Contents

1 Digital Concepts 2

1–1 Digital and Analog Quantities 4
1–2 Binary Digits, Logic Levels, and Digital Waveforms 6
1–3 Basic Logic Operations 12
1–4 Overview of Basic Logic Functions 14
1–5 Fixed-Function Integrated Circuits 19
1–6 Introduction to Programmable Logic 22
1–7 Test and Measurement Instruments 27
Digital System Application 38

2 Number Systems, Operations, and Codes 46

2–1 Decimal Numbers 48
2–2 Binary Numbers 50
2–3 Decimal-to-Binary Conversion 53
2–4 Binary Arithmetic 56
2–5 1's and 2's Complements of Binary Numbers 60
2–6 Signed Numbers 62
2–7 Arithmetic Operations with Signed Numbers 68
2–8 Hexadecimal Numbers 75
2–9 Octal Numbers 82
2–10 Binary Coded Decimal (BCD) 84
2–11 Digital Codes 87
2–12 Error Detection and Correction Codes 95

3 Logic Gates 112

3–1 The Inverter 114
3–2 The AND Gate 117
3–3 The OR Gate 124
3–4 The NAND Gate 129
3–5 The NOR Gate 134
3–6 The Exclusive-OR and Exclusive-NOR Gates 139
3–7 Programmable Logic 143
3–8 Fixed-Function Logic 150
3–9 Troubleshooting 160

4 Boolean Algebra and Logic Simplification 182

4–1 Boolean Operations and Expressions 184
4–2 Laws and Rules of Boolean Algebra 185
4–3 DeMorgan's Theorem 191
4–4 Boolean Analysis of Logic Circuits 194
4–5 Simplification Using Boolean Algebra 196
4–6 Standard Forms of Boolean Expressions 200
4–7 Boolean Expressions and Truth Tables 206
4–8 The Karnaugh Map 210
4–9 Karnaugh Map SOP Minimization 212
4–10 Karnaugh Map POS Minimization 221
4–11 Five-Variable Karnaugh Maps 225
4–12 VHDL 228
Digital System Application 230

Contents

5 Combinational Logic Analysis 244

5–1 Basic Combinational Logic Circuits 246
5–2 Implementing Combinational Logic 250
5–3 The Universal Property of NAND and NOR Gates 256
5–4 Combinational Logic Using NAND and NOR Gates 258
5–5 Logic Circuit Operation with Pulse Waveform Inputs 263
5–6 Combinational Logic with VHDL 266
5–7 Troubleshooting 272
Digital System Application 278

6 Functions of Combinational Logic 296

6–1 Basic Adders 298
6–2 Parallel Binary Adders 301
6–3 Ripple Carry versus Look-Ahead Carry Adders 308
6–4 Comparators 311
6–5 Decoders 316
6–6 Encoders 324
6–7 Code Converters 329
6–8 Multiplexers (Data Selectors) 331
6–9 Demultiplexers 340
6–10 Parity Generators/Checkers 342
6–11 Troubleshooting 345
Digital System Application 348

7 Latches, Flip-Flops, and Timers 370

7–1 Latches 372
7–2 Edge-Triggered Flip-Flops 378
7–3 Flip-Flop Operating Characteristics 390
7–4 Flip-Flop Applications 393
7–5 One-Shots 398
7–6 The 555 Timer 403
7–7 Troubleshooting 409
Digital System Application 411

8 Counters 426

8–1 Asynchronous Counter Operation 428
8–2 Synchronous Counter Operation 436
8–3 Up/Down Synchronous Counters 444
8–4 Design of Synchronous Counters 447
8–5 Cascaded Counters 457
8–6 Counter Decoding 461
8–7 Counter Applications 464
8–8 Logic Symbols with Dependency Notation 469
8–9 Troubleshooting 471
Digital System Application 475

Contents

9 Shift Registers 492

9–1 Basic Shift Register Functions 494
9–2 Serial In/Serial Out Shift Registers 495
9–3 Serial In/Parallel Out Shift Registers 499
9–4 Parallel In/Serial Out Shift Registers 501
9–5 Parallel In/Parallel Out Shift Registers 505
9–6 Bidirectional Shift Registers 507
9–7 Shift Register Counters 510
9–8 Shift Register Applications 514
9–9 Logic Symbols with Dependency Notation 521
9–10 Troubleshooting 522
Digital System Application 525

10 Memory and Storage 536

10–1 Basics of Semiconductor Memory 538
10–2 Random-Access Memories (RAMs) 542
10–3 Read-Only Memories (ROMs) 555
10–4 Programmable ROMs (PROMs and EPROMs) 560
10–4 Flash Memories 563
10–6 Memory Expansion 568
10–7 Special Types of Memories 574
10–8 Magnetic and Optical Storage 579
10–9 Troubleshooting 585
Digital System Application 589

1 DIGITAL CONCEPTS

CHAPTER OUTLINE

1–1 Digital and Analog Quantities

1–2 Binary Digits, Logic Levels, and Digital Waveforms

1–3 Basic Logic Operations

1–4 Overview of Basic Logic Functions

1–5 Fixed-Function Integrated Circuits

1–6 Introduction to Programmable Logic

1–7 Test and Measurement Instruments

Digital System Application

CHAPTER 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 basic logic operations of NOT, AND, and OR
- Describe the logic functions of the comparator, adder, code converter, encoder, decoder, multiplexer, demultiplexer, counter, and register
- Identify fixed-function digital integrated circuits according to their complexity and the type of circuit packaging
- Identify pin numbers on integrated circuit packages
- Describe programmable logic, discuss the various types, and describe how PLDs are programmed
- Recognize various instruments and understand how they are used in measurement and troubleshooting digital circuits and systems
- Show how a complete digital system is formed by combining the basic functions in a practical application

INTRODUCTION

The term *digital* is derived from the way computers perform operations, 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 areas 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 discrete transistors to complex integrated circuits, some of which contain millions of transistors.

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

KEY TERMS

Key terms are in order of appearance in the chapter.

- Analog
- Digital
- Binary
- Bit
- Pulse
- Clock
- Timing diagram
- Data
- Serial
- Parallel
- Logic
- Input
- Output
- Gate
- NOT
- Inverter
- AND
- OR
- Integrated circuit (IC)
- SPLD
- CPLD
- FPGA
- Compiler
- Troubleshooting

1–1 DIGITAL AND ANALOG QUANTITIES

Electronic circuits 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 quantities
- State the advantages of digital over analog
- Give examples of how digital and analog quantities are used in electronics

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.

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.

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.

An Analog Electronic System

A public address system, used to amplify sound so that it can be heard by a large audience, is one simple example of an application of analog electronics. The basic diagram in Figure 1–3 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 1–3**

A basic audio public address system.

A System Using Digital and Analog Methods

The compact disk (CD) player is an example of a system in which both digital and analog circuits are used. The simplified block diagram in Figure 1–4 illustrates the basic principle. Music in digital form is stored on the compact disk. A laser diode optical system picks up the digital data from the rotating disk and transfers it to the **digital-to-analog converter (DAC).**

FIGURE 1–4

Basic block diagram of a CD player. Only one channel is shown.

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 REVIEW

Answers are at the end of the chapter.

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.

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

Digital electronics involves circuits and systems 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, bits and bumps on a CD or DVD, etc. 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.

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.

COMPUTER NOTE

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.

Figure 1–5 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 unallowed range can appear as either a HIGH or a LOW to a given circuit and is therefore not an acceptable value. For example, the HIGH values for a certain type of digital circuit called CMOS may range from 2 V to 3.3 V and the LOW values may range from 0 V to 0.8 V. So, for example, 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 1–5

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 1–6(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 1–6(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.

FIGURE 1–6

Ideal pulses.

The Pulse As indicated in Figure 1–6, 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 in Figure 1–6 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 1–7 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

FIGURE 1–7
Nonideal pulse characteristics.

capacitive effects. The droop can be caused by stray capacitive 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 1–7. 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_W) 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 1–7.

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 1–8.

FIGURE 1–8

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:

$$f = \frac{1}{T} \qquad \text{Equation 1–1}$$

$$T = \frac{1}{f} \qquad \text{Equation 1–2}$$

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

$$\text{Duty cycle} = \left(\frac{t_W}{T}\right)100\% \qquad \text{Equation 1–3}$$

EXAMPLE 1–1

A portion of a periodic digital waveform is shown in Figure 1–9. The measurements are in milliseconds. Determine the following:

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

FIGURE 1–9

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) $f = \dfrac{1}{T} = \dfrac{1}{10\text{ ms}} = \mathbf{100\ Hz}$

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

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.

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 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 1–10. 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.

COMPUTER NOTE

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.

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 1–11 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 and they all change back LOW at the end of bit time 7 (shaded area).

FIGURE 1–11

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 1–12, binary data are transferred in two ways: serial and parallel.

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 1–12(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.

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 1–12(b) for the example of eight bits being transferred from a computer to a printer. 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 takes longer to transfer a given number of bits than with parallel transfer. 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.

(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 1–12

Illustration of serial and parallel transfer of binary data. Only the data lines are shown.

EXAMPLE 1–2

(a) Determine the total time required to serially transfer the eight bits contained in waveform *A* of Figure 1–13, and indicate the sequence of bits. The left-most bit is the first to be transferred. The 100 kHz clock is used as reference.

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

▲ FIGURE 1–13

Solution

(a) Since the frequency of the clock is 100 kHz, the period is

$$T = \frac{1}{f} = \frac{1}{100\text{ kHz}} = 10\ \mu\text{s}$$

It takes 10 μs to transfer each bit in the waveform. The total transfer time for 8 bits is

$$8 \times 10\ \mu\text{s} = \mathbf{80\ \mu s}$$

To determine the sequence of bits, examine the waveform in Figure 1–13 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 1–14. The left-most bit is the first to be transferred.

▲ **FIGURE 1–14**

(b) A parallel transfer would take **10** μs for all eight bits.

Related Problem

If binary data are transferred at the rate of 10 million bits per second (10 Mbits/s), how long will it take to parallel transfer 16 bits on 16 lines? How long will it take to serially transfer 16 bits?

SECTION 1–2 REVIEW

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 BASIC 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 4.

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 1–15. Other standard symbols for 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 1–15

e basic logic operations and nbols.

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 1–16. When the input is HIGH (1), the output is LOW (0). When the input is LOW, 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.**

FIGURE 1–16

The NOT operation.

AND

The **AND** operation produces a HIGH output only when all the inputs are HIGH, as indicated in Figure 1–17 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 1–17

The AND operation.

OR

The **OR** operation produces a HIGH output when one or more inputs are HIGH, as indicated in Figure 1–18 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 1–18

The OR operation.

SECTION 1-3 REVIEW

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 OVERVIEW OF BASIC LOGIC FUNCTIONS

The three basic logic elements AND, OR, and NOT can be combined to form more complex logic circuits that perform many useful operations and that are used to build complete digital systems. Some of the common logic functions are comparison, arithmetic, code conversion, encoding, decoding, data selection, storage, and counting. This section provides a general overview of these important functions so that you can begin to see how they form the building blocks of digital systems such as computers. Each of the basic logic functions will be covered in detail in later chapters.

After completing this section, you should be able to

- Identify nine basic types of logic functions
- Describe a basic magnitude comparator
- List the four arithmetic functions
- Describe a basic adder
- Describe a basic encoder
- Describe a basic decoder
- Define multiplexing and demultiplexing
- State how data storage is accomplished
- Describe the function of a basic counter

The Comparison Function

Magnitude comparison is performed by a logic circuit called a **comparator,** covered in Chapter 6. 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 1–19. One number in binary form (represented by logic levels) is applied to input A,

(a) Basic magnitude comparator

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

FIGURE 1–19

The comparison function.

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. (We discuss the binary representation of numbers and symbols 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.

The Arithmetic Functions

Addition Addition is performed by a logic circuit called an **adder,** covered in Chapter 6. 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 1–20(a). Figure 1–20(b) illustrates the addition of 3 and 9. You know that the sum is 12; the adder indicates this result by producing 2 on the sum output and 1 on the carry output. Assume that the carry input in this example is 0.

FIGURE 1–20

The addition function.

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 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.

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.

COMPUTER NOTE

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.

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. Various types of codes are covered in Chapter 2, and code converters are covered in Chapter 6.

The Encoding Function

The encoding function is performed by a logic circuit called an **encoder,** covered in Chapter 6. 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 1–21 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 1–21

An encoder used to encode a calculator keystroke into a binary code for storage or for calculation.

The Decoding Function

The decoding function is performed by a logic circuit called a **decoder,** covered in Chapter 6. The decoder converts coded information, such as a binary number, into a non-coded 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 1–22 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 1–22

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 1–23 illustrates this type of application where digital data from three sources are sent out along a single line to three terminals at another location.

In Figure 1–23, 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 Storage Function

Storage is a function that is required in most digital systems, and its purpose is to retain binary data for a period of time. Some storage devices are used for short-term storage and some are used for long-term storage. A storage device can "memorize" a bit or a group of bits and retain the information as long as necessary. Common types of storage devices are flip-flops, registers, semiconductor memories, magnetic disks, magnetic tape, and optical disks (CDs).

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 7.

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*. Shift registers are covered in Chapter 9.

The two basic types of shift registers are serial and parallel. The bits are stored in a serial shift register one at a time, as illustrated in Figure 1–24. 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.

FIGURE 1–24

Example of the operation of a 4-bit serial shift register. Each block represents one storage "cell" or flip-flop.

The bits are stored in a parallel register simultaneously from parallel lines, as shown in Figure 1–25. For this case, a good analogy is loading passengers on a roller coaster where they enter all of the cars in parallel.

FIGURE 1–25

Example of the operation of a 4-bit parallel shift register.

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. Examples are the so-called floppy disks used in computers and 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.

COMPUTER NOTE

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

The Counting Function

The counting function is important in digital systems. There are many types of digital **counters,** but their basic purpose is to count events 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 1–26 illustrates the basic idea of counter operation. Counters are covered in Chapter 8.

FIGURE 1–26

Illustration of basic counter operation.

SECTION 1-4 REVIEW

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 storage devices.
7. What does a counter do?

1–5 FIXED-FUNCTION INTEGRATED CIRCUITS

All the logic elements and functions that have been discussed are generally available in integrated circuit (IC) form. 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 (LCCC)
- 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 1–27 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 1–27

Cutaway view of one type of fixed-function IC package showing the chip mounted inside, with connections to input and output pins.

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 1–28(a).

(a) Dual in-line package (DIP)

(b) Small-outline IC (SOIC)

FIGURE 1–28

Examples of through-hole and surface-mounted devices. The DIP is larger than the SOIC with the same number of leads. This particular DIP is approximately 0.785 in. long, and the SOIC is approximately 0.385 in. long.

Another type of IC package uses 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 1–28(b).

Three common types of SMT packages are the **SOIC** (small-outline IC), the **PLCC** (plastic leaded chip carrier), and the **LCCC** (leadless ceramic chip carrier). These types of packages are available in various sizes depending on the number of leads (more leads are required for more complex circuits). Examples of each type are shown in Figure 1–29. As you can see, the leads of the SOIC are formed into a "gull-wing" shape. The leads of the PLCC are turned under the package in a J-type shape. Instead of leads, the LCCC has metal contacts molded into its ceramic body. Other variations of SMT packages include **SSOP** (shrink small-outline package), **TSSOP** (thin shrink small-outline package), and **TVSOP** (thin very small-outline package).

FIGURE 1–29

Examples of SMT package configurations.

Pin Numbering

All IC packages have a standard format for numbering the pins (leads). The dual in-line packages (DIPs) and the small-outline IC packages (SOICs) have the numbering arrangement illustrated in Figure 1–30(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 LCCC 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 1–30(b) illustrates this format for a 20-pin PLCC package.

FIGURE 1–30

Pin numbering for two 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). A type of fixed-function digital circuit technology that uses bipolar junction transistors is TTL (transistor-transistor logic). BiCMOS uses a combination of both CMOS and TTL.

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 TTL. LSI, VLSI, and ULSI are generally implemented with CMOS or NMOS because it requires less area on a chip and consumes less power. There is more on these integrated technologies in Chapter 3. In addition, Chapter 14 provides a complete circuit-level coverage.

Handling Precautions for CMOS Because of their particular structure, CMOS devices are very sensitive to static charge and can be damaged by electrostatic discharge (ESD) if not handled properly. The following precautions should be taken when you work with CMOS devices:

- CMOS devices should be shipped and stored in conductive foam.
- All instruments and metal benches used in testing should be connected to earth ground.
- The handler's wrist should be connected to earth ground with a length of wire and high-value series resistor.
- Do not remove a CMOS device (or any device for that matter) from a circuit while the dc power is on.
- Do not connect ac or signal voltages to a CMOS device while the dc power supply is off.

SECTION 1-5 REVIEW

1. What is an integrated circuit?
2. Define the terms DIP, SMT, SOIC, SSI, MSI, LSI, VLSI and ULSI.
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

1–6 INTRODUCTION TO PROGRAMMABLE LOGIC

Programmable logic requires both hardware and software. Programmable logic devices 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

Many types of programmable logic 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 1–31. PLDs are either SPLDs (simple PLDs) or CPLDs (complex PLDs).

FIGURE 1–31

Programmable logic.

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 1–32(a). A **GAL** (generic array logic) is a

FIGURE 1–32

Block diagrams of simple programmable logic devices (SPLDs).

device that is basically a PAL that can be 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 1–32(b). A typical SPLD package is shown in Figure 1–33 and generally has from 24 to 28 pins.

FIGURE 1–33

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 1–34 shows a basic CPLD block diagram with four logic array 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 1–35.

FIGURE 1–35

Typical CPLD packages.

(a) 84-pin PLCC package (b) 128-pin PQFP package

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 1–36. The three basic elements in an FPGA are the logic block, the programmable interconnections, and the input/output (I/O) blocks.

FIGURE 1–36

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 1–37. These types of packages can have over 1000 input and output pins.

FIGURE 1–37

A typical ball-grid array package configuration.

The Programming Process

An SPLD, CPLD, or FPGA can be thought of as a "blank slate" on which you implement a specified circuit or 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 1–38.

FIGURE 1–38

Basic configuration for programming a PLD or FPGA.

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 1–39. As indicated, the design flow has access to a design library.

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 1–39. As indicated, the design flow has access to a design library.

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,

FIGURE 1–39

Basic programmable logic design flow block diagram.

AHDL, or ABEL. 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 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.

SECTION 1-6 REVIEW

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.

1–7 TEST AND MEASUREMENT 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 dc power supply, function generator, and digital multimeter (DMM)

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 signals on the screen at one time, enabling their time relationship to be observed. A typical oscilloscope is shown in Figure 1–40.

FIGURE 1–40

A typical dual-channel oscilloscope. Used with permission from Tektronix, Inc.

Two basic types of oscilloscopes, analog and digital, can be used to view digital waveforms. As shown in Figure 1–41(a), the 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. As shown in Figure 1–41(b), the 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.

(a) Analog

(b) Digital

FIGURE 1–41

Comparison of analog and digital oscilloscopes.

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 1–42.

FIGURE 1–42
Block diagram of an analog oscilloscope.

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.

FIGURE 1–43

Block diagram of a digital oscilloscope.

The data then goes to the reconstruction and display circuits for display in its original analog form. Figure 1–43 shows a basic block diagram for a digital oscilloscope.

Oscilloscope Controls A front panel view of a typical dual-channel oscilloscope is shown in Figure 1–44. Instruments vary depending on model and manufacturer, but most have certain common features. For example, the two vertical sections contain a Position control, a channel menu button, and a V/div control. The horizontal section contains a sec/div control.

FIGURE 1–44

A typical dual-channel oscilloscope. Numbers below screen indicate the values for each division on the vertical (voltage) and horizontal (time) scales and can be varied using the vertical and horizontal controls on the scope.

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 1–44, there are identical controls for each of the two channels (CH1 and CH2). The Position control lets you move a displayed waveform up or down vertically on the screen. The Menu button provides 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 V/div, probe attenuation, and other parameters. The V/div control adjusts the number of volts represented by each vertical division on the screen. The V/div setting for each channel is displayed on the bottom of the screen. The Math Menu button provides a selection of operations that can be performed on the input waveforms, such as subtraction, addition, or inversion.

Horizontal Controls In the horizontal section, the controls apply to both channels. The Position control lets you move a displayed waveform left or right horizontally on the screen. The Menu button provides for the selection of several items that appear on the screen such as the main time base, expanded view of a portion of a waveform, and other parameters. 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 1–45 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 1–45

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 1–46 illustrates the result of DC and AC coupling using a pulse waveform that has a dc component.

The voltage probe, shown in Figure 1–47, 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

(a) DC coupled waveform

(b) AC coupled waveform

FIGURE 1–47

An oscilloscope voltage probe. Used with permission from Tektronix, Inc.

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 1–48 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.

FIGURE 1–48

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 1–49. Also, calculate the frequency.

FIGURE 1–49

Solution The V/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

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

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 1–50, and a simplified block diagram is in Figure 1–51.

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

FIGURE 1–50

Typical logic analyzer. Used with permission from Tektronix, Inc.

FIGURE 1–51

Simplified block diagram of a logic analyzer.

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.

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 show-

ing 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 1–52 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

(b) Listing display

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-based system. In this mode the op-codes and the mnemonics (English-like names) of instructions are generally displayed as well as their corresponding memory address. Many logic analyzers also include a source code debug mode, which essentially allows you to see what is actually going on in the system under test when a program instruction is executed.

Probes Three basic types of probes are used with logic analyzers. One is a multichannel compression probe that can be attached to points on a circuit board, as shown in Figure 1–53. Another type of multichannel probe, similar to the one shown, plugs into dedicated sockets mounted on a circuit board. A third type is a single-channel clip-on probe.

FIGURE 1–53

A typical multichannel logic analyzer probe. Used with permission from Tektronix, Inc.

Signal Generators

Logic Signal Source These instruments are also known as pulse generators and pattern generators. They are specifically designed to generate digital signals with precise edge

placement and amplitudes and to produce the streams of 1s and 0s needed to test computer buses, microprocessors, and other digital systems.

Arbitrary Waveform Generators and Function Generators The arbitrary waveform generator can be used to generate standard signals like sine waves, triangular waves, and pulses as well as signals with various shapes and characteristics. Waveforms can be defined by mathematical or graphical input. A typical arbitrary waveform generator is shown in Figure 1–54(a).

The function generator provides pulse waveforms as well as sine waves and triangular waves. Most function generators have logic-compatible outputs to provide the proper level and drive for inputs to digital circuits. Typical function generators are shown in Figure 1–54(b).

(a) An arbitrary waveform generator. (b) Examples of function generators.

FIGURE 1–54

Typical signal generators. Used with permission from Tektronix, Inc.

The Logic Probe and Logic Pulser The logic probe is a convenient, inexpensive hand-held tool that provides a means of troubleshooting a digital circuit by sensing various conditions at a point in a circuit, as illustrated in Figure 1–55. The probe can detect high-level voltage, low-level voltage, single pulses, repetitive pulses, and opens on a PC

FIGURE 1–55

Illustration of how a logic pulser and a logic probe can be used to apply a pulse to a given point and check for resulting pulse activity at another part of the circuit.

board. The probe lamp indicates the condition that exists at a certain point, as indicated in the figure.

The logic pulser produces a repetitive pulse waveform that can be applied to any point in a circuit. You can apply pulses at one point in a circuit with the pulser and check another point for resulting pulses with a logic probe.

Other Instruments

The DC Power Supply This instrument is an indispensable instrument on any test bench. The power supply converts ac power from the standard wall outlet into regulated dc voltage. All digital circuits require dc voltage. Many logic circuits require +5 V or +3.3 V to operate. The power supply is used to power circuits during design, development, and troubleshooting when in-system power is not available. Typical test bench dc power supplies are shown in Figure 1–56.

FIGURE 1–56

Typical dc power supplies. Courtesy of B+K Precision.®

The Digital Multimeter (DMM) The DMM is used for measuring dc and ac voltage and resistance. Figure 1–57 shows typical test bench and handheld DMMs.

FIGURE 1–57

Typical DMMs. Courtesy of B+K Precision.®

SECTION 1-7 REVIEW

1. What is the main difference between a digital and an analog oscilloscope?
2. Name two main differences between a logic analyzer and an oscilloscope?
3. What does the V/div control on an oscilloscope do?
4. What does the sec/div control on an oscilloscope do?
5. What is the purpose of a function generator?

FIGURE 1–58

Simplified basic block diagram for a tablet-counting and bottling control system.

SUMMARY

- An analog quantity has continuous values.
- A digital quantity has a discrete set of values.
- A binary digit is called a bit.
- A pulse is characterized by rise time, fall time, pulse width, and amplitude.

