CS2022: 數位系統設計

Gate-Level Minimization

Outline

- Introduction
- The Map Method
- Four-Variable Map
- Product-of-Sums Simplification
- Don't-Care Conditions
- NAND and NOR Implementation
- Other Two-Level Implementation
- Exclusive-OR Function

Introduction

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

Review of Boolean Function

- **Boolean function**
  - Sum of minterms (or product of maxterms) – canonical form
    - Another form of truth table representation
  - Sum of products (or product of sums) – standard form
    - A circuit with less hardware resource in the simplest form
      - Minimum number of terms
      - Minimum number of literals
    - The simplified expression may not be unique

The Map Method

- The complexity of the digital logic gates
  - The complexity of the algebraic expression
- Logic minimization
  - Algebraic approach is lack of specific procedure applying the theorems
  - The Karnaugh map
    - A simple straight forward procedure
    - A pictorial form of a truth table
    - Applicable if the # of variables $\leq$ 6
- A diagram made up of squares
  - Each square represents one minterm

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**
  - **Fig. 3.2(a): $xy = m_3$**
  - **Fig. 3.2(b): $x + y = x'y + xy' + xy = m_1 + m_2 + m_3$**

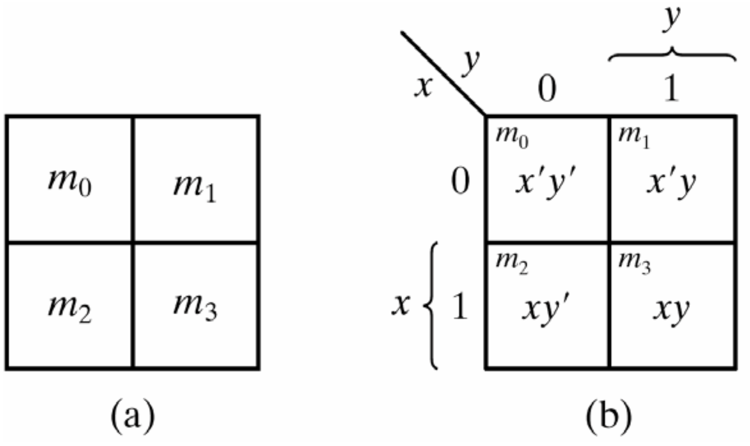


Figure 3.1 Two-variable Map

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)

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

(b)

Figure 3.3 Three-variable Map

A Three-Variable Map

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

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)

Fig. 3-3 Three-variable Map

Example 1

- **Example 1: simplify the Boolean function F(x, y, z) = Σ(2, 3, 4, 5)**
  - $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

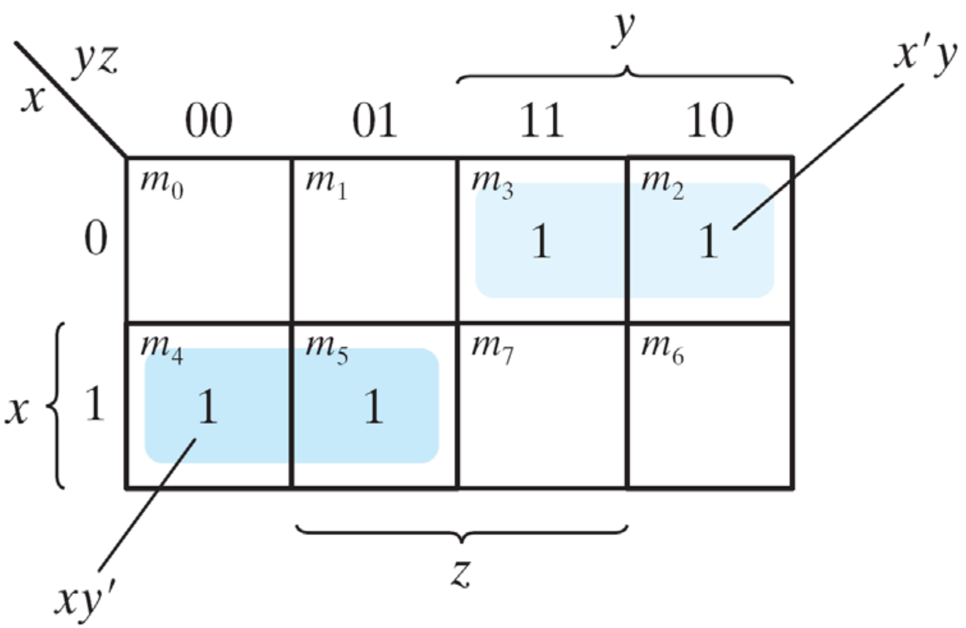


Figure 4 Map for Example 1, $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

Example 2

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

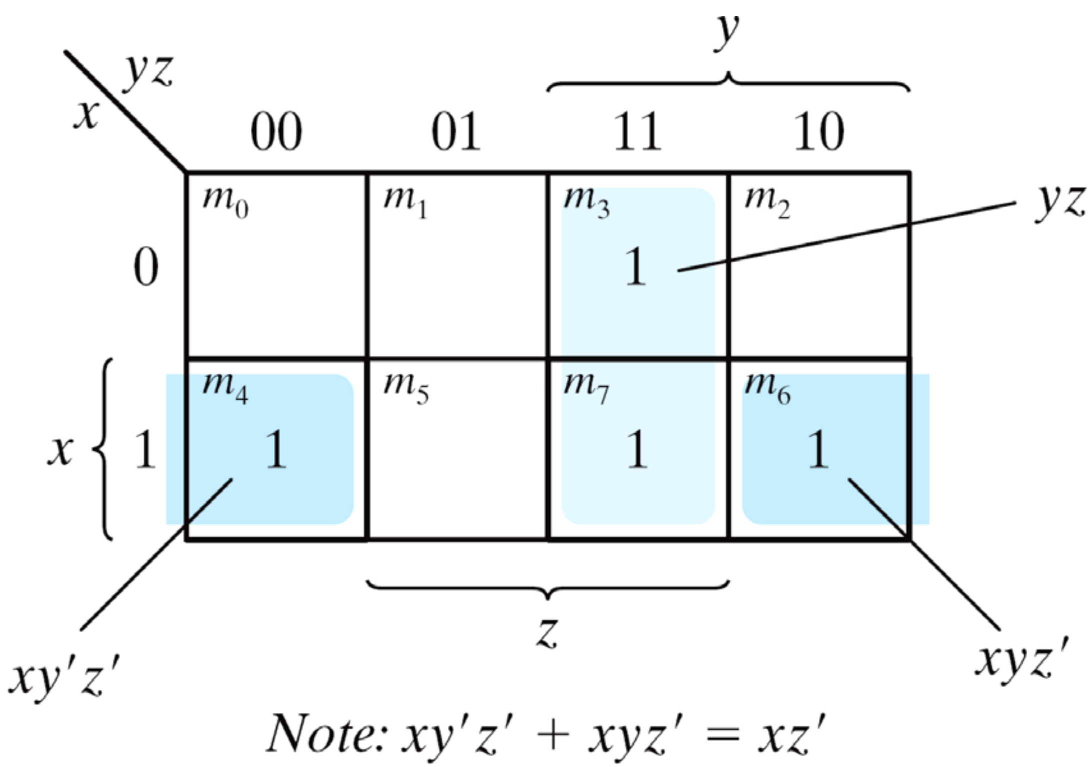


Figure 5 Map for Example 2; ***F*(*x, y, z*) = Σ(3, 4, 6, 7) = *yz* + *xz*'**

Four adjacent Squares

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

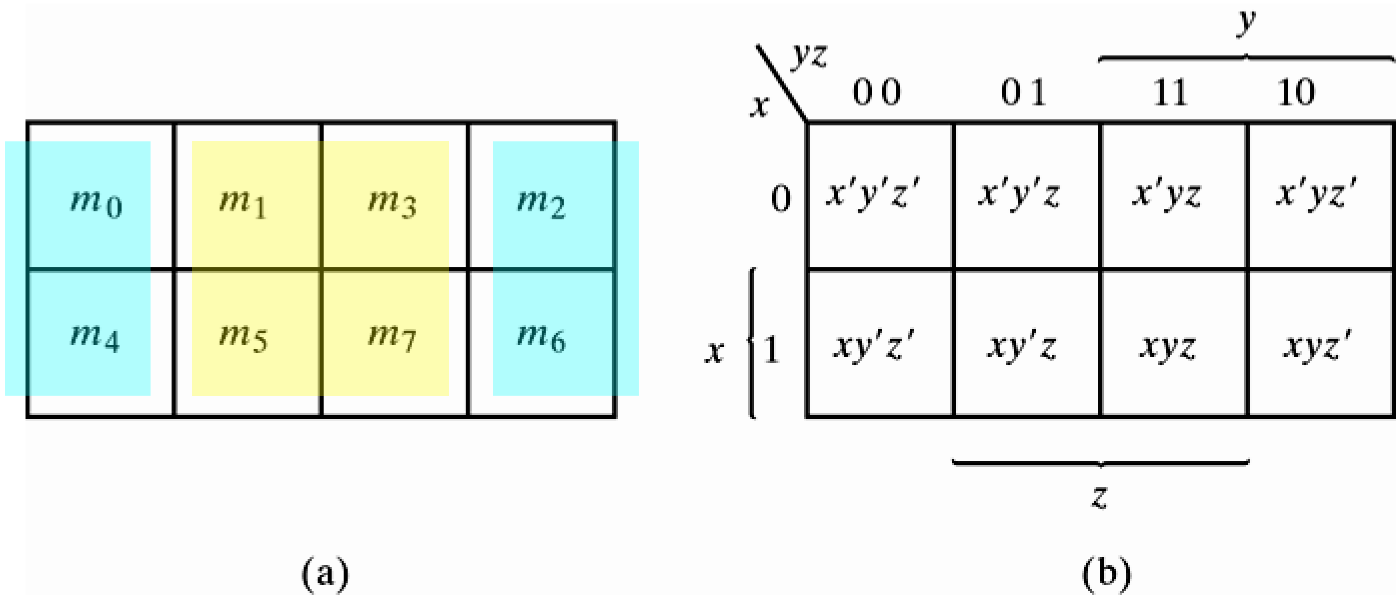


Figure 3 Three-variable Map

Example 3

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

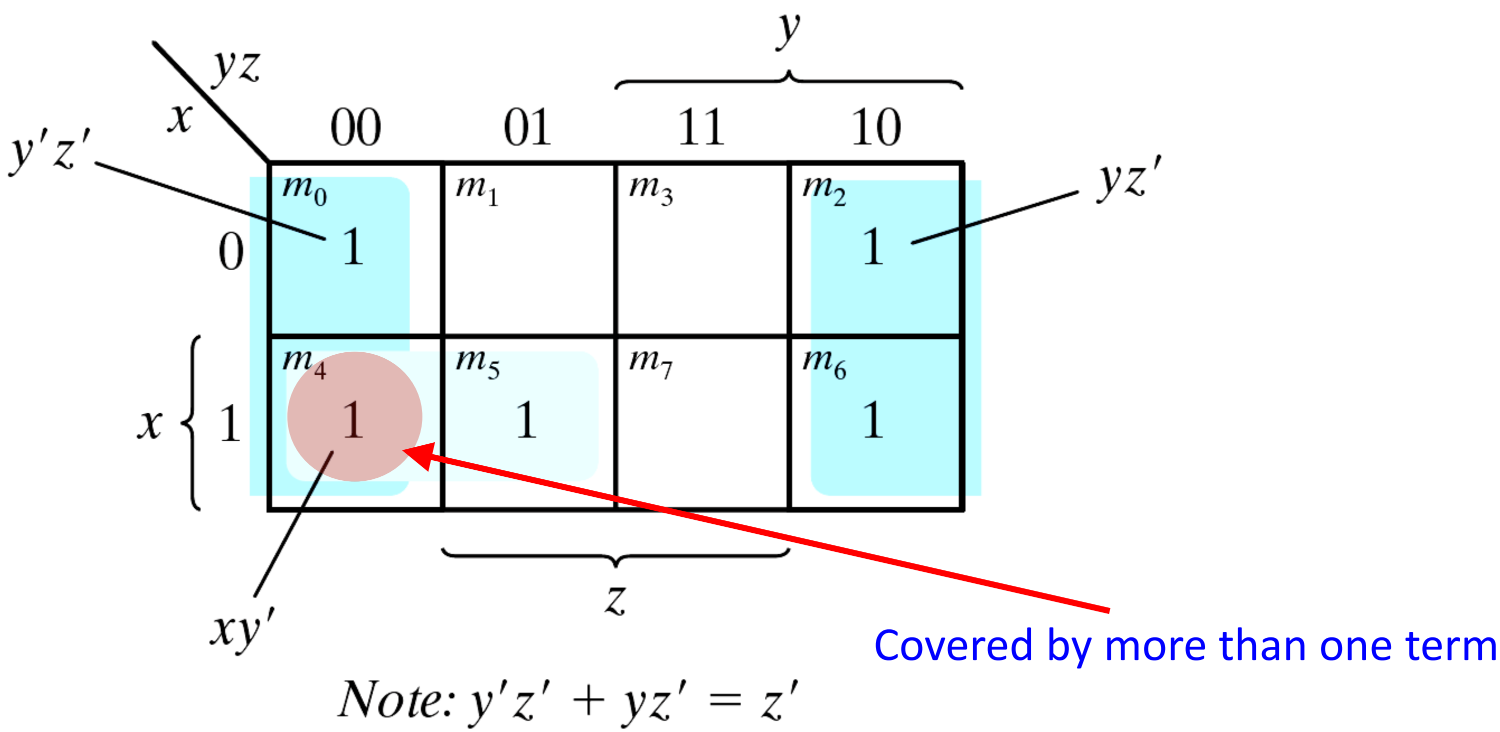


Figure 6 Map for Example 3, $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

