Solution

There are four 1s in the output column and the corresponding binary values are 011, 100, 110, and 111. Convert these binary values to product terms as follows:

$$
\begin{aligned}
011 &\longrightarrow \overline{A}BC \\
100 &\longrightarrow A\overline{B}\,\overline{C} \\
110 &\longrightarrow AB\overline{C} \\
111 &\longrightarrow ABC
\end{aligned}
$$

The resulting standard SOP expression for the output X is

$$X = \overline{A}BC + A\overline{B}\,\overline{C} + AB\overline{C} + ABC$$

For the POS expression, the output is 0 for binary values 000, 001, 010, and 101. Convert these binary values to sum terms as follows:

$$
\begin{aligned}
000 &\longrightarrow A + B + C \\
001 &\longrightarrow A + B + \overline{C} \\
010 &\longrightarrow A + \overline{B} + C \\
101 &\longrightarrow \overline{A} + B + \overline{C}
\end{aligned}
$$

The resulting standard POS expression for the output X is

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

Related Problem

By substitution of binary values, show that the SOP and the POS expressions derived in this example are equivalent; that is, for any binary value each SOP and POS term should either both be 1 or both be 0, depending on the binary value.

SECTION 4–7 CHECKUP

1. If a certain Boolean expression has a domain of five variables, how many binary values will be in its truth table?
2. In a certain truth table, the output is a 1 for the binary value 0110. Convert this binary value to the corresponding product term using variables W, X, Y, and Z.
3. In a certain truth table, the output is a 0 for the binary value 1100. Convert this binary value to the corresponding sum term using variables W, X, Y, and Z.

4–8 The Karnaugh Map

A Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression. As you have seen, the effectiveness of algebraic simplification depends on your familiarity with all the laws, rules, and theorems of Boolean algebra and on your ability to apply them. The Karnaugh map, on the other hand, provides a "cookbook" method for simplification. Other simplification techniques include the Quine-McCluskey method and the Espresso algorithm.

After completing this section, you should be able to

- Construct a Karnaugh map for three or four variables
- Determine the binary value of each cell in a Karnaugh map
- Determine the standard product term represented by each cell in a Karnaugh map
- Explain cell adjacency and identify adjacent cells

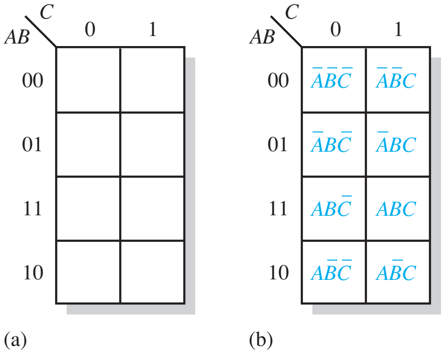
The purpose of a Karnaugh map is to simplify a Boolean expression.

A **Karnaugh map** is similar to a truth table because it presents all of the possible values of input variables and the resulting output for each value. Instead of being organized into columns and rows like a truth table, the Karnaugh map is an array of **cells** in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. Karnaugh maps can be used for expressions with two, three, four, and five variables, but we will discuss only 3-variable and 4-variable situations to illustrate the principles. *A discussion of 5-variable Karnaugh maps is available on the website*.

The number of cells in a Karnaugh map, as well as the number of rows in a truth table, is equal to the total number of possible input variable combinations. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is $2^4 = 16$.

The 3-Variable Karnaugh Map

The 3-variable Karnaugh map is an array of eight cells, as shown in Figure 4–25(a). In this case, *A*, *B*, and *C* are used for the variables although other letters could be used. Binary values of *A* and *B* are along the left side (notice the sequence) and the values of *C* are across the top. The value of a given cell is the binary values of *A* and *B* at the left in the same row combined with the value of *C* at the top in the same column. For example, the cell in the upper left corner has a binary value of 000 and the cell in the lower right corner has a binary value of 101. Figure 4–25(b) shows the standard product terms that are represented by each cell in the Karnaugh map.


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

(a) (b)

FIGURE 4–25 A 3-variable Karnaugh map showing Boolean product terms for each cell.

The 4-Variable Karnaugh Map

The 4-variable Karnaugh map is an array of sixteen cells, as shown in Figure 4–26(a). Binary values of *A* and *B* are along the left side and the values of *C* and *D* are across the top. The value of a given cell is the binary values of *A* and *B* at the left in the same row combined with the binary values of *C* and *D* at the top in the same column. For example, the cell in the upper right corner has a binary value of 0010 and the cell in the lower right corner has a binary value of 1010. Figure 4–26(b) shows the standard product terms that are represented by each cell in the 4-variable Karnaugh map.

Cell Adjacency

Cells that differ by only one variable are adjacent.

Cells with values that differ by more than one variable are not adjacent.

The cells in a Karnaugh map are arranged so that there is only a single-variable change between adjacent cells. **Adjacency** is defined by a single-variable change. In the 3-variable map the 010 cell is adjacent to the 000 cell, the 011 cell, and the 110 cell. The 010 cell is not adjacent to the 001 cell, the 111 cell, the 100 cell, or the 101 cell.

Physically, each cell is adjacent to the cells that are immediately next to it on any of its four sides. A cell is not adjacent to the cells that diagonally touch any of its corners. Also, the cells in the top row are adjacent to the corresponding cells in the bottom row and

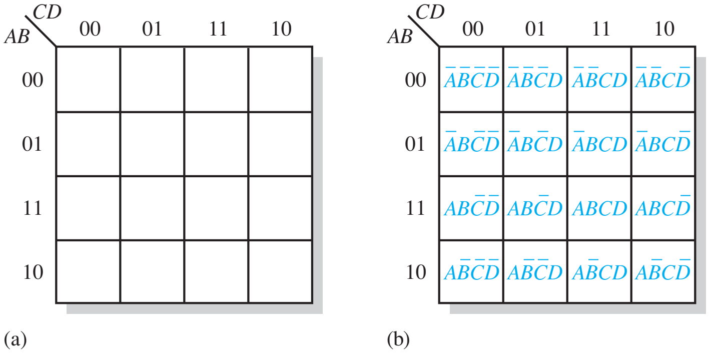

CD
AB
00
01
11
10
00
01
11
10
(a)
CD
AB
00
01
11
10
00
01
11
10
$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
$\overline{A}\,\overline{B}\,\overline{C}D$
$\overline{A}\,\overline{B}CD$
$\overline{A}\,\overline{B}C\overline{D}$
$\overline{A}B\overline{C}\,\overline{D}$
$\overline{A}B\overline{C}D$
$\overline{A}BCD$
$\overline{A}BC\overline{D}$
$AB\overline{C}\,\overline{D}$
$AB\overline{C}D$
$ABCD$
$ABC\overline{D}$
$A\overline{B}\,\overline{C}\,\overline{D}$
$A\overline{B}\,\overline{C}D$
$A\overline{B}CD$
$A\overline{B}C\overline{D}$
(b)

FIGURE 4–26 A 4-variable Karnaugh map.

the cells in the outer left column are adjacent to the corresponding cells in the outer right column. This is called "wrap-around" adjacency because you can think of the map as wrapping around from top to bottom to form a cylinder or from left to right to form a cylinder. Figure 4–27 illustrates the cell adjacencies with a 4-variable map, although the same rules for adjacency apply to Karnaugh maps with any number of cells.

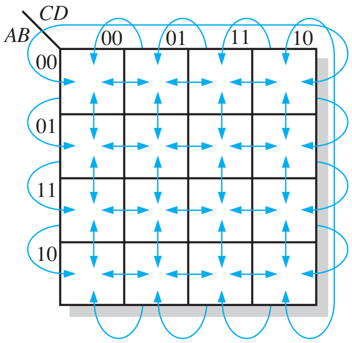


FIGURE 4–27 Adjacent cells on a Karnaugh map are those that differ by only one variable. Arrows point between adjacent cells.

The Quine-McCluskey Method

Minimizing Boolean functions using Karnaugh maps is practical only for up to four or five variables. Also, the Karnaugh map method does not lend itself to be automated in the form of a computer program.

The Quine-McCluskey method is more practical for logic simplification of functions with more than four or five variables. It also has the advantage of being easily implemented with a computer or programmable calculator.

The Quine-McCluskey method is functionally similar to Karnaugh mapping, but the tabular form makes it more efficient for use in computer algorithms, and it also gives a way to check that the minimal form of a Boolean function has been reached. This method is sometimes referred to as the *tabulation method*. An introduction to the Quine-McCluskey method is provided in Section 4–11.

Espresso Algorithm

Although the Quine-McCluskey method is well suited to be implemented in a computer program and can handle more variables than the Karnaugh map method, the result is still far from efficient in terms of processing time and memory usage. Adding a variable to the function will roughly double both of these parameters because the truth table length increases exponentially with the number of variables. Functions with a large number of variables have to be minimized with other methods such as the Espresso logic minimizer, which has become the de facto world standard. *An Espresso algorithm tutorial is available on the website*.

Compared to the other methods, Espresso is essentially more efficient in terms of reducing memory usage and computation time by several orders of magnitude. There is essentially no restrictions to the number of variables, output functions, and product terms of a combinational logic function. In general, tens of variables with tens of output functions can be handled by Espresso.

The Espresso algorithm has been incorporated as a standard logic function minimization step in most logic synthesis tools for programmable logic devices. For implementing a function in multilevel logic, the minimization result is optimized by factorization and mapped onto the available basic logic cells in the target device, such as an FPGA (Field-Programmable Gate Array).

SECTION 4–8 CHECKUP

1. In a 3-variable Karnaugh map, what is the binary value for the cell in each of the following locations:

(a) upper left corner **(b)** lower right corner

(c) lower left corner **(d)** upper right corner

2. What is the standard product term for each cell in Question 1 for variables X, Y, and Z?

3. Repeat Question 1 for a 4-variable map.

4. Repeat Question 2 for a 4-variable map using variables W, X, Y, and Z.

4–9 Karnaugh Map SOP Minimization

As stated in the last section, the Karnaugh map is used for simplifying Boolean expressions to their minimum form. A minimized SOP expression contains the fewest possible terms with the fewest possible variables per term. Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression. In this section, Karnaugh maps with up to four variables are covered.

After completing this section, you should be able to

- Map a standard SOP expression on a Karnaugh map
- Combine the 1s on the map into maximum groups
- Determine the minimum product term for each group on the map
- Combine the minimum product terms to form a minimum SOP expression
- Convert a truth table into a Karnaugh map for simplification of the represented expression
- Use “don’t care” conditions on a Karnaugh map

Mapping a Standard SOP Expression

For an SOP expression in standard form, a 1 is placed on the Karnaugh map for each product term in the expression. Each 1 is placed in a cell corresponding to the value of a product term. For example, for the product term $A\overline{B}C$, a 1 goes in the 101 cell on a 3-variable map.

