Gate Level Minimization

2019. 3. 18.

K-S. Sohn

Contents

- The map method
- Four-variable map
- Product-of-sums simplification
- Don't care conditions
- NAND-NOR Implementation
- Other two-level implementation
- XOR function

Introduction

- Gate-level minimization
  - Gate-level minimization is the design task of finding an optimal gate-level implementation of Boolean functions describing a digital circuit.
- Importance
  - CAD can minimize Boolean equations efficiently and quickly.
  - Nevertheless, designer should understand the underlying mathematical description and solution of the problem

The Map Method

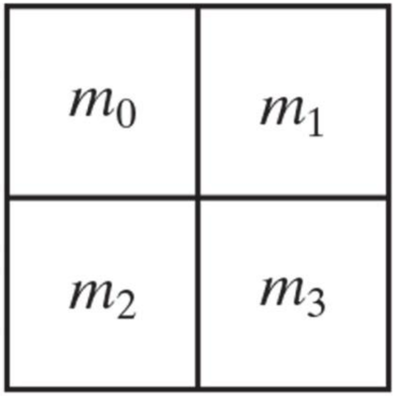
- The complexity of the digital logic gates
  - the complexity of the algebraic expression
- Logic minimization
  - algebraic approaches: lack specific rules
  - the Karnaugh map (or K-map)
    - a simple straight forward procedure
    - a pictorial form of a truth table
    - applicable if the # of variables < 7
- A diagram made up of squares
  - each square represents one minterm

Boolean Function

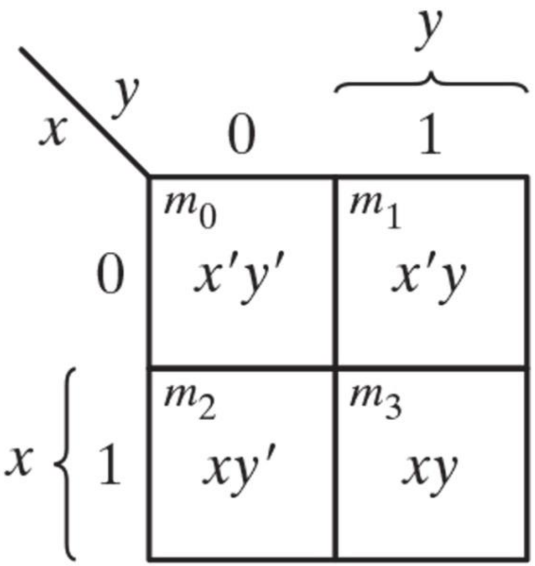
- Standard form of a Boolean function
  - sum of product
  - product of sum
- Simplest algebraic expression
  - a minimum number of terms
  - a minimum number of literals in each term
  - The simplified expression may not be unique
- K-map (Karnaugh map)
  - a diagram made up of squares with each square representing one minterm of the Boolean function to be minimized

Two-Variable K-Map

- A two-variable Boolean function
  - four minterms
- A two-variable map
  - four squares, one for each minterm
  - x' = row 0; x = row 1
  - y' = column 0; y = column 1



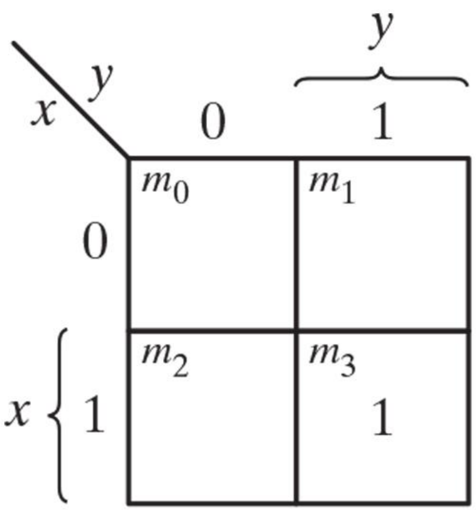
(a)



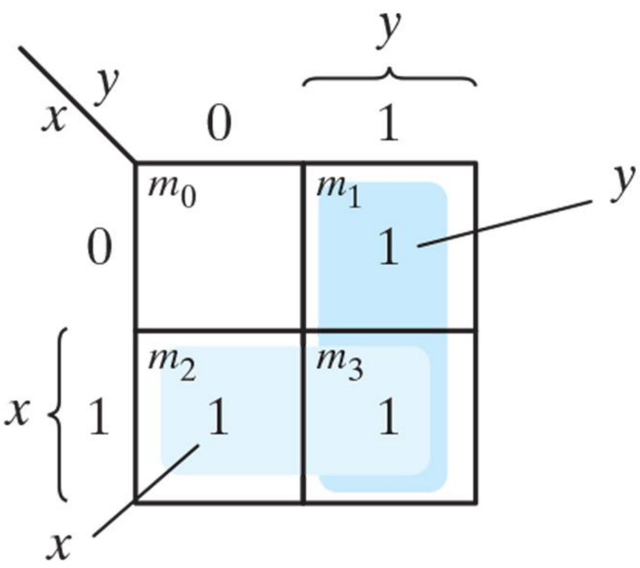
(b)

Two-Variable K-Map

- Representation of functions in the K-map
  - map a row of a truth table into square located by (x,y) in the diagram
  - xy
  - x+y



(a) xy

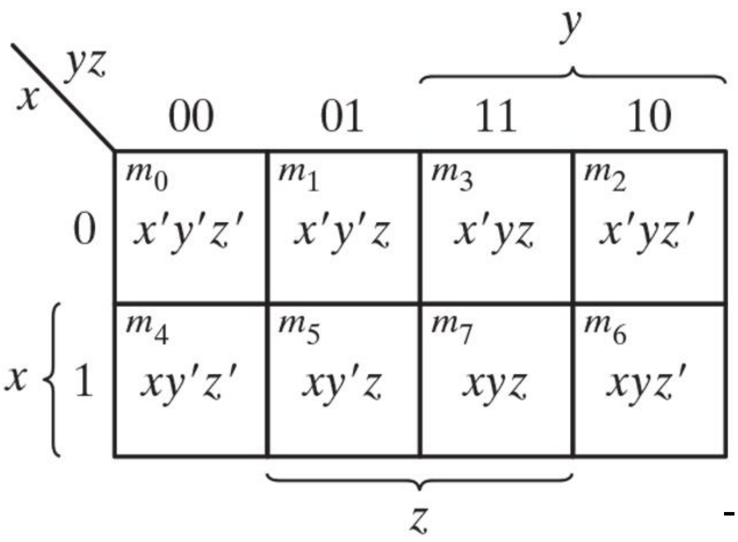


(b) $x + y$

Three-Variable Map

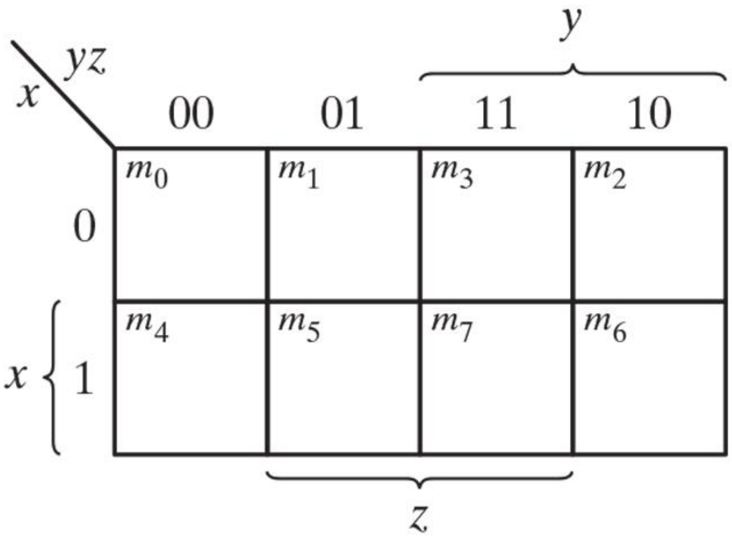
- Three-variable Boolean function
  - eight minterms
- Three-variable K-map
  - minterms arranged in a Gray code sequence
  - any two adjacent squares in the map differ by only one variable
    - primed in one square and unprimed in the adjacent one

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



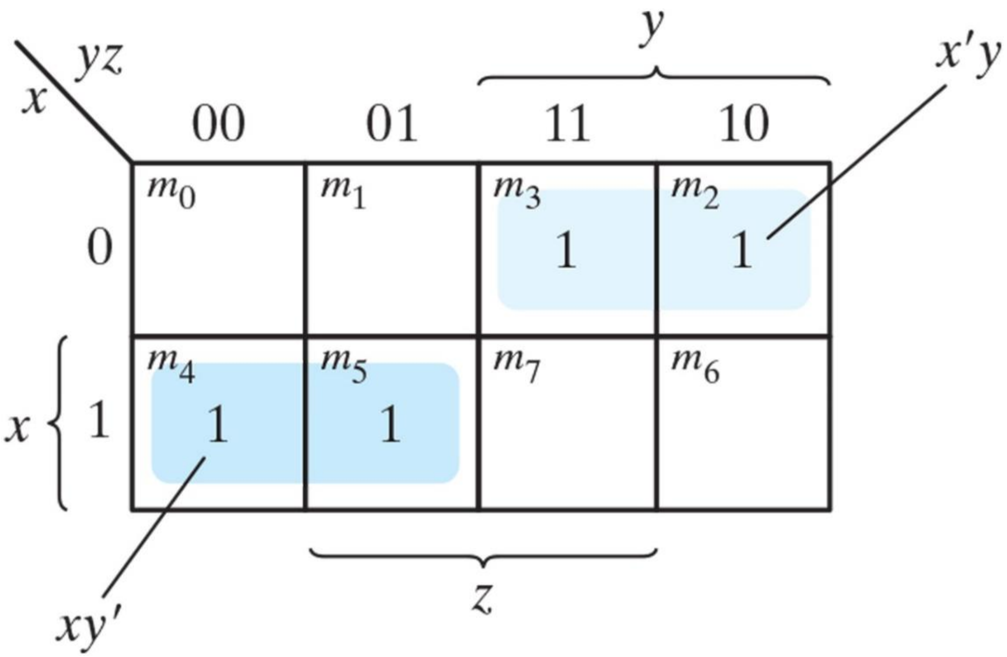
Three-Variable K-Map

- Adjacency of squares
  - in row, in column, and rotation is permitted in each dimension
- Example
  - An adjacent pair m5 and m7 can be simplified
  - m5+ m7 = xy'z + xyz = xz (y'+y) = xz



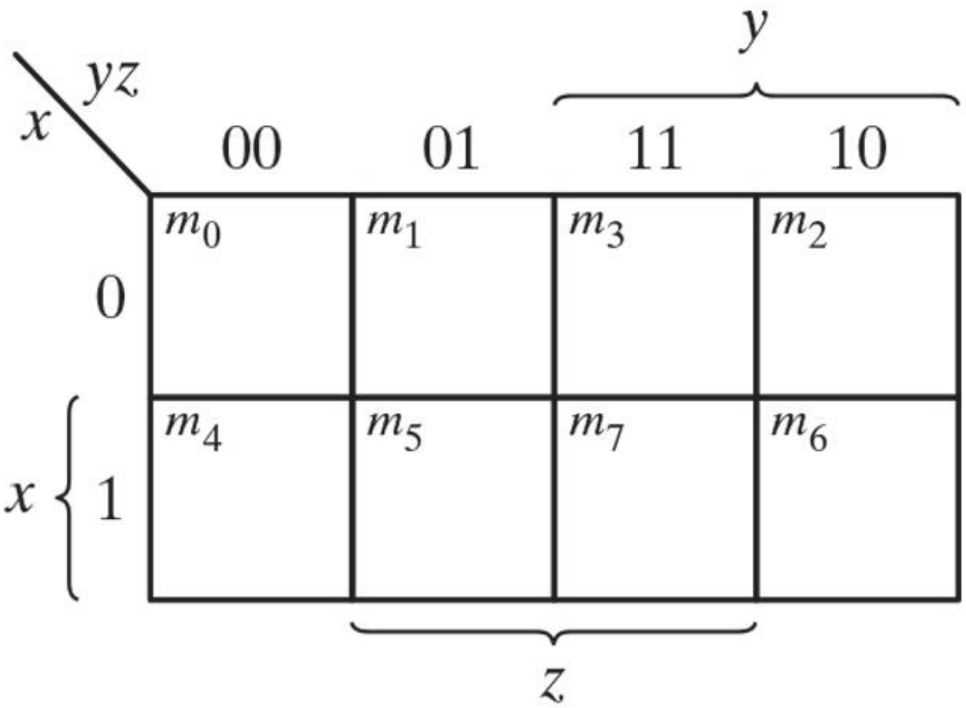
Three-Variable K-Map

- Example 3.1
  - F(x,y,z) = ∑(2,3,4,5) → F = x'y + xy'



Simplifying with Two Adjacent Squares

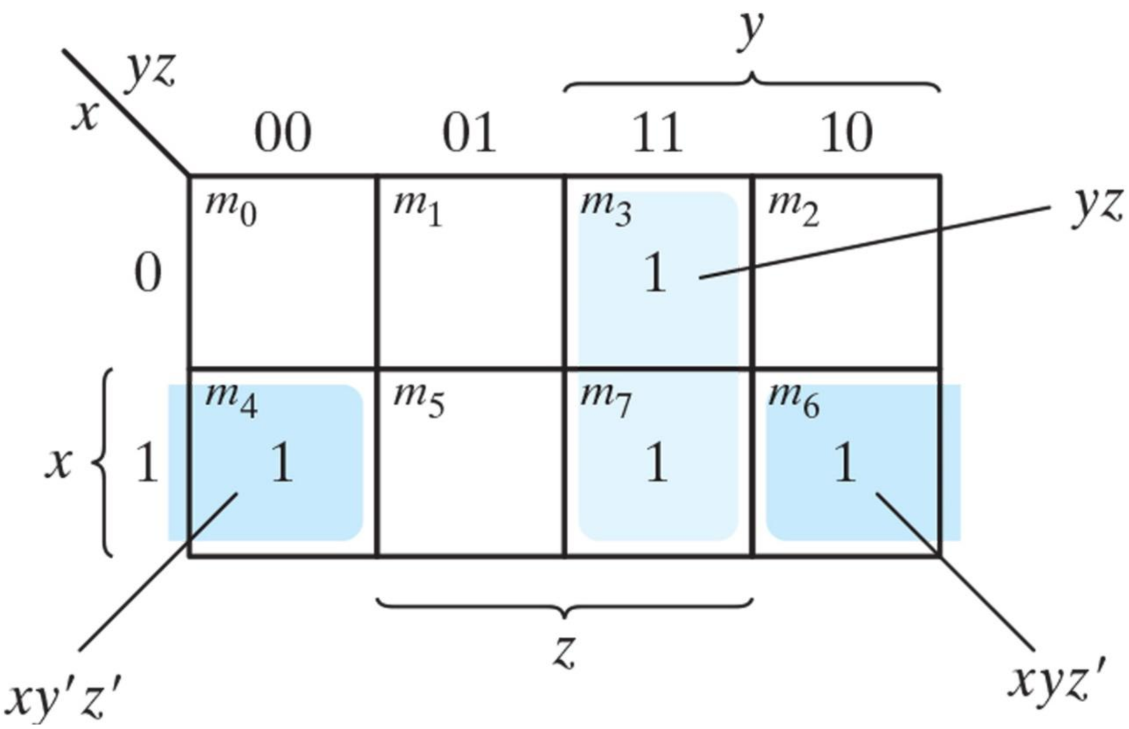
- Example of adjacency
  - m_0 and m_2 (m_4 and m_6) are adjacent
  - $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y'+y) = x'z'$
  - $m_4 + m_6 = xy'z' + xyz' = xz'(y'+y) = xz'$



x \ yz	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

Simplifying with Two Adjacent Squares

- Example 3.2
  - F(x,y,z) = ∑(3,4,6,7) = yz+ xz'



Note: xy'z' + xyz' = xz'

