GLOBAL EDITION

Logic and Computer Design Fundamentals

FIFTH EDITION

Morris Mano • Charles R. Kime • Tom Martin

ALWAYS LEARNING

PEARSON

Logic and Computer Design Fundamentals

Fifth Edition
Global Edition

M. Morris Mano
California State University, Los Angeles

Charles R. Kime
University of Wisconsin, Madison

Tom Martin
Virginia Tech

PEARSON

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

Vice President and Editorial Director, ECS: *Marcia J. Horton*
Executive Editor: *Tracy Johnson*
Acquisitions Editor: *Julie Bai*
Assistant Acquisitions Editor, Global Editions: *Aditee Agarwal*
Executive Marketing Manager: *Tim Galligan*
Marketing Assistant: *Jon Bryant*
Senior Managing Editor: *Scott Disanno*
Production Project Manager: *Greg Dulles*
Project Editor, Global Editions: *Donald Villamero*
Program Manager: *Joanne Manning*
Global HE Director of Vendor Sourcing and Procurement: *Diane Hynes*
Director of Operations: *Nick Sklitsis*
Operations Specialist: *Maura Zaldivar-Garcia*
Senior Production Manufacturing Controller, Global Editions: *Trudy Kimber*
Media Production Manager, Global Editions: *Vikram Kumar*
Cover Art: © *Shaparniy/Shutterstock*
Manager, Rights and Permissions: *Rachel Youdelman*
Associate Project Manager, Rights and Permissions: *Timothy Nicholls*
Composition: *Jouve India*

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsonglobaleditions.com

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

10 9 8 7 6 5 4 3 2 1

ISBN 10: 1-292-09607-1
ISBN 13: 978-1-292-09607-0

Typeset by Jouve, in 10/12 Times Ten LT Std
Printed and bound in Great Britain by Courier Westford

Contents

Chapter 12 635

Preface

The objective of this text is to serve as a cornerstone for the learning of logic design, digital system design, and computer design by a broad audience of readers. This fifth edition marks the continued evolution of the text contents. Beginning as an adaptation of a previous book by the first author in 1997, it continues to offer a unique combination of logic design and computer design principles with a strong hardware emphasis. Over the years, the text has followed industry trends by adding new material such as hardware description languages, removing or de-emphasizing material of declining importance, and revising material to track changes in computer technology and computer-aided design.

New to This Edition

The fifth edition reflects changes in technology and design practice that require computer system designers to work at higher levels of abstraction and manage larger ranges of complexity than they have in the past. The level of abstraction at which logic, digital systems, and computers are designed has moved well beyond the level at which these topics are typically taught. The goal in updating the text is to more effectively bridge the gap between existing pedagogy and practice in the design of computer systems, particularly at the logic level. At the same time, the new edition maintains an organization that should permit instructors to tailor the degree of technology coverage to suit both electrical and computer engineering and computer science audiences. The primary changes to this edition include:

- Chapter 1 has been updated to include a discussion of the layers of abstractions in computing systems and their role in digital design, as well as an overview of the digital design process. Chapter 1 also has new material on alphanumeric codes for internationalization.
- The textbook introduces hardware description languages (HDLs) earlier, starting in Chapter 2. HDL descriptions of circuits are presented alongside logic schematics and state diagrams throughout the chapters on combinational and sequential logic design to indicate the growing importance of HDLs in contemporary digital system design practice. The material on propagation delay, which is a first-order design constraint in digital systems, has been moved into Chapter 2.
- Chapter 3 combines the functional block material from the old Chapter 3 and the arithmetic blocks from the old Chapter 4 to present a set of commonly

occurring combinational logic functional blocks. HDL models of the functional blocks are presented throughout the chapter. Chapter 3 introduces the concept of hierarchical design.

- Sequential circuits appear in Chapter 4, which includes both the description of design processes from the old Chapter 5, and the material on sequential circuit timing, synchronization of inputs, and metastability from the old Chapter 6. The description of JK and T flip-flops has been removed from the textbook and moved to the online Companion Website.
- Chapter 5 describes topics related to the implementation of digital hardware, including design of complementary metal-oxide (CMOS) gates and programmable logic. In addition to much of the material from the old Chapter 6, Chapter 5 now includes a brief discussion of the effect of testing and verification on the cost of a design. Since many courses employing this text have lab exercises based upon field programmable gate arrays (FPGAs), the description of FPGAs has been expanded, using a simple, generic FPGA architecture to explain the basic programmable elements that appear in many commercially available FPGA families.
- The remaining chapters, which cover computer design, have been updated to reflect changes in the state-of-the art since the previous edition appeared. Notable changes include moving the material on high-impedance buffers from the old Chapter 2 to the bus transfer section of Chapter 6 and adding a discussion of how procedure call and return instructions can be used to implement function calls in high level languages in Chapter 9.

Offering integrated coverage of both digital and computer design, this edition of *Logic and Computer Design Fundamentals* features a strong emphasis on fundamentals underlying contemporary design. Understanding of the material is supported by clear explanations and a progressive development of examples ranging from simple combinational applications to a CISC architecture built upon a RISC core. A thorough coverage of traditional topics is combined with attention to computer-aided design, problem formulation, solution verification, and the building of problem-solving skills. Flexibility is provided for selective coverage of logic design, digital system design, and computer design topics, and for coverage of hardware description languages (none, VHDL, or Verilog®).

With these revisions, Chapters 1 through 4 of the book treat logic design, Chapters 5 through 7 deal with digital systems design, and Chapters 8 through 12 focus on computer design. This arrangement provides solid digital system design fundamentals while accomplishing a gradual, bottom-up development of fundamentals for use in top-down computer design in later chapters. Summaries of the topics covered in each chapter follow.

Logic Design

Chapter 1, Digital Systems and Information, introduces digital computers, computer systems abstraction layers, embedded systems, and information representation including number systems, arithmetic, and codes.

