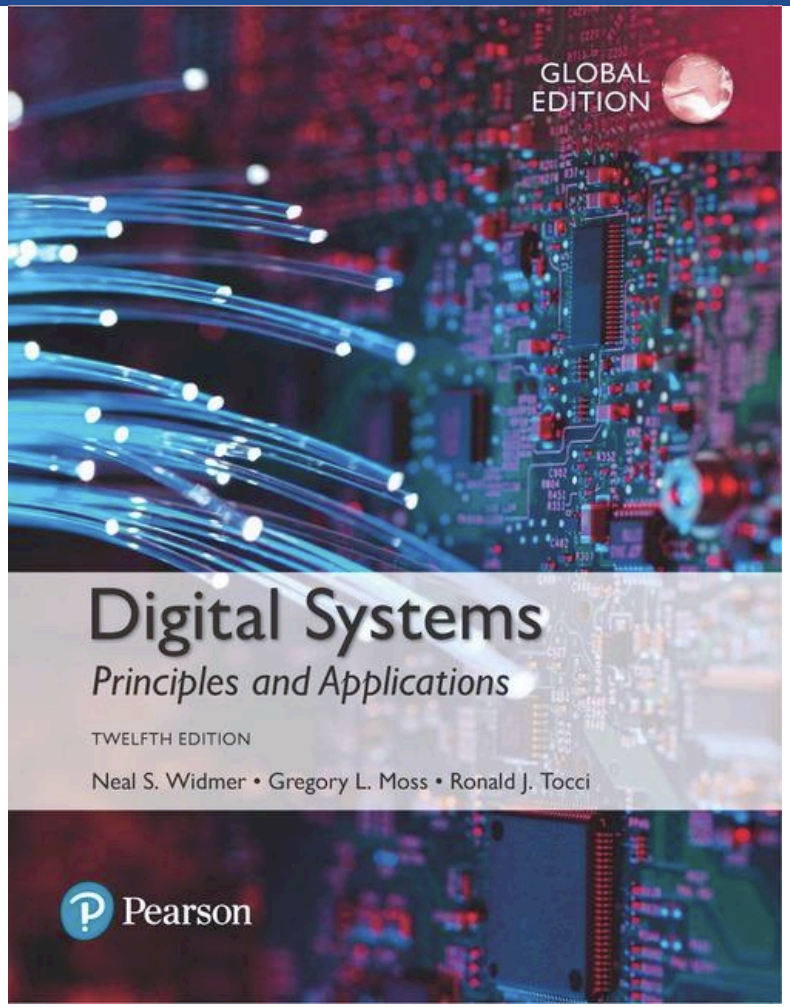
Digital Systems Principles and Applications

TWELFTH EDITION, GLOBAL EDITION



CHAPTER 3

Describing Logic Circuits

Chapter 3 Objectives

- Perform the three basic logic operations.
- Describe the operation of and construct the truth tables for the AND, NAND, OR, and NOR gates, and the NOT (INVERTER) circuit.
- Draw timing diagrams for the various logic-circuit gates.

Chapter 3 Objectives

- Write the Boolean expression for the logic gates and combinations of logic gates.
- Implement logic circuits using basic AND, OR, and NOT gates.
- Use Boolean algebra to simplify complex logic circuits.
- Use DeMorgan's theorems to simplify logic expressions.

Chapter 3 Objectives

- Use either of the universal gates (NAND or NOR) to implement a circuit represented by a Boolean expression.
- Explain the advantages of constructing a logic-circuit diagram using the alternate gate symbols versus the standard logic-gate symbols.
- Describe the concept of active-LOW and active-HIGH logic signals.

Chapter 3 Objectives

- Describe and measure propagation delay time.
- Use several methods to describe the operation of logic circuits.
- Interpret simple circuits defined by a hardware description language (HDL).
- Explain the difference between an HDL and a computer programming language.

Chapter 3 Objectives

- Create an HDL file for a simple logic gate.
- Create an HDL file for combinational circuits with intermediate variables.

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

- The three basic logic operations:
  - OR, AND, and NOT.

3-2 Truth Tables

- 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 22 = 4 entries.
  - A 3-input table would have 23 = 8 entries.

3-2 Truth Tables

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)

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:

$1 + 1 = 1$ *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)

3-3 OR Operation With OR Gates

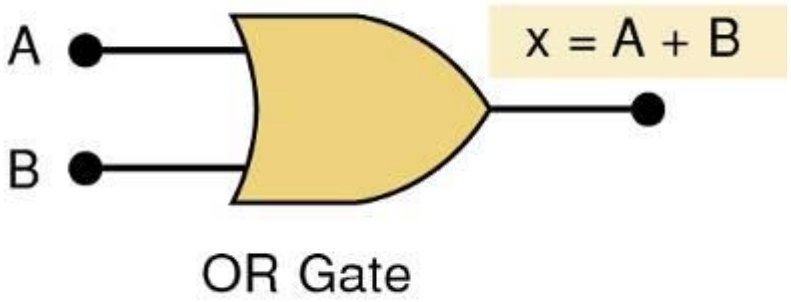
- An OR gate is a circuit with two or more inputs, whose output is equal to the OR combination of the inputs.

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

(a)

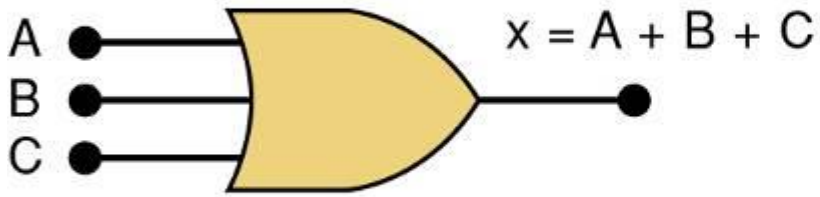


(b)

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.

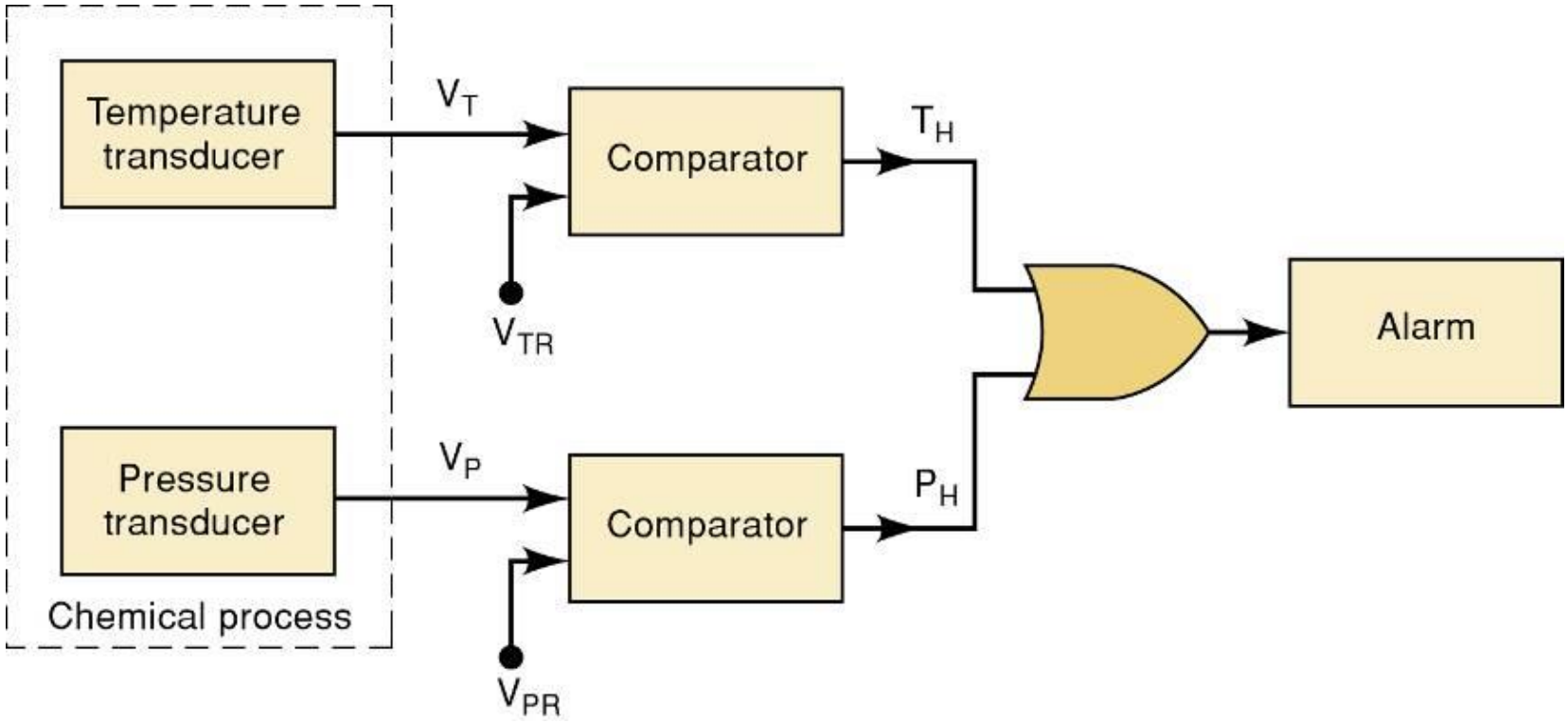
Truth table/circuit symbol for a three input OR gate.



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

3-3 OR Operation With OR Gates

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



3-4 AND Operations with AND gates

- The AND operation is similar to multiplication:

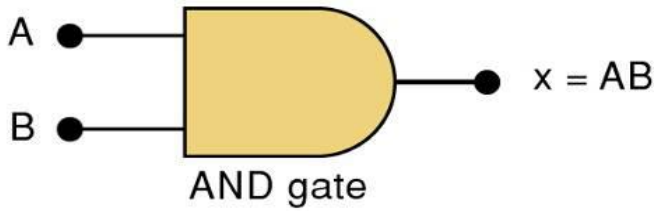
  $X = A \cdot B \cdot C$ — Read as "*X* equals *A* AND *B* AND *C*"

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 · B
0	0	0
0	1	0
1	0	0
1	1	1

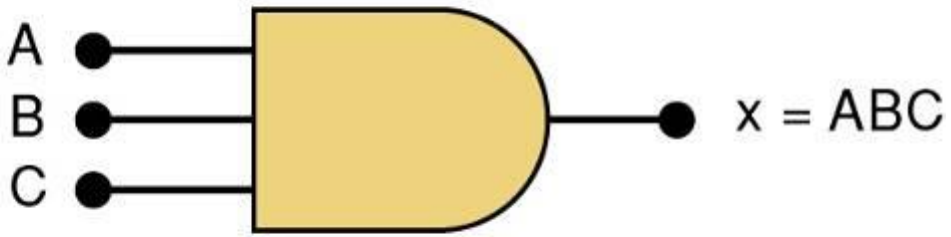


Truth table — Gate symbol.

3-4 AND Operations with AND gates

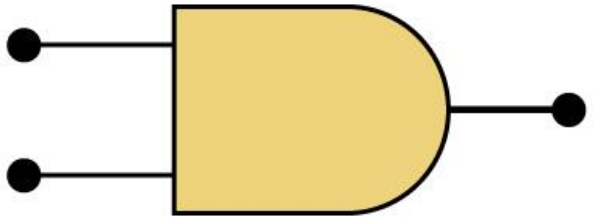
Truth table/circuit symbol for a three input AND gate.

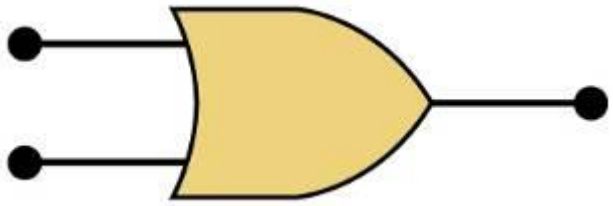
A	B	C	x = ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



AND / OR

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.

3-5 NOT Operation

- The Boolean expression for the NOT operation:

The overbar represents the NOT operation.

— Read as:

$X = \overline{A}$

"*X* equals NOT *A*"

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

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

$A' = \overline{A}$

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

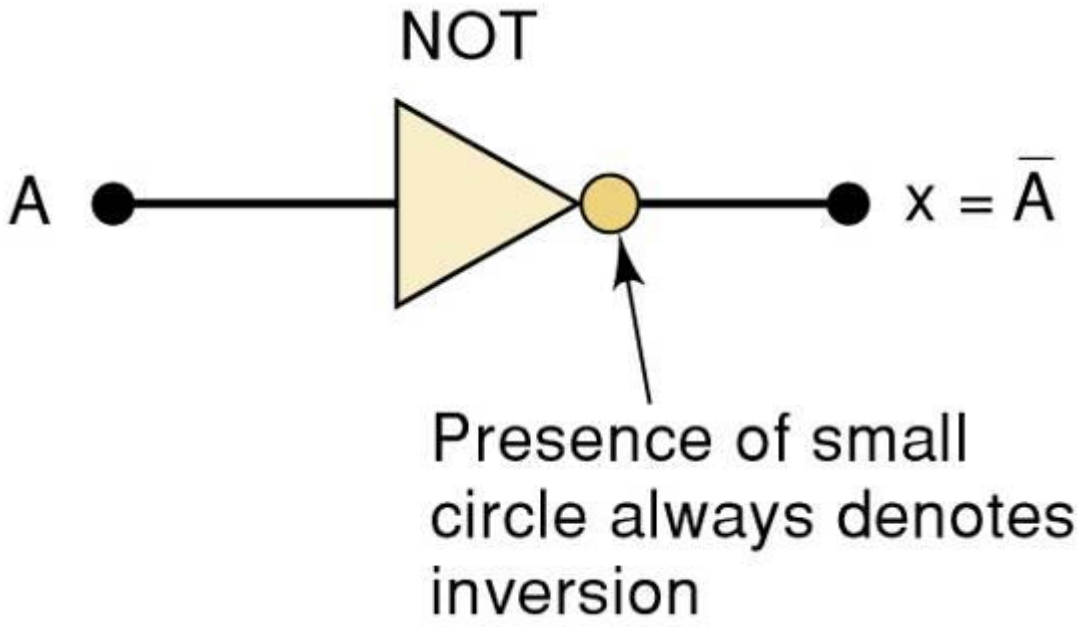
NOT

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

NOT Truth Table

3-5 NOT Operation

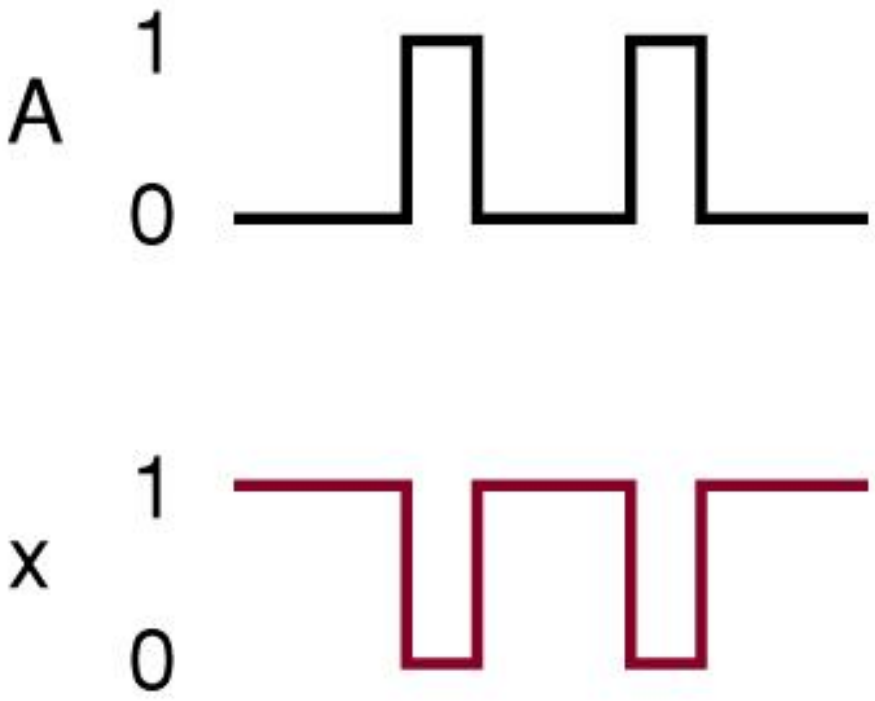
A NOT circuit—commonly called an INVERTER.



