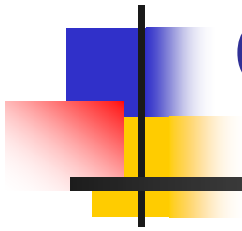


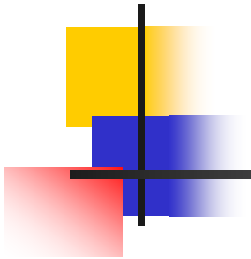
Chapter 2

Boolean Algebra and Logic Gates

Reference : Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog, 6th Edition
M. Morris R. Mano, Michael D. Ciletti

Basic Definitions

- A *binary operator* defined on a set *S* of elements is a rule that assigns, to each pair of elements from *S*, a unique element from *S*.
- The most common postulates used to formulate various algebraic structures are as follows:
  1. ***Closure.*** A set *S* is closed with respect to a binary operator if, for every pair of elements of *S*, the binary operator specifies a rule for obtaining a unique element of *S*.
  2. ***Associative law.*** A binary operator * on a set *S* is said to be associative whenever
     $(x * y) * z = x * (y * z)$ for all $x, y, z, \in S$



3. ***Commutative law.*** A binary operator * on a set S is said to be commutative whenever

   $x * y = y * x$ for all $x, y \in S$

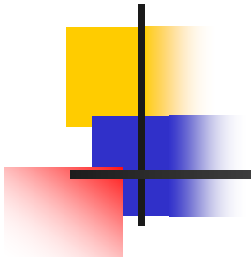
4. ***Identity element.*** A set S is said to have an identity element with respect to a binary operation * on S if there exists an element $e \in S$ with the property that

   $e * x = x * e = x$ for every $x \in S$

   Example: The element 0 is an identity element with respect to the binary operator + on the set of integers $I = \{c, -3, -2, -1, 0, 1, 2, 3, c\}$, since

   $x + 0 = 0 + x = x$ for any $x \in I$

   The set of natural numbers, N, has no identity element, since 0 is excluded from the set.



5. ***Inverse.*** A set S having the identity element e with respect to a binary operator * is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that

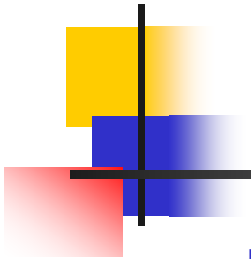
   $x * y = e$

   Example: In the set of integers, I, and the operator +, with $e = 0$, the inverse of an element a is $(-a)$, since $a + (-a) = 0$.

6. ***Distributive law.*** If * and • are two binary operators on a set S, * is said to be distributive over • whenever

   $x * (y \bullet z) = (x * y) \bullet (x * z)$

- A *field* is an example of an algebraic structure.

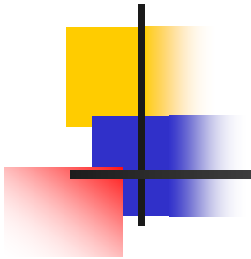


- The field of real numbers is the basis for arithmetic and ordinary algebra.
  - The binary operator + defines addition.
  - The additive identity is 0.
  - The additive inverse defines subtraction.
  - The binary operator • defines multiplication.
  - The multiplicative identity is 1.
  - For $a \neq 0$, the multiplicative inverse of $a = 1/a$ defines division (i.e., $a \cdot 1/a = 1$).
  - The only distributive law applicable is that of • over +:

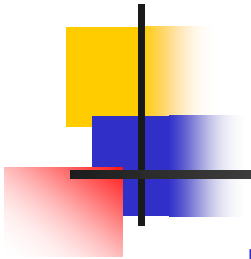
    $$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

Axiomatic Definition of Boolean Algebra

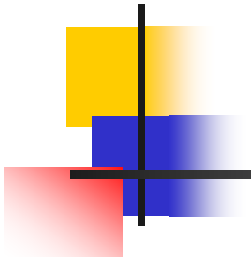
- In 1854, George Boole developed an algebraic system now called *Boolean algebra.*
- two binary operators, + and •, (Huntington) postulates:
  1. (a) The structure is closed with respect to the operator +.
     (b) The structure is closed with respect to the operator •.
  2. (a) The element 0 is an identity element with respect to +; that is, $x + 0 = 0 + x = x$.
     (b) The element 1 is an identity element with respect to •; that is, $x \cdot 1 = 1 \cdot x = x$.



3. (a) The structure is commutative with respect to +; that is, $x + y = y + x$.
   (b) The structure is commutative with respect to • ; that is, $x \cdot y = y \cdot x$.
4. (a) The operator • is distributive over +; that is, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   (b) The operator + is distributive over •; that is, $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. For every element $x \in B$, there exists an element $x \in B$ (called the complement of x) such that (a) $x + x = 1$ and (*b*) $x \cdot x = 0$.
6. There exist at least two elements $x, y \in B$ *such that* $x \neq y$.



- Comparing Boolean algebra with arithmetic and ordinary algebra
  1. Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived (for both operators) from the other postulates.
  2. The distributive law of + over • (i.e., $x + (y \cdot z) = (x + y) \cdot (x + z)$) is valid for Boolean algebra, but not for ordinary algebra.
  3. Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.



4. Postulate 5 defines an operator called the complement that is not available in ordinary algebra.
5. Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra deals with the as yet undefined set of elements, *B*, but in the two–valued Boolean algebra defined next (and of interest in our subsequent use of that algebra), *B* is defined as a set with only two elements, 0 and 1.

Two-valued Boolean Algebra

- B = {0,1}
- The rules of operations

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

- Closure
- The identity elements

  (1) +: 0

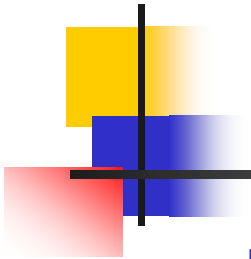
  (2) • : 1

- The commutative laws
- The distributive laws

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

$y + z$	$x \cdot (y + z)$
0	0
1	0
1	0
1	0
0	0
1	1
1	1
1	1

$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	1	1
1	0	1
1	1	1



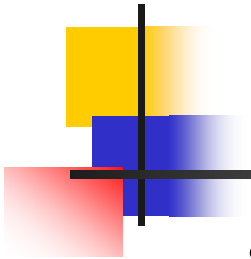
- Complement
  - $x+x'=1$: 0+0'=0+1=1; 1+1'=1+0=1
  - $x \cdot x'=0$: 0 • 0'=0 • 1=0; 1 • 1'=1 • 0=0
- Has two distinct elements 1 and 0, with $0 \neq 1$
- Note
  - a set of two elements
  - + : OR operation; • : AND operation
  - a complement operator: NOT operation
  - Binary logic is a two-valued Boolean algebra

Basic Theorems and Properties of Boolean Algebra

- Duality
  - the binary operators are interchanged; AND ⇔ OR
  - the identity elements are interchanged; 1 ⇔ 0

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$



- Theorem 1(a): $x+x = x$
  - $x+x = (x+x)\ 1$ by postulate: 2(b)
    $= (x+x)\ (x+x')$ 5(a)
    $= x+xx'$ 4(b)
    $= x+0$ 5(b)
    $= x$ 2(a)
  - Theorem 1(b): x • x = x
    - $x \cdot x = x\ x + 0$
      $= xx + xx'$
      $= x\ (x + x')$
      $= x \cdot 1$
      $= x$

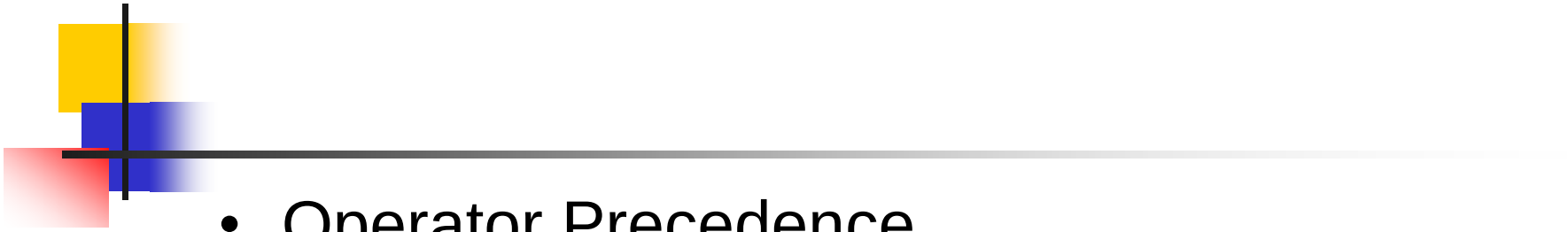
- Theorem 2
  - $x + 1 = 1 \cdot (x + 1)$
    $= (x + x')(x + 1)$
    $= x + x' 1$
    $= x + x'$
    $= 1$
  - $x \cdot 0 = 0$ by duality
- Theorem 3: $(x')' = x$
  - Postulate 5 defines the complement of x, $x + x' = 1$ and $x \cdot x' = 0$
  - The complement of x' is x is also $(x')'$

- Theorem 6
  - $x + xy = x \cdot 1 + xy$
    $= x\,(1 + y)$
    $= x \cdot 1$
    $= x$
  - $x\,(x + y) = x$ by duality
- By means of truth table

x	y	xy	$x + xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

- DeMorgan's Theorems
  - $(x+y)' = x'\,y'$
  - $(x\,y)' = x' + y'$

x	y	$x+y$	$(x+y)'$	x'	y'	$x'y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0



- Operator Precedence
  - parentheses
  - NOT
  - AND
  - OR
  - examples
    - $x\,y' + z$
    - $(x\,y + z)'$

Boolean Functions

- A Boolean function
  - binary variables
  - binary operators OR and AND
  - unary operator NOT
  - parentheses
- Examples
  - $F_1 = x + y\,z'$
  - $F_2 = x'\,y'\,z + x'\,y\,z + x\,y'$

- The truth table for F_1 and F_2

Table 2.2
Truth Tables for F_1 and F_2

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

- Implementation with logic gates

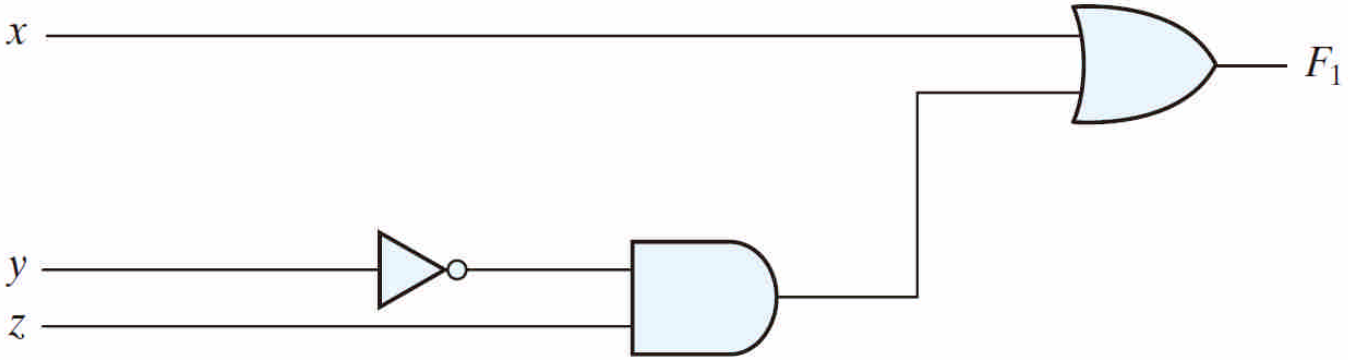
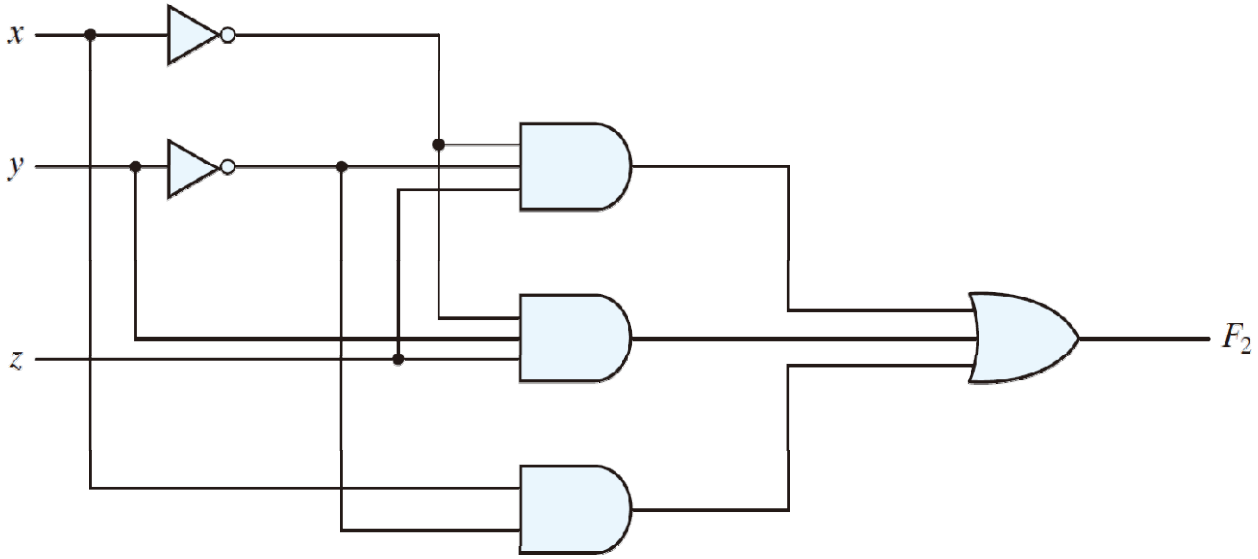
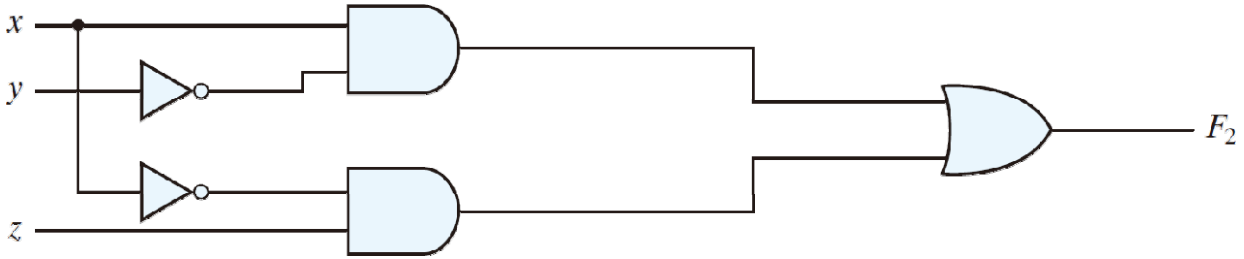


FIGURE 2.1
Gate implementation of $F_1 = x + y'z$

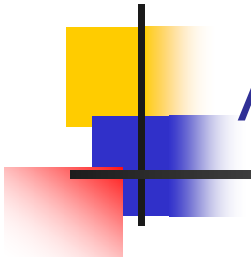


(a) $F_2 = x'y'z + x'yz + xy'$



(b) $F_2 = xy' + x'z$

FIGURE 2.2
Implementation of Boolean function F_2 with gates



Algebraic Manipulation

- To minimize Boolean expressions
  - literal: a primed or unprimed variable (an input to a gate)
  - term: an implementation with a gate
  - The minimization of the number of literals and the number of terms => a circuit with less equipment

EXAMPLE 2.1

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$

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

3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$

4.
$$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$

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

Complement of a Function

- an interchange of 0's for 1's and 1's for 0's in the value of *F*
- by DeMorgan's theorem
- (A+B+C)' = (A+X)' let B+C = X
  = A'X' by DeMorgan's
  = A'(B+C)'
  = A'(B'C') by DeMorgan's
  = A'B'C' associative

- generalizations
  - (A+B+C+ ... +F)' = A'B'C' ... F'
  - (ABC ... F)' = A'+ B'+C'+ ... +F'

EXAMPLE 2.2

Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$. By applying DeMorgan's theorems as many times as necessary, the complements are obtained as follows:

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$

$$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)'$$

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

$$= x' + yz' + y'z$$

EXAMPLE 2.3

Find the complement of the functions F_1 and F_2 of Example 2.2 by taking their duals and complementing each literal.

1. $F_1 = x'yz' + x'y'z$.

The dual of F_1 is $(x' + y + z')(x' + y' + z)$.

Complement each literal: $(x + y' + z)(x + y + z') = F_1'$.

2. $F_2 = x(y'z' + yz)$.

The dual of F_2 is $x + (y' + z')(y + z)$.

Complement each literal: $x' + (y + z)(y' + z') = F_2'$.

