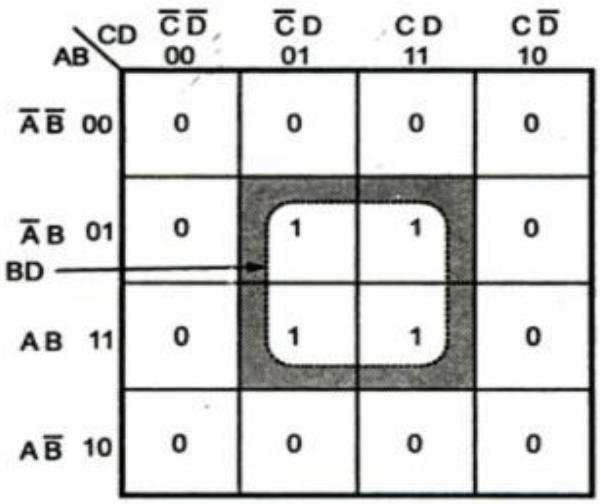
1. Grouping Two adjacent Pairs & Grouping Four adjacent ones (Quad)



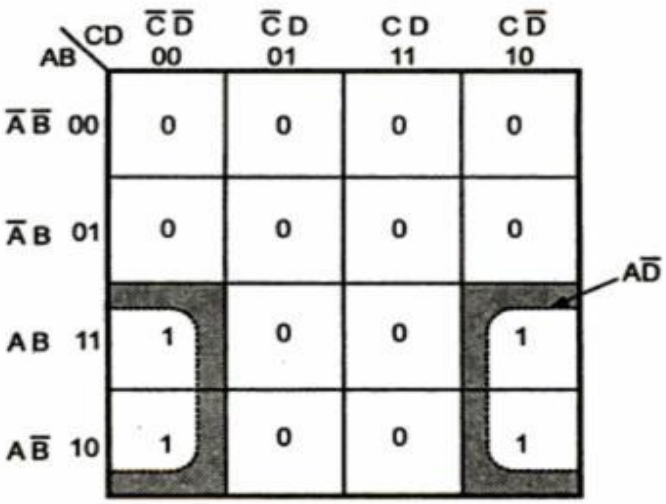
(a) Y = A



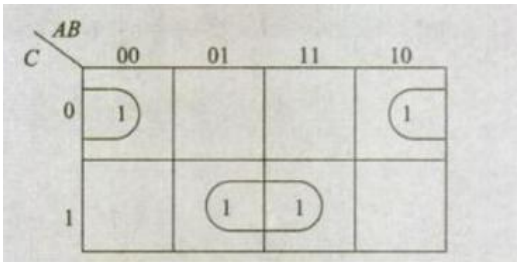
(b) Y = CD

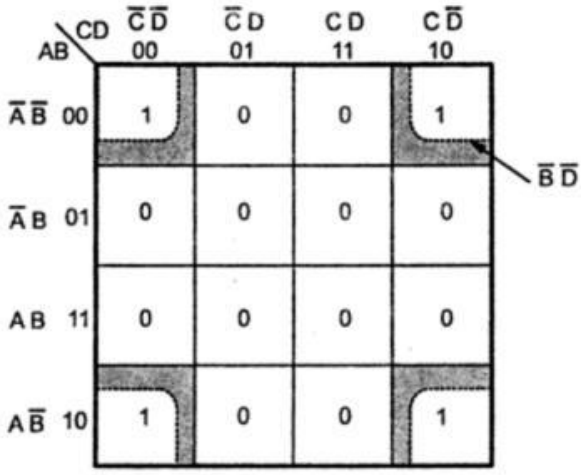


(c) Y = BD

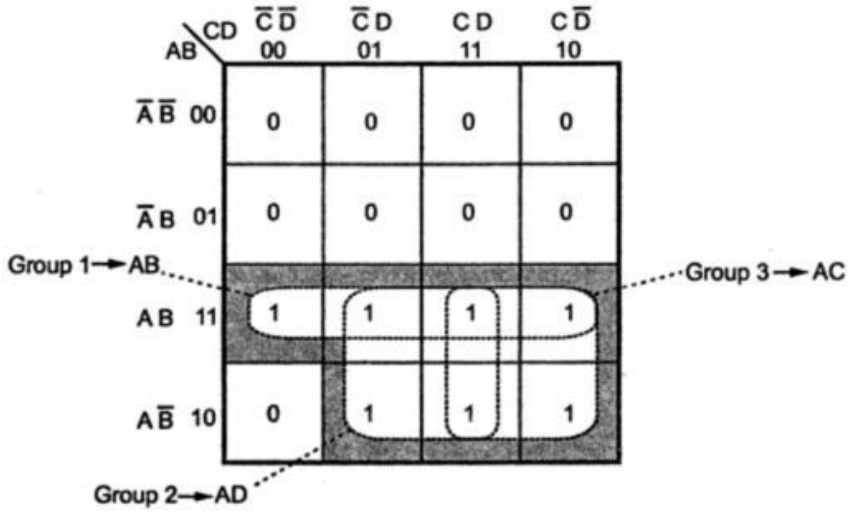


(d) Y = $A\overline{D}$



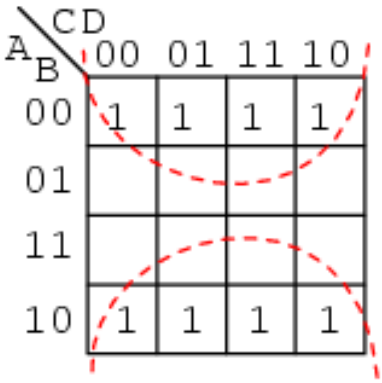


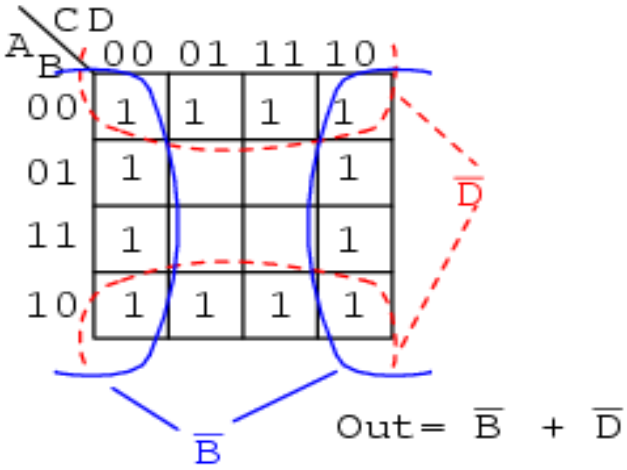
(e) $Y = \bar{B}\bar{D}$



(f) $Y = AB + AD + AC$

