

EC 8392 – DIGITAL ELECTRONICS

UNIT – I : DIGITAL FUNDAMENTALS

INTRODUCTION:

In 1854, George Boole, an English mathematician, proposed algebra for symbolically representing problems in logic so that they may be analyzed mathematically. The mathematical systems founded upon the work of Boole are called **Boolean algebra** in his honor.

The application of a Boolean algebra to certain engineering problems was introduced in 1938 by C.E. Shannon.

For the formal definition of Boolean algebra, we shall employ the postulates formulated by E.V. Huntington in 1904.

FUNDAMENTAL POSTULATES OF BOOLEAN ALGEBRA:

The postulates of a mathematical system forms the basic assumption from which it is possible to deduce the theorems, laws and properties of the system.

The most common postulates used to formulate various structures are

i) **Closure:**

A set S is closed w.r.t. a binary operator, if for every pair of elements of S , the binary operator specifies a rule for obtaining a unique element of S .

The result of each operation with operator (+) or (.) is either 1 or 0 and $1, 0 \in B$.

ii) **Identity element:**

A set S is said to have an identity element w.r.t a binary operation $*$ on S , if there exists an element $e \in S$ with the property,

$$e * x = x * e = x$$

Eg: $0 + 0 = 0$ $0 + 1 = 1 + 0 = 1$ a) $x + 0 = x$

$1 \cdot 1 = 1$ $1 \cdot 0 = 0 \cdot 1 = 0$ b) $x \cdot 1 = x$

iii) **Commutative law:**

binary operator * on a set S is said to be commutative if,

$$x * y = y * x \quad \text{for all } x, y \in S$$

E $0 + 1 = 1 + 0 = 1$ a) $x + y = y + x$

g: $0 \cdot 1 = 1 \cdot 0 = 0$ b) $x \cdot y = y \cdot x$

iv) **Distributive law:**

If * and • are two binary operation on a set S, • is said to be distributive over

+ whenever,

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

Similarly, + is said to be distributive over • whenever,

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

v) **Inverse:**

A set S having the identity element e, w.r.t. binary operator * is said to have an inverse, whenever for every $x \in S$, there exists an element $x' \in S$ such that,

$$x \cdot x' \in e$$

a) $x + x' = 1$, since $0 + 0' = 0 + 1$ and $1 + 1' = 1 + 0 = 1$

b) $x \cdot x' = 0$, since $0 \cdot 0' = 0 \cdot 1$ and $1 \cdot 1' = 1 \cdot 0 = 0$

Summary:

Table: 1.1 - Postulates of Boolean algebra:

POSTULATES	(a)	(b)
Postulate 2 (Identity)	$x + 0 = x$	$x \cdot 1 = x$
Postulate 3 (Commutative)	$x + y = y + x$	$x \cdot y = y \cdot x$
Postulate 4 (Distributive)	$x(y + z) = xy + xz$	$x + yz = (x + y) \cdot (x + z)$
Postulate 5 (Inverse)	$x + x' = 1$	$x \cdot x' = 0$

Basic theorem and properties of Boolean algebra:

Basic Theorems:

The theorems, like the postulates are listed in pairs; each relation is the dual of the one paired with it. The postulates are basic axioms of the algebraic structure and need no proof. The theorems must be proven from the postulates. The proofs of the theorems with one variable are presented below. At the right is listed the number of the postulate that justifies each step of the proof.

1) a) $x + x = x$

$$\begin{aligned}
 x + x &= (x + x) \cdot 1 && \text{by postulate 2(b) [} x \cdot 1 = x \text{]} \\
 &= (x + x) \cdot (x + x') && \text{5(a) [} x + x' = 1 \text{]} \\
 &= x + xx' && \text{4(b) [} x + yz = (x + y)(x + z) \text{]}
 \end{aligned}$$

$$= x + 0 \quad \text{-----} \quad 5(b) [x \cdot x' = 0]$$

$$= x \quad \text{-----} \quad 2(a) [x + 0 = x]$$

b) $x \cdot x = x$

$$x \cdot x = (x \cdot x) + 0 \quad \text{-----} \quad \text{by postulate } 2(a)$$

$$[x + 0 = x]$$

$$= (x \cdot x) + (x \cdot x') \quad \text{-----} \quad 5(b) [x \cdot x' = 0]$$

$$= x (x + x') \quad \text{-----} \quad 4(a) [x (y+z) = (xy) + (xz)]$$

$$= x (1) \quad \text{-----} \quad 5(a) [x + x' = 1]$$

$$= x \quad \text{-----} \quad 2(b) [x \cdot 1 = x]$$

2) a) $x + 1 = 1$

$$x + 1 = 1 \cdot (x + 1) \quad \text{-----} \quad \text{by postulate } 2(b) [x \cdot 1 = x]$$

$$= (x + x') \cdot (x + 1) \quad \text{-----} \quad 5(a) [x + x' = 1]$$

$$= x + x' \cdot 1 \quad \text{-----} \quad 4(b) [x + yz = (x+y)(x+z)]$$

$$= x + x' \quad \text{-----} \quad 2(b) [x \cdot 1 = x]$$

$$= 1 \quad \text{-----} \quad 5(a) [x + x' = 1]$$

b) $x \cdot 0 = 0$

3) $(x')' = x$

From postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$, which defines the complement of x . The complement of x' is x and is also $(x')'$. Therefore, since the complement is unique,

$$(x')' = x.$$

4) Absorption Theorem:

a) $x + xy = x$

$$\begin{aligned}
 x + xy &= x \cdot 1 + xy && \text{----- by postulate 2(b) } [x \cdot 1 = x] \\
 &= x(1 + y) && \text{----- 4(a) } [x(y+z) = (xy) + (xz)] \\
 &= x(1) && \text{----- by theorem 2(a) } [x + 1 = x] \\
 &= x. && \text{----- by postulate 2(a) } [x \cdot 1 = x]
 \end{aligned}$$

b) **$x \cdot (x + y) = x$**

$$\begin{aligned}
 x \cdot (x + y) &= x \cdot x + x \cdot y && \text{----- 4(a) } [x(y+z) = (xy) + (xz)] \\
 &= x + x \cdot y && \text{----- by theorem 1(b) } [x \cdot x = x] \\
 &= x. && \text{----- by theorem 4(a) } [x + xy = x]
 \end{aligned}$$

c) **$x + x'y = x + y$** $x + x'y = x + xy + x'y$ ----- by theorem 4(a) $[x + xy = x]$

$$= x + y(x + x') \text{ ----- by postulate 4(a) } [x(y+z) = (xy) + (xz)]$$

$$= x + y(1) \text{ ----- 5(a) } [x + x' = 1]$$

$$= x + y \text{ ----- 2(b) } [x \cdot 1 = x]$$

d) **$x \cdot (x' + y) = xy$**

$$x \cdot (x' + y) = x \cdot x' + xy \text{ ----- by postulate 4(a) } [x(y+z) = (xy) + (xz)]$$

$$= 0 + xy \text{ ----- 5(b) } [x \cdot x' = 0]$$

$$= xy. \text{ ----- 2(a) } [x + 0 = x]$$

Properties of Boolean algebra:

1. Commutative property:

Boolean addition is commutative, given by

$$\mathbf{x + y = y + x}$$

According to this property, the order of the OR operation conducted on the variables makes no difference.

Boolean algebra is also commutative over multiplication given by,

$$x \cdot y = y \cdot x$$

This means that the order of the AND operation conducted on the variables makes no difference.

2. **Associative property:**

The associative property of addition is given by,

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

The OR operation of several variables results in the same, regardless of the grouping of the variables.

The associative law of multiplication is given by,

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

It makes no difference in what order the variables are grouped during the AND operation of several variables.

3. **Distributive property:**

The Boolean addition is distributive over Boolean multiplication, given by

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

The Boolean multiplication is distributive over Boolean addition, given by

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

4. Duality:

It states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.

If the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

$$\mathbf{x + x' = 1 \text{ is } \mathbf{x \cdot x' = 0}}$$

Duality is a very important property of Boolean algebra.

Summary:

Table: 1.2 - Theorems of Boolean algebra:

	THEOREMS	(a)	(b)
1	Idempotent	$\mathbf{x + x = x}$	$\mathbf{x \cdot x = x}$
		$\mathbf{x + 1 = 1}$	$\mathbf{x \cdot 0 = 0}$
2	Involution	$\mathbf{(x')' = x}$	
3	Absorption	$\mathbf{x + xy = x}$	$\mathbf{x (x + y) = x}$
		$\mathbf{x + x'y = x + y}$	$\mathbf{x \cdot (x' + y) = xy}$
4	Associative	$\mathbf{x + (y + z) = (x + y) + z}$	$\mathbf{x (yz) = (xy) z}$
5	DeMorgan's Theorem	$\mathbf{(x + y)' = x' \cdot y'}$	$\mathbf{(x \cdot y)' = x' + y'}$

DeMorgan's Theorems:

Two theorems that are an important part of Boolean algebra were proposed by DeMorgan.

The first theorem states that the complement of a product is equal to the sum of the complements.

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

The second theorem states that the complement of a sum is equal to the product of the complements.

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

Consensus Theorem:

In simplification of Boolean expression, an expression of the form $AB + A'C + BC$, the term BC is redundant and can be eliminated to form the equivalent expression $AB +$

$A'C$. The theorem used for this simplification is known as consensus theorem and is stated as,

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

The dual form of consensus theorem is stated as,

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

EC 8392 – DIGITAL ELECTRONICS

UNIT – I : DIGITAL FUNDAMENTALS

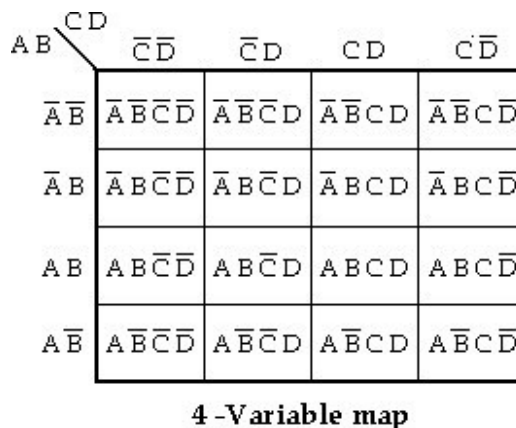
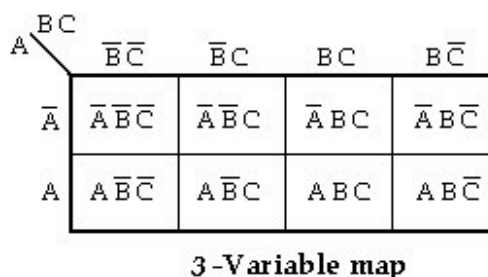
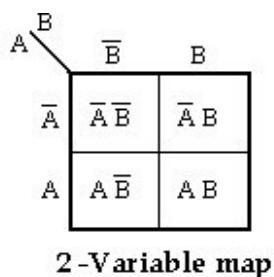
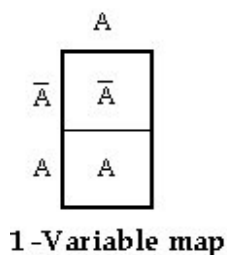
KARNAUGH MAP MINIMIZATION:

The simplification of the functions using Boolean laws and theorems becomes complex with the increase in the number of variables and terms. The map method, first proposed by Veitch and slightly improvised by Karnaugh, provides a simple, straightforward procedure for the simplification of Boolean functions. The method is called **Veitch diagram** or **Karnaugh map**, which may be regarded as a pictorial representation of a truth table.

The Karnaugh map technique provides a systematic method for simplifying and manipulation of Boolean expressions. A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized. For n variables on a Karnaugh map there are 2^n numbers of squares. Each square or cell represents one of the minterms. It can be drawn directly from either minterm (sum-of-products) or maxterm (product-of-sums) Boolean expressions.

Two- Variable, Three Variable and Four Variable Maps

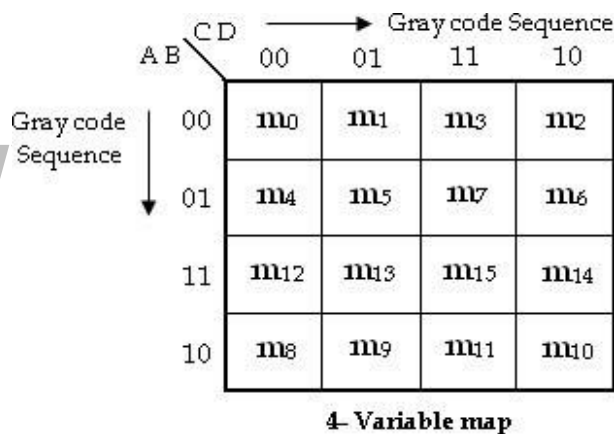
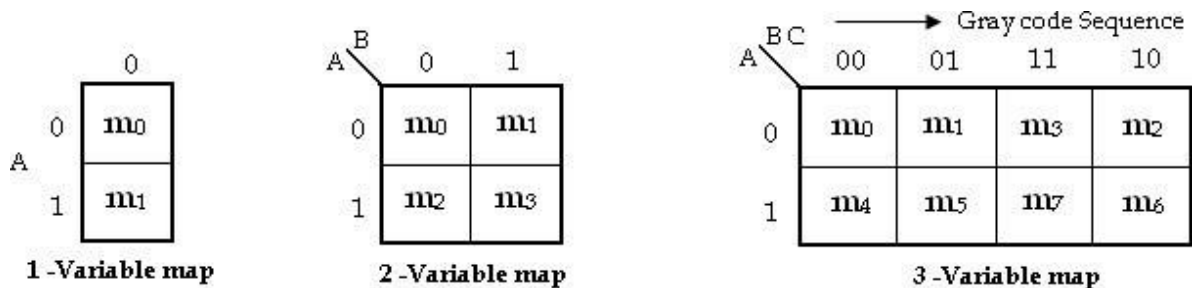
Karnaugh maps can be used for expressions with two, three, four and five variables. The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is $2^4 = 16$.



Product terms are assigned to the cells of a K-map by labeling each row and each column of a map with a variable, with its complement or with a combination of variables & complements. The below figure shows the way to label the rows & columns of a 1, 2, 3 and 4- variable maps and the product terms corresponding to each cell.

It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complement or to an uncomplemented form). Irrespective of number of variables the labels along each row and column must

conform to a single change. Hence gray code is used to label the rows and columns of K-map as shown ow.

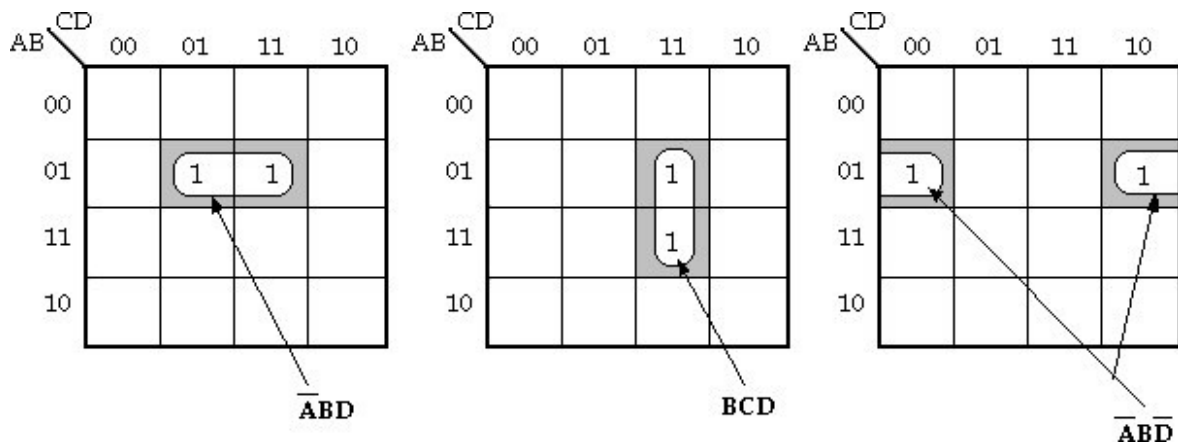


Grouping cells for Simplification:

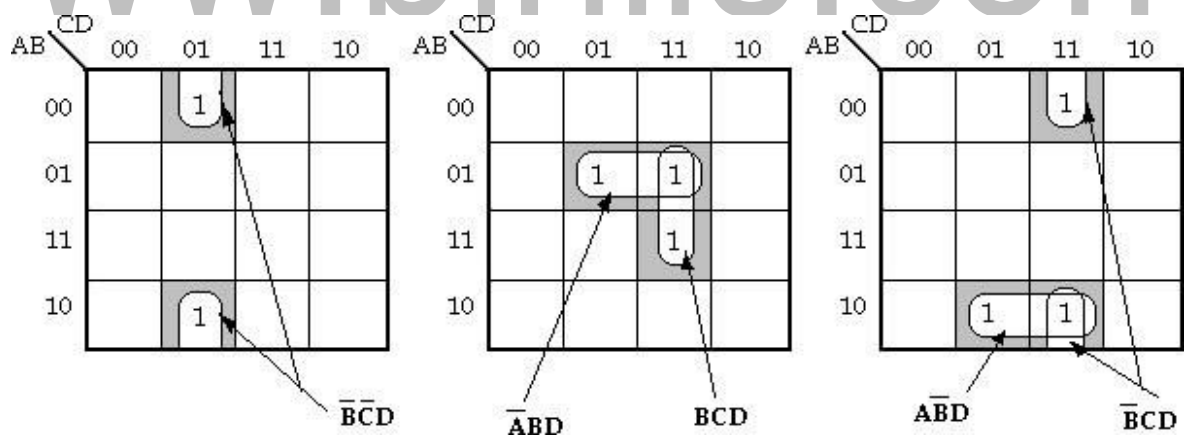
The grouping is nothing but combining terms in adjacent cells. The simplification is achieved by grouping adjacent 1's or 0's in groups of 2^i , where $i = 1, 2, \dots, n$ and n is the number of variables. When adjacent 1's are grouped then we get result in the sum of product form; otherwise we get result in the product of sum form.