Four Adjacent Squares

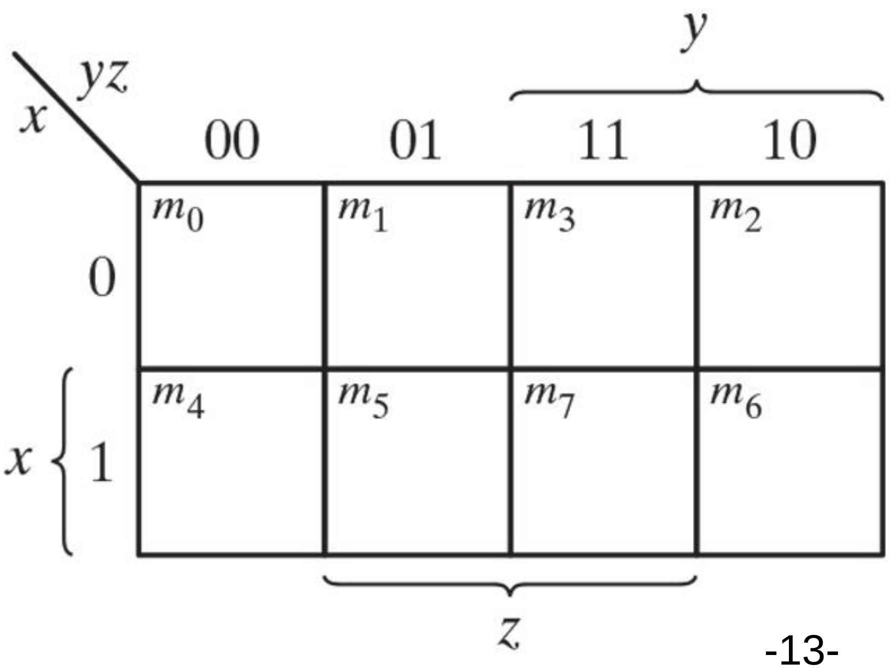
- Simplification in Boolean function
  - F = $m_0+m_2+m_4+m_6$ = x'y'z'+x'yz'+xy'z'+xyz'

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

    = x'z' + xz'= z'
  - F = $m_1+m_3+m_5+m_7$ = x'y'z+x'yz+xy'z+xyz

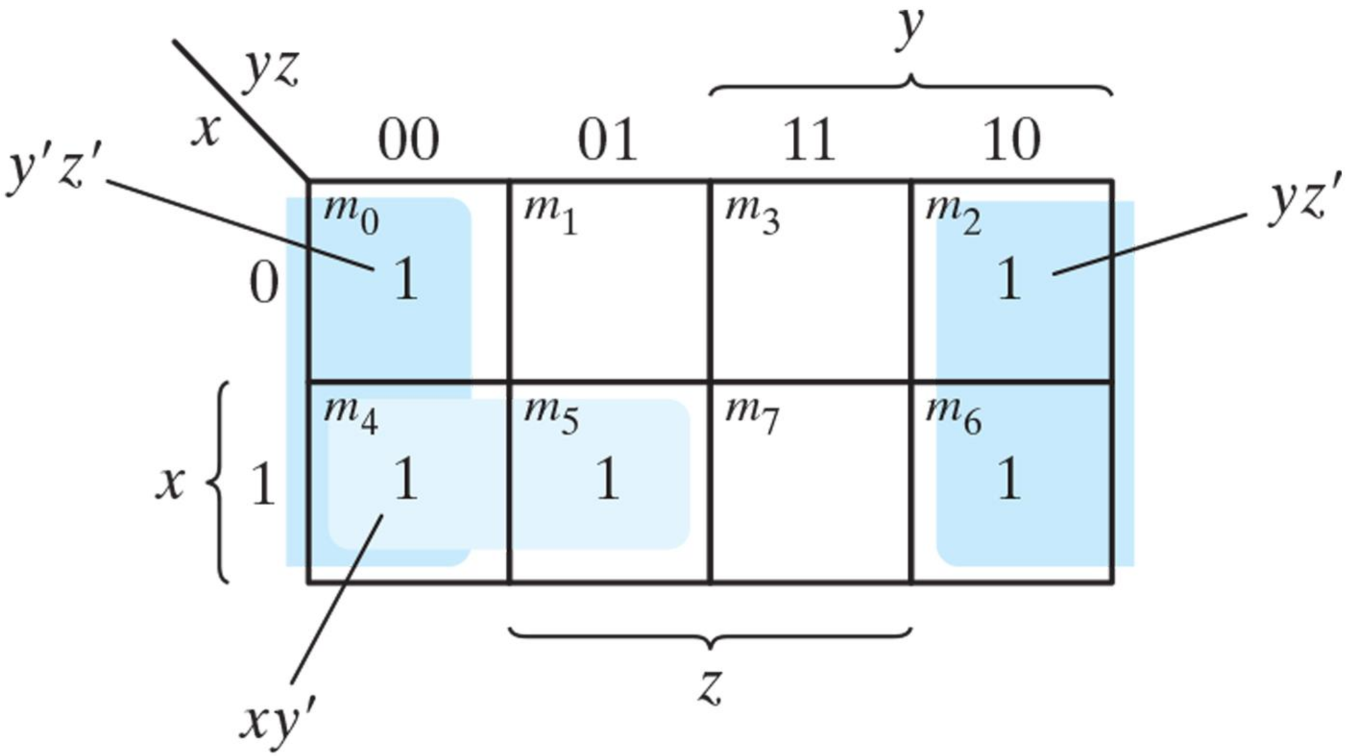
    = x'z(y'+y) + xz(y'+y)

    = x'z + xz = z



Simplifying on Four Adjacent Squares

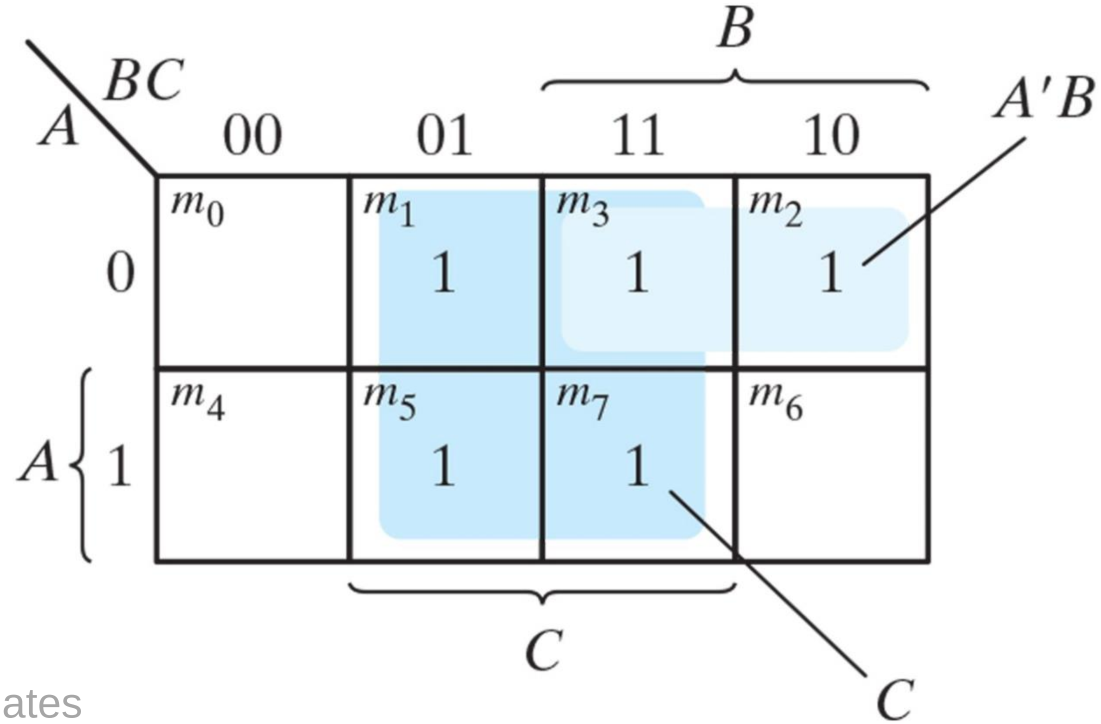
- Example 3.3
  - $F(x,y,z) = \sum (0,2,4,5,6) \rightarrow F = z' + xy'$



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

Simplifying on Four Adjacent Squares

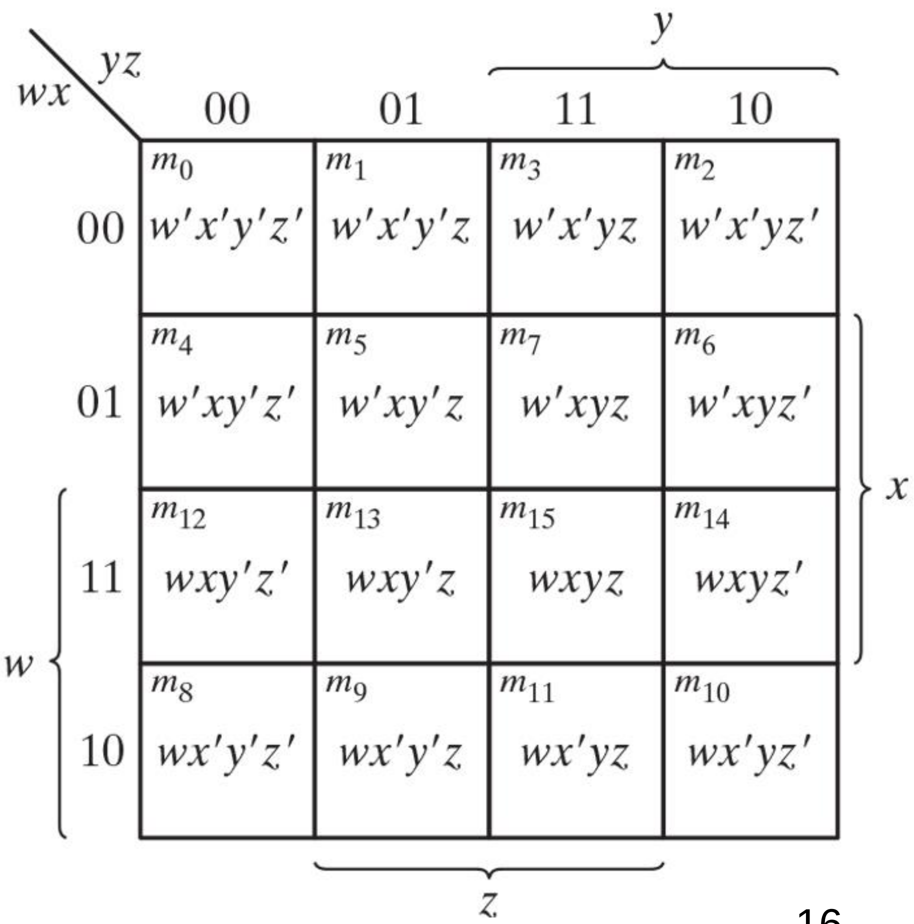
- Example 3.4
  - F = A'C + A'B + AB'C + BC
  - express it in sum of minterms
  - find the minimal sum of products expression



Four-Variable K-Map

- Four-variable K-map diagram
  - 16 minterms

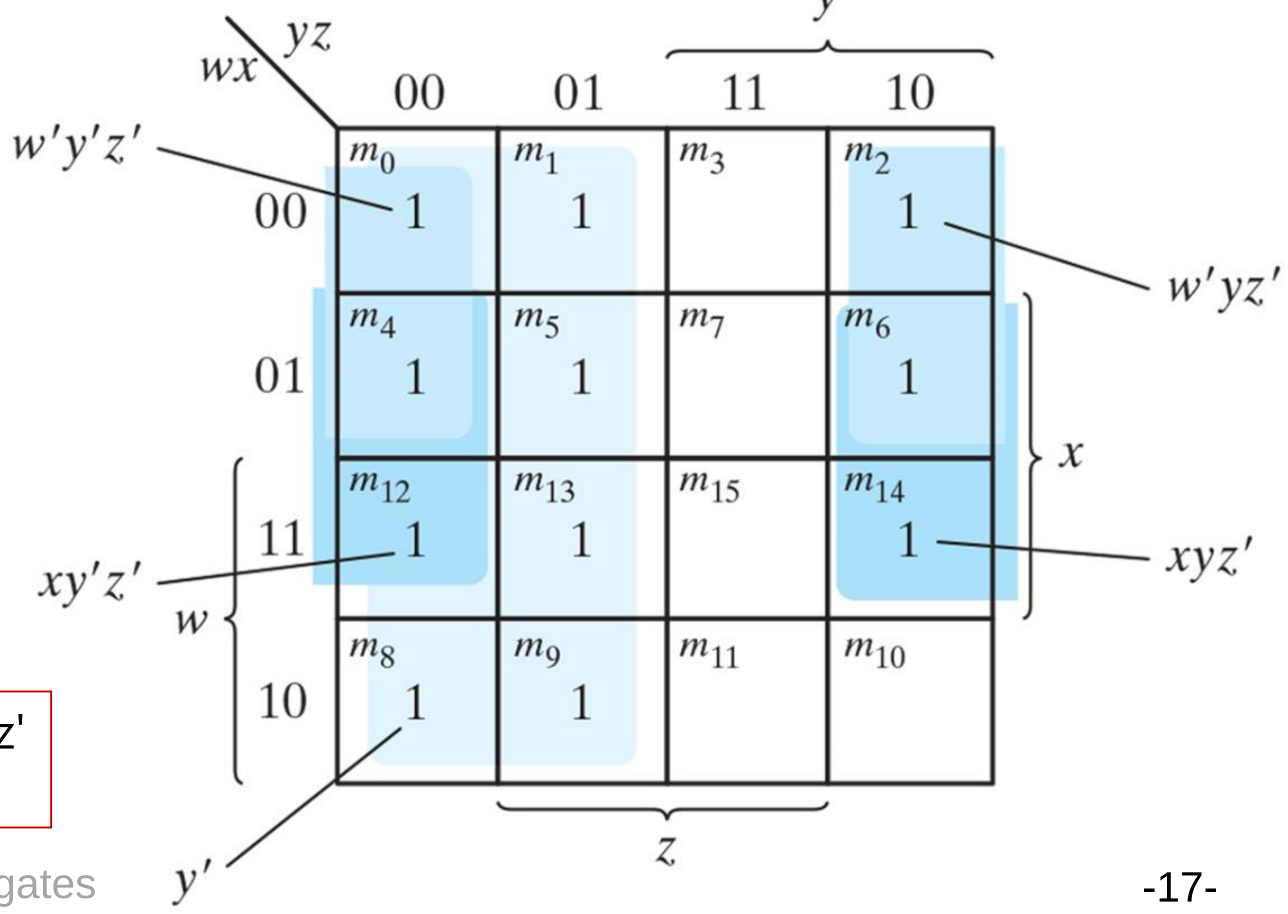
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



$wx \backslash yz$	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} $wx'yz'$

Use 16, 8, 4, and 2 Adjacent Squares A. P.

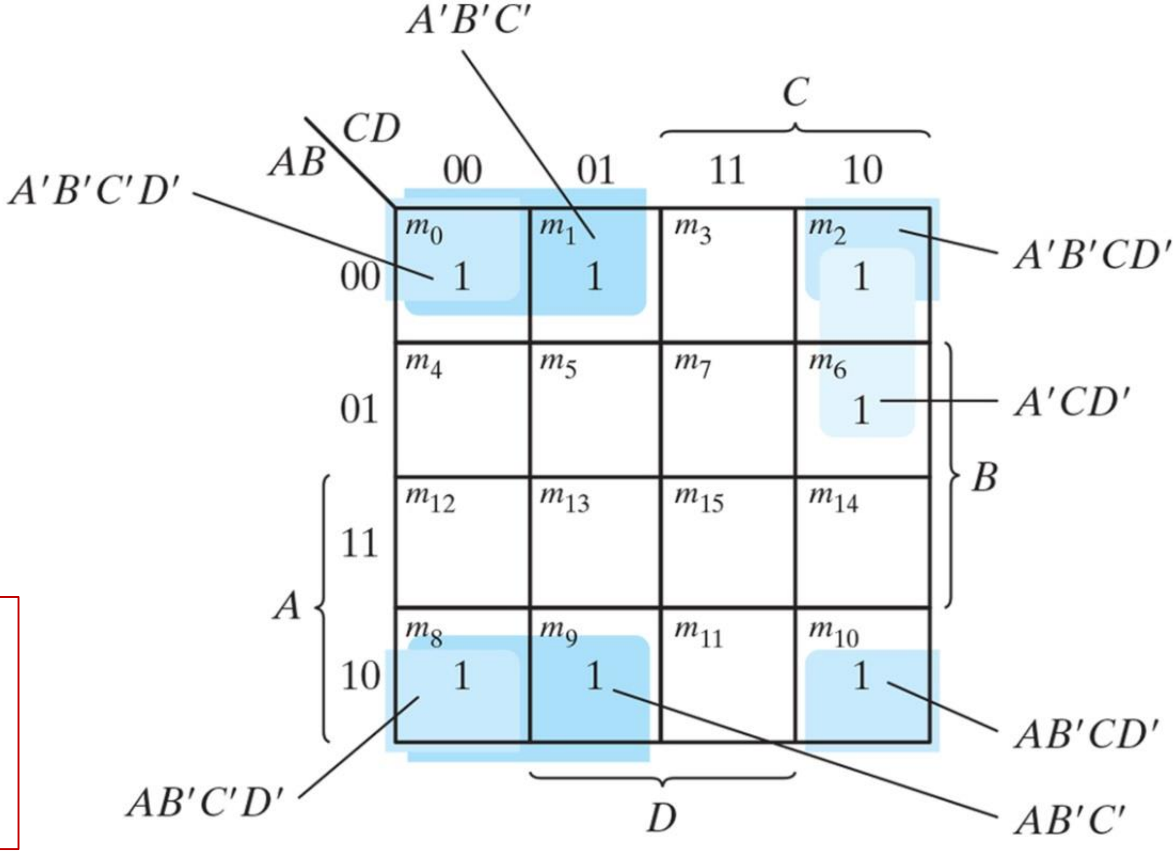
- Example 3.5
  - F(w,x,y,z) = ∑ (0,1,2,4,5,6,8,9,12,13,14) → F = y'+w'z'+xz'



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

Use 16, 8, 4, and 2 Adjacent Squares A. P.

