Gate-Level Minimization

Hsi-Pin Ma 馬席彬

https://eeclass.nthu.edu.tw/course/3452

Department of Electrical Engineering

National Tsing Hua University

Outline

- The Map Method
- Technology Mapping

The Map Method

Map Representation

- A function’s truth-table representation is *unique*, while its algebraic expression is *not unique*.
- Complexity of digital circuit (gate count) $\propto$ complexity of algebraic expression (literal count)
  - $F_2=x'y'z+x'yz+xy'$ (3 AND, 1 OR term, 8 literals)
  - $F_2=x'z+xy'$ (2 AND terms, 1 OR terms, 4 literals)
- The simplest algebraic expression is one that has **minimum number of terms** with the **smallest possible number of literals** in each term

Karnaugh Map (K-map)

- An array of squares each representing one minterm to be minimized
- Each K-map defines a unique Boolean function
  - A Boolean function can be represented by a truth table, a Boolean expression, or a map
- K-map is a visual diagram of all possible ways a function may be expressed
  - Provide visual aid to identify PIs and EPIs
  - For manual minimization of Boolean functions

Merging Minterms

- In function F_2, m_1 and m_3 in the truth table differ only in one position

x	y	z	F_2
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

001
011
→ 0X1

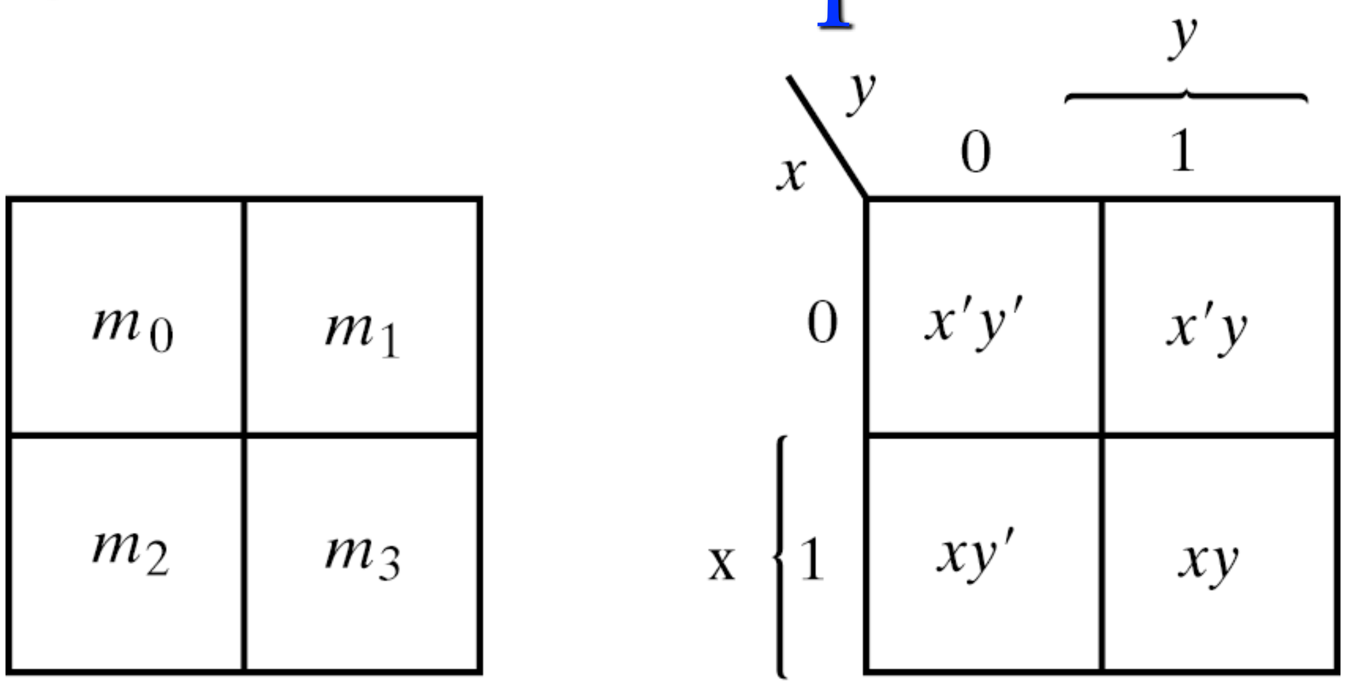
  - X: matches either 0 or 1

- The minterms in a function can be merged to form a larger (or simpler) product term

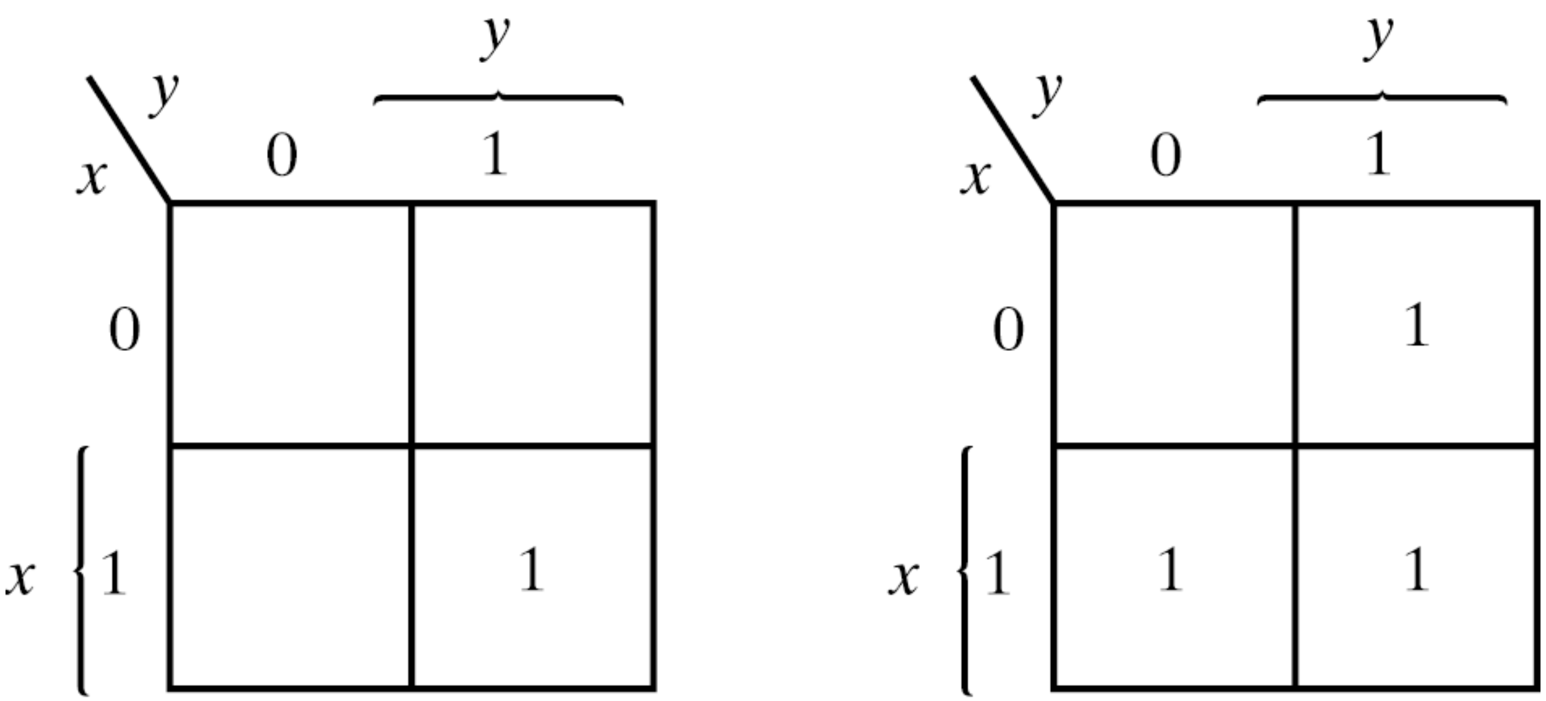
$$f_{0X1} = x'y'z + x'yz = x'z(y'+y) = x'z$$

Two-Variable Map

x	y	f
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3



(a) (b)



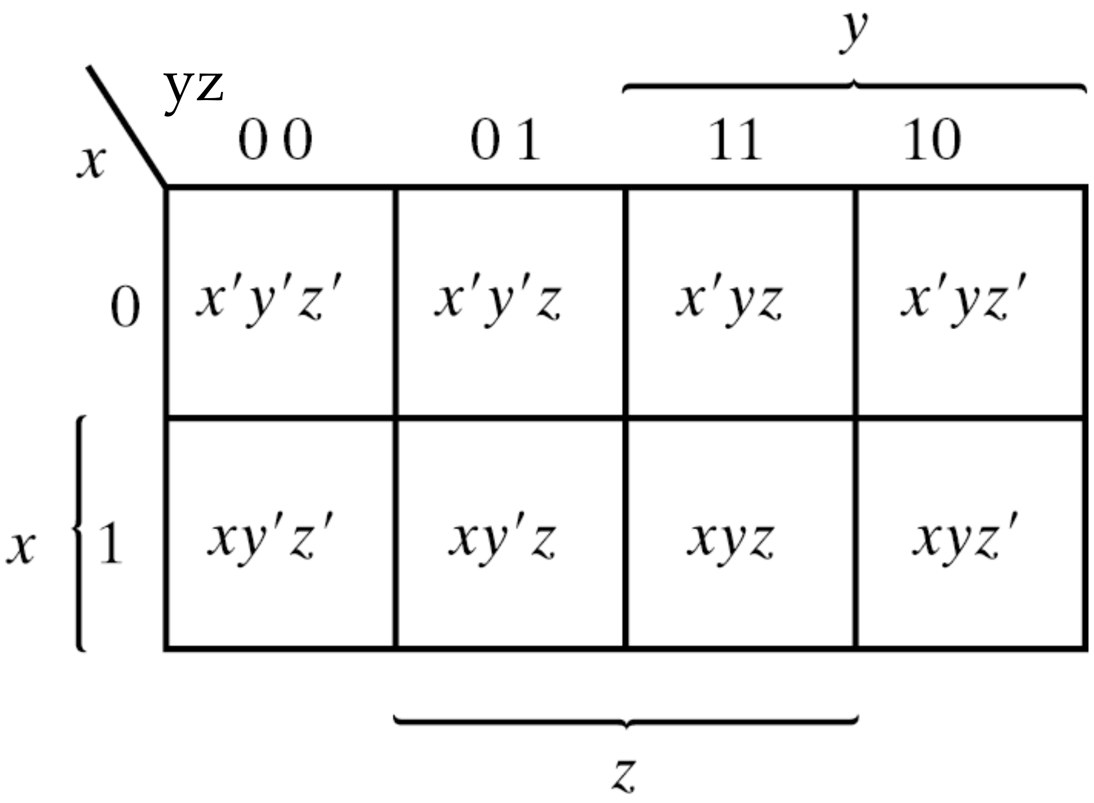
(a) xy (b) $x + y$

m_3 $(m_1+m_2+m_3)$

Three-Variable Map (1/5)

- Minterms are arranged in the Gray-code sequence
- Any 2 (***horizontally*** or ***vertically***) adjacent squares differ by exactly 1 variable, which is complemented in one square and uncomplemented in the other.
- Any 2 minterms in adjacent squares that are ORed together will cause a removal of the different variable

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



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

Three-Variable Map (2/5)

- Example (adjacent squares)
  - m_5 OR m_7 can be simplified
    - $m_5+m_7=xy'z+xyz=xz(y+y')=xz$
  - m_0 OR m_2 can be simplified
    - $m_0+m_2=x'y'z'+x'yz'=x'z'(y+y')=x'z'$
  - m_1 OR m_3 OR m_5 OR m_7 can be simplified
    - $m_1+m_3+m_5+m_7=x'y'z+x'yz+xy'z+xyz=x'z(y+y')+xz(y+y')=x'z+xz=z$

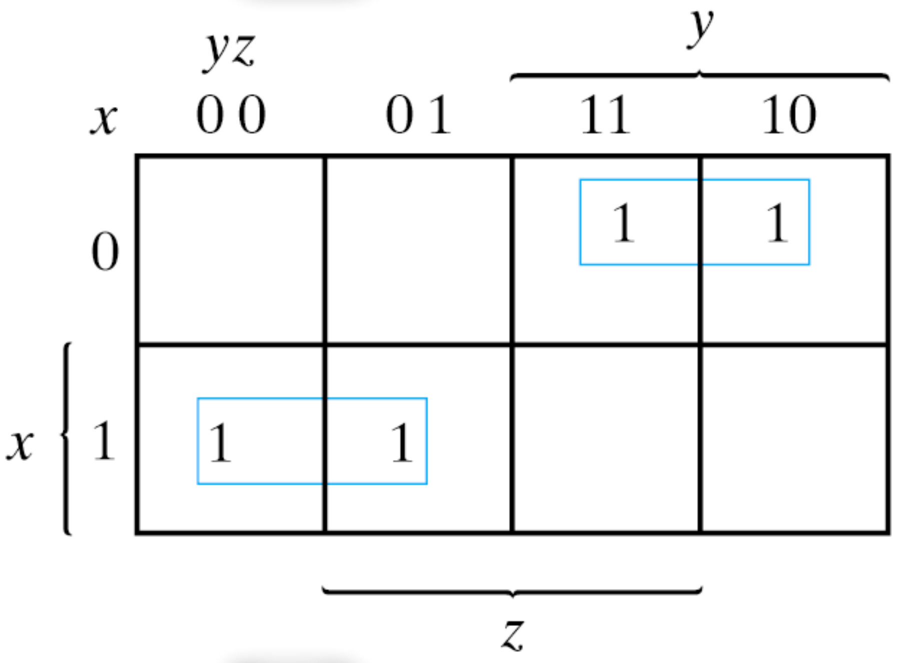
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

$$\begin{array}{cc|c|c|c|c|} & & & & \multicolumn{2}{c}{\overbrace{\hspace{6em}}^{y}} \\ & x \backslash yz & 00 & 01 & 11 & 10 \\ \hline & 0 & x'y'z' & x'y'z & x'yz & x'yz' \\ \hline x & 1 & xy'z' & xy'z & xyz & xyz' \\ \hline & & & \multicolumn{2}{c}{\underbrace{\hspace{6em}}_{z}} & \end{array}$$

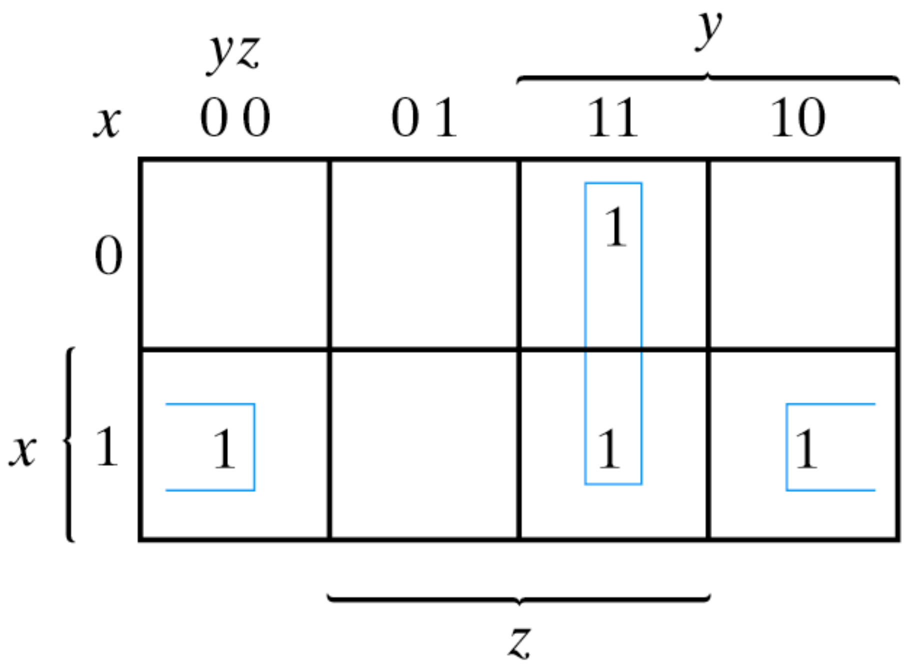
Three-Variable Map (3/5)

