

# SystemVerilog for Verification



Chris Spear • Greg Tumbush

# SystemVerilog for Verification

A Guide to Learning the Testbench  
Language Features

Third Edition



Springer

Chris Spear  
Synopsis, Inc.  
Marlborough, MA, USA

Greg Tumbush  
University of Colorado, Colorado Springs  
Colorado Springs, CO, USA

ISBN 978-1-4614-0714-0 e-ISBN 978-1-4614-0715-7  
DOI 10.1007/978-1-4614-0715-7  
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011945681

© Springer Science+Business Media, LLC 2012

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

*This book is dedicated to my wife Laura, who takes care of everything, my daughter Allie, long may you travel, my son Tyler, welcome back, and all the mice.*

*– Chris Spear*

*This book is dedicated to my wife Carolye, who shrugged off my “I need to work on the book” requests with a patient smile, and to my toddler son Lucca who was always available for play time.*

*– Greg Tumbush*



# Preface

## What is this Book About?

This book should be the first one you read to learn the SystemVerilog verification language constructs. It describes how the language works and includes many examples on how to build a basic coverage-driven, constrained-random, layered testbench using Object-Oriented Programming (OOP). The book has many guidelines on building testbenches, to help you understand how and why to use classes, randomization, and functional coverage. Once you have learned the language, pick up some of the methodology books listed in the References section for more information on building a testbench.

## Who Should Read this Book?

If you create testbenches, you need this book. If you have only written tests using Verilog or VHDL and want to learn SystemVerilog, this book shows you how to move up to the new language features. Vera and Specman users can learn how one language can be used for both design and verification. You may have tried to read the SystemVerilog Language Reference Manual but found it loaded with syntax but no guidelines on which construct to choose.

Chris originally wrote this book because, like many of his customers, he spent much of his career using procedural languages such as C and Verilog to write tests, and had to relearn everything when OOP verification languages came along. He made all the typical mistakes, and wrote this book so you won't have to repeat them.

Before reading this book, you should be comfortable with Verilog-1995. You do not need to know about Verilog-2001 or SystemVerilog design constructs, or SystemVerilog Assertions in order to understand the concepts in this book.

## What is New in the Third Edition?

This new edition of SystemVerilog for Verification has many improvements over the first two editions, written in 2006 and 2008, respectively.

- Our universities need to train future engineers in the art of verification. This edition is suitable for the academic environment, with exercise questions at the end of each chapter to test your understanding.
- Qualified instructors should visit <http://extras.springer.com> for additional materials such as slides, tests, homework problems, solutions, and a sample syllabus suitable for a semester-long course.
- The 2009 version of the IEEE 1800 SystemVerilog Language Reference Manual (LRM) has many changes, both large and small. This book tries to include the latest relevant information.
- Accellera created UVM (Universal Verification Methodology) with ideas from VMM (Verification Methodology Manual), OVM (Open Verification Methodology), eRM (e Reuse Methodology), and other methodologies. Many of the examples in this book are based on VMM because its explicit calling of phases is easier to understand if you are new to verification. New examples are provided that show UVM concepts such as the test registry and configuration database.
- When looking for a specific topic, engineers read books backwards, starting with the index, so we boosted the number of entries.
- Lastly, a big thanks to all the readers who spotted mistakes in the previous editions, from poor grammar to code that was obviously written on the morning after an 18-hour flight from Asia to Boston, or, even worse, changing a diaper. This edition has been checked and reviewed many times over, but once again, all mistakes are ours.

## Why was SystemVerilog Created?

In the late 1990s, the Verilog Hardware Description Language (HDL) became the most widely used language for describing hardware for simulation and synthesis. However, the first two versions standardized by the IEEE (1364-1995 and 1364-2001) had only simple constructs for creating tests. As design sizes outgrew the verification capabilities of the language, commercial Hardware Verification Languages (HVLs) such as OpenVera and *e* were created. Companies that did not want to pay for these tools instead spent hundreds of man-years creating their own custom tools.

This productivity crisis, along with a similar one on the design side, led to the creation of Accellera, a consortium of EDA companies and users who wanted to create the next generation of Verilog. The donation of the OpenVera language formed the basis for the HVL features of SystemVerilog. Accellera's goal was met



in November 2005 with the adoption of the IEEE standard 1800-2005 for SystemVerilog, IEEE (2005). In December 2009, the latest Verilog LRM, 1364-2005, was merged with the aforementioned 2005 SystemVerilog standard to create the IEEE standard 1800-2009 for SystemVerilog. Merging these two standards into a single one means there is now one language, SystemVerilog, for both design and verification.

## **Importance of a Unified Language**

Verification is generally viewed as a fundamentally different activity from design. This split has led to the development of narrowly focused languages for verification and to the bifurcation of engineers into two largely independent disciplines. This specialization has created substantial bottlenecks in terms of communication between the two groups. SystemVerilog addresses this issue with its capabilities for both camps. Neither team has to give up any capabilities it needs to be successful, but the unification of both syntax and semantics of design and verification tools improves communication. For example, while a design engineer may not be able to write an object-oriented testbench environment, it is fairly straightforward to read such a test and understand what is happening, enabling both the design and verification engineers to work together to identify and fix problems. Likewise, a designer understands the inner workings of his or her block, and is the best person to write assertions about it, but a verification engineer may have a broader view needed to create assertions between blocks.

Another advantage of including the design, testbench, and assertion constructs in a single language is that the testbench has easy access to all parts of the environment without requiring a specialized Application Programming Interface (API). The value of an HVL is its ability to create high-level, flexible tests, not its loop constructs or declaration style. SystemVerilog is based on the Verilog, VHDL, and C/C++ constructs that engineers have used for decades.

## **Importance of Methodology**

There is a difference between learning the syntax of a language and learning how to use a tool. This book focuses on techniques for verification using constrained-random tests that use functional coverage to measure progress and direct the verification. As the chapters unfold, language and methodology features are shown side by side. For more on methodology, see Bergeron et al. (2006).

The most valuable benefit of SystemVerilog is that it allows the user to construct reliable, repeatable verification environments, in a consistent syntax, that can be used across multiple projects.

## Overview of the Book

The SystemVerilog language includes features for design, verification, assertions, and more. This book focuses on the constructs used to verify a design. There are many ways to solve a problem using SystemVerilog. This book explains the trade-offs between alternative solutions.

Chapter 1, **Verification Guidelines**, presents verification techniques to serve as a foundation for learning and using the SystemVerilog language. These guidelines emphasize coverage-driven random testing in a layered testbench environment.

Chapter 2, **Data Types**, covers the new SystemVerilog data types such as arrays, structures, enumerated types, and packed arrays and structures.

Chapter 3, **Procedural Statements and Routines**, shows the new procedural statements and improvements for tasks and functions.

Chapter 4, **Connecting the Testbench and Design**, shows the new SystemVerilog verification constructs, such as program blocks, interfaces, and clocking blocks, and how they are used to build your testbench and connect it to the design under test.

Chapter 5, **Basic OOP**, is an introduction to Object-Oriented Programming, explaining how to build classes, construct objects, and use handles.

Chapter 6, **Randomization**, shows you how to use SystemVerilog's constrained-random stimulus generation, including many techniques and examples.

Chapter 7, **Threads and Interprocess Communication**, shows how to create multiple threads in your testbench, use interprocess communication to exchange data between these threads and synchronize them.

Chapter 8, **Advanced OOP and Testbench Guidelines**, shows how to build a layered testbench with OOP so that the components can be shared by all tests.

Chapter 9, **Functional Coverage**, explains the different types of coverage and how you can use functional coverage to measure your progress as you follow a verification plan.

Chapter 10, **Advanced Interfaces**, shows how to use virtual interfaces to simplify your testbench code, connect to multiple design configurations, and create interfaces with procedural code so your testbench and design can work at a higher level of abstraction.

Chapter 11, **A Complete SystemVerilog Testbench**, shows a constrained random testbench using the guidelines shown in Chapter 8. Several tests are shown to demonstrate how you can easily extend the behavior of a testbench without editing the original code, which always carries risk of introducing new bugs.

Chapter 12, **Interfacing with C / C++**, describes how to connect your C or C++ Code to SystemVerilog using the Direct Programming Interface.

## Icons used in this book

**Table i.1** Book icons



The compass shows verification methodology to guide your usage of SystemVerilog testbench features.



The bug shows common coding mistakes such as syntax errors, logic problems, or threading issues.

## About the Authors

Chris Spear has been working in the ASIC design and verification field for 30 years. He started his career with Digital Equipment Corporation (DEC) as a CAD Engineer on DECsim, connecting the first Zycad box ever sold, and then a hardware Verification engineer for the VAX 8600, and a hardware behavioral simulation accelerator. He then moved on to Cadence where he was an Application Engineer for Verilog-XL, followed a a stint at Viewlogic. Chris is currently employed at Synopsys Inc. as a Verification Consultant, a title he created a dozen years ago. He has authored the first and second editions of SystemVerilog for Verification. Chris earned a BSEE from Cornell University in 1981. In his spare time, Chris enjoys road biking in the mountains and traveling with his wife.

