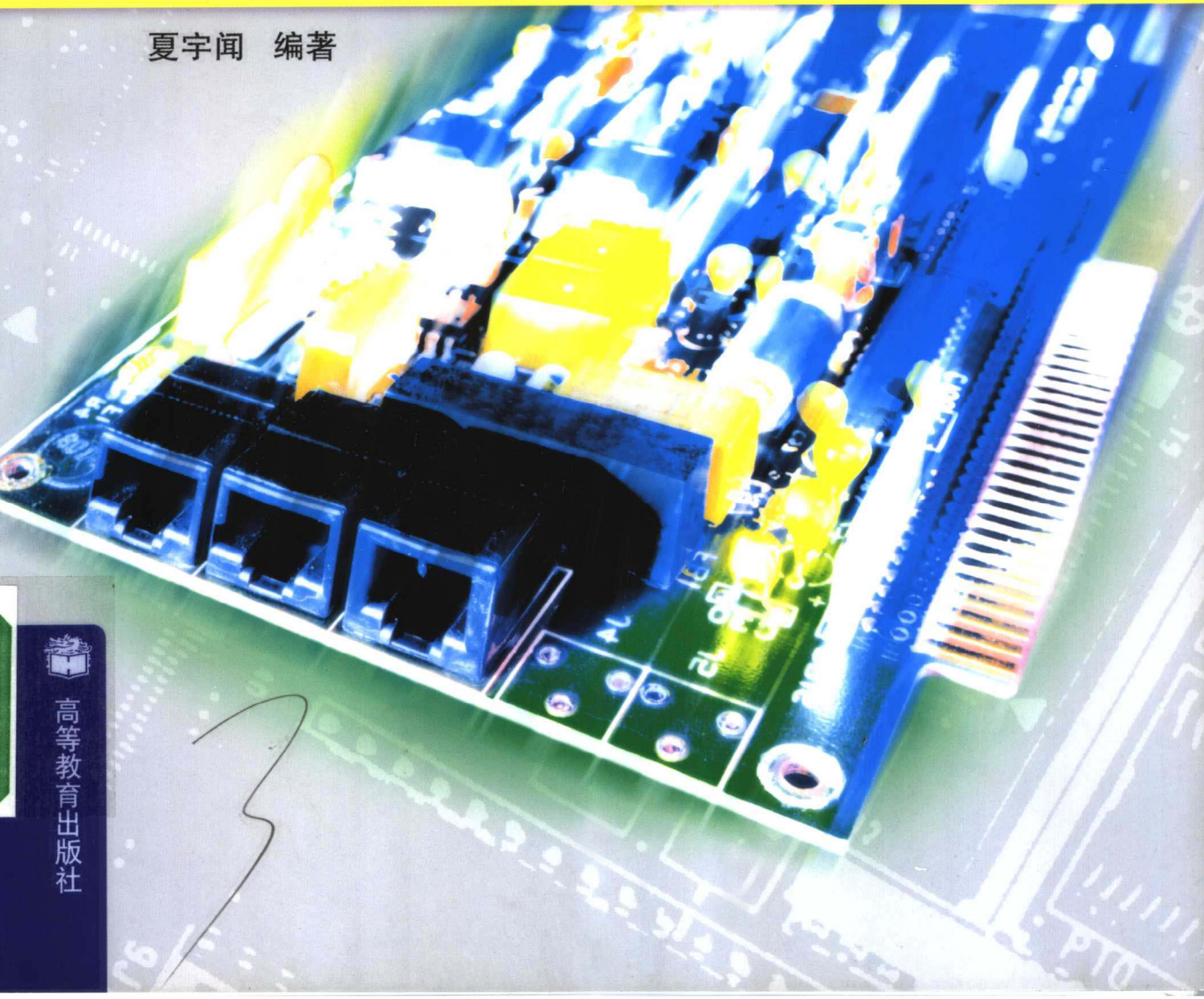
高 ◆ 等 ◆ 学 ◆ 校 ◆ 教 ◆ 材

Verilog HDL
实验练习与语法手册

夏宇闻 编著



高等教育出版社

高等学校教材

Verilog HDL 实验练习与语法手册

夏宇闻　编著

高等教育出版社

内容提要

本书是《数字系统设计——Verilog 实现》(夏宇闻编著)的配套辅导用书,为想真正掌握 Verilog HDL 设计方法的读者精心设计了丰富的上机练习和范例,并附有常用语法手册,能有效地帮助读者理解主教材中讲解的知识,并将其用到实践当中去。本书可以与主教材配套使用,也可单独作为高等学校电子信息、计算机等相关专业本科生和研究生学习数字电路设计的参考用书,也可供其他工程设计人员参考使用。

图书在版编目(CIP)数据

Verilog HDL 实验练习与语法手册/夏宇闻编著.
2 版. —北京: 高等教育出版社, 2006.1
ISBN 7-04-017199-6

Ⅰ.V... Ⅱ.夏... Ⅲ.①数字系统-系统设计-高等学校-教学参考资料②硬件描述语言,VHDL-程序设计-高等学校-教学参考资料 Ⅳ.①TP271②TP312

中国版本图书馆 CIP 数据核字(2005)第 155479 号

策划编辑 董建波 **责任编辑** 董建波
封面设计 张申申 **责任印制** 孔 源

出版发行	高等教育出版社	**购书热线**	010-58581118
社 址	北京市西城区德外大街 4 号	**免费咨询**	800-810-0598
邮政编码	100011	**网 址**	http://www.hep.edu.cn
总 机	010-58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
经 销	蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印 刷	北京铭成印刷有限公司	**畅想教育**	http://www.widedu.com
		版 次	2002 年 12 月第 1 版
开 本	787×1092 1/16		2006 年 1 月第 2 版
