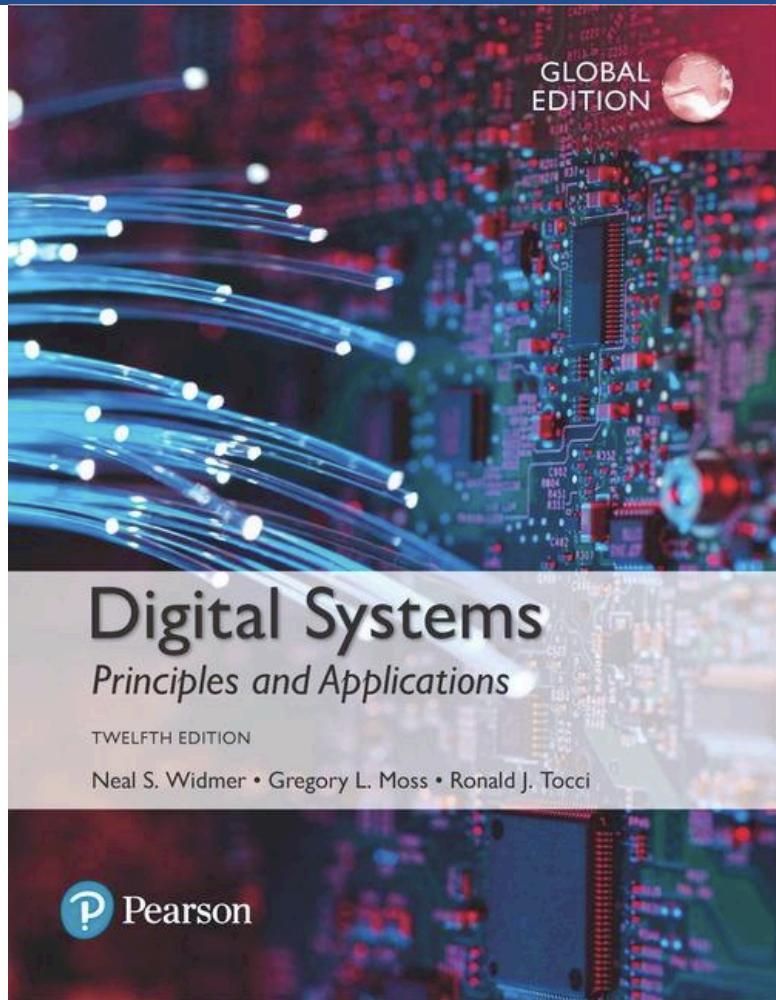


Digital Systems Principles and Applications

TWELFTH EDITION, GLOBAL EDITION



CHAPTER 4

Combinational Logic Circuits

Chapter 4 Objectives

- Convert a logic expression into a sum-of-products expression.
- Perform the necessary steps to reduce a sum-of-products expression to its simplest form.
- Use Boolean algebra and the Karnaugh map as tools to simplify and design logic circuits.

Chapter 4 Objectives

- Explain the operation of both exclusive-OR and exclusive-NOR circuits.
- Design simple logic circuits without the help of a truth table.
- Describe how to implement enable circuits.
- Cite the basic characteristics of TTL and CMOS digital ICs.

Chapter 4 Objectives

- Use the basic troubleshooting rules of digital systems.
- Deduce from observed results the faults of malfunctioning combinational logic circuits.
- Describe the fundamental idea of programmable logic devices (PLDs).

Chapter 4 Objectives

- Describe the steps involved in programming a PLD to perform a simple combinational logic function.
- Describe hierarchical design methods.
- Identify proper data types for single-bit, bit array, and numeric value variables.

Chapter 4 Objectives

- Describe logic circuits using HDL control structures IF/ELSE, IF/ELSIF, and CASE.
- Select the appropriate HDL control structure for a given problem.

4-1 Sum-of-Products Form

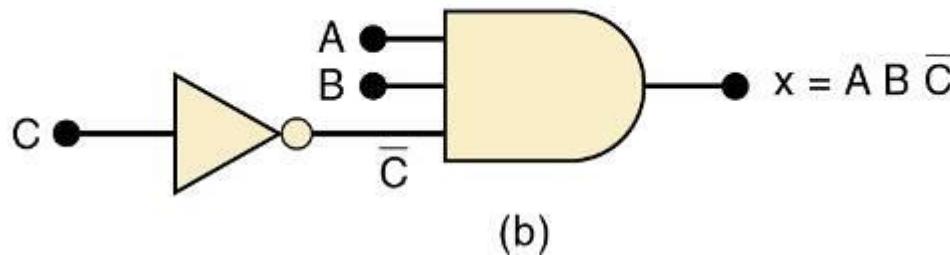
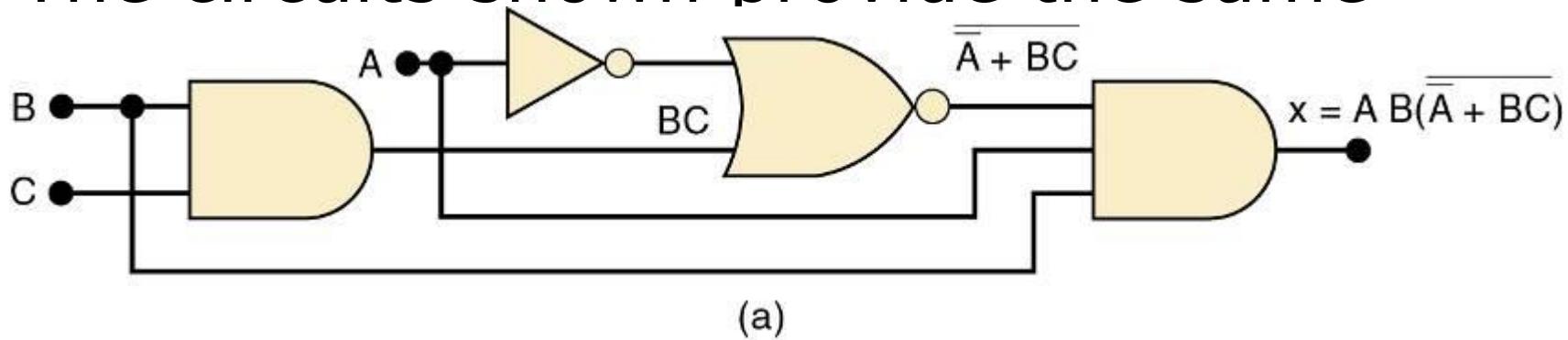
- A Sum-of-products (SOP) expression will appear as two or more AND terms ORed together.
 1. $ABC + \bar{A}\bar{B}\bar{C}$
 2. $AB + \bar{A}\bar{B}\bar{C} + \bar{C}\bar{D} + D$
 3. $\bar{A}B + C\bar{D} + EF + GK + H\bar{L}$

4-1 Sum-of-Products Form

- The product-of-sums (POS) form consists of two or more OR terms (sums) ANDed together.
 - $(A + \bar{B} + C)(A + \bar{C})$
 - $(A + \bar{B})(\bar{C} + D)F$
 - $(A + C)(B + \bar{D})(\bar{B} + C)(A + \bar{D} + \bar{E})$

4-2 Simplifying Logic Circuits

- The circuits shown provide the same

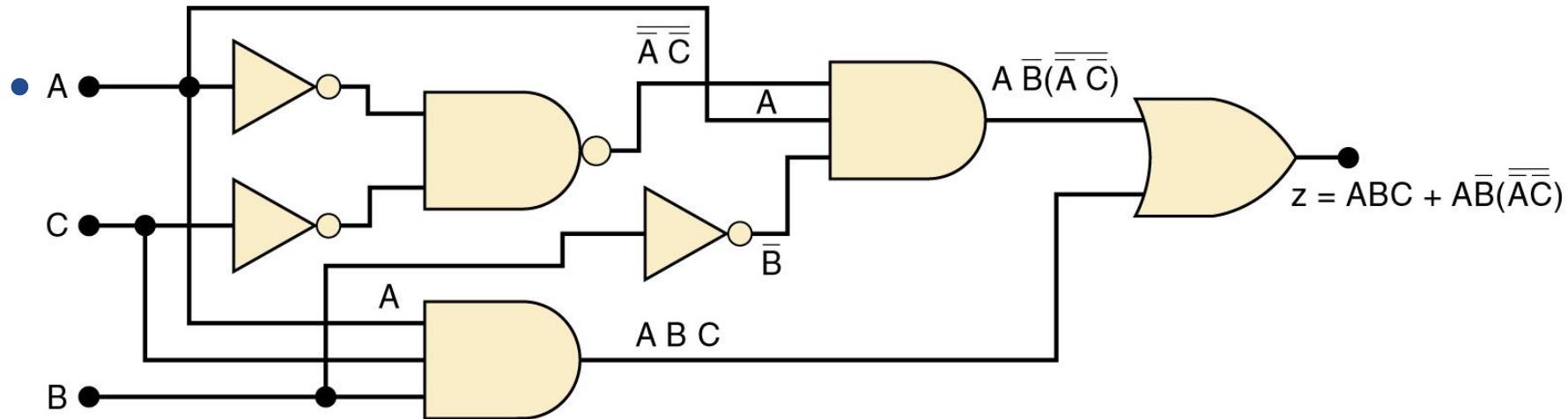


Logic circuits can be simplified using Boolean algebra and Karnaugh mapping.

4-3 Algebraic Simplification

- Place the expression in SOP form by applying DeMorgan's theorems and multiplying terms.
- Check the SOP form for common factors.
 - Factoring where possible should eliminate one or more terms.

4-3 Algebraic Simplification



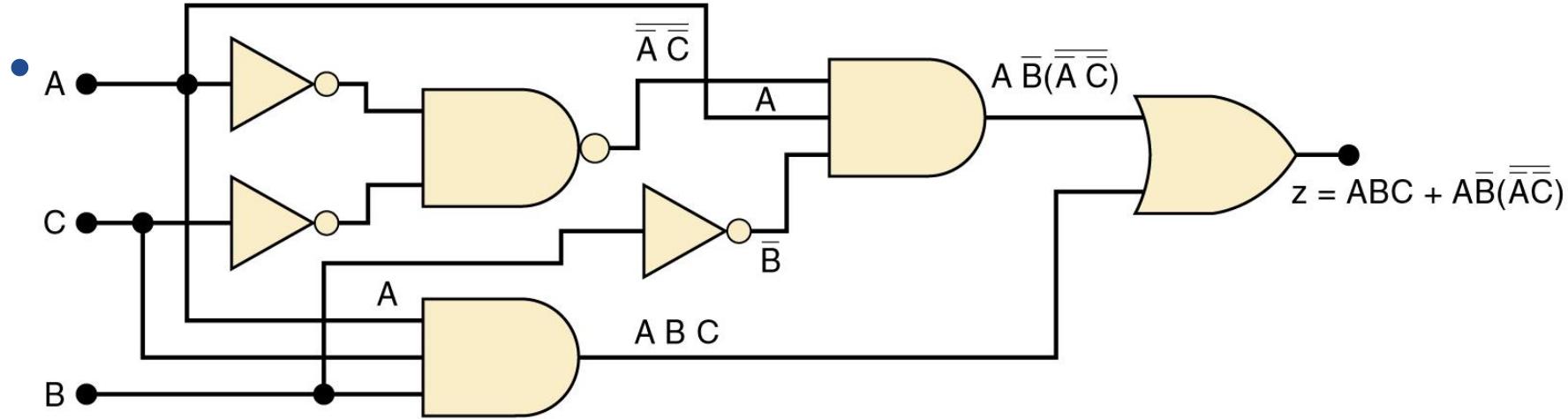
The first step is to determine the expression for the output: $=$

Once the expression is determined, break down large inverter signs by DeMorgan's theorems & multiply out all terms.

$$z = ABC + AB \cdot (A \bar{C})$$

$$\begin{aligned} z &= ABC + A\bar{B}(\bar{A} + \bar{C}) && [\text{theorem (17)}] \\ &= ABC + A\bar{B}(A + C) && [\text{cancel double inversions}] \\ &= ABC + A\bar{B}A + A\bar{B}C && [\text{multiply out}] \\ &= ABC + A\bar{B} + A\bar{B}C && [A \cdot A = A] \end{aligned}$$

4-3 Algebraic Simplification



Factoring—the first & third terms at $z = AC(B + \bar{B}) + A\bar{B}$
AC in common, which can be factored out:

Since $B + \bar{B} = 1$,

then...

Factor out **A**, which
results in...

