Chapter 1

Introduction

In this chapter, we discuss

- What is a digital system and how it differs from an analog system.
- Why are digital systems important and where are they used.
- The basic types of digital systems: combinational and sequential.
- The specification and implementation of digital systems.
- The processes of analysis and design.
- The use of CAD tools.

1.1 About digital sytems

What is a digital system?

A **digital system** is a system in which signals have a **finite** number of **discrete** values. This contrasts with **analog systems,** in which signals have values from a continuous (infinite) set. As an elementary example, a digital scale measures weight through discrete signals indicating pounds and ounces (or kilograms and grams); on the other hand, an analog scale measures weight through a continuous signal corresponding to the position of a pointer (the hand) over a scale.

Analog and digital signals are illustrated in Figure 1.1. In the digital case, time may also be discretized (as shown in Figure 1.1c), so that signals may change only at discrete instants; these instants are labeled with integer values. Systems with this type of digital signals are called **synchronous**, whereas those in which changes may occur at any instant are called **asynchronous.** In this text, we focus on synchronous systems, because they are more widely

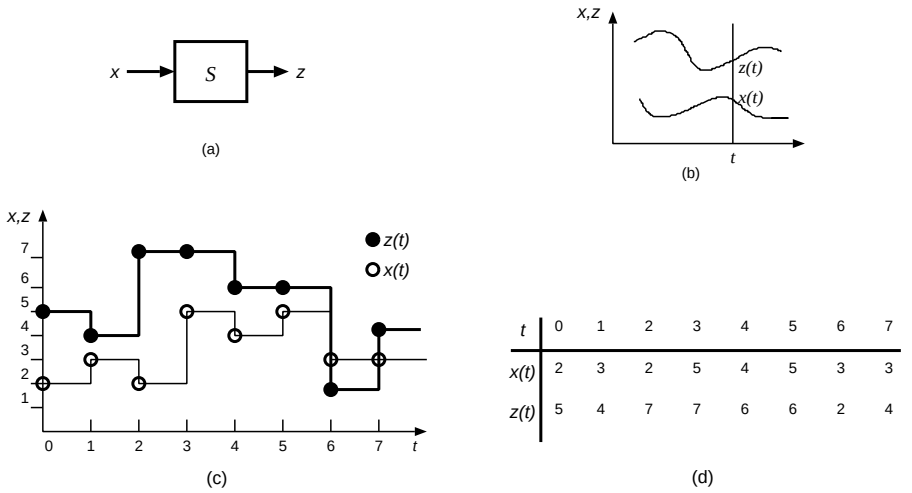

Figure 1.1: a) Block representation of system S. b) Analog input and output signals. c) Synchronous digital input and output signals. d) Pair of input-output sequences.

used. Figure 1.1d illustrates the representation of digital signals by sequences of values.

Why are digital systems important?

Digital systems are used in information processing (also called data processing and signal processing, depending on the specific application), wherein they have become prevalent and have displaced the earlier analog systems. Some of the benefits of digital systems are:

1. Digital representation is well suited for both numerical and nonnumerical information processing. An example of digital nonnumeric information is the written language in which the letters have values from the finite alphabet A,B,C..., and so on.

2. Information processing can use a general-purpose system (a **computer**) which is programmed for a particular processing task, eliminating the need to have a different system for each task. The representation of the program and its interpretation can use the same techniques employed for the representation and transformation of the data.

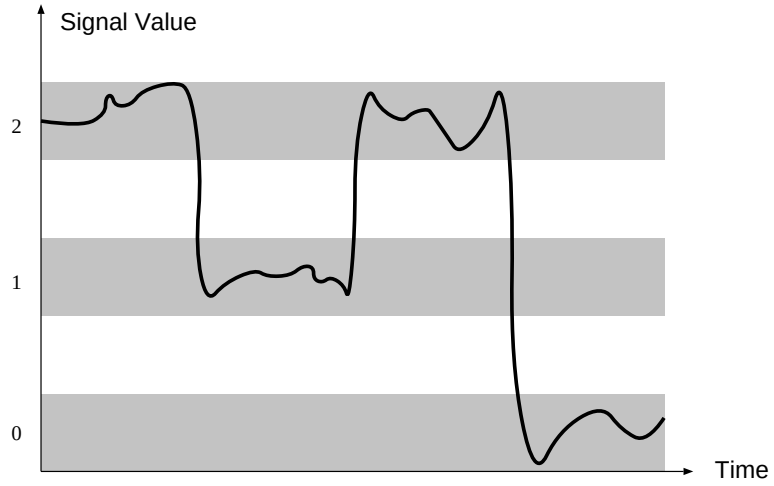

Figure 1.2: Separation of digital signal values.

