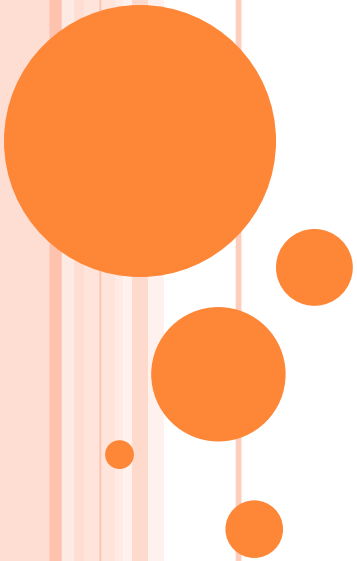


CSE 205: Digital Logic Design

Logic Minimization

- The complexity of the digital logic gates
  - The complexity of the algebraic expression

- Algebraic approaches:
  - The procedure of simplifying Boolean expressions is difficult since it lacks specific rules to predict the successive steps in the simplification process.

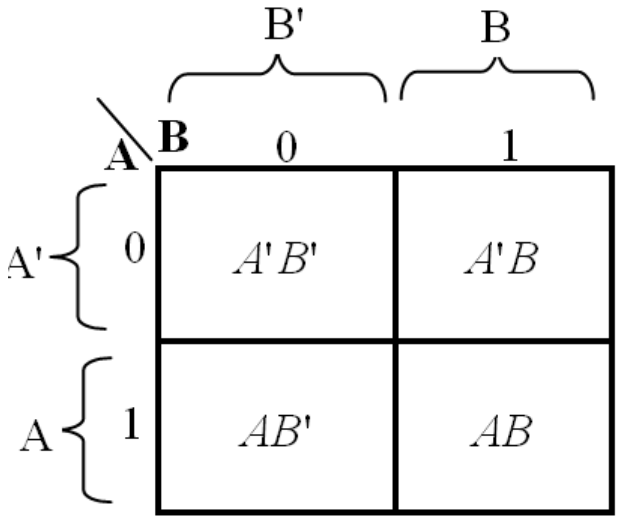
The Map Method

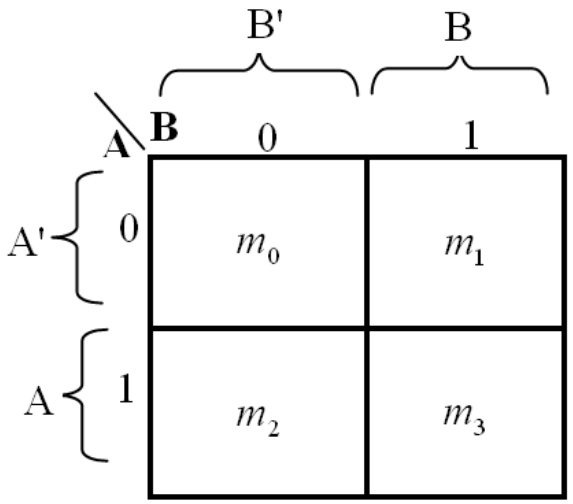
- The Karnaugh map (K-Map)
  - A simple straight forward procedure
  - A pictorial form of a truth table

- A diagram made up of squares
  - Each square represents one minterm

Two-Variable Map

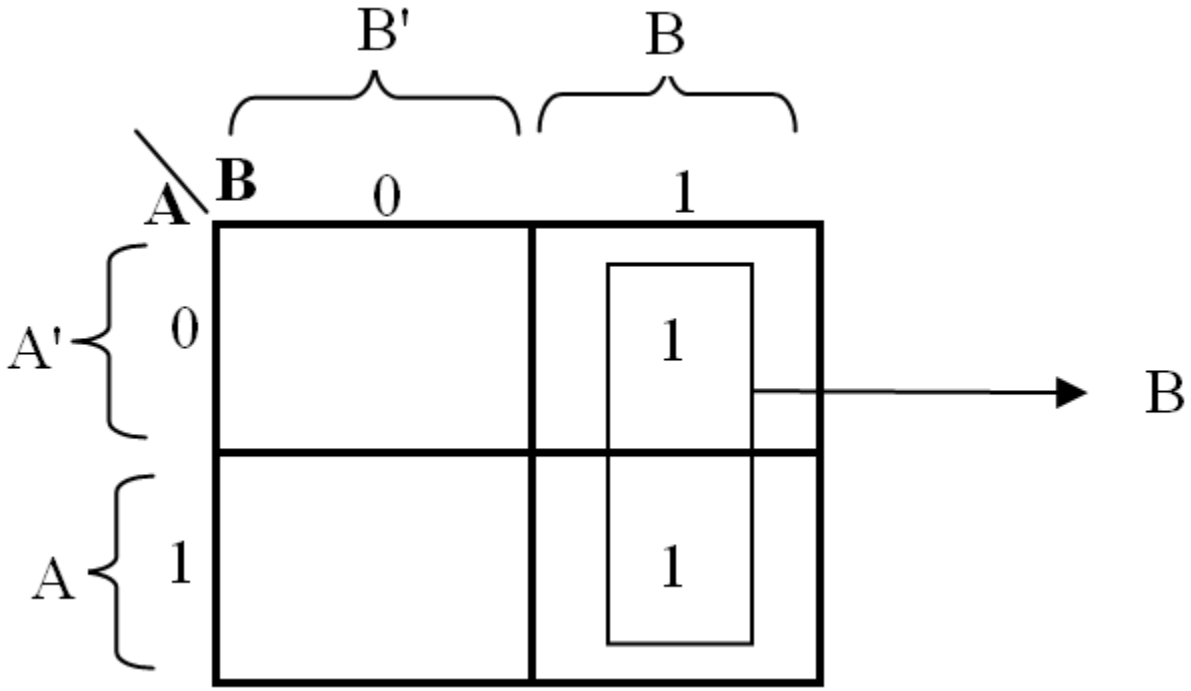
- Four minterms
- A' = row 0; A = row 1
- B' = column 0; B = column 1





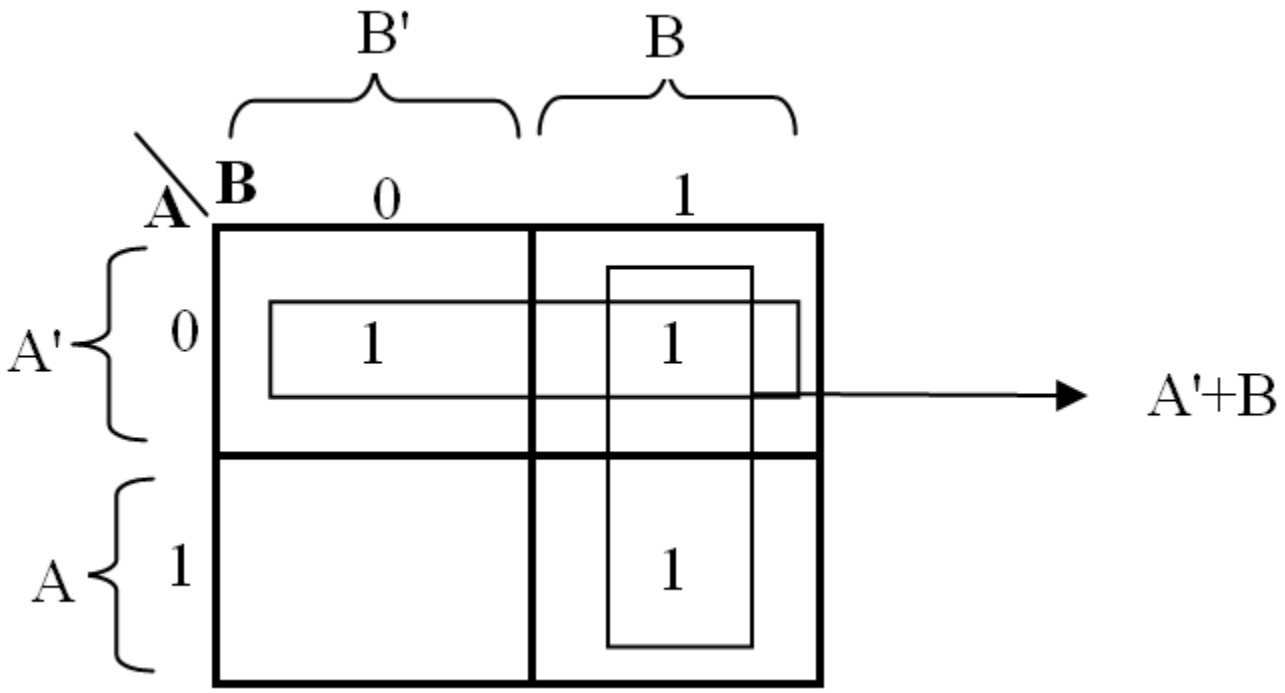
Example

$$F = AB + A'B = B$$



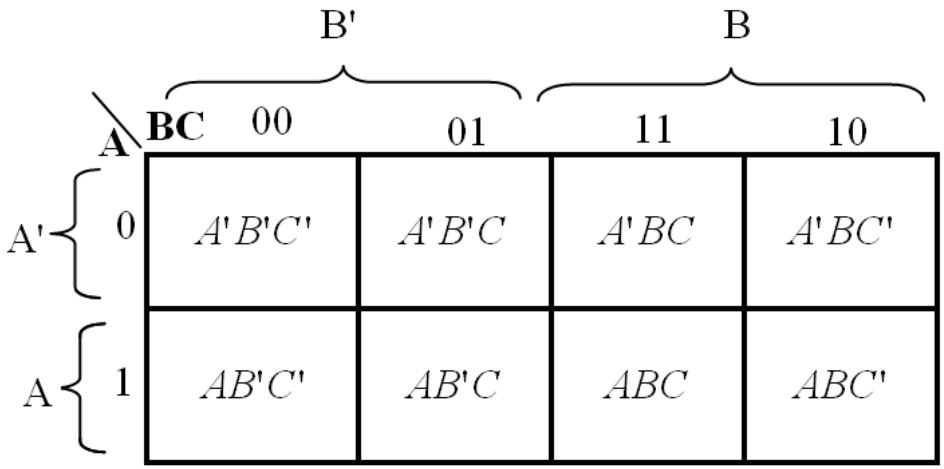
Example

$$F = AB + A'B + A'B' = A' + B$$



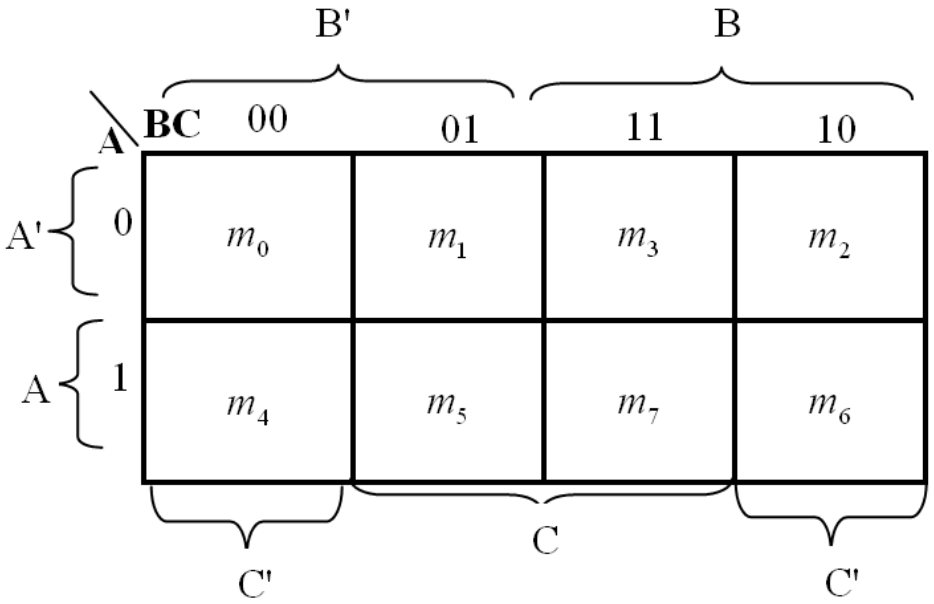
Three-Variable Map

- Eight minterms
- The Gray code sequence
- Any two adjacent squares in the map differ by only on variable