2. Grouping Eight adjacent ones (Octet)

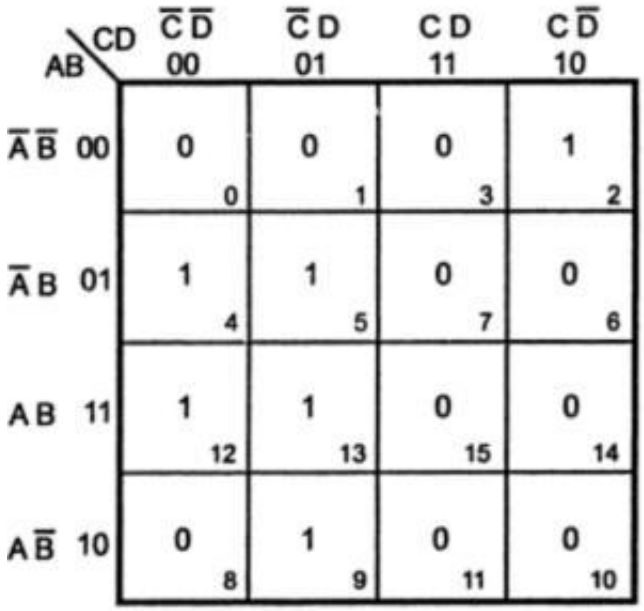


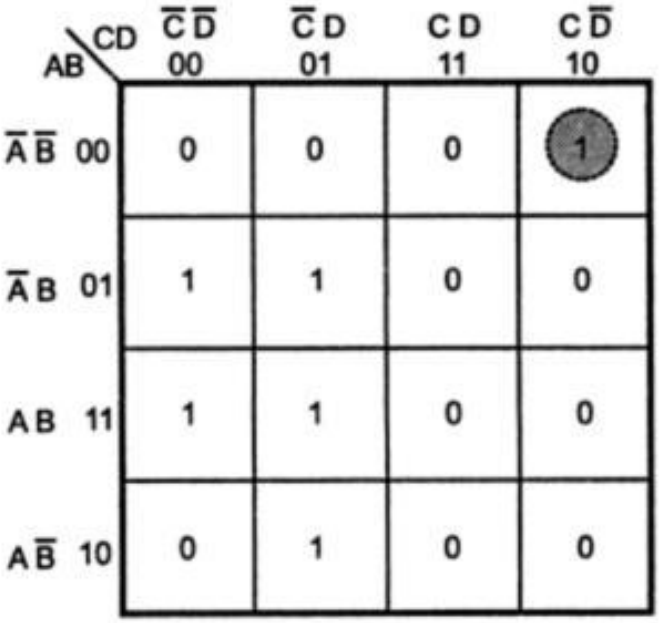


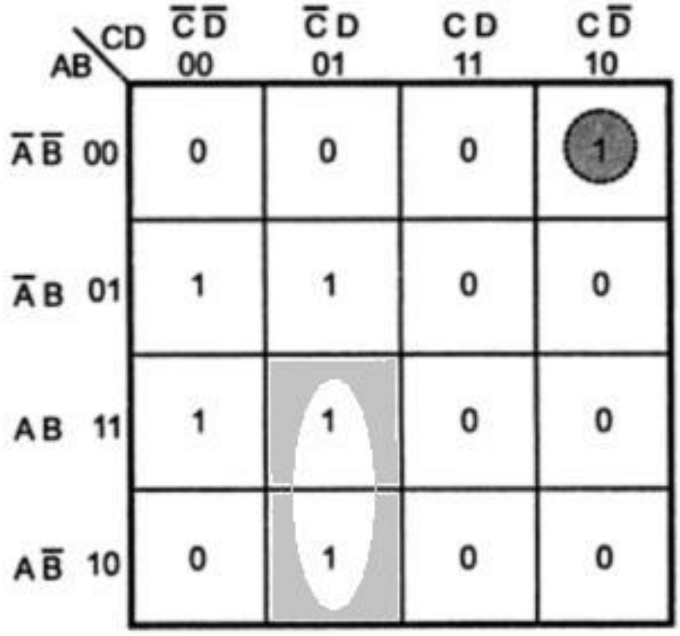
Simplification of Sum of Products Expression (SOP)

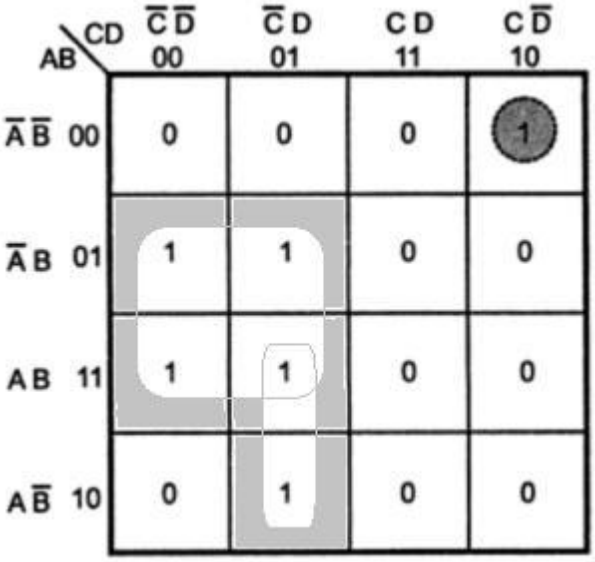
Example 1:

Minimize the Boolean expression $Y=A'BC'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD'$ on Karnaugh map




CD
AB
$\overline{C}\,\overline{D}$ 00
$\overline{C}\,D$ 01
$C\,D$ 11
