
# Verilog HDL语言规范

主 讲：何宾

**Email：hebin@mail.buct.edu.cn**

**2014.06**

# Verilog HDL行为描述语句

本部分介绍行为描述语句。通过行为级建模把一个复杂的系统分解成可操作的若干个模块，每个模块之间的逻辑关系通过行为模块的仿真加以验证。同时行为级建模还可以用来生成仿真激励信号，对已设计模块进行仿真验证。

# Verilog HDL行为描述语句
## --过程语句

过程分配用于更新reg，integer，time，real，realtime和存储器数据类型。对于过程分配和连续分配来说，有下面的不同之处：

- 连续分配

连续分配驱动网络。只要一个输入操作数的值发生变化，则更新和求取所驱动网络的值。

- 过程分配

在过程流结构的控制下，过程分配更新流结构内变量的值。

# Verilog HDL行为描述语句
## --过程语句

过程分配的右边可以是求取值的任何表达式。左边应该是一个变量，它接收右边表达式分配的值。

过程分配的左边可以是下面的一种格式：

- reg、integer、real、realtime或者time数据类型分配给这些数据类型所引用的名字。
- reg、integer、real、realtime或者time数据类型的位选择分配到单个的比特位

# Verilog HDL行为描述语句
# --过程语句

- reg、integer、real、realtime或者time数据类型的部分选择

  一个或者多个连续的比特位的部分选择。

- 存储器字

  存储器的单个字

- 任何上面的并置（连接）或者嵌套的并置（连接）

  上面四种形式的并置或者嵌套的并置。这些语句对右边的表达式进行有效的分割，将分割的部分按顺序分配到并置或者嵌套并置的不同部分中。

# Verilog HDL行为描述语句
# --过程语句

## Verlig HDL包含两种类型的过程赋值语句：

- 阻塞过程分配（赋值）语句
- 非阻塞过程分配（赋值）语句

# Verilog HDL行为描述语句
# --过程语句

## 阻塞过程分配

- 以分配操作符“=”来标识分配的操作称为阻塞过程分配。阻塞分配语句不会阻止并行块内阻塞过程分配语句后面语句的执行。

# Verilog HDL行为描述语句
# --过程语句

**阻塞过程分配Verilog HDL描述的例子。**

```
rega = 0;
rega[3] = 1;                    //位选择
rega[3:5] = 7;                  //部分选择
mema[address] = 8'hff;          //分配到一个存储器元素
{carry, acc} = rega + regb;     //并置（连接）
```

# Verilog HDL行为描述语句
# --过程语句

## 非阻塞过程分配

- 非阻塞过程分配允许分配调度，但不会阻塞过程内的流程。在相同的时间段内，当有多个变量分配时，使用非阻塞过程分配。这个分配不需要考虑顺序，或者互相之间的依赖性。

# Verilog HDL行为描述语句
# --过程语句

- 以操作符“<=”来标识非阻塞过程分配。
  - ✓ 其形式和小于等于操作符是一样的。
- 赋值操作出现在initial和always块语句中。在非阻塞赋值语句中，赋值符号“<=”左边的赋值对象也必须是寄存器型变量，不像在阻塞过程赋值语句那样在语句结束时即刻得到，非阻塞赋值在该块语句结束才可得到值。

# Verilog HDL行为描述语句
# --过程语句

## 也可以这样理解这两种语句

- 阻塞过程分配没有“时序的概念”，这一点和c语言是一致的，
- 非阻塞过程分配有“时序的概念”，这一点和c语言是有区别的。

# Verilog HDL行为描述语句
# --过程语句

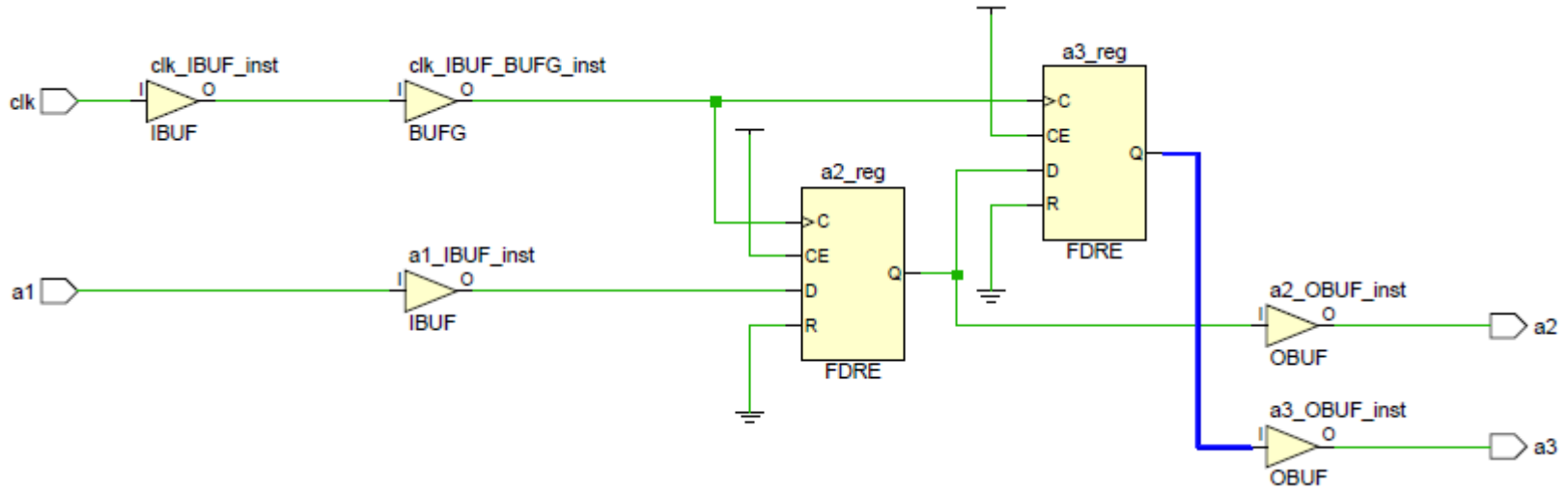
## 非阻塞过程分配Verilog HDL描述的例子1

```
module block(a3,a2,a1,clk);
input clk,a1; output reg a3,a2;
always @(posedge clk)
begin
        a2<=a1;
        a3<=a2;
end
endmodule
```

该设计保存在本书提供资料的 \eda_verilog\non_blocking目录下

# Verilog HDL行为描述语句
# --过程语句

## Vivado综合后得到的模块结构



# Verilog HDL行为描述语句
# --过程语句

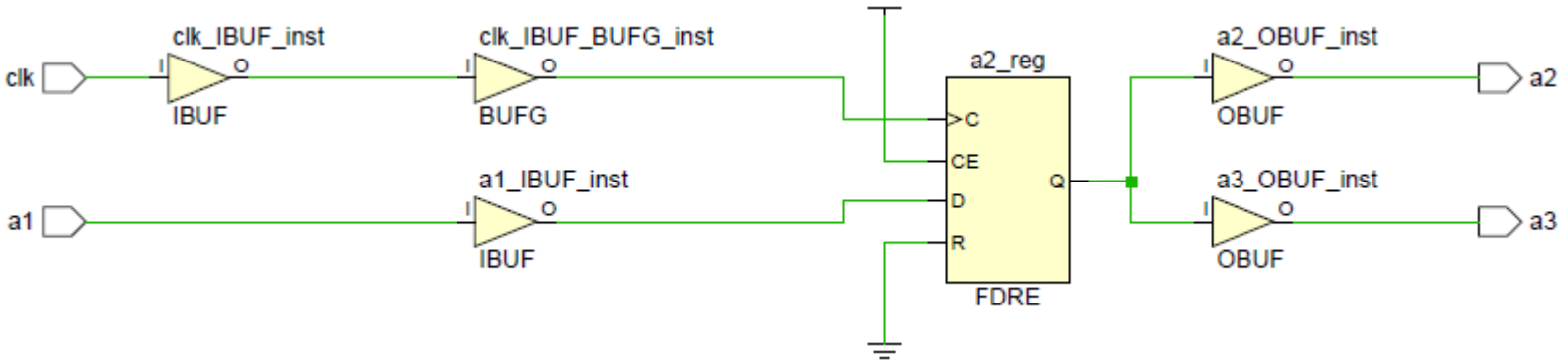
## 阻塞过程分配Verilog HDL描述的例子1

```
module block(a3,a2,a1,clk);
input clk,a1; output reg a3,a2;
always @(posedge clk)
begin
      a2=a1;
      a3=a2;
end
endmodule
```