This circuit *always* has only a single input, and the out-put logic level is always *opposite* to the logic level of this input.

3-5 NOT Operation

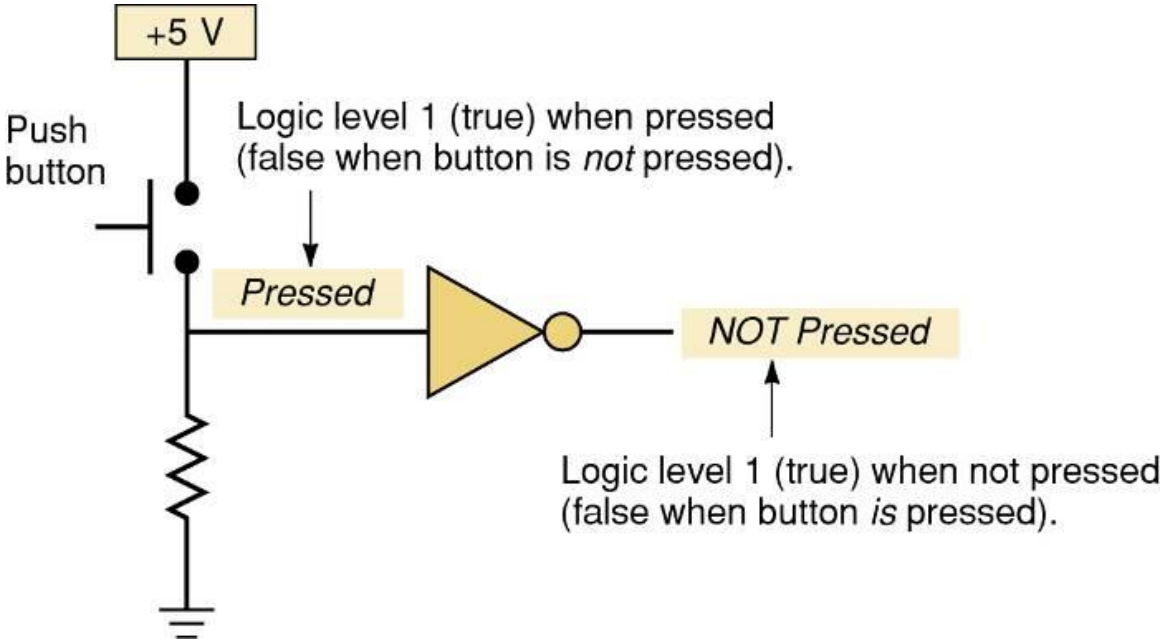
The INVERTER inverts (*complements*) the input signal at all points on the waveform.



Whenever the input = 0, output = 1, and vice versa.

3-5 NOT Operation

- Typical application of the NOT gate.



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

Boolean Operations

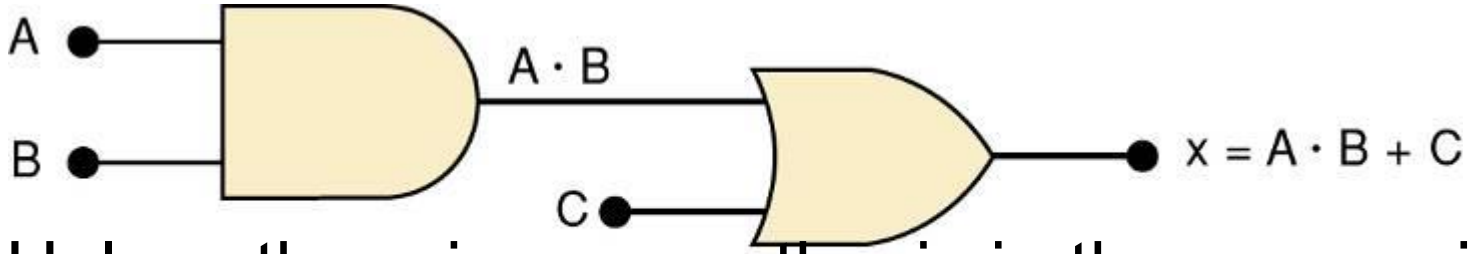
- Summarized rules for OR, AND and NOT

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

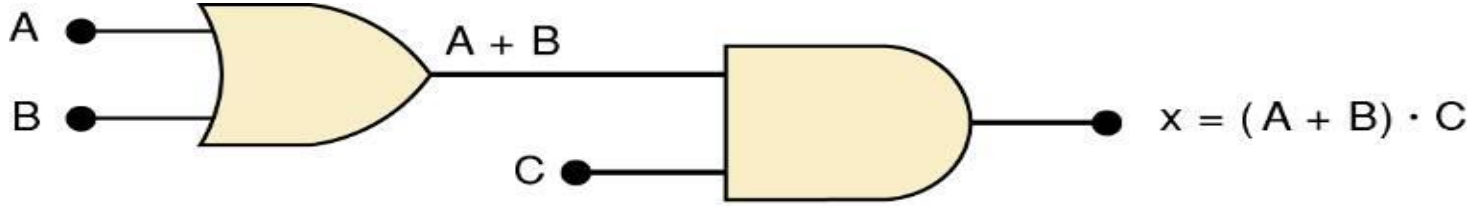
These three basic Boolean operations can describe any logic circuit.

3-6 Describing Logic Circuits Algebraically

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

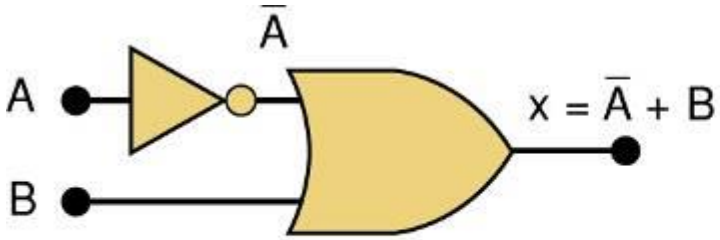


- Unless there is a parenthesis in the expression.

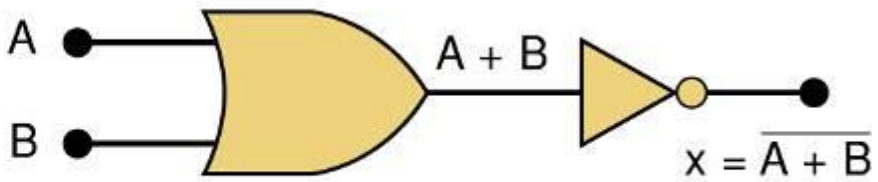


3-6 Describing Logic Circuits Algebraically

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



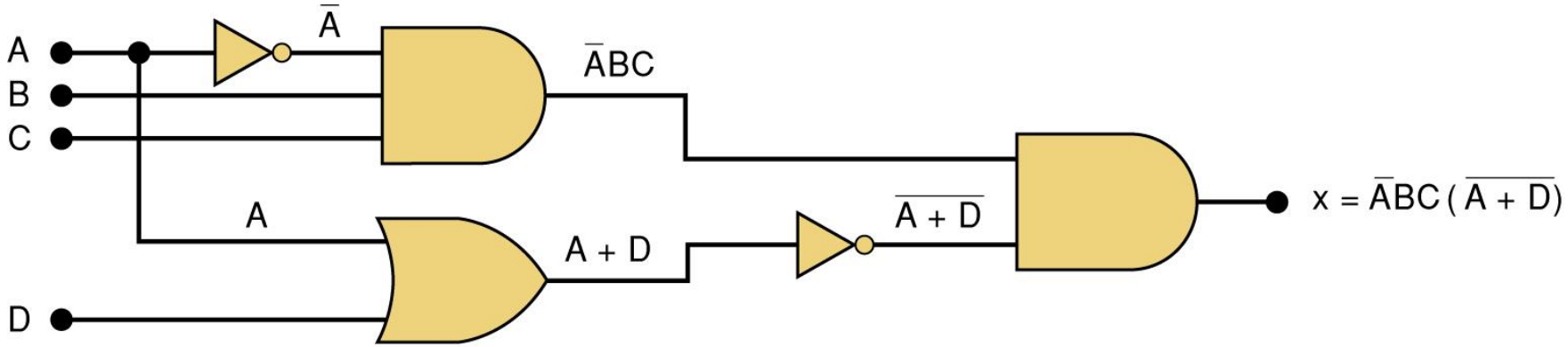
(a)



(b)

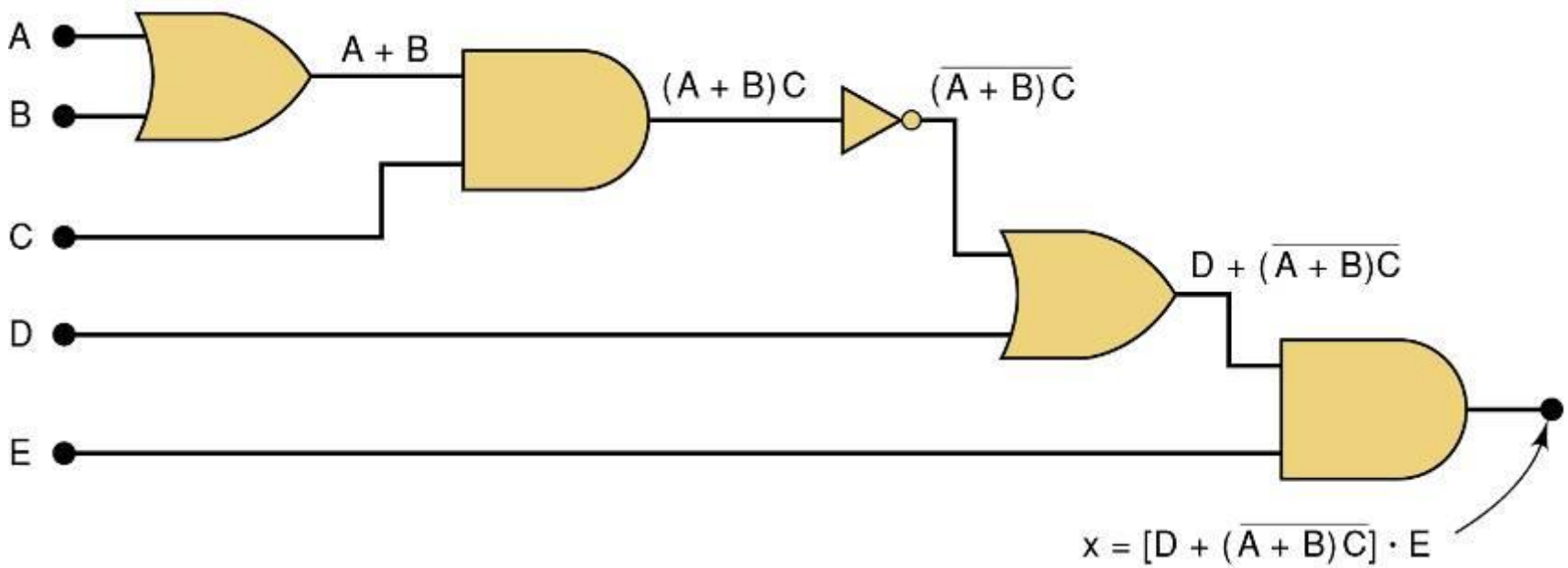
3-6 Describing Logic Circuits Algebraically

- Further examples…



3-6 Describing Logic Circuits Algebraically

- Further examples…



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.

3-7 Evaluating Logic Circuit Outputs

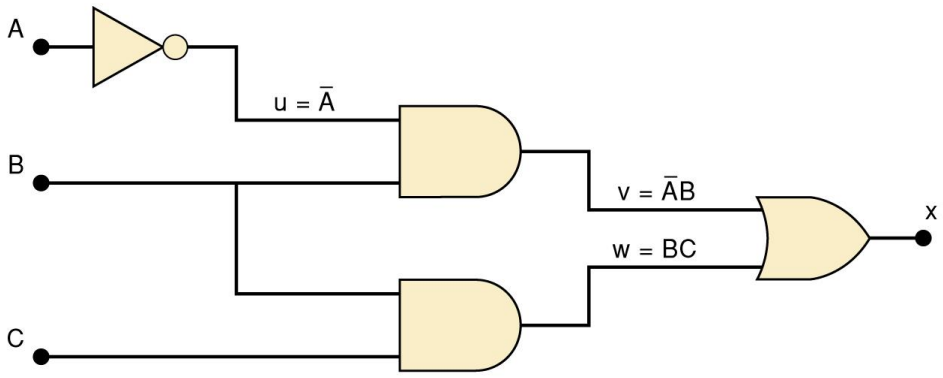
- If an expression has a bar over it, perform operations inside the expression, and then invert the result.

- 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.

3-7 Evaluating Logic Circuit Outputs

- When you are done, you have a table of tremendous benefit in troubleshooting the logic circuit.
- The first step after listing all input combinations is to create a column in the truth table for each intermediate signal (node).

