



Peter J. Ashenden

Digital Design

An Embedded
Systems Approach Using
VHDL

MK[®]
MORGAN KAUFMANN

In Praise of *Digital Design: An Embedded Systems Approach Using VHDL*

“Peter Ashenden is leading the way towards a new curriculum for educating the next generation of digital logic designers. Recognizing that digital design has moved from being gate-centric assembly of custom logic to processor-centric design of embedded systems, Dr. Ashenden has shifted the focus from the gate to the modern design and integration of complex integrated devices that may be physically realized in a variety of forms. Dr. Ashenden does not ignore the fundamentals, but treats them with suitable depth and breadth to provide a basis for the higher-level material. As is the norm in all of Dr. Ashenden’s writing, the text is lucid and a pleasure to read. The book is illustrated with copious examples and the companion Web site offers all that one would expect in a text of such high quality.”

— GRANT MARTIN, Chief Scientist, Tensilica Inc.

“Dr. Ashenden has written a textbook that enables students to obtain a much broader and more valuable understanding of modern digital system design. Readers can be sure that the practices described in this book will provide a strong foundation for modern digital system design using hardware description languages.”

— GARY SPIVEY, George Fox University

“The convergence of miniaturized, sophisticated electronics into handheld, low-power embedded systems such as cell phones, PDAs, and MP3 players depends on efficient, digital design flows. Starting with an intuitive exploration of the basic building blocks, Digital Design: An Embedded Systems Approach introduces digital systems design in the context of embedded systems to provide students with broader perspectives. Throughout the text, Peter Ashenden’s practical approach engages students in understanding the challenges and complexities involved in implementing embedded systems.”

— GREGORY D. PETERSON, University of Tennessee

“Digital Design: An Embedded Systems Approach places emphasis on larger systems containing processors, memory, and involving the design

and interface of I/O functions and application-specific accelerators. The book's presentation is based on a contemporary view that reflects the real-world digital system design practice. At a time when the university curriculum is generally lagging significantly behind industry development, this book provides much needed information to students in the areas of computer engineering, electrical engineering and computer science."

— DONALD HUNG, San Jose State University

"Digital Design: An Embedded Systems Approach presents the design flow of circuits and systems in a way that is both accessible and up-to-date. Because the use of hardware description languages is state-of-the-art, it is necessary that students learn how to use these languages along with an appropriate methodology. This book presents a modern approach for designing embedded systems starting with the fundamentals and progressing up to a complete system—it is application driven and full of many examples. I will recommend this book to my students."

— GOERAN HERRMANN, TU Chemnitz

"Digital Design: An Embedded Systems Approach is surprisingly easy to read despite the complexity of the material. It takes the reader in a journey from the basics to a real understanding of digital design by answering the 'whys' and 'hows'—it is persuasive and instructive as it moves deeper and deeper into the material."

— ANDREY KOPTYUG, Mid Sweden University

"This up-to-date text on digital design is written in a very accessible style using a modern design methodology and the real world of embedded systems as its contexts. Digital Design: An Embedded Systems Approach provides excellent coverage of all aspects of the design of embedded systems, with chapters not just on logic design itself, but also on processors, memories, input/output interfacing and implementation technologies. It's particularly good at emphasizing the need to consider more than just logic design when designing a digital system: the design has to be implemented in the real world of engineering, where a whole variety of constraints, such as circuit area, circuit interconnections, interfacing requirements, power and performance, must be considered. For those who think logic design is mundane, this book brings the subject to life."

— ROLAND IBBETT, University of Edinburgh

Digital Design
An Embedded Systems Approach
Using VHDL

ABOUT THE AUTHOR

Peter J. Ashenden is an Adjunct Associate Professor at Adelaide University and the founder of Ashenden Designs, a consulting business specializing in electronics design automation (EDA).

From 1990 to 2000, Dr. Ashenden was a member of the faculty in the Department of Computer Science at Adelaide. He developed curriculum and taught in a number of areas for both the Computer Science and the Electrical and Electronic Engineering departments. Topics included computer organization, computer architecture, digital logic design, programming and algorithms, at all levels from undergraduate to graduate courses. He was also actively involved in academic administration at many levels within the university.

In 2000, Dr. Ashenden established Ashenden Designs. His services include training development and delivery, advising on design methodology, research in EDA tool technology, development of design languages, and standards writing. His clients include industry and government organization in the United States, Europe and SE Asia.

Since 1992, Dr. Ashenden has been involved in the IEEE VHDL standards committees, and continues to play a major role in ongoing development of the language. From 2003 to 2005 he was Chair of the IEEE Design Automation Standards Committee, which oversees development of all IEEE standards in EDA. He is currently Technical Editor for the VHDL, VHDL-AMS, and Rosetta specification language standards.

In addition to his research publications, Dr. Ashenden is author of *The Designer's Guide to VHDL* and *The Student's Guide to VHDL*, and coauthor of *The System Designer's Guide to VHDL-AMS* and *VHDL-2007: Just the New Stuff*. His VHDL books are highly regarded and are the best-selling references on the subject. From 2000 to 2004, he was Series Coeditor of the Morgan Kaufmann Series on Systems on Silicon, and from 2001 to 2004 he was a member of the Editorial Board of *IEEE Design and Test of Computers* magazine.

Dr. Ashenden is a Senior Member of the IEEE and the IEEE Computer Society. He is also a volunteer Senior Firefighter of 12 years standing with the South Australian Country Fire Service.

Digital Design An Embedded Systems Approach Using VHDL

PETER J. ASHENDEN

*Adjunct Associate Professor
School of Computer Science
University of Adelaide*



AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann is an imprint of Elsevier



MORGAN KAUFMANN PUBLISHERS

<i>Publishing Director</i>	Joanne Tracy
<i>Publisher</i>	Denise E. M. Penrose
<i>Acquisitions Editor</i>	Charles Glaser
<i>Publishing Services Manager</i>	George Morrison
<i>Senior Production Editor</i>	Dawnmarie Simpson
<i>Developmental Editor</i>	Nate McFadden
<i>Editorial Assistant</i>	Kimberlee Honjo
<i>Production Assistant</i>	Lianne Hong
<i>Cover Design</i>	Eric DeCicco
<i>Cover Image</i>	Getty Images
<i>Composition</i>	diacriTech
<i>Technical Illustration</i>	Peter Ashenden
<i>Copyeditor</i>	JC Publishing
<i>Proofreader</i>	Janet Cocker
<i>Indexer</i>	Joan Green
<i>Interior printer</i>	Sheridan Books, Inc.
<i>Cover printer</i>	Phoenix Color, Inc.