Greg Tumbush has been designing and verifying ASICs and FPGAs for 13 years. After working as a researcher in the Air Force Research Labs (AFRL) he moved to beautiful Colorado to work with Astek Corp as a Lead ASIC Design Engineer. He then began a 6 year career with Starkey Labs, AMI Semiconductor, and ON Semiconductor where he was an early adopter of SystemC and SystemVerilog. In 2008, Greg left ON Semiconductor to form Tumbush Enterprises, where he has been consulting clients in the areas of design, verification, and backend to ensure first pass success. He is also a 1/2 time Instructor at the University of Colorado, Colorado Springs where he teaches senior and graduate level digital design and verification courses. He has numerous publications which can be viewed at [www.tumbush.com](http://www.tumbush.com). Greg earned a PhD from the University of Cincinnati in 1998.

## Final comments

If you would like more information on SystemVerilog and Verification, you can find many resources at: <http://chris.spear.net/systemverilog>. This site has the source code for many of the examples in this book. Academics who want to use this book in their classes can access slides, tests, homework problems, solutions, and a sample syllabus on the book's webpage at <http://www.springer.com>.

Most of the code samples in the book were verified with Synopsys' Chronologic VCS, Mentor's QuestaSim, and Cadence Incisive. Any errors were caused by Chris' evil twin, Skippy. If you think you have found a mistake in this book, please check his web site for the Errata page. If you are the first to find a technical mistake in a chapter, we will send you a free, autographed book. Please include "SystemVerilog" in the subject line of your email.

Chris Spear  
Greg Tumbush

# Acknowledgments

We thank all the people who spent countless hours helping us learn SystemVerilog and reviewing the book that you now hold in your hands. We especially would like to thank all the people at Synopsys and Cadence for their help. Thanks to Mentor Graphics for supplying Questa licenses through the Questa Vanguard program, and to Tim Plyant at Cadence who checked hundreds of examples for us.

A big thanks to Mark Azadpour, Mark Barrett, Shalom Bresticker, James Chang, Benjamin Chin, Cliff Cummings, Al Czamara, Chris Felton, Greg Mann, Ronald Mehler, Holger Meiners, Don Mills, Mike Mintz, Brad Pierce, Tim Plyant, Stuart Sutherland, Thomas Tessier, and Jay Tyer, plus Professor Brent Nelson and his students who reviewed some very rough drafts and inspired many improvements. However, the mistakes are all ours!

Janick Bergeron provided inspiration, innumerable verification techniques, and top-quality reviews. Without his guidance, this book would not exist.

The following people pointed out mistakes in the second edition, and made valuable suggestions on areas where the book could be improved: Alok Agrawal, Ching-Chi Chang, Cliff Cummings, Ed D'Avignon, Xiaobin Chu, Jaikumar Devaraj, Cory Dearing, Tony Hsu, Dave Hamilton, Ken Imboden, Brian Jensen, Jim Kann, John Keen, Amirtha Kasturi, Devendra Kumar, John Mcandrew, Chet Nibby, Eric Ohana, Simon Peter, Duc Pham, Hani Poly, Robert Qi, Ranbir Rana, Dan Shupe, Alex Seibulescu, Neill Shepherd, Daniel Wei, Randy Wetzel, Jeff Yang, Dan Yingling and Hualong Zhao.

Lastly, a big thanks to Jay Mcinerney for his brash pronoun usage.

All trademarks and copyrights are the property of their respective owners. If you can't take a joke, don't sue us.



# Contents

<b>1</b>	<b>Verification Guidelines.....</b>	<b>1</b>
1.1	The Verification Process .....	2
1.1.1	Testing at Different Levels .....	3
1.1.2	The Verification Plan.....	4
1.2	The Verification Methodology Manual.....	4
1.3	Basic Testbench Functionality .....	5
1.4	Directed Testing.....	5
1.5	Methodology Basics .....	6
1.6	Constrained-Random Stimulus.....	8
1.7	What Should You Randomize?.....	9
1.7.1	Device and Environment Configuration.....	9
1.7.2	Input Data.....	10
1.7.3	Protocol Exceptions, Errors, and Violations .....	10
1.7.4	Delays and Synchronization.....	11
1.7.5	Parallel Random Testing .....	11
1.8	Functional Coverage.....	12
1.8.1	Feedback from Functional Coverage to Stimulus.....	12
1.9	Testbench Components .....	13
1.10	Layered Testbench.....	14
1.10.1	A Flat Testbench.....	14
1.10.2	The Signal and Command Layers.....	17
1.10.3	The Functional Layer .....	17
1.10.4	The Scenario Layer .....	18
1.10.5	The Test Layer and Functional Coverage .....	18
1.11	Building a Layered Testbench.....	19
1.11.1	Creating a Simple Driver .....	20
1.12	Simulation Environment Phases .....	20
1.13	Maximum Code Reuse.....	21
1.14	Testbench Performance .....	22
1.15	Conclusion.....	22
1.16	Exercises .....	23

<b>2</b>	<b>Data Types .....</b>	<b>25</b>
2.1	Built-In Data Types.....	25
2.1.1	The <code>Logic</code> Type.....	26
2.1.2	2-State Data Types .....	26
2.2	Fixed-Size Arrays .....	27
2.2.1	Declaring and Initializing Fixed-Size Arrays.....	28
2.2.2	The Array Literal.....	29
2.2.3	Basic Array Operations — <i>for</i> and <i>Foreach</i> .....	30
2.2.4	Basic Array Operations – Copy and Compare .....	32
2.2.5	Bit and Array Subscripts, Together at Last .....	32
2.2.6	Packed Arrays.....	33
2.2.7	Packed Array Examples .....	33
2.2.8	Choosing Between Packed and Unpacked Arrays .....	34
2.3	Dynamic Arrays .....	35
2.4	Queues .....	36
2.5	Associative Arrays .....	38
2.6	Array Methods .....	41
2.6.1	Array Reduction Methods .....	41
2.6.2	Array Locator Methods .....	42
2.6.3	Array Sorting and Ordering .....	44
2.6.4	Building a Scoreboard with Array Locator Methods .....	45
2.7	Choosing a Storage Type .....	46
2.7.1	Flexibility .....	46
2.7.2	Memory Usage.....	46
2.7.3	Speed.....	47
2.7.4	Data Access .....	47
2.7.5	Choosing the Best Data Structure .....	48
2.8	Creating New Types with <code>typedef</code> .....	48
2.9	Creating User-Defined Structures.....	50
2.9.1	<i>Creating a Struct and a New Type</i> .....	50
2.9.2	Initializing a Structure.....	51
2.9.3	Making a Union of Several Types.....	51
2.9.4	Packed Structures .....	52
2.9.5	Choosing Between Packed and Unpacked Structures.....	52
2.10	Packages .....	53
2.11	Type Conversion .....	54
2.11.1	The Static Cast .....	54
2.11.2	The Dynamic Cast.....	55
2.12	Streaming Operators.....	55
2.13	Enumerated Types.....	57
2.13.1	Defining Enumerated Values.....	58
2.13.2	Routines for Enumerated Types .....	59
2.13.3	Converting to and from Enumerated Types .....	60



2.14	Constants .....	61
2.15	Strings .....	61
2.16	Expression Width .....	62
2.17	Conclusion.....	63
2.18	Exercises .....	64
<b>3</b>	<b>Procedural Statements and Routines .....</b>	<b>69</b>
3.1	Procedural Statements .....	69
3.2	Tasks, Functions, and Void Functions .....	71
3.3	Task and Function Overview .....	72
3.3.1	Routine Begin...End Removed .....	72
3.4	Routine Arguments.....	72
3.4.1	C-style Routine Arguments .....	72
3.4.2	Argument Direction .....	73
3.4.3	Advanced Argument Types .....	73
3.4.4	Default Value for an Argument .....	75
3.4.5	Passing Arguments by Name .....	76
3.4.6	Common Coding Errors .....	77
3.5	Returning from a Routine.....	78
3.5.1	The Return Statement.....	78
3.5.2	Returning an Array from a Function .....	78
3.6	Local Data Storage.....	79
3.6.1	Automatic Storage.....	80
3.6.2	Variable Initialization .....	80
3.7	Time Values.....	81
3.7.1	Time Units and Precision .....	81
3.7.2	Time Literals .....	82
3.7.3	Time and Variables.....	82
3.7.4	\$time vs. \$realtime .....	83
3.8	Conclusion.....	83
3.9	Exercises .....	83
<b>4</b>	<b>Connecting the Testbench and Design.....</b>	<b>87</b>
4.1	Separating the Testbench and Design.....	88
4.1.1	Communication Between the Testbench and DUT .....	88
4.1.2	Communication with Ports.....	89
4.2	The Interface Construct.....	90
4.2.1	Using an Interface to Simplify Connections .....	91
4.2.2	Connecting Interfaces and Ports.....	93
4.2.3	Grouping Signals in an Interface Using Modports .....	94
4.2.4	Using Modports with a Bus Design .....	95
4.2.5	Creating an Interface Monitor .....	95
4.2.6	Interface Trade-Offs .....	96
4.2.7	More Information and Examples .....	97
4.2.8	Logic vs. Wire in an Interface .....	97

