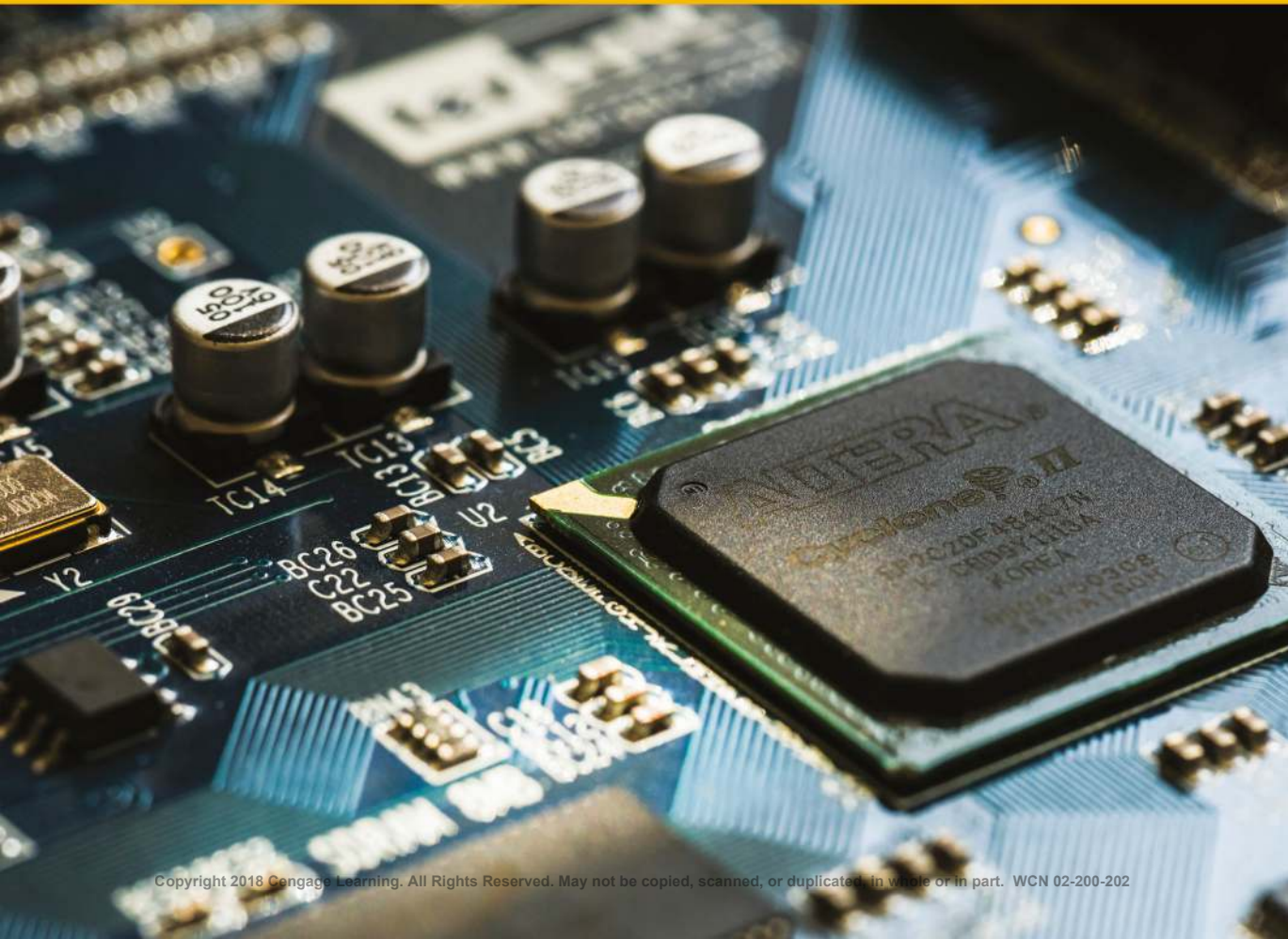
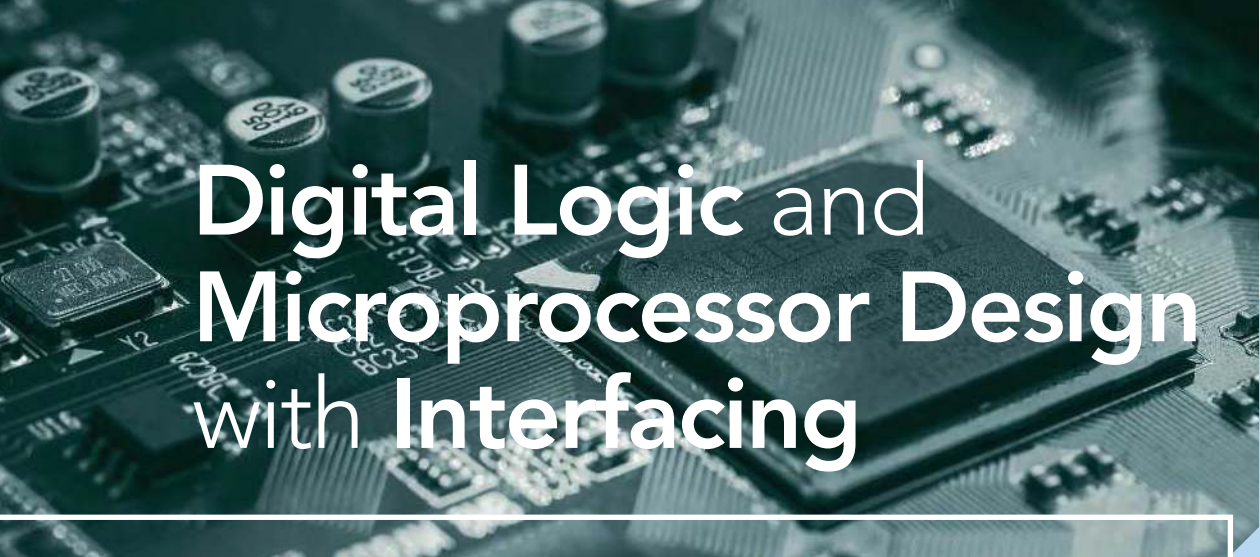


Digital Logic & Microprocessor Design with **Interfacing** 2nd Edition

Enoch O. Hwang



Digital Logic and Microprocessor Design with Interfacing



Digital Logic and Microprocessor Design with Interfacing

2nd Edition

Enoch O. Hwang

La Sierra University
Riverside, California, USA



***Digital Logic and Microprocessor Design with
Interfacing, Second Edition***
Enoch O. Hwang

Product Director, Global Engineering:
Timothy L. Anderson

Associate Media Content Developer:
Ashley Kaupert

Product Assistant: Alexander Sham

Marketing Manager: Kristin Stine

Director, Higher Education Production:
Sharon L. Smith

Senior Content Project Manager: Kim Kusnerak

Production Service: SPi Global

Senior Art Director: Michelle Kunkler

Cover/Internal Designer: Red Hangar Design, LLC

Cover/Internal Image: Jonathan Y. Hwang

Intellectual Property

Analyst: Christine Myaskovsky

Project Manager: Sarah Shainwald

Text and Image Permissions Researcher:
Kristiina Paul

Manufacturing Planner: Doug Wilke

© 2018, 2006 Cengage Learning®

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced or distributed in any form or by any means, except as permitted by U.S. copyright law, without the prior written permission of the copyright owner.

