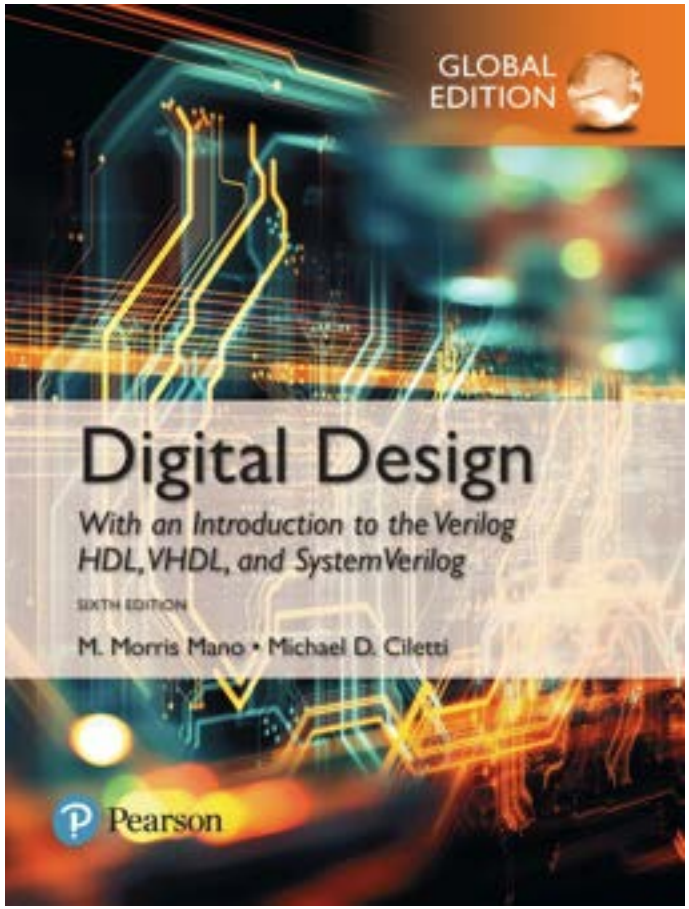
Digital Design (디지털 디자인)

With an Introduction to the Verilog HDL, VHDL, and SystemVerilog

5th or 6th Edition, Global Edition



Chapter 03

Gate-Level Minimization

게이트 레벨 최소화

전자공학과 김동한 교수

3.1 개요 & 3.2 카노 맵 방법

- 게이트 레벨 최소화: 부울 함수의 최적의 게이트 레벨 구현을 위한 설계 작업
- 카노 맵(Karnaugh map) 또는 K 맵: 부울 함수의 간략화를 위한 간단하고 직접적인 방법
  - 네모 칸들로 구성된 도면이며, 각 네모 칸은 하나의 최소항(minterm)을 나타냄
  - 간략화된 표현에서 맵은 곱의 합 또는 합의 곱이라는 표준 형식 중 하나로 표현됨
  - 항의 갯수가 최소이고 각 항이 최소의 리터럴 수를 갖도록 선택
  - 어떤 2개의 인접한 네모 칸은 1개의 변수만 다르게 표현됨

3.2 카노 맵 방법

- 2-변수 카노 맵

Figure 3.1
Two-variable K-map.

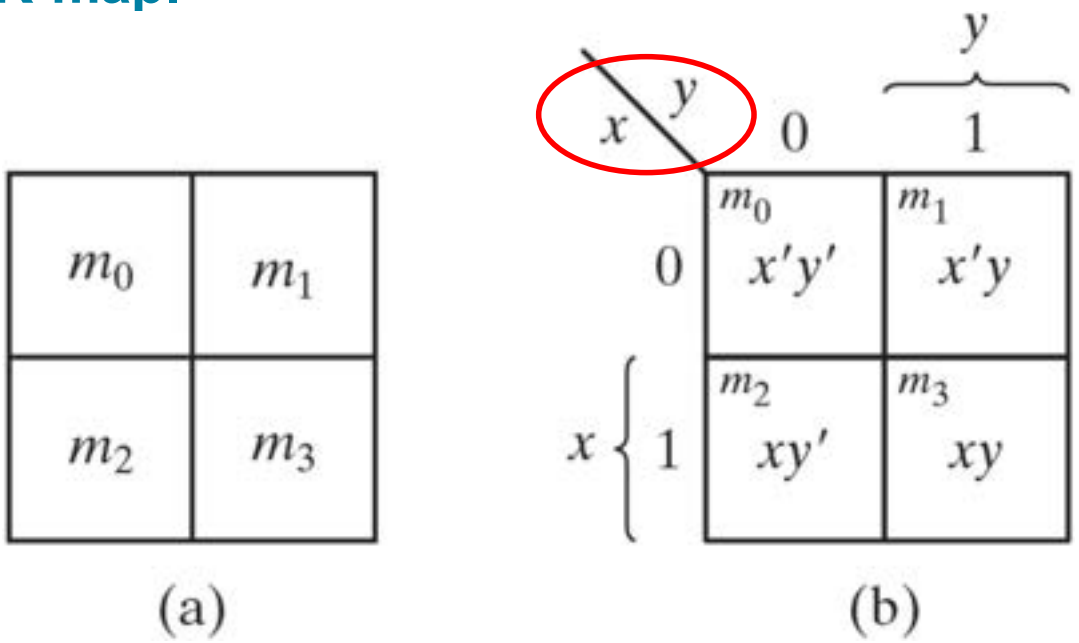
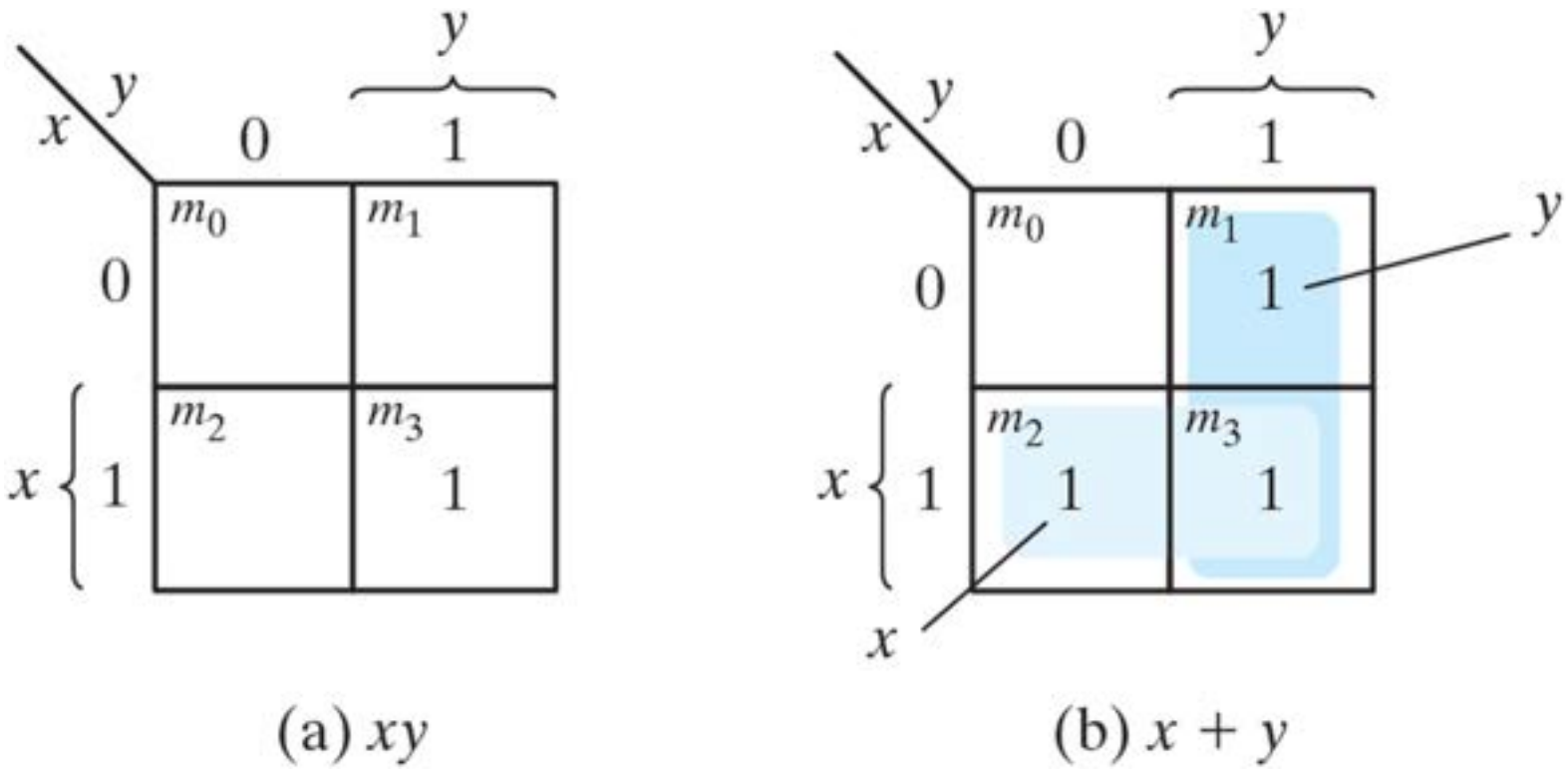


Figure 3.2
Representation of functions in the K-map.

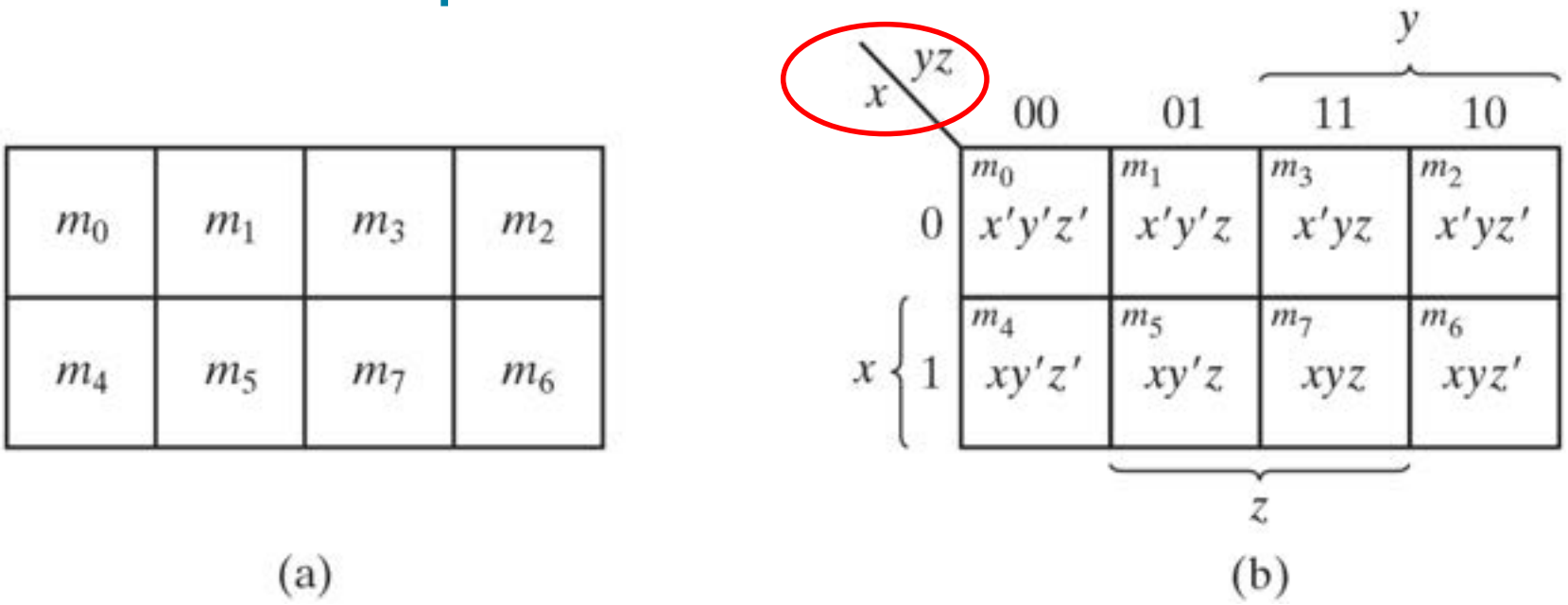


m1 + m2 + m3 = x'y + xy' + xy = x + y

3.2 카노 맵 방법

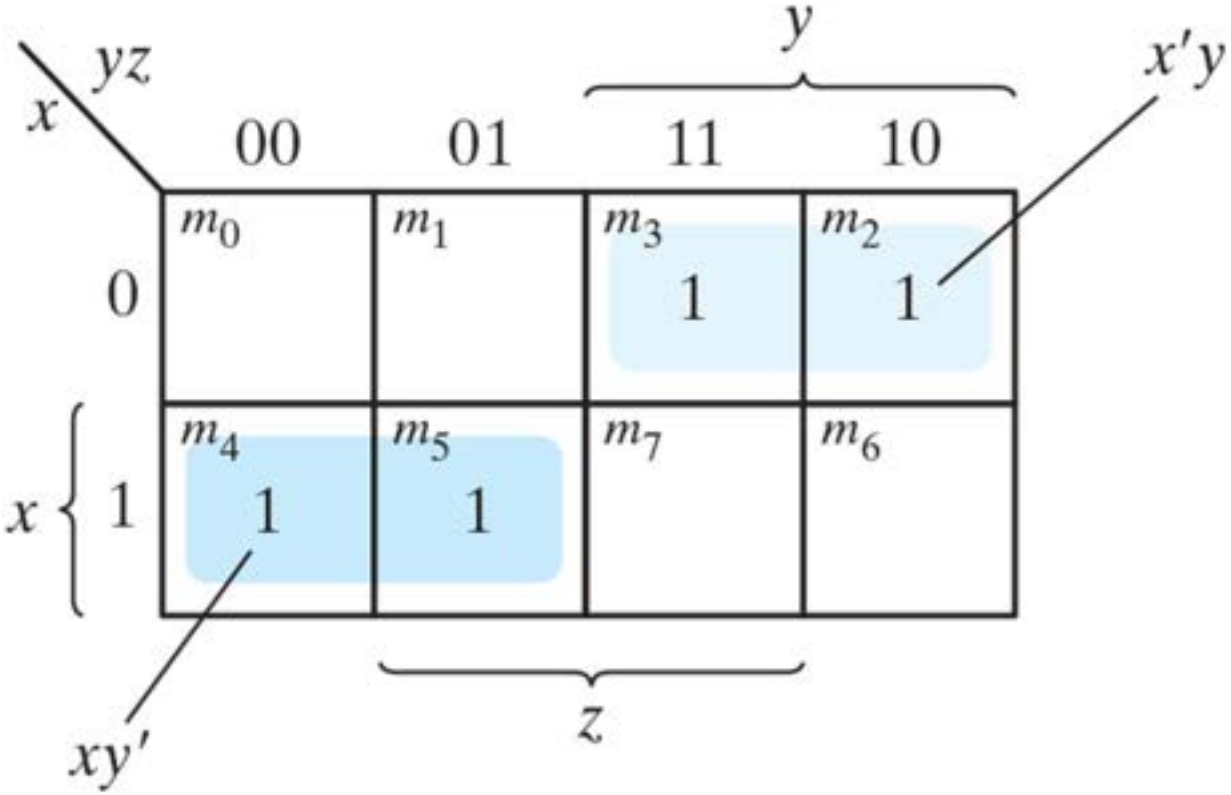
- 3-변수 카노 맵

Figure 3.3
Three-variable K-map.



