Verilog HDL语言规范

主 讲：何宾

Email：hebin@mail.buct.edu.cn

2014.06

Verilog HDL任务和函数

任务和函数提供了在一个描述中，从不同位置执行公共程序的能力。它们也提供了将一个大的程序分解成较小程序的能力。这样，更容易阅读和调试源文件描述。

Verilog HDL任务和函数
--任务和函数的区别

下面给出了任务和函数的区别规则：

- 在一个仿真时间单位内执行函数；一个任务可以包含时间控制的语句。
- 函数不能使能任务。但是，一个任务可以使能其它任务和函数。
- 函数至少有一个input类型的参数，没有ouput或者inout类型的参数；而一个任务可以有零个或者更多任意类型的参数。

Verilog HDL任务和函数
--任务和函数的区别

- 一个函数返回一个单个的值，而任务不返回值。
- 函数的目的是通过返回一个值来响应一个输入的值。一个任务可以支持多个目标，可以计算多个结果的值。
- 通过一个任务调用，只能返回传递的output和inout类型的参数结果。
- 使用函数作为表达式内的一个操作数，由函数返回操作数的值。

Verilog HDL任务和函数
--任务和函数的区别

- 函数定义中，不能包含任何时间控制的语句，比如：#、@或者wait。而任务无此限制。
- 函数定义中必须包含至少一个输入参数。而任务无此限制。
- 函数不能有任何非阻塞分配或者过程连续分配。
- 函数不能有任何事件触发器。

Verilog HDL任务和函数
--任务和函数的区别

一个任务可以声明为下面的格式：

```
switch_bytes (old_word, new_word);
```

一个函数可以声明为下面的格式：

```
new_word = switch_bytes (old_word);
```

Verilog HDL任务和函数
--任务和任务使能

定义任务

- 定义任务的格式一如下：

```
task task_name;
    input automatic input_name;
    more_inputs
    output output_name;
    more_outputs
    begin
      statements;
    end
endtask
```

Verilog HDL任务和函数
--任务和任务使能

其中：

- automatic：可选的关键字，用于声明一个自动的任务，该任务是可重入的，动态的分配每一个并发执行的任务入口。当没有该关键字时，表示一个静态的任务。在层次中，不能访问自动任务条目。可以通过使用它们的层次化名字来调用自动化任务。
- task_name为任务名。
- input_name为输入端口的名字。

Verilog HDL任务和函数
--任务和任务使能

- output_name为输出端口的名字。
- statements为描述语句。
- 任务可以没有或有一个或多个参数。

Verilog HDL任务和函数
--任务和任务使能

任务使能和参数传递

- 一个任务由任务使能语句调用。
- 任务使能语句给出传入任务的参数值和接收结果的变量值。
- 任务使能语句是过程性语句，可以在always语句或initial语句中使用。其语法格式如下：

```
task_name(comma_separated_inputs,comma_separated_outputs);
```

Verilog HDL任务和函数
--任务和任务使能

其中：

- **task_name为任务的名称。**
- **comma_separated_inputs为“，”分割的输入端口的名字。**
- **comma_separated_outputs为“,”分割的输出端口的名字。**

注：

任务使能语句中参数列表必须与任务定义中的输入、输出和输入输出参数说明的顺序匹配。任务可以没有参数存在。

Verilog HDL任务和函数
--任务和任务使能

带有五个参数的task基本结构的Veriog HDL描述的例子

```
task my_task;
input a, b;
inout c;
output d, e;
begin
. . . // 执行任务的语句
. . .
c = foo1; // 分配用于初始化结果寄存器
d = foo2;
e = foo3;
end
endtask
```

Verilog HDL任务和函数
--任务和任务使能

或者采用下面的描述方式：

```
task my_task (input a, b, inout c, output d, e);
begin
. . .       //执行任务的语句
. . .
c = foo1;   //分配用于初始化结果寄存器
d = foo2;
e = foo3;
end
endtask
```

Verilog HDL任务和函数
--任务和任务使能

下面的语句使能任务：

```
my_task (v, w, x, y, z);
```

其中：

- 任务使能参数（v,w,x,y,z）对应于任务所定义的参数（a,b,c,d,e）。
- 在使能任务期间，input和inout类型的参数（a,b,c）接受传递的值（v,w,x）。

Verilog HDL任务和函数
--任务和任务使能

这样，执行任务使能调用产生下面的分配：

```
a=v;

b=w;

c=x;
```

Verilog HDL任务和函数
--任务和任务使能

作为任务处理的一部分，任务定义my_task将计算的结果分配到c,d,e。当任务完成时，下面的分配将计算得到的值，返回到被执行的调用过程。

```
x=c;
y=d;
z=e;
```

