Verilog and SystemVerilog Gotchas

101 Common Coding Errors and How to Avoid Them

Stuart Sutherland
Don Mills

Verilog and SystemVerilog Gotchas

101 Common Coding Errors and How to Avoid Them

Springer

Stuart Sutherland
Sutherland HDL, Inc.
Tualatin, OR
USA

Don Mills
LCDM Engineering
Chandler, AZ
USA

Library of Congress Control Number: 2007926706

ISBN 978-0-387-71714-2 e-ISBN 978-0-387-71715-9

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

Dedication

To my wonderful wife, LeeAnn, and my children, Ammon, Tamara, Hannah, Seth and Samuel — thank you for your patience during the many long hours and late nights you tolerated while this book was being written.

Stu Sutherland
Portland, Oregon

To my wife and sweetheart Geri Jean, and my children, Sara, Kirsten, Adam, Alex, Dillan, Donnelle, Grant and Gina — thanks to each of you for the patience you have had with me as I have dealt with debugging many of these gotchas on designs over the years.

Don Mills
Chandler, Arizona

About the Authors

Mr. Stuart Sutherland is a member of the IEEE 1800 working group that oversees both the Verilog and SystemVerilog standards. He has been involved with the definition of the Verilog standard since its inception in 1993, and the SystemVerilog standard since work began in 2001. In addition, Stuart is the technical editor of the official IEEE Verilog and SystemVerilog Language Reference Manuals (LRMs). Stuart is an independent Verilog consultant, specializing in providing comprehensive expert training on the Verilog HDL, SystemVerilog and PLI. Stuart is a co-author of the books "*SystemVerilog for Design*", "*Verilog-2001: A Guide to the New Features in the Verilog Hardware Description Language*" and is the author of "*The Verilog PLI Handbook*", as well as the popular "*Verilog HDL Quick Reference Guide*" and "*Verilog PLI Quick Reference Guide*". He has also authored a number of technical papers on Verilog and SystemVerilog, which are available at www.sutherland-hdl.com/papers. You can contact Stuart at stuart@sutherland-hdl.com.

visit the author's web page at www.sutherland-hdl.com

Mr. Don Mills has been involved in ASIC design since 1986. During that time, he has worked on more than 30 ASIC projects. Don started using top-down design methodology in 1991 (Synopsys Design Compiler 1.2). Don has developed and implemented top-down ASIC design flows at several companies. His specialty is integrating tools and automating the flow. Don works for Microchip Technology Inc. as an internal SystemVerilog and Verilog consultant. Don is a member of the IEEE Verilog and System Verilog committees that are working on language issues and enhancements. Don has authored and co-authored numerous papers, such as "SystemVerilog Assertions are for Design Engineers Too!" and "RTL Coding Styles that Yield Simulation and Synthesis Mismatches". Copies of these papers can be found at www.lcdm-eng.com. Mr. Mills can be reached at mills@lcdm-eng.com or don.mills@microchip.com.

visit the author's web page at www.lcdm-eng.com

Acknowledgments

The authors express their sincere appreciation to the contributions of several Verilog and SystemVerilog experts.

Chris Spear of Synopsys, Inc. suggested several of the verification related gotchas, provided the general descriptions of these gotchas, and ran countless tests for us.

Shalom Bresticker of Intel also suggested several gotchas.

Jonathan Bromley of Doulos, Ltd., ***Clifford Cummings*** of Sunburst Design, ***Tom Fitzpatrick*** of Mentor Graphics, ***Steve Golson*** of Trilobyte Systems, ***Gregg Lahti*** of Microchip Technology, Inc. and ***Chris Spear*** of Synopsys, Inc. provided thorough technical reviews of this book, and offered invaluable comments on how to improve the gotcha descriptions.

