第3章 硬件描述语言Verilog HDL基础

3.1 硬件描述语言概述

硬件描述语言是一种用形式化方法来描述数字逻辑电路和系统的语言。利用这种语言,数字电路系统的设计可以从上层到下层、从抽象到具体逐层描述,用一系列分层次的模块实现复杂的数字系统。

硬件描述语言可以在结构级(也称为逻辑门级)、行为级、寄存器传输级(Register Transfer Level,RTL)三种层次上对电路进行描述。通过逻辑综合,后两种层次的硬件描述语言源文件可以被转换到低抽象级别的门级描述。

硬件描述语言有系统仿真和硬件实现两种用途。如果仅用于仿真,所有的语法和编程方法都可以使用。如果用于硬件实现就必须保证程序"可综合"。也就是说,所有的硬件描述语言描述都可以用于仿真,但不是所有的硬件描述语言描述都能用于硬件实现。

目前,常用的硬件描述语言是 VHDL 和 Verilog HDL。VHDL 是美国军方组织开发的,在 1987 年成为 IEEE 标准,1993 年 IEEE 对 VHDL 进行了修订,从更高的抽象层次和系统描述能力上扩展 VHDL 的内容,推出了 IEEE 标准的 1076-1993 版本(简称 93 版)。

Verilog HDL 是 1983 年由 Gateway Design Automation 公司设计的,该公司 1990 年被 Cadence 收购。1990 年 Open Verilog International 将 Verilog HDL 开放。1995 年,Verilog HDL 成为 IEEE 1364—1995 标准(简称 Verilog-95);2001 年 Verilog-95 做了重大改进,称为 Verilog-2001,这是主流版本,被大多数电子设计自动化软件支持。

Verilog HDL 和 VHDL 具有以下共同特点:

(1) 能形式化地抽象表示逻辑电路的行为和结构;

(2) 逻辑电路描述由高层到低层的综合;

(3) 硬件描述和硬件实现与工艺无关。

Verilog HDL 可以进行系统级、算法级、RTL 级、门级、开关级的逻辑设计,完成数字逻辑系统的仿真验证、时序分析、逻辑综合。一个复杂的数字逻辑系统的 Verilog HDL 模型由若干 Verilog HDL 模块构成,每个模块又可以由若干子模块构成。

3.2 Verilog 语言基本概念

本节介绍 Verilog HDL 的基本要素,包括标识符、数值集合、数据类型和程序基本结构等。

3.2.1 Verilog 程序基本结构

下面以一位全加器为例说明 Verilog 程序的结构特点。一位全加器的逻辑图如图 3-2-1 所示,由 2 个半加器和 1 个或门构成。其中,半加器的逻辑图如图 3-2-2 所示,由 1 个与门和 1 个异或门构成。

用 Verilog 语言对一位全加器的描述如图 3-2-3 所示,其中半加器子模块的 Verilog 语言描述如图 3-2-4 所示。

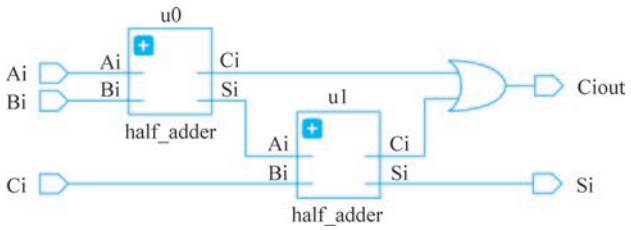

图 3-2-1　全加器逻辑图

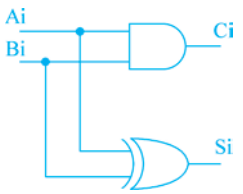

图 3-2-2　半加器逻辑图

```
//功能描述：全加器，由两个半加器组成
module full_adder(
  input Ai,   //输入端口
  input Bi,   //输入端口
  input Ci,   //低位进位,输入端口
  outputSi,   //本位和，输出端口
  outputCiout);//向高位进位，输出端口
//线网说明
wire    C0,A0,C1;
assign Ciout = C0|C1; //全加器输出
//第1个半加器例化
half_adder u0(
    .Ai(Ai),  //端口A
    .Bi(Bi),  //端口B
    .Si(A0),  //端口S
    .Ci(C0)); //端口C
//第2个半加器例化
half_adder u1(
    .Ai(A0),  //端口A
    .Bi(Ci),  //端口B
    .Si(Si),  //端口S
    .Ci(C1)); //端口C
endmodule
```

图 3-2-3　全加器 full_adder. v

```
//半加器
module half_adder(//模块名为
half_adder
  input  Ai, //输入端口
  input  Bi, //输入端口
  output Si, //输出端口和
  output Ci  //输出端口进位
  );
//用门电路实现半加器
  assign Si = Ai ^ Bi;//异或
  assign Ci = Ai & Bi;//与
endmodule    //模块结束关键字
```

