Chapter 3
Simplification of Boolean Functions

Simplification of Boolean Functions

3.1 The Map Method

- The map method provides a simple straightforward procedure for minimizing Boolean functions.
- The map method, first proposed by Veitch (1) and slightly modified by Karnaugh (2), is also known as the “Veitch diagram” or the “Karnaugh map.”
- The map is a diagram made up of squares.
- Each square represents one minterm

3.2 Two- and Three-variable Maps

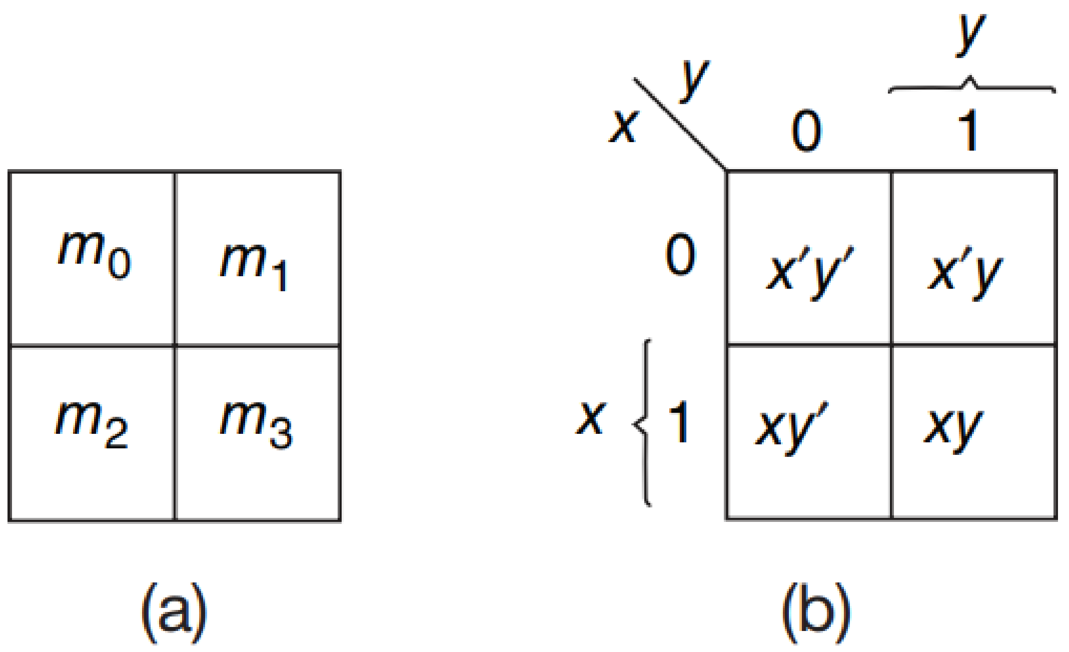


Figure 3.1 Two-variable map

3.2 Two- and Three-variable Maps

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

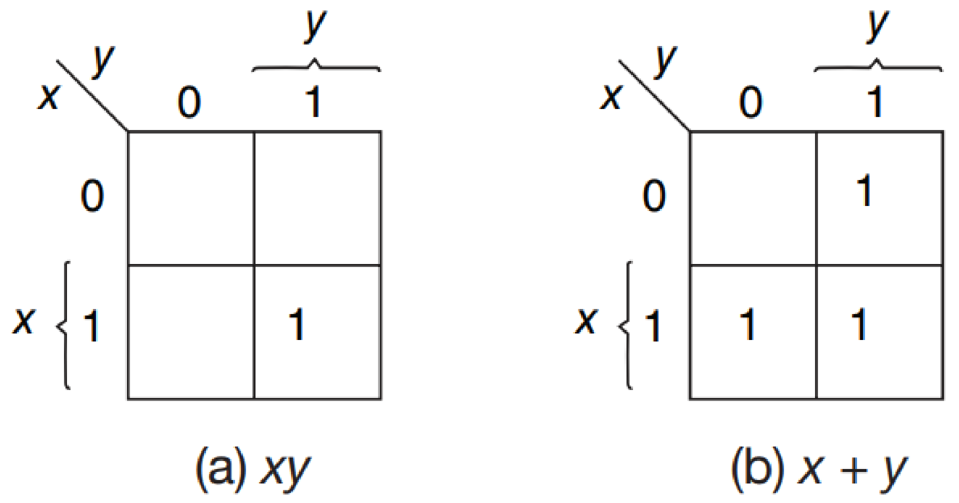
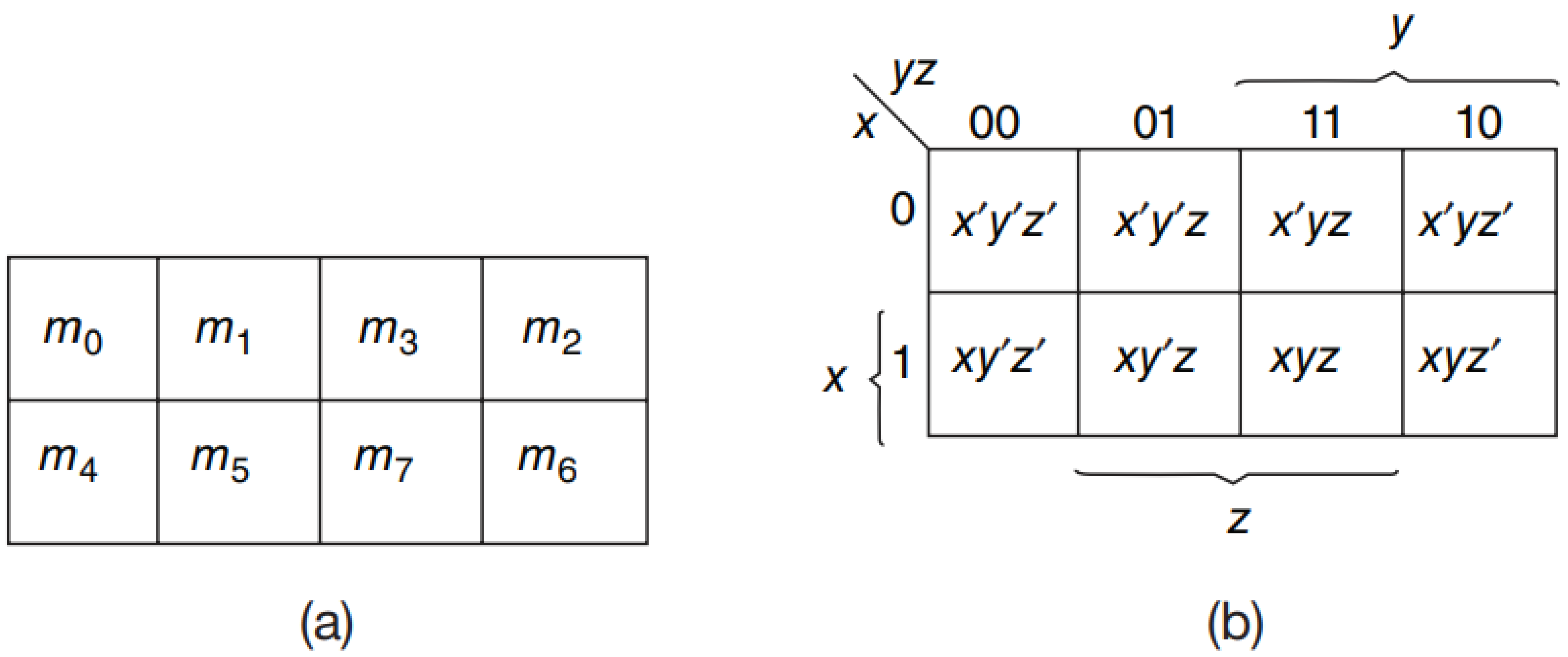


Figure 3-2 Representation of functions in the map

3.2 Two and Three-variable Maps



m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

$x \backslash yz$	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

(b)

Figure 3-3 Three-variable map

3.2 Two and Three-variable Maps

EXAMPLE 3-1: Simplify the Boolean function:

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

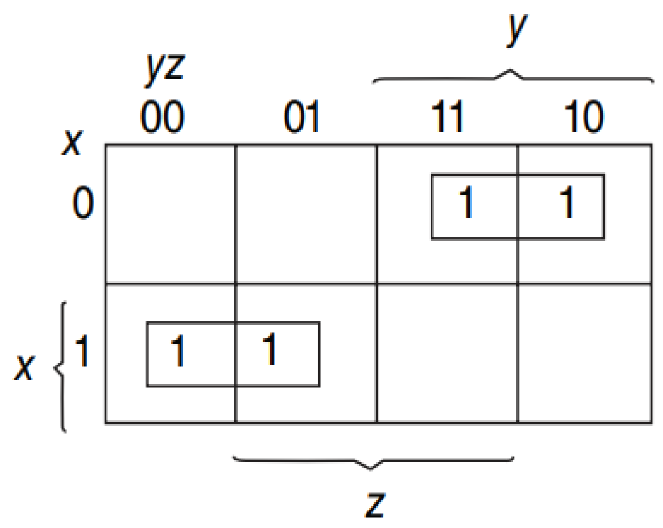
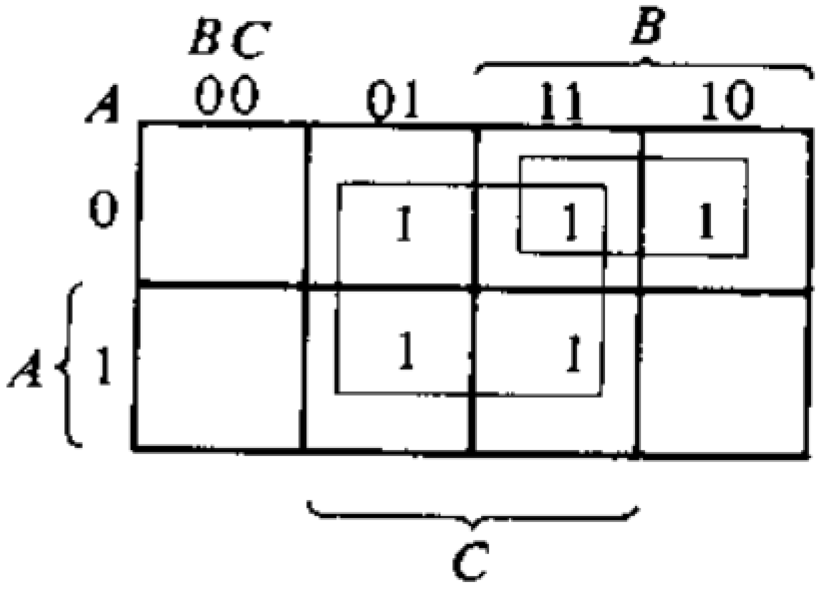


Figure 3.4 Map for Example 3-1; $x'yz + x'yz' + xy'z' + xy'z = x'y + xy'$

3.2 Two and Three-variable Maps

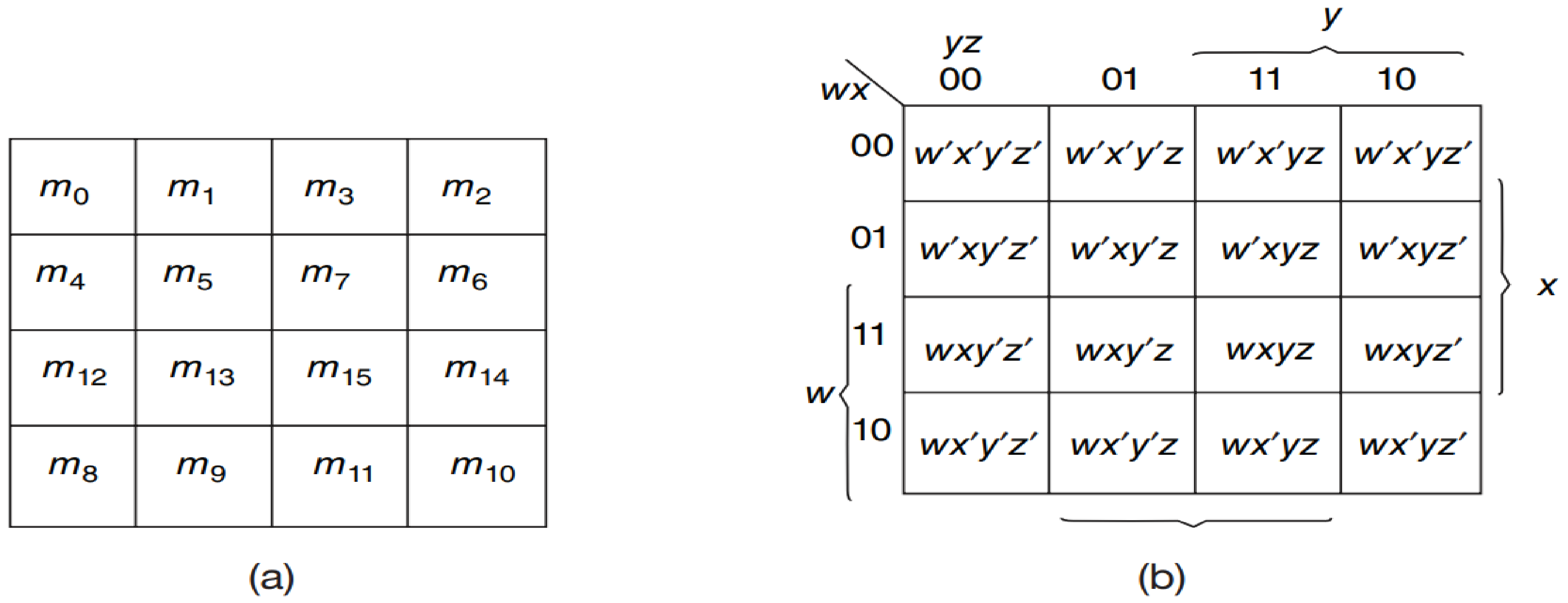
EXAMPLE 3-3: Simplify the Boolean function:

F = A′C + A′B + AB′C + BC



3.3 Four-variable Map

The map for Boolean functions of four binary variables are listed the 16 minterms and the squares assigned to each.



(a)

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(b)

