第3章Verilog HDL的基本语法

第3章 Verilog HDL的基本语法

本章节介绍Verilog 模块、Verilog HDL基本要素，主要包括标识符、空白符、运算符、数字、关键字、字符串、注释等。Verilog HDL与C语言有许多相似之处，例如分号用于结束每个语句，注释符(/* ... */和//)用法相同，运算符“==”也用来测试相等性。由于Verilog是硬件描述语言的一种，许多概念的物理意义与C语言有所不同，在学习过程中应加以注意。

3.1 Verilog 模块

Verilog的基本设计单元为模块（module），如图3.1模块结构组成图所示，一个模块通常由两部分组成：一部分为接口和数据类型说明，另一部分为逻辑功能描述。

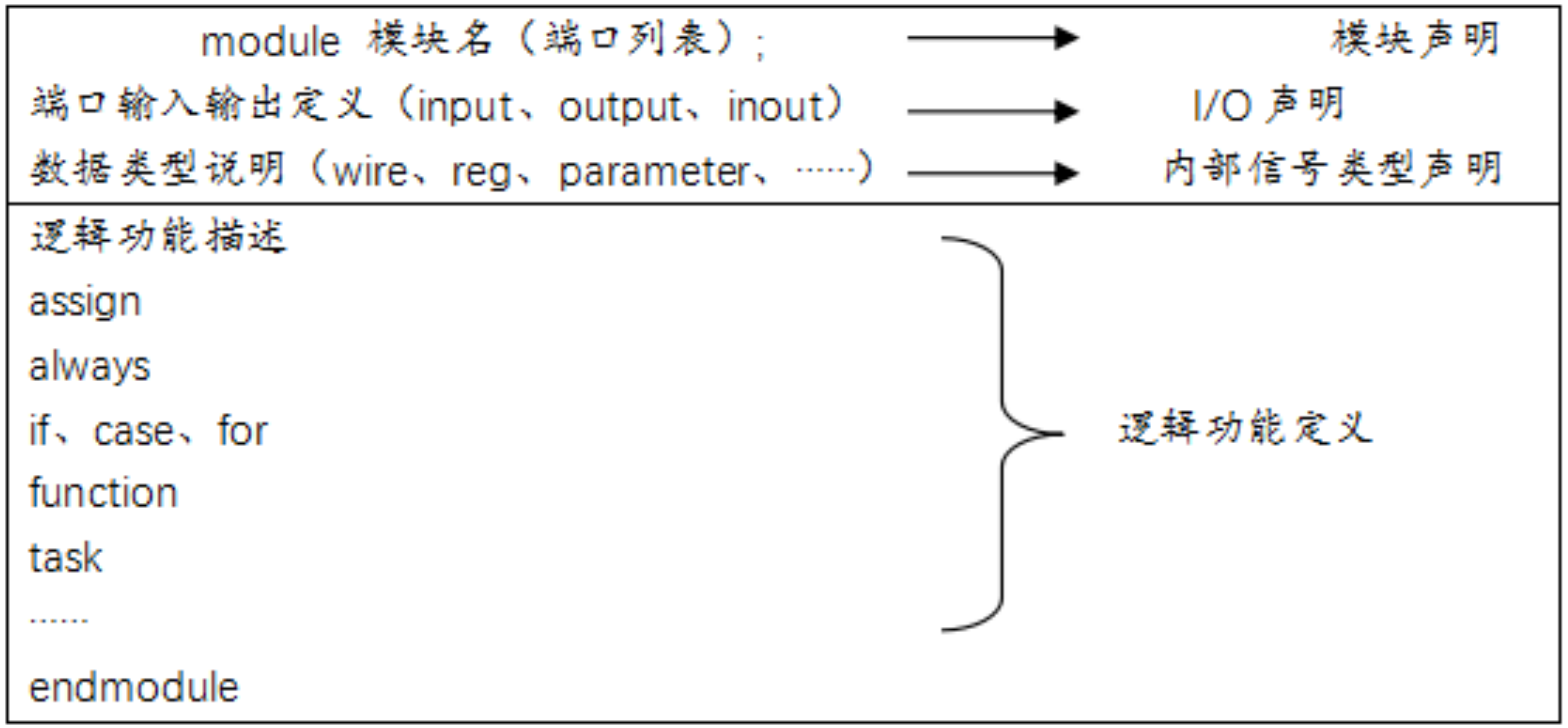

图 3.1 模块结构组成图

从图3.1可看出Verilog模块结构是包含在module和endmudule关键之间，而且每个Verilog程序包括4个主要部分：模块声明、端口定义、信号类型声明和逻辑功能定义。