3. The finite number of values in a digital signal can be represented by a vector of signals with just two values (**binary signals**). For example, the ten values of a decimal digit can be represented by a vector of four binary signals as follows:

digit	0	1	2	3	4	5	6	7	8	9
vector	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

   This representation allows implementations in which all signals are binary; as a result, the devices which process these signals are very simple (essentially, just switches with two states: open and closed).

4. Digital signals are quite insensitive to variations of component parameter values (such as operating temperature). As illustrated in Figure 1.2, the physical representation of the values are sufficiently separated so that small variations do not change the value. This is especially true for binary signals.

5. Numerical digital systems can be made more accurate by simply increasing the number of digits used in the representation. For example, the time of day can be represented with a low-precision signal as 12:35 (hours and minutes), or more precisely as 12:35:07 (hours, minutes, and seconds).

6. The advances of microelectronics technology in recent years have made possible the fabrication of extremely complex digital systems, which are

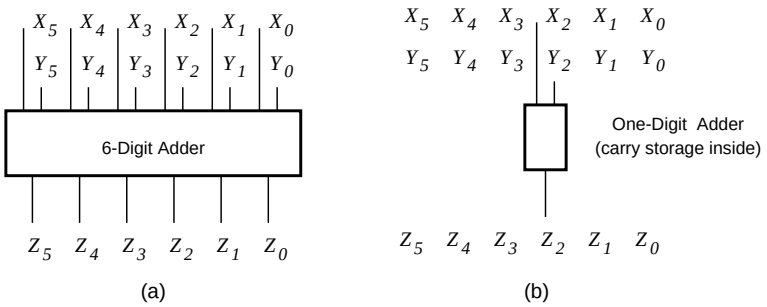

Figure 1.3: Six-digit adder: a) parallel implementation; b) serial implementation.

small, fast, and cheap. Complex digital systems are built as **integrated circuits** composed of a large number of very simple devices.

7. It is possible to select among different implementations of systems which trade-off speed and amount of hardware. For example, if we want to add two integers represented by six decimal digits, we can choose among a parallel implementation (see Figure 1.3a) which simultaneously adds all six digits, or a serial implementation (Figure 1.3b) in which the digits of the result are obtained in sequence. In the parallel case, the system is faster but uses more hardware (a six-digit adder), whereas in the serial case it is slower but uses six times a one-digit adder.

When are digital systems used?

Digital representation and processing methods have been used for a long time. The adding machine and typing machine are just two examples. The development of digital artifacts underwent a dramatic increase with the invention of the digital computer circa 1940. According to today's standards, at that time computers were few, very expensive, not very powerful, not very reliable, and hard to program and use. Since then, extraordinary progress has been made in all these aspects, making the computer indispensable in almost every aspect of modern society.

The development of computer technology, and digital microelectronics in particular, has made possible the cost-effective production of a large variety of specialized digital systems. Some examples are digital watches and timers, calculators, instruments, controllers, video games, cameras, locks, communication equipment, digital music recording, and digital video recording. This trend continues as new applications for digital systems are constantly developed; in some cases the new applications are replacing analog systems, but in many oth-

ers they are making possible applications that did not previously exist. As a consequence, knowledge about the design and use of digital systems is required in a large variety of human activities.

Analog and digital signals

Since signals in the physical world are analog, it is necessary to convert from an analog signal to a digital signal and vice versa whenever digital systems have to interact with these physical signals. As an example, consider a system for processing voice signals produced by a singer (see Figure 1.4a). Analog signals are generated and transmitted through the air, until a microphone converts them into electric signals (still analog). Then, an analog-to-digital converter transforms the signals to digital, and these digital signals are processed and stored. To transmit the resulting signals to an audience, they are converted to analog and then applied to a speaker system.