$wx \backslash yz$	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

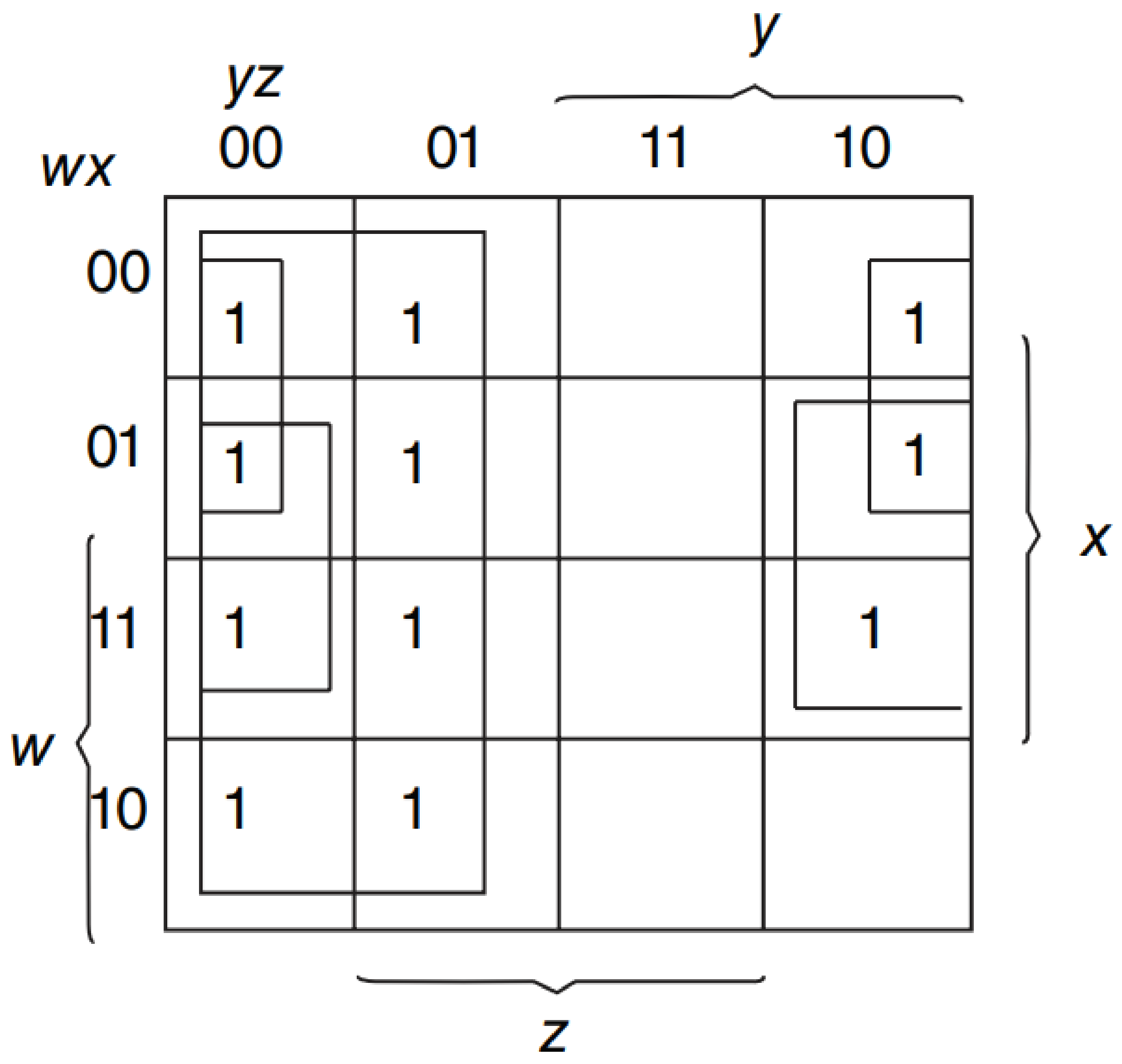
Figure 3.8 Four-variable map

3.3 Four-variable Map

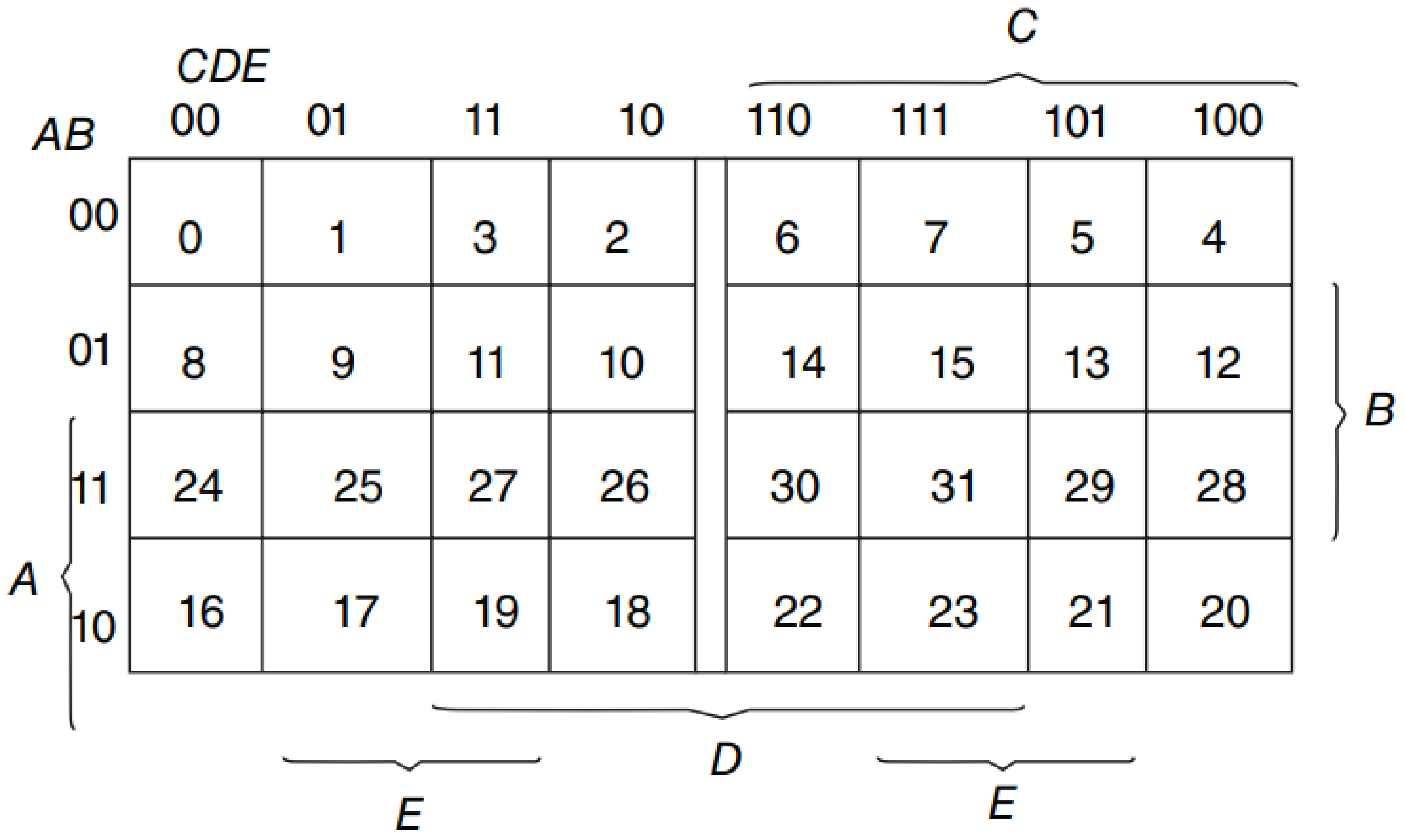
EXAMPLE 3-5: Simplify the Boolean function;

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

y′ + w′z′ + xz′

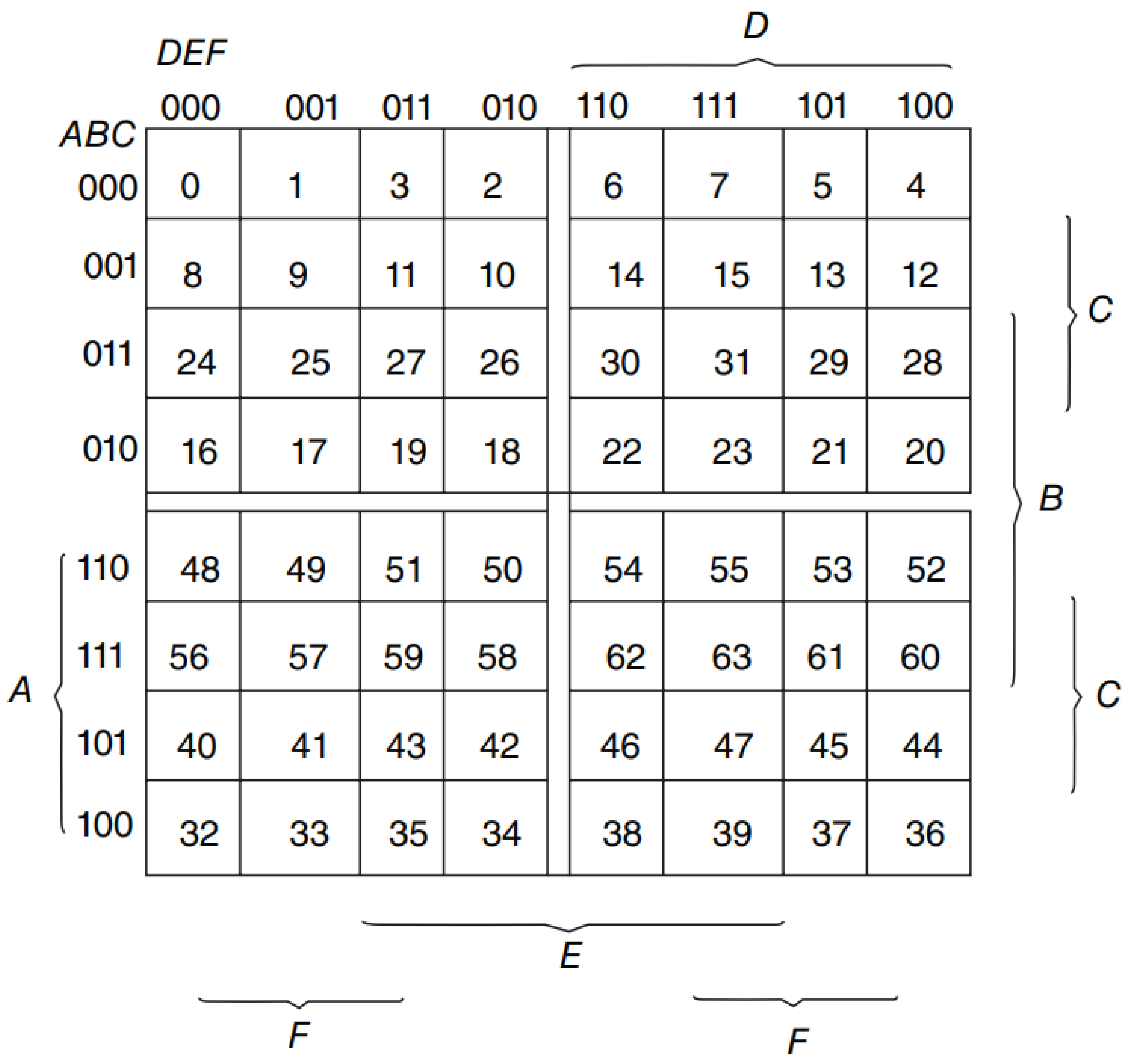


3.4 Five- and Six-Variable Maps



AB \ CDE	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20

3.4 Five- and Six-Variable Maps



ABC \ DEF	000	001	011	010	110	111	101	100
000	0	1	3	2	6	7	5	4
001	8	9	11	10	14	15	13	12
011	24	25	27	26	30	31	29	28
010	16	17	19	18	22	23	21	20
110	48	49	51	50	54	55	53	52
111	56	57	59	58	62	63	61	60
101	40	41	43	42	46	47	45	44
100	32	33	35	34	38	39	37	36

3.5 Product of Sums Simplification

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, i.e., of F’.

The complement of F’ gives us back the function F.