Grouping Two Adjacent 1's: (Pair)

In a Karnaugh map we can group two adjacent 1's. The resultant group is called Pair.



www.binils.com

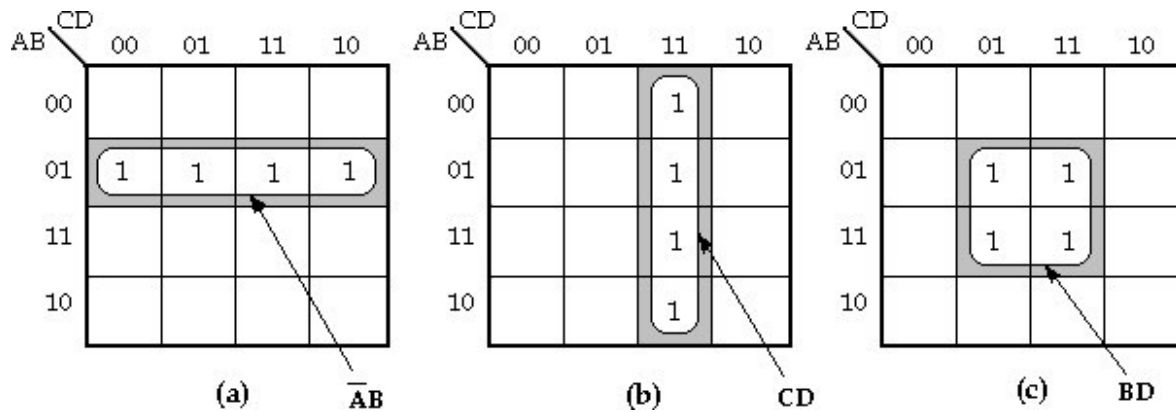


Examples of Pairs

Grouping Four Adjacent 1's: (Quad)

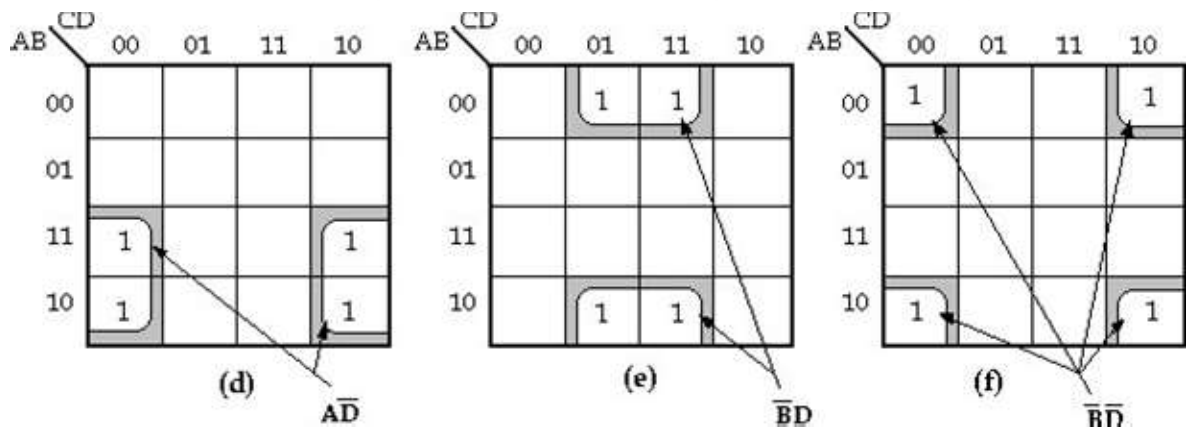
In a Karnaugh map we can group four adjacent 1's. The resultant group is called Quad. Fig (a) shows the four 1's are horizontally adjacent and Fig (b) shows

they are vertically adjacent. Fig (c) contains four 1's in a square, and they are considered adjacent to each other.



Examples of Quads

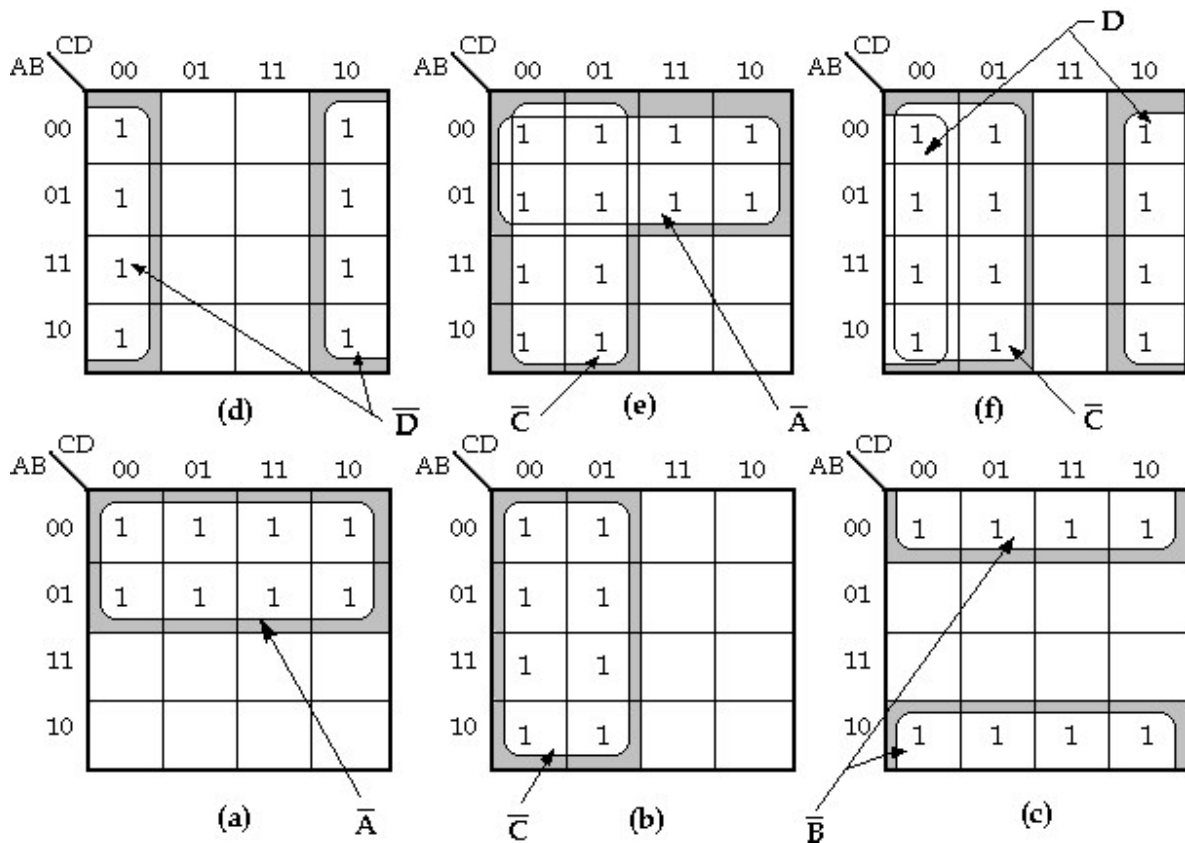
www.binils.com



The four 1's in fig (d) and fig (e) are also adjacent, as are those in fig (f) because, the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

Grouping Eight Adjacent 1's: (Octet)

In a Karnaugh map we can group eight adjacent 1's. The resultant group is called Octet.



Simplification of Sum of Products Expressions: (Minimal Sums)

The generalized procedure to simplify Boolean expressions as follows:

1. Plot the K-map and place 1's in those cells corresponding to the 1's in the sum of product expression. Place 0's in the other cells.
2. Check the K-map for adjacent 1's and encircle those 1's which are not adjacent to any other 1's. These are called **isolated 1's**.
3. Check for those 1's which are adjacent to only one other 1 and encircle such **pairs**.
4. Check for **quads** and **octets** of adjacent 1's even if it contains some 1's that have already been encircled. While doing this make sure that there are minimum number of groups.

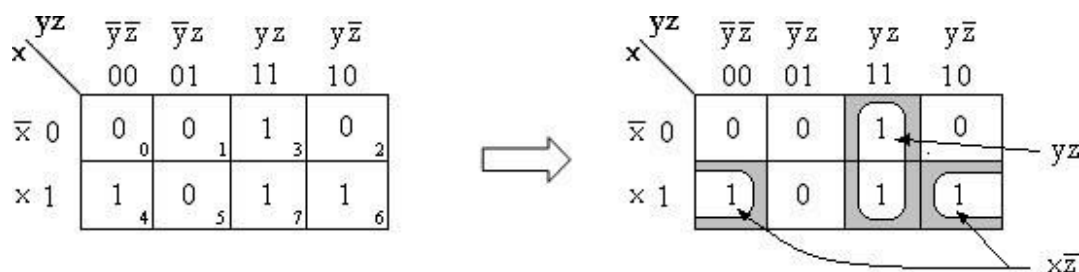
5. Combine any pairs necessary to include any 1's that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

Three- Variable Map:

1. Simplify the Boolean expression,

$$F(x, y, z) = \sum m (3, 4, 6, 7).$$

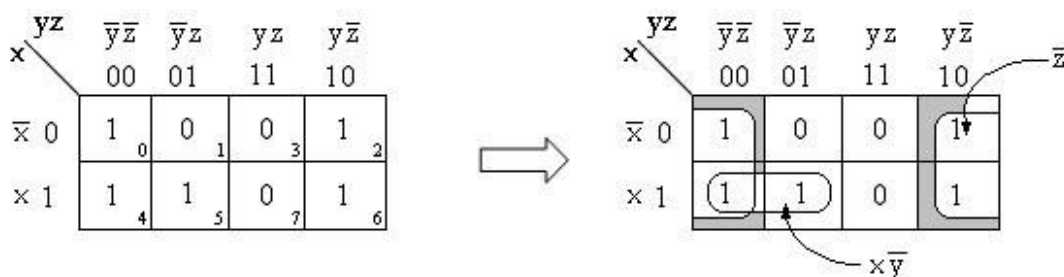
Soln:



$$F = yz + x\bar{z}$$

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

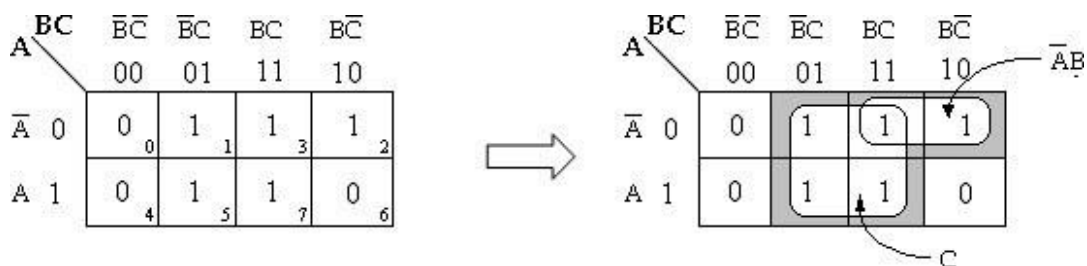
Soln:



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

3. $F = A'C + A'B + AB'C + BC$ **Soln:**

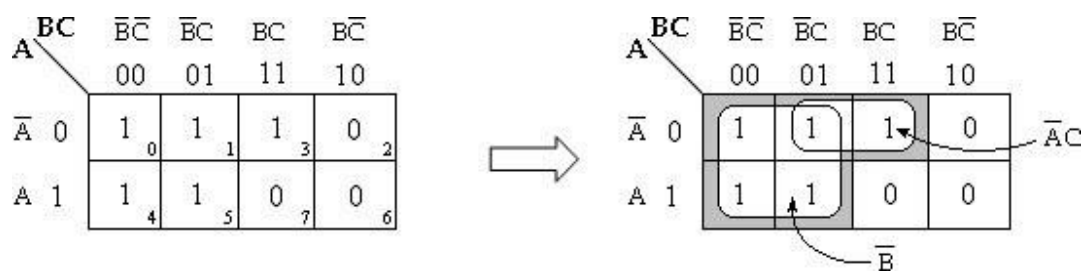
$$\begin{aligned}
 &= A'C(B+B') + A'B(C+C') + AB'C + BC(A+A') \\
 &= \underline{A'BC} + A'B'C + \underline{A'BC} + A'BC' + AB'C + ABC + \underline{A'BC} \\
 &= A'BC + A'B'C + A'BC' + AB'C + ABC \\
 &= m_3 + m_1 + m_2 + m_5 + m_7 \\
 &= \sum m(1, 2, 3, 5, 7)
 \end{aligned}$$



F = C + A'B

4. $AB'C + A'B'C + A'BC + AB'C' + A'B'C'$ Soln:

$$\begin{aligned}
 &= m_5 + m_1 + m_3 + m_4 + m_0 \\
 &= \sum m(0, 1, 3, 4, 5)
 \end{aligned}$$

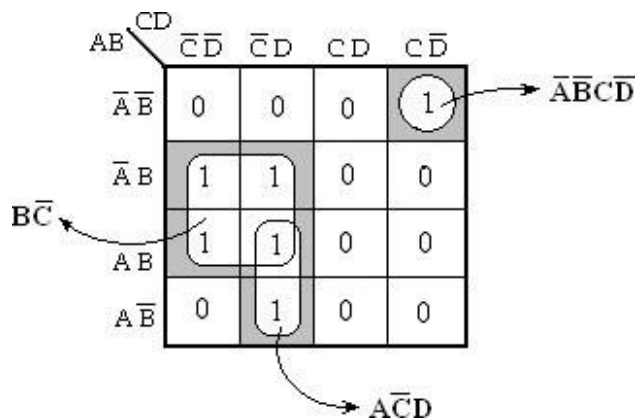


F = A'C + B'

Four - Variable Map:

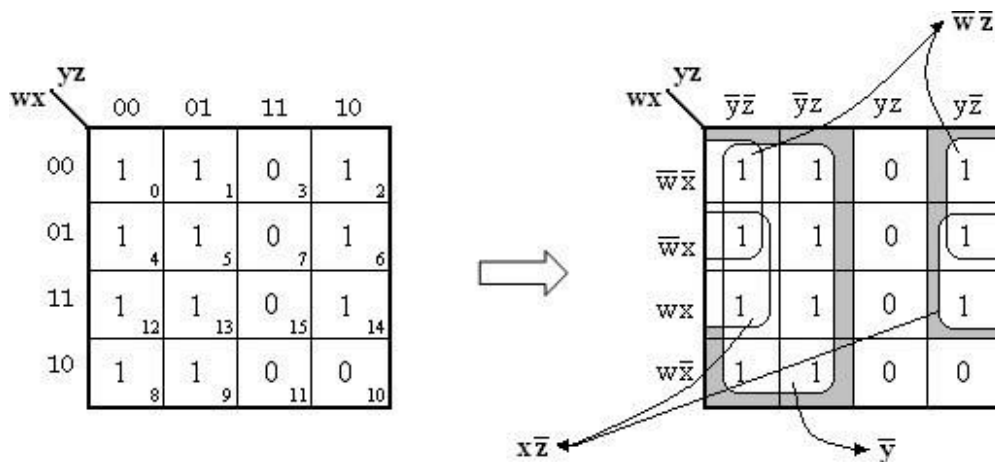
1. Simplify the Boolean expression,

$Y = A'BC'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD'$ Soln:



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

2. **$F(w, x, y, z) = \sum m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$** Soln:



Therefore,

$F = y' + w'z' + xz'$

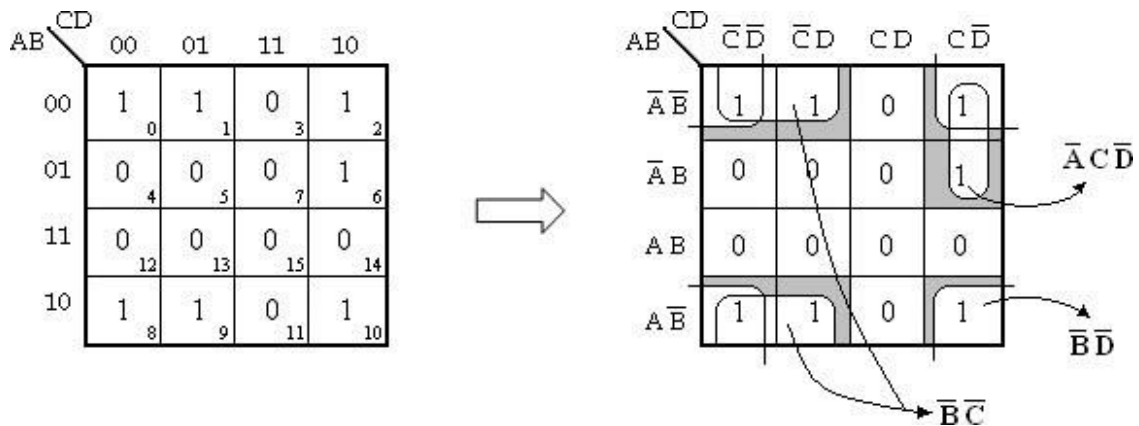
3. $F = A'B'C' + B'CD' + A'BCD' + AB'C'$

$$= A'B'C' (D + D') + B'CD' (A + A') + A'BCD' + AB'C' (D + D')$$

$$= A'B'C'D + A'B'C'D' + AB'CD' + A'B'CD' + A'BCD' + AB'C'D + AB'C'D'$$

$$= m_1 + m_0 + m_{10} + m_2 + m_6 + m_9 + m_8$$

$$= \sum m (0, 1, 2, 6, 8, 9, 10)$$



Therefore, $F = B'D' + B'C' + A'CD'$.

4. $Y = ABCD + AB'C'D' + AB'C + AB$

$$= ABCD + AB'C'D' + AB'C (D + D') + AB (C + C') (D + D')$$

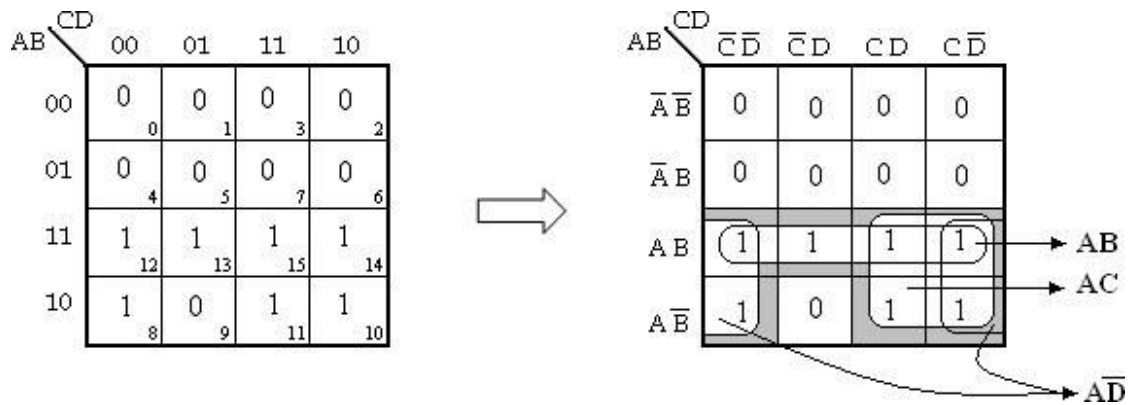
$$= ABCD + AB'C'D' + AB'CD + AB'CD' + (ABC + ABC') (D + D')$$

$$= \underline{ABCD} + AB'C'D' + AB'CD + AB'CD' + \underline{ABCD} + ABCD' + ABC'D + ABC'D'$$

$$= ABCD + AB'C'D' + AB'CD + AB'CD' + ABCD' + ABC'D + ABC'D'$$

$$= m_{15} + m_8 + m_{11} + m_{10} + m_{14} + m_{13} + m_{12}$$

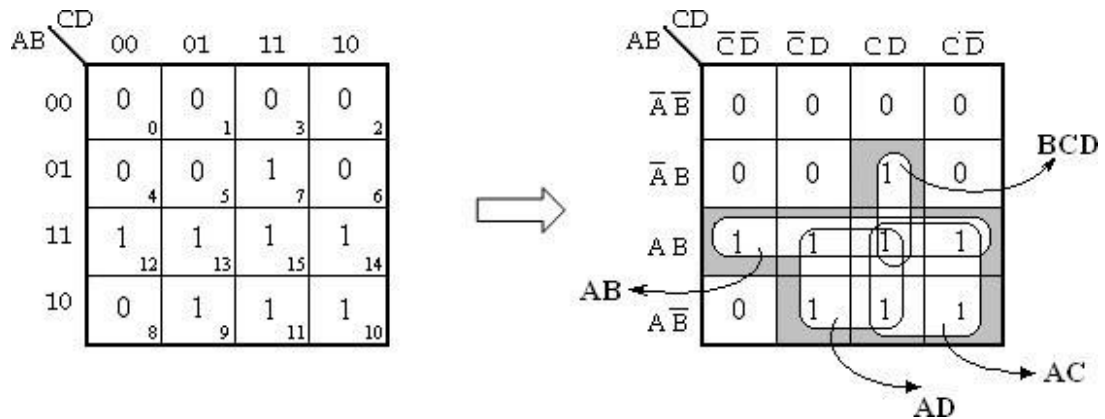
$$= \sum m (8, 10, 11, 12, 13, 14, 15)$$



Therefore, $Y = AB + AC + AD'$.

