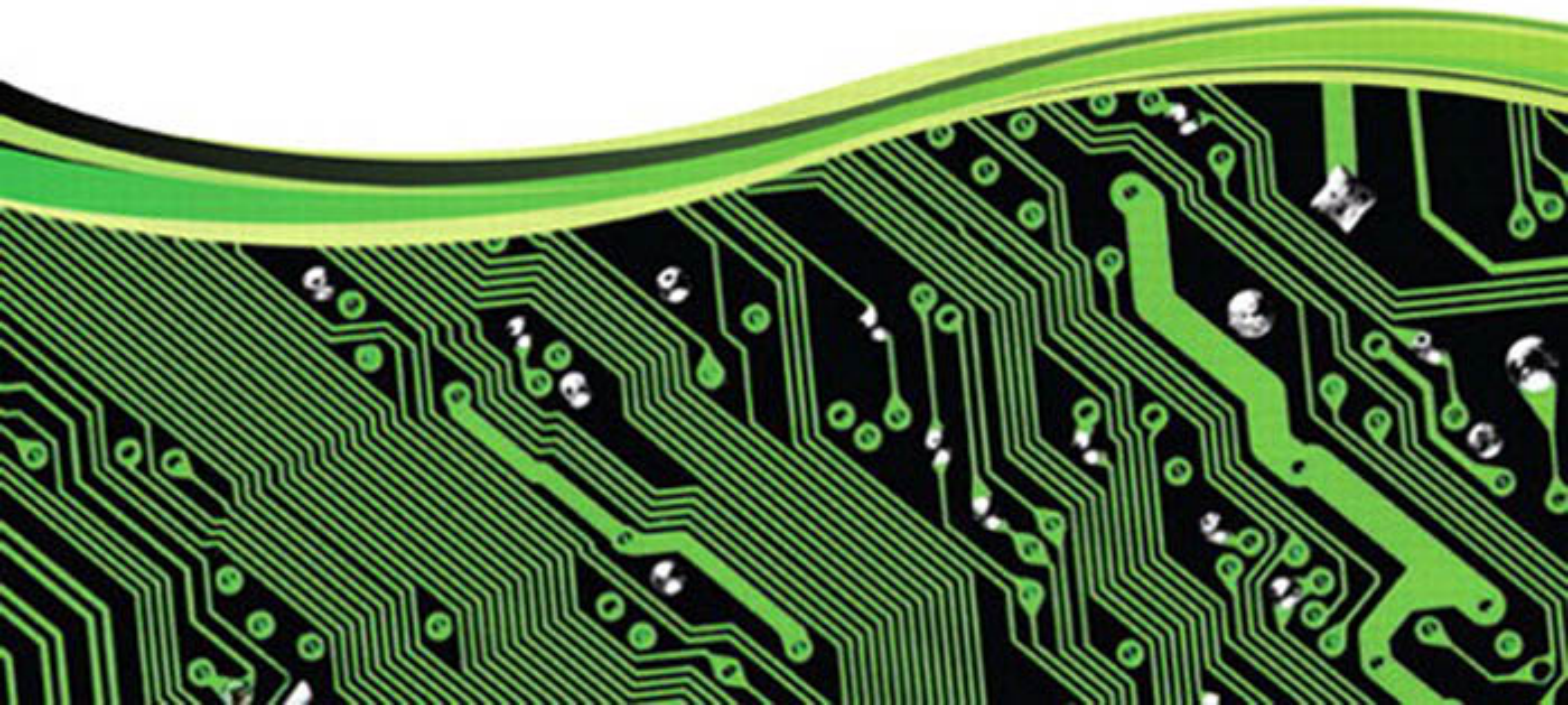
Digital Fundamentals

A SYSTEMS APPROACH

THOMAS L. FLOYD



DIGITAL FUNDAMENTALS

A SYSTEMS APPROACH

THOMAS L. FLOYD

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director: Vernon R. Anthony
Senior Acquisitions Editor: Lindsey Prudhomme
Development Editor: Dan Trudden
Editorial Assistant: Yvette Schlarman
Director of Marketing: David Gesell
Marketing Manager: Harper Coles
Senior Marketing Coordinator: Alicia Wozniak
Senior Marketing Assistant: Les Roberts
Senior Managing Editor: JoEllen Gohr
Senior Project Manager: Rex Davidson
Senior Operations Supervisor: Pat Tonneman
Creative Director: Andrea Nix
Art Director: Diane Y. Ernsberger
Text and Cover Designer: Candace Rowley
Cover Image: Kayros Studio "Be Happy!"/Shutterstock.com
Media Project Manager: Karen Bretz
Full-Service Project Management: Penny Walker/Aptara®, Inc.
Composition: Aptara®, Inc.
Printer/Binder: R. R. Donnelley & Sons
Cover Printer: Lehigh/Phoenix Color Hagerstown
Text Font: Times Roman

Credits and acknowledgments for materials borrowed from other sources and reproduced, with permission, in this textbook appear on the appropriate page within text.

Library of Congress Cataloging-in-Publication Data

Floyd, Thomas L.
Digital fundamentals: a systems approach / Thomas L. Floyd.
p. cm.
ISBN-13: 978-0-13-293395-7
ISBN-10: 0-13-293395-0
1. Digital electronics. I. Title.
TK7868.D5F534 2013
621.39'5—dc23

2012020323

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN 10: 0-13-293395-0
ISBN 13: 978-0-13-293395-7

CONTENTS

PREFACE

This first edition of *Digital Fundamentals: A Systems Approach* provides a unique coverage of digital technology with a system emphasis. This textbook provides a fundamental grounding in the basic concepts of digital technology and systems reinforced by an abundance of illustrations, examples, applications, and exercises. There are system examples and system notes throughout many chapters in addition to traditional worked examples. Many chapters have a system section that presents a certain type of system and discusses its operation as related to topics covered in that chapter and to its implementation in programmable logic. Most chapters include a troubleshooting section that emphasizes the system approach. Additionally, system level chapters cover digital data transmission; data processing and control; and buses, networks, and interfacing.

Core fundamentals and basic logic functions are presented using a practical approach with emphasis on operation and application rather than on analysis and design. Mathematical topics are limited to only essential coverage that a technician or technologist will need to understand the basic concepts. Programmable logic is emphasized whereas fixed-function logic is introduced on a limited basis.

Features

- Core fundamentals are presented without being intermingled with more advanced topics.
- Many chapters feature an entire section devoted to a specific type of system.
- System examples are used to illustrate how basic concepts and logic elements are applied in a system application.
- System notes present interesting facts and information about system-related issues.
- Multisim is used in selected examples, figures, and problems to provide practice in simulating logic circuits and systems and in troubleshooting.
- Worked examples illustrate core fundamentals and logic functions and require some basic analytical thought.
- Related problems in each worked example relate to the coverage of the example.
- Hands-on-tips (HOT) provide useful and practical information.
- Many chapters include a section on hardware description languages (VHDL and Verilog), which are used to show how logic functions and systems can be described and implemented in a programmable logic device (PLD).
- Many chapters have a troubleshooting section that relates to topics covered in the chapter and emphasizes troubleshooting techniques, and the use of instrumentation, and circuit simulation (Multisim).
- Each chapter begins with a list of sections (Outline), chapter objectives, introduction, key terms list, and website reference.
- Each section within a chapter begins with an introduction and objectives.
- Each section concludes with Checkup exercises that emphasize the main concepts presented in the section.
- Each chapter ends with a summary, key term glossary, true/false quiz, self-test, and sectionalized problem set.
- Answers to related problems, section checkups, true/false quiz, and self-test are at the end of each chapter.
- An end-of-book glossary contains all bold and color terms in the text.

- Answers to odd-numbered problems are at the end of the book
- Website (www.pearsonhighered.com) includes files related to the text such as tutorials and Multisim files.

Student Resources

- *Experiments in Digital Fundamentals: A Systems Approach* (ISBN 0132989840) by David Buchla and Doug Joksch. Lab exercises are coordinated with the text.
- *Multisim Experiments for the DC/AC, Digital, and Devices Courses* (ISBN 0132113880) by Gary Snyder and David Buchla. Students take data, analyze results, and write a conclusion to simulate an actual laboratory experience.
- **Multisim Files Available on the Website** Circuit files coordinated with this text in Versions 11 and 12 of Multisim are available for download from www.pearsonhighered.com/floyd. Circuit files with prefix F are figure circuits and files with prefix P are problem circuits. Also, a few files are prefixed with E or T, represesting examples or tables.

In order to use the Multisim circuit files, you must have Multisim software installed on your computer. Multisim software is available at **www.ni.com/Multisim**. Although the Multisim circuit files are intended to complement classroom, textbook, and laboratory study, these files are not essential to successfully using this text.

Instructor Resources

Instructor resources are available from Pearson's **Instructor's Resource Center**.

- PowerPoint® slides (ISBN 013298962x) support the topics in each chapter.
- Instructor's Resource Manual (ISBN 0132989832) contains the solutions to the text problems and the solutions to the lab manual.
- TestGen (ISBN 0132988615) This electronic bank of test questions can be used to develop customized quizzes, tests, and/or exams.

To access supplementary materials online, instructors need to request an instructor access code. Go to **www.pearsonhighered.com/irc**, where you can register for an instructor access code. Within 48 hours after registering, you will receive a confirming e-mail, including an instructor access code. Once you have received your code, go to the site and log on for full instructions on downloading the materials you wish to use.

Illustrations of Textbook Features

Chapter Opener A typical chapter opener is shown in Figure P–1.

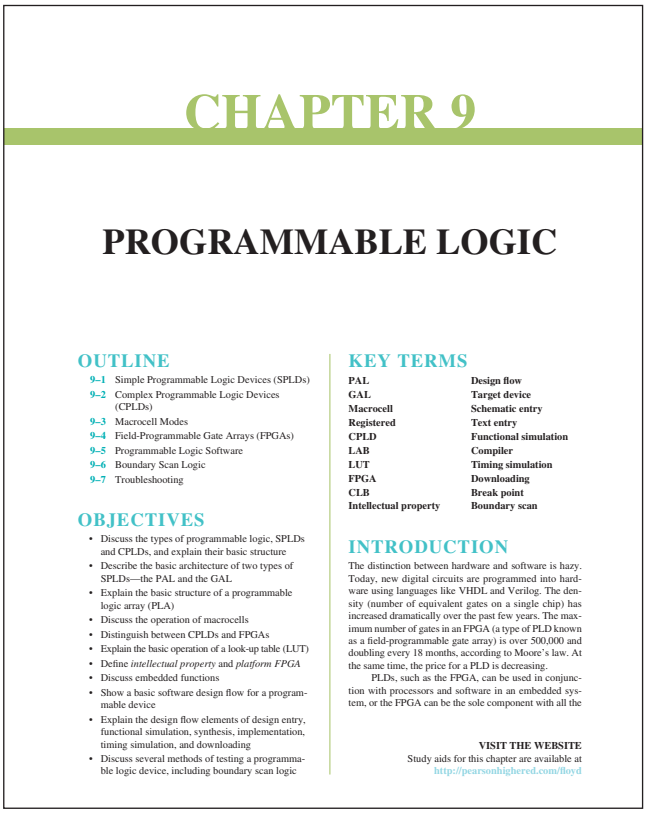
Worked Example and Related Problem Worked-out examples illustrate basic concepts or specific procedures. A Related Problem reinforces or expands on the content of the example. A typical worked-out example with a Related Problem is shown in Figure P–2.

Section Opener Each section in a chapter begins with a brief introduction that includes a general overview and section objectives, as shown in Figure P–2.

Section Checkup A typical Section Checkup is shown in Figure P–2. (Answers to the Section Checkups are at the end of the chapter.)

Hands On Tip A typical Hands On Tip is shown in Figure P–2.

System Section Typical pages from a System Section are shown in Figure P–3.



CHAPTER 9

PROGRAMMABLE LOGIC

OUTLINE

OBJECTIVES

- Discuss the types of programmable logic, SPLDs and CPLDs, and explain their basic structure
- Describe the basic architecture of two types of SPLDs—the PAL and the GAL
- Explain the basic structure of a programmable logic array (PLA)
- Discuss the operation of macrocells
- Distinguish between CPLDs and FPGAs
- Explain the basic operation of a look-up table (LUT)
- Define *intellectual property* and *platform FPGA*
- Discuss embedded functions
- Show a basic software design flow for a programmable device
- Explain the design flow elements of design entry, functional simulation, synthesis, implementation, timing simulation, and downloading
- Discuss several methods of testing a programmable logic device, including boundary scan logic

KEY TERMS

PAL
GAL
Macrocell
Registered
CPLD
LAB
LUT
FPGA
CLB
Intellectual property
Design flow
Target device
Schematic entry
Text entry
Functional simulation
Compiler
Timing simulation
Downloading
Break point
Boundary scan

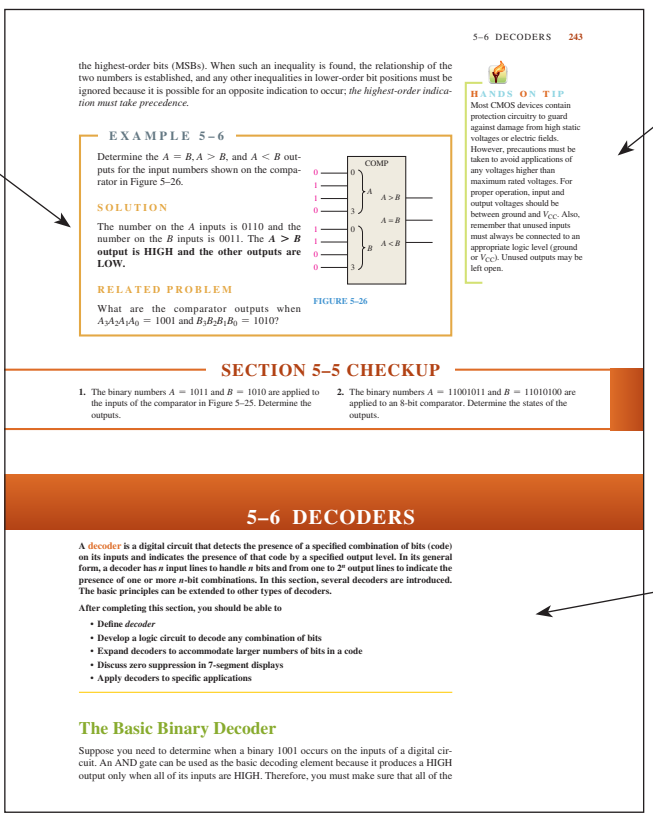
INTRODUCTION

The distinction between hardware and software is hazy. Today, new digital circuits are programmed into hardware using languages like VHDL and Verilog. The density (number of equivalent gates on a single chip) has increased dramatically over the past few years. The maximum number of gates in an FPGA (a type of PLD known as a field-programmable gate array) is over 500,000 and doubling every 18 months, according to Moore's law. At the same time, the price for a PLD is decreasing.

PLDs, such as the FPGA, can be used in conjunction with processors and software in an embedded system, or the FPGA can be the sole component with all the

VISIT THE WEBSITE
Study aids for this chapter are available at
http://pearsonhighered.com/floyd

FIGURE P–1 Chapter opener.



Worked Example with Related Problem

Hands On Tip

Section Checkup

Section Opener

5–6 DECODERS 243

the highest-order bits (MSBs). When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored because it is possible for an opposite indication to occur; *the highest-order indication must take precedence.*

EXAMPLE 5–6

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 5–26.

SOLUTION

The number on the A inputs is 0110 and the number on the B inputs is 0011. The $A > B$ **output is HIGH and the other outputs are LOW.**

RELATED PROBLEM

What are the comparator outputs when $A_3A_2A_1A_0 = 1001$ and $B_3B_2B_1B_0 = 1010$?

FIGURE 5–26

HANDS ON TIP
Most CMOS devices contain protection circuitry to guard against damage from high static voltages or electric fields. However, precautions must be taken to avoid applications of any voltages higher than maximum rated voltages. For proper operation, input and output voltages should be between ground and V_{CC}. Also, remember that unused inputs must always be connected to an appropriate logic level (ground or V_{CC}). Unused outputs may be left open.

SECTION 5–5 CHECKUP

1. The binary numbers $A = 1011$ and $B = 1010$ are applied to the inputs of the comparator in Figure 5–25. Determine the outputs.
2. The binary numbers $A = 11001011$ and $B = 11010100$ are applied to an 8-bit comparator. Determine the states of the outputs.

5–6 DECODERS

A decoder is a digital circuit that detects the presence of a specified combination of bits (code) on its inputs and indicates the presence of that code by a specified output level. In its general form, a decoder has *n* input lines to handle *n* bits and from one to 2^n output lines to indicate the presence of one or more *n*-bit combinations. In this section, several decoders are introduced. The basic principles can be extended to other types of decoders.

After completing this section, you should be able to

- **Define *decoder***
- **Develop a logic circuit to decode any combination of bits**
- **Expand decoders to accommodate larger numbers of bits in a code**
- **Discuss zero suppression in 7-segment displays**
- **Apply decoders to specific applications**

The Basic Binary Decoder

Suppose you need to determine when a binary 1001 occurs on the inputs of a digital circuit. An AND gate can be used as the basic decoding element because it produces a HIGH output only when all of its inputs are HIGH. Therefore, you must make sure that all of the

FIGURE P–2 Worked example with Related Problem, Hands On Tip, Section Checkup, and Section Opener.

12–1 A SYSTEM

In this section, a system for tracking the movement of the sun in order to control the position of a solar panel, Fresnel mirror array, or other type of solar collection device is discussed. The system senses the orientation of the solar panel relative to the sun's position in the sky and produces a signal that causes the solar panel to rotate until it is directly aligned with the sun. Analog-to-digital converters (ADCs), which are covered later in the chapter, are an important part of this system.

After completing this section, you should be able to

- **Describe the overall system**
- **Discuss how ADCs are used in the system**
- **Use VHDL to implement a portion of the system**

The Sunlight Sensors

The sun tracker system tracks the movement of the sun through the day using photosensitive devices. As illustrated in Figure 12–1, the sunlight sensor consists of two photoresistors separated by a plate that optically isolates the photoresistors from each other. The resistance of a photoresistor decreases as the light intensity increases. Other light-sensitive devices, such as the photodiode and phototransistor, could also be used. The sensor assembly is attached to the solar panel so that it is always pointing in the same direction as the panel. When the sensor arrangement and solar panel are not pointing directly at the sun, the photoresistor R_2 gets more sunlight than R_1, as illustrated in part (a) of the figure. When the sensors and solar panel are pointing directly at the sun, R_1 and R_2 get the same amount of sunlight, as shown in part (b). The voltages across the sensors depend on the orientation of the panel. After being processed through the system, the sensor voltages cause the panel to rotate to follow the sun's movement, as shown in Figure 12–2. When the solar panel and sensor arrangement are pointed directly at the sun, the light levels for both photoresistors are balanced. The solar panel does not rotate until the sun's position changes by a certain amount, and the photoresistors begin to receive different amounts of sunlight.

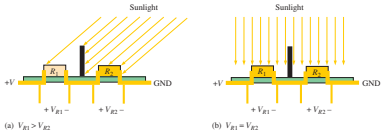


FIGURE 12–1 Sunlight sensor arrangement. The orientation produces voltages proportional to the incident sunlight.

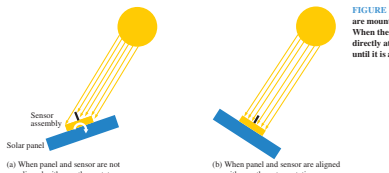


FIGURE 12–2 The sensors are mounted to the solar panel. When the panel is not pointing directly at the sun, it rotates until it is aligned.

The Complete System

The complete sun tracker system is shown in block diagram form in Figure 12–3. The system consists of the sunlight sensor, analog-to-digital converters, frequency divider, analog-to-digital processor, motor control, motor interface, and stepper motor. If the resistance of R_1 in Figure 12–1 is less than that of R_2, V_{R2} is less than V_{R1}, and the resulting direction signal (acting through the A/D processing block) causes the motor to turn the solar panel clockwise. If R_1 and R_2 are equal, the voltages are equal and the motor stops because the panel is directly aligned with the sun. The frequency divider divides the system clock down to a frequency compatible with the rate at which the stepper motor can be advanced.

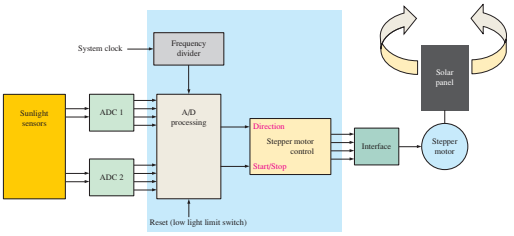


FIGURE 12–3 The complete sun tracker system block diagram.

The A/D processing block receives the digitized light level inputs from the ADCs and compares the two values to determine the required direction of rotation. If the ADC1 output is greater than the ADC2 output ($V_{R1} > V_{R2}$), a start bit is issued, and the motor control initiates a bit sequence to produce the motor stepping action, which causes the solar panel to rotate clockwise. If the ADC outputs are equal ($V_{R1} = V_{R2}$), the solar panel will stop because it is pointing directly at the sun. As the sun moves further west, the system follows until the light level of both sensors decreases below a certain threshold value, and a reset signal is generated. At that point, the panel has reached its western-most position and the sunlight is greatly diminished. The motor control reverses the bit sequence to the stepper motor, and the solar panel is rotated counterclockwise back to its eastern-most position. When the sun rises the next day, the cycle repeats.