3.1 Verilog 模块

【例3.1】二输入与非门的Verilog HDL模块的描述。

```
module and_or(a,b,c);          //模块名为and_or,端口有a,b,c
    input a,b;                 //声明a,b为输入端口
    output c;                  //声明c为输出端口
    wire d;                    // 声明内部信号d
    assign d=a&b;              //描述功能
    assign c=!d;               //描述功能
endmudule                      //模块结束
```

3.1.1 Verilog HDL 模块声明

模块声明包括模块名字、模块输入、输出端口列表。模块定义格式如下：

```
module 模块名（端口1，端口2，… …）；
…
…
endmodule
```

其中，模块名字是该模块唯一标识，名字后面的括号内容为端口列表定义，端口名定义必须符合标识符命名规则，端口名之间用逗号隔开，模块声明以分号结束。

3.1.2 Verilog HDL 端口定义

端口是模块与外部其他模块进行信号传递的通道或通讯信号线，其类型有三种：输入端口（input）、输出（output）、双向端口（inout）。每个模块要先对端口进行定义，说明端口的类型，再对模块功能进行描述。

端口定义的语法格式为：

```
input   端口1，端口2，……端口n;       //输入端口
output  端口1，端口2，……端口n;      //输出端口
inout   端口1，端口2，……端口n;      //双向端口
```

端口定义时需注意以下几点：

（1）端口名之间用逗号分割。

（2）每个端口不仅需要声明是输入、输出或者双向端口类型，还要声明其数据类型，是wire型变量或reg型变量或其他数据类型。如不声明端口的数据类型，则端口**默认为网络类型wire**。

（3）输入端口（input）和双向端口（inout）**不允许声明为寄存器类型**。

（4）模块仿真测试文件不需要定义端口。

3.1.3 Verilog HDL 内部信号类型声明

对模块中所有信号都必须进行数据类型的定义，这些信号主要包括端口信号和节点信号。下面举例说明典型的几种信号类型定义。

1、端口定义、数据类型分开定义。

```
module count(a,b,c);
  input a,b;
  output c;
  wire[3:0] a,b;      //定义输入信号a、b为4条总线组成的wire型变量
  reg[3:0] c;         //定义信号c为4条总线组成的reg型变量
  wire d;             //定义信号节点d的数据类型为wire型变量
  … …
endmodule
```

2、端口定义和数据类型写在同一条语句中。

```
module count(a,b,c);
  input wire[3:0] a,b; //定义输入信号a、b为4条总线组成的wire型变量
  output reg[3:0] c;     //定义信号c为4条总线组成的reg型变量
  wire d;               //定义信号节点d的数据类型为wire型变量
  … …
endmodule
```

3.1.3 Verilog HDL 内部信号类型声明

3、端口定义和数据类型放在端口列表中。

```
module count(input wire[3:0] a,b,output reg[3:0] c);
 wire d;
 … …
endmodule
```

以上三种书写风格中，第三种在书写形式上更加简洁，端口类型和信号类型放在模块列表中声明后，在模块内部就不需要再重复声明。

【例3.2】用Verilog HDL对二选一数据选择器的描述。

```
module MUX2_1(input a,b,sel,output out);
      assign out=sel?b:a;
endmudule
```

3.1.4 Verilog HDL 逻辑功能定义

模块最核心的部分是逻辑功能定义，可以使用数据流、行为描述和元件例化三种不同风格定义和描述逻辑电路的功能。

1、数据流描述风格

数据流描述风格通常采用连续赋值语句（assign）对电路的逻辑功能进行描述，通过说明数据的流向对模块进行描述。数据流描述方式比较适合对组合逻辑电路进行描述。

例如：

```
module aor(a,b,c,out);
    input a,b,c;
    ouput out ;
    assign out=~((a&b)|(~c));
endmodule
```

