CS204: 數位系統設計

Gate-Level Minimization

Outline of Chapter 3

- 3.1 Introduction
- 3.2 The Map Method
- 3.3 Four-Variable Map
- 3.4 Product-of-Sums Simplification
- 3.5 Don’t-Care Conditions
- 3.6 NAND and NOR Implementation
- 3.7 Other Two-Level Implementation
- 3.8 Exclusive-OR Function
- 3.9 Hardware Description Language

3-1 Introduction (p.89)

- **Gate-level minimization refers to the design task of finding an optimal gate-level implementation of Boolean functions describing a digital circuit**
- **The complexity of the digital logic gates**
  - **The complexity of the algebraic expression**
- **Algebraic approaches: lack specific rules**

3-2 The Map Method (p.89, 90)

- **The Karnaugh map**
  - A simple and straight forward procedure
  - A pictorial form of a truth table
  - Applicable if the # of variables ≤ 6
- **A diagram made up of squares**
  - Each square represents one minterm

Two-Variable Map (p.90)

- **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): $x + y = x'y + xy' + xy = m_1 + m_2 + m_3$**

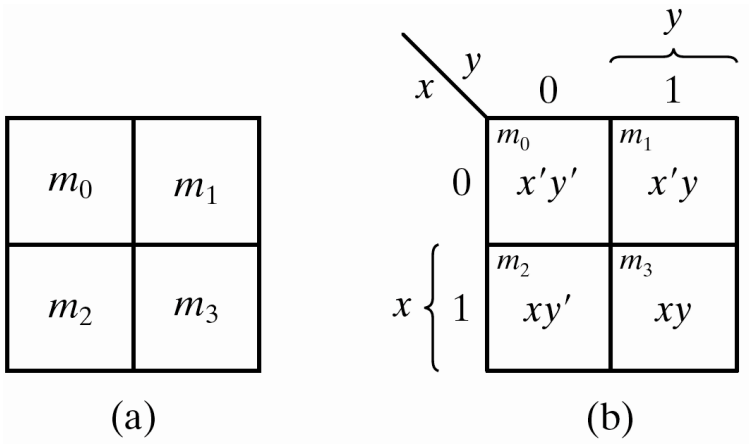


Figure 3.1 Two-variable Map

A Three-Variable Map (p.91)

- **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
    - » $m_5 + m_7 = xy'z + xyz = xz\,(y' + y) = xz$

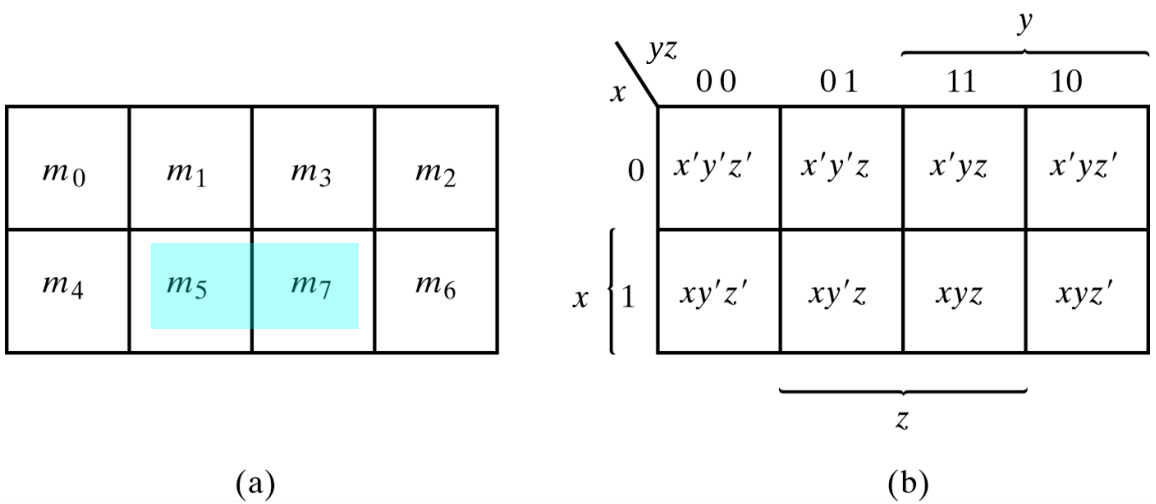


Figure 3.3 Three-variable Map

A Three-Variable Map (p.91, 92)

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

y (11, 10); z (01, 11)

(b)

Fig. 3-3 Three-variable Map

Example 3.1 (p.92)

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

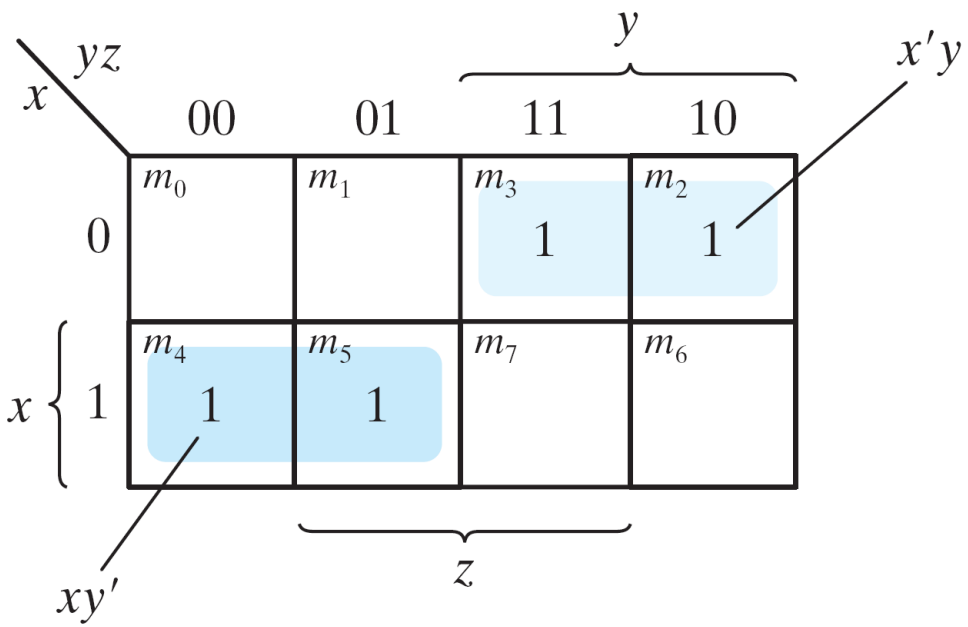


Figure 3.4 Map for Example 3.1, ***F*(*x*, *y*, *z*) = Σ(2, 3, 4, 5) = *x'y* + *xy'***

Example 3.2 (p.93)

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

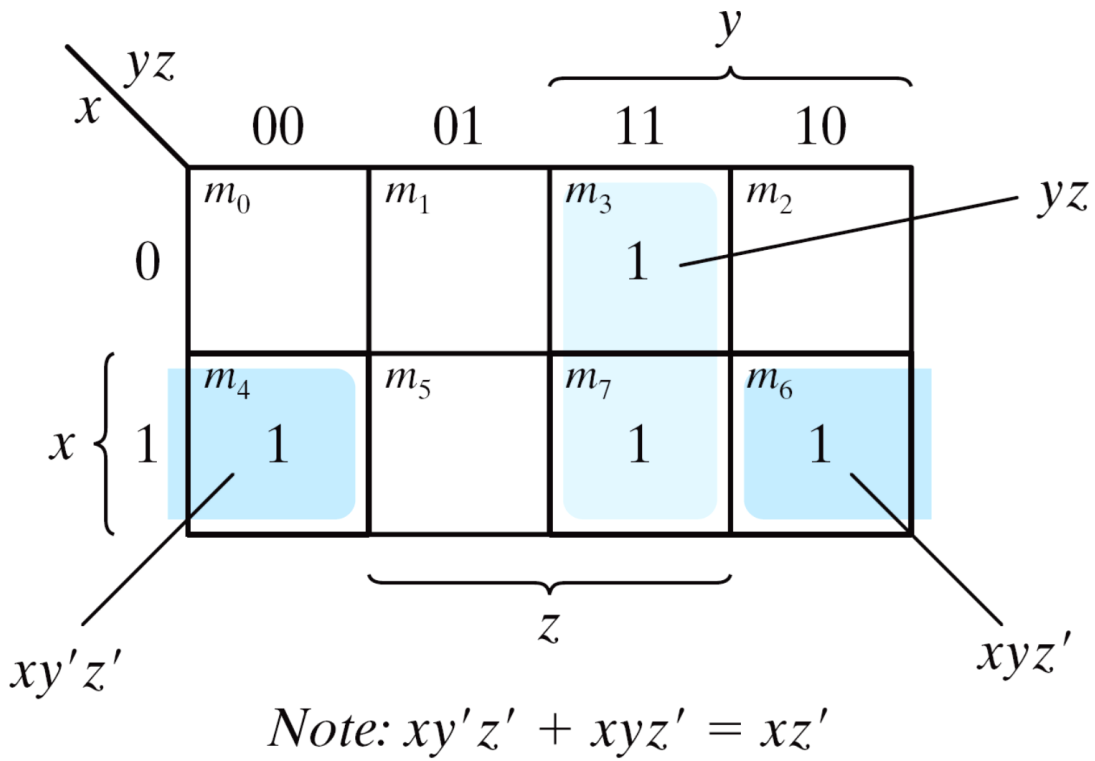


Figure 3.5 Map for Example 3-2; **$F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$**

Four adjacent Squares (p.94)

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

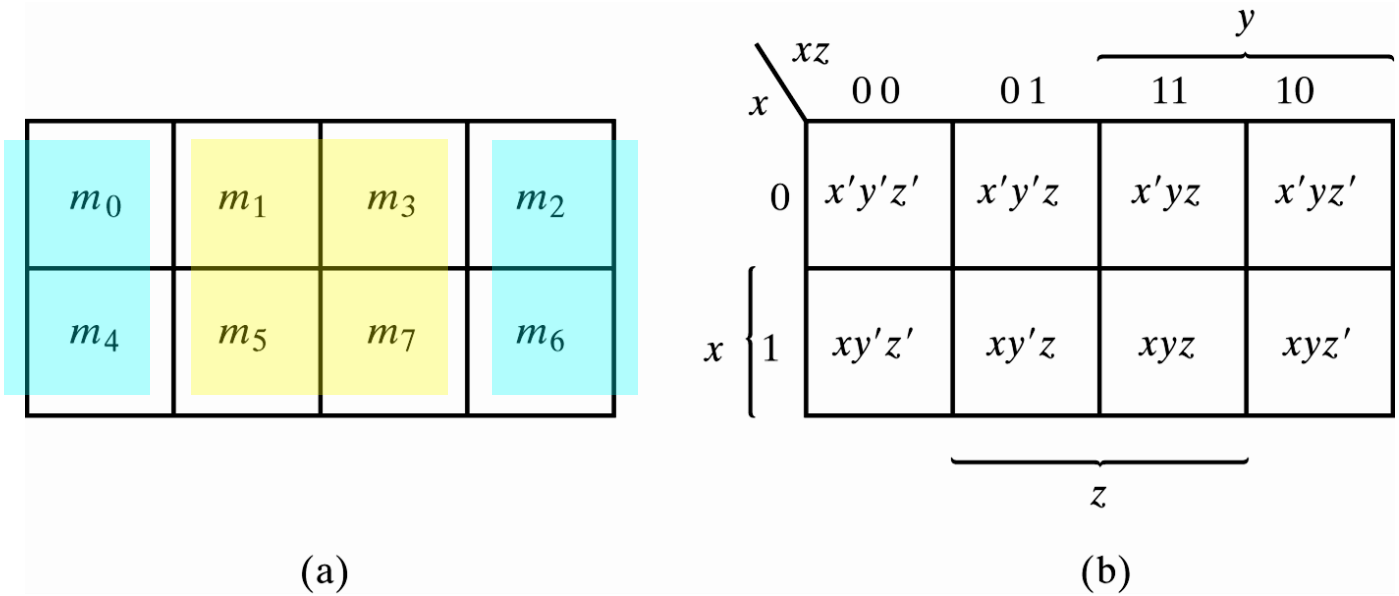


Figure 3.3 Three-variable Map

Example 3.3 (p.94)

