# Chap 3
# Gate-Level Minimization

Hanwool Jeong

hwjeong@kw.ac.kr

# Introduction

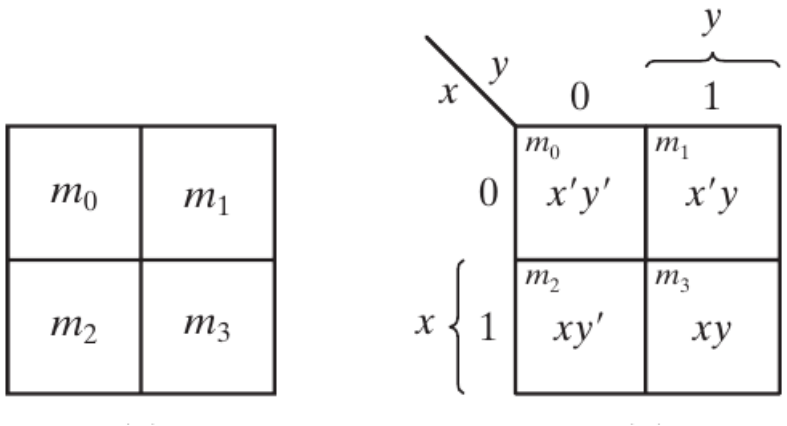
- Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions.
- Although the truth table representation is unique, algebraic expression can appear in many different forms.
- Boolean expressions may be simplified by algebraic means as discussed in Section 2.4.
- However, this procedure of minimization is awkward, susceptible to human error, and uncertain.

# Map Method

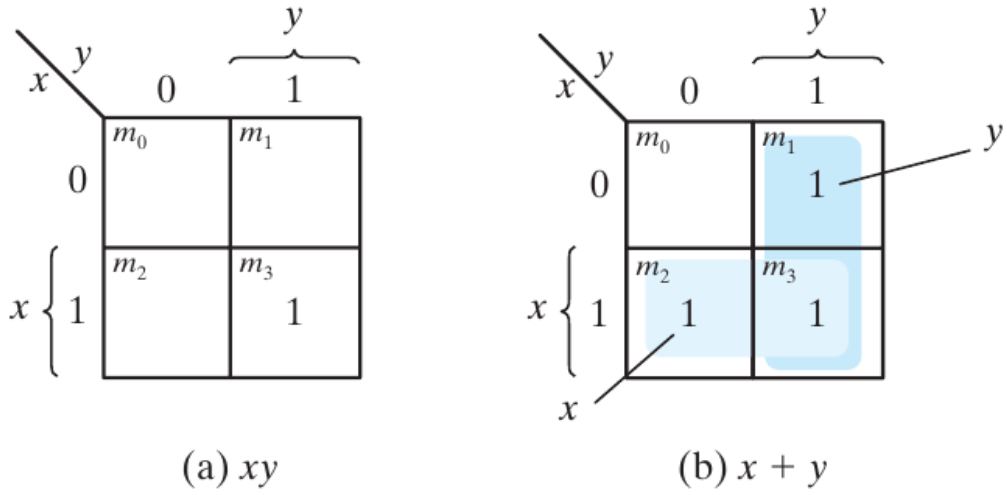
- The map method presented here provides a simple, straightforward procedure for minimizing Boolean functions.
- Regarded as a pictorial form of a truth table, the map method is also known as the Karnaugh map or K-map.
- A K-map is a diagram made up of squares, with each square representing one minterm of the function to be minimized.
- The simplest algebraic expression with a minimum number of terms and with the smallest possible number of literals in each term can be systematically derived using K-map

# Two-Variable K-Map

- Structure of K-map



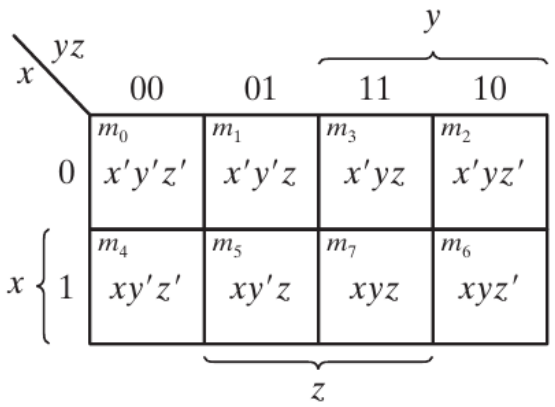

- Representation of functions in the map



# Three-Variable K-Map

- Using Gray code: 00 01 11 10 not 00 01 10 11
  ➔ only one bit changes from a adjacent column to the next.

| | | | |
|---|---|---|---|
| $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| $m_4$ | $m_5$ | $m_7$ | $m_6$ |




