

Verilog 极简语法手册

v1.0

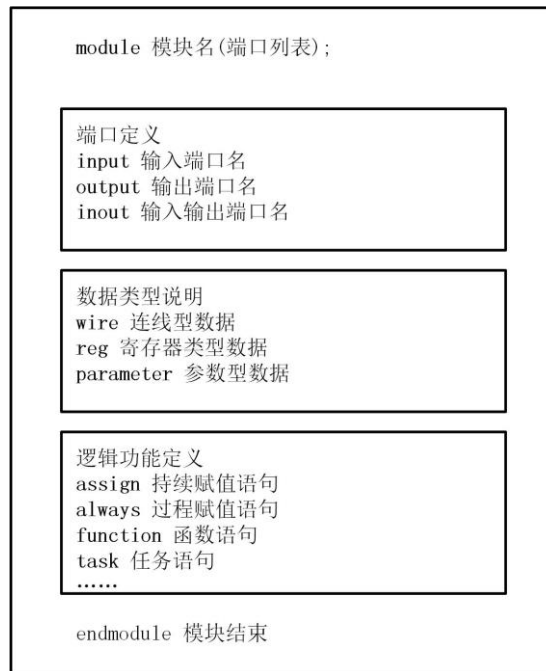
申明：

1. 此文档是作者学习笔记，仅做学习用途，欢迎传阅；
2. 如发现错误和不足之处欢迎联系微信公众号提供意见和建议

原创：微信公众号 **EECScat**

时间：**2020 年 4 月 9 日**

1. Verilog 模块基本结构



2. 词法 (Lexical tokens)

2.1 空白符 (White space)

空白符包括空格、制表位 (tab)、换行、换页。

2.2 注释 (Comments)

两种：单行注释：以“//”开始到本行结束；

块注释：以“/*”开始到“*/”结束。

2.3 数字和字符串 (Numbers & Strings)

2.3.1 逻辑状态

Verilog HDL 有 4 种基本逻辑状态：

- ◆ 0：低电平、逻辑 0 或逻辑非
- ◆ 1：高电平、逻辑 1 或逻辑真
- ◆ x/X：不确定或者未知的逻辑状态
- ◆ z/Z：高阻态

2.3.2 整数常量 (Integer constants)

整数书写方式：+/-<size>'<base><value>

即 +/-<位宽>'<进制><数值>

size 是对应二进制数的宽度；base 为进制；value 是基于进制的数值序列。

其中进制有 4 种形式：

- ◆ 二进制 (b/B)
- ◆ 十进制 (d/D/缺省)
- ◆ 十六进制 (h/H)
- ◆ 八进制 (o/O)

应用时注意：

- ◆ 在十六进制中不区分大小写，例如，4'hF 和 4'hf 表示同样的值
- ◆ 较长的书中间可以用下划线分开以提高可读性，如 12'b1100_0101_1010
- ◆ 当数字不说明位宽时，默认值为 32 位
- ◆ x 或 z 在二进制中表示 1 位 x 或 z，在八进制中表示 3 位 x 或 z，在 16 进制中表示 4 位 x 或 z，其代表的宽度取决于所用的进制，如 8'h9x 等价于 8'b1001xxxx
- ◆ 如果没有定义一个整数的位宽，其宽度为相应值中定义的位数，如'o721 表示 9 位八进制，'hAF 表示 8 位十六进制
- ◆ 如果定义的位宽比实际的位数长，通常在左边填'0'，但如果最左边为 z 或 z 则相应地用 x 或 z 在左边补位
如： 10'b10 等价于 10'b0000000010;
10'bx0x1 等价于 10'bxxxxxx0x1
如果定义的位宽比实际的位数小，那么最左边的位数相应地被截断
如： 3'b1001_0011 等价于 3'b11
5'h0FFF 等价于 5'h1F
- ◆ “?”是高阻态 z 的另一种表示符号，两种数字表示完全等价
- ◆ 整数可以带符号（正、负号）
- ◆ 缺省位宽与缺省进制代表十进制的数，如：28 表示十进制 28
- ◆ 数字中不能有空格

2.3.3 实数 (Real constants)