When an SOP expression is completely mapped, there will be a number of 1s on the Karnaugh map equal to the number of product terms in the standard SOP expression. The cells that do not have a 1 are the cells for which the expression is 0. Usually, when working with SOP expressions, the 0s are left off the map. The following steps and the illustration in Figure 4–28 show the mapping process.

Step 1: Determine the binary value of each product term in the standard SOP expression. After some practice, you can usually do the evaluation of terms mentally.

Step 2: As each product term is evaluated, place a 1 on the Karnaugh map in the cell having the same value as the product term.

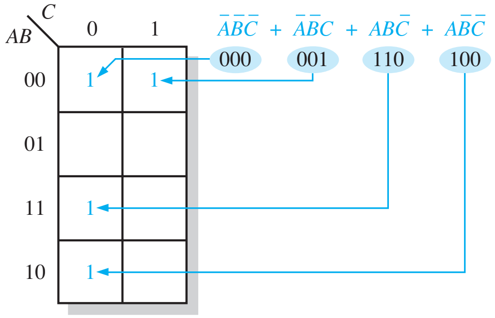

C
AB
0
1
00
01
11
10
1
1
1
1
$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C}$
000
001
110
100

FIGURE 4–28 Example of mapping a standard SOP expression.

EXAMPLE 4–23

Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}\,\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

Solution

Evaluate the expression as shown below. Place a 1 on the 3-variable Karnaugh map in Figure 4–29 for each standard product term in the expression.

$$\begin{array}{cccc} \overline{A}\,\overline{B}C + & \overline{A}B\overline{C} + & AB\overline{C} + & ABC \\ 0\,0\,1 & 0\,1\,0 & 1\,1\,0 & 1\,1\,1 \end{array}$$

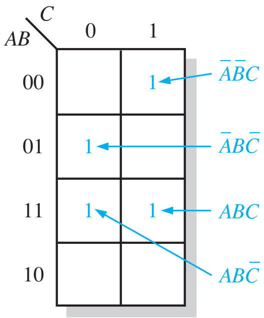


FIGURE 4–29

Related Problem

Map the standard SOP expression $\overline{A}BC + A\overline{B}C + A\overline{B}\,\overline{C}$ on a Karnaugh map.

EXAMPLE 4–24

Map the following standard SOP expression on a Karnaugh map:

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

Solution

Evaluate the expression as shown below. Place a 1 on the 4-variable Karnaugh map in Figure 4–30 for each standard product term in the expression.

$$\begin{array}{ccccccccccccc} \overline{A}\,\overline{B}CD & + & \overline{A}B\overline{C}\,\overline{D} & + & AB\overline{C}D & + & ABCD & + & AB\overline{C}\,\overline{D} & + & \overline{A}\,\overline{B}\,\overline{C}D & + & A\overline{B}C\overline{D} \\ 0011 & & 0100 & & 1101 & & 1111 & & 1100 & & 0001 & & 1010 \end{array}$$

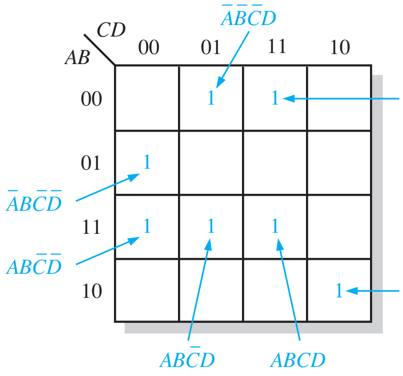

$\overline{A}\,\overline{B}\,\overline{C}D$
CD
AB
00
01
11
10
00
01
11
10
1
1
$\overline{A}\,\overline{B}CD$
1
$\overline{A}B\overline{C}\,\overline{D}$
1
1
1
$AB\overline{C}\,\overline{D}$
1
$A\overline{B}C\overline{D}$
$AB\overline{C}D$
$ABCD$

FIGURE 4–30

Related Problem

Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}BC\overline{D} + ABC\overline{D} + AB\overline{C}\,\overline{D} + ABCD$$

Mapping a Nonstandard SOP Expression

A Boolean expression must first be in standard form before you use a Karnaugh map. If an expression is not in standard form, then it must be converted to standard form by the procedure covered in Section 4–6 or by numerical expansion. Since an expression should be evaluated before mapping anyway, numerical expansion is probably the most efficient approach.

Numerical Expansion of a Nonstandard Product Term

Recall that a nonstandard product term has one or more missing variables. For example, assume that one of the product terms in a certain 3-variable SOP expression is $A\overline{B}$. This term can be expanded numerically to standard form as follows. First, write the binary value of the two variables and attach a 0 for the missing variable $\overline{C}$: 100. Next, write the binary value of the two variables and attach a 1 for the missing variable C: 101. The two resulting binary numbers are the values of the standard SOP terms $A\overline{B}\,\overline{C}$ and $A\overline{B}C$.

As another example, assume that one of the product terms in a 3-variable expression is B (remember that a single variable counts as a product term in an SOP expression). This term can be expanded numerically to standard form as follows. Write the binary value of the variable; then attach all possible values for the missing variables A and C as follows:

B
010
011
110
111

The four resulting binary numbers are the values of the standard SOP terms $\overline{A}B\overline{C}$, $\overline{A}BC$, $AB\overline{C}$, and ABC.

EXAMPLE 4–25

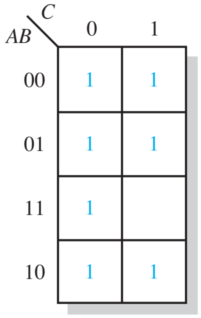
Map the following SOP expression on a Karnaugh map: $\overline{A} + A\overline{B} + AB\overline{C}$.

Solution

The SOP expression is obviously not in standard form because each product term does not have three variables. The first term is missing two variables, the second term is missing one variable, and the third term is standard. First expand the terms numerically as follows:

$\overline{A}$	$+ A\overline{B}$	$+ AB\overline{C}$
000	100	110
001	101	
010		
011		

Map each of the resulting binary values by placing a 1 in the appropriate cell of the 3-variable Karnaugh map in Figure 4–31.



AB \ C	0	1
00	1	1
01	1	1
11	1	
10	1	1

FIGURE 4–31

Related Problem

Map the SOP expression $BC + \overline{A}\,\overline{C}$ on a Karnaugh map.

EXAMPLE 4–26

Map the following SOP expression on a Karnaugh map:

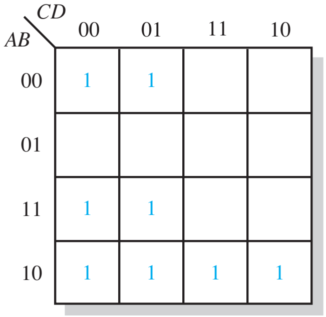
$$\overline{B}\,\overline{C} + A\overline{B} + AB\overline{C} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}CD$$

Solution

The SOP expression is obviously not in standard form because each product term does not have four variables. The first and second terms are both missing two variables, the third term is missing one variable, and the rest of the terms are standard. First expand the terms by including all combinations of the missing variables numerically as follows:

$\overline{B}\,\overline{C}$	$+ A\overline{B}$	$+ AB\overline{C}$	$+ A\overline{B}C\overline{D}$	$+ \overline{A}\,\overline{B}\,\overline{C}D$	$+ A\overline{B}CD$
0000	1000	1100	1010	0001	1011
0001	1001	1101			
1000	1010				
1001	1011				

Map each of the resulting binary values by placing a 1 in the appropriate cell of the 4-variable Karnaugh map in Figure 4–32. Notice that some of the values in the expanded expression are redundant.



AB \ CD	00	01	11	10
00	1	1		
01				
11	1	1		
10	1	1	1	1

FIGURE 4–32

Related Problem

Map the expression $A + \overline{C}D + AC\overline{D} + \overline{A}BC\overline{D}$ on a Karnaugh map.

Karnaugh Map Simplification of SOP Expressions

The process that results in an expression containing the fewest possible terms with the fewest possible variables is called **minimization**. After an SOP expression has been mapped, a minimum SOP expression is obtained by grouping the 1s and determining the minimum SOP expression from the map.

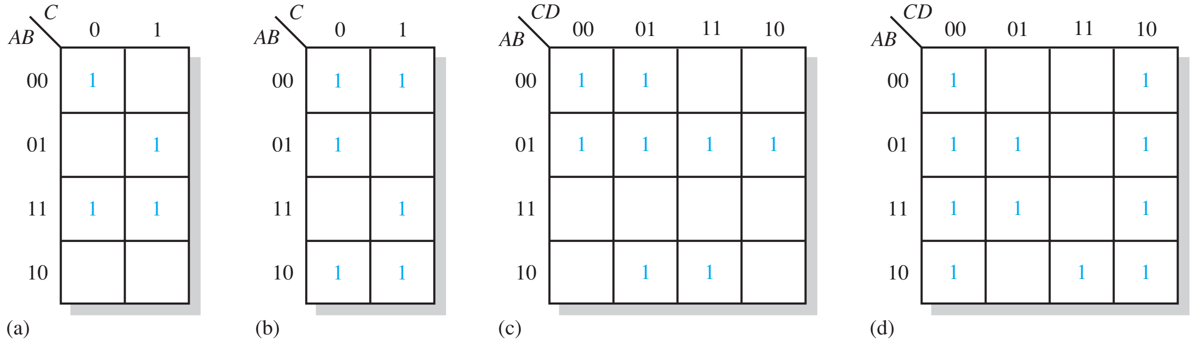
Grouping the 1s

You can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s. The goal is to maximize the size of the groups and to minimize the number of groups.

1. A group must contain either 1, 2, 4, 8, or 16 cells, which are all powers of two. In the case of a 3-variable map, $2^3 = 8$ cells is the maximum group.
2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.
3. Always include the largest possible number of 1s in a group in accordance with rule 1.
4. Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include noncommon 1s.

EXAMPLE 4–27

Group the 1s in each of the Karnaugh maps in Figure 4–33.



(a)

AB \ C	0	1
00	1	
01		1
11	1	1
10		

(b)

AB \ C	0	1
00	1	1
01	1	
11		1
10	1	1

(c)

AB \ CD	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10		1	1	

(d)

AB \ CD	00	01	11	10
00	1			1
01	1	1		1
11	1	1		1
10	1		1	1

FIGURE 4–33

Solution

The groupings are shown in Figure 4–34. In some cases, there may be more than one way to group the 1s to form maximum groupings.

