functionality, and will provide clear steps for converting finite-state machines to circuits. As with any form of design, we often use pre-designed building blocks, or components, to make our design task easier. Chapter 4 describes several such components, known as datapath components, including registers (for storing digital data), adders, comparators, multipliers, and small memories called register files, among other blocks. Chapter 5 introduces the modern approach to digital design, known as register-transfer level design, wherein we design systems consisting of datapath components controlled by controllers, to implement interesting and useful custom digital circuits. In fact, that chapter shows how to convert a C program to a custom digital circuit—clearly demonstrating that any desired function can be implemented as software on a microprocessor or as a custom digital circuit. That chapter also introduces some additional components, including ROM and RAM memories, and queues. Chapters 1 through 5 form the core of this book—after those five chapters, the reader can specify a wide variety of desired functionality and can convert that functionality to a working custom digital circuit.

Chapter 6 introduces methods for designing *better* digital circuits. The chapter describes methods for improving basic combinational circuits, basic sequential circuits, datapath components, and register-transfer level designs. That chapter emphasizes the important notion of *tradeoffs*, wherein we might make one aspect of the design better, but at the expense of worsening another aspect. Tradeoffs are the essence of design.

Chapter 7 describes different physical devices on which we can implement our digital circuits, including application-specific integrated circuits, field-programmable gate-arrays (FPGAs), simple programmable logic devices, and cheap off-the-shelf ICs.

Chapter 8 applies the digital design methods of earlier chapters to build a common type of digital circuit—a programmable microprocessor. The chapter demystifies the workings of a microprocessor, using a very simple design to illustrate the concepts.

Chapter 9 introduces hardware description languages, which are widely used in modern digital design for describing desired circuit functionality as well as for representing the final custom digital circuit design. Hardware description languages, looking much like software programming languages but with important extensions and differences, serve as the input to most modern digital design tools.

## ▶ 1.5 EXERCISES

### SECTION 1.2: THE WORLD OF DIGITAL SYSTEMS

1.1 What is a digital signal, and how does it differ from an analog signal? Give two everyday examples of digital phenomena (e.g., a window can be open or closed) and two everyday examples of analog phenomena.

1.2 Suppose an analog audio signal comes in over a wire, and the voltage on the wire can range from 0 Volts (V) to 3 V. You want to convert the analog signal to a digital signal. You decide to encode each sample using two bits, such that 0 V would be encoded as `00`, 1 V as `01`, 2 V as `10`, and 3 V as `11`. You sample the signal every 1 millisecond and detect the following sequence of voltages: 0V 0V 1V 2V 3V 2V 1V. Show the signal converted to digital as a stream of `0`s and `1`s.

1.3 Assume that 0 V is encoded as `00`, 1 V as `01`, 2 V as `10`, and 3 V as `11`. You are given a digital encoding of an audio signal as follows: `1111101001010000`. Plot the re-created analog signal with time on the *x*-axis and voltage on the *y*-axis. Assume that each encoding's corresponding voltage should be output for 1 millisecond.

1.4 Assume that a signal is encoded using 12 bits. Assume that many of the encodings turn out to be either `000000000000`, `000000000001`, or `111111111111`. We thus decide to create compressed encodings by representing `000000000000` as `00`, `000000000001` as `01`, and `111111111111` as `10`. `11` means that an uncompressed encoding follows. Using this encoding scheme, decompress the following encoded stream:

```
00 00 01 10 11 010101010101 00 00 10 10
```

1.5 Using the same encoding scheme as in Exercise 1.4, compress the following unencoded stream:

```
000000000000 000000000001 100000000000 111111111111
```

1.6 Encode the following words into bits using the ASCII encoding table in Figure 1.9.
(a) LET
(b) RESET!
(c) HELLO $1

1.7 Suppose you are building a keypad that has buttons A through G. A three-bit output should indicate which button is currently being pressed. `000` represents no button being pressed. Decide on a 3-bit encoding to represent each button being pressed.

