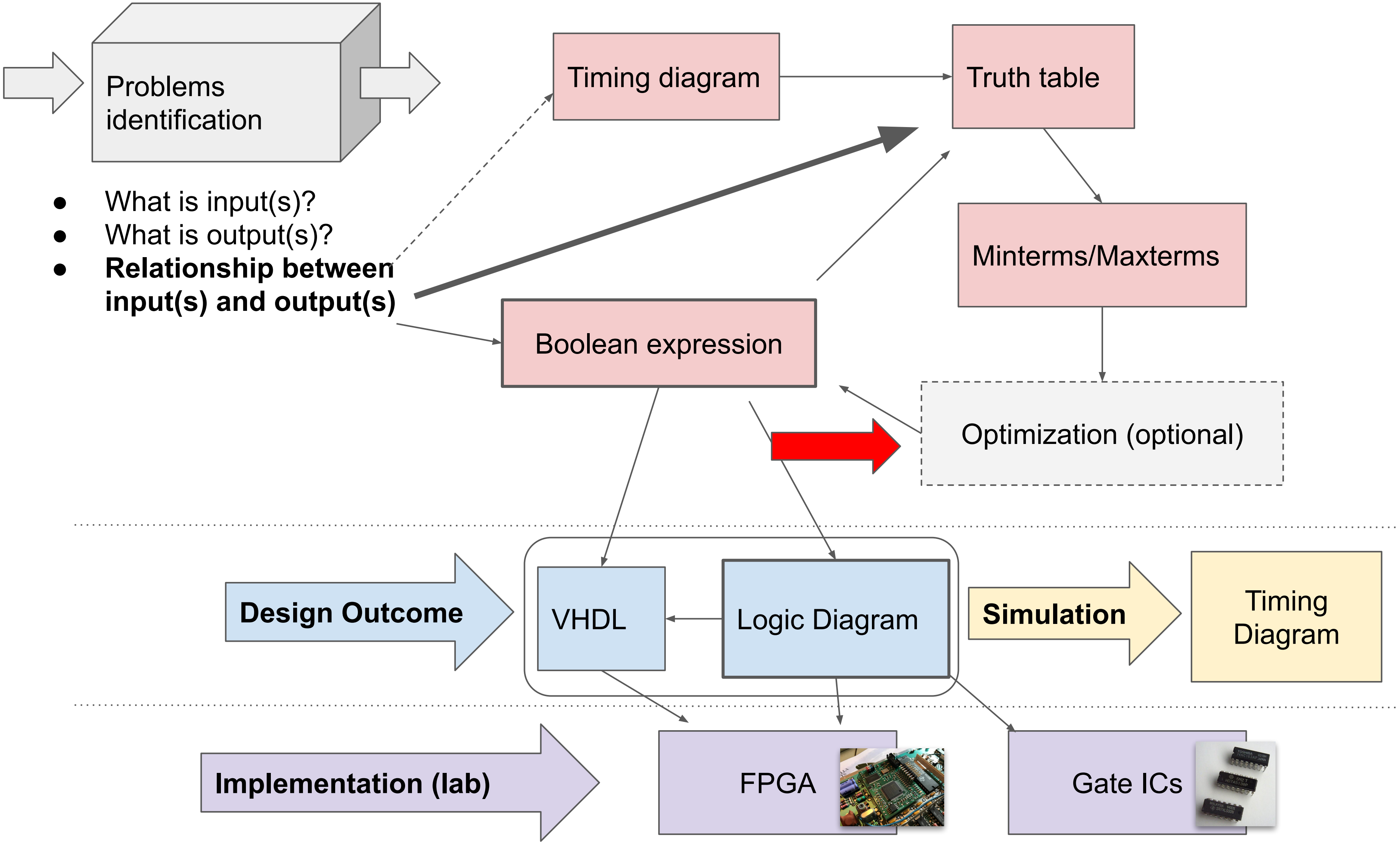
ELECTRICAL
Engineering KMITNB

Optimization Implementation of Logic Functions

Chapter 4

Summary of Digital Logic Design



Problems identification

- What is input(s)?
- What is output(s)?
- **Relationship between input(s) and output(s)**

Timing diagram

Truth table

Minterms/Maxterms

Boolean expression

Optimization (optional)

Design Outcome

VHDL

Logic Diagram

Simulation

Timing Diagram

Implementation (lab)

FPGA

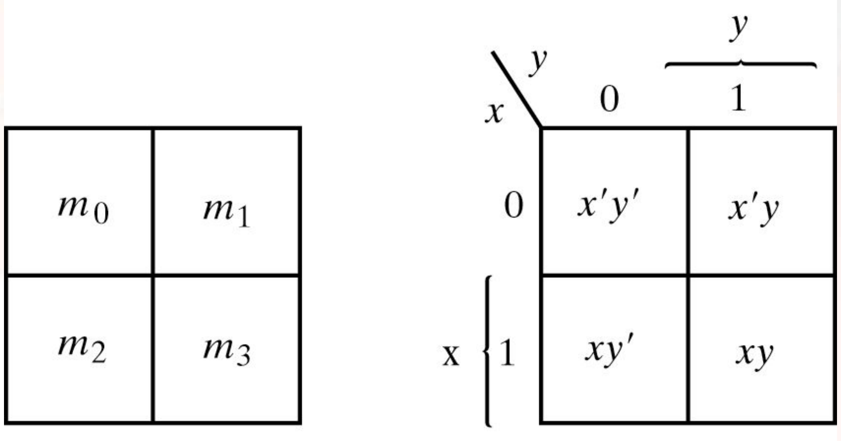
Gate ICs

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 straightforward procedure
    - a pictorial form of a truth table
    - applicable if the # of variables < 7
  - QM Method (optional)
- A diagram made up of squares
  - each square represents one minterm

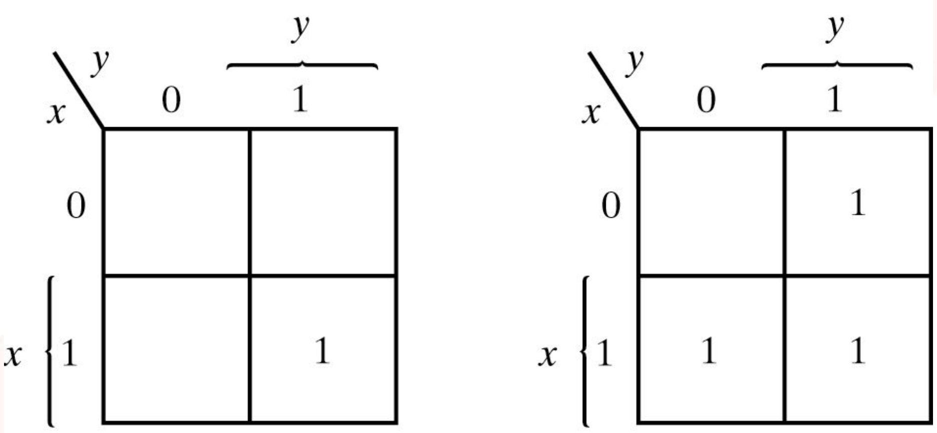
Two-Variable Map

- A two-variable map
  - four minterms
  - x' = row 0; x = row 1
  - y' = column 0; y = column 1
  - a truth table in square diagram
  - xy = m_3
  - x+y = Π(0)
    = Σ(1, 2, 3)



m_0	m_1
m_2	m_3

$x \backslash y$	0	1
0	$x'y'$	$x'y$
1	xy'	xy



$x \backslash y$	0	1
0		
1		1

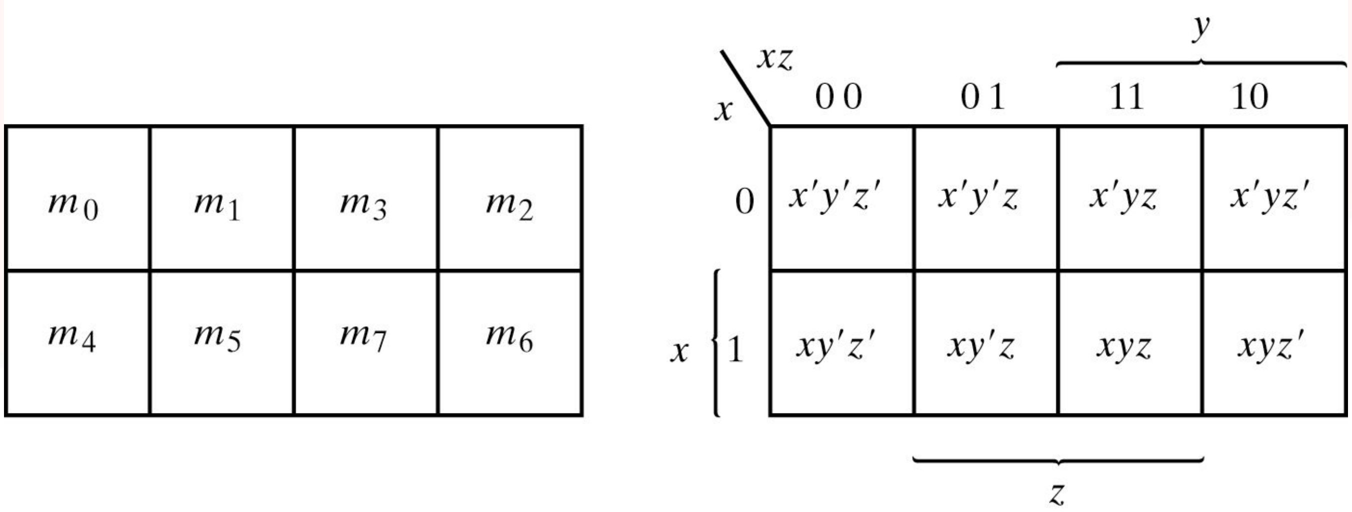
(a) xy

$x \backslash y$	0	1
0		1
1	1	1

(b) $x + y$

A three-variable map

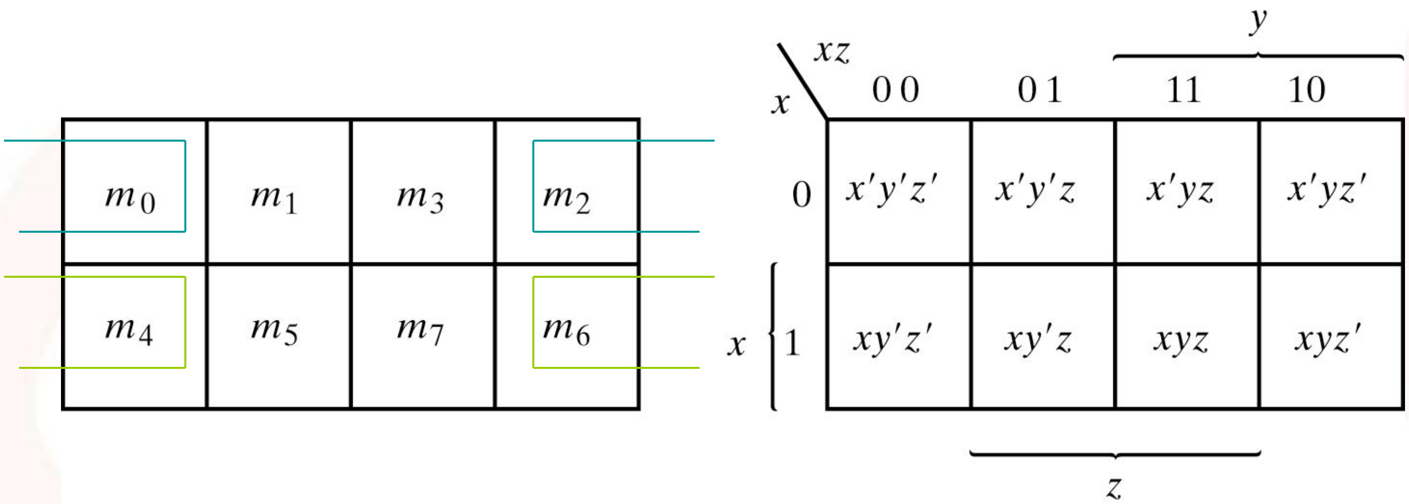
- eight minterms
- the Gray code sequence
  - any two adjacent squares in the map differ by only on variable
    - primed in one square and unprimed in the other
    - e.g., m_5 and m_7 can be simplified
    - $m_5 + m_7 = xy'z + xyz = xz\,(y'+y) = xz$



