

# Digital Logic Design

## Gate-Level Minimization

# 3-1 Introduction

---

- **Gate-level minimization** refers to 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
    - 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

# Review of Boolean Function

---

## ■ 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
  - $x' = \text{row } 0$ ;  $x = \text{row } 1$
  - $y' = \text{column } 0$ ;  $y = \text{column } 1$
  - A truth table in square diagram
  - Fig. 3.2(a):  $xy = m_3$
  - Fig. 3.2(b):  $x+y = x'y+xy' + xy = m_1+m_2+m_3$

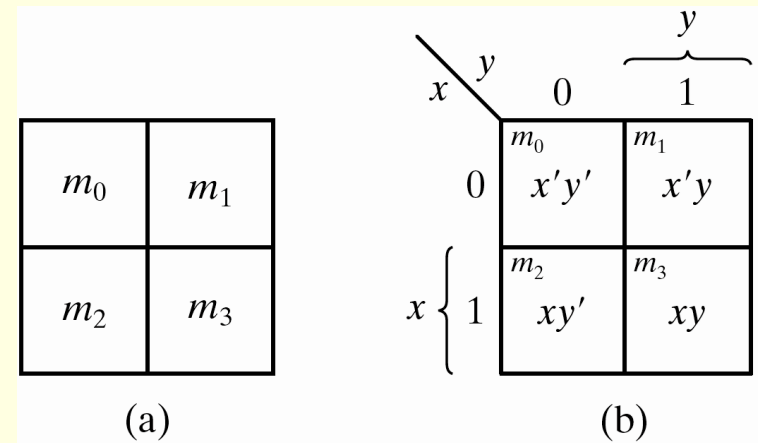


Figure 3.1 Two-variable Map

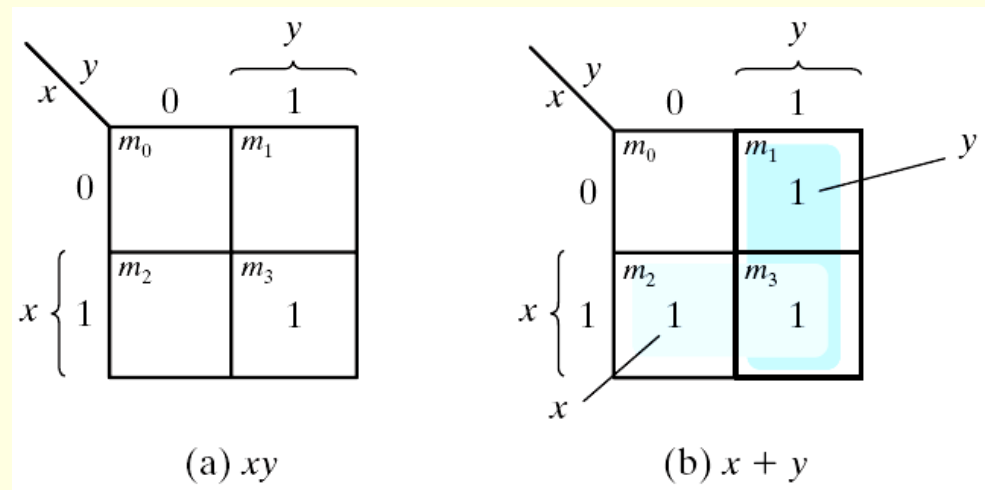


Figure 3.2 Representation of functions in the map

# A Three-variable Map

- A three-variable map
  - Eight minterms
  - The Gray code sequence
  - Any two adjacent squares in the map differ by only one 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)

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

(b)

Figure 3.3 Three-variable Map

# A Three-variable Map

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

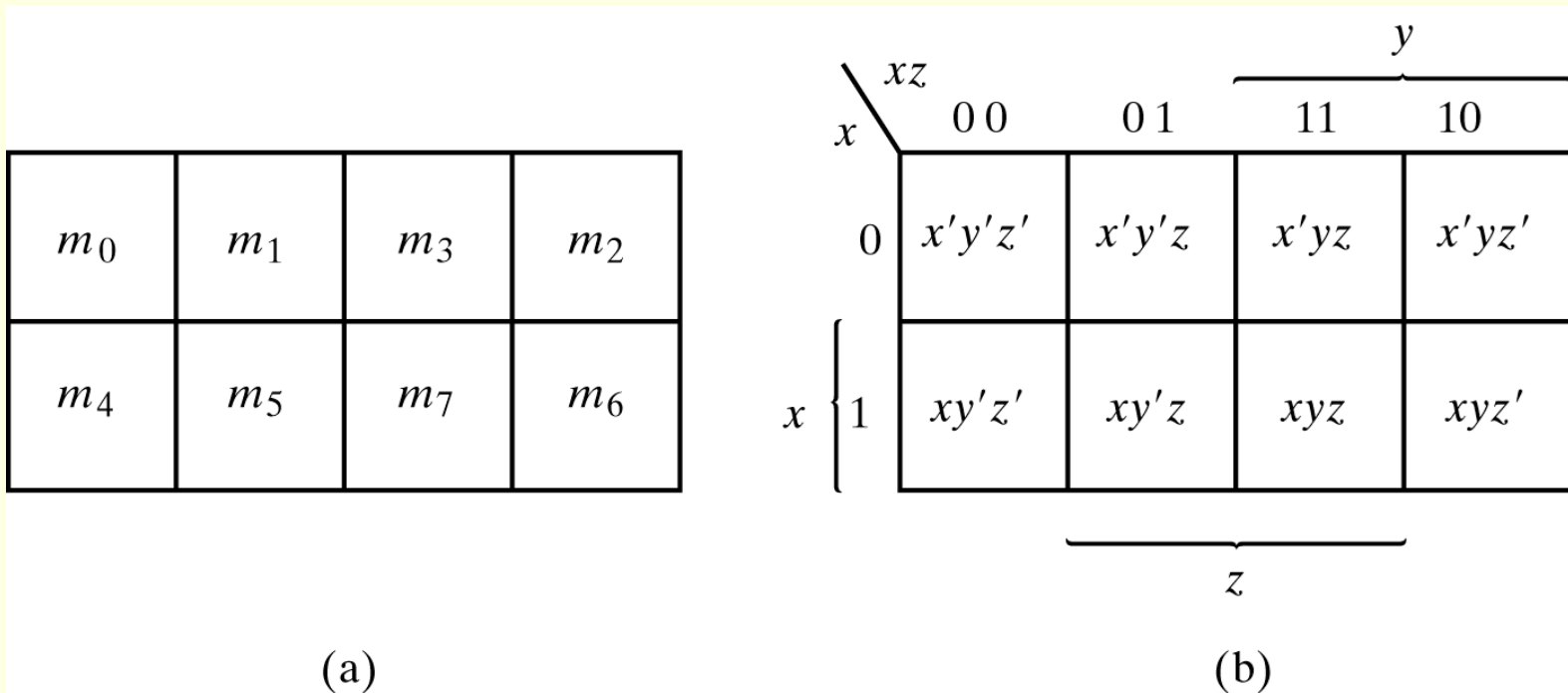


Fig. 3-3 Three-variable Map

# Example 3.1

- Example 3.1: simplify the Boolean function  $F(x, y, z) = \Sigma(2, 3, 4, 5)$ 
  - $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

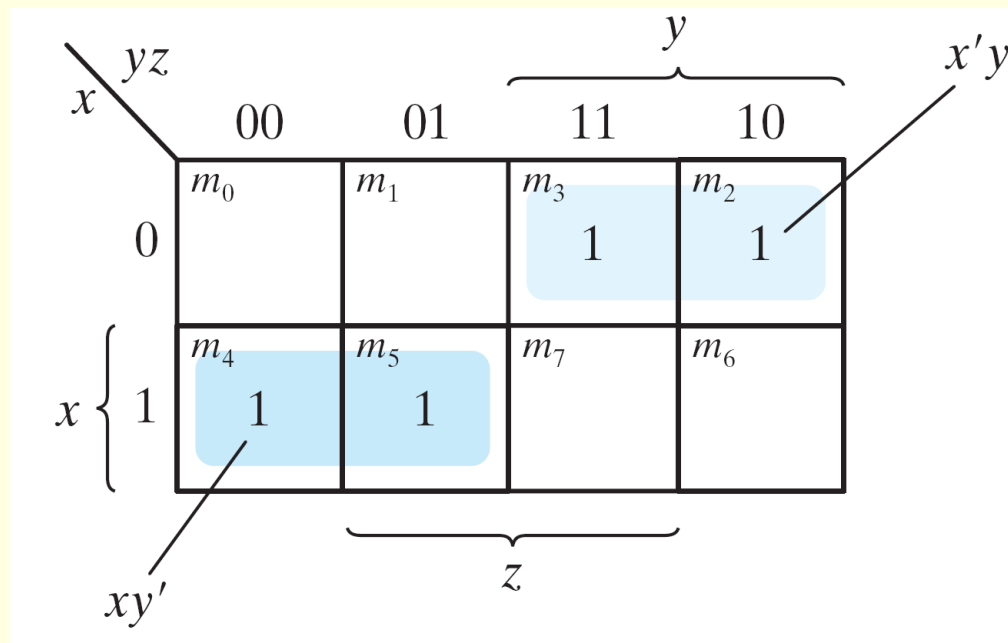


Figure 3.4 Map for Example 3.1,  $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$



# Example 3.2

- Example 3.2: simplify  $F(x, y, z) = \Sigma(3, 4, 6, 7)$ 
  - $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

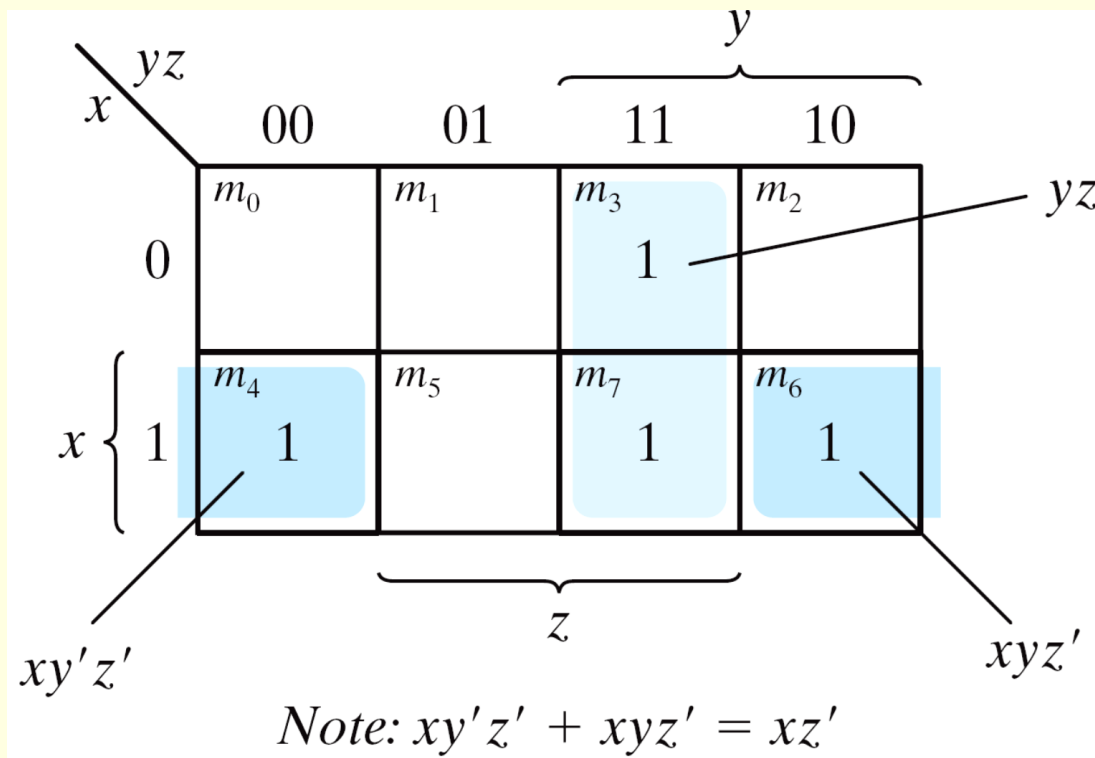


Figure 3.5 Map for Example 3-2;  $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