- See the case of $m_5 + m_7$ to see the reason of using gray code

## EXAMPLE 3.1

Simplify the Boolean function

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

## EXAMPLE 3.2

Simplify the Boolean function

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

## EXAMPLE 3.3

Simplify the Boolean function

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

## EXAMPLE 3.4

For the Boolean function

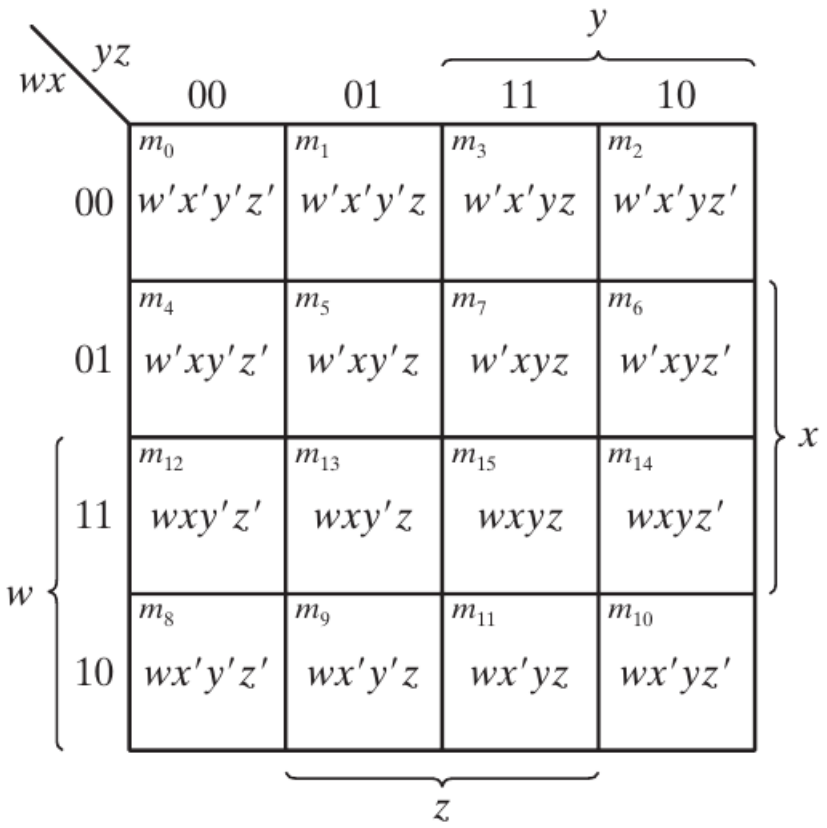
$$F = A'C + A'B + AB'C + BC$$

(a) Express this function as a sum of minterms.

(b) Find the minimal sum-of-products expression.

# Four Variable K-Map

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



## EXAMPLE 3.5

Simplify the Boolean function

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

## EXAMPLE 3.6

Simplify the Boolean function

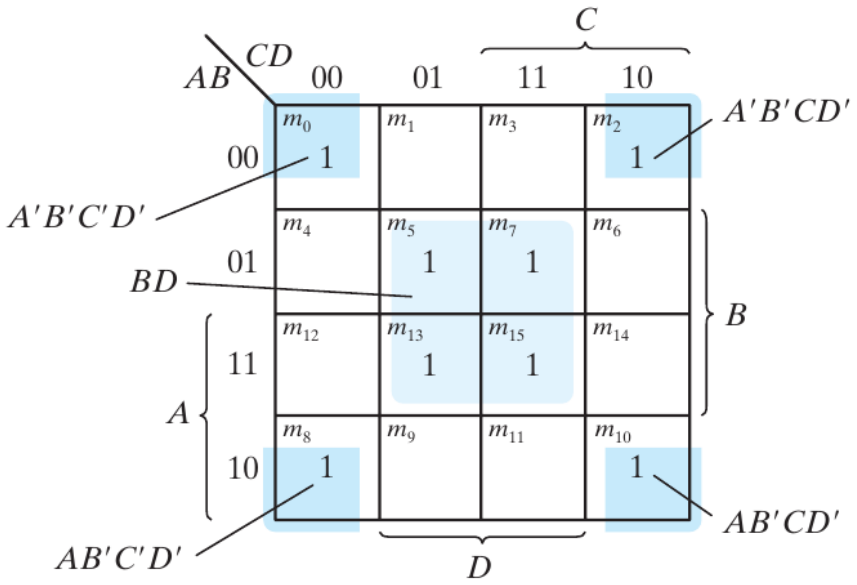
$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

# Prime Implicant

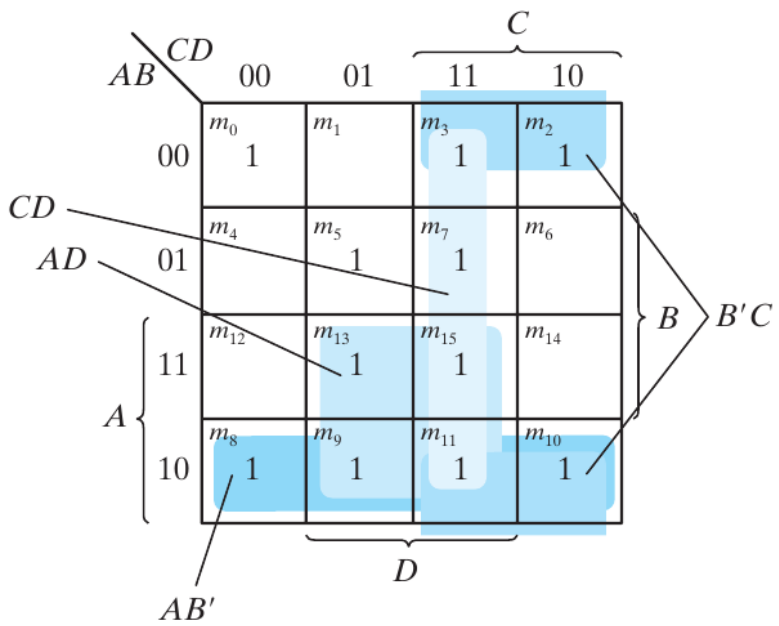
- In choosing adjacent squares in a map, we must ensure that
  (1) all the minterms of the function are covered
  (2) the number of terms in the expression is minimized
  (3) there are no redundant terms
- Sometimes there may be two or more expressions that satisfy the simplification criteria.
- Procedure for combining squares in the map is more systematic by understanding the meaning of two special types of terms.
- A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.
- If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be essential.

