5.2 Applications-Reading

Notebook:	Discrete Mathematics [CM1020]	
Created:	2019-10-07 2:31 PM	**Updated:**　2019-11-23 12:34 PM
Author:	SUKHJIT MANN	

Cornell Notes	**Topic:** 5.2 Applications-Reading	Course: BSc Computer Science
		Class: Discrete Mathematics-Reading
		Date: November 23, 2019

Essential Question:

What are gates and how do we combine them. Also what is the meant by the minimization of a Boolean function?

Questions/Cues:

- What are the different types of gates and what are combinational circuits?
- What is a half and full adder?
- What is meant by the minimization of a Boolean function?
- What are benefits of minimization when designing circuits?
- What is a Karnaugh Map?
- What is a K-map in three variables?
- What is a K-map in four variables?
- What are Don't Care Conditions?
- What is the Quine-McCluskey Method?
- What are the steps for using the Quine-McCluskey Method to simplify a sum-of-products expression?

Notes

Each type of gate implements a Boolean operation. In this section we define several types of gates. Using these gates, we will apply the rules of Boolean algebra to design circuits that perform a variety of tasks. The circuits that we will study in this chapter give output that depends only on the input, and not on the current state of the circuit. In other words, these circuits have no memory capabilities. Such circuits are called **combinational circuits** or **gating networks**.

We will construct combinational circuits using three types of elements. The first is an **inverter**, which accepts the value of one Boolean variable as input and produces the complement of this value as its output. The symbol used for an inverter is shown in Figure 1(a). The input to the inverter is shown on the left side entering the element, and the output is shown on the right side leaving the element.

The next type of element we will use is the **OR gate**. The inputs to this gate are the values of two or more Boolean variables. The output is the Boolean sum of their values. The symbol used for an OR gate is shown in Figure 1(b). The inputs to the OR gate are shown on the left side entering the element, and the output is shown on the right side leaving the element.

The third type of element we will use is the **AND gate**. The inputs to this gate are the values of two or more Boolean variables. The output is the Boolean product of their values. The symbol used for an AND gate is shown in Figure 1(c). The inputs to the AND gate are shown on the left side entering the element, and the output is shown on the right side leaving the element.

We will permit multiple inputs to AND and OR gates. The inputs to each of these gates are shown on the left side entering the element, and the output is shown on the right side.
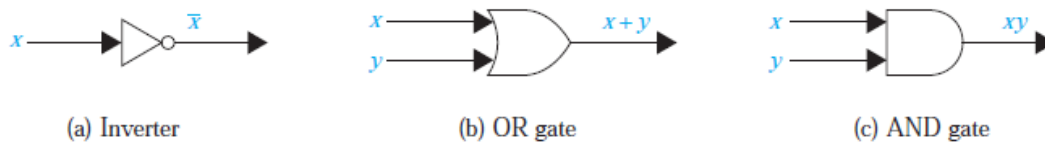

(a) Inverter　　　　　　(b) OR gate　　　　　　(c) AND gate

FIGURE 1　Basic Types of Gates.

FIGURE 2　Gates with n Inputs.

Combinations of Gates

Combinational circuits can be constructed using a combination of inverters, OR gates, and AND gates. When combinations of circuits are formed, some gates may share inputs. This is shown in one of two ways in depictions of circuits. One method is to use branchings that indicate all the gates that use a given input. The other method is to indicate this input separately for each gate. Figure 3 illustrates the two ways of showing gates with the same input values. Note also that output from a gate may be used as input by one or more other elements, as shown in Figure 3. Both drawings in Figure 3 depict the circuit that produces the output $xy + \overline{x}y$.
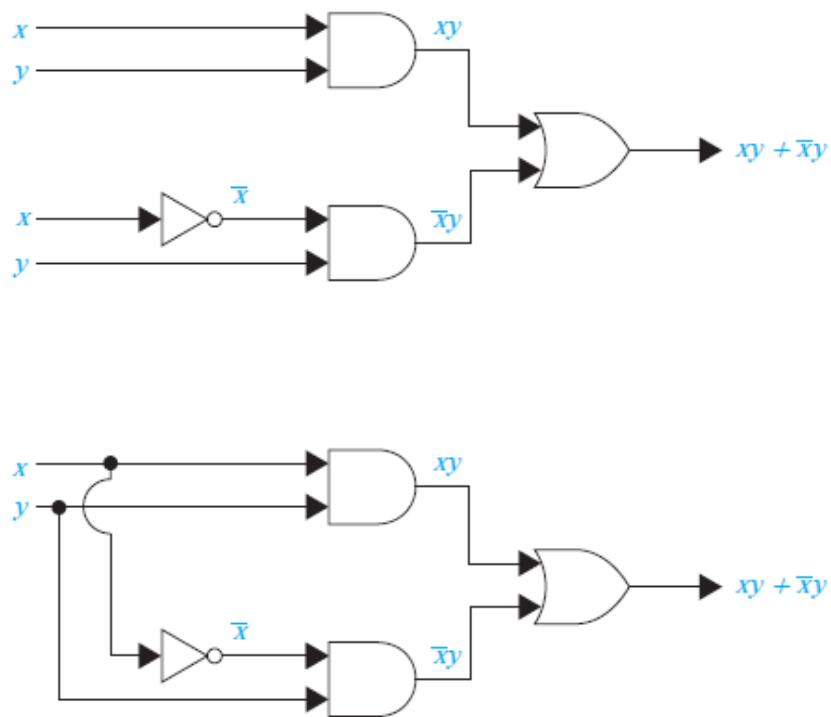
FIGURE 3 Two Ways to Draw the Same Circuit.

Construct circuits that produce the following outputs: (a) $(x + y)\bar{x}$, (b) $\bar{x}\,\overline{(y + \bar{z})}$, and (c) $(x + y + z)(\bar{x}\,\bar{y}\,\bar{z})$.

