

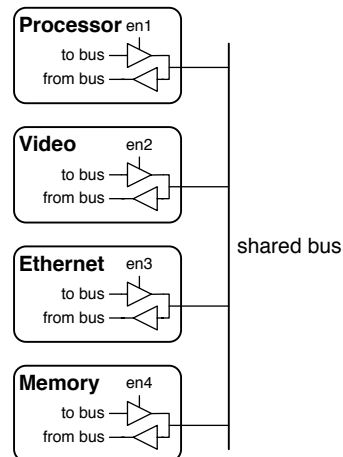


Figure 2.42 Tristate bus connecting multiple chips

the buffer is enabled. We show that the signal is active low by putting a bubble on its input wire. We often indicate an active low input by drawing a bar over its name, $\bar{E}$, or appending the word “bar” after its name, *Ebar*.

Tristate buffers are commonly used on *busses* that connect multiple chips. For example, a microprocessor, a video controller, and an Ethernet controller might all need to communicate with the memory system in a personal computer. Each chip can connect to a shared memory bus using tristate buffers, as shown in Figure 2.42. Only one chip at a time is allowed to assert its enable signal to drive a value onto the bus. The other chips must produce floating outputs so that they do not cause contention with the chip talking to the memory. Any chip can read the information from the shared bus at any time. Such tristate busses were once common. However, in modern computers, higher speeds are possible with *point-to-point links*, in which chips are connected to each other directly rather than over a shared bus.

2.7 KARNAUGH MAPS

After working through several minimizations of Boolean equations using Boolean algebra, you will realize that, if you’re not careful, you sometimes end up with a completely *different* equation instead of a simplified equation. *Karnaugh maps* (*K-maps*) are a graphical method for simplifying Boolean equations. They were invented in 1953 by Maurice Karnaugh, a telecommunications engineer at Bell Labs. K-maps work well for problems with up to four variables. More important, they give insight into manipulating Boolean equations.

Maurice Karnaugh, 1924–. Graduated with a bachelor’s degree in physics from the City College of New York in 1948 and earned a Ph.D. in physics from Yale in 1952. Worked at Bell Labs and IBM from 1952 to 1993 and as a computer science professor at the Polytechnic University of New York from 1980 to 1999.

Gray codes were patented (U.S. Patent 2,632,058) by Frank Gray, a Bell Labs researcher, in 1953. They are especially useful in mechanical encoders because a slight misalignment causes an error in only one bit.

Gray codes generalize to any number of bits. For example, a 3-bit Gray code sequence is:

000, 001, 011, 010,
110, 111, 101, 100

Lewis Carroll posed a related puzzle in *Vanity Fair* in 1879.

“The rules of the Puzzle are simple enough. Two words are proposed, of the same length; and the puzzle consists of linking these together by interposing other words, each of which shall differ from the next word in one letter only. That is to say, one letter may be changed in one of the given words, then one letter in the word so obtained, and so on, till we arrive at the other given word.”

For example, SHIP to DOCK:

SHIP, SLIP, SLOP,
SLOT, SOOT, LOOT,
LOOK, LOCK, DOCK.

Can you find a shorter sequence?

Recall that logic minimization involves combining terms. Two terms containing an implicant, P , and the true and complementary forms of some variable, A , are combined to eliminate A : $PA + P\bar{A} = P$. Karnaugh maps make these combinable terms easy to see by putting them next to each other in a grid.

Figure 2.43 shows the truth table and K-map for a three-input function. The top row of the K-map gives the four possible values for the A and B inputs. The left column gives the two possible values for the C input. Each square in the K-map corresponds to a row in the truth table and contains the value of the output, Y , for that row. For example, the top left square corresponds to the first row in the truth table and indicates that the output value $Y = 1$ when $ABC = 000$. Just like each row in a truth table, each square in a K-map represents a single minterm. For the purpose of explanation, Figure 2.43(c) shows the minterm corresponding to each square in the K-map.