For product information and technology assistance, contact us at
Cengage Learning Customer & Sales Support, 1-800-354-9706.

For permission to use material from this text or product,
submit all requests online at **www.cengage.com/permissions.**

Further permissions questions can be emailed to
permissionrequest@cengage.com.

Library of Congress Control Number: 2016952181

ISBN: 978-1-305-85945-6

Cengage Learning

20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com.**

Cengage Learning products are represented in Canada by Nelson Education Ltd.

To learn more about Cengage Learning Solutions, visit **www.cengage.com/engineering.**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com.**

Unless otherwise noted, all items © Cengage Learning.

Printed in the United States of America
Print Number: 01 Print Year: 2016

*To my wife Windy,
the love of my life,
for her endless love and support.*

CONTENTS

Preface.....	xv
About the Author.....	xxi

CHAPTER 1 Introduction to Microprocessor Design 1

1.1 Overview of Microprocessor Design.....	3
1.2 Design Abstraction Levels.....	6
1.3 Examples of a 2-to-1 Multiplexer.....	7
1.3.1 Behavioral Level.....	7
1.3.2 Gate Level.....	9
1.3.3 Transistor Level.....	11
1.4 Introduction to Hardware Description Language.....	11
1.5 Synthesis.....	15
1.6 Going Forward.....	16
1.7 Problems.....	17

CHAPTER 2 Fundamentals of Digital Circuits 18

2.1 Binary Numbers.....	19
2.1.1 Counting in Binary.....	20
2.1.2 Converting between Binary and Decimal.....	20
2.1.3 Octal and Hexadecimal Notations.....	23
2.1.4 Binary Number Arithmetic.....	25
2.2 Negative Numbers.....	27
2.2.1 Two's Complement Representation.....	27
2.2.2 Sign Extension.....	29
2.2.3 Signed Number Arithmetic.....	30
2.3 Binary Switch.....	32
2.4 Basic Logic Operators and Logic Expressions.....	33
2.5 Logic Gates.....	35
2.6 Truth Tables.....	36
2.7 Boolean Algebra and Boolean Equations.....	38
2.7.1 Boolean Algebra.....	38
2.7.2 Duality Principle.....	41
2.7.3 Boolean Functions and Their Inverses.....	41

2.8	Minterms and Maxterms	46
2.8.1	Minterms	46
2.8.2	Maxterms	49
2.9	Canonical, Standard, and Non-Standard Forms	52
2.10	Digital Circuits	53
2.11	Designing a Car Security System	54
2.12	Verilog and VHDL Code for Digital Circuits	57
2.12.1	Verilog Code for a Boolean Function.....	57
2.12.2	VHDL Code for a Boolean Function.....	58
2.13	Problems	59

CHAPTER 3

Combinational Circuits 65

3.1	Analysis of Combinational Circuits	66
3.1.1	Using a Truth Table	67
3.1.2	Using a Boolean Function	70
3.2	Synthesis of Combinational Circuits	72
3.2.1	Using Only NAND Gates	75
3.3	Minimization of Combinational Circuits	76
3.3.1	Boolean Algebra	77
3.3.2	Karnaugh Maps.....	78
3.3.3	Don't-Cares	85
3.3.4	Tabulation Method	86
3.4	Timing Hazards and Glitches	89
3.4.1	Using Glitches	91
3.5	BCD to 7-Segment Decoder	92
3.6	Verilog and VHDL Code for Combinational Circuits	95
3.6.1	Structural Verilog Code	95
3.6.2	Structural VHDL Code	97
3.6.3	Dataflow Verilog Code.....	101
3.6.4	Dataflow VHDL Code.....	102
3.6.5	Behavioral Verilog Code	103
3.6.6	Behavioral VHDL Code	104
3.7	Problems	106

CHAPTER 4

Standard Combinational Components 112

4.1	Signal Naming Conventions	113
4.2	Multiplexer	114
4.3	Adder	117
4.3.1	Full Adder	117
4.3.2	Ripple-Carry Adder.....	118
4.3.3	Carry-Lookahead Adder	120

4.4	Subtractor	123
4.5	Adder-Subtractor Combination	125
4.6	Arithmetic Logic Unit	129
4.7	Decoder	137
4.8	Tri-State Buffer	140
4.9	Comparator	142
4.10	Shifter	146
4.11	Multiplier	149
4.12	Problems	151

CHAPTER 5

Sequential Circuits 157

5.1	Bistable Element	159
5.2	SR Latch	160
5.3	Car Security System—Version 2	163
5.4	SR Latch with Enable	164
5.5	D Latch	164
5.6	D Latch with Enable	166
5.7	Verilog and VHDL Code for Memory Elements	166
	5.7.1 VHDL Code for a D Latch with Enable	168
	5.7.2 Verilog Code for a D Latch with Enable	169
5.8	Clock	169
5.9	D Flip-Flop	171
	5.9.1 Alternative Smaller Circuit	175
5.10	D Flip-Flop with Enable	176
	5.10.1 Asynchronous Inputs	177
5.11	Description of a Flip-Flop	180
	5.11.1 Characteristic Table	180
	5.11.2 Characteristic Equation	180
	5.11.3 State Diagram	180
5.12	Register	181
5.13	Register File	182
5.14	Memories	188
	5.14.1 ROM	190
	5.14.2 RAM	192
5.15	Shift Registers	197
	5.15.1 Serial-to-Parallel Shift Register	199
	5.15.2 Serial-to-Parallel and Parallel-to-Serial Shift Register	200
	5.15.3 Linear Feedback Shift Register	202
5.16	Counters	205
	5.16.1 Binary Up Counter	205
	5.16.2 Binary Up Counter with Parallel Load	207

5.17	Timing Issues.....	210
5.18	Problems	211

CHAPTER 6

Finite-State Machines 215

6.1	Finite-State Machine Models.....	217
6.2	State Diagrams.....	221
6.3	Analysis of Finite-State Machines	224
6.3.1	Next-State Equations	225
6.3.2	Next-State Table	226
6.3.3	Output Equations	228
6.3.4	Output Table.....	228
6.3.5	State Diagram	229
6.3.6	Example.....	230
6.4	Synthesis of Finite-State Machines	234
6.4.1	State Diagram	235
6.4.2	Next-State Table	236
6.4.3	Next-State Equations	237
6.4.4	Output Table and Output Equations.....	237
6.4.5	FSM Circuit.....	238
6.5	Optimizations for FSMs	239
6.5.1	State Reduction.....	239
6.5.2	State Encoding.....	240
6.5.3	Unused States	243
6.6	FSM Construction Examples.....	243
6.6.1	Car Security System—Version 3.....	243
6.6.2	Modulo-6 Up-Counter.....	245
6.6.3	One-Shot Circuit.....	249
6.6.4	Simple Microprocessor Control Unit	251
6.6.5	Elevator Controller Using a Moore FSM	254
6.6.6	Elevator Controller Using a Mealy FSM	258
6.7	Verilog and VHDL Code for FSM Circuits	261
6.7.1	Behavioral Verilog Code for a Moore FSM.....	261
6.7.2	Behavioral Verilog Code for a Mealy FSM	265
6.7.3	Behavioral VHDL Code for a Moore FSM.....	266
6.7.4	Behavioral VHDL Code for a Mealy FSM	269
6.8	Problems	270