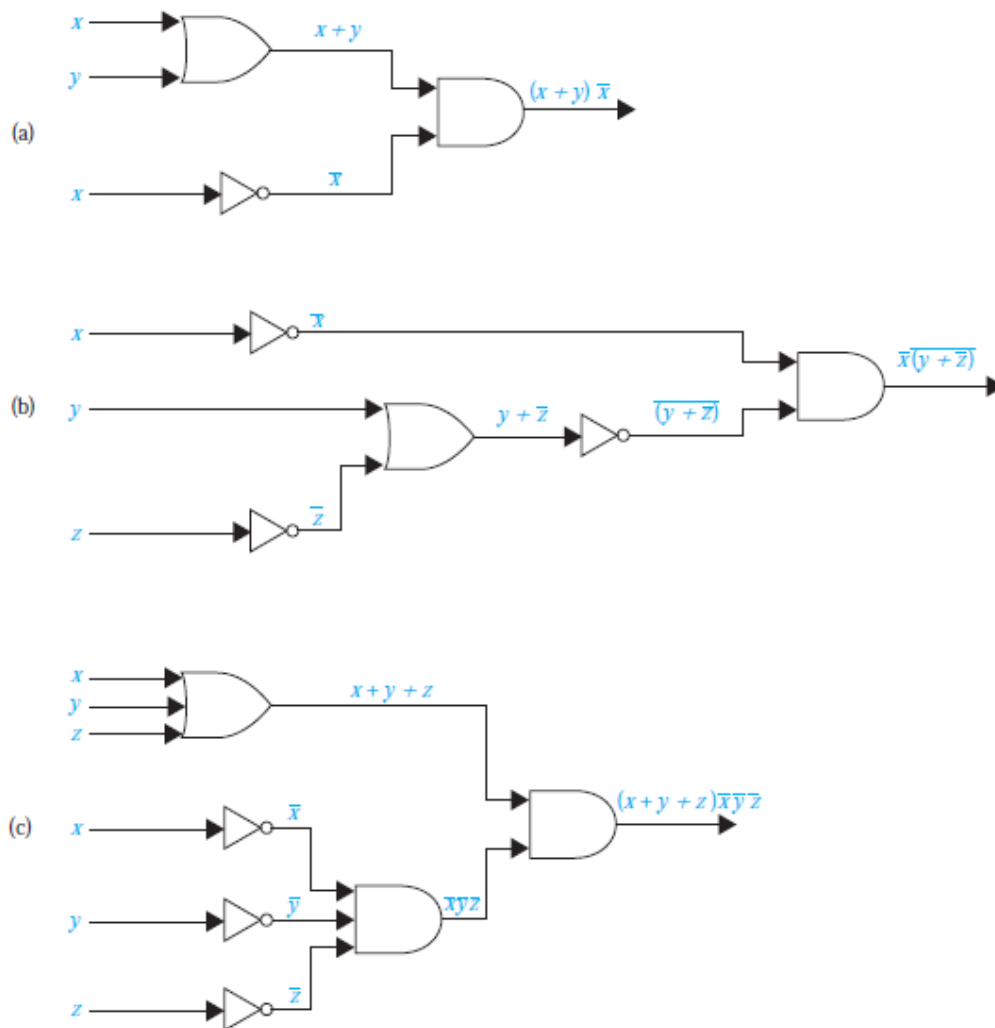*Solution:* Circuits that produce these outputs are shown in Figure 4. ◀

(a)

(b)

(c)

FIGURE 4 Circuits that Produce the Outputs Specified in Example 1.

A committee of three individuals decides issues for an organization. Each individual votes either yes or no for each proposal that arises. A proposal is passed if it receives at least two yes votes. Design a circuit that determines whether a proposal passes.

Solution: Let $x = 1$ if the first individual votes yes, and $x = 0$ if this individual votes no; let $y = 1$ if the second individual votes yes, and $y = 0$ if this individual votes no; let $z = 1$ if the third individual votes yes, and $z = 0$ if this individual votes no. Then a circuit must be designed that produces the output 1 from the inputs x, y, and z when two or more of x, y, and z are 1. One representation of the Boolean function that has these output values is $xy + xz + yz$ (see Exercise 12 in Section 12.1). The circuit that implements this function is shown in Figure 5. ◀
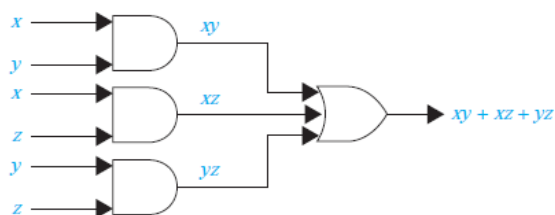

FIGURE 5 A Circuit for Majority Voting.

EXAMPLE 3 Sometimes light fixtures are controlled by more than one switch. Circuits need to be designed so that flipping any one of the switches for the fixture turns the light on when it is off and turns the light off when it is on. Design circuits that accomplish this when there are two switches and when there are three switches.

Solution: We will begin by designing the circuit that controls the light fixture when two different switches are used. Let $x = 1$ when the first switch is closed and $x = 0$ when it is open, and let $y = 1$ when the second switch is closed and $y = 0$ when it is open. Let $F(x, y) = 1$ when the light is on and $F(x, y) = 0$ when it is off. We can arbitrarily decide that the light will be on when both switches are closed, so that $F(1, 1) = 1$. This determines all the other values of F. When one of the two switches is opened, the light goes off, so $F(1, 0) = F(0, 1) = 0$. When the other switch is also opened, the light goes on, so $F(0, 0) = 1$. Table 1 displays these values. Note that $F(x, y) = xy + \overline{x}\,\overline{y}$. This function is implemented by the circuit shown in Figure 6.
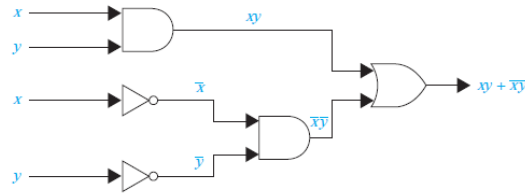
TABLE 1

x	y	F(x, y)
1	1	1
1	0	0
0	1	0
0	0	1

FIGURE 6   A Circuit for a Light Controlled by Two Switches.

We will now design a circuit for three switches. Let x, y, and z be the Boolean variables that indicate whether each of the three switches is closed. We let $x = 1$ when the first switch is closed, and $x = 0$ when it is open; $y = 1$ when the second switch is closed, and $y = 0$ when it is open; and $z = 1$ when the third switch is closed, and $z = 0$ when it is open. Let $F(x, y, z) = 1$ when the light is on and $F(x, y, z) = 0$ when the light is off. We can arbitrarily specify that the light be on when all three switches are closed, so that $F(1, 1, 1) = 1$. This determines all other values of F. When one switch is opened, the light goes off, so $F(1, 1, 0) = F(1, 0, 1) = F(0, 1, 1) = 0$. When a second switch is opened, the light goes on, so $F(1, 0, 0) = F(0, 1, 0) = F(0, 0, 1) = 1$. Finally, when the third switch is opened, the light goes off again, so $F(0, 0, 0) = 0$. Table 2 shows the values of this function.

The function F can be represented by its sum-of-products expansion as $F(x, y, z) = xyz + x\overline{y}\,\overline{z} + \overline{x}y\overline{z} + \overline{x}\,\overline{y}z$. The circuit shown in Figure 7 implements this function. ◀

TABLE 2

x	y	z	F(x, y, z)
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0
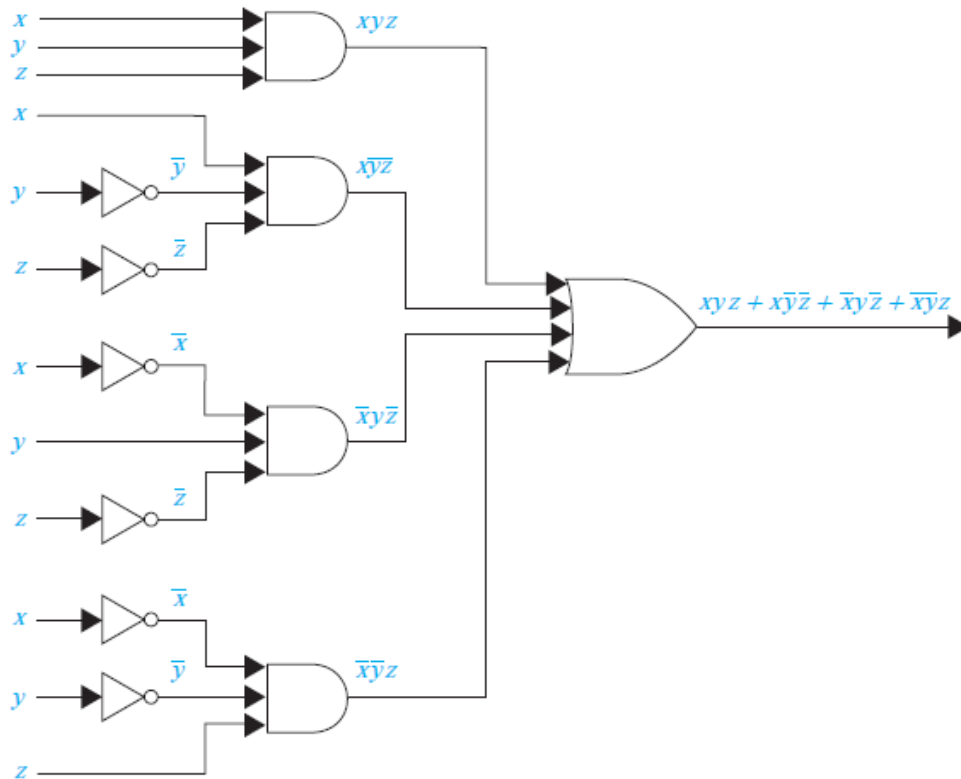
FIGURE 7   **A Circuit for a Fixture Controlled by Three Switches.**