图 3-2-4　半加器 half_adder. v

在图 3-2-3 全加器 Verilog 程序中，描述了 1 个名为 full_adder 的模块，模块是 Verilog 基本描述单位，用于描述逻辑电路的功能或结构，以及与子模块端口关系。full_adder 模块有 3 个输入端口和 2 个输出端口。全加器为顶层模块，半加器为底层模块，通过模块例化调用底层模块。

模块以关键词 module 开始，以关键词 endmodule 结尾，模块中包括端口定义、I/O 说明、内部信号声明和功能定义四个部分。

3.2.2　Verilog 语言要素

在 Verilog 程序中有各种符号，如标识符、操作符、字符串、注释等。

1. 标识符

标识符用于定义模块、端口、实例等名称。标识符可以是任意一组字母、数字、$ 符号和_(下画线)符号的组合，标识符的第一个字符必须是字母或者下画线，字符数不能多于 1024 个。标识符是区分大小写的。标识符的几个例子如下：

```
count     COUNT     clk_100mhz
```

由于标识符区分大小写,count 和 COUNT 是不同的。

2. 关键词

Verilog 语言内部已经使用的词称为关键词或保留字,关键词不能随便使用。需注意关键词都是小写,如 always 是关键词,ALWAYS 不是关键词。

3. 注释

Verilog 程序的注释有多行注释和单行注释两种。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。为了增强程序的可读性和可维护性,应当加上必要的注释。

4. 逻辑值

在 Verilog 程序中有以下四种逻辑值:

(1) 逻辑 0:表示低电平,在逻辑电路中通常接地。

(2) 逻辑 1:表示高电平,在逻辑电路中通常接电源。

(3) 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。

(4) 逻辑 z:表示高阻态,是一个悬空状态。

5. 常量

在程序运行过程中,其值不能被改变的量称为常量。

1) 整型常量

在 Verilog 程序中,整型常量有简单的十进制数格式和基数格式两种书写方式。

简单的十进制格式为带可选的"+"或"-"操作符的数字。例如:

```
32   十进制数 32
-15   十进制数 -15
```

基数格式:<位宽>'<进制符号><数字>。

进制符号:o 或 O 表示八进制,b 或 B 表示二进制,d 或 D 表示十进制,h 或 H 表示十六进制。例如:

```
7'b1000000                    //7 位二进制数
8'h2a                         //8 位十六进制数
```

下画线可以用在整数或实数中,用来提高易读性;但是下画线符号不能放在字首。例如:

```
16'b1010_1011_1111_1010       //合法格式
8'b_0011_1010                 //非法格式
```

2) 字符串型常量

字符串是双引号内的字符序列。例如:"Hello World",该字符串包含 11 个 ASCII 符号,其中 2 组单词 10 个符号,空格为 1 个符号。

3) 参数声明

参数是特殊的常量,经常用于定义延迟时间和变量宽度,参数声明语句用在 module 内部。格式如下:

```
parameter 参数名 1 = 表达式,参数名 2 = 表达式,…,参数名 n = 表达式;
```

例如：parameter byte_size＝8；

为了源代码的可读性和可移植性，在程序中不要直接写特定数值，尽可能采用parameter语句定义常数。

4）数据类型

在Verilog程序中数据类型有很多种，常用的有以下两种类型：

（1）线网类型：用于对结构化器件之间的物理连线的建模，如器件的引脚、内部器件输出等。通常由assign进行赋值，如assign F＝B&C；

当线网类型信号没有被驱动时，默认值为z(高阻)。

在Verilog程序模块中，输入输出信号类型默认定义为线网类型。

线网类型格式：

```
wire [n－1:0] 数据名 1,数据名 2,…;
```

其中：[n－1:0]是位宽。

（2）寄存器类型：表示一个抽象的数据存储单元，只能在always语句和initial语句中被赋值。寄存器数据类型有很多种，如reg、integer、real等，最常用的是reg类型。但是必须注意，reg类型的变量，不一定是寄存器；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。例如：reg[3:0] cnt；

3.2.3 Verilog运算符

在Verilog程序中运算符按其功能可分为算术运算符、赋值运算符、关系运算符、逻辑运算符、条件运算符、位运算符、移位运算符、拼接运算符等。按其所带操作数的个数运算符可分为单目运算符、二目运算符、三目运算符。