Canonical and Standard Forms

- Minterms and Maxterms
  - A minterm: an AND term consists of all literals in their normal form or in their complement form
  - For example, two binary variables *x* and *y*,
    - *xy*, *xy'*, *x'y*, *x'y'*
  - It is also called a standard product
  - *n* variables con be combined to form 2^n minterms
  - A maxterm: an OR term
  - It is also call a standard sum
  - 2^n maxterms

- each maxterm is the complement of its corresponding minterm, and vice versa

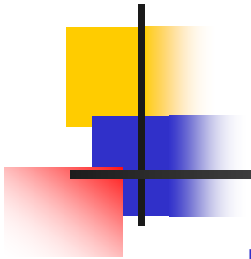
Table 2.3
Minterms and Maxterms for Three Binary Variables

			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

- An Boolean function can be expressed by
  - a truth table
  - sum of minterms
  - $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$
  - $f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$

Table 2.4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

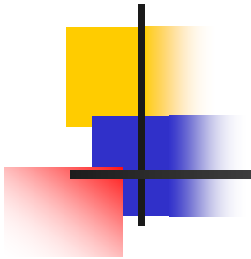


- The complement of a Boolean function
  - the minterms that produce a 0
  - $f_1' = m_0 + m_2 + m_3 + m_5 + m_6$
    $= x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - $f_1 = (f_1')'$
    $= (x+y+z)(x+y'+z)\ (x+y'+z')\ (x'+y+z')(x'+y'+z)$
    $= M_0\, M_2\, M_3\, M_5\, M_6$
  - $f_2 = (x+y+z)(x+y+z)(x+y+z)(x+y+z)$
    $= M_0 M_1 M_2 M_4$
- Any Boolean function can be expressed as
  - a sum of minterms
  - a product of maxterms
  - canonical form

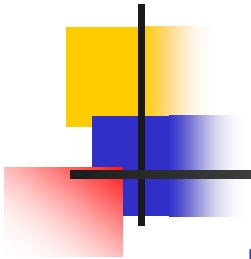
- EXAMPLE 2.4 Express the Boolean function *F=A+B'C* *as a* sum of minterms
  - F = A+B'C
    = A (B+B') + B'C
    = AB +AB' + B'C
    = AB(C+C') + AB'(C+C') + (A+A')B'C
    =ABC+ABC'+AB'C+AB'C'+A'B'C
  - F = A'B'C +AB'C' +AB'C+ABC'+ ABC
    = $m_1 + m_4 + m_5 + m_6 + m_7$
  - F(A,B,C) = $\Sigma(1, 4, 5, 6, 7)$
  - or, built the truth table first

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



- Product of maxterms
  - **Each of the 2^{2n} functions of n binary variables can be also expressed as a product of maxterms.**
  - EXAMPLE 2.5 Express the Boolean function F = xy + x'z as a product of maxterms
  - F = xy + x'z
    = (xy + x') (xy +z)
    = (x+x')(y+x')(x+z)(y+z)
    = (x'+y)(x+z)(y+z)
  - x'+y = x' + y + zz'
    = (x'+y+z)(x'+y+z')
  - F = (x+y+z)(x+y'+z)(x'+y+z)(x'+y+z')
    = $M_0M_2M_4M_5$
  - F(x,y,z) = $\Pi(0,2,4,5)$

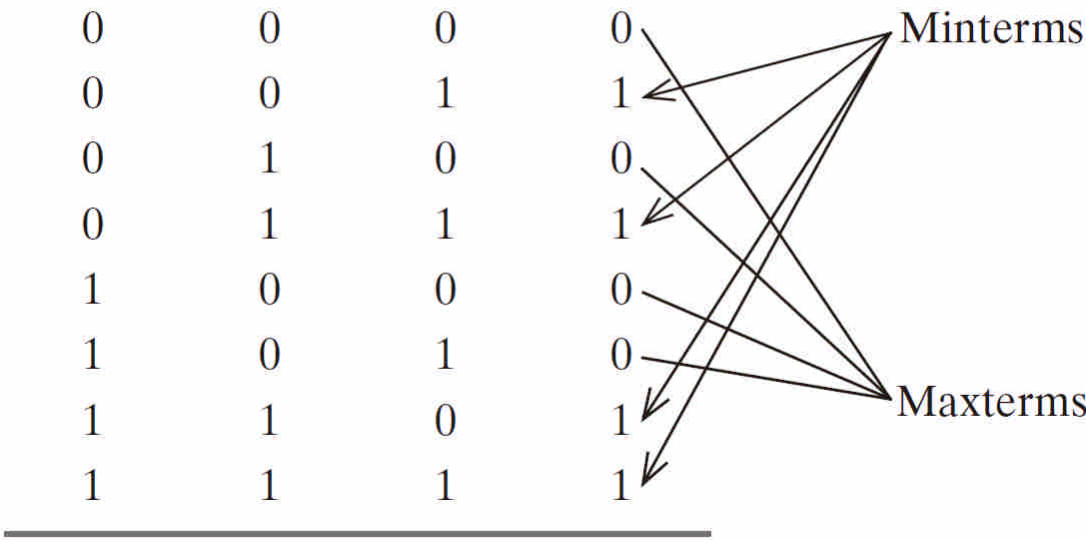


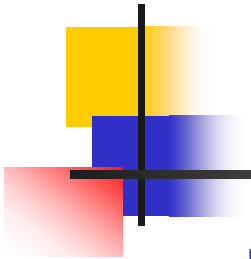
- Conversion between Canonical Forms
  - F(A,B,C) = $\Sigma(1,4,5,6,7)$
  - F‘(A,B,C) = $\Sigma(0,2,3) = m_0+m_1+m_2$
  - By DeMorgan's theorem
    $F = (m_0 + m_2 + m_3)' = m'_0 \bullet m'_2 \bullet m'_3$
    $= M_0M_2M_3 = \Pi(0, 2, 3)$
  - $m_j' = M_j$
  - sum of minterms = product of maxterms
  - interchange the symbols Σ and Π and list those numbers missing from the original form
  - Σ of 1's
  - Π of 0's

- Example
  - $F = xy + x'z$
  - $F(x, y, z) = \Sigma(1, 3, 6, 7)$
  - $F(x, y, z) = \Pi(0, 2, 4, 6)$

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



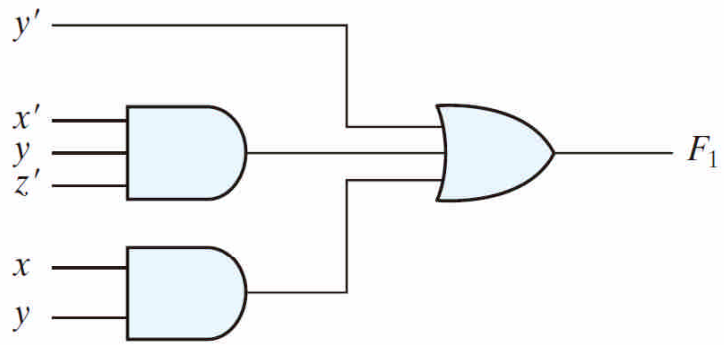


- Standard Forms
  - Canonical forms are seldom used
  - sum of products

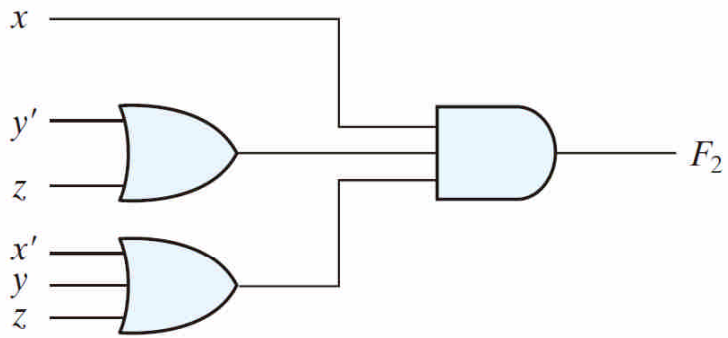
    $F_1 = y' + zy + x'yz'$
  - product of sums

    $F_2 = x(y'+z)(x'+y+z'+w)$
  - $F_3 = AB + C(D + E)$

    $= AB + C(D + E) = AB + CD + CE$

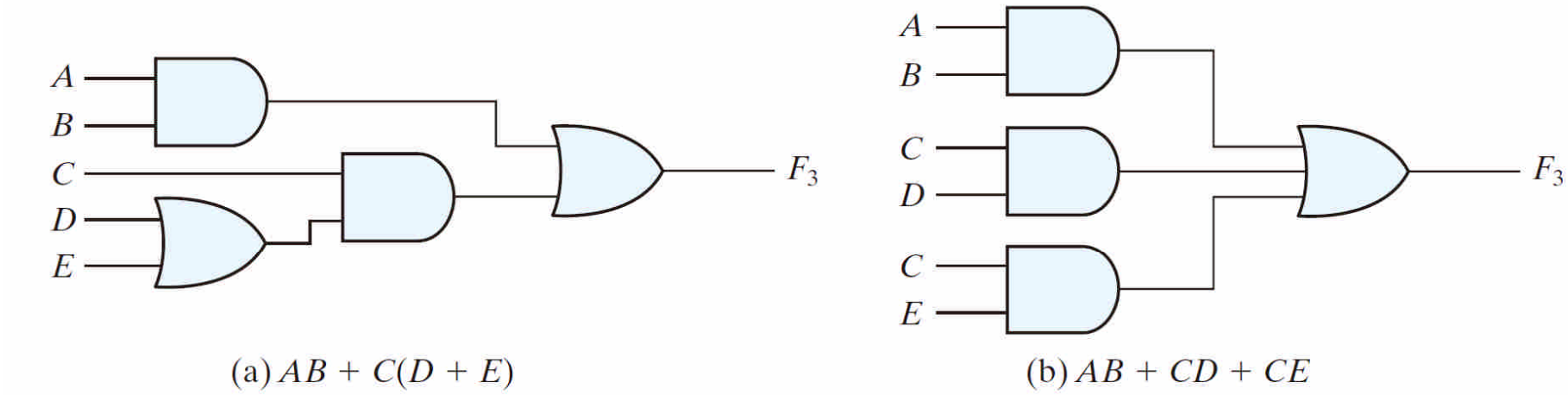


(a) Sum of Products



(b) Product of Sums

FIGURE 2.3
Two-level implementation



(a) $AB + C(D + E)$ (b) $AB + CD + CE$

FIGURE 2.4
Three- and two-level implementation

Other Logic Operations

- 2^n rows in the truth table of n binary variables
- 2^{2^n} functions for n binary variables
- 16 functions of two binary variables

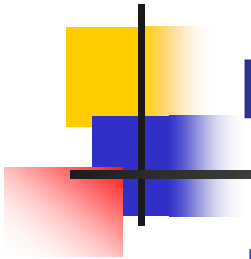
Table 2.7
Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- All the new symbols except for the exclusive-OR symbol are not in common use by digital designers

Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1



Digital Logic Gates

- Boolean expression: AND, OR and NOT operations
- Constructing gates of other logic operations
  - the feasibility and economy
  - the possibility of extending gate's inputs
  - the basic properties of the binary operations
  - the ability of the gate to implement Boolean functions

- Consider the 16 functions
  - two are equal to a constant
  - four are repeated twice
  - inhibition and implication are not commutative or associative
  - the other eight: complement, transfer, AND, OR, NAND, NOR, XOR, and equivalence are used as standard gates
  - complement: inverter
  - transfer: buffer (increasing drive strength)
  - equivalence: XNOR

FIGURE 2.5 Digital logic gates

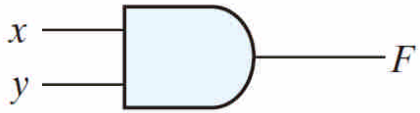
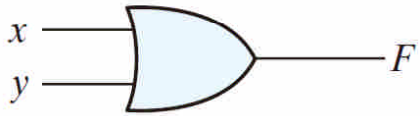
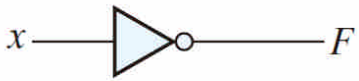
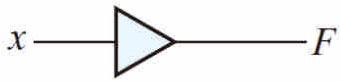
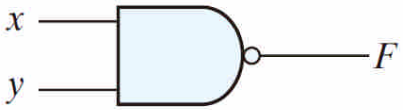
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

FIGURE 2.5 Digital logic gates

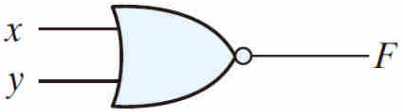
NAND



$F = (xy)'$

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

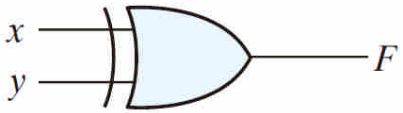
NOR



$F = (x + y)'$

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

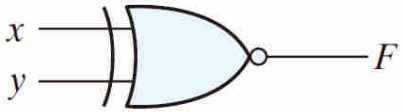
Exclusive-OR (XOR)



$F = xy' + x'y$
$= x \oplus y$

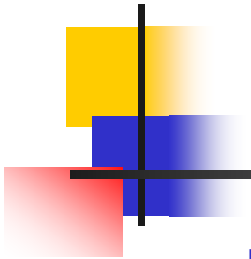
x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR or equivalence



$F = xy + x'y'$
$= (x \oplus y)'$

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



- Extension to multiple inputs
  - A gate can be extended to multiple inputs
    - if its binary operation is commutative and associative
  - AND and OR are commutative and associative
    - (x+y)+z = x+(y+z) = x+y+z
    - (x y)z = x(y z) = x y z

- NAND and NOR are commutative but not associative => they are not extendable

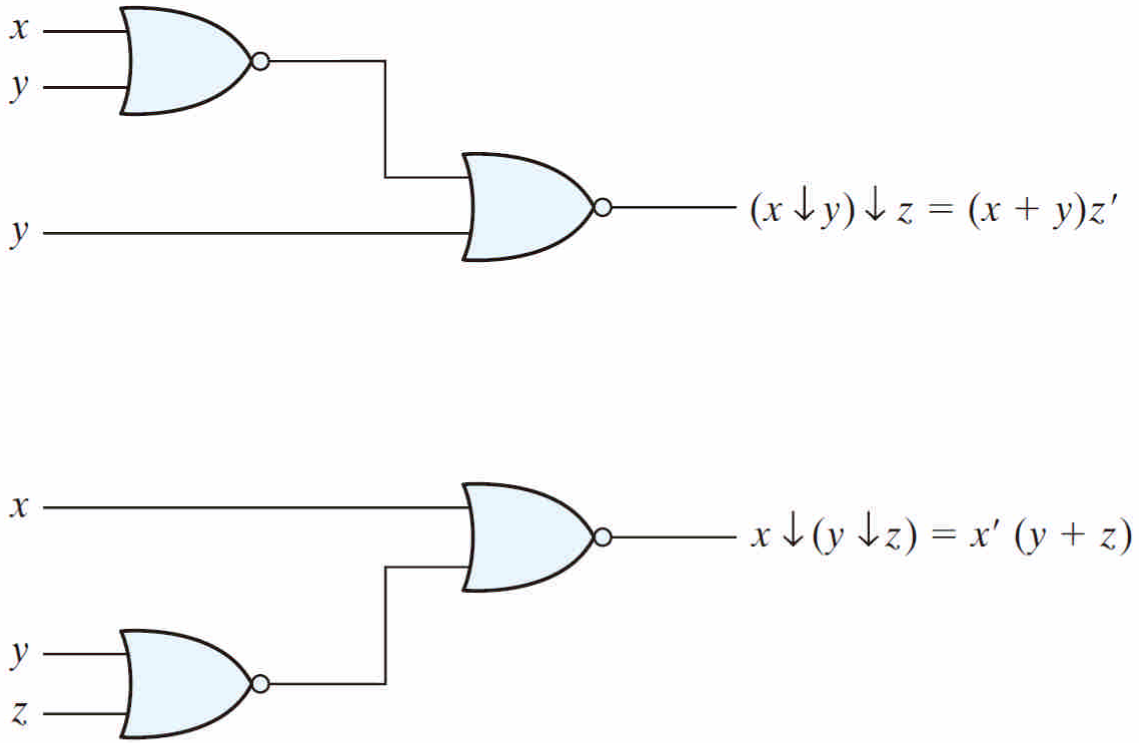


FIGURE 2.6
Demonstrating the nonassociativity of the NOR operator: $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

- Multiple NOR = a complement of OR gate Multiple NAND = a complement of AND
- The cascaded NAND operations = sum of products
- The cascaded NOR operations = product of sums

