6

Boolean Algebra and Simplification Techniques

Boolean algebra is mathematics of logic. It is one of the most basic tools available to the logic designer and thus can be effectively used for simplification of complex logic expressions. Other useful and widely used techniques based on Boolean theorems include the use of Karnaugh maps in what is known as the mapping method of logic simplification and the tabular method given by Quine–McCluskey. In this chapter, we will have a closer look at the different postulates and theorems of Boolean algebra and their applications in minimizing Boolean expressions. We will also discuss at length the mapping and tabular methods of minimizing fairly complex and large logic expressions.

6.1 Introduction to Boolean Algebra

Boolean algebra, quite interestingly, is simpler than ordinary algebra. It is also composed of a set of symbols and a set of rules to manipulate these symbols. However, this is the only similarity between the two. The differences are many. These include the following:

1. In ordinary algebra, the letter symbols can take on any number of values including infinity. In Boolean algebra, they can take on either of two values, that is, 0 and 1.
2. The values assigned to a variable have a numerical significance in ordinary algebra, whereas in its Boolean counterpart they have a logical significance.
3. While '.' and '+' are respectively the signs of multiplication and addition in ordinary algebra, in Boolean algebra '.' means an AND operation and '+' means an OR operation. For instance, $A + B$ in ordinary algebra is read as A plus B, while the same in Boolean algebra is read as A OR B. Basic logic operations such as AND, OR and NOT have already been discussed at length in Chapter 4.

Digital Electronics: Principles, Devices and Applications Anil K. Maini
 ISBN: 978-0-470-03214-5

4. More specifically, Boolean algebra captures the essential properties of both logic operations such as AND, OR and NOT and set operations such as intersection, union and complement. As an illustration, the logical assertion that both a statement and its negation cannot be true has a counterpart in set theory, which says that the intersection of a subset and its complement is a null (or empty) set.
5. Boolean algebra may also be defined to be a set A supplied with two binary operations of logical AND (Λ), logical OR (V), a unary operation of logical NOT ($\neg$) and two elements, namely logical FALSE (0) and logical TRUE (1). This set is such that, for all elements of this set, the postulates or axioms relating to the associative, commutative, distributive, absorption and complementation properties of these elements hold good. These postulates are described in the following pages.

6.1.1 Variables, Literals and Terms in Boolean Expressions

Variables are the different symbols in a Boolean expression. They may take on the value '0' or '1'. For instance, in expression (6.1), A, B and C are the three variables. In expression (6.2), P, Q, R and S are the variables:

$$\overline{A} + A.B + A.\overline{C} + \overline{A}.B.C \tag{6.1}$$

$$(\overline{P} + Q).(R + \overline{S}).(P + \overline{Q} + R) \tag{6.2}$$

The complement of a variable is not considered as a separate variable. Each occurrence of a variable or its complement is called a *literal*. In expressions (6.1) and (6.2) there are eight and seven literals respectively. A term is the expression formed by literals and operations at one level. Expression (6.1) has five terms including four AND terms and the OR term that combines the first-level AND terms.

6.1.2 Equivalent and Complement of Boolean Expressions

Two given Boolean expressions are said to be *equivalent* if one of them equals '1' only when the other equals '1' and also one equals '0' only when the other equals '0'. They are said to be the *complement* of each other if one expression equals '1' only when the other equals '0', and vice versa. The complement of a given Boolean expression is obtained by complementing each literal, changing all '.' to '+' and all '+' to '.', all 0s to 1s and all 1s to 0s. The examples below give some Boolean expressions and their complements:

Given Boolean expression

$$\overline{A}.B + A.\overline{B} \tag{6.3}$$

Corresponding complement

$$(A + \overline{B}).(\overline{A} + B) \tag{6.4}$$

Given Boolean expression

$$(A + B).(\overline{A} + \overline{B}) \tag{6.5}$$

Corresponding complement

$$\overline{A}.\overline{B} + A.B \tag{6.6}$$

When ORed with its complement the Boolean expression yields a '1', and when ANDed with its complement it yields a '0'. The '.' sign is usually omitted in writing Boolean expressions and is implied merely by writing the literals in juxtaposition. For instance, $A.B$ would normally be written as AB.

6.1.3 Dual of a Boolean Expression

The dual of a Boolean expression is obtained by replacing all '.' operations with '+' operations, all '+' operations with '.' operations, all 0s with 1s and all 1s with 0s and leaving all literals unchanged. The examples below give some Boolean expressions and the corresponding dual expressions:

Given Boolean expression

$$\overline{A}.B + A.\overline{B} \tag{6.7}$$

Corresponding dual

$$(\overline{A} + B).(A + \overline{B}) \tag{6.8}$$

Given Boolean expression

$$(A + B).(\overline{A} + \overline{B}) \tag{6.9}$$

Corresponding dual

$$A.B + \overline{A}.\overline{B} \tag{6.10}$$

Duals of Boolean expressions are mainly of interest in the study of Boolean postulates and theorems. Otherwise, there is no general relationship between the values of dual expressions. That is, both of them may equal '1' or '0'. One may even equal '1' while the other equals '0'. The fact that the dual of a given logic equation is also a valid logic equation leads to many more useful laws of Boolean algebra. The principle of duality has been put to ample use during the discussion on postulates and theorems of Boolean algebra. The postulates and theorems, to be discussed in the paragraphs to follow, have been presented in pairs, with one being the dual of the other.

Example 6.1

Find (a) the dual of $A.\overline{B} + B.\overline{C} + C.\overline{D}$ *and (b) the complement of* $[(A.\overline{B} + \overline{C}).D + \overline{E}].F$.

Solution

(a) The dual of $A.\overline{B} + B.\overline{C} + C.\overline{D}$ is given by $(A + \overline{B}).(B + \overline{C}).(C + \overline{D})$.

(b) The complement of $[(A.\overline{B} + \overline{C}).D + \overline{E}].F$ is given by $[(\overline{A} + B).C + \overline{D}].E + \overline{F}$.

Example 6.2

Simplify $(A.B + C.D).[(\overline{A} + \overline{B}).(\overline{C} + \overline{D})]$.

Solution

- Let $(A.B + C.D) = X$.
- Then the given expression reduces to $X.\overline{X}$.
- Therefore, $(A.B + C.D).[(\overline{A} + \overline{B}).(\overline{C} + \overline{D})] = 0$.

6.2 Postulates of Boolean Algebra

The following are the important postulates of Boolean algebra:

1. $1.1 = 1, 0 + 0 = 0$.
2. $1.0 = 0.1 = 0, 0 + 1 = 1 + 0 = 1$.
3. $0.0 = 0, 1 + 1 = 1$.
4. $\overline{1} = 0$ and $\overline{0} = 1$.

Many theorems of Boolean algebra are based on these postulates, which can be used to simplify Boolean expressions. These theorems are discussed in the next section.

6.3 Theorems of Boolean Algebra

The theorems of Boolean algebra can be used to simplify many a complex Boolean expression and also to transform the given expression into a more useful and meaningful equivalent expression. The theorems are presented as pairs, with the two theorems in a given pair being the dual of each other. These theorems can be very easily verified by the method of 'perfect induction'. According to this method, the validity of the expression is tested for all possible combinations of values of the variables involved. Also, since the validity of the theorem is based on its being true for all possible combinations of values of variables, there is no reason why a variable cannot be replaced with its complement, or vice versa, without disturbing the validity. Another important point is that, if a given expression is valid, its dual will also be valid. Therefore, in all the discussion to follow in this section, only one of the theorems in a given pair will be illustrated with a proof. Proof of the other being its dual is implied.

6.3.1 Theorem 1 (Operations with '0' and '1')

$$\text{(a) } 0.X = 0 \quad \text{and} \quad \text{(b) } 1 + X = 1 \tag{6.11}$$

where X is not necessarily a single variable – it could be a term or even a large expression.

Theorem 1(a) can be proved by substituting all possible values of X, that is, 0 and 1, into the given expression and checking whether the LHS equals the RHS:

- For $X = 0$, LHS $= 0.X = 0.0 = 0 =$ RHS.
- For $X = 1$, LHS $= 0.1 = 0 =$ RHS.

Thus, $0.X = 0$ irrespective of the value of X, and hence the proof.

Theorem 1(b) can be proved in a similar manner. In general, according to theorem 1, 0.(Boolean expression) = 0 and 1 + (Boolean expression) = 1. For example, $0.(A.B + B.C + C.D) = 0$ and $1 + (A.B + B.C + C.D) = 1$, where A, B and C are Boolean variables.

6.3.2 Theorem 2 (Operations with '0' and '1')

$$\text{(a) } 1.X = X \quad \text{and} \quad \text{(b) } 0 + X = X \tag{6.12}$$

where X could be a variable, a term or even a large expression. According to this theorem, ANDing a Boolean expression to '1' or ORing '0' to it makes no difference to the expression:

- For $X = 0$, LHS $= 1.0 = 0 =$ RHS.
- For $X = 1$, LHS $= 1.1 = 1 =$ RHS.

Also, 1.(Boolean expression) = Boolean expression and 0 + (Boolean expression) = Boolean expression. For example,

$$1.\ (A + B.C + C.D) = 0 + (A + B.C + C.D) = A + B.C + C.D.$$

6.3.3 Theorem 3 (Idempotent or Identity Laws)

$$\text{(a) } X.X.X.\ldots.X = X \quad \text{and} \quad \text{(b)} X + X + X + \cdots + X = X \tag{6.13}$$

Theorems 3(a) and (b) are known by the name of *idempotent laws*, also known as *identity laws*. Theorem 3(a) is a direct outcome of an AND gate operation, whereas theorem 3(b) represents an OR gate operation when all the inputs of the gate have been tied together. The scope of idempotent laws can be expanded further by considering X to be a term or an expression. For example, let us apply idempotent laws to simplify the following Boolean expression:

$$\begin{aligned}(A.\overline{B}.\overline{B} + C.C).(A.\overline{B}.\overline{B} + A.\overline{B} + C.C) &= (A.\overline{B} + C).(A.\overline{B} + A.\overline{B} + C) \\ &= (A.\overline{B} + C).(A.\overline{B} + C) = A.\overline{B} + C\end{aligned}$$

6.3.4 Theorem 4 (Complementation Law)

$$\text{(a) } X.\overline{X} = 0 \quad \text{and} \quad \text{(b) } X + \overline{X} = 1 \tag{6.14}$$

According to this theorem, in general, any Boolean expression when ANDed to its complement yields a '0' and when ORed to its complement yields a '1', irrespective of the complexity of the expression:

- For $X = 0$, $\overline{X} = 1$. Therefore, $X.\overline{X} = 0.1 = 0$.
- For $X = 1$, $\overline{X} = 0$. Therefore, $X.\overline{X} = 1.0 = 0$.

Hence, theorem 4(a) is proved. Since theorem 4(b) is the dual of theorem 4(a), its proof is implied.

The example below further illustrates the application of complementation laws:

$$(A + B.C)(\overline{A + B.C}) = 0 \quad \text{and} \quad (A + B.C) + (\overline{A + B.C}) = 1$$

Example 6.3

Simplify the following:

$$[1 + L.M + L.\overline{M} + \overline{L}.M].[(L + \overline{M}).(\overline{L}.M) + \overline{L}.\overline{M}.(L + M)].$$

Solution

- We know that (1 + Boolean expression) = 1.
- Also, $(\overline{L}.M)$ is the complement of $(L + \overline{M})$ and $(\overline{L}.\overline{M})$ is the complement of $(L + M)$.
- Therefore, the given expression reduces to $1.(0 + 0) = 1.0 = 0$.

6.3.5 Theorem 5 (Commutative Laws)

$$\text{(a) } X + Y = Y + X \quad \text{and} \quad \text{(b) } X.Y = Y.X \tag{6.15}$$

Theorem 5(a) implies that the order in which variables are added or ORed is immaterial. That is, the result of A OR B is the same as that of B OR A. Theorem 5(b) implies that the order in which variables are ANDed is also immaterial. The result of A AND B is same as that of B AND A.

6.3.6 Theorem 6 (Associative Laws)

$$\text{(a) } X + (Y + Z) = Y + (Z + X) = Z + (X + Y)$$

and

$$\text{(b) } X.(Y.Z) = Y.(Z.X) = Z.(X.Y) \tag{6.16}$$

Theorem 6(a) says that, when three variables are being ORed, it is immaterial whether we do this by ORing the result of the first and second variables with the third variable or by ORing the first variable with the result of ORing of the second and third variables or even by ORing the second variable with the result of ORing of the first and third variables. According to theorem 6(b), when three variables are being ANDed, it is immaterial whether you do this by ANDing the result of ANDing of the first and second variables with the third variable or by ANDing the result of ANDing of the second and third variables with the first variable or even by ANDing the result of ANDing of the third and first variables with the second variable.

For example,

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

Also

$$\overline{A}.B.(C.\overline{D}.\overline{E}.\overline{F}) = C.\overline{D}.(\overline{A}.B.\overline{E}.\overline{F}) = \overline{E}.\overline{F}.(\overline{A}.B.C.\overline{D})$$

Theorems 6(a) and (b) are further illustrated by the logic diagrams in Figs 6.1(a) and (b).

