CHAPTER 1

Introduction to Digital Design Methodology

Classical design methods relied on schematics and manual methods to design a circuit, but today computer-based languages are widely used to design circuits of enormous size and complexity. There are several reasons for this shift in practice. No team of engineers can correctly design and manage, by manual methods, the details of state-of-the-art integrated circuits (ICs) containing several million gates, but using hardware description languages (HDLs) designers easily manage the complexity of large designs. Even small designs rely on language-based descriptions, because designers have to quickly produce correct designs targeted for an ever-shrinking window of opportunity in the marketplace.

Language-based designs are portable and independent of technology, allowing design teams to modify and re-use designs to keep pace with improvements in technology. As physical dimensions of devices shrink, denser circuits with better performance can be synthesized from an original HDL-based model.

HDLs are a convenient medium for integrating intellectual property (IP) from a variety of sources with a proprietary design. By relying on a common design language, models can be integrated for testing and synthesized separately or together, with a net reduction in time for the design cycle. Some simulators also support mixed descriptions based on multiple languages.

The most significant gain that results from the use of an HDL is that a working circuit can be synthesized automatically from a language-based description, bypassing the laborious steps that characterize manual design methods (e.g., logic minimization with Karnaugh maps).

HDL-based synthesis is now the dominant design paradigm used by industry. Today, designers build a software prototype/model of the design, verify its functionality, and then use a synthesis tool to automatically optimize the circuit and create a netlist in a physical technology.

HDLs and synthesis tools focus an engineer's attention on functionality rather than on individual transistors or gates; they synthesize a circuit that will realize the desired functionality, and satisfy area and/or performance constraints. Moreover, alternative architectures can be generated from a single HDL model and evaluated quickly to perform design tradeoffs. Functional models are also referred to as behavioral models.

HDLs serve as a platform for several tools: design entry, design verification, test generation, fault analysis and simulation, timing analysis and/or verification, synthesis, and automatic generation of schematics. This breadth of use improves the efficiency of the design flow by eliminating translations of design descriptions as the design moves through the tool chain.

Two languages enjoy widespread industry support: Verilog™ [1] and VHDL [2]. Both languages are IEEE (Institute of Electrical and Electronics Engineers) standards; both are supported by synthesis tools for ASICs (application-specific integrated circuits) and FPGAs (field-programmable gate arrays). Languages for analog circuit design, such as Spice [3], play an important role in verifying critical timing paths of a circuit, but these languages impose a prohibitive computational burden on large designs, cannot support abstract styles of design, and become impractical when used on a large scale. Hybrid languages (e.g., Verilog-A) [4] are used in designing mixed-signal circuits, which have both digital and analog circuitry. System-level design languages, such as SystemC [5] and Superlog™ [6], are now emerging to support a higher level of design abstraction than can be supported by Verilog or VHDL.

1.1 Design Methodology—An Introduction

ASICs and FPGAs are designed systematically to maximize the likelihood that a design will be correct and will be fabricated without fatal flaws. Designers follow a "design flow" like that shown in Figure 1-1, which specifies a sequence of major steps that will be taken to design, verify, synthesize, and test a digital circuit. ASIC design flows involve several activities, from specification and design entry, to place-and-route and timing closure of the circuit in silicon. Timing closure is attained when all of the signal paths in the design satisfy the timing constraints imposed by the interface circuitry, the circuit's sequential elements, and the system clock. Although the design flow appears to be linear, in practice it is not. Various steps might be revisited as design errors are discovered, requirements change, or performance and design constraints are violated. For example, if a circuit fails to meet timing constraints, a new placement and routing step will have to be taken, perhaps including redesign of critical paths.

Design flows for standard-cell-based ASICs are more complex than those for FPGAs because the architecture of an ASIC is not fixed. Consequently, the performance that can be realized from a design depends on the physical placement and routing of the cells on the die, as well as the underlying device properties. Interconnect delays play a significant role in determining performance in submicron designs below 0.18 μm, in which prelayout estimates of path delays do not guarantee timing closure of the routed design.