- Example 3.6
  - F = A'B'C'+ B'CD'+ A'BCD'+ AB' → B'D' + B'C' + A'CD'



Notes:
A;B'C'D' + A'B'CD' = A'B'D'
AB'C'D' + AB'CD' = AB'D'
A'B'C' + AB'C' = B'C'

Simplification Procedure

- Simplification criteria
  - all the minterms of the function are covered by terms
  - the number of terms in the expression is minimized
  - there are no redundant terms(i.e., minterms already covered by other terms)
- Systematic choice of adjacent squares
  - Find **essential prime implicants** first, and then find **prime implicants** those which cover the given Boolean function.

Prime Implicants

- Implicant
  - a product term which, when true(i.e., 1), always implies that the given Boolean function is true(i.e., 1).
  - F(w,x,y,z) = wx + xy + z → implicants of F: wx, xy, z, wxy, zx, ...
  - then how about w, x, y, or wy?
- Prime implicant
  - a P.I. is a product term obtained from the map by combining all possible maximum numbers of squares
  - a single 1's represents a P.I. if it is not adjacent to any other 1's.
  - Two adjacent 1's form a P.I if they are not within a group of four adjacent 1's. Four adjacent 1's form a P.I if they are not within a group of eight adjacent 1's, and so on.

Prime Implicants

- Essential prime implicant
  - prime implicants that cover an output of the function that no combination of other prime implicants is able to cover
  - one or more minterms in a square are covered by only that one prime implicant
  - the essential P.I. must be included in the minimized expression

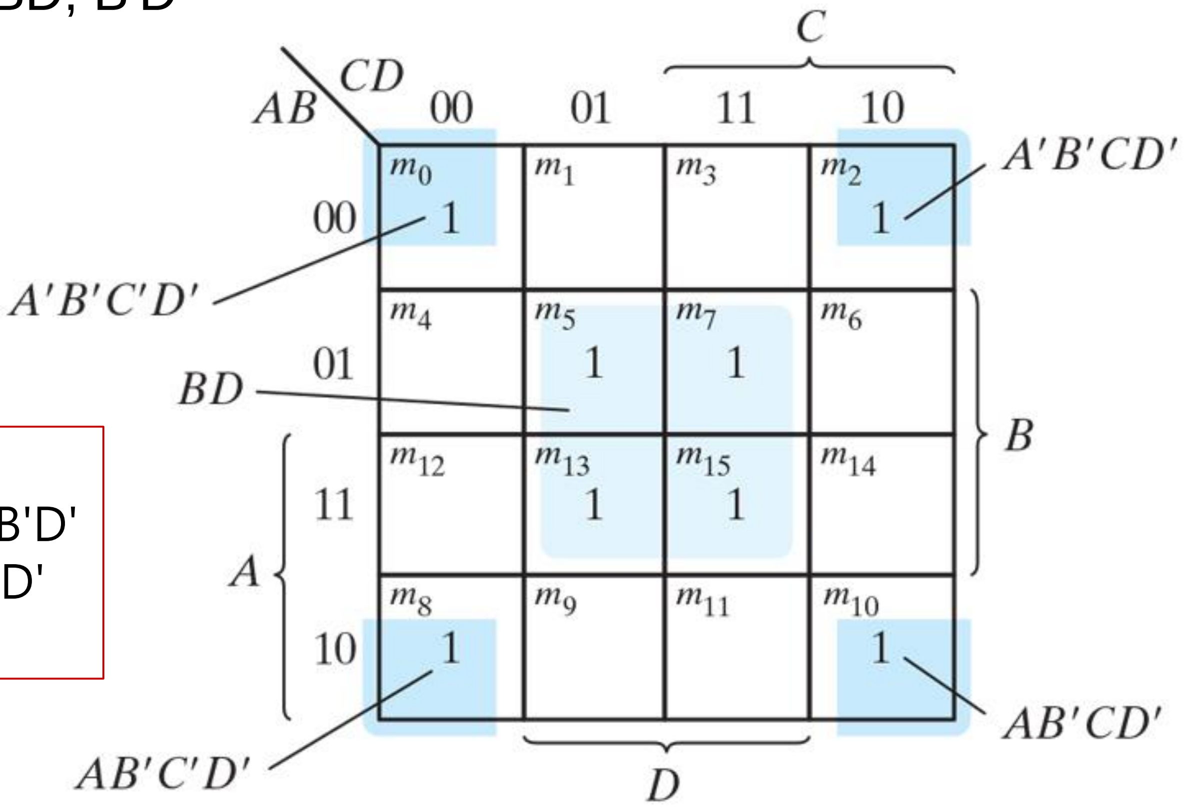
Prime Implicants

- Example: considering essential P.I. and P.I.
  - F(A, B, C, D) = ∑ (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)

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

Prime Implicants

- Example: considering essential P.I. and P.I.
  - F(A, B, C, D) = ∑ (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)
  - essential P.I. : BD; B'D'



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'

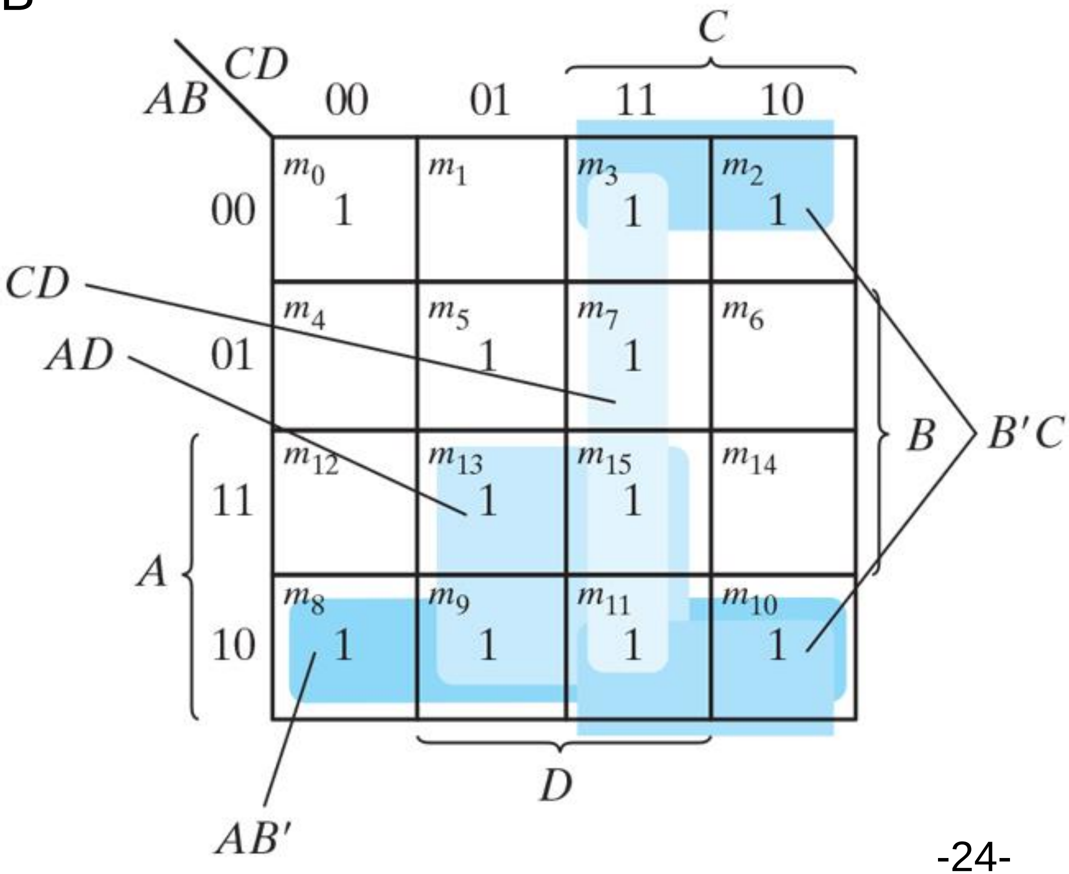
Prime Implicants

- Example: considering essential P.I. and P.I.
  - F(A, B, C, D) = ∑ (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)
  - P.I. : CD, B'C, AD, and AB'
  - F = BD+B'D'+CD+AD

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

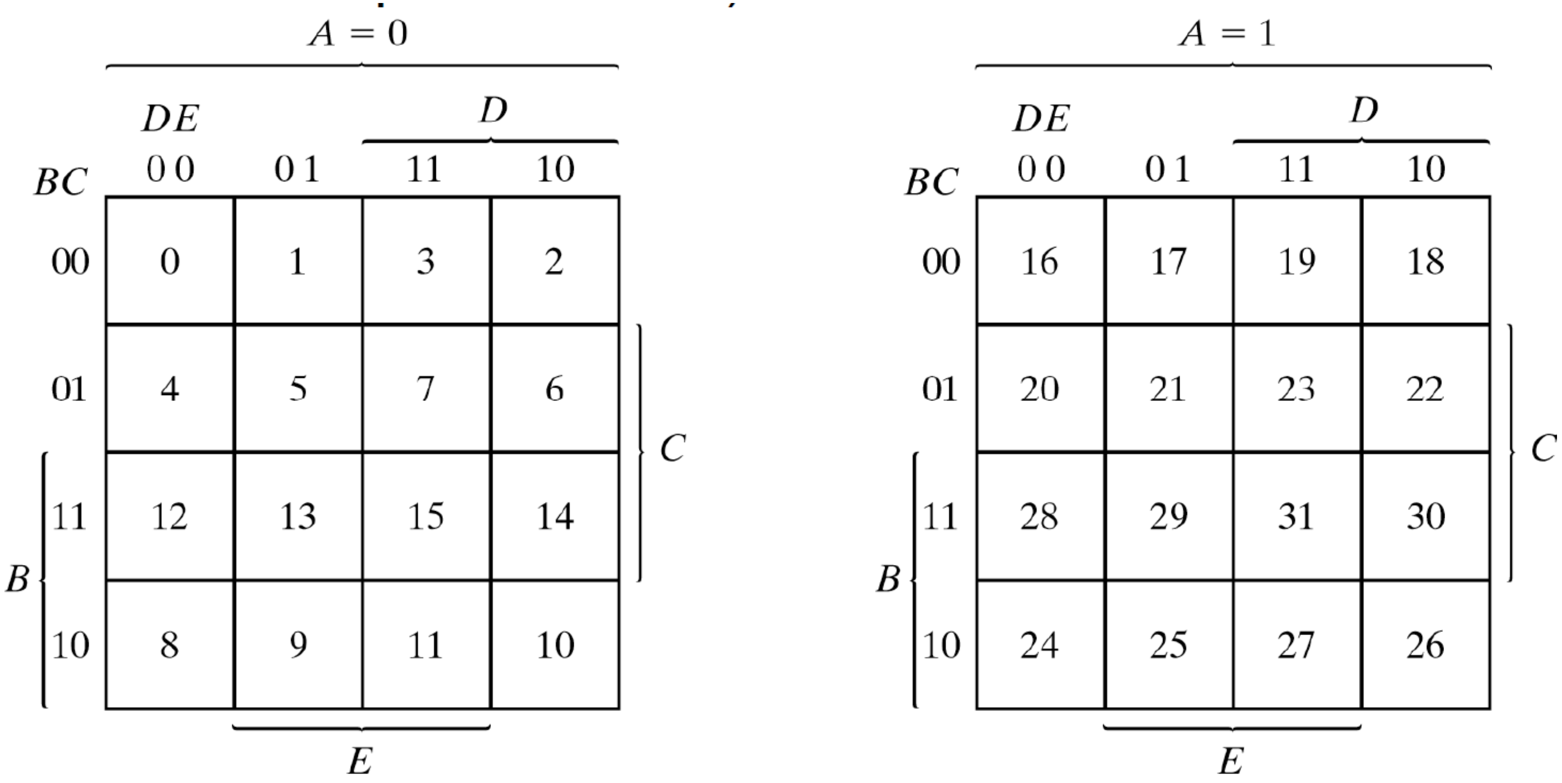
    = BD+B'D'+B'C+AD

    = BD+B'D'+B'C+AB'



Five-Variable K-Map

- Divide and re-combine the map
  - map for more than four variable becomes complicated
  - five-variable map ⇒ divide ⇒ two four-variable map



$A = 0$

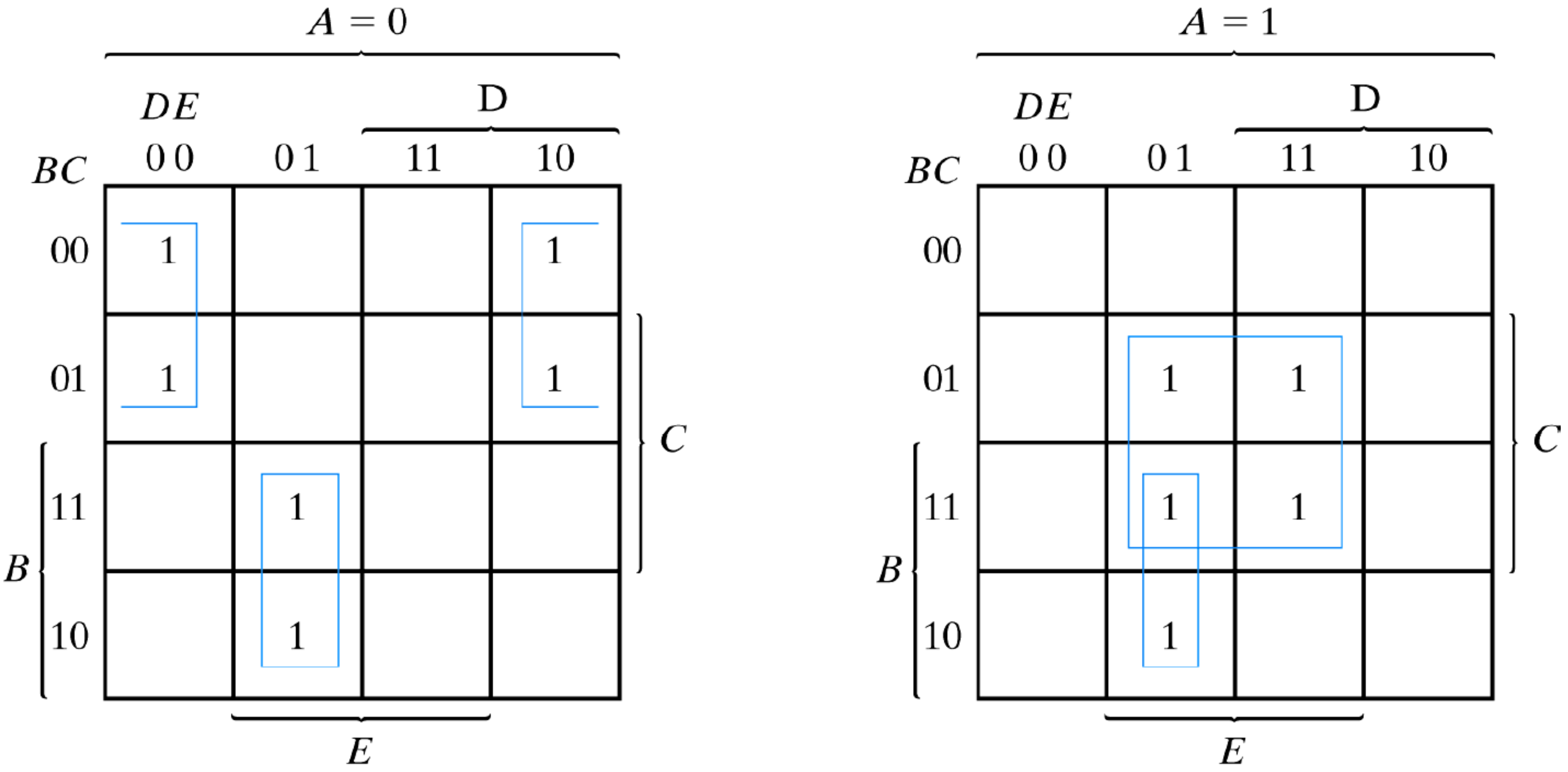
BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$A = 1$