Chapter 2, Combinational Logic Circuits, deals with gate circuits and their types and basic ideas for their design and cost optimization. Concepts include Boolean algebra, algebraic and Karnaugh-map optimization, propagation delay, and gate-level hardware description language models using structural and dataflow models in both VHDL and Verilog.

Chapter 3, Combinational Logic Design, begins with an overview of a contemporary logic design process. The details of steps of the design process including problem formulation, logic optimization, technology mapping to NAND and NOR gates, and verification are covered for combinational logic design examples. In addition, the chapter covers the functions and building blocks of combinational design including enabling and input-fixing, decoding, encoding, code conversion, selecting, distributing, addition, subtraction, incrementing, decrementing, filling, extension and shifting, and their implementations. The chapter includes VHDL and Verilog models for many of the logic blocks.

Chapter 4, Sequential Circuits, covers sequential circuit analysis and design. Latches and edge-triggered flip-flops are covered with emphasis on the D type. Emphasis is placed on state machine diagram and state table formulation. A complete design process for sequential circuits including specification, formulation, state assignment, flip-flop input and output equation determination, optimization, technology mapping, and verification is developed. A graphical state machine diagram model that represents sequential circuits too complex to model with a conventional state diagram is presented and illustrated by two real world examples. The chapter includes VHDL and Verilog descriptions of a flip-flop and a sequential circuit, introducing procedural behavioral VHDL and Verilog language constructs as well as test benches for verification. The chapter concludes by presenting delay and timing for sequential circuits, as well as synchronization of asynchronous inputs and metastability.

Digital Systems Design

Chapter 5, Digital Hardware Implementation, presents topics focusing on various aspects of underlying technology including the MOS transistor and CMOS circuits, and programmable logic technologies. Programmable logic covers read-only memories, programmable logic arrays, programmable array logic, and field programmable gate arrays (FPGAs). The chapter includes examples using a simple FPGA architecture to illustrate many of the programmable elements that appear in more complex, commercially available FPGA hardware.

Chapter 6, Registers and Register Transfers, covers registers and their applications. Shift register and counter design are based on the combination of flip-flops with functions and implementations introduced in Chapters 3 and 4. Only the ripple counter is introduced as a totally new concept. Register transfers are considered for both parallel and serial designs and time–space trade-offs are discussed. A section focuses on register cell design for multifunction registers that perform multiple operations. A process for the integrated design of datapaths and control units using register transfer statements and state machine diagrams is introduced and illustrated by two real world examples. Verilog and VHDL descriptions of selected register types are introduced.

Chapter 7, Memory Basics, introduces static random access memory (SRAM) and dynamic random access memory (DRAM), and basic memory systems. It also describes briefly various distinct types of DRAMs.

Computer design

Chapter 8, Computer Design Basics, covers register files, function units, datapaths, and two simple computers: a single-cycle computer and a multiple-cycle computer. The focus is on datapath and control unit design formulation concepts applied to implementing specified instructions and instruction sets in single-cycle and multiple-cycle designs.

Chapter 9, Instruction Set Architecture, introduces many facets of instruction set architecture. It deals with address count, addressing modes, architectures, and the types of instructions and presents floating-point number representation and operations. Program control architecture is presented including procedure calls and interrupts.

Chapter 10, RISC and CISC Processors, covers high-performance processor concepts including a pipelined RISC processor and a CISC processor. The CISC processor, by using microcoded hardware added to a modification of the RISC processor, permits execution of the CISC instruction set using the RISC pipeline, an approach used in contemporary CISC processors. Also, sections describe high-performance CPU concepts and architecture innovations including two examples of multiple CPU microprocessors.

Chapter 11, Input–Output and Communication, deals with data transfer between the CPU and memory, input–output interfaces and peripheral devices. Discussions of a keyboard, a Liquid Crystal Display (LCD) screen, and a hard drive as peripherals are included, and a keyboard interface is illustrated. Other topics range from serial communication, including the Universal Serial Bus (USB), to interrupt system implementation.

Chapter 12, Memory Systems, focuses on memory hierarchies. The concept of locality of reference is introduced and illustrated by consideration of the cache/main memory and main memory/hard drive relationships. An overview of cache design parameters is provided. The treatment of memory management focuses on paging and a translation lookaside buffer supporting virtual memory.

In addition to the text itself, a Companion Website and an Instructor's Manual are provided. Companion Website (www.pearsonglobaleditions.com/Mano) content includes the following: 1) reading supplements including material deleted from prior editions, 2) VHDL and Verilog source files for all examples, 3) links to computer-aided design tools for FPGA design and HDL simulation, 4) solutions for about one-third of all text chapter problems, 5) errata, 6) PowerPoint® slides for Chapters 1 through 8, 7) projection originals for complex figures and tables from the text, and 8) site news sections for students and instructors pointing out new material, updates, and corrections. Instructors are encouraged to periodically check the instructor's site news so that they are aware of site changes. **Instructor's Manual** content includes suggestions for use of the book and all problem solutions. Online access to this manual is available from Pearson to instructors at academic institutions who adopt the book for classroom use. The suggestions for use provide important detailed information for navigating the text to fit with various course syllabi.

Because of its broad coverage of both logic and computer design, this book serves several different objectives in sophomore through junior level courses. Chapters 1 through 9 with selected sections omitted, provide an overview of hardware for computer science, computer engineering, electrical engineering, or engineering students in general in a single semester course. Chapters 1 through 4 possibly with selected parts of 5 through 7 give a basic introduction to logic design, which can be completed in a single quarter for electrical and computer engineering students. Covering Chapters 1 through 7 in a semester provides a stronger, more contemporary logic design treatment. The entire book, covered in two quarters, provides the basics of logic and computer design for computer engineering and science students. Coverage of the entire book with appropriate supplementary material or a laboratory component can fill a two-semester sequence in logic design and computer architecture. Due to its moderately paced treatment of a wide range of topics, the book is ideal for self-study by engineers and computer scientists. Finally, all of these various objectives can also benefit from use of reading supplements provided on the Companion Website.

