Chapter 3 : Simplification of Boolean Functions

3-1 The Map Method

3-2 Two- and Three-Variable Maps

3-3 Four Variable Map

3-4 Five Variable Map

3-5 Product of Sums Simplification

3-6 NAND and NOR Implementation

3-7 Other Two- Level Implementations

3-8 Don’t Care Conditions

- **3-9 The Tabulation Method**
- **3-10 Determination of Prime Implicants**
- **3-11 Selection of Prime Implicants**

3-1 THE MAP METHOD

- One can simplify Boolean function using boolean algebra.

- However, this procedure of minimization is awkward because it lacks specific rules to predict each succeeding step in the manipulative process.

- The map method provides a simple straightforward procedure for minimizing Boolean functions.

- This method may be regarded either as a pictorial form of a truth table or as an extension of the Venn diagram.

- The map method, first proposed by Veitch and modified by Karnaugh, is also known as the “Veitch diagram” or the “Karnaugh map.”

- The map is a diagram of squares.

- Each square represents a minterm.

- It is known that any Boolean function can be expressed as sum of minterms.
- Hence, a Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function.
- In fact, the map presents a visual diagram of all possible ways a function may be expressed in standard form.
- By recognizing various patterns, the user can derive alternative algebraic expressions for the same function, from which he can select the simplest one.
- We shall assume that the simplest algebraic expression is any one in a sum of products or products of sums that has a minimum number of literals.

3-2 TWO- AND THREE-VARIABLE MAPS

A two variable map is shown in Fig 3-1(a)

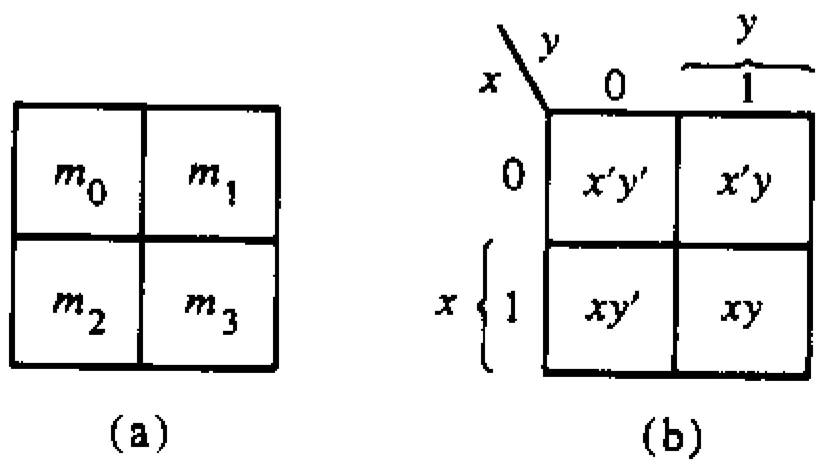


FIGURE 3-1
Two-variable map

- There are four minterms for two variables.
- Therefore, the map consists of four squares, one for each minterm.
- The map is redrawn in (b) to show the relationship between the squares and the two variables.

•The 0’s and 1’s marked for each row and each column designate the values of the variables x and y, respectively.

•Notice that x appears primed in row 0 and unprimed in row 1.

•Similarly, y appears primed in column 0 and unprimed in column 1.

•Let us we mark the squares whose minterms belong to a given function.

•Then, the two-variable map become another useful way to represent any one of the 16 Boolean functions of two variables.

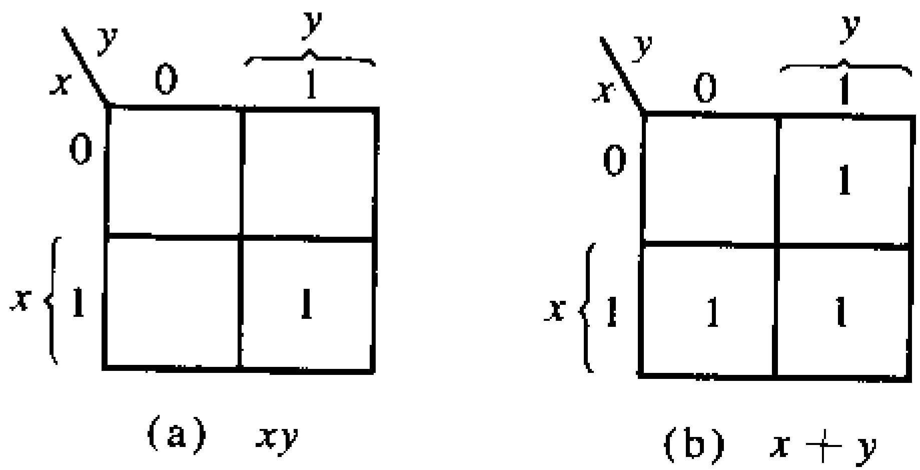


FIGURE 3-2
Representation of functions in the map

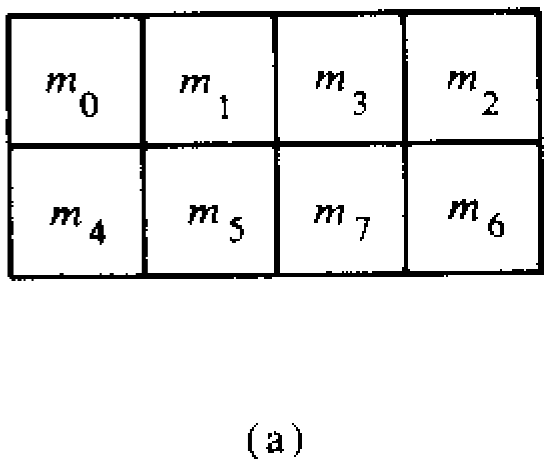
As an example, the function xy is shown in Fig 3-2(a).

Since xy is equal to m3, a 1 is placed inside the square that belongs to m3.

Similarly, the function x + y is represented in the map of Fig 3-2 (b) by three squares marked with 1's.

- These squares are found from the minterms of the function:
    - $x + y = x'y + xy' + xy = m1 + m2 + m3$

- A three variable map is shown in Fig 3-3.



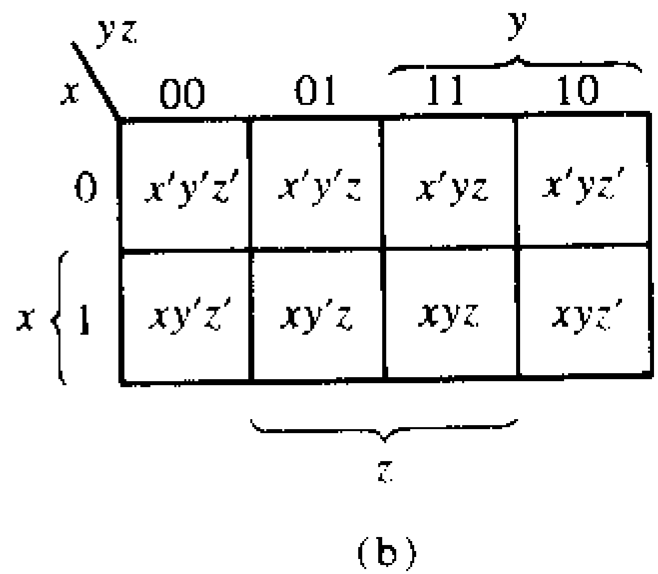


FIGURE 3-3
Three-variable map

- There are eight minterms for three binary variables.

- Therefore, a map consists of eight squares.

- Note that the minterms are not arranged in a binary sequence, but in a sequence similar to the Gray code.

- Only one bit changes from 1 to 0 or from 0 to 1 as we move to adjacent square.

- The map drawn in part(b) is marked with numbers in each row and column to show the relationship between the squares and the three variables.

- For example, the square assigned to m_5 corresponds to row 1 and column 101.

- When these two numbers are read as 101, they give the decimal number 5, hence minterm 5(m_5).

• square $m_5 = xy'z$ can be considered to be in the row marked x and the column belonging to y'z (column 01).

•Note that there are four squares where each variable is equal to 1 and four where each is equal to 0.

•The variable appears unprimed in those four squares where it is equal to 1 and primed in those squares where it is equal to 0.

•To understand the usefulness of the map for simplifying Boolean functions, we must be aware of the basic property possessed by adjacent squares.

•Any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in the other.

•For example, m_5 and m_7, lie in two adjacent squares.

- Variable y is primed in m_5 and unprimed in m_7.
- The other two variables are the same in both squares.
- From the postulates of Boolean Algebra,
    - $m5 + m7 = xy'z + xyz = xz(y + y') = xz$
- Variable y can be removed when the where the two minterms are ORed.
- Thus, any two minterms in adjacent squares that are ORed together will cause the removal of the different variable.
- **Example 3-1**
- Simplify the Boolean function
    - $F(x, y, z) = \sum(2, 3, 4,5)$

- First, a 1 is marked in each minterm that represents the function.
- This is shown in Fig 3-4.
- The squares for minterms 010, 011, 100, and 101 are marked with 1's.

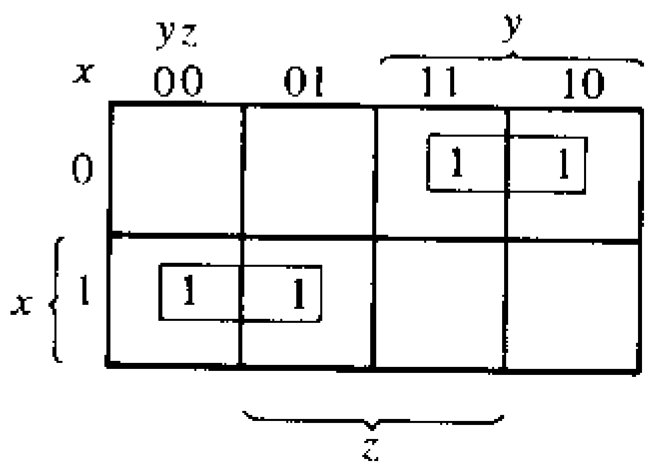


FIGURE 3-4

Map for Example 3-1; $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

- The next step is to find possible adjacent squares.
- These are indicated in the map by two rectangles, each enclosing two 1's.
- The upper right rectangle represents the area enclosed by $x'y$.
- This is determined by observing that the two-area is in row 0, corresponding to x'.
- And the last two column correspond to y.
- Similarly, the lower left rectangle represents the product term xy'.
- The second row represents x.
- And the two left columns represent y'.

- The logical sum of these two product terms gives the simplified expression:
  - $F = x'y + xy'$
- There are cases where the two squares in the map are considered to be adjacent even though they do no touch each other.
- In Fig 2-3, m_0 is adjacent to m_2 and m_4 is adjacent to m_6.
- The reason is that the minterms differ by one variable.
- Consequently, we must modify the definition of adjacent squares to include this and other similar cases.
- This is done by considering the map as being drawn on a surface where the right and left edges touch each other to form adjacent squares.

- **Example 3-2**
- Simplify the Boolean function
  - $F(x, y, z) = \sum(3,4,6,7)$

- The map for this function is shown in Fig 3-5.

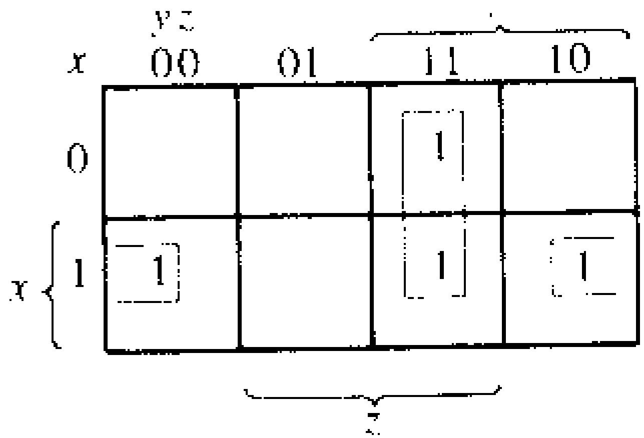


FIGURE 3-5
Map for Example 3-2; $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

- There are four squares marked with 1's, one for each minterm.
- Two adjacent squares are combined in the third column to give yz.
- The remaining two squares with 1's are also adjacent by the new definition.
- They are shown in the diagram with their values enclosed in half rectangles.
- These two squares when combined give the term xz'.
- The simplified function becomes
    - F = yz + xz'

- Consider now any combination of four adjacent squares in the three- variable map.

- Any such combination represents the logic sum of four minterms.

- And it results in and expression of only one literal.

- For example, the logic sum of four adjacent minterms 0, 2, 4, and 6 reduces to a single literal term z'.

- $m_1 + m_2 + m_4 + m_6 = x'y'z + x'yz' + xy'z' + xyz'$

$$= x'z'(y + y') + xz'(y + y')$$
$$= x'z' + xz'$$
$$= z'(x + x') = z'$$

- The number of adjacent squares that may be combined must always represent a number that is a power of two such as 1, 2, 4, and 8.

- As a larger number of adjacent squares are combined, we obtain a product term with fewer literals.

- One square represents one minterm, giving a term of three literals.

- Two adjacent squares represent a term of two literals.

- Four adjacent squares represent a term of one literal.

- Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

- **Example 3-3**
- Simplify the Boolean function
    - $F(x, y, z) = \sum(0, 2, 4, 5, 6)$

- The map for F is shown in Fig 3-6.

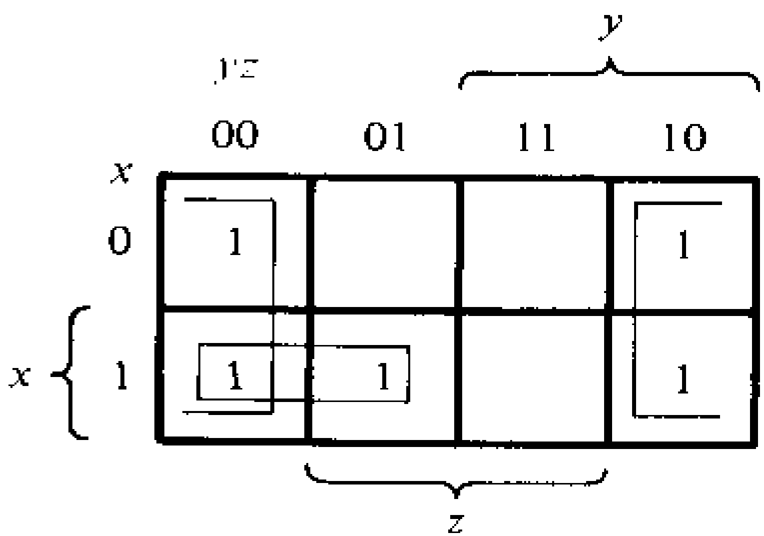


FIGURE 3-6
Map for Example 3-3; $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

- 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 the minterm 5 is combined with an adjacent square that has already been used once.

- This is permitted and is necessary since the two adjacent squares give the two literal term xy’.

- The simplified function is:

    - $F = z' + xy'$

- If a function is not expressed in sum of minterms, it is possible to use the map to obtain the minterms of the function.

- And then we can simplify the function to an expression with a minimum number of terms.

- It is necessary to make sure that the algebraic expression is in sum of products form.

- Each product term can be plotted in the map in one, two, or more squares.