实数的两种表示法：

- ◆ 十进制表示法
例如 2.0、5.678、0.1 是合法的，但是 2.是非法的，小数点两边都必须有数字
- ◆ 科学计数法
例如：43_5.1e2 值为 43510.0；4E-4 值为 0.0005

Verilog 定义了实数转换为整数的方法，遵循四舍五入的规则，如：
42.446, 42.45 //若转换为整数都是 42

2.3.4 字符串 (String constants)

字符串是双引号内的字符序列。字符串不能分成多行书写。

例如：“this is an example”

2.4 标识符 (Identifiers)

Verilog 中的标识符可以是任意一组字母、数字以及符号“\$”和“_”的组合，但标识符的第一个字符必须是字母或者下划线。另外，标识符是区分大小写的。

例如：cnt_1、_A1 是合法的；而 30_t、cout*是不正确的。

注意，标识符还可以是以符号“\”开头，以空白符结尾的任何字符序列，但反斜线和结束空白符不是标识符的一部分。例如：\OutGate 与 OutGate 等价。

2.5 运算符 (Operators)

(见第 4 章)

2.6 关键字 (Keywords)

Verilog 语言内部已经使用的词称为关键字或保留字，这些关键词用户不能用。

注意，所有关键字都是小写的。例如，ALWAYS (标识符) 不是关键字，它与 **always** (关键字) 是不同的。

附录 A 列出了 Verilog HDL 所有关键字。

3. 数据类型 (Data Type)

在 Verilog 中共有 19 种数据类型

3.1 连线型 (Net Type)

连线型数据相当于硬件电路中的物理连接，其特点是输出的值紧跟输入值的变化。对连线型有两种驱动方式，一种方式是在结构描述中将其连接到一个逻辑门或模块的输出端；另一种方式是用持续赋值语句 **assign** 对其进行赋值

类型	功能说明	可综合性
wire,tri	连线类型	√
wor,rior	具有线或特性的连线	
wand,triand	具有线与特性的连线	
tri1,tri0	分别为上拉电阻和下拉电阻	
supply1,supply0	分别为电源(逻辑 1)和地(逻辑 0)	√

wire 是最常用的连线型数据，定义格式如下：

wire 数据名 1, 数据名 2, ……，数据名 n；

wire a,b;

wire c;

wire [3:0] d; // wire 型向量

3.2 寄存器型 (Resister Type)

寄存器变量对应的是具有状态保持作用的电路元件，如触发器、寄存器等。Register 型变量与 net 型变量的根本区别在于：**register** 型变量需要被明确地赋值，并且 **register** 型变量在被重新赋值前一直保持原值。在设计过程中必须将寄存器型变量放在过程语句（如 **initial**、**always**）中，通过过程赋值语句赋值。另外，在 **always**、**initial** 等过程块中，被赋值的每一个信号都必须定义成寄存器型。

在 Verilog HDL 中有 4 种寄存器型变量：

类型	功能说明	可综合性
reg	常用的寄存器变量	√
integer	32 位带符号整型变量	√
real	64 位带符号实型变量	
time	无符号时间变量	

integer、**real** 和 **time** 三种寄存器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。**Reg** 型变量是最常用的一种寄存器型变量，定义格式如下：

reg 数据名 1, 数据名 2, ……，数据名 n；

例如：

reg a,b;

```
reg [7:0] q; // reg 型向量
```

3.3 参数型 (Parameter)

在 Verilog 中用 **parameter** 来定义符号常量，即用 **parameter** 来定义一个标志符代表一个常量。其定义格式如下：

```
parameter 参数名 1 = 表达式 1, 参数名 2 = 表达式 2, 参数名 3 = 表达式 3, ……;
```

例如：

```
parameter sel = 8, code = 8'ha3;
```

```
parameter datawidth = 8, addrwidth = datawidth * 2;
```

3.4 存储器类型 (Memories)

若干个相同宽度的向量构成数据 (array)，就是存储器。

例如：

```
reg [7:0] mem0 [1023:0];
```

上面定义了一个深度为 1024，宽度为 8bit 的存储器。

4. 运算符 (Operators)

4.1 算术运算符 (Arithmetic Operators)

常用的算术运算符包括：

+ 加

- 减

* 乘

/ 除

% 求模 (求余)

以上算术运算符都是双目运算符

4.2 逻辑运算符 (Logical Operators)