# Simplification Using Prime Implicants

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




**Essential prime implicants BD and B'D'**




**Prime implicants CD, B'C, AD and AB**

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

# Procedure for Finding Simplified Expression

- The simplified expression is obtained from determining the logical sum of all the essential prime implicants.
- Then, plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants.

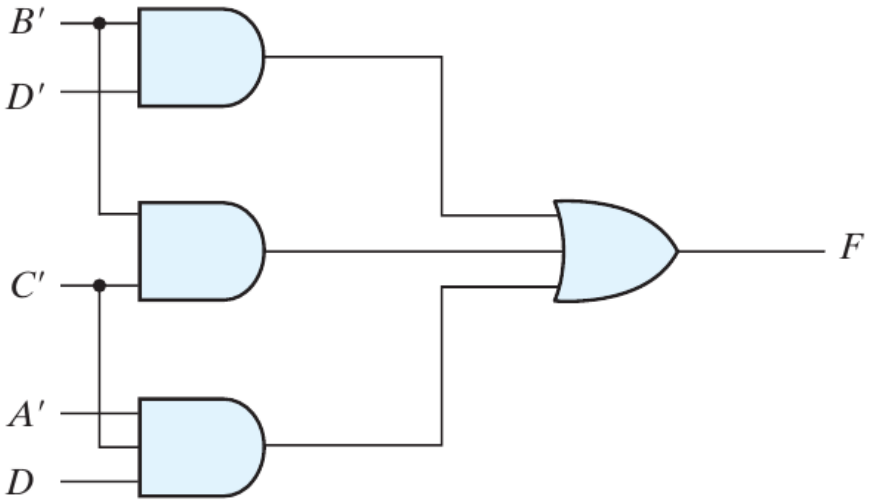
# Product-of-Sums Simplification

**EXAMPLE 3.7**

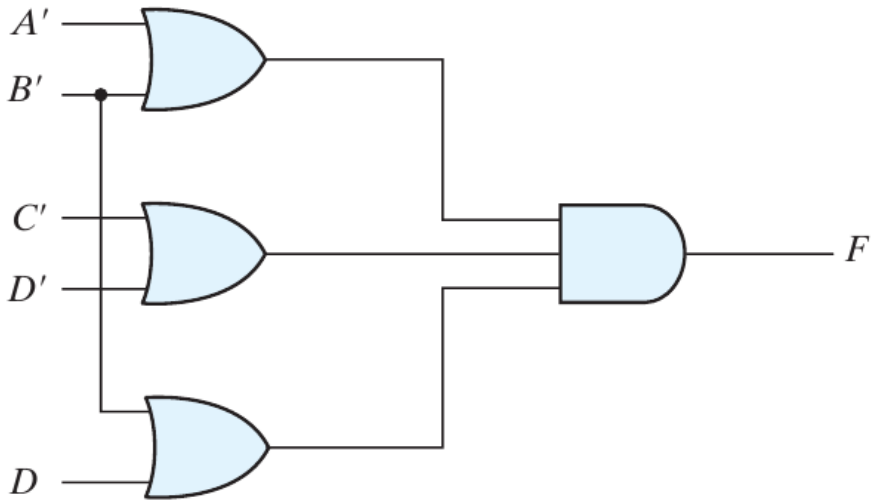
Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:

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

# Gate Implementations




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




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

- ANDs are connected to a single OR for sum-of-products
- ORs are connected to a single AND for product-of-sums.

# Practice Exercise 3.8

- Simplify the following Boolean Function into sum-of-products and product-of-sums. Derive the truth table of F.

$$F(w,x,y,z) = \Sigma(0, 2, 8, 10, 12, 13, 14)$$

# Don’t Care Condition

- In practice, in some applications the function is not specified for certain combinations of the variables.
- For example, the four-bit binary code for the decimal digits has six combinations that are not used and consequently are considered to be unspecified.
- Functions that have unspecified outputs for some input combinations are called *incompletely specified functions*.

# Don’t Care Condition in K-Map

- To distinguish the don’t-care condition from 1’s and 0’s, an X is used.
- In choosing adjacent squares to simplify the function in a map, the don’t-care minterms may be assumed to be either 0 or 1.