4.3	Stimulus Timing .....	98
4.3.1	Controlling Timing of Synchronous Signals with a Clocking Block.....	98
4.3.2	Timing Problems in Verilog .....	99
4.3.3	Testbench – Design Race Condition .....	100
4.3.4	The Program Block and Timing Regions.....	101
4.3.5	Specifying Delays Between the Design and Testbench ...	103
4.4	Interface Driving and Sampling.....	104
4.4.1	Interface Synchronization .....	104
4.4.2	Interface Signal Sample .....	105
4.4.3	Interface Signal Drive .....	106
4.4.4	Driving Interface Signals Through a Clocking Block.....	106
4.4.5	Bidirectional Signals in the Interface.....	108
4.4.6	Specifying Delays in Clocking Blocks .....	109
4.5	Program Block Considerations .....	110
4.5.1	The End of Simulation .....	110
4.5.2	Why are Always Blocks not Allowed in a Program? .....	111
4.5.3	The Clock Generator .....	111
4.6	Connecting It All Together.....	112
4.6.1	An Interface in a Port List Must Be Connected.....	113
4.7	Top-Level Scope .....	114
4.8	Program–Module Interactions.....	115
4.9	SystemVerilog Assertions.....	116
4.9.1	Immediate Assertions.....	116
4.9.2	Customizing the Assertion Actions .....	117
4.9.3	Concurrent Assertions .....	118
4.9.4	Exploring Assertions .....	118
4.10	The Four-Port ATM Router.....	119
4.10.1	ATM Router with Ports .....	119
4.10.2	ATM Top-Level Module with Ports .....	120
4.10.3	Using Interfaces to Simplify Connections .....	123
4.10.4	ATM Interfaces.....	124
4.10.5	ATM Router Model Using an Interface .....	124
4.10.6	ATM Top Level Module with Interfaces.....	125
4.10.7	ATM Testbench with Interface.....	125
4.11	The Ref Port Direction.....	126
4.12	Conclusion.....	127
4.13	Exercises .....	128
<b>5</b>	<b>Basic OOP.....</b>	<b>131</b>
5.1	Introduction .....	131
5.2	Think of Nouns, not Verbs .....	132
5.3	Your First Class.....	133
5.4	Where to Define a Class.....	133
5.5	OOP Terminology .....	134

5.6	Creating New Objects .....	135
5.6.1	Handles and Constructing Objects .....	135
5.6.2	Custom Constructor .....	136
5.6.3	Separating the Declaration and Construction.....	137
5.6.4	The Difference Between New() and New[] .....	138
5.6.5	Getting a Handle on Objects .....	138
5.7	Object Deallocation.....	139
5.8	Using Objects.....	140
5.9	Class Methods .....	141
5.10	Defining Methods Outside of the Class .....	142
5.11	Static Variables vs. Global Variables.....	143
5.11.1	A Simple Static Variable .....	143
5.11.2	Accessing Static Variables Through the Class Name.....	144
5.11.3	Initializing Static Variables .....	145
5.11.4	Static Methods.....	145
5.12	Scoping Rules.....	146
5.12.1	What is This?.....	148
5.13	Using One Class Inside Another .....	149
5.13.1	How Big or Small Should My Class Be?.....	151
5.13.2	Compilation Order Issue .....	151
5.14	Understanding Dynamic Objects .....	152
5.14.1	Passing Objects and Handles to Methods .....	152
5.14.2	Modifying a Handle in a Task .....	153
5.14.3	Modifying Objects in Flight.....	154
5.14.4	Arrays of Handles .....	155
5.15	Copying Objects.....	156
5.15.1	Copying an Object with the <i>New</i> Operator.....	156
5.15.2	Writing Your Own Simple Copy Function.....	158
5.15.3	Writing a Deep Copy Function .....	159
5.15.4	Packing Objects to and from Arrays Using Streaming Operators.....	161
5.16	Public vs. Local .....	162
5.17	Straying Off Course .....	163
5.18	Building a Testbench.....	163
5.19	Conclusion.....	164
5.20	Exercises .....	165
<b>6</b>	<b>Randomization .....</b>	<b>169</b>
6.1	Introduction .....	169
6.2	What to Randomize.....	170
6.2.1	Device Configuration .....	170
6.2.2	Environment Configuration.....	171
6.2.3	Primary Input Data.....	171
6.2.4	Encapsulated Input Data .....	171
6.2.5	Protocol Exceptions, Errors, and Violations .....	172
6.2.6	Delays.....	172

6.3	Randomization in SystemVerilog.....	172
6.3.1	Simple Class with Random Variables .....	173
6.3.2	Checking the Result from Randomization .....	174
6.3.3	The Constraint Solver .....	175
6.3.4	What can be Randomized?.....	175
6.4	Constraint Details.....	175
6.4.1	Constraint Introduction .....	176
6.4.2	Simple Expressions .....	176
6.4.3	Equivalence Expressions.....	177
6.4.4	Weighted Distributions.....	177
6.4.5	Set Membership and the Inside Operator.....	179
6.4.6	Using an Array in a Set .....	180
6.4.7	Bidirectional Constraints.....	183
6.4.8	Implication Constraints .....	184
6.4.9	Equivalence Operator.....	186
6.5	Solution Probabilities .....	186
6.5.1	Unconstrained .....	187
6.5.2	Implication .....	187
6.5.3	Implication and Bidirectional Constraints .....	188
6.5.4	Guiding Distribution with <code>Solve...Before</code> .....	189
6.6	Controlling Multiple Constraint Blocks.....	191
6.7	Valid Constraints .....	192
6.8	In-Line Constraints.....	192
6.9	The <code>pre_randomize</code> and <code>post_randomize</code> Functions.....	193
6.9.1	Building a Bathtub Distribution .....	193
6.9.2	Note on Void Functions.....	195
6.10	Random Number Functions .....	195
6.11	Constraints Tips and Techniques.....	196
6.11.1	Constraints with Variables.....	196
6.11.2	Using Nonrandom Values .....	197
6.11.3	Checking Values Using Constraints .....	198
6.11.4	Randomizing Individual Variables .....	198
6.11.5	Turn Constraints Off and On.....	198
6.11.6	Specifying a Constraint in a Test Using In-Line Constraints.....	199
6.11.7	Specifying a Constraint in a Test with External Constraints.....	199
6.11.8	Extending a Class .....	200
6.12	Common Randomization Problems .....	200
6.12.1	Use Signed Variables with Care .....	201
6.12.2	Solver Performance Tips .....	202
6.12.3	Choose the Right Arithmetic Operator to Boost Efficiency .....	202
6.13	Iterative and Array Constraints .....	203
6.13.1	Array Size.....	203
6.13.2	Sum of Elements .....	203

6.13.3	Issues with Array Constraints.....	205
6.13.4	Constraining Individual Array and Queue Elements .....	207
6.13.5	Generating an Array of Unique Values .....	208
6.13.6	Randomizing an Array of Handles.....	211
6.14	Atomic Stimulus Generation vs. Scenario Generation .....	211
6.14.1	An Atomic Generator with History .....	212
6.14.2	Random Array of Objects.....	212
6.14.3	Combining Sequences.....	213
6.14.4	Randsequence.....	213
6.15	Random Control.....	215
6.15.1	Introduction to <code>randcase</code> .....	215
6.15.2	Building a Decision Tree with <code>randcase</code> .....	216
6.16	Random Number Generators.....	217
6.16.1	Pseudorandom Number Generators .....	217
6.16.2	Random Stability — Multiple Generators .....	217
6.16.3	Random Stability and Hierarchical Seeding .....	219
6.17	Random Device Configuration.....	220
6.18	Conclusion.....	223
6.19	Exercises .....	224
<b>7</b>	<b>Threads and Interprocess Communication .....</b>	<b>229</b>
7.1	Working with Threads.....	230
7.1.1	Using <code>fork...join</code> and <code>begin...end</code> .....	231
7.1.2	Spawning Threads with <code>fork...join_none</code> .....	232
7.1.3	Synchronizing Threads with <code>fork...join_any</code> .....	233
7.1.4	Creating Threads in a Class.....	234
7.1.5	Dynamic Threads .....	235
7.1.6	Automatic Variables in Threads .....	236
7.1.7	Waiting for all Spawned Threads .....	238
7.1.8	Sharing Variables Across Threads.....	239
7.2	Disabling Threads .....	240
7.2.1	Disabling a Single Thread.....	241
7.2.2	Disabling Multiple Threads.....	241
7.2.3	Disable a Task that was Called Multiple Times .....	243
7.3	Interprocess Communication .....	244
7.4	Events.....	244
7.4.1	Blocking on the Edge of an Event.....	245
7.4.2	Waiting for an Event Trigger.....	245
7.4.3	Using Events in a Loop .....	246
7.4.4	Passing Events.....	247
7.4.5	Waiting for Multiple Events.....	248
7.5	Semaphores .....	250
7.5.1	Semaphore Operations .....	251
7.5.2	Semaphores with Multiple Keys .....	252

