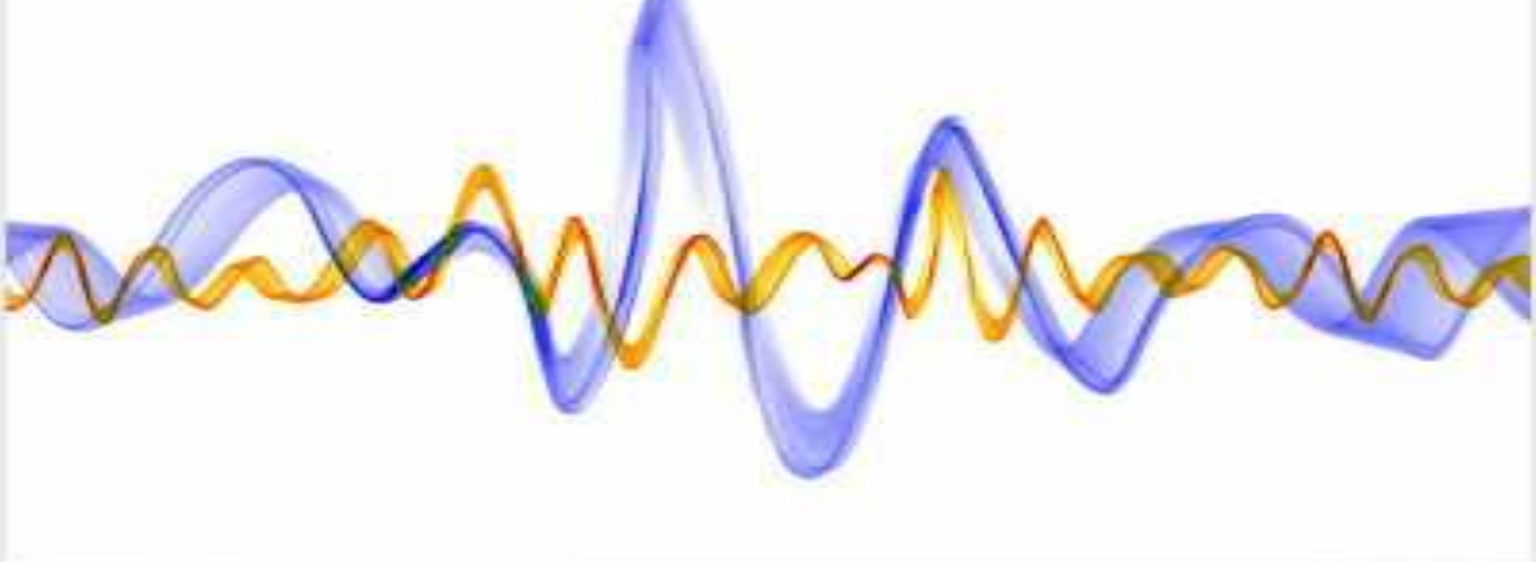
Digital Logic Design



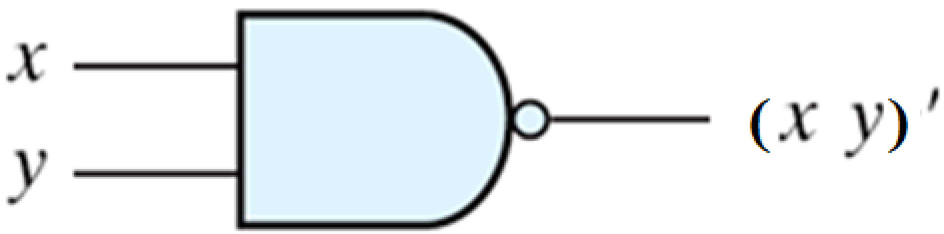
Lecture 5:
Chapter 4: Combinational Logic

Mirvat Al-Qutt, Ph.D
Computer Systems Department , FCIS,
Ain Shams University

NAND-Only Implementation

- **NAND gate is a universal gate**
  - Can implement any digital system using NAND gate only

NAND

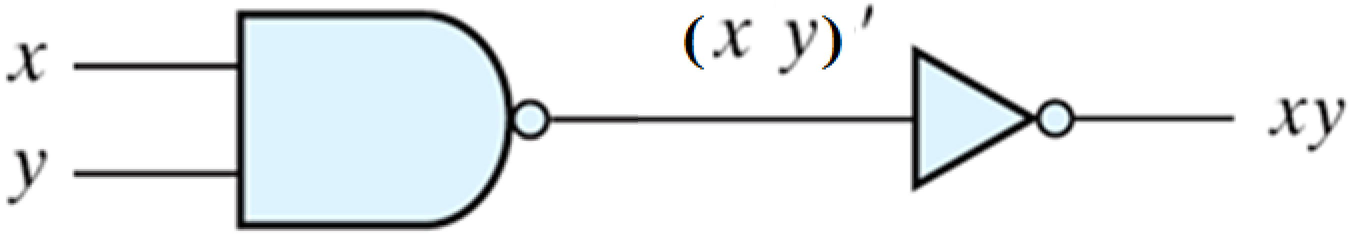


- Universal gate : we can implement all logic Operations with NAND Gates **ONLY**

NAND-Only Implementation

- **NAND gate is a universal gate**
  - Can implement any digital system using NAND gate only

AND

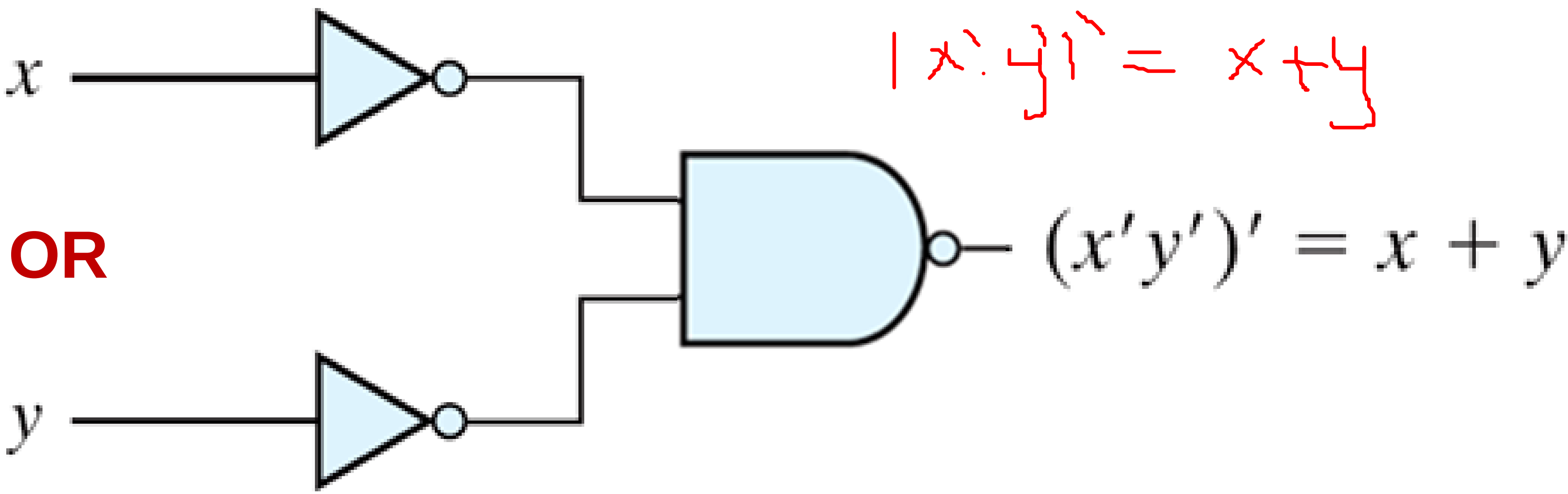


- Universal gate : we can implement all logic Operations with NAND Gates **ONLY**

NAND-Only Implementation

- **NAND gate is a universal gate**
  - Can implement any digital system using NAND gate only

OR

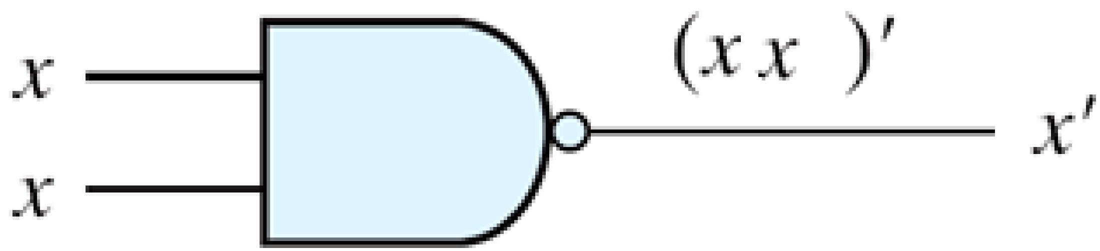


- Universal gate : we can implement all logic Operations with NAND Gates **ONLY**

NAND-Only Implementation

- **NAND gate is a universal gate**
  - Can implement any digital system using NAND gate only

Inverter



- Universal gate : we can implement all logic Operations with NAND Gates **ONLY**

NAND-Only Implementation

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

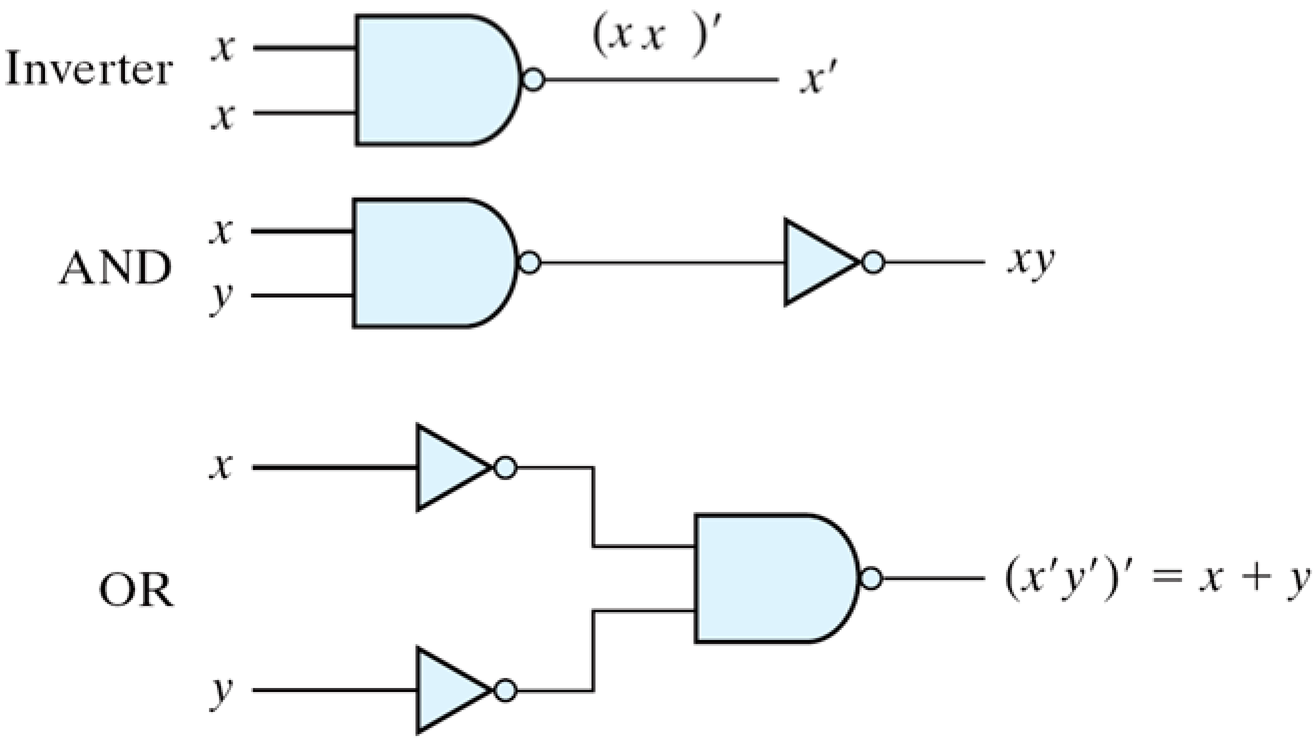
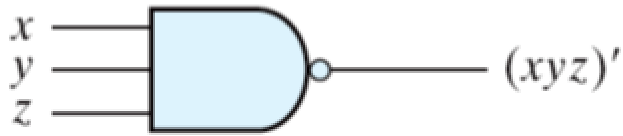


