第3章 Verilog语言要素

——标识符、注释、编译程序指令、系统任务、系统函数、数据类型

西安交大电信学院微电子学系
程　军
jcheng@mail.xjtu.edu.cn

标识符

- 标识符由字母、数字、$符号和_符号组成。
- 标识符第一个字符必须是字母或者_。
- 标识符区分大小写。
- 关键字都是小写标识符。
- 转义标识符以“\”开头，以空格结尾，包含任何可打印字符。转义标识符的主要用途在于不同HDL语言的转换，或者不同CAE系统的转换。不要在正常情况下使用转义标识符。关键字不能是转义标识符。

好的命名原则

- 使用有意义的名字，如：clock, address, master, slave, decode, slave_clk等
- 不要使用$字符，结尾不要用下划线“_”
- 使用大小写混合的名字(如：GoodName和good_name)
- 特殊电路结构采用传统命名习惯，如：clock、muxes、synchronizer等
-

注释

- /*

  这里面都是被注释的内容

  可以注释多行

  */
- // 在这之后到本行结束是注释的内容

格式

- 程序书写是自由格式的，以分号作为语句分隔符。

```
initial begin gre_sel=3'b001; #2 gre_sel=3'b011; end
```

上面的语句与下面的完全等价

```
initial
  begin
    gre_sel=3'b001;
    #2 gre_sel=3'b011;
  end
```

建议：一行的长度必须小于132个字符

系统任务和系统函数

- 以$符号开始的标识符表示系统任务或者系统函数。由Verilog系统(IEEE)标准提供，也可以由用户通过PLI编写。
- 任务通过参数传递返回0个或者多个值，允许有延时。
- 函数只返回一个值，不允许延时。函数的执行不需要时间。
- 详细描述在第10章。

编译器指令

- 以`(反引号)开始的一些标识符是编译器指令，类似C语言中的宏指令。
- `define, `undef
- `ifdef, `ifndef, `else, `elseif, `endif
- `default_nettype
- `include
- `resetall
- `timescale
- `unconnected_drive, `nounconnected_drive
- `celldefine, `endcelldefine
- `line

编译器指令(续)

- **`define**和**`undef**——宏定义和宏定义取消
- **`define**用于文本替换，类似C语言中的#define
- 例1，建议宏定义全部用大写字母

```
`define MAX_BUS_SIZE 32     //注意，结尾没有";"
`define RESULT_SIZE (AWIDTH * BWIDTH)
    ...
  reg [`MAX_BUS_SIZE-1:0] addreg;
```

- 例2, **`undef**指令取消前面定义的宏

```
`define WORD 16     //建立一个文本宏替换
    ...
wire [`WORD : 1] sio_rdy;      //使用宏替换
    ...
`undef WORD          // 这个指令后，WORD宏定义不再有效
```

编译器指令(续)

- **`ifdef、`ifndef、`else、`elseif和`endif——条件编译**
  - 例1

```
`ifdef WINDOWS
    parameter WORD_SIZE = 16
`else                              // `else是可选的
    parameter WORD_SIZE = 32
`endif
```

  - 例2，**`ifndef**与**`ifdef**指令相反，“若没有定义该宏定义，则…”

```
`ifndef RTL_SYNTHESIS
   assign #(PERIOD/2) core_clock = ~core_clock;
`endif
`ifdef ALWAYS_FORM always@(a, b) y<= a | b;
`elseif ASSIGN_FORM assign y = a | b;
`else or u1or (y, a, b);
`endif
```

编译器指令(续)

- `default_nettype——指定没有被说明的连线的类型，默认是wire类型，可以由用户改为其他

`default_nettype wand

则指定的默认线网类型改成了线与类型

- none这个值可以用来指定默认的线网类型，表示不允许使用隐含的线网联系。

编译器指令(续)

- `include——包含文件

`include “../../primitives.v”

则在编译时，该行就会被primitives.v文件中的内容取代。可以使用相对路径，又可以使用绝对路径。

- `resetall——将所有的编译指令重置为默认的值，例如`default_nettype会被重置为wire。

编译器指令(续)