7.6	Mailboxes .....	252
7.6.1	Mailbox in a Testbench.....	255
7.6.2	Bounded Mailboxes .....	256
7.6.3	Unsynchronized Threads Communicating with a Mailbox.....	257
7.6.4	Synchronized Threads Using a Bounded Mailbox and a Peek .....	259
7.6.5	Synchronized Threads Using a Mailbox and Event.....	261
7.6.6	Synchronized Threads Using Two Mailboxes .....	262
7.6.7	Other Synchronization Techniques .....	264
7.7	Building a Testbench with Threads and IPC.....	264
7.7.1	Basic Transactor.....	265
7.7.2	Configuration Class .....	266
7.7.3	Environment Class.....	266
7.7.4	Test Program .....	267
7.8	Conclusion .....	268
7.9	Exercises .....	269
<b>8</b>	<b>Advanced OOP and Testbench Guidelines .....</b>	<b>273</b>
8.1	Introduction to Inheritance .....	274
8.1.1	Basic Transaction.....	275
8.1.2	Extending the <code>Transaction</code> Class.....	275
8.1.3	More OOP Terminology .....	277
8.1.4	Constructors in Extended Classes.....	277
8.1.5	Driver Class .....	278
8.1.6	Simple Generator Class .....	279
8.2	Blueprint Pattern.....	280
8.2.1	The <code>Environment</code> Class.....	281
8.2.2	A Simple Testbench .....	282
8.2.3	Using the Extended <code>Transaction</code> Class .....	283
8.2.4	Changing Random Constraints with an Extended Class.....	283
8.3	Downcasting and Virtual Methods .....	284
8.3.1	Downcasting with <code>\$cast</code> .....	284
8.3.2	Virtual Methods .....	286
8.3.3	Signatures and Polymorphism .....	288
8.3.4	Constructors are Never Virtual .....	288
8.4	Composition, Inheritance, and Alternatives.....	288
8.4.1	Deciding Between Composition and Inheritance .....	288
8.4.2	Problems with Composition .....	289
8.4.3	Problems with Inheritance .....	291
8.4.4	A Real-World Alternative .....	292
8.5	Copying an Object .....	293
8.5.1	Specifying a Destination for Copy .....	294
8.6	Abstract Classes and Pure Virtual Methods.....	295

8.7	Callbacks .....	297
8.7.1	Creating a Callback .....	298
8.7.2	Using a Callback to Inject Disturbances .....	299
8.7.3	A Quick Introduction to Scoreboards.....	300
8.7.4	Connecting to the Scoreboard with a Callback .....	300
8.7.5	Using a Callback to Debug a Transactor.....	302
8.8	Parameterized Classes .....	302
8.8.1	A Simple Stack.....	302
8.8.2	Sharing Parameterized Classes .....	305
8.8.3	Parameterized Class Suggestions.....	305
8.9	Static and Singleton Classes.....	306
8.9.1	Dynamic Class to Print Messages .....	306
8.9.2	Singleton Class to Print Messages .....	307
8.9.3	Configuration Database with Static Parameterized Class .....	308
8.10	Creating a Test Registry .....	311
8.10.1	Test registry with Static Methods.....	311
8.10.2	Test Registry with a Proxy Class .....	313
8.10.3	UVM Factory Build .....	319
8.11	Conclusion.....	319
8.12	Exercises .....	320
<b>9</b>	<b>Functional Coverage.....</b>	<b>323</b>
9.1	Gathering Coverage Data.....	324
9.2	Coverage Types .....	326
9.2.1	Code Coverage .....	326
9.2.2	Functional Coverage .....	327
9.2.3	Bug Rate.....	327
9.2.4	Assertion Coverage .....	328
9.3	Functional Coverage Strategies.....	328
9.3.1	Gather Information, not Data .....	328
9.3.2	Only Measure What you are Going to Use .....	329
9.3.3	Measuring Completeness .....	329
9.4	Simple Functional Coverage Example.....	330
9.5	Anatomy of a Cover Group.....	333
9.5.1	Defining a Cover Group in a Class .....	334
9.6	Triggering a Cover Group.....	335
9.6.1	Sampling Using a Callback.....	335
9.6.2	Cover Group with a User Defined Sample Argument List .....	336
9.6.3	Cover Group with an Event Trigger.....	337
9.6.4	Triggering on a System Verilog Assertion .....	337
9.7	Data Sampling .....	338
9.7.1	Individual Bins and Total Coverage.....	338
9.7.2	Creating Bins Automatically .....	339

9.7.3	Limiting the Number of Automatic Bins Created.....	339
9.7.4	Sampling Expressions .....	340
9.7.5	User-Defined Bins Find a Bug .....	341
9.7.6	Naming the Cover Point Bins .....	342
9.7.7	Conditional Coverage.....	344
9.7.8	Creating Bins for Enumerated Types .....	345
9.7.9	Transition Coverage .....	345
9.7.10	Wildcard States and Transitions.....	346
9.7.11	Ignoring Values .....	346
9.7.12	Illegal Bins .....	347
9.7.13	State Machine Coverage .....	347
9.8	Cross Coverage .....	348
9.8.1	Basic Cross Coverage Example .....	348
9.8.2	Labeling Cross Coverage Bins.....	349
9.8.3	Excluding Cross Coverage Bins.....	351
9.8.4	Excluding Cover Points from the Total Coverage Metric.....	351
9.8.5	Merging Data from Multiple Domains .....	352
9.8.6	Cross Coverage Alternatives .....	352
9.9	Generic Cover Groups.....	354
9.9.1	Pass Cover Group Arguments by Value .....	354
9.9.2	Pass Cover Group Arguments by Reference.....	355
9.10	Coverage Options.....	355
9.10.1	Per-Instance Coverage.....	355
9.10.2	Cover Group Comment.....	356
9.10.3	Coverage Threshold .....	357
9.10.4	Printing the Empty Bins .....	357
9.10.5	Coverage Goal.....	357
9.11	Analyzing Coverage Data .....	358
9.12	Measuring Coverage Statistics During Simulation.....	359
9.13	Conclusion.....	360
9.14	Exercises .....	360
<b>10</b>	<b>Advanced Interfaces.....</b>	<b>363</b>
10.1	Virtual Interfaces with the ATM Router.....	364
10.1.1	The Testbench with Just Physical Interfaces.....	364
10.1.2	Testbench with Virtual Interfaces.....	366
10.1.3	Connecting the Testbench to an Interface in Port List.....	369
10.1.4	Connecting the Test to an Interface with an XMR.....	370
10.2	Connecting to Multiple Design Configurations .....	372
10.2.1	A Mesh Design.....	372
10.2.2	Using <code>Typedefs</code> with Virtual Interfaces .....	375
10.2.3	Passing Virtual Interface Array Using a Port .....	376
10.3	Parameterized Interfaces and Virtual Interfaces.....	377



10.4	Procedural Code in an Interface.....	379
10.4.1	Interface with Parallel Protocol.....	380
10.4.2	Interface with Serial Protocol.....	380
10.4.3	Limitations of Interface Code .....	381
10.5	Conclusion.....	382
10.6	Exercises .....	382
<b>11</b>	<b>A Complete SystemVerilog Testbench .....</b>	<b>385</b>
11.1	Design Blocks .....	385
11.2	Testbench Blocks .....	390
11.3	Alternate Tests.....	411
11.3.1	Your First Test - Just One Cell .....	411
11.3.2	Randomly Drop Cells.....	412
11.4	Conclusion.....	413
11.5	Exercises .....	414
<b>12</b>	<b>Interfacing with C/C++ .....</b>	<b>415</b>
12.1	Passing Simple Values.....	416
12.1.1	Passing Integer and Real Values .....	416
12.1.2	The Import Declaration .....	416
12.1.3	Argument Directions .....	417
12.1.4	Argument Types .....	418
12.1.5	Importing a Math Library Routine.....	419
12.2	Connecting to a Simple C Routine.....	419
12.2.1	A Counter with Static Storage.....	420
12.2.2	The Chandle Data Type.....	421
12.2.3	Representation of Packed Values .....	423
12.2.4	4-State Values.....	424
12.2.5	Converting from 2-State to 4-State .....	426
12.3	Connecting to C++ .....	427
12.3.1	The Counter in C++ .....	427
12.3.2	Static Methods.....	428
12.3.3	Communicating with a Transaction Level C++ Model .....	428
12.4	Simple Array Sharing.....	432
12.4.1	Single Dimension Arrays - 2-State.....	432
12.4.2	Single Dimension Arrays - 4-State.....	433
12.5	Open arrays .....	434
12.5.1	Basic Open Array .....	434
12.5.2	Open Array Methods .....	435
12.5.3	Passing Unsized Open Arrays .....	435
12.5.4	Packed Open Arrays in DPI .....	436
12.6	Sharing Composite Types.....	437
12.6.1	Passing Structures Between SystemVerilog and C .....	438
12.6.2	Passing Strings Between SystemVerilog and C .....	439

12.7	Pure and Context Imported Methods.....	440
12.8	Communicating from C to SystemVerilog .....	441
12.8.1	A simple Exported Function.....	441
12.8.2	C function Calling SystemVerilog Function .....	442
12.8.3	C Task Calling SystemVerilog Task.....	444
12.8.4	Calling Methods in Objects.....	446
12.8.5	The Meaning of Context .....	449
12.8.6	Setting the Scope for an Imported Routine .....	450
12.9	Connecting Other Languages .....	452
12.10	Conclusion .....	453
12.11	Exercises .....	453
<b>References .....</b>		<b>455</b>
<b>Index.....</b>		<b>457</b>

