Chapter 3 Describing Logic Circuits

- *Selected areas covered in this chapter:*
  - Operation of truth tables for **AND**, **NAND**, **OR**, **NOR** gates, and the **NOT** (INVERTER) circuit.
  - Boolean expression for logic gates.
  - DeMorgan's theorems to simplify logic expressions.
  - Universal gates (**NAND** or **NOR**) to implement a circuit represented by a Boolean expression.
  - Concepts of active-LOW & active-HIGH logic signals.
  - Describing and measuring propagation delay time.

Chapter 3-1 Boolean Constants and Variables

- Boolean algebra allows only two values—0 and 1.
  - **Logic 0** can be: *false, off, low, no, open switch.*
  - **Logic 1** can be: *true, on, high, yes, closed switch.*

Logic 0	Logic 1
False	True
Off	On
LOW	HIGH
No	Yes
Open switch	Closed switch

Boolean variables are often used to represent the voltage level present on a wire or at the input/output terminals of a circuit.

Chapter 3-1 Boolean Constants and Variables

- Boolean 0 and 1 do not represent actual numbers but instead represent the state of a voltage variable, or what is called its **logic level.**
- A voltage in a digital circuit is said to be at the logic 0 level or the logic 1 level, depending on its actual numerical value.
- Boolean algebra is a means for expressing the relationship between a logic circuit's inputs and outputs.
- The inputs are considered logic variables whose logic levels at any time determine the output levels.
- There are no fractions, decimals, negative numbers, square roots, cube roots, logarithms, imaginary numbers, and so on.
  - The three basic logic operations:
    – OR, AND, and NOT.

Chapter 3-2 Truth Table

- A truth table describes the relationship between the input and output of a logic circuit.
- The number of entries corresponds to the number of inputs.
  - A 2-input table would have $2^2 = 4$ entries.
  - A 3-input table would have $2^3 = 8$ entries.
  - A n-input table would have 2^n entries.

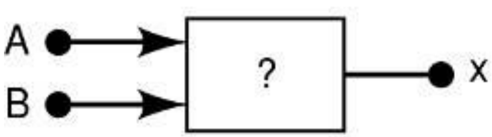
Chapter 3-2 Truth Table

Examples of truth tables with 2, 3, and 4 inputs.

Output

Inputs

A	B	x
0	0	1
0	1	0
1	0	1
1	1	0



(a)

A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b)

A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(c)

Chapter 3-3 OR Operation With OR Gates

- The Boolean expression for the **OR** operation is:

$$X = A + B$$ — Read as "*X* equals *A* OR *B*"

The + sign does *not* stand for ordinary addition—it stands for the OR operation

- The OR operation is similar to addition, but when A = 1 and B = 1, the OR operation produces:

$$1 + 1 = 1 \textit{ not } 1 + 1 = 2$$

In the Boolean expression x = 1 + 1 + 1 = 1…

x is true (1) when A is true (1) OR B is true (1) OR C is true (1)

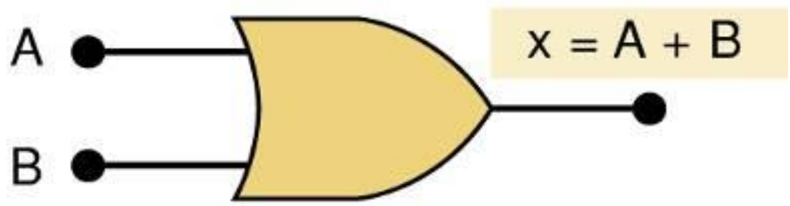
Chapter 3-3 OR Operation With OR Gates

- An OR gate is a circuit with two or more inputs, whose output is equal to the OR combination
of the inputs.
- The OR gate output will be HIGH whenever any input is HIGH

Truth table/circuit symbol for a two input OR gate.

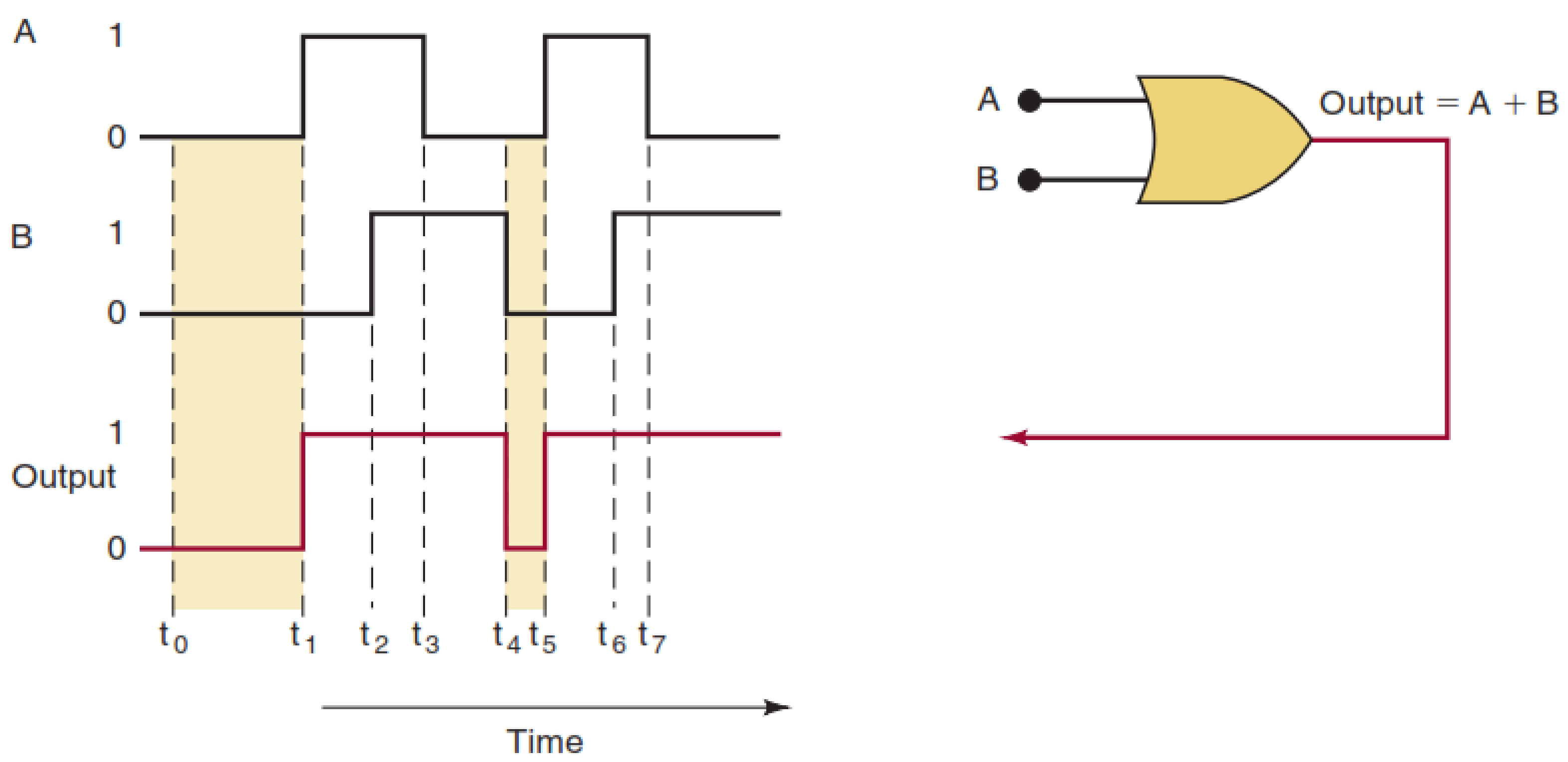
OR

A	B	$x = A + B$
0	0	0
0	1	1
1	0	1
1	1	1



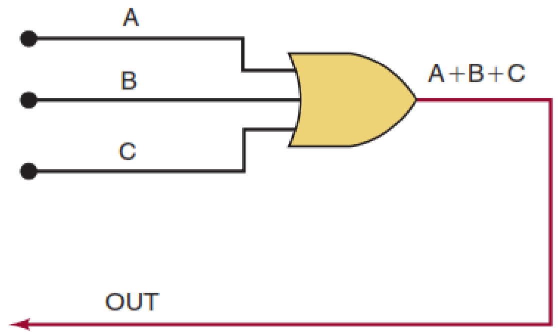
OR Gate

Chapter 3-3 OR Operation With OR Gates

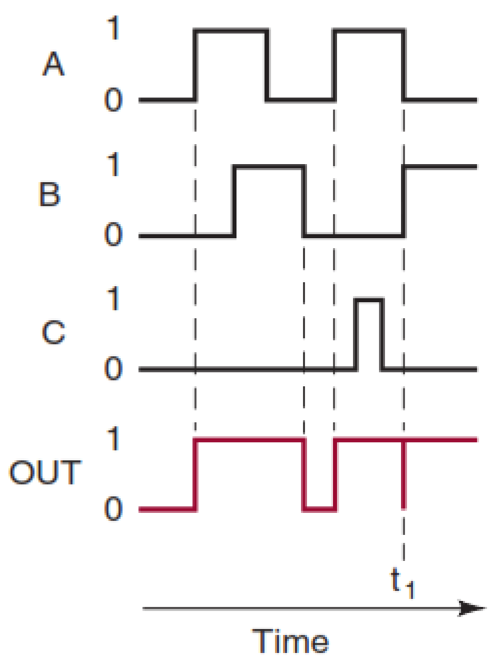


Chapter 3-3 OR Operation With OR Gates

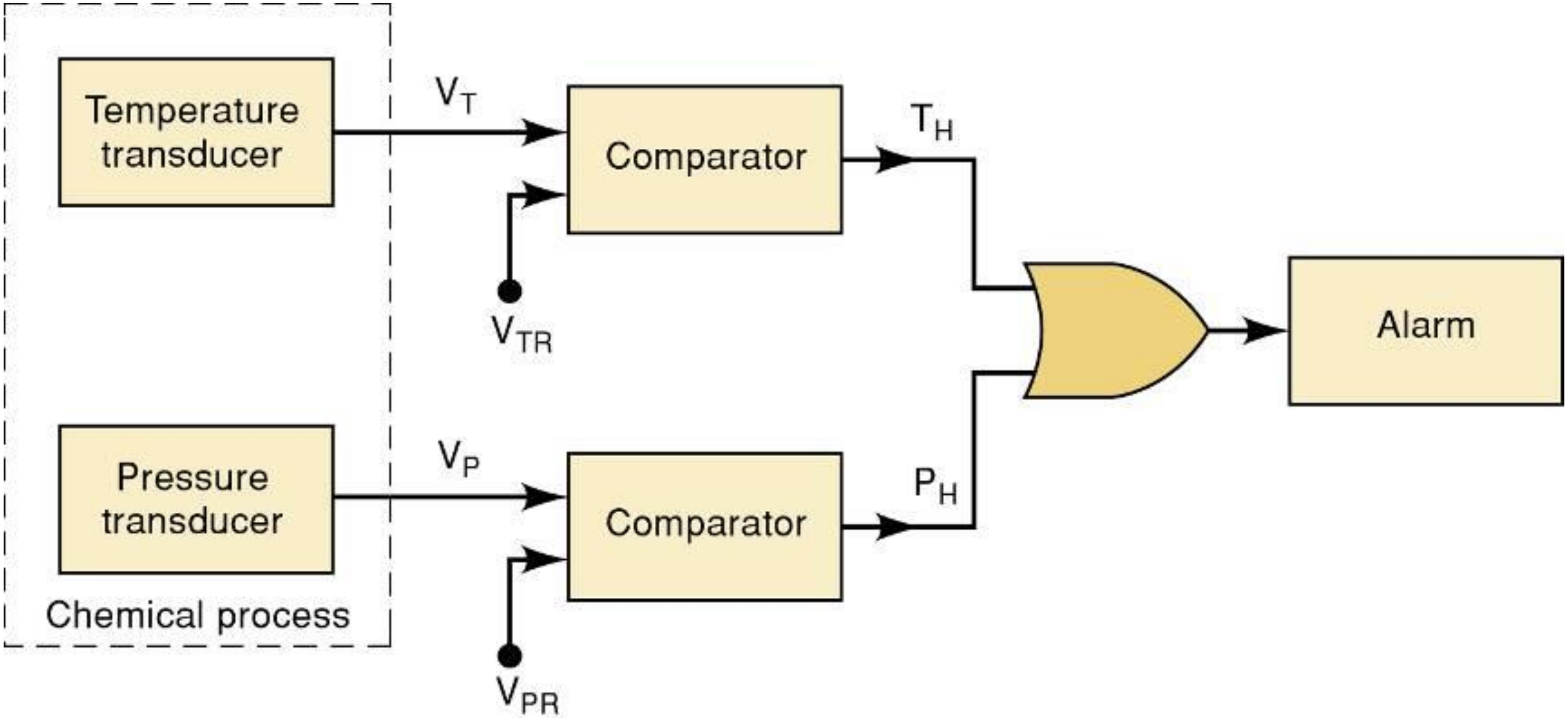
Truth table/circuit symbol for a three input OR gate.



A	B	C	x = A + B + C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Example of the use of an OR gate in an alarm system.



To activate an output function whenever any one of several inputs is activated.

Chapter 3-4 AND Operations with AND gates

- The **AND** operation is similar to multiplication:

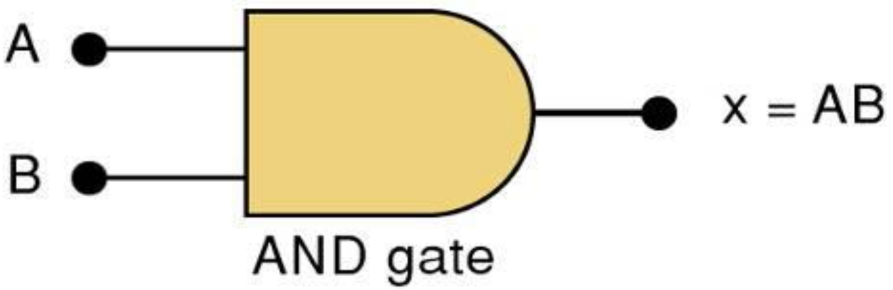
$X = A \bullet B \bullet C$ — Read as "*X* equals *A* AND *B* AND *C*"

The • sign does *not* stand for ordinary multiplication—it stands for the AND operation.

x is true (1) when A AND B AND C are true (1)

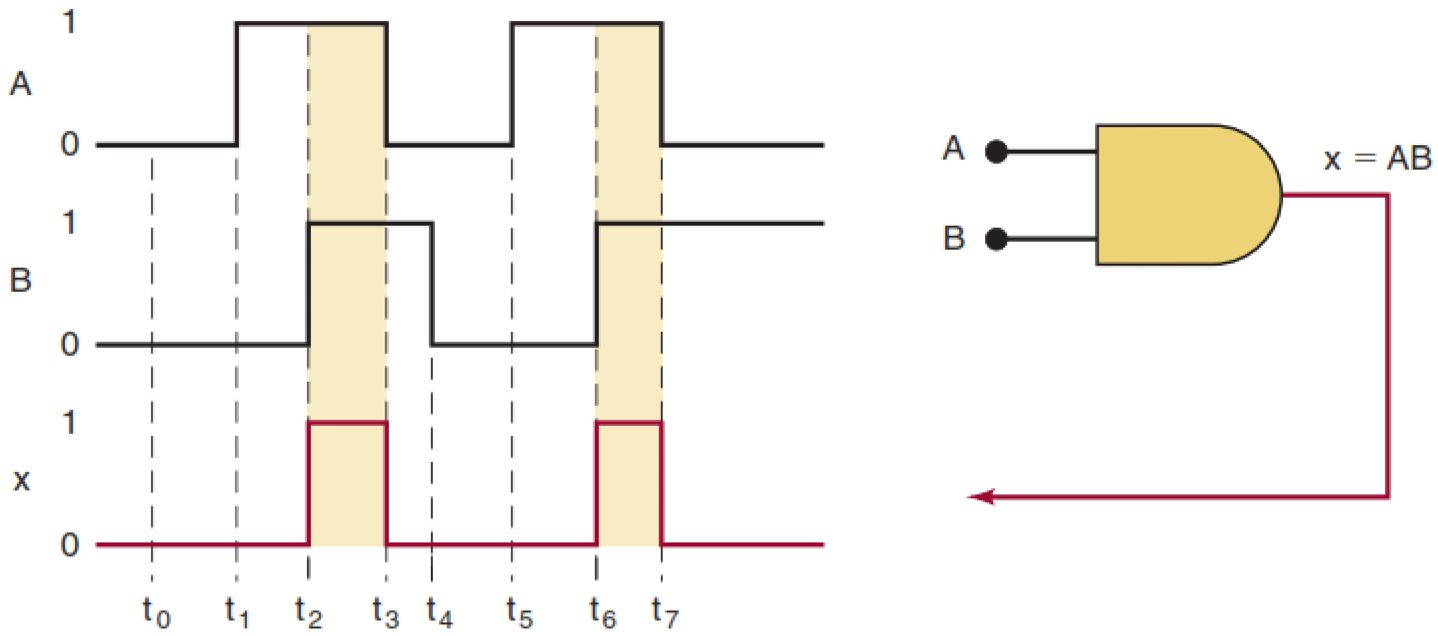
AND

A	B	$x = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1



Truth table — Gate symbol.