- `timescale——指定时间单位和时间精度
- 格式：**`timescale** *time_unit/time_precision*
- time_unit和time_precision由1、10、100的值和s、ms、us、ns、ps和fs单位构成。
- `timescale指令放在模块声明的外部，影响其后所有的延时值。直到遇到另一个`timescale指令或者`resetall指令。
- 多个模块都带有`timescale指令会怎样？
  - 仿真器定位到最小延时精度，将所有的延时换算成最小延时精度。

编译器指令(续)

```
`timescale 1ns/100ps
module and_function(y, a, b);
    output y; input a, b;
    and #(5.22, 6.17) u1and(y, a, b);
endmodule
```

#5.22 --> 5.2ns
#6.17 --> 6.2ns

```
`timescale 10ns/1ns
```

#5.22 --> 52ns
#6.17 --> 62ns

仿真器在仿真tb_and模块时，and_function为其子模块，将所有模块的最小时间精度设为100ps。则tb_and中的延时换算为：52ns->520*100ps, 104ns -> 1040*100ps, 150ns -> 1500*100ps，仿真器使用100ps作为时间精度。

```
`timescale 10ns/1ns
module tb_and;
  reg put_a, put_b;
  wire get_y;
  initial begin
      put_a = 0;
      put_b = 0;
      #5.21 put_b = 1;
      #10.4 put_a = 1;
      #15  put_b = 0;
   end
   and_function u_and_function
            (get_y, put_a, put_b);
endmodule
```

#5.21 --> 52ns
#10.4 --> 104ns
#15 --> 150ns

编译器指令(续)

- `unconnected_drive和`nounconnected_drive

指定未连接的输入端口接高电平或者低电平

例：

```
`unconnected_drive pull1
  ..... //在这两个指令之间的未接输入端口接高电平
`nounconnected_drive

`unconnected_drive pull0
  ..... //在这两个指令之间的未接输入端口接低电平
`nounconnected_drive
```

编译器指令(续)

- `celldefine和`endcelldefine

用于将module标记为单元模块，一些PLI要使用cell，例如计算延时。`celldefine是单元定义开始，`endcelldefine是单元定义结束。

- 单元模块名和端口名一般用大写字母。

```
`timescale 1ns/10ps
`celldefine
module NAND2X1 (Y, A, B);
  input A, B;
  output Y;
  nand (Y, A, B);
  …
endmodule // NAND2X1
`endcelldefine
```

编译器指令(续)

- `line——行号(line number)编译器指令将说明verilog源代码的原始位置。
  - Verilog工具经常在输出的错误和警告信息中包含Verilog文件名和行号的信息；
  - 但是如果Verilog预处理程序修改了文件 (行增加或者减少)，原来的文件名和行号信息就会丢失，出现的错误和警告信息中的文件名或者行号就不对了。
  - 预处理程序可以在被修改的代码中加入`line指令，保留原来的位置信息。
- `line number “filename” level

number—新的行号

filename—新的文件名

level—1：在include文件进入后下一行是第一行；

2：在include文件退出后下一行是第一行；

0：两者都没有

编译器指令(续)

- 可以用在Verilog代码的任何位置；
- 一般用户不会直接使用该指令，Verilog预处理程序使用。
- line编译器指令示例：

```
`line 3 "orig.v" 2
// This line is line 3 of orig.v after exiting include file
```

值集合

- Verilog有4种基本值：
  - 0：逻辑0、或者“假”
  - 1：逻辑1、或者“真”
  - x：未知
  - z：高阻、无驱动
- x、z不区分大小写，门的输入或者表达式中的“z”会被解释成“x”。
- Verilog的常量由这4个基本值构成。包括：整型、实数型、字符串。常量间可以用“_”分隔提高易读性，但“_”不能是常量的首字符。

值集合(续)

- 整型数——简单十进制数格式或者基数格式
  - 十进制数格式：可以有+、-号，代表符号数
    - 32　6位二进制数为100000，5位无法表示
      7位二进制数为0100000
    - -15　5位二进制数10001(补码)、6位110001
  - 基数格式：更常用
    - [size]'[signed]base Value
    - size——指定该数值的二进制表示的位数(位宽)
    - signed——小写s或者大写S，表示符号
    - base——定义基数，可以是o(O)，b(B)，d(D)或者h(H)
    - Value——基于base的数字序列，不能是负数
    - x、z以及十六进制数中的a～f不区分大小写。