- **Example 3.3: simplify *F(x, y, z)* = Σ(0, 2, 4, 5, 6)**
  - ***F*(*x*, *y*, *z*) = Σ(0, 2, 4, 5, 6) = *z*′+ *xy*′**

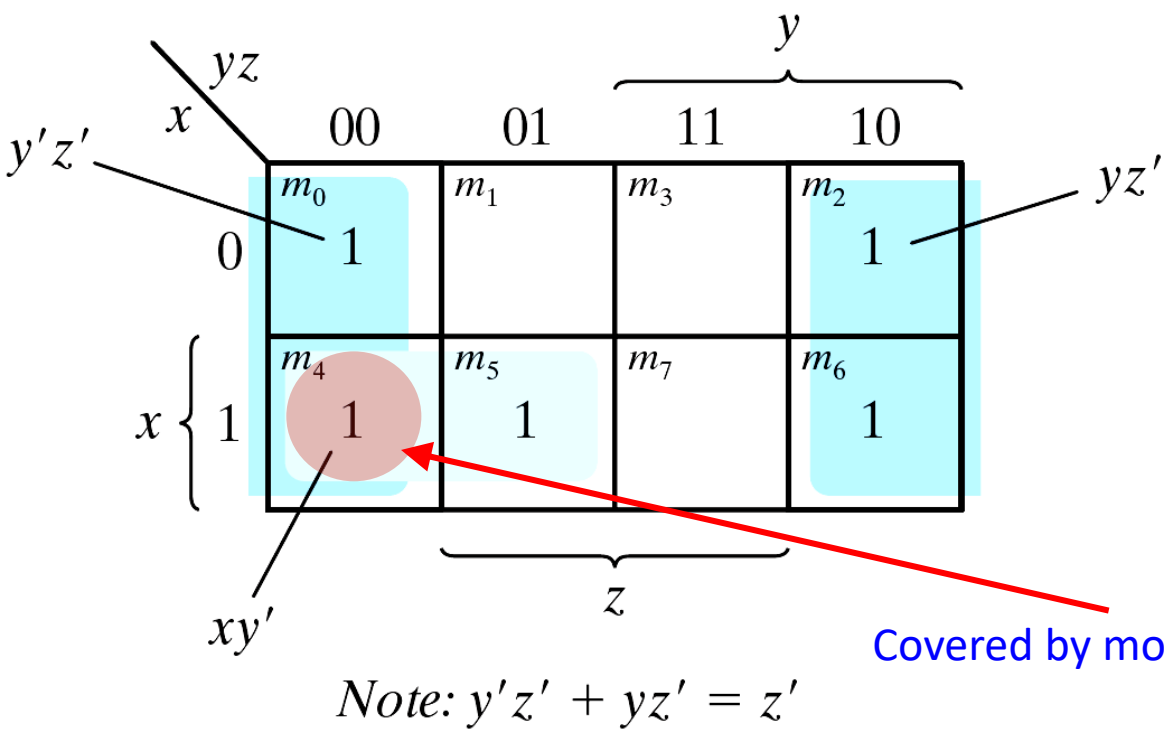


Figure 3.6 Map for Example 3-3, ***F*(*x*, *y*, *z*) = Σ(0, 2, 4, 5, 6) = *z*′ + *xy*′**

Example 3.4 (p.95)

- **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**

    F*(*A*, *B*, *C*) = Σ(1, 2, 3, 5, 7) = *C* + *A'B

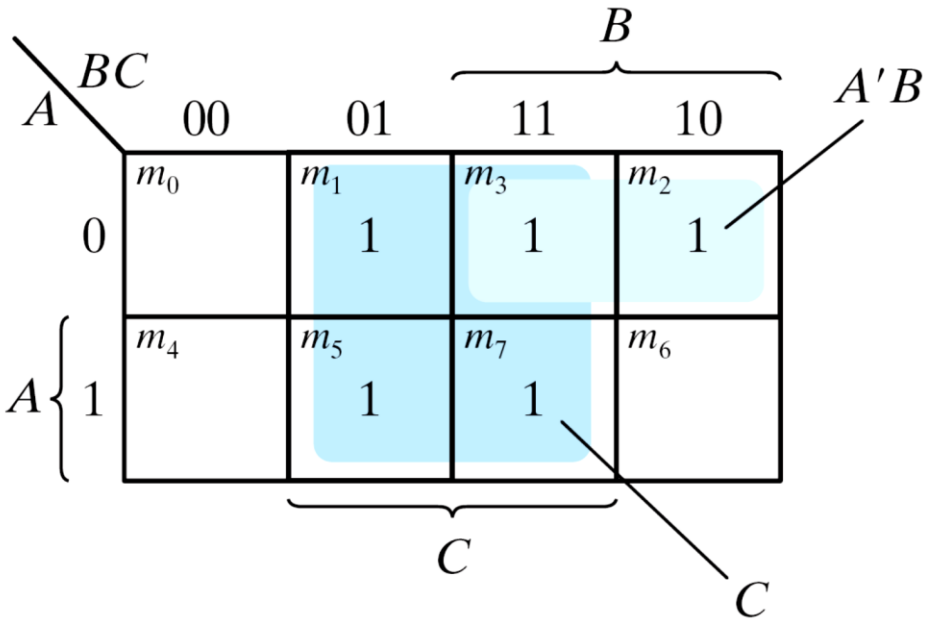


Figure 3.7 Map for Example 3.4, ***A'C + A'B + AB'C + BC = C + A'B***

3.3 Four-Variable Map (p.96)

- **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 \ 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	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

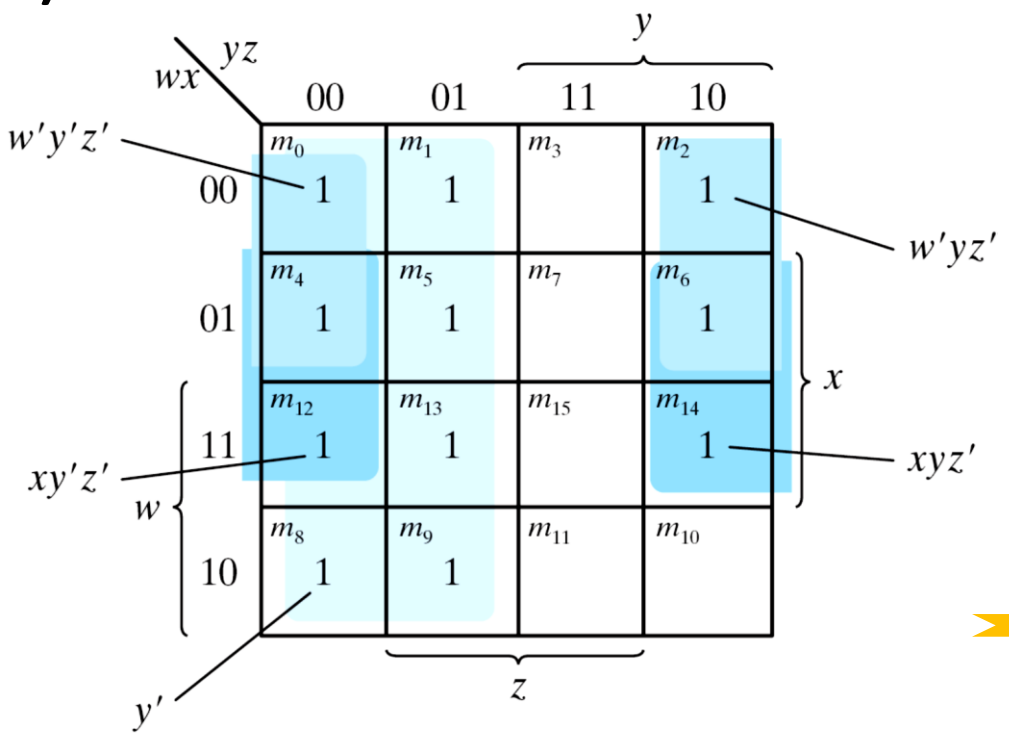
y; x; w; z

(b)

Figure 3.8 Four-variable Map

Example 3.5 (p.97)

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



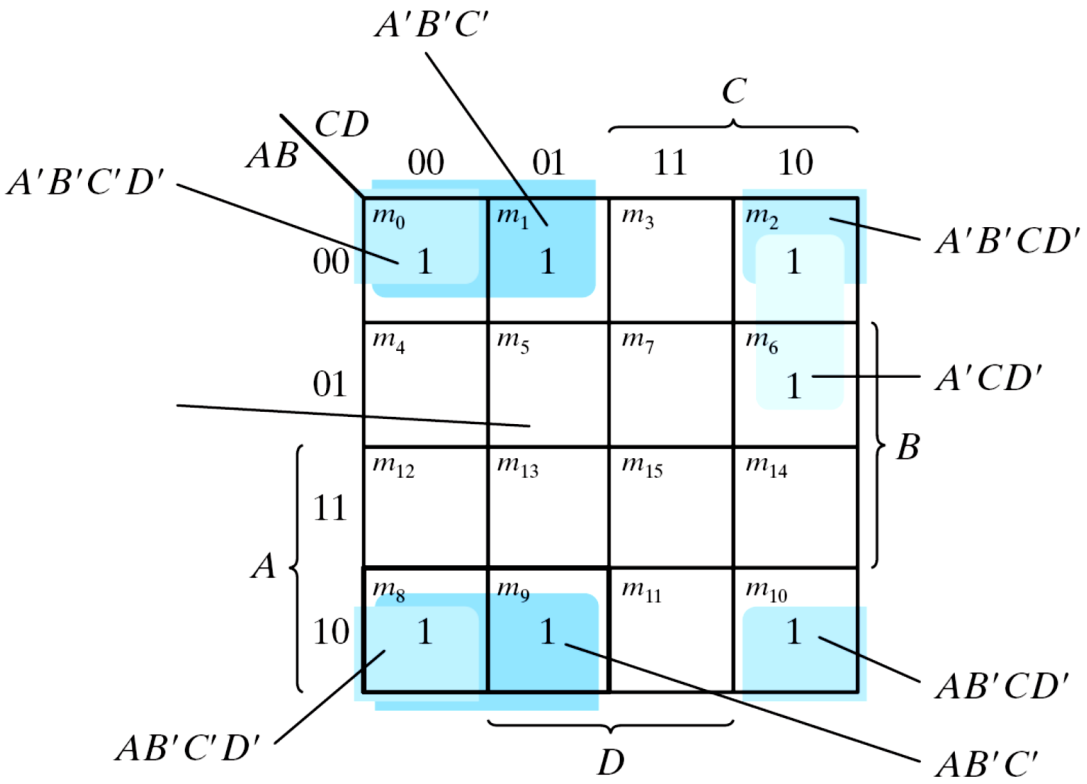
Note: $w'y'z' + w'yz' = w'z'$
$xy'z' + xyz' = xz'$

