Digital Logic Design
Chapter 3

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

- 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 number of variables < 7
- A diagram made up of squares
  - Each square represents one minterm

Two-Variable Map

- A two-variable map
  - Four minterms ***xy, xy', x'y, x'y'***
  - 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): $m_1 + m_2 + m_3 = x'y + xy' + xy$
    $= x(y+y') + y(x+x') = x + y$

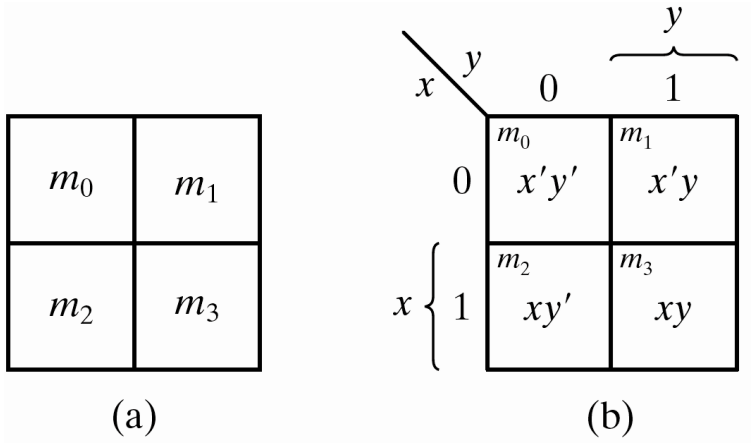


Figure 3.1 Two-variable Map

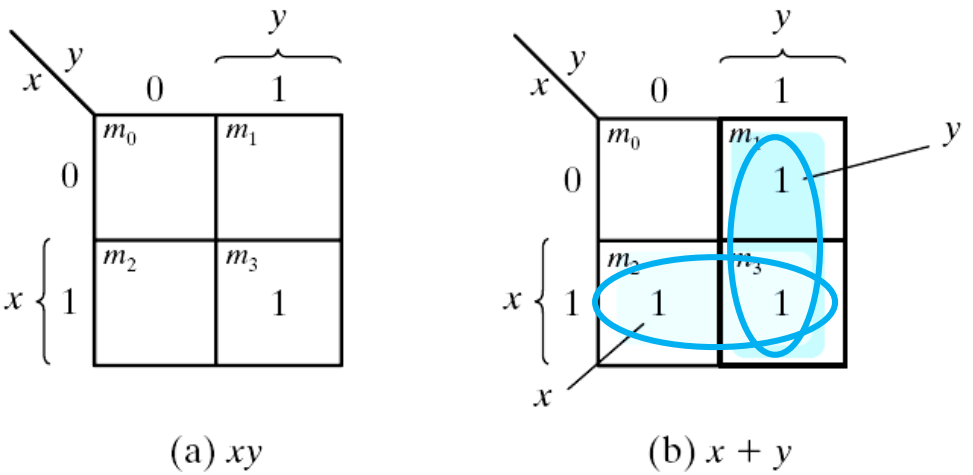


Figure 3.2 Representation of functions in the map

A Three-variable Map

- A three-variable map (Eight minterms)
  - The minterms arranged in Gray code sequence not in binary
  - Any two adjacent squares (vertically or horizontally. But not diagonally) in the map differ by only on variable
    - » e.g., m_5 and m_7 can be simplified
    - » $m_5 + m_7 = xy'z + xyz = xz\,(y'+y) = xz$

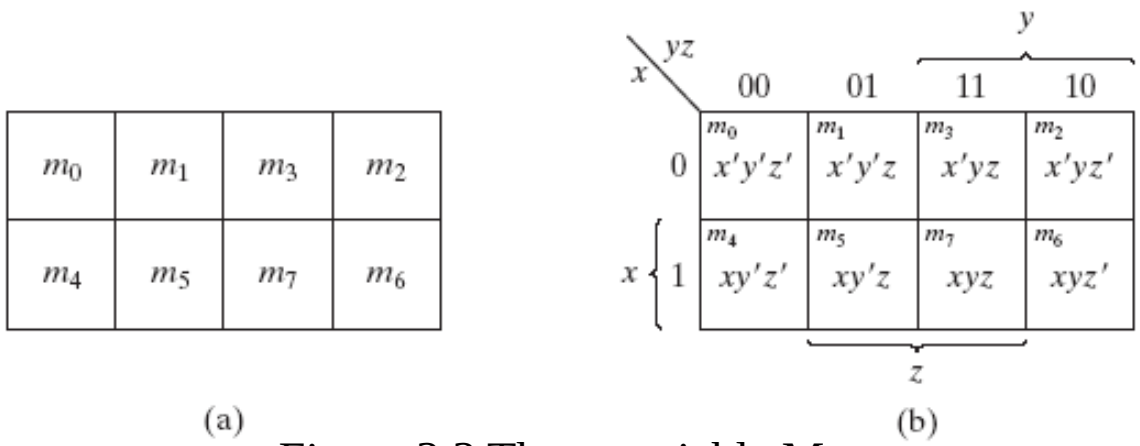


Figure 3.3 Three-variable Map

A Three-variable Map

- 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, 10); x (1)

(b)

Fig. 3-3 Three-variable Map

Example 3.1

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

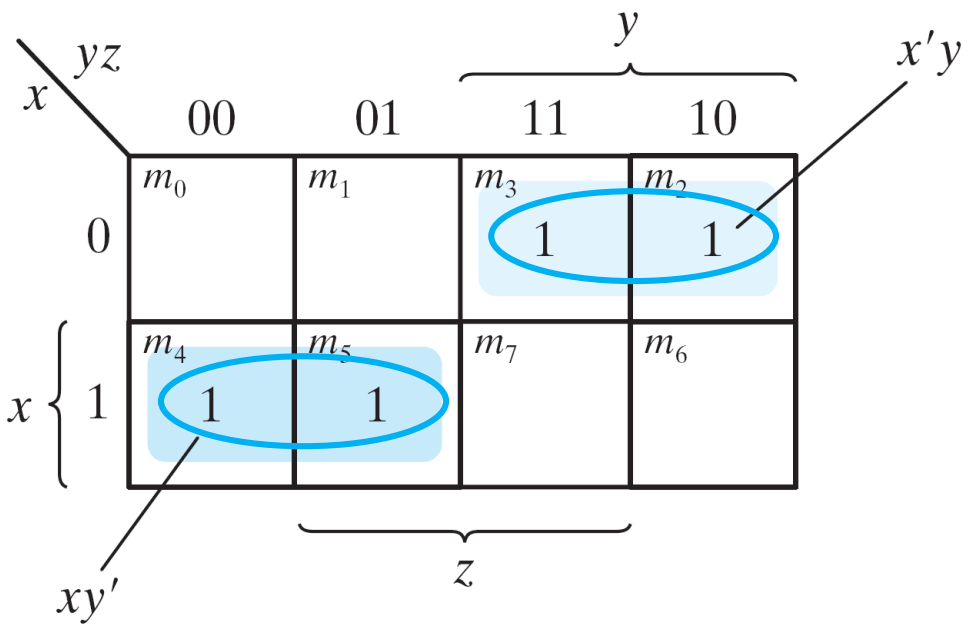


Figure 3.4 Map for Example 3.1, $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

