VerilogHDL 入门

陈铖

2017.09.21

一个 VerilogHDL 程序描述一个数字电路功能模块（module），以关键字 module 开始，endmodule 结束。一个模块实际上就是一个元件，可以小到是一个逻辑门，也可以大到是一个复杂的数字系统。模块是可以被其它模块调用（称之为实例化 instance）的，这样就支撑了自顶向下模块化设计方法。我们可以将一个较大数字系统划分成若干功能相对简单的子模块，子模块分别设计实现，然后再集成子模块实现数字系统。

1　VerilogHDL 程序基本格式

VerilogHDL 程序基本格式如下：

```
module　模块名称（端口信号声明）；
//端口信号定义
//内部信号定义
//功能描述
endmodule
```

括号内声明的端口信号就是该模块的输入和输出信号，也就是该模块与外部其它模块连接的引脚。括号内只列举信号的名称，相互间以英文逗号隔开。注意 module 行最后以英文分号结束。

“端口信号定义”主要定义已经声明的端口信号的传输方向、数量和类型。信号传输方向有三种，关键字分别是：output（输出），input（输入），inout（双向传输）。数量定义主要用于一组序号连续的信号定义，方括号内英文冒号隔开的两个数字分别是连续序号的始序和止序。强烈建议按降序方式书写！如果不带方括号只写信号名称，表示引用整组信号。而带方括号的信号名称表示引用其中一个或连续几个信号。定义时没有方括号，即不定义数量，默认信号位宽为 1。

信号类型主要有两种，关键字分别是：wire（连接线）和 reg（寄存器）。wire 主要用于定义没有记忆功能的连接线，是默认的信号类型。reg 主要用于定义需要保持不变的信号，具有记忆功能。比如触发器、ROM 单元等的定义。但是，reg 类型不一定是存储单元，因为在 always 结构体中描述语句必须使用 reg 型。内部信号定义主要是模块功能描述时需要用到的除了端口信号以外的信号定义，类型也主要有 wire（连接线）和 reg（寄存器）型。

2　VerilogHDL 功能模块描述方式

模块功能描述主要有三种方式：行为描述方式、结构描述方式和数据流描述方式。具体使用哪一种取决于使用方便。

2.1 行为描述方式

一般的，行为描述方式是最常用的方式，比如二选一数据选择器的行为方式描述的 VerilogHDL 程序如下：

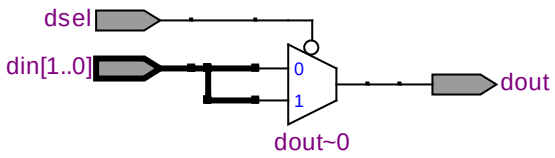
```
module mux21_1(din,dout,dsel);
   input [1:0] din;
   input dsel;
   output dout;
   reg dout;
   always @(din, dsel)
   begin
      if (dsel==0) dout = din[0];
      else dout = din[1];
   end
endmodule
```

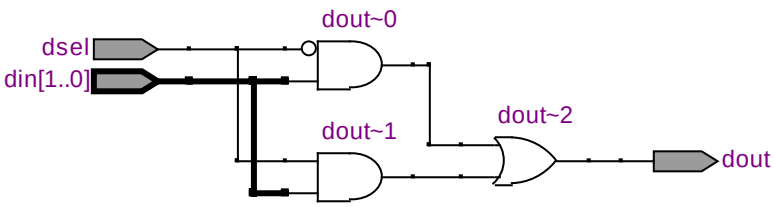
功能描述部分的语句看起来很像 C 语言，因此有一定 C 语言基础的人就可以进行 VerilogHDL 程序设计。但是，必须强调 VerilogHDL 不是计算机语言，而是描述数字电路的硬件描述语言！进行 VerilogHDL 程序设计时心中要有电路，要防止计算机程序串行顺序执行的固有思维方式，还要注意可综合可实现性。

行为描述方式不止上面一种，下面的程序也是行为描述方式：

```
module mux21_2(din,dout,dsel);
   input [1:0] din;
   input dsel;
   output dout;
//    reg dout;
   assign dout = (~dsel)&din[0] | dsel&din[1];
endmodule
```