1. 算术运算符

在Verilog程序中，算术运算符又称为二进制运算符，有下面几种：

（1）＋：加法运算符，或正值运算符，如a＋b，＋3。

（2）－：减法运算符，或负值运算符，如a－3，－3。

（3）＊：乘法运算符，如a＊3。

（4）/：除法运算符，如5/3。

（5）％：模运算符，也称为求余运算符，要求％两侧均为整型数据，如5％3的值为2。

在进行整数除法运算时，结果值略去小数部分，只取整数部分。在进行取模运算时，结果值的符号位采用模运算式里第一个操作数的符号位。

2. 关系运算符

关系运算符如下：

a＜b	a小于b
a＞b	a大于b

a <=b　　a 小于或等于 b
a>=b　　a 大于或等于 b
a==b　　a 逻辑相等 b
a!=b　　a 逻辑不等 b

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

3. 逻辑运算符

逻辑运算符有 &&(逻辑与)、||(逻辑或)和!(逻辑非)。

若两个操作数为逻辑 0 或 1,则运算结果也为逻辑 0 或 1。例如,1'b0&&1'b1 运算结果为 0。若操作数是向量,则非 0 向量是逻辑 1。例如,4'b0100 && 4'b0110 运算结果为 1,4'b0000 && 4'b0110 运算结果为 0。若操作数中含 x,则运算结果为 x。

4. 位逻辑运算符

按位逻辑运算有以下五种:

(1) ~:非运算。

(2) &:与运算,例如,4'b0100 & 4'b0110 运算结果为 4'b0100。

(3) |:或运算,例如,4'b0100 | 4'b0110 运算结果为 4'b0110。

(4) ^:异或运算,例如,4'b0100^ 4'b0110 运算结果为 4'b0010。

(5) ~^,^~:同或运算,例如,4'b0100~^ 4'b0110 运算结果为 4'b1101。

若操作数长度不相等,则长度短的操作数在高位补 0。例如,4'b0110^5'b10000 与 5'b00110^5'b10000 结果相同,为'b10110。

5. 条件运算符

条件运算符是三目运算符,根据条件表达式的值选择表达式,形式如下:

条件 ? 表达式 1:表达式 2

若条件为真(值为 1),则执行表达式 1;若条件为假(为 0),则执行表达式 2。

6. 连接运算符

连接运算符{}也称为拼接运算符,是将{}内的小表达式合并成大表达式,形式如下:

```
{expr1,expr2,…,exprN}
```

例如:

```
wire [7:0] Dbus;
assign Dbus [7:4] = {Dbus [0],Dbus [1],Dbus[2],Dbus[3]};
```

7. 移位运算符

在 Verilog 程序中,有<<(左移位运算符)和>>(右移位运算符)两种移位运算符。执行移位运算时,操作数空出的位置填 0。

例如,data=4'b0011,左移 2 位,即 data << 2 的运算结果为 data=4'b1100。

8. 操作符优先级

操作符优先级如表 3-2-1 所示,顶部优先级最高,底部优先级最低;同一行的操作符

具有相同的优先级。

表 3-2-1 操作符优先级

操作符	优先级
! ~ * / % + - << >> < <= > >= == != === !== & ^ ^~ | && || ? :	最高级 ↓ 最低级

3.3 Verilog 行为语句

Verilog 行为语句有过程语句、块语句、赋值语句、条件语句、循环语句等，如表 3-3-1 所示。

表 3-3-1 Verilog 行为语句

类别	语句	可综合
过程语句	initial	
	always	是
块语句	串行块 begin end	是
	并行块 fork join	
赋值语句	持续赋值 assign	是
	过程赋值 = <=	是
条件语句	if else	是
	case	是
循环语句	for	范围必须是静态时可综合
	repeat	重复值是常数时可综合
	while	
	forever	
编译预处理	'define	是
	'include	是
	'ifdef 'else 'endif	是

3.3.1 赋值语句

在 Verilog 语言中，有持续赋值语句和过程赋值语句，过程赋值有非阻塞赋值和阻塞赋值。

1. 持续赋值语句

assign 为持续赋值语句,主要对 wire 型变量赋值。例如:

```
assign #10 F = A&B;        //延时 10 个时间单位,将 A 与 B 赋值给 F
```

2. 过程赋值语句

过程赋值语句主要对 reg 型变量赋值。

1) 非阻塞赋值方式

符号"<="用于非阻塞赋值,非阻塞赋值语句在执行时不会阻塞后面的语句执行,也就是后面的赋值语句也同时执行。

【例 3-3-1】 非阻塞赋值的 Verilog 程序如图 3-3-1 所示,输入波形如图 3-3-2 所示,试画出输出波形。

解:由非阻塞赋值 Verilog 源码可知,在复位时,a=1,b=2,c=3;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,输出波形如图 3-3-2 所示。

```
//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
```

