第3章 Verilog HDL简介

- 3.1 Verilog HDL概述
- 3.2 Verilog HDL基本要素
  - ① Verilog 的逻辑值
  - ② Verilog 的标识符
  - ③ Verilog 的常量及其表示方式
  - ④ Verilog 的数据类型
  - ⑤ Verilog 的运算符
  - ⑥ Verilog 的注释
- 3.3 Verilog 程序模块结构
- 3.4 Verilog一些说明

3.1 Verilog HDL概述

- Verilog HDL（Hardware Description Language）是在 C 语言的基础上发展起来的一种硬件 描述语言，具有灵活性高、易学易用等特点。
- Verilog 是一种硬件描述语言，以文本形式来描述数字系统硬件的结构和行为的语言，用它可以表示逻 辑电路图、逻辑表达式，还可以表示数字逻辑系统所完成的逻辑功能。
- 数字电路设计者利用这种语言，可以从顶层到底层逐层描述自己的设计思想，用一系列分层次的模块 来表示极其复杂的数字系统。然后利用电子设计自动化（EDA）工具，逐层进行仿真验证，再把其中需要 变为实际电路的模块组合，经过自动综合工具转换到门级电路网表。
- 用集成电路 ASIC 或 FPGA 自动布局布线工具，把网表转换为要实现的具体电路结构。

3.1 Verilog HDL概述

- 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。 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。
- Verilog HDL 语言于 1990 年被推向公众领域。Verilog 语言 于 1995 年成为 IEEE 标准，称为 IEEE Std1364-1995，也就是通常所说的 Verilog-95。
- 设计人员在使用 Verilog-95 的过程中发现了一些可改进之处。为了解决用户在使用此版本 Verilog 过程 中反映的问题，Verilog 进行了修正和扩展，这个扩展后的版本后来成为了电气电子工程师学会 Std1364-2001 标准，即通常所说的 Verilog-2001。
- Verilog-2001 是对 Verilog-95 的一个重大改进版本，它具备一些新的实用功能，例如敏感列表、多维数组、生成语句块、命名端口连接等。
- 目前，Verilog-2001 是 Verilog 的最主流 版本，被大多数商业电子设计自动化软件支持。

3.1 Verilog HDL概述

Verilog 和 VHDL 区别

- 这两种语言都是用于数字电路系统设计的硬件描述语言，而且都已经是 IEEE 的标准。 VHDL 1987 年 成为标准，而 Verilog 是 1995 年才成为标准的。
- 这是因为 VHDL 是美国军方组织开发的，而 Verilog 是由一 个公司的私有财产转化而来。
- 两者共同的特点：
  - 能形式化的抽象表示电路的行为和结构；
  - 支持逻辑设计中层次与范围的描述；
  - 可借用高级语言的精巧结构来简化电路行为和结构；
  - 支持电路描述由高层到低层的综合转换；
  - 硬件描述和实现工艺无关。

3.2 Verilog HDL基本要素
3.2.1 Verilog 的逻辑值

- 在逻辑电路中，逻辑状态有4种：
  - 逻辑 0：表示低电平，在逻辑电路中通常接GND；
  - 逻辑 1：表示高电平，在逻辑电路中通常接VCC；
  - 逻辑 x：表示逻辑状态不定，有可能是高电平，也有可能是低电平；
  - 逻辑 z：表示高阻态，是一个悬空状态。

3.2 Verilog HDL基本要素
3.2.2 Verilog 的标识符

标识符(identifier)用于定义模块、端口、寄存器，连线等元素的名称。

- Verilog HDL 中的标识符可以是任意一组字母、数字、$符号和_(下划线)符号的组合，但标识符的第一个字符必须是字母或者下划线；而且字符数不能多于1024。
- 标识符是区分大小写的。以下是标识符的几个例子：
  - Count
  - COUNT //与Count 不同。
  - R56_68
  - FIVE$