上述两种行为描述方式的 RTL 电路原理图分别如下：

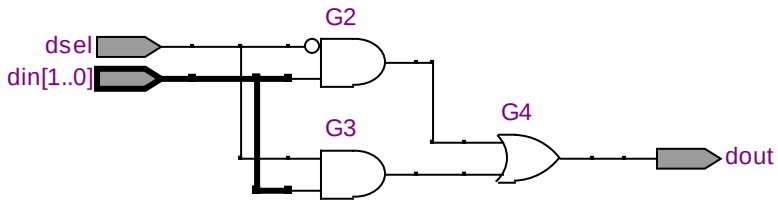




2.2 结构描述方式

比较这两个 RTL 电路原理图可以发现，这两个电路似乎有所不同而实际又相同。图 2 电路中可以直观看到与门、或门等逻辑门，我们可以根据这个电路原理图描述该电路，这种方式称为结构描述方式。程序清单及其 RTL 电路原理图分别如下：

```
module mux21_3(din,dout,dsel);
   input [1:0] din;
   input dsel;
   output dout;
//   reg dout;
   not G1(G1out, dsel);
   and G2(G2out, G1out, din[0]);
   and G3(G3out, dsel, din[1]);
   or G4(dout, G2out, G3out);
endmodule
```



结构描述方式中的语句主要是实例化（类似于调用）库中已有的模块元件，这里是三个基本逻辑门：与门（and）、或门（or）、非门（not），常用逻辑门还有与非门（nand）、或非门（nor）、异或门（xor）。

模块元件实例化的语句格式如下：

模块元件名 例化元件名（端口信号列表）；

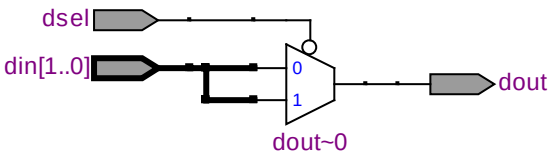
模块元件可以是 QuartusII 内置的库元件，也可以使用户自己设计的模块。例化元件是当前电路中的一个具体应用。例化元件名称有时可以省略，但建议不要省。括号中列表的端口信号是指当前例化元件的端口信号，它们与模块元件的端口信号之间的关联连接有两种方式：位置关联和名称关联。前者适合于例化内置库元件，后者适用于例化用户模块元件。上述结构描述方式中的逻辑门例化语句就是位置关联，内置的逻辑门都是多输入单输出电路，信号列表中的第一个位置是输出信号，后面的都是输入信号，声明列举了几个信号门电路就有几个输入。名称关联信号连接方式的模块元件例化语句举例如下：

假设已有模块 element1(A,B,C)，名称关联的例化语句为 element1 element2(.A(X), .B(Y), .C(Z));该语句的意思是使用一个新元件叫 element2，其功能及端口信号与模块元件 element1 完全相同。并且例化元件 element2 的端口信号 X,Y,Z 分别与模块元件 element1 端口 A,B,C 相连！当然，程序中所有同名信号都是相连的，实际上是一个信号。

2.3 数据流描述方式

功能模块的描述方式还有一种数据流描述方式，示例程序清单及 RTL 电路原理图如下：

```
module mux21_4(din,dout,dsel);
   input [1:0] din;
   input dsel;
   output dout;
//    reg dout;
   assign dout = (dsel==0)? din[0]:din[1];
endmodule
```



3　VerilogHDL 基本语法

3.1 信号值

VerilogHDL 中规定了信号取值有 4 种：0, 1, x, z，分别表示低电平、高电平、未知不确定状态和高阻态。这里 x 和 z 大小写都可以。

总线信号赋值时使用多位二进制数，应该标明位数。如：data[7:0] <= 8'b0000_0000;语句中的数字“8”表明后面二进制数的位数是 8 位。语句中的“'b”标明后面的额数字序列是二进制，如果是十六进制则应该是“'h”形式，如果是十进制则应该是“'d”形式。这里 b, h, d 大小写也都可以。但不管后面数字序列用哪种进位制表示，前面的数字都指定后面数字的二进制位数。后面数字序列中的下划线仅为方便阅读，可以在任何位置，也可以有任意多个，不影响实际数字序列。如果后面数字序列实际位数不同于前面指定的数字，则高位自动被截断或补 0。