值集合(续)

- 例：

5'O37	5位8进制数
4'D2	4位十进制数
4'B1x_01	4位二进制数
7'Hx	7位十六进制数，扩展到xxxxxxx
4'hz	4位z，高阻
8 'h 2A	在位长和字符之间、基数和数字间允许空格
8'sh51	8位有符号数01010001
6'so72	6位有符号数111010，代表十进制-6
4'h-4	非法，数字不能为负
3' b001	非法，'和b之间不能有空格
(2+3)'b10	非法，位长不能是表达式

注意：x(或z)在16进制中代表4位，8进制中代表3位，二进制中代表1位。

值集合(续)

- 基数格式通常是无符号数。
- 位宽定义是可选择的，如果没有定义则至少是32位宽。
- 例：

`o721　　32位8进制无符号数

`hAF　　32位16进制无符号数

`sb1011　　32位二进制数，有符号数-5

值集合(续)

- 基数表示法中，定义位宽比常量位长长时，无符号数左边补0，有符号数左边填符号位补齐。如果最左边为x或者z，则相应地补x或者z。

例：10'b10 ---> 0000000010

10'bx0x1 ---> xxxxxxx0x1

8'sb101101--->11101101 左边补符号位

- 如果定义长度比常量位长短，要截断最左边的位，截断不保留符号位。

例：3'b1001_0011 ---> 3'b011

5'H0FFF ---> 5'H1F

3'sb10100 ---> 3'sb100

- 可以用？代替z值，提高可读性(8.6节无关位)。

值集合(续)

- 实数——十进制计数法和科学计数法
  - 十进制：2.0, 5.678, 11572.12, 0.1等

    2.　　//非法，小数点两边必须有数字
  - 科学计数法：

    23_5.1e2 ---> 235.1x 10^2 ---> 23510.0

    3.6E2 ---> 3.6x10^2 ---> 360.0

    5E-4 ---> 5x10^{-4} ---> 0.0005
- 实数与整数的隐式转换
  - 整数 ---> 实数
  - 实数 ---> 整数：四舍五入到最近的整数
    - 例：42.4 --> 42，92.5 --> 93，92.69 --> 93，

      -15.62 --> -16，-26.22 --> -26

值集合(续)

- 字符串——双引号内的字符序列，不能分行书写。

例：“INTERNAL ERROR”

“REACHED->HERE”

- 每个字符用一个8位ASCII值(无符号数)表示

例：reg [1: 8*14] message;

...

message = “INTERNAL ERROR”;

- 字符转义用反斜线(\)表示

\n	换行符
\t	制表符
\\	字符\本身
\”	字符”
\206	八进制数206 ASCII码对应的字符

数据类型

- Verilog有两大数据类型：线网和变量
- 优点：简洁；缺点：表达抽象数据能力弱
- 线网类型(Net Type)：对应物理连线，由元件(包括门原语、模块实例、连续信号赋值语句)驱动，没有驱动时值为z(默认值)。
- 变量类型(Variable Type)：表示抽象的数据存储单元，只能在过程语句(always和initial)中被赋值，变量的值可以一直保存到下一条赋值语句为止，默认值为x。(2001版之前被称为寄存器类型)

数据类型(续)

- 线网类型(Net Type)
  - wire，tri
  - wor，trior
  - wand，triand
  - trireg
  - tri1，tri0
  - supply1，supply0
  - 说明格式：

net_kind [signed] [[msb: lsb]] net1, net2, ..., netN;

线网类型关键字

表示具有符号值

位宽的范围

线网名称标识符

注意

wire == tri
wand == triand
wor == trior

线网(net)类型

- 例：

```
wire bist_ready, cnt_start;   //2个1位的线网
wand [2:0] haddr;             //haddr是3位向量的"线与"类型线网
`define SIZE 6
wire signed[7:0] pwdata, prdata;        //符号数线网
wire signed[`SIZE-1:0] usb_data;        //形式为2的补码
```