- Example

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

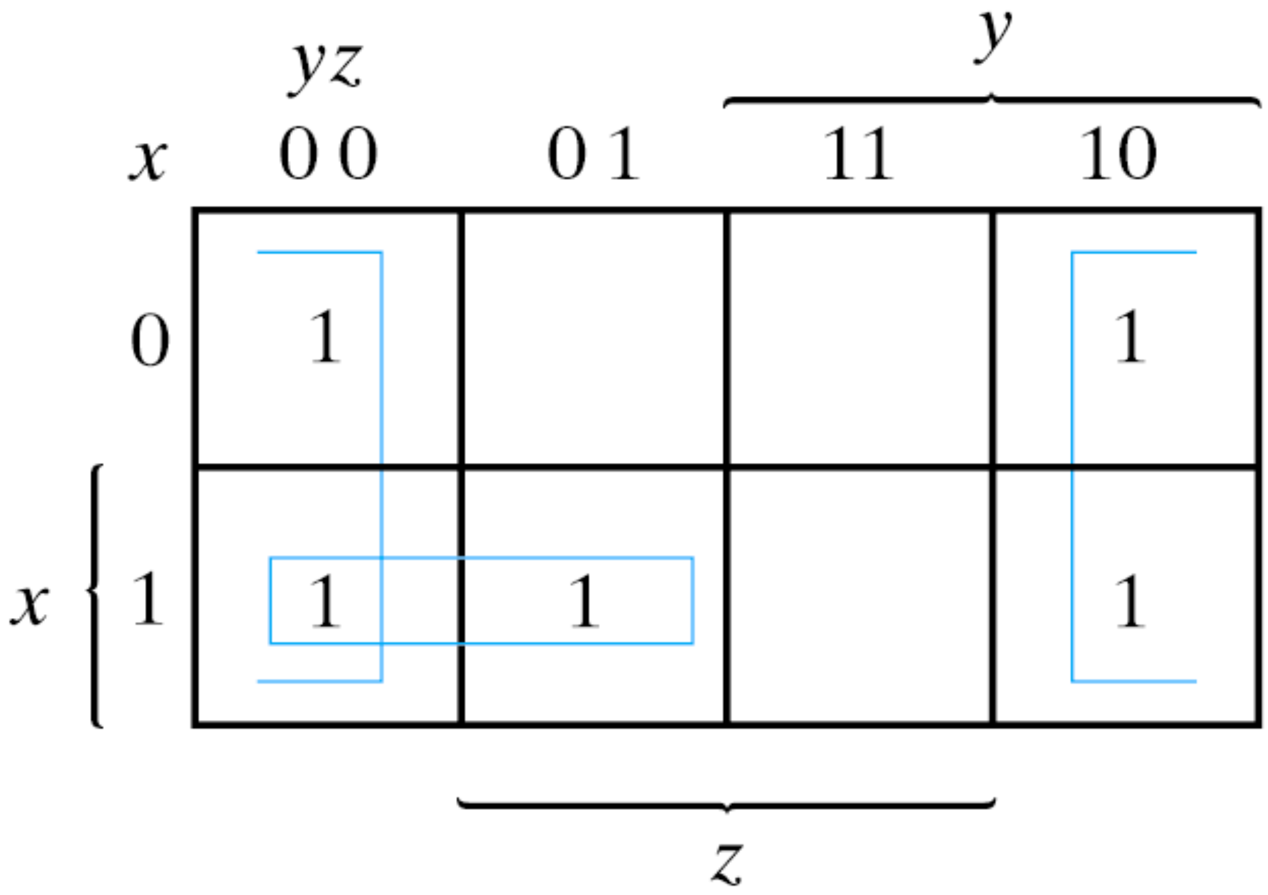


$$F(x,y,z)=\sum(3,4,6,7)=yz+xz'$$



Three-Variable Map (4/5)

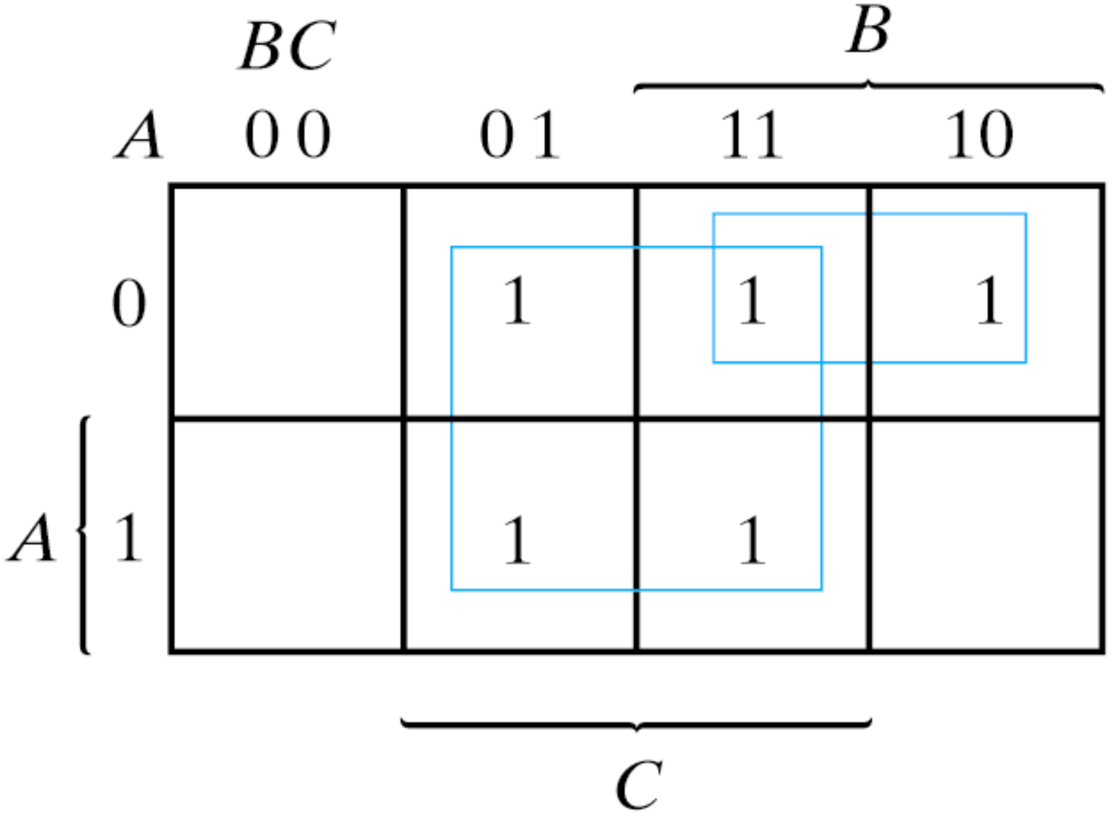
- Example $F(x, y, z) = \sum(0, 2, 4, 5, 6) = z' + xy'$



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

Three-Variable Map (5/5)

- Example $F = A'C + A'B + AB'C + BC$



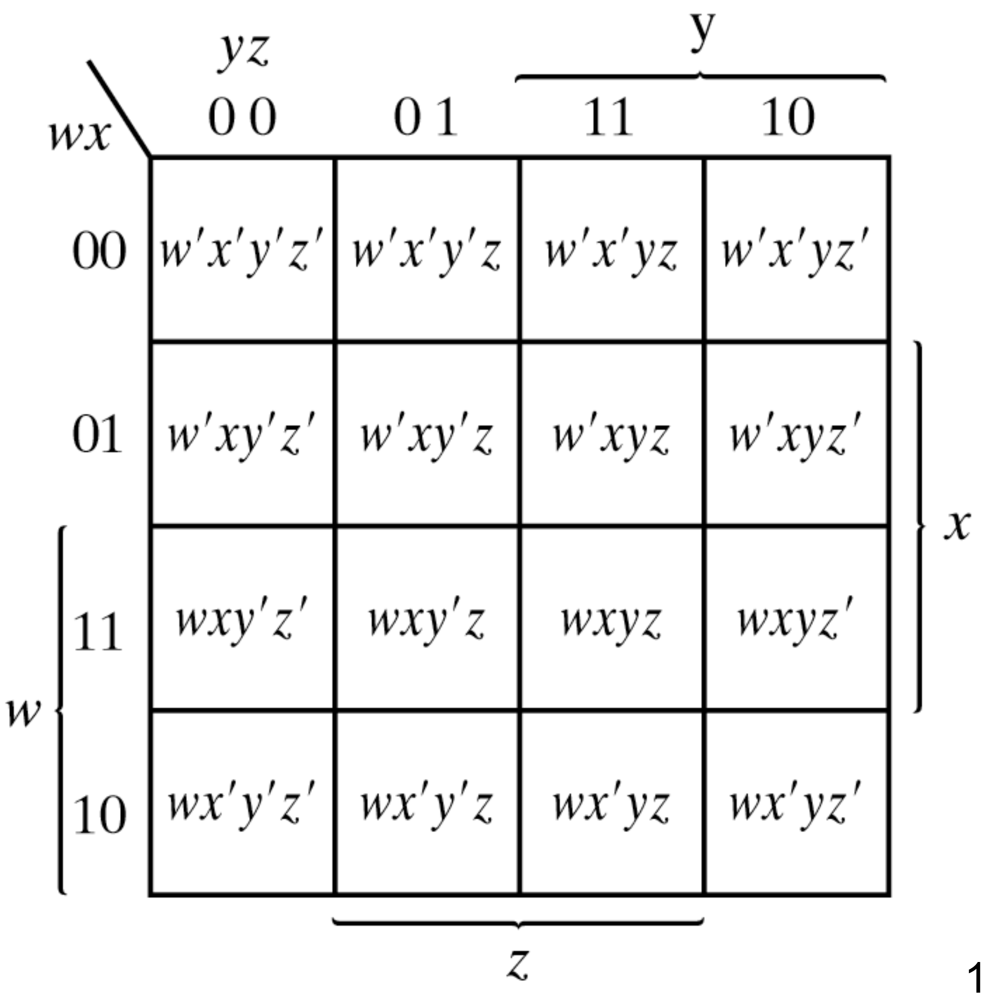
A \ BC	00	01	11	10
0		1	1	1
1		1	1	

$$F(A, B, C) = \sum(1, 2, 3, 5, 7) = C + A'B$$

Four-Variable Map (1/3)

Number of adjacent squares	Number of minterms	Number of literals	example
1	1	4	wxyz
2	2	3	wxy
4	4	2	wx
8	8	1	w
16	16	constant '1'	1

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}

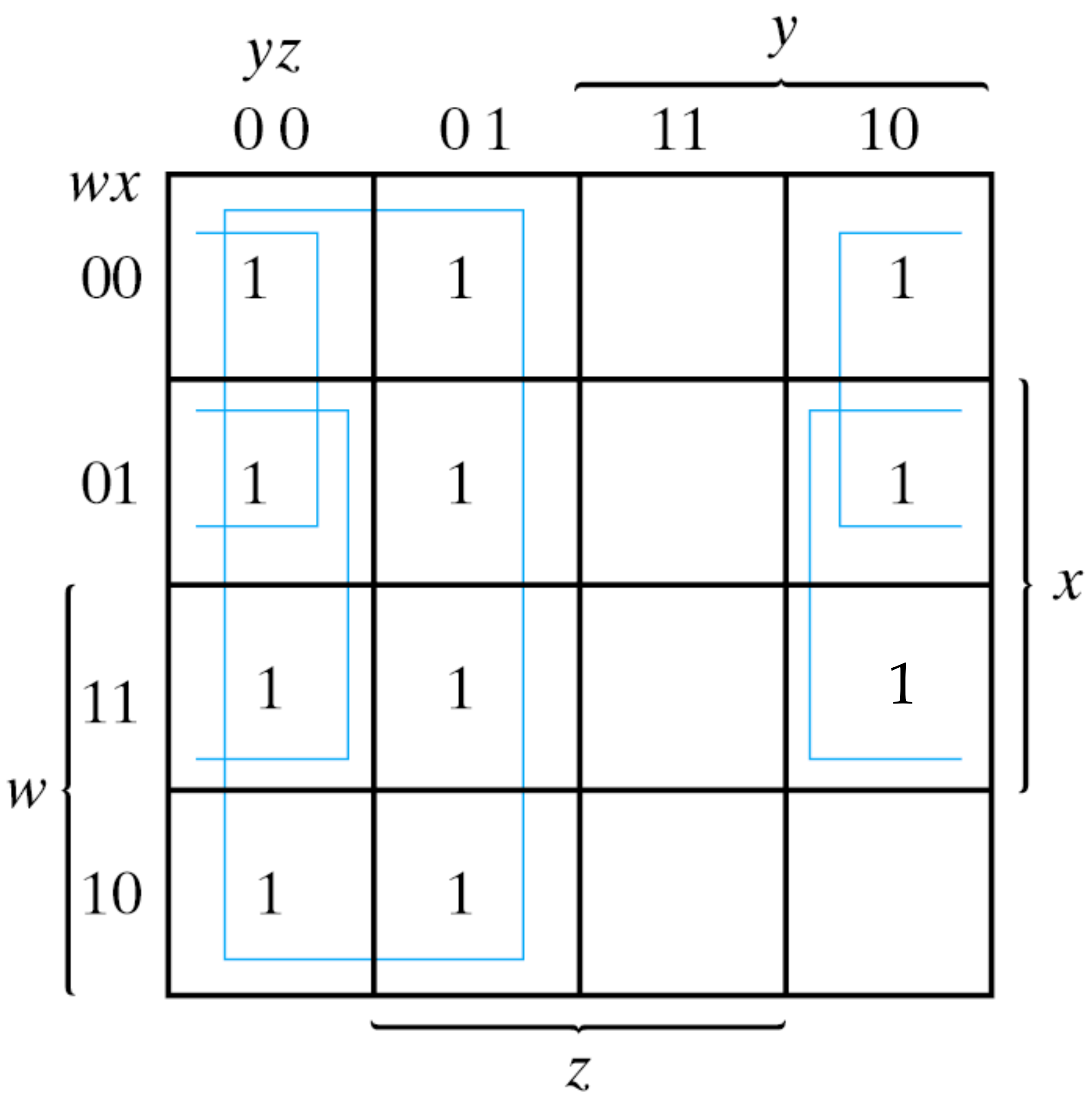


wx \ yz	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

Four-Variable Map (2/3)

- Example

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



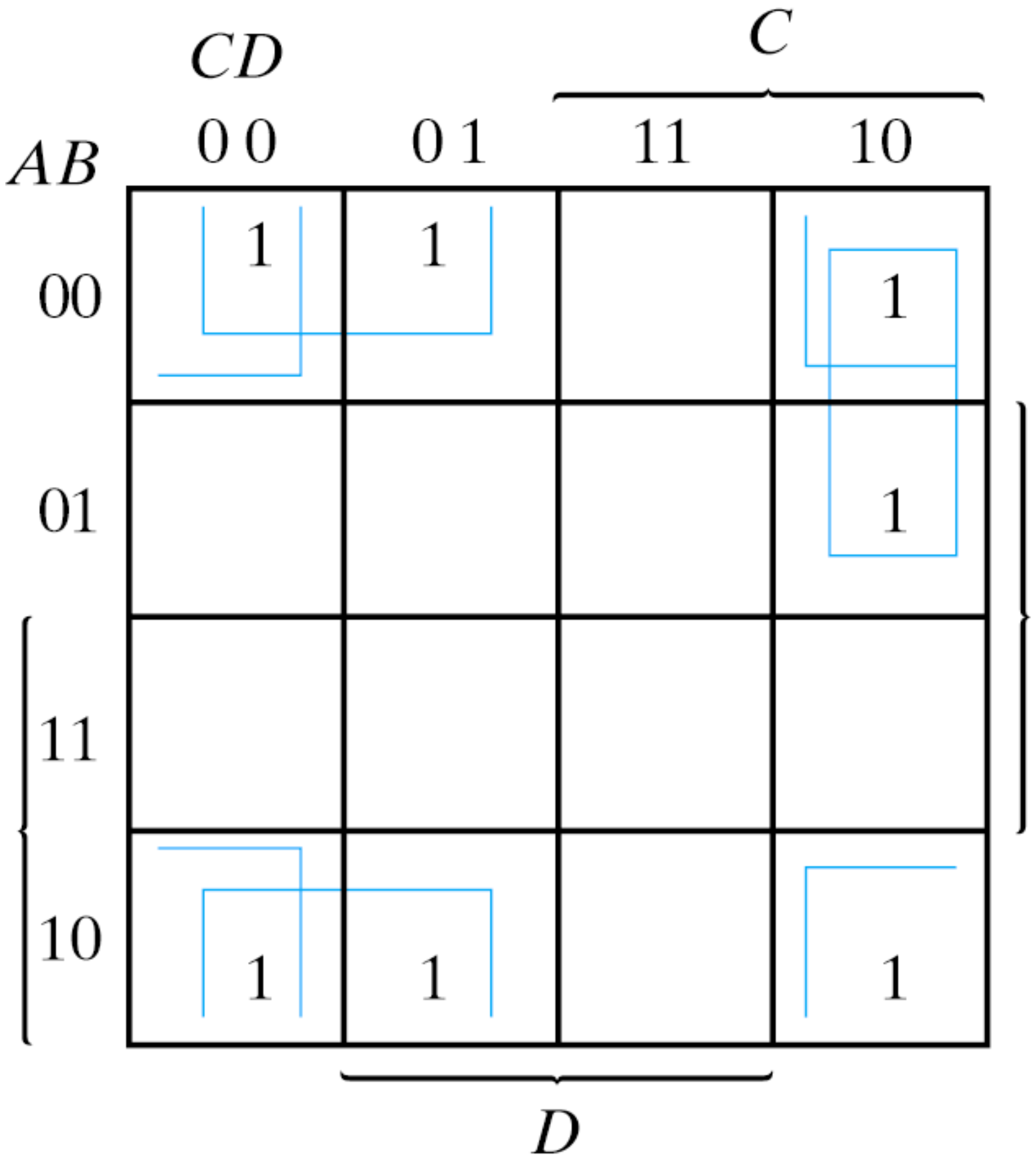
wx \ yz	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

★Minimize the number of groups
★Maximize the group size

Four-Variable Group (3/3)

- Example

$$F = A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$$



	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

★Minimize the number of groups

★Maximize the group size

Implicants

- *Implicant* of a function: any product term that *implies* the function
  - A product term that is only true when a function is true
- Example: in F_2 function

	minterm	implicant
m_1	v	v
m_2	v	x
0X1	x	v

1-minterm

0-minterm

x	y	z	F_2
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Prime and Essential Prime Implicants

- Prime implicant (PI)
  - The implicant that cannot be merged into a larger one
- Essential prime implicant (EPI)
  - The one and only one prime implicant that contains a particular minterm of a function
  - The EPI cannot be removed from a description of a function