**该设计保存在本书提供资料的 \eda_verilog\blocking目录下**

# Verilog HDL行为描述语句
# --过程语句

## Vivado综合后给出的模块结构



# Verilog HDL行为描述语句
# --过程语句

从上述两个例子中，可以清楚地看到两种过程分配语句的明显不同。

- 非阻塞过程分配语句，是并发执行的。
- 阻塞过程分配语句，是按照指定顺序执行的。
  - ✓ 分配的书写顺序对执行的结果也有着直接的影响。

# Verilog HDL行为描述语句
# --过程语句

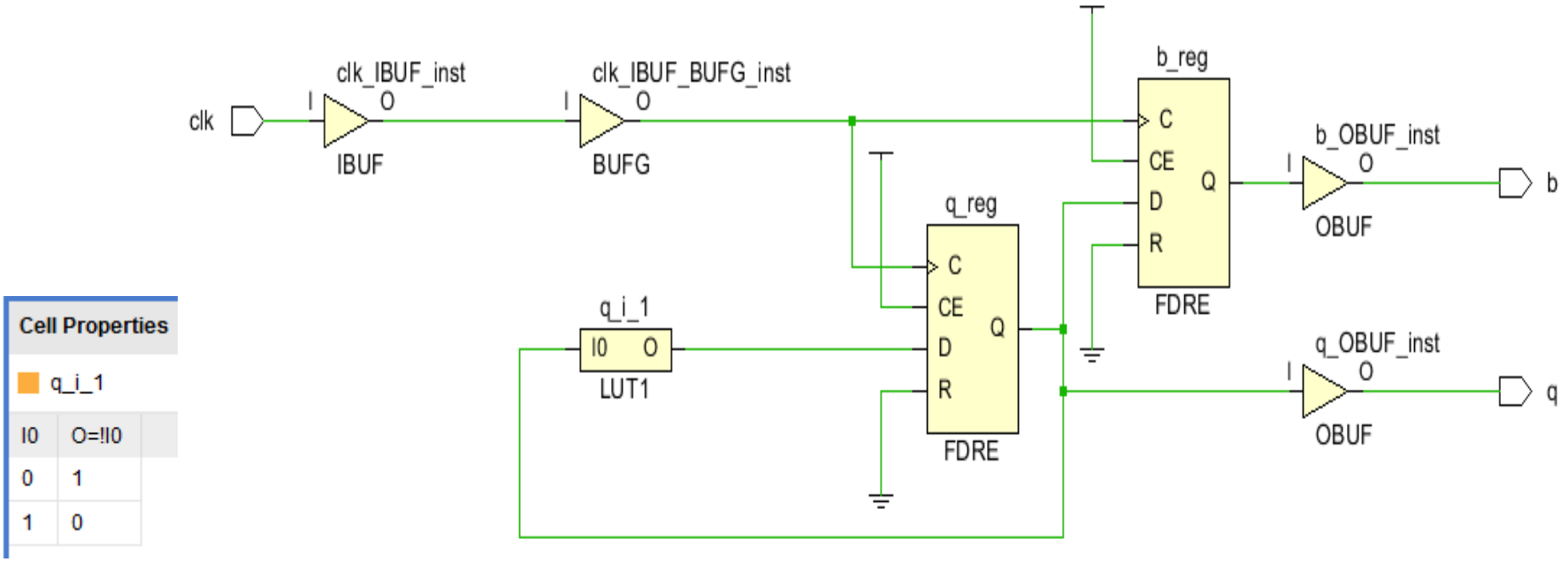
## 阻塞过程分配Verilog HDL描述的例子2

```
module block(q,b,clk);
input clk; output reg q,b;
always @(posedge clk)
begin
    q=~q;
    b=~q;
end
Endmodule
```

**该设计保存在本书提供资料的 \eda_verilog\blocking_1目录下**

# Verilog HDL行为描述语句
# --过程语句

## Vivado综合后得到的模块结构



# Verilog HDL行为描述语句
# --过程语句

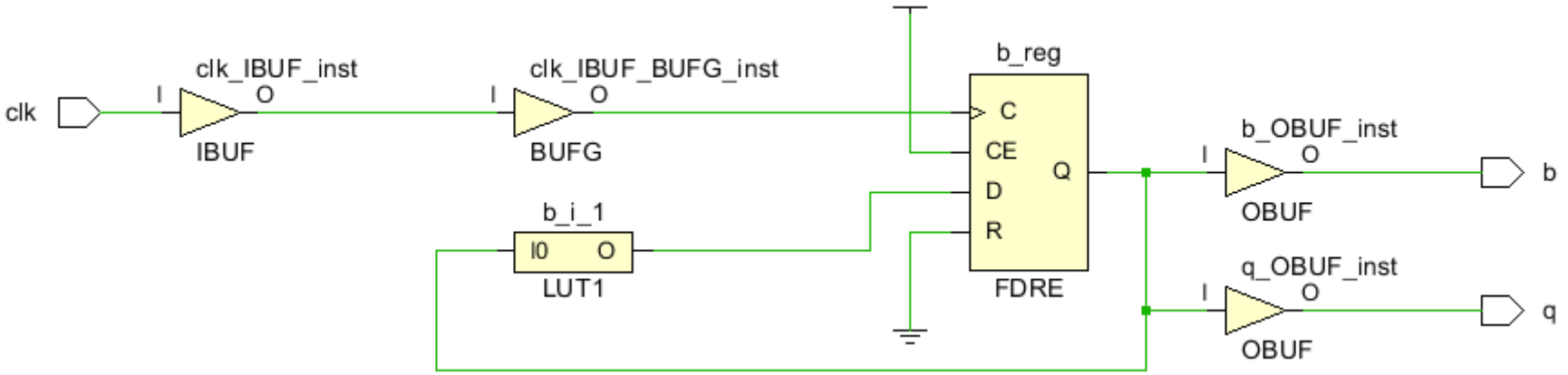
## 阻塞过程分配的Verilog HDL描述的例子3

```
module block(q,b,clk);
input clk; output reg q,b;
always @(posedge clk)
begin
      b=~q;
      q=~q;
end
endmoduld
```

**该设计保存在本书提供资料的 \eda_verilog\blocking_2目录下**

# Verilog HDL行为描述语句
# --过程语句

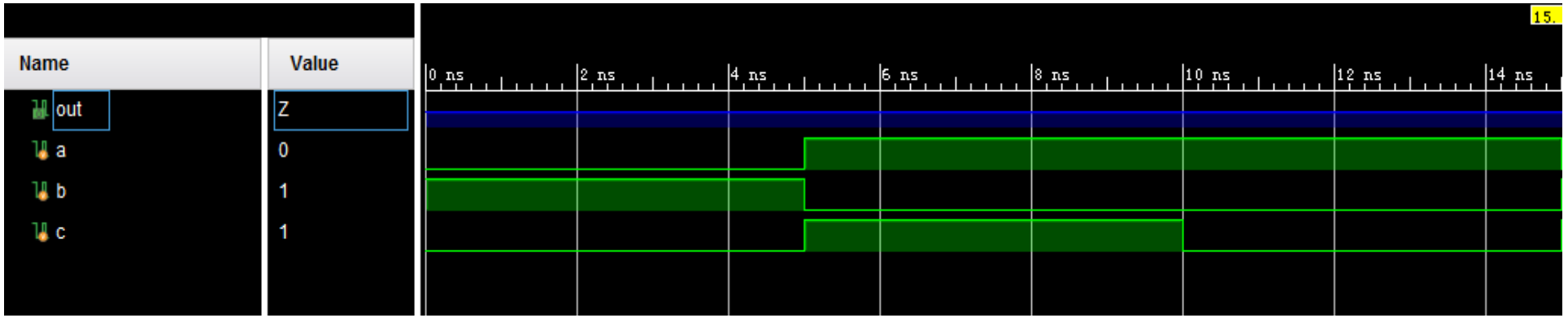
## Vivado综合后得到的模块结构



# 过程语句
## --阻塞和非阻塞过程分配描述的例子1

```
module evaluates2 (out);
output out;
reg a, b, c;
initial
begin
      a = 0;
      b = 1;
      c = 0;
end
```

```
always c = #5 ~c;
always @(posedge c)
begin
      a <= b;
      b <= a;
end
endmodule
```