Morgan Kaufmann Publishers is an imprint of Elsevier.
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA

This book is printed on acid-free paper.

© 2008 by Elsevier Inc. All rights reserved.

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, scanning, or otherwise—without prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, E-mail: permissions@elsevier.com. You may also complete your request online via the Elsevier homepage (<http://elsevier.com>), by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."

Library of Congress Cataloging-in-Publication Data

Ashenden, Peter J.

Digital design: an embedded systems approach using VHDL / Peter J. Ashenden.
p. cm.

Includes index.

ISBN 978-0-12-369528-4 (pbk. : alk. paper) 1. Embedded computer systems. 2. VHDL (Computer hardware description language) 3. System design. I. Title.
TK7895.E42.A69 2007
621.39'16—dc22

2007023241

ISBN: 978-0-12-369528-4

For information on all Morgan Kaufmann publications,
visit our Web site at www.mkp.com or www.books.elsevier.com

Printed in the United States.

07 08 09 10 5 4 3 2 1

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER

BOOK AID
International

Sabre Foundation

To my daughter, Eleanor

—PA

This page intentionally left blank

CONTENTS

Preface	xv
CHAPTER 1 Introduction and Methodology	1
1.1 Digital Systems and Embedded Systems	1
1.2 Binary Representation and Circuit Elements	4
1.3 Real-World Circuits	9
1.3.1 <i>Integrated Circuits</i>	10
1.3.2 <i>Logic Levels</i>	11
1.3.3 <i>Static Load Levels</i>	13
1.3.4 <i>Capacitive Load and Propagation Delay</i>	15
1.3.5 <i>Wire Delay</i>	17
1.3.6 <i>Sequential Timing</i>	17
1.3.7 <i>Power</i>	18
1.3.8 <i>Area and Packaging</i>	19
1.4 Models	21
1.5 Design Methodology	27
1.5.1 <i>Embedded Systems Design</i>	31
1.6 Chapter Summary	34
1.7 Further Reading	35
Exercises	36
CHAPTER 2 Combinational Basics	39
2.1 Boolean Functions and Boolean Algebra	39
2.1.1 <i>Boolean Functions</i>	39
2.1.2 <i>Boolean Algebra</i>	48
2.1.3 <i>VHDL Models of Boolean Equations</i>	51
2.2 Binary Coding	55
2.2.1 <i>Using Vectors for Binary Codes</i>	57
2.2.2 <i>Bit Errors</i>	59
2.3 Combinational Components and Circuits	62
2.3.1 <i>Decoders and Encoders</i>	62
2.3.2 <i>Multiplexers</i>	69
2.3.3 <i>Active-Low Logic</i>	72
2.4 Verification of Combinational Circuits	75
2.5 Chapter Summary	82

2.6	Further Reading	83
	Exercises	84
CHAPTER 3 Numeric Basics		89
3.1	Unsigned Integers	89
3.1.1	<i>Coding Unsigned Integers</i>	89
3.1.2	<i>Operations on Unsigned Integers</i>	94
3.1.3	<i>Gray Codes</i>	119
3.2	Signed Integers	122
3.2.1	<i>Coding Signed Integers</i>	122
3.2.2	<i>Operations on Signed Integers</i>	125
3.3	Fixed-Point Numbers	134
3.3.1	<i>Coding Fixed-Point Numbers</i>	134
3.3.2	<i>Operations on Fixed-Point Numbers</i>	138
3.4	Floating-Point Numbers	141
3.4.1	<i>Coding Floating-Point Numbers</i>	141
3.4.2	<i>Floating-Point Representation in VHDL</i>	144
3.5	Chapter Summary	148
3.6	Further Reading	149
	Exercises	150
CHAPTER 4 Sequential Basics		157
4.1	Storage Elements	157
4.1.1	<i>Flip-flops and Registers</i>	157
4.1.2	<i>Shift Registers</i>	168
4.1.3	<i>Latches</i>	170
4.2	Counters	175
4.3	Sequential Datapaths and Control	184
4.3.1	<i>Finite-State Machines</i>	189
4.4	Clocked Synchronous Timing Methodology	196
4.4.1	<i>Asynchronous Inputs</i>	201
4.4.2	<i>Verification of Sequential Circuits</i>	206
4.4.3	<i>Asynchronous Timing Methodologies</i>	210
4.5	Chapter Summary	213
4.6	Further Reading	214
	Exercises	215
CHAPTER 5 Memories		221
5.1	General Concepts	221
5.2	Memory Types	229
5.2.1	<i>Asynchronous Static RAM</i>	230
5.2.2	<i>Synchronous Static RAM</i>	232

5.2.3	<i>Multiport Memories</i>	239
5.2.4	<i>Dynamic RAM</i>	244
5.2.5	<i>Read-Only Memories</i>	246
5.3	Error Detection and Correction	250
5.4	Chapter Summary	255
5.5	Further Reading	256
	Exercises	257
CHAPTER 6 Implementation Fabrics		261
6.1	Integrated Circuits	261
6.1.1	<i>Integrated Circuit Manufacture</i>	262
6.1.2	<i>SSI and MSI Logic Families</i>	264
6.1.3	<i>Application-Specific Integrated Circuits (ASICs)</i> ...	267
6.2	Programmable Logic Devices	270
6.2.1	<i>Programmable Array Logic</i>	270
6.2.2	<i>Complex PLDs</i>	274
6.2.3	<i>Field-Programmable Gate Arrays</i>	275
6.3	Packaging and Circuit Boards	281
6.4	Interconnection and Signal Integrity	284
6.4.1	<i>Differential Signaling</i>	288
6.5	Chapter Summary	290
6.6	Further Reading	291
	Exercises	292
CHAPTER 7 Processor Basics		293
7.1	Embedded Computer Organization	293
7.1.1	<i>Microcontrollers and Processor Cores</i>	295
7.2	Instructions and Data	297
7.2.1	<i>The Gumnut Instruction Set</i>	299
7.2.2	<i>The Gumnut Assembler</i>	308
7.2.3	<i>Instruction Encoding</i>	310
7.2.4	<i>Other CPU Instruction Sets</i>	312
7.3	Interfacing with Memory	314
7.3.1	<i>Cache Memory</i>	320
7.4	Chapter Summary	323
7.5	Further Reading	323
	Exercises	324
CHAPTER 8 I/O Interfacing		327
8.1	I/O Devices	327
8.1.1	<i>Input Devices</i>	328
8.1.2	<i>Output Devices</i>	333

