EDA技术实用丛书

Verilog HDL 程序设计实例详解

张延伟 杨金岩 葛爱学 等 编著

附光盘

人民邮电出版社
POSTS & TELECOM PRESS

EDA技术实用丛书

Verilog HDL 程序设计实例详解

张延伟 杨金岩 葛爱学 等 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

Verilog HDL 程序设计实例详解 / 张延伟等编著. —北京：人民邮电出版社，2008.4
（EDA 技术实用丛书）
ISBN 978-7-115-17632-5

Ⅰ. V… Ⅱ. 张… Ⅲ. 硬件描述语言，Verilog HDL—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2008）第 018645 号

内 容 提 要

本书通过 100 多个模块实例，详细地讲解了 Verilog HDL 程序设计语言，全书共分 13 章，内容涉及 Verilog HDL 语言基本概念、建模、同步设计、异步设计、功能验证等，实例包括各种加法器/计数器、乘法器/除法器、编码器/译码器、状态机、SPI Master Controller、I2C Master controller、CAN Protocol Controller、Memory 模块、JPEG 图像压缩模块、加密模块、ATA 控制器、8 位 RISC-CPU 等及各个实例模块相应的 Testbench，所举实例具有很强的实用性和代表性，每个实例均给出了介绍、功能分析、程序代码和结果演示。

本书内容来自作者实际工作经验的总结及平常收集整理的相关资料，步骤详细，实例丰富，讲述循序渐进，是广大 IC 设计工程师、电子工程人员和高校师生不可多得的一本 Verilog HDL 参考用书。

EDA 技术实用丛书

Verilog HDL 程序设计实例详解

◆ 编　著　张延伟　杨金岩　葛爱学　等
责任编辑　刘　洋
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷
新华书店总店北京发行所经销
◆ 开本：787×1092　1/16
印张：23.75
字数：577 千字　　2008 年 4 月第 1 版
印数：1 – 5 000 册　　2008 年 4 月北京第 1 次印刷

ISBN 978-7-115-17632-5/TP

定价：48.00 元（附光盘）

读者服务热线：(010)67129258　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

最近10年，作为现代信息技术产业群的核心和基础，以IC为主的半导体产业飞速发展。从1997年至2002年，全球半导体市场总销售额从1400亿美元上升至近2500亿美元，预计到2012年全世界半导体产业的产值将达到1万亿美元，支持6～8万亿美元的电子设备产值，30万亿美元的信息服务业产值。以IC为支柱的半导体产业在全球经济体系中扮演着至关重要的角色。

巨大的潜在市场需求，吸引了全世界的IC巨头们进入中国市场。中国将成为未来全球半导体产业的主要市场，这一点已经成为业界共识。但我国半导体产业与国际水平相比还有不小的差距，特别是IC设计人才匮乏，制约着我国半导体产业的发展，因此加快培养专业人才成为重中之重。

Verilog HDL语言是一种在专用集成电路设计领域具有广泛应用前景的硬件描述语言，于1995年被接纳为IEEE标准，标准编号为IEEE Std 1364-1995。Verilog HDL语言可用于描述介于简单的门和完整的电子数字系统之间的数字系统模块，支持硬件设计开发、验证、综合和测试，硬件数据之间的通信，硬件数据的维护和修改。由于其简单、直观并富有效率，因此，已成为数字系统设计的首选语言，并成为综合、验证和布局布线技术的基础。利用Verilog HDL语言设计数字逻辑电路和数字系统的新方法，是电子电路设计领域的一次革命性的变化，也是21世纪的IC设计工程师所必须掌握的专业知识。

本书第1章介绍Verilog HDL语言基础知识，包括Verilog HDL语法特性、数据类型、运算符、程序结构等，在此基础上简单介绍了Verilog HDL仿真软件ModelSim SE的使用。

第2～5章介绍加法器/计数器、乘法器/除法器、编码器/译码器、状态机设计，这些实例较简单，在各个章节中结合实例介绍Verilog HDL语言的知识点，包括Task任务、Function函数、case语句、if-else语句等。

第6～8章介绍常用的通信协议控制器设计，包括SPI Master Controller、I2C Master Controller、CAN Protocol Controller，每个实例在介绍之前都分析了相关协议，并对各个关键子模块进行了详细分析。

第9章介绍Memory模块设计，包括异步FIFO和DDR SDRAM。

第10章将JPEG图像压缩模块分成色度空间转换CSC、离散余弦变换DCT、量化取整QNR、哈夫曼编码huffman_enc等4个子模块，并逐一介绍，包括子模块测试Testbench。

