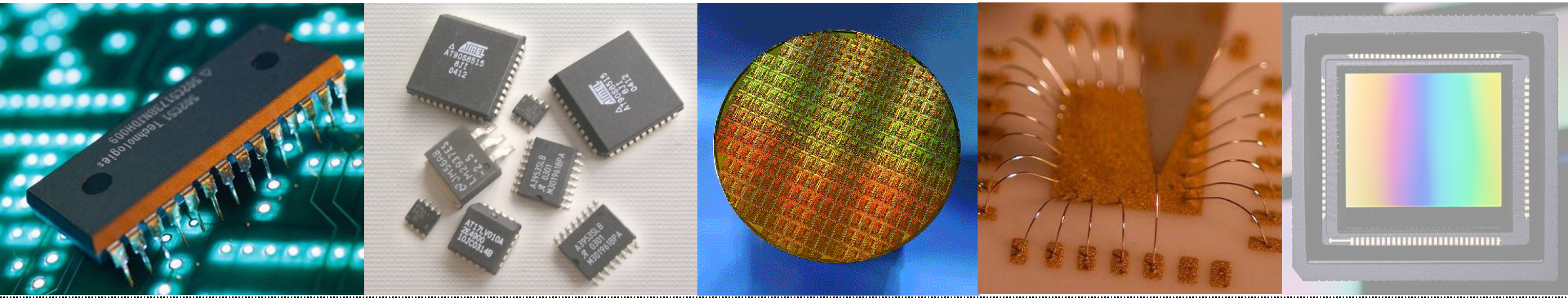


3. Gate-Level Minimization

3.1 Introduction

- The complexity of digital logic gates that implement a Boolean function
  - directly related to the function expression
- So, we want to find the simplest possible expression of a given function
- Method based on Truth Table
  - K-map (Karnaugh map)

3.2 The Map Method

- Two-Variable Map

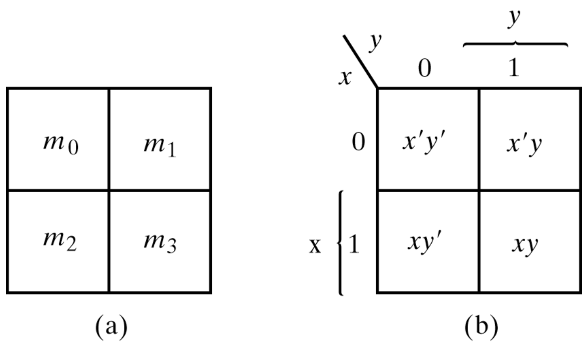


Fig. 3-1 Two-variable Map

- m1+m2+m3 = x'y +xy' +xy = x +y

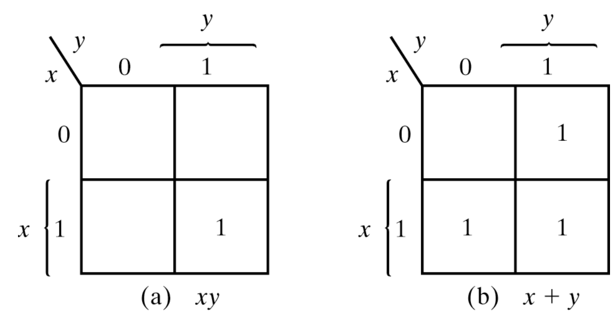


Fig. 3-2 Representation of Functions in the Map

- Three-Variable Map **<u>(Careful! bit sequence is Gray-coded – adjacent entries differ by only one bit)</u>**

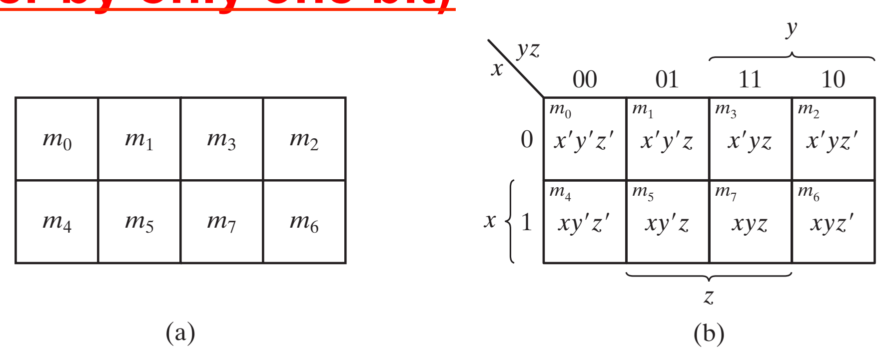


FIGURE 3.3
Three-variable K-map

3.2 The Map Method

- Ex 3-1) Simplify the Boolean function, F(x, y, z) = Σ(2, 3, 4, 5)

  F = x'y + xy'

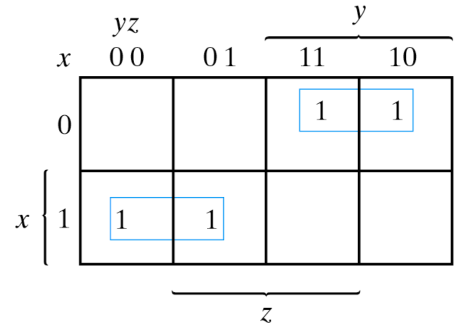


Fig. 3-4 Map for Example 3-1; $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

- Ex 3-4) Given Boolean function, F = A'C + A'B +AB'C +BC

  a) express it in sum of minterms

  F(x, y, z) = Σ(1, 2, 3, 5, 7)

  b) find the minimal sum of products

  F = C + A'B

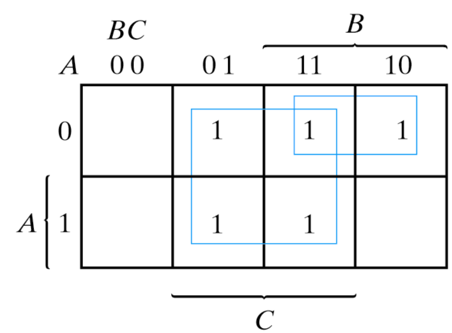
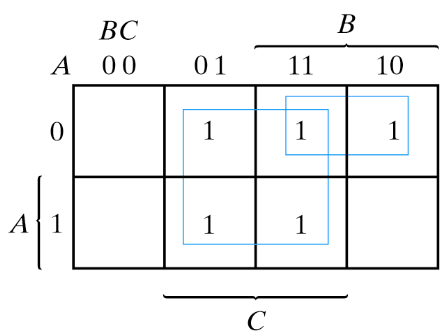


