第 5 章　Verilog HDL 语言

5.1　概　　述

硬件描述语言（Hardware Description Language，HDL）是一种用形式化方法来描述数字电路和系统的语言，它可用一系列分层次的模块来表示复杂的数字系统，从上到下逐层描述，并逐层进行验证仿真。再把具体的模块由综合工具转化成门级网表，接下来利用布局布线工具把网表转化为具体电路结构的实现。现在，这种自顶向下的设计方法已被广泛使用。HDL 语言具有以下主要特征：

- HDL 语言既包含一些高级程序设计语言的结构形式，同时也兼顾描述硬件线路连接的具体结构。
- HDL 语言采用自顶向下的设计方法，通过使用分层的行为描述，可以在不同的抽象层次描述设计。
- HDL 语言是并行处理的，具有同一时刻执行多任务的能力，这和一般的高级设计语言（如 C 语言）的串行执行是不同的。
- HDL 语言具有时序的概念，为了描述这一特征，需要引入时延的概念，因此它不仅可以描述硬件电路的功能，还可以描述电路的时序。

Verilog HDL 和 VHDL 是目前最流行的两种 HDL 语言，均为 IEEE 标准，被广泛应用于基于可编程逻辑器件（Programmable Logic Device, PLD）的项目开发中。与 VHDL 相比，Verilog HDL 的部分语法参照 C 语言语法（但与 C 有本质区别），因此具有 C 语言的优点。从表述形式上来看，Verilog HDL 程序代码简明扼要，使用灵活，且语法规定不是很严谨，很容易上手。Verilog HDL 程序具有很强的电路描述和建模能力，能从多个层次对数字系统进行建模和描述，从而大大简化了硬件设计任务，提高了设计效率和可靠性。Verilog HDL 程序在语言易读性、层次化和结构化设计方面表现出了强大的生命力和应用潜力，在全球范围内用户覆盖率一直处于上升趋势。因此，本书以 Verilog HDL 作为基本硬件描述语言来介绍 EDA 技术及其实践。

5.2　Verilog HDL 基本结构

用 Verilog HDL 描述的电路设计就是该电路的 Verilog HDL 模型，也称为模块（module），是 Verilog HDL 程序的基本描述单位。用模块描述某个设计的功能或结构，以及与其他模块通信的外部接口。一般来说，一个文件就是一个模块，但也有例外。一个 Verilog HDL 程序模块的基本架构如下：

module 模块名（模块端口名表）;

　　模块端口和模块描述

endmodule

下面以一个简单的例子加以说明。例 5-1 用 Verilog HDL 语言描述一个图 5-1 所示的上升沿有效 D 触发器，其中 clk 为触发器的时钟，data 和 q 分别为触发器的输入和输出。

【例 5-1】

```
module dff_pos(data,clk,q);
input data,clk;
output q;
reg q;
always @(posedge clk)
    q=data;
endmodule
```

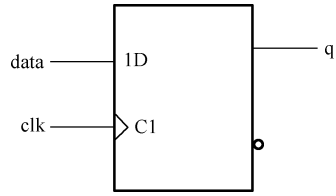

图 5-1　上升沿 D 触发器

结合上例，一个完整的 Verilog HDL 模块由以下 5 部分组成。

1．模块定义

模块定义用来声明电路设计模块名及其输入/输出端口，格式如下：

module 模块名（端口 1，端口 2，端口 3，…）；

当无端口名列表时，括号可以省去。上例中，模块名为 dff_pos，并定义了 3 个端口名 data、clk 和 q。这些端口名等价于硬件中的外接引脚，模块通过这些端口与外接发生联系。

2．端口类型说明

端口类型说明用来声明模块定义中各端口数据的流动方向，Verilog HDL 端口类型只有输入（input）、输出（output）和双向端口（inout）3 种。端口类型说明格式如下：

input 端口 1，端口 2，端口 3，…；　//声明输入端口

output 端口 1，端口 2，端口 3，…；　//声明输出端口

凡是出现在端口名列表中的端口，都必须显示说明其端口类型。上例中，data 和 clk 为输入端，q 为输出端。

3．数据类型说明

用来声明设计电路的功能描述中所用的信号的数据类型。Verilog HDL 支持的数据类型有连线类型和寄存器类型两个大类，每个大类又细分为多种具体的数据类型，除了一位宽的 wire 类可被缺省外，其他凡将在后面的描述中出现的变量都应给出相应的数据类型说明。

上例中，q 是 reg 类型，data 和 clk 没有给出相应的数据类型说明，因而它们都缺省为一位位宽的 wire 类型。由于 q 被定义为 reg 类型，因而可以被接下来的过程赋值语句赋值。reg 类型的行为方式与 C 语言中的一般变量相似，在接受下一次过程赋值语句前，它将保持原值不变；在硬件上，其行为特征类似于一个寄存器，因而称之为 reg 类。