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

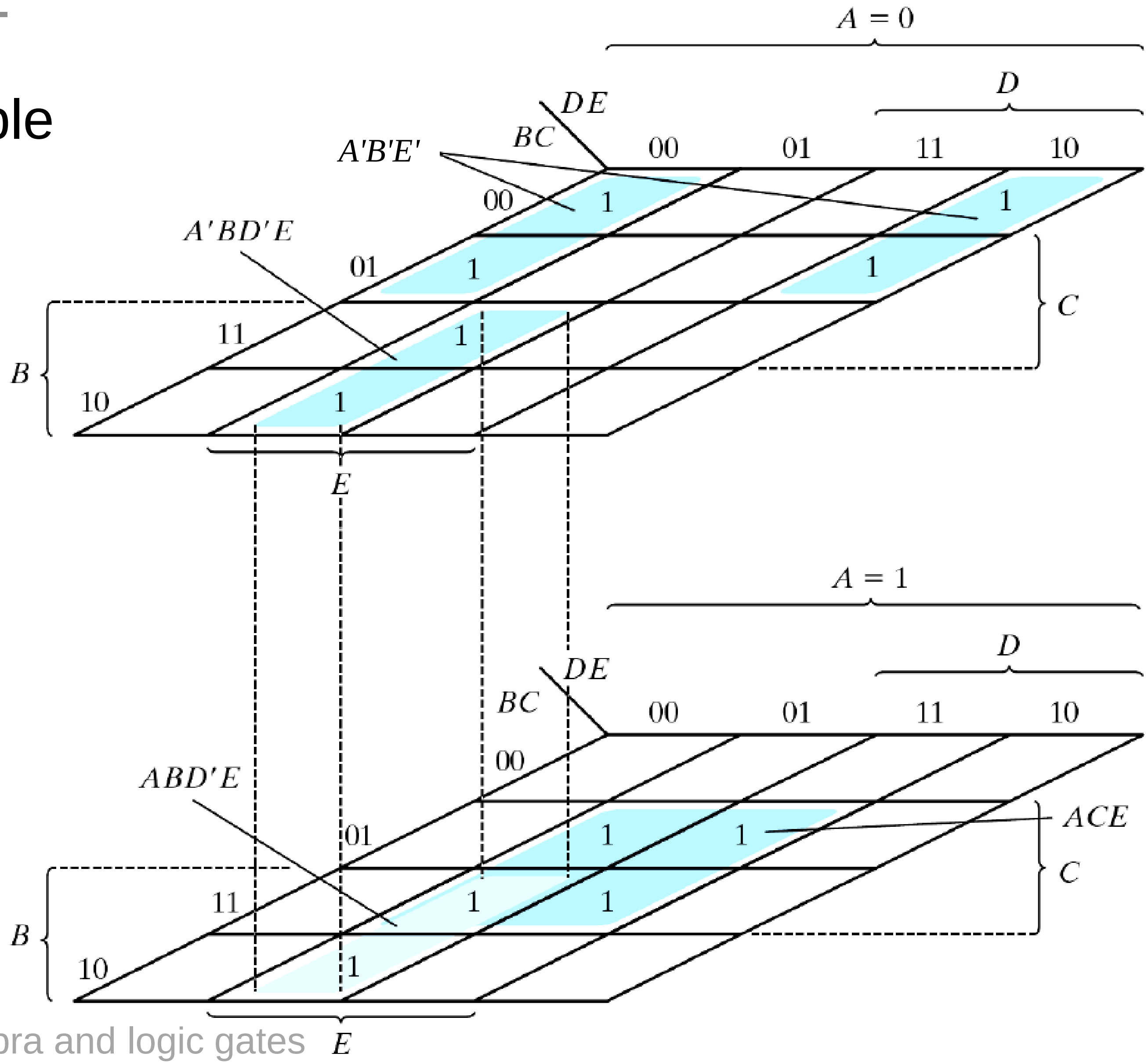
Five-Variable K-Map

- Example
  - F = ∑ (0,2,4,6,9,13,21,23,25,29,31) = A'B'E' + BD'E + ACE



Five-Variable K-Map

- Example



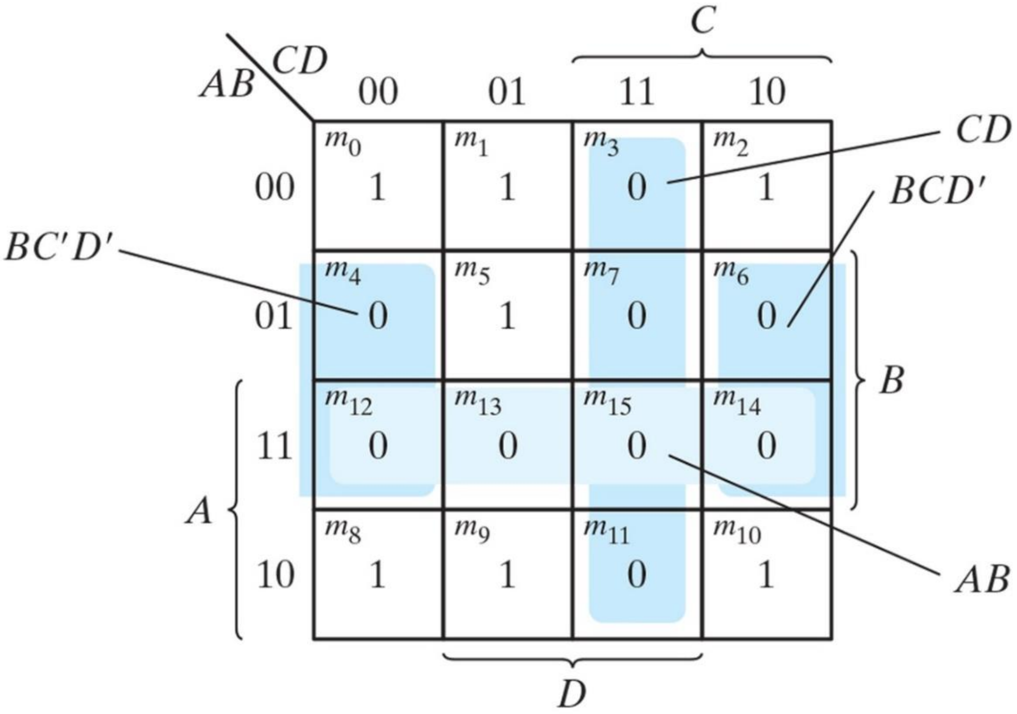
Product of Sums Simplification

- Backgrounds
  - The minimized Boolean function expressed in the product-of-sums can be derived from the map.
  - The minterms not included in the standard sum-of-product form of F denote F'.
- Procedure
  - mark squares by 1's for F on the K-map.
  - mark the remaining squares by 0's for F'
  - obtain minimized F' with combining adjacent 0's-squares in sum-of-product form
  - obtain F in product-of-sum from F' by using such as DeMorgan theorem

Product of Sums Simplification

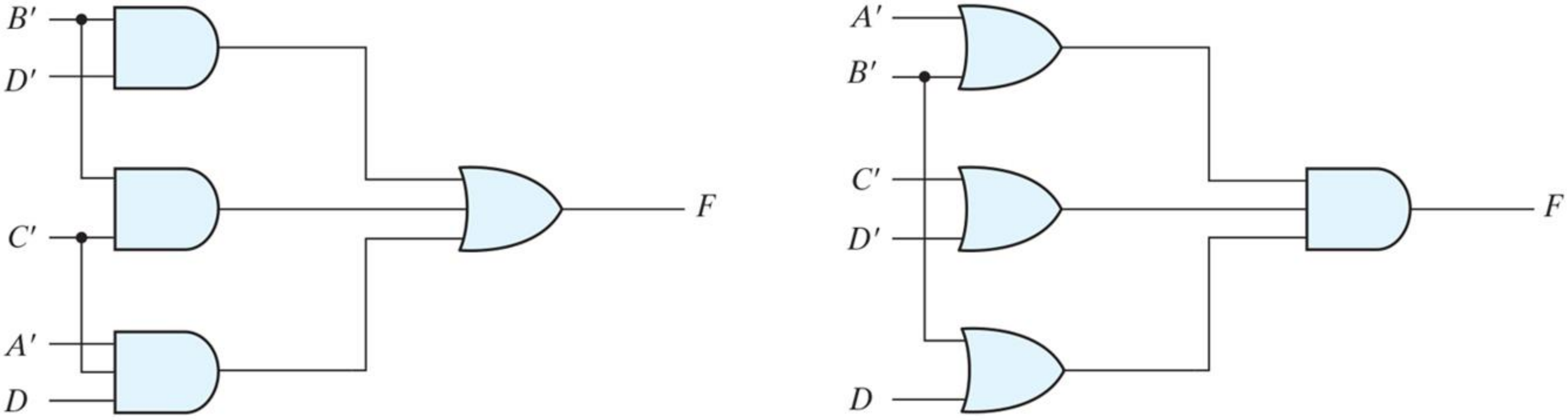
- Example 3.7
  - F(A, B, C, D) = ∑(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D

    = (A' + B')(C' + D')(B' + D).



Product of Sums Simplification

- Example 3.7
  - Gate implementation of the function F



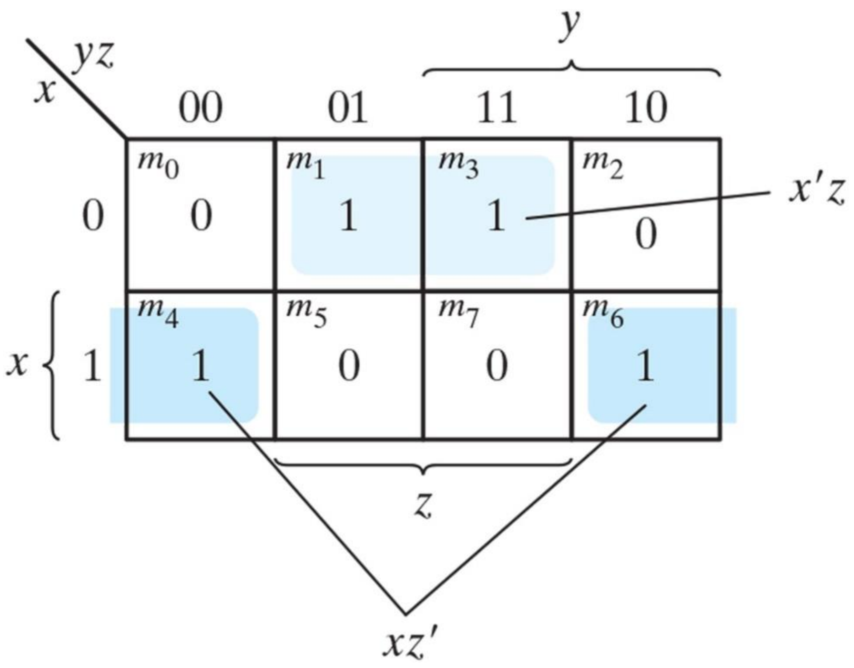
(a) $F = B'D' + B'C' + A'C'D$

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

Product of Sums Simplification

- Given F as product-maxterms (canonical form)
  - an example

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



  - $F = \sum(1,3,4,6)$
  - $F' = \sum(0,2,5,7)$
  - $F = \prod(0,2,5,7)$

$F = x'z + xz'$

$F' = xz + x'z'$

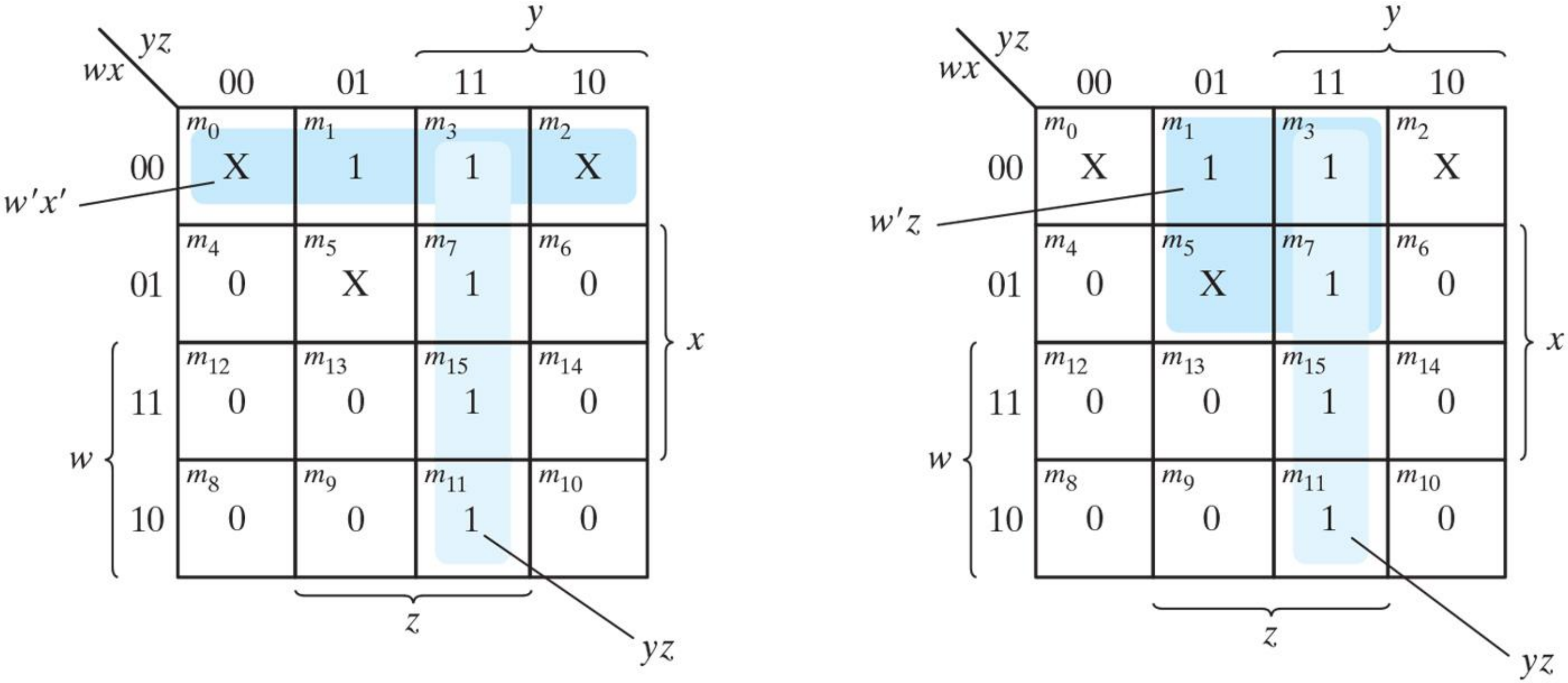
$F = (x'+z')(x+z)$

Don't-Care Conditions

- Don't care condition
  - If we can don't care what value is assumed by the function for an unspecified minterms, we call them as "don't-care conditions."
- Don't-care minterm
  - a don't-care minterm is a combination of variables whose logical value is not specified.
  - the square of the don't-care minterm is marked as 'X' in the K-map.