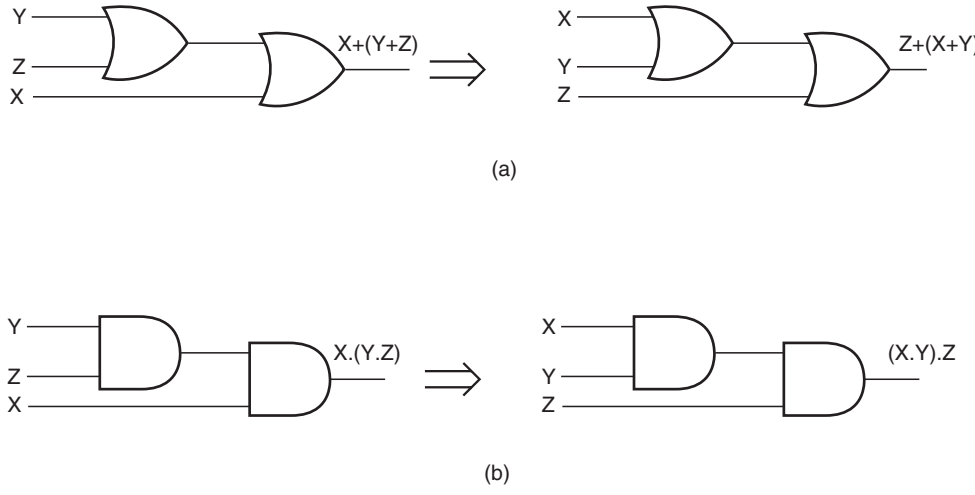

Figure 6.1 Associative laws.

6.3.7 Theorem 7 (Distributive Laws)

$$\text{(a) } X.(Y+Z) = X.Y + X.Z \quad \text{and} \quad \text{(b) } X + Y.Z = (X+Y).(X+Z) \tag{6.17}$$

Theorem 7(b) is the dual of theorem 7(a). The distribution law implies that a Boolean expression can always be expanded term by term. Also, in the case of the expression being the sum of two or more than two terms having a common variable, the common variable can be taken as common as in the case of ordinary algebra. Table 6.1 gives the proof of theorem 7(a) using the method of perfect induction. Theorem 7(b) is the dual of theorem 7(a) and therefore its proof is implied. Theorems 7(a) and (b) are further illustrated by the logic diagrams in Figs 6.2(a) and (b). As an illustration, theorem 7(a) can be used to simplify $\overline{A}.\overline{B} + \overline{A}.B + A.\overline{B} + A.B$ as follows:

$$\overline{A}.\overline{B} + \overline{A}.B + A.\overline{B} + A.B = \overline{A}.(\overline{B}+B) + A.(\overline{B}+B) = \overline{A}.1 + A.1 = \overline{A} + A = 1$$

Table 6.1 Proof of distributive law.

X	Y	Z	Y+Z	XY	XZ	X(Y+Z)	XY+XZ
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1

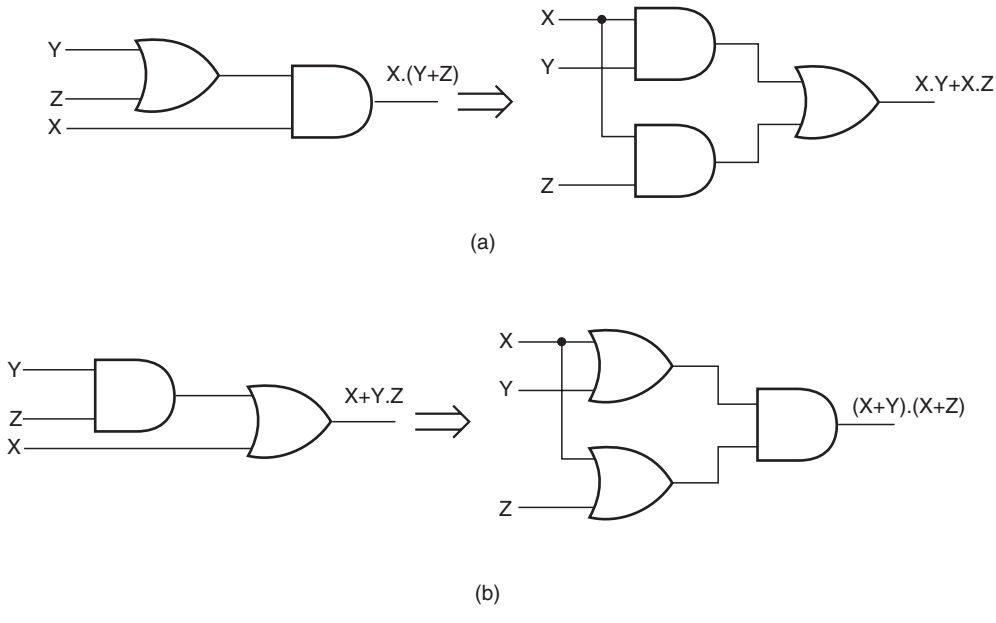

Figure 6.2 Distributive laws.

Theorem 7(b) can be used to simplify $(\overline{A}+\overline{B}).(\overline{A}+B).(A+\overline{B}).(A+B)$ as follows:

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

6.3.8 Theorem 8

$$\text{(a) } X.Y + X.\overline{Y} = X \quad \text{and} \quad \text{(b) } (X+Y).(X+\overline{Y}) = X$$

This is a special case of theorem 7 as

$$X.Y + X.\overline{Y} = X.(Y+\overline{Y}) = X.1 = X \quad \text{and} \quad (X+Y).(X+\overline{Y}) = X + Y.\overline{Y} = X + 0 = X$$

This theorem, however, has another very interesting interpretation. Referring to theorem 8(a), there are two two-variable terms in the LHS expression. One of the variables, Y, is present in all possible combinations in this expression, while the other variable, X, is a common factor. The expression then reduces to this common factor. This interpretation can be usefully employed to simplify many a complex Boolean expression.

As an illustration, let us consider the following Boolean expression:

$$A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.\overline{D} + A.\overline{B}.C.D + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D + A.B.C.\overline{D} + A.B.C.D$$

In the above expression, variables B, C and D are present in all eight possible combinations, and variable A is the common factor in all eight product terms. With the application of theorem 8(a), this expression reduces to A. Similarly, with the application of theorem 8(b), $(A+\overline{B}+\overline{C}).(A+\overline{B}+C).(A+B+\overline{C}).(A+B+C)$ also reduces to A as the variables B and C are present in all four possible combinations in sum terms and variable A is the common factor in all the terms.

6.3.9 Theorem 9

$$\text{(a) } (X+\overline{Y}).Y = X.Y \quad \text{and} \quad \text{(b) } X.\overline{Y}+Y = X+Y \tag{6.18}$$

$$(X+\overline{Y}).Y = X.Y+\overline{Y}.Y = X.Y$$

Theorem 9(b) is the dual of theorem 9(a) and hence stands proved.

6.3.10 Theorem 10 (Absorption Law or Redundancy Law)

$$\text{(a) } X+X.Y = X \quad \text{and} \quad \text{(b) } X.(X+Y) = X \tag{6.19}$$

The proof of absorption law is straightforward:

$$X+X.Y = X.(1+Y) = X.1 = X$$

Theorem 10(b) is the dual of theorem 10(a) and hence stands proved.

The crux of this simplification theorem is that, if a smaller term appears in a larger term, then the larger term is redundant. The following examples further illustrate the underlying concept:

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

and

$$(\overline{A}+B+\overline{C}).(\overline{A}+B).(C+B+\overline{A}) = \overline{A}+B$$

6.3.11 Theorem 11

$$\text{(a) } Z.X+Z.\overline{X}.Y = Z.X+Z.Y$$

and

$$\text{(b) } (Z+X).(Z+\overline{X}+Y) = (Z+X).(Z+Y) \tag{6.20}$$

Table 6.2 gives the proof of theorem 11(a) using the method of perfect induction. Theorem 11(b) is the dual of theorem 11(a) and hence stands proved. A useful interpretation of this theorem is that, when

Table 6.2 Proof of theorem 11(a).

X	Y	Z	ZX	ZY	$Z\overline{X}$	$Z\overline{X}Y$	$ZX+Z\overline{X}Y$	ZX+ZY
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0
0	1	1	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	1	1
1	1	0	0	0	0	0	0	0
1	1	1	1	1	0	0	1	1

a smaller term appears in a larger term except for one of the variables appearing as a complement in the larger term, the complemented variable is redundant.

As an example, $(A+\overline{B}).(\overline{A}+\overline{B}+C).(\overline{A}+\overline{B}+D)$ can be simplified as follows:

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

6.3.12 Theorem 12 (Consensus Theorem)

$$\text{(a) } X.Y+\overline{X}.Z+Y.Z = X.Y+\overline{X}.Z$$

and

$$\text{(b) } (X+Y).(\overline{X}+Z).(Y+Z) = (X+Y).(\overline{X}+Z) \tag{6.21}$$

Table 6.3 shows the proof of theorem 12(a) using the method of perfect induction. Theorem 12(b) is the dual of theorem 12(a) and hence stands proved.

A useful interpretation of theorem 12 is as follows. If in a given Boolean expression we can identify two terms with one having a variable and the other having its complement, then the term that is formed by the product of the remaining variables in the two terms in the case of a sum-of-products expression

Table 6.3 Proof of theorem 12(a).

X	Y	Z	XY	$\overline{X}Z$	YZ	$XY+\overline{X}Z+YZ$	$XY+\overline{X}Z$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1
0	1	0	0	0	0	0	0
0	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	1	0	1	1	1

or by the sum of the remaining variables in the case of a product-of-sums expression will be redundant. The following example further illustrates the point:

$$A.B.C + \overline{A}.C.D + \overline{B}.C.D + B.C.D + A.C.D = A.B.C + \overline{A}.C.D + \overline{B}.C.D$$

If we consider the first two terms of the Boolean expression, $B.C.D$ becomes redundant. If we consider the first and third terms of the given Boolean expression, $A.C.D$ becomes redundant.

Example 6.4

Prove that $A.B.C.D + A.B.\overline{C}.\overline{D} + A.B.C.\overline{D} + A.B.\overline{C}.D + A.B.C.D.E + A.B.\overline{C}.\overline{D}.\overline{E} + A.B.\overline{C}.D.E$ *can be simplified to* $A.B$.

Solution

$$\begin{aligned} &A.B.C.D + A.B.\overline{C}.\overline{D} + A.B.C.\overline{D} + A.B.\overline{C}.D + A.B.C.D.E + A.B.\overline{C}.\overline{D}.\overline{E} + A.B.\overline{C}.D.E \\ &\quad = A.B.C.D + A.B.\overline{C}.\overline{D} + A.B.C.\overline{D} + A.B.\overline{C}.D \\ &\quad = A.B.(C.D + \overline{C}.\overline{D} + C.\overline{D} + \overline{C}.D) = A.B \end{aligned}$$

- $A.B.C.D$ appears in $A.B.C.D.E$, $A.B.\overline{C}.\overline{D}$ appears in $A.B.\overline{C}.\overline{D}.\overline{E}$ and $A.B.\overline{C}.D$ appears in $A.B.\overline{C}.D.E$.
- As a result, all three five-variable terms are redundant.
- Also, variables C and D appear in all possible combinations and are therefore redundant.

6.3.13 Theorem 13 (DeMorgan's Theorem)

$$\text{(a) } [\overline{X_1 + X_2 + X_3 + \ldots + X_n}] = \overline{X_1}.\overline{X_2}.\overline{X_3}.\ldots.\overline{X_n} \tag{6.22}$$

$$\text{(b) } [\overline{X_1.X_2.X_3.\ldots.X_n}] = [\overline{X_1} + \overline{X_2} + \overline{X_3} + \ldots + \overline{X_n}] \tag{6.23}$$

According to the first theorem the complement of a sum equals the product of complements, while according to the second theorem the complement of a product equals the sum of complements. Figures 6.3(a) and (b) show logic diagram representations of De Morgan's theorems. While the first theorem can be interpreted to say that a multi-input NOR gate can be implemented as a multi-input bubbled AND gate, the second theorem, which is the dual of the first, can be interpreted to say that a multi-input NAND gate can be implemented as a multi-input bubbled OR gate.

DeMorgan's theorem can be proved as follows. Let us assume that all variables are in a logic '0' state. In that case

$$\text{LHS} = [\overline{X_1 + X_2 + X_3 + \cdots + X_n}] = [\overline{0 + 0 + 0 + \cdots + 0}] = \overline{0} = 1$$

$$\text{RHS} = \overline{X_1}.\overline{X_2}.\overline{X_3}.\ldots.\overline{X_n} = \overline{0}.\overline{0}.\overline{0}.\ldots.\overline{0} = 1.1.1.\ldots.1 = 1$$

Therefore, LHS = RHS.

Now, let us assume that any one of the n variables, say X_1, is in a logic HIGH state:

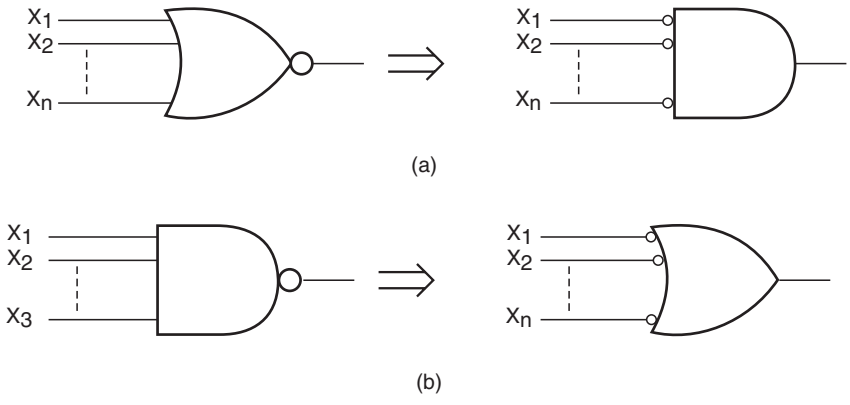

Figure 6.3 DeMorgan's theorem.

$$\text{LHS} = [\overline{X_1 + X_2 + X_3 + \cdots X_n}] = [\overline{1+0+0+\cdots+0}] = \overline{1} = 0$$

$$\text{RHS} = \overline{X_1}.\overline{X_2}.\overline{X_3}.\ldots.\overline{X_n} = \overline{1}.\overline{0}.\overline{0}.\ldots.\overline{0} = 0.1.1.\ldots.1 = 0$$

Therefore, again LHS = RHS.