4．描述体部分

描述体部分是 Verilog HDL 程序设计中最主要的部分，用来描述设计模块的内部结构和模块端口间的逻辑关系，在电路上相当于器件的内部电路结构。描述体部分可以用 assign 语句、元件例化（instantiate）方式、always 块语句、initial 块语句等方法来实现，通常把确定这些设计模块描述的方法称为建模。

（1）assign 语句建模。

assign 语句在 Verilog HDL 中称为连续赋值语句，用于逻辑门和组合逻辑电路的描述，它的格式为：

assign 赋值变量=表达式；

例如，具有 a、b、c、d 4 个输入和 y 为输出的与非门的连续赋值语句为：

assign y=～（a&b&c&d);

连续赋值语句“=”号两边的变量都应该是 wire 型变量。在执行中，输出 y 的变化跟随输入 a、b、c、d 的变化而变化，反映了信息传送的连续性。

（2）元件例化（instantiate）方式建模。

元件例化方式建模是利用 Verilog HDL 提供的元件库实现的。例如，用与门例化元件定义一个三输入与门可以写为：

and myand3(y,a,b,c);

其中，and 是 Verilog HDL 元件库中的与门元件名，myand3 是例化出的三输入与门名，y 是与门的输出端，a、b、c 是输入端。

（3）always 块语句建模。

always 是一个过程语句，常用于时序逻辑的功能描述，它的格式为：

always @（敏感信号及敏感信号列表或表达式）

包括块语句的各类顺序语句

一个程序设计模块中，可以包含一个或多个 always 语句。程序运行时，当敏感信号列表中的事件发生时，将执行一遍后面的块语句中所包含的各条语句，因此敏感信号列表中列出的事件又称为过程的触发条件或激活条件。块语句通常由 begin-end 或 fork-join 所界定，前者为串行块，块中的各条语句按串行方式顺序执行；后者为并行块，块中的语句按并行方式同时执行。

上例中只有一条语句 q=data，串行块标识符 begin-end 可被省略。

always 过程语句在本质上是一个循环语句，每当触发条件被满足时，过程就重新被执行一次。如果没有给出敏感信号列表，即没有给出触发条件，则相当于触发条件一直被满足，循环就将无休止地执行下去。

（4）用 initial 块语句建模。

initial 也是一个过程语句，它与 always 语句类似，但 initial 语句不带触发条件，它只在程序开始时执行一次。

5．结束行

结束行就是用关键词 endmodule 标志模块定义的结束，注意它的后面没有分号。

用 Verilog HDL 进行描述后，整个电子系统就由这样的 module 模块所组成，一个模块可以大到代表一个完整系统，也可以小到仅仅代表一个最基本的逻辑单元。从模块外部加以考察，一个模块由模块名以及相应的端口特征所唯一确定。模块内部的具体行为的描述并不会影响该模块与外部之间的连接关系。一个 Verilog HDL 程序模块可以被任意多个其他模块所调用。但由于 Verilog HDL 所描述的是具体的硬件电路，一个模块代表具有特定功能的一个电路块，每当它被某个其他模块调用一次时，则在该模块内部，被调用的电路块将被原原本本地复制一次。

5.3 Verilog HDL 的描述方式

Verilog HDL 具有行为描述和结构描述功能。行为描述是对设计电路的逻辑功能的描述，并不用关心设计电路使用哪些元件以及这些元件之间的连接关系。行为描述属于高层次的描述方法，在 Veirlog HDL 中，行为描述包括系统级（System Level）、算法级（Algorithm Level）、和寄存器传输级（Register Transfer Level，RTL）等 3 种抽象级别。

结构描述是对设计电路的结构进行描述，即描述电路使用的元件以及这些元件之间的连接关系。结构描述属于低层次的描述方法，在 Verilog HDL 中，结构描述包括门级（Gate Level）和开关级（Switch Level）两种抽象级别。

1．Verilog HDL 行为描述

Verilog HDL 行为描述是最能体现 EDA 风格的硬件描述方式，它和其他软件编程语言类似，通过行为语句来描述电路要实现的功能，表示输入与输出间的转换，不涉及具体结构。

下面以图 5-2 所示的 2 选 1 数据选择器为例，它的 Verilog HDL 行为描述程序模块如下。

【例 5-2】

```
module mux_beh(out,a,b,sel);
    output out;
    input a,b,sel;
    assign out=(sel= =0)?a:b;
endmodule
```

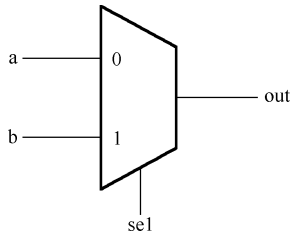

图 5-2　2 选 1 数据选择器

行为描述中，用连续赋值语句 assign 实现了在 sel 的控制下，输出信号 out 与输入信号 a、b 之间的硬件连接关系，每当 sel、a、b 三个信号有任何变化时，都将被随时反映到输出端 out 信号上来。