不建议大小写混合使用，普通内部信号建议全部小写，参数定义建议大写，信号命名最好体现信 号的含义。

3.2 Verilog HDL基本要素
3.2.2 Verilog 的标识符

- 书写规范建议：
  - ① 用有意义的有效的名字如 sum、cpu_addr 等。
  - ② 用下划线区分词语组合，如 cpu_addr
  - ③ 采用一些前缀或后缀，比如：
    - 时钟采用 clk 前缀：clk_50m，clk_cpu；
    - 低电平采用_n 后缀：enable_n；
  - ④ 统一缩写，如全局复位信号 rst。
  - ⑤ 同一信号在不同层次保持一致性，例如同一时钟信号必须在各模块保持一致。
  - ⑥ 自定义的标识符不能与保留字（关键词）同名。
  - ⑦ 参数统一采用大写，如定义参数使用 SIZE。

3.2 Verilog HDL基本要素
3.2.3 Verilog 的常量及其表示方式

- 在程序运行过程中，其值不能被改变的量称为常量。
- 整型常量的进制格式表示方式包括二进制、八进制、十进制和十六进制；一般常用的为二进制、十进制和十六 进制。
  - 二进制表示如下：4’b0101 表示 4 位二进制数字 0101；
  - 十进制表示如下：4’d3表示 4 位十进制数字 3（二进制 0011）；
  - 十六进制表示如下：4’ha 表示 4 位十六进制数字 a（二进制 1010）。
- 当代码中没有指定数字的位宽与进制时，默认为 32 位的十进制，比如 100，实际上表示的值为 32’d100。

3.2 Verilog HDL基本要素

3.2.4 Verilog 的数据类型

在 Verilog 语法中，主要有2类数据类型，即寄存器类型和线网类型。

- 寄存器类型表示一个抽象的数据存储单元，它只能在 always 语句和 initial 语句中被赋值，并且它的值从一个赋值到另一个赋值过程中被保存下来。
  - 如果是时序逻辑，即 always 语句带有时钟信号，则对应为寄存器；
  - 如果是组合逻辑，即 always 语句不带有时钟信号， 则对应为硬件连线；
  - 寄存器类型的缺省值是 x（未知状态，或状态不定）；
  - 寄存器数据类型有很多种，如 reg、integer、real 等，其中最常用的就是 reg 类型。
  - reg的使用方法如下：

```
reg [3:0] cnt; // cnt为4 位寄存器
reg  flag;      //1 位寄存器
```

3.2 Verilog HDL基本要素

3.2.4 Verilog 的数据类型

在 Verilog 语法中，主要有2类数据类型，即寄存器类型和线网类型。

- 线网类型表示结构化器件之间的物理连线。
- 由于线网类型代表的是物理连接线，因此它不存贮逻辑值。必须由器件驱动。通常由assign进行赋值。
- 如果没有驱动元件连接到线网，线网的缺省值为 z（高阻态）。
- 线网类型有很多种， 如 tri 和 wire 等，其中最常用的是 wire 类型，它的使用方法如下：

```
wire [7:0]  data;   // data为8 位数据
wire  data_en;      //1 位使能信号
```

- 在Verilog程序中，输入输出信号类型缺省时，默认为 wire 型。

3.2 Verilog HDL基本要素

3.2.5 Verilog 的运算符

1.算术运算符

算术运算符，就是数学运算里面的加、减、乘、除等。

运算符	用法	描述
+	a+b	a加b
-	a-b	a减b
*	a*b	a乘b
/	a/b	a除b
%	a%b	a模除b

3.2 Verilog HDL基本要素

3.2.5 Verilog 的运算符

2. 关系运算符

- 关系运算符用于一些条件判断。在进行关系运算时，如果声明的关系是假的，则返回 值是 0；如果声明的关系是真的，则返回值是 1。
- 关系运算符的优 先级别低于算术运算符的优先级别。

运算符	用法	描述
>	a>b	a大于b
<	a<b	a小于b
>=	a>=b	a大于等于b
<=	a<=b	a小于等于b
==	a==b	a等于b
!=	a!=b	a不等于b