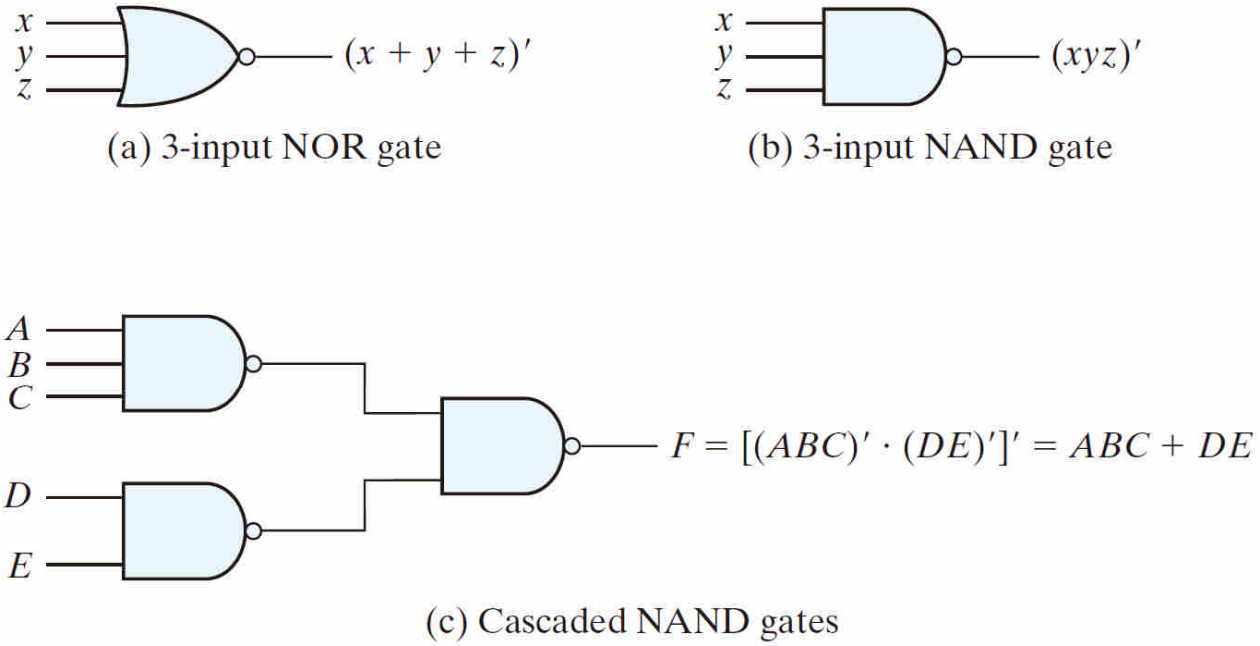
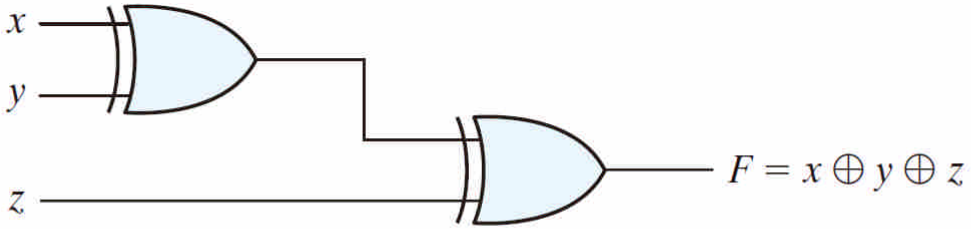
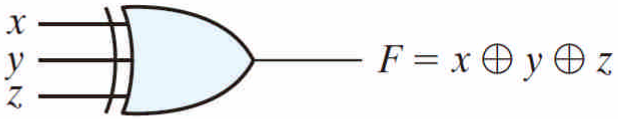


FIGURE 2.7
Multiple-input and cascaded NOR and NAND gates

- The XOR and XNOR gates are commutative and associative
- Multiple-input XOR gates are uncommon?
- XOR is an odd function: it is equal to 1 if the inputs variables have an odd number of 1's



(a) Using 2-input gates



(b) 3-input gate

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

(c) Truth table

FIGURE 2.8
Three-input exclusive-OR gate

- Positive and Negative Logic
  - two signal values <=> two logic values
  - positive logic: H=1; L=0
  - negative logic: H=0; L=1
- Consider a TTL gate
  - a positive logic NAND gate
  - a negative logic OR gate
  - the positive logic is used in this book

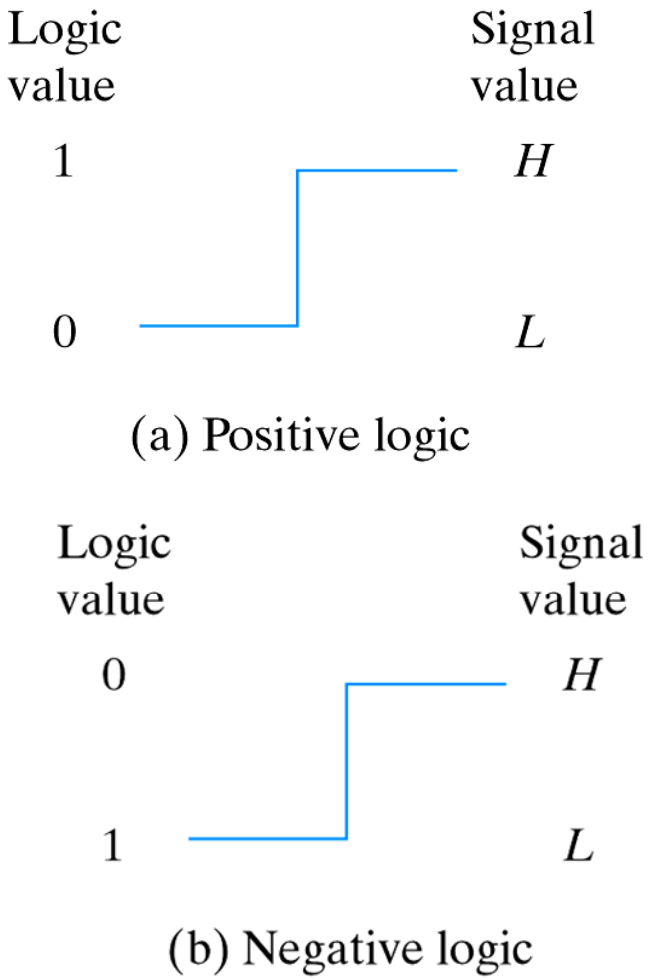
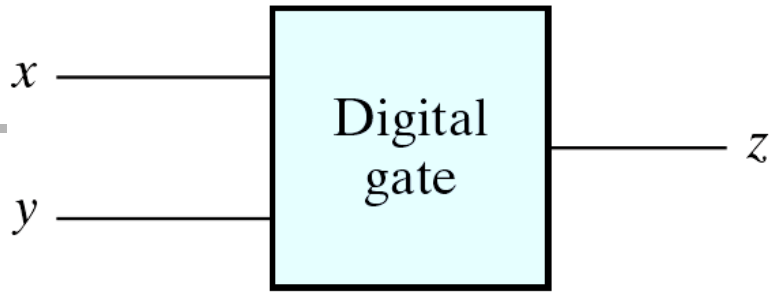


FIGURE 2.9
Signal assignment and logic polarity

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

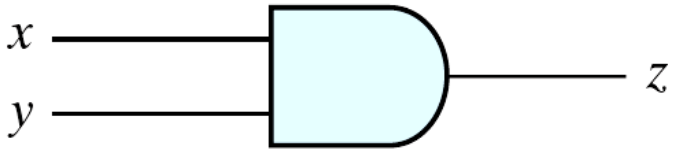
(a) Truth table with H and L



(b) Gate block diagram

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

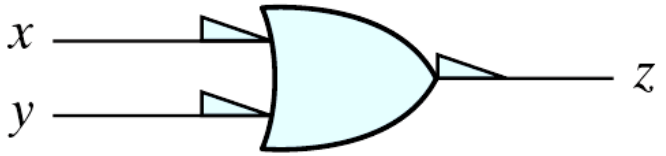
(c) Truth table for positive logic



(d) Positive logic AND gate

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

(e) Truth table for negative logic

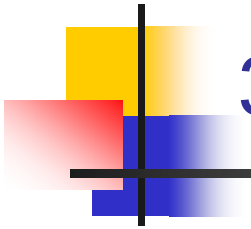


(f) Negative logic OR gate

FIGURE 2.10
Demonstration of positive and negative logic

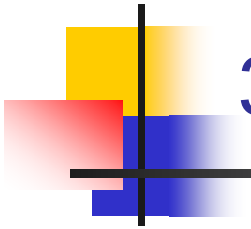
Gate-Level Minimization

Chapter 3



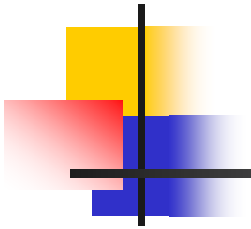
3-1 Introduction

- *Gate-level minimization* is the design task of finding an optimal gate-level implementation of Boolean functions describing a digital circuit.



3-2 The Map Method

- The complexity of the digital logic gates
  - the complexity of the algebraic expression
- Logic minimization
  - algebraic approaches: lack specific rules
  - the Karnaugh map (or K-map)
    - a simple straight forward procedure
    - a pictorial form of a truth table
    - applicable if the # of variables < 7
- A diagram made up of squares
  - each square represents one minterm



- Boolean function
  - sum of minterms
  - sum of products (or product of sum) in the simplest form
  - a minimum number of terms
  - a minimum number of literals
  - The simplified expression may not be unique

Two-Variable Map

- A two-variable map
  - four minterms
  - x' = row 0; x = row 1
  - y' = column 0;

    y = column 1
  - a truth table in square diagram
  - xy
  - x+y =

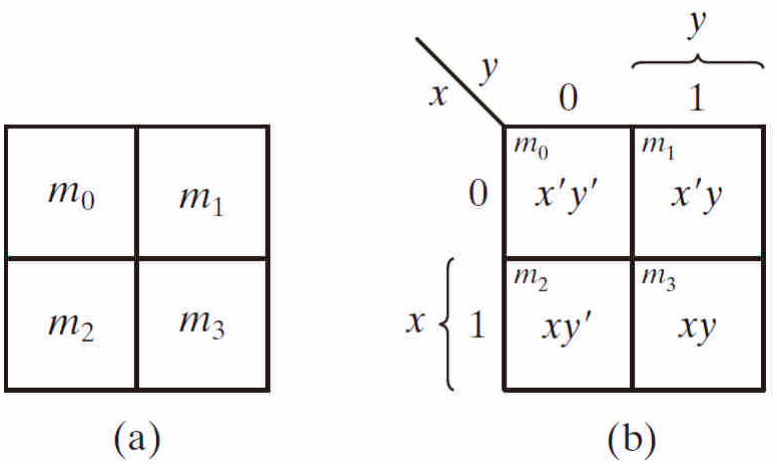


FIGURE 3.1
Two-variable K-map

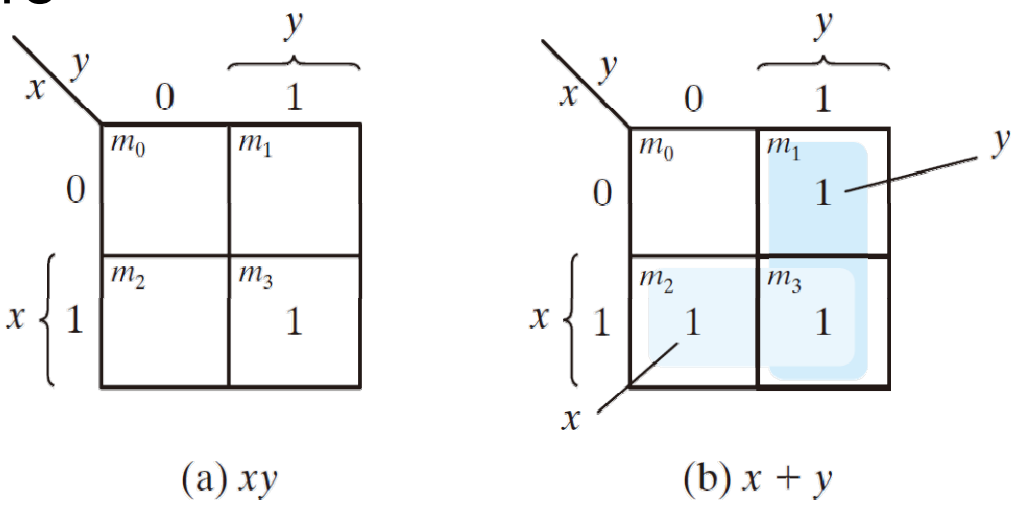


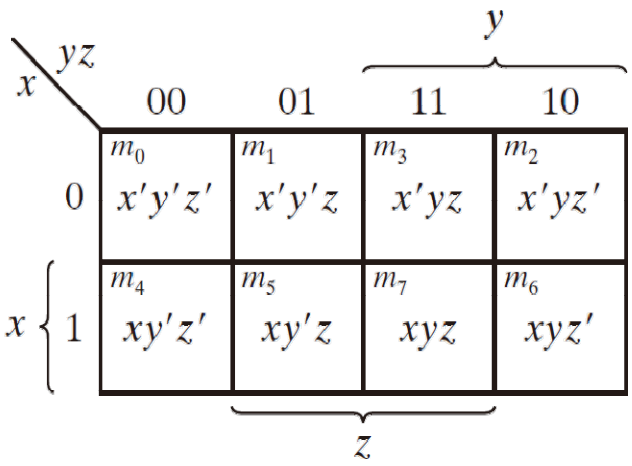
FIGURE 3.2
Representation of functions in the map

Three-variable map

- eight minterms
- the Gray code sequence
- any two adjacent squares in the map differ by only on variable
  - primed in one square and unprimed in the other
  - e.g. m_5 and m_7 can be simplified
  - $m_5 + m_7 = xy'z + xyz = xz\,(y'+y) = xz$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)



(b)

FIGURE 3.3
Three-variable K-map

- Example 3-1
  - F(x,y,z) = Σ(2,3,4,5)
  - F = x'y + xy'

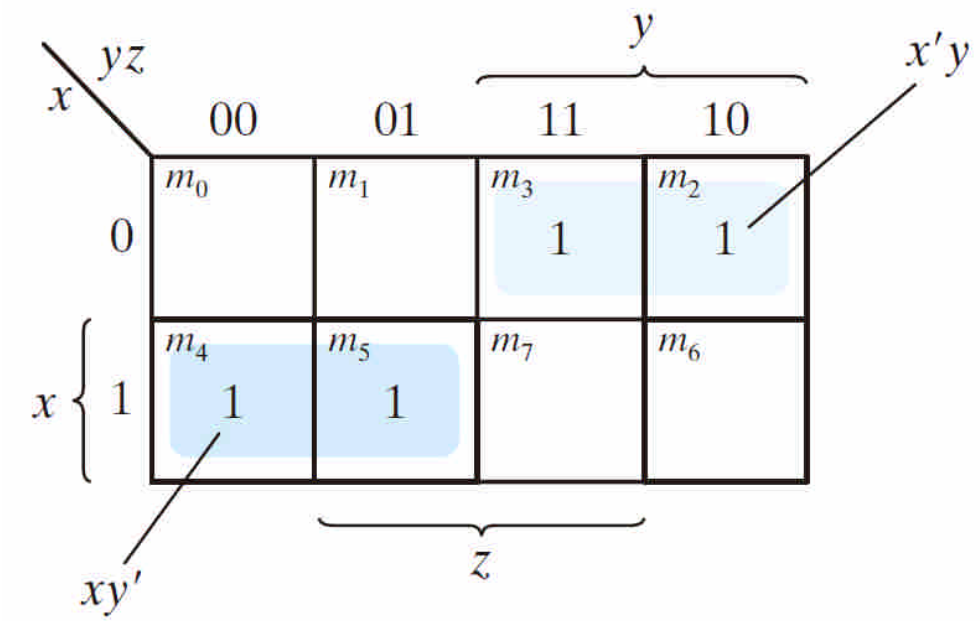


FIGURE 3.4
Map for Example 3.1, $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

- m_0 and m_2 (m_4 and m_6) are adjacent
- $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y'+y) = x'z'$
- $m_4 + m_6 = xy'z' + xyz' = xz'(y'+y) = xz'$

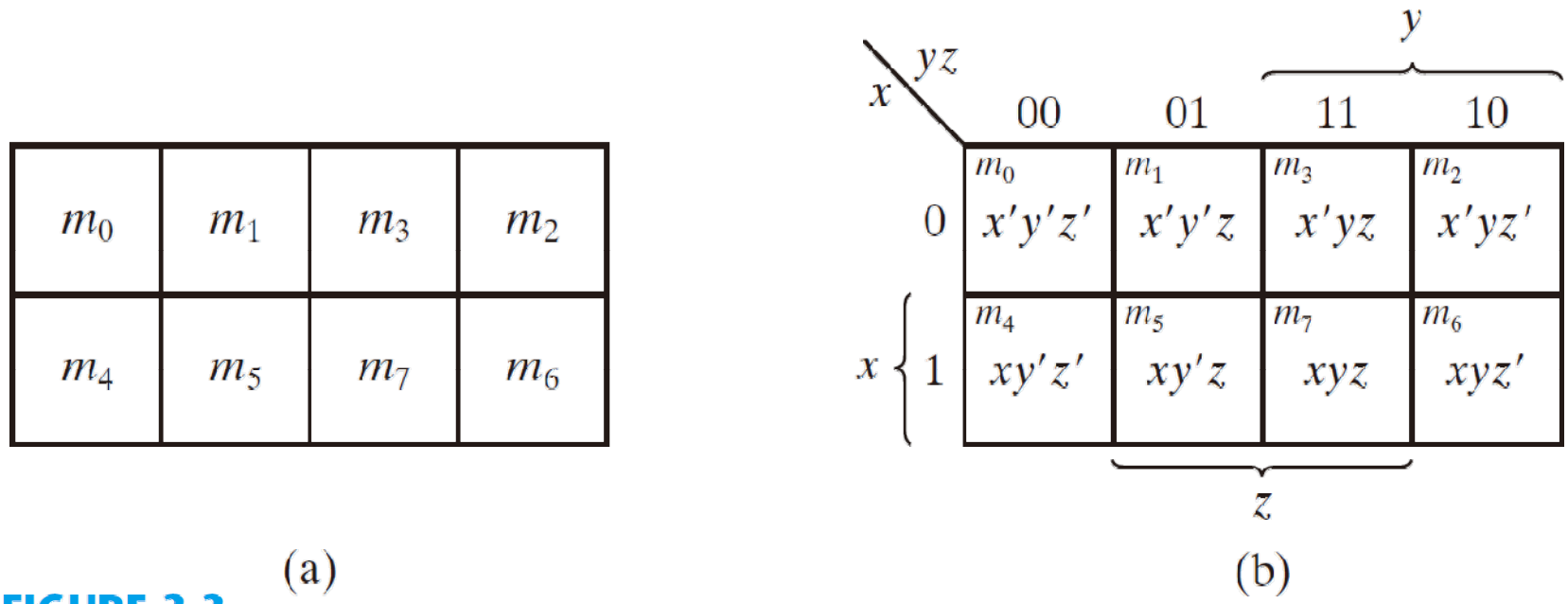


FIGURE 3.3
Three-variable K-map

- Example 3-2
  - F(x,y,z) = Σ(3,4,6,7) = *yz*+ *xz*'