Three-Variable Map

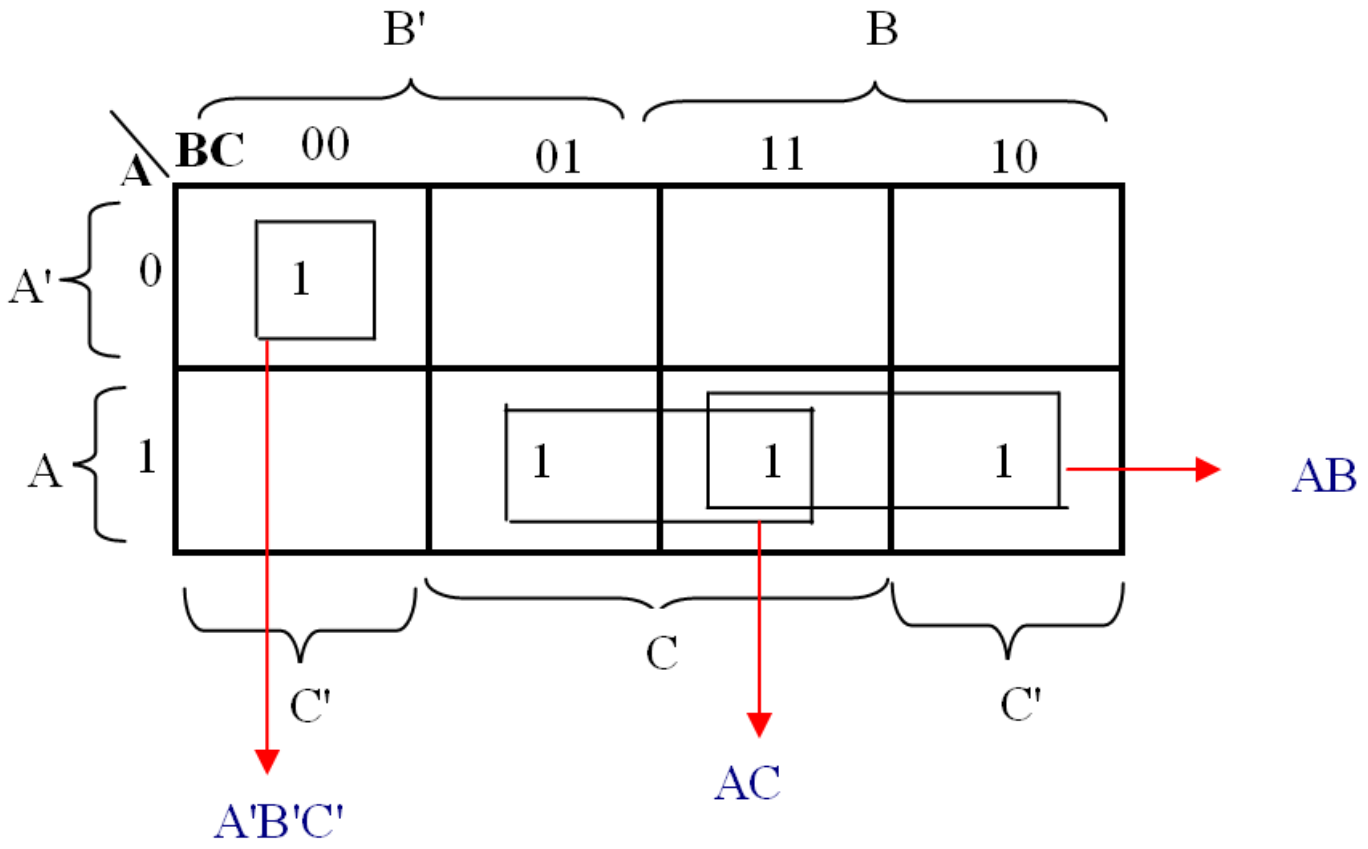
- m_0 and m_2 (m_4 and m_6) are adjacent
- $m_0 + m_2 = A'B'C' + A'BC' = A'C'(B' + B) = A'C'$
- $m_4 + m_6 = AB'C' + A'BC' = AC'(B' + B) = AC'$



EXAMPLE

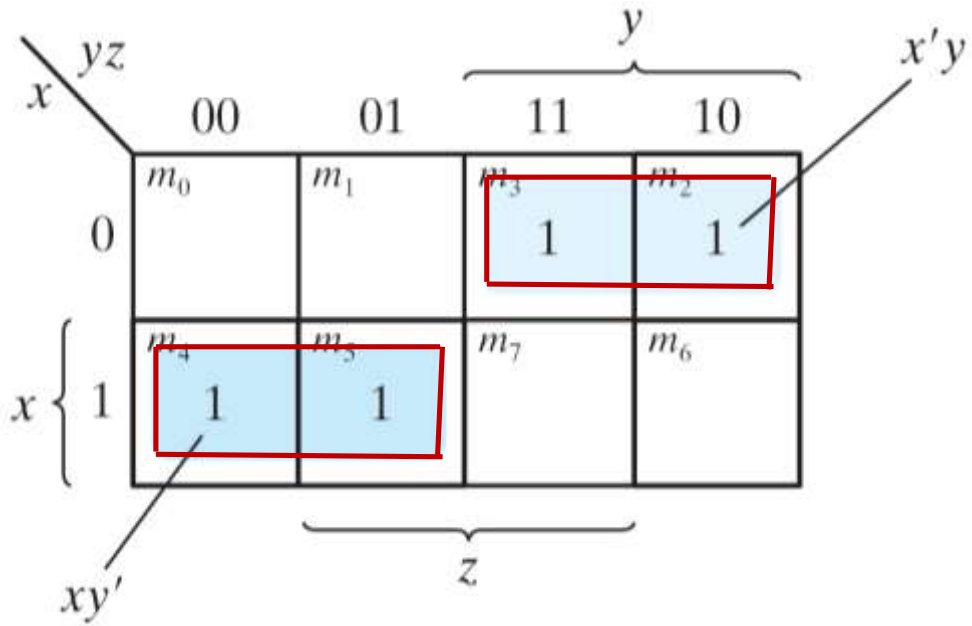
$$F = ABC' + A'B'C' + AB'C + ABC$$

$$= AB + AC + A'B'C'$$



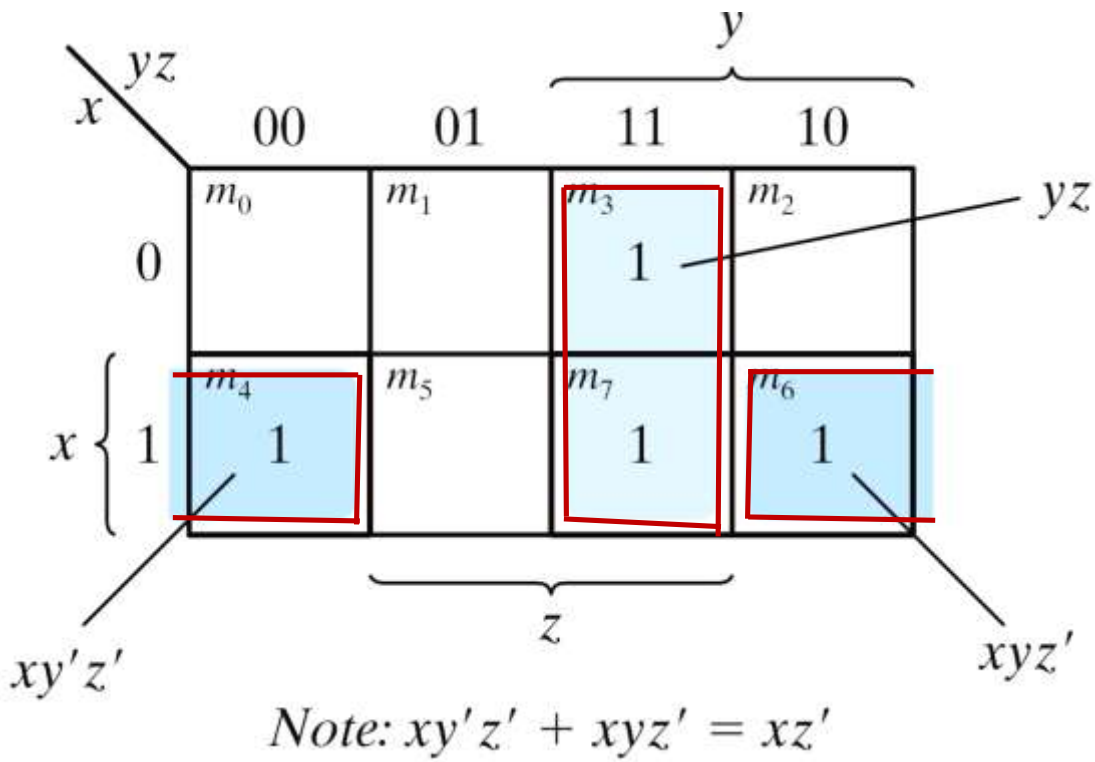
Example

- Simplify the Boolean function F(x, y, z)= $\Sigma(2, 3, 4, 5)$
  - $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$



EXAMPLE

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



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

Four adjacent Squares

- Consider four adjacent 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)

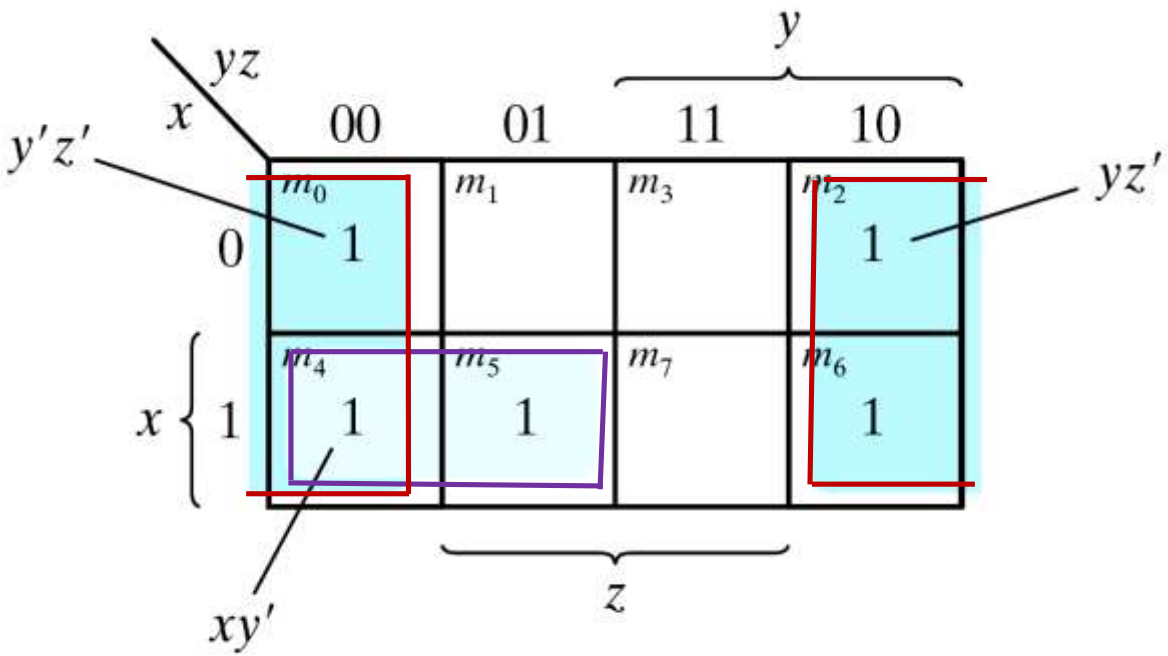
x \ xz	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

y spans 11 and 10; z spans 01 and 11; x labels rows 0 and 1.

(b)

Example

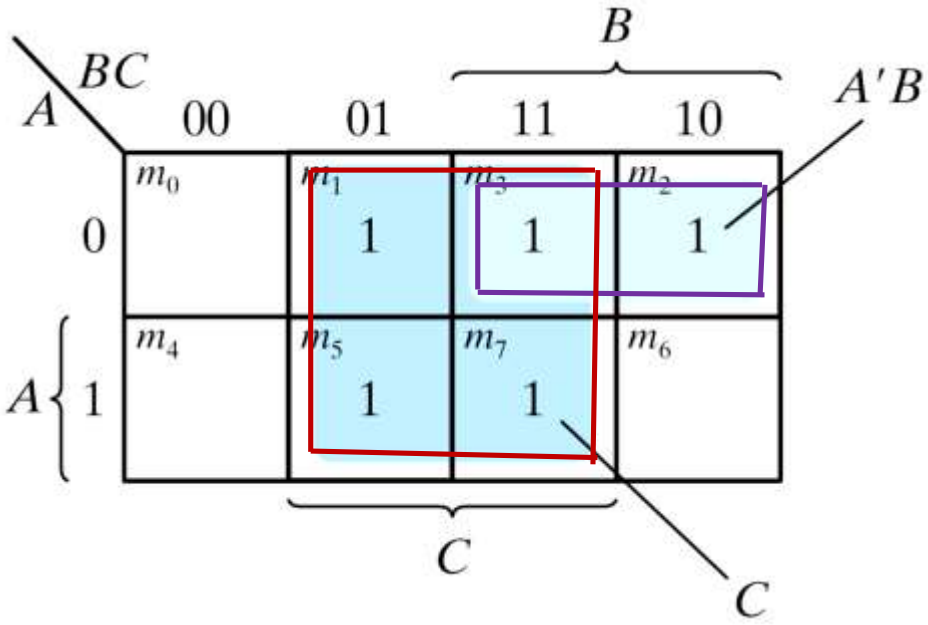
- Simplify $F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$
  - $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$



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

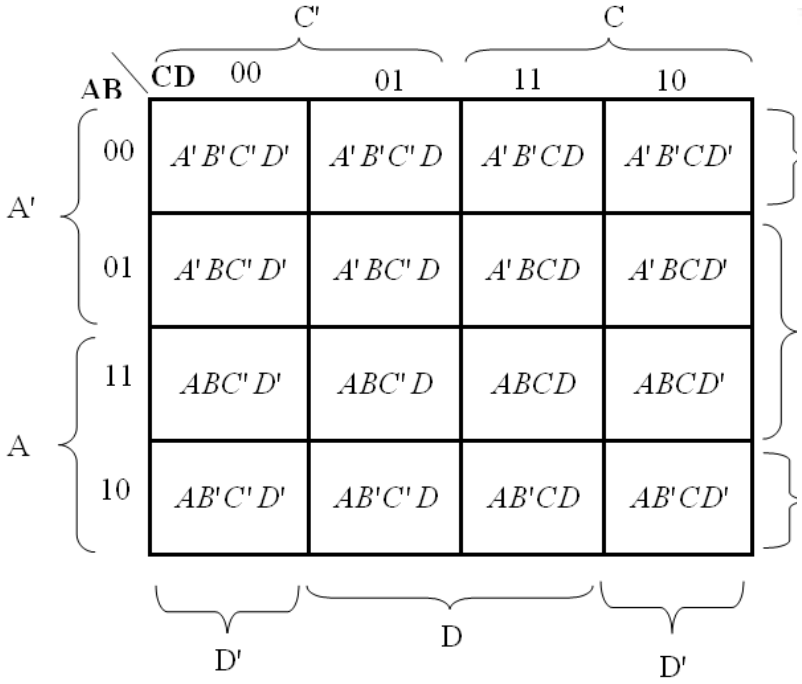
EXAMPLE

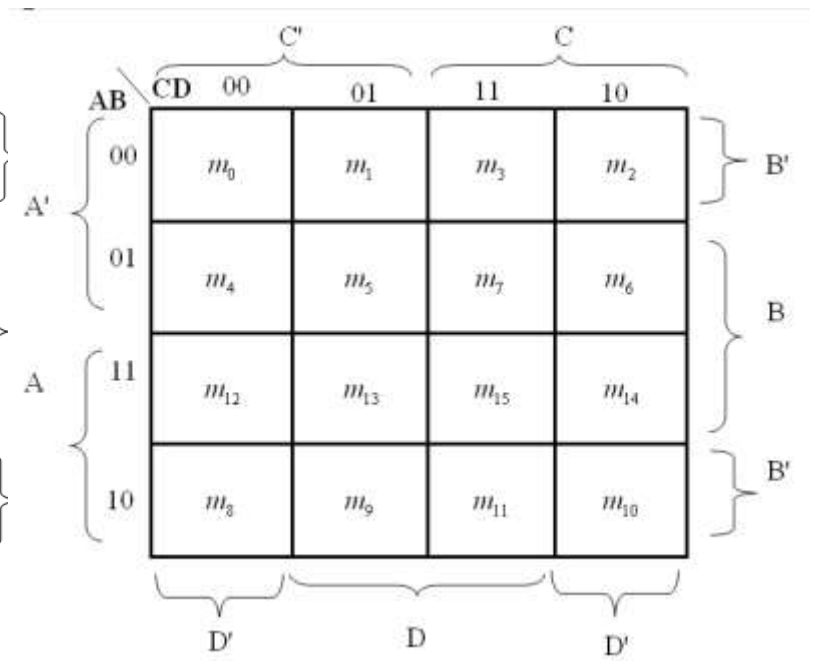
- Let $F = A'C + A'B + AB'C + BC$
  a) Express it in sum of minterms.
  b) Find the minimal sum of products expression.
  $F(A, B, C) = \Sigma(1, 2, 3, 5, 7) = C + A'B$