Steve Golson of Trilobyte Systems provided a wonderful foreword to this book

Lastly, we acknowledge and express our gratitude to our wives, ***LeeAnn Sutherland*** and ***Geri Jean Mills***, for meticulously reviewing this book for grammar and punctuation. If any such errata remain in the book, it could only be due to changes we made after their reviews.

Table of Contents

List of Gotchas

List of Gotchas

Foreword by Steve Golson

Some people collect baseball cards, old car magazines, or maybe rubber duckies.

I collect Verilog books.

It started back in 1989 with a looseleaf copy of "Gateway VERILOG-XL Reference Manual Version 1.5a" in a three-ring binder. Verilog was a bit simpler back then—it's hard to believe we actually designed chips using only one type of procedural assignment (nonblocking assigns were not part of the language yet). And we ran our simulations on a VAX, or maybe a fancy Apollo workstation.

Since then I've bought pretty much every Verilog book that came along. I've got a few synthesis books, and I'll pick up an occasional VHDL reference or maybe a text on the history of hardware description languages, but mostly it's Verilog. Dozens and dozens of books about Verilog.

There's a funny thing about most of these books though. After I leaf through them a few times, they sit on the shelf. I admit that it looks pretty impressive once you have an entire bookcase filled with Verilog books, but the discerning visitor will notice how fresh and new they all are. Unused. Unread. Useless.

I'm often disappointed to find very little information which is useful for the practicing engineer. What I'm looking for is a book I can use every day, a book that will help me get my chip out the door, on time and working.

Stu and Don have written such a book. I've known these guys for many years, and they have probably forgotten more Verilog than I've ever known. They have distilled their collective knowledge into this helpful and extremely useful book. Read it and you won't be disappointed.

If you are an old hand at Verilog try to pick out all the Gotchas that you have found the hard way. Smile and say to yourself "Oh yeah, I remember getting caught by that one!"

Those of you who are new to Verilog and SystemVerilog, welcome aboard! Here's your chance to learn from two of the leading experts in the field. And if you ever have a chance to take a training class from either of these gentlemen, don't hesitate to sign up. I guarantee you won't regret it.

Oh by the way, my favorite Gotcha is "Gotcha 65: Infinite for loops". Why? Well, I built a chip with that bug in it. Believe me, when a modeling error causes you to have broken silicon, you never forget why it happened. Back then I didn't have this book to help me, but you do! Keep this book close at hand, refer to it often, and may all your models compile and all your loops terminate.

Steve Golson
Trilobyte Systems
http://www.trilobyte.com

Chapter 1

Introduction, What Is A Gotcha?

This chapter defines what a "*gotcha*" is, and why programming languages allow gotchas. For the curious, the chapter also provides a brief history of the Verilog and SystemVerilog standards. The topics presented in this chapter include:

- What are Verilog and SystemVerilog
- The definition of a gotcha
- A brief description of the Verilog and SystemVerilog standards

What are Verilog and SystemVerilog?

The terms "Verilog" and "SystemVerilog" are sometimes a source of confusion because the terms are not used consistently in the industry. For the purposes of this book, "Verilog" and SystemVerilog are used as follows:

Verilog is a Hardware Description Language (HDL). It is a specialized programming language used to model digital hardware designs and, to a limited extent, to write test programs to exercise these models.

SystemVerilog is a substantial set of extensions to the Verilog HDL. A primary goal of these extensions is to enable modeling and verifying larger designs with more compact code. By itself, SystemVerilog is not a complete language; it is just a set of additions to the base Verilog language.

What is a Gotcha?

A programming "gotcha" is a language feature, which, if misused, causes unexpected—and, in hardware design, potentially disastrous—behavior. The classic example in the C language is having an assignment within a conditional expression, such as:

```
if (day=15)       /* GOTCHA! assigns value of 15 to day, then */
  do_mid_month_payroll;  /* if day is non-zero, do a payroll */
```

Most likely, what the programmer intended to code is `if (a==b)` instead of `if (a=b)`. The results are very different! This classic C programming Gotcha is not a syntax error; the code is perfectly legal. However, the code probably does not produce the intended results. If the coding error is not detected before a product is shipped, a simple bug like this could lead to serious ramifications in a product.