- The frequency of a periodic waveform is the reciprocal of the period. The formulas relating frequency and period are

$$f = \frac{1}{T} \text{ and } T = \frac{1}{f}$$

- The duty cycle of a pulse waveform is the ratio of the pulse width to the period, expressed by the following formula as a percentage:

$$\text{Duty cycle} = \left(\frac{t_W}{T}\right)100\%$$

- A timing diagram is an arrangement of two or more waveforms showing their relationship with respect to time.
- Three basic logic operations are NOT, AND, and OR. The standard symbols for these are given in Figure 1–59.

FIGURE 1–59

SUMMARY

- The basic logic functions are comparison, arithmetic, code conversion, decoding, encoding, data selection, storage, and counting.
- The two broad physical categories of IC packages are through-hole mounted and surface mounted.
- The categories of ICs in terms of circuit complexity are SSI (small-scale integration), MSI (medium-scale integration), LSI, VLSI, and ULSI (large-scale, very large-scale, and ultra large-scale integration).
- Two types of SPLDs (simple programmable logic devices) are PAL (programmable array logic) and GAL (generic array logic).
- The CPLD (complex programmable logic device) contains multiple SPLDs with programmable interconnections.
- The FPGA (field programmable gate array) has a different internal structure than the CPLD and is generally used for more complex circuits and systems.
- Common instruments used in testing and troubleshooting digital circuits are the oscilloscope, logic analyzer, waveform generator, function generator, dc power supply, digital multimeter, logic probe, and logic pulser.

KEY TERMS

Analog Being continuous or having continuous values.

AND A basic logic operation in which a true (HIGH) output occurs only when all the input conditions are true (HIGH).

Binary Having two values or states; describes a number system that has a base of two and utilizes 1 and 0 as its digits.

Bit A binary digit, which can be either a 1 or a 0.

Clock The basic timing signal in a digital system; a periodic waveform in which each interval between pulses equals the time for one bit.

Compiler 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.

CPLD A complex programmable logic device that consists basically of multiple SPLD arrays with programmable interconnections.

Data Information in numeric, alphabetic, or other form.

Digital Related to digits or discrete quantities; having a set of discrete values.

FPGA Field programmable gate array.

Gate A logic circuit that performs a specified logic operation such as AND or OR.

Input The signal or line going into a circuit.

KEY TERMS

Integrated circuit (IC) A type of circuit in which all of the components are integrated on a single chip of semiconductive material of extremely small size.

Inverter A NOT circuit; a circuit that changes a HIGH to a LOW or vice versa.

Logic In digital electronics, the decision-making capability of gate circuits, in which a HIGH represents a true statement and a LOW represents a false one.

NOT A basic logic operation that performs inversions.

OR A basic logic operation in which a true (HIGH) output occurs when one or more of the input conditions are true (HIGH).

Output The signal or line coming out of a circuit.

Parallel In digital systems, data occurring simultaneously on several lines; the transfer or processing of several bits simultaneously.

Pulse A sudden change from one level to another, followed after a time, called the pulse width, by a sudden change back to the original level.

Serial Having one element following another, as in a serial transfer of bits; occurring in sequence rather than simultaneously.

SPLD Simple programmable logic device.

Timing diagram A graph of digital waveforms showing the time relationship of two or more waveforms.

Troubleshooting The technique or process of systematically identifying, isolating, and correcting a fault in a circuit or system.

Digital Fundamentals

Tenth Edition

Floyd

Chapter 2

© 2008 Pearson Education

Summary

Decimal Numbers

The position of each digit in a weighted number system is assigned a weight based on the **base** or **radix** of the system. The radix of decimal numbers is ten, because only ten symbols (0 through 9) are used to represent any number.

The column weights of decimal numbers are powers of ten that increase from right to left beginning with $10^0 = 1$:

$$\ldots 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0.$$

For fractional decimal numbers, the column weights are negative powers of ten that decrease from left to right:

$$10^2\ 10^1\ 10^0.\ 10^{-1}\ 10^{-2}\ 10^{-3}\ 10^{-4}\ \ldots$$

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Decimal Numbers

Decimal numbers can be expressed as the sum of the products of each digit times the column value for that digit. Thus, the number 9240 can be expressed as

$$(9 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (0 \times 10^0)$$

or

$$9 \times 1{,}000 + 2 \times 100 + 4 \times 10 + 0 \times 1$$

Example

Express the number 480.52 as the sum of values of each digit.

Solution

$$480.52 = (4 \times 10^2) + (8 \times 10^1) + (0 \times 10^0) + (5 \times 10^{-1}) + (2 \times 10^{-2})$$

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Numbers

For digital systems, the binary number system is used. Binary has a radix of two and uses the digits 0 and 1 to represent quantities.

The column weights of binary numbers are powers of two that increase from right to left beginning with $2^0 = 1$:

$$\ldots 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0.$$

For fractional binary numbers, the column weights are negative powers of two that decrease from left to right:

$$2^2\ 2^1\ 2^0.\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ \ldots$$

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Numbers

A binary counting sequence for numbers from zero to fifteen is shown.

Notice the pattern of zeros and ones in each column.

Digital counters frequently have this same pattern of digits:

Decimal Number	Binary Number
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Conversions

The decimal equivalent of a binary number can be determined by adding the column values of all of the bits that are 1 and discarding all of the bits that are 0.

Example Convert the binary number 100101.01 to decimal.

Solution Start by writing the column weights; then add the weights that correspond to each 1 in the number.

2^5	2^4	2^3	2^2	2^1	2^0.	2^{-1}	2^{-2}	
32	16	8	4	2	1 .	½	¼	
1	0	0	1	0	1.	0	1	
32			+4		+1		+¼ =	37¼

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Conversions

You can convert a decimal whole number to binary by reversing the procedure. Write the decimal weight of each column and place 1's in the columns that sum to the decimal number.

Example Convert the decimal number 49 to binary.

Solution The column weights double in each position to the right. Write down column weights until the last number is larger than the one you want to convert.

2^6	2^5	2^4	2^3	2^2	2^1	2^0.
64	32	16	8	4	2	1.
0	1	1	0	0	0	1.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Conversions

You can convert a decimal fraction to binary by repeatedly multiplying the fractional results of successive multiplications by 2. The carries form the binary number.

Example

Convert the decimal fraction 0.188 to binary by repeatedly multiplying the fractional results by 2.

Solution

0.188 x 2 = 0.376	carry = 0	MSB
0.376 x 2 = 0.752	carry = 0	
0.752 x 2 = 1.504	carry = 1	
0.504 x 2 = 1.008	carry = 1	
0.008 x 2 = 0.016	carry = 0	

Answer = .00110 (for five significant digits)

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Conversions

You can convert decimal to any other base by repeatedly dividing by the base. For binary, repeatedly divide by 2:

Example Convert the decimal number 49 to binary by repeatedly dividing by 2.

Solution You can do this by "reverse division" and the answer will read from left to right. Put quotients to the left and remainders on top.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Addition

The rules for binary addition are

0 + 0 = 0	Sum = 0, carry = 0
0 + 1 = 0	Sum = 1, carry = 0
1 + 0 = 0	Sum = 1, carry = 0
1 + 1 = 10	Sum = 0, carry = 1

When an input carry = 1 due to a previous result, the rules are

1 + 0 + 0 = 01	Sum = 1, carry = 0
1 + 0 + 1 = 10	Sum = 0, carry = 1
1 + 1 + 0 = 10	Sum = 0, carry = 1
1 + 1 + 1 = 10	Sum = 1, carry = 1

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Addition

Example Add the binary numbers 00111 and 10101 and show the equivalent decimal addition.

Solution

$$\begin{array}{rcr} {\scriptstyle 0\,1\,1\,1}\,00111 & & 7 \\ 10101 & & 21 \\ \hline 11100 & = & 28 \end{array}$$

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Binary Subtraction

The rules for binary subtraction are

$$0 - 0 = 0$$
$$1 - 1 = 0$$
$$1 - 0 = 1$$
$$10 - 1 = 1 \quad \text{with a borrow of 1}$$

Example Subtract the binary number 00111 from 10101 and show the equivalent decimal subtraction.

Solution

$$\begin{array}{rcr} {\scriptstyle 1\,1\,1} & & \\ 1\cancel{0}\cancel{1}01 & & 21 \\ \underline{00111} & & \underline{7} \\ 01110 & = & 14 \end{array}$$

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

1's Complement

The 1's complement of a binary number is just the inverse of the digits. To form the 1's complement, change all 0's to 1's and all 1's to 0's.

For example, the 1's complement of 11001010 is
00110101

In digital circuits, the 1's complement is formed by using inverters:

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

2's Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

Recall that the 1's complement of 11001010 is

00110101 (1's complement)
+1
00110110 (2's complement)

To form the 2's complement, add 1:

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Signed Binary Numbers

There are several ways to represent signed binary numbers. In all cases, the MSB in a signed number is the sign bit, that tells you if the number is positive or negative.

Computers use a modified 2's complement for signed numbers. Positive numbers are stored in *true* form (with a 0 for the sign bit) and negative numbers are stored in *complement* form (with a 1 for the sign bit).

For example, the positive number 58 is written using 8-bits as 00111010 (true form).

Sign bit

Magnitude bits

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Signed Binary Numbers

Negative numbers are written as the 2's complement of the corresponding positive number.

The negative number –58 is written as:

–58 = 11000110 (complement form)

Sign bit Magnitude bits

An easy way to read a signed number that uses this notation is to assign the sign bit a column weight of –128 (for an 8-bit number). Then add the column weights for the 1's.

Example

Assuming that the sign bit = –128, show that 11000110 = –58 as a 2's complement signed number:

Solution

Column weights: –128 64 32 16 8 4 2 1.

1 1 0 0 0 1 1 0

–128 +64 +4 +2 = –58

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Floating Point Numbers

Floating point notation is capable of representing very large or small numbers by using a form of scientific notation. A 32-bit single precision number is illustrated.

S	E (8 bits)	F (23 bits)
Sign bit	Biased exponent (+127)	Magnitude with MSB dropped

Example Express the speed of light, c, in single precision floating point notation. ($c = 0.2998 \times 10^9$)

Solution In binary, $c = 0001\ 0001\ 1101\ 1110\ 1001\ 0101\ 1100\ 0000_2$.

In scientific notation, $c = 1.001\ 1101\ 1110\ 1001\ 0101\ 1100\ 0000 \times 2^{28}$.

S = 0 because the number is positive. $E = 28 + 127 = 155_{10} = 1001\ 1011_2$.

F is the next 23 bits after the first 1 is dropped.

In floating point notation, $c =$

0	10011011	001 1101 1110 1001 0101 1100

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Arithmetic Operations with Signed Numbers

Using the signed number notation with negative numbers in 2's complement form simplifies addition and subtraction of signed numbers.

Rules for **addition**: Add the two signed numbers. Discard any final carries. The result is in signed form.

Examples:

00011110 = +30	00001110 = +14	11111111 = –1
00001111 = +15	11101111 = –17	11111000 = –8
00101101 = +45	11111101 = –3	~~1~~11110111 = –9

Discard carry

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Arithmetic Operations with Signed Numbers

Note that if the number of bits required for the answer is exceeded, overflow will occur. This occurs only if both numbers have the same sign. The overflow will be indicated by an incorrect sign bit.

Two examples are:

01000000 = +128
01000001 = +129
10000001 = ~~–126~~

10000001 = –127
10000001 = –127
~~100000010~~ = ~~+2~~

Discard carry

Wrong! The answer is incorrect and the sign bit has changed.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Arithmetic Operations with Signed Numbers

Rules for **subtraction**: 2's complement the subtrahend and add the numbers. Discard any final carries. The result is in signed form.

Repeat the examples done previously, but subtract:

00011110	(+30)	00001110	(+14)	11111111	(–1)
– 00001111	–(+15)	– 11101111	–(–17)	– 11111000	–(–8)

2's complement subtrahend and add:

00011110 = +30	00001110 = +14	11111111 = –1
11110001 = –15	00010001 = +17	00001000 = +8
~~1~~00001111 = +15	00011111 = +31	~~1~~00000111 = +7

Discard carry

Discard carry

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Hexadecimal Numbers

Hexadecimal uses sixteen characters to represent numbers: the numbers 0 through 9 and the alphabetic characters A through F.

Large binary number can easily be converted to hexadecimal by grouping bits 4 at a time and writing the equivalent hexadecimal character.

Example Express 1001 0110 0000 1110_2 in hexadecimal:

Solution Group the binary number by 4-bits starting from the right. Thus, 960E

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Hexadecimal Numbers

Hexadecimal is a weighted number system. The column weights are powers of 16, which increase from right to left.

Column weights $\left\{ \begin{array}{cccc} 16^3 & 16^2 & 16^1 & 16^0 \\ 4096 & 256 & 16 & 1 \end{array} \right.$

Example Express $1A2F_{16}$ in decimal.

Solution Start by writing the column weights:

4096	256	16	1
1	A	2	F_{16}

$1(4096) + 10(256) + 2(16) + 15(1) = 6703_{10}$

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Octal Numbers

Octal uses eight characters the numbers 0 through 7 to represent numbers. There is no 8 or 9 character in octal.

Binary number can easily be converted to octal by grouping bits 3 at a time and writing the equivalent octal character for each group.

Example Express $1\ 001\ 011\ 000\ 001\ 110_2$ in octal:

Solution Group the binary number by 3-bits starting from the right. Thus, 113016_8

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Octal Numbers

Octal is also a weighted number system. The column weights are powers of 8, which increase from right to left.

Column weights $\left\{ \begin{array}{cccc} 8^3 & 8^2 & 8^1 & 8^0 \\ 512 & 64 & 8 & 1 \end{array} \right.$

Example Express 3702_8 in decimal.

Solution Start by writing the column weights:

512	64	8	1
3	7	0	2_8

$$3(512) + 7(64) + 0(8) + 2(1) = 1986_{10}$$

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

BCD

Binary coded decimal (BCD) is a weighted code that is commonly used in digital systems when it is necessary to show decimal numbers such as in clock displays.

The table illustrates the difference between straight binary and BCD. BCD represents each decimal digit with a 4-bit code. Notice that the codes 1010 through 1111 are not used in BCD.

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

BCD

You can think of BCD in terms of column weights in groups of four bits. For an 8-bit BCD number, the column weights are: 80 40 20 10 8 4 2 1.

Question: What are the column weights for the BCD number 1000 0011 0101 1001?

Answer:

8000 4000 2000 1000 800 400 200 100 80 40 20 10 8 4 2 1

Note that you could add the column weights where there is a 1 to obtain the decimal number. For this case:

$$8000 + 200 + 100 + 40 + 10 + 8 + 1 = 8359_{10}$$

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

BCD

Invalid Codes 1010, 1011, 1100, 1101, 1110, and 1111

Decimal to binary and Binary to decimal:

simply replace each decimal digit with the appropriate 4 bit code, and vice versa.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

EXAMPLE 2-33

Convert each of the following decimal numbers to BCD:

(a) 35 (b) 98 (c) 170 (d) 2469

Solution

(a) 3 5

↓ ↓

00110101

(b) 9 8

↓ ↓

10011000

(c) 1 7 0

↓ ↓ ↓

000101110000

(d) 2 4 6 9

↓ ↓ ↓ ↓

0010010001101001

Related Problem Convert the decimal number 9673 to BCD.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

EXAMPLE 2-34

Convert each of the following BCD codes to decimal:

(a) 10000110 (b) 001101010001 (c) 1001010001110000

Solution

(a) $\underbrace{1000}_{\downarrow\ 8}\underbrace{0110}_{\downarrow\ 6}$

(b) $\underbrace{0011}_{\downarrow\ 3}\underbrace{0101}_{\downarrow\ 5}\underbrace{0001}_{\downarrow\ 1}$

(c) $\underbrace{1001}_{\downarrow\ 9}\underbrace{0100}_{\downarrow\ 4}\underbrace{0111}_{\downarrow\ 7}\underbrace{0000}_{\downarrow\ 0}$

Related Problem Convert the BCD code 1000001000100 1110110 to decimal.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

BCD Addition

Step 1. Add the two BCD numbers, using the rules for binary addition in Section 2-4.

Step 2. If a 4-bit sum is equal to or less than 9, it is a valid BCD number.

Step 3. If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result, Add 6 (0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added. simply add the carry to the next 4-bit group.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

EXAMPLE 2-35

Add the following BCD numbers:

(a) 0011 + 0100 (c) 10000110 + 000]0011

(b) 00100011 + 00010101 (d) 010001010000 + 010000010111

Solution

The decimal number additions are shown for compalison.

(a) 0011	3	(b) 0010 0011	23
+ 0100	+4	+ 0001 0101	+ 15
0111	7	0011 1000	38

(c) 1000 0110	86	(d) 0100 0101 0000	450
+ 0001 0011	+13	+ 0100 0001 0111	+417
1001 1001	99	1000 0110 0111	867

Related Problem Convert the BCD code 1000001000100 1110110 to decimal.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

EXAMPLE 2-36

Add the following BCD numbers

(a) 1001 + 0100 (b) 1001 + 1001

(c) 00010110 + 00010101 (d) 01100111 + 01010011

Solution

The decimal number additions are shown for comparison.

(a)

	1001		9
	+ 0100		+4
	1101	Invalid BCD number (>9)	13
	+ 0110	Add 6	
0001	0011	Valid BCD number	
↓	↓		
1	3		

(b)

	1001		9
	+ 1001		+ 9
1	0010	Invalid because of carry	18
	+ 0110	Add 6	
0001	1000	Valid BCD number	
↓	↓		
1	8		

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

(c)

0001	0110		16
+ 0001	0101		+ 15
0010	1011	Right group is invalid (>9), left group is valid.	31
	+ 0110	Add 6 to invalid code. Add carry, 0001, to next group.	
0011	0001	Valid BCD number	
↓	↓		
3	1		

(d)

0110	0111		67
+ 0101	0011		+ 53
1011	1010	Both groups are invalid (>9)	120
+ 0110	+ 0110	Add 6 to both groups	
0001	0010	0000	Valid BCD number
↓	↓	↓	
1	2	0	

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

BCD

A lab experiment in which BCD is converted to decimal is shown.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Gray code

Gray code is an unweighted code that has a single bit change between one code word and the next in a sequence. Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

Decimal	Binary	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Binary-to-Gray Code Conversion Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.
2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

For example, the conversion of the binary number 10110 to Gray code is as follows:

1	0	1	1	0	Binary
↓	↓	↓	↓	↓	
1	1	1	0	1	Gray

The Gray code is 11101.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Gray-to-Binary Conversion To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

For example, the conversion of the Gray code word 11011 to binary is as follows:

1	1	0	1	1	Gray
↓ + ↗	↓ + ↗	↓ + ↗	↓ + ↗	↓	
1	0	0	1	0	Binary

The binary number is 10010.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

EXAMPLE 2-37

(a) Convert the binary number 11000110 to Gray code.

(b) Convert the Gray code 10101111 to binary.

(a) Binary to Gray code:

1	1	0	0	0	1	1	0
↓	↓	↓	↓	↓	↓	↓	↓
1	**0**	**1**	**0**	**0**	**1**	**0**	**1**

(b) Gray code to binary:

1	0	1	0	1	1	1	1
↓	↓	↓	↓	↓	↓	↓	↓
1	**1**	**0**	**0**	**1**	**0**	**1**	**0**

(a) Convert binary 101101 to Gray code. **(b)** Convert Gray code 100111 to binary.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Gray code

A shaft encoder is a typical application. Three IR emitter/detectors are used to encode the position of the shaft. The encoder on the left uses binary and can have three bits change together, creating a potential error. The encoder on the right uses gray code and only 1-bit changes, eliminating potential errors.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

ASCII

ASCII is a code for alphanumeric characters and control characters. In its original form, ASCII encoded 128 characters and symbols using 7-bits. The first 32 characters are control characters, that are based on obsolete teletype requirements, so these characters are generally assigned to other functions in modern usage.

In 1981, IBM introduced extended ASCII, which is an 8-bit code and increased the character set to 256. Other extended sets (such as Unicode) have been introduced to handle characters in languages other than English.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

3-5 ERROR DETECTION AND CORRECTION CODES

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Parity Method

The parity method is a method of error detection for simple transmission errors involving one bit (or an odd number of bits). A parity bit is an "extra" bit attached to a group of bits to force the number of 1's to be either even (even parity) or odd (odd parity).

Example The ASCII character for "a" is 1100001 and for "A" is 1000001. What is the correct bit to append to make both of these have odd parity?

Solution The ASCII "a" has an odd number of bits that are equal to 1; therefore the parity bit is 0. The ASCII "A" has an even number of bits that are equal to 1; therefore the parity bit is 1.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Assign the proper even parity bit to the following code groups:

(a) 1010 (b) 111000 (c) 101101

(d) 1000111001001 (e) 101101011111

Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) 01010 (b) 1111000 (c) 0101101

(d) 0100011100101 (e) 1101101011111

An odd parity system receives the following code groups: 10110, 11010, 110011, 110101110100, and 1100010101010. Determine which groups, if any, are in error.

Since odd parity is required, any group with an even number of 1s is incorrect. The following groups are in error: **110011** and **1100010101010.**

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

The Hamming Error Correction Code

A single parity bit allows for the detection of single-bit errors in a code word.

In order to correct a detected error, more information is required because the position of the bit in error also must be identified.

So more than one parity bit must be included in a group of bits to be able to correct a detected error.

The Hamming code provides for single-error correction.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

P: Number of Parity Bits;
d: number of data bits : d=4

$$2^P \geq d+p+1$$

Let p = 2. Then $2^P = 4$ and $d+p+1=4+2+1=7$

Let p = 3. Then $2^P = 8$and $d+p+1=4+3+1=8$

So three parity bits are required to provide single-error correction for four data bits.

The parity bits are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8, . . .), as indicated;

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Data bits (D_n)							
Parity bits (P_n)							

Assignment of Parity Bit Values:

Group P1: (001,011,101,111) = (1,3,5,7)

Group P2: (010,011,110,111) = (2,3,6,7)

Group P3: (100,101,110, 111) = (4,5,6,7)

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Summary

Cyclic Redundancy Check

The cyclic redundancy check (CRC) is an error detection method that can detect multiple errors in larger blocks of data. At the sending end, a checksum is appended to a block of data. At the receiving end, the check sum is generated and compared to the sent checksum. If the check sums are the same, no error is detected.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Other concepts

Consisting of numerals, letters, and other characters

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Selected Key Terms

Byte	A group of eight bits
Floating-point number	A number representation based on scientific notation in which the number consists of an exponent and a mantissa.
Hexadecimal	A number system with a base of 16.
Octal	A number system with a base of 8.
BCD	Binary coded decimal; a digital code in which each of the decimal digits, 0 through 9, is represented by a group of four bits.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Selected Key Terms

Alphanumeric	Consisting of numerals, letters, and other characters
ASCII	American Standard Code for Information Interchange; the most widely used alphanumeric code.
Parity	In relation to binary codes, the condition of evenness or oddness in the number of 1s in a code group.
Cyclic redundancy check (CRC)	A type of error detection code.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Quiz

1. For the binary number 1000, the weight of the column with the 1 is

a. 4

b. 6

c. 8

d. 10

© 2008 Pearson Education

Quiz

2. The 2's complement of 1000 is

a. 0111

b. 1000

c. 1001

d. 1010

© 2008 Pearson Education

Quiz

3. The fractional binary number 0.11 has a decimal value of

a. ¼

b. ½

c. ¾

d. none of the above

© 2008 Pearson Education

Quiz

4. The hexadecimal number 2C has a decimal equivalent value of

a. 14

b. 44

c. 64

d. none of the above

© 2008 Pearson Education

Quiz

5. Assume that a floating point number is represented in binary. If the sign bit is 1, the

a. number is negative

b. number is positive

c. exponent is negative

d. exponent is positive

© 2008 Pearson Education

Quiz

6. When two positive signed numbers are added, the result may be larger that the size of the original numbers, creating overflow. This condition is indicated by

a. a change in the sign bit

b. a carry out of the sign position

c. a zero result

d. smoke

© 2008 Pearson Education

Quiz

7. The number 1010 in BCD is

    a. equal to decimal eight

    b. equal to decimal ten

    c. equal to decimal twelve

    d. invalid

© 2008 Pearson Education

Quiz

8. An example of an unweighted code is

a. binary

b. decimal

c. BCD

d. Gray code

© 2008 Pearson Education

Quiz

9. An example of an alphanumeric code is

a. hexadecimal

b. ASCII

c. BCD

d. CRC

© 2008 Pearson Education

Quiz

10. An example of an error detection method for transmitted data is the

a. parity check

b. CRC

c. both of the above

d. none of the above

© 2008 Pearson Education

Quiz

Answers:

1. c	6. a
2. b	7. d
3. c	8. d
4. b	9. b
5. a	10. c

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

3-5 ZENER DIODES

A

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

Zener Breakdown

The avalanche breakdown occurs

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

EXAMPLE 3-

Find

Solution

Related Problem Calculate

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

SECTION 3-5 REVIEW

1. In.

© 2009 Pearson Education, Upper Saddle River, NJ 07458. All Rights Reserved

3 LOGIC GATES

CHAPTER OUTLINE

3–1 The Inverter

3–2 The AND Gate

3–3 The OR Gate

3–4 The NAND Gate

3–5 The NOR Gate

3–6 The Exclusive-OR and Exclusive-NOR Gates

3–7 Programmable Logic

3–8 Fixed-Function Logic

3–9 Troubleshooting

CHAPTER OBJECTIVES

- Describe the operation of the inverter, the AND gate, and the OR gate
- Describe the operation of the NAND gate and the NOR gate
- Express the operation of NOT, AND, OR, NAND, and NOR gates with Boolean algebra
- Describe the operation of the exclusive-OR and exclusive-NOR gates
- Recognize and use both the distinctive shape logic gate symbols and the rectangular outline logic gate symbols of ANSI/IEEE Standard 91-1984

CHAPTER OBJECTIVES

- Construct timing diagrams showing the proper time relationships of inputs and outputs for the various logic gates
- Discuss the basic concepts of programmable logic
- Make basic comparisons between the major IC technologies—CMOS and TTL
- Explain how the different series within the CMOS and TTL families differ from each other
- Define *propagation delay time, power dissipation, speed-power product,* and *fan-out* in relation to logic gates
- List specific fixed-function integrated circuit devices that contain the various logic gates
- Use each logic gate in simple applications
- Troubleshoot logic gates for opens and shorts by using the oscilloscope

KEY TERMS

- Inverter
- Truth table
- Timing diagram
- Boolean algebra
- Complement
- AND gate
- Enable
- OR gate
- NAND gate
- NOR gate
- Exclusive-OR gate
- Exclusive-NOR gate
- AND array
- Fuse
- Antifuse
- EPROM
- EEPROM
- SRAM
- Target device
- JTAG
- CMOS
- TTL
- Propagation delay time
- Fan-out
- Unit load

INTRODUCTION

The emphasis in this chapter is on the operation, application, and troubleshooting of logic gates. The relationship of input and output waveforms of a gate using timing diagrams is thoroughly covered.

