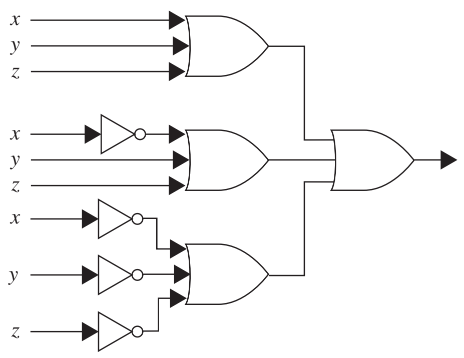
5. 

6. Construct circuits from inverters, AND gates, and OR gates to produce these outputs.

a) $\overline{x} + y$ **b)** $\overline{(x + y)}x$

c) $xyz + \overline{x}\,\overline{y}\,\overline{z}$ **d)** $\overline{(\overline{x} + z)(y + \overline{z})}$

7. Design a circuit that implements majority voting for five individuals.

8. Design a circuit for a light fixture controlled by four switches, where flipping one of the switches turns the light on when it is off and turns it off when it is on.

9. Show how the sum of two five-bit integers can be found using full and half adders.

10. Construct a circuit for a half subtractor using AND gates, OR gates, and inverters. A **half subtractor** has two bits as input and produces as output a difference bit and a borrow.

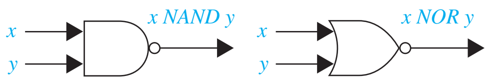
11. Construct a circuit for a full subtractor using AND gates, OR gates, and inverters. A **full subtractor** has two bits and a borrow as input, and produces as output a difference bit and a borrow.

12. Use the circuits from Exercises 10 and 11 to find the difference of two four-bit integers, where the first integer is greater than the second integer.

***13.** Construct a circuit that compares the two-bit integers $(x_1x_0)_2$ and $(y_1y_0)_2$, returning an output of 1 when the first of these numbers is larger and an output of 0 otherwise.

***14.** Construct a circuit that computes the product of the two-bit integers $(x_1x_0)_2$ and $(y_1y_0)_2$. The circuit should have four output bits for the bits in the product.

Two gates that are often used in circuits are NAND and NOR gates. When NAND or NOR gates are used to represent circuits, no other types of gates are needed. The notation for these gates is as follows:



***15.** Use NAND gates to construct circuits with these outputs.

a) $\overline{x}$ **b)** $x + y$

c) xy **d)** $x \oplus y$

***16.** Use NOR gates to construct circuits for the outputs given in Exercise 15.

***17.** Construct a half adder using NAND gates.

***18.** Construct a half adder using NOR gates.

A **multiplexer** is a switching circuit that produces as output one of a set of input bits based on the value of control bits.

19. Construct a multiplexer using AND gates, OR gates, and inverters that has as input the four bits x_0, x_1, x_2, and x_3 and the two control bits c_0 and c_1. Set up the circuit so that x_i is the output, where i is the value of the two-bit integer $(c_1c_0)_2$.

The **depth** of a combinatorial circuit can be defined by specifying that the depth of the initial input is 0 and if a gate has n different inputs at depths $d_1, d_2, \ldots, d_n$, respectively, then its outputs have depth equal to $\max(d_1, d_2, \ldots, d_n) + 1$; this value is also defined to be the depth of the gate. The depth of a combinatorial circuit is the maximum depth of the gates in the circuit.

20. Find the depth of

a) the circuit constructed in Example 2 for majority voting among three people.

b) the circuit constructed in Example 3 for a light controlled by two switches.

c) the half adder shown in Figure 8.

d) the full adder shown in Figure 9.

12.4 Minimization of Circuits

Introduction

The efficiency of a combinational circuit depends on the number and arrangement of its gates. The process of designing a combinational circuit begins with the table specifying the output for each combination of input values. We can always use the sum-of-products expansion of a circuit to find a set of logic gates that will implement this circuit. However, the sum-of-products expansion

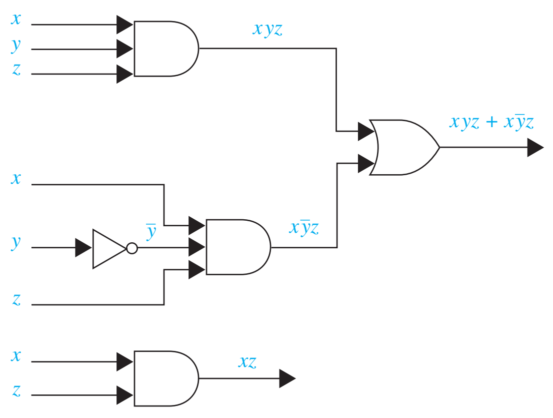


FIGURE 1 **Two Circuits with the Same Output.**

may contain many more terms than are necessary. 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 $xyz + x\overline{y}z$. The two products in this expansion differ in exactly one variable, namely, y. They can be combined as

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

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.

