The logic diagram of the simplified expression is drawn in Fig. l-6(b) and Fig. l-6(c). It requires only four gates rather than the six gates used in the circuit of Fig. l-6(a). The two circuits are equivalent and produce the same truth table relationship between inputs *A*, *B*, *C* and output *F*.

Complement of a Function

The complement of a function *F* when expressed in a truth table is obtained by interchanging l's and O's in the values of *F* in the truth table. When the function is expressed in algebraic form, the complement of the function can be derived by means of DeMorgan's theorem. The general form of DeMorgan's theorem can be expressed as follows:

$$(x_1 + x_2 + x_3 + \cdots + x_n)' = x_1' x_2' x_3' \cdots x_n'$$

$$(x_1 x_2 x_3 \cdots x_n)' = x_1' + x_2' + x_3' + \cdots + x_n'$$

From the general DeMorgan's theorem we can derive a simple procedure for obtaining the complement of an algebraic expression. This is done by changing all OR operations to AND operations and all AND operations to OR operations and then complementing each individual letter variable. As an example, consider the following expression and its complement:

$$F = AB + C'D' + B'D$$

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

The complement expression is obtained by interchanging AND and OR operations and complementing each individual variable. Note that the complement of *C'* is *C*.

1-4   Map Simplification

The complexity of the logic diagram that implements a Boolean function is related directly to the complexity of the algebraic expression from which the function is implemented. The truth table representation of a function is unique, but the function can appear in many different forms when expressed algebraically. The expression may be simplified using the basic relations of Boolean algebra. However, this procedure is sometimes difficult because it lacks specific rules for predicting each succeeding step in the manipulative process. Two methods of simplifying Boolean algebraic expressions are the map method and the tabular method. The map method is used for functions upto six variables. To manipulate functions of a large number of variables, the tabular method also known as the Quine-McCluskey method, is used. If a function to be minimized is not in a canonical form, it must first be converted into canonical form before applying Quine-McCluskey tabular

procedure. Another tabular method, known as the iterative consensus method, begins the simplification process even if the function is not in a canonical form. The map method provides a simple, straightforward procedure for simplifying Boolean expressions. This method may be regarded as a pictorial arrangement of the truth table which allows an easy interpretation for choosing the minimum number of terms needed to express the function algebraically. The map method is also known as the Karnaugh map or K-map.

minterm   Each combination of the variables in a truth table is called a minterm. For example, the truth table of Fig. 1-3 contains eight minterms. When expressed in a truth table a function of n variables will have 2^n minterms, equivalent to the 2^n binary numbers obtained from n bits. A Boolean function is equal to 1 for some minterms and to 0 for others. The information contained in a truth table may be expressed in compact form by listing the decimal equivalent of those minterms that produce a 1 for the function. For example, the truth table of Fig. 1-3 can be expressed as follows:

$$F(x, y, z) = \Sigma \, (1, 4, 5, 6, 7)$$

The letters in parentheses list the binary variables in the order that they appear in the truth table. The symbol Σ stands for the sum of the minterms that follow in parentheses. The minterms that produce 1 for the function are listed in their decimal equivalent. The minterms missing from the list are the ones that produce 0 for the function.

The map is a diagram made up of squares, with each square representing one minterm. The squares corresponding to minterms that produce 1 for the function are marked by a 1 and the others are marked by a 0 or are left empty. By recognizing various patterns and combining squares marked by l's in the map, it is possible to derive alternative algebraic expressions for the function, from which the most convenient may be selected.

The maps for functions of two, three, and four variables are shown in Fig. 1-7. The number of squares in a map of n variables is 2^n. The 2^n minterms are listed by an equivalent decimal number for easy reference. The minterm numbers are assigned in an orderly arrangement such that adjacent squares represent minterms that differ by only one variable. The variable names are listed across both sides of the diagonal line in the corner of the map. The 0's and l's marked along each row and each column designate the value of the variables. Each variable under brackets contains half of the squares in the map where that variable appears unprimed. The variable appears with a prime (complemented) in the remaining half of the squares.

