Verilog 语法说明

（版本号：V1.0）

上海复旦微电子集团股份有限公司

(86-21)6565 5050

(86-21)6565 9115

200433

上海市国泰路 **127** 号复旦国家大学科技园 **4** 号楼

修订记录

日期	修订版本	描述	作者
2021-01-01	1.0	初稿	于晓

目录

1 关于手册

本手册旨在向 Procise 用户说明该软件目前支持、不支持的 verilog 语法。目前 procise 支持的 verilog 标准为 verilog 2K。

2 支持的 verilog 语法

2.1 Verilog 数据类型

2.1.1 wire

wire 型数据常用来表示用于以 assign 关键字指定的组合逻辑信号。Verilog 程序模块中输入输出信号类型缺省时自动定义为 wire 型。wire 型信号可以用作任何方程式的输入，也可以用作“assign”语句或实例元件的输出。

示例：

```
wire a;
wire [7:0] b;
wire [4:1] c, d;
```

2.1.2 reg

reg 型数据常用来表示用于 “always”模块内的指定信号,常代表触发器。通常,在设计中要由“always”块通过使用行为描述语句来表达逻辑关系,在“always”块内被赋值的每一个信号都必须定义成 reg 型。

示例：

```
reg rega;
reg [3:0] regb;
reg [4:1] regc, regd
```

通过 reg 类型可以定义数组，描述 RAM 型存储器；

格式：reg [n-1:0] 存储器名[m-1:0]；

说明：数组类型不能整体进行赋值，对数组进行地址索引的表达式必须是常数表达式；声明 reg 类型时可以进行初始化。

2.1.3 module parameters

parameter 用来定义一个标识符代表一个常量。parameter 是参数型数据的确认符，确认符后跟着一用逗号分隔开的赋值语句表。在每一个赋值语句的右边必须是一个常数表达式。也就是说，该表达式只能包含数字或先前已定义过的参数。

示例：

```
parameter msb=7; //定义参数 msb 为常量 7
parameter e=25, f=29; //定义二个常数参数
parameter average_delay = (r+f)/2;
```

2.1.4 local parameters

local parameters 定义在 module 内部，不能够通过实例化或 defparam 语句进行修改

示例：

```
local parameter msb = 10;
```

2.2 Verilog 表达式

2.2.1 单个操作数

单个操作数即单独的一个变量，可以是 wire、reg、事先定义的常数等，可以用于赋值语句、条件判断等

示例：

赋值语句中 a<=b;

条件中：if(a) 、case a

2.2.2 常数

常数（constval）表示设计中的常数，可以是不同进制的常数：二进制、十进制、十六进制等或 string 类型。

示例：

a<=4'b1010; b<=2;

2.2.3 vector bit-select and part-select

bit-select、part-select 表示取变量或数组某个元素中的某一位或某几位

示例：

reg [9:0] re;

a<=re[5:1]; b<=re[6]; c<=re[5-:3]

reg [31:0] mem [0:1023];

a<=mem[1][10:1]; b<= mem[5][20];

说明：bit-select、part-select 的索引可以为简单的常数，也可以为表达式

2.2.4 数组元素

数组元素 arrayElement 表示获取定义的 memory 其中的某个元素

示例：

reg[31:0] mem[0:1024];

a<=mem[5];

说明：在 verilog 中对 memory 类型只能单个元素进行操作（赋值、读取）

2.2.5 函数调用

functionCall 表示函数调用，可以用于赋值语句、条件判断等

示例：

a<= fun(x,y,z)

if-else 语句的 condtion 表达式：if(function(x,y,z))

2.2.6 二元表达式

二元操作表达式由左右两个操作数构成

算数操作：加法+、减法-、乘法 x、除法/、取模%、乘方**

关系操作：小于< 、大于>、小于等于<=、大于等于>=、等于==、不等于!=

逻辑操作：逻辑与&&、逻辑或||

按位与&、按位或|、按位异或^、按位同或^~、左移、右移等操作。

示例： a+b、a*b、-a+x/y-c ....