CHAPTER 7

Dedicated Microprocessors 283

7.1	Need for a Datapath	286
7.2	Constructing the Datapath.....	287
7.2.1	Selecting Registers	293
7.2.2	Selecting Functional Units.....	294

7.2.3	Data Transfer Methods.....	295
7.2.4	Generating Status Signals.....	297
7.3	Constructing the Control Unit.....	302
7.3.1	Deriving the Control Signals.....	303
7.3.2	Deriving the State Diagram.....	305
7.3.3	Timing Issues.....	312
7.3.4	Deriving the FSM Circuit.....	315
7.4	Constructing the Complete Microprocessor.....	320
7.5	Dedicated Microprocessor Construction Examples.....	323
7.5.1	Greatest Common Divisor.....	323
7.5.2	High-Low Number Guessing Game.....	330
7.5.3	Traffic Light Controller.....	337
7.6	Verilog and VHDL Code for Dedicated Microprocessors.....	341
7.6.1	FSM+D Model.....	342
7.6.2	FSMD Model.....	351
7.6.3	Algorithmic Model.....	354
7.7	Problems.....	356

CHAPTER 8

General-Purpose Microprocessors 363

8.1	Overview of the CPU Design.....	364
8.2	The EC-1 General-Purpose Microprocessor.....	366
8.2.1	Instruction Set.....	366
8.2.2	Datapath.....	367
8.2.3	Control Unit.....	369
8.2.4	Complete Circuit.....	373
8.2.5	Sample Program.....	373
8.2.6	Simulation.....	375
8.2.7	Hardware Implementation.....	375
8.3	The EC-2 General-Purpose Microprocessor.....	376
8.3.1	Instruction Set.....	376
8.3.2	Datapath.....	377
8.3.3	Control Unit.....	379
8.3.4	Complete Circuit.....	384
8.3.5	Sample Program.....	384
8.3.6	Hardware Implementation.....	387
8.4	Extending the EC-2 Instruction Set.....	388
8.5	Using and Interfacing the EC-2.....	391
8.6	Pipelining.....	395
8.6.1	Basic Pipelined Processor.....	395
8.6.2	Pipeline Hazards.....	397
8.7	Verilog and VHDL Code for General-Purpose Microprocessors.....	399
8.7.1	FSM+D Model.....	399
8.7.2	FSMD Model.....	405
8.8	Problems.....	411

CHAPTER 9

Interfacing Microprocessors 415

9.1	Multiplexing 7-Segment LED Display	416
9.1.1	Theory of Operation	416
9.1.2	Controller Design	417
9.2	Issues with Interfacing Switches	420
9.3	3 × 4 Keypad Controller	427
9.3.1	Theory of Operation	427
9.3.2	Controller Design	429
9.4	PS2 Keyboard and Mouse	431
9.4.1	Theory of Operation—PS2 Keyboard	431
9.4.2	Controller Design—PS2 Keyboard	432
9.4.3	Theory of Operation—PS2 Mouse	436
9.4.4	Controller Design—PS2 Mouse	438
9.5	RS-232 Controller for Bluetooth Communication	444
9.5.1	Theory of Operation—RS-232	445
9.5.2	Controller Design—RS-232	446
9.5.3	Implementation	449
9.6	Liquid-Crystal Display Controller	450
9.6.1	Theory of Operation	450
9.6.2	Controller Design	452
9.7	VGA Monitor Controller	457
9.7.1	Theory of Operation	457
9.7.2	Controller Design	460
9.7.3	Implementation	466
9.8	A/D Controller for Temperature Sensor	467
9.8.1	Theory of Operation	467
9.8.2	Controller Design	469
9.8.3	Implementation	474
9.9	I²C Bus Controller for Real-Time Clock	475
9.9.1	Theory of Operation	475
9.9.2	Controller Design	478
9.9.3	Implementation	483
9.10	Problems	484

APPENDIX A

Xilinx Development Tutorial 486

A.1	Starting ISE	486
A.1.1	Creating a New Project	486
A.1.2	Specifying the FPGA	489

A.2	Creating a New Schematic Source File	492
A.2.1	Drawing Your Schematic Circuit.....	493
A.2.2	Creating and Using a Schematic Symbol.....	495
A.2.3	Editing a Schematic Symbol	497
A.2.4	Using a Schematic Symbol in Another Project.....	500
A.3	Creating a New Verilog or VHDL Source File	500
A.4	Setting the Top-Level Module Design File	500
A.5	Mapping the I/O Signals	500
A.5.1	Using PlanAhead for Mapping the Pins.....	501
A.5.2	Faster Alternative Method for Mapping the Pins.....	503
A.6	Synthesis and Implementation	503
A.7	Programming the Circuit to the FPGA	505
A.8	Problems	509

APPENDIX B

Altera Development Tutorial 512

B.1	Starting Quartus	512
B.1.1	Creating a New Project	512
B.1.2	Specifying the FPGA	514
B.2	Using the Graphic Editor	516
B.2.1	Starting the Graphic Editor	516
B.2.2	Drawing Tools.....	516
B.2.3	Inserting Logic Symbols	517
B.2.4	Selecting, Moving, Copying, and Deleting Logic Symbols.....	518
B.2.5	Making and Naming Connections	519
B.2.6	Selecting, Moving, and Deleting Connection Lines.....	521
B.3	Managing Files in a Project	521
B.3.1	Design Files in a Project.....	522
B.3.2	Creating a New Verilog or VHDL Source File.....	522
B.3.3	Opening a Design File.....	522
B.3.4	Adding Design Files to a Project.....	523
B.3.5	Deleting Design Files from a Project.....	523
B.3.6	Setting the Top-Level Entity Design File	523
B.3.7	Saving the Project.....	523
B.4	Analysis and Synthesis	523
B.5	Creating and Using a Logic Symbol	524
B.6	Mapping the I/O Signals	525
B.6.1	Faster Alternative Method for Mapping the Pins.....	527
B.7	Fitting the Netlist and Pins to the FPGA	527
B.8	Programming the Circuit to the FPGA	528
B.9	Problems	529

