$x + yz = (x + y)(x + z)$

$(x + y)' = x'y'$ (b) $(xy)' = x' + y'$

$x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$

$$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$

$(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

$(A + B + C + D + \cdots + F)' = A'B'C'D' \ldots F'$

$(ABCD \ldots F)' = A' + B' + C' + D' + \cdots + F'$

Table 2.3
Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Conversion between Canonical Forms

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms which make the function equal to 1, whereas its complement is a 1 for those minterms for which the function is a 0. As an example, consider the function

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

This function has a complement that can be expressed as

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0M_2M_3 = \Pi(0, 2, 3)$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2.3. From the table, it is clear that the following relation holds:

$$m_j' = M_j$$

That is, the **maxterm with subscript *j* is a complement of the minterm with the same subscript *j* and vice versa.**

is similar. We now state a general conversion procedure: To convert from one canonical form to another, interchange the symbols Σ and Π and list those numbers missing from the original form. In order to find the missing terms, one must realize that the total number of minterms or maxterms is 2^n, where n is the number of binary variables in the function.

Table 2.6
Truth Table for F = xy + x'z

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$
$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

$$m_j' = M_j$$

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0M_2M_4M_5 \end{aligned}$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol, Π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

variables; therefore, the map consists of eight squares. Note that the minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code (Table 1.6). The characteristic of this sequence is that **only one bit changes in value from one adjacent column to the next.** The map drawn in part (b) is marked with numbers in each row and

Sum of Minterms

Previously, we stated that, for n binary variables, one can obtain 2^n distinct minterms and that any Boolean function can be expressed as a sum of minterms. **The minterms whose sum defines the Boolean function are those which give the 1's of the function in a**

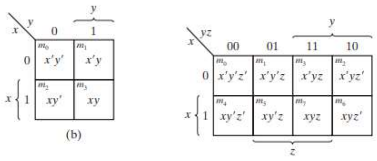
truth table. Since the function can be either 1 or 0 for each minterm, and since there are 2^n minterms, one can calculate all the functions that can be formed with n variables to be 2^{2^n}. It is sometimes convenient to express a Boolean function in its sum-of-minterms

Table 2.5
Truth Table for F = A + B'C

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

1's under F for those combinations for which $A = 1$ and $BC = 01$. From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6, and 7.



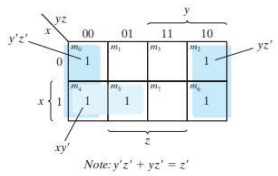
(b)

Simplify the Boolean function

$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$$

The map for F is shown in Fig. 3.6. First, we combine the four adjacent squares in the first and last columns to give the single literal term z'. The remaining single square, representing minterm 5, is combined with an adjacent square that has already been used once. This is not only permissible, but rather desirable, because the two adjacent squares give the two-literal term xy' and the single square represents the three-literal minterm $xy'z$. The simplified function is

$$F = z' + xy'$$



Note: $y'z' + yz' = z'$

FIGURE 3.6
Map for Example 3.3, $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

For the Boolean function

$$F = A'C + A'B + AB'C + BC$$

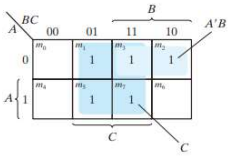
(a) Express this function as a sum of minterms.
(b) Find the minimal sum-of-products expression.

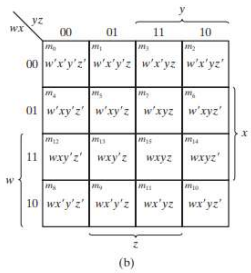
Note that F is a sum of products. Three product terms in the expression have two literals and are represented in a three-variable map by two squares each. The two squares corresponding to the first term, $A'C$, are found in Fig. 3.7 from the coincidence of A' (first row) and C (two middle columns) to give squares 001 and 011. Note that, in marking 1's in the squares, it is possible to find a 1 already placed there from a preceding term. This happens with the second term, $A'B$, which has 1's in squares 011 and 010. Square 011 is common with the first term, $A'C$, though, so only one 1 is marked in it. Continuing in this fashion, we determine that the term $AB'C$ belongs in square 101, corresponding to minterm 5, and the term BC has two 1's in squares 011 and 111. The function has a total of five minterms, as indicated by the five 1's in the map of Fig. 3.7. The minterms are read directly from the map to be 1, 2, 3, 5, and 7. The function can be expressed in sum-of-minterms form as

$$F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$$

The sum-of-products expression, as originally given, has too many terms. It can be simplified, as shown in the map, to an expression with only two terms:

$$F = C + A'B$$





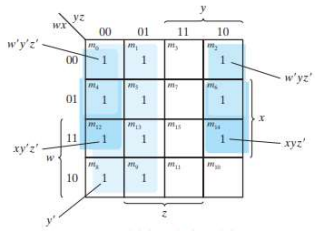
(b)