第11～13章介绍较复杂的模块设计，包括加密模块DES/AES、ATA控制器、8位RISC-CPU。

本书循序渐进，通过实例模块及其测试模块设计来介绍Verilog HDL语言，希望此书对IC设计工程师、电子工程人员和高校相关专业师生有所帮助。由于作者水平有限，加之时间仓促，本书难免有错误和不足之处，敬请广大读者批评指正。本书责任编辑的电子邮箱为liuyang@ptpress.com.cn。

作　者

2008年1月

目　录

第 1 章　Verilog HDL 基础知识 ····· 1
1.1　Verilog HDL 的基础语言知识 ····· 1
1.1.1　综述 ····· 1
1.1.2　Verilog HDL 语法特性 ····· 2
1.1.3　Verilog HDL 数据类型 ····· 4
1.1.4　Verilog HDL 运算符 ····· 5
1.1.5　Verilog HDL 程序结构 ····· 6
1.2　ModelSim SE 使用简介 ····· 8

第 2 章　加法器/计数器实例 ····· 10
2.1　1bit 半加法器 adder 设计实例 ····· 10
2.1.1　1bit 半加法器 adder 设计 ····· 10
2.1.2　adder Testbench 设计 ····· 11
2.1.3　adder Testbench 执行结果及仿真波形 ····· 12
2.2　1bit 全加法器 full_add 设计实例 ····· 13
2.2.1　1bit 全加法器 full_add 设计 ····· 13
2.2.2　full_add Testbench 设计 ····· 15
2.2.3　full_add Testbench 执行结果及仿真波形 ····· 16
2.3　同步 4bit 全加法器 adder4 设计实例 ····· 17
2.3.1　同步 4bit 全加法器 adder4 设计 ····· 17
2.3.2　adder4 Testbench 设计 ····· 18
2.3.3　adder4 Testbench 执行结果及仿真波形 ····· 20
2.4　4bit 计数器 count4 设计实例 ····· 22
2.4.1　4bit 计数器 count4 设计 ····· 22
2.4.2　count4 Testbench 设计 ····· 22
2.4.3　count4 Testbench 执行结果及仿真波形 ····· 23
2.5　8bit BCD 码计数器 count60 设计实例 ····· 24
2.5.1　8bit BCD 码计数器 count60 设计 ····· 24
2.5.2　count60 Testbench 设计 ····· 27
2.5.3　count60 Testbench 执行结果及仿真波形 ····· 27

第 3 章　乘法器/除法器实例 ····· 29
3.1　加法树乘法器 add_tree_mult 设计实例 ····· 29
3.1.1　加法树乘法器 add_tree_mult 设计 ····· 29
3.1.2　add_tree_mult Testbench 设计 ····· 32

3.1.3 add_tree_mult Testbench 执行结果及仿真波形 …… 33
3.2 查找表乘法器 lookup_mult 设计实例 …… 34
3.2.1 查找表乘法器 lookup_mult 设计 …… 34
3.2.2 lookup_mult Testbench 设计 …… 37
3.2.3 lookup_mult Testbench 执行结果及仿真波形 …… 37
3.3 布尔乘法器 booth_mult 设计实例 …… 39
3.3.1 布尔乘法器 booth_mult 设计 …… 39
3.3.2 booth_mult Testbench 设计 …… 42
3.3.3 booth_mult Testbench 执行结果及仿真波形 …… 44
3.4 移位除法器 shift_divider 设计实例 …… 46
3.4.1 移位除法器 shift_divider 设计 …… 46
3.4.2 shift_divider Testbench 设计 …… 52
3.4.3 shift_divider Testbench 执行结果及仿真波形 …… 55
第 4 章 编码器/译码器实例 …… 57
4.1 二进制编码器 bin_enc 设计实例 …… 57
4.1.1 二进制编码器 bin_enc 设计 …… 57
4.1.2 bin_enc Testbench 设计 …… 59
4.1.3 bin_enc Testbench 执行结果及仿真波形 …… 60
4.2 曼彻斯特编译码器 manch_ed 设计实例 …… 60
4.2.1 曼彻斯特编码器 manch_en 设计 …… 60
4.2.2 manch_en Testbench 设计 …… 63
4.2.3 manch_en Testbench 执行结果及仿真波形 …… 64
4.2.4 曼彻斯特译码器 manch_de 设计 …… 65
4.2.5 manch_de Testbench 设计 …… 67
4.2.6 manch_de Testbench 执行结果及仿真波形 …… 68
4.2.7 曼彻斯特编译码器 manch_ed 设计 …… 69
4.3 密勒译码器 miller_de 设计实例 …… 70
4.3.1 密勒译码器 miller_de 总体设计 …… 70
4.3.2 检测模块 signal_detect 设计 …… 71
4.3.3 signal_detect Testbench 设计 …… 73
4.3.4 signal_detect Testbench 执行结果及仿真波形 …… 75
4.3.5 译码模块 decode 设计 …… 76
4.3.6 decode Testbench 设计 …… 79
4.3.7 decode Testbench 执行结果及仿真波形 …… 80
4.3.8 密勒译码器 miller_de 顶层设计 …… 81
第 5 章 状态机实例 …… 83
5.1 状态机介绍 …… 83
5.2 16 位乘法器状态机实现 …… 84