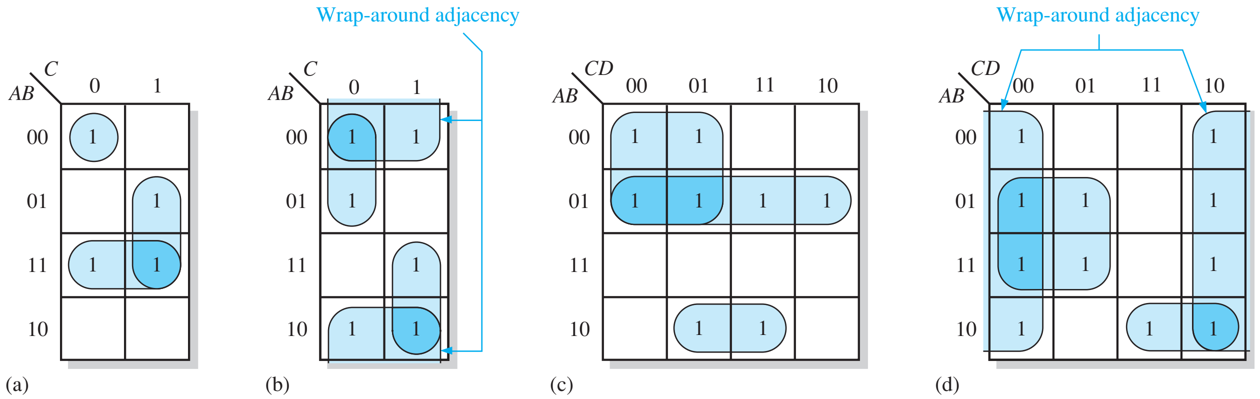


FIGURE 4–34

Related Problem

Determine if there are other ways to group the 1s in Figure 4–34 to obtain a minimum number of maximum groupings.

Determining the Minimum SOP Expression from the Map

When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining the resulting minimum SOP expression begins. The following rules are applied to find the minimum product terms and the minimum SOP expression:

1. Group the cells that have 1s. Each group of cells containing 1s creates one product term composed of all variables that occur in only one form (either uncomplemented or complemented) within the group. Variables that occur both uncomplemented and complemented within the group are eliminated. These are called *contradictory variables.*
2. Determine the minimum product term for each group.
   (a) For a 3-variable map:
   (1) A 1-cell group yields a 3-variable product term
   (2) A 2-cell group yields a 2-variable product term
   (3) A 4-cell group yields a 1-variable term
   (4) An 8-cell group yields a value of 1 for the expression
   (b) For a 4-variable map:
   (1) A 1-cell group yields a 4-variable product term
   (2) A 2-cell group yields a 3-variable product term
   (3) A 4-cell group yields a 2-variable product term
   (4) An 8-cell group yields a 1-variable term
   (5) A 16-cell group yields a value of 1 for the expression
3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression.

EXAMPLE 4–28

Determine the product terms for the Karnaugh map in Figure 4–35 and write the resulting minimum SOP expression.

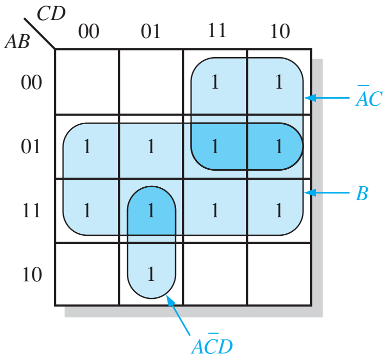


FIGURE 4–35

Solution

Eliminate variables that are in a grouping in both complemented and uncomplemented forms. In Figure 4–35, the product term for the 8-cell group is B because the cells within that group contain both A and $\overline{A}$, C and $\overline{C}$, and D and $\overline{D}$, which are eliminated. The 4-cell group contains B, $\overline{B}$, D, and $\overline{D}$, leaving the variables $\overline{A}$ and C, which form the product term $\overline{A}C$. The 2-cell group contains B and $\overline{B}$, leaving variables A, $\overline{C}$, and D which form the product term $A\overline{C}D$. Notice how overlapping is used to maximize the size of the groups. The resulting minimum SOP expression is the sum of these product terms:

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

Related Problem

For the Karnaugh map in Figure 4–35, add a 1 in the lower right cell (1010) and determine the resulting SOP expression.

EXAMPLE 4–29

Determine the product terms for each of the Karnaugh maps in Figure 4–36 and write the resulting minimum SOP expression.

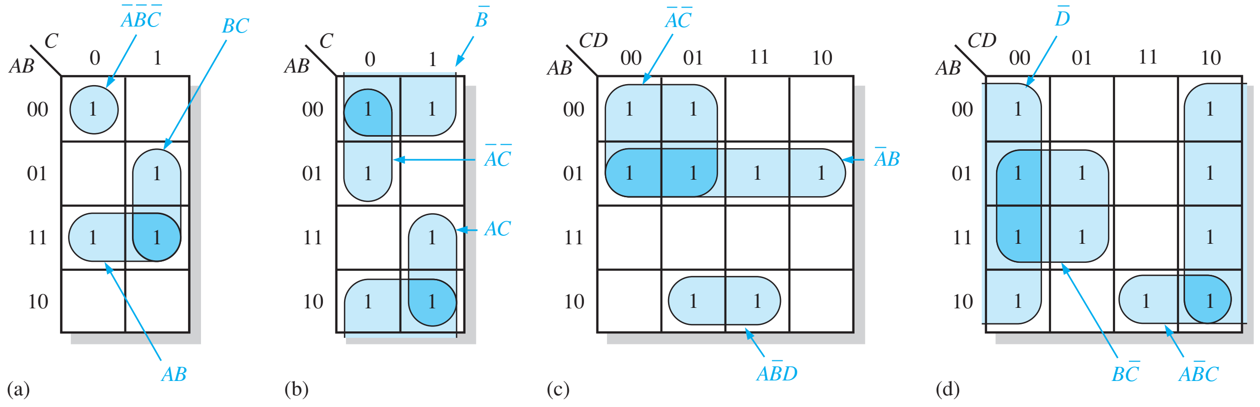


FIGURE 4–36

Solution

The resulting minimum product term for each group is shown in Figure 4–36. The minimum SOP expressions for each of the Karnaugh maps in the figure are

(a) $AB + BC + \overline{A}\,\overline{B}\,\overline{C}$

(b) $\overline{B} + \overline{A}\,\overline{C} + AC$

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

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

Related Problem

For the Karnaugh map in Figure 4–36(d), add a 1 in the 0111 cell and determine the resulting SOP expression.

EXAMPLE 4–30

Use a Karnaugh map to minimize the following standard SOP expression:

$$A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C}$$

Solution

The binary values of the expression are

$$101 + 011 + 001 + 000 + 100$$

Map the standard SOP expression and group the cells as shown in Figure 4–37.

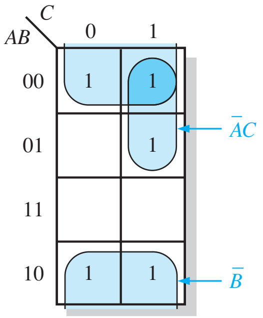


FIGURE 4–37

Notice the “wrap around” 4-cell group that includes the top row and the bottom row of 1s. The remaining 1 is absorbed in an overlapping group of two cells. The group of four 1s produces a single variable term, $\overline{B}$. This is determined by observing that within the group, $\overline{B}$ is the only variable that does not change from cell to cell. The group of two 1s produces a 2-variable term $\overline{A}C$. This is determined by observing that within the group, $\overline{A}$ and C do not change from one cell to the next. The product term for each group is shown. The resulting minimum SOP expression is

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

Keep in mind that this minimum expression is equivalent to the original standard expression.

Related Problem

Use a Karnaugh map to simplify the following standard SOP expression:

$$X\overline{Y}Z + XY\overline{Z} + \overline{X}YZ + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$$

EXAMPLE 4–31

Use a Karnaugh map to minimize the following SOP expression:

$$\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}CD + A\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$$

Solution

The first term $\overline{B}\,\overline{C}\,\overline{D}$ must be expanded into $A\overline{B}\,\overline{C}\,\overline{D}$ and $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ to get the standard SOP expression, which is then mapped; the cells are grouped as shown in Figure 4–38.

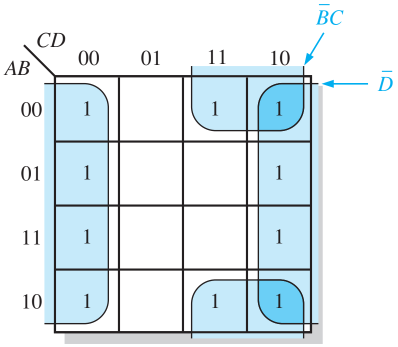


FIGURE 4–38

Notice that both groups exhibit "wrap around" adjacency. The group of eight is formed because the cells in the outer columns are adjacent. The group of four is formed to pick up the remaining two 1s because the top and bottom cells are adjacent. The product term for each group is shown. The resulting minimum SOP expression is

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

Keep in mind that this minimum expression is equivalent to the original standard expression.

Related Problem

Use a Karnaugh map to simplify the following SOP expression:

$$\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} + W\overline{X}YZ + W\overline{X}\,\overline{Y}Z + \overline{W}YZ + W\overline{X}\,\overline{Y}\,\overline{Z}$$

Mapping Directly from a Truth Table

You have seen how to map a Boolean expression; now you will learn how to go directly from a truth table to a Karnaugh map. Recall that a truth table gives the output of a Boolean expression for all possible input variable combinations. An example of a Boolean expression and its truth table representation is shown in Figure 4–39. Notice in the truth table that the output X is 1 for four different input variable combinations. The 1s in the output column of the truth table are mapped directly onto a Karnaugh map into the cells corresponding to the values of the associated input variable combinations, as shown in Figure 4–39. In the figure you can see that the Boolean expression, the truth table, and the Karnaugh map are simply different ways to represent a logic function.

"Don't Care" Conditions

Sometimes a situation arises in which some input variable combinations are not allowed. For example, recall that in the BCD code covered in Chapter 2, there are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111. Since these unallowed states

$X = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + AB\overline{C} + ABC$

Inputs			Output
A	*B*	*C*	*X*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

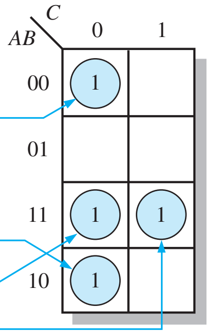


FIGURE 4–39 Example of mapping directly from a truth table to a Karnaugh map.

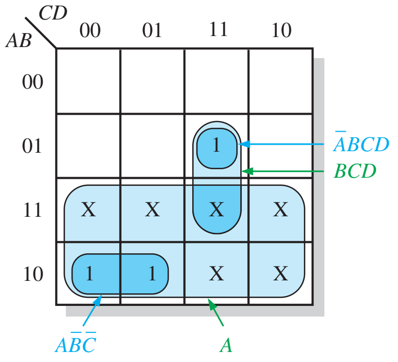
will never occur in an application involving the BCD code, they can be treated as **"don't care"** terms with respect to their effect on the output. That is, for these "don't care" terms either a 1 or a 0 may be assigned to the output; it really does not matter since they will never occur.

