
HZ BOOKS
电子工程丛书
Verilog HDL
硬件描述语言
A Verilog HDL Primer
(Second Edition)
(美) J. Bhasker 著
徐振林 等译
Verilog HDL Primer
机械工业出版社
China Machine Press
Star Galaxy Publishing

电子工程丛书

Verilog HDL 硬件描述语言

(美) J. Bhasker 著
徐振林 等译

机械工业出版社
China Machine Press

本书简要介绍了Verilog硬件描述语言的基础知识，包括语言的基本内容和基本结构，以及利用该语言在各种层次上对数字系统的建模方法。书中列举了大量实例，帮助读者掌握语言本身和建模方法，对实际数字系统设计也很有帮助。

本书是Verilog HDL的初级读本，适用于作为计算机、电子、电气及自控等专业相关课程的教材，也可供有关的科研人员作为参考书。

本书版权登记号：图字：01-1999-2357

图书在版编目(CIP)数据

Verilog HDL硬件描述语言 /（美）贝斯克（Bhasker,J.）著；徐振林等译. – 北京：机械工业出版社，2000.7
(电子工程丛书)
书名原文：A Verilog HDL Primer (Second Edition)
ISBN 7-111-07890-X

Ⅰ.V… Ⅱ.①贝… ②徐… Ⅲ.硬件描述语言，Verilog HDL Ⅳ.TP312

中国版本图书馆CIP数据核字(2000)第26607号

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：陈贤舜
北京昌平环球印刷厂印刷·新华书店北京发行所发行
2000年7月第1版第1次印刷
787mm × 1092mm 1/16 · 11.5 印张
印数：0001-5 000册
定价：19.00元

前　言

本书简要概述了Verilog硬件描述语言的基础知识。

Verilog硬件描述语言，通常被缩写为Verilog HDL(本书在以后章节中亦用此缩写)，用于从开关级到算法级的多个抽象设计层次的数字设计的建模。该语言提供一套功能强大的原语集，包括逻辑门和用户定义的原语；并提供了丰富的结构，这些结构不仅用于硬件的并发行为的建模，而且用于硬件的时序特性和结构的建模。也可以通过编程语言接口（PLI）对这种语言进行扩展。Verilog HDL语言使用简便，功能强大，可以对多个抽象设计层次的建模。Verilog HDL语言于1995年由IEEE进行了标准化，称为IEEE Std 1364-1995，本书也正是基于这一标准。

本书的目的是通过实例解释语言的基本知识和重要结构，向读者介绍Verilog硬件描述语言。本书是Verilog HDL的初级读本。语言的每一方面都使用清晰、简洁的英语描述，初学者易于理解，不会产生畏难情绪。希望本书能够成为您学习Verilog HDL的入门教材。

本书从语言的特点及其在建模中的使用等方面全面描述了Verilog语言的基础知识。对每一语言结构都提供了大量的实例；并用实例解释说明如何使用多个结构进行硬件建模。Verilog HDL支持的多种建模方式在本书中都有详细解释。本书中还介绍了如何使用Verilog语言描述模拟激励和控制，包括响应监控和校验。许多结构的语法都采用易于阅读的方式显示，尽管有时并不完整。这样做的目的是便于理解结构。在附录中提供了Verilog结构完整的语法，以便参考。

本书在本质上并不是理论性的，因此使用普通的术语介绍语言的语法和语义，而没有使用语言正式定义的专业术语。本书并未试图对语言进行完整的讨论，例如编程语言接口、开关级和随机模型的特性在本书中就没有讨论。本书只限于讨论语言中足以对无论是简单还是复杂设备建模的最有用和常用的特性。

本书适用于，包括电路和系统设计者在内的硬件设计者、软件工具开发者以及对学习使用Verilog HDL硬件建模感兴趣的其它人士。本书也可作为计算机辅助设计、硬件建模或综合课程的入门课程。本书也适用于专业人士、本科生和研究生。设计者可以通过本书了解Verilog HDL，亦可将其作为使用Verilog HDL的参考书。对于学生和教授们来说，本书是硬件设计和硬件描述语言的非常有用的教学工具。