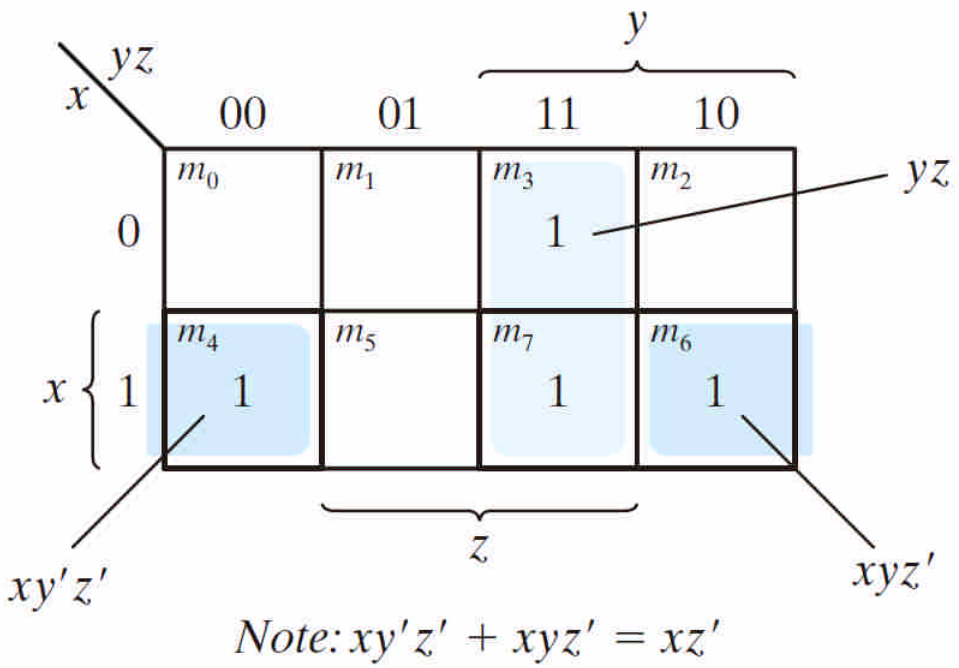


FIGURE 3.5
Map for Example 3.2, $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

Four adjacent squares

- 2, 4, 8 and 16 squares
- $m_0+m_2+m_4+m_6 = x'y'z'+x'yz'+xy'z'+xyz'$
  $= x'z'(y'+y) +xz'(y'+y)$
  $= x'z' + xz' = z'$
- $m_1+m_3+m_5+m_7 = x'y'z+x'yz+xy'z+xyz$
  $=x'z(y'+y) + xz(y'+y)$
  $=x'z + xz = z$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

x \ yz	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

y (over 11, 10); z (under 11, 10); x (rows 0, 1)

(b)

FIGURE 3.3
Three-variable K-map

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

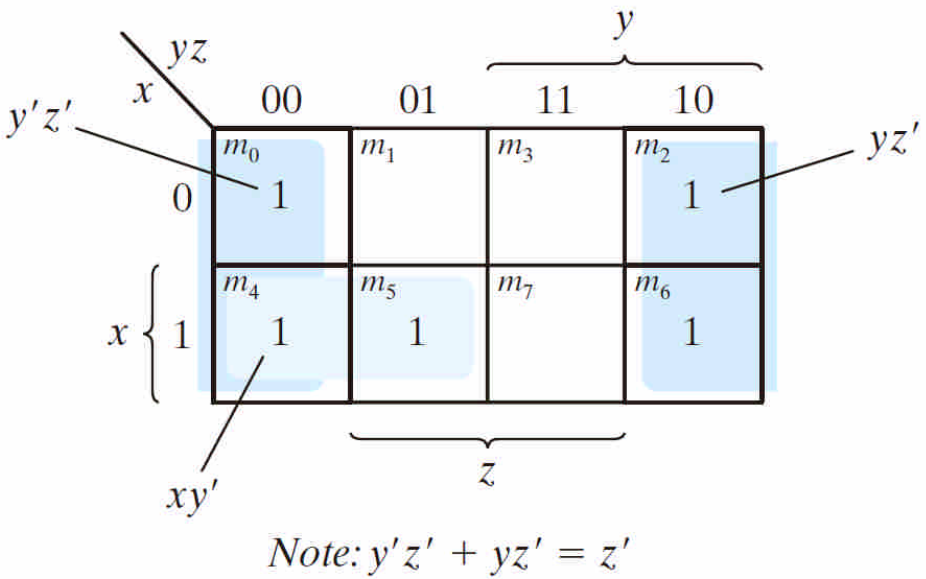


FIGURE 3.6
Map for Example 3.3, $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

- Example 3-4
  - F = A'C + A'B + AB'C + BC
  - express it in sum of minterms
  - find the minimal sum of products expression

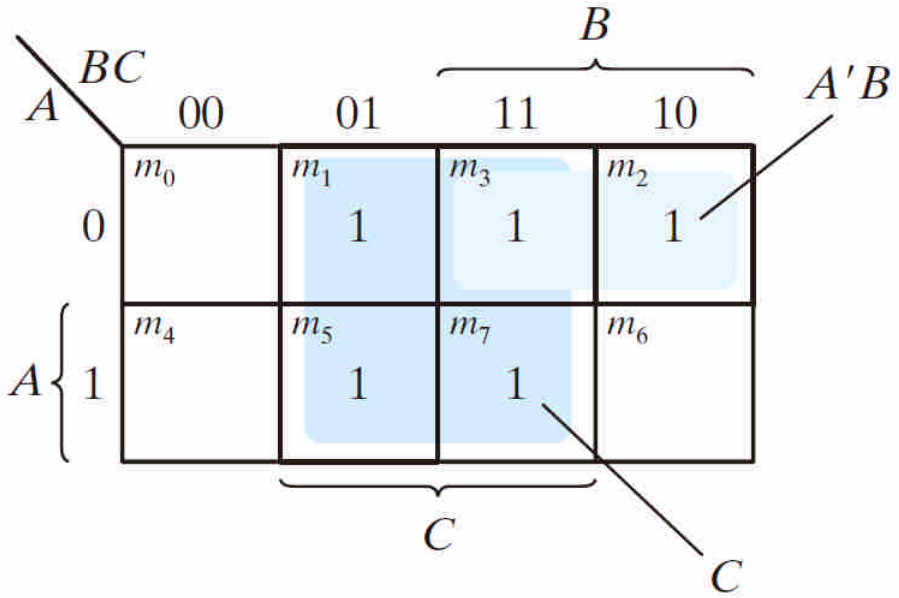


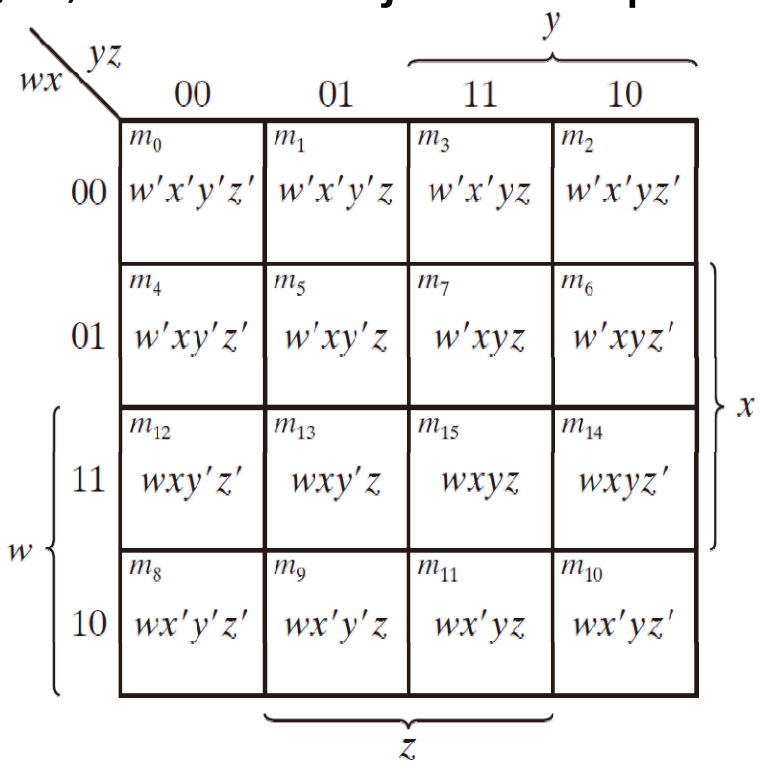
FIGURE 3.7
Map of Example 3.4, $A'C + A'B + AB'C + BC = C + A'B$

3-3 Four-Variable Map

- The map
  - 16 minterms
  - combinations of 2, 4, 8, and 16 adjacent squares

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}

(a)

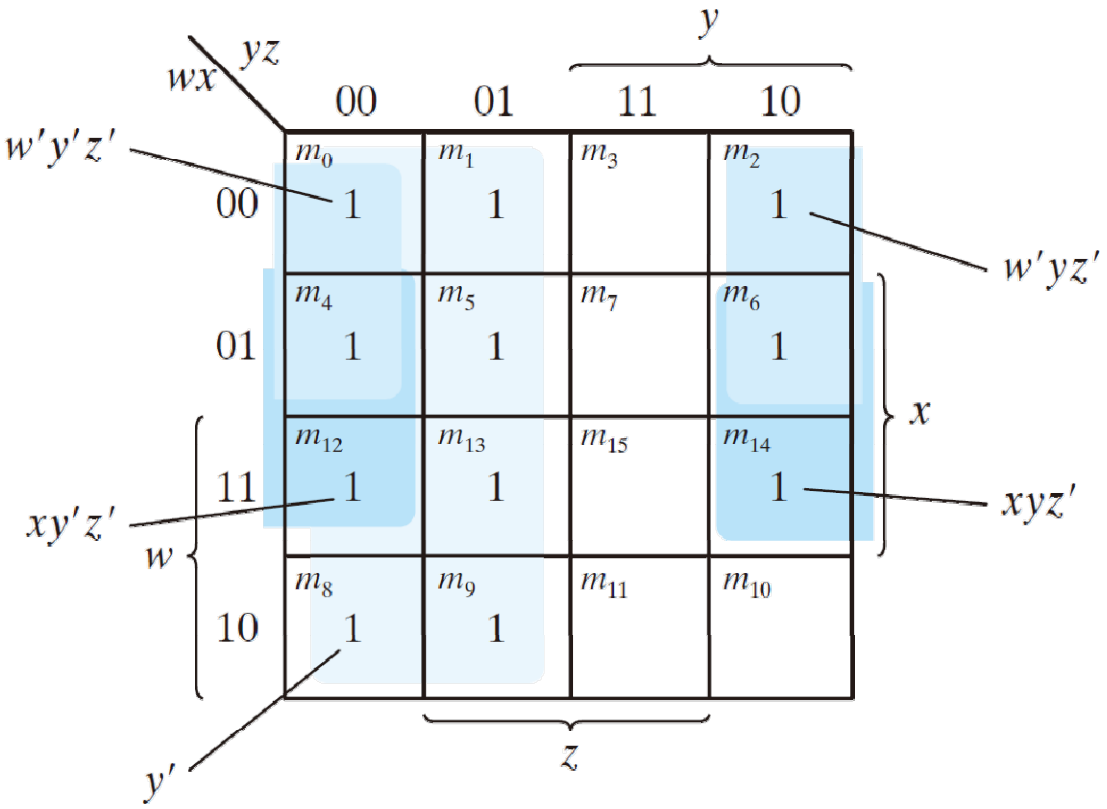


(b)

FIGURE 3.8
Four-variable map

Example 3-5

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



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

$xy'z' + xyz' = xz'$

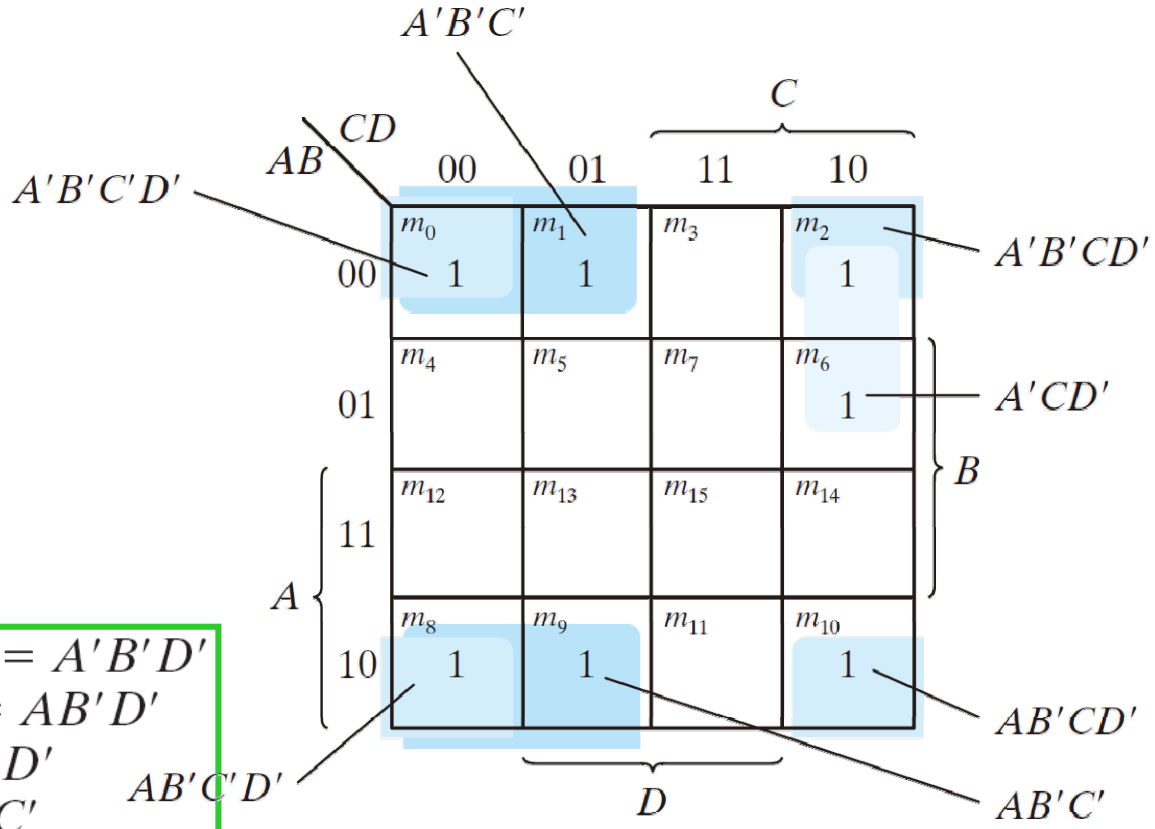
- F = y'+w'z'+xz'

FIGURE 3.9

Map for Example 3.5, $F(w, x, y, z) = \Sigma(0,1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

Example 3-6 Simplify the Boolean function

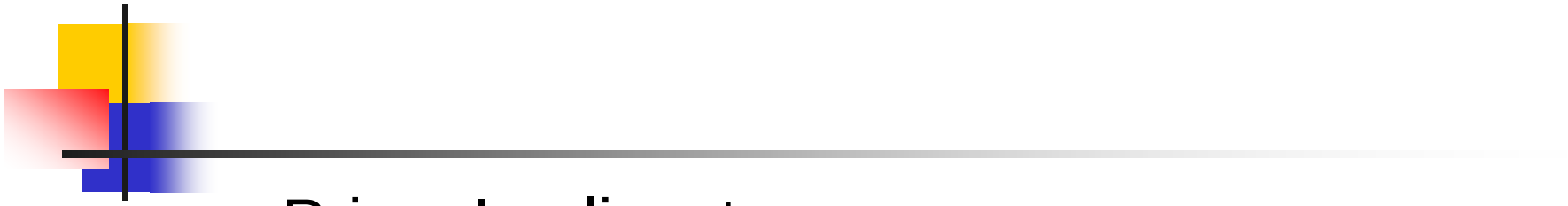
F = A′B′C′ + B′CD′ + A′B′CD′ + AB′C′



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

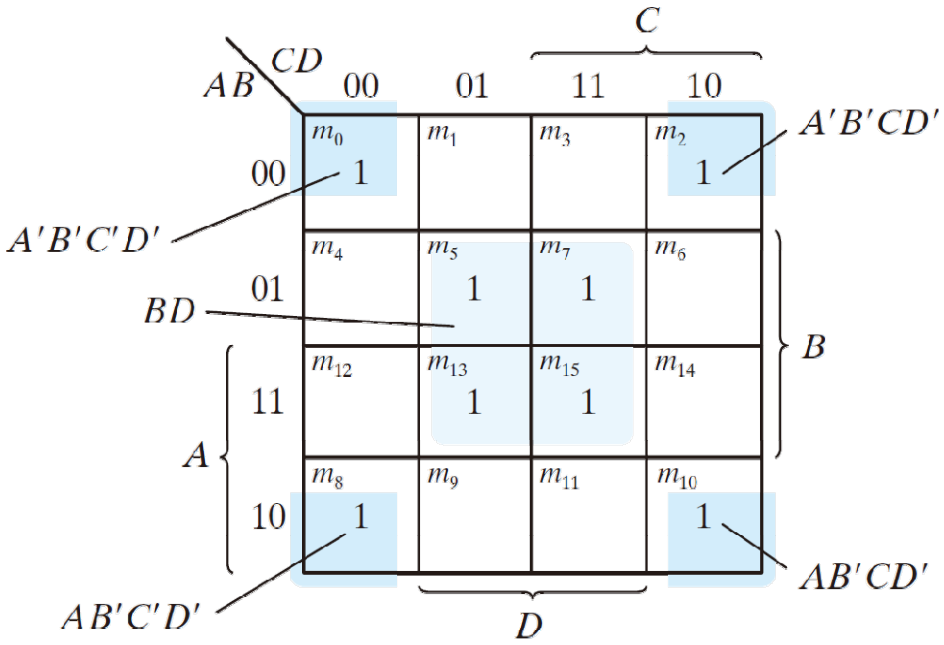
FIGURE 3.10
Map for Example 3.6, $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$