Logic symbols used to represent the logic gates are in accordance with ANSI/IEEE Standard 91-1984. This standard has been adopted by private industry and the military for use in internal documentation as well as published literature.

Both programmable logic and fixed-function logic are discussed in this chapter. Because integrated circuits (ICs) are used in all applications, the logic function of a device is generally of greater importance to the technician or technologist than the details of the component-level circuit operation within the IC package. Therefore, detailed coverage of the devices at the component level can be treated as an optional topic. For those who need it and have the time, a thorough coverage of digital integrated circuit technologies is available in Chapter 14, portions of which may be referenced at appropriate points throughout the text. *Suggestion:* Review Section 1–3 before you start this chapter.

3–1 THE INVERTER

The inverter (NOT circuit) performs the operation called *inversion* or *complementation*. The inverter changes one logic level to the opposite level. In terms of bits, it changes a 1 to a 0 and a 0 to a 1.

FIGURE 3–1

Standard logic symbols for the inverter (ANSI/IEEE Std. 91-1984).

(a) Distinctive shape symbols with negation indicators

(b) Rectangular outline symbols with polarity indicators

The Negation and Polarity Indicators

The negation indicator is a "bubble" (○) that indicates **inversion** or *complementation* when it appears on the input or output of any logic element, as shown in Figure 3–1(a) for the inverter. Generally, inputs are on the left of a logic symbol and the output is on the right. When appearing on the input, the bubble means that a 0 is the active or *asserted* input state, and the input is called an active-LOW input. When appearing on the output, the bubble means that a 0 is the active or asserted output state, and the output is called an active-LOW output. The absence of a bubble on the input or output means that a 1 is the active or asserted state, and in this case, the input or output is called active-HIGH.

The polarity or level indicator is a "triangle" (◺) that indicates inversion when it appears on the input or output of a logic element, as shown in Figure 3–1(b). When appearing on the input, it means that a LOW level is the active or asserted input state. When appearing on the output, it means that a LOW level is the active or asserted output state.

Either indicator (bubble or triangle) can be used both on distinctive shape symbols and on rectangular outline symbols. Figure 3–1(a) indicates the principal inverter symbols used in this text. Note that a change in the placement of the negation or polarity indicator does not imply a change in the way an inverter operates.

Inverter Truth Table

When a HIGH level is applied to an inverter input, a LOW level will appear on its output. When a LOW level is applied to its input, a HIGH will appear on its output. This operation is summarized in Table 3–1, which shows the output for each possible input in terms of levels and corresponding bits. A table such as this is called a **truth table.**

Inverter truth table.

INPUT	OUTPUT
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)

Inverter Operation

Figure 3–2 shows the output of an inverter for a pulse input, where t_1 and t_2 indicate the corresponding points on the input and output pulse waveforms.

When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW, thereby producing an inverted output pulse.

FIGURE 3–3

Timing diagram for the case in Figure 3–2.

A timing diagram shows how two or more waveforms relate in time.

EXAMPLE 3–1

A waveform is applied to an inverter in Figure 3–4. Determine the output waveform corresponding to the input and show the timing diagram. According to the placement of the bubble, what is the active output state?

FIGURE 3–4

Logic Expression for an Inverter

In **Boolean algebra,** which is the mathematics of logic circuits and will be covered thoroughly in Chapter 4, a variable is designated by a letter. The **complement** of a variable is designated by a bar over the letter. A variable can take on a value of either 1 or 0. If a given variable is 1, its complement is 0 and vice versa.

The operation of an inverter (NOT circuit) can be expressed as follows: If the input variable is called A and the output variable is called X, then

$$X = \overline{A}$$

This expression states that the output is the complement of the input, so if $A = 0$, then $X = 1$, and if $A = 1$, then $X = 0$. Figure 3–6 illustrates this. The complemented variable $\overline{A}$ can be read as "A bar" or "not A."

FIGURE 3–6

The inverter complements an input variable.

Boolean algebra uses variables and operators to describe a logic circuit.

An Application

FIGURE 3–7

Example of a 1's complement circuit using inverters.

SECTION 3–1 REVIEW

1. When a 1 is on the input of an inverter, what is the output?
2. An active HIGH pulse (HIGH level when asserted, LOW level when not) is required on an inverter input.
   (a) Draw the appropriate logic symbol, using the distinctive shape and the negation indicator, for the inverter in this application.
   (b) Describe the output when a positive-going pulse is applied to the input of an inverter.

3–2 THE AND GATE

The AND gate is one of the basic gates that can be combined to form any logic function. An AND gate can have two or more inputs and performs what is known as logical multiplication.

The term *gate* is used to describe a circuit that performs a basic logic operation. The AND gate is composed of two or more inputs and a single output, as indicated by the standard logic symbols shown in Figure 3–8. Inputs are on the left, and the output is on the right in each symbol. Gates with two inputs are shown; however, an AND gate can have any number of inputs greater than one. Although examples of both distinctive shape symbols and rectangular outline symbols are shown, the distinctive shape symbol, shown in part (a), is used predominantly in this book.

(a) Distinctive shape

(b) Rectangular outline with the AND (&) qualifying symbol

Operation of an AND Gate

An **AND gate** produces a HIGH output *only* when *all* of the inputs are HIGH. When any of the inputs is LOW, the output is LOW.

An AND gate can have more than two inputs.

For an AND gate, all HIGH inputs make a HIGH output.

INPUTS		OUTPUT
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1

1 = HIGH, 0 = LOW

The total number of possible combinations of binary inputs to a gate is determined by the following formula:

$$N = 2^n$$

A
B
C
X
A
B
C
X

Logic Expressions for an AND Gate

Boolean multiplication is the same as the AND function.

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

$$X = AB$$

When variables are shown together like *ABC*, they are ANDed.

A	B	AB = X
0	0	$0 \cdot 0 = 0$
0	1	$0 \cdot 1 = 0$
1	0	$1 \cdot 0 = 0$
1	1	$1 \cdot 1 = 1$

Applications

The AND Gate as an Enable/Inhibit Device A common application of the AND gate is to **enable** (that is, to allow) the passage of a signal (pulse waveform) from one point to another at certain times and to inhibit (prevent) the passage at other times.

FIGURE 3–15

An AND gate performing an enable/inhibit function for a frequency counter.

pulse repeats at certain intervals and a new updated count is made so that if the frequency changes, the new value will be displayed. Between enable pulses, the counter is reset so that it starts at zero each time an enable pulse occurs. The current frequency count is stored in a register so that the display is unaffected by the resetting of the counter.

A Seat Belt Alarm System

SECTION 3–2 REVIEW

1. When is the output of an AND gate HIGH?
2. When is the output of an AND gate LOW?
3. Describe the truth table for a 5-input AND gate.

3–3 THE OR GATE

The OR gate is another of the basic gates from which all logic functions are constructed. An OR gate can have two or more inputs and performs what is known as logical addition.

FIGURE 3–17

Standard logic symbols for the OR gate showing two inputs (ANSI/IEEE Std. 91-1984).

(a) Distinctive shape

(b) Rectangular outline with the OR (≥ 1) qualifying symbol

An OR gate can have more than two inputs.

Operation of an OR Gate

An OR gate produces a HIGH on the output when *any* of the inputs is HIGH. The output is LOW only when all of the inputs are LOW. Therefore, an OR gate determines when one or more of its inputs are HIGH and produces a HIGH on its output to indicate this condition.

For an OR gate, at least one HIGH input makes a HIGH output.

OR Gate Truth Table

INPUTS A	INPUTS B	OUTPUT X
0	0	0
0	1	1
1	0	1
1	1	1

1 = HIGH, 0 = LOW

This truth table can be expanded for any number of inputs; but regardless of the number of inputs, the output is HIGH when one or more of the inputs are HIGH.

Operation with Waveform Inputs

Input A
Input B
Output X
A
B
X
When either input or both inputs are HIGH,
the output is HIGH.

Inputs
A
B
Output X
A
B
X

For the 3-input OR gate in Figure 3–22, determine the output waveform in proper time relation to the inputs.

Logic Expressions for an OR Gate

The logical OR function of two variables is represented mathematically by a + between the two variables, for example, $A + B$.

Addition in Boolean algebra involves variables whose values are either binary 1 or binary 0. The basic rules for **Boolean addition** are as follows:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

When variables are separated by +, they are ORed.

Boolean addition is the same as the OR function.

Notice that Boolean addition differs from binary addition in the case where two 1s are added. There is no carry in Boolean addition.

A	B	A + B = X
0	0	0 + 0 = 0
0	1	0 + 1 = 1
1	0	1 + 0 = 1
1	1	1 + 1 = 1

An Application

SECTION 3–3 REVIEW

1. When is the output of an OR gate HIGH?
2. When is the output of an OR gate LOW?
3. Describe the truth table for a 3-input OR gate.

3–4 THE NAND GATE

The NAND gate is a popular logic element because it can be used as a universal gate; that is, NAND gates can be used in combination to perform the AND, OR, and inverter operations. The universal property of the NAND gate will be examined thoroughly in Chapter 5.

After completing this section, you should be able to

- Identify a NAND gate by its distinctive shape symbol or by its rectangular outline symbol
- Describe the operation of a NAND gate
- Develop the truth table for a NAND gate with any number of inputs
- Produce a timing diagram for a NAND gate with any specified input waveforms
- Write the logic expression for a NAND gate with any number of inputs
- Describe NAND gate operation in terms of its negative-OR equivalent
- Discuss examples of NAND gate applications

The term *NAND* is a contraction of NOT-AND and implies an AND function with a complemented (inverted) output. The standard logic symbol for a 2-input NAND gate and its equivalency to an AND gate followed by an inverter are shown in Figure 3–25(a), where the symbol ≡ means equivalent to. A rectangular outline symbol is shown in part (b).

The NAND is the same as the AND except the output is inverted.

(a) Distinctive shape, 2-input NAND gate and its NOT/AND equivalent

(b) Rectangular outline, 2-input NAND gate with polarity indicator

Operation of a NAND Gate

A **NAND gate** produces a LOW output only when all the inputs are HIGH. When any of the inputs is LOW, the output will be HIGH. For the specific case of a 2-input NAND gate, as shown in Figure 3–25 with the inputs labeled *A* and *B* and the output labeled *X*, the operation can be stated as follows:

> **For a 2-input NAND gate, output *X* is LOW only when inputs *A* and *B* are HIGH; *X* is HIGH when either *A* or *B* is LOW, or when both *A* and *B* are LOW.**

Note that this operation is opposite that of the AND in terms of the output level. In a NAND gate, the LOW level (0) is the active or asserted output level, as indicated by the bubble on the output. Figure 3–26 illustrates the operation of a 2-input NAND gate for all four input combinations, and Table 3–7 is the truth table summarizing the logical operation of the 2-input NAND gate.

FIGURE 3–26

Operation of a 2-input NAND gate. Open file F03-26 to verify NAND gate operation.

TABLE 3–7

Truth table for a 2-input NAND gate.

INPUTS		OUTPUT
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

1 = HIGH, 0 = LOW.

Operation with Waveform Inputs

Now let's look at the pulse waveform operation of a NAND gate. Remember from the truth table that the only time a LOW output occurs is when all of the inputs are HIGH.

EXAMPLE 3–9

If the two waveforms A and B shown in Figure 3–27 are applied to the NAND gate inputs, determine the resulting output waveform.

▲ **FIGURE 3–27**

Solution Output waveform X is LOW only during the four time intervals when both input waveforms A and B are HIGH as shown in the timing diagram.

Related Problem Determine the output waveform and show the timing diagram if input waveform B is inverted.

EXAMPLE 3–10

Show the output waveform for the 3-input NAND gate in Figure 3–28 with its proper time relationship to the inputs.

▲ FIGURE 3–28

Solution The output waveform X is LOW only when all three input waveforms are HIGH as shown in the timing diagram.

Related Problem Determine the output waveform and show the timing diagram if input waveform A is inverted.

Negative-OR Equivalent Operation of a NAND Gate Inherent in a NAND gate's operation is the fact that one or more LOW inputs produce a HIGH output. Table 3–7 shows that output X is HIGH (1) when any of the inputs, A and B, is LOW (0). From this viewpoint, a NAND gate can be used for an OR operation that requires one or more LOW inputs to produce a HIGH output. This aspect of NAND operation is referred to as **negative-OR.** The term *negative* in this context means that the inputs are defined to be in the active or asserted state when LOW.

For a 2-input NAND gate performing a negative-OR operation, output X is HIGH when either input A or input B is LOW, or when both A and B are LOW.

When a NAND gate is used to detect one or more LOWs on its inputs rather than all HIGHs, it is performing the negative-OR operation and is represented by the standard logic symbol shown in Figure 3–29. Although the two symbols in Figure 3–29 represent the same physical gate, they serve to define its role or mode of operation in a particular application, as illustrated by Examples 3–11 through 3–13.

FIGURE 3–29

Standard symbols representing the two equivalent operations of a NAND gate.

Tank A
Level sensor
+V
HIGH
LOW
Green light indicates both tanks are greater than 1/4 full.
HIGH
Tank B
Level sensor

Tank A
Tank B
HIGH
LOW
HIGH
Red light indicates one or both tanks are less than 1/4 full.

For the 4-input NAND gate in Figure 3–32, operating as a negative-OR, determine the output with respect to the inputs.

Logic Expressions for a NAND Gate

The Boolean expression for the output of a 2-input NAND gate is

$$X = \overline{AB}$$

This expression says that the two input variables, A and B, are first ANDed and then complemented, as indicated by the bar over the AND expression. This is a description in equation form of the operation of a NAND gate with two inputs. Evaluating this expression for all possible values of the two input variables, you get the results shown in Table 3–8.

TABLE 3–8

A	B	$\overline{AB} = X$
0	0	$\overline{0 \cdot 0} = \overline{0} = 1$
0	1	$\overline{0 \cdot 1} = \overline{0} = 1$
1	0	$\overline{1 \cdot 0} = \overline{0} = 1$
1	1	$\overline{1 \cdot 1} = \overline{1} = 0$

SECTION 3–4 REVIEW

1. When is the output of a NAND gate LOW?
2. When is the output of a NAND gate HIGH?
3. Describe the functional differences between a NAND gate and a negative-OR gate. Do they both have the same truth table?
4. Write the output expression for a NAND gate with inputs *A*, *B*, and *C*.

3–5 THE NOR GATE

The NOR gate, like the NAND gate, is a useful logic element because it can also be used as a universal gate; that is, NOR gates can be used in combination to perform the AND, OR, and inverter operations. The universal property of the NOR gate will be examined thoroughly in Chapter 5.

(a) Distinctive shape, 2-input NOR gate and its NOT/OR equivalent

(b) Rectangular outline, 2-input NOR gate with polarity indicator

FIGURE 3–33

Standard NOR gate logic symbols (ANSI/IEEE Std. 91-1984).

For a 2-input NOR gate, output *X* is LOW when either input *A* or input *B* is HIGH, or when both *A* and *B* are HIGH; *X* is HIGH only when both *A* and *B* are LOW.

FIGURE 3–34

Operation of a 2-input NOR gate. Open file F03-34 to verify NOR gate operation.

TABLE 3–9

Truth table for a 2-input NOR gate.

INPUTS		OUTPUT
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0

1 = HIGH, 0 = LOW.

If the two waveforms shown in Figure 3–35 are applied to a NOR gate, what is the resulting output waveform?

Show the output waveform for the 3-input NOR gate in Figure 3–36 with the proper time relation to the inputs.

Negative-AND Equivalent Operation of the NOR Gate

For a 2-input NOR gate performing a negative-AND operation, output *X* is HIGH only when both inputs *A* and *B* are LOW.

FIGURE 3–37

Standard symbols representing the two equivalent operations of a NOR gate.

A device is needed to indicate when two LOW levels occur simultaneously on its inputs and to produce a HIGH output as an indication. Specify the device.

A 2-input NOR gate operating as a negative-AND gate is required to produce a HIGH output when both inputs are LOW, as shown in Figure 3–38.

FIGURE 3–38

EXAMPLE 3–17

As part of an aircraft's functional monitoring system, a circuit is required to indicate the status of the landing gears prior to landing. A green LED display turns on if all three gears are properly extended when the "gear down" switch has been activated in preparation for landing. A red LED display turns on if any of the gears fail to extend properly prior to landing. When a landing gear is extended, its sensor produces a LOW voltage. When a landing gear is retracted, its sensor produces a HIGH voltage. Implement a circuit to meet this requirement.

HANDS ON TIP

When driving a load such as an LED with a logic gate, consult the manufacturer's data sheet for maximum drive capabilities (output current). A regular IC logic gate may not be capable of handling the current required by certain loads such as some LEDs. Logic gates with a buffered output, such as an open-collector (OC) or open-drain (OD) output, are available in many types of IC logic gate configurations. The output current capability of typical IC logic gates is limited to the μA or relatively low mA range. For example, standard TTL can handle output currents up to 16 mA. Most LEDs require currents in the range of about 10 mA to 50 mA.

EXAMPLE 3–18

For the 4-input NOR gate operating as a negative-AND in Figure 3–40, determine the output relative to the inputs.

Logic Expressions for a NOR Gate

The Boolean expression for the output of a 2-input NOR gate can be written as

$$X = \overline{A + B}$$

This equation says that the two input variables are first ORed and then complemented, as indicated by the bar over the OR expression. Evaluating this expression, you get the results shown in Table 3–10. The NOR expression can be extended to more than two input variables by including additional letters to represent the other variables.

TABLE 3–10

A	B	$\overline{A+B}=X$
0	0	$\overline{0+0} = \overline{0} = 1$
0	1	$\overline{0+1} = \overline{1} = 0$
1	0	$\overline{1+0} = \overline{1} = 0$
1	1	$\overline{1+1} = \overline{1} = 0$

SECTION 3–5 REVIEW

1. When is the output of a NOR gate HIGH?
2. When is the output of a NOR gate LOW?
3. Describe the functional difference between a NOR gate and a negative-AND gate. Do they both have the same truth table?
4. Write the output expression for a 3-input NOR with input variables *A*, *B*, and *C*.

3–6 THE EXCLUSIVE-OR AND EXCLUSIVE-NOR GATES

Exclusive-OR and exclusive-NOR gates are formed by a combination of other gates already discussed, as you will see in Chapter 5. However, because of their fundamental importance in many applications, these gates are often treated as basic logic elements with their own unique symbols.

The Exclusive-OR Gate

Standard symbols for an exclusive-OR (XOR for short) gate are shown in Figure 3–41. The XOR gate has only two inputs.

(a) Distinctive shape

(b) Rectangular outline with the XOR

For an exclusive-OR gate, opposite inputs make the output HIGH.

TABLE 3–11

Truth table for an exclusive-OR gate.

INPUTS		OUTPUT
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

EXAMPLE 3–19

A certain system contains two identical circuits operating in parallel. As long as both are operating properly, the outputs of both circuits are always the same. If one of the circuits fails, the outputs will be at opposite levels at some time. Devise a way to detect that a failure has occurred in one of the circuits.

The Exclusive-NOR Gate

Standard symbols for an **exclusive-NOR** (XNOR) **gate** are shown in Figure 3–44. Like the XOR gate, an XNOR has only two inputs. The bubble on the output of the XNOR symbol indicates that its output is opposite that of the XOR gate. When the two input logic levels are opposite, the output of the exclusive-NOR gate is LOW. The operation can be stated as follows (*A* and *B* are inputs, *X* is the output):

For an exclusive-NOR gate, output *X* is LOW when input *A* is LOW and input *B* is HIGH, or when *A* is HIGH and *B* is LOW; *X* is HIGH when *A* and *B* are both HIGH or both LOW.

(a) Distinctive shape

(b) Rectangular outline

TABLE 3–12

Truth table for an exclusive-NOR gate.

INPUTS		OUTPUT
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

EXAMPLE 3–20

Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, *A* and *B*, in Figure 3–47.

An Application

An exclusive-OR gate can be used as a two-bit adder. Recall from Chapter 2 that the basic rules for binary addition are as follows: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10. An examination of the truth table for an XOR gate will show you that its output is the binary sum of the two input bits. In the case where the inputs are both 1s, the output is the sum 0, but you lose the carry of 1. In Chapter 6 you will see how XOR gates are combined to make complete adding circuits. Figure 3–48 illustrates an XOR gate used as a basic adder.

Input bits		Output (sum)
A	*B*	Σ
0	0	0
0	1	1
1	0	1
1	1	0 (without 1 carry)

FIGURE 3–48

An XOR gate used to add two bits.

SECTION 3-6 REVIEW

1. When is the output of an XOR gate HIGH?
2. When is the output of an XNOR gate HIGH?
3. How can you use an XOR gate to detect when two bits are different?

3-7 PROGRAMMABLE LOGIC

Programmable logic was introduced in Chapter 1. In this section, the basic concept of the programmable AND array, which forms the basis for most programmable logic, is discussed, and the major process technologies are covered. A programmable logic device (PLD) is one that does not initially have a fixed-logic function but that can be programmed to implement just about any logic design. As you have learned, two types of PLD are the SPLD and CPLD. In addition to the PLD, the other major category of programmable logic is the FPGA. For simplicity, all of these devices will be referred to as PLDs. Also, some important concepts in programming are discussed.

Basic Concept of the AND Array

Most types of PLDs use some form of **AND array.** Basically, this array consists of AND gates and a matrix of interconnections with programmable links at each cross point, as shown in Figure 3–49(a). The purpose of the programmable links is to either make or break a connection between a row line and a column line in the interconnection matrix. For each input to an AND gate, only one programmable link is left intact in order to connect the desired variable to the gate input. Figure 3–49(b) illustrates an array after it has been programmed.

(a) Unprogrammed

(b) Programmed

EXAMPLE 3–21

Show the AND array in Figure 3–49(a) programmed for the following outputs: $X_1 = A\overline{B}$, $X_2 = \overline{A}B$, and $X_3 = \overline{A}\,\overline{B}$

See Figure 3–50.

Programmable Link Process Technologies

Several different process technologies are used for programmable links in PLDs.

Fuse Technology This was the original programmable link technology. It is still used in some SPLDs. The **fuse** is a metal link that connects a row and a column in the interconnection matrix. Before programming, there is a fused connection at each intersection. To program a device, the selected fuses are opened by passing a current through them sufficient to "blow" the fuse and break the connection. The intact fuses remain and provide a connection between the rows and columns. The fuse link is illustrated in Figure 3–51. Programmable logic devices that use fuse technology are one-time programmable (**OTP**).