Example 4

- **Example 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**

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

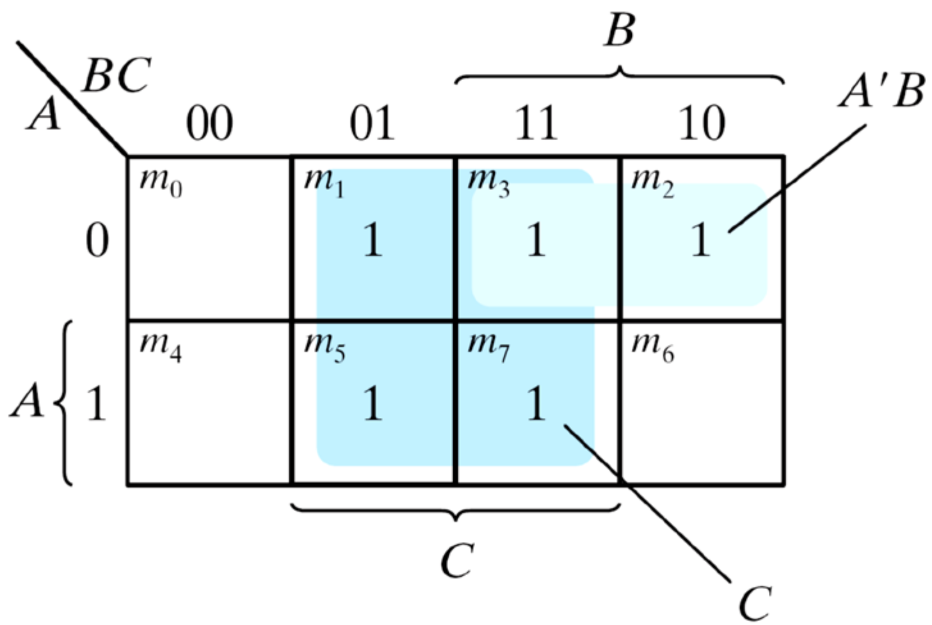


Figure 7 Map for Example 4, ***A'C + A'B + AB'C + BC = C + A'B***

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)

$wx \backslash yz$	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	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

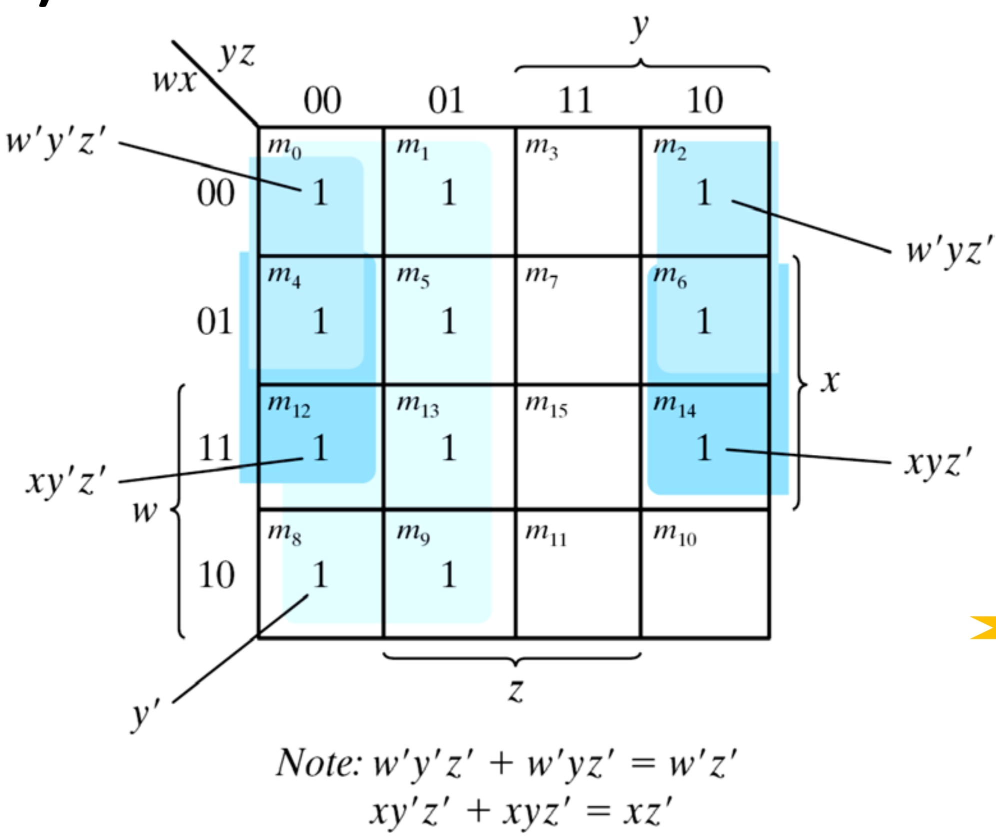
y, x, w, z

(b)

Figure 8 Four-variable Map

Example 5

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

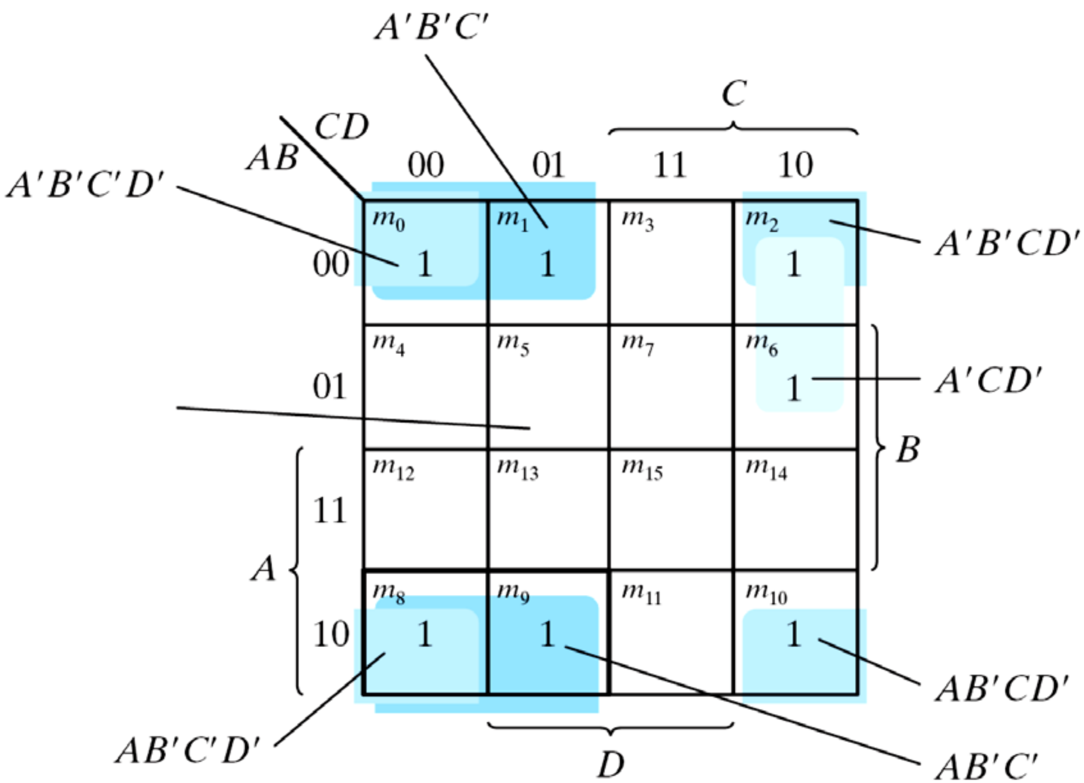


$$F = y' + w'z' + xz'$$

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

Example 6

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



$$\text{Note: } A'B'C'D' + A'B'CD' = A'B'D'$$
$$AB'C'D' + AB'CD' = AB'D'$$
$$A'B'D' + AB'D' = B'D'$$
$$A'B'C' + AB'C' = B'C'$$

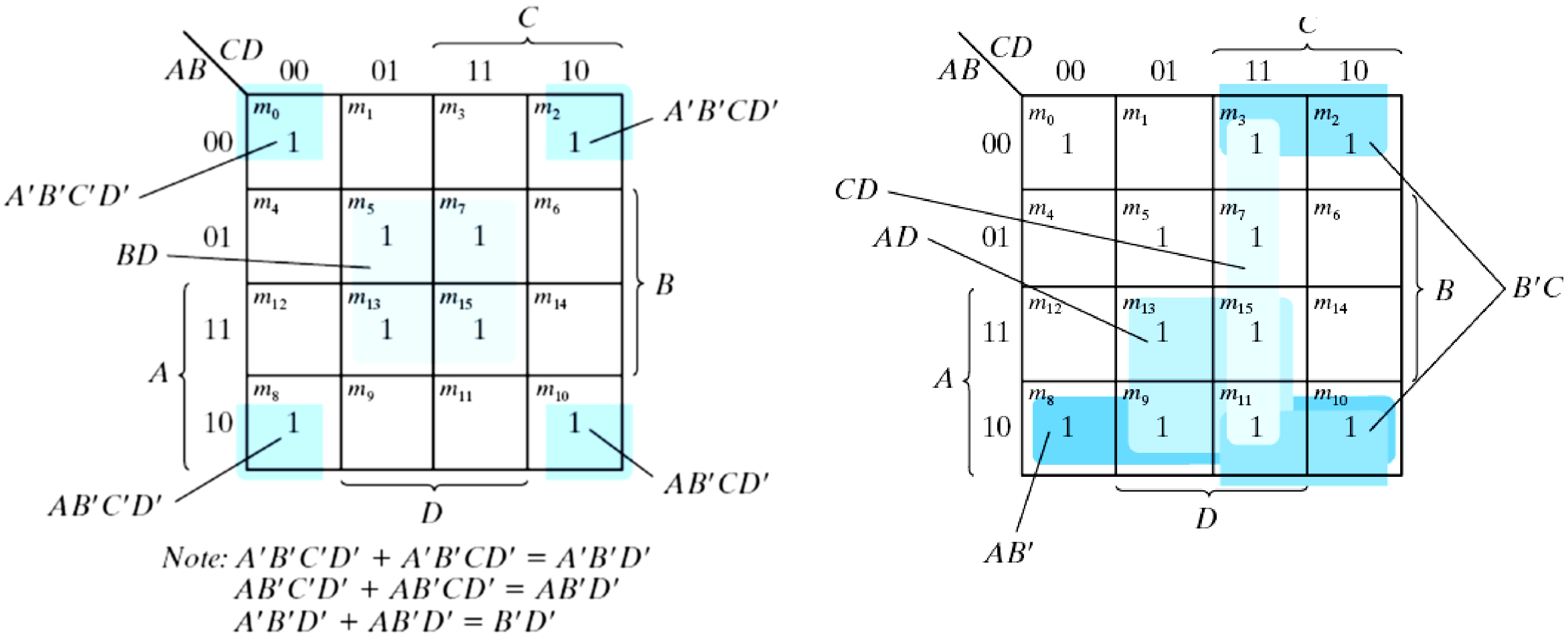
Figure 9 Map for Example 6; $A'B'C' + B'CD' + A'B'C'D' + AB'C' = B'D' + B'C' + A'CD'$

Prime Implicants

- **Design objectives**
  - **All the minterms must be covered**
  - **Minimize the number of terms**
- **Prime Implicant (PI)**
  - **PI: A product term obtained by combining the maximum possible number of adjacent squares in Karnaugh map**
  - **Essential PI: a minterm is covered by only one prime implicant**
    - **The essential PI must be included**

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



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

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

Figure 11 Simplification Using Prime Implicants

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)**

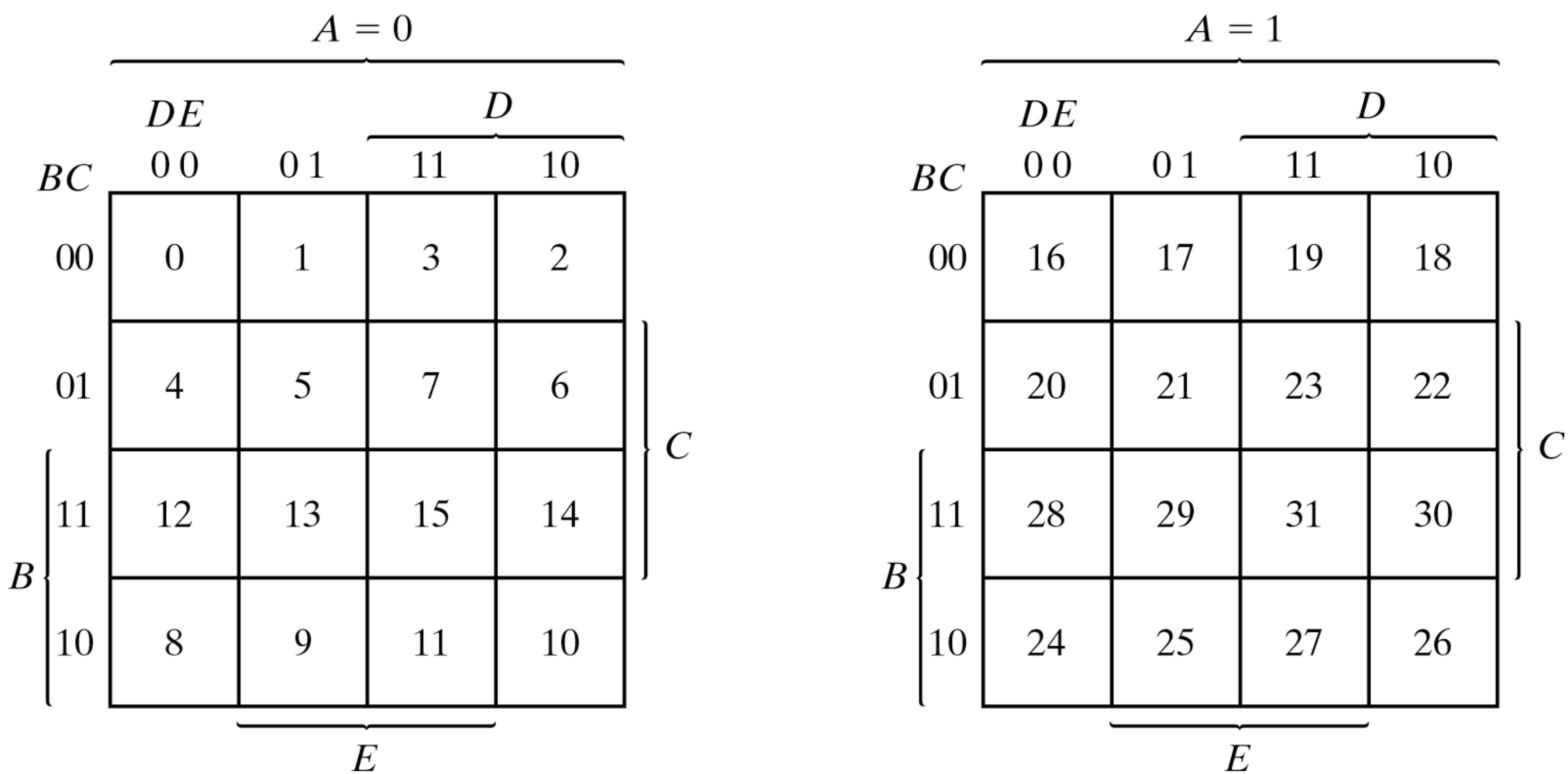


Figure 12 Five-variable Map

Square Number and Literals

- Table 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

	Number of Adjacent Squares	Number of Literals in a Term in an *n*-variable Map			
K	2^k	$n = 2$	$n = 3$	$n = 4$	$n = 5$
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 7