Adders

We will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions. We will build up the circuitry to do this addition from some component circuits. First, we will build a circuit that can be used to find $x + y$, where x and y are two bits. The input to our circuit will be x and y, because these each have the value 0 or the value 1. The output will consist of two bits, namely, s and c, where s is the sum bit and c is the carry bit. This circuit is called a **multiple output circuit** because it has more than one output. The circuit that we are designing is called the **half adder**, because it adds two bits, without considering a carry from a previous addition. We show the input and output for the half adder in Table 3. From Table 3 we see that $c = xy$ and that $s = x\overline{y} + \overline{x}y = (x + y)\overline{(xy)}$. Hence, the circuit shown in Figure 8 computes the sum bit s and the carry bit c from the bits x and y.

We use the **full adder** to compute the sum bit and the carry bit when two bits and a carry are added. The inputs to the full adder are the bits x and y and the carry c_i. The outputs are the sum bit s and the new carry c_{i+1}. The inputs and outputs for the full adder are shown in Table 4.

The two outputs of the full adder, the sum bit s and the carry c_{i+1}, are given by the sum-of-products expansions $xyc_i + x\overline{y}\,\overline{c}_i + \overline{x}y\overline{c}_i + \overline{x}\,\overline{y}c_i$ and $xyc_i + xy\overline{c}_i + x\overline{y}c_i + \overline{x}yc_i$, respectively. However, instead of designing the full adder from scratch, we will use half adders to produce the desired output. A full adder circuit using half adders is shown in Figure 9.

Finally, in Figure 10 we show how full and half adders can be used to add the two three-bit integers $(x_2x_1x_0)_2$ and $(y_2y_1y_0)_2$ to produce the sum $(s_3s_2s_1s_0)_2$. Note that s_3, the highest-order bit in the sum, is given by the carry c_2.
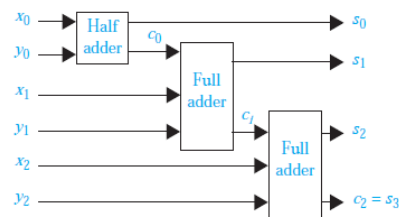
FIGURE 10   Adding Two Three-Bit Integers with Full and Half Adders.
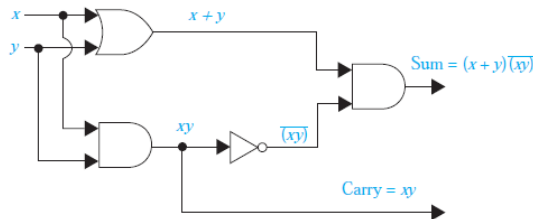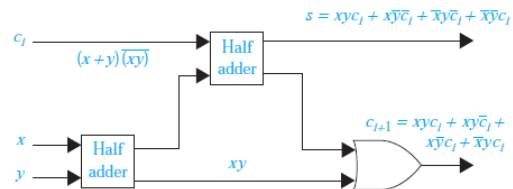
FIGURE 8   The Half Adder.

FIGURE 9   A Full Adder.

Terms in a sum-of-products expansion that differ in just one variable, so that in one term this variable occurs and in the other term the complement of this variable occurs, can be combined. For instance, consider the circuit that has output 1 if and only if $x = y = z = 1$ or $x = z = 1$ and $y = 0$. The sum-of-products expansion of this circuit is $xyz + x\bar{y}z$. The two products in this expansion differ in exactly one variable, namely, y. They can be combined as

$$
\begin{aligned}
xyz + x\bar{y}z &= (y + \bar{y})(xz) \\
&= 1 \cdot (xz) \\
&= xz.
\end{aligned}
$$

Hence, xz is a Boolean expression with fewer operators that represents the circuit. We show two different implementations of this circuit in Figure 1. The second circuit uses only one gate, whereas the first circuit uses three gates and an inverter.

This example shows that combining terms in the sum-of-products expansion of a circuit leads to a simpler expression for the circuit. We will describe two procedures that simplify sum-of-products expansions.

The goal of both procedures is to produce Boolean sums of Boolean products that represent a Boolean function with the fewest products of literals such that these products contain the fewest literals possible among all sums of products that represent a Boolean function. Finding such a sum of products is called **minimization of the Boolean function**. Minimizing a Boolean function makes it possible to construct a circuit for this function that uses the fewest gates and fewest inputs to the AND gates and OR gates in the circuit, among all circuits for the Boolean expression we are minimizing.
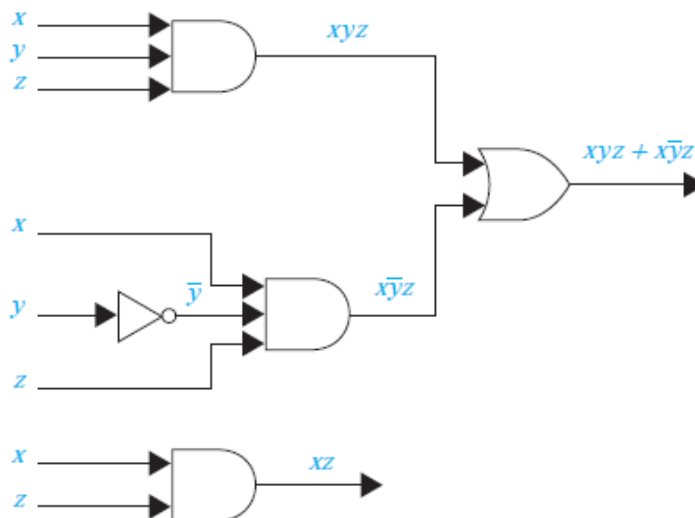
FIGURE 1   Two Circuits with the Same Output.

Until the early 1960s logic gates were individual components. To reduce costs it was important to use the fewest gates to produce a desired output. However, in the mid-1960s, integrated circuit technology was developed that made it possible to combine gates on a single chip. Even though it is now possible to build increasingly complex integrated circuits on chips at low cost, minimization of Boolean functions remains important.

Reducing the number of gates on a chip can lead to a more reliable circuit and can reduce the cost to produce the chip. Also, minimization makes it possible to fit more circuits on the same chip. Furthermore, minimization reduces the number of inputs to gates in a circuit. This reduces the time used by a circuit to compute its output. Moreover, the number of inputs to a gate may be limited because of the particular technology used to build logic gates.

The first procedure we will introduce, known as Karnaugh maps (or K-maps), was designed in the 1950s to help minimize circuits by hand. K-maps are useful in minimizing circuits with up to six variables, although they become rather complex even for five or six variables. The second procedure we will describe, the Quine–McCluskey method, was invented in the 1960s. It automates the process of minimizing combinatorial circuits and can be implemented as a computer program.

