Representing a fun. in

$f(A,B)$ → terms

$f(A,B) = A\bar{B} + \bar{A}B$ → (SSOP form)

SSOP because each product term contain all variable, we have $A\bar{B}$ in both terms

↓

So from SSOP terms

Min term & Max term

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

(1 0) (0 1)

$= \sum m(1,3)$ → Min term

Min

Total terms = $3-1 = (2)^2 = 4$ Min terms

1 less

2 var fun

$\Sigma m(1,3)$

→ 1 is these → decode into Binary

00 → 0
01 → 1
10 → 2
11 → 3

1 → 0 1 in Binary
3 → 1 1 " "

$f(A,B) = AB + \bar{A}B$
$= \Sigma m(1,3)$
$= (A+B)(\bar{A}+B)$

eg: $f(A,B) = (A+B)(\bar{A}+B)$ (SPOS - form)
$= 0\ 0\ 1\ 0$
$= M_0\ M_2$
$= \Pi M[0,2]$

0 is these → decode in to Binary

0 → 0 0 in Binary
2 → 1 0 " "

Boolean Function Representation:

Canonical Form:

1) Standard SOP [SSOP]
- Each product term contains all the variables of the function

2) Standard POS [SPOS]
- Each sum term contains all the variables of the function

Ex $F(A,B,C) = \bar{A}BC + A\bar{B}\bar{C}$ ✓ SSOP

Ex $F(A,B,C) = AB + B\bar{C}\bar{A}$ ✓ SOP
✗ SSOP

SOP → not necessary to contain all the variable.

2) Standard POS [SPOS]

- Each term contains all the variables of the function

Ex $F(A,B,C,D) = AB\bar{C}\bar{D} + \bar{A}\bar{B}CD + A\bar{C}B\bar{D}$

⇒ Converting SSOP to SPOS.

Ex SSOP: $F(A,B,C,D) = AB\bar{C}\bar{D} + \bar{A}\bar{B}CD + A\bar{C}B\bar{D}$

SPOS: $(A+B+\bar{C}+\bar{D})(\bar{A}+\bar{B}+C+D)(A+\bar{C}+B+\bar{D})$

Ex $F(A,B,C,D) = (A+B)(C+D)$ ⇒ POS ✓
SPOS ✗

⊛ Converting SOP to SSOP:

→ Steps:

1. Identify the missing variables in product term
2. Multiply (variable + its complement)
3. Neglect the repeated terms

→ SOP

Ex: $F(A,B,C) = AB + A\bar{B}C + BC$

↓ C is missing: (ABC), $(AB\bar{C})$

↓ A is missing: (ABC) $(\bar{A}BC)$

Possible combinations

$F(A,B,C) = AB(C+\bar{C}) + A\bar{B}C + BC(A+\bar{A})$

$= ABC + AB\bar{C} + A\bar{B}C + ABC + \bar{A}BC$

$F(A,B,C) = ABC + AB\bar{C} + \bar{A}BC$ → SSOP.

Minterm of the expression
↳ How many? what are they?

SSOP is Necessary

$F(A,B,C) = ABC + AB\bar{C} + \bar{A}BC$

$F(A,B,C) = 000 + 001 + 100$

$= 111 + 110 + 011$

Binary ≅ 7 ≅ 6 ≅ 3

$m_7 \quad m_6 \quad m_3$

$= \Sigma m(3,6,7)$

000
001
010
011
100
101
110
111

Converting POS to SPOS:-

→ Steps:

1) Identify the missing variables in sum terms
2) Adding (variable + its complement)
3) Neglect the repeated terms

POS

Ex $f(A,B,C) = A(A+C)$

C, B is missing: $(A+B+C)$, $(A+\bar{B}+\bar{C})$, $(A+\bar{B}+C)$, $(A+B+\bar{C})$ — all possibility

B is missing: $(A+B+C)$, $(A+\bar{B}+C)$ — all possibility

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

SPOS $\leftarrow$ $F(A,B,C) = (A+B+C)(A+\bar{B}+C)(A+\bar{B}+\bar{C})(A+B+C)$

$= 000 \cdot 010 \cdot 011 \cdot 001$

$\simeq 0 \quad \simeq 2 \quad \simeq 3 \quad \simeq 1$

$M_0 \quad M_2 \quad M_3 \quad M_1$

$F(A,B,C) = \Sigma M(0,1,2,3)$

(*) Converting SSOP to SPOS:

SSOP → Ex $F(A,B,C) = \Sigma m(0,1,3,4,7)$ → associated minterms

$\overline{f(A,B,C)} = \Pi M(2,5,6)$

Associated Maxterms is its complement

010 101 110

Actual compile

SPOS $= (A+\bar{B}+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)$

Now,

Bestand PAGE NO.: DATE: / /

Convert the following form into SSOP form.

