Preface

Simplify, Clarify, and Verify

Behavioral modeling with a hardware description language (HDL) is the key to modern design of application-specific integrated circuits (ASICs). Today, most designers use an HDL-based design method to create a high-level, language-based, abstract description of a circuit, and verify its functionality and timing. The language used to teach design methodology in the first edition of this text, IEEE 1464-1995, has undergone two revisions to improve the effectiveness and efficiency of the language: IEEE 1364-2001 followed by a revision in 2005, known as Verilog-2001 and Verilog-2005, respectively.

The motivation behind this edition is basically the same as that which guided the first edition: students preparing to contribute to a productive design team must know how to use a HDL at key stages of the design flow. Thus, there is a need for a course going beyond the basic principles and methods learned in a first course in digital design. This book is written for such a course.

The quantity of books discussing HDLs far exceeds that which was available at the time of the first edition, and most of these are still oriented toward explanations of language syntax, rather than toward design, and are not well-suited for classroom use. Our focus is on design methodology enabled by an HDL. Thus, the language itself has a subordinate role. In this edition, we have made a strong effort to demonstrate by examples the importance of partitioning a digital machine to expose its datapath, status (feedback) signals, and controller (finite state machine). This effort leads, we think, to a much clearer and straightforward approach to designing and verifying complex digital machines. We present an abundance of simulation results, with annotation to help students (1) understand the operation of a sequential machine and (2) appreciate the time-sequential interaction between the signals produced by the controller, the operations in the datapath, and the signals reported back to the controller from the datapath, all with the aim of developing synthesizable, latch-free, race-free designs.

The language enhancements of Verilog 2001, 2005 have been used to reexamine and simplify the code of our models. We emphasize industry practices, and do not unwittingly encourage academic styles of modeling without regard for whether a model can be synthesized. Solutions to selected problems, together with additional solved exercises that are not included in the text, are available for instructors at the companion Web site: http://www.pearsonhighered.com/ciletti. The first edition treated synchronous FIFOs; this edition treats synchronous and asynchronous FIFOs, and presents

an elaborate example of using an asynchronous FIFO to synchronize transfer of data across clock domains.

Design practice is always in a flux. One tension is between the approach of writing a model in C for an embedded controller versus writing a model in Verilog. The C-based approach executes statements; Verilog models execution of multiple concurrent behaviors of a machine. The latter approach compiles hardware; the former compiles statements for a preexisting hardware unit. The compiled hardware of the Verilog model produces equivalent I/O signals at the interface to the host machine for a particular application. The distinction here is that embedded code does not equal embedded hardware. This text aims to teach the hardware modeling/compilation paradigm and to anticipate the results of synthesis. Programming in C anticipates the data produced by a program, and the machine/processor is itself transparent. In contrast, modeling in Verilog anticipates the hardware that will produce the I/O signals demanded by the application, and requires the developer to think in terms of time-sequential control of register operations. Verilog modeling encourages this understanding about the nature of a digital machine.

Our goal in this book is to build on a student's background from a first course in logic design by (1) briefly reviewing basic principles of combinational and sequential logic, (2) introducing the use of HDLs in design, (3) emphasizing descriptive styles that will allow the reader to quickly design working circuits suitable for ASICs and/or field-programmable gate array (FPGA) implementation, and (4) providing design examples having a range of difficulty. The end-of-chapter problems encourage students to simplify, clarify, and verify their designs. The nature of many of the problems is that of open-ended design, requiring verification that the design meets a prescribed specification.

The widely used Verilog HDL (IEEE Standard 1364) serves as a common framework supporting the design activities treated in this book. The first edition focused on developing, verifying, and synthesizing designs of digital circuits, and *not* on the Verilog language. This edition maintains that focus.

Most students taking a second course in digital design will be familiar with at least one programming language and will be able to draw on that background in reading this text. We cover only the core and most widely used features of Verilog. In order to emphasize *using* the language in a synthesis-oriented design environment, we have purposely placed many details, features, and explanations of syntax in the appendices and at the publisher's Web site for reference on an "as needed" basis, but an appendix provides the complete formal syntax of the language.

Most entry-level courses in digital design introduce finite state machines using state transition diagrams, and algorithmic state machine (ASM) charts. We do likewise, but we make heavier use of ASM charts and demonstrate their utility in developing behavioral models of sequential machines. The important problem of systematically designing a finite state machine to control a datapath in a complex digital machine is treated in-depth with ASMD charts, i.e. ASM charts annotated to display the register operations of the controlled datapath. The design of a RISC CPU and other important hardware units are given as examples. Our companion Web site includes the RISC machine's source code and an assembler that can be used to develop programs for applications. The machine also serves as a starting point for developing a more robust instruction set and architectural variants.