图 3-3-1 非阻塞赋值的 Verilog 程序

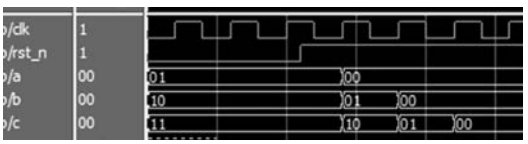

图 3-3-2 例【3-3-1】的输入输出波形

由波形图可知,在 begin 与 end 之间所有语句并行执行,且一个时钟只执行一次。

2) 阻塞赋值方式

符号"="用于阻塞的赋值,阻塞赋值"="在 begin 和 end 之间的语句是顺序执行,属于串行语句。

【例 3-3-2】 阻塞赋值的 Verilog 程序如图 3-3-3 所示,输入波形如图 3-3-4 所示,试画出输出波形。

解:在一个 always 块中,后面的语句会受到前语句的影响,如果一条阻塞赋值语句没有执行结束,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。

由阻塞赋值 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,之后将 a 的值赋值给 b,再将 b 的值赋值给 c;输出波形如图 3-3-4 所示。

```
//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
```

图 3-3-3 阻塞赋值 Verilog 程序

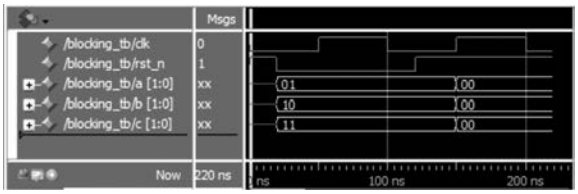

图 3-3-4 例【3-3-2】的输入输出波形

3.3.2 条件语句

条件语句有 if-else 语句和 case 语句，属于顺序语句，应放在 always 块内。

1. if-else 语句

if 语句用来判定所给定的条件是否满足，根据判定的结果决定执行给出的两种操作之一。语句格式：

```
if (表达式)        语句 1
else               语句 2
```

对表达式的值进行判断，若为 0、x、z，则按假处理；若为 1，则按真处理。若为真，则执行语句 1；若为假，则执行语句 2。

【例 3-3-3】 三态门如图 3-3-5 所示，用双重选择 if 语句实现三态门。

解：当三态门输入控制端为 1 时，输出等于输入；否则，输出为高阻态。用 if 语句描述的三态门如图 3-3-6 所示，当控制信号 control 为 1 时，dout＝din；否则，dout 为高阻态。

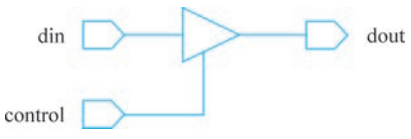

图 3-3-5 【例 3-3-3】的逻辑图

```
//三态门
module tri gate (
    input din,
    input control,
    output reg dout );
always @ ( din or control )
    if ( control ) dout = din;
         else      dout = 1'bz;
endmodule
```

图 3-3-6 【例 3-3-3】的 Verilog 程序

【例 3-3-4】 用多重选择 if 语句实现 4 选 1 数据选择器。

解：用多重选择 if 语句实现的 4 选 1 数据选择器 Verilog 程序如图 3-3-7 所示。当

输入选择端为 00 时,输出为 D0;当输入选择端为 01 时,输出为 D1;当输入选择端为 10 时,输出为 D2;当输入选择端为 11 时,输出为 D3。

2. case 语句

case 语句是一种多分支选择语句,if 语句只有两个分支可供选择,而实际问题中常常需要用到多分支选择,case 语句直接处理多分支选择。case 语句格式:

```
case(敏感表达式)
值 1: 语句 1;
值 2: 语句 2;
……
值 n: 语句 n;
default: 语句 n+1;
endcase
```

当敏感表达式的值为"值 1"时,执行语句 1;为"值 2"时,执行语句 2;以此类推。若列出的值都不符合,则执行 default 后面的语句。若列出了敏感表达式所有可能的取值,则 default 语句可以省略。

if-else 的条件表达式比 case 更为直观。当分支表达式中存在不定值 x 或高阻值 z 时,适合使用 case 语句。

【例 3-3-5】 用 case 语句实现 4 选 1 数据选择器。

解: 用 case 语句实现的 4 选 1 数据选择器 Verilog 程序如图 3-3-8 所示。当输入选择端为 00 时,输出为 D0;当输入选择端为 01 时,输出为 D1;当输入选择端为 10 时,输出为 D2;当输入选择端为 11 时,输出为 D3。

```
//4选1
module MUX4_1(
    input   S1,S0, //输入选择位
    input    D0, //输入低位
    input    D1,
    input    D2,
    input    D3,  //输入高位
    output  reg F ); //输出
always @ ( * ) begin
    if      ({S1,S0} == 2'b00) F = D0;
    else if ({S1,S0} == 2'b01) F = D1;
    else if ({S1,S0} == 2'b10) F = D2;
    else                       F= D3;
end
endmodule
```