Chapter 3-4 AND Operations with AND gates

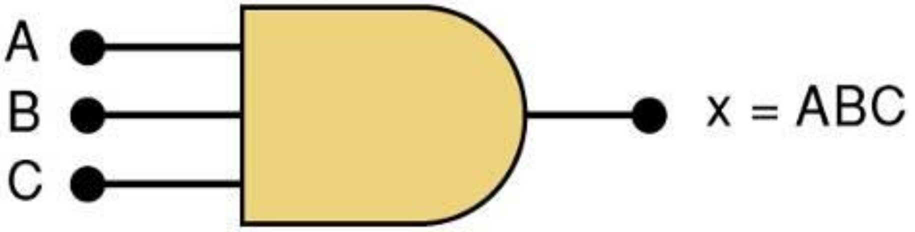


The output of an AND gate is determined by realizing that it will be HIGH only when all inputs are HIGH at the same time.

Chapter 3-4 AND Operations with AND gates

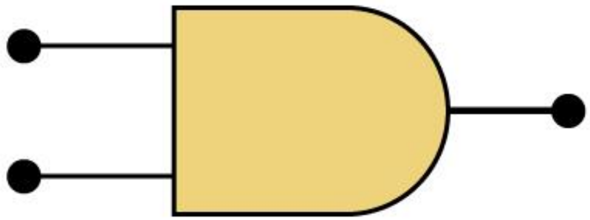
Truth table/circuit symbol for a three input AND gate.

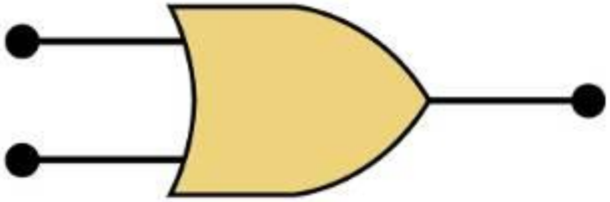
A	B	C	x = ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Chapter 3-4 AND Operations with AND gates

The AND symbol on a logic-circuit diagram tells you output will go HIGH *only* when *all* inputs are HIGH.





The OR symbol means the output will go HIGH when *any* input is HIGH.

Chapter 3-5 NOT Operation

- The Boolean expression for the **NOT** operation:

$$X = \overline{A}$$ — Read as: "*X* equals NOT *A*"

"*X* equals the *inverse* of *A*"

"*X* equals the *complement* of *A*"

The overbar represents the NOT operation.

$$A' = \overline{A}$$

Another indicator for inversion is the prime symbol (').

NOT

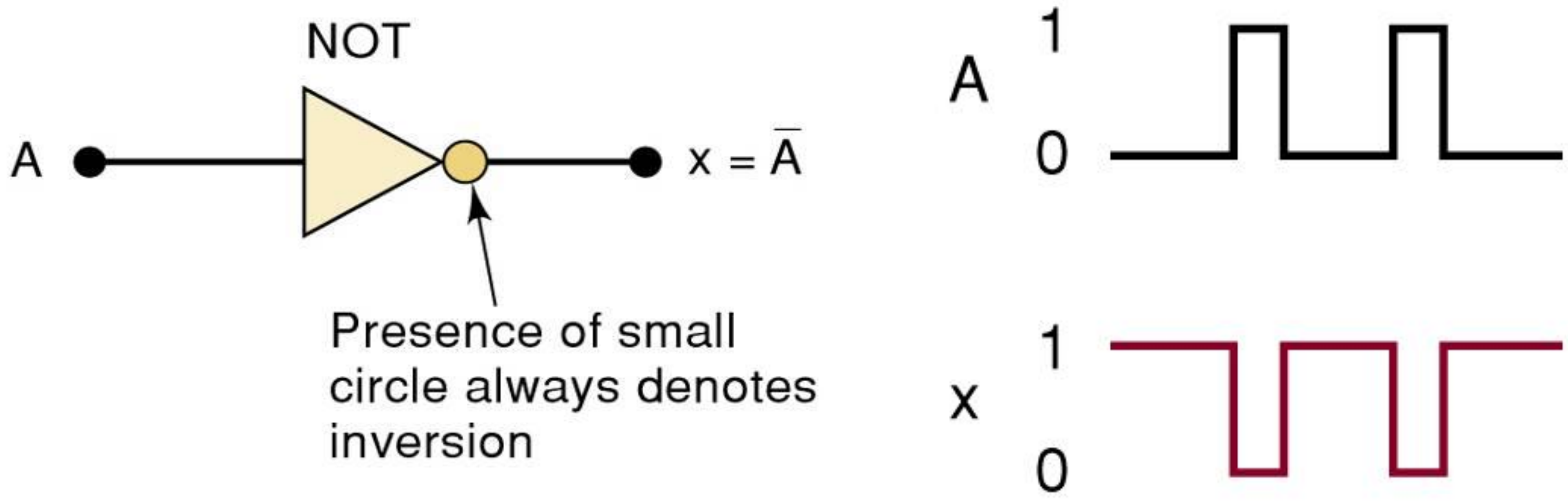
A	$x = \overline{A}$
0	1
1	0

NOT Truth Table

The NOT operation is also referred to as **inversion or complementation,** and these terms will be used interchangeably

Chapter 3-5 NOT Operation

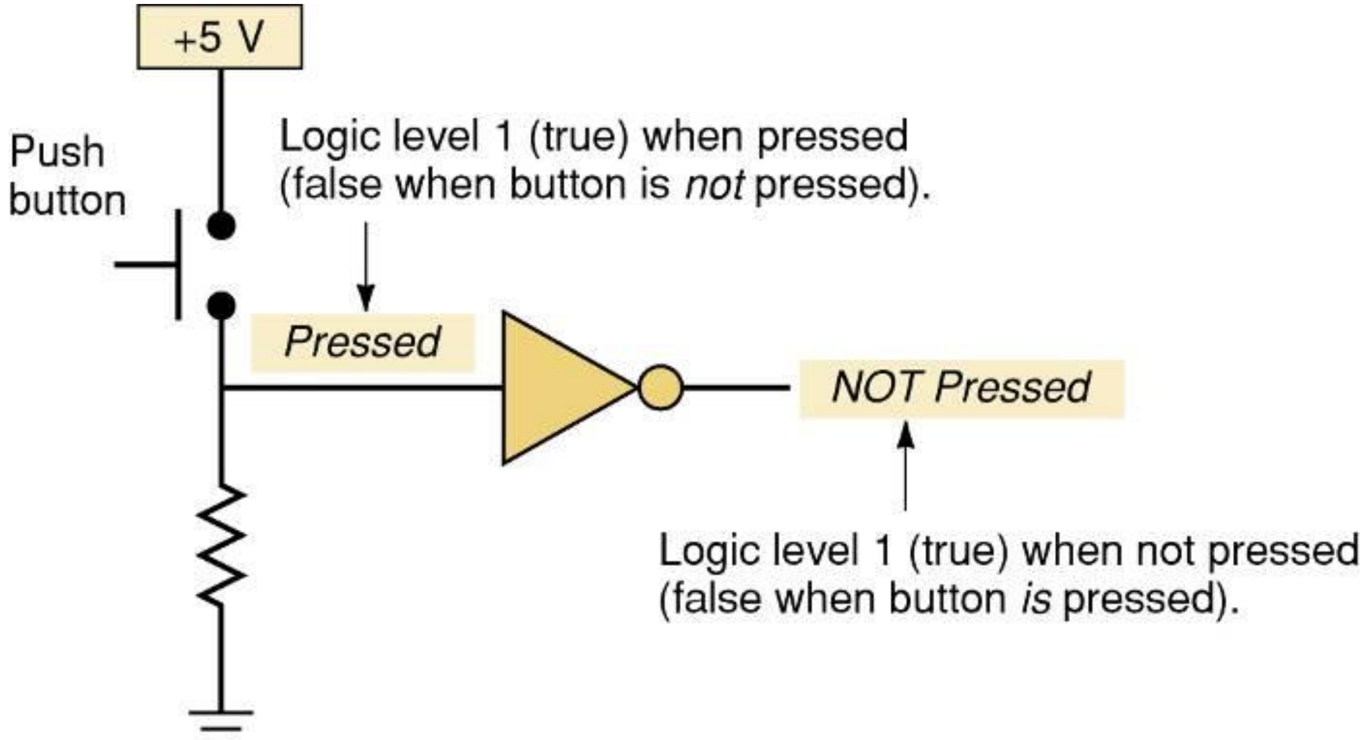
A NOT circuit—commonly called an INVERTER.



- This circuit *always* has only a single input, and the out-put logic level is always *opposite* to the logic level of this input.
- The INVERTER inverts (*complements*) the input signal at all points on the waveform.
- Whenever the input = 0, output = 1, and vice versa

Chapter 3-5 NOT Operation

Typical application of the NOT gate.



This circuit provides an expression that is true when the button is not pressed.

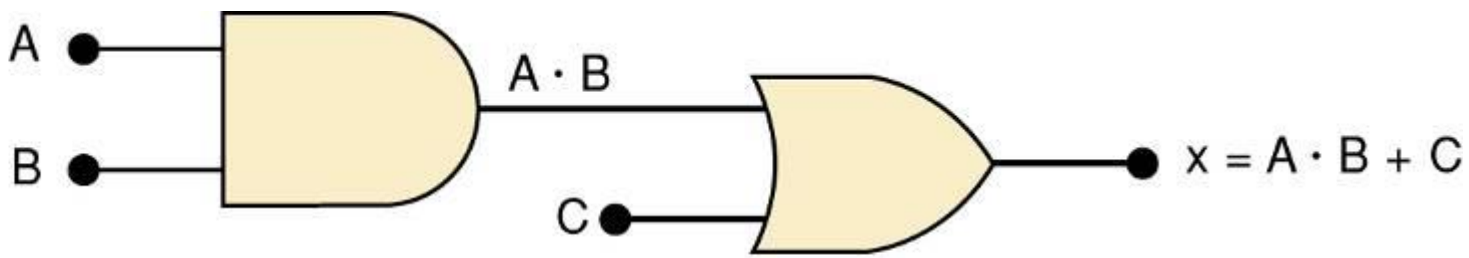
Summarized rules for **OR**, **AND** and **NOT**

OR	***AND***	***NOT***
$0 + 0 = 0$	$0 \cdot 0 = 0$	$\overline{0} = 1$
$0 + 1 = 1$	$0 \cdot 1 = 0$	$\overline{1} = 0$
$1 + 0 = 1$	$1 \cdot 0 = 0$	
$1 + 1 = 1$	$1 \cdot 1 = 1$	

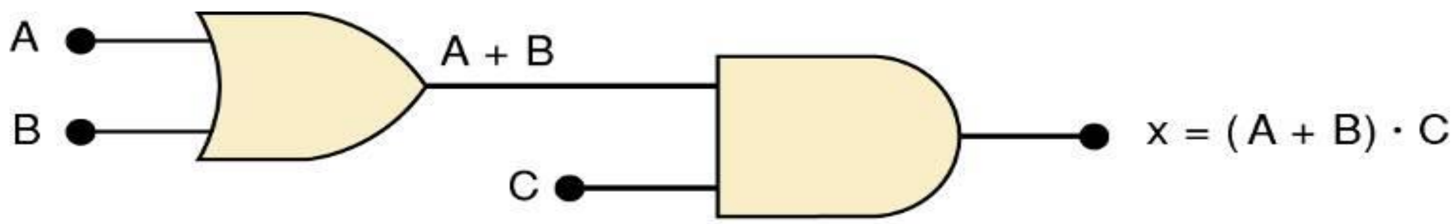
These three basic Boolean operations can describe any logic circuit.

Chapter 3-6 Describing Logic Circuits Algebraically

- If an expression contains both **AND** and **OR** gates, the **AND** operation will be performed first.

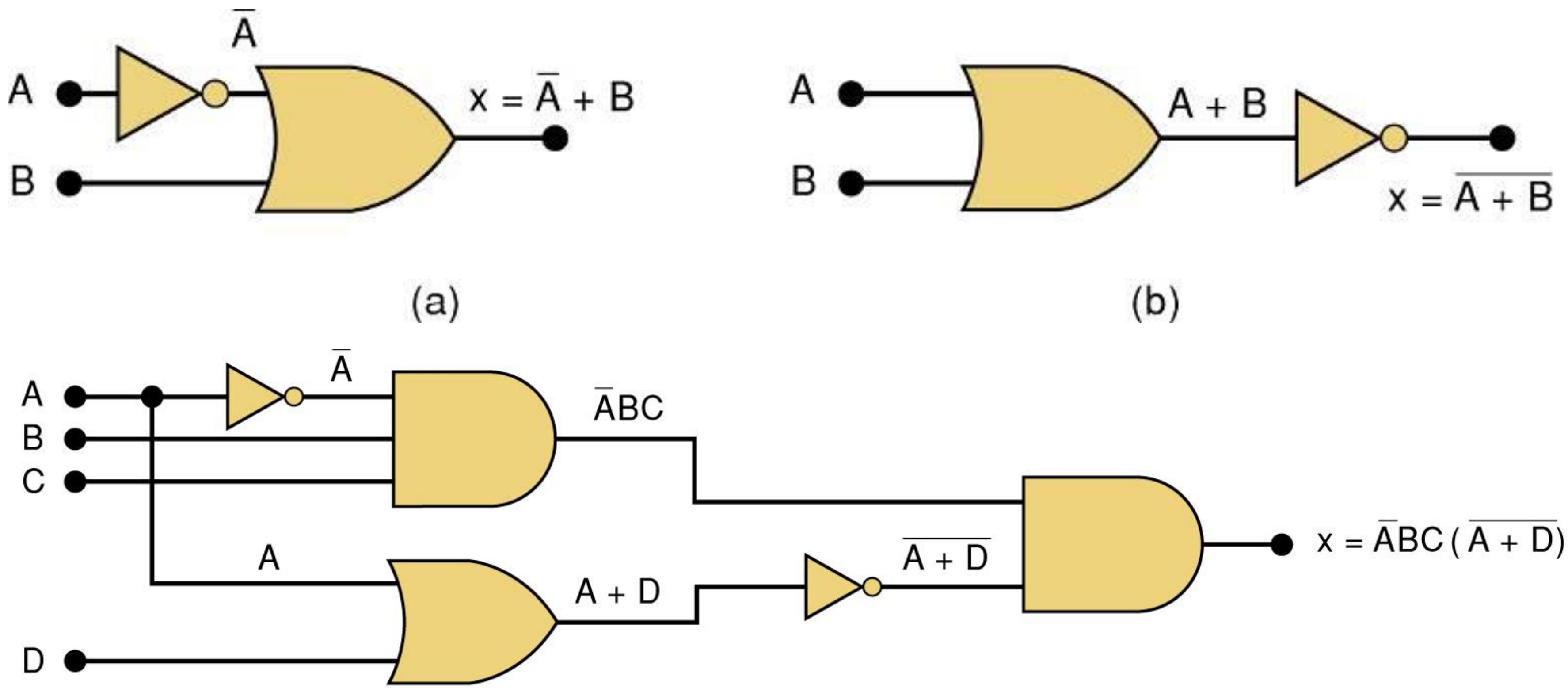


- Unless there is a parenthesis in the expression.

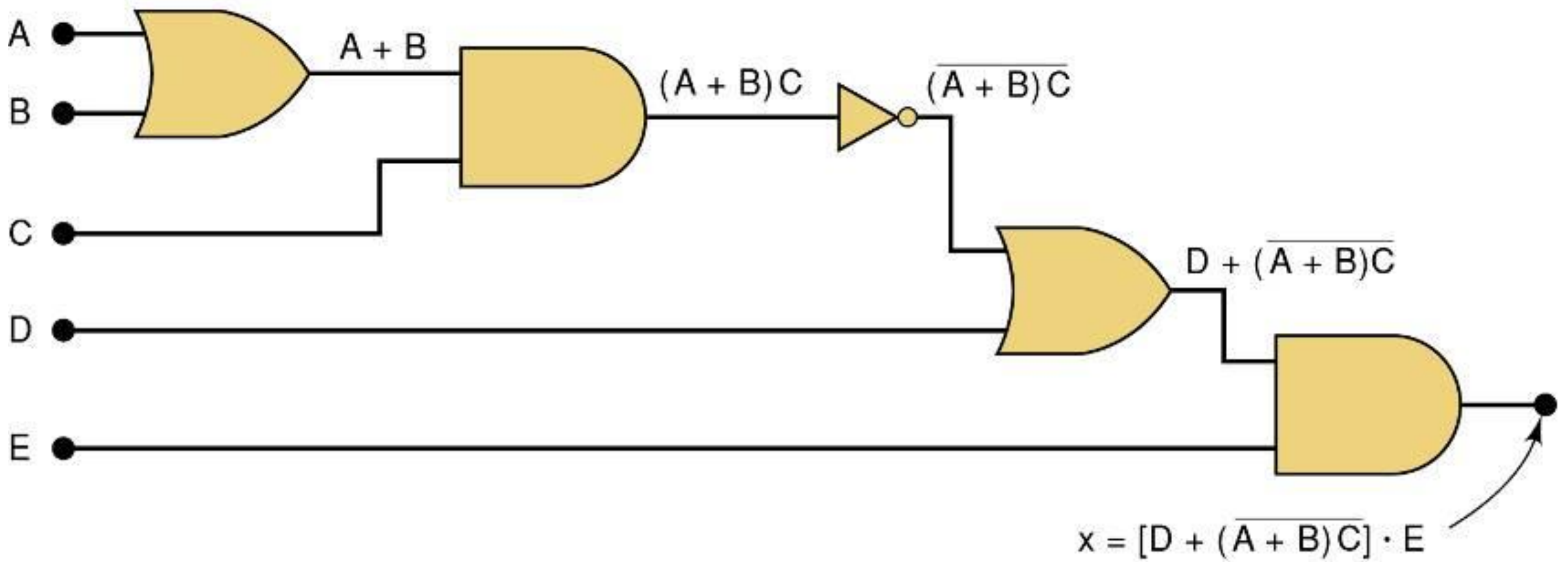


Chapter 3-6 Describing Logic Circuits Algebraically

- Whenever an INVERTER is present, output is equivalent to input, with a bar over it.
  - Input A through an inverter equals $\overline{A}$.