- Example 7: simplify $F = \Sigma(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$

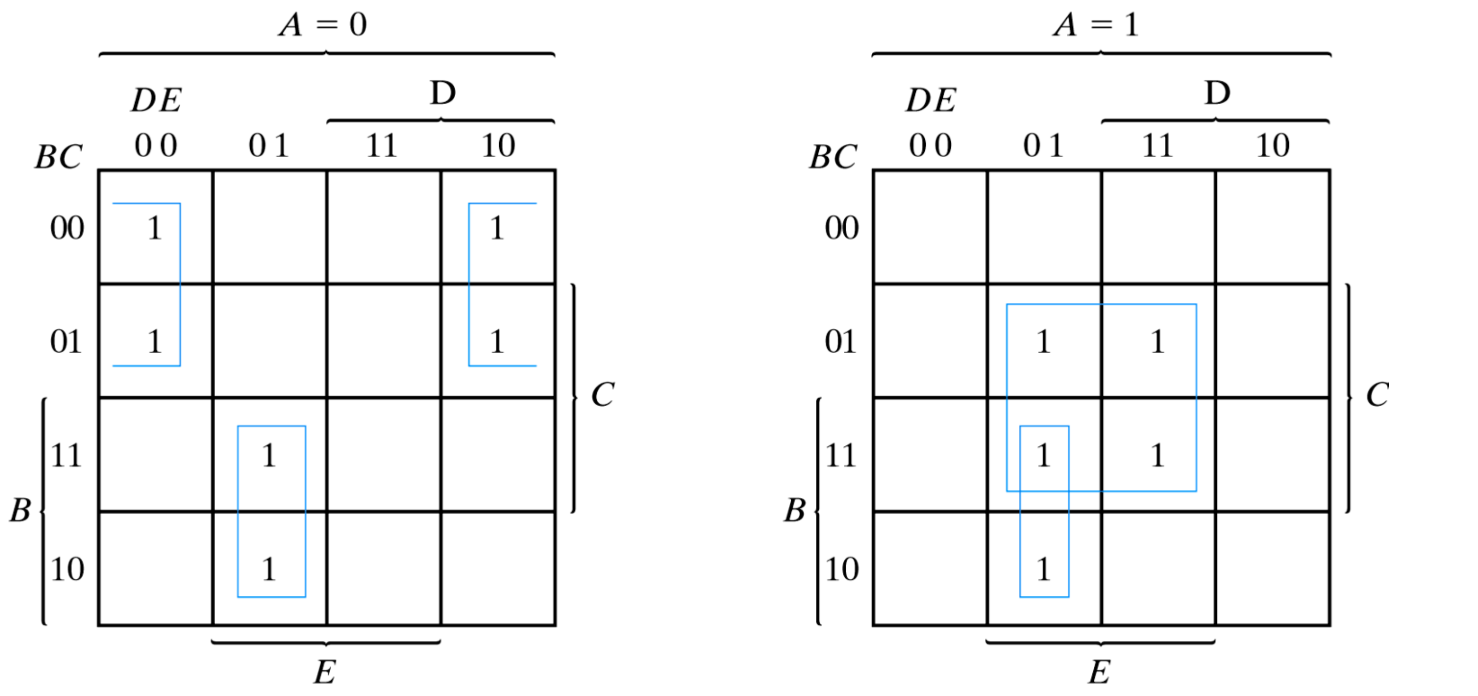


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

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

Example 7 (cont.)

- **Another Map for Example 7**

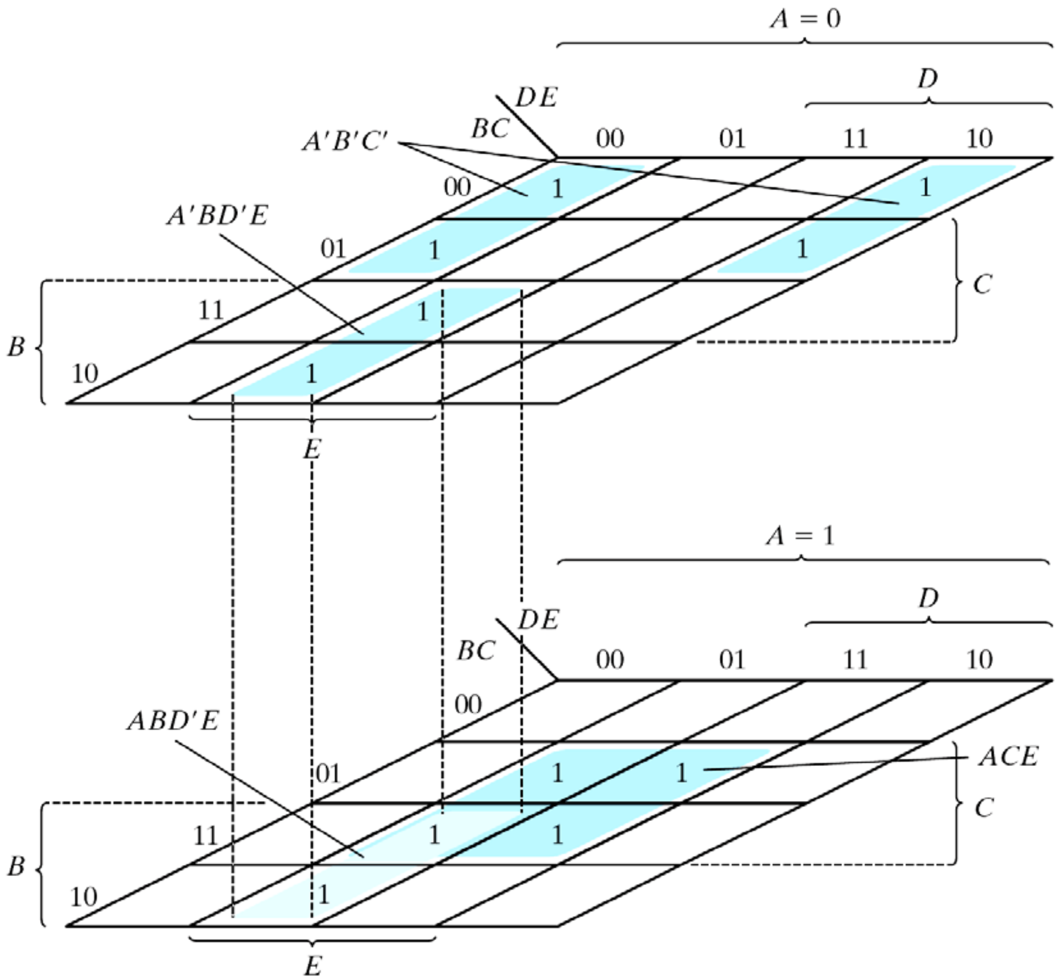
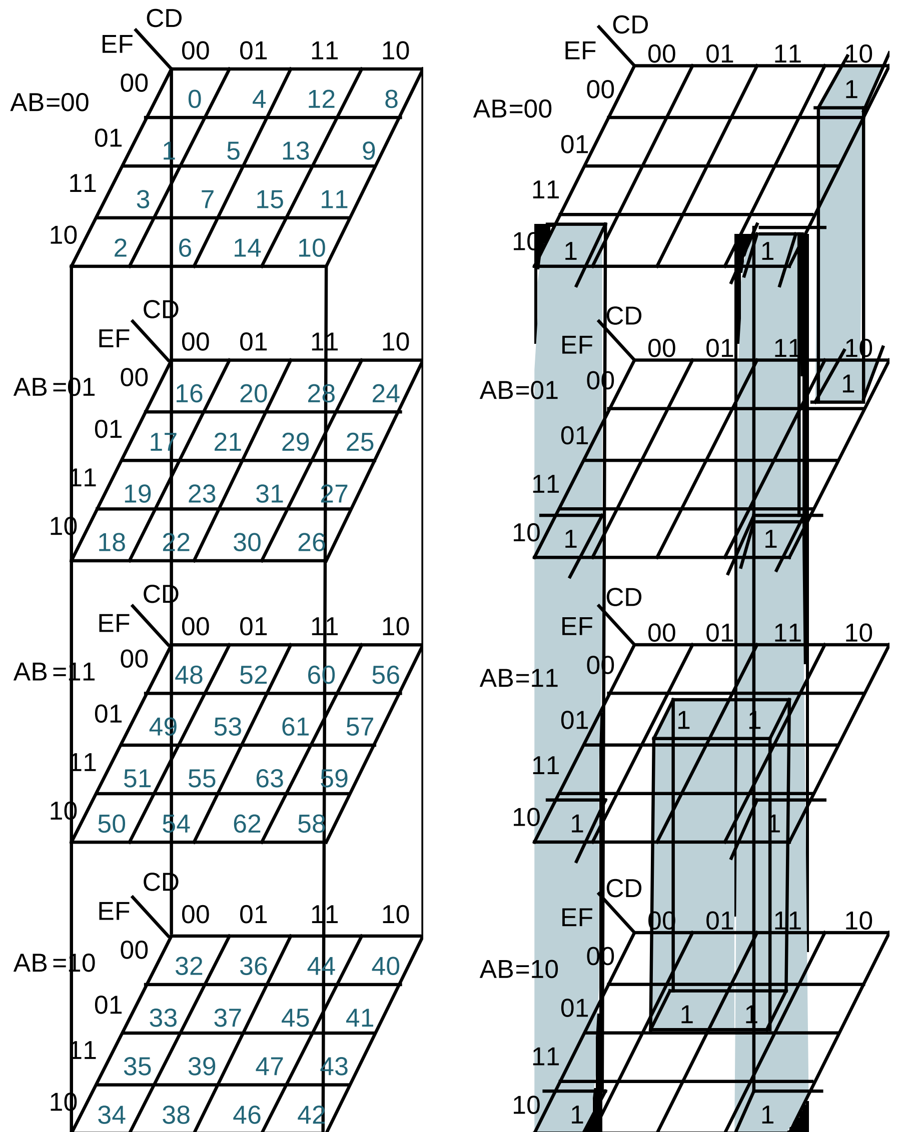


Figure 13 Map for Example 7, $F = A'B'E' + BD'E + ACE$

Six-Variable Map Example

F(A,B,C,D,E,F)

= Σ(2,8,10,18,24,26,34,37,42,45,50,53,58,61)



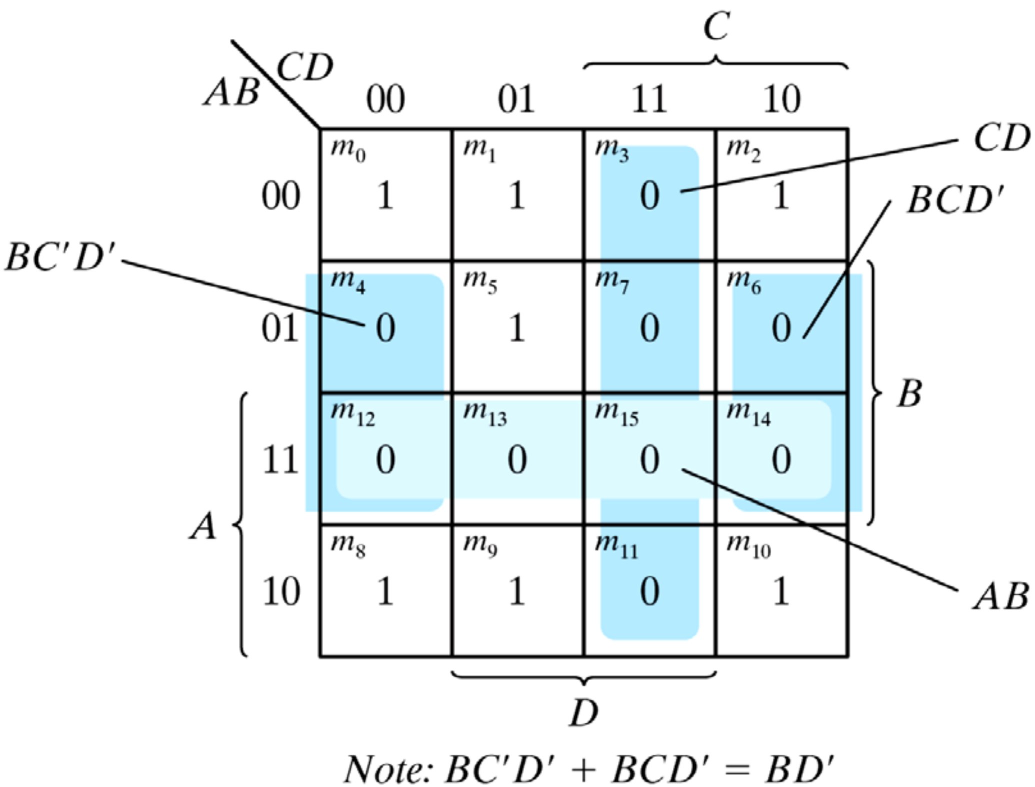
Product of Sums Simplification

- **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 7

- **Example 7: 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*)= S(0, 1, 2, 5, 8, 9, 10) = *B'D'+B'C'+A'C'D***