Don't-Care Conditions

- Example 3.8
  - F (w,x,y,z) = ∑(1,3,7,11,15)
  - d(w,x,y,z) = ∑(0,2,5)



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

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

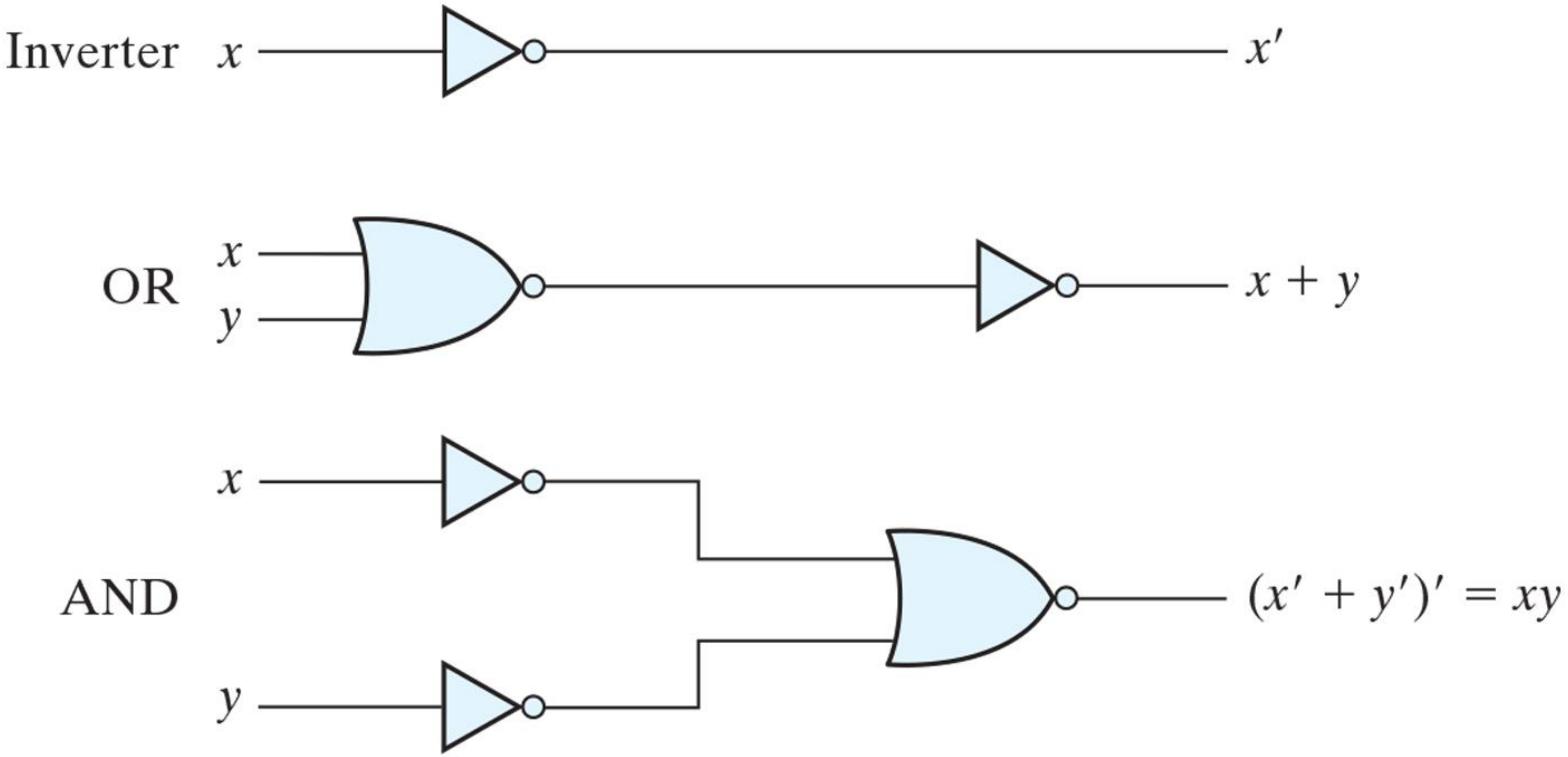
Don't-Care Conditions

- Example 3.8

wxyz	$w'x' + yz$	$w'z + yz$
0000	1	0
0001	1	1
0010	1	0
0011	1	1
0100	0	0
0101	0	1
0110	0	0
0111	1	1
1000	0	0
1001	0	0
1010	0	0
1011	1	1
1100	0	0
1101	0	0
1110	0	0
1111	1	1

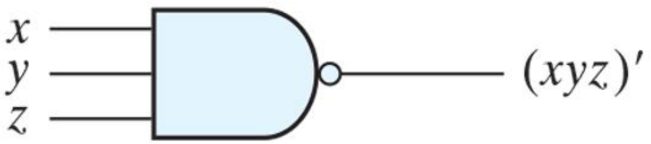
NAND nad NOR Implementation

- NAND
  - A universal gate ⇒ any logic circuit can be implemented

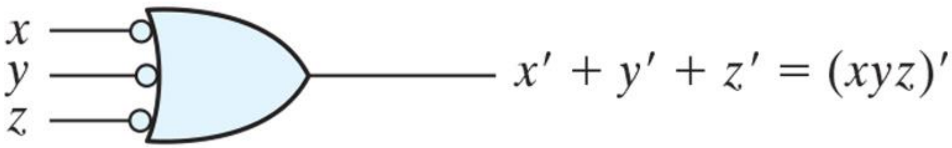


Graphic Symbols for a NAND Gate

- Two graphic symbols for a three input NAND gate
  - AND-invert

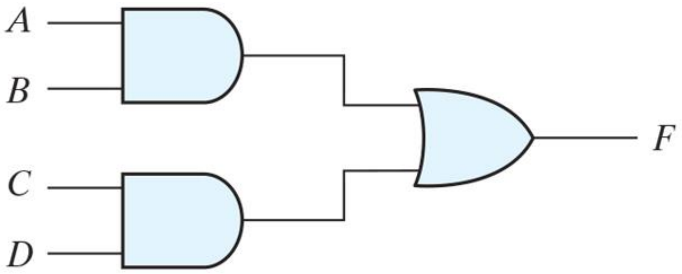


  - invert-OR



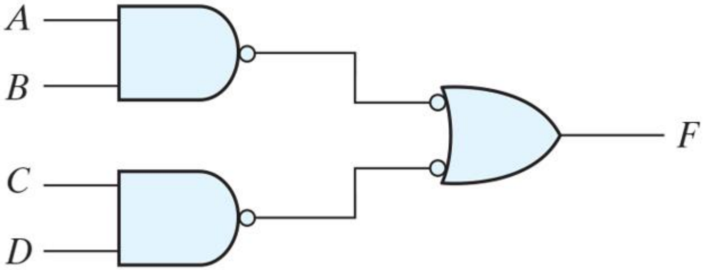
NAND-NAND Two-level Implementation

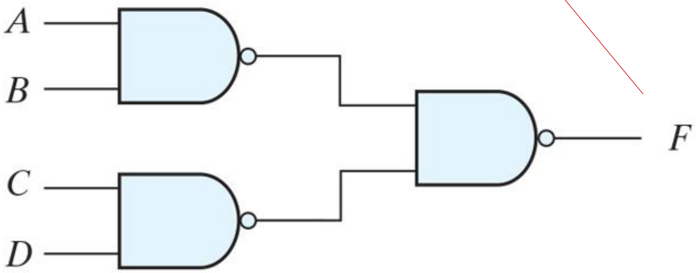
- Original function
  - sum of products
- Example: F = AB + CD
  - AND-OR



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

  - NAND-invert-OR ⇒ NAND-NAND

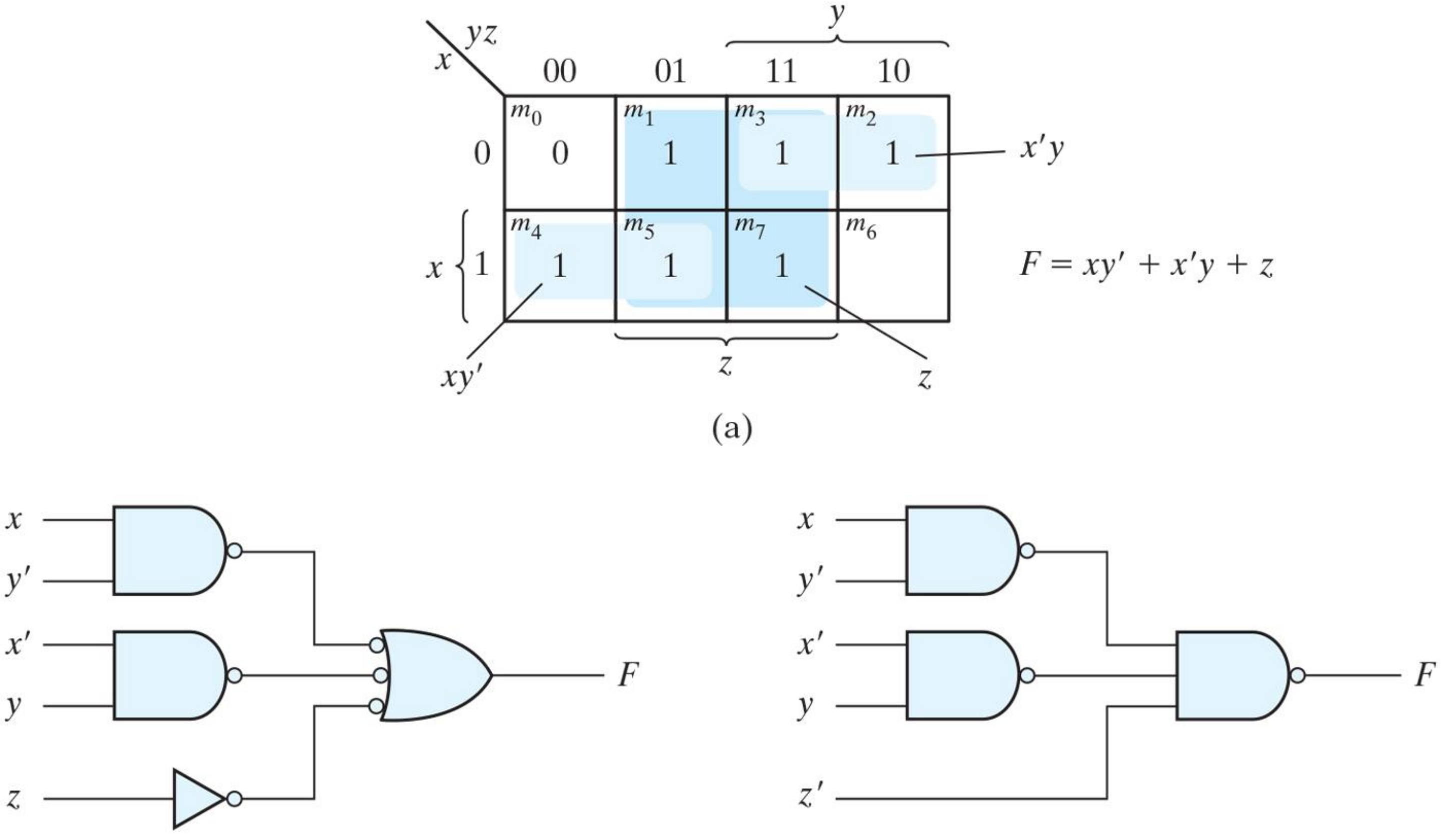




NAND-NAND Implementation

- Example 3.9
  - Implement F with NAND gates
  - F(x,y,z) = (1,2,3,4,5,7)
- 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
  - a single NAND gate for the second sum term
  - a single literal requires an inverter in the first level

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



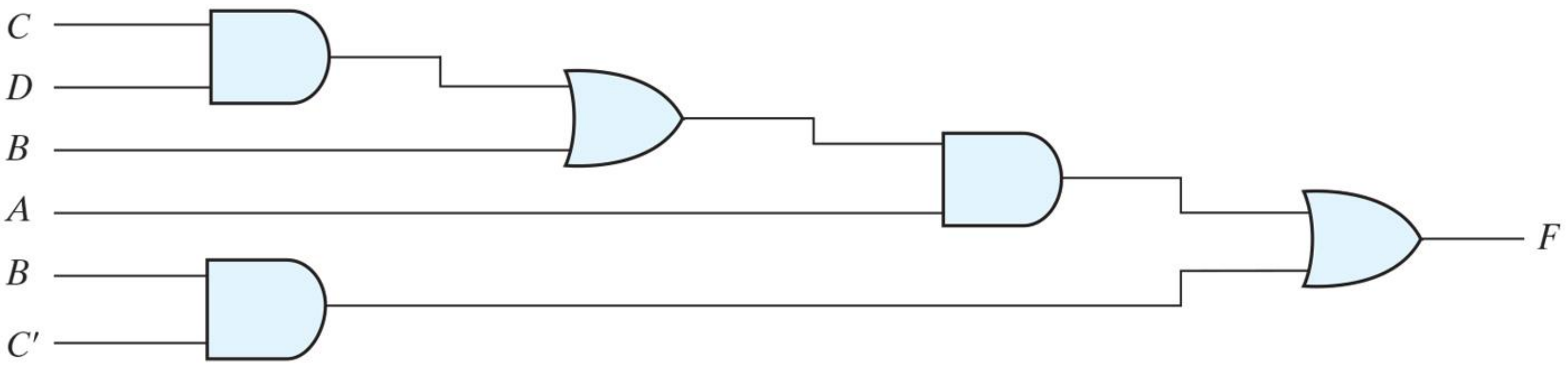
(a)

(b)

(c)

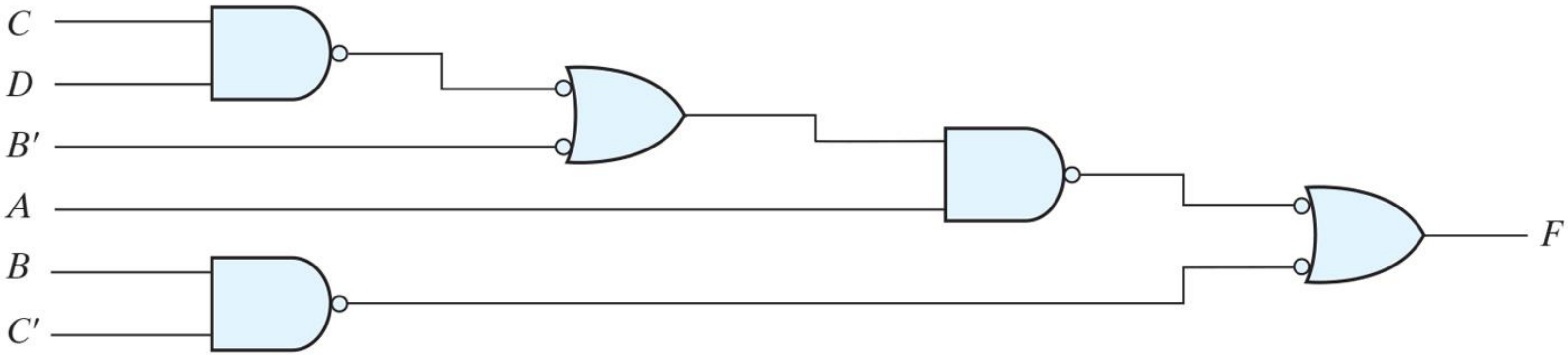
Multilevel NAND Circuits

- Common procedure
  - Implement circuit in terms of AND, OR, and NOT gates
  - convert into an all-NAND circuit
- F = A(CD + B) + BC'
  - omit simplification for illustration
  - AND-OR gates

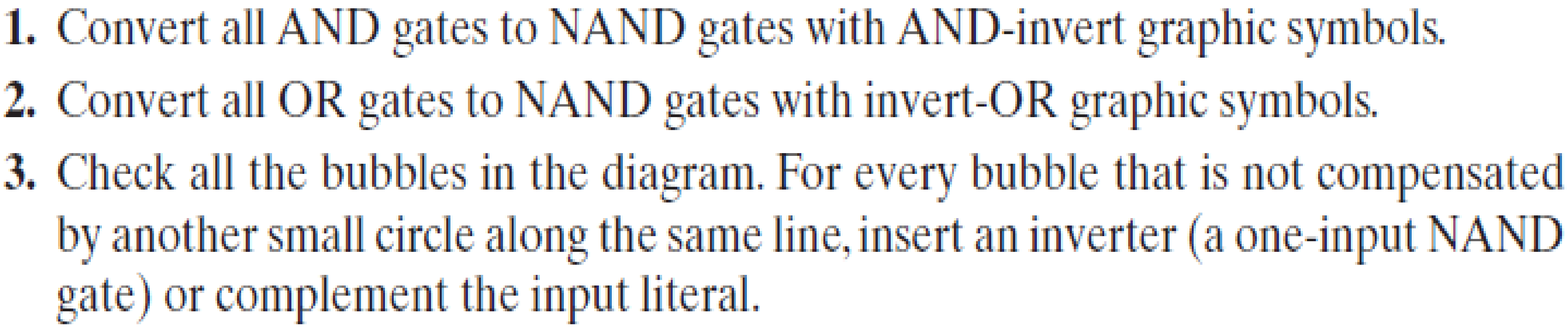


Multilevel NAND Circuits

- F = A(CD + B) + BC'
  - NAND gates



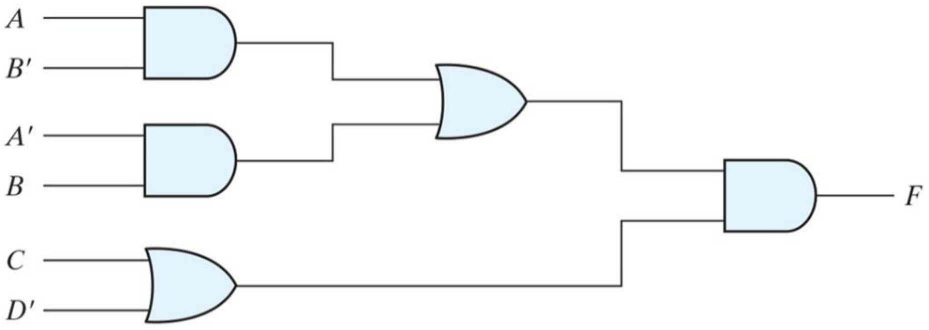
- Procedure of AND-OR to all-NAND



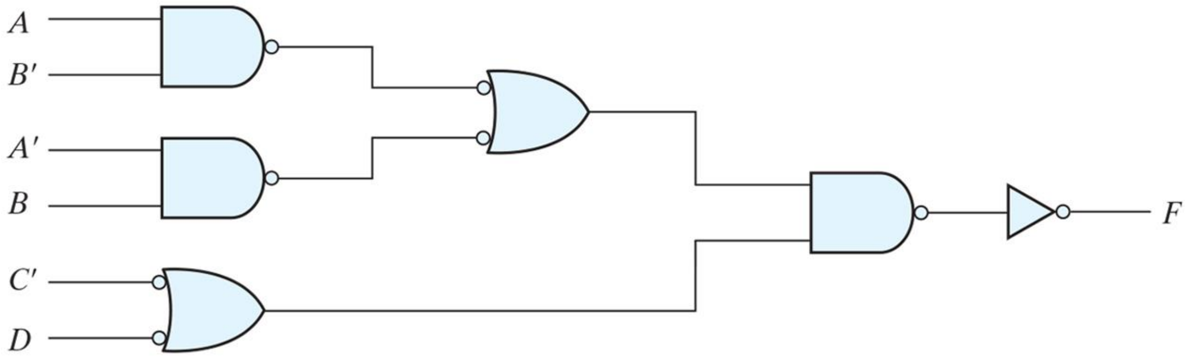
1. Convert all AND gates to NAND gates with AND-invert graphic symbols.
2. Convert all OR gates to NAND gates with invert-OR graphic symbols.
3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.