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.

COMPLEXITY OF BOOLEAN FUNCTION MINIMIZATION Unfortunately, minimizing Boolean functions with many variables is a computationally intensive problem. It has been shown that this problem is an NP-complete problem (see Section 3.3 and [Ka93]), so the existence of a polynomial-time algorithm for minimizing Boolean circuits is unlikely. The Quine–McCluskey method has exponential complexity. In practice, it can be used only when the number of literals does not exceed ten. Since the 1970s a number of newer algorithms have been developed for minimizing combinatorial circuits (see [Ha93] and [KaBe04]). However, with the best algorithms yet devised, only circuits with no more than 25 variables can be minimized. Also, heuristic (or rule-of-thumb) methods can be used to substantially simplify, but not necessarily minimize, Boolean expressions with a larger number of literals.

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.

	y	$\overline{y}$
x	xy	$x\overline{y}$
$\overline{x}$	$\overline{x}y$	$\overline{x}\,\overline{y}$

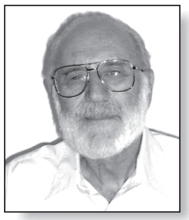
FIGURE 2 K-maps in Two Variables.

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.

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



MAURICE KARNAUGH (BORN 1924) Maurice Karnaugh, born in New York City, received his B.S. from the City College of New York and his M.S. and Ph.D. from Yale University. He was a member of the technical staff at Bell Laboratories from 1952 until 1966 and Manager of Research and Development at the Federal Systems Division of AT&T from 1966 to 1970. In 1970 he joined IBM as a member of the research staff. Karnaugh has made fundamental contributions to the application of digital techniques in both computing and telecommunications. His current interests include knowledge-based systems in computers and heuristic search methods.

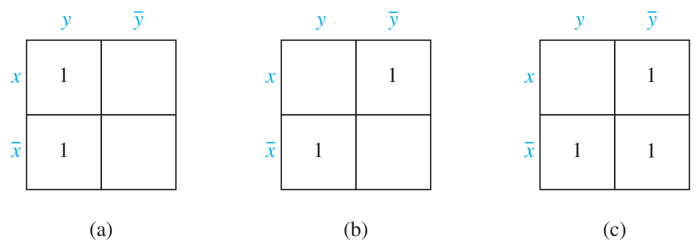


FIGURE 3 **K-maps for the Sum-of-Products Expansions in Example 1.**

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) $\overline{x} + \overline{y}$. ◀

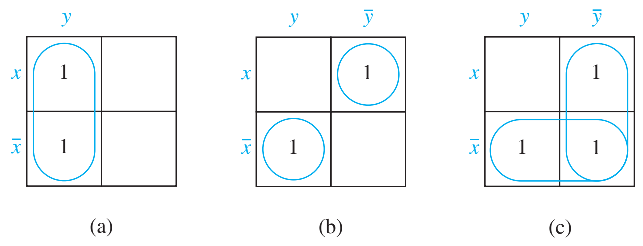


FIGURE 4 **Simplifying the Sum-of-Products Expansions from Example 2.**

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.