1) $f(A,B) = A(A+B)$ → POS. (RHS is missing)

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

$= (A+B)(A+\bar{B})$ ⟶ SPOS

M_0 M_1

~~SSOP = $A\bar{B} + A\bar{B}$~~ $= \pi M\,[0,1]$

$f(A,B) = \pi M\,[0,1]$

$\overline{f(AB)} = \Sigma m\,[2,3]$

10 11

$\overline{f(A,B)} = 10 \quad 11$

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

} 2 variable function

Scanned with CamScanner

representing the values of the n^{th} variable.]

Definitions

1. A *minterm* of n Boolean variables is a Boolean product of the n literals (variables or complements) in which each literal appears exactly once.

For example, ab, $a'b$, ab' and $a'b'$ form the complete set of minterms of two variables a and b, abc, abc', $ab'c$, $a'bc$, $ab'c'$, $a'bc'$, $a'b'c$ and $a'b'c'$ form the complete set of minterms of three variables a, b, c.

2. A *maxterm* of n Boolean variables is a Boolean sum of the n literals in which each literal appears exactlty once.
   For example, $a + b$, $a' + b$, $a + b'$ and $a' + b'$ form the complete set of maxterms in two variables a and b.
3. When a Boolean function is expressed as a sum of minterms, it is called its *sum of products expansion* or it is said to be in the *disjunctive normal form* (DNF).
4. When a Boolean function is expressed as a product of maxterms, it is called its *product of sums expansion* or it is said to be in the *conjunctive normal form* (CNF).
5. Boolean function expressed in the DNF or CNF are said to be in *canonical form.*
6. If a Boolean function in n variables is expressed as the sum (product) of all the 2^n minterms (maxterms), it is said to be in *complete DNF* (*complete CNF*).
7. Boolean functions expressed in complete DNF or complete CNF are said to be *complete canonical form.*

EXPRESSION OF A BOOLEAN FUNCTION IN CANONICAL FORM

1. Truth Table Method

If the Boolean function $f(x, y, z)$ is represented by a truth table, we express $f(x, y, z)$ in DNF as follows:

We note down the rows in which 'f' column entry is 1. The DNF of f is the Boolean sum of the minterms corresponding to the literals in those rows. While forming the minterm corresponding to a row, 1 entry is replaced by the corresponding variable and 0 entry is replaced by the complement of the variable concerned.

For example, let us consider the function $f(x, y, z)$ whose truth table representation is given as follows:

x	y	z	f
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

1's occur in the 'f' column against the second, third and fourth rows. The minterm corrsponding to the second row is xyz', since 1 occurs in each of the x column and y column and 0 occurs in the z column. Similarly the minterms corresponding to the third and fourth rows are $xy'z$ and $xy'z'$ respectively.

Since f is the Boolean sum of these three minterms, the required DNF of f is

$$xyz' + xy'z + xy'z'$$

The CNF of $f(x, y, z)$ represented by a truth table is obtained as follows:

We note down the rows in which the 'f' column entry is 0. The CNF of f is the Boolean product of the maxterms corresponding to the literals in those rows. While forming the maxterm corresponding to a row, 0 entry is replaced by the corresponding variable and 1 entry is replaced by the complement of the variable concerned.

In the above example, 0's occur in the 'f' column against the 1st row and the fifth to the eighth rows. The maxterm corresponding to the first row is $(x' + y' + z')$, since 1 occurs under each of x, y, z.

Similarly the maxterms corrsponding to the other rows are written.

The CNF of f is the Boolean product of these maxterms.

$$\therefore \quad f = (x' + y' + z')\,(x + y' + z')\,(x + y' + z)\,(x + y + z')\,(x + y + z).$$

2. Algebraic Method

To get the DNF of a given Boolean function, we express it as a sum of products. Then each product is multiplied in Boolean sense by $a + a'$, which is equal to 1, if a is the missing literal and simplified. In the end if a product term is repeated, the repetition is avoided since $a + a = a$.