2．Verilog HDL 结构描述

结构描述是将硬件电路描述成一个分级子模块互连的结构。通过对组成电路的各个子模块间互相连接关系的描述来说明电路的组成。在结构描述中，门和 MOS 开关是电路最底层的结构。在 Verilog HDL 中定义了 26 个基本单元，又称基元，见表 5-1。

表 5-1　Verilog HDL 中的基元

基元分类	基元
多输入门	and，nand，or，nor，xor，xnor
多输出门	buf，not
三态门	bufif0，bufif1，notif0，notif1
上拉、下拉电阻	pullup，pulldown
MOS 开关	coms，nmos，pmos，rcmos，rnmos，rpmos
双向开关	tran，tranif0，tranif1，rtran，rtranif0，rtranif1

仍以 2 选 1 数据选择器为例，它的 Verilog HDL 结构描述程序模块见例 5-3。

【例 5-3】

```
module mux_str(out,a,b,sel);
```

```
    output out;
    input a,b,sel;
    wire net1,net2,net3;
    not gate1(net1,sel);
    and gate2(net2,a,net1);
    and gate3(net3,b,sel);
    or gate4(out,net2,net3);
endmodule;
```

对结构描述模块来说，建议将它的 Verilog HDL 描述与图 2-3 所示的逻辑图加以对照。

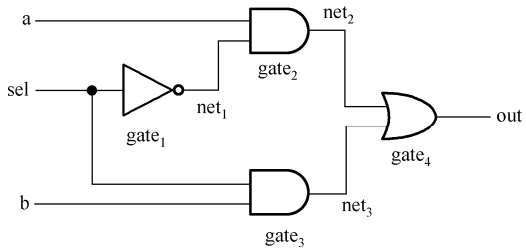

图 5-3　2 选 1 数据选择器逻辑图

可以发现，结构描述只是忠实地将图形方式的逻辑连接关系转变为相应的文字表达而已。在对每一个逻辑电路进行结构描述前，先给电路中的每个元件取一个名字，并以同样的方式给每条内部连线也取相应的名字，然后再依据逻辑图中的连接关系，确定各单元端口间的信号连接，完成描述的全过程。

5.4　Verilog HDL 基本词法

Verilog HDL 的词法符号包括空白符、注释、操作符、常数、字符串、标识符和关键字。

1．空白符和注释

Verilog HDL 语言的空白符包括空格、TAB 键、换行符及换页符，空白符起到分隔符的作用。

Verilog HDL 语言中，注释的定义与 C 语言完全一致，分单行注释与多行注释两类。单行注释以“//”开始到行末结束，不允许续行；多行注释以“/*”开始，到“*/”结束，可以跨越多行，但不允许嵌套。

2．常数

在 Verilog HDL 中，常数包括数字、未知值 x 和高阻值 z 三种。数字可以用二进制、八进制、十进制和十六进制等 4 种不同的数字来表示，完整的数字格式为：

<位宽>'<进制符号><数字>

其中，位宽表示数字对应二进制数的位数宽度；进制符号 b 或 B 表示二进制数，d 或 D 表示十进制数，o 或 O 表示八进制数，h 或 H 表示十六进制数。十进制输的位宽和进制符号可以默认。例如：

```
8b'10110001          //表示位宽为 8 位的二进制数 10110001
```

```
8h'f5                    //表示位宽为 8 位的十六进制数 f5
125                      //表示十进制数 125
```

另外，用 x 和 z 分别表示未知值和高阻值，它们可以出现在除十进制数以外的数字形式中。x 和 z 的位数由所在的数字格式决定，在二进制中，一个 x 或 z 表示 1 位未知位或 1 位高阻位；在八进制中，一个 x 或 z 表示 3 位未知位或 3 位高阻位；在十六进制中，一个 x 或 z 表示 4 位未知位或 4 位高阻位。例如：

```
8b'1111xxxx              //等价于 8h'fx
```

3．字符串

字符串是用双引号“”括起来的字符序列，它必须包含在同一行中，不能多行书写。在表达式或赋值语句中作为操作数的字符串被看作 ASCII 值序列，即一个字符串中的每一个字符对应一个 8 位的 ASCII 值。

4．标识符

标识符是用户编程时为常量、变量、模块、寄存器、端口、连线、示例和 begin-end 块等元素定义的名称。标识符可以是字母、数字和下画线（_）等符号组成的任意序列。定义标识符时应遵循以下规则：