The following sections will clarify the design flow described in Figure 1-1.

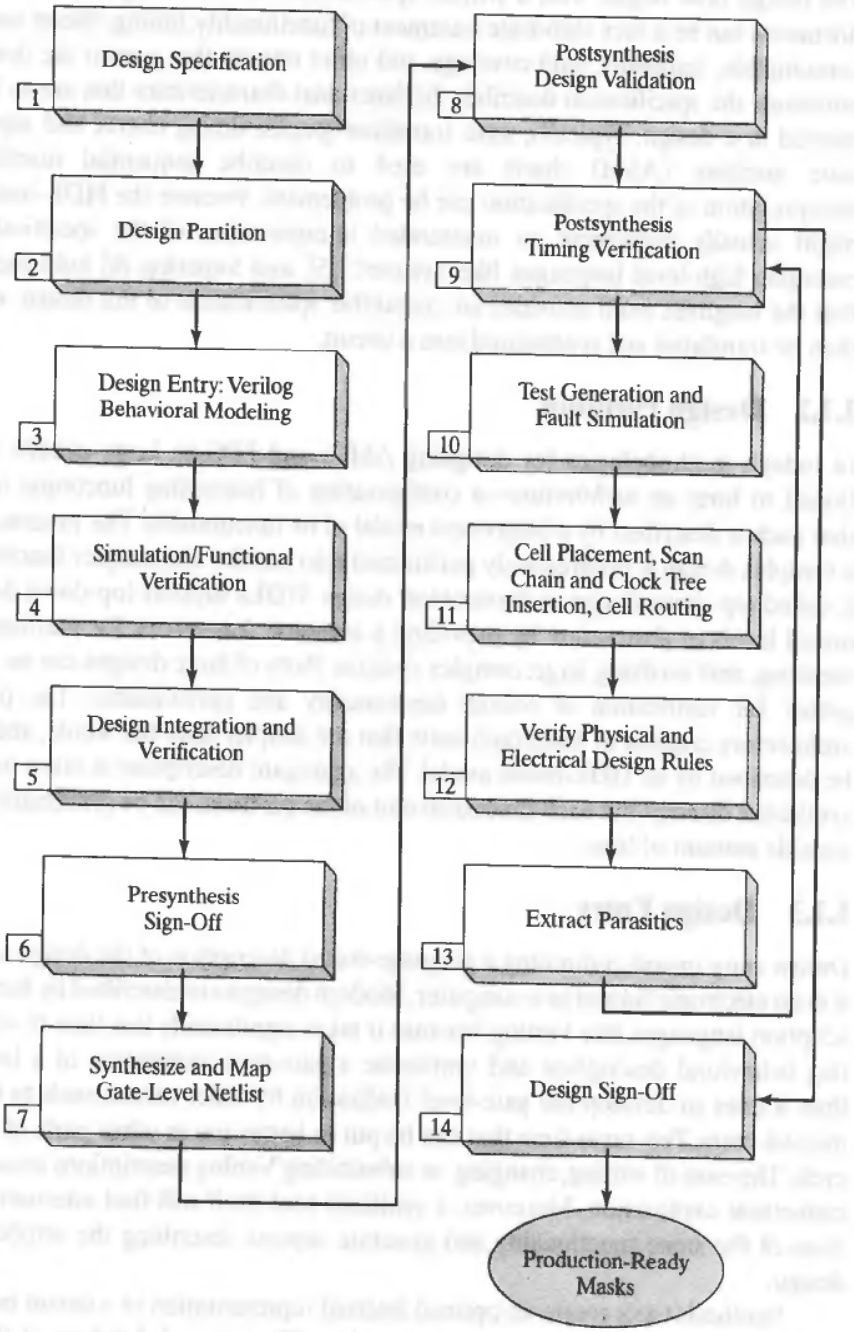

FIGURE 1-1 Design flow for HDL-based ASICs.