★Minimize the number of groups
★Maximize the group size

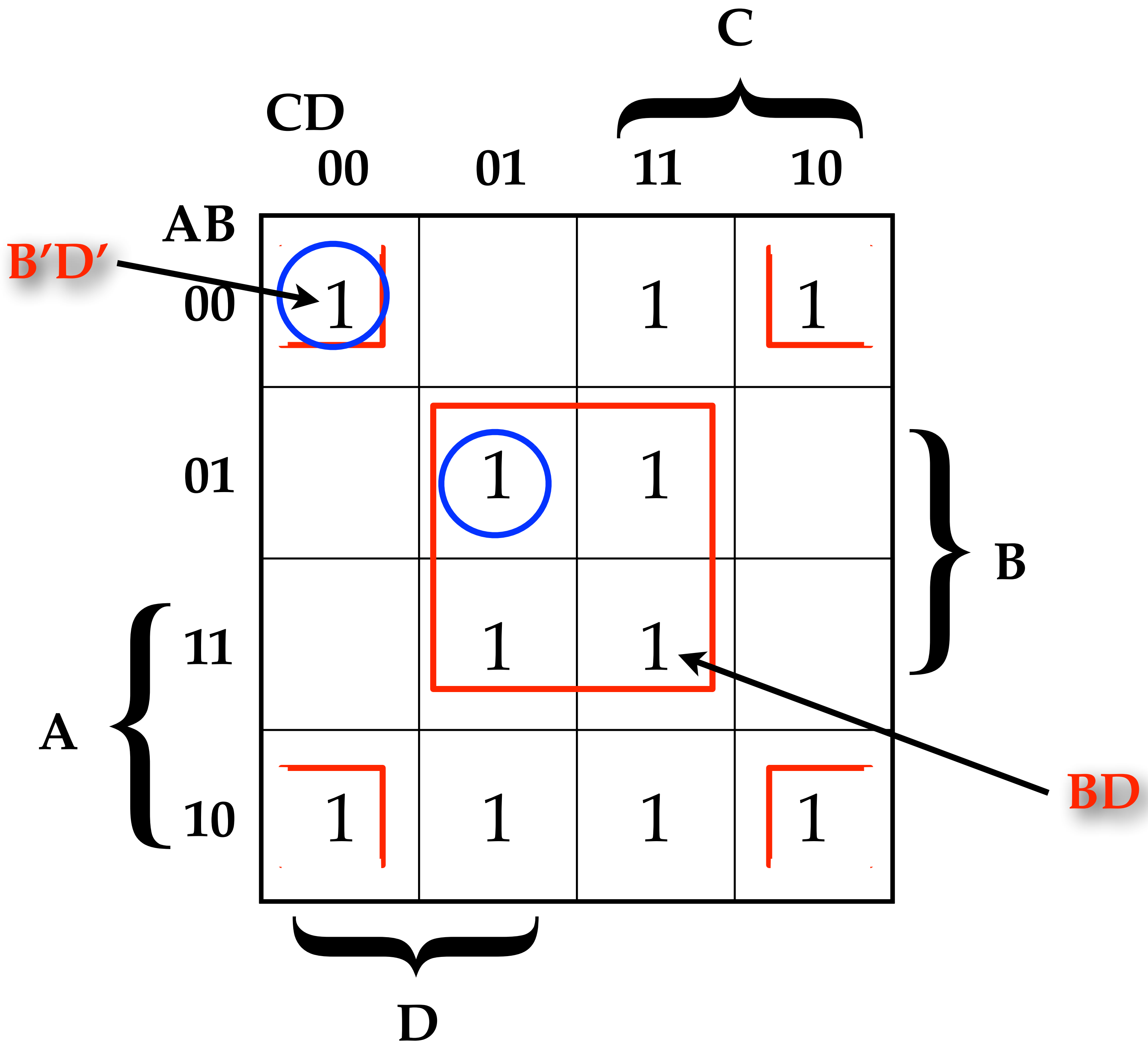
Covering a Function (1/3)

- Procedure to select an inexpensive set of implicants
  - Start with an empty cover
  - Add all essential prime implicants to the cover
  - For each remaining uncovered minterm, add the largest implicant that covers that minterm to the cover
- The procedure will always result in a ***good*** cover, but no guarantee the lowest-cost cover.

Covering a Function (2/3)

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

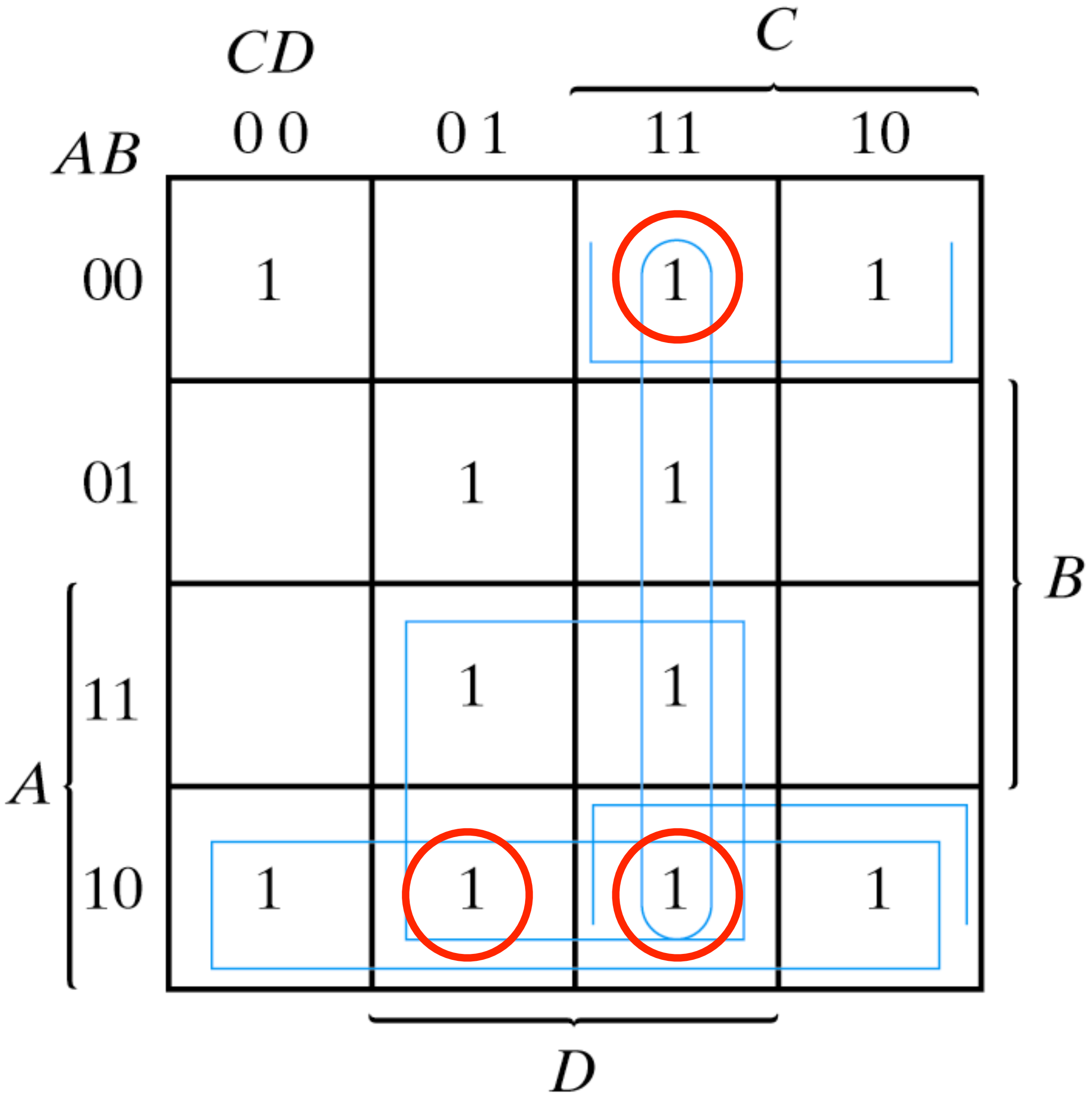
Find EPI first



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

Covering a Function (3/3)

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



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

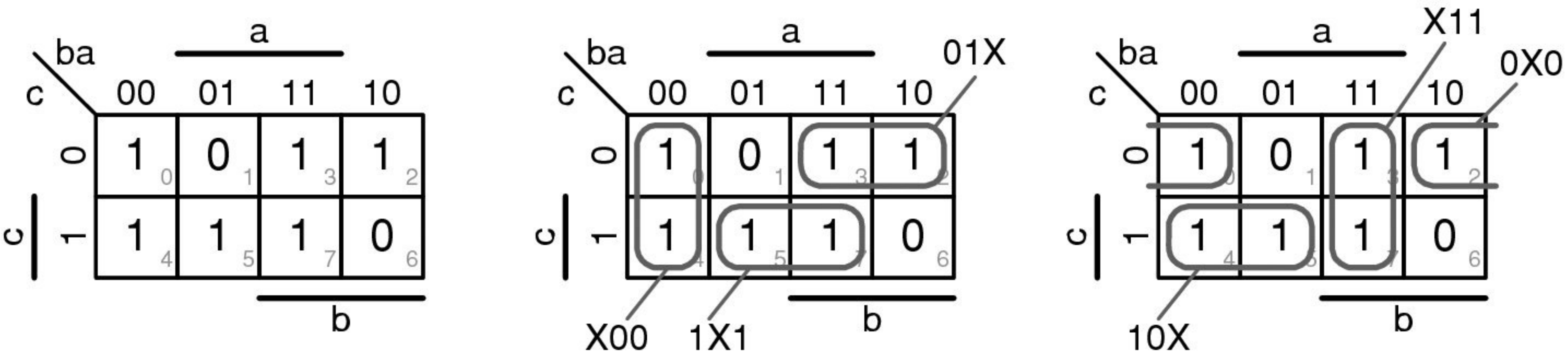
(b) Prime implicants CD, B′C AD, and AB′

Find other PI

$$
\begin{aligned}
F &= BD + B'D' + CD + AD \\
&= BD + B'D' + CD + AB' \\
&= BD + B'D' + B'C + AD \\
&= BD + B'D' + B'C + AB'
\end{aligned}
$$

Non-unique Minimum Cover

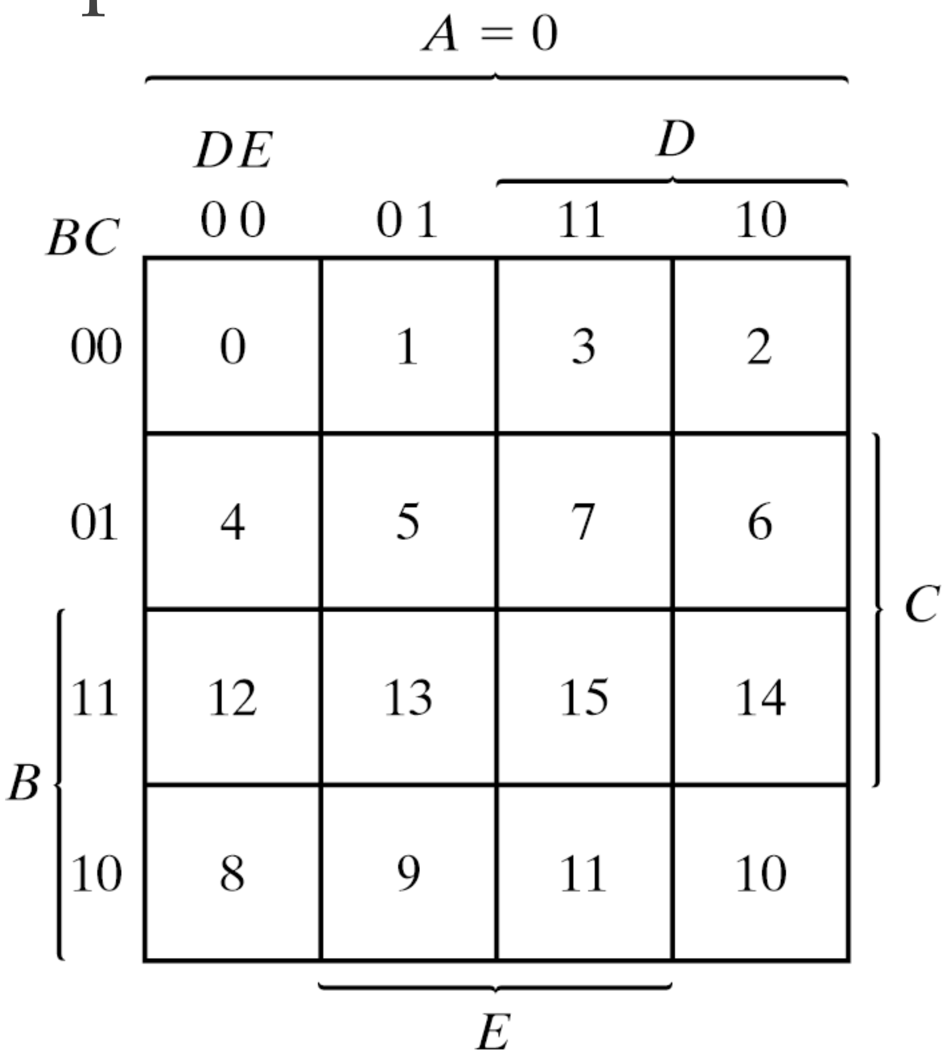
- No essential prime implicants
- Two or more possible covers exist: not unique



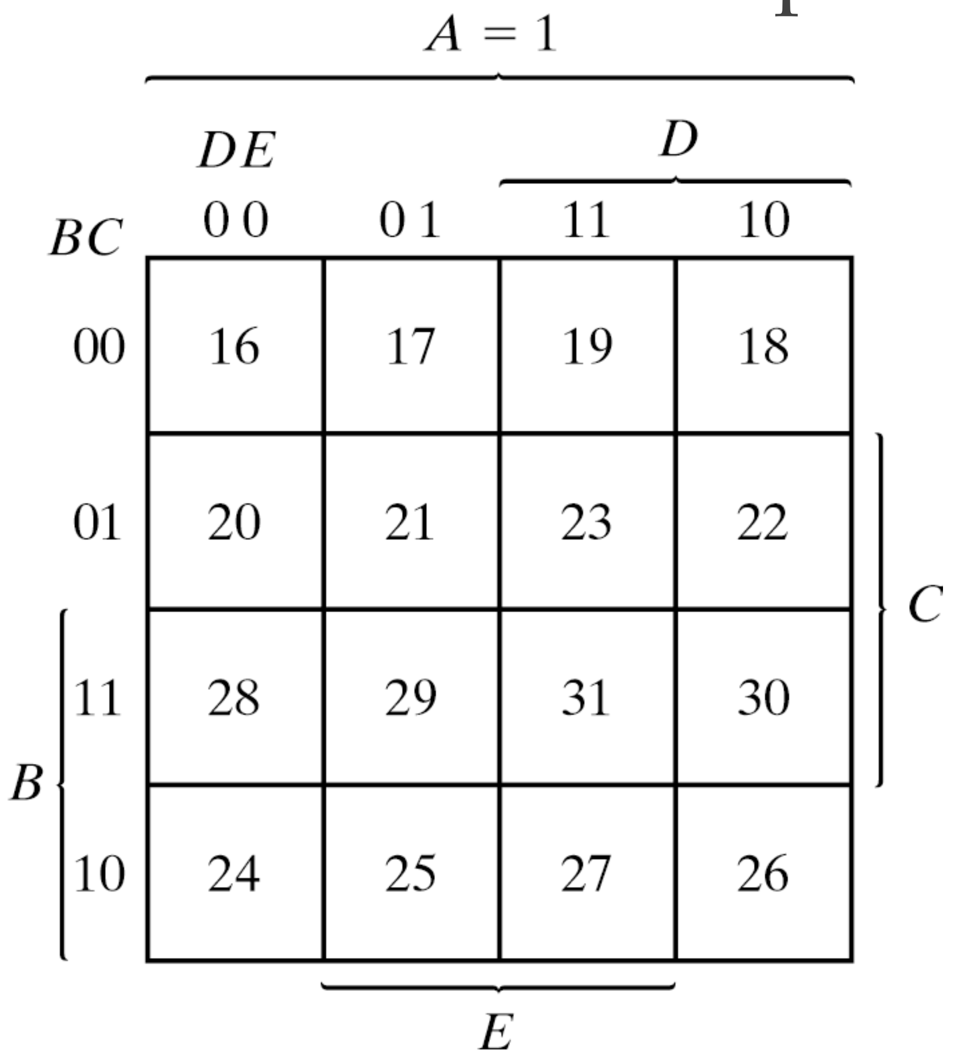
(a) (b) (c)

Five-Variable Map

- Imagine that the 2 maps are superimposed on one another.
  - It is possible to construct a 6-variable map with 4-variable maps by similar procedure.
  - Maps of 6 or more variables are hard to read=> impractical