- 多驱动源的情况：当同一个线网有多个驱动时，不同的线网类型得到的有效值不同。

```
wor qma_ready;     wire kbl_clear, kbl_enable;
…
assign qma_ready = bist_ready & cnt_start;
assign qma_ready = kbl_clear | kbl_enable;
```

qma_ready有两个驱动源

线网(net)类型(续)

- wire和tri——完全一样，用tri只是为了明确说明有多个驱动源驱动同一个线网。

例：wire reset;

```
wire [3:1] mode_enable, clk_enable, clk_mode;
parameter MSB=8, LSB=0;
 tri [MSB-1:LSB+1] rtc_status;
```

多驱动源驱动一个wire或tri时，有效值为：

wire/tri	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	z

线网(net)类型(续)

- 例

```
assign mode_enable=clk_enable & clk_mode;
...
assign mode_enable=clk_enable ^ clk_mode;
```

如果第一个表达式值为01x, 第二个表达式值为11z，
则mode_enable的值为x1x。

- wor和trior——线或，有1则为1。其他与前面类似

wor/trior	0	1	x	z
0	0	1	x	0
1	1	1	1	1
x	x	1	x	x
z	0	1	x	z

线网(net)类型(续)

- wand和triand——线与，有零则为0

wand/triand	0	1	x	z
0	0	0	0	0
1	0	1	X	1
x	0	x	x	x
z	0	1	x	z

- trireg——存储数值，用于电荷存储结点(即电容结点)的建模，它有两个状态：
  - 驱动状态——有一个驱动源给trireg赋值成1，0，或x。
  - 容性状态——驱动源变成高阻(z)时，trireg保存其上的最后一个值。
  - trireg的默认值是x。详细用法在第5章给出。
  - trireg [1:8] bmc_datain, bmc_dataout;

线网(net)类型(续)

- tri0和tri1——模拟上拉和下拉电阻结点
- 当无驱动源的时候，tri0线网的值为0，tri1线网的值为1。这些值的强度为pull。当遇到有驱动源的时候，该线网的值是驱动源的值。

```
tri0 [-3:3] ground_bus;  //下标范围可以是负数
tri1 [0:-5] otp_bus, itp_bus;
```

- 存在多驱动源时，tri0和tri1线网的有效值：

tri0(tri1)	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	0(1)

线网(net)类型(续)

- supply0和supply1——“地”和“电源”建模

例：`define RANGE [2:0]

```
supply0 logic_0, clk_groud, VSS;
supply1 `RANGE logic_1, VDD;
```

线网(net)类型(续)

- 未声明的线网类型
  - 不声明就可以使用的线网类型，默认为1位的wire
  - 用`default_nettype改变默认的线网类型；
  - 例：`default_nettype wand

    不声明的默认线网改成了1位的线与类型。
  - 强迫线网类型必须明确地先声明才能使用

    `default_nettype none

线网(net)类型(续)

- 向量线网和标量线网
  - 关键字scalared和vectored的区别
  - wire vectored [3:1] grb_count; //不允许位选择和部分选择
  - wor scalared [4:0] bst_addr;

// 与wor [4:0] bst_addr相同，允许位选择和部分选择

  - 不写关键字scalared或者vectored，默认为标量(scalared)。

线网(net)类型(续)

- 部分选择——访问矢量的一部分数据
- 位选择——访问矢量的一位
- 例：

```
wire [7:0] da;
wire [3:0] db;
wire dc;
assign db=da[7:4];   //部分选择
assign dc=da[0];     //位选择
```

- 变量类型同样也有部分选择和位选择

数据类型——变量类型(原为寄存器)