1.1.1 Design Specification

The design flow begins with a written specification for the design. The specification document can be a very elaborate statement of functionality, timing, silicon area, power consumption, testability, fault coverage, and other criteria that govern the design. At a minimum, the specification describes the functional characteristics that are to be implemented in a design. Typically, state transition graphs, timing charts, and algorithmic-state machine (ASM) charts are used to describe sequential machines, but interpretation of the specification can be problematic, because the HDL-based model might actually implement an unintended interpretation of the specification. The emerging high-level languages, like SystemC [5], and Superlog [6] hold the promise that the language itself provides an executable specification of the design, which can then be translated and synthesized into a circuit.

1.1.2 Design Partition

In today's methodologies for designing ASICs and FPGAs, large circuits are partitioned to form an *architecture*—a configuration of interacting functional units, such that each is described by a behavioral model of its functionality. The process by which a complex design is progressively partitioned into smaller and simpler functional units is called *top-down design* or *hierarchical design*. HDLs support top-down design with mixed levels of abstraction by providing a common framework for partitioning, synthesizing, and verifying large, complex systems. Parts of large designs can be linked together for verification of overall functionality and performance. The partitioned architecture consists of functional units that are simpler than the whole, and each can be described by an HDL-based model. The aggregate description is often too large to synthesize directly, but each functional unit of the partition can be synthesized in a reasonable amount of time.

1.1.3 Design Entry

Design entry means composing a language-based description of the design and storing it in an electronic format in a computer. Modern designs are described by hardware description languages, like Verilog, because it takes significantly less time to write a Verilog behavioral description and synthesize a gate-level realization of a large circuit than it does to develop the gate-level realization by other means, such as bottom-up manual entry. This saves time that can be put to better use in other parts of the design cycle. The ease of writing, changing, or substituting Verilog descriptions encourages architectural exploration. Moreover, a synthesis tool itself will find alternative realizations of the same functionality and generate reports describing the attributes of the design.

Synthesis tools create an optimal internal representation of a circuit before mapping the description into the target technology. The internal database at this stage is generic, which allows it to be mapped into a variety of technologies. For example, the technology mapping engine of a synthesis tool will use the internal format to migrate a design from an FPGA technology to an ASIC standard cell library, without having to reoptimize the generic description.

HDL-based designs are easier to debug than schematics. A behavioral description encapsulating complex functionality hides underlying gate-level detail, so there is less information to cope with in trying to isolate problems in the functionality of the design. Furthermore, if the behavioral description is functionally correct, it is a gold standard for subsequent gate-level realizations.

HDL-based designs incorporate documentation within the design by using descriptive names, by including comments to clarify intent, and by explicitly specifying architectural relationships, thereby reducing the volume of documentation that must be kept in other archives. Simulation of a language-based model explicitly specifies the functionality of the design. Since the language is a standard, documentation of a design can be decoupled from a particular vendor's tools.

Behavioral modeling is the predominant descriptive style used by industry, enabling the design of massive chips. *Behavioral modeling describes the functionality of a design* by specifying what the designed circuit will do, not how to build it in hardware. It specifies the input–output model of a logic circuit and suppresses details about physical, gate-level implementation.

Behavioral modeling encourages designers to (1) rapidly create a behavioral prototype of a design (without binding it to hardware details), (2) verify its functionality, and then (3) use a synthesis tool to optimize and map the design into a selected physical technology. If the model has been written in a synthesis-ready style, the synthesis tool will remove redundant logic, perform tradeoffs between alternative architectures and/or multilevel equivalent circuits, and ultimately achieve a design that is compatible with area or timing constraints. By focusing the designer's attention on the functionality that is to be implemented rather than on individual logic gates and their interconnections, behavioral modeling provides the freedom to explore alternatives to a design before committing it to production.

Aside from its importance in synthesis, behavioral modeling provides flexibility to a design project by allowing parts of the design to be modeled at different levels of abstraction. The Verilog language accommodates mixed levels of abstraction so that portions of the design that are implemented at the gate level (i.e., structurally) can be integrated and simulated concurrently with other parts of the design that are represented by behavioral descriptions.