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

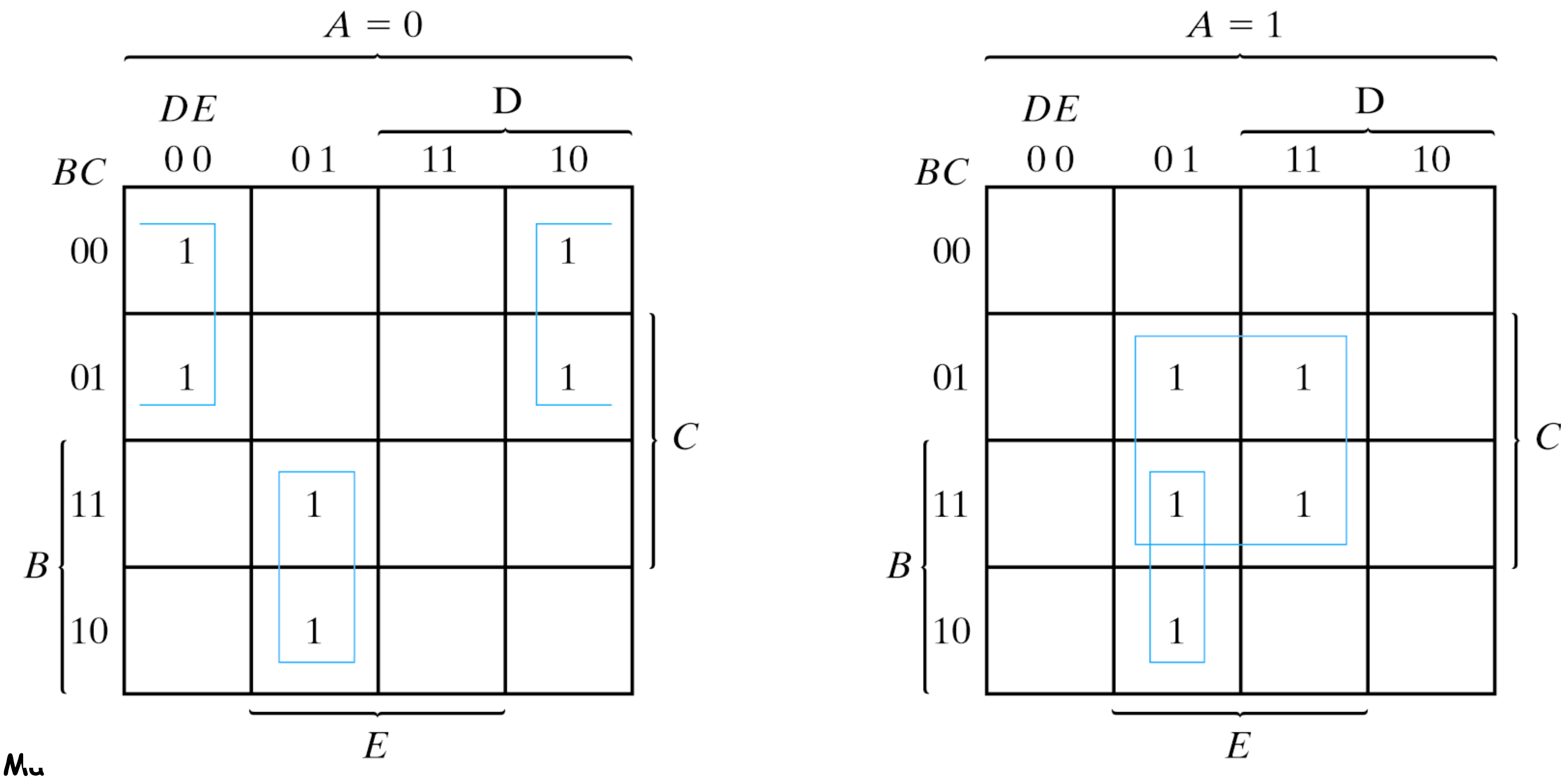


BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

Five-Variable Map

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

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



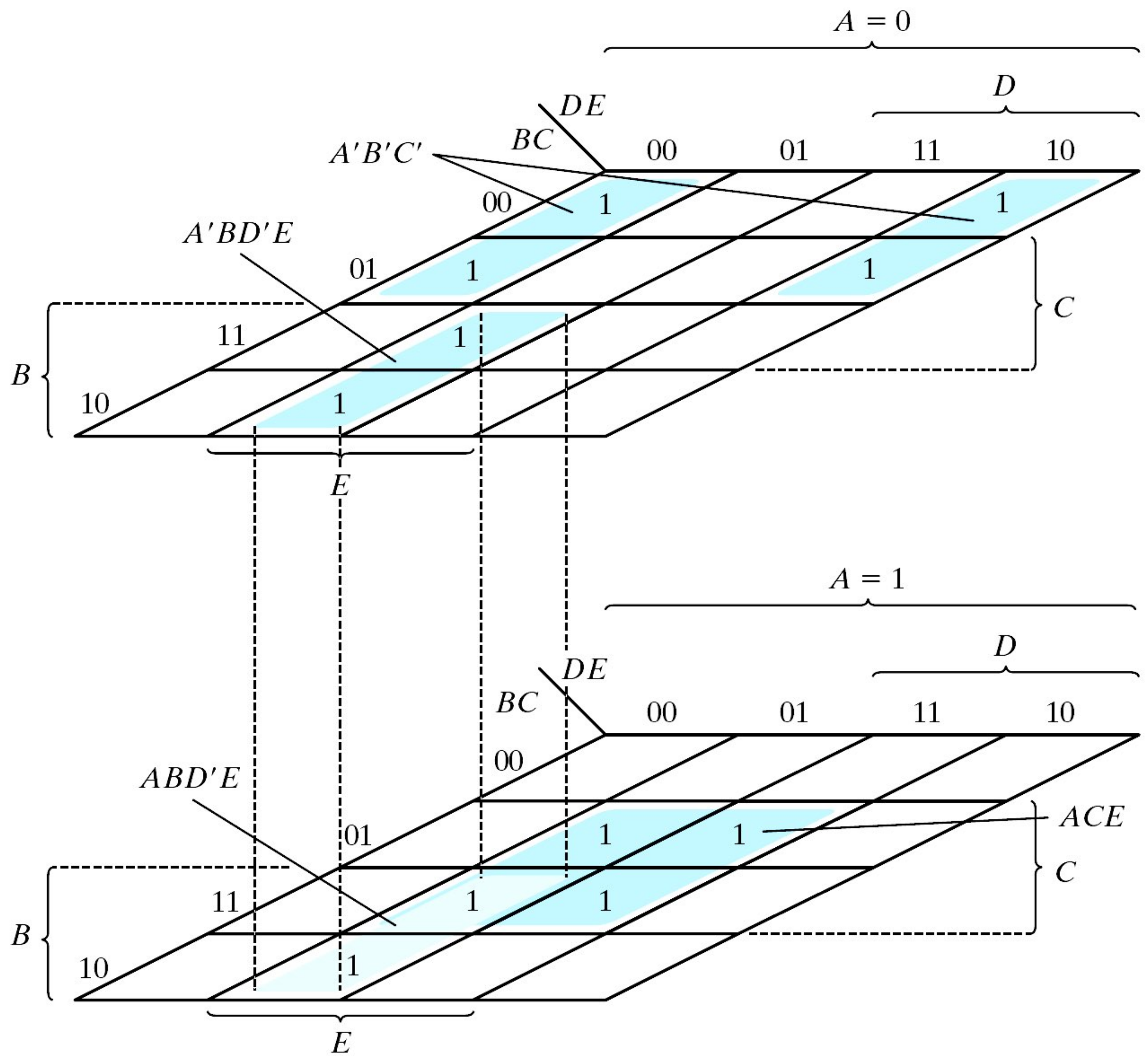
$A = 0$

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		

Five-Variable Map



K-map Summary

- Any 2^k adjacent squares, k=0,1,…,n, in an n-variable map represent an area that gives a product term of n-k literals

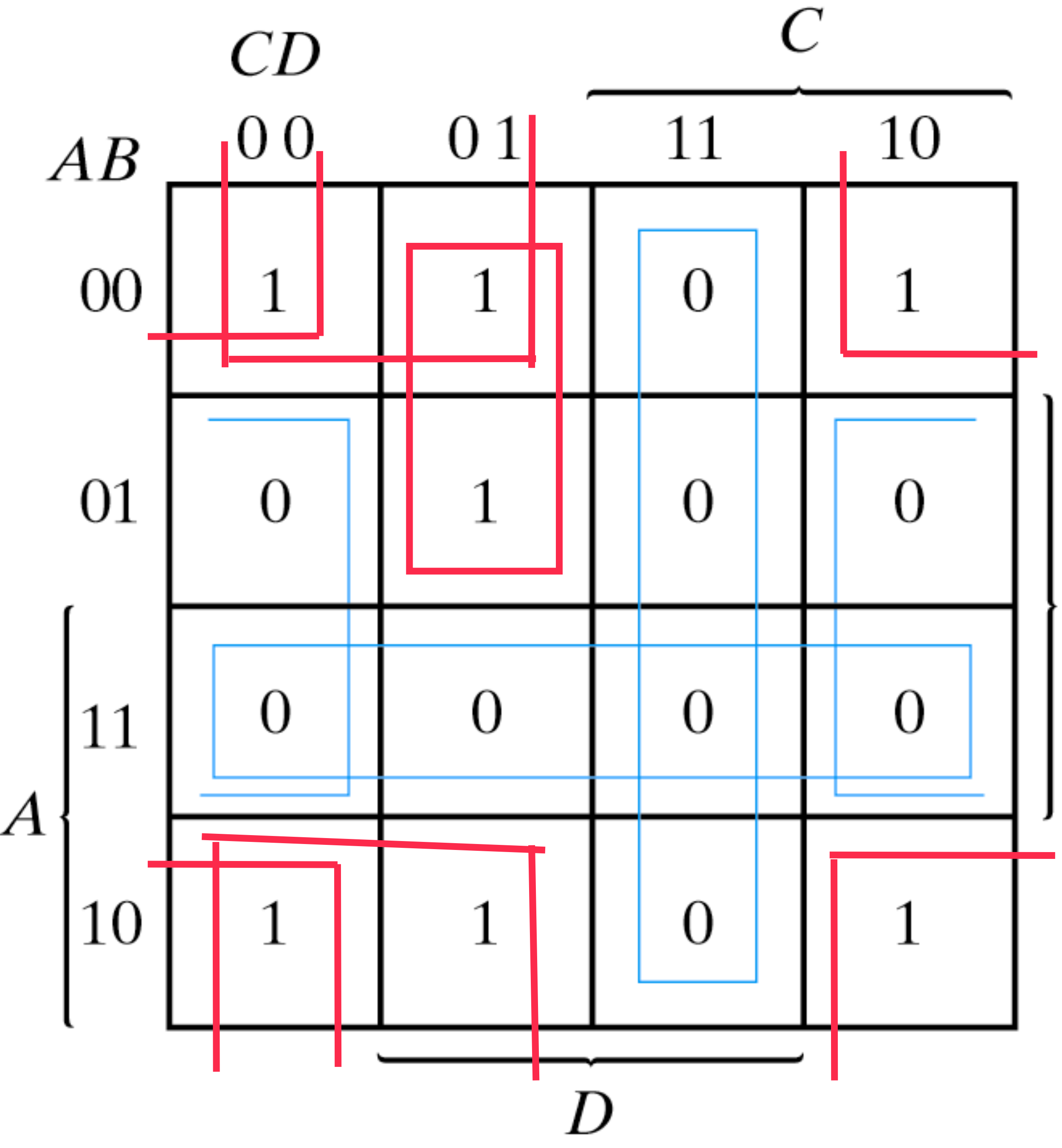
K	# of adjacent squares	# of literals in a term in an n-variable map			
		n=2	n=3	n=4	n=5
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0

Product-of-Sums Simplification

- Based on the generalized DeMorgan’s Theorem
  - (0′s in the K-map): Simplified F′ in the form of sum of products
  - (1′s in the K-map): Apply Demorgan’s Theorem F=(F′)′
    - F′: sum of products => F: product of sums

Product-of-Sums Simplification

- Example: Simplify F in sum-of-product and product-of-sum forms $F(A,B,C,D)=\sum(0,1,2,5,8,9,10)$



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

★sum-of-product (minterm approach)

$$F=B'D'+B'C'+A'C'D$$ **(specify 1's)**

★product-of-sum (DeMorgan's Theorem)

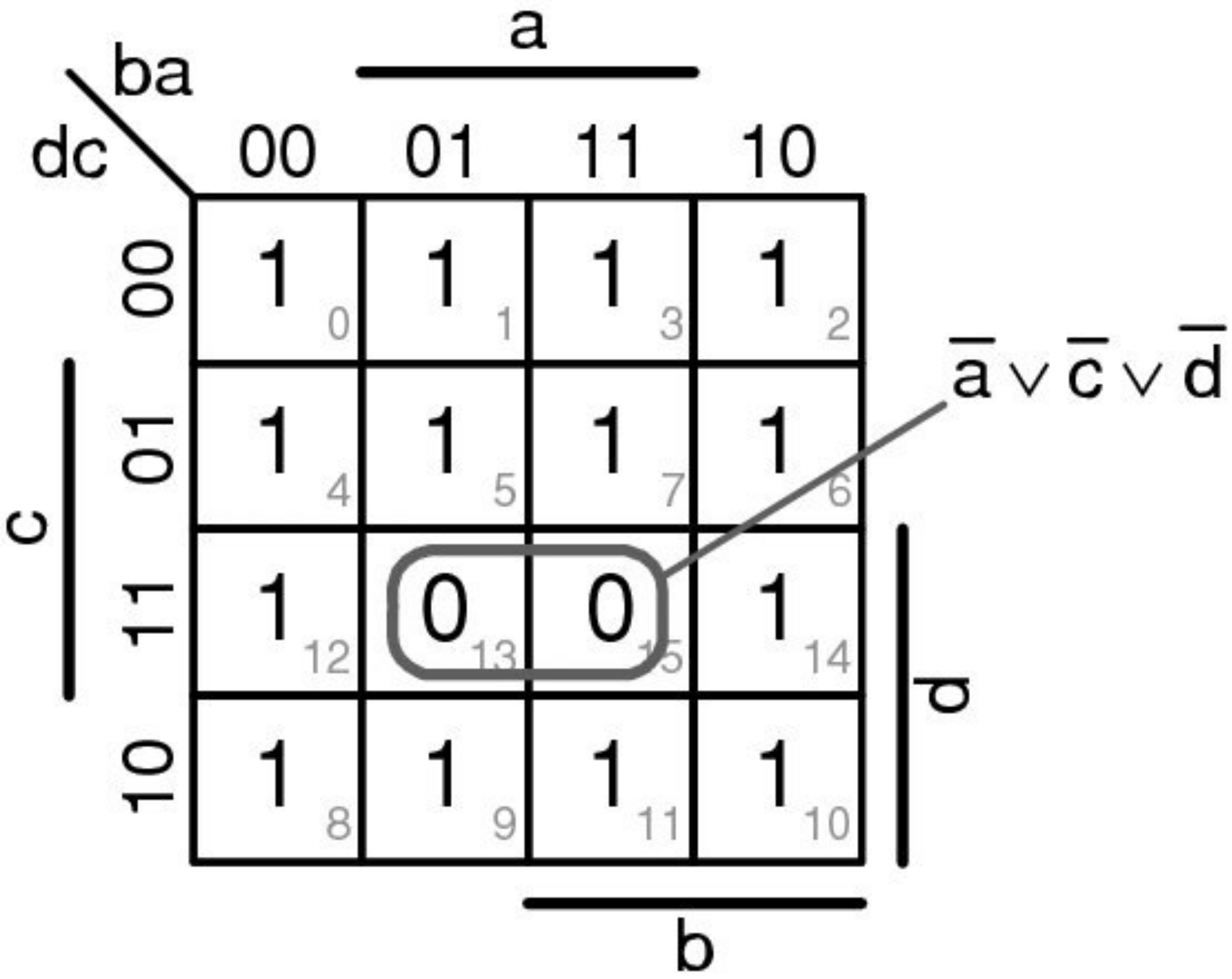
$$F'=AB+CD+BD'$$ **(specify 0's)**

Apply DeMorgan's Theorem

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

Example 1

$$f(d,c,b,a) = \Pi(13,15)$$



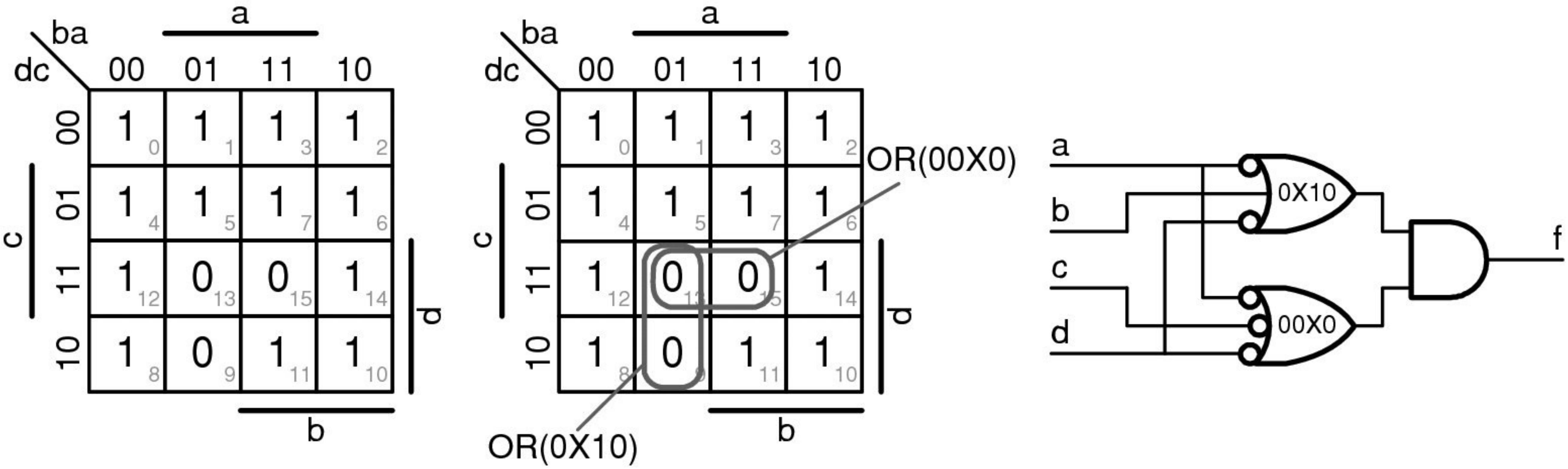
$$(f(d,c,b,a))' = acd$$

$$\Rightarrow$$

$$f(d,c,b,a) = a' + c' + d'$$

Example 2

$$f(d, c, b, a) = \Pi(9, 13, 15)$$



dc \ ba	00	01	11	10
00	1_{0}	1_{1}	1_{3}	1_{2}
01	1_{4}	1_{5}	1_{7}	1_{6}
11	1_{12}	0_{13}	0_{15}	1_{14}
10	1_{8}	0_{9}	1_{11}	1_{10}