8.2	I/O Controllers	342
8.2.1	<i>Simple I/O Controllers</i>	343
8.2.2	<i>Autonomous I/O Controllers</i>	348
8.3	Parallel Buses	350
8.3.1	<i>Multiplexed Buses</i>	351
8.3.2	<i>Tristate Buses</i>	355
8.3.3	<i>Open-Drain Buses</i>	360
8.3.4	<i>Bus Protocols</i>	362
8.4	Serial Transmission	365
8.4.1	<i>Serial Transmission Techniques</i>	365
8.4.2	<i>Serial Interface Standards</i>	369
8.5	I/O Software	372
8.5.1	<i>Polling</i>	373
8.5.2	<i>Interrupts</i>	374
8.5.3	<i>Timers</i>	380
8.6	Chapter Summary	387
8.7	Further Reading	388
	Exercises	389
CHAPTER 9 Accelerators		393
9.1	General Concepts	393
9.2	Case Study: Video Edge-Detection	400
9.3	Verifying an Accelerator	423
9.4	Chapter Summary	435
9.5	Further Reading	435
	Exercises	436
CHAPTER 10 Design Methodology		439
10.1	Design Flow	439
10.1.1	<i>Architecture Exploration</i>	441
10.1.2	<i>Functional Design</i>	443
10.1.3	<i>Functional Verification</i>	445
10.1.4	<i>Synthesis</i>	451
10.1.5	<i>Physical Design</i>	454
10.2	Design Optimization	457
10.2.1	<i>Area Optimization</i>	458
10.2.2	<i>Timing Optimization</i>	460
10.2.3	<i>Power Optimization</i>	464
10.3	Design for Test	467
10.3.1	<i>Fault Models and Fault Simulation</i>	468
10.3.2	<i>Scan Design and Boundary Scan</i>	470
10.3.3	<i>Built-In Self Test (BIST)</i>	474

10.4	Nontechnical Issues	478
10.5	In Conclusion	479
10.6	Chapter Summary	481
10.7	Further Reading	482
APPENDIX A Knowledge Test Quiz Answers		485
APPENDIX B Introduction to Electronic Circuits		517
B.1	Components	517
B.1.1	<i>Voltage Sources</i>	518
B.1.2	<i>Resistors</i>	518
B.1.3	<i>Capacitors</i>	519
B.1.4	<i>Inductors</i>	519
B.1.5	<i>MOSFETs</i>	520
B.1.6	<i>Diodes</i>	522
B.1.7	<i>Bipolar Transistors</i>	523
B.2	Circuits	524
B.2.1	<i>Kirchhoff's Laws</i>	524
B.2.2	<i>Series and Parallel R, C, and L</i>	525
B.2.3	<i>RC Circuits</i>	527
B.2.4	<i>RLC Circuits</i>	528
B.3	Further Reading	531
APPENDIX C VHDL for Synthesis		533
C.1	Data Types and Operations	533
C.2	Combinational Functions	534
C.3	Sequential Circuits	539
C.3.1	<i>Finite-State Machines</i>	543
C.4	Memories	545
APPENDIX D The Gumnut Microcontroller Core		547
D.1	The Gumnut Instruction Set	547
D.1.1	<i>Arithmetic and Logical Instructions</i>	547
D.1.2	<i>Shift Instructions</i>	551
D.1.3	<i>Memory and I/O Instructions</i>	552
D.1.4	<i>Branch Instructions</i>	553
D.1.5	<i>Jump Instructions</i>	553
D.1.6	<i>Miscellaneous Instructions</i>	554
D.2	The Gumnut Bus Interface	554
Index		557

This page intentionally left blank

P R E F A C E

A P P R O A C H

This book provides a foundation in digital design for students in computer engineering, electrical engineering and computer science courses. It deals with digital design as an activity in a larger systems design context. Instead of focusing on gate-level design and aspects of digital design that have diminishing relevance in a real-world design context, the book concentrates on modern and evolving knowledge and design skills.

Most modern digital design practice involves design of embedded systems, using small microcontrollers, larger CPUs/DSPs, or hard or soft processor cores. Designs involve interfacing the processor or processors to memory, I/O devices and communications interfaces, and developing accelerators for operations that are too computationally intensive for processors. Target technologies include ASICs, FPGAs, PLDs and PCBs. This is a significant change from earlier design styles, which involved use of small-scale integrated (SSI) and medium-scale integrated (MSI) circuits. In such systems, the primary design goal was to minimize gate count or IC package count. Since processors had lower performance and memories had limited capacity, a greater proportion of system functionality was implemented in hardware.

While design practices and the design context have evolved, many textbooks have not kept track. They continue to promote practices that are largely obsolete or that have been subsumed into computer-aided design (CAD) tools. They neglect many of the important considerations for modern designers. This book addresses the shortfall by taking an approach that embodies modern design practice. The book presents the view that digital logic is a basic abstraction over analog electronic circuits. Like any abstraction, the digital abstraction relies on assumptions being met and constraints being satisfied. Thus, the book includes discussion of the electrical and timing properties of circuits, leading to an understanding of how they influence design at higher levels of abstraction. Also, the book teaches a methodology based on using abstraction to manage complexity, along with principles and methods for making design trade-offs. These intellectual tools allow students to track evolving design practice after they graduate.