3.2 运算符

序号	类别	运算符	示例	功能说明
1	算术运算	+	a+b	加法
2	算术运算	-	a-b	减法
3	算术运算	*	a*b	乘法
4	逻辑运算	&	a&b	与，逻辑运算按位进行
5	逻辑运算	|	a|b	或
6	逻辑运算	~	~a	非
7	逻辑运算	^	a^b	异或
8	移位运算	<<	a<<n	a 左移 n 位，移空位补 0
9	移位运算	>>	a>>n	右移

3.3 关系符

关系运算结果为真（1）或假（0）。

序号	类别	关系符	示例	功能说明
1	大小关系	>	a>b	大于，关系运算结果为真（1）或假（0）
2	大小关系	<	a<b	小于
3	大小关系	>=	a>=b	大于等于，不小于
4	大小关系	<=	a<=b	小于等于，不大于
5	大小关系	==	a==b	等于
6	大小关系	!=	a!=b	不等于
7	逻辑关系	&&	a&&b	与，逻辑关系运算结果为 1 或 0
8	逻辑关系	||	a||b	或
9	逻辑关系	!	!a	非

3.4 位的拼接组合

花括号“{}”可以把若个较短位信号拼接组合成较宽的总线信号，比如：

d[16:0] = {a[7:0], b[3:0], c[3:0]};

语句中花括号的作用是把 8 位宽的信号 a 和 4 位宽的信号 b 和 4 位宽的信号 c 拼接组成一个 16 位宽的信号，该赋值语句的功能是把拼接好的信号赋值给信号 d。结果是 d[16:8] = a[7:0]，d[7:4] = b[3:0]，d[3:0] = c[3:0]。

4 VerilogHDL 语句

4.1 块语句 begin/end，fork/join。

块语句有两种，一是串行顺序块语句 begin/end，二是并行块语句 fork/join。实际上关键字 begin/end 和 fork/join 就相当于一对括号的左右两个括号，功能是将括号内的语句当成一个整体语句。两者的区别是 begin/end“括号”内语句是按顺序串行执行的，而 fork/join“括号”内语句则是并行执行，与书写顺序无关。

4.2 阻塞赋值和非阻塞赋值

阻塞赋值符号是“=”，非阻塞赋值符号是“<=”。阻塞赋值主要用于组合逻辑电路描述，非阻塞赋值主要用于时序逻辑电路赋值。

4.3 连续赋值语句 assign

assign 信号名 = 表达式;

连续赋值语句 assign 主要用于组合逻辑电路描述，因此 assign 语句中应该用阻塞赋值“=”。

连续赋值的含义是“=”右边表达式任一个变量有变化，表达式立即被计算，计算的结果立即赋给左边信号。注意连续赋值语句之间是并行的，多条 assign 语句与书写顺序无关。

4.4 过程赋值语句 always

always @(敏感信号)

begin ... end

过程赋值语句 always 主要用于时序逻辑电路描述，当然也可以描述组合逻辑电路。描述时序逻辑电路时，信号赋值一定要用非阻塞赋值“<=”。描述组合逻辑电路时，则可以用阻塞赋值“=”给信号赋值。敏感信号实际上是 begin … end 顺序块执行的触发条件，条件每满足一次即每触发一次，顺序块被执行一次。敏感信号通常是某信号的边沿（上升沿关键字 posedge，下降沿关键字 negedge），当然也可以使是电平。敏感信号如果有多个，则用关键字 or 分隔，其中一个条件满足即触发顺序块被执行。

4.5 过程赋值语句 initial

initial 语句的 begin ... end 顺序块只被执行一次，常用于信号初始化，或产生特定的测试信号波形。

4.6 条件语句 if...else...

条件语句格式和用法同 C 语言，如 if (条件) 语句 1; else 语句 2; 再如(条件) ？表达式 1:表达式 2; 条件成立，则执行语句 1 或赋值表达式 1，否则执行语句 2 或赋值表达式 2。

4.7 多分支语句 case

多分支语句 case 格式和用法同 C 语言。如果分支条件中有不确定值 x，则关键字改用 casex。