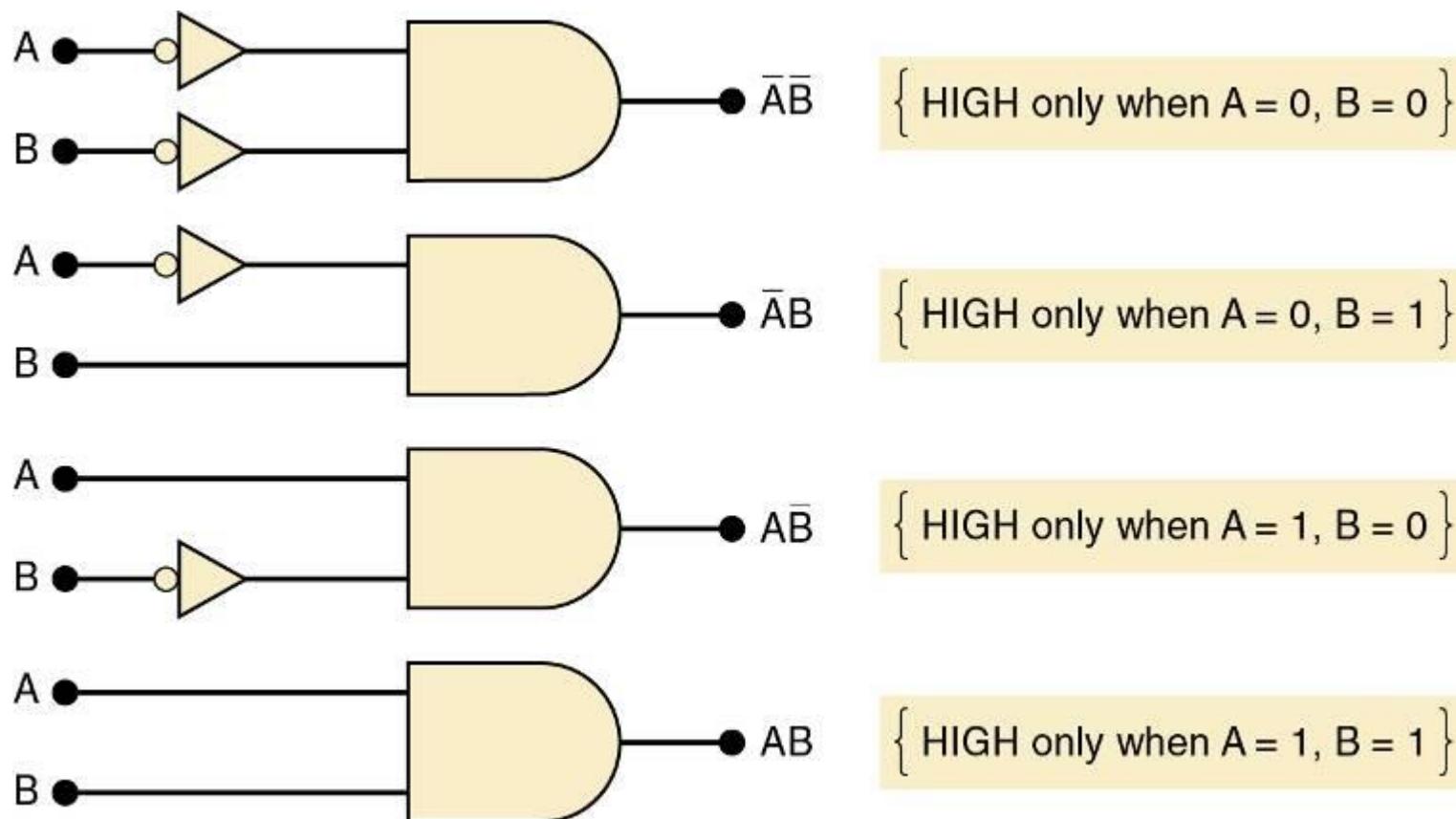
$$\begin{aligned} z &= AC(1) + A\bar{B} \\ &= AC + A\bar{B} \end{aligned}$$

4-4 Designing Combinational Logic Circuits

- To solve any logic design problem:
 - Interpret the problem and set up its truth table.
 - Write the AND (product) term for each case where output = 1.
 - Combine the terms in SOP form.
 - Simplify the output expression if possible.
 - Implement the circuit for the final, simplified expression.

4-4 Designing Combinational Logic Circuits

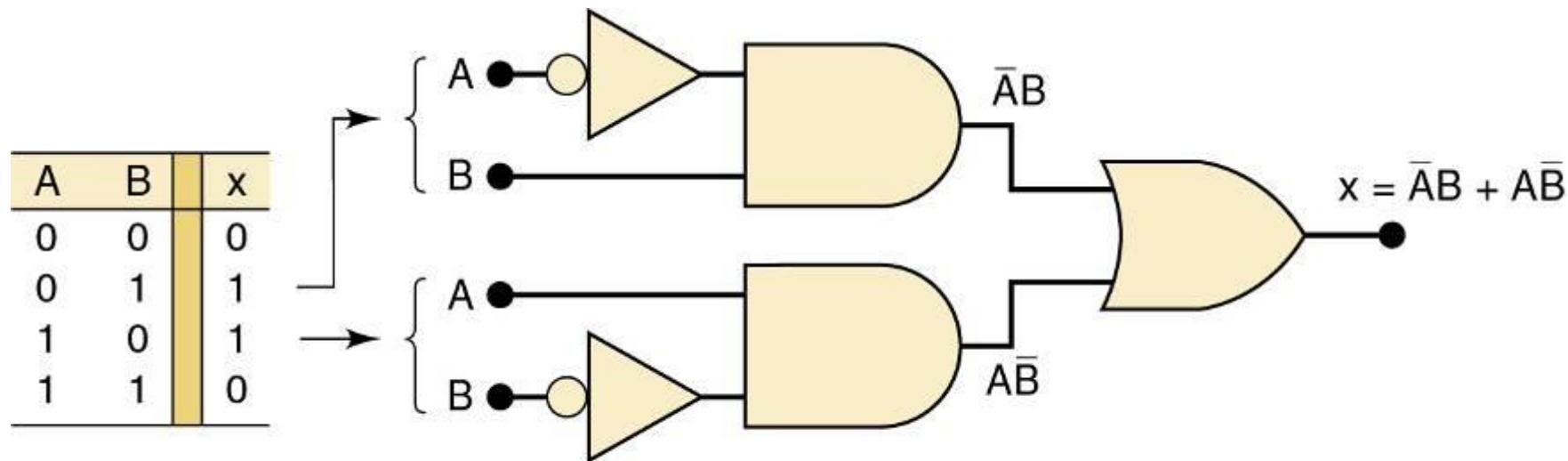
An **AND** gate with appropriate inputs can be used to produce a HIGH output for a specific set of input levels.



4-4 Designing Combinational Logic Circuits

Each set of input conditions that is to produce a 1 output is implemented by a separate **AND** gate.

The **AND** outputs are **ORed** to produce the final output.



4-4 Designing Combinational Logic Circuits

Truth table for a 3-input circuit.

AND terms for each case where output is 1.

| A | B | C | x |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

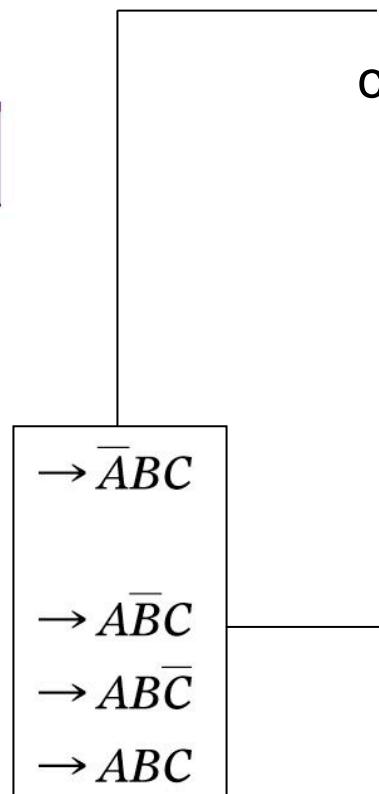
→ $\bar{A}\bar{B}\bar{C}$
→ $\bar{A}\bar{B}C$
→ ABC

4-4 Designing Combinational Logic Circuits

Design a logic circuit with three inputs, A, B, and C. Output to be HIGH only when a majority inputs are HIGH.

Truth table.

| A | B | C | x |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



AND terms for each case where output is 1.

SOP expression for the output:

$$x = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

4-4 Designing Combinational Logic Circuits

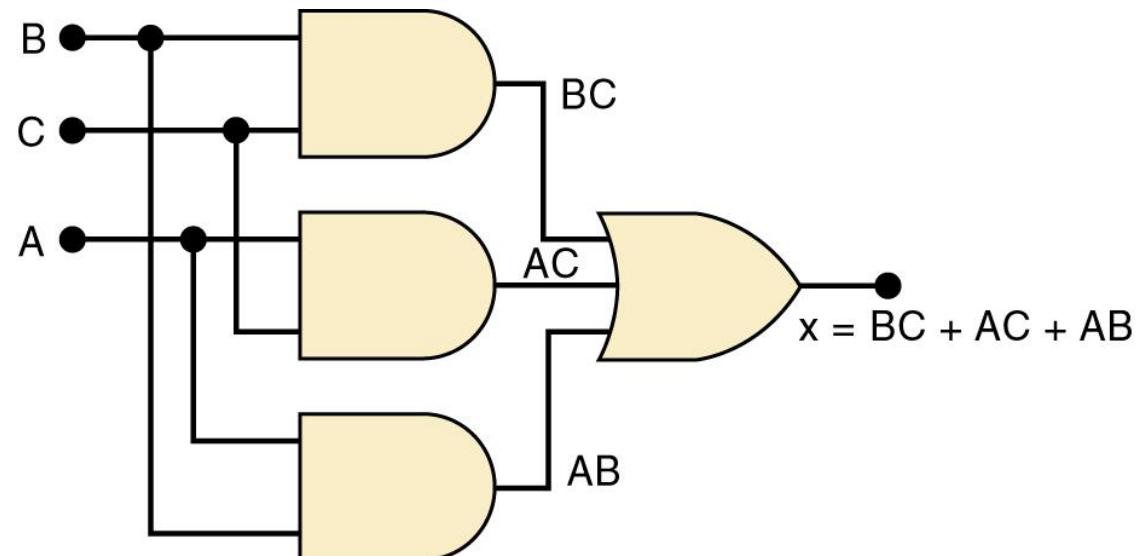
**Design a logic circuit with three inputs, A, B, and C.
Output to be HIGH only when a majority inputs are HIGH.**

Simplified output expression:

$$x = ABC + \overline{ABC} + \overline{AB}\overline{C} + \overline{A}\overline{BC} + \overline{A}\overline{B}\overline{C}$$

Implementing the circuit after factoring:

$$x = BC + AC + AB$$



Since the expression is in SOP form, the circuit is a group of **AND** gates, working into a single **OR** gate,

4-5 Karnaugh Map Method

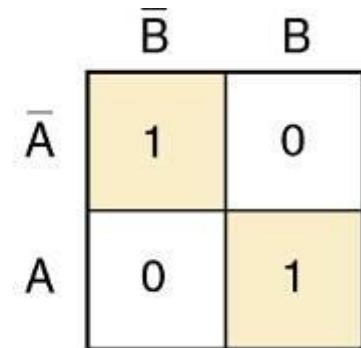
- A graphical method of simplifying logic equations or truth tables—also called a K map.
- Theoretically can be used for any number of input variables—practically limited to 5 or 6 variables.

4-5 Karnaugh Map Method

The truth table values are placed in the K map.
Shown here is a two-variable map.

| A | B | X |
|---|---|----------------------|
| 0 | 0 | 1 → $\bar{A}\bar{B}$ |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 → AB |

$$\left\{ x = \bar{A}\bar{B} + AB \right\}$$



4-5 Karnaugh Map Method

- Four-variable K-Map.

| A | B | C | D | X |
|---|---|---|---|------------------------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 → $\bar{A}B\bar{C}D$ |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 → $\bar{A}B\bar{C}D$ |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 → $\bar{A}B\bar{C}D$ |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 → $A\bar{B}CD$ |

$$\left\{ \begin{array}{l} X = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D \\ \quad + A\bar{B}\bar{C}D + A\bar{B}CD \end{array} \right\}$$

| | $\bar{C}\bar{D}$ | $\bar{C}D$ | CD | $C\bar{D}$ |
|------------------|------------------|------------|----|------------|
| $\bar{A}\bar{B}$ | 0 | 1 | 0 | 0 |
| $\bar{A}B$ | 0 | 1 | 0 | 0 |
| AB | 0 | 1 | 1 | 0 |
| A \bar{B} | 0 | 0 | 0 | 0 |

Adjacent K map square differ in only one variable both horizontally and vertically. A SOP expression can be obtained by **ORing** all squares that contain a 1.

4-5 Karnaugh Map Method

Looping 1s in adjacent groups of 2, 4, or 8 will result in further simplification.

| | \bar{C} | C |
|------------|-----------|-----|
| $\bar{A}B$ | 0 | 0 |
| $\bar{A}B$ | 1 | 1 |
| AB | 0 | 0 |
| $A\bar{B}$ | 0 | 0 |

$X = \bar{A}\bar{B}\bar{C} + \bar{A}BC = \bar{A}B$

| | \bar{C} | C |
|------------|-----------|-----|
| $\bar{A}B$ | 0 | 0 |
| $\bar{A}B$ | 1 | 0 |
| AB | 1 | 0 |
| $A\bar{B}$ | 0 | 0 |

$X = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} = BC$

| | \bar{C} | C |
|------------|-----------|-----|
| $\bar{A}B$ | 1 | 0 |
| $\bar{A}B$ | 0 | 0 |
| AB | 0 | 0 |
| $A\bar{B}$ | 1 | 0 |

$X = \bar{A}\bar{B}\bar{C} + A\bar{B}C = \bar{B}\bar{C}$

| | $\bar{C}\bar{D}$ | $\bar{C}D$ | CD | $C\bar{D}$ |
|------------|------------------|------------|------|------------|
| $\bar{A}B$ | 0 | 0 | 1 | 1 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| AB | 0 | 0 | 0 | 0 |
| $A\bar{B}$ | 1 | 0 | 0 | 1 |

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

**Groups of 4
(Quads)**

| | CD | $\bar{C}D$ | CD | $C\bar{D}$ |
|------------|------|------------|------|------------|
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| AB | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 0 | 0 | 0 | 0 |

$X = AB$

**Groups of 8
(Octets)**

| | CD | $\bar{C}D$ | CD | $C\bar{D}$ |
|------------|------|------------|------|------------|
| $\bar{A}B$ | 1 | 0 | 0 | 1 |
| $\bar{A}B$ | 1 | 0 | 0 | 1 |
| AB | 1 | 0 | 0 | 1 |
| $A\bar{B}$ | 1 | 0 | 0 | 1 |

$X = \bar{D}$

4-5 Karnaugh Map Method

- When the largest possible groups have been looped, only the common terms are placed in the final expression.
 - Looping may also be wrapped between top, bottom, and sides.