1.1.4 Simulation and Functional Verification

The functionality of a design is verified (Step 4 in Figure 1-1) either by simulation or by formal methods [7]. Our discussion will focus on simulation that is reasonable for the size of circuits we can present here. The design flow iterates back to Step 3 until the functionality of the design has been verified. The verification process is threefold; it includes (1) development of a test plan, (2) development of a testbench, and (3) execution of the test.

1.1.4.1 Test Plan Development A carefully documented *test plan* is developed to specify what functional features are to be tested and how they are to be tested. For example, the test plan might specify that the instruction set of an arithmetic and logic unit (ALU) will be verified by an exhaustive simulation of its behavior, for a specific set of

input data. Test plans for sequential machines must be more elaborate to ensure a high level of confidence in the design, because they may have a large number of states. A test plan identifies the stimulus generators, response monitors, and the gold standard response against which the model will be tested.

1.1.4.2 Testbench Development The *testbench* is a Verilog module in which the unit under test (UUT) has been instantiated, together with pattern generators that are to be applied to the inputs of the model during simulation. Graphical displays and/or response monitors are part of the testbench. The testbench is documented to identify the goals and sequential activity that will be observed during simulation (e.g., "Testing the opcodes"). If a design is formed as an architecture of multiple modules, each must be verified separately, beginning with the lowest level of the design hierarchy, then the integrated design must be tested to verify that the modules interact correctly. In this case, the test plan must describe the functional features of each module and the process by which they will be tested, but the plan must also specify how the aggregate is to be tested.

1.1.4.3 Test Execution and Model Verification The testbench is exercised according to the test plan and the response is verified against the original specification for the design, e.g. does the response match that of the prescribed ALU? This step is intended to reveal errors in the design, confirm the syntax of the description, verify style conventions, and eliminate barriers to synthesis. Verification of a model requires a systematic, thorough demonstration of its behavior. *There is no point in proceeding further into the design flow until the model has been verified.*

1.1.5 Design Integration and Verification

After each of the functional subunits of a partitioned design have been verified to have correct functionality, the architecture must be integrated and verified to have the correct functionality. This requires development of a separate testbench whose stimulus generators exercise the input–output functionality of the top-level module, monitor port and bus activity across module boundaries, and observe state activity in any embedded state machines. *This step in the design flow is crucial* and must be executed thoroughly to ensure that the design that is being signed off for synthesis is correct.

1.1.6 Presynthesis Sign-Off

A demonstration of full functionality is to be provided by the testbench, and any discrepancies between the functionality of the Verilog behavioral model and the design specification must be resolved. *Sign-off* occurs after all known functional errors have been eliminated.

1.1.7 Gate-Level Synthesis and Technology Mapping

After all syntax and functional errors have been eliminated from the design and sign-off has occurred, a synthesis tool is used to create an optimal Boolean description and compose it in an available technology. In general, a synthesis tool removes redundant logic and seeks to reduce the area of the logic needed to implement the functionality

and satisfy performance (speed) specifications. This step produces a netlist of standard cells or a database that will configure a target FPGA.

1.1.8 Postsynthesis Design Validation

Design validation compares the response of the synthesized gate-level description to the response of the behavioral model. This can be done by a testbench that instantiates both models, and drives them with a common stimulus, as shown in Figure 1-2. The responses can be monitored by software and/or by visual/graphical means to see whether they have identical functionality. For synchronous designs, the match must hold at the boundaries of the machine's cycle—intermediate activity is of no consequence. If the functionality of the behavioral description and the synthesized realization do not match, painstaking work must be done to understand and resolve the discrepancy. Postsynthesis design validation can reveal software race conditions in the behavioral model that cause events to occur in a different clock cycle than expected.[1] We will discuss how good modeling techniques can prevent this outcome.

