

**【例 3.1】4 位全加器**

```

module adder4(cout,sum,ina,inb,cin);
output [3:0] sum;
output cout;
input [3:0] ina,inb;
input cin;
assign {cout,sum}=ina+inb+cin;
endmodule

```

**【例 3.2】4 位计数器**

```

module count4(out,reset,clk);
output [3:0] out;
input reset,clk;
reg [3:0] out;
always @(posedge clk)
begin
if (reset) out<=0; // 同步复位
else out<=out+1; // 计数
end
endmodule

```

**【例 3.3】4 位全加器的仿真程序**

```

`timescale 1ns/1ns
`include "adder4.v"
module adder_tp; // 测试模块的名字
reg [3:0] a,b; // 测试输入信号定义为 reg 型
reg cin;
wire [3:0] sum; // 测试输出信号定义为 wire 型
wire cout;
integer i,j;

adder4 adder(sum,cout,a,b,cin); // 调用测试对象
always #5 cin=~cin; // 设定 cin 的取值

initial
begin
a=0;b=0;cin=0;
for (i=1;i<16;i=i+1)
#10 a=i; // 设定 a 的取值
end

```

```

initial
begin
for (j=1;j<16;j=j+1)
#10 b=j;           // 设定 b 的取值
end

initial           // 定义结果显示格式
begin
$monitor($time,,,"%d + %d + %b={%b,%d}",a,b,cin,cout,sum);
#160 $finish;
end
endmodule

```

### 【例 3.4】4 位计数器的仿真程序

```

`timescale 1ns/1ns
`include " count4.v "
module coun4_tp;
reg clk,reset;           // 测试输入信号定义为 reg 型
wire [3:0] out;          // 测试输出信号定义为 wire 型
parameter DELY=100;

count4 mycount(out,reset,clk);           // 调用测试对象

always #(DELY/2) clk = ~clk;             // 产生时钟波形
initial
begin                                     // 激励信号定义
clk =0; reset=0;
#DELY reset=1;
#DELY reset=0;
#(DELY*20) $finish;
end
// 定义结果显示格式
initial $monitor($time,,,"clk=%d reset=%d out=%d", clk, reset,out);
endmodule

```

### 【例 3.5】“与-或-非”门电路

```

module AOI(A,B,C,D,F);           // 模块名为 AOI( 端口列表 A , B , C , D , F)
input A,B,C,D;                   // 模块的输入端口为 A , B , C , D
output F;                        // 模块的输出端口为 F

```

```
wire  A,B,C,D,F;           // 定义信号的数据类型
    assign  F= ~(A&B)|(C&D)); // 逻辑功能描述
endmodule
```

**【例 5.1】用 case 语句描述的 4 选 1 数据选择器**

```
module  mux4_1(out,in0,in1,in2,in3,sel);
output  out;
input   in0,in1,in2,in3;
input   [1:0] sel;
reg  out;
always  @(in0   or in1   or in2   or in3   or sel) // 敏感信号列表
    case (sel)
        2'b00:    out=in0;
        2'b01:    out=in1;
        2'b10:    out=in2;
        2'b11:    out=in3;
        default   : out=2'bx;
    endcase
endmodule
```

**【例 5.2】同步置数、同步清零的计数器**

```
module  count(out,data,load,reset,clk);
output  [7:0] out;
input   [7:0] data;
input   load,clk,reset;
reg  [7:0] out;
always  @( posedge  clk)           //clk  上升沿触发
    begin
        if  (!reset)               out = 8'h00;           // 同步清 0 , 低电平有效
        else  if  (load)            out = data;           // 同步预置
        else                               out = out + 1;   // 计数
    end
endmodule
```

**【例 5.3】用 always 过程语句描述的简单算术逻辑单元**

```
`define  add 3'd0
`define  minus 3'd1
`define  band 3'd2
`define  bor 3'd3
`define  bnot 3'd4
```

```

module alu(out,opcode,a,b);
output [7:0] out;
reg [7:0] out;
input [2:0] opcode; // 操作码
input [7:0] a,b; // 操作数
always @(opcode or a or b) // 电平敏感的 always 块
begin
    case (opcode)
        `add: out = a+b; // 加操作
        `minus: out = a-b; // 减操作
        `band: out = a&b; // 求与
        `bor: out = a|b; // 求或
        `bnot: out=~a; // 求反
        default : out=8'hx; // 未收到指令时，输出任意态
    endcase
end
endmodule