Example 3.2

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

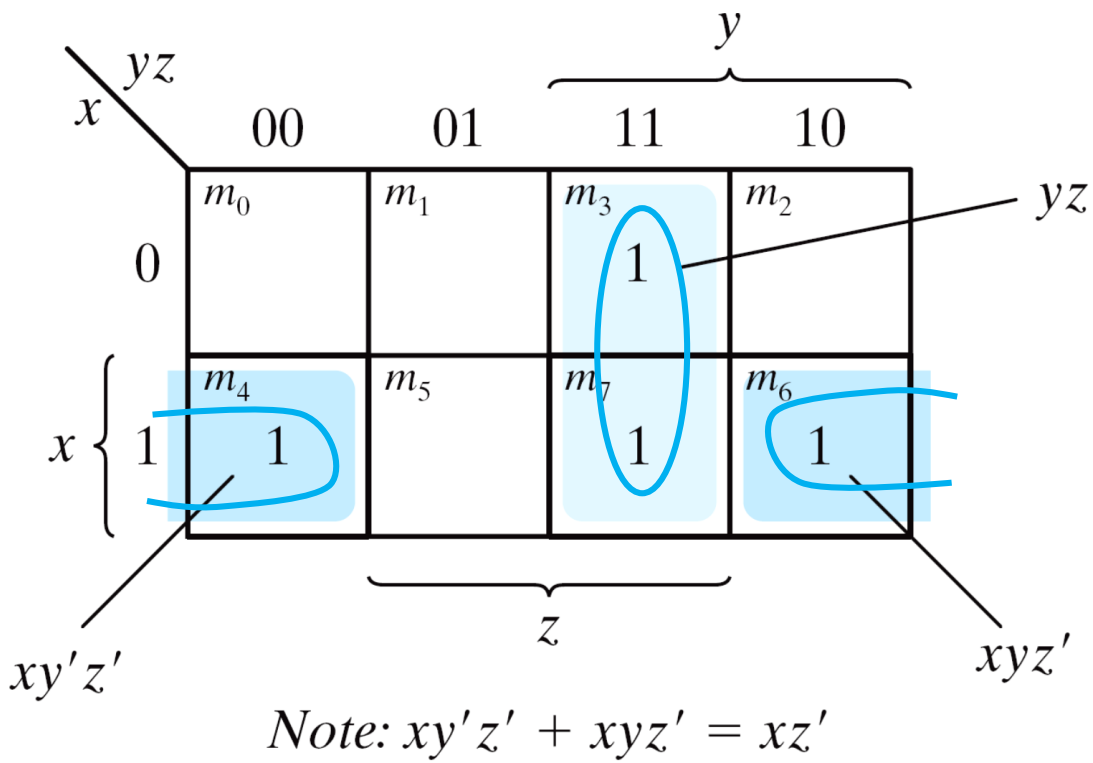


Figure 3.5 Map for Example 3-2; F(x, y, z) = $\Sigma(3, 4, 6, 7) = yz + xz'$

Four adjacent Squares

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

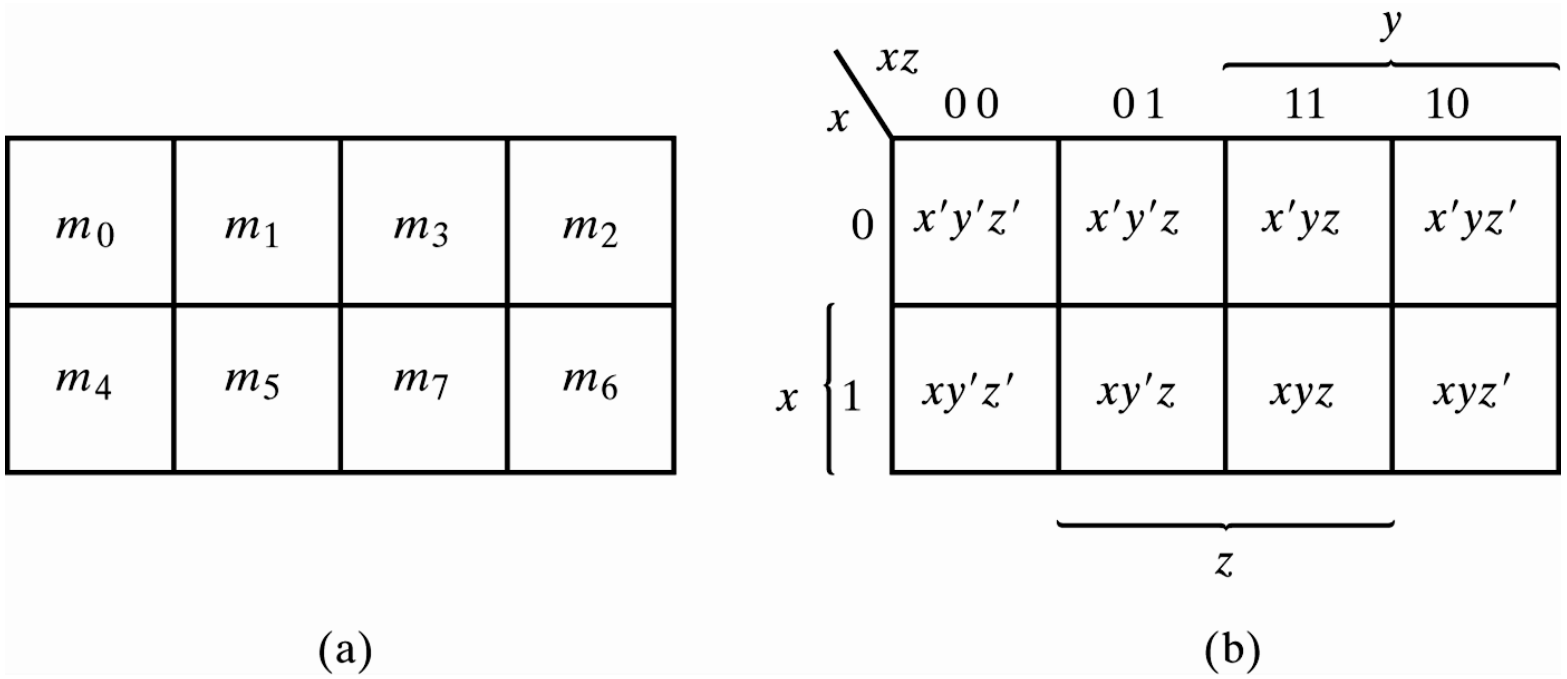
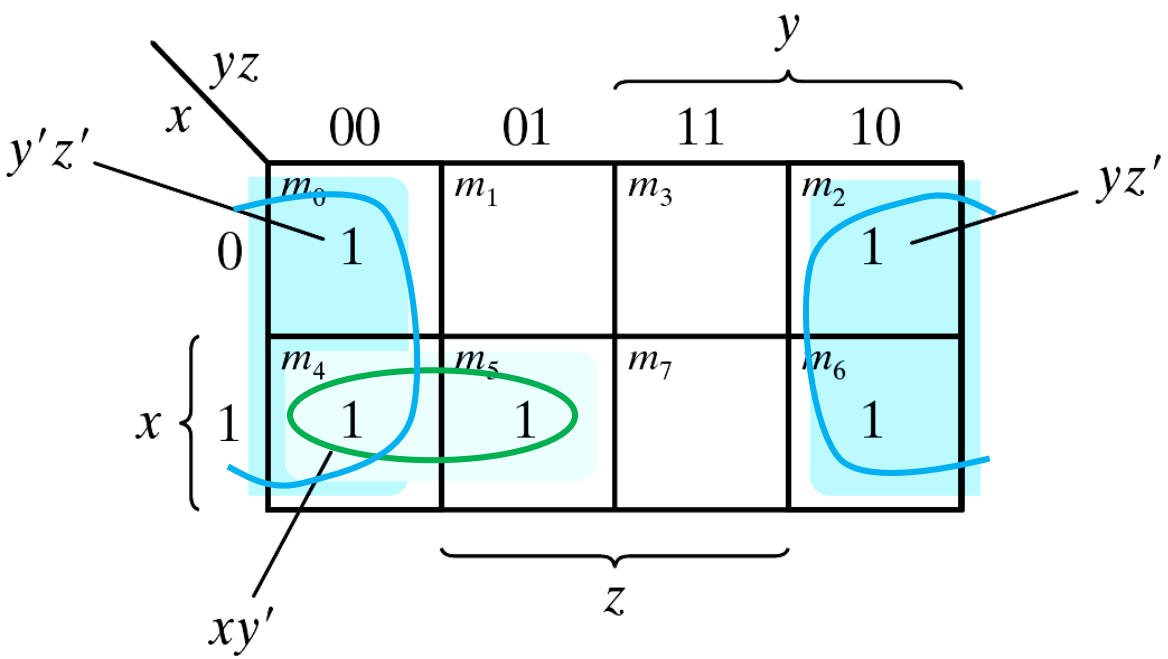


Figure 3.3 Three-variable Map

Example 3.3

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



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

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 $F = A'C + A'B + AB'C + BC$
  - Express in sum of minterms and find the minimal SOP
  - Note: A'C = A' (first row) and C(two middle columns) to give squares 001 & 011.

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

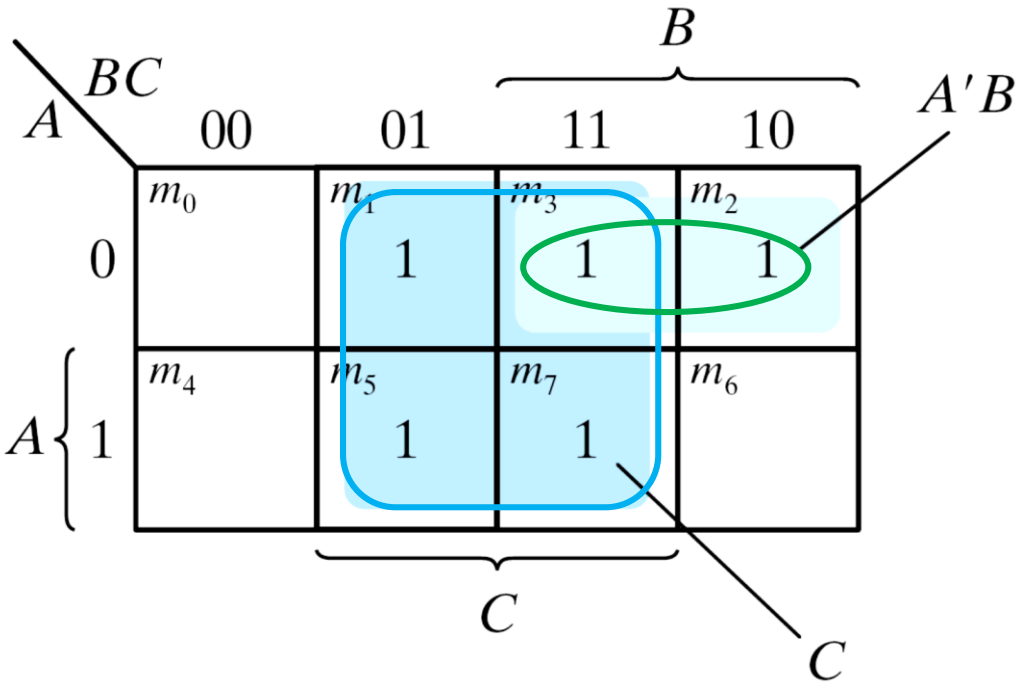


Figure 3.7 Map for Example 3.4, $A'C + A'B + AB'C + BC = C + A'B$

Summary of three-variable Map

- ***One square*** represents one mintem, giving a term with ***three literals***.
- ***Two adjacent squares*** represent a term with ***two literals***.
- ***Four adjacent squares*** represent a term with ***one literal***.
- ***Eight adjacent squares*** encompass the entire map and produce a function that is always equal to ***1***.