The authors would like to acknowledge the instructors whose input contributed to the previous edition of the text and whose influence is still apparent in the current edition, particularly Professor Bharat Bhuva, Vanderbilt University; Professor Donald Hung, San Jose State University; and Professors Katherine Compton, Mikko Lipasti, Kewal Saluja, and Leon Shohet, and Faculty Associate Michael Morrow, ECE, University of Wisconsin, Madison. We appreciate corrections to the previous editions provided by both instructors and students, most notably, those from Professor Douglas De Boer of Dordt College. In getting ready to prepare to think about getting started to commence planning to begin working on the fifth edition, I received valuable feedback on the fourth edition from Patrick Schaumont and Cameron Patterson at Virginia Tech, and Mark Smith at the Royal Institute of Technology (KTH) in Stockholm, Sweden. I also benefited from many discussions with Kristie Cooper and Jason Thweatt at Virginia Tech about using the fourth edition in the updated version of our department's Introduction to Computer Engineering course. I would also like to express my appreciation to the folks at Pearson for their hard work on this new edition. In particular, I would like to thank Andrew Gilfillan for choosing me to be the new third author and for his help in planning the new edition; Julie Bai for her deft handling of the transition after Andrew moved to another job, and for her guidance, support, and invaluable feedback on the manuscript; Pavithra Jayapaul for her help in text production and her patience in dealing with my delays (especially in writing this preface!); and Scott Disanno and Shylaja Gattupalli for their guidance and care in producing the text. Special thanks go to Morris Mano and Charles Kime for their efforts in writing the previous editions of this book. It is an honor and a privilege to have been chosen as their successor. Finally, I would like to thank Karen, Guthrie, and Eli for their patience and support while I was writing, especially for keeping our mutt Charley away from this laptop so that he didn't eat the keys like he did with its short-lived predecessor.

Tom Martin
Blacksburg, Virginia

Global Edition

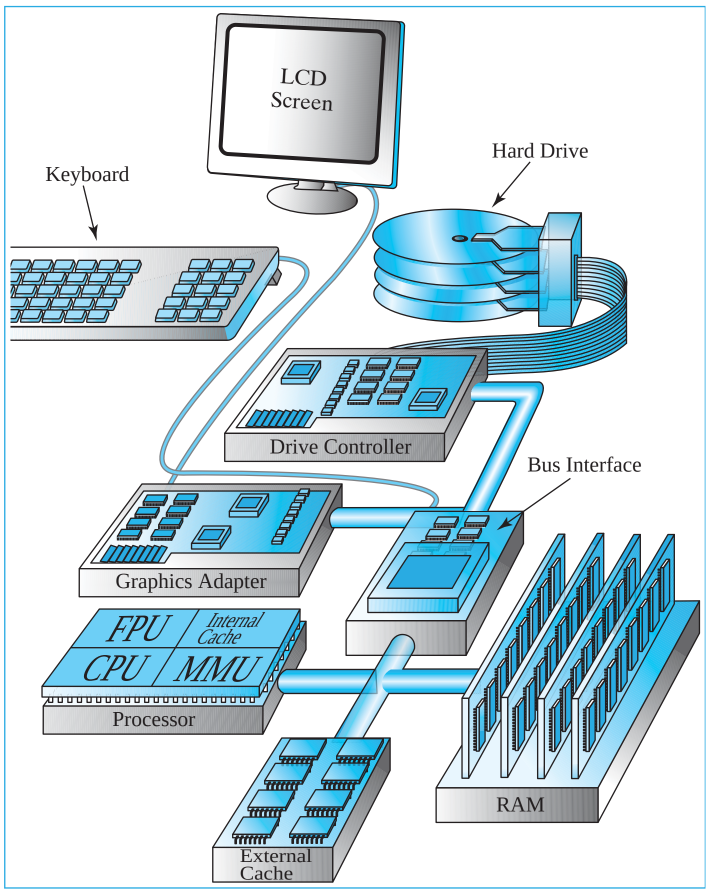
The publishers would like to thank the following for their contribution to the Global Edition:

Contributors

Chiranjib Koley, Associate Professor, *National Institute of Technology,* Durgapur
Lyla B. Das, Associate Professor, *National Institute of Technology,* Calicut

Reviewers

Debaprasad Das, Professor, *Assam University*
Moumita Mitra Manna, Lecturer of Computer Science and Applications at *Bangabasi College, University of Calcutta*
Piyali Sengupta, Freelance



LCD
Screen
Hard Drive
Keyboard
Drive Controller
Bus Interface
Graphics Adapter
FPU
Internal
Cache
CPU
MMU
Processor
RAM
External
Cache

CHAPTER

1

Digital Systems and Information

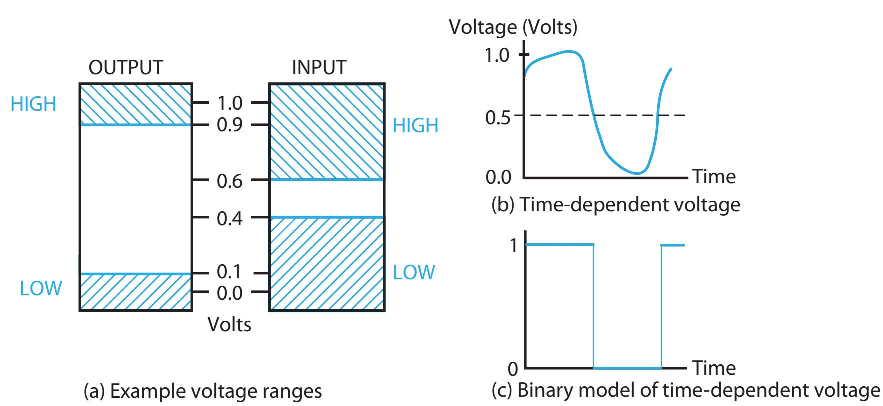
This book deals with logic circuits and digital computers. Early computers were used for computations with discrete numeric elements called *digits* (the Latin word for fingers)—hence the term *digital computer*. The use of “digital” spread from the computer to logic circuits and other systems that use discrete elements of information, giving us the terms *digital circuits* and *digital systems*. The term *logic* is applied to circuits that operate on a set of just two elements with values True (1) and False (0). Since computers are based on logic circuits, they operate on patterns of elements from these two-valued sets, which are used to represent, among other things, the decimal digits. Today, the term “digital circuits” is viewed as synonymous with the term “logic circuits.”