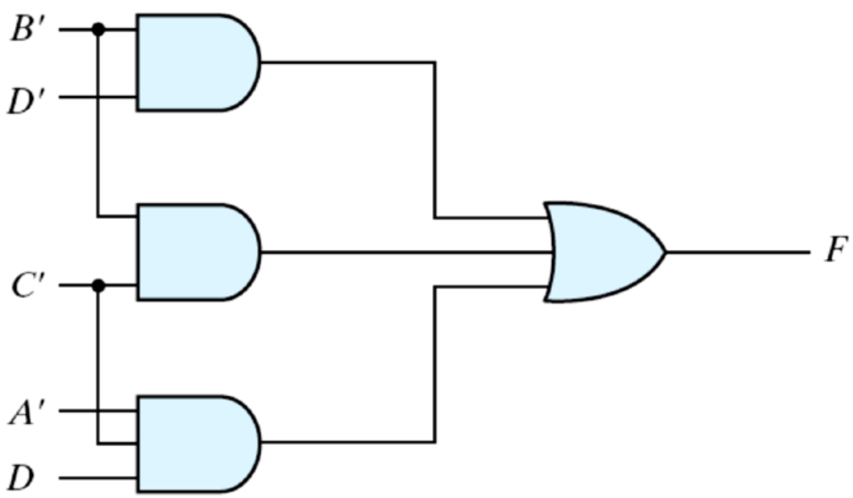
b) ***F' = AB+CD+BD'***

- » **Apply DeMorgan's theorem;** ***F=(A'+B')(C'+D')(B'+D)***

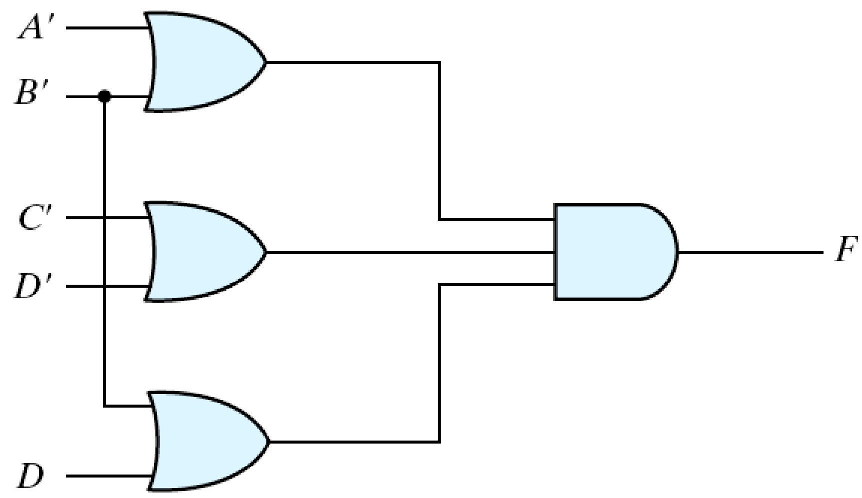
Figure 12 Map for Example 7, $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D'+B'C'+A'C'D$

Example 7 (cont.)

- Gate implementation of the function of Example 7



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



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

Sum-of-products form (AND-OR)

Product-of-sums form (OR-AND)

Figure 13 Gate implementations of the function of Example 7

Product of Maxterms Procedure

- **Consider the function defined in Table 1**
  - **In sum-of-minterm:**

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

  - **In product-of-maxterm:**

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

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

Product of Sums Procedure

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

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

  - Combine the 0's :

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

  - Taking the complement of F′

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

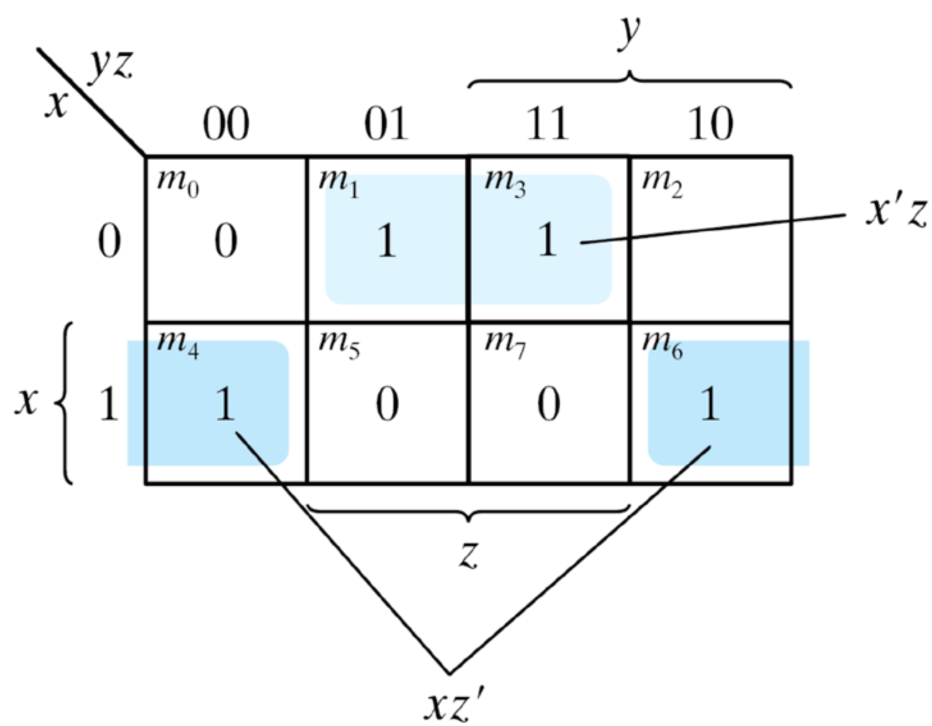


Figure 14 Map for the function of Table 1

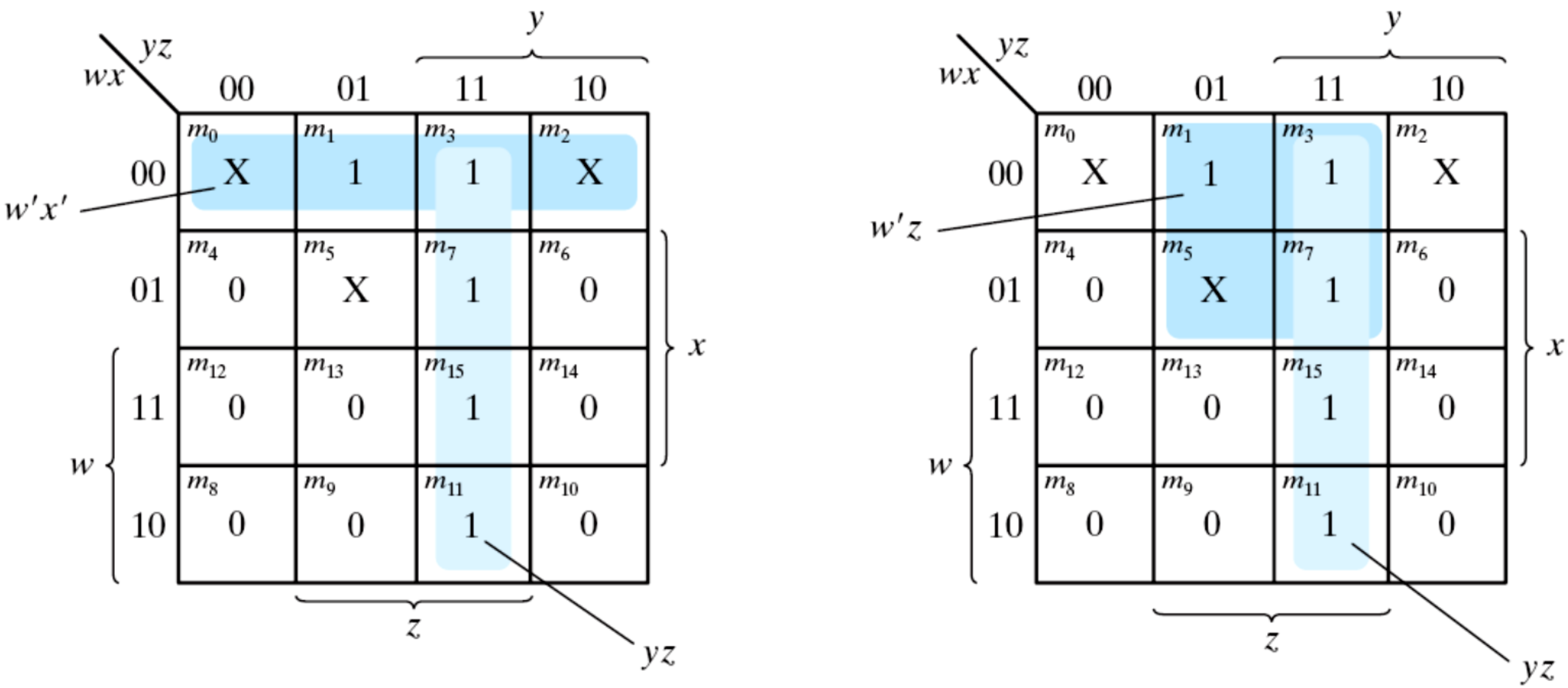
Don't-Care Conditions

- **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 8: 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 8 (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**



