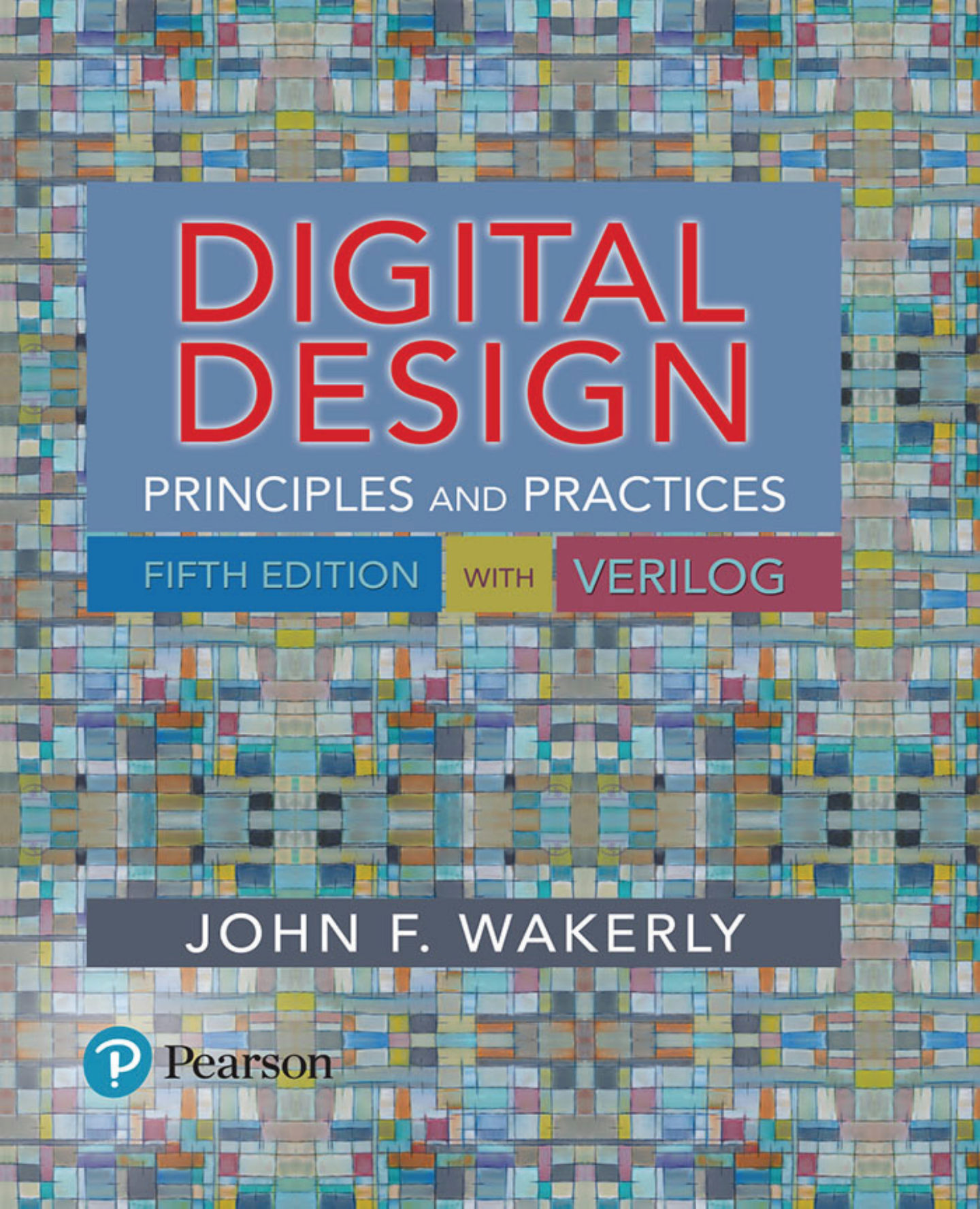

DIGITAL DESIGN
PRINCIPLES AND PRACTICES
FIFTH EDITION WITH VERILOG
JOHN F. WAKERLY
Pearson

DIGITAL DESIGN

Principles and Practices

DIGITAL DESIGN

Principles and Practices

Fifth Edition with Verilog

John F. Wakerly

Pearson 330 Hudson Street, NY NY 10013

Senior Vice President Courseware Portfolio Management: Marcia J. Horton
Director, Portfolio Management: Engineering, Computer Science & Global Editions: Julian Partridge
Portfolio Manager Assistant: Michelle Bayman
Field Marketing Manager: Demetrius Hall
Product Marketing Manager: Yvonne Vannatta
Marketing Assistant: Jon Bryant
Content Managing Producer, ECS and Math: Scott Disanno
Operations Specialist: Maura Zaldivar-Garcia
Manager, Rights and Permissions: Ben Ferrini
Cover Designer: Black Horse Designs
Cover Art: "Tuesday Matinee," by Peter Alan Crowell

Pearson Education Ltd., London
Pearson Education Singapore, Pte. Ltd
Pearson Education Canada, Inc.
Pearson Education Japan
Pearson Education Australia PTY, Ltd
Pearson Education North Asia, Ltd., Hong Kong
Pearson Education de Mexico, S.A. de C.V.
Pearson Education Malaysia, Pte. Ltd.
Pearson Education, Inc., Hoboken

Library of Congress Cataloging-in-Publication Data on File

ISBN-10: 013446009X
1 16 ISBN-13: 9780134460093

Pearson www.pearsonhighered.com

For Ralph and Carm, again

This page intentionally left blank

CONTENTS

PREFACE

This book is for everyone who wants to design and build real digital circuits. It is based on the idea that, in order to do this, you have to grasp the fundamentals, but at the same time you need to understand how things work in the real world. Hence, the "principles and practices" theme.