Karnaugh Maps

To reduce the number of terms in a Boolean expression representing a circuit, it is necessary to find terms to combine. There is a graphical method, called a **Karnaugh map** or **K-map**, for finding terms to combine for Boolean functions involving a relatively small number of variables. The method we will describe was introduced by Maurice Karnaugh in 1953. His method is based on earlier work by E. W. Veitch. (This method is usually applied only when the function involves six or fewer variables.) K-maps give us a visual method for simplifying sum-of-products expansions; they are not suited for mechanizing this process. We will first illustrate how K-maps are used to simplify expansions of Boolean functions in two variables. We will continue by showing how K-maps can be used to minimize Boolean functions in three variables and then in four variables. Then we will describe the concepts that can be used to extend K-maps to minimize Boolean functions in more than four variables.

There are four possible minterms in the sum-of-products expansion of a Boolean function in the two variables x and y. A K-map for a Boolean function in these two variables consists of four cells, where a 1 is placed in the cell representing a minterm if this minterm is present in the expansion. Cells are said to be **adjacent** if the minterms that they represent differ in exactly one literal. For instance, the cell representing $\overline{x}y$ is adjacent to the cells representing xy and $\overline{x}\,\overline{y}$. The four cells and the terms that they represent are shown in Figure 2.

FIGURE 2
K-maps in Two Variables.

EXAMPLE 1 Find the K-maps for (a) $xy + \overline{x}y$, (b) $x\overline{y} + \overline{x}y$, and (c) $x\overline{y} + \overline{x}y + \overline{x}\,\overline{y}$.

Solution: We include a 1 in a cell when the minterm represented by this cell is present in the sum-of-products expansion. The three K-maps are shown in Figure 3. ◀

We can identify minterms that can be combined from the K-map. Whenever there are 1s in two adjacent cells in the K-map, the minterms represented by these cells can be combined into a product involving just one of the variables. For instance, $x\overline{y}$ and $\overline{x}\,\overline{y}$ are represented by adjacent cells and can be combined into $\overline{y}$, because $x\overline{y} + \overline{x}\,\overline{y} = (x + \overline{x})\overline{y} = \overline{y}$. Moreover, if 1s
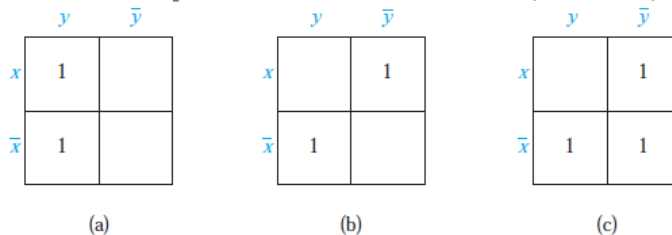
FIGURE 3 **K-maps for the Sum-of-Products Expansions in Example 1.**

are in all four cells, the four minterms can be combined into one term, namely, the Boolean expression 1 that involves none of the variables. We circle blocks of cells in the K-map that represent minterms that can be combined and then find the corresponding sum of products. The goal is to identify the largest possible blocks, and to cover all the 1s with the fewest blocks using the largest blocks first and always using the largest possible blocks.

EXAMPLE 2 Simplify the sum-of-products expansions given in Example 1.

Solution: The grouping of minterms is shown in Figure 4 using the K-maps for these expansions. Minimal expansions for these sums-of-products are (a) y, (b) $x\overline{y} + \overline{x}y$, and (c) $\overline{x} + \overline{y}$. ◀
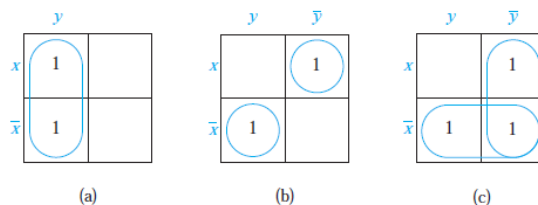
FIGURE 4 **Simplifying the Sum-of-Products Expansions from Example 2.**

A K-map in three variables is a rectangle divided into eight cells. The cells represent the eight possible minterms in three variables. Two cells are said to be adjacent if the minterms that they represent differ in exactly one literal. One of the ways to form a K-map in three variables is shown in Figure 5(a). This K-map can be thought of as lying on a cylinder, as shown in Figure 5(b). On the cylinder, two cells have a common border if and only if they are adjacent.

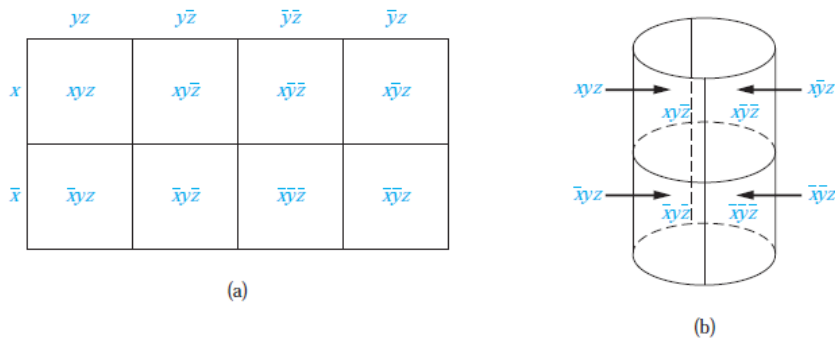|	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
x	xyz	$xy\bar{z}$	$x\bar{y}\bar{z}$	$x\bar{y}z$
$\bar{x}$	$\bar{x}yz$	$\bar{x}y\bar{z}$	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$

(a)

$xyz \rightarrow$ $\leftarrow x\bar{y}z$
$xy\bar{z}$ $x\bar{y}\bar{z}$
$\bar{x}yz \rightarrow$ $\leftarrow \bar{x}\bar{y}z$
$\bar{x}y\bar{z}$ $\bar{x}\bar{y}\bar{z}$

(b)

FIGURE 5   K-maps in Three Variables.

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

(a)

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

(b)

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

(c)

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

(d)

$1 = xyz + xy\bar{z} + x\bar{y}\bar{z} + x\bar{y}z +$
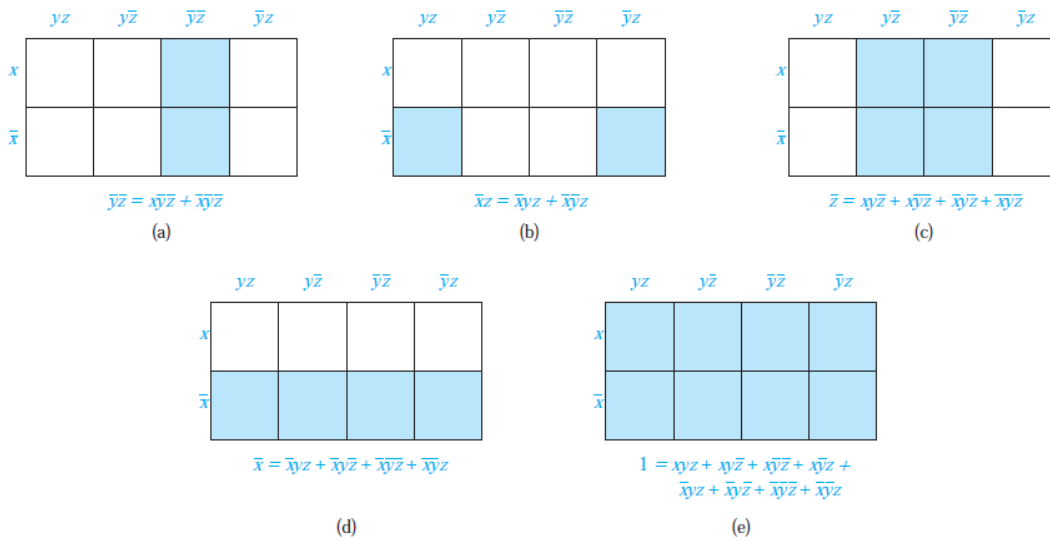$\bar{x}yz + \bar{x}y\bar{z} + \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z$

(e)

FIGURE 6   Blocks in K-maps in Three Variables.