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

Figure 15 Example with don't-care conditions

NAND and NOR Implementation

- **NAND gate is a universal gate**
  - **Can implement any Boolean function**

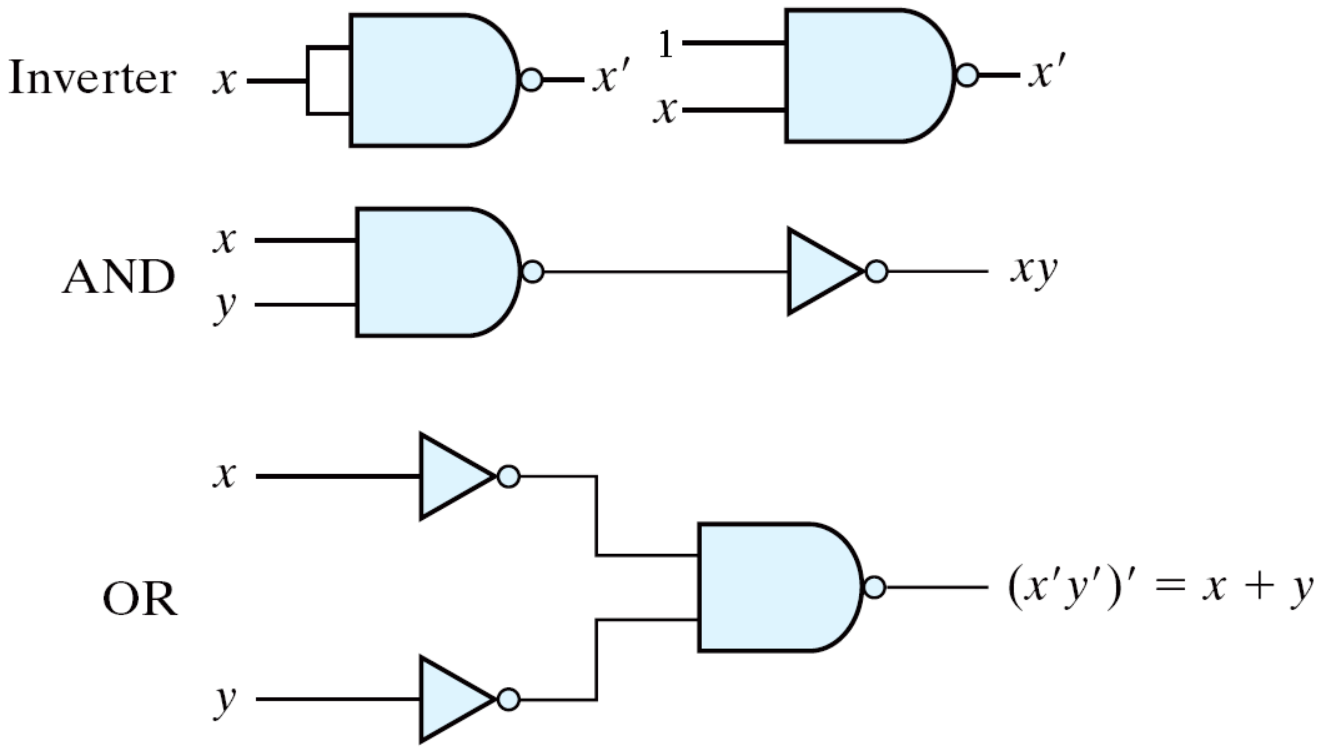


Figure 16 Logic operations with NAND gates

NAND Gate

- **Two graphic symbols for a NAND gate**

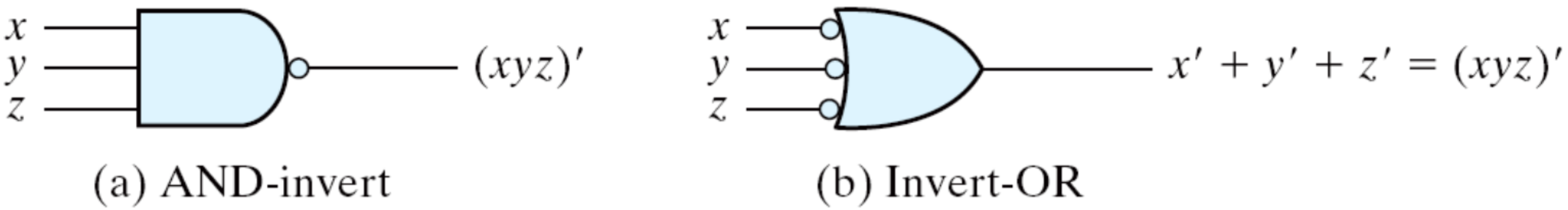


Figure 17 Two graphic symbols for NAND gate

Two-level Implementation

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

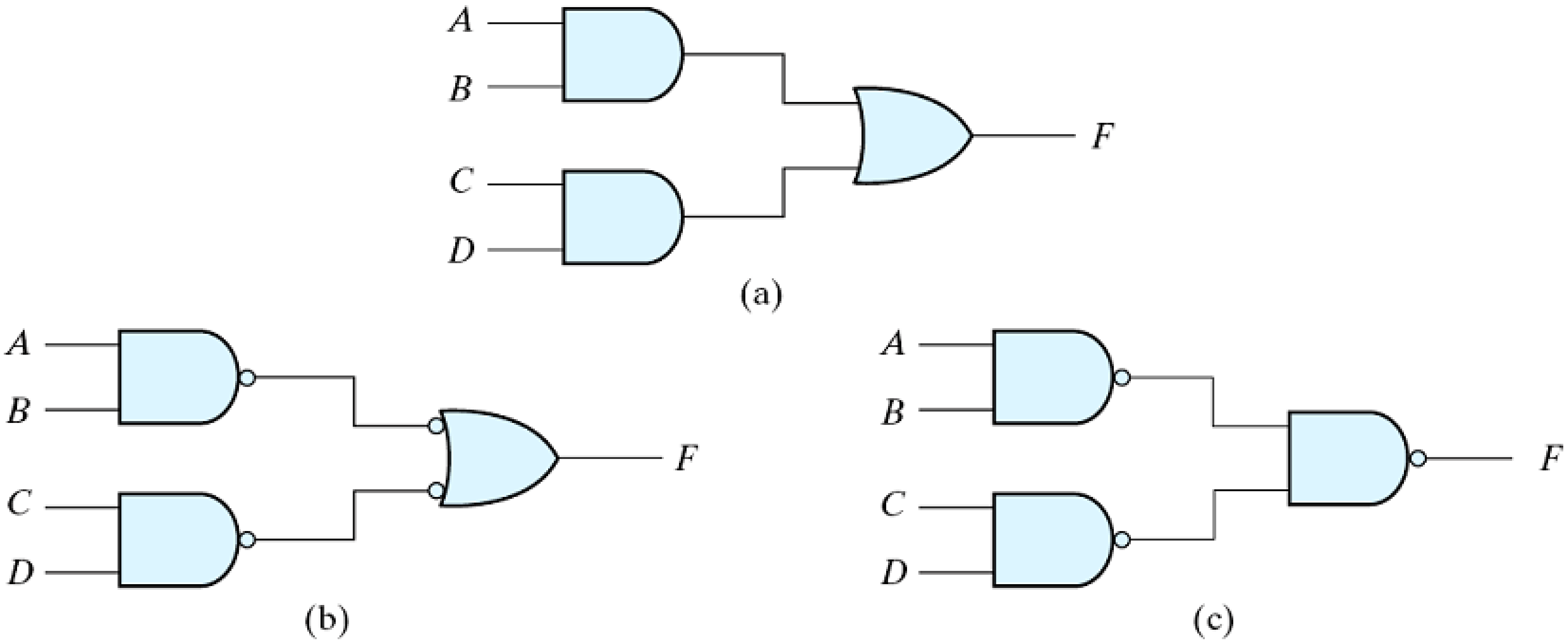


Figure 18 Three ways to implement $F = AB + CD$

Example 9

- **Example 9: implement *F*(*x, y, z*) with NAND gates:**

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

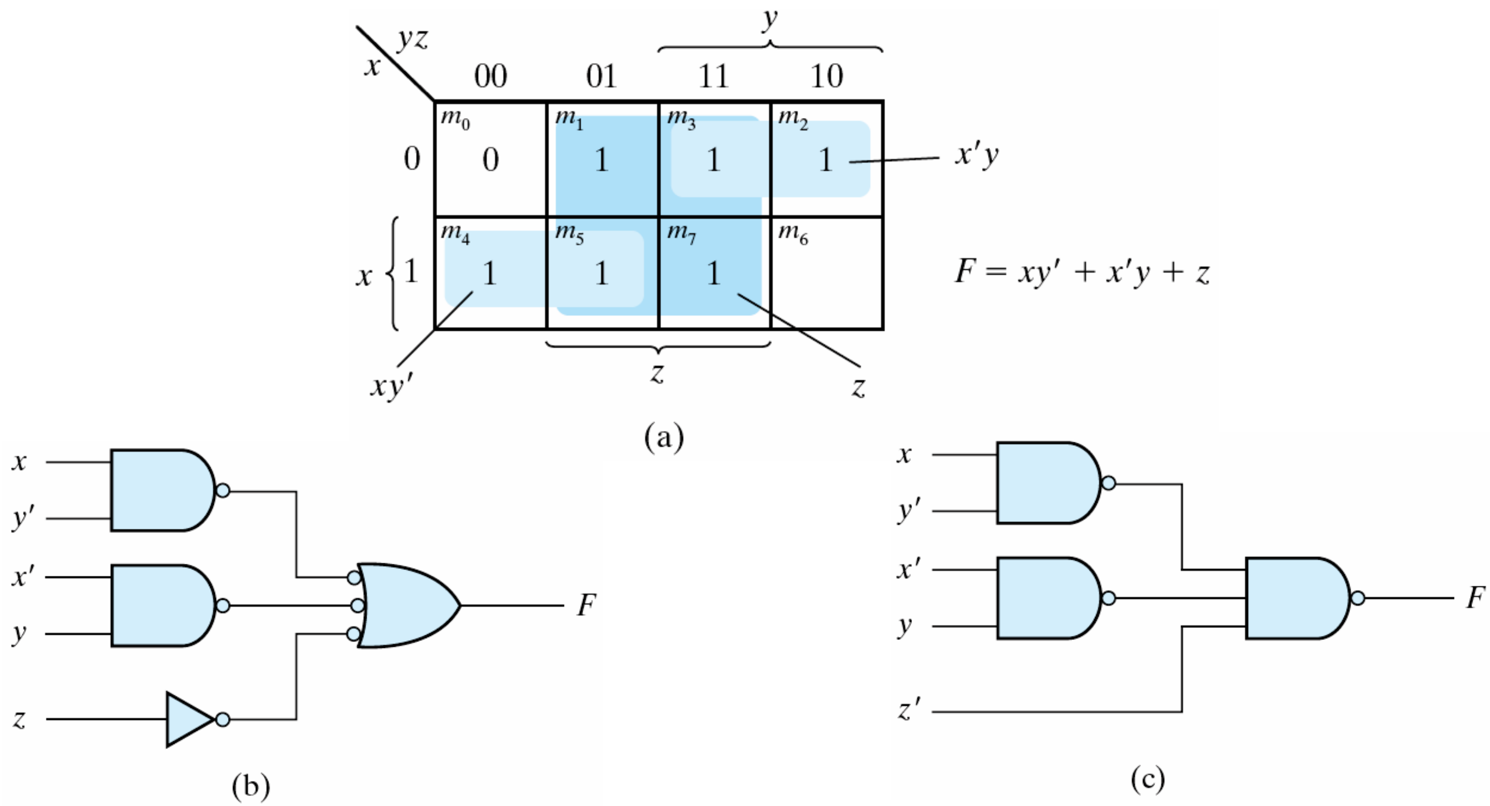


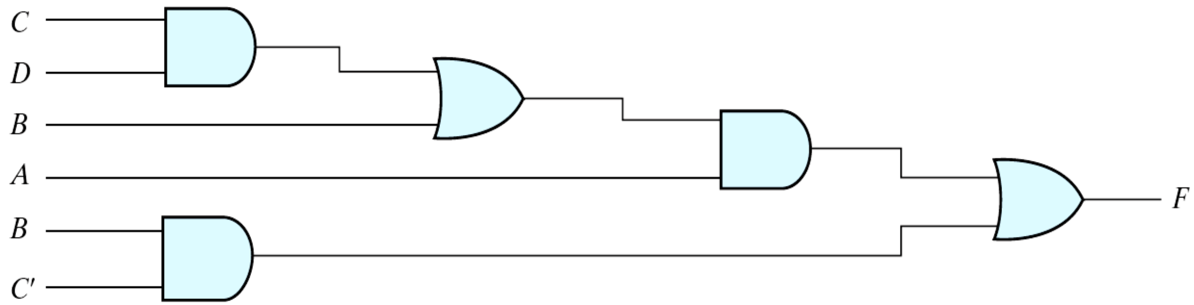
Figure 19 Solution to Example 9

Procedure with Two Levels NAND

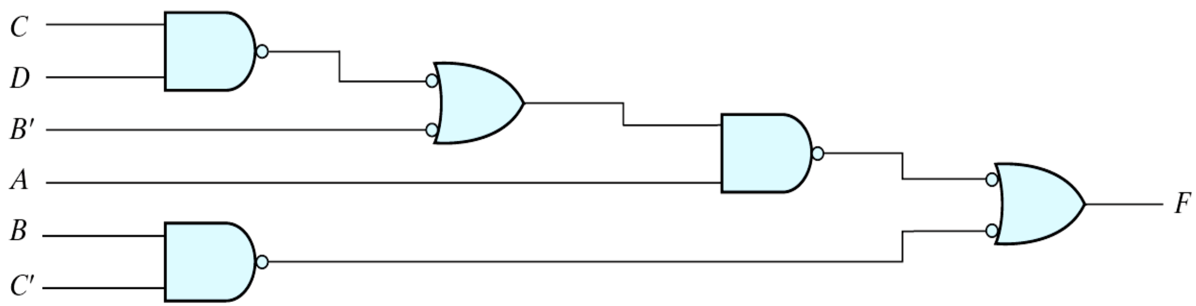
- **The procedure**
  - **Simplified in the form of sum of products;**
  - **A NAND gate for each product term; the inputs to each NAND gate are the literals of the term (the first level);**
  - **A single NAND gate for the second sum term (the second level);**
  - **A term with a single literal requires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

Multilevel NAND Circuits

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



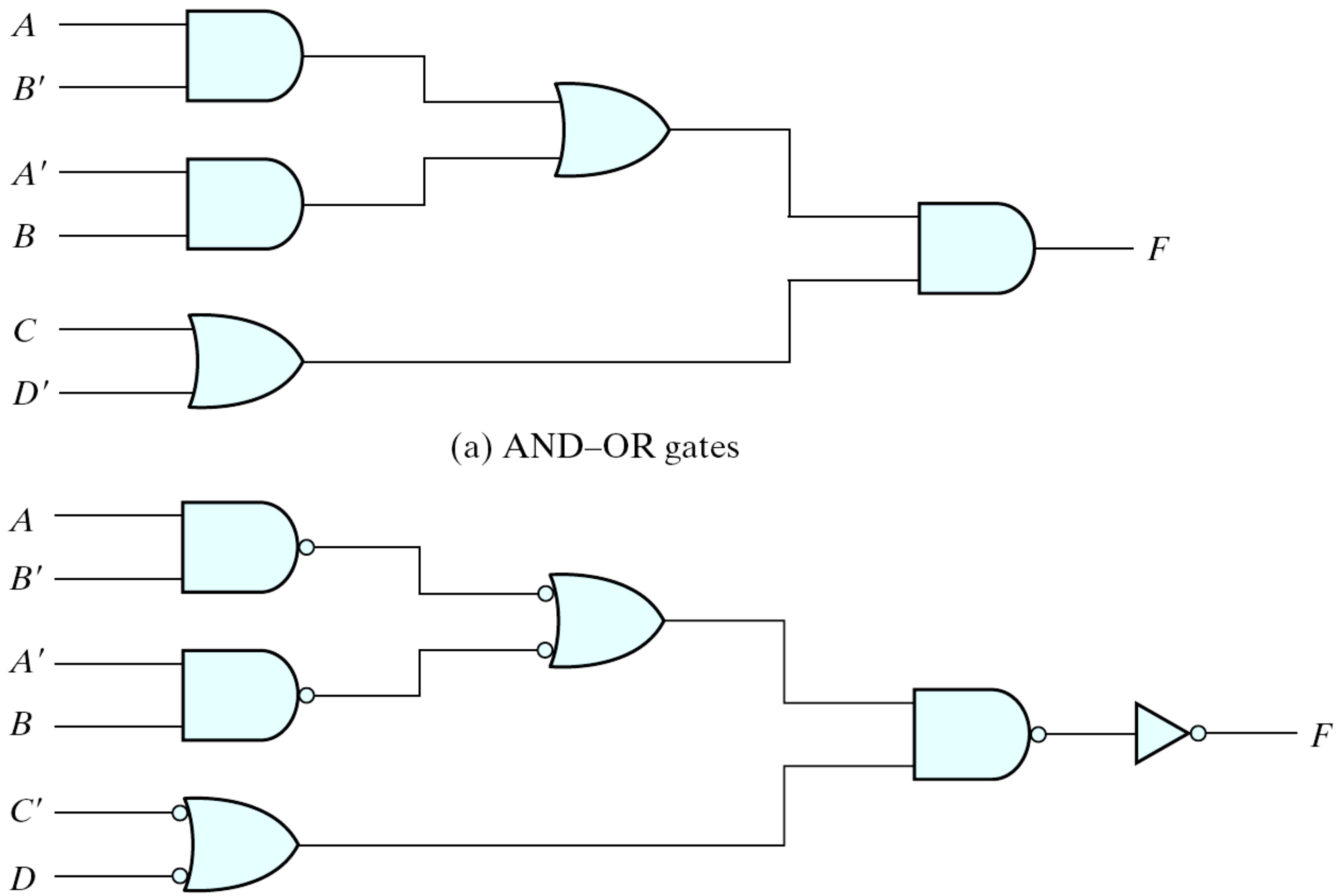
(a) AND–OR gates **Alternating levels of AND and OR gates**



