Gate-level minimization

Computer Organization
502044

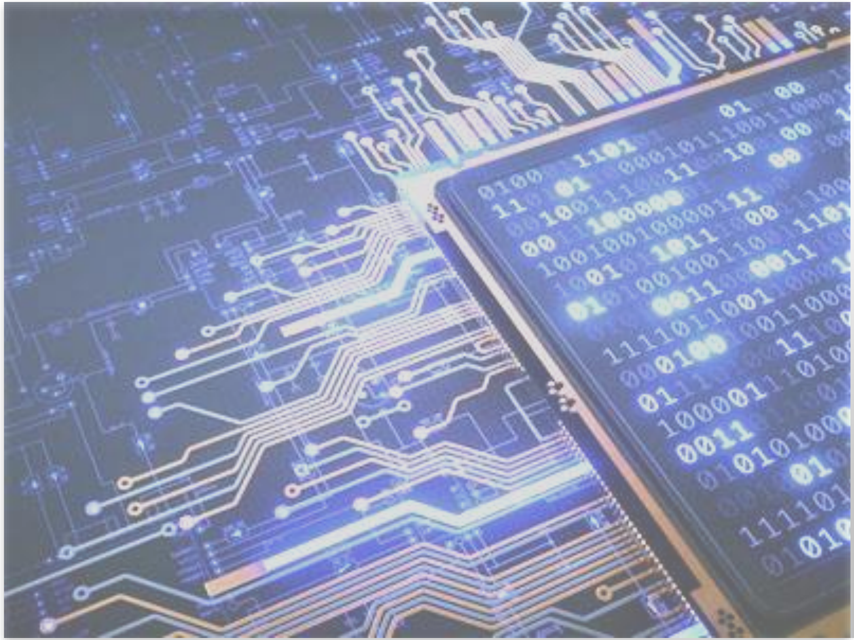
Acknowledgement

This slide show is intended for use in class, and is not a complete document. Students need to refer to the book to read more lessons and exercises. Students have the right to download and store lecture slides for reference purposes; Do not redistribute or use for purposes outside of the course.

[1] Morris R. Mano (Author), Michael D. Ciletti, [2019] **Digital Design: With an Introduction to the Verilog HDL,** chapter 3 - Gate level minimization, 5th Edition.

🖂 **trantrungtin.tdtu.edu.vn**

Chapter Objectives



1. Simplification digital circuits
2. The map method
3. Don’t care condition
4. NAND and NOR implementation
5. +Hardware language

Syllabus

3.1 Introduction

3.2 The Map Method

3.3 Four-Variable K-Map

3.4 Product-of-Sums Simplification

3.5 Don’t-Care Conditions

3.6 NAND and NOR Implementation

3.7 Other Two-Level Implementations

3.8 Exclusive-OR Function

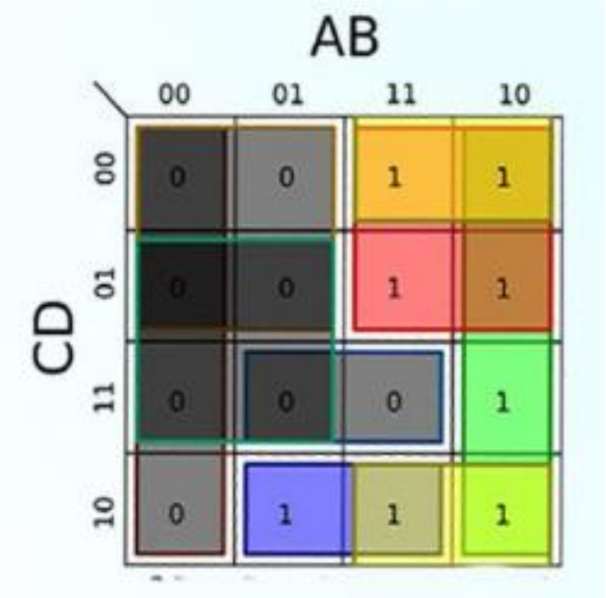
3.9 Hardware Description Language

3.1 Introduction

- Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.
- Computer-based logic synthesis tools can minimize a large set of Boolean equations efficiently and quickly.
- Why simplify?
  - Simpler expression uses fewer logic gates.
  - Thus cheaper, uses less power, (sometimes) faster.

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' = row 0; x = row 1
  - y' = column 0; y = column 1
  - A truth table in square diagram

Fig. 3.2(a): xy = m3

Fig. 3.2(b): x+y = x'y+xy' +xy =
m1+m2+m3

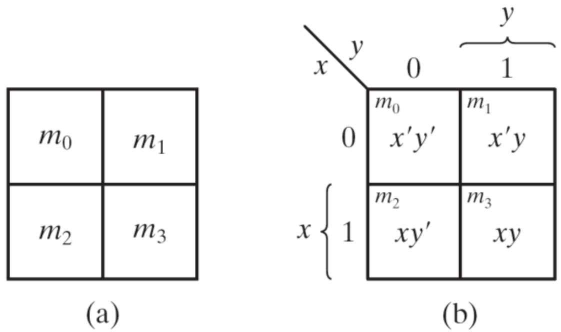


Figure 3.1 Two-variable Map

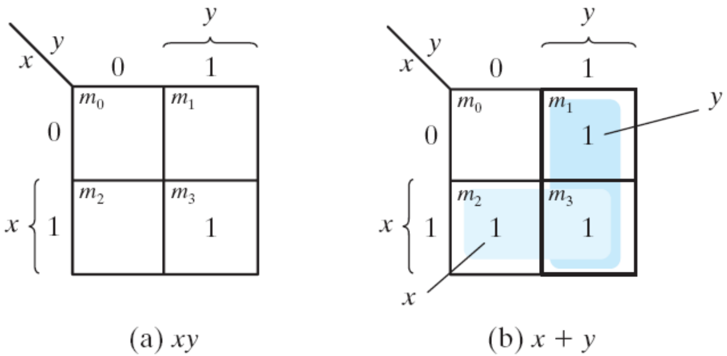


Figure 3.2 Representation of functions in the map

A Three-variable Map

- A three-variable map
  - Eight minterms
  - The Gray code sequence
  - Any two adjacent squares in the map differ by only on variable
    - Primed in one square and unprimed in the other
    - e.g., m5 and m7 can be simplified
    - m5+ m7 = xy'z + xyz = xz (y'+y) = xz