印 张	12.5	**印 次**	2006 年 1 月第 1 次印刷
字 数	250 000	**定 价**	18.60 元

物料号 17199-00

前　言

数字电路设计一直都被认为是既难教，又难学的课程。学好这门课程除了要很好地掌握数字电路的基本概念和理论外，最关键的是要有能力把理论真正应用到设计实践中去，因此学生自己认真地、反复地进行上机练习和思考对于掌握 Verilog 语言和设计技术具有极其重要的意义。

本书是《数字系统设计——Verilog 实现》（夏宇闻编著）的配套辅导用书，为想真正掌握 Verilog HDL 设计方法的读者精心设计了丰富的上机练习范例和思考题，并附有常用语法手册，能有效地帮助读者理解主教材中的知识点，并将其应用到设计实践中。

全书共提供了与主教材配套的 12 个典型的设计练习示范，可以基本满足初级 Verilog 学习者的需求。为了方便初学者，按字母顺序给出了常用 Verilog 语法的参考手册，读者可以随时查询不清楚的内容。

这套书是作者十多年来在数字系统 Verilog 设计教学方面的经验总结。本书可以与主教材配套使用，也可以作为工程设计人员的参考用书。

编者

2005 年 8 月

郑重声明

目　录

第一部分　设计示范与实验练习

在主教材《数字系统设计——Verilog 实现》(夏宇闻编著)学习的基础上，通过完成本书 12 个实验的练习,一定能逐步掌握 Verilog HDL 设计的要点。读者可以先理解设计示范模块中每一条语句的作用,进行功能仿真来加深理解,然后对示范模块进行综合,再分别进行综合后生成的逻辑网表和布线后生成的带布线延迟的门级器件网表的时序仿真,以加深印象和深入理解。在此基础上再独立完成每一阶段规定的练习。当 12 个实验练习做完后,便可以开始设计一些简单的逻辑电路和系统。最好有一到两个月的集中训练时间,有兴趣的读者很快就能设计相当复杂的数字逻辑系统。当然,复杂的数字系统的设计和验证,不但需要系统结构知识和丰富经验的积累,还需要了解更多的语法现象和掌握高级的 Verilog HDL 系统任务,以及与 C 语言模块接口(的方法即 PLI),这些已超出的本书的范围。有兴趣的读者可以阅读 Verilog 语法参考资料和有关文献,自己学习。