2.2.7 一元表达式

一元操作符有：取反~、逻辑非！、缩减运算符&

示例：~a、!b、&B

2.2.8 拼接操作

拼接操作就是将几个表达式连接在一起，组合成一个整体的表达式

示例：

{a,b[3:0],w,3'b101}

{a,b[3],b[2],b[1],b[0],w,1'b1,1'b0,1'b1}

位拼接还可以用重复法来简化操作：

{4{w},b,a[3:0]}

2.2.9 条件表达式

格式：condition ? result1 : result0

实例：assign out = a==b? in1: in2;

2.3 Verilog 语句

2.3.1 阻塞赋值语句

示例： a = b;

这条语句执行完后才会执行下一条语句

2.3.2 非阻塞赋值语句

示例： a<=b

这条语句的执行不会阻碍下面的语句

阻塞、非阻塞赋值语句一般是在 always 块内部进行赋值

2.3.3 条件赋值语句

条件语句主要分为两种 if_else 语句、case 语句，根据条件不同执行不同的语句

1）if_else 语句

if 语句是用来判定所给定的条件是否满足，根据判定的结果（真或假)决定执行给出的两种操作之一。

Verilog 中有三种形式的 if 语句：

```
a. if(condition)
       stmt;
 eg.  if(a>b)
        out<=in;
b. if(condition)
        stmt1;
     else
        stmt2;
eg. if(a>b)
```

```
      out<=in1;
    else
      out<=in2;
```

c. if(condition1)

```
       stmt1;
     else if (condition2)
       stmt2;
     .......
     else
       stmtn;
```

eg. if(a>b)

```
       out<=in1;
     else if (a==b)
       out<=in2;
     else
       out<=in3;
```

stmt 部分可能包含多条语句，多条语句需要在 begin-end 块中

2）case 语句

case 语句是一种多分支选择语句，if 语句只有两个分支可供选择，而实际问题中常常需要用到多分支选择，verilog 语言提供的 case 语句直接处理多分支选择。case 语句的一般形式如下:

1) case(表达式) <case 分支项> endcase

2) casez(表达式)<case 分支项> endcase

3) casex(表达式) <case 分支项> endcase

case 分支项的一般格式：

分支项表达式： 语句

default 项：　　语句

示例：

```
case (a+b)
    n1 : out = in1;
    n2 : out = in2;
    n3 : out = in3;
    default: out = in4;
endcase
casex ({t1[7], t1[0]})
        2'b0x:
            begin
                t2 = { in1[7], t1[1:6], in2[0] };
            end
        2'b1x:
            begin
                t2 = ~in1-in2^t1;
            end
        default:
            begin
                t2 = -in2-+in1++t1;
            end
endcase
casez (i5)
   3'b00z  : o1 = 2 && i1;
```

```
   3'b1z?  : o1 = i1 + i2;
default : o1 = 2 && i2;
```

2.3.4 while 循环语句

格式：

```
while(表达式)
    语句
```

或：

```
while(表达式)
 begin
    多条语句
 end
```

示例：

```
while(reg)
  begin
    if(reg[0])
        count<=conut+1;
     else
        count=count>>1;
  end
```

2.3.5 for 语句

格式：

```
for（表达式 1；表达式 2；表达式 3）
 begin
```

语句

end

示例：

```
for( bindex=1; bindex<=size; bindex=bindex+1 )
    if(opb[bindex])
        result = result + (opa<<(bindex-1));
```

示例：

```
for(i = step(j);i>= temp;i=i-1)
for(i=0;i<func(width);i=i+1)
```

2.3.6 repeat 语句

格式:

repeat(表达式) 语句； 或

repeat(表达式)

begin

多条语句

end

```
repeat(5)
    begin
        if(shift_opb[1])
            result = result + shift_opa;
        shift_opa = shift_opa <<1;
        shift_opb = shift_opb >>1;
    end
```

在 repeat 语句中，其表达式通常为常量表达式

2.3.7 always 块语句

always 语句定义在一个 always 块中，在 always 语句块内部语句是顺序执行的，不同的 always 块之间是并行执行的。

格式：

```
always@（敏感信号列表）
begin
  stmts;
end
```

示例：

```
always @(posedge clock or posedge reset) //由两个沿触发的 always 块
    begin
    ……
    end
always @( a or b or c ) //由多个电平触发的 always 块
    begin
    ……
    end
```

说明：always 关键字后面的敏感列表中可以有多个敏感信号，敏感信号并非如 a\b\c 这样简单，敏感信号还可以是一个表达式。块中的语句可为各种顺序语句：if-else 、case 、blocking、nonblocking 等