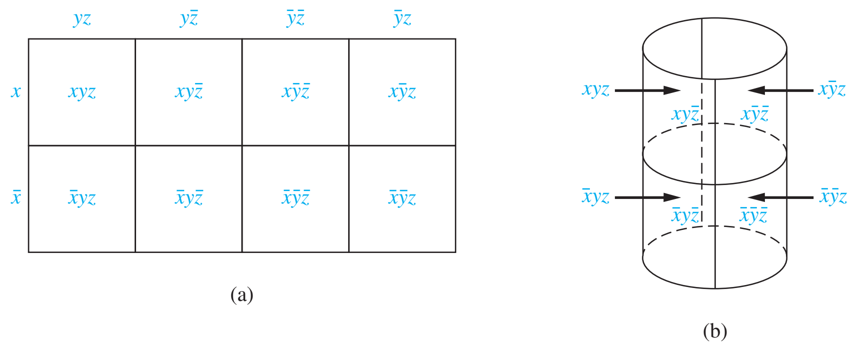


FIGURE 5 **K-maps in Three Variables.**

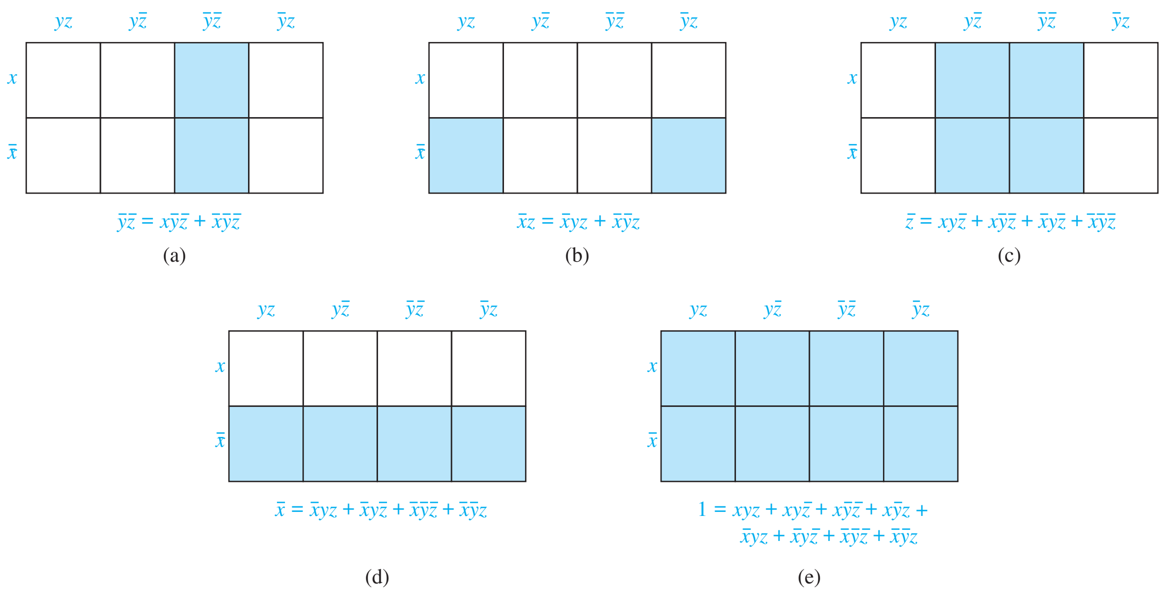

yz
$y\overline{z}$
$\overline{y}\,\overline{z}$
$\overline{y}z$
x
$\overline{x}$
$\overline{y}\,\overline{z} = x\overline{y}\,\overline{z} + \overline{x}\,\overline{y}\,\overline{z}$
(a)
$\overline{x}z = \overline{x}yz + \overline{x}\,\overline{y}z$
(b)
$\overline{z} = xy\overline{z} + x\overline{y}\,\overline{z} + \overline{x}y\overline{z} + \overline{x}\,\overline{y}\,\overline{z}$
(c)
$\overline{x} = \overline{x}yz + \overline{x}y\overline{z} + \overline{x}\,\overline{y}\,\overline{z} + \overline{x}\,\overline{y}z$
(d)
$1 = xyz + xy\overline{z} + x\overline{y}\,\overline{z} + x\overline{y}z + \overline{x}yz + \overline{x}y\overline{z} + \overline{x}\,\overline{y}\,\overline{z} + \overline{x}\,\overline{y}z$
(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 illustrates how K-maps in three variables are used.

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. ◀

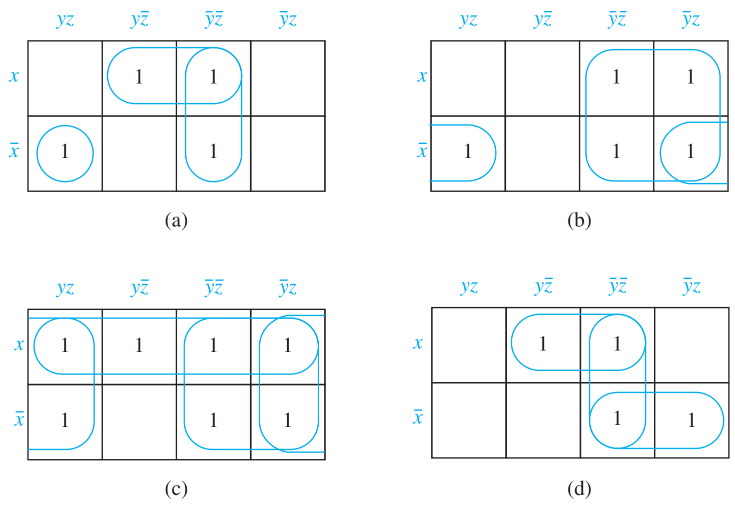
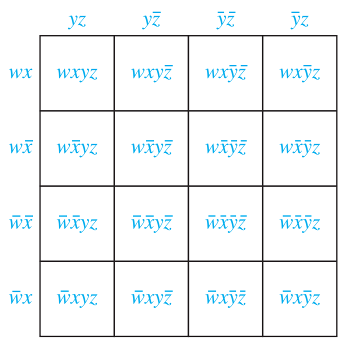