Each square, or minterm, differs from an adjacent square by a change in a single variable. This means that adjacent squares share all the same literals except one, which appears in true form in one square and in complementary form in the other. For example, the squares representing the minterms $\bar{A}\bar{B}\bar{C}$ and $\bar{A}\bar{B}C$ are adjacent and differ only in the variable C . You may have noticed that the A and B combinations in the top row are in a peculiar order: 00, 01, 11, 10. This order is called a *Gray code*. It differs from ordinary binary order (00, 01, 10, 11) in that adjacent entries differ only in a single variable. For example, 01 : 11 only changes A from 0 to 1, while 01 : 10 would change A from 1 to 0 and B from 0 to 1. Hence, writing the combinations in binary order would not have produced our desired property of adjacent squares differing only in one variable.

The K-map also “wraps around.” The squares on the far right are effectively adjacent to the squares on the far left, in that they differ only in one variable, A . In other words, you could take the map and roll it into a cylinder, then join the ends of the cylinder to form a torus (i.e., a donut), and still guarantee that adjacent squares would differ only in one variable.

2.7.1 Circular Thinking

In the K-map in Figure 2.43, only two minterms are present in the equation, $\bar{A}\bar{B}\bar{C}$ and $\bar{A}\bar{B}C$, as indicated by the 1's in the left column. Reading the minterms from the K-map is exactly equivalent to reading equations in sum-of-products form directly from the truth table.

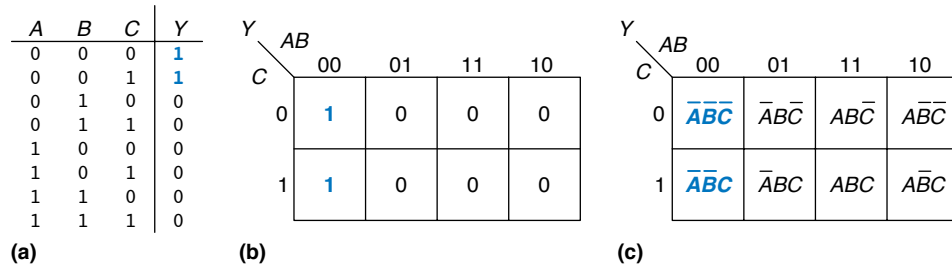


Figure 2.43 Three-input function: (a) truth table, (b) K-map, (c) K-map showing minterms

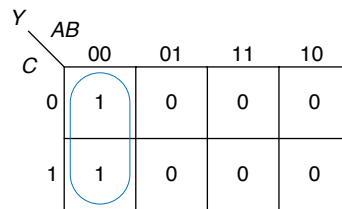


Figure 2.44 K-map minimization

As before, we can use Boolean algebra to minimize equations in sum-of-products form.

$$Y = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C = \overline{A}\overline{B}(\overline{C} + C) = \overline{A}\overline{B} \quad (2.6)$$

K-maps help us do this simplification graphically by *circling* 1's in adjacent squares, as shown in Figure 2.44. For each circle, we write the corresponding implicant. Remember from Section 2.2 that an implicant is the product of one or more literals. Variables whose true *and* complementary forms are both in the circle are excluded from the implicant. In this case, the variable C has both its true form (1) and its complementary form (0) in the circle, so we do not include it in the implicant. In other words, Y is TRUE when $A = B = 0$, independent of C . So the implicant is $\overline{A}\overline{B}$. This K-map gives the same answer we reached using Boolean algebra.

2.7.2 Logic Minimization with K-Maps

K-maps provide an easy visual way to minimize logic. Simply circle all the rectangular blocks of 1's in the map, using the fewest possible number of circles. Each circle should be as large as possible. Then read off the implicants that were circled.

More formally, recall that a Boolean equation is minimized when it is written as a sum of the fewest number of prime implicants. Each circle on the K-map represents an implicant. The largest possible circles are prime implicants.

For example, in the K-map of Figure 2.44, $\overline{A}\overline{B}\overline{C}$ and $\overline{A}\overline{B}C$ are implicants, but *not* prime implicants. Only $\overline{A}\overline{B}$ is a prime implicant in that K-map. Rules for finding a minimized equation from a K-map are as follows:

- ▶ Use the fewest circles necessary to cover all the 1's.
- ▶ All the squares in each circle must contain 1's.
- ▶ Each circle must span a rectangular block that is a power of 2 (i.e., 1, 2, or 4) squares in each direction.
- ▶ Each circle should be as large as possible.
- ▶ A circle may wrap around the edges of the K-map.
- ▶ A 1 in a K-map may be circled multiple times if doing so allows fewer circles to be used.