The same holds good when more than one or all variables are in the logic '1' state. Therefore, theorem 13(a) stands proved. Since theorem 13(b) is the dual of theorem 13(a), the same also stands proved. Theorem 13(b), though, can be proved on similar lines.

6.3.14 Theorem 14 (Transposition Theorem)

$$\text{(a) } X.Y + \overline{X}.Z = (X+Z).(\overline{X}+Y)$$

and

$$\text{(b) } (X+Y).(\overline{X}+Z) = X.Z + \overline{X}.Y \tag{6.24}$$

This theorem can be applied to any sum-of-products or product-of-sums expression having two terms, provided that a given variable in one term has its complement in the other. Table 6.4 gives the proof of theorem 14(a) using the method of perfect induction. Theorem 14(b) is the dual of theorem 14(a) and hence stands proved.

As an example,

$$\overline{A}.B + A.\overline{B} = (A+B).(\overline{A}+\overline{B}) \quad \text{and} \quad A.B + \overline{A}.\overline{B} = (A+\overline{B}).(\overline{A}+B)$$

Incidentally, the first expression is the representation of a two-input EX-OR gate, while the second expression gives two forms of representation of a two-input EX-NOR gate.

Table 6.4 Proof of theorem 13(a).

X	Y	Z	XY	$\overline{X}Z$	X+Z	$\overline{X}+Y$	$XY+\overline{X}Z$	(X+Z)($\overline{X}+Y$)
0	0	0	0	0	0	1	0	0
0	0	1	0	1	1	1	1	1
0	1	0	0	0	0	1	0	0
0	1	1	0	1	1	1	1	1
1	0	0	0	0	1	0	0	0
1	0	1	0	0	1	0	0	0
1	1	0	1	0	1	1	1	1
1	1	1	1	0	1	1	1	1

6.3.15 Theorem 15

$$\text{(a) } X.f(X, \overline{X}, Y, Z, \dots) = X.f(1, 0, Y, Z, \dots) \tag{6.25}$$

$$\text{(b) } X + f(X, \overline{X}, Y, Z, \dots) = X + f(0, 1, Y, Z, \dots) \tag{6.26}$$

According to theorem 15(a), if a variable X is multiplied by an expression containing X and $\overline{X}$ in addition to other variables, then all Xs and $\overline{X}$s can be replaced with 1s and 0s respectively. This would be valid as $X.X = X$ and $X.1 = X$. Also, $X.\overline{X} = 0$ and $X.0 = 0$. According to theorem 15(b), if a variable X is added to an expression containing terms having X and $\overline{X}$ in addition to other variables, then all Xs can be replaced with 0s and all $\overline{X}$s can be replaced with ls. This is again permissible as $X + X$ as well as $X + 0$ equals X. Also, $X + \overline{X}$ and $\overline{X} + 1$ both equal 1.

This pair of theorems is very useful in eliminating redundancy in a given expression. An important corollary of this pair of theorems is that, if the multiplying variable is $\overline{X}$ in theorem 15(a), then all Xs will be replaced by 0s and all $\overline{X}$s will be replaced by ls. Similarly, if the variable being added in theorem 15(b) is $\overline{X}$, then Xs and $\overline{X}$s in the expression are replaced by 1s and 0s respectively. In that case the two theorems can be written as follows:

$$\text{(a) } \overline{X}.f(X, \overline{X}, Y, Z, \dots) = \overline{X}.f(0, 1, Y, Z, \dots) \tag{6.27}$$

$$\text{(b) } \overline{X} + f(X, \overline{X}, Y, Z, \dots) = \overline{X} + f(1, 0, Y, Z, \dots) \tag{6.28}$$

The theorems are further illustrated with the help of the following examples:

1. $A.[\overline{A}.B + A.\overline{C} + (\overline{A} + D).(A + \overline{E})] = A.[0.B + 1.\overline{C} + (0 + D).(1 + \overline{E})] = A.(\overline{C} + D)$.
2. $\overline{A} + [\overline{A}.B + A.\overline{C} + (\overline{A} + B).(A + \overline{E})] = \overline{A} + [0.B + 1.\overline{C} + (0 + B).(1 + \overline{E})] = \overline{A} + \overline{C} + B$.

6.3.16 Theorem 16

$$\text{(a) } f(X, \overline{X}, Y, \dots, Z) = X.f(1, 0, Y, \dots, Z) + \overline{X}.f(0, 1, Y, \dots, Z) \tag{6.29}$$

$$\text{(b) } f(X, \overline{X}, Y, \dots, Z) = [X + f(0, 1, Y, \dots, Z)][\overline{X} + f(1, 0, Y, \dots, Z)] \tag{6.30}$$

The proof of theorem 16(a) is straightforward and is given as follows:

$$f(X, \overline{X}, Y, \ldots, Z) = X.f(X, \overline{X}, Y, \ldots, Z) + \overline{X}.f(X, \overline{X}, Y, \ldots, Z)$$
$$= X.f(1, 0, Y, \ldots, Z) + \overline{X}.f(0, 1, Y, \ldots, Z)[\text{(Theorem 15(a)}]$$

Also

$$f(X, \overline{X}, Y, \ldots, Z) = [X + f(X, \overline{X}, Y, \ldots, Z)][\overline{X} + .f(X, \overline{X}, Y, \ldots, Z)]$$
$$= [X + f(0, 1, Y, \ldots, Z)][\overline{X} + f(1, 0, Y, \ldots, Z)][\text{Theorem 15(b)}]$$

6.3.17 Theorem 17 (Involution Law)

$$\overline{\overline{X}} = X \tag{6.31}$$

Involution law says that the complement of the complement of an expression leaves the expression unchanged. Also, the dual of the dual of an expression is the original expression. This theorem forms the basis of finding the equivalent product-of-sums expression for a given sum-of-products expression, and vice versa.

Example 6.5

Prove the following:

1. $L.(M + \overline{N}) + \overline{L}.\overline{P}.Q = (L + \overline{P}.Q).(\overline{L} + M + \overline{N})$.
2. $[A.\overline{B} + \overline{C} + \overline{D}].[D + (E + \overline{F}).G] = D.(A.\overline{B} + \overline{C}) + \overline{D}.G.(E + \overline{F})$.

Solution

1. Let us assume that $L = X$, $(M + \overline{N}) = Y$ and $\overline{P}.Q = Z$.
   The LHS of the given Boolean equation then reduces to $X.Y + \overline{X}.Z$.
   Applying the transposition theorem,

$$X.Y + \overline{X}.Z = (X + Z).(\overline{X} + Y) = (L + \overline{P}.Q)(\overline{L} + M + \overline{N}) = \text{RHS}$$

2. Let us assume $\overline{D} = X$, $A.\overline{B} + \overline{C} = Y$ and $(E + \overline{F}).G = Z$.
   The LHS of given the Boolean equation then reduces to $(X + Y).(\overline{X} + Z)$.
   Applying the transposition theorem,

$$(X + Y).(\overline{X} + Z) = X.Z + \overline{X}.Y = \overline{D}.G.(E + \overline{F}) + D.(A.\overline{B} + \overline{C}) = \text{RHS}$$

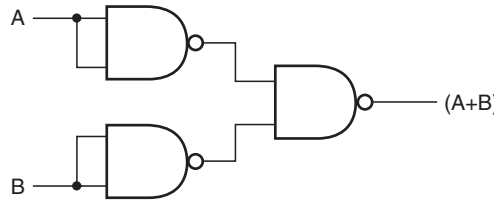

Figure 6.4 Example 6.6.

Example 6.6

Starting with the Boolean expression for a two-input OR gate, apply Boolean laws and theorems to modify it in such a way as to facilitate the implementation of a two-input OR gate by using two-input NAND gates only.

Solution

- A two-input OR gate is represented by the Boolean equation $Y = (A + B)$, where A and B are the input logic variables and Y is the output.
- Now, $(A + B) = (\overline{\overline{A} + \overline{B}})$ Involution law
  $= (\overline{\overline{A}.\overline{B}})$ DeMorgan's theorem
  $= [\overline{(\overline{A.A})}.\overline{(\overline{B.B})}]$ Idempotent law
- Figure 6.4 shows the NAND gate implementation of a two-input OR gate.

Example 6.7

Apply suitable Boolean laws and theorems to modify the expression for a two-input EX-OR gate in such a way as to implement a two-input EX-OR gate by using the minimum number of two-input NAND gates only.

Solution

- A two-input EX-OR gate is represented by the Boolean expression $Y = \overline{A}.B + A.\overline{B}$.
- Now, $\overline{A}.B + A.\overline{B} = \overline{\overline{\overline{A}.B} + \overline{A.\overline{B}}}$ Involution law
  $= \overline{\overline{\overline{A}.B}}.\overline{\overline{A.\overline{B}}}$ DeMorgan's law
  $= [\overline{B.(\overline{A} + \overline{B})}].[\overline{A.(\overline{A} + \overline{B})}]$
  $= (\overline{B.\overline{A.B}}).(\overline{A.\overline{A.B}})$ (6.32)
- Equation (6.32) is in a form that can be implemented with NAND gates only.
- Figure 6.5 shows the logic diagram.

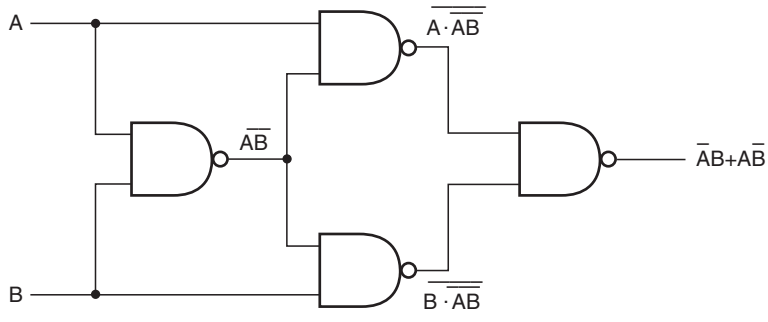

Figure 6.5 Example 6.7.

6.4 Simplification Techniques

In this section, we will discuss techniques other than the application of laws and theorems of Boolean algebra discussed in the preceding paragraphs of this chapter for simplifying or more precisely minimizing a given complex Boolean expression. The primary objective of all simplification procedures is to obtain an expression that has the minimum number of terms. Obtaining an expression with the minimum number of literals is usually the secondary objective. If there is more than one possible solution with the same number of terms, the one having the minimum number of literals is the choice. The techniques to be discussed include:

(a) the Quine–McCluskey tabular method;
(b) the Karnaugh map method.

Before we move on to discuss these techniques in detail, it would be relevant briefly to describe sum-of-products and product-of-sums Boolean expressions. The given Boolean expression will be in either of the two forms, and the objective will be to find a minimized expression in the same or the other form.

6.4.1 Sum-of-Products Boolean Expressions

A sum-of-products expression contains the sum of different terms, with each term being either a single literal or a product of more than one literal. It can be obtained from the truth table directly by considering those input combinations that produce a logic '1' at the output. Each such input combination produces a term. Different terms are given by the product of the corresponding literals. The sum of all terms gives the expression. For example, the truth table in Table 6.5 can be represented by the Boolean expression

$$Y = \overline{A}\,.\overline{B}.\overline{C} + \overline{A}.B.C + A.B.\overline{C} + A.\overline{B}.C \tag{6.33}$$

Considering the first term, the output is '1' when $A = 0$, $B = 0$ and $C = 0$. This is possible only when $\overline{A}$, $\overline{B}$ and $\overline{C}$ are ANDed. Also, for the second term, the output is '1' only when B, C and $\overline{A}$ are ANDed. Other terms can be explained similarly. A sum-of-products expression is also known as a *minterm expression*.

Table 6.5 truth table of boolean expression of equation 6.33.

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

6.4.2 Product-of-Sums Expressions

A product-of-sums expression contains the product of different terms, with each term being either a single literal or a sum of more than one literal. It can be obtained from the truth table by considering those input combinations that produce a logic '0' at the output. Each such input combination gives a term, and the product of all such terms gives the expression. Different terms are obtained by taking the sum of the corresponding literals. Here, '0' and '1' respectively mean the uncomplemented and complemented variables, unlike sum-of-products expressions where '0' and '1' respectively mean complemented and uncomplemented variables.

To illustrate this further, consider once again the truth table in Table 6.5. Since each term in the case of the product-of-sums expression is going to be the sum of literals, this implies that it is going to be implemented using an OR operation. Now, an OR gate produces a logic '0' only when all its inputs are in the logic '0' state, which means that the first term corresponding to the second row of the truth table will be $A+B+\overline{C}$. The product-of-sums Boolean expression for this truth table is given by $(A+B+\overline{C}).(A+\overline{B}+C).(\overline{A}+B+C).(\overline{A}+\overline{B}+\overline{C})$.