练习一　简单的组合逻辑设计

目的:(1) 掌握基本组合逻辑电路的实现方法;

(2) 初步了解两种基本组合逻辑电路的生成方法;

(3) 学习测试模块的编写;

(4) 通过综合和布局布线了解不同层次仿真的物理意义。

下面的模块描述一个可综合的数据比较器。从语句可以很容易看出它的功能是比较数据 a 与 b,如果两个数据相同,则给出结果 1,否则给出结果 0。在 Verilog HDL 中,描述组合逻辑时常使用 assign 结构。注意,equal = (a == b)? 1:0,这是一种在组合逻辑实现分支判断时常使用的格式。

模块源代码(方法一)如下:

```
//----------------------------文件名 compare.v ----------------------------
module compare(equal,a,b);
    input a,b;
    output equal;
        assign equal = (a==b)? 1 : 0;
```

```
        //a 等于 b 时,equal 输出为 1;a 不等于 b 时,equal 输出为 0
endmodule
```

模块源代码(方法二)如下:

```
module compare(equal,a,b);
    input a,b;
    output equal;
    reg equal;
        always @(a or b)
        if(a==b)            //a 等于 b 时,equal 输出为 1
          equal =1;
        else                //a 不等于 b 时,equal 输出为 0
          equal = 0;        //思考:如果不写 else 部分会产生什么逻辑
endmodule
```

测试模块用于检测模块设计得是否正确。它给出模块的输入信号,观察模块的内部信号和输出信号,如果发现结果与预期的有偏差,则需要对设计模块进行修改。

测试模块源代码(方法之一)如下:

```
`timescale 1ns/1ns          //定义时间单位
`include "./compare.v"      //包含模块文件。在有的仿真调试环境中并不需要此语句,
                            //而需要从调试环境的菜单中键入有关模块文件的路径和名称
module t;
    reg a,b;
    wire equal;
    initial                 //initial 常用于仿真时信号的给出
      begin
        a=0;
        b=0;
        #100  a=0; b=1;
        #100  a=1; b=1;
        #100  a=1; b=0;
        #100  a=0; b=0;
        #100  $stop;        //系统任务,暂停仿真以便观察仿真波形
      end
      compare m(.equal(equal),.a(a),.b(b));     //调用被测试模块 t.m
endmodule
```

综合就是把 compare.v 文件送到 synplify 或其他综合器处理,在选定实现器件和选取生

成 Verilog 网表的前提下，启动综合器的编译。综合器会自动生成一系列文件，向操作者报告综合的结果。其中生成的 Verilog Netlist 文件（扩展名为 .vm），表示自动生成的逻辑结构网表，仍然用 Verilog 语句表示，但比输入的源文件更具体，可以用测试模块调用它做同样的仿真。运行的结果更接近实际器件。

布局布线就是把综合后生成的另一种文件（EDIF），在布线工具控制下进行处理，启动布线工具的编译。布局布线工具会自动生成一系列文件，向操作者报告布局布线的结果。其中生成的 Verilog Netlist 文件（扩展名为 .vo）表示自动生成的具体基本门级结构和连接的延迟，仍然用 Verilog 基本部件结构语句和连接线的延迟参数的重新定义表示，库中的基本部件也更进一步具体化了，比综合后的扩展名为 .vm 的文件更具体。可以用同一个测试模块调用它做同样的仿真。运行结果与实际器件运行结果几乎完全一致。