The minterm represented by a square is determined from the binary assignments of the variables along the left and top edges in the map. For example, minterm 5 in the three-variable map is 101 in binary, which may be obtained from the 1 in the second row concatenated with the 01 of the second column. This minterm represents a value for the binary variables A, B, and C, with A and C
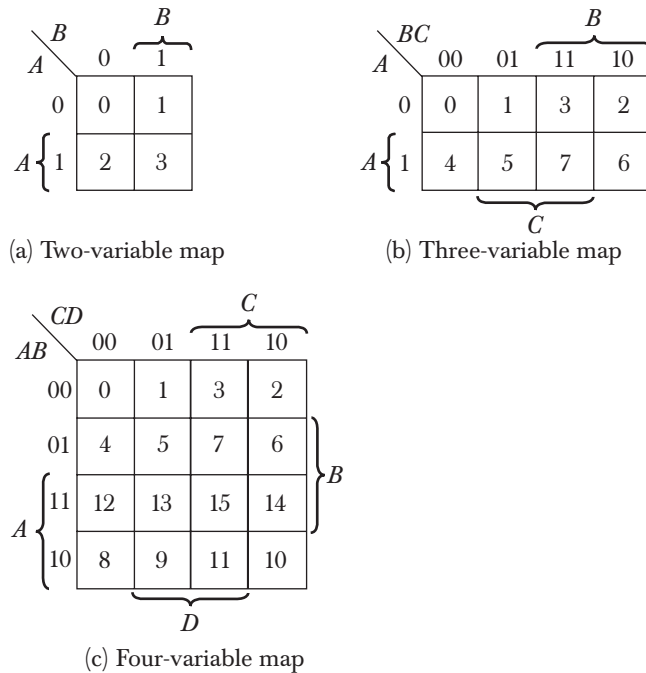
(a) Two-variable map

(b) Three-variable map

(c) Four-variable map

Figure 1-7 Maps for two-, three-, and four-variable functions.

being unprimed and B being primed (i.e., $AB'C$). On the other hand, minterm 5 in the four-variable map represents a minterm for four variables. The binary number contains the four bits 0101, and the corresponding term it represents is $A'BC'D$.

adjacent squares   Minterms of adjacent squares in the map are identical except for one variable, which appears complemented in one square and uncomplemented in the adjacent square. According to this definition of adjacency, the squares at the extreme ends of the same horizontal row are also to be considered adjacent. The same applies to the top and bottom squares of a column. As a result, the four corner squares of a map must also be considered to be adjacent.

A Boolean function represented by a truth table is plotted into the map by inserting l's in those squares where the function is 1. The squares containing l's are combined in groups of adjacent squares. These groups must contain a number of squares that is an integral power of 2. Groups of combined adjacent squares may share one or more squares with one or more groups. Each group of squares represents an algebraic term, and the OR of those terms gives the simplified algebraic expression for the function. The following examples show the use of the map for simplifying Boolean functions.

In the first example we will simplify the Boolean function

$$F(A, B, C) = \Sigma(3, 4, 6, 7)$$
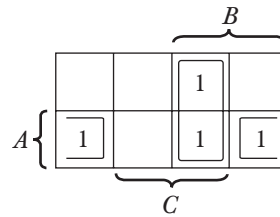
Figure 1-8   Map for $F(A, B, C) = \Sigma\,(3, 4, 6, 7)$.

The three-variable map for this function is shown in Fig. 1-8. There are four squares marked with l's, one for each minterm that produces 1 for the function. These squares belong to minterms 3, 4, 6, and 7 and are recognized from Fig. l-7(b). Two adjacent squares are combined in the third column. This column belongs to both B and C and produces the term BC. The remaining two squares with l's in the two corners of the second row are adjacent and belong to row A and the two columns of C', so they produce the term AC'. The simplified algebraic expression for the function is the OR of the two terms:

$$F = BC + AC'$$

The second example simplifies the following Boolean function:

$$F(A, B, C) = \Sigma\,(0, 2, 4, 5, 6)$$

The five minterms are marked with l's in the corresponding squares of the three-variable map shown in Fig. 1-9. The four squares in the first and fourth columns are adjacent and represent the term C'. The remaining square marked with a 1 belongs to minterm 5 and can be combined with the square of minterm 4 to produce the term AB'. The simplified function is

$$F = C' + AB'$$
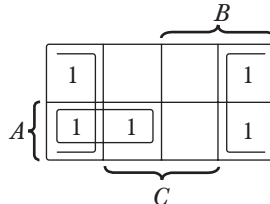
Figure 1-9   Map for $F(A, B, C) = \Sigma\,(3, 4, 6, 7)$.
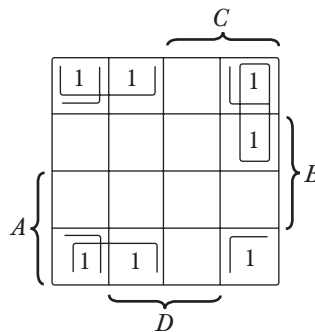
Figure 1-10   Map for $F(A, B, C, D) = \Sigma (0, 1, 2, 6, 8, 9, 10)$.