(a) Fuse intact before programming (b) Programming current (c) Fuse open after programming

Antifuse Technology An **antifuse** programmable link is the opposite of a fuse link. Instead of breaking the connection, a connection is made during programming. An antifuse starts out as an open circuit whereas the fuse starts out as a short circuit. Before programming, there are no connections between the rows and columns in the interconnection matrix. An antifuse is basically two conductors separated by an insulator. To program a device with antifuse technology, a programmer tool applies a sufficient voltage across selected antifuses to break down the insulation between the two conductive materials, causing the insulator to become a low-resistance link. The antifuse link is illustrated in Figure 3–52. An antifuse device is also a one-time programmable (OTP) device.

(a) Antifuse is open before programming

(b) Programming voltage breaks down insulation layer to create contact.

(c) Antifuse is effectively shorted after programming

EPROM Technology In certain programmable logic devices, the programmable links are similar to the memory cells in **EPROMs** (electrically programmable read-only memories). This type of PLD is programmed using a special tool known as a device programmer. The device is inserted into the programmer, which is connected to a computer running the programming software. Most EPROM-based PLDs are one-time programmable (OTP). However, those with windowed packages can be erased with UV (ultraviolet) light and reprogrammed using a standard PLD programming fixture. EPROM process technology uses a special type of MOS transistor, known as a floating-gate transistor, as the programmable link. The floating-gate device utilizes a process called Fowler-Nordheim tunneling to place electrons in the floating-gate structure.

In a programmable AND array, the floating-gate transistor acts as a switch to connect the row line to either a HIGH or a LOW, depending on the input variable. For input variables that are not used, the transistor is programmed to be permanently *off* (open). Figure 3–53 shows one AND gate in a simple array. Variable A controls the state of the transistor in the first column, and variable B controls the transistor in the third column. When a transistor is *off*, like an open switch, the input line to the AND gate is at $+V$ (HIGH). When a transistor is *on*, like a closed switch, the input line is connected to ground (LOW). When variable A or B is 0 (LOW), the transistor is *on*, keeping the input line to the AND gate LOW. When A or B is 1 (HIGH), the transistor is *off*, keeping the input line to the AND gate HIGH.

SRAM Technology Many FPGAs and some CPLDs use a process technology similar to that used in **SRAMs** (static random-access memories). The basic concept of SRAM-based programmable logic arrays is illustrated in Figure 3–54(a). A SRAM-type memory cell is used to turn a transistor *on* or *off* to connect or disconnect rows and columns. For example, when the memory cell contains a 1 (green), the transistor is *on* and connects the associated row and column lines, as shown in part (b). When the memory cell contains a 0 (blue), the transistor is *off* so there is no connection between the lines, as shown in part (c).

SRAM technology is different from the other process technologies discussed because it is a volatile technology. This means that a SRAM cell does not retain data when power is turned *off*. The programming data must be loaded into a memory; and when power is turned *on*, the data from the memory reprograms the SRAM-based PLD.

The fuse, antifuse, EPROM, and EEPROM process technologies are nonvolatile, so they retain their programming when the power is *off*. A fuse is permanently open, an antifuse is permanently closed, and floating-gate transistors used in EPROM and EEPROM-based arrays can retain their *on* or *off* state indefinitely.

FIGURE 3–54

Basic concept of an AND array with SRAM technology.

Device Programming

The general concept of programming was introduced in Chapter 1, and you have seen how interconnections can be made in a simple array by opening or closing the programmable links. SPLDs, CPLDs, and FPGAs are programmed in essentially the same way. The devices with OTP (one-time programmable) process technologies (fuse, antifuse, or EPROM) must be programmed with a special hardware fixture called a *programmer*. The programmer is connected to a computer by a standard interface cable, as shown in Figure 3–55. Development software is installed on the computer, and the device is inserted into the programmer socket. Most programmers have adapters, such as the one shown, that allow different types of packages to be plugged in.

In-System Programming (ISP)

FIGURE 3–58

Simplified illustration of in-system programming via a JTAG interface.

3–8 FIXED-FUNCTION LOGIC

Two major digital integrated circuit (IC) technologies that are used to implement logic gates are CMOS and TTL. The logic operations of NOT, AND, OR, NAND, NOR, and exclusive-OR are the same regardless of the IC technology used; that is, an AND gate has the same logic function whether it is implemented with CMOS or TTL.

CMOS stands for Complementary Metal-Oxide Semiconductor and is implemented with a type of field-effect transistor. **TTL** stands for Transistor-Transistor Logic and is implemented with bipolar junction transistors. Keep in mind that CMOS and TTL differ only in the type of circuit components and values of parameters and not in the basic logic operation. A CMOS AND gate has the same logic operation as a TTL AND gate. This is true for all the other basic logic functions. The difference in CMOS and TTL is in performance characteristics such as switching speed (propagation delay), power dissipation, noise immunity, and other parameters.

CMOS

There is little disagreement about which circuit technology, CMOS or TTL, is the most widely used. It appears that CMOS has become the dominant technology and may eventually replace TTL in small- and medium-scale ICs. Although TTL dominated for many years mainly because it had faster switching speeds and a greater selection of device types, CMOS always had the advantage of much lower power dissipation although that parameter is frequency dependent. The switching speeds of CMOS have been greatly improved and are now competitive with TTL, while low power dissipation and other desirable factors have been retained as the technology has progressed.

CMOS Series The categories of CMOS in terms of the dc supply voltage are the 5 V CMOS, the 3.3 V CMOS, the 2.5 V CMOS, and the 1.8 V CMOS. The lower-voltage CMOS families are a more recent development and are the result of an effort to reduce the power dissipation. Since power dissipation is proportional to the square of the voltage, a reduction from 5 V to 3.3 V, for example, cuts the power by 34% with other factors remaining the same.

Within each supply voltage category, several series of CMOS logic gates are available. These series within the CMOS family differ in their performance characteristics and are designated by the prefix 74 or 54 followed by a letter or letters that indicate the series and then a number that indicates the type of logic device. The prefix 74 indicates commercial grade for general use, and the prefix 54 indicates military grade for more severe environments. We will refer only to the 74-prefixed devices in this textbook. The basic CMOS series for the 5 V category and their designations include

- 74HC and 74HCT—High-speed CMOS (the "T" indicates TTL compatibility)
- 74AC and 74ACT—Advanced CMOS
- 74AHC and 74AHCT—Advanced High-speed CMOS

The basic CMOS series for the 3.3 V category and their designations include

- 74LV—Low-voltage CMOS
- 74LVC—Low-voltage CMOS
- 74ALVC—Advanced Low-voltage CMOS

In addition to the 74 series there is a 4000 series, which is an older, low-speed CMOS technology that is still available, although in limited use. In addition to the "pure" CMOS, there is a series that combines both CMOS and TTL called BiCMOS. The basic BiCMOS series and their designations are as follows:

- 74BCT—BiCMOS
- 74ABT—Advanced BiCMOS
- 74LVT—Low-voltage BiCMOS
- 74ALB—Advanced Low-voltage BiCMOS

TTL

TTL has been a popular digital IC technology for many years. One advantage of TTL is that it is not sensitive to electrostatic discharge as CMOS is and, therefore, is more practical in most laboratory experimentation and prototyping because you do not have to worry about handling precautions.

TTL Series Like CMOS, several series of TTL logic gates are available, all which operate from a 5 V dc supply. These series within the TTL family differ in their performance characteristics and are designated by the prefix 74 or 54 followed by a letter or letters that indicate the series and a number that indicates the type of logic device within the series. A TTL IC can be distinguished from CMOS by the letters that follow the 74 or 54 prefix.

The basic TTL series and their designations are as follows:

- 74—standard TTL (no letter)
- 74S—Schottky TTL
- 74AS—Advanced Schottky TTL
- 74LS—Low-power Schottky TTL
- 74ALS—Advanced Low-power Schottky TTL
- 74F—Fast TTL

Types of Fixed-Function Logic Gates

All of the basic logic operations, NOT, AND, OR, NAND, NOR, exclusive-OR (XOR), and exclusive-NOR (XNOR) are available in both CMOS and TTL. In addition to these, buffered output gates are also available for driving loads that require high currents. The types of gate configurations typically available in IC packages are identified by the last two or three digits in the series designation. For example, 74LS04 is a low-power **Schottky** hex inverter package. Some of the common logic gate configurations and their standard identifier digits are as follows:

- Quad 2-input NAND—**00**
- Quad 2-input NOR—**02**
- Hex inverter—**04**
- Quad 2-input AND—**08**
- Triple 3-input NAND—**10**
- Triple 3-input AND—**11**
- Dual 4-input NAND—**20**
- Dual 2-input AND—**21**
- Triple 3-input NOR—**27**
- Single 8-input NAND—**30**
- Quad 2-input OR—**32**
- Quad XOR—**86**
- Quad XNOR—**266**

IC Packages All of the 74 series CMOS are pin-compatible with the same types of devices in TTL. This means that a CMOS digital IC such as the 74HC00 (quad 2-input NAND), which contains four 2-input NAND gates in one IC package, has the identical package pin numbers for each input and output as does the corresponding TTL device. Typical IC gate packages, the dual in-line package (DIP) for plug-in or feedthrough mounting and the small-outline integrated circuit (SOIC) package for surface mounting, are shown in Figure 3–60. In some cases, other types of packages are also available. The SOIC package is significantly smaller than the DIP. The pin configuration diagrams for most of the fixed-function logic devices listed above are shown in Figure 3–61.

(a) 14-pin dual in-line package (DIP) for feedthrough mounting

(b) 14-pin small outline package (SOIC) for surface mounting

'00

'02

'04

'08

'10

'11

'20

'21

'27

'30

'32

'86

Single-Gate Logic A limited selection of CMOS gates is available in single-gate packages. With one gate to a package, this series comes in tiny 5-pin packages that are intended for use in last-minute modifications for squeezing logic into tight spots where available space is limited.

Logic Symbols The logic symbols for fixed-function integrated circuits use the standard gate symbols and show the number of gates in the IC package and the associated pin numbers for each gate as well as the pin numbers for V_{CC} and ground. An example is shown in Figure 3–62 for a hex inverter and for a quad 2-input NAND gate. Both the distinctive shape and the rectangular outline formats are shown. Regardless of the logic family, all devices with the same suffix are pin-compatible; in other words, they will have the same arrangement of pin numbers. For example, the 7400, 74S00, 74LS00, 74ALS00, 74F00, 74HC00, and 74AHC00 are all pin-compatible quad 2-input NAND gate packages.

Distinctive shape logic diagram

Rectangular outline logic symbol with polarity indicators. The inverter qualifying symbol (1) appears in the top block and applies to all blocks below.

(a) Hex inverter

(b) Quad 2-input NAND

Performance Characteristics and Parameters

Several things define the performance of a logic circuit. These performance characteristics are the switching speed measured in terms of the propagation delay time, the power dissipation, the fan-out or drive capability, the speed-power product, the dc supply voltage, and the input/output logic levels.

Propagation Delay Time This parameter is a result of the limitation on switching speed or frequency at which a logic circuit can operate. The terms *low speed* and *high speed,* applied to logic circuits, refer to the propagation delay time. The shorter the propagation delay, the higher the speed of the circuit and the higher the frequency at which it can operate.

Propagation delay time, t_P, of a logic gate is the time interval between the application of an input pulse and the occurrence of the resulting output pulse. There are two different

measurements of propagation delay time associated with a logic gate that apply to all the types of basic gates:

- t_{PHL}: The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from the HIGH level to the LOW level (HL).
- t_{PLH}: The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from the LOW level to the HIGH level (LH).

Show the propagation delay times of the inverter in Figure 3–63(a).

For standard-series TTL gates, the typical propagation delay is 11 ns and for F-series gates it is 3.3 ns. For HCT-series CMOS, the propagation delay is 7 ns, for the AC series it is 5 ns, and for the ALVC series it is 3 ns. All specified values are dependent on certain operating conditions as stated on a data sheet.

DC Supply Voltage (V_{CC}) The typical dc supply voltage for CMOS is either 5 V, 3.3 V, 2.5 V, or 1.8 V, depending on the category. An advantage of CMOS is that the supply voltages can vary over a wider range than for TTL. The 5 V CMOS can tolerate supply variations from 2 V to 6 V and still operate properly although propagation delay time and power dissipation are significantly affected. The 3.3 V CMOS can operate with supply voltages from 2 V to 3.6 V. The typical dc supply voltage for TTL is 5.0 V with a minimum of 4.5 V and a maximum of 5.5 V.

Power Dissipation The **power dissipation,** P_D, of a logic gate is the product of the dc supply voltage and the average supply current. Normally, the supply current when the gate output is LOW is greater than when the gate output is HIGH. The manufacturer's data sheet usually designates the supply current for the LOW output state as I_{CCL} and for the HIGH state as I_{CCH}. The average supply current is determined based on a 50% duty cycle (output LOW half the time and HIGH half the time), so the average power dissipation of a logic gate is

$$P_D = V_{CC}\left(\frac{I_{CCH} + I_{CCL}}{2}\right)$$

CMOS series gates have very low power dissipations compared to the TTL series. However, the power dissipation of CMOS is dependent on the frequency of operation. At zero frequency the quiescent power is typically in the microwatt/gate range, and at the maximum operating frequency it can be in the low milliwatt range; therefore, power is sometimes specified at a given frequency. The HC series, for example, has a power of 2.75 μW/gate at 0 Hz (quiescent) and 600 μW/gate at 1 MHz.

Power dissipation for TTL is independent of frequency. For example, the ALS series uses 1.4 mW/gate regardless of the frequency and the F series uses 6 mW/gate.

Input and Output Logic Levels V_{IL} is the LOW level input voltage for a logic gate, and V_{IH} is the HIGH level input voltage. The 5 V CMOS accepts a maximum voltage of 1.5 V as V_{IL} and a minimum voltage of 3.5 V as V_{IH}. TTL accepts a maximum voltage of 0.8 V as V_{IL} and a minimum voltage of 2 V as V_{IH}.

V_{OL} is the LOW level output voltage and V_{OH} is the HIGH level output voltage. For 5 V CMOS, the maximum V_{OL} is 0.33 V and the minimum V_{OH} is 4.4 V. For TTL, the maximum V_{OL} is 0.4 V and the minimum V_{OH} is 2.4 V. All values depend on operating conditions as specified on the data sheet.

Speed-Power Product (SPP) This parameter (**speed-power product**) can be used as a measure of the performance of a logic circuit taking into account the propagation delay time and the power dissipation. It is especially useful for comparing the various logic gate series within the CMOS or TTL family or for comparing a CMOS gate to a TTL gate.

The SPP of a logic circuit is the product of the propagation delay time and the power dissipation and is expressed in joules (J), which is the unit of energy. The formula is

$$SPP = t_P P_D$$

Fan-Out and Loading The **fan-out** of a logic gate is the maximum number of inputs of the same series in an IC family that can be connected to a gate's output and still maintain the output voltage levels within specified limits. Fan-out is a significant parameter only for TTL because of the type of circuit technology. Since very high impedances are associated with CMOS circuits, the fan-out is very high but depends on frequency because of capacitive effects.

A certain gate has a propagation delay of 5 ns and $I_{CCH} = 1$ mA and $I_{CCL} = 2.5$ mA with a dc supply voltage of 5 V. Determine the speed-power product.

$$P_D = V_{CC}\left(\frac{I_{CCH} + I_{CCL}}{2}\right) = 5\text{ V}\left(\frac{1\text{ mA} + 2.5\text{ mA}}{2}\right) = 5\text{ V}(1.75\text{ mA}) = 8.75\text{ mW}$$

$$SPP = (5\text{ ns})(8.75\text{ mW}) = \mathbf{43.75\ pJ}$$

Fan-out is specified in terms of **unit loads.** A unit load for a logic gate equals one input to a like circuit. For example, a unit load for a 74LS00 NAND gate equals *one* input to another logic gate in the 74LS series (not necessarily a NAND gate). Because the current from a LOW input (I_{IL}) of a 74LS00 gate is 0.4 mA and the current that a LOW output (I_{OL}) can accept is 8.0 mA, the number of unit loads that a 74LS00 gate can drive in the LOW state is

$$\text{Unit loads} = \frac{I_{OL}}{I_{IL}} = \frac{8.0\text{ mA}}{0.4\text{ mA}} = 20$$

Figure 3–64 shows LS logic gates driving a number of other gates of the same circuit technology, where the number of gates depends on the particular circuit technology. For example, as you have seen, the maximum number of gate inputs (unit loads) that a 74LS series TTL gate can drive is 20.

FIGURE 3–64

The LS TTL NAND gate output fans out to a maximum of 20 LS TTL gate inputs.

4 BOOLEAN ALGEBRA AND LOGIC SIMPLIFICATION

CHAPTER OUTLINE

4-1 Boolean Operations and Expressions
4-2 Laws and Rules of Boolean Algebra
4-3 DeMorgan's Theorems
4-4 Boolean Analysis of Logic Circuits
4-5 Simplification Using Boolean Algebra
4-6 Standard Forms of Boolean Expressions
4-7 Boolean Expressions and Truth Tables
4-8 The Karnaugh Map
4-9 Karnaugh Map SOP Minimization
4-10 Karnaugh Map POS Minimization
4-11 Five-Variable Karnaugh Maps
4-12 VHDL (optional)
Digital System Application

Introduction

- 1854, George Boole paper:“An Investigation of the Laws of Thought” Founded the Mathematical rules of Logic and probabilities.
- Was formulated a convenient and systematic way of expressing and analyzing of logic circuits. "logical algebra” (Boolean algebra),
- Claude Shannon Was the first to apply Boole's work to the analysis and design of logic circuits. In 1938,
- Shannon wrote a thesis at MIT titled “A Symbolic Analysis of Relay and Switching Circuits”

In this chapter:

- The laws and theorems of Boolean algebra and Their application to digital
- Define a circuit with a Boolean expression and evaluate its operation
- How to simplify logic circuits (Boolean algebra and Karnaugh maps)
- VHDL for programming logic devices is introduced.

Chapter Objectives

- Apply the basic laws and rules of Boolean algebra
- Apply DeMorgan's theorems to Boolean expressions
- Describe gate networks with Boolean expressions
- Evaluate Boolean expressions
- Simplify expressions by using the laws and rules of Boolean algebra
- Convert any Boolean expression into a sum-of-products (SOP) form
- Convert any Boolean expression into a product of-sums (POS) form
- Use a Karnaugh map to simplify Boolean expressions
- Use a Karnaugh map to simplify truth table functions
- Utilize "don't care" conditions to simplify logic functions
- Write a VHDL program for simple logic
- Apply Boolean algebra, the Karnaugh map method, and VHDL

to a system application

4-1 BOOLEAN OPERATIONS AND EXPRESSIONS

Variable: is a symbol (usually an italic uppercase letter) used to represent a logical quantity that can be 1 or a 0.

Complement: is the inverse of a variable and is indicate by a bar over the variable (overbar).

Literal: is a variable or the complement of a variable.

Boolean Addition

The OR gate is a Boolean Adder.

Boolean addition is equivalent to the OR operation, In Boolean algebra, a sum term is a sum of literals. In logic circuits, a sum term is produced by an OR operation with no AND operations involved.

$$A + B, A + \overline{B}, A + B + \overline{C}, \text{ and } \overline{A} + B + C + \overline{D}.$$

A sum term=1 when one or more of the literals in the term are 1.
A sum term=0 only if each of the literals is 0.

EXAMPLE 4-1

Determine the values of A, B, C, and D that make the sum term $A + \bar{B} + C + \bar{D}$ equal to 0.

Solution

For the sum term to be 0, each of the literals in the term must be 0. Therefore, A = 0, $\bar{B} = 0$ so $B = 1$, C = 0, and $\bar{D} = 0$ so $D = 1$.

$$A + \bar{B} + C + \bar{D} = 0 + \bar{1} + 0 + \bar{1} = 0 + 0 + 0 + 0 = 0$$

Related Problem Determine the values of A and B that make the sum term $\bar{A} + B$ equal to 0.

Boolean Multiplication

The AND gate is a Boolean multiplier.

In Boolean algebra, a product term is the product of literals.
In logic circuits, a product term is produced by an AND operation without OR operations involved.

AB, A$\bar{B}$, ABC, and A$\bar{B}$ $\bar{C}$D.

A product term=1 only if each of the literals in the term is 1.
A product term=0 when one or more of the literals are 0.

EXAMPLE 4-2

Determine the values of A, B, C, and D that make the product term $A\bar{B}C\bar{D}$ equal to 1.

Solution

For the product term to be 1, each of the literals in term must be 1.

$$A\bar{B}C\bar{D} = 1 \cdot \bar{0} \cdot 1 \cdot \bar{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

Related Problem Determine the values of A and B that make the product term $\bar{A}\bar{B}$ equal to 1.

SECTION 4-1 REVIEW

1. If A = 0, what does $\bar{A}$ equal?

2. Determine the values of A, B, and C that make the sum term $\bar{A}$+ B + $\bar{C}$ equal to 0.

3. Determine the values of A, B, and C that make the product term $A\bar{B}\bar{C}$ equal to 1.

4-2 LAWS AND RULES OF BOOLEAN ALGEBRA

Commutative Law of addition $A + B = B + A$

Commutative Law of multiplication $A.B = B.A$

Associative Laws

Associative Law of addition

$$A + (B + C) = (A + B) + C$$

Associative Law of multiplication

$$A.(B.C) = (A.B).C$$

Distributive Laws

Distributive of multiplication with respect to addition and factoring

$$A.(B + C) = A.B + A.C$$

Distributive of addition with respect to multiplication

$$A + (B.C) = (A + B).(A + C)$$

دقت کنید که بر خلاف قضایای قبلی این قضیه در جبر معمولی برقرار نیست و در جبر بول معتبر است.

Rules of Boolean Algebra

Basic rules that are useful in manipulating and simplifying Boolean expressions.

1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

A, *B*, or *C* can represent a single variable or a combination of variables.

همه این قواعد و قضایا با استفاده از جدول ارزش و تعاریف عملکرد گیتها قابل اثباتند.

Rules of Boolean Algebra

عضو خنثی

$X = A + 0 = A$

$X = A \cdot 1 = A$

$X = A + 1 = 1$

$X = A \cdot 0 = 0$

Rules of Boolean Algebra

$X = A + A = A$

$X = A + \bar{A} = 1$

$X = A \bullet A = A$

$X = A \bullet \bar{A} = 0$

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

Rules of Boolean Algebra

Rule 10. $A + AB = A$

$A + AB = A.1 + A.B$	Rule 4: $A . 1 = A$
$= A.(1+B)$	factoring (distributive)
$= A.1$	Rule 2: $(1+B) = 1$
$= A$	

A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal

Rules of Boolean Algebra

Rule 11. $A + \overline{A}B = A + B$

$A + \overline{A}B = (A + AB) + \overline{A}B$

$= (AA + AB) + \overline{A}B$

$= AA + AB + A\overline{A} + \overline{A}B$

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

$= 1.\ (A + B) = A + B$

با استفاده از توزیع پذیری جمع نسبت به ضرب این قضیه را ساده تر می توان اثبات نمود. (قضیه ۱۲)

A	B	$\overline{A}B$	$A + \overline{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

equal

Rules of Boolean Algebra

Rule 12. $(A + B)(A + C) = A + BC$ توزیع پذیری جمع نسبت به ضرب

$$(A + B)(A + C) = AA + AC + AB + BC$$
$$= A + AC + AB + BC$$
$$= A(1 + C) + AB + BC$$
$$= A.\ 1 + AB + BC$$
$$= A(1 + B) + BC$$
$$= A.\ 1 + BC = A + BC$$

A	B	C	A + B	A + C	(A + B)(A + C)	BC	A + BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal

SECTION 4-2 REVIEW QUESTIONS

1. Apply the associative law of addition to the expression

$$A + (B + C + D).$$

2. Apply the distributive law to the expression $A(B + C + D)$.

4-3 DEMORGAN'S THEOREMS

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practicaJ terms. DeMorgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates.

$$\overline{XY} = \overline{X} + \overline{Y}$$

Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0		
0	1		
1	0		
1	1		

$$\overline{X + Y} = \overline{X}\,\overline{Y}$$

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{X}\,\overline{Y}$
0	0		
0	1		
1	0		
1	1		

Can be used for more than two variable

EXAMPLE 4-5

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

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

)

EXAMPLE 4-6

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

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

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

EXAMPLE 4-7

The Boolean expression for an exclusive-OR gate is $\bar{A}B + A\bar{B}$. With this as a starting point, use DeMorgan's theorems and any other rules or laws that are applicable to develop an expression for the exclusive-NOR gate.

