

# *Advanced Digital Design with the Verilog HDL*

**Michael D. Ciletti**

***Department of Electrical and Computer Engineering  
University of Colorado at Colorado Springs***

Prentice  
Hall

**Pearson Education, Inc.  
Upper Saddle River, New Jersey 07458**

---

# Summary of Key Verilog Features (IEEE 1364)

## Module

Encapsulates functionality; may be nested to any depth.

**module** module\_name (list of ports);

*Declarations*

Port modes: **input**, **output**, **inout** identifier;

Nets (e.g., **wire** A[3:0];)

Register variable (e.g., **reg** B[31: 0];)

Constants: (e.g. **parameter** size = 8;)

Named events

Continuous assignments

(e.g. **assign** sum = A + B;)

Behaviors **always** (cyclic), **initial** (single-pass)

**specify ... endspecify**

**function ... endfunction**

**task ... endtask**

*Instantiations*

primitives

modules

**endmodule**

## Multi Input Primitives

(Each input is a scalar)

**and** (out, in<sub>1</sub>, in<sub>2</sub>, ... in<sub>N</sub>);

**nand** (out, in<sub>1</sub>, in<sub>2</sub>, ... in<sub>N</sub>);

**or** (out, in<sub>1</sub>, in<sub>2</sub>, ... in<sub>N</sub>);

**nor** (out, in<sub>1</sub>, in<sub>2</sub>, ... in<sub>N</sub>);

**xor** (out, in<sub>1</sub>, in<sub>2</sub>, ... in<sub>N</sub>);

**xnor** (out, in<sub>1</sub>, in<sub>2</sub>, ... in<sub>N</sub>);

## Multi-Output Primitives

**buf** (out<sub>1</sub>, out<sub>2</sub>, ..., out<sub>N</sub>, in); // buffer

**not** (out<sub>1</sub>, out<sub>2</sub>, ..., out<sub>N</sub>, in); // inverter

## Three-State Multi-Output Primitives

**bufif0** (out, in, control); **bufif1** (out, in, control);

**notif0** (out, in, control); **notif1** (out, in, control);

## Pullups and Pulldowns

**pullup** (out\_y); **pulldown** (out\_y);

## Propagation Delays

Single delay: **and** #3 G1 (y, a, b, c);

Rise/fall: **and** #(3, 6) G2 (y, a, b, c);

Rise/fall/turnoff: **bufif0** #(3, 6, 5) (y, x\_in, en);

Min:typ:Max: **bufif1** #(3:4:5, 4:5:6, 7,8:9)  
(y, x\_in, en);

Command line options for single delay value simulation:

**+maxdelays**, **+typdelays**, **+mindelays**

*Example:* verilog +mindelays testbench.v

## Concurrent Behavioral Statements

May execute a level-sensitive assignment of value to a net (keyword: **assign**), or may execute the statements of a cyclic (keyword: **always**) or single-pass (keyword: **initial**) behavior. The statements execute sequentially, subject to level-sensitive or edge-sensitive event control expressions.

## Syntax:

**assign** net\_name = [expression];

**always begin** [procedural statements] **end**

**initial begin** [procedural statements] **end**

Cyclic (**always**) and single-pass (**initial**) behaviors may be level sensitive and/or edge sensitive.

## Edge sensitive:

**always @(posedge clock)**

q <= data;

## Level sensitive:

**always @** (enable or data)

**if** (enable) q = data

## Data Types: Nets and Registers

**Nets:** Establish structural connectivity between instantiated primitives and/or modules; may be target of a continuous assignment; e.g., **wire**, **tri**, **wand**, **wor**.

---

---

Value is determined during simulation by the driver of the net; e.g., a primitive or a continuous assignment. (Example: **wire** Y = A + B.)

**Registers:** Store information and retain value until reassigned.

Value is determined by an assignment made by a procedural statement.

Value is retained until a new assignment is made; e.g., **reg**, **integer**, **real**, **realtime**, **time**.