Four-Variable Map

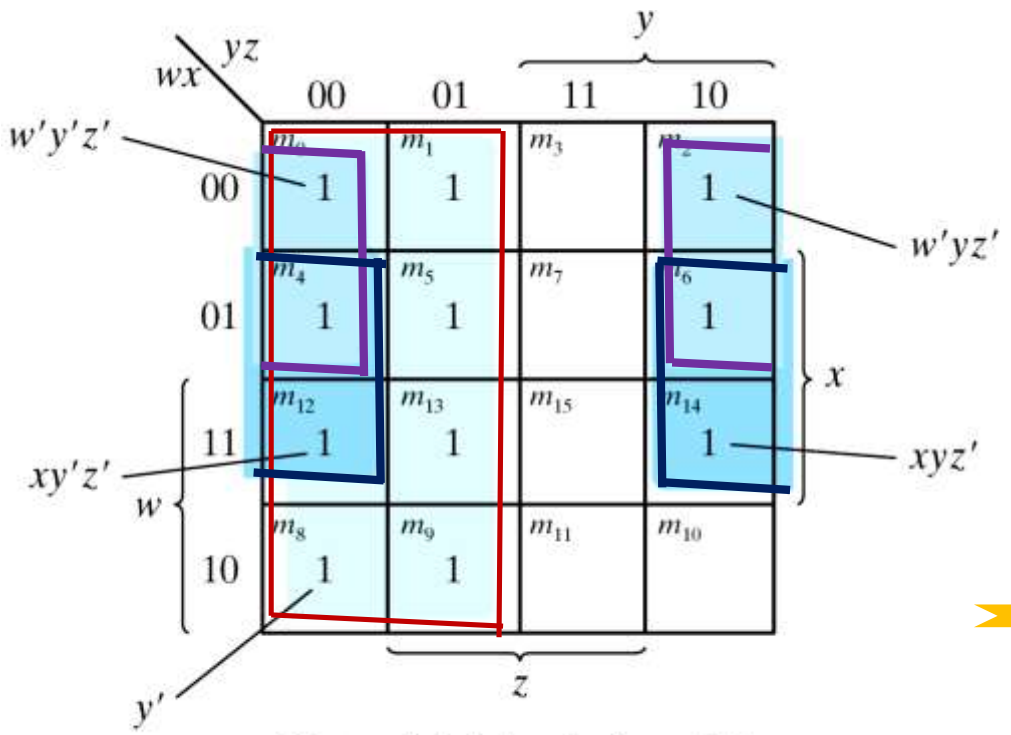
- 16 minterms
- Combinations of 2, 4, 8, and 16 adjacent squares





EXAMPLE 3.5

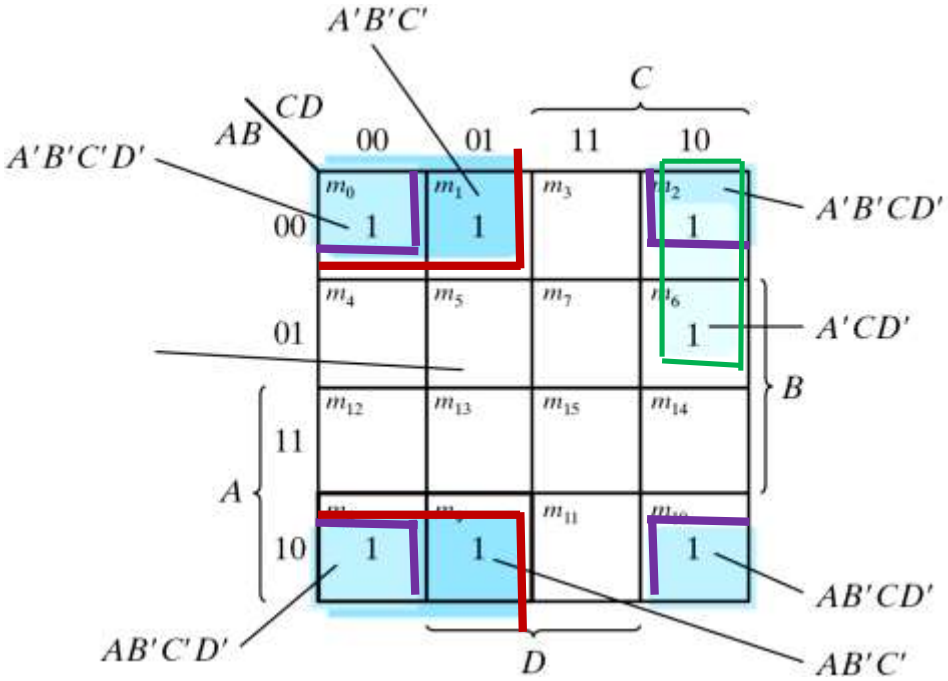
- Example 3.5: simplify $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$



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

EXAMPLE

- Simplify $F = A'B'C' + B'CD' + A'B'C'D' + AB'C'$



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

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

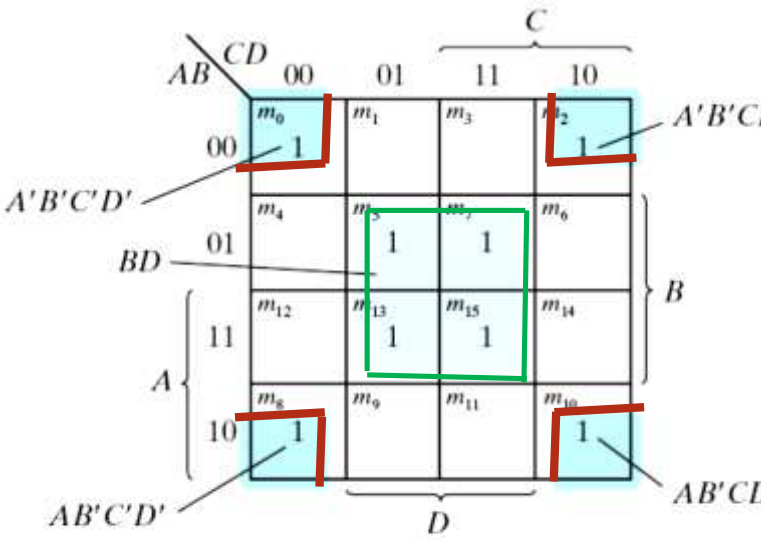
Prime Implicants

- All the minterms are covered.
- Minimize the number of terms.
- There are no redundant terms.

- Implicant: A group of one or more k map cell
- Prime implicant: an implicant that is not a subset of another implicant
- Essential Prime Implicant: a prime implicant that covers at least one cell not covered by another prime implicant

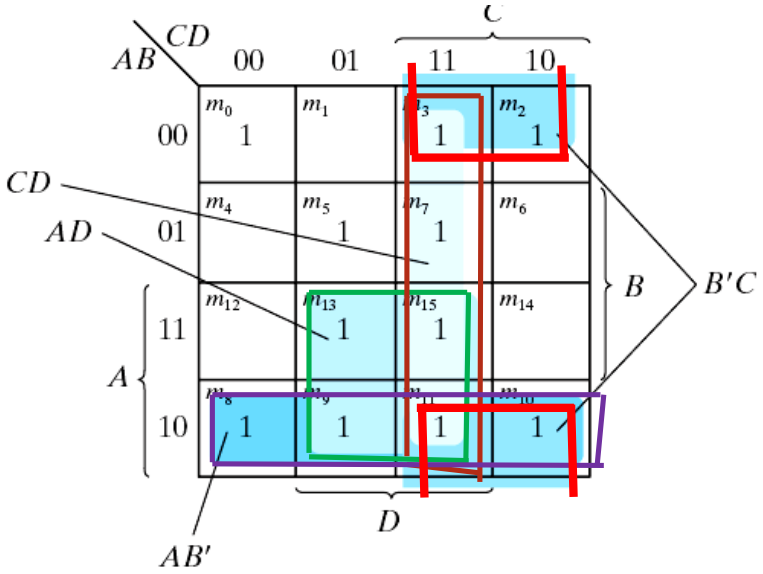
Prime Implicants

- $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$
  - The simplified expression may not be unique
  - $F = BD+B'D'+CD+AD = BD+B'D'+CD+AB'$
    $= BD+B'D'+B'C+AD = BD+B'D'+B'C+AB'$



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

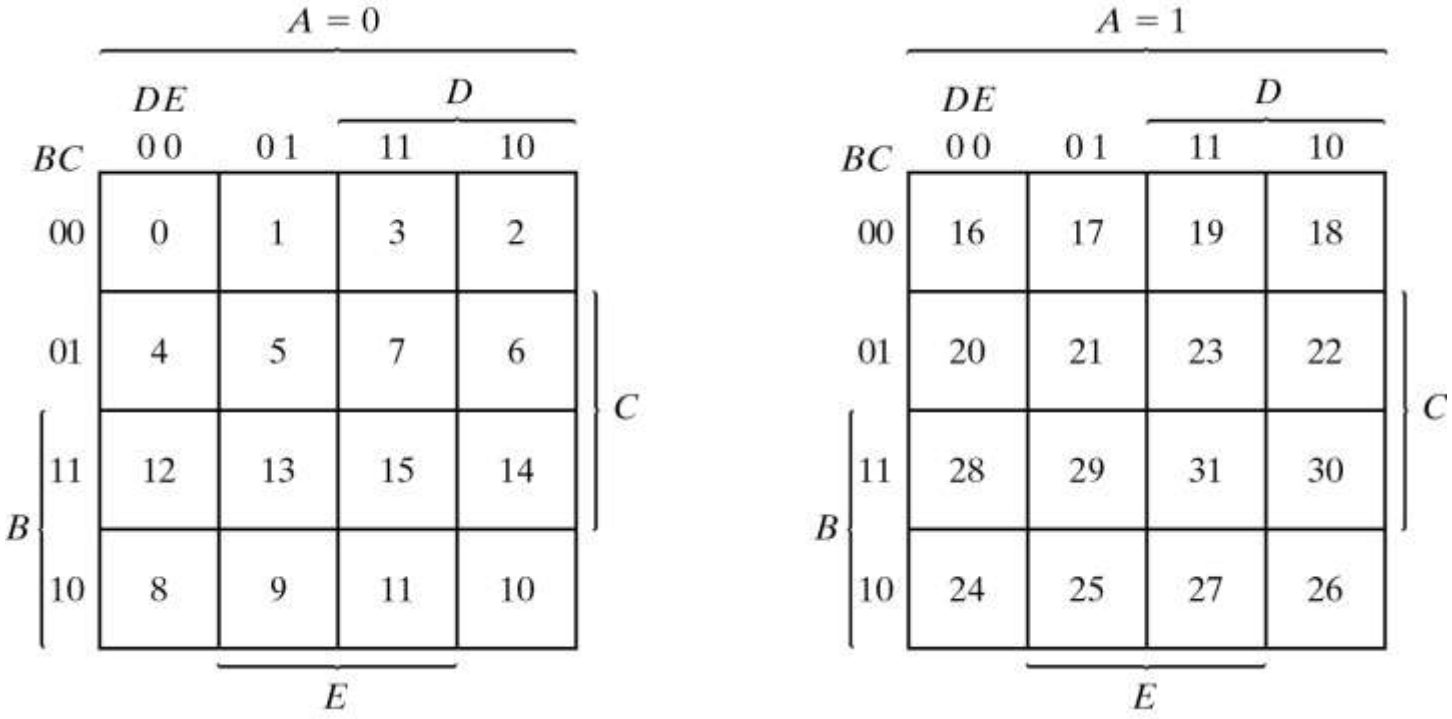
(a) Essential prime implicants BD and $B'D'$



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

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).