Perhaps the most noticeable difference between this book and its predecessors is the omission of material on Karnaugh maps and related

logic optimization techniques. Some reviewers of the manuscript argued that such techniques are still of value and are a necessary foundation for students learning digital design. Certainly, it is important for students to understand that a given function can be implemented by a variety of equivalent circuits, and that different implementations may be more or less optimal under different constraints. This book takes the approach of presenting Boolean algebra as the foundation for gate-level circuit transformation, but leaves the details of algorithms for optimization to CAD tools. The complexity of modern systems makes it more important to raise the level of abstraction at which we work and to introduce embedded systems early in the curriculum. CAD tools perform a much better job of gate-level optimization than we can do manually, using advanced algorithms to satisfy relevant constraints. Techniques such as Karnaugh maps do have a place, for example, in design of specialized hazard-free logic circuits. Thus, students can defer learning about Karnaugh maps until an advanced course in VLSI, or indeed, until they encounter the need in industrial practice. A web search will reveal many sources describing the techniques in detail, including an excellent article in Wikipedia.

The approach taken in this book makes it relevant to Computer Science courses, as well as to Computer Engineering and Electrical Engineering courses. By treating digital design as part of embedded systems design, the book will provide the understanding of hardware needed for computer science students to analyze and design systems comprising both hardware and software components. The principles of abstraction and complexity management using abstraction presented in the book are the same as those underlying much of computer science and software engineering.

Modern digital design practice relies heavily on models expressed in hardware description languages (HDLs), such as Verilog and VHDL. HDL models are used for design entry at the abstract behavioral level and for refinements at the register transfer level. Synthesis tools produce gate-level HDL models for low-level verification. Designers also express verification environments in HDLs. This book emphasizes HDL-based design and verification at all levels of abstraction. The present version uses VHDL for this purpose. A second version, *Digital Design: An Embedded Systems Approach Using Verilog*, substitutes Verilog for the same purpose.

OVERVIEW

For those who are musically inclined, the organization of this book can be likened to a two-act opera, complete with overture, intermezzo, and finale.

Chapter 1 forms the overture, introducing the themes that are to follow in the rest of the work. It starts with a discussion of the basic ideas of the digital abstraction, and introduces the basic digital circuit elements.

It then shows how various non-ideal behaviors of the elements impose constraints on what we can design. The chapter finishes with a discussion of a systematic process of design, based on models expressed in a hardware description language.

Act I of the opera comprises Chapters 2 through 5. In this act, we develop the themes of basic digital design in more detail.

Chapter 2 focuses on combinational circuits, starting with Boolean algebra as the theoretical underpinning and moving on to binary coding of information. The chapter then surveys a range of components that can be used as building blocks in larger combinational circuits, before returning to the design methodology to discuss verification of combinational circuits.

Chapter 3 expands in some detail on combinational circuits used to process numeric information. It examines various binary codes for unsigned integers, signed integers, fixed-point fractions and floating-point real numbers. For each kind of code, the chapter describes how some arithmetic operations can be performed and looks at combinational circuits that implement arithmetic operations.

Chapter 4 introduces a central theme of digital design, sequential circuits. The chapter examines several sequential circuit elements for storing information and for counting events. It then describes the concepts of a datapath and a control section, followed by a description of the clocked synchronous timing methodology.

Chapter 5 completes Act I, describing the use of memories for storing information. It starts by introducing the general concepts that are common to all kinds of semiconductor memory, and then focuses on the particular features of each type, including SRAM, DRAM, ROM and flash memories. The chapter finishes with a discussion of techniques for dealing with errors in the stored data.

The intermezzo, Chapter 6, is a digression away from functional design into physical design and the implementation fabrics used for digital systems. The chapter describes the range of integrated circuits that are used for digital systems, including ASICs, FPGAs and other PLDs. The chapter also discusses some of the physical and electrical characteristics of implementation fabrics that give rise to constraints on designs.

Act II of the opera, comprising Chapters 7 through 9, develops the embedded systems theme.

Chapter 7 introduces the kinds of processors that are used in embedded systems and gives examples of the instructions that make up embedded software programs. The chapter also describes the way instructions and data are encoded in binary and stored in memory and examines ways of connecting the processor with memory components.

Chapter 8 expands on the notion of input/output (I/O) controllers that connect an embedded computer system with devices that sense and

affect real-world physical properties. It describes a range of devices that are used in embedded computers and shows how they are accessed by an embedded processor and by embedded software.

Chapter 9 describes accelerators, that is, components that can be added to embedded systems to perform operations faster than is possible with embedded software running on a processor core. This chapter uses an extended example to illustrate design considerations for accelerators, and shows how an accelerator interacts with an embedded processor.

The finale, Chapter 10, is a coda that returns to the theme of design methodology introduced in Chapter 1. The chapter describes details of the design flow and discusses how aspects of the design can be optimized to better meet constraints. It also introduces the concept of design for test, and outlines some design for test tools and techniques. The opera finishes with a discussion of the larger context in which digital systems are designed.

After a performance of an opera, there is always a lively discussion in the foyer. This book contains a number of appendices that correspond to that aspect of the opera. Appendix A provides sample answers for the Knowledge Test Quiz sections in the main chapters. Appendix B provides a quick refresher on electronic circuits. Appendix C is a summary of the subset of VHDL used for synthesis of digital circuits. Finally, Appendix D is an instruction-set reference for the Gumnut embedded processor core used in examples in Chapters 7 through 9.

For those not inclined toward classical music, I apologize if the preceding is not a helpful analogy. An analogy with the courses of a feast came to mind, but potential confusion among readers in different parts of the world over the terms appetizer, entrée and main course make the analogy problematic. The gastronomically inclined reader should feel free to find the correspondence in accordance with local custom.

COURSE ORGANIZATION

This book covers the topics included in the Digital Logic knowledge area of the Computer Engineering Body of Knowledge described in the IEEE/ACM *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*. The book is appropriate for a course at the sophomore level, assuming only previous introductory courses in electronic circuits and computer programming. It articulates into junior and senior courses in embedded systems, computer organization, VLSI and other advanced topics.

For a full sequence in digital design, the chapters of the book can be covered in order. Alternatively, a shorter sequence could draw on Chapter 1 through Chapter 6 plus Chapter 10. Such a sequence would defer material in Chapters 7 through 9 to a subsequent course on embedded systems design.