4-5 Karnaugh Map Method

- Complete K map simplification process:
 - Construct the K map, place 1s as indicated in the truth table.
 - Loop 1s that are not adjacent to any other 1s.
 - Loop 1s that are in pairs.
 - Loop 1s in octets even if they have already been looped.
 - Loop quads that have one or more 1s not already looped.

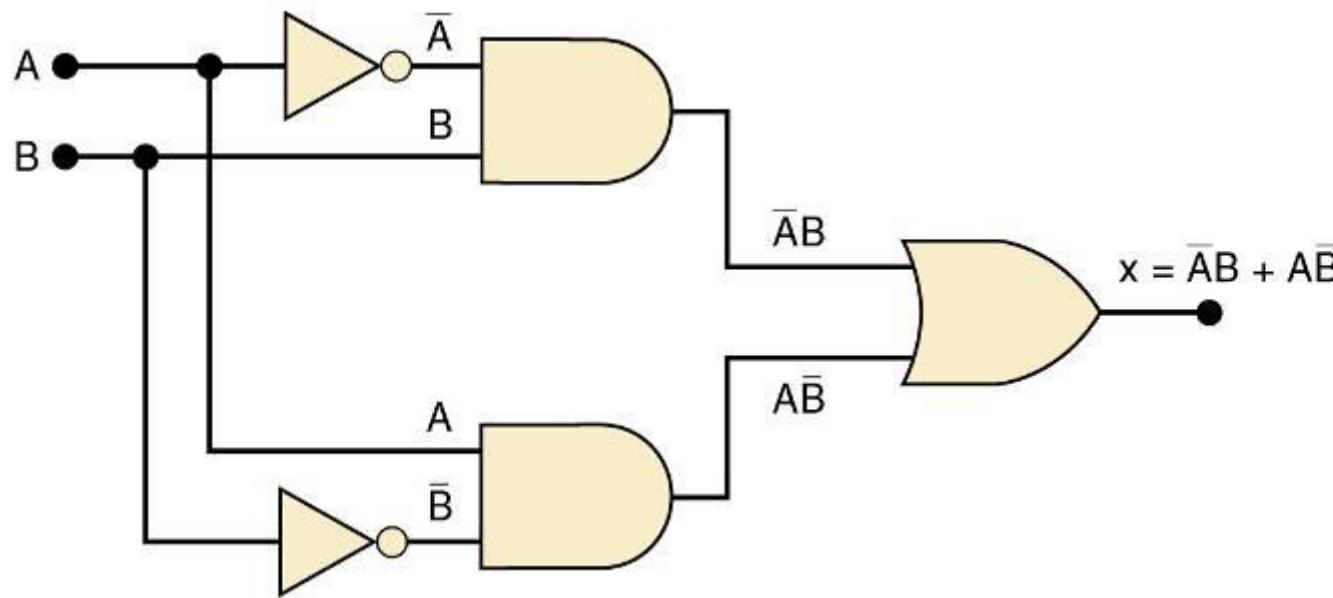
4-5 Karnaugh Map Method

- Loop any pairs necessary to include 1st not already looped.
- Form the OR sum of terms generated by each loop.

4-6 Exclusive OR and Exclusive NOR Circuits

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

4-6 Exclusive OR and Exclusive NOR Circuits



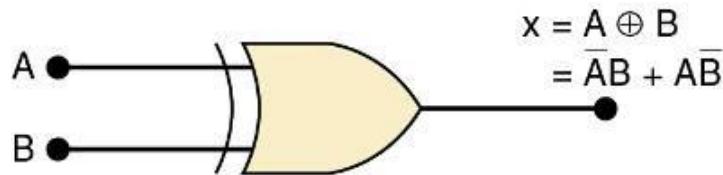
| A | B | x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

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

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

4-6 Exclusive OR and Exclusive NOR Circuits

- Traditional XOR gate symbol.



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

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)

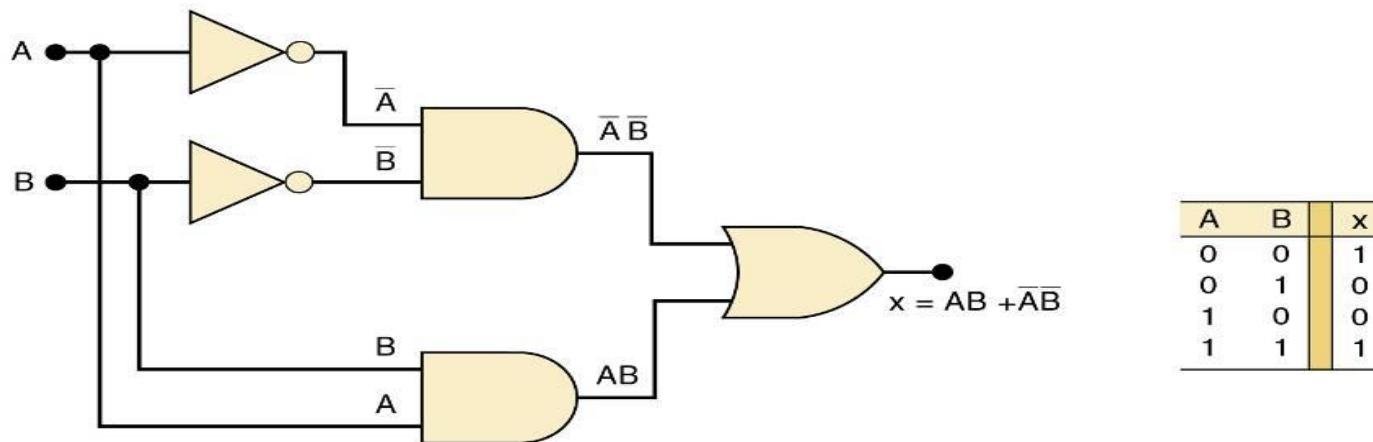
74HC86 Quad **XOR** (high-speed CMOS)

4-6 Exclusive OR and Exclusive NOR Circuits

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

4-6 Exclusive OR and Exclusive NOR Circuits

- Exclusive NOR circuit and truth table.

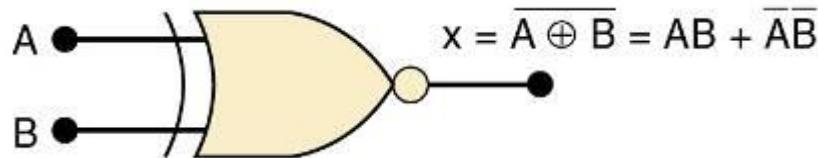


Output expression $x = \overline{AB} + \overline{\overline{AB}}$

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

4-6 Exclusive OR and Exclusive NOR Circuits

- Traditional XNOR gate symbol.



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

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

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.

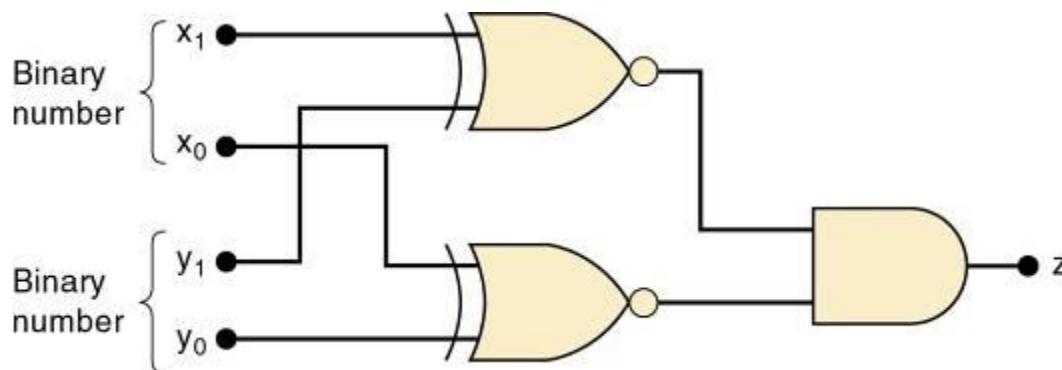
74LS266 Quad **XNOR** (TTL family)

74C266 Quad **XOR** (CMOS)

74HC266 Quad **XOR** (high-speed CMOS).

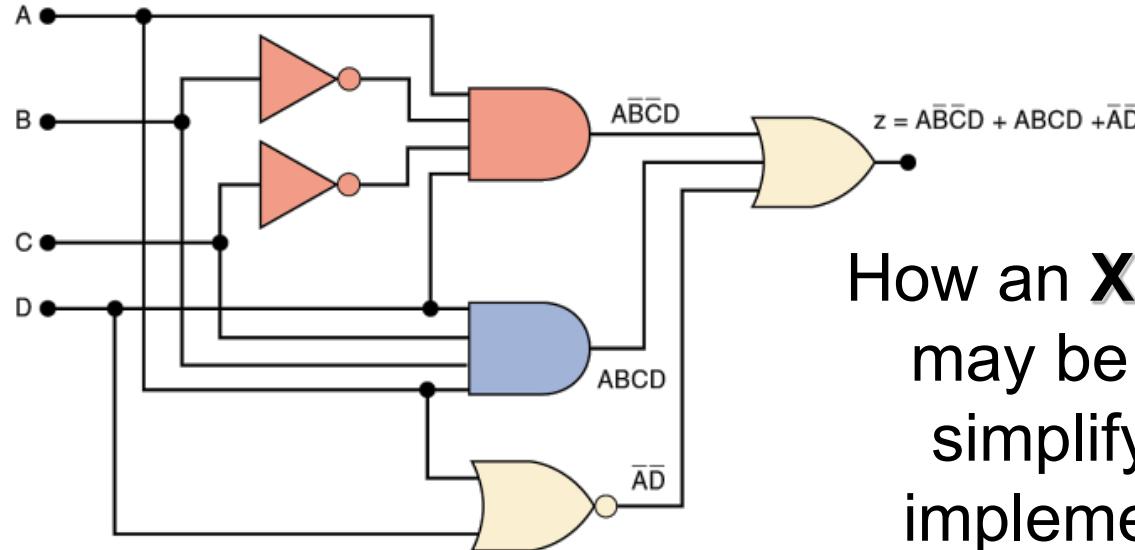
4-6 Exclusive OR and Exclusive NOR Circuits

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

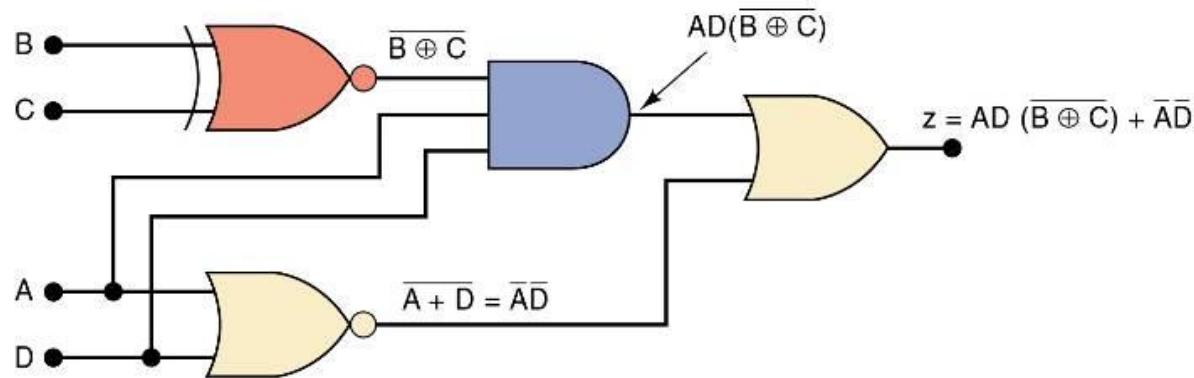


| x_1 | x_0 | y_1 | y_0 | z (Output) |
|-------|-------|-------|-------|--------------|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1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

4-6 Exclusive OR and Exclusive NOR Circuits

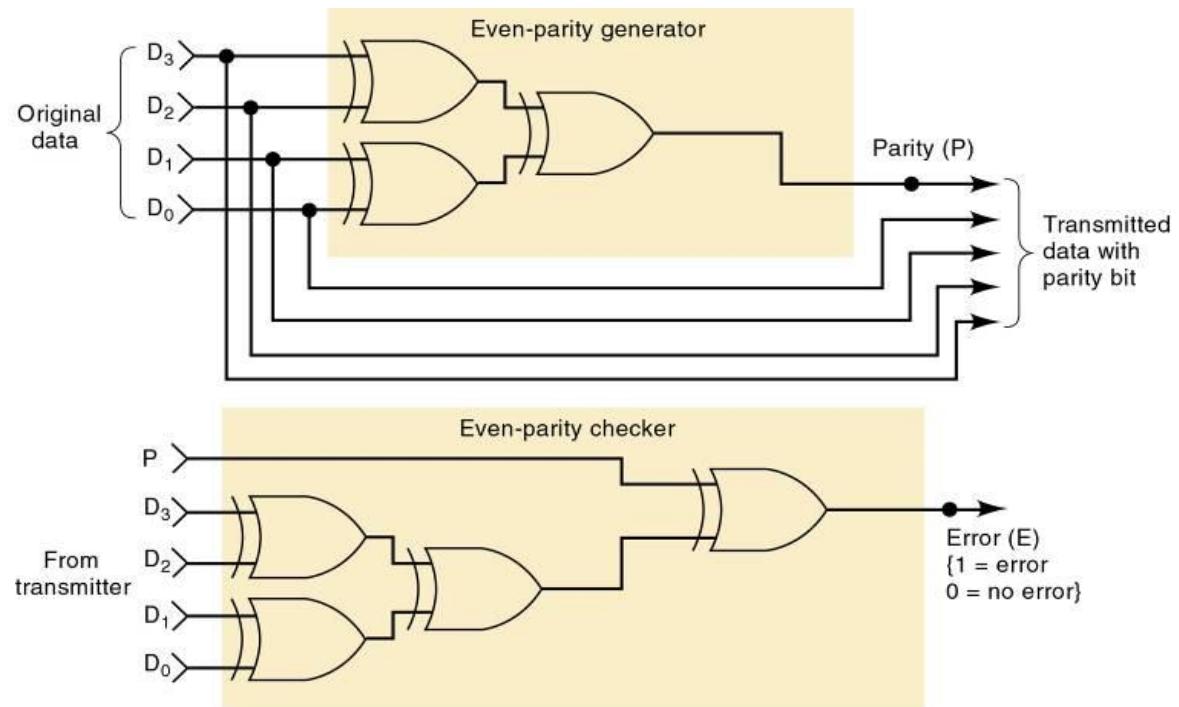


How an **XNOR** gate
may be used to
simplify circuit
implementation.