2.3.8 Initial 语句

设计中可以有多个 initial 语句，各个 initial 语句之间并行执行，initial 内部顺序执行，一般情况下在 initial 语句中对变量进行初始化。

格式：

```
initial
  begin
    stmts
```

end

2.3.9 Generate 语句

generate 语句的最主要功能就是语句或者模块进行复制。generate 语句有 generate_for、generate_if、generate_case 三种语句。

1）generate_for 语句

a.必须使用 genvar 申明一个正整数变量，用作 for 循环的判断

b.需要复制的语句必须写到 begin_end 语句里面。就算只有一句

c.begin_end 需要有一个类似于模块名的名字

示例：

```
genvar  i;
generate for(i=0;i<4;i=i+1)
    begin : gfor
        assign temp[i] = data_in[2*i+1:2*i];
    end
endgenerate
```

2） generate_if

generate_for 用于复制模块，而 generate_if 则是根据模块的参数（必须是常量）作为条件判断，来产生满足条件的电路。相当于判断语句。

示例：

```
generate
    if(S < 7)
        assign d = t0 | t1 | t2;
    else
        assign d = t0 & t1 & t2;
endgenerate
```

3） generate_case

generate_case 其实跟 generate_if 一样，都是根据参数（都必须为常量）作为判断条件，来产生满足条件的电路，不同于使用了 case 语法而已。

示例：

```
generate
    case(S)
    0:
        assign d = t0 | t1 | t2;
    1:
        assign d = t0 & t1 & t2;
    default:
        assign d = t0 & t1 | t2;
    endcase
endgenerate
```

2.3.10 实例化语句

用户实例化语句包含两种情况：实例化用户自己设计的 module，实例化库中的宏单元。

2.3.11 assign 赋值语句

格式：assign var=表达式

其中，var 是被赋值的对象，var 可以是一个完整的变量，也可以是变量中的位选择或者部分位选择，也可以是几个变量的组合，但是必须是线网型变量。

2.4 系统函数

目前支持的系统函数有 signed、unsigned

3 不支持的 Verilog 语法

3.1 system verilog

3.2 real 类型

3.3 verilog 属性

例如：full case、parallel case、keep 属性、ram_style、rom_style、fsm_encoding

clock_buffer_type、async_reg

3.4 两个不同的函数进行递归调用

举例：

```
out1 = fn1(r1 - r2) ;
function automatic integer fn1(input signed [31:0] in1) ;
    begin
        if (in1 | 0)
            fn1 = 0 ;
        else
        begin
            if (in1 == 1)
                fn1 = fn2(in1 + 0) ;
            else
                fn1 = fn2(in1 - 0) ;
        end
    end
    endfunction
```

```
function automatic integer fn2(input    [31:0] in1) ;
begin
    if (in1 != 1)
        fn2 = 0 ;
    else
    begin
        if (in1 || 1)
            fn2 =    fn1(in1 ^ 00) ;
        else
            fn2 = fn1(in1 - 1) ;
    end
end
endfunction
```

3.5 敏感列表中不建议写 bus 类型的信号

举例：

```
wire [3:0] clk;
always@(posedge clk)
    q<=d;
```

3.6 同一个 always 块的敏感列表中有两个以上的 clk 信号

举例：

```
wire clk1，  clk2;
```

always@(posedge clk1，posedge clk2)

q<=d;

3.7 定义的 bus，然后一位一位使用。这种 bus 位宽不建议定义的太大（比如位宽上万），综合会比较慢。

举例：

wire [32767:0] a;

a[0] = i0;

a[1] = i1;

a[2] = i2;

…..

a[32767] = i5;

3.8 specify parameters

3.9 生成的相应的 DFF 需要保留，不能被优化掉

3.10 ForLoop 不支持 index 的初始值或结束值为变量

eg. for (i = size; i < C_WIDTH; i = i+1) begin ...

// input [31:0] size; parameter C_WIDTH = 8;

3.11 verilog 语句后面或者下面可以写 attribute，告诉综合器怎么去综合，告诉综合器一些约束之类的信息。.