3-7 Evaluating Logic Circuit Outputs

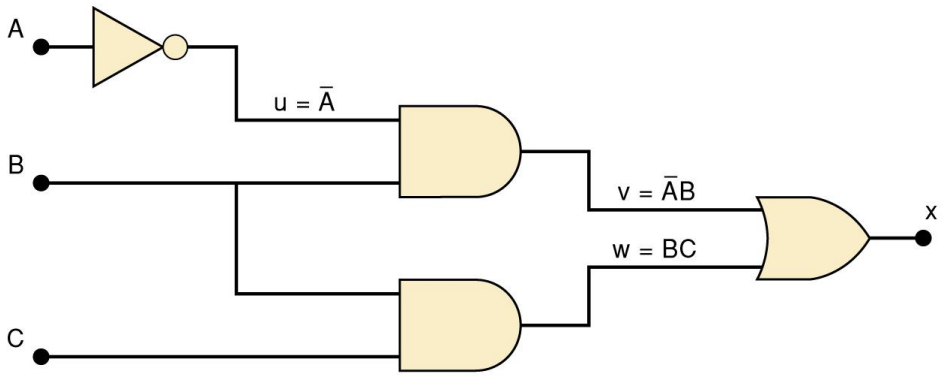


A	B	C	u= Ā	v= ĀB	w= BC	x= v+w
0	0	0	1	0		
0	0	1	1	0		
0	1	0	1	1		
0	1	1	1	1		
1	0	0	0	0		
1	0	1	0	0		
1	1	0	0	0		
1	1	1	0	0		

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

3-7 Evaluating Logic Circuit Outputs

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

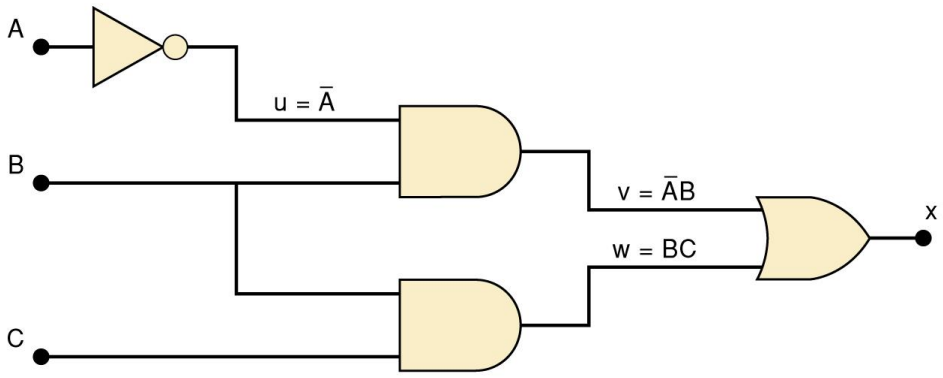


A	B	C	u= Ā	v= ĀB	w= BC	x= v+w
0	0	0	1	0	0	
0	0	1	1	0	0	
0	1	0	1	1	0	
0	1	1	1	1	1	
1	0	0	0	0	0	
1	0	1	0	0	0	
1	1	0	0	0	0	
1	1	1	0	0	1	

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

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= $\bar{A}$	v= $\bar{A}$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

3-7 Evaluating Logic Circuit Outputs

- Output logic levels can be determined directly from a circuit diagram.
  - Output of each gate is noted until final output is found.
    - Technicians frequently use this method.

3-7 Evaluating Logic Circuit Outputs

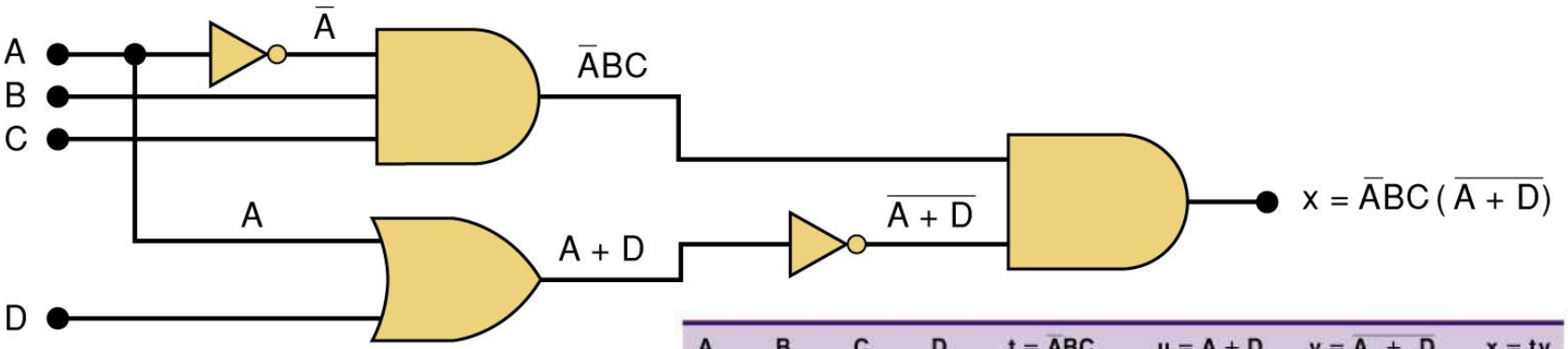


Table of logic state at each node of the circuit shown.

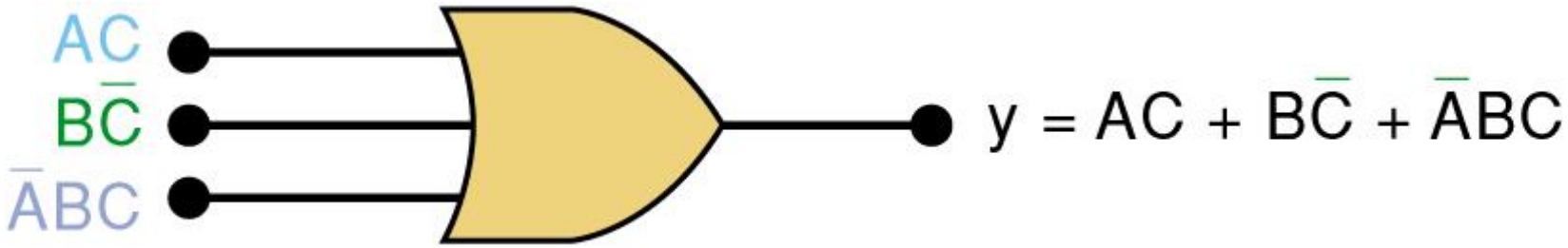
A	B	C	D	$t = \overline{A}BC$	$u = A + D$	$v = \overline{A + D}$	$x = tv$
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	1
0	1	1	1	1	1	0	0
1	0	0	0	0	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	1	0	0
1	1	0	1	0	1	0	0
1	1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

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 • B • C, could be drawn as a three input AND gate.
  - A circuit defined by $X = A + B$, would use a two-input OR gate with an INVERTER on one of the inputs.

3-8 Implementing Circuits From Boolean Expressions

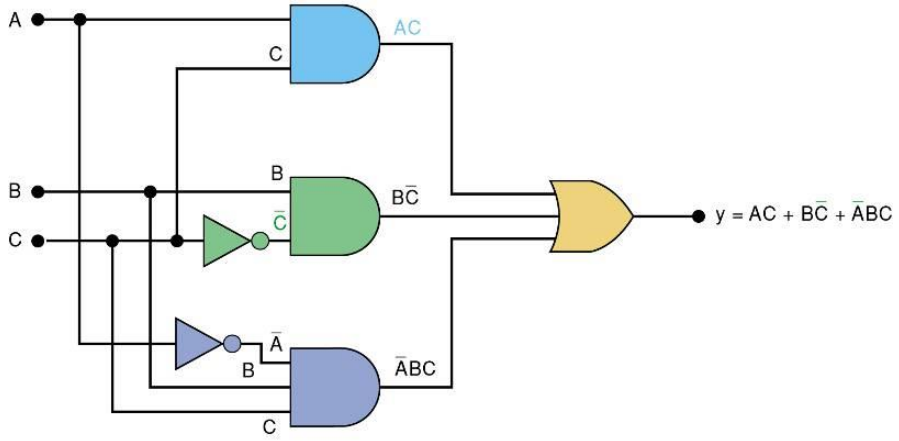
A circuit with output $y = AC + B\overline{C} + \overline{A}BC$ contains three terms which are ORed together.



…and requires a three-input OR gate.

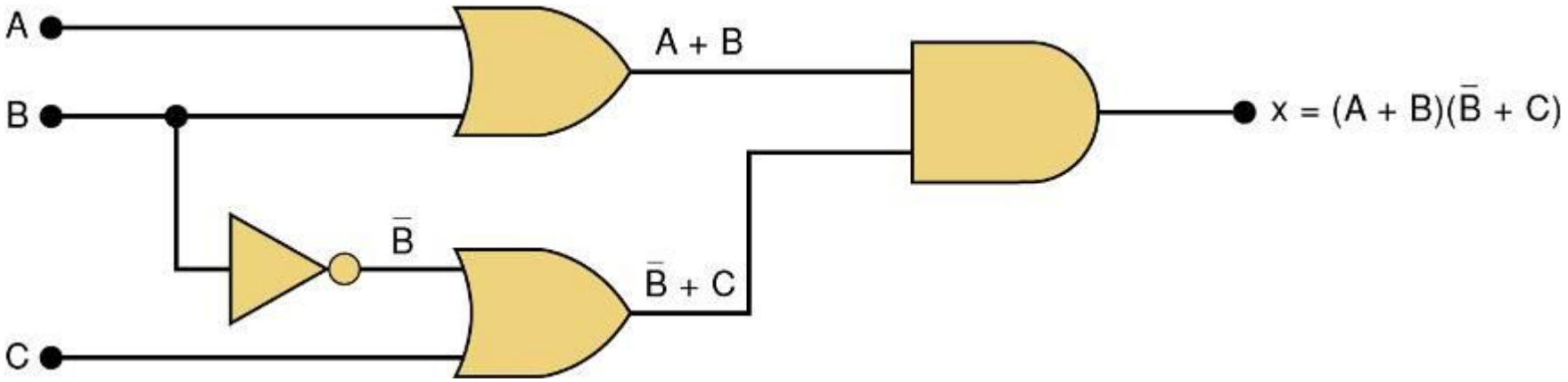
3-8 Implementing Circuits From Boolean Expressions

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



3-8 Implementing Circuits From Boolean Expressions

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

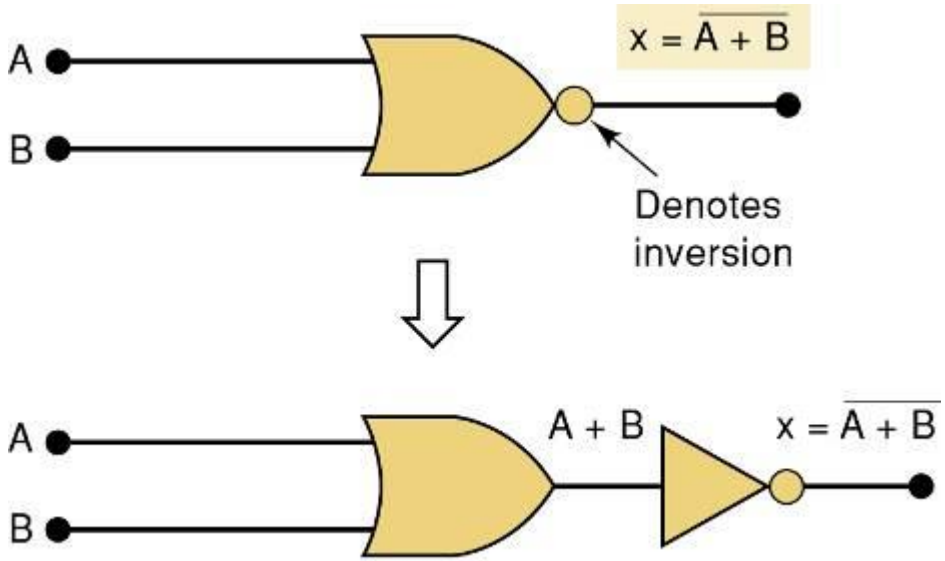


3-9 NOR Gates and NAND Gates

- 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.

3-9 NOR Gates and NAND Gates

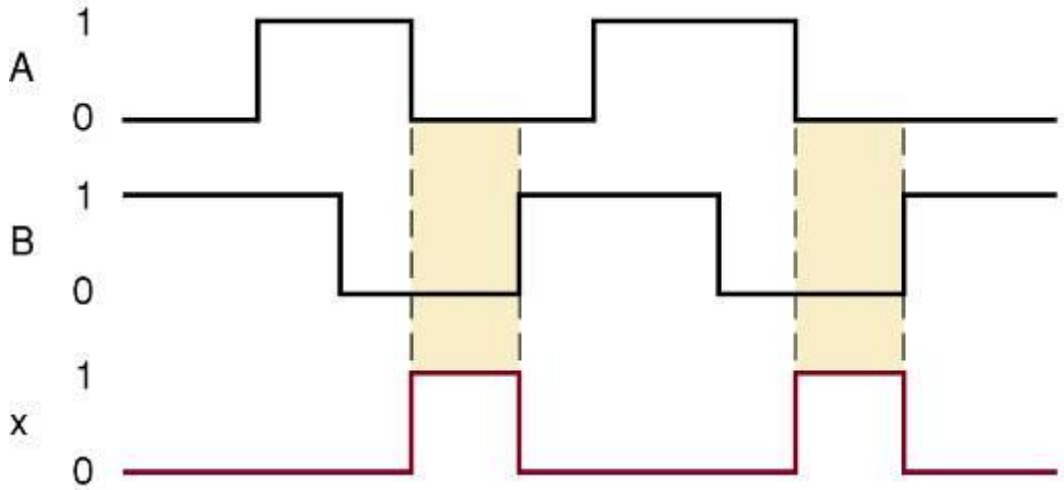
- The NOR gate is an inverted OR gate.
  - An inversion "bubble" is placed at the output of the OR gate, making the Boolean output expression $x = A + B$

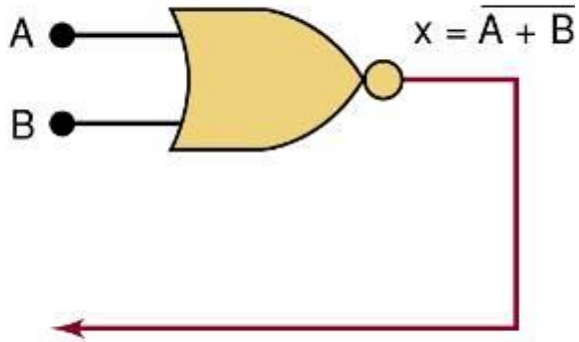


		OR	NOR
A	B	A + B	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

3-9 NOR Gates and NAND Gates

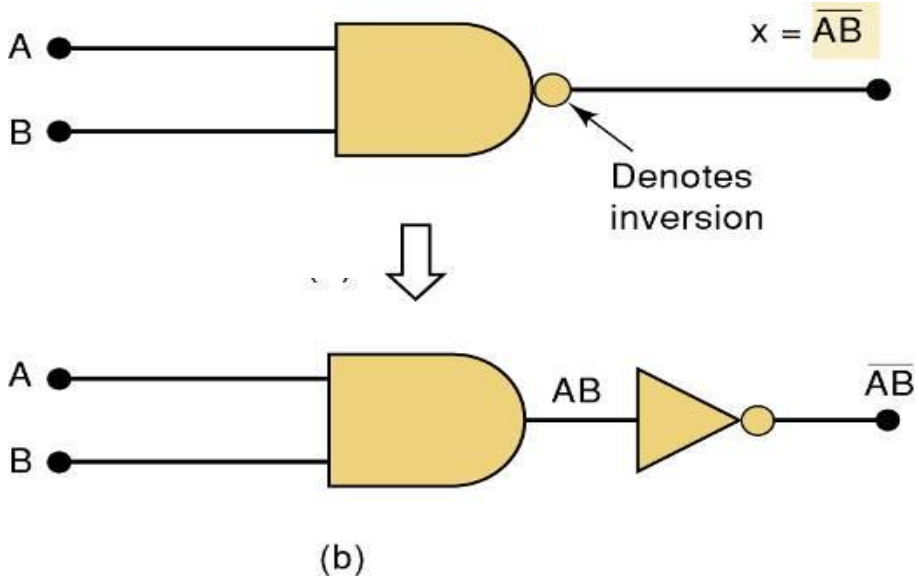
- Output waveform of a NOR gate for the input waveforms shown here.