3.3 Four-Variable Map

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

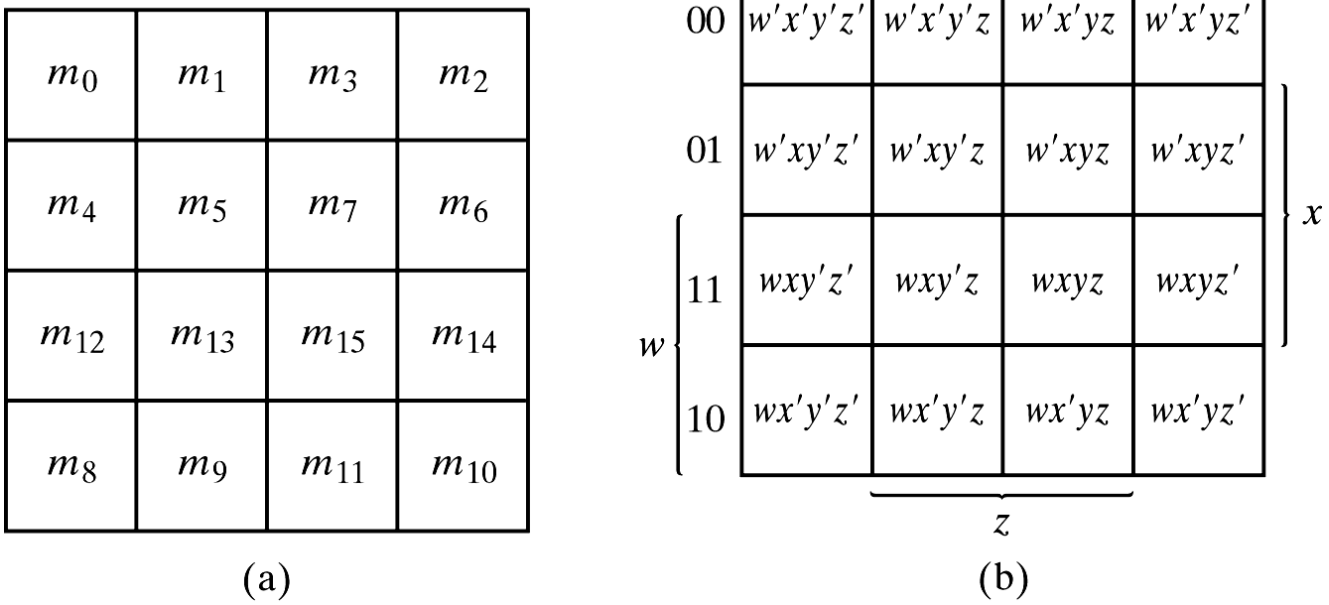


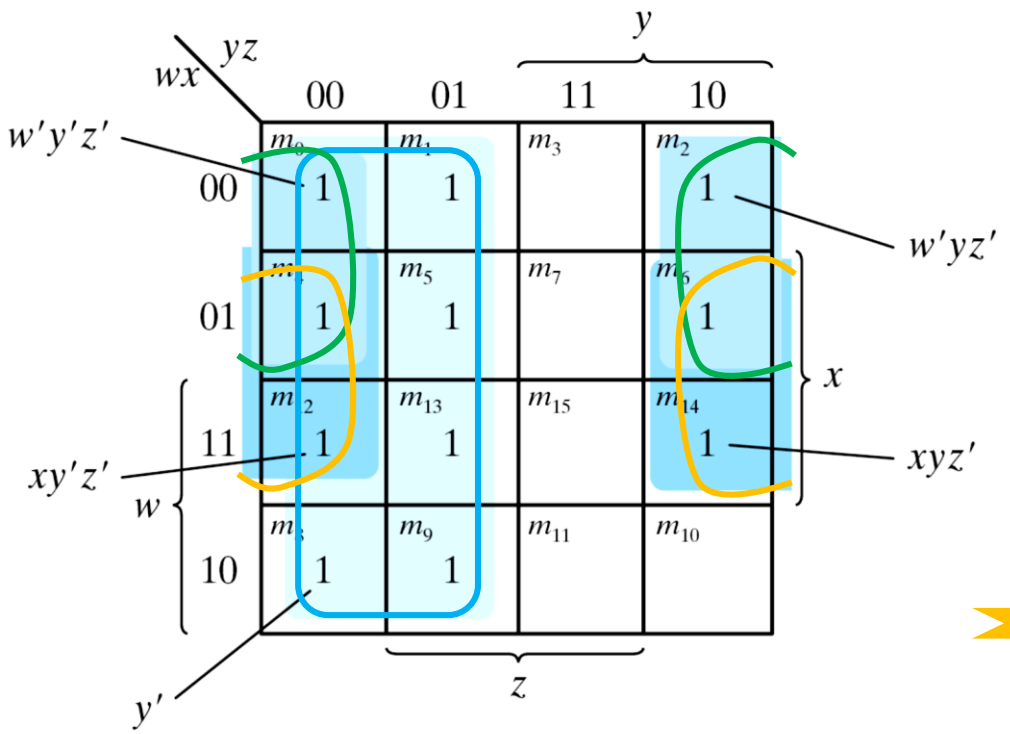
Figure 3.8 Four-variable Map

Summary of four-variable Map

- ***One square*** represents one mintem, giving a term with ***four literals***.
- ***Two adjacent squares*** represent a term with ***three literals***.
- ***Four adjacent squares*** represent a term with ***two literal***.
- ***Eight adjacent squares*** represent a term with ***one literal***.
- ***Sixteen adjacent squares*** produce a function that is always equal to ***1***.

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

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).