- 变量类型——reg、integer、time、real、realtime五种；
- reg变量类型——Verilog中最常见的数据类型，默认值x
  - 说明格式：reg [signed] [[msb:lsb]] reg1, reg2, ..., regN;
  - 例： reg [3:0] ext_bus; //为4位的变量
    reg test_reqa; //1位变量
    reg [1:32] sm_datain, sm_address, tc_bus;
  - reg的位宽可以取任意值，通常为无符号数。使用signed后，表示为有符号数，以2的补码形式保存。

```
reg signed [1:4] xfer_rsp;
……
xfer_rsp = -2;  //xfer_rsp的二进制为1110，2的补码。
xfer_rsp = 5;   // xfer_rsp的值为5(0101)。
parameter MSB=16, LSB=1;
reg signed [MSB: LSB] usim_counter;   //2的补码有符号数
```

变量类型(续)

- 存储器——reg变量组成的一维数组
  - 说明格式：reg [[msb:lsb]] memory1 [up1: low1],
    memory2 [up2: low2], ...;
  - 例：

```
reg [0:3] ebi_mem [0:63]; //ebi_mem是64个4位reg变量组成的数组。
reg gnt_rfile [1:5];      //gnt_rfile是5个1位reg变量组成的数组。
reg [1:5] bog;            //一个5位寄存器

parameter ADDR_SIZE=16, WORD_SIZE=8;
reg [1:WORD_SIZE] par_ram [ADDR_SIZE-1:0], dburst_reg;
//par_ram是16个8位reg变量组成的数组，dburst_reg
//是一个8位reg变量。
```

变量类型(续)

- 单个reg变量可以用一个赋值语句完成赋值，存储器要用多个赋值语句完成赋值。
- 例：

```
reg [1:5] qburst;  qburst = 5'b11011;
reg hold_gnt [1:5];  hold_gnt = 5'b11011; ×
      hold_gnt[1]=1'b1;
      hold_gnt[2]=1'b1;
      hold_gnt[3]=1'b0;
      hold_gnt[4]=1'b1;
      hold_gnt[5]=1'b1;
```

```
reg [0:3] xp_rom [1:4];
…
xp_rom [1] = 4'hA;
xp_rom [2] = 4'h8;
xp_rom [3] = 4'hF;
xp_rom [4] = 4'h2;
```

变量类型(续)

- 存储器之间的赋值——第一种方法

```
parameter WORD_LENTH=8, NUM_WORDS=64;
reg [WORD_LENTH-1:0]
      mem_a [NUM_WORDS-1:0],
      mem_b [NUM_WORDS-1:0];
integer i;
// mem_a = mem_b; 是不允许的
for (i=0;i<NUM_WORDS;i=i+1)
       mem_a[i] = mem_b[i];
```

变量类型(续)

- 使用系统任务$readmemb和$readmemh给存储器赋值。用法一样，只是数据格式不同，b——二进制数，h——十六进制数。
  - 例：reg [1:4] cdn_rom [7:1];
    $readmemb(“ram.patt”, cdn_rom);

ram.patt的内容：

1101 //赋值给cdn_rom[7]
1110 //赋值给cdn_rom[6]
1000
0111
0000
1001
0011 //赋值给cdn_rom[1]

  - $readmemb(“ram.patt”, cdn_rom, 5, 3); // cdn_rom[5], cdn_rom[4], cdn_rom[3]从文件头读取，值为1101、1110、1000。
  - 二进制文件可以包含地址，格式为 @hex_address value。

变量类型(续)

例如，ram.patt的内容可以改为：

@6 1101

@4 1110

@2 1000

@1 0111

@3 0000

@5 1001

@7 0011

- $readmemb(“ram.patt”, cdn_rom);会按照ram.patt中的地址给存储器赋值。

变量类型(续)

- integer变量——整数，用于高层次行为建模

integer integer1, integer2, ..., intergerN [msb:lsb];

- 整数在硬件上至少有32位，存储有符号数，提供2的补码运算结果。可以提供更多的位。
- 整数和位向量可以自动转换，左边多余位被截断。
- 例：

```
integer a, b, c;//3个整数变量
integer hist[3:6];
        //4个整数组成的数组
reg [31:0] sel_reg;
integer sel_int;
   //允许sel_int[6]和sel_int[20:10]，
   //最小下标为0
sel_reg = sel_int;
   //允许sel_reg[6]
```

```
integer j;
reg [3:0] bcq;
j=6;        //j值为32'b0000...00110
bcq=j;     // bcq的值为4'b0110。

bcq=4'b0101;
j=bcq;    //j值为32'b0000...00101
j=-6;      //j值为32'b1111...11010
bcq=j;    //bcq值为4'b1010
```