&& 逻辑与

|| 逻辑或

! 逻辑非

4.3 位运算符 (Bitwise Operators)

~ 按位取反

& 按位与

| 按位或

^ 按位异或

^~, ~^ 按位同或

4.4 归约运算符 (Reduction Operators)

缩位运算符是单目运算符，包括以下几种：

& 与

~& 与或

| 或

~| 或非

\wedge 异或

$\wedge\sim, \sim\wedge$ 同或

例如:

```
reg [3:0] a;
```

```
b = & a; // 等效于 b = ( ( ( a[0] & a[1] ) & a[2] ) & a[3] )
```

4.5 关系运算符 (Relational operators)

< 小于

<= 小于或等于

> 大于

>= 大于或等于

4.6 等式运算符 (Equality operators)

== 等于

!= 不等于

=== 全等

!== 不全等

相等运算符 (==): 参与比较的两个操作数必须逐位相等, 其相等比较的结果才为 1, 如果某些位是不定态或高阻态, 其相等比较得到的结果是不定值;

全等运算符 (===): 在对不定态或高阻态的位也进行比较, 两个操作数必须完全一致, 其结果才是 1。

4.7 移位运算符 (Shift Operators)

>> 右移

<< 左移

4.8 条件运算符 (Conditional Operators)

?:

4.9 位拼接运算符 (Concatenations)

{ }

例如: { 3{a,b} } 等同于 { {a,b} , {a,b} , {a,b} }, 也等同于 { a, b, a, b, a, b }.


4.10 Event or

在信号敏感列表中

always @ (clk or rst) 等效于

always @ (clk , rst)

4.11 运算符优先级 (Precedence)

+ - ! ~ (unary)	Highest precedence
**	
* / %	
+ - (binary)	
<< >> <<< >>>	
< <= > >=	
== != === !==	
& ~&	
^ ~^ ~^	
~	
&&	
?: (conditional operator)	Lowest precedence

5. 行为语句

因为本节比较简单，故只列目录。

5.1 过程语句（initial、always）

5.2 块语句（begin...end、fork...join）

5.3 赋值语句（assign、=、<=）

5.4 条件语句（if...else...）

5.5 选择语句（case、casez、casex）

5.6 循环语句（for、repeat、forever、while）

5.7 编译向导（`define、`include、`ifdef、`else、`endif）

6. 进程、任务、函数

6.1 进程（process）

行为模型的本质是进程，一个进程可以被看做是一个独立的运行单元。

一个 Verilog 模块中有如下表示进程的方式：

- ◆ always 过程块
- ◆ initial
- ◆ assign
- ◆ 元件例化

6.2 任务（task）

利用任务可以把一个大的程序模块分解成许多小的任务和函数，以方便调用，并且能使写出的程序结构更清晰。

例程：

```

module traffic_lights;
reg clock, red, amber, green;
parameter on = 1, off = 0, red_tics = 350,
          amber_tics = 30, green_tics = 200;

// initialize colors.
initial red = off;
initial amber = off;
initial green = off;

always begin                                // sequence to control the lights.
    red = on;                               // turn red light on
    light(red, red_tics);                   // and wait.
    green = on;                             // turn green light on
    light(green, green_tics);              // and wait.
    amber = on;                             // turn amber light on
    light(amber, amber_tics);              // and wait.
end

// task to wait for 'tics' positive edge clocks
// before turning 'color' light off.
task light;
output color;
input [31:0] tics;
begin
    repeat (tics) @ (posedge clock);
    color = off;                            // turn light off.
end
endtask

always begin                                // waveform for the clock.
    #100 clock = 0;
    #100 clock = 1;
end
endmodule // traffic_lights.