To simplify a sum-of-products expansion in three variables, we use the K-map to identify blocks of minterms that can be combined. Blocks of two adjacent cells represent pairs of minterms that can be combined into a product of two literals; 2×2 and 4×1 blocks of cells represent minterms that can be combined into a single literal; and the block of all eight cells represents a product of no literals, namely, the function 1. In Figure 6, 1×2, 2×1, 2×2, 4×1, and 4×2 blocks and the products they represent are shown.

The product of literals corresponding to a block of all 1s in the K-map is called an **implicant** of the function being minimized. It is called a **prime implicant** if this block of 1s is not contained in a larger block of 1s representing the product of fewer literals than in this product.

The goal is to identify the largest possible blocks in the map and cover all the 1s in the map with the least number of blocks, using the largest blocks first. The largest possible blocks are always chosen, but we must always choose a block if it is the only block of 1s covering a 1 in the K-map. Such a block represents an **essential prime implicant**. By covering all the 1s in the map with blocks corresponding to prime implicants we can express the sum of products as a sum of prime implicants. Note that there may be more than one way to cover all the 1s using the least number of blocks.

EXAMPLE 3 Use K-maps to minimize these sum-of-products expansions.

(a) $xy\overline{z} + x\overline{y}\,\overline{z} + \overline{x}yz + \overline{x}\,\overline{y}\,\overline{z}$

(b) $x\overline{y}z + x\overline{y}\,\overline{z} + \overline{x}yz + \overline{x}\,\overline{y}z + \overline{x}\,\overline{y}\,\overline{z}$

(c) $xyz + xy\overline{z} + x\overline{y}z + x\overline{y}\,\overline{z} + \overline{x}yz + \overline{x}\,\overline{y}z + \overline{x}\,\overline{y}\,\overline{z}$

(d) $xy\overline{z} + x\overline{y}\,\overline{z} + \overline{x}\,\overline{y}z + \overline{x}\,\overline{y}\,\overline{z}$

Solution: The K-maps for these sum-of-products expansions are shown in Figure 7. The grouping of blocks shows that minimal expansions into Boolean sums of Boolean products are (a) $x\overline{z} + \overline{y}\,\overline{z} + \overline{x}yz$, (b) $\overline{y} + \overline{x}z$, (c) $x + \overline{y} + z$, and (d) $x\overline{z} + \overline{x}\,\overline{y}$. In part (d) note that the prime implicants $x\overline{z}$ and $\overline{x}\,\overline{y}$ are essential prime implicants, but the prime implicant $\overline{y}\,\overline{z}$ is a prime implicant that is not essential, because the cells it covers are covered by the other two prime implicants. ◄
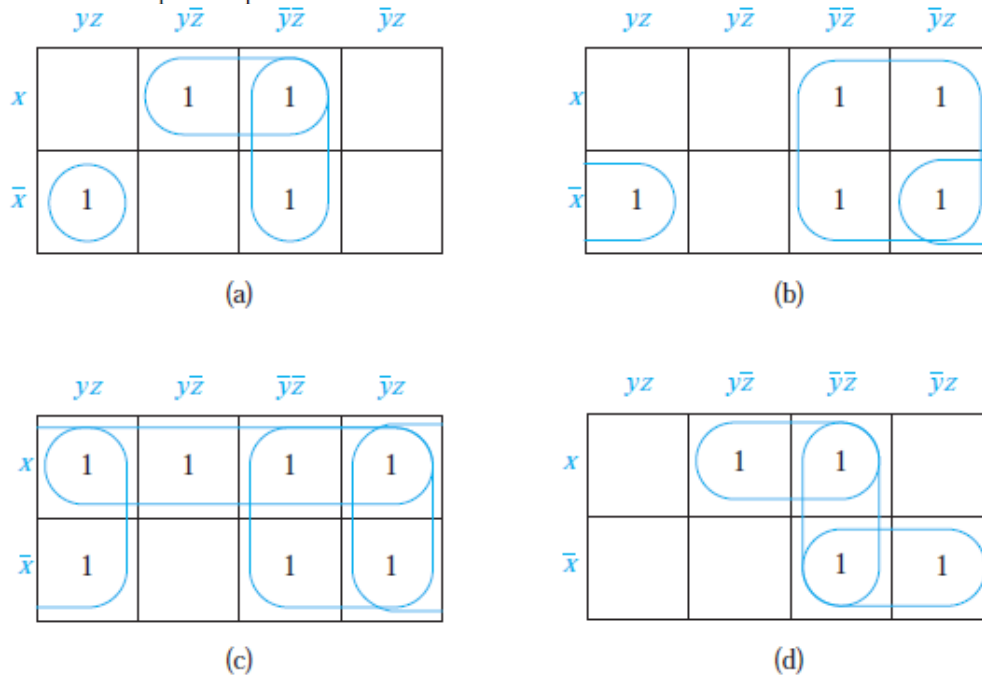

(a)

(b)

(c)

(d)

FIGURE 7 **Using K-maps in Three Variables.**

A K-map in four variables is a square that is divided into 16 cells. The cells represent the 16 possible minterms in four variables. One of the ways to form a K-map in four variables is shown in Figure 8.

Two cells are adjacent if and only if the minterms they represent differ in one literal. Consequently, each cell is adjacent to four other cells. The K-map of a sum-of-products expansion in four variables can be thought of as lying on a torus, so that adjacent cells have a common boundary (see Exercise 28). The simplification of a sum-of-products expansion in four variables is carried out by identifying those blocks of 2, 4, 8, or 16 cells that represent minterms that can be combined. Each cell representing a minterm must either be used to form a product using fewer literals, or be included in the expansion. In Figure 9 some examples of blocks that represent products of three literals, products of two literals, and a single literal are illustrated.

As is the case in K-maps in two and three variables, the goal is to identify the largest blocks of 1s in the map that correspond to the prime implicants and to cover all the 1s using the fewest blocks needed, using the largest blocks first. The largest possible blocks are always used. Example 4 illustrates how K-maps in four variables are used.

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx	$wxyz$	$wxy\bar{z}$	$wx\bar{y}\bar{z}$	$wx\bar{y}z$
$w\bar{x}$	$w\bar{x}yz$	$w\bar{x}y\bar{z}$	$w\bar{x}\bar{y}\bar{z}$	$w\bar{x}\bar{y}z$
$\bar{w}\bar{x}$	$\bar{w}\bar{x}yz$	$\bar{w}\bar{x}y\bar{z}$	$\bar{w}\bar{x}\bar{y}\bar{z}$	$\bar{w}\bar{x}\bar{y}z$
$\bar{w}x$	$\bar{w}xyz$	$\bar{w}xy\bar{z}$	$\bar{w}x\bar{y}\bar{z}$	$\bar{w}x\bar{y}z$

FIGURE 8 K-maps in Four Variables.

$$w\bar{x}z = w\bar{x}yz + w\bar{x}\bar{y}z$$

(a)

$$\bar{w}\bar{x} = \bar{w}\bar{x}yz + \bar{w}\bar{x}y\bar{z} + \bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}\bar{y}z$$

(b)

$$x\bar{z} = wxyz + wx\bar{y}z + \bar{w}xyz + \bar{w}x\bar{y}z$$

(c)

$$\bar{z} = wxy\bar{z} + wx\bar{y}\bar{z} + w\bar{x}y\bar{z} + w\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}y\bar{z} + \bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}xy\bar{z} + \bar{w}x\bar{y}\bar{z}$$

(d)

FIGURE 9 Blocks in K-maps in Four Variables.