Chapter 3-6 Describing Logic Circuits Algebraically



Chapter 3-7 Evaluating Logic Circuit Outputs

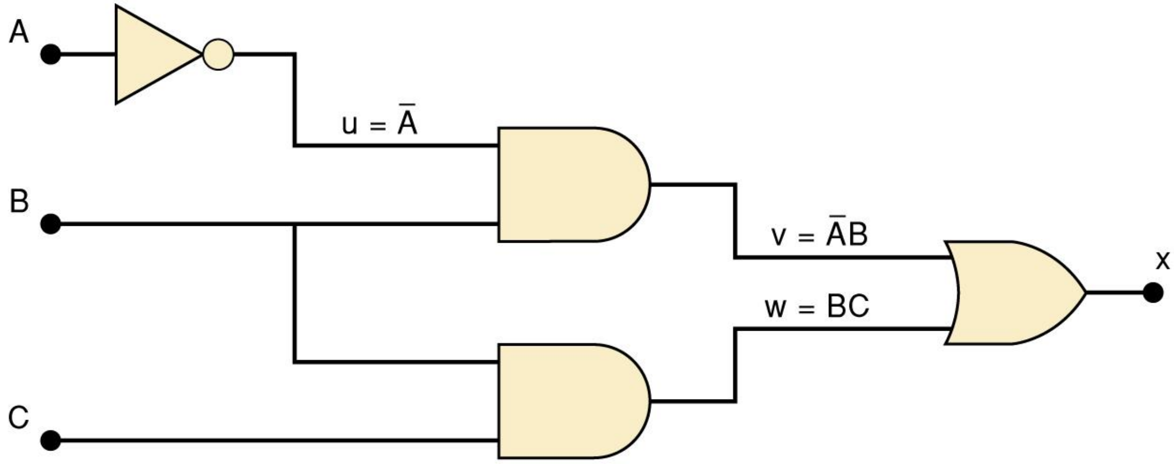
- Rules for evaluating a Boolean expression:
  - Perform all inversions of single terms.
  - Perform all operations within parenthesis.
  - Perform **AND** operation before an **OR** operation unless parenthesis indicate otherwise.
  - If an expression has a bar over it, perform operations inside the expression, and then invert the result.

A = 0, B = 1, C = 1, and D = 1.

$$
\begin{aligned}
x &= \overline{A}BC(\overline{A + D}) \\
&= \overline{0}\cdot 1\cdot 1\cdot(\overline{0 + 1}) \\
&= 1\cdot 1\cdot 1\cdot(\overline{0 + 1}) \\
&= 1\cdot 1\cdot 1\cdot(\overline{1}) \\
&= 1\cdot 1\cdot 1\cdot 0 \\
&= 0
\end{aligned}
$$

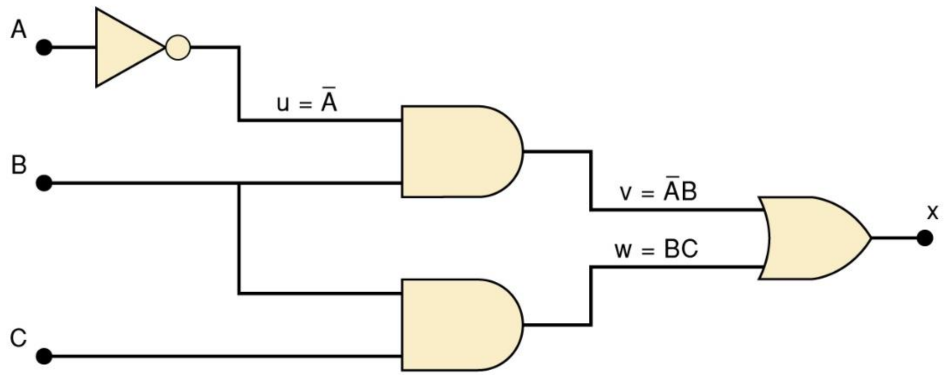
Chapter 3-7 Evaluating Logic Circuit Outputs

- The best way to analyze a circuit made up of multiple logic gates is to use a truth table.
  - It allows you to analyze one gate or logic combination at a time.
  - It allows you to easily double-check your work.
  - When you are done, you have a table of tremendous benefit in troubleshooting the logic circuit.



Chapter 3-7 Evaluating Logic Circuit Outputs

- The first step after listing all input combinations is to create a column in the truth table for each intermediate signal (node).

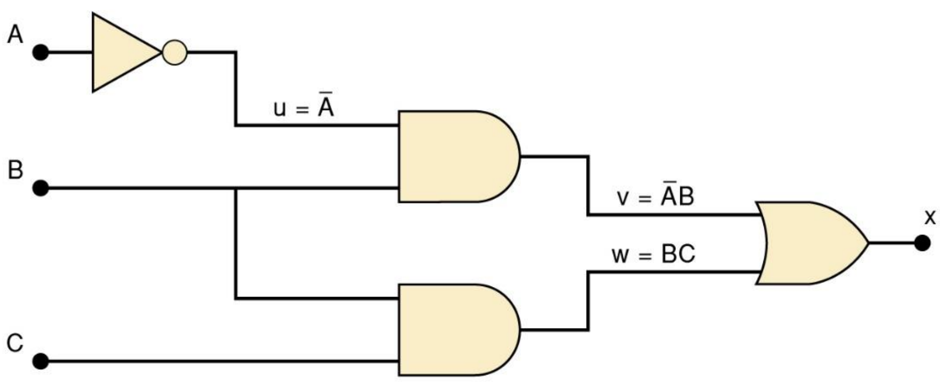


A	B	C	u= $\overline{A}$	v= $\overline{A}B$	w= BC	x= v+w
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	0			
1	0	1	0			
1	1	0	0			
1	1	1	0			

Node *u* has been filled as the complement of *A*

Chapter 3-7 Evaluating Logic Circuit Outputs

- The next step is to fill in the values for column *v*.

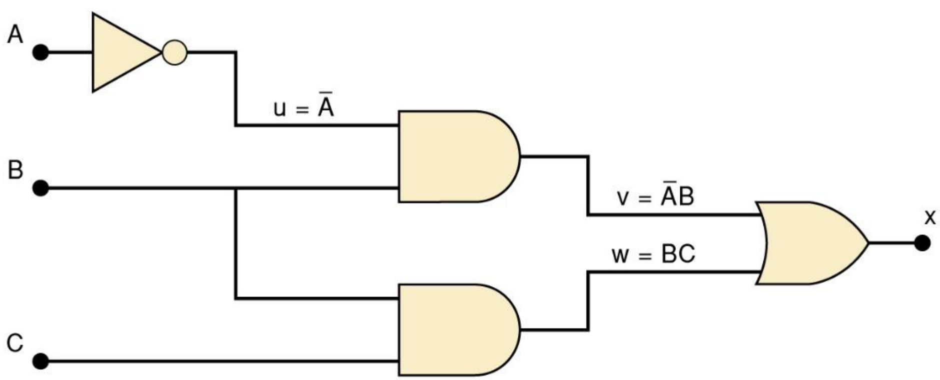


A	B	C	u = $\bar{A}$	v = $\bar{A}B$	w = BC	x = v+w
0	0	0	1	0		
0	0	1	1	0		
0	1	0	1	1		
0	1	1	1	1		
1	0	0	0	0		
1	0	1	0	0		
1	1	0	0	0		
1	1	1	0	0		

v =*AB* — Node *v* should be HIGH
when *A* (node *u*) is HIGH AND *B* is HIGH

Chapter 3-7 Evaluating Logic Circuit Outputs

- The third step is to predict the values at node *w* which is the logical product of *BC*.

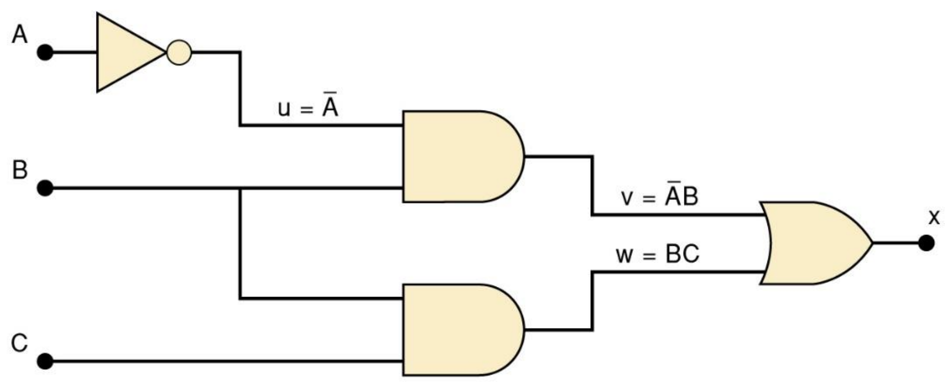


A	B	C	u= $\bar{A}$	v= $\bar{A}B$	w= BC	x= v+w
0	0	0	1	0	0	
0	0	1	1	0	0	
0	1	0	1	1	0	
0	1	1	1	1	1	
1	0	0	0	0	0	
1	0	1	0	0	0	
1	1	0	0	0	0	
1	1	1	0	0	1	

This column is HIGH whenever *B* is HIGH AND *C* is HIGH

Chapter 3-7 Evaluating Logic Circuit Outputs

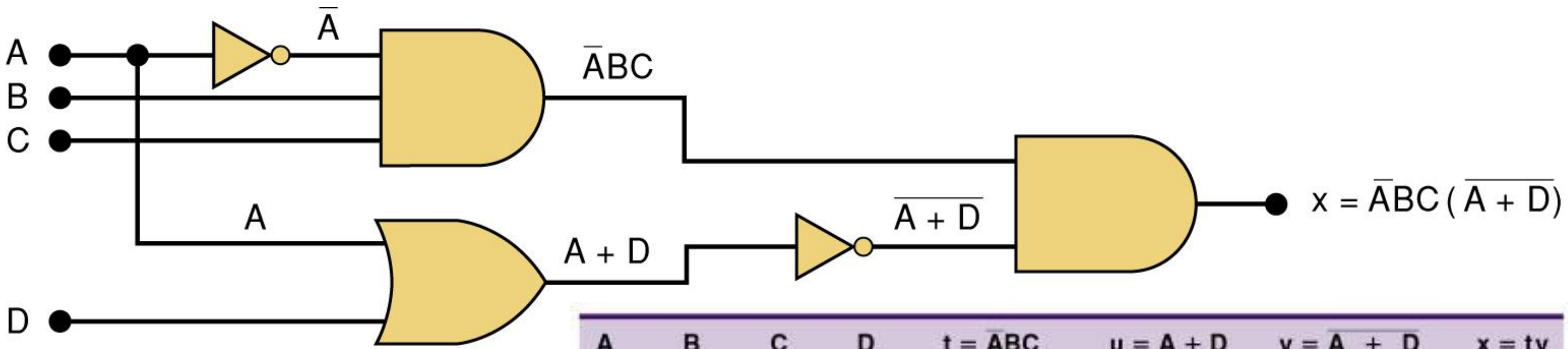
- The final step is to logically combine columns *v* and *w* to predict the output *x*.



A	B	C	u= $\overline{A}$	v= $\overline{A}$B	w= BC	x= v+w
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

Since *x* = *v* + w, the *x* output will be HIGH when *v* OR *w* is HIGH

Chapter 3-7 Evaluating Logic Circuit Outputs



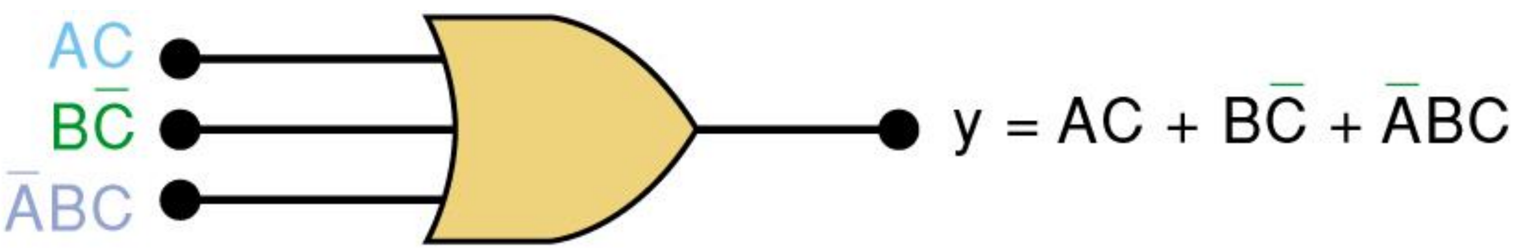
A	B	C	D	$t = \overline{A}BC$	$u = A + D$	$v = \overline{A + D}$	$x = tv$
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	1
0	1	1	1	1	1	0	0
1	0	0	0	0	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	1	0	0
1	1	0	1	0	1	0	0
1	1	1	0	0	1	0	0
1	1	1	1	0	1	0	0

Table of logic state at each node of the circuit shown.

Chapter 3-8 Implementing Circuits From Boolean Expressions

- It is important to be able to draw a logic circuit from a Boolean expression.
  - The expression $X = A \cdot B \cdot C$, could be drawn as a three input **AND** gate.

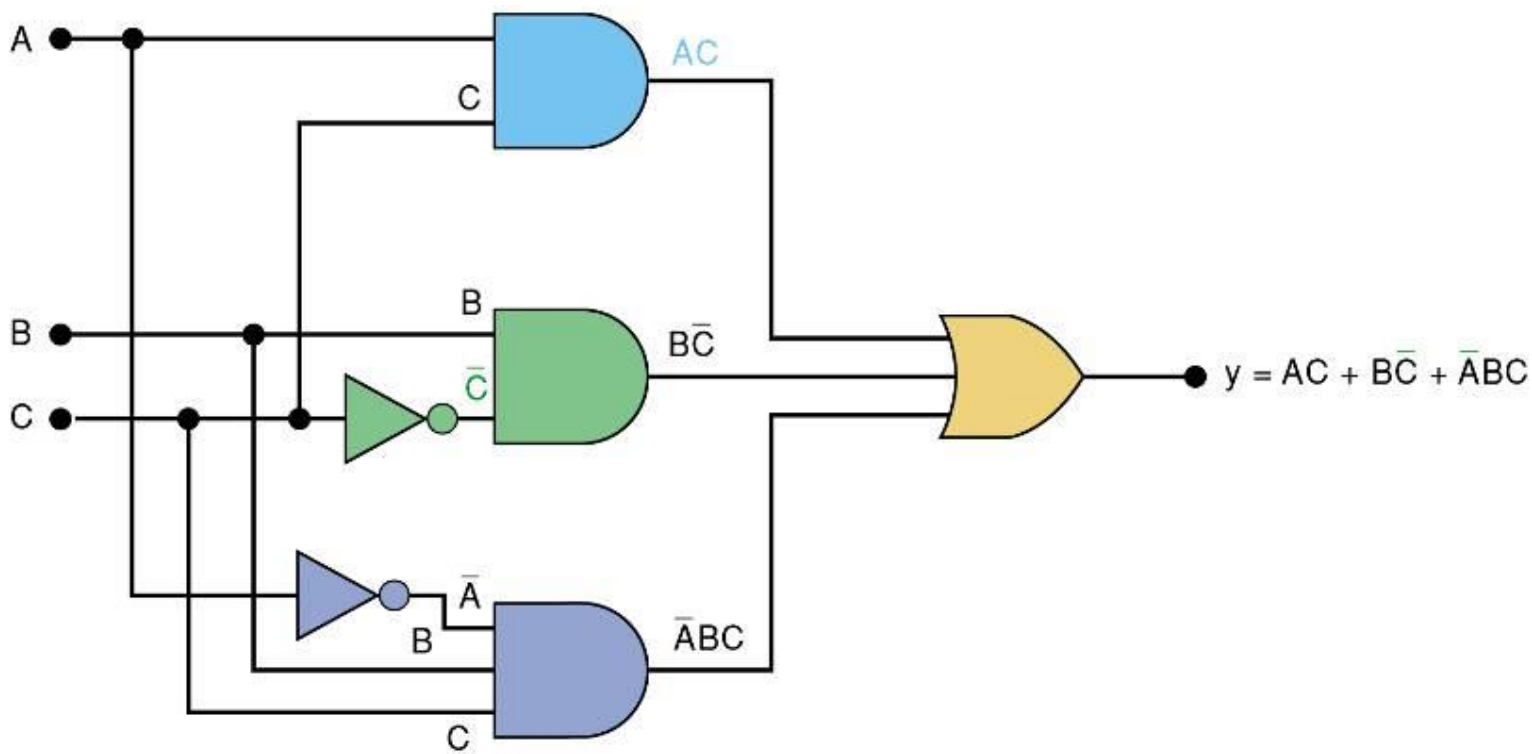
A circuit with output $y = AC + B\overline{C} + \overline{A}BC$ contains three terms which are ORed together.



…and requires a three-input OR gate.