m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

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

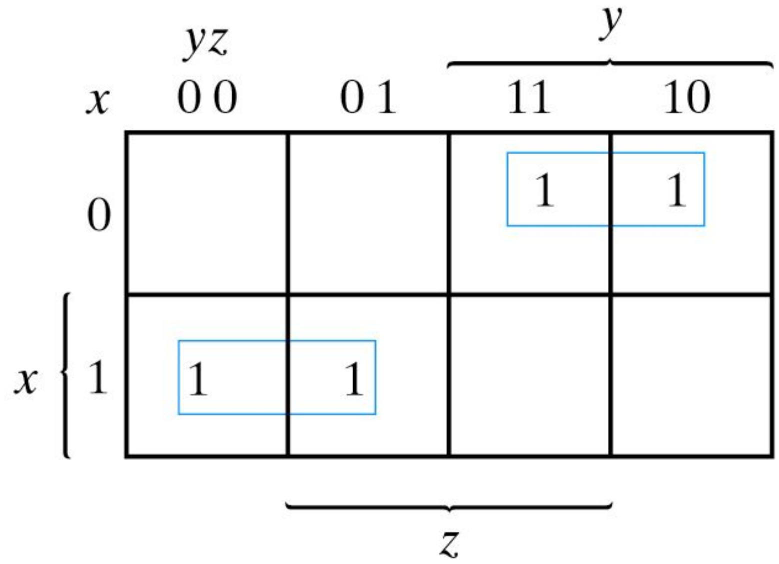
- m_0 and m_2 (m_4 and m_6) are adjacent
- $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y'+y) = x'z'$
- $m_4 + m_6 = xy'z' + xyz' = xz'(y'+y) = xz'$



m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

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

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

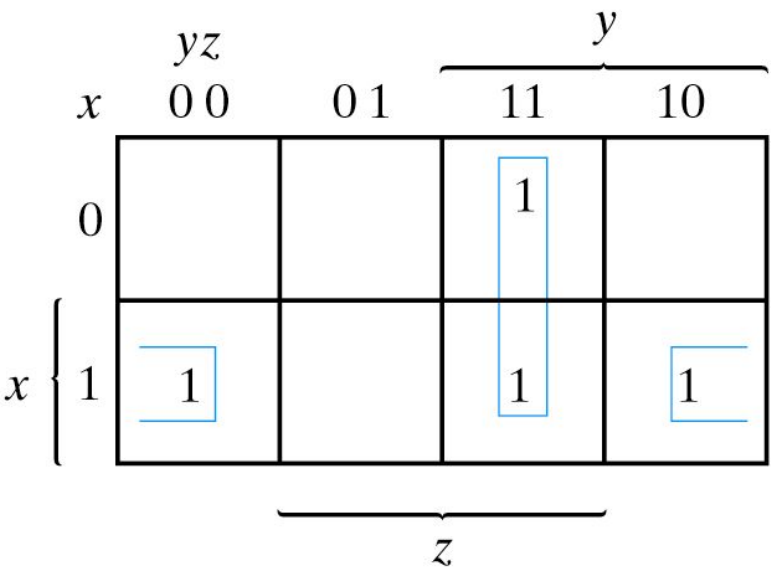


x \ yz	00	01	11	10
0			1	1
1	1	1		

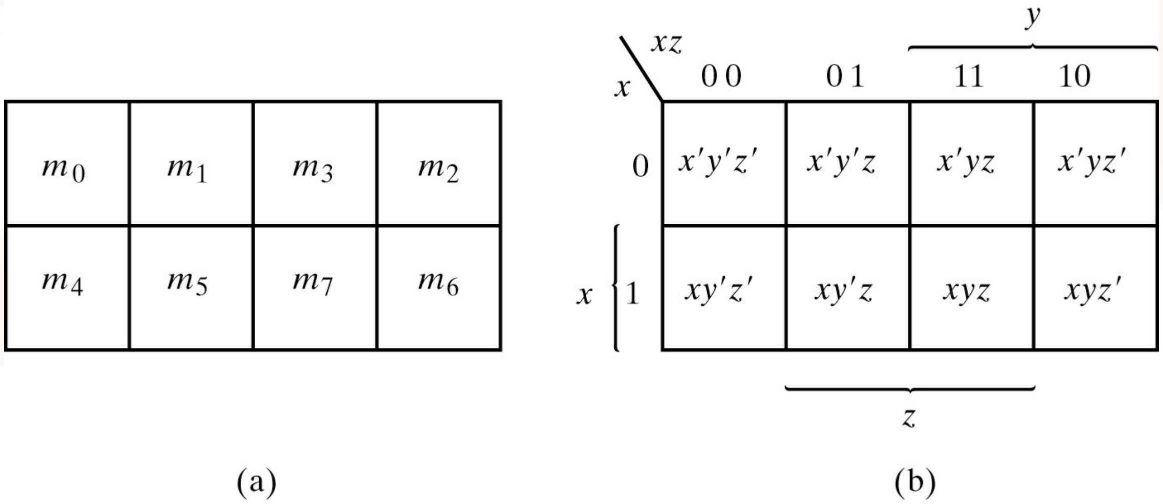
x′y

xy′

- Example 3-2
  - F(x,y,z) = Σm(3,4,6,7) = yz+ xz'



- Four adjacent squares
  - 2, 4, 8 and 16 squares
  - $m_0+m_2+m_4+m_6 = x'y'z'+x'yz'+xy'z'+xyz'$
    $= x'z'(y'+y) + xz'(y'+y) = x'z' + xz' = z'$
  - $m_1+m_3+m_5+m_7 = x'y'z+x'yz+xy'z+xyz$
    $=x'z(y'+y) + xz(y'+y) = x'z + xz = z$



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)

- Example 3-3
  - $F(x,y,z) = \Sigma m(0,2,4,5,6)$

Find the optimized logic function

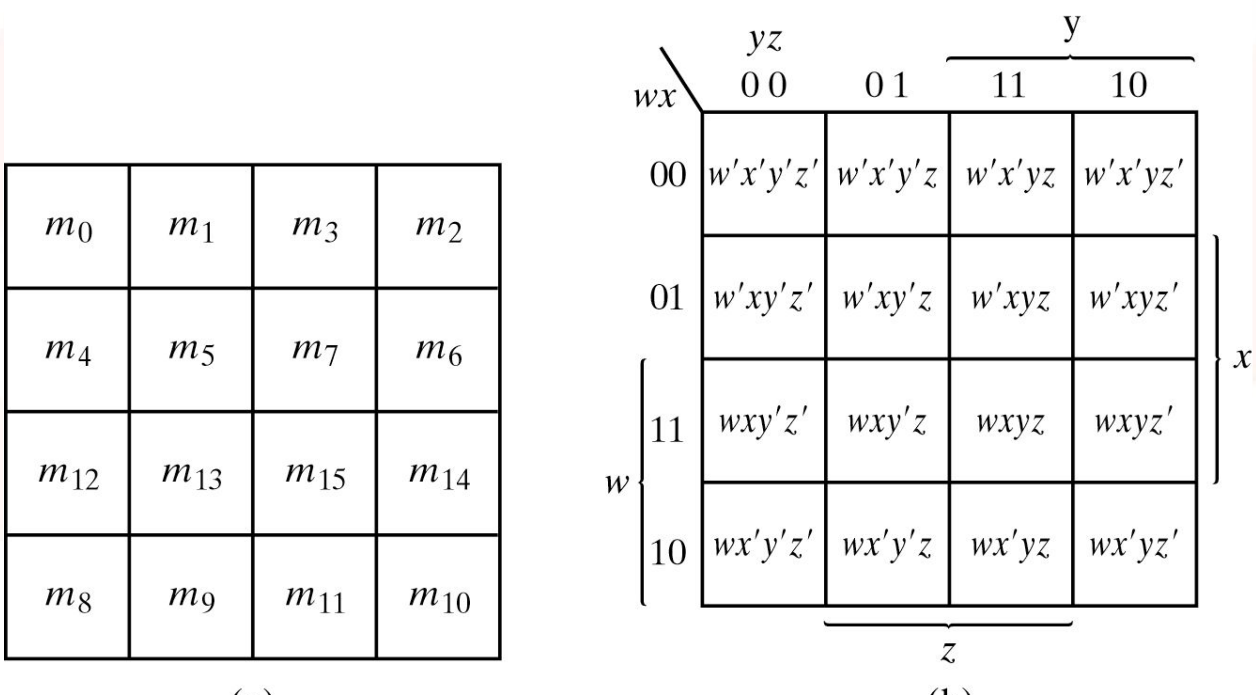
- Prime Implicants
  - all the minterms are covered
  - minimize the number of terms
  - a prime implicant: a product term obtained by combining the maximum possible number of adjacent squares (combining all possible maximum numbers of squares)
  - essential: a minterm is covered by only one prime implicant
  - the **essential PI** must be included