- The minterms of the function are then read directly from the map.

- **Example 3-4**
- Given the following Boolean function:
  - $F = A'C + A'B + AB'C + BC$

- (a) Express it in sum of minterms
- (b) Find the minimal sum of products expression

- Three product terms in the expression have two literals and are represented in a three variable map by two squares each.

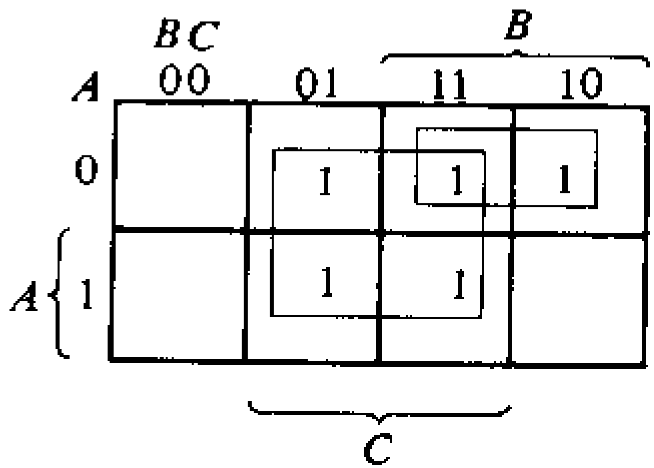


FIGURE 3-7
Map for Example 3-4; $A'C + A'B + AB'C + BC = C + A'B$

- The two squares corresponding to the first term A'C are found in Fig 3-7.

- The coincidence of A'(first row) and C(two middle columns) to give squares 001 and 011.

•Note that when marking 1’s in the squares, it is possible to 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.

•But square 011 is common with the first term A’C, 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 tem 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 mp to be 1, 2, 3, 5, 7

- The function can be expressed in sum of minterms form:
  - $F(A, B, C) = \sum(1, 2, 3, 5, 7)$
- The simplified form results in the following expression:
  - $F = C + A'B$
- **Exercises**
- **Simplify the following function using three variable maps:**
- **(a) $F(x, y, z) = \sum(3,5,6,7)$**
- **(b) $F(x, y, z) = \sum(0, 2, 3, 4, 6)$**

3-3 Four Variable Map

- The map for Boolean function of four binary variables is shown in Fig 3-8.

- In(a) are listed the 16 minterms and the squares assigned to each.

- In (b) the map is redrawn to show the relationship with four variables.

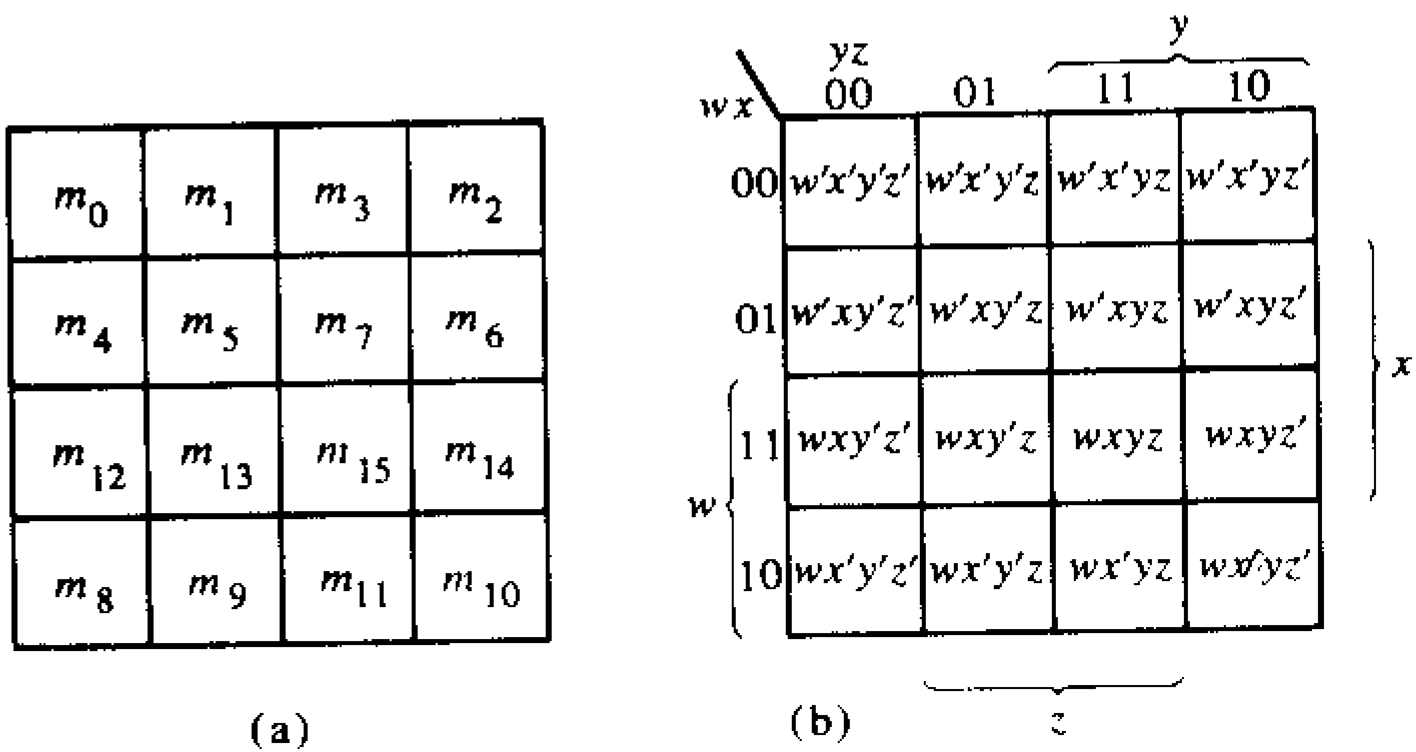


FIGURE 3-8

Four-variable map

•The rows and column are numbered in a reflected-code sequence, with only one digit changing value between two adjacent rows and column.

•The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number.

•For example, the numbers of the third row (11) and the second column (01), when combined give the binary number 1101, equivalent to 13.

•Thus, the square in the third row and the second column represent the minterm m_{13}.

•The map minimization is similar as before where adjacent squares are defined to be squares next to each other.

•The map is considered to lie on a surface with the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares.

•For example, m_0 and m_2 form adjacent squares, as do m_3 and m_{11}.

•One square represent one minterm, giving a term of four literals.

•Two adjacent squares represent a term of three literals.

•Four adjacent squares represent a term of two literals.

•Eight adjacent squares represent a term of one literal.

•Sixteen adjacent squares represent the function equal to 1.

•No other combination of squares can simplify the function.

•The following two examples show the procedure used to simplify four-variable Boolean functions.

- **Example 3-5**
- Simplify the Boolean function

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

- 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.

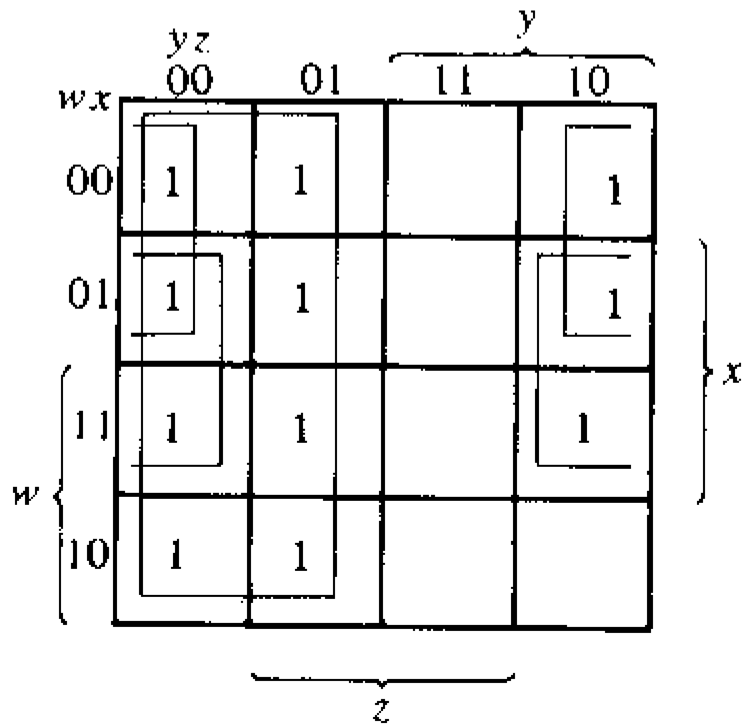


FIGURE 3-9

Map for Example 3-5; $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

- The larger the number of squares combined, the smaller 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 of four literals), we combine it with squares already used to form an area of four adjacent squares.

- These squares comprise the two middle rows and the two end columns.

- They give the term xz'.

- The simplified function is
    - $F = y' + w'z' + xz'$
- **Example 3-6**
- Simplify the Boolean function
    - $F = A'B'C' + B'CD' + A'BCD' + AB'C'$
- The area of the map covered by this function consists of the squares marked with 1's in Fig 3-10.
- This function has four variables, and as expressed, consists of three terms, each with three literals, and one term of four literals.
- Each term of three literals is represented in the map by two squares.

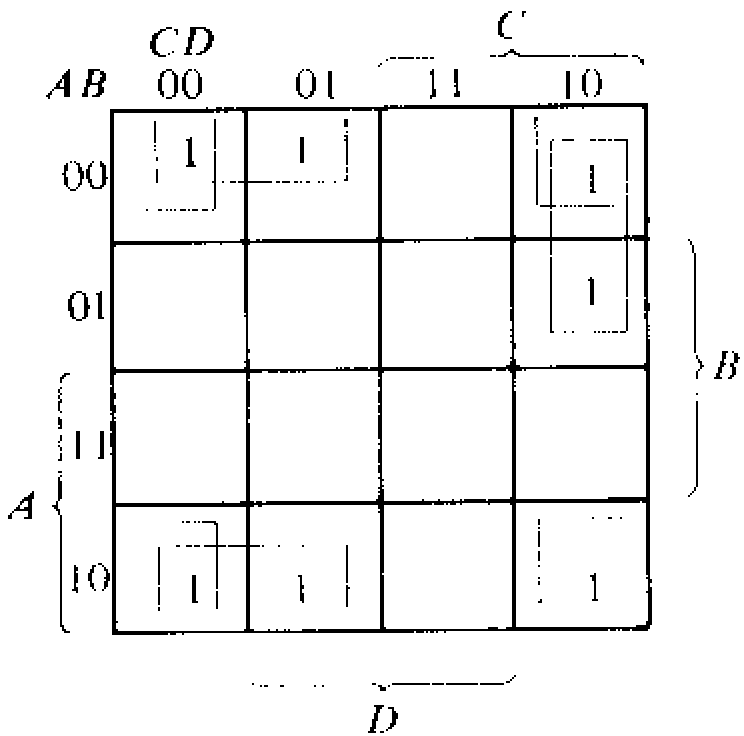


FIGURE 3-10

Map for Example 3-6; $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

- For example, A’B’C’ is represented in squares 0000 and 0001.

- The function can be simplified in the map by taking the 1’s in the four corners to give the term B’D’.

- This is possible because these four squares are adjacent when the map is drawn in a surface with top and bottom or left and right edges touching one another.

- The two left hand 1’s in the top row are combined with the two 1’s in the bottom to give the term B’C’.

- The remaining 1 may be combined in a two-square area to give the term A’CD’.

- The simplified function is:

    - F = B’D’ + B’C’ + A’CD’

Prime Implicants

- When choosing adjacent squares in a map, we must ensure that all the minterms of the function are covered when combining the squares.

- It is also necessary to minimize the number of terms in the expression.

- And it is important to avoid any redundant terms whose minterms are already covered by other terms.

- Sometimes there may be two or more expressions that satisfy the simplification criteria.

- The procedure for combining squares in the map may be made more systematic if we understand the meaning of the terms prime implicants.

- 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 **essential prime implicant.**

•The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares.

•This means that a single 1 on a map represents a prime implicant if it is not adjacent to any other 1's.

•Two adjacent 1's form a prime implicant provided they are not within a group of four adjacent squares.

•Four adjacent 1's form a prime implicant if they are not within a group of eight adjacent 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.

- Consider the following four-variable Boolean function:

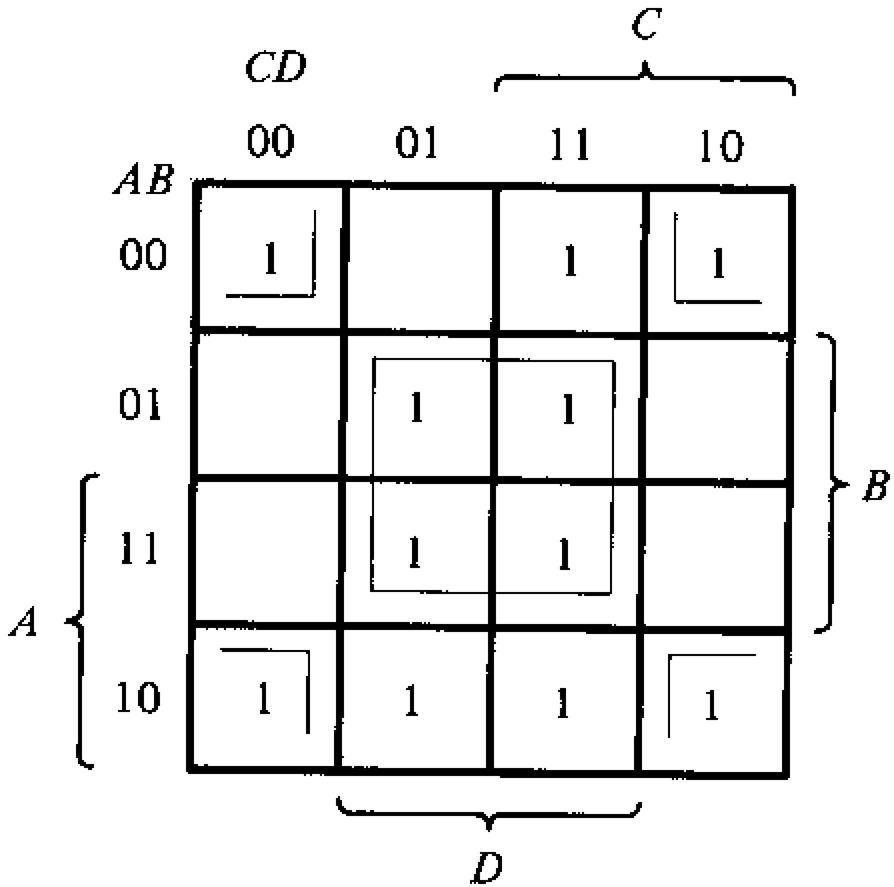
    - $F(A, B, C, D) = \sum(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

- The minterms of the function are marked with 1's in the maps of Fig. 3-11.

- Part(a) of the figure shows two essential prime implicants.

- One term is essential because there is only one way to include minterms m_0 within four adjacent squares.

- These four squares define the term B'D'.