$C\,\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00
$\overline{A}\,B$ 01
$A\,B$ 11
$A\,\overline{B}$ 10
0 0 0 1
1 1 0 0
1 1 0 0
0 1 0 0


CD
AB
$\overline{C}\,\overline{D}$ 00
$\overline{C}\,D$ 01
$C\,D$ 11
$C\,\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00
$\overline{A}\,B$ 01
$A\,B$ 11
$A\,\overline{B}$ 10
0 0 0 1
1 1 0 0
1 1 0 0
0 1 0 0

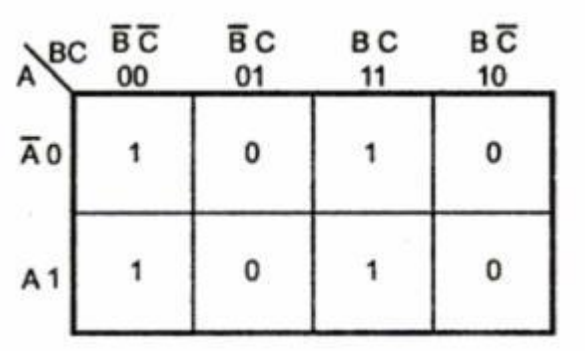


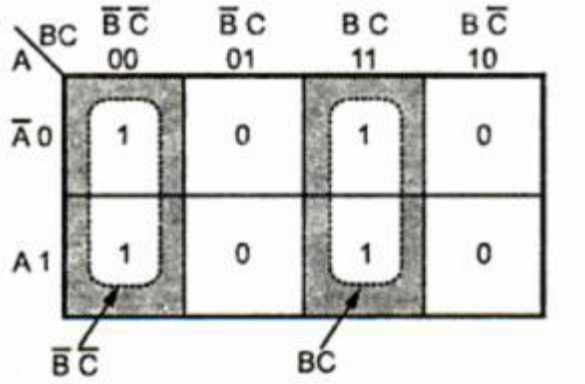
$Y=A'B'CD'+AC'D+BC'$

Example 2:

Simplify the logic function specified by the truth table using Karnaugh map method. Y is the output variable and A,B,C are the input variable

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1





$$Y=B'C'+BC$$

9. DON'T CARE CONDITIONS

- ***An output condition that can be regarded as either high or low***

The logical sum of the minterms associated with a Boolean function specifies the conditions under which the function is equal to 1. The function is equal to 0 for the rest of the minterms. This pair of conditions assumes that all the combinations of the values for the variables of the function are valid. In practice, in some applications the function is not specified for certain combinations of the variables. As an example, the four-bit binary code for the decimal digits has six combinations that are not used and consequently are considered to be unspecified. Functions that have unspecified outputs for some input combinations are called incompletely specified functions. In most applications, we simply don't care what value is assumed by the function for the unspecified minterms. For this reason, it is customary to call the unspecified minterm of a function don't care conditions. These don't care conditions can be used on a map to provide further simplification of the Boolean expression.

A don't care minterm is a combination of variables whose logical value is not specified. Such a minterm cannot be marked with a 1 in the map, because it would require that the function always be a 1 for such a combination. Likewise putting a 0 on the square requires the function to be 0. To distinguish don't care condition from 1's or the 0's an X is used. Thus an X inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to F for the particular minterm.

In choosing the adjacent squares to simplify the function in a map the don't care minterms may be assumed to be either 0 0r 1. When simplifying the function, we can choose to include each don't care minterm with either the 1's or the 0's depending on which combination gives the simplest expression.

Example Problem:

Simplify the Boolean function F(w,x,y,z) = ∑(1,3,7,11,15) which has the don't care conditions d(w,x,y,z) = ∑(0,2,5).

Solution

The minterms of F are the variable combinations that make the function equal to 1. The minterms of "d" are don't care minterms that may be assigned either 0 or 1. The map simplification is shown in fig. the minterms of F are marked by 1's. Those of d are marked by X's and remaining squares are filled with 0's.

To get simplified expression in sum-of- product form we must include all five 1's in the map but we may

yz / wx	00	01	10	11
00	X	1	1	X
01	0	X	1	0
10	0	0	1	0
11	0	0	1	0

In the part of the diagram, don't care minterm 0 and 2 is included the units 1's and the simplified function is now

F = yz+w'x'

yz / wx	00	01	10	11
00	X	1	1	X
01	0	X	1	0
10	0	0	1	0
11	0	0	1	0

In the second don't care minterm 5 is included with the 1's , and the simplified function is now

F = yz + w'z

9.1 NAND AND NOR IMPLIMENTATION

Digital circuits are frequently constructed with NAND and NOR gates rather than with AND and OR gates. NAND and NOR gates are easier to fabricate. So rules and procedures have been developed for the conversion from Boolean functions given in terms of AND, OR and NOT into equivalent NAND and NOR logic diagrams.

Two level NAND- NAND implementation

To facilitate the conversion to NAND logic, it is convenient to define an alternative graphic symbol for the gate. The alternate representation of NAND gate is shown in fig. according to De Morgan's theorem

Steps to be followed

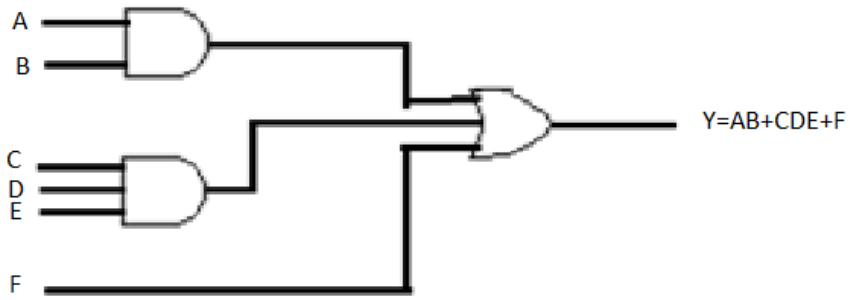
1. Simplify the given logic expression and convert it in the SOP form
2. Draw the logic circuit using AND,OR and NOT gate
3. Replace every AND gate by a NAND gate, Every OR gate by a bubbled OR gate and NOT gate by a NAND inverter.
4. Replace bubbled-OR gate by NAND gate.

