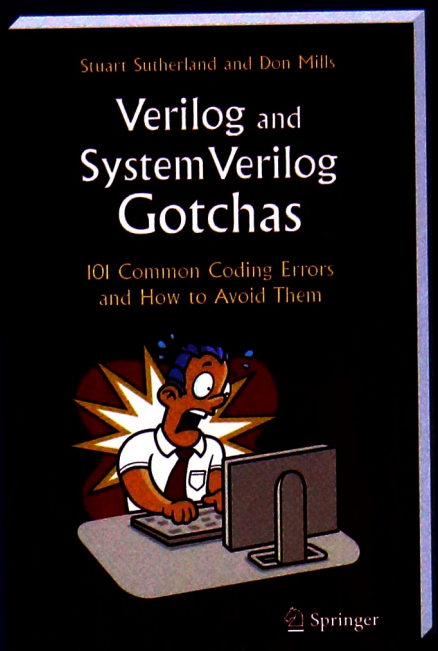
纠错式学习，从“陷阱”中学习编程，加深对语言本身的理解。
逆向式学习，从错误中学习避免错误的方法，让读者写出更好的代码。
案例式学习，将101个“陷阱”分类汇编，以针对性案例引导读者掌握编程要点。

HZ BOOKS 华章科技

Springe

电子与嵌入式系统设计译丛



Verilog and SystemVerilog Gotchas
101 Common CodingErrors and How to Avoid Them

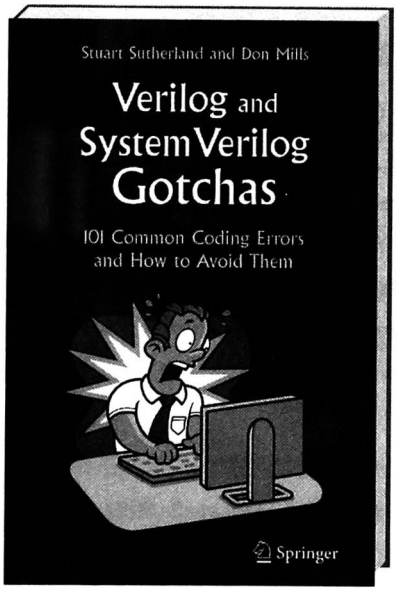
Verilog与SystemVerilog编程陷阱

如何避免101个常犯的编码错误

[美] 斯图尔特·萨瑟兰（Stuart Sutherland） 当·米尔斯（Don Mills） 著
戴成然 高镇 译

机械工业出版社
China Machine Press

电子与嵌入式系统
设计译丛



Verilog and SystemVerilog Gotchas
101 Common CodingErrors and How to Avoid Them

Verilog与SystemVerilog编程陷阱

如何避免101个常犯的编码错误

[美] 斯图尔特·萨瑟兰（Stuart Sutherland） 当·米尔斯（Don Mills） 著
戴成然 高镇 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Verilog 与 SystemVerilog 编程陷阱：如何避免 101 个常犯的编码错误 /（美）萨瑟兰（Sutherland，S.），（美）米尔斯（Mills，D.）著；戴成然，高镇译 . —北京：机械工业出版社，2015.5
（电子与嵌入式系统设计译丛）
书名原文：Verilog and SystemVerilog Gotchas：101 Common Coding Errors and How to Avoid Them

ISBN 978-7-111-50316-3

I. V… II. ① 萨… ② 米… ③ 戴… ④ 高… III. 硬件描述语言－程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2015）第 108222 号

本书版权登记号：图字：01-2013-7848

Verilog 与 SystemVerilog 编程陷阱
如何避免 101 个常犯的编码错误

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：张国强　陈佳媛　　责任校对：殷　虹
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2015 年 6 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：10.25
书　　号：ISBN 978-7-111-50316-3　　定　　价：55.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