### Example:

```
always @ (posedge clock)
if (reset) q_out <= 0;
else q_out <= data_in;
```

## Procedural Statements

Describe logic abstractly; statements execute sequentially to assign value to variables.

```
if (expression_is_true) statement_1; else
statement_2;
case (case_expression)
case_item: statement;
...
default: statement;
endcase
for (conditions ) statement;
repeat constant_expression statement;
while (expression_is_true) statement;
forever statement;
fork statements join // execute in parallel
```

## Assignments

**Continuous:** Continuously assigns the value of an expression to a net.

**Procedural (Blocked):** Uses the = operator; executes statements sequentially; a statement cannot execute until the preceding statement completes execution. Value is assigned immediately.

**Procedural (Nonblocking):** Uses the <= operator; executes statements concurrently, independent of the order in which they are listed. Values are assigned concurrently.

### Procedural (Continuous):

**assign ... deassign** overrides procedural assignments to a net.

**force ... release** overrides all other assignments to a net or a register.

## Operators

{ }, { { } }	concatenation
+ - * /	arithmetic
%	modulus
> >= < <=	relational
!	logical negation
& &	logical and
	logical or
==	logical equality
!=	logical inequality
===	case equality
!==	case inequality
~	bitwise negation
&	bitwise and
	bitwise or
^	bitwise exclusive-or
^~ or ~^	bitwise equivalence
&	reduction and
~&	reduction nand
!	or
~	reduction nor
^	reduction exclusive-or
~^ or ^~	reduction xnor
<<	left shift
>>	right shift
?:	conditional
or	Event or

## Specify Block

### Example: Module Path Delays

```
specify
// specparam declarations (min: typ: max)
specparam t_r = 3:4:5, t_f = 4:5:6;
(A, B) *> Y) = (t_r, t_f); // full
(Bus_1 => Bus_1) = (t_r, t_f); // parallel
if (state == S0) (a, b *> y) = 2; // state dep
(posedge clk => (y -: d_in)) = (3, 4); // edge
endspecify
```

### Example: Timing Checks

```
specify
specparam t_setup = 3:4:5, t_hold = 4:5:6;

$setup (data, posedge clock, t_setup);
$hold (posedge clock, data, t_hold);
endspecify
```

## Memory

Declares an array of words.

### Example: Memory declaration and readout

```
module memory_read_display();
reg [31: 0] mem_array [1: 1024];
integer k;
```

*Advanced Digital Design  
with the Verilog HDL*

## PRENTICE HALL XILINX DESIGN SERIES



CILETTI	<i>Modeling, Synthesis, and Rapid Prototyping with Verilog HDL</i>
CILETTI	<i>Advanced Digital Design with the Verilog HDL</i>
MANO & KIME	<i>Logic and Computer Design Fundamentals, 2/e</i>
SANDIGE	<i>Digital Design Essentials</i>
WAKERLY	<i>Digital Design Principles and Practices, 3/e</i>
XILINX	<i>Xilinx Student Edition: Foundation Series Software</i>
YALAMANCHILI	<i>Introductory VHDL: From Simulation to Synthesis</i>

---

# Preface

---

## Simplify, Clarify, and Verify

---

Behavioral modeling with a hardware description language (HDL) is the key to modern design of application-specific integrated circuits (ASICs). Today, most designers use an HDL-based design method to create a high-level, language-based, abstract description of a circuit, synthesize a hardware realization in a selected technology, and verify its functionality and timing.

Students preparing to contribute to a productive design team must know how to use an HDL at key stages of the design flow. Thus, there is a need for a course that goes beyond the basic principles and methods learned in a first course in digital design. This book is written for such a course.

Many books discussing HDLs are now available, but most are oriented toward robust explanations of language syntax, and are not well-suited for classroom use. Our focus is on design methodology enabled by an HDL.

Our goal in this book is to build on a student's background from a first course in logic design by (1) reviewing basic principles of combinational and sequential logic, (2) introducing the use of HDLs in design, (3) emphasizing descriptive styles that will allow the reader to quickly design working circuits suitable for ASICs and/or field-programmable gate array (FPGA) implementation, and (4) providing in-depth design examples using modern design tools. Readers will be encouraged to simplify, clarify, and verify their designs.