Stepper Motor Operation

The interface between the stepper motor control and the motor converts low-level currents produced by the motor control to currents that are sufficient to drive the motor windings. The stepper motor divides a single rotation of its output shaft into a fixed number of steps, as shown in Figure 12–4. Pulses (1s) are applied sequentially to the four motor windings to drive the stepping operation. For each pulse, the motor advances a set number of degrees, D°, clockwise. After a full cycle of four pulses, the motor has

FIGURE P–3 Partial System Section.

System Example A typical System Example is shown in Figure P–4.

The CAN Bus

The **controller area network (CAN)** bus, a differential serial bus, was developed for automotive applications and is also commonly used in aerospace systems as well as other applications. The bus consists of a terminated twisted pair of signal lines, called CAN H and CAN L, plus ground. Vehicles sold in the United States are required by the SAE (Society of Automotive Engineers) to use the CAN bus protocol. The European Union has similar requirements.

Devices, called *nodes*, can be connected to the bus but are not assigned specific addresses as in the I^2C bus. There are two CAN specifications in use. The standard or basic CAN 2.0A has 11-bit message identifiers and can operate up to 250 kbps, and the full CAN has 29-bit message identifiers and can be used up to 1 Mbps. The message identifier is a label for the contents of a message and goes to each node on the bus. Each receiving node performs a test on the identifier to determine if it is relevant to that node and is used to arbitrate the bus to determine if the message is of highest priority. All of the nodes on the bus can transmit and receive messages. The bus is available to a node with a message with the highest priority (dominant) and can override a message with lower priority (recessive). When the dominant message has been processed, the recessive message is retransmitted.

DATA FRAME The standard data frame is shown in Figure 14–36. Data is transmitted in NRZ format. The frame begins with a start-of-frame (SOF) bit followed by an arbitration field and a control field. The arbitration field contains the message identifier and a remote transmission request (RTR) bit. The control field has two reserve bits and a data length code (DLC) that specifies the length of the data field that follows and can contain up to eight bytes. The cyclic redundancy check (CRC) field provides for error detection. The acknowledge (ACK) verifies the receipt of correct data, and the frame ends with the end-of-frame field (EOF).

SOF	Arbitration field	Control field	Data field	CRC field	ACK	EOF
(1 bit)	Identifier (11bits) RTR (1 bit)	Reserve (2 bits) DLC (4 bits)	(0–8 bytes)	(16 bits)	(2 bits)	(7 bits)

FIGURE 14–36 Standard CAN data frame format.

SYSTEM EXAMPLE 14–3

AN AUTOMOBILE CONTROL SYSTEM

A modern automobile typically has many control units (usually several dozen) for various subsystems, which include the engine control unit and other control units for transmission; ABS; cruise control; power steering; audio system; window, door, and mirror controls; airbags; and others. Figure 14–37 is a block diagram of a partial automotive control system using two CAN buses, one low-speed and one high-speed to control various functions throughout the vehicle.

Each unit connected to the bus contains sensors and other functions that allow it to carry out its unique purpose. For example, the ABS (antilock braking system) can receive a message from sensors in each wheel, indicating that the brake is about to lock up. A sudden and rapid deceleration in the wheel indicates an imminent lock-up condition. The ABS unit then sends a message that causes the valve in the brake line to release pressure to allow acceleration. Then, when acceleration is sensed, the unit causes a pump to restore the pressure. A rapid release-and-restore cycle occurs until the brakes are brought under control. A pulsing of the brake pedal can be felt when the operation occurs.

As another example, part of the engine control unit's operation is to sense parameters, such as engine temperature, oil pressure, fuel consumption, and rpm, and send messages to the driver unit. All of the units on the bus operate as a system to keep the vehicle running as smoothly and as safely as possible, while providing a comfortable environment for the driver and passengers.

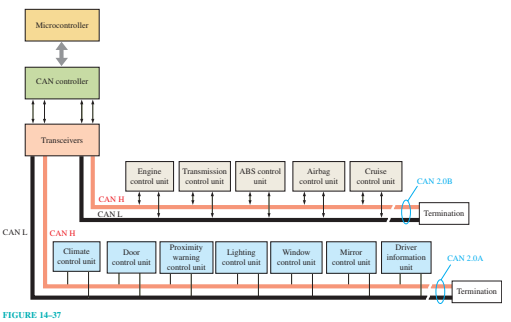


FIGURE 14–37

The Firewire Bus

Firewire, also known as IEEE-1394 and iLink, is a high-speed external serial bus developed by Apple Inc. Firewire is used in high-speed communications and real-time data transfer. It is used in professional audio and video equipment, camcorders, DVD players, external hard drives, computers, and in some auto and aircraft applications. It works very similar to the USB except that it has a higher data rate and can handle more data.

Three types of connectors are used in the Firewire standard: a 4-pin connector, a 6-pin connector, and a 9-pin connector. The cable for the 4-pin connector consists of two twisted pairs that carry data. The cable for the 6-pin connector has the two twisted pairs for data plus a power line and a ground line. The cable for the 9-pin connector has the same wires as the 6-pin configuration plus two wires that provide for a grounded shield and one wire that is currently unused. The Firewire symbol is shown in Figure 14–38(a). End views of the three connector types are shown in part (b), and the pin designations are shown in part (c).

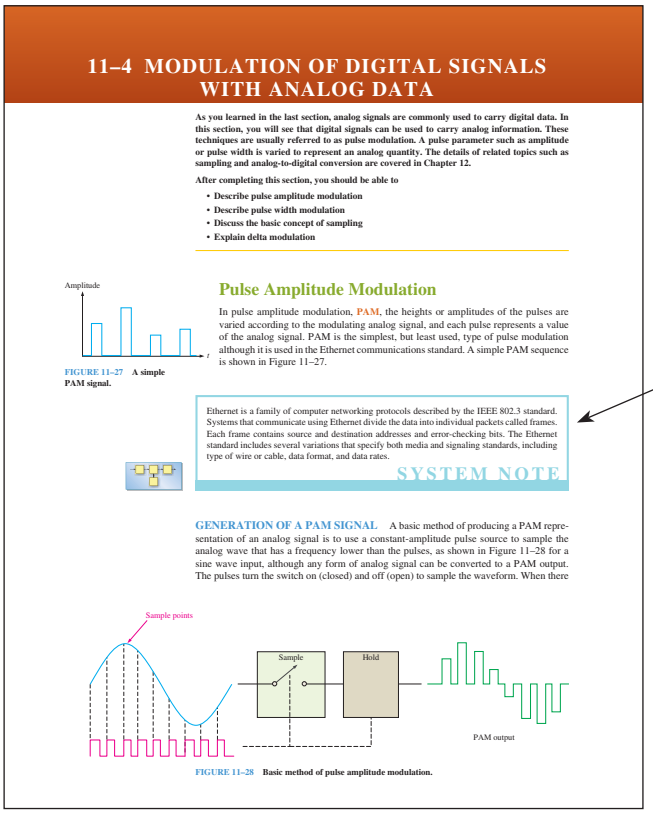
The Firewire bus address has a total of 64 bits. Ten are for bus ID, six are for node ID, and 48 are for individual addresses. This allows up to 1023 buses each having up to 63 nodes. There are six transfer modes in the IEEE-1394 standard and its revisions: S100, S200, S400, S800, S1600, and S3200. The S100 is the base rate of 98.304 Mbps. The S200 is twice the base rate at 196.608 Mbps, and the S400 is four times the base rate at 393.216 Mbps. The S800 is 786.432 Mbps, and the S1600 and S3200 are 16 and 32 times the base rate respectively (1.6 Gbps and 3.2 Gbps). Firewire cable length cannot exceed 15 ft (4.572 m). To increase this length, up to 16 cables can be connected together.

FIREWIRE VERSUS USB In general, any capable node can control the bus in a Firewire system, but a single host is used to control the bus in USB. USB networks use a tiered-star topology, and Firewire uses a tree topology. (Network topologies are covered in section 14–6.) A Firewire device can communicate with any node at any time if the

FIGURE P–4 System Example.

System Notes A typical System Note is shown in Figure P–5.

Programmable Logic Coverage The hardware description languages VHDL and Verilog are used in programmable logic applications. Figure P–6 shows two typical pages.



11–4 MODULATION OF DIGITAL SIGNALS WITH ANALOG DATA

As you learned in the last section, analog signals are commonly used to carry digital data. In this section, you will see that digital signals can be used to carry analog information. These techniques are usually referred to as pulse modulation. A pulse parameter such as amplitude or pulse width is varied to represent an analog quantity. The details of related topics such as sampling and analog-to-digital conversion are covered in Chapter 12.

After completing this section, you should be able to

- **Describe pulse amplitude modulation**
- **Describe pulse width modulation**
- **Discuss the basic concept of sampling**
- **Explain delta modulation**

Pulse Amplitude Modulation

In pulse amplitude modulation, **PAM**, the heights or amplitudes of the pulses are varied according to the modulating analog signal, and each pulse represents a value of the analog signal. PAM is the simplest, but least used, type of pulse modulation although it is used in the Ethernet communications standard. A simple PAM sequence is shown in Figure 11–27.

Amplitude

t

FIGURE 11–27 A simple PAM signal.

Ethernet is a family of computer networking protocols described by the IEEE 802.3 standard. Systems that communicate using Ethernet divide the data into individual packets called frames. Each frame contains source and destination addresses and error-checking bits. The Ethernet standard includes several variations that specify both media and signaling standards, including type of wire or cable, data format, and data rates.

SYSTEM NOTE

GENERATION OF A PAM SIGNAL A basic method of producing a PAM representation of an analog signal is to use a constant-amplitude pulse source to sample the analog wave that has a frequency lower than the pulses, as shown in Figure 11–28 for a sine wave input, although any form of analog signal can be converted to a PAM output. The pulses turn the switch on (closed) and off (open) to sample the waveform. When there

Sample points

Sample

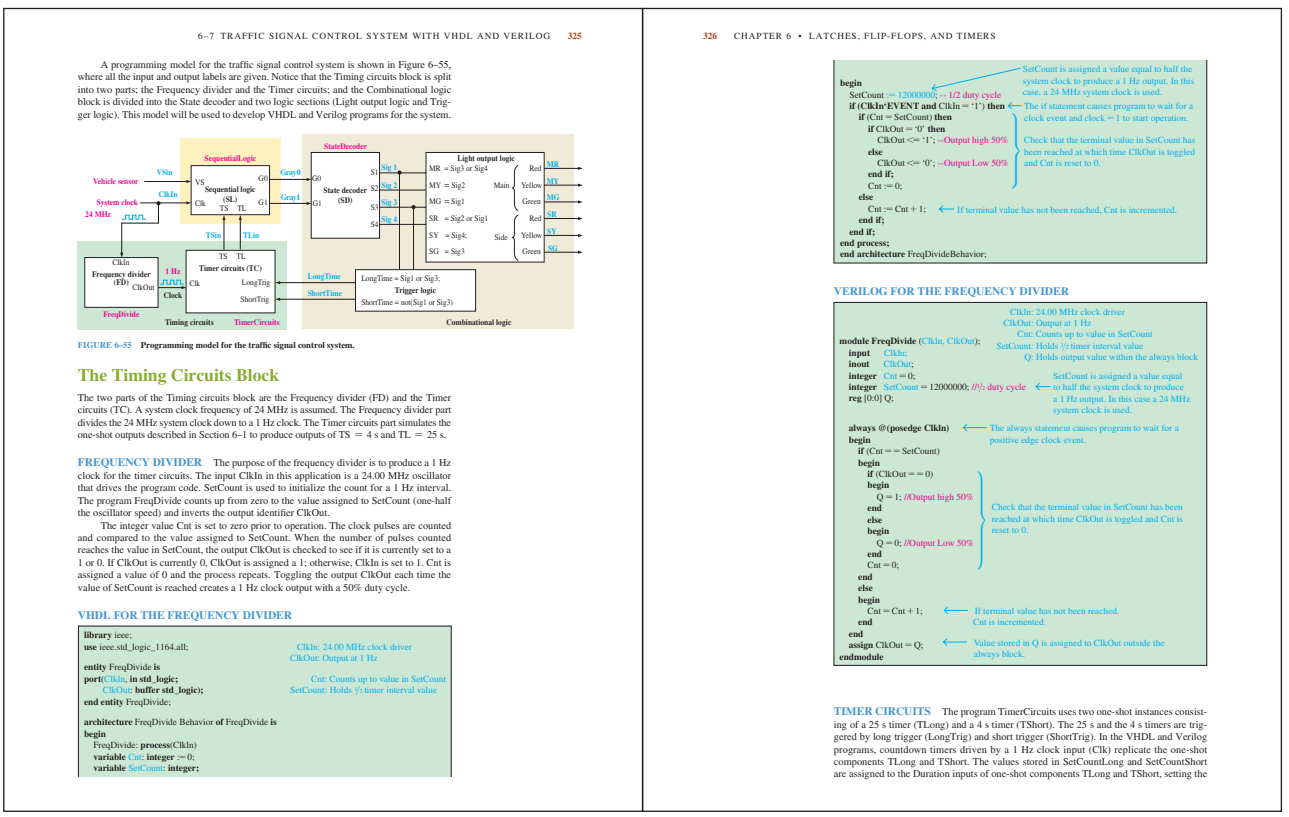
Hold

PAM output

FIGURE 11–28 Basic method of pulse amplitude modulation.

System Note

FIGURE P–5 System Note



6–7 TRAFFIC SIGNAL CONTROL SYSTEM WITH VHDL AND VERILOG 325

A programming model for the traffic signal control system is shown in Figure 6–55, where all the input and output labels are given. Notice that the Timing circuits block is split into two parts; the Frequency divider and the Timer circuits; and the Combinational logic block is divided into the State decoder and two logic sections (Light output logic and Trigger logic). This model will be used to develop VHDL and Verilog programs for the system.

SequentialLogic
StateDecoder
Vehicle sensor
VSin
System clock
24 MHz
ClkIn
Sequential logic (SL)
State decoder (SD)
Light output logic
Gray0
Gray1
TSin
TLin
Frequency divider (FD)
1 Hz Clock
Timer circuits (TC)
LongTime
ShortTime
Trigger logic
FreqDivide
Timing circuits
TimerCircuits
Combinational logic

FIGURE 6–55 Programming model for the traffic signal control system.

The Timing Circuits Block

The two parts of the Timing circuits block are the Frequency divider (FD) and the Timer circuits (TC). A system clock frequency of 24 MHz is assumed. The Frequency divider part divides the 24 MHz system clock down to a 1 Hz clock. The Timer circuits part simulates the one-shot outputs described in Section 6–1 to produce outputs of TS = 4 s and TL = 25 s.

FREQUENCY DIVIDER The purpose of the frequency divider is to produce a 1 Hz clock for the timer circuits. The input ClkIn in this application is a 24.00 MHz oscillator that drives the program code. SetCount is used to initialize the count for a 1 Hz interval. The program FreqDivide counts up from zero to the value assigned to SetCount (one-half the oscillator speed) and inverts the output identifier ClkOut.

The integer value Cnt is set to zero prior to operation. The clock pulses are counted and compared to the value assigned to SetCount. When the number of pulses counted reaches the value in SetCount, the output ClkOut is checked to see if it is currently set to a 1 or 0. If ClkOut is currently 0, ClkOut is assigned a 1; otherwise, ClkIn is set to 1. Cnt is assigned a value of 0 and the process repeats. Toggling the output ClkOut each time the value of SetCount is reached creates a 1 Hz clock output with a 50% duty cycle.

VHDL FOR THE FREQUENCY DIVIDER

```
library ieee;
use ieee.std_logic_1164.all;                    ClkIn: 24.00 MHz clock driver
                                                ClkOut: Output at 1 Hz
entity FreqDivide is
port(ClkIn, in std_logic;                       Cnt: Counts up to value in SetCount
     ClkOut: buffer std_logic);                 SetCount: Holds ½ timer interval value
end entity FreqDivide;

architecture FreqDivide Behavior of FreqDivide is
begin
   FreqDivide: process(ClkIn)
     variable Cnt: integer := 0;
     variable SetCount: integer;
```

326 CHAPTER 6 • LATCHES, FLIP-FLOPS, AND TIMERS

```
begin                                           SetCount is assigned a value equal to half the
  SetCount := 12000000; -- 1/2 duty cycle       system clock to produce a 1 Hz output. In this
  if (ClkIn'EVENT and ClkIn = '1') then         case, a 24 MHz system clock is used.
    if (Cnt = SetCount) then                    The if statement causes program to wait for a
      if ClkOut = '0' then                      clock event and clock = 1 to start operation.
        ClkOut <= '1'; --Output high 50%
      else                                      Check that the terminal value in SetCount has
        ClkOut <= '0'; --Output Low 50%         been reached at which time ClkOut is toggled
      end if;                                   and Cnt is reset to 0.
      Cnt := 0;
    else
      Cnt := Cnt + 1;                           If terminal value has not been reached, Cnt is incremented.
    end if;
  end if;
end process;
end architecture FreqDivideBehavior;
```

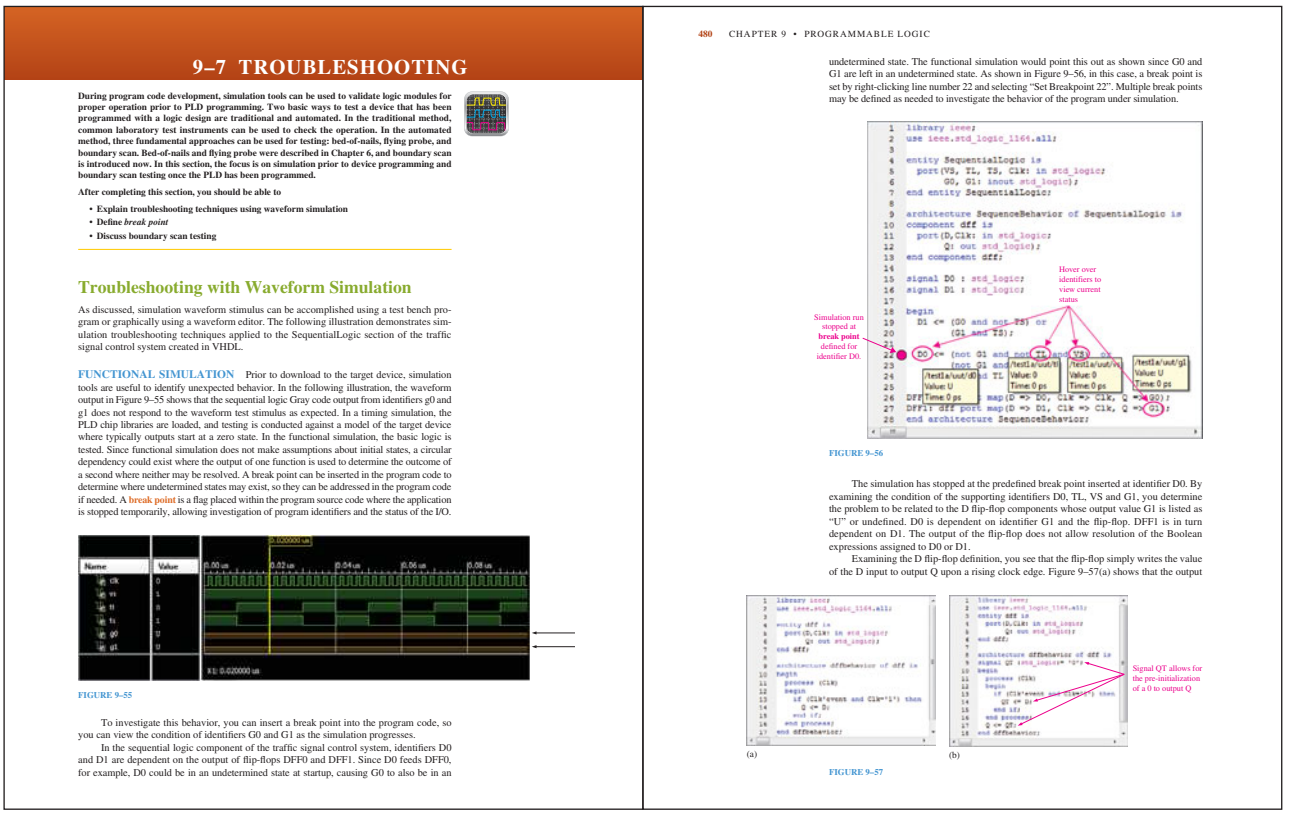
VERILOG FOR THE FREQUENCY DIVIDER

```
                                                ClkIn: 24.00 MHz clock driver
                                                ClkOut: Output at 1 Hz
                                                Cnt: Counts up to value in SetCount
module FreqDivide (ClkIn, ClkOut);              SetCount: Holds ½ timer interval value
  input   ClkIn;                                Q: Holds output value within the always block
  inout   ClkOut;
  integer Cnt = 0;                              SetCount is assigned a value equal
  integer SetCount = 12000000; //½ duty cycle   to half the system clock to produce
  reg [0:0] Q;                                  a 1 Hz output. In this case a 24 MHz
                                                system clock is used.
  always @(posedge ClkIn)                       The always statement causes program to wait for a
  begin                                         positive edge clock event.
    if (Cnt = = SetCount)
    begin
      if (ClkOut = = 0)
      begin                                     Check that the terminal value in SetCount has been
        Q = 1; //Output high 50%                reached at which time ClkOut is toggled and Cnt is
      end                                       reset to 0.
      else
      begin
        Q = 0; //Output Low 50%
      end
      Cnt = 0;
    end
    else
    begin
      Cnt = Cnt + 1;                            If terminal value has not been reached.
    end                                         Cnt is incremented.
  end
  assign ClkOut = Q;                            Value stored in Q is assigned to ClkOut outside the
endmodule                                       always block.
```

TIMER CIRCUITS The program TimerCircuits uses two one-shot instances consisting of a 25 s timer (TLong) and a 4 s timer (TShort). The 25 s and the 4 s timers are triggered by long trigger (LongTrig) and short trigger (ShortTrig). In the VHDL and Verilog programs, countdown timers driven by a 1 Hz clock input (Clk) replicate the one-shot components TLong and TShort. The values stored in SetCountLong and SetCountShort are assigned to the Duration inputs of one-shot components TLong and TShort, setting the

FIGURE P–6 Example pages of system implementation with programmable logic.