To get the CNF of a given Boolean function, we express it as a product of sums. Then to each sum is added in Boolean sense the term aa', which is equal to 0, if a is the missing literal and simplified. In the end if a sum factor is repeated, the repetition is avoided since $a \cdot a = a$. For example, let us consider the Boolean function $f(x, y, z) = x(y' + z')$ and express it in the sum of products canonical form:

$$\begin{aligned} f &= xy' + xz' \\ &= xy' \cdot (z + z') + xz'(y + y') \quad \{\because z \text{ is the missing literal in the first product and } y \text{ is the missing literal in the second product.}\} \\ &= xy'z + xy'z' + xyz' + xy'z' \\ &= xy'z + xy'z' + xyz' \quad (\because xy'z' \text{ is repeated}) \end{aligned}$$

Now let us express the same function in the product of sums canonical form.

$$\begin{aligned} f &= x \cdot (y' + z') \\ &= (x + yy') \cdot (y' + z' + xx') \\ &= (x + y) \cdot (x + y') \cdot (y' + z' + x)\,(y' + z' + x') \\ &= (x + y + zz')\,(x + y' + zz')\,(x + y' + z')\,(x' + y' + z') \\ &= (x + y + z) \cdot (x + y + z') \cdot (x + y' + z)\,(x + y' + z') \\ &\quad (x + y' + z')\,(x' + y' + z') \\ &= (x + y + z) \cdot (x + y + z') \cdot (x + y' + z) \cdot (x + y' + z')\,(x' + y' + z') \end{aligned}$$

(repetition avoided)

LOGIC GATES

A computer or any other electronic device is made up of a number of circuits. Boolean algebra can be used to design the circuits of electronic devices. The basic elements of circuits are solid state devices called *gates*, that implement Boolean operations. The circuits that we consider in this section give the output that depends only on the input and not on the current state of the circuit. In other words these circuits have no memory capabilities. Such circuits are called *combinational circuits gating networks*.

We shall now consider three basic types of gates that are used to construct combinational circuits:

1. *OR gate:* This gate receives two or more inputs (Boolean variables) and produces an output equal to the Boolean sum of the values of the input variables. The symbol used for an OR gate is shown in Fig. 2.29(a). The inputs are shown on the left side entering the symbol and the output on the right side leaving the symbol.

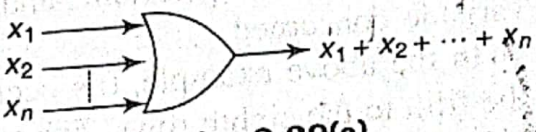


Fig. 2.29(a)

2. *AND gate:* This gate receives two or more inputs (Boolean variables) and produces an outut equal to their Boolean product. The symbol used for an AND gate is shown in Fig. 2.29(b).

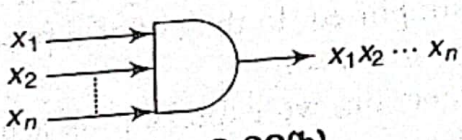


Fig. 2.29(b)

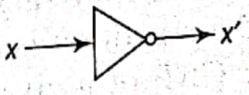


Fig. 2.29(c)

3. *NOT gate or Invertor:* This gate accepts only one input (value of one Boolean variable) and produces the complement of this value as the output. The symbol for this NOT gate is shown in Fig. 2.29(c).

COMBINATION OF GATES

Combinational circuits are formed by interconnecting the basic gates. When such circuits are formed, some gates may share inputs. One method is to indicate the inputs separately for each gate [Fig. 2.30(a)]. The other method is to use branchings that indicate all the gates that use a given input [Fig. 2.30(b)];

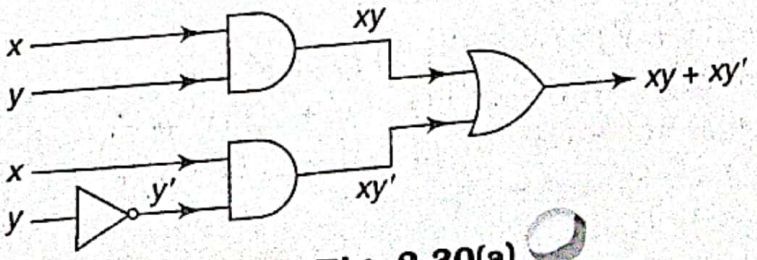


Fig. 2.30(a)