# List of Figures

Fig. 1.1	Directed test progress over time.....	6
Fig. 1.2	Directed test coverage .....	6
Fig. 1.3	Constrained-random test progress over time vs. directed testing .....	7
Fig. 1.4	Constrained-random test coverage .....	8
Fig. 1.5	Coverage convergence .....	8
Fig. 1.6	Test progress with and without feedback .....	12
Fig. 1.7	The testbench — design environment.....	13
Fig. 1.8	Testbench components .....	14
Fig. 1.9	Signal and command layers .....	17
Fig. 1.10	Testbench with functional layer added.....	17
Fig. 1.11	Testbench with scenario layer added.....	18
Fig. 1.12	Full testbench with all layers.....	19
Fig. 1.13	Connections for the driver.....	20
Fig. 2.1	Unpacked array storage.....	29
Fig. 2.2	Packed array layout .....	33
Fig. 2.3	Packed array bit layout.....	34
Fig. 2.4	Associative array .....	39
Fig. 4.1	The testbench – design environment.....	87
Fig. 4.2	Testbench – Arbiter without interfaces .....	89
Fig. 4.3	An interface straddles two modules .....	91
Fig. 4.4	Main regions inside a SystemVerilog time step.....	102
Fig. 4.5	A clocking block synchronizes the DUT and testbench.....	104
Fig. 4.6	Sampling a synchronous interface .....	106
Fig. 4.7	Driving a synchronous interface .....	108
Fig. 4.8	Testbench – ATM router diagram without interfaces .....	119
Fig. 4.9	Testbench - router diagram with interfaces.....	123
Fig. 5.1	Handles and objects after allocating multiple objects.....	138
Fig. 5.2	Static variables in a class.....	144

Fig. 5.3	Contained objects Sample 5.22 .....	149
Fig. 5.4	Handles and objects across methods .....	152
Fig. 5.5	Objects and handles before copy with the <code>new</code> operator.....	157
Fig. 5.6	Objects and handles after copy with the <code>new</code> operator.....	158
Fig. 5.7	Both src and dst objects refer to a single statistics object and see updated startT value .....	158
Fig. 5.8	Objects and handles after deep copy .....	160
Fig. 5.9	Layered testbench.....	163
Fig. 6.1	Building a bathtub distribution.....	194
Fig. 6.2	Random strobe waveforms.....	204
Fig. 6.3	Sharing a single random generator.....	218
Fig. 6.4	First generator uses additional values .....	218
Fig. 6.5	Separate random generators per object .....	219
Fig. 7.1	Testbench environment blocks.....	230
Fig. 7.2	<code>Fork...join</code> blocks .....	230
Fig. 7.3	<code>Fork...join</code> block .....	231
Fig. 7.4	<code>Fork...join</code> block diagram .....	242
Fig. 7.5	A mailbox connecting two transactors .....	252
Fig. 7.6	A mailbox with multiple handles to one object.....	254
Fig. 7.7	A mailbox with multiple handles to multiple objects.....	254
Fig. 7.8	Layered testbench with environment .....	265
Fig. 8.1	Simplified layered testbench .....	274
Fig. 8.2	Base Transaction class diagram.....	275
Fig. 8.3	Extended Transaction class diagram .....	276
Fig. 8.4	Blueprint pattern generator .....	280
Fig. 8.5	Blueprint generator with new pattern.....	280
Fig. 8.6	Simplified extended transaction .....	285
Fig. 8.7	Multiple inheritance problem.....	292
Fig. 8.8	Callback flow .....	297
Fig. 9.1	Coverage convergence .....	324
Fig. 9.2	Coverage flow .....	325
Fig. 9.3	Bug rate during a project.....	327
Fig. 9.4	Coverage comparison.....	330
Fig. 9.5	Uneven probability for packet length.....	358
Fig. 9.6	Even probability for packet length with <code>solve...before</code> .....	359
Fig. 10.1	Router and testbench with interfaces .....	366
Fig. 11.1	The testbench — design environment.....	386
Fig. 11.2	Block diagram for the squat design.....	386
Fig. 12.1	Storage of a 40-bit 2-state variable .....	424
Fig. 12.2	Storage of a 40-bit 4-state variable .....	425

# List of Tables

Table 2.1	ALU Opcodes .....	67
Table 4.1	Primary SystemVerilog scheduling regions.....	102
Table 4.2	AHB Signal Description.....	128
Table 6.1	Solutions for bidirectional constraint .....	184
Table 6.2	Implication operator truth table .....	184
Table 6.3	Equivalence operator truth table.....	186
Table 6.4	Solutions for Unconstrained class .....	187
Table 6.5	Solutions for Imp1 class.....	188
Table 6.6	Solutions for Imp2 class.....	189
Table 6.7	Solutions for solve x before y constraint .....	190
Table 6.8	Solutions for solve y before x constraint .....	190
Table 6.9	Solution probabilities.....	225
Table 8.1	Comparing inheritance to composition .....	289
Table 12.1	Data types mapping between SystemVerilog and C.....	418
Table 12.2	4-state bit encoding.....	424
Table 12.3	Open array query functions .....	435
Table 12.4	Open array locator functions .....	435



# List of Samples

Sample 1.1	Driving the APB pins .....	15
Sample 1.2	A task to drive the APB pins .....	16
Sample 1.3	Low-level Verilog test .....	16
Sample 1.4	Basic transactor code.....	20
Sample 2.1	Using the logic type.....	26
Sample 2.2	Signed data types.....	27
Sample 2.3	Checking for 4-state values .....	27
Sample 2.4	Declaring fixed-size arrays.....	28
Sample 2.5	Calculating the address width for a memory .....	28
Sample 2.6	Declaring and using multi-dimensional arrays.....	28
Sample 2.7	Unpacked array declarations .....	29
Sample 2.8	Initializing an array .....	29
Sample 2.9	Printing with %p print specifier .....	30
Sample 2.10	Using arrays with <code>for-</code> and <code>foreach</code> loops .....	30
Sample 2.11	Initialize and step through a multi-dimensional array.....	30
Sample 2.12	Output from printing multi-dimensional array values.....	31
Sample 2.13	Printing a multi-dimensional array.....	31
Sample 2.14	Output from printing multi-dimensional array values.....	31
Sample 2.15	Array copy and compare operations.....	32
Sample 2.16	Using word and bit subscripts together .....	33
Sample 2.17	Packed array declaration and usage.....	33
Sample 2.18	Declaration for a mixed packed/unpacked array .....	34
Sample 2.19	Using dynamic arrays.....	35
Sample 2.20	Using a dynamic array for an uncounted list.....	36
Sample 2.21	Multi-dimensional dynamic array .....	36
Sample 2.22	Queue methods .....	37
Sample 2.23	Queue operations.....	38
Sample 2.24	Declaring, initializing, and using associative arrays .....	39
Sample 2.25	Using an associative array with a string index .....	40
Sample 2.26	Initializing and printing associative arrays.....	41

Sample 2.27	Array reduction operations .....	41
Sample 2.28	Picking a random element from an associative array .....	42
Sample 2.29	Array locator methods: <code>min</code> , <code>max</code> , <code>unique</code> .....	42
Sample 2.30	Array locator methods: <code>find</code> .....	43
Sample 2.31	Declaring the iterator argument.....	43
Sample 2.32	Array locator methods .....	43
Sample 2.33	Creating the sum of an array of single bits.....	44
Sample 2.34	Sorting an array .....	44
Sample 2.35	Sorting an array of structures .....	45
Sample 2.36	A scoreboard with array methods .....	45
Sample 2.37	User-defined type-macro in Verilog.....	49
Sample 2.38	User-defined type in SystemVerilog.....	49
Sample 2.39	Definition of <code>uint</code> .....	49
Sample 2.40	User-defined array type .....	50
Sample 2.41	User-defined associative array index .....	50
Sample 2.42	Creating a single pixel type .....	50
Sample 2.43	The pixel <code>struct</code> .....	51
Sample 2.44	Initializing a <code>struct</code> .....	51
Sample 2.45	Using <code>typedef</code> to create a union .....	52
Sample 2.46	Packed structure.....	52
Sample 2.47	Package for ABC bus.....	53
Sample 2.48	Importing packages .....	53
Sample 2.49	Importing selected symbols from a package .....	54
Sample 2.50	Converting between <code>int</code> and <code>real</code> with static cast.....	55
Sample 2.51	Basic streaming operator .....	55
Sample 2.52	Converting between queues with streaming operator .....	56
Sample 2.53	Converting between a structure and an array with streaming operators .....	57
Sample 2.54	A simple enumerated type, not recommended .....	57
Sample 2.55	Enumerated types, recommended style .....	58
Sample 2.56	Specifying enumerated values .....	58
Sample 2.57	Incorrectly specifying enumerated values .....	59
Sample 2.58	Correctly specifying enumerated values .....	59
Sample 2.59	Stepping through all enumerated members .....	60
Sample 2.60	Assignments between integers and enumerated types.....	60
Sample 2.61	Declaring a <code>const</code> variable .....	61
Sample 2.62	String methods.....	62
Sample 2.63	Expression width depends on context .....	63
Sample 3.1	New procedural statements and operators .....	70
Sample 3.2	Using <code>break</code> and <code>continue</code> while reading a file.....	70
Sample 3.3	Case-inside statement with ranges.....	71
Sample 3.4	Void function for debug.....	71
Sample 3.5	Ignoring a function's return value .....	71
Sample 3.6	Simple task without <code>begin...end</code> .....	72