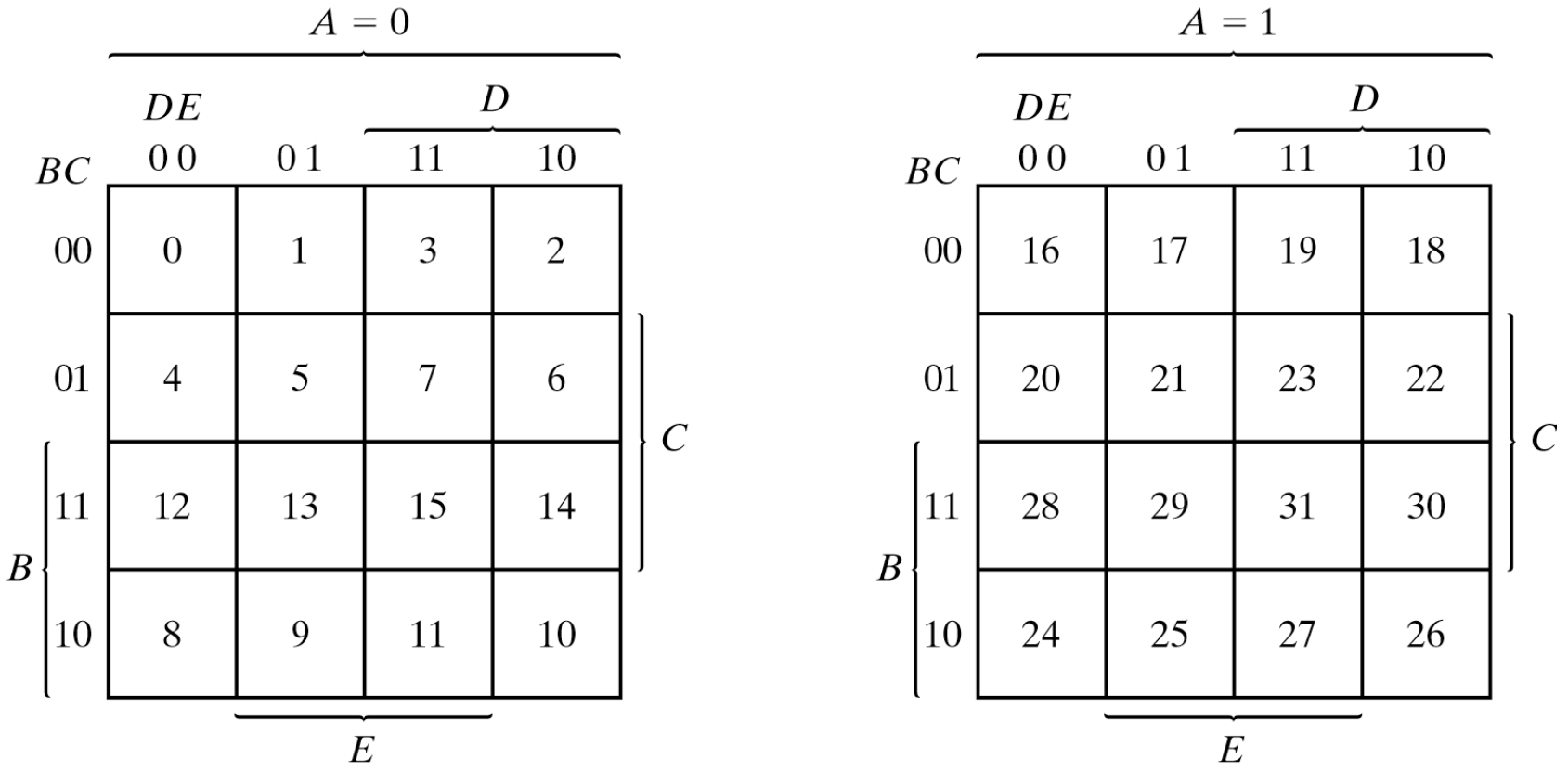


Figure 3.12 Five-variable Map

Example 3.7

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

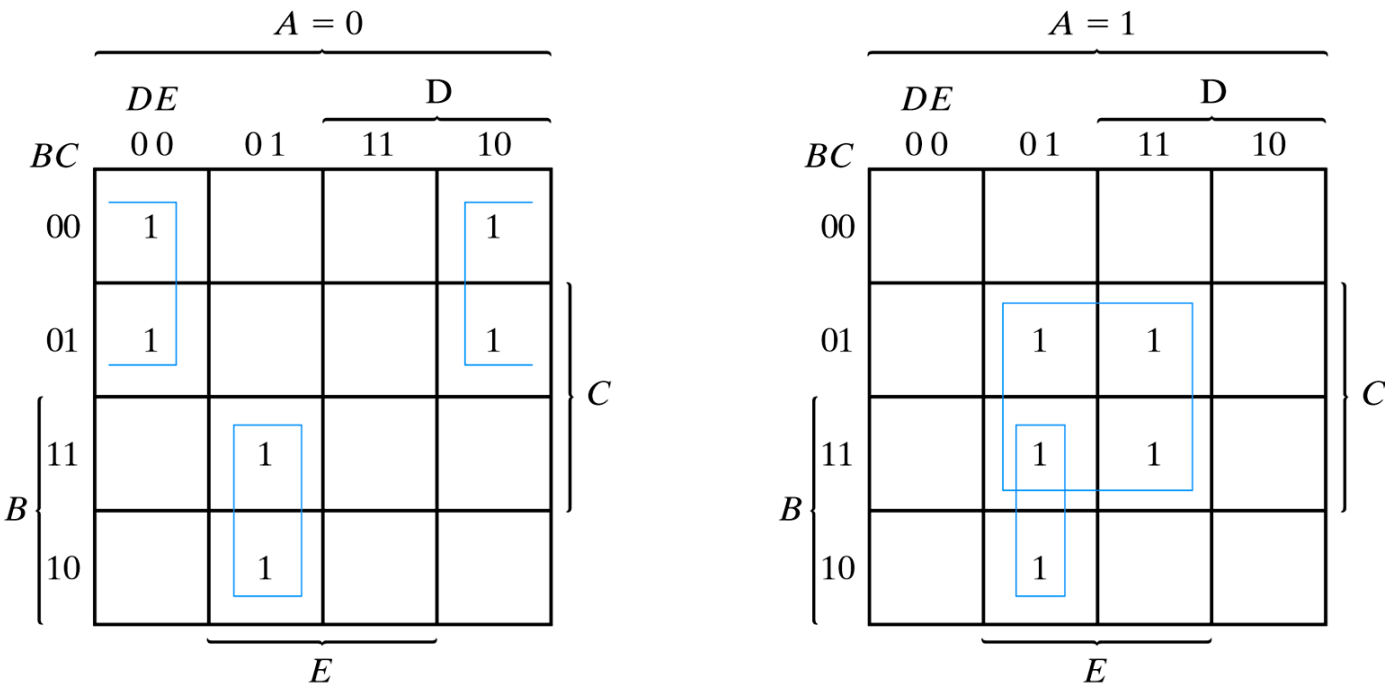


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

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

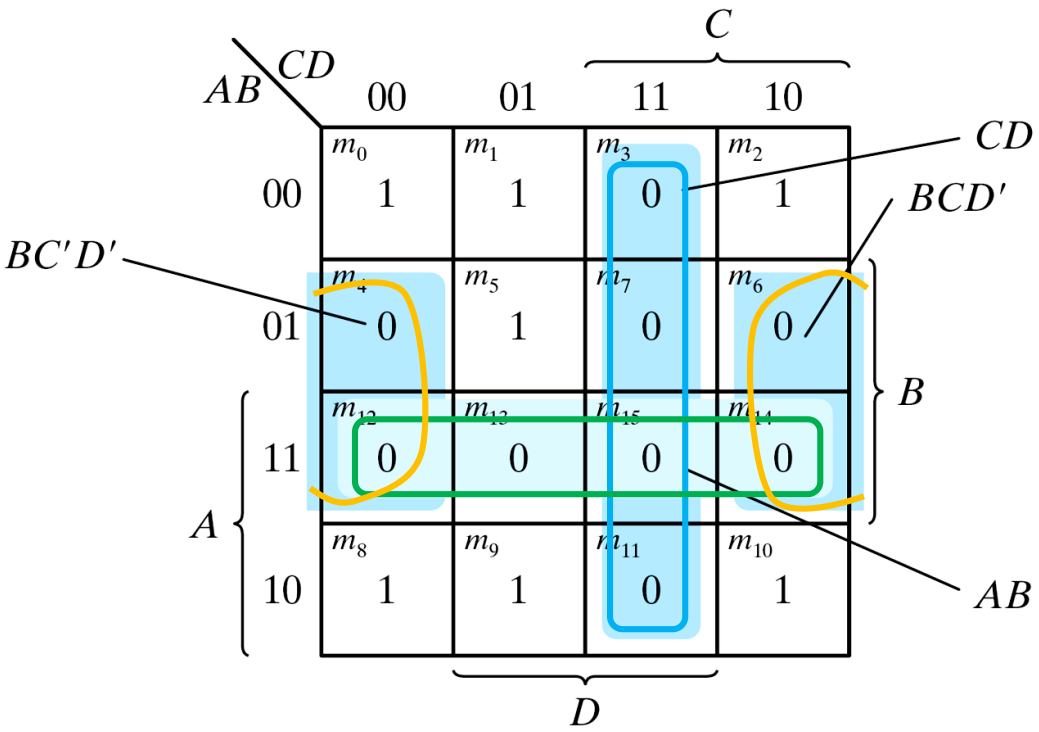
3-5 Product of Sums Simplification

- Approach #1
  - Simplified F' in the form of sum of products
  - Apply DeMorgan's theorem $F = (F')'$
  - F': sum of products → 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 3.8

- 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:



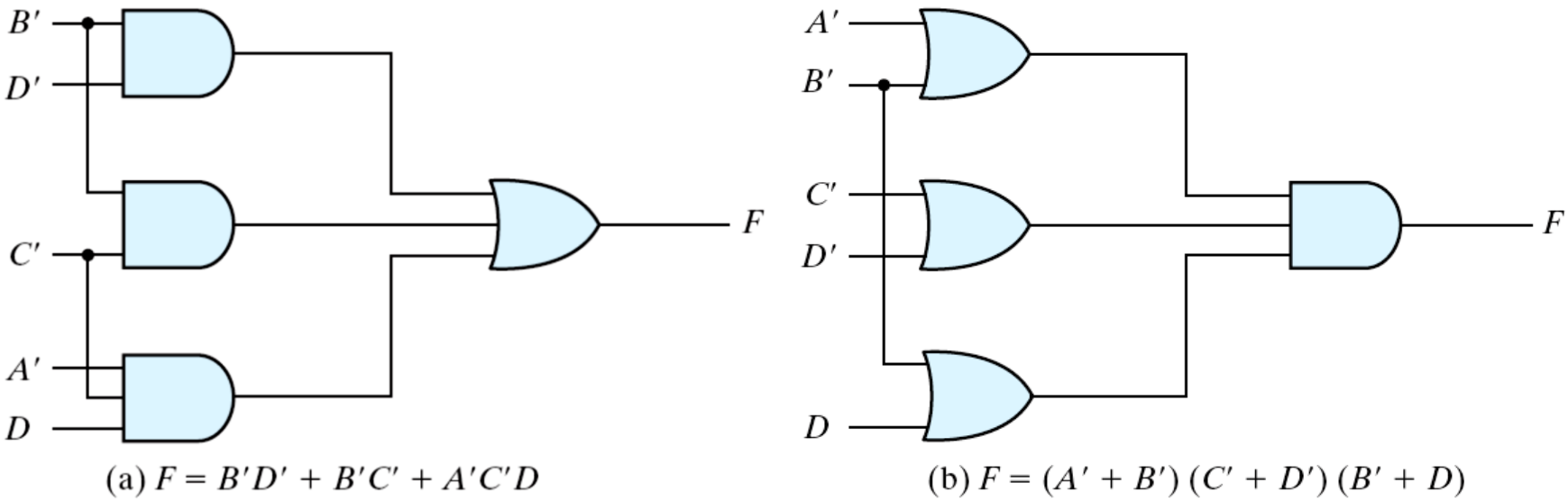
Note: $BC'D' + BCD' = BD'$

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



(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 3.15 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 product-of-maxterm:

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

Table 3.2
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

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 :

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

  - Taking the complement of F′

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

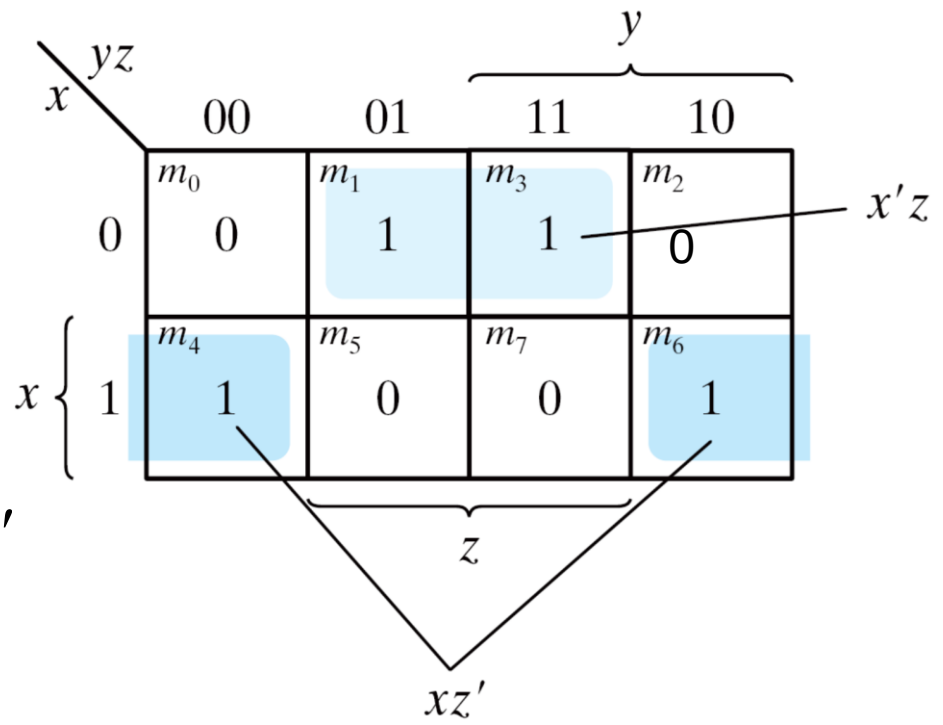


Figure 3.16 Map for the function of Table 3.2

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.)