The "don't care" terms can be used to advantage on the Karnaugh map. Figure 4–40 shows that for each "don't care" term, an X is placed in the cell. When grouping the 1s, the Xs can be treated as 1s to make a larger grouping or as 0s if they cannot be used to advantage. The larger a group, the simpler the resulting term will be.

Inputs				Output
A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Don't cares

(a) Truth table



(b) Without "don't cares" $Y = A\overline{B}\,\overline{C} + \overline{A}BCD$
With "don't cares" $Y = A + BCD$

FIGURE 4–40 Example of the use of "don't care" conditions to simplify an expression.

The truth table in Figure 4–40(a) describes a logic function that has a 1 output only when the BCD code for 7, 8, or 9 is present on the inputs. If the "don't cares" are used as 1s, the resulting expression for the function is $A + BCD$, as indicated in part (b). If the "don't cares" are not used as 1s, the resulting expression is $A\overline{B}\,\overline{C} + \overline{A}BCD$; so you can see the advantage of using "don't care" terms to get the simplest expression.

EXAMPLE 4–32

In a 7-segment display, each of the seven segments is activated for various digits. For example, segment *a* is activated for the digits 0, 2, 3, 5, 6, 7, 8, and 9, as illustrated in Figure 4–41. Since each digit can be represented by a BCD code, derive an SOP expression for segment *a* using the variables *ABCD* and then minimize the expression using a Karnaugh map.

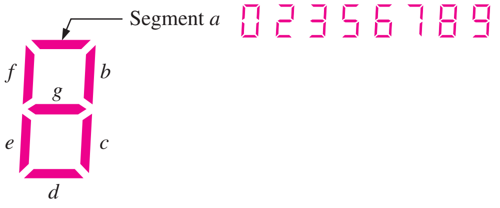


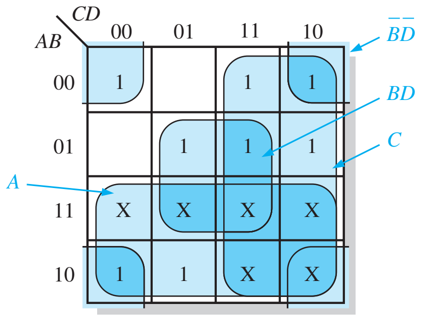
FIGURE 4–41 7-segment display.

Solution

The expression for segment *a* is

$$a = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D$$

Each term in the expression represents one of the digits in which segment *a* is used. The Karnaugh map minimization is shown in Figure 4–42. X's (don't cares) are entered for those states that do not occur in the BCD code.



AB \ CD	00	01	11	10
00	1		1	1
01		1	1	1
11	X	X	X	X
10	1	1	X	X

FIGURE 4–42

From the Karnaugh map, the minimized expression for segment *a* is

$$a = A + C + BD + \overline{B}\,\overline{D}$$

Related Problem

Draw the logic diagram for the segment-*a* logic.

SECTION 4–9 CHECKUP

1. Lay out Karnaugh maps for three and four variables.
2. Group the 1s and write the simplified SOP expression for the Karnaugh map in Figure 4–29.
3. Write the original standard SOP expressions for each of the Karnaugh maps in Figure 4–36.

4–10 Karnaugh Map POS Minimization

In the last section, you studied the minimization of an SOP expression using a Karnaugh map. In this section, we focus on POS expressions. The approaches are much the same except that with POS expressions, 0s representing the standard sum terms are placed on the Karnaugh map instead of 1s.

After completing this section, you should be able to

- Map a standard POS expression on a Karnaugh map
- Combine the 0s on the map into maximum groups
- Determine the minimum sum term for each group on the map
- Combine the minimum sum terms to form a minimum POS expression
- Use the Karnaugh map to convert between POS and SOP

Mapping a Standard POS Expression

For a POS expression in standard form, a 0 is placed on the Karnaugh map for each sum term in the expression. Each 0 is placed in a cell corresponding to the value of a sum term. For example, for the sum term $A + \overline{B} + C$, a 0 goes in the 010 cell on a 3-variable map.

When a POS expression is completely mapped, there will be a number of 0s on the Karnaugh map equal to the number of sum terms in the standard POS expression. The cells that do not have a 0 are the cells for which the expression is 1. Usually, when working with POS expressions, the 1s are left off. The following steps and the illustration in Figure 4–43 show the mapping process.

Step 1: Determine the binary value of each sum term in the standard POS expression. This is the binary value that makes the term equal to 0.

Step 2: As each sum term is evaluated, place a 0 on the Karnaugh map in the corresponding cell.

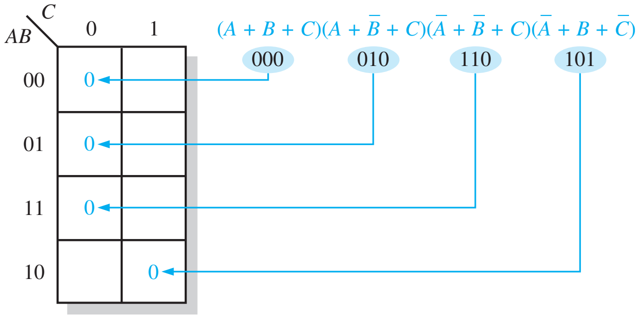

C
AB
0
1
$(A + B + C)(A + \overline{B} + C)(\overline{A} + \overline{B} + C)(\overline{A} + B + \overline{C})$
000
010
110
101
00
01
11
10
0
0
0
0

FIGURE 4–43 Example of mapping a standard POS expression.

EXAMPLE 4–33

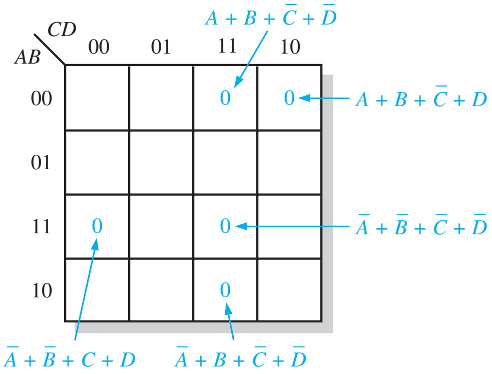
Map the following standard POS expression on a Karnaugh map:

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

Solution

Evaluate the expression as shown below and place a 0 on the 4-variable Karnaugh map in Figure 4–44 for each standard sum term in the expression.

$$\underset{1100}{(\overline{A} + \overline{B} + C + D)}\underset{1011}{(\overline{A} + B + \overline{C} + \overline{D})}\underset{0010}{(A + B + \overline{C} + D)}\underset{1111}{(\overline{A} + \overline{B} + \overline{C} + \overline{D})}\underset{0011}{(A + B + \overline{C} + \overline{D})}$$



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

FIGURE 4–44

Related Problem

Map the following standard POS expression on a Karnaugh map:

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

Karnaugh Map Simplification of POS Expressions

The process for minimizing a POS expression is basically the same as for an SOP expression except that you group 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms. The rules for grouping the 0s are the same as those for grouping the 1s that you learned in Section 4–9.

EXAMPLE 4–34

Use a Karnaugh map to minimize the following standard POS expression:

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

Also, derive the equivalent SOP expression.

Solution

The combinations of binary values of the expression are

$$(0 + 0 + 0)(0 + 0 + 1)(0 + 1 + 0)(0 + 1 + 1)(1 + 1 + 0)$$

Map the standard POS expression and group the cells as shown in Figure 4–45.

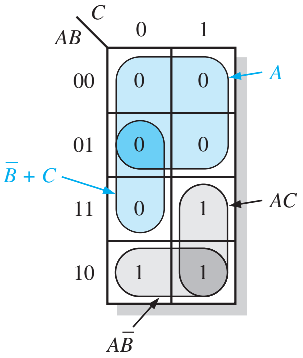


FIGURE 4–45

Notice how the 0 in the 110 cell is included into a 2-cell group by utilizing the 0 in the 4-cell group. The sum term for each blue group is shown in the figure and the resulting minimum POS expression is

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

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.

Grouping the 1s as shown by the gray areas yields an SOP expression that is equivalent to grouping the 0s.

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

Related Problem

Use a Karnaugh map to simplify the following standard POS expression:

$$(X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(\overline{X} + \overline{Y} + Z)(\overline{X} + Y + Z)$$

EXAMPLE 4–35

Use a Karnaugh map to minimize the following POS expression:

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

Solution

The first term must be expanded into $\overline{A} + B + C + D$ and $A + B + C + D$ to get a standard POS expression, which is then mapped; and the cells are grouped as shown in Figure 4–46. The sum term for each group is shown and the resulting minimum POS expression is

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

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.

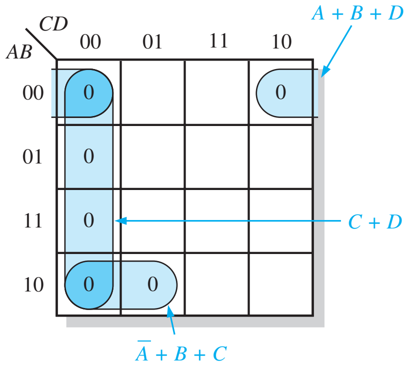


FIGURE 4–46

Related Problem

Use a Karnaugh map to simplify the following POS expression:

$$(W + \overline{X} + Y + \overline{Z})(W + X + Y + Z)(W + \overline{X} + \overline{Y} + Z)(\overline{W} + \overline{X} + Z)$$

Converting Between POS and SOP Using the Karnaugh Map

When a POS expression is mapped, it can easily be converted to the equivalent SOP form directly from the Karnaugh map. Also, given a mapped SOP expression, an equivalent POS expression can be derived directly from the map. This provides a good way to compare both minimum forms of an expression to determine if one of them can be implemented with fewer gates than the other.

For a POS expression, all the cells that do not contain 0s contain 1s, from which the SOP expression is derived. Likewise, for an SOP expression, all the cells that do not contain 1s contain 0s, from which the POS expression is derived. Example 4–36 illustrates this conversion.