- Example 3-4
  - F = A'C + A'B + AB'C + BC

Find the optimized logic function

Four-Variable Map

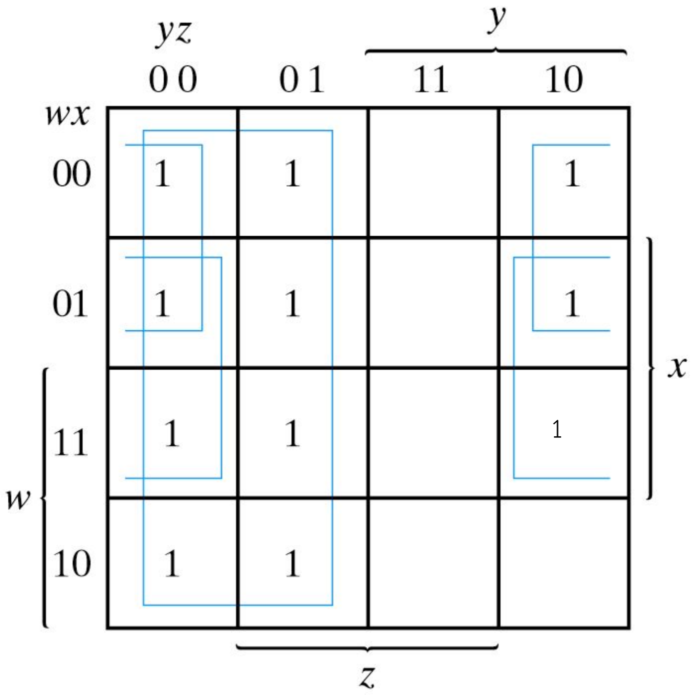
- The map
  - 16 minterms
  - combinations of 2, 4, 8, and 16 adjacent squares



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}

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	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

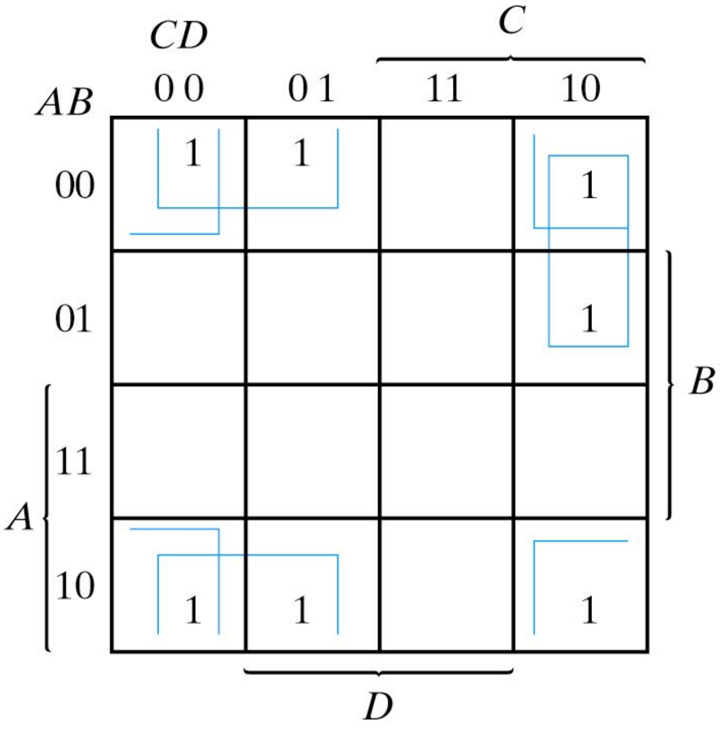
- Example 3-5
  - F(w,x,y,z) = Σ(0,1,2,4,5,6,8,9,12,13,14)



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

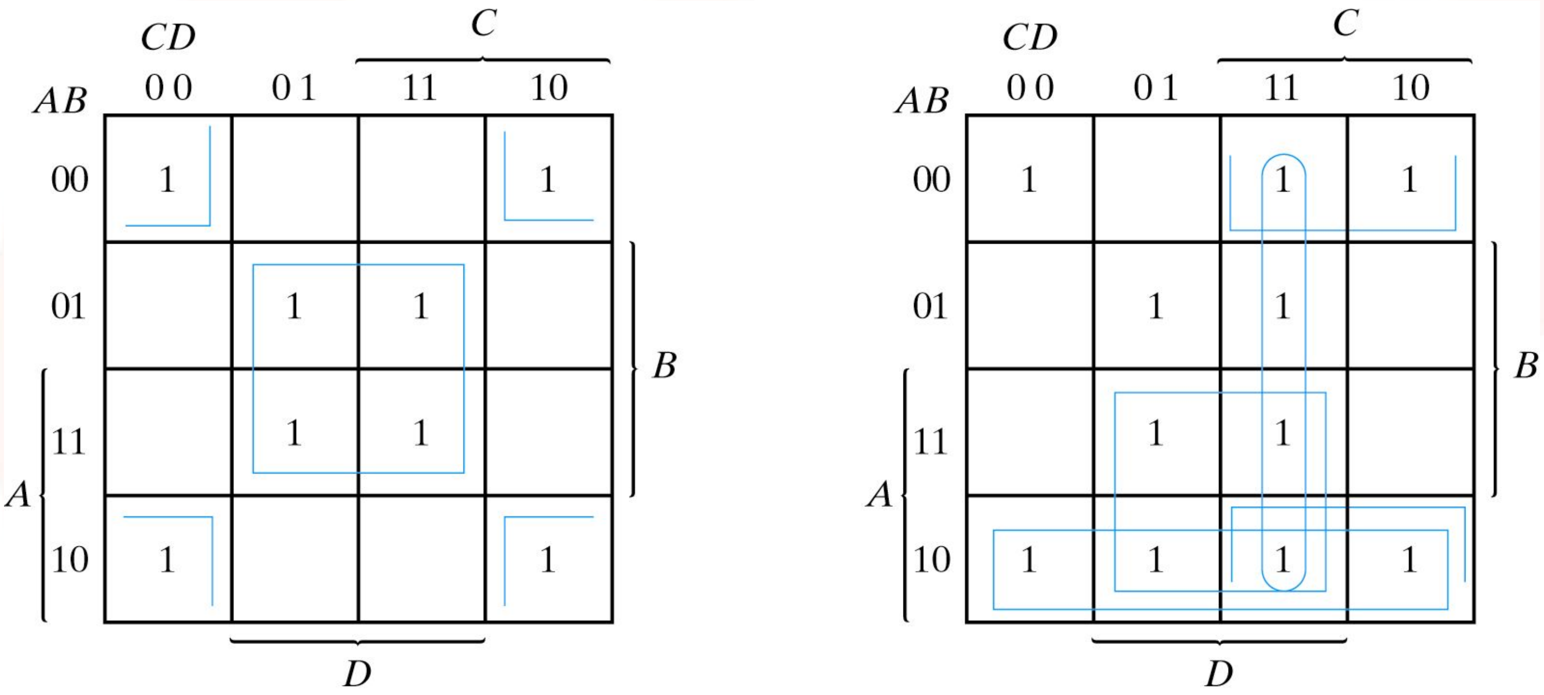
  - F = y'+w'z'+xz'

- Example 3-6
  - F(A,B,C,D) = ?



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

- 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'

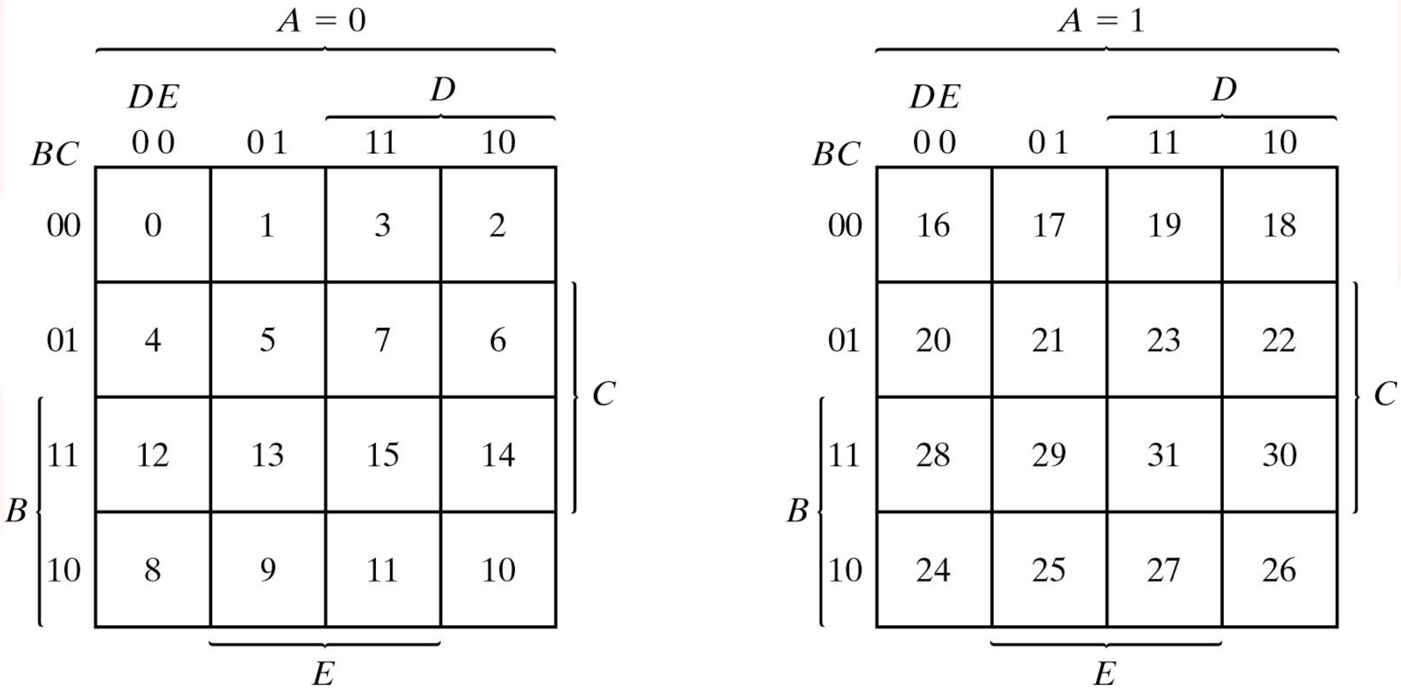


AB \ CD	00	01	11	10
00	1			1
01		1	1	
11		1	1	
10	1			1

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

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)