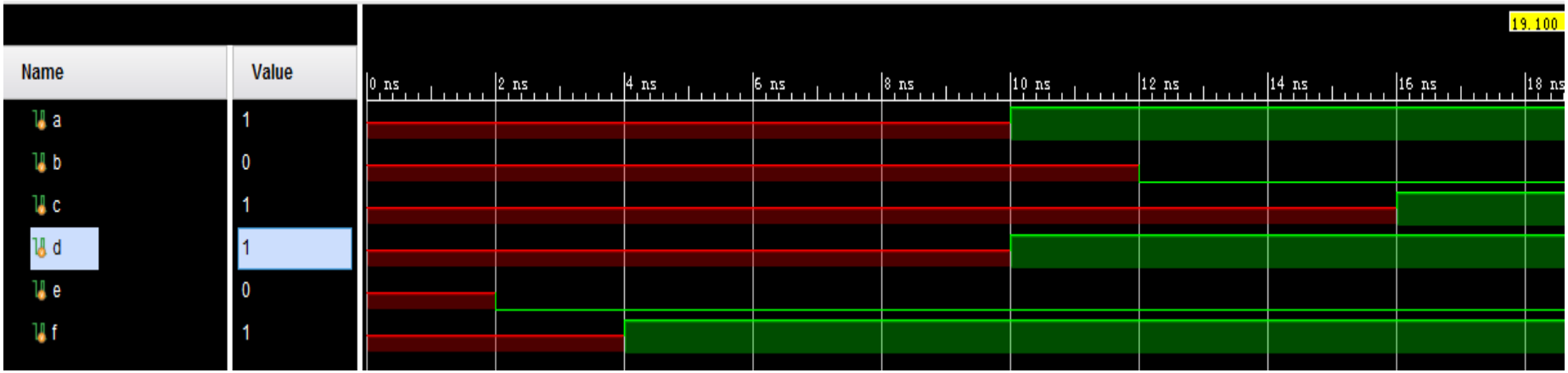

该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_1目录下

# 过程语句
## --阻塞和非阻塞过程分配的例子2

```
module non_block1;
reg a, b, c, d, e, f;
//阻塞分配
initial begin
        a = #10 1;      // 在第10个时间单位时，给a分配1。
        b = #2 0;       // 在第12个时间单位时，给b分配0。
        c = #4 1;       // 在第16个时间单位时，给c分配1。
end
//非阻塞分配
initial begin
        d <= #10 1;     // 在第10个时间单位时，给d分配1。
        e <= #2 0;      // 在第2个时间单位时，给e分配0。
        f <= #4 1;      // 在第4个时间单位时，给e分配1。
end
endmodule
```

该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_2目录下

# 过程语句
## --阻塞和非阻塞过程分配的例子2



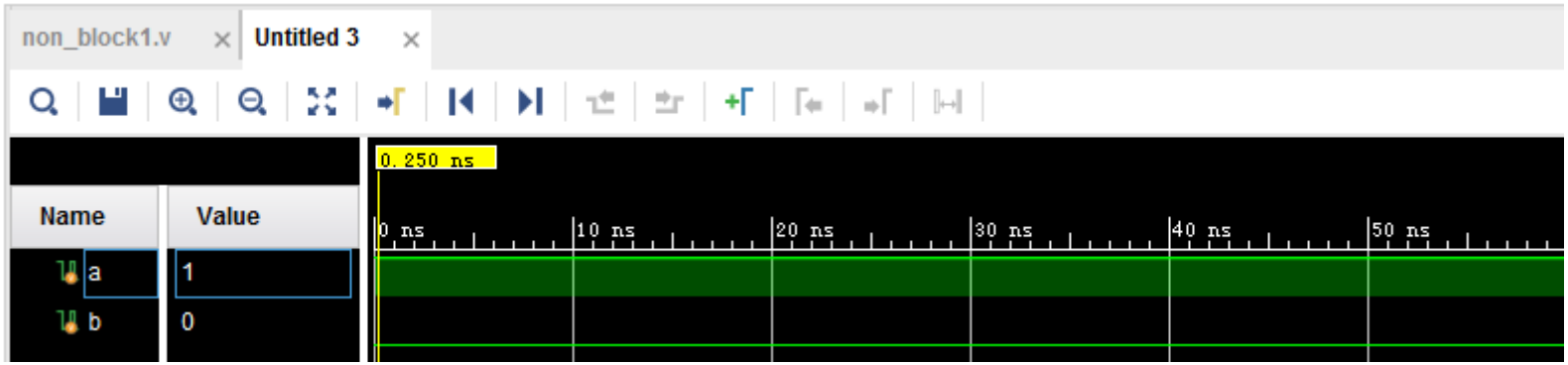

该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_2目录下

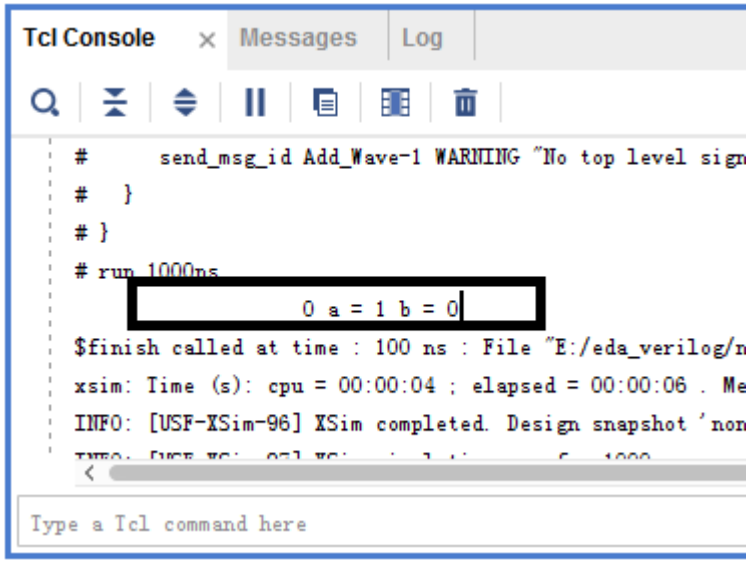
# 过程语句
## --阻塞和非阻塞过程分配的例子3

```
module non_block1;
reg a, b;
initial begin
        a = 0;
        b = 1;
        a <= b;
        b <= a;
end
initial
 begin
         $monitor ($time, ,"a = %b b = %b", a, b);
          #100 $finish;
end
endmodule
```

**该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_3目录下**

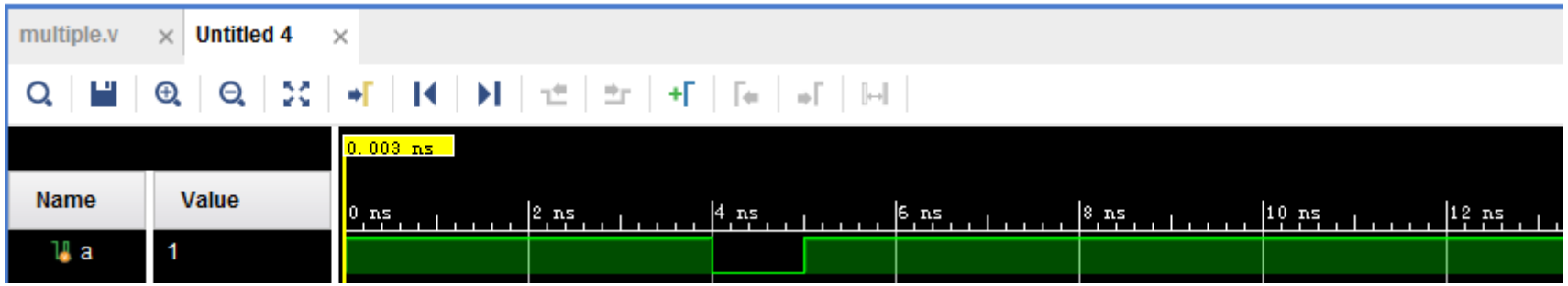
# 过程语句
## --阻塞和非阻塞过程分配的例子3



# 过程语句
## --阻塞和非阻塞过程分配的例子4

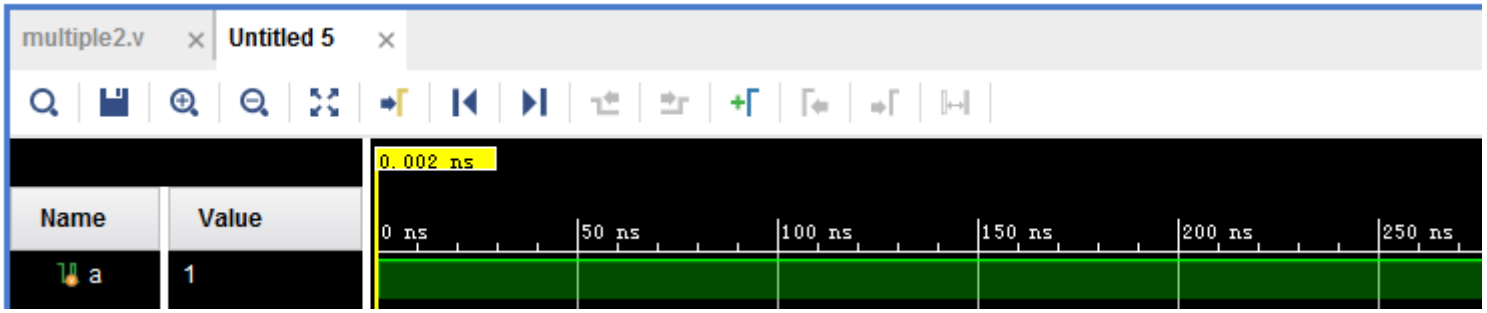
```
module multiple;
reg a;
initial a = 1;
initial begin
        a <= #4 0;    // 在第4个时间单位，调度a = 0
        a <= #5 1;    // 在第5个时间单位，调度a = 1
end
endmodule
```