Simplify the Boolean function

$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

Since the function has four variables, a four-variable map must be used. The minterms listed in the sum are marked by 1's in the map of Fig. 3.9. Eight adjacent squares marked with 1's can be combined to form the one literal term y'. The remaining three 1's on the right cannot be combined to give a simplified term; they must be combined as two or four adjacent squares. The larger the number of squares combined, the smaller is the number of literals in the term. In this example, the top two 1's on the right are combined with the top two 1's on the left to give the term $w'z'$. Note that it is permissible to use the same square more than once. We are now left with a square marked by 1 in the third row and fourth column (square 1110). Instead of taking this square alone (which will give a term with four literals), we combine it with squares already used to form an area of four adjacent squares. These squares make up the two middle rows and the two end columns, giving the term xz'. The simplified function is

$$F = y' + w'z' + xz'$$



Note: $w'y'z' + w'yz' = w'z'$
$xy'z' + xyz' = xz'$

more systematic if we understand the meaning of two special types of terms. **A *prime implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map.** If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be *essential.*

squares, and so on. The essential prime implicants are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it. The prime implicant is essential if it is the only prime implicant that covers the minterm.

3.4 PRODUCT-OF-SUMS SIMPLIFICATION

The minimized Boolean functions derived from the map in all previous examples were expressed in sum-of-products form. With a minor modification, the product-of-sums form can be obtained.

The procedure for obtaining a minimized function in product-of-sums form follows from the basic properties of Boolean functions. The 1's placed in the squares of the map represent the minterms of the function. The minterms not included in the standard sum-of-products form of a function denote the complement of the function. From this observation, we see that the complement of a function is represented in the map by the squares not marked by 1's. If we mark the empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified sum-of-products expression of the complement of the function (i.e., of F'). The complement of F' gives us back the function F in product-of-sums form (a consequence of DeMorgan's theorem). Because of the generalized DeMorgan's theorem, the function so obtained is automatically in product-of-sums form. The best way to show this is by example.

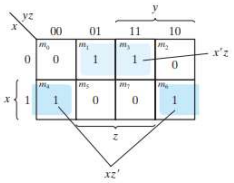


FIGURE 3.14
Map for the function of Table 3.1

In product-of-maxterms form, it is expressed as

$$F(x, y, z) = \Pi(0, 2, 5, 7)$$

In other words, the 1's of the function represent the minterms and the 0's represent the maxterms. The map for this function is shown in Fig. 3.14. One can start simplifying the function by first marking the 1's for each minterm that the function is a 1. The remaining squares are marked by 0's. If, instead, the product of maxterms is initially given, one can start marking 0's in those squares listed in the function; the remaining squares are then marked by 1's. Once the 1's and 0's are marked, the function can be simplified in either one of the standard forms. For the sum of products, we combine the 1's to obtain

$$F = x'z + xz'$$

For the product of sums, we combine the 0's to obtain the simplified complemented function

$$F' = xz + x'z'$$

which shows that the exclusive-OR function is the complement of the equivalence function (Section 2.6). Taking the complement of F', we obtain the simplified function in product-of-sums form:

$$F = (x' + z')(x + z)$$

To enter a function expressed in product-of-sums form into the map, use the complement of the function to find the squares that are to be marked by 0's. For example, the function

$$F = (A' + B' + C')(B + D)$$

can be entered into the map by first taking its complement, namely,

$$F' = ABC + B'D'$$

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:

$$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$$

The 1's marked in the map of Fig. 3.12 represent all the minterms of the function. The squares marked with 0's represent the minterms not included in F and therefore denote the complement of F. Combining the squares with 1's gives the simplified function in sum-of-products form:

(a) $F = B'D' + B'C' + A'C'D$

If the squares marked with 0's are combined, as shown in the diagram, we obtain the simplified complemented function:

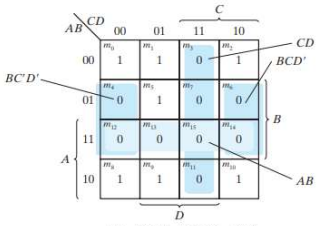
$$F' = AB + CD + BD'$$

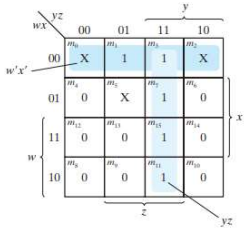
Applying DeMorgan's theorem (by taking the dual and complementing each literal as described in Section 2.4), we obtain the simplified function in product-of-sums form:

(b) $F = (A' + B')(C' + D')(B' + D)$

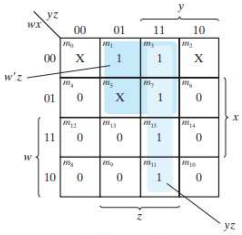
■

The gate-level implementation of the simplified expressions obtained in Example 3.7 is shown in Fig. 3.13. The sum-of-products expression is implemented in (a) with a group of AND gates, one for each AND term. The outputs of the AND gates are connected to the inputs of a single OR gate. The same function is implemented in (b) in its product-of-sums





(a) $F = yz + w'x'$



(b) $F = yz + w'z$

FIGURE 3.15
Example with don't-care conditions

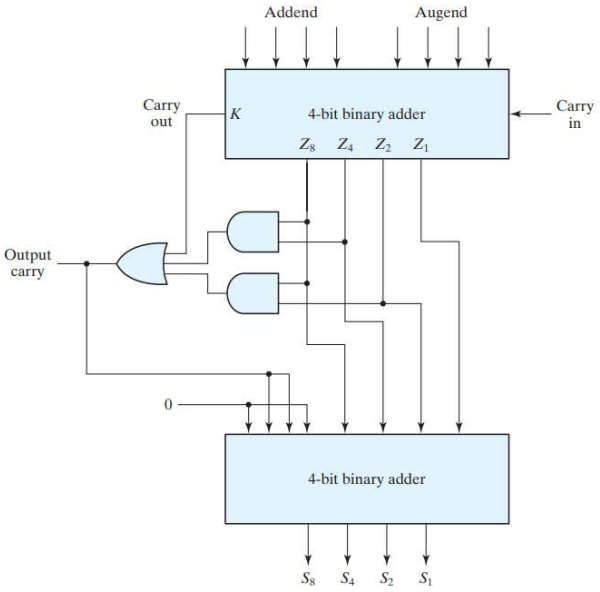
and then marking 0's in the squares representing the minterms of F'. The remaining squares are marked with 1's.

Full Adder

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

Table 4.5
Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19



$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$
$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

Since there are three inputs and a total of eight minterms, we need a three-to-eight-line decoder. The implementation is shown in Fig. 4.21. The decoder generates the eight minterms for x, y, and z. The OR gate for output S forms the logical sum of minterms 1, 2, 4, and 7. The OR gate for output C forms the logical sum of minterms 3, 5, 6, and 7.

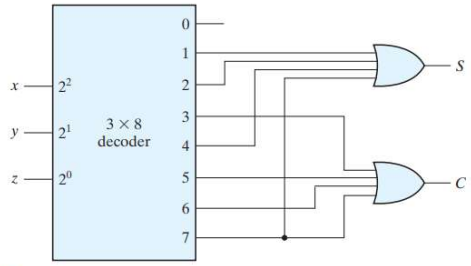


FIGURE 4.21
Implementation of a full adder with a decoder

Convert $(0.6875)_{10}$ to binary. First, 0.6875 is multiplied by 2 to give an integer and a fraction. Then the new fraction is multiplied by 2 to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. The coefficients of the binary number are obtained from the integers as follows:

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

Therefore, the answer is $(0.6875)_{10} = (0.\ a_{-1}\ a_{-2}\ a_{-3}\ a_{-4})_2 = (0.1011)_2$.

patterns of 1's and 0's. Since $2^3 = 8$ and $2^4 = 16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits. The first 16 num-

Table 1.2
Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Diminished Radix Complement

Given a number N in base r having n digits, the $(r - 1)$'s complement of N, i.e., its diminished radix complement, is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$

Radix Complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$. Thus, the 10's complement of decimal 2389 is $7610 + 1 = 7611$ and is obtained by adding 1 to the 9's complement value. The 2's complement of binary 101100 is $010011 + 1 = 010100$ and is obtained by adding 1 to the 1's-complement value.

Subtraction with Complements

The direct method of subtraction taught in elementary schools uses the borrow concept. In this method, we borrow a 1 from a higher significant position when the minuend digit is smaller than the subtrahend digit. The method works well when people perform subtraction with paper and pencil. However, when subtraction is implemented with digital hardware, the method is less efficient than the method that uses complements.