3.2 Verilog HDL基本要素

3.2.5 Verilog 的运算符

3. 逻辑运算符

- 逻辑运算符用于实现逻辑运算。

运算符	用法	描述
!	!a	a的非
&&	a&&b	a与b
||	a||b	a或b

3.2 Verilog HDL基本要素
3.2.5 Verilog 的运算符

4. 条件运算符

- 条件运算符是从两个输入中选择一个作为输出，功能等同于 **always** 中的 **if-else** 语句。

运算符	用法	描述
?:	a ? b : c	如果a为真，选择b；否则选择c

3.2 Verilog HDL基本要素

3.2.5 Verilog 的运算符

5. 位运算符

- 位运算符直接对应数字逻辑中的与、或、非门等逻辑门。

运算符	用法	描述
~	~a	将a的每一位取反
&	a&b	将a和b的相同位按位与
|	a|b	将a和b的相同位按位或
^	a^b	将a和b的相同位按位异或

3.2 Verilog HDL基本要素

3.2.5 Verilog 的运算符

6. 移位运算符

- 移位运算符包括左移位运算符和右移位运算符，这两种移位运算符都用 0 来填补移出的空位。

如果 a 是 00000011，执行a<<2后，a就是 00001100。

运算符	用法	描述
<<	a<<b	将a左移b位
>>	a>>b	将a右移b位

3.2 Verilog HDL基本要素

3.2.5 Verilog 的运算符

7. 拼接运算符

拼接运算符 可以把两个或多个信号的某些位拼接起来。

运算符	用法	描述
{ }	{a,b}	将a和b拼接起来

3.2 Verilog HDL基本要素

3.2.5 Verilog 的运算符

7. 拼接运算符

拼接运算符 可以把两个或多个信号的某些位拼接起来。

运算符	用法	描述
{ }	{a,b}	将a和b拼接起来

3.2 Verilog HDL基本要素

3.2.6 Verilog 的注释

在Verilog 中有两种注释的方式。

- 多行注释：以“/*”符号开始，到“*/”结束；在两个符号之间的语句都是注释语句。
- 单行注释：以//开始到本行结束都属于注释语句，不允许续行。

3.3 Verilog HDL程序模块结构

Verilog HDL程序模块描述方式：

① 行为(Behavioural)描述:是以算法形式对系统模型、功能的描述，与具体硬件结构无关。

② 结构(Structural)描述：在多层次的设计中，高层次的设计模块调用低层次的设计模块，构成层次化的设计。

③ 数据流(Data Flow)描述：也称寄存器传输级(RTL)描述，是以寄存器为特征，在寄存器之间插入组合逻辑电路，即以描述数据流的流向为特征。

3.3 Verilog HDL程序模块结构

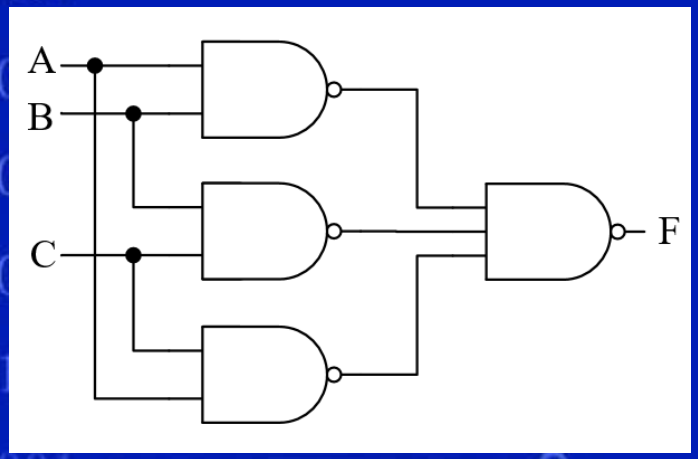
3.3.1 Verilog 程序模块设计举例

模块端口定义

模块内容

- I/O说明
- 信号类型说明
- 功能描述

【例3-3-1】用Verilog 实现图所示的3输入表决电路。当3人中有2人或者超过2人同意，则表决结果为通过；否则表决结果不通过。

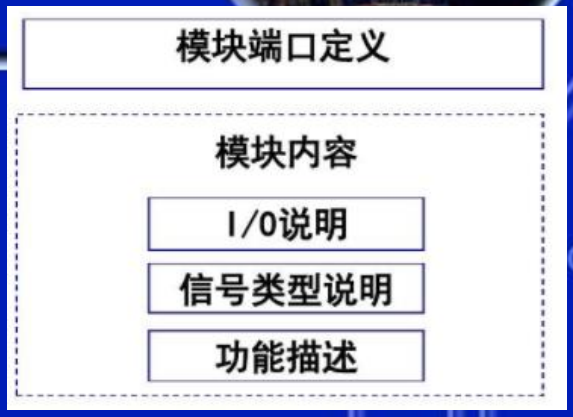


【例3-3-1】3输入表决逻辑电路

```
//3输入表决电路模块
//模块名为voter
//模块声明在module与endmodule之间
//端口列表在()中
module voter (A, B, C, F);
//端口定义
input A, B, C;
output F;
//信号类型说明
wire h1,h2,h3;
//逻辑功能定义
assign h1 = ~(A&B);
assign h2 = ~(A&C);
assign h3 = ~(B&C);
assign F = ~(h1&h2&h3);
endmodule
```