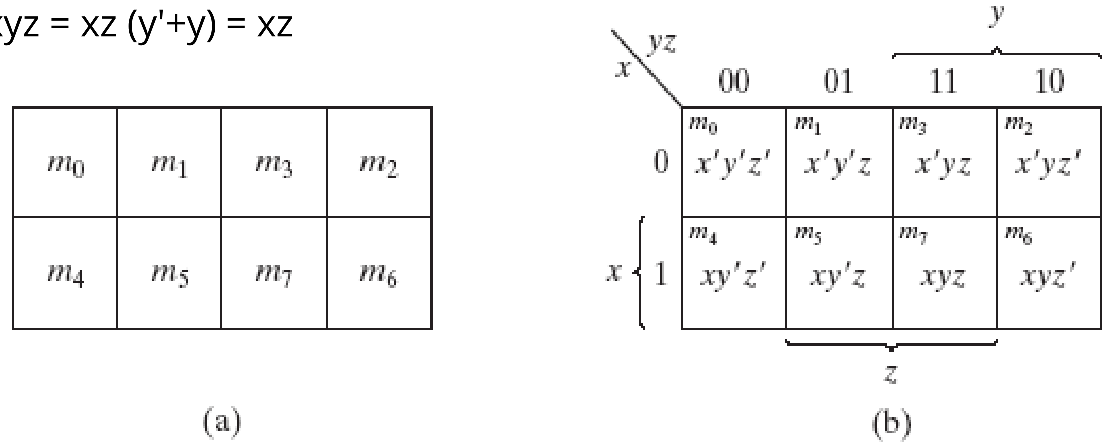


Figure 3.3 Three-variable Map

A Three-variable Map

- m0 and m2 (m4 and m6) are adjacent
- m0+ m2 = x'y'z' + x'yz' = x'z' (y'+y) = x'z'
- m4+ m6 = 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 \ xz	0 0	0 1	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
x 1	$xy'z'$	$xy'z$	xyz	xyz'

(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) = Σ(2, 3, 4, 5) = x'y + xy'

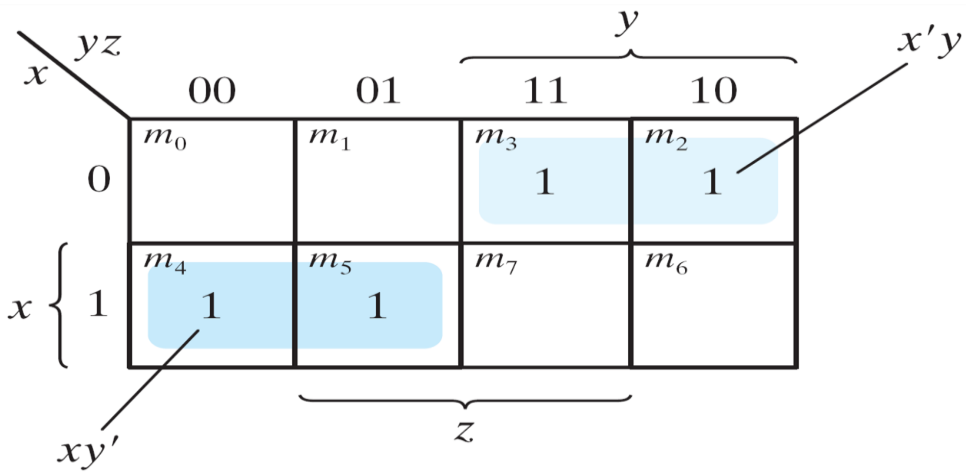


Figure 3.4 Map for Example 3.1, F(x, y, z) = Σ(2, 3, 4, 5) = x'y + xy'

Example 3.2

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

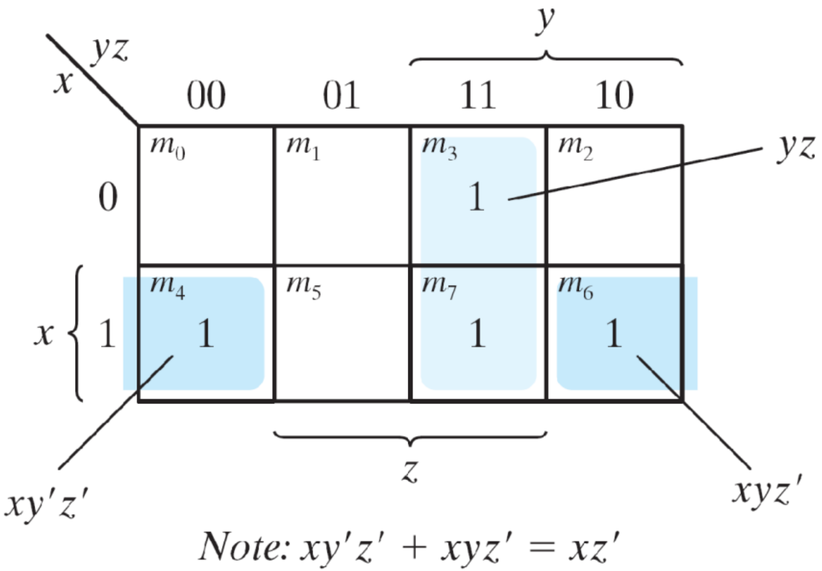


Figure 3.5 Map for Example 3-2; $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

Four adjacent Squares

- Consider four adjacent squares
  - 2, 4, and 8 squares
  - m0+m2+m4+m6 = x'y'z'+x'yz'+xy'z'+xyz' = x'z'(y'+y) +xz'(y'+y) = x'z' + xz‘ = z'
  - m1+m3+m5+m7 = x'y'z+x'yz+xy'z+xyz =x'z(y'+y) + xz(y'+y) =x'z + xz = z

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

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

(b)

Figure 3.3 Three-variable Map

Example 3.3

- 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'

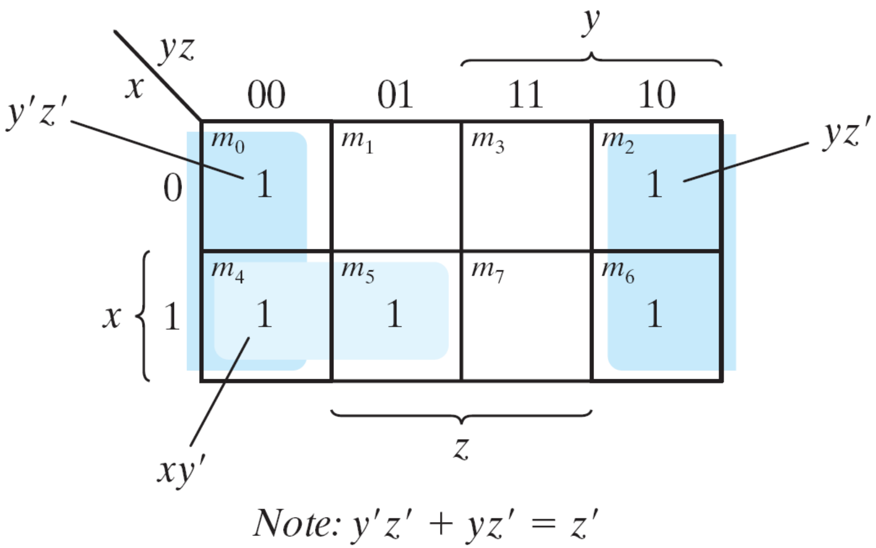


Figure 3.6 Map for Example 3-3, F(x, y, z) = Σ(0, 2, 4, 5, 6) = z′ +xy′

Example 3.4

- Example 3.4: let **F = A'C + A'B + AB'C + BC**
    - Express it in sum of minterms.
    - Find the minimal sum of products expression.
    - Ans:

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

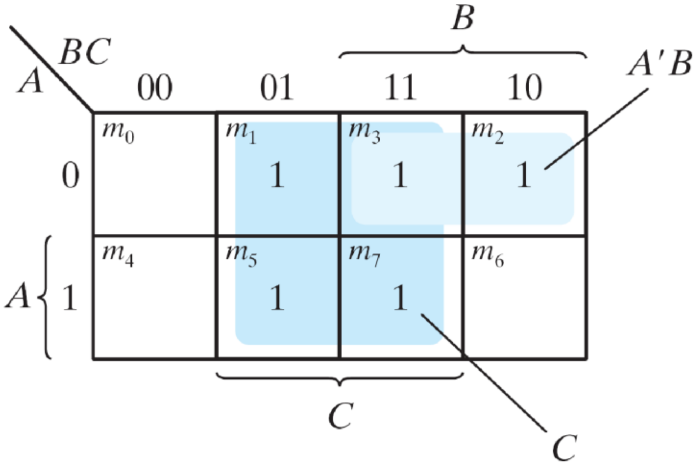


Figure 3.7 Map for Example 3.4, A'C + A'B + AB'C + BC = C + A'B

3.3 Four-Variable Map

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

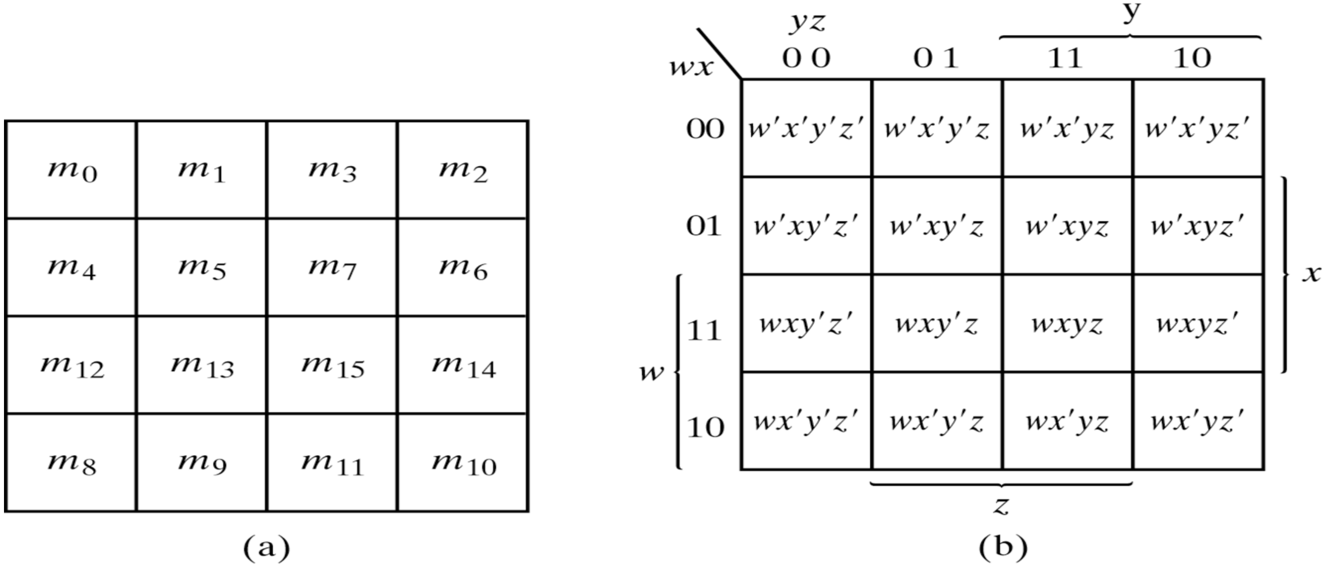
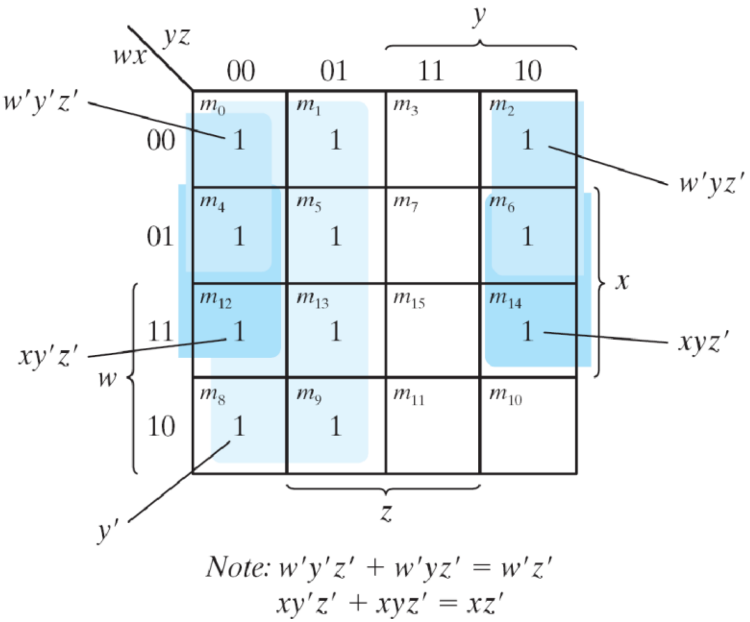