5. $Y(A, B, C, D) = \sum m(7, 9, 10, 11, 12, 13, 14, 15)$

www.binils.com



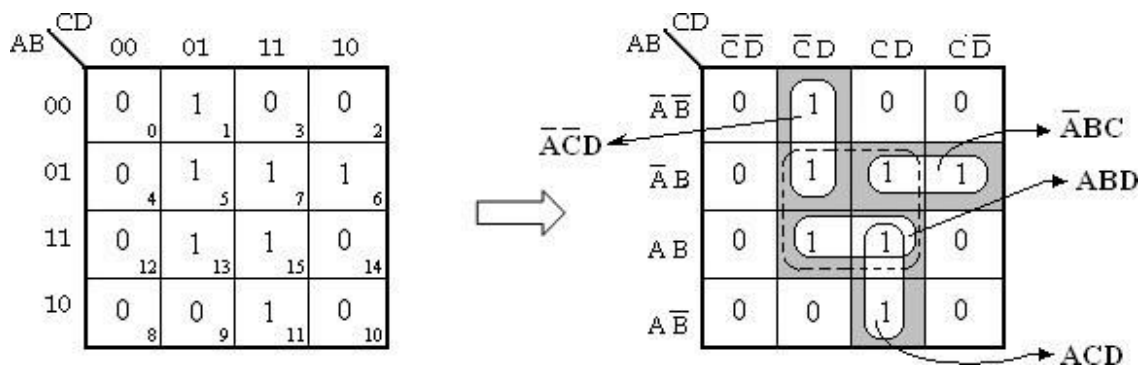
Therefore,

$Y = AB + AC + AD + BCD.$

6. $Y = A'B'C'D + A'BC'D + A'BCD + A'BCD' + ABC'D + ABCD + AB'CD$

$= m_1 + m_5 + m_7 + m_6 + m_{13} + m_{15} + m_{11}$

$= \sum m(1, 5, 6, 7, 11, 13, 15)$

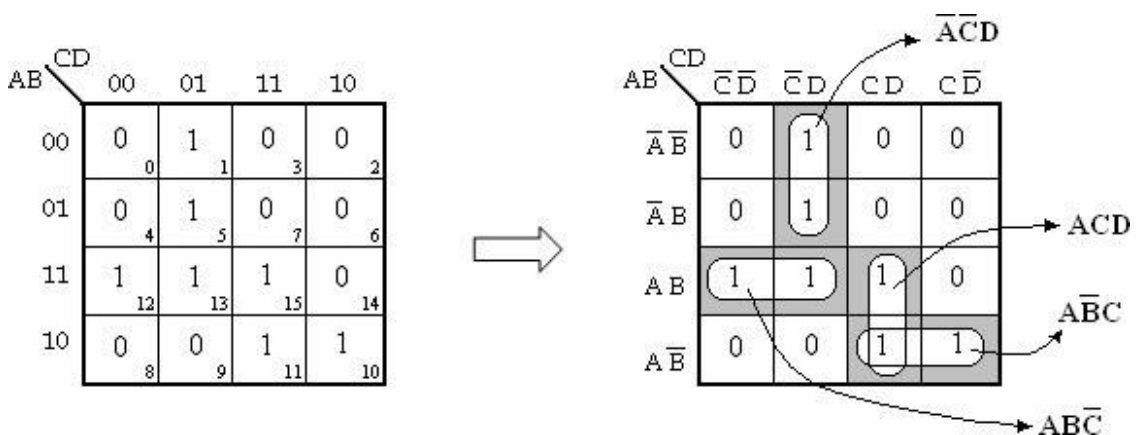


In the above K-map, the cells 5, 7, 13 and 15 can be grouped to form a quad as indicated by the dotted lines. In order to group the remaining 1's, four pairs have to be formed. However, all the four 1's covered by the quad are also covered by the pairs. So, the quad in the above k-map is redundant.

Therefore, the simplified expression will be,

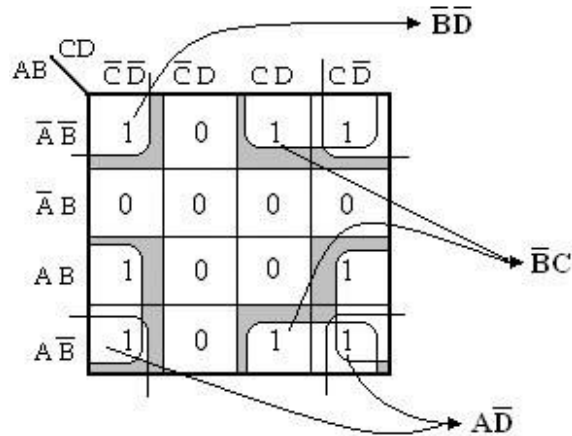
$Y = A'C'D + A'BC + ABD + ACD.$

6. $Y = \sum m(1, 5, 10, 11, 12, 13, 15)$



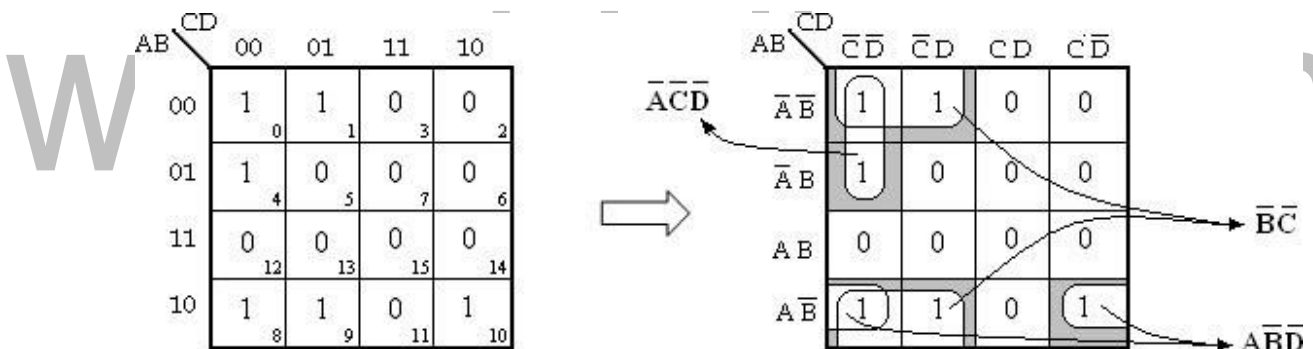
Therefore, **$Y = A'C'D + ABC' + ACD + AB'C.$**

8. $Y = A'B'CD' + ABCD' + AB'CD' + AB'CD + AB'C'D' + ABC'D' + A'B'CD + A'B'C'D'$



Therefore, $Y = AD' + B'C + B'D'$

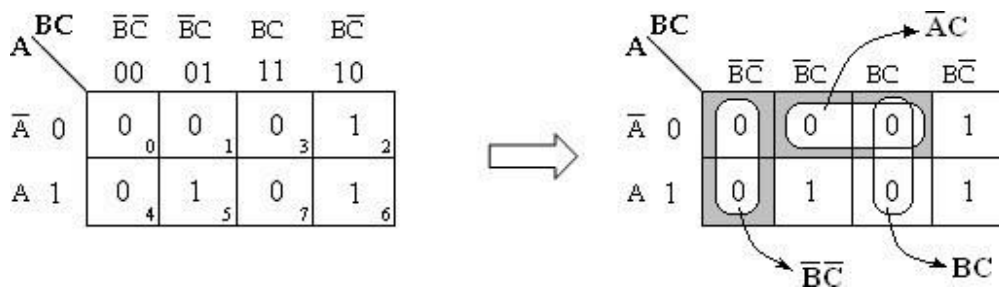
9. $F(A, B, C, D) = \sum m(0, 1, 4, 8, 9, 10)$



Therefore, $F = A'C'D' + AB'D' + B'C'$

Simplification of Sum of Products Expressions: (Minimal Sums)

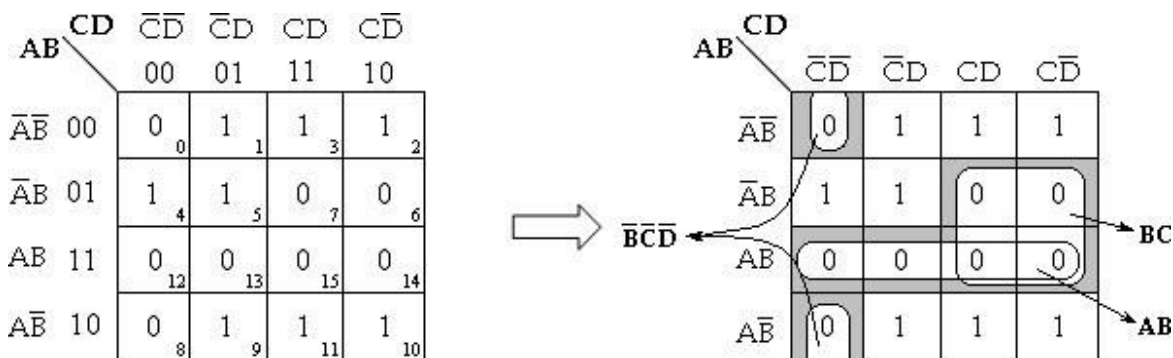
1. $Y = (A + B + C')(A + B' + C')(A' + B + C)(A' + B + C)(A + B + C)$
 $= M_1 \cdot M_3 \cdot M_7 \cdot M_4 \cdot M_0$
 $= \prod M(0, 1, 3, 4, 7)$
 $= \sum m(2, 5, 6)$



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

$$\begin{aligned}
 Y = Y'' &= (B'C' + A'C + BC)' \\
 &= (B'C')' \cdot (A'C)' \cdot (BC)' \\
 &= (B'' + C'') \cdot (A'' + C') \cdot (B' + C') \\
 Y &= (B + C) \cdot (A + C') \cdot (B' + C')
 \end{aligned}$$

2. $Y = (A' + B' + C + D) (A' + B' + C' + D) (A' + B' + C' + D') (A' + B + C + D) (A + B' + C' + D) (A + B' + C' + D') (A + B + C + D) (A' + B' + C + D')$
 $= M_{12} \cdot M_{14} \cdot M_{15} \cdot M_8 \cdot M_6 \cdot M_7 \cdot M_0$
 $M_{13} = \prod M (0, 6, 7, 8, 12, 13, 14, 15)$



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

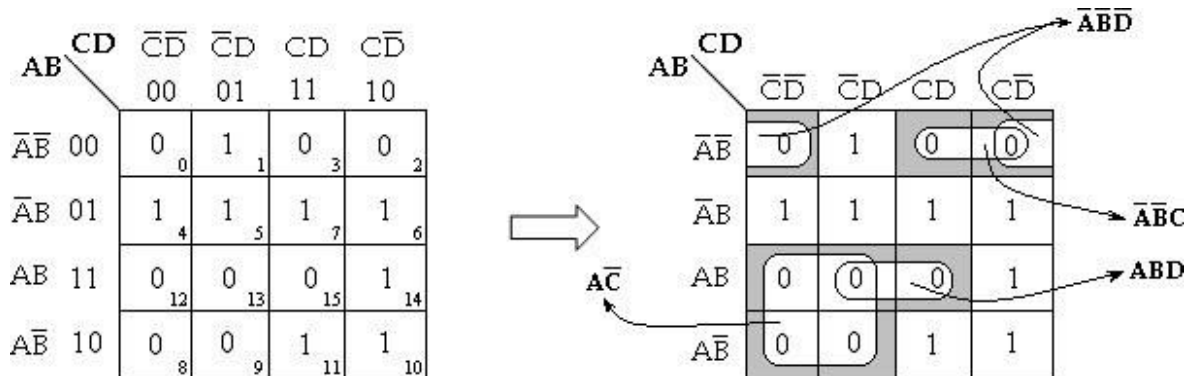
$$\begin{aligned}
 Y = Y'' &= (B'C'D' + AB + BC)' \\
 &= (B'C'D')' \cdot (AB)' \cdot (BC)'
 \end{aligned}$$

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

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

Therefore, $Y = (B + C + D). (A' + B'). (B' + C')$

3. $F(A, B, C, D) = \prod M(0, 2, 3, 8, 9, 12, 13, 14, 15)$



$$Y' = A'B'D' + A'B'C + ABD + AC'$$

$$Y = Y'' = (A'B'D' + A'B'C + ABD + AC')'$$

$$= (A'B'D')'. (A'B'C)'. (ABD)'. (AC')'$$

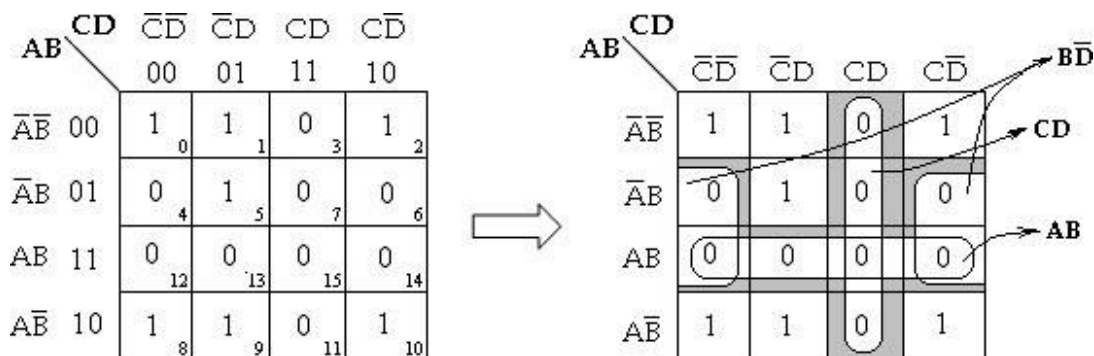
$$= (A'' + B'' + D''). (A'' + B'' + C'). (A' + B' + D'). (A' + C'')$$

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

Therefore, $Y = (A + B + D). (A + B + C'). (A' + B' + D'). (A' + C)$

4. $F(A, B, C, D) = \sum m(0, 1, 2, 5, 8, 9, 10)$

$$= \prod M(3, 4, 6, 7, 11, 12, 13, 14, 15)$$



$Y' = BD' + CD + AB$

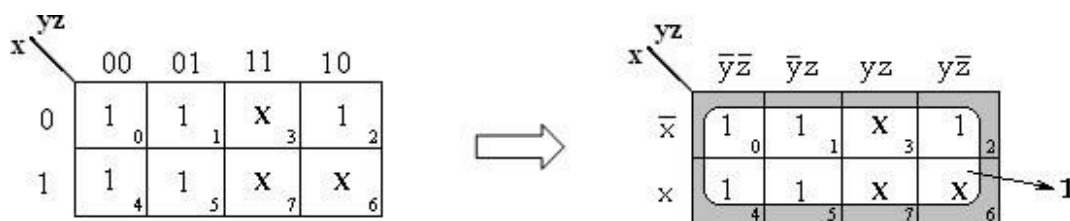
$$\begin{aligned}
 Y = Y'' &= (BD' + CD + AB)' \\
 &= (BD')' \cdot (CD)' \cdot (AB)' \\
 &= (B' + D'') \cdot (C' + D') \cdot (A' + B') \\
 &= (B' + D) \cdot (C' + D') \cdot (A' + B')
 \end{aligned}$$

Therefore, **$Y = (B' + D) \cdot (C' + D') \cdot (A' + B')$**

Don't care Conditions:

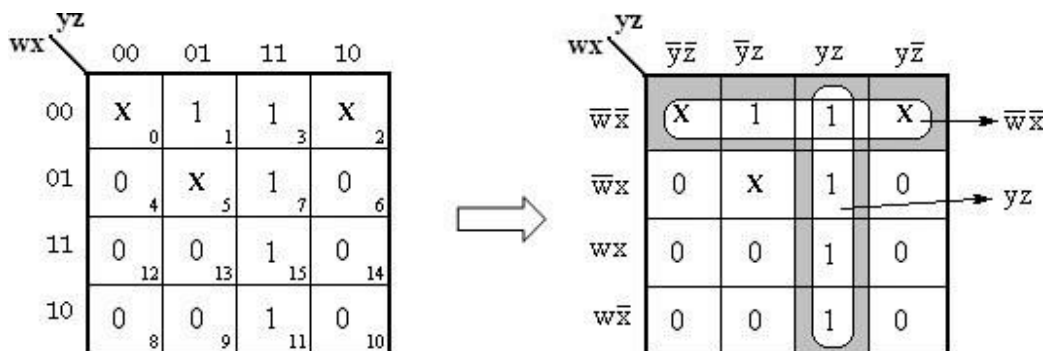
A don't care minterm is a combination of variables whose logical value is not specified. When choosing adjacent squares to simplify the function in a map, the don't care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don't care minterm with either the 1's or the 0's, depending on which combination gives the simplest expression.