APPENDIX C

Verilog Summary 533

C.1	Basic Language Elements	533
C.1.1	Keywords.....	533
C.1.2	Comments.....	534
C.1.3	Identifiers.....	534
C.1.4	Signals	534
C.1.5	Numbers and Strings	534
C.1.6	Constants.....	535
C.1.7	Data Types.....	536
C.1.8	Data Operators	536
C.1.9	Module	537
C.1.10	Module Parameter	539
C.2	Behavioral Model	540
C.2.1	Assignment	540
C.2.2	initial.....	541
C.2.3	always	542
C.2.4	Event Control	542
C.2.5	begin-end	544
C.2.6	if-then-else	545
C.2.7	case, casex, casez.....	545
C.2.8	for	546
C.2.9	while	547
C.2.10	function.....	547
C.2.11	Behavioral Model Example.....	548
C.3	Dataflow Model.....	548
C.3.1	Continuous Assignment.....	549
C.3.2	Conditional Assignment	549
C.3.3	Dataflow Model Example.....	550
C.4	Structural Model	550
C.4.1	Built-in Gates.....	550
C.4.2	User-Defined Module	551
C.4.3	Structural Model Example.....	552

APPENDIX D

VHDL Summary 553

D.1	Basic Language Elements	553
D.1.1	Keywords.....	553
D.1.2	Comments.....	554
D.1.3	Identifiers.....	554
D.1.4	Data Objects	554
D.1.5	Data Types.....	554
D.1.6	Data Operators	557
D.1.7	ENTITY	558

D.1.8 ARCHITECTURE.....	559
D.1.9 GENERIC.....	560
D.1.10 PACKAGE.....	562
D.2 Behavioral Model—Sequential Statements.....	564
D.2.1 PROCESS.....	564
D.2.2 Sequential Signal Assignment.....	564
D.2.3 Variable Assignment.....	565
D.2.4 WAIT.....	565
D.2.5 IF-THEN-ELSE.....	565
D.2.6 CASE.....	566
D.2.7 NULL.....	566
D.2.8 FOR.....	567
D.2.9 WHILE.....	567
D.2.10 LOOP.....	567
D.2.11 EXIT.....	567
D.2.12 NEXT.....	567
D.2.13 FUNCTION.....	568
D.2.14 PROCEDURE.....	569
D.2.15 Behavioral Model Example.....	570
D.3 Dataflow Model—Concurrent Statements.....	570
D.3.1 Concurrent Signal Assignment.....	570
D.3.2 Conditional Signal Assignment.....	571
D.3.3 Selected Signal Assignment.....	571
D.3.4 Dataflow Model Example.....	572
D.4 Structural Model—Concurrent Statements.....	573
D.4.1 COMPONENT Declaration.....	573
D.4.2 PORT MAP.....	574
D.4.3 OPEN.....	574
D.4.4 GENERATE.....	575
D.4.5 Structural Model Example.....	575
D.5 Conversion Routines.....	576
D.5.1 CONV_INTEGER().....	576
D.5.2 CONV_STD_LOGIC_VECTOR(,).....	577
Index.....	578



PREFACE

This book is about the digital logic design of microprocessors, and is intended to provide both an understanding of the basic principles of digital logic design, and how these fundamental principles are applied in the building of complex microprocessor circuits using current technologies. Although the basic principles of digital logic design have not changed, the design process and the implementation of the circuits have. With the advances in fully integrated modern hardware computer-aided design (CAD) tools for logic synthesis, simulation, and the implementation of digital circuits in field-programmable gate arrays (FPGAs), it is now possible to design and implement complex digital circuits very easily and quickly.

Many excellent books on digital logic design have followed the traditional approach of introducing the basic principles and theories of digital logic design and the building of separate standard combinational and sequential components. However, students are left to wonder about the purpose of these individual components and how they are used in the building of more complex digital circuits, such as microcontrollers and microprocessors that are used in controlling real-world electronic devices. The primary goal of this book is to fill in this gap by going beyond the logic principles and the building of basic standard components. The book discusses in detail how the basic components are combined together to form datapaths, how control units are designed, and how these two main components (datapath and control unit) are connected together to produce actual dedicated custom microprocessors and general-purpose microprocessors. The book ends with an entire chapter containing many examples on how microprocessors are interfaced with real-world devices.

Many texts on digital logic design and implementation techniques mainly focus on the logic gate level. At this low level, it is difficult to discuss larger and more complex circuits that are beyond the standard combinational and sequential circuits. However, with the introduction of the register-transfer technique for designing datapaths and the concept of a finite-state machine for control units, we can easily design a dedicated microprocessor for any arbitrary algorithm and then implement it on a FPGA chip to execute that algorithm. The book uses an easy-to-understand ground-up approach with complete circuit diagrams, and both Verilog and VHDL codes, starting with the building of basic digital components. These components are then used in the building of more complex components, and finally the building of the complete dedicated microprocessor circuit. The construction of a general-purpose microprocessor then comes naturally as a generalization of a dedicated microprocessor. At the end, students will have a complete understanding of how to design, construct, and implement fully working custom microprocessors.

Design of Circuits using Verilog and VHDL

Although this book provides coverage on both Verilog and VHDL for all of the circuits, this information can be omitted entirely while gaining an understanding of digital circuits and their design. For an introductory course in digital logic design, learning the basic principles is more important than learning how to use a hardware description language (HDL). In fact, instructors may find that students can get lost in learning the principles while trying to learn the language at the same time. With this in mind, the Verilog and VHDL code in the text is totally independent of the presentation of each topic and may be skipped without any loss of continuity.

On the other hand, by studying the HDL codes, the student can not only learn the use of a hardware description language but also learn how digital circuits can be designed automatically using a synthesizer. This book provides an introduction to both Verilog and VHDL and uses the “learn-by-examples” approach. In writing either Verilog or VHDL code at the dataflow and behavioral levels, the student will see the power and usefulness of a state-of-the-art hardware CAD synthesis tool.

New to This Edition

In this newly revised second edition, a new chapter on interfacing microprocessors with external devices has been added. Just knowing how to design and implement a microprocessor is not sufficient. The main purpose and usage of a microprocessor is to control external devices. This entire chapter contains many real-world examples on interfacing microprocessors with external devices. Students can use these examples to help them in doing their final projects.

Throughout the book, many new examples have been added and old examples updated. This new edition also covers the usage of both Verilog and VHDL, the two industry standard hardware description languages for describing digital circuits. All circuit examples, in addition to having schematic diagrams, also include codes written in both VHDL and Verilog.

In addition to the Altera FPGA development software, a new section in the Appendix is added for using the Xilinx FPGA development software. Using either the Altera or the Xilinx FPGA development software and their respective FPGA hardware development boards, students can actually implement these microprocessor circuits and see them execute, both in software simulation and in hardware. The book contains many interesting examples with complete schematic diagrams and Verilog and VHDL codes for implementing them in hardware. With the hands-on exercises, students will learn not only the principles of digital logic design but, also in practice, how circuits are implemented using current technologies.

To actually see your own microprocessor come to life in real hardware and being able to control real-world external devices is an exciting experience. Hopefully, this will help students to not only remember what they have learned but will also get them interested in the world of microprocessor controllers and digital circuit design.

Using This Book

This book can be used in either an introductory or a more advanced course in digital logic design. For an introductory course with no previous background in digital logic, Chapters 1 and 2 are intended to provide the fundamental basic concepts in digital logic design, while Chapters 3 and 4 cover the design of combinational circuits and standard combinational components. Chapter 5 on the design of sequential circuits can be introduced and lightly covered.