(a)

(b)

(c)

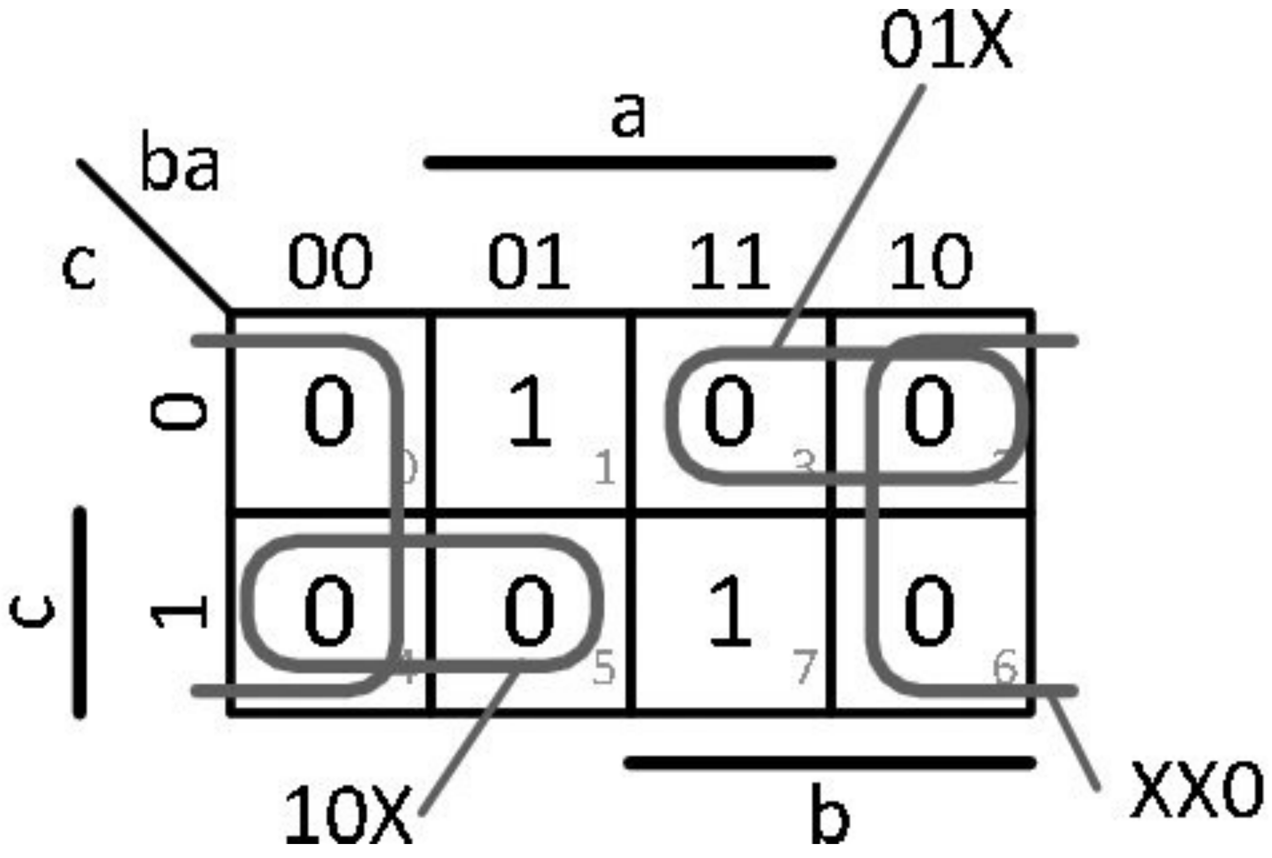
$$(f(d, c, b, a))' = db'a + dca$$

$$f(d, c, b, a) = (d' + b + a')(d' + c' + a')$$

Example 3

- Find a minimal product-of-sums expression

$$f(c,b,a) = \sum m(1,7)$$



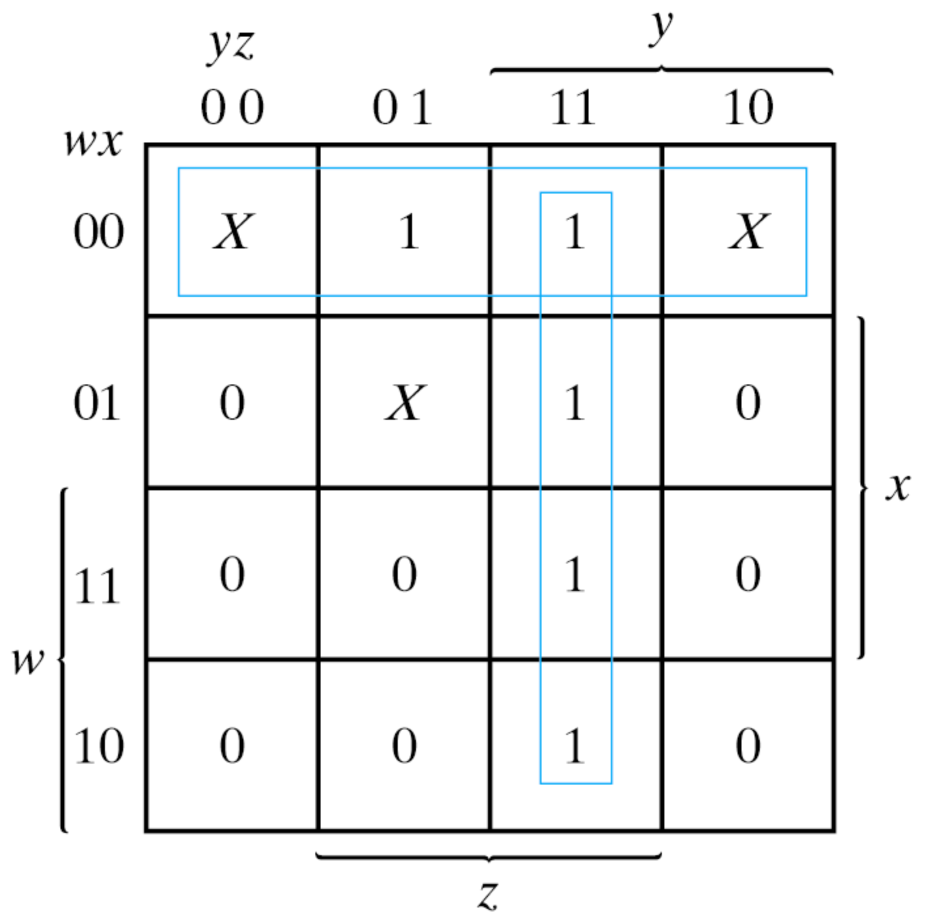
$$(f(c,b,a))' = c'b + cb' + a' \quad \Rightarrow \quad f(c,b,a) = (c+b')(c'+b)a$$

Don’t-Care Conditions

- Incompletely specified functions
  - Functions that have unspecified outputs for some input combinations
    - output are unspecified for 1010 to 1111 in 4-bit BCD code
- Don’t-care conditions
  - Unspecified minterms of a function, don’t-cares, Xs
  - Can be used on a map to provide further simplifications of the Boolean expression
  - Each X can be assigned an arbitrary value, 0 or 1, to help simplification procedure

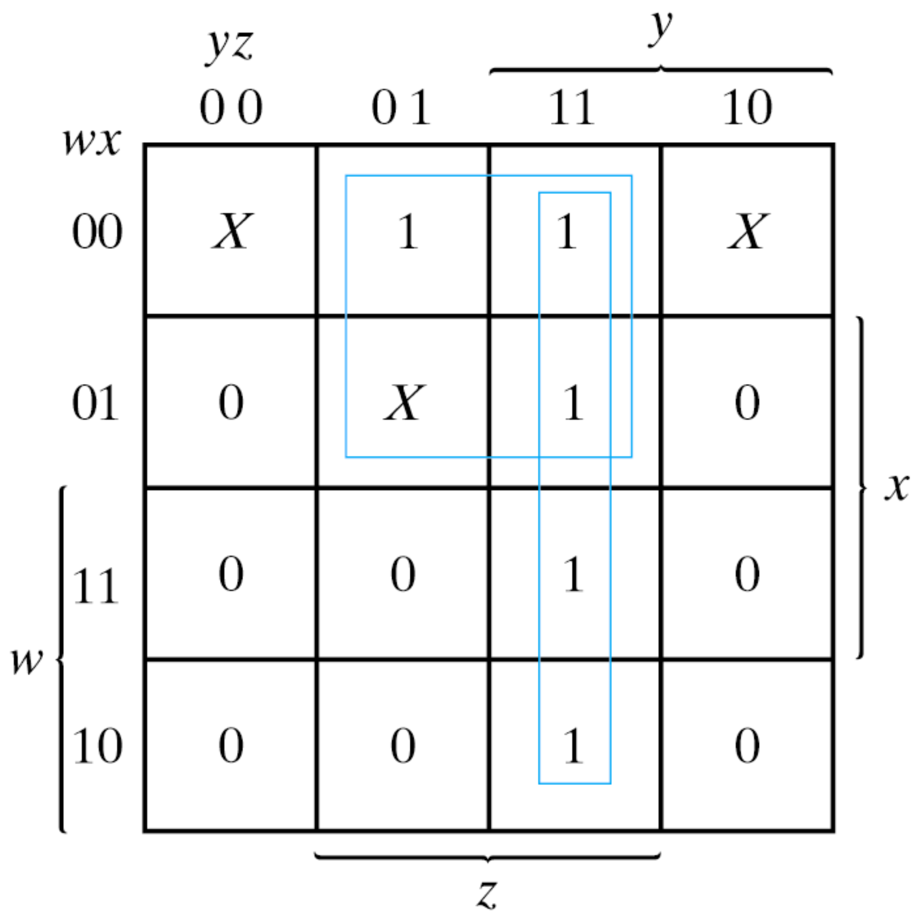
Example 3.8

- Example
  - Boolean function: $F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$
  - Don't-care conditions: $D(w, x, y, z) = \sum(0, 2, 5)$
  - both (a) an (b) are 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

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



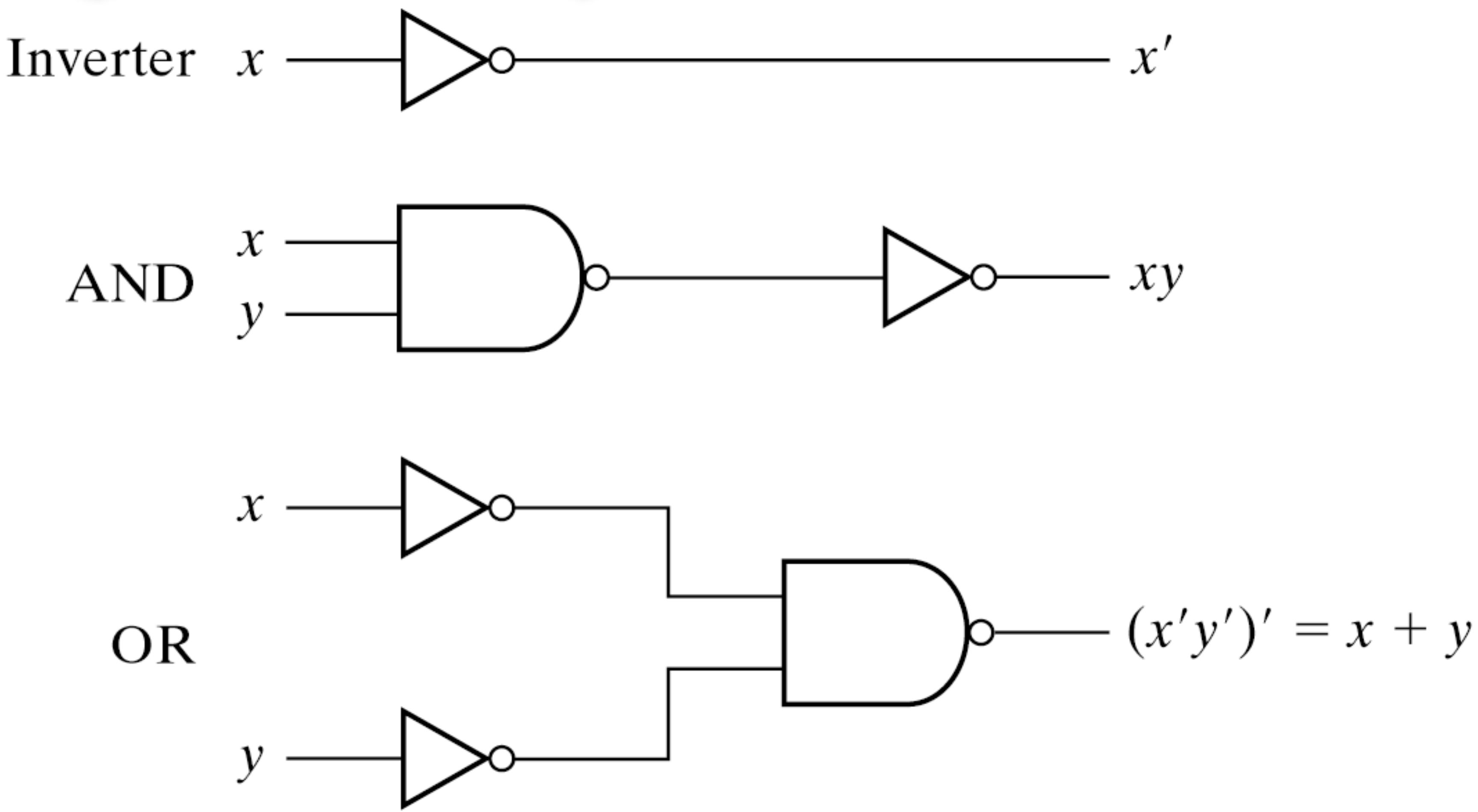
wx \ yz	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

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

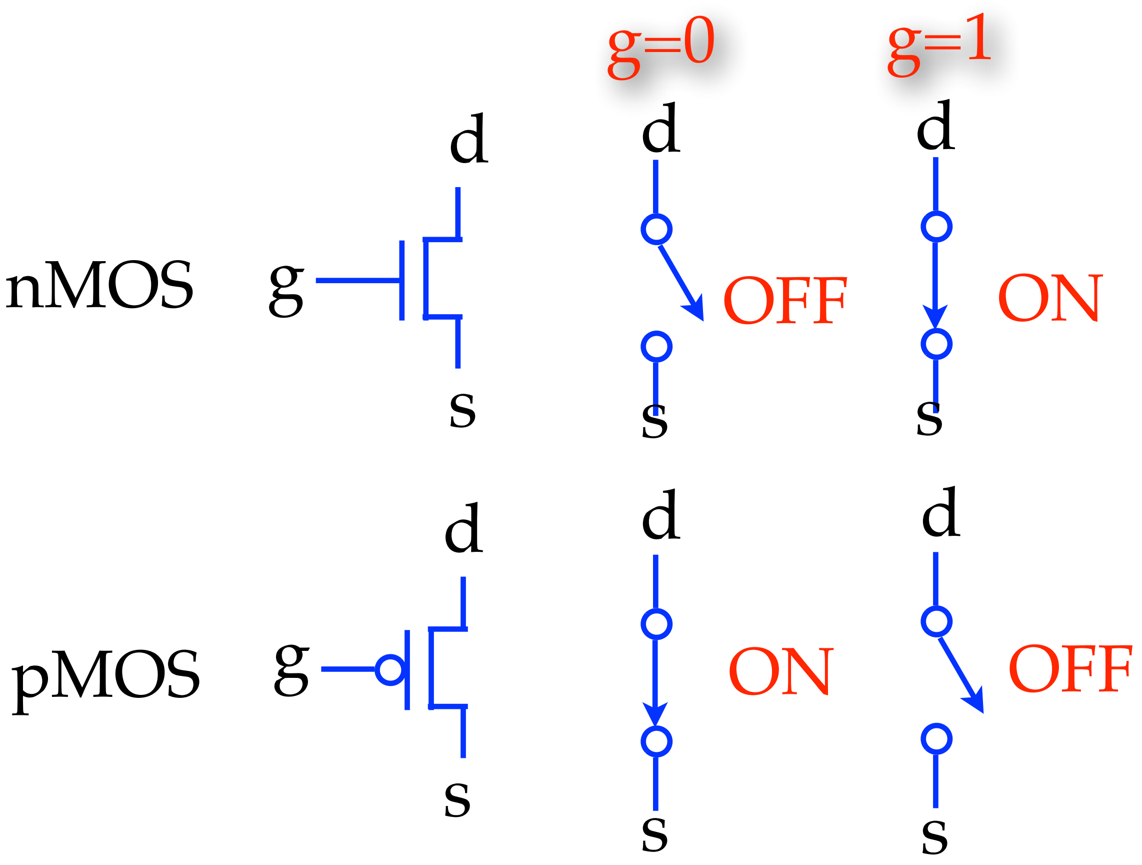
Technology Mapping

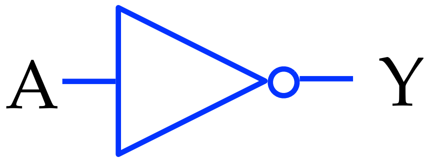
NAND and NOR Implementation

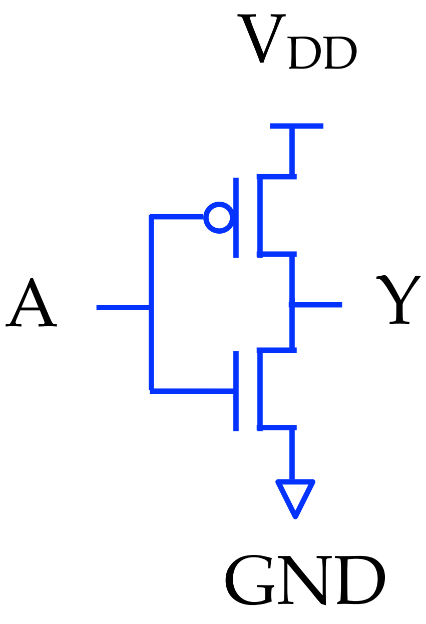
- Digital circuits are more frequently constructed with NAND/NOR gates than with AND/OR/NOT gates due to ease of fabrication.
  - In gate arrays, only NAND (or NOR) gates are used.
- NAND gate is a universal gate because any operation can be implemented by it.