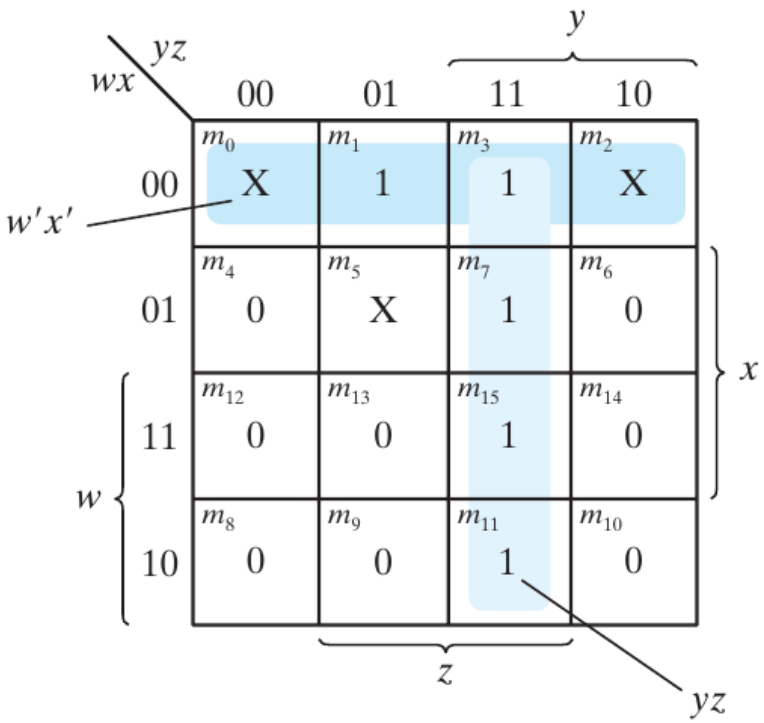
## EXAMPLE 3.8

Simplify the Boolean function

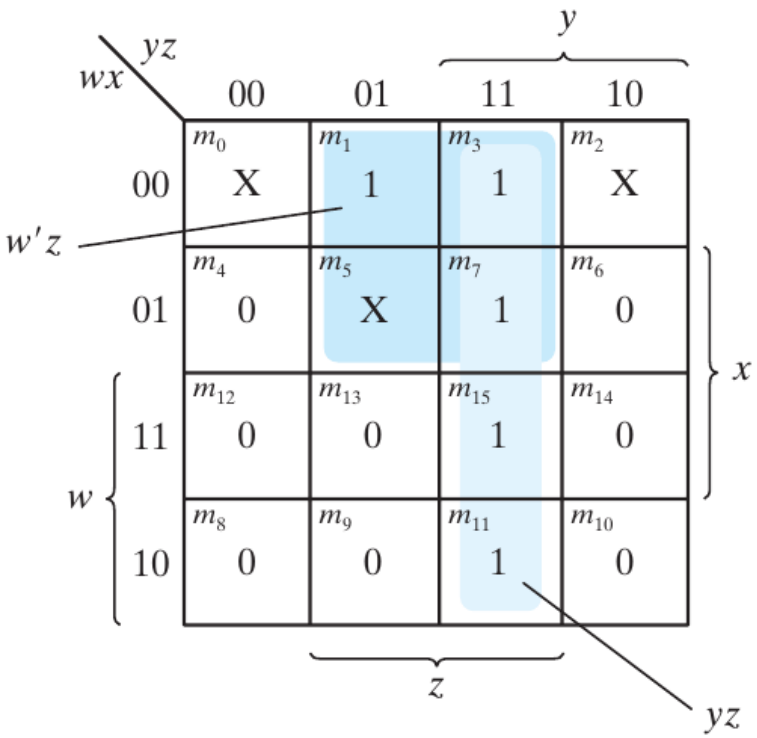
$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions

$$d(w, x, y, z) = \Sigma(0, 2, 5)$$




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



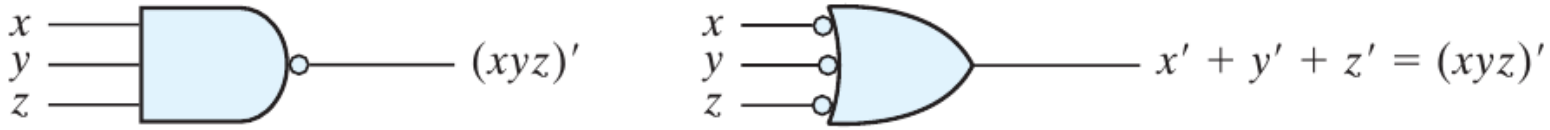

(b) $F = yz + w'z$

# NAND/NOR Implementation

- Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates.
- NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.
- Thus, rules and procedures have been developed for the conversion from Boolean functions given in terms of AND, OR, and NOT into equivalent NAND and NOR logic diagrams.

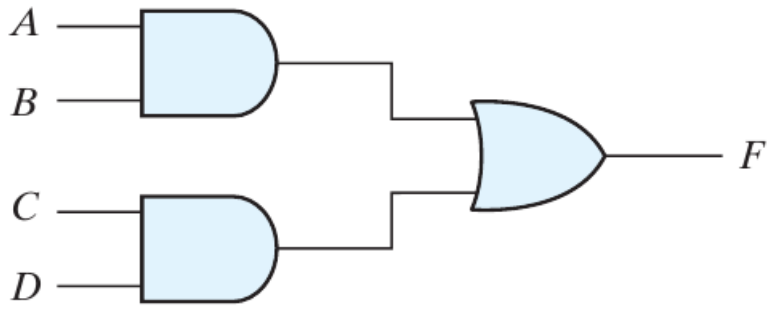
# NAND Circuits

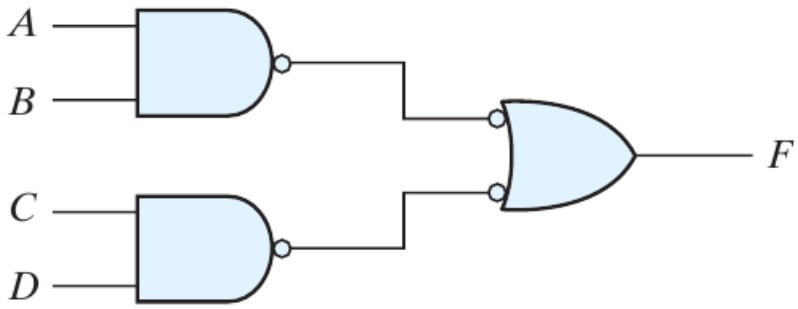
- The NAND gate is said to be a universal gate because any logic circuit can be implemented with it
- Boolean function can be implemented with NAND gates by 1) obtaining the simplified Boolean function in terms of Boolean operators and 2) then convert it into NAND logic.
- To facilitate the conversion to NAND logic, it is convenient to define an alternative graphic symbol for the gate as:

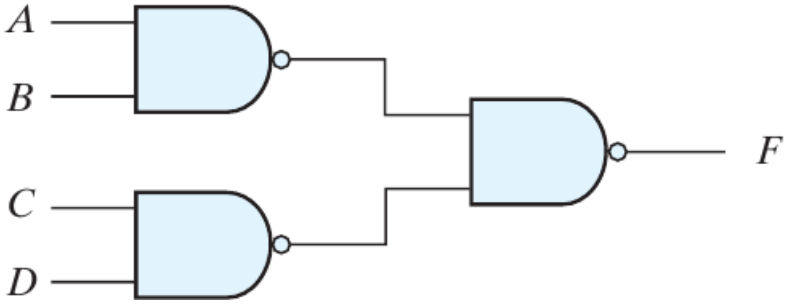



(a) AND-invert (b) Invert-OR

# F = AB + CD



## EXAMPLE 3.9

Implement the following Boolean function with NAND gates:

$$F(x, y, z) = (1, 2, 3, 4, 5, 7)$$

# Procedure Summary

1. Simplify the function with sum-of-products form.
2. Draw a NAND gate for each product term of the expression that has at least two literals.
3. Draw a single gate using the AND-invert or the invert-OR graphic symbol in the second level, with inputs coming from outputs of first-level gates.
4. A term with a single literal requires an inverter in the first level, while, it can be connected directly to the second level NAND gate if the single literal is complemented.