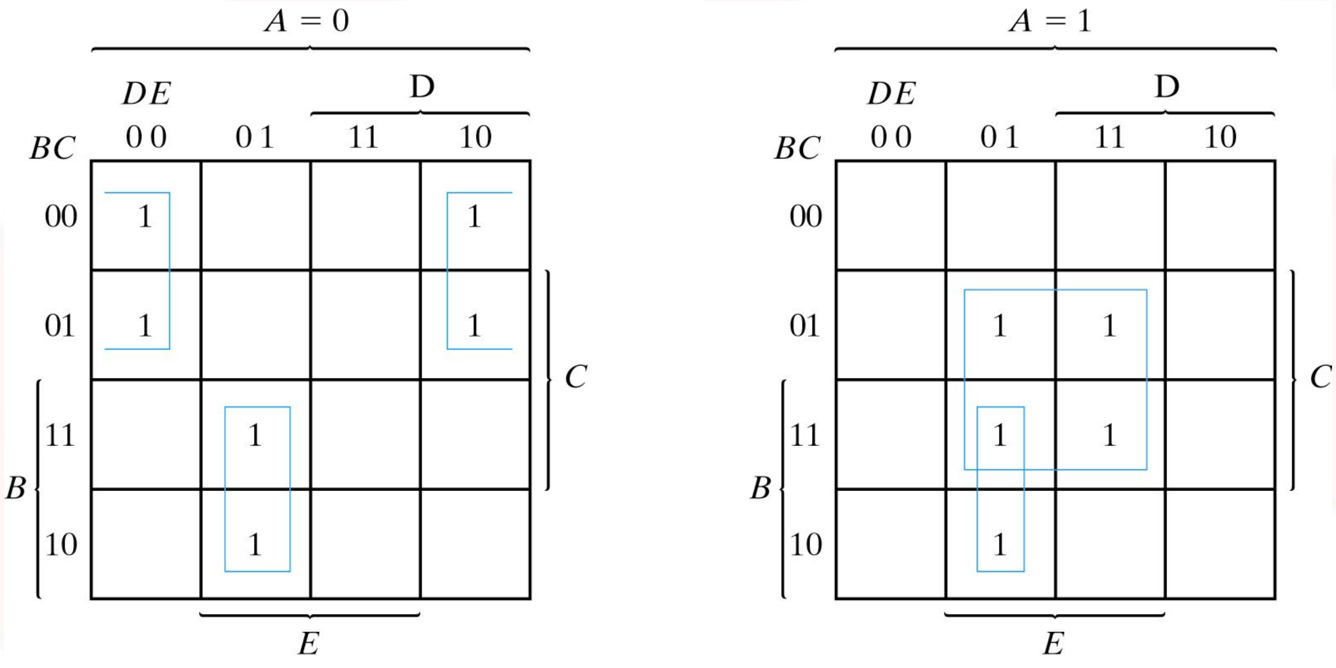
A = 0

BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

A = 1

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

- Example 3-7
  - $F = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$



A = 0

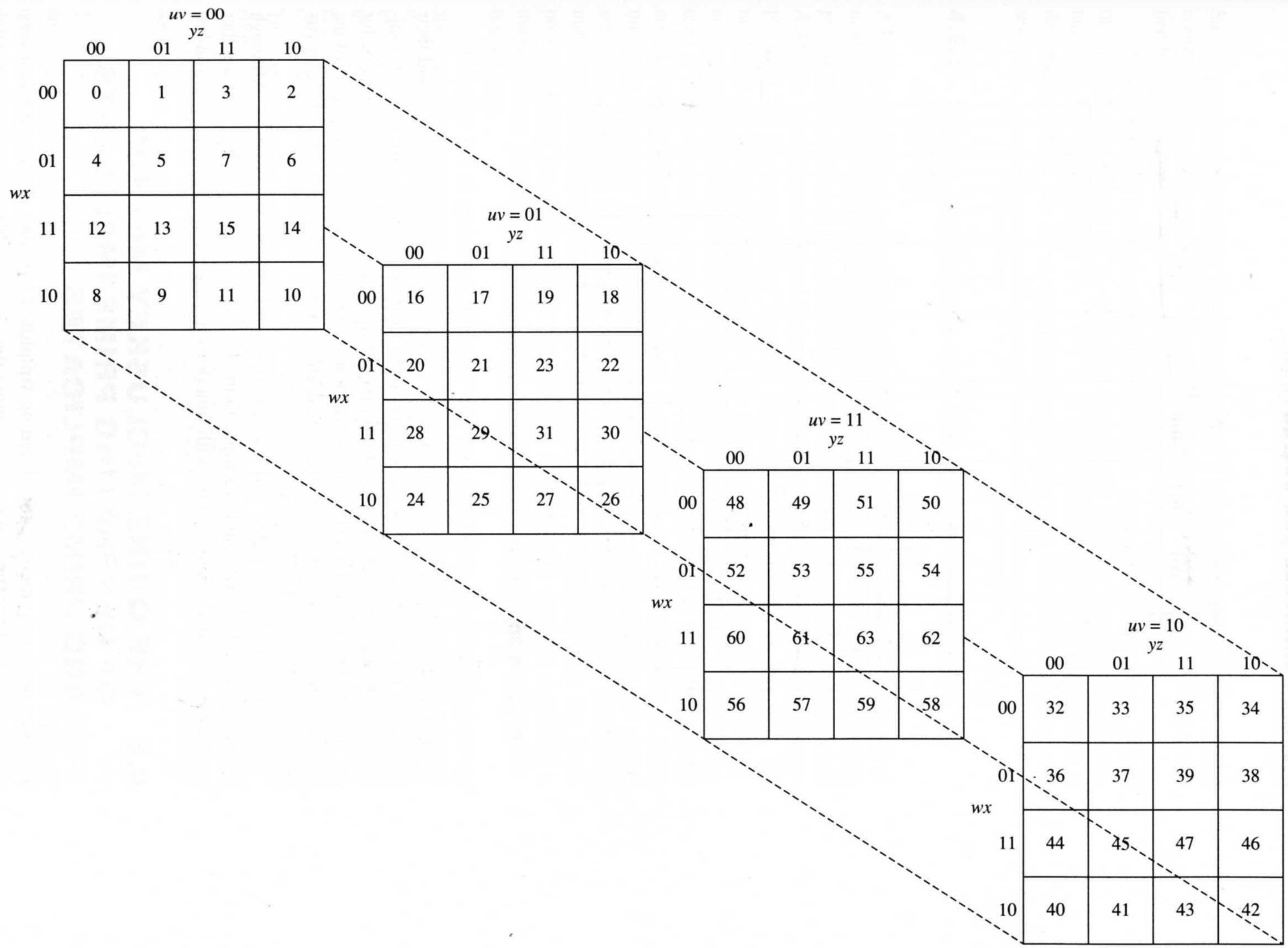
BC \ DE	00	01	11	10
00	1			1
01	1			1
11		1		
10		1		

A = 1

BC \ DE	00	01	11	10
00				
01		1	1	
11		1	1	
10		1		

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

Six-Variable Map



uv = 00
yz

wx	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

uv = 01
yz

wx	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

uv = 11
yz

wx	00	01	11	10
00	48	49	51	50
01	52	53	55	54
11	60	61	63	62
10	56	57	59	58

uv = 10
yz

wx	00	01	11	10
00	32	33	35	34
01	36	37	39	38
11	44	45	47	46
10	40	41	43	42

(*b*)

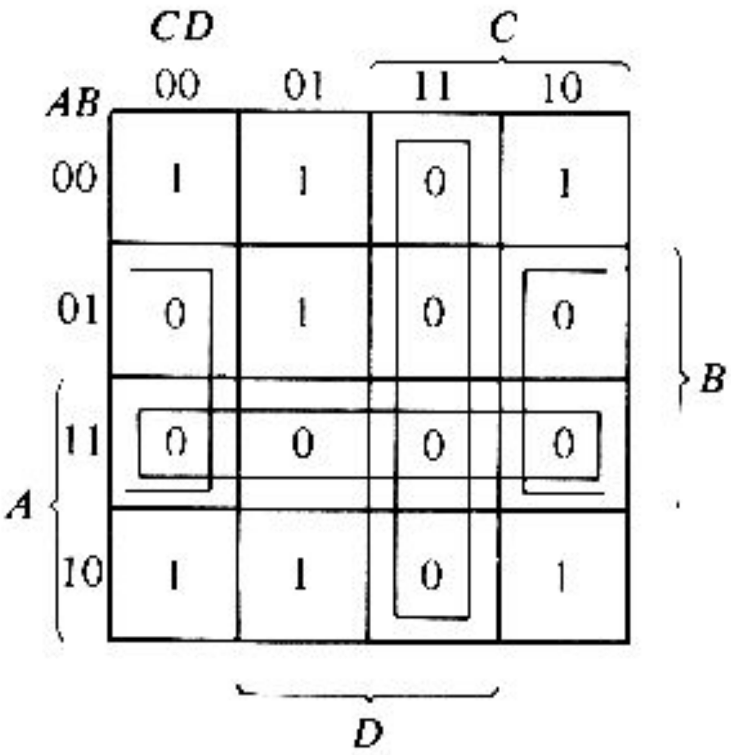
Product of Sums Simplification

- 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

CD

AB	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}

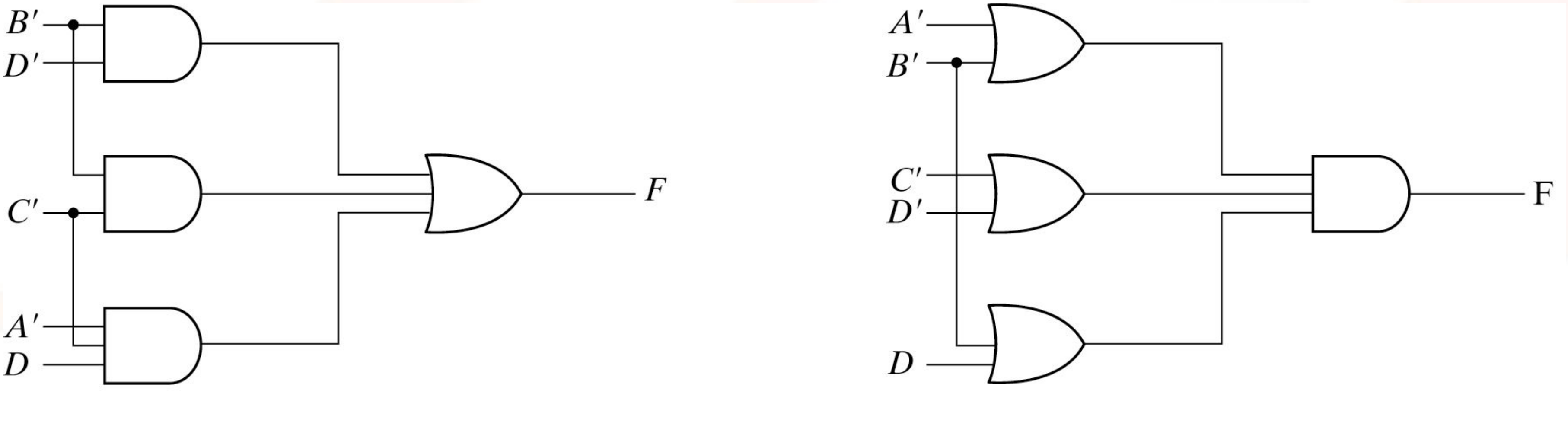
- Example 3-8
  - F = Σ(0,1,2,5,8,9,10)



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

  - F' = AB+CD+BD'
  - Apply DeMorgan's theorem; F=(A'+B')(C'+D')(B'+D)
  - Or think in terms of maxterms