EXAMPLE

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

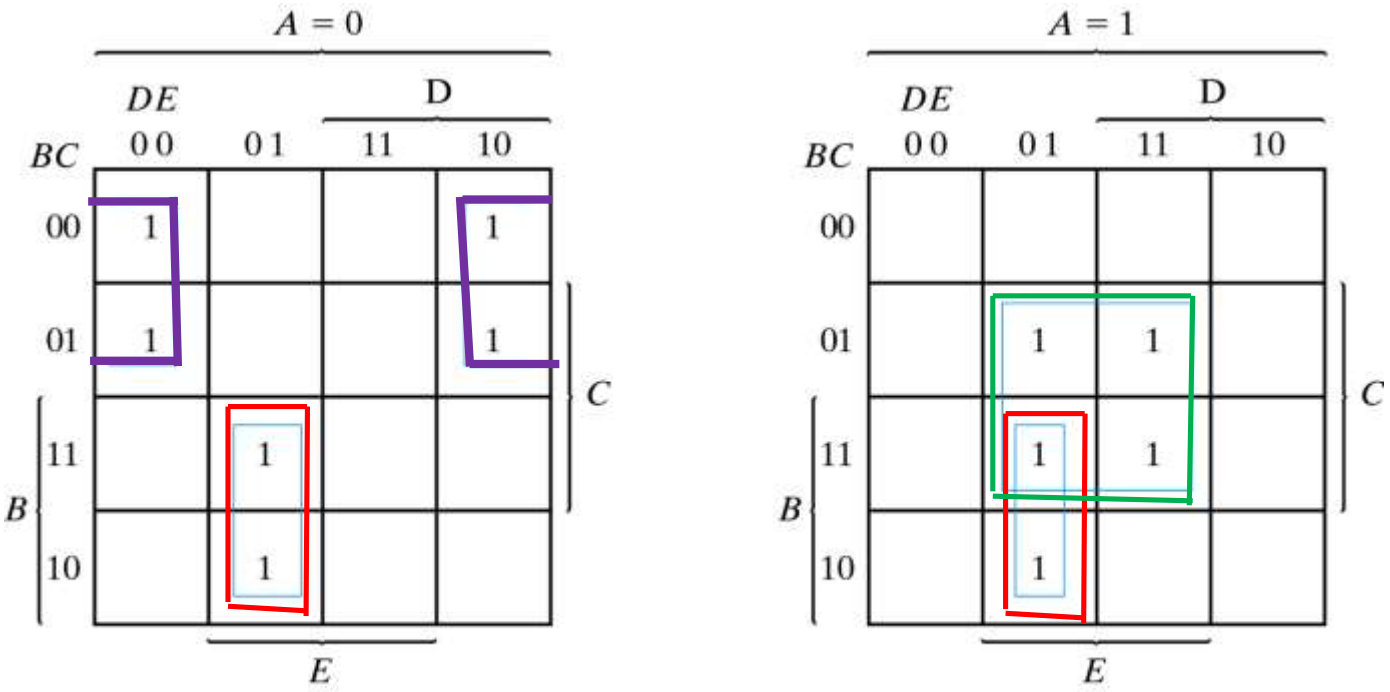
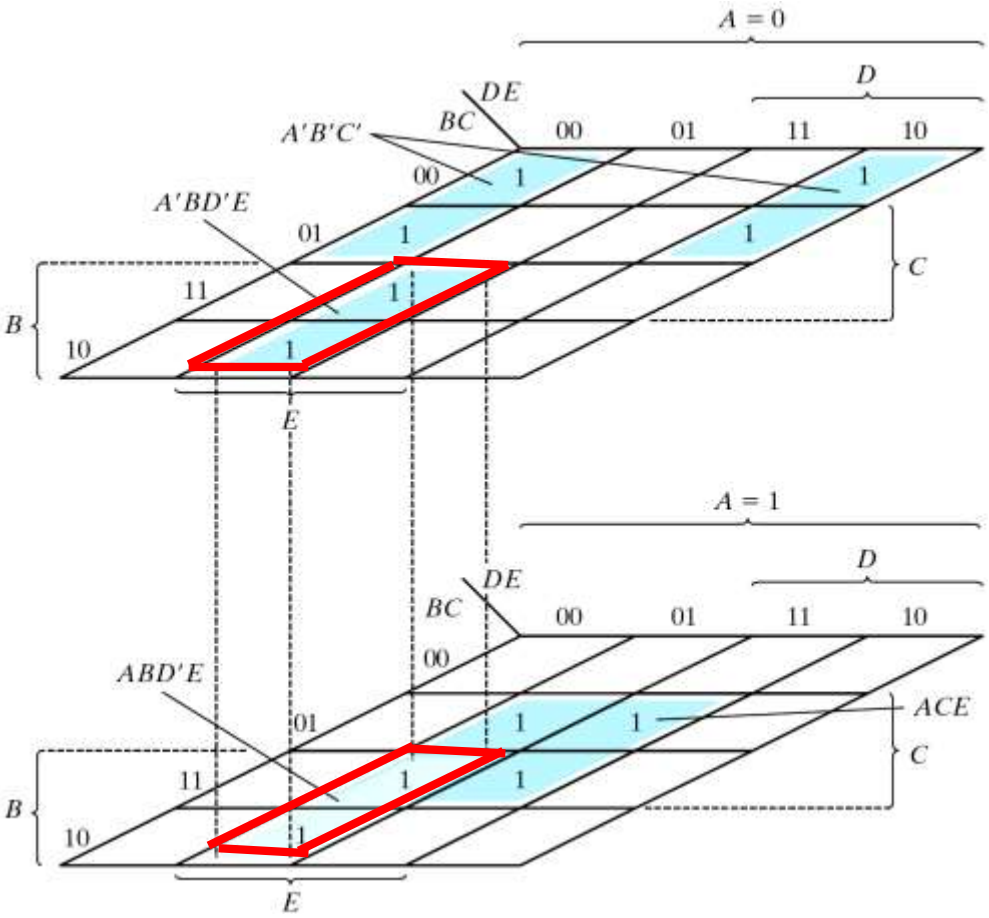


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

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

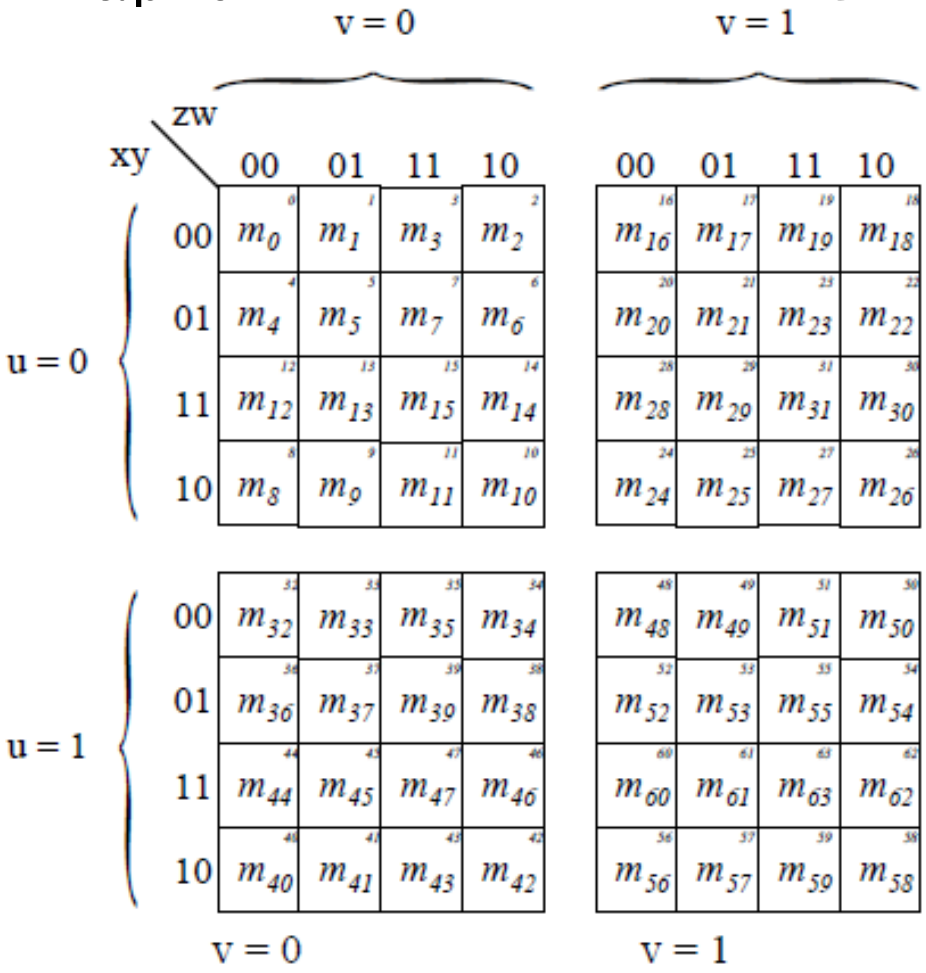
EXAMPLE

- Another Map for $F = A'B'E'+BD'E+ACE$



Six-Variable Map

- Another Map for $F = A'B'E' + BD'E + ACE$

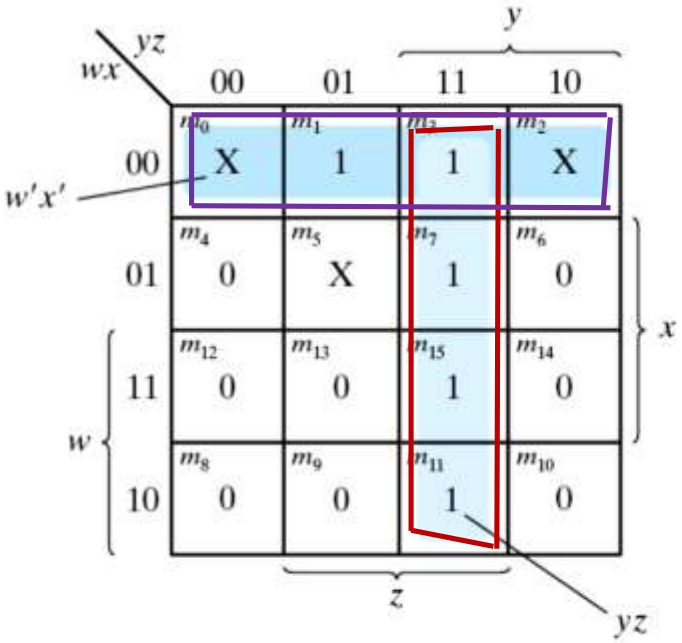


Don't-Care Conditions

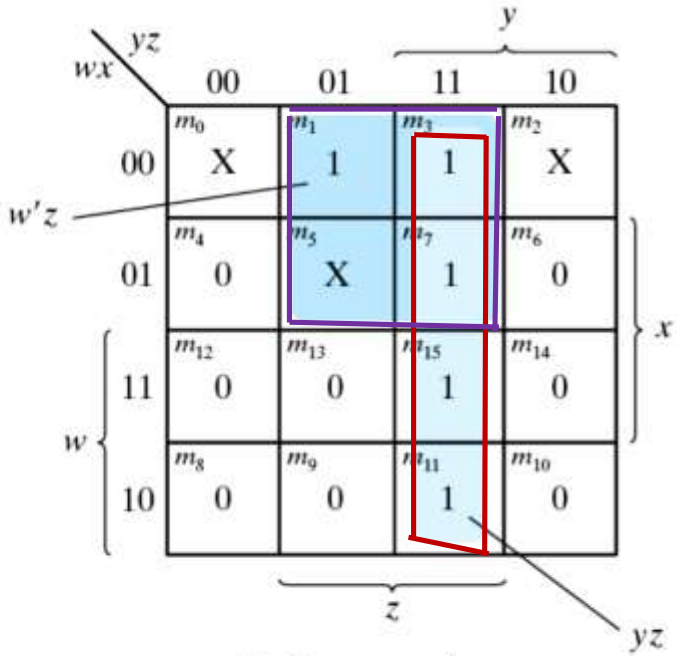
- The value of a function is not specified for certain combinations of variables
  - BCD; 1010-1111
- The unspecified minterms of a function are called the don't-care conditions, or simply the don't-cares, and are denoted as Xs.
- These don't-care conditions can be used on a map to provide further simplification of the Boolean expression.
  - Can be implemented as 0 or 1

EXAMPLE

- Simplify $F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$ which has the don't-care conditions $d(w, x, y, z) = \Sigma(0, 2, 5)$.
- $F = yz + w'x'$; $F = yz + w'z$
- $F = \Sigma(0, 1, 2, 3, 7, 11, 15)$; $F = \Sigma(1, 3, 5, 7, 11, 15)$



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



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

Don't-Care Conditions

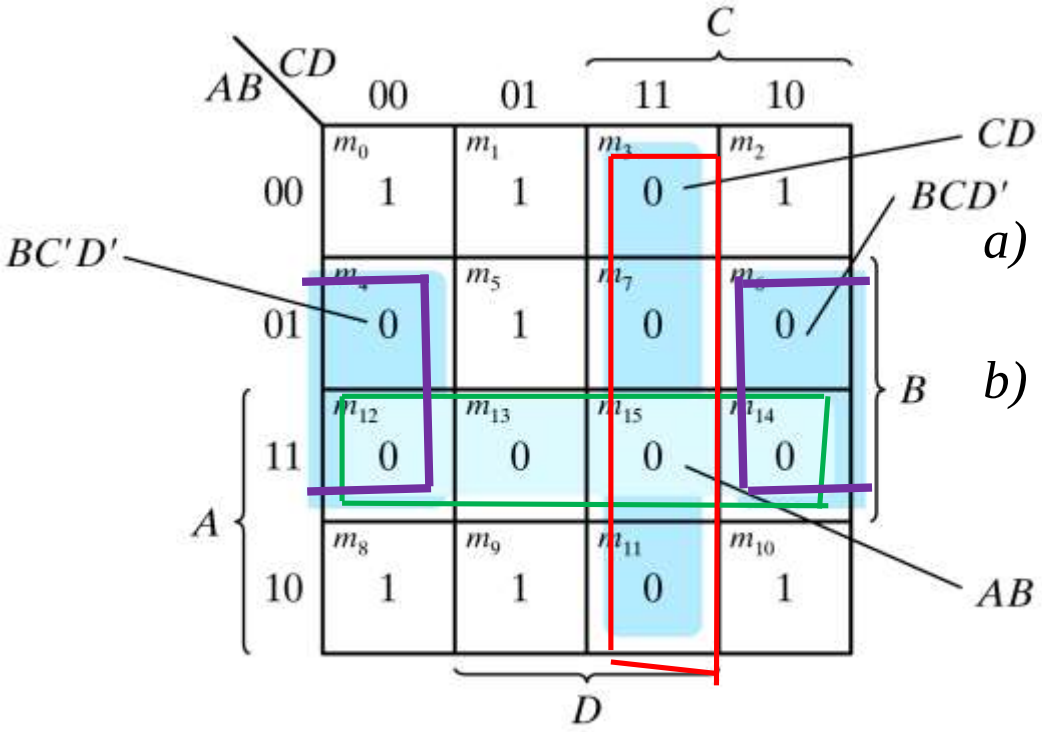
- Either one of the above expressions satisfies the conditions stated.
- Note that the above 2 expressions represent 2 functions that are algebraically unequal: each covers different don't-care terms.
- We may or may not include any of the Xs, while all the 1s must be included.