- 首字符不能是数字；
- 字符数不能多于 1024 个；
- 大小写字母是不同的；
- 不要与关键字同名。

5．关键字

Verilog HDL 语言内部已经使用的词称为关键字，用户应避免使用。所有的关键字都是小写的。表 5-2 给出了 Verilog HDL 关键字的清单。

表 5-2　Verilog HDL 关键字

always	case	edge	endtask	highz1	large
and	casex	else	event	if	macromodule
assign	casez	end	for	ifnone	nand
attribute	cmos	endattribute	force	initial	negedge
begin	deassign	endmodule	forever	inout	nmos
buf	default	endprimitive	fork	input	nor
bufif0	defparam	endspecify	function	integer	not
bufif1	disable	endtable	highz0	join	notif0
notif1	pulldown	rtranif0	strong1	tri1	weak1
or	pullup	rtranif1	supply0	triand	while
output	rcmos	scalared	supply1	trior	wire
parameter	reg	signed	table	trireg	wor
pmos	release	small	task	unsigned	xnor
posedge	repeat	specify	tranif0	vectored	
primitive	rnmos	specparam	tranif1	wait	
pull0	rpmos	strength	tri	wand	
pull1	rtran	strong0	tri0	weak0	

6．操作符

操作符又称运算符，按照操作数的个数，可以分为一元、二元和三元操作符；按功能可以大致分为算术操作符、逻辑操作符、比较操作符等几大类。Verilog HDL 操作符及说明见表 5-3。

表 5-3　Verilog HDL 操作符及说明

分类	操作符及功能		简要说明
算术操作符	+ − * / %	加 减 乘 除 求余	二元操作符，即有两个操作数。操作数可以是物理数据类型，也可以是抽象数据类型
比较操作符	> < >= <= == != === !==	大于 小于 大于等于 小于等于 等于 不等于 全等 不全等	二元操作符，如果操作数之间的关系成立，返回值为 1；否则返回值为 0。若某一个操作数的值不定，则关系是模糊的，返回值是不定值 x
逻辑操作符	&& || !	逻辑与 逻辑或 逻辑非	&&和||为二元操作符；!为一元操作符，即只有一个操作数
位操作符	～ & | ^ ^～(～^)	按位取反 按位与 按位或 按位异或 按位同或	“～”是一元操作符，其余都是二元操作符。将操作数按位进行逻辑运算
缩位操作符	& ～& | ～| ^ ^～(～^)	缩位与 缩位与非 缩位或 缩位或非 缩位异或 缩位同或	一元操作符，对操作数各位的值进行运算。如“&”是对操作数各位的值进行逻辑与运算，得到一个一位的结果值
移位操作符	>> <<	右移 左移	二元操作符，对左侧的操作数进行其右侧操作数指明的位数的移位，空出的位用 0 补全
条件操作符	?:		三元操作符，如：a?b:c，若第一个操作数 a 为逻辑 1，则返回第二个操作数 b，否则返回第三个操作数 c
连接和复制符	{,}		将两个或两个以上用逗号分隔的表达式并置连接在一起

操作符的优先级如表 5-4 所示，表中顶部的操作符优先级最高，底部的最低，列在同一行的操作符优先级相同。所有的操作符（除“?:”外）在表达式中都是从左向右结合的。可以通过括号来改变优先级，并使运算顺序更清晰。

表 5-4　操作符的优先级

优先级序号	操作符	操作符名称
1	!、～	逻辑非、按位取反
2	*、/、%	乘、除、求余
3	+、−	加、减

续表

优先级序号	操作符	操作符名称
4	<<、>>	左移、右移
5	<、<=、>、>=	小于、小于等于、大于、大于等于
6	==、!=、===、!==	等于、不等于、全等、不全等
7	&、~&	缩位与、缩位与非
8	^、^~	缩位异或、缩位同或
9	|、~|	缩位或、缩位或非
10	&&	逻辑与
11	||	逻辑或
12	?:	条件操作符

5.5 Verilog HDL 数据对象

Verilog HDL 程序数据对象包括常量和变量。

1．常量

常量是一个恒定不变的数，一般在程序前面定义。常量定义的格式为：

parameter 常量名 1=表达式 1，常量名 2=表达式 2，…，常量名 *n*=表达式 *n*；

其中，parameter 是常量定义关键字，常量名是用户定义的标识符，表达式是为常量赋的值。

2．变量

变量是在程序运行时其值可以改变的量。在 Verilog HDL 中，变量分为连线类型（Net-type）和寄存器类型（Register-type）两种。

（1）连线类型。

连线类型对应的是电路中的物理信号连接，对它的驱动有两种方式：一种方式是结构描述中把它连接到一个门或模块的输出端；另一种方式是用连续赋值语句 assign 对其进行赋值。由于 assign 语句在物理上等同于信号之间的实际连接，因而该语句不能出现在过程语句（initial 或 always）中后面的过程块语句中。连线类型没有电荷保持作用（trireg 除外），当没有被驱动时，它将处在高阻态 z（对应于 trireg 为 x 态）。