该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_4目录下

# 过程语句
## --阻塞和非阻塞过程分配的例子5

```
module multiple;
reg a;
initial a = 1;
initial begin
      a <= #4 0;    // 在第4个时间单位，调度a = 0
      a <= #4 1;    // 在第4个时间单位，调度a = 1
end
endmodule
```

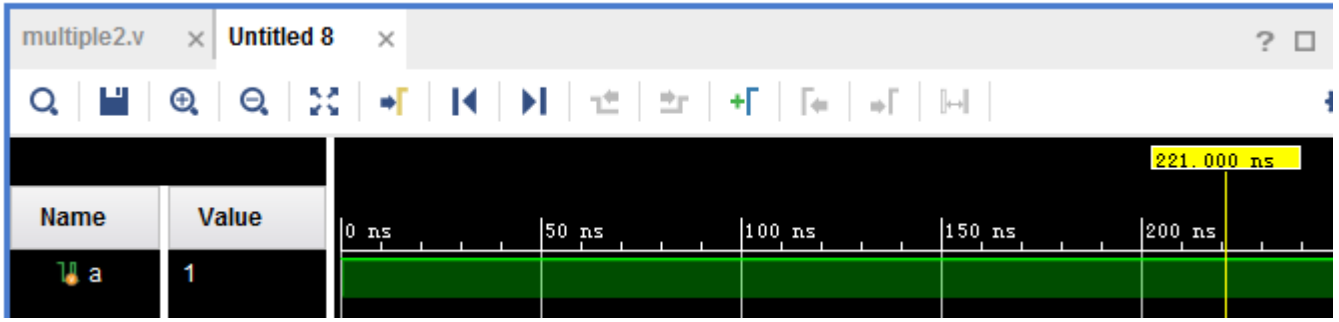



该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_5目录下

# 过程语句
## --阻塞和非阻塞过程分配的例子6

```
module multiple2;
reg a;
initial a = 1;
initial a <= #4  0;  //在第4个时间单位，调度a = 0
initial a <= #4  1;  //在第4个时间单位，调度a =1
// 在第4个时间单位, a = ??，寄存器分配的值是不确定的。
endmodule
```

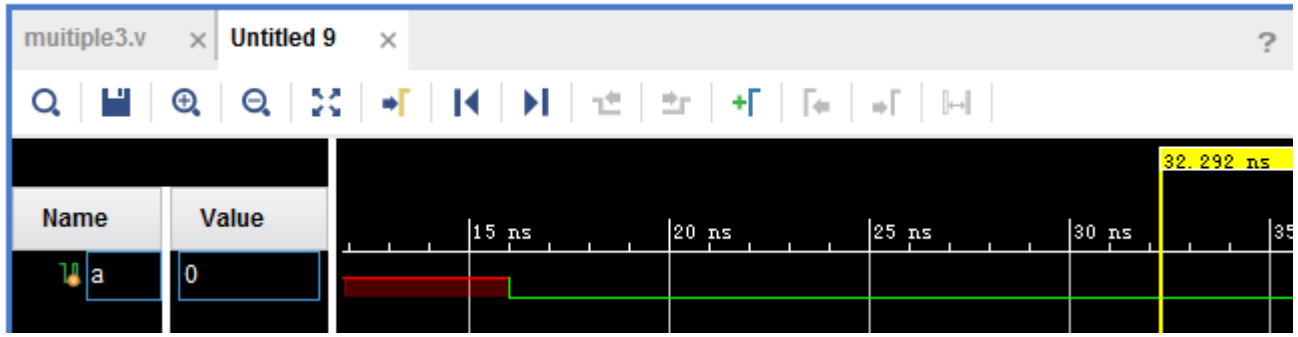
注：如果仿真器同时执行两个过程模块，如果过程模块包含对相同变量的非阻塞分配操作符，则变量最终的值是不确定的。




该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_6目录下

# 过程语句
## --阻塞和非阻塞过程分配的例子7

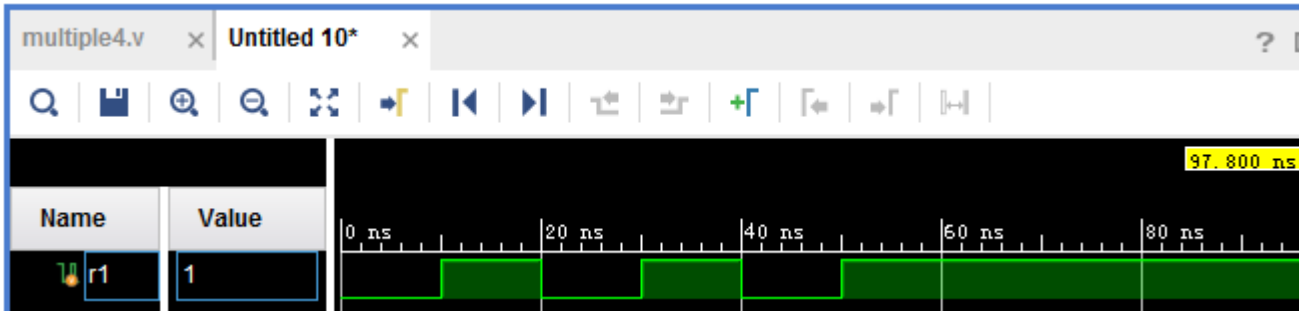
```
module multiple3;
reg a;
initial  #8 a <= #8 1;   //在第8个时间单位执行，在第16个时刻更新为1。
initial  #12 a <= #4 0;  //在第12个时间单位执行，在第16个时刻更新为0。
endmodule
```




该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_7目录下

# 过程语句
## --阻塞和非阻塞过程分配的例子8

```
module multiple4;
reg r1;
reg [2:0] i;
initial begin
for (i = 0; i <= 5; i = i+1)
        r1 <= # (i*10) i[0];
end
endmodule
```




该设计保存在本书提供资料的 \eda_verilog\non_blocking_and_blocking_8目录下

# Verilog HDL行为描述语句
## --过程连续分配

使用关键字assign和force的过程连续分配是过程语句。在变量或者网络上，允许连续驱动表达式。其语法格式如下：

```
assign variable_assignment
deassign variable_assignment
force variable_assignment
force net_assignment
release variable_lvalue
release net_lvalue
```

# Verilog HDL行为描述语句
## --过程连续分配

注：

- 对于assign语句的左边应该是一个参考变量或者变量的并置。它不能是存储器字（参考数组）或者一个变量的比特位选择或者部分选择。
- 对于force语句的左边应该是一个参考变量或者参考网络，它也可以是变量或者网络的并置。它不允许一个向量变量的比特位选择或者部分选择。

# Verilog HDL行为描述语句
# --过程连续分配

## assign和deassign过程语句

- assign过程连续分配语句将覆盖对变量的所有过程分配。
- deassign过程分配将终止对一个变量的过程连续分配。
- 变量将保持相同的值，直到通过一个过程分配或者一个过程连续分配语句，给变量分配一个新的值为止。例如：assign和deassign过程语句允许对一个带异步清除/置位端的D触发器进行建模。

# Verilog HDL行为描述语句
## --过程连续分配

**assign和deassignVerilog HDL描述的例子。**

```
module dff (q, d, clear, preset, clock);
output q;
input d, clear, preset, clock;
reg q;
always @(clear or preset)
    if (!clear)
        assign q = 0;
    else if (!preset)
        assign q = 1;
    else
        deassign q;
always @(posedge clock)
    q = d;
endmodule
```