The third example needs a four-variable map.

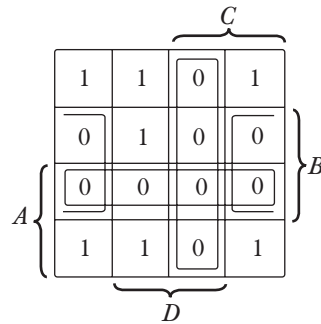$$F(A, B, C, D) = \Sigma (0, 1, 2, 6, 8, 9, 10)$$

The area in the map covered by this four-variable function consists of the squares marked with 1's in Fig. 1-10. The function contains 1's in the four corners that, when taken as a group, give the term $B'D'$. This is possible because these four squares are adjacent when the map is considered with top and bottom or left and right edges touching. The two 1's on the left of the top row are combined with the two 1's on the left of the bottom row to give the term $B'C'$. The remaining 1 in the square of minterm 6 is combined with minterm 2 to give the term $A'CD'$. The simplified function is

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

Product-of-Sums Simplification

The Boolean expressions derived from the maps in the preceding examples were expressed in sum-of-products form. The product terms are AND terms and the sum denotes the ORing of these terms. It is sometimes convenient to obtain the algebraic expression for the function in a product-of-sums form. The sums are OR terms and the product denotes the ANDing of these terms. With a minor modification, a product-of-sums form can be obtained from a map.

The procedure for obtaining a product-of-sums expression follows from the basic properties of Boolean algebra. The 1's in the map represent the minterms that produce 1 for the function. The squares not marked by 1 represent the minterms that produce 0 for the function. If we mark the empty squares with 0's and combine them into groups of adjacent squares, we obtain the complement of the function, F'. Taking the complement of F' produces an expression for F in product-of-sums form. The best way to show this is by example.

Figure 1-11 Map for $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$.

We wish to simplify the following Boolean function in both sum-of-products form and product-of-sums form:

$$F(A, B, C, D) = \Sigma (0, 1, 2, 5, 8, 9, 10)$$

The l's marked in the map of Fig. 1-11 represent the minterms that produce a 1 for the function. The squares marked with 0's represent the minterms not included in F and therefore denote the complement of F. Combining the squares with l's gives the simplified function in sum-of-products form:

$$F = B'D' + B'C' + A'C'D$$

If the squares marked with 0's are combined, as shown in the diagram, we obtain the simplified complemented function:

$$F' = AB + CD + BD'$$

Taking the complement of F', we obtain the simplified function in product-of-sums form:

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

The logic diagrams of the two simplified expressions are shown in Fig. 1-12. The sum-of-products expression is implemented in Fig. l-12(a) with a group of AND gates, one for each AND term. The outputs of the AND gates are connected to the inputs of a single OR gate. The same function is implemented in Fig. l-12(b) in product-of-sums form with a group of OR gates, one for each OR term. The outputs of the OR gates are connected to the inputs of a single AND gate. In each case it is assumed that the input variables are directly available in their complement, so inverters are not included. The pattern established in Fig. 1-12 is the general form by which any Boolean function is implemented when expressed in one of the standard
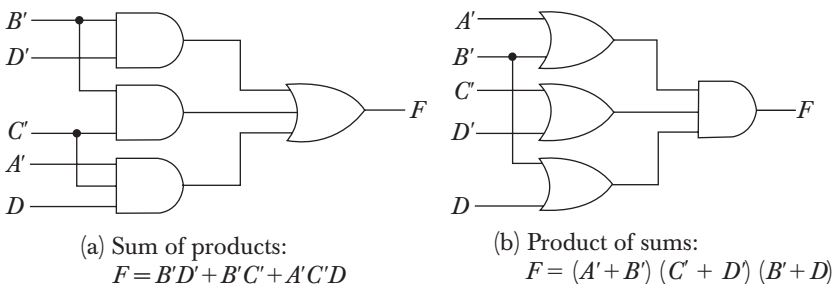
(a) Sum of products:
$F = B'D' + B'C' + A'C'D$

(b) Product of sums:
$F = (A' + B')(C' + D')(B' + D)$

Figure 1-12 Logic diagrams with AND and OR gates.

forms. AND gates are connected to a single OR gate when in sum-of-products form. OR gates are connected to a single AND gate when in product-of-sums form.