3-9 NOR Gates and NAND Gates

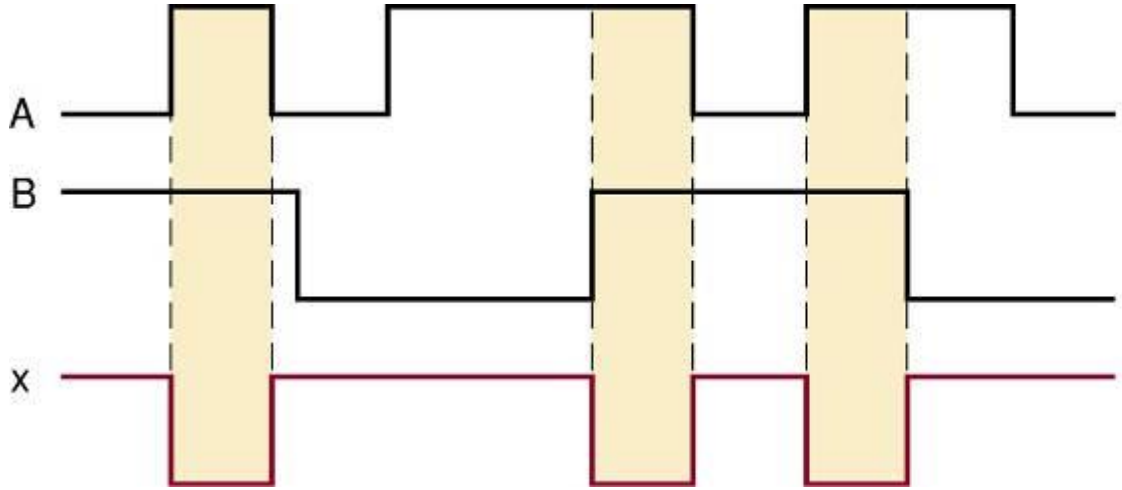
- The NAND gate is an inverted AND gate.
- An inversion "bubble" is placed at the output of the AND gate, making the Boolean output

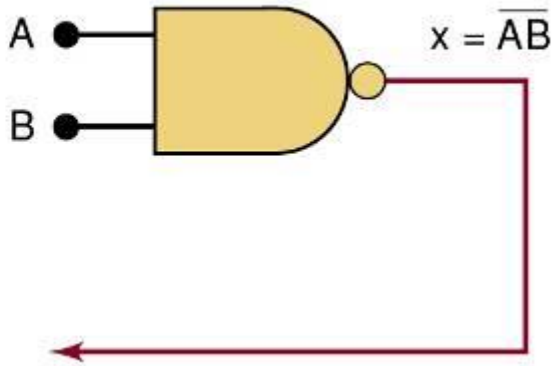


		AND	NAND
A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

3-9 NOR Gates and NAND Gates

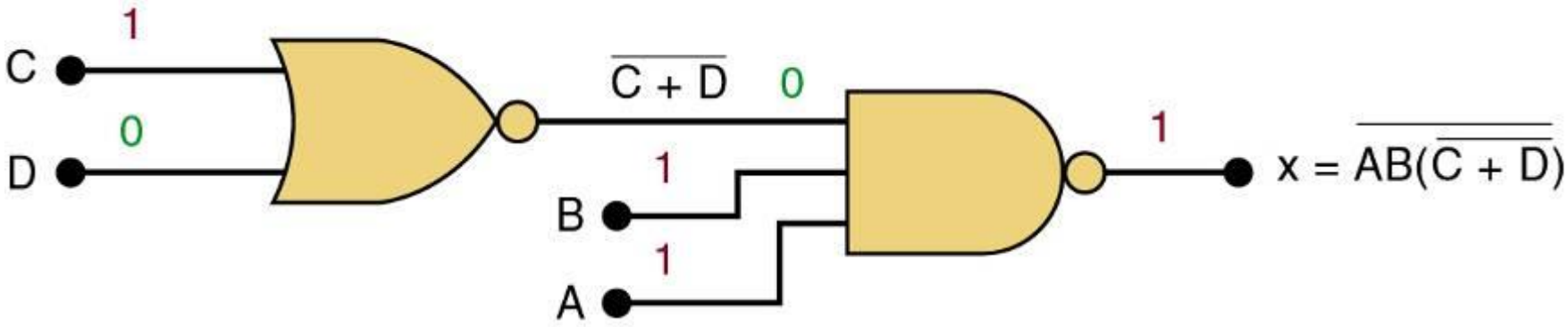
- Output waveform of a NAND gate for the input waveforms shown here.





3-9 NOR Gates and NAND Gates

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



$\overline{AB}$

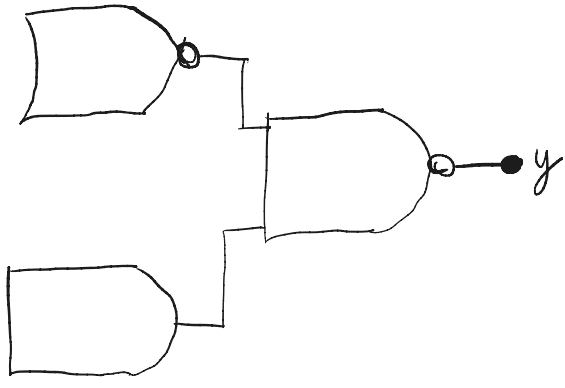
	A	B	C	D	AB	$\overline{C+D}$	$AB \cdot (\overline{C+D})$	$\overline{AB \cdot (\overline{C+D})}$
0	0	0	0	0	0	1	0	1
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	0	1
3	0	1	0	0	0	1	0	1
4	1	0	0	0	0	1	0	1
5	0	0	1	1	0	0	0	1
6	0	1	0	1	0	0	0	1
7	1	0	0	1	0	0	0	1
8	0	1	1	0	0	0	0	1
9	1	0	1	0	0	0	0	1
10	1	1	0	0	1	1	1	0
11	0	1	1	1	0	0	0	1
12	1	0	1	1	0	0	0	1
13	1	1	1	0	1	0	0	1
14	1	1	0	1	1	0	0	1
15	1	1	1	1	1	0	0	1

$F(A,B,C,D)$

$= \Sigma m(0,1,2,3,4,5,6,7,8,9,11,12,13,14,15)$

Dec	A	B	C	D	AB	$\overline{C+D}$	$AB\cdot(C+D)$	$\overline{AB\cdot(\overline{C+D})}$
0	0	0	0	0	0	1	0	1
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	0	1
3	0	0	1	1	0	0	0	1
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	0	0	1
6	0	1	1	0	0	0	0	1
7	0	1	1	1	0	0	0	1
8	1	0	0	0	0	1	0	1
9	1	0	0	1	0	0	0	1
10	1	0	1	0	0	0	0	1
11	1	0	1	1	0	0	0	1
12	1	1	0	0	1	1	1	0
13	1	1	0	1	1	0	0	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	1

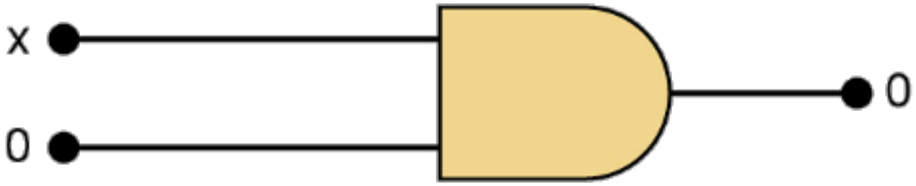
$$\delta(A,B,C,D) = \Sigma_m(0,1,2,3,4,5,6,7,8,9,10,11,13,14,15)$$


y

3-10 Boolean Theorems

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

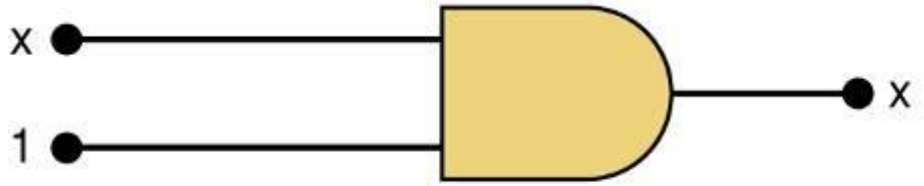
3-10 Boolean Theorems



(1) $x \cdot 0 = 0$

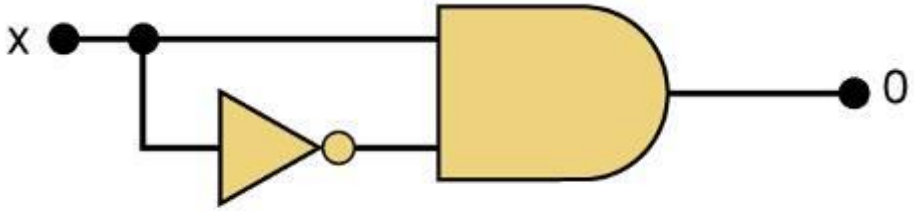
Theorem (1) states that if any variable is ANDed with 0, the result must be 0.

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

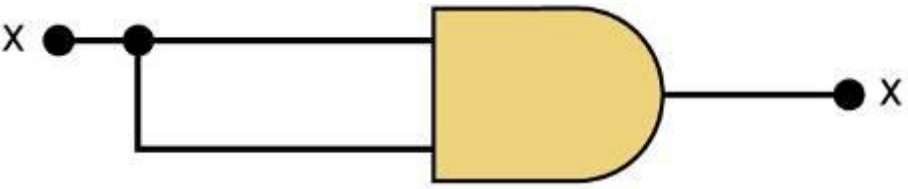


(2) $x \cdot 1 = x$

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



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



(3) $x \cdot x = x$

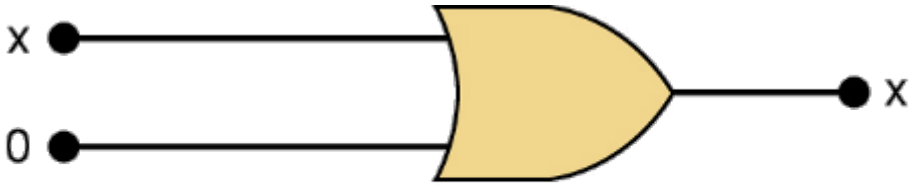
Prove Theorem (3) by trying each case.

If $x = 0$, then $0 \cdot 0 = 0$

If $x = 1$, then $1 \cdot 1 = 1$

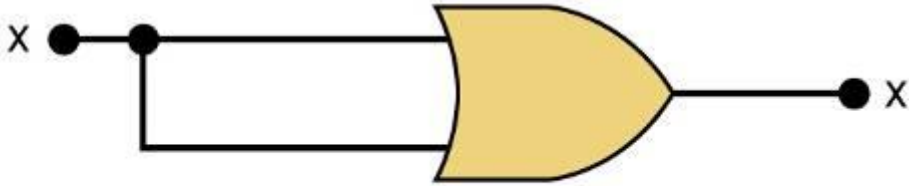
Thus, $x \cdot x = x$

3-10 Boolean Theorems



(5) $x + 0 = x$

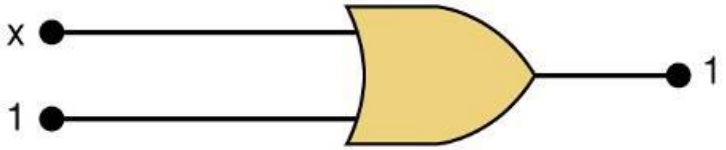
Theorem (6) states that if any variable is ORed with 1, the is always 1.
Check values: 0 + 1 = 1 and 1 + 1 = 1.



(7) $x + x = x$

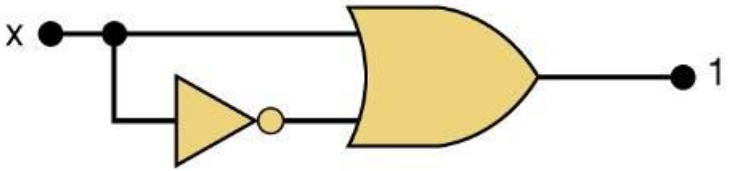
Theorem (8) can be proved similarly.

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



(6) $x + 1 = 1$

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



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

3-10 Boolean Theorems

Multivariable Theorems

Commutative laws

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

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

Associative laws

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

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

Distributive law

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

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

3-10 Boolean Theorems

Multivariable 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*.