The process for converting from analog to digital is called **quantization** or **digitization.** For example, the analog signal in Figure 1.4b is sampled at periodic intervals and quantized into four levels, and then the digital representation is conveyed by the sequence of digits. The accuracy with which the conversion is done depends on the number of levels and on the frequency of the samples (this singer has unusually limited voice!).

Combinational and sequential systems

Digital systems are divided into two classes: **combinational systems** and **sequential systems.** In combinational systems, the output at time t depends **only** on the input at time t (see Figure 1.5a). That is,

$$z(t) = F(x(t))$$

In this case, we can say that the system has no **memory** because the output does not depend on previous inputs. In sequential systems, on the other hand, the output at time t depends on the input at time t and possibly also depends on the input at time prior to t (as shown in Figure 1.5b). This can be written for the general case as

$$z(t) = F(x(0,t))$$

wherein $x(0,t)$ is the input sequence from time 0 to time t.

Example 1.1 and Example 1.2 illustrate a sequential system and a combinational system, respectively.

Example 1.1 A digital system S has an input x with values 0,1, or 2, and an output z with values 0 or 1. The function is defined as follows:

$$z(t) = \begin{cases} 1 & \textbf{if} \text{ the input sequence } (x(0), x(1), \ldots, x(t)) \text{ has} \\ & \quad \text{an even number of } 2'\text{s and an odd number of } 1'\text{s} \\ 0 & \textbf{otherwise} \end{cases}$$

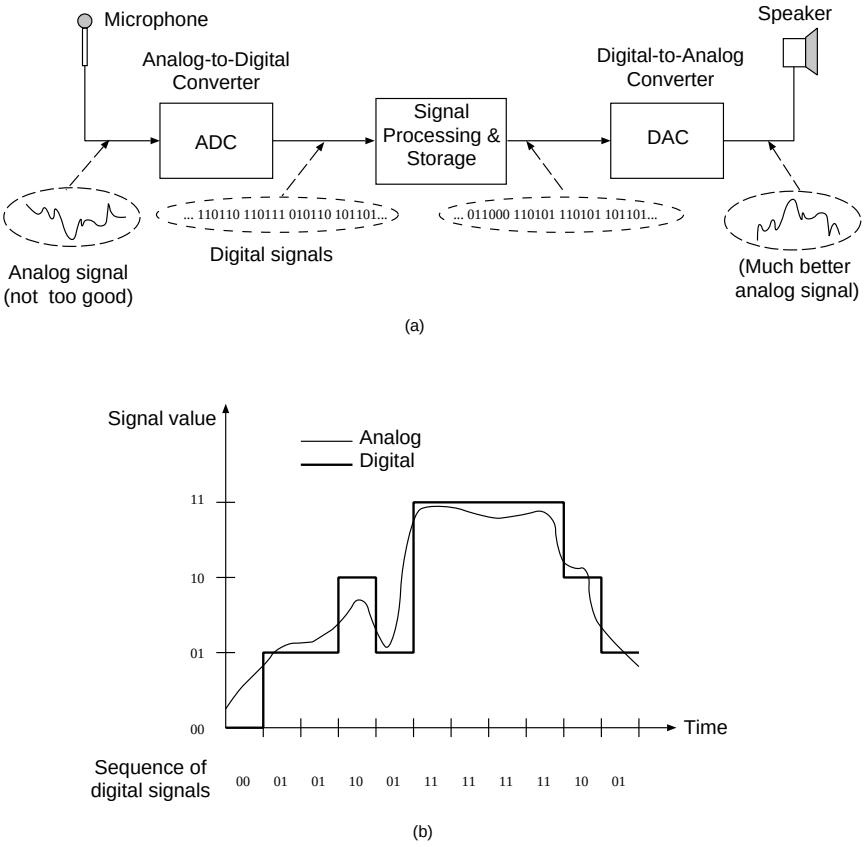

Figure 1.4: a) A system with analog and digital signals. b) Analog-to-digital conversion.

