Verilog HDL语言规范

主 讲：何宾

Email：hebin@mail.buct.edu.cn

2014.06

Verilog HDL程序结构

描述复杂的硬件电路，设计人员总是将复杂的功能划分为简单的功能，模块是提供每个简单功能的基本结构。

1. 设计人员可以采取“自顶向下”的思路，将复杂的功能模块划分为低层次的模块。

2. 自顶向下的设计方式有利于系统级别层次划分和管理，并提高了效率、降低了成本。

Verilog HDL程序结构

使用Verilog描述硬件的基本设计单元是模块（module）。

- 复杂电子电路的构建，主要是通过模块的相互连接调用来实现的。
- 在Verilog中，将模块包含在关键字module、endmodule之内。
- Verilog中的模块类似C语言中的函数，它能够提供输入、输出端口，通过例化调用其他模块。
- 该模块可以被其它模块例化调用，模块中可以包括组合逻辑部分和时序逻辑部分。

Verilog HDL程序结构

一个模块通过它的端口（输入/输出端口）为更高层的设计模块提供必要的连通性，但是又隐藏了其内部的具体实现。

这样，在修改其模块的内部结构时不会对整个设计的其余部分造成影响。

Verilog HDL程序结构

Module 模块名（端口列表）

端口定义
input 输入端口
output 输出端口
inout 输入/输出端口

数据类型说明
wire
reg
parameter

逻辑功能定义
assign
always
function
task
……

endmodule

Verilog HDL程序结构

Verilog结构位于在module和endmodule声明语句之间，每个Verilog程序包括端口定义、数据类型说明和逻辑功能定义部分。

- 模块名

  是模块唯一的标识符。

- 端口列表

  是由模块各个输入、输出和双向端口组成的一个端口列表。

Verilog HDL程序结构

- 数据类型说明

用来说明模块内用到的数据对象是网络类型还是变量类型。

- 逻辑功能定义

通过使用逻辑功能语句实现具体的逻辑功能。

Verilog HDL程序结构

对于Verilog语言来说，有下面的特征：

- 每个Verilog HDL程序源文件都以.v作为文件扩展名。
- Verilog HDL区分大小写，也就是说大小写不同的标识符是不同的。
- Verilog HDL程序的书写与C语言类似，一行可以写多条语句，也可以一条语句分成多行书写。
- 每条语句以分号结束，endmodule语句后不加分号。
- 空白（新行、制表符和空格）没有特殊意义。

Verilog HDL程序结构
--模块声明

模块声明包括模块名字，模块的输入和输出端口列表。模块的定义格式如下：

```
module <module_name>(port_name1,…,port_namen);
    ….
    ….
    ….
endmodule
```

其中：

- module_name为模块名，是该模块的唯一标识。
- port_name为端口名，这些端口名使用“，”分割。

Verilog HDL程序结构
--模块端口定义

端口是模块与外部其它模块进行信号传递的通道（信号线），模块端口分为输入、输出或双向端口。

1．输入端口的定义格式

input <input_port_name>, ...<other_inputs>...;

其中：

- **input**为关键字，用于声明后面的端口为输入端口。
- **input_port_name**为输入端口名字。
- **other_inputs**为用逗号分割的其它输入端口的名字。

Verilog HDL程序结构
--模块端口定义

2．输出端口的定义格式

output <output_port_name>,...<other_outputs>...;

其中：

- output为关键字，用于声明后面的端口为输出端口。
- output_port_name为输出端口名字。
- other_outputs为逗号分割的其它输出端口的名字。

Verilog HDL程序结构
--模块端口定义

3．输入输出端口（双向端口）的定义格式

inout <inout_port_name>,...<other_inouts>...;

其中：

- **inout**为关键字，用于声明后面的端口为输入输出类型的端口。
- **other_inouts**为输入/输出端口的名字。
- **other_inouts**为逗号分割的其它输入/输出端口的名字。

Verilog HDL程序结构
--模块端口定义

在声明端口的时候，除了声明其输入/输出外，还需要注意以下几点：

- 在声明输入端口、输出端口或者输入输出端口时，还要声明其数据类型。
  - ◆对于端口来说，可用的数据类型是网络型（net）或者寄存器（reg）型。
  - ◆当没有明确指定端口类型时，将端口默认为网络类型。

Verilog HDL程序结构
--模块端口定义

- 可以将输出或输入端口重新声明为寄存器类型。无论是在网络类型说明还是在寄存器类型说明中，网络类型或寄存器类型必须与端口说明中指定的宽度相同。
- 不能将输入和双向端口指定为寄存器类型。