An Example for The Conversion Procedure

- F = (AB' + A'B)(C + D')
  - Implement AND-OR circuit directly

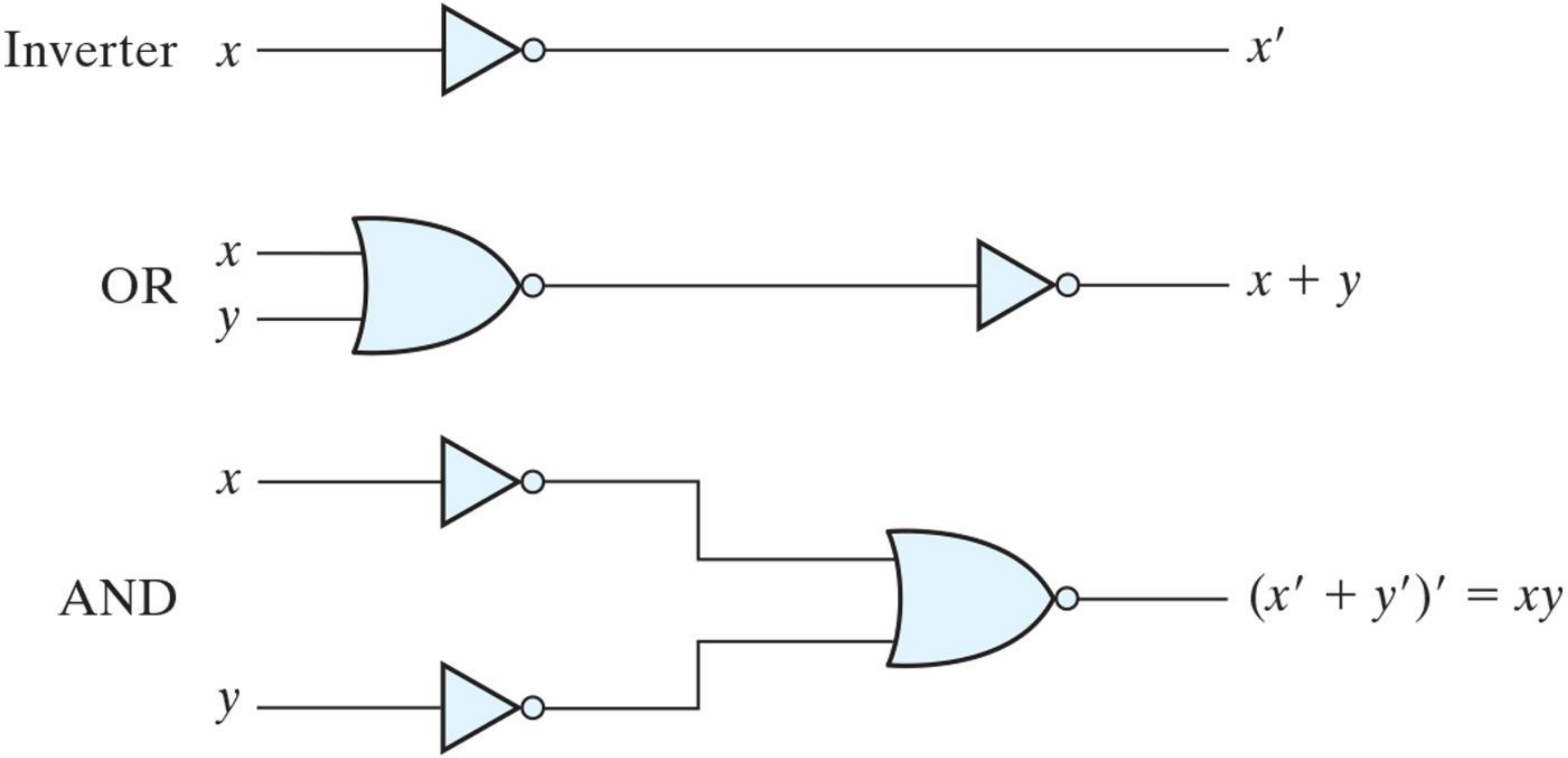


  - Then convert it to all-NAND circuit along the procedure



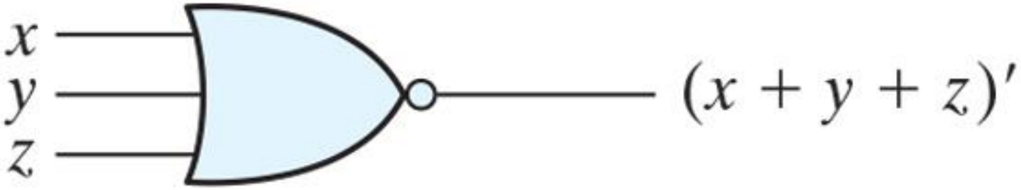
NOR Implementation

- NOR
  - the dual of the NAND operation
  - the NOR gate is also universal

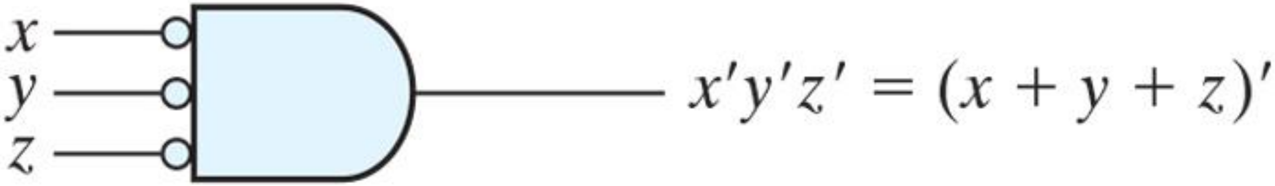


Graphic Symbols for a NOR Gate

- Two graphic symbols for a three input NOR gate
  - OR-invert (c.f. AND-invert)

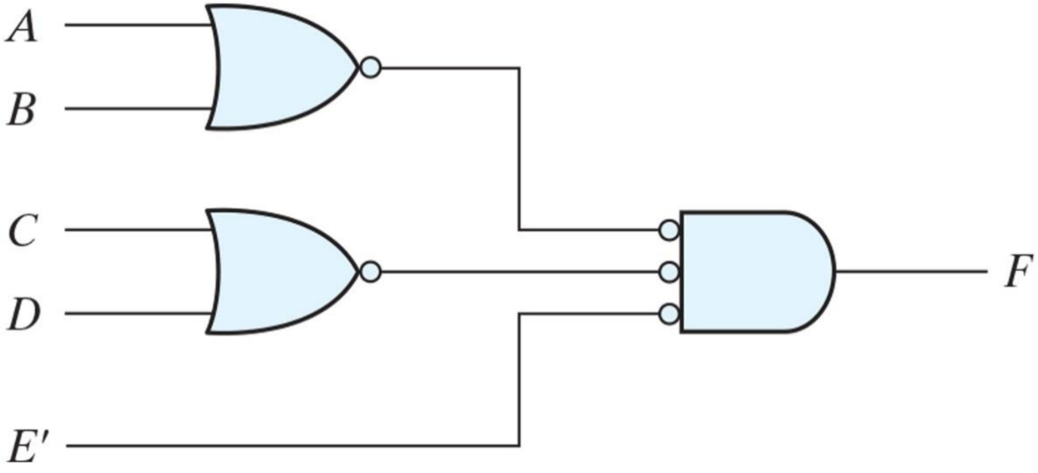


  - invert-AND (c.f. invert-OR)



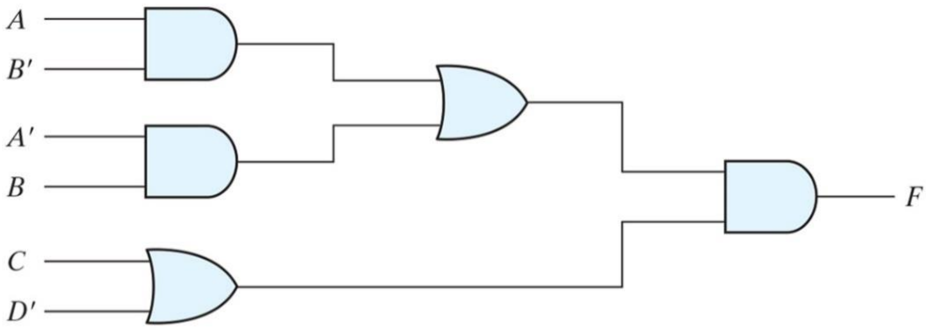
Example of Two-level NOR Circuit

- $F = (A + B)(C + D)E$
  - OR-AND
  - NOR-inverted-AND ⇒ NOR-NOR

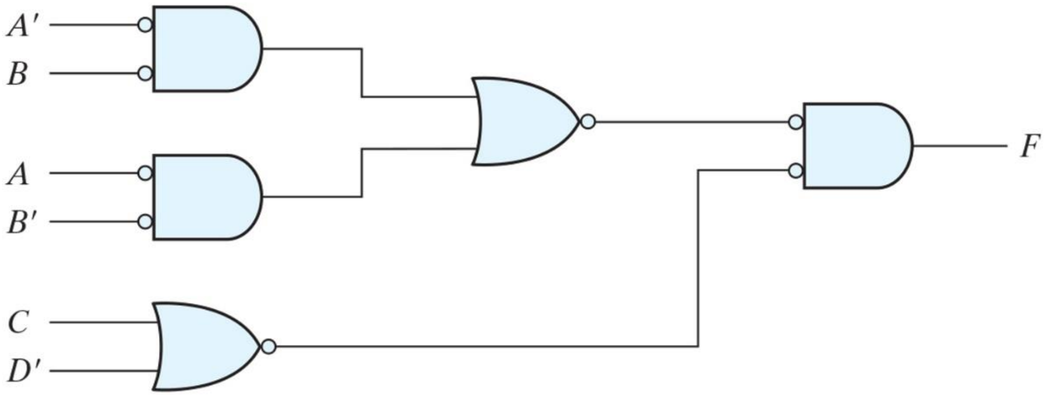


Example of Multilevel NOR Circuit

- F = (AB' + A'B)(C + D)
  - AND-OR-AND circuit

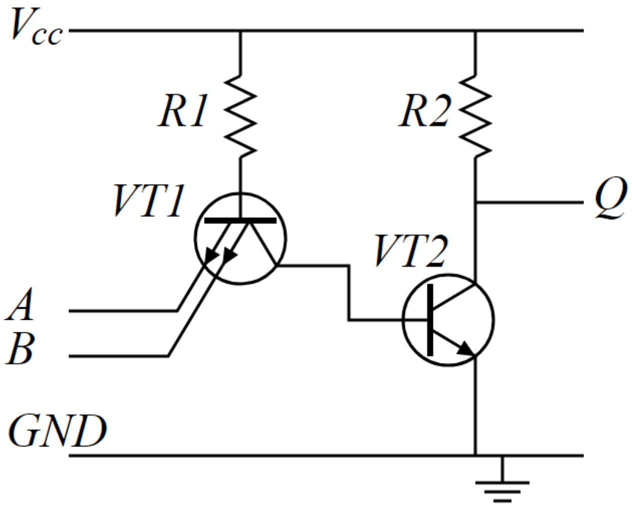


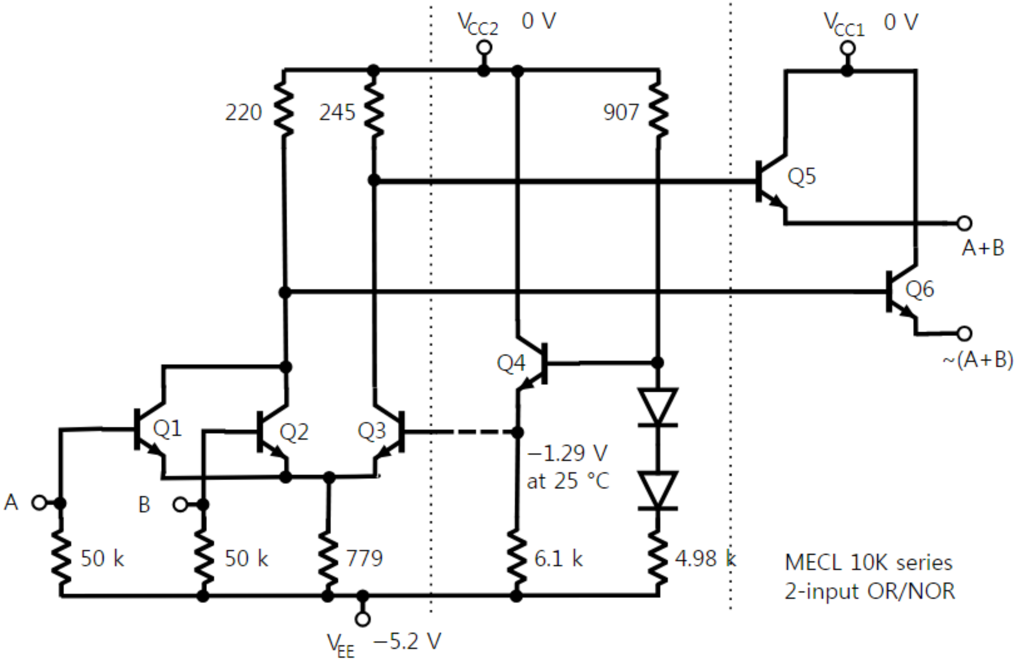
  - NOR-NOR-NOR circuit



Wired Logic

- Wired-AND
  - the function performed by tying outputs of two open-collector TTL NAND gates
- Wired-OR
  - OR function performed by tying outputs of two ECL NOR gates

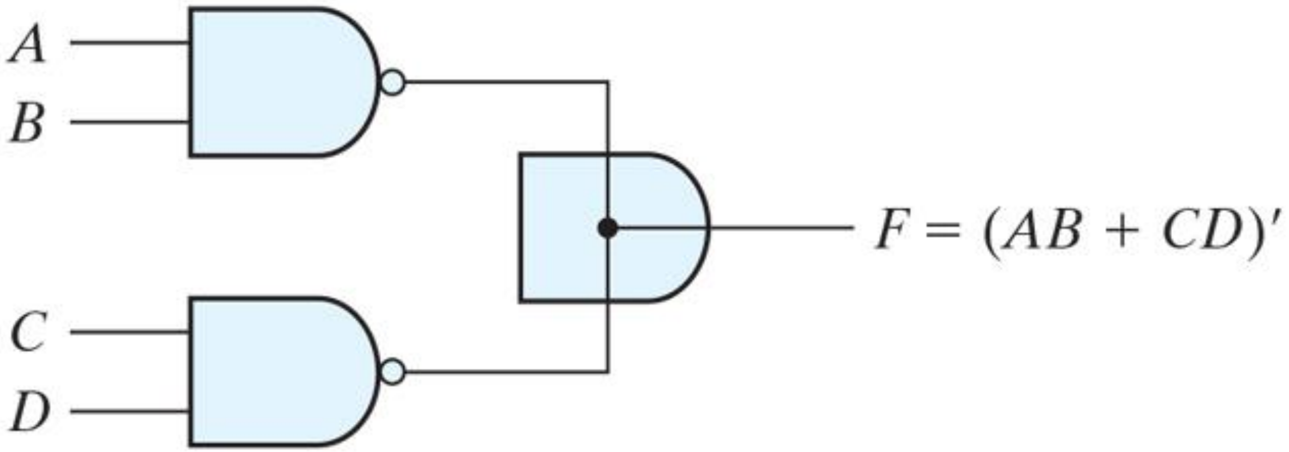




MECL 10K series
2-input OR/NOR

AOI with Wired AND

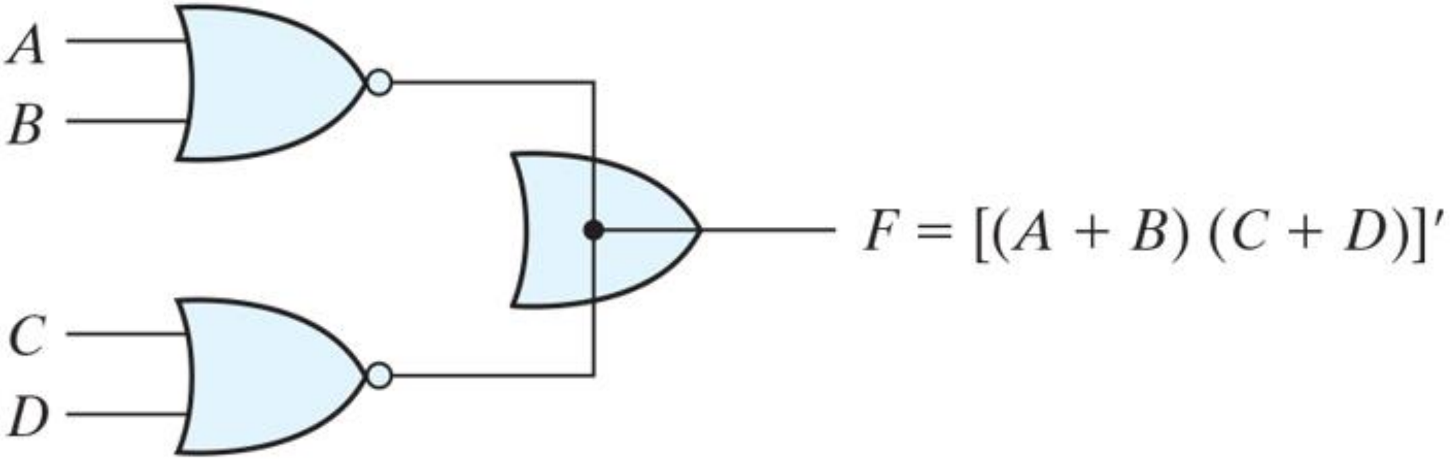
- Two level implementation with NAND-wired-AND
  - F = (AB)'(CD)'