Figure 3.8 Four-variable Map

Example 3.5

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

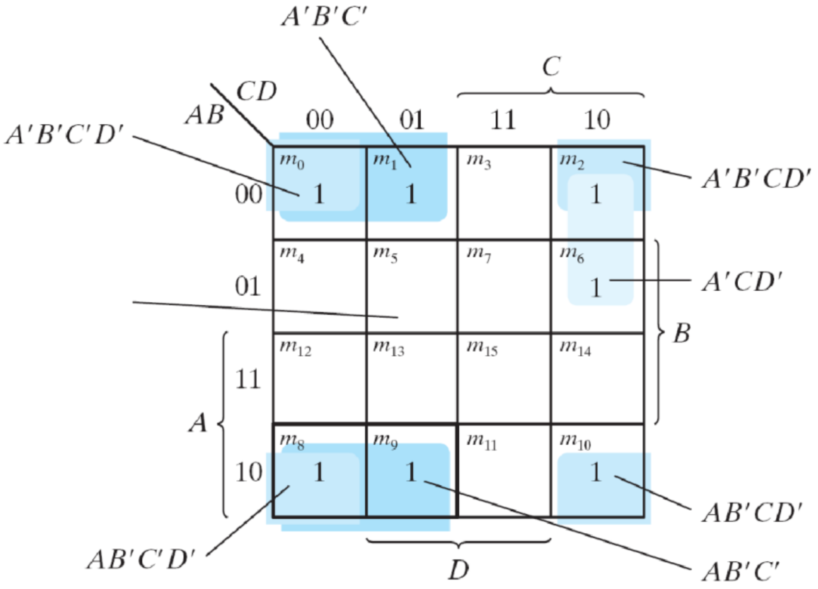


$$\longrightarrow \quad F = y'+w'z'+xz'$$

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

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



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

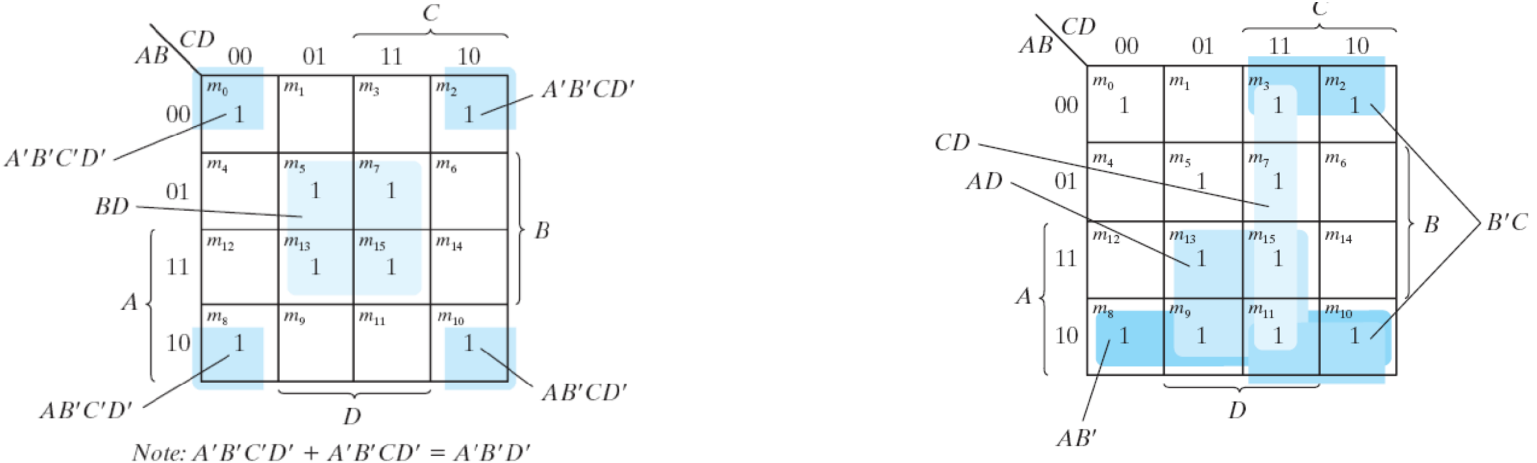
Figure 3.9 Map for Example 3-6; A'B'C' + B'CD' + A'B'C'D' + AB'C'= B'D' + B'C' +A'CD'

Prime Implicants

- 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) = Σ(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)
  - The simplified expression may not be unique
  - F = BD+B'D'+CD+AD = BD+B'D'+CD+AB'
  - = BD+B'D'+B'C+AD = BD+B'D'+B'C+AB'



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

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

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

Figure 3.11 Simplification Using Prime Implicants

3.4 Five-Variable Map

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

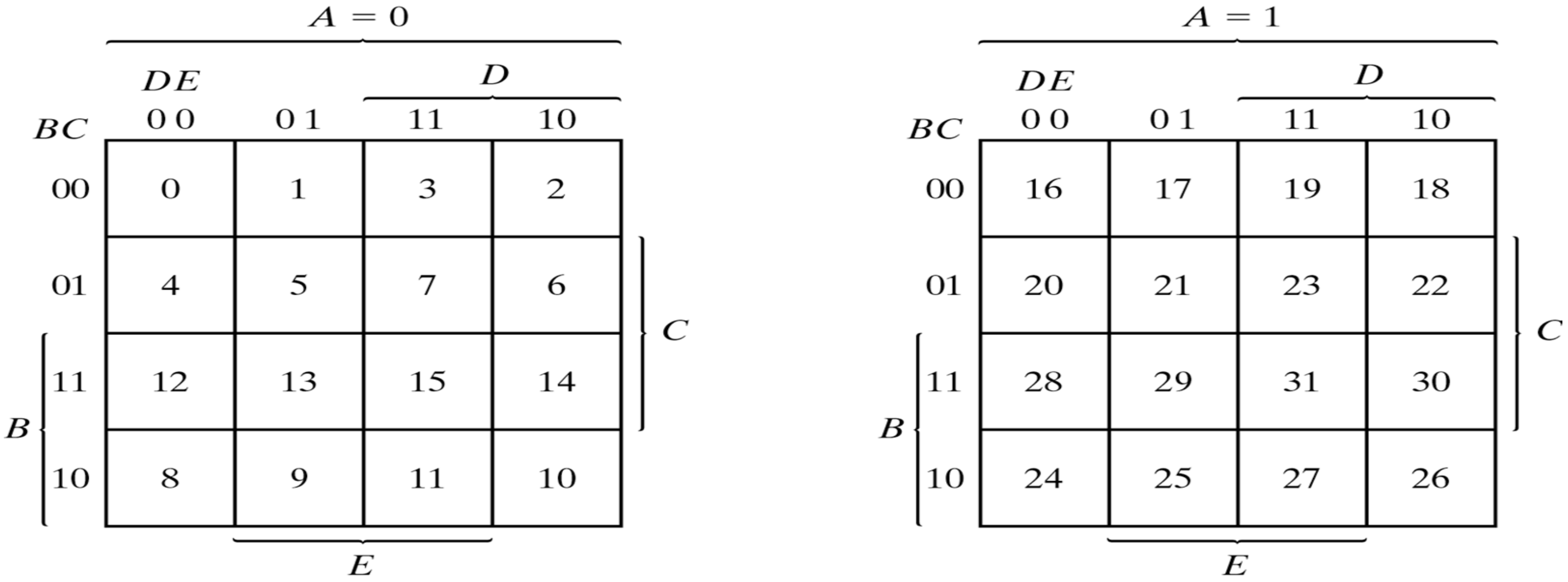


Figure 3.12 Five-variable Map

Notes: number of literals in a term

- 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

	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 3.7

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

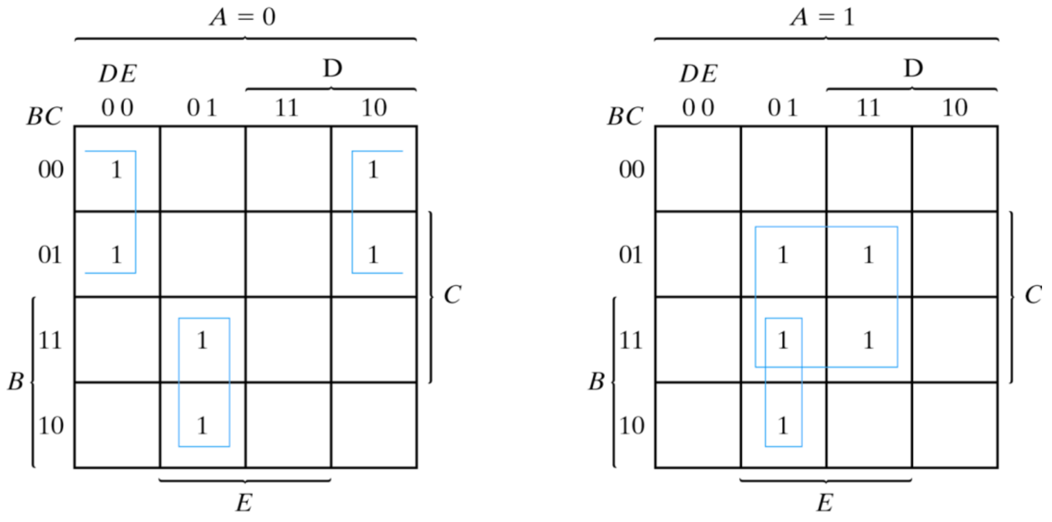


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

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

Example 3.7 (cont.)