Analysis table & factoring for Theorem (14)

$$(14) \quad x + \underline{x}y = x$$

$$(15a) \quad \underline{x} + xy = \underline{x} + y$$

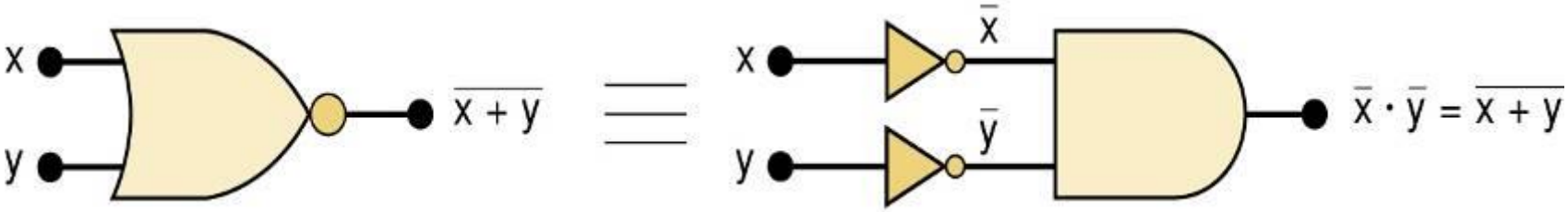
$$(15b) \quad \underline{x} + xy = x + y$$

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

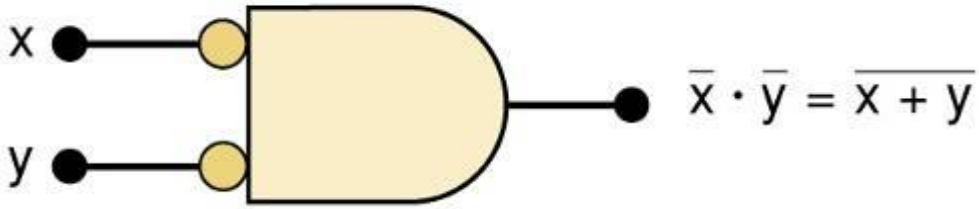
x	y	xy	x + xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

3-11 DeMorgan's Theorems

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



The alternative symbol for the NOR function.

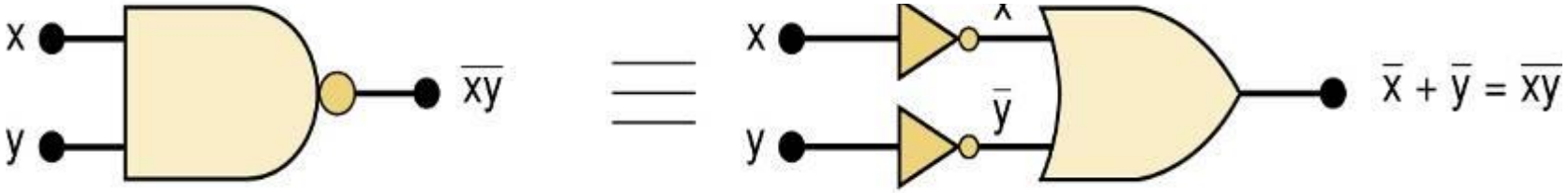


3-11 DeMorgan's Theorems

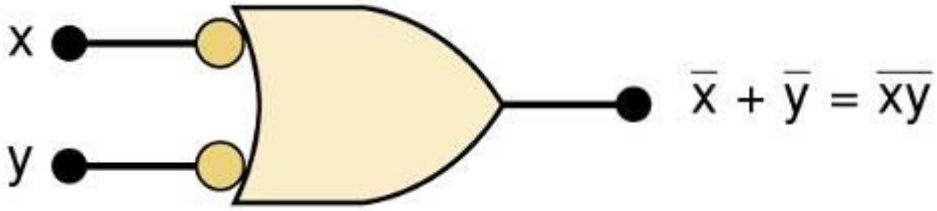
Equivalent circuits implied by Theorem (17)

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

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



The alternative symbol for the NAND function.

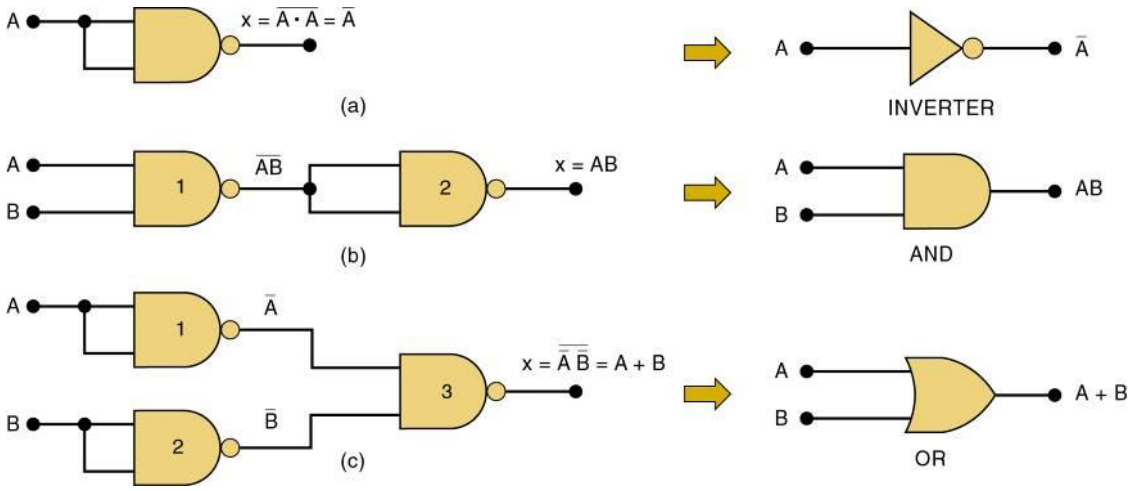


3-12 Universality of NAND and NOR Gates

- 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.

3-12 Universality of NAND and NOR Gates

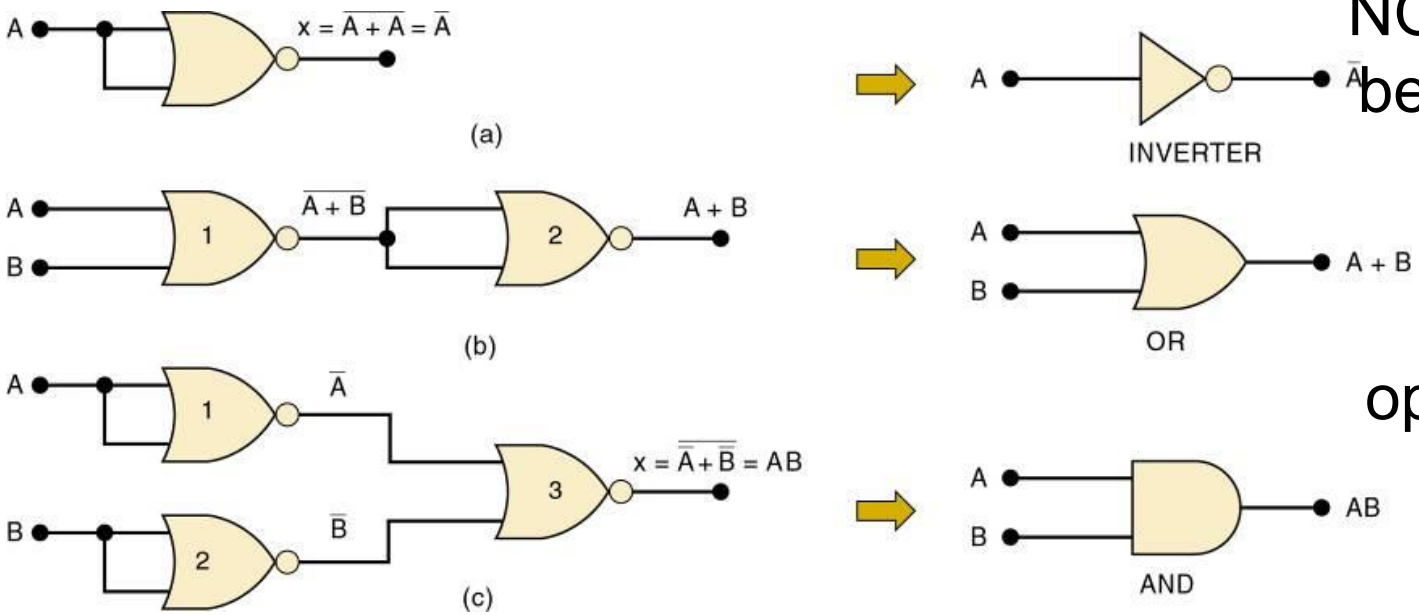
- How combinations of NANDs or NORs are used to create the three logic functions.



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

3-12 Universality of NAND and NOR Gates

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



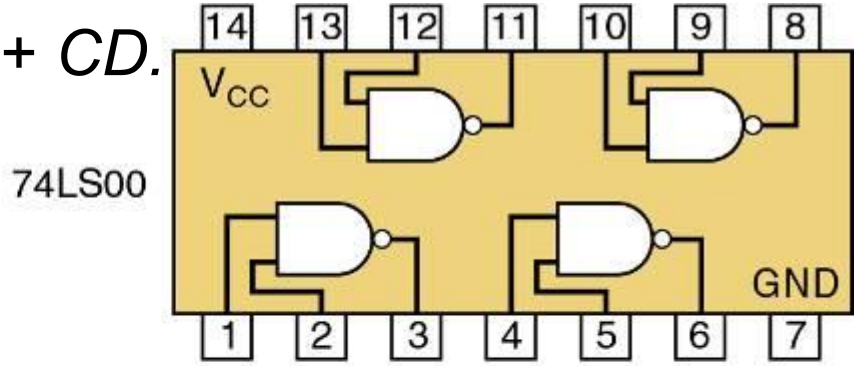
NOR gates can be arranged to implement any of the Boolean operations, as shown.

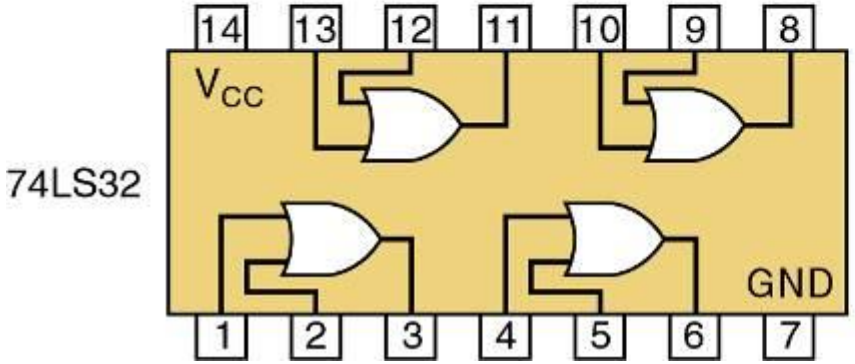
3-12 Universality of NAND and NOR Gates

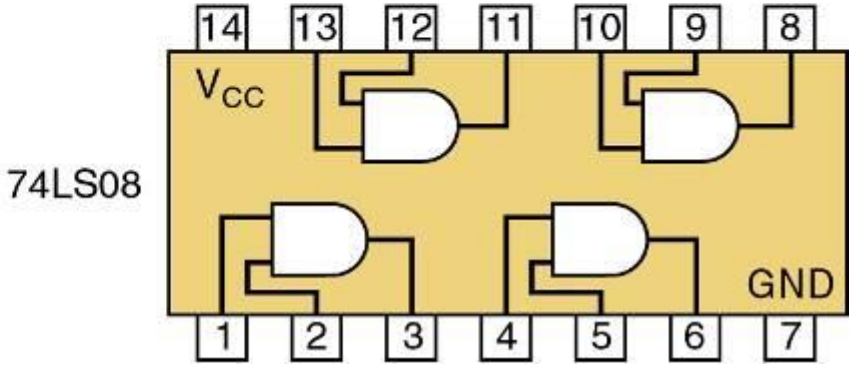
- A logic circuit to generate a signal x, that will go HIGH whenever conditions