(a) Essential prime implicants
BD and $B'D'$

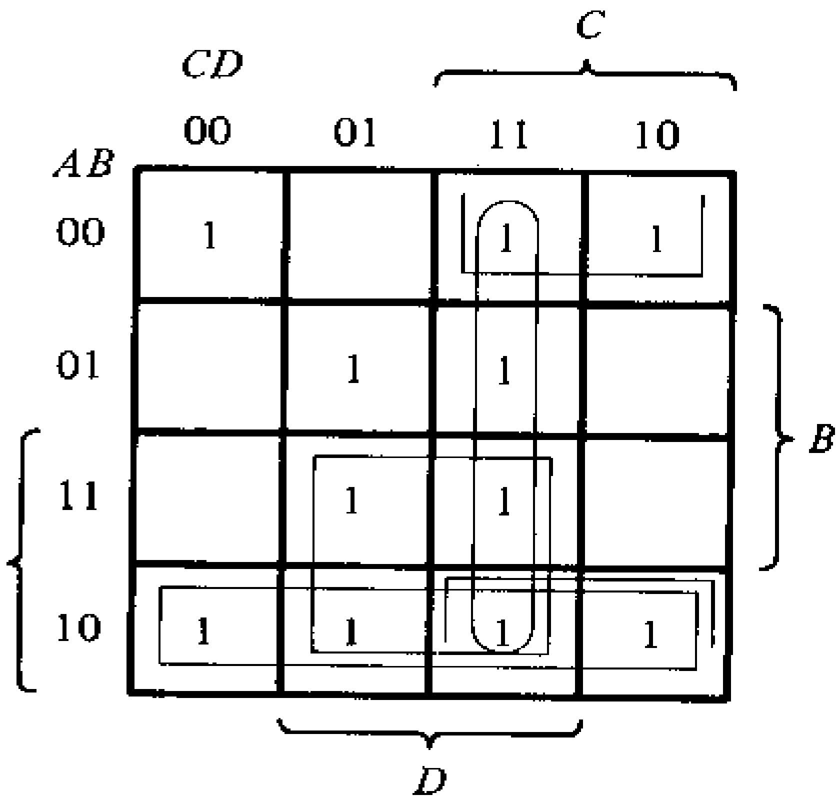
FIGURE 3-11
Simplification using prime implicants

•Similarly, there is only one way that minterm m5 can be combined with four adjacent squares and this gives the second term BD.

•The two essential prime implicants cover eight minterms.

•The remaining three minterms, m3, m9, and m11, must be considered next.

•Fig 3-11(b) shows all possible ways can be covered with prime implicants.



(b) Prime implicants CD, $B'C$, AD, and AB'

FIGURE 3-11

Simplification using prime implicants

•Minterm m_3 can be covered with either prime implicant CD or B'C.

•Minterm m_9 can be covered with either AD or AB'.

•Minterm m_{11} is covered with any one of the four prime implicants.

•The simplified expression is obtained from the logical sum of the two essential prime implicants and any two of the four prime implicants.

•There are four possible ways that the function can be expressed with four product terms of two literals each:

•$F = BD + B'D' + CD + AD$
$= BD + B'D' + CD + AB'$
$= BD + B'D' + B'C + AD$
$= BD + B'D' + B'C + AB'$

•The above example has demonstrated that the identification of the prime implicants in the map helps in determining the alternatives that are available in determining a simplified expression.

•The procedure for finding the simplified expression from the map requires that we first determine all the essential prime implicants.

•The simplified expression is obtained from the logical sum of all the essential prime implicants plus other prime implicants.

•These remaining prime implicants may be needed to cover any remaining minterms not covered by the essential prime implicants.

•Sometimes, there may be more than one way of combining squares and each combination may produce an equally simplified expression.

3-4 FIVE-VARIABLE MAP

•A five-variable map needs 32 squares and a six-variable map needs 64 squares.

•A five-variable map is shown in Fig 3-12.

•It consists of 2 four-variable maps with variables A, B, C, D, and E.

•Variable A has a different value from the first map to the second map.

•The left-hand four –variable map represents the 16 squares where A =0.

•The right-hand four-variable map represents the squares where A=1.

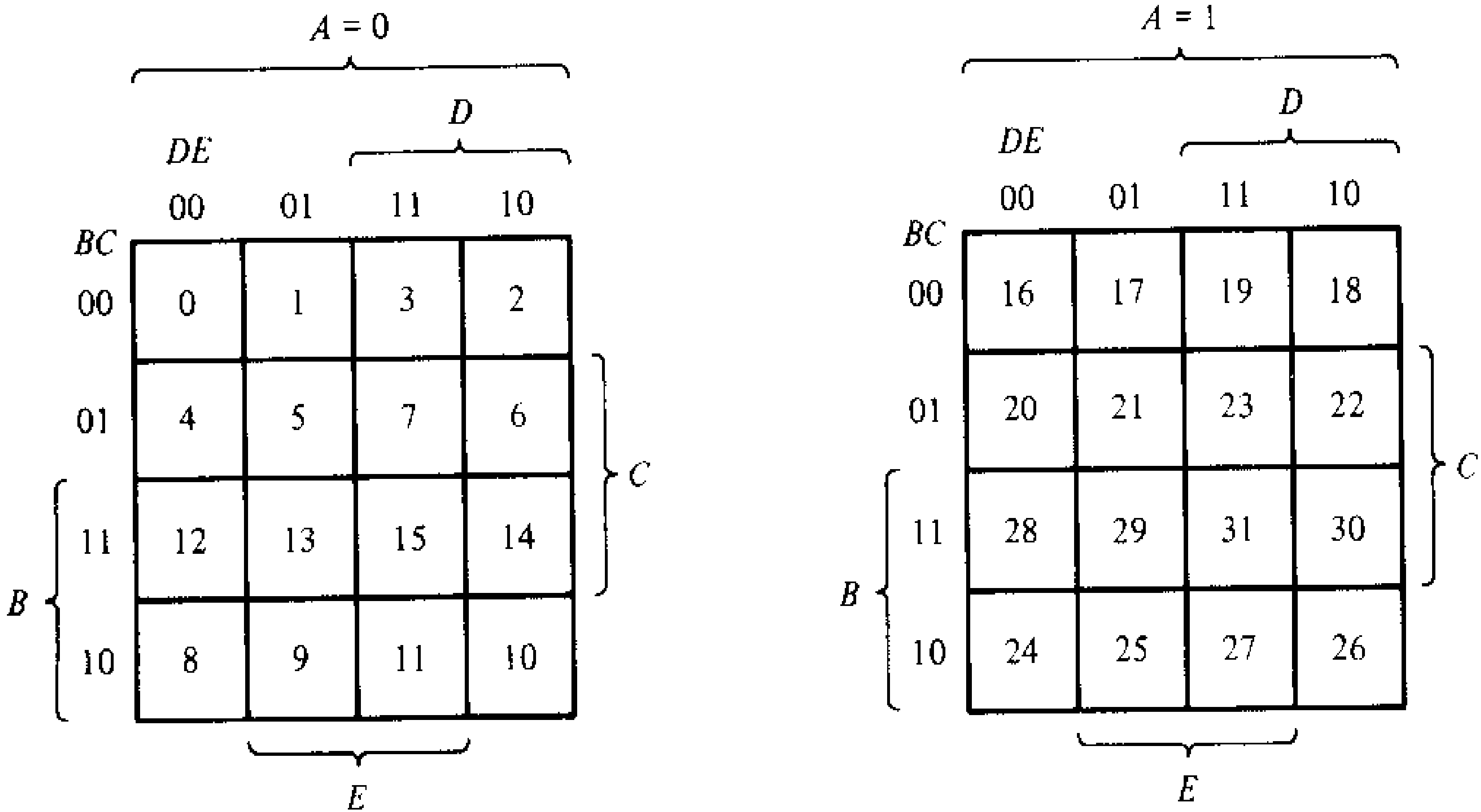


FIGURE 3-12
Five-variable map

•Minterms 0 through 15 belong with A=0 and minterms 16 through 31 with
•A =1.

•Each four-variable map retains the previously defined adjacency when taken separately.

•In addition, each square in the A=0 map is adjacent to the corresponding square in the A =1 map.

•For example, minterm 4 is adjacent to minterm 20 and minterm 15 to 31.

•The best way to visualize this new rule for adjacent squares is to consider the two half maps as being one on top of the other.

•Any two squares that fall one over the other are considered adjacent.

•By following the procedure used for the five-variable map, it is possible to construct a six-variable map with 4 four-variable map to obtain the required 64 squares.

•Maps with six or more variables need too many squares and are impractical to use.

•From inspection, it is possible to show that 2^k(2 to the power of k) adjacent squares, for k=0, 1, 2,….n, in an n-variable map, will represent an area that gives a term of n-k literals.

•For the above statement to be realizable, n must be greater than k.

•When n=k, the entire area of the map is combined to give the identity function.

•Table 3-1 shows the relationship between the number of adjacent squares and the number of literals in the term.

TABLE 3-1
The Relationship Between the Number of Adjacent Squares and the Number of Literals in the Term

k	Number of adjacent squares 2^k	Number of literals in a term in an n-variable map $n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$
0	1	2	3	4	5	6	7
1	2	1	2	3	4	5	6
2	4	0	1	2	3	4	5
3	8		0	1	2	3	4
4	16			0	1	2	3
5	32				0	1	2
6	64					0	1

- Eight adjacent squares combine an area in the five-variable map to give the term of two literals.

- **Example 3-7**
- Simplify the Boolean function
  - $F(A, B,C,D,E) = \sum(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$

- The five variable map for this function is shown in Fig. 3-13.

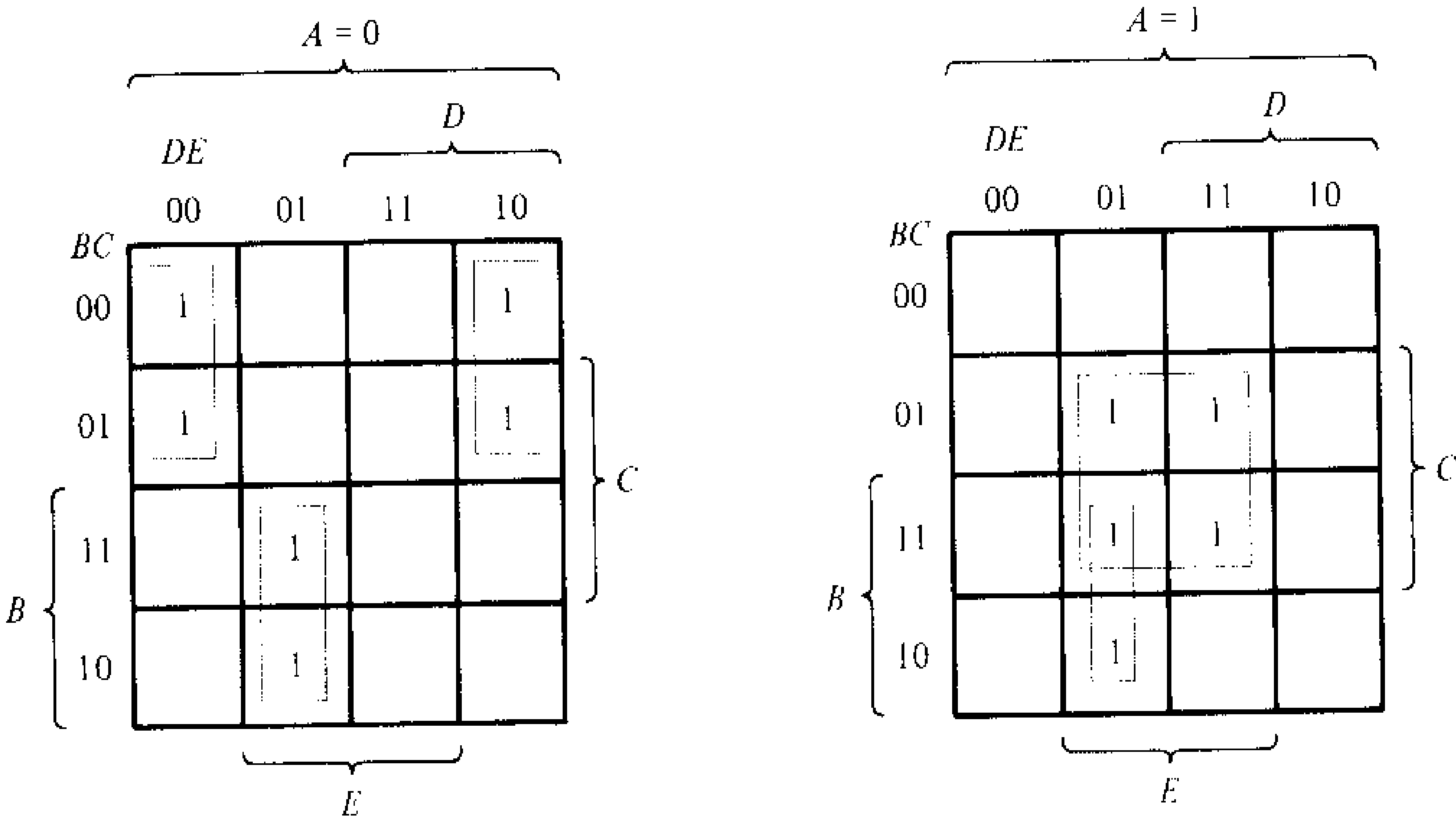


FIGURE 3-13

Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

- There are six minterms from 0 to 15 that belong to the part of the map with A=0.

- The other five minterms belong with A=1.

- Four adjacent squares in the A=0 map are combined to give three-literal term A'B'E'.

- Note that it is necessary to include A' with the term because all the squares are associated with A=0.

- The two squares in the column 01 and the last two rows are common to both parts of the map.

- Therefore, they constitute four adjacent squares and give the three-literal term B'DE.

- Variable A is not included here because the adjacent squares belong to both A=0 and A=1.

- The term ACE is obtained from the four adjacent squares that are entirely within A=1 map.

- The simplified function is the logical sum of the three terms:
    - F = A'B'E' + BD'E + ACE

3-5 PRODUCT OF SUMS SIMPLIFICATION

- The minimized Boolean functions derived from the map in all previous examples were expressed in the sum of products form.

- With a minor modification, the product of sum form can be obtained.

- The procedure for obtaining a minimized function in product of sums follows from the basic properties of Boolean functions.

- The 1's placed in the squares of the map represent the minterms of the function.

- If we mark the empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified expression of the complement of the function.

- The complement of the complement of the function (F' ') gives us back the function F.

- Because of the generalized DeMorgan's theorem, the function so obtained is automatically in the product of sums form.

- **Example 3-8**
- Simplify the following Boolean function in (a) sum of products and (b) product of sums.
  - $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$

- The 1's marked in the map of Fig 3-14 represent all the minterms of the function.