Example Problem:

Implement the following Boolean equation using only NAND gatesY=AB+CDE+F

Solution

Step 1: realization using basic gates

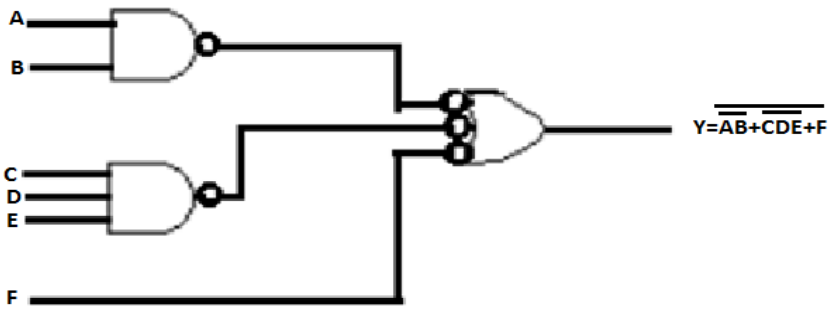


Step 2: replace

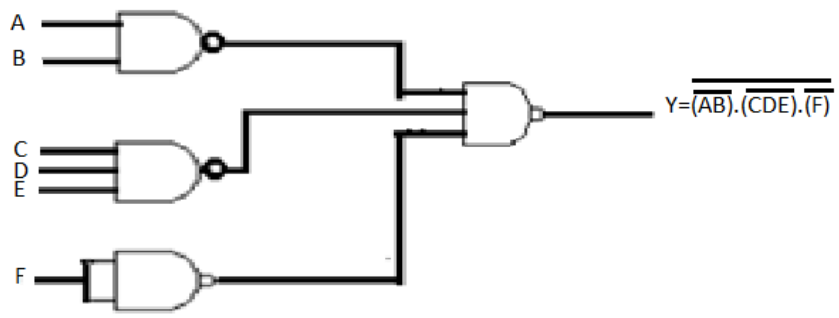
AND →NAND

OR →bubbled – OR

NOT →NAND inverter



Step 3: draw the logic circuit using only NAND gates



9.2 Multilevel NAND circuits

The standard form of expressing Boolean function results in a two-level implementation. If has digital system three or more levels then the most common procedure in the design of multilevel circuits is to express the Boolean function in terms of AND, OR and compliments operations.

The general procedure for converting multilevel AND – OR logic diagram into an all NAND logic diagram is as follows

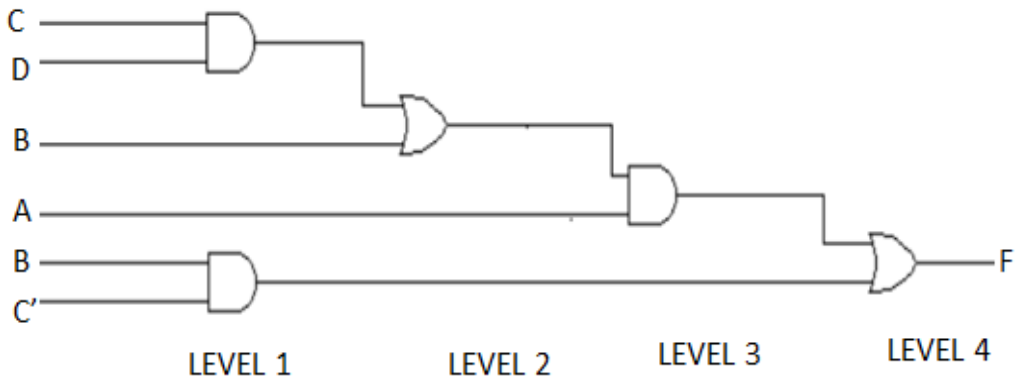
1. Convert all AND gates to NAND gates with AND – invert graphic symbols
2. Convert all OR gates to NAND gates with invert –OR graphic symbol.
3. Check all the bubbles in the diagram. For every bubble that is not compensated by other small circle along the same line insert an inverter or compliment the input literal.