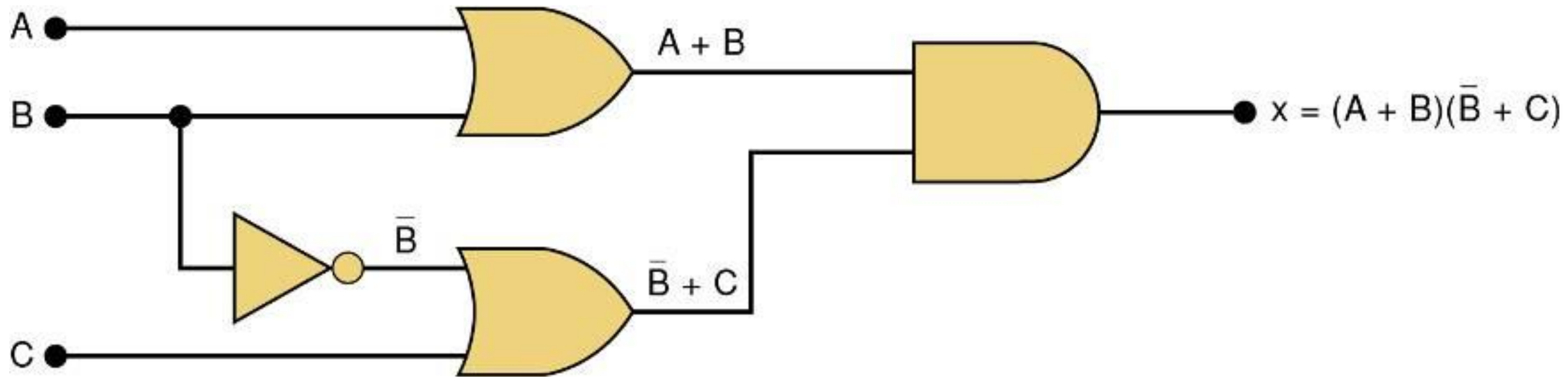
Chapter 3-8 Implementing Circuits From Boolean Expressions

- Each **OR** gate input is an **AND** product term,
  - An **AND** gate with appropriate inputs can be used to generate each of these terms.



Chapter 3-8 Implementing Circuits From Boolean Expressions

Circuit diagram to implement $x = (A + B)\,(\overline{B} + C)$



Review Question

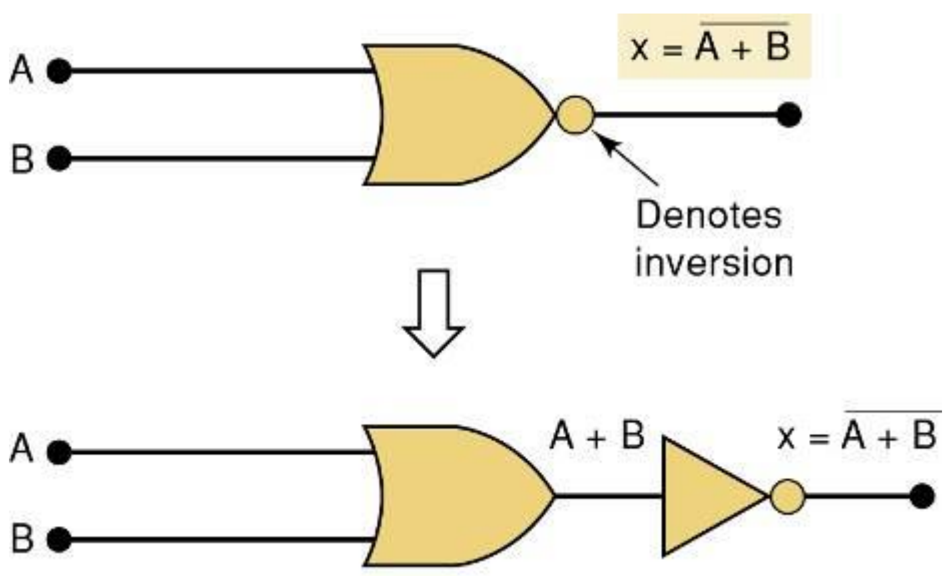
1. Draw the circuit diagram that implements the expression $x = \overline{A}BC(\overline{A + D})$ using gates with no more than three inputs.
2. Draw the circuit diagram for the expression $y = AC + B\overline{C} + \overline{A}BC$.
3. Draw the circuit diagram for $x = [D + (\overline{A + B})\overline{C}] \cdot E$.

Chapter 3-9 NOR Gates and NAND Gates

- Combine basic **AND**, **OR**, and **NOT** operations.
  - Simplifying the writing of Boolean expressions
- Output of **NAND** and **NOR** gates may be found by determining the output of an **AND** or **OR** gate, and inverting it.
  - The truth tables for **NOR** and **NAND** gates show the complement of truth tables for **OR** and **AND** gates.

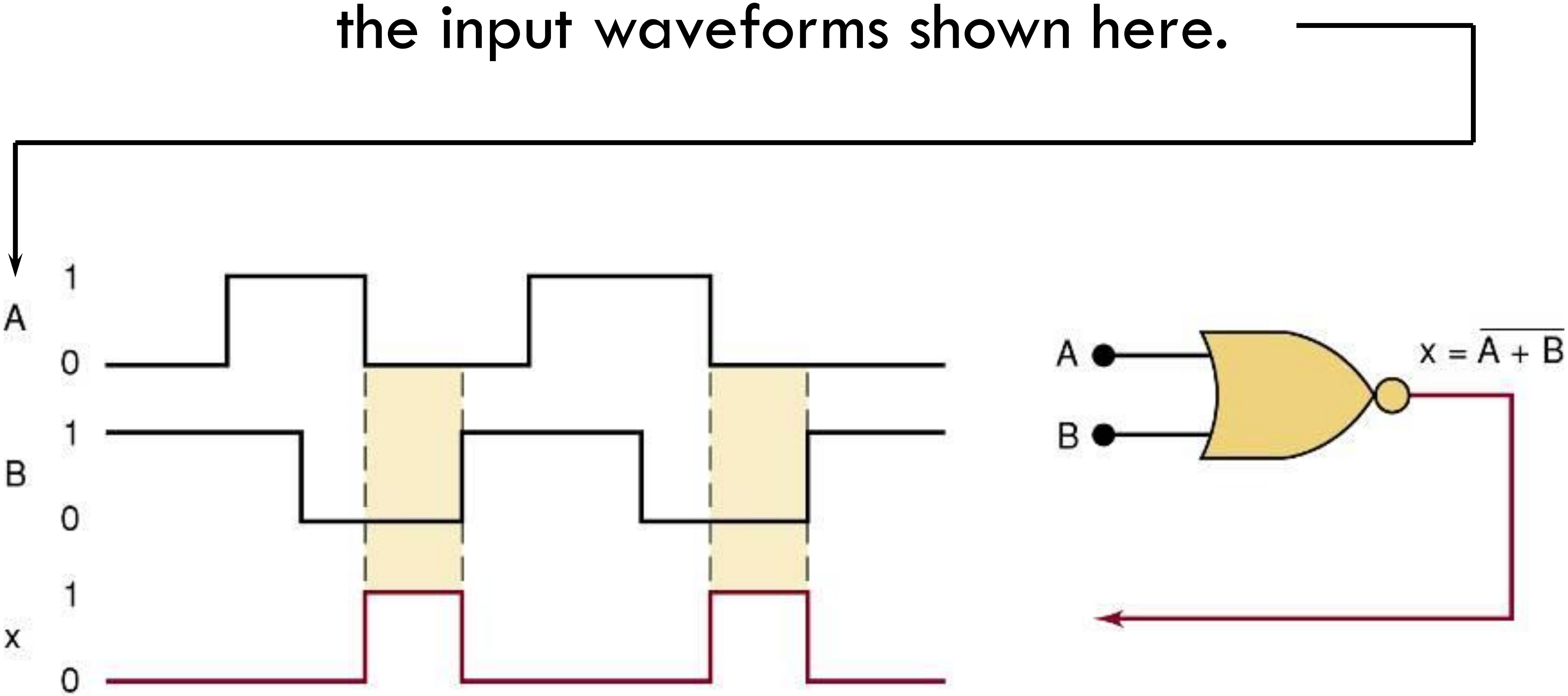
Chapter 3-9 NOR Gates and NAND Gates

- The **NOR** gate is an inverted **OR** gate.
  - An inversion "bubble" is placed at the output of the OR gate, making the Boolean output expression $x = \overline{A + B}$



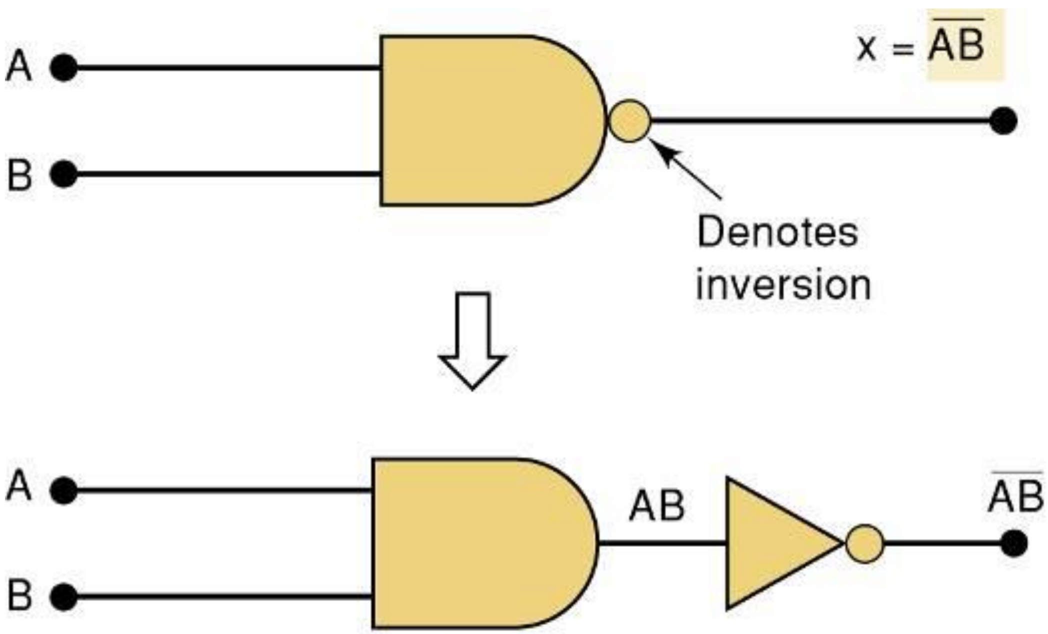
		OR	NOR
A	B	$A + B$	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Output waveform of a **NOR** gate for the input waveforms shown here.



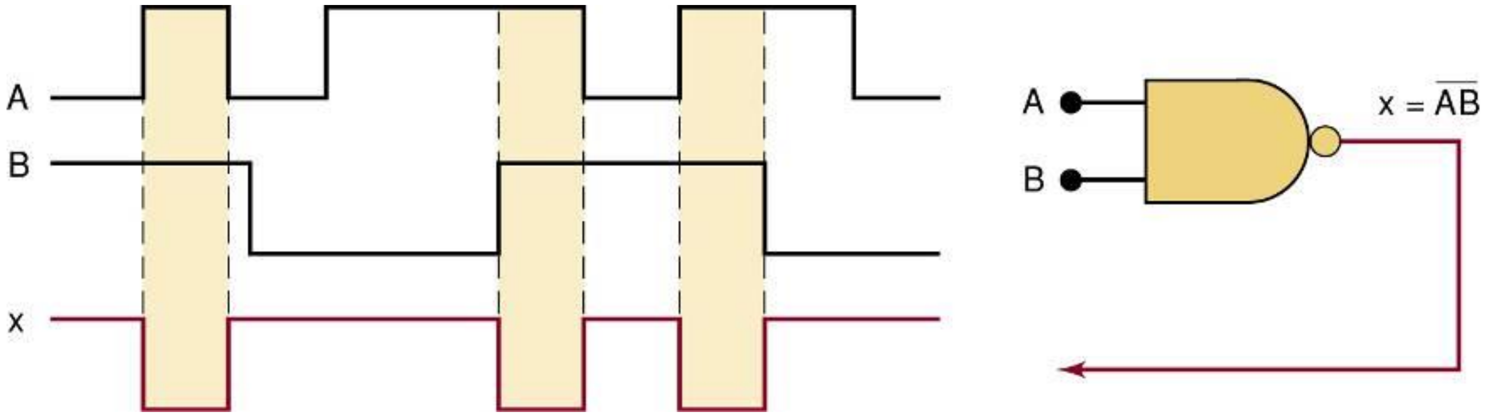
Chapter 3-9 NOR Gates and NAND Gates

- The NAND gate is an inverted AND gate.
  - An inversion “bubble” is placed at the output of the AND gate, making the Boolean output expression $x = \overline{AB}$

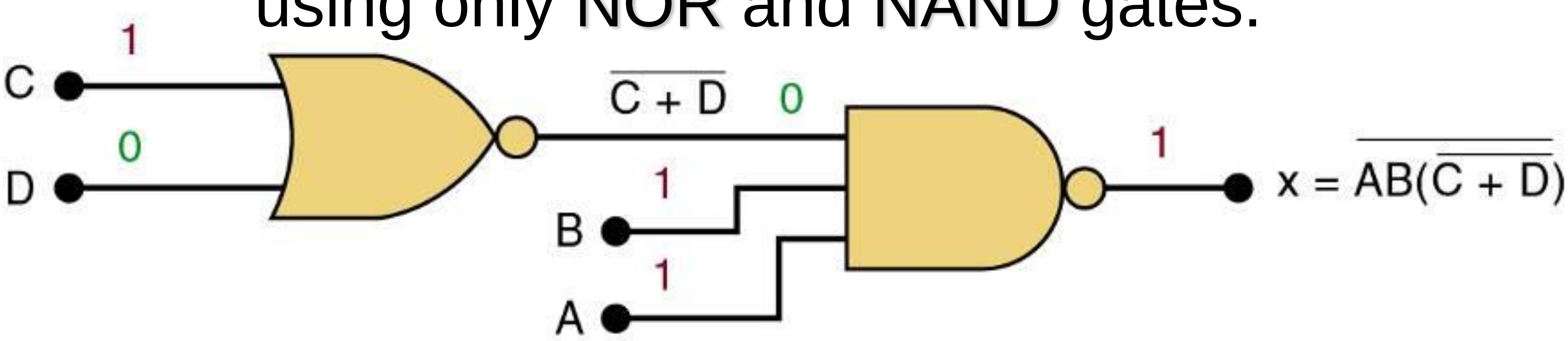


A	B	AND: AB	NAND: $\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Output waveform of a **NAND** gate

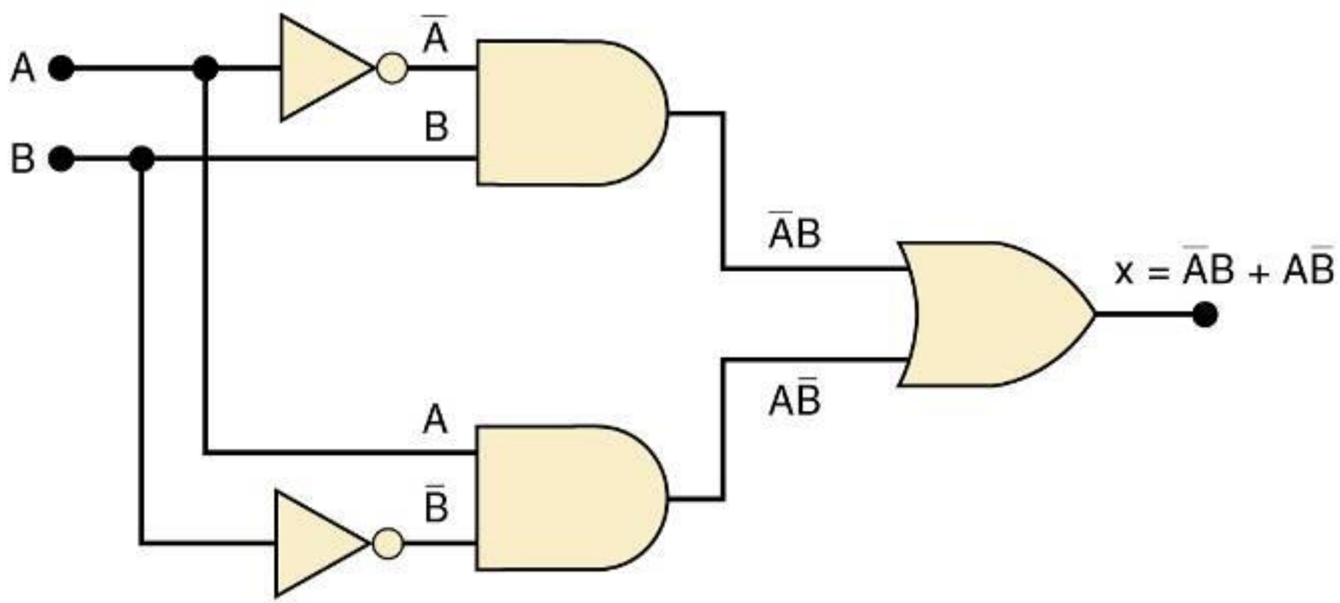


Logic circuit with the expression $x = \overline{AB \cdot (\overline{C + D})}$ using only NOR and NAND gates.



3-10 Exclusive OR and Exclusive NOR Circuits

- The exclusive **OR** (**XOR**) produces a HIGH output whenever the two inputs are at *opposite* levels.