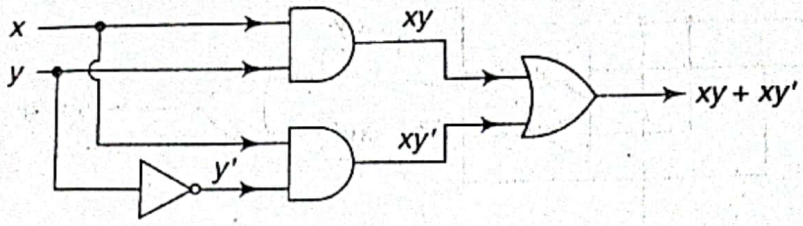


Fig. 2.30(b)

Thus we can compute the value of the output y by tracing the flow through the circuit symbolically from left to right as in the following example. [See Fig. 2.30(c)].

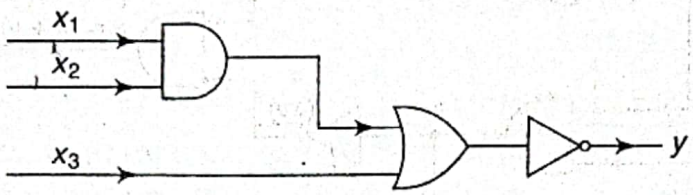


Fig. 2.30(c)

First the Boolean product of x_1 and x_2 is obatined as $x_1 \cdot x_2$. This output is Boolean added with x_3 to produce $x_1x_2 + x_3$. This output is conmplemented to produce the final output $y = (x_1x_2 + x_3)'$.

ADDERS

We shall consider two examples of circuits that perform some useful functions. First we consider a *half adder* that is a logic circuit used to find $x + y$, where x and y are two bits each of which has the value 0 or 1. The output will consist of two bits, namely the sum bits and carry bit c. Circuits of this type having more than one output are called *multiple output circuits*. The truth table for the half adder is given as follows:

Inputs		*Outputs*	
x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, we get $s = xy' + x'y$ and $c = xy$. The half adder circuit is given in Fig. 2.31(a).

If we observe that

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

the half adder circuit can be simplified with only four gates as shown in Fig. 2.31(b).

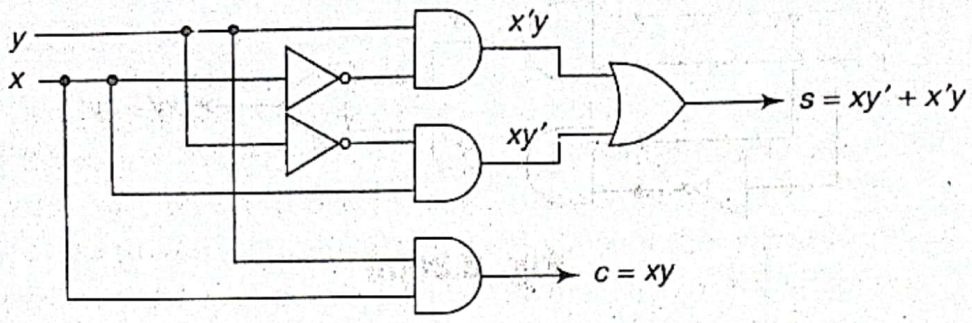


Fig. 2.31(a)

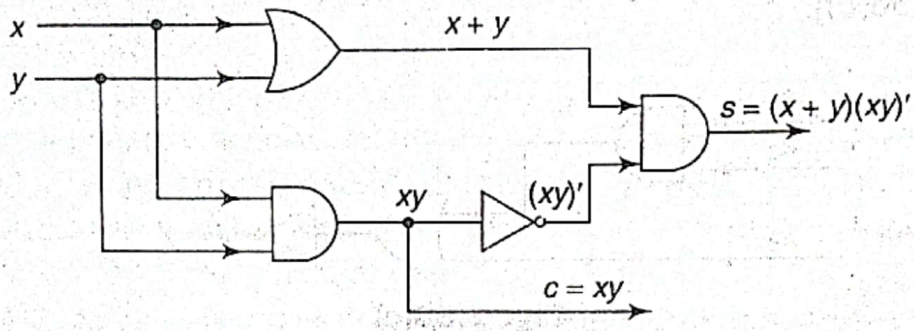


Fig. 2.31(b)

A full adder accepts three bits x, y, z as input and produces two output bits s (sum bit) and c (carry bit). The truth table for the full order is given as follows:

Inputs			*Outputs*	
x	y	z	c	s
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

From the truth table, we get

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

and

$$c = xyz + xyz' + xy'z + x'yz$$

If we observe that

$$\begin{aligned} c &= xyz + xyz' + xy'z + x'yz \\ &= (xyz + xyz') + (xyz + xy'z) + (xyz + x'yz) \\ &= xy(z + z') + zx(y + y') + yz(x + x') \\ &= xy + yz + zx, \end{aligned}$$

the circuit for the full adder can be drawn as given in Fig. 2.32.