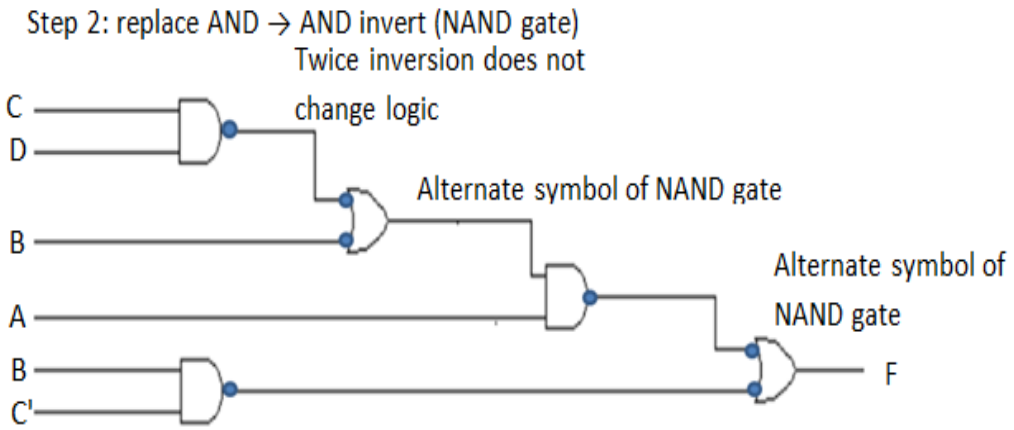
Example Problem:

Implement the following Boolean expression using NAND gates onlyF=A(CD+B)+BC

Solution:

Step 1: Draw logic diagram using AND,OR and NOT gate as shown in the fig.





9.3 NOR IMPLEMENTATION

The NOR operation is the dual of the AND operation. Therefore all procedures and rules for NOR logic are the dual for the corresponding procedures and rules developed for NAND logic. The NOR gate is another universal gate that can be used to implement any Boolean function. The alternative representation of NOR gate according to demorgan's theorem is shown below.

Steps to be followed

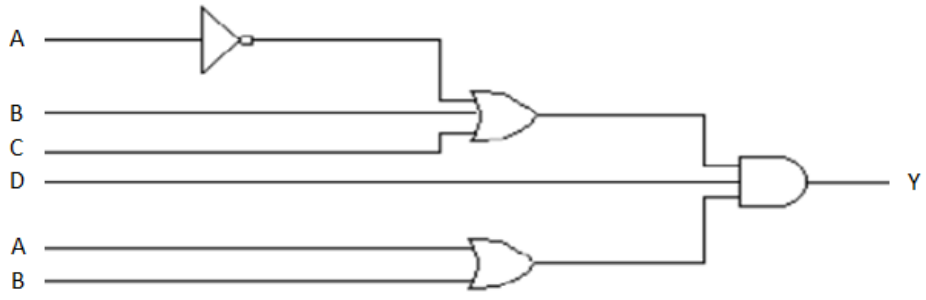
1. Simply the given logic expression and convert it into product of sum (POS) form.
2. Draw the AND – OR-NOT realization.
3. Replace every OR gate by NOR, every AND gate by a bubbled AND gate and ever inerter by a NOR inverter.
4. Draw the final circuit using only the NOR gates.

Example Problem:

Implement the following function by using NOR gates Y=(A'+B+C)(A+B)D

Solution:

Step 1: Implement the given Boolean function by usingAND, ORandNOTgate as shown below.

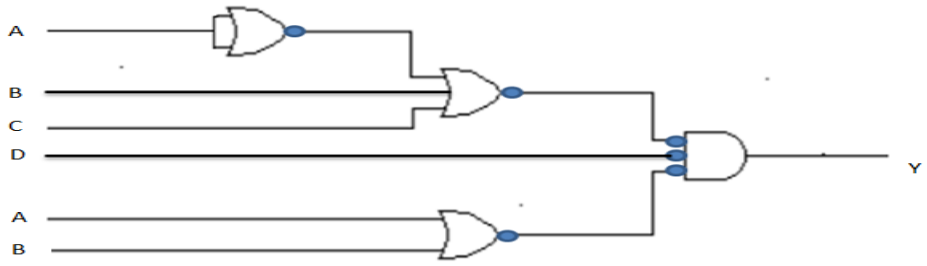


Step 2:

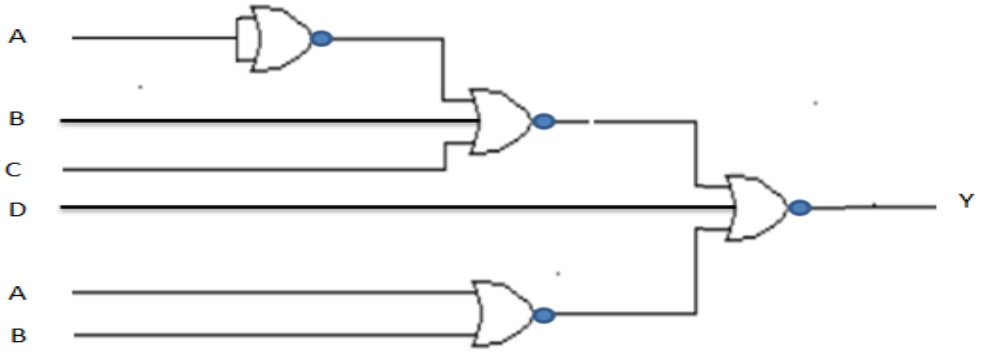
Replace OR → NOR

AND → invert AND

NOT → NOR invert



Step 3: Replace invert AND gate by NOR gate shown in fig.



9.4 MULTILEVEL NOR IMPLEMENTATION

The procedure for converting a multilevel AND-OR diagram to an all NOR diagram is similar to multilevel NAND implements. The following steps are followed for multilevel-NOR implementation

Step 1.impliment the logic function using AND, OR and NOT gate.

Step 2.convert all AND gates to NOR gates with invert-AND graphic symbol.

Step 3.convert all OR gates with OR invert graphic symbols.

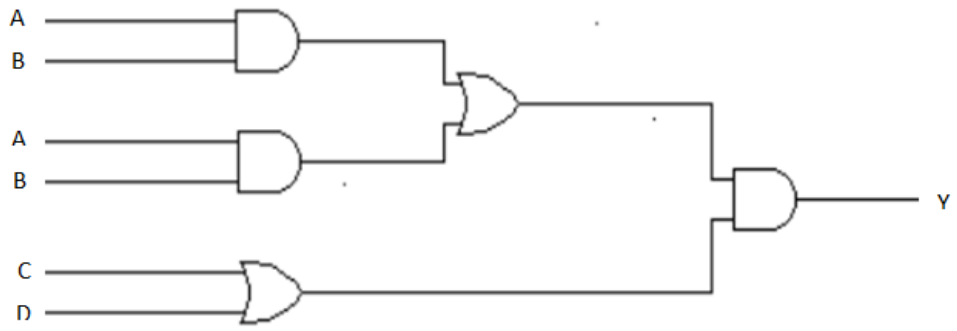
Step 4.Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter or compliment the input literal.

Example Problem:

Implement the following Boolean function using NOR gatesY=(AB'+A'B)(C+D')

Solution

Step 1: Implement the Boolean function using AND,OR and NOT gate as shown in fig.

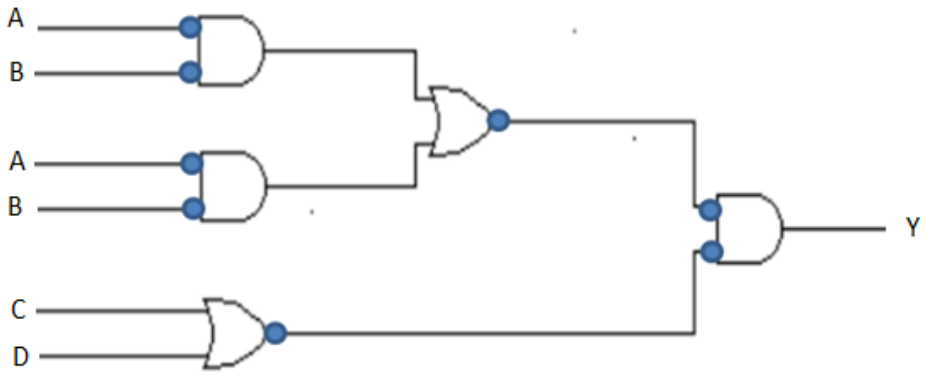


Step 2:

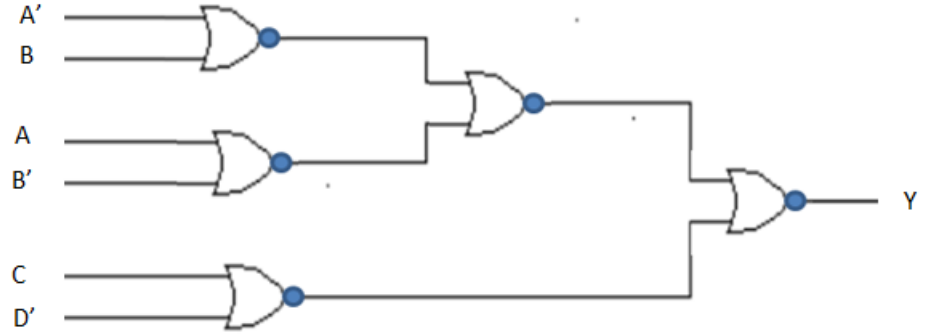
Replace

AND → invert-AND symbol

OR → NOR gate



Step 3: Check each line has even number of bubbles. If any line does not have even number of bubbles the insert bubble (i.e. input **A, B',A', B**has odd number of bubbles. Therefore apply the inverted inputs to make even numbers of bubbles)



10. QUINE-MC CLUSKEY (OR) TABULATION METHOD

11.