- Another Map for Example 3-7

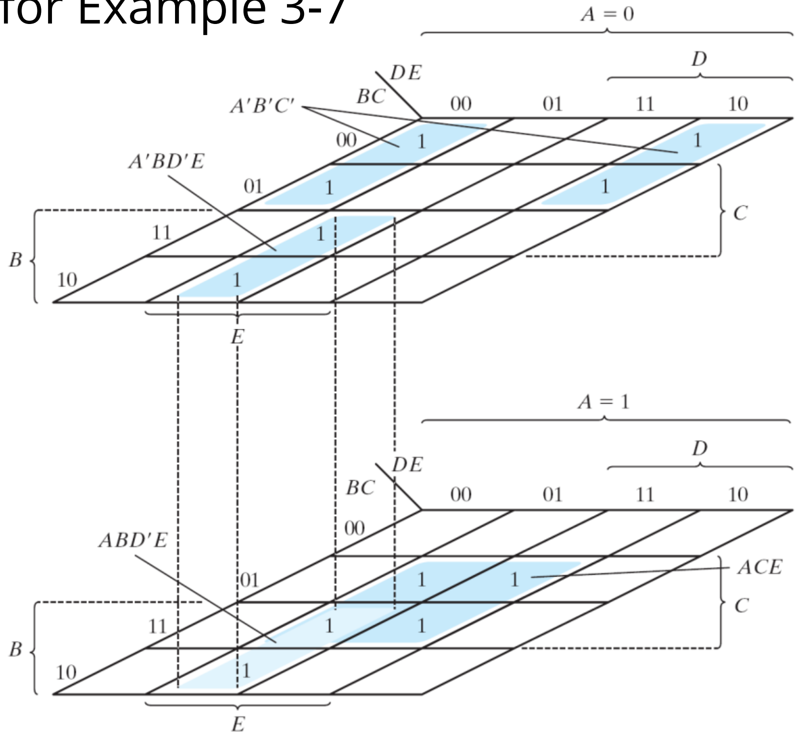


Figure 3.13 Map for Example 3.7, F = A'B'E'+BD'E+ACE

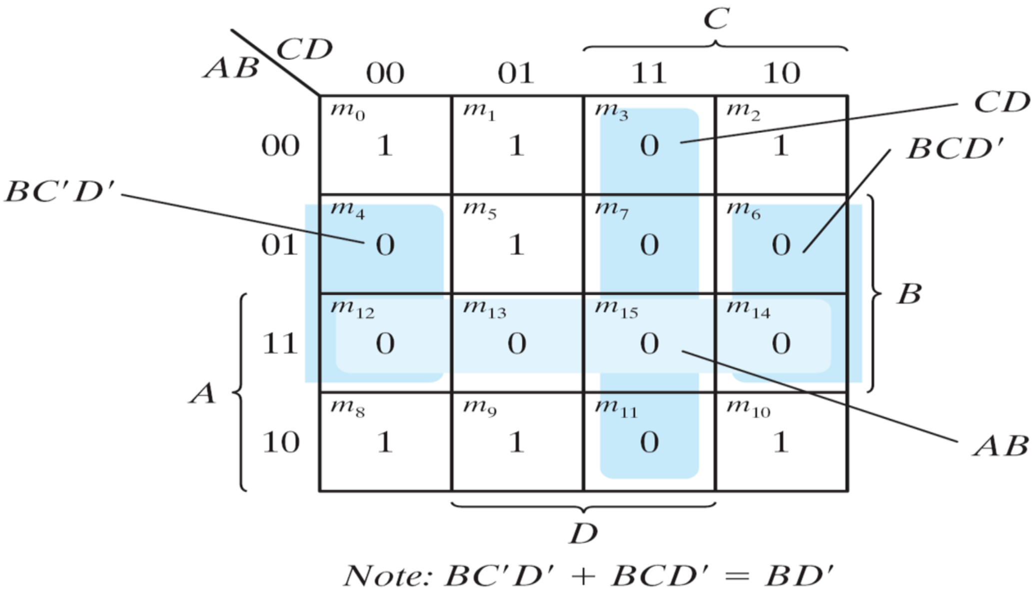
3.5 Product of Sums simplification

- 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)
  - M0M1 = (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 = Σ(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) $F' = AB+CD+BD'$

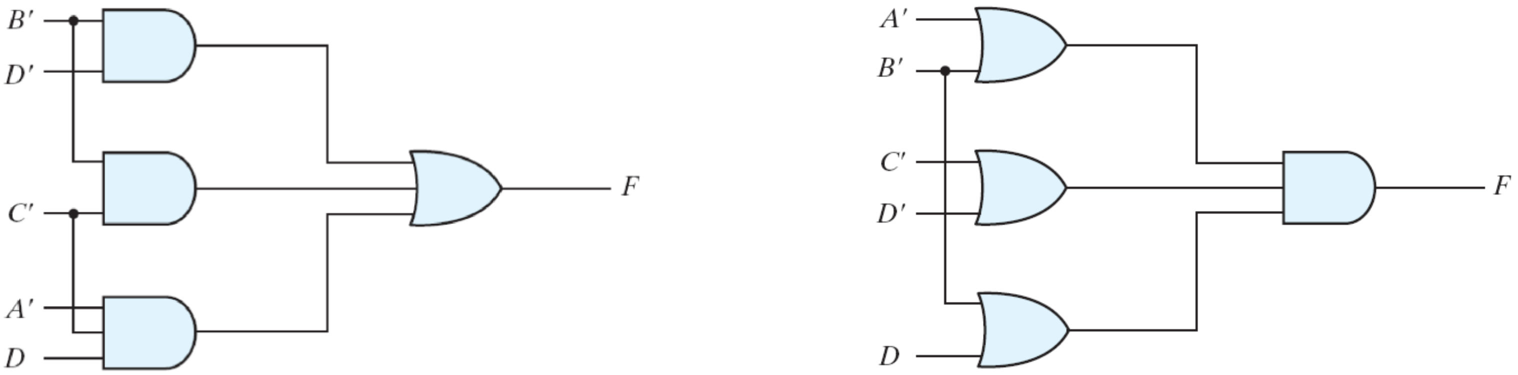
   » Apply DeMorgan's theorem; $F=(A'+B')(C'+D')(B'+D)$

   » 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



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

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

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:

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

  - In sum-of-maxterm:

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

  - Taking the complement of F'

$$\boxed{F(x,y,z)=(x'+z')(x+z)}$$

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:

$$\boxed{F(x, y, z) = x'z + xz'}$$

  - Combine the 0's :