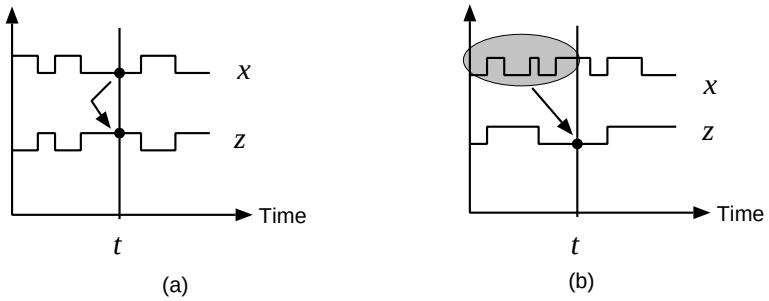

Figure 1.5: Input-output functions for: a) a combinational system; b) a sequential system.

This system is sequential because the output at time t depends on previous inputs. The function description makes it possible to determine the output sequence for a given input sequence. For example, the following is a pair of input-output sequences:

t	0	1	2	3	4	5	6	7	8	9	10	11
x	1	2	2	0	1	2	0	0	0	2	1	1
z	1	0	1	1	0	0	0	0	0	0	1	0

■

Example 1.2 A digital system has two input signals. Input $x(t)$ has values from the set of letters (upper and lower case) and input $y(t)$ has values 0 and 1. The function of the system is to change $x(t)$ to opposite case when $y(t) = 1$ and leave it unchanged when $y(t) = 0$. This is a combinational system because the output at time t depends only on the input at time t. An example of a pair of input-output sequences is

t	0	1	2	3	4	5	6
x	E	X	A	M	P	L	E
y	0	1	0	0	0	1	0
z	E	x	A	M	P	l	E

■

The essential difference between combinational and sequential systems is apparent by looking at the output functions from Examples 1.1 and 1.2. In the first example, the output might have different values even if inputs are identical because the output depends on the previous input values (see, for example, the input/output values for $t = 2$ and $t = 5$). In the second example, on the other hand, the output is the same whenever inputs are the same.

This classification is useful in the study of digital systems, because the simpler combinational systems can be studied first.

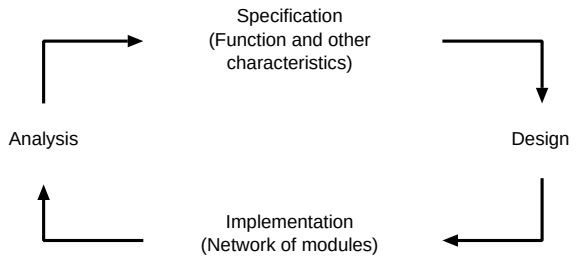

Figure 1.6: Relationship among system specification and implementation.

1.2 Specification and implementation, analysis and design.

The study of any system involves its specification and implementation (see Figure 1.6). The **specification** of a system refers to a description of its function and of other characteristics required for its use, such as speed, technology, and power consumption. Specification is related to **what** the system does without reference to **how** it performs the operation. A specification should be as complete and as simple as possible; all required details must be included, but no irrelevant ones. Moreover, a specification should be formal in the sense that its interpretation is unambiguous.

The specification of a system must describe its function in a way that is adequate for two purposes:

- to use the system as a component in more complex systems; and
- to serve as the basis for the implementation of the system by a network of simpler components.

Several specification methods for digital systems are presented in this text. The most appropriate method for a given system depends on the complexity of the system and on the intended use of the specification. In Chapter 2 we introduce high-level and binary-level specification of combinational systems. Specification of sequential systems is discussed in Chapter 6. Specification of complex digital systems which require a register-transfer model is the subject of Chapter 12.