# Verilog HDL行为描述语句
## --过程连续分配

- force和release过程语句
  - force和release过程语句提供了另一种形式的过程连续分配。
  - 这些语句的功能和assign/deassign类似。
  - force可以用于网络和变量。左边的分配可以是一个变量、网络、一个向量网路的常数比特位选择和部分选择、并置。它不能是一个存储器字（参考数组）或者一个向量变量的比特位选择和部分选择。

# Verilog HDL行为描述语句
## --过程连续分配

- 对一个变量的force将覆盖对变量的一个过程分配或者一个分配过程连续分配，直到对该变量使用了release过程语句为止。
- 当release时，如果当前变量没有一个活动的分配过程连续分配的话，不会立即改变变量的值。变量将保留当前的值，直到对该变量的下一个过程分配或者过程连续分配为止。

# Verilog HDL行为描述语句
## --过程连续分配

- 对一个网络的force过程语句，将覆盖网络的所有驱动器-门输出，模块输出和连续分配，直到在该网络上执行一个release语句为止。
- 当release时，网络将立即分配由网络驱动器所分配的值。

# Verilog HDL行为描述语句
## --过程连续分配

force和release的Verilog HDL描述的例子。

```
module test;
reg a, b, c, d;
wire e;
and and1 (e, a, b, c);
initial begin
    $monitor("%d d=%b,e=%b", $stime, d, e);
    assign d = a & b & c;
    a = 1;
    b = 0;
    c = 1;
    #10:
    force d = (a | b | c);
    force e = (a | b | c);
    #10;
    release d;
    release e;
    #10 $finish;
    end
    endmodule
```

最终的结果：

```
 0   d=0, e=0
10   d=1 ,e=1
20   d=0, e=0
```

# Verilog HDL行为描述语句
## --条件语句

条件语句（或者if-else语句）用于确定是否执行一个语句。

if-else语句有下面的三种表示方法：

- if(condition_1)
  procedural_statement1;
- if(condition_1)
  procedural_statement1;
  else
  procedural_statement2;

# Verilog HDL行为描述语句
# --条件语句

```
if(condition_1)
        procedural_statement_1;
else if(condition_2)
        procedural_statement_2;
……
else if(condition_n)
        procedural_statement_n;
else
        procedural_statement_n+1;
```

# Verilog HDL行为描述语句
## --条件语句

其中：

- conditon_1,….,condition_n为条件表达式。
- procedural_statement_1,….,procedural_ statement_n+1为描述语句。
- 如果对condition_1求值的结果为一个非零值，那么执行procedural_statement_1。如果condition_1的值为0、x或z，那么不执行procedural_statement_1。
- 描述语句可以使一个，也可以是多个。当有多个描述语句时，用“begin-end”语句将其包含进去。

# Verilog HDL行为描述语句
# --条件语句

注：

在if语句中，不要缺少else分支，否则会出现不同的综合结果，在后续章节会详细说明这个问题。

# 条件语句
## --if-else的例子

```
if (index > 0)
begin
        if (rega > regb)
                result = rega;
end
else
        result = regb;
```

# 条件语句
## --if-else if-else的例子

```
// 声明寄存器和参数
reg [31:0] instruction, segment_area[255:0];
reg [7:0] index;
reg [5:0] modify_seg1,
          modify_seg2,
          modify_seg3;
parameter
          segment1 = 0, inc_seg1 = 1,
          segment2 = 20, inc_seg2 = 2,
          egment3 = 64, inc_seg3 = 4,
          data = 128;
```

# 条件语句
## --if-else if-else的例子

```
// 测试索引变量
if (index < segment2) begin
        instruction = segment_area [index + modify_seg1];
        index = index + inc_seg1;
end
else if (index < segment3) begin
        instruction = segment_area [index + modify_seg2];
        index = index + inc_seg2;
end
else if (index < data) begin
       instruction = segment_area [index + modify_seg3];
       index = index + inc_seg3;
end
else
       instruction = segment_area [index];
```

# Verilog HDL行为描述语句
## --case语句

case语句是一个多条件分支语句，用于测试一个表达式是否匹配相应的其它表达式和分支。其语法格式如下：

```
case(case_expr)
   case_item_expr_1:  procedural_statement_1;
   case_item_expr_2:  procedural_statement_2;
                              ...
   case_item_expr_n:  procedural_statement_n;
   default                  :  procedural_statement_n+1;
endcase
```

# Verilog HDL行为描述语句
# --case语句

其中：

- case_expr为条件表达式。
- case_item_expr_1,...,case_item_expr_n为条件值。
- procedual_statement_1,...,procedual_statement_n+1为描述语句。

# Verilog HDL行为描述语句
# --case语句

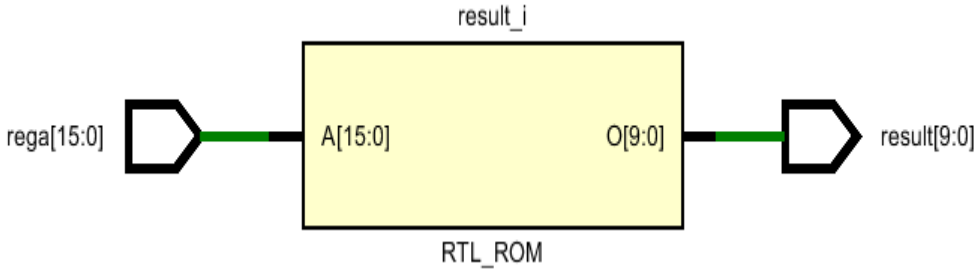
## case语句的规则包括：

- case语句首先对条件表达式case_expr求值，然后依次对各分支项求值并进行比较。将执行第一个与条件表达式值相匹配的分支中的语句。
- 可以在1个分支中定义多个分支项，这些值不需要互斥。
- 默认分支覆盖所有没有被分支表达式覆盖的其他分支。
- 分支表达式和各分支项表达式不必都是常量表达式。

# case语句
# --例子1

```
reg [15:0] rega;
reg [9:0] result;
case (rega)
  16'd0: result = 10'b0111111111;
  16'd1: result = 10'b1011111111;
  16'd2: result = 10'b1101111111;
  16'd3: result = 10'b1110111111;
  16'd4: result = 10'b1111011111;
  16'd5: result = 10'b1111101111;
  16'd6: result = 10'b1111110111;
  16'd7: result = 10'b1111111011;
  16'd8: result = 10'b1111111101;
  16'd9: result = 10'b1111111110;
default result = 'bx;
endcase
```

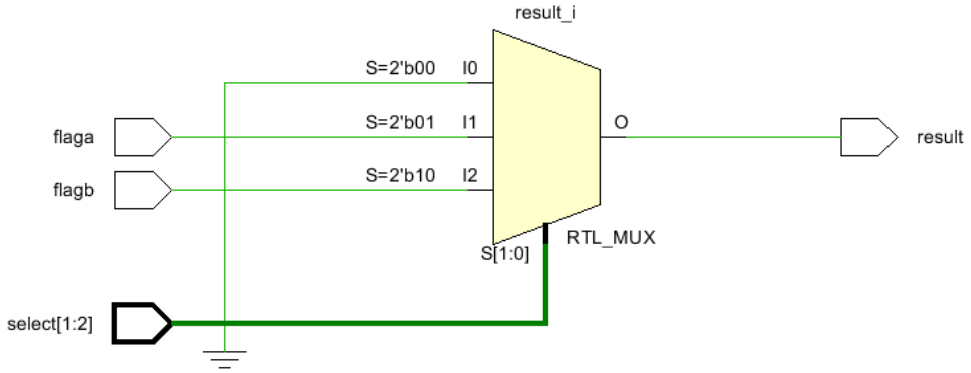
详细描述后的电路




该例子保存在本书配套资源e:\eda_verilog\case_1目录下

# case语句
# --例子2

```
case (select[1:2])
    2'b00: result = 0;
    2'b01: result = flaga;
    2'b0x,
    2'b0z: result = flaga ? 'bx : 0;
    2'b10: result = flagb;
    2'bx0,
    2'bz0: result = flagb ? 'bx : 0;
    default : result = 'bx;
endcase
```




该例子中，如果select[1]=0并且flaga=0时，即使 select[2]=’x’/’z’，结果也是0。

该例子保存在本书配套资源e:\eda_verilog\case_2目录下

# case语句
# --例子3

## case中x和z条件Verilog HDL描述的例子

```
case (sig)
        1'bz: $display("signal is floating");
        1'bx: $display("signal is unknown");
        default: $display("signal is %b", sig);
endcase
```