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

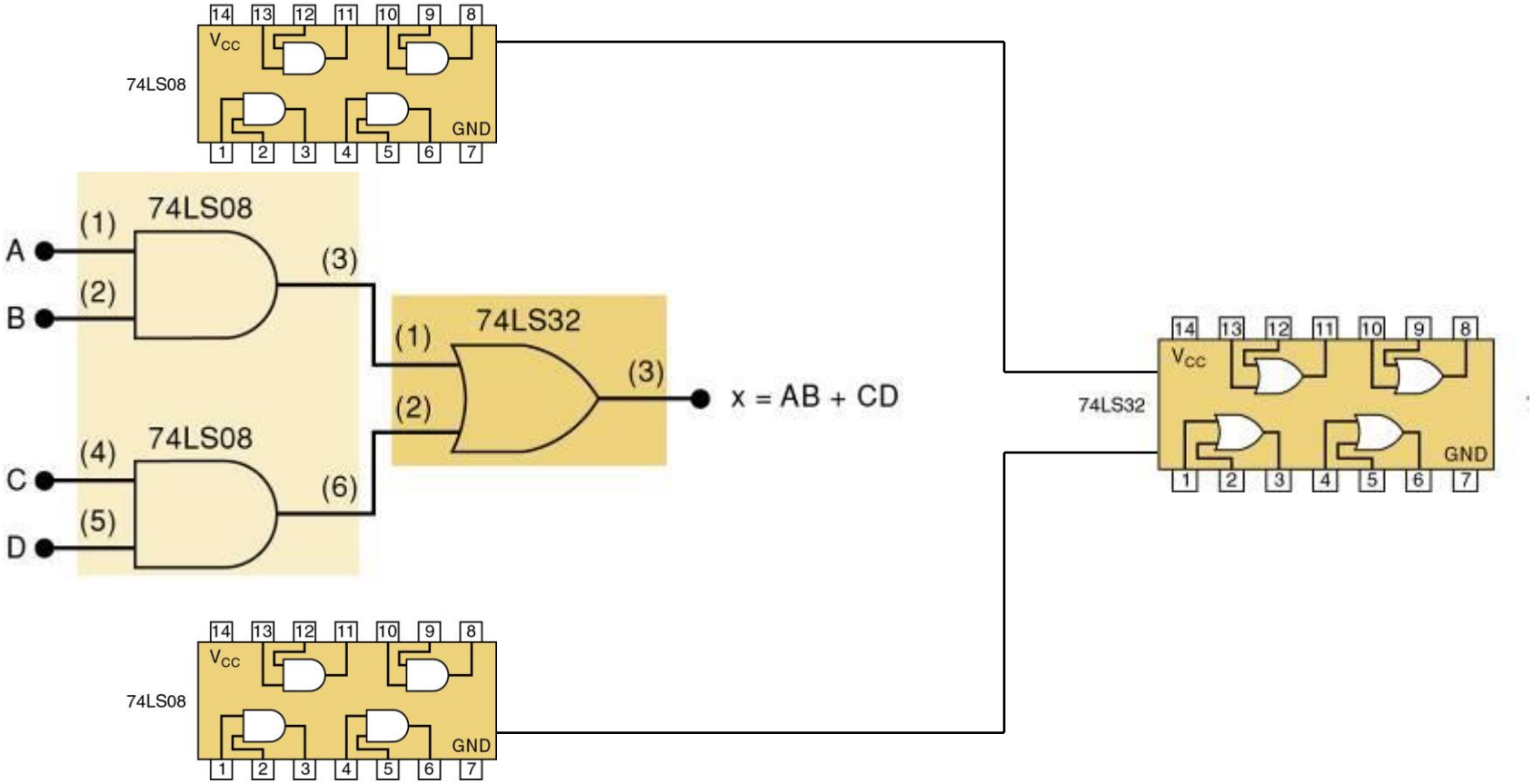






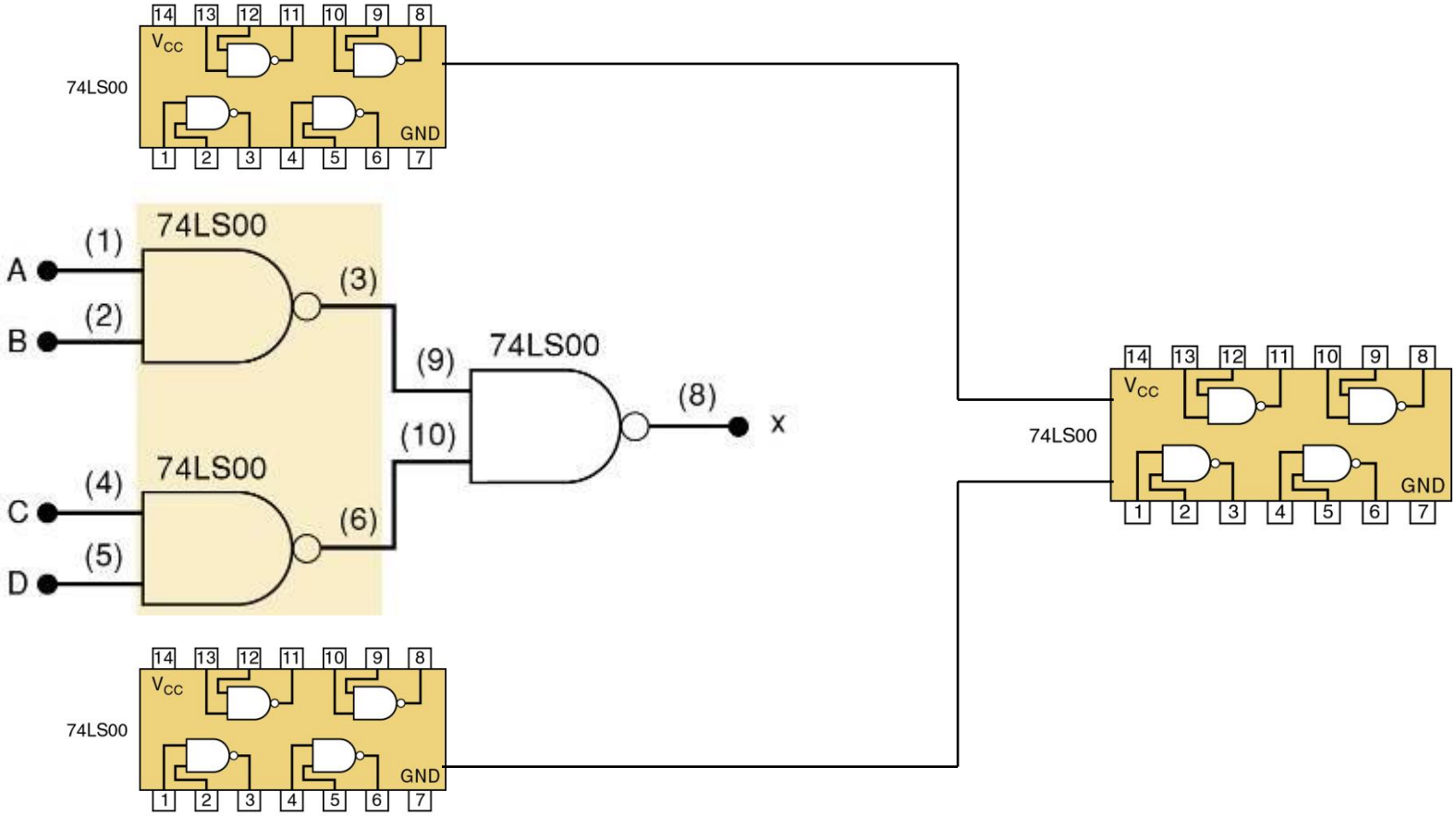
3-12 Universality of NAND and NOR Gates

- Possible Implementations # 1



3-12 Universality of NAND and NOR Gates

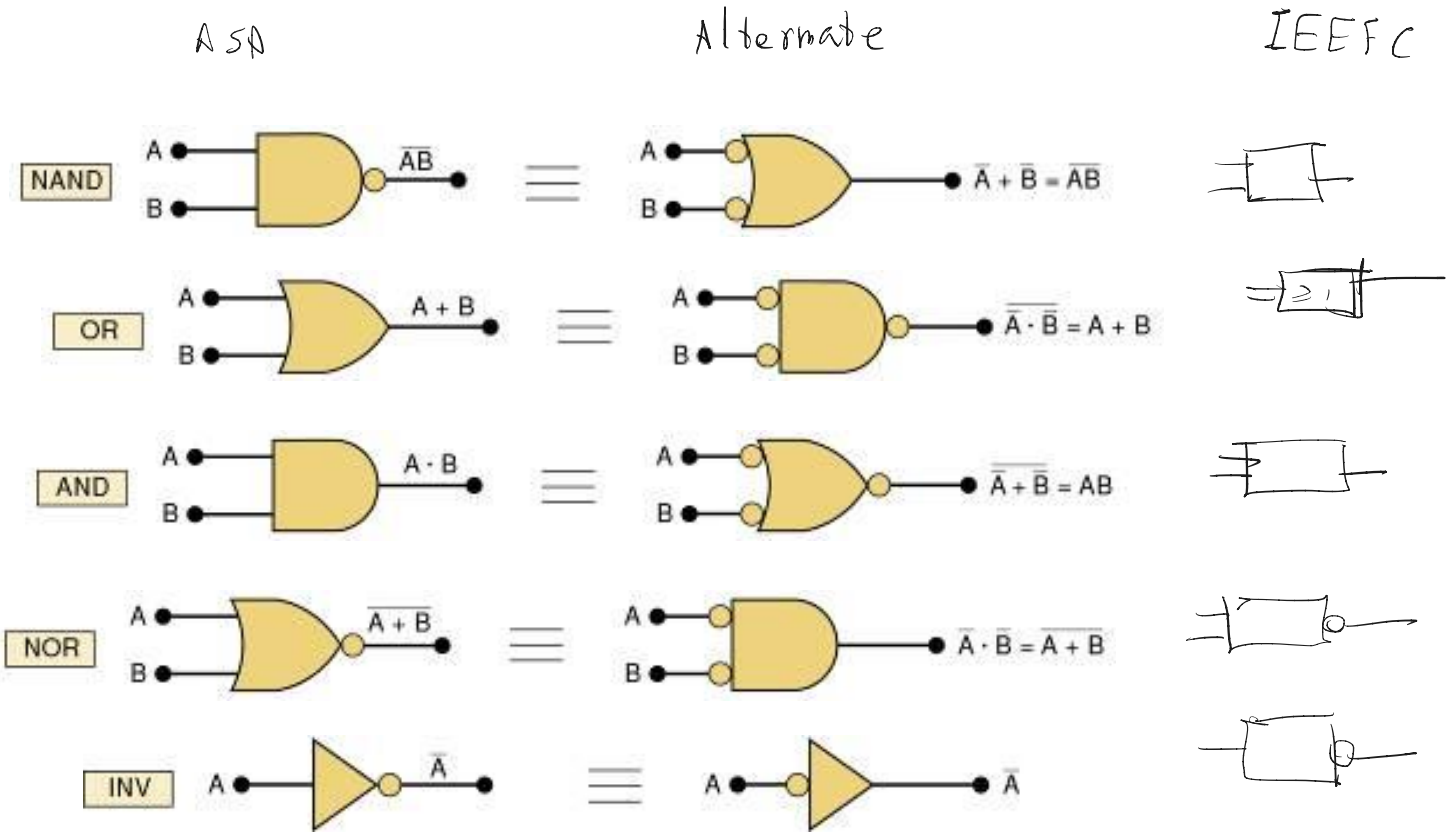
- Possible Implementations #2



3-13 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.

3-13 Alternate Logic-Gate Representations



3-13 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.

3-13 Alternate Logic-Gate Representations

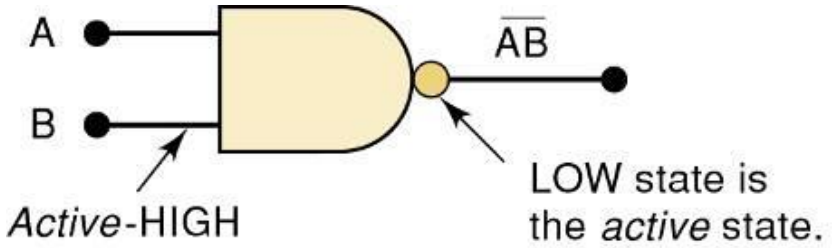
- Standard & alternate symbols for each gate represent the same physical circuit.
- NAND and NOR gates are inverting gates.
  - Both the standard and the alternate symbols for each will have a bubble on either the input or the output.
- AND and OR gates are noninverting gates.

3-13 Alternate Logic-Gate Representations

- The alternate symbols for each will have bubbles
  on both inputs and output.
- Active-HIGH – an input/output has no inversion bubble.
- Active-LOW – an input or output has an inversion bubble.

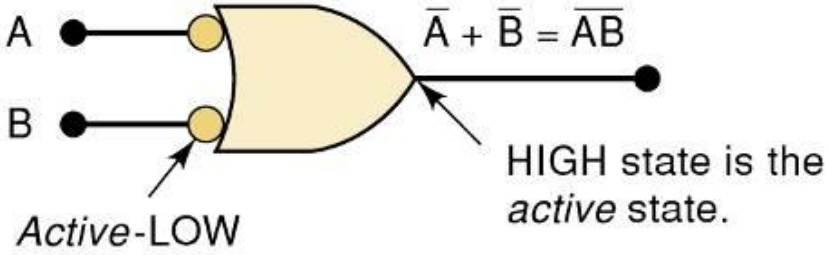
3-13 Alternate Logic-Gate Representations

- Interpretation of the two NAND gate symbols.



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

(a)

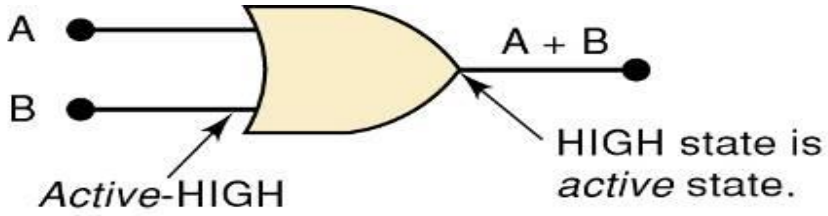


Output is HIGH when *any* input is LOW.

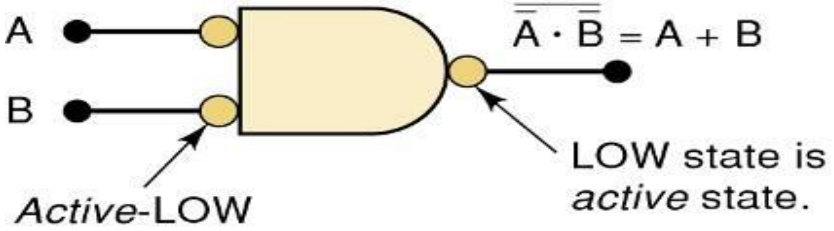
(b)

3-13 Alternate Logic-Gate Representations

- Interpretation of the two OR gate symbols.



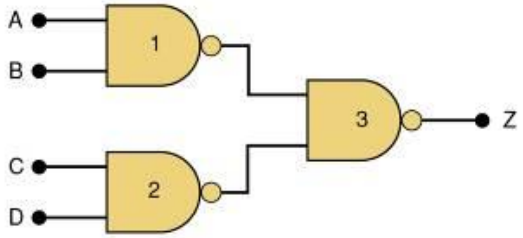
Output goes HIGH when *any* input is HIGH.



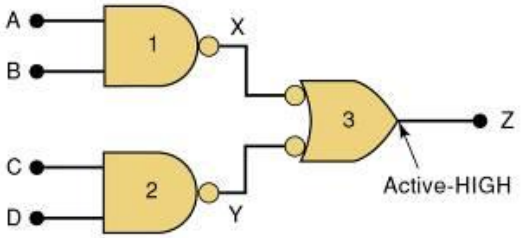
Output goes LOW only when *all* inputs are LOW.

3-14 Which Gate Representation to Use

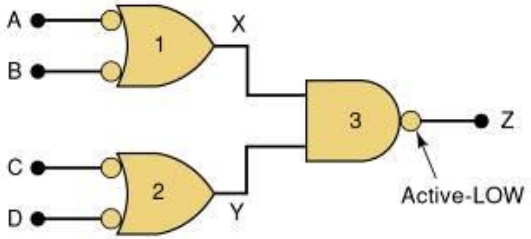
Proper use of alternate gate symbols in the circuit diagram can make circuit operation much clearer.



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

3-14 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

3-14 Which Gate Representation to Use

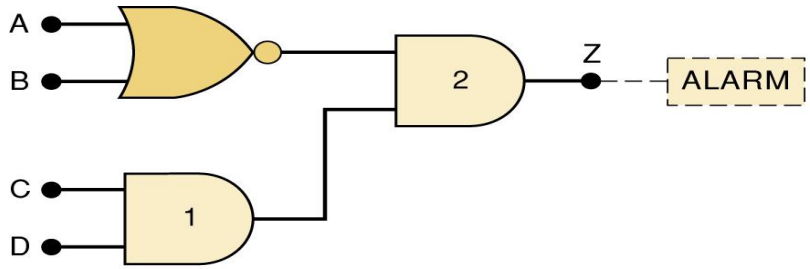
- An output signal can have two active states, with an important function in the HIGH state, and another in the LOW state.
  - It is customary to label such signals so both
    active states are apparent.