1. $F(x, y, z) = \sum m(0, 1, 2, 4, 5) + \sum d(3, 6, 7)$



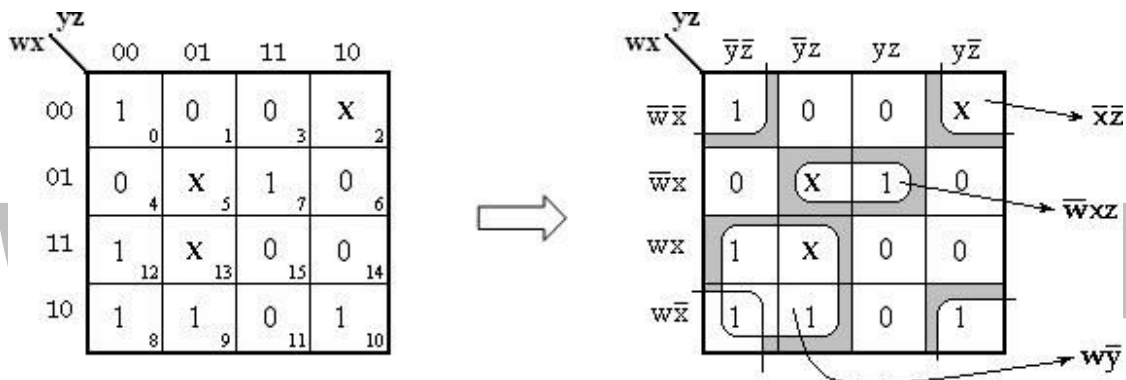
$F(x, y, z) = 1$

2. $F(w, x, y, z) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 5)$



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

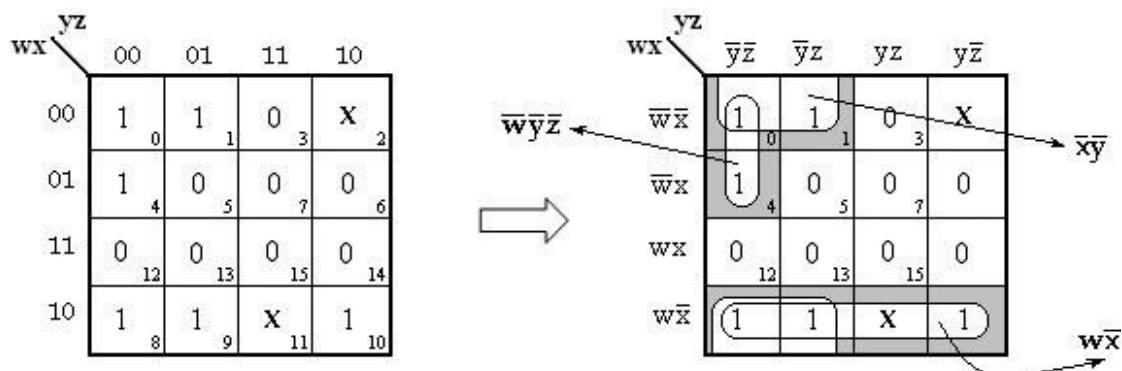
3. $F(w, x, y, z) = \sum m(0, 7, 8, 9, 10, 12) + \sum d(2, 5, 13)$



$F(w, x, y, z) = w'xz + wy' + x'z'$

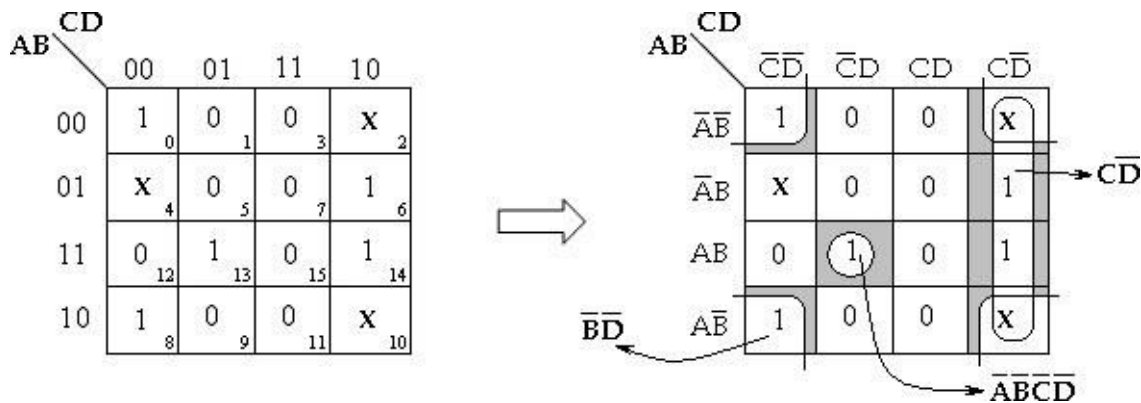
4. $F(w, x, y, z) = \sum m(0, 1, 4, 8, 9, 10) + \sum d(2, 11)$

Soln:



$$F(w, x, y, z) = wx' + x'y' + w'y'z'$$

5. $F(A, B, C, D) = \sum m(0, 6, 8, 13, 14) + \sum d(2, 4, 10)$ **Soln:**



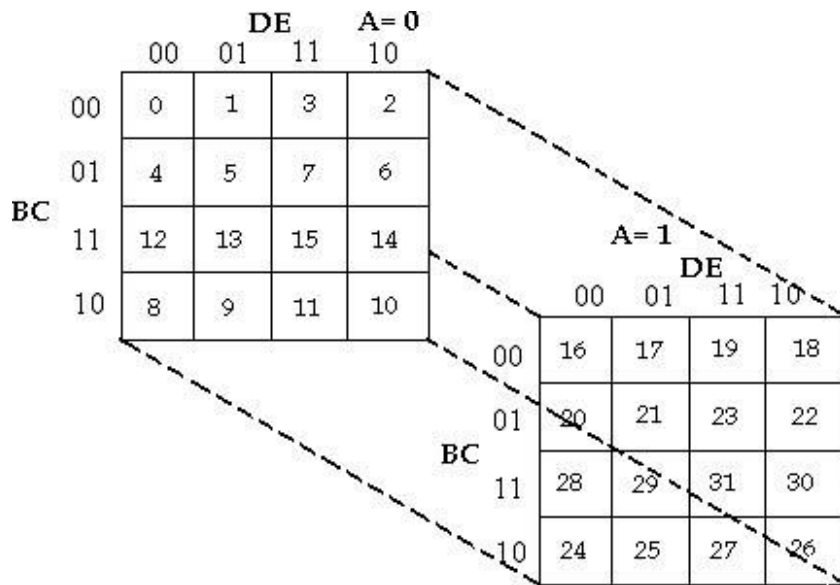
$$F(A, B, C, D) = CD' + B'D' + A'B'C'D'$$

Five- Variable Maps:

A 5- variable K- map requires 25= 32 cells, but adjacent cells are difficult to identify on a single 32-cell map. Therefore, two 16 cell K-maps are used.

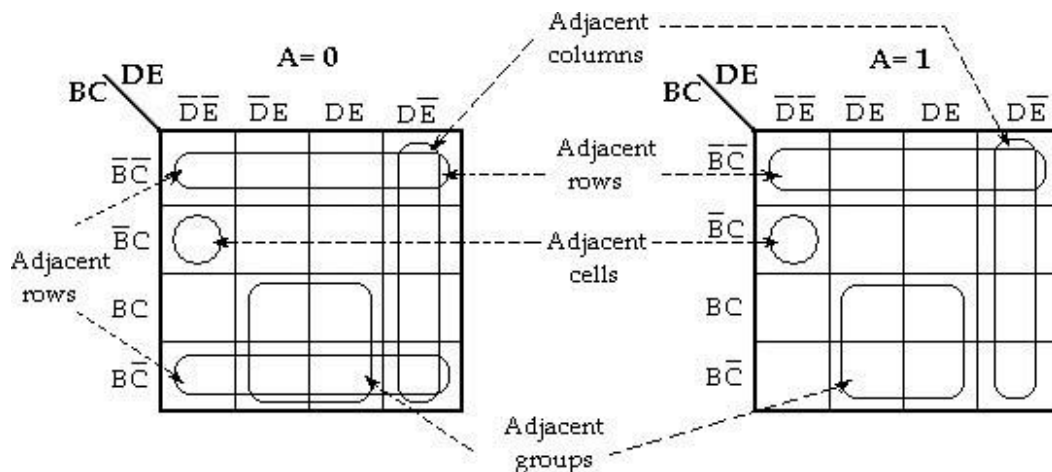
If the variables are A, B, C, D and E, two identical 16- cell maps containing B, C, D and E can be constructed. One map is used for A and other for A'.

In order to identify the adjacent grouping in the 5- variable map, we must imagine the two maps superimposed on one another i.e., every cell in one map is adjacent to the corresponding cell in the other map, because only one variable changes between such corresponding cells.



Five- Variable Karnaugh map (Layer Structure)

Thus, every row on one map is adjacent to the corresponding row (the one occupying the same position) on the other map, as are corresponding columns. Also, the rightmost and leftmost columns within each 16- cell map are adjacent, just as they are in any 16- cell map, as are the top and bottom rows.

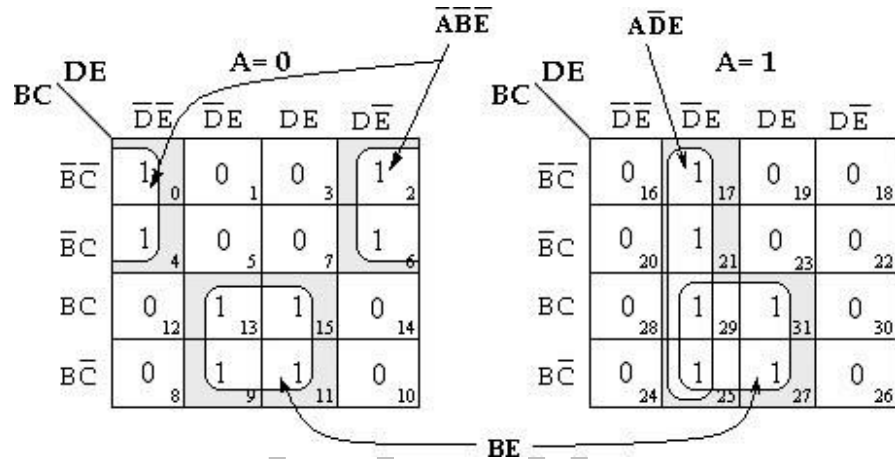


Typical subcubes on a five-variable map

However, the rightmost column of the map is not adjacent to the leftmost column of the other map.

1. Simplify the Boolean function

$$F(A, B, C, D, E) = \sum m(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$$

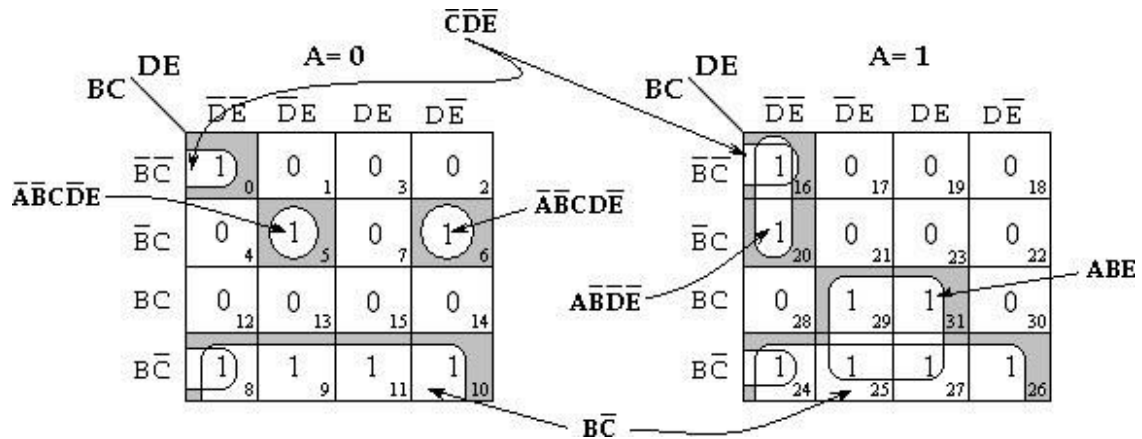


Soln:

$$F(A, B, C, D, E) = \overline{A}B\overline{D}E + BE + A\overline{D}E$$

2. $F(A, B, C, D, E) = \sum m(0, 5, 6, 8, 9, 10, 11, 16, 20, 24, 25, 26, 27, 29, 31)$

Soln:

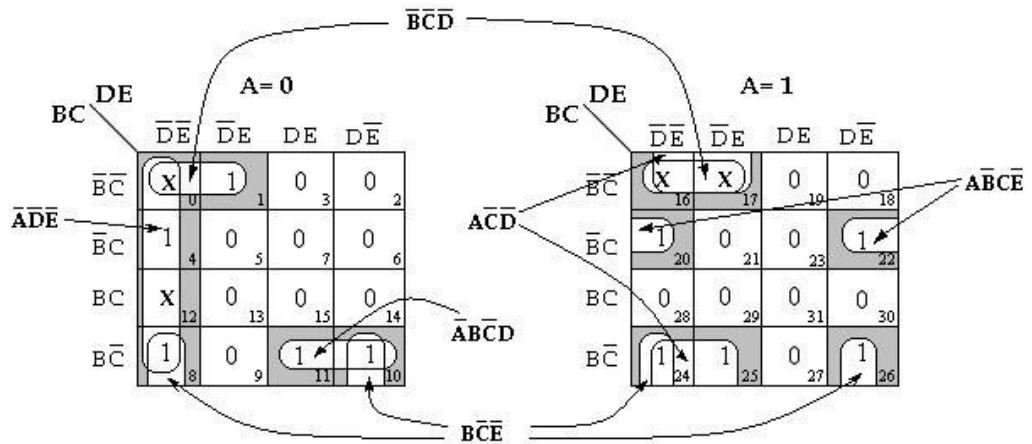


$$F(A, B, C, D, E) = C'D'E' + A'B'CD'E + A'B'CDE' + AB'D'E' + ABE + BC'$$

3. $F(A, B, C, D, E) = \sum m(1, 4, 8, 10, 11, 20, 22, 24, 25, 26) + \sum d(0, 12, 16, 17)$

Soln:

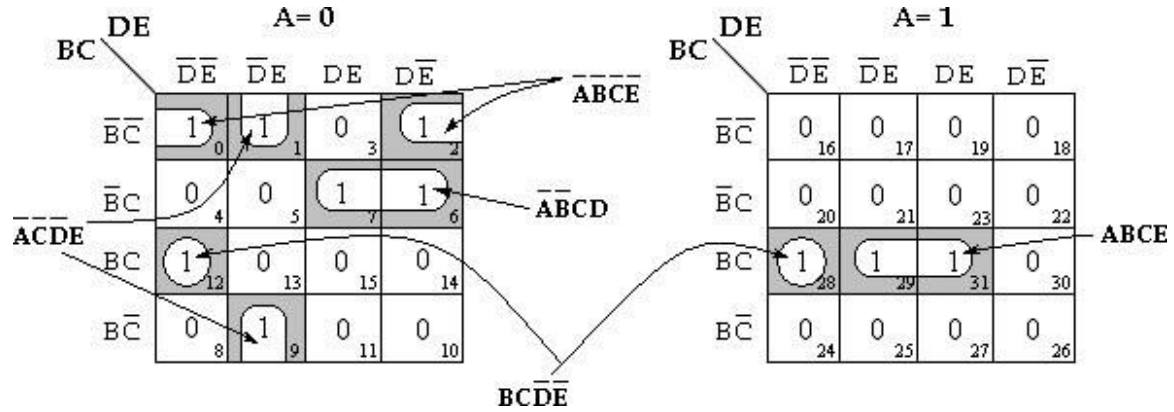
www.binils.com



$$F(A, B, C, D, E) = B'C'D' + A'D'E' + BC'E' + A'BC'D + AC'D' + AB'CE'$$

4. $F(A, B, C, D, E) = \sum m(0, 1, 2, 6, 7, 9, 12, 28, 29, 31)$

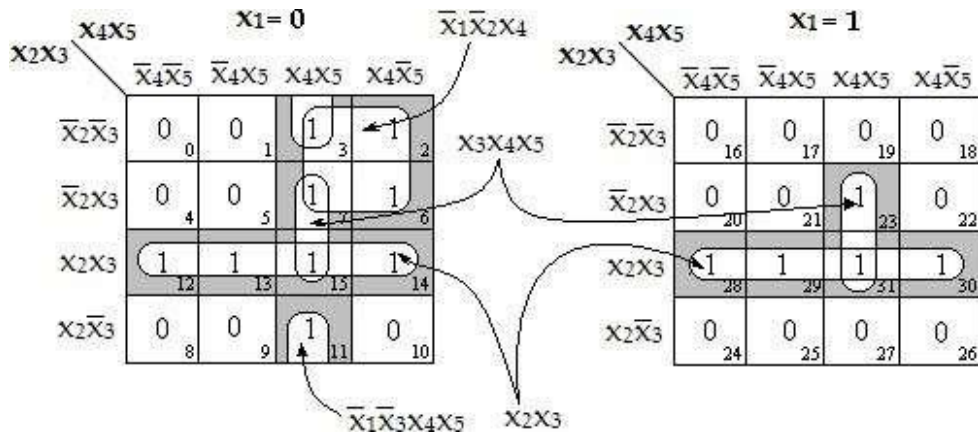
Soln:



$F(A, B, C, D, E) = BCD'E + ABCE + A'B'C'E + A'C'D'E + A'B'CD$

5. $F(x_1, x_2, x_3, x_4, x_5) = \sum m(2, 3, 6, 7, 11, 12, 13, 14, 15, 23, 28, 29, 30, 31)$

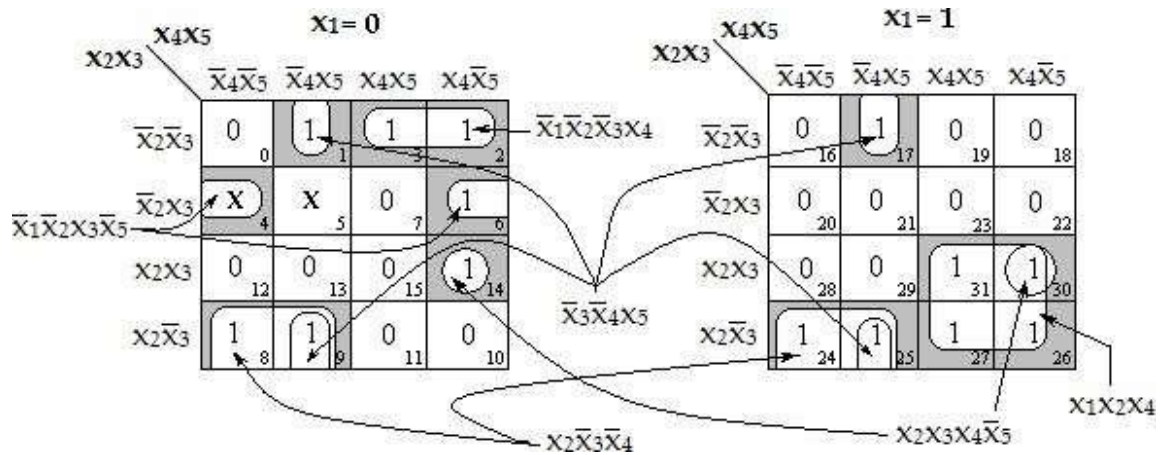
Soln:



$F(x_1, x_2, x_3, x_4, x_5) = x_2x_3 + x_3x_4x_5 + x_1'x_2'x_4 + x_1'x_3'x_4x_5$

6. $F(x_1, x_2, x_3, x_4, x_5) = \sum m(1, 2, 3, 6, 8, 9, 14, 17, 24, 25, 26, 27, 30, 31) + \sum d(4, 5)$

Soln:



$F(x_1, x_2, x_3, x_4, x_5) = x_2x_3'x_4 + x_2x_3x_4x_5 + x_3'x_4'x_5 + x_1x_2x_4 + x_1'x_2'x_3x_5 + x_1'x_2'x_3'x_4$

www.binils.com

EC 8392 – DIGITAL ELECTRONICS

UNIT – I : DIGITAL FUNDAMENTALS

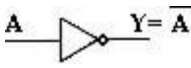
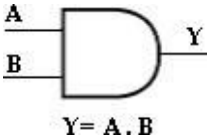
LOGIC GATES

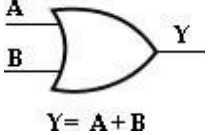
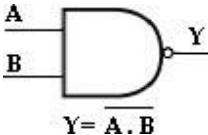
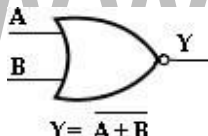
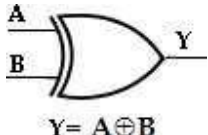
BASIC LOGIC GATES:

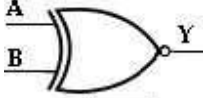
Logic gates are electronic circuits that can be used to implement the most elementary logic expressions, also known as Boolean expressions. The logic gate is the most basic building block of combinational logic.