5.2.1 16 位乘法器 mult16 设计 ····· 84
5.2.2 mult16 Testbench 设计 ····· 86
5.3 交通控制灯控制设计 ····· 87
5.3.1 交通控制灯 traffic 总体构架 ····· 87
5.3.2 traffic 状态机设计 ····· 88
5.3.3 traffic Testbench 设计 ····· 91
5.3.4 traffic Testbench 执行结果及仿真波形 ····· 93
5.4 PCI 总线目标接口状态机设计 ····· 93
5.4.1 PCI 总线介绍 ····· 93
5.4.2 PCI 总线目标接口总体构架 ····· 94
5.4.3 PCI 总线目标接口 State Machine 设计 ····· 95
5.4.4 PCI Target Testbench 设计 ····· 106
5.4.5 PCI Target Testbench 执行结果及仿真波形 ····· 108
第 6 章 SPI Master Controller 实例 ····· 111
6.1 SPI 协议介绍 ····· 111
6.2 SPI Master Controller 设计 ····· 113
6.2.1 SPI Master Controller 总体构架 ····· 113
6.2.2 时钟产生模块 spi_clgen 设计 ····· 113
6.2.3 串行接口模块 spi_shift 设计 ····· 115
6.2.4 spi_top 顶层模块设计 ····· 121
6.3 SPI Master Controller Testbench 设计 ····· 126
6.3.1 spi_top Testbench 总体构架 ····· 126
6.3.2 模拟 Wishbone master 模块设计 ····· 126
6.3.3 模拟 SPI slave 模块设计 ····· 128
6.3.4 spi_top Testbench 顶层模块设计 ····· 129
6.3.5 spi_top Testbench 执行结果及仿真波形 ····· 129
第 7 章 I2C Master Controller 实例 ····· 132
7.1 I2C 总线介绍 ····· 132
7.2 I2C Master Controller 设计 ····· 135
7.2.1 I2C Master Controller 总体构架 ····· 135
7.2.2 bit 传输模块 i2c_master_bit_ctrl 设计 ····· 136
7.2.3 byte 传输模块 i2c_master_byte_ctrl 设计 ····· 142
7.2.4 i2c_master_top 模块设计 ····· 146
7.3 I2C Master Controller Testbench 设计 ····· 150
7.3.1 i2c_master_top Testbench 总体构架 ····· 150
7.3.2 Wishbone master 模块设计 ····· 150
7.3.3 i2c_slave_model 模块设计 ····· 151
7.3.4 i2c_master_top Testbench 顶层模块设计 ····· 155