- 예제 3.1) F(x, y, z) = $\Sigma(2, 3, 4, 5)$

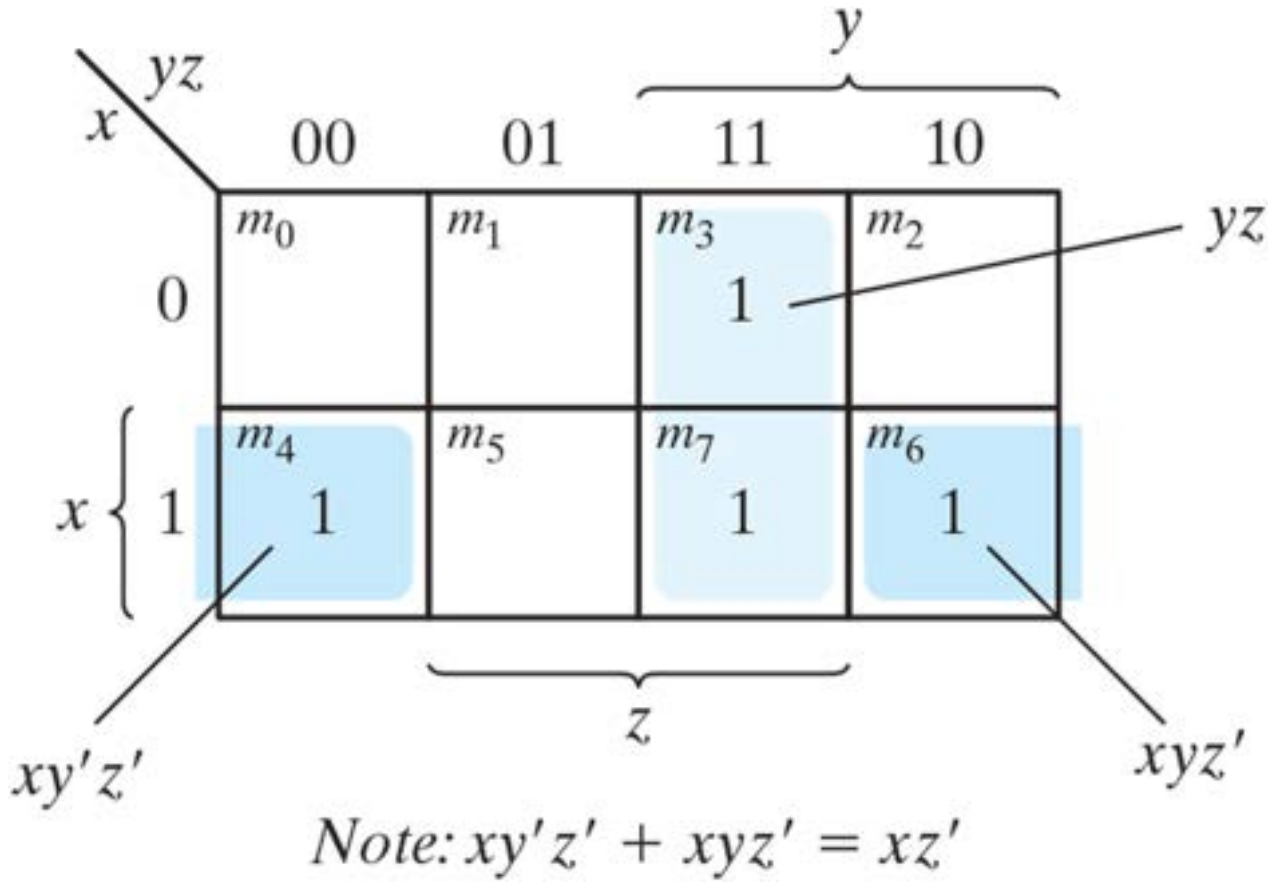
Figure 3.4
Map for Example 3.1, *F(x, y, z)* = Σ(2, 3, 4, 5) = *x'y + xy'*.



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

- 예제 3.2) F(x, y, z) = $\Sigma(3, 4, 6, 7)$

Figure 3.5
Map for Example 3.2, *F(x, y, z)* = Σ(3, 4, 6, 7) = *yz + xz'.*



3.2 카노 맵 방법

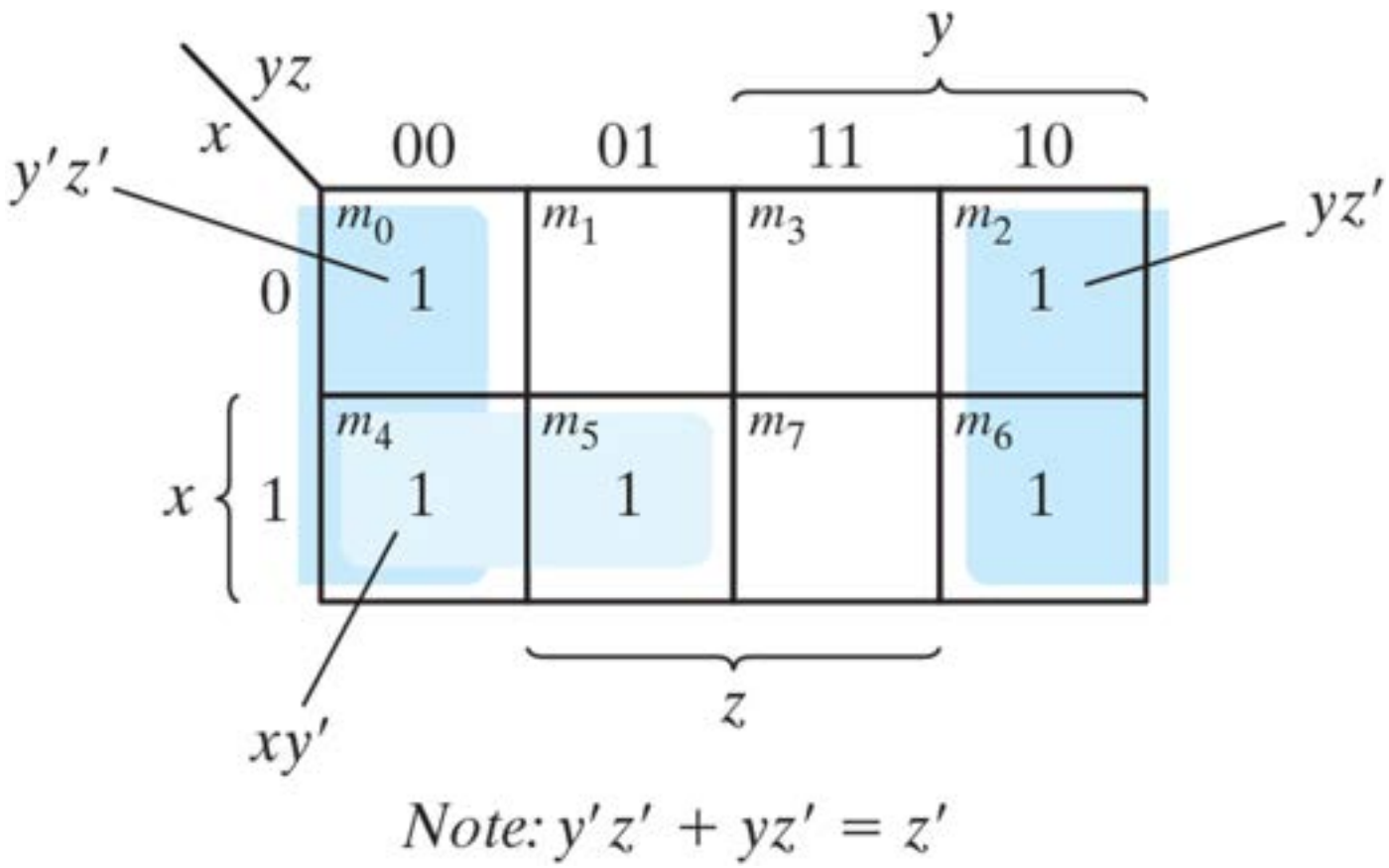
- $m_0 + m_2 + m_4 + m_6 = x'y'z' + x'yz' + xy'z' + xyz'$

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

  $= x'z' + xz' = z'(x' + x) = z'$
- 하나의 네모 칸은 3개의 리터럴 항
- 2개의 인접한 네모 칸들은 2개의 리터럴 항
- 4개의 인접한 네모 칸들은 1개의 리터럴 항
- 8개의 인접한 네모 칸들은 전체 맵을 둘러싸며 이는 1과 같음

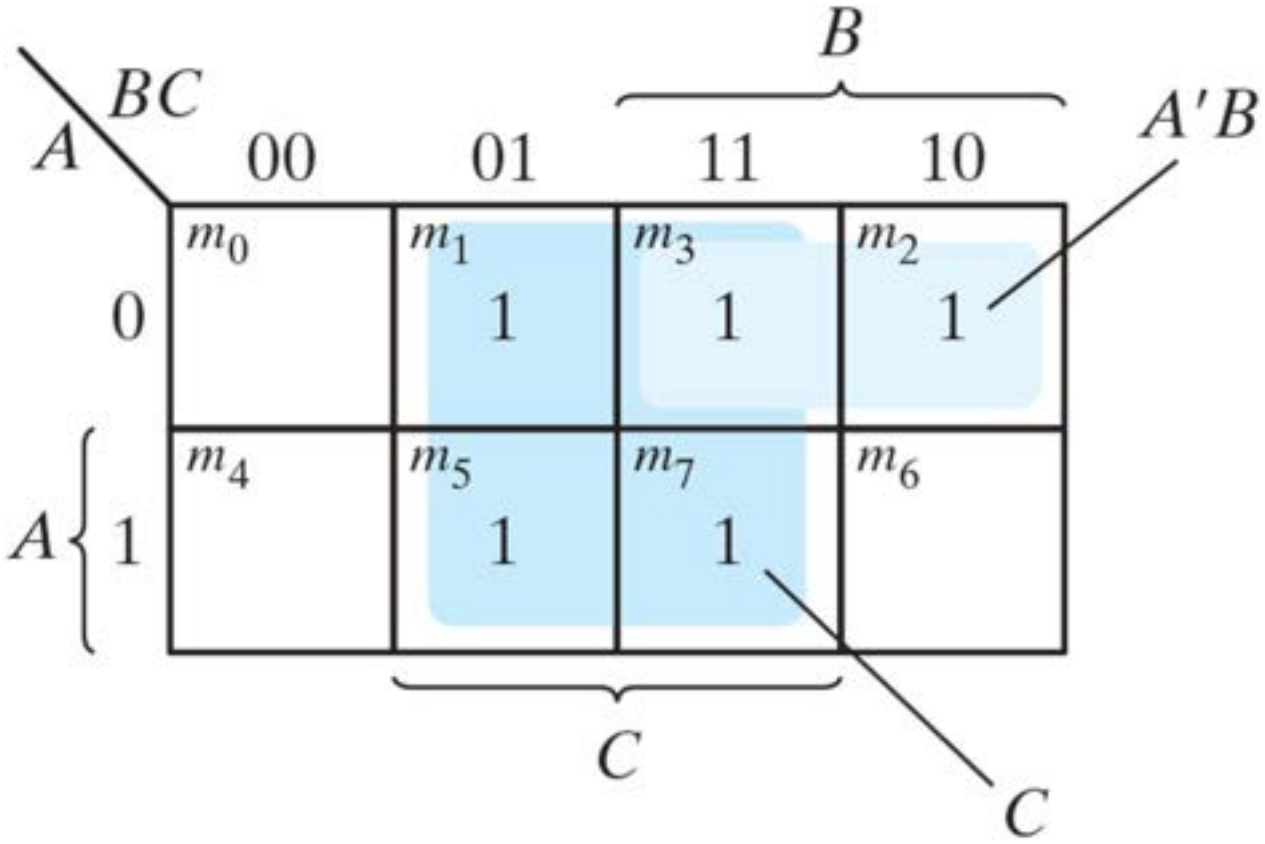
- 예제 3.3) F(x, y, z) = $\Sigma(0, 2, 4, 5, 6)$

Figure 3.6
Map for Example 3.3, *F(x, y, z)* = Σ(0, 2, 4, 5, 6) = *z'* + *xy'*.