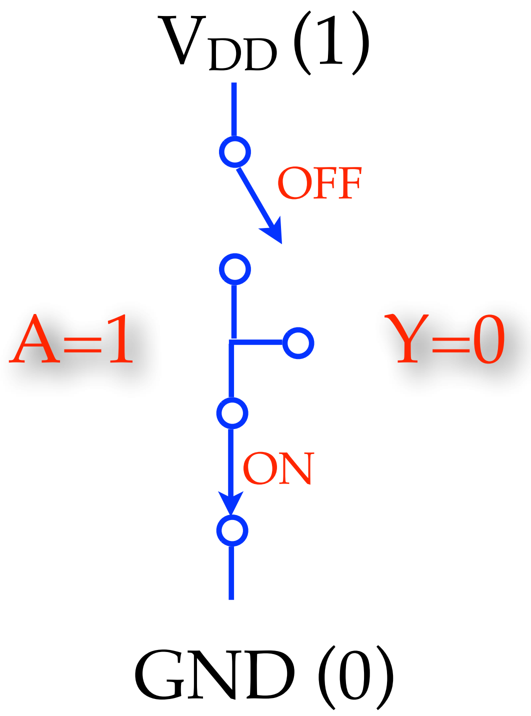


MOS Switches

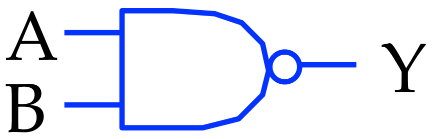


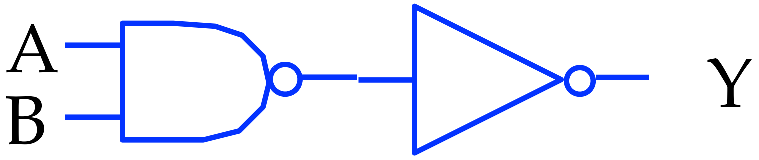


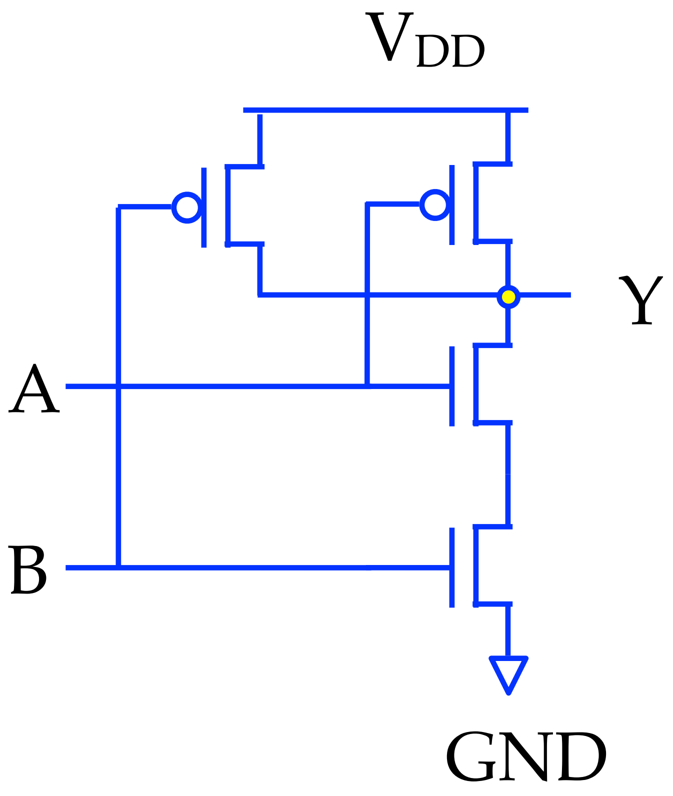


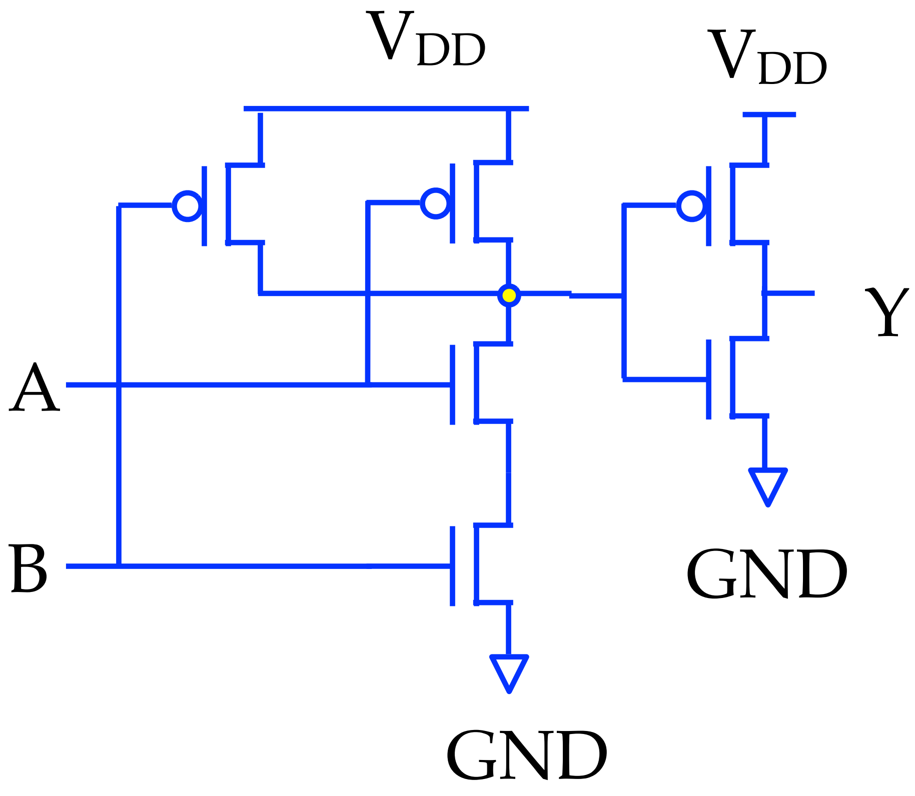


NAND vs. AND



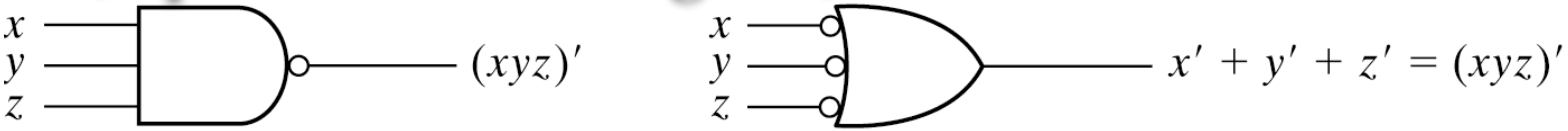






NAND-NAND Implementation (1/6)

- AND-invert and Invert-OR are equivalent. (Equivalent NAND gates)



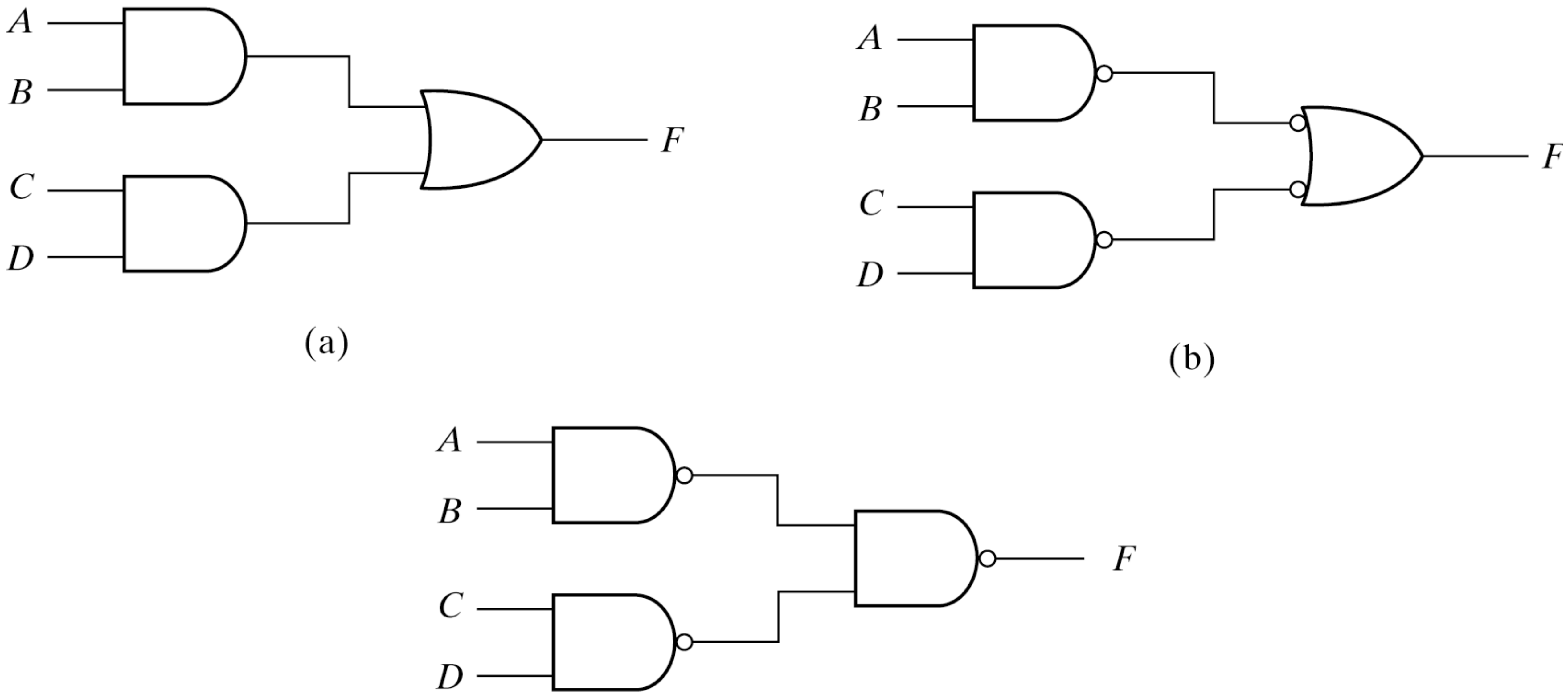
(a) AND–invert

(b) Invert–OR

- Procedure
  - Simplify the function in the form of sum-of-products.
  - Transfer it to 2-level NAND-NAND expression (DeMorgan's Theorem).
  - Draw the corresponding NAND gate implementation. A 1-input NAND gate can be replaced by an inverter.

NAND-NAND Implementation (2/6)

- Example $F(A,B,C,D) = AB + CD$



(a)

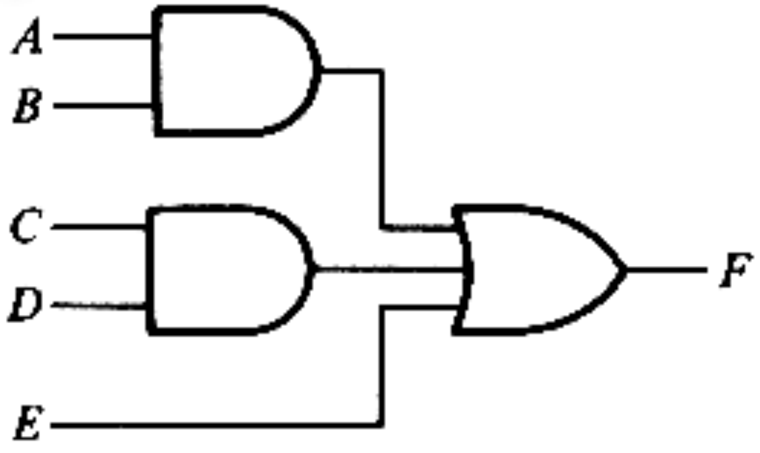
(b)

(c)

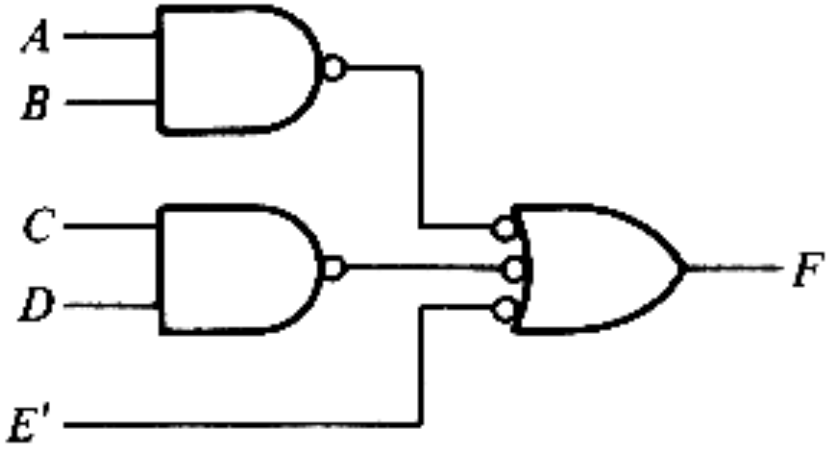
NAND-NAND Implementation (3/6)

- Example

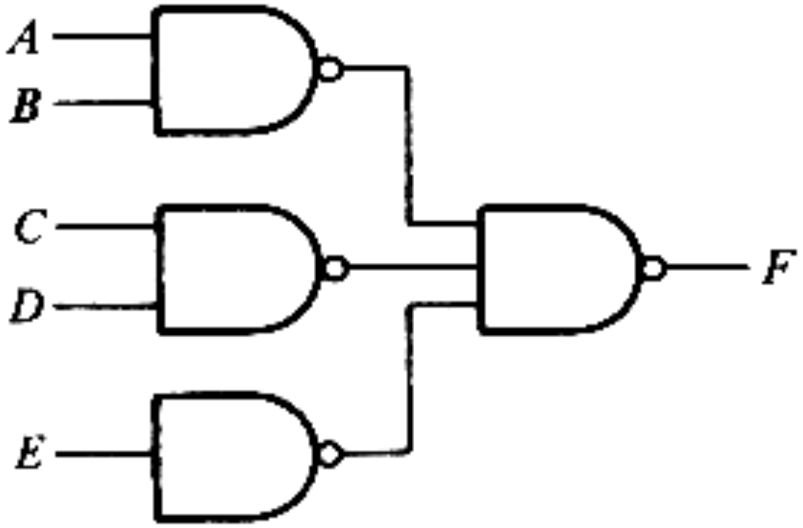
$$F(A,B,C,D,E) = AB + CD + E = ((AB)'(CD)'E')'$$



(a) AND-OR



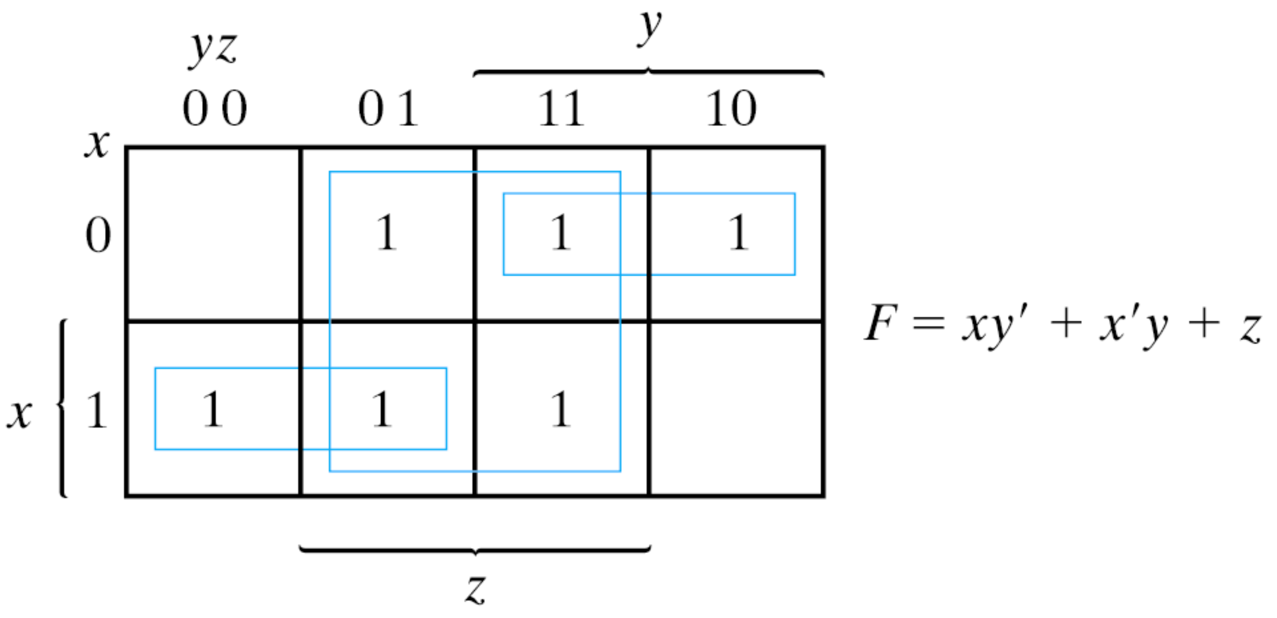
(b) NAND-NAND



(c) NAND-NAND

NAND-NAND Implementation (4/6)