本书假定读者了解数字硬件设计的基本知识，并熟悉如C语言等高级编程语言。

最后，我认为仅通过阅读本书来学习Verilog HDL语言是不现实的。从书中打印出实例，在Verilog模拟器上编译并模拟它们是掌握该语言的最佳途径。一旦掌握了本书，就可以参阅IEEE标准语言参考手册（LRM）关于Verilog HDL标准的完整信息。

本书的结构

第1章介绍了语言简史，概述了语言的主要能力。

第2章通过实例说明设计描述的三种主要形式：数据流、行为和结构形式，对语言作简要概述。

第3章概述语言的基本组成要素，主要描述了标识符、注释、系统任务、编译开关指令和数据类型。

第4章讲解表达式。表达式可以在Verilog描述中的许多地方使用，包括时延，该章同时介绍了用于构成表达式的各种操作符和操作数。

第5章介绍门级建模方法，即使用内置基本门设计建模。同时解释了门时延，并介绍了时间和时延标度的概念。

Verilog HDL 提供了创建用户定义原语的能力，即内置原语以外的原语。这些内容是第6章的主题。同时通过实例，介绍了用户定义组合和时序的原语。

在Verilog HDL 中使用连续赋值语句的数据流建模方式。第7章描述这种形式的赋值方式并解释其执行语义。同时描述了两类时延，赋值时延和线网时延。

第8章介绍了行为建模方式。该章描述了两个主要过程化结构，即initial语句和always语句。该章还详细介绍了过程性赋值方式，并通过实例详细解释了顺序和并行分程序。高级编程结构如条件语句和循环语句也在该章中加以描述。

第9章详细描述了结构化形式的建模方式。该章探讨了层次和端口匹配的概念，同时介绍了如何通过端口关联连接不同模块。

第10章讲述了一些较有深度的主题，如块的定义、值变转储文件（即VCD文件）和信号强度等。该章还介绍了任务（task）和函数（function）。

第11章和第12章讨论了设计验证和建模，是本书最实用的章节。第11章讨论了一些产生波形和响应监控的测试验证程序（test bench）实例。第12章给出了一些建模实例，用以说明Verilog语言结构的综合应用。

最后，附录包括了Verilog语言的完整语法。语法采用巴科斯—诺尔范式（BNF）加以描述，为便于查询，所有语法结构均按字母表顺序编排。

在本书中出现的所有Verilog HDL表达式中，保留字、系统任务和系统函数以及编译指令都使用粗体字。在语法描述中作为语法一部分的操作符和标点标记使用粗体字。语法规则中的可选项使用非粗体方括号（[…]）。非粗体花括号（{…}）标识重复0次或多次的项。有时在Verilog HDL 源代码中的省略号（…）用以表明与讨论无关的代码。以 Courier字体书写的特定词汇用以标明取其Verilog中的意义，而不是英语中的意义，例如and门。

书中的所有实例均采用VeriBest(R) Verilog 14.0版的模拟器进行了验证。

J.Bhasker
Allentown, PA
1997年2月

第2版前言

第2版为每一章增加了习题；希望这会使本书更适于作为大学课程使用。新版增加了关于

共享任务和函数、MOS开关、双向开关以及命名事件等内容。页面格式和字体已重新设计，以使本书更便于阅读。

如果您有什么关于本书的疑问、批评和建议，敬请通过出版商与我联系。

J. Bhasker
1999年1月

第1章 简 介

本章介绍Verilog HDL语言的发展历史和它的主要能力。

1.1 什么是Verilog HDL?

Verilog HDL是一种硬件描述语言，用于从算法级、门级到开关级的多种抽象设计层次的数字系统建模。被建模的数字系统对象的复杂性可以介于简单的门和完整的电子数字系统之间。数字系统能够按层次描述，并可在相同描述中显式地进行时序建模。

Verilog HDL 语言具有下述描述能力：设计的行为特性、设计的数据流特性、设计的结构组成以及包含响应监控和设计验证方面的时延和波形产生机制。所有这些都使用同一种建模语言。此外，Verilog HDL语言提供了编程语言接口，通过该接口可以在模拟、验证期间从设计外部访问设计，包括模拟的具体控制和运行。

Verilog HDL语言不仅定义了语法，而且对每个语法结构都定义了清晰的模拟、仿真语义。因此，用这种语言编写的模型能够使用Verilog仿真器进行验证。语言从C编程语言中继承了多种操作符和结构。Verilog HDL提供了扩展的建模能力，其中许多扩展最初很难理解。但是，Verilog HDL语言的核心子集非常易于学习和使用，这对大多数建模应用来说已经足够。当然，完整的硬件描述语言足以对从最复杂的芯片到完整的电子系统进行描述。