- 예제 3.4) F = A'C + A'B + AB'C + BC
  1. 최소항들의 합? F(A, B, C) = Σ(1, 2, 3, 5, 7)
  2. 최소 곱의 합? F = C + A'B

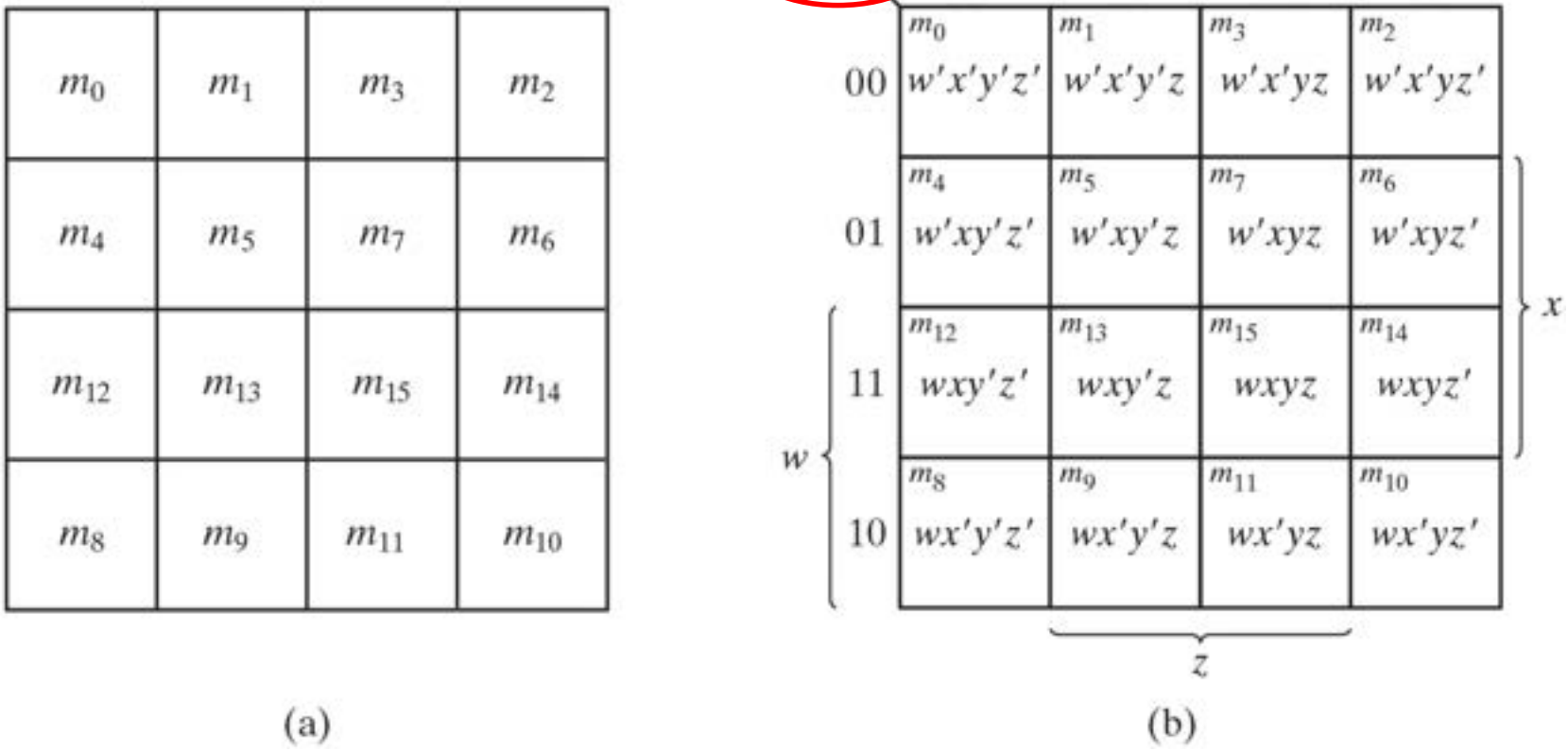
Figure 3.7
Map of Example 3.4, *A'C + A'B + AB'C + BC = C + A'B*.



3.3 4-변수 카노 맵

- 하나의 네모 칸은 4개의 리터럴 항
- 2개의 인접한 네모 칸들은 3개의 리터럴 항
- 4개의 인접한 네모 칸들은 2개의 리터럴 항
- 8개의 인접한 네모 칸들은 1개의 리터럴 항
- 16개의 인접한 네모 칸들은 전체 맵을 둘러싸며 이는 1과 같음

Figure 3.8
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)

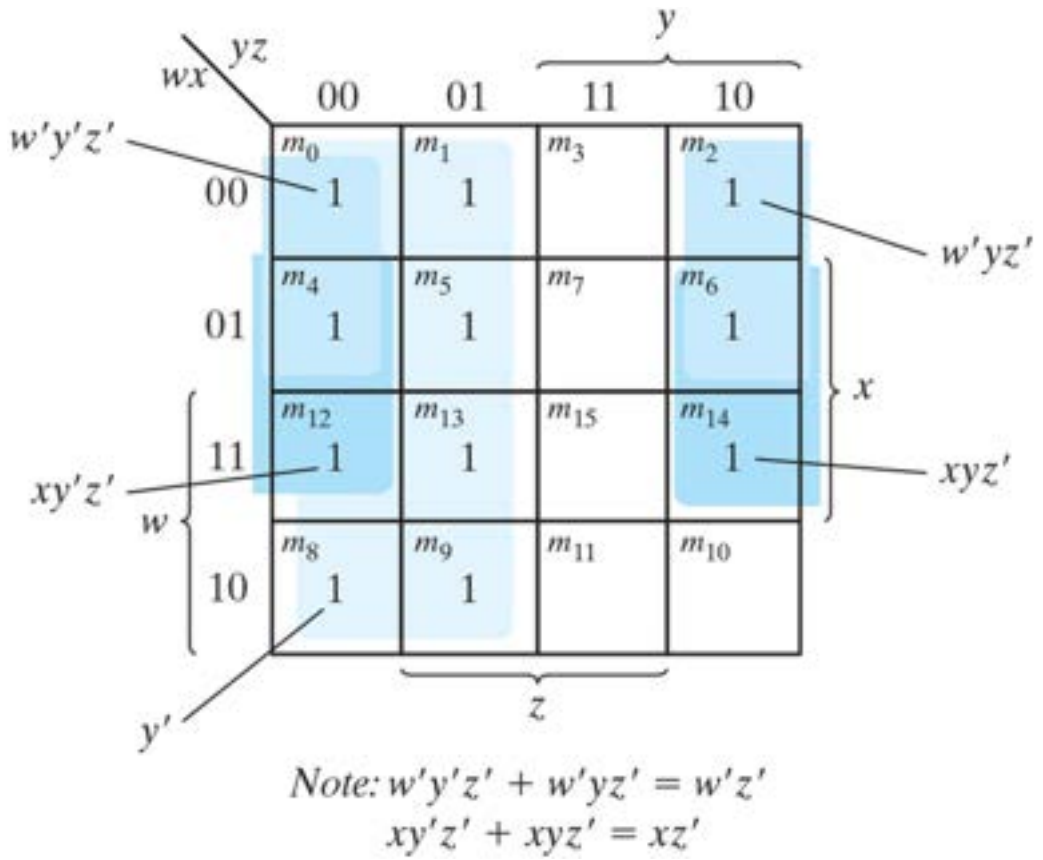
wx \ yz	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

y (11, 10) x (01, 11) w (11, 10) z (01, 11)

(b)

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

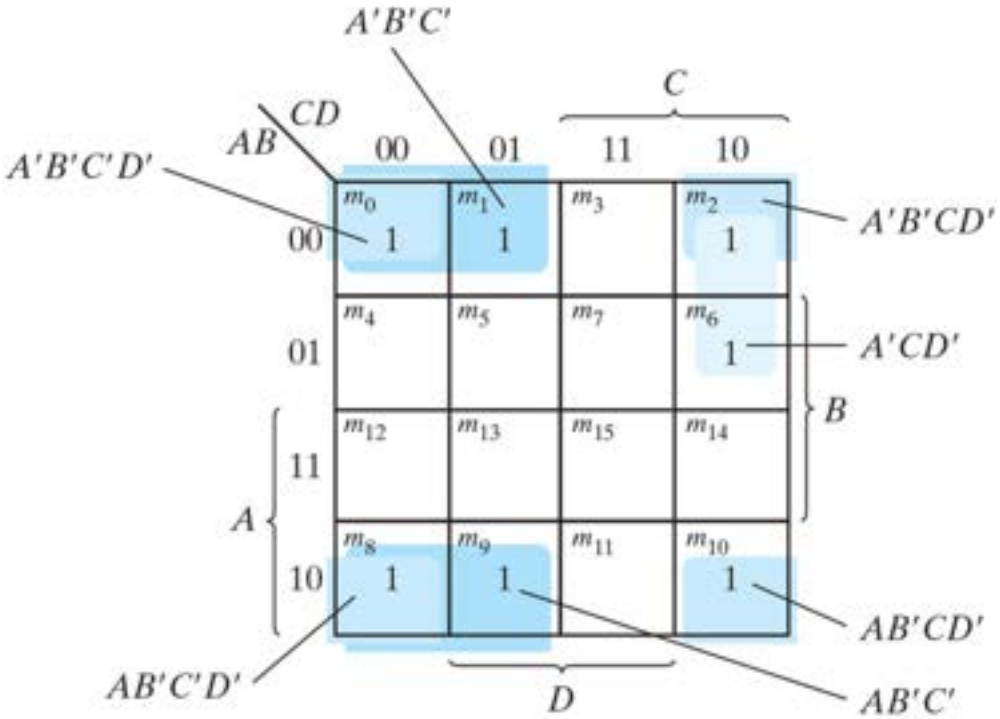
Figure 3.9
Map for Example 3.5, *F(w, x, y, z)* = Σ(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = *y'* + *w'z'* + *xz'*.



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

- 예제 3.6) F = A'B'C' + B'CD' + A'BCD' + AB'C'

Figure 3.10
Map for Example 3.6, *A'B'C' + B'CD' + A'BCD' + AB'C = B'D' + B'C' + A'CD'.*



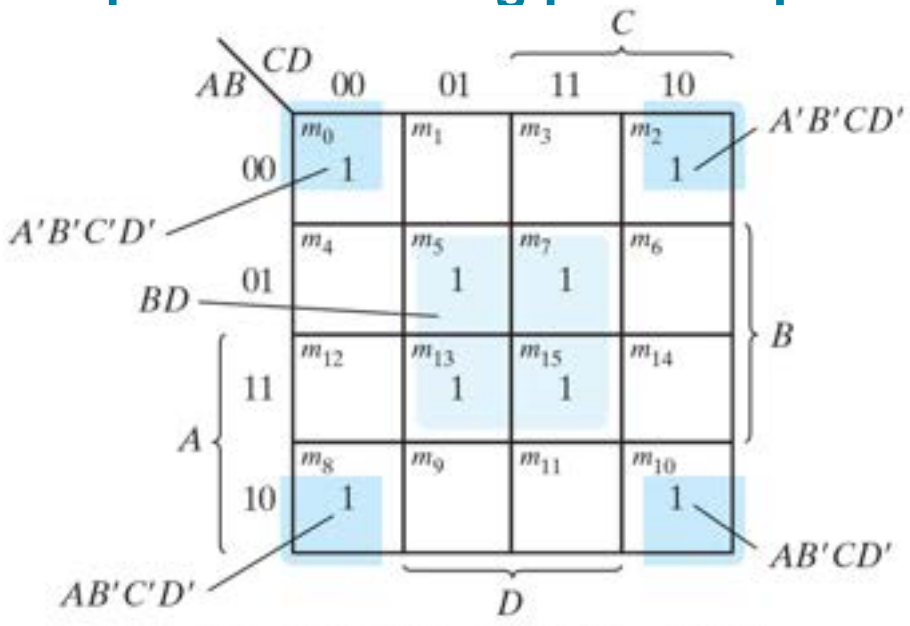
Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

3.3 4-변수 카노 맵

- 주항(prime implicant): 맵 안의 인접한 네모 칸을 최대로 결합했을 때 얻어지는 곱의 항
  - 어떤 함수의 주항들은 네모 칸들을 최대로 결함해서 얻음
- 필수적(essential): 어떤 최소항을 단 하나의 주항으로만 커버할 수 있을때
- 가장 간략화된 표현식을 얻기 위해서는 먼저 모든 필수 주항을 찾아야함
  - 모든 필수 주항과 그 필수 주항들이 커버하지 못한 나머지 최소항 표현에 필요한 추가 주항을 합하면 됨

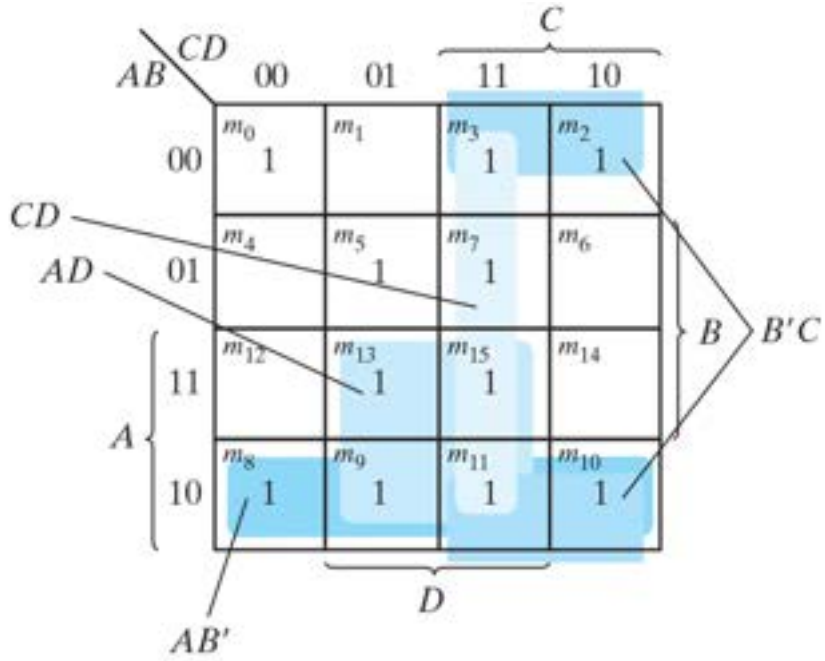
- 예) $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

Figure 3.11
Simplification using prime implicants.