3.3 Verilog HDL程序模块结构

3.3.1 Verilog 程序模块设计举例



Verilog HDL程序模块包含：模块 定义、端口声明、功能描述。

① Verilog HDL模块以关键字module开始，以关键字endmodule结束。

② 模块端口定义

- 模块名在关键字module之后，其定义须符合Verilog HDL标准。
- 端口列表在关键字module之后的括号内，给出模块与外界相互联系的I/O接口信号，I/O信号之间采用逗号分隔。

③ 模块内容

- I/O说明使用关键字input、output以及inout说明端口是输入、输出，还是双向信号。
- 信号类型声明：在声明模块中使用的线网(wire)或者寄存器(reg)类型的变量。
- 功能描述包含：连续赋值语句(assign)，模块实例以及过程赋值语句(always块)等。

```
//3输入表决电路模块
//模块名为voter
//模块声明在module与endmodule之间
//端口列表在()中
module voter (A, B, C, F);
//端口定义
input A, B, C;
output F;
//信号类型说明
wire h1,h2,h3;
//逻辑功能定义
assign h1 = ~(A&B);
assign h2 = ~(A&C);
assign h3 = ~(B&C);
assign F = ~(h1&h2&h3);
endmodule
```

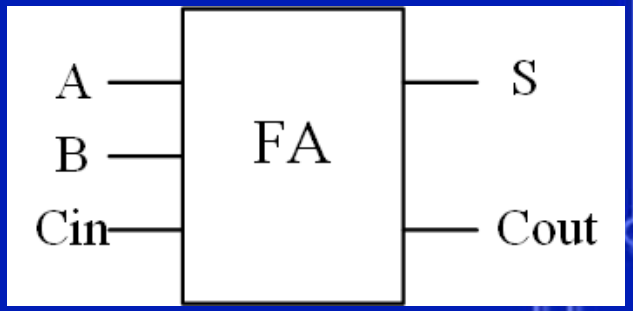
3.3 Verilog HDL程序模块结构

3.3.1 Verilog 程序模块设计举例

模块说明方式

方式1：

```
//全加器
module full_adder (A, B, Cin, S,Cout);
  input A,B,Cin; //输入端口
  output S,Cout; //输出端口
  reg S, Cout;
```



全加器

方式2：

```
//全加器
module full_adder (A, B, Cin, S,Cout);
  input A,B,Cin; //输入端口
  output reg S,Cout; //输出端口
```

方式3：建议使用

```
//全加器
module full_adder (A, B, Cin, S,Cout);
  input A; //输入端口
  input B; //输入端口
  input Cin; //低位进位,输入端口
  output reg S; //本位和，输出端口
  output reg Cout; //向高位进位，输出端口
```

3.3 Verilog HDL程序模块结构

3.3.2 Verilog 行为描述

行为描述抽象级别最高，概括能力最强，只描述数据逻辑，不关注电路实现，电路可控性差，综合效率低 。

【例3-3-2】用Verilog行为描述实现1位全加器。

```
//全加器
module full_adder (A, B, Cin, S,Cout);
  input A; //输入端口
  input B; //输入端口
  input Cin; //低位进位,输入端口
  output S; //本位和，输出端口
  output Cout; //向高位进位，输出端口
  reg S, Cout;
  always @(A, B, Cin) begin
    case ({A, B, Cin})
      3'b000: {S, Cout} = 2'b00;
      3'b001: {S, Cout} = 2'b10;
      3'b010: {S, Cout} = 2'b10;
      3'b011: {S, Cout} = 2'b01;
      3'b100: {S, Cout} = 2'b10;
      3'b101: {S, Cout} = 2'b01;
      3'b110: {S, Cout} = 2'b01;
      3'b111: {S, Cout} = 2'b11;
    endcase
  end
endmodule
```

