Preface

What is this book about?

This book is the first one you should 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, which help show why you want 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 (LRM) but found it loaded with syntax but no guidelines on which construct to choose.

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

Before reading this book, you should be comfortable with Verilog-1995. Knowledge of Verilog-2001, SystemVerilog design constructs, or SystemVerilog Assertions is not required.

What is new in the second edition?

This new edition of SystemVerilog for Verification has many improvements over the first edition that was published in 2006.

- The anticipated 2008 version of the SystemVerilog Language Reference Manual (LRM) has many changes, both large and small. This book tries to include the latest relevant information.
- Many readers asked me for more details on SystemVerilog concepts. Almost all of these conversations have been incorporated into this book as expanded explanations and code samples. Starting with Chap. 2, nearly every paragraph and example has been rewritten, revised, or just tweaked. There are over 50 new pages in the original ten chapters, and over 70 new examples. In all, the new edition is almost 1/3 larger than the original.
- You asked for more examples, especially large ones. This edition has a directed testbench at the end of Chap. 4, and complete constrained random testbench in Chap. 11.
- Not all testbench code is written in SystemVerilog, and so I added Chap. 12 to show how to connect C and C++ code to SystemVerilog with the Direct Programming Interface.
- Most engineers read a book starting with the index, and so I doubled the number of entries. We also love cross references, and so I have added more so that you can read the book nonlinearly.
- Lastly, a big thanks to all the readers who spotted mistakes in the first edition, from poor grammar to code that was obviously written on the morning after a 18-hour flight from Asia to Boston. This edition has been checked and reviewed many times over, but once again, all mistakes are mine.

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 (HVL) 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 P1800-2005 for SystemVerilog, IEEE (2005).

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 language 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 specialized APIs. 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 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.

Comparing SystemVerilog and SystemC for high-level design

Now that SystemVerilog incorporates Object-Oriented Programming, dynamic threads, and interprocess communication, it can be used for system design. When talking about the applications for SystemVerilog, the IEEE standard mentions architectural modeling before design, assertions, and test. SystemC can also be used for architectural modeling.

There are several major differences between SystemC and SystemVerilog:

- SystemVerilog provides one modeling language. You do not have to learn C++ and the Standard Template Library to create your models.
- SystemVerilog simplifies top–down design. You can create your system models in SystemVerilog and then refine each block to the next lower level. The original system-level models can be reused as reference models.
- Software developers want a free or low-cost hardware simulator that is fast. You can create high-performance transaction-level models in both SystemC and SystemVerilog. SystemVerilog simulators require a license that a software developer may not want to pay for. SystemC can be free, but only if all your models are available in SystemC.

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 variables.

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 that 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 Chap. 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 the risk of introducing new bugs.

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

Icons used in this book

Table 1. Book icons

	Shows verification methodology to guide your usage of SystemVerilog testbench features
	Shows common coding mistakes

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. All of the examples have been verified with Synopsys' Chronologic VCS 2005.06, 2006.06, and 2008.03. The SystemVerilog Language Reference Manual covers hundreds of new features. I have concentrated on constructs useful for verification and implemented in VCS. It is better to have verified examples than to show all language features and thus risk having incorrect code. Speaking of mistakes, if you think you have found a mistake, please check my web site for the Errata page. If you are the first to find any mistake in a chapter, I will send you a free, autographed book.

CHRIS SPEAR
Synopsys, Inc.
chris@spear.net

http://www.springer.com/978-0-387-76529-7

SystemVerilog for Verification
A Guide to Learning the Testbench Language Features
Spear, C.
2008, XXXVI, 429 p. 5 illus., Hardcover
ISBN: 978-0-387-76529-7