Transforming the given product-of-sums expression into an equivalent sum-of-products expression is a straightforward process. Multiplying out the given expression and carrying out the obvious simplification provides the equivalent sum-of-products expression:

$$\begin{aligned}&(A+B+\overline{C}).(A+\overline{B}+C).(\overline{A}+B+C).(\overline{A}+\overline{B}+\overline{C})\\&=(A.A+A.\overline{B}+A.C+B.A+B.\overline{B}+B.C+\overline{C}.A+\overline{C}.\overline{B}+\overline{C}.C).(\overline{A}.\overline{A}+\overline{A}.\overline{B}+\overline{A}.\overline{C}+B.\overline{A}+B.\overline{B}\\&\quad+B.\overline{C}+C.\overline{A}+C.\overline{B}+C.\overline{C}\\&=(A+B.C+\overline{B}.\overline{C}).(\overline{A}+B.\overline{C}+C.\overline{B})=A.B.\overline{C}+A.\overline{B}.C+\overline{A}.B.C+\overline{A}.\overline{B}.\overline{C}\end{aligned}$$

A given sum-of-products expression can be transformed into an equivalent product-of-sums expression by (a) taking the dual of the given expression, (b) multiplying out different terms to get the sum-of-products form, (c) removing redundancy and (d) taking a dual to get the equivalent product-of-sums expression. As an illustration, let us find the equivalent product-of-sums expression of the sum-of-products expression

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

The dual of the given expression $=(A+B).(\overline{A}+\overline{B})$:

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

The dual of $(A.\overline{B}+\overline{A}.B)=(A+\overline{B}).(\overline{A}+B)$. Therefore

$$A.B+\overline{A}.\overline{B}=(A+\overline{B}).(\overline{A}+B)$$

6.4.3 Expanded Forms of Boolean Expressions

Expanded sum-of-products and product-of-sums forms of Boolean expressions are useful not only in analysing these expressions but also in the application of minimization techniques such as the Quine–McCluskey tabular method and the Karnaugh mapping method for simplifying given Boolean expressions. The expanded form, sum-of-products or product-of-sums, is obtained by including all possible combinations of missing variables.

As an illustration, consider the following sum-of-products expression:

$$A.\overline{B}+B.\overline{C}+A.B.\overline{C}+\overline{A}.C$$

It is a three-variable expression. Expanded versions of different minterms can be written as follows:

- $A.\overline{B}=A.\overline{B}.(C+\overline{C})=A.\overline{B}.C+A.\overline{B}.\overline{C}$.
- $B.\overline{C}=B.\overline{C}.(A+\overline{A})=B.\overline{C}.A+B.\overline{C}.\overline{A}$.
- $A.B.\overline{C}$ is a complete term and has no missing variable.
- $\overline{A}.C=\overline{A}.C.(B+\overline{B})=\overline{A}.C.B+\overline{A}.C.\overline{B}$.

The expanded sum-of-products expression is therefore given by

$$\begin{aligned}A.\overline{B}.C+A.\overline{B}.\overline{C}+A.B.\overline{C}+\overline{A}.B.\overline{C}+A.B.\overline{C}+\overline{A}.B.C+\overline{A}.\overline{B}.C=A.\overline{B}.C+A.\overline{B}.\overline{C}\\+A.B.\overline{C}+\overline{A}.B.\overline{C}+\overline{A}.B.C+\overline{A}.\overline{B}.C\end{aligned}$$

As another illustration, consider the product-of-sums expression

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

It is four-variable expression with A, B, C and D being the four variables. $\overline{A}+B$ in this case expands to $(\overline{A}+B+C+D).(\overline{A}+B+C+\overline{D}).(\overline{A}+B+\overline{C}+D).(\overline{A}+B+\overline{C}+\overline{D})$.

The expanded product-of-sums expression is therefore given by

$$\begin{aligned}&(\overline{A}+B+C+D).(\overline{A}+B+C+\overline{D}).(\overline{A}+B+\overline{C}+D).(\overline{A}+B+\overline{C}+\overline{D}).(\overline{A}+B+\overline{C}+\overline{D})\\&=(\overline{A}+B+C+D).(\overline{A}+B+C+\overline{D}).(\overline{A}+B+\overline{C}+D).(\overline{A}+B+\overline{C}+\overline{D})\end{aligned}$$

6.4.4 Canonical Form of Boolean Expressions

An expanded form of Boolean expression, where each term contains all Boolean variables in their true or complemented form, is also known as the *canonical form* of the expression.

As an illustration, $f(A.B,C)=\overline{A}.\overline{B}.\overline{C}+\overline{A}.\overline{B}.C+A.B.C$ is a Boolean function of three variables expressed in canonical form. This function after simplification reduces to $\overline{A}.\overline{B}+A.B.C$ and loses its canonical form.

6.4.5 Σ and Π Nomenclature

Σ and Π notations are respectively used to represent sum-of-products and product-of-sums Boolean expressions. We will illustrate these notations with the help of examples. Let us consider the following Boolean function:

$$f(A, B, C, D) = A.\overline{B}.\overline{C} + A.B.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.\overline{B}.\overline{C}.D$$

We will represent this function using Σ notation. The first step is to write the expanded sum-of-products given by

$$\begin{aligned} f(A, B, C, D) &= A.\overline{B}.\overline{C}.(D + \overline{D}) + A.B.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.\overline{B}.\overline{C}.D \\ &= A.\overline{B}.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D} + A.B.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.\overline{B}.\overline{C}.D \end{aligned}$$

Different terms are then arranged in ascending order of the binary numbers represented by various terms, with true variables representing a '1' and a complemented variable representing a '0'. The expression becomes

$$f(A, B, C, D) = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.B.C.D$$

The different terms represent 0001, 0101, 1000, 1001 and 1111. The decimal equivalent of these terms enclosed in the Σ then gives the Σ notation for the given Boolean function. That is, $f(A, B, C, D) = \sum 1, 5, 8, 9, 15$.

The complement of $f(A, B, C, D)$, that is, $f'(A, B, C, D)$, can be directly determined from Σ notation by including the left-out entries from the list of all possible numbers for a four-variable function. That is,

$$f'(A, B, C, D) = \sum 0, 2, 3, 4, 6, 7, 10, 11, 12, 13, 14$$

Let us now take the case of a product-of-sums Boolean function and its representation in Π nomenclature. Let us consider the Boolean function

$$f(A, B, C, D) = (B + \overline{C} + \overline{D}).(\overline{A} + \overline{B} + C + D).(A + \overline{B} + \overline{C} + \overline{D})$$

The expanded product-of-sums form is given by

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

The binary numbers represented by the different sum terms are 0011, 1011, 1100 and 0111 (true and complemented variables here represent 0 and 1 respectively). When arranged in ascending order, these numbers are 0011, 0111, 1011 and 1100. Therefore,

$$f(A, B, C, D) = \prod 3, 7, 11, 12 \quad \text{and} \quad f'(A, B, C, D) = \prod 0, 1, 2, 4, 5, 6, 8, 9, 10, 13, 14, 15$$

An interesting corollary of what we have discussed above is that, if a given Boolean function $f(A,B,C)$ is given by $f(A, B, C) = \sum 0, 1, 4, 7$, then

$$f(A, B, C) = \prod 2, 3, 5, 6 \quad \text{and} \quad f'(A, B, C) = \sum 2, 3, 5, 6 = \prod 0, 1, 4, 7$$

Optional combinations can also be incorporated into Σ and Π nomenclature using suitable identifiers; ϕ or d are used as identifiers. For example, if $f(A, B, C) = \overline{A}.\overline{B}.\overline{C} + A.\overline{B}.\overline{C} + A.\overline{B}.C$ and $\overline{A}.B.C$, $A.B.C$ are optional combinations, then

$$f(A, B, C) = \sum 0, 4, 5 + \sum_{\phi} 3, 7 = \sum 0, 4, 5 + \sum_{d} 3, 7$$

$$f(A, B, C) = \prod 1, 2, 6 + \prod_{\phi} 3, 7 = \prod 1, 2, 6 + \prod_{d} 3, 7$$

Example 6.8

For a Boolean function $f(A, B) = \sum 0, 2$, *prove that* $f(A, B) = \prod 1, 3$ *and* $f'(A, B) = \sum 1, 3 = \prod 0, 2$.

Solution

- $f(A, B) = \sum 0, 2 = \overline{A}.\overline{B} + A.\overline{B} = \overline{B}.(A + \overline{A}) = \overline{B}$.
- Now, $\prod 1, 3 = (A + \overline{B}).(\overline{A} + \overline{B}) = A.\overline{A} + A.\overline{B} + \overline{B}.\overline{A} + \overline{B}.\overline{B} = A.\overline{B} + \overline{A}.\overline{B} + \overline{B} = \overline{B}$.
- Now, $\sum 1, 3 = \overline{A}.B + A.B = B.(\overline{A} + A) = B$.
  and $\prod 0, 2 = (A + B).(\overline{A} + B) = A.\overline{A} + A.B + B.\overline{A} + B.B = A.B + \overline{A}.B + B = B$.
- Therefore, $\sum 1, 3 = \prod 0, 2$.
- Also, $f(A, B) = \overline{B}$.
- Therefore, $f'(A, B) = B$ or $f'(A, B) = \sum 1, 3 = \prod 0, 2$.

6.5 Quine–McCluskey Tabular Method

The Quine–McCluskey tabular method of simplification is based on the complementation theorem, which says that

$$X.Y + X.\overline{Y} = X \tag{6.34}$$

where X represents either a variable or a term or an expression and Y is a variable. This theorem implies that, if a Boolean expression contains two terms that differ only in one variable, then they can be combined together and replaced with a term that is smaller by one literal. The same procedure is applied for the other pairs of terms wherever such a reduction is possible. All these terms reduced by one literal are further examined to see if they can be reduced further. The process continues until the terms become irreducible. The irreducible terms are called *prime implicants*. An optimum set of prime implicants that can account for all the original terms then constitutes the minimized expression. The technique can be applied equally well for minimizing sum-of-products and product-of-sums expressions and is particularly useful for Boolean functions having more than six variables as it can be mechanized and run on a computer. On the other hand, the Karnaugh mapping method, to be discussed later, is a graphical method and becomes very cumbersome when the number of variables exceeds six.

The step-by-step procedure for application of the tabular method for minimizing Boolean expressions, both sum-of-products and product-of-sums, is outlined as follows:

1. The Boolean expression to be simplified is expanded if it is not in expanded form.
2. Different terms in the expression are divided into groups depending upon the number of 1s they have. True and complemented variables in a sum-of-products expression mean '1' and '0' respectively.

The reverse is true in the case of a product-of-sums expression. The groups are then arranged, beginning with the group having the least number of 1s in its included terms. Terms within the same group are arranged in ascending order of the decimal numbers represented by these terms.

As an illustration, consider the expression

$$A.B.C + \overline{A}.B.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + \overline{A}.\overline{B}.\overline{C}$$

The grouping of different terms and the arrangement of different terms within the group are shown below:

$\overline{A}.\overline{B}.\overline{C}$		000	First group
$A.\overline{B}.\overline{C}$		100	Second group
$\overline{A}.B.C$	$\longrightarrow$	011	Third group
$A.\overline{B}.C$		101	
ABC		111	Fourth group

As another illustration, consider a product-of-sums expression given by

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

The formation of groups and the arrangement of terms within different groups for the product-of-sums expression are as follows:

$A.B.C.D$		0000
$A.B.\overline{C}.\overline{D}$		0011
$A.\overline{B}.C.\overline{D}$		0101
$\overline{A}.B.\overline{C}.D$	$\longrightarrow$	1010
$A.\overline{B}.\overline{C}.\overline{D}$		0111
$\overline{A}.\overline{B}.\overline{C}.D$		1110
$\overline{A}.\overline{B}.\overline{C}.\overline{D}$		1111

It may be mentioned here that the Boolean expressions that we have considered above did not contain any optional terms. If there are any, they are also considered while forming groups. This completes the first table.

3. The terms of the first group are successively matched with those in the next adjacent higher-order group to look for any possible matching and consequent reduction. The terms are considered matched when all literals except for one match. The pairs of matched terms are replaced with a

single term where the position of the unmatched literals is replaced with a dash (—). These new terms formed as a result of the matching process find a place in the second table. The terms in the first table that do not find a match are called the prime implicants and are marked with an asterisk (∗). The matched terms are ticked (✓).

4. Terms in the second group are compared with those in the third group to look for a possible match. Again, terms in the second group that do not find a match become the prime implicants.
5. The process continues until we reach the last group. This completes the first round of matching. The terms resulting from the matching in the first round are recorded in the second table.
6. The next step is to perform matching operations in the second table. While comparing the terms for a match, it is important that a dash (—) is also treated like any other literal, that is, the dash signs also need to match. The process continues on to the third table, the fourth table and so on until the terms become irreducible any further.
7. An optimum selection of prime implicants to account for all the original terms constitutes the terms for the minimized expression. Although optional (also called 'don't care') terms are considered for matching, they do not have to be accounted for once prime implicants have been identified.

Let us consider an example. Consider the following sum-of-products expression:

$$\overline{A}.B.C + \overline{A}.\overline{B}.D + A.\overline{C}.D + B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D \tag{6.35}$$

In the first step, we write the expanded version of the given expression. It can be written as follows:

$$\overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + \overline{A}.\overline{B}.C.D + \overline{A}.\overline{B}.\overline{C}.D + A.B.\overline{C}.D + A.\overline{B}.\overline{C}.D + A.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D$$

The formation of groups, the placement of terms in different groups and the first-round matching are shown as follows:

A	B	C	D
0	0	0	1
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	1
1	1	0	0
1	1	0	1

A	B	C	D	
0	0	0	1	✓
0	1	0	0	✓
0	0	1	1	✓
0	1	0	1	✓
0	1	1	0	✓
1	0	0	1	✓
1	1	0	0	✓
0	1	1	1	✓
1	1	0	1	✓

A	B	C	D	
0	0	–	1	✓
0	–	0	1	✓
–	0	0	1	✓
0	1	0	–	✓
0	1	–	0	✓
–	1	0	0	✓
0	–	1	1	✓
0	1	–	1	✓
–	1	0	1	✓
0	1	1	–	✓
1	–	0	1	✓
1	1	0	–	✓

The second round of matching begins with the table shown on the previous page. Each term in the first group is compared with every term in the second group. For instance, the first term in the first group $00-1$ matches with the second term in the second group $01-1$ to yield $0--1$, which is recorded in the table shown below. The process continues until all terms have been compared for a possible match. Since this new table has only one group, the terms contained therein are all prime implicants. In the present example, the terms in the first and second tables have all found a match. But that is not always the case.

