only. By way of illustration, the value of Z_3 in the 2-bit multiplier can be converted to NOR logic form in the following way

$$\begin{aligned} Z_3 &= X_1 \cdot X_0 \cdot Y_1 \cdot Y_0 \\ &= \overline{\overline{X_1 \cdot X_0 \cdot Y_1 \cdot Y_0}} \\ &= \overline{\overline{X}_1 + \overline{X}_0 + \overline{Y}_1 + \overline{Y}_0} \end{aligned}$$

Note that negation may be implemented by an inverter or by a NOR gate with its inputs connected together.

As a final example of NAND logic consider Fig. 2.56. A Boolean expression can be expressed in sum-of-products form as $A \cdot B + C \cdot D$. This expression can be converted to NAND logic as

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

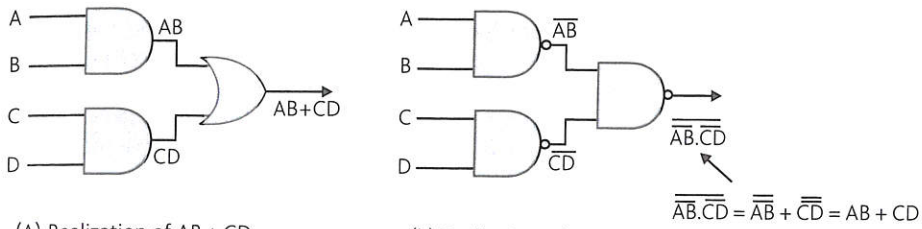
Note how the three-gate circuit in Fig. 2.56(a) can be converted into the three-gate NAND circuit of Fig. 2.56(b).

Fig. 2.57 shows the construction of the two versions of $AB + CD$ in Digital Works. We have provided an LED at each output and manually selectable inputs to enable you to investigate the circuits.

2.5.4 Karnaugh maps

When you use algebraic techniques to simplify a Boolean expression you sometimes reach a point at which you can't proceed, because you're unable to find further simplifications. The *Karnaugh map*, or more simply the *K-map*, is a graphical technique for the representation and simplification of a Boolean expression that shows unambiguously when a Boolean expression has been reduced to its most simple form.

Although the Karnaugh map can simplify Boolean equations with five or six variables, we will use it to solve problems



(A) Realization of AB + CD (AND/OR logic).

(b) Realization of AB + CD (NAND logic).

Figure 2.56 Implementing $A \cdot B + C \cdot D$ in AND/OR and NAND logic.

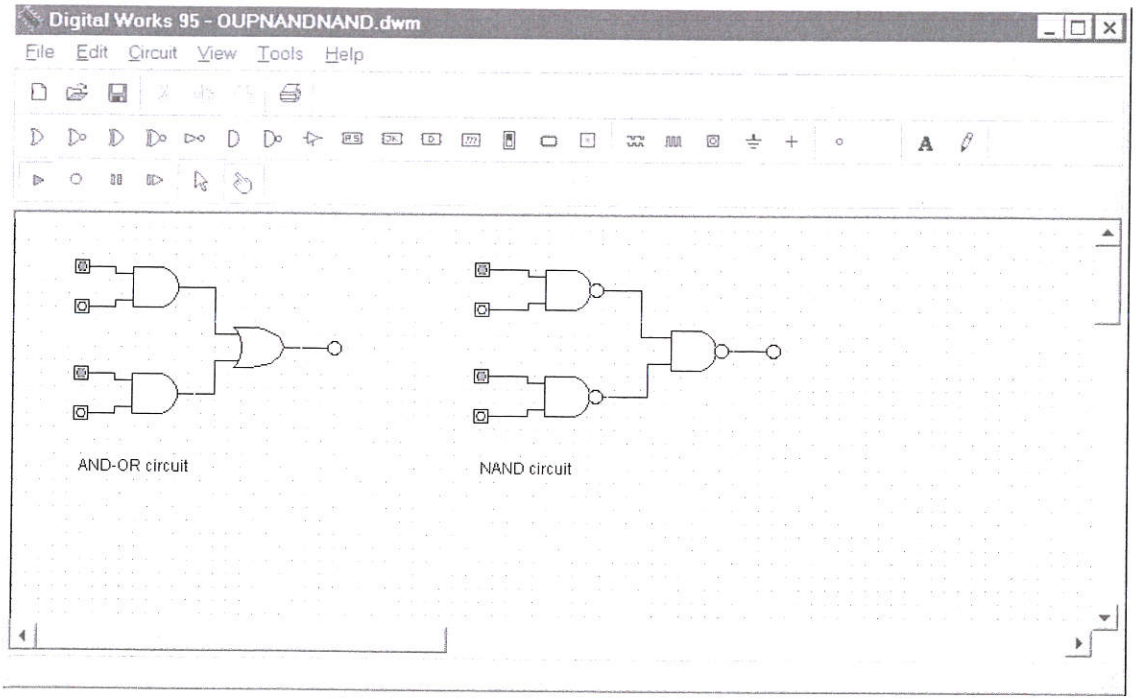


Figure 2.57 Using Digital Works to investigate two circuits.

EXAMPLE

Show that the exclusive or, EOR, operator is associative, so that $A \oplus (B \oplus C) = (A \oplus B) \oplus C$.

$$\begin{aligned} A \oplus (B \oplus C) &= A \oplus (\overline{B}\cdot C + B\cdot\overline{C}) \\ &= A\overline{(\overline{B}\cdot C + B\cdot\overline{C})} + \overline{A}(\overline{B}\cdot C + B\cdot\overline{C}) \\ &= A(B + \overline{C})(\overline{B} + C) + \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} \\ &= A(B\cdot C + \overline{B}\cdot\overline{C}) + \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} \\ &= A\cdot B\cdot C + A\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} \end{aligned}$$

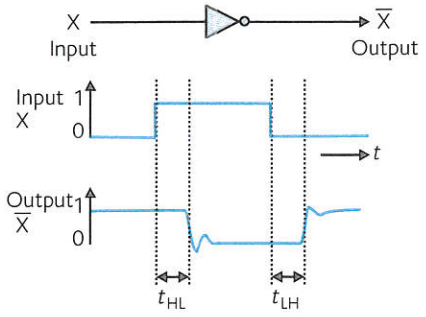
$$\begin{aligned} (A\oplus B)\oplus C &= (\overline{A}\cdot B + A\cdot\overline{B})\oplus C \\ &= (\overline{A}\cdot B + A\cdot\overline{B})\overline{C} + \overline{(\overline{A}\cdot B + A\cdot\overline{B})}C \\ &= \overline{A}\cdot B\cdot\overline{C} + A\cdot\overline{B}\cdot\overline{C} + (\overline{\overline{A}\cdot B}\cdot\overline{A\cdot\overline{B}})C \\ &= \overline{A}\cdot B\cdot\overline{C} + A\cdot\overline{B}\cdot\overline{C} + (A + \overline{B})\cdot(\overline{A} + B)C \\ &= \overline{A}\cdot B\cdot\overline{C} + A\cdot\overline{B}\cdot\overline{C} + (A\cdot B + \overline{A}\cdot\overline{B})C \end{aligned}$$

Both these expressions are equal and therefore the $\oplus$ operator is associative.

EFFECT OF FINITE PROPAGATION DELAYS ON LOGIC ELEMENTS

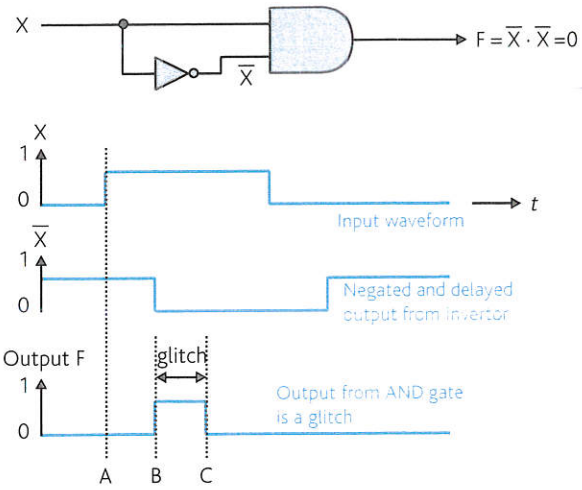
We have assumed that if signals are applied to the input terminals of a circuit, the correct output will appear instantaneously at the output of the circuit. In practice, this is not so. Real gates suffer from an effect called propagation delay and it takes about 1 ns for a change in an input signal to affect the output. One nanosecond is an unbelievably short period of time in human terms—but not in electronic terms. The speed of light is 300×10^8 cm/s and electrical signals in computers travel at about 70% of the speed of light. In 1 ns a signal travels about 20 cm.

The propagation delay introduced by logic elements is one of the greatest problems designers have to contend with. The diagram illustrates the effect of propagation delay on a single inverter where a pulse with sharp (i.e. vertical) rising and falling edges is applied to the input of an inverter. An inverted pulse is produced at its output and is delayed with respect to the input pulse. Moreover, the edges of the output pulse are no longer vertical. The time tHL represents the time delay between the rising edge of the input pulse and the point at which the output of the gate has reached V_{OL}. Similarly, t_{LH} represents the time between the falling edge of the input and the time at which the output reaches V_{OH}.



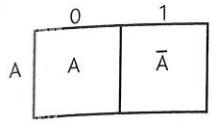
You might think that the effect of time delays on the passage of signals through gates simply reduces the speed at which a digital system may operate. Unfortunately, propagation delays have more sinister effects as demonstrated by the diagram. By the rules of Boolean algebra the output of the AND gate is $X \cdot X$ and should be permanently 0. Now examine its timing diagram.