- Prime Implicants
  - all the minterms are covered
  - minimize the number of terms
  - a prime implicant: a product term obtained by combining the maximum possible number of adjacent squares (combining all possible maximum numbers of squares)
  - essential: a minterm is covered by only one prime implicant
  - the essential P.I. must be included

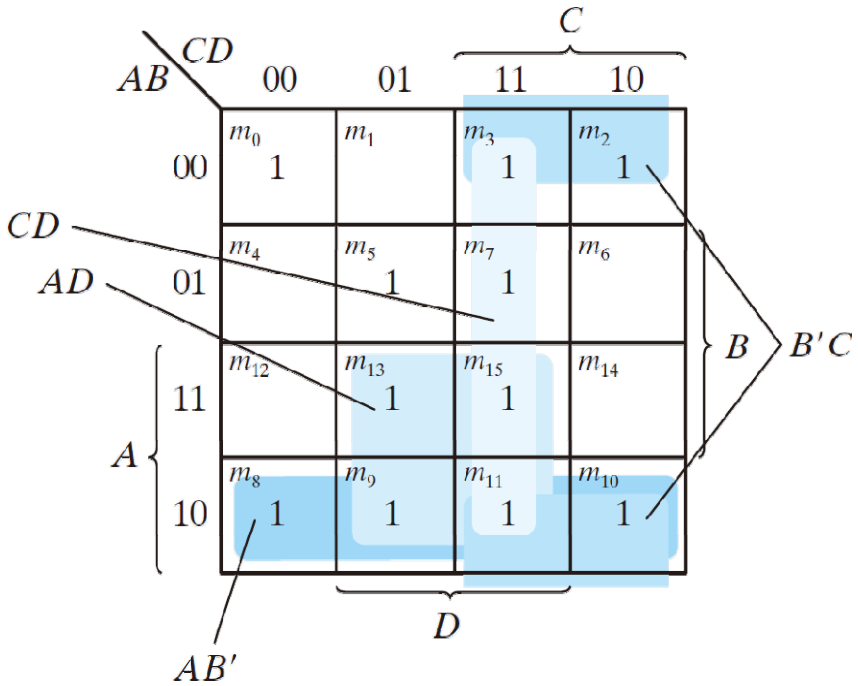
Consider $F(A,B,C,D) = \sum(0,2,3,5,7,8,9,10,11,13,15)$

- the simplified expression may not be unique
- F = BD+B'D'+CD+AD = BD+B'D'+CD+AB′
  = BD+B'D'+B'C+AD = BD+B'D'+B'C+AB'



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$

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

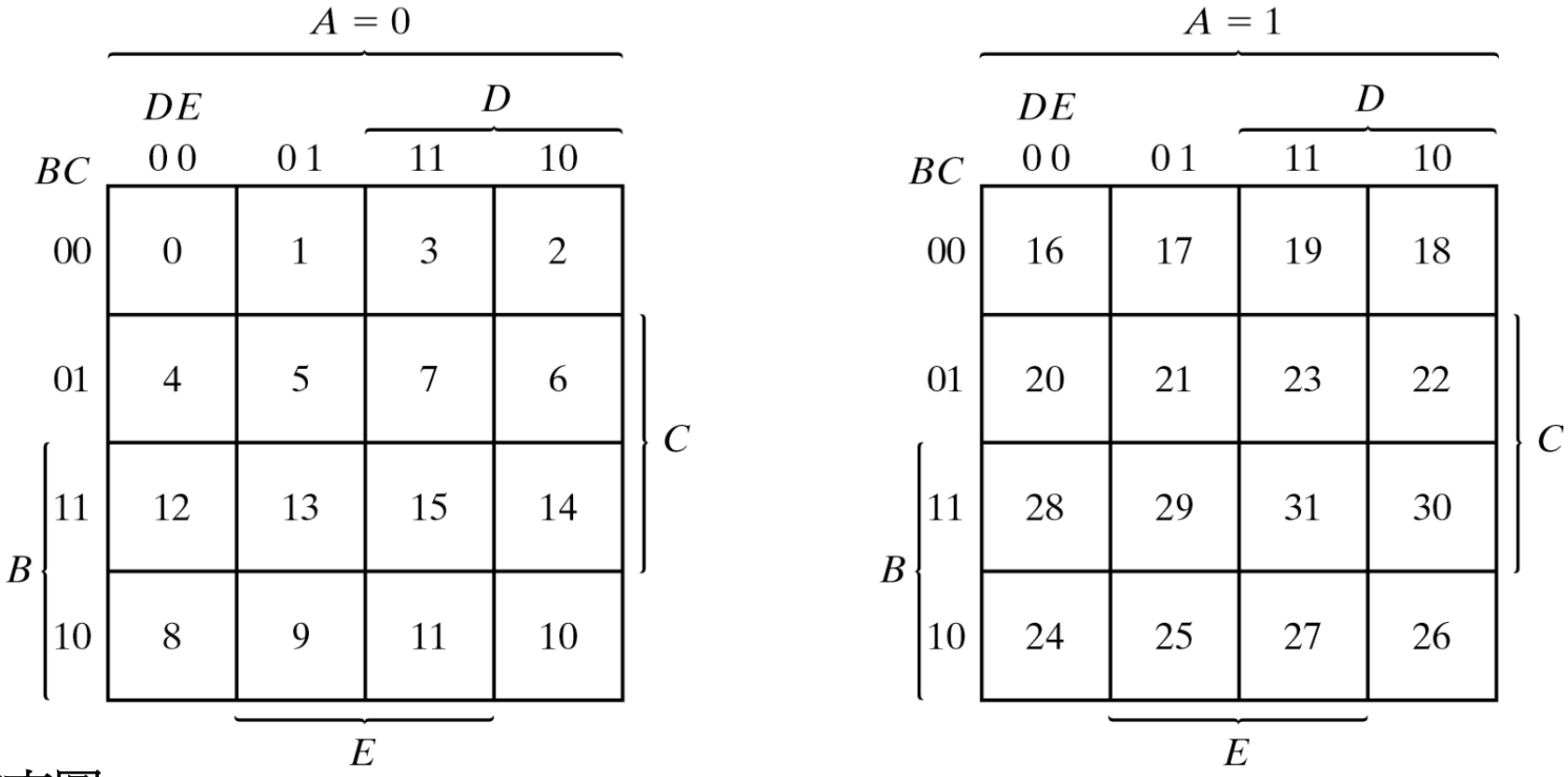


(b) Prime implicants CD, $B'C$, AD, and AB'

FIGURE 3.11
Simplification using prime implicants

3-4 Five-Variable Map

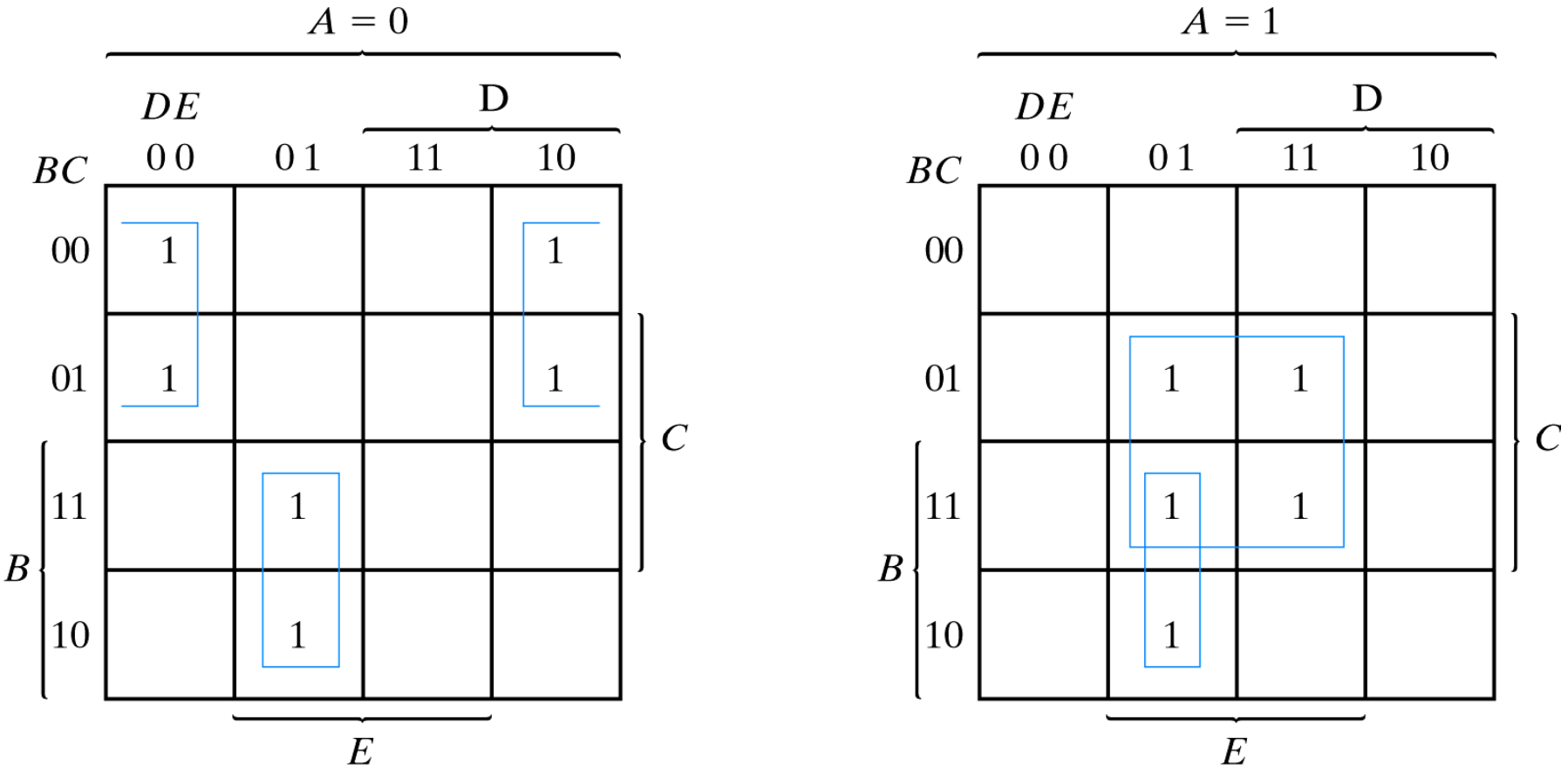
- Map for more than four variables becomes complicated
  - five-variable map: two four-variable map (one on the top of the other)



補充圖 Five-variable Map

Example 補充題

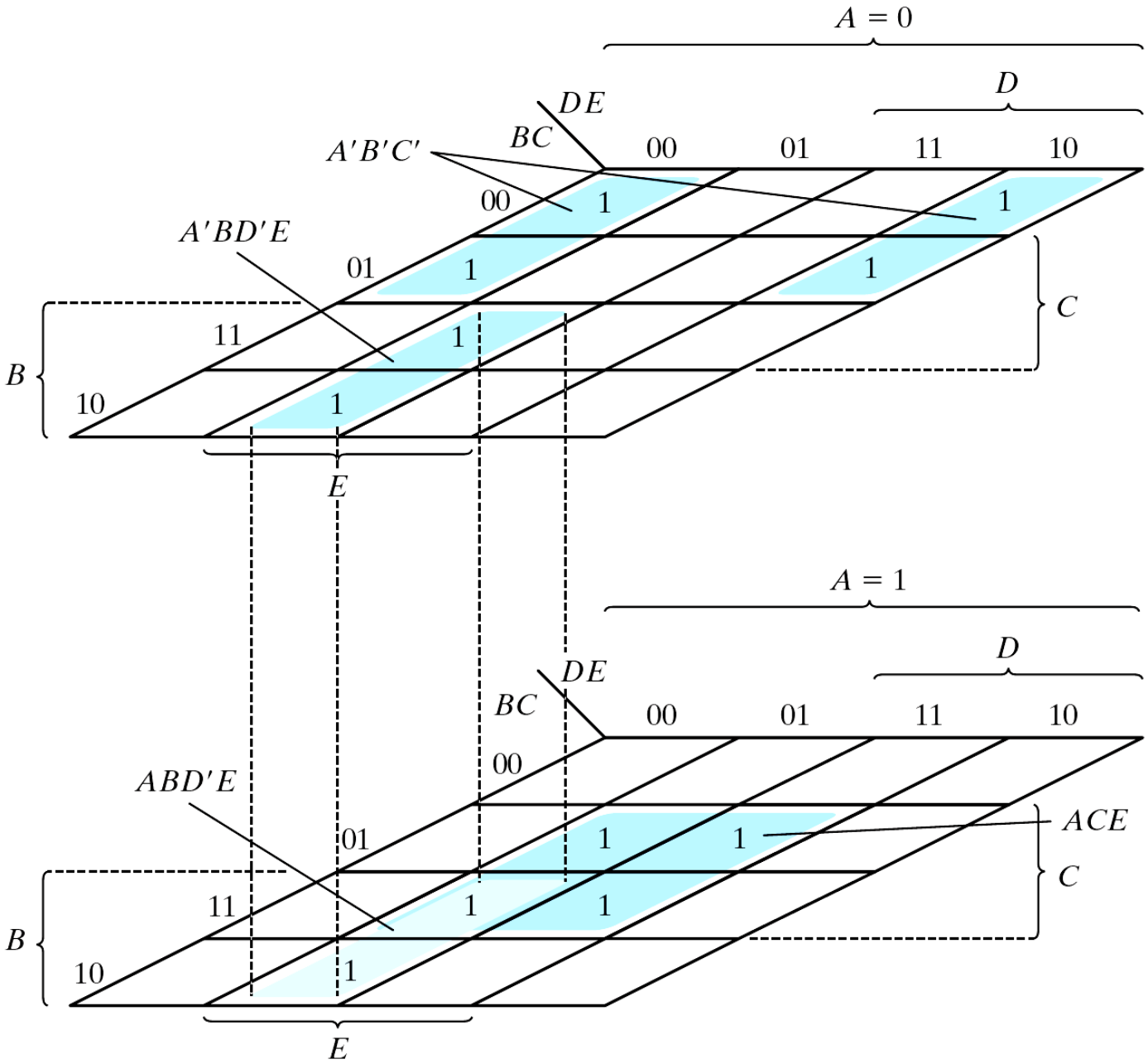
- F = Σ(0,2,4,6,9,13,21,23,25,29,31)



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

F = A'B'E'+BD'E+ACE

♣ Another Map for Example 補充



3-4 Product of Sums Simplification

- Approach #1
  - Simplified *F′* in the form of sum of products
  - Apply DeMorgan's theorem *F* = (*F′*)′
  - *F′*: sum of products => *F*: product of sums
- Approach #2: duality
  - combinations of maxterms (it was minterms)
  - M_0M_1 = (A+B+C+D)(A+B+C+D')
    = (A+B+C)+(DD')
    = A+B+C

AB \ CD	00	01	11	10
00	M_0	M_1	M_3	M_2
01	M_4	M_5	M_7	M_6
11	M_{12}	M_{13}	M_{15}	M_{14}
10	M_8	M_9	M_{11}	M_{10}

- Example 3-7
  - F = Σ(0,1,2,5,8,9,10)