1.2 历史

Verilog HDL语言最初是于1983年由Gateway Design Automation[⊖]公司为其模拟器产品开发的硬件建模语言。那时它只是一种专用语言。由于他们的模拟、仿真器产品的广泛使用，Verilog HDL 作为一种便于使用且实用的语言逐渐为众多设计者所接受。在一次努力增加语言普及性的活动中，Verilog HDL语言于1990年被推向公众领域。 Open Verilog International（OVI）是促进Verilog发展的国际性组织。1992年， OVI决定致力于推广Verilog OVI标准成为IEEE标准。这一努力最后获得成功，Verilog 语言于1995年成为IEEE标准，称为IEEE Std 1364 - 1995。完整的标准在Verilog硬件描述语言参考手册中有详细描述。

1.3 主要能力

下面列出的是Verilog硬件描述语言的主要能力：

- 基本逻辑门，例如**and**、**or**和**nand**等都内置在语言中。
- 用户定义原语（UDP）创建的灵活性。用户定义的原语既可以是组合逻辑原语，也可以是时序逻辑原语。
- 开关级基本结构模型，例如**pmos** 和**nmos**等也被内置在语言中。

⊖ Gateway Design Automation公司后来被Cadence Design Systems公司收购。

- 提供显式语言结构指定设计中的端口到端口的时延及路径时延和设计的时序检查。
- 可采用三种不同方式或混合方式对设计建模。这些方式包括：行为描述方式——使用过程化结构建模；数据流方式——使用连续赋值语句方式建模；结构化方式——使用门和模块实例语句描述建模。
- Verilog HDL中有两类数据类型：线网数据类型和寄存器数据类型。线网类型表示构件间的物理连线，而寄存器类型表示抽象的数据存储元件。
- 能够描述层次设计，可使用模块实例结构描述任何层次。
- 设计的规模可以是任意的；语言不对设计的规模（大小）施加任何限制。
- Verilog HDL不再是某些公司的专有语言而是IEEE标准。
- 人和机器都可阅读Verilog 语言，因此它可作为EDA的工具和设计者之间的交互语言。
- Verilog HDL语言的描述能力能够通过使用编程语言接口（PLI）机制进一步扩展。PLI是允许外部函数访问Verilog 模块内信息、允许设计者与模拟器交互的例程集合。
- 设计能够在多个层次上加以描述，从开关级、门级、寄存器传送级（RTL）到算法级，包括进程和队列级。
- 能够使用内置开关级原语在开关级对设计完整建模。
- 同一语言可用于生成模拟激励和指定测试的验证约束条件，例如输入值的指定。
- Verilog HDL 能够监控模拟验证的执行，即模拟验证执行过程中设计的值能够被监控和显示。这些值也能够用于与期望值比较，在不匹配的情况下，打印报告消息。
- 在行为级描述中，Verilog HDL不仅能够在RTL级上进行设计描述，而且能够在体系结构级描述及其算法级行为上进行设计描述。
- 能够使用门和模块实例化语句在结构级进行结构描述。
- 图1-1显示了Verilog HDL 的混合方式建模能力，即在一个设计中每个模块均可以在不同设计层次上建模。
- Verilog HDL 还具有内置逻辑函数，例如&（按位与）和|（按位或）。
- 对高级编程语言结构，例如条件语句、情况语句和循环语句，语言中都可以使用。
- 可以显式地对并发和定时进行建模。
- 提供强有力的文件读写能力。