7.3.5 i2c_master_top Testbench 执行结果及仿真波形 …… 160
第 8 章 CAN Protocol Controller 实例 …… 162
8.1 CAN Protocol Controller 总体构架 …… 162
8.2 CAN Protocol Controller 模块设计 …… 165
8.2.1 CAN Protocol Controller 总体构架 …… 165
8.2.2 位时序操作模块 can_btl 设计 …… 166
8.2.3 bit stream 处理模块 can_bsp 设计 …… 171
8.3 CAN Protocol Controller Testbench 设计 …… 181
8.3.1 can_top Testbench 总体构架 …… 181
8.3.2 Test task 设计 …… 181
8.3.3 can_top Testbench 顶层模块设计 …… 185
8.3.4 can_top Testbench 执行结果及仿真波形 …… 187
第 9 章 Memory 模块实例 …… 190
9.1 异步 FIFO 设计实例 …… 190
9.1.1 异步 FIFO 简介 …… 190
9.1.2 异步 FIFO 设计 …… 192
9.1.3 异步 FIFO Testbench 设计 …… 194
9.2 DDR SDRAM Controller 设计实例 …… 200
9.2.1 SDRAM 简介 …… 200
9.2.2 DDR SDRAM Controller 设计 …… 201
9.2.3 DDR SDRAM Controller Testbench 设计 …… 212
第 10 章 JPEG 图像压缩模块实例 …… 218
10.1 JPEG 图像压缩模块简介 …… 218
10.2 色度空间转换 CSC 设计实例 …… 219
10.2.1 色度空间简介 …… 219
10.2.2 色度空间转换 CSC 设计 …… 220
10.2.3 色度空间转换 CSC Testbench 设计 …… 223
10.3 离散余弦变换 DCT 设计实例 …… 226
10.3.1 离散余弦变换简介 …… 226
10.3.2 离散余弦变换 DCT 设计 …… 227
10.3.3 离散余弦变换 DCT Testbench 设计 …… 235
10.4 量化取整 QNR 设计实例 …… 238
10.4.1 量化取整简介 …… 238
10.4.2 量化取整 QNR 设计 …… 239
10.4.3 量化取整 QNR Testbench 设计 …… 244
10.5 哈夫曼编码 huffman_enc 设计实例 …… 248
10.5.1 哈夫曼编码简介 …… 248

10.5.2 哈夫曼编码 huffman_enc 设计 …… 248
10.5.3 哈夫曼编码 huffman_enc Testbench 设计 …… 252

第 11 章 DES/AES 加密模块实例 …… 258
11.1 DES 加密模块设计 …… 258
11.1.1 DES 加密算法介绍 …… 258
11.1.2 DES 加密模块设计 …… 260
11.1.3 DES 加密模块 Testbench 设计 …… 276
11.2 AES 加密模块设计 …… 279
11.2.1 AES 加密算法介绍 …… 279
11.2.2 AES 加密模块设计 …… 280
11.2.3 AES 加密模块 Testbench 设计 …… 295

第 12 章 ATA 主机控制器实例 …… 300
12.1 ATA 协议介绍 …… 300
12.1.1 ATA 协议 …… 300
12.1.2 ATA 数据传输方式 …… 300
12.1.3 ATA 命令传输 …… 301
12.2 ATA 主机控制器设计 …… 302
12.2.1 ATA 主机控制器总体构架 …… 302
12.2.2 atahost_controller 设计 …… 304
12.2.3 PIO 时序控制器 atahost_pio_tctrl 设计 …… 307
12.2.4 运行计数器模块 ro_cnt 设计 …… 310
12.2.5 atahost_wb_slave 设计 …… 311
12.3 ATA 主机控制器 Testbench 设计 …… 317
12.3.1 ATA 主机控制器 Testbench 总体构架 …… 317
12.3.2 ATA 设备 ata_device 设计 …… 317
12.3.3 io_test1 Task 设计 …… 321
12.3.4 io_test2 Task 设计 …… 323
12.3.5 int_test Task 设计 …… 327
12.3.6 rst_test Task 设计 …… 329
12.3.7 test_bench_top 设计 …… 330
12.3.8 ATA 主机控制器 Testbench 执行结果及仿真波形 …… 332

第 13 章 8 位 RISC-CPU 实例 …… 335
13.1 RISC-CPU 介绍 …… 335
13.1.1 RISC-CPU 基本构架 …… 335
13.1.2 RISC-CPU 的功能及模块的划分 …… 336
13.2 RISC-CPU 设计 …… 336
13.2.1 RISC-CPU 总体构架 …… 336

13.2.2 算术逻辑单元 alu 设计 …… 339
13.2.3 可选扩展模块 exp 设计 …… 340
13.2.4 指令译码器 idec 设计 …… 342
13.2.5 寄存器文件 regs 设计 …… 345
13.2.6 可编程存储器 pram 设计 …… 346
13.2.7 cpu 设计 …… 347
13.3 RISC-CPU Testbench 设计 …… 358
13.3.1 RISC-CPU Testbench 总体构架 …… 358
13.3.2 RISC-CPU Task 设计 …… 359
13.3.3 RISC-CPU Testbench 顶层设计 …… 363
13.3.4 RISC-CPU Testbench 执行结果及仿真波形 …… 364
缩略语 …… 366
参考文献 …… 368