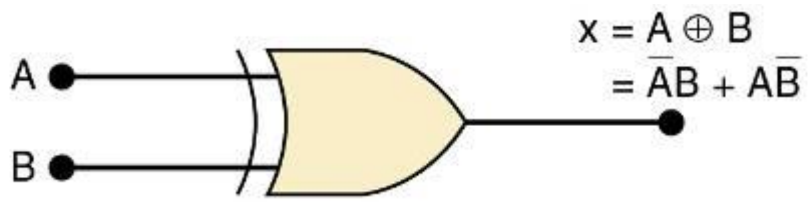
A	B		x
0	0		0
0	1		1
1	0		1
1	1		0

Output expression: $\boldsymbol{x = \bar{A}B + A\bar{B}}$

This circuit produces a HIGH output whenever the two inputs are at opposite levels.

3-10 Exclusive OR and Exclusive NOR Circuits

Traditional **XOR** gate symbol.



An **XOR** gate has only *two* inputs, combined so that $x = \overline{A}B + A\overline{B}$.

A shorthand way indicate the **XOR** output expression is: $x = A \oplus B$.

…where the symbol $\oplus$ represents the **XOR** gate operation.

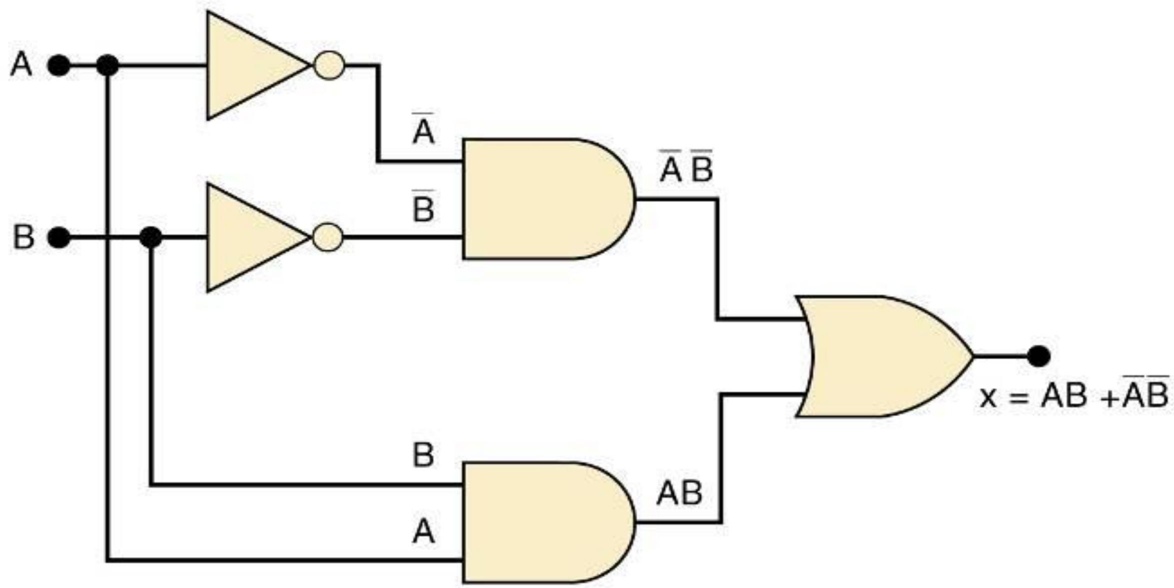
Output is HIGH only when the two inputs are at different levels.

Quad XOR chips containing four XOR gates.

74LS86 Quad **XOR** (TTL family)
74C86 Quad **XOR** (CMOS family)
74HC86 Quad **XOR** (high-speed CMOS)

3-10 Exclusive OR and Exclusive NOR Circuits

- The exclusive **NOR** (**XNOR**) produces a HIGH output whenever the two inputs are at the *same* level.
  - **XOR** and **XNOR** outputs are opposite.

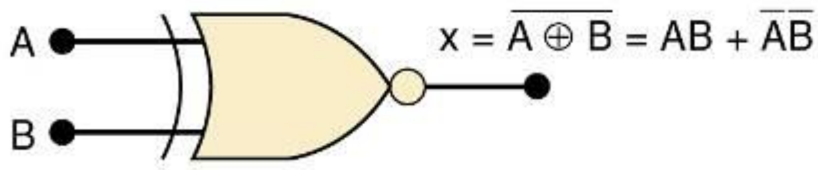


A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

Output expression: $\boldsymbol{x = AB + \overline{A}\,\overline{B}}$

XNOR produces a HIGH output whenever the two inputs are at the same levels.

Traditional **XNOR** gate symbol.



An **XNOR** gate has only *two* inputs, combined so that $\boldsymbol{x = AB + \bar{A}\bar{B}}$.

A shorthand way indicate the **XOR** output expression is: $\boldsymbol{x = \overline{A \oplus B}}$.

XNOR represents inverse of the **XOR** operation.

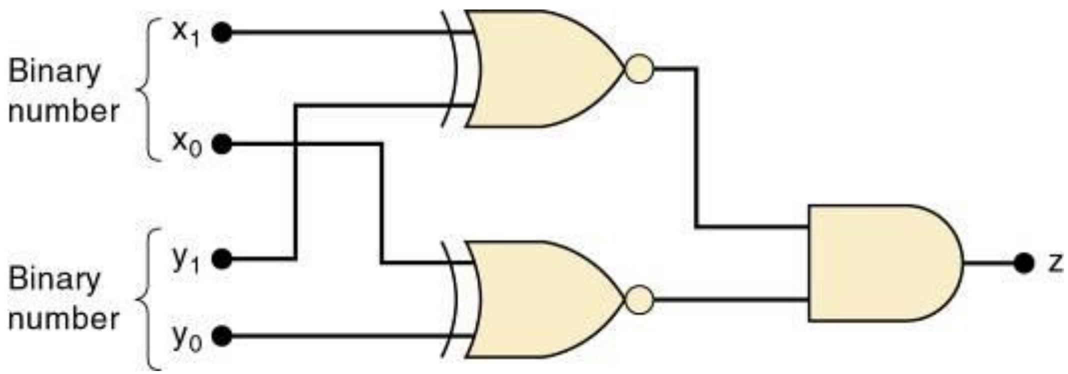
Output is HIGH only when the two inputs are at the same level.

Quad XNOR chips with four XNOR gates.

74LS266 Quad **XNOR** (TTL family)
74C266 Quad **XOR** (CMOS)
74HC266 Quad **XOR** (high-speed CMOS)

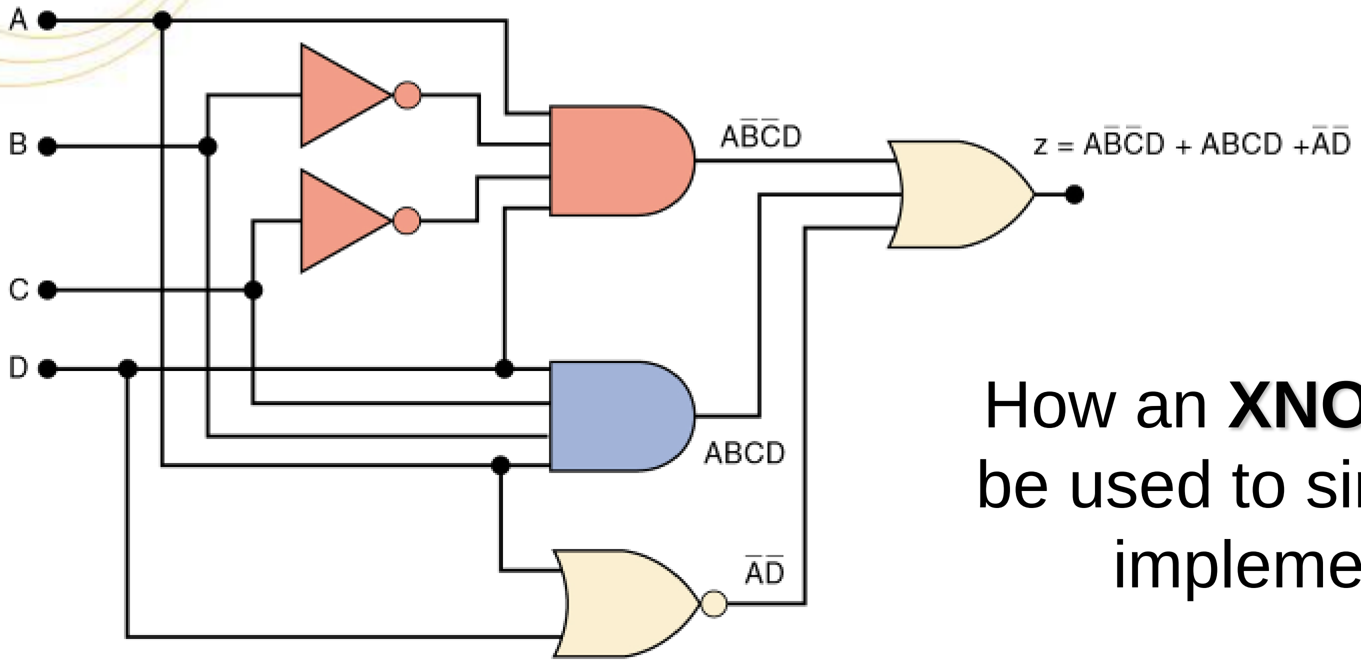
3-10 Exclusive OR and Exclusive NOR Circuits

Truth table and circuit for detecting equality of two-bit binary numbers.

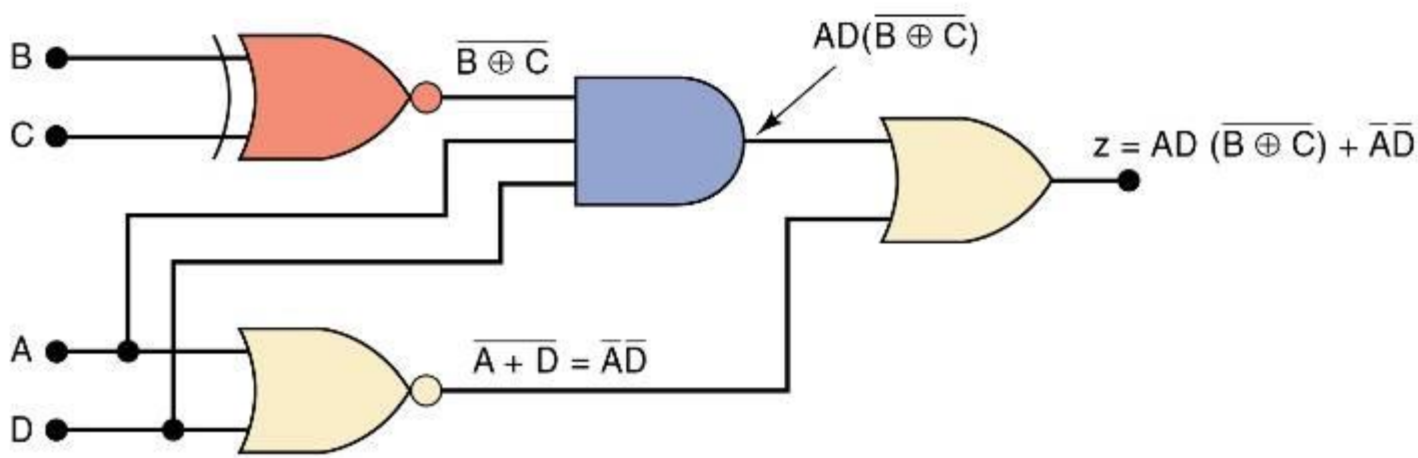


x_1	x_0	y_1	y_0	z (Output)
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

3-10 Exclusive OR and Exclusive NOR Circuits

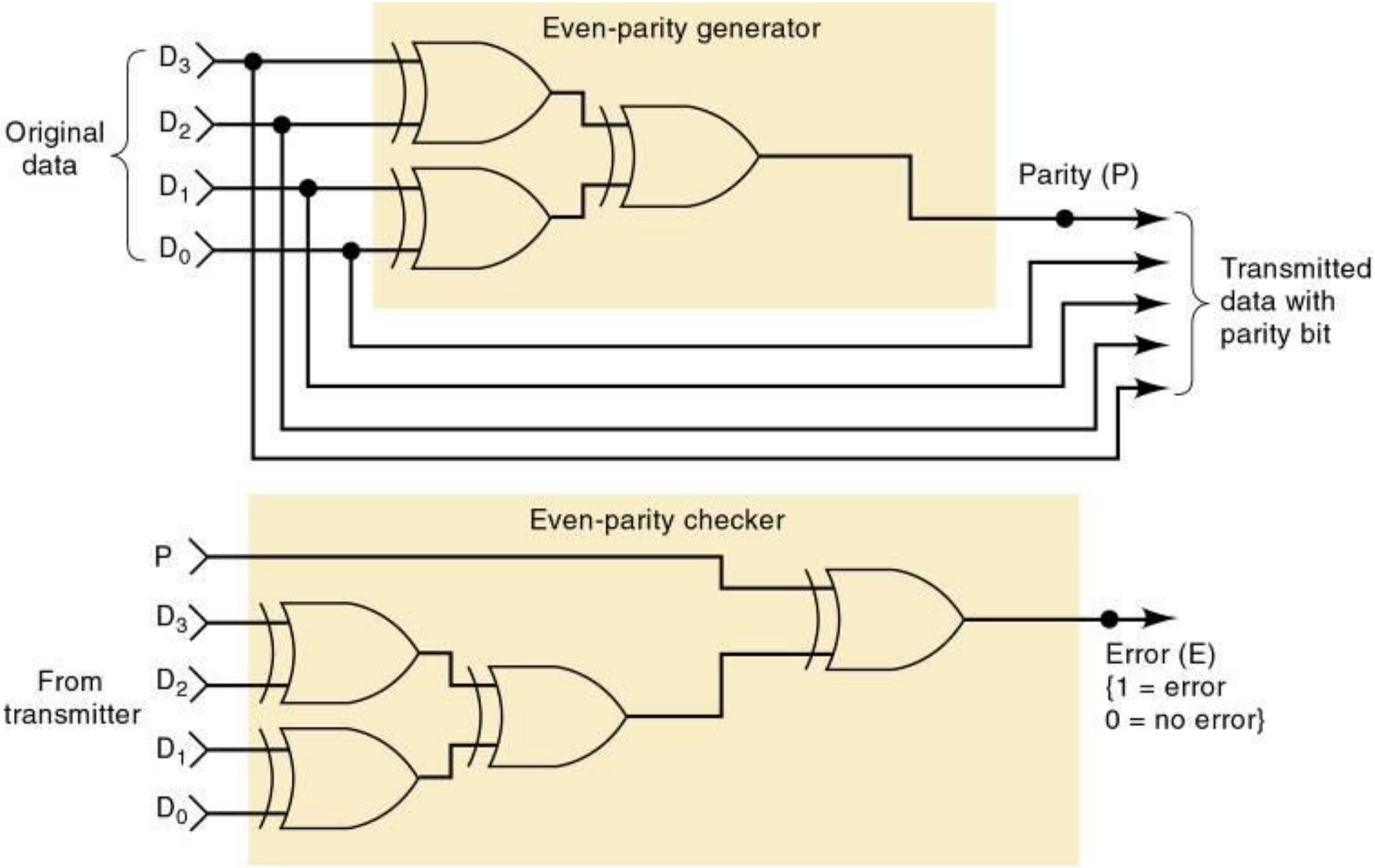


How an **XNOR** gate may be used to simplify circuit implementation.



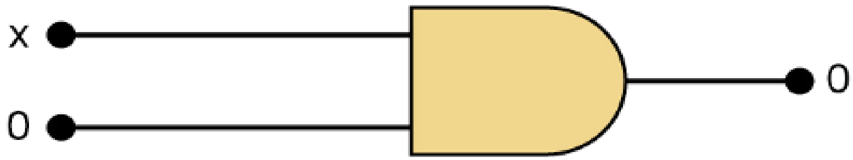
3-11 Parity Generator and Checker

XOR and **XNOR** gates are useful in circuits for parity generation and checking.



Chapter 3-12 Boolean Theorems

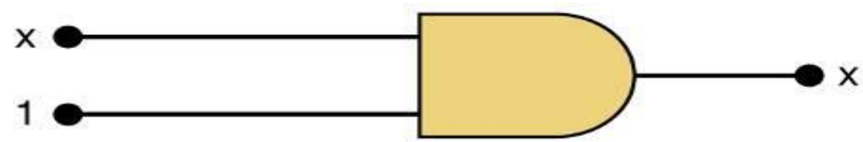
The theorems or laws that follow may represent an expression containing more than one variable.



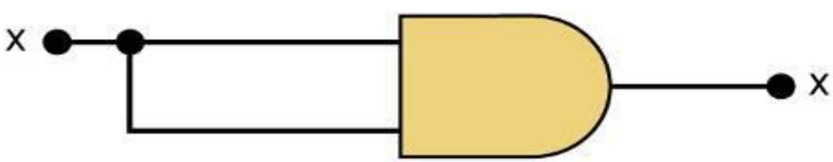
(1) $x \cdot 0 = 0$

Theorem (1) states that if any variable is ANDed with 0, the result must be 0.

Theorem (2) is also obvious by comparison with ordinary multiplication.