变量类型(续)

- 整型变量使用例子

```
module mod_int_array;
    localparam ARRAY_SIZE = 8;
                //不能通过defparam或者实例引用修改，
                //但可以用parameter赋值
    integer int_array [0:ARRAY_SIZE-1];
    initial
        begin
            int_array[0] = 56;
            int_array[1] = 12;
            int_array[2] = int_array[0] / 2;
            $display (“int_array[2] is %d”, int_array[2]);
        end
endmodule
```

变量类型(续)

- time变量——用于存储和处理时间

```
time time_id1, time_id2, ..., time_idN [msb:lsb];
```

  - time变量存储的时间值至少为64位；
  - time变量是无符号数；

```
time events [0:31];   //时间变量数组
time curr_time;       // curr_time存储一个时间值
```

变量类型(续)

- real和realtime变量——实数和实数时间

real real_reg1, real_reg2, ..., real_regN;

  - real和realtime完全相同，可以互换使用，默认值为0；
  - 不允许声明值域、位界限或字节界限；
  - 将x和z赋值给real或realtime，这些值作0处理。

例：real amr_count;

```
...
amr_count = 'b01x1z; //结果为'b01010
```

变量类型(续)

- 数组——线网和变量的多维数组可以用一条数组语句声明，数组元素可以是标量或者向量。

```
wire push_bus [0:4]; //一个由5个1位线网组成的数组
reg [0:7] smc_fifo [0:63], req_stack [0:63];
  //64个元素组成的数组，每个元素是8位向量
tri [0:31] biq_addr [0:1][0:3];
 //三态线网的2维数组，每个元素位宽为32
integer run_stats [0:15][0:15];
 //16x16的数组，每个元素是整型变量
```

一维reg变量的数组被称为存储器

变量类型(续)

- 数组的赋值操作：只能对数组的一个元素进行赋值操作。选择数组元素中的某个位或者某些位进行存取或者赋值操作是可以的。

```
smc_fifo[5]=26;            //给数组的第5个元素赋值可以
smc_fifo=req_stack;        //不能给一个完整数组赋值

biq_addr[0][1]=32'b0;
biq_addr[1][0]=32'b1;
push_bus[0:2]=1;           //不能同时给数组的几个元素或者
                           //某个范围内的元素赋值
biq_addr[0][0][0:5]=6'b100011; //可以给数组中一个元
                               //素的部分位赋值。
```

变量类型(续)

- reg和 wire的不同点

reg：变量
- 只能在always和initial语句中赋值；
- 初始化值为X；
- 不能被赋予强度值；

wire：线网
- 只能用连续信号赋值语句赋值；或者通过模块实例的输出端口赋值；
- 初始化值为Z；
- 可以被赋予强度值；

参数

- 参数是Verilog中的常量。
  - 参数用于定义延时和变量宽度。
  - 用参数声明语句，参数只被赋值一次，且一直保持不变。

```
parameter [signed] [[msb: lsb]] param1 = const_expr1, param2 =
    const_expr2, ..., paramN = const_exprN;
```

例：

```
parameter LINELENGTH = 132, ALL_X_S = 16'bx;
// LINELENGTH的范围是[31:0]，ALL_X_S的范围是[15:0]
parameter BIT=1, BYTE=8, PI=3.14;
parameter STROBE_DELAY=(BYTE+BIT)/2;
parameter TQ_FILE= "/home/bhasker/TEST/add.tq";
parameter signed [3:0] MEM_DR=-5, CPU_SPI=6;
// MEM_DR和CPU_SPI是4位有符号数
```

参数

```
`define WIDTH 16
parameter [`WIDTH-1:0]RED=0, BLUE=1,YELLOW=2;
parameter `LINEAR_SIZE=2*MEM_DR+CPU_SPI;
```