The *general-purpose digital computer* is a digital system that can follow a stored sequence of instructions, called a *program*, that operates on data. The user can specify and change the program or the data according to specific needs. As a result of this flexibility, general-purpose digital computers can perform a variety of information-processing tasks, ranging over a very wide spectrum of applications. This makes the digital computer a highly general and very flexible digital system. Also, due to its generality, complexity, and widespread use, the computer provides an ideal vehicle for learning the concepts, methods, and tools of digital system design. To this end, we use the exploded pictorial diagram of a computer of the class commonly referred to as a PC (personal computer) given on the opposite page. We employ this generic computer to highlight the significance of the material covered and its relationship to the overall system. A bit later in this chapter, we will discuss the various major components of the generic computer and see how they relate to a block diagram commonly used to represent a computer. We then describe the concept of layers of abstraction in digital system design, which enables us to manage the complexity of designing and programming computers constructed using billions of transistors. Otherwise, the remainder of the chapter focuses on the digital systems in our daily lives and introduces approaches for representing information in digital circuits and systems.

1-1 Information Representation

Digital systems store, move, and process information. The information represents a broad range of phenomena from the physical and man-made world. The physical world is characterized by parameters such as weight, temperature, pressure, velocity, flow, and sound intensity and frequency. Most physical parameters are *continuous*, typically capable of taking on all possible values over a defined range. In contrast, in the man-made world, parameters can be discrete in nature, such as business records using words, quantities, and currencies, taking on values from an alphabet, the integers, or units of currency, respectively. In general, information systems must be able to represent both continuous and discrete information. Suppose that temperature, which is continuous, is measured by a sensor and converted to an electrical voltage, which is likewise continuous. We refer to such a continuous voltage as an *analog signal*, which is one possible way to represent temperature. But, it is also possible to represent temperature by an electrical voltage that takes on *discrete* values that occupy only a finite number of values over a range, for example, corresponding to integer degrees centigrade between −40 and +119. We refer to such a voltage as a *digital signal*. Alternatively, we can represent the discrete values by multiple voltage signals, each taking on a discrete value. At the extreme, each signal can be viewed as having only two discrete values, with multiple signals representing a large number of discrete values. For example, each of the 160 values just mentioned for temperature can be represented by a particular combination of eight two-valued signals. The signals in most present-day electronic digital systems use just two discrete values and are therefore said to be *binary*. The two discrete values used are often called 0 and 1, the digits for the binary number system.

We typically represent the two discrete values by ranges of voltage values called HIGH and LOW. Output and input voltage ranges are illustrated in Figure 1-1(a). The HIGH output voltage value ranges between 0.9 and 1.1 volts, and the LOW output voltage value between −0.1 and 0.1 volts. The high input range allows 0.6 to 1.1 volts to be recognized as a HIGH, and the low input range allows



□ **FIGURE 1-1**
Examples of Voltage Ranges and Waveforms for Binary Signals

−0.1 to 0.4 volts to be recognized as a LOW. The fact that the input ranges are wider than the output ranges allows the circuits to function correctly in spite of variations in their behavior and undesirable "noise" voltages that may be added to or subtracted from the outputs.

We give the output and input voltage ranges a number of different names. Among these are HIGH (H) and LOW (L), TRUE (T) and FALSE (F), and 1 and 0. It is natural to associate the higher voltage ranges with HIGH or H, and the lower voltage ranges with LOW or L. For TRUE and 1 and FALSE and 0, however, there is a choice. TRUE and 1 can be associated with either the higher or lower voltage range and FALSE and 0 with the other range. Unless otherwise indicated, we assume that TRUE and 1 are associated with the higher of the voltage ranges, H, and the FALSE and 0 are associated with the lower of the voltage ranges, L. This particular convention is called *positive logic*.

It is interesting to note that the values of voltages for a digital circuit in Figure 1-1(a) are still continuous, ranging from −0.1 to +1.1 volts. Thus, the voltage is actually analog! The actual voltages values for the output of a very high-speed digital circuit are plotted versus time in Figure 1-1(b). Such a plot is referred to as a *waveform*. The interpretation of the voltage as binary is based on a model using voltage ranges to represent discrete values 0 and 1 on the inputs and the outputs. The application of such a model, which redefines all voltage above 0.5 V as 1 and below 0.5 V as 0 in Figure 1-1(b), gives the waveform in Figure 1-1(c). The output has now been interpreted as binary, having only discrete values 1 and 0, with the actual voltage values removed. We note that digital circuits, made up of electronic devices called transistors, are designed to cause the outputs to occupy the two distinct output voltage ranges for 1 (H) and 0 (L) in Figure 1-1, whenever the outputs are not changing. In contrast, analog circuits are designed to have their outputs take on continuous values over their range, whether changing or not.

Since 0 and 1 are associated with the binary number system, they are the preferred names for the signal ranges. A binary digit is called a *bit*. Information is represented in digital computers by groups of bits. By using various coding techniques, groups of bits can be made to represent not only binary numbers, but also other groups of discrete symbols. Groups of bits, properly arranged, can even specify to the computer the program instructions to be executed and the data to be processed.