$$\boxed{F'(x, y, z) = xz + x'z'}$$

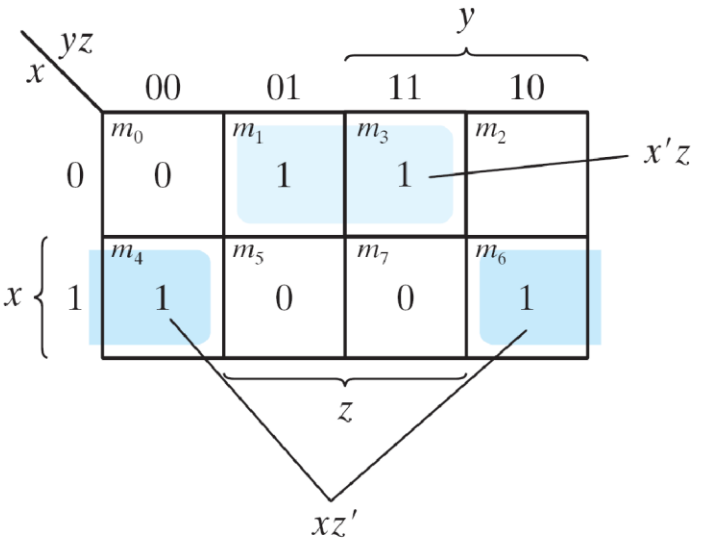


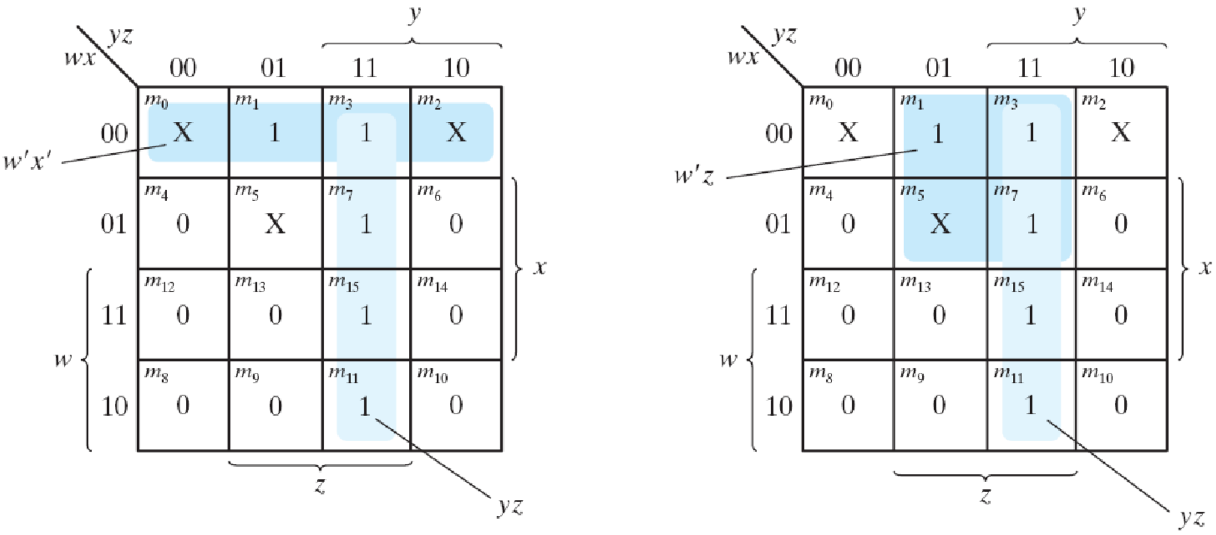
Figure 3.16 Map for the function of Table 3.2

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 = Σ(0, 1, 2, 3, 7, 11, 15) ; F = Σ(1, 3, 5, 7, 11, 15)
- Either expression is acceptable