Fig. 3-7 Map for Example 3-4; $A'C + A'B + AB'C + BC = C + A'B$

1. **Find the largest possible rectangle – size 1,2,4,8,16**
2. **Find the expressions to represent the outputs of the rectangle**

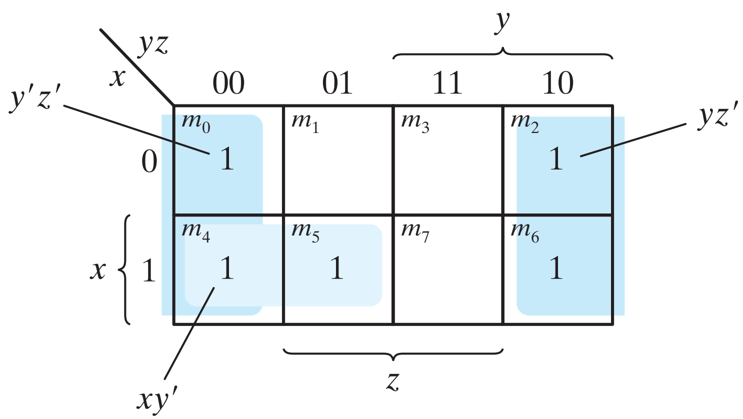
3.2 The Map Method

- Rectangle should be of size 2^n, n=0,1,2,… for example, 1,2,4,8,16
- Why? To remove n variables, 2^n terms are removed
- ex) x’y + x y = (x’+x) y = y => 2 terms removed – 1 variable removed
- ex) abc+a’bc+ab’c+a’b’c = (ab+a’b+ab’+a’b’)c = c => 4 terms removed – 2 variable s removed
- ex) size-4 rectangle
  - Its minterm expression:
  - A’B’C + A’BC + AB’C + ABC = C



- Simplify the Boolean function, F(x, y, z) = Σ(0,2,4,5,6)

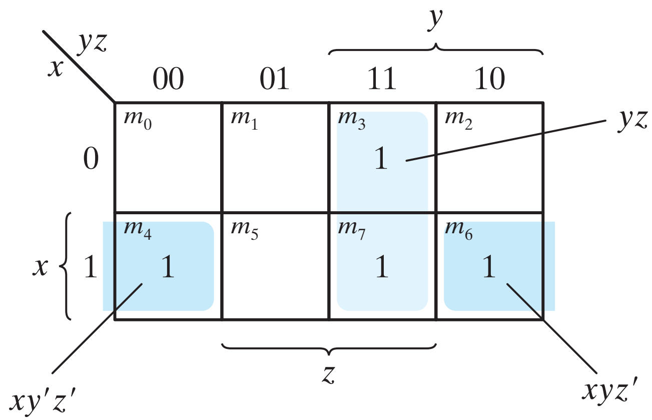
  F = z' + xy'



- Simplify

  F(x, y, z) = Σ(3, 4, 6, 7)

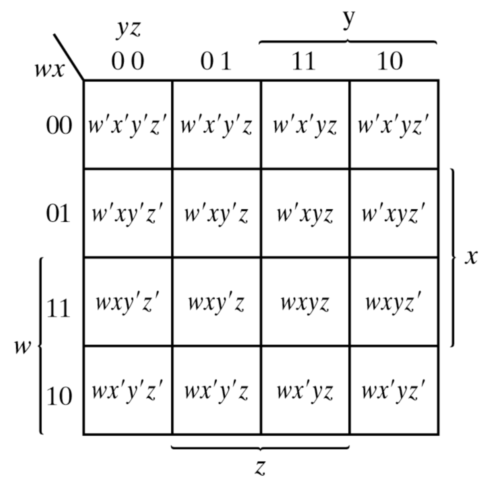
  F = yz + xz'



3.3 Four-Variable Map

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)



(b)

Fig. 3-8 Four-variable Map

Ex 3-5) Simplify the Boolean function,

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

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

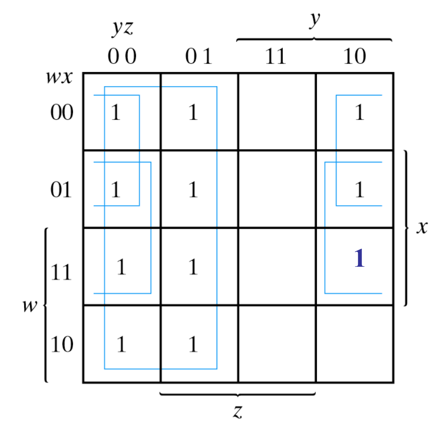
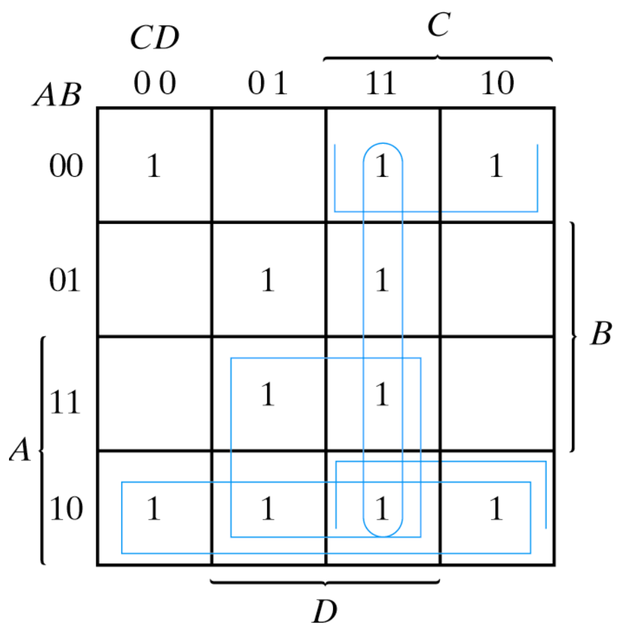


Fig. 3-9 Map for Example 3-5; $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