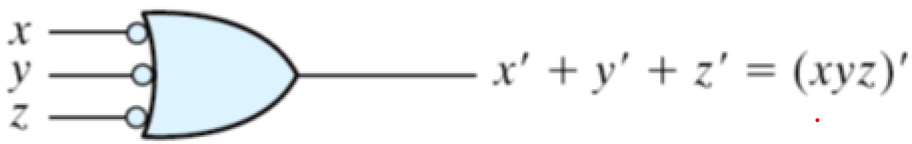
Figure 3.18 Logic Operations with NAND Gates

NAND Gate

- Two graphic symbols for a NAND gate



(a) AND-invert



(b) Invert-OR

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

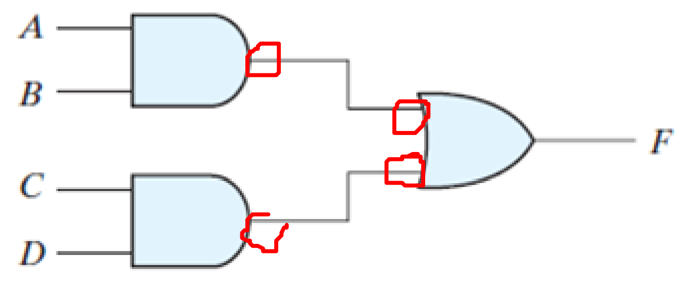
By applying DeMorgan's Theorem

Two-level NAND–Only Implementation

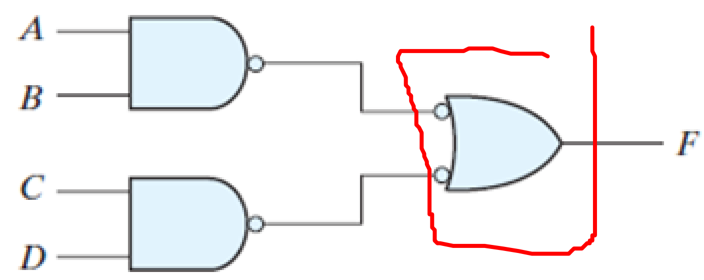
- Two-level logic
  - NAND-NAND = sum of products
  - Example: $F = AB+CD$
  - $F = ((AB)' (CD)')' = AB+CD$

Three ways to implement $F = AB + CD$

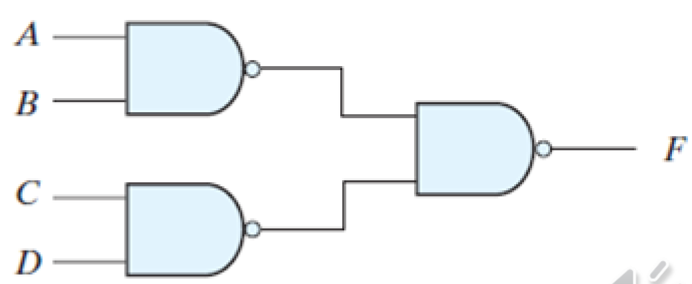
NAND–Only Implementation



(a)



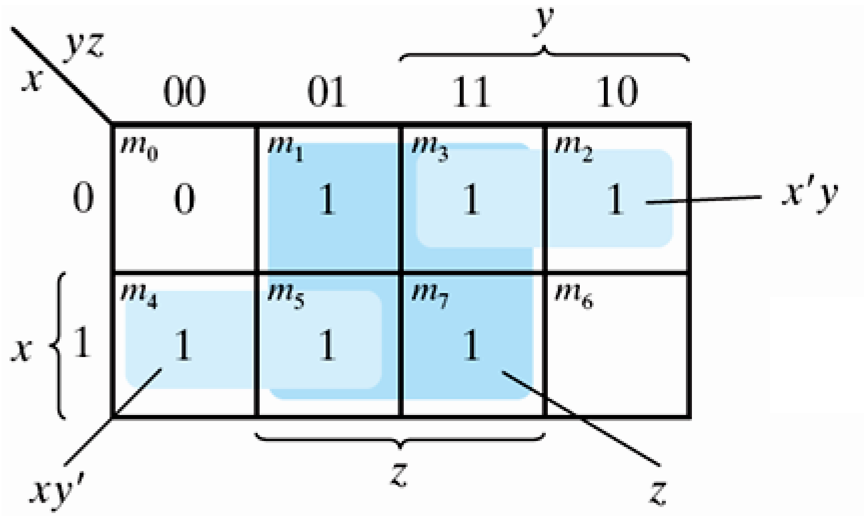
(b)



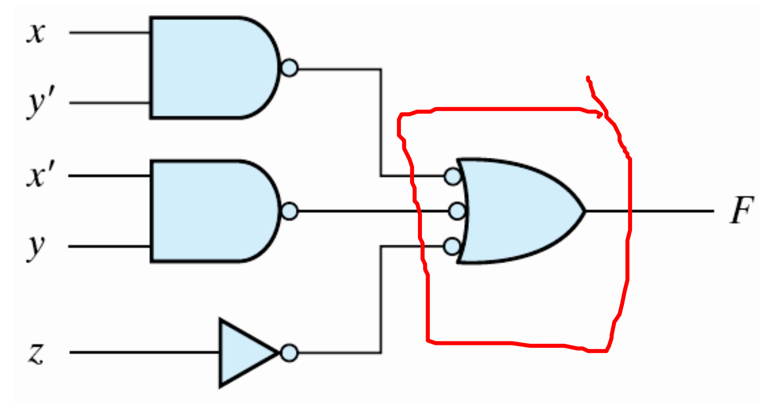
(c)

Two-level NAND–Only Implementation

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

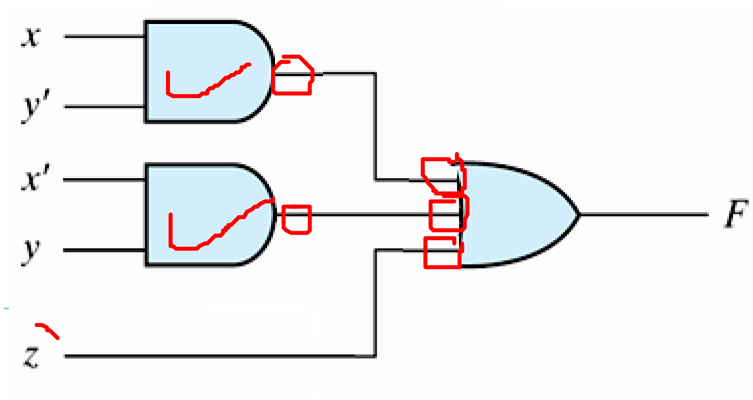


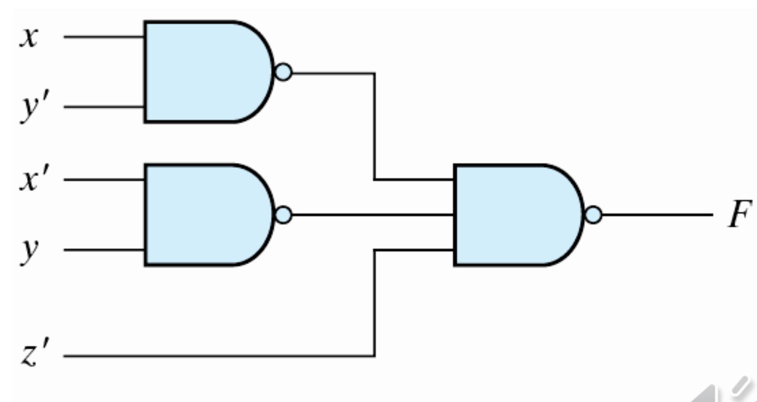
(a)



(b)

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





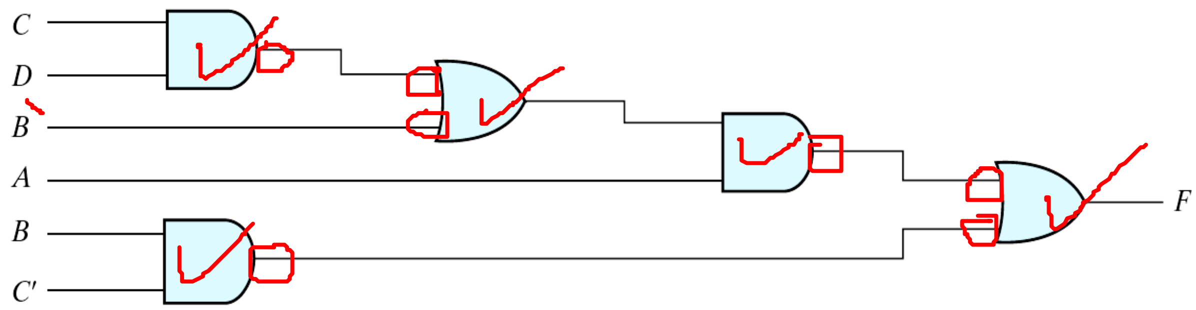
(c)

Two-level NAND–Only Implementation

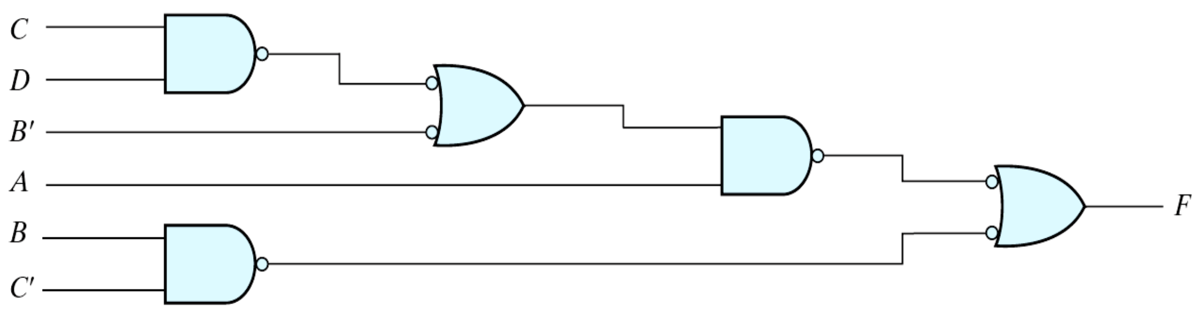
- The procedure
  1. Simplified in the form of sum of products;
  2. A NAND gate for each product term; the inputs to each NAND gate are the literals of the term (the first level);
  3. A single NAND gate for the second sum term (the second level);
  4. A term with a single literal requires an inverter in the first level.