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

$$(\overline{A} + B)(A + \overline{B}) = \overline{A}A + \overline{A}\,\overline{B} + AB + B\overline{B} = \overline{A}\,\overline{B} + AB$$

The final expression for the XNOR is $\bar{A}\bar{B} + AB$. Note that this expression equals 1 any time both variables are 0s or both variables are 1s.

SECTION 4-3 REVIEW QUESTIONS

1. Apply DeMorgan's theorems to the following expressions:

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

4-4 BOOLEAN ANALYSIS OF LOGIC CIRCUITS

Boolean algebra provides a concise way to express the operation of a logic circuit formed by a combination of logic gates so that the output can be determined for various combinations of input values.

Boolean Expression for a Logic Circuit

Begin at the left-most inputs and work toward the final output, writing the expression for each gate.

Constructing a Truth Table for a Logic Circuit

For Putting the Results in Truth Table format of previous example:

Once the Boolean expression for determined, a truth table that shows

1) list the sixteen input variable of the input variables can be devel combinations of 1s and 0s in a binary sequence.

you evaluate the Boolean expressio

2) place a 1 in the output column for each combination of input variables that was determined in the evaluation.

values for the input variables

3) Finally, place a 0 in the output column for all other combinations of input variables.

INPUTS				OUTPUT
A	B	C	D	A(B + CD)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

SECTION 4-4 REVIEW QUESTIONS

1. Replace the AND gates with OR gates and the OR gate with an AND gate in Figure 4-16 and determine the Boolean expression for the output.
2. Construct a truth table for the circuit in Question 1.

4-5 SIMPLIFICATION USING BOOLEAN ALGEBRA

هر دو مداری که به ازای ورودیهای یکسان خروجی یکسان بدهند معادل هم (نه مساوی) خواهند بود.

در بین دو مدار معادل هر چه تعداد عناصر کمتر باشد انتظار داریم که مدار:

- ارزانتر باشد
- دارای تاخیر کمتری باشد
- عیب یابی آن راحتتر باشد.
- حجم کمتری از برد را اشغال کند و ...

4-5 SIMPLIFICATION USING BOOLEAN ALGEBRA

Many times in the application of Boolean algebra, you have to reduce a particular expression to its simplest form or change its form to a more convenient one to implement the expression most efficiently. The approach taken in this section is to use the basic laws, rules, and theorems of Boolean algebra to manipulate and simplify an expression. This method depends on a thorough knowledge of Boolean algebra and considerable practice in its application, not to mention a little ingenuity and cleverness.

At the first we try to using Demorgan's theorems and distribution and any other needed rules to convert Boolean expressions to the form that only include variables or literals

EXAMPLE 4-8

Using Boolean algebra techniques, simplify this expression:

$$AB + A(B + C) + B(B + C)$$

Solution

AB + AB + AC + BB + BC	distributive
AB + AB + AC + B + BC	BB=B
AB + AC + B + BC	AB+AB=AB
AB + AC + B	(B + BC = B)
B + AC	AB + B = B

Related Problem Simplify the Boolean expression:

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

EXAMPLE 4-9

Simplify the following Boolean expression:

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

Solution

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

2) $(A\overline{B}C + A \cdot 0 \cdot D + \overline{A}\,\overline{B})C$

3) $(A\overline{B}C + 0 + \overline{A}\,\overline{B})C$

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

5) $A\overline{B}CC + \overline{A}\,\overline{B}C$

6) $A\overline{B}C + \overline{A}\,\overline{B}C$

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

8) $\overline{B}C \cdot 1 = \overline{B}C$

Related Problem Simplify the Boolean expression

$$[AB(C + \overline{BD}) + \overline{AB}]CD.$$

EXAMPLE 4-10

Simplify the following Boolean expression:

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

Solution

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

$$BC \cdot 1 + A\overline{B}(\overline{C} + C) + \overline{A}\,\overline{B}\,\overline{C}$$

$$BC + A\overline{B} \cdot 1 + \overline{A}\,\overline{B}\,\overline{C}$$

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

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

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

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

Related Problem Simplify the Boolean expression

$$AB\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}\,\overline{C}.$$

EXAMPLE 4-11

Simplify the following Boolean expression:

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

Solution

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

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

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

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

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

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

Related Problem Simplify the Boolean expression

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

SECTION 4-5 REVIEW QUESTIONS

1. Simplify the following Boolean expressions if possible:

(a) $A + AB + A\overline{B}C$ (b) $(\overline{A} + B)C + ABC$ (c) $A\overline{B}C(BD + CDE) + A\overline{C}$

2. Implement each expression in Question 1 as originally stated with the appropriate logic gates. Then implement the simplified expression, and compare the number of gates.

4-6 STANDARD FORMS OF BOOLEAN EXPRESSIONS

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of-sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

Sum-Of-Products (SOP).

A product term was defined as a term consisting of the product of literals (variables or their complements). When two or more product terms are summed by Boolean addition. the resulting expression is a sum-of-products (SOP).

A + BCD

$$AB + ABC$$

$$ABC + CDE + \overline{B}C\overline{D}$$

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

Domain of a Boolean Expression

The domain of a general Boolean expression is the set of variables contained in the expression in either complemented or uncomplemented form.

The domain of $AB + AB\overline{C}$ is the set of variables A, B, C and the domain of the expression $A\overline{B}C + CDE + B\overline{C}\overline{D}$ is the set of variables A, B, C, D, E.

AND/OR Implementation of an SOP Expression

Simplification of any Boolean expressions can results an SOP expression.

Any SOP expression can be implemented by two level AND-OR logic.

NAND/NAND Implementation of an SOP Expression

For converting any AND-OR implementation to NAND-NAND we need only replace any AND-OR gates with NAND gates. Also we can replace any NOT gate with a two input NAND gate that its inputs tied together.

EXAMPLE 4-12

Convert each of the following Boolean expressions to SOP form.

(a) $AB + B(CD + EF)$ (b) $(A + B)(B + C + D)$ (c) $\overline{\overline{(A + B)} + C}$

SOLUTION

(a) $AB + B(CD + EF) = AB + BCD + BEF$

(b) $(A + B)(B + C + D) = AB + AC + AD + BB + BC + BD$

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

Related Problem Convert $\overline{A}B\overline{C} + (A + \overline{B})(B + \overline{C} + A\overline{B})$ to SOP form.

The Standard SOP Form (CANONIC)

A standard SOP expression is one in which all the variables in the domain appear in each product term in the expression. This product term called **minterm.**

Standard SOP expressions are important in constructing truth tables, and in the Karnaugh map simplification method.

Converting Product Terms to Standard SOP

Step 1. Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms. As you know, you can multiply anything by I without changing its value.

Step 2. Repeat Step 1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form.

EXAMPLE 4-13

Convert the following Boolean expression into standard SOP form:

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

SOLUTION

The domain of this SOP expression is A, B, C, D.

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

At the end:

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

Related Problem Convert the following expression to standard SOP form.

$$W\bar{X}Y + \bar{X}Y\bar{Z} + WX\bar{Y}$$

Binary Representation of a Standard Product Term

A standard product term is equal to 1 for only one combination of variable values.

$$A\overline{B}C\overline{D} = 1 \cdot \overline{0} \cdot 1 \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

In this case, minterm has a binary value of 1010 (decimal ten).

An SOP expression is equal to 1 only if one or more of the minterms in the expression is equal to 1.

EXAMPLE 4-14

Determine the binary values for which the following standard SOP expression is equal to 1:

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

SOLUTION

$$ABCD = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$A\overline{B}\,\overline{C}D = 1 \cdot \overline{0} \cdot \overline{0} \cdot 1 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = \overline{0} \cdot \overline{0} \cdot \overline{0} \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

The expression equals 1 when any or all of the three product terms is 1. respective minterm values are 15, 9, 0

Related Problem Determine the binary values for which the following SOP expression is equal to 1. Is this a standard SOP expression?

$$\overline{X}YZ + X\overline{Y}Z + XY\overline{Z} + \overline{X}Y\overline{Z} + XYZ$$

The Product-of-Sums (POS) Form

A sum term was defined in Section 4-1 as a term consisting of the sum (Boolean addition) of literals (variables or their complements). When two or more sum terms are multiplied, the resulting expression is a **product-of-sums (POS)**.

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

Implementation of a POS Expression: Implementing a POS expression simply requires ANDing the outputs of two or more OR gates.

The Standard POS Form

A standard POS expression is one in which all the variables in the domain appear in each sum term in the expression. Each sum terms in a standard POS expression is called a **Maxterm**.

Non standard: $(A + \overline{B} + C)(A + B + \overline{D})(A + \overline{B} + \overline{C} + D)$

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

Converting a Sum Term to Standard POS :

Step 1. Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms.

Step 2. Apply rule 12 distribution: A + BC = (A + B)(A + C)

Step 3. Repeat Step 1 until all resulting sum terms contain all variables in the domain in either complemented or uncomplemented form.

EXAMPLE 4-15

Convert the following Boolean expression into standard POS form:

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

SOLUTION

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

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

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

Related Problem Determine the standard POS expressions for:

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

Binary Representation of a Standard Sum Term (Maxterm)

A standard sum term is equal to 0 tor only one combination of variable values, and is 1 for all other combinations of values for the variables.

$$A + \overline{B} + C + \overline{D} = 0 + \overline{1} + 0 + \overline{1} = 0 + 0 + 0 + 0 = 0$$

EXAMPLE 4-16

Determine the binary values of the variables for which the following standard POS expression is equal to 0:

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

SOLUTION

$$A + B + C + D = 0 + 0 + 0 + 0 = 0$$

$$A + \overline{B} + \overline{C} + D = 0 + \overline{1} + \overline{1} + 0 = 0 + 0 + 0 + 0 = 0$$

$$\overline{A} + \overline{B} + \overline{C} + \overline{D} = \overline{1} + \overline{1} + \overline{1} + \overline{1} = 0 + 0 + 0 + 0 = 0$$

The POS expression equals 0 when any of the sum terms equals 0. Maxterm values that make the output to 0 are 0, 6, 15.

Related Problem Determine the binary values for which the following POS expression is equal to 0. Is this a standard POS expression?

$$(X + \overline{Y} + Z)(\overline{X} + Y + Z)(X + Y + \overline{Z})(\overline{X} + \overline{Y} + \overline{Z})(X + \overline{Y} + \overline{Z})$$

Converting Standard SOP to Standard POS

The binary values of the product terms in a given standard SOP expression are not present in the equivalent standard POS expression. Therefore, to convert from standard SOP to standard POS, the following steps are taken:

Step 1. Evaluate each product term in the SOP expression. That is, determine the binary numbers that represent the product terms.

Step 2. Determine all of the binary numbers not included in the evaluation in Step I.

Step 3. Write the equivalent sum term for each binary number from Step 2 and express in POS form.

Using a similar procedure, you can go from POS to SOP.

EXAMPLE 4-17

Convert the following SOP expression to an equivalent POS expression:

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

SOLUTION

The evaluation is as follows: 000 + 010 + 011 + 101 + 111

Since there are three variables in the domain of this expression. there are a total of 8 possible combinations. The SOP expression contains five of these combinations, so the POS must contain the other three which are 001,100, and 110.

Remember, these are the binary values that make the sum term O. The equivalent POS expression is:

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

Related Problem Verify that the SOP and POS expressions in this example are equivalent by substituting binary values into each.

SECTION 4-6 REVIEW

1. Identify each of the following expressions as SOP, standard SOP, POS, or standard POS:

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

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

(c) $\overline{A}BC + AB\overline{C}$

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

2. Convert each SOP expression in Question 1 to standard form.

3. Convert each POS expression in Question 1 to standard form.

4-7 BOOLEAN EXPRESSIONS AND TRUTH TABLES

All standard Boolean expressions can be easily converted into truth table format using binary values for each term (**minterms or Maxterms**) in the expression. The truth table is a common way of presenting, in a concise format, the logical operation of a circuit.

Also. standard SOP or POS expressions can be determined from a truth table.

Converting SOP Expressions to Truth Table Format

1- Constructing a truth table and list all possible combinations of binary values of the variables in the expression.

2- Convert the SOP expression to standard form if it is not already.

3- Place a 1 in the output column (X) for each binary value that makes the standard SOP expression a 1 and place a 0 for all the remaining binary values.

EXAMPLE 4-18

Develop a truth table for the standard SOP expression

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

SOLUTION

Minterms that make 1 in out put are 001,100,111 (m1, m4, m7)

INPUTS A	B	C	OUTPUT X	PRODUCT TERM
0	0	0	0	
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Related Problem Create a truth table for the standard SOP expression

$$\overline{A}B\overline{C} + A\overline{B}C.$$

Converting POS Expressions to Truth Table Format

Recall that a POS expression is equal to 0 only if at least one of the sum terms (Maxterms) is equal to 0.

1- List all the possible combinations of binary values of the variables just as was done for the POS expression.

2- Convert the POS expression to standard form if it is not already.

3- Place a 0 in the output column (X) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values.

EXAMPLE 4-19

Determine the truth table for the following standard POS expression:

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

SOLUTION

Maxterms that make 0 in output are 000, 010, 011, 101, 110 (M0,M2,M3,M5,M6)

INPUTS			OUTPUT	
A	B	C	X	SUM TERM
0	0	0	0	$(A + B + C)$
0	0	1	1	
0	1	0	0	$(A + \overline{B} + C)$
0	1	1	0	$(A + \overline{B} + \overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A} + B + \overline{C})$
1	1	0	0	$(\overline{A} + \overline{B} + C)$
1	1	1	1	

Related Problem Develop a truth table for the following standard POS expression:

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

Determining Standard Expressions from a Truth Table

SOP:

1- List the binary values of the input variables that make output=1.

2- Convert each binary value to the corresponding product term by replacing each 1 with the corresponding variable and each 0 with the corresponding variable complement.

3- Use OR operation for combining previous steps minterms.

POS:

1- List the binary values of the input variables that make output=0.

2- Convert each binary value to the corresponding sum term by replacing each 0 with the corresponding variable and each 1 with the corresponding variable complement.

3- Use AND operation for combining previous steps Maxterms.

EXAMPLE 4-20

From the truth table, determine the standard SOP expression and the equivalent standard POS expression.

INPUTS			OUTPUT
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

SOLUTION

miterms that make 1 in output are:

$$011 \longrightarrow \overline{A}BC$$
$$100 \longrightarrow A\overline{B}\,\overline{C}$$
$$110 \longrightarrow AB\overline{C}$$
$$111 \longrightarrow ABC$$

So SOP expression is:

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

EXAMPLE 4-19

Maxterms that make 0 in output are:

$$000 \longrightarrow A + B + C$$
$$001 \longrightarrow A + B + \overline{C}$$
$$010 \longrightarrow A + \overline{B} + C$$
$$101 \longrightarrow \overline{A} + B + \overline{C}$$

INPUTS			OUTPUT
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

So POS expression is:

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

SECTION 4-7 REVIEW

1. If a certain Boolean expression has a domain of five variables, how many binary values will be in its truth table?
2. In a certain truth table, the output is a 1 for the binary value 0110. Convert this binary value to the corresponding product term using variables W, X, Y, and Z.
3. In a certain truth table, the output is a 0 for the binary value 1100. Convert this binary value to the corresponding sum term using variables W, X, Y, and Z.

4-8 THE KARNAUGH MAP

ساده سازی جبری دارای این مشکلات است:

- از روی عبارت اولیه قابل تشخیص نیست که آیا اساسا عبارت ساده شدنی است یا خیر
- فاقد یک روش مشخص است و مشخص نیست چه مراحلی را باید طی کنیم.
- ساده سازی به هوش، مهارت و تجربه شخص وابسته است.
- در انتها نیز نمی توان با قطعیت گفت که آیا عبارت حاصله ساده تر می شود یا خیر.

در مقابل روشهای سیستماتیک نظیر جدول کارنو و روش کوئین مک کلوسکی با هیچیک از مشکلات فوق مواجه نیستند و با فرض آنکه کاربران روش را درست بکار برند حتما به جوابهایی با سطح پیچیدگی یکسان خواهند رسید.
در اینجا ما با روش جدول کارنو به عنوان یک روش سیستماتیک دستی که حداکثر برای مدارهایی با ۵ متغیر قابل استفاده است آشنا خواهیم شد.

.

THE KARNAUGH MAP

- Another representation of truth table.
- Instead of organizing into columns and rows , the Karnaugh map is an array of cells in which each cell represents a binary value of the input variables. (midterm or maxterm value)
- The cells are arranged in a way that the position of 1 or 0 in map inform us how to combine the minterms or maxterms for simplifing expressions.
- The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table.

THE KARNAUGH MAP

3 variables

Row & column numbering is in gray code so any horizontal or vertical adjacent cell differs only in one parameter.
Any cell can uniquely related to a minterm or maxterm

AB \ C	0	1
00	0	1
01	2	3
11	6	7
10	4	5

(a)

AB \ C	0	1
00	$A\bar{B}\bar{C}$	$\bar{A}\bar{B}C$
01	$\bar{A}B\bar{C}$	$\bar{A}BC$
11	$AB\bar{C}$	ABC
10	$A\bar{B}\bar{C}$	$A\bar{B}C$

(b)

4 variables

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

(a)

AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
11	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

(b)

CELL ADJUCANCY

هر دو سلولی که تنها در یک متغیر با هم تفاوت دارند مجاور نامیده می شوند و بنابراین

- هر دو سلولی که حداقل یک ضلع مشترک داشته باشند
- عناصر متناظر سطر بالایی با پائینی
- عناصر متناظر در چپ ترین ستون با راسترین ستون

SECTION 4-8 REVIEW

1. In a 3-variable Karnaugh map, what is the binary value for the cell in each of the following locations:

   (a) upper left corner (b) lower right corner

   (c) lower left corner (d) upper right corner

2. What is the standard product term for each cell in Question 1 for variables X, Y, and Z?

3. Repeat Question 1 for a 4-variable map.

4. Repeat Question 2 for a 4-variable map using variables W, X, Y, and Z.

4-9 KARNAUGH MAP SOP MINIMIZATION

برای نگاشتن عبارت جبری در جدول کارنو برای ساده سازی SOP:

۱- عبارت جبری را بفرم استاندارد SOP در می آوریم.

۲- در جدول کارنو مربوطه در سلولهای متناظر با مینترمها عدد ۱ را قرار می دهیم. بقیه خانه ها در واقع ۰ هستند ولی نیازی به نگارش آنها نیست.

برای انتقال عبارت جبری معمولا دانستن شماره سلولها مهم نیست ولی هنگامی که قصد داریم مستقیما از جدول ارزش به جدول کارنو برویم سرعت ما را افزایش می دهد.

EXAMPLE 4-21

Map the following standard SOP expression on a Karnaugh map:

$$\bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC$$

solution:

001	$\bar{A}\bar{B}C$
010	$\bar{A}B\bar{C}$
110	$AB\bar{C}$
111	ABC

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

Related Problem: Map the standard SOP expression $\bar{A}BC + A\bar{B}\ C + A\bar{B}\bar{C}$ on a Karnaugh map.

EXAMPLE 4-22

Map the following standard SOP expression on a Karnaugh map:

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

solution:

0011	$\bar{A}\bar{B}CD$
0100	$\bar{A}B\overline{CD}$
1101	$AB\bar{C}D$
1111	$ABCD$
1100	$AB\overline{CD}$
0001	$\bar{A}\bar{B}\bar{C}D$
1010	$A\bar{B}C\bar{D}$

	$\bar{C}\,\bar{D}$	$\bar{C}\,D$	$C\,D$	$C\,\bar{D}$
$\bar{A}\,\bar{B}$				
$\bar{A}\,B$				
$A\,B$				
$A\,\bar{B}$				

Related Problem: Map the standard SOP expression $\bar{A}BC\bar{D} + ABC\bar{D} + AB\bar{C}D + ABCD$ on a Karnaugh map.

Mapping a Nonstandard SOP Expression

A Boolean expression must first be in standard form before you use a Karnaugh map.

ALGEBRAIC METHOD

A+$\bar{A}$BC =

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

$= A\bar{C}\bar{B} + A\bar{C}B + AC\bar{B} + ACB + \bar{A}BC$

	$\bar{C}$	C
$\bar{A}\ \bar{B}$		
$\bar{A}\ B$		1
$A\ B$	1	1
$A\ \bar{B}$	1	1

NUMERICAL EXPANSION

A	BC	
1	00	$A\bar{B}\bar{C}$
1	01	$A\bar{B}C$
1	10	$AB\bar{C}$
1	11	ABC

EXAMPLE 4-23

Map the following nonstandard SOP expression on a Karnaugh map:

$$\overline{A} + A\overline{B} + AB\overline{C}.$$

solution:

$\overline{A}$	$+ A\overline{B}$	$+ AB\overline{C}$
000	100	110
001	101	
010		
011		

Related Problem: Map the SOP expression BC + $\bar{A}\bar{C}$ on a Karnaugh map.

EXAMPLE 4-24

Map the following nonstandard SOP expression on a Karnaugh map:
$\bar{B}\bar{C} + A\bar{C} + AB\bar{C} + A\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}CD$

solution:

$\bar{B}\bar{C}$	$A\bar{B}$	+	$AB\bar{C}$	+ $A\bar{B}C\bar{D}$	+ $\bar{A}\bar{B}\bar{C}D$	+ $A\bar{B}CD$
0000	1000		1100	1010	0001	1011
0001	1001		1101			
1000	1010					
1001	1011					

Related Problem: Map the SOP expression $A + \bar{C}D + AC\bar{D} + \bar{A}BC\bar{D}$ on a Karnaugh map.

Karnaugh Map Simplification of SOP Expressions

The process that results in an expression containing the fewest possible terms with the fewest possible variables is called **minimization**.

After an SOP expression has been mapped, a minimum SOP expression is obtained by *grouping the Is and determining the minimum SOP expression from the map*.

1. A group must contain either 1, 2, 4, 8, or 16 cells
2. Each cell in a group must be adjacent to one or more cells in that same group.
3. Always include the largest possible number of 1's in a group in accordance with rule 1.
4. Each 1 on the map must be included in at least one group. The 1's already in a group can be included in another group as long as the overlapping groups include noncommon 1's.

EXAMPLE 4-25

Group the 1's in each of the below Karnaugh maps.

solution:

EXAMPLE 4-25

Group the 1's in each of the below Karnaugh maps.

solution:

Related Problem: Determine if there are other ways to group the 1's in Figure 4-30 to obtain a minimum number of maximum groupings.

Determining the Minimum SOP Expression from the Map

1. Group the cells that have 1's. Each group of cells containing 1's creates one product term composed of all comone and unchanged variables.
2. Determine the minimum product term for each group.
3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression.

EXAMPLE 4-26

Determine the product terms for the Karnaugh map in Figure 4-31 and write the resulting minimum SOP expression.

solution:

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

Related Problem: For the Karnaugh map in Figure 4-31, add a 1 in the lower right cell (1010) and detennine the resulting SOP expression.

EXAMPLE 4-27

Detelmine the product telms for each of the Karnaugh maps in Figure 4-32 and write the resulting minimum SOP expression.

solution:

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

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

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

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

Related Problem: For the Kamaugh map in Figure 4-32(d), add a I in the 0111 cell and determine the resulting SOP expression.

EXAMPLE 4-28

Use a Kamaugh map to minimize the following standard SOP expression:

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

solution:

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

101 011 001 000 100

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

Related Problem: Use a Karnaugh map to simplify the following standard SOP expression:

$$X\bar{Y}Z + XY\bar{Z} + \bar{X}YZ + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$

EXAMPLE 4-29

Use a Kamaugh map to minimize the following standard SOP expression:

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

solution:

$$\overline{D} + \overline{B}C$$

Related Problem: Use a Karnaugh map to simplify the following standard SOP expression:

$$\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} + W\overline{X}YZ + W\overline{X}\,\overline{Y}\,\overline{Z} + W\overline{X}\,\overline{Y}Z + \overline{W}YZ$$

Mapping Directly from a Truth Table

$$X = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + AB\overline{C} + ABC$$

Inputs			Output
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Don’t Care Condition

Sometimes a situation arises in which some input variable combinations are not allowed. (ie BCD). Since these un allowed states will never occur in an application involving the BCD code, they can be treated as "don't care" terms with respect to their effect on the output. That is, for these "don't care" terms either a 1 or a 0 may be assigned to the output.

The "don't care" terms can be used to advantage on the Karnaugh map. for For each "don't care" term, an X is placed in the cell. When grouping the 1’s, the X’s can be treated as 1’s to make a larger grouping or as 0’s if they cannot be used to advantage.

The larger a group, the simpler the resulting term will be.