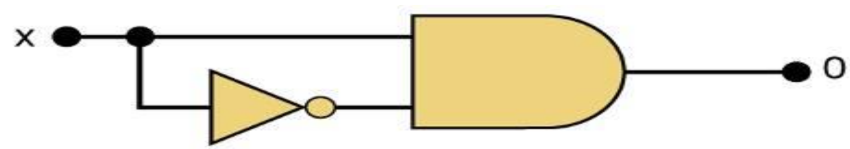
(2) $x \cdot 1 = x$



(3) $x \cdot x = x$

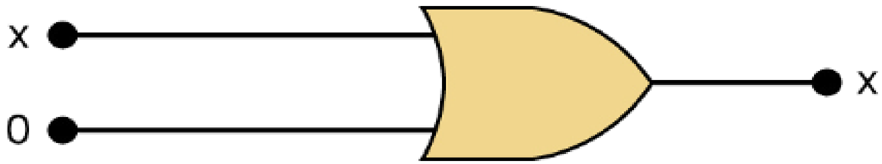
Prove Theorem (3) by trying each case.
If *x* = 0, then 0 • 0 = 0
If *x* = 1, then 1 • 1 = 1
Thus, *x* • x = x

Theorem (4) can be proved in the same manner.



(4) $x \cdot \overline{x} = 0$

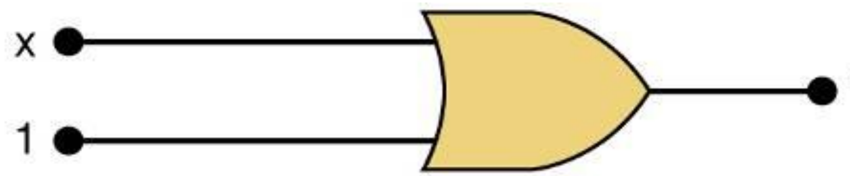
Chapter 3-12 Boolean Theorems



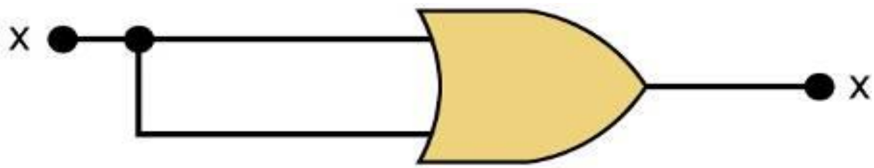
(5) $x + 0 = x$

Theorem (5) is straightforward, as 0 *added* to anything does not affect value, either in regular addition or in OR addition.

Theorem (6) states that if any variable is ORed with 1, the is always 1. Check values: 0 + 1 = 1 and 1 + 1 = 1.



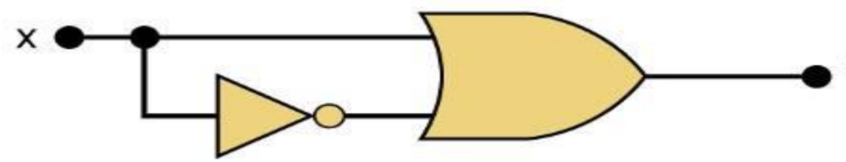
(6) $x + 1 = 1$



(7) $x + x = x$

Theorem (7) can be proved by checking for both values of x: 0 + 0 = 0 and 1 + 1 = 1.

Theorem (8) can be proved similarly.



(8) $x + \overline{x} = 1$

Multivariable Theorems

Commutative laws

(9) $x + y = y + x$

(10) $x \cdot y = y \cdot x$

Associative laws

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

(12) $x(yz) = (xy)z = xyz$

Distributive law

(13a) $x(y + z) = xy + xz$

(13b) $(w + x)(y + z) = wy + xy + wz + xz$

Chapter 3-12 Boolean Theorems

Theorems (14) and (15) do not have counterparts in ordinary algebra. Each can be proved by trying all possible cases for *x* and *y*.

$$(14) \quad x + xy = x$$
$$(15a) \quad x + \bar{x}y = x + y$$
$$(15b) \quad \bar{x} + xy = \bar{x} + y$$

Analysis table & factoring for Theorem (14)

$$\begin{aligned} x + xy &= x(1 + y) \\ &= x \cdot 1 && \text{[using theorem (6)]} \\ &= x && \text{[using theorem (2)]} \end{aligned}$$

x	y	xy	x + xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Chapter 3-13 DeMorgan's theorems

- **DeMorgan's theorems** are extremely useful in simplifying expressions in which a product or sum of variables is inverted.

$$(16) \quad \overline{(x + y)} = \overline{x} \cdot \overline{y}$$

Theorem (16) says inverting the OR sum of two variables is the same as inverting each variable individually, then ANDing the inverted variables.

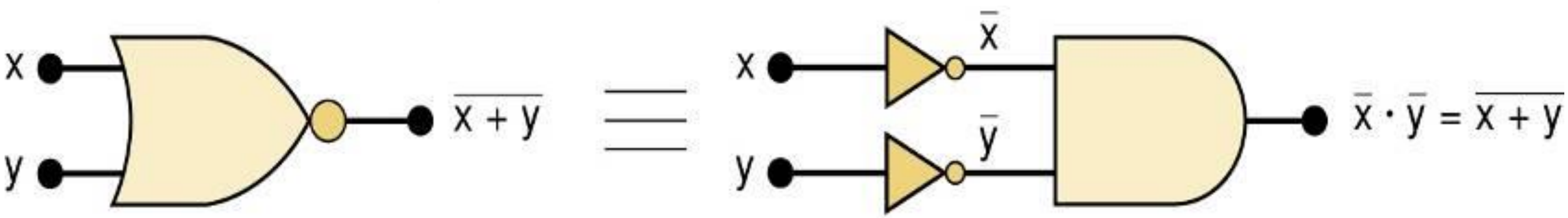
$$(17) \quad \overline{(x \cdot y)} = \overline{x} + \overline{y}$$

Theorem (17) says inverting the AND product of two variables is the same as inverting each variable individually and then ORing them.

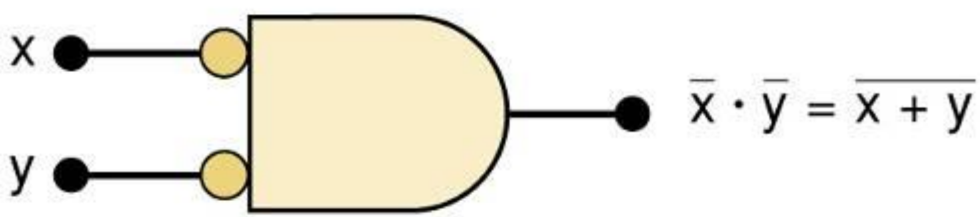
Each of DeMorgan's theorems can readily be proven by checking for all possible combinations of *x* and *y.*

Equivalent circuits implied by Theorem (16)

$$(16) \quad \overline{(x + y)} = \overline{x} \cdot \overline{y}$$



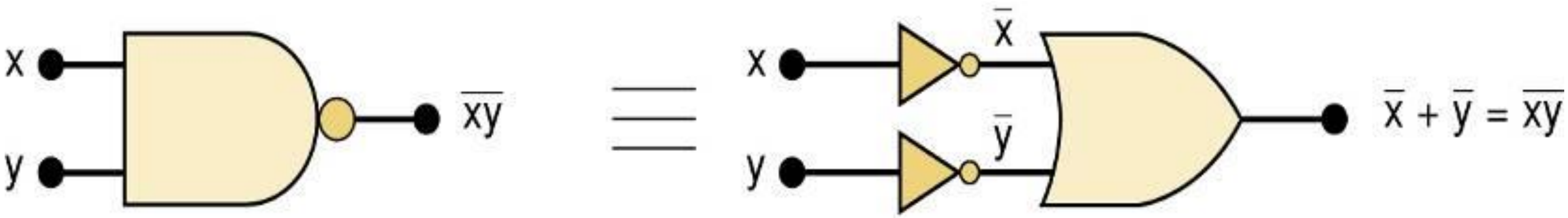
The alternative symbol for the NOR function.



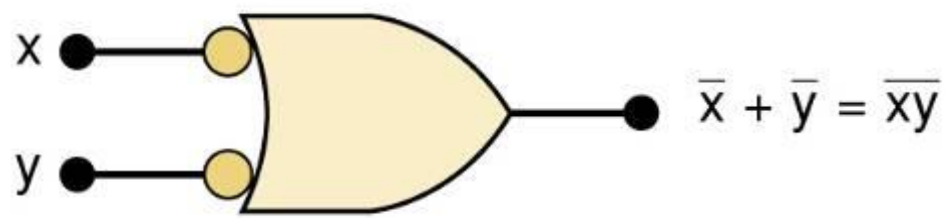
Chapter 3-13 DeMorgan's Theorems

Equivalent circuits implied by Theorem (17)

$$(17)\quad \overline{(x\cdot y)} = \overline{x} + \overline{y}$$



The alternative symbol
for the NAND function.



Chapter 3 - Example

Example

Simplify the following expression using DeMorgan's theorems

(a) $\overline{(A + B + C)D}$

(b) $\overline{ABC + DEF}$

(c) $\overline{A\overline{B} + \overline{C}D + EF}$

Solution

(a) Let $A + B + C = X$ and $D = Y$. The expression $\overline{(A + B + C)D}$ is of the form $\overline{XY} = \overline{X} + \overline{Y}$ and can be rewritten as

$$\overline{(A + B + C)D} = \overline{A + B + C} + \overline{D}$$

Next, apply DeMorgan's theorem to the term $\overline{A + B + C}$.

$$\overline{A + B + C} + \overline{D} = \overline{A}\,\overline{B}\,\overline{C} + \overline{D}$$

Chapter 3 - Example

(b) Let $ABC = X$ and $DEF = Y$. The expression $\overline{ABC + DEF}$ is of the form $\overline{X + Y} = \overline{X}\,\overline{Y}$ and can be rewritten as

$$\overline{ABC + DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{ABC}$ and $\overline{DEF}$.

$$(\overline{ABC})(\overline{DEF}) = (\overline{A} + \overline{B} + \overline{C})(\overline{D} + \overline{E} + \overline{F})$$

(c) Let $A\overline{B} = X$, $\overline{C}D = Y$, and $EF = Z$. The expression $\overline{A\overline{B} + \overline{C}D + EF}$ is of the form $\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$ and can be rewritten as

$$\overline{A\overline{B} + \overline{C}D + EF} = (\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{A\overline{B}}$, $\overline{\overline{C}D}$, and $\overline{EF}$.

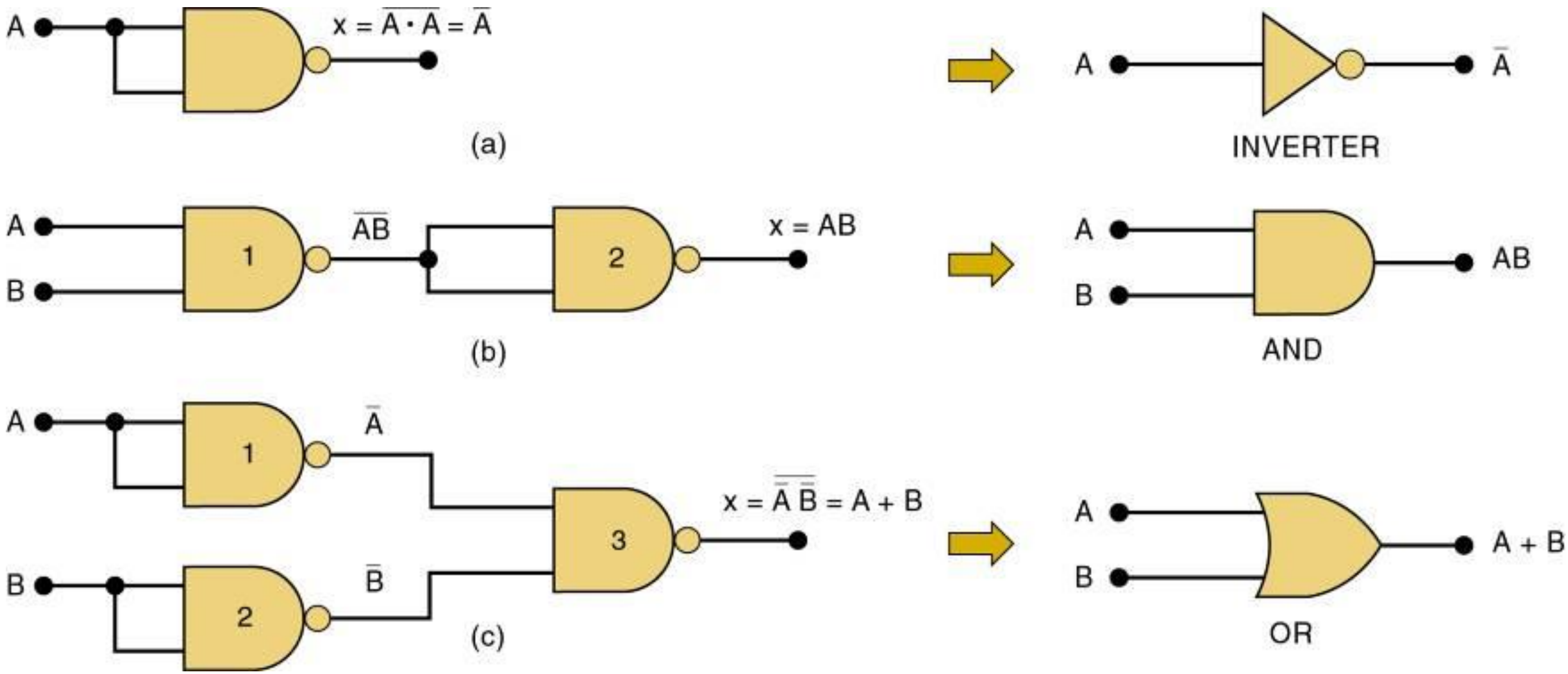
$$(\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF}) = (\overline{A} + B)(C + \overline{D})(\overline{E} + \overline{F})$$

Chapter 3-14 Universality of NAND and NOR Gates

- **NAND** or **NOR** gates can be used to create the three basic logic expressions.
  - **OR**, **AND**, and INVERT.
    - Provides flexibility—very useful in logic circuit design.

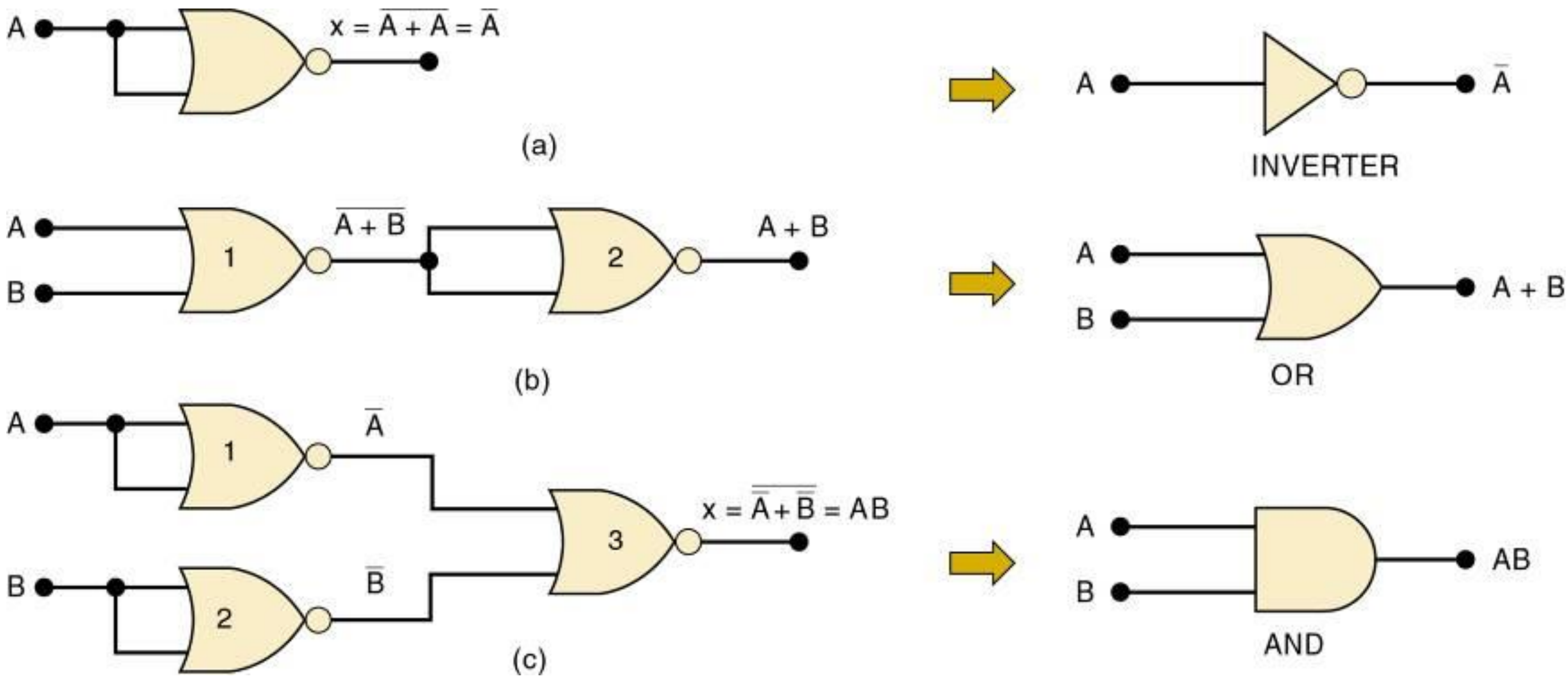
Chapter 3-14 Universality of NAND and NOR Gates

How combinations of NANDs or NORs are used to create the three logic functions.



It is possible, however, to implement any logic expression using *only* NAND gates and no other type of gate, as shown.