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

Figure 3.9 Map for Example 3-5; ***F(w, x, y, z) = Σ(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w' z' +xz'***

Example 3.6 (p.97, 98)

- **Example 3.6: simplify *F = A′B′C′ + B′CD′ + A′BCD′ + 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′C′D′ + AB′C′ = B′D′ + B′C′ + A′CD′***

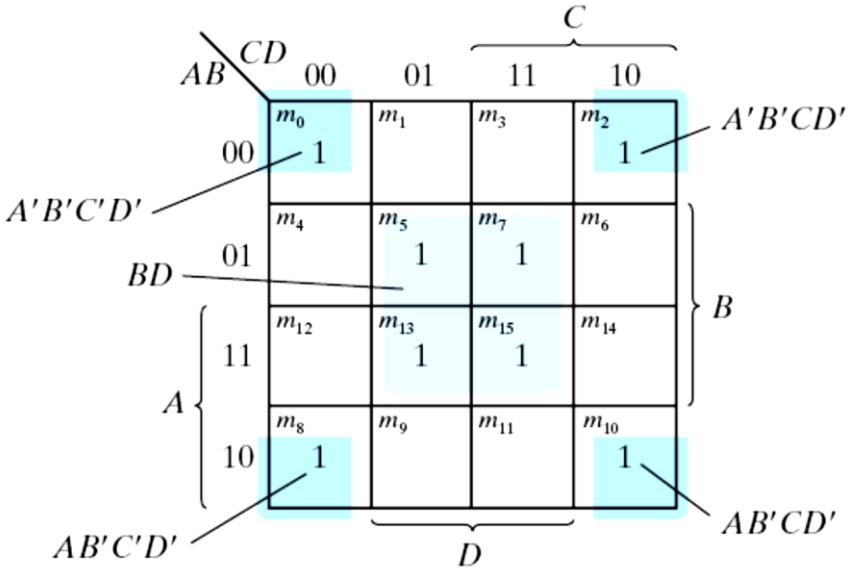
Prime Implicants (p.98)

- **K-map-based minimization**
  - **All the minterms are covered**
  - **Minimize the number of terms**
- **Prime Implicants**
  - **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 prime implicant in which there exists a minterm which is only covered by the prime implicant**
  - **The essential P.I. must be included**
    - **So, select them first**

Prime Implicants (p.99)

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

    $= BD+B'D'+B'C+AD = BD+B'D'+B'C+AB'$

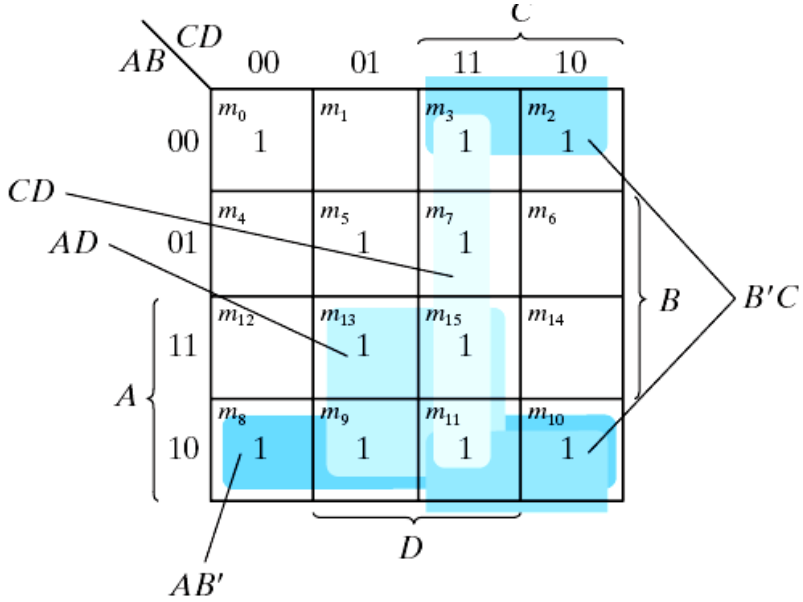


Note: $A'B'C'D' + A'B'CD' = A'B'D'$

$AB'C'D' + AB'CD' = AB'D'$

$A'B'D' + AB'D' = B'D'$

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



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

Figure 3.11 Simplification Using Prime Implicants

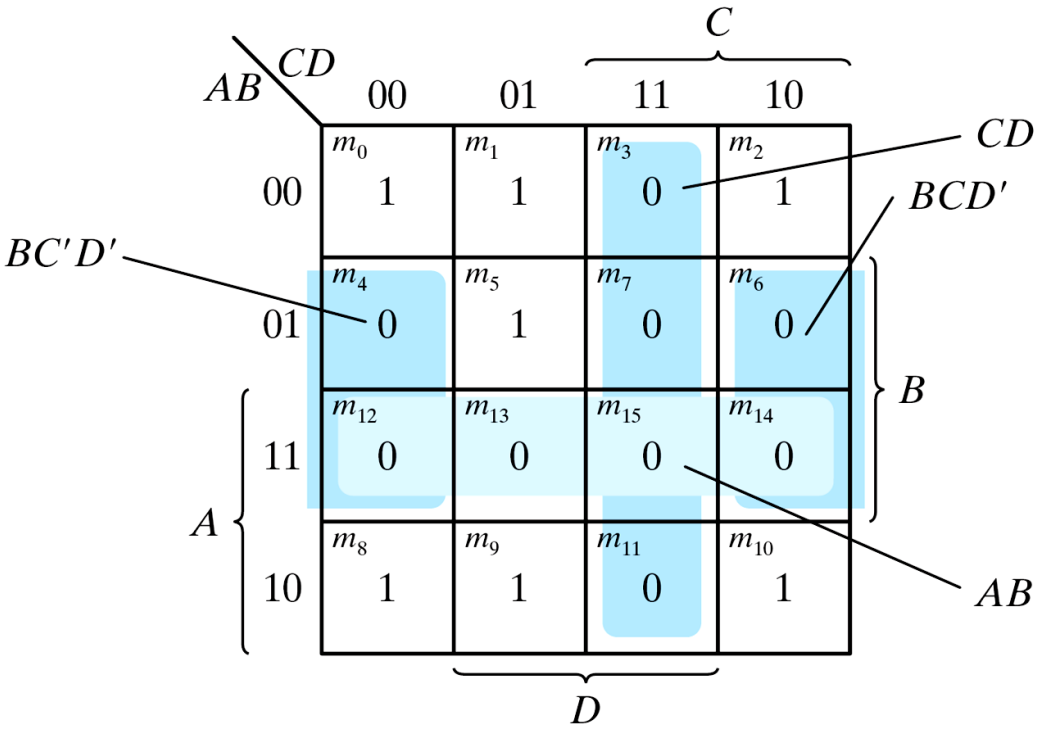
3-4 Product of Sums Simplification
(p.100)

- **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 (p.101)

- **Example 3.7: simplify $F = \Sigma(0, 1, 2, 5, 8, 9, 10)$ into (a) sum-of-products form, and (b) product-of-sums form:**



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

a) ***F(A, B, C, D)= S(0, 1, 2, 5, 8, 9, 10) = B'D'+B'C'+A'C'D***

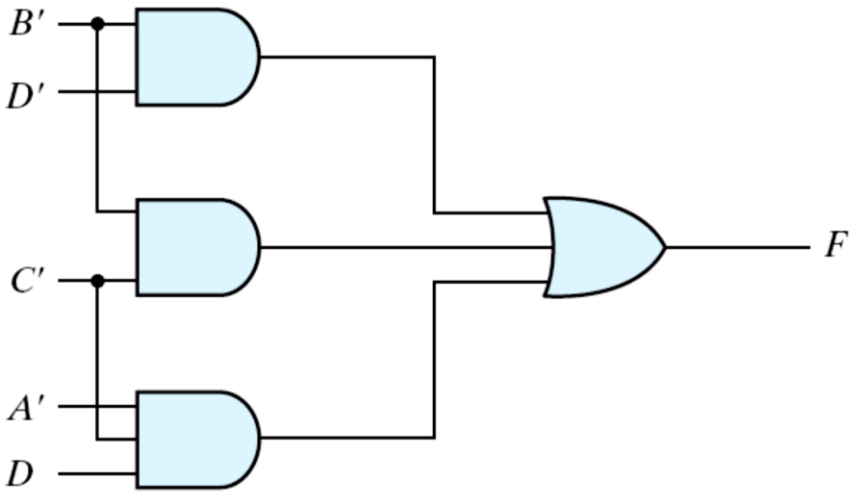
b) ***F' = AB+CD+BD'***

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

Figure 3.12 Map for Example 3.7, F(*A*, *B*, *C*, *D*)= Σ(0, 1, 2, 5, 8, 9, 10) = *B'D'+B'C'+A'C'D*

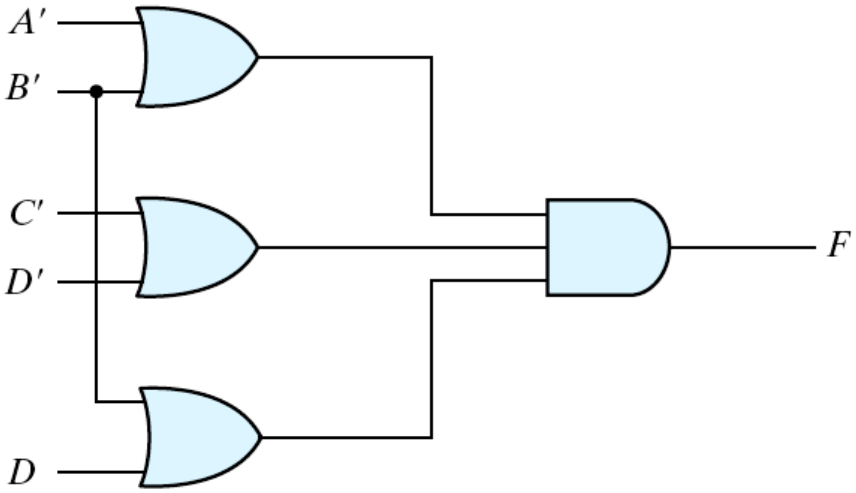
Example 3.7 (cont.) (p.102)

- **Gate implementation of the function of Example 3.7**



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

Sum-of products form



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

Product-of sums form

Figure 3.13 Gate Implementation of the Function of Example 3.7

3-5 Don't-Care Conditions (p.104)