An advanced digital logic design course will start with sequential circuits in Chapter 5, and the design of finite-state machines in Chapter 6. Chapters 7 and 8 cover the design of datapaths and control units, and the building of dedicated and general-purpose microprocessors. Finally, Chapter 9 concludes with the interfacing of microprocessors with the external world.

It is strongly recommended that a lab component be fully integrated with the lecture. With an integrated lab, students can have a hands-on learning experience alongside the theoretical concepts that they have learned in class. In fact, many teachers find that too often not enough hours are given to the lab. As we probably know, it is often easier to understand the theory, but to actually implement a circuit and to get it to work requires much more detail and time. Ready-to-use labs that complement the lecture are available for download from the teachers' resource website at <https://login.cengage.com>.

Chapter 1—Introduction to Microprocessor Design gives an overview of the various components of a microprocessor circuit and the different abstraction levels in which digital circuits can be designed.

Chapter 2—Fundamentals of Digital Circuits provides the basic principles and theories for designing digital logic circuits by introducing binary numbers, the use of truth tables, Boolean algebra, and how the theories get translated into logic gates and circuit diagrams. Also a brief introduction to Verilog and VHDL is given.

Chapter 3—Combinational Circuits shows how combinational circuits are analyzed, synthesized, and optimized.

Chapter 4—Standard Combinational Components discusses the standard combinational components that are used as building blocks for larger digital circuits. These components include the adder, subtractor, arithmetic logic unit, decoder, multiplexer, tri-state buffer, comparator, shifter, and multiplier. In a hierarchical design, these components will be used in the building of the datapath used in the microprocessor.

Chapter 5—Sequential Circuits introduces latches and flip-flops as basic storage elements and then continues with larger storage components such as registers, register files, and memories. Special sequential components such as shift registers and counters are also covered.

Chapter 6—Finite-State Machines shows how finite-state machines are analyzed, synthesized, and optimized.

Chapter 7—Dedicated Microprocessors first introduces the need for a datapath, and then explains how a control unit, in the form of a finite-state machine, is used to

control the datapath. The chapter expands further showing how dedicated microprocessors are constructed by connecting the datapath and the control unit together as one coherent circuit.

Chapter 8—General-Purpose Microprocessors continues on from Chapter 7 to suggest that a general-purpose microprocessor is really a dedicated microprocessor that is dedicated to only read, decode, and execute instructions. The chapter discusses the complete design and construction of two simple general-purpose microprocessors with their own custom instruction set, and how programs written in machine language are executed on them. The highlight of this chapter and this book is that these two fully-working general-purpose microprocessors can be implemented in hardware and have programs executed by them.

Chapter 9—Interfacing Microprocessors provides several complete examples on how to interface microprocessors with real-world external devices. Examples include interfacing with a real-time clock IC using the I²C protocol, Bluetooth communication using RS-232, and drawing graphics on a VGA monitor.

The **Appendixes** provide tutorials on using both the Altera and Xilinx software development tools, and summaries on the Verilog and VHDL hardware description languages.

Supplements

Resources for the book can be found at <https://login.cengage.com/>. The instructor site is password protected and requires a verified instructor login to access the site.

Student Resources

- Chapter on Implementation Technologies
- Labs for each chapter
- All of the example codes from the book in VHDL and Verilog
- Altera FPGA development software download
- Xilinx FPGA development software download

Instructor Resources

- Chapter on Implementation Technologies
- Labs for each chapter
- PowerPoint lecture slides
- Solutions to problems at the end of each chapter
- All of the example codes from the book in VHDL and Verilog
- Altera FPGA development software download
- Xilinx FPGA development software download

Acknowledgments

I want to thank Professor Zhiguo Shi, Ph.D., and many of his graduate students from Zhejiang University, Hangzhou, China, for translating this book into Chinese. In the process, we have become lasting friends.

I also want to thank the following reviewers for their constructive feedback:

Christopher Doss, North Carolina A&T State University

Eric Durant, Milwaukee School of Engineering

Rajiv J. Kapadia, Minnesota State University, Mankato

Emma Regentova, University of Nevada, Las Vegas

Darrin Rothe, Milwaukee School of Engineering

I wish to acknowledge and thank the Global Engineering team at Cengage Learning for their dedication to this new book:

Timothy Anderson, Product Director; Ashley Kaupert, Associate Media Content Developer; Kim Kusnerak, Senior Content Project Manager; Kristin Stine, Marketing Manager; Elizabeth Brown, Learning Solutions Specialist; and Alexander Sham, Product Assistant. They have skillfully guided every aspect of this text's development and production to successful completion.

I also want to thank the College of Information Science and Electronic Engineering at Zhejiang University for inviting me as a visiting professor to teach their Digital Systems Design course (in English) using the contents of this book. During this time, I was able to gather many valuable ideas and feedbacks from the bright and enthusiastic students on how to make the book better. As a result, numerous changes have been made. This book truly is field-tested.

I also want to thank my school, La Sierra University in sunny California, for giving me the time off to be at Zhejiang University and to work on this book. It would have been extremely difficult without this extra time.

Finally, I want to thank my wife, Windy, for her support and giving me the time to focus and to finish this book.

Enoch O. Hwang, Ph.D.
Riverside, California

ABOUT THE AUTHOR

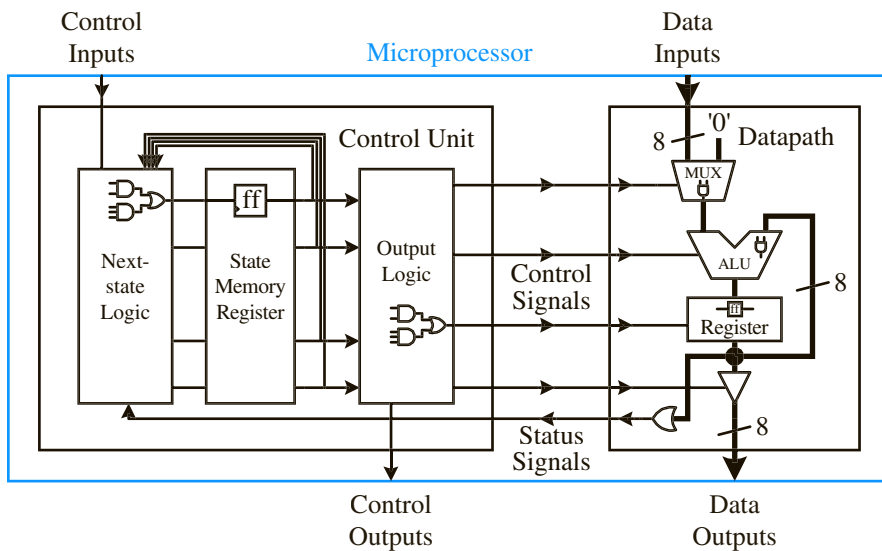


Enoch Hwang has a Ph.D. in Computer Science from the University of California, Riverside. He is currently a professor of computer science at La Sierra University in Southern California teaching digital logic and microprocessor design. In 2015, he was invited as a visiting professor to Zhejiang University in Hangzhou, China, where he taught their Digital Systems Design course. Many new ideas from that class have been incorporated into this edition of the book.