A	B	C	D	
0	–	–	1	*
–	–	0	1	*
0	1	–	–	*
–	1	0	–	*

The next table is what is known as the prime implicant table. The prime implicant table contains all the original terms in different columns and all the prime implicants recorded in different rows as shown below:

0001	0011	0100	0101	0110	0111	1001	1100	1101		
✓	✓		✓		✓				0– –1	$P \rightarrow \overline{A}.D$
✓			✓			✓		✓	– –01	$Q \rightarrow \overline{C}.D$
		✓	✓	✓	✓				01– –	$R \rightarrow \overline{A}.B$
		✓	✓				✓	✓	–10–	$S \rightarrow B.\overline{C}$

Each prime implicant is identified by a letter. Each prime implicant is then examined one by one and the terms it can account for are ticked as shown. The next step is to write a product-of-sums expression using the prime implicants to account for all the terms. In the present illustration, it is given as follows.

$$(P+Q).(P).(R+S).(P+Q+R+S).(R).(P+R).(Q).(S).(Q+S)$$

Obvious simplification reduces this expression to $PQRS$ which can be interpreted to mean that all prime implicants, that is, P, Q, R and S, are needed to account for all the original terms.

Therefore, the minimized expression $= \overline{A}.D+\overline{C}.D+\overline{A}.B+B.\overline{C}$.

What has been described above is the formal method of determining the optimum set of prime implicants. In most of the cases where the prime implicant table is not too complex, the exercise can be done even intuitively. The exercise begins with identification of those terms that can be accounted for by only a single prime implicant. In the present example, 0011, 0110, 1001 and 1100 are such terms. As a result, P, Q, R and S become the essential prime implicants. The next step is to find out if any terms have not been covered by the essential prime implicants. In the present case, all terms have been covered by essential prime implicants. In fact, all prime implicants are essential prime implicants in the present example.

As another illustration, let us consider a product-of-sums expression given by

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

The procedure is similar to that described for the case of simplification of sum-of-products expressions. The resulting tables leading to identification of prime implicants are as follows:

A	B	C	D
0	1	0	1
0	1	1	1
1	1	0	1
1	1	1	0
1	1	1	1

A	B	C	D	
0	1	0	1	✓
0	1	1	1	✓
1	1	0	1	✓
1	1	1	0	✓
1	1	1	1	✓

A	B	C	D	
0	1	–	1	✓
–	1	0	1	✓
–	1	1	1	✓
1	1	–	1	✓
1	1	1	–	*

A	B	C	D	
–	1	–	1	*

The prime implicant table is constructed after all prime implicants have been identified to look for the optimum set of prime implicants needed to account for all the original terms. The prime implicant table shows that both the prime implicants are the essential ones:

0101	0111	1101	1110	1111	Prime implicants
			✓	✓	111–
✓	✓	✓		✓	–1–1

The minimized expression = $(\overline{A}+\overline{B}+\overline{C}).(\overline{B}+\overline{D})$.

6.5.1 Tabular Method for Multi-Output Functions

When it comes to a multi-output logic network, a network that has more than one output, sharing of some logic blocks between different functions is highly probable. For an optimum logic implementation of the multi-output function, different functions cannot be and should not be minimized in isolation because a possible common term that could have been shared may not turn out to be a prime implicant if the functions are worked out individually. The method of applying the tabular approach to multi-output functions is to get a minimized set of expressions that would lead to an optimum overall system. The method is illustrated by the following example.

Consider a logic system with two outputs that is described by the following Boolean expressions:

$$Y_1 = \overline{A}.B.D + \overline{A}.C.D + \overline{A}.\overline{C}.\overline{D} \tag{6.36}$$

$$Y_2 = \overline{A}.B.C + A.C.D + A.\overline{B}.C.\overline{D} + \overline{A}.\overline{B}.C.\overline{D} \tag{6.37}$$

The expanded forms of the two functions are as follows:

$$Y_1 = \overline{A}.B.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.\overline{D}$$

$$Y_1 = \overline{A}.B.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.\overline{D}$$

$$Y_2 = \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + A.B.C.D + A.\overline{B}.C.D + A.\overline{B}.C.\overline{D} + \overline{A}.\overline{B}.C.\overline{D}$$

The rows representing different terms are arranged in the usual manner, with all the terms contained in the two functions finding a place without repetition, as shown in the table below:

ABCD	1	2	
0000	✓		✓
0010		✓	✓
0100	✓		✓
0011	✓		✓
0101	✓		✓
0110		✓	✓
1010		✓	✓
0111	✓	✓	✓
1011		✓	✓
1111		✓	✓

Each term is checked under the column or columns depending upon the functions in which it is contained. For instance, if a certain term is contained in the logic expressions for both output 1 and output 2, it will be checked in both output columns. The matching process begins in the same way as described earlier for the case of single-output functions, with some modifications outlined as follows:

1. Only those terms can be combined that have at least one check mark in the output column in common. For instance, 0000 cannot combine with 0010 but can combine with 0100.
2. In the resulting row, only the common outputs are checked. For instance, when 0101 is matched with 0111, then, in the resulting term 01–1, only output 1 will be checked.
3. A combining term can be checked off only if the resulting term accounts for all the outputs in which the term is contained.

The table below shows the results of the first round of matching:

ABCD	1	2	
0−00	✓		*
0−10		✓	*
−010		✓	*
010−	✓		*
0−11	✓		*
01−1	✓		*
011−		✓	*
101−		✓	*
−111		✓	*
1−11		✓	*

No further matching is possible. The prime implicant table is shown below:

Output 1					Output 2						
0000	0011	0100	0101	0111	0010	0110	0111	1010	1011	1111	ABCD
✓		✓									0−00
					✓	✓					0−10
					✓			✓			−010
		✓	✓								010−
	✓			✓							0−11
			✓	✓							01−1
						✓	✓				011−
								✓	✓		101−
							✓			✓	−111
									✓	✓	1−11

For each prime implicant, check marks are placed only in columns that pertain to the outputs checked off for this prime implicant. For instance, 0-00 has only output 1 checked off. Therefore, the relevant terms under output 1 will be checked off. The completed table is treated as a whole while marking the required prime implicants to be considered for writing the minimized expressions. The minimized expressions are as follows:

$$Y_1 = \overline{A}.\overline{C}.\overline{D} + \overline{A}.C.D + \overline{A}.B.\overline{C} \quad \text{and} \quad Y_2 = B.C.D + A.\overline{B}.C + \overline{A}.C.\overline{D}$$

Example 6.9

Using the Quine–McCluskey tabular method, find the minimum sum of products for $f(A, B, C, D) = \sum(1, 2, 3, 9, 12, 13, 14) + \sum_{\phi}(0, 7, 10, 15)$.

Solution

The different steps to finding the solution to the given problem are tabulated below. As we can see, eight prime implicants have been identified. These prime implicants along with the inputs constitute the prime implicant table. Remember that optional inputs are not considered while constructing the prime implicant table:

A	*B*	*C*	*D*	
0	0	0	0	✓
0	0	0	1	✓
0	0	1	0	✓
0	0	1	1	✓
1	0	0	1	✓
1	0	1	0	✓
1	1	0	0	✓
0	1	1	1	✓
1	1	0	1	✓
1	1	1	0	✓
1	1	1	1	✓

A	*B*	*C*	*D*	
0	0	0	–	✓
0	0	–	0	✓
0	0	–	1	✓
–	0	0	1	*
0	0	1	–	✓
–	0	1	0	*
0	–	1	1	*
1	–	0	1	*
1	–	1	0	*
1	1	0	–	✓
1	1	–	0	✓
–	1	1	1	*
1	1	–	1	✓
1	1	1	–	✓

A	*B*	*C*	*D*	
0	0	–	–	*
1	1	–	–	*

The product-of-sums expression that tells about the combination of prime implicants required to account for all the terms is given by the expression

$$(L+S).(M+S).(N+S).(L+P).(T).(P+T).(Q+T) \tag{6.38}$$

After obvious simplification, this reduces to the expression

$$\begin{aligned} &T.(L+S).(M+S).(N+S).(L+P) \\ &\quad = T.(LM+LS+MS+S).(LN+PN+LS+PS) \\ &\quad = T.(LM+S).(LN+PN+LS+PS) \\ &\quad = T.(LMN+LMPN+LMS+LMPS+LNS+PNS+LS+PS) \\ &\quad = T.(LMN+LMPN+LS+PS) \\ &\quad = TLMN+TLMPN+TLS+TPS \end{aligned} \tag{6.39}$$

0001	0010	0011	1001	1100	1101	1110	Prime implicants	
✓			✓				−001	L
	✓						−010	M
		✓					0−11	N
			✓		✓		1−01	P
						✓	1−10	Q
							−111	R
✓	✓	✓					00−−	S
				✓	✓	✓	11−−	T

The sum-of-products Boolean expression (6.39) states that all the input combinations can be accounted for by the prime implicants (T, L, M, N) or (T, L, M, P, N) or (T, L, S) or (T, P, S). The most optimum expression would result from either TLS or TPS. Therefore, the minimized Boolean function is given by

$$f(A, B, C, D) = A.B + \overline{B}.\overline{C}.D + \overline{A}.\overline{B} \tag{6.40}$$

or by

$$f(A, B, C, D) = A.B + \overline{A}.\overline{B} + A.\overline{C}.D \tag{6.41}$$

Example 6.10

A logic system has three inputs A, B and C and two outputs Y_1 and Y_2. The output functions Y_1 and Y_2 are expressed by $Y_1 = \overline{A}.B.C + B.\overline{C} + \overline{A}.\overline{C} + A.\overline{B}.C + A.B.C$ and $Y_2 = \overline{A}.B + A.\overline{C} + A.B.C$. Determine the minimized output logic functions using the Quine–McCluskey tabular method.

Solution

The expanded forms of Y_1 and Y_2 are written as follows:

$$\begin{aligned} Y_1 &= \overline{A}.B.C + A.B.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.C \\ &= \overline{A}.B.C + A.B.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.C \\ Y_2 &= \overline{A}.B.C + \overline{A}.B.\overline{C} + A.B.C + A.B.\overline{C} + A.\overline{B}.\overline{C} \end{aligned}$$

The different steps leading to construction of the prime implicant table are given in tabular form below:

A	*B*	*C*	1	2	
0	0	0	✓		✓
0	1	0	✓	✓	✓
1	0	0		✓	✓
0	1	1	✓	✓	✓
1	0	1	✓		✓
1	1	0	✓	✓	✓
1	1	1	✓	✓	✓

A	*B*	*C*	1	2	
0	–	0	✓		*
0	1	–	✓	✓	✓
–	1	0	✓	✓	✓
1	–	0		✓	*
–	1	1	✓	✓	✓
1	–	1	✓		*
1	1	–	✓	✓	✓

A	*B*	*C*	1	2	
–	1	–	✓	✓	*

Y_1						Y_2					**ABC**
000	**010**	**011**	**101**	**110**	**111**	**010**	**011**	**100**	**110**	**111**	
✓	✓										**0–0**
								✓	✓		**1–0**
			✓		✓						**1–1**
	✓	✓		✓	✓	✓	✓		✓	✓	**–1–**

From the prime implicant table, the minimized output Boolean functions can be written as follows:

$$Y_1 = B + \overline{A}.\overline{C} + A.C \tag{6.42}$$

$$Y_2 = B + A.\overline{C} \tag{6.43}$$

6.6 Karnaugh Map Method

A Karnaugh map is a graphical representation of the logic system. It can be drawn directly from either minterm (sum-of-products) or maxterm (product-of-sums) Boolean expressions. Drawing a Karnaugh map from the truth table involves an additional step of writing the minterm or maxterm expression depending upon whether it is desired to have a minimized sum-of-products or a minimized product-of-sums expression.

6.6.1 Construction of a Karnaugh Map

An *n*-variable Karnaugh map has 2^n squares, and each possible input is allotted a square. In the case of a minterm Karnaugh map, '1' is placed in all those squares for which the output is '1', and '0'

is placed in all those squares for which the output is '0'. 0s are omitted for simplicity. An 'X' is placed in squares corresponding to 'don't care' conditions. In the case of a maxterm Karnaugh map, a '1' is placed in all those squares for which the output is '0', and a '0' is placed for input entries corresponding to a '1' output. Again, 0s are omitted for simplicity, and an 'X' is placed in squares corresponding to 'don't care' conditions.

The choice of terms identifying different rows and columns of a Karnaugh map is not unique for a given number of variables. The only condition to be satisfied is that the designation of adjacent rows and adjacent columns should be the same except for one of the literals being complemented. Also, the extreme rows and extreme columns are considered adjacent. Some of the possible designation styles for two-, three- and four-variable minterm Karnaugh maps are given in Figs 6.6, 6.7 and 6.8 respectively.