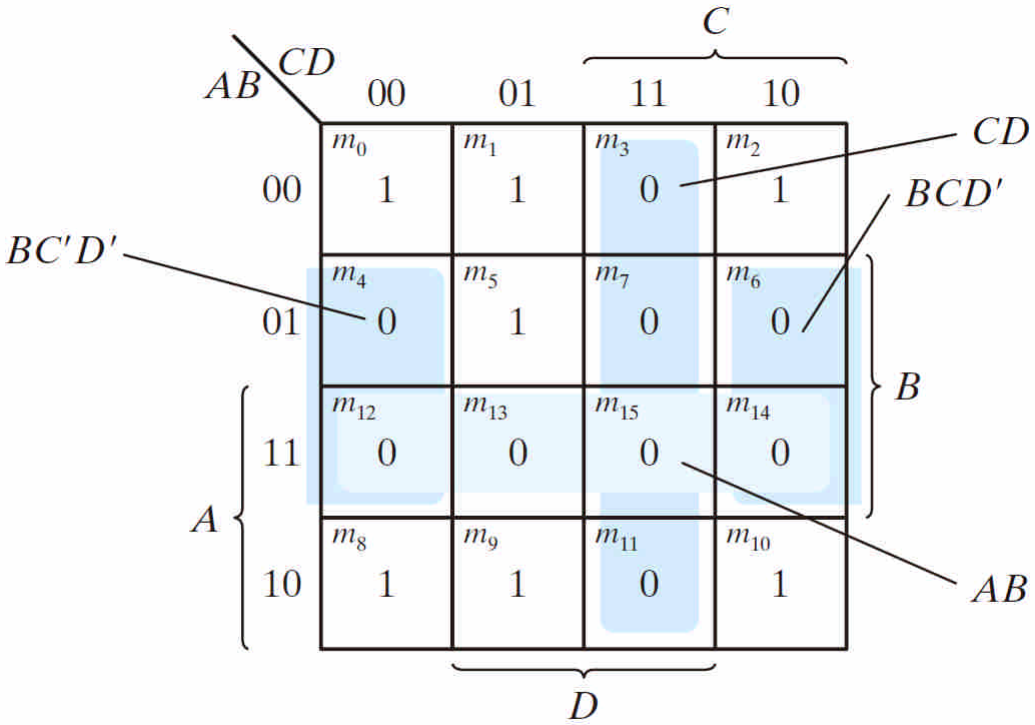
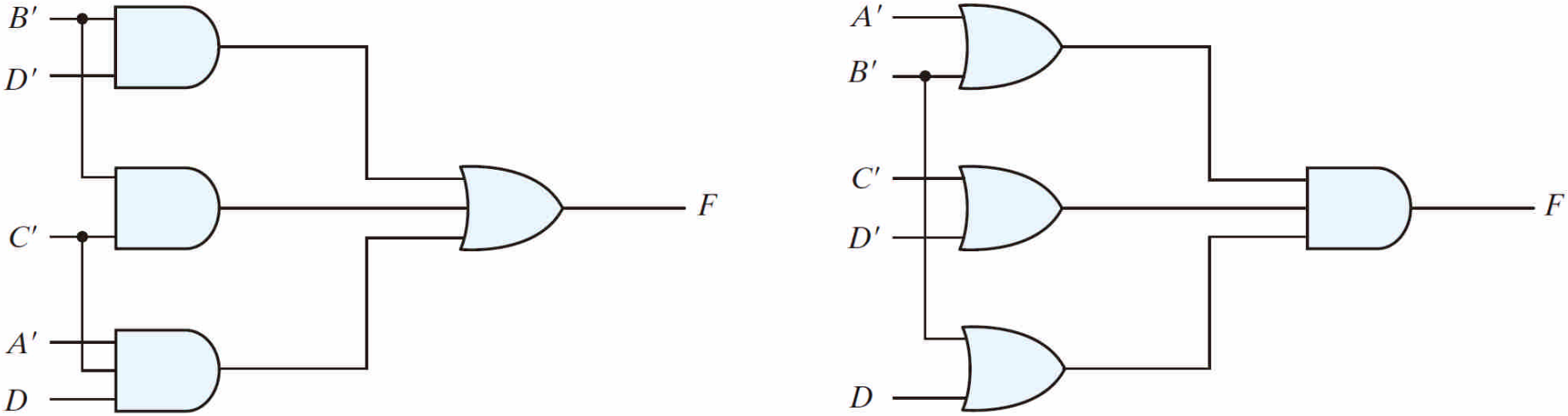


FIGURE 3.12
Map for Example 3.7, $F(A, B, C, D) = \Sigma(0,1,2,5,8,9,10) = B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$

Note: $BC'D' + BCD' = BD'$

  - F' = AB+CD+BD'
  - Apply DeMorgan's theorem; F=(A'+B')(C'+D')(B'+D)
  - Or think in terms of maxterms

- Gate implementation of the function of Example 3-7



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

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

FIGURE 3.13
Gate implementations of the function of Example 3.7

- Consider the function defined in Table 3.1.

In sum-of-minterm:

$$F(x, y, z) = \sum(1, 3, 4, 6)$$

In sum-of-maxterm:

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

Taking the complement of F′

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

Table 3.1
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

- Consider the function defined in Table 3.1.

Combine the 1's:

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

Combine the 0's :

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

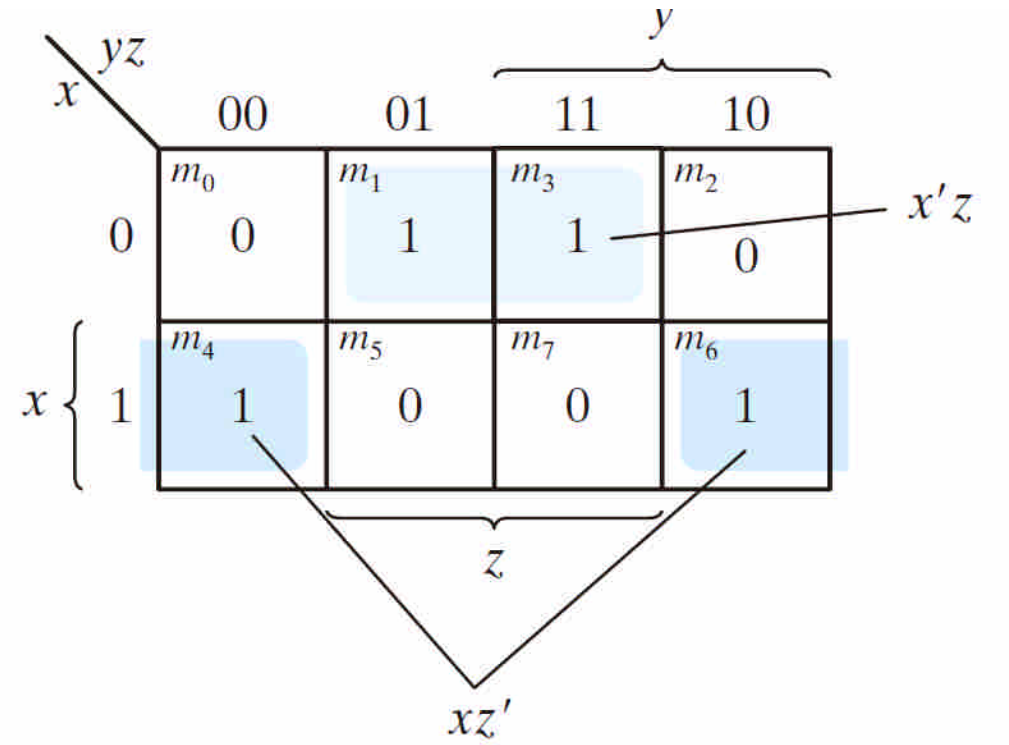


FIGURE 3.14
Map for the function of Table 3.1

3-5 Don't-Care Conditions

- The value of a function is not specified for certain combinations of variables
  - BCD; 1010-1111: don't care
- The don't care conditions can be utilized in logic minimization
  - can be implemented as 0 or 1
- Example 3-8
  - $F(w,x,y,z) = \Sigma(1,3,7,11,15)$
  - $d(w,x,y,z) = \Sigma(0,2,5)$

- Figure 3.15(a) : F = yz + w'x'
- Figure 3.15(b) : F = yz + w'z
- F = Σ(0,1,2,3,7,11,15) ; F = Σ(1,3,5,7,11,15)
- either expression is acceptable

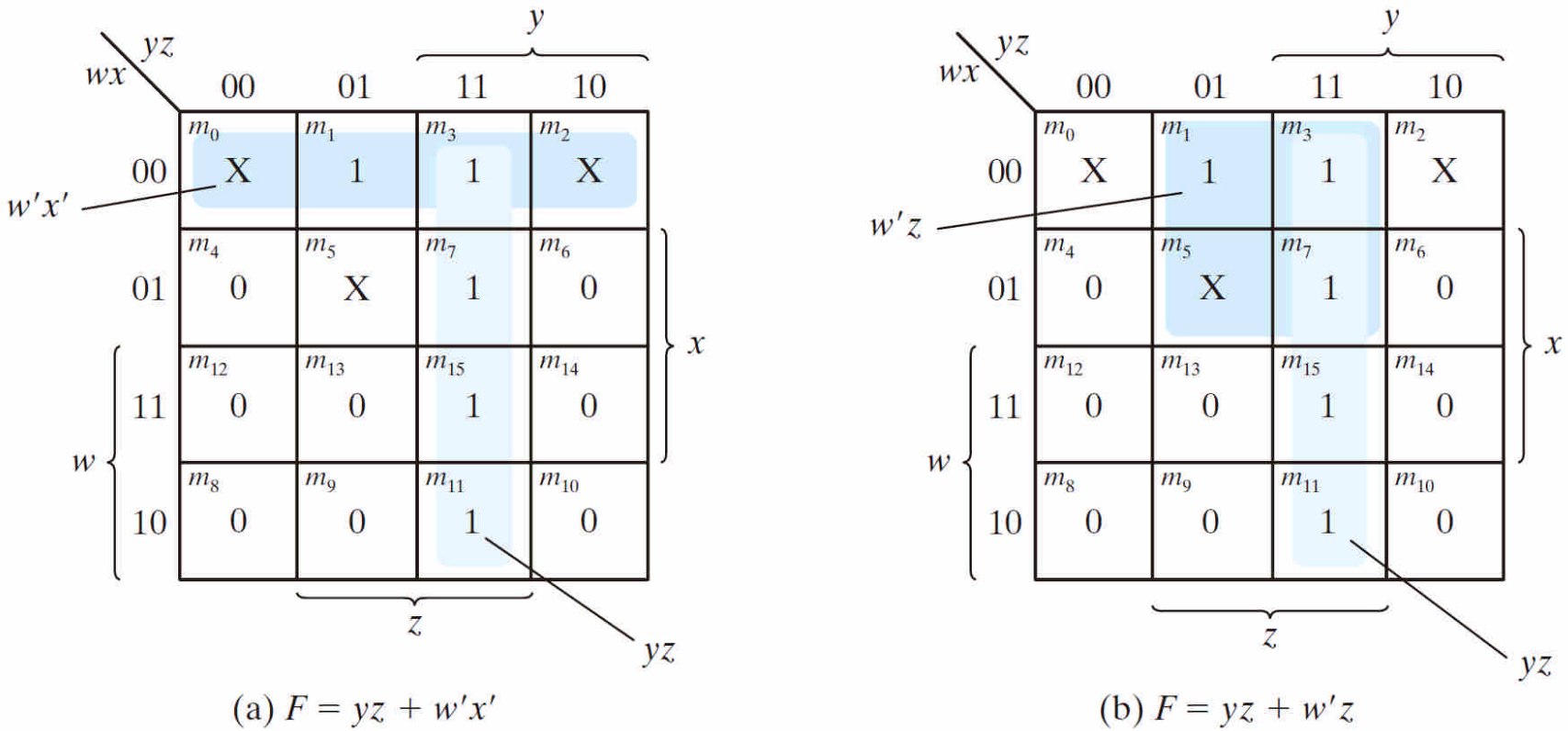


FIGURE 3.15
Example with don't-care conditions

- Also apply to products of sum

3-6 NAND and NOR Implementation

- NAND gate is a universal gate
  - can implement any digital system

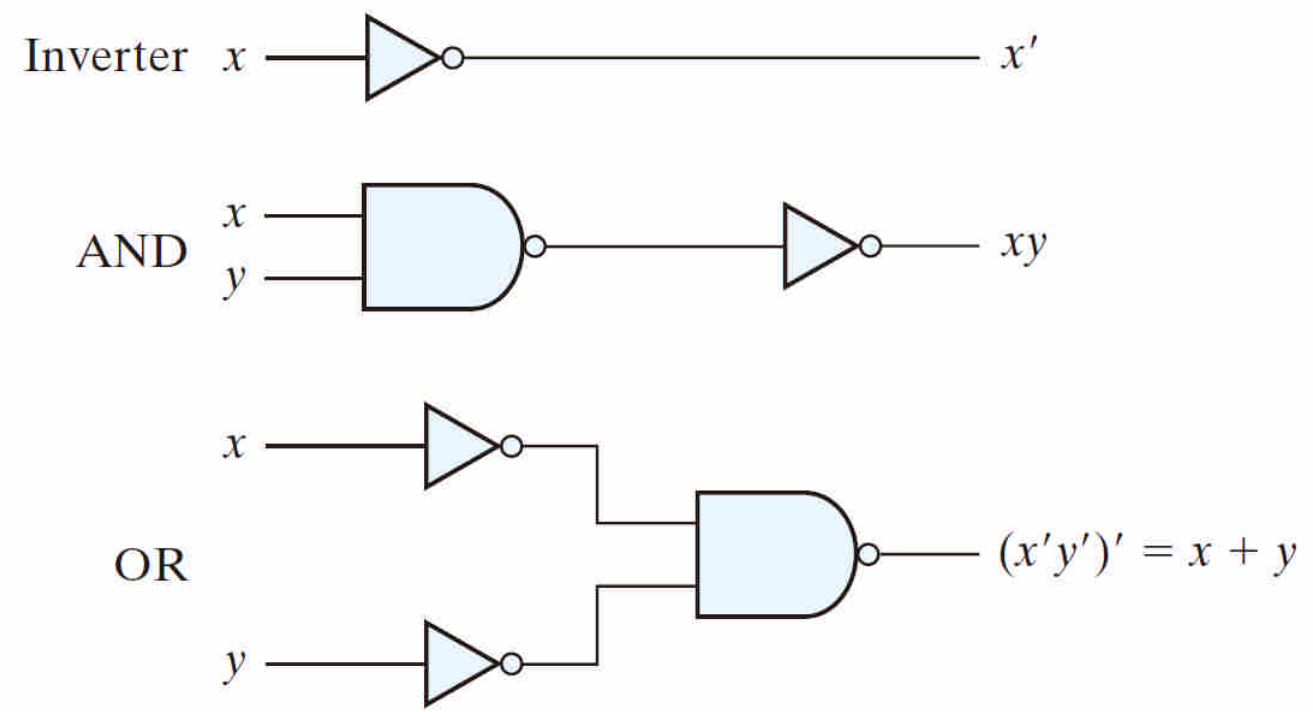


FIGURE 3.16
Logic operations with NAND gates

- Two graphic symbols for a NAND gate

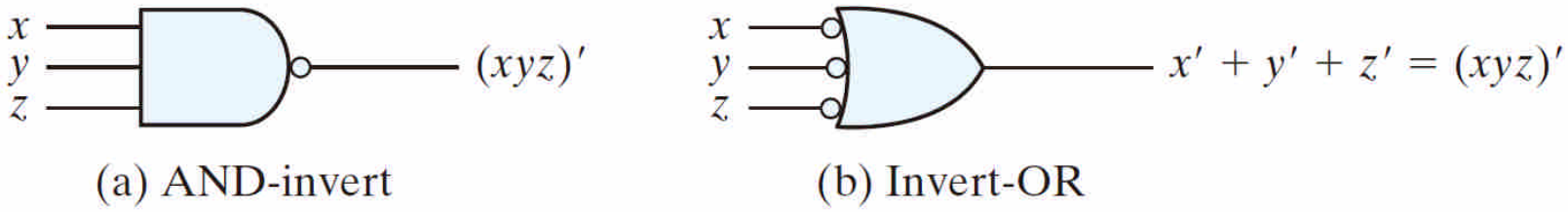


FIGURE 3.17
Two graphic symbols for a three-input NAND gate

Two-level Implementation

- two-level logic
- NAND-NAND = sum of products
- Example: F = AB+CD
- F = ((AB)' (CD)')' =AB+CD