There are three basic logic gates, namely the OR gate, the AND gate and the NOT gate. Other logic gates that are derived from these basic gates are the NAND gate, the NOR gate, the EXCLUSIVE- OR gate and the EXCLUSIVE-NOR gate.

Table 1.4 – Logic Gates

GATE	SYMBOL	OPERATION	TRUTH TABLE															
NOT (7404)		NOT gate (Inversion), produces an inverted output pulse for a given input pulse.	<table border="1"><thead><tr><th>A</th><th>Y = \bar{A}</th></tr></thead><tbody><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></tbody></table>	A	Y = \bar{A}	0	1	1	0									
A	Y = \bar{A}																	
0	1																	
1	0																	
AND (7408)		AND gate performs logical multiplication . The output is HIGH only when all the inputs are HIGH. When any of the inputs are low, the output is LOW.	<table border="1"><thead><tr><th>A</th><th>B</th><th>Y = A . B</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></tbody></table>	A	B	Y = A . B	0	0	0	0	1	0	1	0	0	1	1	1
A	B	Y = A . B																
0	0	0																
0	1	0																
1	0	0																
1	1	1																

<p>OR (7432)</p>	 <p>$Y = A + B$</p>	<p>OR gate performs logical addition. It produces a HIGH on the output when any of the inputs are HIGH. The output is LOW only when all inputs are LOW.</p>	<table border="1" data-bbox="1150 360 1417 595"> <thead> <tr> <th>A</th> <th>B</th> <th>$Y = A + B$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	$Y = A + B$	0	0	0	0	1	1	1	0	1	1	1	1
A	B	$Y = A + B$																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
<p>NAND (7400)</p>	 <p>$Y = A . B$</p>	<p>It is a universal gate. When any of the inputs are LOW, the output will be HIGH. LOW output occurs only when all inputs are HIGH.</p>	<table border="1" data-bbox="1150 779 1417 1014"> <thead> <tr> <th>A</th> <th>B</th> <th>$Y = A . B$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	$Y = A . B$	0	0	1	0	1	1	1	0	1	1	1	0
A	B	$Y = A . B$																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
<p>NOR (7402)</p>	 <p>$Y = \overline{A + B}$</p>	<p>It is a universal gate. LOW output occurs when any of its input is HIGH. When all its inputs are LOW, the output is HIGH.</p>	<table border="1" data-bbox="1150 1099 1417 1335"> <thead> <tr> <th>A</th> <th>B</th> <th>$Y = \overline{A + B}$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	$Y = \overline{A + B}$	0	0	1	0	1	0	1	0	0	1	1	0
A	B	$Y = \overline{A + B}$																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
<p>EX- OR (7486)</p>	 <p>$Y = A \oplus B$</p>	<p>The output is HIGH only when odd number of inputs is HIGH.</p>	<table border="1" data-bbox="1150 1417 1417 1653"> <thead> <tr> <th>A</th> <th>B</th> <th>$Y = A \oplus B$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	$Y = A \oplus B$	0	0	0	0	1	1	1	0	1	1	1	0
A	B	$Y = A \oplus B$																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

EX-NOR	 <p> $Y = \overline{A \oplus B}$ (or) $Y = A \odot B$ </p>	<p>The output is HIGH only when even number of inputs is HIGH.</p> <p>Or when all inputs are zeros.</p>	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y = A ⊙ B</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y = A ⊙ B	0	0	1	0	1	0	1	0	0	1	1	1
	A	B	Y = A ⊙ B															
0	0	1																
0	1	0																
1	0	0																
1	1	1																

UNIVERSAL GATES:

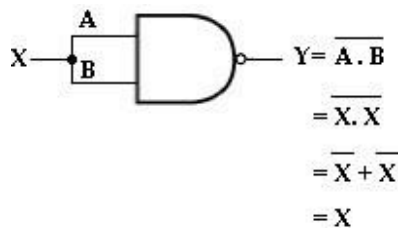
The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates. This is illustrated in the following sections.

a) NAND Gate:

The NAND gate can be used to generate the NOT function, the AND function, the OR function and the NOR function.

i) NOT function:

By connecting all the inputs together and creating a single common input.

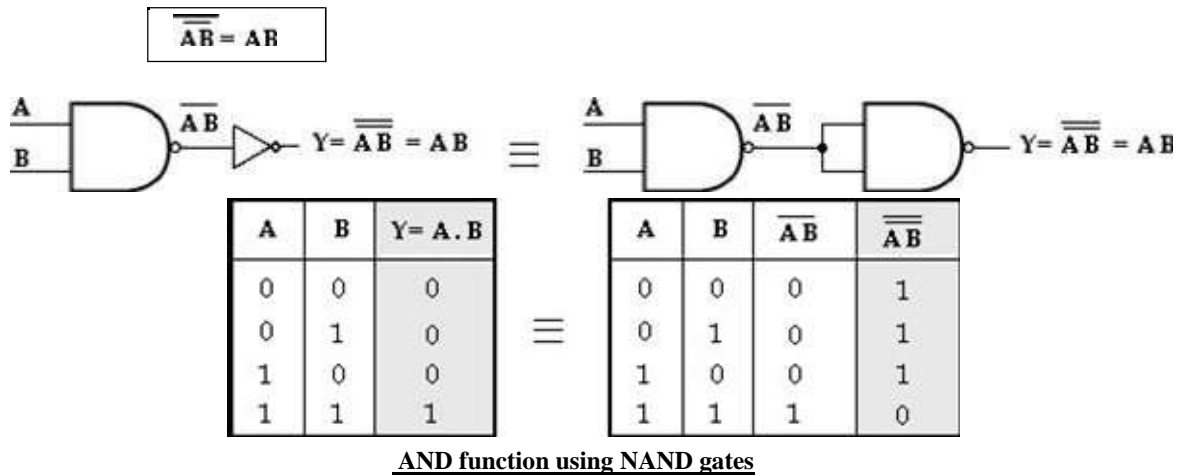


	A	B	Y = A · B	
X=0	0	0	1	Y=1
	0	1	1	
	1	0	1	
X=1	1	1	0	Y=0

NOT function using NAND gate

ii) AND function:

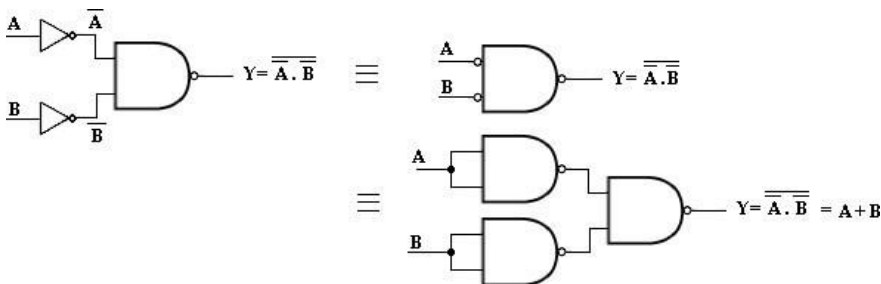
By simply inverting output of the NAND gate. i.e.,



iii) OR function:

By simply inverting inputs of the NAND gate. i.e.,

www.binils.com



OR function using NAND gates

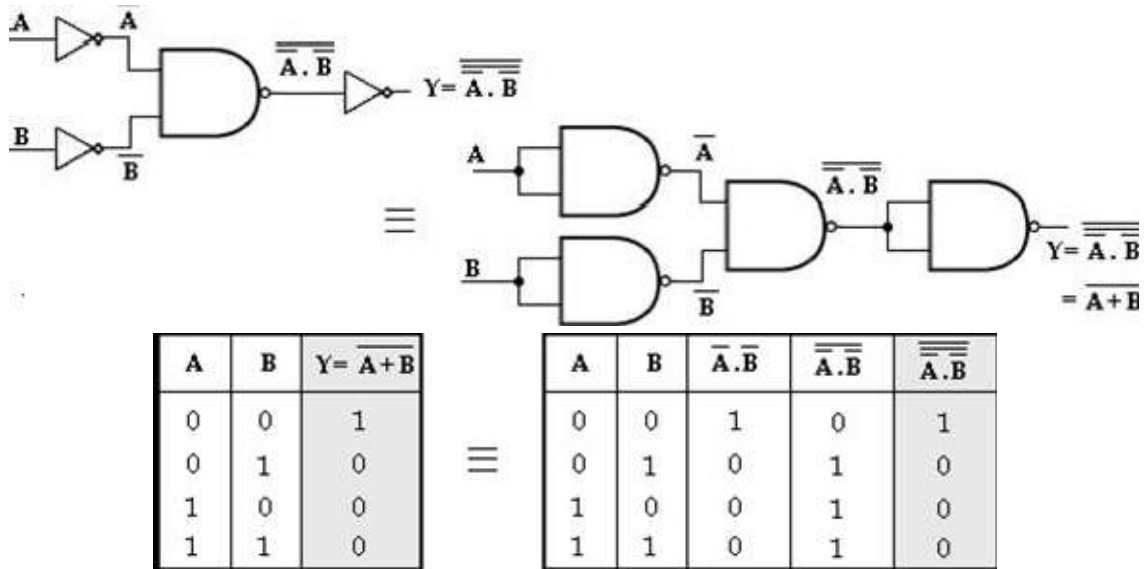
Bubble at the input of NAND gate indicates inverted input.

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

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

iv) **NOR function:**

By inverting inputs and outputs of the NAND gate.



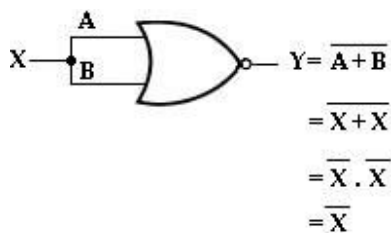
NOR function using NAND gates

b) **NOR Gate:**

Similar to NAND gate, the NOR gate is also a universal gate, since it can be used to generate the NOT, AND, OR and NAND functions.

i) **NOT function:**

By connecting all the inputs together and creating a single common input.



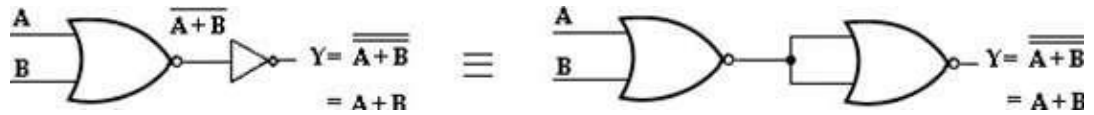
	A	B	$Y = \overline{A + B}$	
X=0	0	0	1	Y=1
	0	1	0	
	1	0	0	
X=1	1	1	0	Y=0

NOT function using NOR gates

ii) OR function:

By simply inverting output of the NOR gate. i.e.,

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



OR function using NOR gates

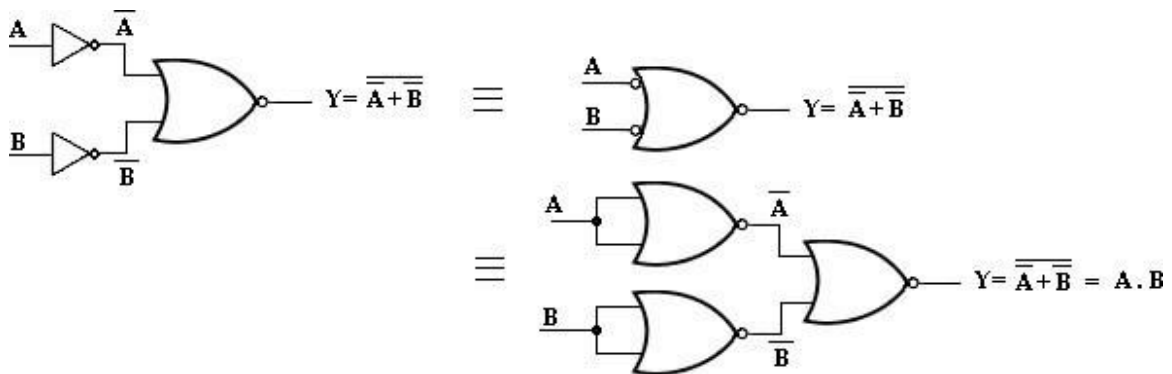
A	B	Y= A+B
0	0	0
0	1	1
1	0	1
1	1	1

≡

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

iii) AND function:

By simply inverting inputs of the NOR gate. i.e.,



AND function using NOR gates

Bubble at the input of NOR gate indicates inverted input.

A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

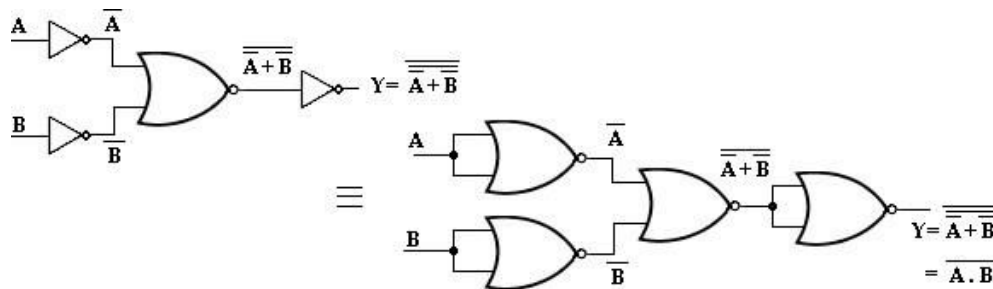
 \equiv

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

Truth table

iv) **NAND Function:**

By inverting inputs and outputs of the NOR gate.



NAND function using NOR gates

www.binils.com

A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

 \equiv

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

Conversion of AND/OR/NOT to NAND/NOR:

1. Draw AND/OR logic.
2. If NAND hardware has been chosen, add bubbles on the output of each AND gate and bubbles on input side to all OR gates.

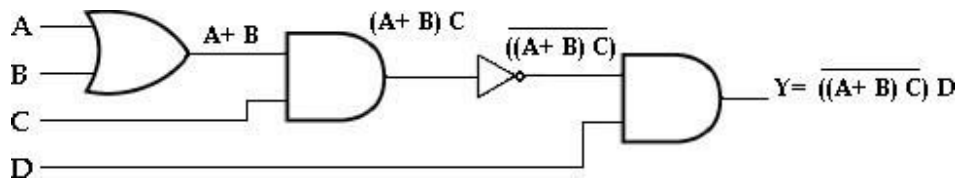
If NOR hardware has been chosen, add bubbles on the output of each OR gate and bubbles on input side to all AND gates.

3. Add or subtract an inverter on each line that received a bubble in step 2.
4. Replace bubbled OR by NAND and bubbled AND by NOR.
5. Eliminate double inversions.

1. **Implement Boolean expression using NAND gates:**

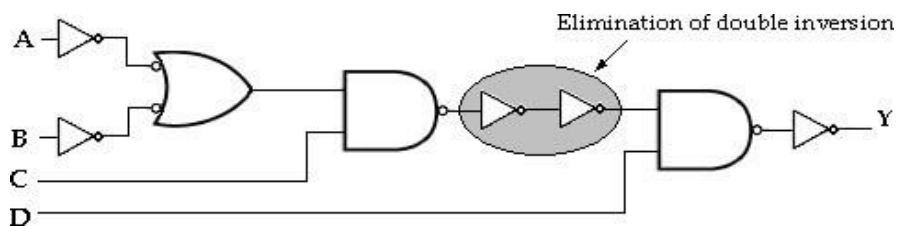
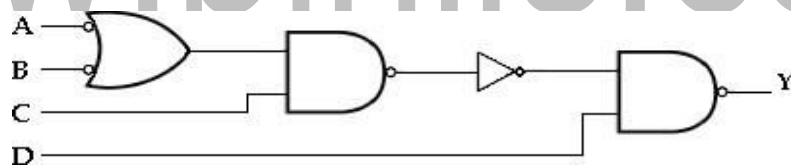
$$\overline{((A + B) C)} D$$

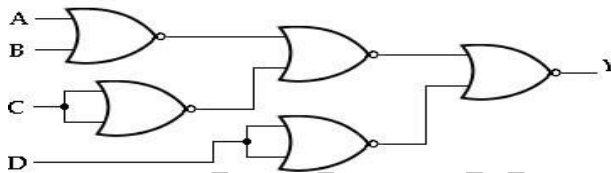
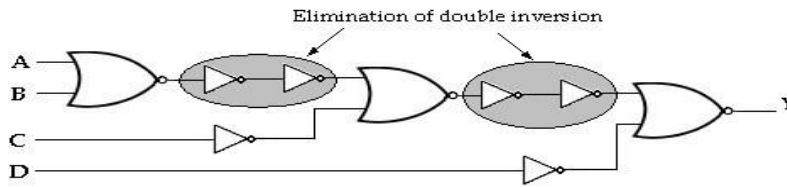
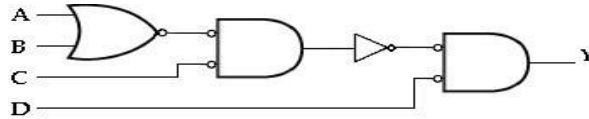
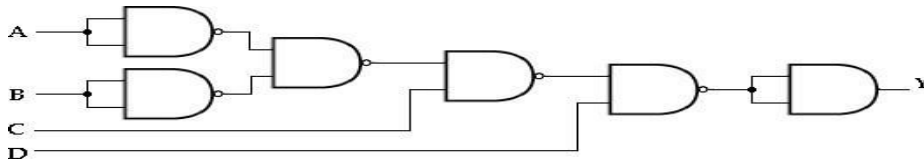
Original Circuit:



Soln:

NAND Circuit:



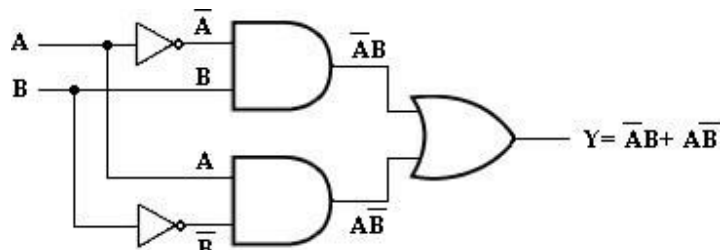


www.binils.com

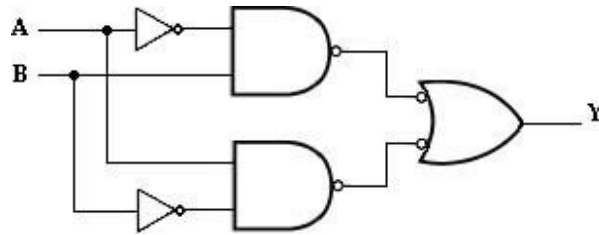
:

2. Implement Boolean expression for EX-OR gate using NAND gates.

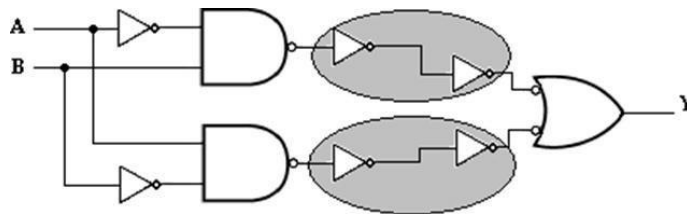
Soln: gate.