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 $B'D'$ (m_0과 m_5를 포함하는 주항을 만드는 유일한 방법)



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

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

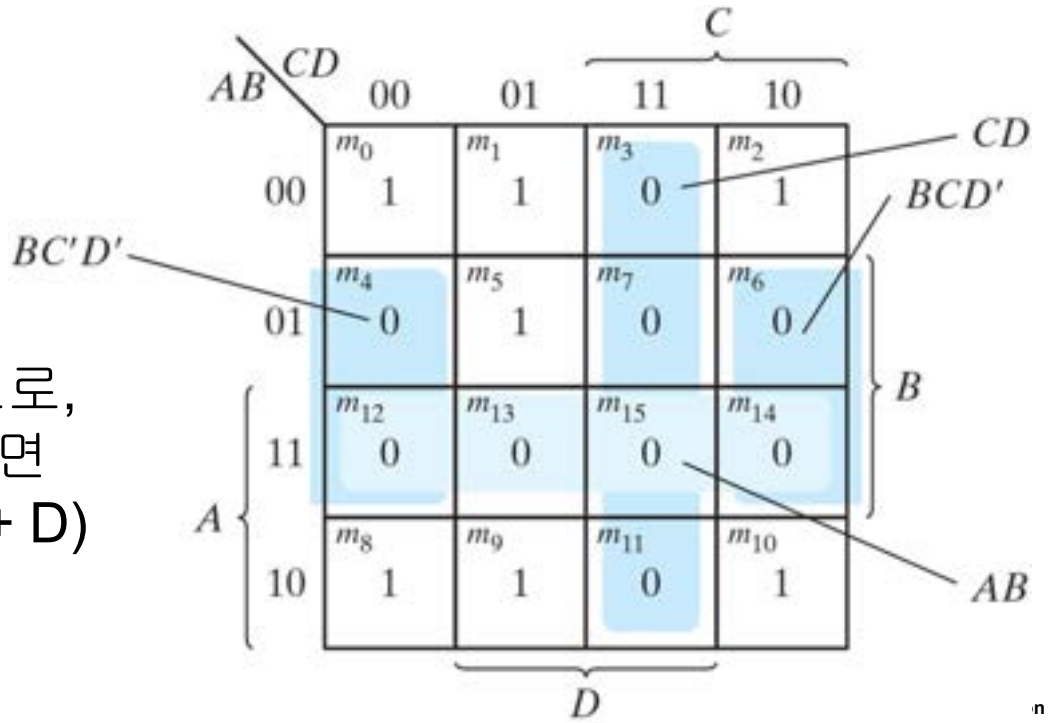
3.4 합의 곱 표현식의 간략화

- 카노맵은 곱의 합을 간략화할 때 사용하므로 합의 곱은 카노맵에서 0으로 표시된 칸을 결합해서 간략화한 후, 보수를 취하면 구해짐

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

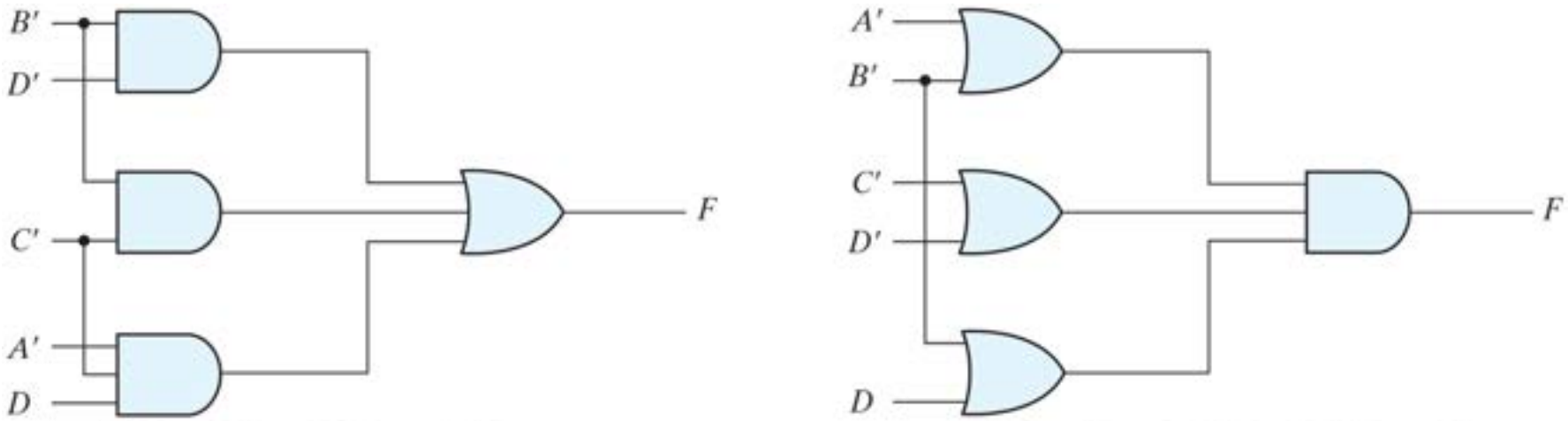
(a) 곱의 합 형식:
F = B'D' + B'C' + A'C'D

(b) 합의 곱 형식:
F' = AB + CD + BD' 이므로,
드모르간 정리를 적용하면
F = (A' + B')(C' + D')(B' + D)



- 곱의 합 형식은 AND 게이트가 OR 게이트에 연결
- 합의 곱 형식은 OR 게이트가 AND 게이트에 연결
- 표준형식(곱의 합, 합의 곱)은 2-레벨 구현

Figure 3.13
Gate implementations of the function of Example 3.7.



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

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

- 예) 최소항의 합으로 표현하면, F(x, y, z) = $\Sigma(1, 3, 4, 6)$
  최대항의 곱으로 표현하면, F(x, y, z) = $\Pi(0, 2, 5, 7)$

F = x'z + xz'

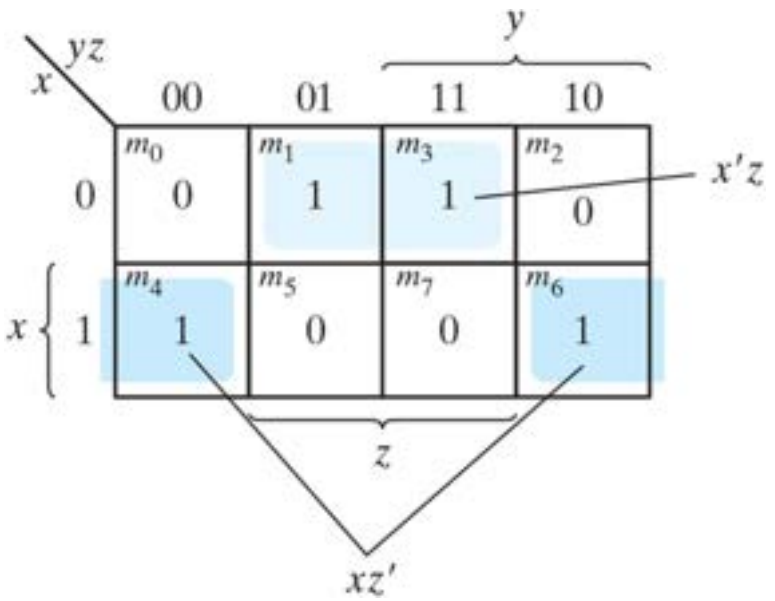
F' = xz +x'z' → 보수 취하면 → F = (x' + z')(x + z)

: 반대로도 가능하도록 연습 필요

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

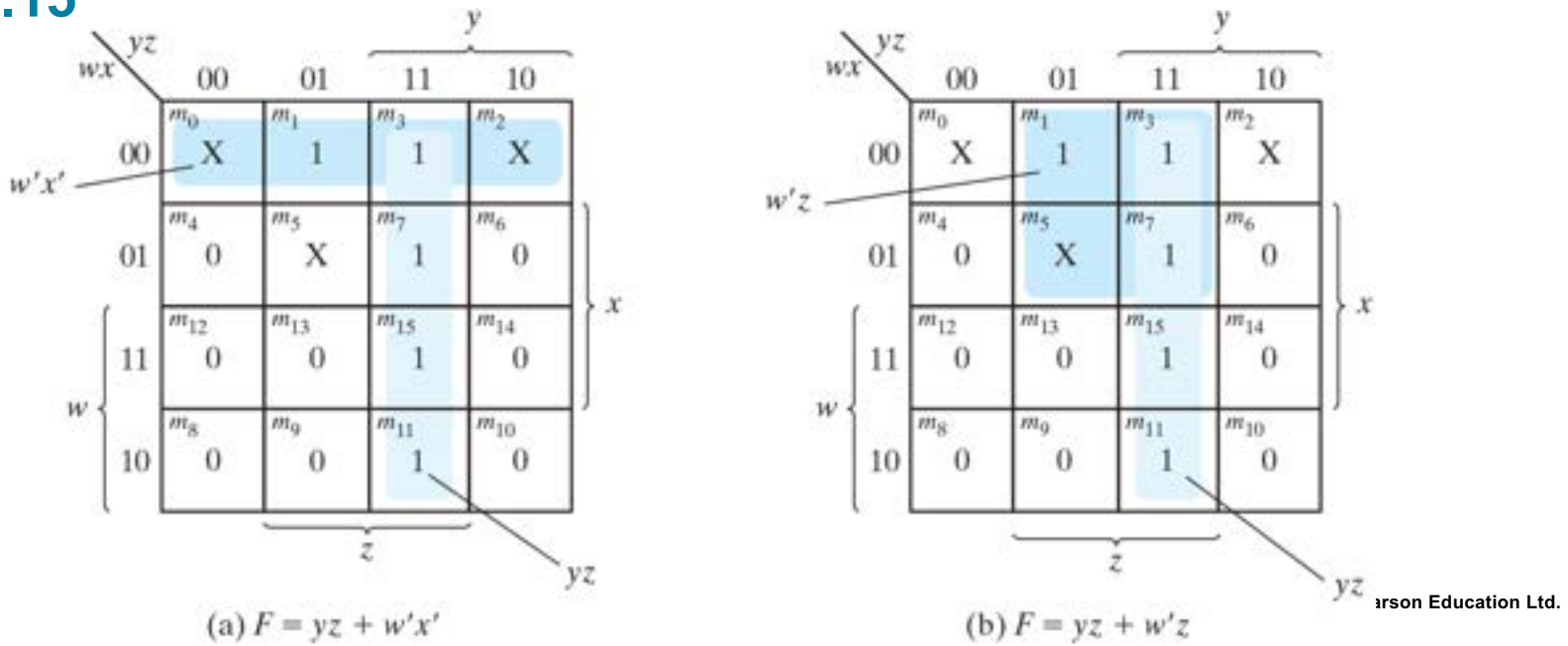
Figure 3.14
Map for the function of Table 3.1.



3.5 don’t care 조건

- 입력 조합에 대해 규정되지 않은 출력을 갖는 함수들은 불완전하게 규정된 함수(incompletely specified function)라고 함. 대부분의 경우에, 0이든 1이든 상관하지 않음. 즉, don’t care 혹은 don’t care 조건이라고 함 (카노맵에서 X로 표시하고 간략화에 활용)
- 예제 3.8) F(w, x, y, z) = Σ(1, 3, 7, 11, 15) 이고,
  d(w, x, y, z) = Σ(0, 2, 5)

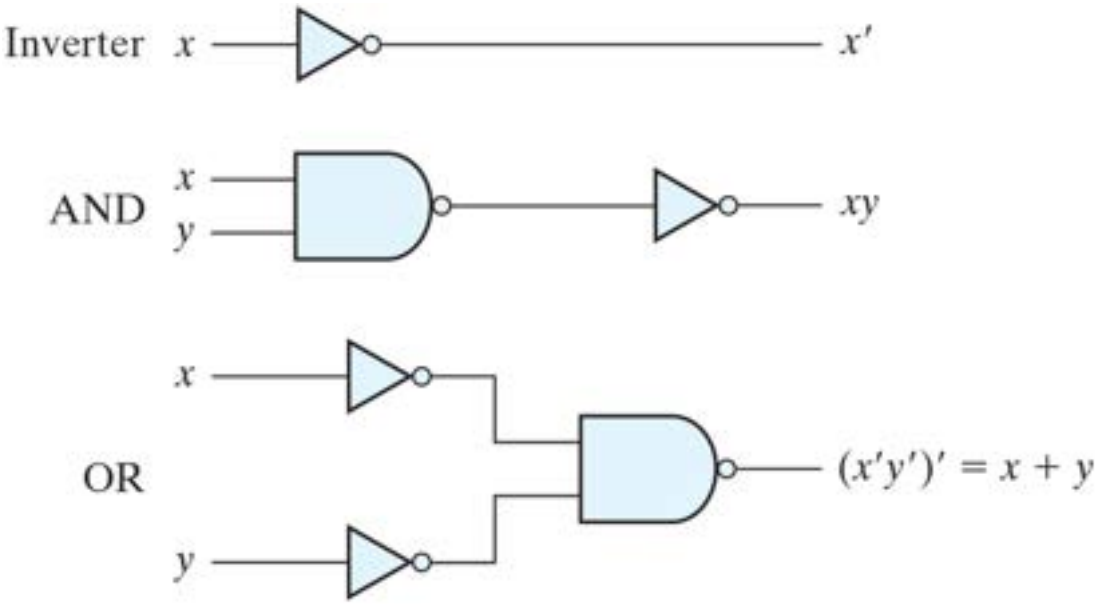
Figure 3.15



3.6 NAND와 NOR의 구현

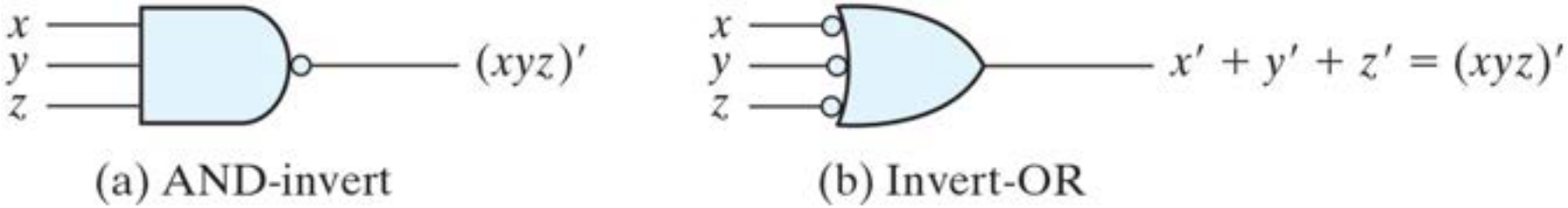
- 디지털 회로에서 AND 및 OR 게이트보다는 NAND와 NOR 게이트로 구성함. 제작이 더 용이하기 때문임. 따라서 AND, OR, NOT의 항으로 주어진 부울 함수를 등가의 NAND및 NOR 논리도로 바꿀 수 있어야 함.