- 参数可以在编译时被改变，有两种方法改变参数值：参数定义语句和模块实例化语句中参数值传递。见第9章。
- 参数声明语句也可以指定一种类型，如整数、实数、时间或者实型时间类型等，但不能使用范围和signed关键字。

```
parameter time TRIG_TIME=10,APPLY_TIME=25;
parameter integer COUNT_LIMIT=25;
```

参数

- parameter和define的区别
  - parameter是局部的，只能用在其被定义的module内部；
  - define是全局的，对同时编译的多个文件起作用；
  - parameter是数值替换，而define是字符串替换；
  - 建议：用参数定义常数；用宏定义指定ASCII文本。

参数

- 局部参数——localparam
  - 局部参数是模块内部的参数；
  - 在该模块实例引用时不能通过参数传递和defparam语句修改局部参数的值；
  - 其他与parameter的使用相同；

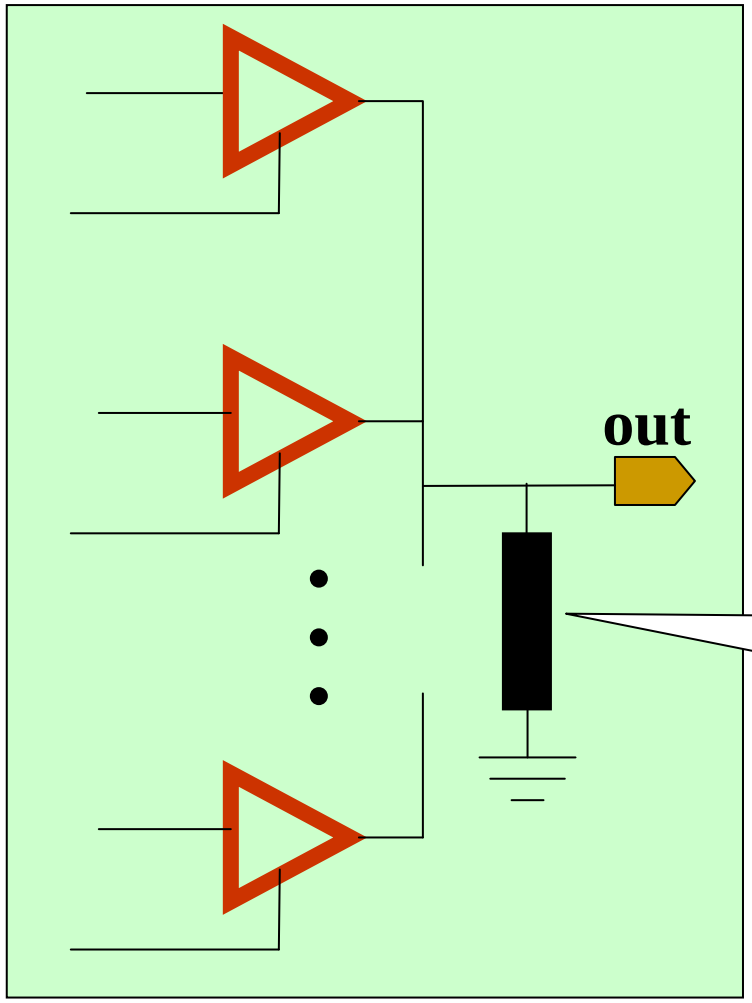
```
localparam TC_IDLE=2'b00;
localparam [3:0] INCR_BY=12;
localparam signed [7:0] TSM_MAX=56;
localparam real TWO_PI=2*3.14;
```

  - 如果局部参数用其他非局部参数定义，则外部赋值使参数变化时，局部参数值间接地发生改变。

```
parameter BYTE=8;
localparam NIBBLE = 2*BYTE;
```

若编译时，BYTE改变，则NIBBLE也将随之发生改变。

Bus Holder模型



Case Study 1

System bus is driven by trisate buffer. To prevent the bus from floating, designer uses “bus-holder” cell to hold the bus to the previous driven logic value.

Bus Holder模型Verilog实现

Case Study 1

One way to code bus holder in Verilog is:

```
module busholder (Y);
inout Y;
trireg y;
endmodule
```