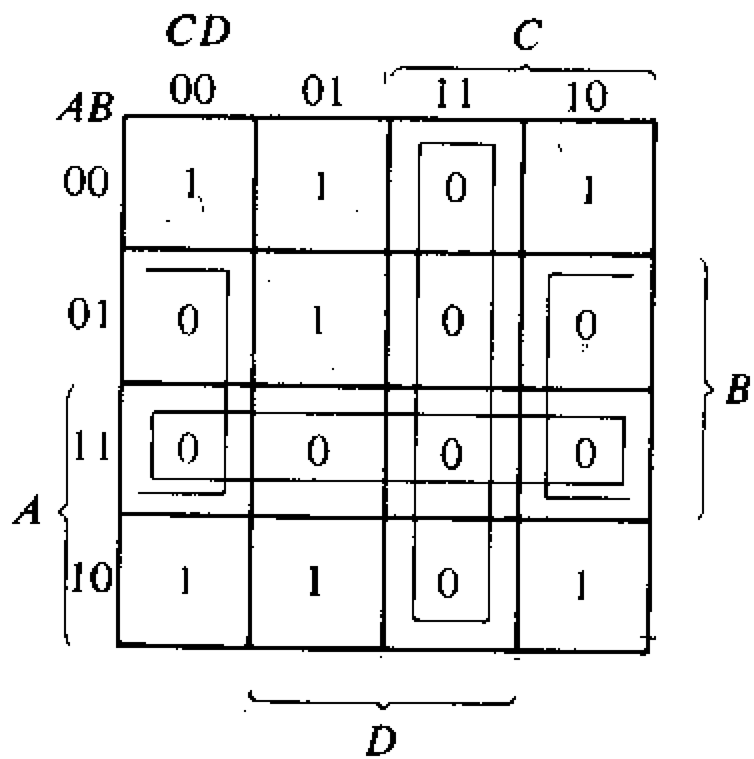


FIGURE 3-14

Map for Example 3-8; $F(A, B, C, D) = \Sigma\,(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$

- 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:

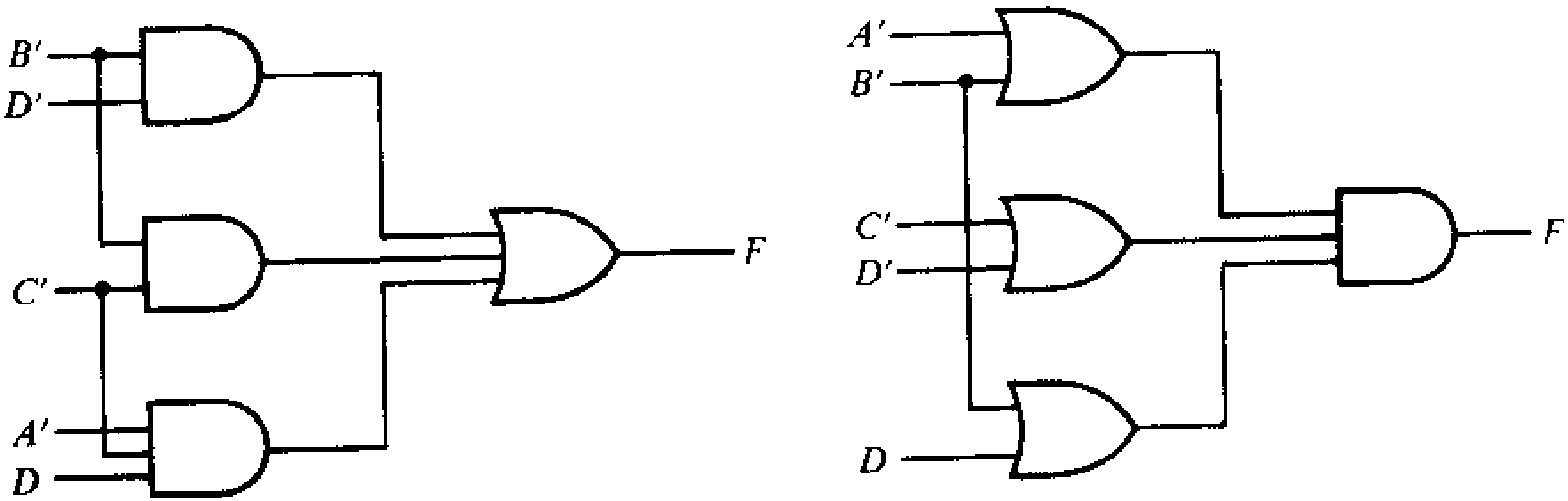
- (a) $F = B'D' + B'C' + A'C'D$

- 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 complement each literal), we obtain the simplified function in product of sums:

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

•The implementation of the simplified expression obtained in Example 3-8 is shown in Fig 3-15.



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

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

FIGURE 3-15

Gate implementation of the function of Example 3-8

- 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 form with a group of OR gates, one for each OR term.

- The outputs of the OR gates are connected to the inputs of a single AND gates.

- In each case, it is assumed that the input variables are directly available in their complement, so inverters are not needed.

- The configuration pattern established in Fig 3-15 is the general form by which any Boolean function is implemented when expressed in one of the standard forms.

- AND gates are connected to a single OR gate when in sum of products.
- OR gates are connected to a single AND gate when in product of sums.
- Either configuration form two level of gates.
- Therefore, the implementation of a function in a standard form is said to be a two-level implementation.
- Example 3-8 showed the procedure for obtaining the products of sum simplification when the function is expressed in sum of minterms canonical form.
- The procedure is also valid when the function is originally expressed in product of maxterms canonical form.
- Consider, for example, the truth table that defines the function F in Table 3-2.

TABLE 3-2
Truth Table of Function *F*

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

- In sum of minterms, this function is expressed as
  - $F(x, y, z) = \sum(1,3,4,6)$
- In product of maxterms, it is expressed as
  - $F(x, y, z) = \prod(0, 2, 5, 7)$
- The 1's in the function represent the minterms, and the 0's represent the maxterms.
- The map for this function is shown in Fig 3-16.

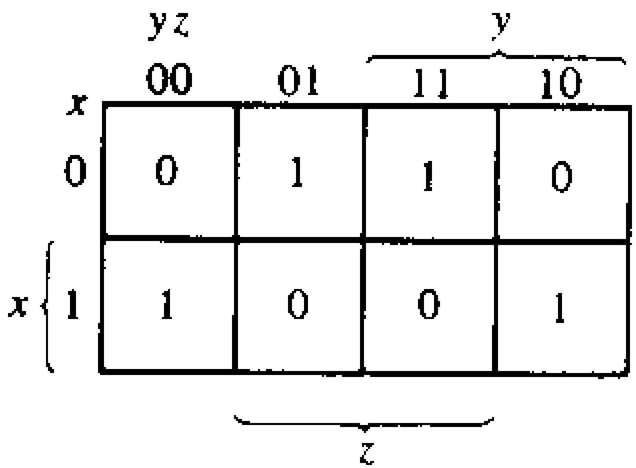


FIGURE 3-16
Map for the function of Table 3-2

- One can start simplifying this function by first marking the 1's for each minterm that function is a 1.

- The remaining squares are marked by 0's.

- If, however, the product of maxterms is initially given, one can start marking 0's in those squares 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.

- Taking the complement of F’, we obtain the function in product of sums:
  - $F = (x' + z')(x + z)$

- To enter a function expressed in product of sums in the map, take the complement of the function and from it find the squares to be marked by 0's.

- For example, the function

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

- can be entered in the map by first taking its complement:
    - $F' = ABC + B'D'$

- And then marking 0's in the squares representing the minterms of F'.

- The remaining squares are marked by 1's.

Exercises

- Simplify the following using a four variable map.
  - $F(w, x, y, z) = \sum(0, 1, 2, 3, 4, 5, 7, 11, 15)$
- Simplify the following Boolean expression using a four-variable map.
  - $F = w'z + xz + x'y + wx'z$
- Find the minterms of the following Boolean function
  - $F = xy + yz + xy'z$
- Simplify the following Boolean function using a five-variable map
  - $F(A, B, C, D, E) = \sum(0,2,3,4,5,6,7,11,15,16,18,19,23,27,31)$

3-6 NAND AND NOR IMPLEMENTATION

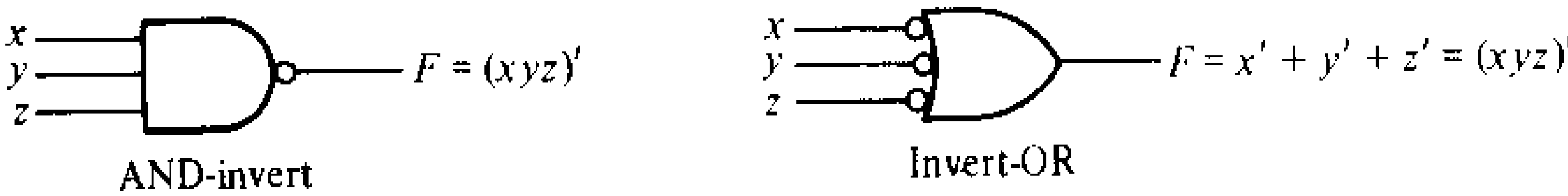
- Digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates.

- NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic circuits.

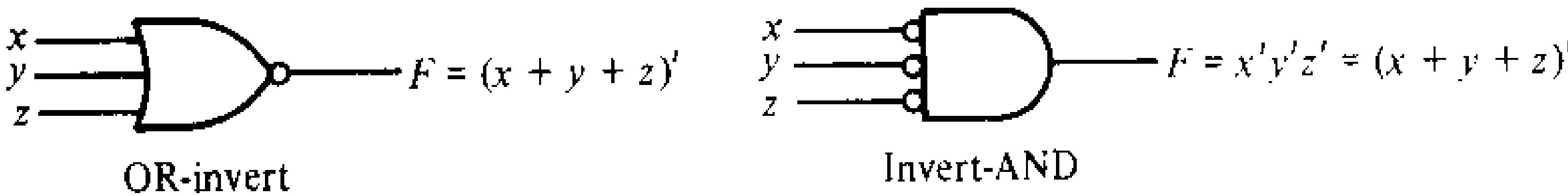
- Rules and procedures have been developed for the conversion from Boolean functions given in terms of AND, OR, and NOT into equivalent NAND and NOR diagrams.

- To facilitate the conversion to NAND and NOR logic, it is convenient to define two other graphic symbols for these gates.

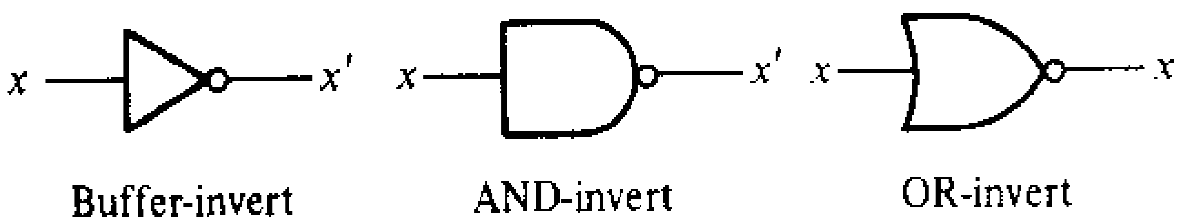
- Two equivalent symbols for the NAND gate are shown in Fig 3-17(a).



(a) Two graphic symbols for NAND gate.



(b) Two graphic symbols for NOR gate.



(c) Three graphic symbols for inverter.

FIGURE 3-17

Graphic symbols for NAND and NOR gates

- The AND-invert symbol has been defined previously and consists of an AND graphic symbol followed by a small circle.

- Instead, it is possible to represent a NAND gate by an OR graphic symbol preceded by small circles in all inputs.

- The invert-OR symbol for the NAND gate follows from DeMorgan’s theorem and from the convention that small circles denote complementation.

- Similarly, there are two graphic symbols for the NOR gate, as shown in Fig. 3-17(b).

- The OR-invert is the conventional symbol.

- But using DeMorgan’s theorem, we come up with the Invert-AND symbol.

- A one input NAND or NOR gate behaves like an inverter.

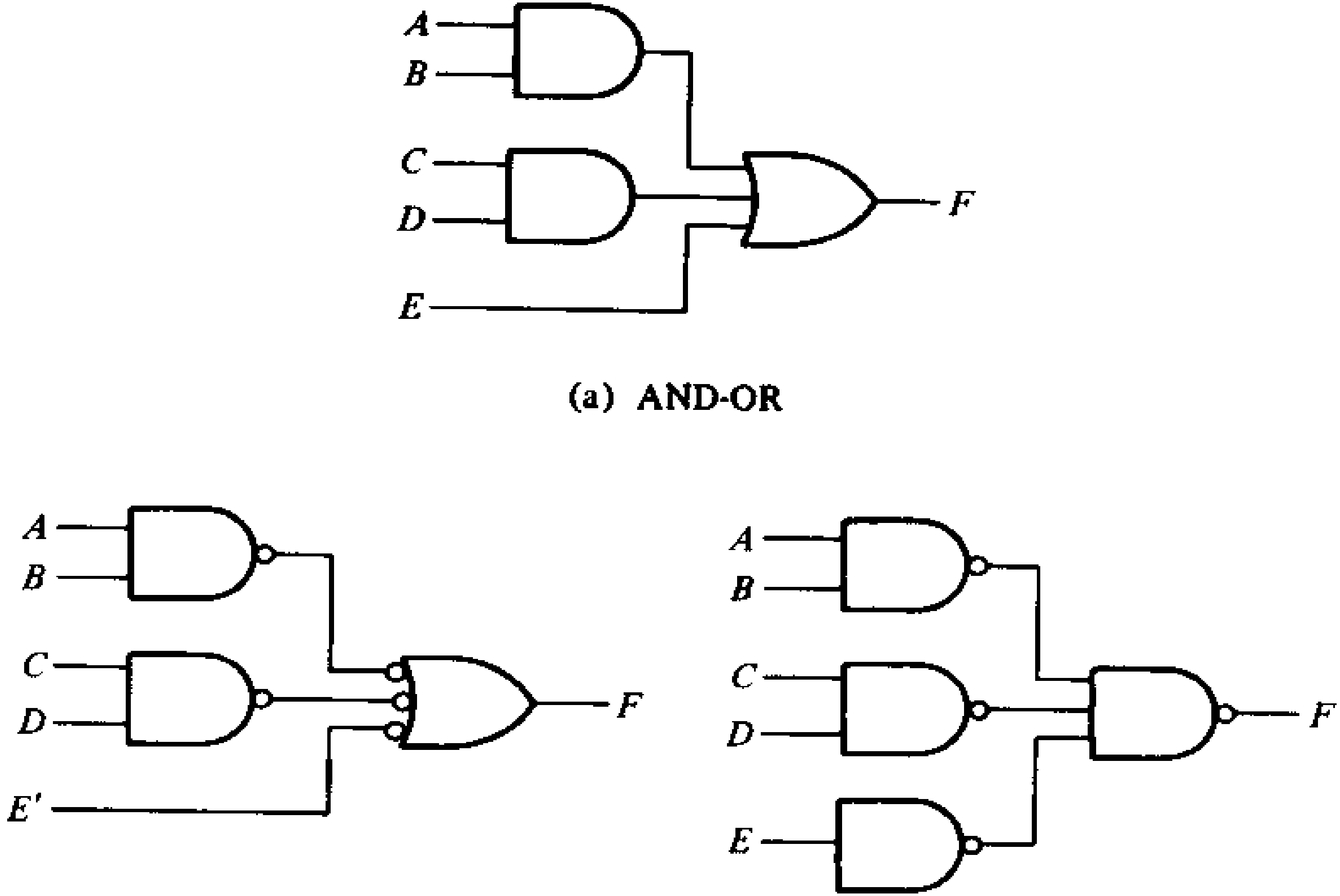
•An inverter gate can be drawn in three different ways (Fig 2-17(c)).