EXAMPLE 4 Use K-maps to simplify these sum-of-products expansions.

(a) $wxyz + wxy\overline{z} + wx\overline{y}\,\overline{z} + w\overline{x}yz + w\overline{x}\,\overline{y}z + w\overline{x}\,\overline{y}\,\overline{z} + \overline{w}x\overline{y}z + \overline{w}\,\overline{x}yz + \overline{w}\,\overline{x}y\overline{z}$

(b) $wx\overline{y}\,\overline{z} + w\overline{x}yz + w\overline{x}y\overline{z} + w\overline{x}\,\overline{y}\,\overline{z} + \overline{w}xyz + \overline{w}xy\overline{z} + \overline{w}x\,\overline{y}\,\overline{z}$

(c) $wxy\overline{z} + wx\overline{y}\,\overline{z} + w\overline{x}yz + w\overline{x}y\overline{z} + w\overline{x}\,\overline{y}\,\overline{z} + \overline{w}xyz + \overline{w}xy\overline{z} + \overline{w}x\,\overline{y}\,\overline{z} + \overline{w}\,\overline{x}y\overline{z} + \overline{w}\,\overline{x}\,\overline{y}\,\overline{z} + \overline{w}\,\overline{x}\,\overline{y}\,\overline{z}$

Solution: The K-maps for these expansions are shown in Figure 10. Using the blocks shown leads to the sum of products (a) $wyz + wx\overline{z} + w\overline{x}\,\overline{y} + \overline{w}\,\overline{x}y + \overline{w}x\overline{y}z$, (b) $\overline{y}\,\overline{z} + w\overline{x}y + \overline{x}\,\overline{z}$, and (c) $\overline{z} + \overline{w}x + w\overline{x}y$. The reader should determine whether there are other choices of blocks in each part that lead to different sums of products representing these Boolean functions. ◀

K-maps can realistically be used to minimize Boolean functions with five or six variables, but beyond that, they are rarely used because they become extremely complicated. However, the concepts used in K-maps play an important role in newer algorithms. Furthermore, mastering these concepts helps you understand these newer algorithms and the computer-aided design (CAD) programs that implement them. As we develop these concepts, we will be able to illustrate them by referring back to our discussion of minimization of Boolean functions in three and in four variables.

The K-maps we used to minimize Boolean functions in two, three, and four variables are built using 2×2, 2×4, and 4×4 rectangles, respectively. Furthermore, corresponding cells in the top row and bottom row and in the leftmost column and rightmost column in each of these
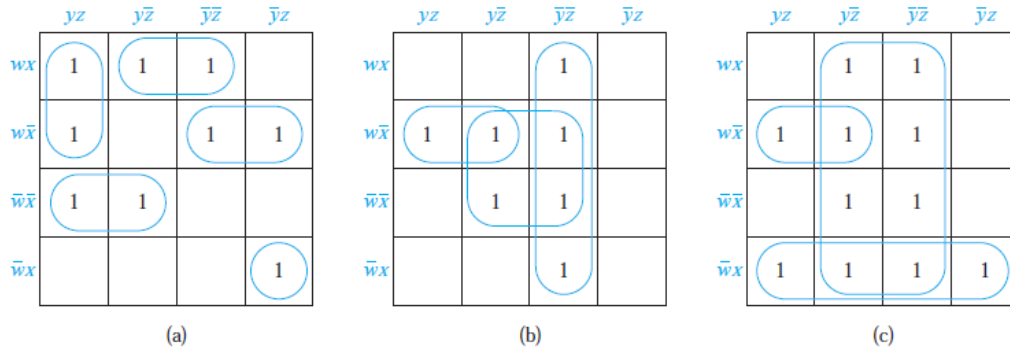

FIGURE 10 Using K-maps in Four Variables.

cases are considered adjacent because they represent minterms differing in only one literal. We can build K-maps for minimizing Boolean functions in more than four variables in a similar way. We use a rectangle containing $2^{\lfloor n/2 \rfloor}$ rows and $2^{\lceil n/2 \rceil}$ columns. (These K-maps contain 2^n cells because $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$.) The rows and columns need to be positioned so that the cells representing minterms differing in just one literal are adjacent or are considered adjacent by specifying additional adjacencies of rows and columns. To help (but not entirely) achieve this, the rows and columns of a K-map are arranged using Gray codes (see Section 10.5), where we associate bit strings and products by specifying that a 1 corresponds to the appearance of a variable and a 0 with the appearance of its complement. For example, in a 10-dimensional K-map, the Gray code 01110 used to label a row corresponds to the product $\overline{x}_1x_2x_3x_4\overline{x}_5$.

EXAMPLE 5 The K-maps we used to minimize Boolean functions with four variables have four rows and four columns. Both the rows and the columns are arranged using the Gray code 11,10,00,01. The rows represent products wx, $w\overline{x}$, $\overline{w}\,\overline{x}$, and $\overline{w}x$, respectively, and the columns correspond to the products yz, $y\overline{z}$, $\overline{y}\,\overline{z}$, and $\overline{y}z$, respectively. Using Gray codes and considering cells adjacent in the first and last rows and in the first and last columns, we ensured that minterms that differ in only one variable are always adjacent. ◄

EXAMPLE 6 To minimize Boolean functions in five variables we use K-maps with $2^3 = 8$ columns and $2^2 = 4$ rows. We label the four rows using the Gray code 11,10,00,01, corresponding to x_1x_2, $x_1\overline{x}_2$, $\overline{x}_1\overline{x}_2$, and $\overline{x}_1x_2$, respectively. We label the eight columns using the Gray code 111,110,100,101,001,000,010,011 corresponding to the terms $x_3x_4x_5$, $x_3x_4\overline{x}_5$, $x_3\overline{x}_4\overline{x}_5$, $x_3\overline{x}_4x_5$, $\overline{x}_3\overline{x}_4x_5$, $\overline{x}_3\overline{x}_4\overline{x}_5$, $\overline{x}_3x_4\overline{x}_5$, and $\overline{x}_3x_4x_5$, respectively. Using Gray codes to label columns and rows ensures that the minterms represented by adjacent cells differ in only one variable. However, to make sure all cells representing products that differ in only one variable are considered adjacent, we consider cells in the top and bottom rows to be adjacent, as well as cells in the first and eighth columns, the first and fourth columns, the second and seventh columns, the third and sixth columns, and the fifth and eighth columns (as the reader should verify). ◄

To use a K-map to minimize a Boolean function in n variables, we first draw a K-map of the appropriate size. We place 1s in all cells corresponding to minterms in the sum-of-products expansion of this function. We then identify all prime implicants of the Boolean function. To do this we look for the blocks consisting of 2^k clustered cells all containing a 1, where $1 \leq k \leq n$. These blocks correspond to the product of $n - k$ literals. (Exercise 33 asks the reader to verify this.) Furthermore, a block of 2^k cells each containing a 1 not contained in a block of 2^{k+1} cells each containing a 1 represents a prime implicant. The reason that this implicant is a prime implicant is that no product obtained by deleting a literal is also represented by a block of cells all containing 1s.