3.3 Verilog HDL程序模块结构

3.3.3 Verilog结构描述

结构描述常用于层次化模块间的调用、以及IP核的例化等。

【例3-3-3】用Verilog结构描述实现1位全加器。

```
//功能描述：全加器，由两个半加器组成
module full_adder(A,B,Cin,S,Cout);
  input       A;                //输入端口
  input       B;                //输入端口
  input       Cin;              //低位进位,输入端口
  output      S;                //本位和，输出端口
  output      Cout;             //向高位进位，输出端口
  wire                          C0;             //线网说明
  wire                          A0;             //线网说明
  wire                          C1;             //线网说明
  assign Cout = C0 | C1; //全加器输出
//第1个半加器例化
half_adder half_adder_0(
                .A(A),          //端口A
                .B(B),          //端口B
                .S(A0),         //端口S
                .C(C0));        //端口C
//第2个半加器例化
half_adder half_adder_1(
                .A(A0),         //端口A
                .B(Cin),        //端口B
                .S(S),          //端口S
                .C(C1));        //端口C
endmodule
```

```
//半加器
module half_adder(A,B,S,C);
  input  A;     //输入端口
  input  B;     //输入端口
  output S;     //和输出端口
  output C;     //进位输出端口
//用门电路实现半加器
  assign S = A ^ B;          //直接赋值，异或
  assign C = A & B;          //直接赋值，与
endmodule
```

3.3 Verilog HDL程序模块结构

3.3.4 Verilog数据流描述

数据流描述是以寄存器为特征，在寄存器之间插入组合逻辑电路，可以用于对设计进行综合。

【例3-3-3】用Verilog数据流描述实现1位全加器。

```
//功能描述：数据流描述的全加器
module full_adder(A,B,Cin,S,Cout);
  input   A;    //输入端口
  input   B;    //输入端口
  input   Cin;  //低位进位,输入端口
  output  S;    //本位和，输出端口
  output  Cout; //向高位进位，输出端口
  wire    h;
    assign   h = A^B;
    assign   S = Cin^h;
    assign   Cout = (A&B)|(h&Cin);
endmodule
```

3.3 Verilog HDL程序模块结构

3.3.5 Verilog仿真

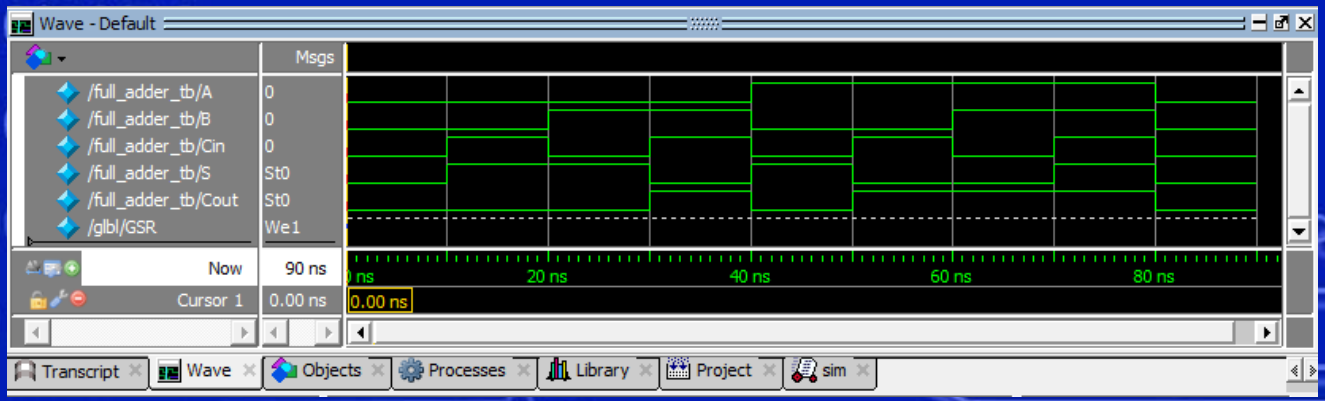
行为描述可用于对设计进行仿真，生成对设计的测试向量。

【例3-3-4】用Verilog行为描述实现1位全加器的仿真。

timescale 定义仿真时间单位与精度，1ns是时间单位，即在仿真中用#10表示延迟10ns。100ps表示时间精度，如写“#3.5547968525 A = 1;”，时间精度有100ps(即在3.5ns时赋值语句生效）。