At point A the input, X, rises from 0 to 1. However, the $\overline{X}$ input to the AND gate does not fall to 0 for a time equal to the propagation delay of the inverter. Consequently, for a short time the inputs of the AND gate are both true, and its output rises to a logical 1 from points B to C (after its own internal delay). The short pulse at the output of the AND gate is called a *glitch*, and can be very troublesome in digital systems. There are two solutions to this problem. One is to apply special design techniques to the Boolean logic to remove the glitch. The other is to connect the output to a flip-flop, and to clock the flip-flop after any glitches have died away.

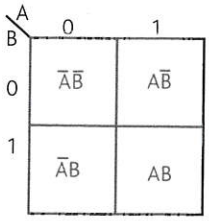


with only three or four variables. Other techniques such as the *Quine–McCluskey* method can be applied to the simplification of Boolean expressions in more than six variables. However, these techniques are beyond the scope of this book.

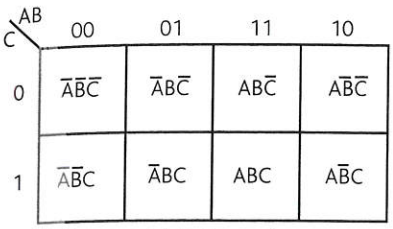
The Karnaugh map is just a two-dimensional form of the truth table, drawn in such a way that the simplification of a Boolean expression can immediately be seen from the location of 1s on the map. A system with n variables has 2^n lines in its truth table and 2^n squares on its Karnaugh map. Each square on the Karnaugh map is associated with a line (i.e. *minterm*) in the truth table. Figure 2.58 shows Karnaugh maps for one to four variables.



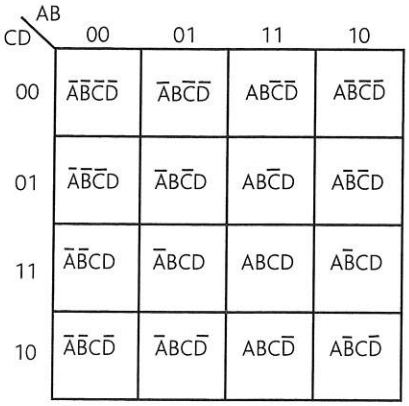
(a) One-variable Karnaugh map.



(b) Two-variable Karnaugh map.

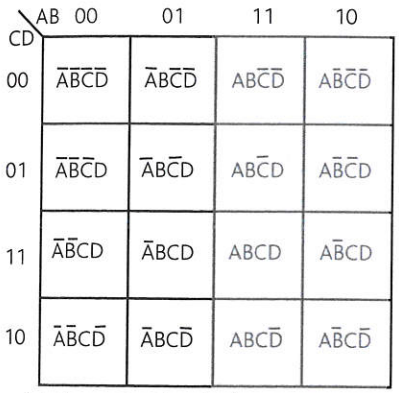


(c) Three-variable Karnaugh map.

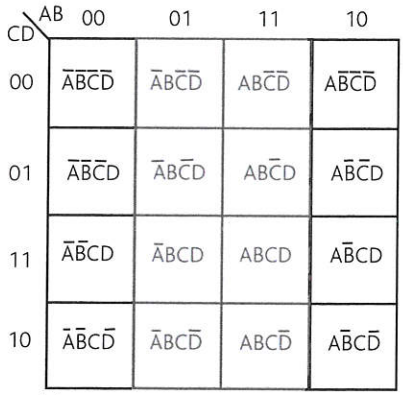


(b) Four-variable Karnaugh map.

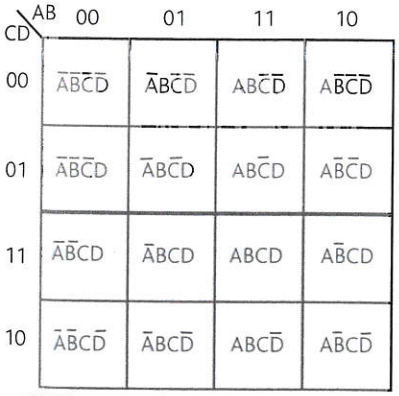
Figure 2.58 The Karnaugh map.



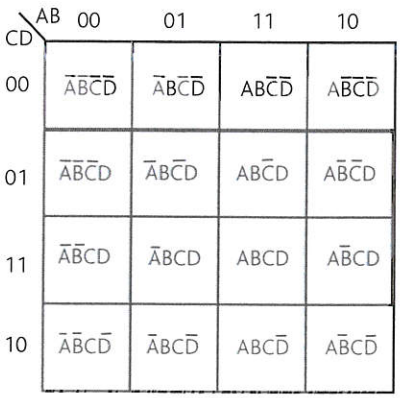
(a) The region for which A is true.



(b) The region for which B is true.



(c) The region for which C is true.



(d) The region for which D is true.

Figure 2.59 Regions of a Karnaugh map.

As you can see from Fig. 2.58, each line in a truth table is mapped onto a Karnaugh map; for example, in four variables each logical combination from $A \cdot B \cdot C \cdot D$ to $\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$ has a unique location. However, the key to the Karnaugh map is the *layout* of the squares. Adjacent squares differ by only one variable. By *adjacent* we mean horizontally and vertically adjacent, but not diagonally adjacent. For example, if you look the three-variable map of Fig. 2.58(c) you will see that the leftmost two terms on the top line are $\overline{A} \cdot \overline{B} \cdot \overline{C}$ and $\overline{A} \cdot B \cdot \overline{C}$. The only difference between these terms is B and $\overline{B}$.

Figure 2.59 demonstrates the structure of a four-variable Karnaugh map with variables A, B, C, and D. This map has been repeated four times and, in each case, the region in which the selected variable is true has been shaded. The unshaded portion of each map represents the region in which the chosen variable is false.

We will soon see that you need to develop three skills to use a Karnaugh map. The first is to plot terms on the map (i.e. transfer a truth table or a Boolean

expression onto the map). The second skill is the ability to group the 1s you've plotted on the map. The third skill is to read the groups of 1s on the map and express each group as a product term.

We now use a simple three-variable map to demonstrate how a truth table is mapped onto a Karnaugh map. One-and two-variable maps represent trivial cases and aren't considered further. Figure 2.60 shows the truth table for a three-variable function and the corresponding Karnaugh map. Each of the three 1s in the truth table is mapped onto its appropriate square on the Karnaugh map.

A three-variable Karnaugh map has four vertical columns, one for each of the four possible values of two out of the three variables. For example, if the three variables are A, B, and C, the four columns represent all the combinations of A and B. The leftmost column is labeled 00 and represents the region for which A = 0, B = 0. The next column is labeled 01, and represents the region for which A = 0, B = 1. The next column is labeled 11 (not 10), and represents the region for which A = 1, B = 1. Remember that adjacent columns differ by only one variable at a time. The fourth column, 10, represents the region for which A = 1, B = 0. In fact, a Karnaugh map is made up of all possible 2^n *minterms* for a system with *n* variables.

The three-variable Karnaugh map in Fig. 2.60 has two horizontal rows, the upper row corresponding to C = 0 and the lower to C = 1. Any square on this Karnaugh map represents a unique combination of the three variables, from $A \cdot B \cdot C$ to $\overline{A} \cdot \overline{B} \cdot \overline{C}$.

Figure 2.60 demonstrates how a function of three variables, $F = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C$ is plotted on a Karnaugh map. If it isn't clear how the entries in the table are plotted on the Karnaugh map, examine Fig. 2.60 and work out which cell on the map is associated with each line in the table. A square containing a logical 1 is said to be covered by a 1.

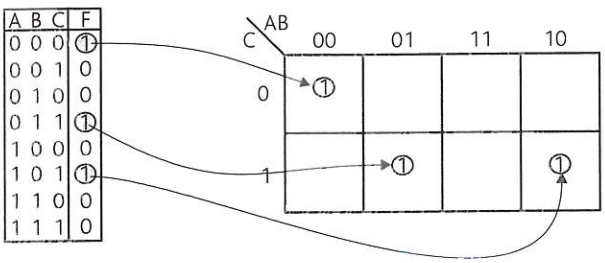


Figure 2.60 Relationship between a Karnaugh map and truth table.

At this point it's worth noting that no two 1s plotted on the Karnaugh map of Fig. 2.60 are adjacent to each other, and that the function $F = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C$ cannot be simplified. To keep the Karnaugh maps as clear and uncluttered as possible, squares that do not contain a 1 are left unmarked even though they must, of course, contain a 0.

Consider Fig. 2.61 in which the function $F_1 = A \cdot B \cdot \overline{C} + A \cdot B \cdot C$ is plotted on the left-hand map. The two minterms in this function are $A \cdot B \cdot \overline{C}$ and $A \cdot B \cdot C$ and occupy the cells for which A = 1, B = 1, C = 0, and A = 1, B = 1, C = 1, respectively. If you still have difficulty plotting minterms, just think of them as coordinates of squares; for example, $A \cdot B \cdot \overline{C}$ has the coordinates 1,1,0 and corresponds to the square ABC = 110.

In the Karnaugh map for F_1 two separate adjacent squares are covered. Now look at the Karnaugh map for $F_2 = A \cdot B$ at the right-hand side of Fig. 2.61. In this case a *group* of two squares is covered, corresponding to the column A = 1, B = 1. As the function for F_2 does not involve the variable C, a 1 is entered in the squares for which A = B = 1 and C = 0, and A = B = 1 and C = 1; that is, a 1 is entered for all values of C for which AB = 11. When plotting a product term like $A \cdot B$ on the Karnaugh map, all you have to do is to locate the region for which AB = 11.

It is immediately obvious that both Karnaugh maps in Fig. 2.61 are identical, so that $F_1 = F_2$ and $A \cdot B \cdot C + A \cdot B \cdot \overline{C} = A \cdot B$. From the rules of Boolean algebra $A \cdot B \cdot C + A \cdot B \cdot \overline{C} = A \cdot B\,(C + \overline{C}) = A \cdot B(1) = A \cdot B$. It should be apparent that two adjacent squares in a Karnaugh map can be grouped together to form a single simpler term. It is this property that the Karnaugh map exploits to simplify expressions.