# Verilog HDL行为描述语句
## --case语句

Verilog HDL提供了case语句的其它两种形式：

- casex

  将x和z值都看作是无关位。

- casez

  将z值看作是无关值。

- 这些形式对x和z值使用不同的解释。除关键字是casex和casez以外，语法与case语句完全一致。

# Verilog HDL行为描述语句
## --casez语句的例子

```
reg [7:0] ir;
casez (ir)
        8'b1???????: instruction1(ir);
        8'b01??????: instruction2(ir);
        8'b00010???: instruction3(ir);
        8'b000001??: instruction4(ir);
endcase
```

# Verilog HDL行为描述语句
## ----casex语句的例子

```
reg [7:0] r, mask;
mask = 8'bx0x0x0x0;
casex (r ^ mask)
        8'b001100xx: stat1;
        8'b1100xx00: stat2;
        8'b00xx0011: stat3;
        8'bxx010100: stat4;
endcase
```

# Verilog HDL行为描述语句
## --case语句

此外，常数表达式也可以作为case表达式。常数表达式的值将要和case条目中的表达式进行比较，寻找匹配的条件。

# Verilog HDL行为描述语句
# --case语句

## case中常数表达式Verilog HDL描述的例子

```
reg [2:0] encode ;
case (1)
        encode[2] : $display("Select Line 2") ;
        encode[1] : $display("Select Line 1") ;
        encode[0] : $display("Select Line 0") ;
        default:  $display("Error: One of the bits expected ON");
endcase
```

# Verilog HDL行为描述语句
## --循环语句

Verilog HDL提供了四类循环语句，用于控制执行语句次数，这四种循环语句包括：

- forever循环
- repeat循环
- while循环
- for 循环

# 循环语句
## --forever循环语句

此循环语句连续执行过程语句。这一形式的循环语句语法格式如下：

```
forever
        procedural_statement；
```

为了退出这样的循环，可以共同使用中止语句和过程语句。同时，在过程语句中必须使用某种形式的时序控制。否则，forever循环将在0时延后永远循环下去。

# 循环语句
# --repeat循环语句

这种循环语句执行指定循环次数的过程语句。repeat循环语句语法格式如下：

```
repeat (loop_count)
        procedural_statement；
```

其中：

- loop_count为循环次数。
- procedural_statement为描述语句。
- 如果循环计数表达式的值不确定，则循环次数按0处理。

# 循环语句
# --repeat循环语句的例子

```
parameter size = 8, longsize = 16;
reg [size:1] opa, opb;
reg [longsize:1] result;
begin : mult
    reg [longsize:1] shift_opa, shift_opb;
    shift_opa = opa;
    shift_opb = opb;
    result = 0;
```

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

**思考与练习：该例子所要实现的功能？（两个8位数相乘）**

**该例子保存在本书配套资源e:\eda_verilog\repeat目录下**

# 循环语句
## --while循环语句

该循环语句循环执行过程赋值语句，直到指定的条件为假。

while 循环语句语法如下：

```
while(condition)
        procedural_statement；
```

- condition为循环的条件表达式。
- procedural_statement为描述语句。
- 如果表达式在开始时为假，那么永远不会执行过程语句。如果条件表达式为x或z，也同样按0（假）处理。

# 循环语句
# --while循环语句

```
begin : count1s
        reg [7:0] tempreg;
        count = 0;
        tempreg = rega;
while (tempreg) begin
        if (tempreg[0])
            count = count + 1;
            tempreg = tempreg >> 1;
        end
end
```

**注：while语句综合会有问题，慎用**

**思考与练习：该例子所要实现的功能？（统计一个8位数中1的个数）**

**该例子保存在本书配套资源e:\eda_verilog\while目录下**

# 循环语句
## --for循环语句

该循环语句按照指定的次数重复执行过程赋值语句若干次。

for循环语句的语法格式如下：

```
for(initial_assignment;condition;step_assignment)
    procedural_statement ;
```

其中：

- initial_addignment为初始值，用于提供循环变量的初始值。
- condition为循环条件表达式，条件表达式指定循环在什么情况下必须结束。

# 循环语句
## --for循环语句

- step_assigment给出要修改的赋值，通常为增加或减少循环变量计数。
- procedural_statement为描述语句。一个只要条件为真，就执行循环中的语句。

# 循环语句
## --for循环语句

```
module countzeros (a, Count);
input [7:0] a;
output reg[2:0] Count;
reg [2:0] Count_Aux;
integer i;
always @(a)
begin
   Count_Aux = 3'b0;
      for (i = 0; i < 8; i = i+1)
         begin
          if (!a[i])
             Count_Aux = Count_Aux+1;
         end
             Count = Count_Aux;
 end
endmodule
```

思考与练习：该例子所要实现的功能？（统计一个8位数中0的个数）

该例子保存在本书配套资源e:\eda_verilog\for目录下

# Verilog HDL行为描述语句
## --过程时序控制

当发生过程语句时，Verilog HDL提供了两种类型的明确的时钟控制，即：

- delay控制
- event表达式

# Verilog HDL行为描述语句
## --过程时序控制

### delay控制

- delay控制由#开头

### 延迟控制Verilog HDL描述例子1

```
#10 rega = regb;
```

# Verilog HDL行为描述语句
## --过程时序控制

### 延迟控制Verilog HDL描述例子2

```
#d rega = regb;                 //d定义为一个参数

#((d+e)/2) rega = regb;         //延迟是d和e的平均值

#regr regr = regr + 1;          // 延迟在regr寄存器中
```

# Verilog HDL行为描述语句
# --过程时序控制

## 事件控制

- 通过一个网络、变量或者一个声明事件的发生，来同步一个过程语句的执行。网络和变量值的变化可以作为一个事件，用于触发语句的执行。这就称为检测一个隐含事件。
- 事件基于变化的方向，即朝着值1（posedge）或者朝着0（negedge）。即：

# Verilog HDL行为描述语句
## --过程时序控制

- negedge
  - 检测到从1跳变到x、z或者0。
  - 检测到从x或z跳变到0。
- posedge
  - 检测到从0跳变到x、z或者1。
  - 检测到从x或z跳变到1。

# Verilog HDL行为描述语句
# --过程时序控制

| From | To | | | |
|---|---|---|---|---|
| | 0 | 1 | x | z |
| 0 | No edge | posedge | posedge | posedge |
| 1 | negedge | No edge | negedge | negedge |
| x | negedge | posedge | No edge | No edge |
| z | negedge | posedge | No edge | No edge |

# Verilog HDL行为描述语句
## --过程时序控制

### 边沿控制语句Verilog HDL描述的例子

```
@r rega = regb;                          // 由寄存器r内值的变化控制

@(posedge clock) rega = regb;            // 由时钟上升沿控制

forever @(negedge clock) rega = regb;  //由时钟下降沿控制
```

# Verilog HDL行为描述语句
# --过程时序控制

## 命名事件

- 除了网络和变量外，可以声明一种新的数据类型-事件。一个用于声明事件数据类型的标识符称为一个命名的事件。它可以用在事件表达式内，用于控制过程语句的执行。
- 命名的事件可以来自一个过程。这样，运行控制其它过程中的多个行为。

# Verilog HDL行为描述语句
# --过程时序控制

## 事件或者操作符

- 可以表示逻辑或者任何数目的事件，这样任何一个事件的发生将触发跟随在该事件后的过程语句。
- 用“，”分割的or关键字，用于一个事件逻辑或操作符。它们的组合同样可以用于事件表达式中。逗号分割的敏感列表和or分割的敏感列表是同步的。

# Verilog HDL行为描述语句
## --过程时序控制

多个事件逻辑或关系Veirlog HDL描述的例子。

```
@(trig or enable) rega = regb;      //由trig或者enable控制

@(posedge clk_a or posedge clk_b or trig) rega = regb;
```

# Verilog HDL行为描述语句
## --过程时序控制

使用逗号作为事件逻辑或操作符Veirlog HDL描述的例子。

```
always @(a, b, c, d, e)

always @(posedge clk, negedge rstn)

always @(a or b, c, d or e)
```

# Verilog HDL行为描述语句
## --过程时序控制

### 隐含的表达式列表

- 在RTL级仿真中，一个事件控制的事件表达式列表是一个公共的漏洞。
- 在时序控制描述中，设计者经常忘记添加需要读取的一些网络或者变量。
- 当比较RTL和门级版本的设计时，经常可以发现这个问题。
- 隐含的表达式，@*是一个简单的方法，用于解决忘记添加网络或者变量的问题。