•The small circles in all inverter symbols can be transferred to the input terminal without changing the logic of the gate.

•**NAND implementation**

•The implementation of a Boolean function with NAND gates requires that the function be simplified in the sum of products form.

•To see the relationship between the sum of products expression and its equivalent NAND implementation, consider Fig 3-18.



(a) AND-OR

(b) NAND-NAND

(c) NAND-NAND

FIGURE 3-18

Three ways to implement $F = AB + CD + E$

- All three diagrams are equivalent and implement the function:
    - F = AB + CD + E

- The function is implemented in Fig. 3-18(a) in sum of products form with AND and OR gates.

- In (b) the AND gates are replaced by NAND gates and the OR gate is replaced by a NAND gate with an invert-OR symbol.

- The single E variable E is complemented and applied to the second-level invert-OR gate.

- Two circles on the same line represent double complementation and both can be removed.

- The complement of E goes through a small circle that complements the variable again to produce the normal value of E.

- Removing the small circles in the gates of Fig. 3-18(b) produces the circuit in (a).

- Therefore, the two diagrams implement the same function and are equivalent.

- In Fig 3-18(c), the output NAND gate is redrawn with the conventional symbol.

- The one-input NAND gate complements variable E.

- It is possible to remove this inverter and apply E’ directly to the input of the second-level NAND gate.

- The diagram in (c) is equivalent to the diagram in (b), which in turn is equivalent to the diagram in (a).

- The NAND implementation in Fig 3-18(c) can be verified algebraically.

- The NAND function it implements can be easily converted to a sum of products form by using DeMorgan's theorem:

    - $F = [(AB)' \cdot (CD)' \cdot E']' = AB + CD + E$

- Hence, a Boolean function can be implemented with two-level of NAND gate, known as two-level implementation.

- The rule for obtaining the NAND logic diagram from a Boolean function is as follows:

- 1. Simplify the function and express it in sum of products.

- 2. Draw a NAND gate for each product term of the function that has at least two literals
    - The inputs to each NAND gate are the literals of the term.

    - This constitutes a group of first-level gates.

•3. Draw a single NAND gate (using the AND-invert or invert-OR graphic symbol) in the second level.

- •The inputs to this NAND gate comes from the outputs of first-level gates.

•4. A term with a single literal requires an inverter in the first level or may be complemented and applied as an input to the second-level NAND gate.

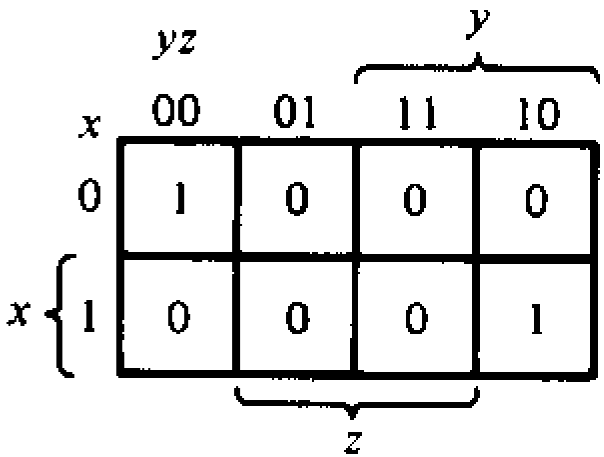
•**Example 3-9**

•Implement the following function with NAND gates:

- •$F(x, y, z) = \sum(0, 6)$

•The first step is to simplify the function in sum of products form.

•This is attempted with the map shown in Fig. 3-19(a).



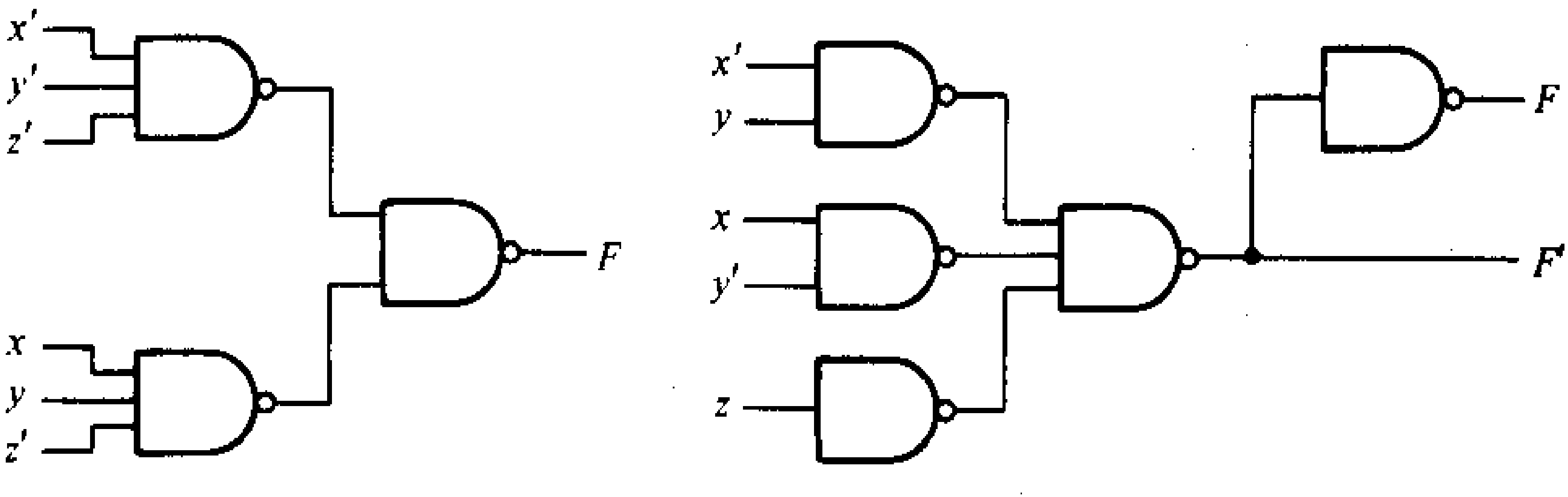
$F = x'y'z' + xyz'$

$F' = x'y + xy' + z$

(a) Map simplification in sum of products.

- There is only two 1’s in the map, and they can not be combined.

- The simplified function in sum of products for this example is
  - F = x’y’z’ + xyz’
- The two level NAND gate implementation is shown in Fig 3-19(b).



(b) $F = x'y'z' + xyz'$

(c) $F' = x'y + xy' + z$

FIGURE 3-19

Implementation of the function of Example 3-9 with NAND gates

- Next we try to simplify the complement of the function in sum of products.
- This is done by combining the 0's in the map:
  - $F' = x'y + xy' + z$
- The two-level NAND gate for generating F' is shown in Fig. 3-19 (c).
- If output F is required, it is necessary to add a one-input NAND gate to invert the function.
- This gives a three-level implementation.
- The one-input NAND gate associated with the single variable z can be removed provided the input is changed to z'.

- **NOR IMPLEMENTATION**
- The NOR function is the dual of the NAND function.

- For this reason, all procedures and rules for NOR logic are the duals of the corresponding procedures and rules developed for NAND logic.

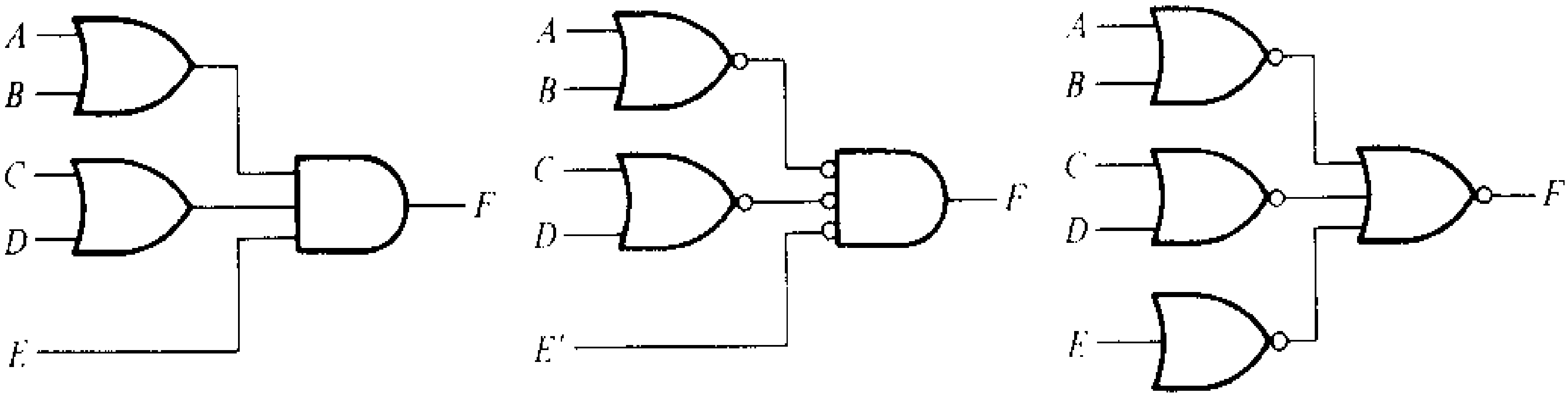
- The implementation of a Boolean function with NOR gates requires that the function be simplified in product of sums form

- A product of sums expression specifies a group of OR gates for the sum terms, followed by an AND gate to produce the product.

- The transformation from the OR-AND to the NOR-NOR diagram is depicted in
- Fig 3-20.

- It is similar to the NAND transformation discussed previously, except that now we use the product of sums expression

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



(a) OR-AND (b) NOR-NOR (c) NOR-NOR

FIGURE 3-20

Three ways to implement $F = (A + B)(C + D)E$

- The rule for obtaining the NOR logic diagram from a Boolean function can be derived from this transformation.

- It is similar to the three-step NAND rule, except that the simplified expression must be in the product of sums.

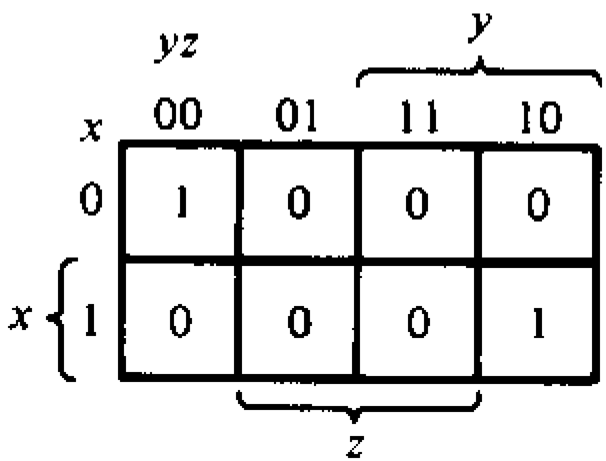
- And the terms for the first-level NOR gates are sum terms.

- A term with a single literal requires a one-input NOR or inverter gate or may be complemented and applied directly to the second-level NOR gate.

- A second way to implement a function with NOR gates would be to use the expression for the complement of the function in product of sum.

- This will give a two-level implementation for F’ and a three-level implementation if the normal output F is required.

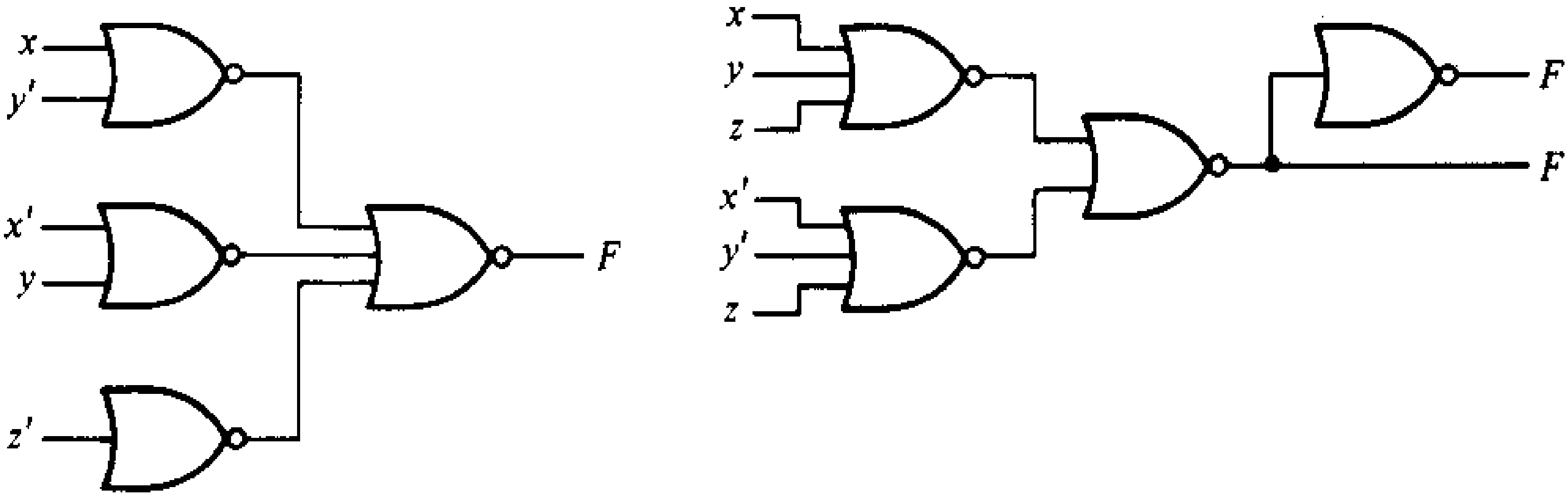
- **Example 3-10**
- Implement the function of Example 3-9 with NOR gates.
  - $F(x, y, z) = \sum(0, 6)$



$$F = x'y'z' + xyz'$$
$$F' = x'y + xy' + z$$

(a) Map simplification in sum of products.

- We combine the 0's in the map to obtain
  - $F' = x'y + xy' + z$
- This is the complement of the function in sum of products.
- Complement F' to obtain
  - $F = (x + y')(x' + y)(z')$
- As required for NOR implementation.
- The two-level implementation with NOR gates is shown in Fig 3-12 (a).



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

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

FIGURE 3-21

Implementation with NOR gates

- A second implementation is possible from the complement of the function in product of sums.
- For this case, first combine the 1's in the map to obtain
    - $F = x'y'z' + xyz'$
- This is the simplified expression in sum of products.
- Complement this function to obtain the complement of the function in product of sums as required for NOR implementation:
    - $F' = (x + y + z)(x' + y' + z)$
- The two-level implementation for F' is shown in Fig 3-21 (b).
- If output F is desired, it can be generated with an inverter in the third level.

•Table 3-3 summarizes the procedures for NAND or NOR implementation.

•We should always remember to simplify the function in order to reduce the number of gates in the implementation.

TABLE 3-3
Rules for NAND and NOR Implementation

Case	Function to simplify	Standard form to use	How to derive	Implement with	Number of levels to F
(a)	F	Sum of products	Combine 1's in map	NAND	2
(b)	F'	Sum of products	Combine 0's in map	NAND	3
(c)	F	Product of sums	Complement F' in (b)	NOR	2
(d)	F'	Product of sums	Complement F in (a)	NOR	3

3-7 OTHER TWO-LEVEL IMPLEMENTATIONS

- **Nondegenerate Forms**

- We will consider all the possible two-level combinations of gates.