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

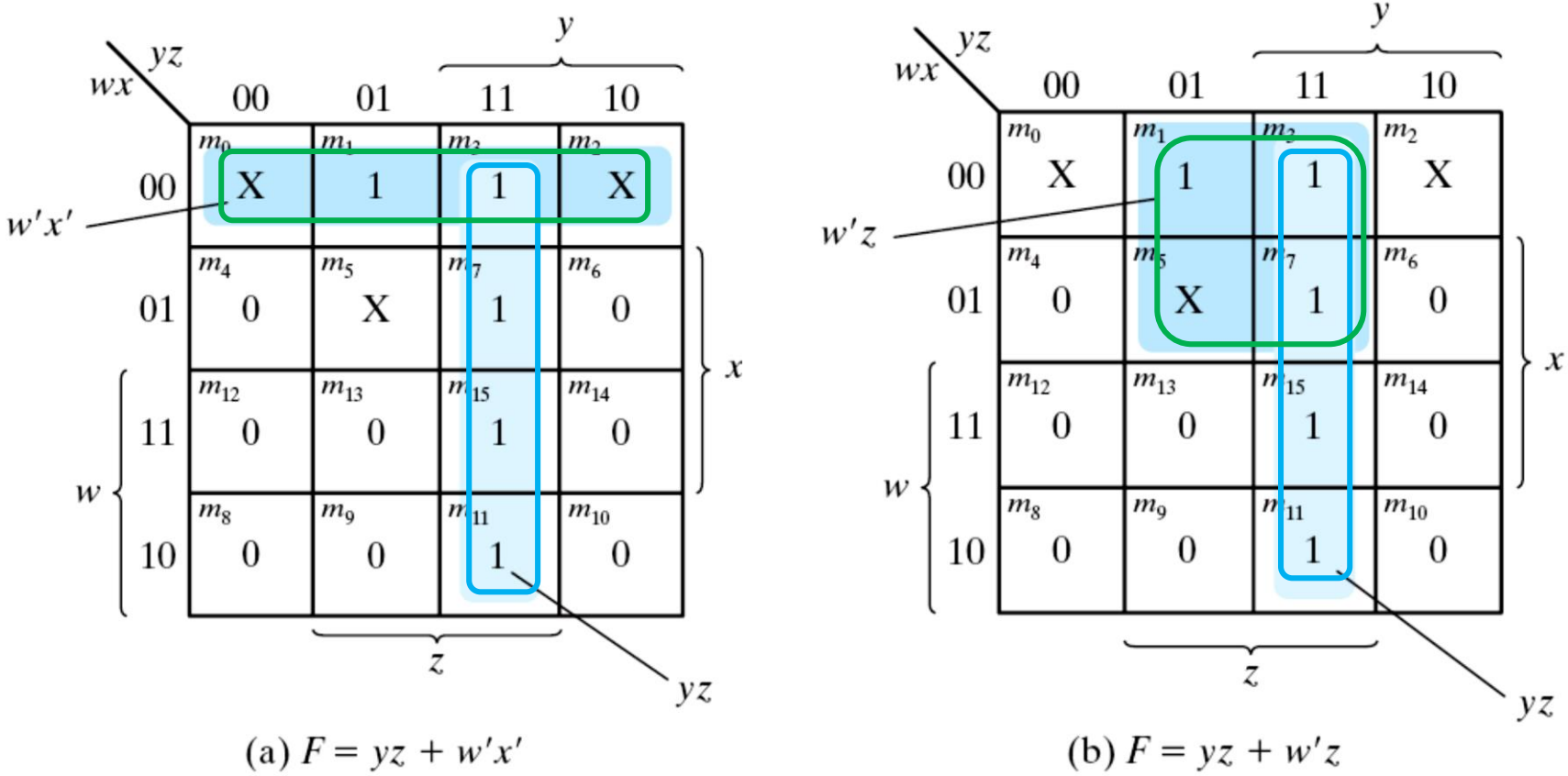


Figure 3.17 Example with don't-care Conditions

Home Work (5)

Digital Design (**4th**)- Morris Mano-Page 116-Problems:

3.2 d, e, f
3. 4 e & f
3.7
3.12
3.15

3-7 NAND and NOR Implementation

- NAND and NOR gates are ***easier*** to fabricate with electronic components.
- Digital circuits are frequently ***constructed*** with NAND or NOR gates rather than with AND and OR gates.
- ***rules and procedures*** have been developed for the ***conversion*** from Boolean functions given in terms of AND, OR and NOT into equivalent NAND and NOR logic diagrams.

NAND Gate

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

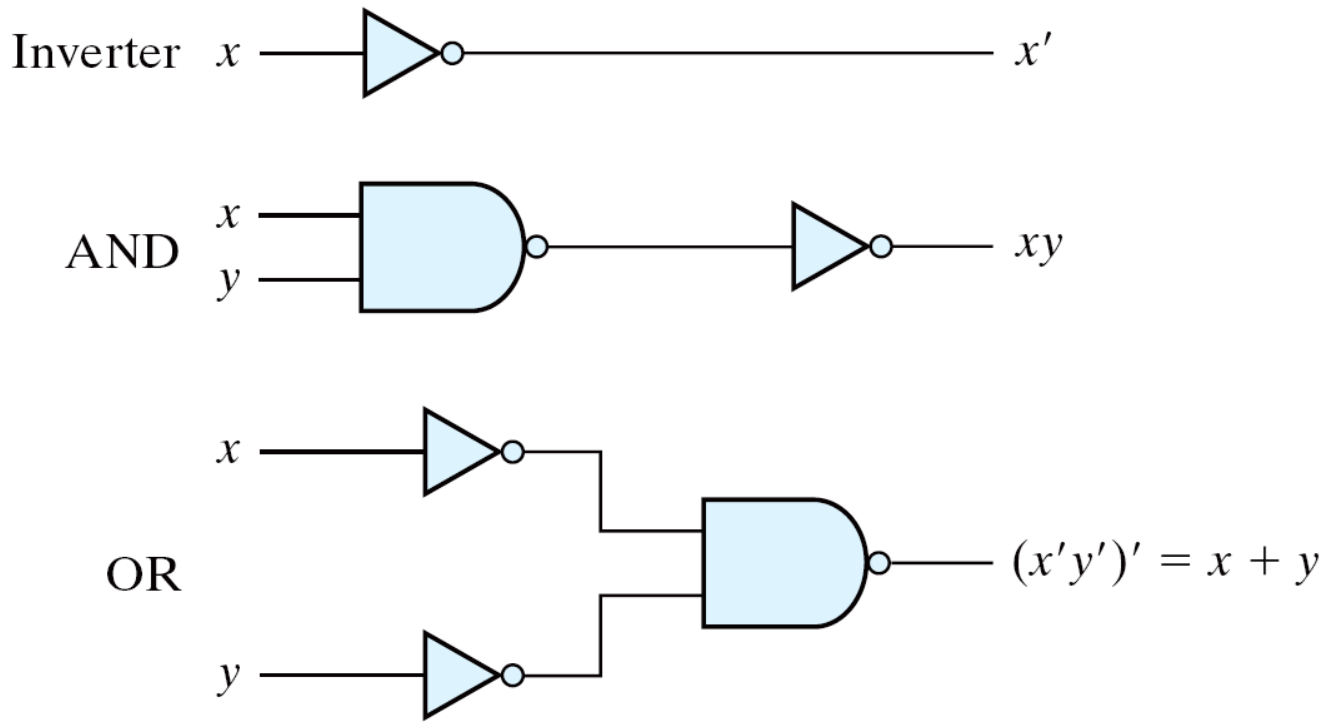
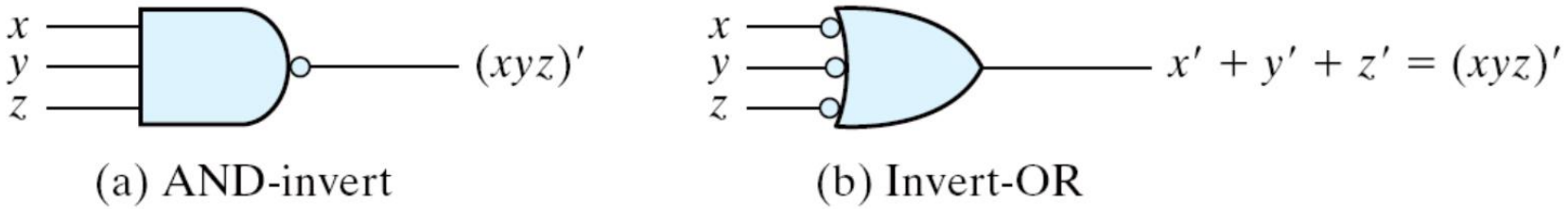


Figure 3.18 Logic Operations with NAND Gates

NAND Gate

- Two graphic symbols for a NAND gate



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

Figure 3.19 Two Graphic Symbols for NAND Gate

Procedure with Two Levels NAND

- The procedure to implement Boolean function with two level NAND:
  - 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.