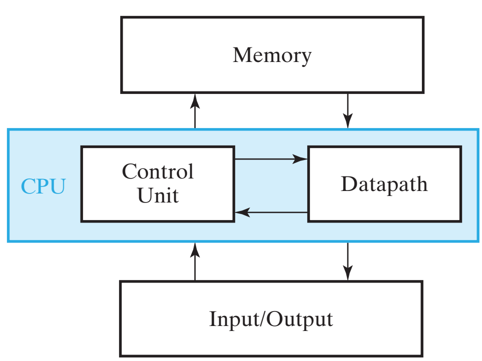
Why is binary used? In contrast to the situation in Figure 1-1, consider a system with 10 values representing the decimal digits. In such a system, the voltages available—say, 0 to 1.0 volts—could be divided into 10 ranges, each of length 0.1 volt. A circuit would provide an output voltage within each of these 10 ranges. An input of a circuit would need to determine in which of the 10 ranges an applied voltage lies. If we wish to allow for noise on the voltages, then output voltage might be permitted to range over less than 0.05 volt for a given digit representation, and boundaries between inputs could vary by less than 0.05 volt. This would require complex and costly electronic circuits, and the output still could be disturbed by small "noise" voltages or small variations in the circuits occurring during their manufacture or use. As a consequence, the use of such multivalued circuits is very limited. Instead, binary circuits are used in which correct circuit operation can be achieved with significant variations in values of the two output voltages and the two input ranges. The resulting transistor circuit with an output that is either HIGH or LOW is simple, easy to design, and extremely reliable. In addition, this use of binary values makes the results calculated repeatable in the sense that the same set of input values to a calculation always gives exactly the same set of outputs. This is not necessarily the case for multivalued or analog circuits, in which noise voltages and small variations due to manufacture or circuit aging can cause results to differ at different times.

The Digital Computer

A block diagram of a digital computer is shown in Figure 1-2. The memory stores programs as well as input, output, and intermediate data. The datapath performs arithmetic and other data-processing operations as specified by the program. The control unit supervises the flow of information between the various units. A datapath, when combined with the control unit, forms a component referred to as a *central processing unit*, or CPU.

The program and data prepared by the user are transferred into memory by means of an input device such as a keyboard. An output device, such as an LCD (liquid crystal display), displays the results of the computations and presents them to the user. A digital computer can accommodate many different input and output devices, such as DVD drives, USB flash drives, scanners, and printers. These devices use digital logic circuits, but often include analog electronic circuits, optical sensors, LCDs, and electromechanical components.

The control unit in the CPU retrieves the instructions, one by one, from the program stored in the memory. For each instruction, the control unit manipulates the datapath to execute the operation specified by the instruction. Both program and data are stored in memory. A digital computer can perform arithmetic computations, manipulate strings of alphabetic characters, and be programmed to make decisions based on internal and external conditions.

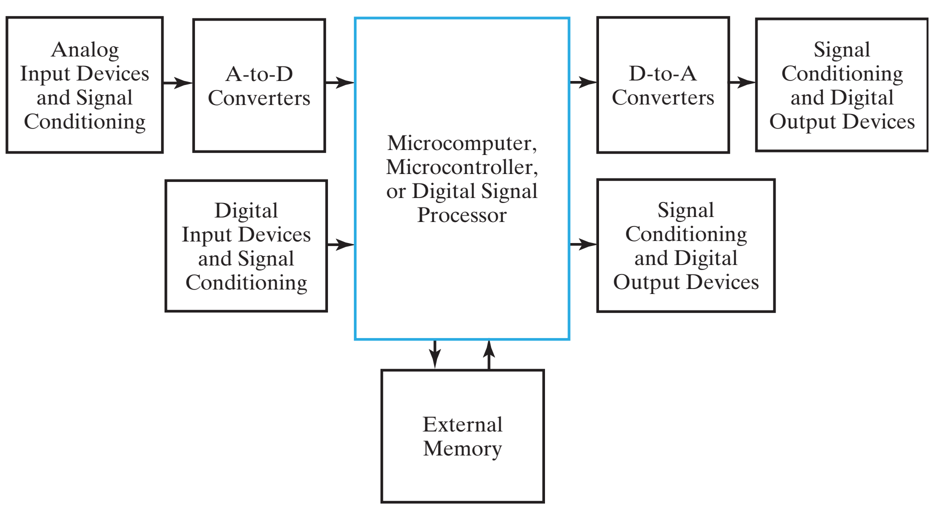


☐ **FIGURE 1-2**
Block Diagram of a Digital Computer

Beyond the Computer

In terms of world impact, computers, such as the PC, are not the end of the story. Smaller, often less powerful, single-chip computers called *microcomputers* or *microcontrollers*, or special-purpose computers called *digital signal processors* (DSPs) actually are more prevalent in our lives. These computers are parts of everyday products and their presence is often not apparent. As a consequence of being integral parts of other products and often enclosed within them, they are called *embedded systems*. A generic block diagram of an embedded system is shown in Figure 1-3. Central to the system is the microcomputer (or its equivalent). It has many of the characteristics of the PC, but differs in the sense that its software programs are often permanently stored to provide only the functions required for the product. This software, which is critical to the operation of the product, is an integral part of the embedded system and referred to as *embedded software*. Also, the human interface of the microcomputer can be very limited or nonexistent. The larger information-storage components such as a hard drive and compact disk or DVD drive frequently are not present. The microcomputer contains some memory; if additional memory is needed, it can be added externally.