Example 2.9 MINIMIZATION OF A THREE-VARIABLE FUNCTION USING A K-MAP

Suppose we have the function $Y = F(A, B, C)$ with the K-map shown in Figure 2.45. Minimize the equation using the K-map.

Solution: Circle the 1's in the K-map using as few circles as possible, as shown in Figure 2.46. Each circle in the K-map represents a prime implicant, and the dimension of each circle is a power of two (2×1 and 2×2). We form the prime implicant for each circle by writing those variables that appear in the circle only in true or only in complementary form.

For example, in the 2×1 circle, the true and complementary forms of B are included in the circle, so we *do not* include B in the prime implicant. However, only the true form of A (A) and complementary form of C ($\overline{C}$) are in this circle, so we include these variables in the prime implicant $A\overline{C}$. Similarly, the 2×2 circle covers all squares where $B = 0$, so the prime implicant is $\overline{B}$.

Notice how the top-right square (minterm) is covered twice to make the prime implicant circles as large as possible. As we saw with Boolean algebra techniques, this is equivalent to sharing a minterm to reduce the size of the

Figure 2.45 K-map for Example 2.9

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

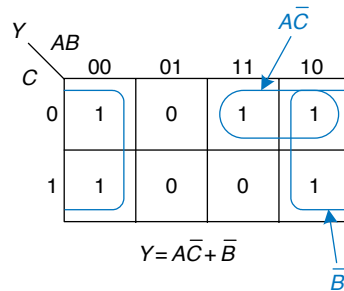


Figure 2.46 Solution for Example 2.9

implicant. Also notice how the circle covering four squares wraps around the sides of the K-map.

Example 2.10 SEVEN-SEGMENT DISPLAY DECODER

A *seven-segment display decoder* takes a 4-bit data input, $D_{3:0}$, and produces seven outputs to control light-emitting diodes to display a digit from 0 to 9. The seven outputs are often called segments a through g , or S_a – S_g , as defined in Figure 2.47. The digits are shown in Figure 2.48. Write a truth table for the outputs, and use K-maps to find Boolean equations for outputs S_a and S_b . Assume that illegal input values (10–15) produce a blank readout.

Solution: The truth table is given in Table 2.6. For example, an input of 0000 should turn on all segments except S_g .

Each of the seven outputs is an independent function of four variables. The K-maps for outputs S_a and S_b are shown in Figure 2.49. Remember that adjacent squares may differ in only a single variable, so we label the rows and columns in Gray code order: 00, 01, 11, 10. Be careful to also remember this ordering when *entering* the output values into the squares.

Next, circle the prime implicants. Use the fewest number of circles necessary to cover all the 1's. A circle can wrap around the edges (vertical *and* horizontal), and a 1 may be circled more than once. Figure 2.50 shows the prime implicants and the simplified Boolean equations.

Note that the minimal set of prime implicants is not unique. For example, the 0000 entry in the S_a K-map was circled along with the 1000 entry to produce the $\bar{D}_2\bar{D}_1\bar{D}_0$ minterm. The circle could have included the 0010 entry instead, producing a $\bar{D}_3\bar{D}_2\bar{D}_0$ minterm, as shown with dashed lines in Figure 2.51.

Figure 2.52 illustrates (see page 78) a common error in which a nonprime implicant was chosen to cover the 1 in the upper left corner. This minterm, $\bar{D}_3\bar{D}_2\bar{D}_1\bar{D}_0$, gives a sum-of-products equation that is *not* minimal. The minterm could have been combined with either of the adjacent ones to form a larger circle, as was done in the previous two figures.