Example 3.10

- Example 3-10: implement $F(x, y, z) = \sum(1,2,3,4,5,7)$

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

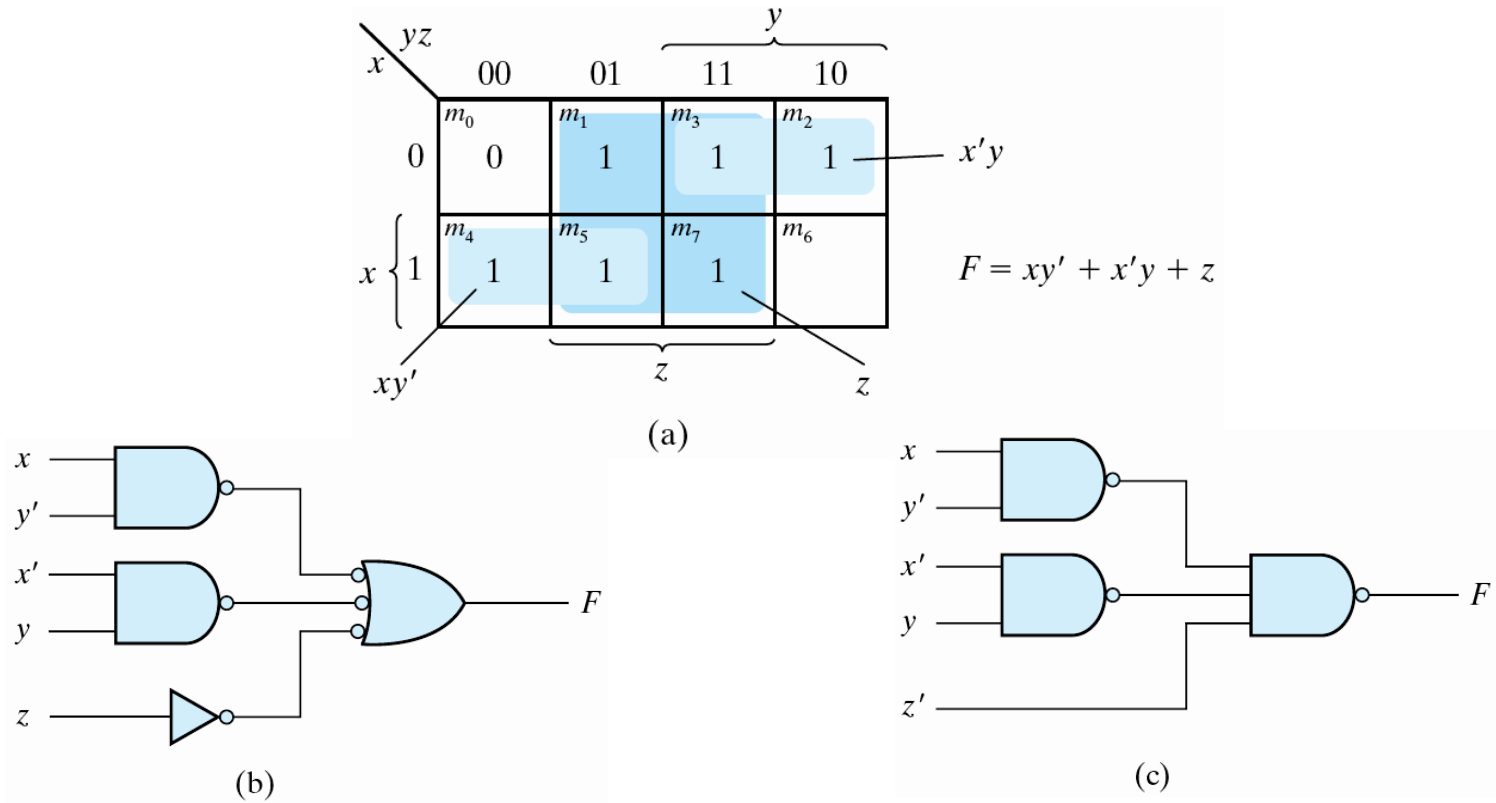


Figure 3.21 Solution to Example 3-10

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 bubble along the same line, insert an inverter.

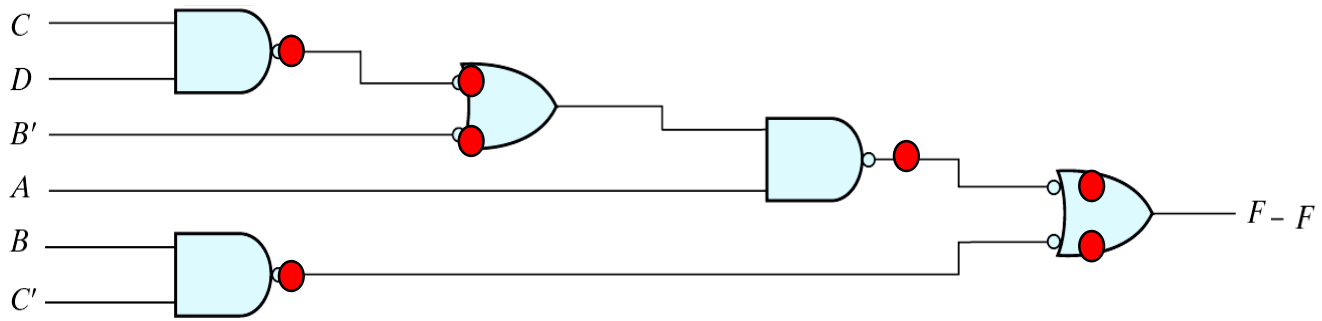
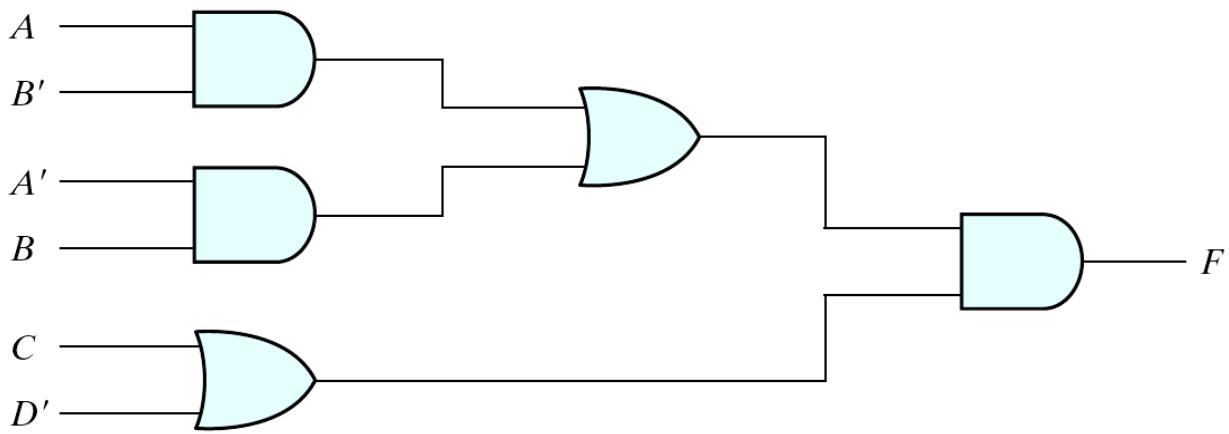
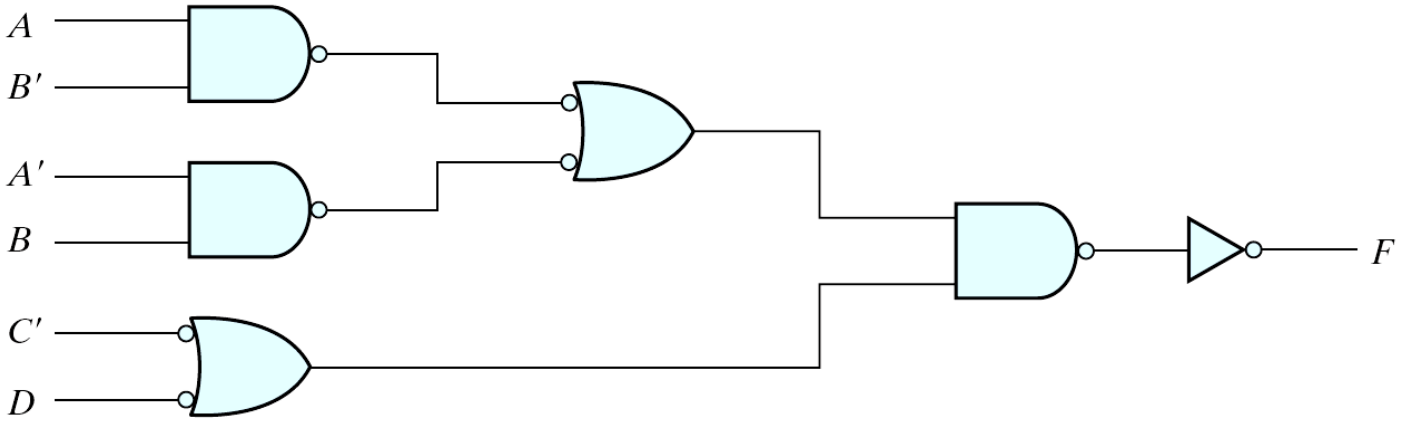