图 3-3-7 【例 3-3-3】的 Verilog 程序

```
//4选1
module MUX4_1(
    input  [1:0]S, //输入选择位
    input    D0, //输入低位
    input    D1,
    input    D2,
    input    D3,  //输入高位
    output  reg F ); //输出
always @ ( * ) begin
    case (S)
      2'b00: F = D0;
      2'b01: F = D1;
      2'b10: F = D2;
      2'b11: F = D3;
    endcase
end
endmodule
```

图 3-3-8 【例 3-3-5】的 Verilog 程序

3.3.3 循环语句

在 Verilog 中有以下四种类型的循环语句,用来控制执行语句的执行次数:

（1）forever：连续执行语句，用在 initial 块中，生成时钟等周期性波形。

（2）repeat：连续执行一条语句 n 次。

（3）while：执行一条语句直到某个条件不满足。如果一开始条件就不满足（为假），则语句一次也不能被执行。

（4）for：有条件循环语句。

1. for 语句

for 语句的格式：

```
for(循环变量赋初值; 循环结束条件; 循环变量增值)
```

例如，由时钟控制实现的 8 位全加器的 Verilog 程序，如图 3-3-9 所示。用 for 语句实现 1 位全加器的 8 次循环，实现了 8 位全加器。

2. repeat 语句

repeat 语句格式：

```
repeat(表达式)  begin  语句或语句块  end
```

其中表达式为常量表达式。用 repeat 循环语句以及加法和移位实现 8 位乘法器，如图 3-3-10 所示。

```
`timescale 1ns / 1ps
module adder(
input [7:0] a,
input [7:0] b,
input clk,
output reg [7:0] sum,
output reg cout );
reg [8:0] c;
integer i;
always @ ( posedge clk ) begin
  c[0]=0;
  for ( i=0; i<=7 ; i=i+1) begin
    sum[i]= a[i]^b[i]^c[i];
    c[i+1]= (a[i]&b[i])|(a[i]&c[i])|(b[i]&c[i]);
  end
  cout=c[8];
end
endmodule
```

图 3-3-9　8 位全加器的 Verilog 程序

```
//8位乘法器
module mult8(
input [7:0] A,
input [7:0] B,
output reg [15:0] AmB );
reg [15:0] tempA,tempB;
always @( A or B ) begin
    AmB = 0;
    tempA = A;
    tempB = B;
      repeat (8) begin
          if (tempB[0])
          AmB = AmB + tempA;
          tempA = tempA<<1;
          tempB = tempB>>1;
      end
    end
endmodule
```

图 3-3-10　8 位乘法器的 Verilog 程序

3.4 Verilog 语言的描述语句

一个复杂的数字逻辑系统的 Verilog 模型是由若干 Verilog 模块构成的，每个模块又可以由若干子模块构成。用 Verilog 语言描述的数字逻辑电路就是该逻辑电路的 Verilog 模型（也称模块），是 Verilog 基本描述单位。一个模块可以是一个元件，也可以是更底层模块，模块是并行运行的。

模块有三种描述方式（也称建模方式），分别为行为级建模、结构级建模以及数据

流建模。若从逻辑电路结构描述电路模块,则称为结构描述形式;若对线性变量进行操作,则称为数据流描述形式;若从功能和行为描述电路模块,则称为行为级描述形式。

数据流建模用连续赋值语句 assign 实现,主要实现组合逻辑电路。例如,由逻辑门电路构成的 4 选 1 数据选择器的逻辑电路如图 3-4-1 所示,4 选 1 数据选择器的数据流描述 Verilog 程序如图 3-4-2 所示。

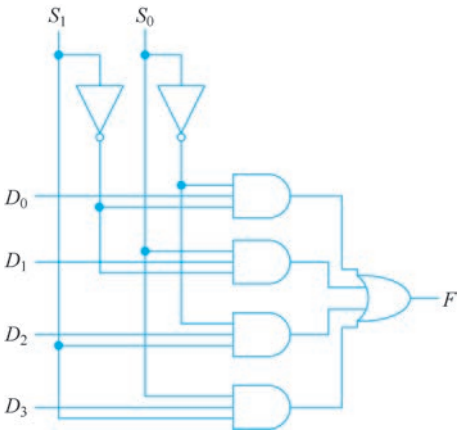

图 3-4-1　4 选 1 结构的逻辑图

```
//由门电路组成的4选1结构
module MUX4_1(
    input  wire  S0, //输入选择低位
    input  wire  S1, //输入选择高位
    input  wire  D0, //输入低位
    input  wire  D1,
    input  wire  D2,
    input  wire  D3,  //输入高位
    output wire  F    //输出
    );
//线网类型
wire    S0n,S1n;
wire    FD0,FD1,FD2,FD3;
//功能定义
assign  S0n =  ~ S0;
assign  S1n =  ~ S1;
assign  FD0 =  S1n & S0n & D0;
assign  FD1 =  S1n & S0  & D1;
assign  FD2 =  S1  & S0n & D2;
assign  FD3 =  S1  & S0  & D3;
assign  F = FD0 | FD1 | FD2 | FD3;
endmodule
```