- 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: 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 3.8 (cont.) (p.104, 105)

- ***F = yz + w'x'; 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**

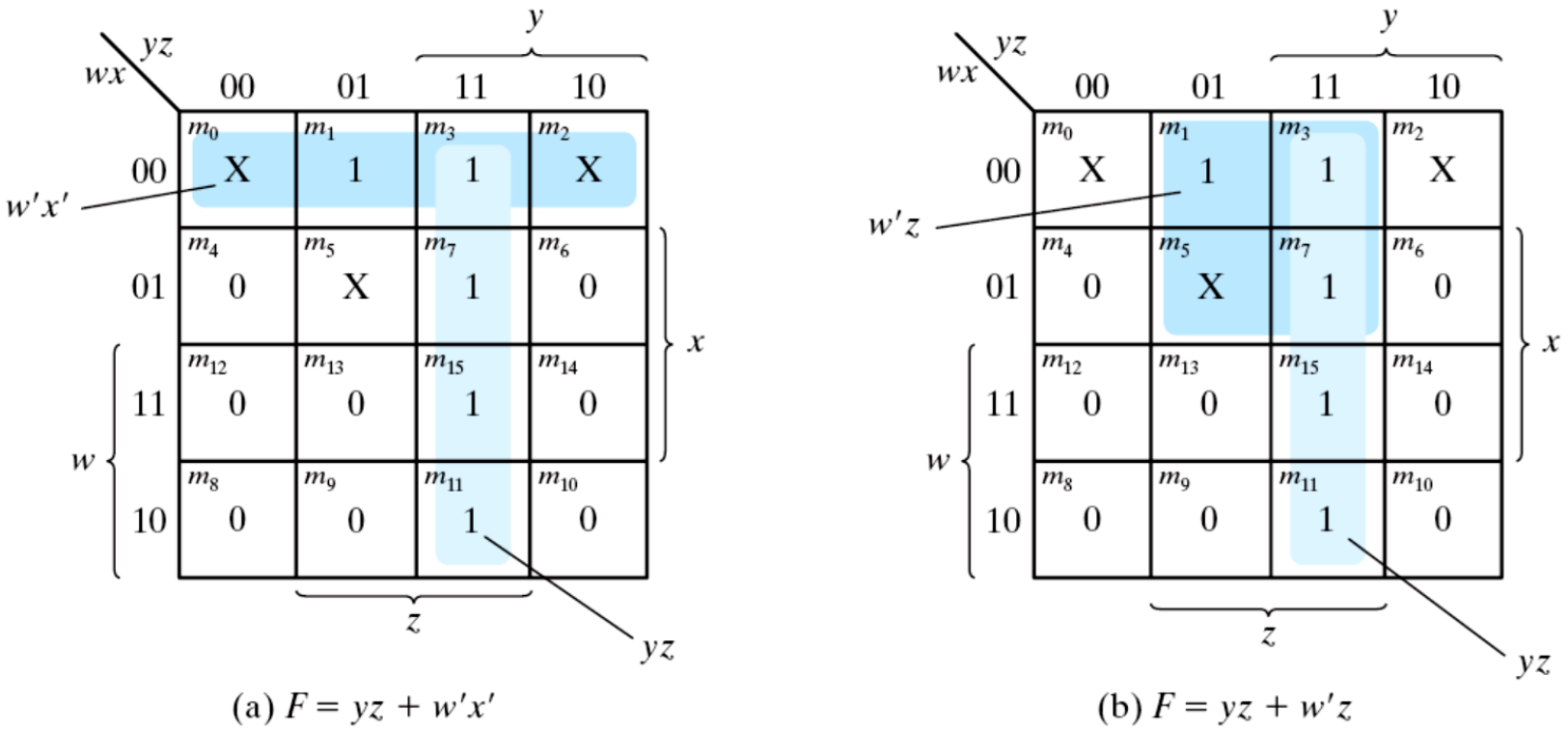


Figure 3.15 Example with don't-care Conditions

3-6 NAND and NOR Implementation (p.106)

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

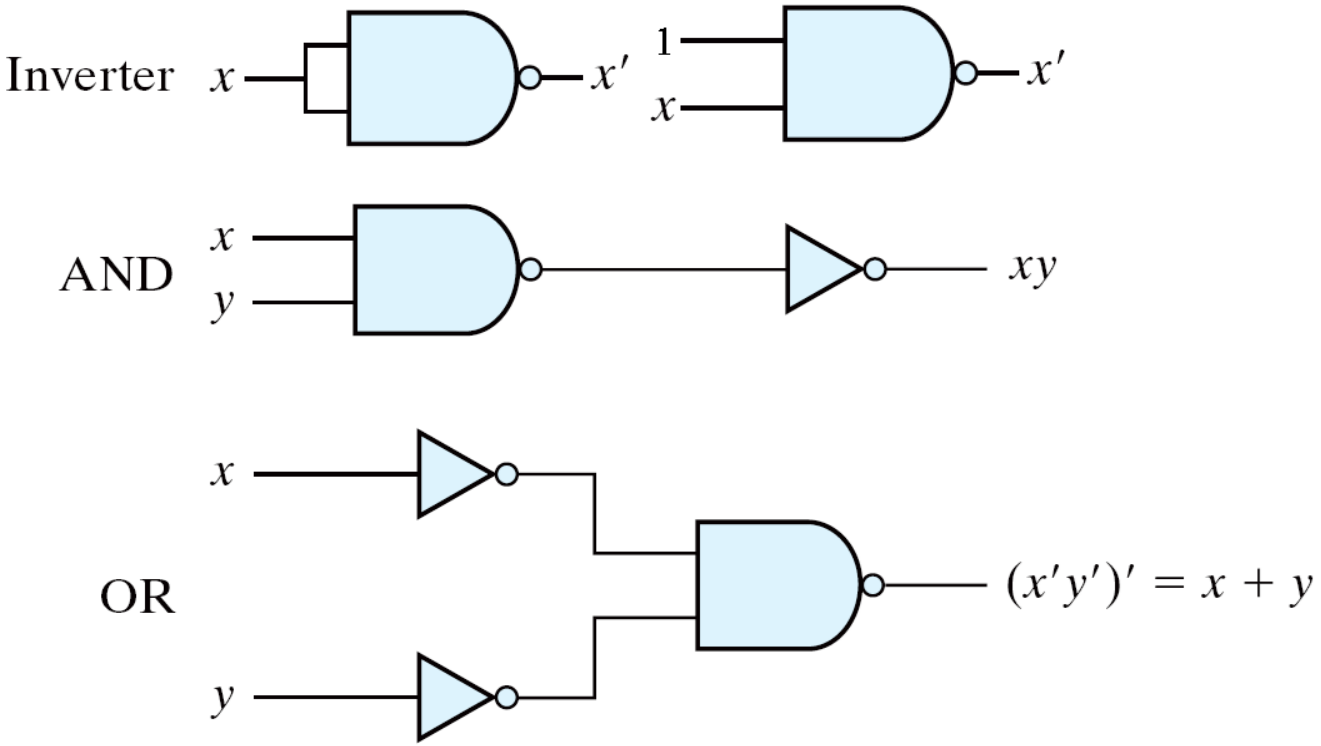
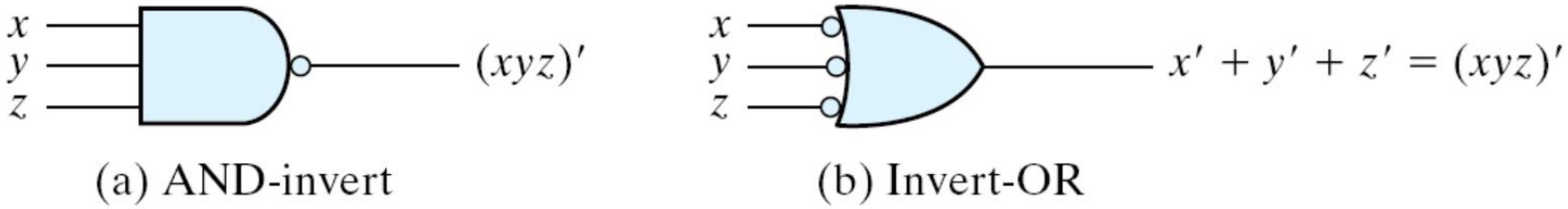


Figure 3.16 Logic Operations with NAND Gates

NAND Gate (p.107)

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



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

Figure 3.17 Two Graphic Symbols for NAND Gate

Two-level Implementation (p.107, 108)

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

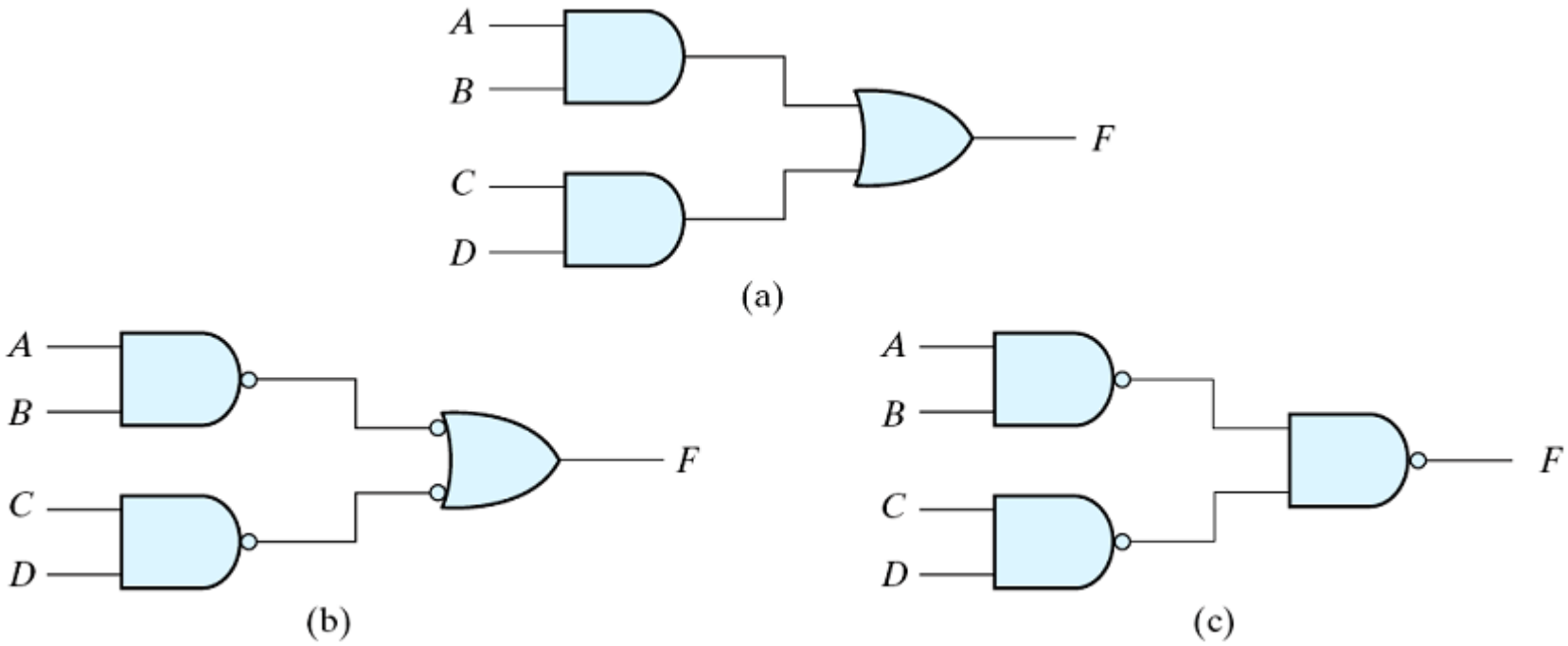
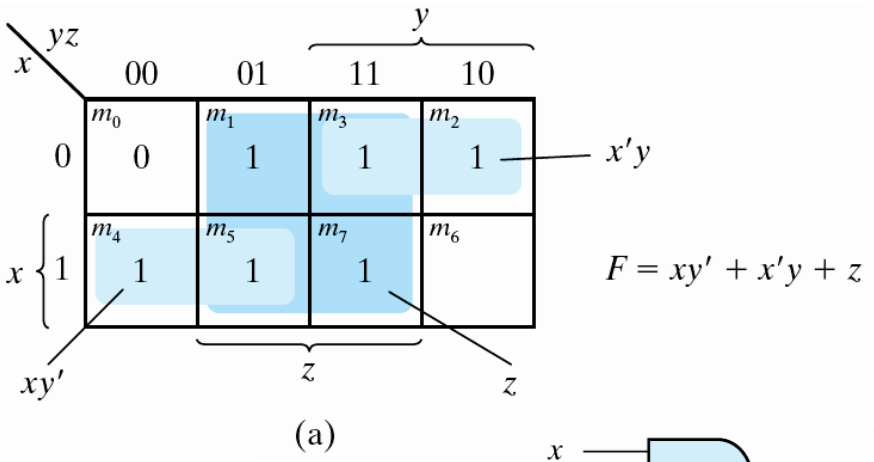