Verilog HDL任务和函数
--任务和任务使能

描述交通灯时序task语句Veriog HDL描述的例子

```
module traffic_lights;
reg clock, red, amber, green;
parameter on = 1, off = 0, red_tics = 350,
                    amber_tics = 30, green_tics = 200;
// 初始化颜色.
initial red = off;
initial amber = off;
initial green = off;
always begin                    // 控制灯的时序
```

Verilog HDL任务和函数
--任务和任务使能

```
        red = on;                       // 打开红灯
        light(red, red_tics);           // 等待.
        green = on;                     // 打开绿灯
        light(green, green_tics);       // 等待.
        amber = on;                     // 打开琥珀色灯
        light(amber, amber_tics);       // 等待.
end
//等待'tics'的任务，上升沿时钟
task light;
output color;
input [31:0] tics;
```

Verilog HDL任务和函数
--任务和任务使能

```
begin
        repeat (tics) @ (posedge clock);
        color = off;                    // 关闭灯.
end
endtask
always begin                            // 时钟波形.
        #100 clock = 0;
        #100 clock = 1;
end
endmodule                               // traffic_lights.
```

Verilog HDL任务和函数
--任务和任务使能

任务存储器使用和并行运行

- 一个任务可以多次并行使能。将并行调用的每个自动任务的所有变量进行复制，用于保存该调用的状态。
- 静态任务的所有变量是静态的，即：在一个模块例化中，有一个单个的变量对应于每个声明的本地变量，而不管并行运行的任务个数。
- 然而，对于静态任务来说，一个模块的不同例化，将有用于每个例化的单独存储。

Verilog HDL任务和函数
--任务和任务使能

在静态任务里声明的变量，包含：input,output和inout类型的参数。在调用时，将保留它们的值。

- 在自动任务里声明的变量，包含：output类型的变量，
- 当进入任务时，将其初始化为默认的初始化值。
- 根据任务使能语句中所列出的参数，初始化input和inout类型参数。

Verilog HDL任务和函数
--任务和任务使能

因为在自动任务中声明的变量在任务调用结束时，进行分配解除。因此，它们不能用于某个结构中，这个结构在该点后可以使用到它们：

- 不能使用非阻塞分配或者过程连续分配，进行值的分配。
- 过程连续分配或者过程force语句，不能引用它们。
- 在内部分配事件控制的非阻塞分配，不能引用它们。
- 系统任务$monitor和$dunpvars不能跟踪它们。

Verilog HDL任务和函数
--禁止命名的块和任务

disable语句提供了一种能力，用于终止和并行活动过程相关的活动，而保持Verilog HDL过程描述的本质。

- disable语句提供了用于在执行所有任务语句前，终止一个任务的一个机制。比如：

  退出一个循环语句；或者跳出语句，用于继续一个循环语句的其它循环。

在处理异常条件时，是非常有用的。比如：

硬件中断和全局复位。

Verilog HDL任务和函数
--禁止命名的块和任务

disable语句的格式如下：

disable hierarchical_task_identifier（任务标识符）

disable hierarchical_block_identifier（块标识符）

Verilog HDL任务和函数
--禁止命名的块和任务

disable块Veriog HDL描述的例子1

```
begin : block_name
        rega = regb;
        disable block_name;
        regc = rega; //不执行该分配
end
```

Verilog HDL任务和函数
--禁止命名的块和任务

disable块Veriog HDL描述的例子2

```
begin : block_name
...
if (a == 0)
disable block_name;
...
end // 结束命名的块
// 继续下面块的代码
...
```

Verilog HDL任务和函数
--禁止命名的块和任务

disable任务Veriog HDL描述的例子3

```
task proc_a;
begin
        ...
        ...
if (a == 0)
        disable proc_a; // 如果真，则返回
        ...
        ...
end
endtask
```

Verilog HDL任务和函数
--禁止命名的块和任务

disable块Veriog HDL描述的例子4

```
begin : break
   for (i = 0; i < n; i = i+1) begin : continue
      @clk
      if (a == 0) // "continue" loop
         disable continue;
      statements
      statements
```

```
@clk
if (a == b) // "break" from loop
      disable break;
      statements
      statements
   end
end
```

在该例子中，disable功能就相当于c语言中的contine和break语句。

Verilog HDL任务和函数
--禁止命名的块和任务

在fork-join块中disable语句Verilog HDL描述的例子

```
fork
    begin : event_expr
        @ev1;
        repeat (3) @trig;
        #d action (areg, breg);
end
        @reset disable event_expr;
join
```