图 3-4-2　4 选 1 结构的 Verilog 程序

结构描述就是在设计中实例化已有的功能模块,这些模块包括 Verilog 语言自带的以及用户自行开发的。例如,由 2 选 1 构成的 4 选 1 数据选择器的逻辑电路如图 3-4-3 所示,4 选 1 数据选择器的结构描述 Verilog 程序如图 3-4-4 所示。

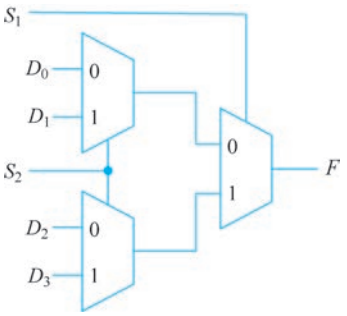

图 3-4-3　4 选 1 结构的逻辑图(2 选 1 结构构成的)

行为描述是对实体的数学模型的描述,只需要描述清楚输入和输出信号的行为,其抽象程度高于结构描述。可综合的行为描述大多采用 always 过程语句,适用于时序逻辑电路,也适用于组合逻辑电路。

```
//4选1结构描述
module MUX4_1(
input  [1:0]S,//输入选择
input   D0, //输入低位
input   D1,
input   D2,
input   D3,  //输入高位
output  F ); //输出
wire Z0,Z1;
MUX2_1 u0(
.S(S[0]),
.D0(D0),
.D1(D1),
.F(Z0));
MUX2_1 u1(
.S(S[0]),
.D0(D2),
.D1(D3),
.F(Z1));
MUX2_1 u2(
.S(S[1]),
.D0(Z0),
.D1(Z1),
.F(F));
endmodule
```

```
//由门电路组成的2选1结构
module MUX2_1(
input  wire S, //选择位
input  wire D0, //低位
input  wire D1, //高位
output wire F   //输出
);
//功能定义
assign F = ~S&D0|S&D1;
endmodule
```

图 3-4-4　4 选 1 结构描述的 Verilog 程序

【例 3-4-1】　已知 1 位全加器真值表(表 3-4-1),用行为描述实现 1 位全加器的 Verilog 程序。

表 3-4-1　全加器的真值表

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

解：用 case 语句实现的行为描述的 1 位全加器组合逻辑电路 Verilog 程序如图 3-4-5 所示。

【例 3-4-2】　已知 4 位二进制计数器 74161 的功能表(表 3-4-2),用行为描述实现 74161 的时序逻辑电路的 Verilog 程序。

解：用 always 语句实现的行为描述 74161 时序逻辑电路 Verilog 程序如图 3-4-6 所示。

```
//1位全加器的行为描述
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
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
```

图 3-4-5　1 位全加器的行为描述 Verilog 程序

表 3-4-2　74161 的功能表

功　能	输　入						输　出	
	CLR	CLK	ENP	ENT	LD	Di	Qi	RCO
清零	L	×	×	×	×	×	L	L
置数	H	↑	×	×	L	L	L	L
	H	↑	×	×	L	H	H	#
计数	H	↑	H	H	H	×	计数	#
保持	H	×	L	×	H	×	Qi	#
	H	×	×	L	H	×	Qi	L

注：Qi 表示在 CLK 上升沿之前的状态，i=0,1,2,3。

表示当计数值是 HHHH，且 ENT 是 H 时，RCO 为 H。

```
//74161行为描述
module LS161(
input  wire CLK,
input  wire CLRn,
input  wire LDn,
input  wire ENP,
input  wire ENT,
input  wire [3:0] D,
output reg [3:0] Q,
output wire RCO );
assign RCO = Q[3]&Q[2]&Q[1]&Q[0]&ENT;
always @( posedge CLK or negedge CLRn ) begin
    if ( ~CLRn ) Q <= 4'b0000;
        else if ( ~LDn )    Q <= D;
            else begin
               case ( {ENT, ENP})
                 2'b11: if ( Q < 4'b1111 )
                          Q <= Q + 1;
                        else if ( Q == 15 )
                          Q <= 4'b0000;
               default: Q <= Q;
               endcase
        end
    end
endmodule
```

图 3-4-6　74161 的行为描述 Verilog 程序

由【例 3-4-1】和【例 3-4-2】可知，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，但信号定义是 reg 类型，可综合得到组合逻辑电路。在 always 带时钟信号时，可综合得到时序逻辑电路。