- Gate implementation of the function of Example 3-8



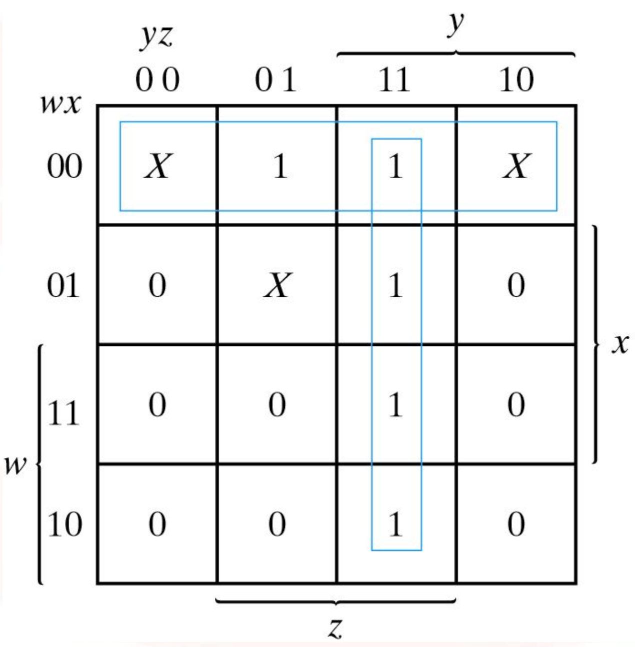
(a) $F = B'D' + B'C' + A'C'D$

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

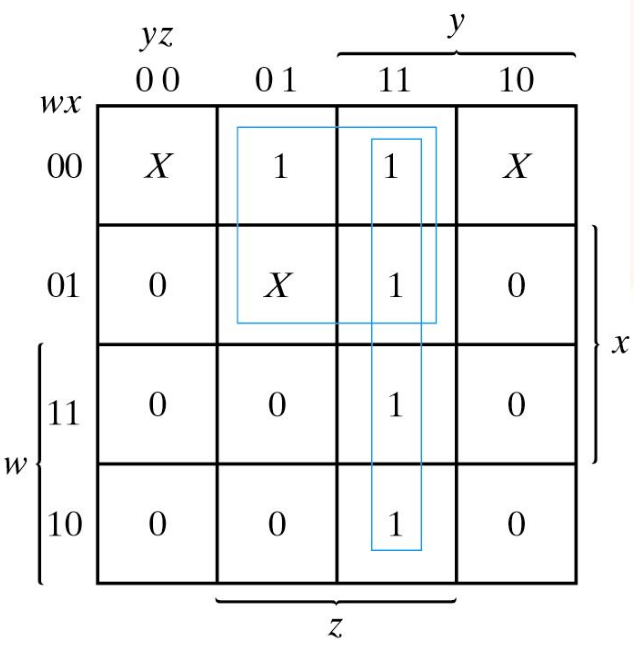
Don't-Care Conditions

- 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
  - $F(w,x,y,z) = \Sigma(1,3,7,11,15)$
  - $d(w,x,y,z) = \Sigma(0,2,5)$

- 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



wx \ yz	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

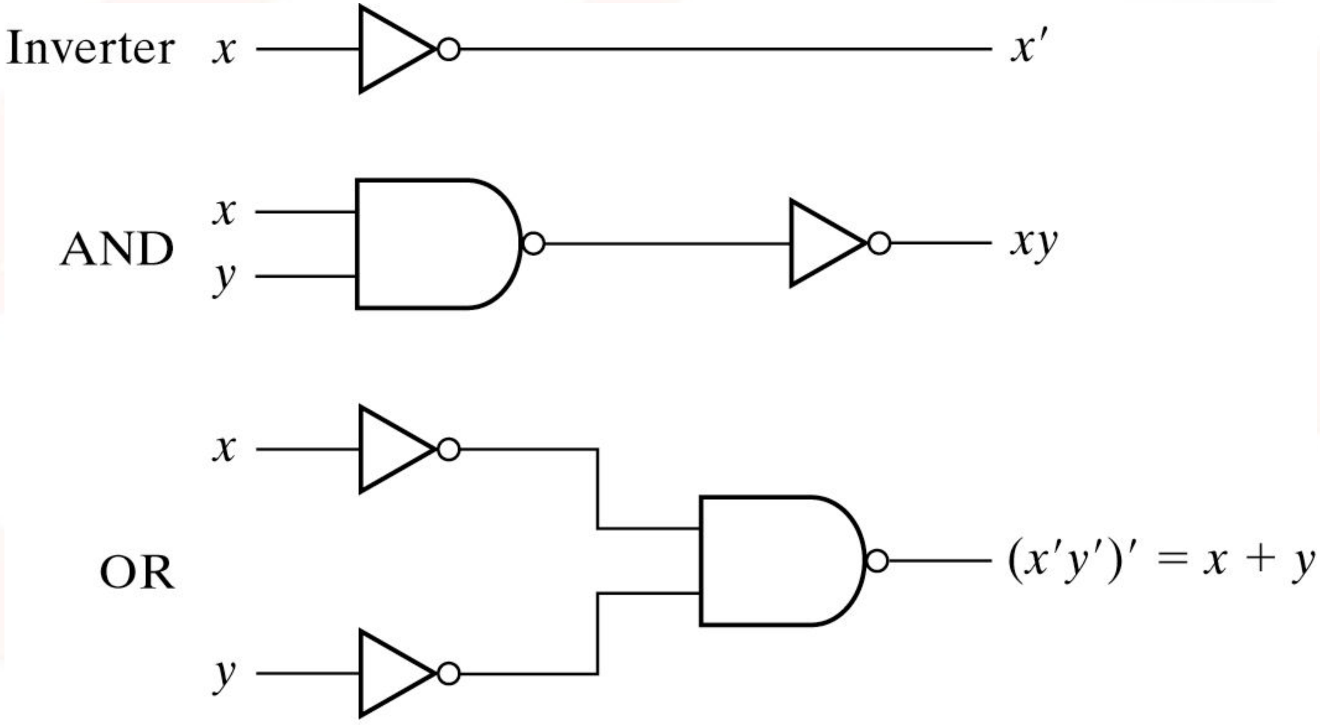


wx \ yz	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

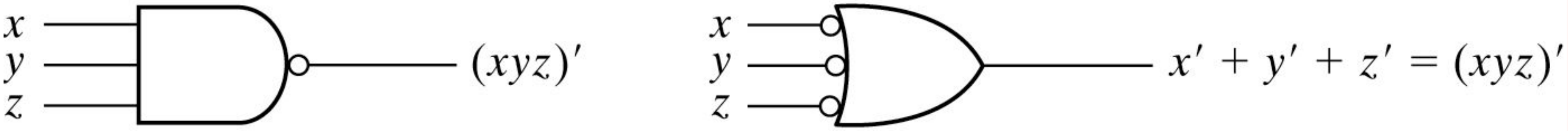
- Also apply to products of sum

NAND and NOR Implementation

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



- Two graphic symbols for a NAND gate

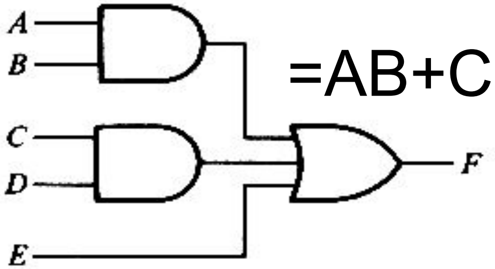


(a) AND–invert

(b) Invert–OR

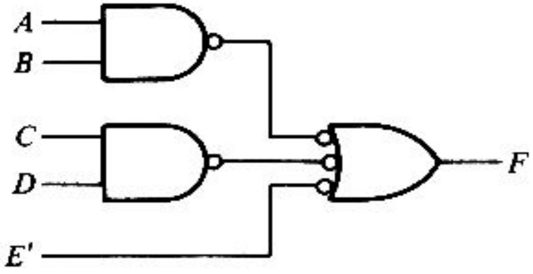
Two-level Implementation

- two-level logic
- NAND-NAND = sum of products
- Example: F = AB+CD+E
- F = ((AB)' (CD)' E')'