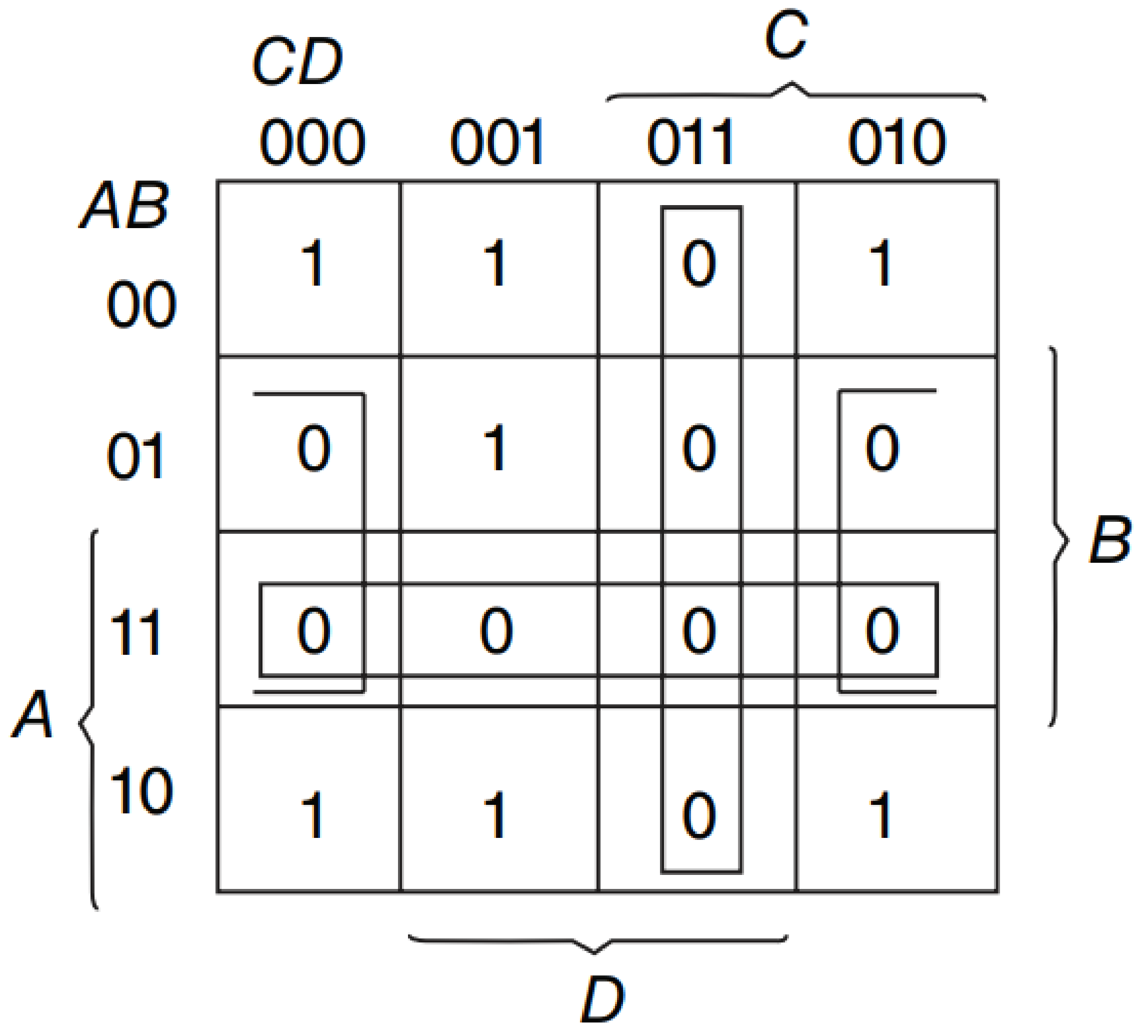
Because of the generalized DeMorgan’s theorem, the function so obtained is automatically in the product of sums form.

3.5 Product of Sums Simplification

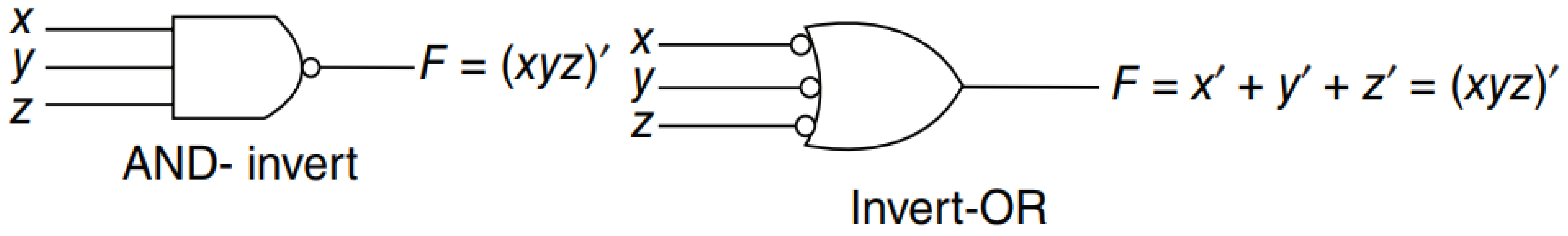
EXAMPLE 3-8: Simplify the following Boolean function in (a) sum of products and (b) product of sums. F (A, B, C, D) = ∑(0, 1, 2, 5, 8, 9, 10)

(a) F = B′D′ + B′C′ + A′C′D

(b) F = (A′ + B′)(C′ + D′)(B′ + D)

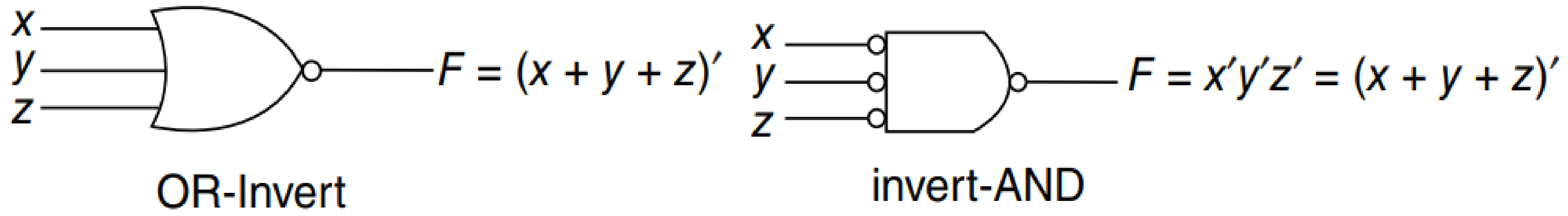


3.6 Nand and Nor Implementation



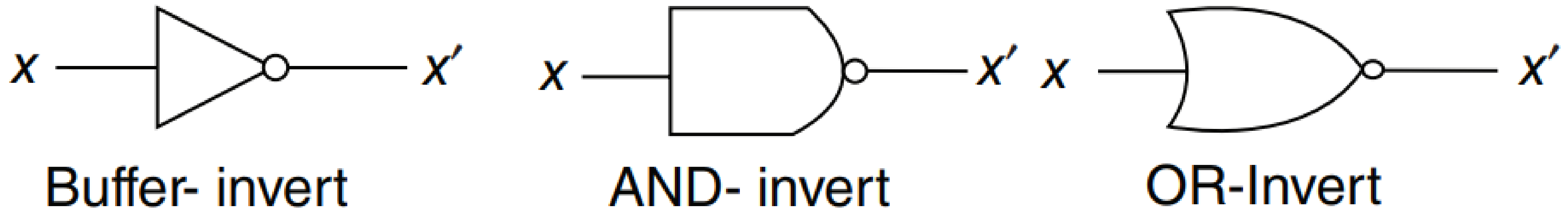
(a) Two grapic symbol for NAND gate

3.6 Nand and Nor Implementation



(b) Two grapic symbols for NOR gate.

3.6 Nand and Nor Implementation



(c) Three grapic symbols for inverter.

3.6.1 NAND 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, with inputs coming from 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.

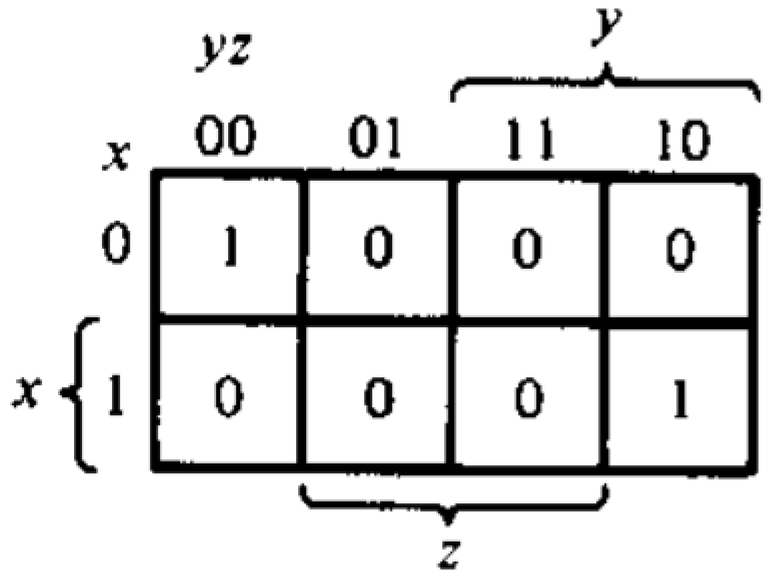
3.6.1 NAND Implementation

EXAMPLE 3-9: Implement the following function with NAND gates:

$F(x, y, z) = \sum (0, 6)$

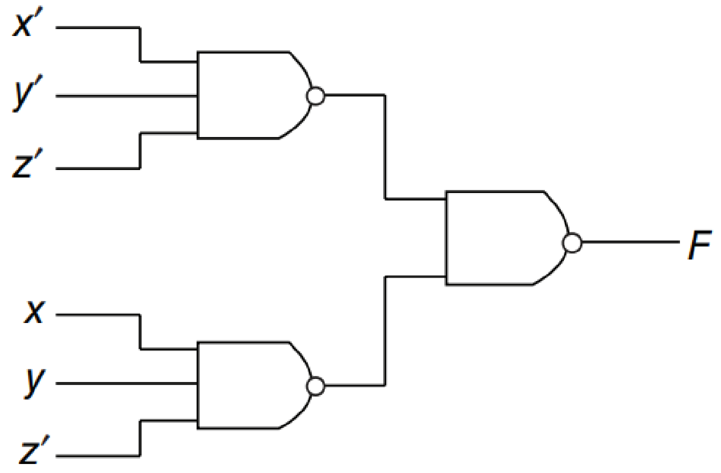
$F = x' y' z' + x y z'$

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

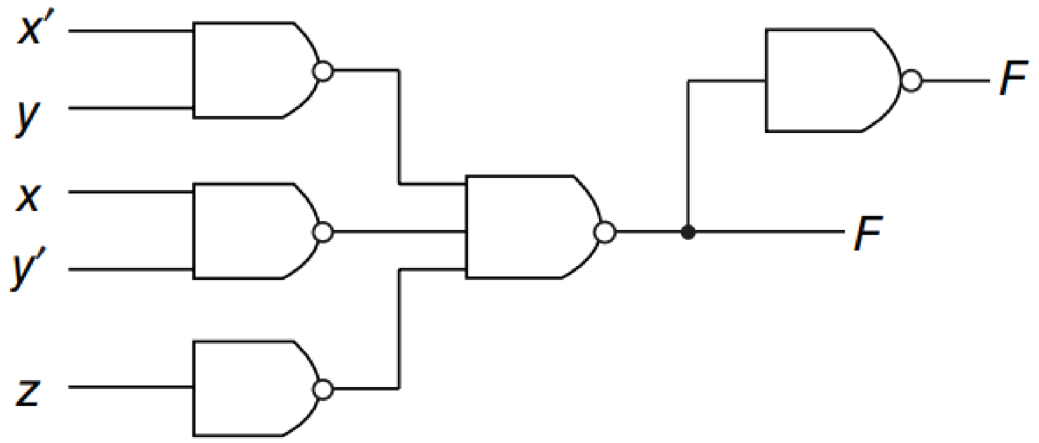


x \ yz	00	01	11	10
0	1	0	0	0
1	0	0	0	1

3.6.1 NAND Implementation



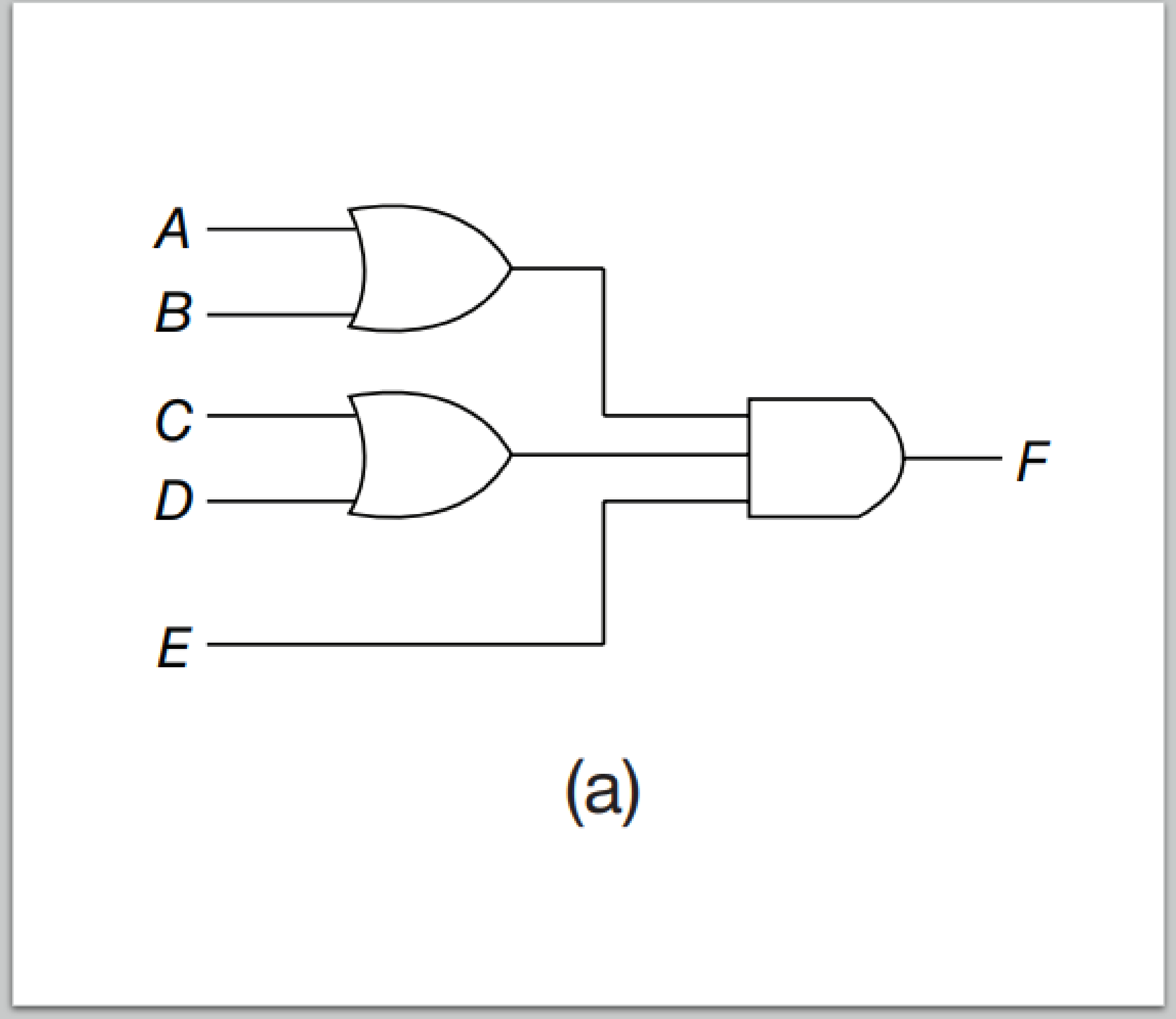
(b) $F = x'y'z' + xyz$



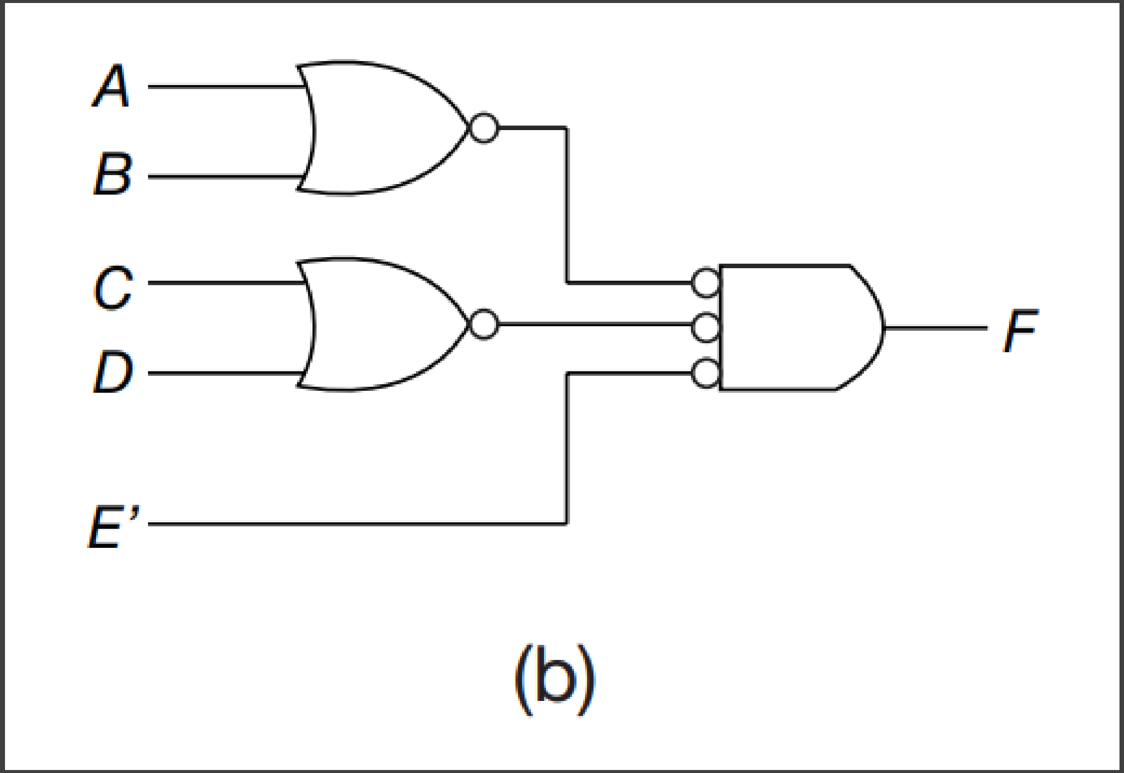
(c) $F' = x'y + xy' + z'$

3.6.2 NOR Implementation

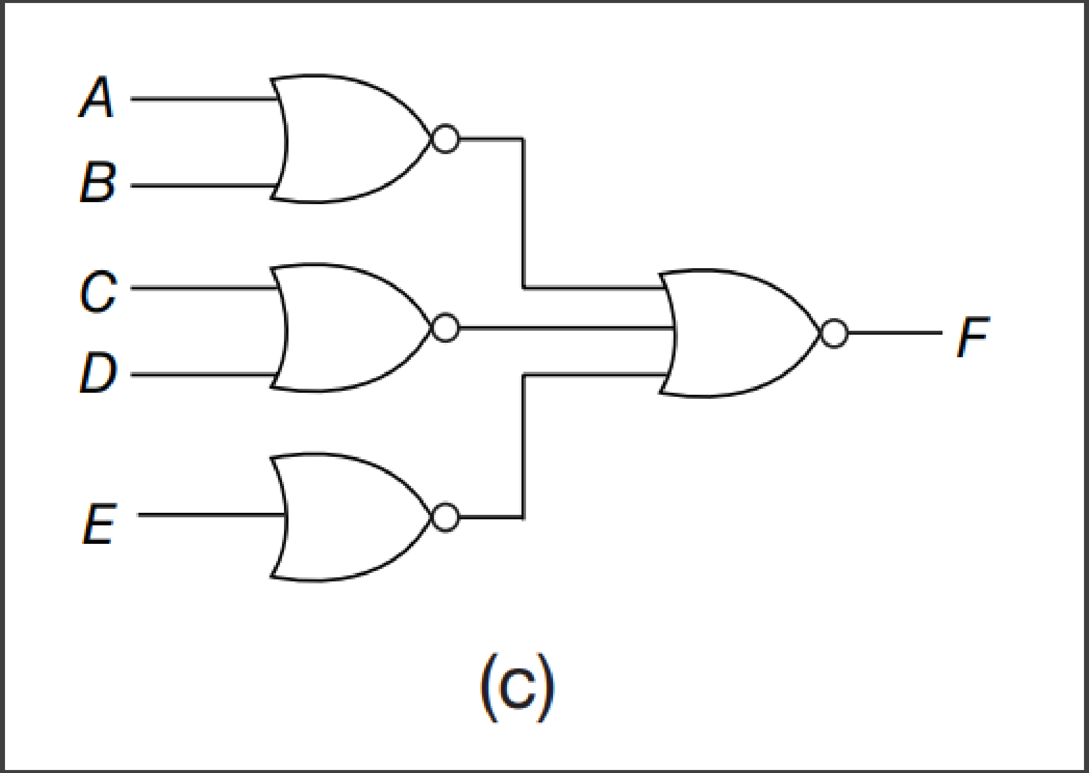
Three ways to implement F = (A + B) (C + D)E