Just like any programming language, Verilog, and the SystemVerilog extensions to Verilog, have gotchas. There are constructs in Verilog and SystemVerilog that can be used in ways that are syntactically correct, but yield unexpected or undesirable results. Some of the primary reasons Verilog and SystemVerilog have gotchas are:

- Inheritance of C and C++ gotchas

  Verilog and SystemVerilog leverage the general syntax and semantics of the C and C++ languages. Verilog and SystemVerilog inherit the strengths of these powerful programming languages, but they also inherit many of the gotchas of C and C++. (Which raises the question, can the common C coding error such as `if (day=15)` be made in Verilog/SystemVerilog? The answer can be found in Gotcha 44 on page 99.)

- Loosely typed operations

  Verilog and SystemVerilog are *loosely typed* languages. As such, operations can be performed on any data type, and underlying language rules take care of how operations should be performed. If a design or verification engineer does not understand these underlying language rules, then unexpected results can occur.

- Allowance to model good and bad designs

  An underlying philosophy of Verilog and SystemVerilog is that engineers should be allowed to model and prove both what works correctly in hardware, and what will not work in hardware. In order to legally model hardware that does not work, the language must also permit unintentional modeling errors when the intent is to model designs that work correctly.

The Verilog and SystemVerilog standards

Verilog is an international standard Hardware Description Language. The official standard is **IEEE Std 1364-2005 Verilog Language Reference Manual** (LRM), commonly referred to as ***"Verilog-2005"***. The Verilog standard defines a rich set of programming and modeling constructs specific to representing the behavior of digital logic. The Verilog Hardware Description Language was first created in 1984. Verilog was designed to meet the needs of engineering in the mid 1980s, when a typical design was under 50,000 gates and ICs were based on 3 micron technology. As digital design size and technologies changed, Verilog evolved to meet new design requirements. Verilog was first standardized by the IEEE in 1995 (IEEE Std 1364-1995). In 2001, The IEEE released the Verilog-2001 standard (IEEE Std 1364-2001) which enhanced Verilog in several ways, such as synthesizable signed arithmetic on any vector size and re-entrant tasks and functions. The IEEE updated the Verilog standard in 2005, but no major modeling enhancements were added in this version. Instead, all enhancements to Verilog were documented in a separate standard, SystemVerilog.

SystemVerilog is a standard set of extensions to the Verilog-2005 Standard. These extensions are documented in a separate standard, **IEEE Std 1800-2005 SystemVerilog Language Reference Manual**, commonly referred to as ***"SystemVerilog-2005"***. The SystemVerilog extensions enable writing synthesizable models that are continuously increasing in size and complexity, as well as verifying these multi-million gate designs. SystemVerilog adds to Verilog features from the SUPERLOG, VERA C, C++, and VHDL languages, along with OVA and PSL assertions. SystemVerilog was first developed by Accellera, a consortium of companies that do electronic design and companies that provide Electronic Design Automation (EDA) tools. Accellera released a preliminary version of the extensions to Verilog in 2002, called ***SystemVerilog 3.0*** (3.0 to show that SystemVerilog was the next generation of Verilog, where Verilog-1995 was the first generation and Verilog 2001 was the second generation). In 2003, Accellera released ***SystemVerilog 3.1*** and in 2004 ***SystemVerilog 3.1a***. This latter Accellera standard was then submitted to the IEEE for full standardization.

The original intent was for the IEEE to fold the Accellera SystemVerilog extensions into the Verilog standard. At the insistence of EDA companies, however, the IEEE made the decision to temporarily keep the SystemVerilog extensions in a separate document to make it easier for EDA companies to implement the extensive set of new features in their Verilog tools.