使用assign 赋值时，赋值语句放在assign语句后面即可。

3.1.4 Verilog HDL 逻辑功能定义

2、行为描述风格

行为描述风格使用过程块结构对电路功能进行描述，过程块通常有initial和always语句等。过程块结构内部是行为语句，如过程赋值语句、if语句、case语句等。行为描述使用比较抽象的高级程序语句来描述逻辑功能，不涉及实现该模块的具体硬件电路结构。过程块always使用频率较高，该语句既可描述组合电路，也可描述时序电路。

例如：

```
module aor(a,b,c,out);
input a,b,c;
ouput reg out ;
    always @ (a,b,c)
    begin
        out=~((a&b)|(~c));
    end
endmodule
```

3.1.4 Verilog HDL 逻辑功能定义

3、元件调用（元件例化）

元件调用是指调用底层模块或调用基本门级元件的方法，即调用已经定义好的低层次模块或Verilog内部预先定义好的门级元件对模块逻辑功能进行描述。只使用门级元件描述的电路也称为门级描述方式。

例如：

```
module instance(a,b,c,out);
input a,b,c;
ouput out ;
    wire w1,w2,w3;
    and(w1,a,b);          //调用与门
    not(w2,c),(out,w3);    //调用非门
    or (w3,w1,w2);        //调用或门
endmodule

module top(a,b,c,out);
input a,b,c;
ouput out ;
instance my_instance (a,b,c,out); //调用底层模块
endmodule
```

使用上述三种方式的描述都可以实现组合逻辑电路的逻辑功能，并且使用数据流和行为描述，经过综合工具综合后，结果一般都是门级结构描述。

3.2基本语法要素

Verilog代码是由大量的基本语句构成，其中基本语法元素包括空白符(White Space)、标识符（Identifier）、关键字(Key Word)、数字(Number)、运算符(Operator)、字符串(String)、注释(Comment)等。

3.2.1 空白符

空白符又称间隔符，在Verilog HDL代码中，空白符主要包括空格字符(\b)、制表符(\t)、换行符(\n)组成。这些空白符起到分隔作用，使代码整齐美观，便于代码的阅读和修改。Verilog HDL中的空白符只用于分隔标识符，不会被编译，但在字符串中这些空白符是有意义的。Verilog HDL代码可以单行书写，也可以多行书写。

【例3.3】单行书写格式。

```
module  mux2_1(a,b,s,y);input a,b,s;output y; assign y=(s==0)?a:b; endmodule
```

【例3.4】多行书写格式。

```
module  mux2_1( a, b , s , y );
    input  a, b, s;
    output  y;
    assign  y= (s == 0 )  ?  a : b;
endmodule
```

多行书写格式中加入了空格、换行等字符后，代码更加整齐美观，便于程序员阅读和修改代码。

3.2.2 标识符

标识符是程序代码编写时为模块、端口、函数、常量等元素定义的名称。程序员可以通过标识符访问或修改程序中的模块、函数、端口等对象。**在Verilog HDL语言中，合法标识符由字母、数字、下划线（_）和符号（$）组成。**标识符不能与关健词同名，而且首字符必须是字母(a~z,A~Z)或下划线(_)。另外，Verilog语言中标识符区分大小写，最长可达1023个字符。

【例 3.5】判断以下标识符的正确性。

```
（1）clk_1              //合法
（2）clk_1              //合法，（1）和（2）中的标识符为两个不同的标识符。
（3）6class             //非法，不允许以数字开头
（4）_$_$               //合法，但不建议起没有意义的名字
（5）Sec_l@             //非法，名字中带有非法字符@
```

转义标识符是为了解决不能以数字、美元符号开头和不能包含其他打印字符等缺陷。转义标识符以反斜杠“\”开始，以空白符（空格、制表符Tab、换行符）结束。这样转义标识符中可以包含任何可打印字符，注意：反斜杠和空白符不属于转义标识符的一部分。

【例 3.6】转义标识符。

```
（1）  \4S
（2）  \80.0￥
（3）  \sum        // 等同于标识符sum
```

3.2.3 关键字

Verilog HDL定义了一系列关键字（保留字）。关键字都是小写比如：if-else，用户定义的标识符不能与之相同。这些保留字用户不能作为变量或节点名字使用。**关键字都是小写的。**

if-else,case-endcase ,always,for,forever,fuction-endfuction,task-endtask,not,and,or,xor,buf,integer,reg,wire...