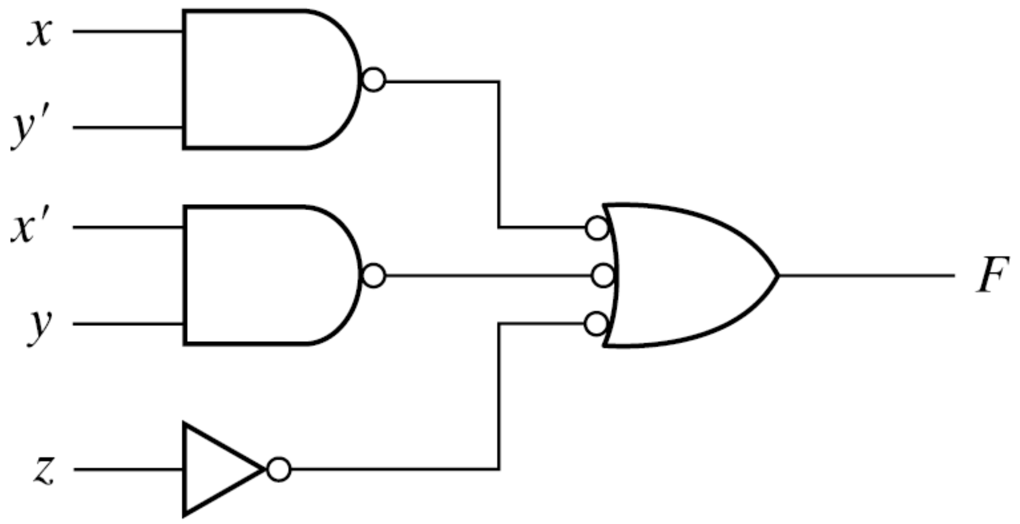
- Example $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$



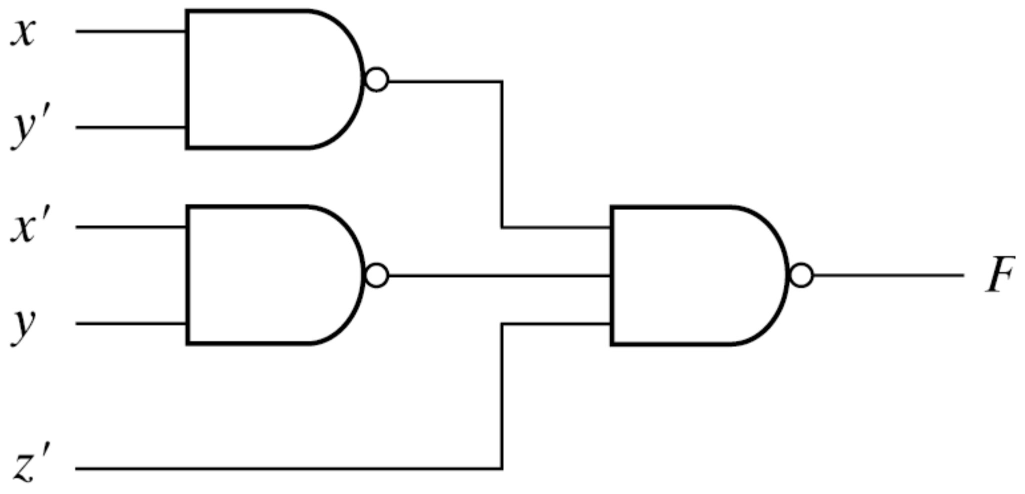
x \ yz	00	01	11	10
0		1	1	1
1	1	1	1	

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

(a)



(b)



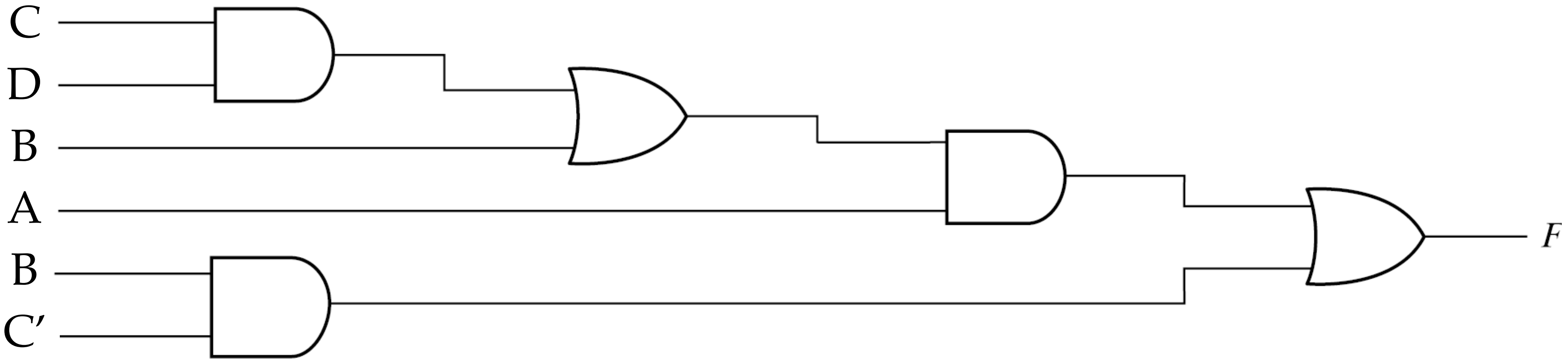
(c)

NAND-NAND Implementation (5/6)

- Multilevel-NAND circuits conversion procedure
  - Convert all AND gates to NAND gates with AND-Invert graphic symbols
  - Convert all OR gates to NAND gates with Invert-OR graphic symbols
  - Check all the bubbles (inverter) in the diagram and insert possible inverter to keep the original function

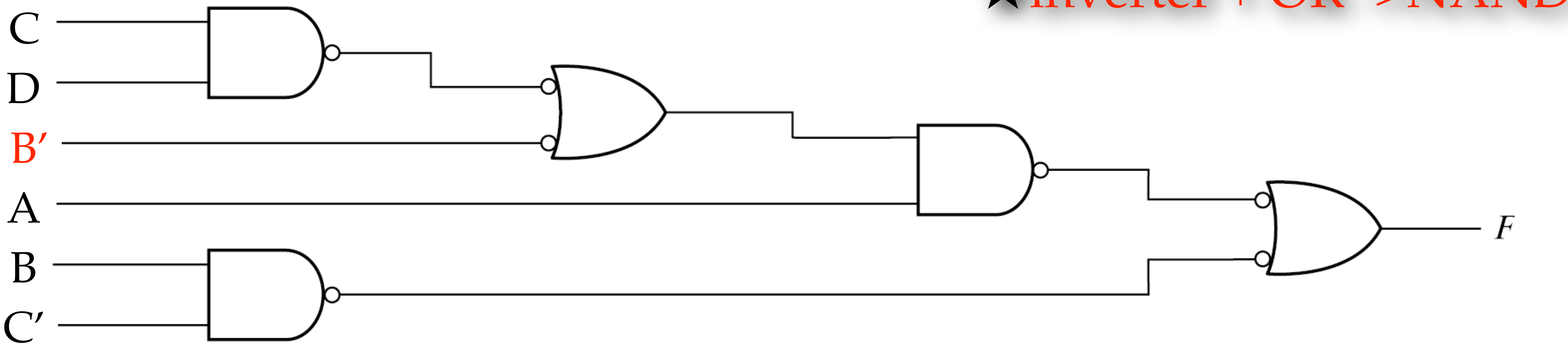
NAND-NAND Implementation (6/6)

- Multilevel NAND example
  - F(A,B,C,D)=A(CD+B)+BC′
  - AND-OR logic -> NAND-NAND logic



(a) AND-OR gates

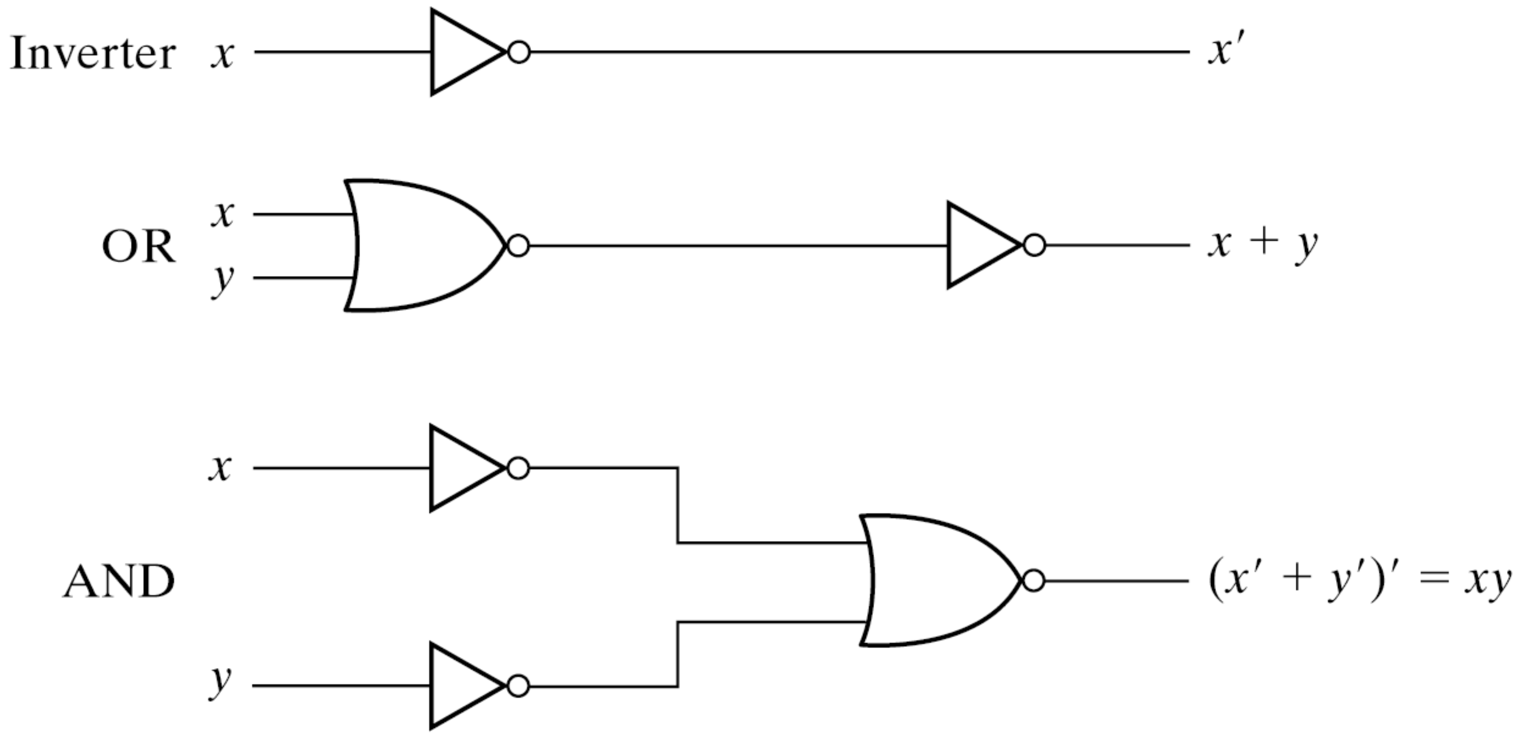
★AND->NAND + inverter
★inverter + OR ->NAND

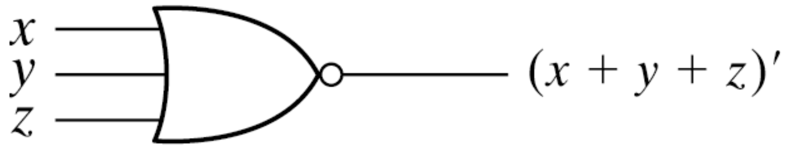


(a) NAND gates

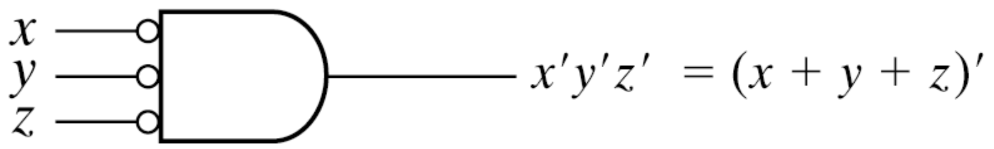
NOR-NOR Implementation (1/2)

- NOR-NOR is the dual of the NAND-NAND implementation
  - AND-OR => NAND-NAND
  - OR-AND => NOR-NOR





(a) OR–invert



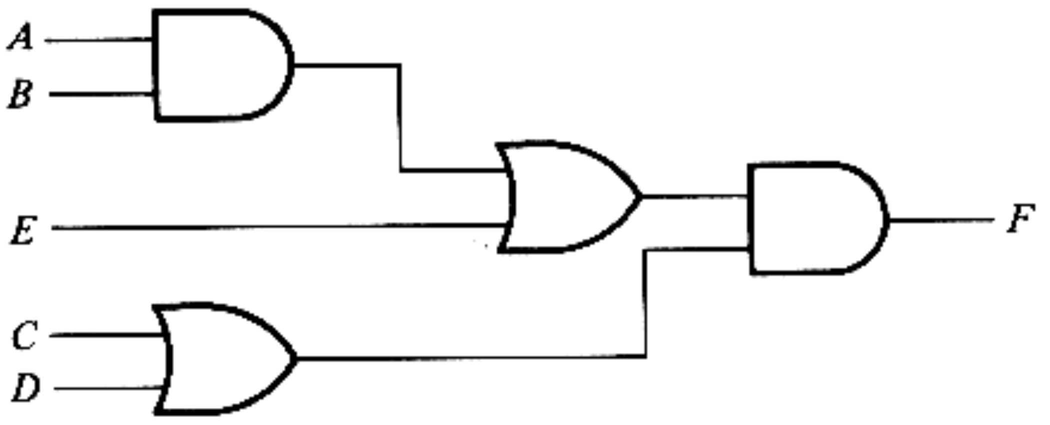
(a) Invert–AND

NOR equivalent gates

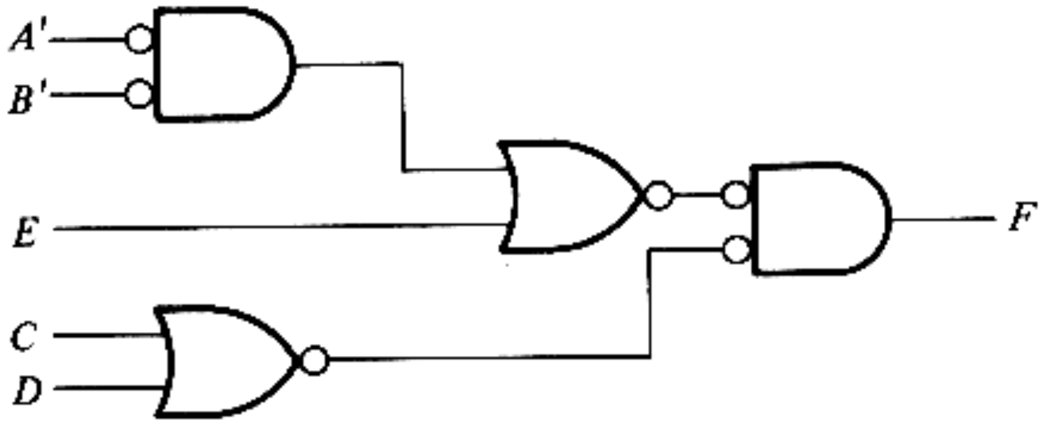
NOR-NOR Implementation (2/2)