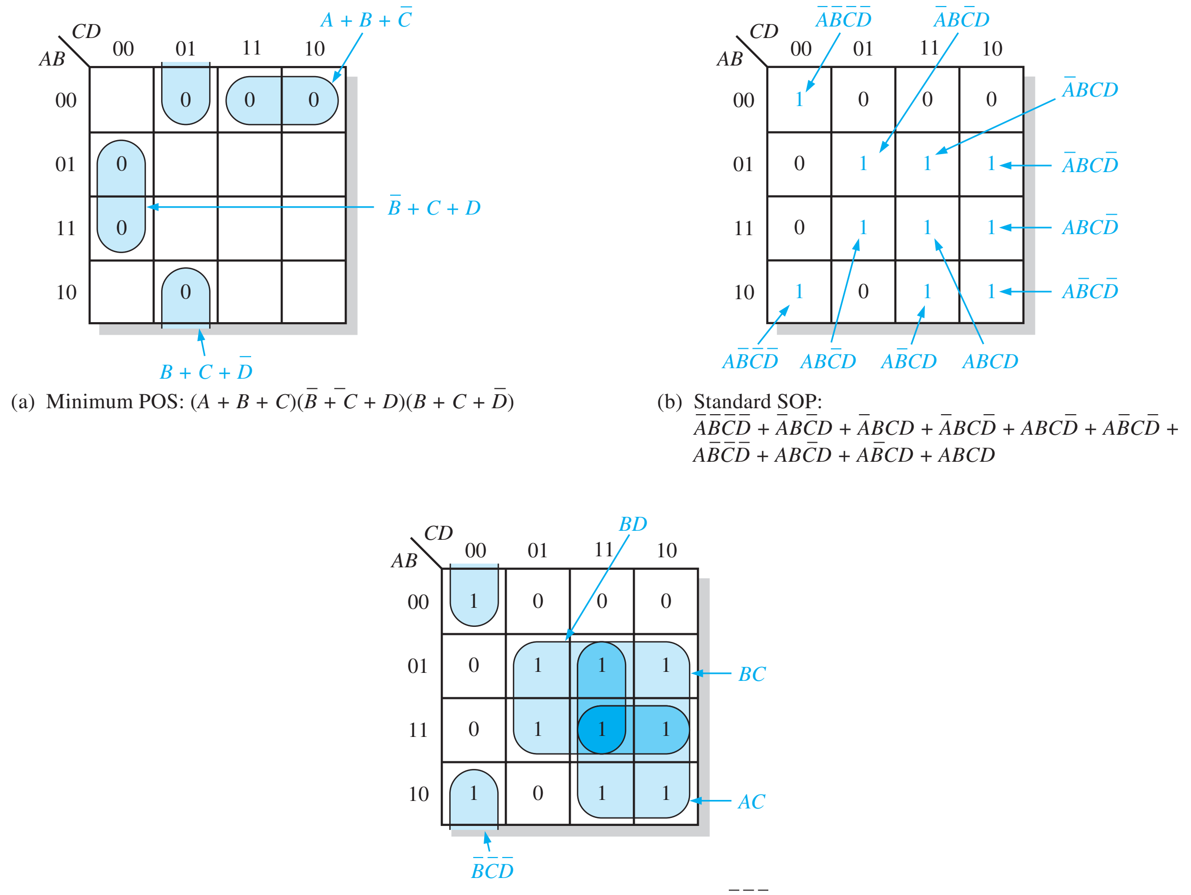
EXAMPLE 4–36

Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.

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

Solution

The 0s for the standard POS expression are mapped and grouped to obtain the minimum POS expression in Figure 4–47(a). In Figure 4–47(b), 1s are added to the cells that do not contain 0s. From each cell containing a 1, a standard product term is obtained as indicated. These product terms form the standard SOP expression. In Figure 4–47(c), the 1s are grouped and a minimum SOP expression is obtained.



CD
AB
$A+B+\overline{C}$
$\overline{B}+C+D$
$B+C+\overline{D}$

	00	01	11	10
00		0	0	0
01	0			
11	0			
10		0		

(a) Minimum POS: $(A+B+C)(\overline{B}+\overline{C}+D)(B+C+\overline{D})$

CD
AB
$\overline{A}\overline{B}\overline{C}\overline{D}$
$\overline{A}B\overline{C}D$
$\overline{A}BCD$
$\overline{A}BC\overline{D}$
$ABC\overline{D}$
$A\overline{B}C\overline{D}$
$A\overline{B}\overline{C}\overline{D}$
$AB\overline{C}D$
$A\overline{B}CD$
$ABCD$

	00	01	11	10
00	1	0	0	0
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

(b) Standard SOP:
$\overline{A}\overline{B}\overline{C}\overline{D}+\overline{A}B\overline{C}D+\overline{A}BCD+\overline{A}BC\overline{D}+ABC\overline{D}+A\overline{B}C\overline{D}+$
$A\overline{B}\overline{C}\overline{D}+AB\overline{C}D+A\overline{B}CD+ABCD$

CD
AB
BD
BC
AC
$\overline{B}\overline{C}\overline{D}$

	00	01	11	10
00	1	0	0	0
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

(c) Minimum SOP: $AC+BC+BD+\overline{B}\overline{C}\overline{D}$

FIGURE 4–47

Related Problem

Use a Karnaugh map to convert the following expression to minimum SOP form:

$$(W+\overline{X}+Y+\overline{Z})(\overline{W}+X+\overline{Y}+\overline{Z})(\overline{W}+\overline{X}+\overline{Y}+Z)(\overline{W}+\overline{X}+\overline{Z})$$

SECTION 4–10 CHECKUP

1. What is the difference in mapping a POS expression and an SOP expression?
2. What is the standard sum term for a 0 in cell 1011?
3. What is the standard product term for a 1 in cell 0010?

4–11 The Quine-McCluskey Method

For Boolean functions up to four variables, the Karnaugh map method is a powerful minimization method. When there are five variables, the Karnaugh map method is difficult to apply and completely impractical beyond five. The Quine-McCluskey method is a formal tabular method for applying the Boolean distributive law to various terms to find the minimum sum of products by eliminating literals that appear in two terms as complements. (For example, $ABCD + ABC\overline{D} = ABC$). *A Quine-McCluskey method tutorial is available on the website.*

After completing this section, you should be able to

- Describe the Quine-McCluskey method
- Reduce a Boolean expression using the Quine-McCluskey method

Unlike the Karnaugh mapping method, Quine-McCluskey lends itself to the computerized reduction of Boolean expressions, which is its principal use. For simple expressions, with up to four or perhaps even five variables, the Karnaugh map is easier for most people because it is a graphic method.

To apply the Quine-McCluskey method, first write the function in standard **minterm** (SOP) form. To illustrate, we will use the expression

$$X = \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD$$

and represent it as binary numbers on the truth table shown in Table 4–9. The minterms that appear in the function are listed in the right column.

TABLE 4–9

ABCD	*X*	**Minterm**
0000	0	
0001	1	m_1
0010	0	
0011	1	m_3
0100	1	m_4
0101	1	m_5
0110	0	
0111	0	
1000	0	
1001	0	
1010	1	m_{10}
1011	0	
1100	1	m_{12}
1101	1	m_{13}
1110	0	
1111	1	m_{15}

The second step in applying the Quine-McCluskey method is to arrange the minterms in the original expression in groups according to the number of 1s in each minterm, as shown in Table 4–10. In this example, there are four groups of minterms. (Note that if m_0 had been in the original expression, there would be five groups.)

TABLE 4–10

Number of 1s	Minterm	*ABCD*
1	m_1	0001
	m_4	0100
2	m_3	0011
	m_5	0101
	m_{10}	1010
	m_{12}	1100
3	m_{13}	1101
4	m_{15}	1111

Third, compare adjacent groups, looking to see if any minterms are the same in every position *except one.* If they are, place a check mark by those two minterms, as shown in Table 4–11. You should check each minterm against all others in the following group, but it is not necessary to check any groups that are not adjacent. In the column labeled *First Level,* you will have a list of the minterm names and the binary equivalent with an x as the placeholder for the literal that differs. In the example, minterm m_1 in Group 1 (0001) is identical to m_3 in Group 2 (0011) except for the *C* position, so place a check mark by these two minterms and enter 00x1 in the column labeled *First Level*. Minterm m_4 (0100) is identical to m_5 (0101) except for the *D* position, so check these two minterms and enter 010x in the last column. If a given term can be used more than once, it should be. In this case, notice that m_1 can be used again with m_5 in the second row with the x now placed in the *B* position.

TABLE 4–11

Number of 1s in Minterm	Minterm	ABCD	First Level
1	m_1	0001 ✓	(m_1, m_3) 00x1
	m_4	0100 ✓	(m_1, m_5) 0x01
2	m_3	0011 ✓	(m_4, m_5) 010x
	m_5	0101 ✓	(m_4, m_{12}) x100
	m_{10}	1010	(m_5, m_{13}) x101
	m_{12}	1100 ✓	(m_{12}, m_{13}) 110x
3	m_{13}	1101 ✓	(m_{13}, m_{15}) 11x1
4	m_{15}	1111 ✓	

In Table 4–11, minterm m_4 and minterm m_{12} are identical except for the *A* position. Both minterms are checked and x100 is entered in the *First Level* column . Follow this procedure for groups 2 and 3. In these groups, m_5 and m_{13} are combined and so are m_{12} and m_{13} (notice that m_{12} was previously used with m_4 and is used again). For groups 3 and 4, both m_{13} and m_{15} are added to the list in the *First Level* column .

In this example, minterm m_{10} does not have a check mark because no other minterm meets the requirement of being identical except for one position. This term is called an *essential prime implicant,* and it must be included in our final reduced expression.

The terms listed in the *First Level* have been used to form a reduced table (Table 4–12) with one less group than before. The number of 1s remaining in the *First Level* are counted and used to form three new groups.

Terms in the new groups are compared against terms in the adjacent group down. You need to compare these terms only if the x is in the same relative position in adjacent groups; otherwise go on. If the two expressions differ by exactly one position, a check mark is

TABLE 4–12

First Level	Number of 1s in First Level	Second Level
(m_1, m_3) 00x1	1	$(m_4, m_5, m_{12}, m_{13})$ x10x
(m_1, m_5) 0x01		$(m_4, m_5, m_{12}, m_{13})$ x10x
(m_4, m_5) 010x ✓		
(m_4, m_{12}) x100 ✓		
(m_5, m_{13}) x101 ✓	2	
(m_{12}, m_{13}) 110x ✓		
(m_{13}, m_{15}) 11x1	3	

placed next to both terms as before and all of the minterms are listed in the Second Level list. As before, the one position that has changed is entered as an x in the *Second Level.*

For our example, notice that the third term in Group 1 and the second term in Group 2 meet this requirement, differing only with the *A* literal. The fourth term in Group 1 also can be combined with the first term in Group 2, forming a redundant set of minterms. One of these can be crossed off the list and will not be used in the final expression.