连线型变量的输出值始终根据输入的变化而更新，它一般用来定义硬件电路中的各种物理连线。表 5-5 给出了 Verilog HDL 提供的连线类型及其功能。

表 5-5 连线类型及其功能

连线类型	功能说明
wire,tri	标准连线（缺省为该类型）
wor,trior	具有线或特性的连线
wand,triand	具有线与特性的连线
trireg	具有电荷保持特性的连线
tir1,tri0	上拉电阻（pullup）和下拉电阻（pulldown）
supply0,supply1	电源（逻辑 1）和地（逻辑 0）

wire 是最常用的连线型变量。在 Verilog HDL 模块中，输入/输出信号类型缺省时自动定义为 1 位宽的 wire 型。对综合而言，wire 型变量的取值可以是 0、1、x 和 z。wire 型变量的定义格式如下：

wire 变量名 1，变量名 2，…，变量名 *n*；

例如：

```
wire a,b,c;          //定义了 3 个 wire 型的变量，位宽均为 1 位，可缺省
wire[7,0] databus;   //定义了 1 个 wire 型的变量，位宽均为 8 位
```

（2）寄存器类型。

寄存器类型对应的是具有状态保持作用的硬件电路元件，如触发器、锁存器等。寄存器类型的驱动可以通过过程赋值语句实现，过程赋值语句类似于 C 语言中的变量赋值语句，在接受下一次的过程赋值之前，将保持原值不变。过程赋值语句只能出现在过程语句（initial 或 always）中后面的过程块语句中。当寄存器类型没有被赋值前，它将处于不定态 x。

在 Verilog HDL 中，有 4 种寄存器类的数据类型，见表 5-6。

表 5-6　寄存器类型及其说明

寄存器类型	功能说明
reg	用于行为描述中对寄存器类的说明，由过程赋值语句赋值
integer	32 位带符号整型变量
real	64 位浮点、双精度、带符号实型变量
time	64 位无符号时间变量

integer、real 和 time 等 3 种寄存器类型变量都是纯数学的抽象描述，不对应任何具体的硬件电路，但它们可以描述与模拟有关的计算。例如，可以利用 time 型变量控制经过特定的时间后执行赋值。

reg 型变量是最常用的寄存器型变量，常用于具体的硬件描述，它的定义格式如下：

reg 变量名 1，变量名 2，…，变量名 *n*；

例如：

```
reg a,b,c;           //定义了 3 个 reg 型的变量 a，b，c，位宽均为 1 位
reg[7:0] data;       //定义了 1 个 reg 型的变量，位宽为 8 位
```

位宽为 1 的变量称为标量，位宽超过 1 位的变量称为矢量。

5.6　Verilog HDL 基本语句

Verilog HDL 的语句包括块语句、赋值语句、条件语句和循环语句等。在这些语句中，有些属于顺序执行语句，有些属于并行执行语句。

5.6.1　块语句

块语句通常用来将两条或多条语句组合在一起。块语句有两种，一种是 begin-end 语句，通常用来标识顺序执行的语句，用它来表示的块称为顺序块；一种是 fork-join 语句，通常用

来标识并序执行的语句，用它来表示的块称为并行块。当块语句中只包含一条语句时，块标识符 begin-end 和 fork-join 可以省略。

1．顺序块

顺序块具有以下特点：

- 块内的语句是按顺序执行的，即只有上面一条语句执行完后下面的语句才能执行。
- 每条语句的延迟时间是相对于前一条语句的仿真时间而言的。
- 直到最后一条语句执行完，程序流程控制才跳出该语句块。

顺序块的格式如下：

begin
　　语句 1；
　　语句 2；
　　⋮
　　语句 *n*；
end

或

begin：块名
　　块内声明语句
　　语句 1；
　　语句 2；
　　⋮
　　语句 *n*；
end

其中：

- 块名即该块的名字，是一个标识符，其作用在后面再详细介绍。
- 块内声明语句可以是参数声明语句、reg/integer/real 型变量声明语句。

【例 5-4】

```
begin
    areg=breg;
    #10 creg=areg;      //creg 的值为 breg 的值
                        //在两条赋值语句间延迟 10 个单位时间
end
```

例：

```
parameter d=50;         //声明 d 是一个参数
reg[7:0]    r;          //声明 r 是一个 8 位的寄存器变量
begin                   //由一系列延迟产生的波形
    #d   r='h35;
    #d   r='hE2;
    #d   r='h00;
    #d   r='hF7;
    #d   ->end_wave;    //触发事件 end_wave
end
```