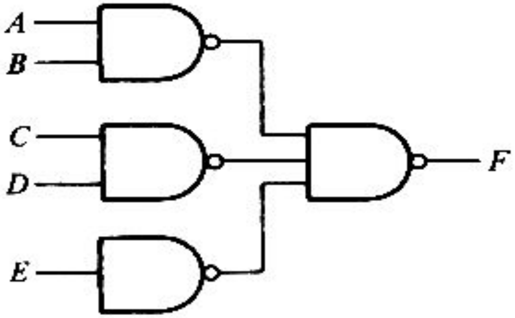


=AB+CD+E

(a) AND-OR

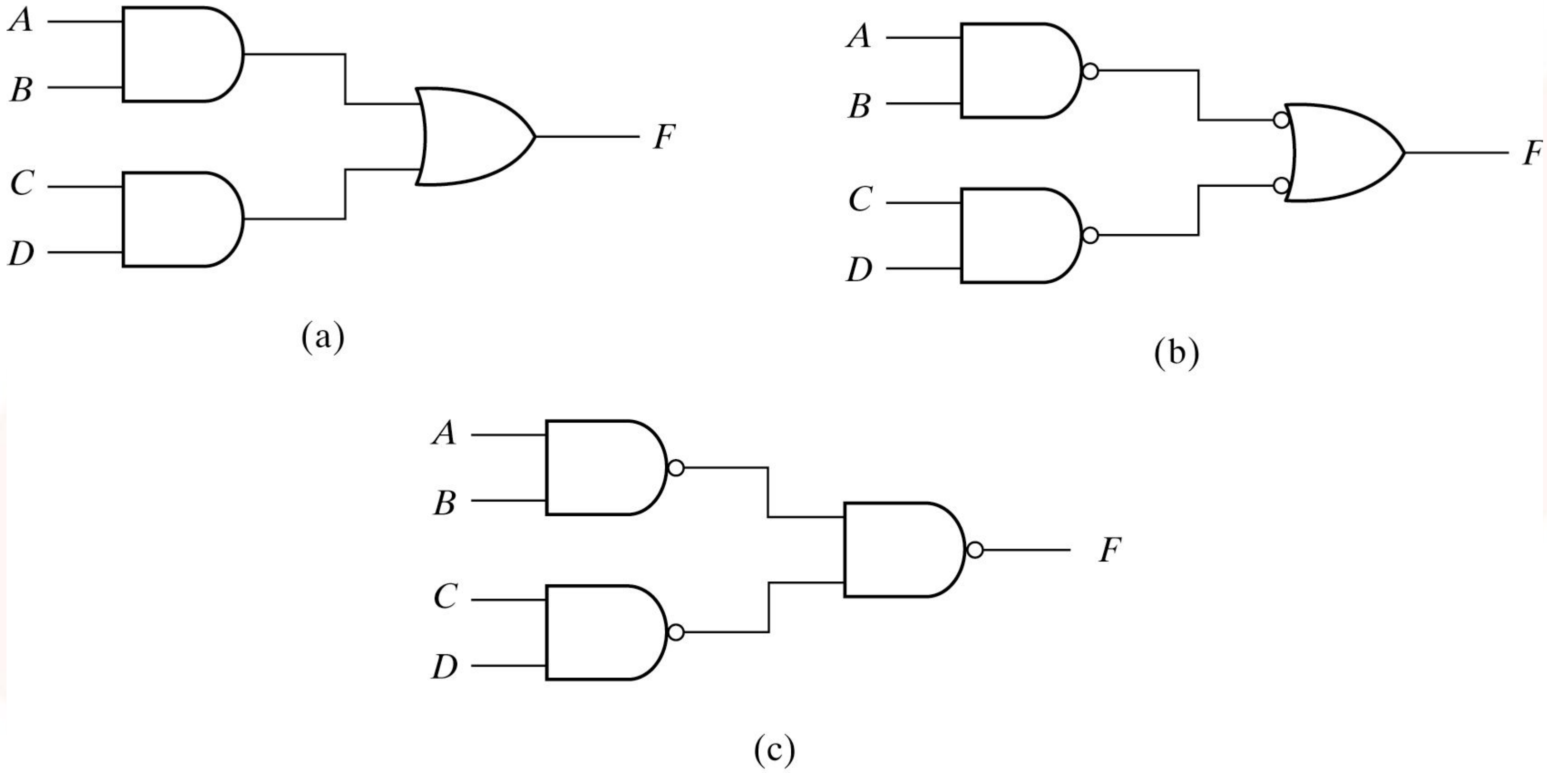


(b) NAND-NAND



(c) NAND-NAND

- F=AB+CD



(a)

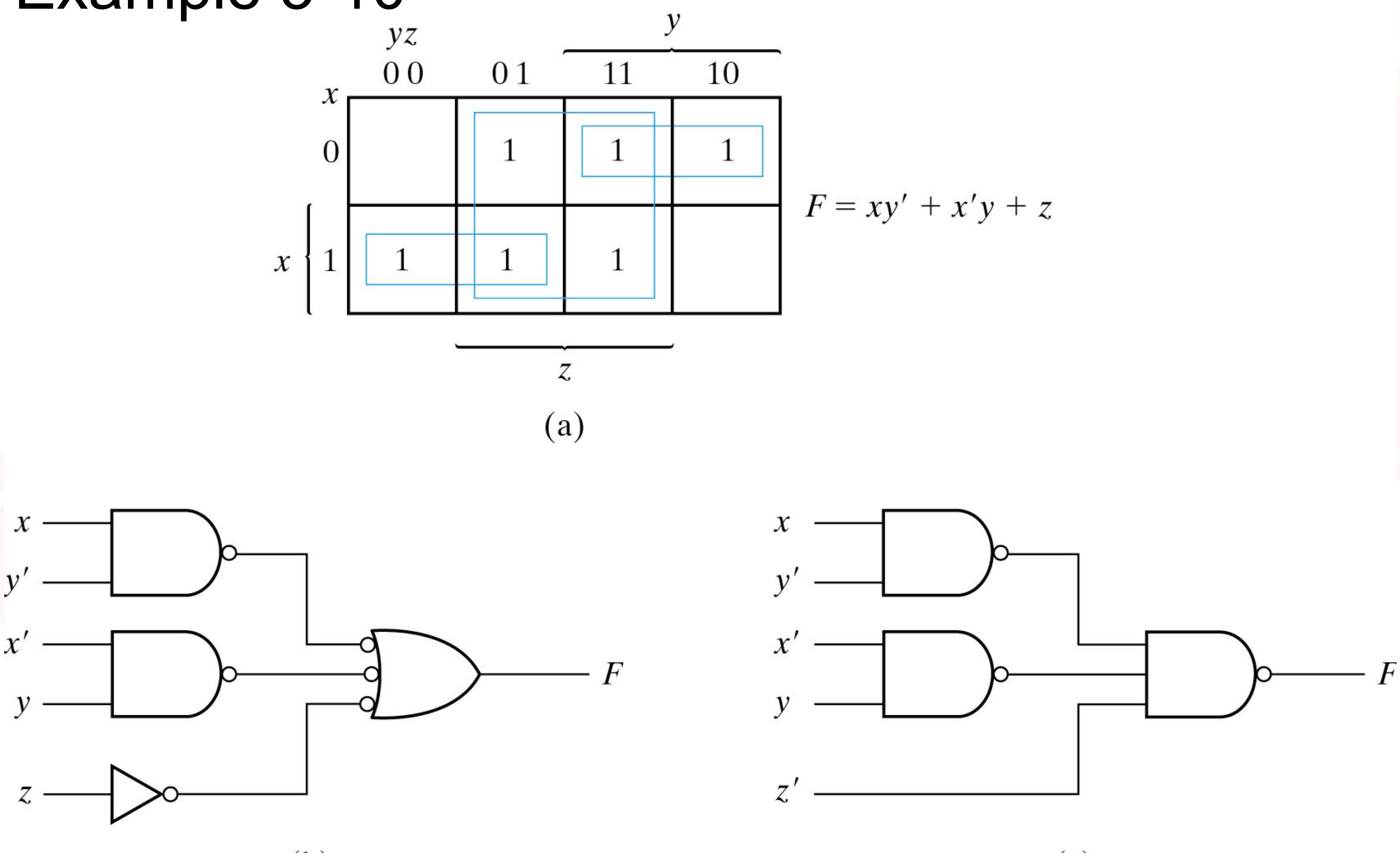
(b)

(c)

Fig. 3-20 Three Ways to Implement $F = AB + CD$

- The procedure for NAND-NAND implementation
  - 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
  - a single NAND gate for the second sum term

Example 3-10



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

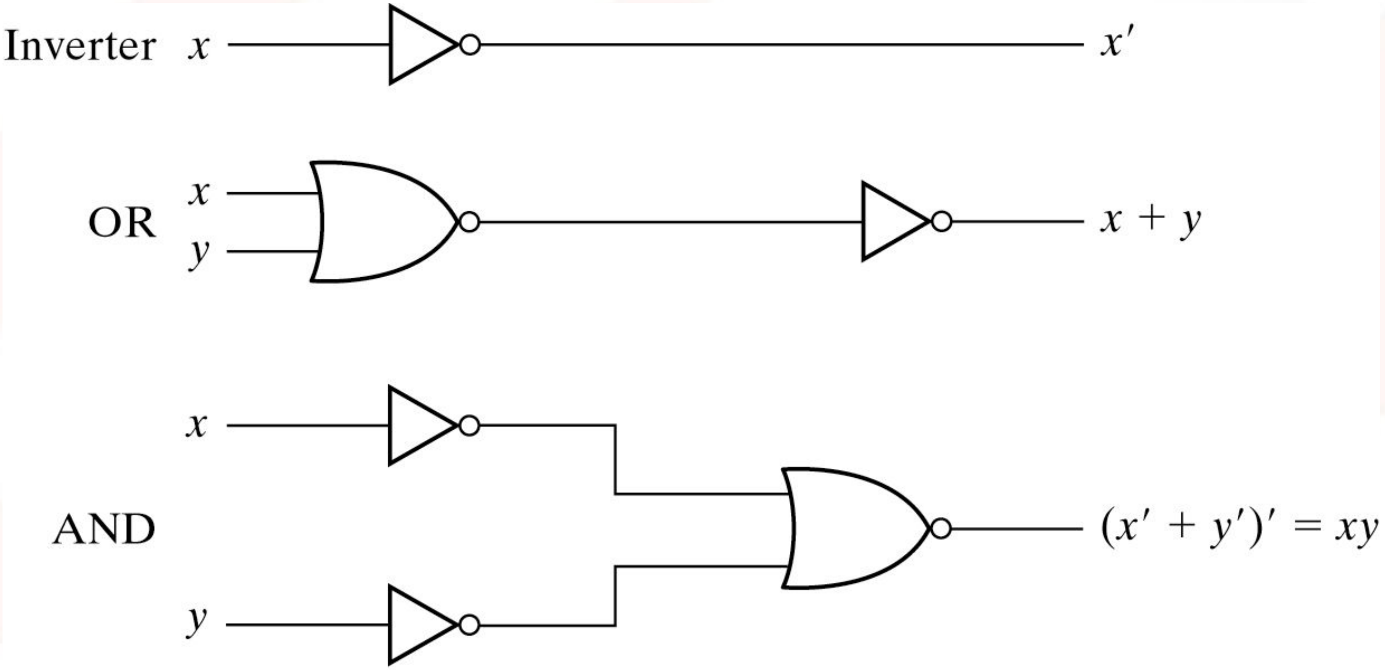
(a)

(b)

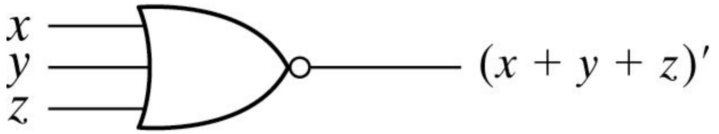
(c)

NOR Implementation

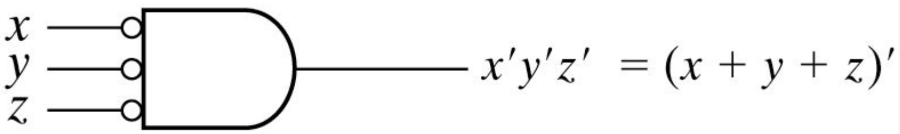
- NOR function is the dual of NAND function
- The NOR gate is also universal