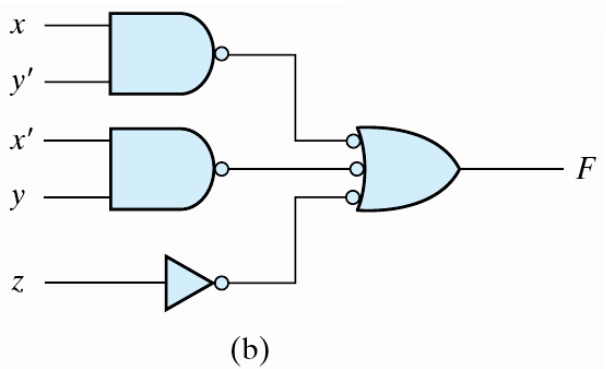
Figure 3.18 Three ways to implement $F = AB + CD$

Example 3.9 (p.108, 109)

- **Example 3-9: implement *F*(*x*, *y*, *z*) =**

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





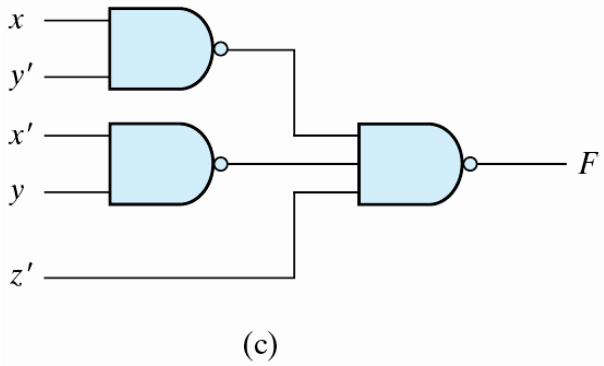


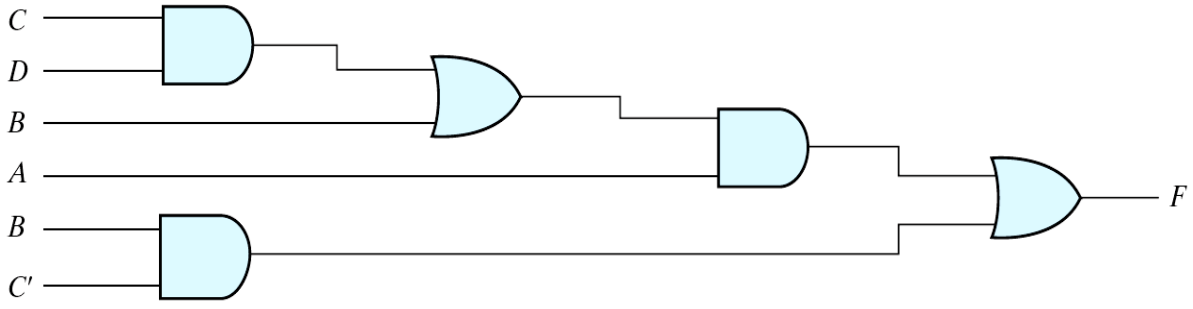
Figure 3.19 Solution to Example 3-9

Procedure with Two Levels NAND (p.109)

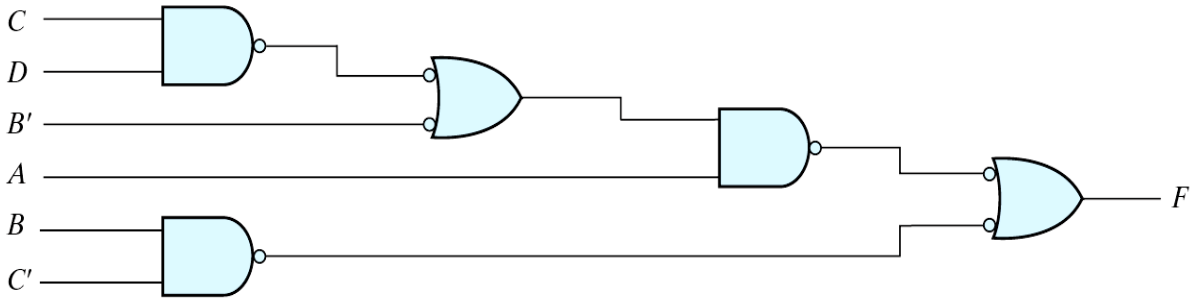
- **The procedure**
  - **Simplified in the form of sum of products;**
  - **A NAND gate for each product term; the inputs to each NAND gate are the literals of the term (the first level);**
  - **A single NAND gate for the second sum term (the second level);**
  - **A term with a single literal requires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

Multilevel NAND Circuits (p.109, 110)

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



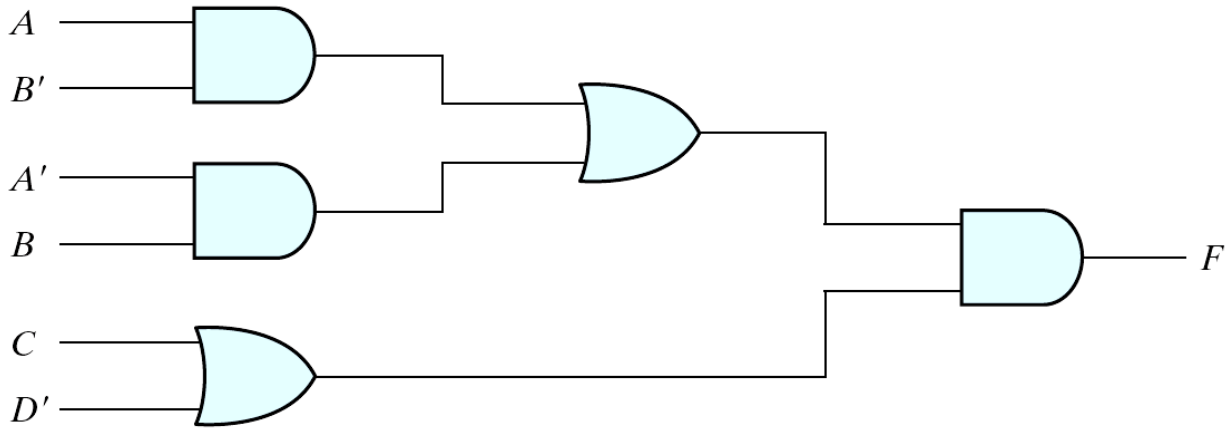
(a) AND–OR gates **Alternating levels of AND and OR gates**



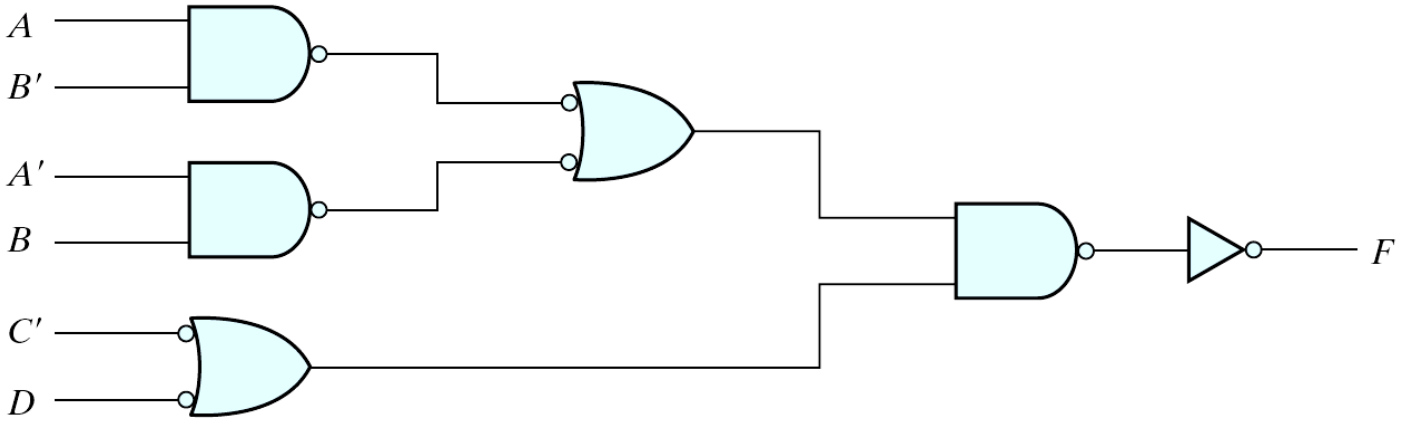
(b) NAND gates

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

NAND Implementation (p.110, 111)



(a) AND–OR gates



(b) NAND gates

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

NOR Implementation (p.111, 112)