3.6.2 NOR Implementation



(b)



(c)

3.6.2 NOR Implementation

EXAMPLE 3-10: Implement the function of Example 3-9 with NOR gates.

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

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

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

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

3.6 Nand and Nor Implementation

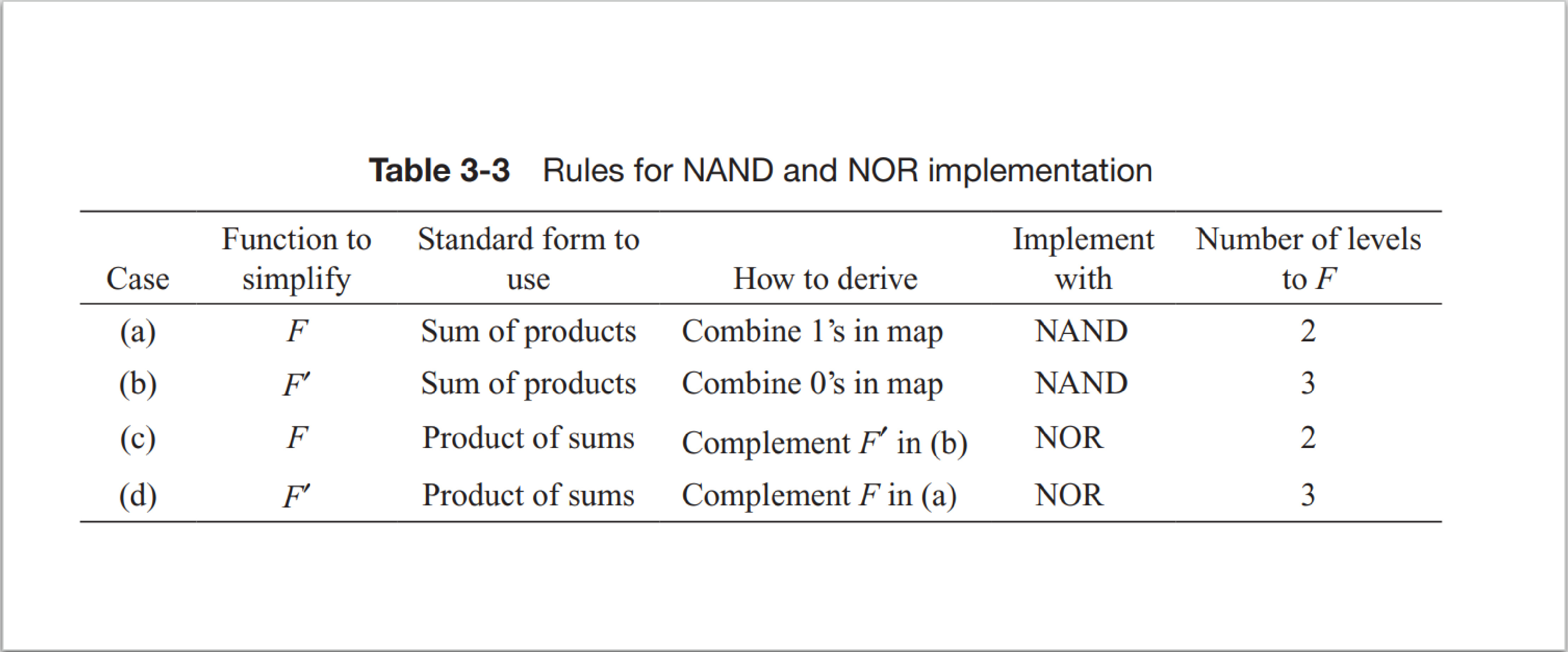
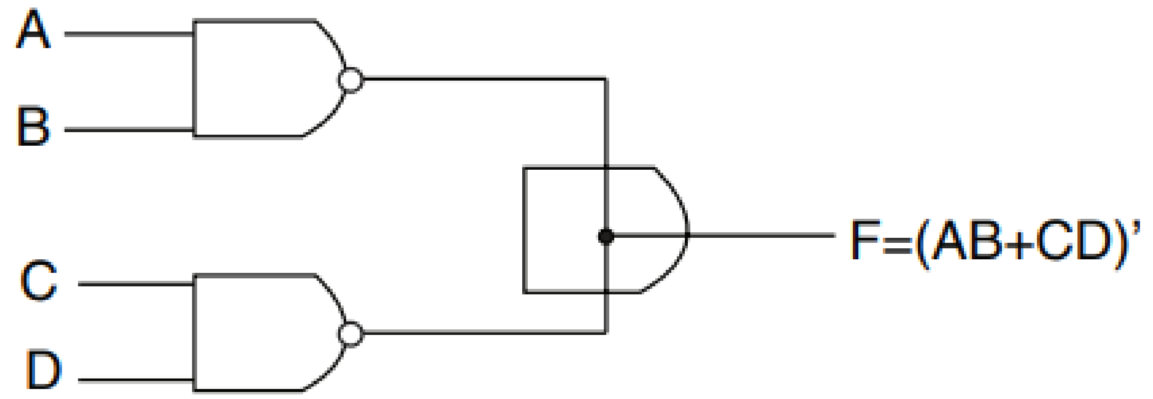