# Four adjacent Squares

## ■ Consider four adjacent squares

□ 2, 4, and 8 squares

□  $m_0 + m_2 + m_4 + m_6 = x'y'z' + x'yz' + xy'z' + xyz' = x'z'(y' + y) + xz'(y' + y) = x'z' + xz' = z'$

□  $m_1 + m_3 + m_5 + m_7 = x'y'z + x'yz + xy'z + xyz = x'z(y' + y) + xz(y' + y) = x'z + xz = z$

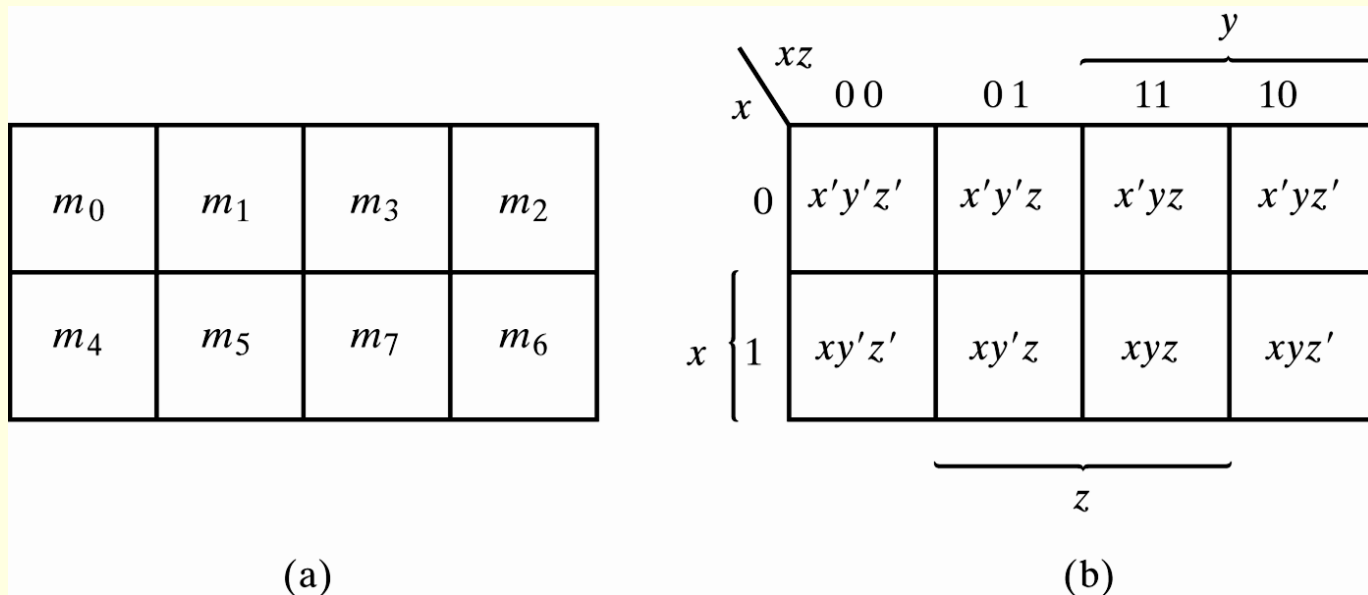


Figure 3.3 Three-variable Map

# Example 3.3

- Example 3.3: simplify  $F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$   
 $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

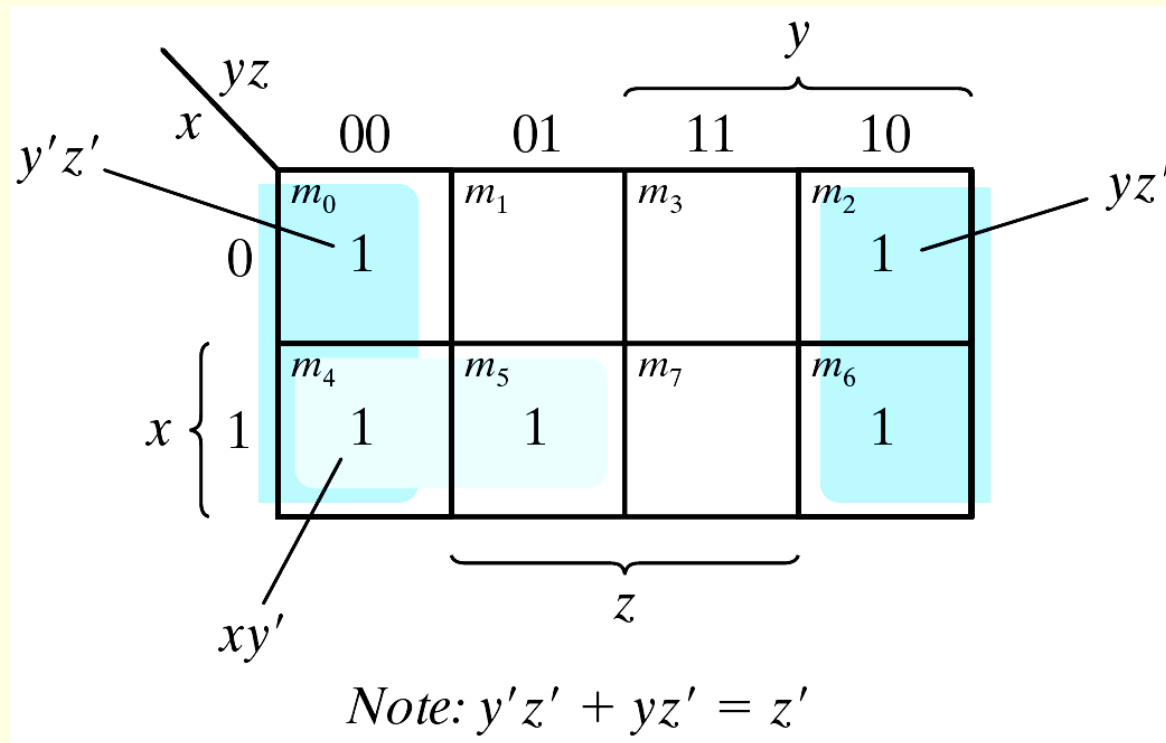


Figure 3.6 Map for Example 3-3,  $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

# Example 3.4

■ Example 3.4: let  $F = A'C + A'B + AB'C + BC$

a) Express it in sum of minterms.

b) Find the minimal sum of products expression.

Ans:

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

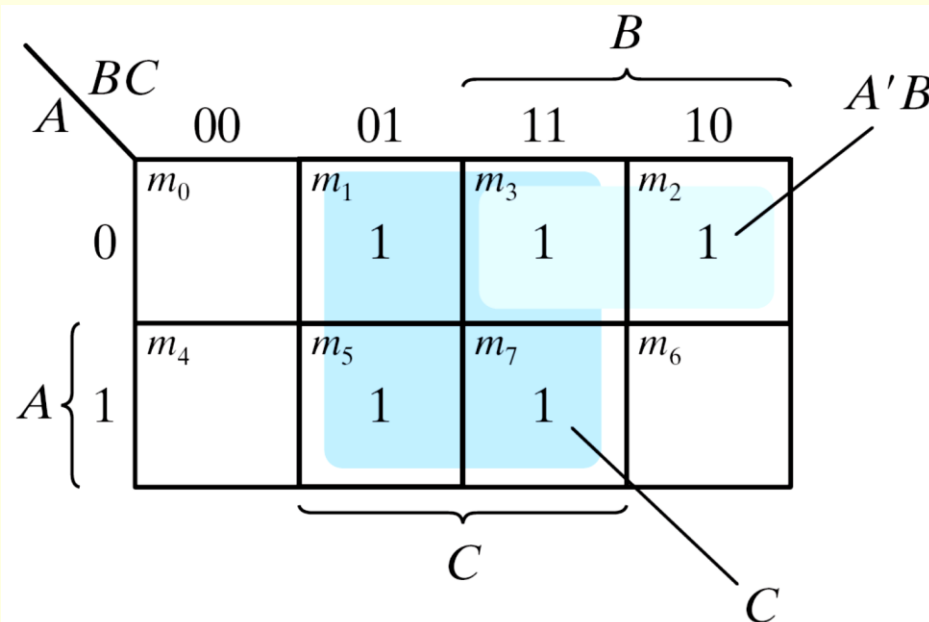


Figure 3.7 Map for Example 3.4,  $A'C + A'B + AB'C + BC = C + A'B$

# 3.3 Four-Variable Map

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

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

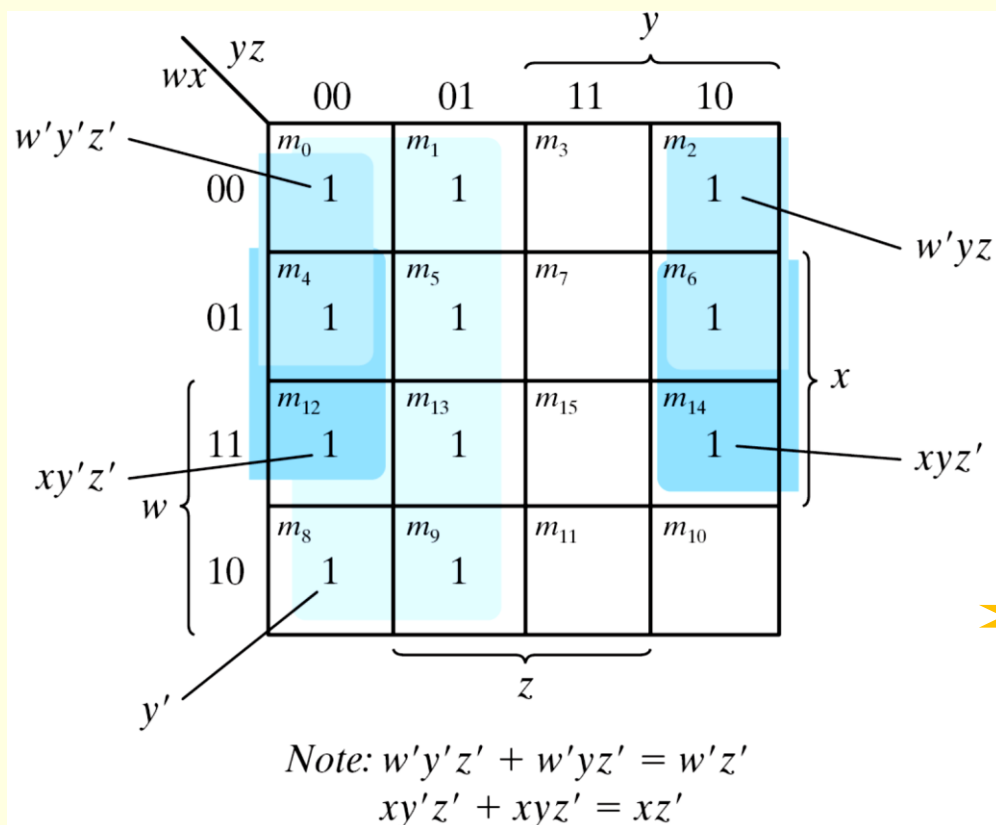
(a)

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

(b)