With the exception of the external memory, the hardware connected to the embedded microcomputer in Figure 1-3 interfaces with the product and/or the outside world. The input devices transform inputs from the product or outside world into electrical signals, and the output devices transform electrical signals into outputs to the product or outside world. The input and output devices are of two types, those which use analog signals and those which use digital signals. Examples of digital input devices include a limit switch which is closed or open depending on whether a force is applied to it and a keypad having ten decimal integer buttons. Examples of



◻ **FIGURE 1-3**
Block Diagram of an Embedded System

analog input devices include a thermistor which changes its electrical resistance in response to the temperature and a crystal which produces a charge (and a corresponding voltage) in response to the pressure applied. Typically, electrical or electronic circuitry is required to "condition" the signal so that it can be read by the embedded system. Examples of digital output devices include relays (switches that are opened or closed by applied voltages), a stepper motor that responds to applied voltage pulses, or an LED digital display. Examples of analog output devices include a loudspeaker and a panel meter with a dial. The dial position is controlled by the interaction of the magnetic fields of a permanent magnet and an electromagnet driven by the voltage applied to the meter.

Next, we illustrate embedded systems by considering a temperature measurement performed by using a wireless weather station. In addition, this example also illustrates analog and digital signals, including conversion between the signal types.

EXAMPLE 1-1 Temperature Measurement and Display

A wireless weather station measures a number of weather parameters at an outdoor site and transmits them for display to an indoor base station. Its operation can be illustrated by considering the temperature measurement illustrated in Figure 1-4 with reference to the block diagram in Figure 1-3. Two embedded microprocessors are used, one in the outdoor site and the other in the indoor base station.

The temperature at the outdoor site ranges continuously from $-40°F$ to $+115°F$. Temperature values over one 24-hour period are plotted as a function of time in Figure 1-4(a). This temperature is measured by a sensor consisting of a thermistor (a resistance that varies with temperature) with a fixed current applied by an electronic circuit. This sensor provides an analog voltage that is proportional to the temperature. Using signal conditioning, this voltage is changed to a continuous voltage ranging between 0 and 15 volts, as shown in Figure 1-4(b).

The analog voltage is sampled at a rate of once per hour (a very slow sampling rate used just for illustration), as shown by the dots in Figure 1-4(b). Each value sampled is applied to an analog-to-digital (A/D) converter, as in Figure 1-3, which replaces the value with a digital number written in binary and having decimal values between 0 and 15, as shown in Figure 1-4(c). A binary number can be interpreted in decimal by multiplying the bits from left to right times the respective weights, 8, 4, 2, and 1, and adding the resulting values. For example, 0101 can be interpreted as $0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 5$. In the process of conversion, the value of the temperature is quantized from an infinite number of values to just 16 values. Examining the correspondence between the temperature in Figure 1-4(a) and the voltage in Figure 1-4(b), we find that the typical digital value of temperature represents an actual temperature range up to 5 degrees above or below the digital value. For example, the analog temperature range between -25 and -15 degrees is represented by the digital temperature value of -20 degrees. This discrepancy between the actual temperature and the digital temperature is called the *quantization error*. In order to obtain greater precision, we would need to increase the number of bits beyond four in the output of the A/D converter. The hardware components for sensing, signal conditioning, and A/D conversion are shown in the upper left corner of Figure 1-3.