Multilevel NAND Circuits

- Boolean function implementation
  - AND-OR logic → NAND-NAND logic
    - AND → NAND + inverter
    - OR: inverter + OR = NAND



(a) AND–OR gates



(b) NAND gates

Figure 3.22 Implementing $F = A(CD + B) + BC'$

NAND-Only Implementation

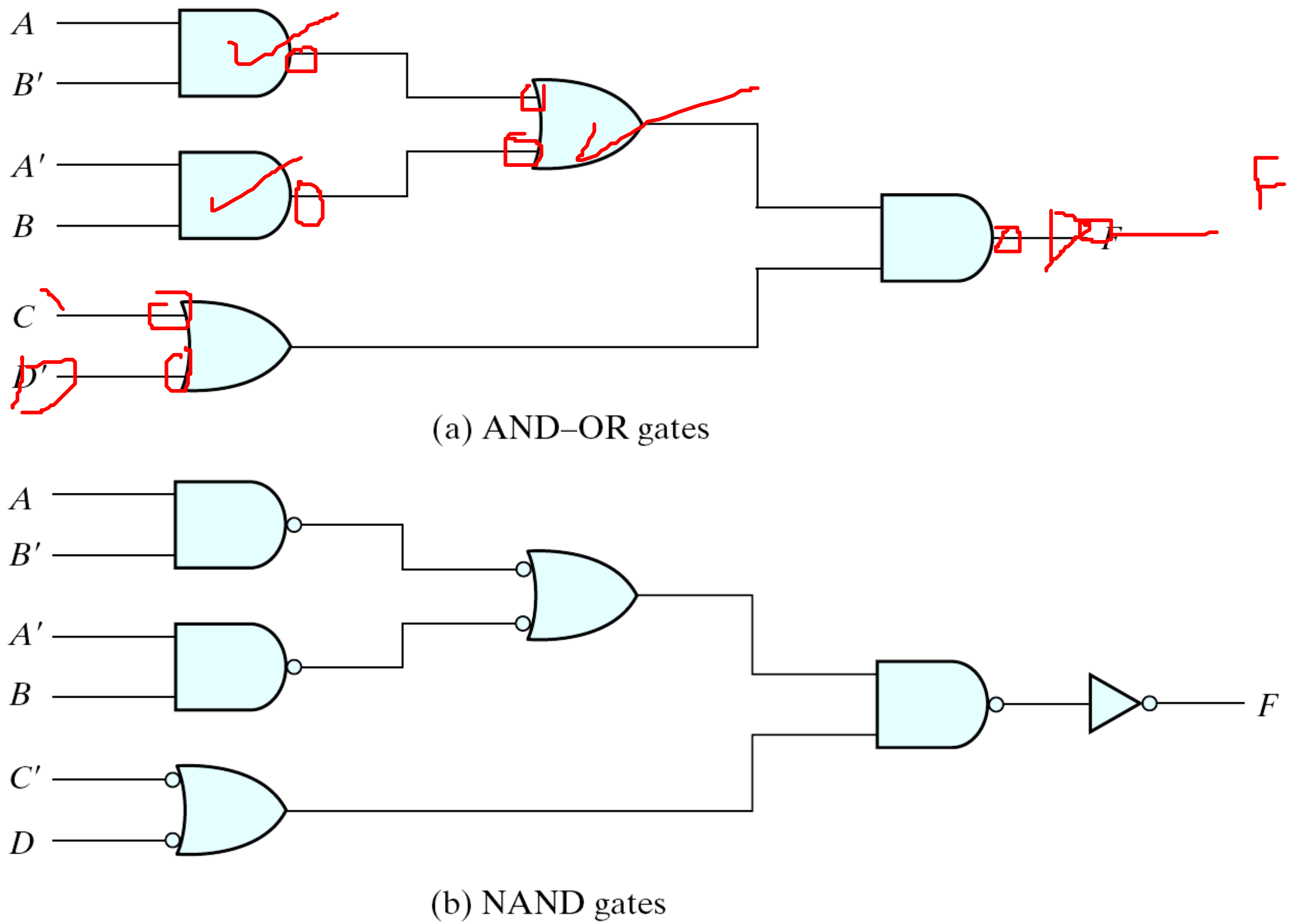
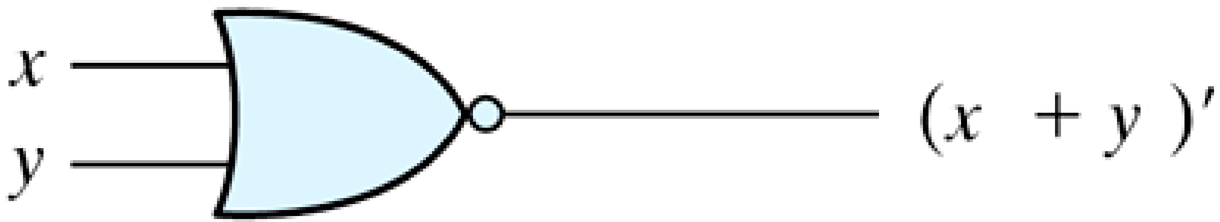


Figure 3.23 Implementing $F = (AB' + A'B)(C + D')$

NOR-Only Implementation

- **NOR gate is a universal gate**
  - Can implement any digital system using NOR gate only

NOR

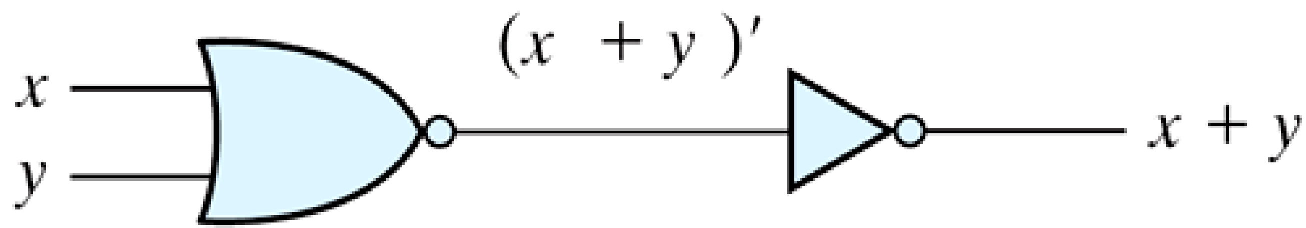


- Universal gate : we can implement all logic Operations with NOR Gates **ONLY**

NOR-Only Implementation

- **NOR gate is a universal gate**
  - Can implement any digital system using NOR gate only

OR



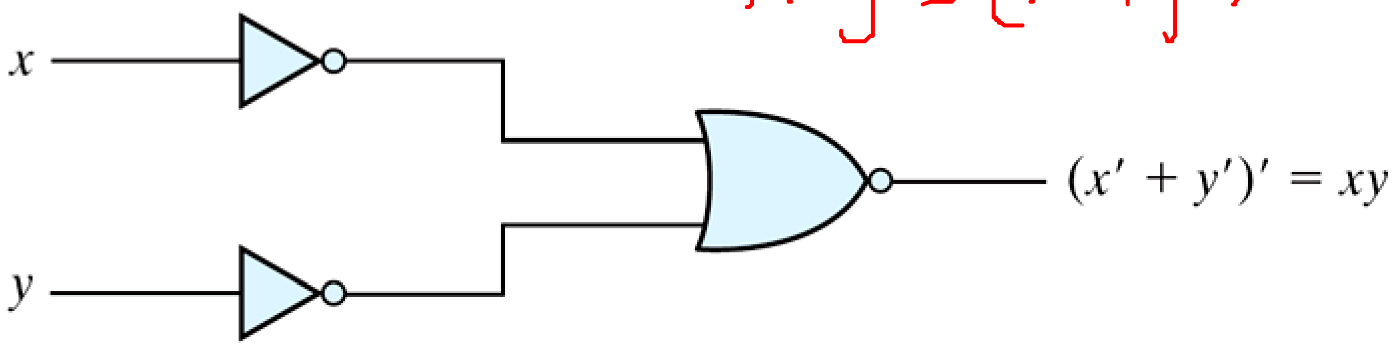
- Universal gate : we can implement all logic Operations with NOR Gates **ONLY**

NOR-Only Implementation

- **NOR gate is a universal gate**
  - Can implement any digital system using NOR gate only

$x \cdot y = (x' + y')'$

AND

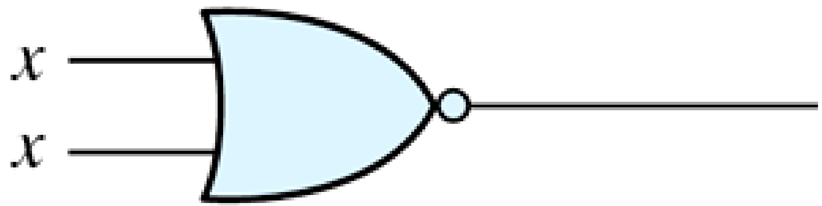


- Universal gate : we can implement all logic Operations with NOR Gates **ONLY**

NOR-Only Implementation

- **NOR gate is a universal gate**
  - Can implement any digital system using NOR gate only