For either sequence, the material in this book should be supplemented by a reference book on the VHDL language. The course work should also include laboratory projects, since hands-on design is the best way to reinforce the principles presented in the book.

WEB SUPPLEMENTS

No textbook can be complete nowadays without supplementary material on a website. For this book, resources for students and instructors are available at the website:

textbooks.elsevier.com/9780123695284

For students, the website contains:

- ▶ Source code for all of the example HDL models in the book
- ▶ Tutorials on the VHDL and Verilog hardware description languages
- ▶ An assembler for the Gumnut processor described in Chapter 7 and Appendix D
- ▶ A link to the ISE WebPack FPGA EDA tool suite from Xilinx
- ▶ A link to the ModelSim Xilinx Edition III VHDL and Verilog simulator from Mentor Graphics Corporation
- ▶ A link to an evaluation edition of the Synplify Pro PFGA synthesis tool from Synplicity, Inc. (see inside back cover for more details).
- ▶ Tutorials on use of the EDA tools for design projects

For instructors, the website contains a protected area with additional resources:

- ▶ An instructor's manual
- ▶ Suggested lab projects
- ▶ Lecture notes
- ▶ Figures from the text in JPG and PPT formats

Instructors are invited to contribute additional material for the benefit of their colleagues.

Despite the best efforts of all involved, some errors have no doubt crept through the review and editorial process. A list of detected errors will be available accumulated on the website mentioned above. Should you detect such an error, please check whether it has been previously recorded. If not, I would be grateful for notice by email to

peter@ashenden.com.au

I would also be delighted to hear feedback about the book and supplementary material, including suggestions for improvement.

ACKNOWLEDGMENTS

This book arose from my long-standing desire to bring a more modern approach to the teaching of digital design. I am deeply grateful to the folks at Morgan Kaufmann Publishers for supporting me in realizing this goal, and for their guidance and advice in shaping the book. Particular thanks go to Denise Penrose, Publisher; Nate McFadden, Developmental Editor and Kim Honjo, Editorial Assistant. Thanks also to Dawnmarie Simpson at Elsevier for meticulous attention to detail and for making the production process go like clockwork.

The manuscript benefited from comprehensive reviews by Dr. A. Bouridane, Queen's University Belfast; Prof. Goeran Herrmann, Chemnitz University of Technology; Prof. Donald Hung, San Jose State University; Prof. Roland Ibbett, University of Edinburgh; Dr. Andrey Koptug, Mid Sweden University; Dr. Grant Martin, Tensilica, Inc.; Dr. Gregory D. Pæerson, University of Tennessee; Brian R. Prasky, IBM; Dr. Gary Spivey, George Fox University; Dr. Peixin Zhong, Michigan State University; and an anonymous reviewer from Rensselaer Polytechnic Institute. Also, my esteemed colleague Jim Lewis of SynthWorks Design, Inc., provided technical reviews of the VHDL code and related text. To all of these, my sincere thanks for their contributions. The immense improvement from my first draft to the final draft is due to their efforts.

The book and the associated teaching materials also benefited from field testing: in alpha form by myself at the University of Adelaide and by Dr. Monte Tull at The University of Oklahoma; and in beta form by James Sterbenz at The University of Kansas. To them and to their students, thanks for your forbearance with the errors and for your valuable feedback.

INTRODUCTION AND METHODOLOGY

1

This first chapter introduces some of the fundamental ideas underlying design of modern digital systems. We cover quite a lot of ground, but at a fairly basic level. The idea is to set the context for more detailed discussion in subsequent chapters.

We start by looking at the basic circuit elements from which digital systems are built, and seeing some of the ways in which they can be put together to perform a required function. We also consider some of the nonideal effects that we need to keep in mind, since they impose constraints on what we can design. We then focus on a systematic process of design, based on models expressed in a hardware description language. Approaching the design process systematically allows us to develop complex systems that meet modern application requirements.

1.1 DIGITAL SYSTEMS AND EMBEDDED SYSTEMS

This book is about digital design. Let's take a moment to explore those two words. *Digital* refers to electronic circuits that represent information in a special way, using just two voltage levels. The main rationale for doing this is to increase the reliability and accuracy of the circuits, but as we will see, there are numerous other benefits that flow from the digital approach. We also use the term *logic* to refer to digital circuits. We can think of the two voltage levels as representing truth values, leading us to use rules of logic to analyze digital circuits. This gives us a strong mathematical foundation on which to build. The word *design* refers to the systematic process of working out how to construct circuits that meet given requirements while satisfying constraints on cost, performance, power consumption, size, weight and other properties. In this book, we focus on the design aspects and build a methodology for designing complex digital systems.

Digital circuits have quite a long and interesting history. They were preceded by mechanical systems, electromechanical systems, and analog electronic systems. Most of these systems were used for numeric computations in business and military applications, for example, in ledger calculations and in computing ballistics tables. However, they suffered from numerous disadvantages, including inaccuracy, low speed, and high maintenance.

Early digital circuits, built in the mid-twentieth century, were constructed with relays. The contacts of a relay are either open, blocking current flow, or closed, allowing current to flow. Current controlled in this manner by one or more relays could then be used to switch other relays. However, even though relay-based systems were more reliable than their predecessors, they still suffered from reliability and performance problems.

The advent of digital circuits based on vacuum tubes and, subsequently, transistors led to major improvements in reliability and performance. However, it was the invention of the *integrated circuit* (IC), in which multiple transistors were fabricated and connected together, that really enabled the “digital revolution.” As manufacturing technology has developed, the size of transistors and the interconnecting wires has shrunk. This, along with other factors, has led to ICs, containing billions of transistors and performing complex functions, becoming commonplace now.

At this point, you may be wondering how such complex circuits can be designed. In your electronic circuits course, you may have learned how transistors operate, and that their operation is determined by numerous parameters. Given the complexity of designing a small circuit containing a few transistors, how could it be possible to design a large system with billions of transistors?