Troubleshooting Section A portion of a typical troubleshooting section is shown in Figure P–7.



9–7 TROUBLESHOOTING

During program code development, simulation tools can be used to validate logic modules for proper operation prior to PLD programming. Two basic ways to test a device that has been programmed with a logic design are traditional and automated. In the traditional method, common laboratory test instruments can be used to check the operation. In the automated method, three fundamental approaches can be used for testing: bed-of-nails, flying probe, and boundary scan. Bed-of-nails and flying probe were described in Chapter 6, and boundary scan is introduced now. In this section, the focus is on simulation prior to device programming and boundary scan testing once the PLD has been programmed.

After completing this section, you should be able to

- Explain troubleshooting techniques using waveform simulation
- Define *break point*
- Discuss boundary scan testing

Troubleshooting with Waveform Simulation

As discussed, simulation waveform stimulus can be accomplished using a test bench program or graphically using a waveform editor. The following illustration demonstrates simulation troubleshooting techniques applied to the SequentialLogic section of the traffic signal control system created in VHDL.

FUNCTIONAL SIMULATION Prior to download to the target device, simulation tools are useful to identify unexpected behavior. In the following illustration, the waveform output in Figure 9–55 shows that the sequential logic Gray code output from identifiers g0 and g1 does not respond to the waveform test stimulus as expected. In a timing simulation, the PLD chip libraries are loaded, and testing is conducted against a model of the target device where typically outputs start at a zero state. In the functional simulation, the basic logic is tested. Since functional simulation does not make assumptions about initial states, a circular dependency could exist where the output of one function is used to determine the outcome of a second where neither may be resolved. A break point can be inserted in the program code to determine where undetermined states may exist, so they can be addressed in the program code if needed. A break point is a flag placed within the program source code where the application is stopped temporarily, allowing investigation of program identifiers and the status of the I/O.

FIGURE 9–55

To investigate this behavior, you can insert a break point into the program code, so you can view the condition of identifiers G0 and G1 as the simulation progresses.

In the sequential logic component of the traffic signal control system, identifiers D0 and D1 are dependent on the output of flip-flops DFF0 and DFF1. Since D0 feeds DFF0, for example, D0 could be in an undetermined state at startup, causing G0 to also be in an