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

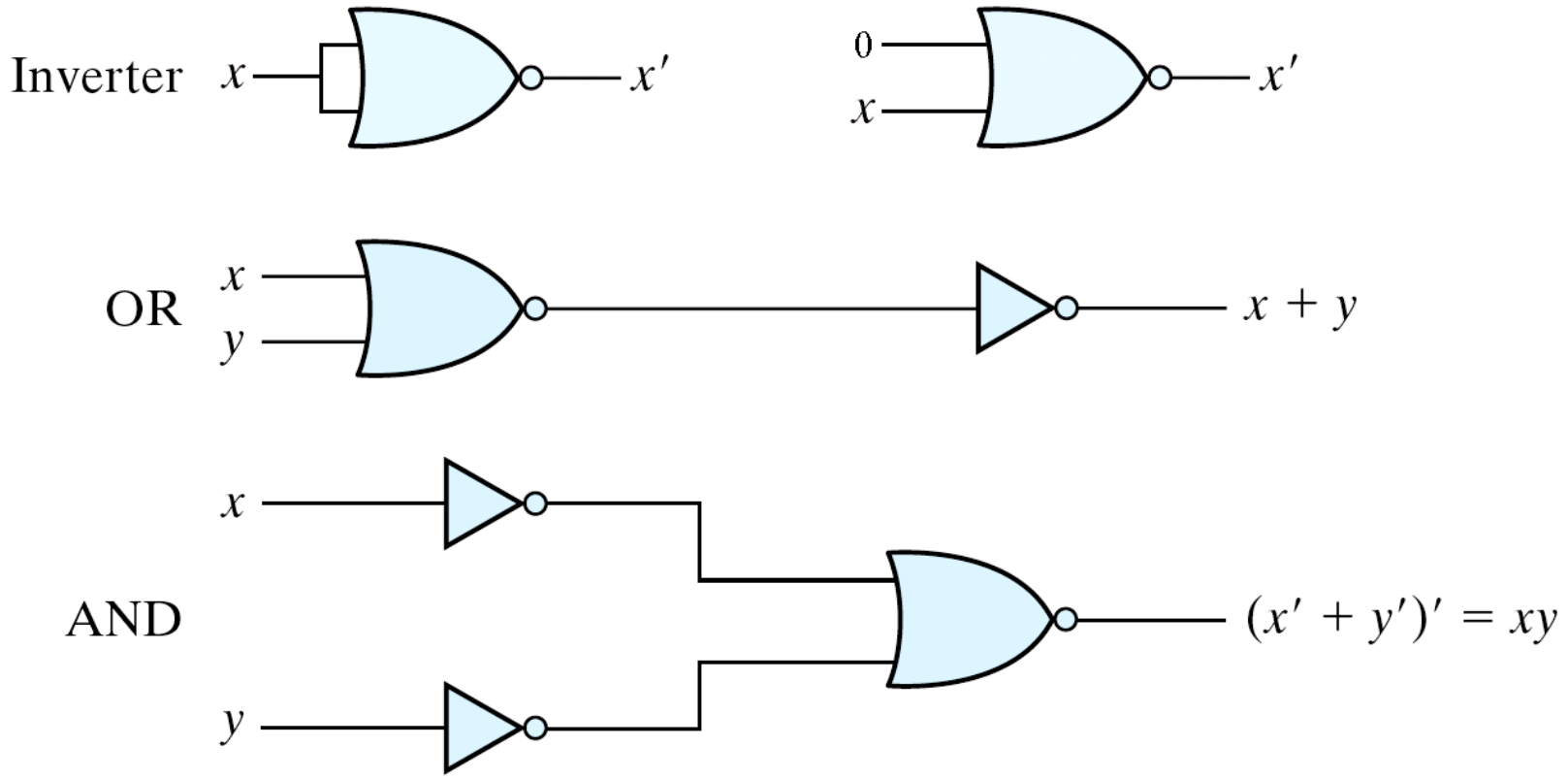


Figure 3.22 Logic Operation with NOR Gates

Two Graphic Symbols for a NOR Gate
(p.112, 113)

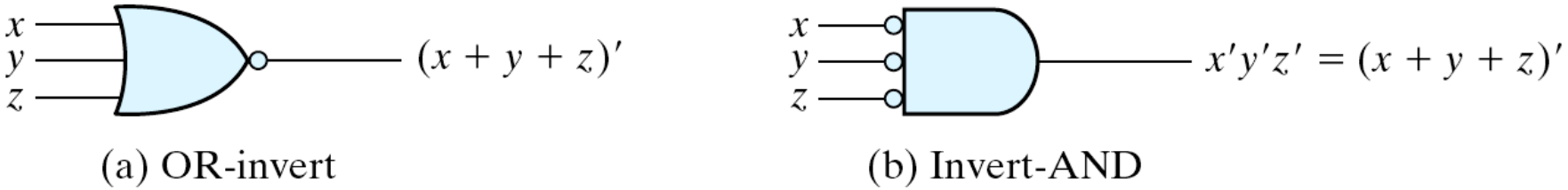


Figure 3.23 Two Graphic Symbols for NOR Gate

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

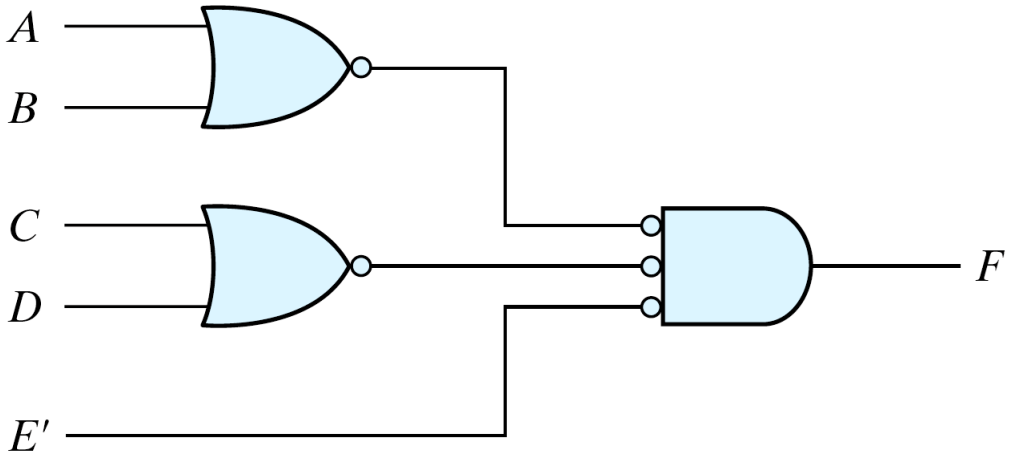


Figure 3.24 Implementing $F = (A + B)(C + D)E$

Example (p.113)

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

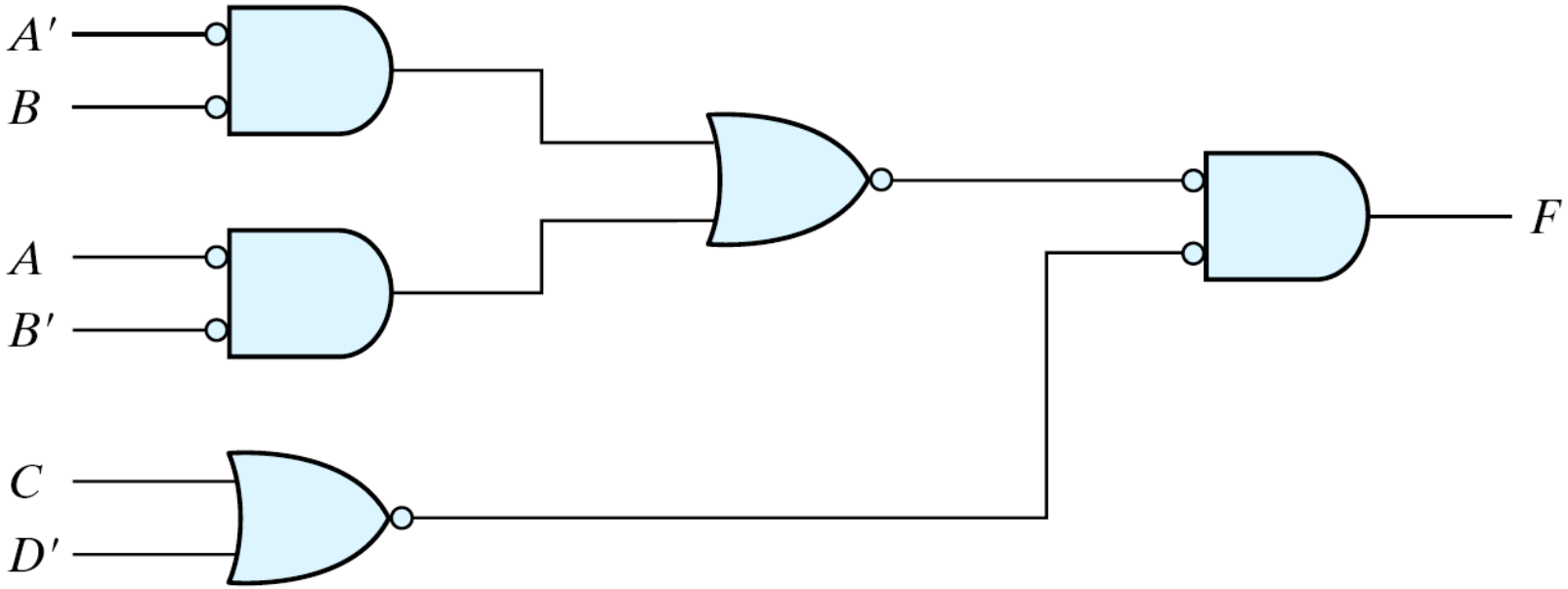


Figure 3.25 Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

3-7 Other Two-level Implementations

- **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 (p.115)

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)

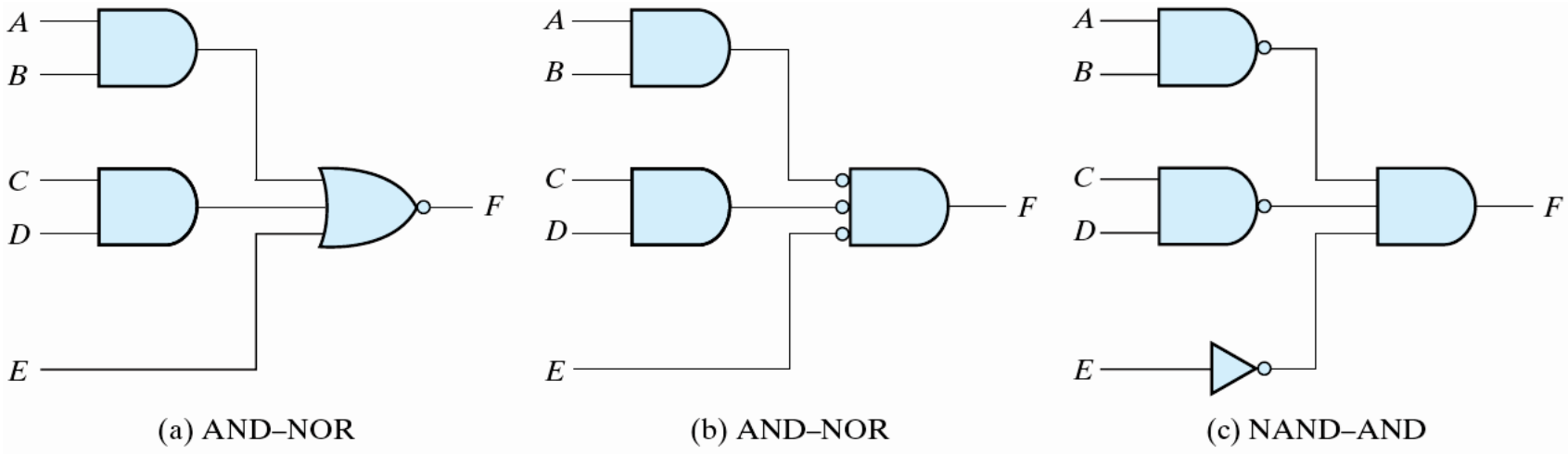


Figure 3.27 AND-OR-INVERT circuits, $F = (AB + CD + E)'$

OR-AND-Invert Implementation (p.116)