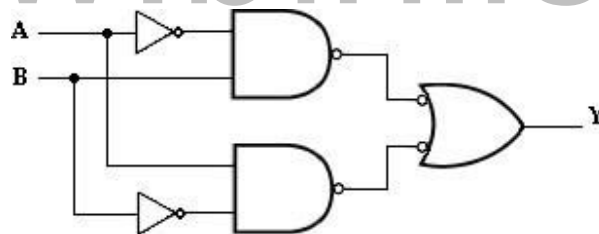
Adding bubbles on the output of each AND gates and on the inputs of each OR



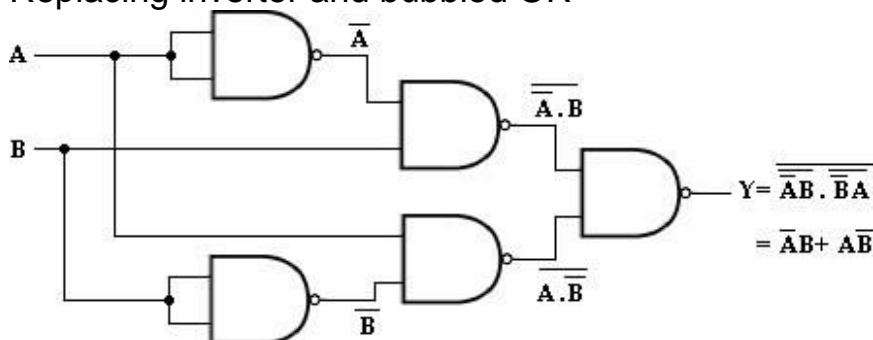
Adding an inverter on each line that received bubble,



Eliminating double inversion,



Replacing inverter and bubbled OR



with NAND, we have

www.binils.com

EC 8392 – DIGITAL ELECTRONICS

UNIT – I : DIGITAL FUNDAMENTALS

BOOLEAN FUNCTIONS:

Minimization of Boolean Expressions:

The Boolean expressions can be simplified by applying properties, laws and theorems of Boolean algebra.

Simplify the following Boolean functions to a minimum number of literals:

1. $x(x'+y)$

$$\begin{aligned} &= xx' + xy \\ &= 0 + xy \\ &= xy. \end{aligned}$$

$$\begin{aligned} &[x \cdot x' = 0] \\ &[x + 0 = x] \end{aligned}$$

2. $x + x'y$

$$\begin{aligned} &= x + xy + x'y \\ &= x + y(x+x') \\ &= x + y(1) \\ &= x + y. \end{aligned}$$

$$\begin{aligned} &[x + xy = \\ &x] [x + x' \\ &= 1] \end{aligned}$$

3. $(x + y)(x + y')$

$$\begin{aligned} &= x \cdot x + xy' + xy + \\ &yy' \\ &= x + xy' + xy + 0 \\ &= x(1 + y' + y) \\ &= x(1) \end{aligned}$$

$$\begin{aligned} &[x \cdot x = 0]; [y \cdot \\ &y' = 0] \\ &[1 + y = 1] \end{aligned}$$

= x.

4. $xy + x'z + yz$

$$\begin{aligned} &= xy + x'z + yz(x + \\ &x') \\ &= xy + x'z + xyz + \\ &x'yz \text{ Re-arranging,} \\ &= xy + xyz + x'z \\ &+ x'yz \end{aligned}$$

$$[x + x' = 1]$$

$$\begin{aligned} &= xy(1+z) + x'z \\ &(1+y) \\ &= xy + x'z. \end{aligned}$$

$$[1+y=1]$$

$$\begin{aligned} 5. & xy + yz + y'z \\ &= xy + z(y+y') \\ &= xy + z(1) \\ &= xy + z. \end{aligned}$$

$$[y+y'=1]$$

$$\begin{aligned} 6. & (x+y)(x'+z)(y+z) \\ &= (x+y)(x'+z) \end{aligned}$$

$$\begin{aligned} &[\text{dual form of consensus theorem,} \\ & (A+B)(A'+C)(B+C) = (A+B)(A'+C)] \end{aligned}$$

$$\begin{aligned} 7. & x'y + xy + x'y' \\ &= y(x'+x) + x'y' \\ &= y(1) + x'y' \\ &[x+x'=1] \\ &= y + x'y'. \end{aligned}$$

$$[x(y+z) = xy + xz]$$

$$[x+x'=1] = y + x'y' \quad [x+x'y' =$$

$$\begin{aligned} 8. & x + xy' + x'y = x(1+y') + x'y \\ &= x(1) + x'y \\ &= x + x'y \\ &= x + y. \end{aligned}$$

$$[1+x=1]$$

$$[x+x'y = x+y]$$

$$\begin{aligned} 9. & AB + (AC)' + AB'C (AB + C) \\ &= AB + (AC)' + AAB'BC + AB'CC \\ &= AB + (AC)' + 0 + AB'CC \\ &= AB + (AC)' + AB'C \\ &= AB + A' + C' + AB'C \\ &= AB + A' + C' + AB' \\ &= A' + B + C' + AB' \end{aligned}$$

$$[B.B' = 0]$$

$$[C.C = 1]$$

$$[(AC)' = A' + C']$$

$$[C' + AB'C = C' + AB']$$

$$[A' + AB = A' + B]$$

Re-arranging,

$$\begin{aligned} &= A' + AB' + B + C' \\ &= A' + B' + B + C' \\ &= A' + 1 + C' \\ &= 1 \end{aligned}$$

$$[A' + AB = A' + B]$$

$$[B' + B = 1]$$

$$[A + 1 = 1]$$

10. $(x' + y)(x + y)$

$$\begin{aligned} &= x'.x + x'y + yx + y.y \\ &= 0 + x'y + xy + y \\ &= y(x' + x + 1) \\ &= y(1) \\ &= y. \end{aligned}$$

$$[x.x' = 0]; [x.x = x]$$

$$[1 + x = 1]$$

11. $xy + xyz + xy(w + z)$

$$\begin{aligned} &= xy(1 + z + w + z) \\ &= xy(1) \\ &= xy. \end{aligned}$$

$$[1 + x = 1]$$

12. $xy + xyz + xyz' + x'yz = xy(1 + z + z') + x'yz$

$$\begin{aligned} &= xy(1) + x'yz \\ &= xy + x'yz \\ &= y(x + x'z) \\ &= y(x + z). \end{aligned}$$

$$[1 + x = 1]$$

$$[x + x'y = x + y]$$

13. $xyz + xy'z + xyz' = xy(z + z') + xy'z$

$$\begin{aligned} &= xy + xy'z \\ &= x(y + y'z) \\ &= x(y + z) \end{aligned}$$

$$[x + x' = 1]$$

$$[x + x'y = x + y]$$

14. $x'y'z' + x'yz' + xy'z' + xyz'$

$$\begin{aligned} &= x'z'(y' + y) + xz'(y' + y) \\ &= x'z' + xz' \\ &= z'(x' + x) \end{aligned}$$

$$[x + x' = 1]$$

$$= z' \quad [x + x' = 1]$$

$$\begin{aligned}
 15. w'xyz' + xyz' + xy'z' + xy'z & \\
 = xyz'(w' + 1) + xy'z' + & \\
 xy'z & [1 + x = 1] \\
 = xyz' + xy'z' + xy'z & \\
 = xz'(y + y') + xy'z & [x + x' = 1] \\
 = xz' + xy'z & \\
 = x(z' + y'z) & [1] \\
 = x(z' + y') & [x' + xy' = x' + y']
 \end{aligned}$$

$$\begin{aligned}
 16. w'xy'z + w'xyz + wxz & \\
 = w'xz(y' + y) + wxz & \\
 = w'xz(1) + wxz & \\
 = w'xz + wxz & [x + x' = 1] \\
 = xz(w' + w) & \\
 = xz & [x + x' = 1]
 \end{aligned}$$

$$\begin{aligned}
 17. x'y'z' + x'y'z + x'yz' + x'yz + xy'z' & \\
 = x'y'(z' + z) + x'y(z' + z) + xy'z' & \\
 = x'y'(1) + x'y(1) + xy'z' & [x + x' = 1] \\
 = x'y' + x'y + xy'z' & \\
 = x'(y' + y) + xy'z' & \\
 = x'(1) + & [x + x' = 1] \\
 xy'z' & \\
 = x' + xy'z' & \\
 = x' + y'z' & [x' + xy' = x' + y']
 \end{aligned}$$

$$\begin{aligned}
 18. w'y(w'xz)' + w'xy'z' + wx'y & \\
 = w'y(w'' + x' + z') + w'xy'z' + wx'y & \\
 = w'y(w + x' + z') + w'xy'z' + wx'y & [x'' = x] \\
 = w'yw + w'yx' + w'yz' + w'xy'z' + wx'y & \\
 = 0 + w'x'y + w'y z' + w'xy'z' + wx'y & [x \cdot x' = 0]
 \end{aligned}$$

Re-arranging,

$$\begin{aligned}
 = w'x'y + wx'y + w'y z' + w'xy'z' & \\
 = x'y(w' + w) + w'z'(y + xy') & \\
 = x'y(1) + w'z'(y + xy') & [x + x' = 1]
 \end{aligned}$$

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

$$[x + x'y = x + y]$$

$$\begin{aligned} 19. & xy + x(y+z) + y(y+z) \\ &= xy + xy + xz + yy + yz \\ &= xy + xz + y + yz \\ &= xy + xz + y \\ &= y + xz \end{aligned}$$

$$\begin{aligned} [x + x = x]; [x \cdot x = x] \\ [x + xy = x] \\ [x + xy = x] \end{aligned}$$

$$\begin{aligned} 20. & [xy'(z+wy) + x'y']z \\ &= [xy'z + xy'wy + x'y']z \\ &= [xy'z + 0 + x'y']z \\ &= xy'z \cdot z + x'y'z \\ &= xy'z + x'y'z \\ &= y'z(x+x') \\ &= y'z(1) \\ &= y'z \end{aligned}$$

$$\begin{aligned} [x \cdot x' = 0] \\ [x \cdot x = x] \\ [x + x' = 1] \end{aligned}$$

$$\begin{aligned} 21. & x'yz + xy'z' + x'y'z' + xy'z + xyz \\ &= yz(x'+x) + xy'z' + x'y'z' + xy'z \\ &= yz(1) + y'z'(x+x') + xy'z \\ &= yz + y'z'(1) + xy'z \\ &= yz + y'z' + xy'z \\ &= yz + y'(z'+xz) \\ &= yz + y'(z'+x) \\ &= yz + y'z' + xy' \end{aligned}$$

$$\begin{aligned} [x + x' = 1] \\ [x + x' = 1] \end{aligned}$$

$$[x' + xy = x' + y]$$

$$\begin{aligned} 22. & [(xy)' + x' + xy]' \\ &= [x' + y' + x' + xy]' \\ &= [x' + y' + xy]' \\ &= [x' + y' + x]' \\ &= [y' + 1]' \end{aligned}$$

$$\begin{aligned} [x + x = x] \\ [x' + xy = x' + y] \\ [x + x' = 1] \end{aligned}$$

$$= [1]'$$

$$[1+x=1]$$

$$= 0.$$

$$23. [xy+xz]'+x'y'z$$

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

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

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

$$x'y'z$$

$$[x+x=x]$$

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

$$x'y'z$$

$$= x'+x'z'+x'y'+y'[z'+$$

$$x'z]$$

$$= x'+x'z'+x'y'+y'[z'+x']$$

$$[x'+xy=x'+y]$$

$$= x'+x'y'+y'[z'+$$

$$[x+xy=x]$$

$$x']$$

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

$$x'y'$$

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

$$[x+xy=x]$$

$$= x'+y'z'.$$

$$[x+xy=x]$$

$$24. xy+xy'(x'z)'$$

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

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

$$[x''=x]$$

$$= xy+xy'x+xy'z$$

$$= xy+xy'+xy'z$$

$$[x.x=x]$$

$$= xy+xy'[1+z]$$

$$= xy+xy'[1]$$

$$[1+x=1]$$

$$= xy+xy'$$

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

$$= x[1]$$

$$[x+x'=1]$$

$$= x.$$

$$25. [(xy'+xyz)'+x(y+xy')]'$$

$$= [x(y'+yz)'+x(y+xy')]'$$

$$[x'+xy=x'+y]; [x+x'y=x+y]$$

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

$$= [x(y'+z)'+xy+x.x)]'$$

$$= [(xy'+xz)'+xy+x)]'$$

$$[x.x=x]$$

$$= [(xy'+xz)'+x)]'$$

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

$$= [(xy)'.(xz)'+x]'$$

$$\begin{aligned} & [x + xy = x] \\ & = [(x' + y) \cdot (x' + z') + x]' \end{aligned}$$

$$[x'' = x]$$

www.binils.com

$$\begin{aligned}
 &= [(x' + yz') + x]' && [(x + y)(x + z) = x + yz] \\
 &= [x' + yz' + x]' \\
 &= [1 + yz']' && [x + x' = 1] \\
 &= [1]' && [1 + x = 1] \\
 &= 0.
 \end{aligned}$$

$$\begin{aligned}
 26. & [(xy + z')((x + y)' + z)]' && \\
 &= [(xy + z')((x' + y') + z)]' \\
 &= [xy \cdot x'y' + xy \cdot z + z' \cdot x'y' + z' \cdot z]' \\
 &= [0 + xyz + x'y'z' + 0]' && [x \cdot x' = 0] \\
 &= [xyz + x'y'z']' \\
 &= (xyz)' \cdot (x'y'z')' \\
 &= (x' + y' + z') \cdot (x'' + y'' + z'') \\
 &= (x' + y' + z') \cdot (x + y + z) && [x'' = x]
 \end{aligned}$$

$$\begin{aligned}
 27. & (x + y)(x'z' + z)(y' + xz)' && \\
 &= (x + y)(x'z' + z)(y'' \cdot (xz)') \\
 &= (x + y)(x' + z)(y \cdot (xz)') && [x + x'y = x + y]; [x'' = x] \\
 &= (x + y)(x' + z)(y \cdot (x' + z')) \\
 &= (x \cdot x' + xz + x'y + yz)(x'y + yz') \\
 &= (0 + xz + x'y + yz)(x'y + yz') \\
 &= (xz + x'y + yz)(x'y + yz') \\
 &= xz \cdot x'y + xz \cdot yz' + x'y \cdot x'y + x'y \cdot yz' + yz \cdot x'y + yz \cdot yz' \\
 &= 0 + 0 + x'y + x'yz' + x'yz + 0 && [x \cdot x' = 0]; [x \cdot x = x] \\
 &= x'y + x'yz' + x'yz \\
 &= x'y(1 + z' + z) \\
 &= x'y(1) && [1 + x = 1] \\
 &= x'y.
 \end{aligned}$$

$$\begin{aligned}
 28. & Y = \sum m(1, 3, 5, 7) \\
 &= x'y'z + x'yz + xy'z + xyz
 \end{aligned}$$

$$\begin{aligned} &= x'z(y'+y) + xz(y'+y) \\ &= x'z(1) + xz(1) && [x+x'=1] \\ &= x'z + xz \\ &= z(x'+x) \\ &= z(1) && [x+x'=1] \\ &= z. \end{aligned}$$

COMPLEMENT OF A FUNCTION :

The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be derived algebraically through DeMorgan's theorem.

DeMorgan's theorems for any number of variables resemble in form the two-variable case and can be derived by successive substitutions similar to the method used in the preceding derivation. These theorems can be generalized as –

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

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

Find the complement of the following functions,

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

$$\begin{aligned} &= (x'' + y' + z'') \cdot (x'' + y'' + z') = \\ &(x + y' + z). (x + y + z'). \end{aligned}$$

2. $F = (xy + y'z + xz) x$.

$$\begin{aligned} F' &= [(xy + y'z + xz) x]' \\ &= (xy + y'z + xz)' + x' \\ &= [(xy)' \cdot (y'z)'. (xz)'] + x' \\ &= [(x'+y'). (y+z'). (x'+z')] + x' \end{aligned}$$

$$\begin{aligned} &= [(x'y + x'z' + 0 + y'z') (x' + z')] + x' \\ &= x'x'y + x'x'z' + x'y'z' + x'yz' + x'z'z' + y'z'z' + x' \\ &= x'y + x'z' + x'y'z' + x'yz' + x'z' + y'z' + x' && [x + x = x], [x \cdot x = x] \\ &= x'y + x'z' + x'z' (y' + y) + y'z' + x' && [x + x' = 1] \\ &= x'y + x'z' + x'z' (1) + y'z' + x' \\ &= x'y + x'z' + y'z' + x' \\ &= x'y + x' + x'z' + y'z' \\ &= x'(y+1) + x'z + y'z' [y+1 = 1] = x'(1+z) + y'z' && [y+1 = 1] \\ &= x' + y'z' \end{aligned}$$

3. $F = x(y'z' + yz)$ $F' = [x$

$$(y'z' + yz)']$$

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

$$= x' + (y'z')' \cdot (yz)'$$

$$(y'' + z'') \cdot (y' + z') = x' +$$

$$(y + z) \cdot (y' + z')$$

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

$$(xy' + x'y)'$$

$$= (xy')' \cdot (x'y)'$$

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

$$x'y' + yx + yy' = x'y' +$$

$$xy.$$

5. $f = wx'y + xy' +$

$wxz \quad f' = (wx'y +$

$xy' + wxz)'$

$= (wx'y)' (xy')' (wxz)'$

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

$= (w'x'+w'y+xx'+xy+x'y'+yy') (w'+x'+z')$

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

$= w'x'. w'+w'y. w'+xy. w'+x'y'. w'+w'x'. x'+w'y. x'+xy. x'+x'y'. x'+$

$w'x'. z'+w'y. z'+xy. z'+x'y'.z'$

$= w'x'+w'y+w'xy+w'x'y'+w'x'+w'x'y+0+x'y'+w'x'z'+w'yz'+xyz'+x'y'z'$

$= w'x'+w'y+w'xy+w'x'y'+w'x'y+x'y'+w'x'z'+w'yz'+xyz'+x'y'z'$

$= w'x'(1+y+y+z')+w'y(1+x+z')+x'y'(1+z')+xyz'$

$= w'x'(1)+w'y(1)+x'y'(1)+xyz'$

$= w'x'+w'y+x'y'+xyz'$

www.binils.com

EC 8392 – DIGITAL ELECTRONICS

UNIT – I : DIGITAL FUNDAMENTALS

Number Systems

We all use numbers to communicate and perform several tasks in our daily lives. Our present day world is characterized by measurements and numbers associated with everything. In fact, many consider if we cannot express something in terms of numbers is not worth knowing. While this is an extreme view that is difficult to justify, there is no doubt that quantification and measurement, and consequently usage of numbers, are desirable whenever possible.