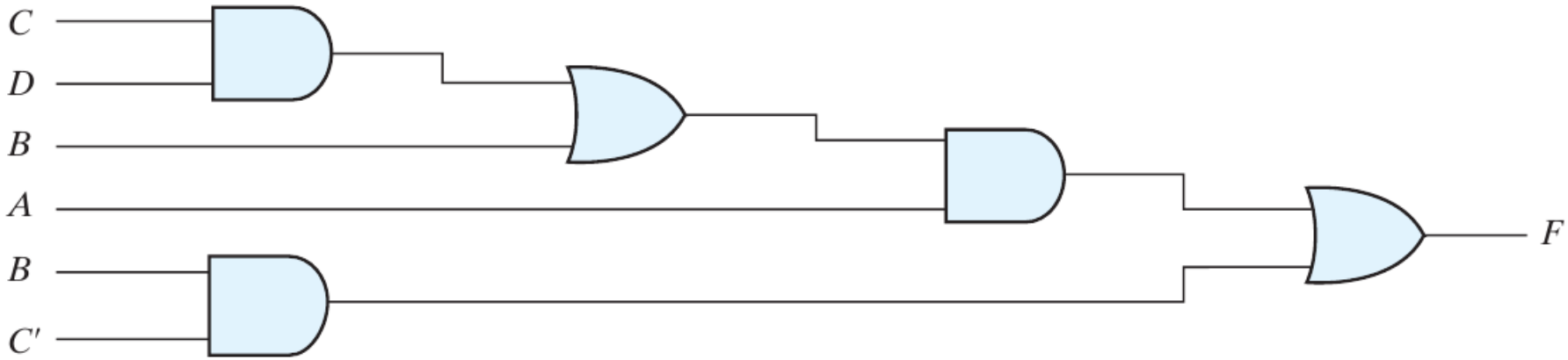
# Practice Exercise 3.10

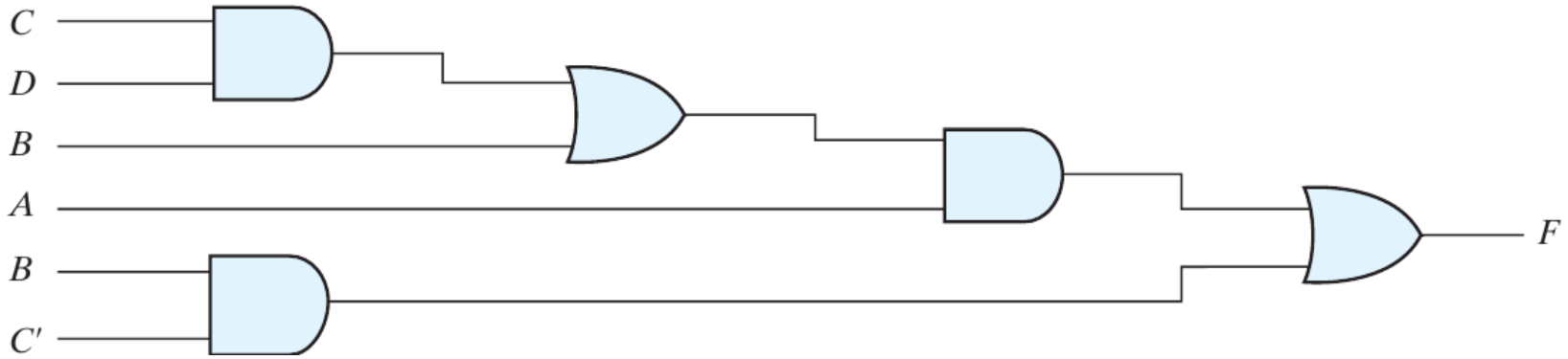
- $F(x,y,z) = \Sigma(0, 1, 3, 5, 6, 7)$

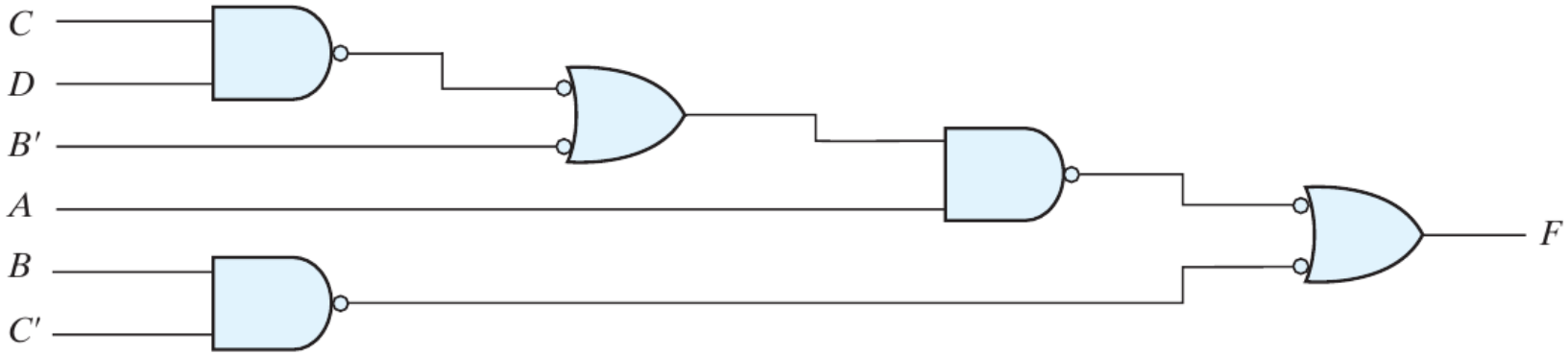
# Multilevel NAND Circuits

- Other than standard form may result in three or more level gate structures, For example,

$$F = A\,(CD + B) + BC'$$



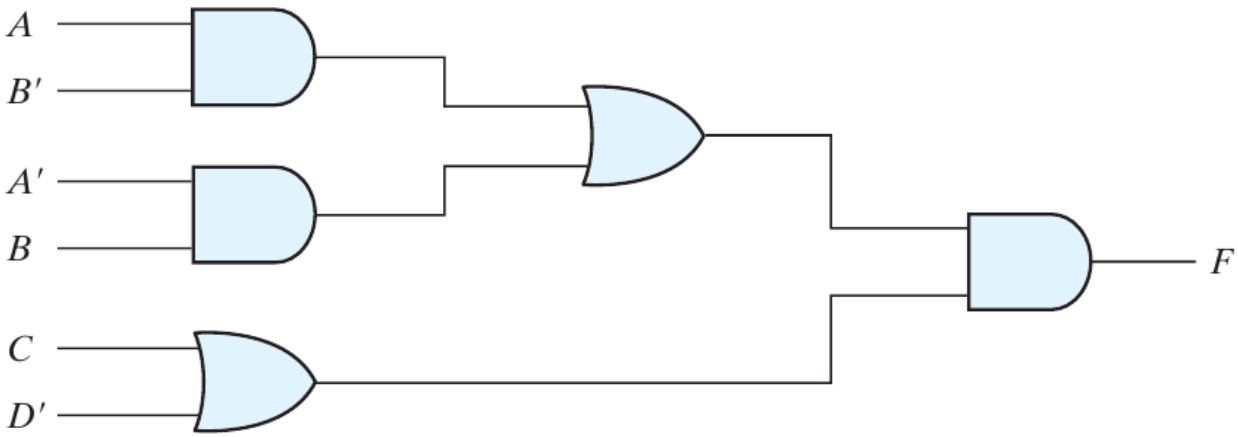

C
D
B
A
B
C′
F


C
D
B′
A
B
C′
F

# Procedure Converting Multilevel AND–OR into an All-NAND

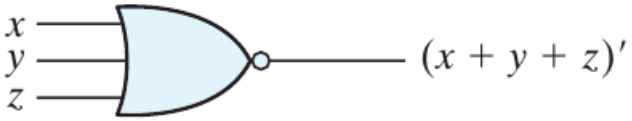
1. Convert all AND gates to NAND gates with AND-invert graphic symbols.
2. Convert all OR gates to NAND gates with invert-OR graphic symbols.
3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input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

# F=(AB'+A'B)(C+D’)

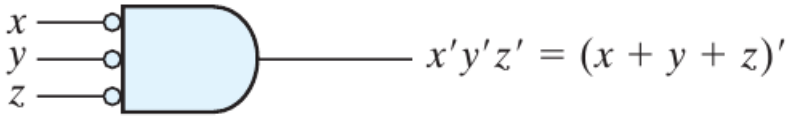


# NOR Implementation

- All procedures and rules for NOR logic are the duals of the corresponding procedures and rules for NAND logic.
- Two graphic symbols for the mixed notation for NOR




(a) OR-invert




(b) Invert-AND

# Practice Exercise 3.11

- Implement F(w,x,y,z) = (y+z’)(wx’+w’x) with NOR gates

# XOR or XNOR Function

- The exclusive-OR (XOR), denoted by the symbol is a logical operation that performs the following Boolean operation:

$$x \oplus y = xy' + x'y$$

- XNOR :