Product of Sums Simplification

- Simplify F' in the form of sum of products
- Apply DeMorgan's theorem $F = (F')'$
- F': sum of products $\rightarrow$ F: product of sums

EXAMPLE

- Simplify $F = \Sigma(0, 1, 2, 5, 8, 9, 10)$ into (a) sum-of-products form, and (b) product-of-sums form:



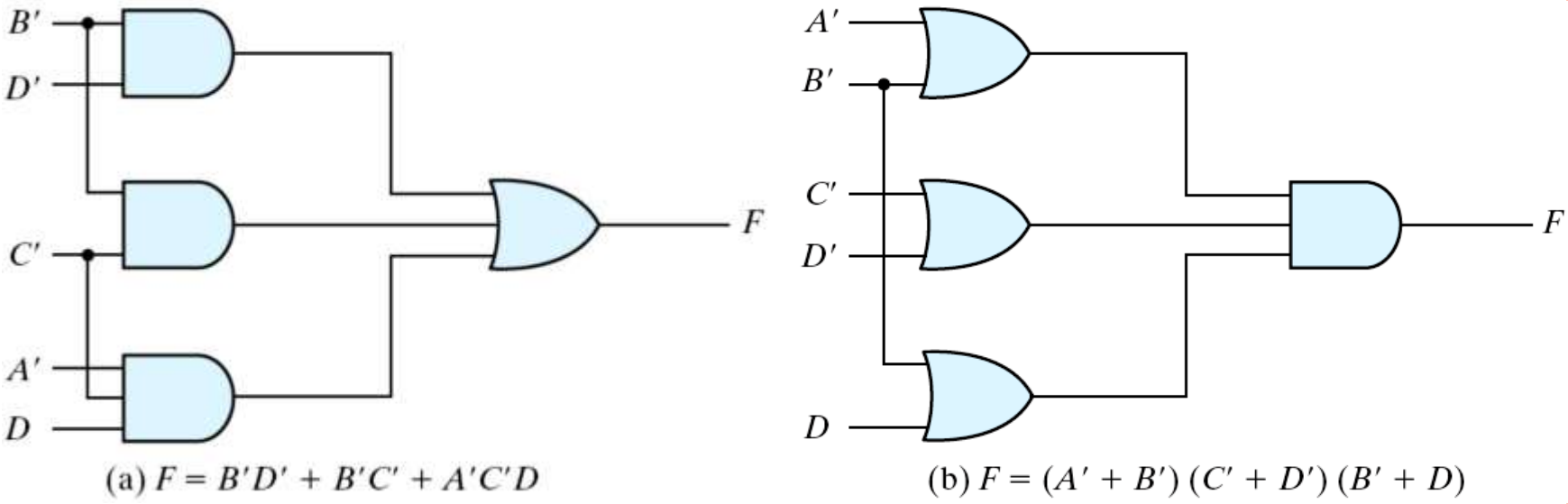
a) $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D$

b) $F' = AB + CD + BD'$

» Apply DeMorgan's theorem; $F = (A' + B')(C' + D')(B' + D)$

EXAMPLE (CONT.)

- Gate implementation of $F = \Sigma(0, 1, 2, 5, 8, 9, 10)$



Sum-of products form

Product-of sums form

Figure 3.15 Gate Implementation of $F = \Sigma(0, 1, 2, 5, 8, 9, 10)$

Product of Maxterms

- Consider the function defined in Table 3.2.
  - In sum-of-minterm:

$$F(x, y, z) = \sum(1,3,4,6)$$

  - In sum-of-maxterm:

$$F(x, y, z) = \prod(0,2,5,7)$$

Table 3.2
Truth Table of Function F

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Product of Maxterms

- Consider the function defined in Table 3.2.
  - Combine the 1's:

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

  - Combine the 0's :

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

  - Taking the complement of F'

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

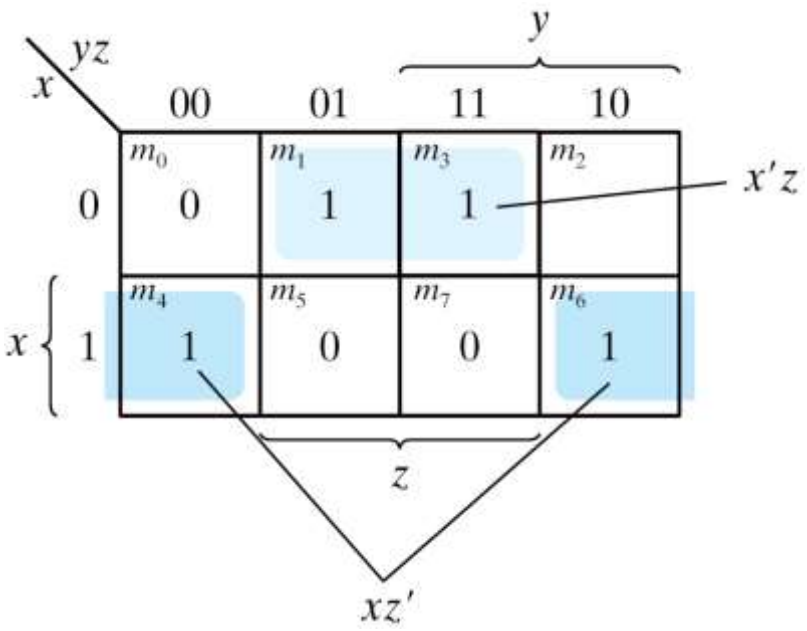


Figure 3.16 Map for the function of Table 3.2

Map Method: Summary

- Simplification procedure:
  - Generate K-map.
  - Determine PIs.
  - Select EPIs.
  - Find minimal cover (set) of PIs.

Simplification of Boolean functions

- 3 common methods:
- Boolean algebra
  - The basis for all methods
  - Difficult to see the best path to take and to know when you are finished
- Karnaugh maps
  - Fast and easy for 2 – 6 variables
  - Pictorial approach based on recognition of patterns
  - Difficult for large numbers of variables
- Tabulation methods
  - Tedious to perform by hand, but well suited to computer implementation
  - No limit on the number of variables

Tabulation Methods

- The tabulation method was first formulated by Quine and later improved by McCluskey. It is also known as the Quine-McCluskey method.

- Simplification consists of two parts:
  - To find candidate prime implicants.
  - To choose the prime implicants that give an expression with the least number of literals (i.e., minimal cover generation).

Determination of prime implicants

- Group minterms by number of 1s.
- Compare minterms and find pairs with distance 1.
- Generate subcubes.
- Repeat the above procedure on generated subcubes until no more subcubes can be generated.

Determination of prime implicants

- $F(w,x,y,z) = \sum(0, 1, 2, 8, 10, 11, 14, 15)$
  $= w'x'y' + x'z' + wy$

(a)			(b)			(c)	
	$wxyz$			$wxyz$			$wxyz$
0	0000	✓	0,1	000–		0,2,8,10	–0–0
			0,2	00–0	✓	0,8,2,10	–0–0
1	0001	✓	0,8	–000	✓		
2	0010	✓				10,11,14,15	1–1–
8	1000	✓	2,10	–010	✓	10,14,11,15	1–1–
			8,10	10–0	✓		
10	1010	✓					
			10,11	101–	✓		
11	1011	✓	10,14	1–10	✓		
14	1110	✓					
			11,15	1–11	✓		
15	1111	✓	14,15	111–	✓		

Determination of prime implicants

- Step 1: Group the minterms according to the number of 1s (see Column (a)).

- Step 2: Combine any 2 minterms that differ from each other by exactly one variable (i.e., dist-1), the unmatched variable removed (see Column (b)). Try this for all possible pairs of minterms. A check (3) is placed to the right of both minterms if they have been used in a match.

Determination of prime implicants

- Step 3: Repeat the process. Combine any 2 product terms from Step that differ from each other by exactly one variable, the unmatched variable removed (see Column (c)).

- Step 4: The unchecked terms in the table form the PIs. Some of the product terms may appear twice in the table. It of course is unnecessary to use the same term twice.

Minimal cover generation

- The selection of PIs that form the minimized function is made from a PI table
- $F(w,x,y,z) = \sum(1, 4, 6, 7, 8, 9, 10, 11, 15)$

	PI	Minterms	1	4	6	7	8	9	10	11	15
✓	$x'y'z$	1,9	X					X			
✓	$w'xz'$	4,6		X	X						
	$w'xy$	6,7			X	X					
	xyz	7,15				X					X
	wyz	11,15								X	X
✓	wx'	8,9,10,11					X	X	X	X	
			✓	✓	✓		✓	✓	✓	✓	

Minimal cover generation

- $F(w,x,y,z) = \sum(1, 4, 6, 7, 8, 9, 10, 11,15)$
- Essential Prime implicants are $x'y'z$, $w'xz'$, wx'
- $F(w,x,y,z) = x'y'z + w'xz' + wx' + xyz$

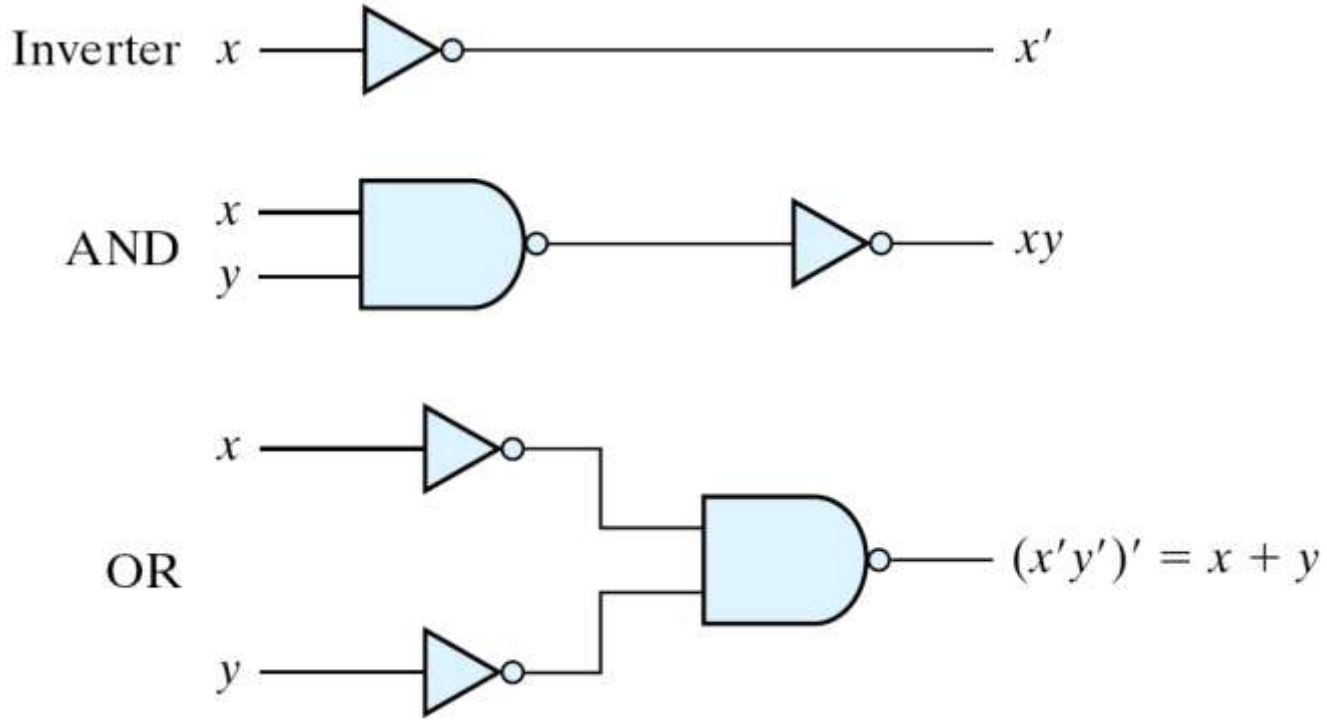
PI	Minterms	7	15
$w'xy$	6, 7	X	
xyz	7, 15	X	X
wyz	11, 15		X

Minimal cover generation

- Step 1: Check the PIs that cover minterms with a single X in their columns. These PIs are the EPIs.
- Step 2: Check each column whose minterm is covered by the selected EPIs.
- Step 3: If there are minterms left uncovered (7 & 15 in this example), some nonessential PIs have to be selected to cover them. We of course will select the PIs with a smallest total number of literals (xyz in this example).