Note If we simplify s, we can draw the circuit for full adder in a simpler way using lesser number of gates.

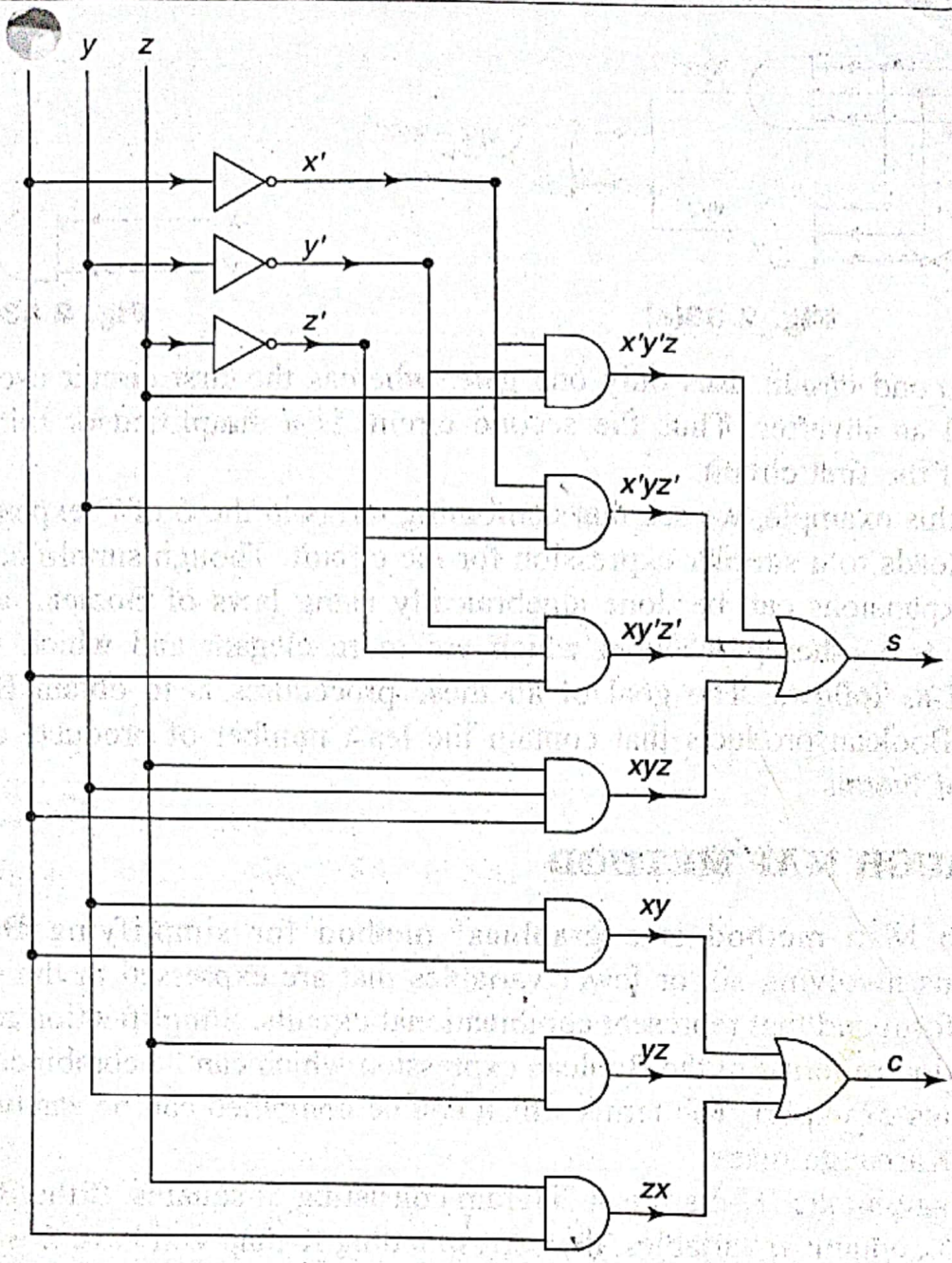


Fig. 2.32

Minimisation of Circuits/Boolean Functions

The important use of Boolean algebra is to express circuit design problems in a simplified form that is more readily understood. The efficiency of a combinatorial circuit depends on the number of gates used and on the manner of arranging them, because the cost of a circuit depends on the number of gates in the circuit to a certain extent.