The key is *abstraction*. By abstraction, we mean identifying aspects that are important to a task at hand, and hiding details of other aspects. Of course, the other aspects can’t be ignored arbitrarily. Rather, we make assumptions and follow disciplines that allow us to ignore those details while we focus on the aspects of interest. As an example, the *digital abstraction* involves only allowing two voltage levels in a circuit, with transistors being either turned “on” (that is, fully conducting) or turned “off” (that is, not conducting). One of the assumptions we make in supporting this abstraction is that transistors switch on and off virtually instantaneously. One of the design disciplines we follow is to regulate switching to occur within well-defined intervals of time, called “clock periods.” We will see many other assumptions and disciplines as we proceed. The benefit of the digital abstraction is that it allows us to apply much simpler analysis and design procedures, and thus to build much more complex systems.

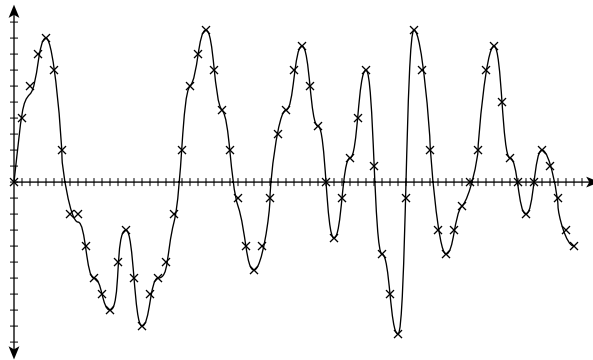


FIGURE 1.1 A pressure waveform of a sound, continuously varying over time, and the discrete representation of the waveform in a digital system.

The circuits that we will deal with in this book all perform functions that involve manipulating information of various kinds over time. The information might be an audio signal, the position of part of a machine, or the temperature of a substance. The information may change over time, and the way in which it is manipulated may vary with time.

Digital systems are electronic circuits that represent information in discrete form. An example of the kind of information that we might represent is an audio sound. In the real world, a sound is a continuously varying pressure waveform, which we might represent mathematically using a continuous function of time. However, representing that function with any significant precision as a continuously varying electrical signal in a circuit is difficult and costly, due to electrical noise and variation in circuit parameters. A digital system, on the other hand, represents the signal as a stream of discrete values sampled at discrete points in time, as shown in Figure 1.1. Each sample represents an approximation to the pressure value at a given instant. The approximations are drawn from a discrete set of values, for example, the set $\{-10.0, -9.9, -9.8, \dots, -0.1, 0.0, 0.1, \dots, 9.9, 10.0\}$. By limiting the set of values that can be represented, we can encode each value with a unique combination of digital values, each of which is either a low or high voltage. We shall see exactly how we might do that in Chapter 2. Furthermore, by sampling the signal at regular intervals, say, every $50\mu\text{s}$, the rate and times at which samples arrive and must be processed is predictable.

Discrete representations of information and discrete sequencing in time are fundamental abstractions. Much of this book is about how to choose appropriate representations, how to process information thus represented, how to sequence the processing, and how to ensure that the assumptions supporting the abstractions are maintained.

The majority of digital systems designed and manufactured today are *embedded systems*, in which much of the processing work is done by one

or more computers that form part of the system. In fact, the vast majority of computers in use today are in embedded systems, rather than in PCs and other general purpose systems. Early computers were large systems in their own right, and were rarely considered as components of larger systems. However, as technology developed, particularly to the stage of IC technology, it became practical to embed small computers as components of a circuit and to program them to implement part of the circuit's functionality. Embedded computers usually do not take the same form as general purpose computers, such as desktop or portable PCs. Instead, an embedded computer consists of a *processor core*, together with memory components for storing the program and data for the program to run on the processor core, and other components for transferring data between the processor core and the rest of the system.

The programs running on processor cores form the *embedded software* of a system. The way in which embedded software is written bears both similarities and differences with software development for general purpose computers. It is a large topic area in its own right and is beyond the scope of this book. Nonetheless, since we are dealing with embedded systems in this book, we need to address embedded software at least at a basic level. We will return to the topic as part of our discussion of interfacing with embedded processor cores in Chapters 8 and 9.

Since most digital systems in use today are embedded systems, most digital design practice involves developing the interface circuitry around processor cores and the application-specific circuitry to perform tasks not done by the cores. That is why this book deals specifically with digital design in the context of embedded systems.

1.2 BINARY REPRESENTATION AND CIRCUIT ELEMENTS

The simplest discrete representation that we see in a digital system is called a *binary* representation. It is a representation of information that can have only two values. Examples of such information include:

- ▶ whether a switch is open or closed
- ▶ whether a light is on or off
- ▶ whether a microphone is active or muted

We can think of these as logical conditions: each is either true or false. In order to represent them in a digital circuit, we can assign a high voltage level to represent the value true, and a low voltage level to represent the value false. (This is just a convention, called *positive logic*, or *active-high logic*. We could make the reverse assignment, leading to *negative logic*, or *active-low logic*, which we will discuss in Chapter 2.) We often use the values 0 and 1 instead of false and true, respectively.

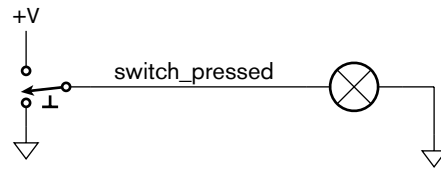


FIGURE 1.2 A circuit in which a switch controls a lamp.

The values 0 and 1 are binary (base 2) digits, or *bits*, hence the term binary representation.

The circuit shown in Figure 1.2 illustrates the idea of binary representation. The signal labeled `switch_pressed` represents the state of the switch. When the push-button switch is pressed, the signal has a high voltage, representing the truth of the condition, “the switch is pressed.” When the switch is not pressed, the signal has a low voltage, representing the falsehood of the condition. Since illumination of the lamp is controlled by the switch, we could equally well have labeled the signal `lamp_lit`, with a high voltage representing the truth of the condition, “the lamp is lit,” and a low voltage representing the falsehood of the condition.

A more complex digital circuit is shown in Figure 1.3. This circuit includes a light sensor with a digital output, `dark`, that is true (high voltage) when there is no ambient light, or false (low voltage) otherwise. The circuit also includes a switch that determines whether the digital signal `lamp_enabled` is low or high (that is, false or true, respectively). The symbol in the middle of the figure represents an *AND gate*, a digital circuit element whose output is only true (1) if both of its inputs are true (1). The output is false (0) if either input is false (0). Thus, in the circuit, the signal `lamp_lit` is true if `lamp_enabled` is true *and* `dark` is true, and is false otherwise. Given this behavior, we can apply laws of logic to analyze the circuit. For example, we can determine that if there is ambient light, the lamp will not light, since the logical AND of two conditions yields falsehood when either of the conditions is false.