NAND and NOR Implementation

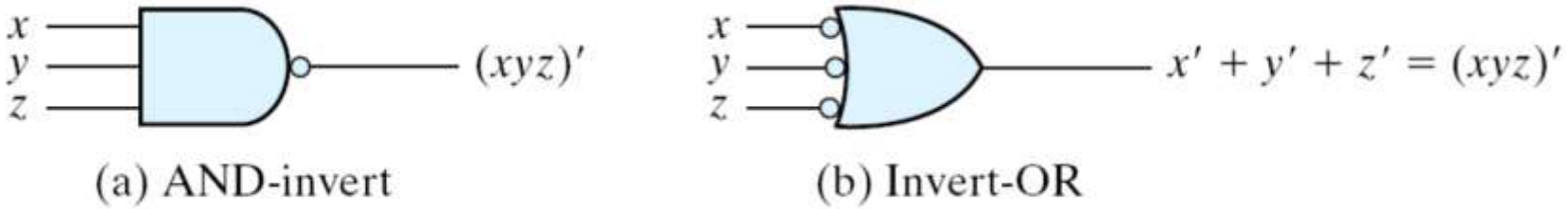
- NAND gate is a universal gate
  - Can implement any digital system



Logic Operations with NAND Gates

NAND Gate

- Two graphic symbols for a NAND gate



Two Graphic Symbols for NAND Gate

Two-level Implementation

- Two-level logic
  - NAND-NAND = sum of products
  - Example: $F = AB+CD$
  - $F = ((AB)'\,(CD)'\,)' = AB+CD$

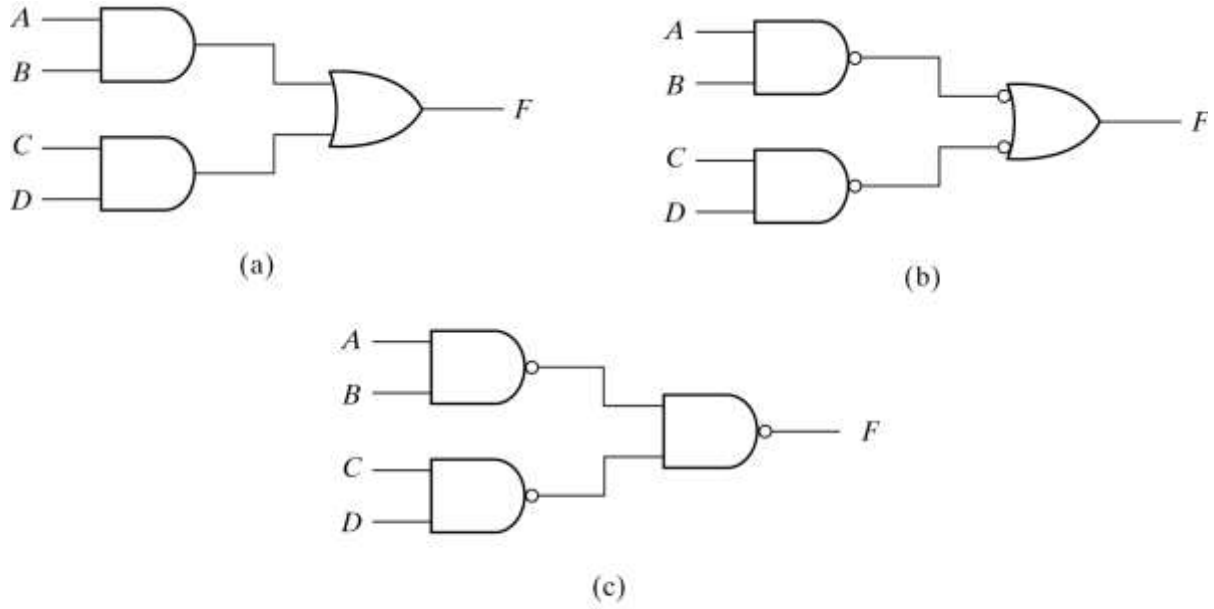
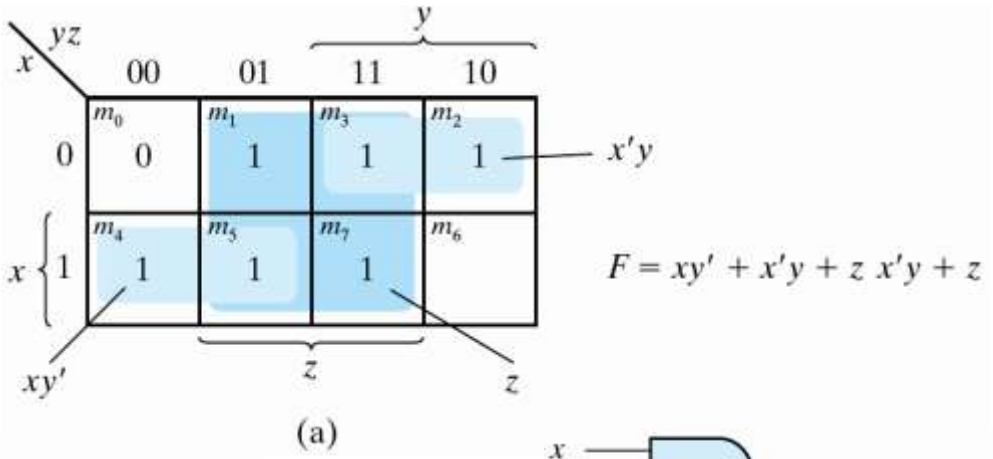


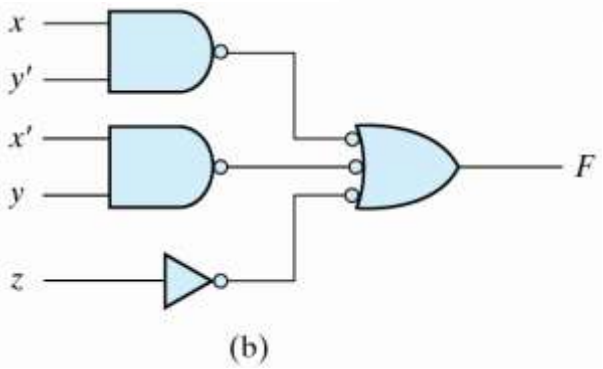
Fig. 3-20 Three Ways to Implement $F = AB + CD$

EXAMPLE

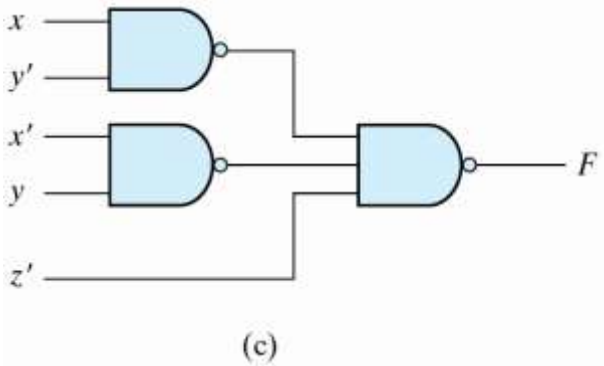
$$F(x,y,z) = \sum(1,2,3,4,5,7) \quad \Rightarrow \quad F(x,y,z) = xy' + x'y + z$$



(a)



(b)



(c)

Two -Level NAND Circuits

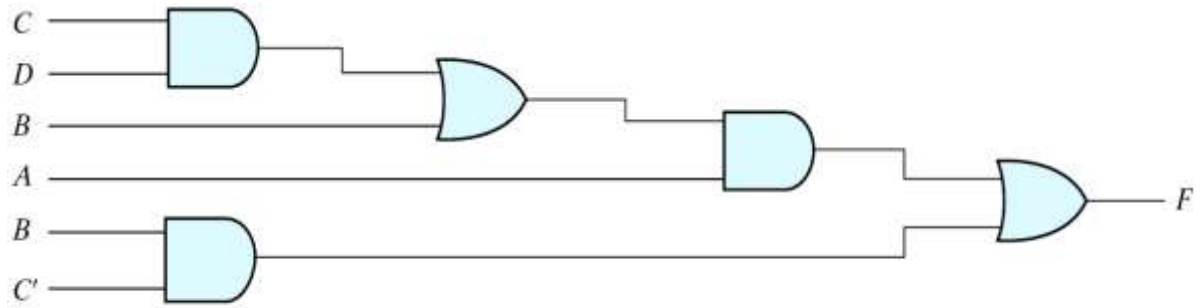
- The procedure
  - Simplify the function and express it in SOP form
  - Draw a NAND gate for each product term
  - Draw a single gate using AND-invert or invert-OR symbol for the sum term
  - A term with single literal, complement if needed

Multilevel NAND Circuits

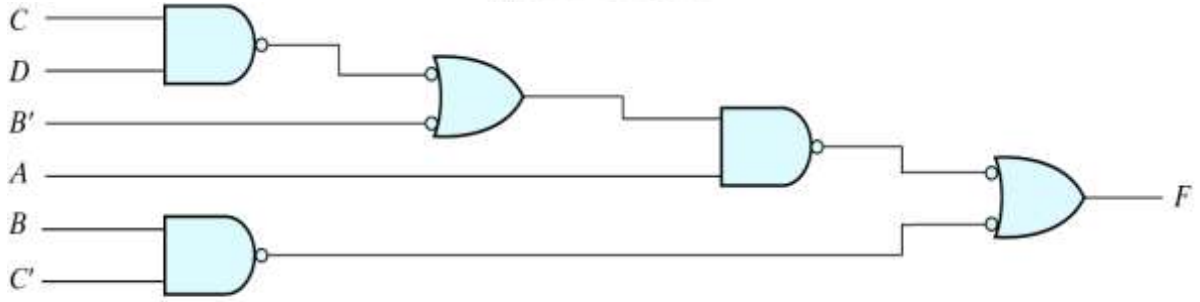
- The procedure
  - Convert all AND gates to NAND gates
  - Convert all OR gates to NAND gates with invert-OR symbols
  - Balance all bubbles, insert an inverter if needed

Multilevel NAND Circuits

- $F = A(CD + B) + BC'$



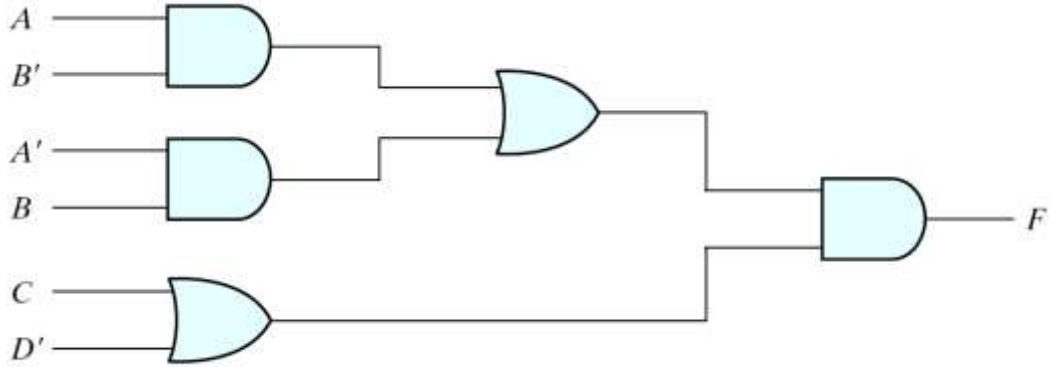
(a) AND–OR gates



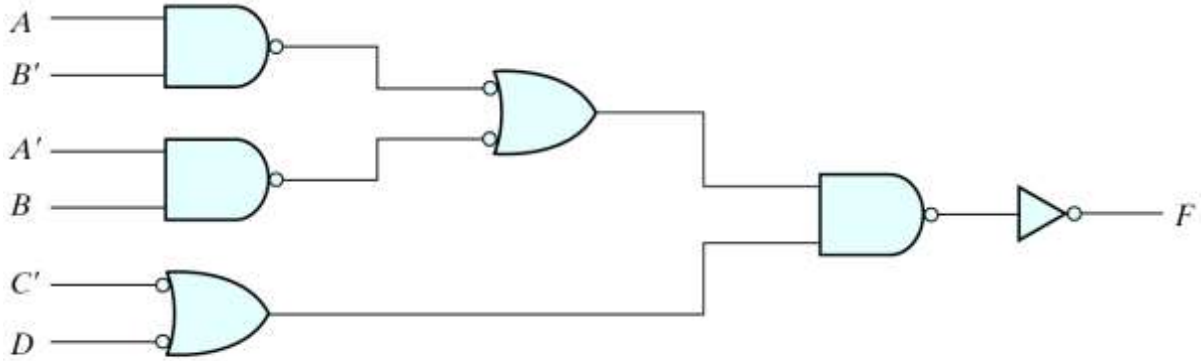
(b) NAND gates

Multilevel NAND Circuits

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



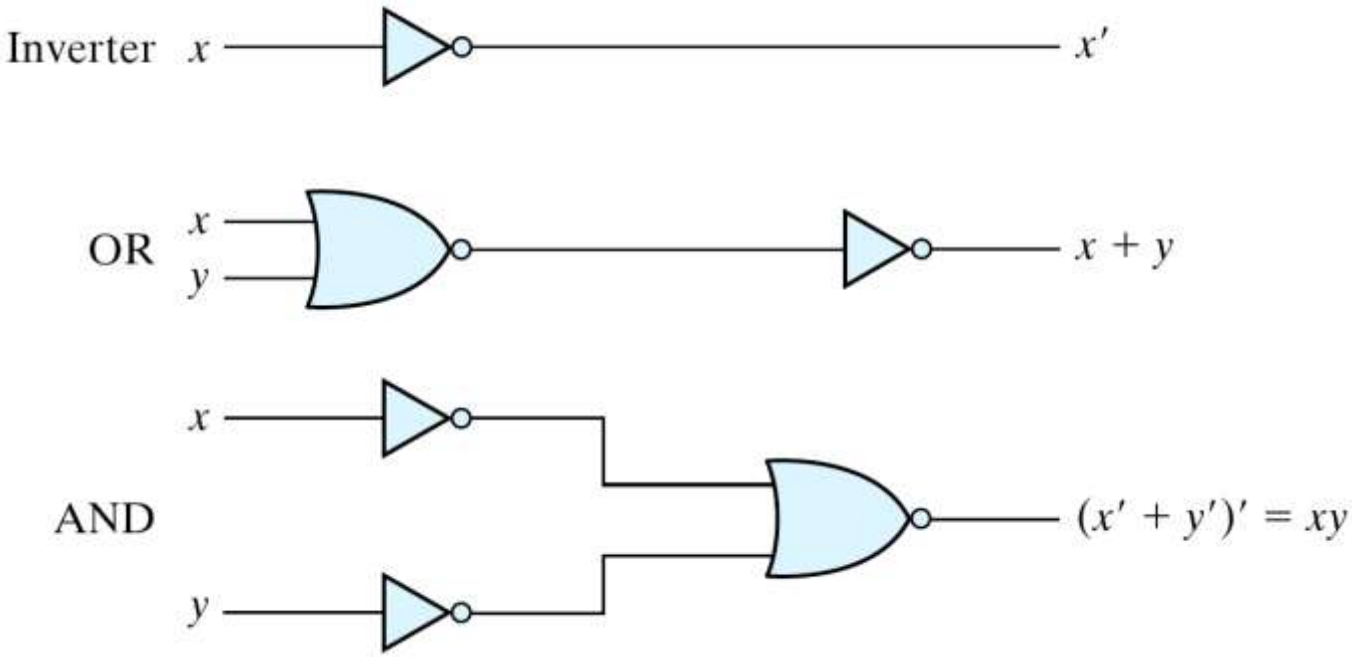
(a) AND–OR gates



(b) NAND gates

NOR Implementation

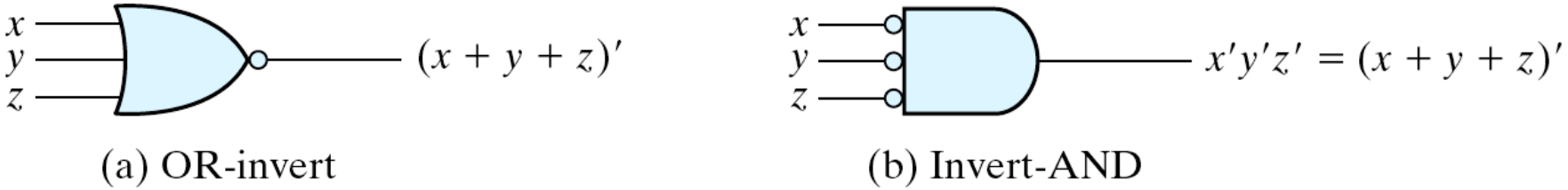
- NOR function is the dual of NAND function.
- The NOR gate is also universal.