The practice of digital design has undergone a major transformation during the past 30 years, a direct result of the stunning increases in integrated-circuit speed and density over the same time period. In the past, when digital designers were building systems with thousands or at most tens of thousands of gates and flip-flops, academic courses emphasized minimization and efficient use of chip and board-level resources.

Today, a single chip can contain tens of millions of transistors and can be programmed to create a system-on-a-chip that, using the technology of the past, would have required hundreds of discrete chips containing millions of individual gates and flip-flops. Successful product development nowadays is limited more by the design team's ability to correctly and completely specify the product's detailed functions, than by the team's ability to cram all the needed circuits into a single board or chip. Thus, a modern academic program must necessarily emphasize design methodologies and software tools, including hardware description languages (HDLs), that allow very large, hierarchical designs to be accomplished by teams of designers.

On one hand, with HDLs, we see the level of abstraction for typical designs moving higher, above the level of individual gates and flip-flops. But at the same time, the increased speed and density of digital circuits at both the chip and board level is forcing many digital designers to be more competent at a lower, electrical circuit level.

The most employable and ultimately successful digital designers are skilled, or at least conversant, at both levels of abstraction. This book gives you

the opportunity to learn the basics at the high level (HDLs), at the low level (electrical circuits), and throughout the "vast middle" (gates, flip-flops, and higher-level digital-design building blocks).

Target Audience

introductory courses

The material in this book is appropriate for introductory and second courses on digital logic design in electrical or computer engineering or computer science curricula. Computer science students who are unfamiliar with basic electronics concepts or who just aren't interested in the electrical behavior of digital devices may wish to skip Chapter 14; the rest of the book is written to be independent of this material, as long as you understand the basics in Chapter 1. On the other hand, *anyone* with a basic electronics background who wants to get up to speed on digital electronics can do so by reading Chapter 14. In addition, students with *no* electronics background can get the basics by reading a 20-page electronics tutorial at the author's website, `www.ddpp.com`.

electronics concepts

Although this book's starting level is introductory, it goes beyond that and contains much more material than can be taught in a typical introductory course. I expect that typical courses will use no more than two-thirds of the material here, but each will use a *different* two thirds. Therefore, I've left it to the individual instructors and independent readers to tailor their reading to their own needs. To help these choices along, though, I've marked the headings of *optional sections* with an asterisk. In general, these sections can be skipped without any loss of continuity in the non-optional sections that follow. Also, the material in the sidebars (aka "boxed comments") is generally optional.

optional sections

sidebars
boxed comments
second courses
laboratory courses
fun stuff

Undoubtedly, some people will use this book in second courses and in laboratory courses. Advanced students will want to skip the basics and get right into the fun stuff. Once you know the basics, some of the most important and fun stuff is in the many sections and examples of digital design using Verilog.

marginal notes
marginal pun

All readers should make good use of the comprehensive index and of the *marginal notes* throughout the text that call attention to definitions and important topics. Maybe the highlighted topics in *this* section were more marginal than important, but I just wanted to show off my text formatting system.

Chapter Descriptions

What follows is a list of short descriptions of this book's fifteen chapters. This may remind you of the section in typical software guides, "For People Who Hate Reading Manuals." If you read this list, then maybe you don't have to read the rest of the book.

- Chapter 1 gives a few basic definitions and a preview of a few important topics. It also has a little bit on digital circuits, to enable readers to handle the rest of the book without Chapter 14's "deep dive."

NOT AS LONG AS IT SEEMS

A few reviewers have complained about the length of previous editions of this book. The present edition is a little shorter, but also please keep in mind:

- You don't have to read everything. The headings of sections and subsections that are optional for most readers are marked with an asterisk.
- Stuff written in these "boxed comments" (a.k.a. sidebars) is usually optional too.
- I asked the publisher to print this book in a larger font (11 point) than is typical for technical texts (10 point). This is easier on your eyes and mine, and it also allows me to put in more figures and tables while still keeping most of them on the same facing pages as the referring text. (I do the page layout myself and pay a lot of attention to this.)
- I write my books to be "reference quality," with comprehensive topic coverage and excellent indexing, so you can come back to them in later courses, or later in your career to refresh or even to learn new things. The cost of books being what they are these days, you may not keep this book, but the option is there.