```

**【例 5.4】用 initial 过程语句对测试变量 A、B、C 赋值**

```

`timescale 1ns/1ns
module test;
reg A,B,C;
initial
begin
    A = 0;    B = 1; C = 0;

    #50      A = 1;    B = 0;
    #50      A = 0;    C = 1;
    #50      B = 1;
    #50      B = 0;    C = 0;
    #50      $finish ;
end
endmodule

```

**【例 5.5】用 begin-end 串行块产生信号波形**

```

`timescale 10ns/1ns
module wave1;
reg wave;
parameter cycle=10;
initial
begin

```

```

        wave=0;
    #(cycle/2) wave=1;
    #(cycle/2) wave=0;
    #(cycle/2) wave=1;
    #(cycle/2) wave=0;
    #(cycle/2) wave=1;
    #(cycle/2) $finish ;
end
initial    $monitor($time,, "wave=%b",wave);
endmodule

```



**【例 5.6】用 fork-join 并行块产生信号波形**

```

`timescale    10ns/1ns
module wave2;
reg wave;
parameter    cycle=5;
initial
    fork
        wave=0;
        #(cycle)    wave=1;
        #(2*cycle)    wave=0;
        #(3*cycle)    wave=1;
        #(4*cycle)    wave=0;
        #(5*cycle)    wave=1;
        #(6*cycle)    $finish;
    join
initial    $monitor($time,, "wave=%b",wave);
endmodule

```

**【例 5.7】持续赋值方式定义的 2 选 1 多路选择器**

```

module MUX21_1(out,a,b,sel);
input  a,b,sel;
output out;
assign out=(sel==0)?a:b;
        // 持续赋值，如果 sel 为 0，则 out=a；否则 out=b
endmodule

```

**【例 5.8】阻塞赋值方式定义的 2 选 1 多路选择器**

```

module MUX21_2(out,a,b,sel);
input  a,b,sel;

```

```

output out;
reg out;
always @(a or b or sel)
begin
    if (sel==0) out=a;           // 阻塞赋值
    else out=b;
end
endmodule

```

**【例 5.9】非阻塞赋值**

```

module non_block(c,b,a,clk);
output c,b;
input clk,a;
reg c,b;
always @(posedge clk)
begin
    b<=a;
    c<=b;
end
endmodule

```

**【例 5.10】阻塞赋值**

```

module block(c,b,a,clk);
output c,b;
input clk,a;
reg c,b;
always @(posedge clk)
begin
    b=a;
    c=b;
end
endmodule

```

**【例 5.11】模为 60 的 BCD 码加法计数器**

```

module count60(qout,cout,data,load,cin,reset,clk);
output [7:0] qout;
output cout;
input [7:0] data;
input load,cin,clk,reset;
reg [7:0] qout;
always @(posedge clk)           //clk 上升沿时刻计数

```

```

begin
    if (reset)          qout<=0;           // 同步复位
    else if (load)       qout<=data;       // 同步置数
    else if (cin)
        begin
            if (qout[3:0]==9)              // 低位是否为 9，是则
                begin
                    qout[3:0]<=0;           // 回 0，并判断高位是否为 5
                    if (qout[7:4]==5) qout[7:4]<=0;
                    else
                        qout[7:4]<=qout[7:4]+1; // 高位不为 5，则加 1
                    end
                else
                    // 低位不为 9，则加 1
                    qout[3:0]<=qout[3:0]+1;
                end
        end
    end
assign cout=((qout==8'h59)&cin)?1:0;      // 产生进位输出信号
endmodule

```

### 【例 5.12】BCD 码—七段数码管显示译码器

```

module decode4_7(decodeout,indec);
output [6:0] decodeout;
input [3:0] indec;
reg [6:0] decodeout;
always @(indec)
begin
    case (indec) // 用 case 语句进行译码
        4'd0:decodeout=7'b1111110;
        4'd1:decodeout=7'b0110000;
        4'd2:decodeout=7'b1101101;
        4'd3:decodeout=7'b1111001;
        4'd4:decodeout=7'b0110011;
        4'd5:decodeout=7'b1011011;
        4'd6:decodeout=7'b1011111;
        4'd7:decodeout=7'b1110000;
        4'd8:decodeout=7'b1111111;
        4'd9:decodeout=7'b1111011;
        default : decodeout=7'bx;
    endcase
end