仿真波形（部分）如图 1.1 所示。

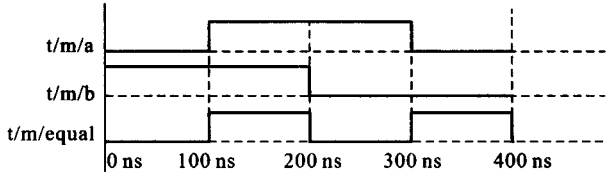


图 1.1　仿真波形

测试模块源代码（方法之二）如下：

```
`timescale 1ns/1ns          //定义时间单位
`include "./compare.v"      //包含模块文件。在有的仿真调试环境中并不需要此语句，
                            //而需要从调试环境的菜单中键入有关模块文件的路径和名称
module t;
    reg a,b;
    reg clock;
    wire equal;
    initial                 //initial 常用于仿真时信号的给出
      begin
        a=0;
        b=0;
        clock = 0;          //定义一个时钟变量
    end
    always #50 clock = ~clock;      //产生周期性的时钟
    always @ (posedge clock)        //在每次时钟正跳变沿时刻产生不同的 a 和 b
      begin
```

```
        a = {$random}%2;          //每次 a 是 0 还是 1 是随机的
        b = {$random}%2;          //每次 b 是 0 还是 1 是随机的
      end
    initial
      begin #100000 $stop; end    //系统任务,暂停仿真以便观察仿真波形
    compare m(.equal(equal),.a(a),.b(b));    //调用被测试模块 t.m
endmodule
```

[练习题] 设计一个字节(8 位)的比较器。

要求:比较两个字节的大小,如 a[7:0]大于 b[7:0]输出高电平,否则输出低电平,改写测试模型,使其能进行比较全面的测试。观测 RTL 级仿真、综合后门级仿真和布线后仿真有什么不同,并说明引起这些不同的原因。从文件系统中查阅自动生成的 compare. vm 、compare. vo 文件,和 compare. v 做比较,说出它们的不同点和相同点。

[思考题] 在测试方法二中,第二个 initial 块有什么用? 它与第一个 initial 块有什么关系? 如果在第二个 initial 块中没有写#100000 或者 $stop,仿真会如何进行? 比较两种测试方法,哪一种测试方法更全面?

练习二　简单分频时序逻辑电路的设计

目的:(1) 掌握最基本时序电路的实现方法;

(2) 学习时序电路测试模块的编写;

(3) 学习综合和不同层次的仿真。

在 Verilog HDL 中,相对于组合逻辑电路,可综合成具体电路结构的时序逻辑电路也有标准的表述方式。在可综合的 Verilog HDL 模型中,通常使用 always 块和 @(posedge clk)或 @(negedge clk)的结构来表述时序逻辑。下面是一个二分频器的可综合模型。

```
//---------------------------- 文件名:half_clk.v ----------------------------
 module half_clk(reset,clk_in,clk_out);
    input clk_in,reset;
    output clk_out;
    reg clk_out;
    always @(posedge clk_in)
      begin
        if(! reset) clk_out =0;
        else        clk_out = ~clk_out;
```

```
        end
endmodule
```

在 always 块中，被赋值的信号都必须定义为 reg 型，这是由时序逻辑电路的特点所决定的。对于 reg 型数据，如果未对它进行赋值，仿真工具会认为它是不定态。为了能正确地观察到仿真结果，并确定时序电路的起始相位，在可综合风格的模块中通常定义一个复位信号 reset，当 reset 为低电平时，对电路中的寄存器进行复位。

测试模块的源代码：

```
//----------------------------文件名 top.v ----------------------------
`timescale 1ns/100ps
`define clk_cycle 50
module top;
    reg clk,reset;
    wire clk_out;
    always #`clk_cycle clk = ~clk; //产生测试时钟
    initial
      begin
        clk = 0;
        reset = 1;
        #10 reset = 0;
        #110 reset = 1;
        #100000 $stop;
        end
        half_clk m0( .reset(reset),.clk_in(clk),.clk_out(clk_out));
endmodule
```

仿真波形如图 1.2 所示。

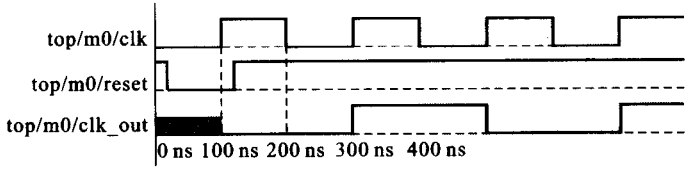


图 1.2　仿真波形

[练习题]　依然作 clk_in 的二分频 clk_out，要求输出时钟的相位与上例的输出正好反相。编写测试模块，给出仿真波形。改变输入时钟的频率，观测 RTL 级仿真、综合后门级仿