3.5 Verilog 仿真验证

仿真是对所设计逻辑电路的一种检测方法，Verilog HDL 不仅提供了设计描述的能力，而且提供了对激励、响应和设计验证的建模能力。

进行逻辑电路的仿真需要用仿真器，ModelSim 是常用的仿真器。在 Verilog 语言中，将测试程序称为 testbench，意思是测试平台。测试程序与逻辑电路描述的 Verilog 程序类似，也是由模块组成，模块将描述测试激励、被测试对象和测试结果。测试激励用初始化语句产生，被测试模块在测试程序中作为一个实例嵌入，通过被测试模块的输出响应，判断模块功能是否正确。下面通过两个例子说明测试程序的编写方法。

【例 3-5-1】 针对【3-4-1】1 位全加器，编写相应的测试程序。

解：测试程序如图 3-5-1 所示。其中时间标尺用于定义模块的时间单位和时间精度，例如："timescale 1ns/100ps"，时延单位是 1ns，时延精度是 100ps。用 initial、always 定义激励信号波形。图 3-5-1(a)所示的测试程序是按照真值表定义的激励波形，ModelSim 仿真结果如图 3-5-2 所示。图 3-5-1(b)所示的测试程序是用 for 语句定义的激励波形，ModelSim 仿真结果如图 3-5-3 所示。

```
`timescale 1ns/100ps
module full_adder_tb;
// Inputs
    reg A,B,Cin;
// Outputs
    wire S,Cout;
// Instantiate
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
```

(a)

```
`timescale 1ns/100ps
module full_adder_tb;
// Inputs
    reg A,B,Cin;
// Outputs
    wire S,Cout;
// Instantiate
full_adder DUT(
.A(A),
.B(B),
.Cin(Cin),
.S(S),
.Cout(Cout));
reg [2:0] i;
initial begin
   for (i=0; i<=7; i=i+1)
   begin
   { A,B,Cin } = i; #100;
   end
end
endmodule
```

(b)

图 3-5-1 1 位全加器的测试程序

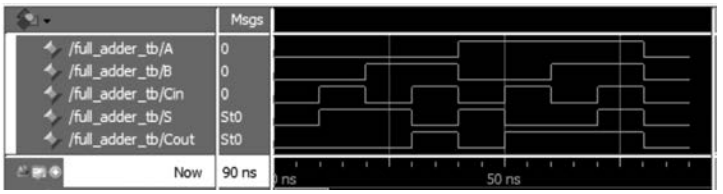

图 3-5-2　图 3-5-1(a)所示的测试程序 ModelSim 仿真结果

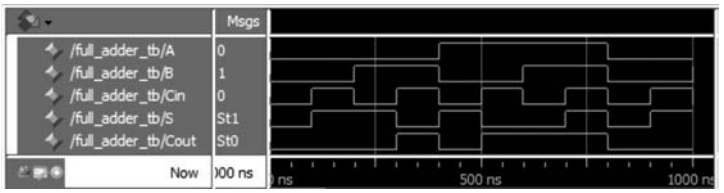

图 3-5-3　图 3-5-1(b)所示的测试程序 ModelSim 仿真结果

【例 3-5-2】　针对【3-4-2】4 位二进制计数器 74161，编写相应的测试程序。

解：测试程序如图 3-5-4 所示，ModelSim 仿真结果如图 3-5-5 所示。

```
`timescale 1ns/100ps
module LS161_tb();
reg  CLK,CLRn,LDn,ENP,ENT;
reg  [3:0] D;
wire [3:0] Q;
wire RCO;
//实例74161
LS161 DUT(
.CLK  (CLK),
.CLRn (CLRn),
.LDn  (LDn),
.ENP  (ENP),
.ENT  (ENT),
.D    (D),
.Q    (Q),
.RCO  (RCO));
//输入初始化
initial begin
    CLK  = 1'b0;
    D    = 4'b0000;
    LDn  = 1'b1;
    CLRn = 1'b1;
    ENP  = 1'b0;
    ENT  = 1'b0;
end
```

```
always #10 CLK = ~ CLK;
initial begin
    #10 CLRn = 1'b0;
    #15 CLRn = 1'b1;
end
initial begin
    #16 D = 4'b1100;
    #56 D = 4'b0000;
end
initial begin
    #35 LDn = 1'b0;
    #30 LDn = 1'b1;
end
initial begin
    #70  ENP = 1'b1;
    #260 ENP = 1'b0;
    #60  ENP = 1'b1;
end
initial begin
    #70   ENT = 1'b1;
    #425 ENT = 1'b0;
    #5    ENT = 1'b1;
    #40  $stop;
end
endmodule
```