The widely used Verilog hardware description language (IEEE Standard 1364) serves as a common framework supporting the design activities treated in this book, **but our focus is on developing, verifying, and synthesizing designs of digital circuits, not on the Verilog language.** Most students taking a second course in digital design will be familiar with at least one programming language and will be able to draw on that background in reading this textbook. We cover only the core and most widely used features of Verilog. In order to emphasize *using* the language in a synthesis-oriented design environment, we have purposely placed many details, features, and explanations of syntax in the Appendices for reference on an “as-needed” basis.

Most entry-level courses in digital design introduce state machines, state-transition graphs, and algorithmic-state machine (ASM) charts. We make heavy use of ASM charts and demonstrate their utility in developing behavioral models of sequential machines. The important problem of

---

designing a finite-state machine to control a complex datapath in a digital machine is treated in-depth with ASMD charts (i.e., ASM charts annotated to display the register operations of the controlled datapath). The design of a reduced instruction-set computer central processing unit (RISC CPU) and other important hardware units are given as examples. Our companion website includes the RISC machine's source code and an assembler that can be used to develop programs for applications. The machine also serves as a starting point for developing a more robust instruction set and architectural variants.

The Verilog language is introduced in an integrated, but selective manner, only as needed to support design examples. The text has a large set of examples illustrating how to address the key steps in a very large scale integrated (VLSI) circuit design methodology using the Verilog HDL. Examples are complete, and include source code that has been verified with the Silos-III simulator to be correct. Source code for all of the examples will be available (with important test suites) at our website.

## The Intended Audience

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

## Special Features of the Book

- Begins with a brief review of basic principles in combinational and sequential logic
  - Focuses on modern digital design methodology
  - Illustrates and promotes a synthesis-ready style of register transfer level (RTL) and algorithmic modeling with Verilog
  - Demonstrates the utility of ASM charts for behavioral modeling
  - In-depth treatment of algorithms and architectures for digital machines (e.g., an image processor, digital filters and circular buffers)
  - In-depth treatment of synthesis for cell-based ASICs and FPGAs
  - A practical treatment of timing analysis, fault simulation, testing, and design for testability, with examples
  - Comprehensive treatment of behavioral modeling
  - Comprehensive design examples, including a RISC machine and datapath controller
  - Numerous graphical illustrations
  - Provides several problems with a wide range of difficulty after each chapter
  - Contains a worked example with JTAG and BIST for testing
-

- Contains over 250 fully verified examples
- An indexed list of all models developed in the examples
- A set of Xilinx FPGA-based laboratory-ready exercises linked to the book (e.g., arithmetic and logic unit [ALU], a programmable lock, a key pad scanner with a FIFO, a serial communications link with error correction, an SRAM controller, and first in, first out [FIFO] memory)
- Contains an up-to-date chapter on programmable logic device (PLDs) and FPGAs
- Contains a packaged CD-ROM with the popular Silos-III Verilog design environment and simulator and the Xilinx integrated synthesis environment (ISE) synthesis tool for FPGAs
- Contains an Appendix with full formal syntax of the Verilog HDL
- Covers major features of Verilog 2001, with examples
- Supported by an ongoing website containing:
  1. Source files of models developed in the examples
  2. Source files of testbenches for simulating examples
  3. An Instructor's Classroom Kit containing transparency files for a course based on the subject matter
  4. Solutions to selected problems
  5. Jump-start tutorials helping students get immediate results with the Silos-III simulation environment, the Xilinx FPGA synthesis tool, the Synopsys synthesis tools, and the Synopsys Prime Time static timing analyzer
  6. ASIC standard-cell library with synthesis and timing database
  7. Answers to frequently asked questions (FAQs)
  8. Clever examples submitted by readers
  9. Revisions