# Example 3.5

- Example 3.5: simplify  $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

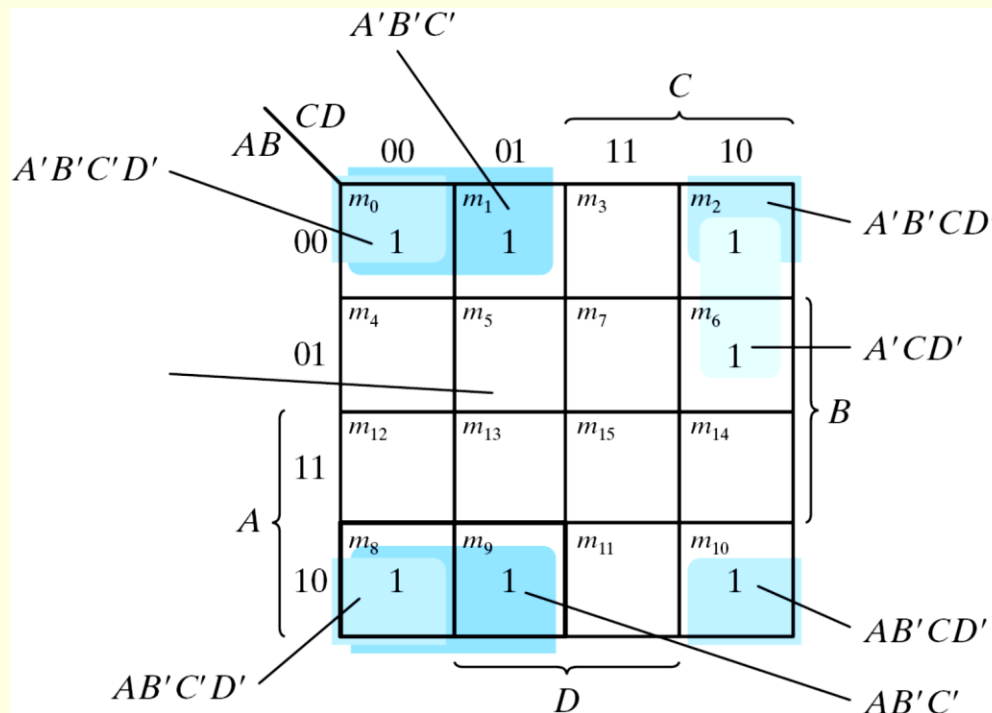


$$F = y' + w'z' + xz'$$

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

- Example 3-6: simplify  $F = A'B'C' + B'CD' + A'B'CD' + AB'C'$



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

Figure 3.9 Map for Example 3-6;  $A'B'C' + B'CD' + A'B'CD' + AB'C' = B'D' + B'C' + A'CD'$

# Prime Implicants

---

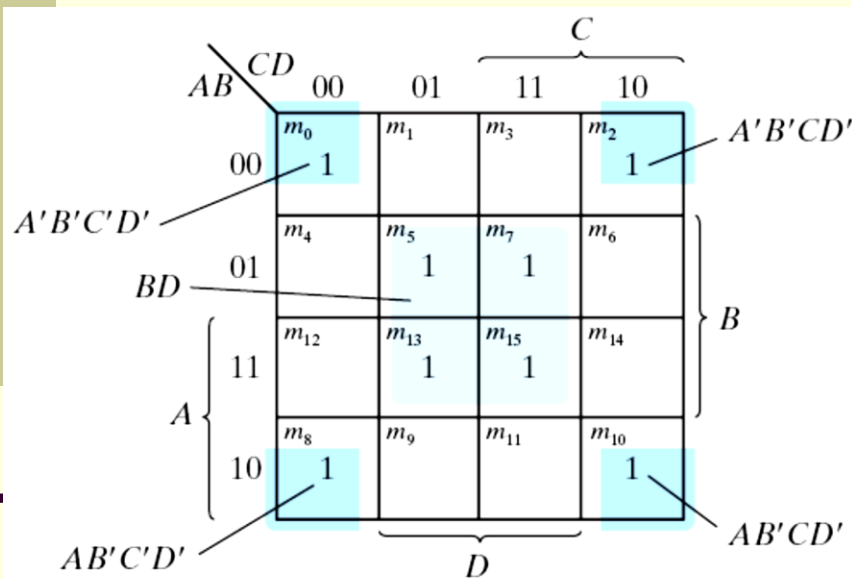
## ■ Prime Implicants

- ❑ 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 P.I.: a minterm is covered by only one prime implicant.
- ❑ The essential P.I. must be included.



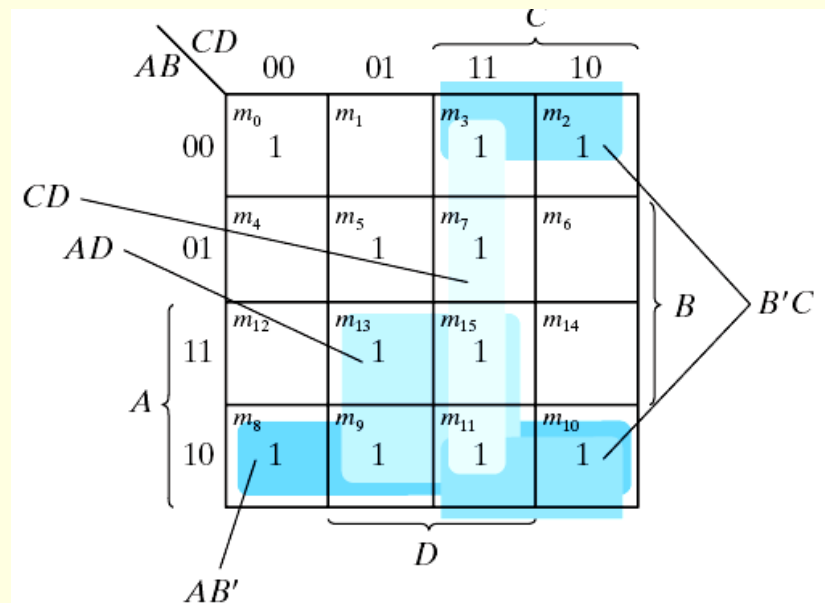
# Prime Implicants

- Consider  $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$ 
  - 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:  $A'B'C'D' + A'B'CD' = A'B'D'$   
 $AB'C'D' + AB'CD' = AB'D'$   
 $A'B'D' + AB'D' = B'D'$

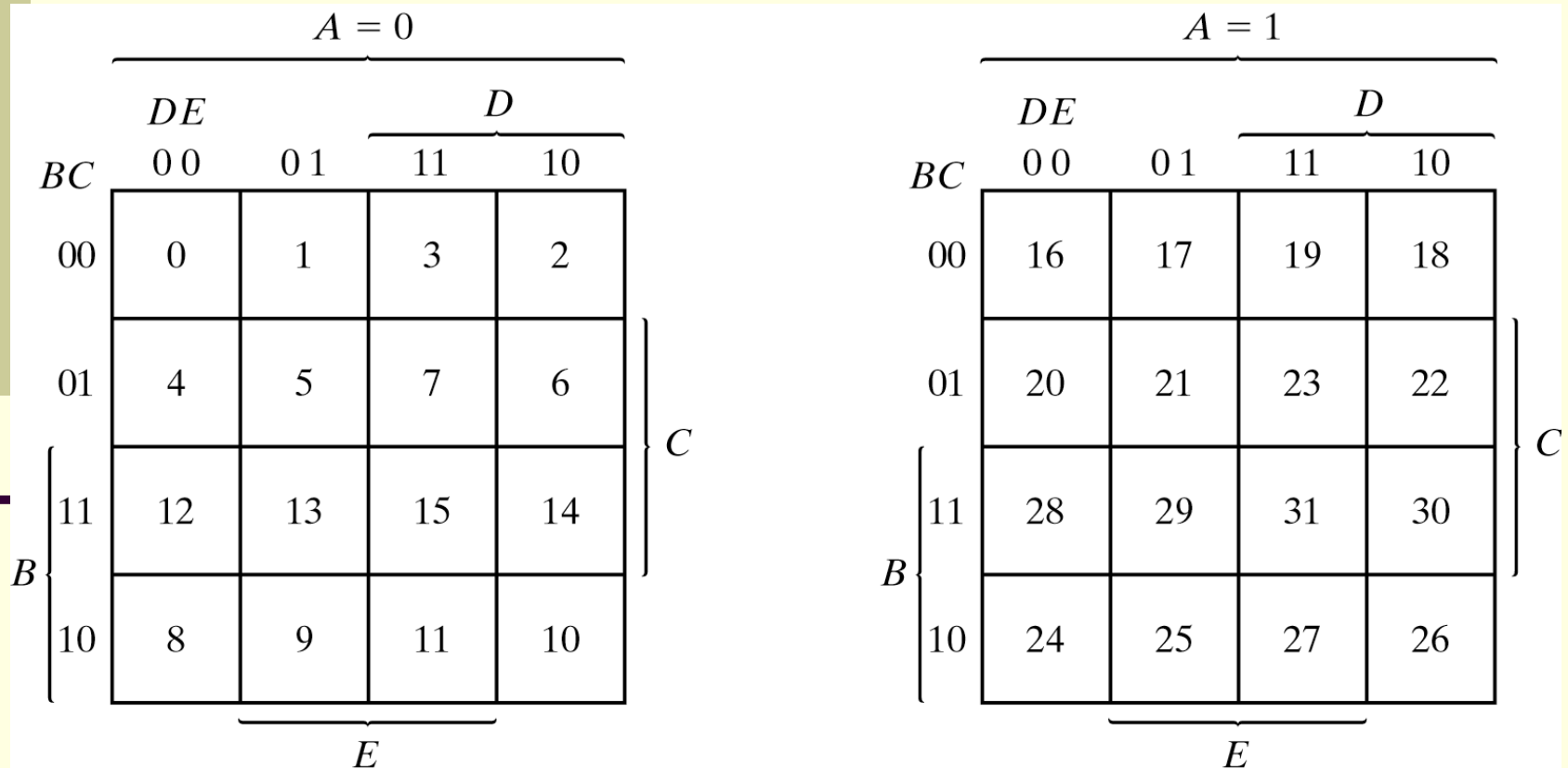
(a) Essential prime implicants  
 $BD$  and  $B'D'$



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

# 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).



- 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*

<b><math>K</math></b>	<b>Number of Adjacent Squares <math>2^k</math></b>	<b>Number of Literals in a Term in an <math>n</math>-variable Map</b>			
		<b><math>n = 2</math></b>	<b><math>n = 3</math></b>	<b><math>n = 4</math></b>	<b><math>n = 5</math></b>
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0

# Example 3.7

- Example 3.7: simplify  $F = \Sigma(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$