```

6.3 函数（function）

函数的目的是返回一个值，以用于表达式的计算。

例程：


```

module tryfact;

// define the function
function automatic integer factorial;
input [31:0] operand;
integer i;
if (operand >= 2)
    factorial = factorial (operand - 1) * operand;
else
    factorial = 1;
endfunction

// test the function
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

```

The simulation results are as follows:

```

0 factorial=1
1 factorial=1
2 factorial=2
3 factorial=6
4 factorial=24
5 factorial=120
6 factorial=720
7 factorial=5040

```

7. 测试平台 Testbench

7.1 系统任务与系统函数

系统任务和系统函数有以下一些特点：

- ◆ 系统任务和系统函数一般以符号“\$”开头，如\$monitor、\$readmemh等；
- ◆ 使用系统任务和系统函数，可以显示模拟结果，对文件进行操作，以及控制模拟的执行过程等；
- ◆ 使用不同的 Verilog 仿真工具（如 VCS、Verilog-XL、Modelsim 等）进行仿真时，这些系统任务和系统函数在使用方法上可能存在差异，应根据使用手册来使用；
- ◆ 一般在 initial 或 always 过程块中调用系统任务和系统函数；
- ◆ 用户可以通过编程语言接口（PLI）将自己定义的系统任务和系统函数加到语言中，以便仿真和调试。

常用的系统任务和系统函数

- ◆ \$display 和 \$write

显示模拟结果，区别是前者在输出结束后能自动换行，而后者不能。

使用格式：

\$display(“格式控制符”，输出变量名列表)

\$write(“格式控制符”，输出变量名列表)

格式控制符

Argument	Description
%h or %H	Display in hexadecimal format
%d or %D	Display in decimal format
%o or %O	Display in octal format
%b or %B	Display in binary format
%c or %C	Display in ASCII character format
%l or %L	Display library binding information
%v or %V	Display net signal strength

%m or %M	Display hierarchical name
%s or %S	Display as a string
%t or %T	Display in current time format
%u or %U	Unformatted 2 value data
%z or %Z	Unformatted 4 value data

转义字符

Argument	Description
\n	The newline character
\t	The tab character
\\	The \ character
\"	The " character
\ddd	A character specified by 1 to 3 octal digits
%%	The % character

◆ \$monitor 和 strobe

使用格式:

\$monitor (“格式控制符”，输出变量名列表)

\$strobe (“格式控制符”，输出变量名列表)

例如:

\$monitor(\$time,,,"A=%d B=%d C=%d", A, B, C);

// 只要 A、B、C 三个变量中任何一个的值发生变化都会将 A、B、C 的值输出

◆ \$time 与 realtime

当这两个函数被调用时，都返回当前时刻距离仿真开始之后的时间量值，所不同的是，\$time 函数以 64 位整数值形式返回模拟时间，\$realtime 函数则以实数型数据返回模拟时间。

◆ **\$finish 与\$stop**

\$finish 表示结束仿真，**\$stop** 表示中断仿真

使用格式：

```
$stop;  
$stop(n);  
$finish;  
$finish(n);
```

n 可以使 0、1、2 等值，

0：不输出任何信息；

1：给出仿真时间和位置；

2：给出仿真时间和位置，还有其他一些运行统计数据。

当仿真程序执行到**\$stop** 语句时将暂停仿真，此时设计者可以输入命令对仿真器进行交互控制。而仿真程序执行到**\$finish** 语句时，则终止仿真，结束整个的仿真过程，返回主操作系统。

◆ **\$readmemh 和 readmemb**

从外部文件读取数据放入存储器中，**\$readmemh** 读取十六进制数据，**readmemb** 读取二进制数据。

使用格式：

```
$readmemh("数据文件名", 存储器名, 起始地址, 结束地址);  
$readmemb("数据文件名", 存储器名, 起始地址, 结束地址);
```

其中，起始地址和结束地址可以省略，如果省略起始地址，表示从存储器的首地址开始存储；如果缺省结束地址，表示一直存储到存储器的结束地址。

例如：

```
reg [7:0] my_mem [0:255];  
initial  
begin  
    $readmemh("mem.hex",mymem);  
end
```

◆ **\$random**

\$random 是一个产生随机数的系统函数，每次调用该函数，将返回一个 32 位的随机数，该随机数是一个带符号的整数。

例如：

```
Data = $random; //产生一个随机数然后赋值给 Data
```

◆ **文件输出**

例如：

```
integer write_out_file; //定义一个文件指针  
integer write_out_file=$fopen("write_out_file.txt");  
$fdisplay(write_out_file,"@%h\n%h", addr, data);  
$fclose("write_out_file");
```

7.2 用户自定义元件（UDP）

User Defined Primitives，用户可以自己定义基本逻辑单元的功能，可以向调用基本门元件一样调用这些自己定义的元件。UDP 不能用于可综合的设计描述中，而只能用于仿真程序中。