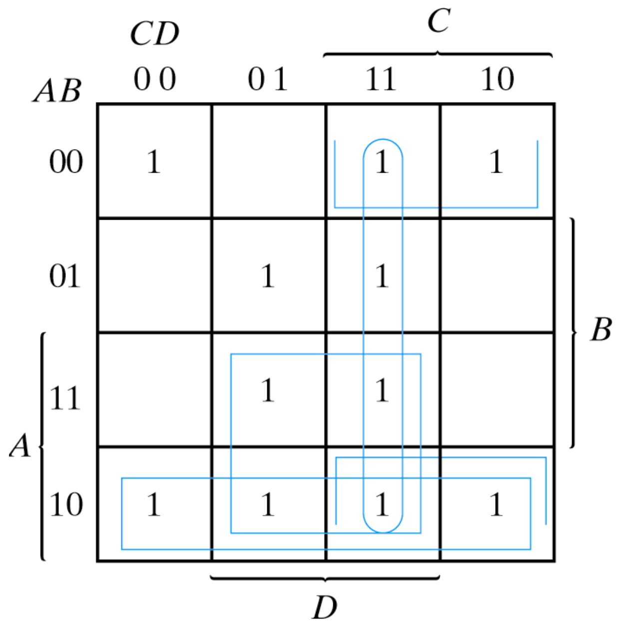
3.3 Four-Variable Map – Prime Implicants

- F(A,B,C,D) = Σ(0,2,3,5,7,8,9,10,11,13,15)



- Prime implicant: the largest rectangles, can be of size 1,2,4,8 or 16.
- Essential prime implicant: prime implicant which contains a square which i s not covered by other implicants

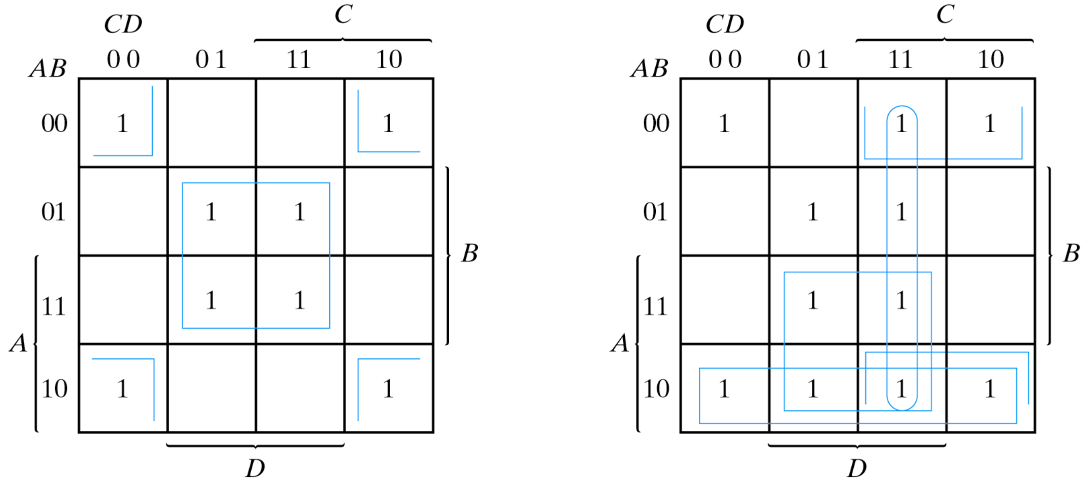
- F(A,B,C,D) = Σ(0,2,3,5,7,8,9,10,11,13,15)



- There are 6 prime implicants
- there are 2 essential prime implicants

3.3 Four-Variable Map – Prime Implicants

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



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

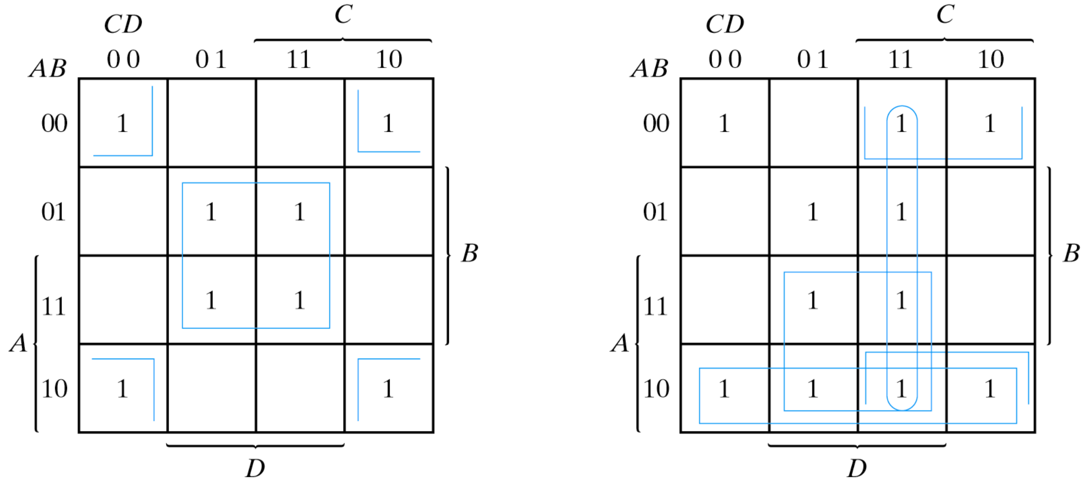
(b) Prime implicants CD, B′C AD, and AB′

Fig. 3-11 Simplification Using Prime Implicants

- Procedure:
  1. Find essential prime implicants, and derive corresponding minterms
  2. Find other prime implicants which contain uncovered squares in Step 1, and derive minterms

3.3 Four-Variable Map – Prime Implicants

- F(A,B,C,D) = Σ(0,2,3,5,7,8,9,10,11,13,15)



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

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

Fig. 3-11 Simplification Using Prime Implicants

F=BD+B'D'+CD+AD

=BD+B'D'+CD+AB'

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

=BD+B'D'+CD+AB'

3.4 Five-Variable Map (NOT COVERED)

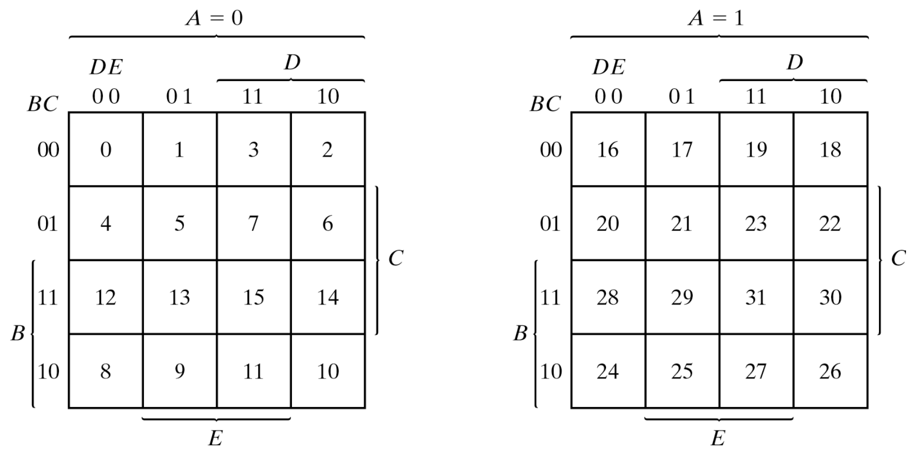


Fig. 3-12 Five-variable Map

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

3.4 Five-Variable Map (NOT COVERED)

- Ex 3-7) Simplify the Boolean function,
  $F(A,B,C,D,E)=\Sigma(0,2,4,6,9,13,21,23,25,29,31)$