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

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

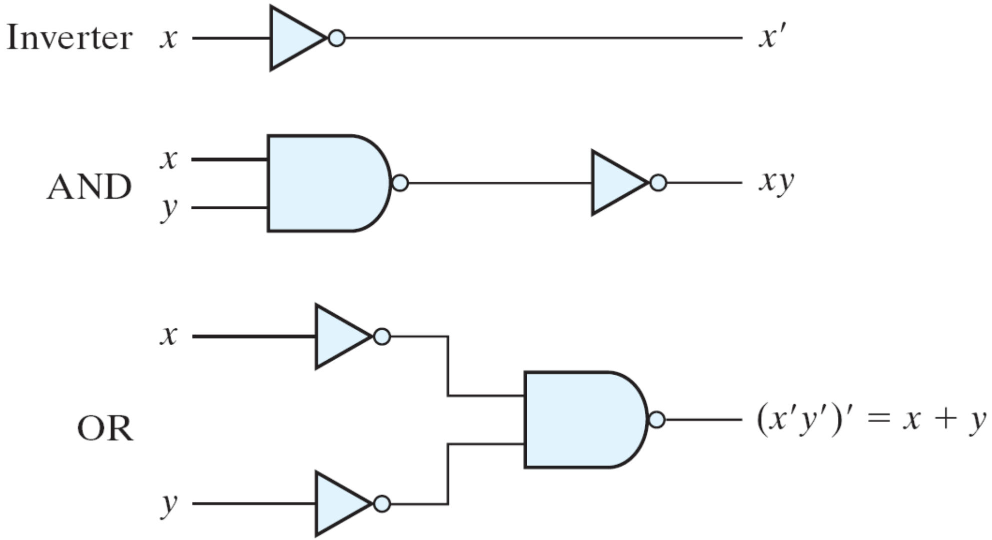


Figure 3.18 Logic Operations with NAND Gates

NAND Gate

- Two graphic symbols for a NAND gate

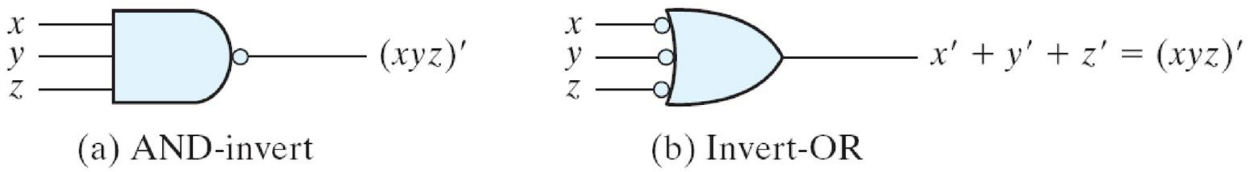


Figure 3.19 Two Graphic Symbols for NAND Gate

Two-level Implementation

Two-level logic

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

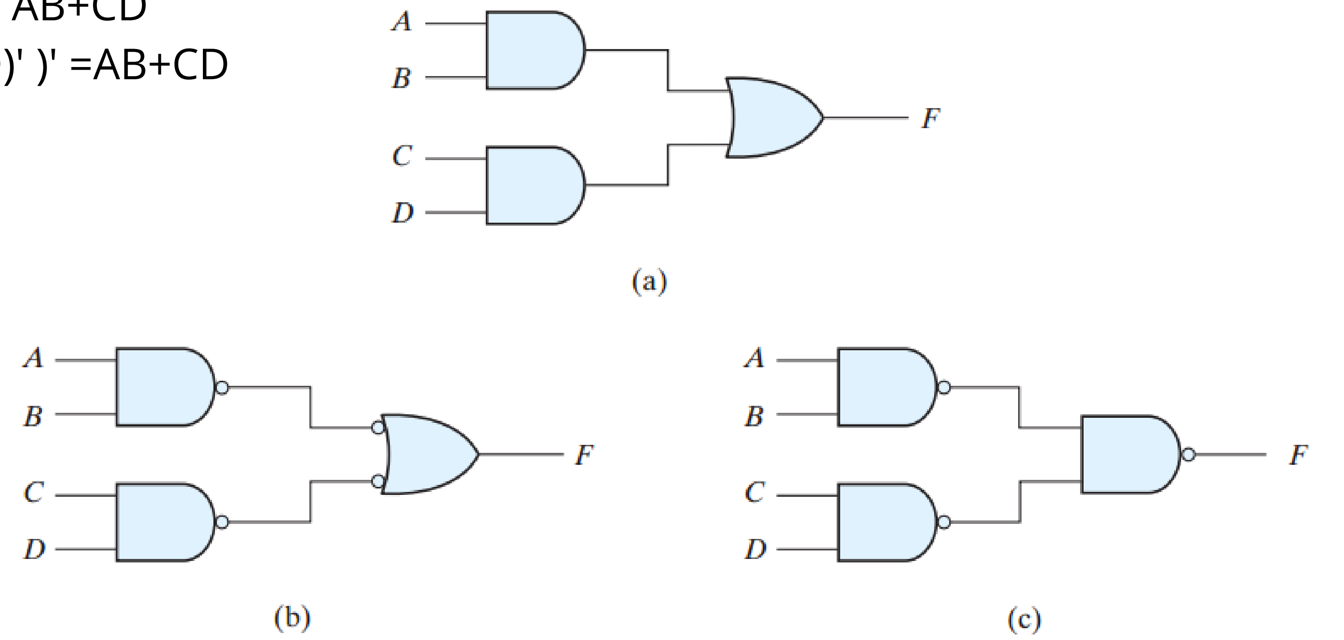


Figure 3.20 Three ways to implement F = AB + CD

Example 3.10

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

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

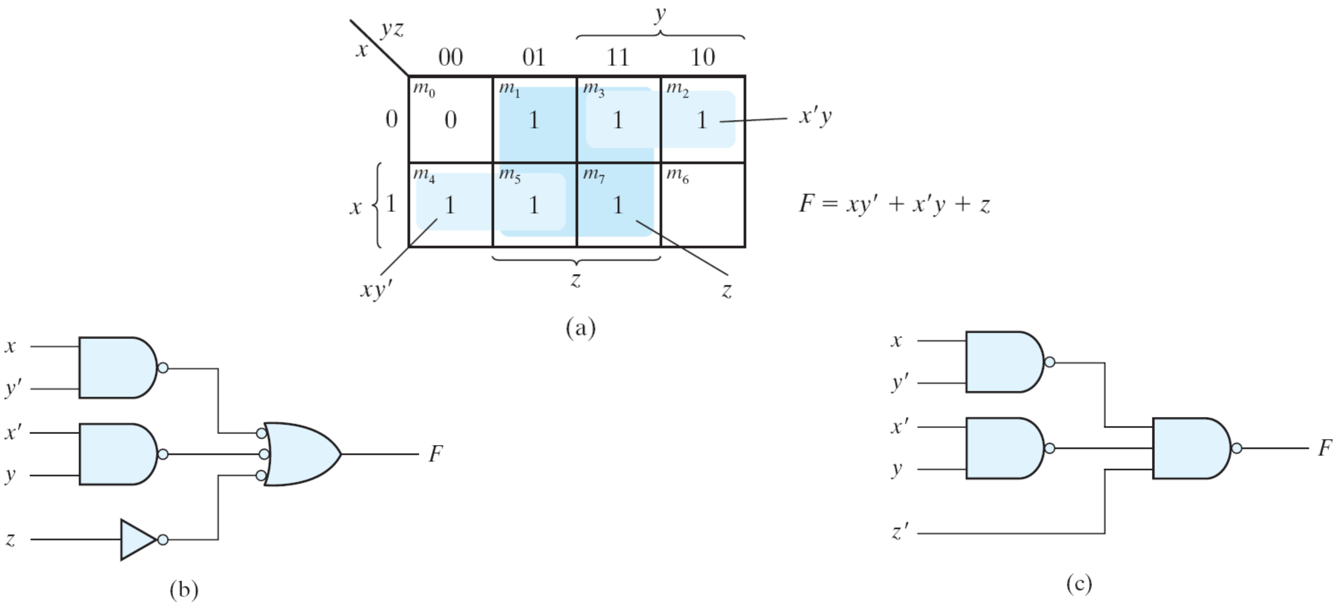


Figure 3.21 Solution to Example 3-10

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.

