We hope that our discussion in this section has helped you to see how digital logic and error correction algorithms fit together. The same can be done with any algorithm that can be represented using one of the finite state machines described. In fact, the convolutional code just described is also referred to as a (2, 1) convolutional code because two symbols are output for every one symbol input. Other convolutional codes provide somewhat deeper error correction, but they are too complex for economical hardware implementation.

3.7 DESIGNING CIRCUITS

In the preceding sections, we introduced many different components used in computer systems. We have, by no means, provided enough detail to allow you to start designing circuits or systems. Digital logic design requires someone not only familiar with digital logic, but also well versed in **digital analysis** (analyzing the relationship between inputs and outputs), **digital synthesis** (starting with a truth table and determining the logic diagram to implement the given logic function), and the use of computer-aided design (CAD) software. Recall from our previous discussions that great care needs to be taken when designing the circuits to ensure that they are minimized. A circuit designer faces many problems, including finding efficient Boolean functions, using the smallest number of gates, using an inexpensive combination of gates, organizing the gates of a circuit board to use the smallest surface area and minimal power requirements, and attempting to do all of this using a standard set of modules for implementation. Add to this the many problems we have not discussed, such as signal propagation, fan out, synchronization issues, and external interfacing, and you can see that digital circuit design is quite complicated.

Up to this point, we have discussed how to design registers, counters, memory, and various other digital building blocks. Given these components, a circuit designer can implement any given algorithm in hardware (recall the Principle of Equivalence of Hardware and Software from Chapter 1). When you write a program, you are specifying a sequence of Boolean expressions. Typically, it is much easier to write a program than it is to design the hardware necessary to implement the algorithm. However, there are situations in which the hardware implementation is better (e.g., in a real-time system, the hardware implementation is faster, and faster is definitely better.) However, there are also cases in which a software implementation is better. It is often desirable to replace a large number of digital components with a single programmed microcomputer chip, resulting in an **embedded system**. Your microwave oven and your car most likely contain embedded systems. This is done to replace additional hardware that could present mechanical problems. Programming these embedded systems requires design software that can read input variables and send output signals to perform such tasks as turning a light on or off, emitting a beep, sounding an alarm, or opening a door. Writing this software requires an understanding of how Boolean functions behave.

CHAPTER SUMMARY

The main purpose of this chapter is to acquaint you with the basic concepts involved in logic design and to give you a general understanding of the basic circuit configurations used to construct computer systems. This level of familiarity will not enable you to design these components; rather, it gives you a much better understanding of the architectural concepts discussed in the following chapters.

In this chapter we examined the behaviors of the standard logical operators AND, OR, and NOT and looked at the logic gates that implement them. Any Boolean function can be represented as a truth table, which can then be transformed into a logic diagram, indicating the components necessary to implement the digital circuit for that function. Thus, truth tables provide us with a means to express the characteristics of Boolean functions as well as logic circuits. In practice, these simple logic circuits are combined to create components such as adders, ALUs, decoders, multiplexers, registers, and memory.

There is a one-to-one correspondence between a Boolean function and its digital representation. Boolean identities can be used to reduce Boolean expressions, and thus, to minimize both combinational and sequential circuits. Minimization is extremely important in circuit design. From a chip designer's point of view, the two most important factors are speed and cost: minimizing circuits helps to both lower the cost and increase performance.

Digital logic is divided into two categories: combinational logic and sequential logic. Combinational logic devices, such as adders, decoders, and multiplexers, produce outputs that are based strictly on the current inputs. The AND, OR, and NOT gates are the building blocks for combinational logic circuits, although universal gates, such as NAND and NOR, could also be used. Sequential logic devices, such as registers, counters, and memory, produce outputs based on the combination of current inputs and the current state of the circuit. These circuits are built using SR, D, and JK flip-flops.

You have seen that sequential circuits can be represented in a number of different ways, depending on the particular behavior that we want to emphasize. Clear pictures can be rendered by Moore, Mealy, and algorithmic state machines. A lattice diagram expresses transitions as a function of time. These finite state machines differ from DFAs in that, unlike DFAs, they have no final state because circuits produce output rather than accept strings.

These logic circuits are the building blocks necessary for computer systems. In Chapter 4 we put these blocks together and take a closer, more detailed look at how a computer actually functions.

If you are interested in learning more about Kmaps, there is a special section that focuses on Kmaps located at the end of this chapter, after the exercises.

FURTHER READING

Most computer organization and architecture books have a brief discussion of digital logic and Boolean algebra. The books by Stallings (2006), Tanenbaum (2006),

and Patterson and Hennessy (2005) contain good synopses of digital logic. Mano (1993) presents a good discussion on using Kmaps for circuit simplification (discussed in the focus section of this chapter) and programmable logic devices, as well as an introduction to the various circuit technologies. For more in-depth information on digital logic, see the Wakerly (2000), Katz (1994), or Hayes (1993) books.

Martin Davis (2000) traces the history of computer theory, including biographies of all the seminal thinkers, in his *Universal Computer* book. This book is a joy to read. For a good discussion of Boolean algebra in lay terms, check out the book by Gregg (1998). The book by Maxfield (1995) is an absolute delight to read and contains informative and sophisticated concepts on Boolean logic, as well as a trove of interesting and enlightening bits of trivia (including a wonderful recipe for seafood gumbo!). For a very straightforward and easy book to read on gates and flip-flops (as well as a terrific explanation of what computers are and how they work), see the book by Petgold (1989). Davidson (1979) presents a method of decomposing NAND-based circuits (of interest because NAND is a universal gate).

Moore, Mealy, and algorithmic state machines were first proposed in papers by Moore (1956), Mealy (1955), and Clare (1973). Daniel Cohen's (1991) computer theory book is one of the most easily understandable on this topic. In it you will find excellent presentations of Moore, Mealy, and finite state machines in general, including DFAs. Forney's (1973) well-written tutorial on the Viterbi algorithm in a paper by that same name explains the concept and the mathematics behind this convolutional decoder. Fischer's 1996 article explains how PRML is used in disk drives.

If you are interested in actually designing some circuits, there are several nice simulators freely available. One set of tools is called the Chipmunk System. It performs a wide variety of applications, including electronic circuit simulation, graphics editing, and curve plotting. It contains four main tools, but for circuit simulation, *Log* is the program you need. The *Diglog* portion of Log allows you to create and actually test digital circuits. If you are interested in downloading the program and running it on your machine, the general Chipmunk distribution can be found at www.cs.berkeley.edu/~lazzaro/chipmunk/. The distribution is available for a wide variety of platforms (including PCs and Unix machines).

Another very nice package is Multimedia Logic (MMLogic) by Softronix, but it is currently available for Windows platforms only. This fully functional package has a very nice GUI with drag and drop components and comprehensive on-line help. It includes not only the standard complement of devices (such as ANDs, ORs, NANDs, NORs, adders, and counters), but also special multimedia devices (including bitmap, robot, network, and buzzer devices). You can create logic circuits and interface them to real devices (keyboards, screens, serial ports, etc.) or even other computers. The package is advertised for use by beginners, but allows users to build quite complex applications (such as games that run over the Internet). MMLogic can be found at: www.softronix.com/logic.html, and the distribution includes not only the executable package, but also the source code so users can modify or extend its capabilities.

Either of these simulators can be used to build the MARIE architecture discussed next in Chapter 4.