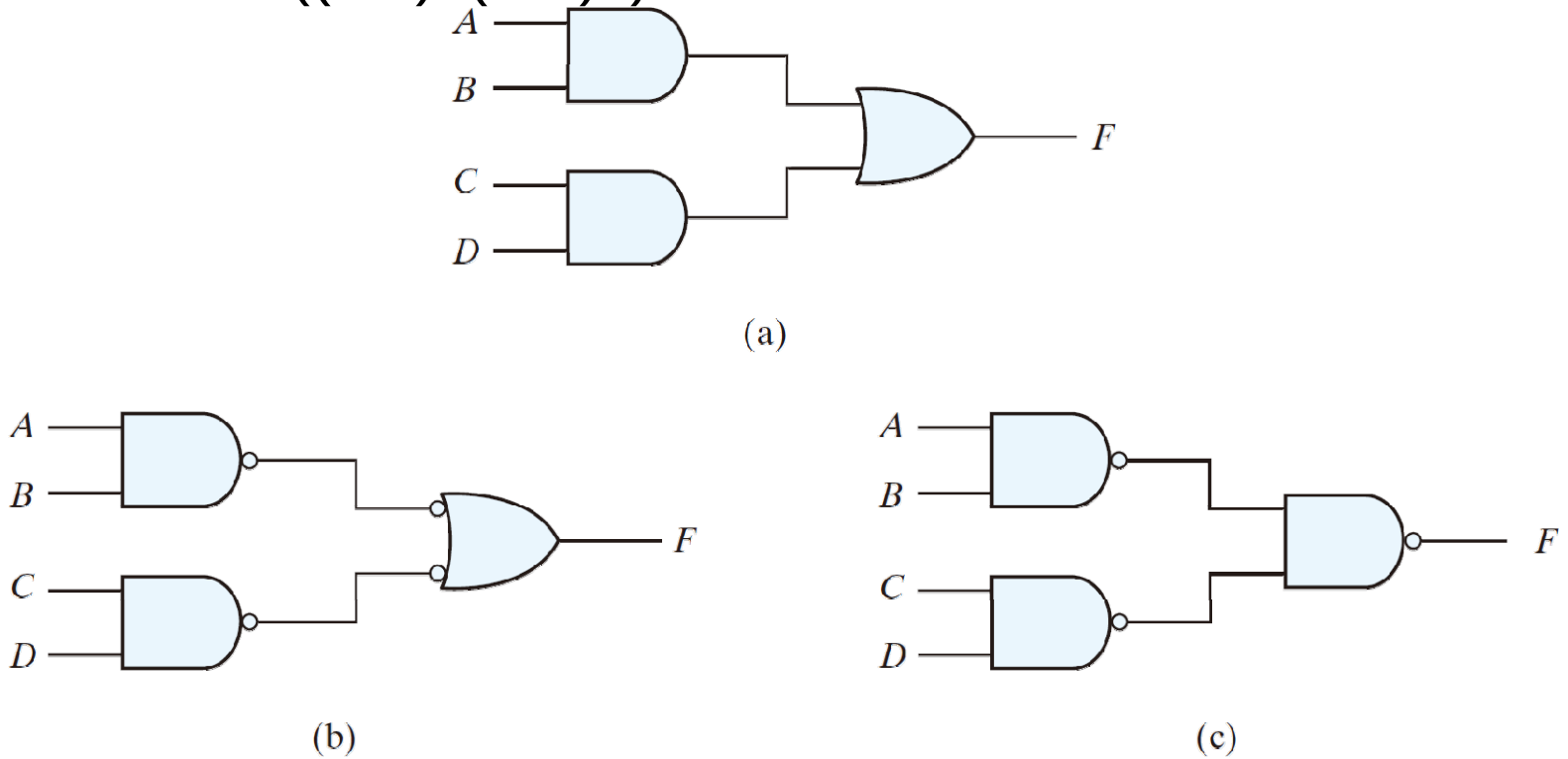
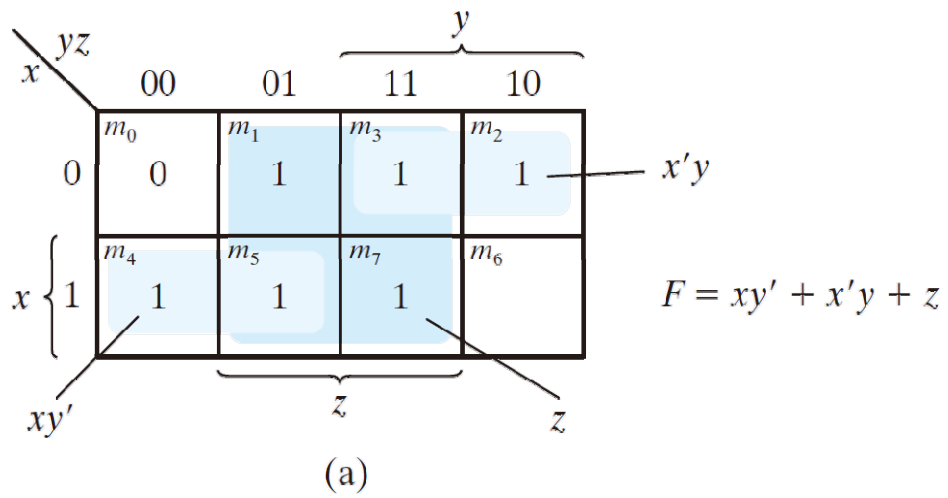


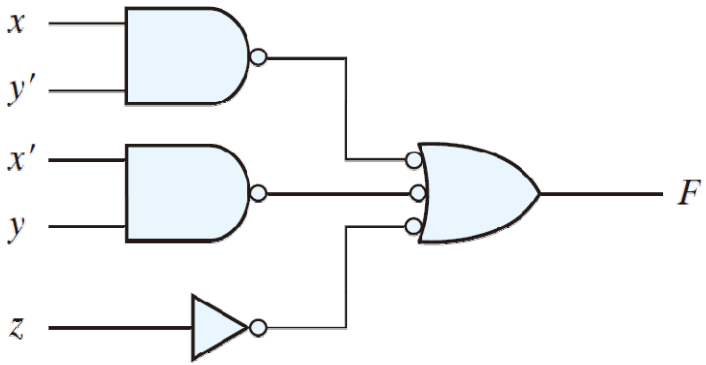
FIGURE 3.18
Three ways to implement $F = AB + CD$

Example 3-9

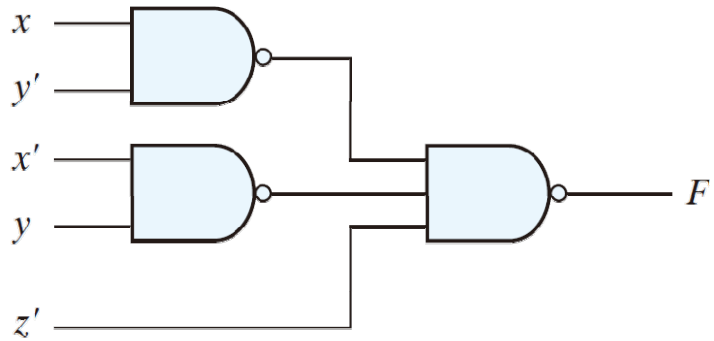
$$F(x,y,z) = \sum(1,2,3,4,5,7) \quad \Rightarrow \quad F(x,y,z) = xy' + x'y + z$$



(a)

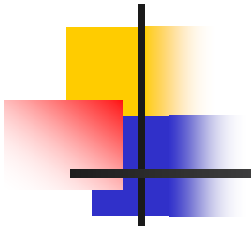


(b)



(c)

FIGURE 3.19
Solution to Example 3.9



- The procedure
  - simplified in the form of sum of products
  - a NAND gate for each product term; the inputs to each NAND gate are the literals of the term
  - a single NAND gate for the second sum term
  - A single literal requires an inverter in the first level

Multilevel NAND Circuits

- Boolean function implementation
  - AND-OR logic => NAND-NAND logic
    - AND => NAND + inverter
    - OR: inverter + OR = NAND

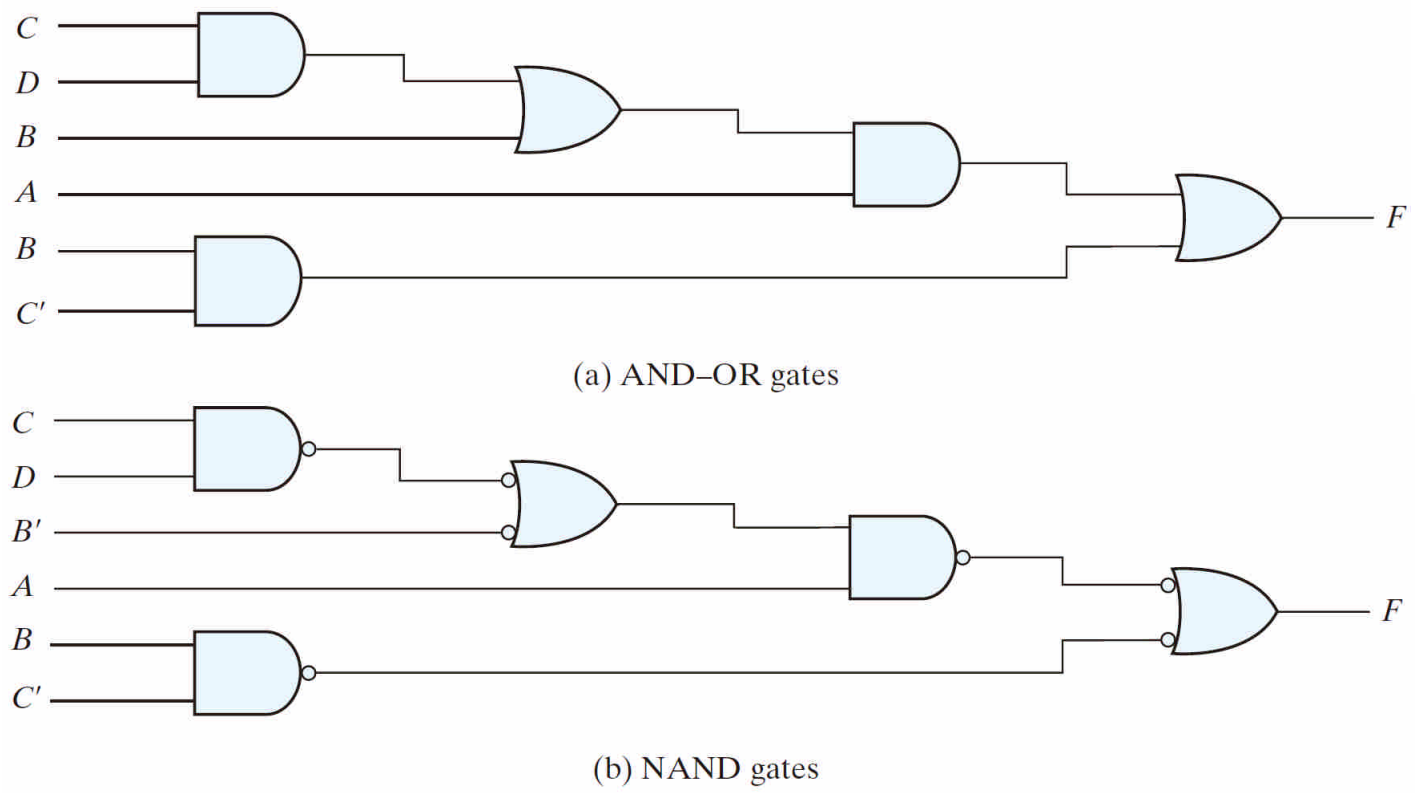
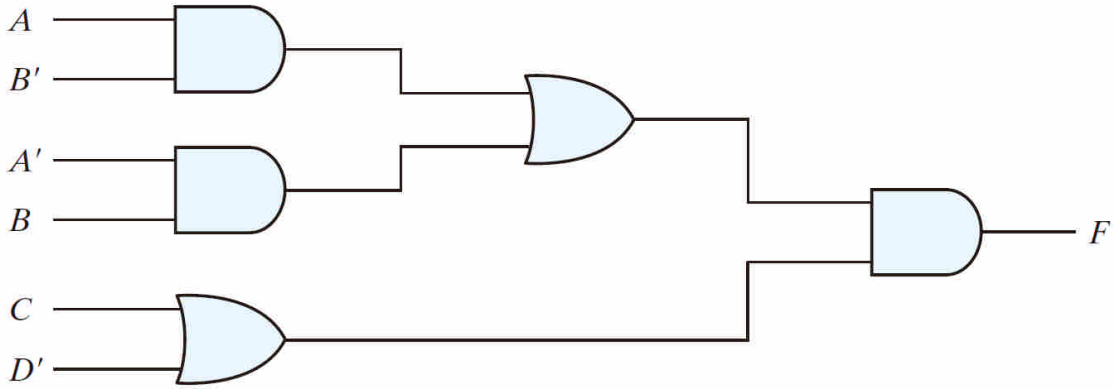
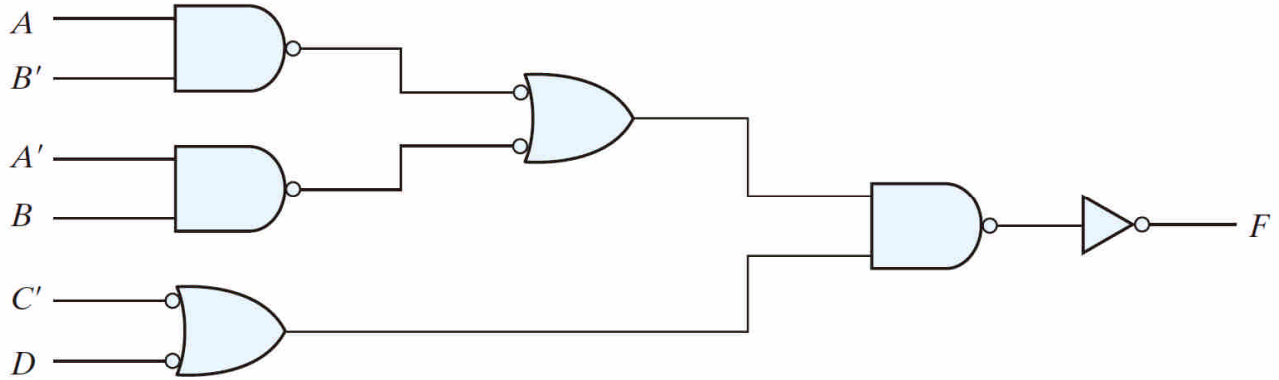


FIGURE 3.20
Implementing $F = A(CD + B) + BC'$

NAND Implementation



(a) AND–OR gates



(b) NAND gates

FIGURE 3.21
Implementing $F = (AB' + A'B)(C + D')$

NOR Implementation

- NOR function is the dual of NAND function
- The NOR gate is also universal

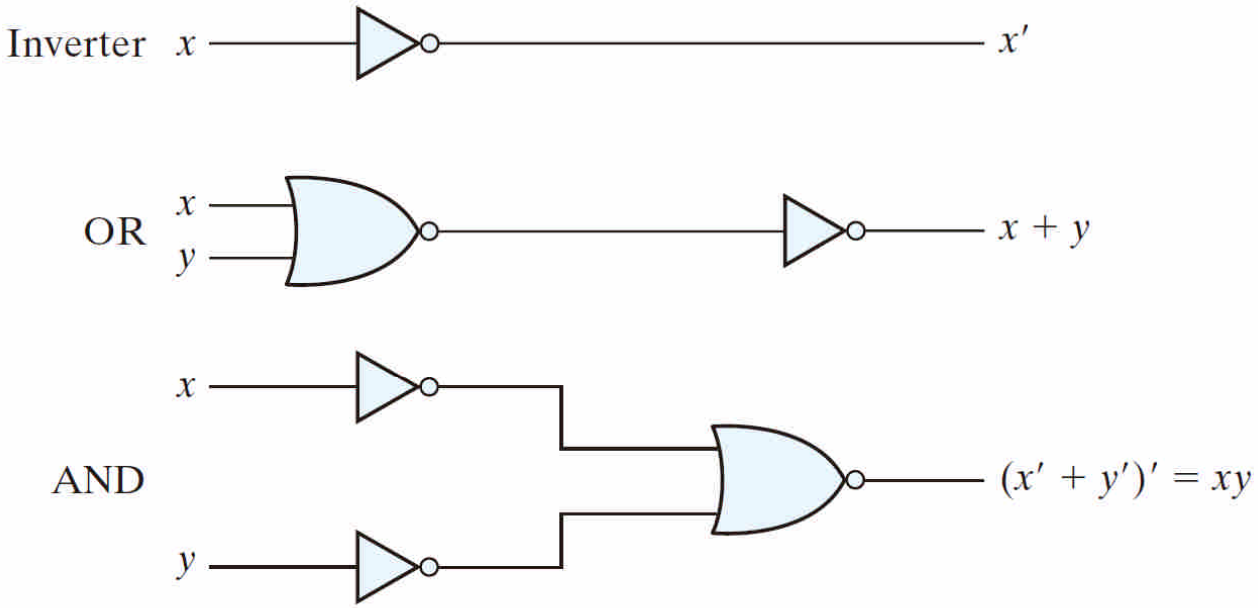
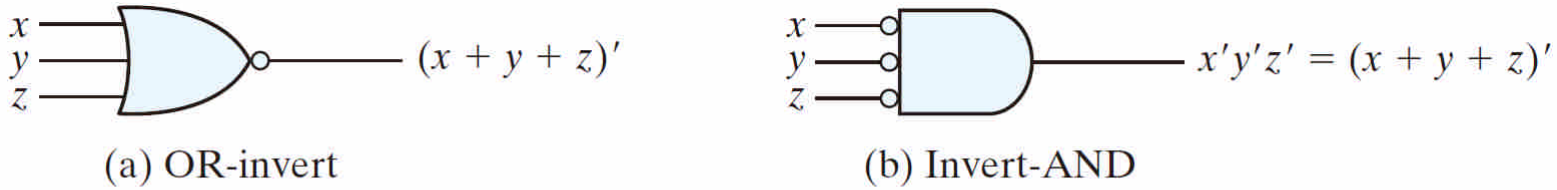