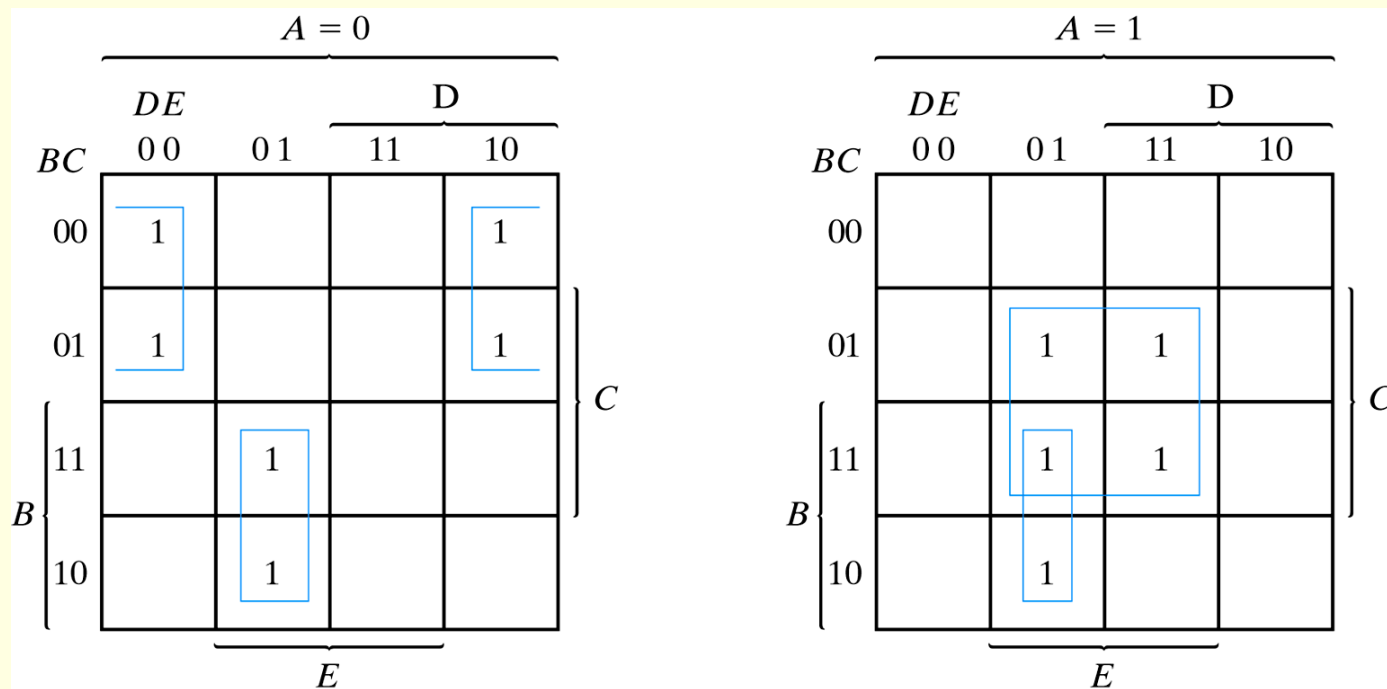


Fig. 3-13 Map for Example 3-7;  $F = A'B'E' + BD'E + ACE$



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

# 3-5 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  $\rightarrow F$ : product of sums

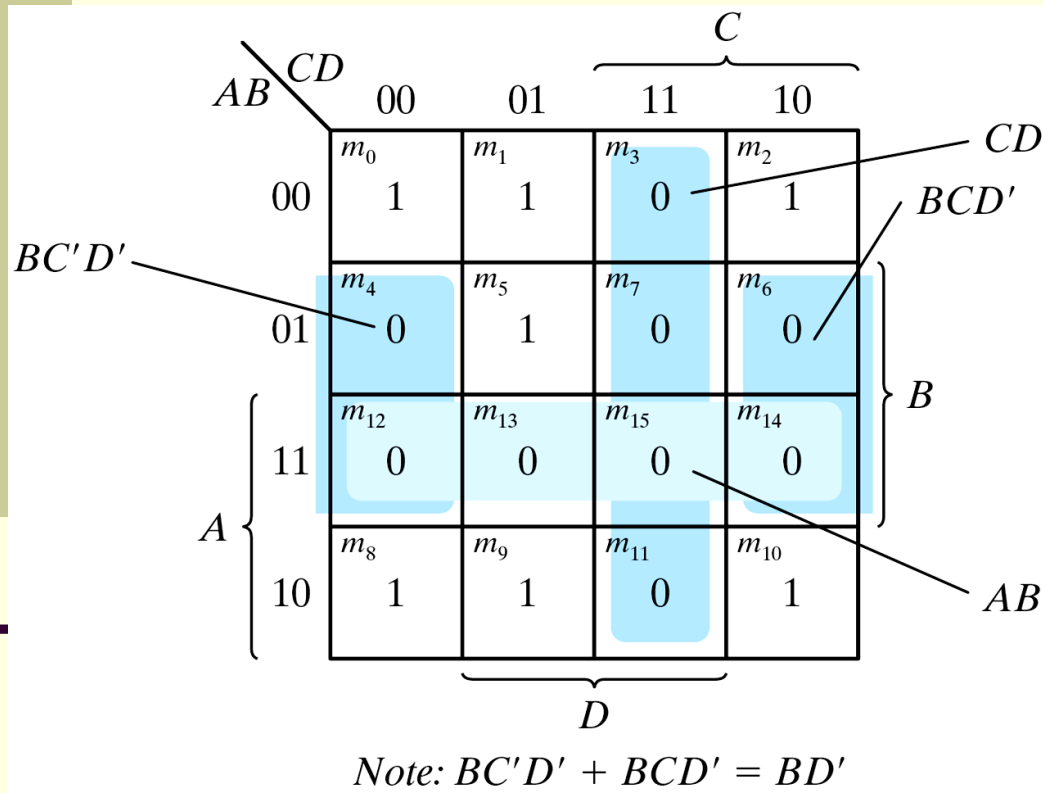
## Approach #2: duality

- Combinations of maxterms (it was minterms)
- $M_0 M_1 = (A+B+C+D)(A+B+C+D') = (A+B+C) + (DD') = A+B+C$

AB \ CD	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 3.8

- Example 3.8: simplify  $F = \Sigma(0, 1, 2, 5, 8, 9, 10)$  into (a) sum-of-products form, and (b) product-of-sums form:



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

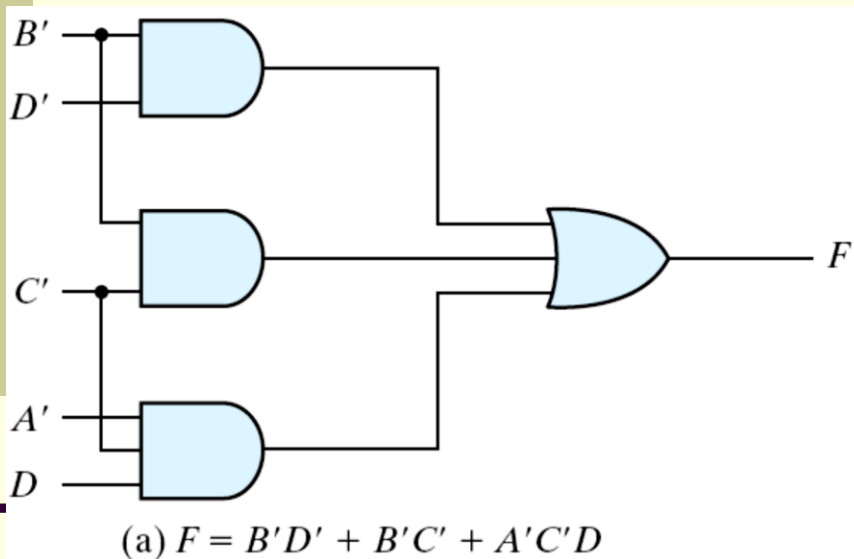
b)  $F' = AB + CD + BD'$

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

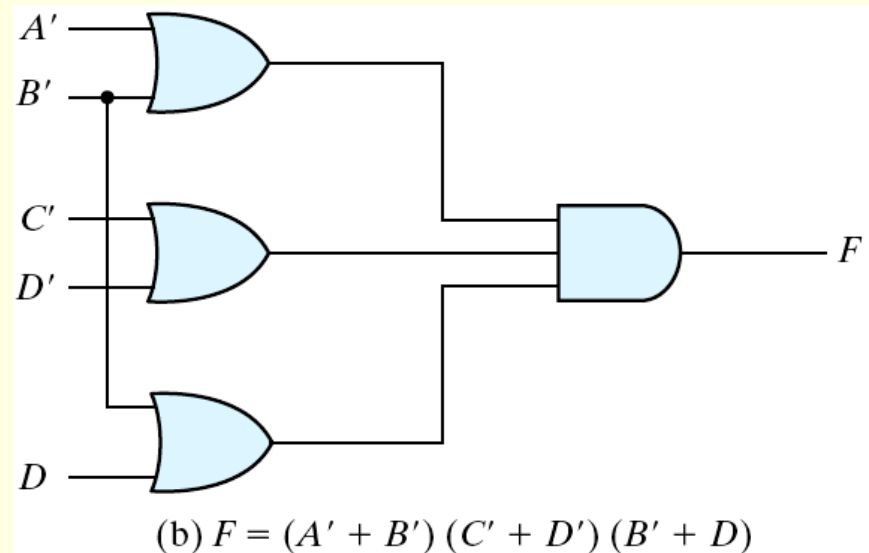
Figure 3.14 Map for Example 3.8,  $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D$

# Example 3.8 (cont.)

- Gate implementation of the function of Example 3.8



Sum-of products form



Product-of sums form

Figure 3.15 Gate Implementation of the Function of Example 3.8

# Sum-of-Minterm Procedure

- Consider the function defined in Table 3.2.
  - 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 3.2**

*Truth Table of Function F*

<b>x</b>	<b>y</b>	<b>z</b>	<b>F</b>
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



# Sum-of-Minterm Procedure

- Consider the function defined in Table 3.2.
  - Combine the 1's:

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

- Combine the 0's :

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

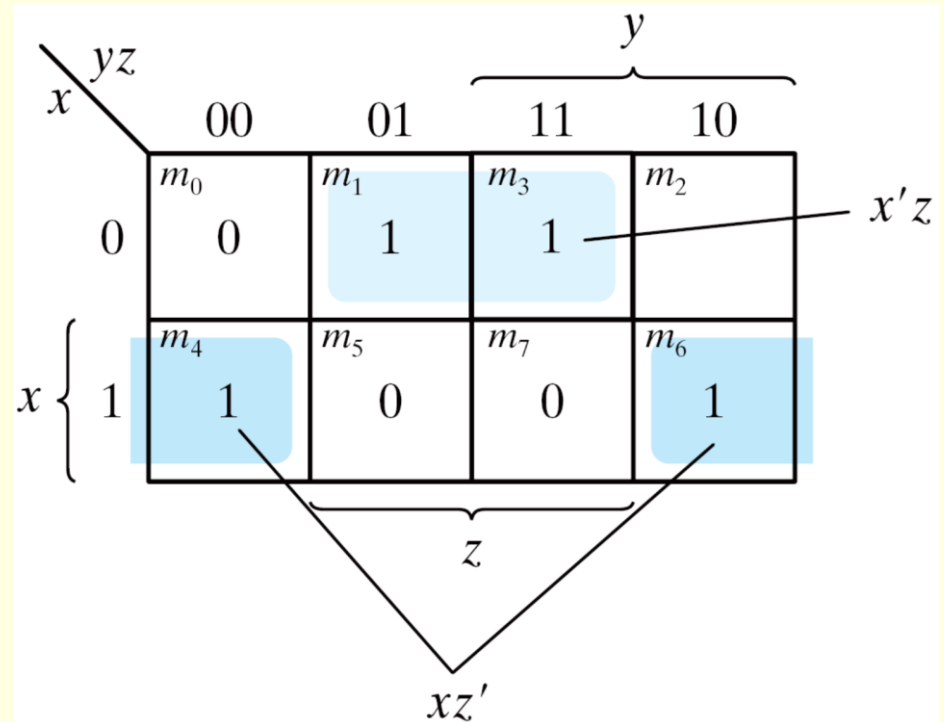


Figure 3.16 Map for the function of Table 3.2

## 3-6 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 3.9: simplify  $F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$  which has the don't-care conditions  $d(w, x, y, z) = \Sigma(0, 2, 5)$ .