The Verilog language is introduced in an integrated, but selective manner, only as needed to support design examples. The text has a large set of examples illustrating how to address the key steps in a VLSI circuit design methodology using the Verilog HDL. The source code of the examples has been verified to be correct. Source code for all of the examples and their testbenches are available at the publisher's Web site.

The Intended Audience

This book is for students in an advanced course in digital design, and for professional engineers interested in learning Verilog by example, in the context of its use in the design flow of modern integrated circuits. The level of presentation is appropriate for seniors and first-year graduate students in electrical engineering, computer engineering, and computer science, as well as for professional engineers who have had an introductory course in logic design. The book presumes that the reader has a basic background in Boolean algebra and its use in logic circuit design, and a familiarity with synchronous finite state machines. Building on this foundation, the book addresses the design of several important circuits used in computer systems, digital signal processing, image processing, data transfer across clock domains, built-in self-test (BIST) and, other applications. The examples cover the key design problems of modeling, architectural trade-offs, pipelining, multiprocessor implementations, functional verification, timing analysis, test generation, fault simulation, design for testability, logic synthesis, and post-synthesis verification.

What's New in this Edition

- Exploits key features of Verilog 2001, 2005
- Illustrates and promotes a synthesis-ready style of register transfer level (RTL) and algorithmic modeling with Verilog 2001, 2005
- Provides an in-depth treatment of algorithms and architectures for digital machines (e.g. an image processor, digital filters, and circular buffers) with Verilog 2001, 2005
- Includes comprehensive design examples (e.g. a RISC machine and various datapath controllers) with Verilog 2001, 2005
- Includes numerous annotated and explained graphical illustrations of simulation results
- Contains over 150 fully verified examples with Verilog 2001, 2005
- Contains a worked example with JTAG and BIST for testing with Verilog 2001, 2005
- Contains an Appendix with full formal syntax of the Verilog 2001, 2005 HDL
- Treats asynchronous and synchronous FIFOs

Special Features of the Book

- Provides a brief review of basic principles in combinational and sequential logic
- Focuses on modern digital design methodology
- Demonstrates the utility of ASM and ASMD charts for behavioral modeling
- Clearly distinguishes between synthesizable and nonsynthesizable loops
- Provides practical treatment of timing analysis, fault simulation, testing, and design for testability, with examples
- Provides several problems with a wide range of difficulty after each chapter
- Combines a solution manual with an on-line repository of additional worked exercises
- Lists an index of all models developed in the examples
- Includes a set of FPGA-based, lab-ready exercises linked to the book (e.g. arithmetic and logic unit (ALU), programmable lock, a keypad scanner with a FIFO, a serial communications link with error correction, an SRAM controller, and first in, first out (FIFO) memory, RISC CPU, and FIFO)
- Supported by an ongoing Companion Web site containing:
  1. Source files of all models developed in the examples
  2. Source files of testbenches for simulating all of the examples
  3. An Instructor's Classroom Kit containing transparency files for a complete course based on the subject matter is available for instructors only

4. Solutions to selected problems is available for instructors only
5. Additional worked problems
6. Jump-start tutorials helping students get immediate results with selected software tools (e.g. simulator)
7. Answers to frequently asked questions (FAQs)

Sequences for Course Presentation

The material in the text begins with a brief review of combinational and sequential logic design, but then progresses in the order dictated by the design flow for an ASIC or an FPGA. Chapters 1 to 6 treat design topics through synthesis, and should be covered in order, but Chapters 7 to 10 can be covered in any order. The homework exercises are challenging, and the lab-ready FPGA-based exercises are suitable for a companion lab or for end-of-semester projects. Chapter 10 presents several architectures for arithmetic operations, affording a diversity of coverage. Chapter 11 treats post-synthesis design validation, timing analysis, fault simulation, and design for testability. The coverage of these topics can be omitted, depending on the level and focus of the course.

Some Caveats

We do not adhere to common practice for using upper and lower case text, or for using courier font in code listings. Our choices have been based on maximizing the overall visual appeal and readability of the listed code. The visual result offsets, we think, the extra care required to ensure that the code is composed correctly in our examples. Block diagrams have been simplified to reduce the visual clutter, so we typically show only the actual, external names of signals, and omit their formal, internal counterparts. D-type flip-flops are used almost exclusively because they play a dominant role in synthesis with modern EDA tools.

Chapter Descriptions

Chapter 1 briefly discusses the role of HDLs in design flows for cell-based ASICs and FPGAs. Chapters 2 and 3 review mainstream topics that would be covered in a first course in digital design, using classical methods, i.e. Karnaugh maps. This material will refresh the reader's background, and the examples will be used later to introduce HDL-based methods of design. Chapters 4 and 5 introduce modeling of combinational and sequential logic with the Verilog HDL, and place emphasis on coding styles that are used in behavioral modeling. Chapter 6 addresses cell-based synthesis of ASICs, and introduces synthesis of combinational and sequential logic. Here we pursue two main objectives: (1) present synthesis-friendly coding styles and (2) form a foundation that will enable the reader to anticipate the results of synthesis, especially when synthesizing sequential machines. Many sequential machines are partitioned into a datapath and a controller. Chapter 7 covers examples that illustrate how to design a controller for a datapath, including machines having feedback of status signals from the datapath to the controller. The designs of a simple RISC CPU and a UART[1] serve as platforms for the subject matter.