For example, let us consider the following circuit, the output of which is $xyz + xyz'$, that is in the sum of products form.

Since the two products in this example differ in only one variable, namely z, they can be combined as follows:

$$\begin{aligned} xyz + xyz' &= xy(z + z') \\ &= xy \cdot 1 \\ &= xy \end{aligned}$$

The circuit for the simplified function xy is shown in Fig. 2.33(b).

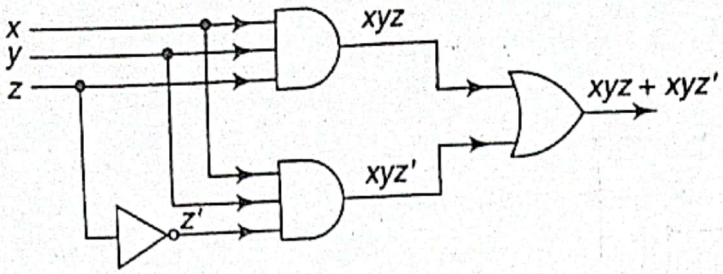


Fig. 2.33(a)

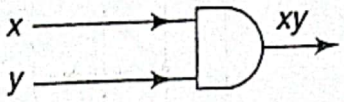


Fig. 2.33(b)

The second circuit uses only one gate, whereas the first circuit used three gates and an inverter. Thus the second circuit is a simplified or minimised version of the first circuit.

From this example, we see that combining terms in the S of P expansion of a circuit leads to a simpler expression for the circuit. Though simplification of S of P expansions can be done algebraically using laws of Boolean algebra, there are two other procedures which are more elegant and which will be described as follows. The goal of all these procedures is to obtain Boolean sums of Boolean products that contain the least number of products of least number of literals.

KARNAUGH MAP METHOD

Karnaugh Map method is a graphical method for simplifying Boolean expressions involving six or fewer variables that are expressed in the sum of products form and that represent combinational circuits. Simplification requires identification of terms in the Boolean expression which can be combined (as in the previous example). The terms which can be combined can be easily found out from Karnaugh maps.

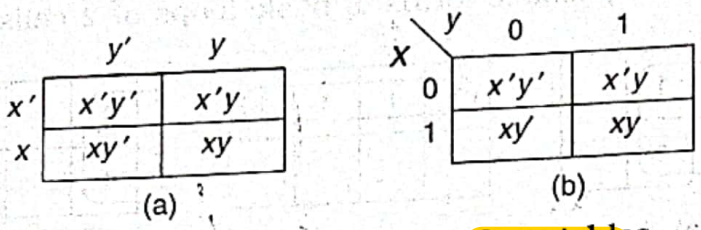
A Karnaugh map (K-map) is a diagram consisting of squares. If the Boolean expression contains n variables, the corresponding K-map will have 2^n squares, each of which represents a minterm. A '1' is placed in the square representing a minterm if it is present in the given expression. A '0' is placed in the square that corresponds to the minterm not present in the expression. The simplified Boolean expression that represents the output is then obtained by combining or grouping adjacent squares that contain 1. Adjacent squares are those that represent minterms differing by only one litreral.

To identify adjacent cells (squares) in the K-map for grouping, the following points may be borne in mind:

1. The number of cells in a group must be a power of 2, i.e., 2, 4, 8, 16, etc.
2. A cell containing 1 may be included in any number of groups.
3. To minimise the expression to the maximum possible extent, largest possible groups must be preferred. viz., a group of two cells should not be considered, if these cells can be included in a group of four cells and so on.
4. Adjacent cells exist not only within the interior of the K-map, but also at the extremes of each column and each row vi[illegible]e top cell in any column is adjacent to the bottom cell in the same column. The left most cell in

any row is adjacent to the rightmost cell in that row. [see Fig. 2.37 and 2.38]

Karnaugh maps for 2, 3 and 4 variables in two forms for each are given in Figs. 2.34, 2.35 and 2.36. The minterms which the cells represent are written within the cells.