Inverter



$(x + x)' = x'$

- Universal gate : we can implement all logic Operations with NOR Gates **ONLY**

NOR-Only Implementation

- **NOR gate is a universal gate**

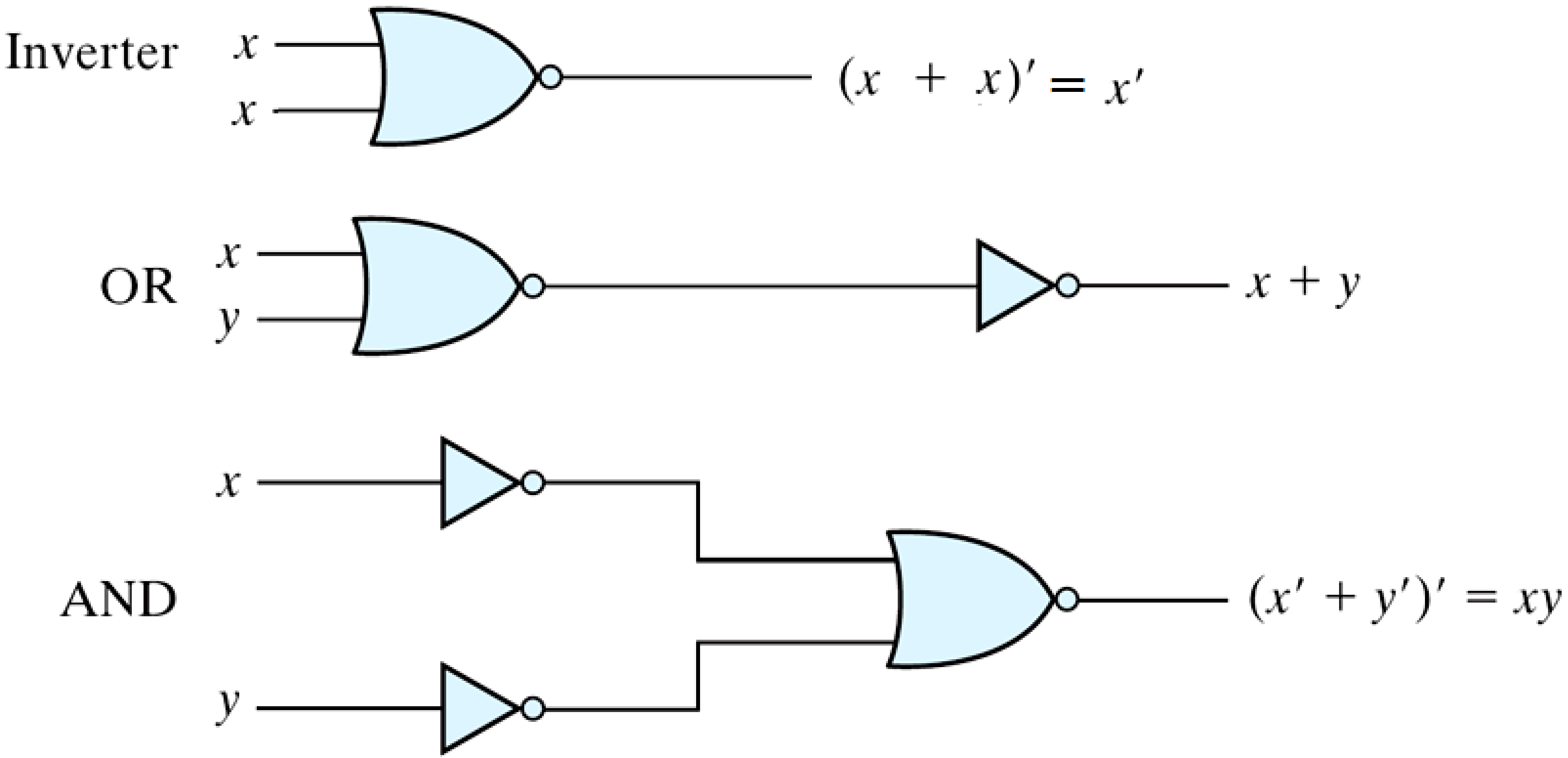
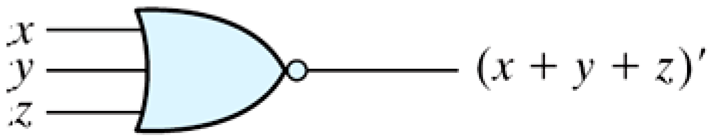


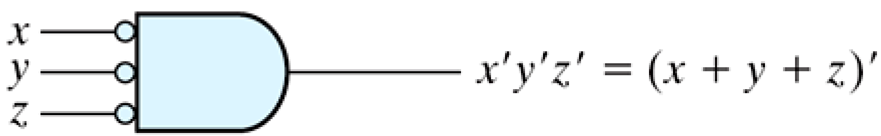
Figure 3.24 Logic Operation with NOR Gates

NOR-Only Implementation

- Two graphic symbols for a **NOR** gate



(a) OR-invert



(b) Invert-AND

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

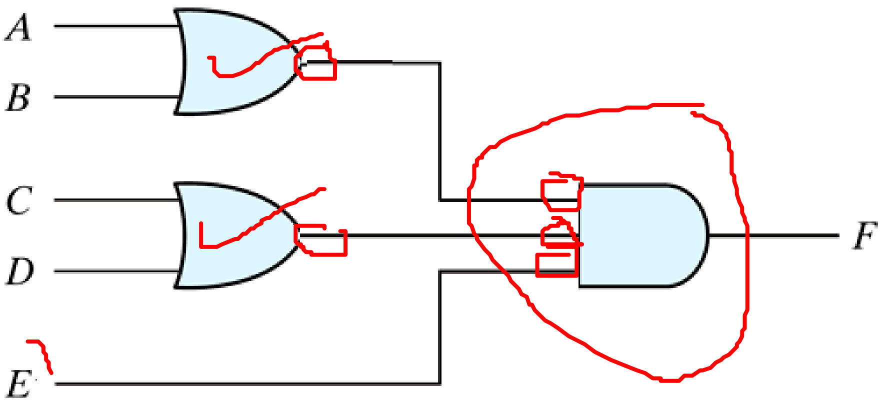
By applying DeMorgan's Theorem

Figure 3.25 Two Graphic Symbols for NOR Gate

NOR-Only Implementation

- Two graphic symbols for a **NOR** gate

Example: $F = (A + B)(C + D)E$



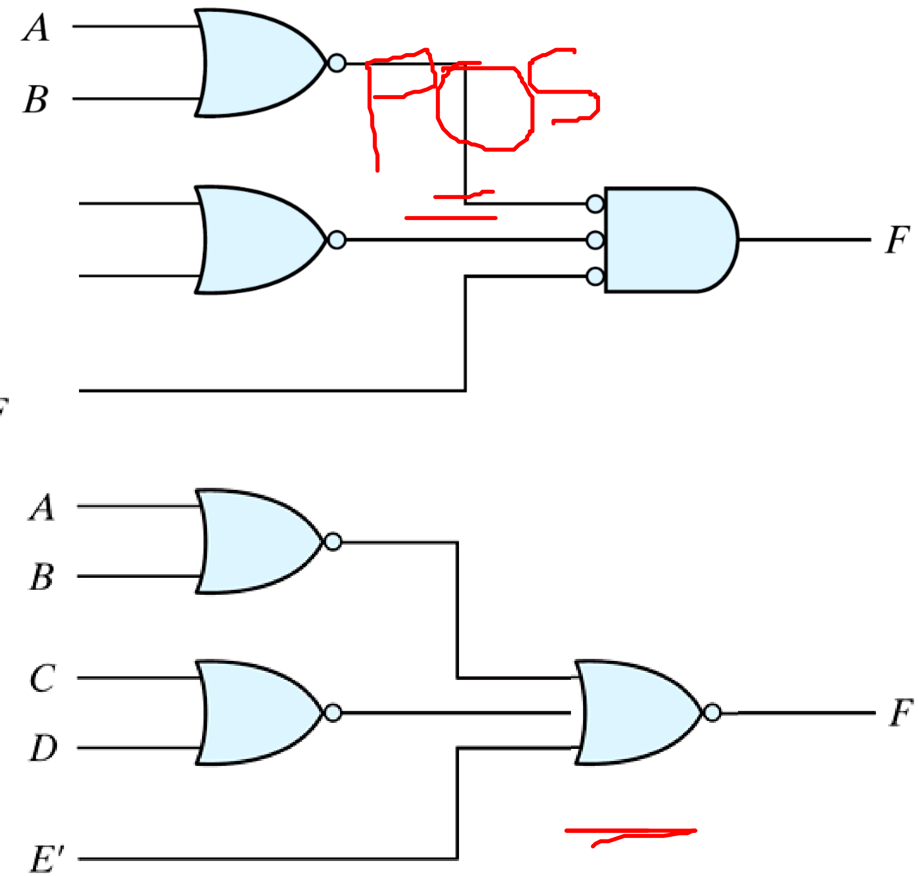
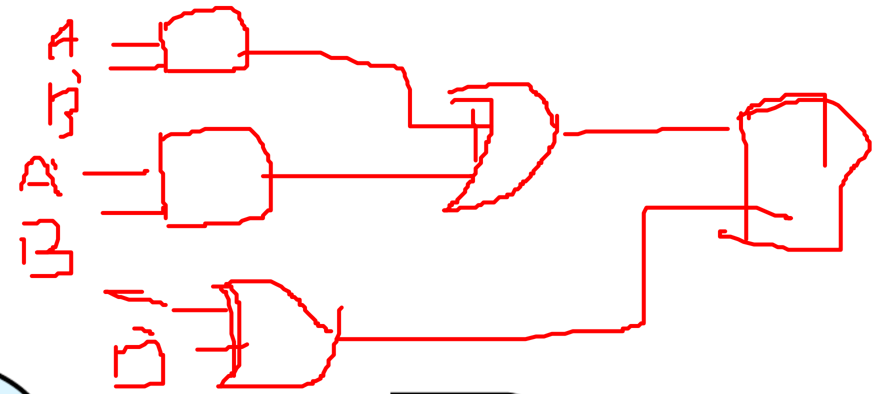


Figure 3.26 Implementing $F = (A + B)(C + D)E$

NOR-Only Implementation

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



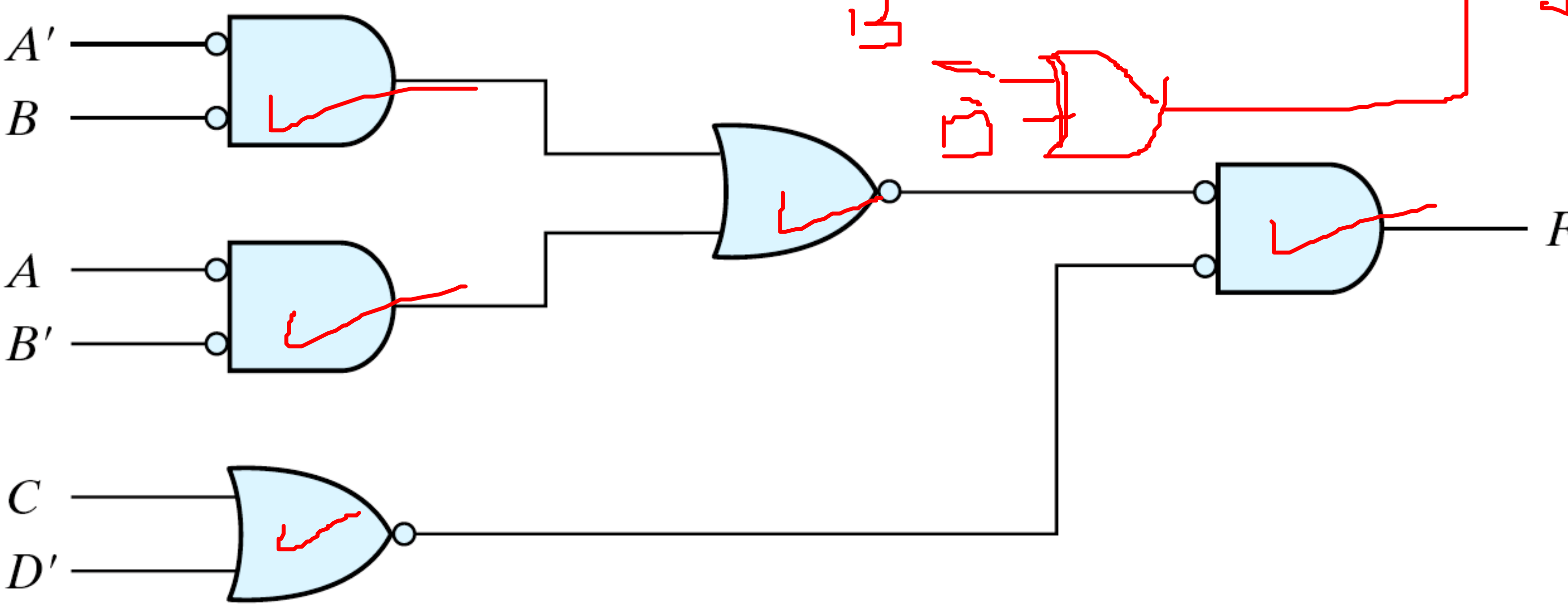


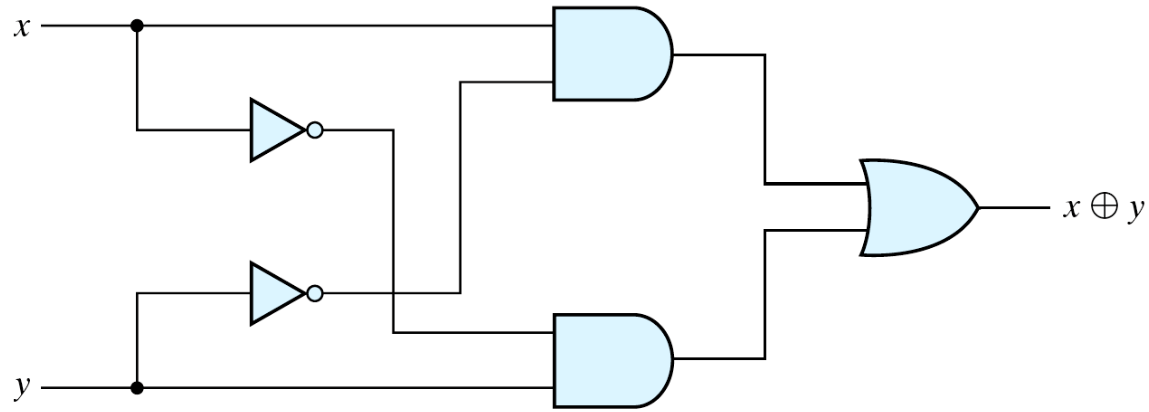
Figure 3.27 Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