Table 3-3 Rules for NAND and NOR implementation

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

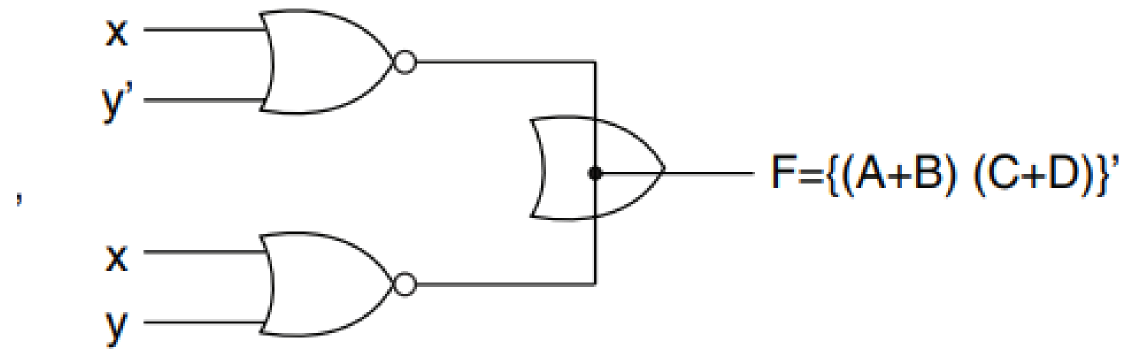
F = (AB)′ • (CD)′ = (AB + CD)′



(a) Wired - And in open – collector TTL NAND gates (AND - OR - INVERT)

3.7 Other Two-level Implementations

F = (A + B)′ + (C + D)′ = [(A + B) (C + D)]′



(b) Wired-OR in ECL gates
(OR – AND - INVERT)

3.8 Don't-care Conditions

EXAMPLE 3-12: Simplify the Boolean function:

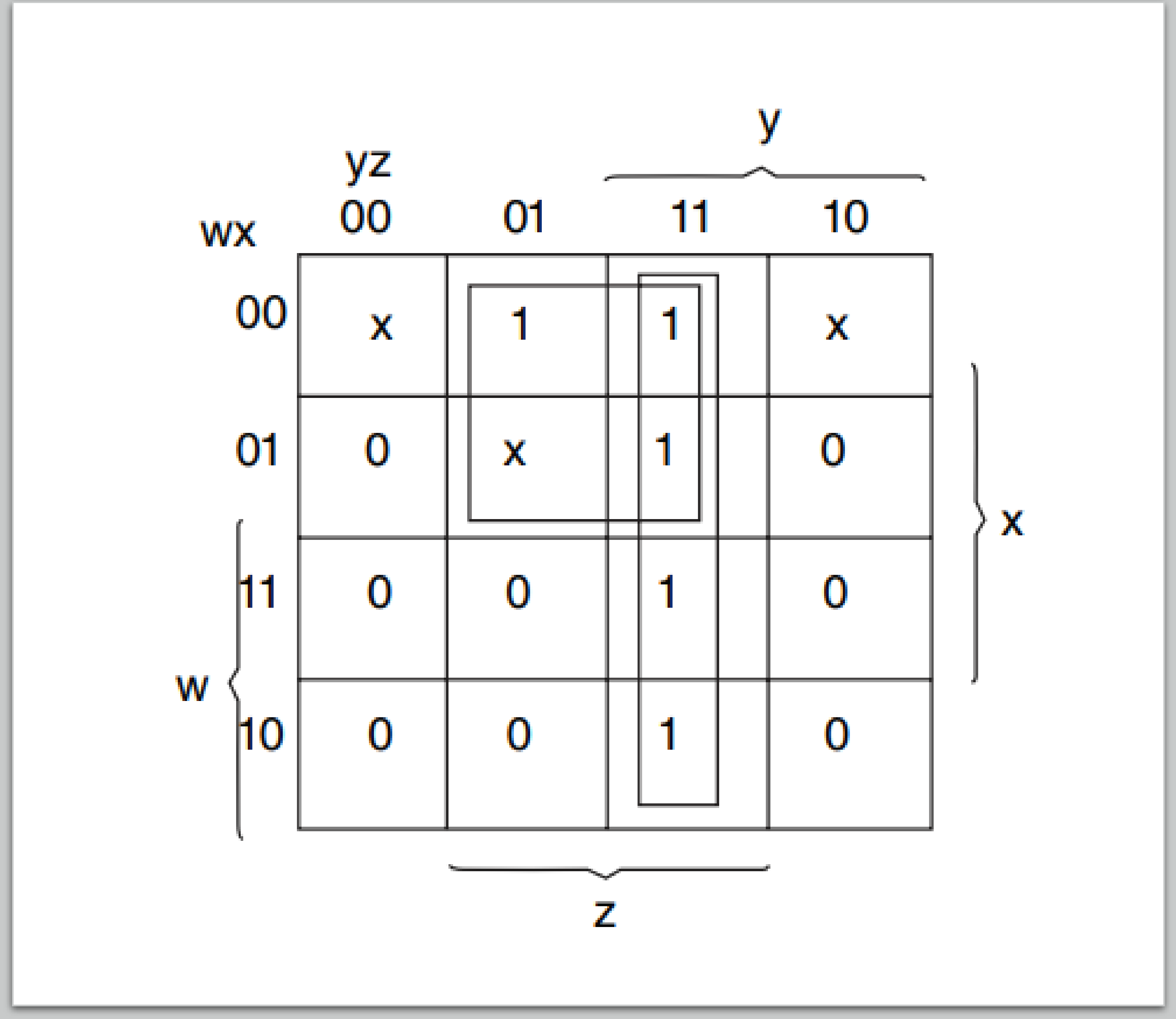
F(w, x, y, z) = ∑(1,3, 7, 11, 15)

and the don't-care conditions:

d(w, x, y, z) = ∑(0, 2, 5)

F = w′ x′ + y z (DYS)

F = w′ z + y z



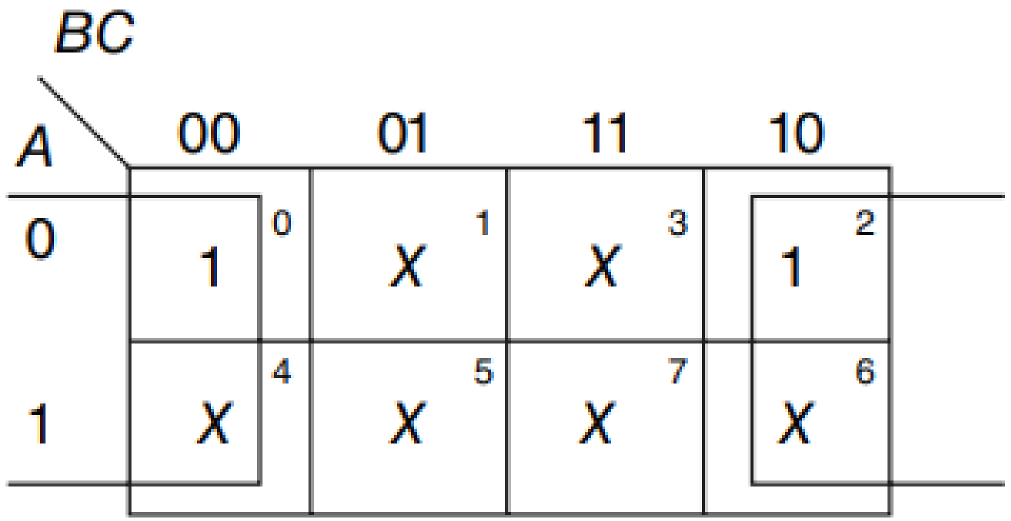
wx \ yz	00	01	11	10
00	x	1	1	x
01	0	x	1	0
11	0	0	1	0
10	0	0	1	0