EXAMPLE 7 A block of eight cells representing a product of two literals in a K-map for minimizing Boolean functions in five variables all containing 1s is a prime implicant if it is not contained in a larger block of 16 cells all containing 1s representing a single literal. ◄

Once all prime implicants have been identified, the goal is to find the smallest possible subset of these prime implicants with the property that the cells representing these prime implicants cover all the cells containing a 1 in the K-map. We begin by selecting the essential prime implicants because each of these is represented by a block that covers a cell containing a 1 that is not covered by any other prime implicant. We add additional prime implicants to ensure that all 1s in the K-map are covered. When the number of variables is large, this last step can become exceedingly complicated.

Don't Care Conditions

In some circuits we care only about the output for some combinations of input values, because other combinations of input values are not possible or never occur. This gives us freedom in producing a simple circuit with the desired output because the output values for all those combinations that never occur can be arbitrarily chosen. The values of the function for these combinations are called *don't care* conditions. A d is used in a K-map to mark those combinations of values of the variables for which the function can be arbitrarily assigned. In the minimization process we can assign 1s as values to those combinations of the input values that lead to the largest blocks in the K-map. This is illustrated in Example 8.

EXAMPLE 8 One way to code decimal expansions using bits is to use the four bits of the binary expansion of each digit in the decimal expansion. For instance, 873 is encoded as 100001110011. This encoding of a decimal expansion is called a **binary coded decimal expansion**. Because there are 16 blocks of four bits and only 10 decimal digits, there are six combinations of four bits that are not used to encode digits. Suppose that a circuit is to be built that produces an output of 1 if the decimal digit is 5 or greater and an output of 0 if the decimal digit is less than 5. How can this circuit be simply built using OR gates, AND gates, and inverters?

Solution: Let $F(w, x, y, z)$ denote the output of the circuit, where $wxyz$ is a binary expansion of a decimal digit. The values of F are shown in Table 1. The K-map for F, with ds in the *don't care* positions, is shown in Figure 11(a). We can either include or exclude squares with ds from blocks. This gives us many possible choices for the blocks. For example, excluding all squares with ds and forming blocks, as shown in Figure 11(b), produces the expression $w\overline{x}\,\overline{y} + \overline{w}xy + \overline{w}xz$. Including some of the ds and excluding others and forming blocks, as

TABLE 1

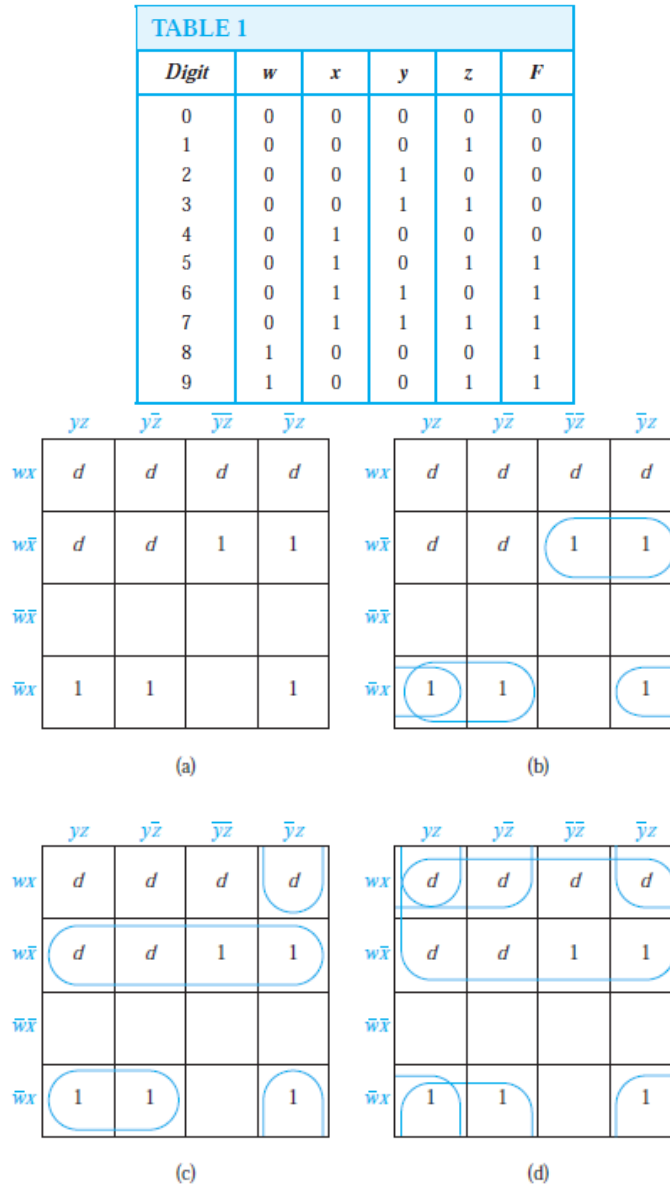| Digit	w	x	y	z	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1

(a)   (b)

(c)   (d)

FIGURE 11 The K-map for F Showing Its *Don't Care* Positions.

shown in Figure 11(c), produces the expression $w\overline{x} + \overline{w}xy + x\overline{y}z$. Finally, including all the ds and using the blocks shown in Figure 11(d) produces the simplest sum-of-products expansion possible, namely, $F(x, y, z) = w + xy + xz$.

The Quine–McCluskey Method

We have seen that K-maps can be used to produce minimal expansions of Boolean functions as Boolean sums of Boolean products. However, K-maps are awkward to use when there are more than four variables. Furthermore, the use of K-maps relies on visual inspection to identify terms to group. For these reasons there is a need for a procedure for simplifying sum-of-products expansions that can be mechanized. The Quine–McCluskey method is such a procedure. It can be used for Boolean functions in any number of variables. It was developed in the 1950s by W. V. Quine and E. J. McCluskey, Jr. Basically, the Quine–McCluskey method consists of two parts. The first part finds those terms that are candidates for inclusion in a minimal expansion as a Boolean sum of Boolean products. The second part determines which of these terms to actually use. We will use Example 9 to illustrate how, by successively combining implicants into implicants with one fewer literal, this procedure works.

EXAMPLE 9 We will show how the Quine–McCluskey method can be used to find a minimal expansion equivalent to

$$xyz + x\bar{y}z + \bar{x}yz + \bar{x}\,\bar{y}z + \bar{x}\,\bar{y}\,\bar{z}.$$

We will represent the minterms in this expansion by bit strings. The first bit will be 1 if x occurs and 0 if $\bar{x}$ occurs. The second bit will be 1 if y occurs and 0 if $\bar{y}$ occurs. The third bit will be 1 if z occurs and 0 if $\bar{z}$ occurs. We then group these terms according to the number of 1s in the corresponding bit strings. This information is shown in Table 2.

Minterms that can be combined are those that differ in exactly one literal. Hence, two terms that can be combined differ by exactly one in the number of 1s in the bit strings that represent them. When two minterms are combined into a product, this product contains two literals. A product in two literals is represented using a dash to denote the variable that does not occur. For instance, the minterms $x\bar{y}z$ and $\bar{x}\,\bar{y}z$, represented by bit strings 101 and 001, can be combined into $\bar{y}z$, represented by the string –01. All pairs of minterms that can be combined and the product formed from these combinations are shown in Table 3.