FIGURE 7 **Using K-maps in Three Variables.**

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.**

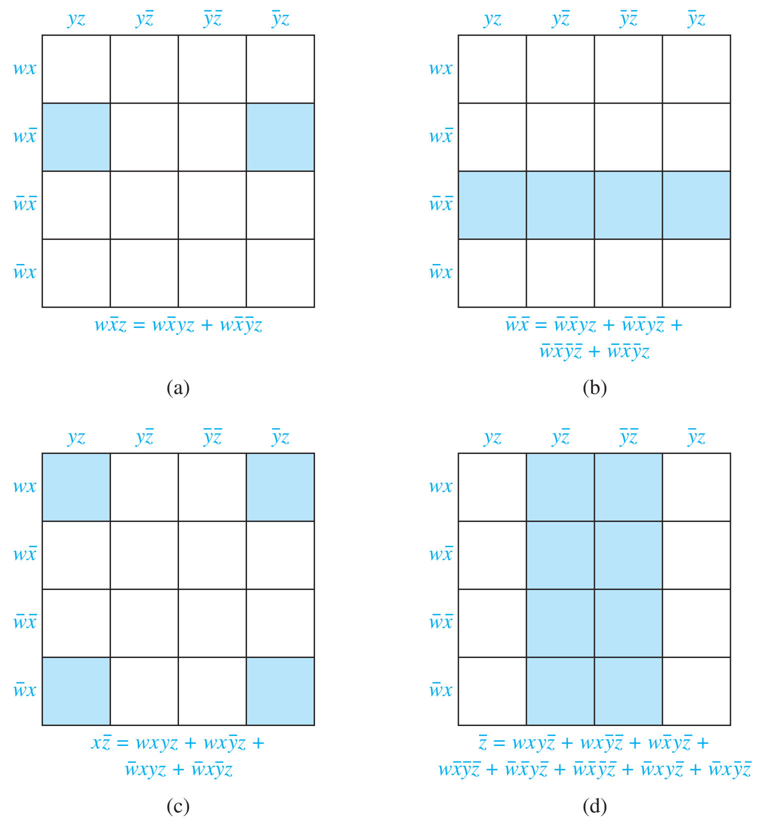

yz $y\bar z$ $\bar y\bar z$ $\bar yz$
wx $w\bar x$ $\bar w\bar x$ $\bar wx$
$w\bar xz = w\bar xyz + w\bar x\bar yz$
(a)
$\bar w\bar x = \bar w\bar xyz + \bar w\bar xy\bar z + \bar w\bar x\bar y\bar z + \bar w\bar x\bar yz$
(b)
$x\bar z = wxyz + wx\bar yz + \bar wxyz + \bar wx\bar yz$
(c)
$\bar z = wxy\bar z + wx\bar y\bar z + w\bar xy\bar z + w\bar x\bar y\bar z + \bar w\bar xy\bar z + \bar w\bar x\bar y\bar z + \bar wxy\bar z + \bar wx\bar y\bar z$
(d)

FIGURE 9 **Blocks in K-maps in Four Variables.**

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

(a) $wxyz + wxy\bar z + wx\bar y\,\bar z + w\bar xyz + w\bar x\,\bar yz + w\bar x\,\bar y\,\bar z + \bar wx\bar yz + \bar w\,\bar xyz + \bar w\,\bar xy\bar z$

(b) $wx\bar y\,\bar z + w\bar xyz + w\bar xy\bar z + w\bar x\,\bar y\,\bar z + \bar wx\bar y\,\bar z + \bar w\,\bar xy\bar z + \bar w\,\bar x\,\bar y\,\bar z$

(c) $wxy\bar z + wx\bar y\,\bar z + w\bar xyz + w\bar xy\bar z + w\bar x\,\bar y\,\bar z + \bar wxyz + \bar wxy\bar z + \bar wx\bar y\,\bar z + \bar wx\bar yz + \bar w\,\bar xy\bar z + \bar w\,\bar x\,\bar y\,\bar z$

Solution: The K-maps for these expansions are shown in Figure 10. Using the blocks shown leads to the sum of products (a) $wyz + wx\bar z + w\bar x\,\bar y + \bar w\,\bar xy + \bar wx\bar yz$, (b) $\bar y\,\bar z + w\bar xy + \bar x\,\bar z$, and (c) $\bar z + \bar wx + w\bar xy$. 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

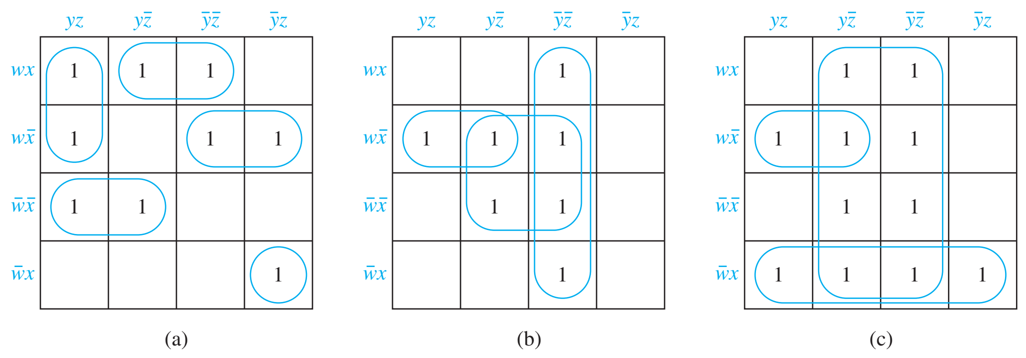


FIGURE 10 **Using K-maps in Four Variables.**

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 $\overline{x}_1 x_2 x_3 x_4 \overline{x}_5$.