Sample 3.7	Verilog-1995 routine arguments .....	72
Sample 3.8	C-style routine arguments.....	73
Sample 3.9	Verbose Verilog-style routine arguments.....	73
Sample 3.10	Routine arguments with sticky types.....	73
Sample 3.11	Passing arrays using <code>ref</code> and <code>const</code> .....	74
Sample 3.12	Using <code>ref</code> across threads.....	75
Sample 3.13	Function with default argument values .....	76
Sample 3.14	Using default argument values .....	76
Sample 3.15	Binding arguments by name.....	77
Sample 3.16	Original task header.....	77
Sample 3.17	Task header with additional array argument.....	77
Sample 3.18	Task header with additional array argument.....	77
Sample 3.19	Return in a task .....	78
Sample 3.20	Return in a function .....	78
Sample 3.21	Returning an array from a function with a typedef .....	79
Sample 3.22	Passing an array to a function as a <code>ref</code> argument .....	79
Sample 3.23	Specifying automatic storage in program blocks .....	80
Sample 3.24	Static initialization bug.....	81
Sample 3.25	Static initialization fix: use <code>automatic</code> .....	81
Sample 3.26	Static initialization fix: break apart declaration and initialization .....	81
Sample 3.27	Time literals and <code>\$timeformat</code> .....	82
Sample 3.28	Time variables and rounding .....	83
Sample 4.1	Arbiter model using ports.....	89
Sample 4.2	Testbench module using ports .....	90
Sample 4.3	Top-level module with ports.....	90
Sample 4.4	Simple interface for arbiter.....	91
Sample 4.5	Arbiter using a simple interface .....	92
Sample 4.6	Testbench using a simple arbiter interface .....	92
Sample 4.7	Top module with a simple arbiter interface .....	92
Sample 4.8	Bad test module includes interface.....	93
Sample 4.9	Connecting an interface to a module that uses ports .....	93
Sample 4.10	Interface with modports.....	94
Sample 4.11	Arbiter model with interface using modports.....	94
Sample 4.12	Testbench with interface using modports .....	94
Sample 4.13	Top level module with modports .....	95
Sample 4.14	Arbiter monitor with interface using modports .....	96
Sample 4.15	Driving logic and wires in an interface .....	97
Sample 4.16	Interface with a clocking block .....	99
Sample 4.17	Race condition between testbench and design .....	101
Sample 4.18	Testbench using interface with clocking block.....	103
Sample 4.19	Signal synchronization .....	105
Sample 4.20	Synchronous interface sample and drive from module .....	105
Sample 4.21	Testbench using interface with clocking block.....	106

Sample 4.22	Interface signal drive .....	107
Sample 4.23	Driving a synchronous interface.....	107
Sample 4.24	Interface signal drive .....	108
Sample 4.25	Bidirectional signals in a program and interface.....	109
Sample 4.26	Clocking block with default statement.....	109
Sample 4.27	Clocking block with delays on individual signals.....	110
Sample 4.28	A <i>final</i> block.....	110
Sample 4.29	Bad clock generator in program block .....	111
Sample 4.30	Good clock generator in module .....	112
Sample 4.31	Top module with implicit port connections.....	112
Sample 4.32	Module with just port connections .....	113
Sample 4.33	Module with an interface.....	113
Sample 4.34	Top module connecting DUT and interface.....	113
Sample 4.35	Top-level scope for arbiter design .....	114
Sample 4.36	Cross-module references with <i>\$root</i> .....	115
Sample 4.37	Checking a signal with an <i>if</i> -statement.....	116
Sample 4.38	Simple immediate assertion.....	116
Sample 4.39	Error from failed immediate assertion.....	116
Sample 4.40	Creating a custom error message in an immediate assertion .....	117
Sample 4.41	Error from failed immediate assertion.....	117
Sample 4.42	Creating a custom error message.....	118
Sample 4.43	Concurrent assertion to check for X/Z .....	118
Sample 4.44	ATM router model header with ports .....	120
Sample 4.45	Top-level module without an interface.....	121
Sample 4.46	Verilog-1995 testbench using ports .....	122
Sample 4.47	Rx interface with modports and clocking block.....	124
Sample 4.48	Tx interface with modports and clocking block.....	124
Sample 4.49	ATM router model with interface using modports .....	125
Sample 4.50	Top-level module with interface.....	125
Sample 4.51	Testbench using an interface with a clocking block.....	126
Sample 4.52	Ref ports .....	127
Sample 5.1	Simple transaction class .....	133
Sample 5.2	Class in a package .....	134
Sample 5.3	Importing a package in a program.....	134
Sample 5.4	Declaring and using a handle .....	136
Sample 5.5	Simple user-defined <i>new()</i> function .....	136
Sample 5.6	A <i>new()</i> function with arguments .....	137
Sample 5.7	Calling the right <i>new()</i> function .....	137
Sample 5.8	Allocating multiple objects.....	138
Sample 5.9	Creating multiple objects.....	139
Sample 5.10	Using variables and routines in an object.....	140
Sample 5.11	Routines in the class .....	141
Sample 5.12	Out-of-block method declarations .....	142

Sample 5.13	Out-of-body method missing class name .....	143
Sample 5.14	Class with a static variable .....	144
Sample 5.15	The class scope resolution operator.....	145
Sample 5.16	Static storage for a handle .....	145
Sample 5.17	Static method displays static variable.....	146
Sample 5.18	Name scope .....	147
Sample 5.19	Class uses wrong variable .....	148
Sample 5.20	Move class into package to find bug .....	148
Sample 5.21	Using <code>this</code> to refer to class variable.....	149
Sample 5.22	<code>Statistics</code> class declaration .....	150
Sample 5.23	Encapsulating the <code>Statistics</code> class .....	150
Sample 5.24	Using a <code>typedef</code> class statement .....	152
Sample 5.25	Passing objects.....	153
Sample 5.26	Bad transaction creator task, missing <code>ref</code> on handle.....	154
Sample 5.27	Good transaction creator task with <code>ref</code> on handle .....	154
Sample 5.28	Bad generator creates only one object.....	155
Sample 5.29	Good generator creates many objects.....	155
Sample 5.30	Using an array of handles .....	156
Sample 5.31	Copying a simple class with <code>new</code> .....	156
Sample 5.32	Copying a complex class with <code>new</code> operator.....	157
Sample 5.33	Simple class with <code>copy</code> function.....	158
Sample 5.34	Using a <code>copy</code> function.....	159
Sample 5.35	Complex class with deep copy function.....	159
Sample 5.36	<code>Statistics</code> class declaration .....	160
Sample 5.37	Copying a complex class with <code>new</code> operator.....	160
Sample 5.38	Transaction class with <code>pack</code> and <code>unpack</code> functions.....	161
Sample 5.39	Using the <code>pack</code> and <code>unpack</code> functions .....	162
Sample 5.40	Basic Transactor .....	164
Sample 6.1	Simple random class.....	173
Sample 6.2	Randomization check macro and example .....	174
Sample 6.3	Constraint without random variables.....	175
Sample 6.4	Constrained-random class .....	176
Sample 6.5	Bad ordering constraint .....	177
Sample 6.6	Result from incorrect ordering constraint.....	177
Sample 6.7	Constrain variables to be in a fixed order .....	177
Sample 6.8	Weighted random distribution with <code>dist</code> .....	178
Sample 6.9	Dynamically changing distribution weights.....	179
Sample 6.10	Random sets of values .....	179
Sample 6.11	Inverted random set constraint .....	180
Sample 6.12	Random set constraint for an array.....	180
Sample 6.13	Equivalent set of constraints.....	180
Sample 6.14	Choose any value except those in an array.....	180
Sample 6.15	Printing a histogram .....	181
Sample 6.16	Histogram for inside constraint .....	181

Sample 6.17	Class to choose from an array of possible values .....	181
Sample 6.18	Choosing from an array of values .....	182
Sample 6.19	Using randc to choose array values in random order .....	182
Sample 6.20	Testbench for randc choosing array values in random order .....	183
Sample 6.21	Bidirectional constraints .....	183
Sample 6.22	Constraint block with implication operator .....	184
Sample 6.23	Implication operator .....	185
Sample 6.24	Constraint block with if implication operator .....	185
Sample 6.25	Constraint block with if-else operator .....	185
Sample 6.26	Constraint block with multiple if-else operator .....	186
Sample 6.27	Equivalence constraint .....	186
Sample 6.28	Class <code>Unconstrained</code> .....	187
Sample 6.29	Class with implication constraint .....	188
Sample 6.30	Class with implication constraint and additional constraint .....	189
Sample 6.31	Class with implication and <code>solve...before</code> .....	189
Sample 6.32	Using <code>constraint_mode</code> .....	191
Sample 6.33	Checking write length with a valid constraint .....	192
Sample 6.34	The <code>randomize()</code> with statement .....	193
Sample 6.35	Building a bathtub distribution .....	194
Sample 6.36	<code>\$urandom</code> range usage .....	195
Sample 6.37	Constraint with a variable bound .....	196
Sample 6.38	<code>dist</code> constraint with variable weights .....	196
Sample 6.39	<code>rand_mode</code> disables randomization of variables .....	197
Sample 6.40	Randomizing a subset of variables in a class .....	198
Sample 6.41	Class with an external constraint .....	199
Sample 6.42	Program defining an external constraint .....	200
Sample 6.43	Signed variables cause randomization problems .....	201
Sample 6.44	Randomizing unsigned 32-bit variables .....	201
Sample 6.45	Randomizing unsigned 8-bit variables .....	201
Sample 6.46	Expensive constraint with <code>mod</code> and unsized variable .....	202
Sample 6.47	Efficient constraint with <code>bit extract</code> .....	202
Sample 6.48	Constraining dynamic array size .....	203
Sample 6.49	Random strobe pattern class .....	204
Sample 6.50	First attempt at sum constraint: <code>bad_sum1</code> .....	205
Sample 6.51	Program to try constraint with array sum .....	205
Sample 6.52	Output from <code>bad_sum1</code> .....	205
Sample 6.53	Second attempt at sum constraint: <code>bad_sum2</code> .....	206
Sample 6.54	Output from <code>bad_sum2</code> .....	206
Sample 6.55	Third attempt at sum constraint: <code>bad_sum3</code> .....	206
Sample 6.56	Output from <code>bad_sum3</code> .....	206
Sample 6.57	Fourth attempt at sum constraint: <code>bad_sum4</code> .....	207
Sample 6.58	Output from <code>bad_sum4</code> .....	207
Sample 6.59	Simple <code>foreach</code> constraint: <code>good_sum5</code> .....	207