- We consider four types of gates: AND, OR, NAND and NOR.

- If we assign one type of gate for the first level and one type for the second level, we find that there are 16 possible combinations of two-level forms.

- The same type of gate can be in the first and second levels, as in NAND-NAND implementation.

- Eight of these combinations are said to be degenerate forms because they degenerate to a single operation.

- This can be seen from a circuit with AND gates in the first level and an AND gate in the second level.

- The output of the circuit is merely the AND function of all input variables.

- The other eight nondegenerate forms produce an implementation in sum of products or product of sums.

- The eight nondegenerate forms are:
  - AND-OR OR-AND
  - NAND-NAND NOR-NOR
  - NOR-OR NAND-AND
  - OR-NAND AND-NOR

- The first gate listed in each of the forms constitutes a first level in the implementation.

- The second gate listed is a single gate placed in the second level

- Note that any two forms listed in the same line are the duals of each other.
- The AND-OR and OR-AND forms and the NAND-NAND and NOR-NOR were discussed earlier.
- The remaining four forms are investigated in this section.
- **AND-OR-INVERT Implementation**
- The two forms NAND-AND and AND-NOR are equivalent and can be treated together.
- Both perform the AND-OR-INVERT function, as shown in Fig. 3-23.

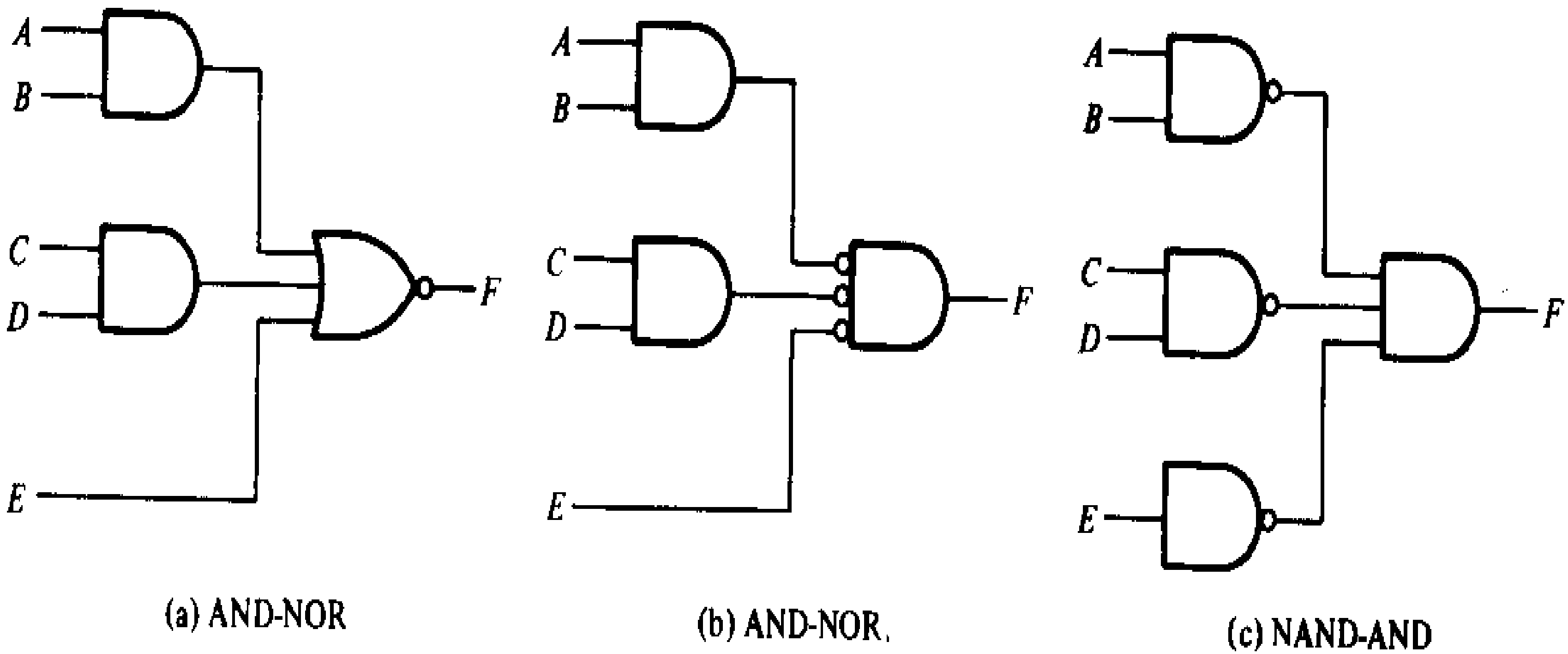


FIGURE 3-23

AND-OR-INVERT circuits; $F = (AB + CD + E)'$

- The AND-NOR form resembles the AND-OR form with an inversion done by the small circle in the output of the NOR gate.

- It implements the function
  - $F = (AB + CD + E)'$

- By using the alternate graphic symbol for the NOR gate, we obtain the diagram of Fig. 3-23(b).

- Note that the single variable E is not complemented because the only change made is in the graphic symbol of the NOR gate.

- Now we move the circles from the input terminal of the second-level gate to the output terminals of the first-level gates.

- An inverter is needed for the single variable to maintain the circle.

- Alternatively, the inverter can be removed as long we complement E.

•The circuit of Fig. 3-23(c) is a NAND-AND form.

•An AND-OR implementation requires an expression in sum of products.

•The AND-OR-INVERT implementation is similar except for the inversion.

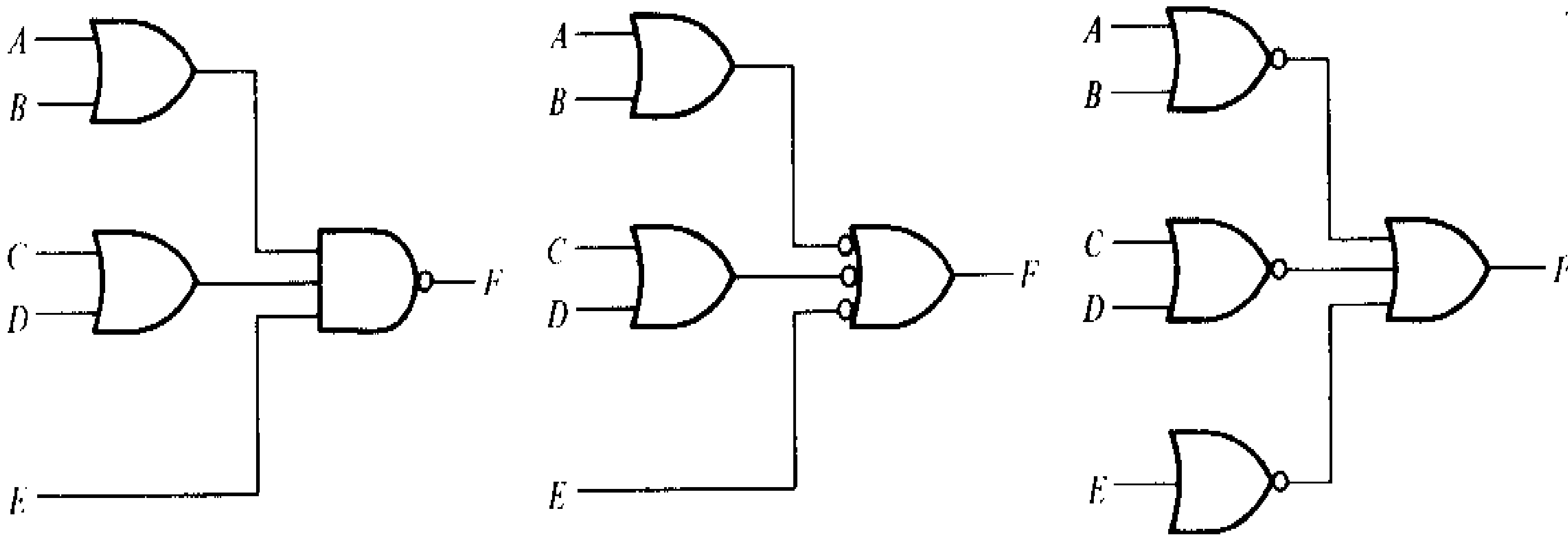
•Therefore, first the complement of the function is simplified in sum of products (by combining the 0’s in the map).

•Then it will be possible to implement F’ with AND-OR part of the function.

•When F’ passes through the always present output inversion (the INVERT part), it will generate the output F of the function.

•**OR-AND-INVERT Implementation**

•The OR-NAND and NOR-OR forms perform the OR-AND-INVERT function. As depicted in Fig 3-24.



(a) OR-NAND (b) OR-NAND (c) NOR-OR

FIGURE 3-24

OR-AND-INVERT circuits; $F = [(A + B)(C + D)E]'$

- The OR-NAND form resembles the OR-AND form, except for the inversion done by the circle in the NAND gate.

- It implements the function:
    - $F=[(A + B)(C+D)E]'$

- By using the alternate graphic symbol for the NAND gate, we obtain the diagram of Fig 3-24(b).

- The circuit in (c) is obtained by moving the small circles form the inputs of the second-level gate to the outputs of the first level gates.

- The circuit of Fig. 3-24(c) is a NOR-OR form.

- The OR-AND-INVERT implementation requires an expression in product of sums.

•If the complement of the function is simplified in product of sums, we can implement F’ with the OR-AND part of the function.

•When F’ passes through the INVERT part, we obtain the complement F’, or F, in the output.

•**Tabular Summary and Example**

•Table 3-4 summarizes the procedures for implementing a Boolean function in any one of the four two-level forms.

•Because of the INVERT part in each case, it is convenient to use the simplification of F’ (the complement) of the function.

•When F’ is implemented in one of these forms, we obtain the complement of the function in the AND-OR or OR-AND form

•The four two level forms invert this function, giving an output that is the complement of F’ (F).

TABLE 3-4
Implementation with Other Two-Level Forms

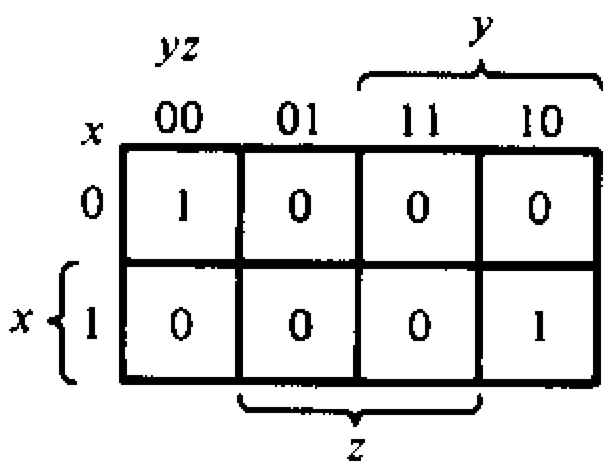
Equivalent nondegenerate form (a)	Equivalent nondegenerate form (b)*	Implements the function	Simplify F' in	To get an output of
AND-NOR	NAND-AND	AND-OR-INVERT	Sum of products by combining 0's in the map	F
OR-NAND	NOR-OR	OR-AND-INVERT	Product of sums by combining 1's in the map and then complementing	F

*Form (b) requires a one-input NAND or NOR (inverter) gate for a single literal term.

Example 3-11

•Implement the function F = x’y’z’ + xyz’ in the four two-level forms.

•The karnaugh map for the function is shown.



$F = x'y'z' + xyz'$

$F' = x'y + xy' + z$

(a) Map simplification in sum of products.

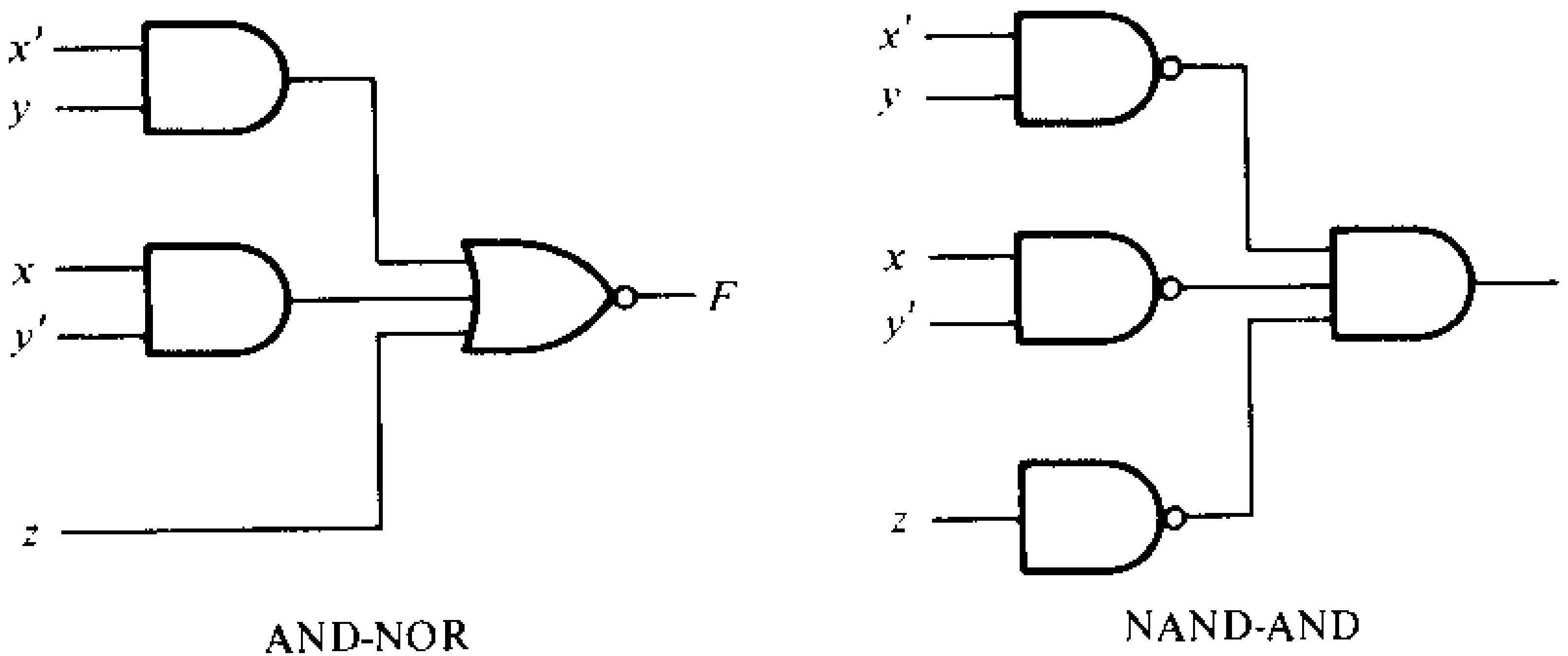
•The complement of the function is simplified in sum of products by combining the 0’s in the map.

- •F’ = x’y + xy’ + z

- The normal output for this function can be expressed as
    - $F = (x'y + xy' + z)'$

- Which is in the AND-OR-INVERT form.