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

Links

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

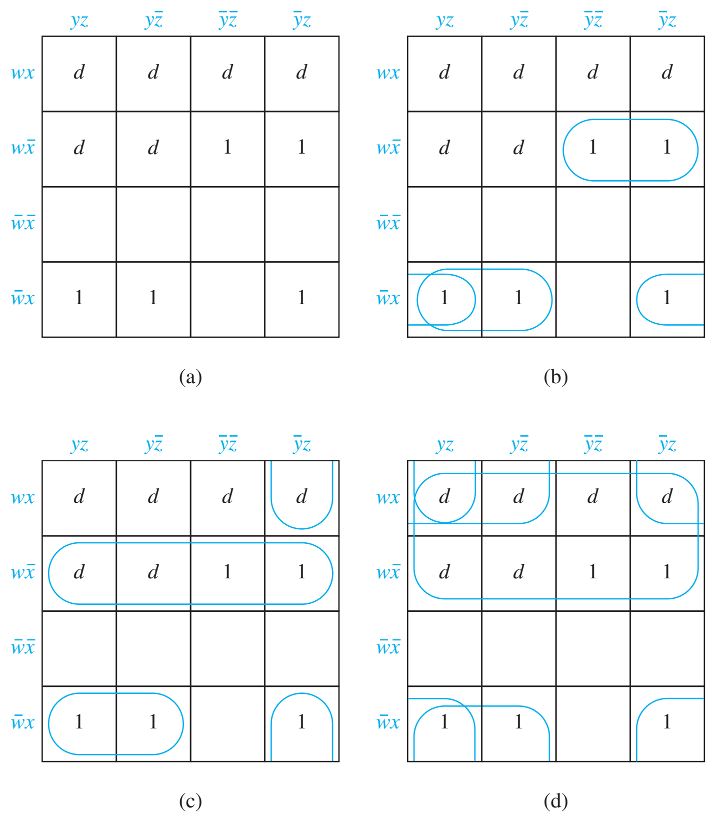
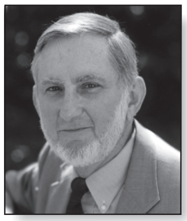


FIGURE 11 **The K-map for F Showing Its *Don't Care* Positions.**

shown in Figure 11(c), produces the expression $w\bar{x} + \bar{w}xy + x\bar{y}z$. 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



EDWARD J. McCLUSKEY (BORN 1929) Edward McCluskey attended Bowdoin College and M.I.T., where he received his doctorate in electrical engineering in 1956. He joined Bell Telephone Laboratories in 1955, remaining there until 1959. McCluskey was professor of electrical engineering at Princeton University from 1959 until 1966, also serving as director of the Computer Center at Princeton from 1961 to 1966. In 1967 he took a position as professor of computer science and electrical engineering at Stanford University, where he also served as director of the Digital Systems Laboratory from 1969 to 1978. McCluskey has worked in a variety of areas in computer science, including fault-tolerant computing, computer architecture, testing, and logic design. He is the director of the Center for Reliable Computing at Stanford University where he is now an emeritus professor. McCluskey is also an ACM Fellow.

TABLE 2

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

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

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

We will represent the minterms in this expansion by bit strings. The first bit will be 1 if x occurs and 0 if $\overline{x}$ occurs. The second bit will be 1 if y occurs and 0 if $\overline{y}$ occurs. The third bit will be 1 if z occurs and 0 if $\overline{z}$ 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 $x\overline{y}z$ and $\overline{x}\,\overline{y}z$, represented by bit strings 101 and 001, can be combined into $\overline{y}z$, 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 $\overline{y}z$, 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 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	$x\overline{y}z$	101	(1,3)	yz	–11			
3	$\overline{x}yz$	011	(2,4)	$\overline{y}z$	–01			
4	$\overline{x}\,\overline{y}z$	001	(3,4)	$\overline{x}z$	0–1			
5	$\overline{x}\,\overline{y}\,\overline{z}$	000	(4,5)	$\overline{x}\,\overline{y}$	00–			

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

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 $z + \overline{x}\,\overline{y}$. ◀

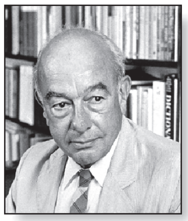
As was illustrated in Example 9, the Quine–McCluskey method uses this sequence of steps to simplify a sum-of-products expression.