# Verilog HDL行为描述语句
## --过程时序控制

### 隐含事件Verilog HDL描述的例子1

```
always @(*) // 等效于@(a or b or c or d or f)
y = (a & b) | (c & d) | myfunction(f);
```

# Verilog HDL行为描述语句
## --过程时序控制

### 隐含事件Verilog HDL描述的例子2

```
always @* // 等效于@(a or b or c or d or tmp1 or tmp2)
begin
        tmp1 = a & b;
        tmp2 = c & d;
        y = tmp1 | tmp2;
end
```

# Verilog HDL行为描述语句
## --过程时序控制

### 隐含事件Verilog HDL描述的例子3

```
always @*
begin          // 等效于@(b)
    @(i) kid = b;   // i没有添加到@*
end
```

# Verilog HDL行为描述语句
## --过程时序控制

### 隐含事件Verilog HDL描述的例子4

```
always @* //等效于@(a or b or c or d)
   begin
      x = a ^ b;
       @*                  // 等效于@(c or d)
      x = c ^ d;
   end
```

# Verilog HDL行为描述语句
## --过程时序控制

### 隐含事件Verilog HDL描述的例子5

```
always @* // 和 @(a or en)一样
begin
    y = 8'hff;
    y[a] = !en;
 end
```

# Verilog HDL行为描述语句
## --过程时序控制

### 隐含事件Verilog HDL描述的例子6

```
always @* begin // 等效于 @(state or go or ws)
        next = 4'b0;
case (1'b1)
        state[IDLE]: if (go) next[READ] = 1'b1;
                     else next[IDLE] = 1'b1;
        state[READ]: next[DLY ] = 1'b1;
        state[DLY ]: if (!ws) next[DONE] = 1'b1;
                     else next[READ] = 1'b1;
        state[DONE]: next[IDLE] = 1'b1;
endcase
end
```

# Verilog HDL行为描述语句
# --过程时序控制

## 电平敏感的事件控制

- 可以延迟一个过程语句的执行，直到一个条件变成真。
- 使用wait语句可以实现这个延迟控制，它是一种特殊形式的事件控制语句。
- 等待语句本质对电平敏感。
- 等待语句对条件进行评估。当条件为假时，在等待语句后面的过程语句将保持阻塞状态，直到条件变为真为止。

# Verilog HDL行为描述语句
## --过程时序控制

### 等待事件Verilog HDL描述的例子

```
begin
wait (!enable) #10 a = b;
                   #10 c = d;
 end
```

# Verilog HDL行为描述语句
## --过程时序控制

### 内部分配时序控制

- 前面介绍的延迟和事件控制结构，是在一个语句和延迟执行的前面。相比较而言，在一个分配语句中包含内部分配延迟和事件，以不同的方式修改活动的顺序。
- 内部分配的时序控制可以用于阻塞分配和非阻塞分配。repeat事件说明在一个事件发生了指定的数目后的内部分配延迟。
- 如果重复计数的数字有符号寄存器所保存的重复次数小于或者等于0，将立即产生分配，这就好像不存在重复结构。

# Verilog HDL行为描述语句
## --过程时序控制

**下表给出了内部分配时序控制的等效性比较。**

| 带有内部分配时序控制结构 | 没有内部分配时序控制结构 |
| --- | --- |
| a = #5 b; | begin<br>temp = b;<br>#5 a = temp;<br>end |
| a = @(posedge clk) b; | begin<br>temp = b;<br>@(posedge clk) a = temp;<br>end |

# Verilog HDL行为描述语句
## --过程时序控制

| 带有内部分配时序控制结构 | 没有内部分配时序控制结构 |
| --- | --- |
| a = repeat(3)<br>@(posedge clk) b; | begin<br>temp = b;<br>@(posedge clk);<br>@(posedge clk);<br>@(posedge clk) a = temp;<br>end |

# Verilog HDL行为描述语句
## --过程时序控制

所有在fork-join结构之间的语句都是并发执行的。

- 下面使用fork-join行为结构。

# Verilog HDL行为描述语句
## --过程时序控制

### fork-join结构的Verilog HDL描述的例子1

```
fork
    #5 a = b;
    #5 b = a;
join
```

上面的例子产生了竞争条件。

下面的描述消除了竞争条件。

```
fork // 数据交换
    a = #5 b;
    b = #5 a;
join
```

# Verilog HDL行为描述语句
## --过程时序控制

### fork-join结构的Verilog HDL描述的例子2

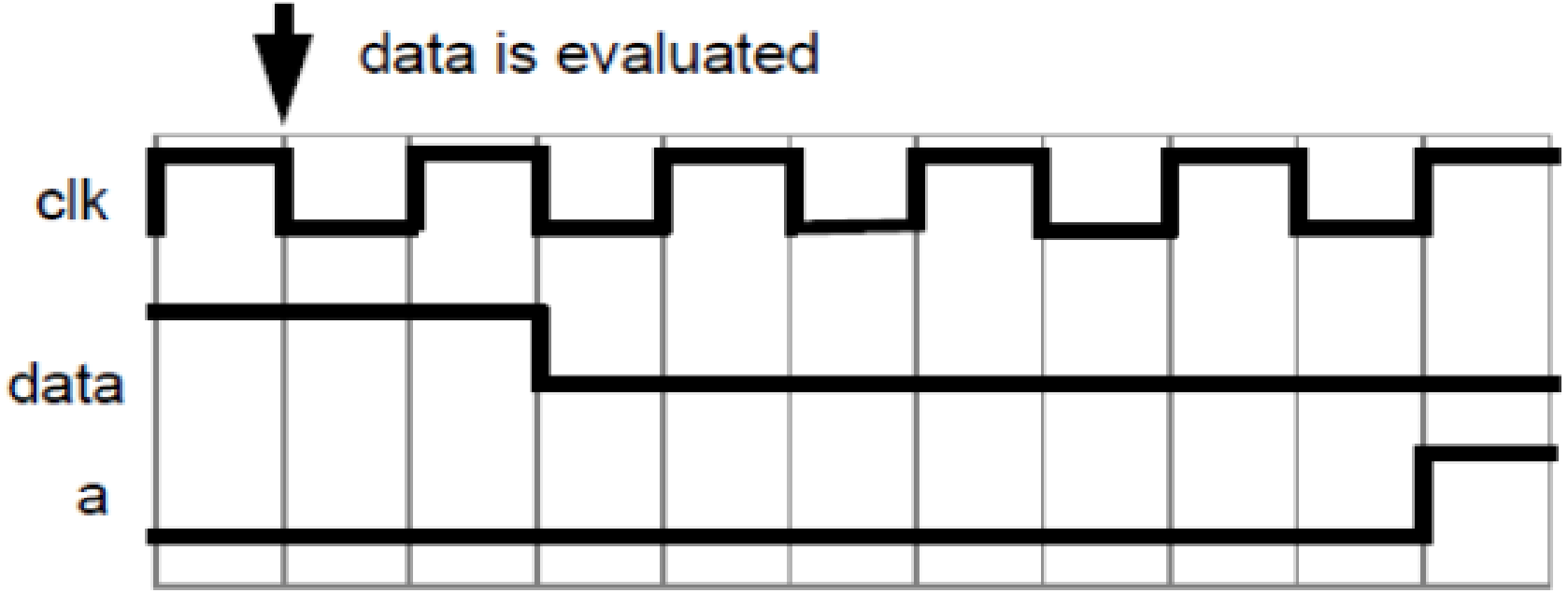
```
fork // 数据移位
    a = @(posedge clk) b;
    b = @(posedge clk) c;
join
```

# Verilog HDL行为描述语句
## --过程时序控制

### fork-join结构的Verilog HDL描述的例子3

```
a <= repeat(5) @(posedge clk) data;
```

下图给出了仿真波形图。在5个上升沿后给a分配值。



# Verilog HDL行为描述语句
# --过程时序控制

## fork-join结构的Verilog HDL描述的例子4

```
a <= repeat(a+b) @(posedge phi1 or negedge phi2) data;
```

# Verilog HDL行为描述语句
## --语句块

语句块提供一个方法，可以将多条语句组合在一起。这样，它们看上去好像一个语句。Verilog HDL中有两类语句块即：

- 顺序语句块(begin...end)

  语句块中的语句按给定次序顺序执行。

- 并行语句块(fork...join)

  语句块中的语句并行执行。

# Verilog HDL行为描述语句
# --语句块

## 顺序语句块

- 顺序语句块中的语句按顺序方式执行。
- 每条语句中的时延值与其前面的语句执行的仿真时间相关。
- 一旦顺序语句块执行结束，继续执行顺序语句块过程的下一条语句。