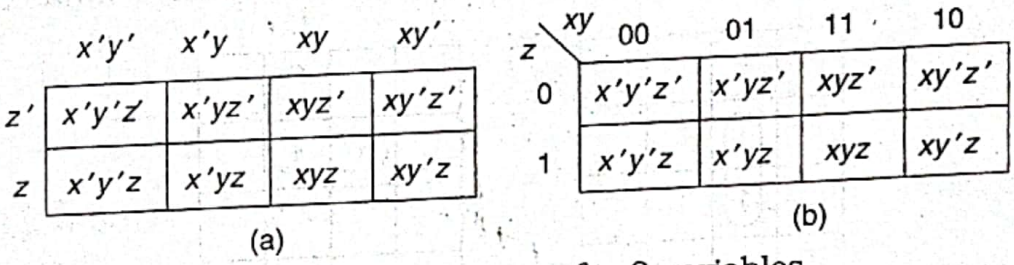
	y'	y
x'	x'y'	x'y
x	xy'	xy

(a)

x \ y	0	1
0	x'y'	x'y
1	xy'	xy

(b)

Fig. 2.34 K-map for 2 variables



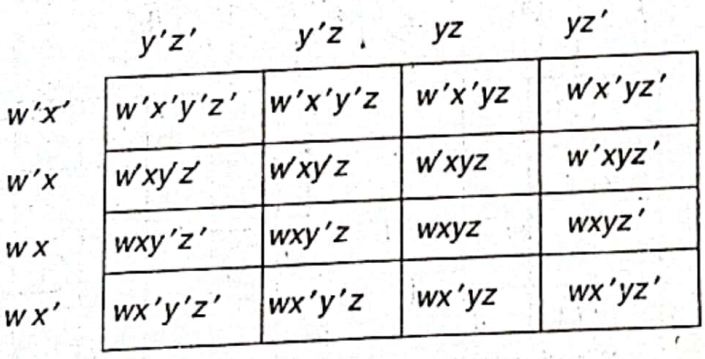
	x'y'	x'y	xy	xy'
z'	x'y'z'	x'yz'	xyz'	xy'z'
z	x'y'z	x'yz	xyz	xy'z

(a)

z \ xy	00	01	11	10
0	x'y'z'	x'yz'	xyz'	xy'z'
1	x'y'z	x'yz	xyz	xy'z

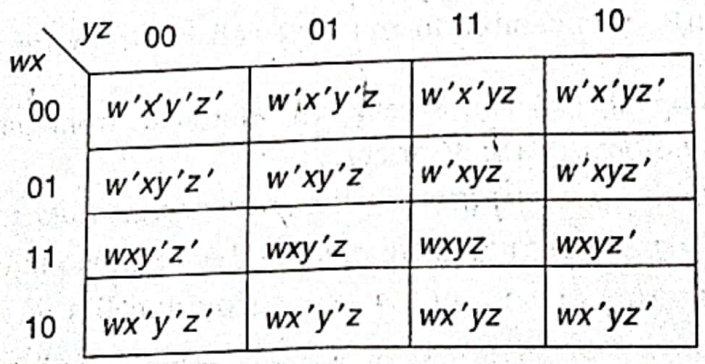
(b)

Fig. 2.35 K-map for 3 variables



	y'z'	y'z	yz	yz'
w'x'	w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
w'x	w'xy'z'	w'xy'z	w'xyz	w'xyz'
wx	wxy'z'	wxy'z	wxyz	wxyz'
wx'	wx'y'z'	wx'y'z	wx'yz	wx'yz'

(a)



wx \ yz	00	01	11	10
00	w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
01	w'xy'z'	w'xy'z	w'xyz	w'xyz'
11	wxy'z'	wxy'z	wxyz	wxyz'
10	wx'y'z'	wx'y'z	wx'yz	wx'yz'

(b)

Fig. 2.36 K-map for 4 Variables

While minimising Boolean expressions by K-map method, it will be advantageous if we are familiar with patterns of adjacent cells and groups of 1's, that will be enclosed by loops. All the basic patterns are given as follows for 3 and 4 variable K maps.

$y' + z')$