Don’t Care Condition

Inputs A B C D	Output Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	1
1 0 0 0	1
1 0 0 1	1
1 0 1 0	X
1 0 1 1	X
1 1 0 0	X
1 1 0 1	X
1 1 1 0	X
1 1 1 1	X

Don’t cares

(a) Truth table

"don't cares" are not used as 1’s:

AB C + ABCD

"don't cares" are used as 1’s:

A + BCD. Simpler expression

SECTION 4-9 REVIEW

1. Layout Karnaugh maps for three and four variables.
2. Group the 1's and write the simplified SOP expression for the Karnaugh map in Figure 4-25.
3. Write the original standard SOP expressions for each of the Karnaugh maps in Figure 4-32.

4-9 KARNAUGH MAP POS MINIMIZATION

برای نگاشتن عبارت جبری در جدول کارنو برای ساده سازی POS:

۱- عبارت جبری را بفرم استاندارد POS در می آوریم.

۲- در جدول کارنو مربوطه در سلولهای متناظر با ماکسترمها عدد ۰ را قرار می دهیم.

بقیه خانه ها در واقع ۱ هستند ولی نیازی به نمایش آنها نیست.

EXAMPLE 4-30

Map the following standard POS expression on a Karnaugh map:

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

solution:

$$\underset{1100}{(\overline{A}+\overline{B}+C+D)}\underset{1011}{(\overline{A}+B+\overline{C}+\overline{D})}\underset{0010}{(A+B+\overline{C}+D)}\underset{1111}{(\overline{A}+\overline{B}+\overline{C}+\overline{D})}\underset{0011}{(A+B+\overline{C}+\overline{D})}$$

Related Problem: Map the following standard POS expression on a Karnaugh map:

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

Karnaugh Map Simplification of POS Expressions

The process for minimizing a POS expression is basically the same as for an SOP expression except that you group 0's to produce minimum sum terms instead of grouping 1's to produce minimum product terms. The rules for grouping the 0's are the same as those for grouping the 1's.

EXAMPLE 4-31

Use a Karnaugh map to minimize the following standard POS expression:

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

solution:

Maxterms: 000, 001, 010, 011, 110

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

Minterms: 111, 101, 100

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

Related Problem: Use a Karnaugh map to simplify the following standard POS expression:

$$(X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(\overline{X} + \overline{Y} + Z)(\overline{X} + Y + Z)$$

EXAMPLE 4-32

Use a Karnaugh map to minimize the following POS expression:

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

solution:

Maxterms: 0000, 1000, 0010, 1001, 1000, 1100

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

Related Problem: Use a Karnaugh map to simplify the following POS expression:

$$(W + \overline{X} + Y + \overline{Z})(W + X + Y + Z)(W + \overline{X} + \overline{Y} + Z)(\overline{W} + \overline{X} + Z)$$

Converting Between POS and SOP Using the Karnaugh Map

When a POS expression is mapped, it can easily be converted to the equivalent SOP form directly from the Karnaugh map. Also, given a mapped SOP expression, an equivalent POS expression can be derived directly from the map. This provides a good way to compare both minimum forms of an expression to determine if one of them can be implemented with fewer gates than the other.

For a POS expression, all the cells that do not contain 0's contain 1's, from which the SOP expression is derived. Likewise, for an SOP expression, all the cells that do not contain 1's contain 0's, from which the POS expression is derived.

EXAMPLE 4-33

Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.

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

solution:

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

(b) Standard SOP:
$\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D} +$
$A\overline{B}\overline{C}\overline{D} + AB\overline{C}\overline{D} + A\overline{B}CD + ABCD$

EXAMPLE 4-33

(c) Minimum SOP: $AC + BC + BD + \overline{B}\,\overline{C}\,\overline{D}$

Related Problem: Use a Karnaugh map to convert the following expression to minimum SOP form:

$$(W + \overline{X} + Y + \overline{Z})(\overline{W} + X + \overline{Y} + \overline{Z})(\overline{W} + \overline{X} + \overline{Y} + Z)(\overline{W} + \overline{X} + \overline{Z})$$

SECTION 4-10 REVIEW

1. What is the difference in mapping a POS expression and an SOP expression?
2. What is the standard sum term expressed with variables A, B, C, and D for a 0 in cell 1011 of the Kamaugh map?
3. What is the standard product term expressed with variables A, B, C, and D for a 1 in cell 0010 of the Karnaugh map?

4–11 FIVE-VARIABLE KARNAUGH MAPS

Boolean functions with five variables can be simplified using a 32-cell Karnaugh map. Actually, two 4-variable maps (16 cells each) are used to construct a 5-variable map. You already know the cell adjacencies within each of the 4-variable maps and how to form groups of cells containing 1s to simplify an SOP expression. All you need to learn for five variables is the cell adjacencies between the two 4-variable maps and how to group those adjacent 1s.

- The term for the yellow group is $D\overline{E}$.
- The term for the orange group is $\overline{B}CE$.
- The term for the light red group is $\overline{A}B\overline{D}$.
- The term for the gray cell grouped with the red cell is $B\overline{C}\,\overline{D}E$.

Combining these terms into the simplified SOP expression yields

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

EXAMPLE 4–34

Use a Karnaugh map to minimize the following standard SOP 5-variable expression:

$$X = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E} + \overline{A}\,\overline{B}C\overline{D}\,\overline{E} + \overline{A}BC\overline{D}\,\overline{E} + \overline{A}B\overline{C}\,\overline{D}\,\overline{E} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}E + \overline{A}BC\overline{D}E$$
$$+ \overline{A}BCDE + A\overline{B}\,\overline{C}\,\overline{D}\,\overline{E} + A\overline{B}\,\overline{C}\,\overline{D}E + ABC\overline{D}E + ABCDE + A\overline{B}CDE$$

Map the SOP expression. Figure 4–44 shows the groupings and their corresponding terms. Combining the terms yields the following minimized SOP expression:

$$X + \overline{A}\,\overline{D}\,\overline{E} + \overline{B}\,\overline{C}\,\overline{D} + BCE + ACDE$$

SECTION 4-10 REVIEW

1. Why does as-variable Karnaugh map require 32 cells?
2. What is the expression represented by a 5-variable Karnaugh map in which each cell contains a 1?

4–12 VHDL (optional)

This optional section provides a brief introduction to VHDL and is not meant to teach the complete structure and syntax of the language. For more detailed information and instruction, refer to the footnote. Hardware description languages (HDLs) are tools for logic design entry, called text entry, that are used to implement logic designs in programmable logic devices. Although VHDL provides multiple ways to describe a logic circuit, only the simplest and most direct programming examples of text entry are discussed here.

The V in VHDL* stands for VHSIC (*V*ery *H*igh *S*peed *I*ntegrated *C*ircuit) and the HDL, of course, stands for hardware description language. As mentioned, **VHDL** is a standard language adopted by the IEEE (Institute of Electrical and Electronics Engineers) and is designated IEEE Std. 1076-1993. VHDL is a complex and comprehensive language and using it to its full potential involves a lot of effort and experience.

VHDL provides three basic approaches to describing a digital circuit using software: *behavioral, data flow,* and *structural.* We will restrict this discussion to the data flow approach in which you write Boolean-type statements to describe a logic circuit. Keep in mind that VHDL, as well as the other HDLs, is a tool for implementing digital designs and is, therefore, a means to an end and not an end in itself.

It is relatively easy to write programs to describe simple logic circuits in VHDL. The logical operators are the following VHDL keywords: **and, or, not, nand, nor, xor,** and **xnor.** The two essential elements in any VHDL program are the entity and the architecture, and they must be used together. The **entity** describes a given logic function in terms of its external inputs and outputs, called ports. The **architecture** describes the internal operation of the logic function.

In its simplest form, the entity element consists of three statements: The first statement assigns a name to a logic function; the second statement, called the *port* statement which is indented, specifies the inputs and outputs; and the third statement is the *end* statement. Although you would probably not write a VHDL program for a single gate, it is instructive to start with a simple example such as an AND gate. The VHDL entity declaration for a 2-input AND gate is

```
entity AND_Gate2 is
   port (A, B: in bit; X: out bit);
end entity AND_Gate2;
```

The blue boldface terms are VHDL keywords; the other terms are identifiers that you assign; and the parentheses, colons, and semicolons are required VHDL syntax. As you can see, A and B are specified as input bits and X is specified as an output bit. The port identifiers A, B, and X as well as the entity name AND_Gate2 are user-defined and can be renamed. As in all HDLs, the placement of colons and semicolons is crucial and must be strictly adhered to.

The VHDL architecture element of the program for the 2-input AND gate described by the entity is

```
architecture LogicFunction of AND_Gate2 is
begin
   X ⇐ A and B;
end architecture LogicFunction;
```

Again, the VHDL keywords are blue boldface, and the semicolons and the assignment operator ⇐ are required syntax. The first statement of the architecture element must reference the entity name.

The entity and the architecture are combined into a single VHDL program to describe an AND gate, as illustrated in Figure 4–45.

```
entity AND_Gate2 is
  port (A, B: in bit; X: out bit);
end entity AND_Gate2;

architecture LogicFunction of AND_Gate2 is
begin
  X <= A and B;
end architecture LogicFunction;
```

Writing Boolean Expressions in VHDL As you saw, the expression for a 2-input AND gate, $X = AB$, is written in VHDL as X ⇐ A **and** B;. Any Boolean expression can be written using VHDL keywords **not, and, or, nand, nor, xor,** and **xnor.** For example, the Boolean expression $X = A + B + C$ is written in VHDL as X ⇐ A **or** B **or** C;. The Boolean expression $X = A\overline{B} + \overline{C}D$ can be written as the VHDL statement X ⇐ (A **and not** B) **or** (**not** C **and** D);. As another example, the VHDL statement for a 2-input NAND gate can be written as X ⇐ **not**(A **and** B); or it can be written as X ⇐ A **nand** B;.

EXAMPLE 4–35

Write a VHDL program to describe the logic circuit in Figure 4–46.

FIGURE 4–46

Solution This AND/OR logic circuit is described in Boolean algebra as

$$X = AB + CD$$

The VHDL program follows. The entity name is AND_OR.

```
entity AND_OR is
   port (A, B, C, D: in bit; X: out bit);
end entity AND_OR;
architecture LogicFunction of AND_OR is
begin
   X <= (A and B) or (C and D);
end architecture LogicFunction;
```

SECTION 4–12 REVIEW

1. What is an HDL?
2. Name the two essential design elements in a VHDL program.
3. What does the entity do?
4. What does the architecture do?

DIGITAL SYSTEM APPLICATION

Seven-segment displays are used in many types of products. The tablet-counting and control system that was described in Chapter 1 has two 7-segment displays. These displays are used with logic circuits that decode a binary coded decimal (BCD) number and activate the appropriate digits on the display. In this digital system application, we focus on a minimum-gate design for this to illustrate an application of Boolean expressions and the Karnaugh map. As an option, VHDL is also applied.

The 7-Segment Display

Figure 4–47 shows a common display format composed of seven elements or segments. Energizing certain combinations of these segments can cause each of the ten decimal digits to be displayed. Figure 4–48 illustrates this method of digital display for each of the ten digits by using a red segment to represent one that is energized. To produce a 1, segments *b* and *c* are energized; to produce a 2, segments *a*, *b*, *g*, *e*, and *d* are used; and so on.

LED Displays One common type of 7-segment display consists of light-emitting diodes (**LEDs**) arranged as shown in Figure 4–49. Each segment is an LED that emits light when there is current through it. In Figure 4–49(a) the common-anode arrangement requires the driving circuit to provide a low-level voltage in order to activate a given segment. When a LOW is applied to a segment input, the LED is turned on, and there is current through it. In Figure 4–49(b) the common-cathode arrangement requires the driver to provide a high-level voltage to activate a segment. When a HIGH is applied to a segment input, the LED is turned on and there is current through it.

LCD Displays Another common type of 7-segment display is the liquid crystal display (**LCD**). LCDs operate by polarizing light so that a nonactivated segment reflects incident light and thus appears invisible against its background. An activated segment does not reflect incident light and thus appears dark. LCDs consume much less power than LEDs but

FIGURE 4–47

Seven-segment display format showing arrangement of segments.

FIGURE 4–48

Display of decimal digits with a 7-segment device.

FIGURE 4–49

Arrangements of 7-segment LED displays.

cannot be seen in the dark, while LEDs can.

Segment Logic

Each segment is used for various decimal digits, but no one segment is used for all ten digits. Therefore, each segment must be activated by its own decoding circuit that detects the occurrence of any of the numbers in which the segment is used. From Figures 4–47 and 4–48, the segments that are required to be activated for each displayed digit are determined and listed in Table 4–9.

Truth Table for the Segment Logic

The segment decoding logic requires four binary coded decimal (BCD) inputs and seven outputs, one for each segment in the display, as indicated in the block diagram of Figure 4–50. The multiple-output truth table, shown in Table 4–10, is actually seven truth tables in one and could be separated into a separate table for each segment. A 1 in the segment output columns of the table indicates an activated segment.

Since the BCD code does not include the binary values 1010, 1011, 1100, 1101, 1110, and 1111, these combinations will never appear on the inputs and can therefore be treated as "don't care" (X) conditions, as indicated in the truth table. To conform with the practice of most IC manufacturers, *A* represents the least significant bit and *D* represents the most significant bit in this particular application.

Boolean Expressions for the Segment Logic From the truth table, a standard SOP or POS expression can be written for each segment. For example, the standard SOP expression for segment *a* is

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

and the standard SOP expression for segment *e* is

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

Expressions for the other segments can be similarly developed. As you can see, the expression for segment *a* has eight product terms and the expression for segment *e* has four product terms representing each of the BCD inputs that activate that segment. This means that the standard SOP implementation of segment-*a* logic requires an AND-OR circuit consisting of eight 4-input AND gates and one 8-input OR gate. The implementation of segment-*e* logic requires four 4-input AND gates and one 4-input OR gate. In both cases, four inverters are required to produce the complement of each variable.

Karnaugh Map Minimization of the Segment Logic Let's begin by obtaining a minimum SOP expression for segment *a*. A Karnaugh map for segment *a* is shown in Figure 4–51 and the following steps are carried out:

Step 1. The 1s are mapped directly from Table 4–10.

TABLE 4–9

Active segments for each decimal digit.

DIGIT	SEGMENTS ACTIVATED
0	*a, b, c, d, e, f*
1	*b, c*
2	*a, b, d, e, g*
3	*a, b, c, d, g*
4	*b, c, f, g*
5	*a, c, d, f, g*
6	*a, c, d, e, f, g*
7	*a, b, c*
8	*a, b, c, d, e, f, g*
9	*a, b, c, d, f, g*

FIGURE 4–50

Block diagram of 7-segment logic and display.

DECIMAL DIGIT	INPUTS				SEGMENT OUTPUTS						
	D	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

Output = 1 means segment is activated (on)

Output = 0 means segment is not activated (off)

Output = X means "don't care"

Step 2. All of the "don't cares" (X) are placed on the map.

Step 3. The 1s are grouped as shown. "Don't cares" and overlapping of cells are utilized to form the largest groups possible.

Step 4. Write the minimum product term for each group and sum the terms to form the minimum SOP expression.

Keep in mind that "don't cares" do not have to be included in a group, but in this case all of them are used. Also, notice that the 1s in the corner cells are grouped with a "don't care" using the "wrap around" adjacency of the corner cells.

Standard SOP expression:

$\bar{D}\bar{C}\bar{B}\bar{A} + \bar{D}\bar{C}\bar{B}A + \bar{D}\bar{C}BA + \bar{D}C\bar{B}A + \bar{D}CB\bar{A} + \bar{D}CBA + D\bar{C}\bar{B}\bar{A} + D\bar{C}\bar{B}A$

Minimum SOP expression: $D + B + CA + \bar{C}\bar{A}$

FIGURE 4–51

Karnaugh map minimization of the segment-*a* logic expression.

FIGURE 4–52

The minimum logic implementation for segment *a* of the 7-segment display.

Minimum Implementation of Segment-*a* Logic The minimum SOP expression taken from the Karnaugh map in Figure 4–52 for the segment-*a* logic is

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

This expression can be implemented with two 2-input AND gates, one 4-input OR gate and two inverters as shown in Figure 4–52. Compare this to the standard SOP implementation for segment-*a* logic discussed earlier; you'll see that the number of gates and inverters has been reduced from thirteen to five and, as a result, the number of interconnections has been significantly reduced.

The minimum logic for each of the remaining six segments (*b*, *c*, *d*, *e*, *f*, and *g*) can be obtained with a similar approach.

VHDL Implementation (optional)

All of the segment logic can be described by VHDL for implementation in a programmable logic device. Segment-*a* logic can be described by the following VHDL program:

```
entity SEGLOGIC is
   port (A, B, C, D: in bit; SEGa: out bit);
end entity SEGLOGIC;
architecture LogicFunction of
   SEGLOGIC is
begin
   SEGa <= (A and C) or (not A
      and not C) or B or D;

end architecture LogicFunction;
```

System Assignment

- *Activity 1:* Determine the minimum logic for segment *b*.
- *Activity 2:* Determine the minimum logic for segment *c*.
- *Activity 3:* Determine the minimum logic for segment *d*.
- *Activity 4:* Determine the minimum logic for segment *e*.
- *Activity 5:* Determine the minimum logic for segment *f*.
- *Activity 6:* Determine the minimum logic for segment *g*.
- *Optional Activity:* Complete the VHDL program for all seven segments by including each segment logic description in the architecture.

SUMMARY

- Gate symbols and Boolean expressions for the outputs of an inverter and 2-input gates are shown in Figure 4–53.

FIGURE 4–53

- Commutative laws: $A + B = B + A$
  $AB = BA$
- Associative laws: $A + (B + C) = (A + B) + C$
  $A(BC) = (AB)C$
- Distributive law: $A(B + C) = AB + AC$
- Boolean rules:

1.	$A + 0 = A$	**7.**	$A \cdot A = A$
2.	$A + 1 = 1$	**8.**	$A \cdot \overline{A} = 0$
3.	$A \cdot 0 = 0$	**9.**	$\overline{\overline{A}} = A$
4.	$A \cdot 1 = A$	**10.**	$A + AB = A$
5.	$A + A = A$	**11.**	$A + \overline{A}B = A + B$
6.	$A + \overline{A} = 1$	**12.**	$(A + B)(A + C) = A + BC$

- DeMorgan's theorems:

  1. The complement of a product is equal to the sum of the complements of the terms in the product.

     $$\overline{XY} = \overline{X} + \overline{Y}$$

  2 The complement of a sum is equal to the product of the complements of the terms in the sum.

     $$\overline{X + Y} = \overline{X}\,\overline{Y}$$

- Karnaugh maps for 3 and 4 variables are shown in Figure 4–54. A 5-variable map is formed from two 4-variable maps.

FIGURE 4–54

- The basic design element in VHDL is an entity/architecture pair.

KEY TERMS

Complement The inverse or opposite of a number. In Boolean algebra, the inverse function, expressed with a bar over a variable. The complement of a 1 is 0, and vice versa.

"Don't care" A combination of input literals that cannot occur and can be used as a 1 or a 0 on a Karnaugh map for simplification.

Karnaugh map An arrangement of cells representing the combinations of literals in a Boolean expression and used for a systematic simplification of the expression.

Minimization The process that results in an SOP or POS Boolean expression that contains the fewest possible literals per term.

Product-of-sums (POS) A form of Boolean expression that is basically the ANDing of ORed terms.

Product term The Boolean product of two or more literals equivalent to an AND operation.

Sum-of-products (SOP) A form of Boolean expression that is basically the ORing of ANDed terms.

Sum term The Boolean sum of two or more literals equivalent to an OR operation.

Variable A symbol used to represent a logical quantity that can have a value of 1 or 0, usually designated by an italic letter.

VHDL A standard hardware description language. IEEE Std. 1076-1993.

EXAMPLE 4-30

Map the following standard POS expression on a Karnaugh map:

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

solution:

$$\underset{1100}{(\bar{A}+\bar{B}+C+D)}\underset{1011}{(\bar{A}+B+\bar{C}+\bar{D})}\underset{0010}{(A+B+\bar{C}+D)}\underset{1111}{(\bar{A}+\bar{B}+\bar{C}+\bar{D})}\underset{0011}{(A+B+\bar{C}+\bar{D})}$$

0000 0	0001 1	0011 3 0	0010 2 0
0100 4	0101 5	0111 7	0110 6
1100 12 0	1101 13	1111 15 0	1110 14
1000 8	1001 9	1011 11 0	1010 10

Related Prob[illegible] simplify the following standard SOP expression:

$$\bar{W}\bar{X}\bar{Y}\bar{Z}+W\bar{X}YZ+W\bar{X}\bar{Y}\bar{Z}+W\bar{X}\bar{Y}Z+\bar{W}YZ$$

به نام خدا

فصل پنجم
تحلیل منطقی ترکیبی مدارهای

5 COMBINATIONAL LOGIC ANALYSIS

CHAPTER OUTLINE

5–1 Basic Combinational Logic Circuits

5–2 Implementing Combinational Logic

5–3 The Universal Property of NAND and NOR Gates

5–4 Combinational Logic Using NAND and NOR Gates

5–5 Logic Circuit Operation with Pulse Waveform Inputs

5–6 Combinational Logic with VHDL (optional)

5–7 Troubleshooting

Digital System Application

CHAPTER OBJECTIVES

- Analyze basic combinational logic circuits, such as AND-OR, AND-OR-Invert, exclusive-OR, and exclusive-NOR
- Use AND-OR and AND-OR-Invert circuits to implement sum-of-products (SOP) and product-of-sums (POS) expressions
- Write the Boolean output expression for any combinational logic circuit
- Develop a truth table from the output expression for a combinational logic circuit
- Use the Karnaugh map to expand an output expression containing terms with missing variables into a full SOP form

- Design a combinational logic circuit for a given Boolean output expression
- Design a combinational logic circuit for a given truth table
- Simplify a combinational logic circuit to its minimum form
- Use NAND gates to implement any combinational logic function
- Use NOR gates to implement any combinational logic function
- Write VHDL programs for simple logic circuits
- Troubleshoot faulty logic circuits
- Troubleshoot logic circuits by using signal tracing and waveform analysis
- Apply combinational logic to a system application

KEY TERMS

- Universal gate
- Negative-OR
- Negative-AND
- Component
- Signal
- Node
- Signal tracing

INTRODUCTION

In Chapters 3 and 4, logic gates were discussed on an individual basis and in simple combinations. You were introduced to SOP and POS implementations, which are basic forms of combinational logic. When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is in the category of **combinational logic.** In combinational logic, the output level is at all times dependent on the combination of input levels. This chapter expands on the material introduced in earlier chapters with a coverage of the analysis, design, and troubleshooting of various combinational logic circuits. The VHDL structural approach is introduced and applied to combinational logic.

5–1 BASIC COMBINATIONAL LOGIC CIRCUITS

In Chapter 4, you learned that SOP expressions are implemented with an AND gate for each product term and one OR gate for summing all of the product terms. As you know, this SOP implementation is called AND-OR logic and is the basic form for realizing standard Boolean functions. In this section, the AND-OR and the AND-OR-Invert are examined; the exclusive-OR and exclusive-NOR gates, which are actually a form of AND-OR logic, are also covered.

AND-OR Logic

(a) Logic diagram (ANSI standard distinctive shape symbols)

(b) ANSI standard rectangular outline symbol

An AND-OR circuit directly implements an SOP expression, assuming the complements (if any) of the variables are available. The operation of the AND-OR circuit in Figure 5–1 is stated as follows:

INPUTS						OUTPUT
A	B	C	D	AB	CD	X
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

EXAMPLE 5–1

In a certain chemical-processing plant, a liquid chemical is used in a manufacturing process. The chemical is stored in three different tanks. A level sensor in each tank produces a HIGH voltage when the level of chemical in the tank drops below a specified point.

Design a circuit that monitors the chemical level in each tank and indicates when the level in any two of the tanks drops below the specified point.

AND-OR-Invert Logic

When the output of an AND-OR circuit is complemented (inverted), it results in an AND-OR-Invert circuit. Recall that AND-OR logic directly implements SOP expressions. POS expressions can be implemented with AND-OR-Invert logic. This is illustrated as follows, starting with a POS expression and developing the corresponding AND-OR-Invert expression.

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

The sensors in the chemical tanks of Example 5–1 are being replaced by a new model that produces a LOW voltage instead of a HIGH voltage when the level of the chemical in the tank drops below a critical point.

Exclusive-OR Logic

(a) Logic diagram

$$X = A\overline{B} + \overline{A}B$$

$$X = A \oplus B$$

(b) ANSI distinctive shape symbol

(c) ANSI rectangular outline symbol

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR Logic

$$X = \overline{A\overline{B} + \overline{A}B} = \overline{(A\overline{B})}\,\overline{(\overline{A}B)} = (\overline{A} + B)(A + \overline{B}) = \overline{A}\,\overline{B} + AB$$

(a) $X = \overline{A\overline{B} + \overline{A}B}$

(b) $X = \overline{A}\,\overline{B} + AB$

SECTION 5-1 REVIEW

Answers are at the end of the chapter.