- The AND-NOR and NAND-AND implementations are shown in Fig. 3-25(a).



(a) $F = (x'y + xy' + z)'$

•Note that a one-input NAND or inverter gate is needed in the NAND-AND implementation, but not in the AND-NOR case.

•The OR-AND-INVERT forms require a simplified expression of the complement of the function in product of sums.

•To obtain this expression, we must first combine the 1's in the map

- •$F = x'y'z' + xyz'$

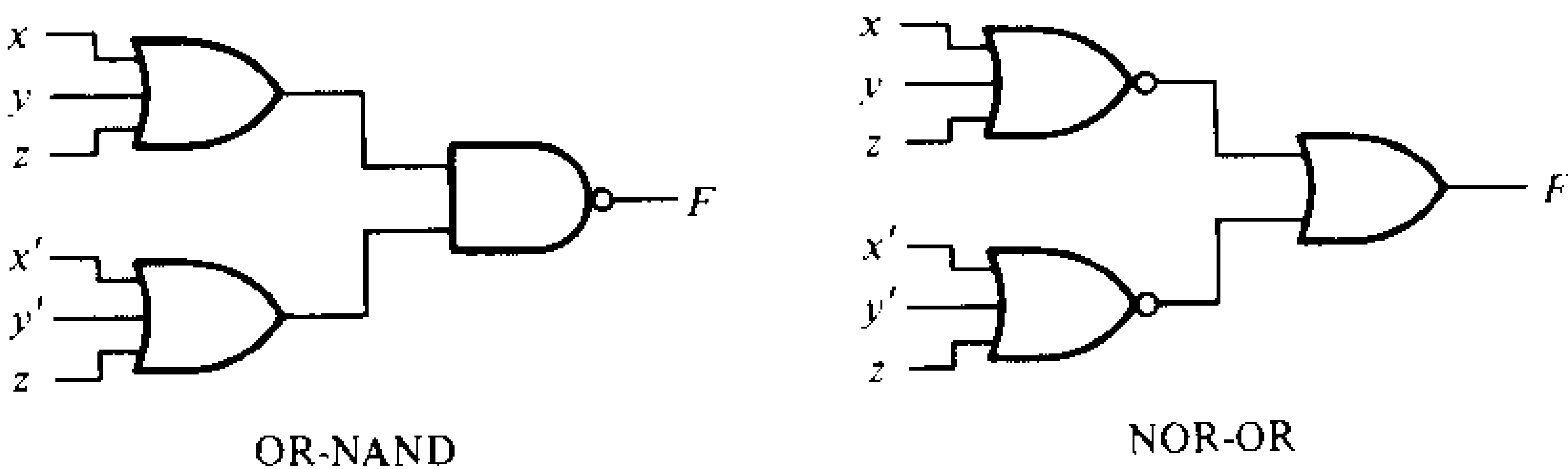
•Then we take the complement of the function

- •$F' = (x + y + z)(x' + y' + z)$

•The normal output F can now be expressed in the form

- •$F = [(x + y + z)(x' + y' + z)]'$

•Which is in the OR AND-INVERT form. From this expression we can implement the function in OR-NAND and NOR-OR forms as shown in Fig 3-25(b).



(b) $F = [(x + y + z)(x' + y' + z)]'$

FIGURE 3-25

Other two-level implementations

- **Exercises**
- 1-Simplify the following expressions and implement it with two-level NAND gate circuits:
- $F= BD + BCD' + AB'C'D$

- 2-Simplify the following function and implement it with three-level NOR gate circuit:

$F = wx' + y'z' + w'yz'$

- 3. Simplify and implement Problem no.2 with two level NOR gate circuit.

- 3. Implement the function F with the following two-level forms: NAND-AND, AND-NOR, OR-NAND and NOR-OR.
    - $F(A, B, C,D) = \sum(0, 1,2,3,4,8,9,12)$

3-8 DON'T-CARE CONDITIONS

•The logical sum of the minterms associated with a Boolean function specifies the condition under which the function is equal to 1.

•The function is equal to 0 for the rest of the minterms.

•This assumes that all the combinations of the values for the variables of the function are valid.

•In practice, there are some applications where the function is not specified for certain combinations of the variables.

•As an example, the four-bit binary code for the decimal digits has six combinations that are not used and consequently are considered unspecified.

•Functions that have unspecified outputs for some input combinations are called incompletely specified functions.

•22

- In most applications, we simply don’t care what value is assumed by the function for the unspecified minterms.

- For this reason, it is customary to call the unspecified minterms of a function don’t care conditions.

- These don’t-care conditions can be used on the map to provide further simplification of the Boolean expression.

- It should be noted that don’t-care minterm is a combination of variables whose logical value is not specified.

- It cannot be marked with a 1 in the map because it would require that the function be 1 for such combination.

- Similarly, putting a 0 on the square requires the function be 0.

- To distinguish the don’t-care condition from 1’s and 0’s, an X is used.

- Thus, an X inside a square in the map indicates that we don’t care whether the value of 0 or 1 is assigned to F for the particular minterm.

- When choosing adjacent squares to simplify the function in the map, the don’t care minterms may be assumed to be either 0 or 1.

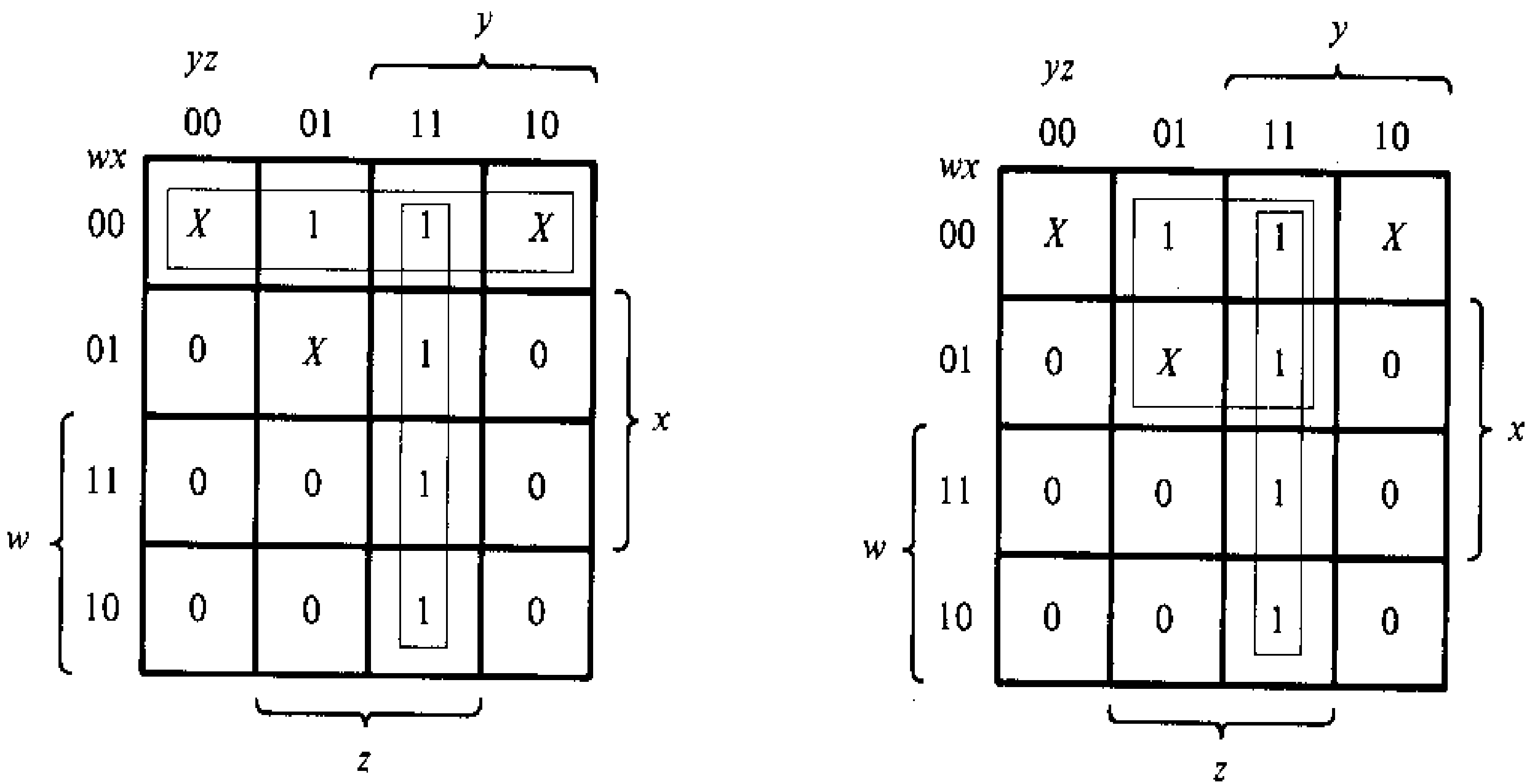
- When simplifying the function, we can choose to include each don’t care minterm with either the 1’s or the 0’s, depending on which combination gives the simples expression.

- **Example 3-12**
- Simplify the Boolean function
  - $F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$
- That has the don’t-care conditions
  - $d(w, x, y, z) =\sum(0, 2, 5)$

•The minterms of F are the variable combinations that make the function equal to 1.

•The minterms of d are the don’t-care minterms that may be assigned either 0 or 1.

•The map simplification is shown in Fig. 3-26.



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

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

FIGURE 3-26

Example with don't-care conditions

•The minterms of F are marked by 1’s, those of d are marked by X’s, and the remaining squares are filled with 0’s.

•To get the simplified expression in sum of products, we must include all the five 1’s in the map.

•But we may or may not include any of the X’s, depending on the way the function is simplified.

•The term yz covers the four minterms in the third column.

•The remaining minterm m_1 can be combined with minterm m_3 to give the three-literal term w’x’z.

•However, by including one or two adjacent X’s we can combine four adjacent squares to give a two-literal form.

•In part (a) of the diagram, don't-care minterms 0 and 2 are included with 1's, which results in the simplified function

- F = yz + w'x'

•In part (b), don't-care minterm 5 is included with the 1's and the simplified function now is

- F = yz + w'z

•Either one of the above expressions satisfies the conditions states for this example.

•The above example has shown that the don't-care minterms in the map are initially marked with X's and are considered as being either 0 or 1.

•The choice between 0 and 1 is made depending on the way the incompletely specified function is simplified.

- Once the choice is made, the simplified function so obtained will consist of a sum of minterms that includes those minterms that were initially unspecified and have been chosen to be included with the 1's.

- Consider the two simplified expressions obtained in Example 3-12:

    - $F(w, x, y, z) = yz + w'x' = \sum(0, 1, 2, 3, 7, 11, 15)$
    - $F(w, x, y, z) = yz + w'z = \sum(1, 3, 5, 7, 11, 15)$

- Both expressions include minterms 1, 3, 7, 11 and 15 that make the function equal to 1.

- The don't-care minterms 0, 3, and 5 are treated differently in each expression.

- The first expression includes minterms 0 and 2 with the 1's and leaves minterm 5 with the 0's.

- The second expression includes minterm 5 with 1's and leaves minterms 0 and 2 with the 0's.

- The two expressions represent two functions that are algebraically unequal.

- Both cover the specified minterms of the function, but each covers different don't-care minterms.

- As far as the incompletely specified function is concerned, either expression is acceptable since the only difference is in the value of F for the don't-care minterms.

- It is also possible to obtain a simplified product of sums expression for the function of Fig. 3-26.

- In this case , the only way to combine the 0's is to include don't-care minterms 0 and 2 with the 0's to give a simplified complemented function:
    - $F' = z' + wy'$

•Taking the complement of F' gives the simplified expression in product of sums:

•$F(w, x, y, z) = z'(w + y') = \sum(1, 3, 5, 7, 11, 15)$

•For this case, we include minterms 0 and 2 with the 0's and minterm 5 with 1's.

•3-9 THE TABULATION METHOD

•The map method of simplification is convenient as long as the number of variables does not exceed five or six.

•As the number of variables increases, the excessive number of squares prevents a reasonable selection of adjacent squares.

•The obvious disadvantage of the map is that it is essentially trial-and-error procedure that relies on the ability of the human to recognize certain patterns.

•For functions of six or more variables, it is difficult to be sure that the best selection has been made.

•The tabulation method overcomes this difficulty.

•It is a specific step-by-step procedure that is guaranteed to produce a simplified standard-form expression for a function.

•It can be applied to problems with many variables and has the advantage of being suitable for machine computation.

•However, it is quite tedious for human use and is prone to mistakes because of its routine process.

•The tabulation method was first formulated by Quine and later improved by McCluskey.

•It is also known as the Quine-McCluskey method.

3-10 DETERMINATION OF PRIME IMPLICANTS

- The starting point of the tabulation method is the list of minterms that specify the function.

- The first tabular operation is to find the prime implicants by using a matching process.

- This process compares each minterm with every other minterm.

- If two minterms differ in only one variable, that variable is removed and a term with one less literal is found.

- This process is repeated for every minterm until the exhaustive search is completed.

- The matching-process cycle is repeated for those new terms just found.

•Third and further cycles are continued until a single pass through a cycle yields no further elimination of literals.

•The remaining terms and all the terms that did not match during the process comprise the prime implicants.

Example 3—13

Simplify the following Boolean function by using the tabulation method.

$$F(w, x, y, z) = \sum(0, 1, 2, 8, 10, 11, 14, 15)$$

Step1: Group binary representation of the minterms according to the number of 1's contained, as shown in Table 3-5, column (a).

TABLE 3-5
Determination of Prime Implicants for Example 3-13

(a)

	w x y z	
0	0 0 0 0	√
1	0 0 0 1	√
2	0 0 1 0	√
8	1 0 0 0	√
10	1 0 1 0	√
11	1 0 1 1	√
14	1 1 1 0	√
15	1 1 1 1	√

(b)

	w x y z	
0, 1	0 0 0 –	
0, 2	0 0 – 0	√
0, 8	– 0 0 0	√
2, 10	– 0 1 0	√
8, 10	1 0 – 0	√
10, 11	1 0 1 –	√
10, 14	1 – 1 0	√
11, 15	1 – 1 1	√
14, 15	1 1 1 –	√

(c)

	w x y z
0, 2, 8, 10	– 0 – 0
0, 8, 2, 10	– 0 – 0
10, 11, 14, 15	1 – 1 –
10, 14, 11, 15	1 – 1 –

This is done by grouping the minterms into five sections separted by horizontal lines.

•The first section contains the number with no 1's in it.'

•The second section contains those numbers that have only one 1.

•The third, fourth, and fifth sections contain those binary numbers with two, three, and four 1's, respectively.

•The decimal equivalents of the minterms are also carried along for identification.

•Step 2: Any two minterms that differ from each other by only one variable can be combined, and the unmatched variable removed.

•Two minterm numbers fit into this category if they both have the same bit value in all positions except one.

•The minterms in one section are compared with those of the next section down only, because two terms differing by more than one bit cannot match.

•The minterm in the first section is compared with each of the three minterms in the second section.

•If any two numbers are the same in every position but one, a check is placed to the right of both minters to show that hey have been used.

•The resulting term, together with the decimal equivalent of the selected minterms, is listed in column (b) of the table.