Logic Operation with NOR Gates

NOR Gate

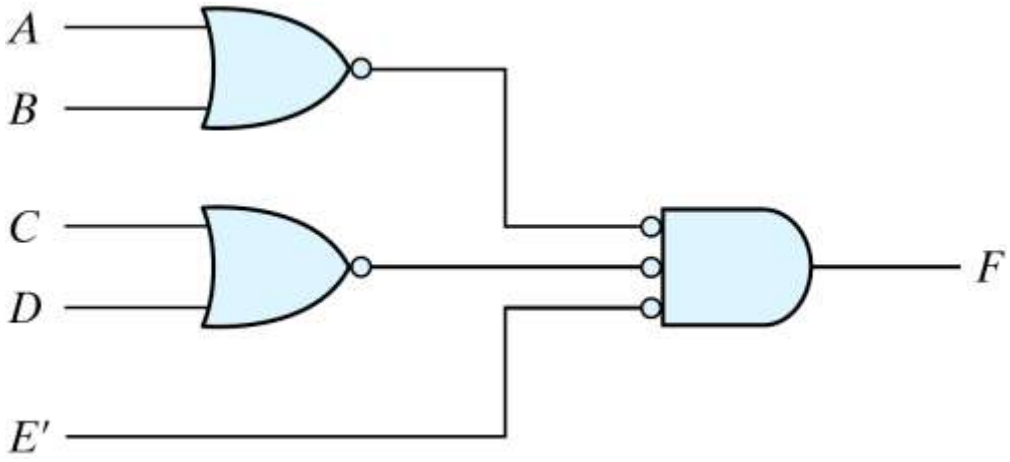
- Two graphic symbols for a NOR gate



Two Graphic Symbols for NOR Gate

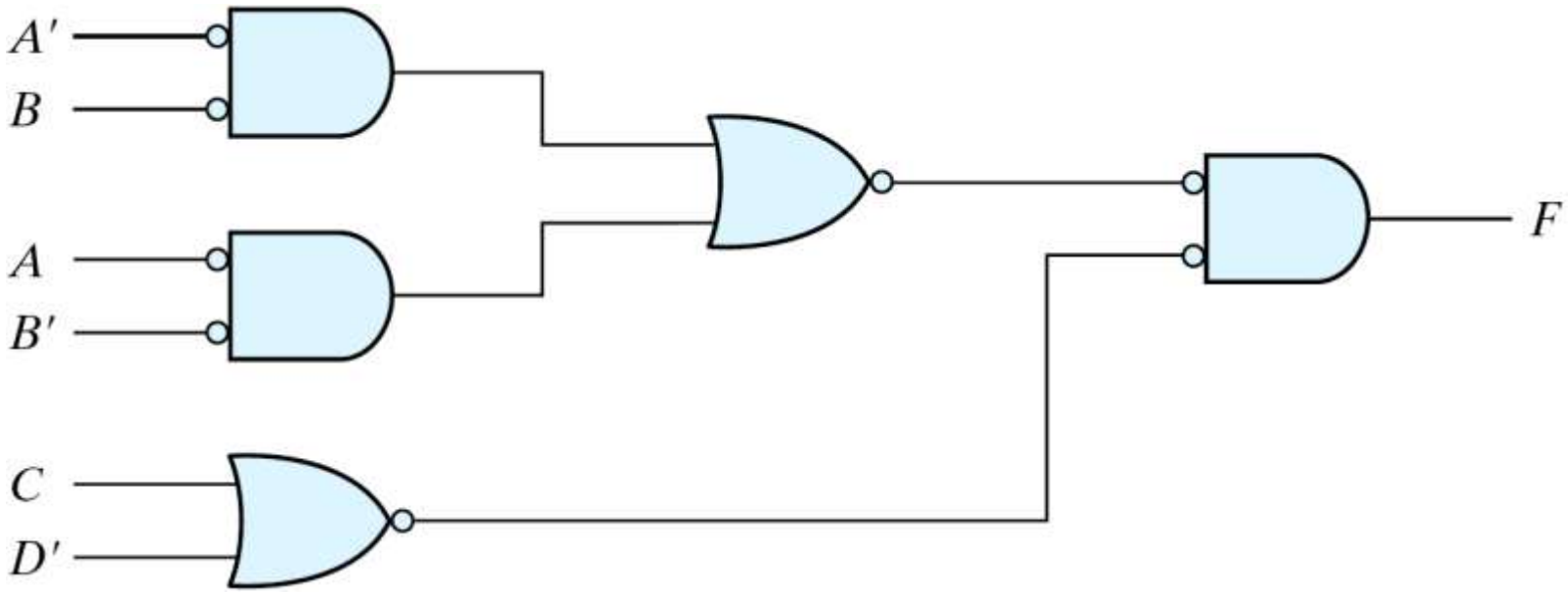
NOR Implementation

- $F = (A + B)(C + D)E$



NOR Implementation

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



Other Two-level Implementations

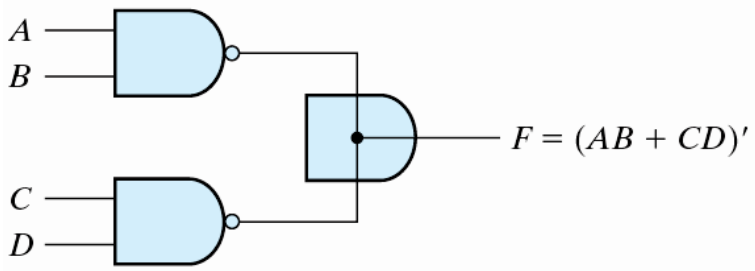
- Wired logic
  - In some technologies, it is possible to short O/Ps of some logic gates

$$F = (AB)' \cdot (CD)' = (AB + CD)' = (A' + B')(C' + D')$$

AND-OR-INVERT function

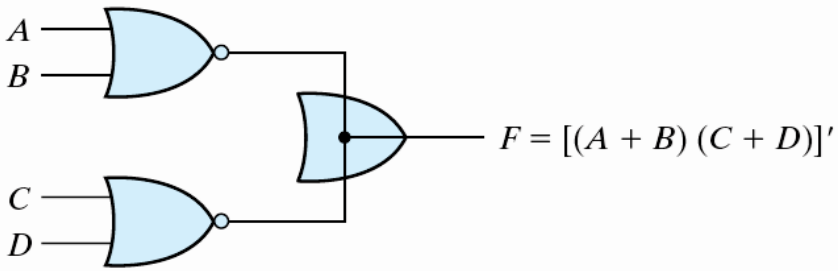
$$F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$$

OR-AND-INVERT function



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

(AND–OR–INVERT)



(b) Wired-OR in ECL gates

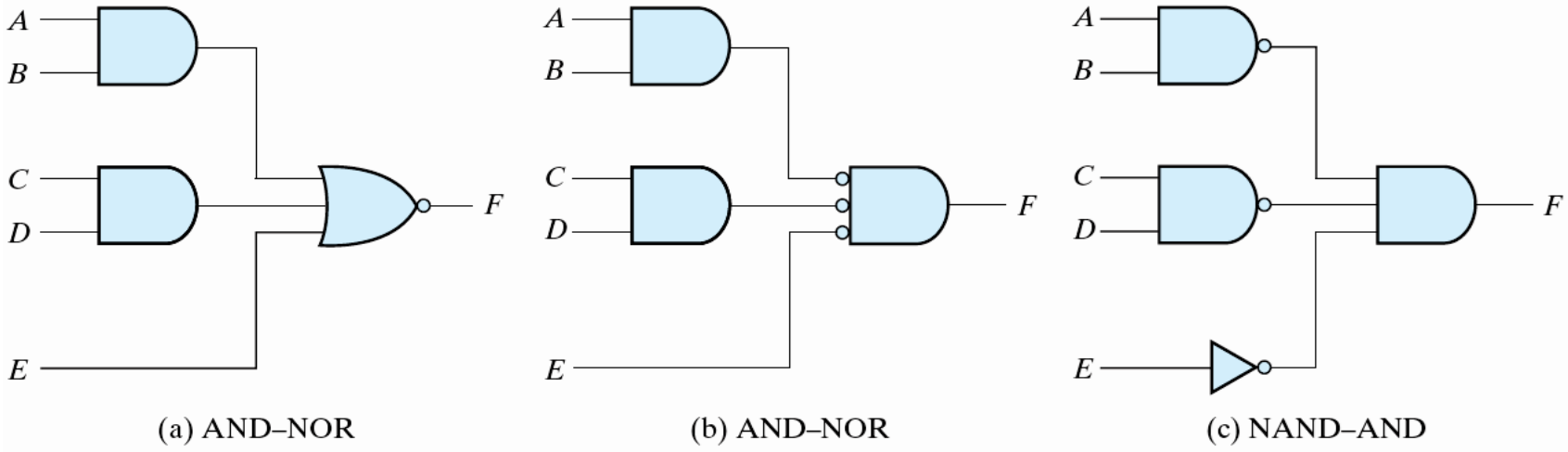
(OR–AND–INVERT)

Non-degenerate Forms

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

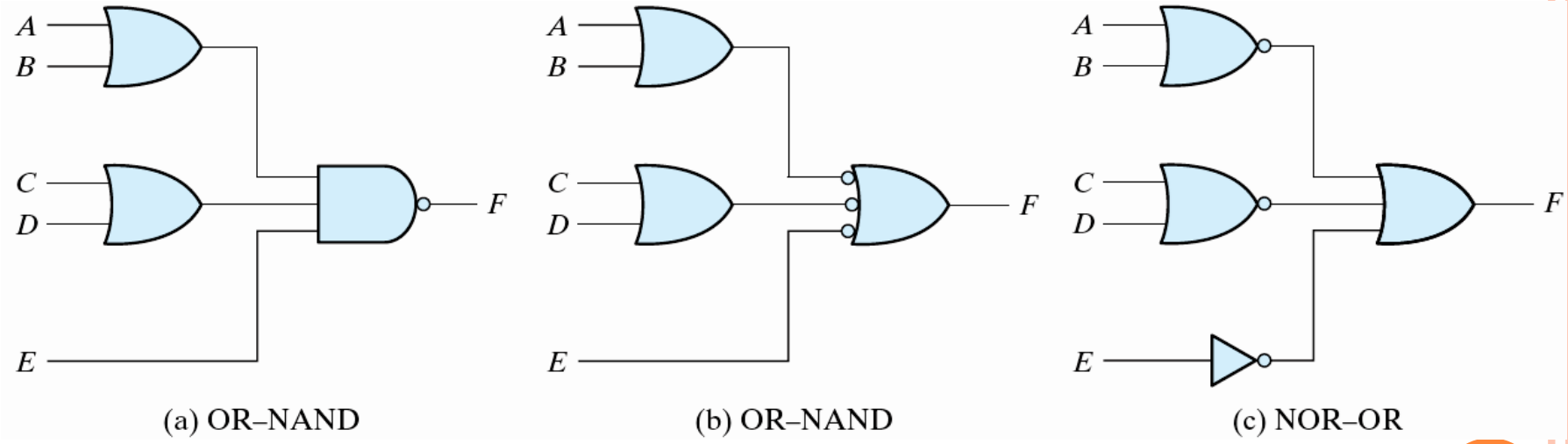
- AND-OR-INVERT (AOI) Implementation
  - NAND-AND = AND-NOR = AOI
  - $F = (AB+CD+E)'$
  - $F' = AB+CD+E$ (sum of products)



AND-OR-INVERT circuits, $F = (AB + CD + E)'$

OR-AND-INVERT IMPLEMENTATION

- 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)



OR-AND-INVERT circuits, $F = ((A+B)(C+D)E)'$

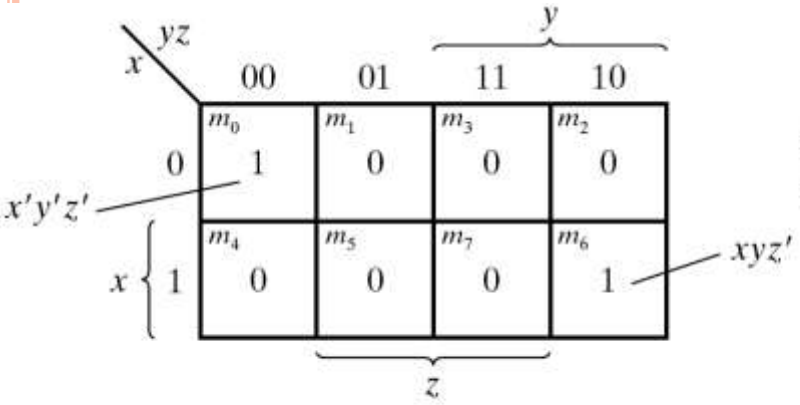
Tabular Summary and Examples

Table 3.3
Implementation with Other Two-Level Forms

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.