第 1 章　Verilog HDL 基础知识

1.1　Verilog HDL 的基础语言知识

1.1.1　综述

Verilog HDL 于 1983 年由 GateWay Design Automation 公司的 Phil Moorby 首创，Phil Moorby 后来成为 Verilog-XL 的主要设计者和 Cadence 公司的第一合伙人。1984～1985 年，Phil Moorby 设计出了第一个名为 Verilog-XL 的仿真器；1986 年，他对 Verilog HDL 的发展又做出了另一个巨大的贡献：提出了用于快速门级仿真的 XL 算法。

随着 Verilog-XL 算法的成功，Verilog HDL 语言得到了迅速发展。Cadence 公司于 1990 年公开 Verilog HDL 语言。OVI（Open Verilog International）是促进 Verilog 发展的国际性组织，负责促进 Verilog HDL 语言的发展。基于 Verilog HDL 的优越性，IEEE 于 1995 年制定了 Verilog HDL 的 IEEE 标准，即 Verilog HDL 1364-1995；2001 年发布了 Verilog HDL 1364-2001 标准。在这个标准中，加入了 Verilog HDL-A 标准，使 Verilog 有了模拟设计描述的能力。

Verilog HDL 是一种用于数字逻辑电路设计的语言，用于多种抽象层次的数字系统建模，按层次描述介于简单的门和完整的数字电子系统之间的数字系统模块，并可在相同描述中进行时序建模，这些抽象层次按级别和它们对应的模型类型可分为以下 5 种[1]。

- 系统级（System Level）：用高级语言结构设计实现模块的外部性能；
- 算法级（Algorithmic Level）：用高级语言结构设计实现算法；
- 寄存器传送级（Register Transfer Level）：描述数据在寄存器之间的流动和如何处理这些数据；
- 门级（Gate Level）：描述逻辑门以及逻辑门之间的连接；
- 开关级（Switch Level）：描述器件中三极管和存储节点以及它们之间的连接。

Verilog HDL 语言可使用同一建模语言描述设计的行为特性、数据流特性、结构组成、响应监控、验证时延和波形产生机制，在模拟和验证中，Verilog HDL 语言提供了编程语言接口，基于该接口可以从设计外部访问设计模型。

一个复杂电路的完整 Verilog HDL 模型由若干个 Verilog HDL 模块构成，每一个模块又可以由若干个子模块构成。利用 Verilog HDL 语言结构所提供的这种功能就可以构造一个模块间的清晰层次结构来描述极其复杂的大型设计。

Verilog HDL 行为描述语言作为一种结构化和过程性的语言，其语法结构非常适用于算法级和 RTL 级的模型设计，这种行为描述语言具有以下功能。

- 基本逻辑门，例如 and、or 和 nand 等都内置在语言中；

- 开关级基本结构模型，例如 pmos 和 nmos 等也被内置在语言中；
- 用户定义原语创建的灵活性，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；
- 可描述顺序执行或并行执行的程序结构；
- 可采用 3 种不同方式或混合方式设计建模，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；
- 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，能够使用门和模块实例化语句在结构级进行结构描述；
- 用延迟表达式或事件表达式来明确地控制过程的启动时间；
- 通过命名的事件来触发其他过程里的激活行为或停止行为；
- 提供了条件、if-else、case、循环程序结构；
- 提供了可带参数且非零延续时间的任务（Task）程序结构；
- 提供了可定义新的操作符的函数（Function）结构；
- 提供了用于建立表达式的算术运算符、逻辑运算符、位运算符；
- 提供了延迟和输出强度的原语来建立精确程度很高的信号模型，信号值可以有不同的强度，可以通过设定宽范围的模糊值来降低不确定条件的影响。

1.1.2 Verilog HDL 语法特性

Verilog HDL 语言中常数可以是整数或实数，整数可以标明位数也可以不标明位数，表示方法如下：

```
（位数）‘（基数）（值）
```

其中位数表明该数用二进制的几位来表示，基数可以是二（b）、八（o）、十（d）或十六（h）进制，值可以是所选基数的任何合法的值，包括不定值 x 位和高阻值 z，如：16‘h33ff、8‘b1001_1001、‘h93e。

实常数可以用十进制表示也可以用科学浮点数表示，如：45E-4（表示 0.004 5）、4.7E4（表示 47 000）。

Verilog HDL 语言中，字符串常常用于表示命令内需要显示的信息，用“ ”括起来的一行字符串，换新一行用“\n”字符，在字符串中可以用 C 语言中的各种格式控制符，如：\t、\”、\\…。