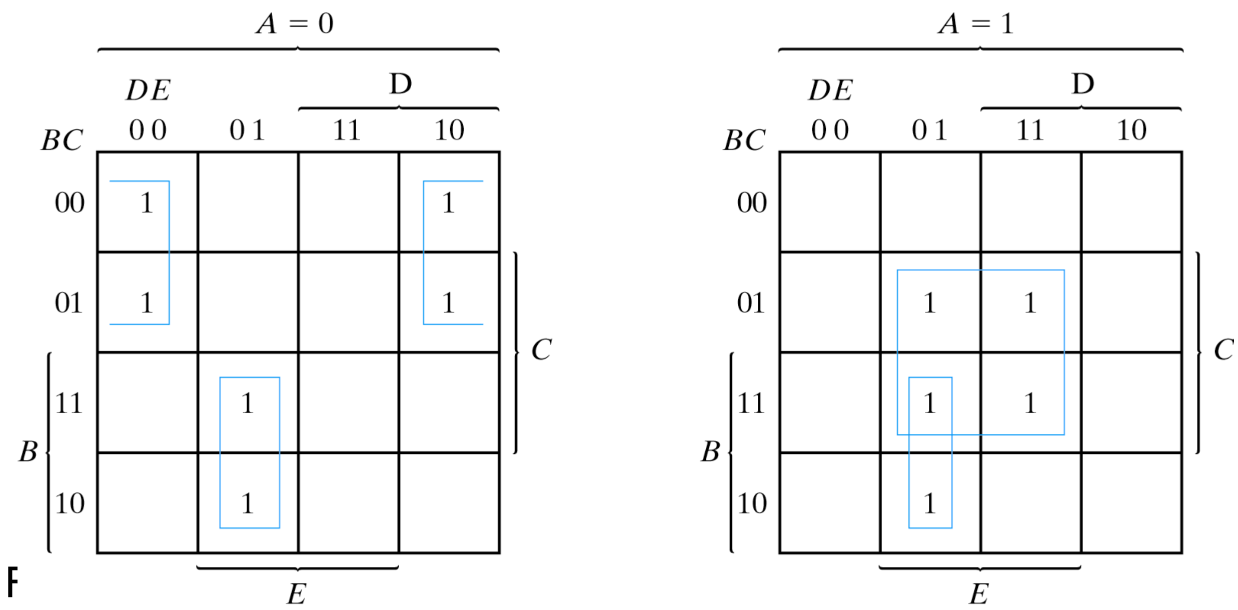


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

3.5 Product of Sums Simplification

- Ex 3-8) Simplify the Boolean function,

  $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 7, 9, 10)$

  a) sum of products

  $F = B'D' + B'D' + A'C'D'$

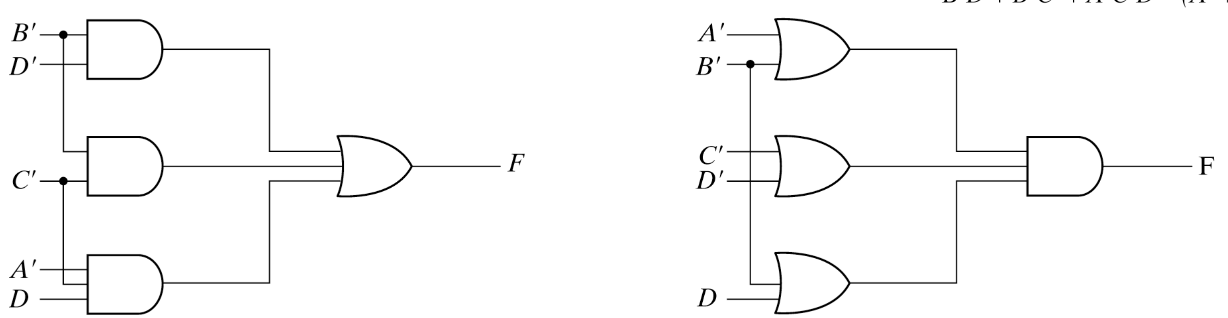
  b) product of sum

  $F' = AB + CD + BD'$ (use K-map for 0 entries)

  $F = (A' + B')(C' + D')(B' + D)$ (deMorgan's law)

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

Fig. 3-14 Map for Example 3-8; $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$ $= B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$



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

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

Fig. 3-15 Gate Implementation of the Function of Example 3-8

3.5 Product of Sums Simplification

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

x \ yz	00	01	11	10
0	0	1	1	0
1	1	0	0	1

Fig. 3-16 Map for the Function of Table 3-2

- F(x, y, z) = Σ(1, 3, 4, 6) = Π(0, 2, 5, 7)

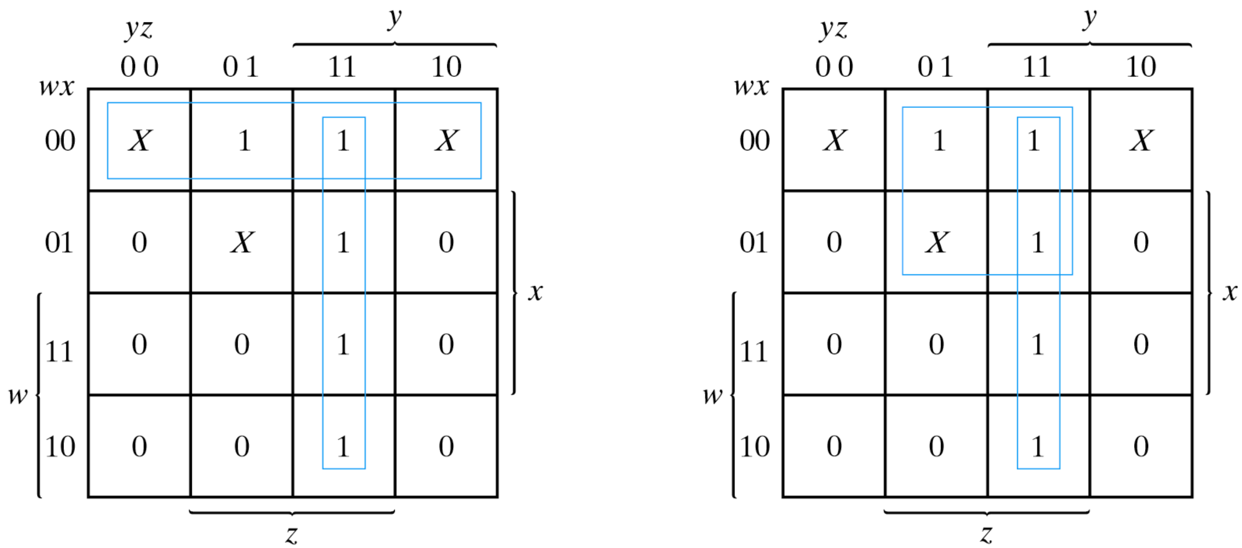
  F = x'z +xz'