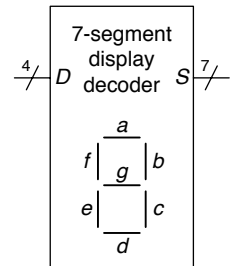
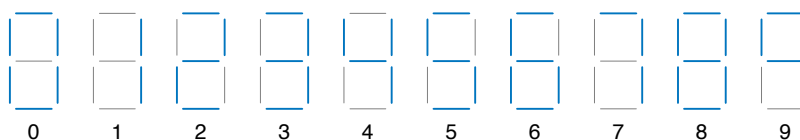


Figure 2.47 Seven-segment display decoder icon

Figure 2.48 Seven-segment display digits**Table 2.6** Seven-segment display decoder truth table

$D_{3:0}$	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	0	0	1	1
others	0	0	0	0	0	0	0

Figure 2.49 Karnaugh maps for S_a and S_b

S_a	$D_{1:0}$	$D_{3:2}$			
		00	01	11	10
	00	1	0	0	1
	01	0	1	0	1
	11	1	1	0	0
	10	1	1	0	0

S_b	$D_{1:0}$	$D_{3:2}$			
		00	01	11	10
	00	1	1	0	1
	01	1	0	0	1
	11	1	1	0	0
	10	1	0	0	0

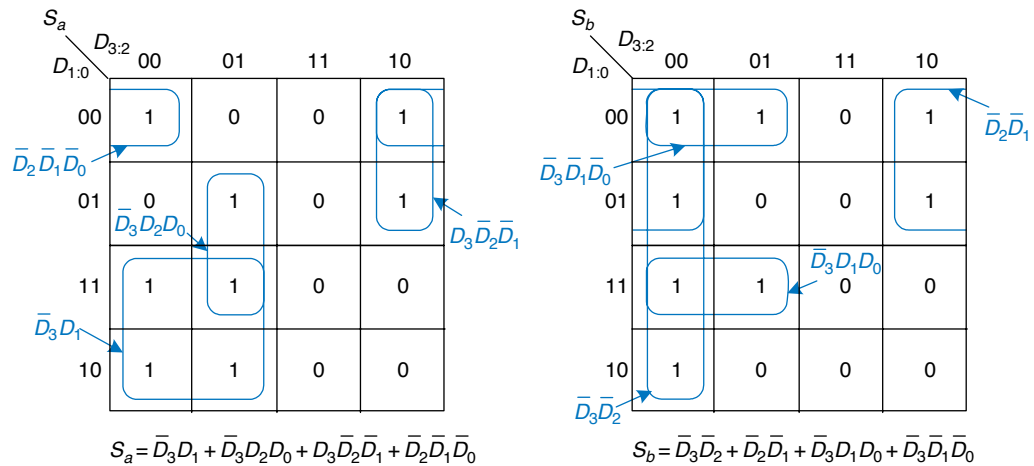
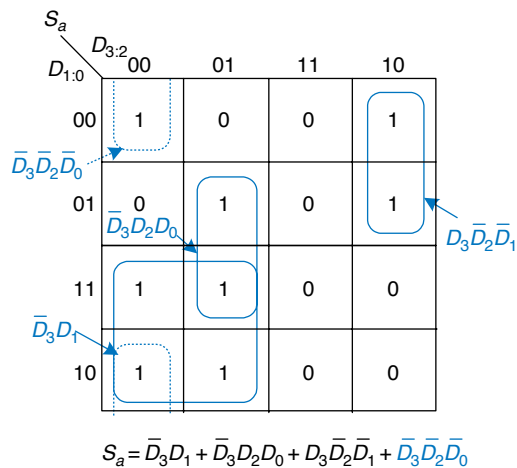


Figure 2.50 K-map solution for Example 2.10

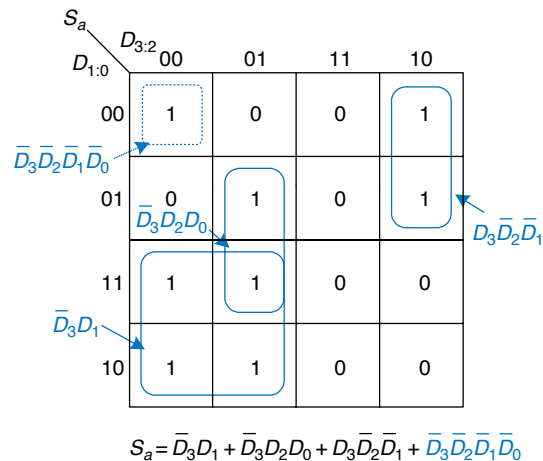
Figure 2.51 Alternative K-map for S_a showing different set of prime implicants

2.7.3 Don't Cares

Recall that “don’t care” entries for truth table inputs were introduced in Section 2.4 to reduce the number of rows in the table when some variables do not affect the output. They are indicated by the symbol X, which means that the entry can be either 0 or 1.