480 CHAPTER 9 • PROGRAMMABLE LOGIC

undetermined state. The functional simulation would point this out as shown since G0 and G1 are left in an undetermined state. As shown in Figure 9–56, in this case, a break point is set by right-clicking line number 22 and selecting "Set Breakpoint 22". Multiple break points may be defined as needed to investigate the behavior of the program under simulation.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity SequentialLogic is
5    port(VS, TL, TS, Clk: in std_logic;
6         G0, G1: inout std_logic);
7  end entity SequentialLogic;
8
9  architecture SequenceBehavior of SequentialLogic is
10 component dff is
11   port(D,Clk: in std_logic;
12        Q: out std_logic);
13 end component dff;
14
15 signal D0 : std_logic;
16 signal D1 : std_logic;
17
18 begin
19   D1 <= (G0 and not TS) or
20         (G1 and TS);
21
22   D0 <= (not G1 and not TL and VS) or
23         (not G1 and TL and not VS);
24
25 DFF0 : dff port map(D => D0, Clk => Clk, Q => G0);
26 DFF1 : dff port map(D => D1, Clk => Clk, Q => G1);
27 end architecture SequenceBehavior;
```

Simulation run stopped at break point defined for identifier D0.

Hover over identifiers to view current status

FIGURE 9–56

The simulation has stopped at the predefined break point inserted at identifier D0. By examining the condition of the supporting identifiers D0, TL, VS and G1, you determine the problem to be related to the D flip-flop components whose output value G1 is listed as "U" or undefined. D0 is dependent on identifier G1 and the flip-flop. DFF1 is in turn dependent on D1. The output of the flip-flop does not allow resolution of the Boolean expressions assigned to D0 or D1.

Examining the D flip-flop definition, you see that the flip-flop simply writes the value of the D input to output Q upon a rising clock edge. Figure 9–57(a) shows that the output

(a) (b)

Signal QT allows for the pre-initialization of a 0 to output Q

FIGURE 9–57

FIGURE P–7 Partial Troubleshooting Section.

Other Features

End of Chapter The following features are at the end of each chapter.

- Summary
- Key term glossary
- True/False quiz
- Self-test
- Sectionalized and categorized problem set
- Answers to section checkups, related problems for examples, true/false quiz, and self-test

End of Book The following features are at the end of book.

- Appendices: Code conversions and Powers-of-two table; Security System Component programs
- Comprehensive glossary
- Answers to odd-numbered problems
- Index

Website The website (www.pearsonhighered.com/floyd) offers the topics related to the textbook for reference or advanced informations, as shown in Figure P–8.

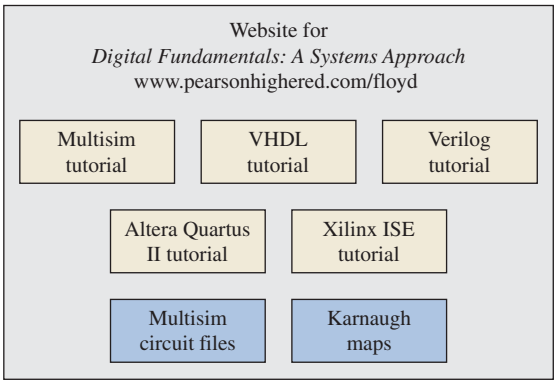


FIGURE P–8 **Website for *Digital Fundamentals: A Systems Approach.***

To the Student

Today, it seems that digital technology pervades most everything and is continuously changing. This makes it essential that you obtain a thorough grounding in the fundamentals because even though technology continues to change, the fundamentals remain intact. By fully understanding fundamental concepts, you can adapt to changing conditions.

A digital system is a combination of many logic functions that operate together to produce a desired result. This book not only covers the fundamentals of digital technology but presents several basic systems and shows how fundamental concepts and individual logic devices are used in them. Those working in electronics technology should have a basic grasp of the system concept and a practical knowledge of how to apply the fundamentals.

Today, logic devices are integrated circuits that can be programmed to perform a desired function. Although some fixed-function devices remain, the trend is mainly toward the use and application of programmable logic devices (PLDs). Therefore, two major programming languages used for logic programming are introduced in this text (VHDL and Verilog). The coverage in this book is not intended to make you an expert on the programming languages, but only to familiarize you with some basic concepts. Tutorials are available at the website, and more advanced coverage can be found from many sources.

Anyone working in the field needs to be able to troubleshoot systems. The troubleshooting methods and examples provided in this book will help you to get started as a troubleshooter.

To the Instructor

Time limitations and/or program emphasis generally are major factors in the amount of material and the topics covered in a course. It is not uncommon to omit or condense selected topics or to alter the sequence of certain topics in order to accommodate the requirements of a course. To this end, topics have been organized with a ”modular” approach so that certain topics are not integrated or intermingled with the more basic core topics. Topics such as programmable logic, PLD programming, and troubleshooting are contained in separate chapters, dedicated sections within a chapter, or on the website to permit more flexible treatment.

A fairly strong emphasis is placed on programmable logic devices because they are so prevalent in the implementation of today’s systems. PLD and programming coverage using VHDL and Verilog is introduced at a fundamental level to provide a basic foundation and is not intended to be a comprehensive treatment. You may choose to cover either VHDL or Verilog or both. Tutorials for PLD programming (VHDL, Verilog, Altera Quartus II, and Xilinx ISE) are provided on the Internet to assist students in their study. Of course, much more extensive coverage of these topics can be found at many Internet sites if that is deemed necessary. Fixed-function logic devices (7400 series for example) are introduced but given a very light treatment due to their declining availability and use.

Customizing the Contents You can structure your course around this text from a minimal coverage to a full-blown coverage. Table P–1 provides some suggestions with Option 1 being minimal and Option 9 being full coverage. You can decide the best approach for your course using one of these suggestions or you may decide to choose some other combination of topics.

Table P–1 • Some suggested combinations of topics. Many others are possible.

OPTIONS	CORE + SYSTEMS	PLDS	PLD PROG	TROUBLESHOOTING	CH 11	CH12	CH13	CH14
1	Yes	No	No	No	No	No	No	No
2	Yes	No	No	Yes	Yes	No	No	No
3	Yes	No	No	Yes	Yes	Yes	No	No
4	Yes	No	No	No	Yes	Yes	Yes	Yes
5	Yes	No	No	Yes	Yes	Yes	Yes	Yes
6	Yes	Yes	No	Yes	No	No	No	No
7	Yes	Yes	Yes	No	No	No	No	No
8	Yes	Yes	Yes	Yes	Yes	No	No	Yes
9	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Acknowledgments

The concept of this series of systems-oriented textbooks is credited to suggestions and discussions with senior instructional staff at ITT Schools and Vern Anthony at Pearson Education. The staff and others at Pearson Education, by their hard work and dedication, have helped make the textbook a reality. Lois Porter, who did the manuscript editing, has done a great job and she has helped me to turn my original rough manuscript into a top quality marketable product. Rex Davidson skillfully guided the work through its many detailed phases of production to create the end product that you are now looking at. Lindsey Prudhomme, acquisitions editor, and Dan Trudden, development editor, have provided effective overall guidance for this project.

Many thanks also to Gary Snyder and Doug Joksch for their extensive contributions to this book. Doug developed the VHDL and Verilog programs and much of the supporting text in the area of programmable logic. Gary provided the Multisim circuit files and wrote much of the chapter on data processing and control.

TOM FLOYD

CHAPTER 1

INTRODUCTION TO DIGITAL SYSTEMS

OUTLINE

OBJECTIVES

- Explain the basic differences between digital and analog quantities
- Show how voltage levels are used to represent digital quantities
- Describe various parameters of a pulse waveform such as rise time, fall time, pulse width, frequency, period, and duty cycle
- Explain the logic operations of NOT, AND, and OR
- Describe several types of logic functions
- Describe programmable logic, discuss the various types, and describe how PLDs are programmed using VHDL and Verilog with system software
- Describe the basics of a microcontroller
- Identify fixed-function digital integrated circuits according to their technology and the type of packaging
- Discuss how various logic functions are used in a digital system
- Recognize various instruments and understand how they are used in measurement and troubleshooting digital devices and systems

KEY TERMS

Key terms are in order of appearance in the chapter.

Analog
Digital
Digital system
Binary
Bit
Pulse
Duty cycle
Clock
Timing diagram
Data
Serial
Parallel
Logic
Input
Output
Gate
NOT
Inverter
AND
OR
Programmable logic device
SPLD
CPLD
FPGA
Compiler
Microcontroller
Embedded system
Integrated circuit (IC)
Fixed-function logic
Troubleshooting

VISIT THE WEBSITE
Study aids for this chapter are available at
http://pearsonhighered.com/floyd

INTRODUCTION

The term *digital* is derived from the way operations are performed, by counting digits. For many years, applications of digital electronics were confined to computer systems. Today, digital technology is applied in a wide range of systems in addition to computers. Such applications as television, communications systems, radar, navigation and guidance systems, military systems, medical instrumentation, industrial process control, and consumer electronics use digital techniques. Over the years digital technology has progressed from vacuum-tube circuits to fixed-function integrated circuits to programmable logic and embedded microcontrollers.

This chapter introduces you to digital electronics and provides a broad overview of many important concepts, applications, and methods.

1–1 DIGITAL AND ANALOG SIGNALS AND SYSTEMS

Electronic systems can be divided into two broad categories, digital and analog. Digital electronics involves quantities with discrete values, and analog electronics involves quantities with continuous values. Although you will be studying digital fundamentals in this book, you should also know something about analog because many applications require both; and interfacing between analog and digital is important.

After completing this section, you should be able to

- **Define *analog***
- **Define *digital***
- **Explain the difference between digital and analog signals**
- **State the advantages of digital over analog**
- **Discuss modulation methods**
- **Describe two types of digital systems**

An **analog*** quantity is one having continuous values. A **digital** quantity is one having a discrete set of values. Most things that can be measured quantitatively occur in nature in analog form. For example, the air temperature changes over a continuous range of values. During a given day, the temperature does not go from, say, 70° to 71° instantaneously; it takes on all the infinite values in between. If you graphed the temperature on a typical summer day, you would have a smooth, continuous curve similar to the curve in Figure 1–1. Other examples of analog quantities are time, pressure, distance, and sound.

Rather than graphing the temperature on a continuous basis, suppose you just take a temperature reading every hour. Now you have sampled values representing the temperature at discrete points in time (every hour) over a 24-hour period, as indicated in Figure 1–2. You have effectively converted an analog quantity to a form that can now be digitized by representing each sampled value by a digital code. It is important to realize that Figure 1–2 itself is not the digital representation of the analog quantity.

*All bold terms are important and are defined in the end-of-book glossary. The bold terms in color are key terms and are included in a Key Term glossary at the end of each chapter.

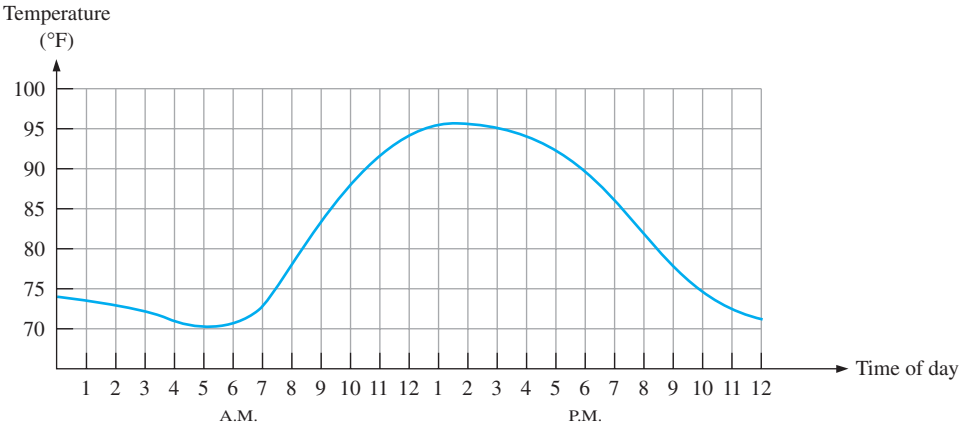


FIGURE 1–1 **Graph of an analog quantity (temperature versus time).**

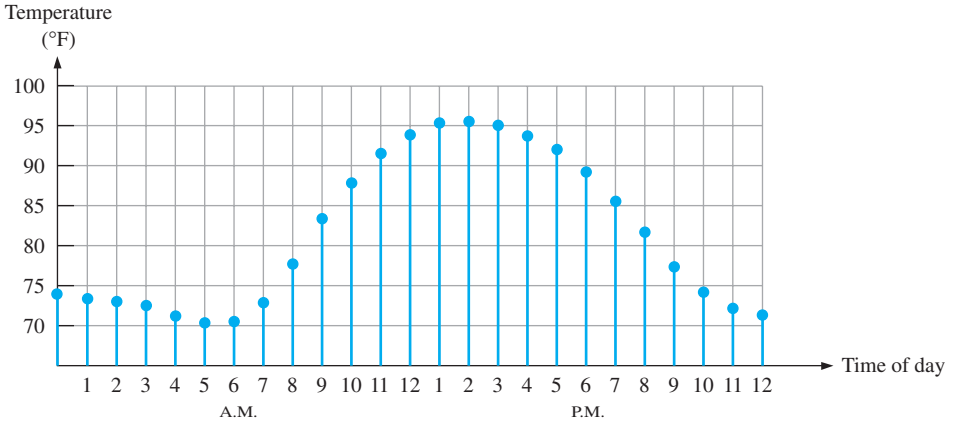


FIGURE 1–2 **Sampled-value representation (quantization) of the analog quantity in Figure 1–1. Each value represented by a dot can be digitized by representing it as a digital code that consists of a series of 1s and 0s.**

THE DIGITAL ADVANTAGE Digital representation has certain advantages over analog representation in electronics applications. For one thing, digital data can be processed and transmitted more efficiently and reliably than analog data. Also, digital data has a great advantage when storage is necessary. For example, music when converted to digital form can be stored more compactly and reproduced with greater accuracy and clarity than is possible when it is in analog form. Noise (unwanted voltage fluctuations) does not affect digital data nearly as much as it does analog signals.

Analog Signals

An analog quantity, such as voltage, that is repetitive or varies in a certain manner is an analog signal. An analog signal can be a repetitive waveform, such as the sine wave in Figure 1–3(a), or a continuously varying audio signal that carries information (music, the spoken word, or other sounds), as shown in part (b). Other examples of analog signals are amplitude-modulated signals (AM) and frequency-modulated signals (FM), as illustrated in parts (c) and (d). In AM, a lower-frequency information signal, such as voice, varies the amplitude of a high-frequency sine wave. In FM, the information signal varies the frequency of the sine wave.

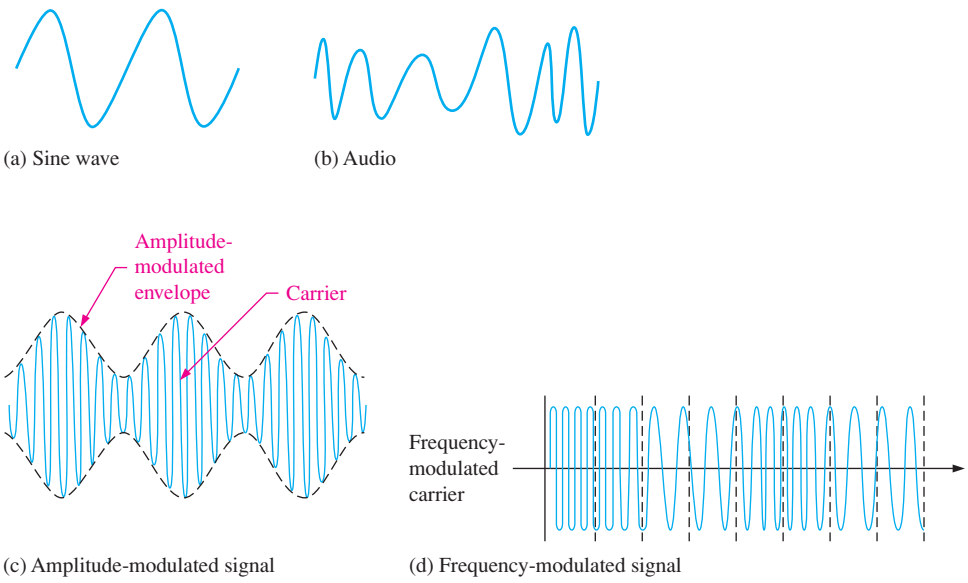


FIGURE 1–3 **Examples of analog signals.**

Digital Signals

A digital signal is a representation of a sequence of discrete values that are coded into a stream of 1s and 0s. A bit stream appears as a train of pulses or voltage levels where a high voltage level conveys a binary 1 and a low voltage level conveys a binary 0. Bit streams are used in telecommunications, computers, and other system applications. Figure 1–4 illustrates one type of digital signal. The duration of each bit (bit time) is indicated by the hash marks.

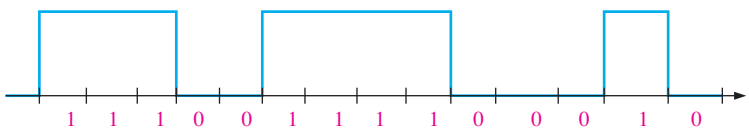


FIGURE 1–4 **Example of a digital waveform.**

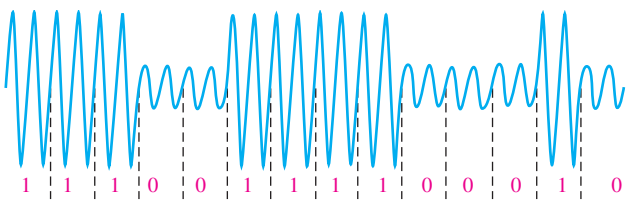


FIGURE 1–5 **Example of a digital-modulated signal.**

DIGITAL MODULATION In some applications, analog and digital signals are combined with a sine wave, called a *carrier*, by amplitude modulating the sine wave with the digital waveform. A common example is a modem that turns digital data from a computer into modulated signals in the voice frequency range for transmission over telephone lines. A digital-modulated signal is shown in Figure 1–5 where the digital signal (bit stream) in Figure 1–4 modulates the sine wave. Dashed lines mark the bit times. The frequency of the sine wave is shown arbitrarily low in relation to the digital-modulating signal for illustration.

PULSE-CODE MODULATION (PCM) A PCM signal represents sampled analog signals with a sequence of digital codes. It is used in computers for digital audio, in Blu-ray, compact disc and DVD formats, and in digital telephone systems. The sampling process results in a "stair-step" voltage as shown in Figure 1–6. The analog signal is sampled at each step, and each sampled value is converted (quantized) to a digital code. The

digital signal would be the time sequence of the digital codes where the binary numbers shown for each step appear in sequence beginning at the left. The more steps there are the more accurate is the digital representation. The length of the code depends on the number of steps.

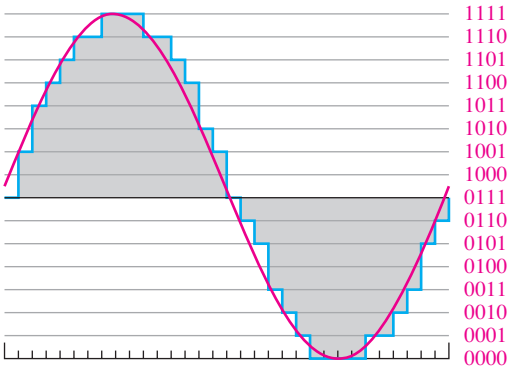


FIGURE 1–6 **Illustration of pulse-code modulation.**

Digital Systems

A **digital system** is an arrangement of the individual logic functions connected to perform a specified operation or produce a defined output. An example of a digital system is a computer, as shown in Figure 1–7 in basic block diagram form. A computer processes, transfers, and stores data in digital form (1s and 0s). To make a complete system, the computer is interfaced with peripheral devices such as a modem, a mouse, a keyboard, and a monitor.

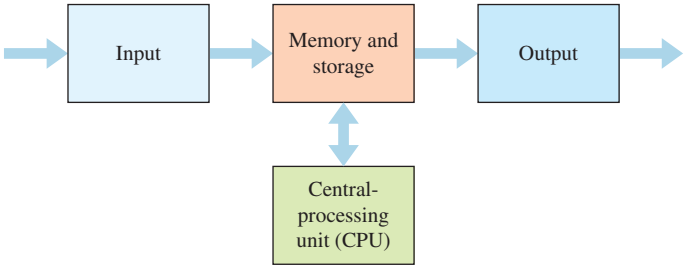


FIGURE 1–7 **Basic block diagram of a computer.**

Figure 1–8, another example of a digital system, shows the traffic light controller that you will study in Chapter 6. All of the digital signals that the system uses to properly sequence the traffic light are internally generated, making the controller a type of finite state machine.

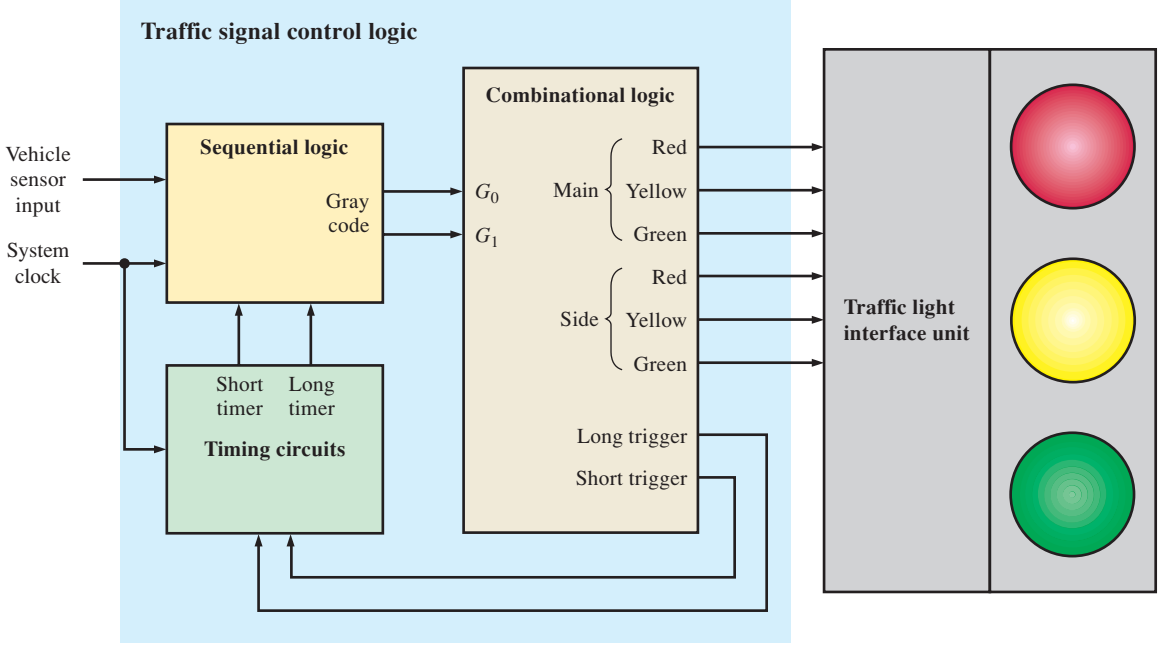


FIGURE 1–8 **A digital traffic light controller.**

Analog Systems

An **analog system** is one that processes data in analog form only. One example is a public address system, used to amplify sound so that it can be heard by a large audience. The basic diagram in Figure 1–9 illustrates that sound waves, which are analog in nature, are picked up by a microphone and converted to a small analog voltage called the audio signal. This voltage varies continuously as the volume and frequency of the sound changes and is applied to the input of a linear amplifier. The output of the amplifier, which is an increased reproduction of input voltage, goes to the speaker(s). The speaker changes the amplified audio signal back to sound waves that have a much greater volume than the original sound waves picked up by the microphone.

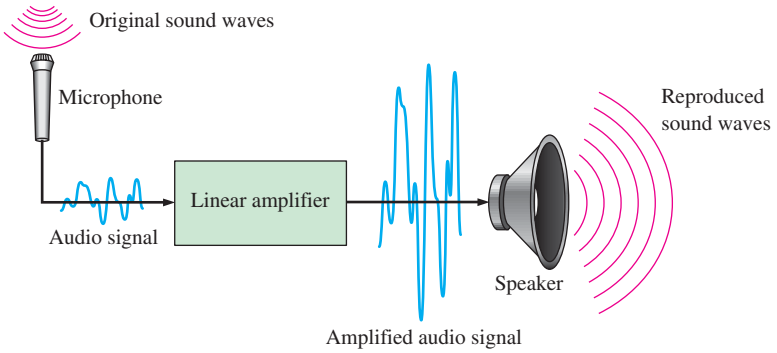


FIGURE 1–9 **A basic audio public address system.**

Another example of an analog system is the FM receiver. The system processes the incoming frequency-modulated carrier signal, extracts the audio signal for amplification, and produces audible sound waves. A block diagram is shown in Figure 1–10 with a representative signal shown at each point in the system.