# Example 3.9 (cont.)

- ❑  $F = yz + w'x'$ ;  $F = yz + w'z$
- ❑  $F = \Sigma(0, 1, 2, 3, 7, 11, 15)$ ;  $F = \Sigma(1, 3, 5, 7, 11, 15)$
- ❑ Either expression is acceptable

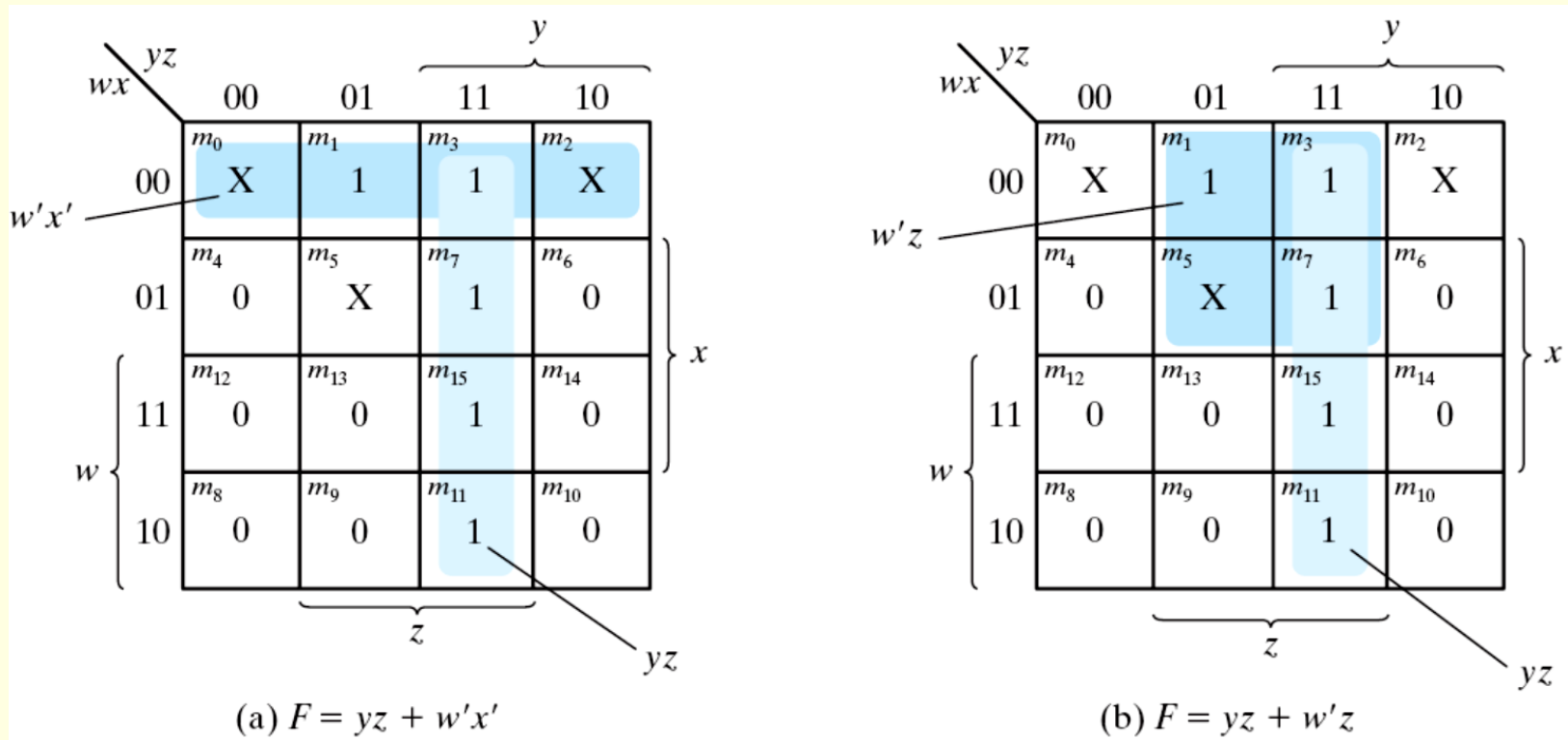


Figure 3.17 Example with don't-care Conditions

# 3-7 NAND and NOR Implementation

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

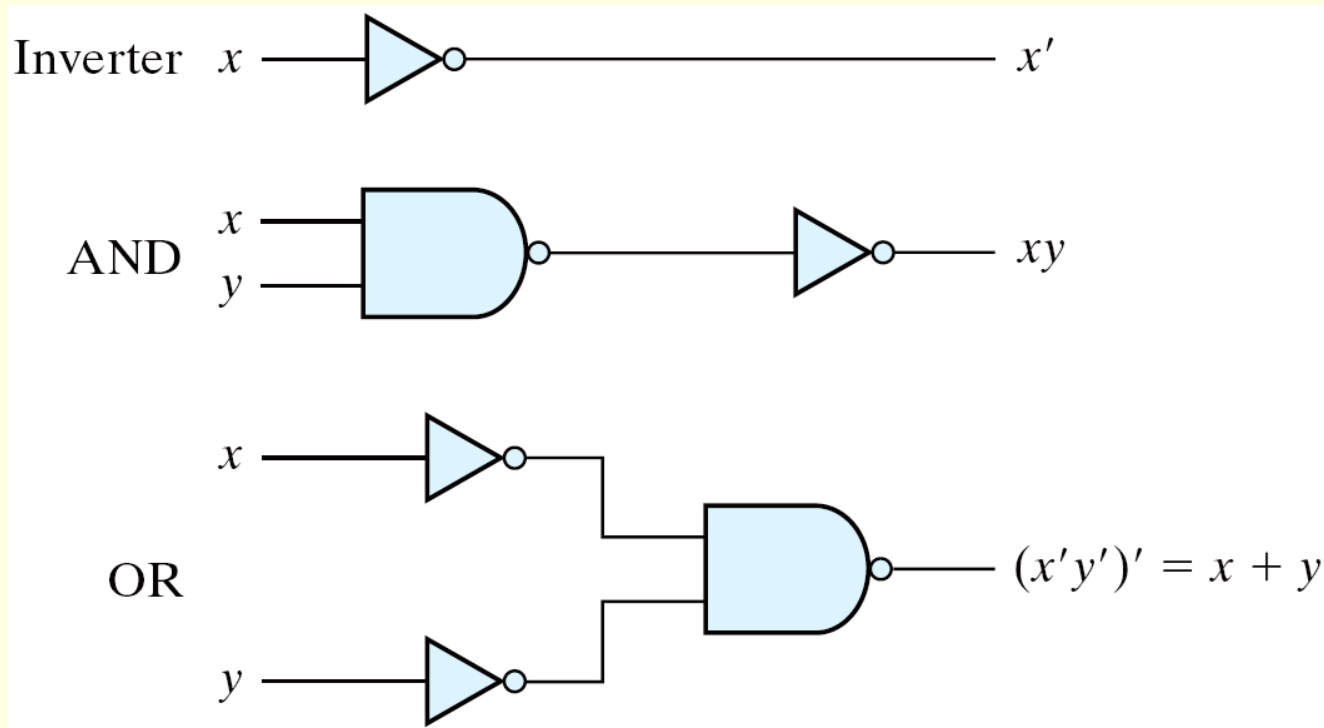


Figure 3.18 Logic Operations with NAND Gates

# NAND Gate

- Two graphic symbols for a NAND gate

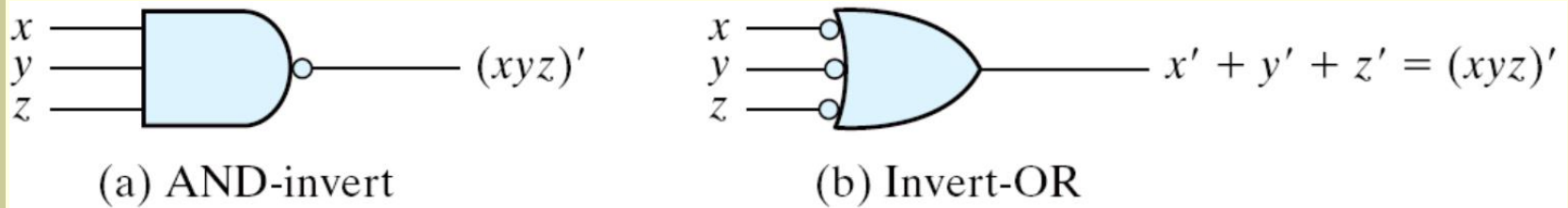


Figure 3.19 Two Graphic Symbols for NAND Gate

# Two-level Implementation

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

# Example 3.10

- Example 3-10: implement  $F(x, y, z) =$

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

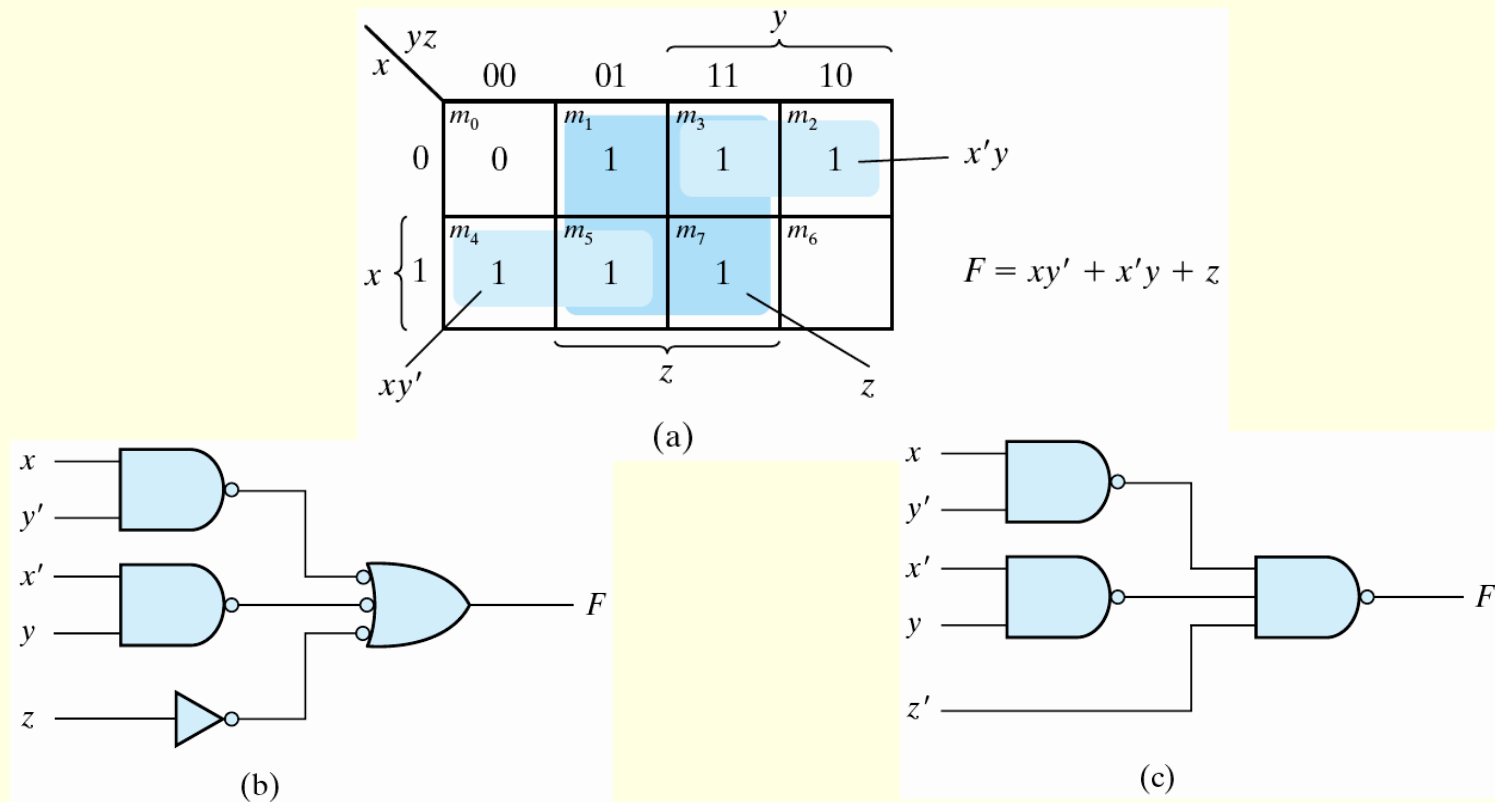


Figure 3.21 Solution to Example 3-10

# Procedure with Two Levels NAND

---

## ■ 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 (the first level);
- A single NAND gate for the second sum term (the second level);
- A term with a single literal requires an inverter in the first level.