Simplifying Sum-of-Product expressions with a Karnaugh map

The first step in simplifying a Boolean expression by means of a Karnaugh map is to plot all the 1s (i.e. minterms) in the function's truth table on the Karnaugh map. The next step is to combine adjacent 1s into groups of one, two, four, eight, or

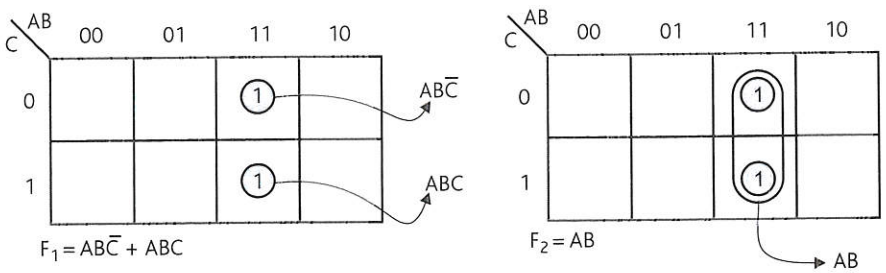


Figure 2.61 Plotting two functions on Karnaugh maps.

16. The groups of minterms should be as large as possible—a single group of four minterms yields a simpler expression than two groups of two minterms. The final stage in simplifying an expression is reached when each of the groups of minterms (i.e. the product terms) are ORed together to form the simplified sum-of-products expression. This process is best demonstrated by means of examples. In what follows, a four-variable map is chosen to illustrate the examples.

Transferring a truth table to a Karnaugh map is easy because each 1 in the truth table is placed in a unique square on the map. We now have to demonstrate how the product terms of a general Boolean expression are plotted on the map. Figures 2.62–2.67 present six functions plotted on Karnaugh maps. In these diagrams various *sum-of-products* expressions have been plotted directly from the equations themselves, rather than from the minterms of the truth table. The following notes should help in understanding these diagrams.

1. For a four-variable Karnaugh map

   one-variable product term covers 8 squares
   two-variable product terms cover 4 squares
   three-variable product terms cover 2 squares
   four-variable product terms cover 1 square.

2. A square covered by a 1 may belong to more than one term in the sum-of-products expression. For example, in Fig. 2.63 the minterm $\overline{A} \cdot \overline{B} \cdot C \cdot D$ belongs to two groups, $\overline{A} \cdot \overline{B}$ and $C \cdot D$. If a 1 on the Karnaugh map appears in two groups, it is equivalent to adding the corresponding minterm to the overall expression for the function plotted on the map *twice*. Repeating a term in a Boolean expression does not alter the value of the expression, because one of the axioms of Boolean algebra is $X + X = X$.

3. The Karnaugh map is not a square or a rectangle as it appears in these diagrams. A Karnaugh map is a *torus* or *doughnut* shape. That is, the top edge is adjacent to the bottom edge and, the left-hand edge is adjacent to the right-hand edge. For example, in Figure 2.65 the term $\overline{A} \cdot \overline{D}$ covers the two minterms $\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$ and $\overline{A} \cdot B \cdot \overline{C} \cdot \overline{D}$ at the top, and the two minterms $\overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$ and $\overline{A} \cdot B \cdot C \cdot \overline{D}$ at the bottom of the map. Similarly, in Fig. 2.66 the term $\overline{B} \cdot \overline{D}$ covers all four corners of the map. Whenever a group

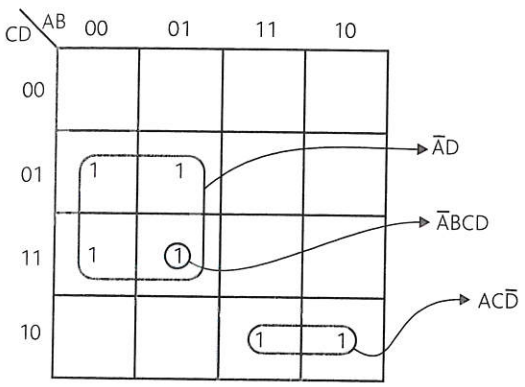


Figure 2.62 Plotting $F = \overline{A}D + AC\overline{D} + \overline{A}BCD$ on a Karnaugh map.

The two-variable term $\overline{A} \cdot D$ covers four squares (the region A = 0 and D = 1). The term $\overline{A} \cdot B \cdot C \cdot D$ covers one square and is part of the same group as $\overline{A} \cdot D$.

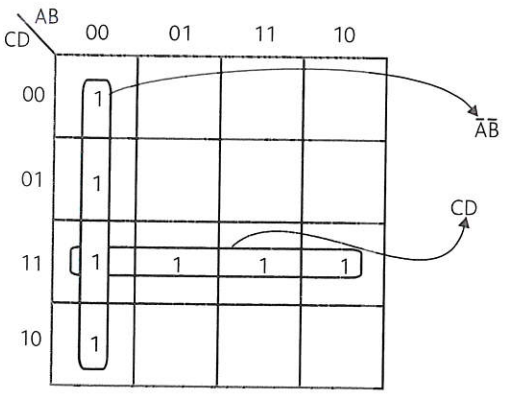
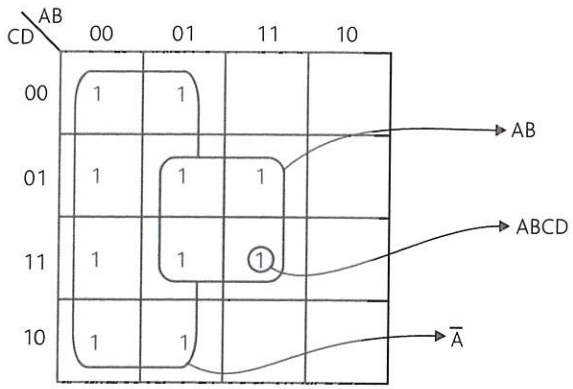


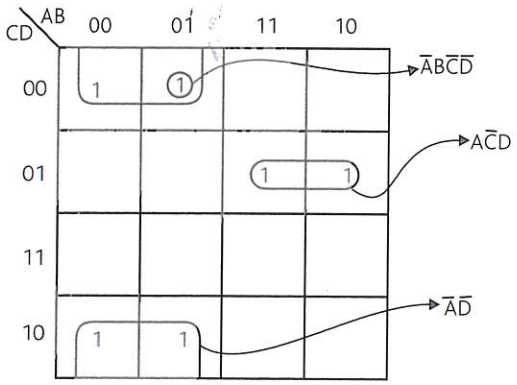
Figure 2.63 Plotting $F = AB + CD$ on a Karnaugh map.

The two-variable term $\overline{A} \cdot \overline{B}$ covers four squares (the region A = 0 and B = 0). The two-variable term C · D covers four squares (the region C = 1 and D = 1). The term $\overline{A} \cdot \overline{B} \cdot C \cdot D$ is common to both groups.



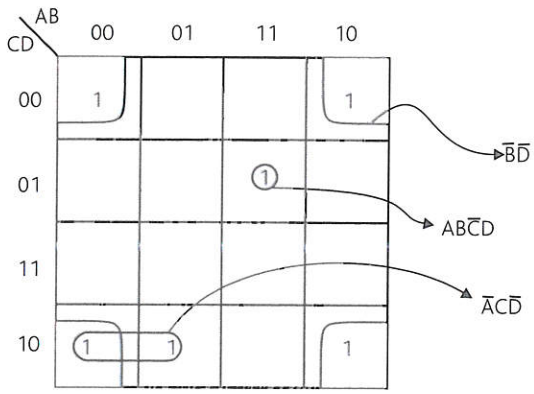
The one-variable term $\overline{A}$ covers four squares (the region A = 0).

Figure 2.64 Plotting $F = \overline{A} + BD + ABCD$ on a Karnaugh map.



The four-variable term $\overline{A} \cdot \overline{D}$ covers four squares (the region A = 0, D = 0). Note that two squares are at the top (A = 0, C = 0, D = 0) and two are at the bottom (A = 0, C = 1, D = 0).

Figure 2.65 Plotting $F = \overline{A}\,\overline{D} + A\overline{C}D + \overline{A}B\overline{C}\,\overline{D}$ on a Karnaugh map.



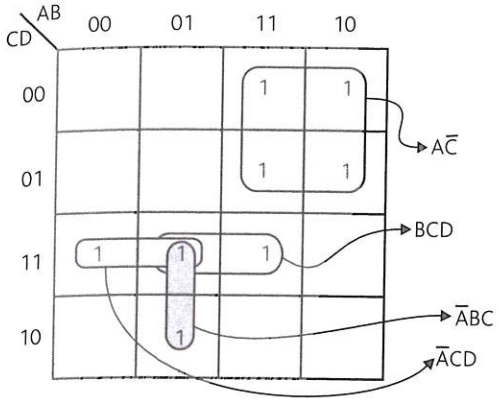
The four-variable term $\overline{B} \cdot \overline{D}$ covers four squares (the region B = 0, D = 0). In this case the adjacent squares are the corner squares. If you examine any pair of horizontally or vertically adjacent corners, you will find that they differ in one variable only.

Figure 2.66 Plotting $F = \overline{B}\,\overline{D} + AB\overline{C}D + \overline{A}C\overline{D}$ on a Karnaugh map.

of terms extends across the edge of a Karnaugh map, we have shaded it to emphasize the wraparound nature of the map.

4. In order either to read a product term from the map, or to plot a product term on the map, it is necessary to ask the question, 'what minterms (squares) are covered by this term?' Consider the term $\overline{A} \cdot D$ in Fig. 2.62. This term covers all squares for which A = 0 and D = 1 (a group of 4).

Having shown how terms are plotted on the Karnaugh map, the next step is to apply the map to the simplification of



Three two-variable groups overlap. In this example, the square (i.e. minterm) $\overline{A} \cdot B \cdot C \cdot D$ belongs to groups $B \cdot C \cdot D$, $\overline{A} \cdot B \cdot C$, and $\overline{A} \cdot C \cdot D$.