4-7 Parity Generator and Checker

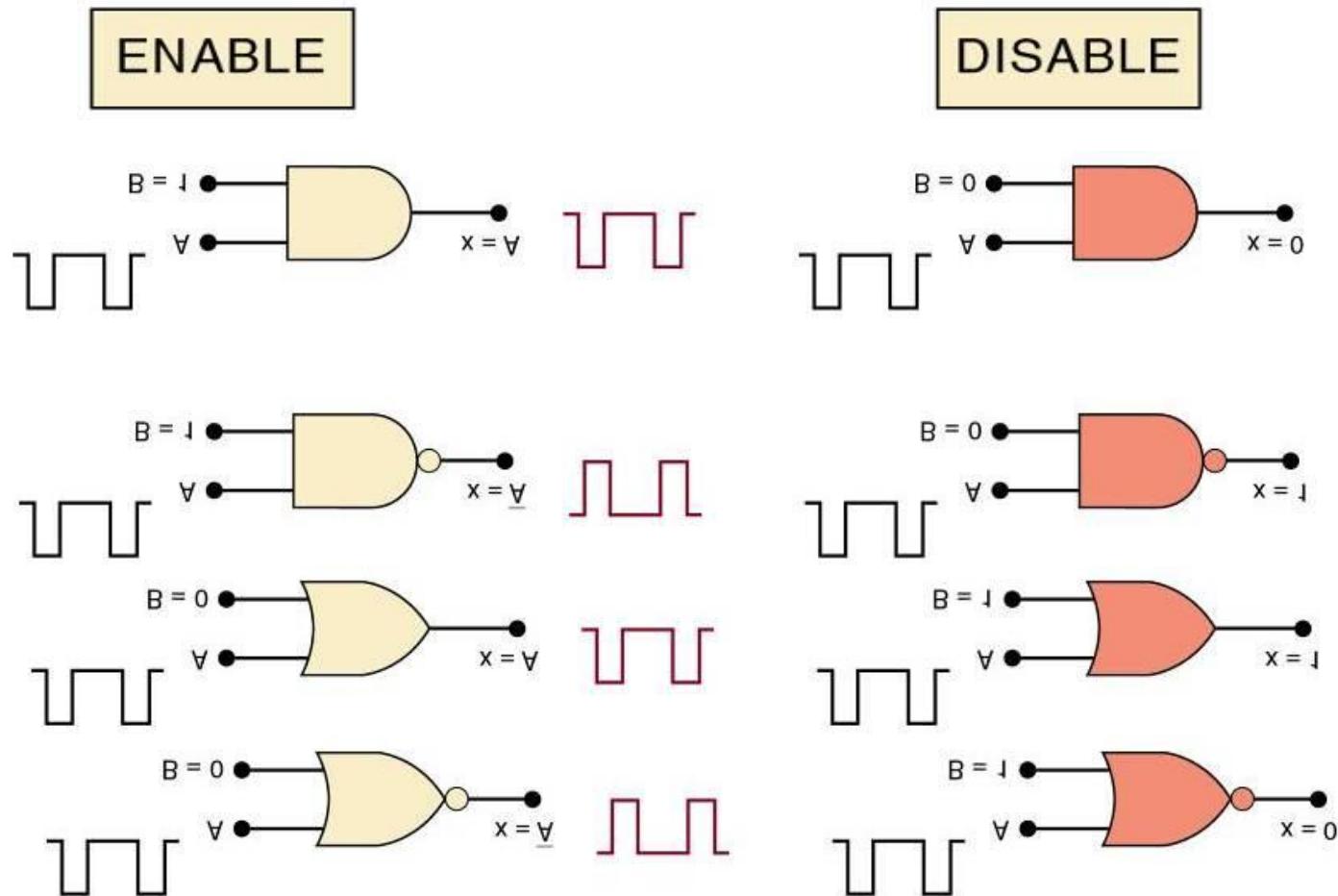
- XOR and XNOR gates are useful in circuits for parity generation and checking.



4-8 Enable/Disable Circuits

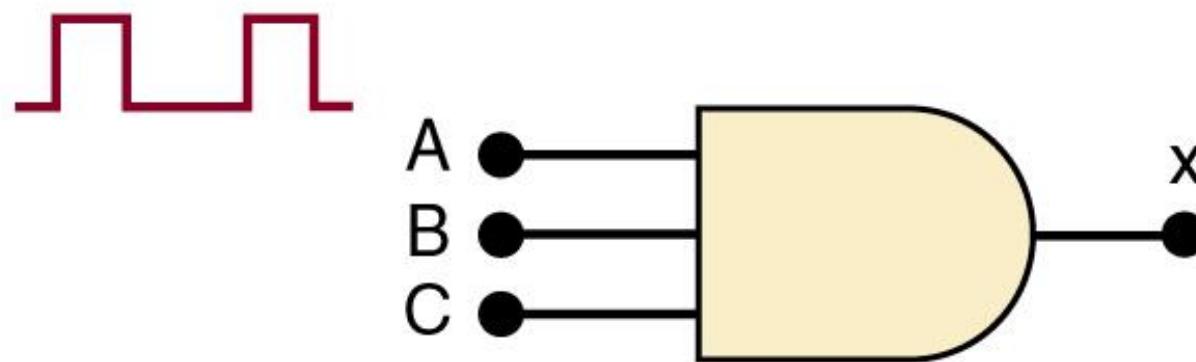
- Situations requiring enable/disable circuits occur frequently in digital circuit design.
 - A circuit is enabled when it allows the passage of an input signal to the output.
 - A circuit is disabled when it prevents the passage of an input signal to the output.

4-8 Enable/Disable Circuits



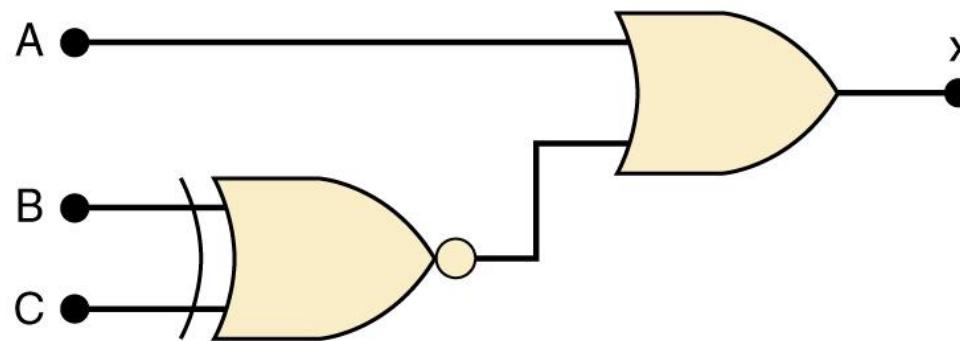
4-8 Enable/Disable Circuits

- A logic circuit that will allow a signal to pass to output only when control inputs B and C are both HIGH.
- Otherwise, output will stay LOW.



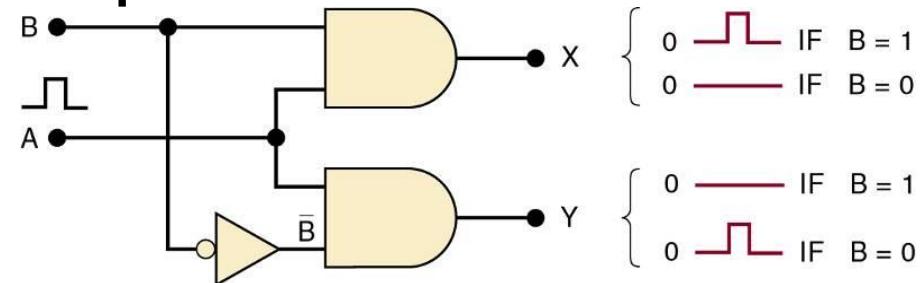
4-8 Enable/Disable Circuits

- A logic circuit that will allow a signal to pass to output only when one, but not both control inputs are HIGH.
- Otherwise, output will stay HIGH.



4-8 Enable/Disable Circuits

- A logic circuit with input signal A, control input B, and outputs X and Y, which operates as:
- When $B = 1$, output X will follow input A, and output Y will be 0.
- When $B = 0$, output X will be 0, and output Y will follow input A.



4-9 Basic Characteristics of Digital ICs

- IC “chips” consist of resistors, diodes & transistors fabricated on a piece of semiconductor material called a substrate.

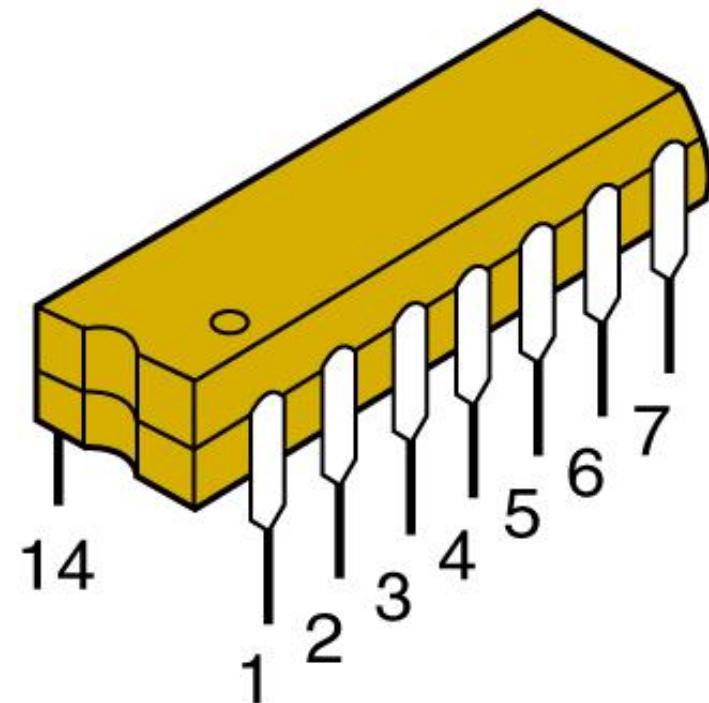
Digital ICs are often categorized by complexity, according to the number of logic gates on the

| Complexity | Gates per Chip |
|--------------------------------------|--------------------|
| Small-scale integration (SSI) | Fewer than 12 |
| Medium-scale integration (MSI) | 12 to 99 |
| Large-scale integration (LSI) | 100 to 9999 |
| Very large-scale integration (VLSI) | 10,000 to 99,999 |
| Ultra large-scale integration (ULSI) | 100,000 to 999,999 |
| Giga-scale integration (GSI) | 1,000,000 or more |

4-9 Basic Characteristics of Digital ICs

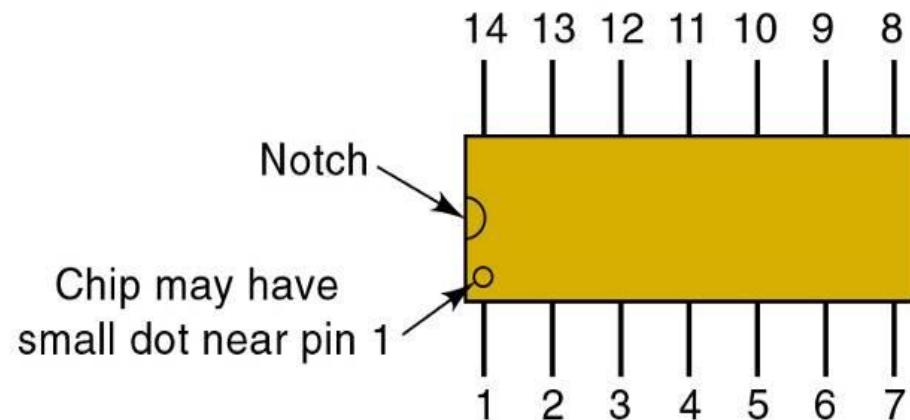
- The dual-in-line package (DIP) contains two parallel rows of pins.

The DIP is probably the most common digital IC package found in older digital equipment.



4-9 Basic Characteristics of Digital ICs

- Pins are numbered counterclockwise, viewed from the top of the package, with respect to an identifying notch or dot at one end.