(b) NAND gates

Figure 20 Implementing $F = A(CD + B) + BC'$

NAND Implementation



(a) AND–OR gates

(b) NAND gates

Figure 21 Implementing $F = (AB' + A'B)(C + D')$

NOR Implementation

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

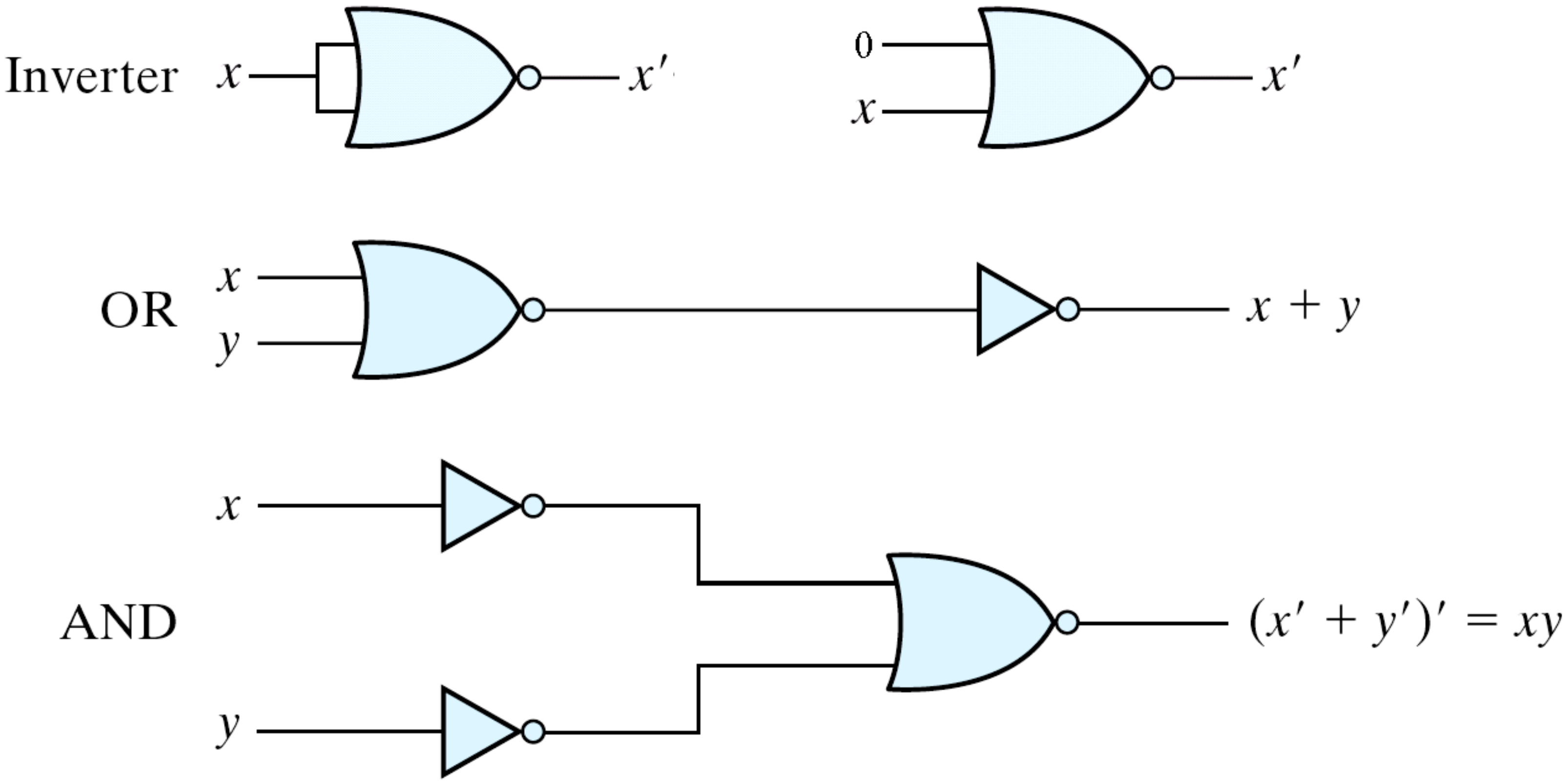


Figure 22 Logic Operation with NOR Gates

Two Graphic Symbols for a NOR Gate

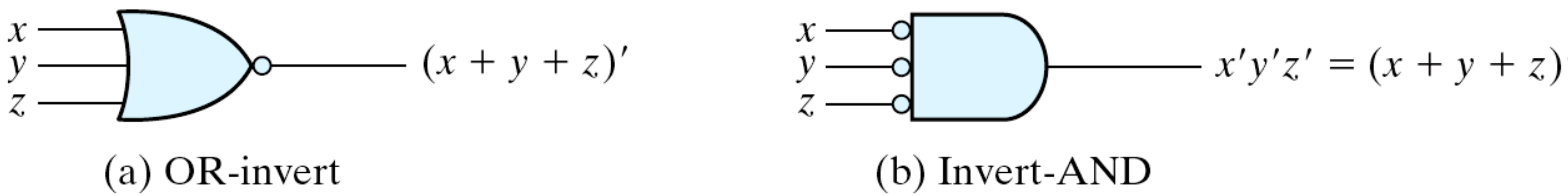


Figure 23 Two Graphic Symbols for NOR Gate

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

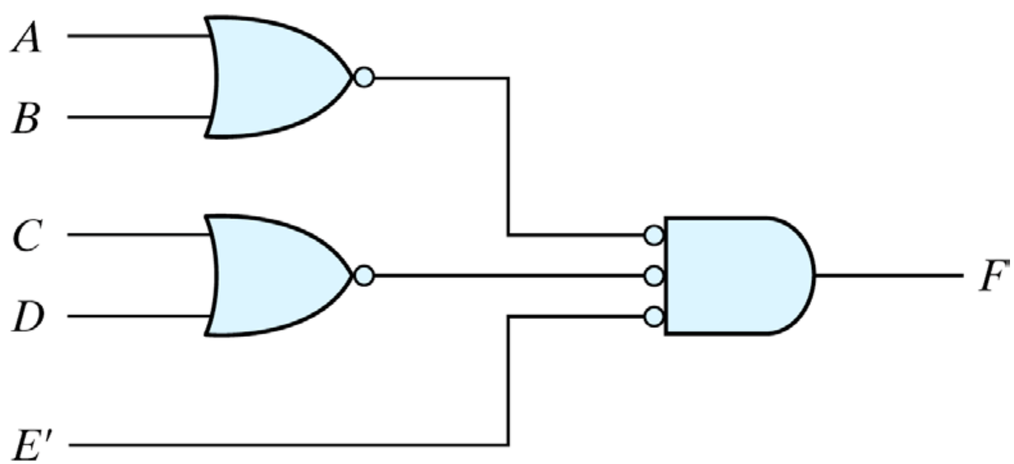


Figure 24 Implementing $F = (A + B)(C + D)E$

Example

Example: Implement $F = (AB' + A'B)(C + D')$ with NOR gates

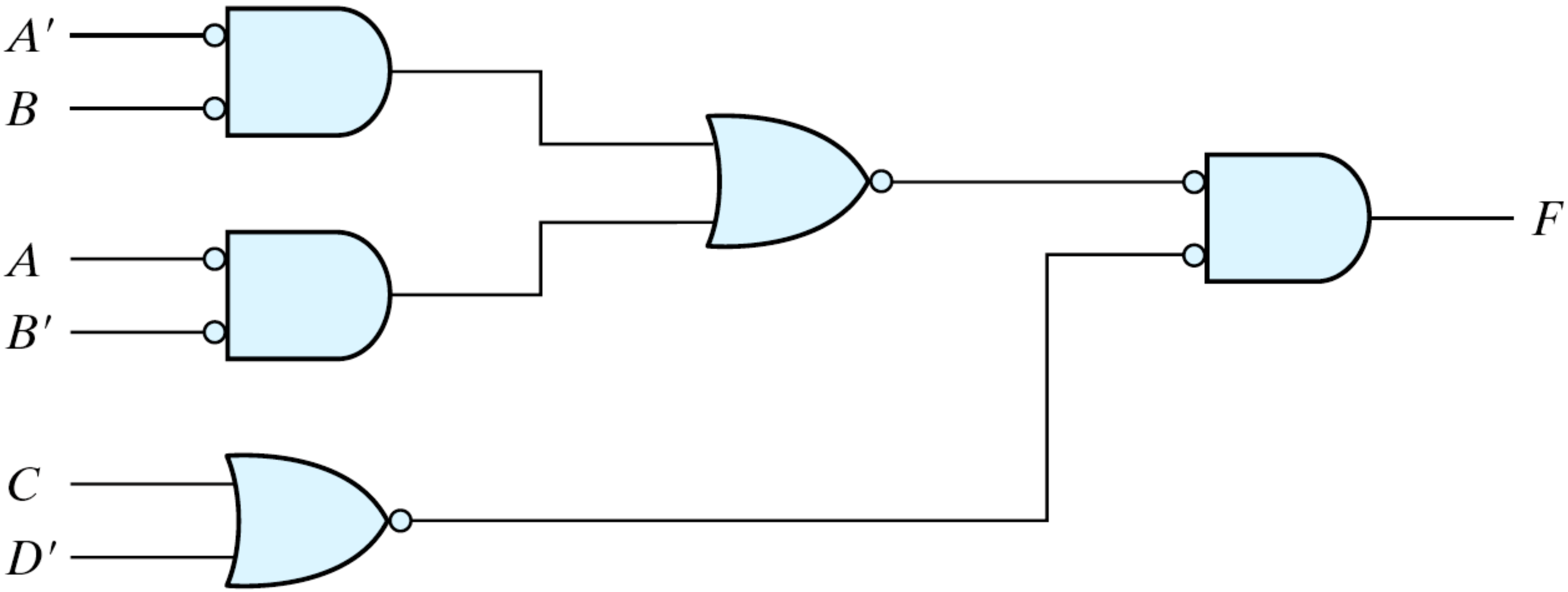


Figure 25 Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

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*

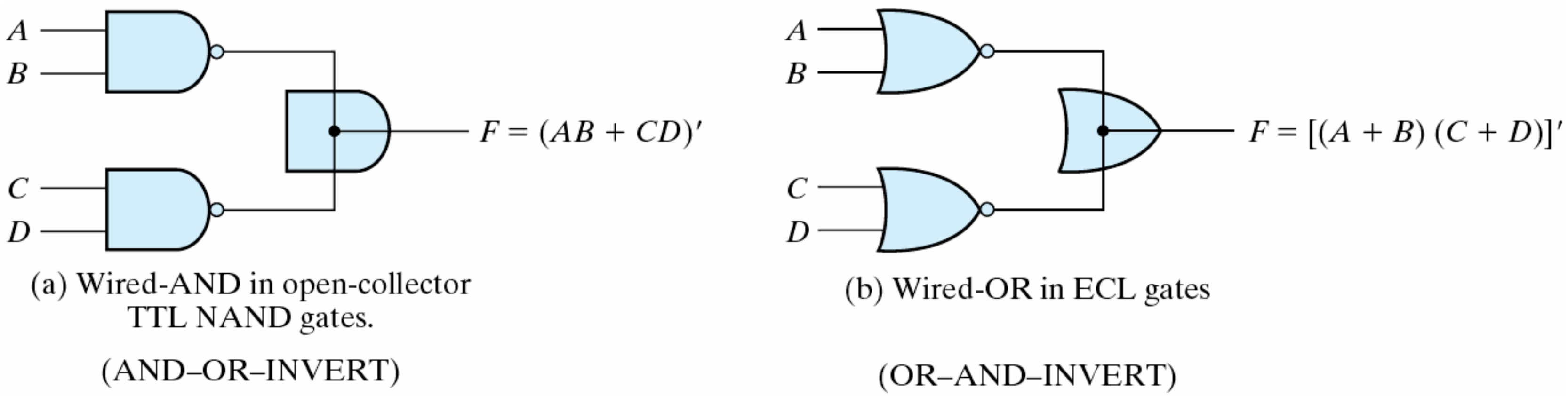
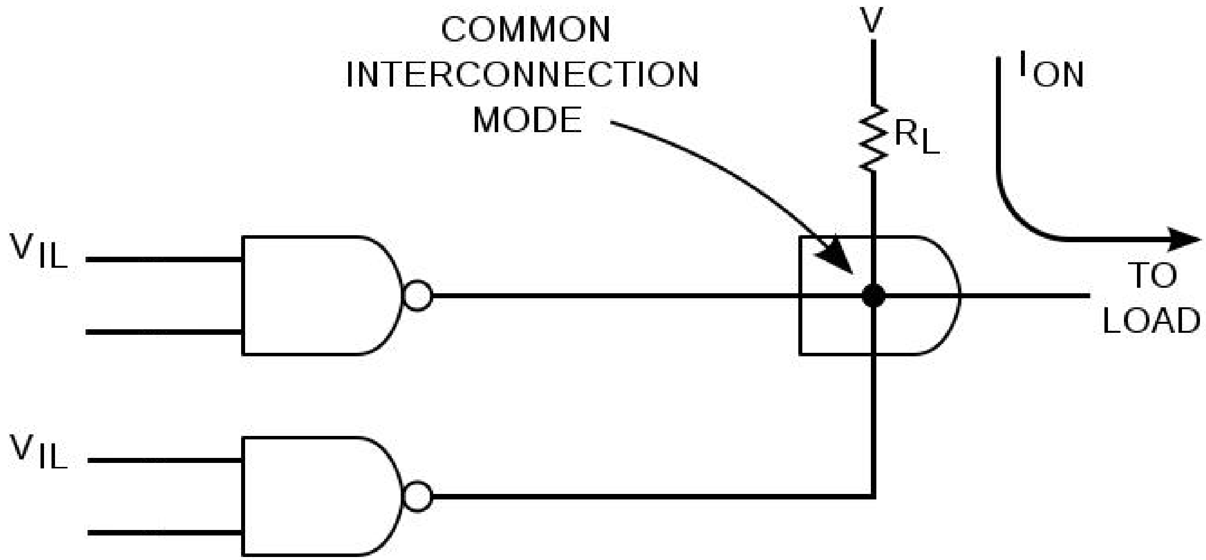
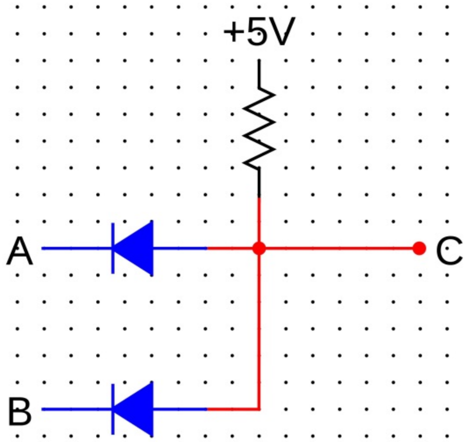