- Chapter 2 is an introduction to binary number systems and codes. Readers who are already familiar with binary number systems from a software course should still read Sections 2.10–2.13 to get an idea of how binary codes are used by hardware. Advanced students can get a nice introduction to error-detecting codes by reading Sections 2.14 and 2.15. The material in Section 2.16.1 should be read by everyone; it is used in a lot of modern systems.
- Chapter 3 teaches combinational logic design principles, including switching algebra and combinational-circuit analysis, synthesis, and minimization.
- Chapter 4 introduces various digital-design practices, starting with documentation standards, probably the most important practice for aspiring designers to start practicing. Next, it introduces timing concepts, especially for combinational circuits, and it ends with a discussion of HDLs, design flow, and tools.
- Chapter 5 is a tutorial and reference on Verilog, the HDL that is used throughout the rest of the book. The first few sections should be read by all, but some readers may wish to skip the rest until it's needed, since new Verilog constructs are summarized in later chapters "on the fly" the first time they're used, mainly in Chapter 6.
- Chapter 6 describes two "universal" combinational building blocks, ROMs and PLDs. It then describes the two most commonly used functional building blocks, decoders and multiplexers; gate-level and Verilog-based designs are shown for each. It's possible for the reader to go from here directly to state machines in Chapter 9, and come back to 7 and 8 later.

- Chapter 7 continues the discussion of combinational building blocks, at both the gate level and in Verilog, for three-state devices, priority encoders, XOR and parity functions, and comparators, then concludes with a Verilog example design for a nontrivial "random logic" function.
- Chapter 8 covers combinational circuits for arithmetic functions, including adding and subtracting, shifting, multiplying, and dividing.
- Chapter 9 is a traditional introduction to state machines using D flip-flops, including analysis and synthesis using state tables, state diagrams, ASM charts, and Verilog.
- Chapter 10 introduces other sequential elements including latches, more edge-triggered devices, and their Verilog behavioral models. This chapter also describes the sequential elements in a typical FPGA and, for interested readers, has sections on sequential PLDs and feedback sequential circuits.
- Chapter 11 is focused on the two most commonly used sequential-circuit building blocks, counters and shift registers, and their applications. Both gate-level and Verilog-based examples are given.
- Chapter 12 gives a lot more details on how to model state machines using Verilog and gives many examples.
- Chapter 13 discusses important practical concepts for sequential-circuit design, including synchronous system structure, clocking and clock skew, asynchronous inputs and metastability, and a detailed two-clock synchronization example in Verilog.
- Chapter 14 describes digital circuit operation, placing primary emphasis on the external electrical characteristics of logic devices. The starting point is a basic electronics background including voltage, current, and Ohm's law. This chapter may be omitted by readers who aren't interested in how to make real circuits work, or who have the luxury of having someone else to do the dirty work.
- Chapter 15 is all about memory devices and FPGAs. Memory coverage includes read-only memory and static and dynamic read/write memories in terms of both internal circuitry and functional behavior. The last section gives more details of an FPGA architecture, the Xilinx 7 series.

Most of the chapters contain references, drill problems, and exercises. Drill problems are typically short-answer or "turn-the-crank" questions that can be answered directly based on the text material, while exercises typically require a little more thinking. The drill problems in Chapter 14 are particularly extensive and are designed to allow non-EEs to ease into this material.

Differences from the Fourth Edition

For readers and instructors who have used previous editions of this book, this fifth edition has several key differences in addition to general updates:

- This edition covers Verilog only; there's no VHDL. Bouncing between the languages is just too distracting. Moreover, Verilog and its successor SystemVerilog are now the HDLs of choice in non-government settings. See the excellent, well-reasoned and nicely documented paper by Steve Golson and Leah Clark, "Language Wars in the 21st Century: Verilog versus VHDL—Revisited" (2016 Synopsys Users Group Conference), and jump to the last section if you don't want to read the whole article.
- This edition has many more HDL examples and a much greater emphasis on design flow and on test benches, including purely stimulative as well as self-checking ones.
- To make the book more accessible to non-EE computer engineering students, detailed coverage of CMOS circuits has been moved to Chapter 14 and a minimal amount of electronics has been added to Chapter 1 so that the CMOS chapter can be skipped entirely if desired.
- TTL, SSI, MSI, 74-series logic, PLDs, and CPLDs have been deprecated.
- Karnaugh-map-based minimization has finally been, well, minimized.
- While the book still has a comprehensive Verilog tutorial and reference in Chapter 5, Verilog concepts are interspersed "just in time" in sidebars in Chapters 6 and 7 so students can go straight to "the good stuff" there.
- There is a greater emphasis on FPGA-based design, FPGA architectural features, and synthesis results and trade-offs.
- The chapter on combinational-logic elements has been split into three, to facilitate going straight to state machines after just the first if desired. This also allows more coverage of arithmetic circuits in the last.
- An entire chapter has been devoted to state-machine design in Verilog, including many examples.
- The chapter on synchronous design methodology now contains a detailed control-unit-plus-datapath example and a comprehensive example on crossing clocking domains using asynchronous FIFOs.
- The jokes aren't quite as bad, I hope.