3.8 Don't-care Conditions

Simplify the following equation by K-Map

$F(A, B, C) = \sum m(0, 2, 4) + \sum d(1, 3, 5, 6, 7)$

$F(A, B, C) = C'$

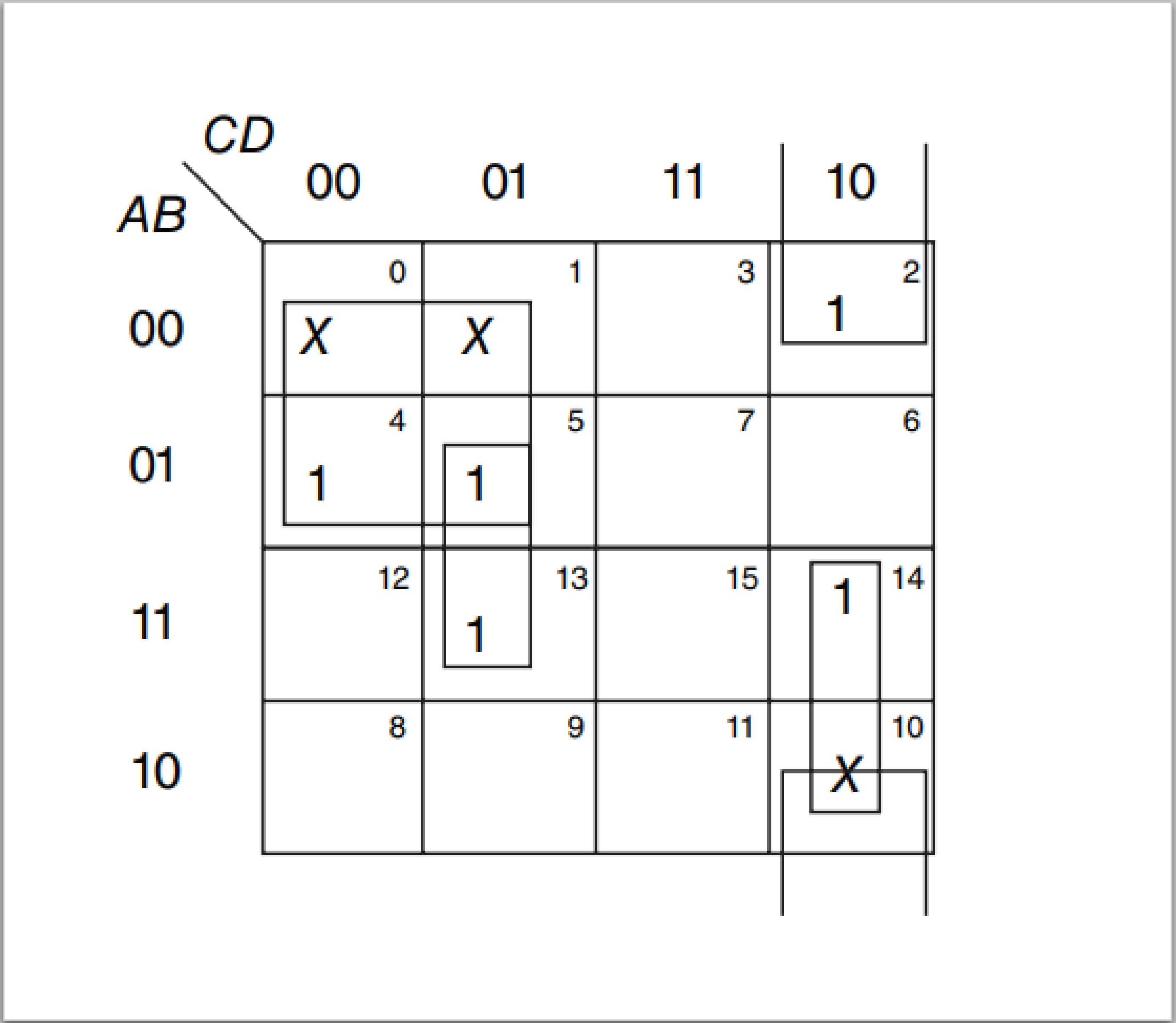


3.8 Don’t-care Conditions

Simplify the following equation by K-Map:

F(A, B, C, D) = ∑m(2, 4, 5, 13, 14) + ∑d(0, 1, 8, 10)

F = A’C’ + BC’D + B’CD’ + ACD’



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 a trial-and-error procedure which relies on the ability of the human user to recognize certain patterns.
- For functions of six or more variables, it is difficult to be sure that the best selection has been made.

3.9 The Tabulation Method

- 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.
- The tabular method of simplification consists of two parts:
- The first is to find by an exhaustive search all the terms that are candidates for inclusion in the simplified function. These terms are called prime-implicants.
- The second operation is to choose among the prime-implicants those that give an expression with the least number of literals.

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.

3.10 Determination of Prime-implicants

- EXAMPLE 3-13: Simplify the following Boolean function by using the tabulation method: F = ∑(0, 1, 2, 8, 10, 11, 14, 15).

- F = w′ x′ y′ + x′ z′ + w y

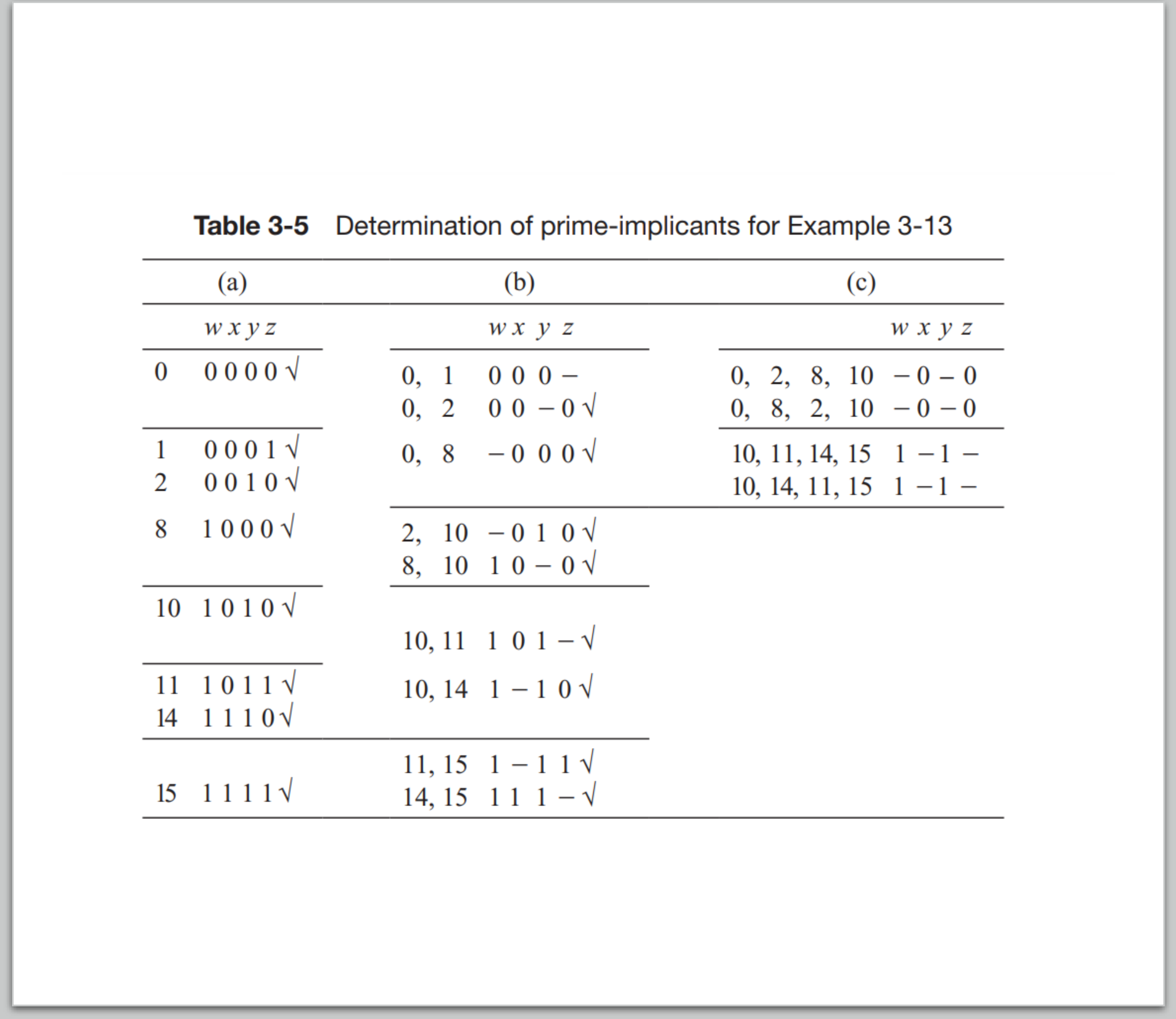


Table 3-5 Determination of prime-implicants for Example 3-13

(a)		(b)		(c)	
	w x y z		*w x y z*		*w x y z*
0	0 0 0 0 √	0, 1	0 0 0 –	0, 2, 8, 10	– 0 – 0
		0, 2	0 0 – 0 √	0, 8, 2, 10	– 0 – 0
1	0 0 0 1 √	0, 8	– 0 0 0 √	10, 11, 14, 15	1 – 1 –
2	0 0 1 0 √			10, 14, 11, 15	1 – 1 –
8	1 0 0 0 √	2, 10	– 0 1 0 √		
		8, 10	1 0 – 0 √		
10	1 0 1 0 √	10, 11	1 0 1 – √		
11	1 0 1 1 √	10, 14	1 – 1 0 √		
14	1 1 1 0 √				
		11, 15	1 – 1 1 √		
15	1 1 1 1 √	14, 15	1 1 1 – √		

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.
- Crosses 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.

3.11 Selection Of Prime-implicants

- Minimize the function of : F (w, x, y, z) = Σ(1, 4, 6, 7, 8, 9, 10, 11, 15)

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			
√ $w' x z'$	4, 6		X	X						
w' x y	6, 7			X	X					
$x y z$	7, 15				X					X
$w y z$	11, 15								X	X
√ $w x'$	8, 9, 10, 11					X	X	X	X	
		√	√	√		√	√	√	√	

3.12 Concluding Remarks

- Two methods of Boolean function simplification were introduced in this chapter.
- The criterion for simplification was taken to be the minimization of the number of literals in sum of products or product of sums expressions.
- Both the map and the tabulation methods are restricted in their capabilities since they are useful for simplifying only Boolean functions expressed in the standard forms.
- Although this is a disadvantage of the methods, it is not very critical. Most applications prefer the standard forms over any other form.