Exclusive-OR Function

Exclusive-OR (XOR)	$x \oplus y = xy' + x'y$
Exclusive-NOR (XNOR)	$(x \oplus y)' = (x \odot y) = xy + x'y'$
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	$A \oplus B = B \oplus A$
Associative	$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

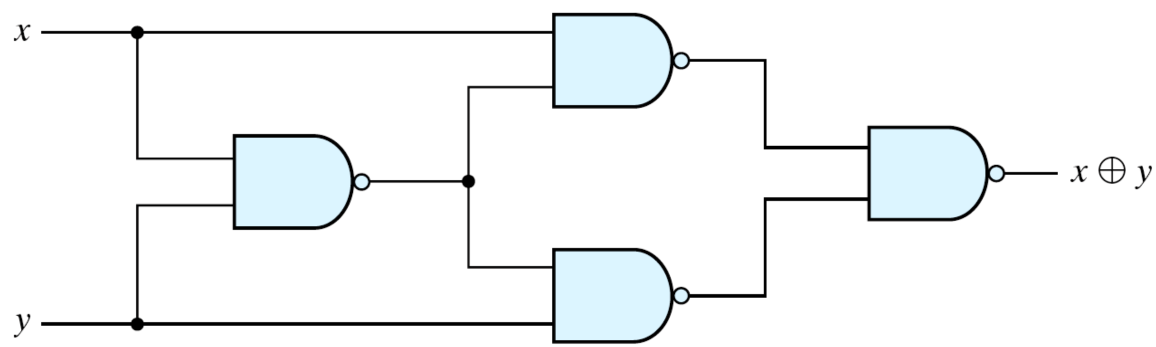
Exclusive-OR Implementations

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



(a) With AND–OR–NOT gates

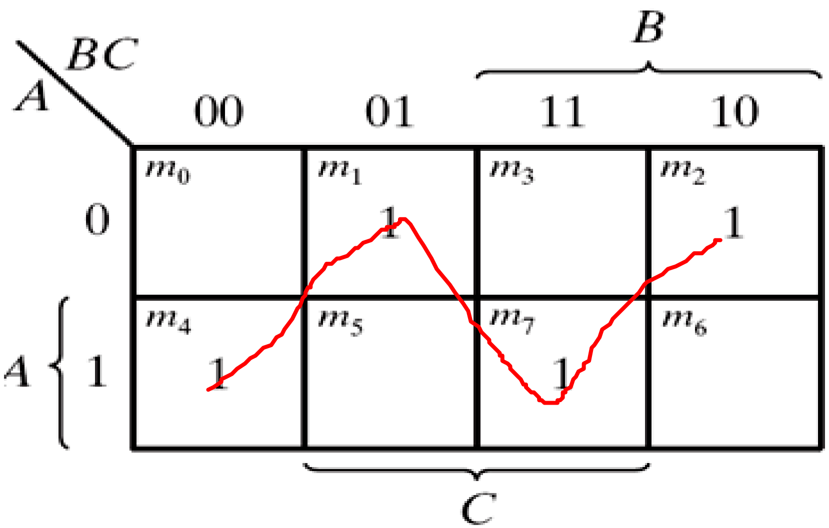
- $x \oplus y = (x'+y')x + (x'+y')y$



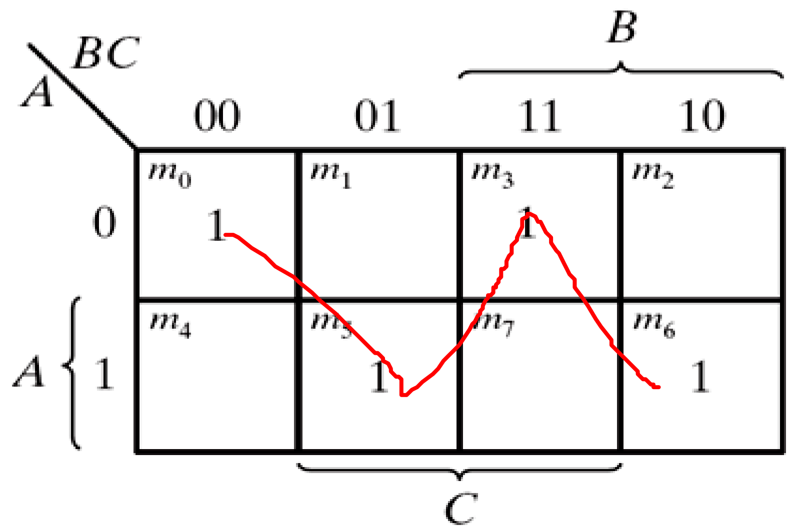
(b) With NAND gates

Odd Function

- $A \oplus B \oplus C = (AB'+A'B)C' + (AB+A'B')C$
- $= AB'C'+A'BC'+ABC+A'B'C = \Sigma(1, 2, 4, 7)$



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



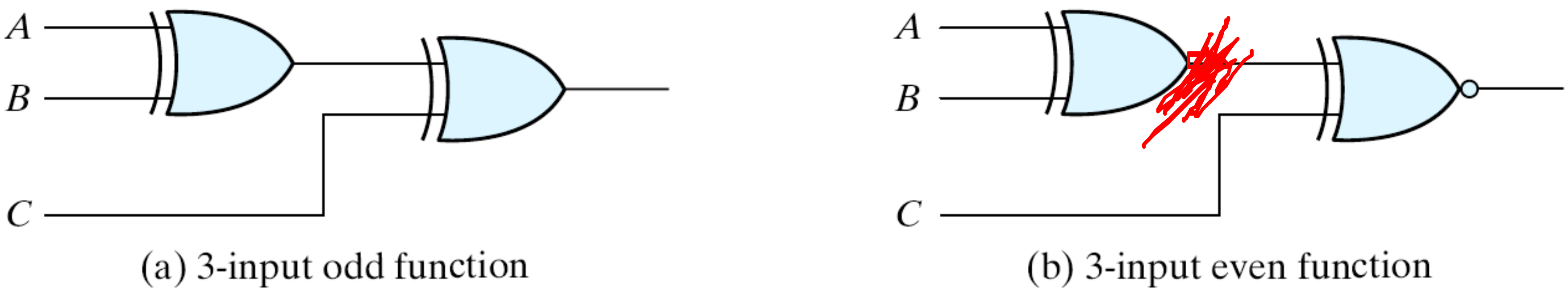
(b) Even function $F = (A \oplus B \oplus C)'$

XOR is a odd function → an odd number of 1's, then *F* = 1.

XNOR is a even function → an even number of 1's, then *F* = 1.

XOR and XNOR

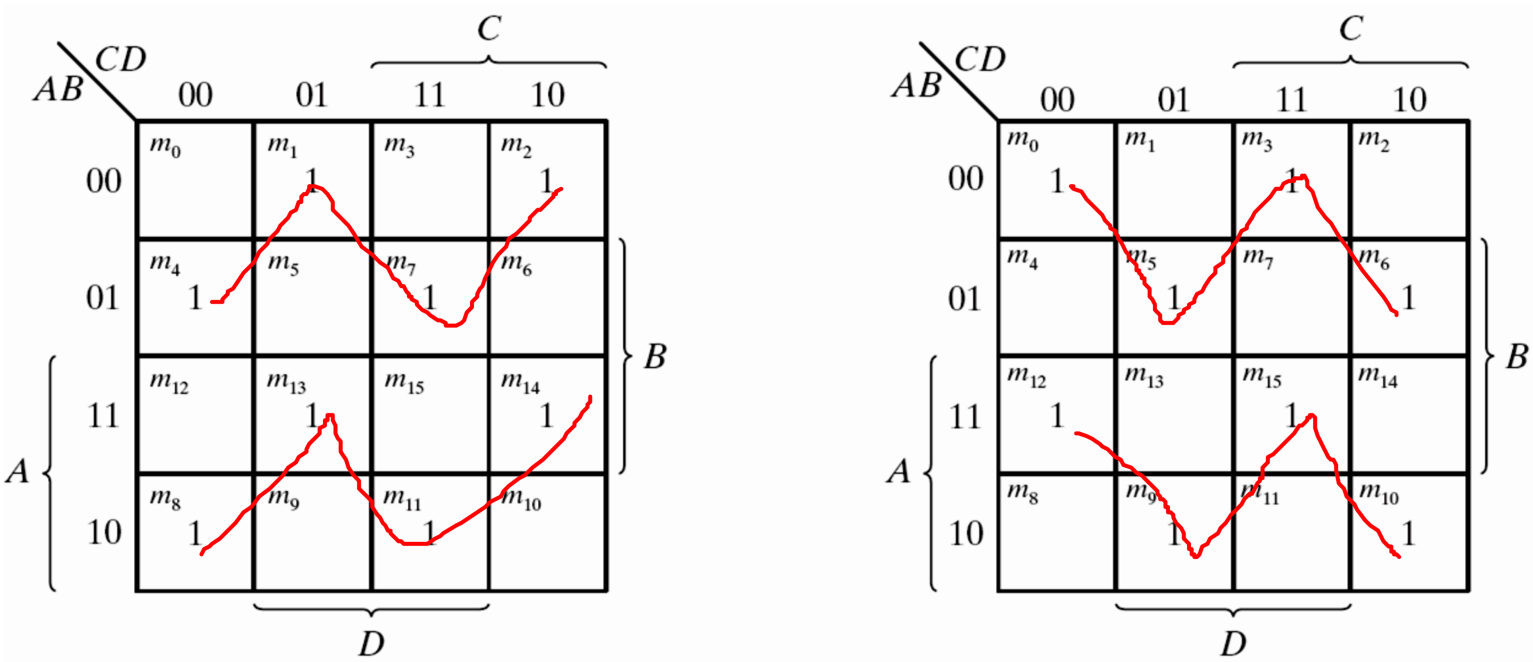
- Logic diagram of odd and even functions