1. Determine the output (1 or 0) of a 4-variable AND-OR-Invert circuit for each of the following input conditions:

   (a) $A = 1, B = 0, C = 1, D = 0$ (b) $A = 1, B = 1, C = 0, D = 1$

   (c) $A = 0, B = 1, C = 1, D = 1$

2. Determine the output (1 or 0) of an exclusive-OR gate for each of the following input conditions:

   (a) $A = 1, B = 0$ (b) $A = 1, B = 1$

   (c) $A = 0, B = 1$ (d) $A = 0, B = 0$

3. Develop the truth table for a certain 3-input logic circuit with the output expression $X = A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + ABC$.

4. Draw the logic diagram for an exclusive-NOR circuit.

5–2 IMPLEMENTING COMBINATIONAL LOGIC

In this section, examples are used to illustrate how to implement a logic circuit from a Boolean expression or a truth table. Minimization of a logic circuit using the methods covered in Chapter 4 is also included.

From a Boolean Expression to a Logic Circuit

Let's examine the following Boolean expression:

$$X = AB + CDE$$

1. One inverter to form $\overline{D}$
2. Two 2-input AND gates to form $C\overline{D}$ and EF
3. One 2-input OR gate to form $C\overline{D} + EF$
4. One 3-input AND gate to form X

$$AB(C\overline{D} + EF) = ABC\overline{D} + ABEF$$

(a)

(b) Sum-of-products implementation of the circuit in part (a)

From a Truth Table to a Logic Circuit

INPUTS A	B	C	OUTPUT X	PRODUCT TERM
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

$$X = \overline{A}BC + A\overline{B}\,\overline{C}$$

EXAMPLE 5–3

Design a logic circuit to implement the operation specified in the truth table of Table 5–4.

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

TABLE 5–4

INPUTS A	INPUTS B	INPUTS C	OUTPUT X	PRODUCT TERM
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	0	
1	0	1	1	$A\overline{B}C$
1	1	0	1	$AB\overline{C}$
1	1	1	0	

EXAMPLE 5–4

Develop a logic circuit with four input variables that will only produce a 1 output when exactly three input variables are 1s.

A	B	C	D	PRODUCT TERM
0	1	1	1	$\bar{A}BCD$
1	0	1	1	$A\bar{B}CD$
1	1	0	1	$AB\bar{C}D$
1	1	1	0	$ABC\bar{D}$

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

EXAMPLE 5–5

Reduce the combinational logic circuit in Figure 5–12 to a minimum form.

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

$$\begin{aligned} X &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}})C + \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} + D \\ &= AC + BC + CC + A + B + C + D \\ &= AC + BC + C + A + B + \cancel{C} + D \\ &= C(A + B + 1) + A + B + D \\ X &= A + B + C + D \end{aligned}$$

EXAMPLE 5–6

Minimize the combinational logic circuit in Figure 5–14. Inverters for the complemented variables are not shown.

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

Solution

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

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

(a)

(b)

SECTION 5-2 REVIEW

1. Implement the following Boolean expressions as they are stated:

   (a) $X = ABC + AB + AC$ (b) $X = AB(C + DE)$

2. Develop a logic circuit that will produce a 1 on its output only when all three inputs are 1s or when all three inputs are 0s.

3. Reduce the circuits in Question 1 to minimum SOP form.

5–3 THE UNIVERSAL PROPERTY OF NAND AND NOR GATES

Up to this point, you have studied combinational circuits implemented with AND gates, OR gates, and inverters. In this section, the universal property of the NAND gate and the NOR gate is discussed. The universality of the NAND gate means that it can be used as an inverter and that combinations of NAND gates can be used to implement the AND, OR, and NOR operations. Similarly, the NOR gate can be used to implement the inverter (NOT), AND, OR, and NAND operations.

The NAND Gate as a Universal Logic Element

(a) One NAND gate used as an inverter

(b) Two NAND gates used as an AND gate

(c) Three NAND gates used as an OR gate

The NOR Gate as a Universal Logic Element

(a) One NOR gate used as an inverter

(b) Two NOR gates used as an OR gate

(c) Three NOR gates used as an AND gate

SECTION 5–3 REVIEW

1. Use NAND gates to implement each expression:

   (a) $X = \overline{A} + B$ (b) $X = A\overline{B}$

2. Use NOR gates to implement each expression:

   (a) $X = \overline{A} + B$ (b) $X = A\overline{B}$

5–4 COMBINATIONAL LOGIC USING NAND AND NOR GATES

In this section, you will see how NAND and NOR gates can be used to implement a logic function. Recall from Chapter 3 that the NAND gate also exhibits an equivalent operation called the negative-OR and that the NOR gate exhibits an equivalent operation called the negative-AND. You will see how the use of the appropriate symbols to represent the equivalent operations makes "reading" a logic diagram easier.

NAND Logic

$$\overline{AB} = \overline{A} + \overline{B}$$

NAND → $\overline{AB}$; negative-OR → $\overline{A} + \overline{B}$

FIGURE 5–18

NAND logic for $X = AB + CD$.

$$X = \overline{(\overline{AB})(\overline{CD})}$$
$$= \overline{(\overline{A} + \overline{B})(\overline{C} + \overline{D})}$$
$$= (\overline{\overline{A} + \overline{B}}) + (\overline{\overline{C} + \overline{D}})$$
$$= \overline{\overline{A}}\,\overline{\overline{B}} + \overline{\overline{C}}\,\overline{\overline{D}}$$
$$= AB + CD$$

(c) AND-OR equivalent

(b) Equivalent NAND/Negative-OR logic diagram

(a) Several Boolean steps are required to arrive at final output expression.

(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

EXAMPLE 5–7

Redraw the logic diagram and develop the output expression for the circuit in Figure 5–21 using the appropriate dual symbols.

EXAMPLE 5–8

Implement each expression with NAND logic using appropriate dual symbols:

(a) $ABC + DE$ **(b)** $ABC + \overline{D} + \overline{E}$

(a)

(b)

NOR Logic

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

NOR negative-AND

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

(a) Final output expression is obtained after several Boolean steps.

(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

EXAMPLE 5–9

Using appropriate dual symbols, redraw the logic diagram and develop the output expression for the circuit in Figure 5–27.

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

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

SECTION 5-4 REVIEW

1. Implement the expression $X = \overline{(\overline{A} + \overline{B} + \overline{C})DE}$ by using NAND logic.
2. Implement the expression $X = \overline{\overline{A}\,\overline{B}\,\overline{C} + (D + E)}$ with NOR logic.

5–6 COMBINATIONAL LOGIC WITH VHDL (optional)

The purpose of describing logic using VHDL is so that it can be programmed into a PLD. The data flow approach to writing a VHDL program was described in Chapter 4. In this optional section, both the data flow approach using Boolean expressions and the structural approach are used to develop VHDL code for describing logic circuits. The VHDL component is introduced and used to illustrate structural descriptions. Some aspects of software development tools are discussed.

Structural Approach to VHDL Programming

The structural approach to writing a VHDL description of a logic function can be compared to installing IC devices on a circuit board and interconnecting them with wires. With the structural approach, you describe logic functions and specify how they are connected together. The VHDL **component** is a way to predefine a logic function for repeated use in a program or in other programs. The component can be used to describe anything from a simple logic gate to a complex logic function. The VHDL **signal** can be thought of as a way to specify a "wire" connection between components.

Figure 5–35 provides a simplified comparison of the structural approach to a hardware implementation on a circuit board.

(a) Hardware implementation with fixed-function logic

(b) VHDL structural implementation

FIGURE 5–35

Simplified comparison of the VHDL structural approach to a hardware implementation. The VHDL signals correspond to the interconnections on the circuit board, and the VHDL components correspond to the IC devices.

VHDL Components

A VHDL component describes predefined logic that can be stored as a package declaration in a VHDL library and called as many times as necessary in a program. You can use components to avoid repeating the same code over and over within a program. For example, you can create a VHDL component for an AND gate and then use it as many times as you wish without having to write a program for an AND gate every time you need one.

VHDL components are stored and are available for use when you write a program. This is similar to having, for example, a storage bin of ICs available when you are constructing a circuit. Every time you need to use one in your circuit, you reach into the storage bin and place it on the circuit board.

The VHDL program for any logic function can become a component and used whenever necessary in a larger program with the use of a component declaration of the following general form. **Component** is a VHDL keyword.

```
component name_of_component is
port (port definitions);
end component name_of_component;
```

For simplicity, let's assume that there are predefined VHDL data flow descriptions of a 2-input AND gate with the entity name AND_gate and a 2-input OR gate with the entity name OR_gate, as shown in Figure 5–36.

Next, assume that you are writing a program for a logic circuit that has several AND gates. Instead of rewriting the program in Figure 5–36 over and over, you can use

2-input AND gate

```
entity AND_gate is
  port (A, B: in bit; X: out bit);
end entity AND_gate;

architecture ANDfunction of AND_gate is
begin
  X <= A and B;
end architecture ANDfunction;
```

2-input OR gate

```
entity OR_gate is
  port (A, B: in bit; X: out bit);
end entity OR_gate;

architecture ORfunction of OR_gate is
begin
  X <= A or B;
end architecture ORfunction;
```

a component declaration to specify the AND gate. The port statement in the component declaration must correspond to the port statement in the entity declaration of the AND gate.

```
component AND_gate is
   port (A, B: in bit; X: out bit);
end component AND_gate;
```

Using Components in a Program To use a component in a program, you must write a component instantiation statement for each instance in which the component is used. You can think of a component instantiation as a request or call for the component to be used in the main program. For example, the simple SOP logic circuit in Figure 5–37 has two AND gates and one OR gate. Therefore, the VHDL program for this circuit will have two components and three component instantiations or calls.

Signals In VHDL, signals are analogous to wires that interconnect components on a circuit board. The signals in Figure 5–37 are named OUT1 and OUT2. Signals are the *internal* connections in the logic circuit and are treated differently than the inputs and outputs. Whereas the inputs and outputs are declared in the entity declaration using the port statement, the signals are declared within the architecture using the signal statement. **Signal** is a VHDL keyword.

The Program The program for the logic in Figure 5–37 begins with an entity declaration as follows:

```
--Program for the logic circuit in Figure 5–37
entity AND_OR_Logic is
   port (IN1, IN2, IN3, IN4: in bit; OUT3: out bit);
end entity AND_OR_Logic;
```

The architecture declaration contains the component declarations for the AND gate and the OR gate, the signal definitions, and the component instantiations.

```
architecture LogicOperation of AND_OR_Logic is
    component AND_gate is
      port (A, B: in bit); X: out bit);
    end component AND_gate;

    component OR_gate is
      port (A, B: in bit; X: out bit);
    end component OR_gate;

    signal OUT1, OUT2: bit;
begin
    G1: AND_gate port map (A => IN1, B => IN2, X => OUT1);
    G2: AND_gate port map (A => IN3, B => IN4, X => OUT2);
    G3: OR_gate port map (A => OUT1, B => OUT2, X => OUT3);
end architecture LogicOperation;
```

Component declaration for the AND gate

Component declaration for the OR gate

Signal declaration

Component instantiations

Component Instantiations Let's look at the component instantiations. First, notice that the component instantiations appear between the keyword **begin** and the **end** statement. For each instantiation an identifier is defined, such as G1, G2, and G3 in this case. Then the component name is specified. The port map essentially makes all the connections for the logic function using the operator =>. For example, the first instantiation,

```
G1: AND_gate port map (A => IN1, B => IN2, X => OUT1);
```

can be explained as follows: *Input A of AND gate G1 is connected to input IN1, input B of the gate is connected to input IN2, and the output X of the gate is connected to the signal OUT1.*

The three instantiation statements together completely describe the logic circuit in Figure 5–37, as illustrated in Figure 5–38.

Although the data flow approach using Boolean expressions wou
probably the best way to describe this particular circuit, we have us
explain the concept of the structural approach. Example 5–14 compares the structural and data flow approaches to writing a VHDL program for an SOP logic circuit.

EXAMPLE 5–14

Write a VHDL program for the SOP logic circuit in Figure 5–39 using the structural approach. Assume that VHDL components for a 3-input NAND gate and for a 2-input NAND are available. Notice the NAND gate G4 is shown as a negative-OR.

The components and component instantiations are highlighted.

```
--Program for the logic circuit in Figure 5–39
entity SOP_Logic is
   port (IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8: in bit; OUT4: out bit);
end entity SOP_Logic;

architecture LogicOperation of SOP_Logic is
-- component declaration for 3-input NAND gate
   component NAND_gate3 is
      port (A, B, C: in bit X: out bit);
   end component NAND_gate3;
-- component declaration for 2-input NAND gate
   component NAND_gate2 is
      port (A, B: in bit; X: out bit);
   end component NAND_gate;

   signal OUT1, OUT2, OUT3: bit;
begin
```

```
G1: NAND_gate3 port map (A => IN1, B => IN2, C => IN3, X => OUT1);
G2: NAND_gate3 port map (A => IN4, B => IN5, C => IN6, X => OUT2);
G3: NAND_gate2 port map (A => IN7, B => IN8, X => OUT3);
G4: NAND_gate3 port map (A => OUT1, B => OUT2, C => OUT3, X => OUT4);
end architecture LogicOperation;
```

For comparison purposes, let's write the program for the logic circuit in Figure 5–39 using the data flow approach.

```
entity SOP_Logic is
   port (IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8: in bit; OUT4: out bit);
end entity SOP_Logic;

architecture LogicOperation of SOP_Logic is
begin
   OUT4 <= (IN1 and IN2 and IN3) or (IN4 and IN5 and IN6) or (IN7 and IN8);
end architecture LogicOperation;
```

As you can see, the data flow approach results in a much simpler code for this particular logic function. However, in situations where a logic function consists of many blocks of complex logic, the structural approach might have an advantage over the data flow approach.

5–7 TROUBLESHOOTING

Page 287

Page 293

SUMMARY

- AND-OR logic produces an output expression in SOP form.
- AND-OR-Invert logic produces a complemented SOP form, which is actually a POS form.
- The operational symbol for exclusive-OR is $\oplus$. An exclusive-OR expression can be stated in two equivalent ways:

$$A\overline{B} + \overline{A}B = A \oplus B$$

- To do an analysis of a logic circuit, start with the logic circuit, and develop the Boolean output expression or the truth table or both.
- Implementation of a logic circuit is the process in which you start with the Boolean output expressions or the truth table and develop a logic circuit that produces the output function.
- All NAND or NOR logic diagrams should be drawn using appropriate dual symbols so that bubble outputs are connected to bubble inputs and nonbubble outputs are connected to nonbubble inputs.
- When two negation indicators (bubbles) are connected, they effectively cancel each other.
- A VHDL component is a predefined logic function stored for use throughout a program or in other programs.
- A component instantiation is used to call for a component in a program.
- A VHDL signal effectively acts as an internal interconnection in a VHDL structural description.

KEY TERMS

Component A VHDL feature that can be used to predefine a logic function for multiple use throughout a program or programs.

Negative-AND The dual operation of a NOR gate when the inputs are active-LOW.

Negative-OR The dual operation of a NAND gate when the inputs are active-LOW.

Node A common connection point in a circuit in which a gate output is connected to one or more gate inputs.

Signal A waveform; a type of VHDL object that holds data.

Signal tracing A troubleshooting technique in which waveforms are observed in a step-by-step manner beginning at the input and working toward the output or vice versa. At each point the observed waveform is compared with the correct signal for that point.

Universal gate Either a NAND gate or a NOR gate. The term *universal* refers to the property of a gate that permits any logic function to be implemented by that gate or by a combination of gates of that kind.

6 FUNCTIONS OF COMBINATIONAL LOGIC

CHAPTER OUTLINE

1. Basic Adders
2. Parallel Binary Adders
3. Ripple Carry versus Look-Ahead Carry Adders
4. Comparators
5. Decoders
6. Encoders
7. Code Converters
8. Multiplexers (Data Selectors)
9. Demultiplexers
10. Parity Generators/Checkers
11. Troubleshooting
12. Digital System Application

Chapter Objectives

- Distinguish between half-adders and full-adders
- . Use full-adders to implement multibit parallel binary adders
- Explain the differences between ripple carry and look-ahead
- carry parallel adders
- . Use the magnitude comparator to determine the relationship
- between two binary numbers and use cascaded comparators to
- handle the comparison of larger numbers
- Implement a basic binary decoder
- Use BCD-to-7 -segment decoders in display systems
- Apply a decimal-to-BCD priority encoder in a simple keyboard
- application
- Convert from binary to Gray code, and Gray code to binary by
- using logic devices
- Apply multiplexers in data selection, multiplexed displays, logic

6-1 BASIC ADDERS

Adders are important in computers and also in other types of digital systems in which numerical data are processed. An understanding of the basic adder operation is fundamental to the study of digital systems. In this section, the half-adder and the full adder are introduced.

HALF ADDER

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

$$C_{out} = AB$$

$$\Sigma = A \oplus B$$

FULL ADDER

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C_{in} = input carry, sometimes designated as CI

C_{out} = output carry, sometimes designated as CO

Σ = sum

A and B = input variables (operands)

$$\Sigma = (A \oplus B) \oplus C_{in}$$

$$C_{out} = AB + (A \oplus B)C_{in}$$

FULL ADDER

(a) Arrangement of two half-adders to form a full-adder

EXAMPLE 6-1

For each of the three full-adders in Figure 6-6, determine the outputs for the inputs shown.

SECTION 6-1 REVIEW

1. Determine the sum (Σ) and the output carry (C_{out} of a half-adder for each set of input bits:

(a) 01 (b) 00 (c) 10 (d) 11

2. A full-adder has $C_{in} = 1$. What are the sum (Σ) and the output carry C_{out} when A = 1 and B = 1?

6-2 PARALLEL BINARY ADDERS

For n bits addition we need n-1 full adders and 1 half adder but in IC package there are n full adder?

EXAMPLE 6-2

Determine the sum generated by the 3-bit parallel adder in Figure 6-8 and show the intermediate carries when the binary numbers 101 and 011 are being added.

Four-Bit Parallel Adders

A group of four bits is called a **nibble**. A basic 4-bit parallel adder is implemented with four full-adder stages. The Carry output of each adder is connected to the carry input of the next higher-order adder as indicated. These are called **internal carries**.

(a) Block diagram

Four-Bit Parallel Adders

In terms of the method used to handle carries in a parallel adder, there are two **k-ahead** adder.

EXAMPLE 6-3

Use the 4-bit parallel adder truth table (Table 6-3) to find the sum and output carry for the addition of the following two 4-bit numbers if the input carry (C_{n-1}) is 0:

$$A_4A_34_2A_1 = 1100 \text{ and } B_4B_3B_2B_1 = 1100$$

THE 74LS283 4-BIT PARALLEL ADDER

This device may be available in other TTL or CMOS families. Check the Texas Instruments website at www.ti.com

THE 74LS283 4-BIT PARALLEL ADDER

IC Data Sheet Characteristics Recall that logic gates have one specified propagation delay time, t_p , from an input to the output. For IC logic, there may be several different specifications for t_p. The 4-bit parallel adder has the four t_p specifications shown in Figure 6-11. which is part of a 74LS283 data sheet.

Symbol	Parameter	Limits			Unit
		Min	Typ	Max	
t_{PLH} t_{PHL}	Propagation delay. C_0 input to any Σ output		16 15	24 24	ns
t_{PLH} t_{PHL}	Propagation delay. any A or B input to Σ outputs		15 15	24 24	ns
t_{PLH} t_{PHL}	Propagation delay. C_0 input to C_4 output		11 11	17 22	ns
t_{PLH} t_{PHL}	Propagation delay. any A or B input to C_4 output		11 12	17 17	ns

ADDER EXPANSION

(a) Cascading of two 4-bit adders to form an 8-bit adder

process is known as **cascading.**

EXAMPLE 6-4

Show how two 74LS283 adders can be connected to form an 8-bit parallel adder?

Show output bits for the following 8-bit input numbers:

$A_8A_7A_6A_5A_4A_3A_2A_1 = 10111001$ and $B_8B_7B_6B_5B_4B_3B_2B_1 = 10011110$

An Application

A simple voting system that can be used to simultaneously provide the number of "yes" votes and the number of "no" votes for immediately determining opinions.

SECTION 6-2 REVIEW QUESTIONS

1. Two 4-bit numbers (1101 and 1011) are applied to a 4-bit parallel adder. The input carry is 1. Determine the sum Σ and the output carry.

1. 2. How many 74LS283 adders would be required to add two binary numbers each representing decimal numbers up through 1000_{10}?

6-3 RIPPLE CARRY Vs. LOOK-AHEAD CARRY ADDERS

A ripple carry adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage.

Carry propagation delay: The time from the application of the input carry until the output carry occurs, assuming other inputs are already present.

The look-Ahead Carry Adder

The speed with which an addition can be performed is limited by the time required for the carries to propagate, or ripple, through all the stages of a parallel adder. One method of speeding up the addition process by eliminating this ripple carry delay is called **look-ahead carry** addition.

$$C_{out} = AB + (A \oplus B)C_{in}$$

$$C_1 = A_0 B_0 + (A_0 \oplus B_0).C_0 = f_1 (A_0, B_0, C_0)$$

$$C_2 = A_1 B_1 + (A_1 \oplus B_1).C_1 = f_2 (A_0, B_0, A_1, B_1, C_0)$$

$$C_3 = A_2 B_2 + (A_2 \oplus B_2).C_2 = f_2 (A_0, B_0, A_1, B_1, A_2, B_2, C_0)$$

$$C_4 = A_3 B_3 + (A_3 \oplus B_3).C_3 = f_2 (A_0, B_0, A_1, B_1, A_2, B_2, A_3, B_3, C_0)$$

The look-Ahead Carry Adder

و این مدار حداکثر در دو سطح قابل ساخت است و نیازی نیست تا پاسخ هر طبقه منتظر رقم نقلی طبقه قبل بماند

FIGURE 6–18

Logic diagram for a 4-stage look-ahead carry adder.

Combination look-Ahead and Ripple Carry Adders in cascaded adders

6-4 COMPARATORS

The basic function of a comparator is to compare the magnitudes of two binary quantities to determine the relationship of those quantities. In its simplest form, a comparator circuit determines whether two numbers are equal.

Equality

Equality

Can be expanded to any number of bits

EXAMPLE 6-5

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6-21, and determine the output by following the logic levels through the circuit.

(a) 10 and 10 (b) 11 and 10

Related Problem Repeat the process for binary inputs of 01 and 10.

INEQULITY

1. If A3 = I and B3 = 0, number A is greater than number B.

2. If A3 = 0 and B3 =1 number A is less than number B.

3. If A3 = B 3 , then you must examine the next lower bit position for an inequality.

EXAMPLE 6-6

Determine the A = B, A > B, and A < B outputs for the input numbers shown on the comparator.

Related Problem What are the comparator outputs when $A_3 4_2 A_1 A_0$= 1001 and $B_3B_2B_1B_0$ = LOW?

THE 74HC85 4-BIT MAGNITUDE COMPARATOR

The 74HC85 is a comparator, also available in other IC families. This device has all the inputs and outputs of the generalized comparator previously discussed and, in addition, has three cascading inputs: A<B, A = B, A>B.

EXAMPLE 6-7

Use 74HC85 comparators to compare the magnitudes of two 8-bit numbers. Show the comparators with proper interconnections.

Related Problem Expand the circuit to a 16-bit comparator.

SECTION 6-4 REVIEW QUESTIONS

1. The binary numbers A = 1011 and B = 1010 are applied to the inputs of a 74HC85. Determine the outputs.

2. The binary numbers A = 11001011 and B = 11010100 are applied to the 8-bit comparator in previous example. Determine the states of output pins 5, 6, and 7 on each 74HC85.

COMPUTER NOTE

Addition is performed by computers on two numbers at a time, called operands. The source operand is a number that is to be added to an existing number called the destination operand, which is held in an ALU register, such as the accumulator. The sum of the two numbers is then stored back in the accumulator. Addition is performed on integer numbers or floating-point numbers using ADD or FADD instructions respectively.

COMPUTER NOTE

In a computer, the cache is a very fast intermediate memory between the central processing unit (CPU) and the slower main memory. The CPU requests data by sending out its address (unique location) in memory. Part of this address is called a tag. The tag address comparator compares the tag from the CPU with the tag from the cache directory. If the two agree, the addressed data is already in the cache and is retrieved very quickly. If the tags disagree, the data must be retrieved from the main memory at a much slower rate.