The AND gate shown in Figure 1.3 is just one of several basic digital logic components. Some others are shown in Figure 1.4. The AND gate, as

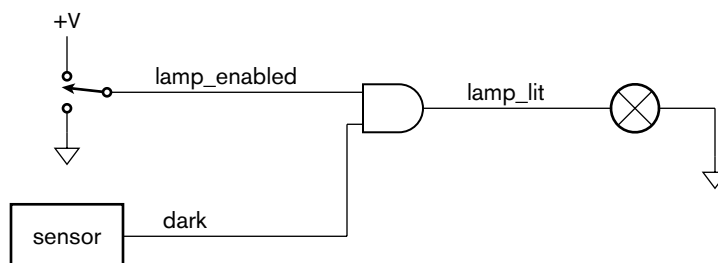


FIGURE 1.3 A digital circuit for a night-light that is only lit when the switch is on and the light sensor shows that it is dark.

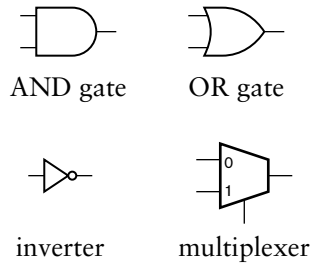


FIGURE 1.4 Basic digital logic gates.

we mentioned above, produces a 1 on its output if both inputs are 1, or a 0 on the output if either input is 0. The *OR gate* produces the “inclusive or” of its inputs. Its output is 1 if either or both of the inputs is 1, or 0 if both inputs are 0. The *inverter* produces the “negation” of its input. Its output is 1 if the input is 0, or 0 if the input is 1. Finally, the *multiplexer* selects between the two inputs labeled “0” and “1” based on the value of the “select” input at the bottom of the symbol. If the select input has the value 0, then the output has the same value as that on the “0” input, whereas if the select input has the value 1, then the output has the same value as that on the “1” input.

We can use these logic gates to build digital circuits that implement more complex logic functions.

EXAMPLE 1.1 Suppose a factory has two vats, only one of which is used at a time. The liquid in the vat in use needs to be at the right temperature, between 25°C and 30°C. Each vat has two temperature sensors indicating whether the temperature is above 25°C and above 30°C, respectively. The vats also have low-level sensors. The supervisor needs to be woken up by a buzzer when the temperature is too high or too low or the vat level is too low. He has a switch to select which vat is in use. Design a circuit of gates to activate the buzzer as required.

SOLUTION For the selected vat, the condition for activating the buzzer is “temperature not above 25°C or temperature above 30°C, or level low.” This can be implemented with a gate circuit for each vat. The switch can be used to control the select input of a multiplexer to choose between the circuit outputs for the two vats. The output of the multiplexer then activates the buzzer. The complete circuit is shown in Figure 1.5.

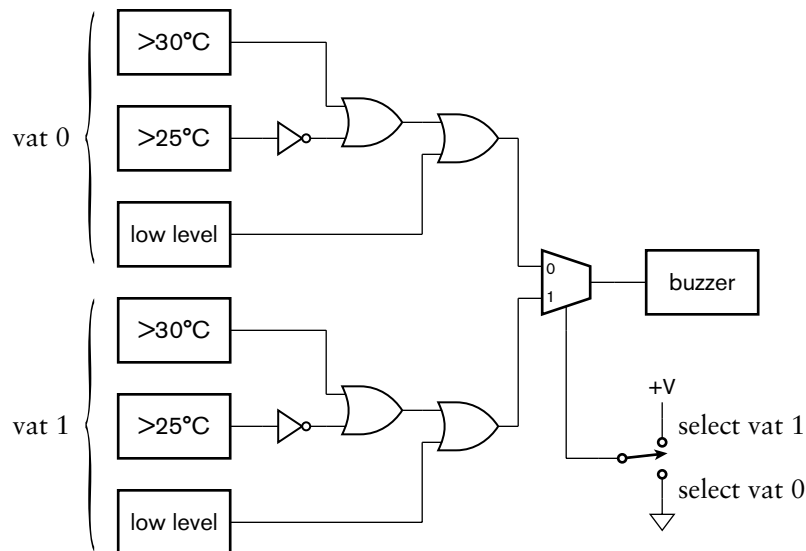


FIGURE 1.5 The vat buzzer circuit.

Circuits such as those considered above are called *combinational*. This means that the values of the circuit's outputs at any given time are determined purely by combining the values of the inputs at that time. There is no notion of storage of information, that is, dependence on values at previous times. While combinational circuits are important as parts of larger digital systems, nearly all digital systems are *sequential*. This means that they do include some form of storage, allowing the values of outputs to be determined by both the current input values and previous input values.

One of the simplest digital circuit elements for storing information is called, somewhat prosaically, a *flip-flop*. It can “remember” a single bit of information, allowing it to “flip” and “flop” between a stored 0 state and a stored 1 state. The symbol for a *D flip-flop* is shown in Figure 1.6. It is called a “D” flip-flop because it has a single input, D, representing the value of the data to be stored: “D” for “data.” It also has another input, *clk*, called the clock input, that indicates when the value of the D input should be stored. The behavior of the D flip-flop is illustrated in the *timing diagram* in Figure 1.7. A timing diagram is a graph of the values of one or more signals as they change with time. Time extends along the horizontal axis, and the signals of interest are listed on the vertical axis. Figure 1.7 shows the D input of the flip-flop changing at irregular intervals and the *clk* input changing periodically. A transition of *clk* from 0 to 1 is called a *rising edge* of the signal. (Similarly, a transition from 1 to 0 is called a *falling edge*.) The small triangular marking next to the *clk* input specifies that the D value is stored only on a rising edge of the *clk* input. At that time, the Q output changes to reflect the stored value. Any subsequent changes on the D input are ignored until the next rising edge of *clk*. A circuit element that behaves in this way is called *edge-triggered*.