1.8 Convert the following binary numbers to decimal numbers:
(a) 100
(b) 1011
(c) 0000000000001
(d) 111111
(e) 101010

1.9 Convert the following binary numbers to decimal numbers:
(a) 1010
(b) 1000000
(c) 11001100
(d) 11111
(e) 10111011001

1.10 Convert the following binary numbers to decimal numbers:
(a) 000011
(b) 1111
(c) 11110
(d) 111100
(e) 0011010

1.11 Convert the following decimal numbers to binary numbers using the addition method:
(a) 9
(b) 15
(c) 32
(d) 140

1.12 Convert the following decimal numbers to binary numbers using the addition method:
(a) 19
(b) 30
(c) 64
(d) 128

1.13 Convert the following decimal numbers to binary numbers using the addition method:
(a) 3
(b) 65
(c) 90
(d) 100

1.14 Convert the following decimal numbers to binary numbers using the divide-by-2 method:
(a) 9
(b) 15
(c) 32
(d) 140

1.15 Convert the following decimal numbers to binary numbers using the divide-by-2 method:
(a) 19
(b) 30
(c) 64
(d) 128

1.16 Convert the following decimal numbers to binary numbers using the divide-by-2 method:
(a) 3
(b) 65
(c) 90
(d) 100

1.17 Convert the following decimal numbers to binary numbers using the divide-by-2 method:
(a) 23
(b) 87
(c) 123
(d) 101

1.18 Convert the following binary numbers to hexadecimal:
(a) 11110000
(b) 11111111
(c) 01011010
(d) 1001101101101

1.19 Convert the following binary numbers to hexadecimal:
(a) 11001101
(b) 10100101
(c) 11110001
(d) 1101101111100

1.20 Convert the following binary numbers to hexadecimal:
(a) 11100111
(b) 11001000
(c) 10100100
(d) 011001101101101

1.21 Convert the following hexadecimal numbers to binary:
(a) FF
(b) F0A2
(c) 0F100
(d) 100

1.22 Convert the following hexadecimal numbers to binary:
(a) 4F5E
(b) 3FAD
(c) 3E2A
(d) DEED

1.23 Convert the following hexadecimal numbers to binary:
(a) B0C4
(b) 1EF03
(c) F002
(d) BEEF

1.24 Convert the following hexadecimal numbers to decimal:
(a) FF
(b) F0A2
(c) 0F100
(d) 100

1.25 Convert the following hexadecimal numbers to decimal:
(a) 10
(b) 4E3
(c) FF0
(d) 200

1.26 Convert the decimal number 128 to the following number systems:
(a) binary
(b) hexadecimal
(c) base three
(d) base five
(e) base fifteen

1.27 Compare the number of digits necessary to represent the following decimal numbers in binary, octal, decimal, and hexadecimal representations. You need not determine the actual representations—just the number of required digits. For example, representing the decimal number 12 requires four digits in binary (1100 is the actual representation), two digits in octal (14), two digits in decimal (12), and one digit in hexadecimal (C).
(a) 8
(b) 60
(c) 300
(d) 1000
(e) 999,999

1.28 Determine the decimal number ranges that can be represented in binary, octal, decimal, and hexadecimal using the following numbers of digits. For example, 2 digits can represent decimal number range 0 through 3 in binary (00 through 11), 0 through 63 in octal (00 through 77), 0 through 99 in decimal (00 through 99), and 0 through 255 in hexadecimal (00 through FF).
(a) 1
(b) 3
(c) 6
(d) 8

1.29 Rewrite the following bit quantities as byte quantities, using the most appropriate metric quantity; e.g., 16,000 bits is 2,000 bytes, most appropriately written as 2 Kbytes.
(a) 8,000,000
(b) 32,000,000,000
(c) 1,000,000,000

## SECTION 1.3: IMPLEMENTING DIGITAL SYSTEMS: PROGRAMMING MICROPROCESSORS VERSUS DESIGNING DIGITAL CIRCUITS

1.30 Use a microprocessor like that in Figure 1.23 to implement a system that sounds an alarm whenever there is motion detected at the same time in three different rooms. Each room's motion sensor is an input to the microprocessor; a `1` means motion, a `0` means no motion. The microprocessor can sound the alarm by setting an output wire "alarm" to `1`. Show the connections to and from the microprocessor, and the C code to execute on the microprocessor. Hint: this problem is similar to Example 1.10.