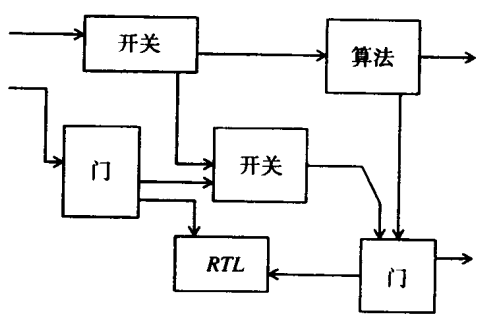

图1-1 混合设计层次建模

- 语言在特定情况下是非确定性的，即在不同的模拟器上模型可以产生不同的结果；例如，事件队列上的事件顺序在标准中没有定义。

习题

1. Verilog HDL 是在哪一年首次被IEEE标准化的？
2. Verilog HDL支持哪三种基本描述方式？
3. 可以使用Verilog HDL描述一个设计的时序吗？

4. 语言中的什么特性能够用于描述参数化设计?
5. 能够使用Verilog HDL 编写测试验证程序吗?
6. Verilog HDL 是由哪个公司最先开发的?
7. Verilog HDL中的两类主要数据类型什么?
8. UDP代表什么?
9. 写出两个开关级基本门的名称。
10. 写出两个基本逻辑门的名称。

第2章 HDL指南

本章提供HDL语言的速成指南。

2.1 模块

模块是Verilog 的基本描述单位，用于描述某个设计的功能或结构及其与其他模块通信的外部端口。一个设计的结构可使用开关级原语、门级原语和用户定义的原语方式描述；设计的数据流行为使用连续赋值语句进行描述；时序行为使用过程结构描述。一个模块可以在另一个模块中使用。

一个模块的基本语法如下：

```
module module_name (port_list);
  Declarations:
    reg, wire, parameter,
    input, output, inout,
    function, task, . . .
  Statements:
    Initial statement
    Always statement
    Module instantiation
    Gate instantiation
    UDP instantiation
    Continuous assignment
endmodule
```

说明部分用于定义不同的项，例如模块描述中使用的寄存器和参数。语句定义设计的功能和结构。说明部分和语句可以散布在模块中的任何地方；但是变量、寄存器、线网和参数等的说明部分必须在使用前出现。为了使模块描述清晰和具有良好的可读性，最好将所有的说明部分放在语句前。本书中的所有实例都遵守这一规范。

图2-1为建模一个半加器电路的模块的简单实例。

```
module HalfAdder (A, B, Sum, Carry);
  input A, B;
  output Sum, Carry;

  assign #2 Sum = A ^ B;
  assign #5 Carry = A & B;
endmodule
```

模块的名字是*HalfAdder*。模块有4个端口：两个输入端口*A*和*B*，两个输出端口*Sum*和*Carry*。由于没有定义端口的位数，所有端口大小都为1位；同时，由于没有各端口的数据类型说明，这四个端口都是线网数据类型。

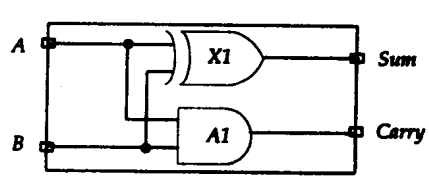

图2-1 半加器电路

模块包含两条描述半加器数据流行为的连续赋值

语句。从这种意义上讲，这些语句在模块中出现的顺序无关紧要，这些语句是并发的。每条语句的执行顺序依赖于发生在变量A和B上的事件。

在模块中，可用下述方式描述一个设计：

1) 数据流方式;

2) 行为方式;

3) 结构方式;

4) 上述描述方式的混合。

下面几节通过实例讲述这些设计描述方式。不过有必要首先对Verilog HDL的时延作简要介绍。

2.2 时延

Verilog HDL模型中的所有时延都根据时间单位定义。下面是带时延的连续赋值语句实例。

```
assign #2 Sum = A ^ B;
```

#2指2个时间单位。

使用编译指令将时间单位与物理时间相关联。这样的编译器指令需在模块描述前定义，如下所示:

```
` timescale 1ns /100ps
```

此语句说明时延时间单位为1ns并且时间精度为100ps (时间精度是指所有的时延必须被限定在0.1ns内)。如果此编译器指令所在的模块包含上面的连续赋值语句, #2 代表2ns。

如果没有这样的编译器指令, Verilog HDL 模拟器会指定一个缺省时间单位。IEEE Verilog HDL 标准中没有规定缺省时间单位。

2.3 数据流描述方式

用数据流描述方式对一个设计建模的最基本的机制就是使用连续赋值语句。在连续赋值语句中，某个值被指派给线网变量。连续赋值语句的语法为:

```
assign [delay] LHS_net = RHS_ expression;
```

右边表达式使用的操作数无论何时发生变化，右边表达式都重新计算，并且在指定的时延后变化值被赋予左边表达式的线网变量。时延定义了右边表达式操作数变化与赋值给左边表达式之间的持续时间。如果没有定义时延值，缺省时延为0。