Manipulation of numbers is one of the early skills that the present day child is trained to acquire. The present day technology and the way of life require the usage of several number systems. Usage of decimal numbers starts very early in one's life. Therefore, when one is confronted with number systems other than decimal, some time during the high-school years, it calls for a fundamental change in one's framework of thinking. There have been two types of numbering systems in use through out the world.

One type is symbolic in nature. Most important example of this symbolic numbering system is the one based on Roman numerals

I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 and M = 1000 IIMVII - 2007

While this system was in use for several centuries in Europe it is completely superseded by the weighted-position system based on Indian numerals. The

Roman number system is still used in some places like watches and release dates of movies.

The weighted-positional system based on the use of radix 10 is the most commonly used numbering system in most of the transactions and activities of today's world. However, the advent of computers and the convenience of using devices that have two well defined states brought the binary system, using the radix 2, into extensive use. The use of binary number system in the field of computers and electronics also lead to the use of octal (based on radix 8) and hex-decimal system (based on radix 16). The usage of binary numbers at various levels has become so essential that it is also necessary to have a good understanding of all the binary arithmetic operations. Here we explore the weighted-position number systems and conversion from one system to the other.

Weighted-Position Number System

In a weighted-position numbering system using Indian numerals the value associated with a digit is dependent on its position. The value of a number is weighted sum of its digits. Consider the decimal number 2357. It can be expressed as

$$2357 = 2 \times 10^3 + 3 \times 10^2 + 5 \times 10^1 + 7 \times 10^0$$

Each weight is a power of 10 corresponding to the digit's position. A decimal point allows negative as well as positive powers of 10 to be used;

$$526.47 = 5 \times 10^2 + 2 \times 10^1 + 6 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2}$$

Here, 10 is called the base or radix of the number system.

In a general positional number system, the radix may be any integer $r > 2$, and a digit position i has weight r^i . The general form of a number in such a system is

$$d_{p-1} d_{p-2}, \dots, d_1, d_0 . d_{-1} d_{-2} \dots d_{-n}$$

where there are p digits to the left of the point (called radix point) and n digits to the right of the point. The value of the number is the sum of each digit multiplied by the corresponding power of the radix.

Except for possible leading and trailing zeros, the representation of a number in positional system is unique (00256.230 is the same as 256.23). Obviously the values d_i 's can take are limited by the radix value. For example a number like $(356)_5$, where the suffix 5 represents the radix will be incorrect, as there can not be a digit like 5 or 6 in a weighted position number system with radix 5.

If the radix point is not shown in the number, then it is assumed to be located near the last right digit to its immediate right. The symbol used for the radix point is a point (.). However, a comma is used in some countries. For example 7,6 is used, instead of 7.6, to represent a number having seven as its integer component and six as its fractional. As much of the present day electronic hardware is dependent on devices that work reliably in two well defined states, a numbering system using 2 as its radix has become necessary and popular. With the radix value of 2, the binary number system will have only two numerals, namely 0 and 1. Consider the number

$$(N)_2 = (11100110)_2.$$

It is an eight digit binary number. The binary digits are also known as bits. Consequently the above number would be referred to as an 8-bit number. Its decimal value is given by

$$\begin{aligned}(N)_2 &= 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 128 + 64 + 32 + 0 + 0 + 4 + 2 + 0 \\ &= (230)_{10}\end{aligned}$$

Consider a binary fractional number $(N)_2 = 101.101$.

Its decimal value is given by

$$\begin{aligned}(N)_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 4 + 0 + 1 + 1/2 + 0 + 1/8 \\ &= 5 + 0.5 + 0.125 \\ &= (5.625)_{10}\end{aligned}$$

From here on we consider any number without its radix specifically mentioned, as a decimal number. With the radix value of 2, the binary number system requires very long strings of 1s and 0s to represent a given number. Some of the problems associated with handling large strings of binary digits may be eased by grouping them into three digits or four digits.

We can use the following groupings. Octal (radix 8 to group three binary digits) Hexadecimal (radix 16 to group four binary digits) In the octal number

system the digits will have one of the following eight values 0, 1, 2, 3, 4, 5, 6 and 7. In the hexadecimal system we have one of the sixteen values 0 through 15. However, the decimal values from 10 to 15 will be represented by alphabet A (=10), B (=11), C (=12), D (=13), E (=14) and F (=15).

Conversion of a binary number to an octal number or a hexadecimal number is very simple, as it requires simple grouping of the binary digits into groups of three or four. Consider the binary number 11011011.

It may be converted into octal or hexadecimal

numbers as $(11011001)_2 = (011) (011) (001)$

$= (331)_8$

$= (1101) (1001)$

$= (D9)_{16}$

Note that adding a leading zero does not alter the value of the number. Similarly for grouping the digits in the fractional part of a binary number, trailing zeros may be added without changing the value of the number. Number System Conversions In general, conversion between numbers with different radices cannot be done by simple substitutions. Such conversions would involve arithmetic operations. Let us work out procedures for converting a number in any radix to radix 10, and viceversa.

Decimal value of the number is determined by converting each digit of the number to its radix-10 equivalent and expanding the formula using radix-10 arithmetic.

Some examples are:

$$\begin{aligned}(331)_8 &= 3 \times 8^2 + 3 \times 8^1 + 1 \times 8^0 \\ &= 192 + 24 + 1 \\ &= (217)_{10} \quad (D9)_{16} \\ &= 13 \times 16^1 + 9 \times 16^0 \\ &= 208 + 9 \\ &= (217)_{10}\end{aligned}$$

$$\begin{aligned}(33.56)_8 &= 3 \times 8^1 + 3 \times 8^0 + 5 \times 8^{-1} + 6 \times 8^{-2} \\ &= (27.69875)_{10} \quad (E5.A)_{16} \\ &= 14 \times 16^1 + 5 \times 16^0 + 10 \times 16^{-1} \\ &= (304.625)_{10}\end{aligned}$$

The conversion formula can be rewritten as $D = ((\dots ((d_{n-1}) \cdot r + d_{n-2}) \cdot r + \dots) \cdot r + d_1) \cdot r + d_0$. This forms the basis for converting a decimal number D to a number with radix r . If we divide the right hand side of the above formula by r , the remainder will be d_0 , and the quotient will be $Q = ((\dots ((d_{n-1}) \cdot r + d_{n-2}) \cdot r + \dots) \cdot r + d_1)$. Thus, d_0 can be computed as the remainder of the long division of D by the radix r . As the quotient Q has the same form as D , another long division by r

will give d_1 as the remainder. This process can continue to produce all the digits of the number with radix r .

Consider the following examples.

	Quotient	Remainder
$156 \div 2$	78	0
$78 \div 2$	39	0
$39 \div 2$	19	1
$19 \div 2$	9	1
$9 \div 2$	4	1
$4 \div 2$	2	0
$2 \div 2$	1	0
$1 \div 2$	0	1

$$(156)_{10} = (10011100)_2$$

	Quotient	Remainder
$678 \div 8$	84	6
$84 \div 8$	10	4
$10 \div 8$	1	2

$$1 \div 8 \qquad 0 \qquad 1$$

$$(678)_{10} = (1246)_8$$

Quotient

Remainder $678 \div 16$

42 \qquad 6

$$42 \div 16 \qquad 2 \qquad A$$

$$2 \div 16 \qquad 0 \qquad 2$$

$$(678)_{10} = (2A6)_{16}$$

Representation of Negative Numbers In our traditional arithmetic we use the “+” sign before a number to indicate it as a positive number and a “-” sign to indicate it as a negative number. We usually omit the sign before the number if it is positive. This method of representation of numbers is called “sign-magnitude” representation. But using “+” and “-” signs on a computer is not convenient, and it becomes necessary to have some other convention to represent the signed numbers. We replace “+” sign with “0” and “-” with “1”. These two symbols already exist in the binary system. Consider the following examples:

$$(+1100101)_2 (01100101)_2$$

$$(+101.001)_2 (0101.001)_2$$

$$(-10010)_2 (110010)_2$$

$$(-110.101)_2 (1110.101)_2$$

In the sign-magnitude representation of binary numbers the first digit is always treated separately. Therefore, in working with the signed binary numbers in sign-magnitude form the leading zeros should not be ignored. However, the leading zeros can be ignored after the sign bit is separated.

For example,

1000101.11 = -

101.11

While the sign-magnitude representation of signed numbers appears to be natural extension of the traditional arithmetic, the arithmetic operations with signed numbers in this form are not that very convenient, either for implementation on the computer or for hardware implementation. There are two other methods of representing signed numbers.

Diminished Radix Complement (DRC) or (r-1)-complement Radix Complement (RX) or r-complement When the numbers are in binary form Diminished Radix Complement will be known as “one’s-complement” Radix complement will be known as “two’s-complement”. If this representation is extended to the decimal numbers they will be known as 9’s complement and 10’s- complement respectively.

One’s Complement Representation

Let A be an n-bit signed binary number in one’s complement form. The most significant bit represents the sign. If it is a “0” the number is positive and if it is a “1” the number is negative. The remaining (n-1) bits represent the magnitude, but not necessarily as a simple weighted number.

Consider the following one's complement numbers and their decimal equivalents:

0111111	+ 63
0000110	--> + 6
0000000	--> + 0
1111111	--> + 0
1111001	--> - 6
1000000	--> - 63

There are two representations of "0", namely 000.....0 and 111.....1.

From these illustrations we observe

- If the most significant bit (MSD) is zero the remaining (n-1) bits directly indicate the magnitude.
- If the MSD is 1, the magnitude of the number is obtained by taking the complement of all the remaining (n-1) bits.

For example consider one's complement representation of -6 as given above.

- Leaving the first bit '1' for the sign, the remaining bits 111001 do not directly represent the magnitude of the number -6.
- Take the complement of 111001, which becomes 000110 to determine the magnitude.

In the example shown above a 7-bit number can cover the range from +63 to -63. In general an n-bit number has a range from $+(2^{n-1} - 1)$ to $-(2^{n-1} - 1)$ with two representations for zero. The representation also suggests that if A is an integer in one's complement form, then one's complement of A = -A

One's complement of a number is obtained by merely complementing all the digits. This relationship can be extended to fractions as well.

For example

if $A = 0.101 (+0.625)_{10}$,

then the one's complement of A is 1.010,

which is one's complement representation of $(-0.625)_{10}$.

Similarly consider the case of a mixed number.

$A = 010011.0101 (+19.3125)_{10}$

One's complement of A = 101100.1010 (- 19.3125)₁₀

This relationship can be used to determine one's complement representation of negative decimal numbers. Example 1: What is one's complement binary representation of decimal number -75? Decimal number 75 requires 7 bits to represent its magnitude in the binary form. One additional bit is needed to represent the sign.

Therefore, one's complement representation of 75 =

01001011 one's complement representation of -75 =

10110100

Two's Complement Representation Let A be an n-bit signed binary number in two's complement form. The most significant bit represents the sign. If it is a "0", the number is positive, and if it is "1" the number is negative. The remaining (n-1) bits represent the magnitude, but not as a simple weighted number. Consider the following two's complement numbers and their decimal equivalents:

$$0111111 + 63$$

$$0000110 + 6$$

$$0000000 + 0$$

$$1111010 - 6$$

$$1000001 - 63$$

$$1000000 - 64$$

There is only one representation of "0", namely 000 0.

From these illustrations we observe If most significant bit (MSD) is zero the remaining (n-1) bits directly indicate the magnitude.

If the MSD is 1, the magnitude of the number is obtained by taking the complement of all the remaining (n-1) bits and adding a 1.

Consider the two's complement representation of -

6. We assume we are representing it as a 7-bit

number. Leave the sign bit.

The remaining bits are 111010.

These have to be complemented

(that is 000101) and a 1 has to be added (that is $000101 + 1 = 000110 = 6$).

In the example shown above a 7-bit number can cover the range from +63 to -64.

In general an n-bit number has a range from $+(2^{n-1} - 1)$ to $-(2^{n-1})$ with one representation for zero. The representation also suggests that if A is an integer in two's complement form, then Two's complement of $A = -A$. Two's complement of a number is obtained by complementing all the digits and adding '1' to the LSB. This relationship can be extended to fractions as well.

If $A = 0.101$ ($+0.625$)₁₀,

then the two's complement of A is 1.011,

which is two's complement representation of (-0.625) ₁₀.

Similarly consider the case of a mixed number.

$A = 010011.0101$ ($+19.3125$)₁₀

Two's complement of A = 101100.1011 (-19.3125)₁₀

This relationship can be used to determine two's complement representation of negative decimal numbers.

Example 2:

What is two's complement binary representation of decimal number -75?

Decimal number 75 requires 7 bits to represent its magnitude in the binary form. One additional bit is needed to represent the sign. Therefore, Two's complement representation of 75 = 01001011 Two's complement representation of -75 = 10110101

CODES:

When we wish to send information over long distances unambiguously it becomes necessary to modify (encoding) the information into some form before sending, and convert (decode) at the receiving end to get back the original information. This process of encoding and decoding is necessary because the channel through which the information is sent may distort the transmitted information. Much of the information is sent as numbers. While these numbers are created using simple weighted-positional numbering systems, they need to be encoded before transmission. The modifications to numbers were based on changing the weights, but predominantly on some form of binary encoding. There are several codes in use in the context of present day information technology, and more and more new codes are being generated to meet the new demands.

Coding is the process of altering the characteristics of information to make it more suitable for intended application. By assigning each item of information a unique combination of 1s and 0s we transform some given information into binary coded form. The bit combinations are referred to as "words" or "code words". In the field of digital systems and computers different bit combinations have different designations.

Bit - a binary digit 0 or 1

Nibble - a group of four bits

Byte - a group of eight bits

Word - a group of sixteen

bits;

a word has two bytes or four nibbles

Sometimes 'word' is used to designate a larger group of bits also, for example 32 bit or 64 bit words. We need and use coding of information for a variety of reasons

- to increase efficiency of transmission,
- to make it error free,
- to enable us to correct it if errors occurred,
- to inform the sender if an error occurred in the received information etc.
- for security reasons to limit the accessibility of information
- to standardise a universal code that can be used by all

Coding schemes have to be designed to suit the security requirements and the complexity of the medium over which information is transmitted.

Binary Coded Decimal Codes

The main motivation for binary number system is that there are only two elements in the binary set, namely 0 and 1. While it is advantageous to perform all computations on hardware in binary forms, human beings still prefer to work with decimal numbers. Any electronic system should then be able to accept

decimal numbers, and make its output available in the decimal form. One method, therefore, would be to

- convert decimal number inputs into binary form
- manipulate these binary numbers as per the required functions, and
- convert the resultant binary numbers into the decimal form

However, this kind of conversion requires more hardware, and in some cases considerably slows down the system. Faster systems can afford the additional circuitry, but the delays associated with the conversions would not be acceptable. In case of smaller systems, the speed may not be the main criterion, but the additional circuitry may make the system more expensive.

We can solve this problem by encoding decimal numbers as binary strings, and use them for subsequent manipulations.

There are ten different symbols in the decimal number system: 0, 1, 2, . . . , 9. As there are ten symbols we require at least four bits to represent them in the binary form. Such a representation of decimal numbers is called binary coding of decimal numbers.

As four bits are required to encode one decimal digit, there are sixteen four-bit groups to select ten groups. This would lead to nearly 30×10^{10} ($16C_{10} \cdot 10!$) possible codes. However, most of them will not have any special properties that would be useful in hardware design. We wish to choose codes that have some desirable properties like

- ease of coding
- ease in arithmetic operations

- minimum use of hardware
- error detection property
- ability to prevent wrong output during transitions

In a weighted code the decimal value of a code is the algebraic sum of the weights of 1s appearing in the number. Let $(A)_{10}$ be a decimal number encoded in the binary form as $a_3a_2a_1a_0$. Then

$$(A)_{10} = w_3a_3 + w_2a_2 + w_1a_1 + w_0a_0$$

where w_3, w_2, w_1 and w_0 are the weights selected for a given code, and a_3, a_2, a_1 and a_0 are either 0s or 1s.

The more popularly used codes have the weights as

w_3	w_2	w_1	w_0
8	4	2	1
2	4	2	1
8	4	-2	-1

Table 1.1 - The decimal numbers in various codes

Decimal digit	Weights				Weights				Weights			
	8	4	2	1	2	4	2	1	8	4	-2	-1
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	1	1
2	0	0	1	0	0	0	1	0	0	1	1	0
3	0	0	1	1	0	0	1	1	0	1	0	1
4	0	1	0	0	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1	1	0	1	1
6	0	1	1	0	1	1	0	0	1	0	1	0
7	0	1	1	1	1	1	0	1	1	0	0	1
8	1	0	0	0	1	1	1	0	1	0	0	0
9	1	0	0	1	1	1	1	1	1	1	1	1

In all the cases only ten combinations are utilized to represent the decimal digits. The remaining six combinations are illegal. However, they may be utilized for error detection purposes.

Consider, for example, the representation of the decimal number 16.85 in Natural Binary Coded Decimal code (NBCD)

$$(16.85)_{10} = (0001\ 0110 . 1000\ 0101)_{NBCD}$$

1 6 8 5

There are many possible weights to write a number in BCD code. Some codes have desirable properties, which make them suitable for specific applications. Two such desirable properties are: 1. Self-complementing codes 2. Reflective codes When we perform arithmetic operations, it is often required to take the “complement” of a given number. If the logical complement of a coded number is also its arithmetic complement, it will be convenient from hardware

point of view. In a self-complementing coded decimal number, $(A)_{10}$, if the individual bits of a number are complemented it will result in $(9 - A)_{10}$.

Table 1.2 : Three self-complementing codes

Decimal Digit	Excess-3 Code	631-1 Code	2421 Code
0	0011	0011	0000
1	0100	0010	0001
2	0101	0101	0010
3	0110	0111	0011
4	0111	0110	0100
5	1000	1001	1011
6	1001	1000	1100
7	1010	1010	1101
8	1011	1101	1110
9	1100	1100	1111

Unit Distance Codes

There are many applications in which it is desirable to have a code in which the adjacent codes differ only in one bit. Such codes are called Unit distance Codes. "Gray code" is the most popular example of unit distance code. The 3-bit and 4-bit Gray codes are

Table 1.3 : Unit distance Codes

Decimal	3-bit Gray	4-bit Gray
0	000	0000
1	001	0001
2	011	0011
3	010	0010
4	110	0110
5	111	0111
6	101	0101
7	100	0100
8	-	1100
9	-	1101
10	-	1111
11	-	1110
12	-	1010
13	-	1011
14	-	1001
15	-	1000

Alphanumeric Codes

When information to be encoded includes entities other than numerical values, an expanded code is required. For example, alphabetic characters (A, B,Z) and special operation symbols like +, -, /, *, (,) and other special symbols are used in digital systems. Codes that include alphabetic characters are commonly referred to as Alphanumeric Codes. However, we require adequate number of bits to encode all the characters. As there was a need for alphanumeric codes in a wide variety of applications in the early era of computers, like teletype, punched tape and punched cards, there has always been a need for evolving a standard for these codes. Alphanumeric keyboard has become ubiquitous with the

popularization of personal computers and notebook computers. These keyboards use ASCII (American Standard Code for Information Interchange) code

Table 1.4 : Alpha Numeric Codes

b4	b3	b2	b1	b7 b6 b5							
				000	001	010	011	100	101	110	111
0	0	0	0	NUL	DLE	SP	0	@	P	'	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	"	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	\$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	,	7	G	W	g	w
1	0	0	0	BS	CAN	(8	H	X	h	x
1	0	0	1	HT	EM)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[k	{
1	1	0	0	FF	FS	,	<	L	\	l	
1	1	0	1	CR	GS	-	=	M]	m	}
1	1	1	0	SO	RS	.	>	N		n	~
1	1	1	1	SI	US	/	?	O	-	o	DEL

Alphanumeric codes like EBCDIC (Extended Binary Coded Decimal Interchange Code) and 12-bit Hollerith code are in use for some applications. However, ASCII code is now the standard code for most data communication networks. Therefore, the reader is urged to become familiar with the ASCII code.