On the other hand, an **implementation** of a system refers to how the system is constructed from simpler components. In the case of digital systems, the implementation is a **digital network** that consists of the interconnection of digital modules. This network can be defined at several levels depending on the complexity of the primitive modules used, which can range from very simple **gates** to complex **processors.**

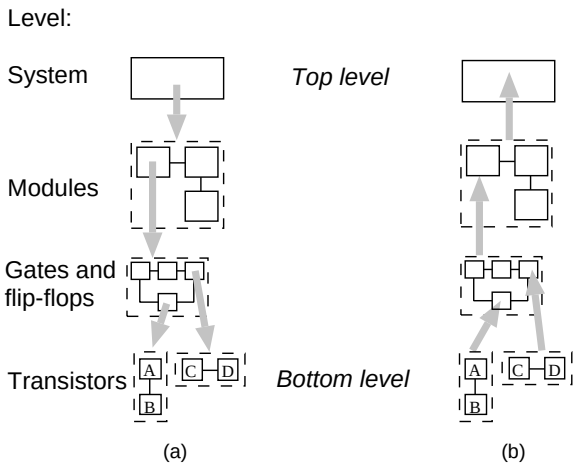

Figure 1.7: Hierarchical implementation: a) a top-down approach; b) a bottom-up approach.

At the physical level, all digital systems are implemented by a complex interconnection of elementary electronic elements such as transistors, resistors, and so on (these elements are described in Chapter 3). Therefore, the implementation of any digital system could be described as an electronic circuit by indicating these components and their connections. This is impractical, however, because of the complexity of most digital systems. It would be like describing the human body by giving the characteristics of each cell and their relative positions. It is necessary, therefore, to define intermediate levels of modules of increasing complexity, whose description includes only the characteristics that are relevant for their use as components in a more complex system (see Figure 1.7). This corresponds to a **hierarchical implementation**.

Modules are more than just conceptual entities used to simplify the description of an implementation. They are most often designed and built separately and then assembled together to form the final system. Moreover, there exist standard modules of several levels of complexity that can be used in the design of large numbers of different systems. This fact has been important in the cost-effectiveness of digital systems.

The interconnection of modules in an implementation has to follow certain rules for the network to perform adequately. These rules are, in general, specified independently from the definition of the modules, so that they can be used with any type of module at all levels of the implementation.

In this text, we emphasize the separation of the specification of a system

from its implementation. This distinction is very important in complex systems because

- it shields the description required to use the system from irrelevant implementation details; and
- it allows choosing an implementation from different alternatives, without influencing the description required for using the system.

The implementation of combinational systems at the gate level is discussed in Chapters 4 and 5, and at the module level in Chapters 8, 9, and 11. The implementation of sequential systems is the subject of two chapters: implementation of elementary sequential systems is discussed in Chapter 7, whereas implementations of more complex sequential modules is covered in Chapters 10 and 11. Implementation of register-transfer level systems is presented in Chapters 13 and 14.

Structured analysis and design

The **analysis** of a system has as objective the determination of its specification from an implementation (see Figure 1.6). The system thus analyzed can be a module in a larger system, resulting in a multilevel analysis process. For complex systems, this multilevel approach is indispensable to have a manageable analysis.

On the other hand, the **design** process consists of obtaining an implementation that satisfies the specification of a system (see Figure 1.6). If the system is complex, it is also necessary to use a multilevel approach; two variants of this approach have been proposed:

The top-down approach, illustrated in Figure 1.7a, decomposes the system into subsystems that themselves are decomposed into simpler subsystems, until a level is reached at which the subsystems can be realized directly with available modules.

This method has the disadvantage that no systematic procedure exists to assure that the decomposition at a particular level optimizes the final implementation. The success of the approach depends on the experience of the designer to choose an adequate decomposition at each level. (An inadequate decomposition in the top-down approach leads to a "bottomless implementation," as one of our colleagues remarked once.)

The bottom-up approach, illustrated in Figure 1.7b, connects available modules to form subsystems, and these subsystems are connected to other subsystems until the required functional specification is fulfilled.

This approach has a similar disadvantage to that discussed for the top-down case. The composition of subsystems should be done in a way that

results in the correct system specification. Again, in general, there is no systematic procedure which assures that this will occur.

Consequently, a combination of the two approaches should be used: the system is decomposed into subsystems (top-down) but the specific decomposition depends on which subsystems can be cost-effectively composed from primitive modules (bottom-up).

In this book, analysis and design are discussed at several levels. In the first eleven chapters we consider combinational and sequential systems; we develop a bottom-up approach, in which we begin with the use of primitive building blocks, namely gates and flip-flops, and then introduce standard modules, such as decoders, adders, and counters. Then, in the last three chapters we present, in a top-down manner, the implementation of a register-transfer level system consisting of a data subsystem and a control subsystem, each of which is composed of the modules studied in earlier chapters.