```

```
endmodule
```

**【例 5.13】用 casez 描述的数据选择器**

```
module mux_casez(out,a,b,c,d,select);
output out;
input a,b,c,d;
input [3:0] select;
reg out;
always @(select or a or b or c or d)
begin
    casez (select)
        4'b???1: out = a;
        4'b??1?: out = b;
        4'b?1??: out = c;
        4'b1???: out = d;
    endcase
end
endmodule
```

**【例 5.14】隐含锁存器举例**

```
module buried_ff(c,b,a);
output c;
input b,a;
reg c;
always @(a or b)
begin
    if ((b==1)&&(a==1)) c=a&b;
end
endmodule
```

**【例 5.15】用 for 语句描述的七人投票表决器**

```
module voter7(pass,vote);
output pass;
input [6:0] vote;
reg [2:0] sum;
integer i;
reg pass;
always @(vote)
begin
    sum=0;
```



```

        for (i=0;i<=6;i=i+1)                //for 语句
            if (vote[i]) sum=sum+1;
            if (sum[2])    pass=1;           // 若超过 4 人赞成，则 pass=1
            else          pass=0;
        end
    endmodule

```

**【例 5.16】用 for 语句实现 2 个 8 位数相乘**

```

module mult_for(outcome,a,b);
parameter    size=8;
input  [size:1] a,b;                // 两个操作数
output [2*size:1] outcome;         // 结果
reg [2*size:1] outcome;
integer    i;

always @(a or b)
begin
    outcome=0;
    for (i=1; i<=size; i=i+1)        //for 语句
        if (b[i]) outcome=outcome +(a << (i-1));
    end
endmodule

```

**【例 5.17】用 repeat 实现 8 位二进制数的乘法**

```

module mult_repeat(outcome,a,b);
parameter    size=8;
input  [size:1] a,b;
output [2*size:1] outcome;
reg [2*size:1] temp_a,outcome;
reg [size:1] temp_b;
always @(a or b)
begin
    outcome=0;
    temp_a=a;
    temp_b=b;
    repeat (size)                    //repeat 语句，size 为循环次数
    begin
        if (temp_b[1])                // 如果 temp_b 的最低位为 1，就执行下面的加法
            outcome=outcome+temp_a;
        temp_a=temp_a<<1;             // 操作数 a 左移一位
    end
end

```

```

        temp_b=temp_b>>1;           // 操作数 b 右移一位
    end

end

endmodule

```

### 【例 5.18】同一循环的不同实现方式

```

module loop1;                       // 方式 1
integer i;
initial
    for (i=0;i<4;i=i+1)             //for 语句
    begin
        $display( " i=%h" ,i);
    end
endmodule

module loop2;                       // 方式 2
integer i;
initial begin
    i=0;
    while (i<4)                     //while 语句
    begin
        $display ("i=%h",i);
        i=i+1;
    end
end
endmodule

module loop3;                       // 方式 3
integer i;
initial begin
    i=0;
    repeat (4)                      //repeat 语句
    begin
        $display ("i=%h",i);
        i=i+1;
    end
end
endmodule

```

### 【例 5.19】使用了 `include 语句的 16 位加法器

```

`include    "adder.v"

module  adder16(cout,sum,a,b,cin);
output  cout;
parameter  my_size=16;
output  [my_size-1:0] sum;
input  [my_size-1:0] a,b;
input  cin;

adder my_adder(cout,sum,a,b,cin);           // 调用 adder 模块
endmodule
// 下面是 adder 模块代码
module  adder(cout,sum,a,b,cin);
parameter  size=16;
output  cout;
output  [size-1:0] sum;
input  cin;
input  [size-1:0] a,b;
    assign  {cout,sum}=a+b+cin;
endmodule

```

#### 【例 5.20】条件编译举例

```

module  compile(out,A,B);
output  out;
input  A,B;
`ifdef    add                               // 宏名为 add
    assign  out=A+B;
`else
    assign  out=A-B;
`endif
endmodule

```

#### 【例 6.1】加法计数器中的进程

```

module  count(data,clk,reset,load,cout,qout);
output  cout;
output  [3:0] qout;
reg  [3:0] qout;
input  [3:0] data;
input  clk,reset,load;

```

```

always @(posedge clk) // 进程 1 , always 过程块
begin
    if (!reset)         qout= 4'h00; // 同步清 0 , 低电平有效
    else if (load)      qout= data;  // 同步预置
    else                qout=qout + 1; // 加法计数
end

assign cout=(qout==4'hf)?1:0; // 进程 2 , 用持续赋值产生进位信号
endmodule