3.2.4 注释

为了增加程序的可读性，有必要在代码中加入注释，方便拥护对程序的阅读和修改。注释的方法有两种：单行注释和多行注释。

单行注释：以“//”开始，到本行行尾结束。

多行注释：以“/*”开始，到“*/”结束。

注意：凡是注释的内容均不会进行编译。多行注释不能嵌套使用，而单行注释可以嵌套在多行注释里面。

【例 3.7】单行注释与多行注释举例。

```
（1） a=b+c;  //单行注释

（2）/*  多行
注释       */

（3）/*   /*  多行注释嵌套
                为不合法的注释 */   */

（4）/*    //单行注释嵌套在多行注释里面
             //为合法的注释
 */
```

3.3常量

在程序运行的过程中，其值不能被改变的量称为常量。在Verilog HDL中主要常量类型有整数、实数和字符串。整数可以被综合，而实数和字符串不能被综合。

3.3.1 整数型常量

1、进制表示形式

在Verilog HDL中，整数型常量主要有四种进制表示形式：
（1）二进制整数，用字母b或B表示。
（2）十进制整数，用字母d或D表示。
（3）十六进制整数，用字母h或H表示。
（4）八进制整数，用字母o或O表示。

缺省字母时，默认为十进制整数。

3.3.1 整数型常量

2、整数表达方式

在Verilog HDL中，整数书写格式如下：

±<位宽>'<进制表示形式><数值>

位宽表示所写整数的二进制位数；进制是数字的标识，是指定数值的格式，不区分大小写，如二进制的表示用b和B含义相同；数值是基于进制的数字序列，其中数字的x(未知状态)和z（高阻态）以及十六进制中的a到f不区分大小写。

注：撇号和进制之间不能有空格，也不允许插入其他符号。

非对齐宽度整数的处理，遵循以下规则：

① 当位宽小于实际位数时，截断相应的高位部分。

② 当位宽大于实际位数时，且数值的最高位是0或1时，相应的高位部分补0。

③ 当位宽大于实际位数时，且数值的最高位（最左边位）是x或z时，相应的高位部分补x或者z。

④ 如果未指定位宽，则默认位宽为32位，如实际位宽大于32位，则按实际位宽计算。

注：符号？可以替代高阻态Z，在数字表示中，？与Z（z）等价。

在数字之间，可以加入下划线符号“_”，提高可读性。

3.3.1 整数型常量

【例3.8】以下为指定位宽整数书写的例子。

① 8'B1110_1011　//位宽为八位的二进制数1110_1011
② 8'h5f　　//位宽为八位的十六进制数5f
③ 6'o71　　//位宽为六位的八进制数71
④ 4'b11_0111　//位宽小于实际位宽，最高位两位二进制11被截掉
⑤ 10'hz1　　//位宽大于实际位宽且最高位为z，
　　则左边补z，等价于10'bzz_zzzz_0001
⑥ 8'b1?　　//位宽大于实际位宽且最高位为1，则左边补0，等价8'b0000001z

【例3.9】以下为未指定位宽整数书写的例子。

① 599　　//未指定位宽和进制，默认为十进制整数599，
② 'H56_DF　　//十六进制数56_DF，默认32位的位宽
③ 'o562　　//八进制数562，默认32位的位宽
④ 'HEDA_8760_8760　　//十六进制数EDA_8760_8760，按实际44位宽算

3.3.2 实数型常量

实数有十进制表示法和科学计数法两种。

注：十进制小数点两边至少各要有一个数字。

1、十进制表示法

【例3.10】有效实数常量表示的例子。

```
3.14
0.5
3.0
8.                          //无效的实数常量
.1                          //无效的实数常量
```

2、科学计数表示法

【例3.11】有效实数常量表示的例子。

```
876e2                       //等同87600
0.1e-0                      //等同0.1
2.3223_6554e4               //等同23223.6554,用下划线隔开
3.e2                        //无效的实数常量
.5e2                        //无效的实数常量
```