Even from his childhood days, he has been fascinated with electronic circuits. In one of his first experiments, he attempted to connect a microphone to the speaker inside a portable radio through the ear-phone plug. Instead of hearing sound from the microphone through the speaker, smoke was seen coming out of the radio. Thus ended that experiment and his family's only radio. He now continues his interest in digital circuits with research in embedded microprocessor systems, controller automation, power optimization, and robotics.

CHAPTER 1

Introduction to Microprocessor Design



Electronic devices are an integral part of our lives. Every day and everywhere we see and use electronic devices, from cellular telephones to electronic billboards, cars, toys, TVs, elevators, musical greeting cards, personal computers, traffic lights, and many more. Inside each and every one of them, there is a microprocessor that controls their operations. Microprocessors are at the heart of all of these “smart” devices. Their smartness is a direct result of the work of the microprocessor, without which none of these electronic devices would be able to operate as they do.

There are generally two types of microprocessors: **general-purpose microprocessors** and **dedicated microprocessors**. General-purpose microprocessors, such as the Intel Core™ i7 CPU shown in Figure 1.1(a) can perform different tasks under the control of different software programs. General-purpose microprocessors typically are much more powerful in terms of processing power and speed. However, they usually require external components for their memory and supporting input/output (I/O) peripherals. They are used in all personal computers.

Dedicated microprocessors, also known as **microcontrollers** or **application-specific integrated circuits (ASICs)**, on the other hand, are designed to perform just one specific task. For example, inside your cell phone is a dedicated microcontroller that does nothing else but control its entire operation. Microcontrollers therefore are usually not as powerful (because they do not need to perform so many tasks) as a microprocessor and are much smaller in size. However, they usually will have the memory and supporting I/O peripherals included inside the chip, hence the entire system can be on a single chip. For example, the Atmel ATtiny13A microcontroller shown in Figure 1.1(b) has built-in flash memory, electrically erasable programmable read-only memory (EEPROM), static random-access memory (SRAM), general-purpose I/Os, timers, serial interface, and analog-to-digital converters (ADC). Dedicated microcontrollers are used in almost all smart electronic devices. Although the small dedicated microcontrollers are not as powerful and are slower in speed as compared to general-purpose microprocessors, they are being sold much more and are used in a lot more places than general-purpose microprocessors.



© Oleksiy Maksymenko Photography / Alamy Stock Photo

(a)



Dmitry S. Gordienko / Shutterstock.com

(b)

FIGURE 1.1 Microprocessors: (a) General-purpose Intel Core™ i7 CPU; (b) Dedicated Atmel ATtiny13A microcontroller.

In this book, I will show you in detail how to design, implement, and interface a microprocessor. At the end, you will be able to design your own custom microprocessor and use it to control your own electronic device. I will use a hands-on approach to guide you step-by-step through the entire design process with complete circuits that you actually can implement in hardware. The exciting part is that at the end, you can actually, very easily and inexpensively, implement your own custom microprocessor in a real integrated circuit (IC) and see that it really can execute software programs, make lights flash, or do whatever you have designed it to do.

We will start with the fundamentals of digital logic circuit design in Chapter 2, which will provide you with a good foundation and basic building blocks for creating larger and more complex digital circuits. Chapters 3 and 4 will discuss the design of simple digital circuits and common circuits that are used as building blocks for larger circuits. Chapter 5 talks about the design of memory circuits. Typically, an introduction to digital logic design course will cover the materials from Chapters 1 to 5 only. Moving on to more advanced digital logic design, Chapter 6 talks about control unit circuits. Chapter 7 talks about the datapath and how to connect it with the control unit to produce a dedicated microcontroller. Chapter 8 extends the dedicated microcontroller from Chapter 7 to produce a general-purpose microprocessor. Finally, Chapter 9 concludes with examples of how to interface these microprocessors and microcontrollers in the real world.

1.1 Overview of Microprocessor Design

The microprocessor or microcontroller is an electronic digital logic circuit that is implemented inside an IC chip. Any digital electronic circuit at the lowest physical level understands only whether there is electricity or no electricity, which is typically represented by the use of a 1 or a 0. The question is how do we design a microprocessor so that it can understand the 1s and 0s, and then do something meaningful with that understanding? To design a microprocessor is to design its logic circuit to do whatever it is intended to do. To implement the microprocessor is to put the logic circuit of the microprocessor onto an IC chip.

Previously, making an IC chip with any circuit was a long and expensive process. With the advance of large-capacity field-programmable gate array (FPGA) chips, digital circuits of almost any size can be implemented in a chip easily and quickly. Moreover, because FPGA chips are erasable, you can use the same FPGA chip over and over again to implement different circuits. If you put an adder circuit in the FPGA chip, that chip will be an adder, and if you put a traffic light controller circuit in the FPGA chip, that chip will be a traffic light controller. So implementing any digital circuit in a FPGA chip is quite simple. The challenge now is how to design the circuit; how do we design the adder circuit or the traffic light controller circuit?

A block diagram of a microprocessor circuit is shown in Figure 1.2. As you can see, it is divided into two main parts: the **control unit** and the **datapath**. The datapath is responsible for the execution of all of the microprocessor's data operations, such as the addition of two numbers inside the arithmetic logic unit (ALU). The datapath also includes registers for the temporary storage of data and comparators for testing data values. These and many other functional units are connected together

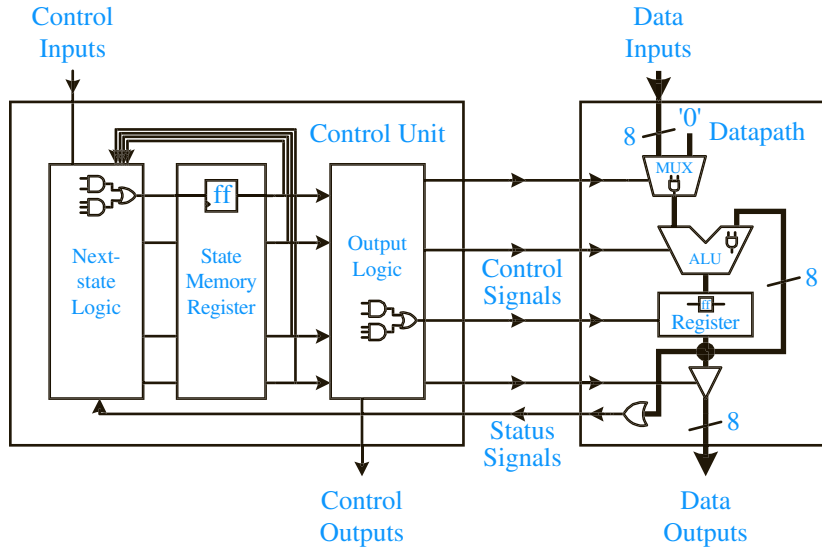


FIGURE 1.2 Internal parts of a microprocessor.

with multiplexers and data signal lines. The data signal lines are for transferring data between two functional units. Sometimes, several data signal lines are grouped together to form a **bus**. The width of the bus (i.e., the number of data signal lines in the group) is annotated next to the bus line. In the figure, the bus lines are thicker and are 8-bit wide. Multiplexers, also known as MUXs, are for selecting data from two or more sources to go to one destination. In the figure, a 2-to-1 multiplexer is used to select between the input data and the constant “0” to go to the left operand of the ALU. The output of the ALU is connected to the input of the register. The output of the register is connected to three different destinations: (1) the right operand of the ALU; (2) an OR gate used as a comparator for the test “not equal to 0”; and (3) a tri-state buffer, which is used to control the output of the data from the register.