Figure 3.22 Implementing F = A(CD + B) + BC′

NAND Implementation



(a) AND–OR gates



(b) NAND gates

Figure 3.23 Implementing $F = (AB' + A'B)(C + D')$

NOR Implementation

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

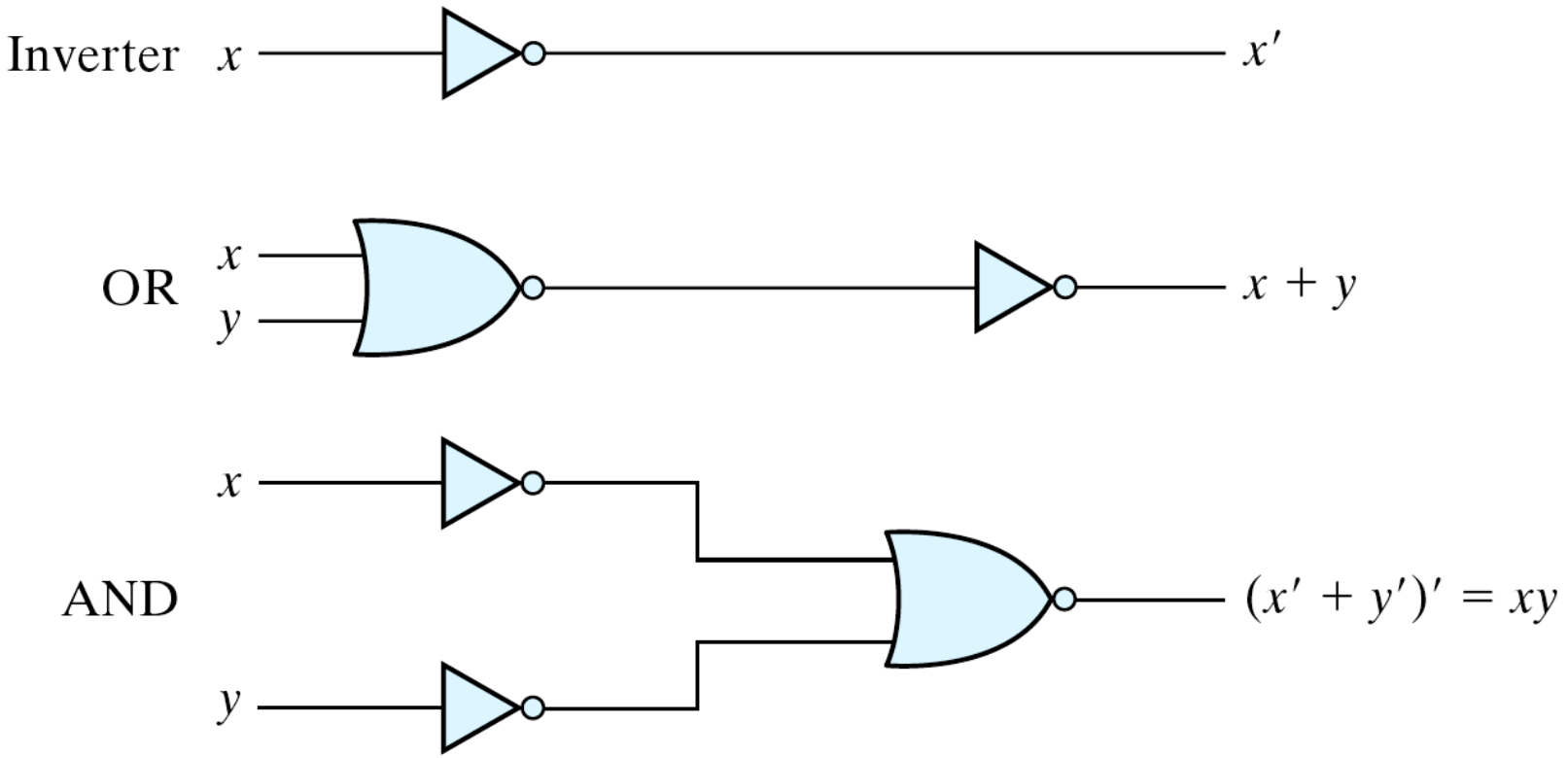


Figure 3.24 Logic Operation with NOR Gates

Two Graphic Symbols for a NOR Gate

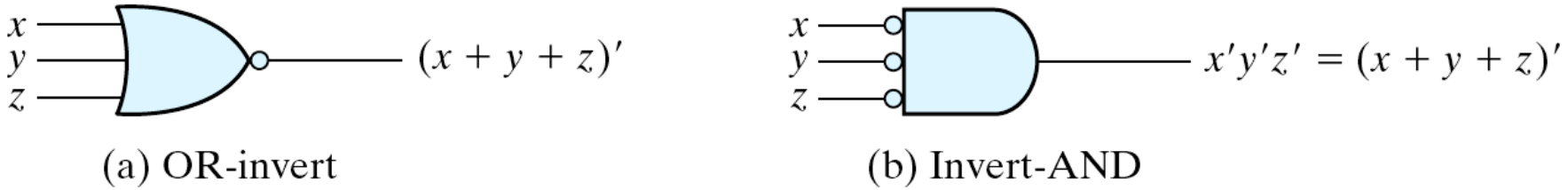


Figure 3.25 Two Graphic Symbols for NOR Gate

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

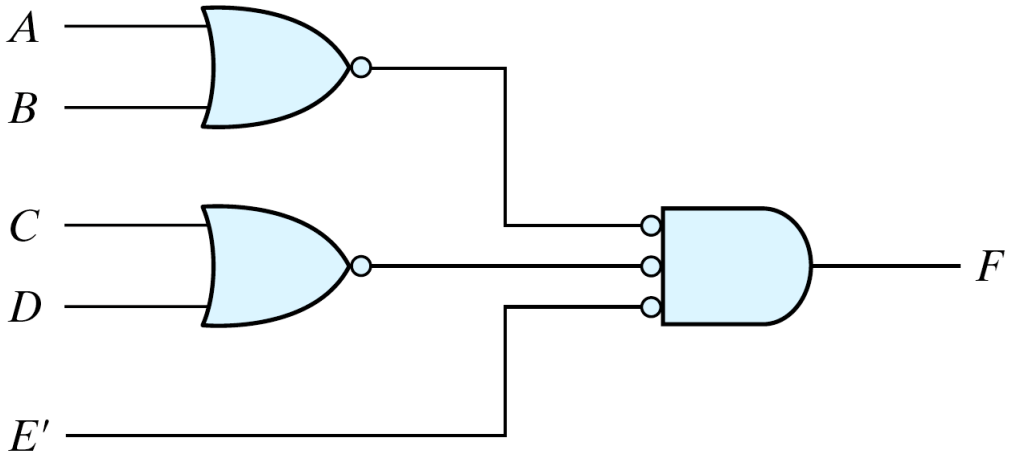


Figure 3.26 Implementing $F = (A + B)(C + D)E$

Example

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

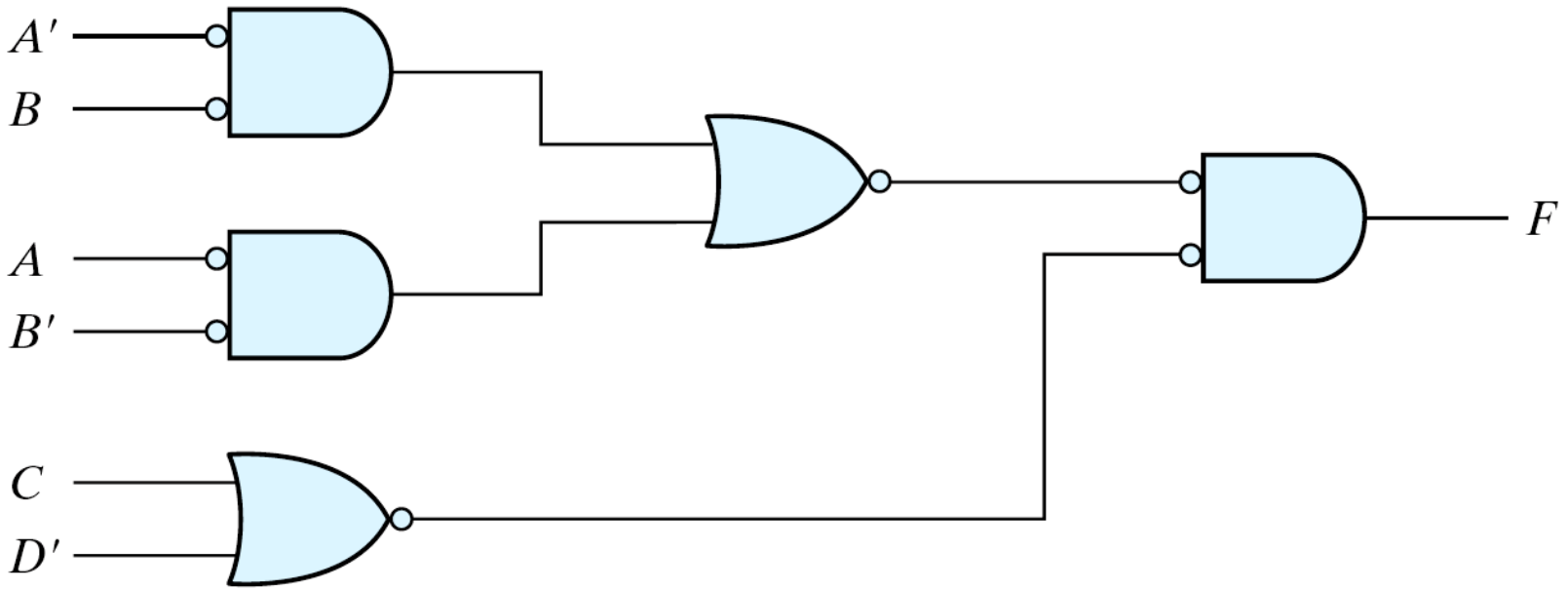


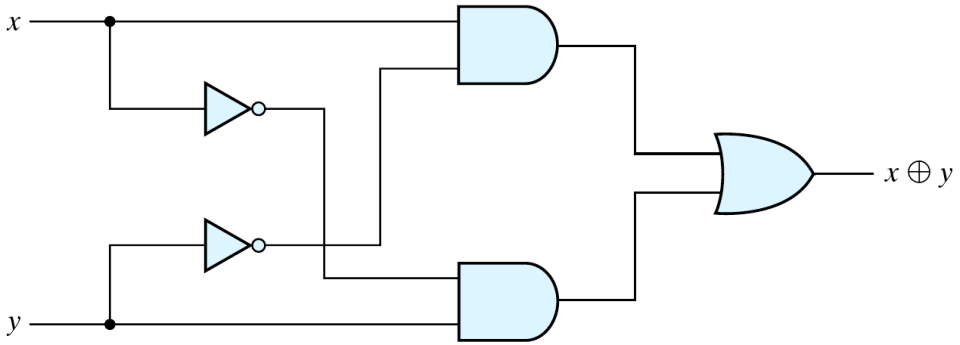
Figure 3.27 Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

3-9 Exclusive-OR Function

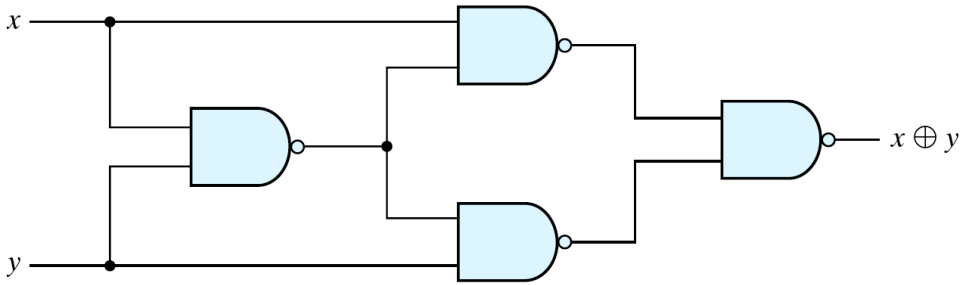
- Exclusive-OR (XOR) $x \oplus y = xy' + x'y$
- Exclusive-NOR (XNOR)
  - $(x \oplus y)' = xy + x'y'$
- Some identities
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = (x \oplus y)'$
  - $x' \oplus y = (x \oplus y)'$
- Commutative and associative
  - $A \oplus B = B \oplus A$
  - $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

3.9 Exclusive-OR Implementations