EC 8392 – DIGITAL ELECTRONICS

UNIT – I : DIGITAL FUNDAMENTALS

Quine-McCluskey Method of Minimization

Karnaugh Map provides a good method of minimizing a logic function. However, it depends on our ability to observe appropriate patterns and identify the necessary implicants. If the number of variables increases beyond five, K-map or its variant Variable Entered Map can become very messy and there is every possibility of committing a mistake. What we require is a method that is more suitable for implementation on a computer, even if it is inconvenient for paper- and-pencil procedures. The concept of tabular minimisation was originally formulated by Quine in 1952. This method was later improved upon by McClusky in 1956, hence the name Quine-McClusky.

This Learning Unit is concerned with the Quine-McClusky method of minimisation. This method is tedious, time-consuming and subject to error when performed by hand. But it is better suited to implementation on a digital computer.

Principle of Quine-McClusky Method

The Quine-McClusky method is a two stage simplification process. Generate prime implicants of the given Boolean function by a special tabulation process. Determine the minimal set of implicants is determined from the set of implicants generated in the first stage. The tabulation process starts with a listing of the specified minterms for the 1s (or 0s) of a function and don't-cares (the

unspecified minterms) in a particular format. All the prime implicants are generated from them using the simple logical adjacency theorem, namely, $AB' + AB = A$. The main feature of this stage is that we work with the equivalent binary number of the product terms. For example in a four variable case, the minterms $A'BCD'$ and $A'BC'D'$ are entered as 0110 and 0100. As the two logically adjacent minterms $A'BCD'$ and $A'BC'D'$ can be combined to form a product term $A'BD'$, the two binary terms 0110 and 0100 are combined to form a term represented as "01-0", where '-' (dash) indicates the position where the combination took place. Stage two involves creating a prime implicant table. This table provides a means of identifying, by a special procedure, the smallest number of prime implicants that represents the original Boolean function. The selected prime implicants are combined to form the simplified expression in the SOP form. While we confine our discussion to the creation of minimal SOP expression of a Boolean function in the canonical form, it is easy to extend the procedure to functions that are given in the standard or any other forms.

Generation of Prime Implicants

The process of generating prime implicants is best presented through an example.

Example 1: $F = \Sigma (1,2,5,6,7,9,10,11,14)$

All the minterms are tabulated as binary numbers in sectionalised format, so that each section consists of the equivalent binary numbers containing the same number of 1s, and the number of 1s in the equivalent binary numbers of

each section is always more than that in its previous section. This process is illustrated in the table as below.

Table: 1.5 – No of 1's available in each binary

Section	Column 1		Decimal
	No. of 1s	Binary	
1	1	0001	1
		0010	2
2	2	0101	5
		0110	6
		1001	9
		1010	10
3	3	0111	7
		1011	11
		1110	14

The next step is to look for all possible combinations between the equivalent binary numbers in the adjacent sections by comparing every binary number in each section with every binary number in the next section. The combination of two terms in the adjacent sections is possible only if the two numbers differ from each other with respect to only one bit. For example 0001 (1) in section 1 can be combined with 0101 (5) in section 2 to result in 0-01 (1, 5). Notice that combinations cannot occur among the numbers belonging to the same section. The results of such combinations are entered into another column, sequentially along with their decimal equivalents indicating the binary equivalents from which the result of combination came, like (1, 5) as mentioned above. The second column also will get sectionalised based on the number of 1s. The entries of one section in the second column can again be combined together with entries

in the next section, in a similar manner. These combinations are illustrated in the Table below

Table:1.6 – 1st and 2nd level reduction

Section	Column 1			Column 2		Column 3	
	No.of 1s Decimal	Binary					
1	1	0001 ✓	1	1-01	(1,5)	--10	(2,6,10,14)
		0010 ✓	2	-001	(1,9)	--10	(2,10,6,14)
2	2	0101 ✓	5	0-10	(2,6)✓		
		0110 ✓	6	-010	(2,10) ✓		
		1001 ✓	9				
		1010 ✓	10				
3	3	0111 ✓	7	01-1	(5,7)		
		1011 ✓	11	011-	(6,7)		
		1110 ✓	14	-110	(6,14) ✓		
				10-1	(9,11)		
			101-	(10,11)			
			1-10	(10,14) ✓			

All the entries in the column which are paired with entries in the next section are checked off. Column 2 is again sectionalised with respect to the number of 1s. Column 3 is generated by pairing off entries in the first section of the column 2 with those items in the second section. In principle this pairing could continue until no further combinations can take place. All those entries that are paired can be checked off. It may be noted that combination of entries in column 2 can only take place if the corresponding entries have the dashes at the same place. This rule is applicable for generating all other columns as well. After the tabulation is completed, all those terms which are not checked off constitute the set of prime implicants of the given function. The repeated terms, like --10 in the column 3, should be eliminated. Therefore, from the above tabulation procedure, we obtain seven prime implicants (denoted by their decimal equivalents) as (1,5),

(1,9), (5,7), (6,7), (9,11), (10,11), (2,6,10,14). The next stage is to determine the minimal set of prime implicants.

Determination of the Minimal Set of Prime Implicants

The prime implicants generated through the tabular method do not constitute the minimal set. The prime implicants are represented in so called "prime implicant table". Each column in the table represents a decimal equivalent of the minterm. A row is placed for each prime implicant with its corresponding product appearing to the left and the decimal group to the right side. Asterisks are entered at those intersections where the columns of binary equivalents intersect the row that covers them. The prime implicant table for the function under consideration is shown in the figure.

www.binils.com

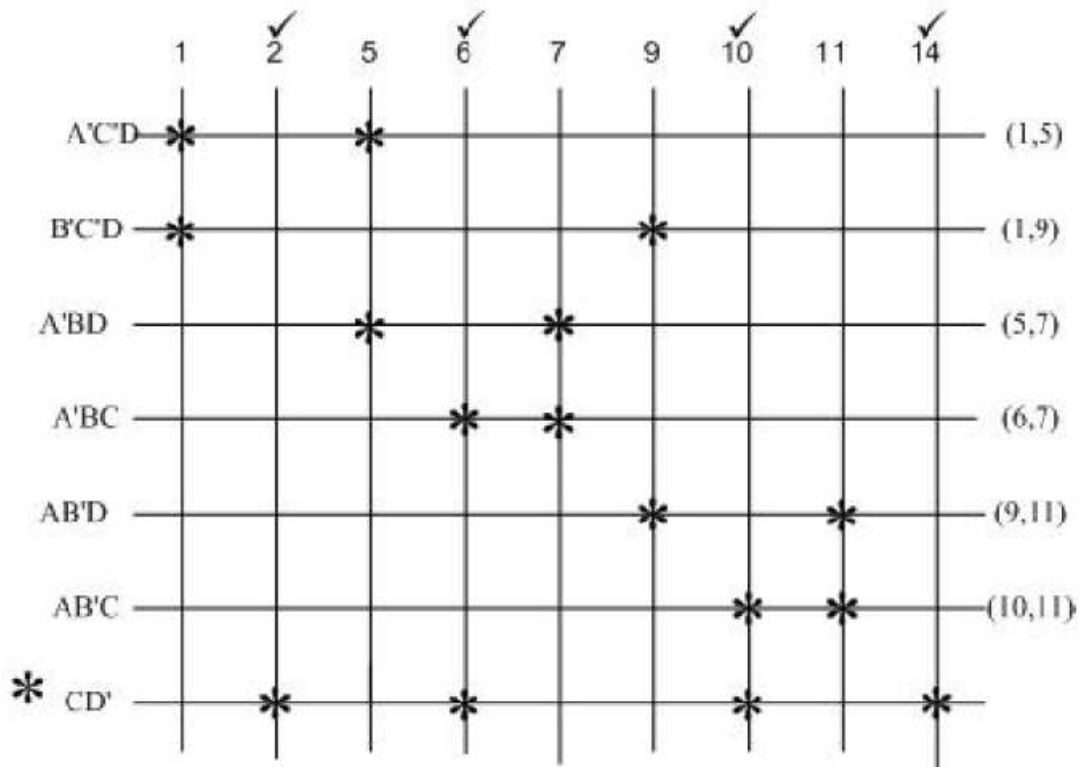


Fig 1. 1 – Prime implicants

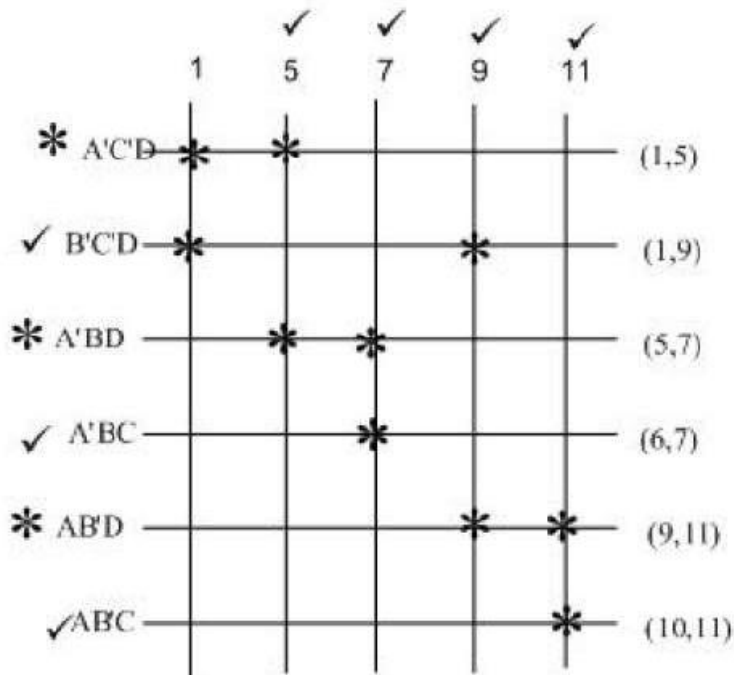


Fig 1. 2 – Essential Prime implicants mapping

We then select dominating prime implicants, which are the rows that have more asterisks than others. For example, the row $A'BD$ includes the minterm 7, which is the only one included in the row represented by $A'BC$. $A'BD$ is dominant implicant over $A'BC$, and hence $A'BC$ can be eliminated. Mark $A'BD$ by an asterisk and check off the column 5 and 7. We then choose $AB'D$ as the dominating row over the row represented by $AB'C$. Consequently, we mark the row $AB'D$ by an asterisk, and eliminate the row $AB'C$ and the columns 9 and 11 by checking them off. Similarly, we select $A'C'D$ as the dominating one over $B'C'D$. However, $B'C'D$ can also be chosen as the dominating prime implicant and eliminate the implicant $A'C'D$. Retaining $A'C'D$ as the dominant prime implicant the minimal set of prime implicants is $\{CD', A'C'D, A'BD$ and $AB'D\}$. The corresponding minimal SOP expression for the Boolean function is:

$$F = CD' + A'C'D + A'BD + AB'D$$

If we choose $B'C'D$ instead of $A'C'D$, then the minimal SOP expression for the Boolean function is:

$$F = CD' + B'C'D + A'BD + AB'D$$

EC 8392 – DIGITAL ELECTRONICS

UNIT – I : DIGITAL FUNDAMENTALS

CANONICAL AND STANDARD FORMS:

Minterms and Maxterms:

A binary variable may appear either in its normal form (x) or in its complement form (x'). Now either two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations:

$x'y'$, $x'y$, xy' and xy

Each of these four AND terms is called a '*minterm*'.

In a similar fashion, when two binary variables x and y combined with an OR operation, there are four possible combinations:

$x'+y'$, $x'+y$, $x+y'$ and $x+y$ Each of these four OR terms is called a '*maxterm*'.

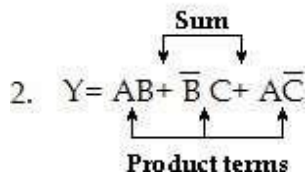
Table: 1.3 - The minterms and maxterms of a 3- variable

Variables			Minterms	Maxterms
x	y	z	m_i	M_i
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	$xyz = m_7$	$x'+y'+z' = M_7$

Sum of Minterm: (Sum of Products)

The logical sum of two or more logical product terms is called sum of products expression. It is logically an OR operation of AND operated variables such as:

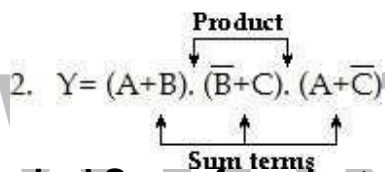
1. $Y = AB + BC + AC$



Sum of Maxterm: (Product of Sums)

A product of sums expression is a logical product of two or more logical sum terms. It is basically an AND operation of OR operated variables such as,

1. $Y = (A+B) \cdot (B+C) \cdot (A+C)$



Canonical Sum of product expression:

If each term in SOP form contains all the literals then the SOP is known as standard (or) canonical SOP form. Each individual term in standard SOP form is called minterm canonical form.

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

Steps to convert general SOP to standard SOP form:

1. Find the missing literals in each product term if any.
2. AND each product term having missing literals by ORing the literal and its complement.

- Expand the term by applying distributive law and reorder the literals in the product term.
- Reduce the expression by omitting repeated product terms if any.

Obtain the canonical SOP form of the function:

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

$$\begin{aligned} &= A \cdot (B + B') + B \cdot (A + A') = \\ &\underline{AB} + AB' + \underline{A'B} + A'B = AB + \\ &AB' + A'B. \end{aligned}$$

2. $Y(A, B, C) = A + ABC$

$$\begin{aligned} &= A \cdot (B + B') \cdot (C + C') + ABC \\ &= (AB + AB') \cdot (C + C') + ABC \\ &= \underline{ABC} + ABC' + AB'C + AB'C' + \underline{ABC} \\ &= ABC + ABC' + AB'C + AB'C' \\ &= m_7 + m_6 + m_5 + m_4 \\ &= \sum m(4, 5, 6, 7). \end{aligned}$$

3. $Y(A, B, C) = A + BC$

$$\begin{aligned} &= A \cdot (B + B') \cdot (C + C') + (A + A') \cdot BC \\ &= (AB + AB') \cdot (C + C') + ABC + A'BC \\ &= \underline{ABC} + ABC' + AB'C + AB'C' + \underline{ABC} + A'BC \\ &= ABC + ABC' + AB'C + AB'C' + A'BC \\ &= m_7 + m_6 + m_5 + m_4 + m_3 \\ &= \sum m(3, 4, 5, 6, 7). \end{aligned}$$

4. $Y(A, B, C) = AC + AB + BC$

$$\begin{aligned} &= AC (B+ B') + AB (C+ C') + BC (A+ A') \\ &= \underline{ABC} + AB'C + \underline{ABC} + ABC' + \underline{ABC} + A'BC \\ &= ABC + AB'C + ABC' + A'BC \\ &= \sum m (3, 5, 6, 7). \end{aligned}$$

5. $Y (A, B, C, D) = AB + ACD$

$$\begin{aligned} &= AB (C+ C') (D+ D') + ACD (B+ B') \\ &= (ABC + ABC') (D+ D') + ABCD + \underline{ABCD} + AB'CD = \\ &= \underline{ABCD} + ABCD' + ABC'D + \\ & \quad ABC'D' + \\ & \quad ABCD + ABCD' + ABC'D + ABC'D' + AB'CD. \end{aligned}$$

Canonical Product of sum expression:

If each term in POS form contains all literals then the POS is known as standard (or) Canonical POS form. Each individual term in standard POS form is called Maxterm canonical form.

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

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

Steps to convert general POS to standard POS form:

1. Find the missing literals in each sum term if any.
2. OR each sum term having missing literals by ANDing the literal and its complement.
3. Expand the term by applying distributive law and reorder the literals in the sum term.
4. Reduce the expression by omitting repeated sum terms if any.

Obtain the canonical POS expression of the functions:

1. **$Y = A + B'C$**

$$= (A + B') (A + C) \quad [A + BC = (A+B) (A+C)]$$

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

$$= \underline{(A + B' + C)} (A + B' + C') (A + B + C) \underline{(A + B' + C)}$$

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

$$= M_2. M_3. M_0$$

$$= \prod M (0, 2, 3)$$

2. **$Y = (A+B) (B+C) (A+C)$**

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

$$= \underline{(A+B+C)} (A+B+C') \underline{(A+B+C)} (A'+B+C) \underline{(A+B+C)} (A+B'+C)$$

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

$$= M_0. M_1. M_4. M_2$$

$$= \prod M (0, 1, 2, 4)$$

3. **$Y = A. (B + C + A)$**

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

$$= \underline{(A+B+C)} (A+B+C') (A+B'+C) (A + B' + C') \underline{(A+B+C)}$$

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

$$= M_0. M_1. M_2. M_3$$

$$= \prod M (0, 1, 2, 3)$$

4. **$Y = (A+B') (B+C) (A+C')$**

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

$$= (A+B'+C) \underline{(A+B'+C')} (A+B+C) (A'+B+C) (A+B+C') \underline{(A+B'+C')}$$

$$\begin{aligned} &= (A+B'+C) (A+B'+C') (A+B+C) (A'+B+C) (A+B+C') \\ &= M_2. M_3. M_0. M_4. M_1 \\ &= \prod M (0, 1, 2, 3, 4) \end{aligned}$$

5. **$Y = xy + x'z$**

$$\begin{aligned} &= (xy + x') (xy + z) \text{ Using distributive law, convert the function into OR terms.} \\ &= (x+x') (y+x') (x+z) (y+z) \quad [x+x'=1] \\ &= (x'+y) (x+z) (y+z) \\ &= (x'+y+z.z') (x+z+y.y') (y+z+x.x') \\ &= \underline{(x'+y+z)} (x'+y+z') \underline{(x+y+z)} (x+y'+z) \underline{(x+y+z)} (x'+y+z) \\ &= (x'+y+z) (x'+y+z') (x+y+z) (x+y'+z) \\ &= M_4. M_5. M_0. M_2 \\ &= \prod M (0, 2, 4, 5). \end{aligned}$$

www.binils.com