  F' = xz +x'z'

  F = (x'+z')(x + z)

3.6 Don't-Care Conditions (~~NOT COVERED~~) Covered!

- Ex 3-9) Simplify the Boolean function, F(w, x, y, z) = Σ(1,3,7,11,15)
  Don't-care conditions, d(w, x, y, z) = Σ(0, 2, 5)



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

Fig. 3-17 Example with don't-care Conditions

F(w, x, y, z) = yz + w'x' = Σ(0, 1, ,2, 3, 7, 11, 15)

F(w, x, y, z) = yz + w'z = Σ(1, 3, 5, 7, 11, 15)

3.7 NAND and NOR Implementation – NAND Circuit

1. **Every function can be implemented by using only NAND gates**
2. **NAND gates are simple to implement in hardware**
   - **(same argument for NOR)**

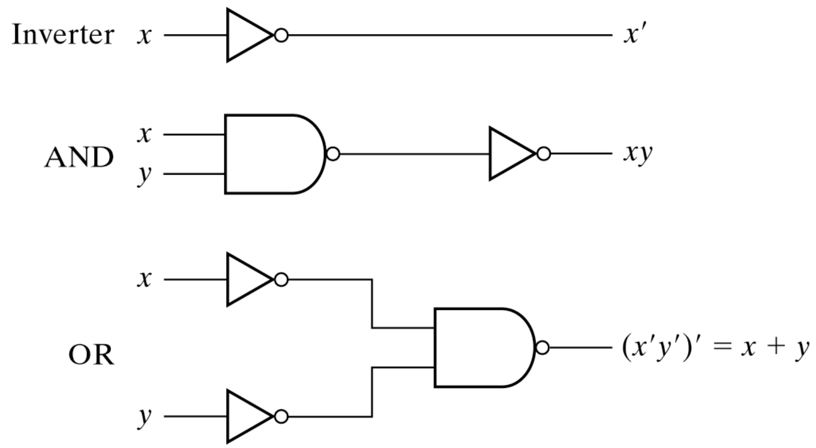
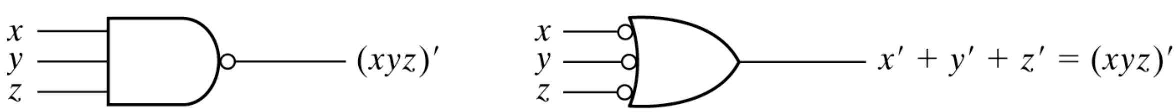


Fig. 3-18 Logic Operations with NAND Gates



(a) AND–invert (b) Invert–OR

Fig. 3-19 Two Graphic Symbols for NAND Gate

3.7 NAND and NOR Implementation – Two Level Implementation

- F = ((AB)'(CD)')' = AB + CD

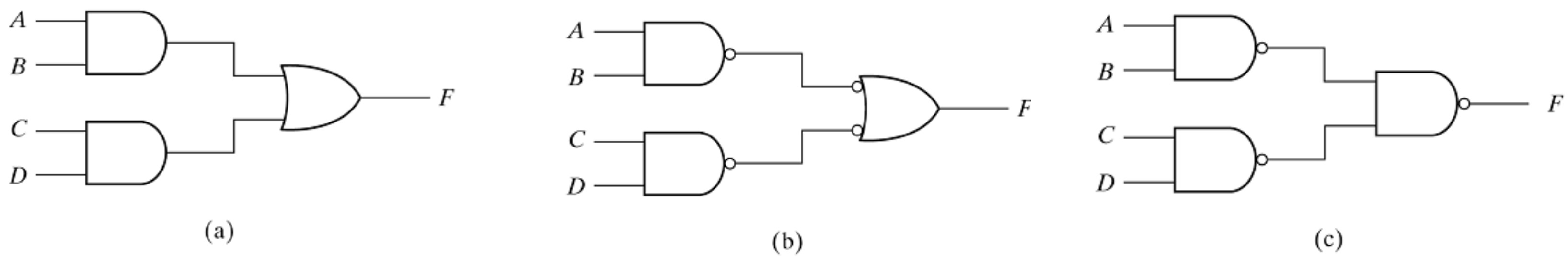


Fig. 3-20 Three Ways to Implement $F = AB + CD$ **(b) and (c) uses NAND gates**

- Ex 3-10) Implement the following Boolean function with NAND gates:

  F(x, y, z) = Σ(1, 2, 3, 4, 5, 7) = xy' + x'y + z

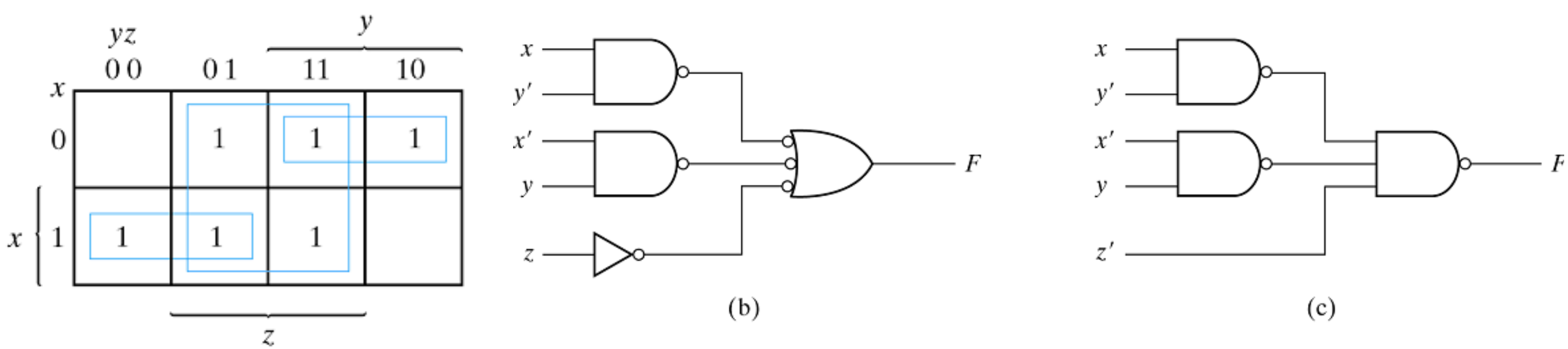


Fig. 3-21 Solution to Example 3-10

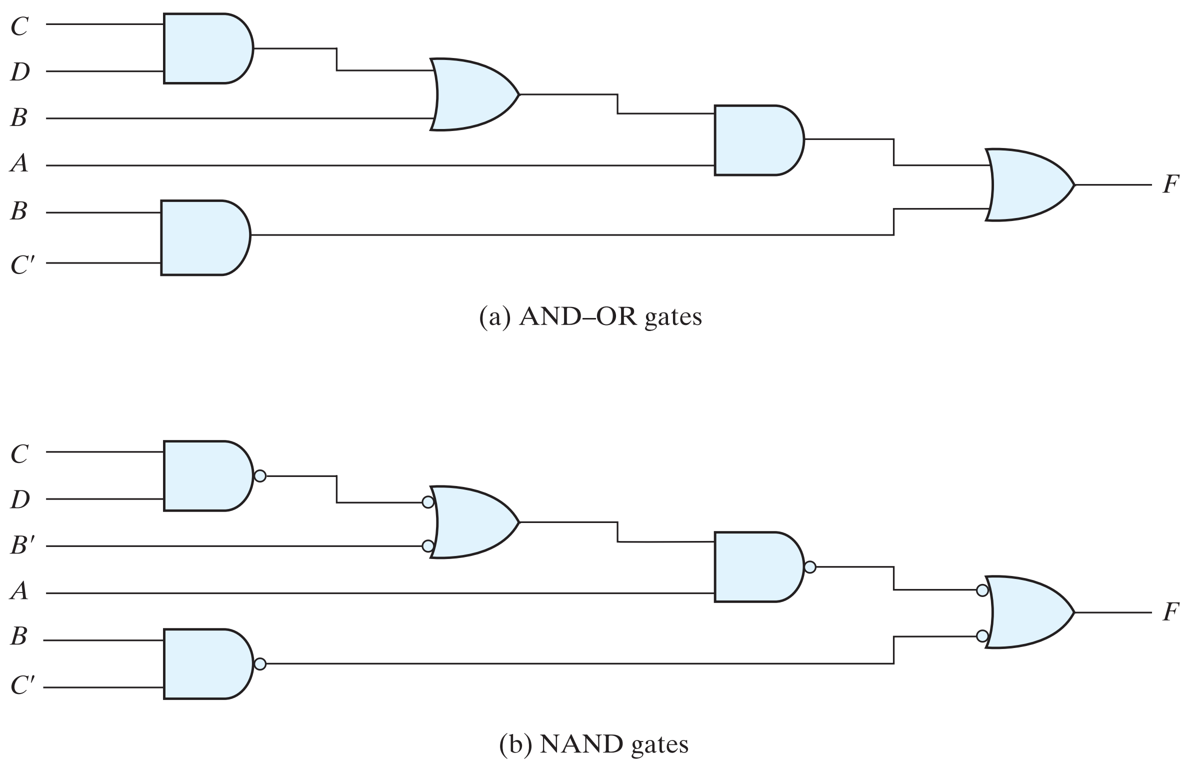


FIGURE 3.20
Implementing $F = A(CD + B) + BC'$

How to convert to NAND-only circuit

1. change AND to NAND (output bubble)
2. change OR to NAND (input bubbles)
3. check if bubbles cancel each other; if not, put inverters

NOR Implementation

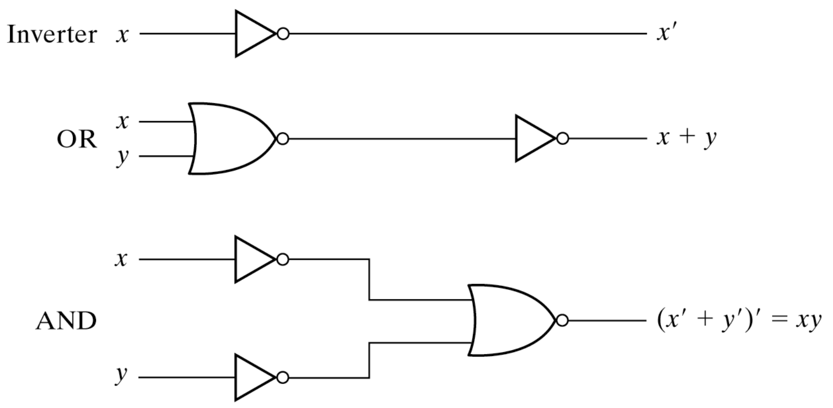


Fig. 3-24 Logic Operations with NOR Gates

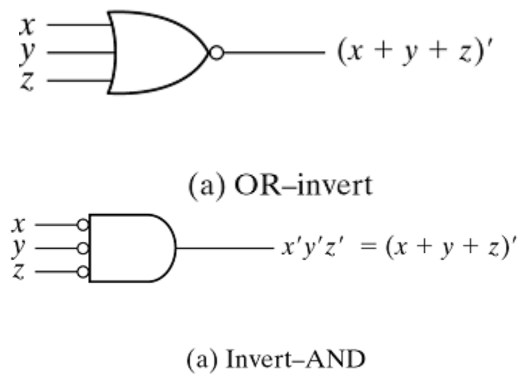


Fig. 3-25 Two Graphic Symbols for NOR Gate

F = (AB' + A'B)(C + D')