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



(a) With AND–OR–NOT gates

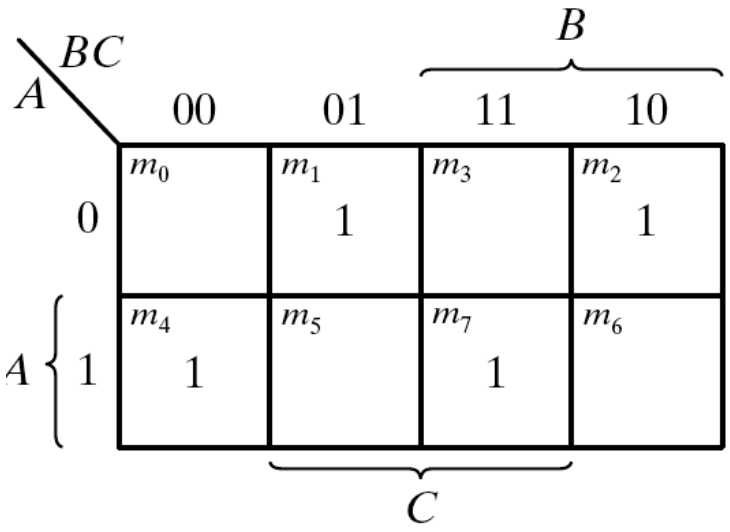


(b) With NAND gates

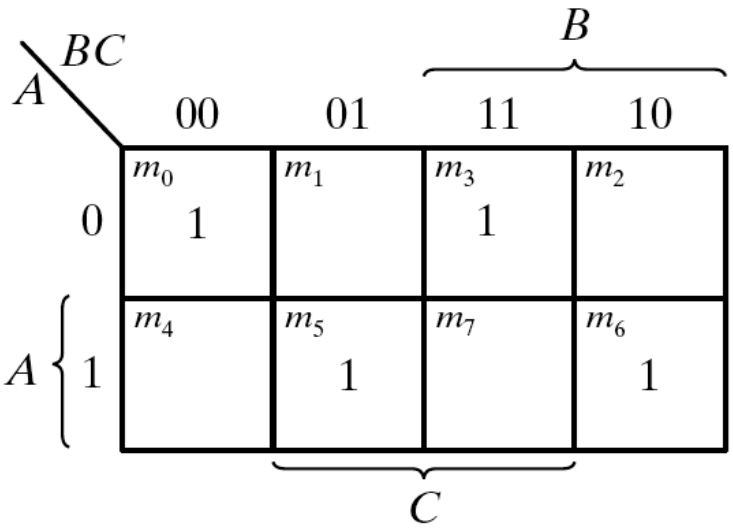
Figure 3.32 Exclusive-OR Implementations

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 a odd function → an odd number of 1's, then $F = 1$.
- XNOR is a even function → an even number of 1's, then $F = 1$.



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

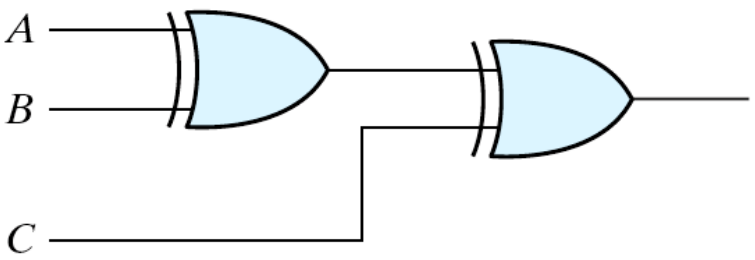


(b) Even function $F = (A \oplus B \oplus C)'$

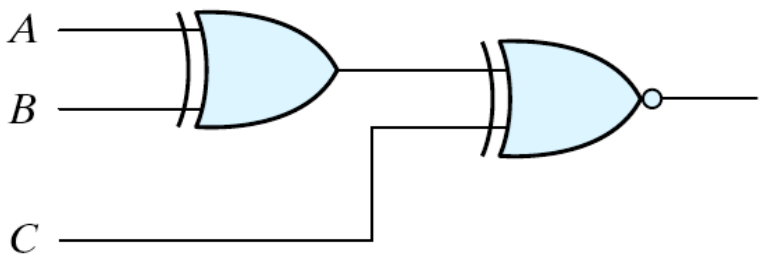
Figure 3.33 Map for a Three-variable Exclusive-OR Function

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

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 number of data bit error

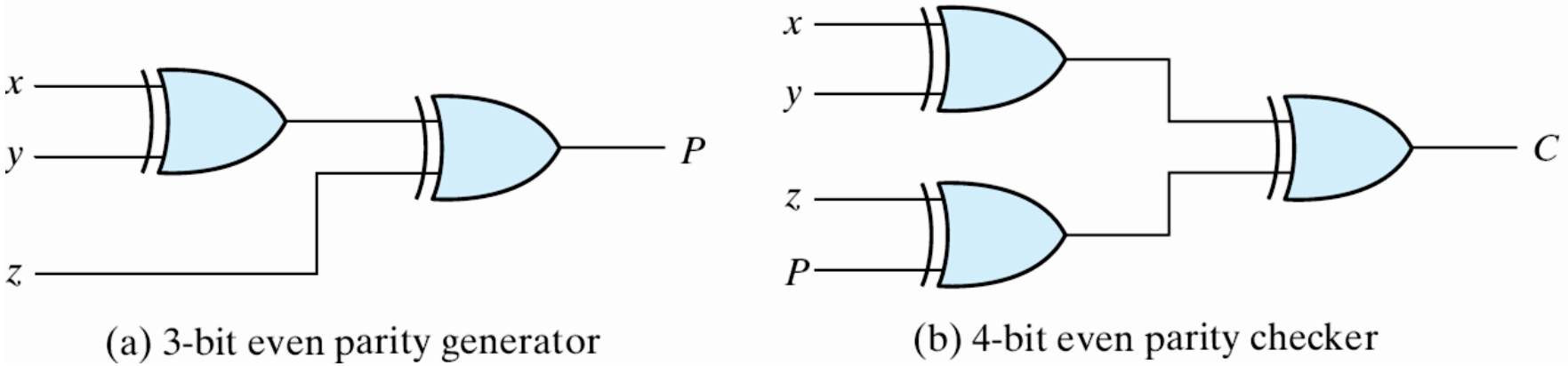


Figure 3.36 Logic Diagram of a Parity Generator and Checker

Parity Generation and Checking

Table 3.4
Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
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 3.5
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

Home Work (6)

Digital Design (**4th**)- Morris Mano-Page 116-Problems:

3.16 b, c,
3. 17
3.20
3.21
3.23

تم بحمد الله