Digital-Design Software Tools

All of the Verilog examples in this book have been compiled and tested using the Xilinx Vivado® suite, which includes tools for targeting Verilog, SystemVerilog, and VHDL designs to Xilinx 7-series FPGAs. However, in general there's no

special requirement for the examples to be compiled and synthesized using Vivado or even to be targeted to Xilinx or any other FPGA. Also, this book does *not* contain a tutorial on Vivado; Xilinx has plenty of online materials for that. Thus, a reader will able to use this text with any Verilog tools, including the ones described below.

The free "Webpack" edition of Vivado can be downloaded from Xilinx; it supports smaller 7-series FPGAs, Zynq® SoC-capable FPGAs, and evaluation boards. It's a big download, over 10 gigabytes, but it's a comprehensive tool suite. Pre-7-series FPGAs as well as the smaller Zynq FPGAs are supported by the Xilinx ISE® (Integrated Software Environment) tool suite, also available in a free "Webpack" edition. Note that ISE is supported in "legacy" mode and has not been updated since 2013. For either suite, go to `www.xilinx.com` and search for "Webpack download."

If you're using Altera (now part of Intel) devices, they also have a good University Program and tools; search for "Altera university support" and then navigate to the "For Students" page. Free tools include their Quartus™ Prime Lite Edition for targeting Verilog, SystemVerilog, and VHDL designs to their entry-level FPGAs and CPLDs, and a starter edition of industry-standard ModelSim® software for simulating them.

Both Altera and Xilinx offer inexpensive evaluation boards suitable for implementing FPGA-based student projects, either directly or through third parties. Such boards may include switches and LEDs, analog/digital converters and motion sensors, and even USB and VGA interfaces, and may cost less than $100 through the manufacturers' university programs.

Another long-time source of professional digital design tools with good university support is Aldec, Inc. (`www.aldec.com`). They offer a student edition of their popular Active-HDL™ tool suite for design entry and simulation; besides the usual HDL tools, it also includes block-diagram and state-machine graphical editors, and its simulator also includes a waveform editor for creating stimuli interactively. The Active-HDL simulator can be installed as a plug-in with Vivado to use its features instead of the Vivado simulator.

All of the above tools, as well as most other engineering design tools, run on Windows PCs, so if you are a Mac fan, get used to it! Depending on the tools, you may or may not have some success running them on a Mac in a Windows emulation environment like VMware's. The most important thing you can do to make the tools "go fast" on your PC is to equip it with a solid-state disk drive (SSD), not a rotating one.

Even if you're not ready to do your own original designs, you can use any of the above tools to try out and modify the examples in the text, since the source code for all of them is available online, as discussed next.

Engineering Resources and www.ddpp.com

Abundant support materials for this book are available on the Web at Pearson's "Engineering Resources" site. At the time of publication, the Pearson link was `media.pearsoncmg.com/bc/abp/engineering-resources`, but you know how it goes with long links. It's easier just to go to the author's website, `www.ddpp.com`, which contains a link to Pearson's site. Also, the author's site will contain the latest errata and other materials that may be added or changed "on the fly," and perhaps even a blog someday.

Resources at the Pearson site include downloadable source-code files for all Verilog modules in the book, selected drill and exercise solutions, and supplementary materials, such as a 20-page introduction to basic electronics concepts for non-EEs.

For Instructors

Pearson maintains a website with a comprehensive set of additional materials for instructors only. Go to the Engineering Resources site mentioned above, navigate to this book, and click on the "Instructor Resources" link. Registration is required, and it may take a few days for your access to be approved. Resources include additional drill and exercise solutions, additional source code, more exercises, and line art and tables from the book for use in your lectures. Upon request, materials from previous editions may also be posted there to aid instructors who are transitioning their courses from older technology.

Other resources for instructors include the author's site, `www.ddpp.com`, and the university programs at Xilinx, Altera, and Aldec; go to `www.ddpp.com` for up-to-date links to them. The manufacturer sites offer a variety of product materials, course materials, and discounts on chips and boards you can use in digital-design lab courses, and in some cases "full-strength" tool packages that you can obtain at a steep discount for use in your advanced courses and research.

Errors

Warning: This book may contain errors. The author and the publisher assume no liability for any damage—incidental, brain, or otherwise—caused by errors.

There, that should make the lawyers happy. Now, to make *you* happy, let me assure you that a great deal of care has gone into the preparation of this book to make it as error free as possible. I am anxious to learn of the remaining errors so that they may be fixed in future printings, editions, and spin-offs. Therefore, I will pay $5 via PayPal to the first finder of each undiscovered error—technical, typographical, or otherwise—in the printed book. Please email your comments to me by using the appropriate link at `www.ddpp.com`.

An up-to-date list of discovered errors can always be obtained using the appropriate link at `www.ddpp.com`. It will be a very short file transfer, I hope.

Acknowledgements

Many people helped make this book possible. Most of them helped with the first four editions and are acknowledged there. For the ideas on the "principles" side of this book, I still owe great thanks to my teacher, research advisor, and friend, the late Ed McCluskey. On the "practices" side, I got good advice from my friend Jesse Jenkins, from Xilinx staffers Parimal Patel and Trevor Bauer, and from fellow McCluskey advisee Prof. Subhasish Mitra of Stanford.