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.

Links



WILLARD VAN ORMAN QUINE (1908–2000) Willard Quine, born in Akron, Ohio, attended Oberlin College and later Harvard University, where he received his Ph.D. in philosophy in 1932. He became a Junior Fellow at Harvard in 1933 and was appointed to a position on the faculty there in 1936. He remained at Harvard his entire professional life, except for World War II, when he worked for the U.S. Navy decrypting messages from German submarines. Quine was always interested in algorithms, but not in hardware. He arrived at his discovery of what is now called the Quine–McCluskey method as a device for teaching mathematical logic, rather than as a method for simplifying switching circuits. Quine was one of the most famous philosophers of the twentieth century. He made fundamental contributions to the theory of knowledge, mathematical logic and set theory, and the philosophies of logic and language. His books, including *New Foundations of Mathematical Logic* published in 1937 and *Word and Object* published in 1960, have had a profound impact. Quine retired from Harvard in 1978 but continued to commute from his home in Beacon Hill to his office there. He used the 1927 Remington typewriter on which he prepared his doctoral thesis for his entire life. He even had an operation performed on this machine to add a few special symbols, removing the second period, the second comma, and the question mark. When asked whether he missed the question mark, he replied, “Well, you see, I deal in certainties.” There is even a word *quine*, defined in the *New Hacker’s Dictionary* as a program that generates a copy of its own source code as its complete output. Producing the shortest possible quine in a given programming language is a popular puzzle for hackers.

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.

A final example will illustrate how this procedure is used to simplify a sum-of-products expansion in four variables.

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

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 $\overline{w}z$, $wy\overline{z}$, $w\overline{x}y$, and $\overline{x}yz$. In Table 7 we show the minterms covered by each of these products. To cover these minterms we must include $\overline{w}z$ and $wy\overline{z}$, because these products are the only products that cover $\overline{w}xyz$ and $wxy\overline{z}$, respectively. Once these two products are included, we see that only one of the two products left is needed. Consequently, we can take either $\overline{w}z + wy\overline{z} + w\overline{x}y$ or $\overline{w}z + wy\overline{z} + \overline{x}yz$ as the final answer. ◀

TABLE 5

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

TABLE 7

	$wxy\bar z$	$w\bar x yz$	$\bar w xyz$	$w\bar x y\bar z$	$\bar w x\bar y z$	$\bar w\bar x yz$	$\bar w\bar x\bar y z$
$\bar w z$			X		X	X	X
$wy\bar z$	X			X			
$w\bar x y$		X		X			
$\bar x yz$		X				X	

Exercises

1. a) Draw a K-map for a function in two variables and put a 1 in the cell representing $\bar x y$.

b) What are the minterms represented by cells adjacent to this cell?

2. Find the sum-of-products expansions represented by each of these K-maps.

a)

	y	$\bar y$
x	1	
$\bar x$	1	1

b)

	y	$\bar y$
x	1	1
$\bar x$		

c)

	y	$\bar y$
x	1	1
$\bar x$	1	1

3. Draw the K-maps of these sum-of-products expansions in two variables.

a) $x\bar y$ **b)** $xy + \bar x\bar y$

c) $xy + x\bar y + \bar x y + \bar x\bar y$

4. Use a K-map to find a minimal expansion as a Boolean sum of Boolean products of each of these functions of the Boolean variables x and y.

a) $\bar x y + \bar x\bar y$

b) $xy + x\bar y$

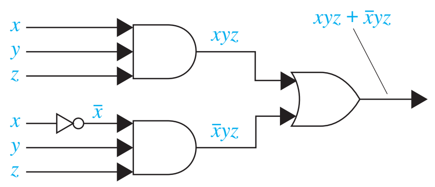
c) $xy + x\bar y + \bar x y + \bar x\bar y$

5. a) Draw a K-map for a function in three variables. Put a 1 in the cell that represents $\bar x y\bar z$.

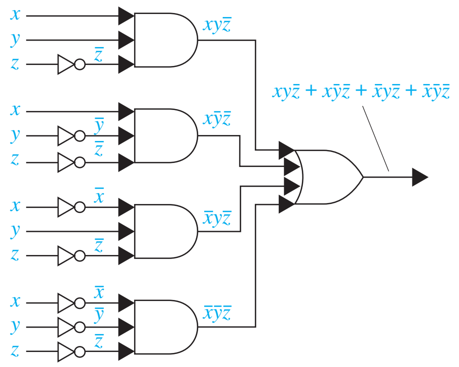
b) Which minterms are represented by cells adjacent to this cell?