- **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)**

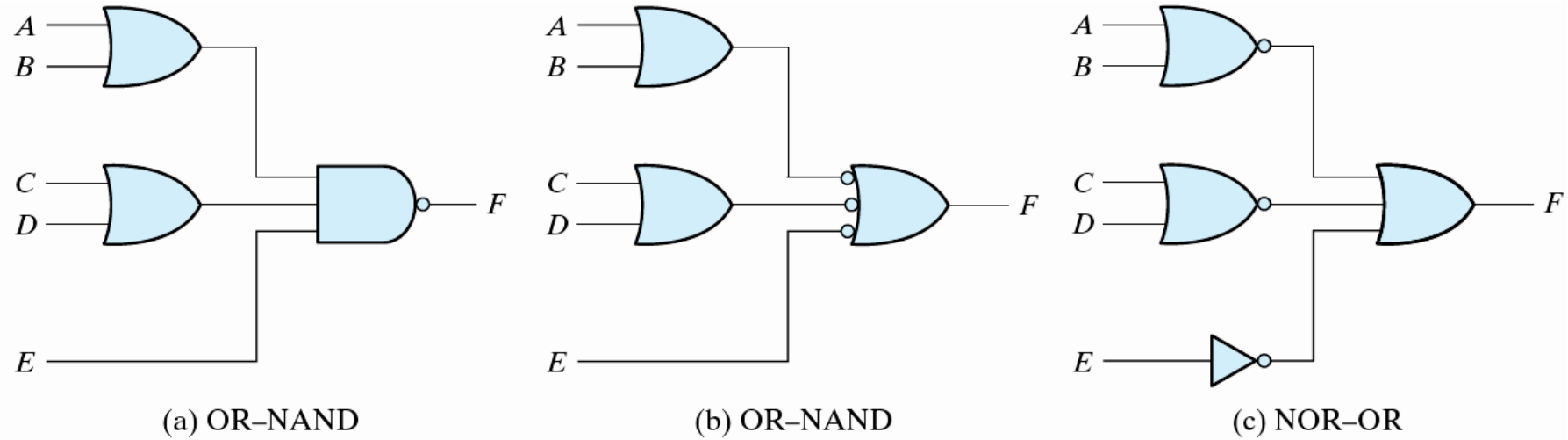


Figure 3.28 OR-AND-INVERT circuits, $F = ((A+B)(C+D)E)'$

Tabular Summary and Examples (p.117)

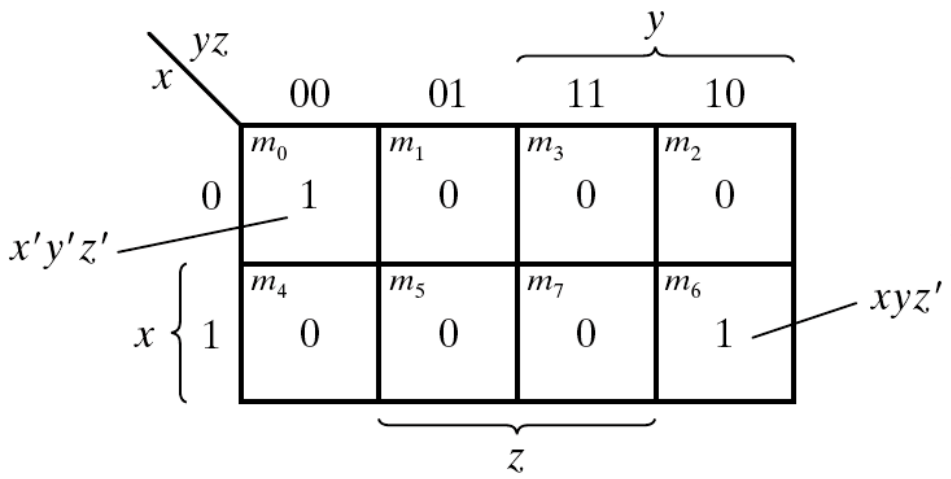
Table 3.2
Implementation with Other Two-Level Forms

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.

Tabular Summary and Examples (p.117)

- **Example 3-10: Implement the following function with (a) AND-NOR (b) NAND-AND (c) OR-NAND (d) NOR-OR forms**



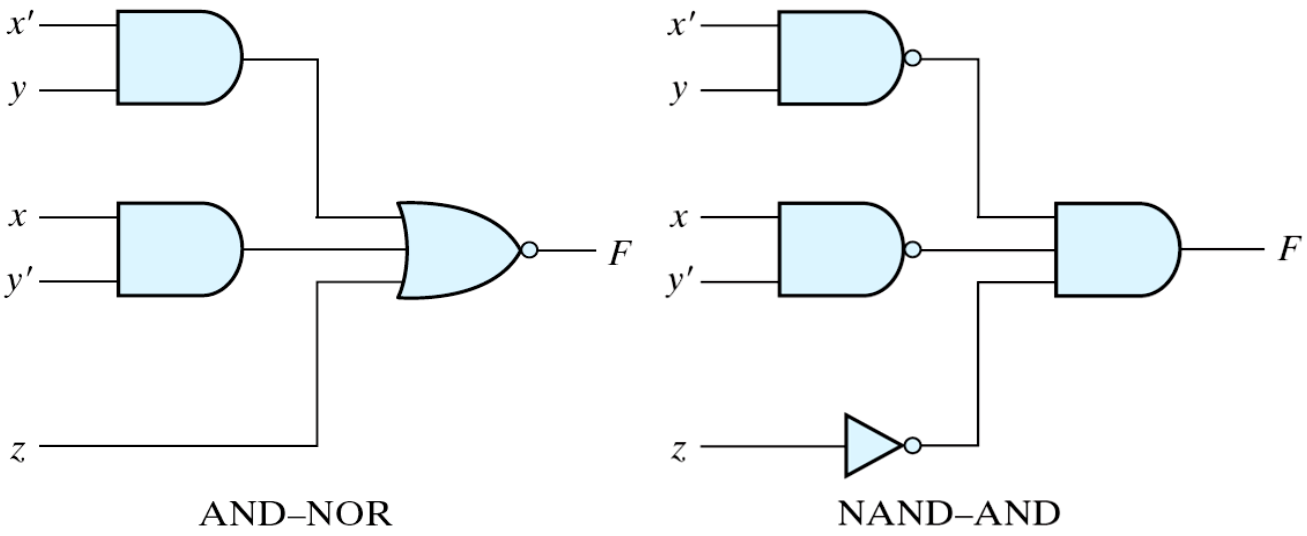
$F = x'y'z' + xyz'$

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

(a) Map simplification in sum of products

Tabular Summary and Examples (p.118)

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

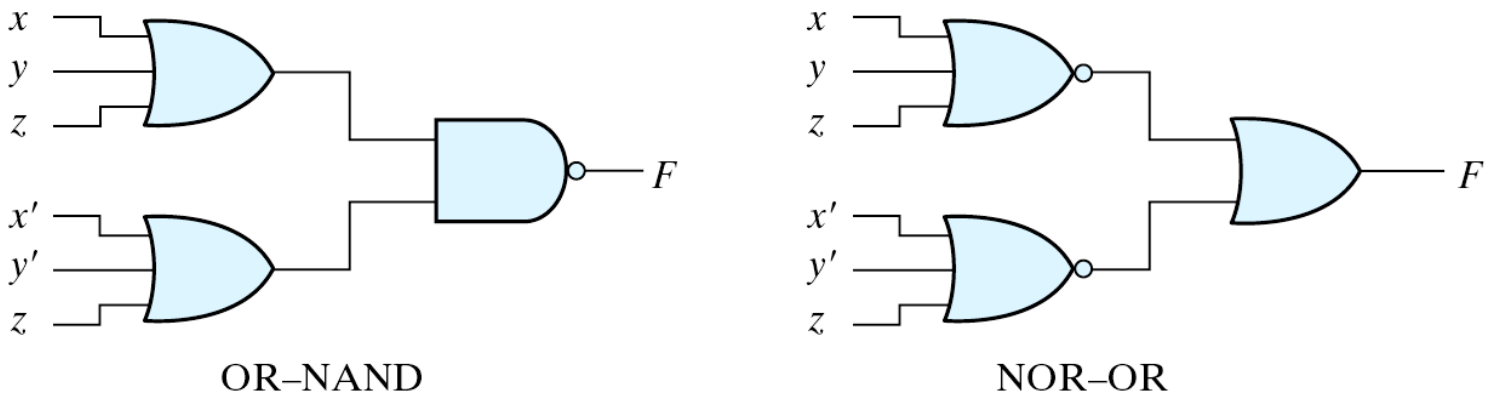


(b) $F = (x'y + xy' + z)'$

Tabular Summary and Examples (p.118)

(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)



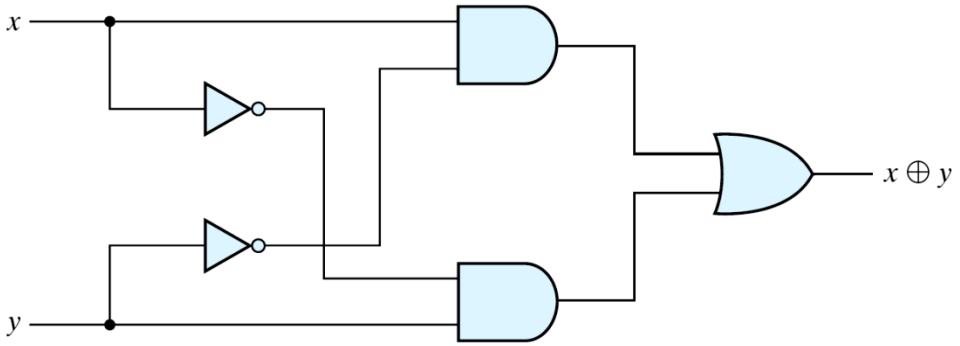
(c) $F = [(x + y + z)(x' + y' + z)]'$

3-8 Exclusive-OR Function (p.119)