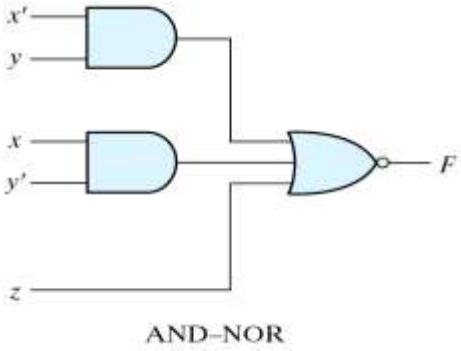
EXAMPLE



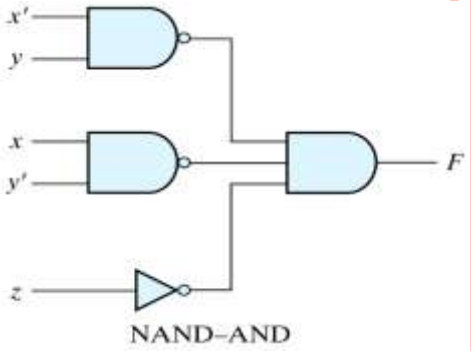
(a) Map simplification in sum of products

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

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

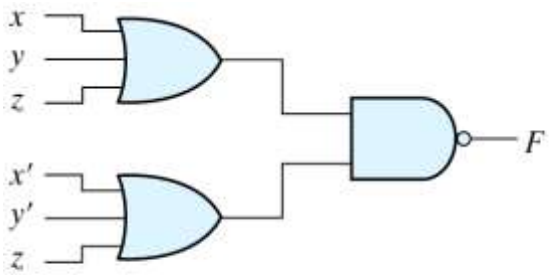


AND–NOR

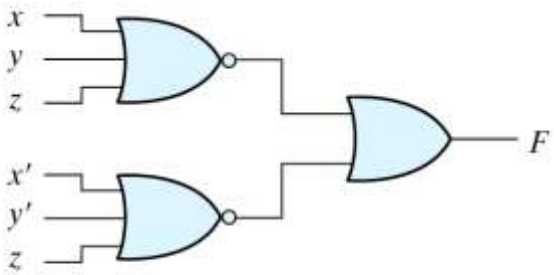


NAND–AND

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



OR–NAND



NOR–OR

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

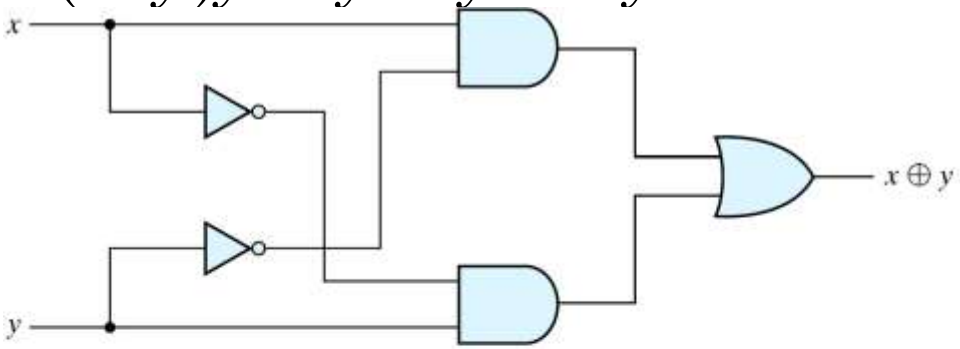
Other Two-level Implementations

Exclusive-OR Function

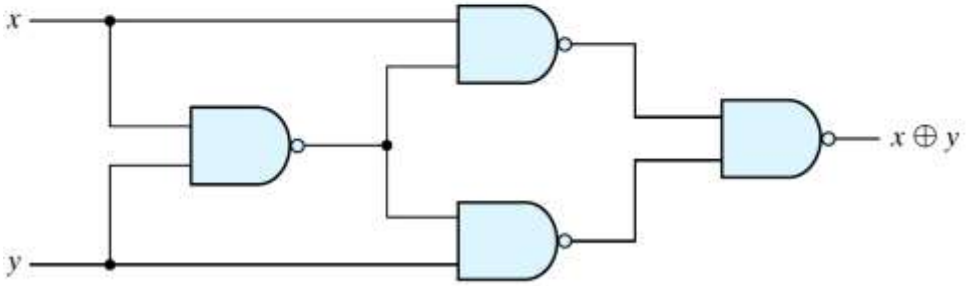
- Exclusive-OR (XOR)
  - $x \oplus y = xy' + x'y$
- Exclusive-NOR (XNOR)
  - $(x \oplus y)' = xy + x'y'$
- Some identities
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = (x \oplus y)'$
  - $x' \oplus y = (x \oplus y)'$
- Commutative and associative
  - $A \oplus B = B \oplus A$
  - $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

Exclusive-OR Implementations

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



(a) With AND–OR–NOT gates

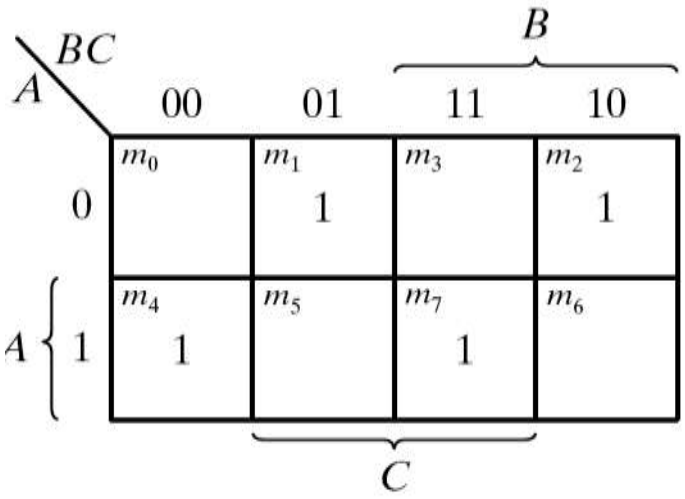


(b) With NAND gates

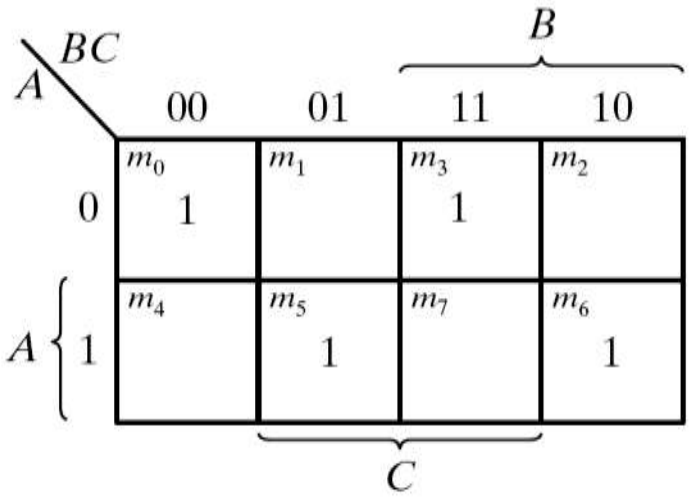
Exclusive-OR Implementations

Odd Function

- $A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C = AB'C' + A'BC' + ABC + A'B'C = \Sigma(1, 2, 4, 7)$
- XOR is a odd function → an odd number of 1's, then $F = 1$.



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

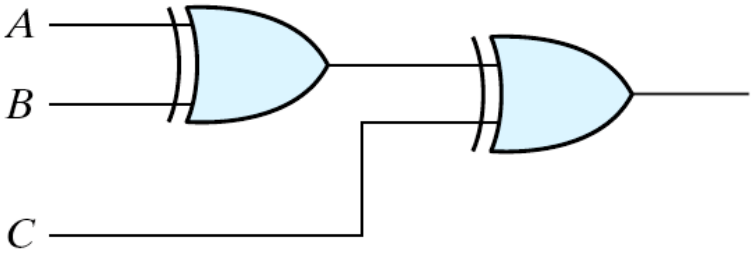


(b) Even function $F = (A \oplus B \oplus C)'$

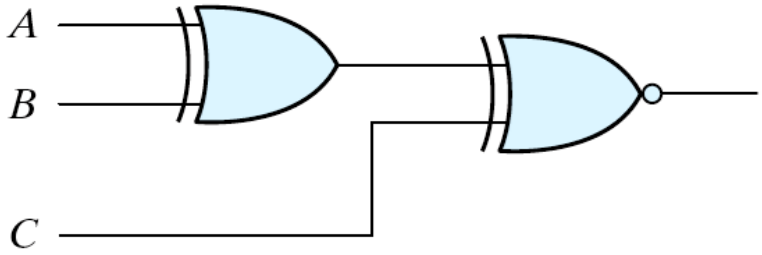
Map for a Three-variable Exclusive-OR Function

XOR and XNOR

- Logic diagram of odd and even functions



(a) 3-input odd function

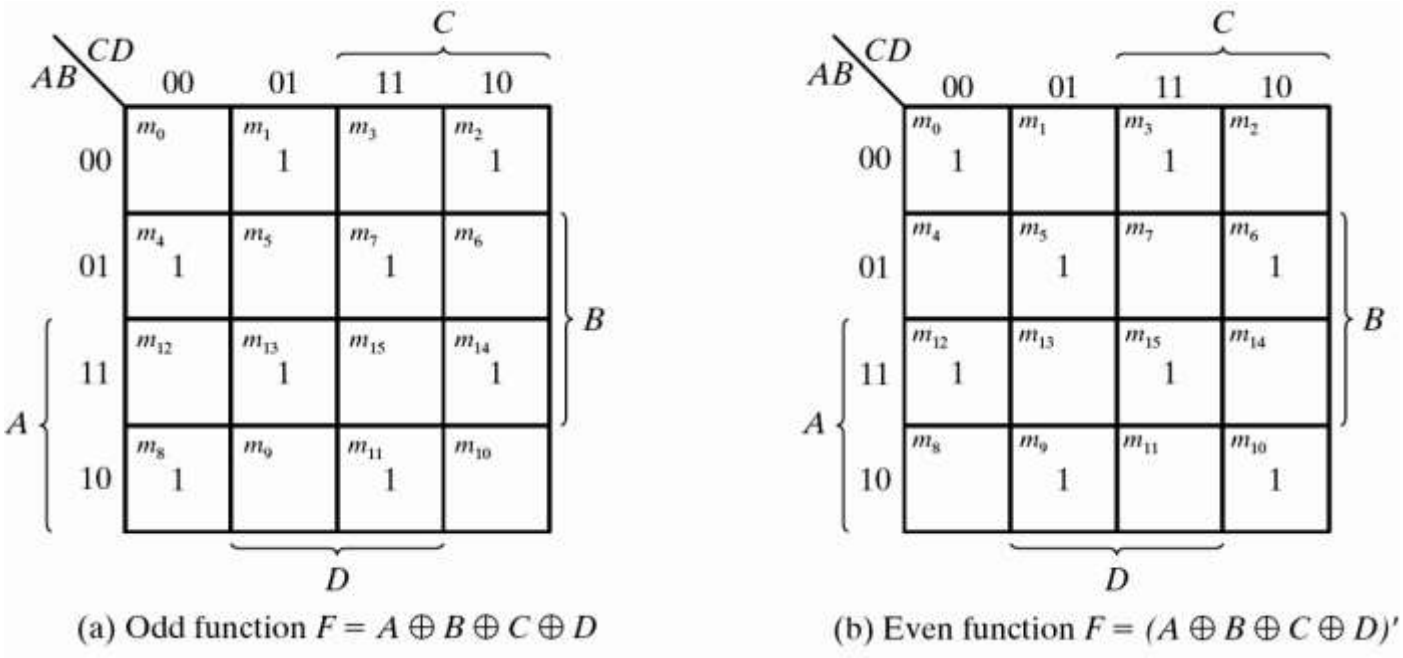


(b) 3-input even function

Logic Diagram of Odd and Even Functions

Four-variable Exclusive-OR function

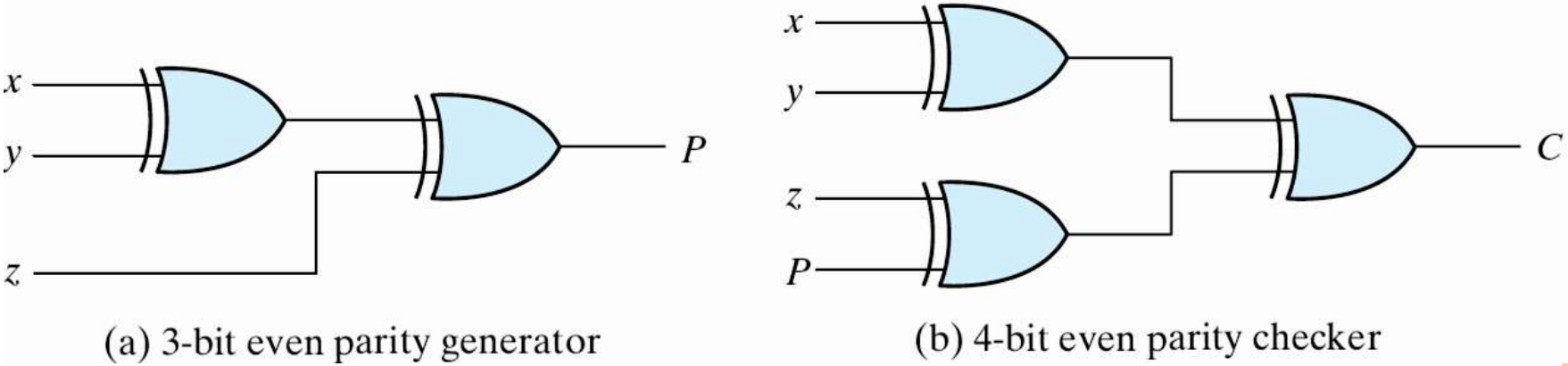
- Four-variable Exclusive-OR function
  - $A \oplus B \oplus C \oplus D = (AB'+A'B) \oplus (CD'+C'D) =$
    $(AB'+A'B)(CD+C'D')+(AB+A'B')(CD'+C'D)$



Map for a Four-variable Exclusive-OR Function

Parity Generation and Checking

- Parity Generation and Checking
  - A parity bit: $P = x \oplus y \oplus z$
  - Parity check: $C = x \oplus y \oplus z \oplus P$
    - C=1: one bit error or an odd number of data bit error
    - C=0: correct or an even # of data bit error



(a) 3-bit even parity generator (b) 4-bit even parity checker

Logic Diagram of a Parity Generator and Checker

Parity Generation and Checking

Table 3.4

Even-Parity-Generator Truth Table

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

Parity Generation and Checking

Table 3.5

Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0