图2-2显示了使用数据流描述方式对2-4解码器电路的建模的实例模型。

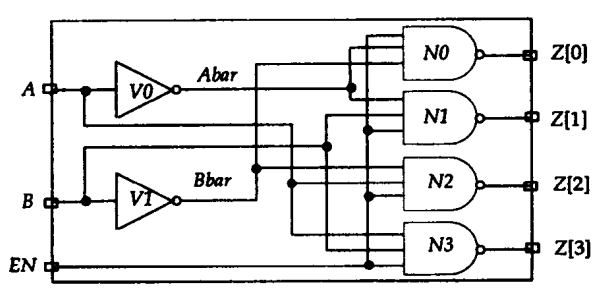

图2-2 2-4解码器电路

```
`timescale 1ns/ 1ns
module Decoder2x4 (A, B, EN, Z);
  input A, B, EN;
  output [ 0 :3] Z;
  wire Abar, Bbar;

  assign #1 Abar = ~ A;                          / / 语句 1。
  assign #1 Bbar = ~ B;                          / / 语句 2。
  assign #2 Z[0] = ~ (Abar & Bbar & EN) ;        / / 语句 3。
  assign #2 Z[1] = ~ (Abar & B & EN) ;           / / 语句 4。
  assign #2 Z[2] = ~ (A & Bbar & EN) ;           / / 语句 5。
  assign #2 Z[3] = ~ (A & B & EN) ;              / / 语句 6。
endmodule
```

以反引号“`”开始的第一条语句是编译器指令，编译器指令`**timescale** 将模块中所有时延的单位设置为1 ns，时间精度为1 ns。例如，在连续赋值语句中时延值#1和#2分别对应时延1 ns和2 ns。

模块*Decoder2x4*有3个输入端口和1个4位输出端口。线网类型说明了两个连线型变量*Abar*和*Bbar*(连线类型是线网类型的一种)。此外，模块包含6个连续赋值语句。

参见图2-3中的波形图。当*EN*在第5 ns变化时,语句3、4、5和6执行。这是因为*EN*是这些连续赋值语句中右边表达式的操作数。Z[0]在第7 ns时被赋予新值0。当*A*在第15 ns变化时，语句1、5和6执行。执行语句5和6不影响Z[0]和Z[1]的取值。执行语句5导致Z[2]值在第17 ns变为0。执行语句1导致*Abar*在第16 ns被重新赋值。由于*Abar*的改变，反过来又导致Z[0]值在第18 ns变为1。

请注意连续赋值语句是如何对电路的数据流行为建模的；这种建模方式是隐式而非显式的建模方式。此外,连续赋值语句是并发执行的，也就是说各语句的执行顺序与其在描述中出现的顺序无关。

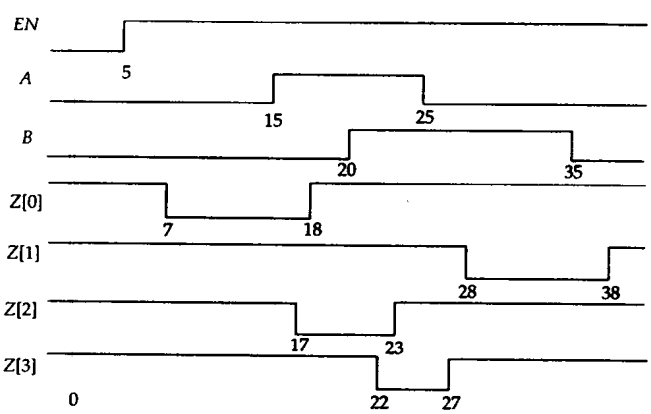

图2-3 连续赋值语句实例

2.4 行为描述方式

设计的行为功能使用下述过程语句结构描述：

1) initial语句：此语句只执行一次。

2) always语句：此语句总是循环执行，或者说此语句重复执行。

只有寄存器类型数据能够在这两种语句中被赋值。寄存器类型数据在被赋新值前保持原有值不变。所有的初始化语句和always语句在0时刻并发执行。

下例为always语句对1位全加器电路建模的示例，如图2-4。

```
module FA_Seq (A, B, Cin, Sum, Cout);
  input A, B, Cin;
  output Sum, Cout;
  reg Sum, Cout;
  reg T1, T2, T3;
  always
    @ ( A or B or Cin ) begin
      Sum = (A ^ B) ^ Cin;
      T1 = A & Cin;
      T2 = B & Cin;
      T3 = A & B;
      Cout = (T1| T2) | T3;
    end
endmodule
```