Logic Diagram of Odd and Even Functions

Four-variable Exclusive-OR function

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



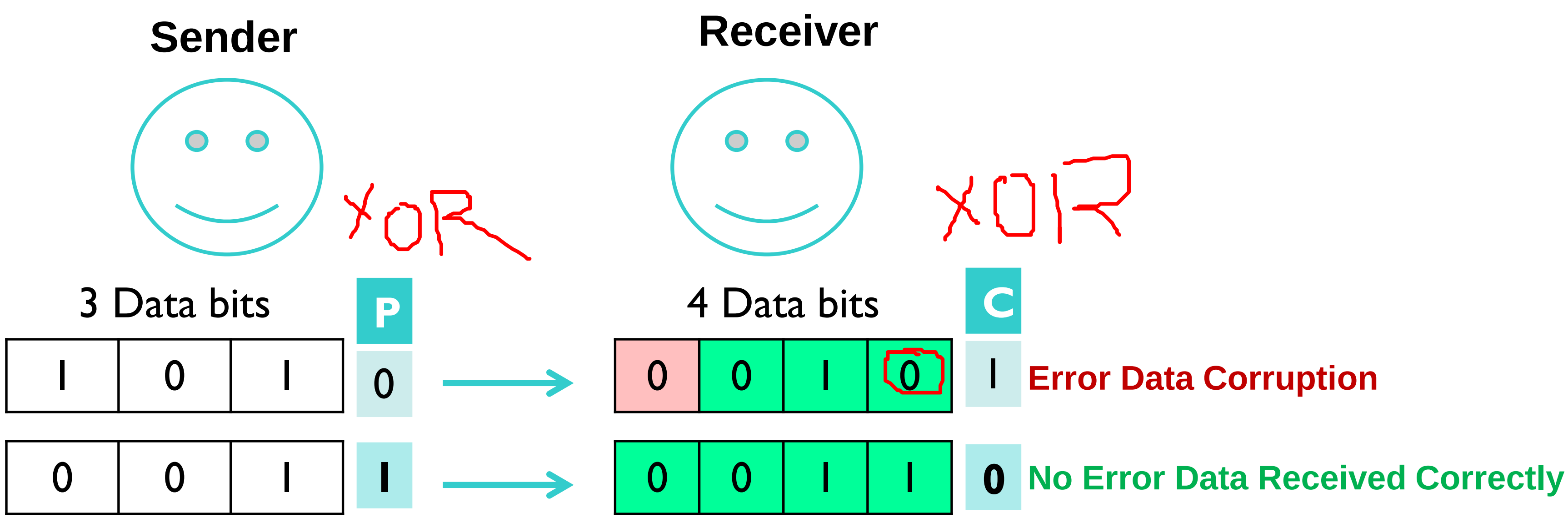
(a) Odd function $F = A \oplus B \oplus C \oplus D$

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

Exclusive-OR Function Example

One Common Application of XOR is

Parity Generation and Checking

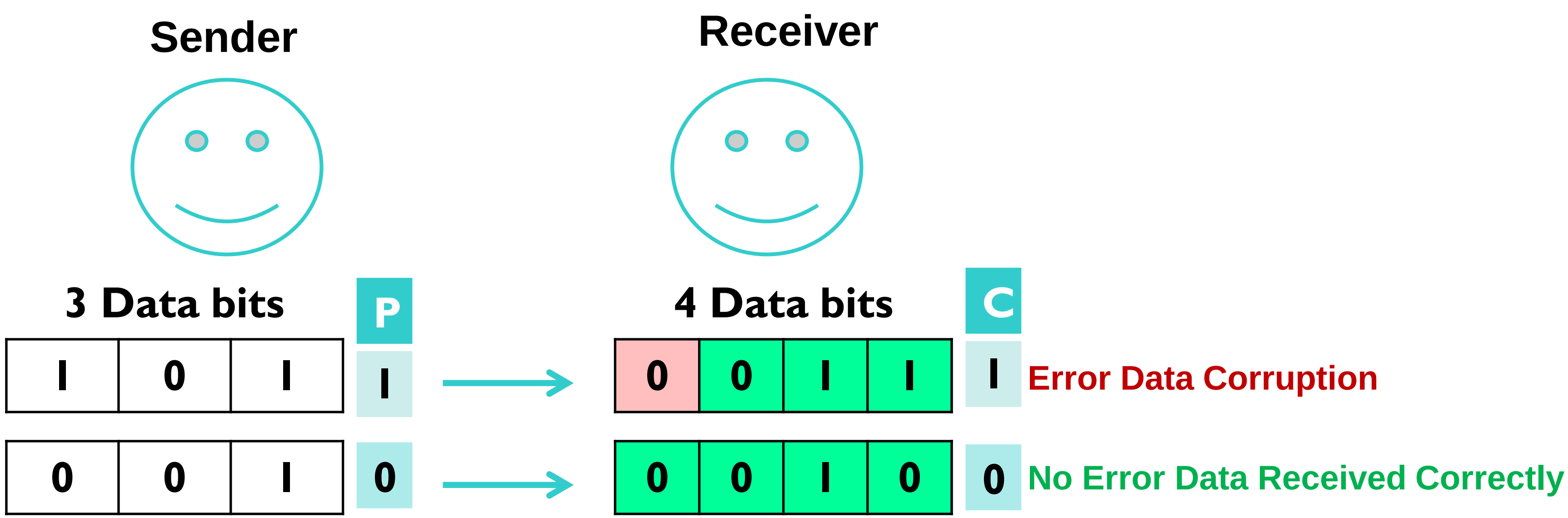


Even Parity Generator **Even Parity Checking**

Exclusive-OR Function Example

One Common Application of XOR is

Parity Generation and Checking



Odd Parity Generator

Odd Parity Checking

Even 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

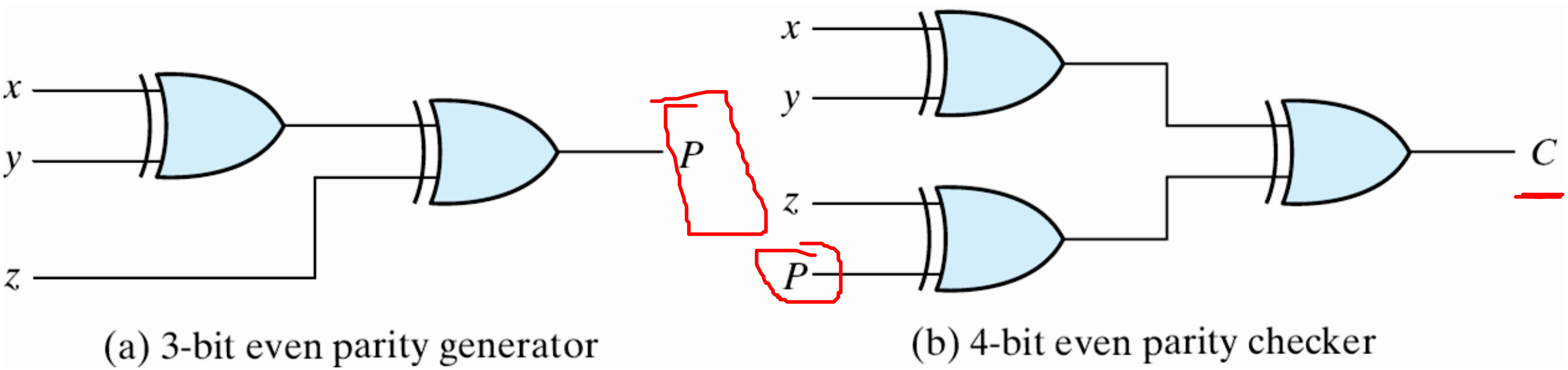


Figure 3.36 Logic Diagram of a Parity Generator and Checker

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

Combinational Logic

- **Logic circuits** for digital systems may be **combinational** or **sequential**.
- A **combinational circuit** consists of input variables, logic gates, and output variables.

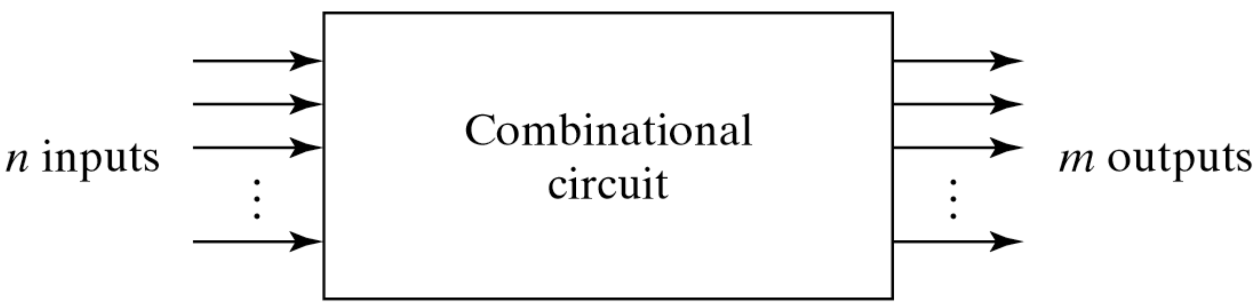


Fig. 4-1 Block Diagram of Combinational Circuit

Combinational Logic

- Combinational circuits:
  - Consist of logic gates only
  - Outputs are determined from the present values of inputs
- Sequential circuits:
  - Consist of logic gates and storage elements
  - Outputs are a function of the inputs and the state of the storage elements
    - Depend not only on present inputs, but also on past values

Combinational Logic

- A combinational circuit consists of:
  - Input variables
  - Logic gates
  - Output variables
- Transform binary information from the given input data to a required output data.