With complicated expressions, the process described can be continued. For our example, we can read the *Second Level* expression as $B\overline{C}$. The terms that are unchecked will form other terms in the final reduced expression. The first unchecked term is read as $\overline{A}\,\overline{B}D$. The next one is read as $\overline{A}\,\overline{C}D$. The last unchecked term is ABD. Recall that m_{10} was an essential prime implicant, so is picked up in the final expression. The reduced expression using the unchecked terms is:

$$X = B\overline{C} + \overline{A}\,\overline{B}D + \overline{A}\,\overline{C}D + ABD + A\overline{B}C\overline{D}$$

Although this expression is correct, it may not be the minimum possible expression. There is a final check that can eliminate any unnecessary terms. The terms for the expression are written into a prime implicant table, with minterms for each prime implicant checked, as shown in Table 4–13.

TABLE 4–13

	Minterms							
Prime Implicants	m_1	m_3	m_4	m_5	m_{10}	m_{12}	m_{13}	m_{15}
$B\overline{C}$ $(m_4, m_5, m_{12}, m_{13})$			✓	✓		✓	✓	
$\overline{A}\,\overline{B}\,D$ (m_1, m_3)	✓	✓						
$\overline{A}\,\overline{C}D$ (m_1, m_5)	✓			✓				
ABD (m_{13}, m_{15})							✓	✓
$A\overline{B}C\overline{D}$ (m_{10})					✓			

If a minterm has a single check mark, then the prime implicant is essential and must be included in the final expression. The term ABD must be included because m_{15} is only covered by it. Likewise m_{10} is only covered by $A\overline{B}C\overline{D}$, so it must be in the final expression. Notice that the two minterms in $\overline{A}\,\overline{C}D$ are covered by the prime implicants in the first two rows, so this term is unnecessary. The final reduced expression is, therefore,

$$X = B\overline{C} + \overline{A}\,\overline{B}D + ABD + A\overline{B}C\overline{D}$$

SECTION 4–11 CHECKUP

1. What is a minterm?
2. What is an essential prime implicant?

4–12 Boolean Expressions with VHDL

The ability to create simple and compact code is important in a VHDL program. By simplifying a Boolean expression for a given logic function, it is easier to write and debug the VHDL code; in addition, the result is a clearer and more concise program. Many VHDL development software packages contain tools that automatically optimize a program when it is compiled and converted to a downloadable file. However, this does not relieve you from creating program code that is clear and concise. You should not only be concerned with the number of lines of code, but you should also be concerned with the complexity of each line of code. In this section, you will see the difference in VHDL code when simplification methods are applied. Also, three levels of abstraction used in the description of a logic function are examined. *A VHDL tutorial is available on the website.*

After completing this section, you should be able to

- Write VHDL code to represent a simplified logic expression and compare it to the code for the original expression
- Relate the advantages of optimized Boolean expressions as applied to a target device
- Understand how a logic function can be described at three levels of abstraction
- Relate VHDL approaches to the description of a logic function to the three levels of abstraction

Boolean Algebra in VHDL Programming

The basic rules of Boolean algebra that you have learned in this chapter should be applied to any applicable VHDL code. Eliminating unnecessary gate logic allows you to create compact code that is easier to understand, especially when someone has to go back later and update or modify the program.

In Example 4–37, DeMorgan's theorems are used to simplify a Boolean expression, and VHDL programs for both the original expression and the simplified expression are compared.

EXAMPLE 4–37

First, write a VHDL program for the logic described by the following Boolean expression. Next, apply DeMorgan's theorems and Boolean rules to simplify the expression. Then write a program to reflect the simplified expression.

$$X = (\overline{AC + \overline{B\overline{C}} + D}) + \overline{\overline{BC}}$$

Solution

The VHDL program for the logic represented by the original expression is

```
entity OriginalLogic is
  port (A, B, C, D: in bit; X: out bit);
end entity OriginalLogic;
architecture Expression1 of OriginalLogic is
begin
  X <= not((A and C) or not(B and not C) or D)  or not(not(B and C));
end architecture Expression1;
```

Four inputs and one output are described.

The original logic contains four inputs, 3 AND gates, 2 OR gates, and 3 inverters.

By selectively applying DeMorgan's theorem and the laws of Boolean algebra, you can reduce the Boolean expression to its simplest form.

$$
\begin{aligned}
\overline{(AC + \overline{B\overline{C}} + D)} + \overline{\overline{BC}} &= (\overline{AC})(\overline{\overline{B\overline{C}}})\overline{D} + \overline{\overline{BC}} && \text{Apply DeMorgan} \\
&= (\overline{AC})(B\overline{C})\overline{D} + BC && \text{Cancel double complements} \\
&= (\overline{A} + \overline{C})B\overline{C}\,\overline{D} + BC && \text{Apply DeMorgan and factor} \\
&= \overline{A}B\overline{C}\,\overline{D} + B\overline{C}\,\overline{D} + BC && \text{Distributive law} \\
&= B\overline{C}\,\overline{D}(1 + \overline{A}) + BC && \text{Factor} \\
&= B\overline{C}\,\overline{D} + BC && \text{Rule: } 1 + A = 1
\end{aligned}
$$

The VHDL program for the logic represented by the reduced expression is

```
entity ReducedLogic is
   port (B, C, D: in bit; X: out bit);
end entity ReducedLogic;
architecture Expression2 of ReducedLogic is
begin
   X <= (B and not C and not D) or ( B and C);
end architecture Expression2;
```

3 inputs and 1 output are described.

The simplified logic contains three inputs, 3 AND gates, 1 OR gate, and 2 inverters.

As you can see, Boolean simplification is applicable to even simple VHDL programs.

Related Problem

Write the VHDL architecture statement for the expression $X = (\overline{A} + B + C)D$ as stated. Apply any applicable Boolean rules and rewrite the VHDL statement.

Example 4–38 demonstrates a more significant reduction in VHDL code complexity, using a Karnaugh map to reduce an expression.

EXAMPLE 4–38

(a) Write a VHDL program to describe the following SOP expression.

(b) Minimize the expression and show how much the VHDL program is simplified.

$$
\begin{aligned}
X = {}& \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}\,\overline{D} + \overline{A}BC\overline{D} + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} \\
& + A\overline{B}C\overline{D} + ABC\overline{D} + AB\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D
\end{aligned}
$$

Solution

(a) The VHDL program for the SOP expression without minimization is large and hard to follow as you can see in the following VHDL code. Code such as this is subject to error. The VHDL program for the original SOP expression is as follows:

```
entity OriginalSOP is
   port (A, B, C, D: in bit; X: out bit);
end entity OriginalSOP;
architecture Equation1 of OriginalSOP is
begin
   X <= (not A and not B and not C and not D) or
             (not A and not B and not C and D) or
             (not A and B and not C and not D) or
             (not A and B and C and not D) or
             (not A and not B and C and not D) or
             (A and not B and not C and not D) or
             (A and not B and C and not D) or
             (A and B and C and not D) or
             (A and B and not C and not D) or
```

```
            (A and not B and not C and D) or
            (not A and B and not C and D) or
            (A and B and not C and D);
    end architecture Equation1;
```

(b) Now, use a four-variable Karnaugh map to reduce the original SOP expression to a minimum form. The original SOP expression is mapped in Figure 4–48.

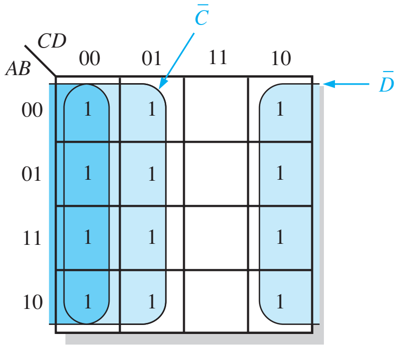


FIGURE 4–48

The original SOP Boolean expression that is plotted on the Karnaugh map in Figure 4–48 contains twelve 4-variable terms as indicated by the twelve 1s on the map. Recall that only the variables that do not change within a group remain in the expression for that group. The simplified expression taken from the map is developed next.

Combining the terms from the Karnaugh map, you get the following simplified expression, which is equivalent to the original SOP expression.

$$X = \overline{C} + \overline{D}$$

Using the simplified expression, the VHDL code can be rewitten with fewer terms, making the code more readable and easier to modify. Also, the logic implemented in a target device by the reduced code consumes much less space in the PLD. The VHDL program for the simplified SOP expression is as follows:

```
entity SimplifiedSOP is
   port (A, B, C, D: in bit; X: out bit);
end entity SimplifiedSOP;
architecture Equation2 of SimplifiedSOP is
begin
   X <= not C or not D
end architecture Equation2;
```

Related Problem

Write a VHDL architecture statement to describe the logic for the expression

$$X = A(BC + \overline{D})$$

As you have seen, the simplification of Boolean logic is important in the design of any logic function described in VHDL. Target devices have finite capacity and therefore require the creation of compact and efficient program code. Throughout this chapter, you have learned that the simplification of complex Boolean logic can lead to the elimination of unnecessary logic as well as the simplification of VHDL code.

Levels of Abstraction

A given logic function can be described at three different levels. It can be described by a truth table or a state diagram, by a Boolean expression, or by its logic diagram (schematic).

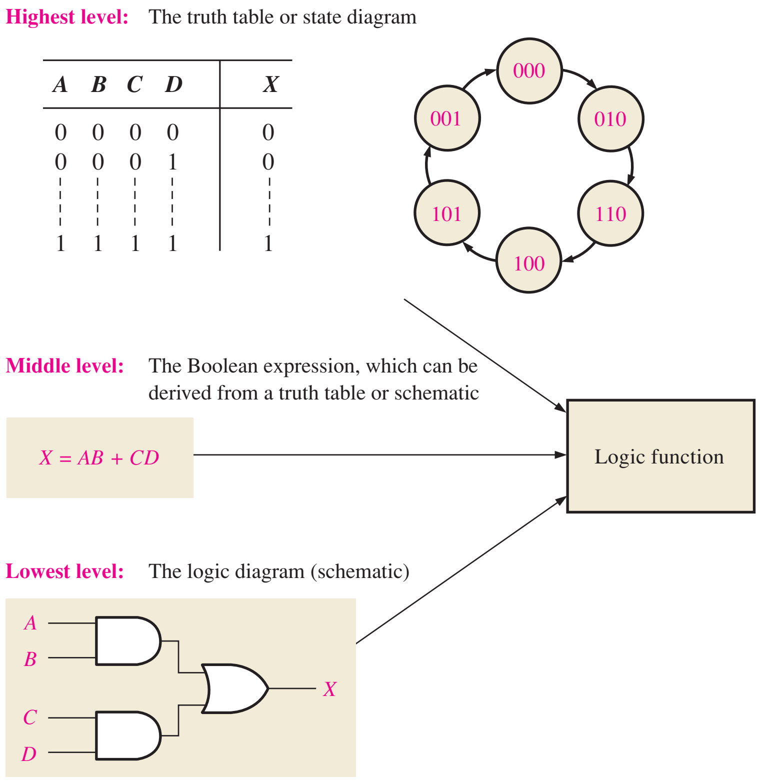

Highest level: The truth table or state diagram
A B C D X
0 0 0 0 0
0 0 0 1 0
1 1 1 1 1
000
010
110
100
101
001
Middle level: The Boolean expression, which can be derived from a truth table or schematic
X = AB + CD
Logic function
Lowest level: The logic diagram (schematic)
A
B
C
D
X

FIGURE 4–49 Illustration of the three levels of abstraction for describing a logic function.