译 者 序

随着电子设计自动化（Electronic Design Automation，EDA）技术的广泛应用，使用硬件语言完成现场可编程门阵列（Field-Programmable Gate Array，FPGA）是大势所趋。迄今为止，主要的硬件描述语言包括 VHDL、Verilog HDL 和 SystemVerilog。相比于后两者，VHDL 发展更早，语法更为严格，因而对初学者来说上手较慢，在实际使用中经常因为不规范的代码遭遇编译报错。相比而言，Verilog 及 SystemVerilog 具有语法灵活的优势，当然也会因这种灵活性在综合过程中产生程序员意料之外的结果。目前，相比于 VHDL，Verilog 和其高级版本的 SystemVerilog 应用更为广泛。这是因为在集成电路（Integrated Circuit，IC）设计领域，90% 以上的公司都采用 Verilog 进行 IC 设计。因此对集成电路设计（Application Specific Integrated Circuit, ASIC）人员来说，必须掌握 Verilog。另外，由于 Verilog 和 C 语言有很多类似的语法和关键字，鉴于 C 语言普及人群广泛，Verilog 更能使初学者突破语言障碍，快速入门。正如我们提到的，Verilog 由于其固有的灵活性，便于初学者入门。而也正因于此，过于灵活的语法经常会使工程师们陷入不当使用 Verilog 的陷阱，从而造成意料之外的错误。

本书是由 Verilog 和 SystemVerilog 方面的资深专家根据自身实际工程经验倾心撰写的一本实用的 Verilog 和 SystemVerilog 手册。本书给出了硬件工程师在使用 Verilog 和 SystemVerilog 时容易陷入的 101 个陷阱。

本书的翻译工作主要由戴成然和高镇完成，其中戴成然完成了前言和第 1 ～ 4 章的翻译，高镇完成了第 5 ～ 8 章的翻译。由于译者水平有限，文中难免出现不妥之处，敬请读者批评指正。

高 镇

2015 年 3 月

序

人的爱好各有不同，有人喜欢收集棒球卡片，有人喜欢收集老旧的汽车杂志，也有人喜欢收集橡皮鸭子，而我喜欢收集 Verilog 方面的书籍。

这要追溯到 1989 年一本用三孔夹夹着的活页《 VERILOG-XL Reference Manual Version 1.5a 》复印版。当时的 Verilog 相对简单——令人难以相信的是，现如今已经可以用一种类型的程序分配来设计芯片（在当时非阻塞式分配还不是一种语言）。不仅如此，当时我们只能在 VAX 或阿波罗工作站上仿真。

从那时起，我几乎买下了关于 Verilog 的每一本书。我所写过的书中已经包含了一些综合性的书。将来我将写一本 VHDL 的必备手册，又或者是一本介绍硬件描述性语言历史的书，当然书的内容大多是有关 Verilog 的。

对于这些书，有一件有趣的事儿。在我翻阅它们后，就会把它们放在架子上，很少再次翻阅。我承认当你看到我装满 Verilog 书的书架时，你会对此留下很深的印象。目光敏锐的客人会注意到我的书是崭新的。确实，我没用过它们，没读过它们，因为它们对我帮助很少。

令人失望的是，我很难找到一本对应用工程师实用的书。我想要的是一本每天带在身边，及时和有效地助我迈出第一步的书。

Stu 和 Don 编著了本书。我们认识很多年了，他们在 Verilog 上有很深的造诣。他们将毕生所学的精华总结成这本非常实用的书。本书一定不会让你失望。

如果你是一个通过惨痛教训而跳出 Verilog 陷阱的内行，微笑着对自己说：“太好了，我曾经在这出过问题。”

欢迎刚开始学习 Verilog 和 SystemVerilog 的朋友们！通过阅读本书，你可以拥有向这个领域的两个顶级专家学习的机会。如果你有机会参加这两位专家中任意一位的课程，一定要报名。我保证你不会后悔。

顺便提一下，我最喜欢的陷阱是“陷阱 65：死循环”。因为我曾经犯过这样的错误。相信我，如果你曾经因为建模错误致使打坏计算机，那么这辈子你都不会忘记错误的原因。可是你比我幸运，因为本书可以帮助你。将这本书放在手边，经常翻阅，那么你的模型都可以通过编译，循环都可以终止。

Steve Golson

关于作者

Stuart Sutherland 是 IEEE 1800 工作组的成员，该工作组负责起草 Verilog 和 SystemVerilog 标准。早在 1993 年也就是 Verilog 标准的诞生之际，他就已经涉足其标准的定义。同时他参与 SystemVerilog 标准也可追溯到 2001 年。此外，Stuart 是 IEEE 官方 Verilog 和 SystemVerilog 语言参考手册的技术编辑。Stuart 先生作为独立 Verilog 顾问，专注提供针对 Verilog HDL、SystemVerilog 和 PLI 的综合性专家训练。Stuart 是《 SystemVerilog for Design 》《 Verilog-2001 ： A Guide to the New Features in the Verilog Hardware Description Language 》的合著者，也是《 The Verilog PLI Handbook 》和颇受推崇的《 Verilog HDL Quick Reference Guide 》及《 Verilog PLI Quick Reference Guide 》的作者。Stuart 同时发表了诸多涉及 Verilog 和 SystemVerilog 的技术文章，这些文章可以在 www.sutherland-hdl.com/papers 找到。可以通过邮箱 stuart@sutherland-hdl.com 联系 Stuart。