输入为reg类型，输出为wire类型。

```
`timescale 1ns/100ps
module full_adder_tb;
// Inputs
  reg A,B,Cin;
// Outputs
  wire S,Cout;
// Instantiate UUT
full_adder DUT(
  .A(A),
  .B(B),
  .Cin(Cin),
  .S(S),
  .Cout(Cout)
);
initial begin
        A=0;B=0;Cin=0;
  #10; A=0;B=0;Cin=1;
  #10; A=0;B=1;Cin=0;
  #10; A=0;B=1;Cin=1;
  #10; A=1;B=0;Cin=0;
  #10; A=1;B=0;Cin=1;
  #10; A=1;B=1;Cin=0;
  #10; A=1;B=1;Cin=1;
  #10; A=0;B=0;Cin=0;
  #10; $stop;
  end
endmodule
```



3.4 Verilog HDL一些说明

输入输出定义

- 一行只定义一个信号；
- 信号全部对齐；
- 同一组的信号放在一起。

parameter 定义

- **module 中的 parameter 声明，不建议随处乱放；**
- **将 parameter 定义放在紧跟着 module 的输入输出定义之后；**
- **parameter 等常量命名全部使用大写。**

3.4 Verilog HDL一些说明

define定义

- define是global constants
- 不同模块都可使用,放在模块之前

```
// Configure the maximum page size
`ifdef M14K_SMARTMIPS
`define  M14K_PAGESIZE16M    1
`else
`define  M14K_PAGESIZE256M   1
`endif
```

parameter 定义

- parameter 仅在模块内有效
- parameter 是local constants

```
module moore11111(
 input clk,
 input clr,
 input x,
 output reg z,
 output reg [2:0] current_state);
 reg[2:0] next_state;
 parameter [2:0] zero = 3'b000,  one = 3'b001,  two = 3'b010,
                 three = 3'b011, four = 3'b100, five = 3'b101;
```

3.4 Verilog HDL说明

wire/reg 定义

① 将 reg 与 wire 的定义放在紧跟着 parameter 之后；
② 建议具有相同功能的信号集中放在一起；
③ 信号需要对齐，reg 和位宽需要空 2 格，位宽和信号名字至少空2格；
④ 位宽使用降序描述，[1:0]；
⑤ 时钟使用前缀 clk，复位使用后缀 rst；
⑥ 不能使用 Verilog 关键字作为信号名字；
⑦ 一行只定义一个信号。

```
//阻塞赋值
module blocking (
   input   wire          clk,    //系统时钟
   input   wire          rst_n,  //系统复位，低电平有效
   output  reg  [1:0]    a, //寄存器类型，输出信号
   output  reg  [1:0]    b, //寄存器类型，输出信号
   output  reg  [1:0]    c  //寄存器类型，输出信号
   );
always@(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
                  a = 1;
                  b = 2;
                  c = 3;
                  end
                  else begin
                  a = 0;
                  b = a;
                  c = b;
                  end
   end
endmodule
```

3.4 Verilog HDL说明

信号命名

① 信号命名需要体现其意义，如 ram_wr 代表 RAM 读写使能；

② 用“_”隔开信号，如 sys_clk；

③ 信号不要使用大写，也不要使用大小写混合，建议全部使用小写；

④ 模块名字使用小写；

⑤ 低电平有效的信号，使用_n 作为信号后缀。

3.4 Verilog HDL说明

空格和 TAB

- 由于不同的编辑器对TAB 编译不一致，建议不使用 TAB，全部使用空格。

注释

- 添加注释可以增加代码的可读性。
- 注释描述需要清晰、简洁；
- 注释描述不要废话，冗余；
- 注释描述用“//”；
- 注释描述需要对齐；
- 核心代码和信号定义之间需要增加注释。

3.4 Verilog HDL说明

注意事项

① 代码写的越简单越好；

② 不使用 repeat 等循环语句；

③ 在数据流描述的Verilog中，不使用 initial 语句，在仿真的Verilog中可以使用；

④ 避免产生 Latch 锁存器，比如组合逻辑里面的 if 不带 else 分支、case 缺少 default 语句；

⑤ 避免使用太复杂和少见的语法，可能造成语法综合器优化力度较低。

3.4 Verilog HDL说明

阻塞赋值（Blocking）

- 阻塞赋值。在一个 always 块中，后面的语句会受到前语句的影响，一条阻塞赋值语句如果没有执行结束，那么该语句后面的语句就不能被执行，即被“阻塞”。也 就是说 always 块内的语句是一种顺序关系。
- 符号“=”用于阻塞的赋值（如b = a），阻塞赋值“=”在 begin 和 end 之间的语句是顺序执行，属于串行语句。