Figure 2.67 Plotting $F = \overline{A}CD + \overline{A}BC + BCD + A\overline{C}$ on a Karnaugh map.

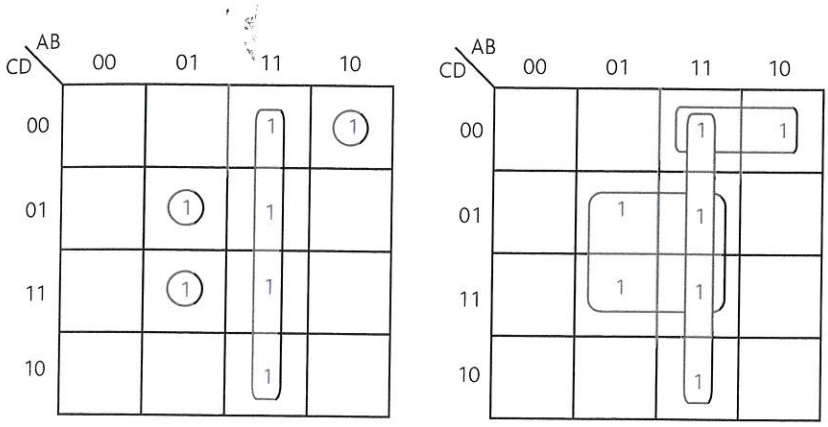


Figure 2.68 Karnaugh map for Example 1.

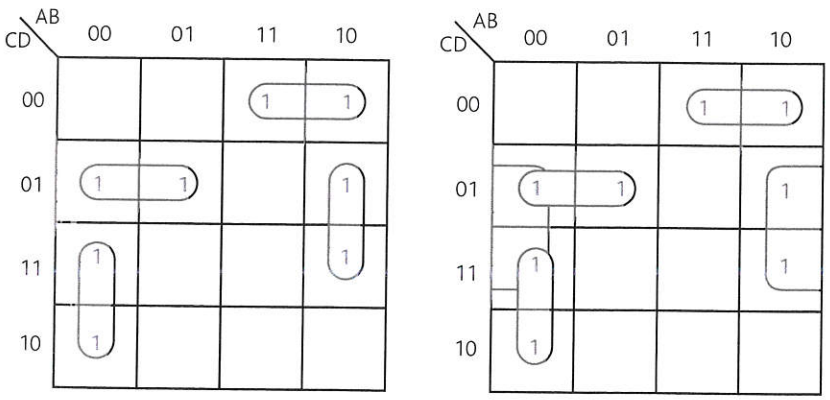


Figure 2.69 Karnaugh map for Example 2.

the expressions. Once again, we demonstrate this process by means of examples. In each case, the original function is plotted on the left-hand side of the figure and the regrouped ones (i.e. minterms) are plotted on the right-hand side.

Example 1 Figure 2.68 gives a Karnaugh map for the expression $F = A \cdot B + \overline{A} \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$. The simplified function is $F = A \cdot B + B \cdot D + A \cdot \overline{C} \cdot \overline{D}$.

Example 2 $F = A \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot D$ (Fig. 2.69). In this case only one regrouping is possible. The simplified function is $F = \overline{B} \cdot D + A \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot C$.

Example 3 $F = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot \overline{D}$ (Fig. 2.70). This function can be simplified to two product terms with $F = \overline{B} \cdot \overline{D} + B \cdot D$.

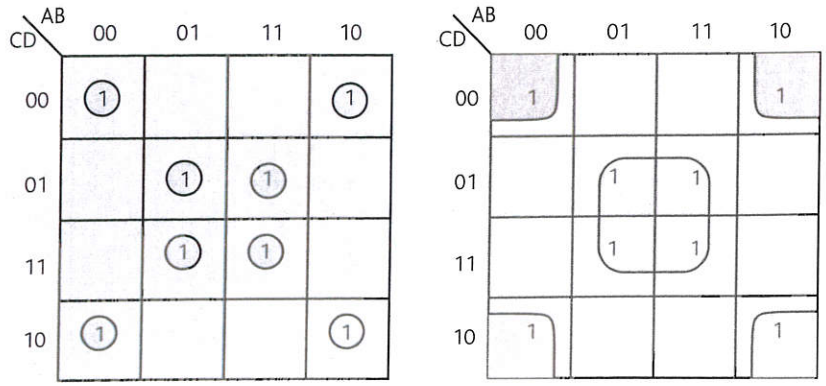
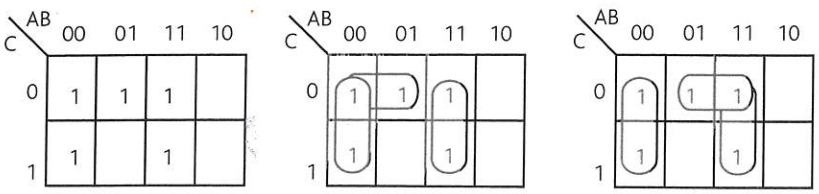


Figure 2.70 Karnaugh map for Example 3.



(a) Ones placed. (b) Ones grouped. (c) Alternate grouping.

Figure 2.71 Karnaugh map for Example 4.

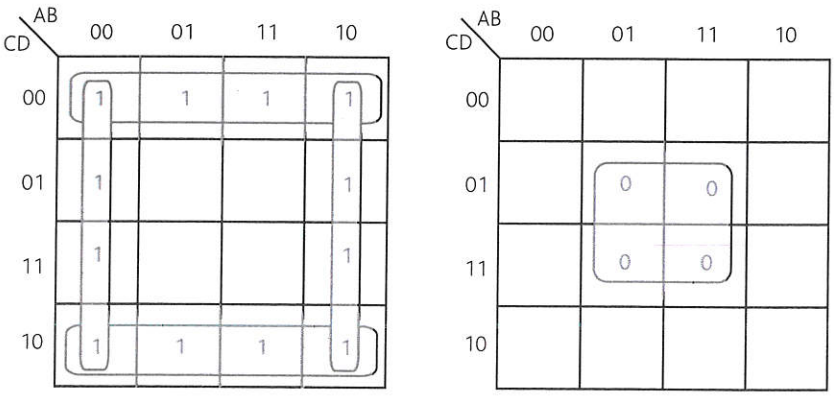


Figure 2.72 Example 5—using a Karnaugh map to obtain the complement of a function.

Example 4 $F = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} + A\cdot B\cdot\overline{C} + A\cdot B\cdot C$ (Fig. 2.71). We can group the minterms together in two ways, both of which are equally valid; that is, there are two equally correct simplifications of this expression. We can write either $F = \overline{A}\cdot\overline{B} + \overline{A}\cdot\overline{C} + A\cdot B$ or $F = \overline{A}\cdot\overline{B} + B\cdot\overline{C} + A\cdot B$.

Applications of Karnaugh maps

Karnaugh maps can also be used to convert *sum-of-products* expressions to the corresponding *product-of-sums* form. The first step in this process involves the generation of the complement of the sum-of-products expression.

Example 5 The Karnaugh map in Fig. 2.72 demonstrates how we can obtain the complement of a sum-of-products expression. Consider the expression $F = \overline{C}\cdot\overline{D} + \overline{A}\cdot\overline{B} + A\cdot\overline{B} + C\cdot\overline{D}$ (left-hand side of Fig. 2.72). If the squares on a Karnaugh map covered by 1s represent the function F, then the remaining squares covered by 0s must represent $\overline{F}$, the complement of F. In the right-hand side of Fig. 2.72, we have plotted the complement of this function. The group of four 0s corresponds to the expression $\overline{F} = B\cdot D$.

Example 6 We can use a Karnaugh map to convert of sum-of-products expression into a product-of-sums expression. In Example 5, we used the Karnaugh map to get the complement of a function in a product-of-sums form. If we then complement the complement, we get the function but in a sum-of-products form (because de Morgan's theorem allows us to step between SoP and PoS forms). Let's convert $F = A\cdot B\cdot C + \overline{C}\cdot D + \overline{A}\cdot B\cdot D$ into product of sums form (Fig. 2.73).

The complement of F is defined by the zeros on the map and may be read from the right-hand map as

$$\overline{F} = \overline{C}\cdot\overline{D} + \overline{B}\cdot C + \overline{A}\cdot\overline{D}$$

$$F = \overline{\overline{C}\cdot\overline{D} + \overline{B}\cdot C + \overline{A}\cdot\overline{D}}$$

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

We now have an expression for F in product-of-sums form.

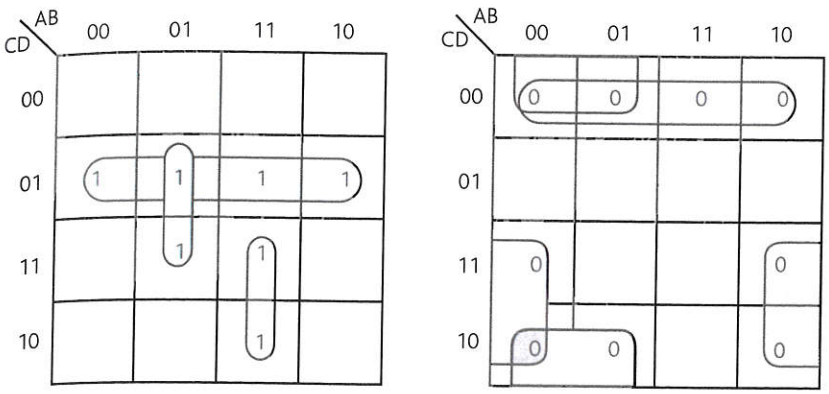


Figure 2.73 Example 6—using a Karnaugh map to convert an expression from SoP to PoS form.