3.12 rangeSelect 都为初始和结束位置都为变量

3.13 对 memory 赋值情况

1、memory 的初始值定义在单独的文件中

2、initial 语句中使用 for 语句对 memory 赋值

举例：

```
reg [31:0] mem [1024:0];
initial
begin
for (int i=0;i<1024;i++)
   mem[i] = 32'b0;
end
```

3、在不同的并行块（always、generate）中对同一个 memory 的相同地址赋值

举例：

```
always @(posedge clk)
        begin
            if (wen[0])
                begin
```

```
                mem_r[addr][7:0] <=    din[7:0] ;
            end
        end
always@(posedge clk)
        begin
            if (wen[1])
                begin
                    mem_r[addr][15:8] <=    din[15:8] ;
                end
        end
always@(posedge clk)
        begin
            if (wen[2])
                begin
                    mem_r[addr][23:16] <=    din[23:16] ;
                end
        end
```

4 不支持的 VHDL 语法

4.1 不建议写 offset 为变量或者 offset 无法计算的 range_select 的写法

举例：s(a downto b)，xst 和 synplify 也不允许这么写，vivado 可以

v(a to a+2)，目前我们识别不出来此 range select 的 offset 是常数

4.2 subprogram

1. 不支持 subprogram 的 port 或 return 类型为 array of record， record include array 等的各种复杂嵌套

举例：

```
TYPE arr IS ARRAY(0 to 3) OF STD_LOGIC_VECTOR(1 downto 0);
TYPE rec IS RECORD
 a : BIT;
 b : arr;
END RECORD;
TYPE arr1 IS ARRAY(0 to 1) OF rec;
FUNCTION func1 (arg : arr1) RETURN arr1 IS ...
PROCEDURE proc1 (arg1 : IN BIT ; arg2 OUT arr1) IS .....
```

2. 不支持 subprogram 的递归调用

举例：

```
FUNCTION func1 (arg : BOOLEAN) RETURN STD_LOGIC_VECTOR IS
   VARIABLE v1 : BOOLEAN := false;
   VARIABLE result : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
  IF (v1 = true) THEN
     result := func1(arg);
   END IF;
    RETURN result;
END func1;
```

3. 尽量避免 function 的多个 return 语句

举例：

```
FUNCTION func1 (arg : STD_LOGIC_VECTOR(1 DOWNTO 0)) RETURN STD_LOGIC
_VECTOR IS
BEGIN
   IF (arg = "00" ) THEN
      RETURN "11";          -- return_stmt1
   ELSIF    (arg = "10" ) THEN
     RETURN "01";      -- return_stmt2
   ELSIF    (arg = "01" ) THEN
     RETURN "10";      -- return_stmt3
   ELSIF    (arg = "11" ) THEN
     RETURN "00";      -- return_stmt4
   END IF;
   RETURN "00";              -- return_stmt5
END func1;
```

4.3 不支持不定位宽的 array，record 互相嵌套的使用

举例：

```
TYPE rec IS RECORD
   a : BIT;
   b : arr;
 END RECORD;
 TYPE arr1 IS ARRAY natural range<> OF rec;
 SIGNAL s : arr1(3 downto 0）；
```

4.4 目前还不支持的敏感列表书写格式

举例 1. process(clk, rst)　　--此种情况下提取不出来 rst 这个异步清零（或异步置一）控制端

```
begin
  if rst = '0' then
    if rising_edge(clk) then
      o <= i1;
    end if;
  else
    o <= "00";
  end if;
end process;
```

举例 2. process(clk, rst)　　-- process 块中同时含有 clk 的上升沿和下降沿

```
begin
  if (clk'event and clk = '1') then
    o <= i;
  end if;
  if (rst = '1') then
    o <= "00";
  elsif (clk'event and clk = '0') then
    o <= "11";
  end if;
end process;
```

4.5 不支持 vhdl 语句中 xilinx 的 Attribute

vhdl 语句后面或者下面可以写 attribute，告诉综合器怎么去综合，告诉综合器一些约束之类的，或者布局布线的约束。我们现在不处理 attribute。

举例一个 attribute：

```
attribute syn_preserve: boolean;
attribute syn_preserve of rtl: architecture is true;
```

syn_preserve 的作用是告诉综合器某个 module 内的等价的 FF 不要被优化掉。

支持这些 attibute 需要 RTL 组和逻辑综合组和 DB 组共同完成。

4.6 VHDL 不建议 loop 循环次数过大

举例：

```
function func1 (arg : integer) return integer is
    variable result : integer;
begin
    for i in 0 to 2147483647 loop
        if (i = arg) then
            result := i;
            break;
        end if;
    end loop;
    retult := -1;
    return result;
end function;
```