## Sequences for Course Presentation

The material in the text begins with a review of combinational and sequential logic design, but then progresses in the order dictated by the design flow for an ASIC or an FPGA. Chapters 1 to 6 treat design topics through synthesis, and should be covered in order, but Chapters 7 to 10 can be covered in any order. The homework exercises are challenging, and the laboratory-ready Xilinx-based exercises are suitable for a companion laboratory or for end-of-semester projects. Chapter 10 presents several architectures for arithmetic operations, affording a diversity of coverage. Chapter 11 treats postsynthesis design validation, timing analysis, fault simulation, and design for testability. The coverage of these topics can be omitted, depending on the level and focus of the course. Tools supporting Verilog 2001 are emerging, so an appendix discusses and illustrates the important new features of the language.

## Chapter Descriptions

Chapter 1 briefly discusses the role of HDLs in design flows for cell-based ASICs and FPGAs. Chapters 2 and 3 review mainstream topics that would be covered in a first course in digital design, using classical methods (i.e. Karnaugh maps). This material will refresh the reader's background, and the examples will be used later to introduce HDL-based methods of design. Chapters 4 and 5



introduce modeling of combinational and sequential logic with the Verilog HDL, and place emphasis on coding styles that are used in behavioral modeling. Chapter 6 addresses cell-based synthesis of ASICs, and introduces synthesis of combinational and sequential logic. Here we pursue two main objectives: (1) present synthesis-friendly coding styles, and (2) form a foundation that will enable the reader to anticipate the results of synthesis, especially when synthesizing sequential machines. Many sequential machines are partitioned into a datapath and a controller. Chapter 7 covers examples that illustrate how to design a controller for a datapath. The designs of a simple RISC CPU and a UART<sup>1</sup> serve as platforms for the subject matter. Chapter 8 covers PLDs, complex PLDs (CPLDs), ROMs, and static random-access memories (SRAMs), then expands the synthesis target to include FPGAs. Verilog has been used extensively to design computers and signal processors. Chapter 9 treats the modeling and synthesis of computational units and algorithms found in computer architectures, digital filters, and other processors. Chapter 10 develops and refines algorithms and architectures for the arithmetic units of digital machines. In Chapter 11 we use the Verilog HDL in conjunction with fault simulators and timing analyzers to revisit a selection of previously designed machines and consider performance/timing issues and testability, to complete the treatment of design flow tasks that rely heavily on designer intervention. Chapter 11 models the test access port (TAP) controller defined by the IEEE 1149.1 standard (commonly known as the JTAG standard), and presents an example of its use. Another elaborate example covers built-in self test (BIST).

## Acknowledgments

The author is grateful for the support of colleagues and students who expanded his vision of Verilog and contributed to this textbook. The reviewers of the original manuscript provided encouragement, critical judgment, and many helpful suggestions. Stu Sutherland helped the author gain a deeper appreciation for the issue of race conditions that can creep into the models of a digital system. These insights led to the disciplined style of adhering to nonblocking assignments for modeling edge-sensitive behavior and blocked assignments for modeling level-sensitive behavior. I owe a debt of gratitude to Dr. Jim Tracy and Dr. Rodger Ziemer, who supported my efforts to develop courses in VLSI circuit design; to Bill Fuchs, who introduced me to the Silos-III Verilog simulator from Simucad, Inc., and placed a user-friendly design environment in the hands of our students. Kirk Sprague and Scott Kukel were helpful in developing a Hamming encoder to work with the UART. Cris Hagan's thesis led to the models presented in Chapter 9 for decimators and other functional units found in digital signal processors. Rex Anderson proofread several chapters and scrubbed down my work. Terry Hansen and Lisa Horton provided the inspiration for the coffee vending machine example, and developed the assembler that supports the RISC CPU. Dr. Greg Sajdak developed material relating chip defects to test coverage and process yield. Dr. Bruce Harmon provided material for a FIR filter example. My editors, Tom Robbins and Eric Frank, have been a delight to work with. They supported the concept, encouraged my work and guided this book through the production process. My deep thanks to all of you.

---

<sup>1</sup>Universal asynchronous receiver and transmitter (UART), a circuit used in data transmission between systems.