在字符串中可以用 C 语言中的各种数值型控制符，如：%b（二进制）、%o（八进制）、%d（十进制）、%h（十六进制）、%t（时间类型）、%s（字符串类型）…。

所谓标识符就是用户为程序描述中的 Verilog HDL 对象所起的名字，标识符必须以英语字母（a-z，A-Z）起头，或者用下横线符（_）起头。其中可以包含数字、$符和下横线符，标识符最长可以达到 1 023 个字符，模块名、端口名和实例名都是标识符，Verilog HDL 语言是大小写敏感的，因此 MUX 和 mux 是两个不同的标识符。

合法的标识符如下：

```
adder_reg_a
```

```
opera_point
index33
```

非法的标识符如下：

```
45bus      // 不能用数字开头
a*b_net    // 不能含有非字母符号*
a@mail     // 不能含有非字母符号@
```

特别标识符是以“\”符开始，以空格符结束的标识符。它可以包含任何可打印的 ASCII 字符。但“\”符和空格并不算是标识符的一部分。特别标识符往往是由 RTL 级源代码或电路图类型的设计输入经过综合器自动综合生成的网表结构型。

举例说明：

```
\~#@sel ,  \bus+index ,  \{A,B} ,
Top.\3inst .net1 , // 在层次模块中的标识名
```

$<标识符>，‘$’ 符号表示 Verilog HDL 的系统任务和函数，常用的系统任务和函数有下面几种：

```
$time                    // 找到当前的仿真时间
$display, $monitor       // 显示和监视信号值的变化
$stop                    // 暂停仿真
$finish                  // 结束仿真
```

例：initial $monitor（$time,,”a=%b, b=%b”, a, b）; // 每当 a 或 b 值变化时该系统任务都显示当前的仿真时刻，并分别用二进制和十六进制显示信号 a 和 b 的值。

编译引导语句用键盘左上角小写键“`”起头，用于指导仿真编译器在编译时采取一些特殊处理使编译引导语句一直保持有效，直到被取消或重写。`resetall 编译引导语句把所有设置的编译引导恢复到默认状态，常用的编译引导列举如下。

（1）`define：使用`define 编译引导能提供简单的文本替代功能，格式：

```
`define <宏名>  <宏文本>
```

在编译时会用宏文本来替代源代码中的宏名，例如：

```
`define or_delay #3
`define me_off  1
```

在程序中可以用有含义的文字来表示没有意思的数码，提高了程序的可读性，在程序中可以用`or_delay，me_off 分别表示#3 和 1。

（2）`include：使用`include 编译引导，在编译时能把其指定的整个文件包括进来一起处理，如：

```
`include “control.v”
```

合理地使用`include 可以使程序简洁、清晰、条理清楚、易于查错。

（3）`timescale 用于说明程序中的时间单位和仿真精度，如：

```
`timescale 1ns/100ps
module MCONTROL(out,a,b,sel);
…
not #1 not1(nsel, sel);
and #2 and1(a1, a, nsel);
…
endmodule
```

`timescale 语句必须放在模块边界前面，尽可能地使精度与时间单位接近，只要满足设计的实际需要就行，仿真步长即仿真单位是所有参加仿真模块中由`timescale 指定的精度中最高（即时间最短）的那个决定的。

（4）`uselib：用于定义仿真器到哪里去找库元件，如果该引导语句启动，它就一直有效直到遇到另外一个`uselib 的定义或`resetall 语句。`uselib 格式如下：

```
`uselib    器件库 1 的地点      器件库 2 的地点
```

上面的器件库地点可用以下两种方法表示：

① file = 库文件名的路径

② dir = 库目录名的路径　　libext =文件扩展

例如：

```
`uselib dir =/lib/SLOW_lib/
`uselib dir =/lib/TTL_lib/        libext=.v
                file = /libs/TTL_U/udp.lib
```

1.1.3 Verilog HDL 数据类型

Verilog HDL 规定了 4 种逻辑值：0、1、X、Z，如图 1-1 所示。0 表示低、伪、逻辑低、地、VSS、负插入；1 表示高、真、逻辑高、电源、VDD、正插入；X 表示不确定，逻辑冲突无法确定其逻辑值；Z 表示 HiZ、高阻抗、三态、无驱动源。