- NAND 회로
  - NAND 게이트는 어떤 디지털 시스템도 구현할 수 있기 때문에 보편적인 게이트(universal gate)라고 함



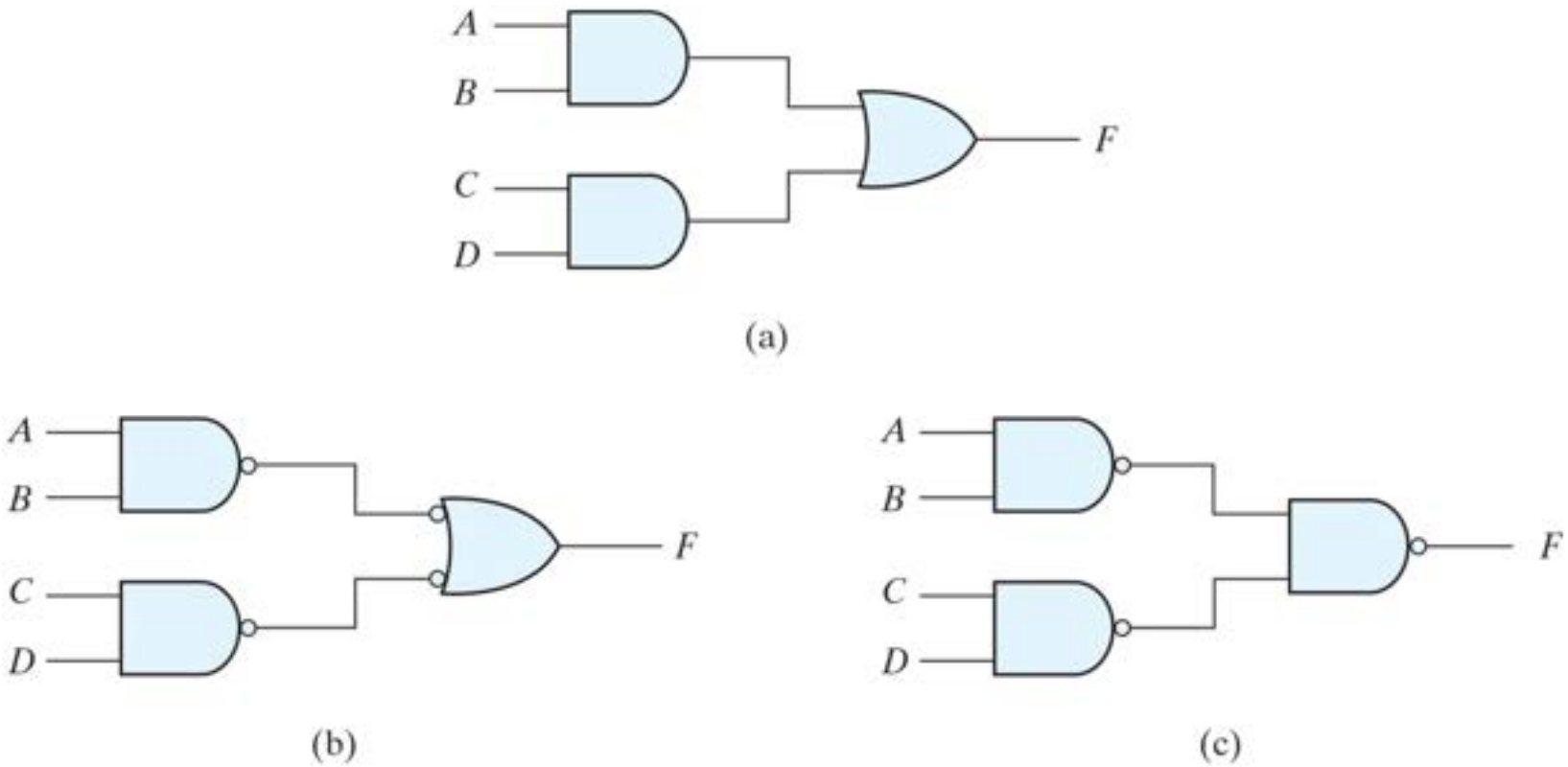
- NAND 게이트를 표현하는 2가지 기호는 아래와 같음

Figure 3.17
Two graphic symbols for a three-input NAND gate.



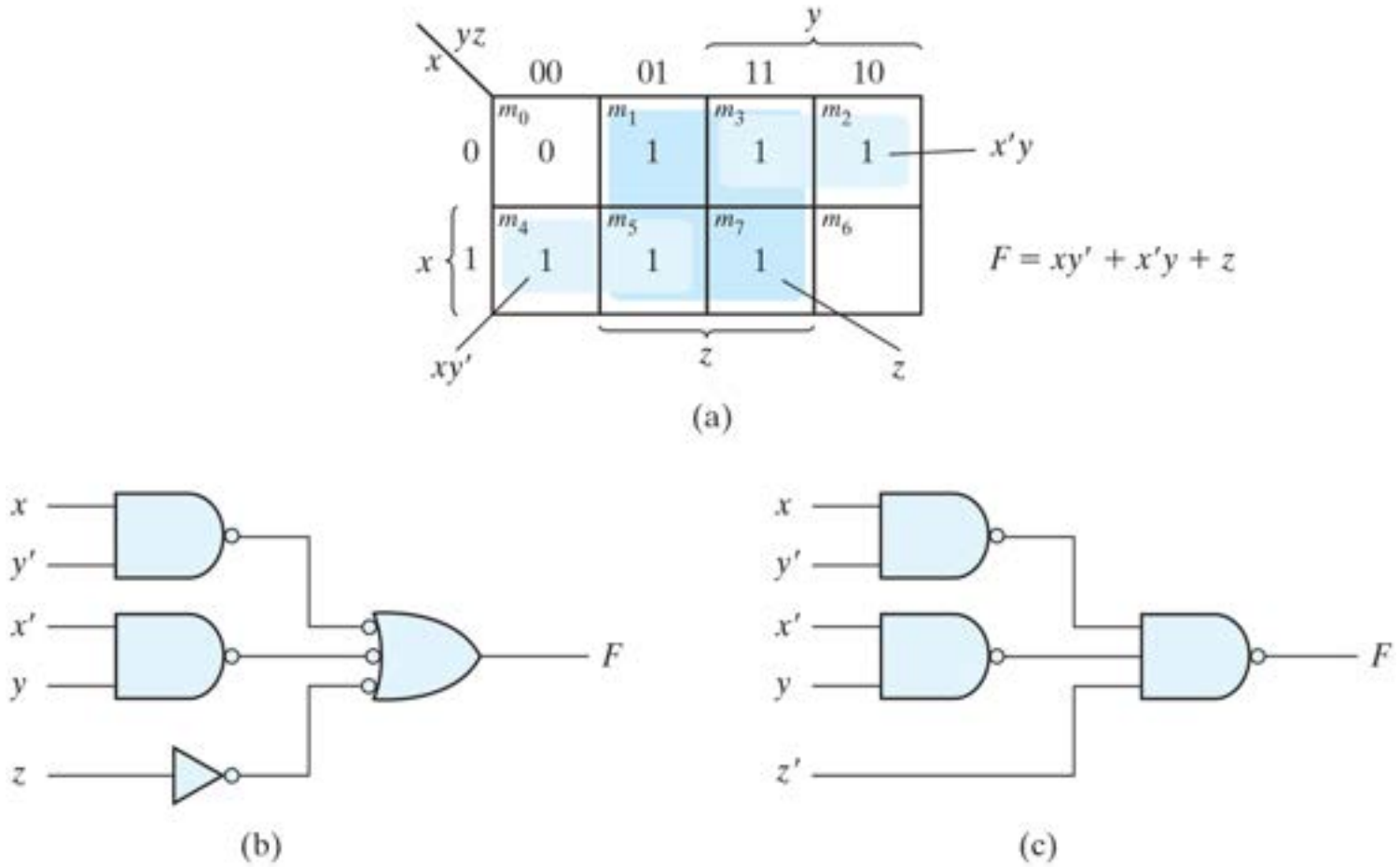
- NAND 게이트만으로 부울 함수를 구현하는 방법: 부울 연산자 형식으로 간략화된 부울 함수로 나타낸 후, NAND 논리로 변환함.
- 2-레벨 구현: 곱의 합 형식으로 간략화한 후 AND, OR 게이트로 표현. AND 게이트를 NAND 게이트로, OR 게이트를 invert-OR 형식의 NAND로 바꿈. 마지막으로 Invert-OR 형식을 AND-invert 형식으로 바꿈,

Figure 3.18
Three ways to implement *F* = *AB* + *CD*.



- 예제 3.9) F(x, y, z) = Σ(1, 2, 3, 4, 5, 7)

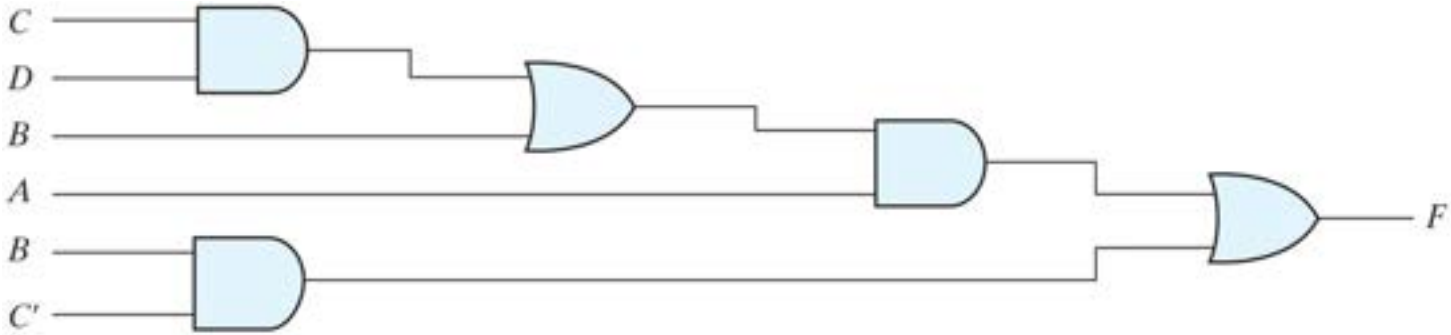
Figure 3.19
Solution to Example 3.9.



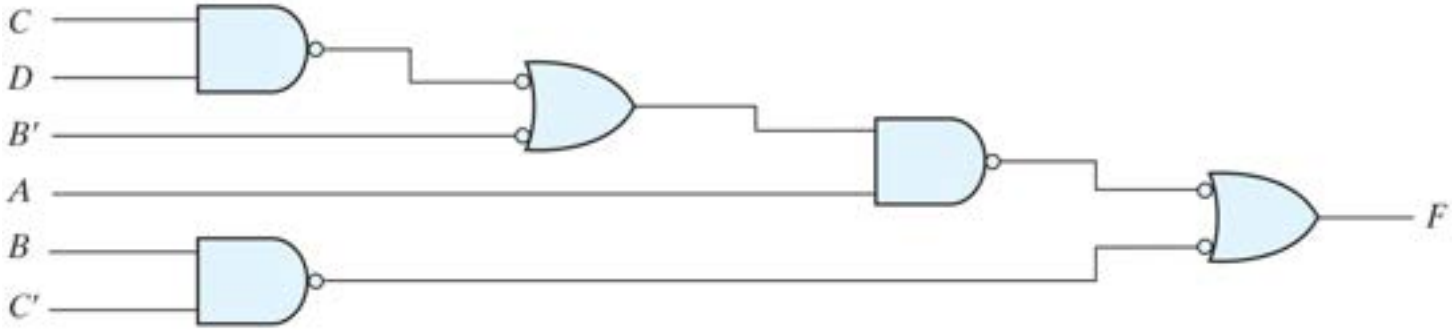
- 부울 함수로부터 논리도를 얻는 과정 정리
  1. 함수를 곱의 합 형식으로 표현
  2. (1-레벨) 곱의 항에 NAND 게이트를 그림. 단, 1개의 입력(리터럴)만 있으면 그냥 인버터를 사용
  3. 1-레벨 게이트의 출력을 입력으로 하여 2-레벨 당일 NAND 게이트(AND-invert 또는 invert-OR)를 그림

- 다중레벨 NAND 회로
  - 부울 함수의 표준 형식의 표현은 2-레벨 회로로 구현. 그 이상 레벨의 게이트 구조도 먼저 AND, OR 게이트로 구현하고 나서 NAND로 변환