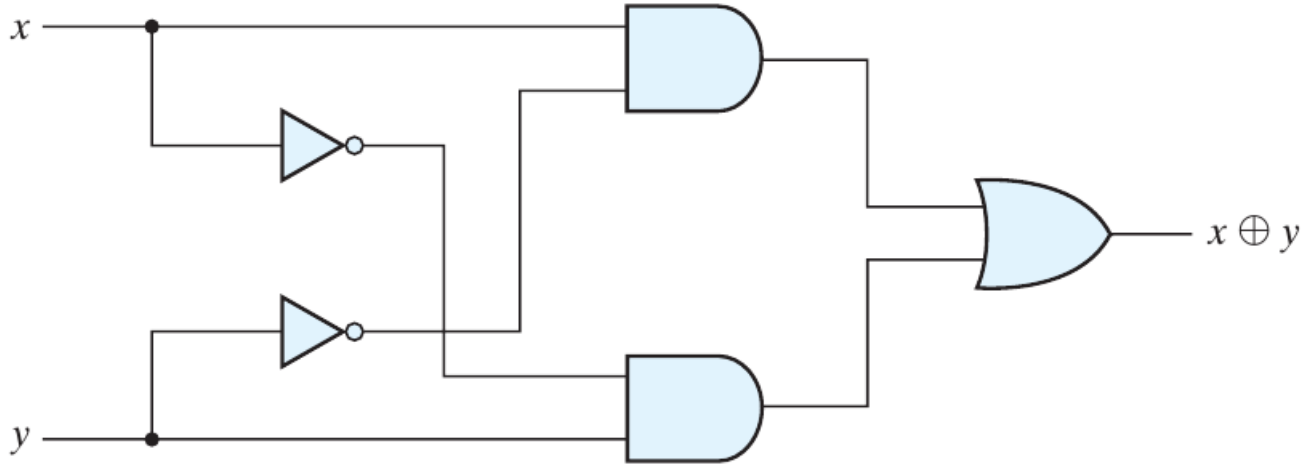
$$(x \oplus y)' = xy + x'y'$$

- Identities:

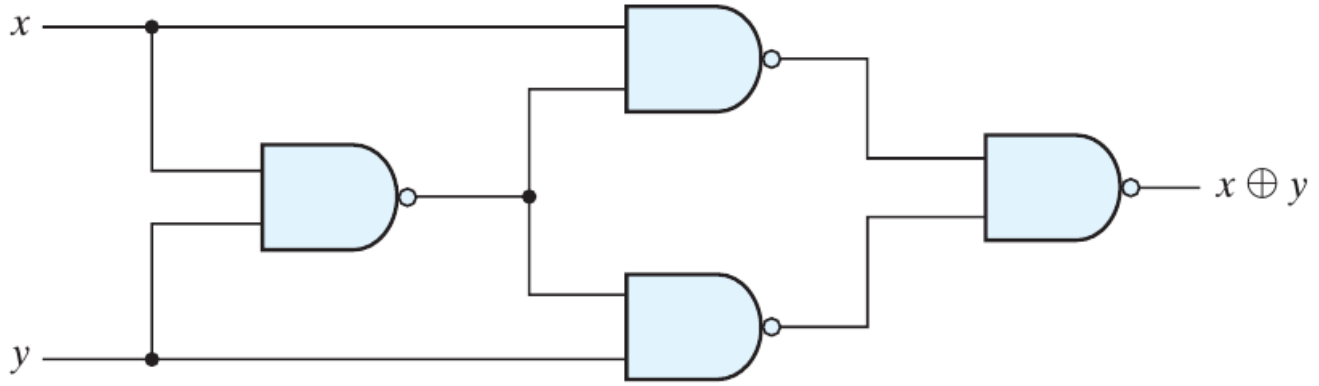
$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$
$$x \oplus x = 0$$
$$x \oplus x' = 1$$
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

- Associative and commutative

# Construction of XOR2




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




(b) Exclusive-OR with NAND gates

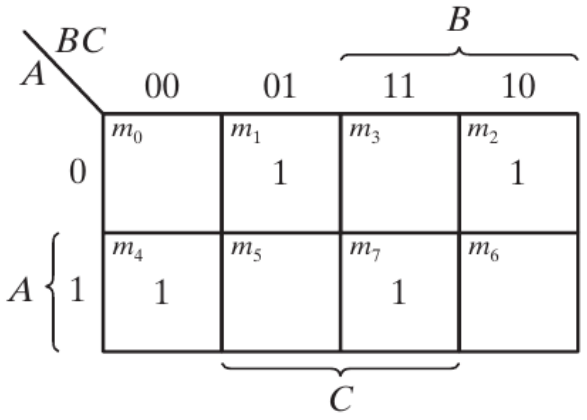
# Odd Function of Multiple Input XOR

- ODD function

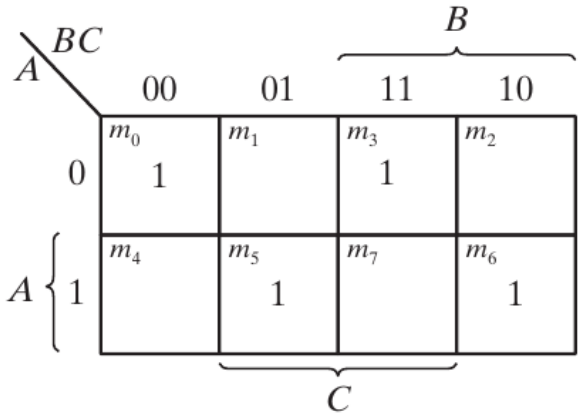
$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \Sigma(1, 2, 4, 7) \end{aligned}$$

- The Boolean expression clearly indicates that the three-variable exclusive-OR function is equal to 1 if only one variable is equal to 1 or if all three variables are equal to 1.