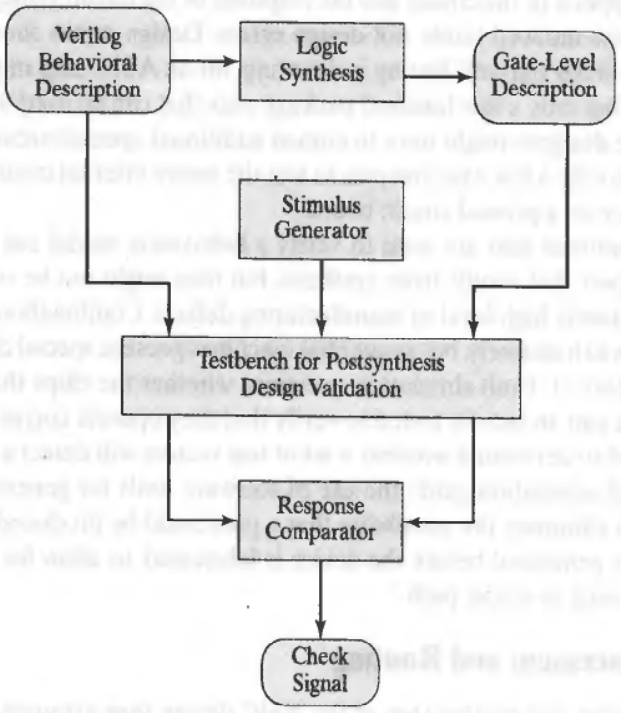

FIGURE 1-2 Postsynthesis design validation.

[1]Postsynthesis validation in an ASIC design flow is followed by a step for postlayout timing verification.

1.1.9 Postsynthesis Timing Verification

Although the synthesis process is intended to produce a circuit that meets timing specifications, the circuit's timing margins must be checked to verify that speeds are adequate on critical paths (Step 9). This step is repeated after Step 13, because synthesis tools do not accurately anticipate the effect of the capacitive delays induced by interconnect metalization in the layout. Ultimately, these delays must be extracted from the properties of the materials and the geometric details of the fabrication masks. The extracted delays are used by a static timing analyzer to verify that the longest paths do not violate timing constraints. The circuit might have to be resynthesized or re-placed and rerouted to meet specifications. Resynthesis might require (1) transistor resizing, (2) architectural modifications/substitutions, and (3) device substitution (more speed at the cost of more area).

1.1.10 Test Generation and Fault Simulation

After fabrication, integrated circuits must be tested to verify that they are free of defects and operate correctly. Contaminants in the clean-room environment can cause defects in the circuit and render it useless. In this step of the design flow a set of test vectors is applied to the circuit and the response of the circuit is measured. Testing considers process-induced faults, not design errors. Design errors should be detected before presynthesis sign-off. Testing is daunting, for an ASIC chip might have millions of transistors, but only a few hundred package pins that can be used to probe the internal circuits. The designer might have to embed additional, special circuits that will enable a tester to use only a few external pins to test the entire internal circuitry of the ASIC, either alone or on a printed circuit board.

The patterns that are used to verify a behavioral model can be used to test the fabricated part that results from synthesis, but they might not be robust enough to detect a sufficiently high level of manufacturing defects. Combinational logic can be tested for faults exhaustively, but sequential machines present special challenges, as we will see in Chapter 11. Fault simulation questions whether the chips that come off the fabrication line can, in fact, be tested to verify that they operate correctly. Fault simulation is conducted to determine whether a set of test vectors will detect a set of faults. The results of fault simulation guide the use of software tools for generating additional test patterns. To eliminate the possibility that a part could be produced but not tested, test patterns are generated before the device is fabricated, to allow for possible changes in the design, such as a scan path.[2]

1.1.11 Placement and Routing

The placement and routing step of the ASIC design flow arranges the cells on the die and connects their signal paths. In cell-based technology the individual cells are integrated to form a global mask that will be used to pattern the silicon wafer with gates.

[2] Scan paths are formed by replacing ordinary flip-flops with specially designed flip-flops that can be connected together in test mode to form a shift register. Test patterns can be scanned into the design, and applied to the internal circuitry. The response of the circuit can be captured in the scan chain and shifted out for analysis.