真和布线后仿真的不同，并写出报告。

[**思考题**]　如果没有 reset 信号，能否控制二分频 clk_out 信号的相位？只用 clk 时钟沿的触发（即不用二分频产生的时钟沿）如何直接产生四分频、八分频或者十六分频的时钟？如何只用 clk 时钟沿的触发（不用二分频产生的时钟沿）直接产生占空比不同的分频时钟？

练习三　利用条件语句实现计数分频时序电路

目的：(1) 掌握条件语句在简单时序模块设计中的使用；

(2) 学习在 Verilog 模块中应用计数器；

(3) 学习测试模块的编写、综合和不同层次的仿真。

与常用的高级程序语言一样，为了描述较为复杂的时序关系，Verilog HDL 提供了条件语句供分支判断时使用。在可综合风格的 Verilog HDL 模型中常用的条件语句有 if-else 和 case-endcase 两种结构，用法和 C 程序语言中类似。两者相比较，if-else 用于不很复杂的分支关系，实际编写可综合风格的模块特别是用状态机构成的模块时，更常用的是 case-endcase 风格的代码。下面给出的是有关 if-else 的范例，有关 case-endcase 结构的代码以后会经常用到。

下面给出的范例也是一个可综合风格的分频器，是将 10M 的时钟分频为 500K 的时钟。基本原理与二分频器是一样的，但是需要定义一个计数器，以便准确获得二十分频器。

模块源代码如下：

```
//------------------------------ fdivision.v ------------------------------
module fdivision(RESET,F10M,F500K);
    input F10M,RESET;
    output F500K;
    reg F500K;
    reg [7:0]j;
      always @(posedge F10M)
      if(! RESET)        //低电平复位
        begin
          F500K <= 0;
          j <= 0;
        end
      else
        begin
          if(j == 19)   //对计数器进行判断，以确定 F500K 信号是否反转
            begin
```

```
            j <= 0;
            F500K <= ~F500K;
          end
        else
          j <= j+1;
      end
  endmodule
```

测试模块源代码如下：

```
  //---------------------------- fdivision_Top.v ----------------------------

  `timescale 1ns/100ps
  `define clk_cycle 50

  module division_Top;

    reg F10M,RESET;

    wire F500K_clk;

    always #`clk_cycle F10M_clk = ~F10M_clk;

    initial
      begin
      RESET = 1;
      F10M = 0;
      #100 RESET = 0;
      #100 RESET = 1;
      #10000 $stop;
    end

    fdivision fdivision (.RESET(RESET),.F10M(F10M),.F500K(F500K_clk));

  endmodule
```