模块*FA_Seq* 有三个输入和两个输出。由于*Sum*、*Cout*、*T1*、*T2*和*T3*在always 语句中被赋值，它们被说明为 reg 类型(reg 是寄存器数据类型的一种)。always 语句中有一个与事件控制(紧跟在字符@ 后面的表达式)。相关联的顺序过程(begin-end对)。这意味着只要*A*、*B*或*Cin* 上发生事件，即*A*、*B*或*Cin*之一的值发生变化，顺序过程就执行。在顺序过程中的语句顺序执行，并且在顺序过程执行结束后被挂起。顺序过程执行完成后，always 语句再次等待*A*、*B*或*Cin*上发生的事件。

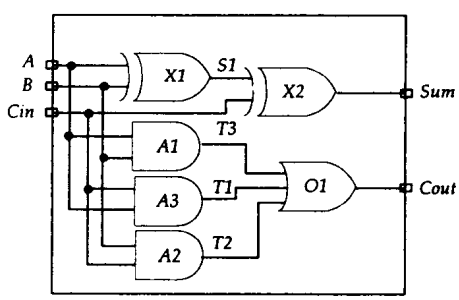

图2-4 1位全加器电路

在顺序过程中出现的语句是过程赋值模块化的实例。模块化过程赋值在下一条语句执行前完成执行。过程赋值可以有一个可选的时延。

时延可以细分为两种类型:

1) 语句间时延: 这是时延语句执行的时延。

2) 语句内时延: 这是右边表达式数值计算与左边表达式赋值间的时延。

下面是语句间时延的示例：

```
Sum = (A ^ B) ^ Cin;
#4 T1 = A & Cin;
```

在第二条语句中的时延规定赋值延迟4个时间单位执行。就是说，在第一条语句执行后等待4个时间单位，然后执行第二条语句。下面是语句内时延的示例。

```
Sum = #3 (A^ B) ^ Cin;
```

这个赋值中的时延意味着首先计算右边表达式的值，等待3个时间单位，然后赋值给*Sum*。

如果在过程赋值中未定义时延，缺省值为0时延，也就是说，赋值立即发生。这种形式以及在always 语句中指定语句的其他形式将在第8章中详细讨论。

下面是initial语句的示例：

```
`timescale 1ns / 1ns
```

```
module Test (Pop, Pid);
  output Pop, Pid;
  reg Pop, Pid;

  initial
    begin
      Pop = 0;          // 语句 1。
      Pid = 0;          // 语句 2。
      Pop = #5 1;       // 语句 3。
      Pid = #3 1;       // 语句 4。
      Pop = #6 0;       // 语句 5。
      Pid = #2 0;       // 语句 6。
  end
endmodule
```

这一模块产生如图2-5所示的波形。initial语句包含一个顺序过程。这一顺序过程在0 ns时开始执行，并且在顺序过程中所有语句全部执行完毕后，initial语句永远挂起。这一顺序过程包含带有定义语句内时延的分组过程赋值的实例。语句1和2在0 ns时执行。第三条语句也在0时刻执行,导致*Pop* 在第5 ns时被赋值。语句4在第5 ns执行,并且*Pid* 在第8 ns被赋值。同样，Pop在14 ns被赋值0，Pid在第16 ns被赋值0。第6条语句执行后，initial语句永远被挂起。第8章将更详细地讲解initial语句。

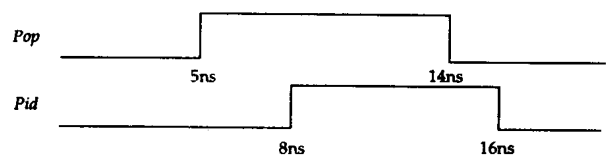

图2-5 *Test* 模块的输出波形

2.5 结构化描述形式

在Verilog HDL中可使用如下方式描述结构:

1) 内置门原语(在门级);

2) 开关级原语(在晶体管级);

3) 用户定义的原语(在门级);