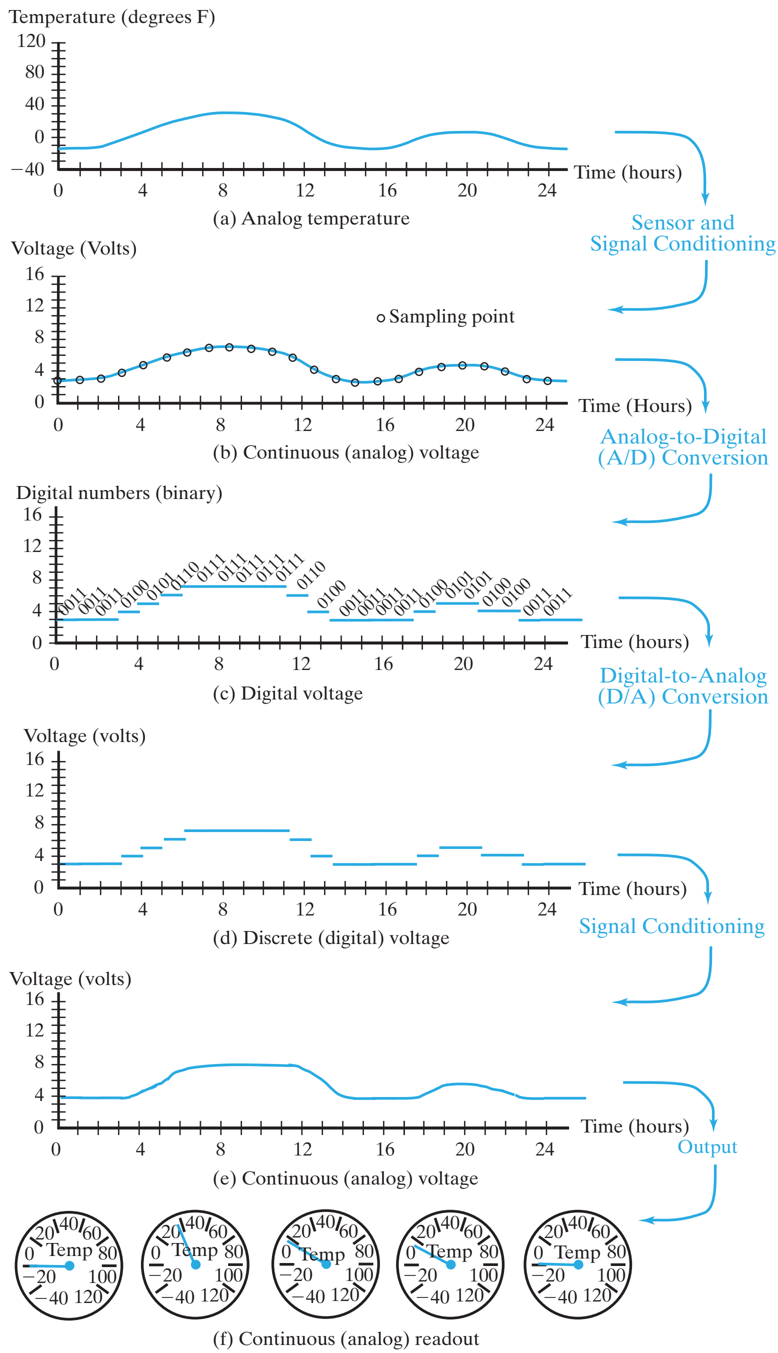


FIGURE 1-4
Temperature Measurement and Display

Next, the digital value passes through the microcomputer to a wireless transmitter as a digital output device in the lower right corner of Figure 1-3. The digital value is transmitted to a wireless receiver, which is a digital input device in the base station. The digital value enters the microcomputer at the base station, where calculations may be performed to adjust its value based on thermistor properties. The resulting value is to be displayed with an analog meter shown in Figure 1-4(f) as the output device. In order to support this display, the digital value is converted to an analog value by a digital-to-analog converter, giving the quantized, discrete voltage levels shown in Figure 1-4(d). Signal conditioning, such as processing of the output by a low-pass analog filter, is applied to give the continuous signal in Figure 1-4(e). This signal is applied to the analog voltage display, which has been labeled with the corresponding temperature values shown for five selected points over the 24-hour period in Figure 1-4(f). ■

You might ask: "How many embedded systems are there in my current living environment?" Do you have a cell phone? An iPod™? An Xbox™? A digital camera? A microwave oven? An automobile? All of these are embedded systems. In fact, a late-model automobile can contain more than 50 microcontrollers, each controlling a distinct embedded system, such as the engine control unit (ECU), automatic braking system (ABS), and stability control unit (SCU). Further, a significant proportion of these embedded systems communicate with each other through a CAN (controller area network). A more recently developed automotive network, called FlexRay, provides high-speed, reliable communication for safety-critical tasks such as braking-by-wire and steering-by-wire, eliminating primary dependence on mechanical and hydraulic linkages and enhancing the potential for additional safety features such as collision avoidance. Table 1-1 lists examples of embedded systems classified by application area.

Considering the widespread use of personal computers and embedded systems, digital systems have a major impact on our lives, an impact that is not often fully appreciated. Digital systems play central roles in our medical diagnosis and treatment, in our educational institutions and workplaces, in moving from place to place, in our homes, in interacting with others, and in just having fun! The complexity of many of these systems requires considerable care at many levels of design abstraction to make the systems work. Thanks to the invention of the transistor and the integrated circuit and to the ingenuity and perseverance of millions of engineers and programmers, they indeed work and usually work well. In the remainder of this text, we take you on a journey that reveals how digital systems work and provide a detailed look at how to design digital systems and computers.

More on the Generic Computer

At this point, we will briefly discuss the generic computer and relate its various parts to the block diagram in Figure 1-2. At the lower left of the diagram at the beginning of this chapter is the heart of the computer, an integrated circuit called the *processor*. Modern processors such as this one are quite complex and consist of tens to hundreds of millions of transistors. The processor contains four functional modules: the CPU, the FPU, the MMU, and the internal cache.

TABLE 1-1
Embedded System Examples

Application Area	Product
Banking, commerce and manufacturing	Copiers, FAX machines, UPC scanners, vending machines, automatic teller machines, automated warehouses, industrial robots, 3D printers
Communication	Wireless access points, network routers, satellites
Games and toys	Video games, handheld games, talking stuffed toys
Home appliances	Digital alarm clocks, conventional and microwave ovens, dishwashers
Media	CD players, DVD players, flat panel TVs, digital cameras, digital video cameras
Medical equipment	Pacemakers, incubators, magnetic resonance imaging
Personal	Digital watches, MP3 players, smart phones, wearable fitness trackers
Transportation and navigation	Electronic engine controls, traffic light controllers, aircraft flight controls, global positioning systems

We have already discussed the CPU. The FPU (*floating-point unit*) is somewhat like the CPU, except that its datapath and control unit are specifically designed to perform floating-point operations. In essence, these operations process information represented in the form of scientific notation (e.g., 1.234×10^7), permitting the generic computer to handle very large and very small numbers. The CPU and the FPU, in relation to Figure 1-2, each contain a datapath and a control unit.

The MMU is the *memory management unit*. The MMU plus the internal cache and the separate blocks near the bottom of the computer labeled "External Cache" and "RAM" (*random-access memory*) are all part of the memory in Figure 1-2. The two caches are special kinds of memory that allow the CPU and FPU to get at the data to be processed much faster than with RAM alone. RAM is what is most commonly referred to as memory. As its main function, the MMU causes the memory that appears to be available to the CPU to be much, much larger than the actual size of the RAM. This is accomplished by data transfers between the RAM and the hard drive shown at the top of the drawing of the generic computer. So the hard drive, which we discuss later as an input/output device, conceptually appears as a part of both the memory and input/output.

The connection paths shown between the processor, memory, and external cache are the pathways between integrated circuits. These are typically implemented as fine copper conductors on a printed circuit board. The connection paths below the bus interface are referred to as the processor bus. The connections above the bus interface are the input/output (I/O) bus. The processor bus and the I/O bus attached to the bus interface carry data having different numbers of bits and have different ways of controlling the movement of data. They may also operate at different speeds. The bus interface hardware handles these differences so that data can be communicated between the two buses.

All of the remaining structures in the generic computer are considered part of I/O in Figure 1-2. In terms of sheer physical volume, these structures dominate. In order to enter information into the computer, a keyboard is provided. In order to view output in the form of text or graphics, a graphics adapter card and LCD (*liquid crystal display*) screen are provided. The hard drive, discussed previously, is an electromechanical magnetic storage device. It stores large quantities of information in the form of magnetic flux on spinning disks coated with magnetic materials. In order to control the hard drive and transfer information to and from it, a drive controller is used. The keyboard, graphics adapter card, and drive controller card are all attached to the I/O bus. This allows these devices to communicate through the bus interface with the CPU and other circuitry connected to the processor buses.