Sample 6.60	Output from <code>good_sum5</code> .....	207
Sample 6.61	Creating ascending array values with <code>foreach</code> .....	208
Sample 6.62	Creating unique array values with <code>foreach</code> .....	208
Sample 6.63	Creating unique array values with a <code>randc</code> helper class .....	209
Sample 6.64	Unique value generator.....	209
Sample 6.65	Class to generate a random array of unique values .....	210
Sample 6.66	Using the <code>UniqueArray</code> class .....	210
Sample 6.67	Constructing elements in a random array class .....	211
Sample 6.68	Simple random sequence with ascending values.....	213
Sample 6.69	Command generator using <code>randsequence</code> .....	214
Sample 6.70	Random control with <code>randcase</code> and <code>\$urandom_range</code> .....	215
Sample 6.71	Equivalent constrained class.....	215
Sample 6.72	Creating a decision tree with <code>randcase</code> .....	216
Sample 6.73	Simple pseudorandom number generator .....	217
Sample 6.74	Test code before modification.....	219
Sample 6.75	Test code after modification .....	220
Sample 6.76	Ethernet switch configuration class .....	221
Sample 6.77	Building environment with random configuration .....	221
Sample 6.78	Simple test using random configuration .....	222
Sample 6.79	Simple test that overrides random configuration.....	223
Sample 7.1	Interaction of <code>begin...end</code> and <code>fork...join</code> .....	231
Sample 7.2	Output from <code>begin...end</code> and <code>fork...join</code> .....	232
Sample 7.3	<code>Fork...join_none</code> code .....	232
Sample 7.4	<code>Fork...join_none</code> output.....	233
Sample 7.5	<code>Fork...join_any</code> code.....	233
Sample 7.6	Output from <code>fork...join_any</code> .....	234
Sample 7.7	Generator / Driver class with a run task .....	234
Sample 7.8	Dynamic thread creation.....	235
Sample 7.9	Bad <code>fork...join_none</code> inside a loop .....	236
Sample 7.10	Execution of bad <code>fork...join_none</code> inside a loop .....	237
Sample 7.11	Automatic variables in a <code>fork...join_none</code> .....	237
Sample 7.12	Steps in executing automatic variable code.....	238
Sample 7.13	Automatic variables in a <code>fork...join_none</code> .....	238
Sample 7.14	Using <code>wait fork</code> to wait for child threads.....	239
Sample 7.15	Bug using shared program variable .....	240
Sample 7.16	Disabling a thread.....	241
Sample 7.17	Limiting the scope of a disable fork .....	242
Sample 7.18	Using <code>disable</code> label to stop threads .....	243
Sample 7.19	Using <code>disable</code> label to stop a task.....	243
Sample 7.20	Blocking on an event in Verilog .....	245
Sample 7.21	Output from blocking on an event.....	245
Sample 7.22	Waiting for an event .....	246
Sample 7.23	Output from waiting for an event .....	246
Sample 7.24	Waiting on event causes a zero delay loop.....	247

Sample 7.25	Waiting for an edge on an event .....	247
Sample 7.26	Passing an event into a constructor .....	248
Sample 7.27	Waiting for multiple threads with wait fork .....	249
Sample 7.28	Waiting for multiple threads by counting triggers.....	249
Sample 7.29	Waiting for multiple threads using a thread count.....	250
Sample 7.30	Semaphores controlling access to hardware resource .....	251
Sample 7.31	Mailbox declarations .....	253
Sample 7.32	Bad generator creates only one object.....	253
Sample 7.33	Good generator creates many objects.....	254
Sample 7.34	Good driver receives transactions from mailbox.....	254
Sample 7.35	Exchanging objects using a mailbox: the Generator class .....	255
Sample 7.36	Bounded mailbox.....	256
Sample 7.37	Output from bounded mailbox .....	257
Sample 7.38	Producer–consumer without synchronization .....	258
Sample 7.39	Producer–consumer without synchronization output .....	259
Sample 7.40	Producer–consumer synchronized with bounded mailbox.....	260
Sample 7.41	Output from producer–consumer with bounded mailbox .....	260
Sample 7.42	Producer–consumer synchronized with an event .....	261
Sample 7.43	Output from producer–consumer with event.....	262
Sample 7.44	Producer–consumer synchronized with a mailbox.....	263
Sample 7.45	Output from producer–consumer with mailbox .....	264
Sample 7.46	Basic Transactor .....	265
Sample 7.47	Configuration class .....	266
Sample 7.48	Environment class .....	266
Sample 7.49	Basic test program .....	268
Sample 8.1	Base Transaction class.....	275
Sample 8.2	Extended Transaction class.....	276
Sample 8.3	Constructor with arguments in an extended class .....	277
Sample 8.4	Driver class.....	278
Sample 8.5	Bad generator class.....	279
Sample 8.6	Generator class using blueprint pattern .....	281
Sample 8.7	Environment class .....	282
Sample 8.8	Simple test program using environment defaults .....	282
Sample 8.9	Injecting an extended transaction into testbench.....	283
Sample 8.10	Adding a constraint with inheritance.....	284
Sample 8.11	Base and extended class .....	285
Sample 8.12	Copying extended handle to base handle .....	285
Sample 8.13	Copying a base handle to an extended handle.....	286
Sample 8.14	Using \$cast to copy handles .....	286
Sample 8.15	Transaction and BadTr classes.....	287
Sample 8.16	Calling class methods .....	287
Sample 8.17	Building an Ethernet frame with composition.....	290
Sample 8.18	Building an Ethernet frame with inheritance.....	291
Sample 8.19	Building a flat Ethernet frame .....	292

Sample 8.20	Base transaction class with a virtual <code>copy</code> function.....	293
Sample 8.21	Extended transaction class with virtual <code>copy</code> method.....	293
Sample 8.22	Base transaction class with <code>copy</code> function.....	294
Sample 8.23	Extended transaction class with new <code>copy</code> function.....	294
Sample 8.24	Abstract class with pure virtual methods.....	295
Sample 8.25	Transaction class extends abstract class .....	296
Sample 8.26	Base callback class .....	298
Sample 8.27	Driver class with callbacks.....	298
Sample 8.28	Test using a callback for error injection .....	299
Sample 8.29	Simple scoreboard for atomic transactions.....	300
Sample 8.30	Test using callback for scoreboard .....	301
Sample 8.31	Stack using the <code>int</code> type.....	302
Sample 8.32	Parameterized class for a stack.....	303
Sample 8.33	Creating the parameterized stack class.....	303
Sample 8.34	Parameterized generator class using blueprint pattern .....	304
Sample 8.35	Simple testbench using parameterized generator class .....	304
Sample 8.36	Common base class for parameterized generator class .....	305
Sample 8.37	Dynamic print class with static variables .....	306
Sample 8.38	Transactor class with dynamic print object .....	307
Sample 8.39	Static print class.....	307
Sample 8.40	Transactor class with static print class.....	308
Sample 8.41	Configuration database with global methods .....	309
Sample 8.42	Configuration database with parameterized class.....	309
Sample 8.43	Configuration database with static parameterized class .....	310
Sample 8.44	Testbench for configuration database .....	310
Sample 8.45	Base test class .....	311
Sample 8.46	Test registry class.....	312
Sample 8.47	Simple test in a class.....	312
Sample 8.48	Program block for test classes .....	313
Sample 8.49	Test class that puts a bad transaction in the generator.....	313
Sample 8.50	Common SVM base class.....	314
Sample 8.51	Component class.....	315
Sample 8.52	Common base class for proxy class.....	315
Sample 8.53	Parameterized proxy class .....	316
Sample 8.54	Factory class .....	317
Sample 8.55	Base test class and registration macro .....	318
Sample 8.56	Test program.....	318
Sample 8.57	UVM factory build example.....	319
Sample 9.1	Incomplete D-flip flop model missing a path .....	326
Sample 9.2	Functional coverage of a simple object.....	331
Sample 9.3	Coverage report for a simple object .....	332
Sample 9.4	Coverage report for a simple object, 100% coverage.....	332
Sample 9.5	Functional coverage inside a class .....	334
Sample 9.6	Test using functional coverage callback.....	336