This step also might involve inserting a clock tree into the layout, to provide a skew-free distribution of the clock signal to the sequential elements of the design. If a scan path is to be used, it will be inserted in this step too.

1.1.12 Physical and Electrical Design Rule Checks

The physical layout of a design must be checked to verify that constraints on material widths, overlaps, and separations are satisfied. Electrical rules are checked to verify that fanout constraints are met and that signal integrity is not compromised by electrical crosstalk and power-grid drop. Noise levels are also checked to determine whether electrical transients are problematic. Power dissipation is modeled and analyzed in this step to verify that the heat generated by the chip will not damage the circuitry.

1.1.13 Parasitic Extraction

Parasitic capacitance induced by the layout is extracted by a software tool and then used to produce a more accurate verification of the electrical characteristics and timing performance of the design (Step 13). The results of the extraction step are used to update the loading models that are used in timing calculations. Then the timing constraints are checked again to confirm that the design, as laid out, will function at the specified clock speed.

1.1.14 Design Sign-Off

Final sign-off occurs after all of the design constraints have been satisfied and timing closure has been achieved. The mask set is ready for fabrication. The description consists of the geometric data (usually in GDS-II format) that will determine the photomasking steps of the fabrication process. At this point significant resources have been expended to ensure that the fabricated chip will meet the specifications for its functionality and performance.

1.2 IC Technology Options

Figure 1-3 shows various options for creating the physical realization of a digital circuit in silicon, ranging from programmable logic devices (PLDs) to full-custom ICs. Fixed-architecture programmable logic devices serve the low end of the market (i.e., low volume and low performance requirements). They are relatively cheap commodity parts, targeted for low-volume designs.

The physical database of a design might be implemented as (1) a full-custom layout of high-performance circuitry, (2) a configuration of standard cells, or (3) gate arrays (field- or mask-programmable), depending on whether the anticipated market for the ASIC offsets the cost of designing it, and the required profit. Full-custom ICs occupy the high end of the cost–performance domain, where sufficient volume or a customer with corporate objectives and sufficient resources warrant the development time and investment required to produce fully custom designs having minimal area and maximum speed. FPGAs have a fixed, but electrically programmable architecture

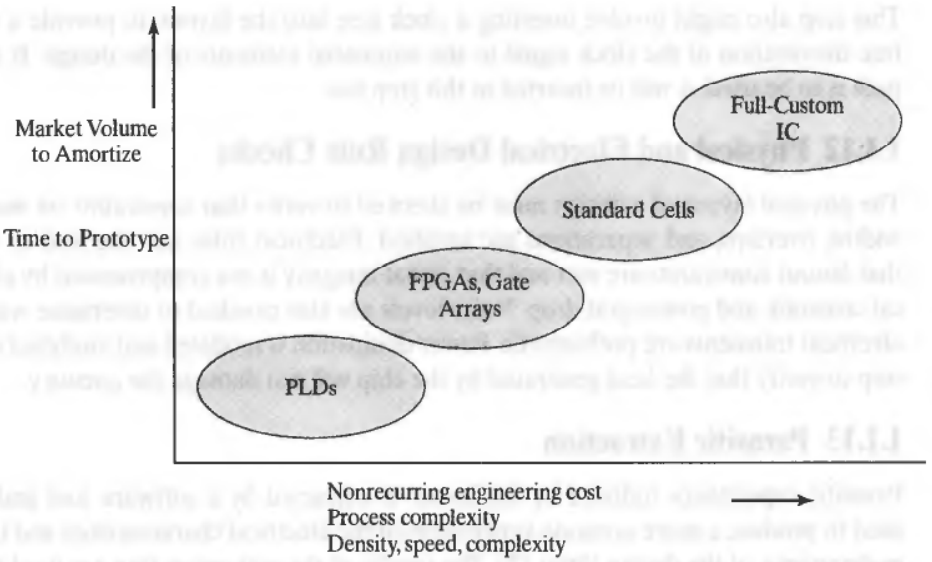

FIGURE 1-3 Alternative technologies for IC implementation.

for implementing modest-sized designs. The tools supporting this technology allow a designer to write and synthesize a Verilog description into a working physical part on a prototype board in a matter of minutes. Consequently, design revisions can be made at very low cost. Board layout can proceed concurrently with the development of the part because the footprint and pin configuration of an FPGA are known. Low-volume prototyping sets the stage for the migration of a design to mask-programmable and standard-cell-based parts.