Using the Karnaugh map to design a circuit with NAND logic

Now that we've demonstrated how Karnaugh maps are used to simplify and transform Boolean expressions, we're going to apply the Karnaugh map to the design of a simple logic circuit using NAND logic only.

A fire detection system protects a room against fire by means of four sensors. These sensors comprise a flame detector, a smoke detector, and two high-temperature detectors located at the opposite ends of the room. Because such sensors are prone to errors (i.e. false alarms or the failure to register a fire), the fire alarm is triggered only when two or more of the sensors indicate the presence of a fire simultaneously. The output of a sensor is a logical 1 if a fire is detected, otherwise a logical 0.

The output of the fire alarm circuit is a logical 1 whenever two or more of its inputs are a logical one. Table 2.19 gives the truth table for the fire detector circuit. The inputs from the four sensors are labeled A, B, C, and D. Because it is necessary only to detect two or more logical 1s on any of the lines, the actual order of A, B, C, and D columns doesn't matter. The circuit is to be constructed from two-input and three-input NAND gates only.

The output of the circuit, F, can be written down directly from Table 2.19 by ORing the 11 minterms to get the expression

$$F = \overline{A}\cdot\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot\overline{C}\cdot D + \overline{A}\cdot B\cdot C\cdot\overline{D} + \overline{A}\cdot B\cdot C\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot D + A\cdot\overline{B}\cdot C\cdot\overline{D} + A\cdot\overline{B}\cdot C\cdot D + A\cdot B\cdot\overline{C}\cdot\overline{D} + A\cdot B\cdot\overline{C}\cdot D + A\cdot B\cdot C\cdot\overline{D} + A\cdot B\cdot C\cdot D$$

Plotting these 11 minterms terms on a Karnaugh map we get Fig. 2.74(a). The next step is to group these terms together into six groups of four minterms (Fig. 2.74(b)). Note that the minterm $A\cdot B\cdot C\cdot D$ belongs to all six groups.

Therefore, the simplified sum-of-products form of F is given by

$$F = A\cdot B + A\cdot C + A\cdot D + B\cdot C + B\cdot D + C\cdot D$$

This expression is (as you might expect) the sum of all possible two-variable combinations.

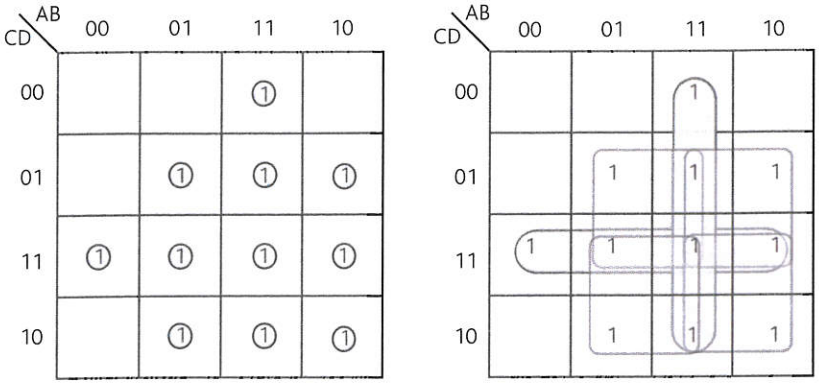
Inputs				Output
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 2.19 Truth table for a fire detector.

In order to convert the expression into NAND logic only form, we have to eliminate the five logical OR operators. We do that by complementing F twice and then using de Morgan's theorem.

$$F = \overline{\overline{F}} = \overline{\overline{A\cdot B + A\cdot C + A\cdot D + B\cdot C + B\cdot D + C\cdot D}}$$
$$= \overline{\overline{A\cdot B}\cdot\overline{A\cdot C}\cdot\overline{A\cdot D}\cdot\overline{B\cdot C}\cdot\overline{B\cdot D}\cdot\overline{C\cdot D}}$$

Although we have realized the expression in NAND logic as required, it calls for a six-input NAND gate. If the expression for F is examined, it can be seen that six terms are NANDed together, which is the same as ANDing them and then inverting the result. Because of the associative property



(a) Location of the 1s. (b) After grouping the 1s.

Figure 2.74 Karnaugh map corresponding to Table 2.19.

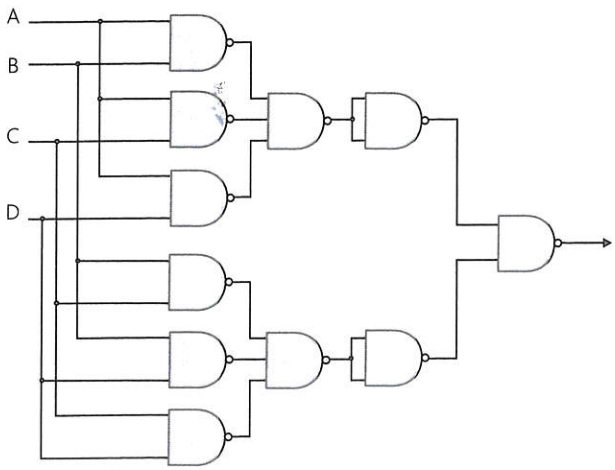


Figure 2.75 NAND-only circuit for fire detector.

of Boolean variables, we can write $X(Y \cdot Z) = (X \cdot Y)Z$ and hence extending this to our equation we get

$$F = \overline{\overline{A \cdot B} \cdot \overline{A \cdot C} \cdot \overline{A \cdot D} \cdot \overline{B \cdot C} \cdot \overline{B \cdot D} \cdot \overline{C \cdot D}}$$

Figure 2.75 shows how this expression can be implemented in terms of two- and three-input NAND gates.

Using Karnaugh Maps—an example

A circuit has four inputs, A, B, C, and D, representing the 16 natural binary integers from 0000 to 1111 (i.e. 0 to 15). The output of the circuit, F, is true if the input is divisible by a multiple of 4, 5, 6, or 7, with the exception of 15, in which case the output is false. Zero is not divisible by 4, 5, 6, or 7. Suppose we wish to design a logic circuit to implement F using NAND gates only.

We can obtain a sum-of-products expression for F from Table 2.20 by writing down the sum of the minterms (i.e. the lines with a 1).

$$F = \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot \overline{D}$$

Inputs				Number	F	
A	B	C	D			
0	0	0	0	0	0	
0	0	0	1	1	0	
0	0	1	0	2	0	
0	0	1	1	3	0	
0	1	0	0	4	1	Divisible by 4
0	1	0	1	5	1	Divisible by 5
0	1	1	0	6	1	Divisible by 6
0	1	1	1	7	1	Divisible by 7
1	0	0	0	8	1	Divisible by 4
1	0	0	1	9	0	
1	0	1	0	10	1	Divisible by 5
1	0	1	1	11	0	
1	1	0	0	12	1	Divisible by 6
1	1	0	1	13	0	
1	1	1	0	14	1	Divisible by 7
1	1	1	1	15	0	False by definition

Table 2.20 Truth table for example.

By means of Boolean algebra the expression can be simplified to

$$\begin{aligned} F &= \overline{A} \cdot B \cdot \overline{C}(D + \overline{D}) + \overline{A} \cdot B \cdot C(D + \overline{D}) + A \cdot \overline{B} \cdot \overline{D}(\overline{C} + C) \\ &\quad + A \cdot B \cdot \overline{D}(\overline{C} + C) \\ &= \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{D} + A \cdot B \cdot \overline{D} \\ &= \overline{A} \cdot B(C + \overline{C}) + A \cdot \overline{D}(\overline{B} + B) \\ &= \overline{A} \cdot B + A \cdot \overline{D} \end{aligned}$$

Figure 2.76 gives the Karnaugh map for F. In Fig. 2.77 the squares covered by 1s are formed into two groups of four. This gives $F = \overline{A} \cdot B + A \cdot \overline{D}$, which is reassuringly the same as the result obtained above.

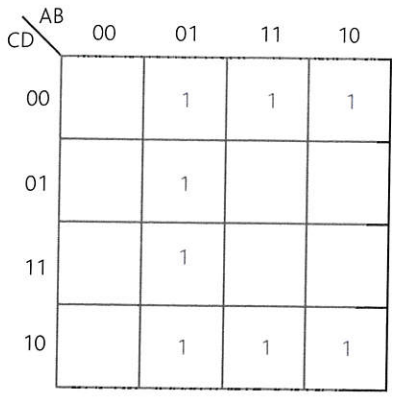


Figure 2.76 Karnaugh map for F.

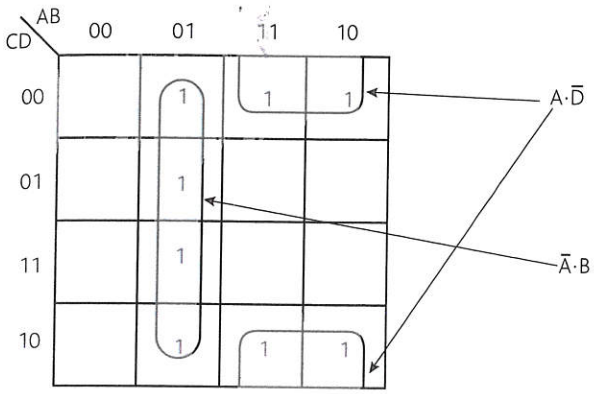


Figure 2.77 Karnaugh map after regrouping the minterms.

To obtain a product-of-sums expression, it's necessary to generate the complement of F in a sum-of-products form and then complement it.