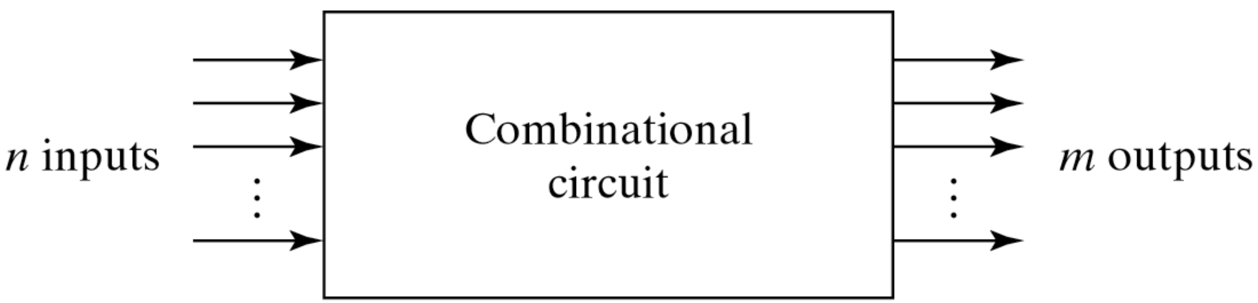


Fig. 4-1 Block Diagram of Combinational Circuit

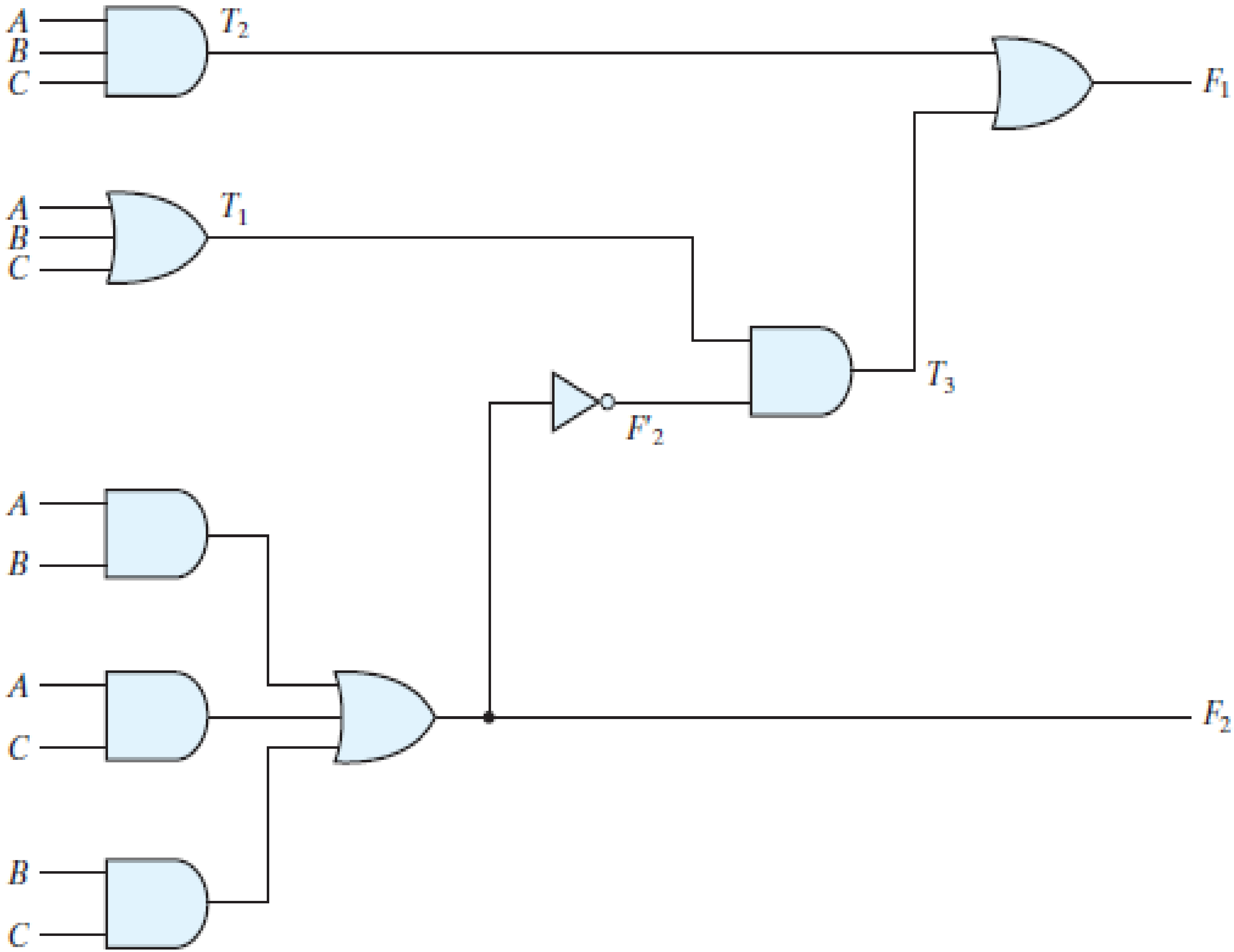


FIGURE 4.2
Logic diagram for analysis example

Combinational Logic

- There are 2^n possible binary input combinations for n input variable
- Only one possible output value for each possible input combination
  - Can be specified with a truth table, ***m*** Boolean functions, one for each output variable , Each output function is expressed in terms of n input variables

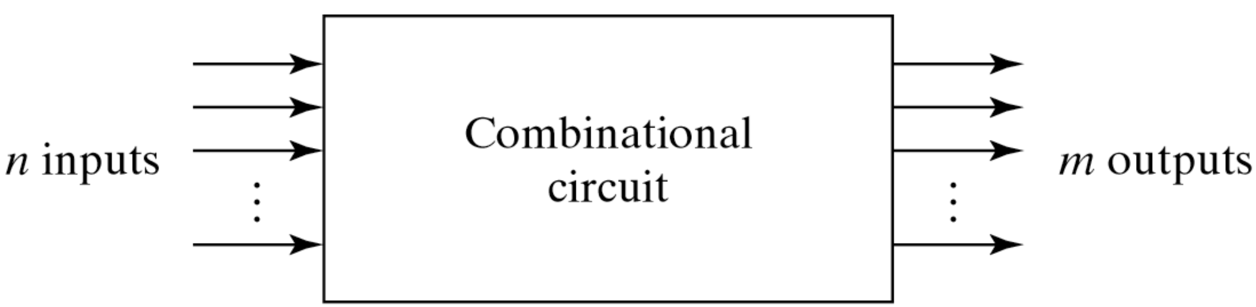


Fig. 4-1 Block Diagram of Combinational Circuit

Analysis Procedure

- **The "analysis" is the reverse of "design".**
- Analysis: determine the function that the circuit implements
  - Often start with a given logic diagram
- **First step**: make sure that circuit is combinational and not sequential.
  - Without feedback paths or memory elements
- **Second step**: obtain the output Boolean functions or the truth table

Analysis Procedure

- To obtain the output Boolean functions from a logic diagram, proceed as follows: (**do it backward**)

1. **Label all gate outputs that are a function of input variables with arbitrary symbols**. Determine the Boolean functions for each gate output.
2. **Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols**. Find the Boolean functions for these gates.
3. **Repeat the process outlined in step 2 until the outputs of the circuit are obtained.**
4. **By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.**

Analysis Procedure - Example

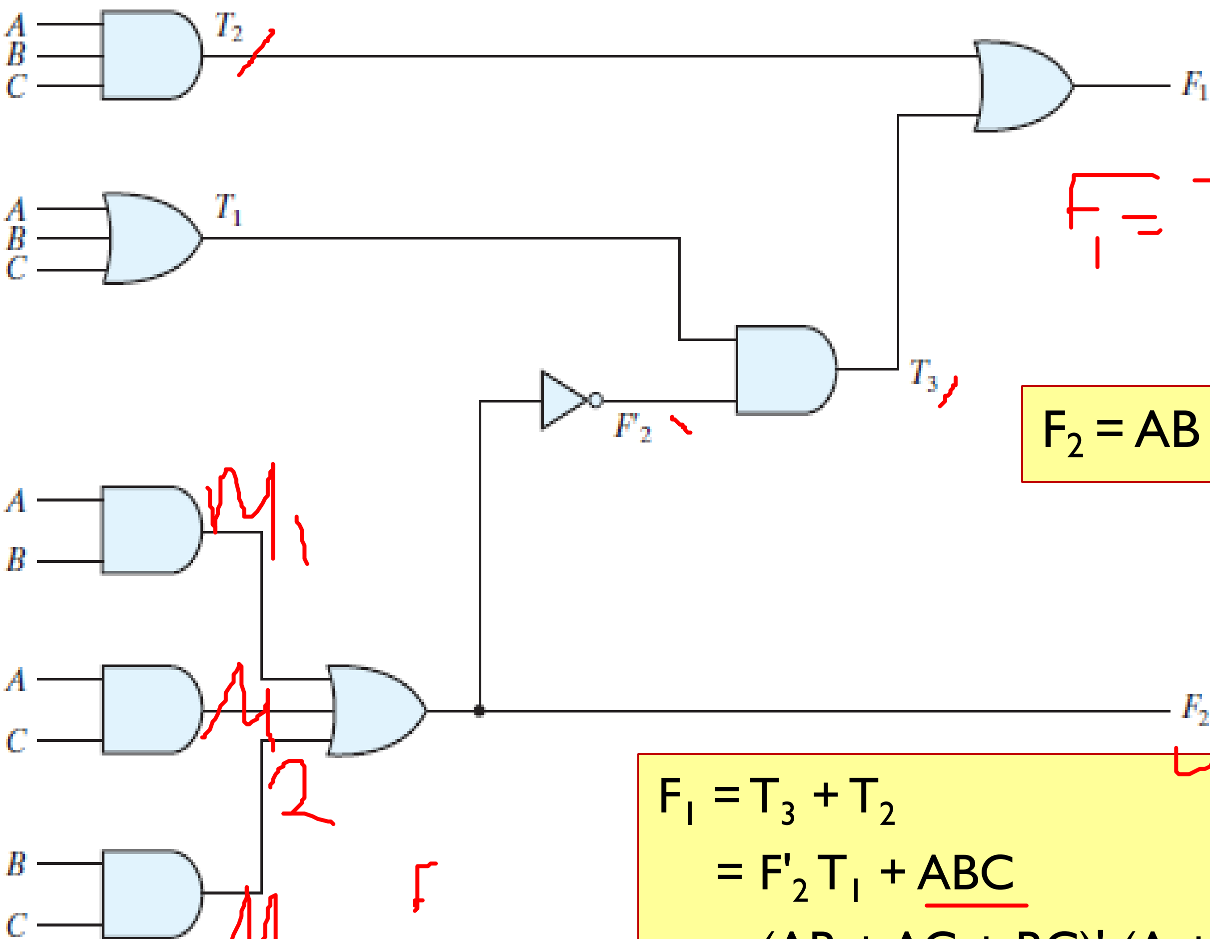


FIGURE 4.2
Logic diagram for analysis example

$F_2 = AB + AC + BC;$

$$
\begin{aligned}
F_1 &= T_3 + T_2 \\
&= F'_2 T_1 + ABC \\
&= (AB + AC + BC)'\,(A + B + C) + ABC \\
&= (A' + B')(A' + C')(B' + C')\,(A + B + C) + ABC \\
&= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\
&= A'BC' + A'B'C + AB'C' + ABC
\end{aligned}
$$

Analysis procedure - Example

- **Truth Table:** We can derive the truth table by using the logic gate diagram

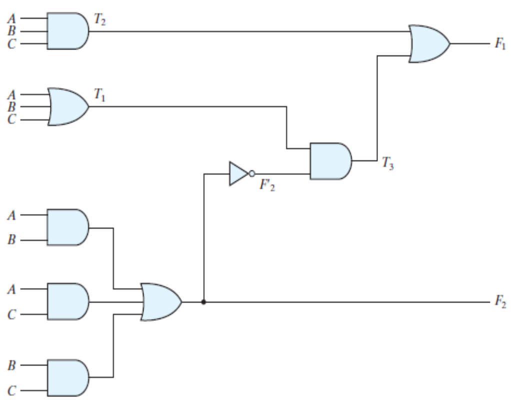


FIGURE 4.2
Logic diagram for analysis example

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Analysis procedure - Example

- **Truth Table:** We can derive the truth table by using the logic gate diagram

- To obtain the truth table from the logic diagram:

1. Determine the number of input variables
   - For n inputs:
     - 2^n possible combinations
     - List the binary numbers from 0 to $2^n - 1$ in a table
2. Label the outputs of selected gates
3. Obtain the truth table for the outputs of those gates that are a function of the input variables only
4. Obtain the truth table for those gates that are a function of previously defined variables at step 3
     - Repeatedly until all outputs are determined

Design Procedure

- Input: the specification of the problem.
- Output: the logic circuit diagram or Boolean functions.

1. determine the required **number of inputs** and **outputs** from the specification
2. **derive the truth table** that defines the required relationship between inputs and outputs
3. obtain the **simplified Boolean function** for each output as a function of the input variables
4. **draw the logic diagram** and verify the correctness of the design.