The style of row identification need not be the same as that of column identification as long as it meets the basic requirement with respect to adjacent terms. It is, however, accepted practice to adopt a uniform style of row and column identification. Also, the style shown in Figs 6.6(a), 6.7(a) and 6.8(a) is more commonly used. Some more styles are shown in Fig. 6.9. A similar discussion applies for maxterm Karnaugh maps.

Having drawn the Karnaugh map, the next step is to form groups of 1s as per the following guidelines:

1. Each square containing a '1' must be considered at least once, although it can be considered as often as desired.
2. The objective should be to account for all the marked squares in the minimum number of groups.
3. The number of squares in a group must always be a power of 2, i.e. groups can have 1, 2, 4, 8, 16, . . . squares.
4. Each group should be as large as possible, which means that a square should not be accounted for by itself if it can be accounted for by a group of two squares; a group of two squares should not be made if the involved squares can be included in a group of four squares and so on.
5. 'Don't care' entries can be used in accounting for all of 1-squares to make optimum groups. They are marked 'X' in the corresponding squares. It is, however, not necessary to account for all 'don't care' entries. Only such entries that can be used to advantage should be used.

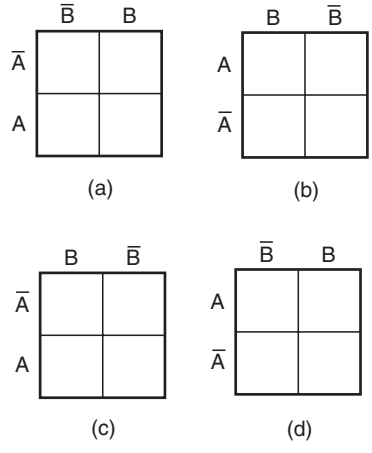

Figure 6.6 Two-variable Karnaugh map.

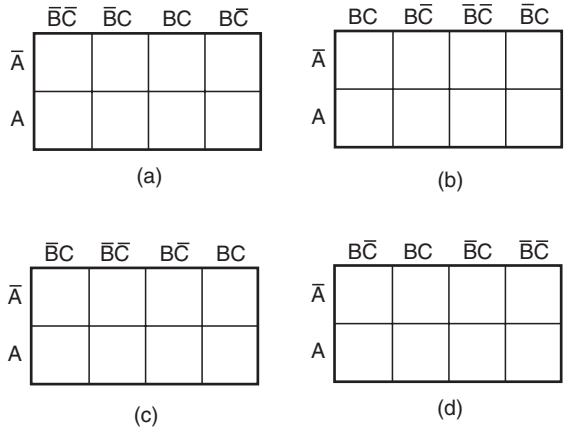

Figure 6.7 Three-variable Karnaugh map.

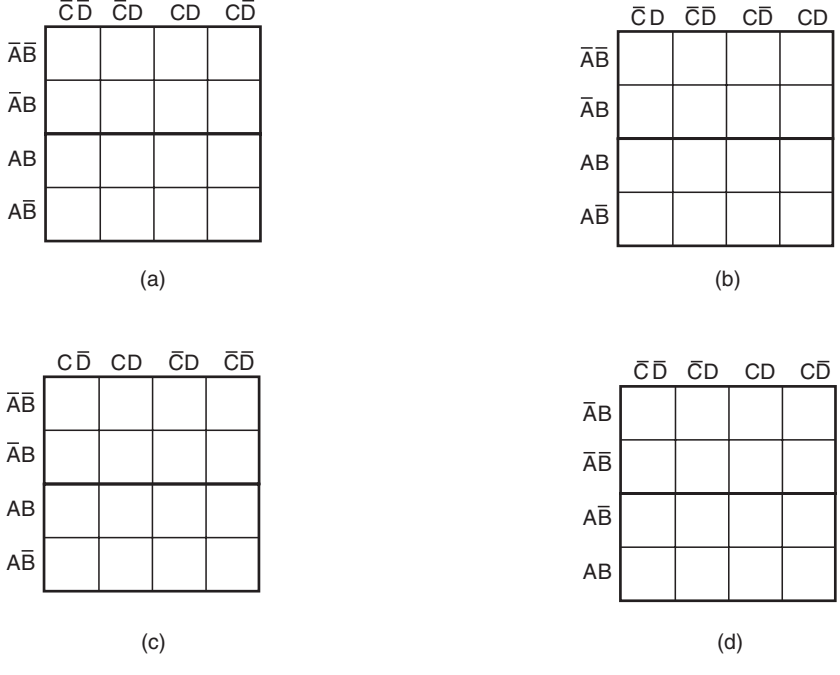

Figure 6.8 Four-variable Karnaugh map.

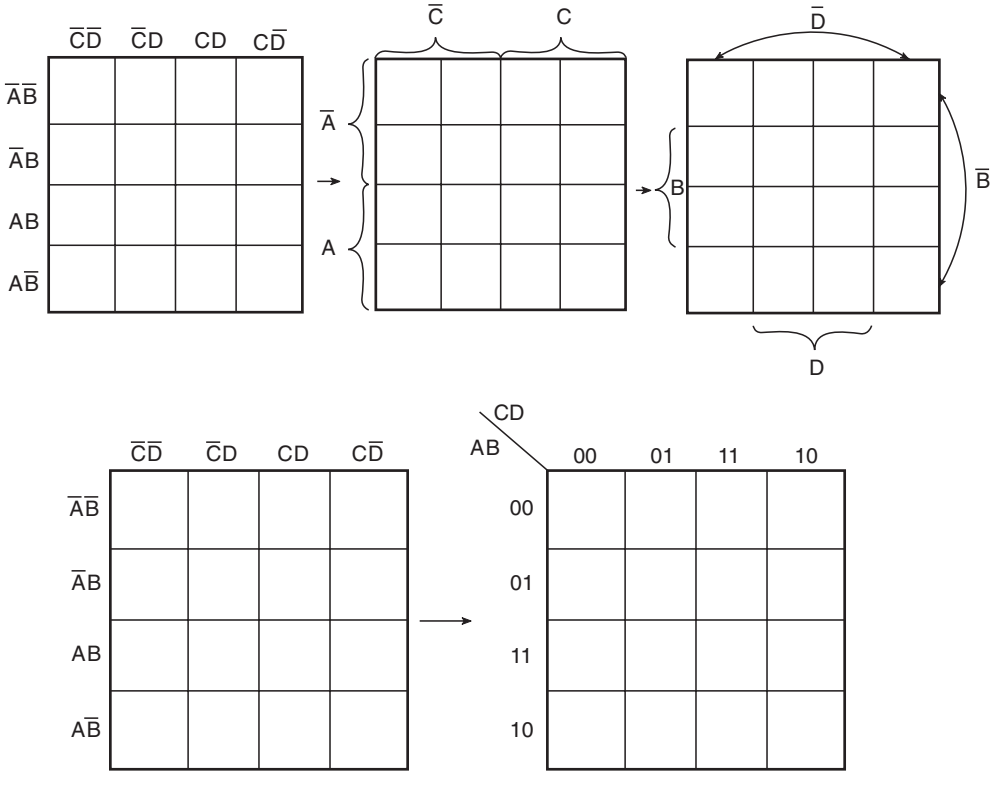

Figure 6.9 Different styles of row and column identification.

Having accounted for groups with all 1s, the minimum 'sum-of-products' or 'product-of-sums' expressions can be written directly from the Karnaugh map.

Figure 6.10 shows the truth table, minterm Karnaugh map and maxterm Karnaugh map of the Boolean function of a two-input OR gate. The minterm and maxterm Boolean expressions for the two-input OR gate are as follows:

$$Y = A + B \text{ (maxterm or product-of-sums)} \tag{6.44}$$

$$Y = \overline{A}.B + A.\overline{B} + A.B \text{ (minterm or sum-of-products)} \tag{6.45}$$

Figure 6.11 shows the truth table, minterm Karnaugh map and maxterm Karnaugh map of the three-variable Boolean function

$$Y = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.\overline{C} \tag{6.46}$$

$$Y = (\overline{A} + \overline{B} + \overline{C}).(\overline{A} + B + \overline{C}).(A + \overline{B} + \overline{C}).(A + B + \overline{C}) \tag{6.47}$$

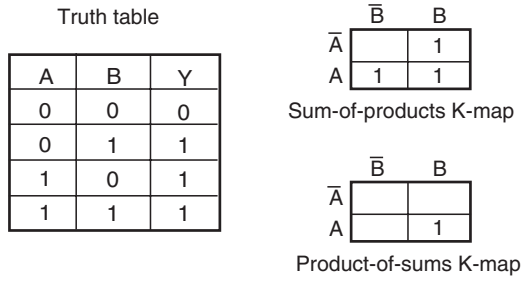

Figure 6.10 Two-variable Karnaugh maps.

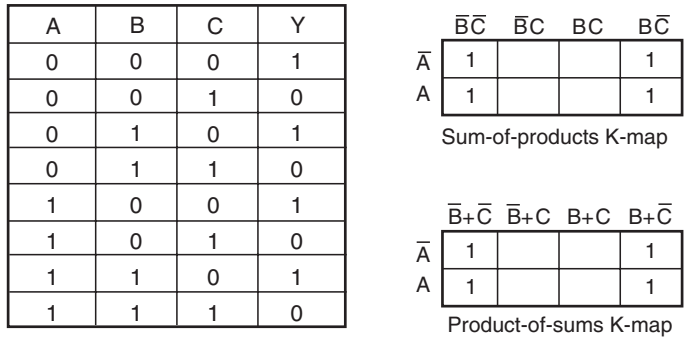

Figure 6.11 Three-variable Karnaugh maps.

Figure 6.12 shows the truth table, minterm Karnaugh map and maxterm Karnaugh map of the four-variable Boolean function

$$Y = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D \quad (6.48)$$

$$\begin{aligned} Y = {} & (A+B+\overline{C}+D).(A+B+\overline{C}+\overline{D}).(A+\overline{B}+\overline{C}+D).(A+\overline{B}+\overline{C}+\overline{D}) \\ & .(\overline{A}+B+\overline{C}+D).(\overline{A}+B+\overline{C}+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+D).(\overline{A}+\overline{B}+\overline{C}+\overline{D}) \end{aligned} \quad (6.49)$$

To illustrate the process of forming groups and then writing the corresponding minimized Boolean expression, Figs 6.13(a) and (b) respectively show minterm and maxterm Karnaugh maps for the Boolean functions expressed by equations (6.50) and (6.51). The minimized expressions as deduced from Karnaugh maps in the two cases are given by Equation (6.52) in the case of the minterm Karnaugh map and Equation (6.53) in the case of the maxterm Karnaugh map:

$$Y = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.C.\overline{D} + A.B.\overline{C}.D + A.B.C.D \quad (6.50)$$

$$\begin{aligned} Y = {} & (A+B+C+\overline{D}).(A+B+\overline{C}+\overline{D}).(A+\overline{B}+C+\overline{D}).(A+\overline{B}+C+\overline{D}).(A+\overline{B}+\overline{C}+\overline{D}) \\ & .(A+\overline{B}+\overline{C}+D).(\overline{A}+\overline{B}+C+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+\overline{D}).(\overline{A}+B+C+\overline{D}).(\overline{A}+B+\overline{C}+\overline{D}) \end{aligned} \quad (6.51)$$

$$Y = \overline{B}.\overline{D} + B.D \quad (6.52)$$

$$Y = \overline{D}.(A+\overline{B}) \quad (6.53)$$

Truth table

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

	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	1	1		
$\overline{A}B$	1	1		
AB	1	1		
$A\overline{B}$	1	1		

Sum-of-products K-map

	$\overline{C}+\overline{D}$	$\overline{C}+D$	$C+D$	$C+\overline{D}$
$\overline{A}+\overline{B}$	1	1		
$\overline{A}+B$	1	1		
$A+B$	1	1		
$A+\overline{B}$	1	1		

Product-of-sums K-map

Figure 6.12 Four-variable Karnaugh maps.

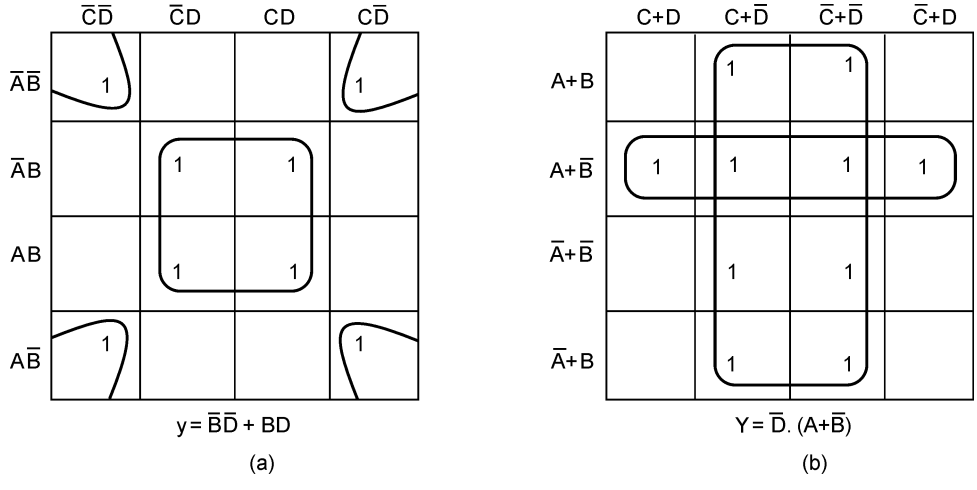

Figure 6.13 Group formation in minterm and maxterm Karnaugh maps.