这个例子用顺序块和延迟控制组合来产生一个时序波形。

2．并行块

并行块具有以下特点：

- 块内语句是同时执行的，即程序流程控制一进入到该并行块，块内语句开始同时并行地执行。

- 块内每条语句的延迟时间是相对于程序流程控制进入到块内的仿真时间的。
- 延迟时间是用来给赋值语句提供执行时序的。
- 当按时间时序排序在最后的语句执行完后，或一个 disable 语句执行时，程序流程控制跳出该程序块。

并行块的格式如下：

fork
　　语句 1；
　　语句 2；
　　⋮
　　语句 *n*；
join

或

fork：块名
　　块内声明语句
　　语句 1；
　　语句 2；
　　⋮
　　语句 *n*；
join

其中：

- 块名即标识该块的一个名字，相当于一个标识符。
- 块内声明语句可以是参数声明语句、reg/integer/real 型变量声明语句或事件（event）声明语句。

【例 5-5】

```
fork
    #50     r='h35;
    #100    r='hE2;
    #150    r='h00;
    #200    r='hF7;
    #250    ->end_wave;
join
```

在这个例子中用并行块代替了前面例子中的顺序块来产生波形，用这两种方法生成的波形是一样的。

3．块名

在 Verilog HDL 中，可以给每个块取一个名字，只需将名字加在关键词 begin 或 fork 后面即可。这样做的原因有以下几点：

- 可以在块内定义局部变量，即只在块内使用的变量。
- 可以允许块被其他语句调用，如 disable 语句。
- 在 Verilog HDL 程序语言里，所有的变量都是静态的，即所有的变量都只有一个唯一的存储地址，因此进入或跳出块并不影响存储在变量内的值。

基于以上原因，块名就提供了一个在任何仿真时刻确认变量值的方法。

5.6.2　赋值语句

在 Verilog HDL 中，赋值语句常用于描述硬件设计电路输出与输入之间的信息传送。Verilog HDL 有过程赋值、连续赋值和基本门单元赋值。

1．过程赋值语句

Verilog HDL 对模块的行为描述由一个或多个并行运行的过程块组成，而位于过程块中的赋值语句称为过程赋值语句。过程赋值语句只能对寄存器类的变量进行赋值。

过程赋值语句出现在 initial 和 always 块语句中，过程赋值语句有两种赋值方式：阻塞型过程赋值与非阻塞型过程赋值。

（1）阻塞型过程赋值。

阻塞型过程赋值语句的赋值符号是“=”，语句格式为：

赋值变量=表达式；

阻塞型赋值语句的值在该语句结束时就可以得到，如果一个顺序块中包含若干条阻塞型过程赋值语句，那么这些赋值语句是按照语句在程序中的顺序由上至下一条一条执行的，前面的语句没有完成，后边的语句就不能执行，就如同被阻塞了一样。

（2）非阻塞型过程赋值。

非阻塞型过程赋值语句的赋值符号是“<=”，语句格式为：

赋值变量<=表达式；

非阻塞型赋值语句的值不是在该语句结束时得到，而是在该块语句结束后才能得到。在一个顺序块中，一条非阻塞语句的执行并不会影响块中其他语句的执行。当一个顺序块中的语句全部由非阻塞型赋值语句构成时，这个顺序块的执行与并行块是完全一致的。

通过下面两个例子可以比较一下这两种赋值语句。

【例 5-6】

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

【例 5-7】 b

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

例 2-6 中 always 块用了阻塞型赋值语句，在 clk 上升沿时，b 马上取 a 的值，c 马上取 b 的值，所以该例的综合结果如图 5-4 所示。

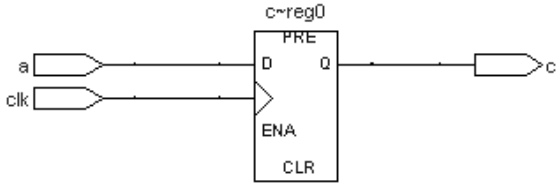

图 5-4　例 5-6 阻塞型赋值语句综合结果

例 5-7 中 always 块用了非阻塞型赋值语句，两条赋值语句在顺序块结束语句 end 处同时完成，它的综合结果如图 5-5 所示。

2．连续赋值语句

Verilog HDL 中的赋值语句主要有两类，一类是上面介绍的过程赋值语句，另一类就是连续赋值语句，它们之间的主要差别如下。

（1）赋值对象不同。

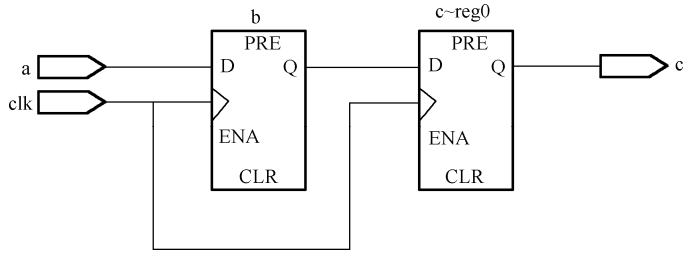

图 5-5　例 5-7 非阻塞型赋值语句综合结果

连续赋值语句用于对连线类变量的赋值，过程赋值语句完成对寄存器类变量的赋值。

（2）赋值过程实现方式不同。

连线类变量一旦被连续赋值语句赋值后，赋值语句右端表达式中的信号有任何变化，都将随时反映到左端的连线变量中；过程赋值语句只有在语句被执行到时赋值过程才进行一次，且赋值过程的具体执行时刻还受到定时控制及延时模式等多方面因素的影响。

