

# Verilog HDL

## 第四讲

### 3. 循环语句forever

- forever 块语句

- 永久循环语句
- 格式：forever 块语句
- 例：

```
module microprocessor ;  
  always  
    begin  
      power-on-initializations;  
      forever  
        begin  
          fetch-decoder-\  
            execution-of-instruction;  
        end  
      end  
    end  
endmodule
```

### 举例

- 用initial描述时钟产生

```
initial  
  begin  
    clock=0;  
    forever #50 clock=~clock;  
  end
```

注：一般习惯用always来描述时钟

### 循环语句repeat

- 格式：

repeat (循环次数计算表达式) 块语句

- 步骤：

- 计算循环次数，并保存
- 块语句执行一次
- 循环次数减一
- 结果为零，循环结束，否则继续循环

## 用移位加的方法实现乘法器

```
module multiplier(result,op_a,op_b);
  parameter size=8;
  input [size:1] op_a,op_b;
  output [2*size:1] result;
  reg [2*size:1] shift_opa,result;
  reg[size:1] shift_opb;
  always@(op_a or op_b)
  begin
    result=0;
    shift_opa=op_a;
    shift_opb=op_b;
    repeat(size)
    begin
      #100 if(shift_opb[1]) result=result+shift_opa;
      shift_opa=shift_opa<<1;
      shift_opb=shift_opb>>1;
    end
  end
endmodule

result=shift_opa x shift_opb
注意寄存器位宽定义
size应用,可参数化设计
```

## 循环语句while

- 格式：  
while (循环执行条件表达式) 块语句
- 步骤：
  - 判断条件表达式
  - 条件表达式为真，执行块语句；为假，跳出循环
  - 执行块语句结束后继续判断，直至跳出循环

## 举例：计算变量中1的个数

```
module demo_count(var,count);
  parameter var_size=8,cnt_size=4;
  input [var_size:0] var;
  output [cnt_size:0] count;
  reg [cnt_size:0] count;
  reg [var_size:0] tmp_var;
  always@(var)
  begin
    count=0;
    tmp_var=var;
    while(tmp_var)
    begin
      if(tmp_var[0]) count=count+1;
      tmp_var=tmp_var>>1;
    end
  end
endmodule
```

## 举例：另一种描述方式

```
module demo_count(var,count);
  parameter var_size=8,cnt_size=4;
  input [var_size:0] var;
  output [cnt_size:0] count;
  integer i;
  reg [cnt_size:0] count;
  always@(var)
  begin
    count=0;
    i=0;
    while(i<var_size)
    begin
      count=count+tmp_var[i];
      i=i+1;
    end
  end
endmodule

缺点：
```

## 循环语句for

### ■ 格式

- for ( 表达式1 ; 表达式2 ; 表达式3 ) 块语句
- for语句可以看作时while的一种简约表达形式

## 举例：综合性

```
module demo_decode(.....);
//port and data type declaration
always@(posedge clock)
begin
    casez(opcode)
    3'b1??: alu_out=acc;
    3'b000: while(bloc_wr_enable)
        repeat(5) @(posedge clock)
        begin
            ram[address]=data_bus;
            address=address+1;
        end
    3'b011: begin: load
        integer i;
        for(i=0; i<256; i=i+1)
            @(negedge clock) data_bus=ram[i];
        end
    default: $disp("Illegal opcode!\n ***fatal error***");
    endcase
end
endmodule
```

## 循环中断语句disable

- 中断正在执行的循环或任务
- 格式  
disable 块名或任务名
- 实际改变有名块的执行过程，只是与循环语句结合后，为循环进程的中断控制提供了实现的手段

## 举例

```
...
begin: top
for(i=0; i<n; i=i+1)
begin: inner
if(a=0)
disable inner;
...
if(a=b)
disable top;
...
end
end
...
```

## 8.Wait语句

- 格式  
wait (条件表达式) 块语句
- 相当定时控制中的时间控制，但电平触发
- 执行过程
  - 判断条件表达式所代表的信号电平
  - 高电平，则继续执行；低电平，则等待

## 举例：完整的收发系统

### 顶层模块

```
module produce_receiver;  
  Wire [7:0] data;  
  wire prod_ready,cons_ready;  
  producer   prod_bloc(data,pro_ready,cons_ready);  
  receiverrec_bloc(data,prod_ready,cons_ready);  
enmodule
```

## 数据接收模块

```
module receiver(data,prod_ready,con_ready);  
  input [7:0] data;  
  input prod_ready;  
  output cons_ready;  
  reg cons_ready;  
  reg [7:0] data_buf;  
  always  
  begin  
    cons_ready=1;  
    forever  
    begin  
      wait(prod_ready)  
      data_buf=data;  
      cons_ready=0;  
      ...  
      wait(!prod_ready)  
      cons_ready=1;  
    end  
  end  
endmodule
```

## 数据发送模块

```
module producer(data,prod_ready,con_ready);  
  output [7:0] data;  
  output prod_ready;  
  input cons_ready;  
  reg prod_ready;  
  reg [7:0] data_buf;  
  always  
  begin  
    prod_ready=0;  
    forever  
    begin  
      ...  
      wait(cons_ready)  
      data=new_produced_data;  
      prod_ready=1;  
      wait(!cons_ready)  
      prod_ready=0;  
    end  
  end  
endmodule
```

## 9.有名事件

- 格式
  - event 事件名列表
- 除连线类型和寄存器类型外的另一种特殊数据类型，无值的概念，只表明一个抽象的事件在某一特定时刻或特定条件下被触发。触发过程语句：  
- > 事件名；
- 事件触发状态可以通过@(事件名)的方式进行检测
- 主要用于模块间的通信握手