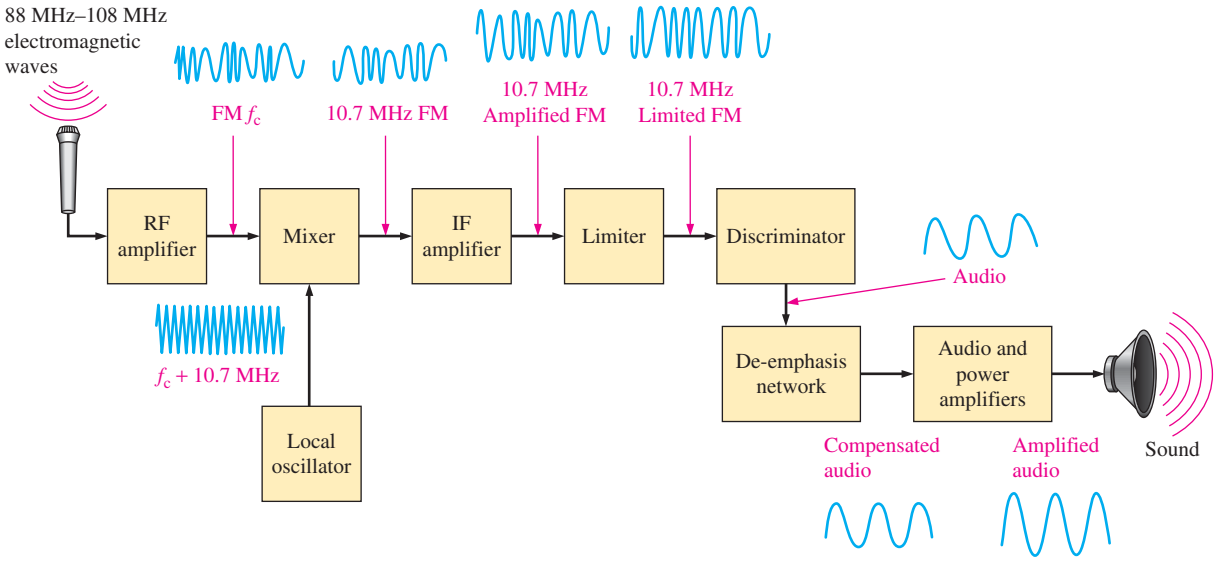


FIGURE 1–10 **Block diagram of superheterodyne FM receiver.**

A Combination Digital and Analog System

The compact disk (CD) player is an example of a system in which both digital and analog elements are used. The simplified block diagram in Figure 1–11 illustrates the basic system. Music in digital form is stored on the compact disk. A laser diode optical system picks up

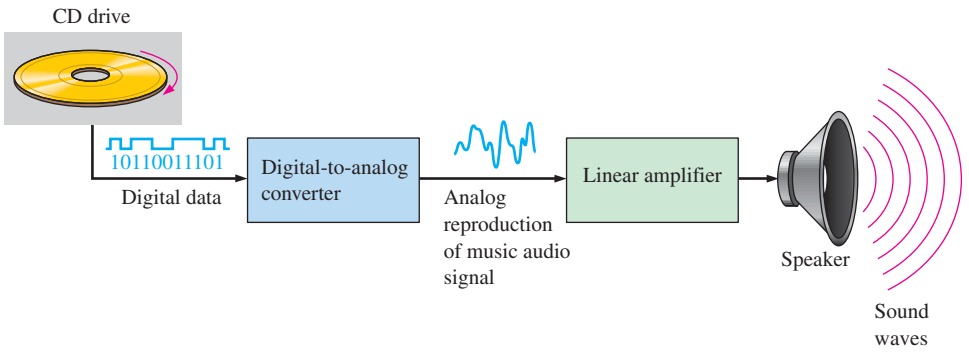


FIGURE 1–11 **Simplified diagram of a compact disk player.**

the digital data from the rotating disk and transfers it to the **digital-to-analog converter (DAC).** The DAC changes the digital data into an analog signal that is an electrical reproduction of the original music. This signal is amplified and sent to the speaker for you to enjoy. When the music was originally recorded on the CD, a process, essentially the reverse of the one described here, using an **analog-to-digital converter (ADC)** was used.

SECTION 1–1 CHECKUP*

1. Define *analog.*
2. Define *digital.*
3. Explain the difference between a digital quantity and an analog quantity.
4. Give an example of a system that is analog and one that is a combination of both digital and analog. Name a system that is entirely digital.

***Answers are at the end of the chapter.**

1–2 BINARY DIGITS, LOGIC LEVELS, AND DIGITAL WAVEFORMS

Digital systems involve operations in which there are only two possible states. These states are represented by two different voltage levels: A HIGH and a LOW. The two states can also be represented by current levels or pits and lands on a CD or DVD. In digital systems such as computers, combinations of the two states, called *codes,* are used to represent numbers, symbols, alphabetic characters, and other types of information. The two-state number system is called *binary,* and its two digits are 0 and 1. A binary digit is called a *bit.*

After completing this section, you should be able to

- **Define *binary***
- **Define *bit***
- **Name the bits in a binary system**
- **Explain how voltage levels are used to represent bits**
- **Explain how voltage levels are interpreted by a digital circuit**
- **Describe the general characteristics of a pulse**
- **Determine the amplitude, rise time, fall time, and width of a pulse**
- **Identify and describe the characteristics of a digital waveform**
- **Determine the amplitude, period, frequency, and duty cycle of a digital waveform**
- **Explain what a timing diagram is and state its purpose**
- **Explain serial and parallel data transfer and state the advantage and disadvantage of each**

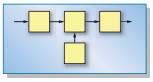
Binary Digits

Each of the two digits in the **binary** system, 1 and 0, is called a **bit**, which is a contraction of the words *binary digit.* In digital circuits, two different voltage levels are used to represent the two bits. Generally, 1 is represented by the higher voltage, which we will refer to as a HIGH, and a 0 is represented by the lower voltage level, which we will refer to as a LOW. This is called **positive logic** and will be used throughout the book.

HIGH = 1 and LOW = 0

Another system in which a 1 is represented by a LOW and a 0 is represented by a HIGH is called *negative logic.*

Groups of bits (combinations of 1s and 0s), called *codes,* are used to represent numbers, letters, symbols, instructions, and anything else required in a given application.



The concept of a digital computer can be traced back to Charles Babbage, who developed a crude mechanical computation device in the 1830s. John Atanasoff was the first to apply electronic processing to digital computing in 1939. In 1946, an electronic digital computer called ENIAC was implemented with vacuum-tube circuits. Even though it took up an entire room, ENIAC didn't have the computing power of your handheld calculator.

SYSTEM NOTE

Logic Levels

The voltages used to represent a 1 and a 0 are called *logic levels.* Ideally, one voltage level represents a HIGH and another voltage level represents a LOW. In a practical digital circuit, however, a HIGH can be any voltage between a specified minimum value and a specified maximum value. Likewise, a LOW can be any voltage between a specified minimum and a specified maximum. There can be no overlap between the accepted range of HIGH levels and the accepted range of LOW levels.

Figure 1–12 illustrates the general range of LOWs and HIGHs for a digital circuit. The variable $V_{H(max)}$ represents the maximum HIGH voltage value, and $V_{H(min)}$ represents the minimum HIGH voltage value. The maximum LOW voltage value is represented by $V_{L(max)}$, and the minimum LOW voltage value is represented by $V_{L(min)}$. The voltage values between $V_{L(max)}$ and $V_{H(min)}$ are unacceptable for proper operation. A voltage in the unacceptable range can appear as either a HIGH or a LOW to a given circuit. For example, the HIGH input values for a certain type of digital circuit technology called CMOS may range from 2 V to 3.3 V and the LOW input values may range from 0 V to 0.8 V. If a voltage of 2.5 V is applied, the circuit will accept it as a HIGH or binary 1. If a voltage of 0.5 V is applied, the circuit will accept it as a LOW or binary 0. For this type of circuit, voltages between 0.8 V and 2 V are unacceptable.

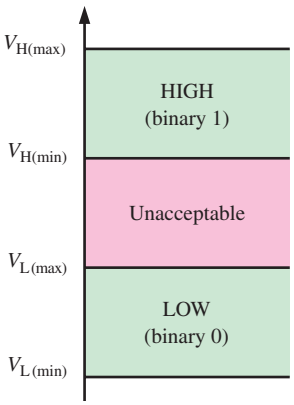
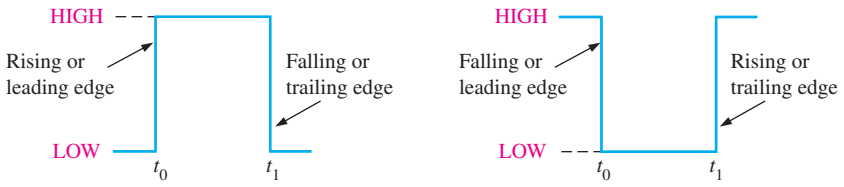


FIGURE 1–12 Logic level ranges of voltage for a digital circuit.

Digital Waveforms

Digital waveforms consist of voltage levels that are changing back and forth between the HIGH and LOW levels or states. Figure 1–13(a) shows that a single positive-going **pulse** is generated when the voltage (or current) goes from its normally LOW level to its HIGH level and then back to its LOW level. The negative-going pulse in Figure 1–13(b) is generated when the voltage goes from its normally HIGH level to its LOW level and back to its HIGH level. A digital waveform is made up of a series of pulses.

THE PULSE As indicated in Figure 1–13, a pulse has two edges: a **leading edge** that occurs first at time t_0 and a **trailing edge** that occurs last at time t_1. For a positive-going pulse, the leading edge is a rising edge, and the trailing edge is a falling edge. The pulses



(a) Positive–going pulse

(b) Negative–going pulse

FIGURE 1–13 **Ideal pulses.**

in Figure 1–13 are ideal because the rising and falling edges are assumed to change in zero time (instantaneously). In practice, these transitions never occur instantaneously, although for most digital work you can assume ideal pulses.

Figure 1–14 shows a nonideal pulse. In reality, all pulses exhibit some or all of these characteristics. The overshoot and ringing are sometimes produced by stray inductive and capacitive effects. The droop can be caused by stray capacitance and circuit resistance, forming an *RC* circuit with a low time constant.

The time required for a pulse to go from its LOW level to its HIGH level is called the **rise time** (t_r), and the time required for the transition from the HIGH level to the LOW level is called the **fall time** (t_f). In practice, it is common to measure rise time from 10% of the pulse **amplitude** (height from baseline) to 90% of the pulse amplitude and to measure the fall time from 90% to 10% of the pulse amplitude, as indicated in Figure 1–14. The bottom 10% and the top 10% of the pulse are not included in the rise and fall times because of the nonlinearities in the waveform in these areas. The **pulse width** (t_{PW}) is a measure of the duration of the pulse and is often defined as the time interval between the 50% points on the rising and falling edges, as indicated in Figure 1–14.

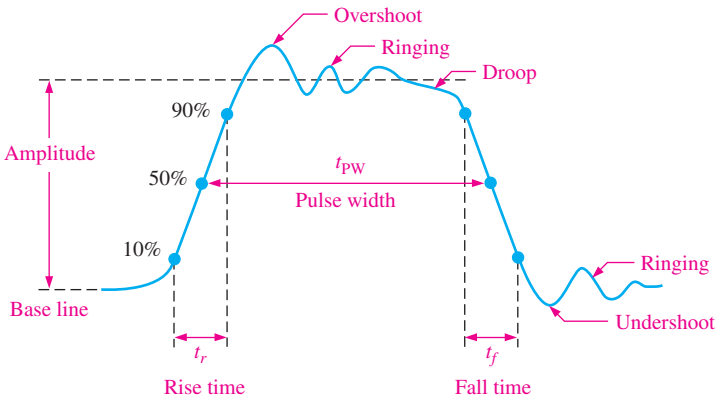
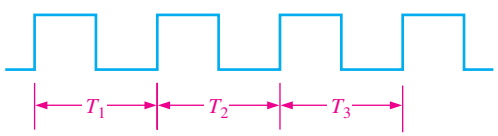


FIGURE 1–14 **Nonideal pulse characteristics.**

WAVEFORM CHARACTERISTICS Most waveforms encountered in digital systems are composed of series of pulses, sometimes called *pulse trains,* and can be classified as either periodic or nonperiodic. A **periodic** pulse waveform is one that repeats itself at a fixed interval, called a **period (*T*).** The **frequency (*f*)** is the rate at which it repeats itself and is measured in hertz (Hz). A nonperiodic pulse waveform, of course, does not repeat itself at fixed intervals and may be composed of pulses of randomly differing pulse widths and/or randomly differing time intervals between the pulses. An example of each type is shown in Figure 1–15.



Period = $T_1 = T_2 = T_3 = \ldots = T_n$

Frequency = $\frac{1}{T}$

(a) Periodic (square wave)

(b) Nonperiodic

FIGURE 1–15 **Examples of digital waveforms.**

The frequency (f) of a pulse (digital) waveform is the reciprocal of the period. The relationship between frequency and period is expressed as follows:

$$f = \frac{1}{T} \tag{1–1}$$

$$T = \frac{1}{f} \tag{1–2}$$

An important characteristic of a periodic digital waveform is its **duty cycle,** which is the ratio of the pulse width (t_{PW}) to the period (T). It can be expressed as a percentage.

$$\text{Duty cycle} = \left(\frac{t_{PW}}{T}\right)100\% \tag{1–3}$$

EXAMPLE 1–1

A portion of a periodic digital waveform is shown in Figure 1–16. The measurements are in milliseconds. Determine the following:

(a) period **(b)** frequency **(c)** duty cycle

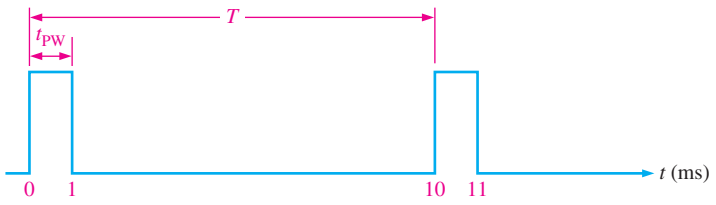


FIGURE 1–16

SOLUTION

(a) The period is measured from the edge of one pulse to the corresponding edge of the next pulse. In this case T is measured from leading edge to leading edge, as indicated. T equals **10 ms.**

(b) $f = \frac{1}{T} = \frac{1}{10\text{ ms}} = \mathbf{100\ Hz}$

(c) $\text{Duty cycle} = \left(\frac{t_{PW}}{T}\right)100\% = \left(\frac{1\text{ ms}}{10\text{ ms}}\right)100\% = \mathbf{10\%}$

RELATED PROBLEM*

A periodic digital waveform has a pulse width of 25 μs and a period of 150 μs. Determine the frequency and the duty cycle.

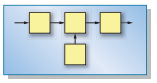
***Answers are at the end of the chapter.**

A Digital Waveform Carries Binary Information

Binary information that is handled by digital systems appears as waveforms that represent sequences of bits. When the waveform is HIGH, a binary 1 is present; when the waveform is LOW, a binary 0 is present. Each bit in a sequence occupies a defined time interval called a **bit time.**

The speed at which a computer can operate depends on the type of microprocessor used in the system. The speed specification, for example 3.5 GHz, of a computer is the maximum clock frequency at which the microprocessor can run.

SYSTEM NOTE



THE CLOCK In digital systems, all waveforms are synchronized with a basic timing waveform called the **clock**. The clock is a periodic waveform in which each interval between pulses (the period) equals the time for one bit.

An example of a clock waveform is shown in Figure 1–17. Notice that, in this case, each change in level of waveform *A* occurs at the leading edge of the clock waveform. In other cases, level changes occur at the trailing edge of the clock. During each bit time of the clock, waveform *A* is either HIGH or LOW. These HIGHs and LOWs represent a sequence of bits as indicated. A group of several bits can be used as a piece of binary information, such as a number or a letter. The clock waveform itself does not carry information.

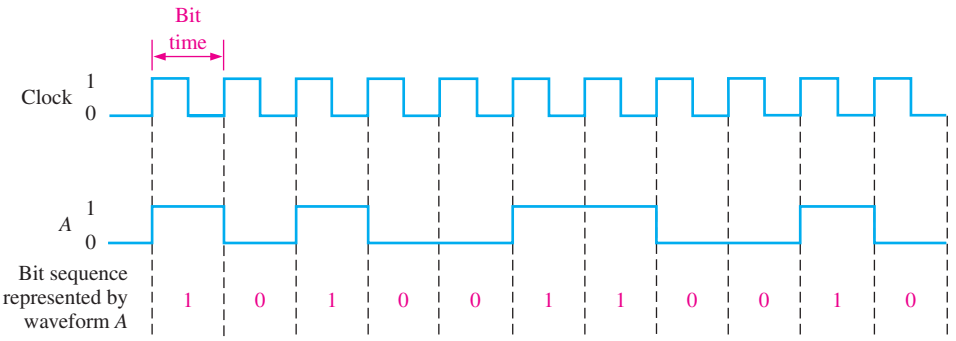


FIGURE 1–17 **Example of a clock waveform synchronized with a waveform representation of a sequence of bits.**

TIMING DIAGRAMS A **timing diagram** is a graph of digital waveforms showing the actual time relationship of two or more waveforms and how each waveform changes in relation to the others. By looking at a timing diagram, you can determine the states (HIGH or LOW) of all the waveforms at any specified point in time and the exact time that a waveform changes state relative to the other waveforms. Figure 1–18 is an example of a timing diagram made up of four waveforms. From this timing diagram you can see, for example, that the three waveforms *A*, *B*, and *C* are HIGH only during bit time 7 (shaded area) and they all change back LOW at the end of bit time 7.

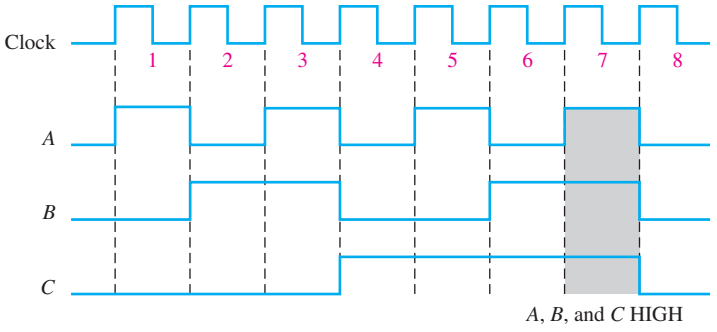
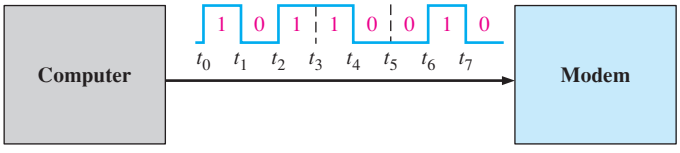


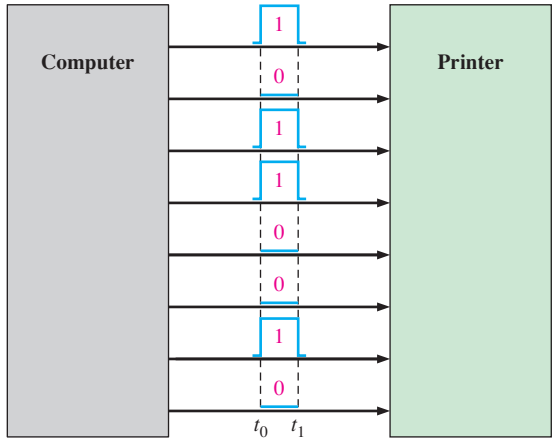
FIGURE 1–18 **Example of a timing diagram.**

Data Transfer

Data refers to groups of bits that convey some type of information. Binary data, which are represented by digital waveforms, must be transferred from one circuit to another within a digital system or from one system to another in order to accomplish a given purpose. For example, numbers stored in binary form in the memory of a computer must be transferred to the computer's central processing unit in order to be added. The sum of the addition must then be transferred to a monitor for display and/or transferred back to the memory. In computer systems, as illustrated in Figure 1–19, binary data are transferred in two ways: serial and parallel.



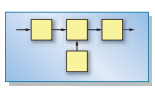
(a) Serial transfer of 8 bits of binary data from computer to modem. Interval t_0 to t_1 is first.



(b) Parallel transfer of 8 bits of binary data from computer to printer. The beginning time is t_0.

FIGURE 1–19 **Illustration of serial and parallel transfer of binary data. Only the data lines are shown.**

When bits are transferred in **serial** form from one point to another, they are sent one bit at a time along a single line, as illustrated in Figure 1–19(a) for the case of a computer-to-modem transfer. During the time interval from t_0 to t_1, the first bit is transferred. During the time interval from t_1 to t_2, the second bit is transferred, and so on. To transfer eight bits in series, it takes eight time intervals.



Universal Serial Bus (USB) is a serial bus standard for device interfacing. It was originally developed for the personal computer but has become widely used on many types of handheld and mobile devices. USB is expected to replace other serial and parallel ports. USB operated at 12 Mbps (million bits per second) when first introduced in 1995, but it now operates at up to 5 Gbps.

SYSTEM NOTE

When bits are transferred in **parallel** form, all the bits in a group are sent out on separate lines at the same time. There is one line for each bit, as shown in Figure 1–19(b) for the example of eight bits being transferred from a computer to a printer or other device. To transfer eight bits in parallel, it takes one time interval compared to eight time intervals for the serial transfer.

To summarize, an advantage of serial transfer of binary data is that a minimum of only one line is required. In parallel transfer, a number of lines equal to the number of bits to be transferred at one time is required. A disadvantage of serial transfer is that it can take longer to transfer a given number of bits than with parallel transfer at the same clock frequency. For example, if one bit can be transferred in 1 μs, then it takes 8 μs to serially transfer eight bits but only 1 μs to parallel transfer eight bits. A disadvantage of parallel transfer is that it takes more lines than serial transfer.

EXAMPLE 1–2

(a) Determine the total time required to serially transfer the eight bits contained in waveform *A* of Figure 1–20, and indicate the sequence of bits. The left-most bit is the first to be transferred. The 1 MHz clock is used as reference.

(b) What is the total time to transfer the same eight bits in parallel?

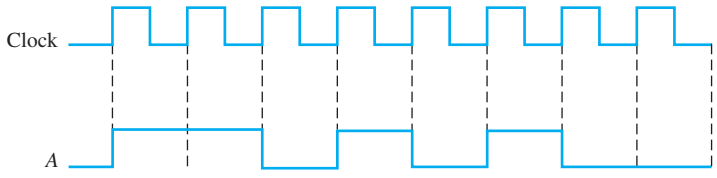


FIGURE 1–20

SOLUTION

(a) Since the frequency of the clock is 1 MHz, the period is

$$T = \frac{1}{f} = \frac{1}{1\text{ MHz}} = 1\ \mu\text{s}$$

It takes 1 μs to transfer each bit in the waveform. The total transfer time for 8 bits is

$$8 \times 1\ \mu\text{s} = \mathbf{8\ \mu s}$$

To determine the sequence of bits, examine the waveform in Figure 1–20 during each bit time. If waveform *A* is HIGH during the bit time, a 1 is transferred. If waveform *A* is LOW during the bit time, a 0 is transferred. The bit sequence is illustrated in Figure 1–21. The left-most bit is the first to be transferred.

FIGURE 1–21

(b) A parallel transfer would take **1 μs** for all eight bits.

RELATED PROBLEM

If binary data are transferred on a USB at the rate of 480 million bits per second (480 Mbps), how long will it take to serially transfer 16 bits?

SECTION 1–2 CHECKUP

1. Define *binary*.
2. What does *bit* mean?
3. What are the bits in a binary system?
4. How are the rise time and fall time of a pulse measured?
5. Knowing the period of a waveform, how do you find the frequency?
6. Explain what a clock waveform is.
7. What is the purpose of a timing diagram?
8. What is the main advantage of parallel transfer over serial transfer of binary data?

1–3 LOGIC OPERATIONS

In its basic form, logic is the realm of human reasoning that tells you a certain proposition (declarative statement) is true if certain conditions are true. Propositions can be classified as true or false. Many situations and processes that you encounter in your daily life can be expressed in the form of propositional, or logic, functions. Since such functions are true/false or yes/no statements, digital circuits with their two-state characteristics are applicable.

After completing this section, you should be able to

- **List three basic logic operations**
- **Define the NOT operation**
- **Define the AND operation**
- **Define the OR operation**