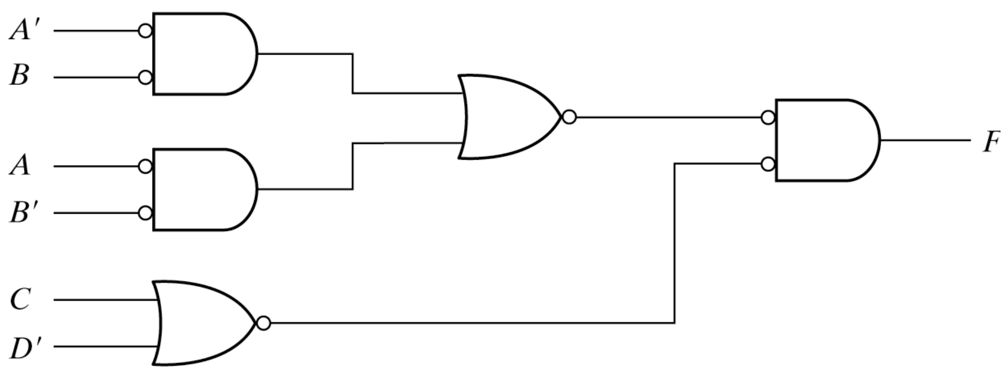
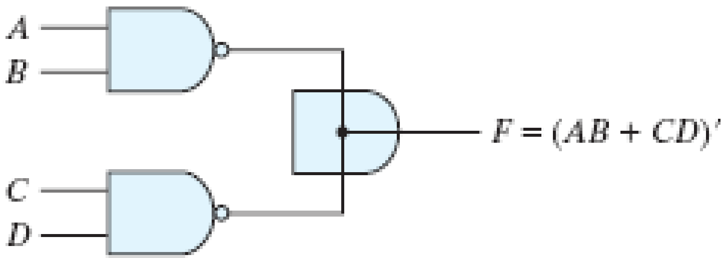


Fig. 3-27 Implementing $F = (AB' + A'B)(C + D')$ with NOR Gates

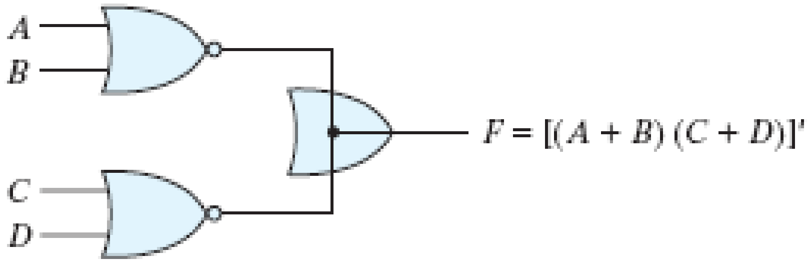
3.8 Other Two-Level Implementation (NOT COVERED)

- (a) $F = (AB)' \cdot (CD)' = (AB + CD)'$
- (b) $F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$



(a) Wired-AND in open-collector TTL NAND gates.

(AND–OR–INVERT)

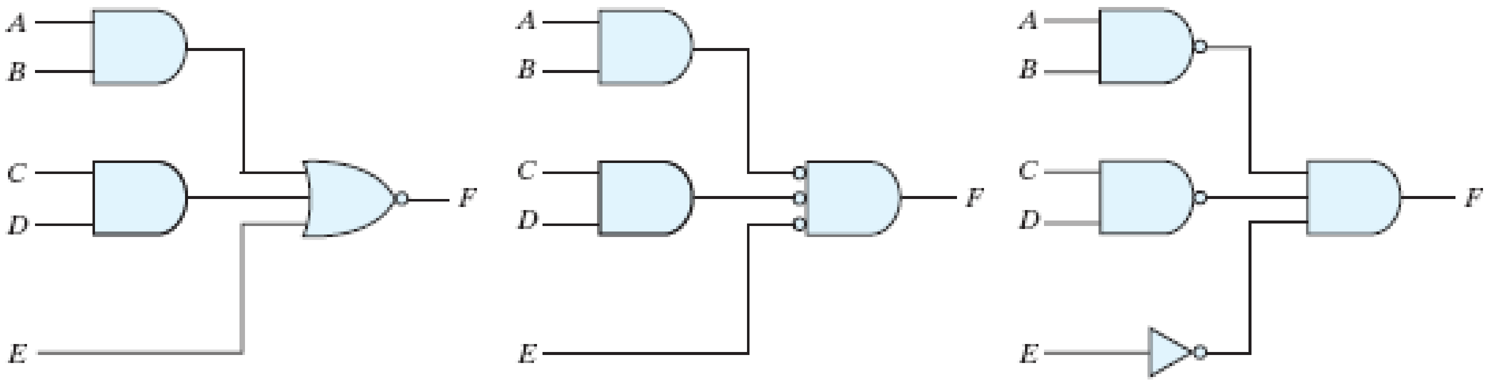


(b) Wired-OR in ECL gates

(OR–AND–INVERT)

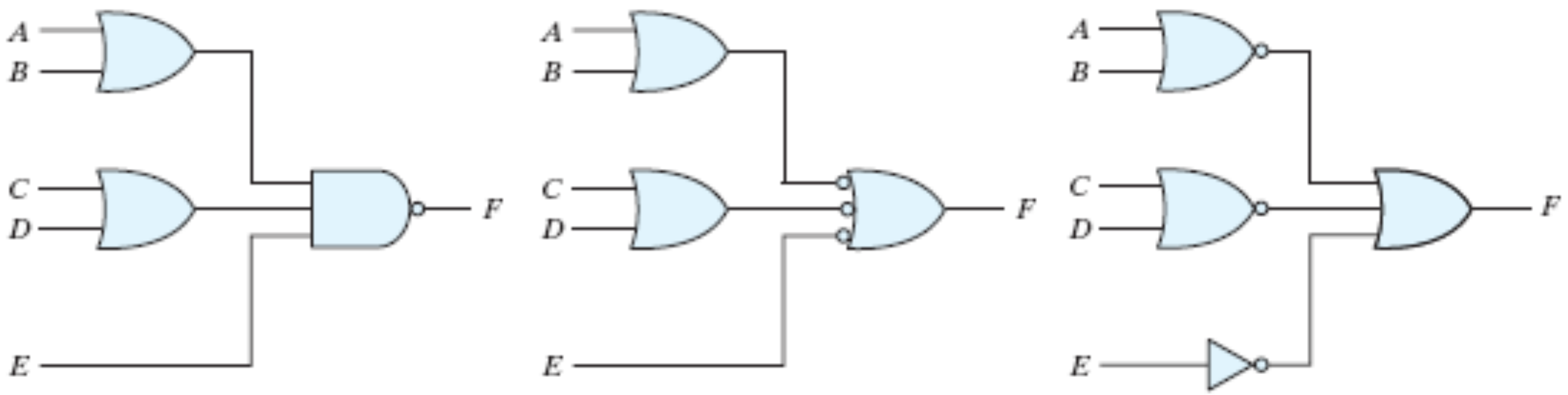
3.8 Other Two-Level Implementation (NOT COVERED)

- AND-OR-Invert Circuits
  - $F = (AB + CD + E)'$



(a) AND–NOR (b) AND–NOR (c) NAND–AND

- OR-AND-Invert Circuits
  - F = [(A + B)(C + D)E]'



(a) OR–NAND (b) OR–NAND (c) NOR–OR

3.8 Other Two-Level Implementation (NOT COVERED)

Table 3.3
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.