## 举例：计数并统计1的位数

```
module top ;  
    wire [15:0] number;  
    wire [3:0] count;  
    number_gen    ng(number);  
    bit_count     bc(number,count);  
endmodule
```

## 计数模块

```
module number_gen(number);  
    output [15:0] number;  
    reg [15:0]    number;  
    event        ready;  
    initial      number=0;  
    always  
        begin  
            #50    number=number+1;  
            #50    ->ready;  
        end  
endmodule
```

## 统计模块

```
module bit_count(number,count);  
    input [15:0]    number;  
    output [3:0]    count;  
    reg [3:0]       count;  
    reg [15:0]      num_buf;  
    integer         i;  
    always  
        begin  
            @top.ng.ready num_buf=number;  
            count=0;  
            for(i=0; i<16; i=i+1)  
                if(num_buf[i]) count=count+1;  
        end  
endmodule
```

## 五、任务与函数

### 1. 任务

- 格式：

```
task    任务名
    端口与类型说明
    局部变量说明
    块语句
endtask
```

## 说明

- 任务的定义和引用都在一个模块内部
- 任务定义与模块定义类似
- 任务内部没有过程语句，但可以包含定时控制部分
- 任务引用与模块调用一样，需通过任务名端口调用实现
- 任务可以调用别的任务和函数，也可以调用任务本身
- 可以在任务内用disable将任务中断

## 举例：用task实现译码器中的乘法器

```
module demo_dec_task;
    reg [15:0] mem[0:1023];
    reg [9:0] pc;
    reg [31:0] acc;
    reg [15:0] ir;
    initial pc=0;
    always
        begin
            ir=mem[pc];
            case (ir[15:13])
                ...
                3'b111: multiply(acc,mem[ir[12:0]]);
            endcase
            #100 pc=pc+1;
        end
end
```

## 续

```
task multiply(op_a,op_b);
    inout [32:1] op_a;
    input [16:1] op_b;
    reg [32:1] shift_opa,result,op_a;
    reg [16:1] shift_opb;
    begin
        result=0;
        shift_opa=op_a;
        shift_opb=op_b;
        repeat(16)
            begin
                #100 if(shift_opb[1])
                    result=result+shift_opa;
                shift_opa=shift_opa<<1;
                shift_opb=shfit_opb>>1;
            end
        end
        op_a=result;
    end
endtask
endmodule
```

## disable task 举例

```
module tmp;
...
task demo_procedure;
...
begin
...
if(a==0) disable demo_procedure;
...
end
endtask
endmodule
```

## 2. 函数

### ■ 格式

```
function <位宽说明> 函数名 ;
    输入端口与类型说明 ;
    局部变量说明 ;
    块语句
endfunction
```

## 说明

- 与任务一样，位于模块内部
- 函数不能调用任务，而任务可以调用函数
- 任务可以无输入变量和I/O变量，而函数只能有输入变量，且至少有一个
- 函数名就是输出变量名，通过函数名返回值，而任务通过I/O端口实现传值
- 函数中不能出现任何类型的定时控制描述，也不允许有wait语句和disable语句，而任务则可以
- 函数可以出现在连续赋值语句的右端表达式中

## 举例：用function实现译码器中的乘法器

```
module demo_dec_fun;
reg [15:0] mem[0:1023];
reg [9:0] pc;
reg [31:0] acc;
reg [15:0] ir;
initial pc=0;
always
begin
ir=mem[pc];
case (ir[15:13])
...
3'b111: acc=multiply(acc,mem[ir[12:0]]);
endcase
#100 pc=pc+1;
end
```

续

```
function [32:1] multiply(op_a,op_b);
input [32:1] op_a;
input [16:1] op_b;
reg [32:1] shift_opa,result,op_a;
reg [16:1] shift_opb;
begin
    multiply=0;
    shift_opa=op_a;
    shift_opb=op_b;
    repeat(16)
        begin
            if(shift_opb[1])
                multiply=multiply+shift_opa;
            shift_opa=shift_opa<<1;
            shift_opb=shift_opb>>1;
        end
    end
endfunction
endmodule
```

## 第六章 Verilog结构描述

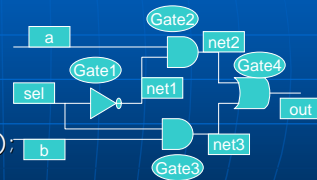
- 侧重反映模块内部的结构组成
- 门级描述：
  - 对由基本逻辑门级元件互连而成的具有一定功能的电路模块
- 开关级描述
  - 是构成VerilogHDL对硬件设计最低层次的描述
  - 通常的综合工具不支持开关级描述

## 结构描述方法

- 将电路形式的连接关系转换成对应的基本逻辑门的描述
- 步骤：
  - 给电路图中的输入输出引脚赋以端口名
  - 给电路图中的每条连线信号定义名称
  - 给电路图中的每个逻辑单元定义单元名（即调用名）
  - 给这个电路模块定义一个模块名
  - 用module定义相应模块名的结构描述，并将所有输入输出端口名列入端口名表项中，再对各端口类型说明
  - 连线类型说明，并按电路连接关系，确定各单元间各端口的信号连接，完成对电路内部的结构描述
  - endmodule结束

## 举例：MUX

- 结构描述
- ```
module mux_str(out,a,b,sel);
output out;
Input a,b,sel;
not gate1(net1,sel);
and gate2(net2,a,net1);
and gate3(net3,b,sel);
or gate4(out,net2,net3);
endmodule
```



not、and、or为VerilogHDL的基本门级元件，gatex为单元电路（门）名称，netx为电路内部连线名称。