6.6.2 *Karnaugh Map for Boolean Expressions with a Larger Number of Variables*

The construction of Karnaugh maps for a larger number of variables is a complex and cumbersome exercise, although manageable up to six variables. Five- and six-variable representative Karnaugh maps are shown in Figs 6.14(a) and (b) respectively. One important point to remember while forming groups in Karnaugh maps involving more than four variables is that terms equidistant from the central horizontal and central vertical lines are considered adjacent. These lines are shown thicker in Figs 6.14(a) and (b). Squares marked 'X' in Figs 6.14(a) and (b) are adjacent and therefore can be grouped.

Boolean expressions with more than four variables can also be represented by more than one four-variable map. Five-, six-, seven- and eight-variable Boolean expressions can be represented by two, four, eight and 16 four-variable maps respectively. In general, an n-variable Boolean expression can be represented by 2^{n-4} four-variable maps. In such multiple maps, groups are made as before, except that, in addition to adjacencies discussed earlier, corresponding squares in two adjacent maps are also considered adjacent and can therefore be grouped. We will illustrate the process of formation of groups in multiple Karnaugh maps with a larger number of variables with the help of examples. Consider the five-variable Boolean function given by the equation

$$Y = A.\overline{B}.\overline{C}.D.E + A.\overline{B}.C.D.E + \overline{A}.B.C.\overline{D}.E + A.B.\overline{C}.D.E + \overline{A}.\overline{B}.C.D.\overline{E} + \overline{A}.B.C.D.\overline{E} + A.B.C.D.\overline{E} + A.\overline{B}.C.D.\overline{E} + \overline{A}.B.C.\overline{D}.\overline{E} \quad (6.54)$$

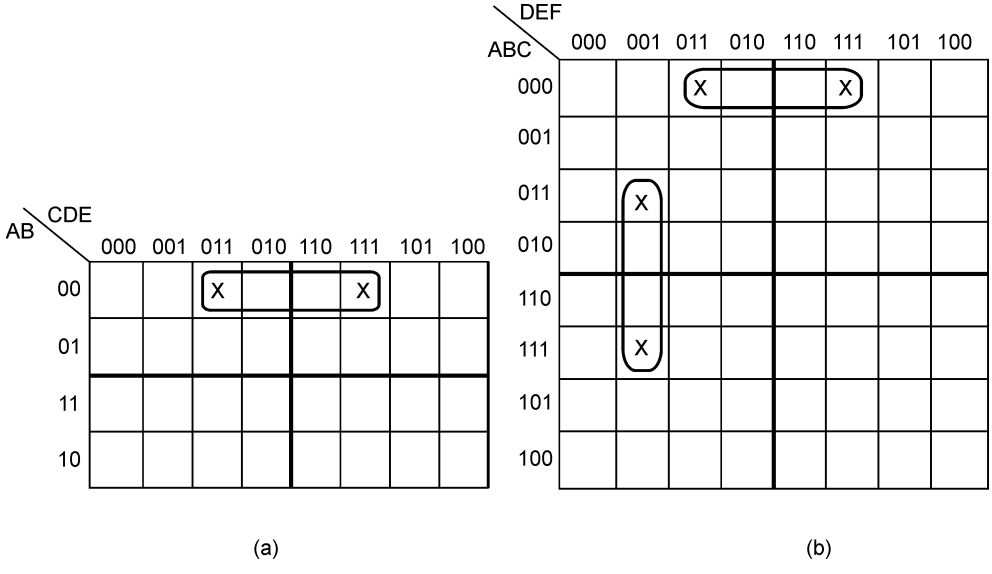

Figure 6.14 Five-variable and six-variable Karnaugh maps.

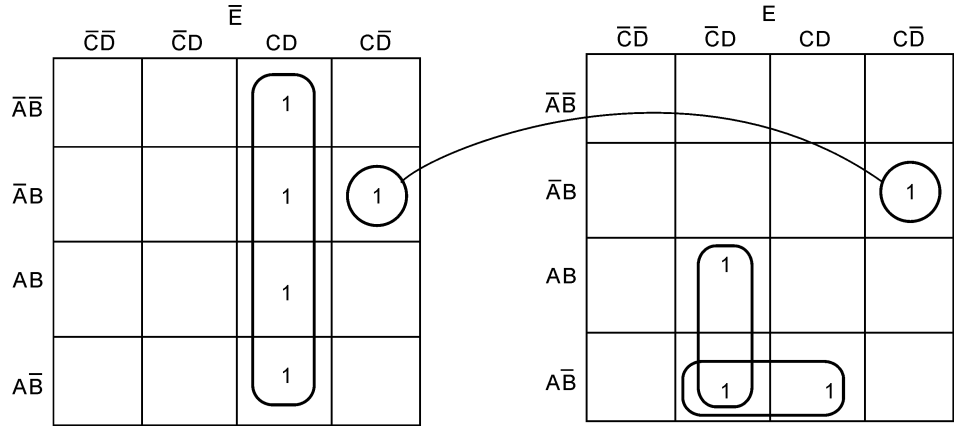

Figure 6.15 Multiple Karnaugh map for a five-variable Boolean function.

The multiple Karnaugh map for this five-variable expression is shown in Fig. 6.15. The construction of the Karnaugh map and the formation of groups are self-explanatory.

The minimized expression is given by the equation

$$Y = C.D.\overline{E} + \overline{A}.B.C.\overline{D} + A.\overline{C}.D.E + A.\overline{B}.D.E \tag{6.55}$$

As another illustration, consider a six-variable Boolean function given by the equation

$$\begin{aligned} Y = {} & \overline{A}.B.C.\overline{D}.\overline{E}.\overline{F} + A.B.\overline{C}.D.\overline{E}.F + \overline{A}.\overline{B}.\overline{C}.\overline{D}.\overline{E}.\overline{F} + A.B.C.D.E.F + A.\overline{B}.C.D.E.\overline{F} \\ & + \overline{A}.\overline{B}.\overline{C}.\overline{D}.\overline{E}.F + \overline{A}.B.C.\overline{D}.E.\overline{F} \end{aligned} \tag{6.56}$$

Figure 6.16 gives the Karnaugh map for this six-variable Boolean function, comprising four four-variable Karnaugh maps. The figure also shows the formation of groups. The minimized expression is given by the equation

$$Y = \overline{A}.\overline{B}.\overline{C}.\overline{D}.\overline{E} + \overline{A}.B.C.\overline{D}.\overline{F} + A.\overline{B}.C.D.E.\overline{F} + A.B.\overline{C}.D.\overline{E}.F + A.B.C.D.E.F \tag{6.57}$$

Example 6.11

Minimize the Boolean function

$$f(A, B, C) = \sum 0, 1, 3, 5 + \sum_{\phi} 2, 7$$

using the mapping method in both minimized sum-of-products and product-of-sums forms.

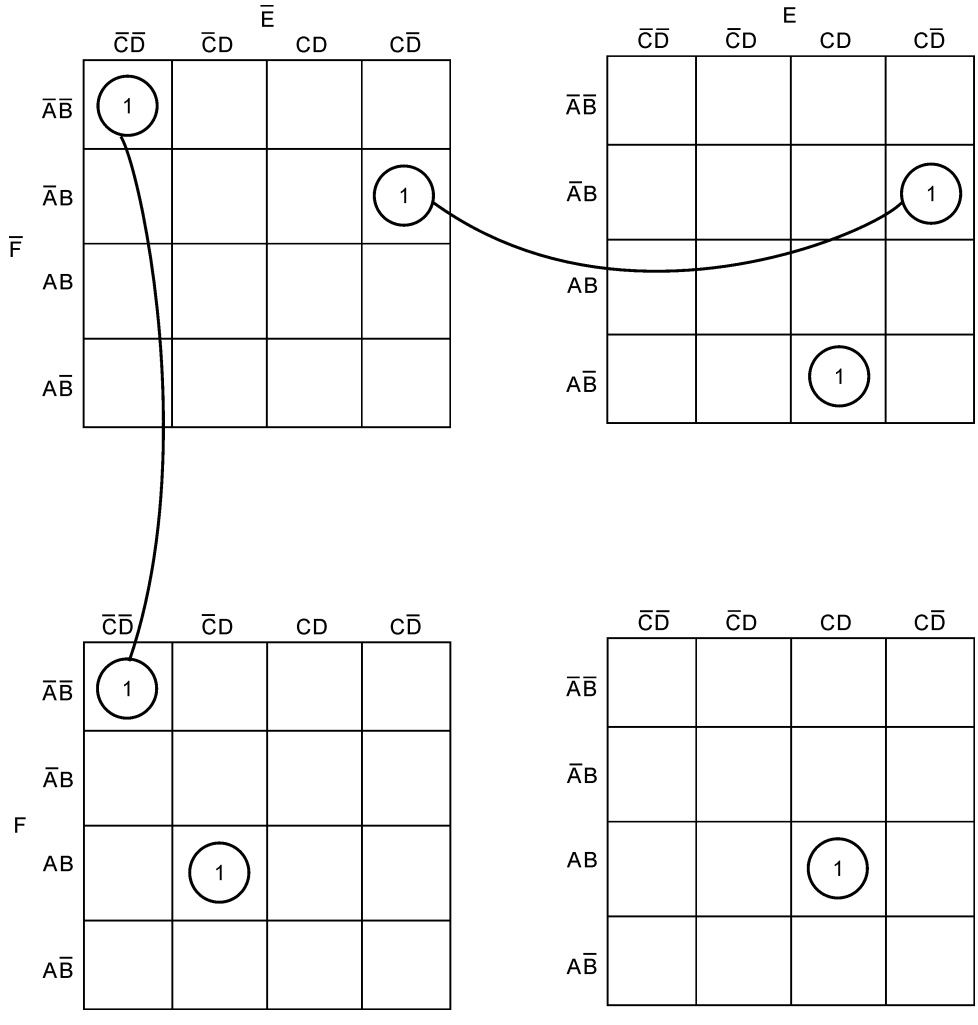

Figure 6.16 Multiple Karnaugh map for a six-variable Boolean function.

Solution

- $f(A, B, C) = \sum 0, 1, 3, 5 + \sum_{\phi} 2, 7 = \prod 4, 6 + \prod_{\phi} 2, 7.$
- From given Boolean functions in Σ and Π notation, we can write sum-of-products and product-of-sums Boolean expressions as follows:

$$f(A, B, C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C \tag{6.58}$$

$$f(A, B, C) = (\overline{A} + B + C).(\overline{A} + \overline{B} + C) \tag{6.59}$$

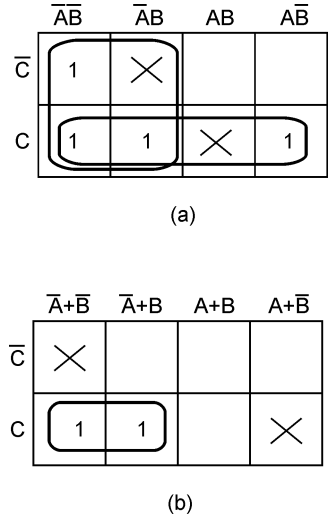

Figure 6.17 Example 6.11.

- The 'don't care' input combinations for the sum-of-products Boolean expression are $\overline{A}.B.\overline{C}$, $A.B.C$.
- The 'don't care' input combinations for the product-of-sums expression are $(A+\overline{B}+C).(\overline{A}+\overline{B}+\overline{C})$.
- The Karnaugh maps for the two cases are shown in Figs 6.17(a) and (b).
- The minimized sum-of-products and product-of-sums Boolean functions are respectively given by the equations

$$f(A, B, C) = C + \overline{A} \tag{6.60}$$

$$f(A, B, C) = \overline{A} + C \tag{6.61}$$

6.6.3 Karnaugh Maps for Multi-Output Functions

Karnaugh maps can be used for finding minimized Boolean expressions for multi-output functions. To begin with, a Karnaugh map is drawn for each function following the guidelines described in the earlier pages. In the second step, two-function Karnaugh maps are drawn. In the third step, three-function Karnaugh maps are drawn. The process continues until we have a single all-function Karnaugh map. As an illustration, for a logic system having four outputs, the first step would give four Karnaugh maps for individual functions. The second step would give six two-function Karnaugh maps (1−2, 1−3, 1−4, 2−3, 2−4 and 3−4). The third step would yield four three−function Karnaugh maps (1−2−3, 1−2−4, 1−3−4 and 2−3−4) and lastly we have one four-function Karnaugh map. A multifunction Karnaugh map is basically an intersection of the Karnaugh maps of the functions involved. That is, a '1' appears in a square of a multifunction map only if a '1' appears in the corresponding squares of the maps of all the relevant functions. To illustrate further, a two-function map involving functions 1 and 2 would be an intersection of maps for functions 1 and 2. In the two-function map, squares will have a '1' only when the corresponding squares in functions 1 and 2 also have a '1'. Figure 6.18 illustrates the formation of a three-function Karnaugh map from three given individual functions.

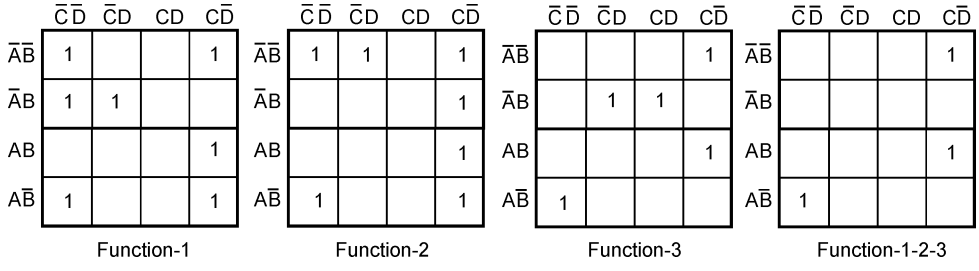

Figure 6.18 Three-function Karnaugh map.