Verilog HDL程序结构
--模块端口定义

例：端口声明实例

```
module test(a,b,c,d,e,f,g,h);
input [7:0] a;              // 没有明确的说明–网络是无符号的
input [7:0] b;
input signed [7:0] c;
input signed [7:0] d;       // 明确的网络说明-网络是有符号的
output [7:0] e;             // 没有明确的说明–网络是无符号的
output [7:0] f;
output signed [7:0] g;
output signed [7:0] h;      // 明确的网络说明-网络是有符号的
```

Verilog HDL程序结构
--模块端口定义

```
wire signed [7:0] b;  // 从网络声明中，端口b继承了有符号的属性
wire [7:0] c;         // 网络c继承了来自端口的有符号的属性
reg signed [7:0] f;   // 从寄存器声明中，端口f继承了有符号的属性
reg [7:0] g;          // 寄存器类型g继承来自端口的 有符号的属性
endmodule
```

Verilog HDL程序结构
--模块端口定义

注：

在verilog中，也可以使用ANSI C风格进行端口声明。

这种风格的声明的优点是避免了端口名在端口列表和端口声明语句中的重复。

如果声明中未指明端口的数据类型，那么默认端口具有wire数据类型。

Verilog HDL程序结构
--模块端口定义

例： ANSI C风格的端口说明实例

```
module test (
input [7:0] a,
input signed [7:0] b, c, d,  // 多个共享属性的端口，可以一起声明
output [7:0] e,        //在每个端口声明中，单独 声明每个端口的属性
output reg signed [7:0] f, g,
output signed [7:0] h) ;
endmodule
```

Verilog HDL程序结构
--逻辑功能定义

逻辑功能定义是Verilog程序结构中最重要的部分，逻辑功能定义用于实现模块中的具体的功能。

在逻辑功能定义部分可以有多种方法来实现逻辑功能，主要的方法有：

- 分配实现逻辑定义

分配是最简单的逻辑功能描述，用assign分配语句定义：

```
assign F=~((A&B)|(~(C&D)));
```

Verilog HDL程序结构
--逻辑功能定义

模块调用

所谓模块的调用，是指从模块模板生成实际的电路结构对象的操作。

这样的电路结构对象被称为模块实例，模块调用也被称为实例化。每一个实例都有它自己的名字、变量、参数和I/O接口。

一个Verilog模块可以被任意多个其他模块调用。

Verilog HDL程序结构
--逻辑功能定义

在Verilog HDL语言中，模块不能被嵌套定义，即在一个模块的定义内不能包含另一模块的定义。但是却可以包含对其他模块复制，即调用其他模块的例化

模块的定义和模块的例化是两个不同的概念，在一个设计中，只有通过模块调用（实例化）才能使用一个模块。

Verilog HDL程序结构
--逻辑功能定义

例：顶层模块调用底层模块的例子

```
module top;
reg clk;
reg [0:4] in1;
reg [0:9] in2;
wire [0:4] o1;
wire [0:9] o2;
    vdff m1 (o1, in1, clk);
    vdff m2 (o2, in2, clk);
endmodule
```

Verilog HDL程序结构
--逻辑功能定义

在always过程赋值

always块经常用来描述时序逻辑电路。下面先给出一个使用always过程来实现计数器的例子。

```
always @(posedge clk)
begin
    if(reset) out<=0;
    else out<=out+1;
end
```

Verilog HDL程序结构
--逻辑功能定义

函数和任务调用

模块调用和函数调用非常相似，但是在本质上又有很大差别。

- 一个模块代表拥有特定功能的一个电路块，每当一个模块在其他模块内被调用一次，被调用模块所表示的电路结构就会在调用模块代表的电路内部被复制一次（即生成被调用模块的一个实例）。
- 模块调用不能像函数调用一样具有“退出调用”的操作，因为硬件电路结构不会随着时间而发生变化，被复制的电路块将一直存在。

Verilog HDL程序结构
--逻辑功能定义

例：函数调用的例子

```
module tryfact;                    // 定义函数
function automatic integer factorial;
input [31:0] operand;
integer i;
if (operand >= 2)
        factorial = factorial (operand - 1) * operand;
else
    factorial = 1;
endfunction                        // 测试函数
```

Verilog HDL程序结构
--逻辑功能定义

```
integer result;
integer n;
initial
begin
    for (n = 0; n <= 7; n = n+1)
        begin
            result = factorial(n);
            $display("%0d factorial=%0d", n, result);
        end
   end
endmodule
```