例如：

The following example defines a multiplexer with two data inputs and a control input.

```
primitive multiplexer (mux, control, dataA, dataB);
output mux;
input control, dataA, dataB;
table
// control dataA dataB mux
    0      1      0 : 1 ;
    0      1      1 : 1 ;
    0      1      x : 1 ;
    0      0      0 : 0 ;
    0      0      1 : 0 ;
    0      0      x : 0 ;
    1      0      1 : 1 ;
    1      1      1 : 1 ;
    1      x      1 : 1 ;
    1      0      0 : 0 ;
    1      1      0 : 0 ;
    1      x      0 : 0 ;
    x      0      0 : 0 ;
    x      1      1 : 1 ;
endtable
endprimitive
```

7.3 延时模型

在仿真中还涉及延时表示的问题。延时包括门延时、assign 赋值延时和连线延时等。们演示为从门的输入端发生变化到输出发生变化的延迟时间；assign 赋值延时指等号右端某个值发生变化到等号左端发生相应变化的延迟时间；连线延时则体现了信号在连线上的传输延时。如果没有定义时延值，缺省时延为 0。

7.3.1 时间标尺定义 timescale

`timescale 语句用于定义模块的时间单位和时间精度，其使用格式如下：

`timescale <时间单位> / <时间精度>

表示时间度量的符号：s, ms, us, ns, ps, fs

例如：`timescale 1ns/100ps

7.3.2 延时的表示方法

delaytime; // 表示延迟时间为 delaytime

(d1, d2); // d1 表示上升延迟，d2 表示下降延迟

(d1, d2, d3); // d3 则表示转换到高阻态 z 的延迟

例如：

not #4 gate1(out, in); // 延迟时间为 4 的非门

and #(5, 7) gate2(out, a, b); // 与门的上升延迟为 5，下降延迟为 7

or #5 gate3(out, a, b); // 或门的上升延迟和下降延迟都为 5

bufif0 #(3, 4, 6) gate4(cout, in, enable) // bufif0 门的上升延迟为 3，下降延迟为 4，高阻延迟为 6

7.3.3 延时说明块（specify 块）

Verilog 可对模块中某一指定的路径进行延迟定义，这一路径连接模块的输入端口（或 inout 端口）与输出端口（或 inout 端口），利用延迟定义块在一个独立的块结构中定义模块的延迟。在延迟定义块中，要描述模块中的不同路径并给这些路径赋值。

延迟定义块的内容应放在关键字 **specify** 与 **endspecify** 之间，并且必须放在一个模块中，还可以使用 **specparam** 关键字定义参数。

例如：

```
module delay(out,a, b, c);
  output out;
  input a, b, c;
  and a1(n1, a, b);
  or o1(out, c, n1);
  specify
    (a=>out) =2;
    (b=>out) =3;
    (c=>out) =1;
  endspecify
endmodule
```

附录 A Verilog 关键字

always	ifnone	rnmos
and	incdir	rpmos
assign	include	rtran
automatic	initial	rtranif0
begin	inout	rtranif1
buf	input	scalared
bufif0	instance	showcancelled
bufif1	integer	signed
case	join	small
casex	large	specify
casez	liblist	specparam
cell	library	strong0
cmos	localparam	strong1
config	macromodule	supply0
deassign	medium	supply1
default	module	table
defparam	nand	task
design	negedge	time
disable	nmos	tran
edge	nor	tranif0
else	noshowcancelled	tranif1
end	not	tri
endcase	notif0	tri0
endconfig	notif1	tri1
endfunction	or	triand
endgenerate	output	trior
endmodule	parameter	trireg
endprimitive	pmos	unsigned
endspecify	posedge	use
endtable	primitive	vectored
endtask	pull0	wait
event	pull1	wand
for	pulldown	weak0
force	pullup	weak1
forever	pulsetyle_oneevent	while
fork	pulsetyle_ondetect	wire
function	rcmos	wor
generate	real	xnor
genvar	realtime	xor
highz0	reg	
highz1	release	
if	repeat	

参考资料:

王金明《Verilog HDL 程序设计教程》

IEEE standard 1364 《IEEE Standard Verilog Hardware Description Language》