In mask-programmable gate-array technology a wafer is populated with an array of individual transistors that can be interconnected to create logic gates that implement a desired functionality. The wafers are prefabricated and later personalized with metal interconnect for a customer. All but the metalization masks are common to all wafers, so the time and cost required to complete masks is greatly reduced, and the other nonrecurring engineering (NRE) costs are amortized over the entire customer base of a silicon foundry.

Standard cell technology predesigns and characterizes individual logic gates to the mask level and assembles them in a shared library. A place-and-route tool places the cells in channels on the wafer, interconnects them, and integrates their masks to create the functionality for a specific application. The mask set for a customer is specific to the logic being implemented and can cost over $500K for large circuits, but the NRE costs associated with designing and characterizing the cell library are amortized over the entire customer base. In high-volume applications, the unit cost of the parts can be relatively cheap compared to the unit cost of PLDs and FPGAs.

1.3 Overview

The following chapters will cover most of the steps in the design flow presented in Figure 1-1, but not cell placement and routing, design-rule checking, or parasitic extraction. These steps are conducted by separate tools, which operate on the physical mask database rather than on an HDL model of the design, and they presume that a functionally correct design has been synthesized successfully. The steps we cover are the mainstream designer-driven steps in the overall ASIC flow.

In the remaining chapters, we will review manual methods for designing combinational and sequential logic design in Chapters 2 and 3. Then we will treat combinational logic design (Chapter 4) and sequential logic design (Chapter 5) using Verilog, and by example, contrast manual and HDL-based methods. This chapter also introduces the use of ASM charts and algorithmic state machine and datapath (ASMD) charts, which prove to be very useful in writing behavioral models of sequential machines. Chapter 6 covers synthesis of combinational and sequential logic with Verilog models. This chapter equips the designer with the background to compose synthesis-friendly designs and to avoid common pitfalls that can thwart a design. Chapter 7 continues with a treatment of datapath controllers, including a RISC CPU and a UART. Chapter 8 introduces PLDs, CPLDs, RAMS and ROMS, and FPGAs. The problems at the end of this chapter specify designs that can be implemented on a widely available prototyping board. Chapter 9 covers algorithms and architectures for digital processors, and Chapter 10 treats architectures for arithmetic operations. Chapter 11 treats the postsynthesis issues of timing verification, test generation, and fault simulation, including JTAG and BIST.

Three things matter in learning design with an HDL: examples, examples, and examples. We present several examples, with increasing difficulty, and make available their Verilog descriptions. Several challenging problems are included at the end of each chapter that require design with Verilog. We urge the reader to embrace the mantra: simplify, clarify, verify.

REFERENCES

1. *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*, Language Reference Manual (LRM), IEEE Std.1364–1995. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 1996.
2. *IEEE Standard VHDL Language Reference Manual* (LRM), IEEE Std, 1076–1987. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 1988.
3. Negel LW. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, Memo ERL-M520, Department of Electrical Engineering and Computer Science, University of California at Berkeley, May 9, 1975.
4. Fitzpatrick D, Miller I. *Analog Behavioral Modeling with the Verilog-A Language*, Boston: Kluwer, 1998.
5. SystemC Draft Specification, Mountain View, CA: Synopsys, 1999.

6. Rich, D., Fitzpatrick, T., "Advanced Verification Using the Superlog Language," Proc. Int. HDL Conference, San Jose, March 2002.
7. Chang H, et al. *Surviving the SOC Revolution*, Boston: Kluwer, 1999.