Definition: It is used to simplify the Boolean expression for more variables.

The map methodf implication is convenient method as long as the numbers of variables do not exceed fivevariables. If the number of variable increases, it is difficult to make the simplification of expression. F the number of variables increases it is difficult to make the simplification of expression. To avoid this complex and to meet this need W.V. Quine and E.J. McCluskey developed an exact tabulation method to simplify the Boolean expression. This method is called as tabulation method or Quine McCluskey method.

The summary steps are as follows to simplify the Boolean expression.

Step 1. List all minterms in the binary form.

Step 2. Separate the number of groups according to the number of 1's.

Step 3. Compare each binary number with every group in the adjacent next highest category group and they differ only one bit position. Put check mark if comparison is possible (-) and copy

remaining term in the next column. Put (√) mark for every comparison.The essential prime implicants are identified if they have no tick mark.

Step 4. Apply the same process described in step 3 for the resultant column and continue the cycles until a single pass through cycle yieldsfurther elimination of literals.

Step 5. From prime implicant chart

(a) The prime implicants should be represented in rowsand each minterm of the function in a column.
(b) Crosses (X) should be placed in each row to show the composition of minterm that makes the prime implicants.
(c) A completed prime implicants table should be inspected for columns containing only a single cross in their columns are called essential prime implicants.

Step 6. Getting the simplified expression after the above step

Example Problem: Simplify the Boolean function by using tabulation method.

$F(a,b,c,d)=\sum m(0,1,2,5,6,7,8,9,10,14)$

Solution

Group	Column I		Column II		Column III
	abcd		abcd		abcd
Group 0	0	0000√	0,1	000√	0,1,8,9-00-
Group 1 Number of 1's one	1	0001√	0,2	00-0√	0,2,8,10-0-0
	2	0010√	0,8	-000√	0,8,1,-00-
	8	1000√	1,5	0-01√	0,8,2,10-0-0
Group 2 Number of 1's two	5	0101√	1,9	-001√	2,6,10,14-10
	6	0110√	2,6	0-10√	2,10,6,14-10
	9	1001√	2,10	-0-10√	
	10	1010√	8,9	100√	
Group 3 Number of 1's three	7	0111√	8,10	10-0√	
	14	1110√	5,7	01-1	
			6,7	011-	
			6,14	=110√	
			10,14	1-10√	

Table: Prime implicant table

Prime implicants	minterms										
		0	1	2	5	6	7	8	9	10	14
1,5	d'c'd		X		X						
5,7	a'bd				X		X				
6,7	a'bc					X	X				
0,1,8,9*	b'c'	X	X					X	X		
0,2,8,10	b'd'	X		X				X		X	
2,10,6,14*	cd'			X		X				X	X
									√		√

Note that the cells (5,7), (1,5), (6,7),(0,1,8,9),(0,2,8,10) and (2,10,6,14)are prime implicants. The prime implicants table can be plotted as shown in table above. All the unticked terms in the above simplification are given as prime implicant of this Boolean expression, these prime implicants chart is

shown in table. In the chart all the specified implicants form columns a cross is put in the row of each prime implicant under the columns of the implicants which it covers.

A tick mark is placed against every essential prime implicant (which column contains a single cross(X). the sum of essential prime implicantsF = b'c' + cd'

The prime implicants which covers the minterms 0,1,8,9 and 2,10,6,14 therefore in order to cover the remaining minterms, the reduced prime implicants chart is formed as follows.

To cover the minterms the prime implicants (6,7) and (0,2,8,10) can be selected in additionto the essential prime implicants for obtaining the minimal Boolean expression is given

F = b'c'+cd'+a'bc+b'd'

Reduced prime implicant table

Prime implicants	Minterms									
		0	1	2	5	6	7	8	9	10	14
1,5	a'c'd		X		X						
5,7	a'bd				X		X				
6,7	a'bc					X	X				
0,2,8,10*	b'd'	X		X				X		X	
		√		√		√		√		√	