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



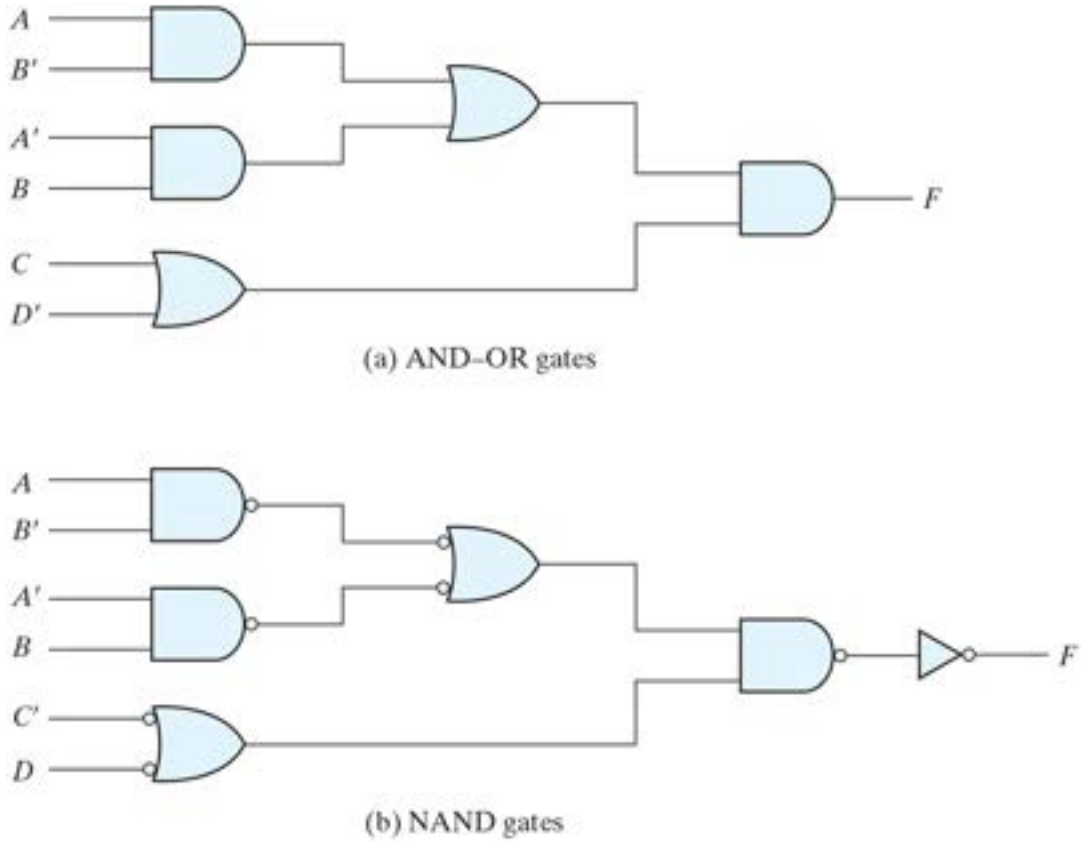
(a) AND–OR gates



(b) NAND gates

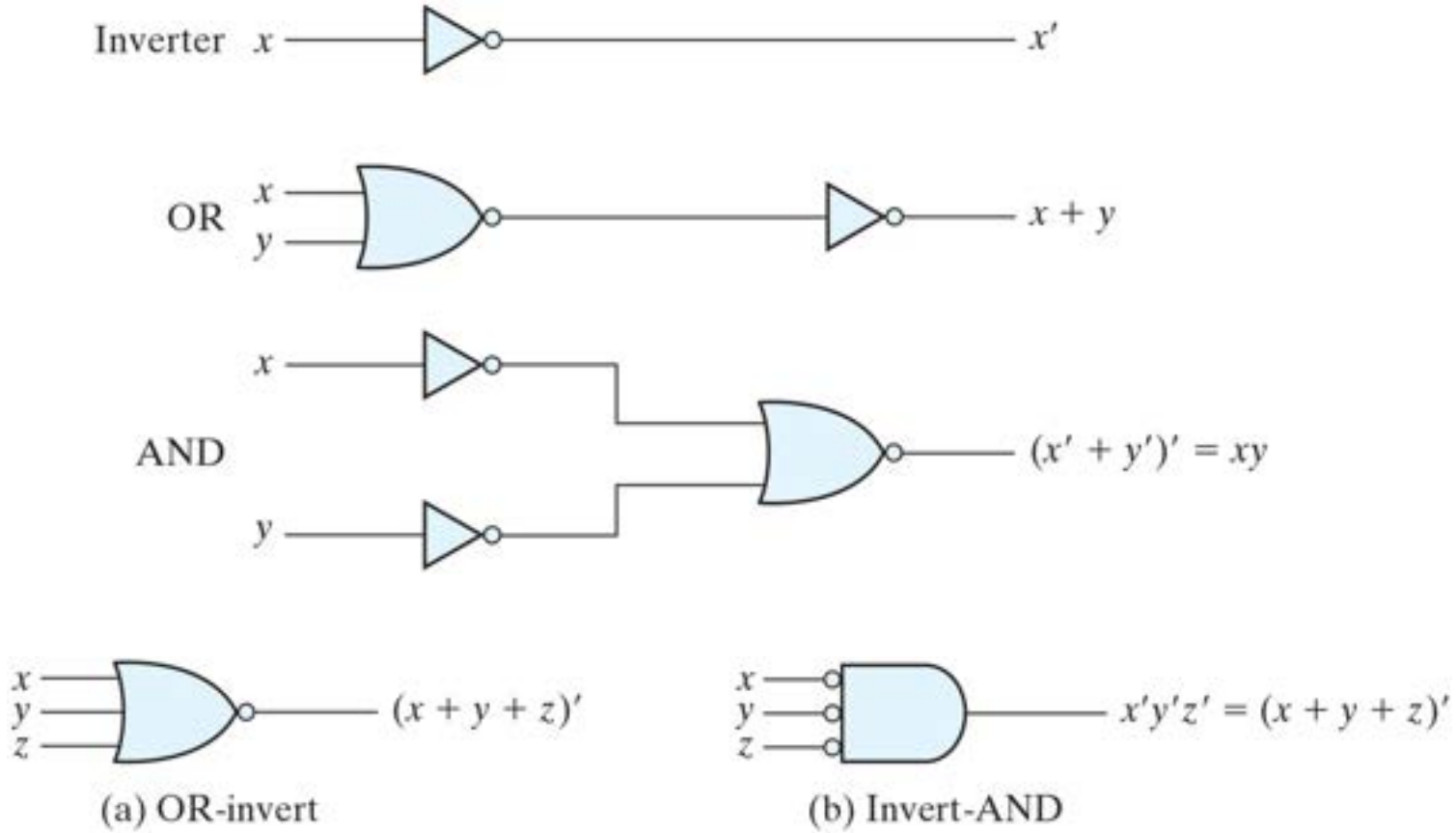
- 혼합 표현 방식을 사용하여 다중레벨 AND-OR 다이어그램을 NAND 다이어그램으로 변환하는 절차
  1. AND 게이트를 AND-invert 형식의 NAND 게이트로 변환
  2. OR 게이트를 invert-OR 형식의 NAND 게이트로 변환
  3. 다이어그램의 버블을 검사해서 같은 선상에서 다른 버블로 상쇄되지 않은 버블은 인버터 또는 입력 리터럴을 반전시킴

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



- NOR 구현
  - NAND 와 쌍대임

Figure 3.22
Logic operations with NOR gates.



- 합의 곱으로 표현된 함수를 NOR 게이트로 구현
  - 1-레벨의 OR 게이트를 OR-invert 형식의 NOR 게이트로
  - 2-레벨의 AND 게이트를 invert-AND 형식의 NOR 게이트로

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

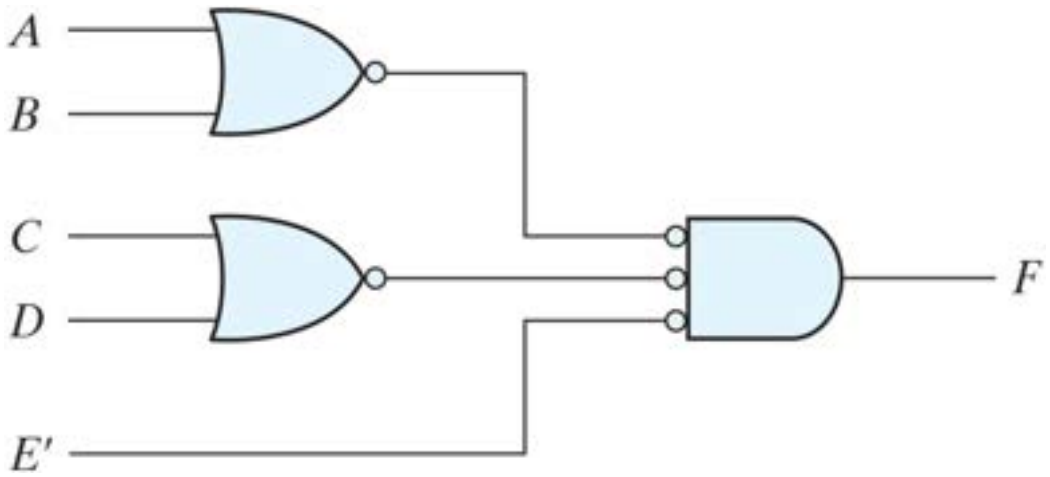
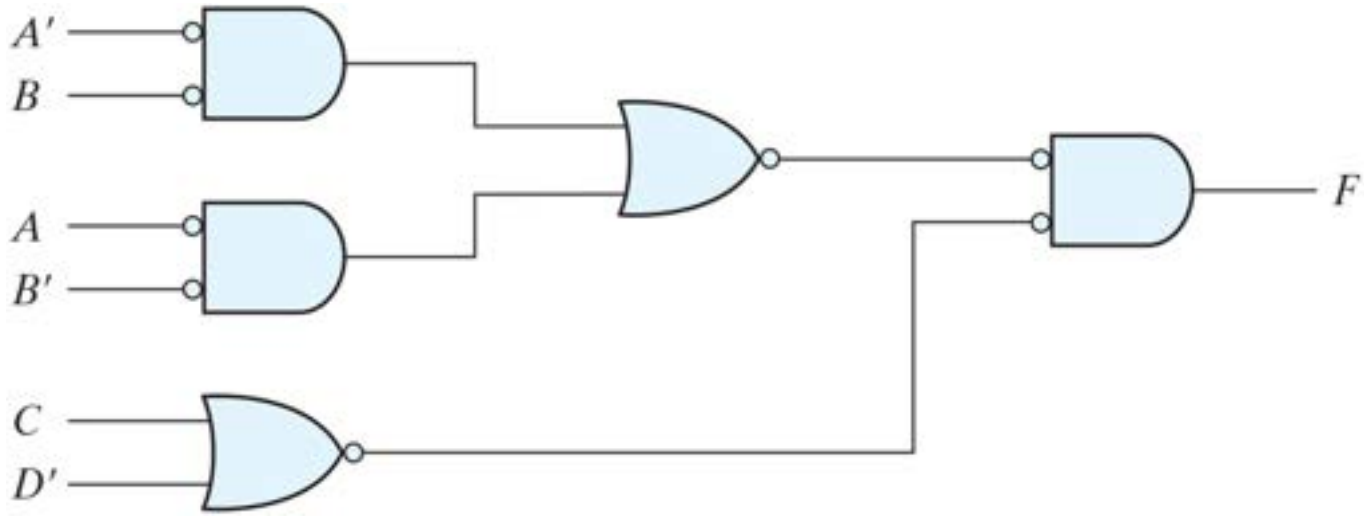


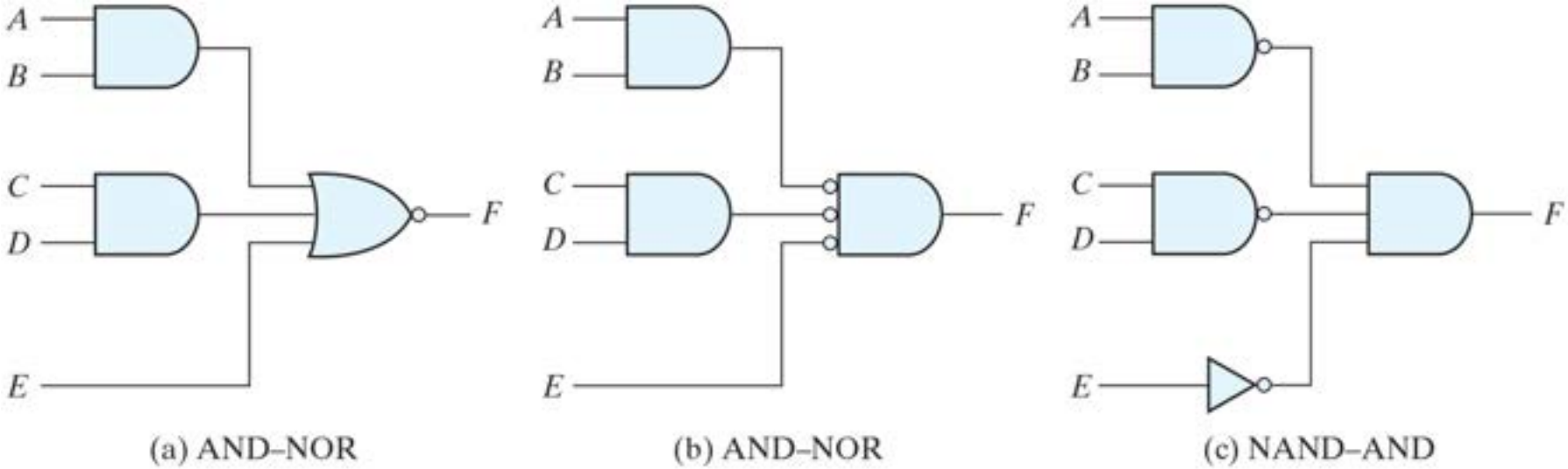
Figure 3.25
Implementing *F = (AB' + A'B)(C + D')* with NOR gates.