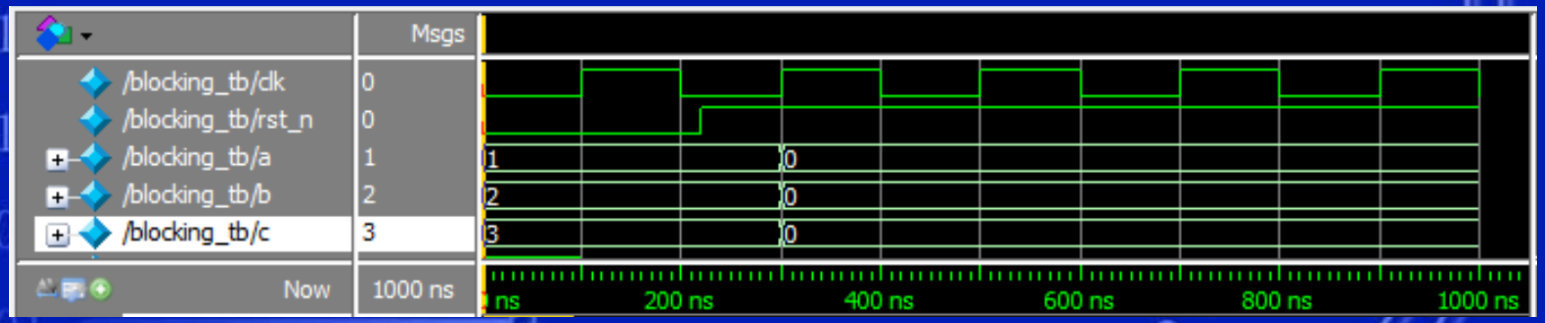
3.4 Verilog HDL说明

阻塞赋值（Blocking）

【例3-4-1】阻塞赋值举例。

阻塞赋值Verilog源码可知，在复位时，a=1，b=2，c=3；而在没有复位时，a 的值清零， 同时将 a 的值赋值给 b，b 的值赋值给 c，Modelsim仿真结果见波形图。

```
//阻塞赋值
module blocking (clk,rst_n,a,b,c);
input wire clk,rst_n;
output reg [1:0] a,b,c;
always@(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
                a = 1;
                b = 2;
                c = 3;
            end
            else begin
                a = 0;
                b = a;
                c = b;
            end
    end
endmodule
```



3.4 Verilog HDL说明

非阻塞赋值（Non-Blocking）

- 符号“<=”用于非阻塞赋值（如:b <= a;），非阻塞赋值是由时钟节拍决定，在时钟上升到来时，执行 赋值语句右边，然后将 begin与end 之间的所有赋值语句同时赋值到赋值语句的左边。
- 注意：在begin与end 之 间的所有语句，一起执行；且一个时钟只执行一次，属于并行执行语句。