Chapter 3-14 Universality of NAND and NOR Gates



NOR gates can be arranged to implement any of the Boolean operations, as shown.

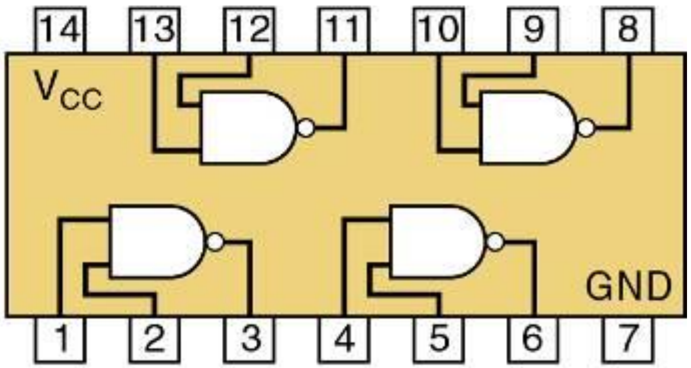
Chapter 3-15 Example

A logic circuit to generate a signal *x*, that will go HIGH whenever conditions *A* and *B* exist simultaneously, or whenever conditions *C* and *D* exist simultaneously.

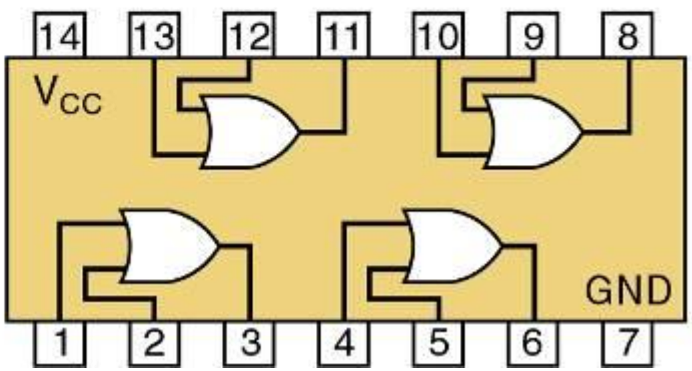
The logic expression will be *x* = *AB* + *CD*.

Each of the TTL ICs shown here will fulfill the function. Each IC is a *quad*, with *four* identical gates on one chip

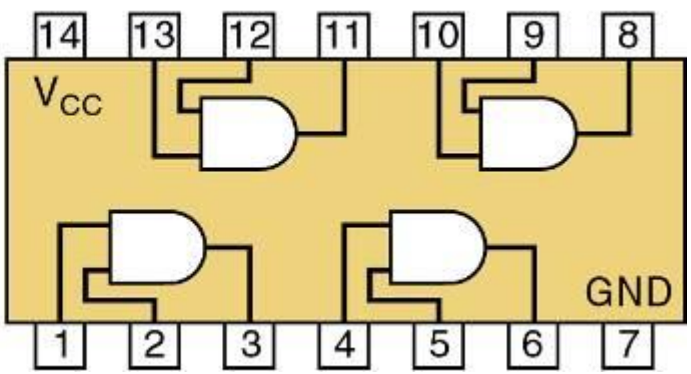
74LS00



74LS32

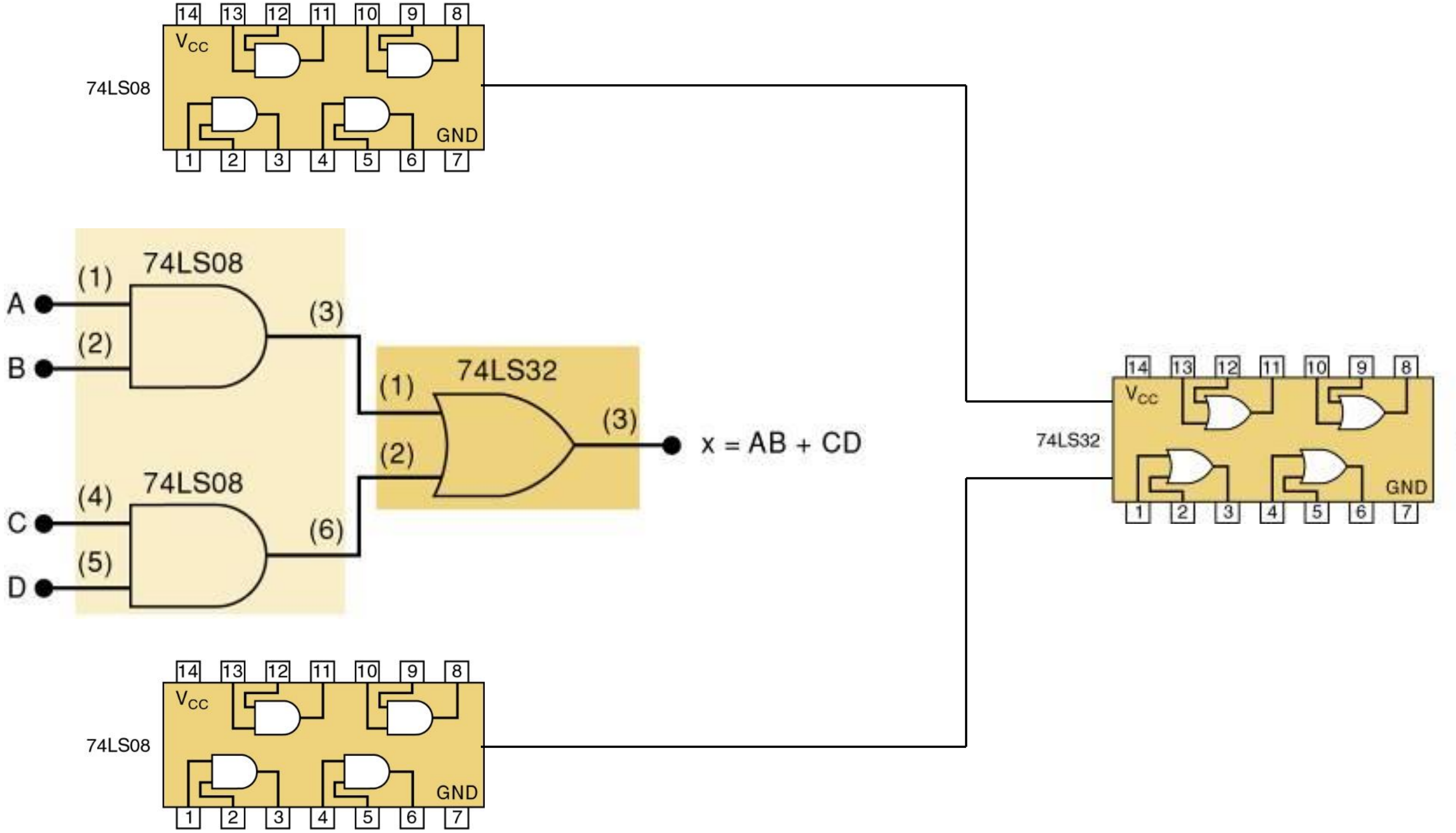


74LS08



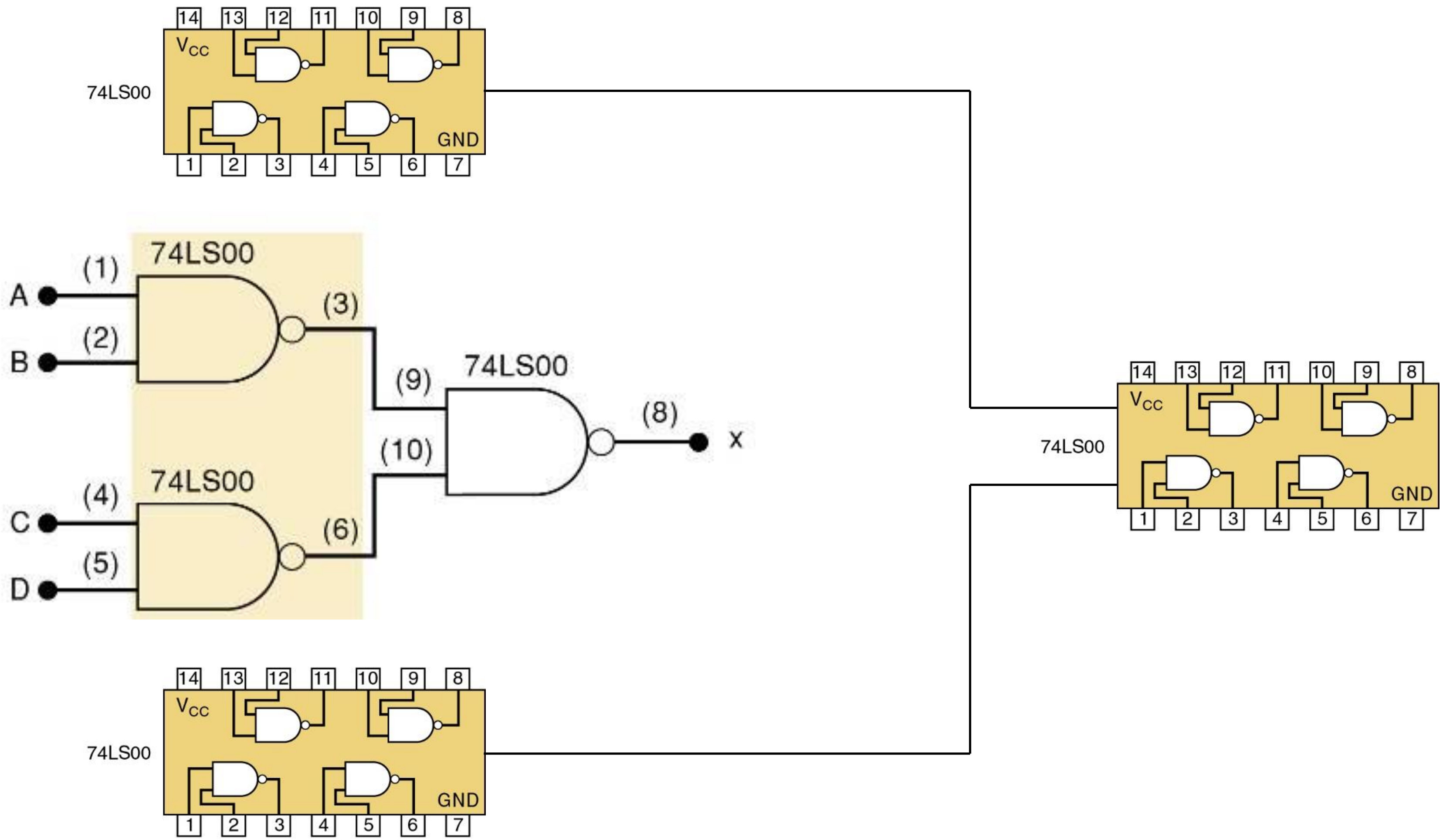
Chapter 3-15 Example

Possible Implementations # 1



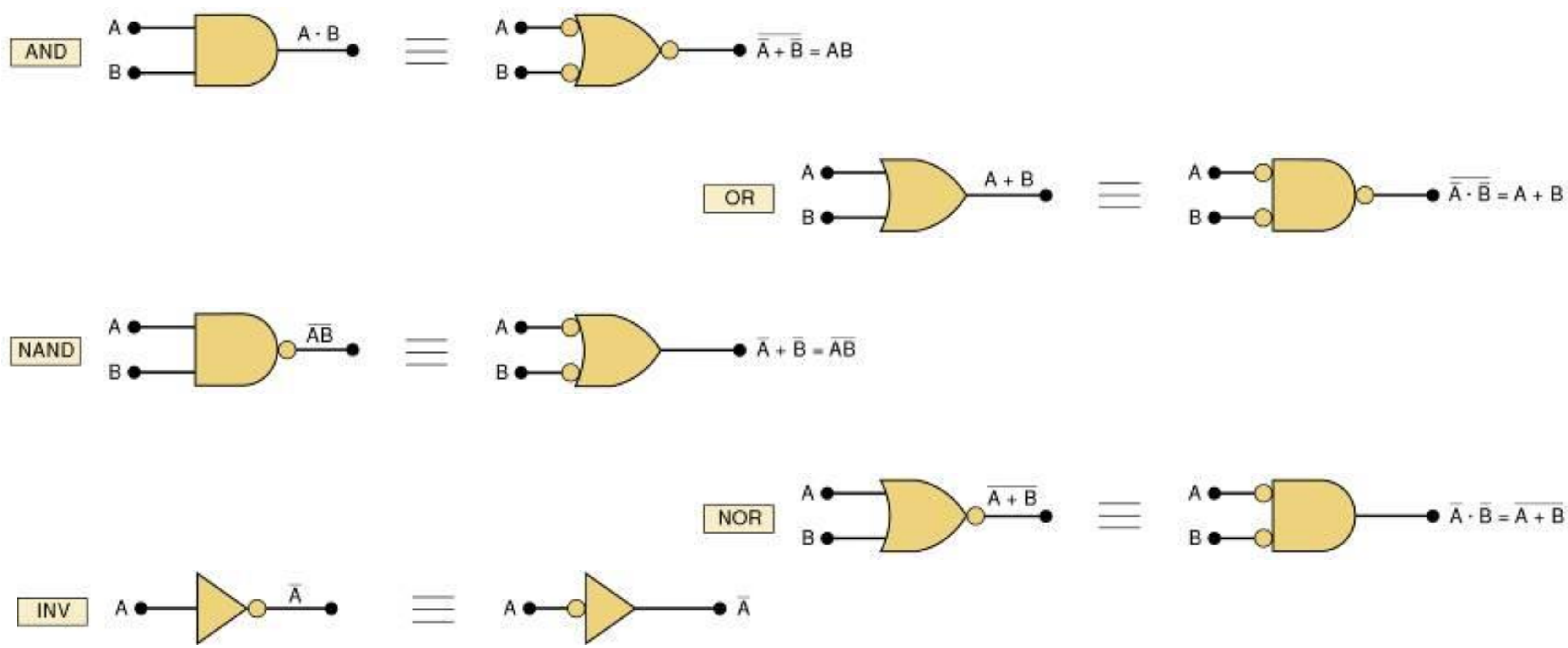
Chapter 3-15 Example

Possible Implementations #2



Chapter 3-16 Alternate Logic-Gate Representations

- To convert a standard symbol to an alternate:
  - Invert each input and output in standard symbols.
    - Add an inversion bubble where there are none.
    - Remove bubbles where they exist.



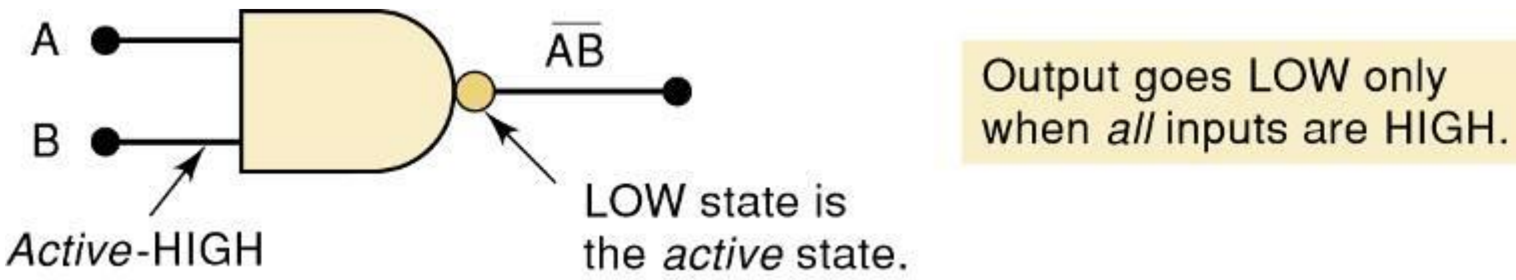
Chapter 3-16 Alternate Logic-Gate Representations

- Points regarding logic symbol equivalences:
  - The equivalences can be extended to gates with *any* number of inputs.
  - None of the standard symbols have bubbles on their inputs, and all the alternate symbols do.
  - Standard & alternate symbols for each gate represent the same physical circuit.
  - **NAND** and **NOR** gates are inverting gates.
    - Both the standard and the alternate symbols for each will have a bubble on *either* the input or the output.
  - **AND** and **OR** gates are *noninverting* gates.
    - The alternate symbols for each will have bubbles on *both* inputs and output.

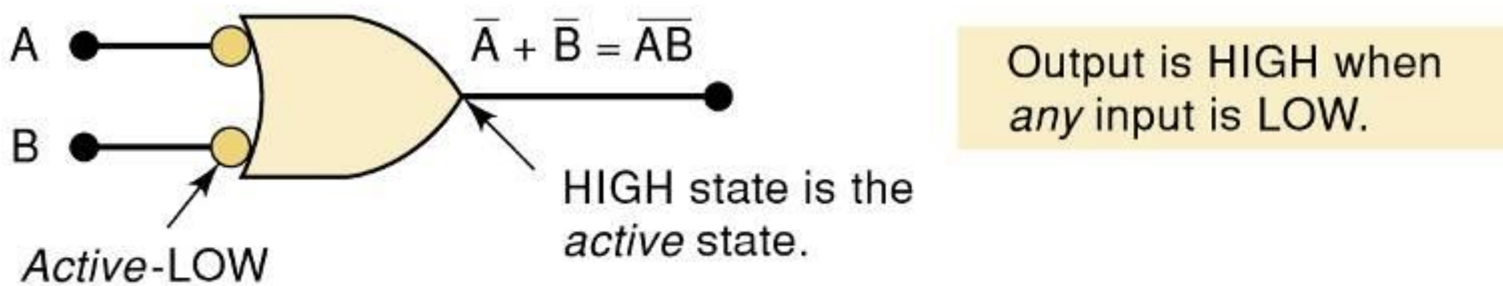
Chapter 3-16 Alternate Logic-Gate Representations

- Active-HIGH – an input/output has *no* inversion bubble.
- Active-LOW – an input or output has an inversion bubble.