（3）语句出现位置不同。

连续赋值语句不能出现在任何一个过程块中；过程赋值语句则只能出现在过程块中。

（4）语句结构不同。

连续赋值语句的格式为：assign 赋值变量=表达式；语句中的赋值算符只有阻塞型一种形式；过程赋值语句不需要相应的先导关键词，语句中的赋值算符有阻塞型和非阻塞型两类。

（5）冲突处理方式不同。

一条连线可被多条连续赋值语句同时驱动，最后的结果依据连线类型的不同有相应的冲突处理方式；寄存器变量在同一时刻只允许一条过程赋值语句对其进行赋值。

3．基本门单元赋值

基本门单元赋值语句的格式为：

基本逻辑门关键字（门输出，门输入 1，门输入 2，…，门输入 *n*）；

其中，基本门逻辑关键字是 Verilog HDL 预定义的逻辑门，包括 and、or、not、xor、nand 和 nor 等。

例如，具有 a、b、c、d 四个输入和输出 y 的与非门的基本门单元赋值语句为：

```
nand(y,a,b,c,d);          //该语句与 assign y=～(a&b&c&d)等效
```

5.6.3　条件语句

条件语句包含 if 语句和 case 语句，它们都是顺序语句。

1．if 语句

在 Verilog HDL 中，完整的 if 语句结构如下：

```
if（表达式）
    begin
        语句;
    end
else if （表达式）
```

```
    begin
        语句;
    end
else
    begin
        语句;
    end
```

根据需要，if 语句可以写为另外两种变化形式：

① if（表达式）

```
    begin
        语句;
    end
```

② if（表达式）

```
    begin
        语句;
    end
  else
    begin
        语句;
    end
```

在 if 语句中，“表达式”一般为逻辑表达式或关系表达式，也可以是位宽为 1 位的变量。系统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，执行相应的语句。语句可以是多句，多句是用“begin-end”语句括起来；也可以是单句，单句的“begin-end”可以省略。对于 if 嵌套语句，如果不清楚 if 和 else 的匹配，最好用“begin-end”语句括起来。

if 语句在程序中用来改变控制流程。

举例：用 if 语句设计 8-3 线优先编码器。8-3 线优先编码器功能表如表 5-7 所示。

表 5-7　8-3 线优先编码器功能表

输入								输出		
din7	din6	din5	din4	din3	din2	din1	din0	dout2	dout1	dout0
0	x	x	x	x	x	x	x	1	1	1
1	0	x	x	x	x	x	x	1	1	0
1	1	0	x	x	x	x	x	1	0	1
1	1	1	0	x	x	x	x	1	0	0
1	1	1	1	0	x	x	x	0	1	1
1	1	1	1	1	0	x	x	0	1	0
1	1	1	1	1	1	0	x	0	0	1
1	1	1	1	1	1	1	0	0	0	0

Verilog HDL 代码如下。

【例 5-8】

```
module encode8_3(y,a);
input[7:0] din;
output[2:0] dout;
reg[2:0]    y;
always @(a)
    begin
        if (~din [7])      dout <=3'b111;
        else if (~din [6]) dout <=3'b110;
        else if (~din [5]) dout <=3'b101;
        else if (~din [4]) dout <=3'b100;
        else if (~din [3]) dout <=3'b011;
        else if (~din [2]) dout <=3'b010;
        else if (~din [1]) dout <=3'b001;
        else               dout <=3'b000;
    end
endmodule
```

2．case 语句

case 语句是一种多分支的条件语句，完整的 case 语句的格式为：

case（表达式）

选择值 1：　　语句 1；

选择值 2：　　语句 2；

⋮

选择值 *n*：　　语句 *n*；

default：　　语句 *n*+1；

endcase

执行 case 语句时，首先计算表达式的值，然后执行在条件句中找到的与“选择值”相同的分支，执行后面的语句。当表达式的值与所有分支中的“选择值”不同时，执行“default”后的语句，“default”语句如果不需要则可以省略。

case 语句多用于数字系统中的译码器、数据选择器、状态机及微处理器的指令译码器等电路的描述。

举例：用 case 语句设计一个四选一的数据选择器。

四选一数据选择器的逻辑符号如图 5-6 所示，其逻辑功能表如表 5-8 所示。它的功能是：在控制输入信号 s1 和 s2 的控制下，从输入数据信号 a、b、c、d 中选择一个传送到输出 out。