## **Dedication**

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

---

# **Introduction to Digital Design Methodology**

---

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

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

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

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

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

## Library of Congress Cataloging-in-Publication Data

Ciletti, Michael D.

Advanced digital design with Verilog HDL / Michael Ciletti.-- 1st ed.  
p. cm. -- (Prentice Hall Xilinx design series)

Includes bibliographical references and index.

ISBN 0-13-089161-4

1. Digital electronics. 2. Logic circuits--Computer-aided design. 3. Verilog (Computer hardware description language) I. Title. II. Series.

TK7868.D5 .C48 2002

621.39'5--dc21

2002074816

Vice President and Editorial Director, ECS: *Marcia J. Horton*

Publisher: *Tom Robbins*

Editorial Assistant: *Jody McDonnell*

Vice President and Director of Production and Manufacturing, ESM: *David W. Riccardi*

Executive Managing Editor: *Vince O'Brien*

Managing Editor: *David A. George*

Production Editor: *Kevin Bradley*

Director of Creative Services: *Paul Belfanti*

Creative Director: *Carole Anson*

Art Director: *Jayne Conte*

Cover Designer: *Bruce Kenselaar*

Art Editor: *Greg Dulles*

Manufacturing Manager: *Trudy Piscioti*

Manufacturing Buyer: *Lynda Castillo*

Marketing Manager: *Holly Stark*

**Prentice  
Hall**

©2003 by Pearson Education, Inc.

Pearson Education, Inc.

Upper Saddle River, NJ 07458

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Silos and Simucad are registered trademarks of Simucad, Inc., 32970 Alvarado-Niles Road, Union City, CA 94587.

Verilog is a registered trademark of Cadence Design Systems, Inc., 2655 Seely Avenue, San Jose, CA 95134.

Printed in the United States of America

10 9 8 7 6 5 4 3

**ISBN 0-13-089161-4**

Pearson Education Ltd., *London*

Pearson Education Australia Pty. Ltd., *Sydney*

Pearson Education Singapore, Pte. Ltd.

Pearson Education North Asia Ltd., *Hong Kong*

Pearson Education Canada, Inc., *Toronto*

Pearson Educación de México, S.A. de C.V.

Pearson Education—Japan, *Tokyo*

Pearson Education Malaysia, Pte. Ltd.

Pearson Education, Inc., *Upper Saddle River, New Jersey*

---

# Contents

---

## **Preface xvii**

Simplify, Clarify, and Verify xviii

## **1 Introduction to Digital Design Methodology 1**

- 1.1 Design Methodology—An Introduction 2
    - 1.1.1 Design Specification 4
    - 1.1.2 Design Partition 4
    - 1.1.3 Design Entry 4
    - 1.1.4 Simulation and Functional Verification 5
    - 1.1.5 Design Integration and Verification 6
    - 1.1.6 Presynthesis Sign-Off 6
    - 1.1.7 Gate-Level Synthesis and Technology Mapping 6
    - 1.1.8 Postsynthesis Design Validation 7
    - 1.1.9 Postsynthesis Timing Analysis 8
    - 1.1.10 Test Generation and Fault Simulation 8
    - 1.1.11 Placement and Routing 8
    - 1.1.12 Physical and Electrical Design Rules 9
    - 1.1.13 Parasitic Extraction 9
    - 1.1.14 Design Sign-Off 9
  - 1.2 IC Technology Options 9
  - 1.3 Overview 11
  - References 11
-