Interpretation of the two **NAND** gate symbols.

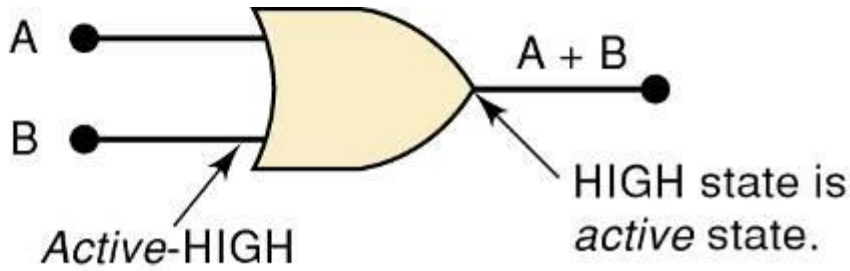


(a)

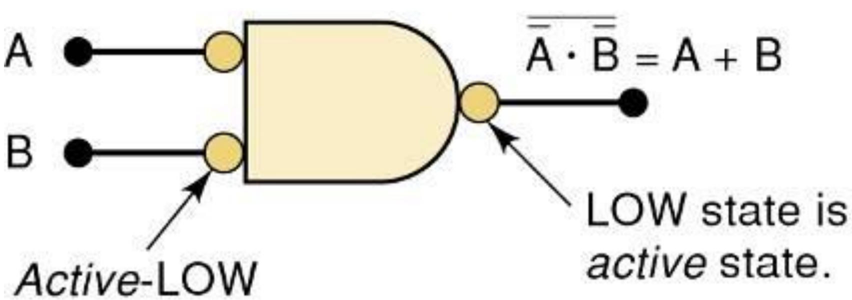


(b)

Interpretation of the two **OR** gate symbols.



Output goes HIGH when *any* input is HIGH.



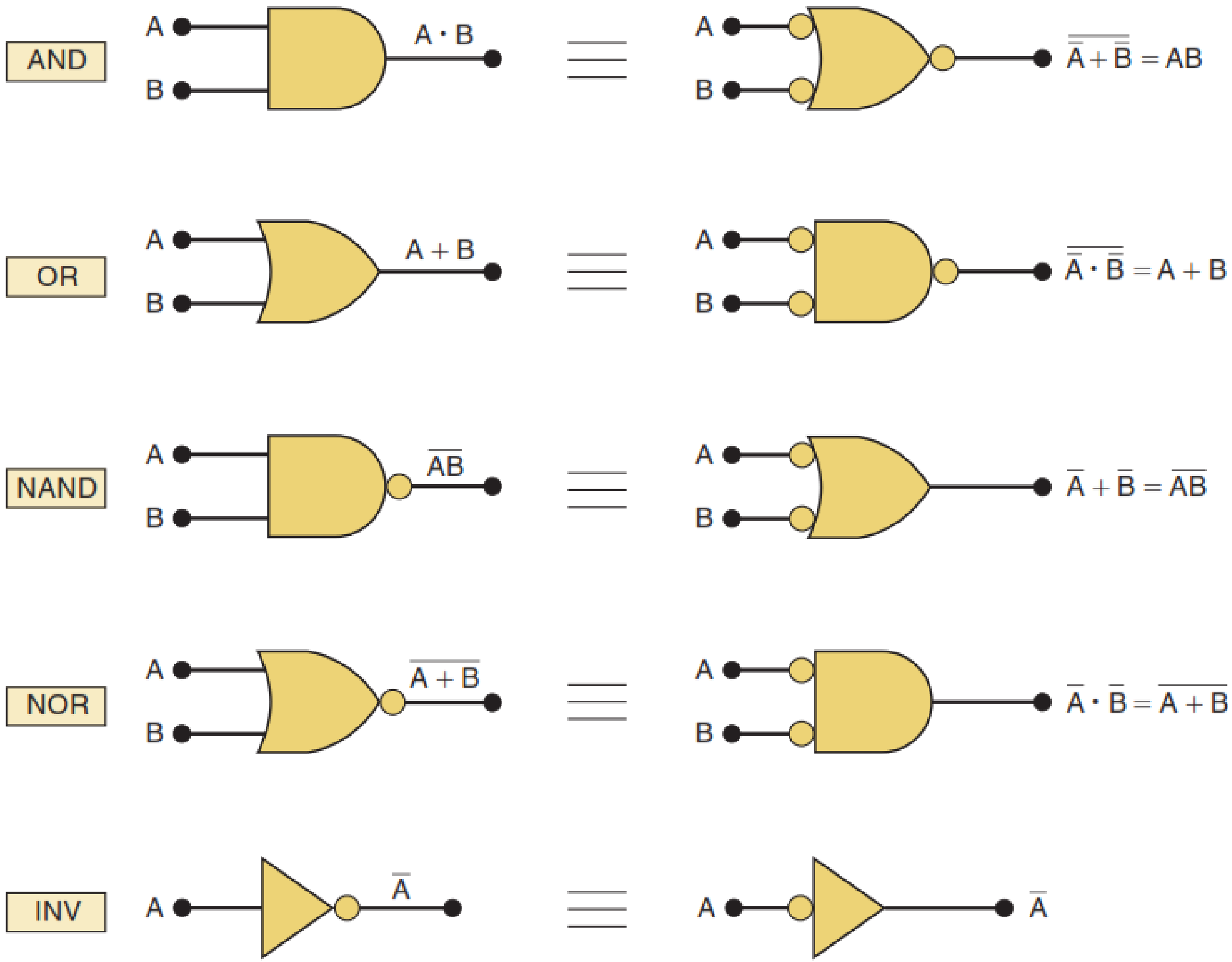
Output goes LOW only when *all* inputs are LOW.

Chapter 3-16 Alternate Logic-Gate Representations

The alternate symbol for each gate is obtained from the standard symbol by doing the following:

- Invert each input and output of the standard symbol. This is done by adding bubbles (small circles) on input and output lines that do not have bubbles and by removing bubbles that are already there.
- Change the operation symbol from AND to OR, or from OR to AND. (In the special case of the INVERTER, the operation symbol is not changed.)

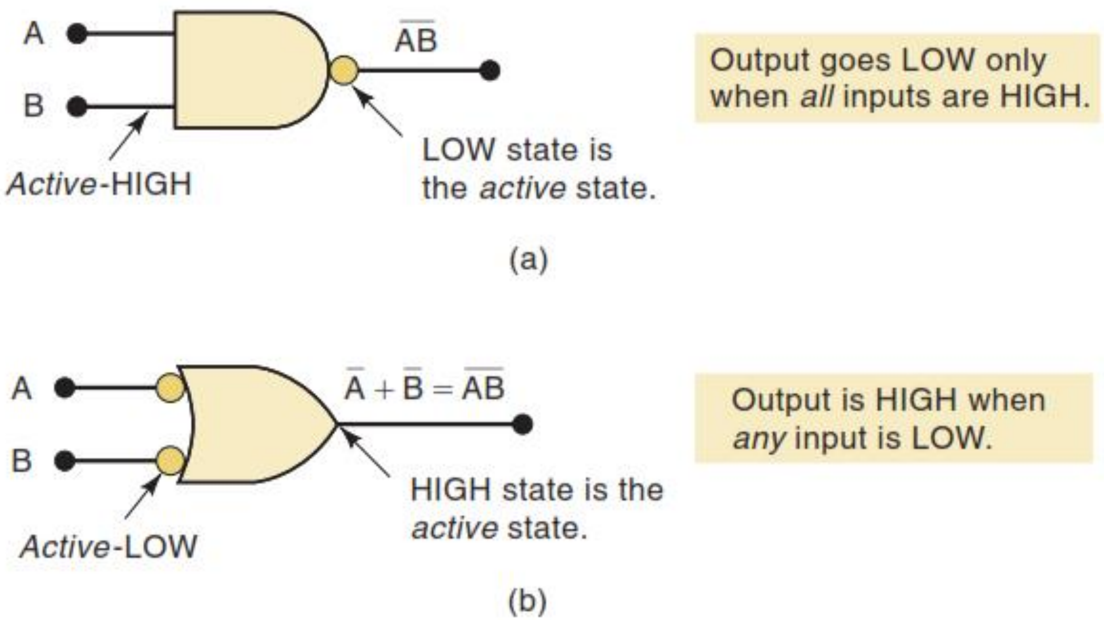
Chapter 3-16 Alternate Logic-Gate Representations



Chapter 3-17 Which Gate Representation to Use

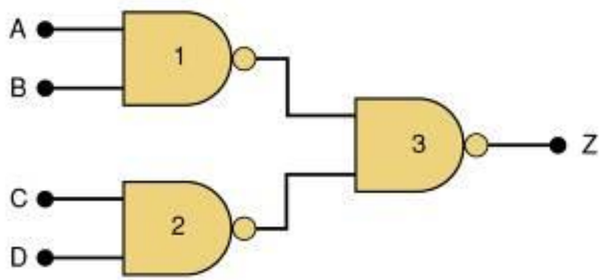
Active Logic Levels

When an input or output line on a logic circuit symbol has no bubble on it, that line is said to be active-HIGH.
When an input or output line does have a bubble on it, that line is said to be active-LOW.



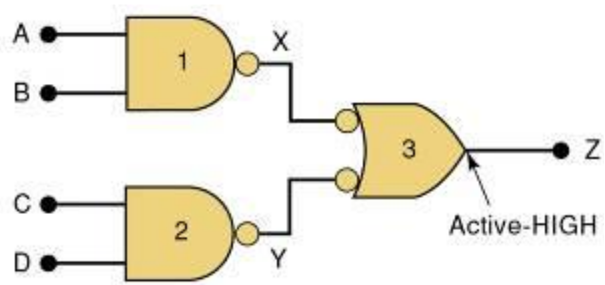
Chapter 3-17 Which Gate Representation to Use

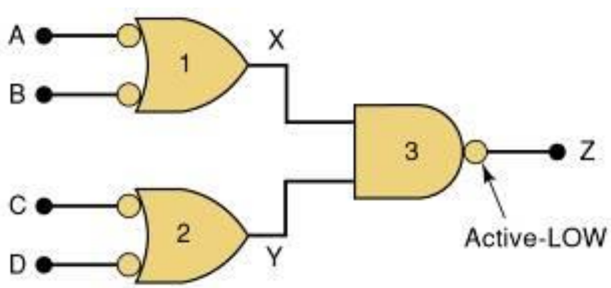
Proper use of alternate gate symbols in the circuit diagram can make circuit operation much clearer.



Original circuit using standard NAND symbols.

Equivalent representation where output *Z* is active-HIGH.





Equivalent representation where output *Z* is active-LOW.

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Chapter 3-17 Which Gate Representation to Use

- When a logic signal is in the *active* state (HIGH or LOW) it is said to be *asserted*.
- When a logic signal is in the *inactive* state (HIGH or LOW) it is said to be *unasserted*.

A bar over a signal means asserted (active) LOW.	$\overline{RD}$	Absence of a bar means asserted (active) HIGH	RD

Chapter 3-17 Which Gate Representation to Use

- An output signal can have two active states, with an important function in the HIGH state, and another in the LOW state.
  - It is customary to label such signals so both active states are apparent.

A common example is the read/write signal.

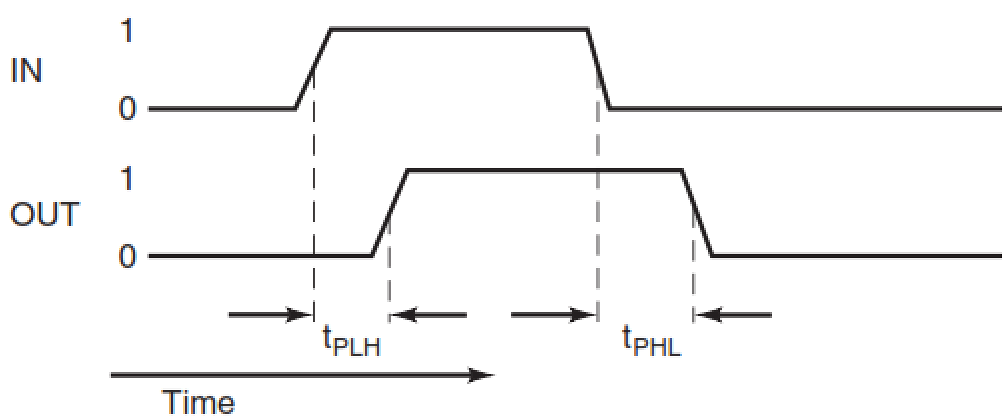
$$RD/\overline{WR}$$

When this signal is HIGH, the read operation (*RD*) is performed; when it is LOW, the write operation ($\overline{WR}$) is performed.

- When possible, choose gate symbols so bubble outputs are connected to bubble input.
  - Nonbubble outputs connected to nonbubble inputs.

Chapter 3-18 Propagation Delay

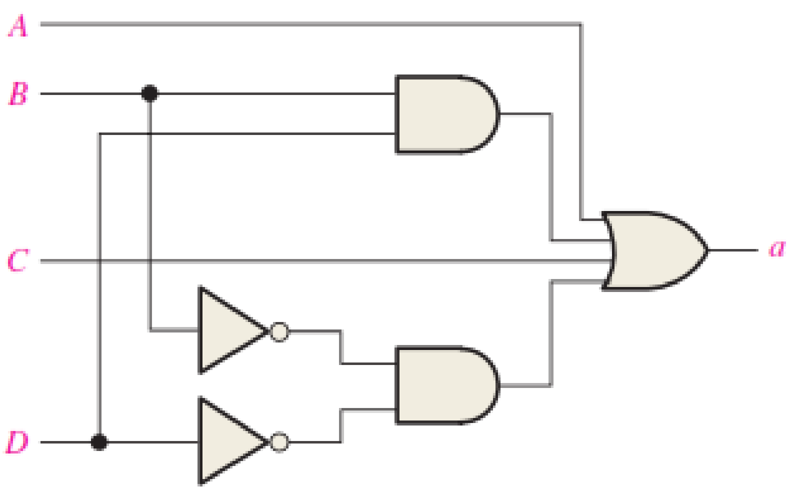
- Propagation delay is the time it takes for a system to produce output after it receives an input.
  - Speed of a logic circuit is related to propagation delay.
- Parts to implement logic circuits have a data sheet that states the value of propagation delay.
  - Used to assure that the circuit can operate fast enough for the application.



Example 1

Write the Boolean expression for output x

Determine the value of x for all possible input conditions, and list the values in a truth table.



Example 2

For each of the following expressions, construct the corresponding logic circuit, using AND and OR gates and INVERTERs.

.

$$\text{(a)*}\; x = \overline{AB(C + D)}$$

$$\text{(b)*}\; z = \overline{A + B + \overline{C}D\overline{E})} + \overline{B}C\overline{D}$$

$$\text{(c)}\;\; y = (\overline{M + N} + \overline{P}Q)$$

$$\text{(d)}\;\; x = \overline{W + P\overline{Q}}$$

$$\text{(e)}\;\; z = MN(P + \overline{N})$$

$$\text{(f)}\;\; x = (A + B)(\overline{A} + \overline{B})$$

$$\text{(g)}\;\; g = AC + B\overline{C}$$

$$\text{(h)}\;\; h = \overline{\overline{AB} + \overline{CD}}$$

Chapter 3 - Example

Example 3

Simplify the following expression.

(a) $(\overline{A} + B)(A + C)$

(b) $A\overline{B} + A\overline{B}C + A\overline{B}CD + A\overline{B}CDE$

(c) $BC + \overline{BCD} + B$

(d) $(B + \overline{B})(BC + BC\overline{D})$

(e) $BC + (\overline{B} + \overline{C})D + BC$

Example 4

Simplify the following expression using DeMorgan's theorems

$$\overline{(\overline{A + B}) + \overline{C}}$$

$$\overline{(\overline{A} + B) + CD}$$

$$\overline{(A + B)\overline{C}\,\overline{D} + E + \overline{F}}$$

(a)* $\overline{\overline{A}B\overline{C}}$

(b) $\overline{\overline{A} + \overline{B}C}$

(c)* $\overline{AB\overline{CD}}$

(d) $\overline{A + \overline{B}}$

(e)* $\overline{\overline{AB}}$

(f) $\overline{\overline{A} + \overline{C} + \overline{D}}$

(g)* $\overline{A(\overline{B + \overline{C}})D}$

(h) $\overline{(M + \overline{N})(\overline{M} + N)}$

(i) $\overline{\overline{\overline{AB}C}D}$

Chapter 3 - Example

Example 5

Simplify the following expression.

$$[A\overline{B}(C + BD) + \overline{A}\,\overline{B}]C$$

$$\overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + ABC$$

$$\overline{AB + AC} + \overline{A}\,\overline{B}C$$

Example 6

Simplify the following expression.

(a) $CE + C(E + F) + \overline{E}(E + G)$

(b) $\overline{B}\,\overline{C}D + (\overline{B + C + D}) + \overline{B}\,\overline{C}\,\overline{D}E$

(c) $(C + CD)(C + \overline{C}D)(C + E)$

(d) $BCDE + BC(\overline{DE}) + (\overline{BC})DE$

(e) $BCD[BC + \overline{D}(CD + BD)]$