While the behavior of a flip-flop does not depend on the clock input being periodic, in nearly all digital systems, there is a single clock signal that synchronizes all of the storage elements in the system. The system is composed of combinational circuits that perform logical functions on the values of signals and flip-flops that store intermediate results. As we

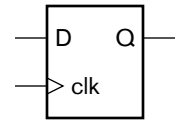


FIGURE 1.6 A D flip-flop.

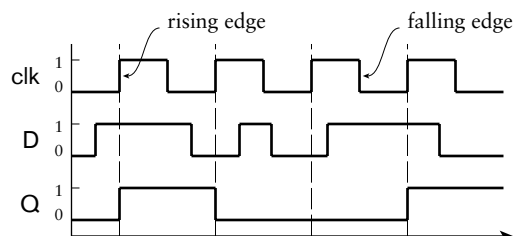


FIGURE 1.7 Timing diagram for a D flip-flop.

shall see, use of a single periodic synchronizing clock greatly simplifies design of the system. The clock operates at a fixed *frequency* and divides time into discrete intervals, called *clock periods*, or *clock cycles*. Modern digital circuits operate with clock frequencies in the range of tens to hundreds of megahertz (MHz, or millions of cycles per second), with high-performance circuits extending up to several gigahertz (GHz, or billions of cycles per second). Division of time into discrete intervals allows us to deal with time in a more abstract form. This is another example of abstraction at work.

EXAMPLE 1.2 Develop a sequential circuit that has a single data input signal, S , and produces an output Y . The output is 1 whenever S has the same value over three successive clock cycles, and 0 otherwise. Assume that the value of S for a given clock cycle is defined at the time of the rising clock edge at the end of the clock cycle.

SOLUTION In order to compare the values of S in three successive clock cycles, we need to save the values of S for the previous two cycles and compare them with the current value of S . We can use a pair of D flip-flops, connected in a *pipeline* as shown in Figure 1.8, to store the values. When a clock edge occurs, the first flip-flop, ff1, stores the value of S from the preceding clock cycle. That value is passed onto the second flip-flop, ff2, so that at the next clock edge, ff2 stores the value of S from two cycles prior.

The output Y is 1 if and only if three successive value of S are all 1 or are all 0. Gates $g1$ and $g2$ jointly determine if the three values are all 1. Inverters $g3$, $g4$ and $g5$ negate the three values, and so gates $g6$ and $g7$ determine if the three values are all 0. Gate $g8$ combines the two alternatives to yield the final output.

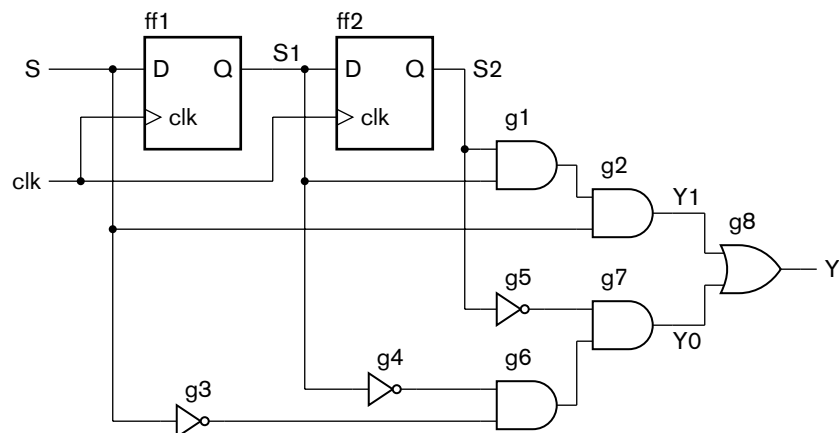


FIGURE 1.8 A sequential circuit for comparing successive bits of an input.

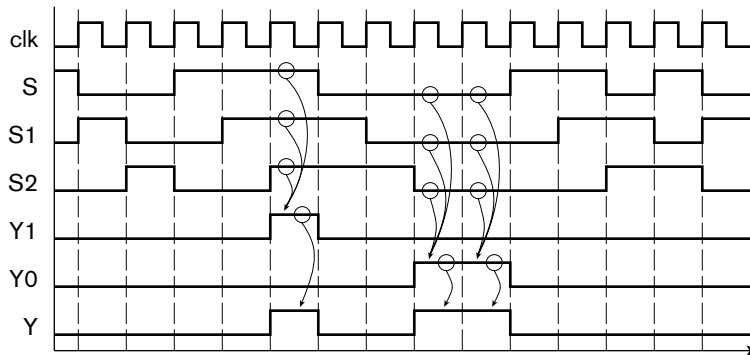


FIGURE 1.9 Timing diagram for the sequential comparison circuit.

Figure 1.9 shows a timing diagram of the circuit for a particular sequence of input values on S over several clock cycles. The outputs of the two flip-flops follow the value of S , but are delayed by one and two clock cycles, respectively. This timing diagram shows the value of S changing at the time of a clock edge. The flip-flop will actually store the value that is on S immediately before the clock edge. The circles and arrows indicate which signals are used to determine the values of other signals, leading to a 1 at the output. When all of S , $S1$ and $S2$ are 1, $Y1$ changes to 1, indicating that S has been 1 for three successive cycles. Similarly, when all of S , $S1$ and $S2$ are 0, $Y0$ changes to 1, indicating that S has been 0 for three successive cycles. When either of $Y1$ or $Y0$ is 1, the output Y changes to 1.

1. What are the two values used in binary representation?
2. If one input of an AND gate is 0 and the other is 1, what is the output value? What if both are 0, or both are 1?
3. If one input of an OR gate is 0 and the other is 1, what is the output value? What if both are 0, or both are 1?
4. What function is performed by a multiplexer?
5. What is the distinction between combinational and sequential circuits?
6. How much information is stored by a flip-flop?
7. What is meant by the terms *rising edge* and *falling edge*?

KNOWLEDGE TEST QUIZ

1.3 REAL-WORLD CIRCUITS

In order to analyze and design circuits as we have discussed, we are making a number of assumptions that underlie the digital abstraction. We have assumed that a circuit behaves in an ideal manner, allowing us to think in