The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

The Gray code is used in applications in which the normal sequence of binary numbers generated by the hardware may produce an error or ambiguity during the transition from one number to the next. If binary numbers are used, a change, for example, from 0111 to 1000 may produce an intermediate erroneous number 1001 if the value of the rightmost bit takes longer to change than do the values of the other three bits. This could have serious consequences for the machine using the information. The Gray code eliminates this problem, since only one bit changes its value during any transition between two numbers.

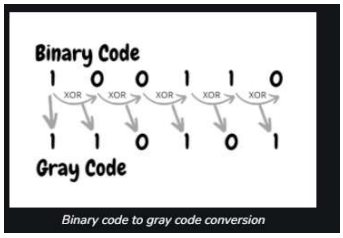
A Convert ⟦(10110011)⟧ $_{BCD}$ to decimal.

Show drafts

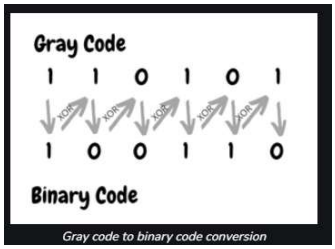
In Binary-Coded Decimal (BCD), each nibble (group of 4 bits) represents a single decimal digit. To convert a BCD number to decimal, we follow these steps:

1. **Split the BCD number into nibbles:** 1011 0001 1
2. **Convert each nibble to its decimal equivalent:**
   - First nibble (1011): 1 * 8 + 0 * 4 + 1 * 2 + 1 * 1 = 11
   - Second nibble (0001): 0 * 8 + 0 * 4 + 0 * 2 + 1 * 1 = 1
   - Third nibble (1): 1 * 8 = 8
3. **Combine the decimal equivalents of each nibble:** 11 (first nibble) * 100 (weight for the hundreds place) + 1 (second nibble) * 10 (weight for the tens place) + 8 (third nibble) * 1 (weight for the units place) = 1118

Therefore, ⟦(10110011)⟧ $_{BCD}$ is equal to **1118** in decimal.



Binary code to gray code conversion



Gray code to binary code conversion

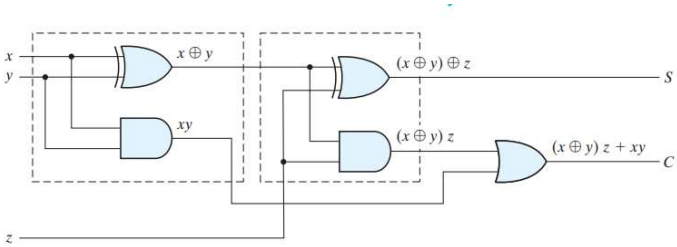


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

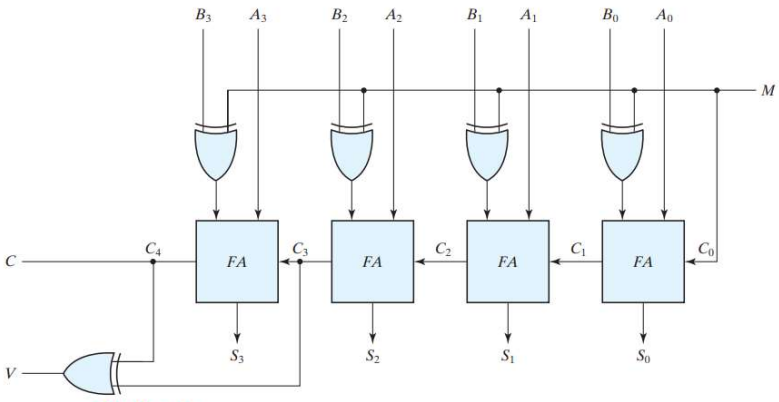


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

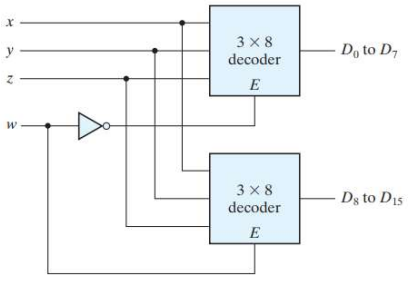


FIGURE 4.20
4 × 16 decoder constructed with two 3 × 8 decoders

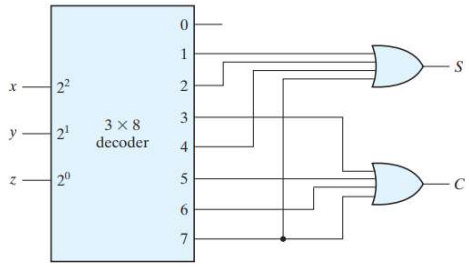


FIGURE 4.21
Implementation of a full adder with a decoder