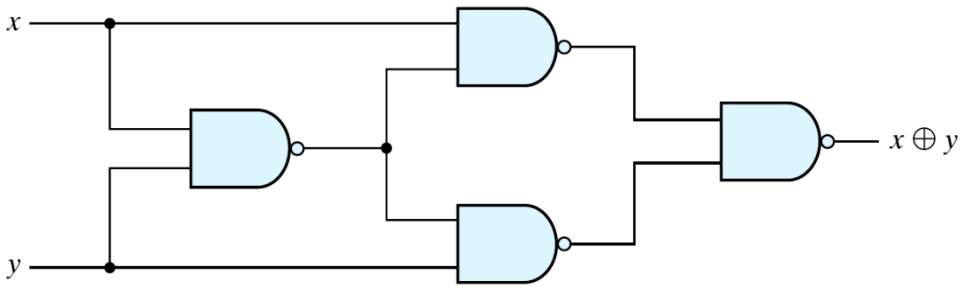
- **Exclusive-OR (XOR)**
  - $x \oplus y = xy' + x'y$
- **Exclusive-NOR (XNOR)**
  - $(x \oplus y)' = xy + x'y'$
- **Some identities**
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = (x \oplus y)'$
  - $x' \oplus y = (x \oplus y)'$
- **Commutative 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 (p.120)

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



(a) With AND–OR–NOT gates

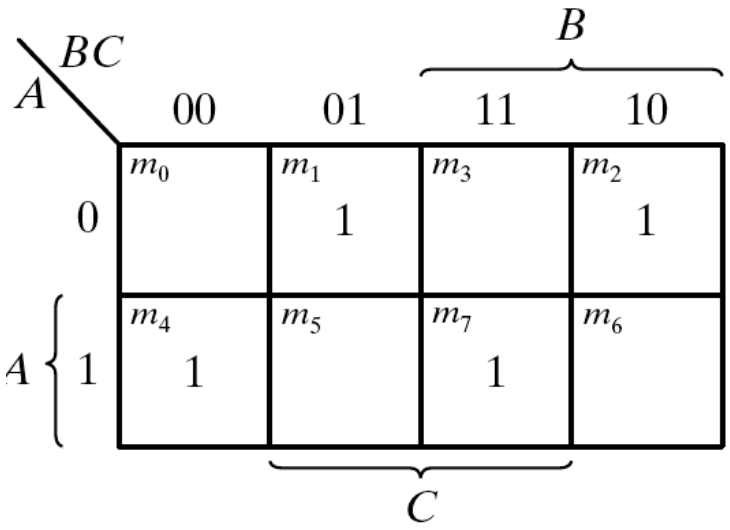


(b) With NAND gates

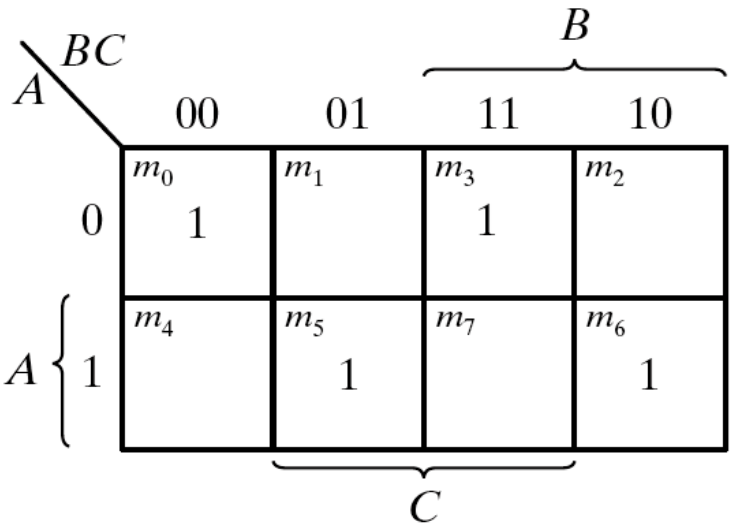
Figure 3.30 Exclusive-OR Implementations

Odd Function (p.120, 121)

- $A \oplus B \oplus C = (AB'+A'B)C' + (AB+A'B')C = AB'C'+A'BC'+ABC+A'B'C = \Sigma(1, 2, 4, 7)$
- **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 $F = A \oplus B \oplus C$

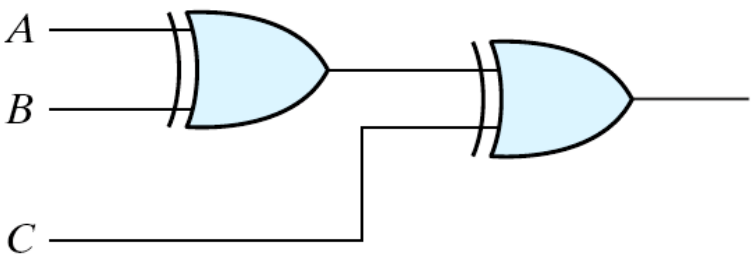


(b) Even function $F = (A \oplus B \oplus C)'$

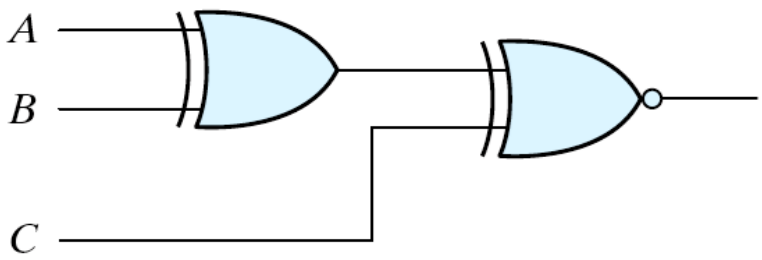
Figure 3.31 Map for a Three-variable Exclusive-OR Function

XOR and XNOR (p.121)

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



(a) 3-input odd function



(b) 3-input even function

Figure 3.32 Logic Diagram of Odd and Even Functions

Four-variable Exclusive-OR function

Four-variable Exclusive-OR function

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

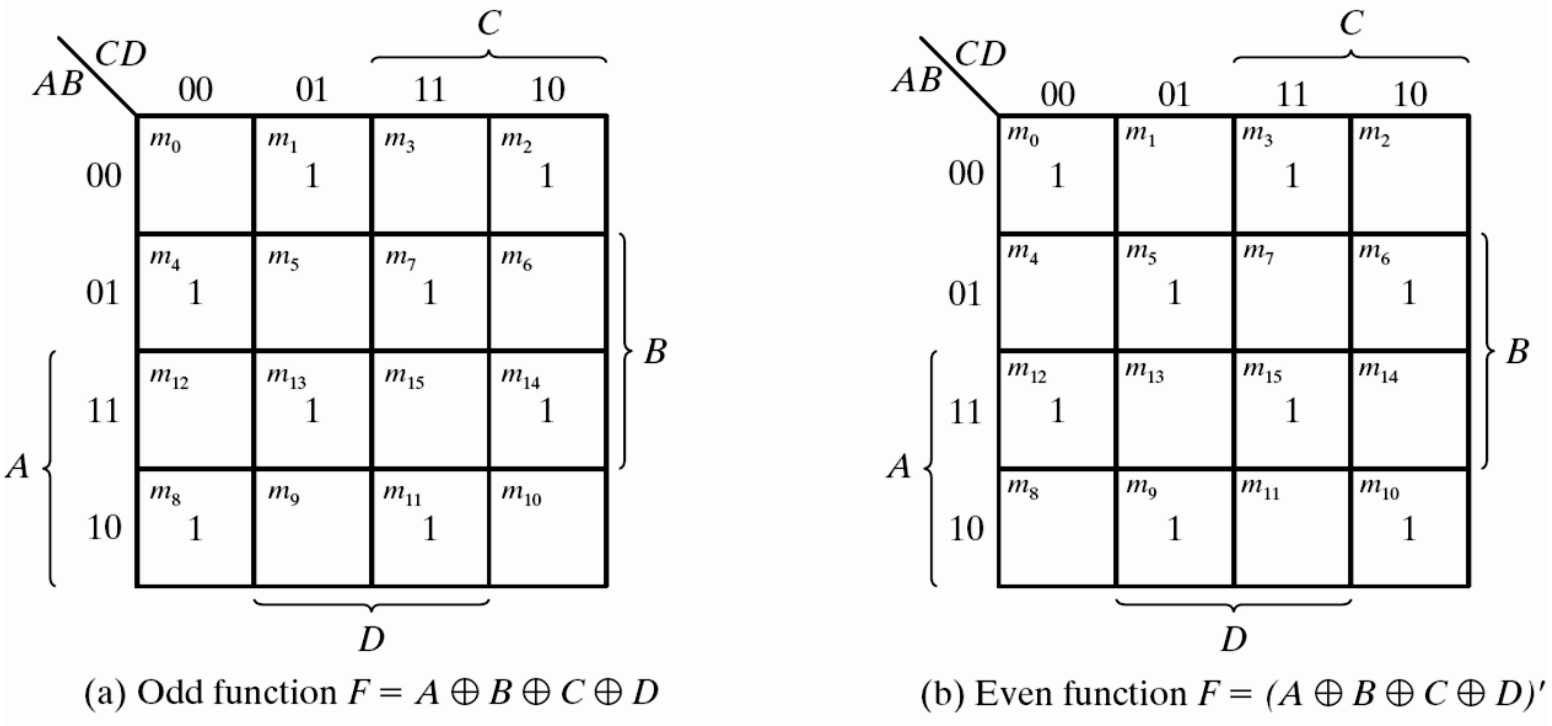


Figure 3.33 Map for a Four-variable Exclusive-OR Function

Parity Generation and Checking (p.122, 123)

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

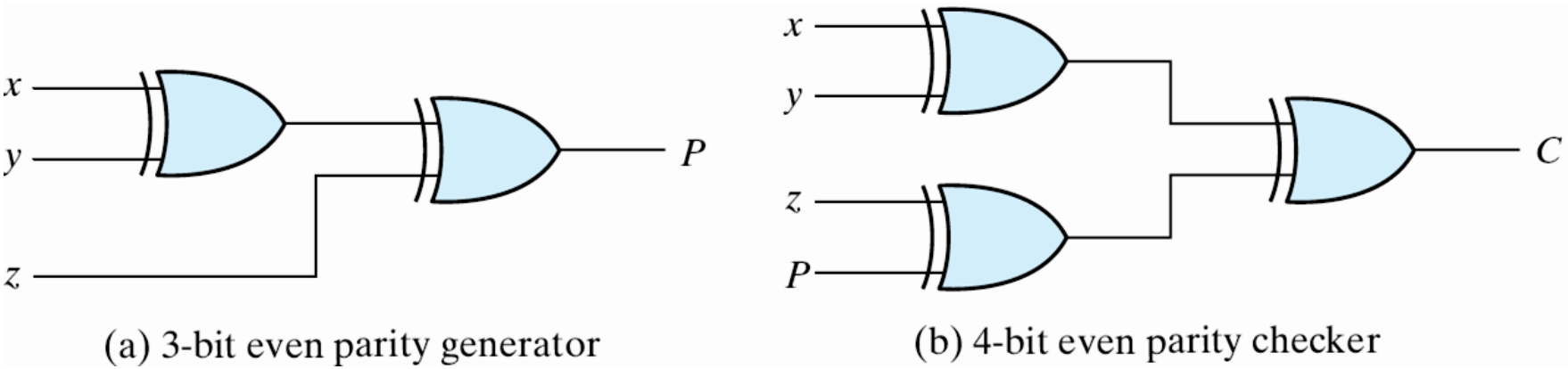


Figure 3.34 Logic Diagram of a Parity Generator and Checker

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 (p.124)

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