  - F = (AB + CD)' ⇒ AND-OR-INVERT (i.e., AOI) function

OAI with Wired-OR

- Two-level implementation with NOR-wired-OR
  - F = (A + B)' + (C + D)'



  - F = ((A + B)(C + D))' ⇒ OR-AND-INVERT (i.e., OAI) function

Nondegenerate Forms

- 16 possible combination of two-level forms
  - {AND, OR, NAND, NOR} x {AND, OR, NAND, NOR}
- Degenerate form
  - 8 degenerate form = a single operation
  - ex.: AND-AND ⇒ AND; OR-OR ⇒ OR
- Nondegenerate form
  - 8 nondegenerate form = sum of products or product of sums

AND-OR	OR-AND
NAND-NAND	NOR-NOR
NOR-OR	NAND-AND
OR-NAND	AND-NOR

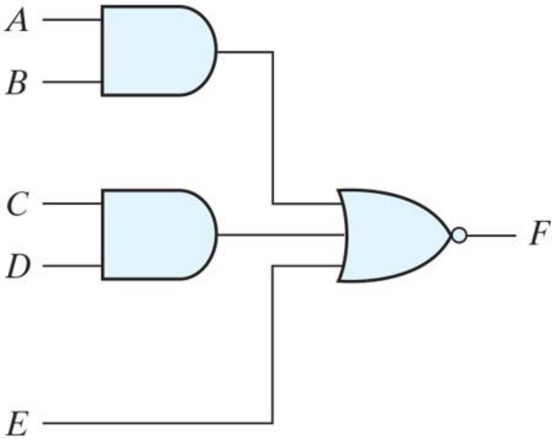
in each line, left form and right one are dual to each other

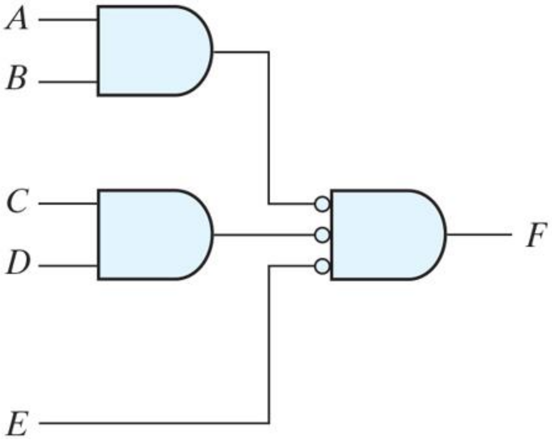
Nondegenerate Form

- Sum of product
  - AND-OR, NAND-NAND
- Product of sum
  - OR-AND, NOR-NOR
- AOI
  - AND-NOR, NAND-AND
  - complement of AOI ⇒ sum of product
- OAI
  - OR-NAND, NOR-OR
  - complement of OAI ⇒ product of sum

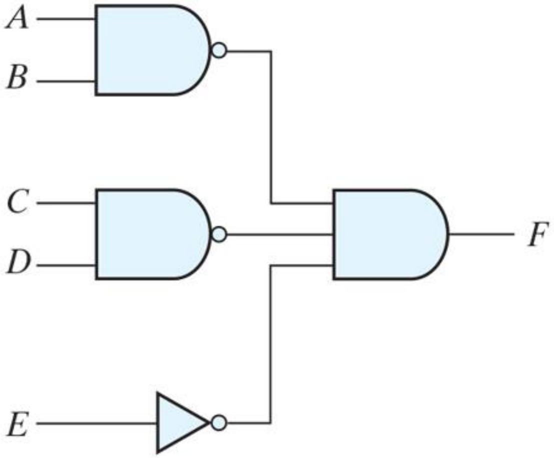
OAI(OR-AND-INVERT) Implementation

- F = (AB + CD + E)'
  - AND-NOR



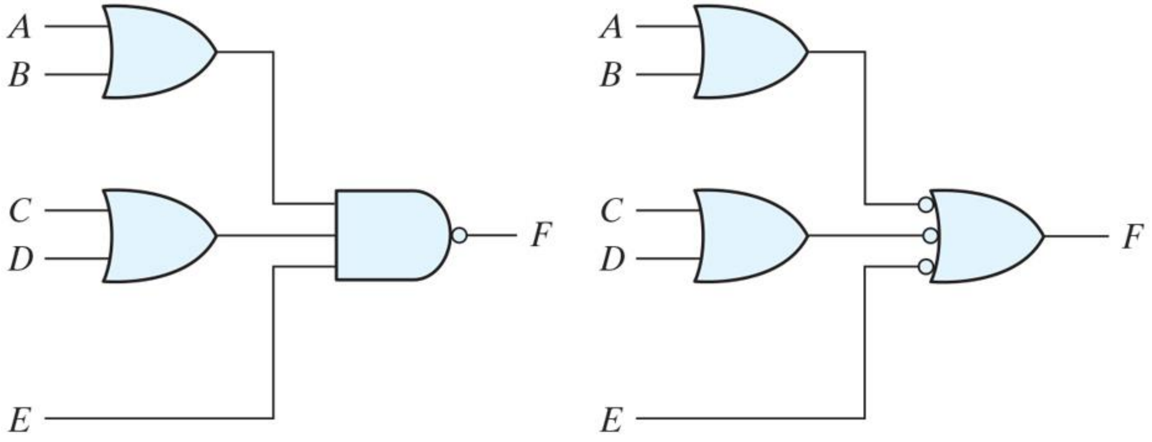


  - NAND-AND

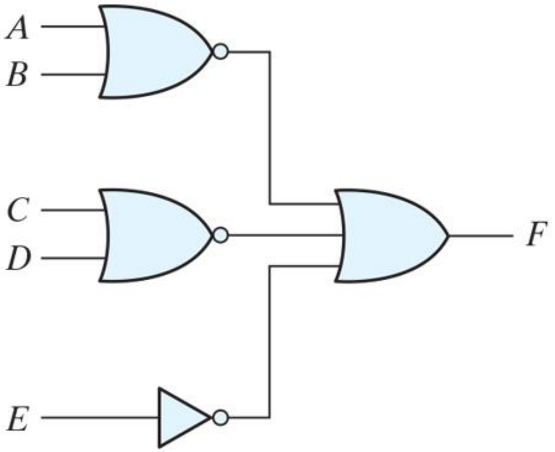


OAI(OR-AND-INVERT) Implementation

- F = ((A + B)(C + D)E)'
  - OR-NAND

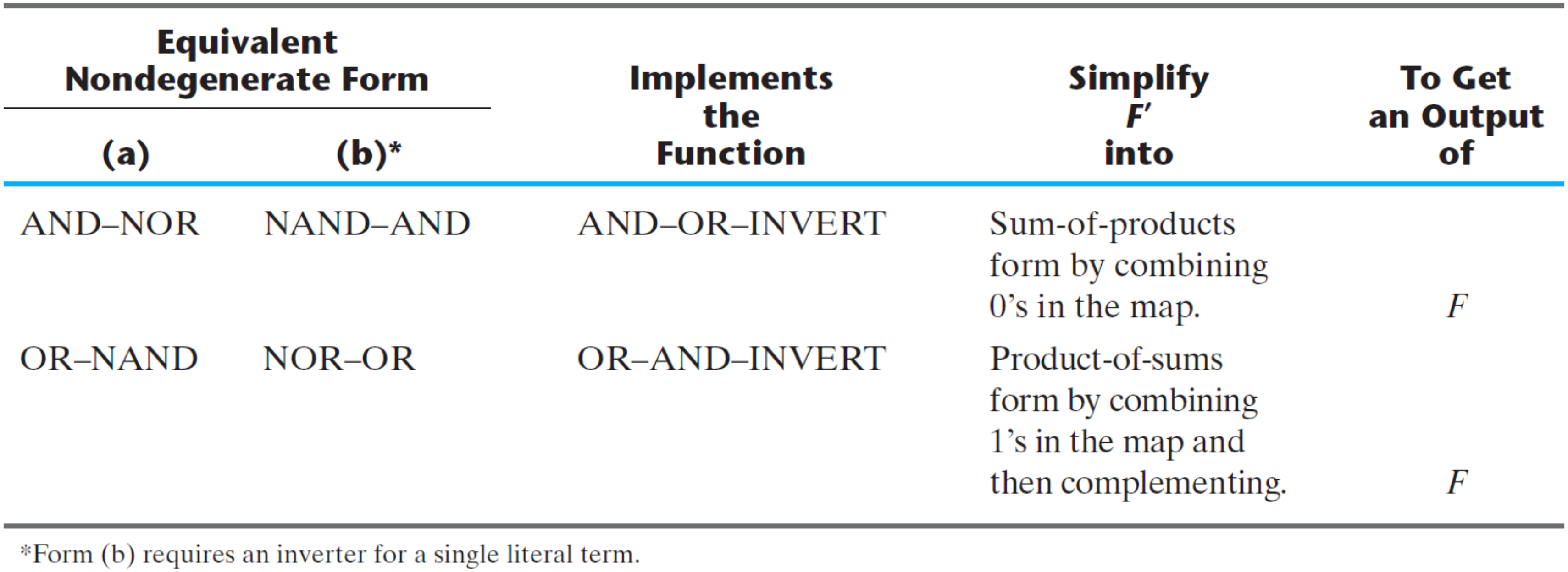


  - NOR-OR



Implementation with Other Two-level Forms

- Tabular summary

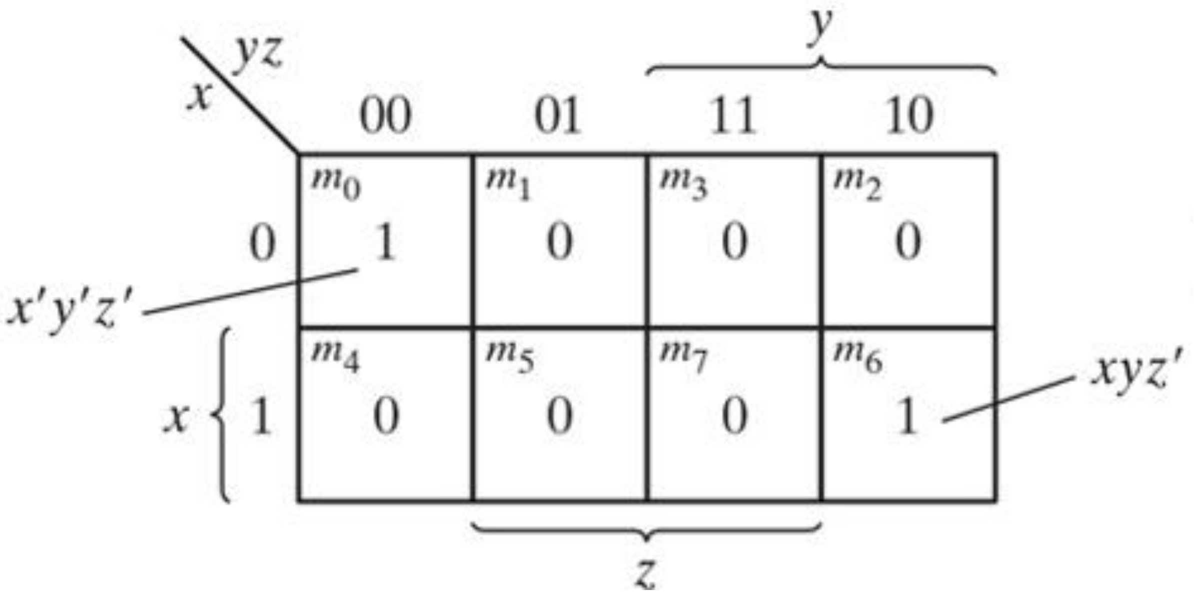


Equivalent Nondegenerate Form		Implements the Function	Simplify F' into	To Get an Output of
(a)	(b)*			
AND–NOR	NAND–AND	AND–OR–INVERT	Sum-of-products form by combining 0’s in the map.	F
OR–NAND	NOR–OR	OR–AND–INVERT	Product-of-sums form by combining 1’s in the map and then complementing.	F

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

Example 3.10

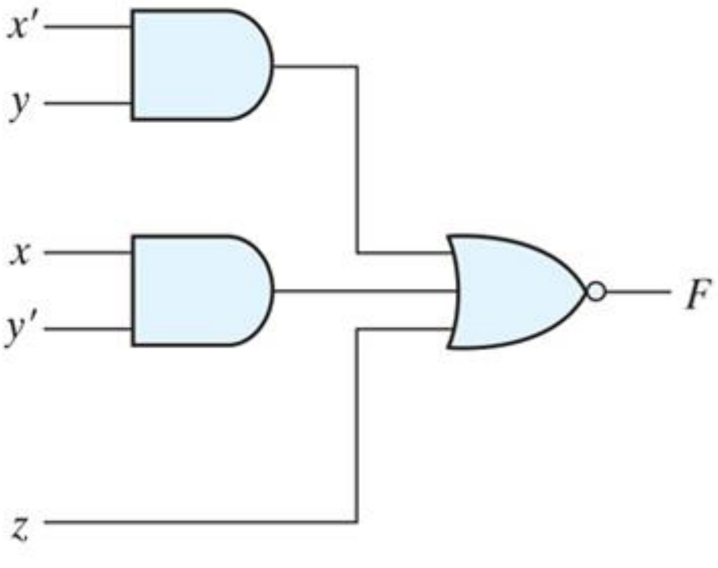
- Given function presentation



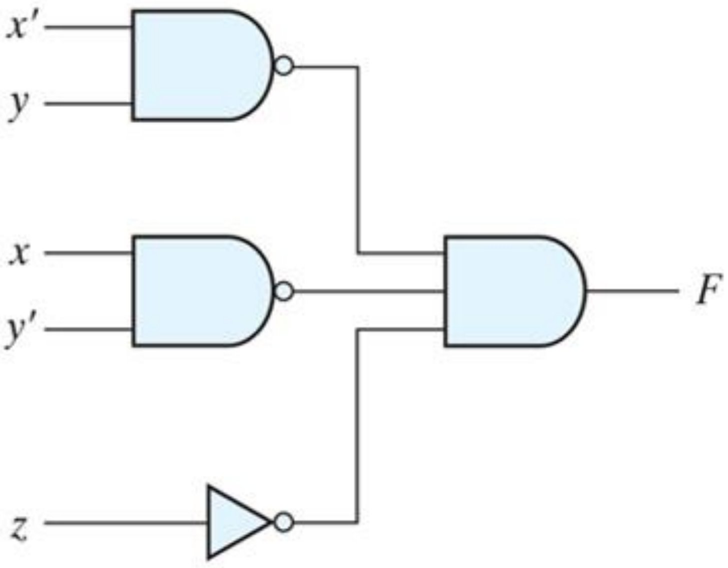
  - F = x'y'z' + xyz'
  - F' = (x + y + z)(x' + y' + z) (product-of-sums form)
  - F' = x'y + xy' + z (sum-of-products form)