4) 模块实例 (创建层次结构)。

通过使用线网来相互连接。下面的结构描述形式使用内置门原语描述的全加器电路实例。该实例基于图2-4所示的逻辑图。

```
module FA_Str (A, B, Cin, Sum, Cout);
  input A, B, Cin;
  output Sum, Cout;
  wire S1, T1, T2, T3;

  xor
    X1 (S1, A, B),
    X2 (Sum, S1, Cin);

  and
```

```
    A1 (T3, A, B),
    A2 (T2, B, Cin),
    A3 (T1, A, Cin),

  or
    O1 (Cout, T1, T2, T3);
endmodule
```

在这一实例中，模块包含门的实例语句，也就是说包含内置门**xor**、**and**和**or** 的实例语句。门实例由线网类型变量*S1*、*T1*、*T2*和*T3*互连。由于没有指定的顺序，门实例语句可以以任何顺序出现；图中显示了纯结构；xor、and和or是内置门原语；*X1*、*X2*、*A1*等是实例名称。紧跟在每个门后的信号列表是它的互连；列表中的第一个是门输出，余下的是输入。例如，*S1*与xor 门实例*X1*的输出连接，而*A*和*B*与实例*X1*的输入连接。

4位全加器可以使用4个1位全加器模块描述，描述的逻辑图如图2-6所示。下面是4位全加器的结构描述形式。

```
module FourBitFA (FA, FB, FCin, FSum, FCout );
  parameter SIZE = 4;
  input [SIZE:1] FA, FB;
  output [SIZE:1] FSum
  input FCin;
  input FCout;
  wire [ 1: SIZE-1] FTemp;
FA_Str
  FA1( .A (FA[1]), .B(FB[1]), .Cin(FCin),
     .Sum(FSum[1]), .Cout(FTemp[2])),
  FA2( .A (FA[2]), .B(FB[2]), .Cin(FTemp[1]),
     .Sum(FSum[2]), .Cout(FTemp[2])),
  FA3(FA[3], FB[3], FTemp[2], FSum[3], FTemp[3],
  FA4(FA[4], FB[4], FTemp[3], FSum[4], FCout);
endmodule
```

在这一实例中，模块实例用于建模4位全加器。在模块实例语句中，端口可以与名称或位置关联。前两个实例*FA1*和*FA2*使用命名关联方式，也就是说，端口的名称和它连接的线网被显式描述（每一个的形式都为“*.port_name* (*net_name*)）。最后两个实例语句，实例*FA3*和*FA4*使用位置关联方式将端口与线网关联。这里关联的顺序很重要，例如，在实例*FA4*中，第一个*FA*[4]与*FA_Str* 的端口*A*连接，第二个*FB*[4]与*FA_Str* 的端口*B*连接，余下的由此类推。

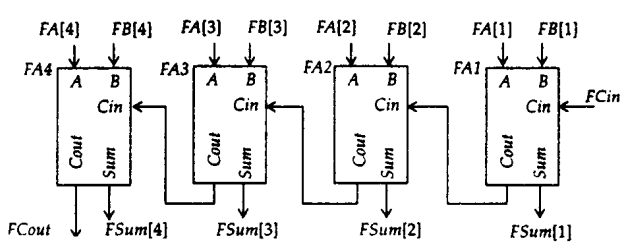

图2-6 4位全加器

2.6 混合设计描述方式

在模块中，结构的和行为的结构可以自由混合。也就是说，模块描述中可以包含实例化

的门、模块实例化语句、连续赋值语句以及always语句和initial语句的混合。它们之间可以相互包含。来自always语句和initial语句（切记只有寄存器类型数据可以在这两种语句中赋值）的值能够驱动门或开关，而来自于门或连续赋值语句（只能驱动线网）的值能够反过来用于触发always语句和initial语句。

下面是混合设计方式的1位全加器实例。

```
module FA_Mix (A, B, Cin, Sum, Cout);
  input A,B, Cin;
  output Sum, Cout;
  reg Cout;
  reg T1, T2, T3;
  wire S1;

  xor X1(S1, A, B);              // 门实例语句。

  always
    @ ( A or B or Cin ) begin   // always 语句。
      T1 = A & Cin;
      T2 = B & Cin;
      T3 = A & B;
      Cout = (T1| T2) | T3;
    end

  assign Sum = S1 ^ Cin;   // 连续赋值语句。
endmodule
```