•The variable eliminated during the matching is denoted by a dash in its position.

•In this case, m_0(0000) combines with m_1 (0001) to form (000-).

- This combination is equivalent to the algebraic operation
  - $m_0 + m_1 = w'x'y'z' + w'x'y'z = w'x'y'$

- Minterm m_0 also combines with m_2 to form (00 -0) and with m_8 to form (-000).

- The result of this comparison is entered into the first section of column (b).

- The minterms of sections two and three of column (a) are next compared to produce the terms listed in the second section of column (b).

- All other section of (a) are similarly compared and subsequent sections formed in (b).

- This exhaustive comparing process results in the four sections of (b).

- Step 3:
- The terms of column (b) have only three variables.

- A 1 under the variable means it is unprimed, a 0 means it is primed, and a dash means the variable is not included in the term.

- The searching and comparing process is repeated for the terms in column (b) to form the two-variable terms of column (c).

- Again, terms in each section need to be compared only if they have dashes in the same position.

- Note that the term (000-) does not match with any other term.

- Therefore, it has no check mark at its right.

- The decimal equivalents are written on the left-hand side of each entry for identification purposes.

- The comparing process should be carried again in column (c) and in subsequent columns as long as proper matching is encountered.

•In the present example, the operation stops at the third column.

•Step 4:
•The unchecked terms in the table form the prime implicants.

•In this example, we have the term w’x’y’ (000_) in column (b), and the terms x’z’ (-0-0) and wy (1-1-) in column (c).

•Note that each term in column (c) appears twice in the table, and as long as the term forms a prime implicant, it is unnecessary to use the same term twice.

•The sum of the prime implicants gives a simplified expression for the function.

•This is because each checked term in the table has been taken into account by an entry of a simpler term in subsequent column.

•Therefore, the unchecked entries (prime implicants) are the terms left to formulate the function.

•For the present example, the sum of prime implicants gives the minimized function in sum of products:

  •F = w'x'y' + x'z' + wy

•It is worth comparing this answer with that obtained by the map method.

•Figure 3-27 shows the map simplification of this function.

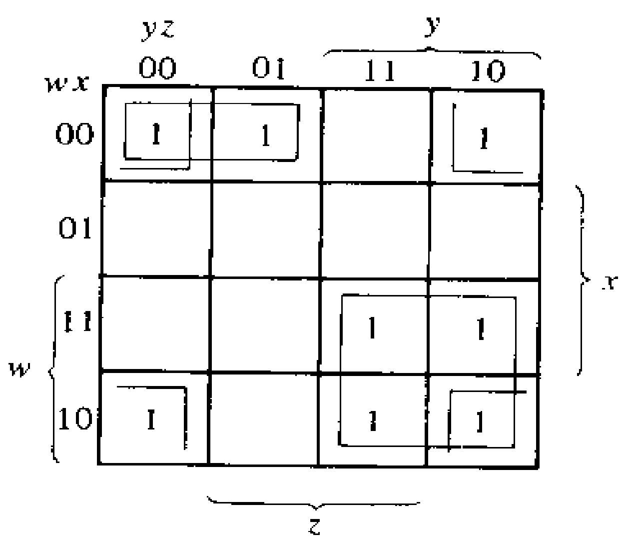


FIGURE 3-27

Map for the function of Example 3-13;
$F = w'x'y' + x'z' + wy$

- The combinations of adjacent squares give the three prime implicants of the function.
- The sum of these three terms is the simplified expression in sum of products.
- The work when using the tabulation method is reduced if the comparing is done with decimal numbers instead of binary.
- A method will be used that uses subtraction of decimal numbers instead of the comparing and matching of binary numbers.
- We note that each 1 in a binary number represents the coefficient multiplied by a power of 2.
- When two minterms are the same in every position except one, the minterm with the extra 1 must be larger than the number of the other minterm by a power of 2.

- Therefore, two minterms can be combined if the number of the first minterm differs by a power of 2 from a second larger number in the next section down the table.

- We shall illustrate this procedure by repeating Example 3-13.

- As shown in Table 3-6, column (a), the minterms are arranged in sections as before except that now only the decimal equivalents of the minterms are listed.

- The process of comparing minterms is as follows: Inspect every two decimal numbers in adjacent sections of the table.

- If the number in the section below is greater than the number in the section above by a power of 2 (I.e., 1, 2, 4, 8, 16, etc), check both numbers to show that they have been used, and write them down in column (b).

•The paid of numbers transferred to column (b) includes a third number in parentheses that designates the power of 2 by which the numbers differ.

•The number in parenthesis also tells us the position of the dash in the binary notation.

•The results of all comparisons of column (a) are shown in column (b).

•The comparison between adjacent sections in column (b) is carried out in a similar fashion, except that only those terms with the same number in parenthesis are compared.

•The pair of numbers In section must differ by a power of 2 from the pair of numbers in the next section.

•And the numbers in the next section below must be greater for the combination to take place.

- In column (c), write all four decimal numbers with the two numbers in parentheses designating the positions of the dashes.

TABLE 3-6
Determination of Prime Implicants of Example 3-13 with Decimal Notation

(a)		(b)		(c)
0	√	0, 1 (1)		0, 2, 8, 10 (2, 8)
		0, 2 (2)	√	0, 2, 8, 10 (2, 8)
1	√	0, 8 (8)	√	
2	√			10, 11, 14, 15 (1, 4)
8	√	2, 10 (8)	√	10, 11, 14, 15 (1, 4)
		8, 10 (2)	√	
10	√			
		10, 11 (1)	√	
11	√	10, 14 (4)	√	
14	√			
		11, 15 (4)	√	
15	√	14, 15 (1)	√	

- The prime implicants are those terms not checked in the table.
- These are the same as before, except that they are given decimal notation.
- To convert from decimal notation to binary , convert all decimal numbers in the term to binary and then insert a dash in those positions designated by the numbers in parentheses.
- Thus 0, 1 (1) is converted to binary as 0000, 0001; a dash in the first position of either number results in (000--).
- Similarly, 0, 2, 8, 10 (2, 8) is converted to the binary notation from 0000, 0010, 1000, and 1010, and a dash inserted in positions 2 and 8, to result in (-- 0 --0).

- **Example 3-14**
- Determine the prime implicants of the function.

  - $F(w, x, y, z) = \sum(1, 4, 6, 7, 8, 9, 10, 11, 15)$

- The minterms are grouped in sections, as shown in Table 3-7, column (a).

- The binary equivalent of the minterm is included for the purpose of counting the number of 1's.

- The binary numbers in the first section have only one 1, in the second section, two 1's, etc.

TABLE 3-7
Determination of Prime Implicants for Example 3-14

(a)			(b)			(c)
0001	1	✓	1, 9	(8)		8, 9, 10, 11 (1, 2)
0100	4	✓	4, 6	(2)		8, 9, 10, 11 (1, 2)
1000	8	✓	8, 9	(1)	✓	
			8, 10	(2)	✓	
0110	6	✓	6, 7	(1)		
1001	9	✓	9, 11	(2)	✓	
1010	10	✓	10, 11	(1)	✓	
0111	7	✓	7, 15	(8)		
1011	11	✓	11, 15	(4)		
1111	15	✓				

Prime implicants

Decimal	Binary w	x	y	z	Term
1, 9 (8)	—	0	0	1	$x'y'z$
4, 6 (2)	0	1	—	0	$w'xz'$
6, 7 (1)	0	1	1	—	$w'xy$
7, 15 (8)	—	1	1	1	xyz
11, 15 (4)	1	—	1	1	wyz
8, 9, 10, 11 (1, 2)	1	0	—	—	wx'

- The minterm numbers are compared by the decimal method and a match is found if the number in the section below is greater than that in the section above.

- If the number in the section below is smaller than the one above, a match is not recorded even if the two numbers differ by a power of 2.

- The exhaustive search in column (a) results in the terms of column (b), with all minterms in column (a) being checked.

- There are only two matches of terms in column (b).

- Each gives the same two literal term recorded in column (c).

- The prime implicants consists of all the unchecked terms in the table.

- The conversion from the decimal to the binary notation is shown at the bottom of the table.

- The prime implicants are found to be x’y’z, w’xz’, w’xy, xyz, wyz, and wx’.
- The sum of the prime implicants gives a valid algebraic expression for the function.
- However, this expression is not necessarily the one with the minimum number of terms.
- This can be demonstrated from inspection of the map for the function of Example 3-14.
- As shown in Fig 3-28, the minimized function is recognized to be
    - $F = x'y'z + w'xz' + xyz + wx'$
- Which consists of the sum of four of the six prime implicants derived in Example 3-14.

•The tabular procedure for selecting the prime implicants that give the minimized function is the subject of the next section.

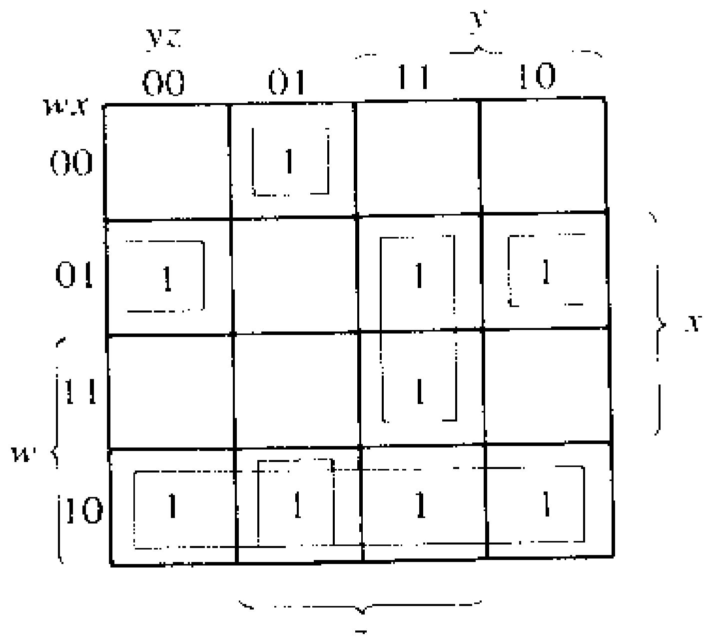


FIGURE 3-28

Map for the function of Example 3-14;
$F = x'y'z' + w'xz' + xyz + wx'$

3-11 SELECTION OF PRIME IMPLICANTS

- The selection of prime implicants that form the minimized function is made from a prime implicant table.

- In this table, each prime implicant is represented in a row and each minterm in a column.

- X's are placed in each row to show the composition of minterms that make the prime implicants.

- A minimum set of prime implicants is then chosen that covers all the minterms in the function.

- This procedure is illustrated in Example 3-15.

Example 3-15

- Minimize the function in Example 3-14.

- The prime-implicant table for this example is shown in Table 3-8.

TABLE 3-8
Prime Implicant Table for Example 3-15

		1	4	6	7	8	9	10	11	15
√ $x'y'z$	1, 9	X					X			
√ $w'xz'$	4, 6		X	X						
$w'xy$	6, 7			X	X					
xyz	7, 15				X					X
wyz	11, 15								X	X
√ wx'	8, 9, 10, 11					X	X	X	X	
		√	√	√		√	√	√	√	

- There are six rows, one for each prime implicant (derived in Example 3-14).

- There are nine columns, each representing one minterm of the function.

- X's are placed in each row to indicate the minterms contained in the prime implicant of that row.

- For example, the two X's in the first row indicate that minterms 1 and 9 are contained in the prime implicant x'y'z.

- It is advisable to include the decimal equivalent of the prime implicants in each row, as it gives the minterms contained in it.

- After all the X's have been marked, we proceed to select a minimum number of prime implicants.

•The completed prime-implicant table is inspected for columns containing only a single X.

•In this example, there are four minterms whose columns have a single X: 1, 4, 8, and 10.

•Minterm 1 is covered by prime implicant x’y’z.

•Therefore, the selection of the prime implicant x’y’z guarantees that minterm 1 is included in the function.

•Similarly, minterm 4 is covered by prime implicant w’xz’, and minterms 9 and 10 , by wx’.

•Prime implicants that cover minterms with a single X in their column are called **essential prime implicants.**

- To make the final simplified version contain all the minterms, we have to include essential prime implicants.

- A check mark is placed in the table next to the essential prime implicants to indicate that they have been selected.

- Next we check each column whose minterm is covered by the selected essential prime implicants.

- For example, the selected prime implicant x’y’z covers minterms 1 and 9.

- A check is inserted in the bottom of the columns.

- Similarly, prime implicant w’xz’ covers minterms 4 and 6, and wx’ covers minterms 8, 9, 10, and 11.

- Inspection of the prime-implicant table shown that the selection of the essential prime implicants covers all the minterms of the function except 7 and 15.

- These two minterms must be included by the selection of one or more prime implicants.

- In this example, it is clear that prime implicant xyz covers both minterms and is therefore the one to be selected.

- We have thus found the minimum set of prime implicants whose sum gives the required minimized function:

  - $F = x'y'z + w'xz' + wx' + xyz$

- The simplified expression derived in the preceding example were all the sum of products form.

- The tabulation method can be adapted to give a simplified expression in product of sums.

•As in the map method, we have to start with the complement of the function by taking the 0’s as the initial list of minterms.

•This list contains those minterms not included in the original function that are numerically equal to the maxterms of the function.

•The tabulation process is carried out with the 0’s of the function and terminates with a simplified expression in sum of products of the complement of the function.

•By taking the complement again, we obtain the simplified product of sum expression.

•A function with don’t-care conditions can be simplified by the tabulation method after a slight modification.

•The don’t-care terms are included in the list of minterms, when the prime implicants are determined.

•This allows the derivation of prime implicants with the least number of literals.

•The don’t-care terms are not included in the list of minterms when the prime implicant table is set up.

•The reason is that don’t-care terms do not have to be covered by the selected prime implicants.

- **REVIEW EXERCISES**

1. Simplify the Boolean function F together with the don't care conditions d in (i) sum of products and (ii) products of sums

- $F(w, x, y, z) = \sum(0, 1,2,3,7,8, 10)$
- $d(w, x, y, z) = \sum(5,6,11,15)$

2. Simplify the following Boolean functions by means of the tabulation method:

(a) $P(A, B, C, D, E, F, G) = \sum(20, 28, 52, 60)$

(b) $P(A, B, C, D, E, F, G) = \sum(20, 28, 38, 39, 52, 60, 102, 103, 127)$

3. Simplify the following Boolean function and draw the logic diagram of the simplified version.

$F = xy'z + x'y'z + w'xy + wx'y + wxy$

4. Represent the following decimal number 8620 in (a) BCD, (b) excess-3 code, (c) 2421 code

5. The binary number listed have a sign in the leftmost position and if negative, are in 2's complement form. Perform the arithmetic operations indicated and verify the answers.

a)111001 + 001010

b)101011 – 100110

6. Convert the following expression in to sum of products and product of sums
$x' + x(x + y')(y + z')$

7.Express the following function in sum of minterms or product of maxterms
$F(x, y, z) = (xy + z)(xz + y)$

8. Convert the following to the other canonical form
$F(x, y, z) = \sum(1,3,7)$
$F(A, B, C, D) = \prod(0, 1, 2, 3, 4, 6, 12)$