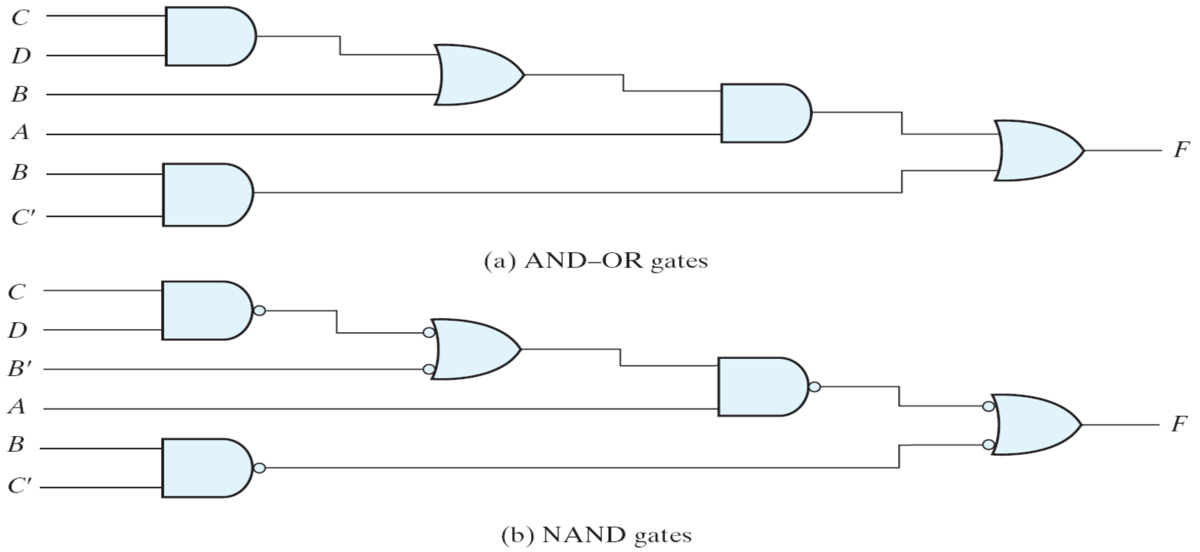


Figure 3.22 Implementing $F = A(CD + B) + BC'$

NAND Implementation

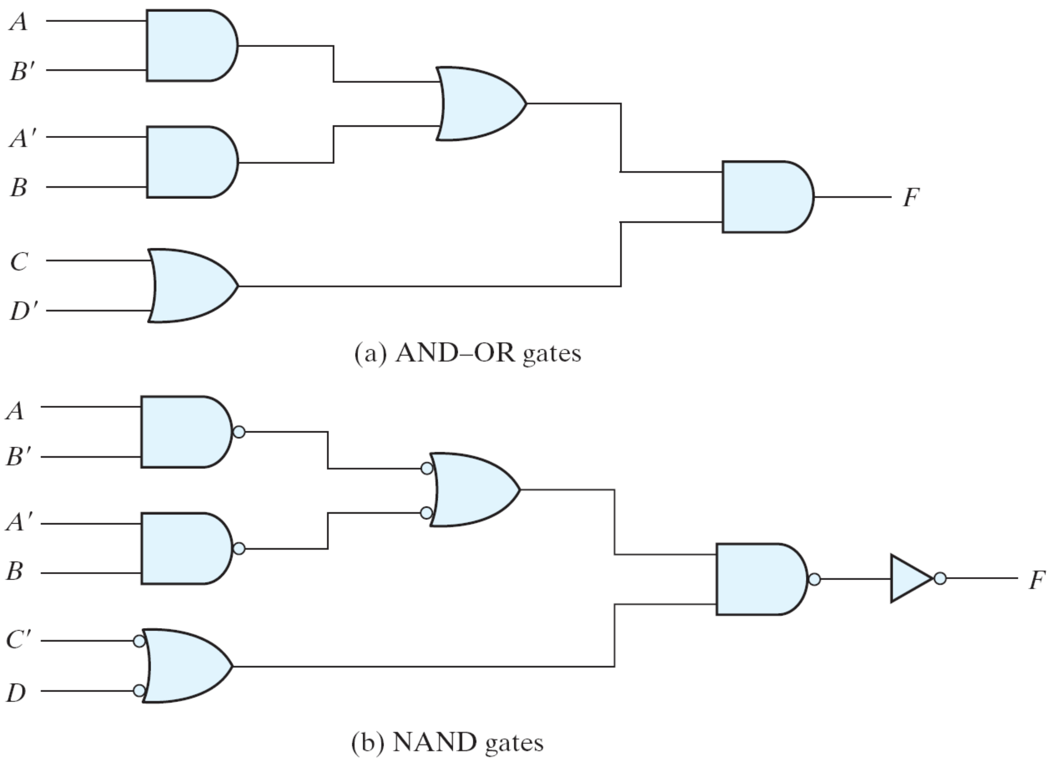


Figure 3.23 Implementing F = (AB′ +A′B)(C+ D′)

NOR Implementation

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

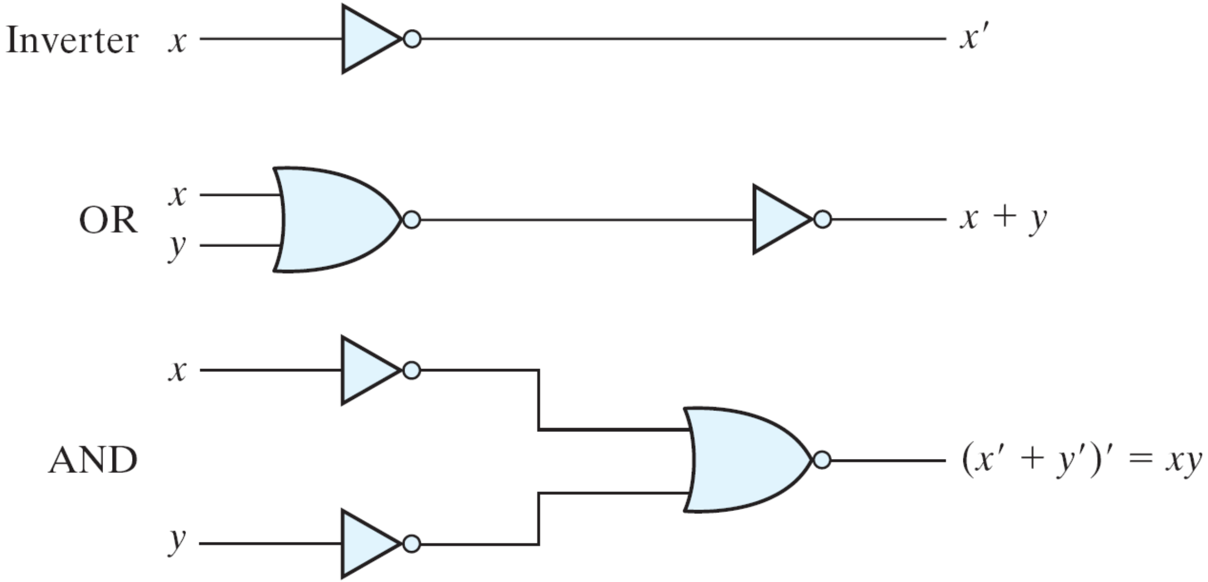


Figure 3.24 Logic Operation with NOR Gates

Two Graphic Symbols for a NOR Gate

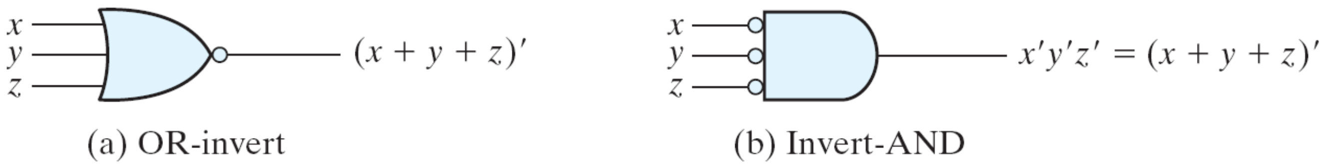


Figure 3.25 Two Graphic Symbols for NOR Gate

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

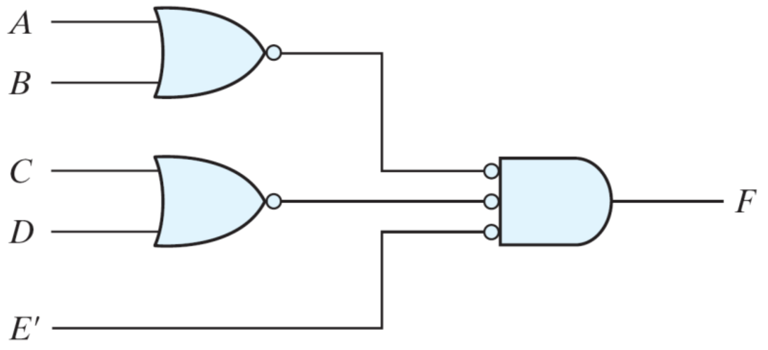


Figure 3.26 Implementing $F = (A + B)(C + D)E$

Example

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

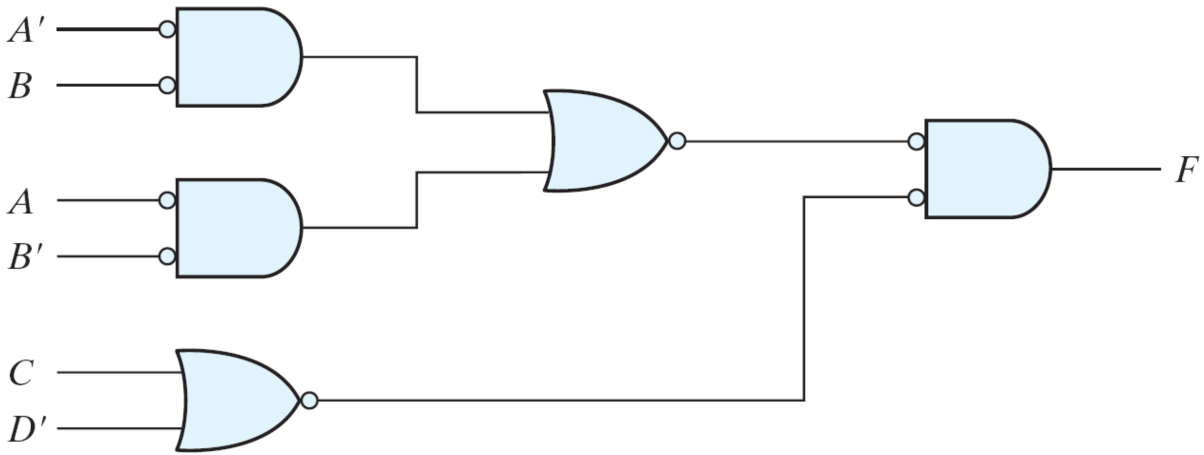


Figure 3.27 Implementing F = (AB' +A'B)(C + D') with NOR gates