Sample 9.7	Callback for functional coverage.....	336
Sample 9.8	Defining an argument list to the sample method.....	337
Sample 9.9	Cover group with a trigger .....	337
Sample 9.10	Module with SystemVerilog Assertion.....	338
Sample 9.11	Triggering a cover group with an SVA.....	338
Sample 9.12	Using <code>auto_bin_max</code> set to 2 .....	339
Sample 9.13	Report with <code>auto_bin_max</code> set to 2.....	340
Sample 9.14	Using <code>auto_bin_max</code> for all cover points .....	340
Sample 9.15	Using an expression in a cover point.....	340
Sample 9.16	Defining bins for transaction length .....	341
Sample 9.17	Coverage report for transaction length .....	342
Sample 9.18	Specifying bin names .....	343
Sample 9.19	Report showing bin names .....	343
Sample 9.20	Specifying ranges with \$ .....	344
Sample 9.21	Conditional coverage — disable during reset .....	344
Sample 9.22	Using <code>stop</code> and <code>start</code> functions .....	344
Sample 9.23	Functional coverage for an enumerated type.....	345
Sample 9.24	Coverage report with enumerated types .....	345
Sample 9.25	Specifying transitions for a cover point.....	345
Sample 9.26	Wildcard bins for a cover point .....	346
Sample 9.27	Cover point with <code>ignore_bins</code> .....	346
Sample 9.28	Cover point with <code>auto bin max</code> and <code>ignore bins</code> .....	347
Sample 9.29	Cover point with <code>illegal_bins</code> .....	347
Sample 9.30	Basic cross coverage.....	348
Sample 9.31	Coverage summary report for basic cross coverage.....	349
Sample 9.32	Specifying cross coverage bin names.....	350
Sample 9.33	Cross coverage report with labeled bins.....	350
Sample 9.34	Excluding bins from cross coverage.....	351
Sample 9.35	Specifying cross coverage weight .....	352
Sample 9.36	Cross coverage with bin names .....	353
Sample 9.37	Cross coverage with <code>binsof</code> .....	353
Sample 9.38	Mimicking cross coverage with concatenation .....	354
Sample 9.39	Covergroup with simple argument .....	354
Sample 9.40	Pass-by-reference .....	355
Sample 9.41	Specifying per-instance coverage.....	356
Sample 9.42	Specifying comments for a cover group.....	356
Sample 9.43	Specifying comments for a cover group instance.....	356
Sample 9.44	Report all bins including empty ones .....	357
Sample 9.45	Specifying the coverage goal.....	358
Sample 9.46	Original class for packet length.....	358
Sample 9.47	<code>solve...before</code> constraint for packet length .....	359
Sample 10.1	Rx interface with clocking block.....	364
Sample 10.2	Tx interface with clocking block.....	364
Sample 10.3	Testbench using physical interfaces .....	365



Sample 10.4	Top level module with array of interfaces.....	366
Sample 10.5	Testbench using virtual interfaces .....	367
Sample 10.6	Testbench using virtual interfaces .....	367
Sample 10.7	Monitor class using virtual interfaces .....	368
Sample 10.8	Test harness using an interface in the port list.....	369
Sample 10.9	Test with an interface in the port list .....	370
Sample 10.10	Top module with a second interface in the test's port list .....	370
Sample 10.11	Test with two interfaces in the port list .....	370
Sample 10.12	Test with virtual interface and XMR.....	370
Sample 10.13	Test harness without interfaces in the port list .....	371
Sample 10.14	Test harness with a second interface .....	371
Sample 10.15	Test with two virtual interfaces and XMRs.....	371
Sample 10.16	Interface for 8-bit counter.....	372
Sample 10.17	Counter model using <code>X_if</code> interface.....	373
Sample 10.18	Top-level module with an array of virtual interfaces .....	373
Sample 10.19	Counter testbench using virtual interfaces .....	374
Sample 10.20	<code>Driver</code> class using virtual interfaces .....	375
Sample 10.21	Interface with a typedef.....	375
Sample 10.22	Testbench using a typedef for virtual interfaces.....	376
Sample 10.23	<code>Driver</code> using a typedef for virtual interfaces.....	376
Sample 10.24	Testbench using an array of virtual interfaces.....	376
Sample 10.25	Testbench passing virtual interfaces with a port .....	377
Sample 10.26	Parameterized counter model using <code>X_if</code> interface.....	378
Sample 10.27	Parameterized interface for 8-bit counter.....	378
Sample 10.28	Parameterized top-level module with an array of virtual interfaces.....	378
Sample 10.29	Parameterized counter testbench using virtual interfaces .....	379
Sample 10.30	<code>Driver</code> class using virtual interfaces .....	379
Sample 10.31	Interface with tasks for parallel protocol.....	380
Sample 10.32	Interface with tasks for serial protocol .....	381
Sample 11.1	Top level module .....	387
Sample 11.2	Testbench program .....	388
Sample 11.3	CPU Management Interface .....	388
Sample 11.4	Utopia interface .....	389
Sample 11.5	Environment class header.....	390
Sample 11.6	Environment class methods.....	391
Sample 11.7	Callback class connects driver and scoreboard .....	394
Sample 11.8	Callback class connects monitor and scoreboard .....	394
Sample 11.9	Callback class connects the monitor and coverage .....	395
Sample 11.10	Environment configuration class .....	396
Sample 11.11	Cell configuration type .....	396
Sample 11.12	Configuration class methods .....	397
Sample 11.13	UNI cell format .....	397
Sample 11.14	NNI cell format .....	397

Sample 11.15	ATMCellType.....	398
Sample 11.16	UNI_cell definition.....	398
Sample 11.17	UNI_cell methods.....	399
Sample 11.18	UNI_generator class.....	402
Sample 11.19	driver class.....	402
Sample 11.20	Driver callback class.....	405
Sample 11.21	Monitor callback class.....	405
Sample 11.22	The Monitor class.....	405
Sample 11.23	The Scoreboard class.....	407
Sample 11.24	Functional coverage class.....	409
Sample 11.25	The CPU_driver class.....	410
Sample 11.26	Test with one cell.....	412
Sample 11.27	Test that drops cells using driver callback.....	413
Sample 12.1	SystemVerilog code calling C factorial routine.....	416
Sample 12.2	C factorial function.....	416
Sample 12.3	Changing the name of an imported function.....	417
Sample 12.4	Argument directions.....	418
Sample 12.5	C factorial routine with const argument.....	418
Sample 12.6	Importing a C math function.....	419
Sample 12.7	Counter routine using a static variable.....	420
Sample 12.8	Testbench for an 7-bit counter with static storage.....	421
Sample 12.9	Counter routine using instance storage.....	422
Sample 12.10	Testbench for an 7-bit counter with per-instance storage.....	423
Sample 12.11	Testbench for counter that checks for Z or X values.....	425
Sample 12.12	Counter routine that checks for Z and X values.....	426
Sample 12.13	Counter class.....	427
Sample 12.14	Static methods and linkage.....	428
Sample 12.15	C++ counter communicating with methods.....	429
Sample 12.16	Static wrapper for C++ transaction level counter.....	430
Sample 12.17	Testbench for C++ model using methods.....	431
Sample 12.18	C routine to compute Fibonacci series.....	432
Sample 12.19	Testbench for Fibonacci routine.....	433
Sample 12.20	C routine to compute Fibonacci series with 4-state array.....	433
Sample 12.21	Testbench for Fibonacci routine with 4-state array.....	433
Sample 12.22	Testbench code calling a C routine with an open array.....	434
Sample 12.23	C code using a basic open array.....	434
Sample 12.24	Testbench calling C code with multi-dimensional open array.....	436
Sample 12.25	C code with multi-dimensional open array.....	436
Sample 12.26	Testbench for packed open arrays.....	437
Sample 12.27	C code using packed open arrays.....	437
Sample 12.28	C code to share a structure.....	438
Sample 12.29	Testbench for sharing structure.....	439
Sample 12.30	Returning a string from C.....	440

Sample 12.31	Returning a string from a heap in C .....	440
Sample 12.32	Importing a pure function.....	441
Sample 12.33	Imported context tasks.....	441
Sample 12.34	Exporting a SystemVerilog function .....	442
Sample 12.35	Calling an exported SystemVerilog function from C .....	442
Sample 12.36	Output from simple export .....	442
Sample 12.37	C code to read simple command file and call exported function.....	443
Sample 12.38	SystemVerilog module for simple memory model.....	443
Sample 12.39	Command file for simple memory model .....	444
Sample 12.40	SystemVerilog module for memory model with exported tasks.....	444
Sample 12.41	C code to read command file and call exported function.....	445
Sample 12.42	Command file for simple memory model .....	446
Sample 12.43	Command file for exported methods with OOP memories .....	446
Sample 12.44	SystemVerilog module with memory model class .....	447
Sample 12.45	C code to call exported tasks with OOP memory.....	448
Sample 12.46	Second module for simple export example .....	449
Sample 12.47	Output from simple example with two modules .....	449
Sample 12.48	C code getting and setting context .....	450
Sample 12.49	Modules calling methods that get and set context.....	451
Sample 12.50	Output from svSetScope code .....	451
Sample 12.51	SystemVerilog code calling C wrapper for Perl .....	452
Sample 12.52	C wrapper for Perl script .....	452
Sample 12.53	Perl script called from C and SystemVerilog .....	452
Sample 12.54	VCS command line to run Perl script.....	453