Even though the datapath is capable of performing all of the microprocessor’s data operations, it cannot, however, do it on its own. In order for the datapath to execute the operations automatically and correctly, a control unit is required. The control unit, also known as the **controller**, controls all of the operations of the datapath and therefore, the operations of the entire microprocessor. The control unit is also called a **finite-state machine** (FSM) because it is a machine that executes by going from one state to another, and there are only a finite number of states for the machine to go to. The control unit is made up of three parts: (1) the **next-state logic**; (2) the **state memory**; and (3) the **output logic**. The purpose of the state memory is to remember the current state that the FSM is in. The next-state logic is the circuit that determines what the machine’s next state should be. The output logic is the circuit that generates the actual control signals for controlling the datapath and/or external devices.

Every digital logic circuit, regardless of whether it is part of the control unit or the datapath, is categorized as either a **combinational circuit** or a **sequential circuit**.

A combinational circuit is one where the output of the circuit is dependent only on the current inputs to the circuit, and therefore has no memory about what has happened before. For example, an adder is a combinational circuit because it will produce a sum when given any two input numbers.

A sequential circuit, on the other hand, is dependent not only on the current inputs, but also on all of the previous inputs. In other words, a sequential circuit has to remember its past history. For example, a register is a sequential circuit because it can remember a value indefinitely. Because sequential circuits are dependent on the history, they must contain memory elements to remember that history. Combinational circuits, on the other hand, do not need to remember the history, and so they do not have memory elements.

An analogy of the difference between a combinational circuit and a sequential circuit is the combination lock that we are familiar with. There are actually two different types of combination locks as shown in Figure 1.3. For the lock in Figure 1.3(a), you just turn the three number dials in any order you like to the correct number and the lock will open. For the lock in Figure 1.3(b), you also have three numbers that you need to turn to, but you need to turn to these three numbers in the correct sequence. If you turn to these three numbers in the wrong sequence the lock will not open even if you have the numbers correct. The lock in Figure 1.3(a) is like a combinational circuit where the order in which the inputs are entered into the circuit does not matter, whereas, a sequential circuit is like the lock in Figure 1.3(b) where the sequence of the inputs does matter.

Examples of combinational circuits inside the microprocessor include the ALU, multiplexers, tri-state buffers, and comparators in the datapath, and the next-state logic and output logic circuits in the control unit. Examples of sequential circuits include the register for the state memory in the control unit and the registers in the datapath.

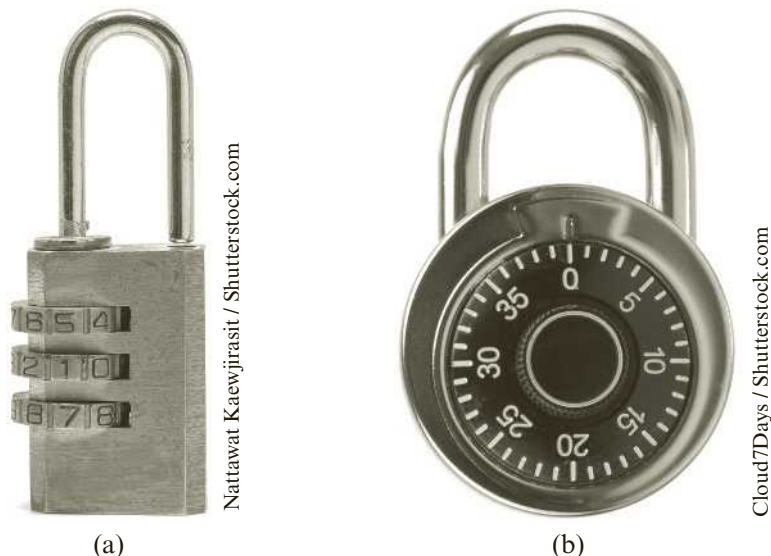


FIGURE 1.3 Two types of combination locks: (a) the order in which you enter the numbers does not matter; (b) the order in which you enter the numbers does matter.

All digital logic circuits, whether they are combinational or sequential, are made up of the three basic logic gates: **AND**, **OR**, and **NOT** gates. From these three basic gates, the most powerful computer can be made. Furthermore, these basic gates are built using transistors—the fundamental building blocks for all digital logic circuits. Transistors are simply electronic binary switches that can be turned on and off. The 1s and 0s that we, as computer scientists, often talk about are used to represent the on and off states of a transistor.

To summarize, transistors, as the lowest-level building blocks, are used to build the basic logic gates. Logic gates are connected together to form either combinational circuits or sequential circuits. The difference between these two types of circuits is only in the way the logic gates are connected together. Certain combinational circuits and sequential circuits are used as standard building blocks for larger circuits and so are kept in standard libraries. These standard combinational and sequential components are connected together to form either the datapath or the control unit. Finally, combining the datapath and the control unit together will produce the circuit for either a dedicated or a general-purpose microprocessor.

1.2 Design Abstraction Levels

Digital circuits can be designed at any one of several abstraction levels. When designing a circuit at the **transistor level**, which is the lowest level, you are dealing with discrete transistors and connecting them together to form the circuit. The next level up in the abstraction is the **gate level**. At this level, you are working with logic gates to build the circuit. In using logic gates, a designer usually creates standard combinational and sequential components for building larger circuits. In this way, a very large circuit, such as a microprocessor, can be built in a hierarchical fashion. Design methodologies have shown that solving a problem hierarchically is always easier than trying to solve the entire problem as a whole from the ground up. These combinational and sequential components are used at the **register-transfer level** to build the datapath and the control unit in the microprocessor. At the register-transfer level, we are concerned with how the data is transferred between the various registers and functional units to realize or solve the problem at hand. Finally, at the highest level, called the **behavioral level**, we can describe the behavior or operation of the circuit using a high-level hardware description language, and we can use a synthesizer, which is equivalent to a compiler, to automatically generate the logic circuit for it. Designing at this level does not require knowledge of the underlying logic gates and circuits because the synthesizer will automatically create the logic circuit for you. This is very similar to writing a computer program using a high-level programming language, and then using the compiler to automatically translate the program into machine language that the computer can execute.