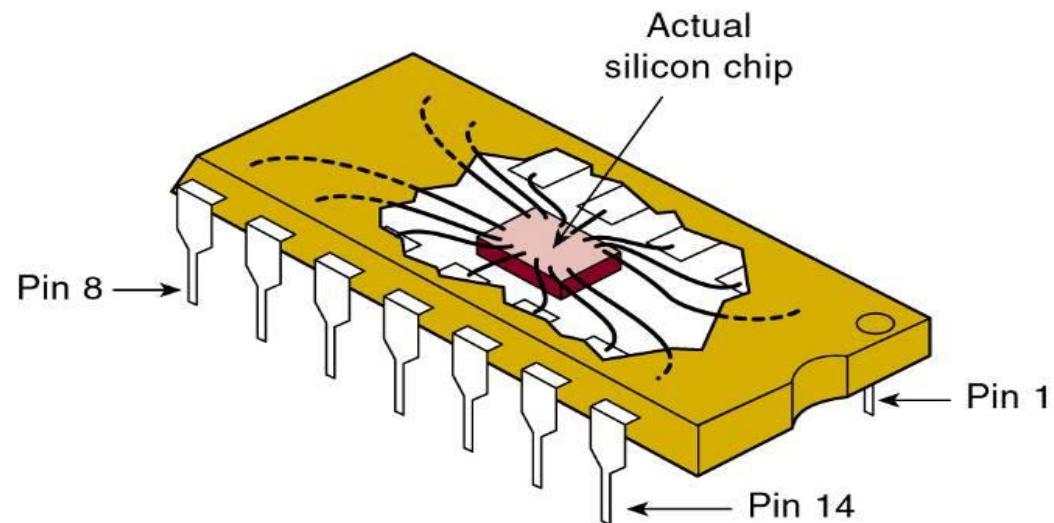


**Shown is a 14-pin DIP
that measures .75" x .25".**

4-9 Basic Characteristics of Digital ICs

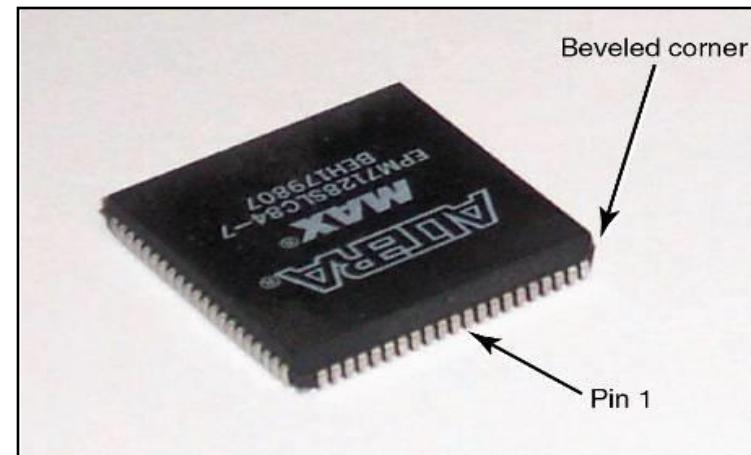
- The actual silicon chip is much smaller than the DIP—typically about 0.05" square.

The silicon chip is connected to the pins of the DIP by very fine (1-mil) wires.



4-9 Basic Characteristics of Digital ICs

- The PLCC is one of many packages common in modern digital circuits.
 - This type uses J-shaped leads which curl under the IC.



4-9 Basic Characteristics of Digital ICs

- ICs are also categorized by the type of components used in their circuits.
 - Bipolar ICs use NPN and PNP transistors
 - Unipolar ICs use FET transistors.

4-9 Basic Characteristics of Digital ICs

- The transistor-transistor logic (TTL) family consists of subfamilies shown here:

| TTL Series | Prefix | Example IC |
|---------------------------------|--------|------------------------|
| Standard TTL | 74 | 7404 (hex INVERTER) |
| Schottky TTL | 74S | 74S04 (hex INVERTER) |
| Low-power Schottky TTL | 74LS | 74LS04 (hex INVERTER) |
| Advanced Schottky TTL | 74AS | 74AS04 (hex INVERTER) |
| Advanced low-power Schottky TTL | 74ALS | 74ALS04 (hex INVERTER) |

Differences between the TTL devices is limited to electrical characteristics such as power dissipation & switching speed.
Pin layout and logic operations are the same.

4-9 Basic Characteristics of Digital ICs

- The Complimentary Metal-Oxide Semiconductor (CMOS) family consists of several series

| CMOS Series | Prefix | Example IC |
|--|--------|--------------------------|
| Metal-gate CMOS | 40 | 4001 (quad NOR gates) |
| Metal-gate, pin-compatible with TTL | 74C | 74C02 (quad NOR gates) |
| Silicon-gate, pin-compatible with TTL, high-speed | 74HC | 74HC02 (quad NOR gates) |
| Silicon-gate, high-speed, pin-compatible and electrically compatible with TTL | 74HCT | 74HCT02 (quad NOR gates) |
| Advanced-performance CMOS, not pin-compatible or electrically compatible with TTL | 74AC | 74AC02 (quad NOR) |
| Advanced-performance CMOS, not pin-compatible with TTL, but electrically compatible with TTL | 74ACT | 74ACT02 (quad NOR) |

CMOS devices perform the same function as, but are not necessarily pin for pin compatible with TTL devices.

4-9 Basic Characteristics of Digital ICs

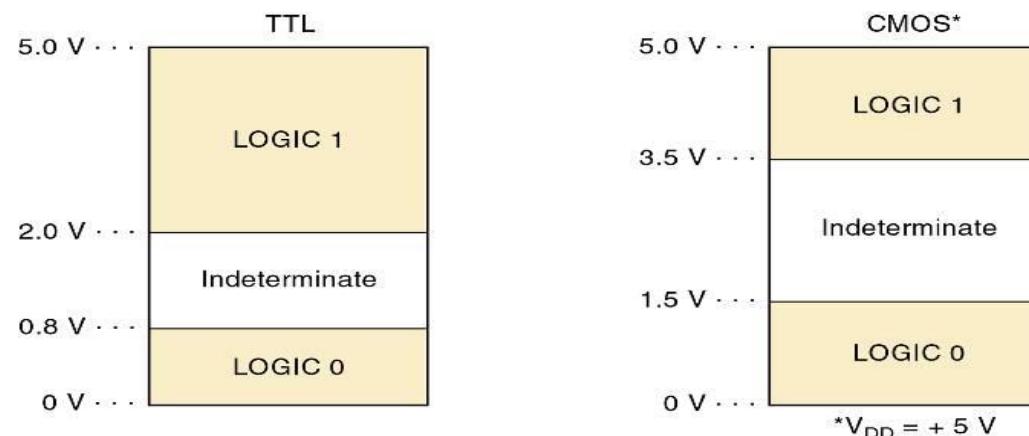
- Inputs not connected are said to be floating.
 - Floating TTL input acts like a logic 1.
 - Voltage measurement may appear indeterminate, but the device behaves as if there is a 1 on the floating input
 - Floating CMOS inputs can cause overheating and damage to the device.

4-9 Basic Characteristics of Digital ICs

- Some ICs have protection circuits built in.
The best practice is to tie all unused inputs.
 - Either high or low.

4-9 Basic Characteristics of Digital ICs

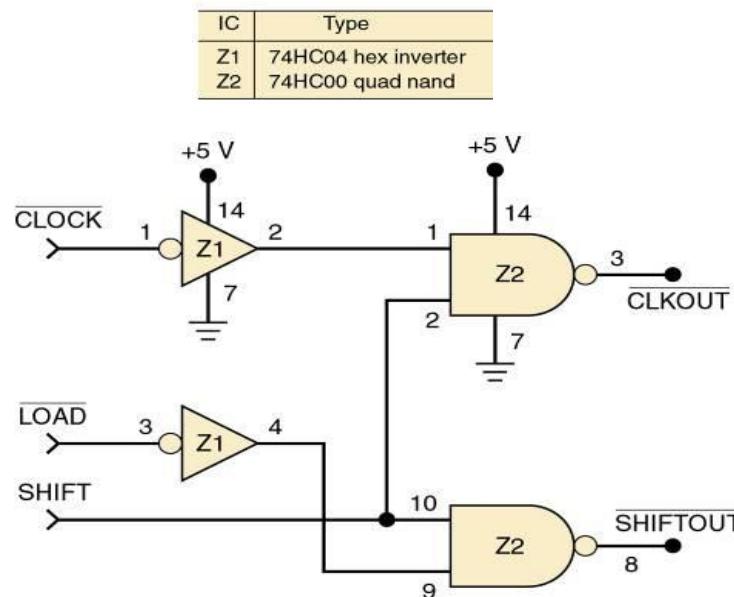
- Voltages in the indeterminate range provide unpredictable results and should be avoided.



Logic levels for TTL and CMOS devices.

4-9 Basic Characteristics of Digital ICs

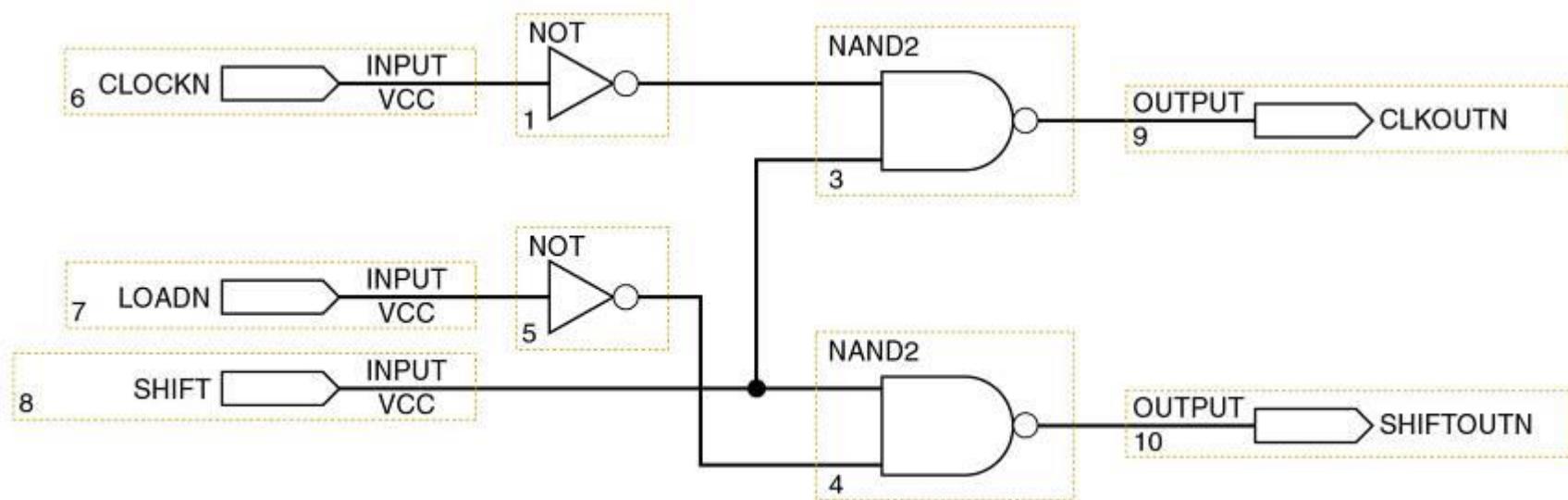
- A connection diagram shows all electrical connections, pin numbers, IC numbers, component values, signal names, and power supply voltages.



This circuit uses logic gates from two different ICs. Each gate input & output pin number is indicated on the diagram, to easily reference any point in the circuit. Power/ ground connections to each IC are shown.

4-9 Basic Characteristics of Digital ICs

- Logic diagram using Quartus II schematic capture.

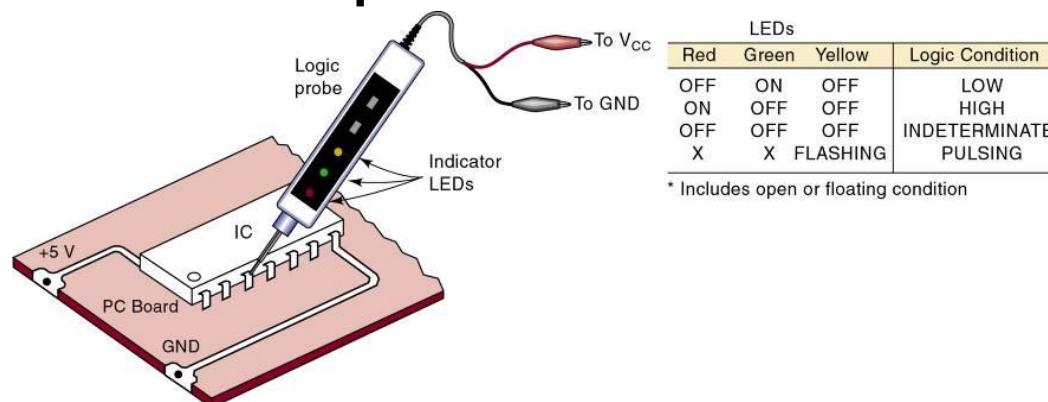


4-10 Troubleshooting Digital Systems

- Three basic steps in fixing a digital circuit or system that has a fault (failure):
 - Fault detection—determine operation to expected operation.
 - Fault isolation—test & measure to isolate the fault.
 - Fault correction—repair the fault.

4-10 Troubleshooting Digital Systems

- The basic troubleshooting tools are the logic probe, oscilloscope, and logic pulser.
- The logic probe will indicate the presence or absence of a signal when touched to a pin as indicated below.



4-11 Internal Digital IC Faults

- Most common internal failures:
 - Malfunction in the internal circuitry.
 - Outputs do not respond properly to inputs.
 - Outputs are unpredictable.
 - Inputs or outputs shorted to ground or VCC .
 - The input will be stuck in LOW or HIGH state.