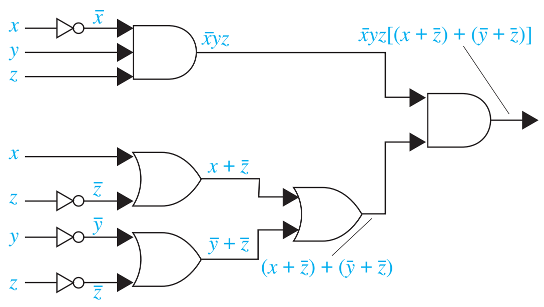
6. Use K-maps to find simpler circuits with the same output as each of the circuits shown.

a)



b)



c) 
x
y
z
$\overline{x}$
$\overline{x}yz$
$\overline{x}yz[(x+\overline{z})+(\overline{y}+\overline{z})]$
x
z
$\overline{z}$
$x+\overline{z}$
y
$\overline{y}$
$\overline{y}+\overline{z}$
z
$\overline{z}$
$(x+\overline{z})+(\overline{y}+\overline{z})$

7. Draw the K-maps of these sum-of-products expansions in three variables.

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

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

8. Construct a K-map for $F(x, y, z) = xz + yz + xy\overline{z}$. Use this K-map to find the implicants, prime implicants, and essential prime implicants of $F(x, y, z)$.

9. Construct a K-map for $F(x, y, z) = x\overline{z} + xyz + y\overline{z}$. Use this K-map to find the implicants, prime implicants, and essential prime implicants of $F(x, y, z)$.

10. Draw the 3-cube Q_3 and label each vertex with the minterm in the Boolean variables x, y, and z associated with the bit string represented by this vertex. For each literal in these variables indicate the 2-cube Q_2 that is a subgraph of Q_3 and represents this literal.

11. Draw the 4-cube Q_4 and label each vertex with the minterm in the Boolean variables w, x, y, and z associated with the bit string represented by this vertex. For each literal in these variables, indicate which 3-cube Q_3 that is a subgraph of Q_4 represents this literal. Indicate which 2-cube Q_2 that is a subgraph of Q_4 represents the products wz, $\overline{x}y$, and $\overline{y}\,\overline{z}$.

12. Use a K-map to find a minimal expansion as a Boolean sum of Boolean products of each of these functions in the variables x, y, and z.

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

b) $xyz + xy\overline{z} + \overline{x}yz + \overline{x}y\overline{z}$

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

d) $xyz + x\overline{y}z + x\overline{y}\,\overline{z} + \overline{x}yz + \overline{x}y\overline{z} + \overline{x}\,\overline{y}\,\overline{z}$

13. a) Draw a K-map for a function in four variables. Put a 1 in the cell that represents $\overline{w}xy\overline{z}$.

b) Which minterms are represented by cells adjacent to this cell?

14. Use a K-map to find a minimal expansion as a Boolean sum of Boolean products of each of these functions in the variables w, x, y, and z.

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

b) $wxy\overline{z} + wx\overline{y}z + w\overline{x}yz + \overline{w}x\overline{y}z + \overline{w}\,\overline{x}y\overline{z} + \overline{w}\,\overline{x}\,\overline{y}z$

c) $wxyz + wxy\overline{z} + wx\overline{y}z + w\overline{x}\,\overline{y}z + w\overline{x}\,\overline{y}\,\overline{z} + \overline{w}x\overline{y}z + \overline{w}\,\overline{x}y\overline{z} + \overline{w}\,\overline{x}\,\overline{y}z$

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

15. Find the cells in a K-map for Boolean functions with five variables that correspond to each of these products.

a) $x_1x_2x_3x_4$ **b)** $\overline{x}_1x_3x_5$ **c)** x_2x_4

d) $\overline{x}_3\overline{x}_4$ **e)** x_3 **f)** $\overline{x}_5$

16. How many cells in a K-map for Boolean functions with six variables are needed to represent x_1, $\overline{x}_1x_6$, $\overline{x}_1x_2\overline{x}_6$, $x_2x_3x_4x_5$, and $x_1\overline{x}_2x_4\overline{x}_5$, respectively?

17. a) How many cells does a K-map in six variables have?

b) How many cells are adjacent to a given cell in a K-map in six variables?

18. Show that cells in a K-map for Boolean functions in five variables represent minterms that differ in exactly one literal if and only if they are adjacent or are in cells that become adjacent when the top and bottom rows and 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 are considered adjacent.

19. Which rows and which columns of a 4×16 map for Boolean functions in six variables using the Gray codes 1111, 1110, 1010, 1011, 1001, 1000, 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101 to label the columns and 11, 10, 00, 01 to label the rows need to be considered adjacent so that cells that represent minterms that differ in exactly one literal are considered adjacent?

***20.** Use K-maps to find a minimal expansion as a Boolean sum of Boolean products of Boolean functions that have as input the binary code for each decimal digit and produce as output a 1 if and only if the digit corresponding to the input is

a) odd. **b)** not divisible by 3.

c) not 4, 5, or 6.