Example 3.10

- Implement F in terms of AOI form
  - F' = x'y + xy' + z



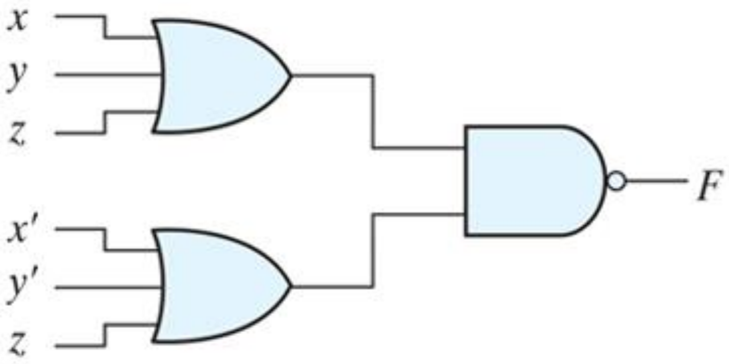
AND-NOR



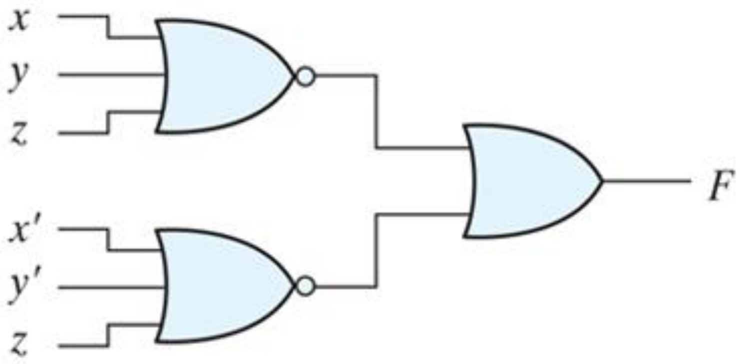
NAND-AND

Example 3.10

- Implement F in terms of OAI form
  - $F' = (x + y + z)(z' + y' + z)$



OR-NAND



NOR-OR

Exclusive-OR Function

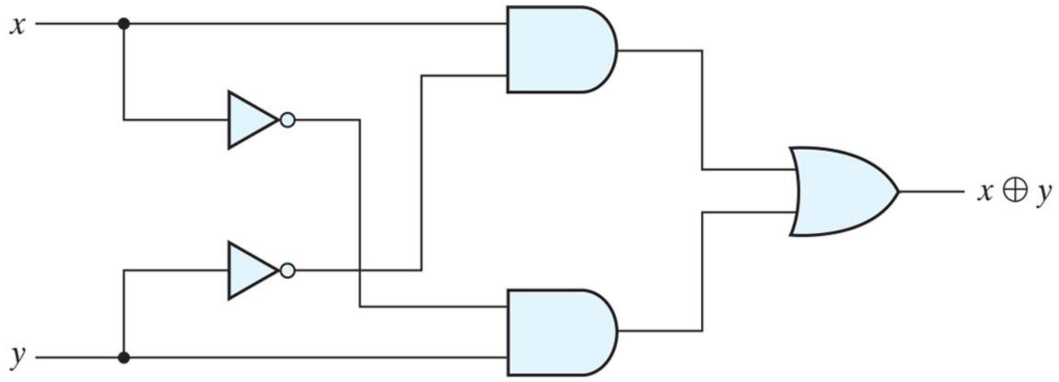
- Exclusive-OR
  - $x \oplus y = x'y + xy'$
- Exclusive-NOR
  - $(x \oplus y)' = xy + x'y'$
- Identities in XOR operation
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = x' \oplus y = (x \oplus y)'$

Exclusive-OR Functions

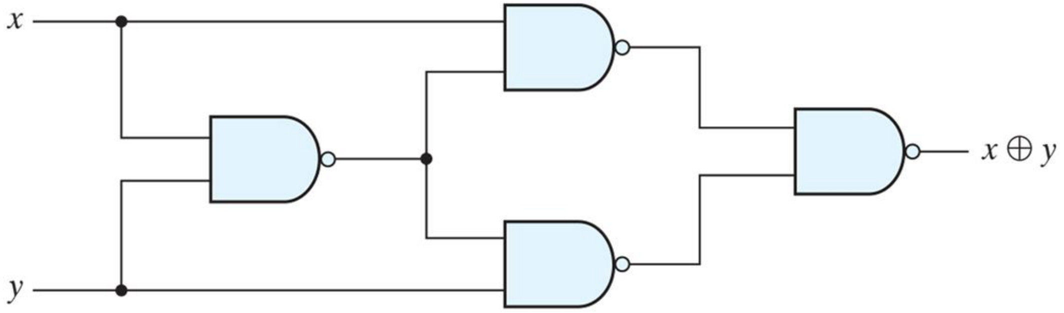
- Commutativeness
  - $A \oplus B = B \oplus A$
  - what is the physical meaning?
- Associativeness
  - $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$
  - what is the physical meaning?

XOR Implementation

- $x \oplus y = x'y + xy'$
  - NOT, AND, OR gates

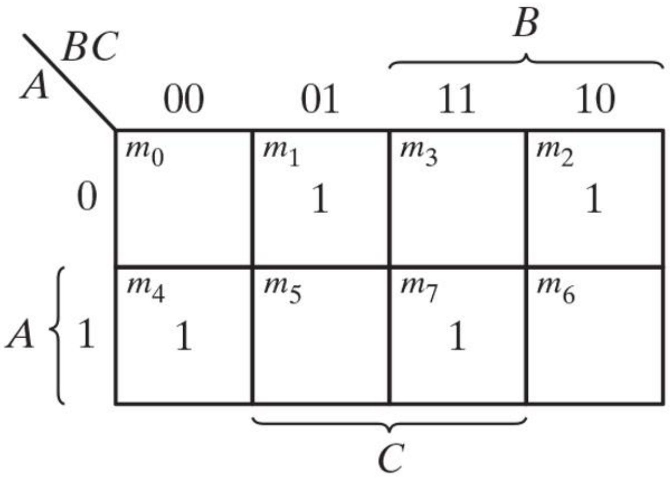


- $x \oplus y = (xy)'x + (xy)'y$
  - NAND gates



Odd Function

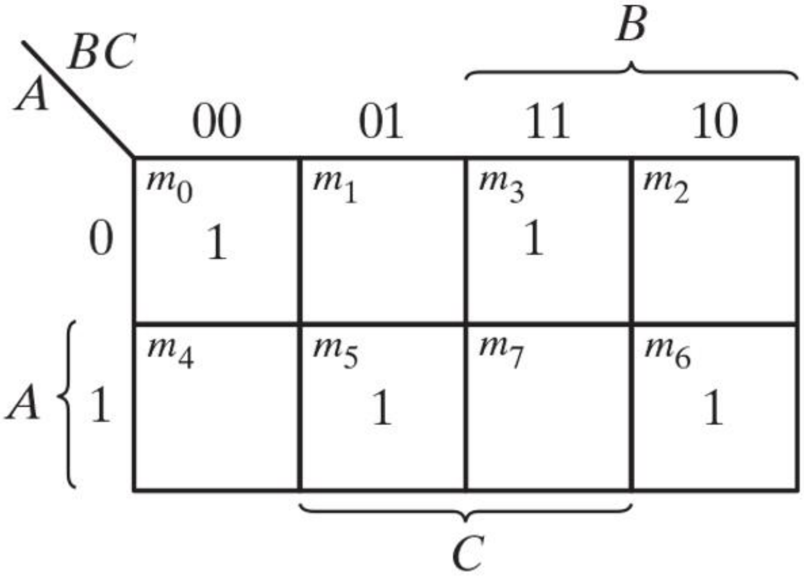
- 3 variable XOR function
  - $A \oplus B \oplus C = \sum (1, 2, 4, 7)$



- Definition: odd function
  - Multi-variable XOR function is equal to 1 if the number of variables that are equal to 1 is odd.

Even Function

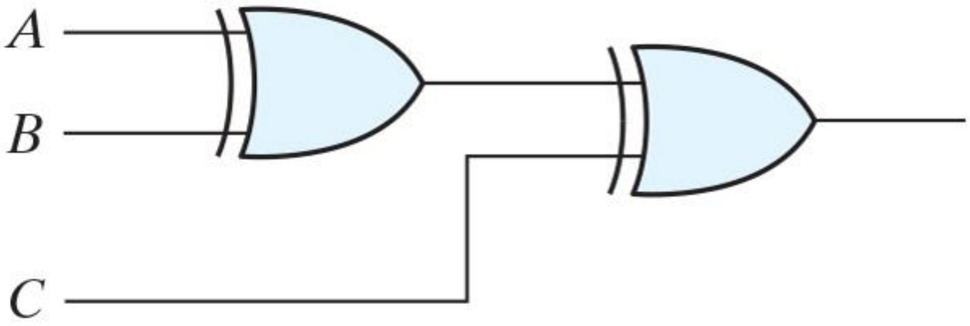
- Definition
  - complement of the multi-variable XOR function is equal to 1 if the number of variables that are equal to 1 is even.
  - $(A \oplus B \oplus C)' = \sum(0, 3, 5, 6)$



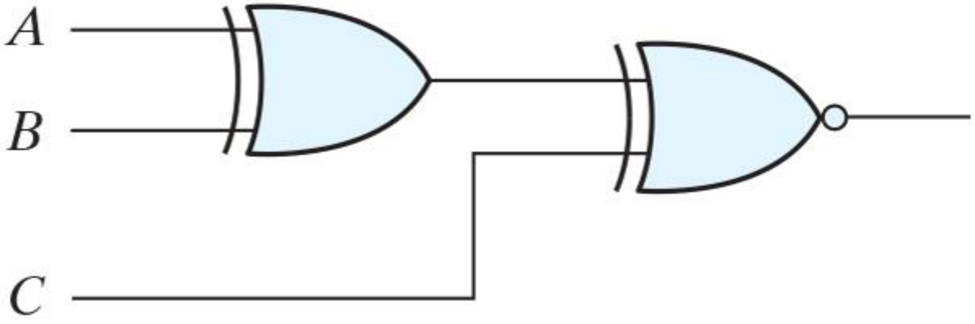
A \ BC	00	01	11	10
0	m_0 1	m_1	m_3 1	m_2
1	m_4	m_5 1	m_7	m_6 1

Implementation of 3-variable XOR

- Odd function



- Even function

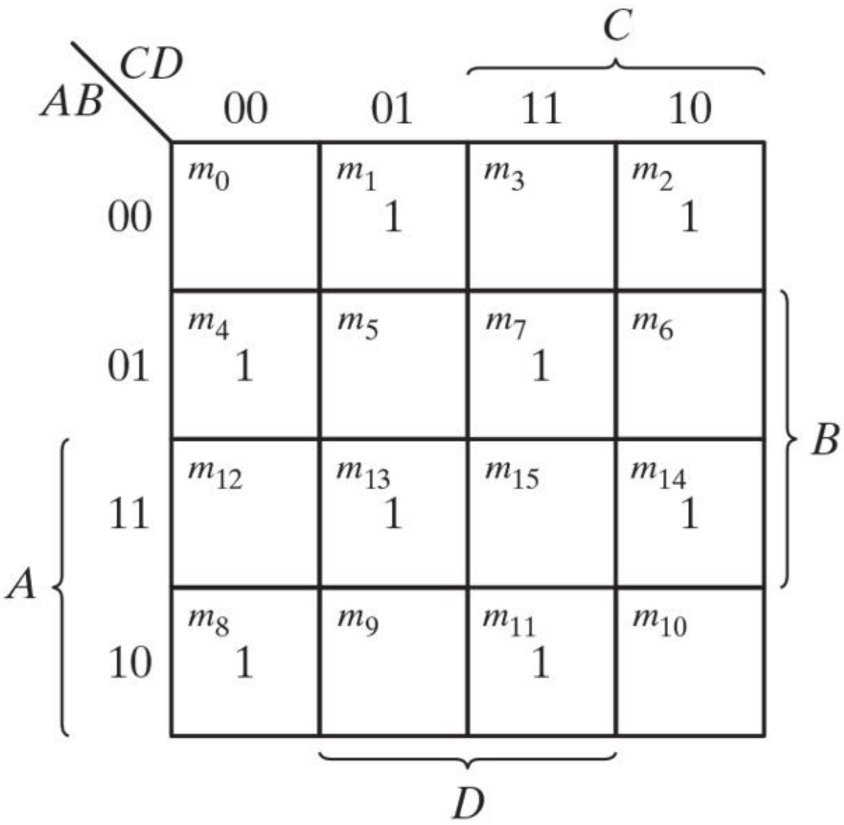


Four-variable XOR function

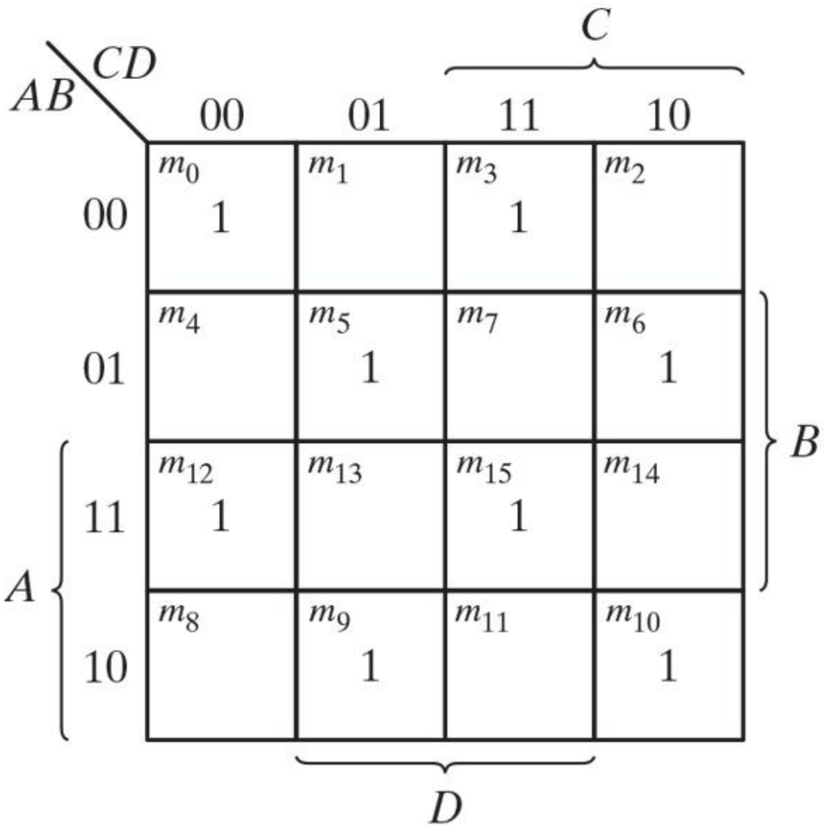
- Odd function
  - A ⊕ B ⊕ C ⊕ D

Even function

(A ⊕ B ⊕ C ⊕ D)'



AB \ CD	00	01	11	10
00	m_0	m_1 1	m_3	m_2 1
01	m_4 1	m_5	m_7 1	m_6
11	m_{12}	m_{13} 1	m_{15}	m_{14} 1
10	m_8 1	m_9	m_{11} 1	m_{10}

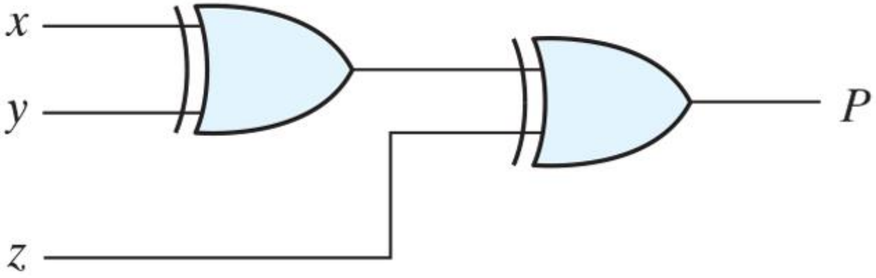


AB \ CD	00	01	11	10
00	m_0 1	m_1	m_3 1	m_2
01	m_4	m_5 1	m_7	m_6 1
11	m_{12} 1	m_{13}	m_{15} 1	m_{14}
10	m_8	m_9 1	m_{11}	m_{10} 1

Parity Generation

- Even parity generator for 3-bit message
  - 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

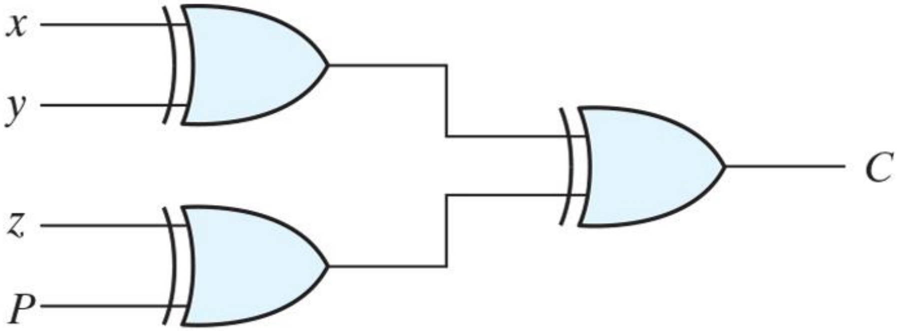


  - P is an odd function ⇒ $P = x \oplus y \oplus z$

Parity Checker

- Even parity checker for 3-bit message
  - Truth table
  - C(x,y,z,P) is an odd function

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



Discussion~~~