Since the fourth edition was published, I have received many helpful comments from readers. In addition to suggesting or otherwise motivating many improvements, readers have spotted dozens of typographical and technical errors whose fixes are incorporated in this fifth edition.

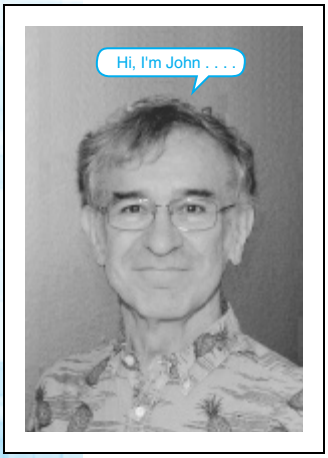
The most substantial influence and contribution to this edition came from ten anonymous (to me) academic reviewers, all of whom teach digital design courses using my fourth edition or one of its competitors. I did my best to incorporate their suggestions, which often meant deleting material that experienced designers like me (aka old-timers) are perhaps too attached to, while greatly enhancing the coverage of modern concepts in HDL-based design flow, test benches, synthesis, and more.

My sponsoring editor at Pearson, Julie Bai, deserves thanks for shepherding this project over the past couple of years; she's my first editor who actually took a digital design course using a previous edition of this book. Unfortunately, she's also the fourth or fifth editor who has changed jobs after almost completing one of my book projects, convincing me that working with me inevitably leads to an editor's burnout or success or both. Special thanks go to her boss's boss, Marcia Horton, who has kept an eye on my projects for a couple of decades, and to Scott Disanno and Michelle Bayman, who guided the production and launch processes for this edition.

Thanks also go to artist Peter Crowell, whose paintings I discovered on Ebay when editor Julie Bai suggested we do a cover based on Piet Mondrian's work, some of which she said "almost looks like an abstract take on logic circuits." Crowell's "Tuesday Matinee" fits the bill beautifully. His painting is "tiled" on the cover and in the chapter-opening art in much the same way that logic blocks and interconnect are tiled in an FPGA. Our cover designer Marta Samsel took my engineering-ish concept and adapted it beautifully.

Finally, my wife Joanne Jacobs was very supportive of this project, letting me work in peace "upstairs" while she worked "downstairs" on her education blog. She didn't even complain that the Christmas tree was still up in February.

John F. Wakerly
Los Altos, California



chapter 1

Introduction

Welcome to the world of digital design. Perhaps you're a computer science student who knows all about computer software and programming, but you're still trying to figure out how all that fancy hardware could possibly work. Or perhaps you're an electrical engineering student who already knows something about analog electronics and circuit design, but you wouldn't know a bit if it bit you. No matter. Starting from a fairly basic level, this book will show you how to design digital circuits and subsystems.

We'll give you the basic principles that you need to figure things out, and we'll give you lots of examples. Along with principles, we'll try to convey the flavor of real-world digital design by discussing practical matters whenever possible. And I, the author, will often refer to myself as "we" in the hope that you'll be drawn in and feel that we're walking through the learning process together.

1.1 About Digital Design

Some people call it "logic design." That's OK, but ultimately the goal of design is to build systems. To that end, we'll cover a whole lot more in this text than logic equations and theorems.

This book claims to be about principles and practices. Most of the principles that we present will continue to be important years from now;

some may be applied in ways that have not even been discovered yet. As for practices, they are sure to be a little different from what's presented here by the time you start working in the field, and they will continue to change throughout your career. So you should treat the "practices" material in this book as a way to reinforce principles, and as a way to learn design methods by example.

One of the book's goals is to present enough about basic principles for you to know what's happening when you use software tools to "turn the crank" for you. The same basic principles can help you get to the root of problems when the tools happen to get in your way.

Listed in the box below are several key points that you should learn through your studies with this text. Many of these items may not make sense to you right now, but you can come back and review them later.

Digital design is engineering, and engineering means "problem solving." My experience is that only 5% to 10% of digital design is "the fun stuff"—the creative part of design, the flash of insight, the invention of a new approach. Much of the rest is just "turning the crank." To be sure, turning the crank is much

IMPORTANT THEMES IN DIGITAL DESIGN

- Good tools do not guarantee good design, but they help a lot by taking the pain out of doing things right.
- Digital circuits have analog characteristics.
- Know when to worry and when not to worry about the analog aspects of digital design.
- Transistors and all the digital components built with them are cheap and plentiful; make sensible trade-offs between minimizing the size of your designs and your engineering time.
- Always document your designs to make them understandable to yourself and to others.
- Use consistent coding, organizational, and documentation styles in your HDL-based designs, following your company's guidelines.
- Understand and use standard functional building blocks.
- State-machine design is like programming; approach it that way.
- Design for minimum cost at the system level, including your own engineering effort as part of the cost.
- Design for testability and manufacturability.
- Use programmable logic to simplify designs, reduce cost, and accommodate last-minute modifications.
- Avoid asynchronous design. Practice synchronous design until a better methodology comes along (if ever).
- Pinpoint the unavoidable asynchronous interfaces between different subsystems and the outside world, and provide reliable synchronizers.

easier now than it was 25 or even 10 years ago, but you still can't spend 100% or even 50% of your time on the fun stuff.