Several propositions, when combined, form propositional, or logic, functions. For example, the propositional statement "The light is on" will be true if "The bulb is not burned out" is true and if "The switch is on" is true. Therefore, this logical statement can be made: *The light is on only if the bulb is not burned out and the switch is on.* In this example the first statement is true only if the last two statements are true. The first statement ("The light is on") is then the basic proposition, and the other two statements are the conditions on which the proposition depends.

In the 1850s, the Irish logician and mathematician George Boole developed a mathematical system for formulating logic statements with symbols so that problems can be written and solved in a manner similar to ordinary algebra. Boolean algebra, as it is known today, is applied in the design and analysis of digital systems and will be covered in detail in Chapter 3.

The term **logic** is applied to digital circuits used to implement logic functions. Several kinds of digital logic **circuits** are the basic elements that form the building blocks for such complex digital systems as the computer. We will now look at these elements and discuss their functions in a very general way. Later chapters will cover these circuits in detail.

Three basic logic operations (NOT, AND, and OR) are indicated by standard distinctive shape symbols in Figure 1–22. An alternate standard symbol for each of these logic operations will be introduced in Chapter 3. The lines connected to each symbol are the **inputs** and **outputs.** The inputs are on the left of each symbol and the output is on the right. A circuit that performs a specified logic operation (AND, OR) is called a logic **gate.** AND and OR gates can have any number of inputs, as indicated by the dashes in the figure.

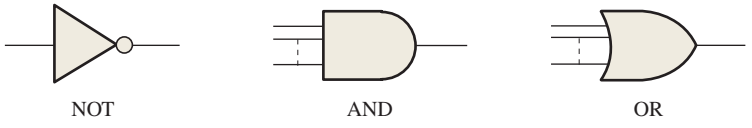


FIGURE 1–22 **The basic logic operations and symbols.**

In logic operations, the true/false conditions mentioned earlier are represented by a HIGH (true) and a LOW (false). Each of the three basic logic operations produces a unique response to a given set of conditions.

NOT

The **NOT** operation changes one logic level to the opposite logic level, as indicated in Figure 1–23. When the input is HIGH (1), the output is LOW (0). When the input is LOW,

FIGURE 1–23 **The NOT operation.**

the output is HIGH. In either case, the output is *not* the same as the input. The NOT operation is implemented by a logic circuit known as an **inverter.**

AND

The **AND** operation produces a HIGH output only when all the inputs are HIGH, as indicated in Figure 1–24 for the case of two inputs. When one input is HIGH *and* the other input is HIGH, the output is HIGH. When any or all inputs are LOW, the output is LOW. The AND operation is implemented by a logic circuit known as an *AND gate.*

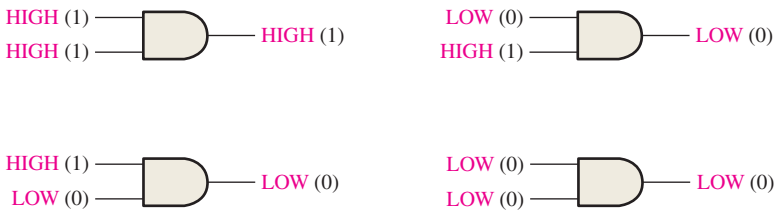


FIGURE 1–24 **The AND operation.**

OR

The **OR** operation produces a HIGH output when one or more inputs are HIGH, as indicated in Figure 1–25 for the case of two inputs. When one input is HIGH *or* the other input is HIGH *or* both inputs are HIGH, the output is HIGH. When both inputs are LOW, the output is LOW. The OR operation is implemented by a logic circuit known as an *OR gate.*

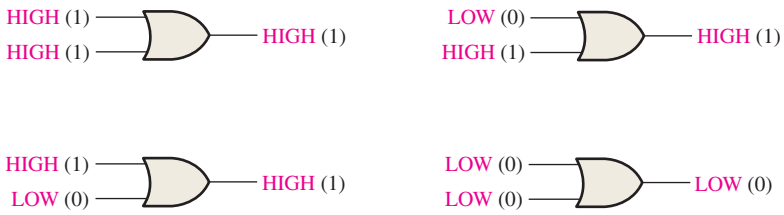


FIGURE 1–25 **The OR operation.**

SECTION 1–3 CHECKUP

1. When does the NOT operation produce a HIGH output?
2. When does the AND operation produce a HIGH output?
3. When does the OR operation produce a HIGH output?
4. What is an inverter?
5. What is a logic gate?

1–4 COMBINATIONAL AND SEQUENTIAL LOGIC FUNCTIONS

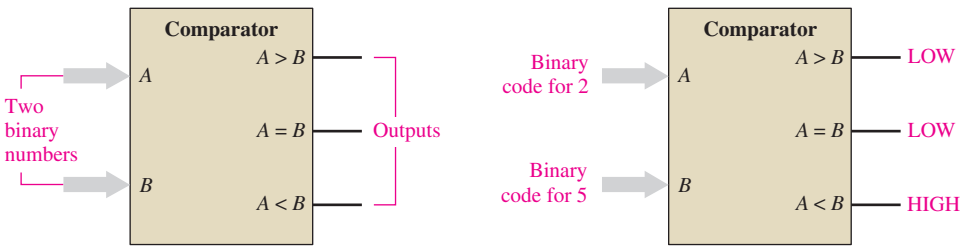
The three basic logic elements AND, OR, and NOT can be combined to form various types of logic functions: comparison, arithmetic, code conversion, encoding, decoding, data selection, counting, and storage. This section provides an overview of important logic functions and illustrates how they can be used in a specific system.

After completing this section, you should be able to

- **List several types of logic functions**
- **Describe comparison and list the four arithmetic functions**
- **Describe code conversion, encoding, and decoding**
- **Describe multiplexing and demultiplexing**
- **Describe the counting function**
- **Describe the storage function**

The Comparison Function

Magnitude comparison is performed by a logic circuit called a **comparator**. A comparator compares two quantities and indicates whether or not they are equal. For example, suppose you have two numbers and wish to know if they are equal or not equal and, if not equal, which is greater. The comparison function is represented in Figure 1–26. One number in binary form (represented by logic levels) is applied to input *A*, and the other number in binary form (represented by logic levels) is applied to input *B*. The outputs indicate the relationship of the two numbers by producing a HIGH level on the proper output line. Suppose that a binary representation of the number 2 is applied to input *A* and a binary representation of the number 5 is applied to input *B*. (The binary representation of numbers and symbols is discussed in Chapter 2.) A HIGH level will appear on the $A < B$ (*A* is less than *B*) output, indicating the relationship between the two numbers (2 is less than 5). The wide arrows represent a group of parallel lines on which the bits are transferred.



(a) Basic magnitude comparator

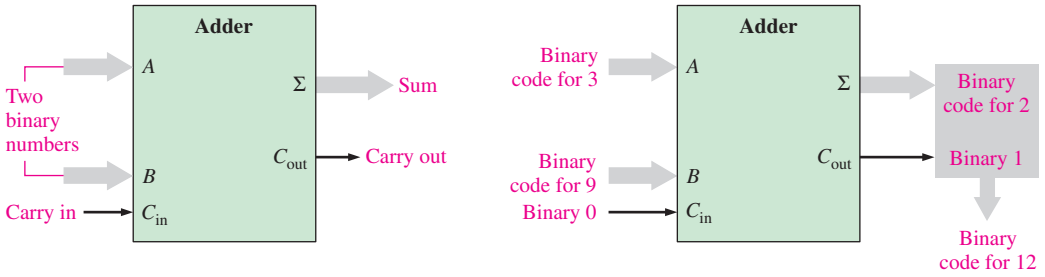
(b) Example: *A* is less than *B* (2 < 5) as indicated by the HIGH output ($A < B$)

FIGURE 1–26 **The comparison function.**

The Arithmetic Functions

ADDITION Addition is performed by a logic circuit called an **adder.** An adder adds two binary numbers (on inputs *A* and *B*) with a carry input C_{in} and generates a sum (Σ) and a carry output (C_{out}), as shown in Figure 1–27(a). Figure 1–27(b) illustrates the addition of 3 and 9. You know that the sum is 12; the adder indicates this result by producing the code for 2 on the sum output and 1 on the carry output. Assume that the carry input in this example is 0.

SUBTRACTION Subtraction is also performed by a logic circuit. A **subtracter** requires three inputs: the two numbers that are to be subtracted and a borrow input. The



(a) Basic adder

(b) Example: *A* plus *B* (3 + 9 = 12)

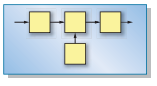
FIGURE 1–27 **The addition function.**

two outputs are the difference and the borrow output. When, for instance, 5 is subtracted from 8 with no borrow input, the difference is 3 with no borrow output. You will see in Chapter 2 how subtraction can actually be performed by an adder because subtraction is simply a special case of addition.

MULTIPLICATION Multiplication is performed by a logic circuit called a *multiplier*. Numbers are always multiplied two at a time, so two inputs are required. The output of the multiplier is the product. Because multiplication is simply a series of additions with shifts in the positions of the partial products, it can be performed by using an adder in conjunction with other circuits.

In a microprocessor, the arithmetic logic unit (ALU) performs the operations of add, subtract, multiply, and divide as well as the logic operations on digital data as directed by a series of instructions. A typical ALU is constructed of many thousands of logic gates.

SYSTEM NOTE



DIVISION Division can be performed with a series of subtractions, comparisons, and shifts, and thus it can also be done using an adder in conjunction with other circuits. Two inputs to the divider are required, and the outputs generated are the quotient and the remainder.

The Code Conversion Function

A **code** is a set of bits arranged in a unique pattern and used to represent specified information. A code converter changes one form of coded information into another coded form. Examples are conversion between binary and other codes such as the binary coded decimal (BCD) and the Gray code.

The Encoding Function

The encoding function is performed by a logic circuit called an **encoder.** The encoder converts information, such as a decimal number or an alphabetic character, into some coded form. For example, one certain type of encoder converts each of the decimal digits, 0 through 9, to a binary code. A HIGH level on the input corresponding to a specific decimal digit produces logic levels that represent the proper binary code on the output lines.

Figure 1–28 is a simple illustration of an encoder used to convert (encode) a calculator keystroke into a binary code that can be processed by the calculator circuits.

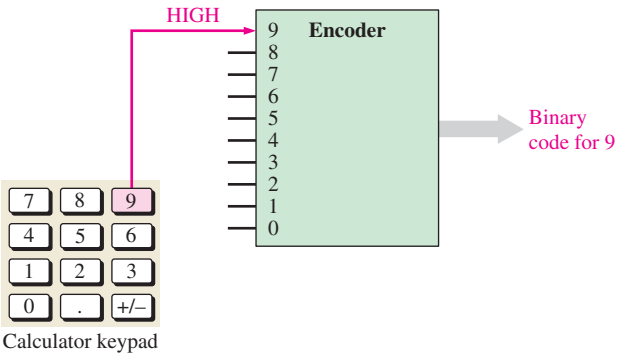


FIGURE 1–28 **A calculator keystroke encoded into a binary code for processing by the calculator system.**

The Decoding Function

The decoding function is performed by a logic circuit called a **decoder.** The decoder converts coded information, such as a binary number, into a noncoded form, such as a decimal form. For example, one particular type of decoder converts a 4-bit binary code into the appropriate decimal digit.

Figure 1–29 is a simple illustration of one type of decoder that is used to activate a 7-segment display. Each of the seven segments of the display is connected to an output line from the decoder. When a particular binary code appears on the decoder inputs, the appropriate output lines are activated and light the proper segments to display the decimal digit corresponding to the binary code.

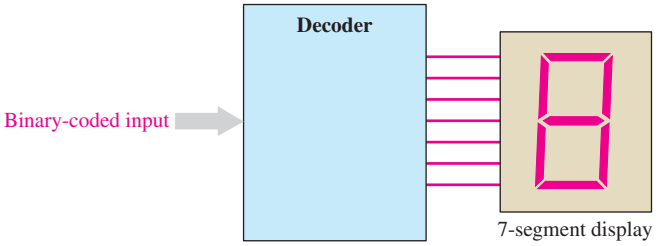


FIGURE 1–29 **A decoder used to convert a special binary code into a 7-segment decimal readout.**

The Data Selection Function

Two types of circuits that select data are the multiplexer and the demultiplexer. The **multiplexer,** or mux for short, is a logic circuit that switches digital data from several input lines onto a single output line in a specified time sequence. Functionally, a multiplexer can be represented by an electronic switch operation that sequentially connects each of the input lines to the output line. The **demultiplexer** (demux) is a logic circuit that switches digital data from one input line to several output lines in a specified time sequence. Essentially, the demux is a mux in reverse.

Multiplexing and demultiplexing are used when data from several sources are to be transmitted over one line to a distant location and redistributed to several destinations. Figure 1–30 illustrates this type of application where digital data from three sources are sent out along a single line to three terminals at another location.

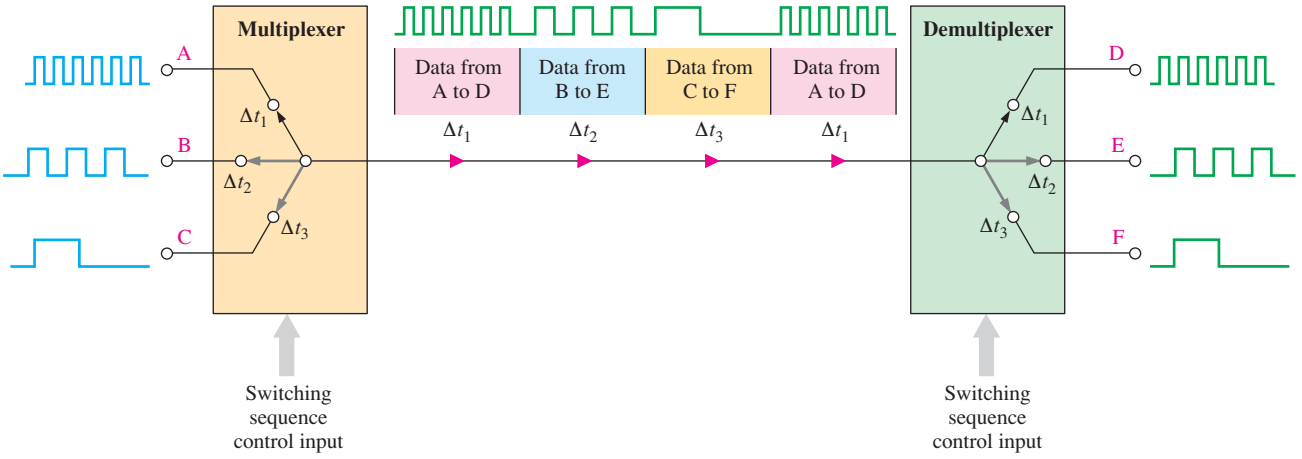


FIGURE 1–30 **Illustration of a basic multiplexing/demultiplexing application.**

In Figure 1–30, data from input A are connected to the output line during time interval Δt_1 and transmitted to the demultiplexer that connects them to output D. Then, during interval Δt_2, the multiplexer switches to input B and the demultiplexer switches to output E. During interval Δt_3, the multiplexer switches to input C and the demultiplexer switches to output F.

To summarize, during the first time interval, input A data go to output D. During the second time interval, input B data go to output E. During the third time interval, input C data go to output F. After this, the sequence repeats. Because the time is divided up among

several sources and destinations where each has its turn to send and receive data, this process is called *time division multiplexing* (TDM).

The Memory and Storage Functions

Memory and **storage** are functions that are required in most digital systems, and their purpose is to retain binary data for a period of time. Generally, *memory* refers to relatively short-term data retention, and *storage* refers to long-term data retention. A storage device can "memorize" a bit or a group of bits and retain the information as long as necessary. Memories include flip-flops, registers, and semiconductor memory. Storage includes magnetic disks (hard drives), optical disks (CDs), and magnetic tape.

FLIP-FLOPS A **flip-flop** is a bistable (two stable states) logic circuit that can store only one bit at a time, either a 1 or a 0. The output of a flip-flop indicates which bit it is storing. A HIGH output indicates that a 1 is stored and a LOW output indicates that a 0 is stored. Flip-flops are implemented with logic gates and are covered in Chapter 6.

REGISTERS A **register** is formed by combining several flip-flops so that groups of bits can be stored. For example, an 8-bit register is constructed from eight flip-flops. In addition to storing bits, registers can be used to shift the bits from one position to another within the register or out of the register to another circuit; therefore, these devices are known as *shift registers.*

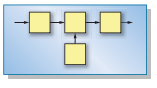
The two basic types of shift registers are serial and parallel. The bits are stored in a serial shift register one at a time. A good analogy to the serial shift register is loading passengers onto a bus single file through the door. They also exit the bus single file. The bits are stored in a parallel register simultaneously from parallel lines. For this case, a good analogy is loading and unloading passengers on a roller coaster where they enter all of the cars in parallel and exit in parallel.

SEMICONDUCTOR MEMORIES Semiconductor memories are devices typically used for storing large numbers of bits. In one type of memory, called the *r*ead-*o*nly *m*emory or ROM, the binary data are permanently or semipermanently stored and cannot be readily changed. In the *r*andom-*a*ccess *m*emory or RAM, the binary data are temporarily stored and can be easily changed. Memories are covered in Chapter 10.

MAGNETIC MEMORIES Magnetic disk memories are used for mass storage of binary data. An example is the computer's internal hard disk. Magneto-optical disks use laser beams to store and retrieve data. Magnetic tape is still used in memory applications and for backing up data from other storage devices.

The internal computer memories, RAM and ROM, as well as the smaller caches are semiconductor memories. The registers in a microprocessor are constructed of semiconductor flip-flops. Opto-magnetic disk memories are used in the internal hard drive and for the CD-ROM.

SYSTEM NOTE



The Counting Function

A **counter** is a sequential device and is a type of state machine because it has a unique internal sequence of states. The counting function is important in digital systems. There are many types of digital counters, but their basic purpose is to count events or to generate sequences represented by changing levels or pulses. To count, the counter must "remember" the present number so that it can go to the next proper number in sequence. Therefore, storage capability is an important characteristic of all counters, and flip-flops are generally used to implement them. Figure 1–31 illustrates the basic idea of counter operation.

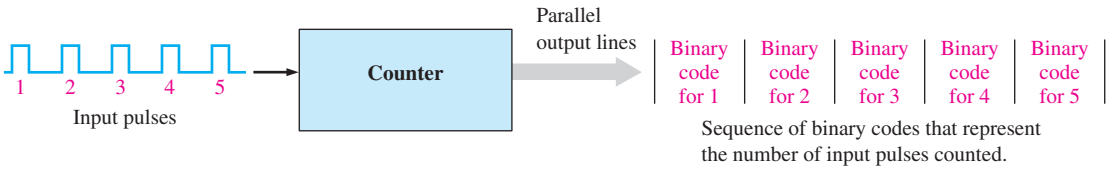


FIGURE 1–31 **Illustration of basic counter operation.**

SECTION 1–4 CHECKUP

1. What does a comparator do?
2. What are the four basic arithmetic operations?
3. Describe encoding and give an example.
4. Describe decoding and give an example.
5. Explain the basic purpose of multiplexing and demultiplexing.
6. Name four types of memory and storage devices.
7. What does a counter do?

1–5 PROGRAMMABLE LOGIC

A programmable logic device (PLD) is a type of integrated circuit (IC) that starts as a "blank slate" and into which a logic design is programmed. Programmable logic requires both hardware and software. PLDs can be programmed to perform specified logic functions by the manufacturer or by the user. One advantage of programmable logic over fixed-function logic is that the devices use much less board space for an equivalent amount of logic. Another advantage is that, with programmable logic, designs can be readily changed without rewiring or replacing components. Also, a logic design can generally be implemented faster and with less cost with programmable logic than with fixed-function ICs.

After completing this section, you should be able to

- **State the major types of programmable logic and discuss the differences**
- **Discuss methods of programming**
- **List the major programming languages used for programmable logic**
- **Discuss the programmable logic design process**

Types of Programmable Logic Devices (PLDs)

Many types of programmable logic devices are available, ranging from small devices that can replace a few fixed-function devices to complex high-density devices that can replace thousands of fixed-function devices. Two major categories of user-programmable logic are **PLD** (programmable logic device) and **FPGA** (field-programmable gate array), as indicated in Figure 1–32. PLDs are either SPLDs (simple PLDs) or CPLDs (complex PLDs).

SIMPLE PROGRAMMABLE LOGIC DEVICE (SPLD) The SPLD was the original PLD and is still available for small-scale applications. Generally, an **SPLD** can replace up to ten fixed-function ICs and their interconnections, depending on the type of functions and the specific SPLD. Most SPLDs are in one of two categories: PAL and GAL. A **PAL** (programmable array logic) is a device that can be programmed one time. It consists of a programmable array of AND gates and a fixed array of OR gates, as shown in Figure 1–33(a). A **GAL** (generic array logic) is a device that is basically a PAL that can be

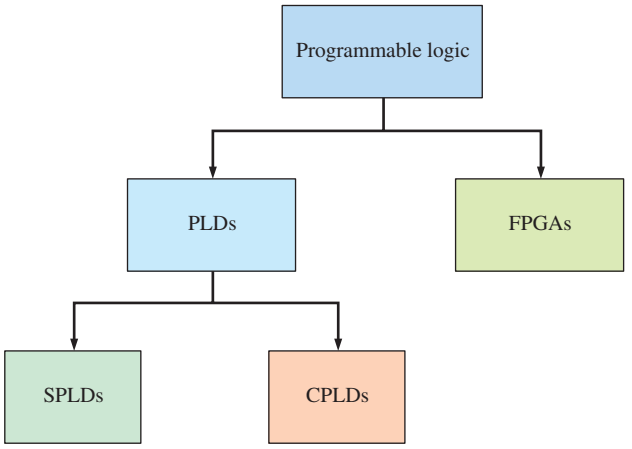


FIGURE 1–32 **Programmable logic hierarchy.**

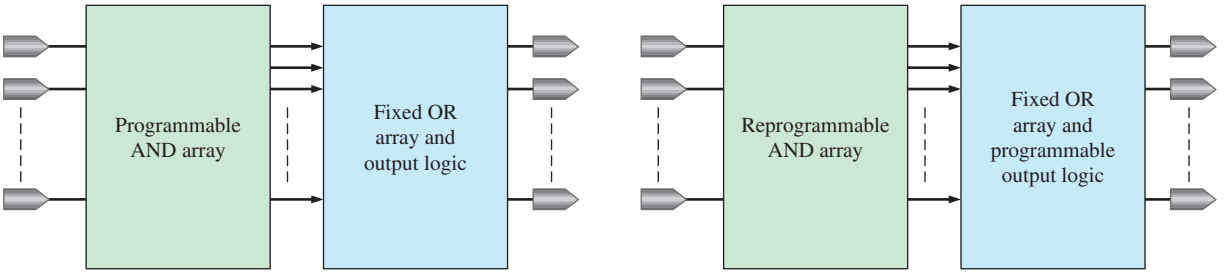


FIGURE 1–33 **Block diagrams of simple programmable logic devices (SPLDs).**