[1]Universal Asynchronous Receiver and Transmitter (UART), a circuit used in data transmission between systems.

Chapter 8 covers PLDs, complex PLDs (CPLDs), ROMs, and SRAMs, and then expands the synthesis target to include FPGAs. Chapter 9 treats the modeling and synthesis of computational units and algorithms found in computer architectures, digital filters, and other signal processors. Chapter 10 develops and refines algorithms and architectures for the arithmetic units of digital machines. In Chapter 11, we use the Verilog HDL in conjunction with fault simulators and timing analyzers to revisit a selection of previously designed machines and consider performance/timing issues and testability, to complete the treatment of design flow tasks that rely heavily on designer intervention. This chapter models the test access port (TAP) controller defined by IEEE 1149.1 standard (commonly known as the JTAG standard), and presents an example of its use. Another elaborate example covers BIST.

Acknowledgments

The author is grateful for the support of colleagues and students who expanded his vision of Verilog and contributed to this text. It represents the combined experience of my developing and teaching courses at the University of Colorado, taking sabbaticals in industry with Hewlett-Packard and Ford Microelectronics Inc., Prisma Inc., spending a sabbatical at the Technical University of Delft (Netherlands), and developing and presenting short courses in Europe and Asia. Some of the companies now exist only in memory, but I am deeply grateful to many individuals in industry and at the University of Colorado for their helping to shape my path in the realm of VLSI circuit design. The reviewers of the original manuscript for the first edition provided encouragement, critical judgment, and many helpful suggestions. Dr. Jim Tracey and Dr. Rodger Ziemer supported and encouraged my initial efforts to focus my work on the design of VLSI circuits. Deepak Goel (Ford Microelectronics) introduced me to VLSI design on the then state-of-the-art Daisy workstations at Ford Microelectronics. Bill Fuchs (Simucad, Inc.) supported my efforts to acquire an industrial-strength Verilog simulator. Tom Saponas and Dave Ritchey (Hewlett-Packard) placed me at the helm of a project to reverse-engineer a dynamic timing analyzer. Two students, David Uranek and Jerry Barnett, assured success. Dave Still (Prisma Corp.), my manager during a summer job, provided the environment and encouragement for me to tackle a significant problem in modeling of a high-performance, multicore system; Stu Sutherland (Sutherland HDL) helped the author gain a deeper appreciation for the issue of race conditions that can creep into the models of a digital system. These insights led to my adhering to the disciplined style of using nonblocking assignments for modeling edge-sensitive behavior and blocked assignments for modeling level-sensitive behavior, and to my efforts to help students understand the operation and design of synchronous digital machines. Rich discussions with Dr. Hubert Kaeslin, friend and colleague (Swiss Federal Institute of Technology – Zurich), led to my delving more deeply into algorithms and architectures for digital processors. Kirk Sprague and Scott Kukel were helpful in developing a Hamming encoder to work with a UART. Cris Hagan's thesis led to the models presented in Chapter 9 for decimators and other functional units found in digital signal processors. Rex Anderson proofread several chapters and scrubbed down the work of the first edition. Students Terry Hansen and Lisa Horton provided the inspiration for the coffee vending machine example, and developed the assembler that supports the RISC CPU. Thanks also to the many students whose classroom dialogue was helpful in the second edition. Profs. Greg Tumbush (University of Colorado at Colorado Springs) and Chen-Huan Chiang (Temple University) provided critical and helpful suggestions for this edition. Thanks also to the many students whose classroom dialogue was helpful in the second edition. Scott Disanno and Irwin Zucker shepherded this edition through the publication process, and Haseen Khan orchestrated the composition of the book from my manuscript. My deep thanks to all of you.

Dedication

This book is dedicated to the memory of Sr. Laurencia Rihn, RSM, and Fr. Jerry Wilson, CSC. My life has been shaped by their faith, encouragement, and love. To my wife, Jerilynn, and our children, Monica, Lucy, Rebecca, Christine, and Michael, their spouses, Mike McCormick, David Steigerwald, Peter Van Dusen, and Michelle Puhr Ciletti, and our grandchildren, the "cousin dozen": Michael Angus, Katherine, Brigid, David, Jackson, Samantha, Peter, Matthew, Ella, Anthony, Abigail, and Joseph—thank you for the journey and love we've shared.