$F = \overline{A}\cdot B + A\cdot\overline{D}$

$\overline{F} = \overline{\overline{A}\cdot B + A\cdot\overline{D}}$ Get the complement of F

$= (A + \overline{B})(\overline{A} + D)$ Complement of F in product-of-sums form

$= A\cdot\overline{A} + A\cdot D + \overline{A}\cdot\overline{B} + \overline{B}\cdot D$ Multiply out sum terms

$= A\cdot D + \overline{A}\cdot\overline{B} + \overline{B}\cdot D$ Complement of F in sum-of-products form

$= A\cdot D + \overline{A}\cdot\overline{B}$ Complement in simplified sum-of-products form

$F = \overline{A\cdot D + \overline{A}\cdot\overline{B}}$ Complement the complement to get F

$= (\overline{A} + \overline{D})(A + B)$ Function in required product-of-sums form

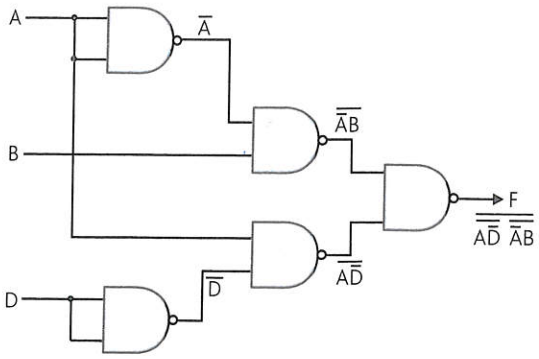


Figure 2.78 NAND-only circuit.

Note that the complement of F in sum-of-products form could have been obtained directly from the Karnaugh map of F by considering the squares covered by zeros.

To convert the expression $F = \overline{A}\cdot B + A\cdot\overline{D}$ into NAND logic form, the '+' must be eliminated.

$$F = \overline{\overline{F}} = \overline{\overline{A}\cdot B + A\cdot\overline{D}} = \overline{\overline{A}\cdot B}\cdot\overline{A\cdot\overline{D}}$$

The inverse functions $\overline{A}$ and $\overline{D}$ can be generated by two-input NAND gates with their inputs connected together. Figure 2.78 implements F in NAND logic only.

Karnaugh maps and don't care conditions

We now demonstrate how Karnaugh maps can be applied to problems in which the truth table isn't fully specified; that is, for certain input conditions the output is undefined. Occasionally, a system exists in which a certain combination of inputs can't happen; or, if it does, we don't care what the output is. In such cases, the output may be defined as either true or false.

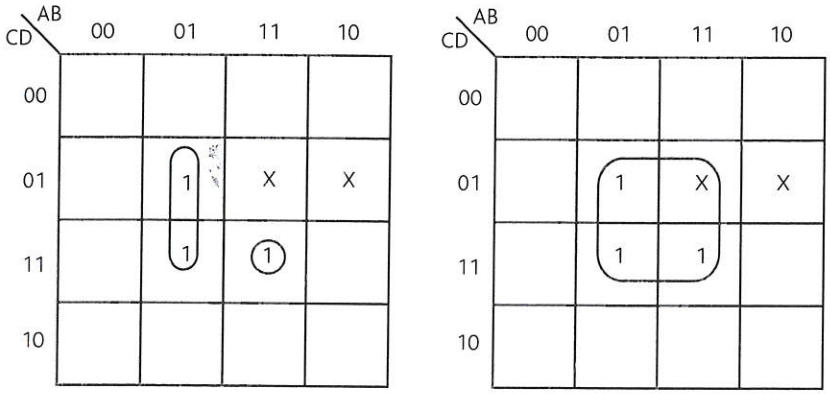
Consider the Karnaugh map of Fig. 2.79 for $F = \overline{A}\cdot B\cdot D + A\cdot B\cdot C\cdot D$. Now suppose that the input conditions $A\cdot B\cdot\overline{C}\cdot D$ and $A\cdot\overline{B}\cdot\overline{C}\cdot D$ cannot occur. We have marked these two inputs on the map with an X. The value of X is undefined (if the input can't occur then the value of the output is undefined).

If an input can't occur and the output is undefined, we can cover that square with either a 0 or a 1. In Fig. 2.79(b) we have made one of the Xs a 1 and one of the Xs a zero. We can express the output function as $F = B\cdot D$, which is simpler than the function in Fig. 2.79(a).

A don't care condition is set to a 0 or a 1 in order to simplify the solution. There is an important exception. Although an impossible input can't occur in normal circumstances, it could under fault conditions (e.g. when an input circuit fails). No designer would assign an output to an impossible input condition that might lead to an unsafe or dangerous situation. However, the ultimate aim is to cover all the 1s in

the map and to incorporate them in the smallest number of large groups.

The following example demonstrates the concept of *impossible* input conditions. An air conditioning system has two temperature control inputs. One input, C, from a cold-sensing thermostat is true if the temperature is below 15°C and false otherwise. The other input, H, from a hot-sensing thermostat is true if the temperature is above 22°C and false otherwise. Table 2.21 lists the four possible logical conditions for the two inputs.



(a) The function F = $\bar{A}BD + ABCD$. Note that the inputs $AB\bar{C}D$ and $A\bar{B}\bar{C}D$ cannot occur

(b) The function F = BD. Minterm $AB\bar{C}D$ is included to simplify the expression

Figure 2.79 The effect of don't care conditions.

Inputs		Meaning
C	H	
0	0	Temperature OK
0	1	Too hot
1	0	Too cold
1	1	Impossible condition

Table 2.21 Truth table for a pair of temperature sensors.

The input condition C = 1, H = 1 in Table 2.20 has no real meaning, because it's impossible to be too hot and too cold simultaneously. Such an input condition could arise only if at least one of the thermostats failed. Consider now the example of an air conditioning unit with four inputs and four outputs. Table 2.22 defines the meaning of the inputs to the controller.

The controller has four outputs P, Q, R, and S. When P = 1 a heater is switched on and when Q = 1 a cooler is switched on. Similarly, a humidifier is switched on by R = 1 and a dehumidifier by S = 1. In each case a logical 0 switches off the appropriate device. The relationship between the inputs and outputs is as follows.

- If the temperature and humidity are both within limits, switch off the heater and the cooler. The humidifier and dehumidifier are both switched off unless stated otherwise.
- If the humidity is within limits, switch on the heater if the temperature is too low and switch on the cooler if the temperature is too high.
- If the temperature is within limits, switch on the heater if the humidity is too low and the cooler if the humidity is too high.
- If the humidity is high and the temperature low, switch on the heater. If the humidity is low and the temperature high, switch on the cooler.
- If both the temperature and humidity are high switch on the cooler and dehumidifier.
- If both the temperature and humidity are too low switch on the heater and humidifier.

The relationship between the inputs and outputs can now be expressed in terms of a truth table (Table 2.23). We can draw Karnaugh maps for P to S, plotting a 0 for a zero state, a 1 for a one state, and an X for an impossible state. Remember that an X on the Karnaugh map corresponds to a state that cannot exist and therefore its value is known as a *don't care* condition.

Input	Name	Meaning when input = 0	Meaning when input = 1
H	Hot	Temperature < upper limit	Temperature > upper limit
C	Cold	Temperature > lower limit	Temperature < lower limit
W	Wet	Humidity < upper limit	Humidity > upper limit
D	Dry	Humidity > lower limit	Humidity < lower limit

Table 2.22 Truth table for a climate controller.

Inputs				Condition	Outputs			
H	C	W	D		P	Q	R	S
					heater	cooler	humidifier	dehumidifier
0	0	0	0	OK	0	0	0	0
0	0	0	1	Dry	1	0	0	0
0	0	1	0	Wet	0	1	0	0
0	0	1	1	Impossible	X	X	X	X
0	1	0	0	Cold	1	0	0	0
0	1	0	1	Cold and dry	1	0	1	0
0	1	1	0	Cold and wet	1	0	0	0
0	1	1	1	Impossible	X	X	X	X
1	0	0	0	Hot	0	1	0	0
1	0	0	1	Hot and dry	0	1	0	0
1	0	1	0	Hot and wet	0	1	0	1
1	0	1	1	Impossible	X	X	X	X
1	1	0	0	Impossible	X	X	X	X
1	1	0	1	Impossible	X	X	X	X
1	1	1	0	Impossible	X	X	X	X
1	1	1	1	Impossible	X	X	X	X

Table 2.23 Truth table for a climate controller.

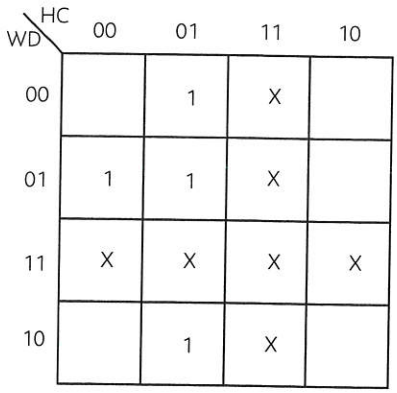


Figure 2.80 Karnaugh map for P (the heater).

Figure 2.80 provides a Karnaugh map corresponding to output P, the heater. We have marked all the don't care conditions with an X. We could replace the Xs by 1s or 0s. However, by forcing some of the don't care outputs to be a 1, we can convert a group of 1s into a larger group.

Figure 2.81 provides Karnaugh maps for outputs P, Q, R, and S. In each case we have chosen the don't care conditions to simplify the output function. For example, the Karnaugh map of Fig. 2.81(a) corresponds to output P where we have included six of the don't care conditions within the groupings to get $P = C + H \cdot D$.

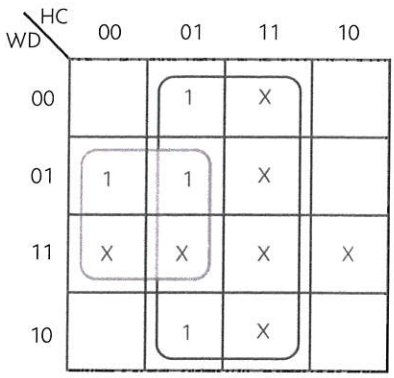
You should appreciate that by taking this approach we have designed a circuit that sets the output 1 for some don't care inputs and 0 for other don't care inputs. You cannot avoid this. The output of any digital circuit must always be in a 0 or a 1 state. As we said at the beginning of this chapter, there is no such state as an indeterminate state. It is up to the designer to choose what outputs are to be assigned to don't care inputs.

