SystemVerilog for Verification

Chris Spear · Greg Tumbush

SystemVerilog for Verification

A Guide to Learning the Testbench Language Features

Third Edition

Springer

Chris Spear
Synopsys, 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

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. 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.

Contents

List of Figures

List of Tables

List of Samples