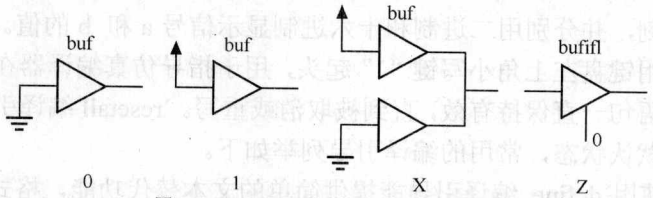

图 1-1　Verilog HDL 规定的四种逻辑值

Verilog HDL 有 3 种主要的数据类型：Nets 表示器件之间的物理连接，称为网络连接类型；Register 表示抽象的存储单元，称为寄存器/变量类型；Parameter 表示运行时的常数，称为参数类型。

（1）Nets（网络连线）：由模块或门驱动的连线，驱动端信号的改变会立刻传递到输出的连线上。在为不同工艺的基本元件建立库模型的时候，常常需要用不同的连接类型来与之对应，使其行为与实际器件一致，常见的有以下几种。

类型	功能
wire, tri	标准的互连线（默认）
supply1, supply2	电源线或接地线
wor, trior	有多个驱动源的线或逻辑连接
wand, triand	有多个驱动源的线与逻辑连接
trireg	有电容存在能暂时存储电平的连接
tri1, tri0	需要上拉或下拉的连接

如果不明确地说明连接是何种类型，一般是指 wire 类型。

（2）Register（寄存器）类型变量：Register 型变量能保持其值，直到它被赋予新的值，Register 型变量常用于行为建模，产生测试的激励信号，常用行为语句结构来给寄存器类型

的变量赋值，寄存器类型变量共有以下 4 种数据类型。

类型	功能
reg	无符号整数变量，可以选择不同的位宽
integer	有符号整数变量，32 位宽
real	有符号的浮点数，双精度
time	无符号整数变量，64 位宽

（3）Parameter（参数）类型：常用参数来声明运行时的常数，可用字符串表示的任何地方，都可以用定义的参数来代替，参数是本地的，其定义只在本模块内有效。

1.1.4 Verilog HDL 运算符

Verilog HDL 语言参考了 C 语言中大多数运算符的语义和句法，运算符包括算术运算符、逻辑运算符、关系运算符、等式运算符、移位运算符、位拼接运算符、缩减运算符等。

（1）算术运算符，又称为二进制运算符，Verilog HDL 语言中规定了 5 种算术运算符。

① + 加法或正值运算；

② – 减法或负值运算；

③ * 乘法运算；

④ / 除法运算；

⑤ % 模或求余运算。

（2）逻辑运算符，Verilog HDL 语言中规定了 3 种逻辑运算符。

① && 逻辑与；

② || 逻辑或；

③ ! 逻辑非。

逻辑运算符是二目运算符，它要求有两个操作数，如(a>b)&&(b>c)、(a>b)||(b>c)“!”是一目运算符，只要求一个操作数，如(! a)。逻辑运算符中“&&”和“||”的优先级别低于关系运算符，“!”高于算术运算符。

（3）关系运算符，Verilog HDL 语言中规定了以下 4 种关系运算符。

① a < b　　a 小于 b；

② a > b　　a 大于 b；

③ a <= b　　a 小于或等于 b；

④ a >= b　　a 大于或等于 b。

在进行关系运算时，如果声明的关系是假的（flase），则返回值是 0；如果声明的关系是真的（true），则返回值是 1；如果某个操作数的值不定，则关系是模糊的，返回值是不定值。

（4）等式运算符，Verilog HDL 语言中规定了 4 种等式运算符。

① == （等于）；

② != （不等于）；

③ === （等于）；

④ !== （不等于）。

这 4 个运算符都是二目运算符，它要求有两个操作数。“==”和“!=”又称为逻辑等式

运算符。其结果由两个操作数的值决定。由于操作数中某些位可能是不定值 x 或高阻值 z，结果可能为不定值 x。而“===”和“!==”运算符则不同，它们在对操作数进行比较时对某些位的不定值 x 和高阻值 z 也进行比较，两个操作数必须完全一致，其结果才是 1，否则为 0。“===”和“!==”运算符常用于 case 表达式的判别，所以又称为“case 等式运算符”。这 4 个等式运算符的优先级别是相同的。

（5）移位运算符，Verilog HDL 语言中规定了 2 种移位运算符。

<<(左移位运算符) 和 >>(右移位运算符)。

其使用方法如下：

```
a >> n  和  a << n
```

a 代表要进行移位的操作数，n 代表要移几位。这两种移位运算都用 0 来填补移出的空位。

（6）位拼接运算符，Verilog HDL 语言中规定位拼接运算符{}可以把两个或多个信号的某些位拼接起来进行运算操作。其使用方法如下：

```
{信号 1 的某几位，信号 2 的某几位，…，信号 n 的某几位}
```