3.4 Verilog HDL说明

非阻塞赋值（Non-Blocking）

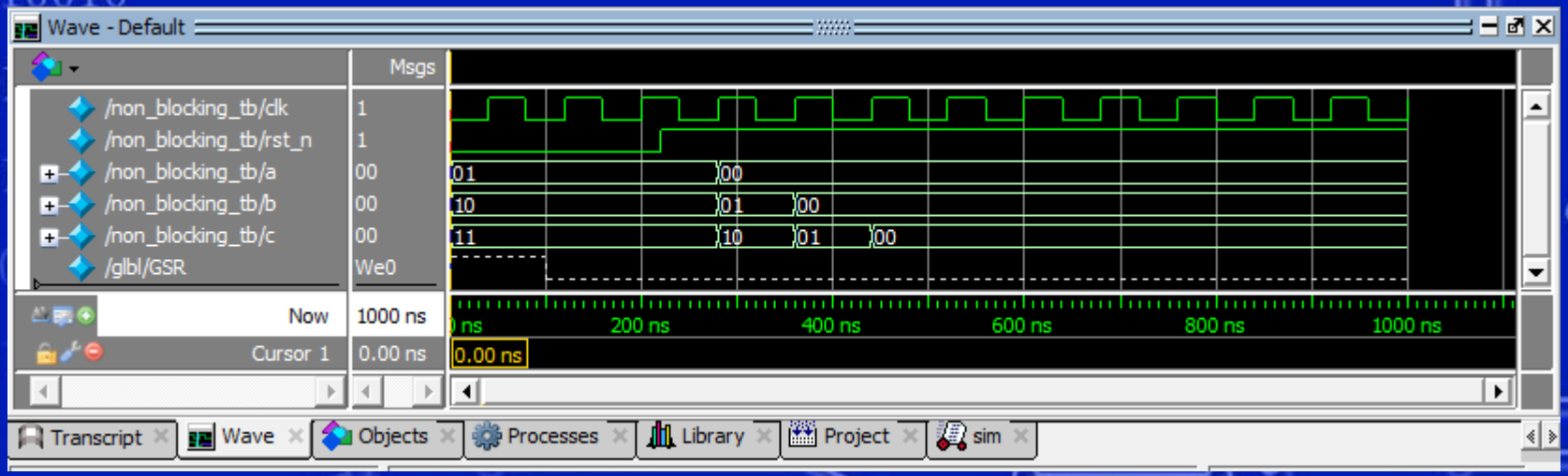
【例3-4-2】非阻塞赋值举例。

由非阻塞赋值Verilog源码可知，在复位时，a=1，b=2，c=3；而在没有复位时，a 的值清零，同时将 a 的值赋值给 b，b 的值赋值给 c，Modelsim仿真结果见波形图。

由波形图可知，在begin与end 之 间的所有语句，并行执行；且一个时钟只执行一次。

```
//非阻塞赋值
module non_blocking(clk,rst_n,a,b,c);
input wire clk,rst_n;
output reg [1:0] a,b,c;

always@(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
                  a <= 1;
                  b <= 2;
                  c <= 3;
               end
               else begin
                  a <= 0;
                  b <= a;
                  c <= b;
               end
   end
endmodule
```



3.4 Verilog HDL说明

assign 语句和 always 语句

- assign 语句和 always 语句是 Verilog 中的两个基本语句。
- assign 语句使用时不能带时钟。
- always 语句可以带时钟，也可以不带时钟。
- 在 always 不带时钟时，逻辑功能和 assign 完全一致，都是只产生组合逻辑。
- 比较简单的组合逻辑推荐使用 assign 语句，比较复杂的组合逻辑推荐使用 always 语句。

3.4 Verilog HDL说明

assign 块描述方式

① assign 逻辑不能太复杂，否则易读性不好；

② assign 前面需要有注释；

③ 组合逻辑使用阻塞赋值。

3.4 Verilog HDL说明

带时钟和不带时钟的 always

- always 语句可以带时钟，也可以不带时钟。
- 在 always 不带时钟时，逻辑功能和 assign 完全一致，产生的信号定义是 reg 类型，但是组合逻辑。
- 在 always 带时钟信号时，产生寄存器。

3.4 Verilog HDL说明

always 块描述方式

① if 需要空四格；

② 一个 always 需要配一个 begin 和 end；

③ always 前面需要有注释；

④ beign 建议和 always 放在同一行；

⑤ 一个 always 和下一个 always 空一行即可，不要空多行；

⑥ 时钟复位触发描述使用 posedge sys_clk 和 negedge sys_rst_n；

⑦ 一个 always 块只包含一个时钟和复位；

⑧ 时序逻辑使用非阻塞赋值。

3.4 Verilog HDL说明

锁存器和触发器

- 锁存器和寄存器都是基本存储单元，锁存 器是电平触发的存储器，寄存器是边沿触发的存储器。
- 锁存器latch 的主要危害是可能会产生毛刺（glitch）。
- 出现 latch 的原因：在不带时钟的 always 语句中，if 或者 case 语句不完整的描述；如 if 缺少 else 分支， case 缺少 default 分支，导致代码在综合过程中出现了 latch。解决办法就是 if 必须带 else 分支，case 必须带 default 分支。在带时钟的 always语句 中，即使if 或者 case 语句不完整描述，也不会产生 latch。