The formation of groups begins with the largest multifunction map, which is nothing but the intersection of maps of all individual functions. Then we move to the Karnaugh maps one step down the order. The process continues until we reach the maps corresponding to individual functions. The groups in all the Karnaugh maps other than the largest map are formed subject to the condition that, once a group is identified in a certain function, then the same cannot be identified in any map of a subset of that function. For example, a group identified in a four-function map cannot be identified in a three-, two- or one-function map. With the formation of groups, prime implicants are identified. These prime implicants can be compiled in the form of a table along with input combinations of different output functions in the same way as for the tabular method to write minimized expressions. If the expressions corresponding to different output functions are not very complex, then the minimized expressions can even be written directly from the set of maps.

Example 6.12

Using Karnaugh maps, write the minimized Boolean expressions for the output functions of a two-output logic system whose outputs Y_1 and Y_2 are given by the following Boolean functions:

$$Y_1 = \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.C + \overline{A}.\overline{B}.\overline{C} \tag{6.62}$$

$$Y_2 = \overline{A}.\overline{B}.C + A.B.\overline{C} + \overline{A}.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.C \tag{6.63}$$

Solution

The individual Karnaugh maps and the two-function map are shown in Fig. 6.19 along with the formation of groups. The prime implicant table along with the input combinations for the two output functions is given below:

Y_1				Y_2					Prime implicants		
000	010	100	111	000	001	101	110	111			
✓				✓					0	0	0
			✓					✓	1	1	1
✓	✓								0	–	0
✓		✓							–	0	0
							✓	✓	1	1	–
					✓	✓			–	0	1

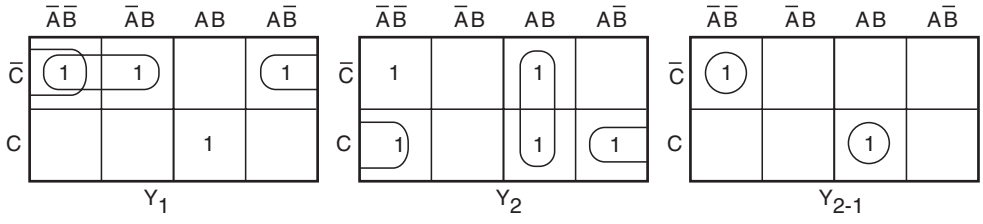

Figure 6.19 Example 6.12.

The minimized expressions for Y_1 and Y_2 are as follows:

$$Y_1 = \overline{B}.\overline{C} + \overline{A}.\overline{C} + A.B.C \tag{6.64}$$

$$Y_2 = A.B + \overline{A}.\overline{B}.\overline{C} + \overline{B}.C \tag{6.65}$$

Example 6.13

Write the simplified Boolean expression given by the Karnaugh map shown in Fig. 6.20.

Solution

- The Karnaugh map is shown in Fig. 6.21.
- Consider the group of four 1s at the top left of the map. It yields a term $\overline{A}.\overline{C}$.
- Consider the group of four 1s, two on the extreme left and two on the extreme right. This group yields a term $\overline{A}.\overline{D}$.
- The third group of two 1s is in the third row of the map. The third row corresponds to the intersection of A and B, as is clear from the map. Therefore, this group yields a term ABC.
- The simplified Boolean expression is given by $\overline{A}.\overline{C} + \overline{A}.\overline{D} + A.B.C$.

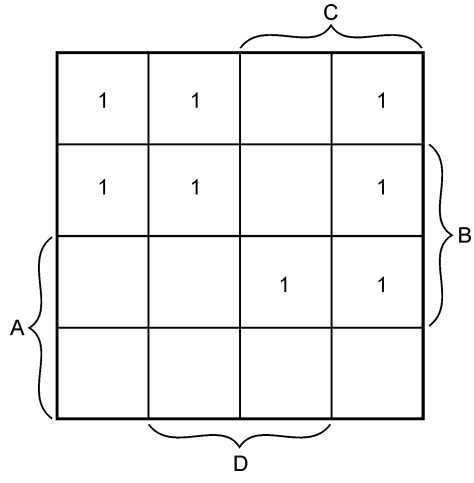

Figure 6.20 Example 6.13.

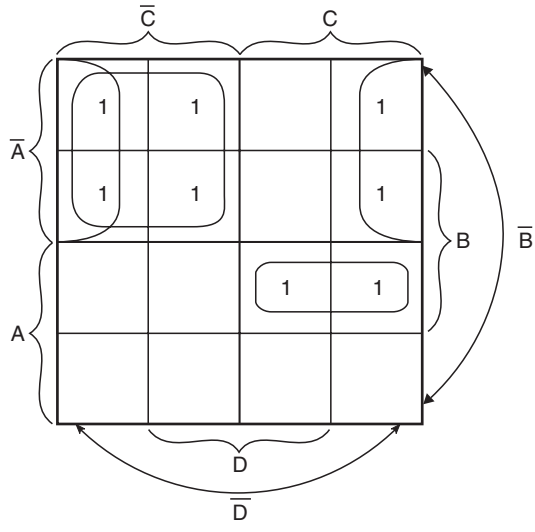

Figure 6.21 Solution to example 6.13.

Example 6.14

Minimizing a given Boolean expression using the Quine–McCluskey tabular method yields the following prime implicants: −0−0, −1−1, 1−10 and 0−00. Draw the corresponding Karnaugh map.

Solution

- As is clear from the prime implicants, the expression has four variables. If the variables are assumed to be A, B, C and D, then the given prime implicants correspond to the following terms:

1. $-0-0 \rightarrow \overline{B}.\overline{D}.$
2. $-1-1 \rightarrow B.D.$
3. $1-10 \rightarrow A.C.\overline{D}.$
4. $0-00 \rightarrow \overline{A}.\overline{C}.\overline{D}.$

- The Karnaugh map can now be drawn as shown in Fig. 6.22.

Example 6.15

$\overline{A}.B+C.D$ is a simplified Boolean expression of the expression $A.B.C.D+\overline{A}.\overline{B}.C.D+\overline{A}.B.$ Determine if there are any 'don't care' entries.

Solution

The expanded version of the given expression is given by the equation

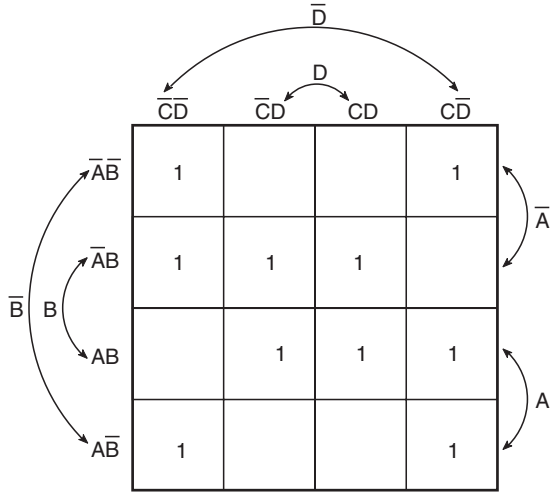

Figure 6.22 Solution to example 6.14.

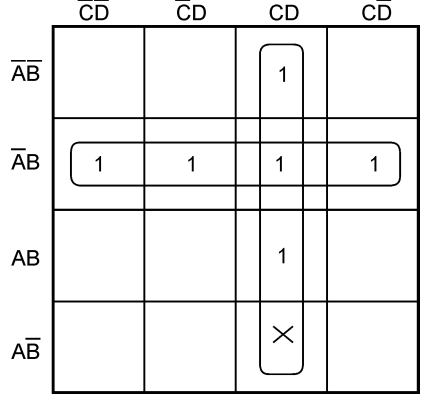

Figure 6.23 Example 6.15.

$$A.B.C.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.(\overline{C}.\overline{D} + \overline{C}.D + C.D + C.\overline{D}) \tag{6.66}$$

$$= A.B.C.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} \tag{6.63}$$

The Karnaugh map for this Boolean expression is shown in Fig. 6.23. Now, if it is to be a simplified version of the expression $\overline{A}.B + C.D$, then the lowermost square in the *CD* column should not be empty. This implies that there is a 'don't care' entry. This has been reflected in the map by putting X in the relevant square. With the groups formed along with the 'don't care' entry, the simplified expression becomes the one stated in the problem.

Review Questions

1. Read the following statements carefully. For each one of these, identify the law associated with it. Define the law and illustrate the same with one or two examples.

   (a) While a NAND gate is equivalent to a bubbled OR gate, a NOR gate is equivalent to a bubbled AND gate.
   (b) When all the inputs of an AND gate or an OR gate are tied together to get a single-input, single-output gate, both AND and OR gates with all their inputs tied together produce an output that is the same as the input.
   (c) When a variable is ORed with its complement the result is a logic '1', and when it is ANDed with its complement the result is a logic '0', irrespective of the logic status of the variable.
   (d) When two variables are ANDed and the result of the AND operation is ORed with one of the variables, the result is that variable. Also, when two variables are ORed and the result of the OR operation is ANDed with one of the variables, the result is that variable.

2. Write both sum-of-products and product-of-sums Boolean expressions for (a) a two-input AND gate, (b) a two-input NAND-gate, (c) a two-input EX-OR gate and (d) a two-input NOR gate from their respective truth tables.
3. What do you understand by canonical and expanded forms of Boolean expressions? Illustrate with examples.
4. With the help of an example, prove that in an n-variable Karnaugh map, a group formed with 2^{n-m} 1s will yield a term having m literals, where m = 1, 2, 3, . . . , n.
5. With the help of an example, prove that the dual of the complement of a Boolean expression is the same as the complement of the dual of the same.

Problems

1. Simplify the following Boolean expressions:

   (a) $A.B.C + A.B.\overline{C} + A.\overline{B}.C + A.\overline{B}.\overline{C} + \overline{A}.B.C + \overline{A}.B.\overline{C} + \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C$;
   (b) $(\overline{A} + B + \overline{C}).(\overline{A} + B + C).(C + D).(C + D + E)$.

   (a) 1; (b) $(\overline{A} + B).(C + D)$

2. (a) Find the dual of $A.B.C.\overline{D} + A.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.\overline{C}.\overline{D}$.
   (b) Find the complement of $A + [(B + \overline{C}).D + \overline{E}].F$.

   (a) $(A + B + C + \overline{D}).(A + \overline{B} + \overline{C} + D).(\overline{A} + \overline{B} + \overline{C} + \overline{D})$; (b) $\overline{A}.[(\overline{B}.C + \overline{D}).E + \overline{F}]$

3. The dual of the complement of a certain Boolean expression is given by $A.B.C + \overline{D}.E + B.\overline{C}.E$. Find the expression.

   $\overline{A}.\overline{B}.\overline{C} + D.\overline{E} + \overline{B}.C.\overline{E}$

4. Consider the Boolean expression given by

$$\overline{B}.\overline{C}.\overline{D}.\overline{E} + B.\overline{C}.\overline{D}.E + \overline{A}.B.C.E + A.B.C.D.E + A.\overline{B}.C.\overline{D}.\overline{E} + \overline{A}.B.\overline{C}.D.E + \overline{A}.\overline{B}.D.\overline{E} + \overline{A}.\overline{B}.C.\overline{D}.\overline{E} + A.\overline{B}.\overline{C}.D.\overline{E}$$

The simplified version of this Boolean expression is given by $B.E + \overline{B}.D.\overline{E} + \overline{B}.\overline{D}.\overline{E}$. Determine if there are any 'don't care' entries. If yes, find them.

Yes, $A.B.\overline{C}.D.E$, $A.B.C.\overline{D}.E$, $A.\overline{B}.C.D.\overline{E}$

5. Write minterm and maxterm Boolean functions expressed by $f(A, B, C) = \Pi\ 0, 3, 7$

minterm: $\overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C}$

maxterm: $(A + B + C).(A + \overline{B} + \overline{C}).(\overline{A} + \overline{B} + \overline{C})$

6. Write a simplified maxterm Boolean expression for Π 0, 4, 5, 6, 7, 10, 14 using the Karnaugh mapping method.

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

7. Simplify the following Boolean functions using the Quine–McCluskey tabulation method:

   (a) $f(A, B, C, D, E, F, G) = \Sigma\ (20, 21, 28, 29, 52, 53, 60, 61)$;
   (b) $f(A, B, C, D, E, F) = \Sigma\ (6, 9, 13,18,19, 25, 26, 27, 29, 41, 45, 57, 61)$.

(a) $\overline{A}.C.E.\overline{F}$; *(b)* $C.\overline{E}.F + \overline{A}.B.\overline{D}.E + \overline{A}.\overline{B}.\overline{C}.D.E.\overline{F}$

8. (a) Simplify the Boolean function $f(X, Y, Z) = Y.Z + \overline{X}.\overline{Z}$ for the 'don't care' condition expressed as $X.\overline{Y} + X.Y.\overline{Z} + \overline{X}.\overline{Y}.Z$.
   (b) Simplify the Boolean function given by $f(A, B, C) = (A + B + C).(\overline{A} + B + \overline{C}).(A + \overline{B} + C)$ for the don't care condition expressed as $(\overline{A} + \overline{B}).(\overline{A} + B + C)$.

(a) 1; (b) $\overline{A}.C$

Further Reading

1. Holdsworth, B. and Woods, C. (2002) *Digital Logic Design*, Newnes, Oxford, UK.
2. Chen, W.-K. (2003) *Logic Design*, CRC Press, FL, USA.
3. Floyd, T. L. (2005) *Digital Fundamentals*, Prentice-Hall Inc., USA.
4. Tokheim, R. L. (1994) *Schaum's Outline Series of Digital Principles*, McGraw-Hill Companies Inc., USA.