NAND implementation

A sum-of-products expression can be implemented with NAND gates as shown in Fig. l-13(a). Note that the second NAND gate is drawn with the graphic symbol of Fig. l-5(b). There are three lines in the diagram with small circles at both ends. Two circles in the same line designate double complementation, and since $(x')' = x$, the two circles can be removed and the resulting diagram is equivalent to the one shown in Fig. l-12(a). Similarly, a product-of-sums expression can be imple-

NOR implementation

mented with NOR gates as shown in Fig. 1-13(b). The second NOR gate is drawn with the graphic symbol of Fig. l-4(b). Again the two circles on both sides of each line may be removed, and the diagram so obtained is equivalent to the one shown in Fig. l-12(b).

Don't-Care Conditions

The l's and 0's in the map represent the minterms that make the function equal to 1 or 0. There are occasions when it does not matter if the function produces 0 or 1 for a given minterm. Since the function may be either 0 or 1, we say that we don't care what the function output is to be for this minterm. ==Minterms that may

don't-care conditions

produce either 0 or 1 for the function are said to be don't-care conditions and are marked with an $\times$ in the map. These don't-care conditions can be used to provide further simplification of the algebraic expression.==

Figure 1-13 Logic diagrams with NAND or NOR gates.

(a) With NAND gates

(b) With NOR gates
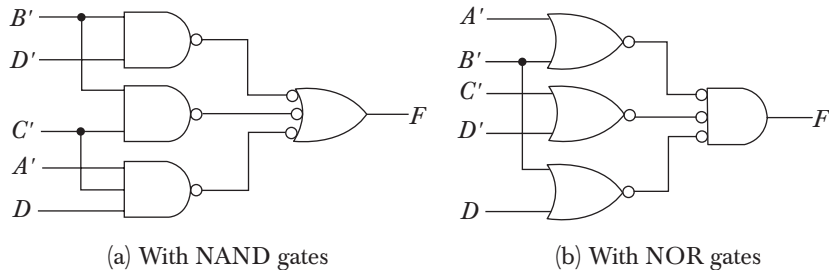
When choosing adjacent squares for the function in the map, the $\times$'s may be assumed to be either 0 or 1, whichever gives the simplest expression. In addition, an $\times$ need not be used at all if it does not contribute to the simplification of the function. In each case, the choice depends only on the simplification that can be achieved. As an example, consider the following Boolean function together with the don't-care minterms:

$$F(A, B, C) = \Sigma (0, 2, 6)$$
$$d(A, B, C) = \Sigma (1, 3, 5)$$

The minterms listed with F produce a 1 for the function. The don't-care minterms listed with d may produce either a 0 or a 1 for the function. The remaining minterms, 4 and 7, produce a 0 for the function. The map is shown in Fig. 1-14. The minterms of F are marked with 1's, those of d are marked with $\times$'s, and the remaining squares are marked with 0's. The 1's and $\times$'s are combined in any convenient manner so as to enclose the maximum number of adjacent squares. It is not necessary to include all or any of the $\times$'s, but all the 1's must be included. By including the don't care minterms 1 and 3 with the 1's in the first row we obtain the term A'. The remaining 1 for minterm 6 is combined with minterm 2 to obtain the term BC'. The simplified expression is

$$F = A' + BC'$$

Note that don't-care minterm 5 was not included because it does not contribute to the simplification of the expression. Note also that if don't-care minterms 1 and 3 were not included with the 1's, the simplified expression for F would have been

$$F = A'C' + BC'$$

This would require two AND gates and an OR gate, as compared to the expression obtained previously, which requires only one AND and one OR gate.

The function is determined completely once the $\times$'s are assigned to the 1's or 0's in the map. Thus the expression

$$F = A' + BC'$$

represents the Boolean function

$$F(A, B, C) = \Sigma (0, 1, 2, 3, 6)$$

It consists of the original minterms 0, 2, and 6 and the don't-care minterms 1 and 3. Minterm 5 is not included in the function. Since minterms 1, 3, and 5 were specified as being don't-care conditions, we have chosen minterms 1 and 3 to produce a 1 and minterm 5 to produce a 0. This was chosen because this assignment produces the simplest Boolean expression.