HANDS ON TIPS

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 Vcc . Also, remember that unused inputs must always be connected to an appropriate logic level (ground or V_{cc}. Unused outputs may be left open.

6-5 DECODER

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.

The Basic Binary Decoder

Suppose you need to determine when a binary 1001 occurs on the inputs of a digital circuit.

In the representation of a binnry number or other weighted code in this book, the LSB is the right-most bit in a horizontal arrangement and the top most bit in a vertical arrangement, unless specified otherwise.

If a NAND gate is used, a LOW output will indicate the presence of the proper binary code.

The 4-Bit Decoder

In order to decode all possible combinations of four bits, 16 decoding gates are required ($2^4 = 16$). This type of decoder is commonly called either a 4-line-to-16-line decoder or a 1-of-16 decoder

DECIMAL DIGIT	BINARY INPUTS A_3	A_2	A_1	A_0	DECODING FUNCTION	OUTPUTS 0	1	2	3	4	5	6	7	15
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$	0	1	1	1	1	1	1	1	1
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$	1	0	1	1	1	1	1	1	1
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$	1	1	0	1	1	1	1	1	1
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	0	1	1	1	1
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$	1	1	1	1	1	0	1	1	1
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	0	1	1
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1
10	1	0	1	0	$A_3\bar{A}_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1
11	1	0	1	1	$A_3\bar{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1
12	1	1	0	0	$A_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	1
13	1	1	0	1	$A_3A_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1
14	1	1	1	0	$A_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	0

به ازای هر خروجی یک گیت NAND وجود دارد که تنها به ازای آن کد ورودی خاص صفر خواهد شد. در صورتی که خروجیها فعال با 1 باشند به جای گیتهای NAND گیت AND بکار خواهد رفت.

دکودر باینری دارای N ورودی و 2^N خروجی می باشد.

EXAMPLE 6-8

Determine the logic required to decode the binary number 1011 by producing a HIGH level on the output.

Related Problem Develop the logic required to detect the binary code 10010 and produce an active LOW output.

THE 74HC154 1-0F-16 DECODER

In order to decode all possible combinations of four bits, 16 decoding gates are required ($2^4 = 16$). This type of decoder is commonly called either a 4-line-to-16-line decoder or a 1-of-16 decoder

(a) Pin diagram (b) Logic symbol

COMPUTER NOTE

An instruction tells the computer what operation to perform. Instructions are in machine code (1's and 0's) and, in order for the computer to carry out an instruction, the instruction must be decoded. Instruction decoding is one of the steps in instruction pipelining, which are as follows:

Instruction is read from the memory (instruction fetch), instruction is decoded, operand(s) is (are) read from memory (operand fetch), instruction is executed, and result is written back to memory. Basically, pipelining allows the next instruction to begin processing before the current one is completed.

EXAMPLE 6-9

A certain application requires that a 5-bit number be decoded. Use 74HCl54 decoders to implement the logic. The binary number is represented by the format $A_4A_34_2A_1A_0$.

Related Problem Simplify the Boolean expression

An Application

Decoders are used in many types of applications. One example is in computers for input/output selection. Each I/O port has a number, called an address, which uniquely identifies it. When the computer wants to communicate with a particular device, it issues the appropriate address code for the I/O port to which that particular device is connected. This binary port address is decoded and the appropriate decoder output is activated to enable the I/O port.

The BCD-to-Decimal Decoder

4-line-to-10- line decoder or a 1-of- 10 decoder. Implemented with ten NAND gates.

DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0	DECODING FUNCTION
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$

EXAMPLE 6-10

The 74HC42 is an integrated circuit BCD-to-decimal decoder. show the output waveforms.

Related Problem Sim

The BCD-to-7-Segment Decoder

LED Display

LCD Displays liquid crystal display. L operate by polarizing so that a nonactiv segment reflects inc light and thus ap invisible against background. An activ segment does not re incident light and appears dark. L consume much less power than LEDs but cannot be seen in the dark, while LEDs can.

The BCD-to-7-Segment Decoder

حتما digital system application فصل 4 مطالعه شود به عنوان یک مثال کامل از مراحل طراحی مدار داخلی این دکودر مرور شده و ممکن است مسئله مشابهی در امتحان میان ترم ارائه شود.

THE 74LS47 BCD-TO-7-SEGMENT DECODER/DRIVER

Lamp Test When a LOW is applied to the $\overline{LT}$ input and the $\overline{BI}/\overline{RBO}$ is HIGH, all of the 7 segments in the display are turned on.

$\overline{RBI}$ (ripple blanking input), and $\overline{BI}/\overline{RBO}$ (blanking input/ripple blanking output) functions.

The outputs can drive a common-anode 7-segment display directly.

THE 74LS47 BCD-TO-7-SEGMENT DECODER/DRIVER

Zero suppression is a feature used for multidigit displays to blank out unnecessary zeros.

Blanking the zeros at the front of a number is called leading zero suppression and blanking the zeros at the back of the number is called trailing zero suppression

Zero suppression in the 74LS47 is accomplished using the RBI and BI/RBO functions.

BI is the blanking input that shares the same pin with RBO; in other words, the BI/RBO pin can be used as an input or an output. When used as a BI (blanking input), all segment outputs are HIGH (nonactive) when BI is LOW, which overrides all other inputs. The BI function is not part of the zero suppression capability of the device.

All of the segment outputs of the decoder are nonactive (HIGH) if a zero code (0000) is on its BCD in puts and if its RBI is LOW. This causes the display to be blank and produces a LOW RBO.

The logic diagram in Figure 6-36ta) illustrates leading zero suppression for a whole number. The highest-order digit position (left-most) is always blanked if a zero code is on

THE 74LS47 BCD-TO-7-SEGMENT DECODER/DRIVER

(b) Illustration of trailing zero suppression

SECTION 6-5 REVIEW QUESTIONS

1. A 3-line-to-8-line decoder can be used for octal-to-decimal decoding. When a binary 101 is on the inputs, which output line is activated?
2. How many 74HC154 1-of-16 decoders are necessary to decode a 6-bit binary number?
3. Would you select a decoder/driver with active-HIGH or active-LOW outputs to drive a common-cathode 7-segment LED display?

6-6 ENCODERS

An **encoder** is a combinational logic circuit that essentially performs a "reverse" decoder function. An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary. Encoders can also be devised to encode various symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called **encoding.**

The Decimal to BCD Encoder

10-line-to-4-line encoder.

DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

$$A_3 = 8 + 9$$

$$A_2 = 4 + 5 + 6 + 7$$

$$A_1 = 2 + 3 + 6 + 7$$

$$A_0 = I + 3 + 5 + 7 + 9$$

مشکل این انکودر چیست؟

THE 74HC147 DECIMAL-TO-BCD ENCODER

Priority encoder with active-LOW inputs and outputs
Zero output is represented when none of the inputs is active.

THE 74LS148 8-LINE-TO-3-LINE ENCODER

- Priority encoder
- EI (enable input) must be LOW.
- It also has the EO (enable output) and GS output for expansion purposes.
- The EO is LOW when the EI is LOW and none of the inputs (0 through 7) is active.
- GS is LOW when EI is LOW and any of the inputs is active.

- The 74LS148 can be expanded to a 16-1ine-to-4-line encoder by connecting the EO of the higher-order encoder to the EI of the lower-order encoder and negative-ORing the corresponding binary outputs.
- The EO is used as the fourth and most significant bit. This particular configuration produces active-HIGH outputs for the 4-bit binary number.

An Application

FIGURE 6–42

A simplified keyboard encoder.

SECTION 6-6 REVIEW

1. Suppose the HIGH levels are applied to the 2 input and the 9 input of the circuit in Figure 6–38.
   (a) What are the states of the output lines?
   (b) Does this represent a valid BCD code?
   (c) What is the restriction on the encoder logic in Figure 6–38?
2. (a) What is the $\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$ output when LOWs are applied to pins 1 and 5 of the 74HC147 in Figure 6–39?
   (b) What does this output represent?

6–7 CODE CONVERTERS

BCD-to-Binary Conversion

One method of BCD-to-binary code conversion uses adder circuits. The basic conversion process is as follows:

1. The value, or weight, of each bit in the BCD number is represented by a binary number.
2. All of the binary representations of the weights of bits that are 1s in the BCD number are added.
3. The result of this addition is the binary equivalent of the BCD number.

A more concise statement of this operation is

The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.

Let's examine an 8-bit BCD code (one that represents a 2-digit decimal number) to understand the relationship between BCD and binary. For instance, you already know that the decimal number 87 can be expressed in BCD as

$$\underbrace{1000}_{8}\quad\underbrace{0111}_{7}$$

The left-most 4-bit group represents 80, and the right-most 4-bit group represents 7. That is, the left-most group has a weight of 10, and the right-most group has a weight of 1. Within each group, the binary weight of each bit is as follows:

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

The binary equivalent of each BCD bit is a binary number representing the weight of that bit within the total BCD number. This representation is given in Table 6–7.

BCD BIT	BCD WEIGHT	(MSB) BINARY REPRESENTATION 64	32	16	8	4	2	(LSB) 1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

If the binary representations for the weights of all the 1s in the BCD number are added, the result is the binary number that corresponds to the BCD number. Example 6–12 illustrates this.

EXAMPLE 6–12

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Solution

Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.

80	40	20	10	8	4	2	1		
0	0	1	0	0	1	1	1		
								0000001	1
								0000010	2
								0000100	4
								+ 0010100	20
								0011011	Binary number for decimal 27

80	40	20	10	8	4	2	1		
1	0	0	1	1	0	0	0		
								0001000	8
								0001010	10
								+ 1010000	80
								1100010	Binary number for decimal 98

With this basic procedure in mind, let's see how the process can be implemented with logic circuits. Once the binary representation for each 1 in the BCD number is determined, adder circuits can be used to add the 1s in each column of the binary representation. The 1s occur in a given column only when the corresponding BCD bit is a 1. The occurrence of a BCD 1 can therefore be used to generate the proper binary 1 in the appropriate column of the adder structure. To handle a two-decimal-digit (two-decade) BCD code, eight BCD input lines and seven binary outputs are required. (It takes seven bits to represent binary numbers through ninety-nine.)

Binary-to-Gray and Gray-to-Binary Conversion

The basic process for Gray-binary conversions was covered in Chapter 2. Exclusive-OR gates can be used for these conversions. Programmable logic devices (PLDs) can also be programmed for these code conversions. Figure 6–43 shows a 4-bit binary-to-Gray code converter, and Figure 6–44 illustrates a 4-bit Gray-to-binary converter.

SECTION 6-7 REVIEW

1. Convert the BCD number 10000101 to binary.
2. Draw the logic diagram for converting an 8-bit binary number to Gray code.

MUX DMUX

Multiplexer
Demultiplexer
A
B
C
D
E
F
Data from A to D
Data from B to E
Data from C to F
Data from A to D
Δt_1
Δt_2
Δt_3
Δt_1
Δt_2
Δt_3
Δt_1
Δt_2
Δt_3
Switching sequence control input
Switching sequence control input

6–8 MULTIPLEXERS (DATA SELECTORS)

DATA-SELECT INPUTS S_1	S_0	INPUT SELECTED
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.

The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1\overline{S}_1S_0$.

The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2S_1\overline{S}_0$.

The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3S_1S_0$.

$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

کاربردهای MUX

انتخاب اطلاعات

مبدل اطلاعات موازی به سریال

ساخت توابع ترکیبی

EXAMPLE 6–14

The data-input and data-select waveforms in Figure 6–48(a) are applied to the multiplexer in Figure 6–47. Determine the output waveform in relation to the inputs.

THE 74HC157 QUAD 2-INPUT DATA SELECTOR/MULTIPLEXER

(a) Pin diagram

(b) Logic symbol

THE 74LS151 8-INPUT DATA SELECTOR/MULTIPLEXER

(a) Pin diagram

(b) Logic symbol

EXAMPLE 6–15

Use 74LS151s and any other logic necessary to multiplex 16 data lines onto a single data-output line.

Solution

Applications

A 7-Segment Display Multiplexer

EXAMPLE 6–16

Implement the logic function specified in Table 6–9 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

INPUTS			OUTPUT
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$Y = \overline{A}_2\overline{A}_1A_0 + \overline{A}_2A_1A_0 + A_2\overline{A}_1A_0 + A_2A_1\overline{A}_0$$

EXAMPLE 6–17

Implement the logic function in Table 6–10 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

DECIMAL DIGIT	INPUTS A_3	A_2	A_1	A_0	OUTPUT Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

	D0	D1	D2	D3	D4	D5	D6	D7
$\overline{A3}$	0	1	2	3	4	5	6	7
A3	8	9	10	11	12	13	14	15
	A3	A3	VCC	GND	A3	VCC	A3	VCC

جدولی مشابه جدول فوق تشکیل می دهیم

سه بیت کم ارزشتر ورودی را به ترتیب باینری به ورودیهای انتخاب وصل می کنیم

شماره سطرهایی که در جدول ارزش دارای خروجی 1 هستند را مشخص می کنیم.

پایه I مربوط به هر ستون را به ترتیب به سیگنالهای زیر وصل می کنیم

الف) اگر هر دو سطر 0 بودند به GND یا سطح صفر منطقی

ب) اگر هر دو 1 بودند به Vcc یا سطح 1 منطقی

ج) اگر در سطر اول 1 بود به مکمل A3 و اگر در سطر دوم 1 بود به خود A3

	D0	D1	D2	D3	D4	D5	D6	D7
$\overline{A3}$	0	1	2	3	4	5	6	7
A3	8	9	10	11	12	13	14	15
	A3	$\overline{A3}$	VCC	GND	A3	VCC	$\overline{A3}$	VCC

SECTION 6–8 REVIEW

1. In Figure 6–47, $D_0 = 1$, $D_1 = 0$, $D_2 = 1$, $D_3 = 0$, $S_0 = 1$, and $S_1 = 0$. What is the output?
2. Identify each device.

   (a) 74LS157 (b) 74LS151
3. A 74LS151 has alternating LOW and HIGH levels on its data inputs beginning with $D_0 = 0$. The data-select lines are sequenced through a binary count (000, 001, 010, and so on) at a frequency of 1 kHz. The enable input is LOW. Describe the data output waveform.
4. Briefly describe the purpose of each of the following devices in Figure 6–52:

   (a) 74LS157 (b) 74LS47 (c) 74LS139

6–9 DEMULTIPLEXERS

A **demultiplexer (DEMUX)** basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. As you will learn, decoders can also be used as demultiplexers.

FIGURE 6–55

A 1-line-to-4-line demultiplexer.

EXAMPLE 6–18

The serial data-input waveform (Data in) and data-select inputs (S_0 and S_1) are shown in Figure 6–56. Determine the data-output waveforms on D_0 through D_3 for the demultiplexer in Figure 6–55.

THE 74HC154 DEMULTIPLEXER

FIGURE 6–57

The 74HC154 decoder used as a demultiplexer.

SECTION 6–9 REVIEW

1. Generally, how can an decoder be used as a demultiplexer?
2. The 74HC154 demultiplexer in Figure 6–57 has a binary code of 1010 on the data-select lines, and the data-input line is LOW. What are the states of the output lines?

6–10 PARITY GENERATORS/CHECKERS

Basic Parity Logic

The sum (disregarding carries) of an even number of 1s is always 0, and the sum of an odd number of 1s is always 1.

(a) Summing of two bits (b) Summing of four bits

THE 74LS280 9-BIT PARITY GENERATOR/CHECKER

(a) Traditional logic symbol

Number of Inputs A–I That Are HIGH	Outputs Σ Even	Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

(b) Function table

Parity Checker When this device is used as an even parity checker, the number of input bits should always be even; and when a parity error occurs, the Σ Even output goes LOW and the Σ Odd output goes HIGH. When it is used as an odd parity checker, the number of input bits should always bé odd; and when a parity error occurs, the Σ Odd output goes LOW and the Σ Even output goes HIGH.

Parity Generator If this device is used as an even parity generator. the parity bit is taken at the Σ Odd output because this output is a 0 if there is an even number of input bits and it is a 1 if there is an odd number. When used as an odd parity generator, the parity bit is taken at the Σ Even output because it is a 0 when the number of inputs bits is odd.

FIGURE 6–60

Simplified data transmission system with error detection.

*Storage devices are introduced in Chapter 9 and used in other later chapters.

0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
S_0
S_1
S_2
Bit received incorrectly (0 was transmitted)
Data stream at DEMUX input
D_0 D_1 D_2 D_3 D_4 D_5 D_6 P D_0 D_1 D_2 D_3 D_4 D_5 D_6 P
Error

COMPUTER NOTE

The Pentium microprocessor performs internal parity checks as well as parity checks of the external data and address buses. In a read operation, the external system can transfer the parity information together with the data bytes. The Pentium checks whether the resulting parity is even and sends out the corresponding signal. When it sends out an address code, the Pentium does not perform an address parity check, but it does generate an even parity bit for the address.

SECTION 6–10 REVIEW

1. Add an even parity bit to each of the following codes:
   (a) 110100 (b) 01100011
2. Add an odd parity bit to each of the following codes:
   (a) 1010101 (b) 1000001
3. Check each of the even parity codes for an error.
   (a) 100010101 (b) 1110111001

6–11 TROUBLESHOOTING

DIGITAL SYSTEM APPLICATION

SUMMARY

- Half-adder and full-adder operations are summarized in Figure 6–74.
- Logic symbols with pin numbers for the ICs used in this chapter are shown in Figure 6–75. Pin designations may differ from some manufacturers' data sheets.
- Standard logic functions from the 74XX series are available for use in a programmable logic design.

Half-adder

Inputs A	Inputs B	Carry Out C_{out}	Sum Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full-adder

Inputs A	Inputs B	Carry In C_{in}	Carry Out C_{out}	Sum Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

DIGITAL SYSTEM APPLICATION

Seven-segment displays are used in many types of products. The tablet-counting and control system that was described in Chapter 1 has two 7-segment displays. These displays are used with logic circuits that decode a binary coded decimal (BCD) number and activate the appropriate digits on the display. In this digital system application, we focus on a minimum-gate design for this to illustrate an application of Boolean expressions and the Karnaugh map. As an option, VHDL is also applied.

The 7-Segment Display

Figure 4–47 shows a common display format composed of seven elements or segments. Energizing certain combinations of these segments can cause each of the ten decimal digits to be displayed. Figure 4–48 illustrates this method of digital display for each of the ten digits by using a red segment to represent one that is energized. To produce a 1, segments *b* and *c* are energized; to produce a 2, segments *a*, *b*, *g*, *e*, and *d* are used; and so on.

LED Displays One common type of 7-segment display consists of light-emitting diodes (**LEDs**) arranged as shown in Figure 4–49. Each segment is an LED that emits light when there is current through it. In Figure 4–49(a) the common-anode arrangement requires the driving circuit to provide a low-level voltage in order to activate a given segment. When a LOW is applied to a segment input, the LED is turned on, and there is current through it. In Figure 4–49(b) the common-cathode arrangement requires the driver to provide a high-level voltage to activate a segment. When a HIGH is applied to a segment input, the LED is turned on and there is current through it.

LCD Displays Another common type of 7-segment display is the liquid crystal display (**LCD**). LCDs operate by polarizing light so that a nonactivated segment reflects incident light and thus appears invisible against its background. An activated segment does not reflect incident light and thus appears dark. LCDs consume much less power than LEDs but

FIGURE 4–47

Seven-segment display format showing arrangement of segments.

FIGURE 4–48

Display of decimal digits with a 7-segment device.

FIGURE 4–49

Arrangements of 7-segment LED displays.

cannot be seen in the dark, while LEDs can.

Segment Logic

Each segment is used for various decimal digits, but no one segment is used for all ten digits. Therefore, each segment must be activated by its own decoding circuit that detects the occurrence of any of the numbers in which the segment is used. From Figures 4–47 and 4–48, the segments that are required to be activated for each displayed digit are determined and listed in Table 4–9.

Truth Table for the Segment Logic

The segment decoding logic requires four binary coded decimal (BCD) inputs and seven outputs, one for each segment in the display, as indicated in the block diagram of Figure 4–50. The multiple-output truth table, shown in Table 4–10, is actually seven truth tables in one and could be separated into a separate table for each segment. A 1 in the segment output columns of the table indicates an activated segment.

Since the BCD code does not include the binary values 1010, 1011, 1100, 1101, 1110, and 1111, these combinations will never appear on the inputs and can therefore be treated as "don't care" (X) conditions, as indicated in the truth table. To conform with the practice of most IC manufacturers, *A* represents the least significant bit and *D* represents the most significant bit in this particular application.

Boolean Expressions for the Segment Logic From the truth table, a standard SOP or POS expression can be written for each segment. For example, the standard SOP expression for segment *a* is

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

and the standard SOP expression for segment *e* is

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

Expressions for the other segments can be similarly developed. As you can see, the expression for segment *a* has eight product terms and the expression for segment *e* has four product terms representing each of the BCD inputs that activate that segment. This means that the standard SOP implementation of segment-*a* logic requires an AND-OR circuit consisting of eight 4-input AND gates and one 8-input OR gate. The implementation of segment-*e* logic requires four 4-input AND gates and one 4-input OR gate. In both cases, four inverters are required to produce the complement of each variable.

Karnaugh Map Minimization of the Segment Logic Let's begin by obtaining a minimum SOP expression for segment *a*. A Karnaugh map for segment *a* is shown in Figure 4–51 and the following steps are carried out:

Step 1. The 1s are mapped directly from Table 4–10.

TABLE 4–9

Active segments for each decimal digit.

DIGIT	SEGMENTS ACTIVATED
0	*a, b, c, d, e, f*
1	*b, c*
2	*a, b, d, e, g*
3	*a, b, c, d, g*
4	*b, c, f, g*
5	*a, c, d, f, g*
6	*a, c, d, e, f, g*
7	*a, b, c*
8	*a, b, c, d, e, f, g*
9	*a, b, c, d, f, g*

FIGURE 4–50

Block diagram of 7-segment logic and display.

DECIMAL DIGIT	INPUTS				SEGMENT OUTPUTS						
	D	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

Output = 1 means segment is activated (on)

Output = 0 means segment is not activated (off)

Output = X means "don't care"

Step 2. All of the "don't cares" (X) are placed on the map.

Step 3. The 1s are grouped as shown. "Don't cares" and overlapping of cells are utilized to form the largest groups possible.

Step 4. Write the minimum product term for each group and sum the terms to form the minimum SOP expression.

Keep in mind that "don't cares" do not have to be included in a group, but in this case all of them are used. Also, notice that the 1s in the corner cells are grouped with a "don't care" using the "wrap around" adjacency of the corner cells.

Standard SOP expression:

$\bar{D}\bar{C}\bar{B}\bar{A} + \bar{D}\bar{C}\bar{B}A + \bar{D}\bar{C}BA + \bar{D}C\bar{B}A + \bar{D}CB\bar{A} + \bar{D}CBA + D\bar{C}\bar{B}\bar{A} + D\bar{C}\bar{B}A$

Minimum SOP expression: $D + B + CA + \bar{C}\bar{A}$

FIGURE 4–51

Karnaugh map minimization of the segment-*a* logic expression.

FIGURE 4–52

The minimum logic implementation for segment *a* of the 7-segment display.

Minimum Implementation of Segment-*a* Logic The minimum SOP expression taken from the Karnaugh map in Figure 4–52 for the segment-*a* logic is

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

This expression can be implemented with two 2-input AND gates, one 4-input OR gate and two inverters as shown in Figure 4–52. Compare this to the standard SOP implementation for segment-*a* logic discussed earlier; you'll see that the number of gates and inverters has been reduced from thirteen to five and, as a result, the number of interconnections has been significantly reduced.

The minimum logic for each of the remaining six segments (*b*, *c*, *d*, *e*, *f*, and *g*) can be obtained with a similar approach.

VHDL Implementation (optional)

All of the segment logic can be described by VHDL for implementation in a programmable logic device. Segment-*a* logic can be described by the following VHDL program:

```
entity SEGLOGIC is
   port (A, B, C, D: in bit; SEGa: out bit);
end entity SEGLOGIC;
architecture LogicFunction of
   SEGLOGIC is
begin
   SEGa <= (A and C) or (not A
      and not C) or B or D;

end architecture LogicFunction;
```

System Assignment

- *Activity 1:* Determine the minimum logic for segment *b*.
- *Activity 2:* Determine the minimum logic for segment *c*.
- *Activity 3:* Determine the minimum logic for segment *d*.
- *Activity 4:* Determine the minimum logic for segment *e*.
- *Activity 5:* Determine the minimum logic for segment *f*.
- *Activity 6:* Determine the minimum logic for segment *g*.
- *Optional Activity:* Complete the VHDL program for all seven segments by including each segment logic description in the architecture.