FIGURE 3.22
Logic operations with NOR gates

Two graphic symbols for a NOR gate



(a) OR-invert (b) Invert-AND

FIGURE 3.23
Two graphic symbols for the NOR gate

Example: F = (A + B)(C + D)E

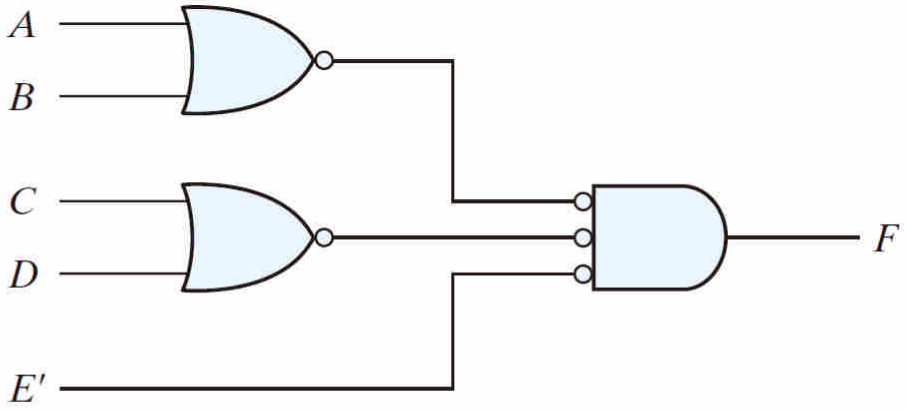


FIGURE 3.24
Implementing $F = (A + B)(C + D)E$

Example: F = (AB′ +A′B)(C + D′)

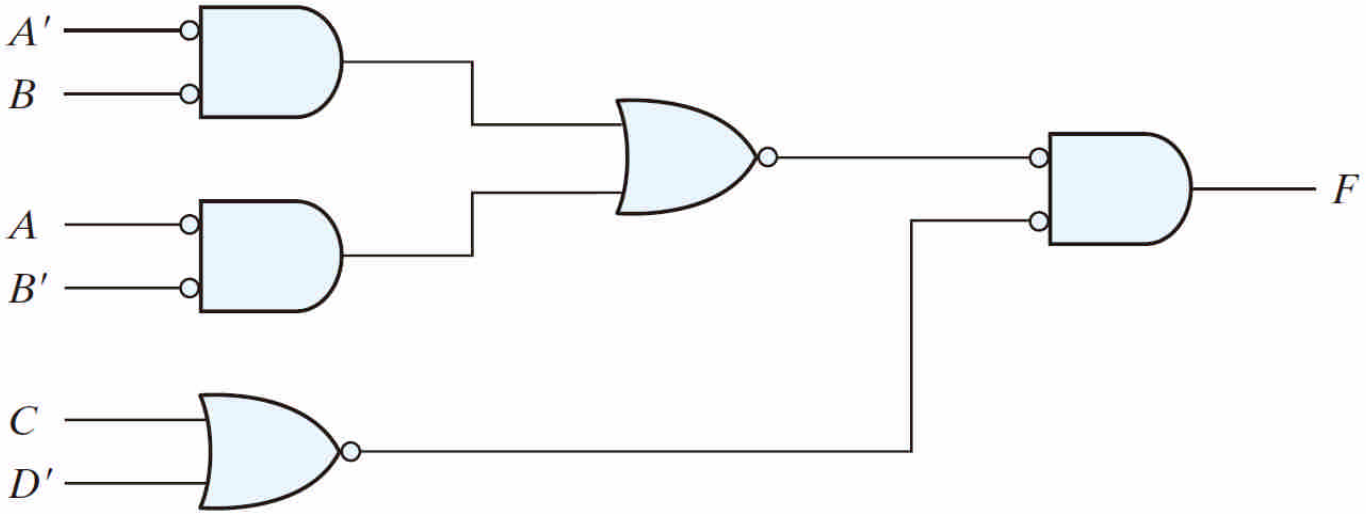


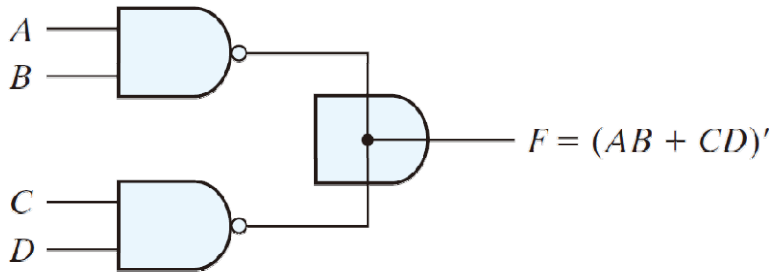
FIGURE 3.25
Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

3-7 Other Two-level Implementations

- Wired logic
  - a wire connection between the outputs of two gates
  - open-collector TTL NAND gates: wired-AND logic
  - the NOR output of ECL gates: wired-OR logic

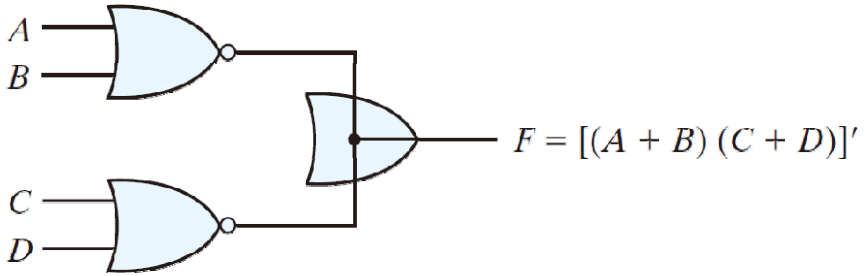
$$F = (AB)' \cdot (CD)' = (AB + CD)' = (A' + B')(C' + D')$$ AND-OR-INVERT function

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



(a) Wired-AND in open-collector TTL NAND gates.

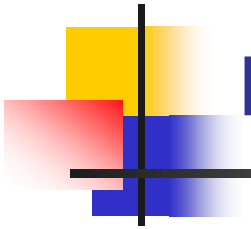
(AND–OR–INVERT)



(b) Wired-OR in ECL gates

(OR–AND–INVERT)

FIGURE 3.26
Wired logic
(a) Wired-AND logic with two NAND gates
(b) Wired-OR in emitter-coupled logic (ECL) gates



Nondegenerate Forms

- 16 possible combinations of two-level forms
  - eight of them: degenerate forms = a single operation
  - The eight nondegenerate forms
    - AND-OR, OR-AND, NAND-NAND, NOR-NOR, NOR-OR, NAND-AND, OR-AND, AND-OR
    - AND-OR and NAND-NAND = sum of products
    - OR-AND and NOR-NOR = product of sums
    - NOR-OR, NAND-AND, OR-AND, AND-OR = ?

AND-OR-Invert Implementation

- AND-OR-INVERT (AOI) Implementation
  - NAND-AND = AND-NOR = AOI
  - F = (AB+CD+E)'
  - F' = AB+CD+E(sum of products)

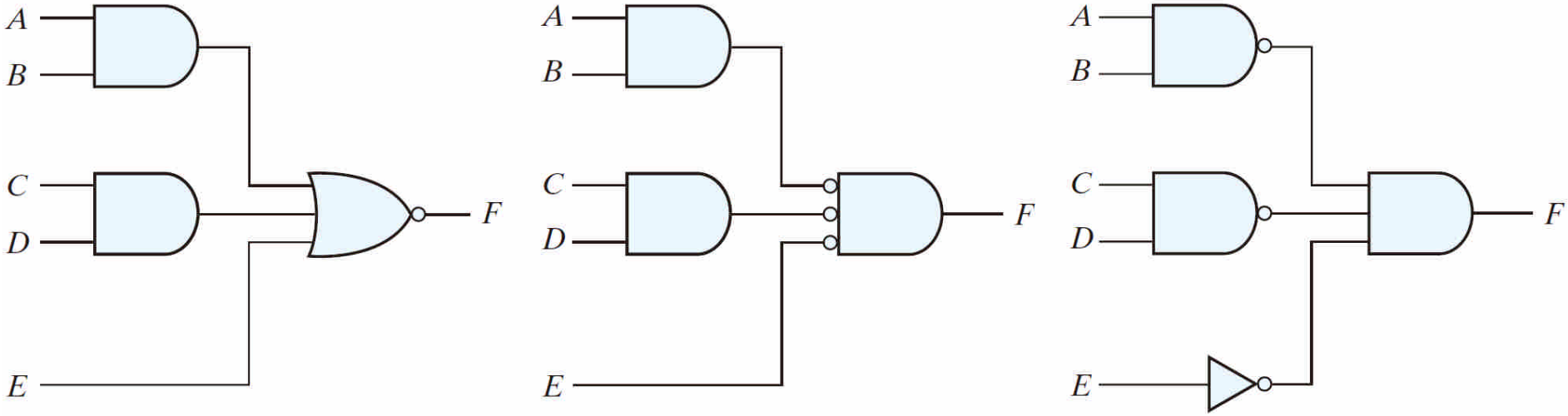


FIGURE 3.27
AND–OR–INVERT circuits, $F = (AB + CD + E)'$

- simplify F' in sum of products

- OR-AND-INVERT (OAI) Implementation
  - OR-NAND = NOR-OR = OAI
  - F = ((A+B)(C+D)E)'
  - F' = (A+B)(C+D)E (product of sums)

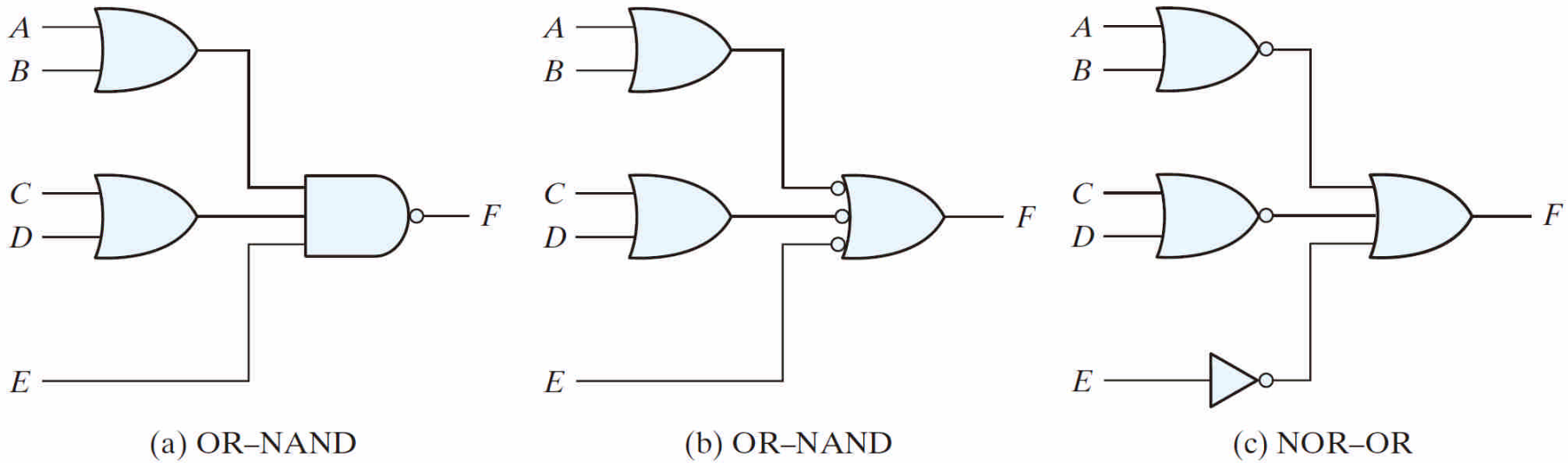


FIGURE 3.28
OR–AND–INVERT circuits, $F = [(A + B)(C + D)E]'$

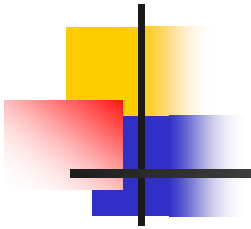
  - simplified F' in products of sum

Tabular Summary and Examples

Table 3.2
Implementation with Other Two-Level Forms

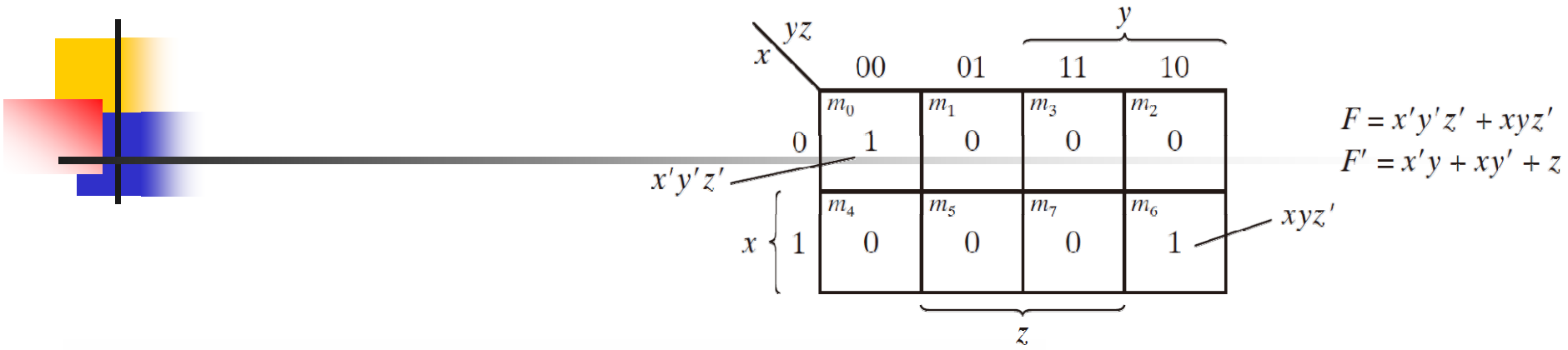
Equivalent Nondegenerate Form		Implements the Function	Simplify F' into	To Get an Output of
(a)	(b)*			
AND–NOR	NAND–AND	AND–OR–INVERT	Sum-of-products form by combining 0's in the map.	F
OR–NAND	NOR–OR	OR–AND–INVERT	Product-of-sums form by combining 1's in the map and then complementing.	F

*Form (b) requires an inverter for a single literal term.

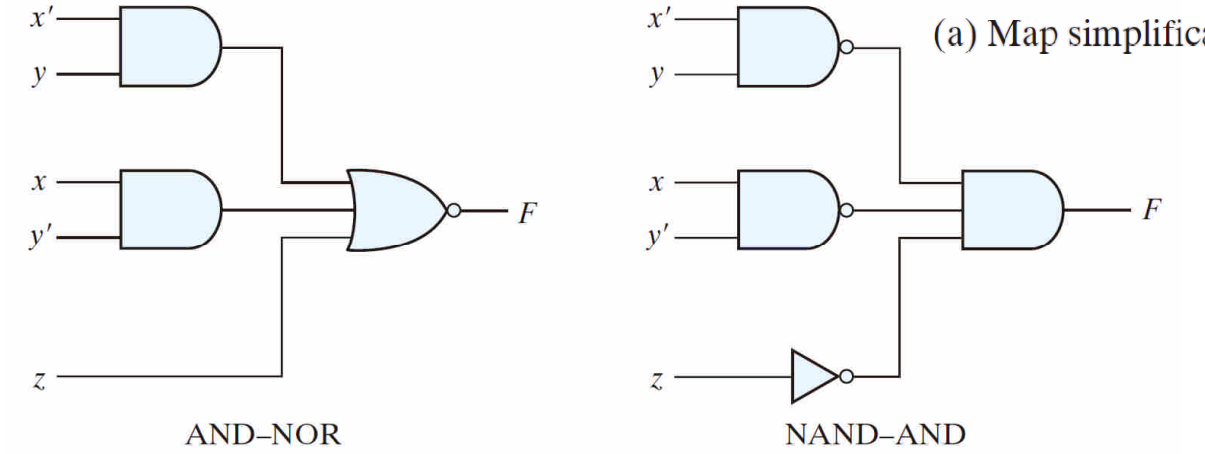


- Example 3-10
  - F' = x'y+xy'+z (F': sum of products)
  - F = (x'y+xy'+z)' (F: AOI implementation)

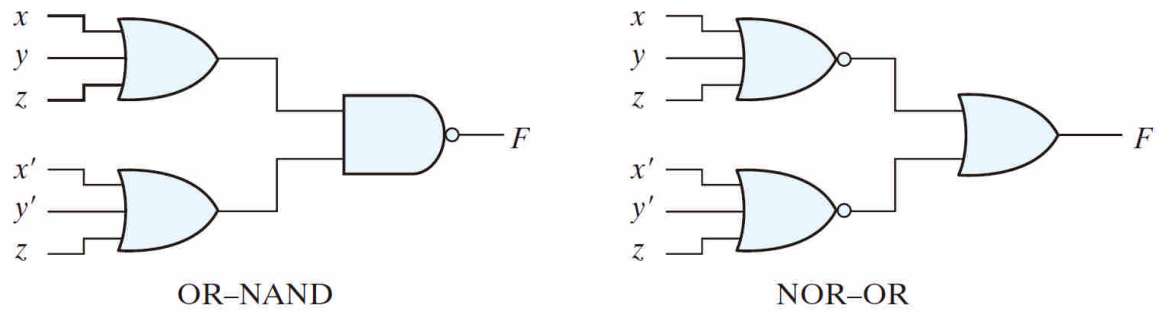
  - F = x'y'z' + xyz' (F: sum of products)
  - F' = (x+y+z)(x'+y'+z) (F': product of sums)
  - F = ((x+y+z)(x'+y'+z))' (F: OAI)



(a) Map simplification in sum of products



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



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

FIGURE 3.29
Other two-level implementations