1.31 A security camera company wishes to add a face recognition feature to their cameras such that the camera only broadcasts video when a human face is detected in the video. The camera records 30 video frames per second. For each frame, the camera would execute a face recognition application. The application implemented on a microprocessor requires 50 ms. The application implemented as a custom digital circuit requires 1 ms. Compute the maximum number of frames per second that each implementation supports, and indicate which implementation is sufficient for 30 frames per second.

1.32 Suppose that a particular banking system supports encrypted transactions, and that decrypting each transaction consists of three sub-tasks A, B, and C. The execution times of each task on a microprocessor versus a custom digital circuit are 50 ms versus 1 ms for A, 20 ms versus 2 ms for B, and 20 ms versus 1 ms for C. Partition the tasks among the microprocessor and custom digital circuitry, such that you minimize the amount of custom digital circuitry, while meeting the constraint of decrypting at least 40 transactions per second. Assume that each task requires the same amount of digital circuitry.

1.33 How many possible partitionings are there of a set of $N$ tasks, where each task can be implemented either on a microprocessor or as a custom digital circuit? How many possible partitionings are there of a set of 20 tasks (expressed as a number without any exponents)?

## ▶ *DESIGNER PROFILE*

Kelly first became interested in engineering while attending a talk about engineering at a career fair in high school. "I was dazzled by the interesting ideas and the cool graphs." While in college, though, she learned that "there was much more to engineering than ideas and graphs. Engineers *apply* their ideas and skills to build things that really make a difference in people's lives, for generations to come."

In her first few years as an engineer, Kelly has worked on a variety of projects that may help numerous individuals. One project was a ventilator system like the one mentioned earlier in this chapter. "We designed a new control system that may enable people on ventilators to breathe with more comfort while still getting the proper amount of oxygen." In addition, she examined alternative implementations of that control system, including on a microprocessor, as a custom digital circuit, and as a combination of the two. "Today's technologies, like FPGAs, provide so many different options. We examined several options to see what the tradeoffs were among them. Understanding the tradeoffs among the options is quite important if we want to build the best system possible."

She also worked on a project that developed small self-explanatory electronic blocks that people could connect together to build useful electronic systems involving almost any kind of sensor, like motion or light sensors. "Those blocks could be used by kids to learn basic concepts of logic and computers, concepts which are quite important to learn these days. Our hope is that these blocks will be used as teaching tools in schools. The blocks can also be used to help adults set up useful systems in their homes, perhaps to monitor an aging parent, or a child at home sick. The potential for these blocks is great—it will be interesting to see what impact they have.

"My favorite thing about engineering is the variety of skills and creativity involved. We are faced with problems that need to be solved, and we solve them by applying known techniques in creative ways. Engineers must continually learn new technologies, hear new ideas, and track current products, in order to be good designers. It's all very exciting and challenging. Each day at work is different. Each day is exciting and is a learning experience.

"Studying to be an engineer can be a great deal of work, but it's worth it. The key is to stay focused, to keep your mind open, and to make good use of available resources. Staying focused means to keep your priorities in order—for example, as a student, studying comes first, recreation second. Keeping your mind open means to always be willing to listen to different ideas and to learn about new technologies. Making good use of resources means to aggressively seek information, from the Internet, from colleagues, from books, and so on. You never know where you are going to get your next important bit of information, and you won't get that information unless you seek it."