The truth table and state diagram are the most abstract ways to describe a logic function. A Boolean expression is the next level of abstraction, and a schematic is the lowest level of abstraction. This concept is illustrated in Figure 4–49 for a simple logic circuit. VHDL provides three approaches for describing functions that correspond to the three levels of abstraction.

- The data flow approach is analogous to describing a logic function with a Boolean expression. The data flow approach specifies each of the logic gates and how the data flows through them. This approach was applied in Examples 4–37 and 4–38.
- The structural approach is analogous to using a logic diagram or schematic to describe a logic function. It specifies the gates and how they are connected, rather than how signals (data) flow through them. The structural approach is used to develop VHDL code for describing logic circuits in Chapter 5.
- The behavioral approach is analogous to describing a logic function using a state diagram or truth table. However, this approach is the most complex; it is usually restricted to logic functions whose operations are time dependent and normally require some type of memory.

SECTION 4–12 CHECKUP

1. What are the advantages of Boolean logic simplification in terms of writing a VHDL program?
2. How does Boolean logic simplification benefit a VHDL program in terms of the target device?
3. Name the three levels of abstraction for a combinational logic function and state the corresponding VHDL approaches for describing a logic function.

Applied Logic

Seven-Segment Display

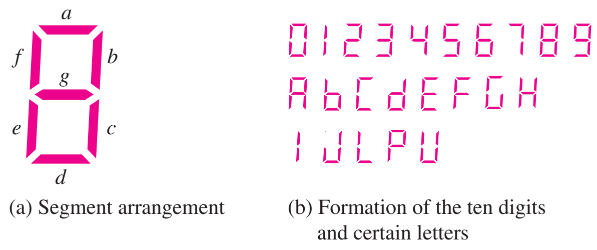
Seven-segment displays are used in many types of products that you see every day. A 7-segment display was used in the tablet-bottling system that was introduced in Chapter 1. The display in the bottling system is driven by logic circuits that decode a binary coded decimal (BCD) number and activate the appropriate digits on the display. BCD-to-7-segment decoder/drivers are readily available as single IC packages for activating the ten decimal digits.

In addition to the numbers from 0 to 9, the 7-segment display can show certain letters. For the tablet-bottling system, a requirement has been added to display the letters A, b, C, d, and E on a separate common-anode 7-segment display that uses a hexadecimal keypad for both the numerical inputs and the letters. These letters will be used to identify the type of vitamin tablet that is being bottled at any given time. In this application, the decoding logic for displaying the five letters is developed.

The 7-Segment Display

Two types of 7-segment displays are the LED and the LCD. Each of the seven segments in an LED display uses a light-emitting diode to produce a colored light when there is current through it and can be seen in the dark. An LCD or liquid-crystal display operates by polarizing light so that when a segment is not activated by a voltage, it reflects incident light and appears invisible against its background; however, when a segment is activated, it does not reflect light and appears black. LCD displays cannot be seen in the dark.

The seven segments in both LED and LCD displays are arranged as shown in Figure 4–50 and labeled *a*, *b*, *c*, *d*, *e*, *f*, and *g* as indicated in part (a). Selected segments are activated to create each of the ten decimal digits as well as certain letters of the alphabet, as shown in part (b). The letter *b* is shown as lowercase because a capital B would be the same as the digit 8. Similarly, for *d*, a capital letter would appear as a 0.



(a) Segment arrangement

(b) Formation of the ten digits and certain letters

FIGURE 4–50 Seven-segment display.

Exercise

1. List the segments used to form the digit 2.
2. List the segments used to form the digit 5.
3. List the segments used to form the letter A.
4. List the segments used to form the letter E.
5. Is there any one segment that is common to all digits?
6. Is there any one segment that is common to all letters?

Display Logic

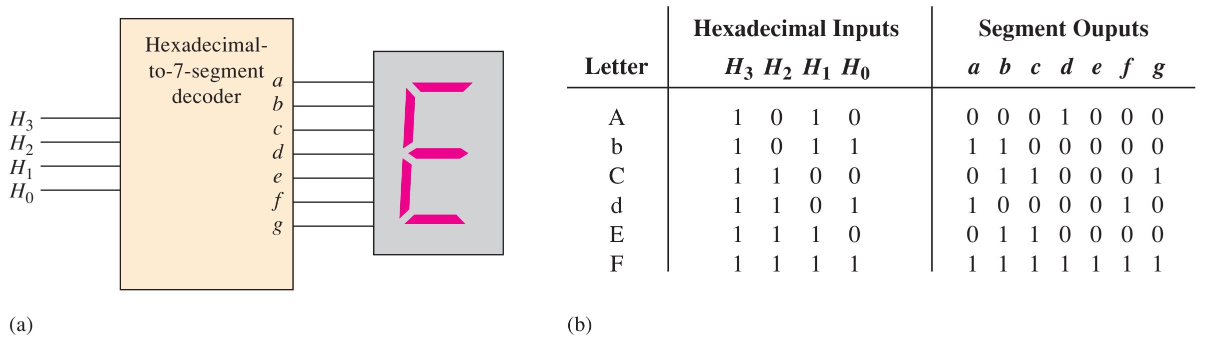
The segments in a 7-segment display can be used in the formation of various letters as shown in Figure 4–50(b). Each segment must be activated by its own decoding circuit that detects the code for any of the letters in which that segment is used. Because a common-anode display is used, the segments are turned *on* with a LOW (0) logic level and turned *off* with a HIGH (1) logic level. The active segments are shown for each of the letters required for the tablet-bottling system in Table 4–14. Even though the active level is LOW (lighting the LED), the logic expressions are developed exactly the same way as discussed in this chapter, by mapping the desired output (1, 0, or X) for every possible input, grouping the 1s on the map, and reading the SOP expression from the map. In effect, the reduced logic expression is the logic for keeping a given segment OFF. At first, this may sound confusing, but it is simple in practice and it avoids an output current capability issue with bipolar (TTL) logic (discussed in Chapter 15 on the website).

TABLE 4–14

Active segments for each of the five letters used in the system display.

Letter	Segments Activated
A	*a, b, c, e, f, g*
b	*c, d, e, f, g*
C	*a, d, e, f*
d	*b, c, d, e, g*
E	*a, d, e, f, g*

A block diagram of a 7-segment logic and display for generating the five letters is shown in Figure 4–51(a), and the truth table is shown in part (b). The logic has four hexadecimal inputs and seven outputs, one for each segment. Because the letter F is not used as an input, we will show it on the truth table with all outputs set to 1 (OFF).



Letter	Hexadecimal Inputs				Segment Ouputs						
	H_3	H_2	H_1	H_0	*a*	*b*	*c*	*d*	*e*	*f*	*g*
A	1	0	1	0	0	0	0	1	0	0	0
b	1	0	1	1	1	1	0	0	0	0	0
C	1	1	0	0	0	1	1	0	0	0	1
d	1	1	0	1	1	0	0	0	0	1	0
E	1	1	1	0	0	1	1	0	0	0	0
F	1	1	1	1	1	1	1	1	1	1	1

FIGURE 4–51 Hexadecimal-to-7-segment decoder for letters *A* through *E*, used in the system.

Karnaugh Maps and the Invalid BCD Code Detector

To develop the simplified logic for each segment, the truth table information in Figure 4–51 is mapped onto Karnaugh maps. Recall that the BCD numbers will not be shown on the letter display. For this reason, an entry that represents a BCD number will be entered as an "X" ("don't care") on the K-maps. This makes the logic much simpler but would put some strange outputs on the display unless steps are taken to eliminate that possibility. Because all of the letters are *invalid* BCD characters, the display is activated only when an invalid BCD code is entered into the keypad, thus allowing only letters to be displayed.

Expressions for the Segment Logic

Using the table in 4–51(b), a standard SOP expression can be written for each segment and then minimized using a K-map. The desired outputs from the truth table are entered in the appropriate cells representing the hex inputs. To obtain the minimum SOP expressions for the display logic, the 1s and Xs are grouped.

Segment a Segment *a* is used for the letters A, C, and E. For the letter A, the hexadecimal code is 1010 or, in terms of variables, $H_3\overline{H}_2H_1\overline{H}_0$. For the letter C, the hexadecimal code is 1100 or $H_3H_2\overline{H}_1\overline{H}_0$. For the letter E, the code is 1110 or $H_3H_2H_1\overline{H}_0$. The complete standard SOP expression for segment *a* is

$$a = H_3\overline{H}_2H_1\overline{H}_0 + H_3H_2\overline{H}_1\overline{H}_0 + H_3H_2H_1\overline{H}_0$$

Because a LOW is the active output state for each segment logic circuit, a 0 is entered on the Karnaugh map in each cell that represents the code for the letters in which the segment is *on*. The simplification of the expression for segment *a* is shown in Figure 4–52(a) after grouping the 1s and Xs.

Segment b Segment *b* is used for the letters A and d. The complete standard SOP expression for segment *b* is

$$b = H_3\overline{H}_2H_1\overline{H}_0 + H_3H_2\overline{H}_1H_0$$

The simplification of the expression for segment *b* is shown in Figure 4–52(b).

Segment c Segment *c* is used for the letters A, b, and d. The complete standard SOP expression for segment *c* is

$$c = H_3\overline{H}_2H_1\overline{H}_0 + H_3\overline{H}_2H_1H_0 + H_3H_2\overline{H}_1H_0$$

The simplification of the expression for segment *c* is shown in Figure 4–52(c).