仿真波形如图 1.3 所示。

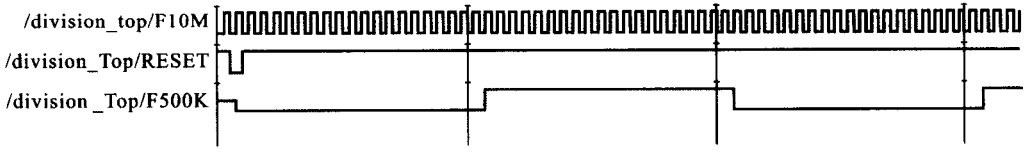


图 1.3　仿真波形

[**练习题**]　利用 10M 的时钟，设计一个单周期形状如图 1.4 所示的周期波形。

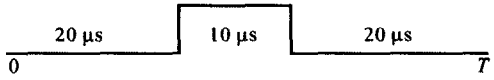


图 1.4　周期波形

练习四　阻塞赋值与非阻塞赋值的区别

目的：(1) 通过实验观察掌握阻塞赋值与非阻塞赋值的概念和区别；

(2) 了解非阻塞和阻塞赋值的不同使用场合；

(3) 学习测试模块的编写、综合和不同层次的仿真。

阻塞赋值与非阻塞赋值，在主教材中已经了解了它们之间在语法上的区别以及综合后所得到的电路结构上的区别。在 always 块中，阻塞赋值可以理解为赋值语句是顺序执行的，而非阻塞赋值可以理解为赋值语句是并发执行的。在实际的时序逻辑设计中，一般的情况下非阻塞赋值语句被更多地使用，有时为了在同一周期实现相互关联的操作，也使用了阻塞赋值语句(注意，在实现组合逻辑的 assign 结构中，必须采用阻塞赋值语句)。

下例通过分别采用阻塞赋值语句和非阻塞赋值语句的两个看上去非常相似的模块 blocking. v 和 non_blocking. v 来阐明两者之间的重要区别。

模块源代码如下：

```
//------------------------------ blocking.v ------------------------------
module blocking(clk,a,b,c);
    output [3:0] b,c;
    input  [3:0] a;
    input        clk;
    reg    [3:0] b,c;
    always @(posedge clk)
    begin
      b = a;
      c = b;
      $display("Blocking: a = %d, b = %d, c = %d ",a,b,c);
    end
endmodule
//------------------------------ non_blocking.v ------------------------------
```

```
module non_blocking(clk,a,b,c);
    output [3:0] b,c;
    input  [3:0] a;
    input        clk;
    reg    [3:0] b,c;

    always @(posedge clk)
    begin
      b <= a;
      c <= b;
      $display("Non_Blocking: a = %d, b = %d, c = %d ",a,b,c);
    end

endmodule
```

测试模块源代码如下：

```
//---------------------------- compareTop.v ----------------------------
`timescale 1ns/100ps
`include "./blocking.v"
`include "./non_blocking.v"

module compareTop;

    wire [3:0] b1,c1,b2,c2;
    reg  [3:0] a;
    reg        clk;

    initial
    begin
      clk = 0;
      forever #50 clk = ~clk; //思考:如果在本句后还有语句,能否执行? 为什么?
    end

    initial
    begin
      a = 4'h3;
      $display("_____________________________");
      #100 a = 4'h7;
      $display("_____________________________");
      #100 a = 4'hf;
      $display("_____________________________");
      #100 a = 4'ha;
```

```
        $display("_____ _____");
        #100 a = 4'h2;
        $display("_____");
        #100 $display("_____");
        $stop;
    end
    non_blocking non_blocking(clk,a,b2,c2);
    blocking blocking(clk,a,b1,c1);
endmodule
```

仿真波形(部分)如图 1.5 所示。

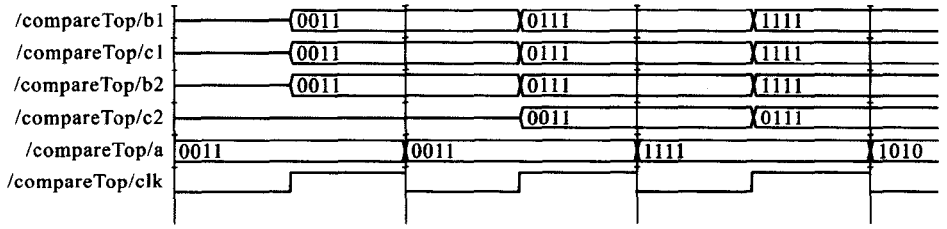


图 1.5　仿真波形

[**思考题**]　在 blocking 模块中按如下写法,仿真与综合的结果会有什么样的变化? 做出仿真波形,分析综合结果。

(1)

```
always @(posedge clk)
        begin
          c = b;
          b = a;
        end
```

(2)

```
always @(posedge clk) b=a;
always @(posedge clk) c=b;
```

练习五　用 always 块实现较复杂的组合逻辑电路

目的:(1) 掌握用 always 块实现较大组合逻辑电路的方法;

(2) 进一步了解 assign 与 always 两种组合电路实现方法的区别和注意点;