作者的个人网站：www.sutherland-hdl.com。

Don Mills 从 1986 年开始涉足 ASIC 的设计。在此期间，他参与了超过 30 个 ASIC 项目。Don 从 1991 开始使用自顶向下的设计方法（综合设计编译器 1.2 ）。Don 在几个公司开发并实施了自顶向下的 ASIC 设计流程。他精通工具整合和流程自动化。Don 作为 SystemVerilog 和 Verilog 内部咨询师服务于美国微芯技术公司。Don 是 IEEE Verilog 和 SystemVerilog 委员会的成员，该委员会致力于 Verilog 和 SystemVerilog 语言的发布和完善。Don 是多篇文章的作者或合著者，例如《 SystemVerilog Assertions are for Design Engineers Too! 》及《 RTLCoding Styles that Yield Simulation and Synthesis Mismatches 》。这些文章可以在 www.lcdm-eng.com 网站上找到。Mills 先生的联系方式为 mills@lcdm-eng.como 或 don.mills@microchip.com。

作者的个人网站：www.lcdm-eng.com。

目　录

第 1 章

什么是“编程陷阱”

本章定义了“陷阱”以及为什么编程语言允许陷阱。为了打消读者的疑虑，本章也简要介绍了 Verilog 和 SystemVerilog 标准。本章的主题包括：

- 什么是 Verilog 和 SystemVerilog
- 陷阱的定义
- Verilog 和 SystemVerilog 标准简介

什么是 Verilog 和 SystemVerilog

术语“ Verilog”和“ SystemVerilog”有时是混淆的源头，因为工业界对这两个术语的使用也并不统一。本书为表述方便起见，“ Verilog”和“ SystemVerilog”将分别在下述情况中使用：

Verilog 是硬件描述语言（HDL）。它是专门用来建立数字硬件设计模型的编程语言，并在一定程度上可以编写测试程序用于测试这些模型。

SystemVerilog 是对 Verilog HDL 的扩展。这些扩展的主要目标是用更紧凑的代码来对更大规模的设计进行建模和验证。若单独来说，SystemVerilog 并不是完整的语言，它仅仅是以 Verilog 语言为基础的扩展。

什么是陷阱

编程中的“陷阱”是一个语言特征，如果错误使用，将会在硬件设计中引起意想不到的潜在灾难性的行为。在 C 语言中一个典型的例子是条件表达式内的赋值，比如

```
if (day=15)                /* 陷阱！将 15 赋给 day          */
  do_mid_month_payroll;    /* 如果 day 非零，则处理薪水账册 */
```

在上述例子中，程序员很有可能将代码 if(a= =b) 误写为 if(a=b)，而结果将大相径庭！这个经典的 C 语言编程陷阱并非语法错误，它完全符合语法。然而，这段代码可能不会实现预期的结果。如果在产品出厂之前没有检测到这种代码错误，诸如此类的简单漏洞将严重影响产品的使用。

和任何编程语言一样，Verilog 和基于 Verilog 扩展的 SystemVerilog 也有陷阱。即便在 Verilog 和 SystemVerilog 上语法正确的结构，也可能产生意料之外的不良结果。其中导致 Verilog 和 SystemVerilog 的陷阱的主要原因有：

❑ 继承了 C 和 C++ 的陷阱

Verilog 和 SystemVerilog 沿用了 C 和 C++ 语言常用的语法和语义。Verilog 和 SystemVerilog 继承了 C 和 C++ 这些功能强大的编程语言的优势，但也继承了它们的许多陷阱。（这里不禁要问，常见的 C 代码错误，类似 if（day = 15），是否也会出现在 Verilog / SystemVerilog 中呢？ 陷阱 44 回答了该问题 ）。

❑ 松散类型操作

Verilog 和 SystemVerilog 是松散类型（loosely typed）语言。就其定义而言，其操作可以执行任何数据类型，且基本语言规则关心的是如何执行操作。如果设计或验证工程师并不了解这些基本语言规则，那么很可能得到意想不到的结果。