Exploiting don't care conditions—constructing a seven-segment decoder

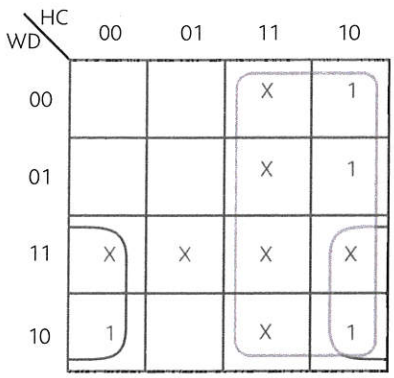
We now design a BCD-to-seven-segment decoder (BCD means binary-coded decimal). The decoder has a 4-bit natural binary BCD input represented by D, C, B, A, where A is the least-significant bit. Assume that the BCD input can never be greater than 9 (Chapter 4 describes BCD codes). The seven-segment decoder illustrated by Fig. 2.82 has seven outputs (a to g), which are used to illuminate any combination of bars a to g of a seven-segment display; for example, if the code for 2 (i.e. 0010) is sent to the decoder, segments a, b, d, e, and g are illuminated to form a '2'.

The truth table for this problem is given in Table 2.24. This table has four inputs and seven outputs (one for each of the segments).

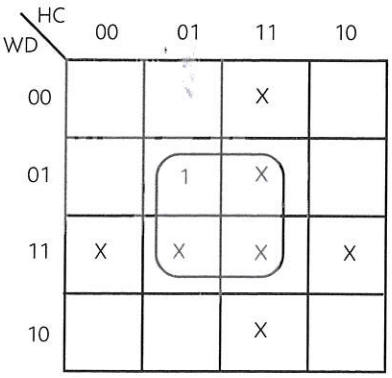
We can now solve the equation for segments a to g. By using Karnaugh maps the don't care conditions can be catered for.



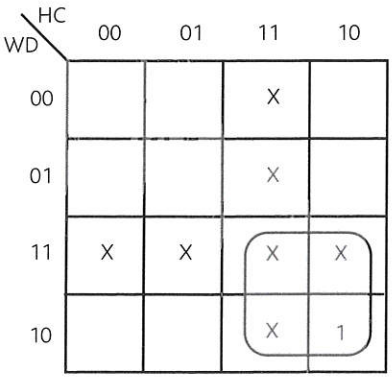
(a) $P = C + \overline{H}D$.



(b) $Q = H + \overline{C}W$.



(c) $R = CD$.



(d) $S = HW$.

Figure 2.81 Karnaugh maps for outputs P, Q, R, and S.

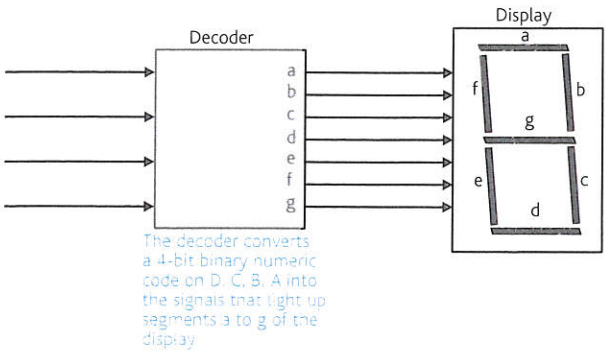


Figure 2.82 The seven-segment display.

Figure 2.83 gives the Karnaugh map for segment a. From the Karnaugh map we can write down the expression for $a = D + B + C{\cdot}A + \overline{C}{\cdot}\overline{A}$.

An alternative approach is to obtain a by considering the zeros on the map to get the complement of a. From the Karnaugh map in Fig. 2.84 we can write $a = \overline{\overline{D}{\cdot}\overline{C}{\cdot}\overline{B}{\cdot}A + C{\cdot}\overline{B}{\cdot}\overline{A}}$. Therefore,

$$
\begin{aligned}
a &= \overline{\overline{D}{\cdot}\overline{C}{\cdot}\overline{B}{\cdot}A + C{\cdot}\overline{B}{\cdot}\overline{A}} \\
&= (D + C + B + \overline{A})\,(\overline{C} + B + A) \\
&= D{\cdot}\overline{C} + D{\cdot}B + D{\cdot}A + C{\cdot}\overline{C} + C{\cdot}B + C{\cdot}A + \overline{C}{\cdot}B \\
&\quad + B{\cdot}B + B{\cdot}A + \overline{C}{\cdot}\overline{A} + B{\cdot}\overline{A} + A{\cdot}\overline{A} \\
&= D{\cdot}\overline{C} + D{\cdot}B + D{\cdot}A + C{\cdot}B + C{\cdot}A + \overline{C}{\cdot}B + B \\
&\quad + B{\cdot}A + \overline{C}{\cdot}\overline{A} + B{\cdot}\overline{A} \\
&= D{\cdot}\overline{C} + D{\cdot}A + C{\cdot}A + B + \overline{C}{\cdot}\overline{A} \\
&= D{\cdot}\overline{C} + C{\cdot}A + B + \overline{C}{\cdot}\overline{A}
\end{aligned}
$$

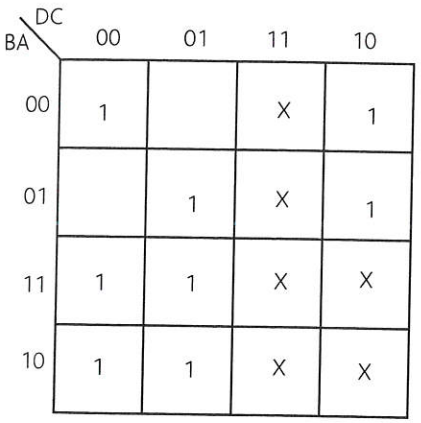
This expression offers no improvement over the first realization of a.

Figure 2.85 provides the Karnaugh map for segment b, which gives $b = \overline{C} + \overline{B}{\cdot}\overline{A} + B{\cdot}A$. We can proceed as we did for segment a and see what happens if we use $\overline{b}$. Plotting zeros on the Karnaugh map for $\overline{b}$ we get $\overline{b} = C + \overline{B}{\cdot}A{\cdot}C{\cdot}B{\cdot}\overline{A}$ Fig. 2.86. Therefore,

$$
\begin{aligned}
b &= \overline{C{\cdot}\overline{B}{\cdot}A + C{\cdot}B{\cdot}\overline{A}} \\
&= (\overline{C} + B + \overline{A})(\overline{C} + \overline{B} + A) \\
&= \overline{C} + B{\cdot}A + \overline{B}{\cdot}\overline{A}
\end{aligned}
$$

Inputs	Character	Outputs						
D C B A		a	b	c	d	e	f	g
0 0 0 0		1	1	1	1	1	1	0
0 0 0 1		0	1	1	0	0	0	0
0 0 1 0		1	1	0	1	1	0	1
0 0 1 1		1	1	1	1	0	0	1
0 1 0 0		0	1	1	0	0	1	1
0 1 0 1		1	0	1	1	0	1	1
0 1 1 0		1	0	1	1	1	1	1
0 1 1 1		1	1	1	0	0	0	0
1 0 0 0		1	1	1	1	1	1	1
1 0 0 1		1	1	1	0	0	1	1
1 0 1 0	Forbidden code	X	X	X	X	X	X	X
1 0 1 1		X	X	X	X	X	X	X
1 1 0 0		X	X	X	X	X	X	X
1 1 0 1		X	X	X	X	X	X	X
1 1 1 0		X	X	X	X	X	X	X
1 1 1 1		X	X	X	X	X	X	X

Table 2.24 Truth table for a seven—segment display.



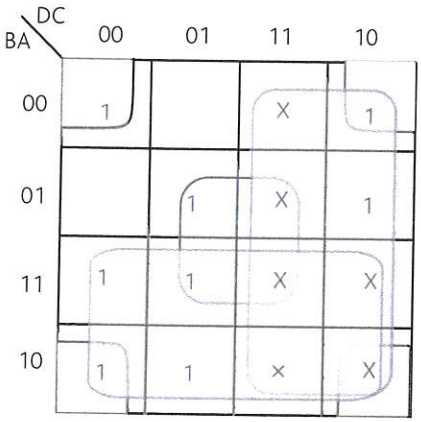
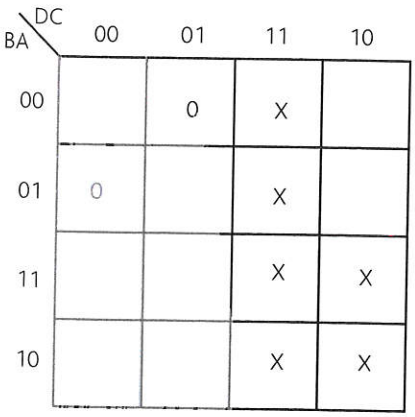


Figure 2.83 Karnaugh map for the segment a control signal.



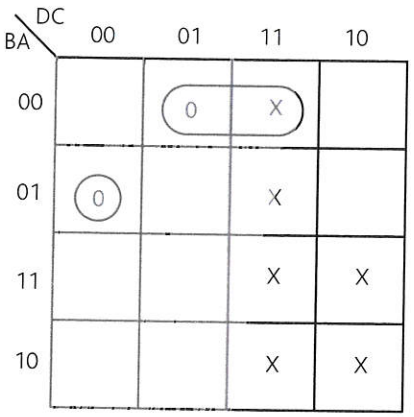


Figure 2.84 Karnaugh map for the complement of segment a.