Besides the fun stuff and turning the crank, there are many other areas in which a successful digital designer must be competent, including the following:

- *Debugging.* It's next to impossible to be a good designer without being a good troubleshooter. Successful debugging takes planning, a systematic approach, patience, and logic: if you can't discover where a problem *is*, find out where it is *not*!
- *Business requirements and practices.* A digital designer's work is affected by a lot of non-engineering factors, including documentation standards, component availability, feature definitions, target specifications, task scheduling, office politics, and going to lunch with vendors.
- *Risk-taking*. When you begin a design project, you must carefully balance risks against potential rewards and consequences, in areas ranging from component selection (Will it be available when I'm ready to build the first prototype?) to schedule commitments (Will I still have a job if I'm late?).
- *Communication.* Eventually, you'll hand off your successful designs to other engineers, other departments, and customers. Without good communication skills, you'll never complete this step successfully. Keep in mind that communication includes not just transmitting but also receiving—learn to be a good listener!

In the rest of this chapter, and throughout the text, I'll continue to state some opinions about what's important and what is not. I think I'm entitled to do so as a moderately successful practitioner of digital design.

1.2 Analog versus Digital

Analog devices and systems process time-varying signals that can take on any value across a continuous range of voltage, current, or other measurable physical quantity. So do *digital* circuits and systems; the difference is that we can pretend that they don't! A digital signal is modeled as taking on, at any time, only one of two discrete values, which we call *0* and *1* (or LOW and HIGH, FALSE and TRUE, negated and asserted, Frank and Teri, or whatever).

analog

digital

0

1

Digital computers have been around since the 1940s, and they've been in widespread commercial use since the 1960s. Yet only in the past few decades has the "digital revolution" spread to many other aspects of life. Examples of once-analog systems that have now "gone digital" include the following:

- *Still pictures*. Twenty years ago, the majority of cameras still used silver-halide film to record images. Today, inexpensive digital cameras and smartphones record a picture as a 1920×1080 or larger array of pixels, where each pixel stores the intensities of its red, green, and blue color com-

ponents as 8 or more bits each. This data, almost 50 million bits in this example, is usually processed and compressed in JPEG format down to as few as 5% of the original number of bits. So, digital cameras rely on both digital storage and digital processing.

- *Video recordings*. "Films" are no longer stored on film. A Blu-ray disc (BD) stores video in a highly compressed digital format called MPEG-4. This standard compresses a small fraction of the individual video frames into a format similar to JPEG, and encodes each other frame as the difference between it and the previous one. The capacity of a dual-layer BD is about 400 billion bits, sufficient for about 2 hours of high-definition video.
- *Audio recordings*. Once made exclusively by impressing analog waveforms onto magnetic tape or vinyl, audio recordings are now made and delivered digitally, using a sequence of 16- to 24-bit values corresponding to samples of the original analog waveform, and up to 192,000 samples per second per audio channel. The number of bits, samples, and channels depends on the recording format; a compact disc (CD) stores two channels of 44,100 16-bit values for up to 73 minutes of stereo audio. Like a still picture or a video recording, an audio recording may be compressed for delivery to or storage on a device such as a smartphone, typically using a format called MP3.
- *Automobile carburetors*. Once controlled strictly by mechanical linkages (including clever "analog" mechanical devices that sensed temperature, pressure, etc.), automobile engines are now controlled by embedded microprocessors. Various electronic and electromechanical sensors convert engine conditions into numbers that the microprocessor can examine to determine how to control the flow of fuel and oxygen to the engine. The microprocessor's output is a time-varying sequence of numbers that operate electromechanical actuators which, in turn, control the engine.
- *The telephone system*. It started out over a hundred years ago with analog microphones and receivers connected to the ends of a pair of copper wires (or was it string?). Even today, many homes still use analog telephones, which transmit analog signals to the phone company's central office (CO). However, in the majority of COs, these analog signals are converted into a digital format before they are routed to their destinations, be they in the same CO or across the world. For many years, private branch exchanges (PBXs) used by businesses have carried the digital format all the way to the desktop. Now most businesses, COs, and traditional telephony service providers have converted to integrated systems that combine digital voice with data traffic over a single IP (Internet Protocol) network.
- *Traffic lights.* Stop lights used to be controlled by electromechanical timers that would give the green light to each direction for a predetermined amount of time. Later, relays were used in controllers that could activate

the lights according to the pattern of traffic detected by sensors embedded in the pavement. Today's controllers use microprocessors and can control the lights in ways that maximize vehicle throughput or, in Sunnyvale, California, frustrate drivers with all kinds of perverse behavior.

- *Movie effects.* Special effects used to be created exclusively with miniature clay models, stop action, trick photography, and numerous overlays of film on a frame-by-frame basis. Today, spaceships, cities, bugs, and monsters are synthesized entirely using digital computers. Even actors and actresses have been created or recreated using digital effects.