# Verilog HDL行为描述语句
# --语句块

## 顺序语句块的语法格式如下：

```
begin：<block_name>
   //declaration
   //behavior statement1
   ……
   //behavior statementn
end
```

# Verilog HDL行为描述语句
## --语句块

其中：

- block_name为模块的标识符，该标识符是可选的。
- declaration为块内局部变量的声明，这些声明可以是reg型变量声明、integer型变量声明及real型变量声明。
- behavior statement为行为描述语句。

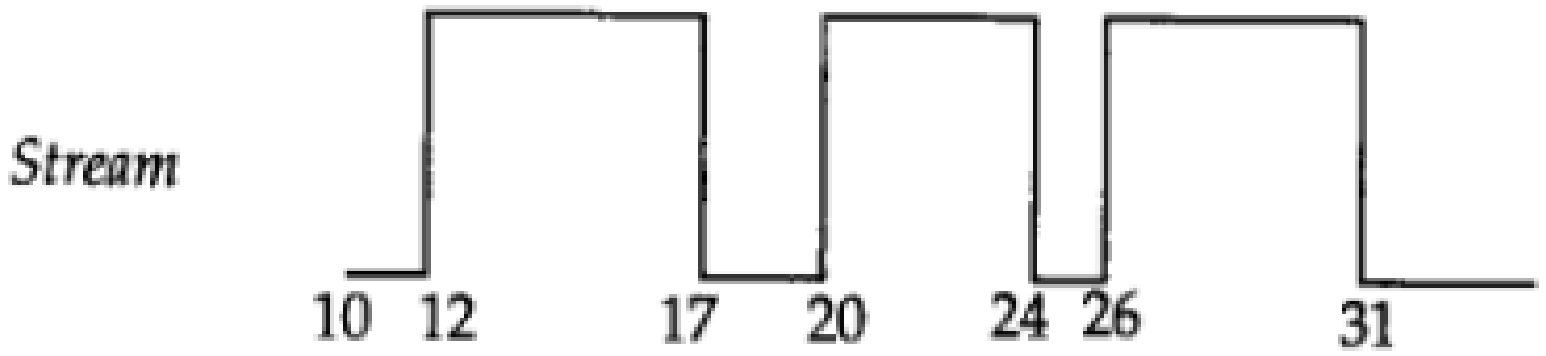
# Verilog HDL行为描述语句
# --语句块

## 顺序语句块Verilog HDL描述的例子1

```
begin
    #2 Stream=1;
    #5 Stream=0;
    #3 Stream=1;
    #4 Stream=0;
    #2 Stream=1;
    #5 Stream=0;
end
```

# Verilog HDL行为描述语句
## --语句块

说明： 如下图所示，假定顺序语句块在第10个时间单位开始执行。两个时间单位后，执行第1条语句，即在第12个时间单位执行第1条语句。当执行完该条语句后，在第17个时间（延迟5个时间单位）单位执行下一条语句。然后，在第20个时间单位执行下1条语句，以此类推。



# Verilog HDL行为描述语句
# --语句块

## 顺序语句块Verilog HDL描述的例子2

```
begin
    pat=mask|mat;
    @ (negedge clk) ;
    ff=& pat
end
```

在该例中，首先执行第1条语句。然后，执行第2条语句。当然，只有在clk上出现下降沿时才执行第2条语句中的赋值过程。

# Verilog HDL行为描述语句
# --语句块

## 并行语句块

- 并行语句块内的各条语句并行执行。
- 并行语句块内的各条语句指定的时延值都与语句块开始执行的时间相关。
- 当并行语句块中最后的动作执行完成后（执行的并不一定是最后的语句），继续执行顺序语句块的语句。换一种说法就是并行语句块内的所有语句必须在控制转出语句块前完成执行。

# Verilog HDL行为描述语句
## --语句块

**并行语句块语法如下：**

```
fork：<block_name>
        //declaration
        //behavior statement1
           ……
         //behavior statement2
 join
```

**其中：**

- block_name为模块标识符。
- declaration为块内局部变量声明，声明可以是reg型变量声明、integer型变量声明、real型变量声明、time型变量声明和事件（event）声明语句。

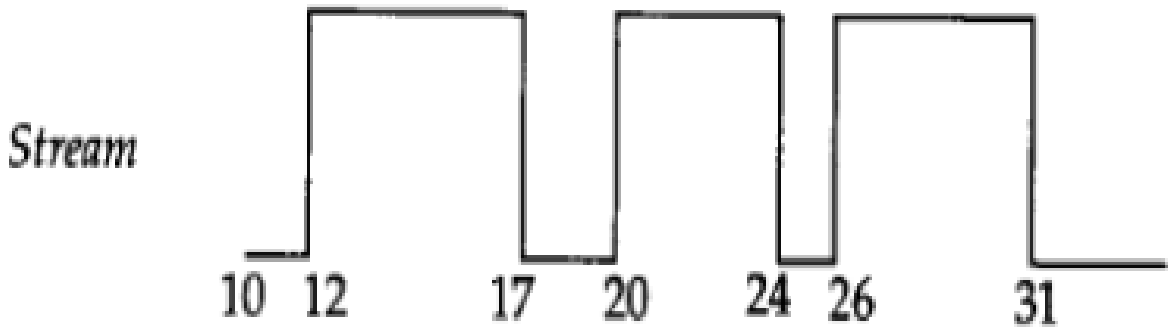
# Verilog HDL行为描述语句
# --语句块

## 并行语句块Verilog HDL描述的例子

```
fork
    #2 Stream = 1;
    #7 Stream = 0;
    #10 Stream = 1;
    #14 Stream = 0;
    #16 Stream = 1;
    #21 Stream = 0;
join
```

# Verilog HDL行为描述语句
## --语句块

如下图，如果在第10个时间单位开始执行并行语句块，所有的语句并行执行。并且，所有的时延都是相对于时刻10的。例如，在第20个时间单位执行第3个赋值，在第26个时间单位执行第5个赋值，以此类推。



# Verilog HDL行为描述语句
# --语句块

## 混合顺序语句块和并行语句块Verilog HDL描述的例子

```
fork
@enable_a
begin
        #ta wa = 0;
        #ta wa = 1;
        #ta wa = 0;
end
```

```
@enable_b
begin
        #tb wb = 1;
        #tb wb = 0;
        #tb wb = 1;
end
join
```

# Verilog HDL行为描述语句
# --结构化的过程

在Verilog HDL中所有的过程由下面四种语句指定：

- initial结构
- always结构
- 任务
- 函数

# Verilog HDL行为描述语句
# --结构化的过程

## initial 语句

- initial语句只执行一次。
- 当用于仿真的时候，initial 语句在仿真开始时执行，Initial语句通常用于仿真模块对激励向量的描述。
- 也用于在综合实现过程中给寄存器变量赋初值。

# Verilog HDL行为描述语句
# --结构化的过程

**initial语句的语法格式如下：**

```
initial
    begin
        statement1;  //描述语句1
        statement2;  //描述语句2
            .....
end
```

# Verilog HDL行为描述语句
# --结构化的过程

顺序过程最常使用在进程语句中。这里的时序控制可以是：

- 时延控制，即等待一个确定的时间；
- 或事件控制，即等待确定的事件发生或某一特定的条件为真。
- initial语句的各个进程语句仅执行一次。

# Verilog HDL行为描述语句
# --结构化的过程

## initial语句的Verilog HDL描述的例子

```
initial begin
        areg = 0; // 初始化一个寄存器
for (index = 0; index < size; index = index + 1)
            memory[index] = 0; //初始化存储器字
end
```

# Verilog HDL行为描述语句
# --结构化的过程

## 带有时延控制的initial语句Verilog HDL描述的例子

```
initial begin
    inputs = ‘b000000; // 在0时刻初始化
    #10 inputs = ‘b011001; // 第一个模式
    #10 inputs = ‘b011011; // 第二个模式
    #10 inputs = ‘b011000; // 第三个模式
    #10 inputs = ‘b001000; // 第四个模式
end
```

# Verilog HDL行为描述语句
# --结构化的过程

## always语句

- 在用在仿真期间内，always结构连续的重复。always结构由于其循环本质，当和一些形式的时序控制一起使用时，它是有用的。
- 如果一个always结构没有控制达到仿真时间，将引起死锁条件。

# Verilog HDL行为描述语句
# --结构化的过程

带有零时延控制的always语句Verilog HDL描述的例子

```
always areg = ~areg;
```

带有时延控制的always语句Verilog HDL描述的例子

```
always #half_period areg = ~areg;
```