- Two graphic symbols for a NOR gate

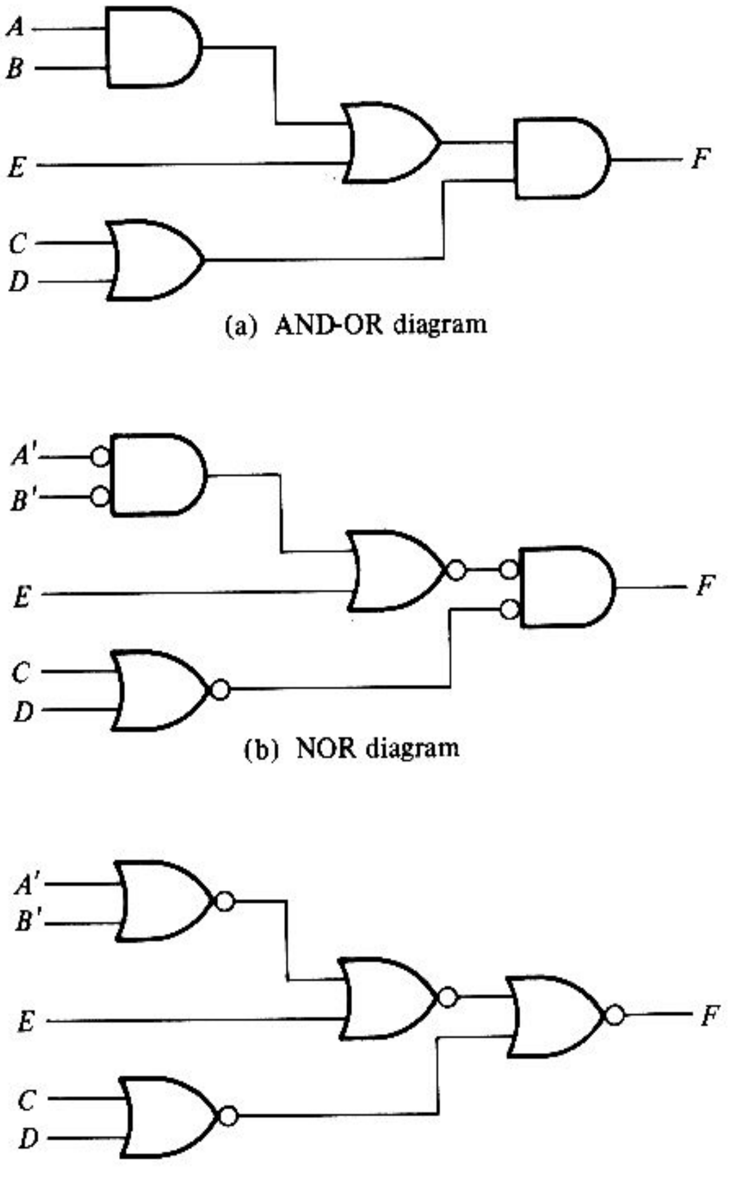


(a) OR–invert



(a) Invert–AND

- Boolean-function implementation
  - OR => NOR + INV
  - AND
    - INV + AND = NOR



(a) AND-OR diagram

(b) NOR diagram

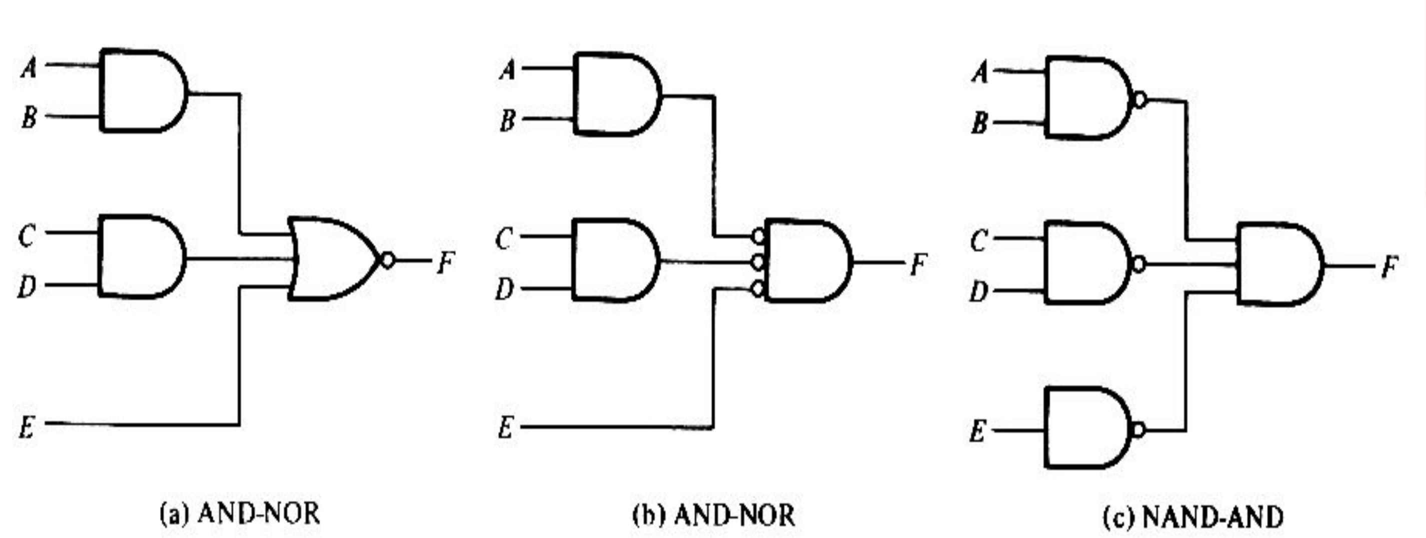
(c) Alternate NOR diagram

Other two level implementations

- 16 possible combinations of two-level forms
  - eight of them: degenerate forms = a single operation
    - AND-AND, OR-OR, AND-NAND, OR-NOR, NAND-OR, NOR-AND, NOR-NAND, NAND-NOR
  - 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
    - NOR-OR, NAND-AND, OR-NAND, AND-NOR = ?

AND-OR-Invert Implementation

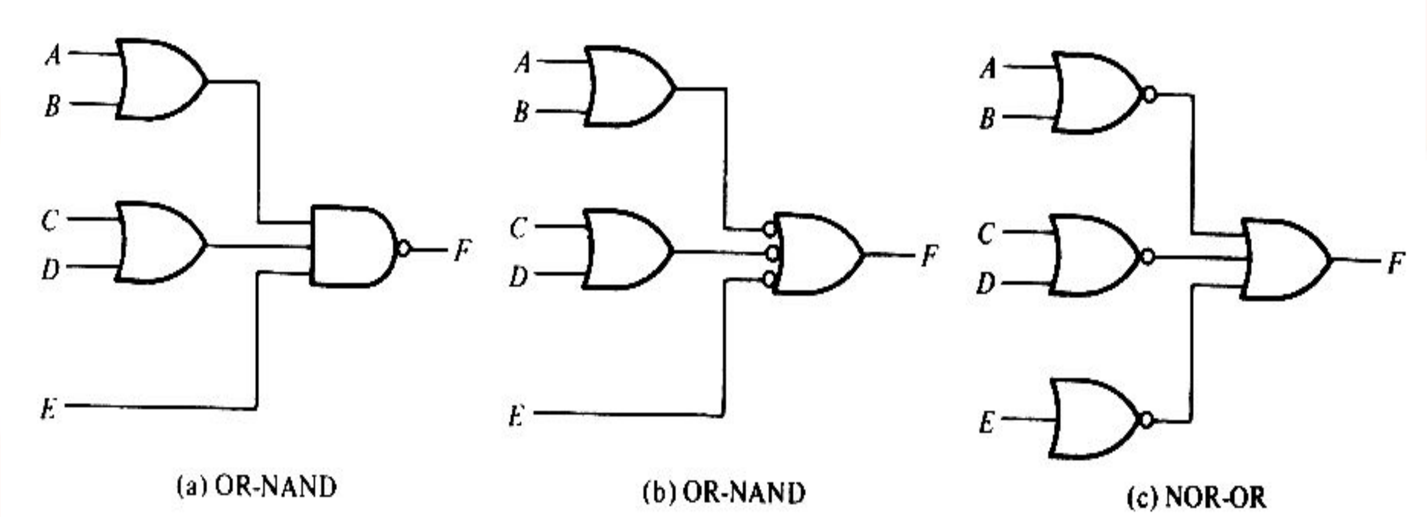
- AND-OR-INVERT (AOI) Implementation
  - NAND-AND = AND-NOR = AOI
  - F = (AB+CD+E)'
  - F' = AB+CD+E (sum of products)



(a) AND-NOR (b) AND-NOR (c) NAND-AND

  - simplify F' in sum of products

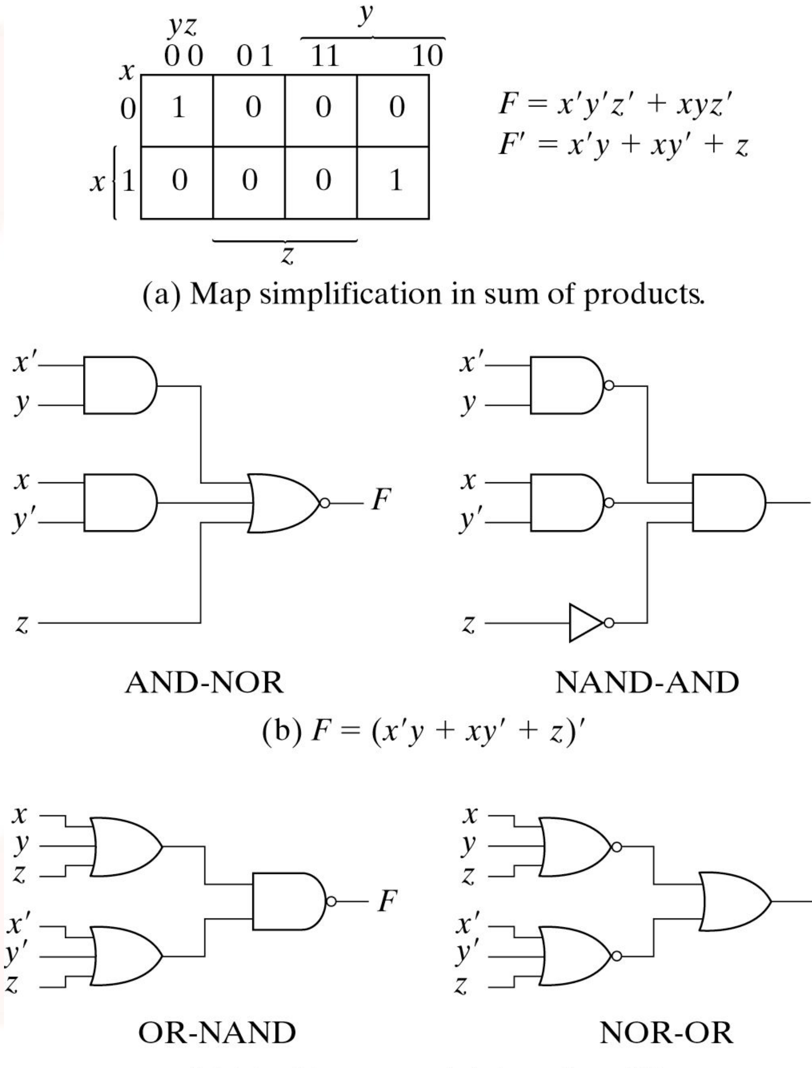
- OR-AND-INVERT (OAI) Implementation
  - OR-NAND = NOR-OR = OAI
  - F = ((A+B)(C+D)E)'
  - F' = (A+B)(C+D)E (product of sums)



(a) OR-NAND
(b) OR-NAND
(c) NOR-OR

  - simplified F' in products of sum

- Example 3-11
  - F' = x'y+xy'+z (F': sum of products)
  - F = (x'y+xy'+z)' (F: AOI implementation)
  - 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)



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

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

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

(a) Map simplification in sum of products.