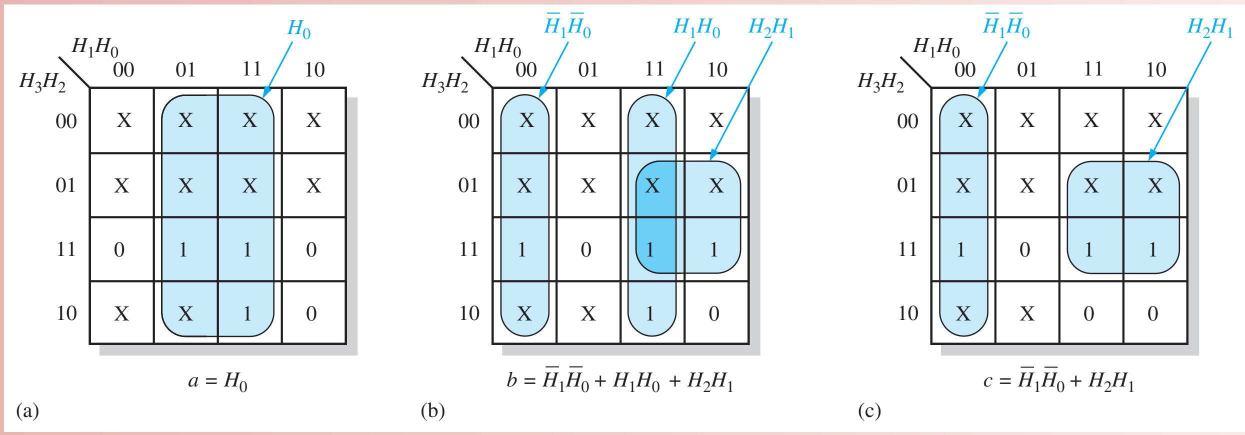


FIGURE 4–52 Minimization of the expressions for segments *a*, *b*, and *c*.

Exercise

7. Develop the minimum expression for segment *d*.
8. Develop the minimum expression for segment *e*.
9. Develop the minimum expression for segment *f*.
10. Develop the minimum expression for segment *g*.

The Logic Circuits

From the minimum expressions, the logic circuits for each segment can be implemented. For segment *a*, connect the H_0 input directly (no gate) to the *a* segment on the display. The segment *b* and segment *c* logic are shown in Figure 4–53 using AND or OR gates. Notice that two of the terms (H_2H_1 and $\overline{H}_1\overline{H}_0$) appear in the expressions for both *b* and *c* logic so two of the AND gates can be used in both, as indicated.

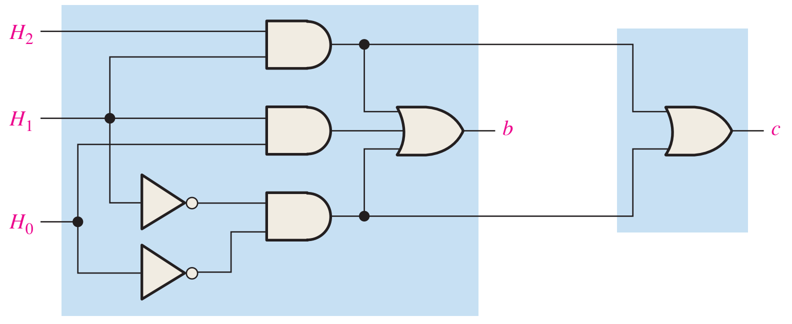


FIGURE 4–53 Segment-*b* and segment-*c* logic circuits.

Exercise

11. Show the logic for segment *d*.
12. Show the logic for segment *e*.
13. Show the logic for segment *f*.
14. Show the logic for segment *g*.

Describing the Decoding Logic with VHDL

The 7-segment decoding logic can be described using VHDL for implementation in a programmable logic device (PLD). The logic expressions for segments *a*, *b*, and *c* of the display are as follows:

$$a = H_0$$
$$b = \overline{H}_1\overline{H}_0 + H_1H_0 + H_2H_1$$
$$c = \overline{H}_1\overline{H}_0 + H_2H_1$$

- The VHDL code for segment *a* is

```
entity SEGLOGIC is
   port (H0: in bit; SEGa: out bit);
end entity SEGLOGIC;
architecture LogicFunction of SEGLOGIC is
begin
   SEGa <= H0;
end architecture LogicFunction;
```

- The VHDL code for segment *b* is

```
entity SEGLOGIC is
   port (H0, H1, H2: in bit; SEGb: out bit);
end entity SEGLOGIC;
architecture LogicFunction of SEGLOGIC is
begin
   SEGb <= (not H1 and not H0) or (H1 and H0) or (H2 and H1);
end architecture LogicFunction;
```

- The VHDL code for segment *c* is

```
entity SEGLOGIC is
   port (H0, H1, H2: in bit; SEGc: out bit);
end entity SEGLOGIC;
architecture LogicFunction of SEGLOGIC is
begin
   SEGc <= (not H1 and not H0) or (H2 and H1);
end architecture LogicFunction;
```

Exercise

15. Write the VHDL code for segments *d*, *e*, *f*, and *g*.

Simulation

The decoder simulation using Multisim is shown in Figure 4–54 with the letter E selected. Subcircuits are used for the segment logic to be developed as activities or in the lab. The purpose of simulation is to verify proper operation of the circuit.

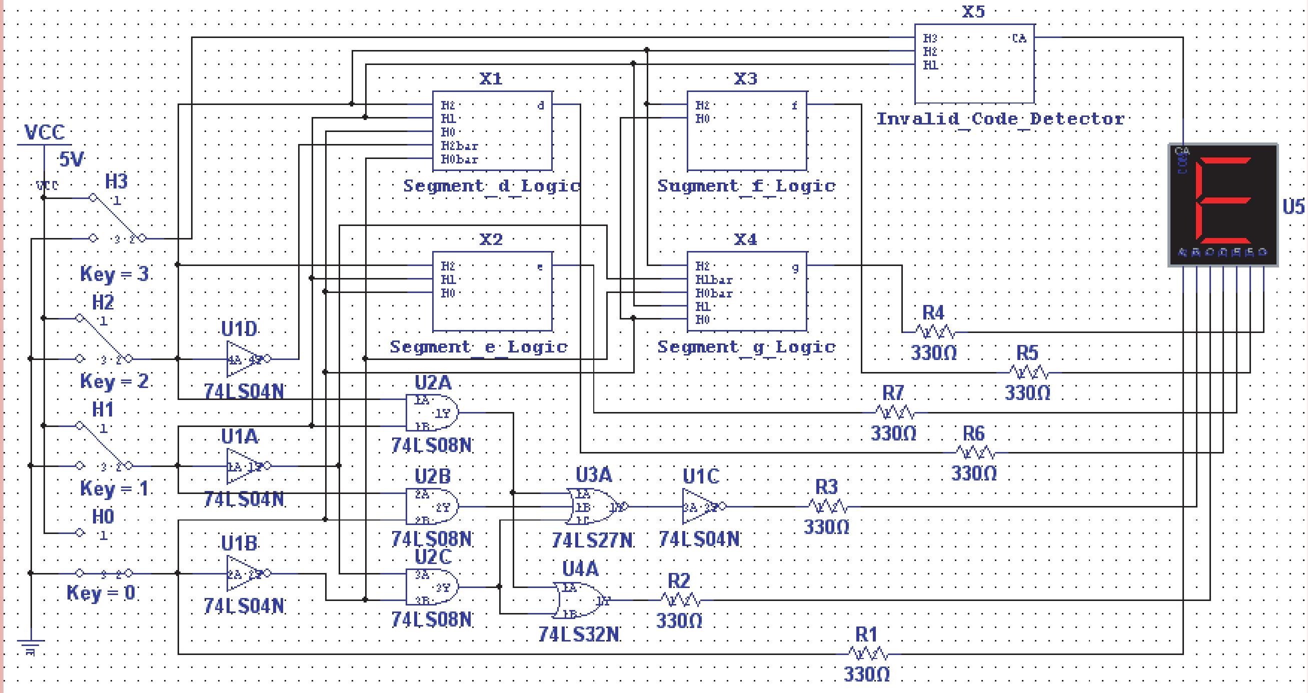


FIGURE 4–54 Multisim circuit screen for decoder and display.

Open file AL04 in the Applied Logic folder on the website. Run the simulation of the decoder and display using your Multisim software. Observe the operation for the specified letters.

Putting Your Knowledge to Work

How would you modify the decoder for a common-cathode 7-segment display?

SUMMARY

- Gate symbols and Boolean expressions for the outputs of an inverter and 2-input gates are shown in Figure 4–55.

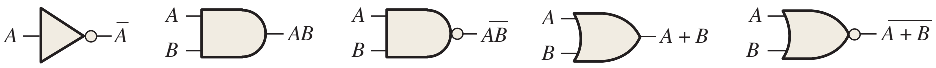


FIGURE 4–55

- Commutative laws: $A + B = B + A$
  $AB = BA$
- Associative laws: $A + (B + C) = (A + B) + C$
  $A(BC) = (AB)C$
- Distributive law: $A(B + C) = AB + AC$
- Boolean rules:
  1. $A + 0 = A$
  2. $A + 1 = 1$
  3. $A \cdot 0 = 0$
  4. $A \cdot 1 = A$
  5. $A + A = A$
  6. $A + \overline{A} = 1$
  7. $A \cdot A = A$
  8. $A \cdot \overline{A} = ($
  9. $\overline{\overline{A}} = A$
  10. $A + AB = A$
  11. $A + \overline{A}B = A + B$
  12. $(A + B)(A + C) = A + BC$
- DeMorgan's theorems:
  1. The complement of a product is equal to the sum of the complements of the terms in the product.

     $$\overline{XY} = \overline{X} + \overline{Y}$$

  2. The complement of a sum is equal to the product of the complements of the terms in the sum.

     $$\overline{X + Y} = \overline{X}\,\overline{Y}$$

- Karnaugh maps for 3 variables have 8 cells and for 4 variables have 16 cells.
- Quinn-McCluskey is a method for simplification of Boolean expressions.
- The three levels of abstraction in VHDL are data flow, structural, and behavioral.

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Complement The inverse or opposite of a number. In Boolean algebra, the inverse function, expressed with a bar over a variable. The complement of a 1 is 0, and vice versa.

"Don't care" A combination of input literals that cannot occur and can be used as a 1 or a 0 on a Karnaugh map for simplification.

Karnaugh map An arrangement of cells representing the combinations of literals in a Boolean expression and used for a systematic simplification of the expression.

Minimization The process that results in an SOP or POS Boolean expression that contains the fewest possible literals per term.

Product-of-sums (POS) A form of Boolean expression that is basically the ANDing of ORed terms.

Product term The Boolean product of two or more literals equivalent to an AND operation.

Sum-of-products (SOP) A form of Boolean expression that is basically the ORing of ANDed terms.

Sum term The Boolean sum of two or more literals equivalent to an OR operation.

Variable A symbol used to represent an action, a condition, or data that can have a value of 1 or 0, usually designated by an italic letter or word.

TRUE/FALSE QUIZ

Answers are at the end of the chapter.

1. *Variable, complement,* and *literal* are all terms used in Boolean algebra.

2. Addition in Boolean algebra is equivalent to the NOR function.

3. Multiplication in Boolean algebra is equivalent to the AND function.

4. The commutative law, associative law, and distributive law are all laws in Boolean algebra.

5. The complement of 0 is 0 itself.

6. When a Boolean variable is multiplied by its complement, the result is the variable.