3-14 Which Gate Representation to Use

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

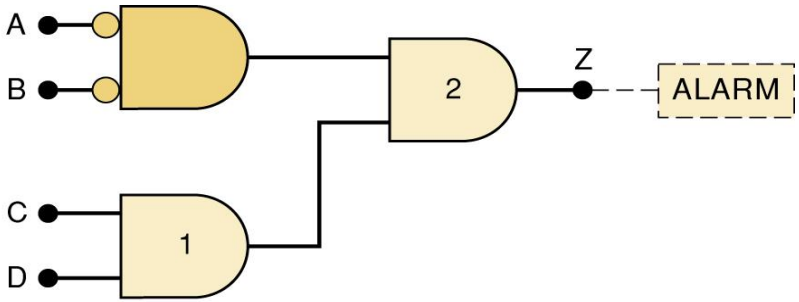
3-14 Which Gate Representation to Use

- The logic circuit shown activates an alarm when output Z goes HIGH.



Modify the circuit diagram so it represents the circuit operation more effectively.

The NOR gate symbol should be changed to the alternate symbol with a nonbubble (active-HIGH) output to match the nonbubble input of AND gate 2.



(b)

The circuit now has nonbubble outputs connected to nonbubble inputs of gate 2.

3-15 Propogation 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.

3-16 Summary of Methods To Describe Logic Circuits

- We must be able to represent these logical decisions.
- We must be able to combine these logic functions and implement a decision-making system.

3-16 Summary of Methods To Describe Logic Circuits

- We have learned how to represent each of the basic logic functions using:
  - Logical statements in our own language
  - Truth tables
  - Traditional graphic logic symbols
  - Boolean algebra expressions
  - Timing diagrams

3-17 Description vs. Programming Languages

- HDL – Hardware Description Languages allow rigidly defined language to represent logic circuits.
  - AHDL – Altera Hardware Description Language.
    - Developed by Altera to configure Altera Programmable Logic Devices (PLDs).
    - Not intended to be used as a universal language for describing any logic circuit.

3-17 Description vs. Programming Languages

- VHDL – Very High Speed Integrated circuit Hardware Description Language.
  - Developed by U.S. Department of Defense (DoD).
  - Standardized by IEEE.
  - Widely used to translate designs into bit patterns that program actual devices.
- It is important to distinguish between hardware description languages & programming languages

3-17 Description vs. Programming Languages

- In both, a language is used to program a device.

- Computers operate by following a list of tasks, each of which must be done in sequential order.
  - Speed of operation is determined by how fast the computer can execute each instruction.

3-17 Description vs. Programming Languages

- A digital logic circuit is limited in speed only by how quickly the circuitry can change outputs in response to changes in the inputs.
  - It is monitoring all in-puts concurrently & responding to any changes.

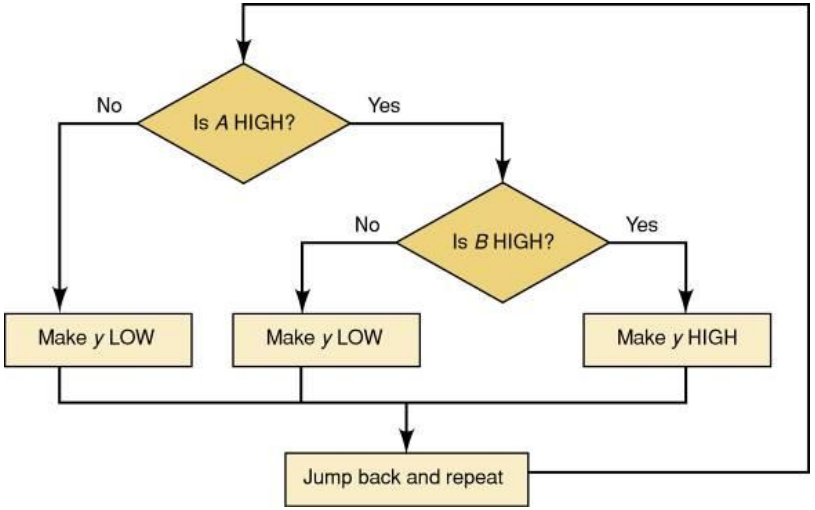
3-17 Description vs. Programming Languages

- Comparing operation of a computer and a logic circuit in performing the logical operation of y = AB.

Each shape in the flowchart represents one instruction.

If each takes 20 ns, it will take a minimum of two or three instructions (40–60 ns) to respond to changes in the inputs.

The computer must run a program of instructions that makes decisions.



3-18 Implementing Logic Circuits With PLDs

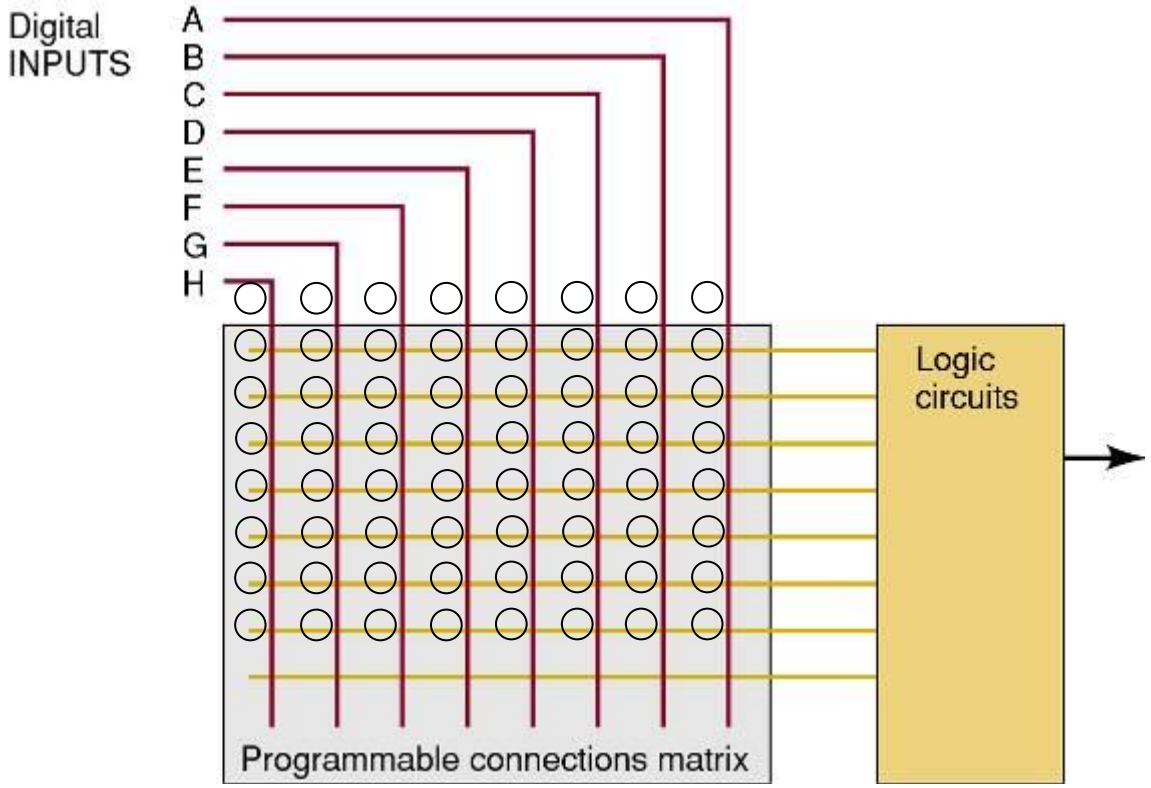
- Programmable Logic Devices (PLDs) are devices that can be configured in many ways to perform logic functions.
  - Internal connections are made electronically to program devices.

3-18 Implementing Logic Circuits With PLDs

PLDs are configured electronically & their internal circuits are "wired" together electronically to form a logic circuit.

This programmable wiring can be thought of as thousands of connections, either connected (1), or not connected (0).

Each intersection of a row (horizontal wire) & column (vertical wire) is a programmable connection.



3-18 Implementing Logic Circuits With PLDs

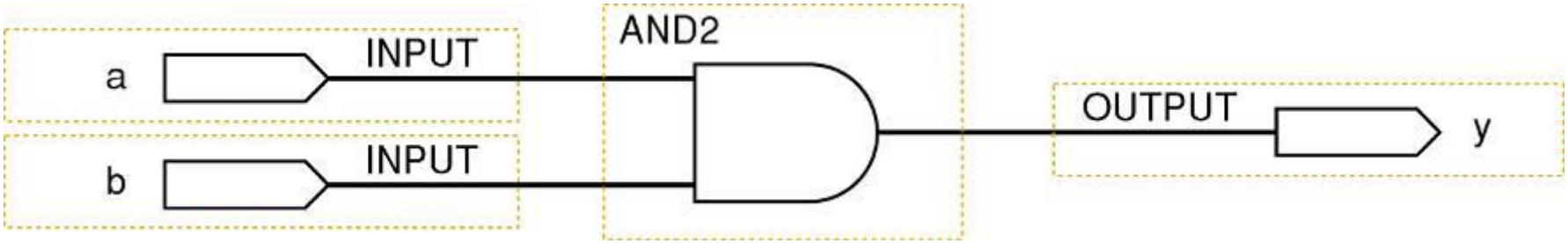
- The hardware description language defines the connections to be made.
  - It is loaded into the device after translation by a compiler.
- The higher-level hardware description language, makes programming the PLDs much easier than trying to use Boolean algebra, schematic drawings, or truth tables.

3-19 HDL Format and Syntax

- Languages that are interpreted by computers must follows strict rules of syntax.
  - Syntax refers to the order of elements.

3-19 HDL Format and Syntax

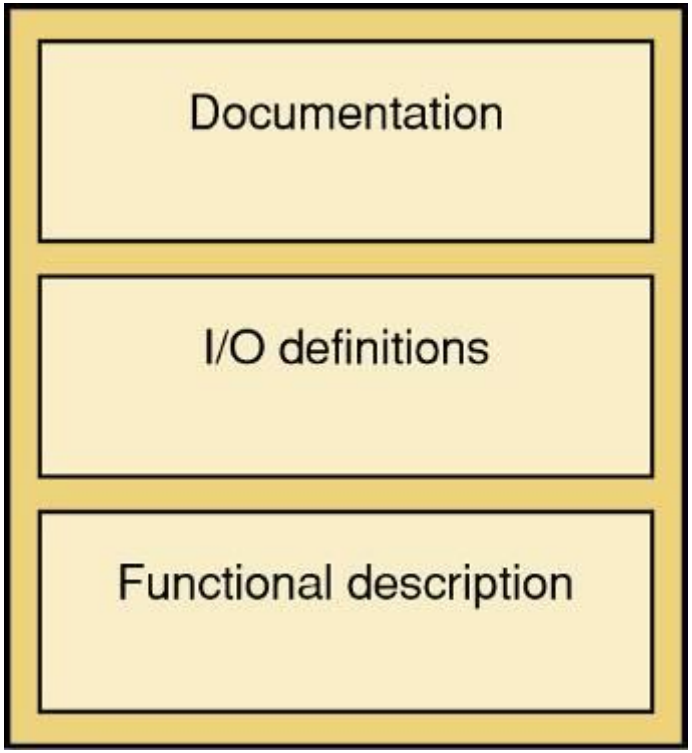
- On the left side of the diagram is the set of inputs, and on the right is the set of outputs.



3-19 HDL Format and Syntax

- Format refers to a definition of inputs, outputs & how the output responds to the input (operation).

Format of HDL files.



3-19 HDL Format and Syntax

- In a text-based language, the circuit described must be given a name.
  - The definition of the operation is contained in a set of statements that follow the input/output (I/O) definition.
- Inputs & outputs (ports) must be assigned names and defined according to the nature of the port.
  - The mode defines whether it is input, output, or both.

3-19 HDL Format and Syntax

- The type refers to the number of bits and how those bits are grouped and interpreted.
  - A single bit input, can have only two values: 0 and 1.
  - A four-bit binary number can have any one of 16 different values (00002 - 11112).

3-19 HDL Format and Syntax - AHDL

- The keyword SUBDESIGN gives a name to the circuit block, which in this case is and gate. The name of the file must also be and_gate.tdf.

```
SUBDESIGN and_gate
(
    a, b        :INPUT;
    y           :OUTPUT;
)
BEGIN
    y = a & b;
END;
```

Variables for inputs are separated by commas & followed by :INPUT;

In AHDL, input/output definition is enclosed in parentheses.

In AHDL, the single-bit type is assumed unless the variable is designated as multiple bits.

Single-output bit is declared with the mode :OUTPUT;

3-19 HDL Format and Syntax - AHDL

- The keyword SUBDESIGN gives a name to the circuit block, which in this case is and_gate. The name of the file must also be and_gate.tdf.

```
SUBDESIGN and_gate
(
    a, b        :INPUT;
    y           :OUTPUT;
)
BEGIN
    y = a & b;
END;
```

Statements describing operation of the AHDL circuit are contained in the logic section between the keywords BEGIN and END. END must be followed by a semicolon.

Statements between BEGIN and END are evaluated constantly and concurrently.

The order in which they are listed makes no difference.

3-19 HDL Format and Syntax - VHDL

- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase

```
ENTITY and_gate IS
PORT (   a, b    :IN BIT;
         y       :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
        y <= a AND b;
END ckt;
```

The keyword PORT tells the compiler that we are defining inputs and outputs to this circuit block.

3-19 HDL Format and Syntax - VHDL

- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase.

```
ENTITY and_gate IS
PORT (   a, b    :IN BIT;
         y       :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
       y <= a AND b;
END ckt;
```

The BIT description tells the compiler that each variable in the list is a single bit.

3-19 HDL Format and Syntax - VHDL

- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase.

```
ENTITY and_gate IS
PORT (   a, b    :IN BIT;
         y       :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
       y <= a AND b;
END ckt;
```

The ARCHITECTURE declaration is used to describe the operation of everything inside the block.
Every ENTITY must have at least one ARCHITECTURE associated with it.

3-19 HDL Format and Syntax - VHDL

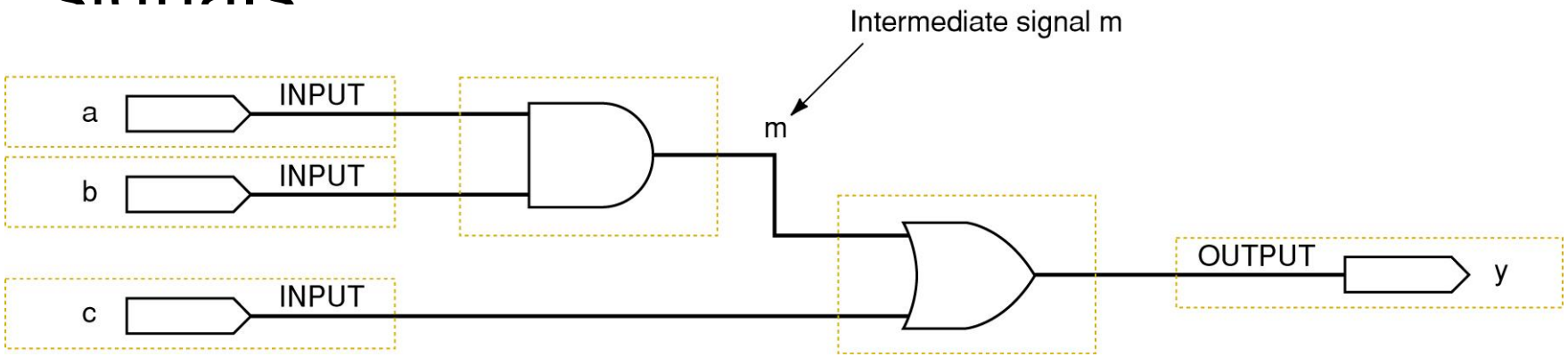
- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase.

```
ENTITY and_gate IS
PORT (    a, b    :IN BIT;
          y       :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
        y <= a AND b;
END ckt;
```

Within the body (between BEGIN and END) is the description of the block's operation.