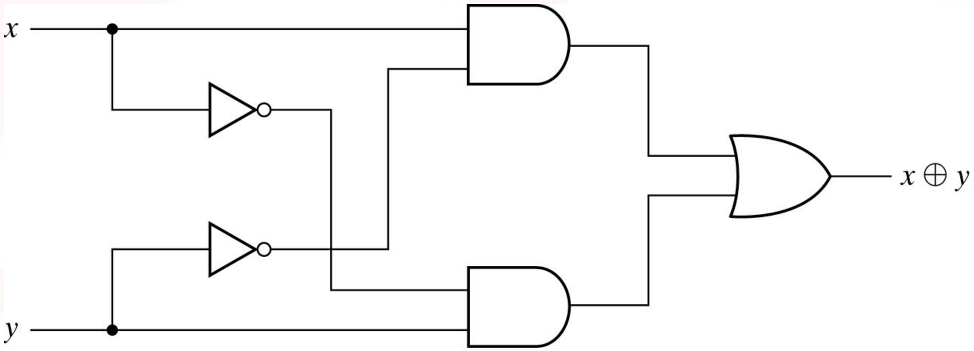
(b) $F = (x'y + xy' + z)'$

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

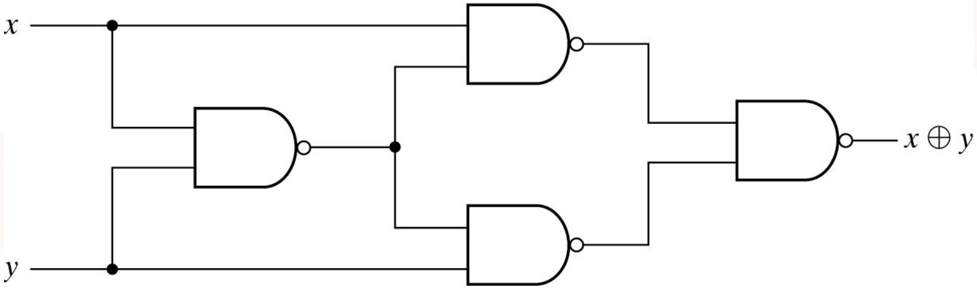
Exclusive-OR Function

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

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



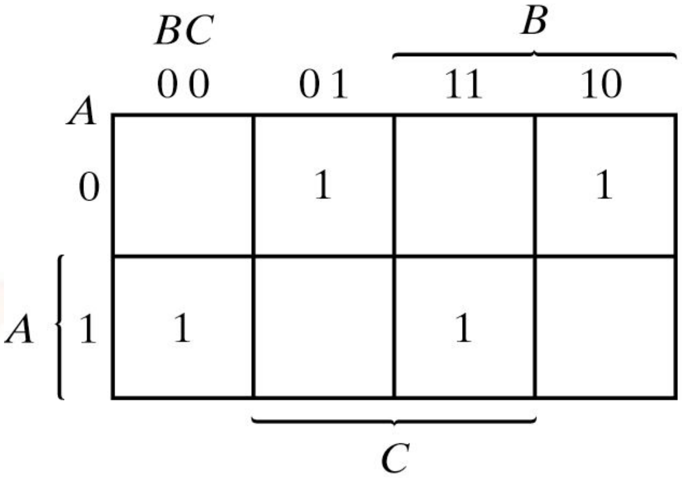
(a) With AND-OR-NOT gates



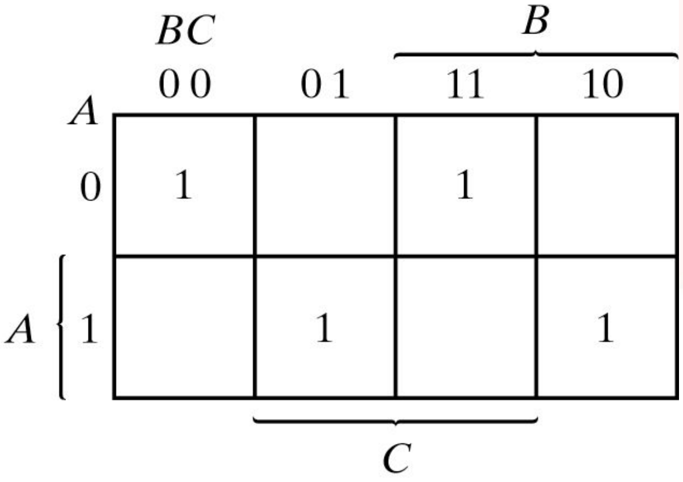
(b) With NAND gates

Odd and Even functions

- A⊕B⊕C = (AB'+A'B)C' +(AB+A'B')C
- = AB'C'+A'BC'+ABC+A'B'C = Σ(1,2,4,7)
- an odd number of 1's

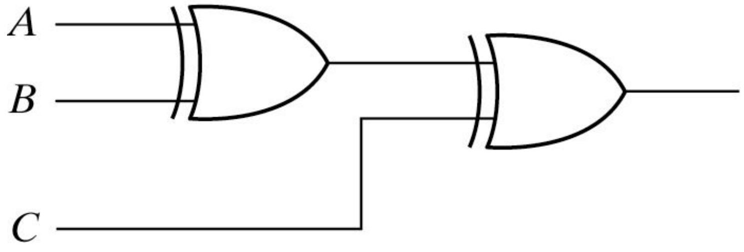


(a) Odd function
$F = A \oplus B \oplus C$

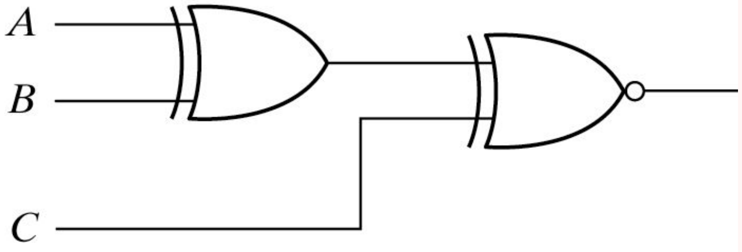


(a) Even function
$F = (A \oplus B \oplus C)'$

- Logic diagram of odd and even functions

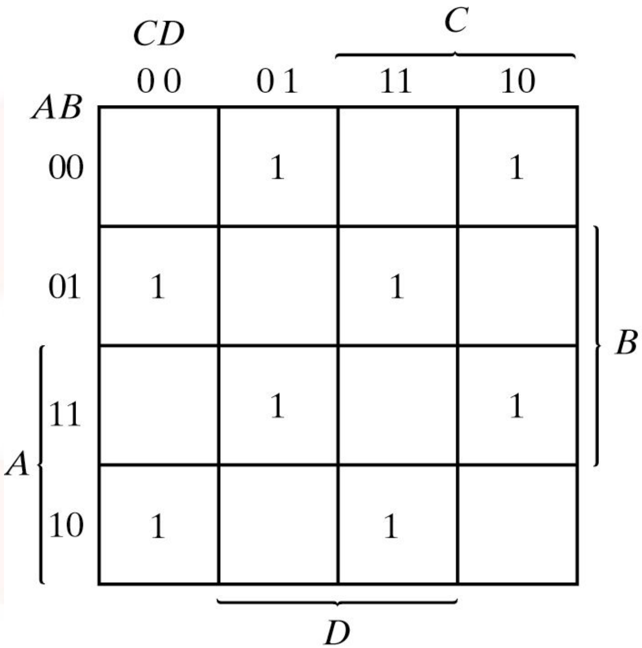


(a) 3-input odd function



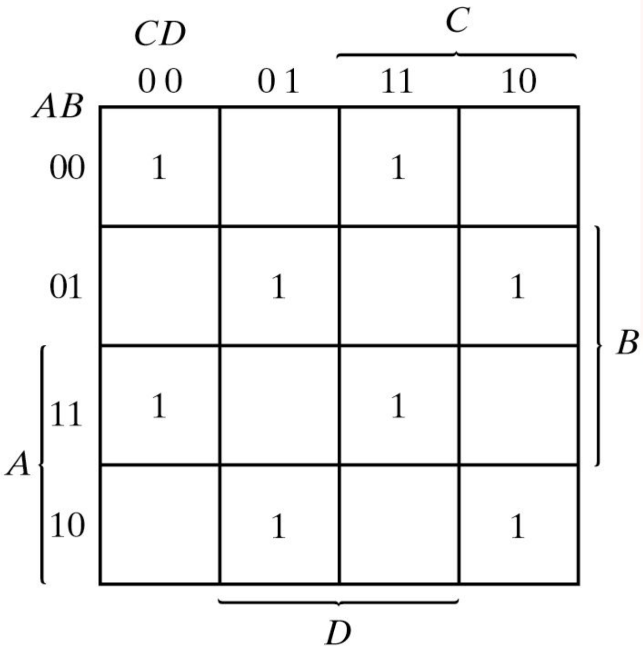
(b) 3-input even function

- Four-variable Exclusive-OR function
  - $A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D) = (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D)$



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

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

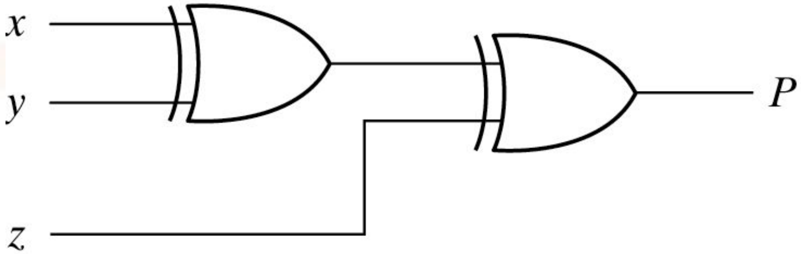


AB \ CD	00	01	11	10
00	1		1	
01		1		1
11	1		1	
10		1		1

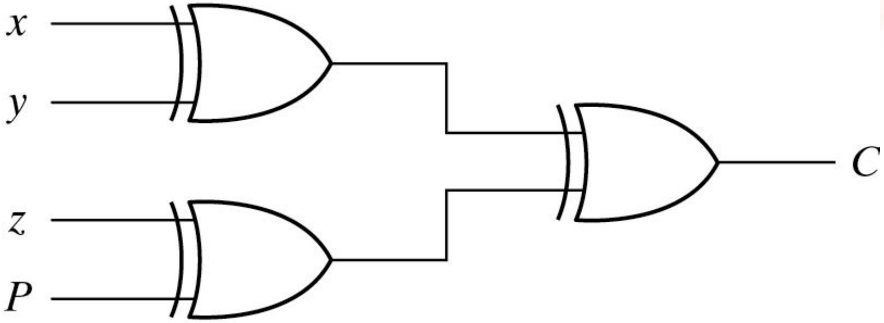
(b) Even function
$F = (A \oplus B \oplus C \oplus D)'$

Parity Generation and Checking

- Parity Generation and Checking
  - a parity bit: P = x⊕y⊕z
  - parity check: C = x⊕y⊕z⊕P
    - C=1: an odd number of data bit error
    - C=0: correct or an even # of data bit error



(a) 3-bit even parity generator



(a) 4-bit even parity checker