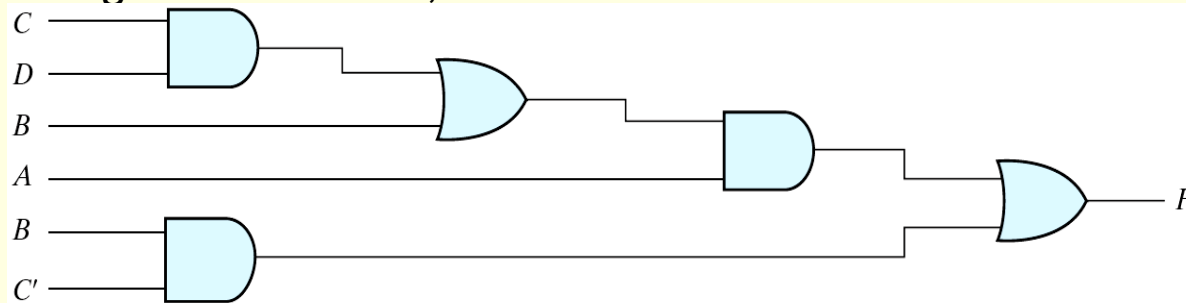


# Multilevel NAND Circuits

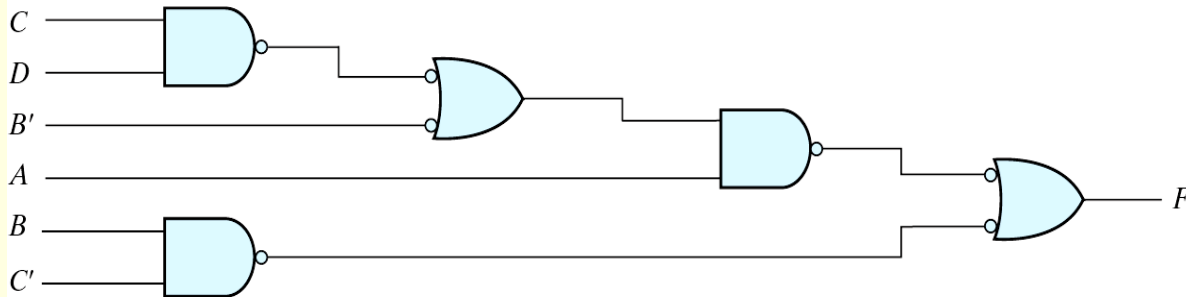
## ■ Boolean function implementation

### □ AND-OR logic → NAND-NAND logic

- AND → AND + inverter
- OR: inverter + OR = NAND
- For every bubble that is not compensated by another small circle along the same line, insert an inverter.

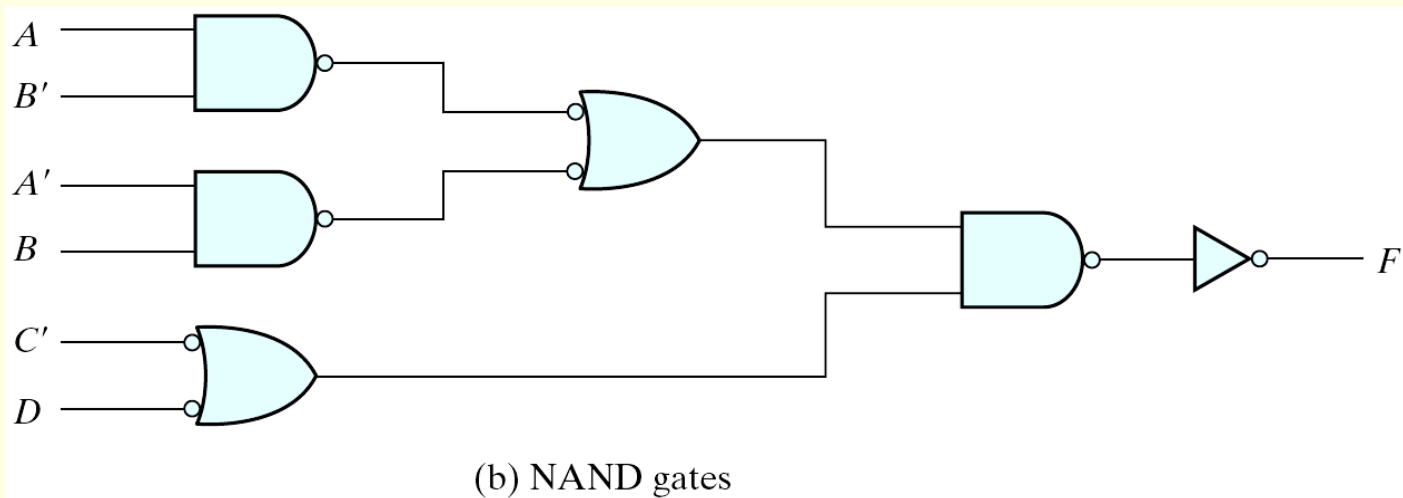
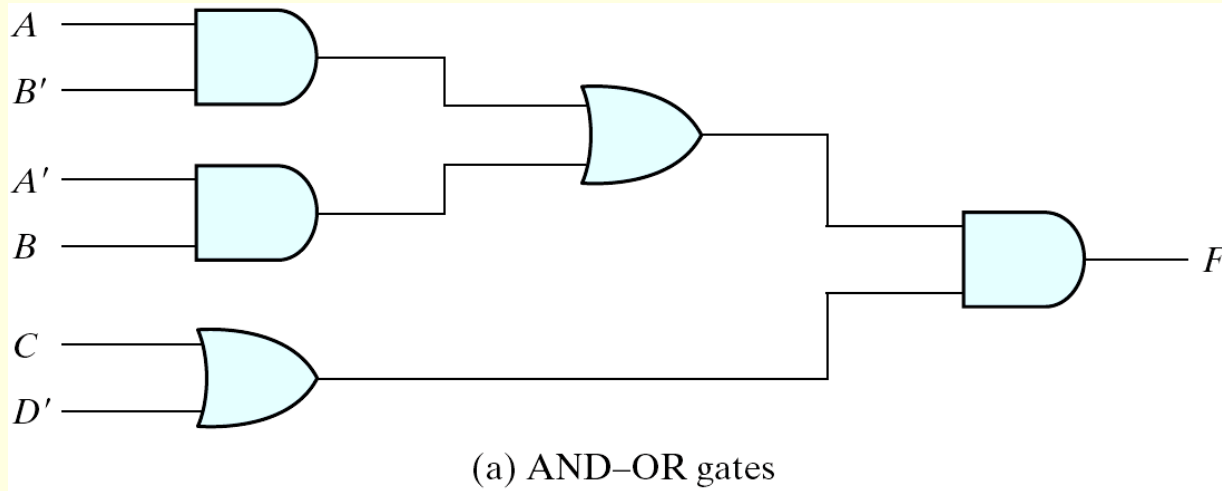


(a) AND-OR gates



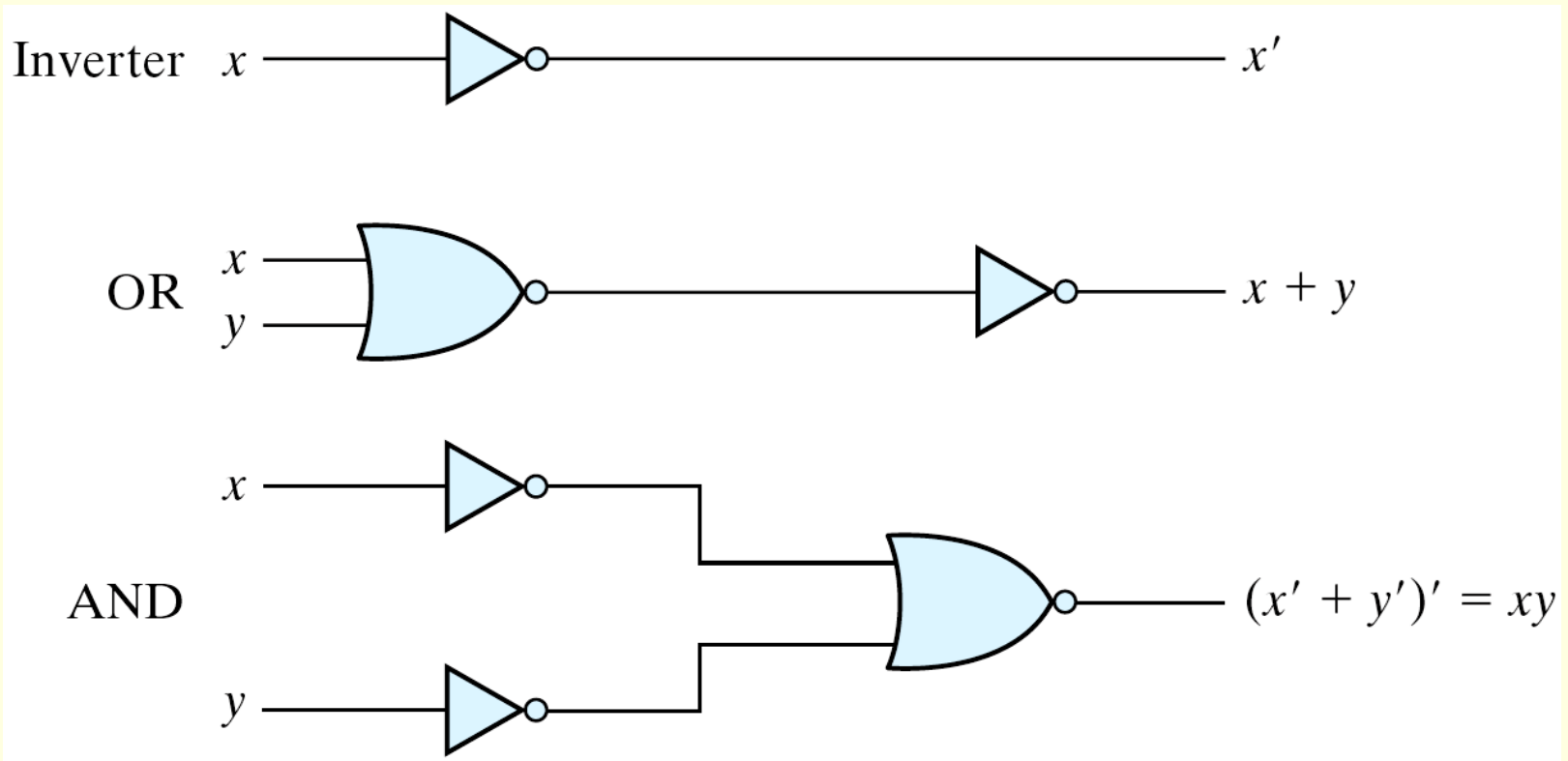
(b) NAND gates

# NAND Implementation



# NOR Implementation

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



# Two Graphic Symbols for a NOR Gate

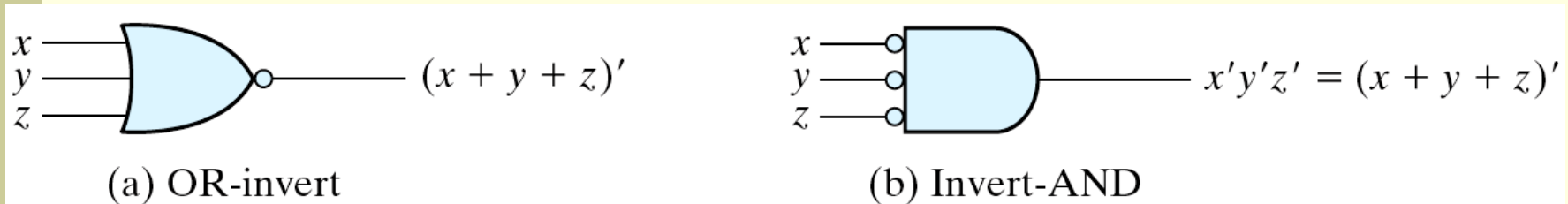
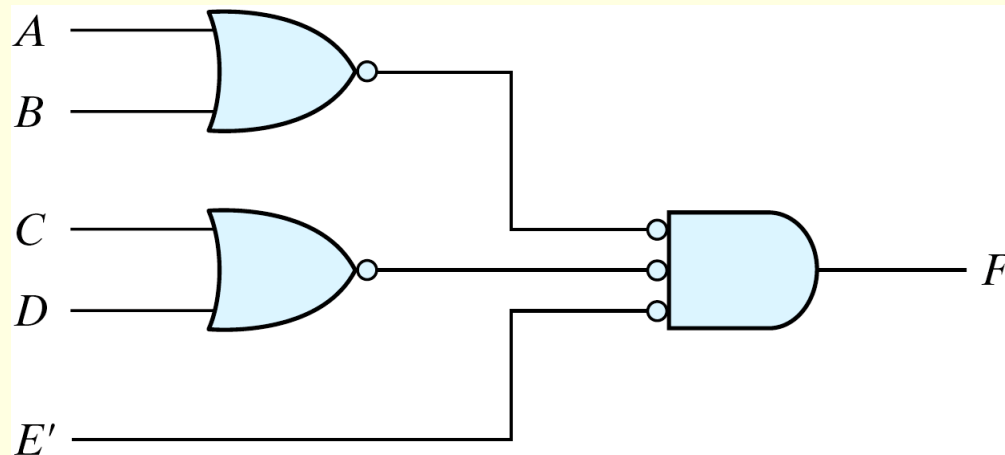