reprogrammed many times. It consists of a reprogrammable array of AND gates and a fixed array of OR gates with programmable ouputs, as shown in Figure 1–33(b). A typical SPLD package is shown in Figure 1–34 and generally has from 24 to 28 pins.

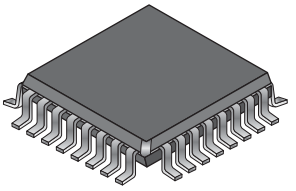


FIGURE 1–34 **A typical SPLD package.**

COMPLEX PROGRAMMABLE LOGIC DEVICE (CPLD) As technology progressed and the amount of circuitry that could be put on a chip (chip density) increased, manufacturers were able to put more than one SPLD on a single chip and the CPLD was born. Essentially, the **CPLD** is a device containing multiple SPLDs and can replace many fixed-function ICs. Figure 1–35 shows a basic CPLD block diagram with four logic array

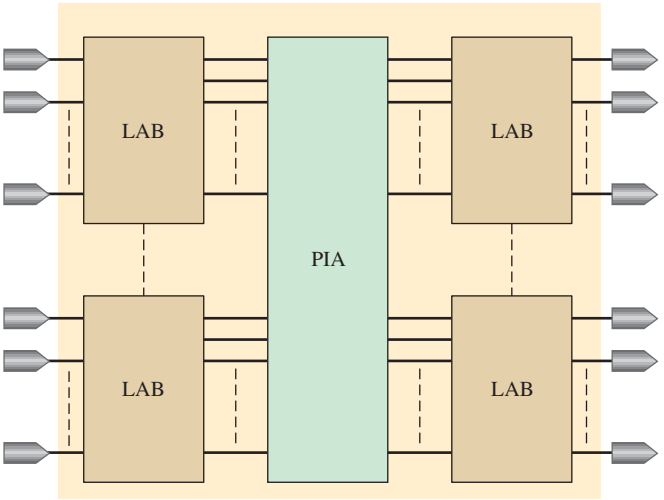
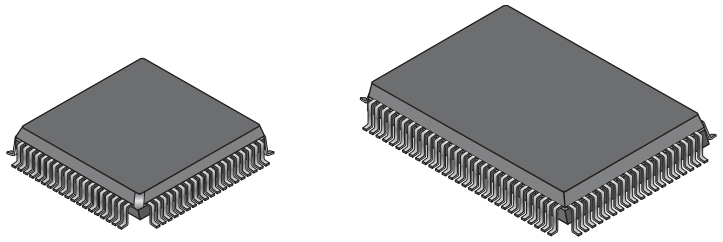


FIGURE 1–35 **General block diagram of a CPLD.**

blocks (LABs) and a programmable interconnection array (PIA). Depending on the specific CPLD, there can be from two to sixty-four LABs. Each logic array block is roughly equivalent to one SPLD.

Generally, CPLDs can be used to implement any of the logic functions discussed earlier, for example, decoders, encoders, multiplexers, demultiplexers, and adders. They are available in a variety of configurations, typically ranging from 44 to 160 pin packages. Examples of CPLD packages are shown in Figure 1–36.



(a) 80-pin PQFP (b) 128-pin PQFP

FIGURE 1–36 **Typical CPLD packages.**

FIELD-PROGRAMMABLE GATE ARRAY (FPGA) An **FPGA** is generally more complex and has a much higher density than a CPLD, although their applications can sometimes overlap. As mentioned, the SPLD and the CPLD are closely related because the CPLD basically contains a number of SPLDs. The FPGA, however, has a different internal structure (architecture), as illustrated in Figure 1–37. The three basic elements in an FPGA are the logic block, the programmable interconnections, and the input/output (I/O) blocks.

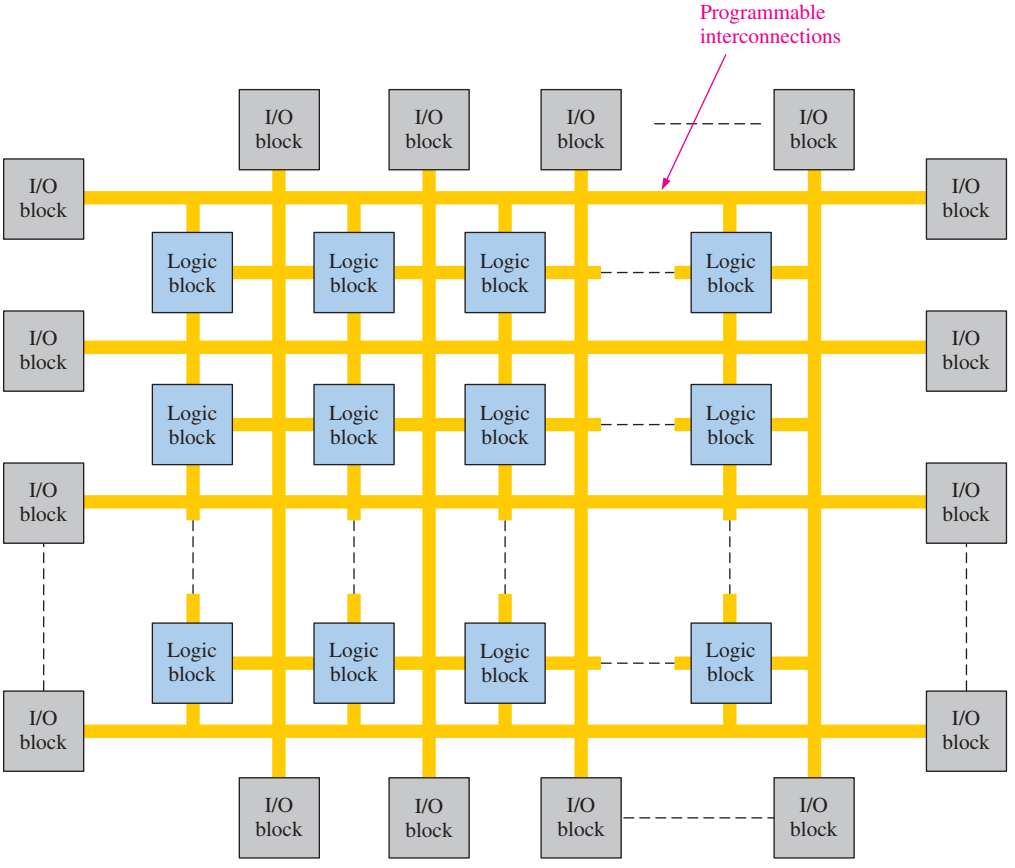
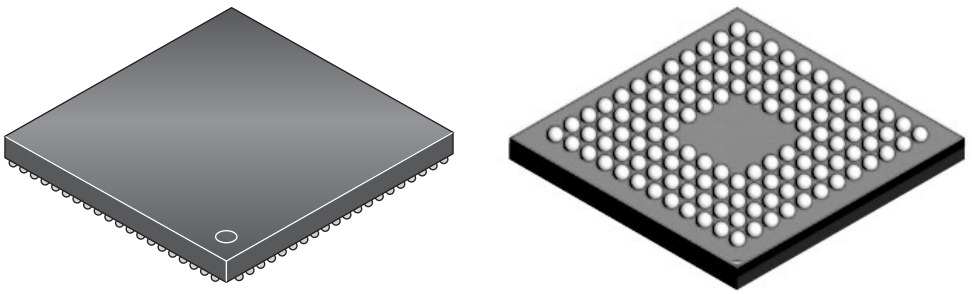


FIGURE 1–37 **Basic structure of an FPGA.**

The logic blocks in an FPGA are not as complex as the logic array blocks (LABs) in a CPLD, but generally there are many more of them. When the logic blocks are relatively simple, the FPGA architecture is called *fine-grained.* When the logic blocks are larger and more complex, the architecture is called *coarse-grained.* The I/O blocks are on the outer edges of the structure and provide individually selectable input, output, or bidirectional access to the outside world. The distributed programmable interconnection matrix provides for interconnection of the logic blocks and connection to inputs and outputs. Large FPGAs can have tens of thousands of logic blocks in addition to memory and other resources. A typical FPGA ball-grid array package is shown in Figure 1–38. These types of packages can have over 1000 input and output pins.



(a) Top view (b) Bottom view

FIGURE 1–38 **A typical ball-grid array (BGA) package.**

The Programming Process

An SPLD, CPLD, or FPGA can be thought of as a "blank slate" on which you implement a specified system design using a certain process. This process requires a software development package installed on a computer to implement a circuit design in the programmable chip. The computer must be interfaced with a development board or programming fixture containing the device, as illustrated in Figure 1–39.

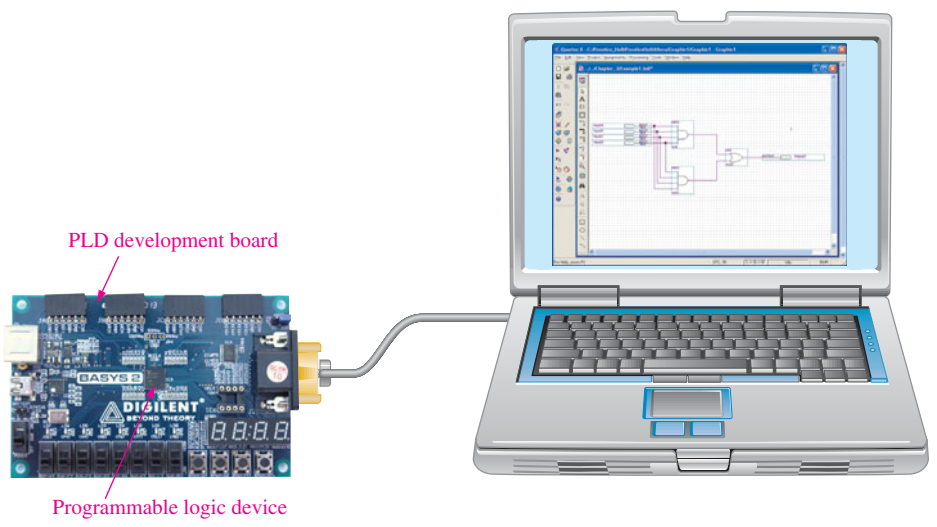


FIGURE 1–39 **Basic setup for programming a PLD or FPGA.** **(Photo courtesy of Digilent, Inc.)**

Several steps, called the *design flow,* are involved in the process of implementing a digital logic design in a programmable logic device. A block diagram of a typical programming process is shown in Figure 1–40. As indicated, the design flow has access to a design library.

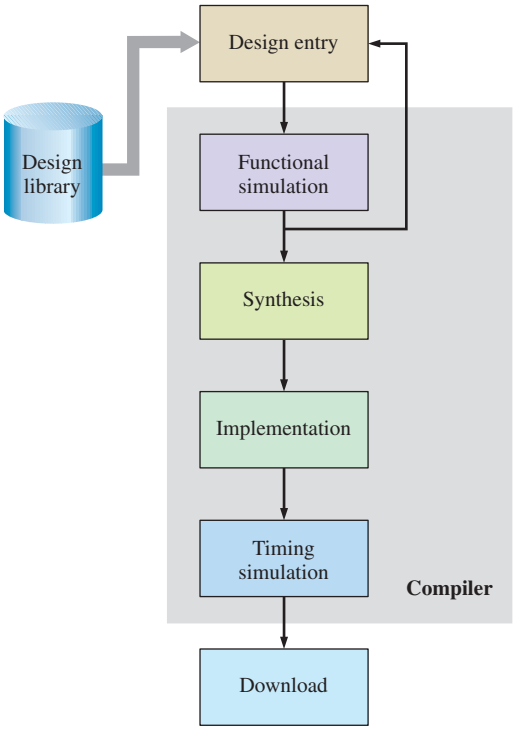


FIGURE 1–40 **Basic programming flow block diagram.**

DESIGN ENTRY This is the first programming step. The circuit or system design must be entered into the design application software using text-based entry, graphic entry (schematic capture), or state diagram description. Design entry is device independent. Text-based entry is accomplished with a hardware description language (HDL) such as VHDL, Verilog, or AHDL. Graphic (schematic) entry allows prestored logic functions from a library to be selected, placed on the screen, and then interconnected to create a logic design. State-diagram entry requires specification of both the states through which a sequential logic circuit progresses and the conditions that produce each state change.

Once a design has been entered, it is compiled. A **compiler** is a program that controls the design flow process and translates source code into object code in a format that can be logically tested or downloaded to a target device. The source code is created during design entry, and the object code is the final code that actually causes the design to be implemented in the programmable device.

FUNCTIONAL SIMULATION The entered and compiled design is simulated by software to confirm that the logic circuit functions as expected. The functional simulation will verify that correct outputs are produced for a specified set of inputs. A device-independent software tool for doing this is generally called a *waveform editor.* Any flaws demonstrated by the simulation would be corrected by going back to design entry and making appropriate changes.

SYNTHESIS **Synthesis** is where the design is translated into a netlist, which has a standard form and is device independent.

IMPLEMENTATION **Implementation** is where the logic structures described by the netlist are mapped into the actual structure of the specific device being programmed. The implementation process is called *fitting* or *place and route* and results in an output called a bitstream, which is device dependent.

TIMING SIMULATION This step comes after the design is mapped into the specific device. The timing simulation is basically used to confirm that there are no design flaws or timing problems due to propagation delays.

DOWNLOAD Once a bitstream has been generated for a specific programmable device, it has to be downloaded to the device to implement the software design in hardware. Some programmable devices have to be installed in a special piece of equipment called a *device programmer* or on a development board. Other types of devices can be programmed while in a system—called in-system programming (ISP)—using a standard JTAG (Joint Test Action Group) interface. Some devices are volatile, which means they lose their contents when reset or when power is turned off. In this case, the bitstream data must be stored in a memory and reloaded into the device after each reset or power-off. Also, the contents of an ISP device can be manipulated or upgraded while it is operating in a system. This is called "on-the-fly" reconfiguration.

The Microcontroller

A microcontroller is different than a PLD. The internal circuits of a microcontroller are fixed, and a program (series of instructions) directs the microcontroller operation in order to achieve a specific outcome. The internal circuitry of a PLD is programmed into it, and once programmed, the circuitry performs required operations. Thus, a program determines microcontroller operation, but in a PLD a program determines the logic function. Microcontrollers are generally programmed with either the C language or the BASIC language.

A **microcontroller** is basically a special-purpose small computer. Microcontrollers are generally used for embedded system applications. An **embedded system** is one that is designed to perform one or a few dedicated functions. By contrast, a general-purpose computer, such as a laptop, is designed to perform a wide range of functions and applications.

Embedded microcontrollers are used in many common applications. The embedded microcontroller is part of a complete system, which may include additional electronics and mechanical parts. For example, a microcontroller in a television set displays the input from the remote unit on the screen and controls the channel selection, audio, and various menu adjustments like brightness and contrast. In an automobile a microcontroller takes engine sensor inputs and controls spark timing and fuel mixture. Other applications include home appliances, thermostats, cell phones, and toys.

SECTION 1–5 CHECKUP

1. List three major categories of programmable logic devices and specify their acronyms.
2. How does a CPLD differ from an SPLD?
3. Name the steps in the programming process.
4. Briefly explain each step named in question 3.
5. What are the two main functional characteristics of a microcontroller?

1–6 FIXED-FUNCTION LOGIC DEVICES

All the logic elements and functions that have been discussed are generally available in integrated circuit (IC) form. A fixed-function device is one that cannot be programmed like a PLD. Digital systems have incorporated ICs for many years because of their small size, high reliability, low cost, and low power consumption. It is important to be able to recognize the IC packages and to know how the pin connections are numbered, as well as to be familiar with the way in which circuit complexities and circuit technologies determine the various IC classifications.

After completing this section, you should be able to

- **Recognize the difference between through-hole devices and surface-mount fixed-function devices**
- **Identify dual in-line packages (DIP)**
- **Identify small-outline integrated circuit packages (SOIC)**
- **Identify plastic leaded chip carrier packages (PLCC)**
- **Identify leadless ceramic chip carrier packages (LCC)**
- **Determine pin numbers on various types of IC packages**
- **Explain the complexity classifications for fixed-function ICs**

A monolithic **integrated circuit (IC)** is an electronic circuit that is constructed entirely on a single small chip of silicon. All the components that make up the circuit—transistors, diodes, resistors, and capacitors—are an integral part of that single chip. Fixed-function logic and programmable logic are two broad categories of digital ICs. In **fixed-function logic**, the logic functions are set by the manufacturer and cannot be altered.

Figure 1–41 shows a cutaway view of one type of fixed-function IC package with the circuit chip shown within the package. Points on the chip are connected to the package pins to allow input and output connections to the outside world.

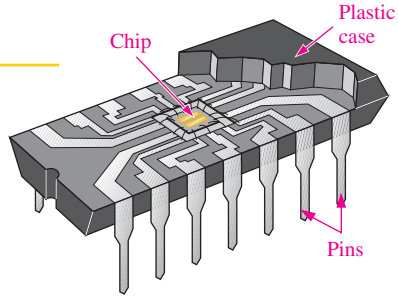
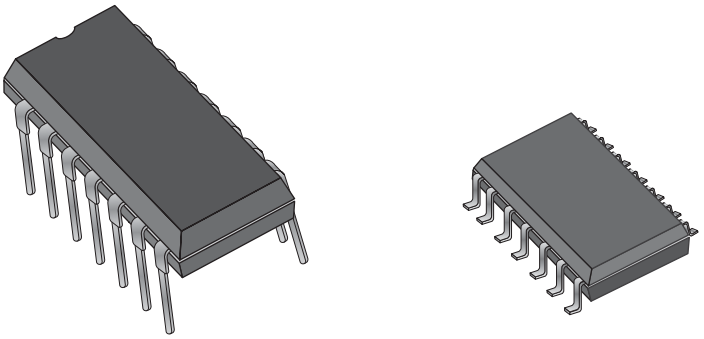


FIGURE 1–41 Cutaway view of one type of fixed-function surface-mount IC package, showing the chip mounted inside and connections to input and output pins.



(a) Dual in-line package (DIP) (b) Small-outline IC (SOIC)

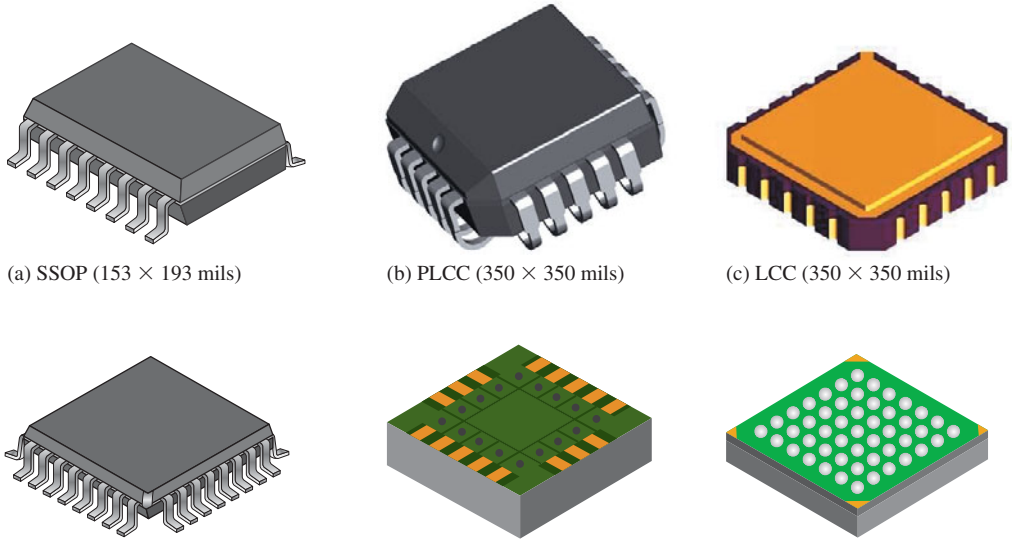
FIGURE 1–42 **Examples of through-hole (DIP) and surface-mounted devices. The DIP is larger than the SOIC with the same number of leads.**

IC Packages

Integrated circuit (IC) packages are classified according to the way they are mounted on printed circuit (PC) boards as either through-hole mounted or surface mounted. The through-hole type packages have pins (leads) that are inserted through holes in the PC board and can be soldered to conductors on the opposite side. The most common type of through-hole package is the dual in-line package (**DIP**) shown in Figure 1–42(a). The DIP is useful in lab work because it plugs in easily to a protoboard.

Most IC packages use surface-mount technology (**SMT**). Surface mounting is a space-saving alternative to through-hole mounting. The holes through the PC board are unnecessary for SMT. The pins of surface-mounted packages are soldered directly to conductors on one side of the board, leaving the other side free for additional circuits. Also, for a circuit with the same number of pins, a surface-mounted package is much smaller than a dual in-line package because the pins are placed closer together. An example of a surface-mounted package is the small-outline integrated circuit (**SOIC**) shown in Figure 1–42(b).

Various types of SMT packages are available in a range of sizes, depending on the number of leads (more leads are required for more complex circuits and lead configurations). Examples of several types are shown in Figure 1–43. As you can see, the leads of the **SSOP** (shrink small-outline package) are formed into a "gull-wing" shape. The leads of the **PLCC** (plastic-leaded chip carrier) are turned under the package in a J-type shape. Instead of leads, the **LCC** (leadless ceramic chip) has metal contacts molded into its ceramic body. The LQFP also has gull-wing leads. Both the CSP (chip scale package) and the FBGA (fine-pitch ball grid array) have contacts embedded in the bottom of the package.



(a) SSOP (153 × 193 mils) (b) PLCC (350 × 350 mils) (c) LCC (350 × 350 mils)

(d) LQFP (7 × 7 mm) (e) Laminate CSP (3.5 × 3.5 mm) (f) FBGA (4 × 4 mm)

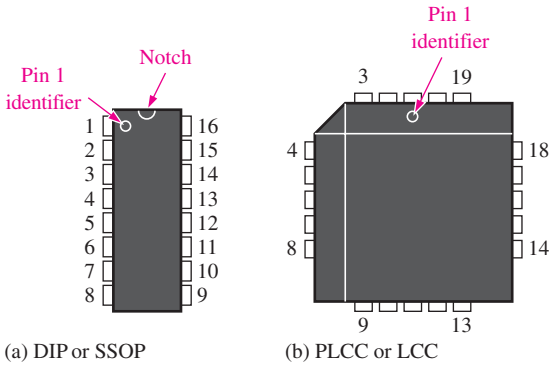
FIGURE 1–43 **Typical SMT package configurations. Parts (e) and (f) show bottom views.**

Pin Numbering

All IC packages have a standard format for numbering the pins (leads). The dual in-line packages (DIPs) and the shrink small-outline packages (SSOP) have the numbering arrangement illustrated in Figure 1–44(a) for a 16-pin package. Looking at the top of the package, pin 1 is indicated by an identifier that can be either a small dot, a notch, or a beveled

edge. The dot is always next to pin 1. Also, with the notch oriented upward, pin 1 is always the top left pin, as indicated. Starting with pin 1, the pin numbers increase as you go down, then across and up. The highest pin number is always to the right of the notch or opposite the dot.

The PLCC and LCC packages have leads arranged on all four sides. Pin 1 is indicated by a dot or other index mark and is located at the center of one set of leads. The pin numbers increase going counterclockwise as viewed from the top of the package. The highest pin number is always to the right of pin 1. Figure 1–44(b) illustrates this format for a 20-pin PLCC package.



(a) DIP or SSOP (b) PLCC or LCC

FIGURE 1–44 **Pin numbering for standard types of IC packages. Top views are shown.**

Complexity Classifications for Fixed-Function ICs

Fixed-function digital ICs are classified according to their complexity. They are listed here from the least complex to the most complex. The complexity figures stated here for SSI, MSI, LSI, VLSI, and ULSI are generally accepted, but definitions may vary from one source to another.

- **Small-scale integration (SSI)** describes fixed-function ICs that have up to ten equivalent gate circuits on a single chip, and they include basic gates and flip-flops.
- **Medium-scale integration (MSI)** describes integrated circuits that have from 10 to 100 equivalent gates on a chip. They include logic functions such as encoders, decoders, counters, registers, multiplexers, arithmetic circuits, small memories, and others.
- **Large-scale integration (LSI)** is a classification of ICs with complexities of from more than 100 to 10,000 equivalent gates per chip, including memories.
- **Very large-scale integration (VLSI)** describes integrated circuits with complexities of from more than 10,000 to 100,000 equivalent gates per chip.
- **Ultra large-scale integration (ULSI)** describes very large memories, larger **microprocessors,** and larger single-chip computers. Complexities of more than 100,000 equivalent gates per chip are classified as ULSI.

Integrated Circuit Technologies

The types of transistors with which all integrated circuits are implemented are either MOSFETs (metal-oxide semiconductor field-effect transistors) or bipolar junction transistors. A circuit technology that uses MOSFETs is **CMOS** (complementary MOS). **Bipolar** is a type of fixed-function digital circuit technology that uses bipolar junction transistors and is sometimes called **TTL** (transistor-transistor logic). **BiCMOS** uses a combination of both CMOS and bipolar. All the types of logic gates and logic functions that have been discussed are generally available as ICs.

All gates and other functions can be implemented with either type of circuit technology. SSI and MSI circuits are generally available in both CMOS and bipolar in the 74XX series, but CMOS is the most common.

SECTION 1–6 CHECKUP

1. What is an integrated circuit?

2. Define the terms DIP, SMT, SOIC, SSI, MSI, LSI, and VLSI.

3. Generally, in what classification does a fixed-function IC with the following number of equivalent gates fall?

(a) 10 **(b)** 75 **(c)** 500 **(d)** 15,000 **(e)** 200,000

1–7 A SYSTEM

A tablet-bottling system illustrates how the logic functions covered in this chapter can be used in a system environment. The functions used in this system are the encoder, decoder, code converter, adder, multiplexer, demultiplexer, register, and counter. This system could be implemented in three ways: with a PLD, with a microcontroller, or with fixed-function ICs. The first two are how all digital systems are currently implemented.

After completing this section, you should be able to

- **Understand basic system operation and how certain components work together**
- **Explain the purpose of each logic function in the total system**
- **Describe the transfer of digital data throughout the system**

A Process Control System

A system for bottling vitamin tablets is shown in the block diagram of Figure 1–45. To begin, the tablets are fed into a large funnel-type hopper. The narrow neck of the hopper creates a serial flow of tablets into a bottle on the conveyor belt below. Only one tablet at a time passes the sensor, so the tablets can be counted.