❑ 允许良莠不齐的模型设计

Verilog 和 SystemVerilog 的一个基本理念是允许工程师建模并验证哪些可以在硬件上正确工作，哪些不能在硬件上正常工作。为了建立一个符合语法而又不能工作的硬件模型，语言必然也允许那些原本想正常工作的，但却因无心的建模误差而导致的硬件模型不能正常工作的情况。

Verilog 和 SystemVerilog 标准

Verilog 是国际标准硬件描述语言，其官方标准是 IEEE.Std 1364-2005 Verilog language Reference Manual（LRM），常称为“Verilog-2005”。该 Verilog 标准针对表达数字逻辑行为定义了丰富的编程和建模结构。Verilog 硬件描述语言在 1984 年诞生。Verilog 是为满足 20 世纪 80 年代中期工业的需求而设计的，当时一个典型的设计不超过 50 000 个门，集成电路基于 3mm 技术。随着数字设计尺寸和技术的发展，Verilog 也随之演进以满足新的设计需求。Verilog 于 1995 年第一次由 IEEE 标准化（IEEE Std 1364-1995）。在 2001 年，IEEE 发布了 Verilog-2001 标准（IEEE Std 1364-2001），对 Verilog 的一些方面进行了改善，诸如针对任何矢量位宽及可重入任务和功能的有符号综合算法。IEEE 于 2005 年更新了 Verilog 标准，该版本没有加入重要的建模改进。而

所有对 Verilog 的增强都被编入一个单独标准——SystemVerilog。

SystemVerilog 是一套针对 Verilog-2005 标准的扩展。这些扩展单独写入一个标准，即 IEEE Std 1800-2005 SystemVerilog Language Reference Manual，常称为 SystemVerilog 2005。SystemVerilog 扩展使得编写尺寸和复杂度持续增长的综合模型，以及验证上万门的电路设计成为可能。SystemVerilog 为 Verilog 融入了 SUPERLOG、VERA C、C++ 及 VHDL 语言的特征，还有 OVA 和 PSL 断言。SystemVerilog 由 Accellera 发起，Accellera 是一个立足于电子设计的企业财团，其旗下的电子设计公司提供电子设计自动化（EDA）工具。Accellera 于 2002 年发布了一个针对 Verilog 的扩展版本，命名为 SystemVerilog 3.0（3.0 寓意 SystemVerilog 是下一代的 Verilog，其中 Verilog-1995 是第一代，Verilog2001 是第二代）。Accellera 在 2003 年和 2004 年分别发布了 SystemVerilog 3.1 和 SystemVerilog 3.1a。而后者被提交到 IEEE 作为完整的标准化。

IEEE 的初衷是将 Accellera 的 SystemVerilog 扩展合并到 Verilog 标准中。然而，在一些 EDA 公司的坚持下，IEEE 决定暂时将 SystemVerilog 扩展保持为一个独立的文件，以方便这些 EDA 公司在其 Verilog 工具中扩展新特性。

Verilog 和 SystemVerilog 的官方标准有：

- *IEEE 1364-2005 standard for the Verilog Hardware Description Language*, IEEE, Pascataway, NewJersey, 2002. ISBN 978-1-4020-7089-1.
- *IEEE 1800-2005 standard for the SystemVerilog Hardware Description and Verification Language*, IEEE, Pascataway, NewJersey, 2001. ISBN 0-73814811-3.

关于 Verilog 和 SystemVerilog 语言的更多详细内容可参考以下书籍：

- *The Verilog Hardware Description Language*, *5th edition*, by Donald Thomas and Philip Moorby. Published by Springer, Boston, MA, 2002, ISBN 978-1-4020-7089-1.
- *Verilog-2001: A Guide to the New Features in the Verilog Hardware Description Language*, by Stuart Sutherland. Published by Springer, Boston, MA, 2002, ISBN 978-0-7923-7568-5.
- *SystemVerilog for Design: A Guide to Using System Veri/og for Hardware Design and Modeling*, *Second Edition*, by Stuart Sutherland, Simon Davidmann and Peter Flake. Published by Springer, Boston, MA,2006, ISBN 978-0-387-33399-1.
- *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*, by Chris Spear. Published by Springer, Boston, MA, 2006, ISBN 978-0-387-27036-4.