3.9 Exclusive-OR Function

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

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

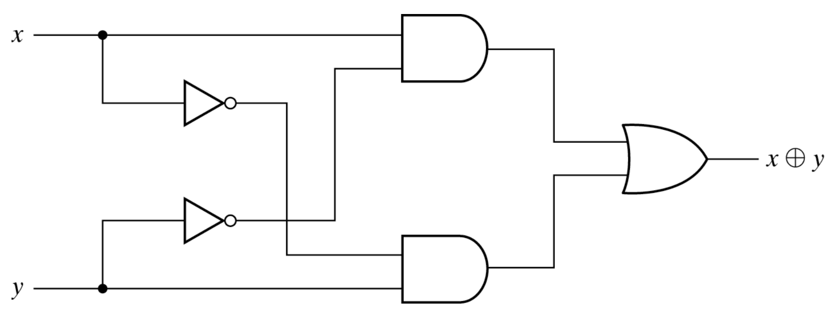
  $x \oplus 0 = x$ $\quad$ $x \oplus 1 = x'$

  $x \oplus x = 0$ $\quad$ $x \oplus x' = 1$

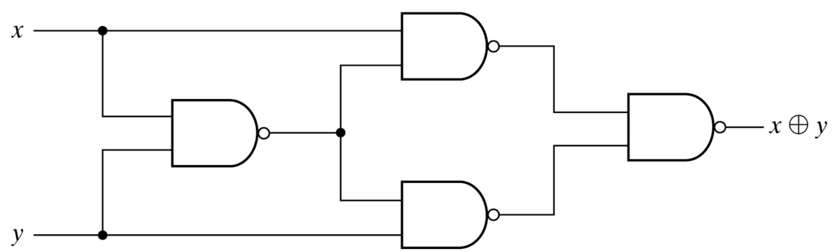
  $x \oplus y' = x' \oplus y = (x \oplus y)'$

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

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



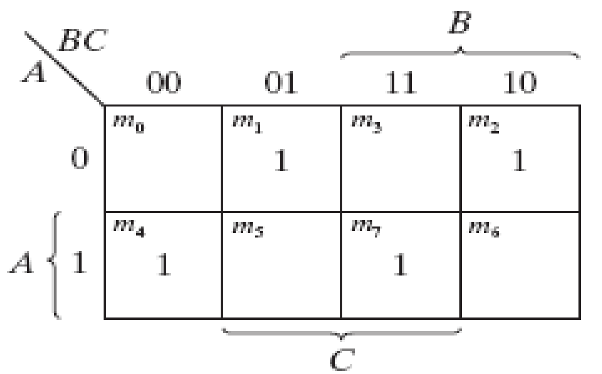
(a) With AND-OR-NOT gates



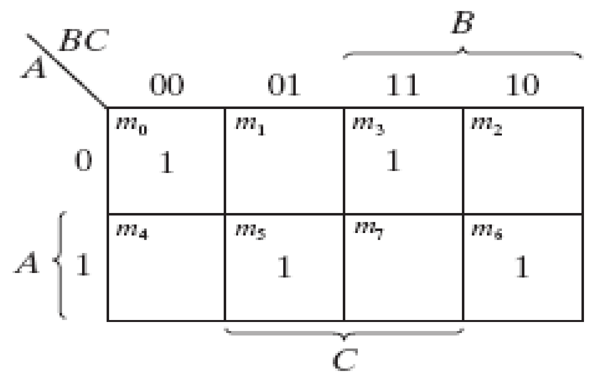
(b) With NAND gates

Fig. 3-32 Exclusive-OR Implementations

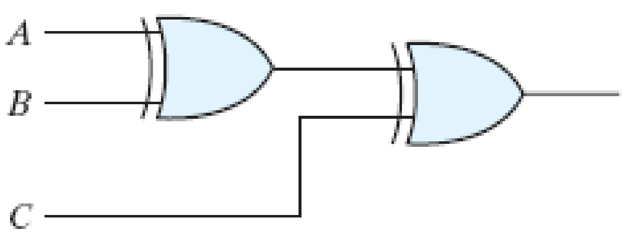
- Odd / Even Function



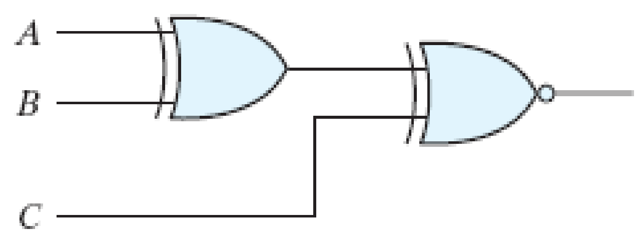
(a) Odd function $F = A \oplus B \oplus C$



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



(a) 3-input odd function



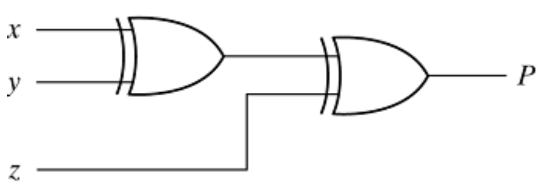
(b) 3-input even function

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

$$P = x \oplus y \oplus z$$

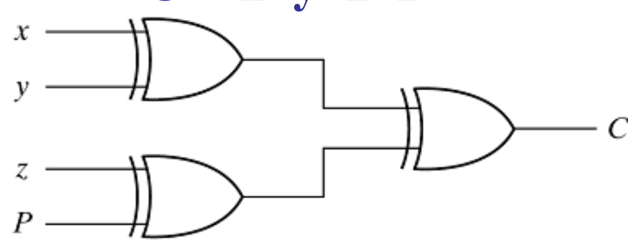


(a) 3-bit even parity generator

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

$$C = x \oplus y \oplus z \oplus P$$



(a) 4-bit even parity checker

Fig. 3-36 Logic Diagram of a Parity Generator and Checker