The electronics revolution has been going on for quite some time now, and the "solid-state" revolution began with analog devices and applications like transistors and transistor radios. So why has there now been a *digital* revolution? There are in fact many reasons to favor digital circuits over analog ones, including:

- *Reproducibility of results*. Given the same set of inputs (in both value and time sequence), a properly designed digital circuit always produces exactly the same results. The outputs of an analog circuit vary with temperature, power-supply voltage, component aging, and other factors.
- *Ease of design*. Digital design, often called "logic design," is logical. No special math skills are needed, and the behavior of small logic circuits can be mentally visualized without any special insights about the operation of capacitors, transistors, or other devices that require calculus to model.
- *Flexibility and functionality.* Once a problem has been reduced to digital form, it can be solved using a set of logical steps in space and time. For example, you can design a digital circuit that scrambles your recorded voice so it is absolutely indecipherable by anyone who does not have your "key" (password), but it can be heard virtually undistorted by anyone who does. Try doing that with an analog circuit.
- *Programmability.* You're probably already quite familiar with digital computers and the ease with which you can design, write, and debug programs for them. Well, guess what? Most of digital design is done today by writing "programs" too, in *hardware description languages (HDLs)*.

  hardware description language (HDL)

  While they're not "programming" languages in the sense of C++ or Java, HDLs allow both structure and function of a digital circuit to be specified or *modeled* with language-based constructs rather than a circuit diagram. Moreover, besides a compiler, an HDL also comes with simulation and synthesis programs that are used to test the hardware model's behavior before any real hardware is built, and then to synthesize the model into a circuit in a particular component technology. This saves a lot of work, because the synthesized circuit typically has a lot more detail than the model that generated it.

  hardware model

PROGRAMS, MODELS, MODULES, AND CODE

As you'll see throughout this text, Verilog HDL examples look a lot like "programs" and are even labeled as such. But generally they are *not* programs in the sense that C++ or Java programs execute a sequence of instructions to produce a result. Rather, they are *models* of hardware structures that receive input signals and produce output signals on wires, and that's something quite different. Since we'll show you hardware basics before we get into HDL models, you should be able to understand the difference when we get there. To help you, we will avoid calling an HDL model a "program."

Verilog can also be used to write procedural programs called "test benches" that do not model hardware. A test bench *exercises* a hardware model, applying a sequence of inputs to it and observing the resulting outputs, and we will actually sometimes call it a "program" and never a "model."

To model a piece of hardware, Verilog typically uses statements in a construct called a *module*, which may be stored in a single text file. We could call such a text file either a module or a model, and we will. However, a complex piece of hardware may be modeled hierarchically using *multiple* modules, so in that case, its model is a collection of modules.

If none of the above terms seems quite appropriate for describing a particular bit of Verilog, we may just call it Verilog "code," for lack of a better short term.

- *Speed.* Today's digital devices are very fast. Individual transistors in the fastest integrated circuits can switch in less than 10 picoseconds, and a complex circuit built from these transistors can examine its inputs and produce an output in less than a nanosecond. A device incorporating such circuits can produce a billion or more results per second.
- *Economy.* Digital circuits can provide a lot of functionality in a small space. Circuits that are used repetitively can be "integrated" into a single "chip" and mass-produced at very low cost, making possible throw-away items like calculators, digital watches, and singing birthday cards. (You may ask, "Is this such a good thing?" Never mind!)
- *Steadily advancing technology.* When you design a digital system, you almost always know that there will be a faster, cheaper, or otherwise better technology for it in a few years. Clever designers can accommodate these expected advances during the initial design of a system, to forestall system obsolescence and to add value for customers. For example, desktop computers often have "expansion sockets" to accommodate faster processors or larger memories than are available at the time of the computer's introduction.

So, that's enough of a sales pitch on digital design. The rest of this chapter will give you a bit more technical background to prepare you for the rest of the book.

SHORT TIMES A *millisecond (ms)* is 10^{-3} second, and a *microsecond (μs)* is 10^{-6} second. A *nanosecond (ns)* is just 10^{-9} second, and a *picosecond (ps)* is 10^{-12} second. In a vacuum, light travels about a foot in a nanosecond, and an inch in 85 picoseconds. With individual transistors in the fastest integrated circuits now switching in less than 10 picoseconds, the speed-of-light delay between these transistors across a half-inch-square silicon chip has become a limiting factor in circuit design.

1.3 Analog Signals

Marketing hype notwithstanding, we live in an analog world, not a digital one. Voltages, currents, and other physical quantities in real circuits take on values that are infinitely variable, depending on properties of the real devices that comprise the circuits. Because real values are continuously variable, we could use a physical quantity such as a signal voltage in a circuit to represent a real number (e.g., 3.14159265358979 volts represents the mathematical constant *pi* to 14 decimal digits of precision).

However, stability and accuracy in physical quantities are difficult to obtain in real circuits. They can be affected by manufacturing variations, temperature, power-supply voltage, cosmic rays, and noise created by other circuits, among other things. If we used an analog voltage to represent *pi*, we might find that instead of being an absolute mathematical constant, *pi* varied over a range of 10% or more.

Also, many mathematical and logical operations can be difficult or impossible to perform with analog quantities. While it is possible with some cleverness to build an analog circuit whose output voltage is the square root of its input voltage, no one has ever built a 100-input, 100-output analog circuit whose outputs are a set of voltages identical to the set of input voltages, but sorted arithmetically.