Figure 26 Wired Logic

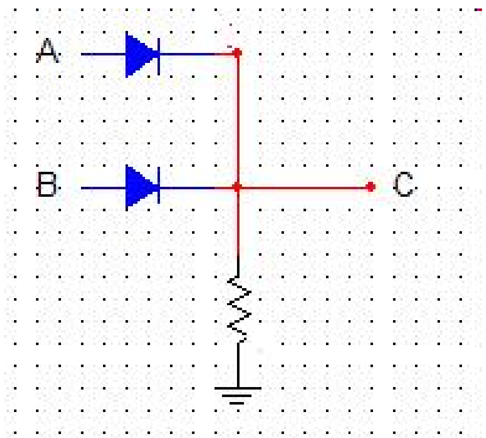
Wired Logic



Wired-AND Logic



Wired-AND Logic



Wired-OR 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
    - NAND-AND and AND-NOR = AND-OR-INVERT
    - NOR-OR and OR-NAND = OR-AND-INVERT

AND-OR-Invert Implementation

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

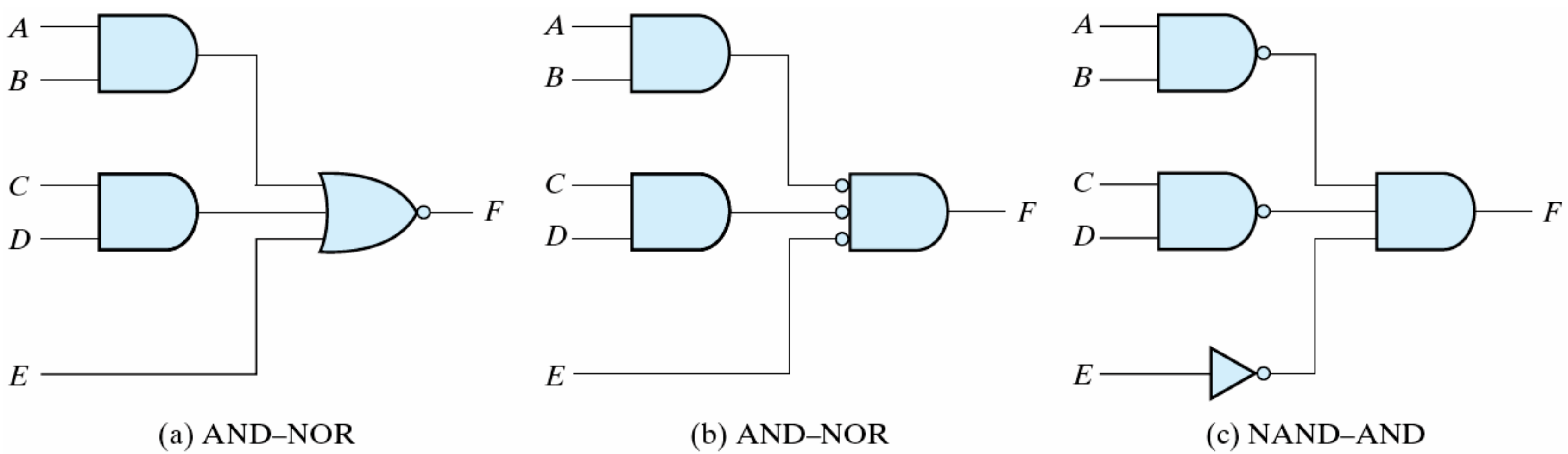


Figure 27 AND-OR-INVERT circuits, $F = (AB + CD + E)'$

OR-AND-Invert Implementation

▣ OR-AND-INVERT (OAI) Implementation

- OR-NAND = NOR-OR = OAI
- $F = ((A+B)(C+D)E)'$ (product of sums + Inverter)
- $F' = (A+B)(C+D)E$ (product of sums)

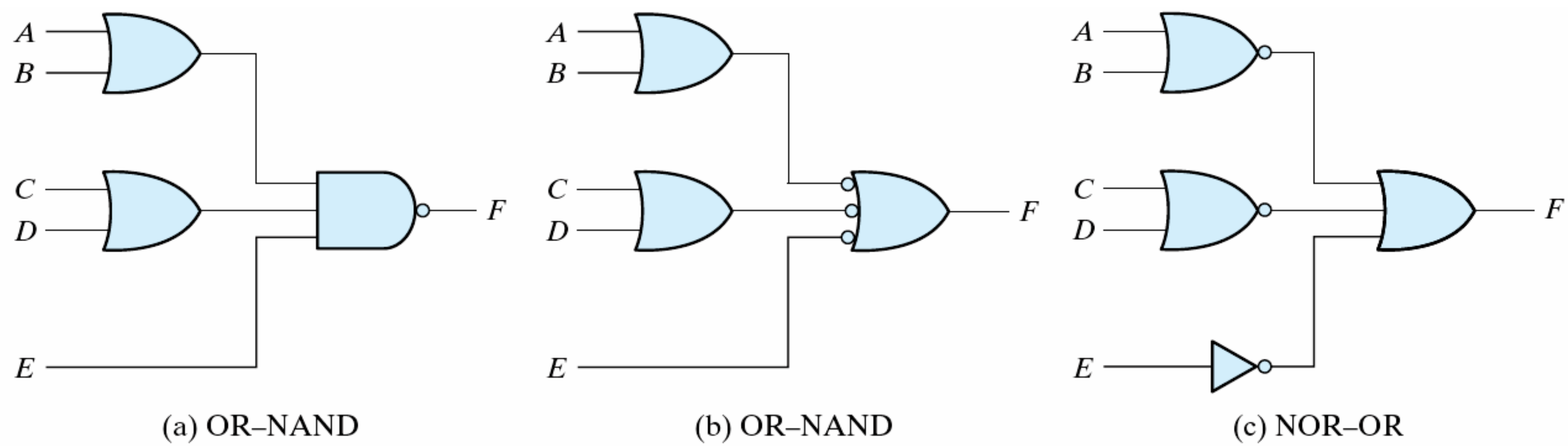


Figure 28 OR-AND-INVERT circuits, $F = ((A+B)(C+D)E)'$

Tabular Summary and Examples

Table 3.2
Implementation with Other Two-Level Forms

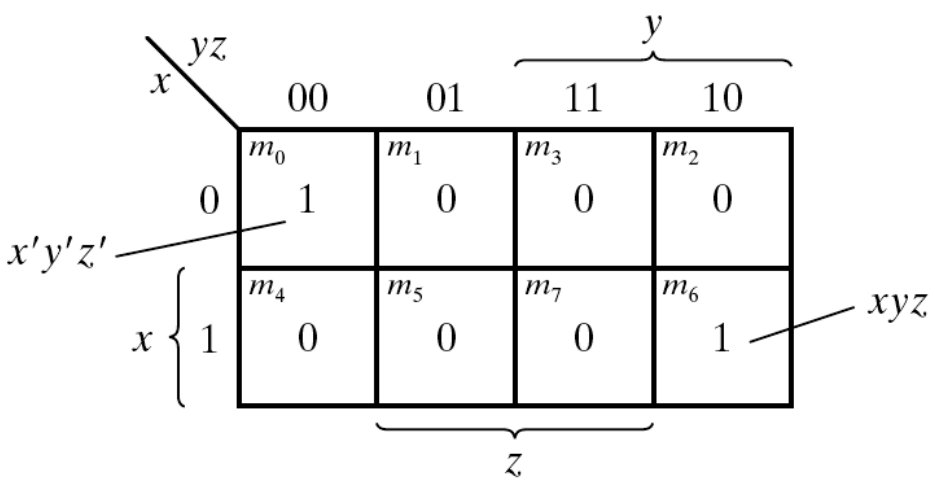
Equivalent Nondegenerate Form (a)	Equivalent Nondegenerate Form (b)*	Implements the Function	Simplify F' into	To Get an Output of
AND–NOR	NAND–AND	AND–OR–INVERT	Sum-of-products form by combining 0's in the map.	F
OR–NAND	NOR–OR	OR–AND–INVERT	Product-of-sums form by combining 1's in the map and then complementing.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

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

STUDENT

Tabular Summary and Examples

- **Example 10: Implement the following function with (a) AND-NOR (b) NAND-AND (c) OR-NAND (d) NOR-OR forms**



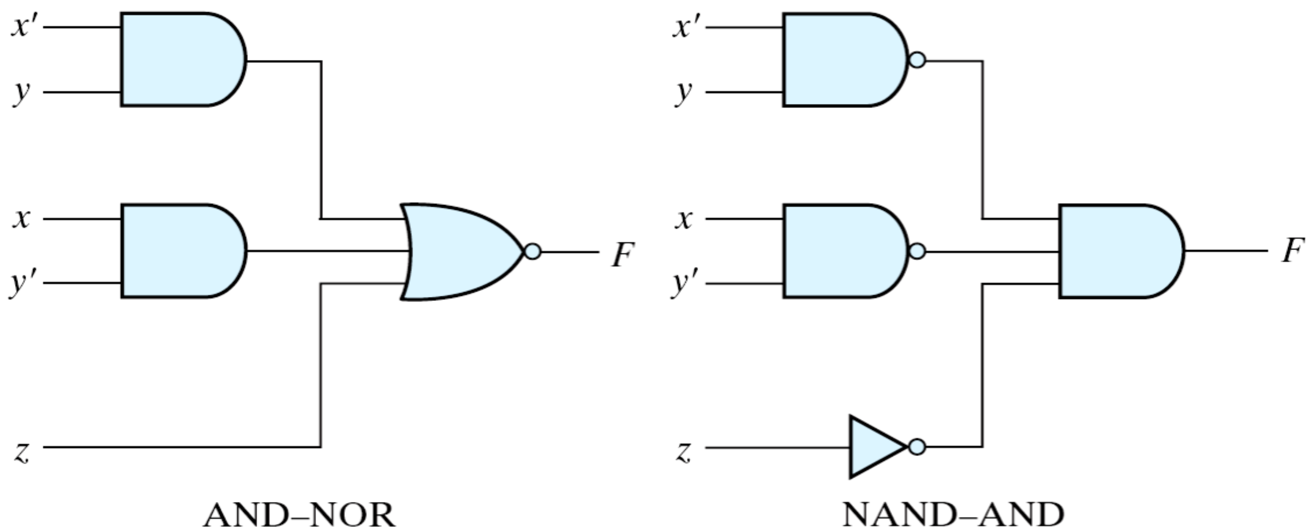
$F = x'y'z' + xyz'$

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

(a) Map simplification in sum of products

Tabular Summary and Examples

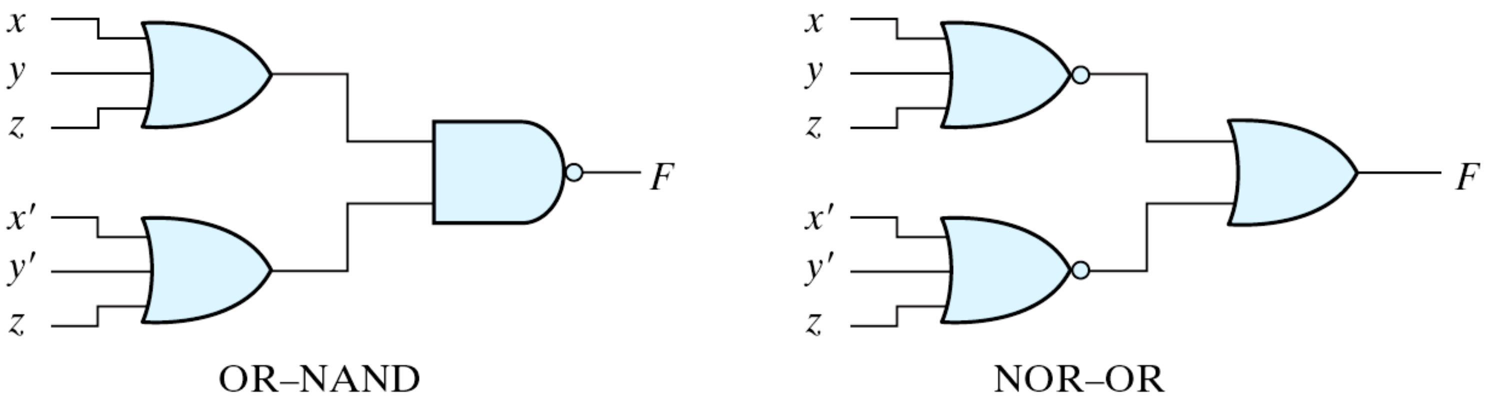
- **(a) AND-NOR (b) NAND-AND**
  - $F' = x'y+xy'+z$ **(*F'*: sum of products)**
  - $F = (x'y+xy'+z)'$ **(*F*: AOI implementation)**