1-2 Abstraction Layers in Computer Systems Design

As described by Moggridge, design is the process of understanding all the relevant constraints for a problem and arriving at a solution that balances those constraints. In computer systems, typical constraints include functionality, speed, cost, power, area, and reliability. At the time that this text is being written in 2014, leading edge integrated circuits have billions of transistors—designing such a circuit one transistor at a time is impractical. To manage that complexity, computer systems design is typically performed in a "top down" approach, where the system is specified at a high level and then the design is decomposed into successively smaller blocks until a block is simple enough that it can be implemented. These blocks are then connected together to make the full system. The generic computer described in the previous section is a good example of blocks connected together to make a full system. This book begins with smaller blocks and then moves toward putting them together into larger, more complex blocks.

A fundamental aspect of the computer systems design process is the concept of "layers of abstraction." Computer systems such as the generic computer can be viewed at several layers of abstraction from circuits to algorithms, with each higher layer of abstraction hiding the details and complexity of the layer below. Abstraction removes unnecessary implementation details about a component in the system so that a designer can focus on the aspects of the component that matter for the problem being solved. For example, when we write a computer program to add two variables and store the result in a third variable, we focus on the programming language constructs used to declare the variables and describe the addition operation. But when the program executes, what really happens is that electrical charge is moved around by transistors and stored in capacitive layers to represent the bits of data and

Algorithms
Programming Languages
Operating Systems
Instruction Set Architecture
Microarchitecture
Register Transfers
Logic Gates
Transistor Circuits

FIGURE 1-5
Typical Layers of Abstraction in Modern Computer Systems

control signals necessary to perform the addition and store the result. It would be difficult to write programs if we had to directly describe the flow of electricity for individual bits. Instead, the details of controlling them are managed by several layers of abstractions that transform the program into a series of more detailed representations that eventually control the flow of electrical charges that implement the computation.

Figure 1-5 shows the typical layers of abstraction in contemporary computing systems. At the top of the abstraction layers, algorithms describe a series of steps that lead to a solution. These algorithms are then implemented as a program in a high-level programming language such as C++, Python, or Java. When the program is running, it shares computing resources with other programs under the control of an operating system. Both the operating system and the program are composed of sequences of instructions that are particular to the processor running them; the set of instructions and the registers (internal data memory) available to the programmer are known as the instruction set architecture. The processor hardware is a particular implementation of the instruction set architecture, referred to as the microarchitecture; manufacturers very often make several different microarchitectures that execute the same instruction set. A microarchitecture can be described as underlying sequences of transfers of data between registers. These register transfers can be decomposed into logic operations on sets of bits performed by logic gates, which are electronic circuits implemented with transistors or other physical devices that control the flow of electrons.

An important feature of abstraction is that lower layers of abstraction can usually be modified without changing the layers above them. For example, a program written in C++ can be compiled on any computer system with a C++ compiler and then executed. As another example, an executable program for the Intel™ x86 instruction set architecture can run on any microarchitecture (implementation) of that architecture, whether that implementation is from Intel™ or AMD. Consequently, abstraction allows us to continue to use solutions at higher layers of abstraction even when the underlying implementations have changed.

This book is mainly concerned with the layers of abstraction from logic gates up to operating systems, focusing on the design of the hardware up to the interface between the hardware and the software. By understanding the interactions of the layers of abstraction, we can choose the proper layer of abstraction on which to concentrate for a given design, ignoring unnecessary details and optimizing the aspects of the system that are likely to have the most impact on achieving the proper balance of constraints for a successful design. Oftentimes, the higher layers of abstraction have the potential for much more improvement in the design than can be found at the lower layers. For example, it might be possible to re-design a hardware circuit for multiplying two numbers so that it runs 20–50% faster than the original, but it might be possible to have much bigger impact on the speed of the overall circuit if the algorithm is modified to not use multiplication at all. As technology has progressed and computer systems have become more complex, the design effort has shifted to higher layers of abstraction and, at the lower layers, much of the design process has been automated. Effectively using the automated processes requires an understanding of the fundamentals of design at those layers of abstraction.

An Overview of the Digital Design Process

The design of a digital computer system starts from the specification of the problem and culminates in representation of the system that can be implemented. The design process typically involves repeatedly transforming a representation of the system at one layer of abstraction to a representation at the next lower level of abstraction, for example, transforming register transfers into logic gates, which are in turn transformed into transistor circuits.

While the particular details of the design process depend upon the layer of abstraction, the procedure generally involves specifying the behavior of the system, generating an optimized solution, and then verifying that the solution meets the specification both in terms of functionality and in terms of design constraints such as speed and cost. As a concrete example of the procedure, the following steps are the design procedure for combinational digital circuits that Chapters 2 and 3 will introduce:

1. **Specification:** Write a specification for the behavior of the circuit, if one is not already available.
2. **Formulation:** Derive the truth table or initial Boolean equations that define the required logical relationships between inputs and outputs.
3. **Optimization:** Apply two-level and multiple-level optimization to minimize the number of logic gates required. Draw a logic diagram or provide a netlist for the resulting circuit using logic gates.
4. **Technology Mapping:** Transform the logic diagram or netlist to a new diagram or netlist using the available implementation technology.
5. **Verification:** Verify the correctness of the final design.

For digital circuits, the specification can take a variety of forms, such as text or a description in a hardware description language (HDL), and should include the respective symbols or names for the inputs and outputs. Formulation converts the specification into forms that can be optimized. These forms are typically truth tables or Boolean expressions. It is important that verbal specifications be interpreted correctly when formulating truth tables or expressions. Often the specifications are incomplete, and any wrong interpretation may result in an incorrect truth table or expression.