4-11 Internal Digital IC Faults

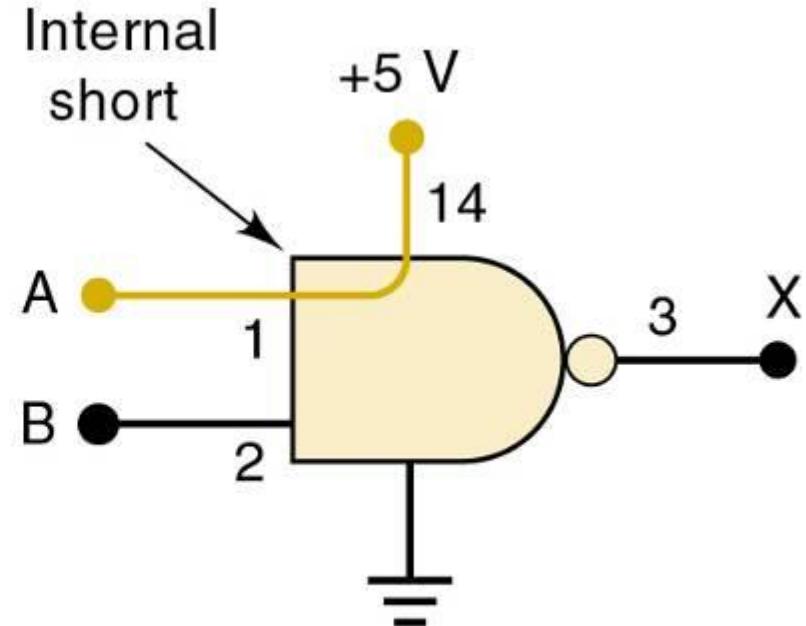
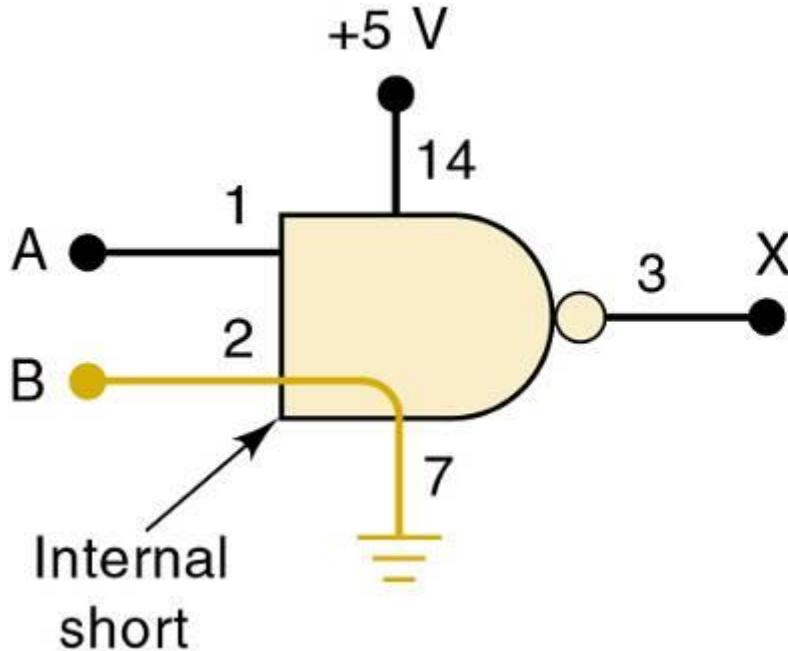
- Inputs or outputs open-circuited .
 - An open output will result in a floating indication.
 - Floating input in a TTL will result in a HIGH output.
 - Floating input in a CMOS device will result in erratic or possibly destructive output.

4-11 Internal Digital IC Faults

- Short between two pins (other than ground or VCC).
 - The signal at those pins will always be identical.

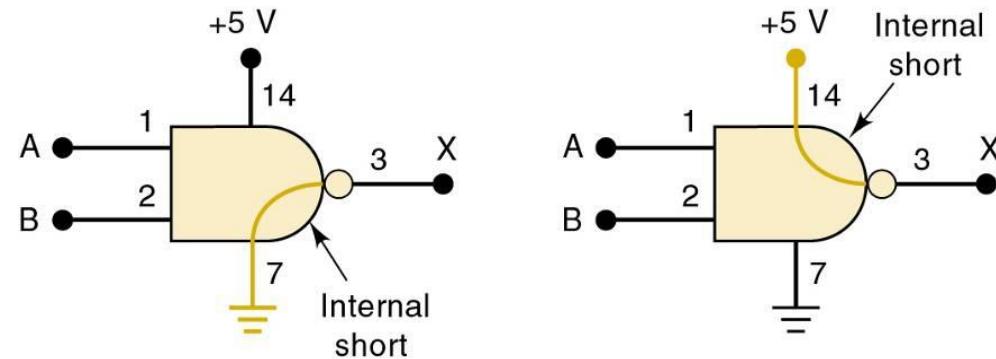
4-11 Internal Digital IC Faults

- These two types of failures force the input signal at the shorted pin to stay in the same state.
- Left—IC input internally shorted to ground.
- Right—IC input internally shorted to supply voltage.



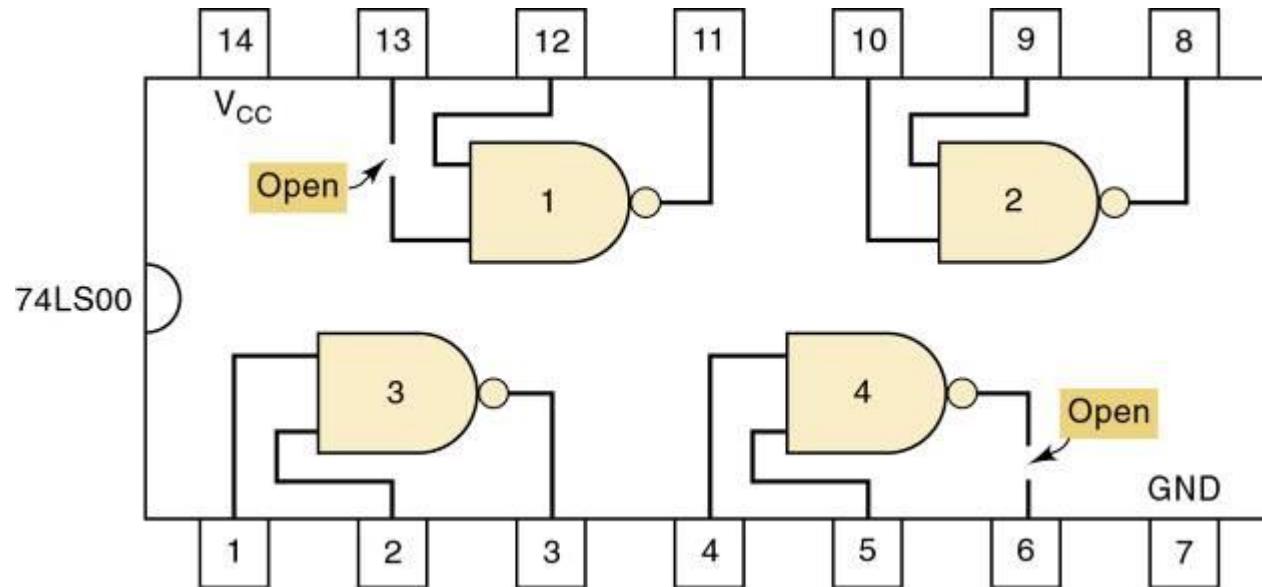
4-11 Internal Digital IC Faults

- These two types of failures do not affect signals at the IC inputs.
- Left—IC output internally shorted to ground.
- Right—IC output internally shorted to supply voltage.



4-11 Internal Digital IC Faults

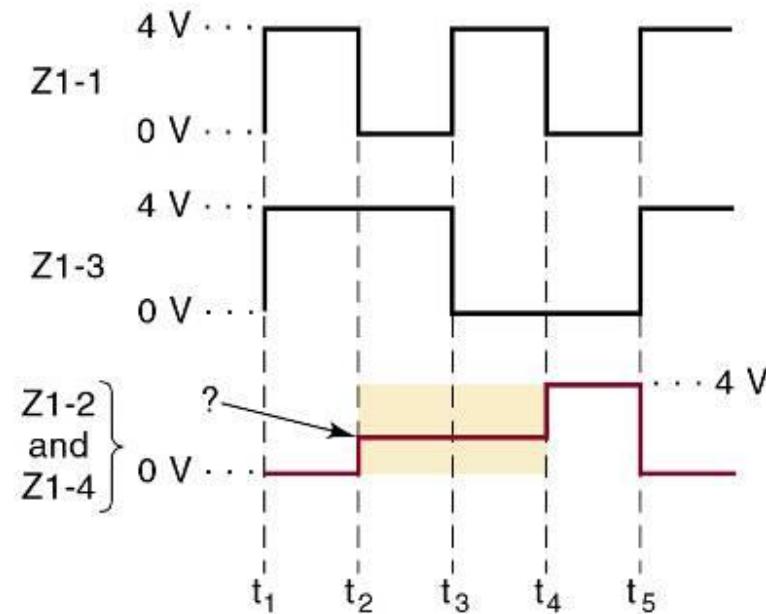
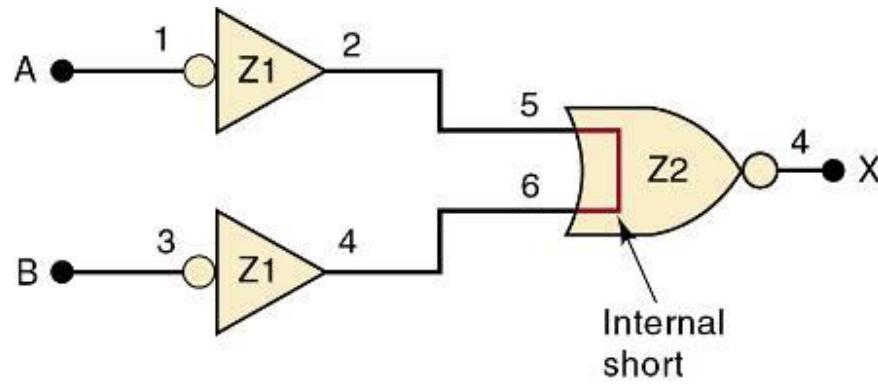
An IC with an internally open input will not respond to signals applied to that input pin.



An internally open output will produce an unpredictable voltage at that output pin.

4-11 Internal Digital IC Faults

An internal short between two pins of an IC will force the logic signals at those pins always to be identical.



When two input pins are internally shorted, the signals driving these pins are forced to be identical, and usually a signal with three distinct levels results.

4-12 External Faults

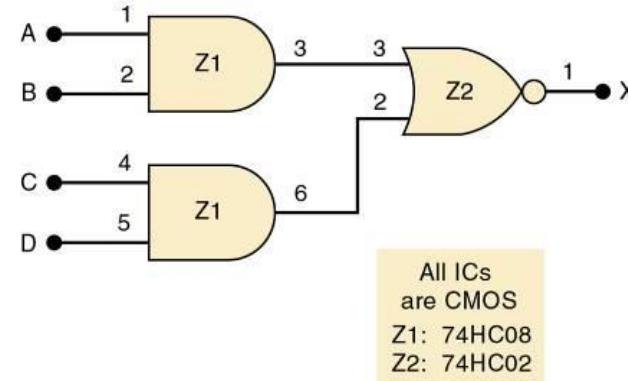
- Open signal lines—signal prevented from moving between points—can be caused by:
 - Broken wire.
 - Poor connections (solder or wire-wrap).
 - Cut or crack on PC board trace.
 - Bent or broken IC pins.

4-12 External Faults

- Faulty IC socket.
 - This type of fault can be detected visually and verified with an ohmmeter between the points in question.

4-12 External Faults

- What is the most probable fault in the circuit shown?



| Pin | Condition |
|------|---------------|
| Z1-1 | Pulsing |
| Z1-2 | HIGH |
| Z1-3 | Pulsing |
| Z1-4 | LOW |
| Z1-5 | Pulsing |
| Z1-6 | LOW |
| Z2-3 | Pulsing |
| Z2-2 | Indeterminate |
| Z2-1 | Indeterminate |

The indeterminate level at the NOR gate output is probably due to the indeterminate input at pin 2.

Because there is a LOW at Z1-6, this LOW should *also* be at Z2-2.

4-12 External Faults

- Shorted signal lines—the same signal appears on two or more pins—and VCC or ground may also be shorted, caused by:
 - Sloppy wiring.
 - Solder bridges.
 - Incomplete etching.
 - This type of fault can be detected visually and verified with an ohmmeter between the points in question.

4-12 External Faults

- Faulty power supply—ICs will not operate or will operate erratically.
 - May lose regulation due to an internal fault or because circuits are drawing too much current.
- Verify that power supplies provide the specified range of voltages and are properly grounded.

4-12 External Faults

- Use an oscilloscope to verify that AC ripple is not present and verify that DC voltages stay regulated.
- Some ICs are more tolerant of power variations and may operate properly—others do not.
 - Check power and ground levels at each IC that appears to be operating incorrectly.

4-12 External Faults

- Output loading—caused by connecting too many inputs to the output of an IC, exceeding output current rating.
 - Output voltage falls into the indeterminate range.
 - Called loading the output signal.
 - Usually a result of poor design or bad connection.