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

Tabular Summary and Examples

- **(c) OR-NAND (d) NOR-OR forms**
  - $F = x'y'z' + xyz'$ (*F*: sum of products)
  - $F' = (x+y+z)(x'+y'+z)$ (*F'*: product of sums)
  - $F = ((x+y+z)(x'+y'+z))'$ (F: OAI)



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

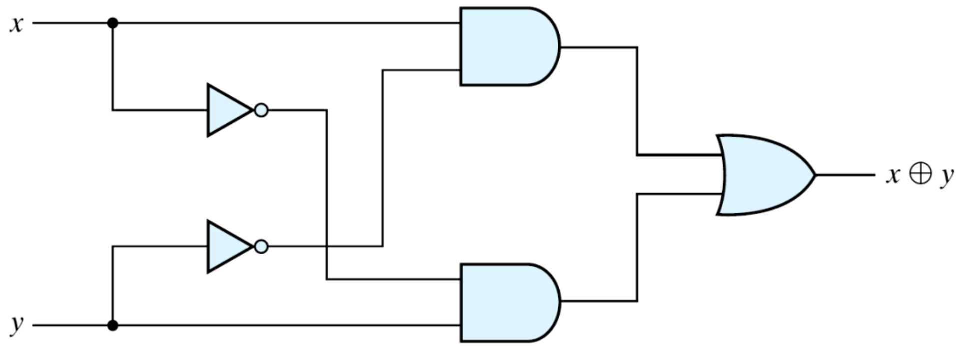
Exclusive-OR Function

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

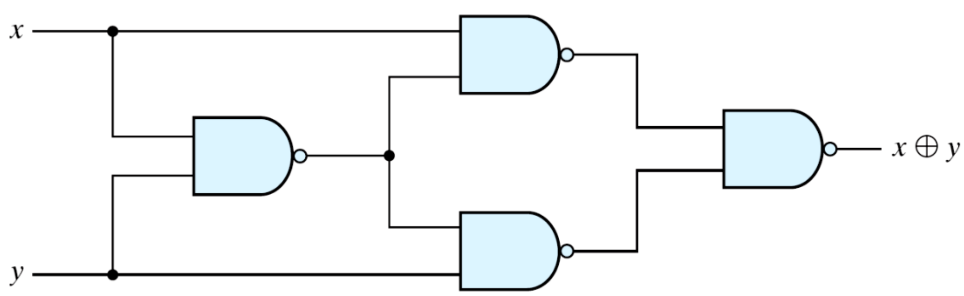
Exclusive-OR Implementations

Implementations

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



(a) With AND–OR–NOT gates

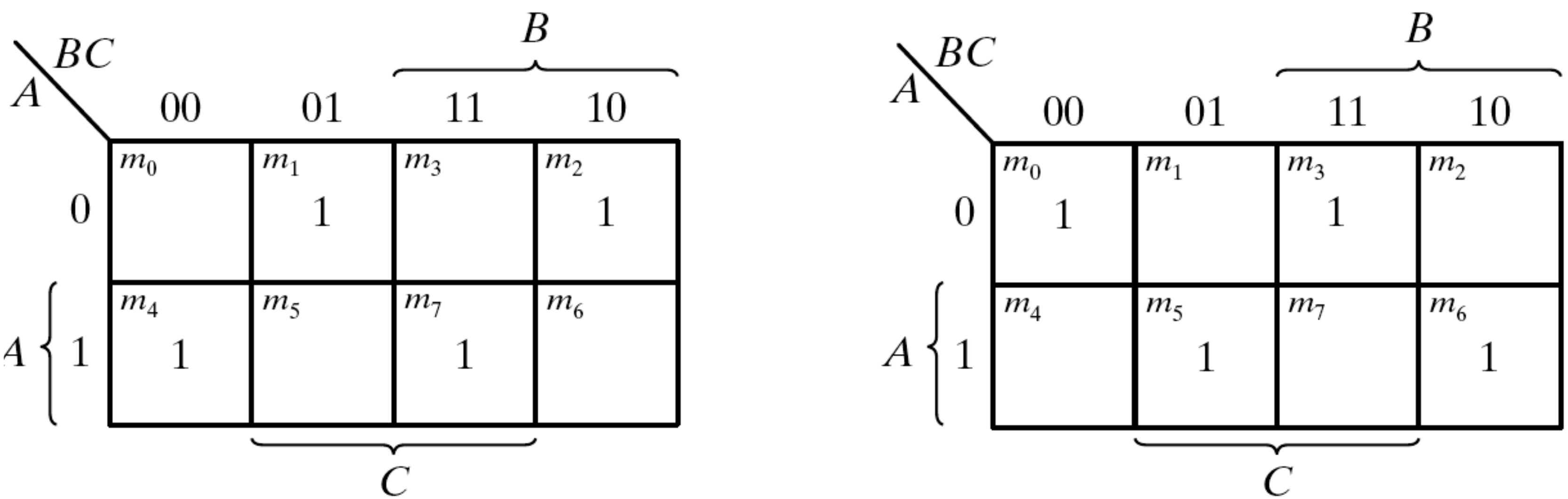


(b) With NAND gates

Figure 30 Exclusive-OR Implementations

Odd/Even 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 → an odd number of 1's, then *F* = 1**
- **XNOR is an even function → an even number of 1's, then *F* = 1**



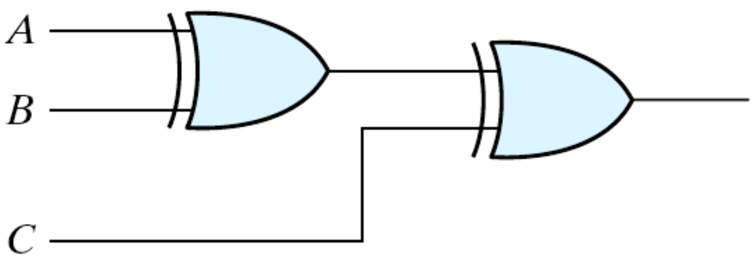
(a) Odd function $F = A \oplus B \oplus C$

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

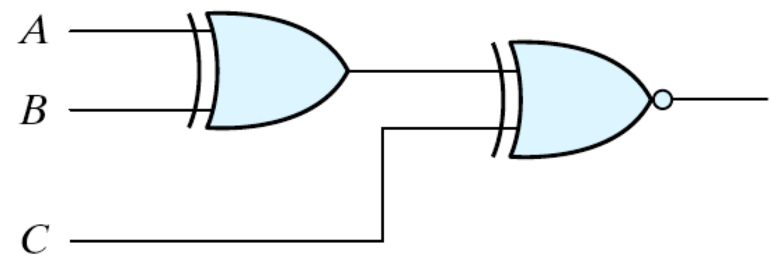
Figure 31 Map for a Three-variable Exclusive-OR Function

XOR and XNOR

- **Logic diagram of odd and even functions**



(a) 3-input odd function



(b) 3-input even function

Figure 32 Logic Diagram of Odd and Even Functions

Four-variable Exclusive-OR function

- **Four-variable Exclusive-OR function**
  - $A \oplus B \oplus C \oplus D = (AB'+A'B) \oplus (CD'+C'D) = (AB'+A'B)(CD+C'D')+(AB+A'B')(CD'+C'D)$

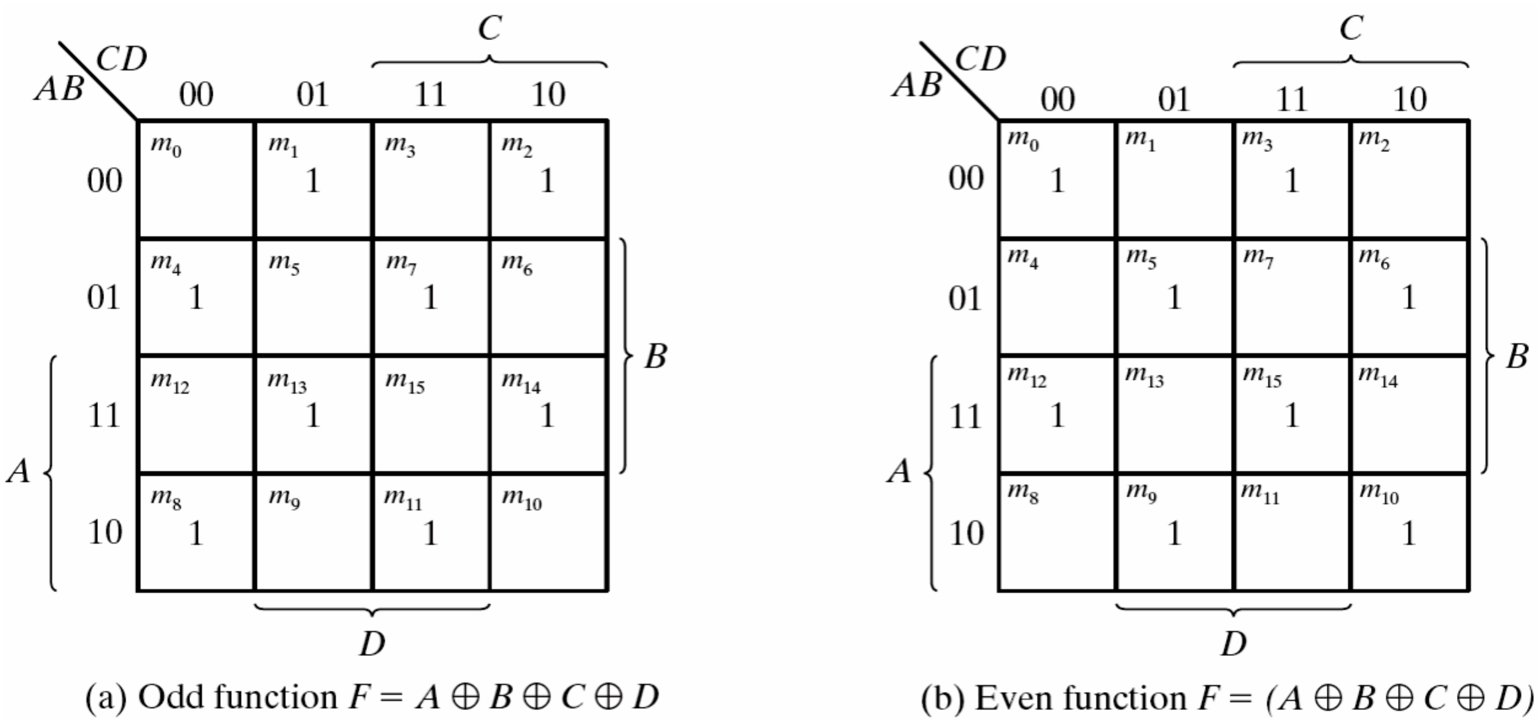


Figure 33 Map for a Four-variable Exclusive-OR Function

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

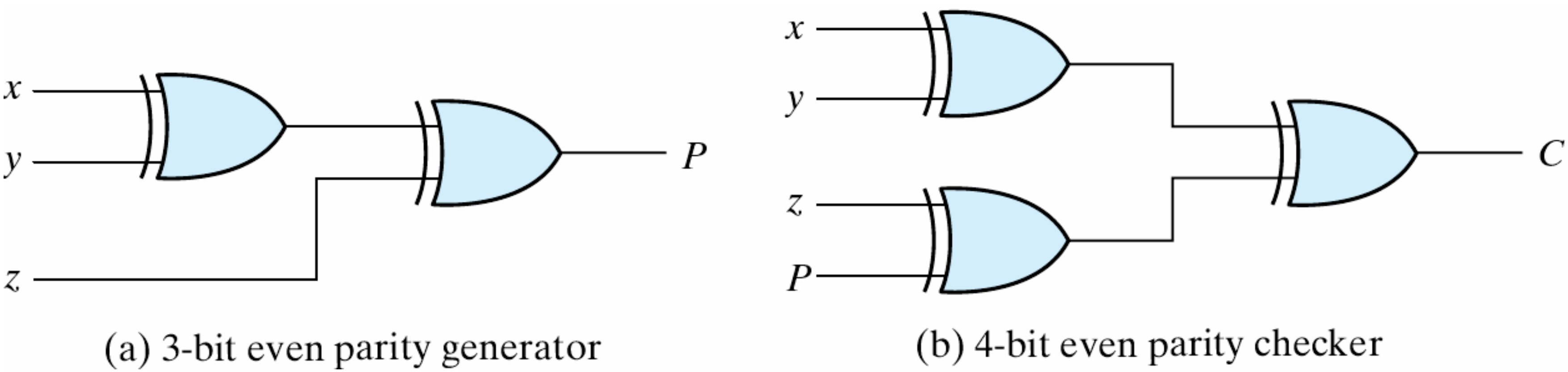


Figure 34 Logic Diagram of a Parity Generator and Checker

Parity Generation and Checking

Table 3.3
Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	y	z	P
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 3.4
Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
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