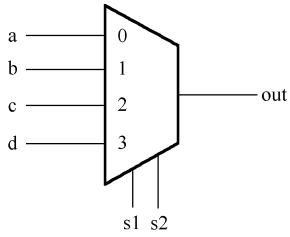

图 5-6　四选一数据选择器逻辑符号

表 5-8　数据选择器逻辑功能表

s2	s1	out
0	0	a
0	1	b
1	0	c
1	1	d

s2 和 s1 有 4 种组合值，可以用 case 语句实现其功能。四选一数据选择器 Verilog HDL 代码如下。

【例 5-9】

```
module mux41(out,a,b,c,d,s1,s2);
input s1,s2;
input a,b,c,d;
output out;
reg out;
always @ (s1 or s2)
    begin
        case ({s2,s1})
            2'b00:          out=a;
            2'b01:          out=b;
            2'b10:          out=c;
            2'b11:          out=d;
        endcase
    end
endmodule
```

5.6.4 循环语句

循环语句包括 for 语句、repeat 语句、while 语句和 forever 语句。

1．for 语句

for 语句的语法格式为：

for（循环指针=初值；循环指针<终值；循环指针=循环指针+步长）

begin

语句；

end

举例：用 for 语句描述 8 位奇偶校验器。a 为输入信号，它是位宽为 8 的矢量。当 a 中有奇数个 1 时，奇偶校验器输出为 1；否则为 0。它的 Verilog HDL 代码如下。

【例 5-10】

```
module parityfor (a,out);
input[7:0]  a;
output out;
reg out;
integer n;
always @ (a)
    begin
        out=0;
        for (n=0;n<8;n=n+1) out=out^a[n];
    end
endmodule
```

2．repeat 语句

repeat 语句的语法格式为：

repeat （循环次数表达式） 语句；

举例：用 repeat 语句实现上例 8 位奇偶校验器，Verilog HDL 代码如下。

【例 5-11】

```
module parityrep (a,out);
parameter size=7;
input[7:0]  a;
output out;
reg out;
integer n;
always @ (a)
    begin
        out=0;
        n=0;
        repeat(size)
            begin
                out=out^a[n];
                n=n+1;
            end
    end
endmodule
```

3．while 语句

while 语句的语法格式为：

while（循环执行条件表达式）

begin

重复执行语句；

修改循环条件语句；

end

while 语句在执行时，首先判断循环执行条件表达式是否为真。若为真，则执行后面的语句；否则，不执行，即循环结束。为了使 while 语句能够结束，在循环执行的语句中必须包含一条能改变循环条件的语句。

举例：用 while 语句实现上例 8 位奇偶校验器，Verilog HDL 代码如下。

【例 5-12】

```
module paritywh (a,out);
input[7:0]  a;
output out;
reg out;
integer n;
always @ (a)
    begin
        out=0;
```

```
        n=0;
        while (n<8)
            begin
                out=out^a[n];
                n=n+1;
            end
    end
endmodule
```

4．forever 语句

forever 语句的语法格式为：

forever

begin

语句;

end

forever 是一种无限循环语句，它不断执行后面的语句或语句快，永远不会结束。forever 语句常用来产生周期性的波形，作为仿真激励信号。例如，产生时钟 clk 的语句为：

```
#10 forever #10 clk=!clk;
```

5.7 Verilog HDL 状态机描述

有限状态机（FSM）及其设计技术是数字系统设计中的重要组成部分，也是实现高效率、高可靠和高速控制逻辑系统的重要途径。从广义上说，任何时序模型都可以归结为一个状态机。

5.7.1 状态机的一般结构

从状态机的信号输出方式上分，有 Moore 型和 Mealy 型两种状态机。Moore 型状态机的输出仅为当前状态的函数，Mealy 型状态机的输出是当前状态和输入信号的函数。从输出时序来看，前者属于同步状态机，而后者属于异步状态机。最常用的 Verilog HDL 程序状态机一般包括说明部分、主控时序过程、主控组合过程和辅助过程等几部分，下面分别予以说明。

1．说明部分

包含状态变量的定义和所有可能状态的说明，必要时还要确定每一状态的编码形式。Verilog HDL 程序状态机的说明部分用参数说明关键词 parameter 来定义各状态，且必须写明各状态的具体取值或编码；接下来再分别定义现态/次态变量：current_state/next_state。例如：

```
parameter [1:0] s0=0,s1=1,s2=2,s3=3;
reg [1:0] current_state,next_state;
```

2．主控时序过程

负责状态机运转和在时钟驱动下状态的转换过程，一般设计比较固定、单一和简单。