Next, all pairs of products of two literals that can be combined are combined into one literal. Two such products can be combined if they contain literals for the same two variables, and literals for only one of the two variables differ. In terms of the strings representing the products, these strings must have a dash in the same position and must differ in exactly one of the other two slots. We can combine the products yz and $\bar{y}z$, represented by the strings –11 and –01, into z, represented by the string – –1. We show all the combinations of terms that can be formed in this way in Table 3.

TABLE 2		
Minterm	*Bit String*	*Number of 1s*
xyz	111	3
$x\bar{y}z$	101	2
$\bar{x}yz$	011	2
$\bar{x}\,\bar{y}z$	001	1
$\bar{x}\,\bar{y}\,\bar{z}$	000	0

TABLE 3							
			Step 1			Step 2	
	Term	*Bit String*		*Term*	*String*		*Term*	*String*
1	xyz	111	(1,2)	xz	1–1	(1,2,3,4)	z	– –1
2	$x\bar{y}z$	101	(1,3)	yz	–11			
3	$\bar{x}yz$	011	(2,4)	$\bar{y}z$	–01			
4	$\bar{x}\,\bar{y}z$	001	(3,4)	$\bar{x}z$	0–1			
5	$\bar{x}\,\bar{y}\,\bar{z}$	000	(4,5)	$\bar{x}\,\bar{y}$	00–			

In Table 3 we also indicate which terms have been used to form products with fewer literals; these terms will not be needed in a minimal expansion. The next step is to identify a minimal set of products needed to represent the Boolean function. We begin with all those products that were not used to construct products with fewer literals. Next, we form Table 4, which has a row for each candidate product formed by combining original terms, and a column for each original term; and we put an X in a position if the original term in the sum-of-products expansion was used to form this candidate product. In this case, we say that the candidate product **covers** the original minterm. We need to include at least one product that covers each of the original minterms. Consequently, whenever there is only one X in a column in the table, the product corresponding to the row this X is in must be used. From Table 4 we see that both z and $\overline{x}\,\overline{y}$ are needed. Hence, the final answer is $z + \overline{x}\,\overline{y}$. ◀

TABLE 4

	xyz	$x\overline{y}z$	$\overline{x}yz$	$\overline{x}\,\overline{y}z$	$\overline{x}\,\overline{y}\,\overline{z}$
z	X	X	X	X	
$\overline{x}\,\overline{y}$				X	X

As was illustrated in Example 9, the Quine–McCluskey method uses this sequence of steps to simplify a sum-of-products expression.

1. Express each minterm in n variables by a bit string of length n with a 1 in the ith position if x_i occurs and a 0 in this position if $\overline{x}_i$ occurs.

2. Group the bit strings according to the number of 1s in them.

3. Determine all products in $n - 1$ variables that can be formed by taking the Boolean sum of minterms in the expansion. Minterms that can be combined are represented by bit strings that differ in exactly one position. Represent these products in $n - 1$ variables with strings that have a 1 in the ith position if x_i occurs in the product, a 0 in this position if $\overline{x}_i$ occurs, and a dash in this position if there is no literal involving x_i in the product.

4. Determine all products in $n - 2$ variables that can be formed by taking the Boolean sum of the products in $n - 1$ variables found in the previous step. Products in $n - 1$ variables that can be combined are represented by bit strings that have a dash in the same position and differ in exactly one position.

5. Continue combining Boolean products into products in fewer variables as long as possible.

6. Find all the Boolean products that arose that were not used to form a Boolean product in one fewer literal.

7. Find the smallest set of these Boolean products such that the sum of these products represents the Boolean function. This is done by forming a table showing which minterms are covered by which products. Every minterm must be covered by at least one product. The first step in using this table is to find all essential prime implicants. Each essential prime implicant must be included because it is the only prime implicant that covers one of the minterms. Once we have found essential prime implicants, we can simplify the table by eliminating the rows for minterms covered by these prime implicants. Furthermore, we can eliminate any prime implicants that cover a subset of minterms covered by another prime implicant (as the reader should verify). Moreover, we can eliminate from the table the row for a minterm if there is another minterm that is covered by a subset of the prime implicants that cover this minterm. This process of identifying essential prime implicants that must be included, followed by eliminating redundant prime implicants and identifying minterms that can be ignored, is iterated until the table does not change. At this point we use a backtracking procedure to find the optimal solution where we add prime implicants to the cover to find possible solutions, which we compare to the best solution found so far at each step.

EXAMPLE 10 Use the Quine–McCluskey method to simplify the sum-of-products expansion $wxy\bar{z} + w\bar{x}yz + \overline{w}xy\bar{z} + \overline{w}xyz + \overline{w}\bar{x}\bar{y}z + \overline{w}\,\overline{x}\bar{y}z$.

Solution: We first represent the minterms by bit strings and then group these terms together according to the number of 1s in the bit strings. This is shown in Table 5. All the Boolean products that can be formed by taking Boolean sums of these products are shown in Table 6.

The only products that were not used to form products in fewer variables are $\overline{w}z$, $wy\bar{z}$, $w\bar{x}y$, and $\bar{x}yz$. In Table 7 we show the minterms covered by each of these products. To cover these minterms we must include $\overline{w}z$ and $wy\bar{z}$, because these products are the only products that cover $\overline{w}\bar{x}yz$ and $wxy\bar{z}$, respectively. Once these two products are included, we see that only one of the two products left is needed. Consequently, we can take either $\overline{w}z + wy\bar{z} + w\bar{x}y$ or $\overline{w}z + wy\bar{z} + \bar{x}yz$ as the final answer. ◀

TABLE 5

Term	Bit String	Number of 1s
$wxy\bar{z}$	1110	3
$w\bar{x}yz$	1011	3
$\overline{w}xyz$	0111	3
$w\bar{x}y\bar{z}$	1010	2
$\overline{w}x\bar{y}z$	0101	2
$\overline{w}\bar{x}yz$	0011	2
$\overline{w}\,\bar{x}\bar{y}z$	0001	1

TABLE 6

				Step 1			Step 2	
	Term	Bit String		Term	String		Term	String
1	$wxy\bar{z}$	1110	(1,4)	$wy\bar{z}$	1–10	(3,5,6,7)	$\overline{w}z$	0 – –1
2	$w\bar{x}yz$	1011	(2,4)	$w\bar{x}y$	101–			
3	$\overline{w}xyz$	0111	(2,6)	$\bar{x}yz$	–011			
4	$w\bar{x}y\bar{z}$	1010	(3,5)	$\overline{w}xz$	01–1			
5	$\overline{w}x\bar{y}z$	0101	(3,6)	$\overline{w}yz$	0–11			
6	$\overline{w}\bar{x}yz$	0011	(5,7)	$\overline{w}\bar{y}z$	0–01			
7	$\overline{w}\,\bar{x}\bar{y}z$	0001	(6,7)	$\overline{w}\bar{x}z$	00–1			

TABLE 7

	$wxy\bar{z}$	$w\bar{x}yz$	$\overline{w}xyz$	$\overline{w}x\bar{y}z$	$\overline{w}\bar{x}yz$	$\overline{w}\bar{x}yz$	$\overline{w}\,\bar{x}\bar{y}z$
$\overline{w}z$			X		X	X	X
$wy\bar{z}$	X		X				
$w\bar{x}y$		X		X			
$\bar{x}yz$		X				X	

Summary

In this week, we learned about gates, combinational gates. Expanding on this, we learned about the minimization of a Boolean function and its benefit when designing circuits. Lastly we explored Karnaugh Maps and the Quine McCluskey Method for minimizing a Boolean function, also known as simplifying a sum of products Boolean expression.