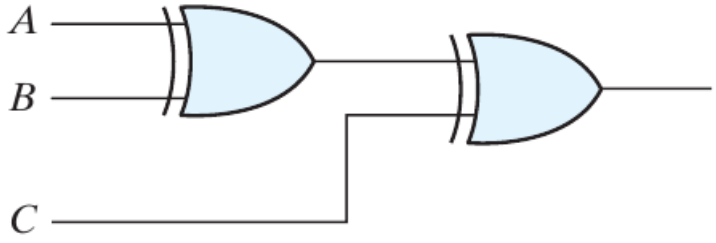
# K-Map and Gate Implementation

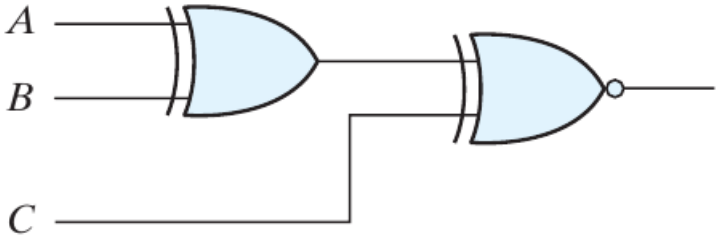



(a) Odd function $F = A \oplus B \oplus C$



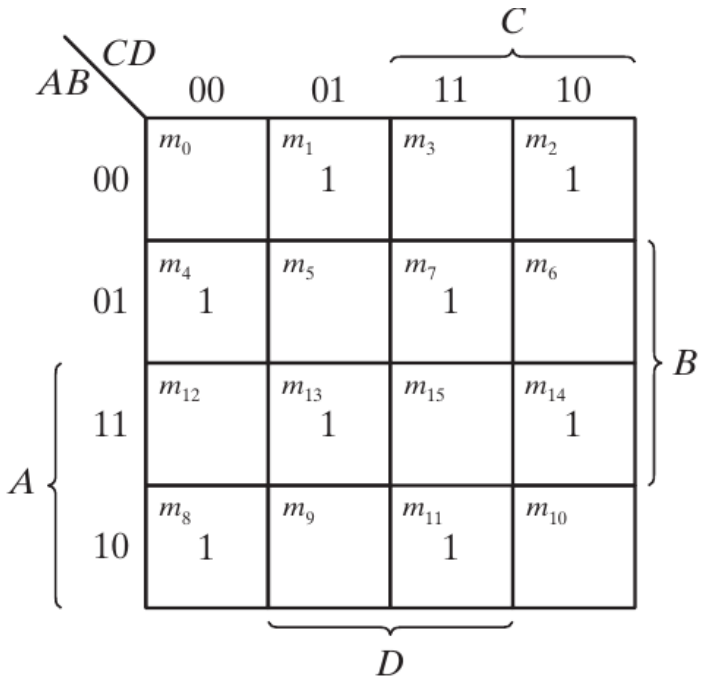

(b) Even function $F = (A \oplus B \oplus C)'$



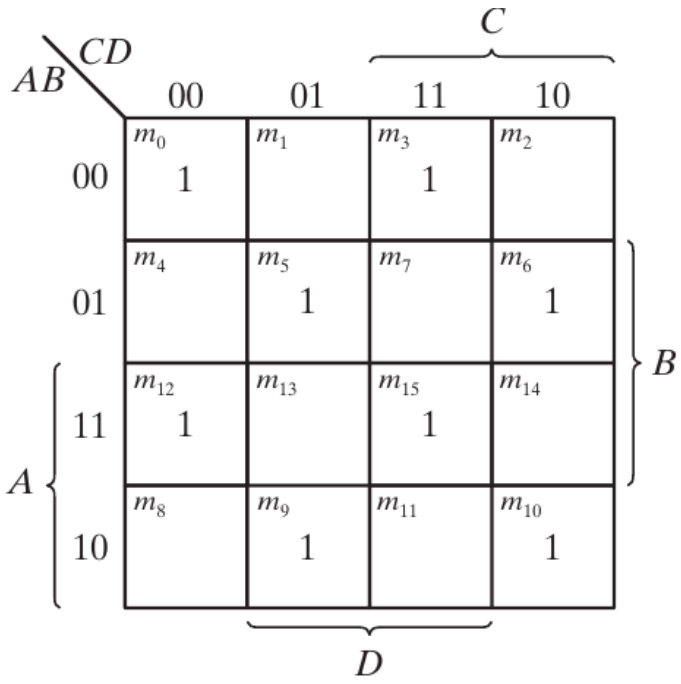



# Four-variable XOR operation

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




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

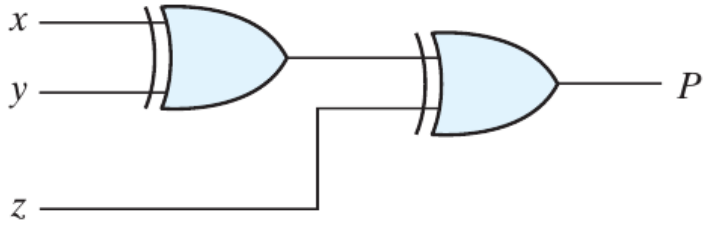



(b) Even function $F = (A \oplus B \oplus C \oplus D)'$

# Parity Generation & Check

*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 |




(a) 3-bit even parity generator

# Parity Error Check

$$C = x \oplus y \oplus z \oplus P$$

**Table 3.4**
*Even-Parity-Checker Truth Table*

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