An important point to realize is that there are many different ways to create the same functional circuit. Although they are all functionally equivalent, they are different in other respects, such as the actual circuit (how the transistors or gates are connected together), size (how big the circuit is or how many transistors or gates it uses), speed (how long it takes for the output result to be valid), cost (how much it costs to manufacture), and power usage (how much power it uses). Hence, when designing a circuit, in addition to being functionally correct, we also should consider

the economic versus performance tradeoffs. In this book, we will focus mainly on how to design a functionally correct circuit with some discussion about how to optimize the circuit size.

1.3 Examples of a 2-to-1 Multiplexer

As an introduction example, let us look at the design of the 2-to-1 multiplexer from different abstraction levels. At this point, don't worry too much if you don't understand the details of how all of these circuits are built. This example is intended just to give you an idea of what the circuit looks like at the different abstraction levels. We will get to the details in the rest of the book.

The multiplexer is a component that is used a lot in the datapath. An analogy for the operation of the 2-to-1 multiplexer is similar in principle to a railroad switch in which two railroad tracks are to be merged onto one track. The switch controls which one of the two trains on the two separate tracks will move onto the one track. Similarly, the 2-to-1 multiplexer has two data inputs, d_1 and d_0 , and a select input, s . The select input determines which data from the two data inputs will pass to the output, y .

Figure 1.4 shows the graphical symbol, also referred to as the **logic symbol**, for the 2-to-1 multiplexer. From looking at the logic symbol, you can tell how many signal lines the 2-to-1 multiplexer has, and the name or function designated for each line. For the 2-to-1 multiplexer, there are two data input signals, d_1 and d_0 , a select input signal, s , and an output signal, y .

1.3.1 Behavioral Level

We can describe the operation of the 2-to-1 multiplexer simply (using the same names as in the logic symbol) by saying that

if $s = 0$ then d_0 passes to y ,

otherwise

d_1 passes to y

Or more precisely, the value that is at d_0 passes to y if $s = 0$, and the value that is at d_1 passes to y if $s = 1$.

We use a hardware description language (HDL), which is quite similar to many high-level computer programming languages, to describe the circuit at the **behavioral** level. When describing a circuit at this level, you would write basically the same thing as in the description, except that you have to use the correct syntax required by the hardware description language. Figure 1.5 shows the description of the 2-to-1 multiplexer using the hardware description language called **Verilog**, and Figure 1.6 shows the description of the same 2-to-1 multiplexer using another hardware description language called **VHDL**, which stands for VHSIC Hardware Description Language (VHSIC, in turn, stands for Very High Speed Integrated Circuit). Verilog and VHDL are two standard hardware description languages used for digital logic design.

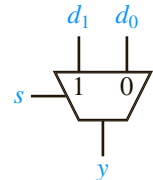


FIGURE 1.4 Logic symbol for the 2-to-1 multiplexer.


```
module multiplexer (  
    input s,  
    input d0,  
    input d1,  
    output reg y  
);  
  
    always @ (s or d0 or d1) begin  
        if (s == 0) begin  
            y = d0;  
        end else begin  
            y = d1;  
        end  
    end  
endmodule
```

FIGURE 1.5 Behavioral Verilog code for a 2-to-1 multiplexer.

In the Verilog code shown in Figure 1.5, the declaration of the component begins with the keyword `module` followed by the name of the component, which in the example, is the user identifier `multiplexer`. All of the words used in Verilog are case sensitive. The input and output interface signals are listed next using the keywords `input` and `output`. For ease of reference, the user-defined names used for these signals match those shown earlier in the logic symbol. The `always` block is followed by its sensitivity list of signals inside the parentheses. The `always` block is executed each and every time when any one of the signals in the sensitivity list changes value. The statements inside the `always` block (bracketed by the `begin` and `end` keywords) are executed sequentially. In the example, there is only one `if-then-else` statement inside the block. Like any `if` statement in other programming languages, the assignment statement `y = d0` is executed when the condition “s equals to 0” is true, otherwise the assignment statement `y = d1` is executed. For the two assignment statements, the value for the expression on the right side of the equal sign is assigned to the signal on the left side of the equal sign. Notice that the output signal `y` on the left side of the equal sign is declared as a `reg` because assignment statements inside the `always` block cannot drive a `wire` data type, but can only drive a register or an integer data type. Finally, the module is terminated with the keyword `endmodule`. A summary of the Verilog language can be found in Appendix C.

In the VHDL code shown in Figure 1.6, the `LIBRARY` and `USE` statements are similar to the “`#include`” and “`using namespace`” preprocessor commands in `C++`. None of the words used in VHDL is case sensitive, however, in the examples, the keywords are shown in upper case. The `IEEE` library contains the definition for the `STD_LOGIC` type used in the declaration of signals. The `ENTITY` section declares the interface for the circuit by specifying the input and output signals of the circuit. In this example, there are three input signals of type `STD_LOGIC` and one output signal also of type `STD_LOGIC`. Again, the names used for these signals match those shown earlier in the logic symbol. The `ARCHITECTURE` section defines the actual operation of the circuit.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY multiplexer IS PORT (
    s, d0, d1: IN STD_LOGIC;
    y: OUT STD_LOGIC);
END multiplexer;

ARCHITECTURE Behavioral OF multiplexer IS
BEGIN
    PROCESS(s, d0, d1)
    BEGIN
        IF (s = '0') THEN
            y <= d0;
        ELSE
            y <= d1;
        END IF;
    END PROCESS;
END Behavioral;

```

FIGURE 1.6 Behavioral VHDL code for a 2-to-1 multiplexer.

The ARCHITECTURE keyword is followed by a user identifier name and the entity that it is for. The PROCESS block with its sensitivity list is like the always block in Verilog. The operation of the multiplexer is defined in the conditional IF-THEN-ELSE statement. The two signal assignment statements, which use the symbol <= to denote the signal assignment, in conjunction with the IF-THEN-ELSE statement, says that the signal y gets the value of d_0 if s is equal to 0; otherwise, y gets the value of d_1 . The PROCESS block is terminated by the END PROCESS statement, and the ARCHITECTURE block is terminated by the END keyword followed by the name of this architecture. A summary of the VHDL language can be found in Appendix D.

Having written the behavioral code, either in Verilog or VHDL, we will use a synthesizer to automatically construct the netlist (which is the circuit connections) that will operate according to the description of the code. As you can see, when designing circuits at the behavioral level, we do not need to know what logic gates are needed or how they are connected together. We only need to know their interface and functional operation, and then describe it using an HDL.

1.3.2 Gate Level

At the gate level, you can draw a **schematic diagram** or **circuit diagram**, which shows how the logic gates are connected together. Two different schematic diagrams of a 2-to-1 multiplexer circuit are shown in Figures 1.7(a) and (b). In Figure 1.7(a), the circuit uses three NOT gates (\neg), four 3-input AND gates (\exists), and one 4-input OR gate (\equiv). In Figure 1.7(b), only one NOT gate, two 2-input AND gates, and one 2-input OR gate are needed. Although one circuit is larger (in terms of the number of gates needed) than the other, both of these circuits realize the same 2-to-1 multiplexer function. Therefore, when we want to actually implement a 2-to-1 multiplexer circuit, we will want to use the second, smaller circuit rather than the first.