3.7 기타 2-레벨 구현

- AND, OR, NAND, NOR 게이트로 만들 수 있는 2-레벨 조합은 16개임. 이 중 8개는 단일 동작(단일 레벨, 단일 게이트)으로 표현됨. 이를 축퇴(degenerate) 형식이라 함.
  - AND-AND, OR-OR, AND-NAND, OR-NOR, NAND-OR, NOR-AND, NOR-NAND, NAND-NOR
- 나머지 8개의 조합은 단일 레벨 혹은 단일 게이트로 줄일 수 없음. 이를 비축퇴(nondegenerate) 형식이라고 함.
  - AND-OR, OR-AND, NAND-NAND, NOR-NOR(앞 절에서 이미 설명함), NOR-OR, OR-NAND, AND-NOR, NAND-AND
- AND-OR-INVERT 구현
  - 카노맵에서 보수(complement, 0)를 결합하여 곱의 합 형식으로 나타내서 F'를 구하고 출력반전를 통과시켜 구현

Figure 3.27
AND–OR–INVERT circuits, *F = (AB + CD + E)'.*



3.7 기타 2-레벨 구현

- 나머지 8개의 조합은 단일 레벨 혹은 단일 게이트로 줄일 수 없음. 이를 비축퇴(nondegenerate) 형식이라고 함.
  - AND-OR, OR-AND, NAND-NAND, NOR-NOR(앞 절에서 이미 설명함), NOR-OR, OR-NAND, AND-NOR, NAND-AND
- OR-AND-INVERT 구현
  - 카노맵에서 1을 결합하고 보수화하여 합의 곱 형식으로 나타내서 구현

Figure 3.28
OR–AND–INVERT circuits, *F = [(A + B)(C + D)E]'.*

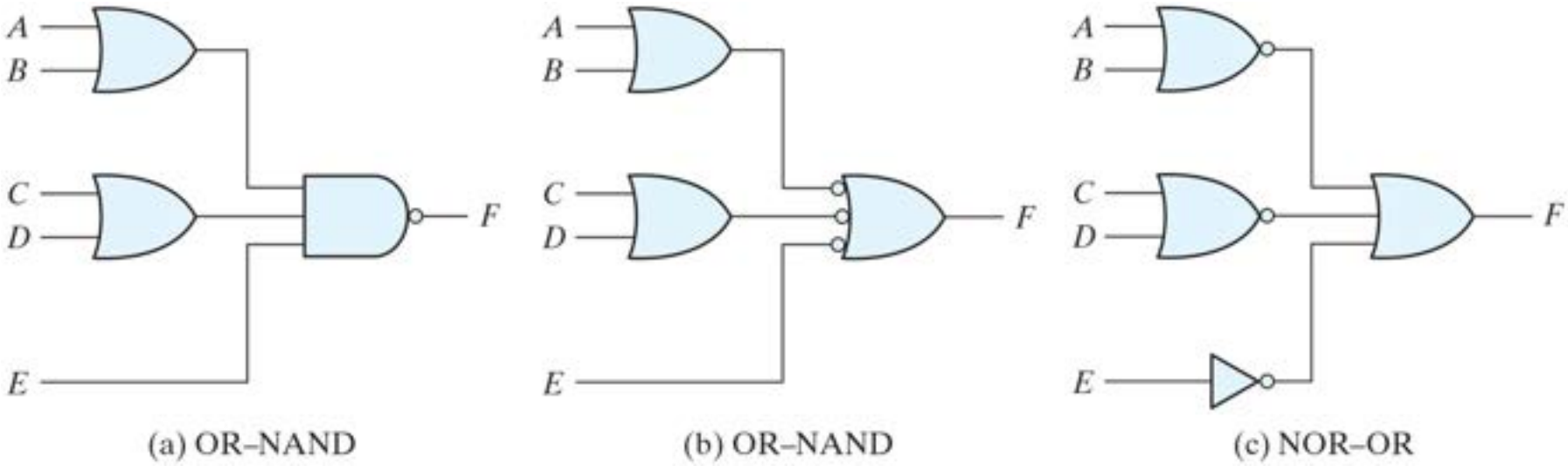


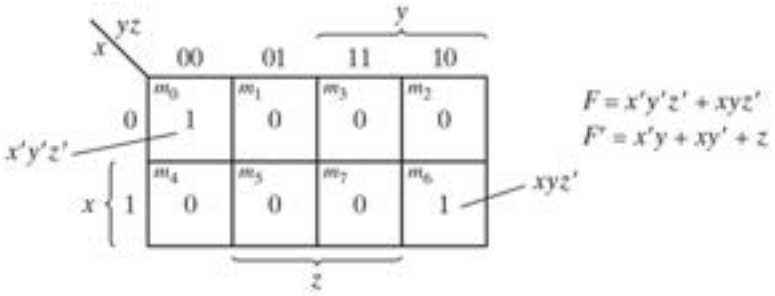
Table 3.2
Implementation with Other Two-Level Forms.

Equivalent Nondegenerate Implementation		Implements the Form	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

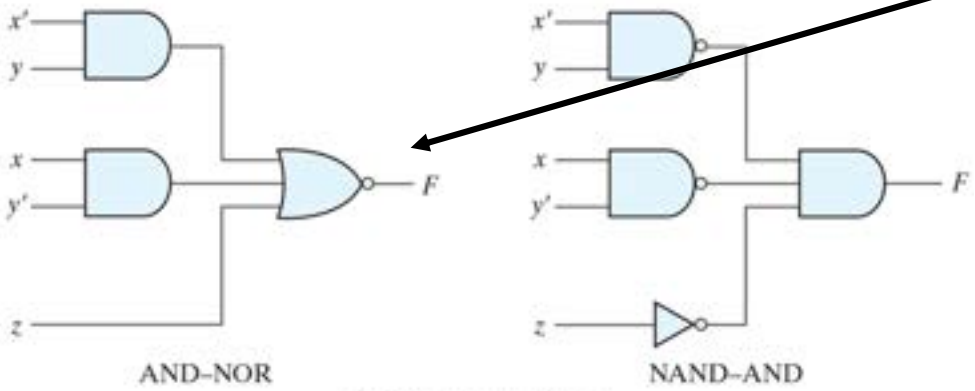
*Form (b) requires an inverter for a single literal term.

- 예제 3.10)

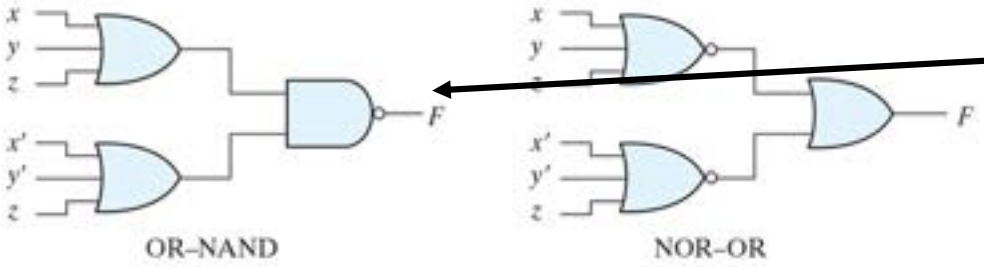
Figure 3.29
Other two-level implementations.



(a) Map simplification in sum of products



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



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

- 카노맵에서 보수(0)을 결합해서 곱의 합 형식으로 간략화하면,
- F' = x'y + xy' + z 이고

F = (x'y + xy' + z)' 가 됨.

(AND-OR-INVERT 형식으로 AND-NOR, NAND-AND로 구현)

- 카노맵에서 1을 결합해서 보수를 취하면
- F = x'y'z' + xyz'

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

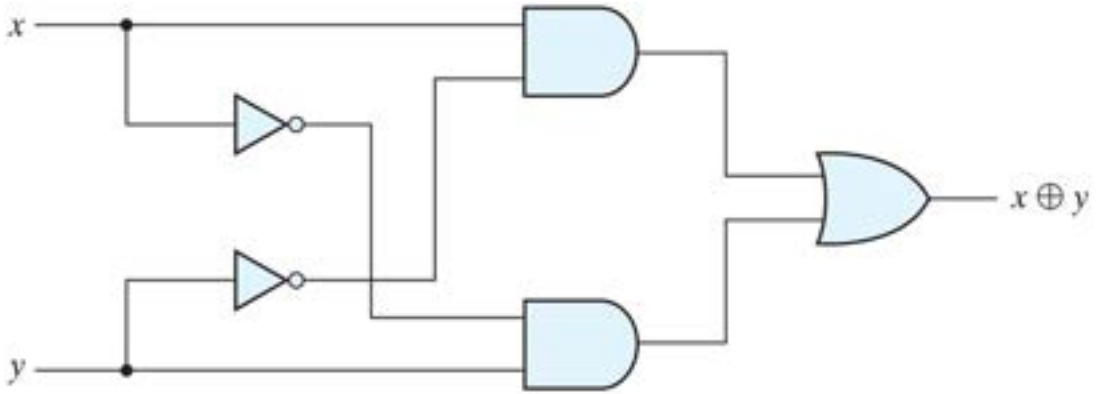
F = [(x + y + z)(x' + y' + z)]'

3.8 XOR(⊕) 구현

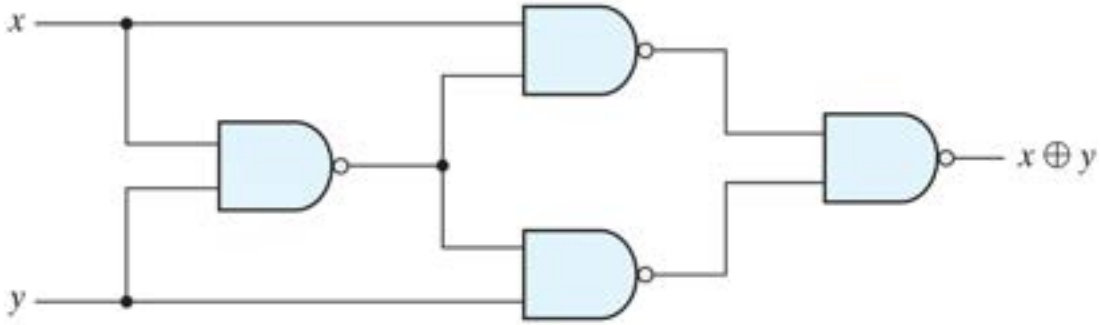
- 산술연산과 에러 검출 및 수정 회로에 사용함
- XOR (배타적논리합, exclusive OR): $x \oplus y = xy' + x'y$
  - x가 1이거나 y가 1이면 1임 → 1이 홀수개면 1임 (기함수, odd func.)
- XNOR (배타적논리부정합, exclusive NOR): $(x \oplus y)' = (xy' + x'y)'$
  - x, y가 둘 다 1이거나 둘 다 0이면 1임
- $x \oplus 0 = x$, $x \oplus 1 = x'$, $x \oplus x = 0$, $x \oplus x' = 1$, $x \oplus y' = x' \oplus y = (x \oplus y)'$
- $x \oplus y = y \oplus x$, $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$

- $x \oplus y = xy' + x'y = (x' + y')x + (x' + y')y$ 이므로 그림 3.30처럼 구현 가능

Figure 3.30
Logic diagrams for exclusive-OR implementations.



(a) Exclusive-OR with AND–OR–NOT gates

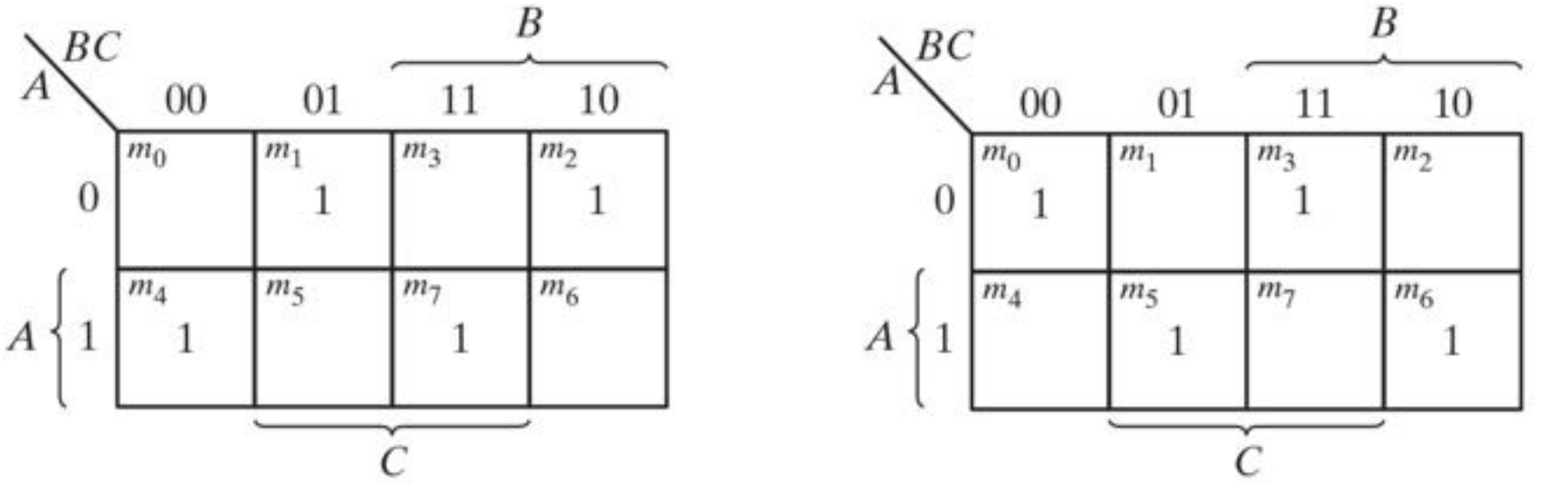


(b) Exclusive-OR with NAND gates

- 기함수(odd function): 1의 갯수가 홀수인 함수
- 기함수의 보수는 우함수(even function): 1의 갯수가 짝수인 함수

$$A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' \oplus C'D)$$
$$= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D)$$
$$= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$$

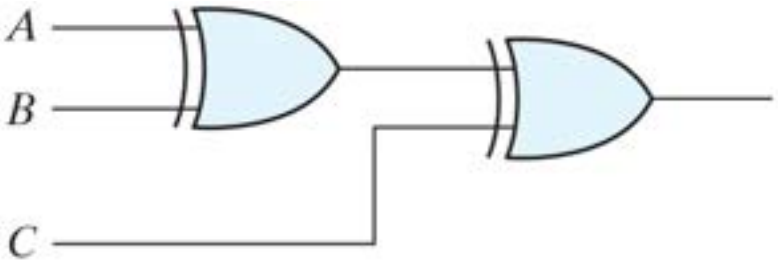
Figure 3.31
Map for a three-variable exclusive-OR function.



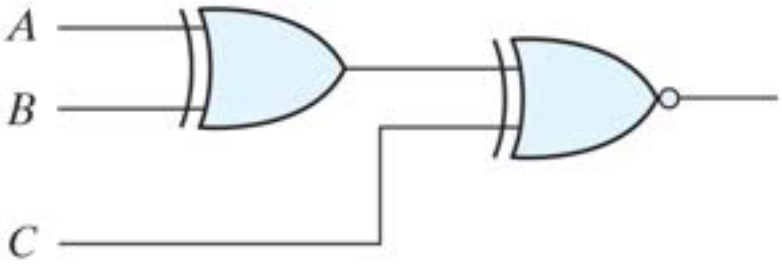
(a) Odd function $F = A \oplus B \oplus C$

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

Figure 3.32
Logic diagram of odd and even functions.