## **2 Review of Combinational Logic Design 13**

- 2.1 Combinational Logic and Boolean Algebra 13
  - 2.1.1 ASIC Library Cells 13
  - 2.1.2 Boolean Algebra 16
  - 2.1.3 DeMorgan's Laws 18
- 2.2 Theorems for Boolean Algebraic Minimization 18
- 2.3 Representation of Combinational Logic 21
  - 2.3.1 Sum of Products Representation 23
  - 2.3.2 Product-of-Sums Representation 26
- 2.4 Simplification of Boolean Expressions 27
  - 2.4.1 Simplification with Exclusive-Or 36
  - 2.4.2 Karnaugh Maps (SOP Form) 36
  - 2.4.3 Karnaugh Maps (POS Form) 39
  - 2.4.4 Karnaugh Maps and Don't-Cares 40
  - 2.4.5 Extended Karnaugh Maps 41
- 2.5 Glitches and Hazards 42
  - 2.5.1 Elimination of Static Hazards (SOP Form) 44
  - 2.5.2 Summary: Elimination of Static Hazards in Two-Level Circuits 48
  - 2.5.3 Static Hazards in Multilevel Circuits 49
  - 2.5.4 Summary: Elimination of Hazards in Multilevel Circuits 52
  - 2.5.5 Dynamic Hazards 52
- 2.6 Building Blocks for Logic Design 55
  - 2.6.1 NAND–NOR Structures 55
  - 2.6.2 Multiplexers 60
  - 2.6.3 Demultiplexers 61
  - 2.6.4 Encoders 62
  - 2.6.5 Priority Encoder 63
  - 2.6.6 Decoder 64
  - 2.6.7 Priority Decoder 66
- References 67
- Problems 67

## **3 Fundamentals of Sequential Logic Design 69**

- 3.1 Storage Elements 69
    - 3.1.1 Latches 70
    - 3.1.2 Transparent latches 71
  - 3.2 Flip-Flops 71
    - 3.2.1 D-Type Flip-Flop 71
    - 3.2.2 Master–Slave Flip-Flop 73
    - 3.2.3 J-K Flip-Flop 75
    - 3.2.4 T Flip-Flop 75
-

---

3.3	Busses and Three-State Devices	76
3.4	Design of Sequential Machines	80
3.5	State-Transition Graphs	82
3.6	Design Example: BCD to Excess-3 Code Converter	84
3.7	Serial-Line Code Converter for Data Transmission	89
3.7.1	A Mealy-Type FSM for Serial Line-Code Conversion	92
3.7.2	A Moore-Type FSM for Serial Line-Code Conversion	93
3.8	State Reduction and Equivalent States	95
	References	99
	Problems	100

## **4 Introduction to Logic Design with Verilog 103**

4.1	Structural Models of Combinational Logic	104
4.1.1	Verilog Primitives and Design Encapsulation	104
4.1.2	Verilog Structural Models	107
4.1.3	Module Ports	107
4.1.4	Some Language Rules	108
4.1.5	Top-Down Design and Nested Modules	108
4.1.6	Design Hierarchy and Source-Code Organization	111
4.1.7	Vectors in Verilog	113
4.1.8	Structural Connectivity	114
4.2	Logic Simulation, Design Verification, and Test Methodology	119
4.2.1	Four-Valued Logic and Signal Resolution in Verilog	119
4.2.2	Test Methodology	120
4.2.3	Signal Generators for Testbenches	123
4.2.4	Event-Driven Simulation	125
4.2.5	Testbench Template	125
4.2.6	Sized Numbers	126
4.3	Propagation Delay	126
4.3.1	Inertial Delay	129
4.3.2	Transport Delay	131
4.4	Truth Table Models of Combinational and Sequential Logic with Verilog	132
	References	140
	Problems	140

## **5 Logic Design with Behavioral Models of Combinational and Sequential Logic 143**

5.1	Behavioral Modeling	143
5.2	A Brief Look at Data Types for Behavioral Modeling	145
5.3	Boolean-Equation-Based Behavioral Models of Combinational Logic	145