Verilog HDL任务和函数
--禁止命名的块和任务

在always块中disable语句Verilog HDL描述的例子

```
always begin : monostable
        #250 q = 0;
end
always @retrig begin
        disable monostable;
        q = 1;
end
```

Verilog HDL任务和函数
--函数和函数调用

函数声明

- 函数声明部分可以出现在模块说明中的任何位置，函数的输入参数由输入说明指定，格式如下：

```
function [<lower>:<upper>] <output_name> ;
    input <name>;
    begin
      <statements>
    end
endfunction
```

Verilog HDL任务和函数
--函数和函数调用

其中：

- lower:upper声明了函数输出的数据宽度。
- output_name为函数输出（即返回值）的名字。
- name为输入参数的名字。
- statements为描述语句。

注：

- 如果函数说明部分中没有指定函数取值范围，则其默认为1位二进制数。
- 和任务一样的，函数也可以使用automatic关键字。
- 函数返回值的名字是output_name所定义的标识符。

Verilog HDL任务和函数
--函数和函数调用

函数声明Verilog HDL描述的例子1

```
function [7:0] getbyte;
input [15:0] address;
begin
    . . .
    getbyte = result_expression;
end
endfunction
```

Verilog HDL任务和函数
--函数和函数调用

也可以用下面的函数格式：

```
function [7:0] getbyte (input [15:0] address);
begin
        . . .
        getbyte = result_expression;
end
endfunction
```

Verilog HDL任务和函数
--函数和函数调用

函数调用

- 由函数调用语句调用一个函数，函数调用语句的语法格式如下：

```
signal = function_name(comma_separated_inputs);
```

其中：

- signal为与调用函数返回参数宽度一样的信号名字。
- function_name为函数名字。
- comma_separated_inputs为“,”分割的输入信号的名字。

Verilog HDL任务和函数
--函数和函数调用

函数调用Verilog HDL描述的例子1

```
word = control ? {getbyte(msbyte), getbyte(lsbyte)}:0;
```

Verilog HDL任务和函数
--函数和函数调用

可重入函数调用Verilog HDL描述的例子2

```
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
```

Verilog HDL任务和函数
--函数和函数调用

```
// 测试函数
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      // tryfact结尾
```

Verilog HDL任务和函数
--函数和函数调用

仿真结果是：

0 factorial=1

1 factorial=1

2 factorial=2

3 factorial=6

4 factorial=24

5 factorial=120

6 factorial=720

7 factorial=5040

Verilog HDL任务和函数
--函数和函数调用

常数函数

- 常数函数的调用支持在进行详细的描述时，建立复杂计算的值。
- 一个常数函数调用，模块所调用函数的参数是一个常数表达式。
- 常数函数是Veriog HDL普通函数的子集。

Verilog HDL任务和函数
--函数和函数调用

常数函数应该满足以下的约束：

- 它们没有包含层次引用
- 在一个常数函数内的任意函数调用，应该是当前模块的本地函数。
- 它可以调用任何常数表达式中所允许的系统函数。对其它系统函数的调用是非法的。
- 忽略在一个常数函数内的所有系统任务。
- 在使用一个常数函数调用前，应该定义函数内所有的参数值。

Verilog HDL任务和函数
--函数和函数调用

- 所有不是参数和函数的标识符，应该在当前函数内本地声明。
- 如果使用defparam语句直接或者间接影响的任何参数值，则结果是未定义的。这将导致一个错误，或者函数返回一个未确定的值。
- 它们应该在一个生成模块内声明。
- 在任何要求一个常数表达式的上下文中，它们本身不能使用常数函数。

Verilog HDL任务和函数
--函数和函数调用

下面的例子定义了一个常数函数clogb2，根据ram来确定一个ram地址线的宽度。

常数函数调用的Verilog HDL描述的例子

```
module ram_model (address, write, chip_select, data);
    parameter data_width = 8;
    parameter ram_depth = 256;
    localparam addr_width = clogb2(ram_depth);
    input [addr_width - 1:0] address;
    input write, chip_select;
    inout [data_width - 1:0] data;
```

Verilog HDL任务和函数
--函数和函数调用

```
//定义clogb2函数
        function integer clogb2;
            input [31:0] value;
          begin
            value = value - 1;
            for (clogb2 = 0; value > 0; clogb2 = clogb2 + 1)
                  value = value >> 1;
          end
         endfunction
reg [data_width - 1:0] data_store[0:ram_depth - 1];
//ram model剩余部分
```

Verilog HDL任务和函数
--函数和函数调用

例化这个ram_model，带有参数分配：

```
ram_model #(32,421) ram_a0(a_addr,a_wr,a_cs,a_data);
```