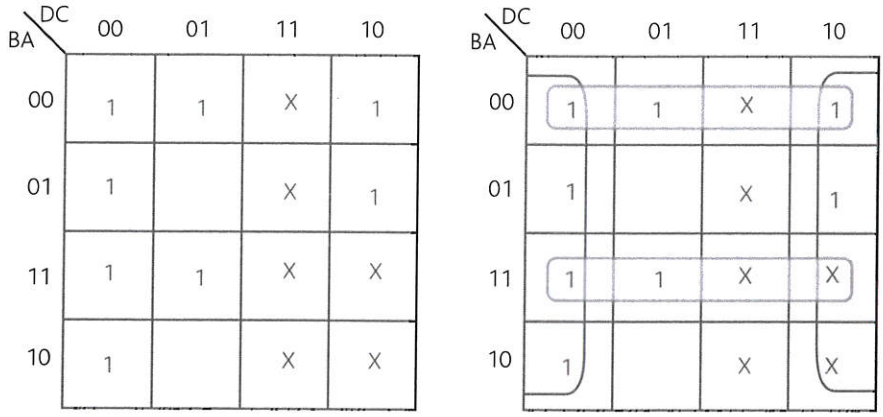


Figure 2.85 Karnaugh map for segment b.

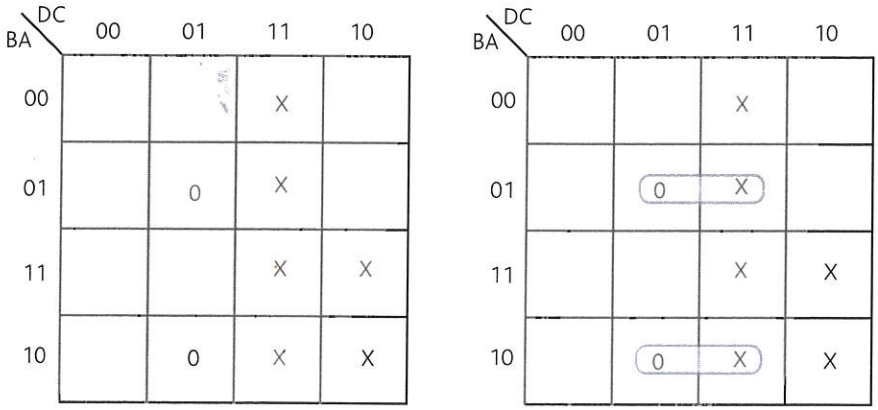


Figure 2.86 Karnaugh map for the complement of segment b.

This expression yields the same result as that obtained directly by considering the 1s on the Karnaugh map. The equations for the remaining five segments can be considered in a similar way.

Example of the use of Karnaugh maps to implement a circuit

A logic circuit has four inputs D, C, B, A, which represent two pairs of bits (D,C) and (B,A). Bits (B,A) are subtracted from bits (D,C) to give a result F_1, F_0 and an n-bit that indicates a negative result. Table 2.25 provides a truth table for this problem.

We first construct three Karnaugh maps for the outputs and use them to obtain simplified sum-of-product expressions (Table 2.25).

Figure 2.87 provides the three Karnaugh maps corresponding to outputs n, F_1, and F_0 in the truth table. The 1s have been regrouped under each truth table to provide the minimum number of large groups.

We can write down expressions for n, F_1, and F_0 from Fig. 2.87 as

$$n = \overline{D}\cdot B + \overline{C}\cdot B\cdot A + \overline{D}\cdot\overline{C}\cdot A$$
$$F_1 = \overline{D}\cdot\overline{C}\cdot B + D\cdot C\cdot\overline{B} + D\cdot\overline{B}\cdot\overline{A} + \overline{D}\cdot B\cdot A$$
$$F_0 = \overline{C}\cdot A + C\cdot\overline{A}$$

Inputs				Number	Outputs		
					n	F_1	F_0
D	C	B	A				
0	0	0	0	0 − 0 = 0	0	0	0
0	0	0	1	0 − 1 = −1	1	0	1
0	0	1	0	0 − 2 = −2	1	1	0
0	0	1	1	0 − 3 = −3	1	1	1
0	1	0	0	1 − 0 = 1	0	0	1
0	1	0	1	1 − 1 = 0	0	0	0
0	1	1	0	1 − 2 = −1	1	0	1
0	1	1	1	1 − 3 = −2	1	1	0
1	0	0	0	2 − 0 = 2	0	1	0
1	0	0	1	2 − 1 = 1	0	0	1
1	0	1	0	2 − 2 = 0	0	0	0
1	0	1	1	2 − 3 = −1	1	0	1
1	1	0	0	3 − 0 = 3	0	1	1
1	1	0	1	3 − 1 = 2	0	1	0
1	1	1	0	3 − 2 = 1	0	0	1
1	1	1	1	3 − 3 = 0	0	0	0

Table 2.25 Truth table for a two-bit subtractor.

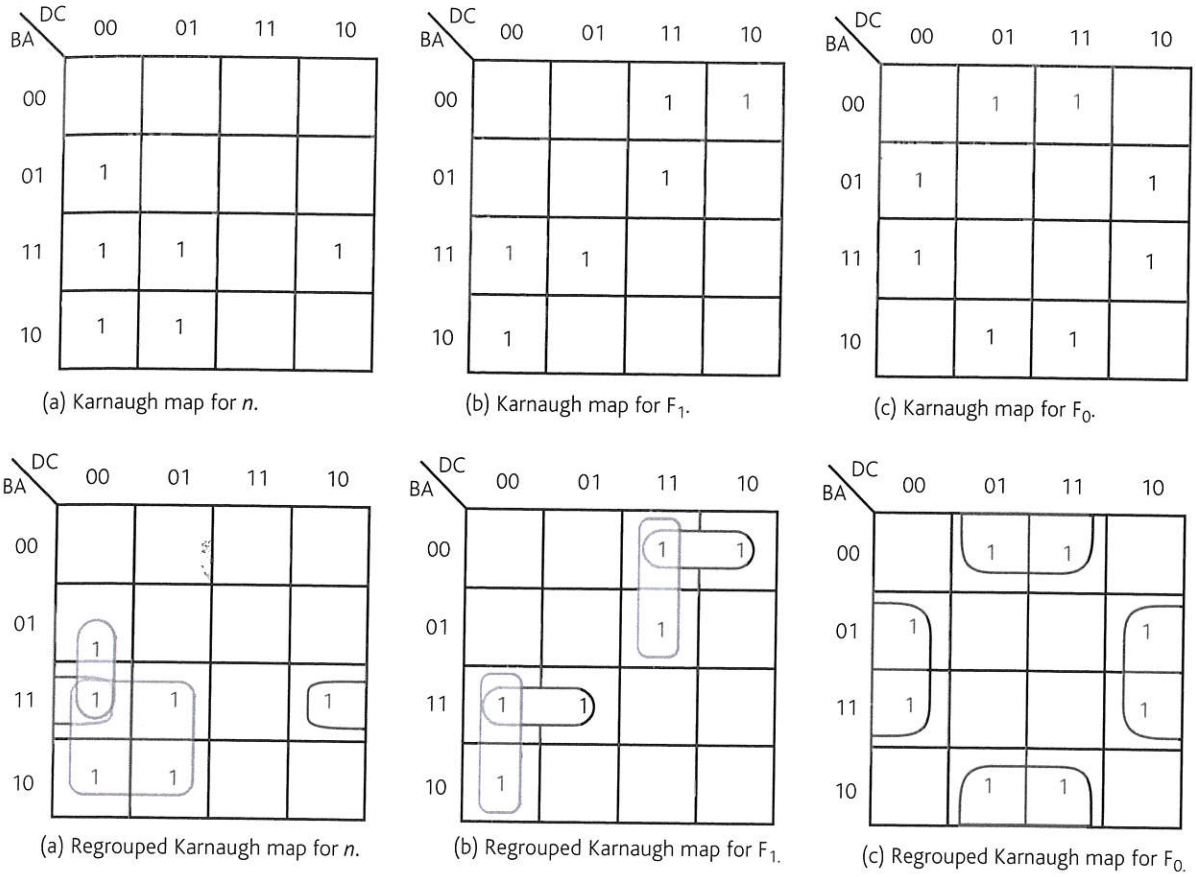


Figure 2.87 The Karnaugh maps for the subtractor.

2.6 Special-purpose logic elements

So far, we've looked at the primitive logic elements from which all digital systems can be constructed. As technology progressed, more and more components were fabricated on single chips of silicon to produce increasingly complex circuits. Today, you can buy chips with tens of millions of gates that can be interconnected electronically (i.e. the chip provides a digital system whose structure can be modified electronically by the user). Indeed, by combining microprocessor technology, electronically programmable with arrays of gates, we can now construct self-modifying (self-adaptive) digital systems.

Let's briefly review the development of digital circuits. The first digital circuits contained a few basic NAND, NOR, AND gates, and were called *small-scale integration* (SSI). Basic SSI gates were available in 14-pin *dual-in-line* (DIL) packages. Dual-in-line simply means that there are two parallel rows of pins (i.e. contacts) forming the interface between the chip and the outside world. The rows are 0.3 inches apart and the pins are spaced by 0.1 inch. Two pins are used for the power supply (V_{cc} = +5.0 V and ground = 0 V). These devices are often called *74-series* logic elements because the part number of each chip begins with 74; for example, a 7400 chip contains four NAND gates. Today, the packaging of such gates has shrunk to the point where the packages are very tiny and are attached to circuit boards by automatic machines.

It soon became possible to put tens of gates on a chip and manufacturers connected gates together to create logic functions such as a 4-bit adder, a multiplexer, and a decoder. Such circuits are called *medium-scale integration* (MSI). By the 1970s entire systems began to appear on a single silicon chip, of which the microprocessor is the most spectacular example. The technology used to make such complex systems is called *large-scale integration* (LSI). In the late 1980s LSI gave way to *very-large-scale integration* (VLSI), which allowed designers to fabricate millions of transistors on a chip. Initially, VLSI technology was applied to the design of memories rather than microprocessors. Memory systems are much easier to design because they have a regular structure (i.e. a simple memory cell is replicated millions of times).