Don’t cares also appear in truth table outputs where the output value is unimportant or the corresponding input combination can never happen. Such outputs can be treated as either 0’s or 1’s at the designer’s discretion.

Figure 2.52 Alternative K-map for S_a showing incorrect nonprime implicant



In a K-map, X's allow for even more logic minimization. They can be circled if they help cover the 1's with fewer or larger circles, but they do not have to be circled if they are not helpful.

Example 2.11 SEVEN-SEGMENT DISPLAY DECODER WITH DON'T CARES

Repeat Example 2.10 if we don't care about the output values for illegal input values of 10 to 15.

Solution: The K-map is shown in Figure 2.53 with X entries representing don't care. Because don't cares can be 0 or 1, we circle a don't care if it allows us to cover the 1's with fewer or bigger circles. Circled don't cares are treated as 1's, whereas uncircled don't cares are 0's. Observe how a 2×2 square wrapping around all four corners is circled for segment S_a . Use of don't cares simplifies the logic substantially.

2.7.4 The Big Picture

Boolean algebra and Karnaugh maps are two methods for logic simplification. Ultimately, the goal is to find a low-cost method of implementing a particular logic function.

In modern engineering practice, computer programs called *logic synthesizers* produce simplified circuits from a description of the logic function, as we will see in Chapter 4. For large problems, logic synthesizers are much more efficient than humans. For small problems, a human with a bit of experience can find a good solution by inspection. Neither of the authors has ever used a Karnaugh map in real life to

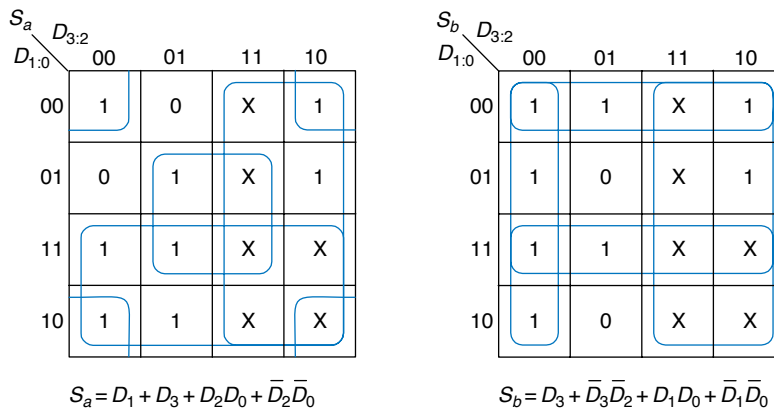


Figure 2.53 K-map solution with don't cares

solve a practical problem. But the insight gained from the principles underlying Karnaugh maps is valuable. And Karnaugh maps often appear at job interviews!

2.8 COMBINATIONAL BUILDING BLOCKS

Combinational logic is often grouped into larger building blocks to build more complex systems. This is an application of the principle of abstraction, hiding the unnecessary gate-level details to emphasize the function of the building block. We have already studied three such building blocks: full adders (from Section 2.1), priority circuits (from Section 2.4), and seven-segment display decoders (from Section 2.7). This section introduces two more commonly used building blocks: multiplexers and decoders. Chapter 5 covers other combinational building blocks.

2.8.1 Multiplexers

Multiplexers are among the most commonly used combinational circuits. They choose an output from among several possible inputs based on the value of a *select* signal. A multiplexer is sometimes affectionately called a *mux*.

2:1 Multiplexer

Figure 2.54 shows the schematic and truth table for a 2:1 multiplexer with two data inputs, D_0 and D_1 , a select input, S , and one output, Y . The multiplexer chooses between the two data inputs based on the select: if $S = 0$, $Y = D_0$, and if $S = 1$, $Y = D_1$. S is also called a *control signal* because it controls what the multiplexer does.

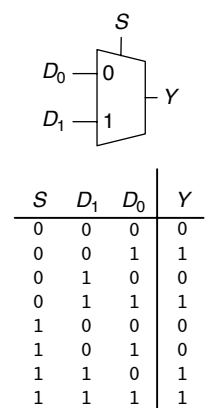


Figure 2.54 2:1 multiplexer symbol and truth table