---

---

5.4	Propagation Delay and Continuous Assignments	148
5.5	Latches and Level-Sensitive Circuits in Verilog	150
5.6	Cyclic Behavioral Models of Flip-Flops and Latches	153
5.7	Cyclic Behavior and Edge Detection	154
5.8	A Comparison of Styles for Behavioral Modeling	156
5.8.1	Continuous-Assignment Models	156
5.8.2	Dataflow/RTL Models	158
5.8.3	Algorithm-Based Models	162
5.8.4	Port Names: A Matter of Style	164
5.8.5	Simulation with Behavioral Models	164
5.9	Behavioral Models of Multiplexers, Encoders, and Decoders	165
5.10	Dataflow Models of a Linear-Feedback Shift Register	174
5.11	Modeling Digital Machines with Repetitive Algorithms	176
5.11.1	Intellectual Property Reuse and Parameterized Models	181
5.11.2	Clock Generators	183
5.12	Machines with Multicycle Operations	185
5.13	Design Documentation with Functions and Tasks: Legacy or Lunacy?	186
5.13.1	Tasks	187
5.13.2	Functions	189
5.14	Algorithmic State Machine Charts for Behavioral Modeling	190
5.15	ASMD Charts	194
5.16	Behavioral Models of Counters, Shift Registers, and Register Files	196
5.16.1	Counters	197
5.16.2	Shift Registers	203
5.16.3	Register Files and Arrays of Registers (Memories)	207
5.17	Switch Debounce, Metastability, and Synchronizers for Asynchronous Signals	210
5.18	Design Example: Keypad Scanner and Encoder	216
	References	224
	Problems	225

## **6 Synthesis of Combinational and Sequential Logic 233**

6.1	Introduction to Synthesis	234
6.1.1	Logic Synthesis	235
6.1.2	RTL Synthesis	243
6.1.3	High-Level Synthesis	244
6.2	Synthesis of Combinational Logic	245
6.2.1	Synthesis of Priority Structures	250
6.2.2	Exploiting Logical Don't-Care Conditions	251
6.2.3	ASIC Cells and Resource Sharing	256
6.3	Synthesis of Sequential Logic with Latches	258
6.3.1	Accidental Synthesis of Latches	260
6.3.2	Intentional Synthesis of Latches	264

---



---

6.4	Synthesis of Three-State Devices and Bus Interfaces	268
6.5	Synthesis of Sequential Logic with Flip-Flops	271
6.6	Synthesis of Explicit State Machines	275
6.6.1	Synthesis of a BCD-to-Excess-3 Code Converter	275
6.6.2	Synthesis of a Mealy-Type NRZ-to-Manchester Line Code Converter	280
6.6.3	Synthesis of a Moore-Type NRZ-to-Manchester Line Code Converter	282
6.6.4	Synthesis of a Sequence Recognizer	283
6.7	Registered Logic	292
6.8	State Encoding	299
6.9	Synthesis of Implicit State Machines, Registers, and Counters	301
6.9.1	Implicit State Machines	301
6.9.2	Synthesis of Counters	302
6.9.3	Synthesis of Registers	304
6.10	Resets	309
6.11	Synthesis of Gated Clocks and Clock Enables	313
6.12	Anticipating the Results of Synthesis	314
6.12.1	Synthesis of Data Types	314
6.12.2	Operator Grouping	314
6.12.3	Expression Substitution	316
6.13	Synthesis of Loops	319
6.13.1	Static Loops without Embedded Timing Controls	319
6.13.2	Static Loops with Embedded Timing Controls	322
6.13.3	Nonstatic Loops without Embedded Timing Controls	326
6.13.4	Nonstatic Loops with Embedded Timing Controls	328
6.13.5	State-Machine Replacements for Unsynthesizable Loops	331
6.14	Design Traps to Avoid	338
6.15	Divide and Conquer: Partitioning a Design	338
	References	339
	Problems	339

## **7 Design and Synthesis of Datapath Controllers 347**

7.1	Partitioned Sequential Machines	347
7.2	Design Example: Binary Counter	349
7.3	Design and Synthesis of a RISC Stored-Program Machine	355
7.3.1	RISC SPM: Processor	357
7.3.2	RISC SPM: ALU	357
7.3.3	RISC SPM: Controller	357
7.3.4	RISC SPM: Instruction Set	358
7.3.5	RISC SPM: Controller Design	360
7.3.6	RISC SPM: Program Execution	375
7.4	Design Example: UART	378
7.4.1	UART Operation	379