3.3.2 实数型常量

3. 实数到整数转换

Verilog HDL语法规定，通过四舍五入的方法将实数转换为整数。

【例3.12】将实数转为整数。

95.49　与95.449　转为整数后都为95
95.5　与95.9　转为整数后都为96
-95.49　与-95.449　转为整数后都为-95
-95.5　与-95.9　转为整数后都为-96

3.3.3 字符串

字符串是用双引号括起来的字符序列，字符串数值的表示为每一个字符所对应的ASCII码值。例如：字符串“abc”等价于24'h616263。字符串变量是寄存器型变量，它的位宽等于字符串的字符个数乘以8。

例如：

```
module string_test;
reg[8*6:0] str1;
        initial
        begin
            str1=“china”;
            $display(“%s=%h”,str1,str1);
            #200;
            str1=“china!!”;
            $display(“%s=%h”,str1,str1);
        end
endmodule
```

输出结果为

```
china=006368696e61
china!!=68696e612121
```

字符串主要是用于仿真，例如显示一些相关信息。

3.4数据类型

数据类型是用来表示数字电路中的物理连线、数据存储和传输单元类型。Verilog HDL的数据类型有很多，本节介绍逻辑状态、网络、寄存器、向量四种数据类型。

3.4.1 逻辑状态

Verilog HDL的信号逻辑状态有以下四种：

（1）0：低电平、逻辑0或者逻辑假；

（2）1：高电平、逻辑1或者逻辑真；

（3）x或X：未知状态或不确定；

（4）z或Z：高阻态。

未知状态X和高阻态Z都不区分大小写。比如：8'Hzx与8'HZX等价。

3.4.2 网络类型

网络型(net)数据表示硬件电路之间的物理连线，该类型的变量不保存值，需要门或者模块的不断驱动，其输出的值始终紧随输入的变化而变化。通常net型变量有以下两种驱动方式：一种是将门电路元器件或者模块输出端连接到该变量（元件例化）；另一种是用assign持续赋值语句对该变量赋值。

注：网络型（net）变量没有连接到驱动时，其值为高阻态z（除trireg以外，trireg默认初始值为x）。

常用net型变量如表3.1所示，其中“√”表示可综合。本小节主要介绍wire和tri两种网络类型。

表 3.1 常用的网络型变量

类　型	功　能	可综合性
wire/tri	内部连线类型	√
supply1/supply0	电源/地	√
wor/trior	多驱动连线，具有线或特性	X
wand/triand	多驱动连线，具有线与特性	X
tri1/tri0	无驱动时上拉/下拉状态	X
trireg	具有电荷保存，可存储数值	X

3.4.2 网络类型

1、wire型变量

wire型变量是最常用的net型变量，在verilog模块中定义输入/输出端口时，**如果没有声明端口的数据类型，则默认为wire型变量。**当输入端口声明为wire型信号时，在assign语句描述和组合逻辑电路中，输出端口也常声明为wire型变量。wire型变量的取值可以是0、1、X、Z，**如果wire型变量没有连接驱动或者初始化时，其值为高阻态Z。**

wire型变量定义格式如下：

wire[msb:lsb] 数据名1，数据名2，…，数据名n;

注：[ ]可以省略，省略[ ]表示变量的宽度为一个位。

【例3.13】定义1位宽wire型变量a和b。

```
wire a,b;
```

【例3.14】定义10位宽wire型变量a、b、c。

```
wire[9:0] a,b,c;
wire[10:1]a,b,c;
```

3.4.2 网络类型

2、tri(三态)网络类型

wire型变量和tri型变量在功能、语法上用法一致。tri只是为了增加程序的可读性，可以清楚表示信号电路连线具有三态功能。三态网络可以用于描述多个驱动源驱动同一根线的网络类型。如果多个驱动源驱动一个连线，则确定网络的有效值如表3.2所示。

表 3.2 wire 和 tri 的真值表

wire/tri	0	1	X	Z
0	0	X	X	0
1	X	1	X	1
X	X	X	X	X
Z	0	1	X	Z