Figure 3.25 Two Graphic Symbols for NOR Gate

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



# Example

Example:  $F = (AB' + A'B)(C + D')$

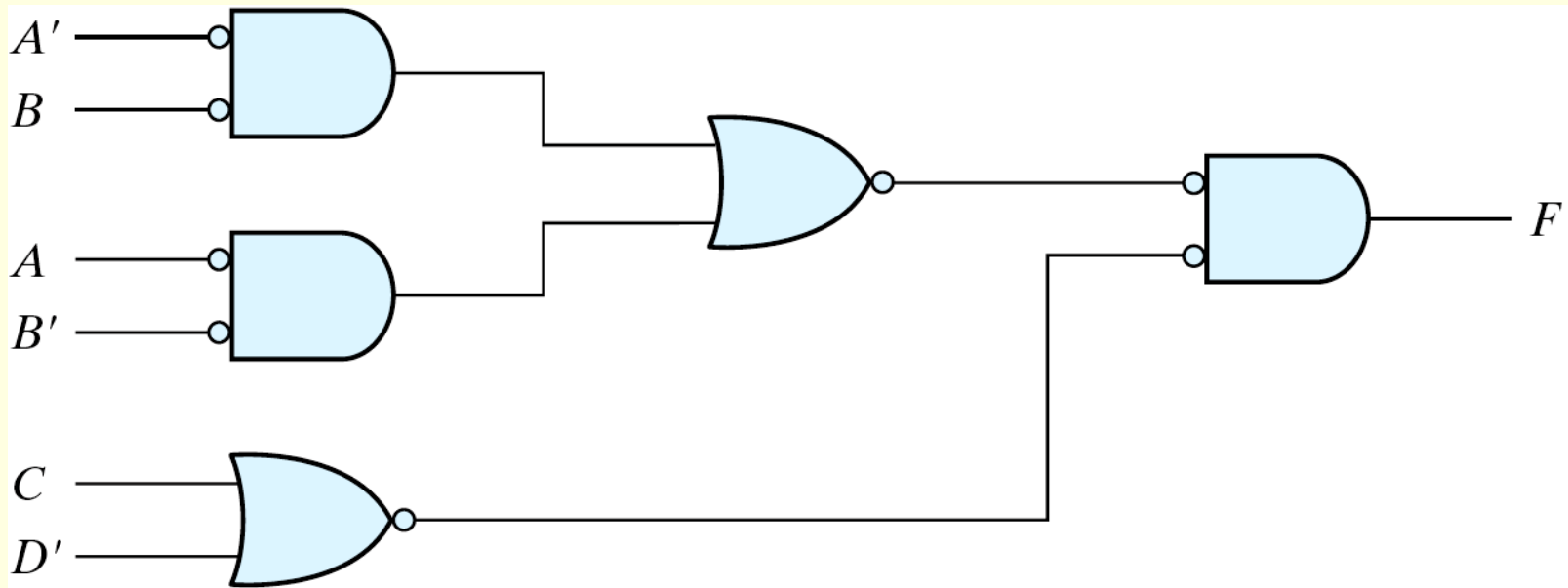


Figure 3.27 Implementing  $F = (AB' + A'B)(C + D')$  with NOR gates

# 3-8 Other Two-level Implementations (

## ■ Wired logic

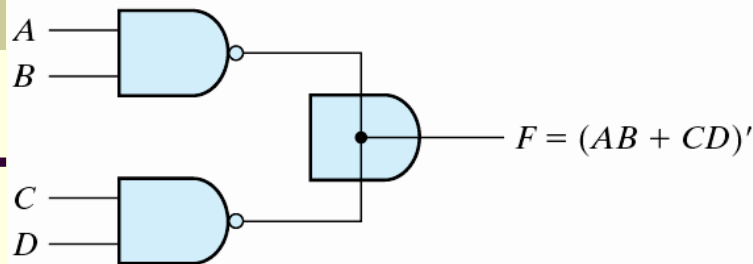
- ❑ A wire connection between the outputs of two gates
- ❑ Open-collector TTL NAND gates: wired-AND logic
- ❑ The NOR output of ECL gates: wired-OR logic

$$F = (AB)' \cdot (CD)' = (AB + CD)' = (A' + B')(C' + D')$$

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

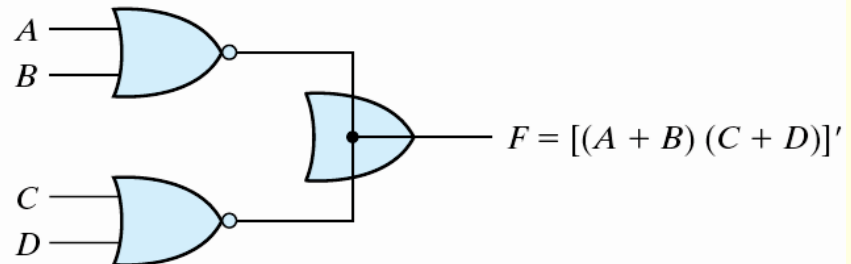
*AND-OR-INVERT function*

*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 3.28 Wired Logic

# Non-degenerate Forms

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

# AND-OR-Invert Implementation

## ■ AND-OR-INVERT (AOI) Implementation

- $\text{NAND-AND} = \text{AND-NOR} = \text{AOI}$
- $F = (AB + CD + E)'$
- $F' = AB + CD + E$  (sum of products)

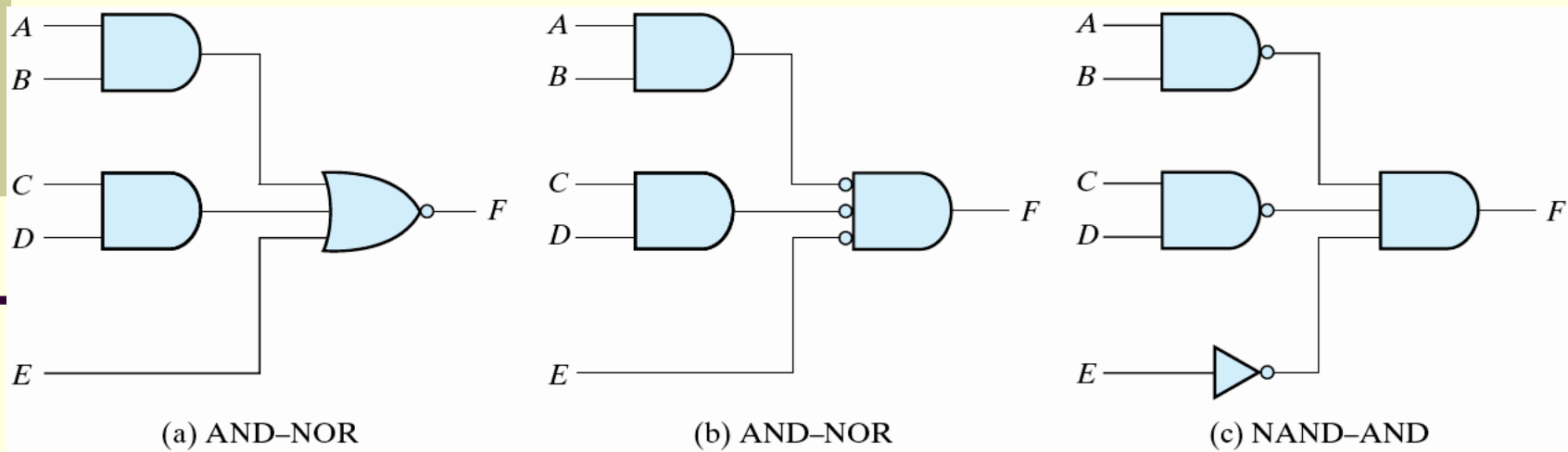


Figure 3.29 AND-OR-INVERT circuits,  $F = (AB + CD + E)'$



# OR-AND-Invert Implementation

## ■ 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)

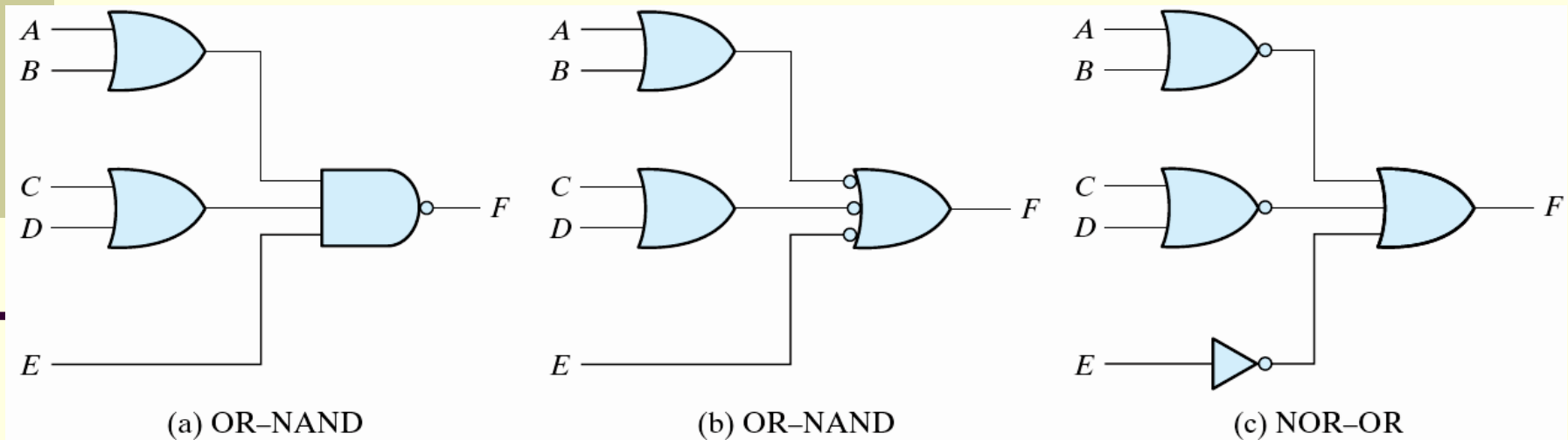


Figure 3.30 OR-AND-INVERT circuits,  $F = ((A+B)(C+D)E)'$

# Tabular Summary and Examples

## ■ Example 3-11: $F = x'y'z' + xyz'$

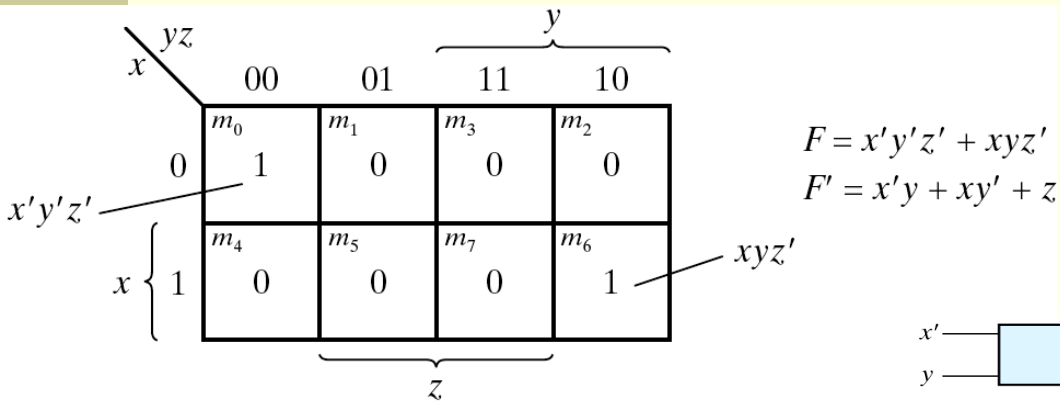
- $F' = x'y + xy' + z$  ( $F'$ : sum of products)
- $F = (x'y + xy' + z)'$  ( $F$ : AOI implementation)
- $F = x'y'z' + xyz'$  ( $F$ : sum of products)
- $F' = (x + y + z)(x' + y' + z)$  ( $F'$ : product of sums)
- $F = ((x + y + z)(x' + y' + z))'$  ( $F$ : OAI)