***21.** Suppose that there are five members on a committee, but that Smith and Jones always vote the opposite of Marcus. Design a circuit that implements majority voting of the committee using this relationship between votes.

22. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Example 3.

23. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Exercise 12.

24. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Example 4.

25. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Exercise 14.

***26.** Explain how K-maps can be used to simplify product-of-sums expansions in three variables. [*Hint:* Mark with a 0 all the maxterms in an expansion and combine blocks of maxterms.]

27. Use the method from Exercise 26 to simplify the product-of-sums expansion $(x + y + z)(x + y + \overline{z})(x + \overline{y} + \overline{z})(x + \overline{y} + z)(\overline{x} + y + z)$.

***28.** Draw a K-map for the 16 minterms in four Boolean variables on the surface of a torus.

29. Build a circuit using OR gates, AND gates, and inverters that produces an output of 1 if a decimal digit, encoded using a binary coded decimal expansion, is divisible by 3, and an output of 0 otherwise.

In Exercises 30–32 find a minimal sum-of-products expansion, given the K-map shown with *don't care* conditions indicated with *d*s.

30.

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx	d	1	d	1
$w\bar{x}$		d	d	
$\bar{w}\bar{x}$		d	1	
$\bar{w}x$		1	d	

31.

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx	1			1
$w\bar{x}$		d	1	
$\bar{w}\bar{x}$		1	d	
$\bar{w}x$	d			d

32.

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx		d	d	1
$w\bar{x}$	d	d	1	d
$\bar{w}\bar{x}$				
$\bar{w}x$	1	1	1	d

33. Show that products of k literals correspond to 2^{n-k}-dimensional subcubes of the n-cube Q_n, where the vertices of the cube correspond to the minterms represented by the bit strings labeling the vertices, as described in Example 8 of Section 10.2.

Key Terms and Results

TERMS

Boolean variable: a variable that assumes only the values 0 and 1

$\bar{x}$ **(complement of x):** an expression with the value 1 when x has the value 0 and the value 0 when x has the value 1

$x \cdot y$ (or xy) **(Boolean product** or **conjunction of x and y):** an expression with the value 1 when both x and y have the value 1 and the value 0 otherwise

$x + y$ **(Boolean sum** or **disjunction of x and y):** an expression with the value 1 when either x or y, or both, has the value 1, and 0 otherwise

Boolean expressions: the expressions obtained recursively by specifying that $0, 1, x_1, \ldots, x_n$ are Boolean expressions and $\overline{E}_1$, $(E_1 + E_2)$, and (E_1E_2) are Boolean expressions if E_1 and E_2 are

dual of a Boolean expression: the expression obtained by interchanging + signs and · signs and interchanging 0s and 1s

Boolean function of degree n: a function from B^n to B where $B = \{0, 1\}$

Boolean algebra: a set B with two binary operations $\vee$ and $\wedge$, elements 0 and 1, and a complementation operator $^-$ that satisfies the identity, complement, associative, commutative, and distributive laws

literal of the Boolean variable x: either x or $\bar{x}$

minterm of $x_1, x_2, \ldots, x_n$: a Boolean product $y_1y_2 \cdots y_n$, where each y_i is either x_i or $\bar{x}_i$

sum-of-products expansion (or **disjunctive normal form**): the representation of a Boolean function as a disjunction of minterms

functionally complete: a set of Boolean operators is called functionally complete if every Boolean function can be represented using these operators

$x \mid y$ (or x *NAND* y)**:** the expression that has the value 0 when both x and y have the value 1 and the value 1 otherwise

$x \downarrow y$ (or x *NOR* y)**:** the expression that has the value 0 when either x or y or both have the value 1 and the value 0 otherwise

inverter: a device that accepts the value of a Boolean variable as input and produces the complement of the input

OR gate: a device that accepts the values of two or more Boolean variables as input and produces their Boolean sum as output

AND gate: a device that accepts the values of two or more Boolean variables as input and produces their Boolean product as output

half adder: a circuit that adds two bits, producing a sum bit and a carry bit

full adder: a circuit that adds two bits and a carry, producing a sum bit and a carry bit

K-map for n variables: a rectangle divided into 2^n cells where each cell represents a minterm in the variables

minimization of a Boolean function: representing a Boolean function as the sum of the fewest products of literals such that these products contain the fewest literals possible among all sums of products that represent this Boolean function

implicant of a Boolean function: a product of literals with the property that if this product has the value 1, then the value of this Boolean function is 1

prime implicant of a Boolean function: a product of literals that is an implicant of the Boolean function and no product obtained by deleting a literal is also an implicant of this function

essential prime implicant of a Boolean function: a prime implicant of the Boolean function that must be included in a minimization of this function

don't care condition: a combination of input values for a circuit that is not possible or never occurs

RESULTS

The identities for Boolean algebra (see Table 5 in Section 12.1).