此外，当前也有很多更优秀的书籍介绍 Verilog 和 SystemVerilog。

请注意，在编写这本书的时候，IEEE 已经开始将基础 Verilog 语言合并到 SystemVerilog 语言参考手册，用来建立一个统一的语言和标准。随着时间的推移，Verilog 将会消失，而 SystemVerilog 取而代之将成为唯一的、统一的“硬件设计和验证语言”。

第 2 章

声明以及字符表述类陷阱

陷阱 1：字母大小写的敏感性

陷阱：我的代码中的名字看起来并没有问题且在我的 VHDL 模型中可以正常工作，但是在 Verilog/SystemVerilog 环境下却报错“未声明的标识符”。

概要：Verilog 和 SystemVerilog 是对大小写字母敏感的语言，而 VHDL 是对大小写字母不敏感的语言。

Verilog 和 SystemVerilog 中的标识符（identifier）是一些由用户指定对象的名字，这些对象可以是模块、线型、变量或函数。Verilog 和 SystemVerilog 是对大小写敏感的语言，这意味着使用小写字母和大写字母的标识符或关键字有着不同的含义。关键字通常是由小写字母组成。用户指定的标识符可以混合使用大小写字母，也可以使用数字和特殊字符“_”、“$”及“\”(\ 是转义字符)。

对那些初学 Verilog/SystemVerilog 的工程师，尤其是那些从对大小写不敏感语言（如 VHDL）转过来的工程师而言，字母大小写敏感性经常是一个陷阱。即便是经验丰富的工程师有时也会犯字母大小写敏感性错误。一般来说，在同一个标识符有时使用小写字母，有时使用大写字母的情况下，字母大小写敏感性错误经常发生。

注意 本章中的代码示例是人为的，这些例子用于说明每个陷阱。在实际的设计和验证代码中，这些陷阱可能并非如此明显或容易发现。

接下来的示例有三处字母大小写敏感性陷阱。

```
module FSM (...);

  enum logic [1:0] {WAIT, LOAD, READY} State, nState;

  ...
```

```
  always_comb begin
    case (state)                    // GOTCHA!
      WAIT:  nState = L0AD;         // GOTCHA!
      LOAD:  nState = READY;
      READY: nState = wait;         // GOTCHA!
    endcase
  end
endmodule: FSM
```

上例中，第一个陷阱是枚举变量 State 的声明使用了字母大小写混合，而在代码的后半部分，也许是在声明的上百行代码后引用了一个名为 state 的信号。在英语中这些标识符读起来是一样的，但是在 Verilog 或 SystemVerilog 工具中，它们是不同的名字，这是陷阱！

第二个陷阱是枚举标签 LOAD 的所有字母均为大写。而在后面的代码部分却引用了一个叫 L0AD 的标识符。猛一看，这些标识符好像一样，但是对于 Verilog/SystemVerilog 工具而言，它们大相径庭。二者的差异是该枚举标签的名字包含了大写字母“O”(发音同“oh”)，而代码中所引用的名字包含了数字“0”(即零)。又掉入陷阱了！

在上述例子中第三个陷阱涉及枚举标签 WAIT。尽管语法正确，但对一个标识符来说这并非一个好的选择，因为 Verilog/SystemVerilog 的关键词中有 wait。而在该模块的后部分，变量 nState 由 wait 赋值。这样，Verilog/SystemVerilog 工具针对这个并未声明的标识符便不会报错。它将会产生在预期表达式处导致一个关于程序 wait 语句的错误。如果使用大写字母标识符的名字和使用全小写字母关键词的名字相同，将会降低代码可读性，并导致很难发现的代码错误，同时会引起原因不明的语法错误信息。第三次进入陷阱！

在上述例子中，名字的拼写错误会导致编译错误，这或许令人沮丧，但至少不会导致没有错误提示但产生非预期行为的陷阱。大小写敏感错误也会导致通过编译且没有任何错误提示的陷阱，从而带来设计上的功能性问题。陷阱 2 会给出一个这样的例子。

如何避免此类陷阱

一种避免这种字母大小写敏感性陷阱的途径就是在公司内部采用良好的命名规定，并在之后严格执行这些规定。本书中使用的命名规定是：

- 变量和网表的命名用小写字母。
  - 用户自定义类型命名以 _t 结束。
  - 枚举变量命名以 _e 结束。
  - 低态有效信号命名以 _n 结束。
- 常量名称和枚举标签全部使用大写字母（例如 LOAD）。