Levels of an implementation: module, logical, physical

The implementation of a system may be described at different levels. To illustrate these levels, let's consider the system depicted in Figure 1.8, which computes the sum of its inputs, that is,

$$Z(t) = \sum_{i=0}^{t} X(i)$$

At the **module level,** the system consists of two registers and an adder (Figure 1.8a). At the **logical level**, as illustrated in Figure 1.8b, these modules are implemented with **gates** and **flip-flops** (elementary combinational and sequential components, respectively), and signals are binary (have two values, called 0 and 1). Components of these types are then connected to form networks that implement more complex functions.

Underneath the logical level is the **physical level,** in which the components are realized in some technology. For the most part, digital systems are electronic systems, although there are other technologies that are sometimes used, resulting in mechanical systems, hydraulic systems, optical systems, biochemical systems, etc.

In the electronic technology, as shown in Figure 1.8c, signals are electric signals (voltages, currents, charges) and the basic components are transistors (electrically controlled switches) and power supplies. Gates and flip-flops are formed by connecting several of these transistors; networks of gates and flip-flops are realized by connecting the components with some conducting material.

It would be possible to study digital systems completely at the logical level. However, this could lead to systems that might not have the desired characteristics when actually built. Consequently, in the study at the logical level, it is

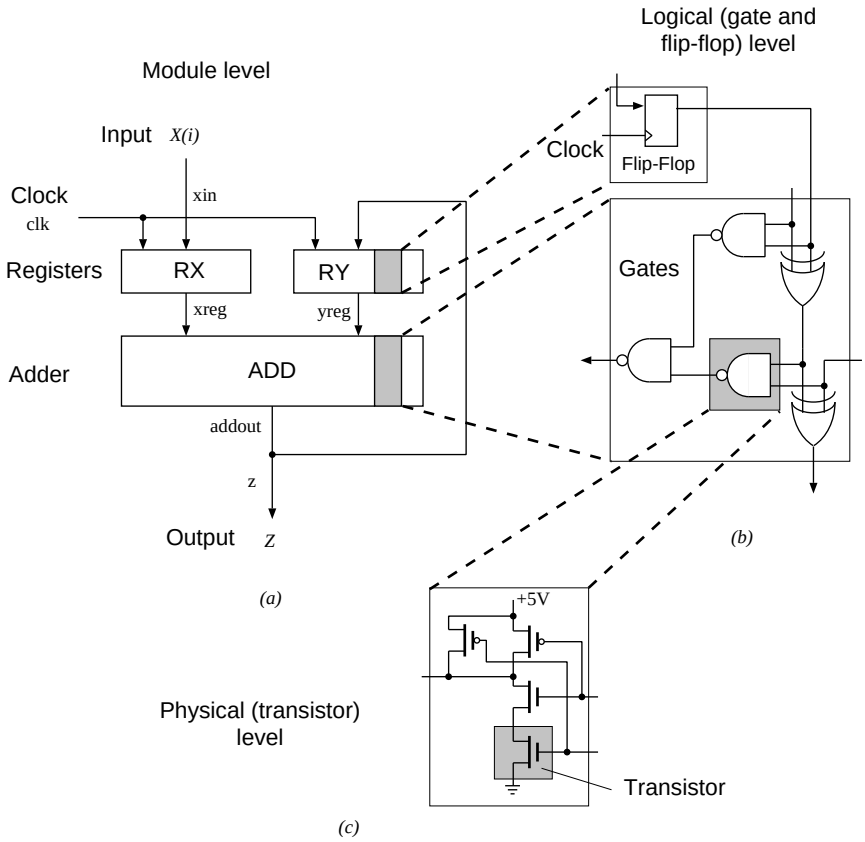

Figure 1.8: Digital system: a) module level; b) logical level; and c) physical level.

necessary to incorporate characteristics that are abstracted from the physical level. For example, if we are designing a system that will be realized in the popular CMOS technology, we need to know what components are effectively realized in this technology (type of gates, number of inputs, speed, area, etc.). Such aspects are addressed in this book.