3-20 Intermediate Signals

- In many designs, there is a need to define signal points “inside” the circuit block—called buried nodes or local signals

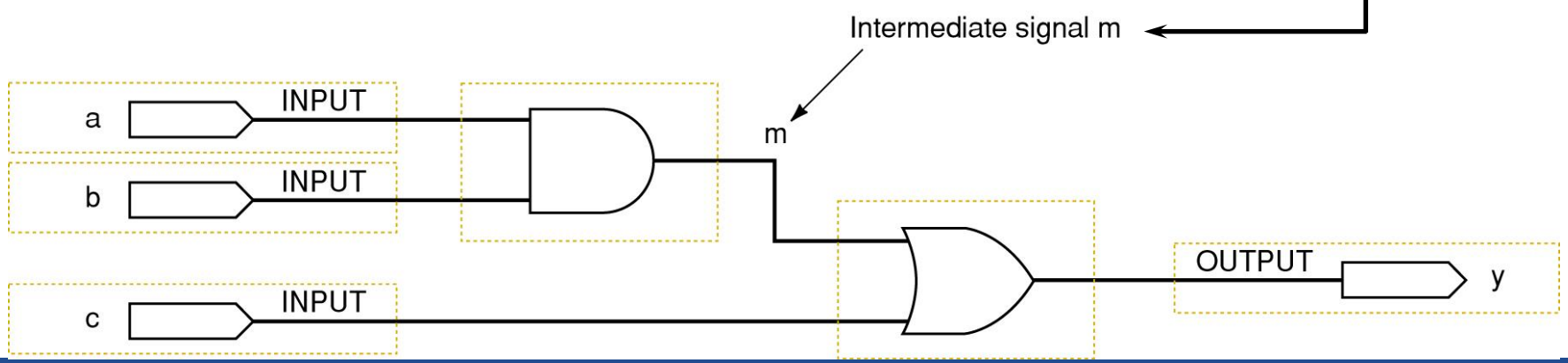


3-20 Intermediate Signals - AHDL

- AHDL local signals:
  - Comments are enclosed by % characters.
  - Text after two dashes is for documentation only.
  - Keyword VARIABLE defines intermediate signal.
  - Keyword NODE designates the nature of the variable.

3-20 Intermediate Signals - AHDL

```
1   %  Intermediate variables in AHDL (Figure 3-49)
2      Digital Systems 11th ed
3      NS Widmer
4      MAY 24, 2010          %
5   SUBDESIGN fig3_50
6   (
7      a,b,c      :INPUT;    -- define inputs to block
8      y          :OUTPUT;   -- define block output
9   )
10  VARIABLE
11     m       :NODE;        -- name an intermediate signal
12  BEGIN
13     m = a & b;            -- generate buried product term
14     y = m # c;            -- generate sum on output
15  END;
```

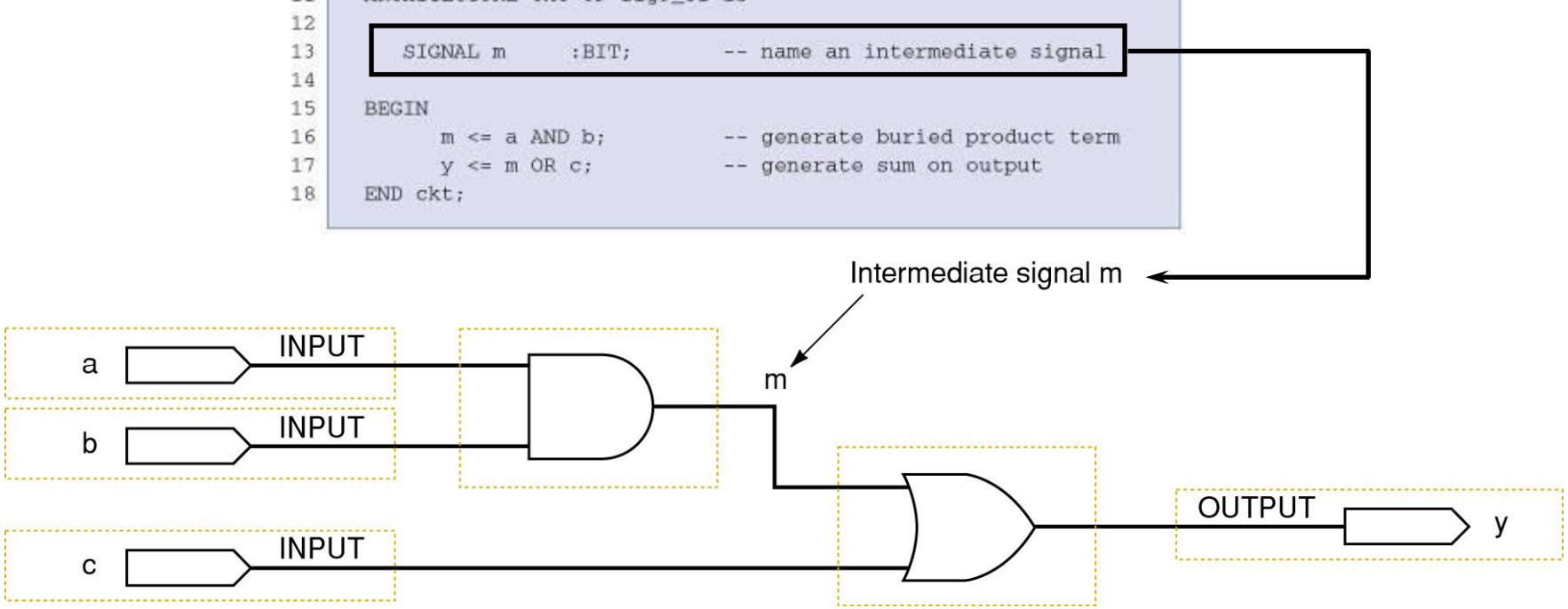


3-20 Intermediate Signals - VHDL

- VHDL local signals:
  - Text after two dashes is for documentation only.
  - Keyword SIGNAL defines intermediate signal.
  - Keyword BIT designates the type of signal

3-20 Intermediate Signals - VHDL

```
3   -- NS Widmer
4   -- MAY 24, 2010
5
6   ENTITY  fig3_51 IS
7   PORT( a, b, c    :IN BIT;     -- define inputs to block
8   y                :OUT BIT);   -- define block output
9   END fig3_51;
10
11  ARCHITECTURE ckt OF fig3_51 IS
12
13    SIGNAL m      :BIT;         -- name an intermediate signal
14
15  BEGIN
16        m <= a AND b;           -- generate buried product term
17        y <= m OR c;            -- generate sum on output
18  END ckt;
```



$ac'd' + a'cd'$ $\quad \bar{x} + xy = \bar{x} + y$

(3)

Dec	I/P A B C D	O/P F E	G
0	0 0 0 0	0 0	0
1	0 0 0 1	0 1	0
2	0 0 1 0	1 0	0
3	0 0 1 1	1 1	0
4	0 1 0 0	0 1	0
5	0 1 0 1	1 0	0
6	0 1 1 0	1 1	0
→ 7	0 1 1 1	0 0	1
8	1 0 0 0	1 0	0
9	1 0 0 1	1 1	0
10	1 0 1 0	0 0	1
11	1 0 1 1	0 1	1
12	1 1 0 0	1 1	0
13	1 1 0 1	0 0	1
14	1 1 1 0	0 1	1
15	1 1 1 1	1 0	1

(4) Minimum (5) Across (6) Best

$F(A,B,C,D) = \Sigma m(2,3,5,6,8,9,12,15)$

$E(A,B,C,D) = \Sigma m(1,3,4,6,9,11,12,14)$

$G(A,B,C,D) = \Sigma m(7,10,11,13,14,15)$ (0111, 1010, 1011, 1101, 1110, 1111)

$= \bar{A}BCD + AB\bar{C}D + ABC\bar{D} + ABCD$

$= \bar{A}BCD + ABC(\bar{D}+D) + AB\bar{C}D$

$= \bar{A}BCD + AB\bar{C}D + AC(\bar{B}+B)$

$= \bar{A}BCD + A(B\bar{C}D + C) = \bar{A}BCD + A(BD + C)$

$= \bar{A}BCD + ABD + AC = BD(\bar{A}C + A) + AC = BD(C + A) + AC$

$= BD(C+A) + AC$

$= BCD + ABD + AC$

$E(A,B,C,D) = \bar{A}\bar{B}CD + A\bar{B}CD + AB\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + AB\bar{C}\bar{D} + AB\bar{C}D$

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

$= \bar{B}CD + B\bar{C}D + B\bar{C}\bar{D} + B\bar{C}\bar{D}$

$= \bar{B}D + B\bar{D}$

$F(A,B,C,D) = \bar{A}\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + AB\bar{C}\bar{D} + AB\bar{C}D$

$= \bar{A}(\bar{B}C\bar{D} + \bar{B}CD + B\bar{C}D + BC\bar{D}) + A(\bar{B}C\bar{D} + B\bar{C}\bar{D} + BC\bar{D} + BCD)$

$= \bar{A}(\bar{B}C(\bar{D}+D) + B\bar{C}D + BC\bar{D}) + A(\bar{B}C + B\bar{C}\bar{D} + BCD)$

$= \bar{A}BC + A\bar{B}C\bar{D} + AB\bar{C}D + A\bar{B}C + AB\bar{C}\bar{D} + A\bar{B}CD + AB\bar{C}\bar{D}$

$\bar{A}BCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + AB\bar{C}D + AB\bar{C}\bar{D} + ABC\bar{D} + ABCD$

$BCD + AC\bar{D} + A\bar{B}CD + AB\bar{C}D$

$A\bar{B}C$

$\bar{A}BCD + A(\bar{B}C\bar{D} + \bar{B}CD + B\bar{C}D + BC\bar{D} + BCD)$

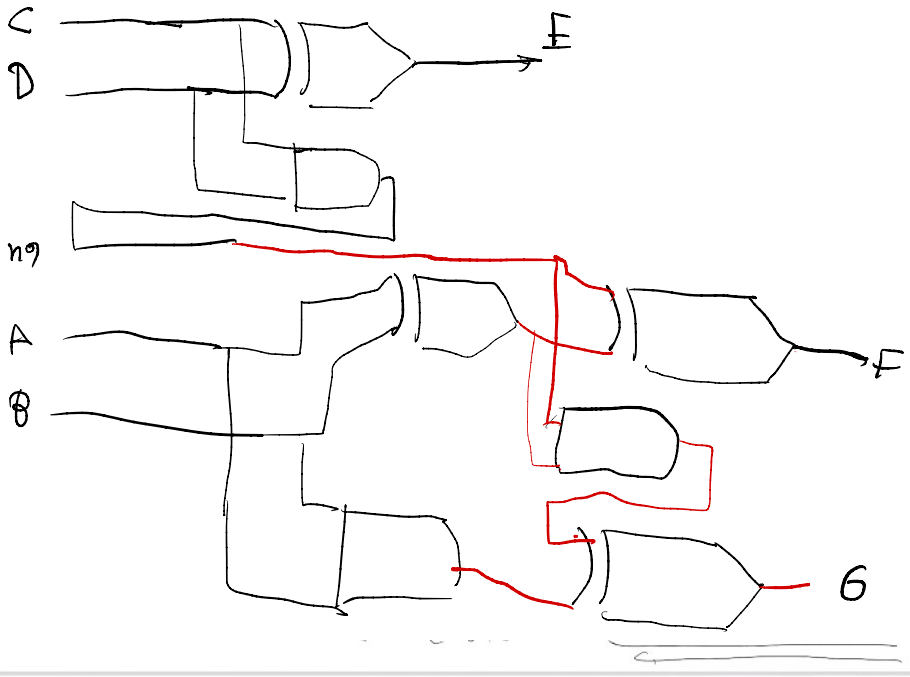
$\bar{A}BCD + A(BD + C\bar{D} + \bar{B}CD)$

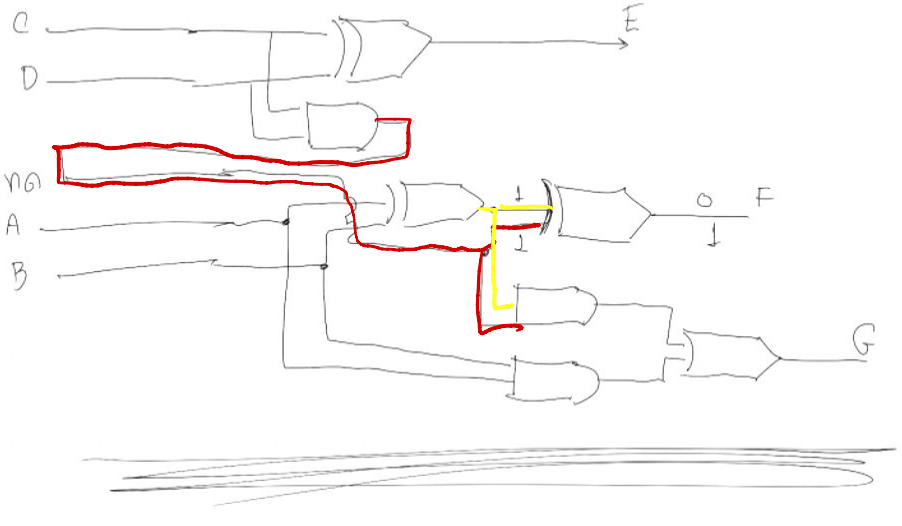
$\bar{A}BCD + ABD + AC\bar{D} + \bar{B}CD$

$CD(A\bar{B} + \bar{B}) + ABD + AC\bar{D}$

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

$\bar{B}CD + \bar{A}CD + ABD + AC\bar{D}$


C
D
E
ng
A
B
F
G


C
D
E
ng
A
B
1
1
0
1
F
G