- 类定义名称以大写字母开头，后面跟小写字母（例如 class Packet）。
- 如果语言本身对大小写敏感，名字的选择不能和关键词冲突（例如用 HOLD 代替 WAIT）。

前面例子的正确写法应该是这样的：

```
module fsm (...);

  enum logic [1:0] {HOLD, LOAD, READY} state_e, nstate_e;

  ...

  always_comb begin
    case (state_e)                  // OK, names match declarations
      HOLD:  nstate_e = LOAD;       // OK, names match declarations
      LOAD:  nstate_e = READY;
      READY: nstate_e = HOLD;       // OK, names match declarations
    endcase
  end
endmodule: fsm
```

陷阱 2：网表的隐式声明

陷阱：我的设计中的引线出现一个笔误但编译器没有报错，而仅仅是在仿真中呈现出功能问题。

摘要：带有笔误的标识符可能会不知不觉地导致符合语法规定的隐式网类型声明。

当面对未声明的标识符时 Verilog/SystemVerilog 工具会如何处理呢？这个问题的答案取决于未声明标识符使用的场景。

- 如果在程序赋值语句的右边或左边使用未声明的标识符，则编译器报错。例如（也可参见陷阱 1）：

```
logic [7:0] foo;
initial foo = bar;    // ERROR: bar not declared
```

- 如果在连续赋值语句的右边使用未声明的标识符，则编译器报错。

```
logic [7:0] foo;
assign foo = bar;    // ERROR: bar not declared
```

- 如果在连续赋值语句的左边使用未声明的标识符，将会导致隐式网类型声明，且没有报错或警告。

```
logic [7:0] foo;
assign bar = foo;    // GOTCHA: bar not declared, but no error
```

- 如果未声明的标识符用来连接一个模块、接口、程序或原语，将会导致隐式网类型声明，且没有报错或警告。

上述最后一条规则将会导致难以发现设计中的一些功能性错误，以如下的两个 1bit

加法器的连接为例说明。

```
module adder (input  logic a, b, ci,
              output logic sum, co);
  ...
endmodule

module top;
  wire a, b, ci, s1, s2, c1;
  adder i1 (.a(a), .b(b), .ci(c1), .sum(s1), .co(c1) ); // GOTCHA!
  adder i2 (.a(a), .b(b), .ci(c),  .sum(s2), .co(co) ); // GOTCHA!
endmodule
```

本例的一个陷阱是 i1 加法器中 c1（第二个字符是数字 1）的声明，但是其引用却是 cl（第二个字符是字母 l）。另一个陷阱是 i2 加法器中未声明的标识符 c。这些笔误并非语法错误。相反，它们会导致设计中的隐式网类型声明，其导致的功能性错误必须找到并调试。又是一个陷阱！

为什么 Verilog/SystemVerilog 允许存在这个陷阱？因为在正确使用的情况下，自动推断隐式数据类型的能力是很有用的。这种隐式数据类型的一个好处就是在数以百万的大规模门电路设计中，没有必要一一声明成千上万的连接线。

如何在使用 Verilog 时避免这种陷阱

一些具有语言识别的编辑器（如含有 Verilog 模型的 Emacs）可以自动完成 Verilog 网表的连接。这是一个避免由此类拼写错误而导致陷阱的简单实用的方法。

包括本书作者在内的大多数工程师都发现了这种 Verilog 语言中隐式网表的强大功能，尽管这会导致潜在的陷阱，即网表的拼写错误会导致功能性错误而非编译错误。Verilog 语言确实提供了一种机制来禁用隐式数据类型。但这样做是有争议的。一些工程师认为引进更多的陷阱可能有助于避免陷阱。这种基于 Verilog 的控制并没有在本书中讨论。

如何在使用 SystemVerilog 时避免此类陷阱

SystemVerilog 提供两个方便的捷径，即“点 + 名”和“点 + 星”，用于模块实例、程序和接口的网表连接。这些快捷方式避免了端口连接重复的命名。

- “点 + 名”的快捷方式明确地将端口与指定的连接线对接，同时根据推断连接相同名字的网类型和端口。
- “点 + 星”的快捷方式自动推断具有相同名称的端口和信号，并将其连接。

下面的例子说明了所有的这三种连接方式：完全明确、“点 + 名”部分推断、“点 + 星”完全推断。