Code Conversion Design Problems

- It is sometimes necessary to use the output of one system as the input to another.
  - A conversion circuit must be inserted between the two system if each uses different codes for the same information.
    - Thus, a code converter is a circuit that makes the two systems compatible even though each uses a different binary code.
  - To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B.

Code Conversion Example

- **BCD to Excess-3 Code Converter**

2

1

- **Input BCD**
- **4 – Variables Input**

- **Output Excess-3**
- **4 – Variables output**

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Code Conver

- **BCD to Excess-3 Code Converter**

1

- **Input BCD**
- **4 – Variables Input**

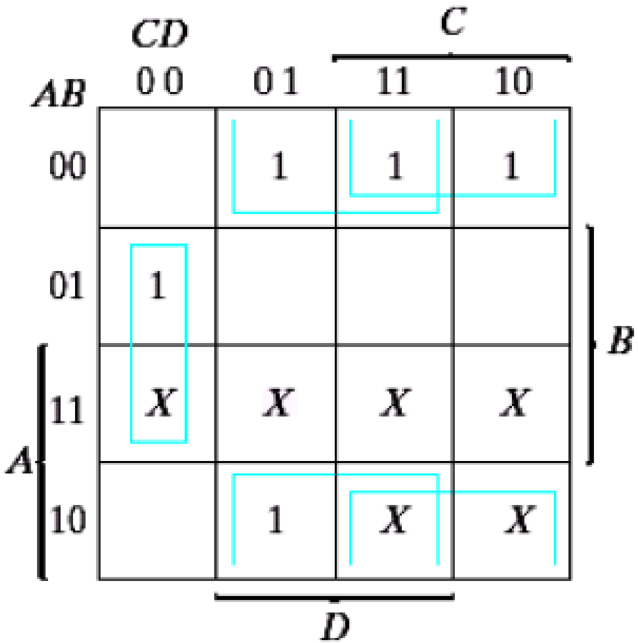
- **Output Excess-3**
- **4 – Variables output**

Input BCD- Code				Output Excess -3 Code			
A	**B**	**C**	**D**	W	X	Y	Z
0	**0**	**0**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**1**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**0**	**1**	**0**	**1**
0	**0**	**1**	**1**	**0**	**1**	**1**	**0**
0	**1**	**0**	**0**	**0**	**1**	**1**	**1**
0	**1**	**0**	**1**	**1**	**0**	**0**	**0**
0	**1**	**1**	**0**	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**1**	**0**	**1**	**0**
1	**0**	**0**	**0**	**1**	**0**	**1**	**1**
1	**0**	**0**	**1**	**1**	**1**	**0**	**0**
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

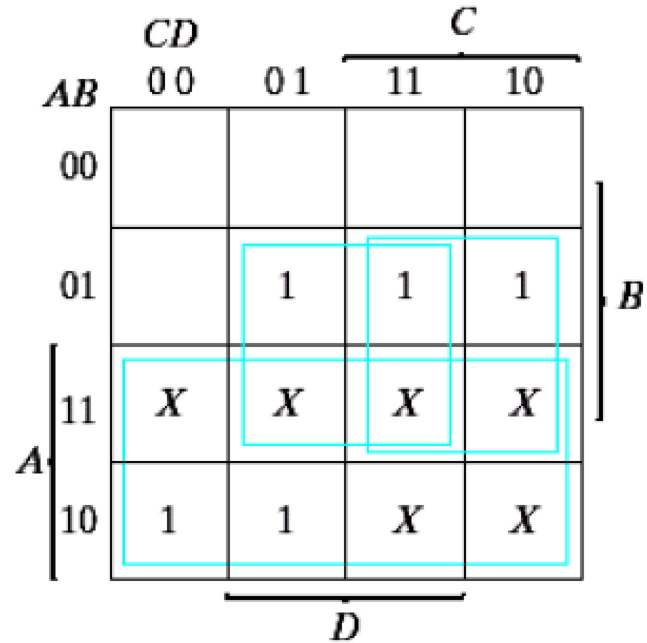
Code Conversion Example

3

- **Boolean Expression :**
- The six don't care minterms (10~15) are marked with X.
- Each of four maps represents one of the four outputs of this circuit as a function of the four input variables.



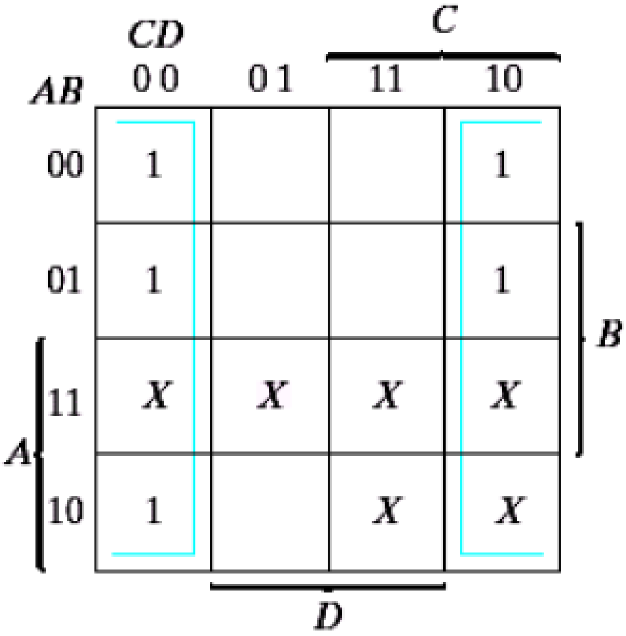
$X = B'C + B'D + BC'D'$



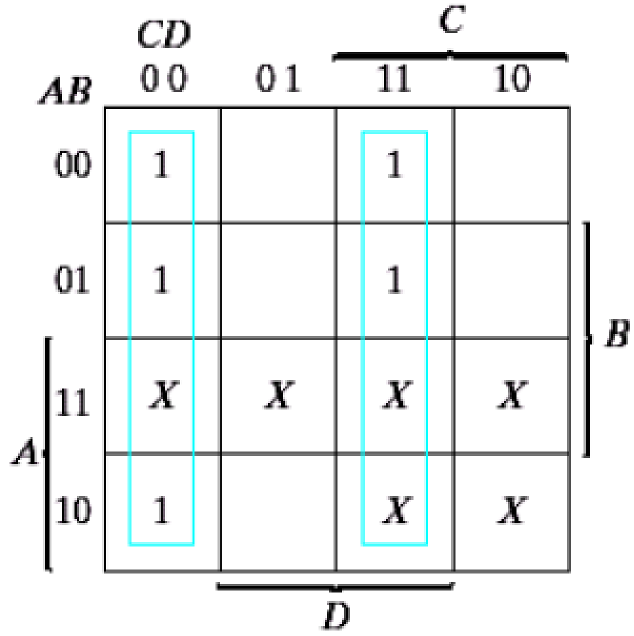
$w = A + BC + BD$

Code Conversion Example

- **Boolean Expression : 3**



$z = D'$



$y = CD + C'D'$

Code Conversion Example

- **Logic Diagram:** Reduce the number of gates used.

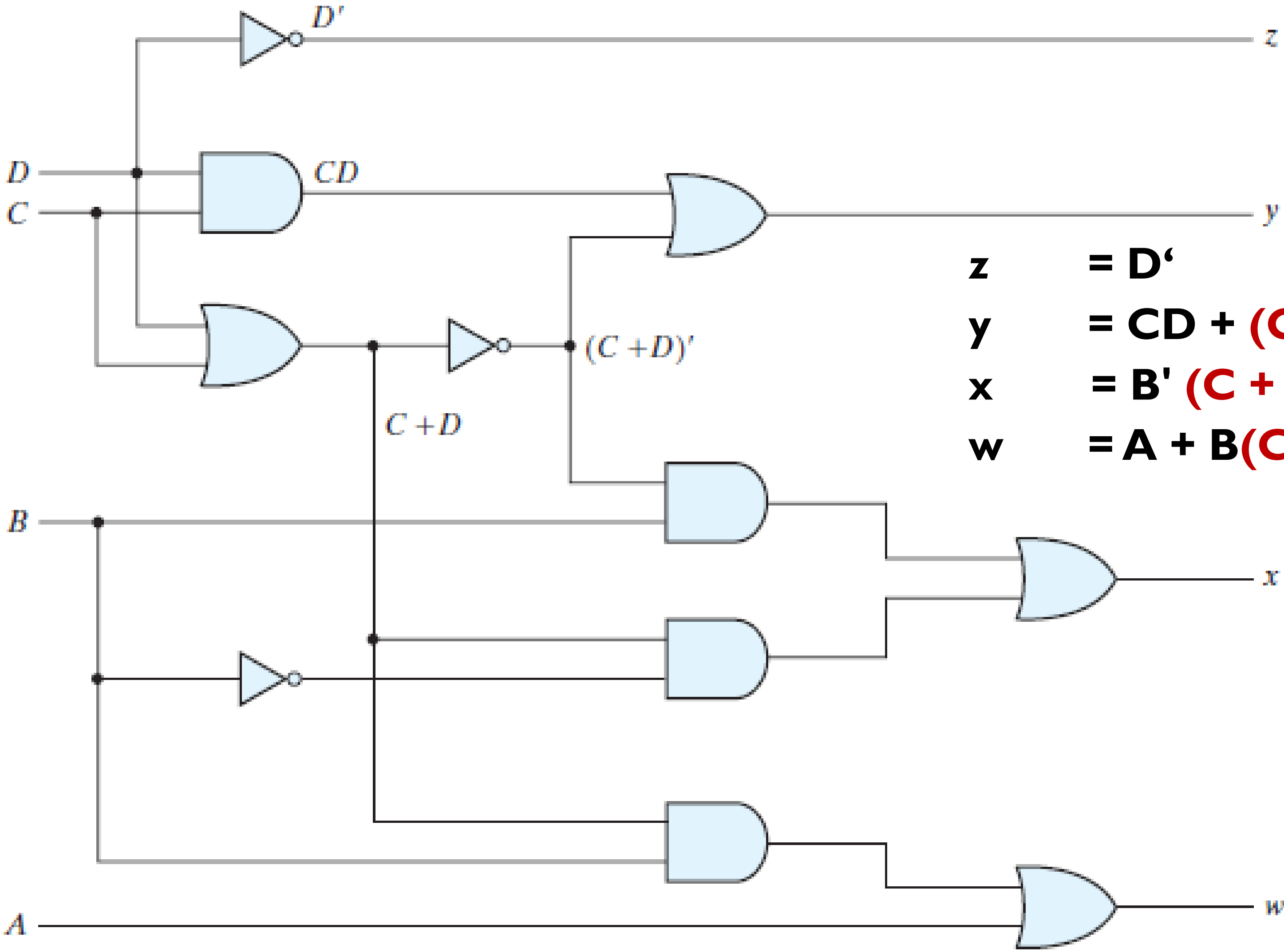
4

z = D‘	x = B'C + B'D + BC' D' = B' (C + D) + BC' D' = B' (C + D) + B(C + D)'
y = CD + C' D‘ = CD + (C + D)'	w = A + BC + BD = A + B(C + D)

- **C + D is used to implement the three outputs.**

Code Conversion Example

4



z = D‘

y = CD + (C + D)‘

x = B' (C + D) + B(C + D)'

w = A + B(C + D)


Thank You!