图 3-5-4　4 位二进制计数器 74161 的测试程序

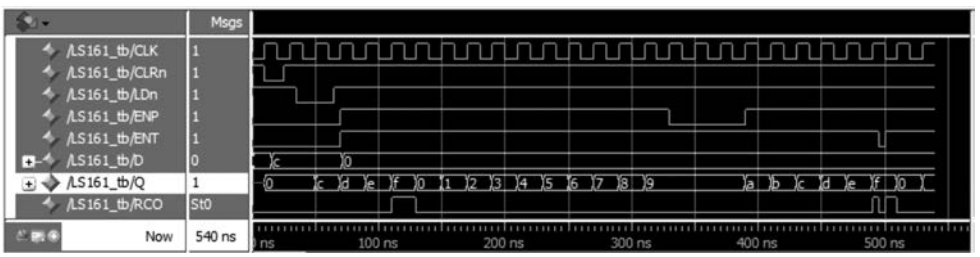

图 3-5-5　4 位二进制计数器 74161 的 ModelSim 仿真结果

3.6 Verilog 关键字

Verilog 程序设计常用的关键字以及含义见表 3-6-1。表 3-6-2 是 Verilog 关键字列表。

表 3-6-1 Verilog 常用关键字

关 键 字	含 义
module	模块开始定义
input	输入端口定义
output	输出端口定义
inout	双向端口定义
parameter	信号的参数定义
wire	线网信号定义
reg	寄存器信号定义
always	产生 reg 信号语句的关键字
assign	产生 wire 信号语句的关键字
begin	块语句的起始标志
end	块语句的结束标志
posedge/negedge	上升沿/下降沿有效时序电路的标志
case	case 语句起始标记
default	case 语句的默认分支标志
endcase	case 语句结束标记
if	if/else 语句标记
else	if/else 语句标记
for	for 语句标记
endmodule	模块结束定义

表 3-6-2 Verilog 关键字

and	always	assign	begin	buf
bufif0	bufif1	case	casex	casez
cmos	deassign	default	defparam	disable
edge	else	end	endcase	endfunction
endprimitive	endmodule	endspecify	endtable	endtask
event	for	force	forever	fork
function	highz0	highz1	if	ifnone
initial	inout	input	integer	join
large	macromodule	medium	module	nand
negedge	nor	not	notif0	notif1
nmos	or	output	parameter	pmos
posedge	primitive	pulldown	pullup	pull0
pull1	rcmos	real	realtime	reg
release	repeat	rnmos	rpmos	rtran
rtranif0	rtranif1	scalared	small	specify
specparam	strength	strong0	strong1	supply0

续表

supply1	table	task	tran	tranif0
tranif1	time	tri	triand	trior
trireg	tri0	tri1	vectored	wait
wand	weak0	weak1	while	wire
wor	xnor	xor		

习题

3-1 用 Verilog 语言描述图 3-P-1 所示逻辑电路。

3-2 用 Verilog 实现二输入异或门 $F=A\oplus B$ 的逻辑功能。

3-3 用 Verilog 语言实现图 3-P-2 所示的逻辑电路。编写图 3-P-3 所示输入波形的 ModelSim 仿真 Verilog 程序,给出 Q 和 Q_n 的仿真结果。

图 3-P-1

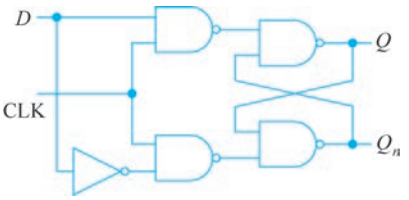

图 3-P-2

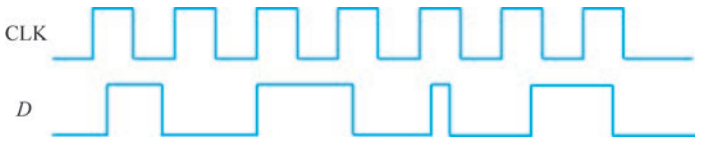

图 3-P-3

3-4 在 Verilog 源程序文件中怎样标明注释?

3-5 用 Verilog 语言实现图 3-P-4 所示 4 选 1 数据选择器。

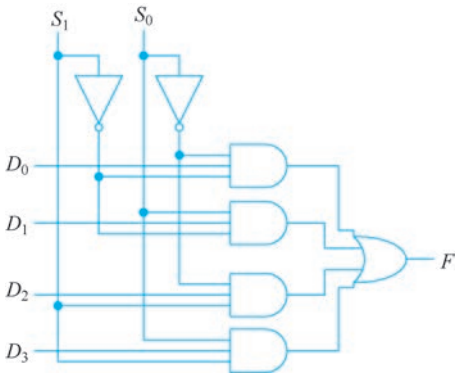

图 3-P-4