# Tabular Summary and Examples

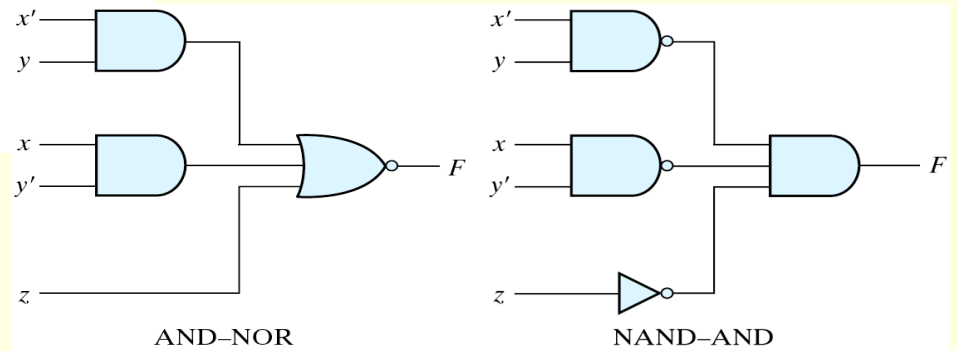
**Table 3.3**  
*Implementation with Other Two-Level Forms*

<b>Equivalent Nondegenerate Form</b>		<b>Implements the Function</b>	<b>Simplify <math>F'</math> into</b>	<b>To Get an Output of</b>
<b>(a)</b>	<b>(b)*</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.	$F$

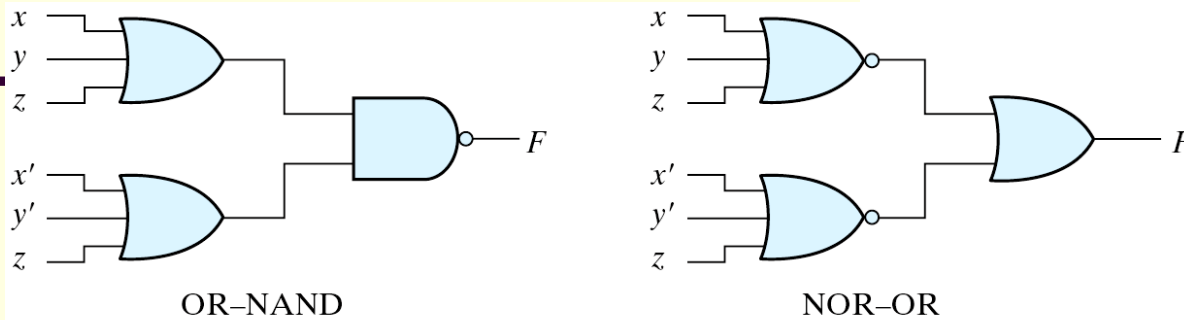
\*Form (b) requires an inverter for a single literal term.



(a) Map simplification in sum of products



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



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

# 3-9 Exclusive-OR Function

- Exclusive-OR (XOR)

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

- Exclusive-NOR (XNOR)

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

- Some identities

- $x \oplus 0 = x$

- $x \oplus 1 = x'$

- $x \oplus x = 0$

- $x \oplus x' = 1$

- $x \oplus y' = (x \oplus y)'$

- $x' \oplus y = (x \oplus y)'$

- Commutative and associative

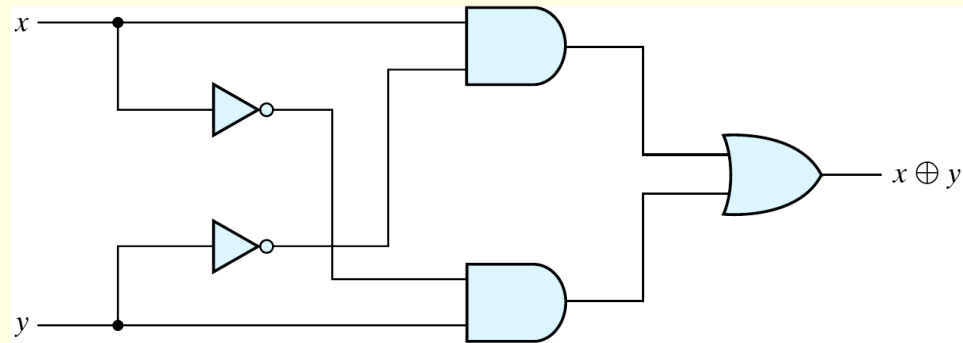
- $A \oplus B = B \oplus A$

- $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

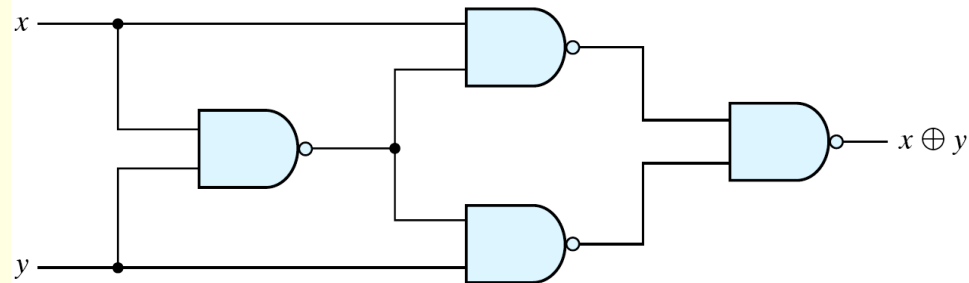
# Exclusive-OR Implementations

## ■ Implementations

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



(a) With AND-OR-NOT gates



(b) With NAND gates

# Odd Function

- $A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C = AB'C' + A'BC' + ABC + A'B'C = \Sigma(1, 2, 4, 7)$
- XOR is an odd function  $\rightarrow$  an odd number of 1's, then  $F = 1$ .
- XNOR is an even function  $\rightarrow$  an even number of 1's, then  $F = 1$ .

BC		B			
		00	01	11	10
A	0	$m_0$	$m_1$	$m_3$	$m_2$
	1	$m_4$	$m_5$	$m_7$	$m_6$
		1		1	

(a) Odd function  $F = A \oplus B \oplus C$

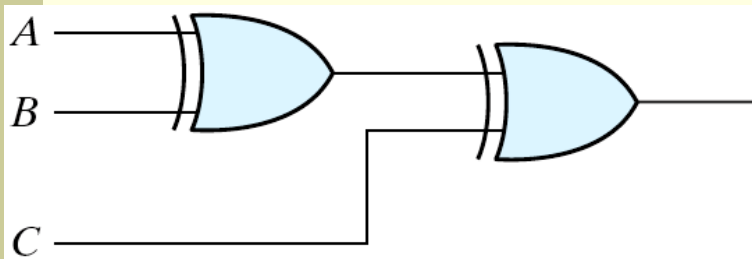
BC		B			
		00	01	11	10
A	0	$m_0$	$m_1$	$m_3$	$m_2$
	1	$m_4$	$m_5$	$m_7$	$m_6$
		1			1

(b) Even function  $F = (A \oplus B \oplus C)'$

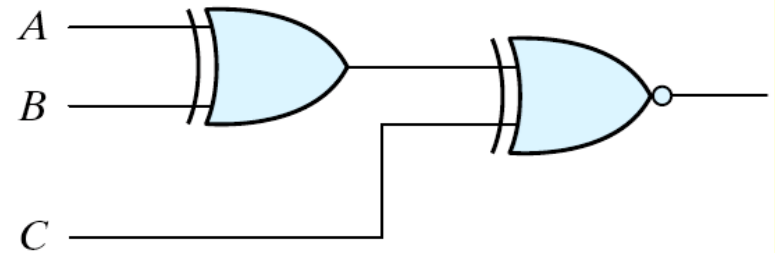
Figure 3.33 Map for a Three-variable Exclusive-OR Function

# XOR and XNOR

- Logic diagram of odd and even functions



(a) 3-input odd function



(b) 3-input even function

Figure 3.34 Logic Diagram of Odd and Even Functions



# Four-variable Exclusive-OR function

## ■ Four-variable Exclusive-OR function

$$\square A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D) = (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D)$$

AB \ CD		C			
		00	01	11	10
A	00	$m_0$	$m_1$ 1	$m_3$	$m_2$ 1
	01	$m_4$ 1	$m_5$	$m_7$ 1	$m_6$
	11	$m_{12}$	$m_{13}$ 1	$m_{15}$	$m_{14}$ 1
	10	$m_8$ 1	$m_9$	$m_{11}$ 1	$m_{10}$
		D			

(a) Odd function  $F = A \oplus B \oplus C \oplus D$

AB \ CD		C			
		00	01	11	10
A	00	$m_0$ 1	$m_1$	$m_3$ 1	$m_2$
	01	$m_4$	$m_5$ 1	$m_7$	$m_6$ 1
	11	$m_{12}$ 1	$m_{13}$	$m_{15}$ 1	$m_{14}$
	10	$m_8$	$m_9$ 1	$m_{11}$	$m_{10}$ 1
		D			

(b) Even function  $F = (A \oplus B \oplus C \oplus D)'$

Figure 3.35 Map for a Four-variable Exclusive-OR Function

# Parity Generation and Checking

## ■ Parity Generation and Checking

- A parity bit:  $P = x \oplus y \oplus z$
- Parity check:  $C = x \oplus y \oplus z \oplus P$ 
  - $C=1$ : one bit error or an odd number of data bit error
  - $C=0$ : correct or an even # of data bit error

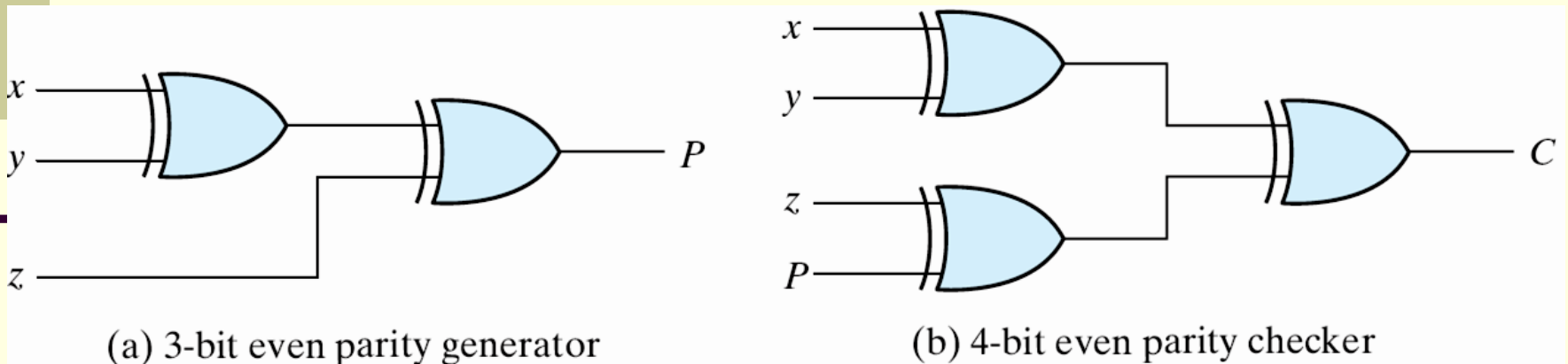


Figure 3.36 Logic Diagram of a Parity Generator and Checker

# Parity Generation and Checking

**Table 3.4**

*Even-Parity-Generator Truth Table*

Three-Bit Message			Parity Bit
<i>x</i>	<i>y</i>	<i>z</i>	<i>P</i>
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

# Parity Generation and Checking

**Table 3.5**  
*Even-Parity-Checker Truth Table*

Four Bits Received				Parity Error Check
<i>x</i>	<i>y</i>	<i>z</i>	<i>P</i>	<i>C</i>
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Thanks To  
Mustafa Kemal Uyguroğlu