The system controls the number of tablets into each bottle and displays a continually updated readout of the total number of tablets bottled. This system utilizes all of the basic logic functions that have been introduced and illustrates how these functions can be connected to work together to produce a specified result. This system is purely for instructional purposes and is not intended to necessarily represent the most efficient or best way to implement the operation.

GENERAL OPERATION The maximum number of tablets per bottle is entered from the keypad, changed to a code by the *Encoder,* and stored in *Register A. Decoder A* changes the code stored in the register to a form appropriate for turning on the display. *Code converter A* changes the code to a binary number and applies it to the *A* input of the *Comparator* (Comp).

An optical sensor in the neck of the hopper detects each tablet that passes and produces a pulse. This pulse goes to the *Counter* and advances it by one count; thus, any time during the filling of a bottle, the binary state of the counter represents the number of tablets in the bottle. The binary count is transferred from the counter to the *B* input of the comparator (Comp). The *A* input of the comparator is the binary number for the maximum tablets per bottle. Now, let's say that the present number of tablets per bottle is 50. When the binary number in the counter reaches 50, the $A = B$ output of the comparator goes HIGH, indicating that the bottle is full.

The HIGH output of the comparator causes the valve in the neck of the hopper to close and stop the flow of tablets. At the same time, the HIGH output of the comparator activates the conveyor, which moves the next empty bottle into place under the hopper. When the bottle is in place, the conveyor control issues a pulse that resets the counter to zero. As a result, the output of the comparator goes back LOW and causes the hopper valve to restart the flow of tablets.

For each bottle filled, the maximum binary number in the counter is transferred to the *A* input of the *Adder.* The *B* input of the adder comes from *Register B* that stores the total number of tablets bottled up through the last bottle filled. The adder produces a new cumulative sum that is then stored in register B, replacing the previous sum. This keeps a running total of the tablets bottled during a given run.

The cumulative sum stored in register B goes to *Decoder B*, which detects when register B has reached its maximum capacity and enables the *MUX*. The binary sum in register B is converted from parallel to serial form by the MUX and transmitted over the single line to the remote *Demultiplexer* (DEMUX), which changes the number back to parallel form for storage in a remote computer for keeping track of the total tablets bottled in a specified time period.

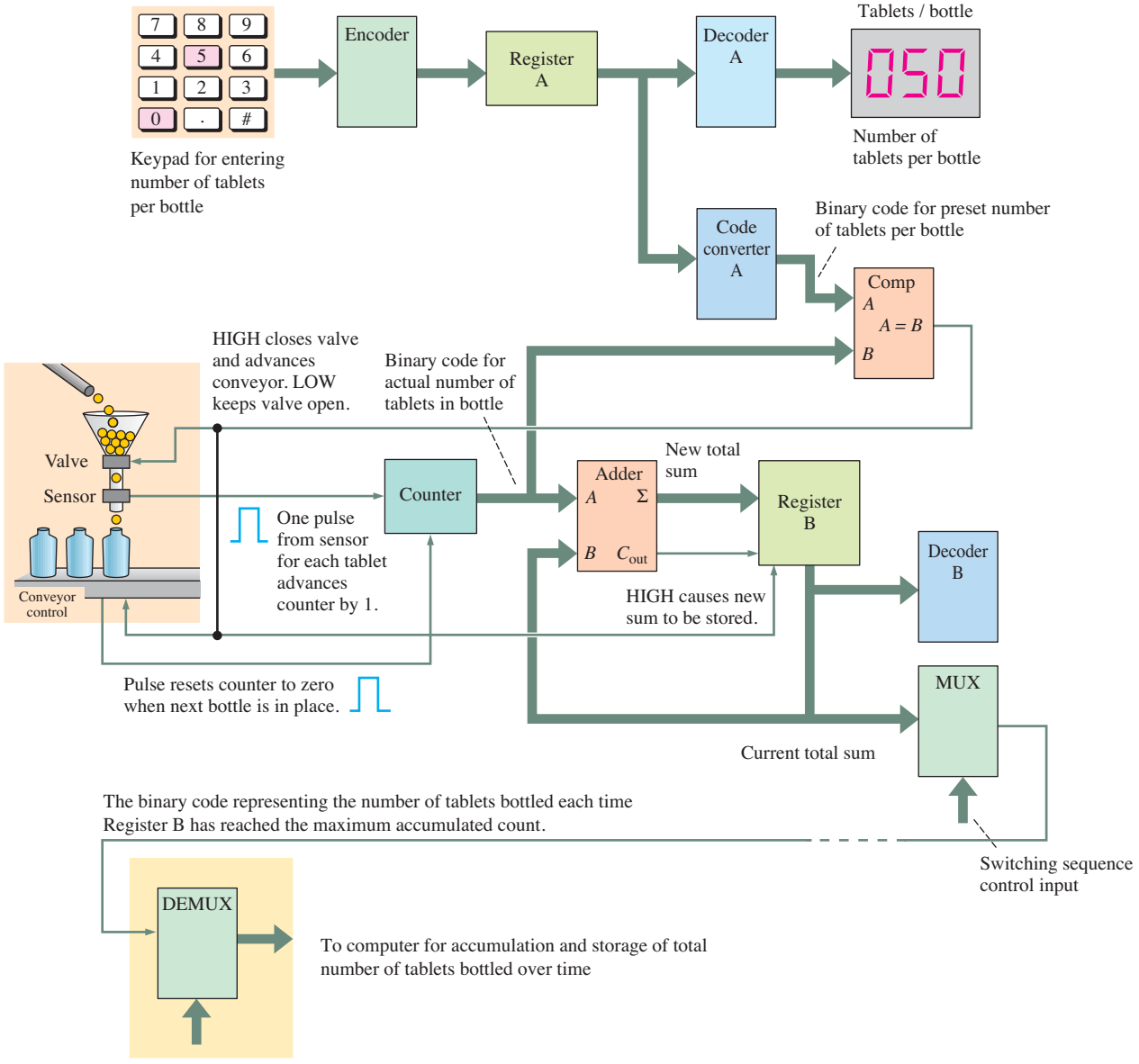


FIGURE 1–45 **Block diagram of a tablet-bottling system.**

SECTION 1–7 CHECKUP

1. How is the number of tablets per bottle entered into the system?
2. How does the system determine when a bottle is full?
3. When is the counter reset?

1–8 MEASURING INSTRUMENTS

Troubleshooting is the process of systematically isolating, identifying, and correcting a fault in a circuit or system. A variety of instruments are available for use in troubleshooting and testing. Some common types of instruments are introduced and discussed in this section.

After completing this section, you should be able to

- **Distinguish between an analog and a digital oscilloscope**
- **Recognize common oscilloscope controls**
- **Determine amplitude, period, frequency, and duty cycle of a pulse waveform with an oscilloscope**
- **Discuss the logic analyzer and some common formats**
- **Describe the purpose of the data pattern generator, the digital multimeter (DMM), the dc power supply, the logic probe, and the logic pulser**

The Oscilloscope

The oscilloscope (scope for short) is one of the most widely used instruments for general testing and troubleshooting. The scope is basically a graph-displaying device that traces the graph of a measured electrical signal on its screen. In most applications, the graph shows how signals change over time. The vertical axis of the display screen represents voltage, and the horizontal axis represents time. Amplitude, period, and frequency of a signal can be measured using the oscilloscope. Also, the pulse width, duty cycle, rise time, and fall time of a pulse waveform can be determined. Most scopes can display at least two, and many can display four signals on the screen at one time, enabling their time relationship to be observed. A typical 4-channel digital oscilloscope is shown in Figure 1–46.

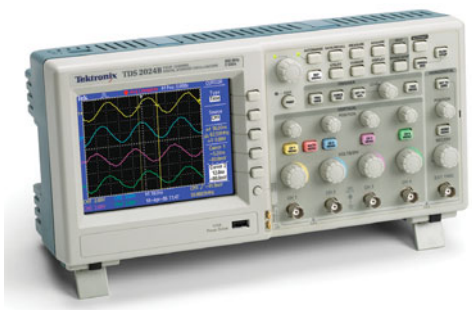


FIGURE 1–46 **A digital oscilloscope.** **Used with permission from Tektronix, Inc.**

Two basic types of oscilloscopes, analog and digital, can be used to view digital waveforms. An analog scope works by applying the measured waveform directly to control the up and down motion of the electron beam in the cathode-ray tube (CRT) as it sweeps across the display screen. As a result, the beam traces out the waveform pattern on the screen. A digital scope converts the measured waveform to digital information by a sampling process in an analog-to-digital converter (ADC). The digital information is then used to reconstruct the waveform on the screen.

The digital scope is more widely used than the analog scope. However, either type can be used in many applications; each has characteristics that make it more suitable for certain situations. An analog scope displays waveforms as they occur in "real time." Digital scopes are useful for measuring transient pulses that may occur randomly or only once. Also, because information about the measured waveform can be stored in a digital scope, it may be viewed at some later time, printed out, or thoroughly analyzed by a computer or other means.

BASIC OPERATION OF ANALOG OSCILLOSCOPES To measure a voltage, a **probe** must be connected from the scope to the point in a circuit at which the voltage is present. Generally, a ×10 probe is used that reduces (attenuates) the signal amplitude by ten. The signal goes through the probe into the vertical circuits where it is either further attenuated or amplified, depending on the actual amplitude and on where you set the vertical control of the scope. The vertical circuits then drive the vertical deflection plates of the CRT. Also, the signal goes to the trigger circuits that trigger the horizontal circuits to initiate repetitive horizontal sweeps of the electron beam across the screen using a sawtooth waveform. There are many sweeps per second so that the beam appears to form a solid line across the screen in the shape of the waveform. This basic operation is illustrated in Figure 1–47.

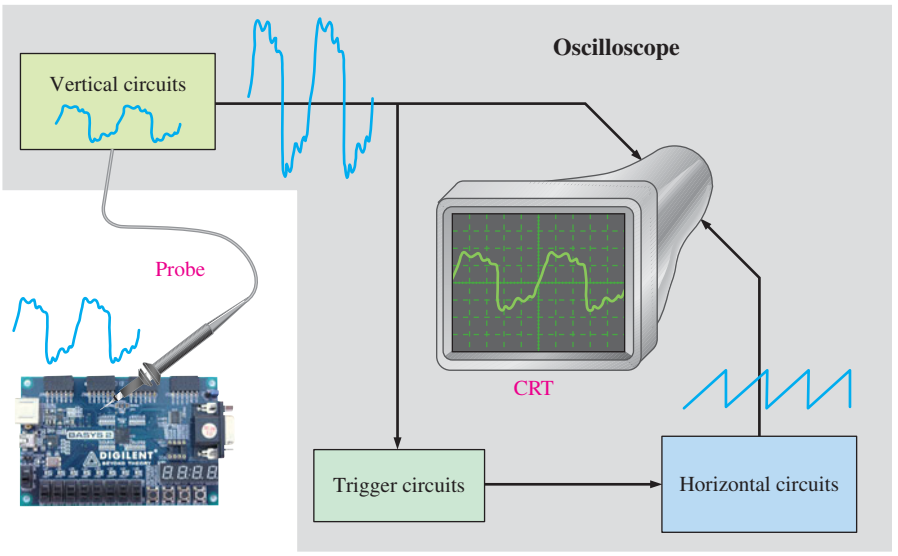


FIGURE 1–47 **Block diagram of an analog oscilloscope.** **(Photo courtesy of Digilent, Inc.)**

BASIC OPERATION OF DIGITAL OSCILLOSCOPES Some parts of a digital scope are similar to the analog scope. However, the digital scope is more complex than an analog scope and typically has an LCD screen rather than a CRT. Rather than displaying a waveform as it occurs, the digital scope first acquires the measured analog waveform and converts it to a digital format using an analog-to-digital converter (ADC). The digital data is stored and processed. The data then goes to the reconstruction and display circuits for display in its original analog form. Figure 1–48 shows a basic block diagram for a digital oscilloscope.

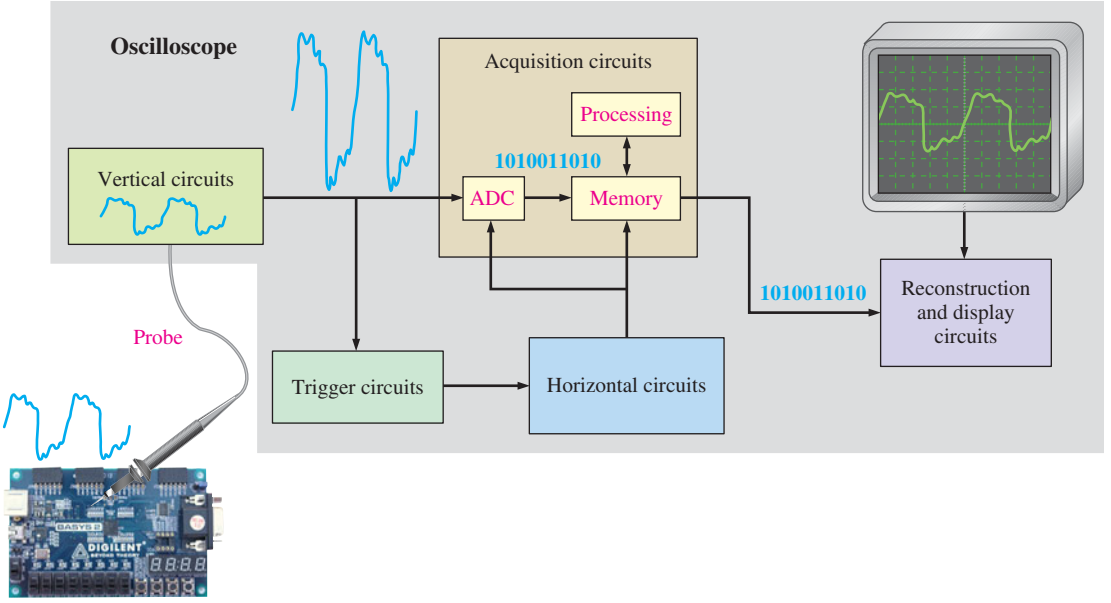


FIGURE 1–48 **Block diagram of a digital oscilloscope.** **(Photo courtesy of Digilent, Inc.)**

OSCILLOSCOPE CONTROLS A front panel view of a typical digital oscilloscope is shown in Figure 1–49. Instruments vary depending on model and manufacturer, but most have certain common features. For example, the four vertical sections contain a Position control, a channel menu button, and a volts/div control. The horizontal section contains a sec/div control.

Some of the main oscilloscope controls are now discussed. Refer to the user manual for complete details of your particular scope.

VERTICAL CONTROLS In the vertical section of the scope in Figure 1–49, there are identical controls for each of the four channels (1, 2, 3, and 4). The Position control lets you move a displayed waveform up or down vertically on the screen. The buttons on the right side of the screen provide for the selection of several items that appear on the screen, such as the coupling modes (ac, dc, or ground), coarse or fine adjustment for the volts/div, signal inversion, and other parameters. The volts/div control adjusts the number of volts represented by each vertical division on the screen. The volts/div setting for each channel is displayed on the bottom of the screen.

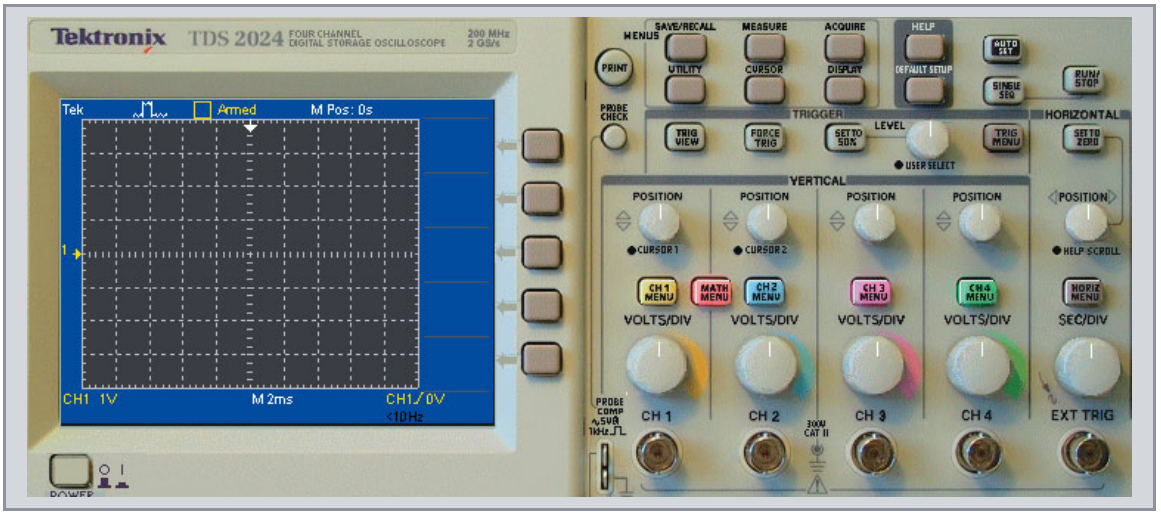
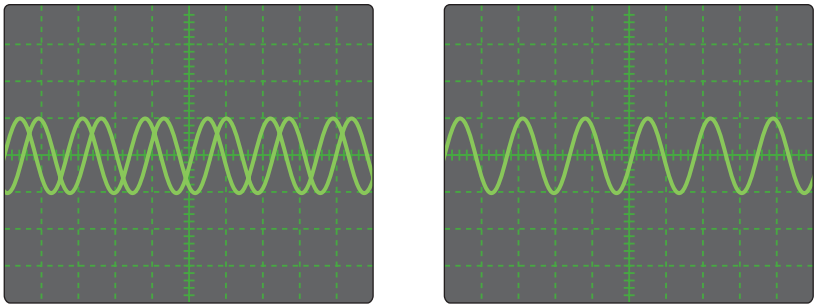


FIGURE 1–49 **A digital oscilloscope front panel.** Used with permission from Tektronix, Inc.

HORIZONTAL CONTROLS In the horizontal section, the controls apply to all channels. The Position control lets you move a displayed waveform left or right horizontally on the screen. The sec/div control adjusts the time represented by each horizontal division or main time base. The sec/div setting is displayed at the bottom of the screen.

TRIGGER CONTROLS In the Trigger control section, the Level control determines the point on the triggering waveform where triggering occurs to initiate the sweep to display input waveforms. The Menu button provides for the selection of several items that appear on the screen, including edge or slope triggering, trigger source, trigger mode, and other parameters. There is also an input for an external trigger signal.

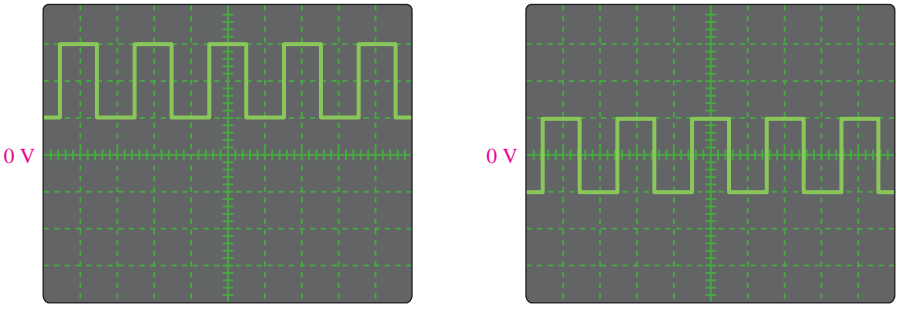
Triggering stabilizes a waveform on the screen or properly triggers on a pulse that occurs only one time or randomly. Also, it allows you to observe time delays between two waveforms. Figure 1–50 compares a triggered to an untriggered signal. The untriggered signal tends to drift across the screen, producing what appears to be multiple waveforms.



(a) Untriggered waveform display (b) Triggered waveform display

FIGURE 1–50 **Comparison of an untriggered and a triggered waveform on an oscilloscope.**

COUPLING A SIGNAL INTO THE SCOPE Coupling is the method used to connect a signal voltage to be measured into the oscilloscope. DC and AC coupling are usually selected from the Vertical menu on a scope. DC coupling allows a waveform including its dc component to be displayed. AC coupling blocks the dc component of a signal so that you see the waveform centered at 0 V. The Ground mode allows you to connect the channel input to ground to see where the 0 V reference is on the screen. Figure 1–51 illustrates the result of DC and AC coupling using a pulse waveform that has a dc component.

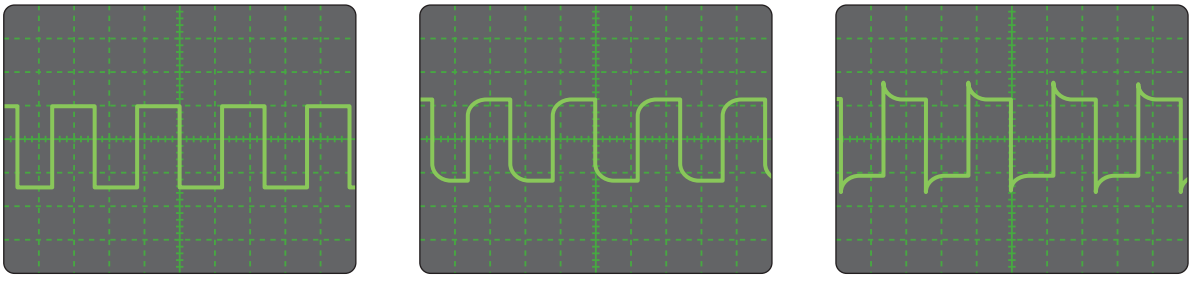


(a) DC coupled waveform (b) AC coupled waveform

FIGURE 1–51 **Displays of the same waveform having a dc component.**

The voltage probe, shown in Figure 1–46, is essential for connecting a signal to the scope. Since all instruments tend to affect the circuit being measured due to loading, most scope probes provide a high series resistance to minimize loading effects. Probes that have a series resistance ten times larger than the input resistance of the scope are called ×10 probes. Probes with no series resistance are called ×1 probes. The oscilloscope adjusts its calibration for the attenuation of the type of probe being used. For most measurements, the ×10 probe should be used. However, if you are measuring very small signals, a ×1 may be the best choice.

The probe has an adjustment that allows you to compensate for the input capacitance of the scope. Most scopes have a probe compensation output that provides a calibrated square wave for probe compensation. Before making a measurement, you should make sure that the probe is properly compensated to eliminate any distortion introduced. Typically, there is a screw or other means of adjusting compensation on a probe. Figure 1–52 shows scope waveforms for three probe conditions: properly compensated, undercompensated, and overcompensated. If the waveform appears either over- or undercompensated, adjust the probe until the properly compensated square wave is achieved.



Properly compensated Undercompensated Overcompensated

FIGURE 1–52 **Probe compensation conditions.**

EXAMPLE 1–3

Based on the readouts, determine the amplitude and the period of the pulse waveform on the screen of a digital oscilloscope as shown in Figure 1–53. Also, calculate the frequency.

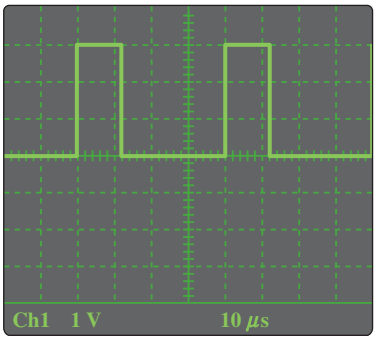


FIGURE 1–53

SOLUTION

The volts/div setting is 1 V. The pulses are three divisions high. Since each division represents 1 V, the pulse amplitude is

$$\text{Amplitude} = (3\text{ div})(1\text{ V/div}) = \mathbf{3\ V}$$

The sec/div setting is 10 μs. A full cycle of the waveform (from beginning of one pulse to the beginning of the next) covers four divisions; therefore, the period is

$$\text{Period} = (4\text{ div})(10\ \mu\text{s/div}) = \mathbf{40\ \mu s}$$

The frequency is calculated as

$$f = \frac{1}{T} = \frac{1}{40\ \mu\text{s}} = \mathbf{25\ kHz}$$

RELATED PROBLEM

For a volts/div setting of 4 V and sec/div setting of 2 ms, determine the amplitude and period of the pulse shown on the screen in Figure 1–53.

FIGURE 1–54 **Typical logic analyzer.** Used with permission from Tektronix, Inc.

The Logic Analyzer

Logic analyzers are used for measurements of multiple digital signals and measurement situations with difficult trigger requirements. Basically, the logic analyzer came about as a result of microprocessors in which troubleshooting or debugging required many more inputs than an oscilloscope offered. Many oscilloscopes have two input channels and some are available with four. Logic analyzers are available with from 34 to 136 input channels. Generally, an oscilloscope is used either when amplitude, frequency, and other timing parameters of a few signals at a time or when parameters such an rise and fall times, overshoot, and delay times need to be measured. The logic analyzer is used when the logic levels of a large number of signals need to be determined and for the correlation of simultaneous signals based on their timing relationships. A typical logic analyzer is shown in Figure 1–54, and a simplified block diagram is in Figure 1–55.

DATA ACQUISITION The large number of signals that can be acquired at one time is a major factor that distinguishes a logic analyzer from an oscilloscope. Generally, the two types of data acquisition in a logic analyzer are the timing acquisition and the state acquisition. Timing acquisition is used primarily when the timing relationships among the various signals need to be determined. State acquisition is used when you need to view the sequence of states as they appear in a system under test.

It is often helpful to have correlated timing and state data, and most logic analyzers can simultaneously acquire that data. For example, a problem may initially be detected as an invalid state. However, the invalid condition may be caused by a timing violation in the system under test. Without both types of information available at the same time, isolating the problem could be very difficult.

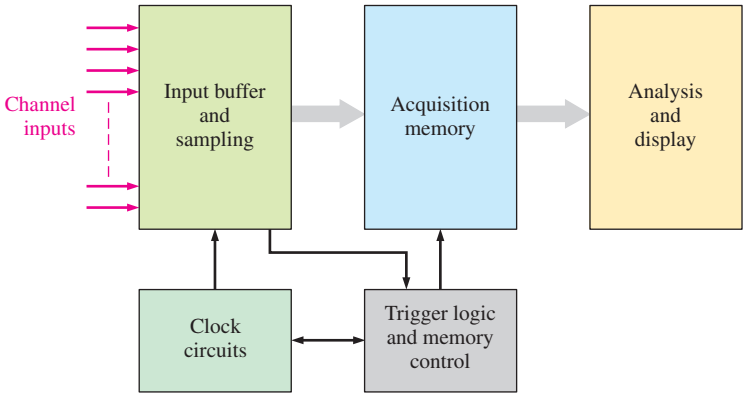
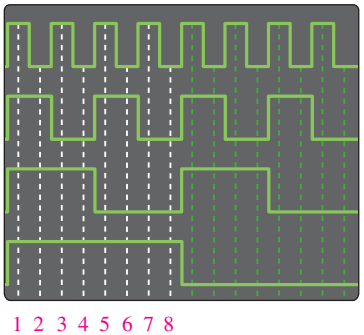


FIGURE 1–55 **Simplified block diagram of a logic analyzer.**

CHANNEL COUNT AND MEMORY DEPTH Logic analyzers contain a real-time acquisition memory in which sampled data from all the channels are stored as they occur. Two features that are of primary importance are the channel count and the memory depth. The acquisition memory can be thought of as having a width equal to the number of channels and a depth that is the number of bits that can be captured by each channel during a certain time interval.

Channel count determines the number of signals that can be acquired simultaneously. In certain types of systems, a large number of signals are present, such as on the data bus in a microprocessor-based system. The depth of the acquisition memory determines the amount of data from a given channel that you can view at any given time.

ANALYSIS AND DISPLAY Once data has been sampled and stored in the acquisition memory, it can typically be used in several different display and analysis modes. The waveform display is much like the display on an oscilloscope where you can view the time relationship of multiple signals. The listing display indicates the state of the system under test by showing the values of the input waveforms (1s and 0s) at various points in time (sample points). Typically, this data can be displayed in hexadecimal or other formats. Figure 1–56 shows simplified versions of these two display modes. The listing display samples correspond to the sampled points shown in red on the waveform display. You will study binary and hexadecimal (hex) numbers in the next chapter.



(a) Waveform display

Sample	Binary	Hex	Time
1	1111	F	1 ns
2	1110	E	10 ns
3	1101	D	20 ns
4	1100	C	30 ns
5	1011	B	40 ns
6	1010	A	50 ns
7	1001	9	60 ns
8	1000	8	70 ns

(b) Listing display

FIGURE 1–56 **Two logic analyzer display modes.**

Two more modes that are useful in computer and microprocessor-based system testing are the instruction trace and the source code debug. The instruction trace determines and displays instructions that occur, for example, on the data bus in a microprocessor-