- Example

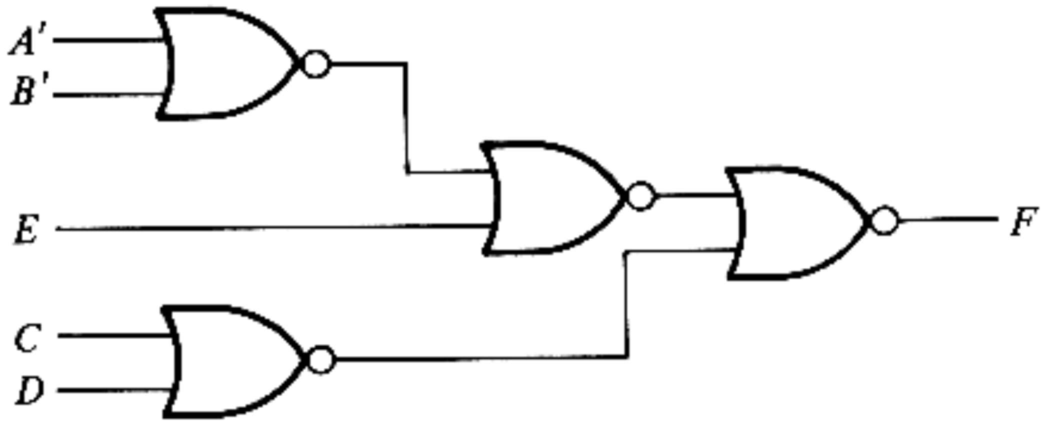
$$F(A,B,C,D,E)=(AB+E)(C+D)$$



(a) AND-OR diagram



(b) NOR diagram



(c) Alternate NOR diagram

Two-level Forms (1/2)

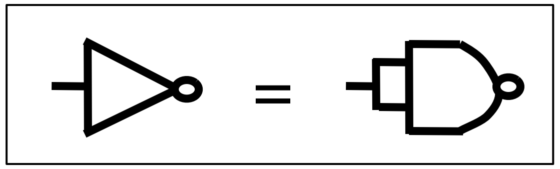
- AND/NAND/OR/NOR have 16 possible combinations of two-level forms
- Eight of them degenerate to a single operation
  - AND-AND => AND
  - OR-OR => OR
  - AND-NAND => NAND
  - OR-NOR => NOR
  - NAND-NOR =>AND
  - NOR-NAND => OR
  - NAND-OR => NAND
  - NOR-AND => NOR

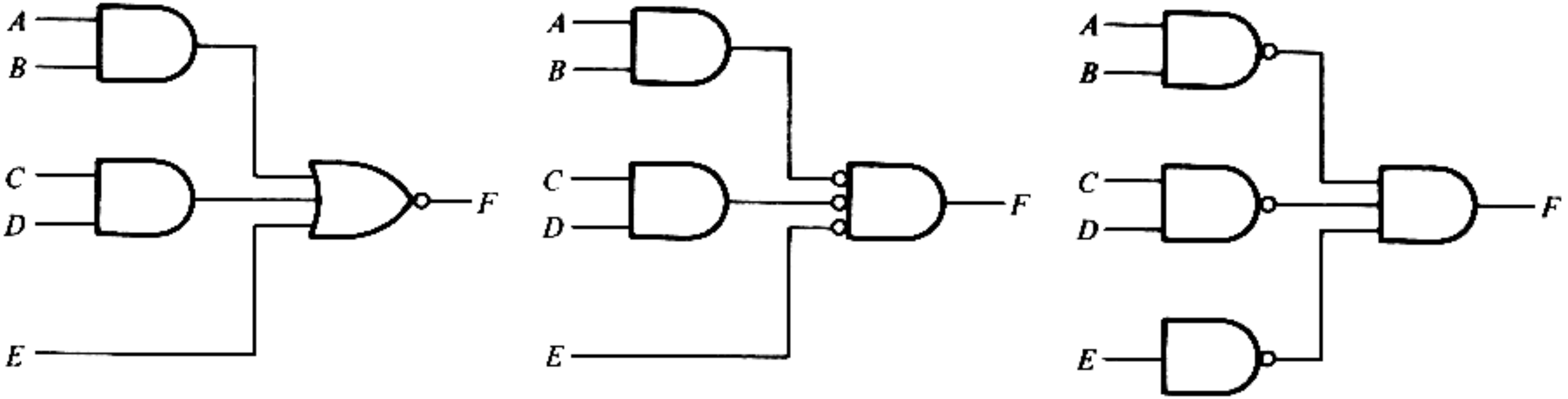
Two-level Forms (2/2)

- Eight are non-degenerate forms
  - AND-OR => standard sum-of-products
  - NAND-NAND => standard sum-of-products
  - OR-AND => standard product-of-sums
  - NOR-NOR => standard product-of-sums
  - NAND-AND/AND-NOR => AND-OR-INVERT (AOI)
    - complement of sum-of-products
  - OR-NAND/NOR-OR => OR-AND-INVERT (OAI)
    - complement of product-of-sums

AND-OR-INVERT Circuits

- NAND-AND = AND-NOR = AOI
- $F(A,B,C,D,E)=(AB+CD+E)'$
- $F'(A,B,C,D,E)=AB+CD+E$



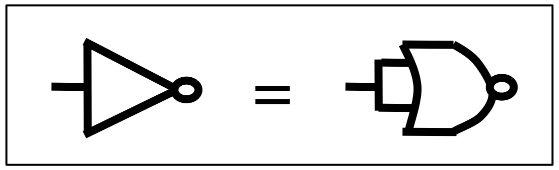


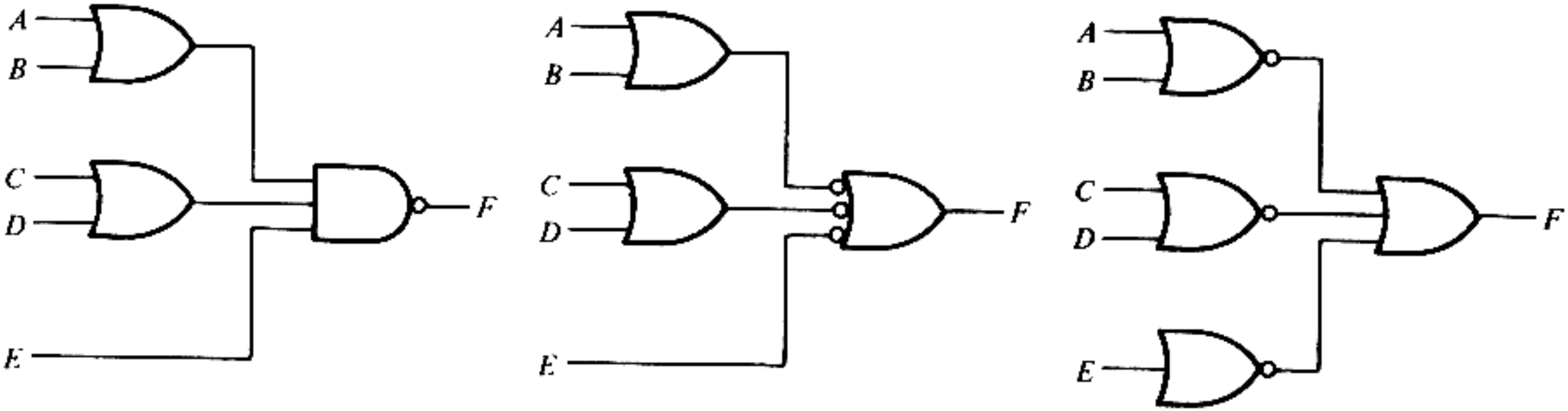
(a) AND-NOR (b) AND-NOR (c) NAND-AND

Combine 0's in K-map to simplify F' in sum-of-products

OR-AND-INVERT Circuits

- OR-NAND = NOR-OR = OAI
- F(A,B,C,D,E)=((A+B)(C+D)E)′



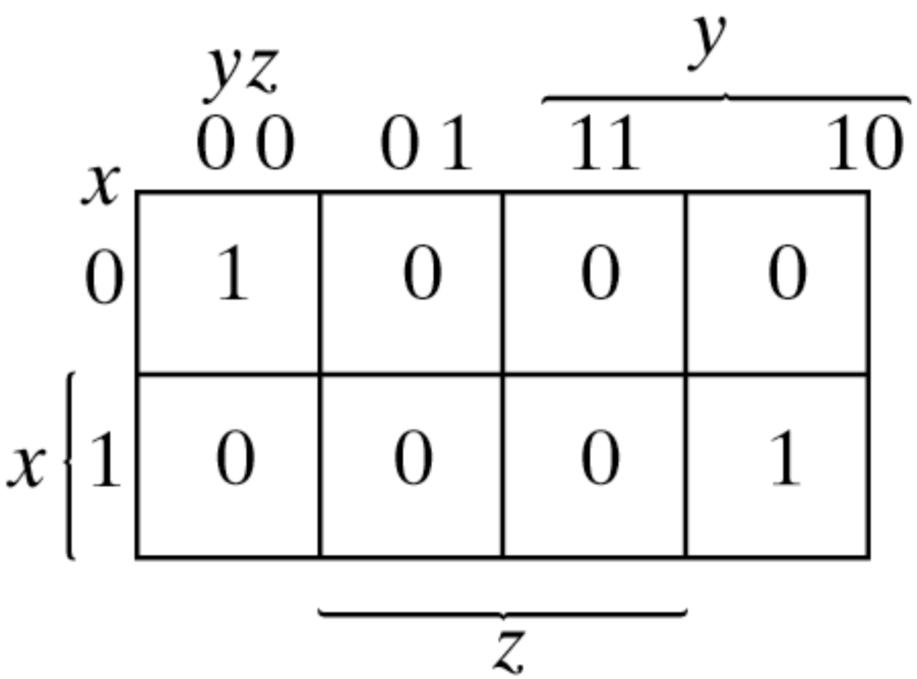


(a) OR-NAND (b) OR-NAND (c) NOR-OR

Combine 1′s in K-map to simplify F′ in product-of-sums and then invert the results

AOI & OAI Implementation (1/2)

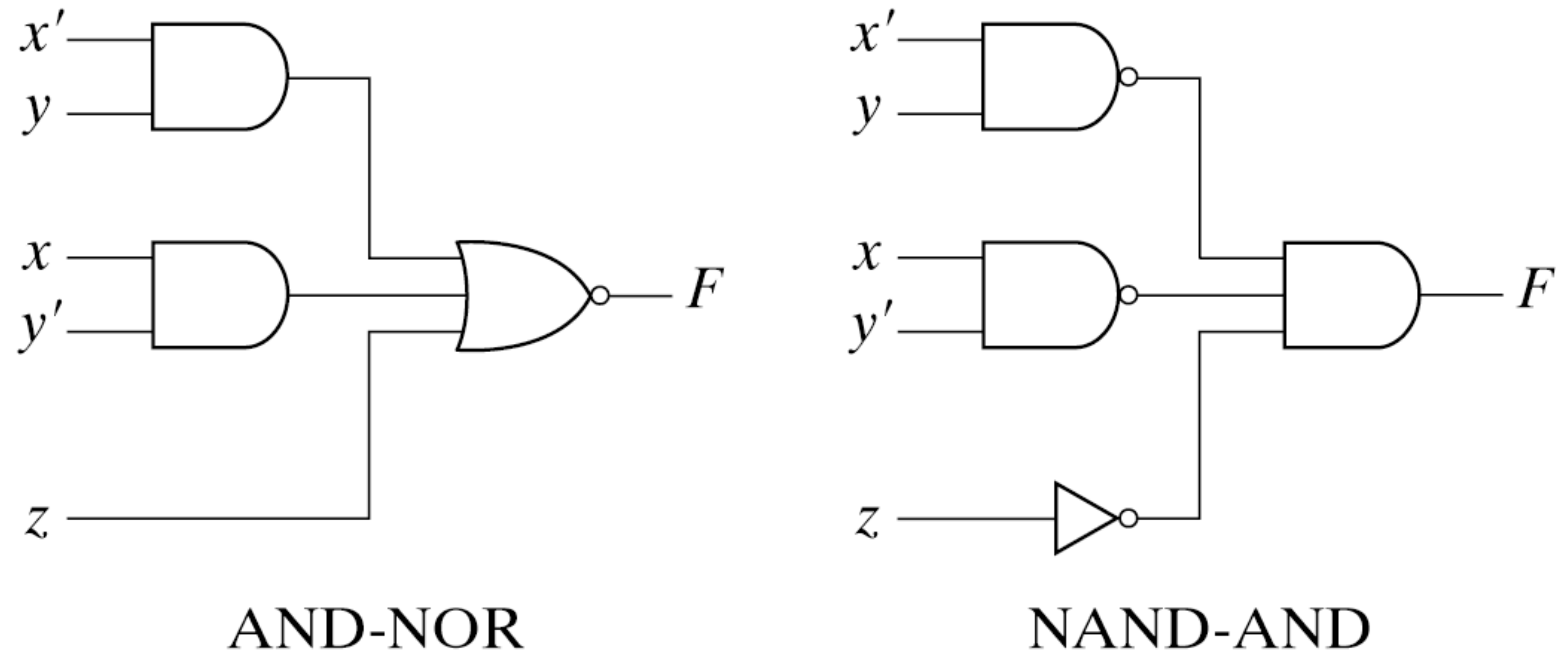
- Example
  - AOI implementation
    - F′=x′y+xy′+z (F′: sop of 0′s) => F=(x′y+xy′+z)′
  - OAI implementation
    - F=x′y′z′+xyz′ (F: sop of 1′s) => F′=(x+y+z)(x′+y′+z) => F=((x+y+z)(x′+y′+z′))′



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

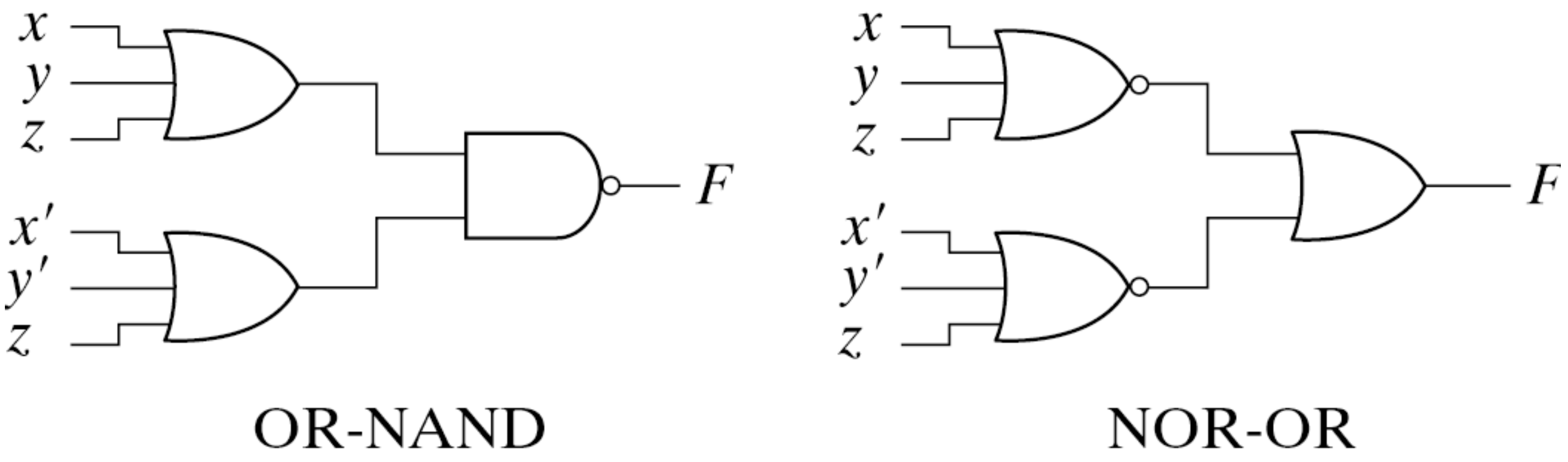
$$F = x'y'z' + xyz'$$
$$F' = x'y + xy' + z$$

AOI & OAI Implementation (2/2)



AND-NOR NAND-AND

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



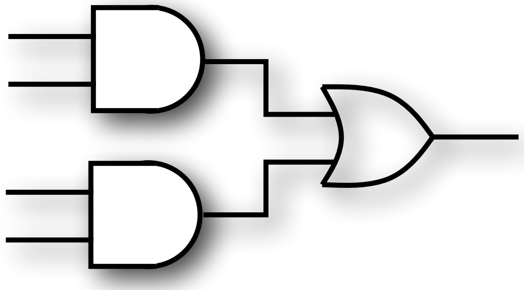
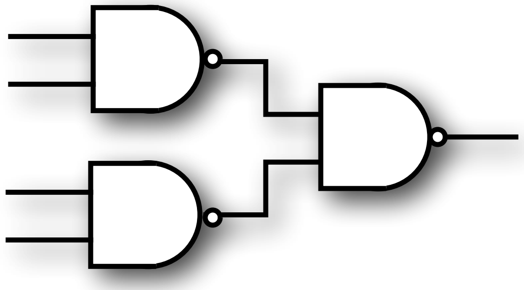
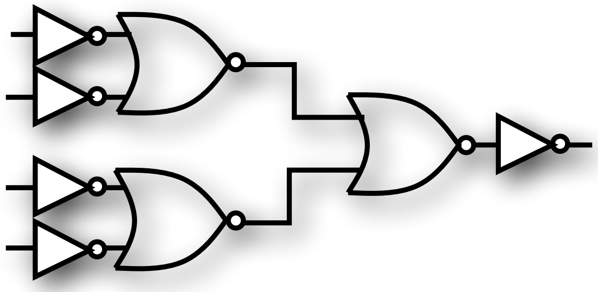
OR-NAND NOR-OR

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

Technology Mapping (1/2)

- The conversion process from an expression/schematic with AND, OR, and NOT gates to one with only NAND or NOR gates
  - Rule 1: $xy = ((xy)')'$ (NAND-Invert)
  - Rule 2: $x+y = ((x+y)')' = (x'y')'$ (Invert-NAND)
  - Rule 3: $xy = ((xy)')' = (x'+y')'$ (Invert-NOR)
  - Rule 4: $x+y = ((x+y)')'$ (NOR-Invert)

Technology Mapping (2/2)

	Standard form	NAND implementation	NOR implementation
sum–of–products			
product–of–sums