1.4 Digital Logic Signals

digital logic
digital signals
logic values

Digital logic hides the pitfalls of the analog world by using *digital signals*, where the infinite set of real values for a physical quantity are mapped into two subsets corresponding to just two possible numbers or *logic values*: 0 and 1. Thus, digital logic circuits can be analyzed and designed functionally, using switching algebra, tables, and other abstract means to describe the operation of well-behaved 0s and 1s in a circuit.

binary digit
bit

A logic value, 0 or 1, is often called a *binary digit*, or *bit*. If an application requires more than two discrete values, additional bits may be used, with a set of n bits representing 2^n different values.

Examples of the physical phenomena used to represent bits in some modern (and not-so-modern) digital technologies are given in Table 1-1. With

Table 1-1 Physical states representing bits in different logic and memory technologies.

Technology	*State Representing Bit*	
	0	*1*
Pneumatic logic	Fluid at low pressure	Fluid at high pressure
Relay logic	Circuit open	Circuit closed
Transistor-transistor logic (TTL)	0–0.8 V	2.0–5.0 V
Complementary metal-oxide semiconductor (CMOS) 2-volt logic	0–0.5 V	1.5–2.0 V
Dynamic memory	Capacitor discharged	Capacitor charged
Nonvolatile, erasable memory	Electrons trapped	Electrons released
On-chip nonvolatile security key	Fuse blown	Fuse intact
Polymer memory	Molecule in state A	Molecule in state B
Fiber optics	Light off	Light on
Magnetic disk or tape	Flux direction "north"	Flux direction "south"
Compact disc (CD), digital versatile disc (DVD), and Blu-ray disc (BD)	No pit	Pit
Writable compact disc (CD-R)	Dye in crystalline state	Dye in noncrystalline state

most phenomena, there is an undefined region between the 0 and 1 states (e.g., voltage = 1.0 V, dim light, capacitor slightly charged, etc.). This undefined region is needed so the 0 and 1 states can be unambiguously defined and reliably detected. Noise can more easily corrupt results if the boundaries separating the 0 and 1 states are too close to each other.

LOW
HIGH

When discussing electronic logic circuits like CMOS, digital designers often use the words "LOW" and "HIGH" in place of "0" and "1" to remind them that they are dealing with real circuits, not abstract quantities:

LOW A signal in the range of algebraically lower voltages, which is interpreted as a logic 0.

HIGH A signal in the range of algebraically higher voltages, which is interpreted as a logic 1.

STATE TRANSITIONS

The last four technologies in Table 1-1 don't actually use absolute states to represent bit values. Rather, they use transitions (or absence of transitions) between states to represent 0s and 1s using a code such as the Manchester code described on page 82.

THE DIGITAL ABSTRACTION

Digital circuits are not exactly a binary version of alphabet soup—with all due respect to our forthcoming descriptions like Figure 1-3, digital circuits don't have little 0s and 1s floating around in them. As you'll see in Chapter 14, digital circuits deal with analog voltages and currents and are built with analog components. The "digital abstraction" allows analog behavior to be ignored in most cases, so circuits can be modeled as if they really did process 0s and 1s.

Note that the assignments of 0 and 1 to LOW and HIGH are somewhat arbitrary. Still, assigning 0 to LOW and 1 to HIGH seems natural and is called *positive logic*, and that's what we use in this book exclusively. The opposite assignment, 1 to LOW and 0 to HIGH, is not often used and is called *negative logic*.

positive logic
negative logic

Because a wide range of physical values represent the same binary value, digital logic is highly immune to component and power-supply variations and noise. Furthermore, *buffer* circuits can be used to regenerate (or amplify) "weak" values into "strong" ones, so that digital signals can be transmitted over arbitrary distances without loss of information. For example, using the voltage ranges in the fourth row of Table 1-1, a buffer for 2-volt CMOS logic converts any LOW input voltage into an output very close to 0.0 V, and any HIGH input voltage into an output very close to 2.0 V.

buffer

1.5 Logic Circuits and Gates

A logic circuit can be represented with a minimum amount of detail simply as a "black box" with a certain number of inputs and outputs. For example, Figure 1-1 shows a logic circuit with three inputs and one output. However, this representation does not describe how the circuit responds to input signals.

From the point of view of electronic circuit design, it takes a lot of information to describe the precise electrical behavior of a circuit. However, since the inputs of a digital logic circuit can be viewed as taking on only discrete 0 and 1 values, the circuit's "logical" operation can be described with a table that ignores electrical behavior and lists only discrete 0 and 1 values.

A logic circuit whose outputs depend only on its current inputs is called a *combinational circuit*. Its operation is fully described by a *truth table* that lists all combinations of input values and the output value(s) produced by each one.

combinational circuit
truth table

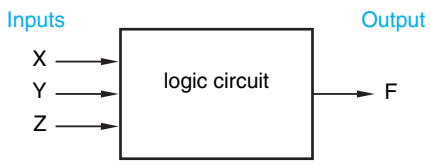


Figure 1-1 "Black-box" representation of a 3-input, 1-output logic circuit.