4-13 Troubleshooting Prototyped Circuits

- Validate your design before you prototype to save yourself frustration in building a circuit from a flawed design.
- Eliminate as many possible locations of the fault as possible.
 - Fault scenario #1: U1B pin 5 is actually connected to **A**, rather than A. (page 192)
 - Troubleshooting steps to isolate the problem

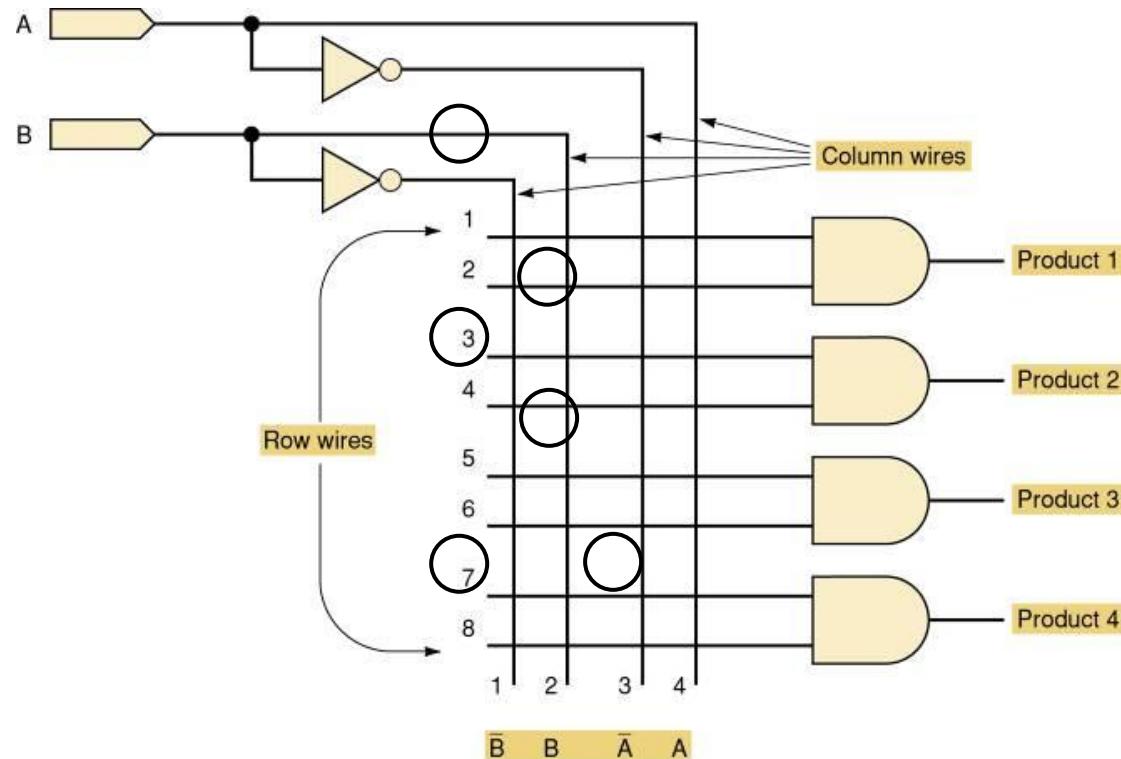
4-14 Programmable Logic Devices

- The concept behind programmable logic devices is simple—lots of logic gates in a single IC.
 - Control of the interconnection of these gates electronically.
- PLDs allow the design process to be automated. Designers identify inputs, outputs, and logical relationships.
 - PLDs are electronically configured to form the defined logic circuits.

4-14 Programmable Logic Devices

- PLDs use a switch matrix that is often referred to as a programmable array.

By deciding which intersections are connected & which are not, we can “program” the way the inputs are connected to the outputs of the array.



4-14 Programmable Logic Devices

- For out-of-system programming the PLD is placed in a programmer, connected to a PC.
 - PC software translates and loads the information.

4-14 Programmable Logic Devices

- In-system programming is done by connecting directly to “portal” pins while the IC remains in the system.
 - An interface cable connects the PLD to a PC running the software that loads the device.

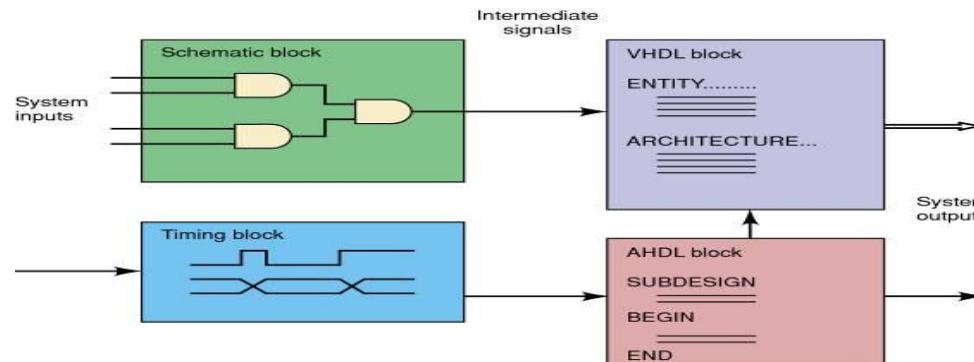
4-14 Programmable Logic Devices

- Logic circuits can be described using schematic diagrams, logic equations, truth tables, and HDL.
 - PLD development software can convert any of these descriptions into 1s and 0s and loaded into the PLD.
- Altera MAX+PLUS II is a development software that allows the user to describe circuits using graphic design files and timing diagrams.

4-14 Programmable Logic Devices

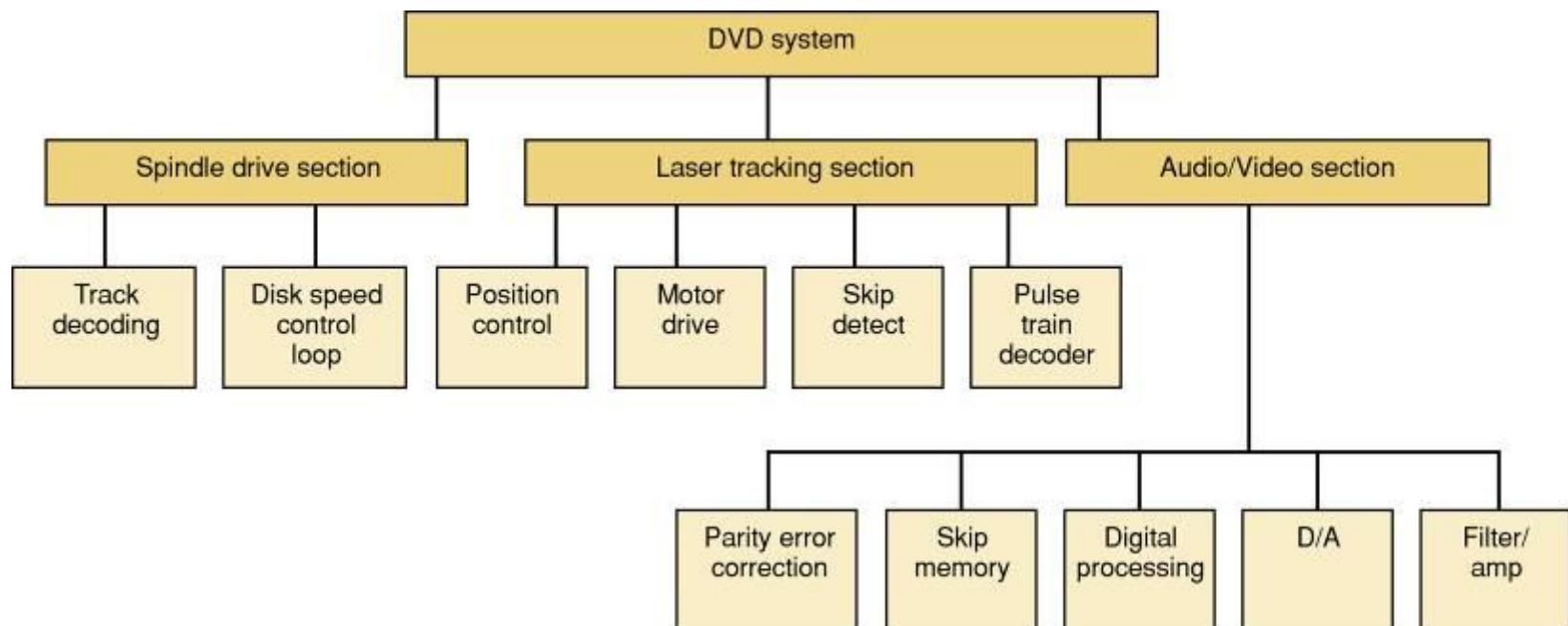
- Hierarchical design—small logic circuits are defined and combined with other circuits to form a large section of a project.
 - Large sections can be combined and connected for form a system.

Combining blocks developed using different description methods.



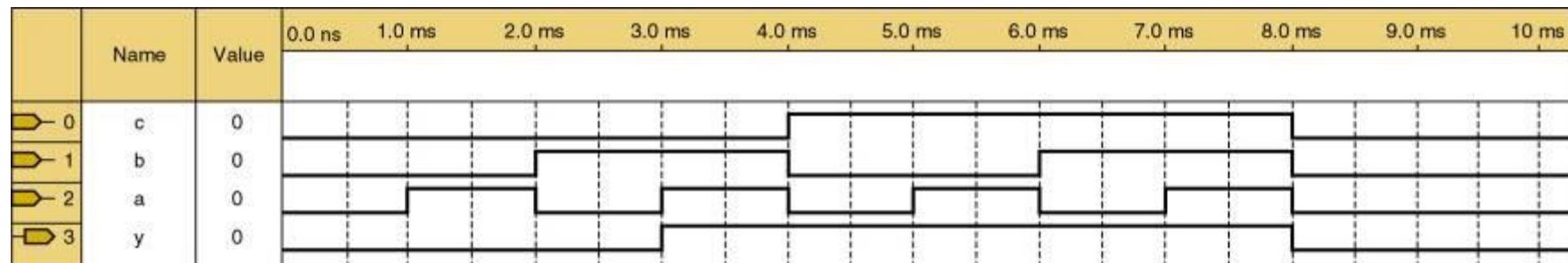
4-14 Programmable Logic Devices

- Top-down design—requires the definition of subsections that will make up the system.



4-14 Programmable Logic Devices

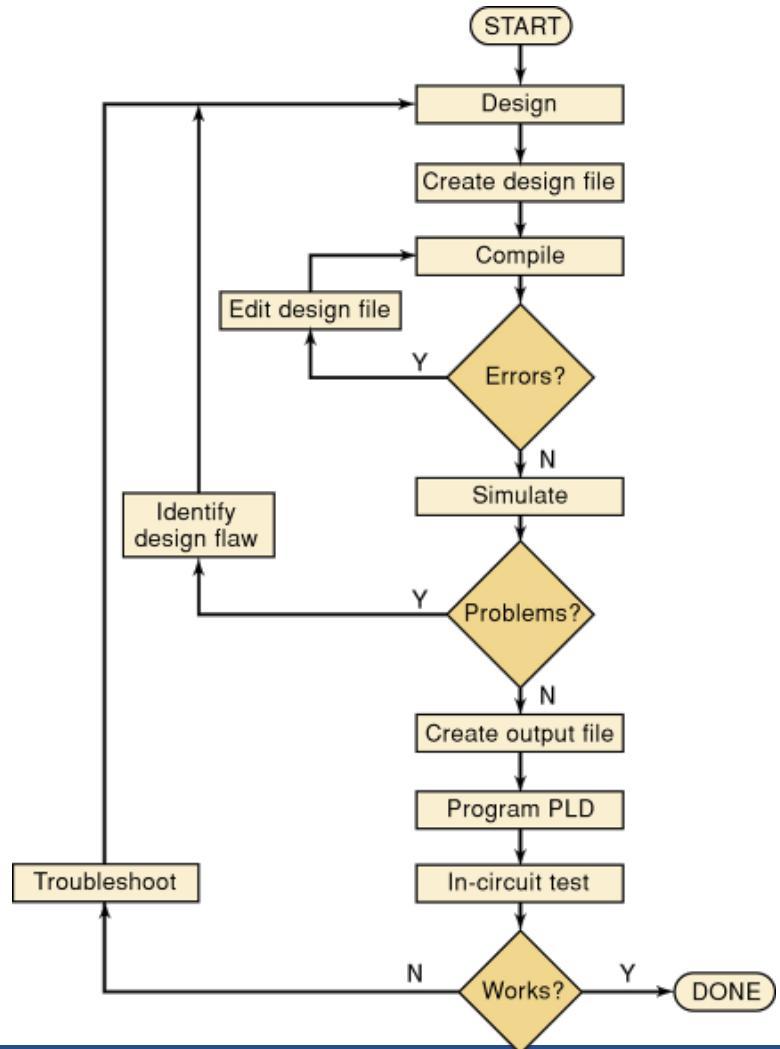
- Timing simulation of a circuit described in HDL.



4-14 Programmable Logic Devices

- A system is built from the bottom up.
- Each block is described by a design file.
- After testing it is compiled using development software.
- The compiled block is tested using a simulator for verify correct operation.
- A PLD is programmed to verify correct operation.

4-14 Programmable Logic Devices



4-15 Representing Data in HDL

- Every programming language & HDL has its own unique way of identifying number systems.
 - Generally done with a prefix to indicate the system.
- When we read one of these number designations, we must think of it as a symbol that represents a binary bit pattern.

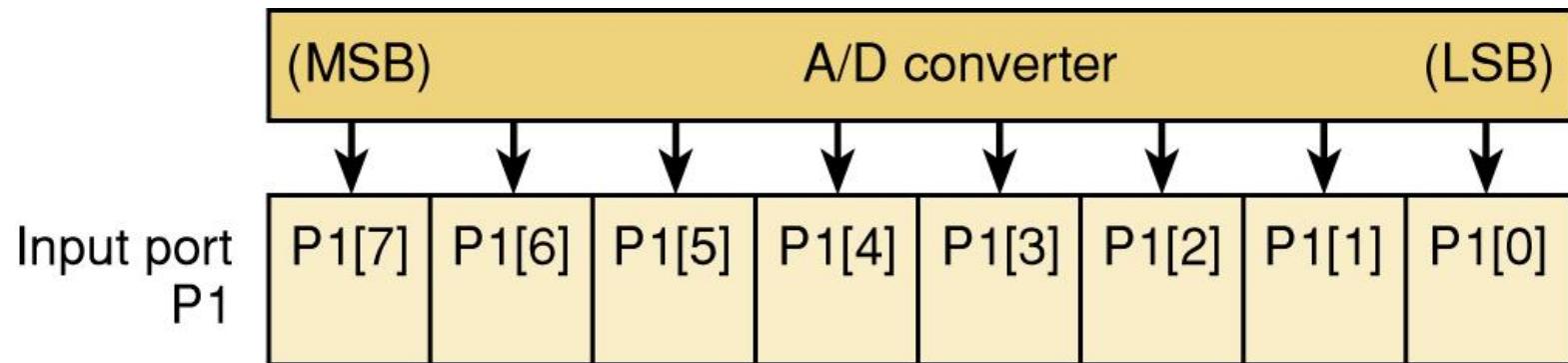
4-15 Representing Data in HDL

- These numeric values are referred to as scalars or literals.

| Number System | AHDL | VHDL | Bit Pattern | Decimal Equivalent |
|---------------|--------|--------|-------------|--------------------|
| Binary | B"101" | B"101" | 101 | 5 |
| Hexadecimal | H"101" | X"101" | 10000001 | 257 |
| Decimal | 101 | 101 | 1100101 | 101 |

- In order to describe a port with more than one data bit we assign a name and the number of bits. This is called a bit array or bit vector.