即把某些信号的某些位详细地列出来，中间用逗号分开，最后用大括号括起来表示一个整体信号。

（7）缩减运算符，缩减运算符是单目运算符。缩减运算是对单个操作数进行或、与、非递推运算，最后的运算结果是一位的二进制数。缩减运算的具体运算过程是这样的：第一步先将操作数的第一位与第二位进行或、与、非运算，第二步将运算结果与第三位进行或、与、非运算，依次类推，直至最后一位。

1.1.5 Verilog HDL 程序结构

作为高级语言的一种，Verilog HDL 语言以模块集合的形式来描述数字系统，其中每一个模块都有接口部分用来描述与其他模块之间的连接。模块可以进行层次嵌套，将大型的数字电路设计分割成不同的小模块来实现特定的功能，最后通过顶层模块调用子模块来实现整体功能。模块可以从简单的门到整个大的系统，如一个计数器，一个存储子系统，一个微处理器等。

一个模块由模块名（module_name），端口列表（port_list），变量声明（reg、wire、parameter），端口声明（input、output、inout），行为描述语句（initial、always），赋值语句（continuous assignment），其他子模块（module instantiation、UDP instantiation），任务及函数（task、function）等组成，如下所示：

```
module module_name (port_list);          // module 模块名 ，端口列表
reg, wire, parameter,                    // 变量声明
input, output, inout,                    // 端口声明
……
task,function,                           // 任务及函数
……
initial statement                        // 行为描述
always statement
continuous assignment                    // 赋值语句
……
module instantiation                     // 模块例化
UDP instantiation                        // 用户自定义模块例化
```

```
.....
endmodule                                    // 模块结束
```

Verilog HDL 语言中每个模块都要进行端口定义，并说明输入、输出口，在 module 中输入用 input 表示，输出用 output 表示，根据端口类型不同，定为不同的数据类型，其中，reg 数据类型表示为寄存器的数据输出端，而 wire 数据类型表示为线的数据输出端，因此，当前 module 作为另一个模块的子模块时，当前模块的 input 不能表示成 reg 类型，但对于另一个 Top module 来说这个 input 并非输入，可表示为 reg 或 wire 类型。

下面是一个 NAND 与非模块的行为型描述，输出 out 是输入 in1 和 in2 相与后求反的结果。

```
module NAND(in1, in2, out);              // module 模块名，端口列表
        input in1, in2;                  // 端口声明
       output out;
    assign out = ~(in1 & in2);           // 连续赋值语句
 endmodule
```

in1、in2 和 out 端口指定为线型的。assign 连续赋值语句不间断地监视等式右端变量，一旦其发生变化，右端表达式被重新赋值后结果传给等式左端进行输出。连续赋值语句用来描述组合电路，一旦其输入发生变动，输出也随之而改变。

芯片设计工程中，工程师必须考虑如何验证设计的正确性，测试验证平台 Testbench 就是用于测试和验证设计的正确性。它给出模块的输入信号，观察模块的内部信号和输出信号，如图 1-2 所示，如果发现结果与预期的有所偏差，则要对设计模块进行修改。

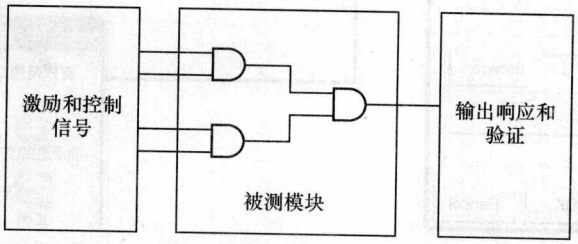

图 1-2　模块测试平台 Testbench

一个典型的测试模块 Testbench 包括模块名（module_test），变量声明（reg，wire），行为语句（initial，always），实例化被测模块（Testedmd）等，其中变量声明用于定义被测模块输入/输出变量类型，行为语句用于产生测试信号，常见的形式如下：

```
module module_test;                      // 模块名
reg ...;                                 // 变量声明
wire...;
initial                                  // 行为语句
begin ...   end
always #delay
begin ...   end
m Testedmd (.in1(ina), .in2(inb),.out1(outa),.out2(outb) );      // 被测模块的实例引用
endmodule
```

Testbench 模块中，实例化待测试模块，激励自动加载于测试模块。