1.3 Computer-aided design tools

The design of digital systems is an involved and laborious process. Various computer-aided design (CAD) tools are available to help making this process efficient, timely and economical. We now briefly overview the subject of CAD tools which, in its details, is beyond the scope of this textbook.

CAD tools are intended to support all phases of digital design: (i) description (specification), (ii) design (synthesis), including various optimizations to reduce cost and improve performance, and (iii) verification (by simulation or formally) of the design with respect to its specification. These three phases typically require several passes to obtain a suitable implementation.

Description of digital systems is performed in a hierarchical manner, as discussed in the previous section. The "traditional" way to describe digital systems consists of a description of its structure through a graphical form (a drawing), as depicted in Figure 1.8. This description provides a **logic diagram** of the system at different levels, showing the modules and their interconnections. Such diagrams used to be drawn manually, but nowadays there are tools which allow generating and editing these drawings on a computer; this process is called **schematic capture** because the tool is used to capture the **schematic** description of the digital system. The process is supported by **libraries** of standard components, so a system can be "built" by using standard parts that are placed together to compose an implementation.

An alternative approach, which is becoming widely accepted and is replacing schematic capture, is the use of a **hardware-description language** (HDL). Several languages of this type have been proposed in the past, with the recent standardization of two of them: Verilog and VHDL. In this text, we use a subset of VHDL (which we call μVHDL) for the description of systems. Figure 1.9 gives a structural μVHDL description of the system depicted in Figure 1.8a, assuming that additional descriptions exist for the modules used in it (`BitReg8, Adder`); such lower level modules can be described in terms of the expected behavior rather than as the interconnection of even lower level components.

Note that generating this description only requires a text editor in a computer, though there are tools which allow other types of input such as a drawing of the description at the top-most level.

```
USE WORK.ALL;
ENTITY sample_system IS
  PORT (xin: IN  BIT_VECTOR;
        z  : OUT BIT_VECTOR;
        clk: IN  BIT      );
END sample_system;

ARCHITECTURE structural OF sample_system IS
  SIGNAL xreg, yreg, addout: BIT_VECTOR(7 DOWNTO 0);

BEGIN
  RX: ENTITY BitReg8 PORT MAP(xin,xreg,clk);
  RY: ENTITY BitReg8 PORT MAP(addout,yreg,clk);
 ADD: ENTITY Adder   PORT MAP(xreg,yreg,addout);
      z <= addout;
END structural;
```

Figure 1.9: μVHDL-based description of a system.

We should note that these two description approaches can co-exist; for example, there are tools which convert from a HDL description to a logic diagram, as well as tools that generate VHDL code from a schematic diagram.

Logic diagrams and HDL descriptions also provide information such as module types, signal names, wire type and pin numbers, which are needed in later phases of a design.

Synthesis and optimization tools help in obtaining an implementation from a given description, and in improving some characteristics such as the number of modules and the network delays. These tools use transformation techniques such as the ones discussed in this text.

Simulation tools are used to verify the operation of the system. These tools use the description of the system to produce the values of the signals (internal and external) for a given input. The simulation is used to detect errors in a design and to determine characteristics, such as delay and power consumption, which are hard to obtain analytically. Evidently, the accuracy of the simulation depends on the accuracy of the model used to describe the components.

In addition to tools for the design at the logical level, there are others for the physical implementation of chips and boards, such as for VLSI layout, printed-circuit board layout and routing, thermal and mechanical analysis of chips and boards, and so on.

1.4 Further readings

The literature on digital systems, computers in particular, and their applications is readily available as books, technical magazines, and popular articles. For a comprehensive introduction to the basic aspects of computing systems (hardware, software, and applications) we suggest *The Mystical Machine*, by J. E. Savage, S. Magidson, and A.M. Stein, Addison-Wesley, Reading, Mass. 1986. A popular book on background topics in electronics is *Intuitive Digital Computer Basics* by T. Fredericksen, McGraw-Hill, New York, NY, 1988. For current reviews of the state-of-the-art in digital technologies, design tools, and applications, a good source is the January issue of *IEEE Spectrum* published by the Institute of Electrical and Electronics Engineers, New York, NY.