4-15 Representing Data in HDL



- Each element (bit) has a unique **index** number (0–7) to describe position in the overall structure.
 - HDLs & computer programming languages use this notation.

4-15 Representing Data in HDL

- AHDL syntax—a name for the bit vector is followed by the range of index designations.
 - Inside square brackets, in the SUBDESIGN section.

4-15 Representing Data in HDL – AHDL Syntax

- Intermediate variables can be declared as an array of bits.

Eight-bit temperature port *p1* assigned to a node named *temp*...

```
VARIABLE temp[7..0] :NODE;  
BEGIN  
    temp[] = p1[]  
END;
```

The empty braces mean that all bits in the array are being connected. Individual bits could be connected by specifying the bits inside the braces.

4-15 Representing Data in HDL – VHDL Syntax

- VHDL syntax—a name for the bit vector is followed by the mode, the type, and the range.
 - Enclosed in parenthesis, in the ENTITY section.

4-15 Representing Data in HDL – VHDL Syntax

- Intermediate variables can be declared as an array of bits—in the ARCHITECTURE section

Eight-bit temperature port *p1* assigned to a signal named *temp*...

```
SIGNAL          temp : BIT_VECTOR { 7 DOWNTO 0 } ;
BEGIN
    temp <= p1;
END ;
```

**When no elements in the bit vector are specified,
all bits in the array are being connected.
Individual bits could be connected by specifying
bit numbers inside the parentheses.**

4-15 Representing Data in HDL

- VHDL offers some standardized data types in libraries—collections of VHDL code that can be used to avoid reinventing the wheel.

| Data Type | Sample Declaration | Possible Values | Use |
|------------------|---|--------------------------------|-------------------------------------|
| BIT | y :OUT BIT; | '0' '1' | y <= '0'; |
| STD_LOGIC | driver :STD_LOGIC | '0' '1' 'z' 'x' 'u' | driver <= 'z'; |
| BIT_VECTOR | bcd_data :BIT_VECTOR (3 DOWNTO 0); | "0101" "1001" "0000" | digit <= bcd_data; |
| STD_LOGIC_VECTOR | dbus :STD_LOGIC_VECTOR (3 DOWNTO 0); | "0Z1X" | IF rd = '0' THEN dbus <= "zzzz"; |
| INTEGER | SIGNAL z:INTEGER RANGE -32 TO 31; | -32.. -2, -1, 0, 1, 2 . . . 31 | IF z > 5 THEN . . . |

4-16 Truth Tables Using HDL - AHDL

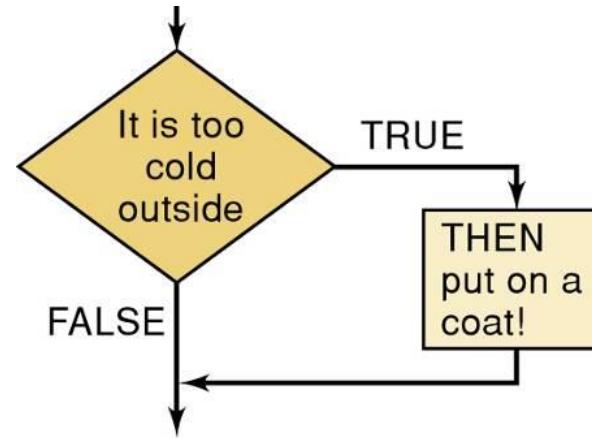
- Circuits can be designed directly from truth tables using AHDL and VHDL.

4-16 Truth Tables Using HDL - VHDL

- Circuits can be designed directly from truth tables using AHDL and VHDL.

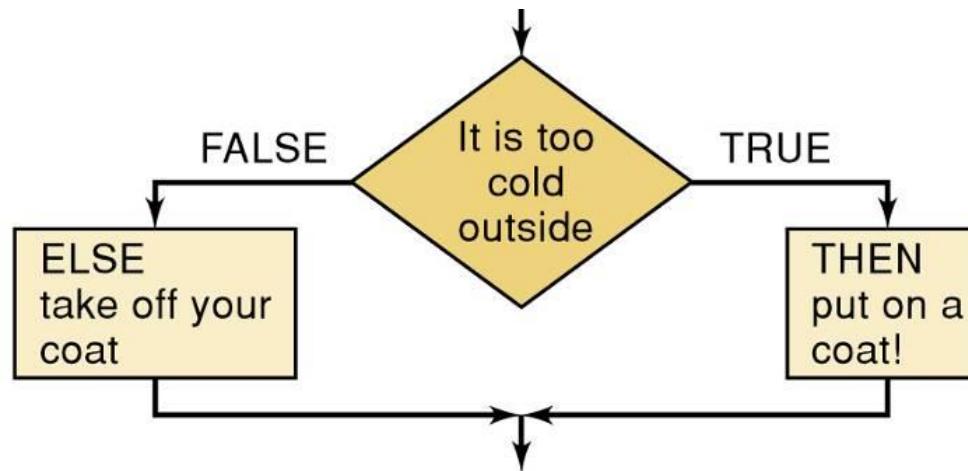
4-17 Decision Control Structures in HDL – IF/THEN/ELSE

- IF/THEN/ELSE statements provide a framework for making logical decisions in a system
 - IF/THEN is used when there is a choice between doing something and doing nothing.



4-17 Decision Control Structures in HDL – IF/THEN/ELSE

- IF/THEN/ELSE statements provide a framework for making logical decisions in a system
 - IF/THEN/ELSE is used when there is a choice of two possible actions.



4-17 Decision Control Structures in HDL – IF/THEN/ELSE

- IF/THEN/ELSE in AHDL:

```
SUBDESIGN FIG4_54
(
    digital_value[3..0] :INPUT; -- define inputs to block
    z                   :OUTPUT; -- define block output
)
BEGIN
    IF digital_value[] > 6 THEN
        z = VCC;           -- output a 1
    ELSE   z = GND;         -- output a 0
    END IF;
END;
```

4-17 Decision Control Structures in HDL – IF/THEN/ELSE

- IF/THEN/ELSE in VHDL:

```
ENTITY fig4_55 IS
PORT( digital_value :IN INTEGER RANGE 0 TO 15; -- 4-bit input
      z          :OUT BIT);
END fig4_55;

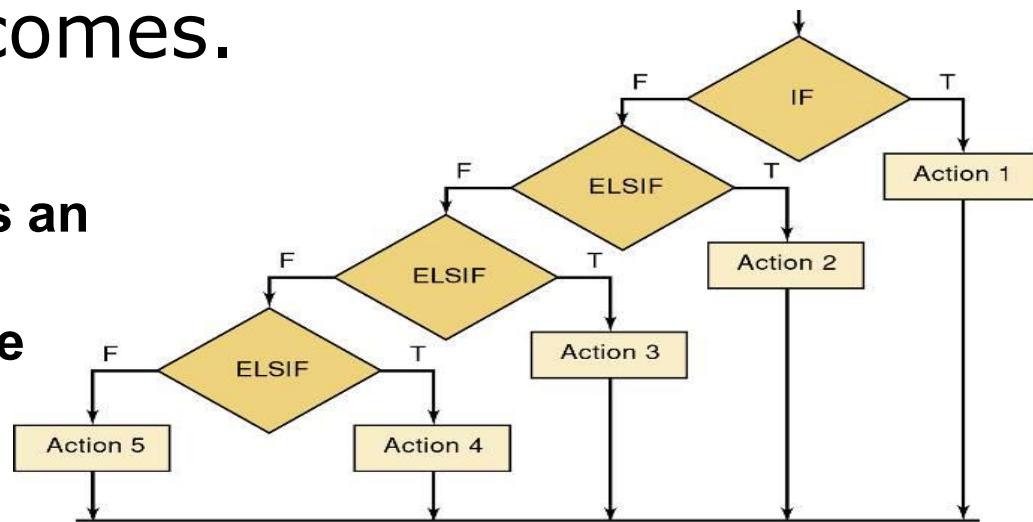
ARCHITECTURE truth OF fig4_55 IS

BEGIN
  PROCESS (digital_value)
  BEGIN
    IF (digital_value > 6) THEN
      z <= '1';
    ELSE
      z <= '0';
    END IF;
  END PROCESS ;
END truth;
```

4-17 Decision Control Structures in HDL – ELSIF

- By combining IF and ELSE decisions, we can create a control structure referred to as ELSIF
 - Which chooses one of many possible outcomes.

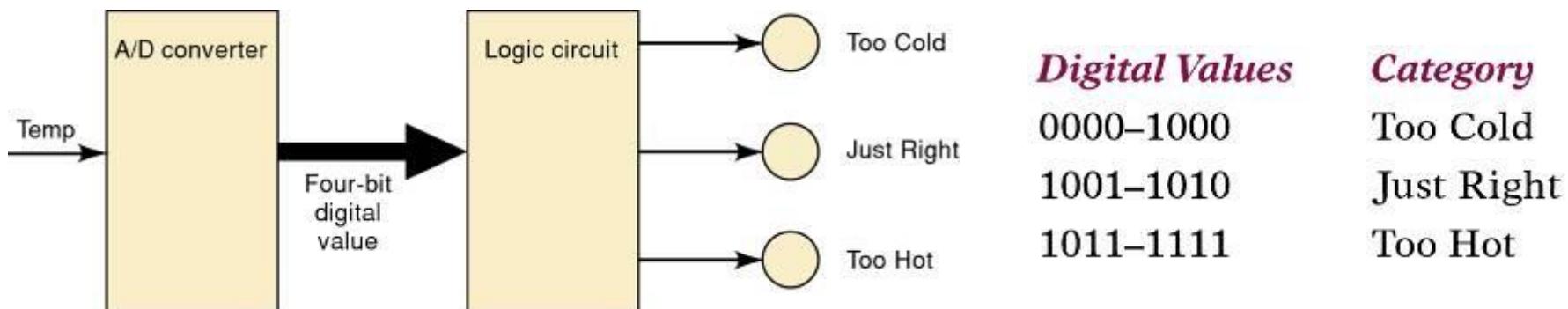
As each condition is evaluated, it performs an action—if true—or goes on to evaluate the next condition.



4-17 Decision Control Structures in HDL – ELSIF

- IF the digital value is less than or equal to 8...THEN light only the Too Cold indicator.

A temperature measuring system
using an A/D converter.



ELSE IF the digital value is greater than 8 **AND** less than 11...
THEN light only the **Just Right** indicator.
ELSE light only the **Too Hot** indicator.

4-17 Decision Control Structures in HDL – ELSIF

- ELSIF in AHDL:

```
SUBDESIGN fig4_58
(
    digital_value[3..0]           :INPUT; --define inputs to block
    too_cold, just_right, too_hot :OUTPUT;--define outputs
)
VARIABLE
status[2..0]      :NODE;--holds state of too_cold, just_right, too_hot
BEGIN
    IF      digital_value[] <= 8 THEN status[] = b"100";
    ELSIF   digital_value[] > 8 AND digital_value[] < 11 THEN
        status[] = b"010";
    ELSE    status[] = b"001";
    END IF;
    (too_cold, just_right, too_hot) = status[]; -- update output bits
END;
```

4-17 Decision Control Structures in HDL – ELSIF

- ELSIF in VHDL:

```
ENTITY fig4_59 IS
PORT(digital_value:IN INTEGER RANGE 0 TO 15;      -- declare 4-bit input
      too_cold, just_right, too_hot :OUT BIT);
END fig4_59 ;

ARCHITECTURE howhot OF fig4_59 IS
SIGNAL status    :BIT_VECTOR (2 downto 0);
BEGIN
  PROCESS (digital_value)
  BEGIN
    IF (digital_value <= 8) THEN status <= "100";
    ELSIF (digital_value > 8 AND digital_value < 11) THEN
      status <= "010";
    ELSE   status <= "001";
    END IF;
  END PROCESS ;
  too_cold    <= status(2);      -- assign status bits to output
  just_right  <= status(1);
  too_hot     <= status(0);
END howhot;
```

4-17 Decision Control Structures in HDL - CASE

- The CASE construct determines the value of an expression or object.
 - Then goes through a list of values (cases) to determine what action to take.
- Different than the IF/ELSEIF, as there is only one action or match for a case statement.

4-17 Decision Control Structures in HDL - CASE

- CASE construct in AHDL:

```
SUBDESIGN fig4_60
(
    p, q, r      :INPUT;      -- define inputs to block
    s            :OUTPUT;     -- define outputs
)
VARIABLE
    status[2..0]   :NODE;
BEGIN
    status[] = (p, q, r);  -- link input bits in order
    CASE status[] IS
        WHEN b"100"      => s = GND;
        WHEN b"101"      => s = GND;
        WHEN b"110"      => s = GND;
        WHEN OTHERS      => s = VCC;
    END CASE;
END;
```

4-17 Decision Control Structures in HDL - CASE

- CASE construct in VHDL:

```
ENTITY fig4_61 IS
PORT( p, q, r      :IN bit;           --declare 3 bits input
      s          :OUT BIT);
END fig4_61;

ARCHITECTURE copy OF fig4_61 IS
SIGNAL status      :BIT_VECTOR (2 downto 0);
BEGIN
    status <= p & q & r;           --link bits in order.
    PROCESS (status)
        BEGIN
            CASE status IS
                WHEN "100" => s <= '0';
                WHEN "101" => s <= '0';
                WHEN "110" => s <= '0';
                WHEN OTHERS => s <= '1';
            END CASE;
        END PROCESS ;
    END copy;
```