只要*A*或*B*上有事件发生，门实例语句即被执行。只要*A*、*B*或*Cin*上有事件发生，就执行always 语句，并且只要*S1*或*Cin*上有事件发生，就执行连续赋值语句。

2.7 设计模拟

Verilog HDL不仅提供描述设计的能力，而且提供对激励、控制、存储响应和设计验证的建模能力。激励和控制可用初始化语句产生。验证运行过程中的响应可以作为“变化时保存”或作为选通的数据存储。最后，设计验证可以通过在初始化语句中写入相应的语句自动与期望的响应值比较完成。

下面是测试模块*Top*的例子。该例子测试2.3节中讲到的*FA_Seq*模块。

```
`timescale 1ns/1ns
module Top;          // 一个模块可以有一个空的端口列表。
  reg PA, PB, PCi;
  wire PCo, PSum;

  // 正在测试的实例化模块：
  FA_Seq F1(PA, PB, PCi, PSum, PCo);      // 定位。

  initial
    begin: ONLY_ONCE
      reg [3:0] Pal;
       //需要4位，Pal才能取值8。

      for (Pal = 0; Pal < 8; Pal = Pal + 1)
```

```
        begin
          {PA, PB, PCi} = Pal;
          #5 $display ( "PA, PB, PCi = %b%b%b", PA, PB, PCi,
                        " : : : PCo, PSum=%b%b", PCo, PSum);
        end
      end
    endmodule
```

在测试模块描述中使用位置关联方式将模块实例语句中的信号与模块中的端口相连接。也就是说，*PA*连接到模块*FA_Seq*的端口*A*，*PB*连接到模块*FA_Seq*的端口*B*，依此类推。注意初始化语句中使用了一个for循环语句，在*PA*、*PB*和*PCi*上产生波形。for 循环中的第一条赋值语句用于表示合并的目标。自右向左，右端各相应的位赋给左端的参数。初始化语句还包含有一个预先定义好的系统任务。系统任务$display将输入以特定的格式打印输出。

系统任务$display调用中的时延控制规定$display任务在5个时间单位后执行。这5个时间单位基本上代表了逻辑处理时间。即是输入向量的加载至观察到模块在测试条件下输出之间的延迟时间。

这一模型中还有另外一个细微差别。*Pal*在初始化语句内被局部定义。为完成这一功能，初始化语句中的顺序过程（**begin-end**）必须标记。在这种情况下，ONLY_ONCE是顺序过程标记。如果在顺序过程内没有局部声明的变量，就不需要该标记。测试模块产生的波形如图2-7显示。下面是测试模块产生的输出。

```
PA, PB, PCi = 000 ::: PCo, PSum = 00
PA, PB, PCi = 001 ::: PCo, PSum = 01
PA, PB, PCi = 010 ::: PCo, PSum = 01
PA, PB, PCi = 011 ::: PCo, PSum = 10
PA, PB, PCi = 100 ::: PCo, PSum = 01
PA, PB, PCi = 101 ::: PCo, PSum = 10
PA, PB, PCi = 110 ::: PCo, PSum = 10
PA, PB, PCi = 111 ::: PCo, PSum = 11
```

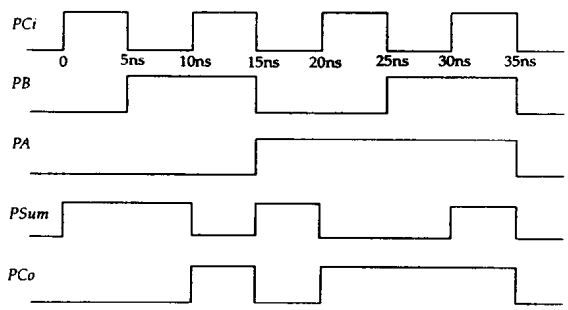

图2-7 测试模块*Top*执行产生的波形

验证与非门交叉连接构成的*RS_FF*模块的测试模块如图2-8所示。

```
`timescale 10ns/1ns
module RS_FF (Q, Qbar, R, S);
  output Q, Qbar;
  input R, S;

  nand #1 (Q, R, Qbar);
```

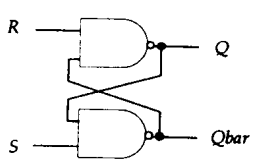

图2-8 交叉连接的与非门