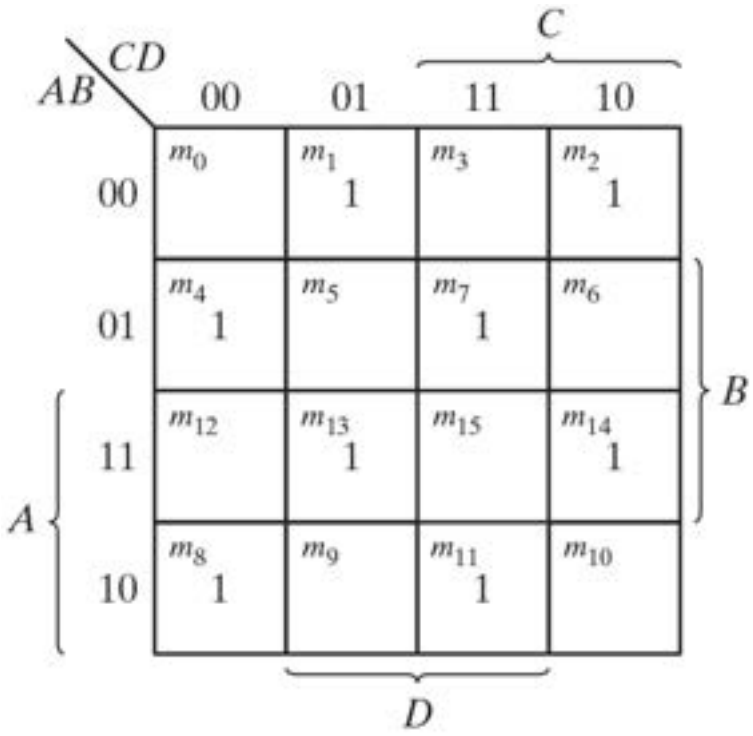


(a) 3-input odd function

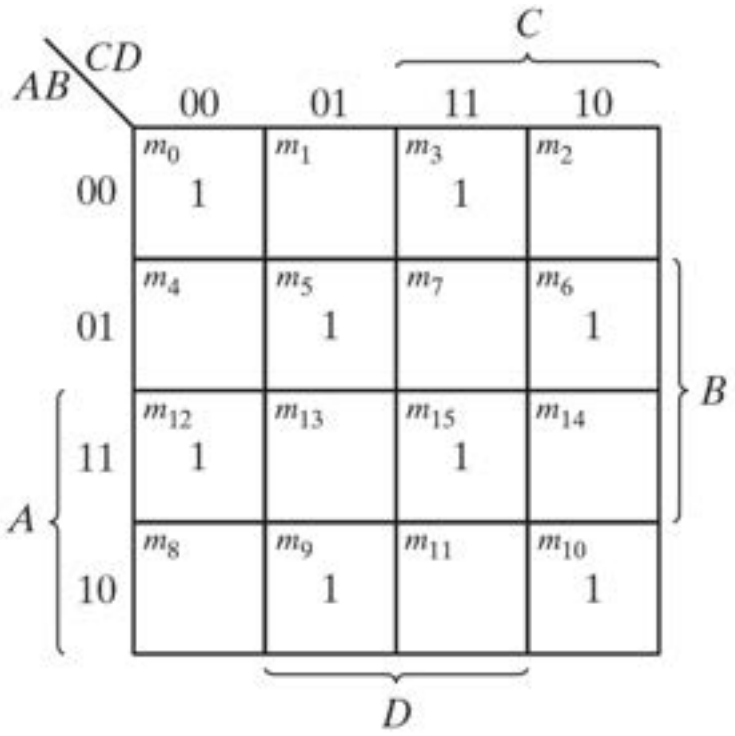


(b) 3-input even function

Figure 3.33
Map for a four-variable exclusive-OR function.



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



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

- 패리티 생성과 검사
  - 패리티 비트는 2진 정보의 전송 중에 에러 검출 목적으로 사용
  - 1의 개수를 홀수 또는 짝수로 만드는 2진 정보를 포함하는 여분의 비트
  - 패리티 비트를 포함하는 메시지는 수신 측에서 1의 개수를 체크해서 에러를 검사함
  - 전송장치에서 패리티 비트를 생성하는 패리티 생성기(parity generator)와 수신측에서 패리티 비트를 검사하는 패리티 검사기(parity checker)가 있음
  - 짝수 패리티(1의 갯수가 짝수)와 홀수 패리티(1의 갯수가 홀수)

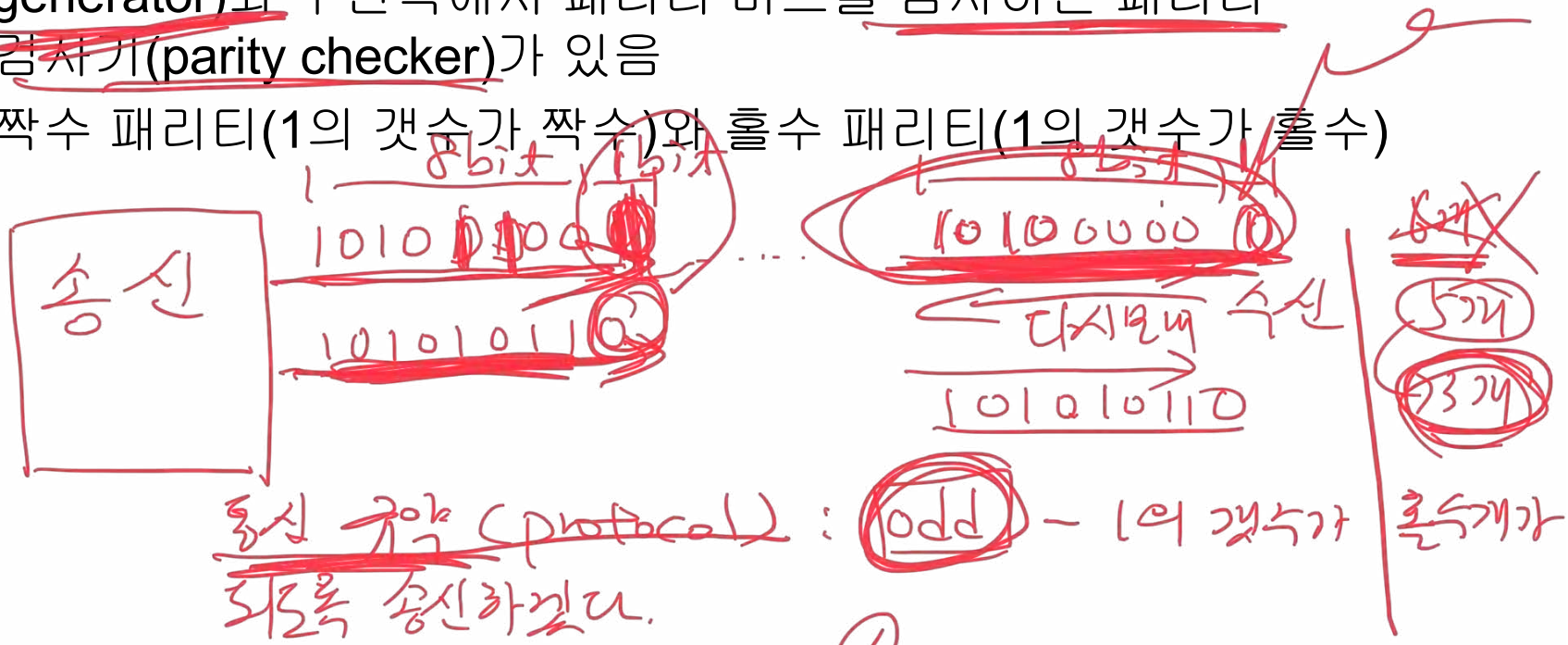


Table 3.3
Even-Parity-Generator Truth Table.

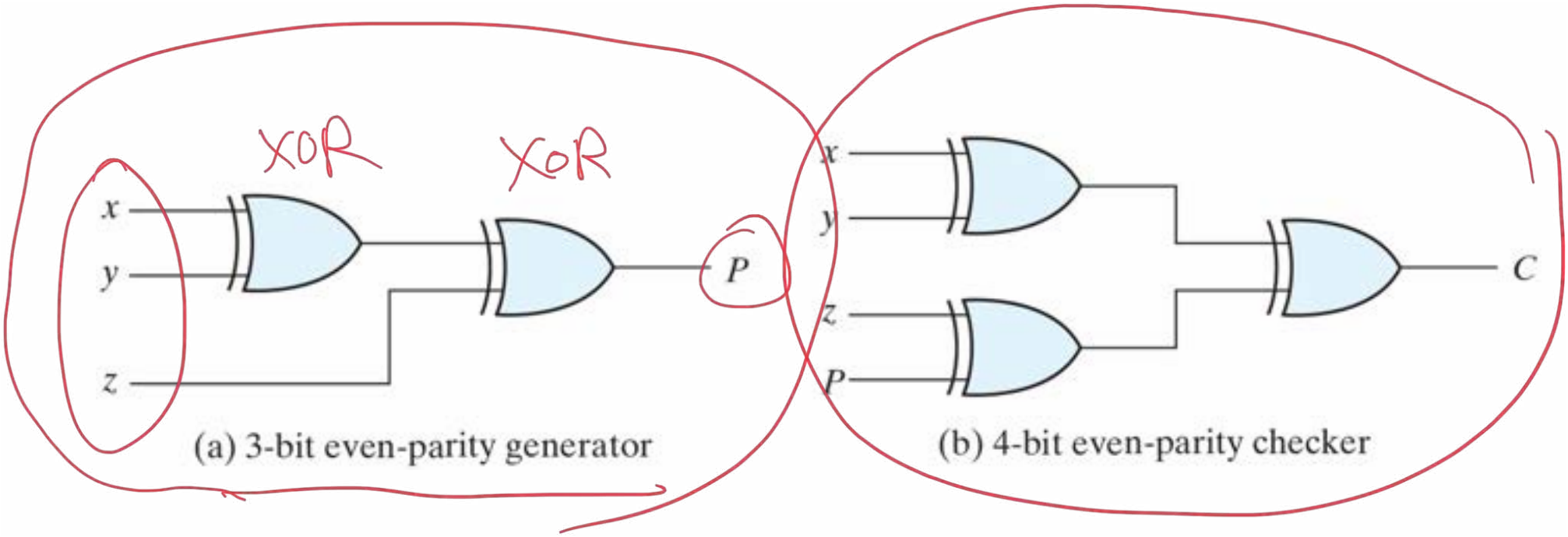
Three-Bit Message			Parity Bit
x	y	z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

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

Pearson

Figure 3.34
Logic diagram of a parity generator and checker.



3.9 하드웨어 기술 언어

- 디지털회로설계 및 언어(EE361)에서 자세히 배움
- 논리회로에서는 하지 않지만, 용어는 이해하면 좋음
- 숙제로 용어 정리해서 1주일 후에 제출
  - HDL
  - 논리 시뮬레이션
  - 논리 합성
  - 타이밍 검증
  - 결함 시뮬레이션

Figure 3.37
Schematic for *and_or_prop_delay.*

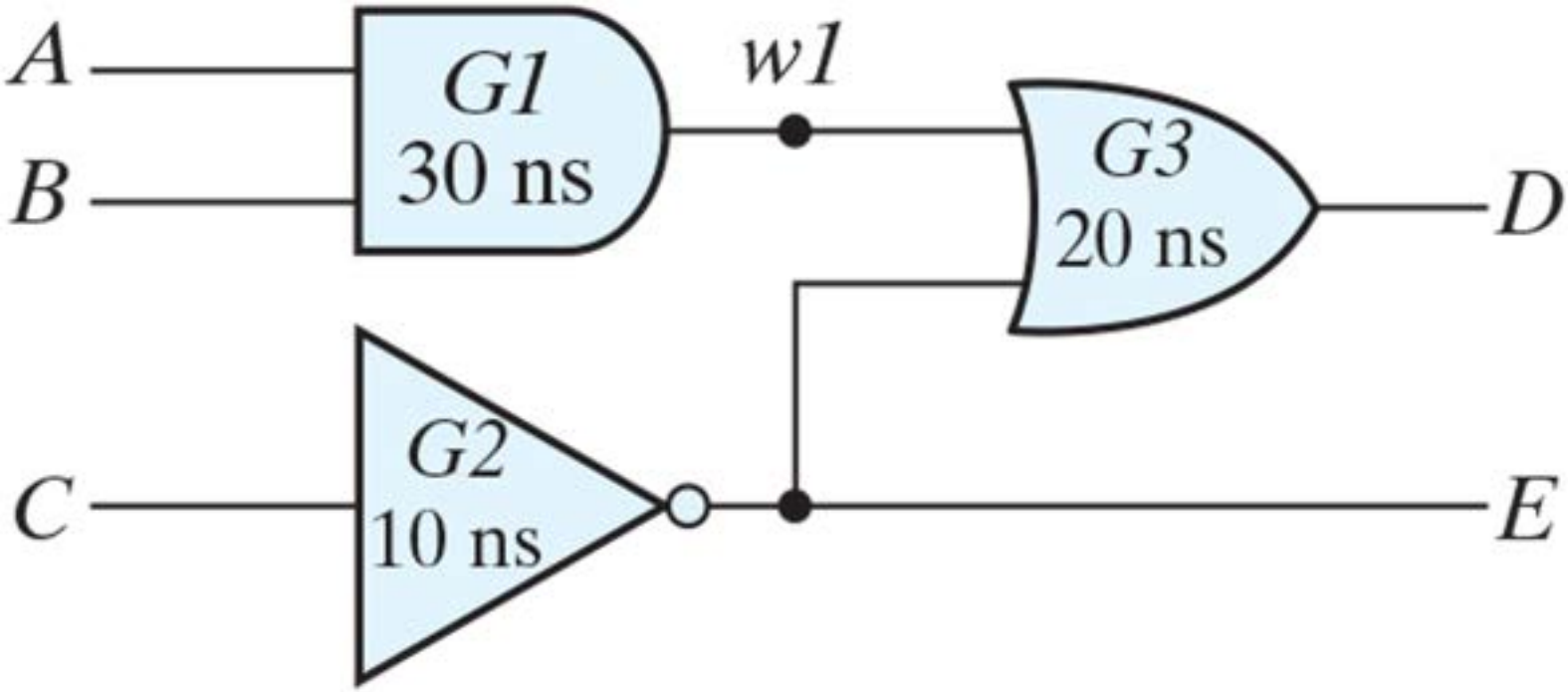


Table 3.5
Output of Gates after Delay.

	Time Units (ns)	Input	Output		
		A B C	*E*	*w1*	*D*
Initial	—	0 0 0	1	0	1
Change	—	1 1 1	1	0	1
	10	1 1 1	0	0	1
	20	1 1 1	0	0	1
	30	1 1 1	0	1	0
	40	1 1 1	0	1	0
	50	1 1 1	0	1	1

Figure 3.38
Simulation waveforms of *and_or_prop_delay*.