```

### 【例 6.2】任务举例

```

module alutask(code,a,b,c);
input  [1:0] code;
input  [3:0] a,b;
output [4:0] c;
reg [4:0] c;

task my_and; // 任务定义 , 注意无端口列表
input  [3:0] a,b; //a,b,out 名称的作用域范围为 task 任务内部
output [4:0] out;
integer i;
begin
    for (i=3;i>=0;i=i-1)
        out[i]=a[i]&b[i]; // 按位与
    end
endtask

always @(code or a or b)
begin
    case (code)
        2'b00: my_and(a,b,c);
        /* 调用任务 my_and , 需注意端口列表的顺序应与任务定义中的一致 , 这里的 a,b,c
        分别对应任务定义中的 a,b,out */
        2'b01: c=a|b; // 或
        2'b10: c=a-b; // 相减
        2'b11: c=a+b; // 相加
    endcase
end
endmodule

```

**【例 6.3】测试程序**

```

`include    "alutask.v"
module  alu_tp;
reg  [3:0] a,b;
reg  [1:0] code;
wire  [4:0] c;
parameter  DELY = 100;

alu task ADD(code,a,b,c);                // 调用被测试模块

initial begin
        code=4'd0; a= 4'b0000; b= 4'b1111;
#DELY    code=4'd0; a= 4'b0111; b= 4'b1101;
#DELY    code=4'd1; a= 4'b0001; b= 4'b0011;
#DELY    code=4'd2; a= 4'b1001; b= 4'b0011;
#DELY    code=4'd3; a= 4'b0011; b= 4'b0001;
#DELY    code=4'd3; a= 4'b0111; b= 4'b1001;
#DELY    $finish;
end
initial    $monitor($time,, "code=%b a=%b b=%b c=%b", code,a,b,c);
endmodule

```

**【例 6.4】函数**

```

function    [7:0] get0;
input  [7:0] x;
reg  [7:0] count;
integer    i;
    begin
        count=0;
        for  (i=0;i<=7;i=i+1)
            if  (x[i]=1'b0) count=count+1;
        get0=count;
    end
endfunction

```

**【例 6.5】用函数和 case 语句描述的编码器（不含优先顺序）**

```

module  code_83(din,dout);
input  [7:0] din;
output  [2:0] dout;

```

```

function    [2:0] code;           // 函数定义
input  [7:0] din;                 // 函数只有输入，输出为函数名本身
    casex  (din)
        8'b1xxx_xxxx : code = 3'h7;
        8'b01xx_xxxx : code = 3'h6;
        8'b001x_xxxx : code = 3'h5;
        8'b0001_xxxx : code = 3'h4;
        8'b0000_1xxx : code = 3'h3;
        8'b0000_01xx : code = 3'h2;
        8'b0000_001x : code = 3'h1;
        8'b0000_000x : code = 3'h0;
        default      : code = 3'hx;
    endcase
endfunction

assign  dout = code(din);         // 函数调用
endmodule

```

### 【例 6.6】阶乘运算函数

```

module  funct(clk,n,result,reset);
output  [31:0] result;
input   [3:0] n;
input   reset,clk;
reg [31:0] result;
always  @( posedge  clk)         // 在 clk  的上升沿时执行运算
    begin
        if (!reset) result<=0;   // 复位
        else begin
            result <= 2 * factorial(n); // 调用 factorial      函数
        end
    end

end

function  [31:0] factorial;       // 阶乘运算函数定义（注意无端口列表）
input  [3:0] opa;                 // 函数只能定义输入端，输出端口为函数名本身
reg [3:0] i;
    begin
        factorial = opa ? 1 : 0;
        for  (i= 2; i <= opa; i = i+1) // 该句若要综合通过，opa 应赋具体的数值
            factorial = i* factorial; // 阶乘运算
    end
endfunction

```

```
endfunction
endmodule
```

### 【例 6.7】测试程序

```
`define    clk_cycle 50
`include    "funct.v"
module  funct_tp;
reg  [3:0] n;
reg  reset,clk;
wire  [31:0] result;

    initial                                // 定义激励向量
    begin
        n=0; reset=1; clk=0;
        for (n=0;n<=15;n=n+1)
            #100 n=n;
        end

initial      $monitor($time,,,"n=%d result=%d",n,result);

always  #`clk_cycle  clk=~clk;           // 定义输出显示格式
                                              产生时钟信号

funct funct_try(.clk(clk),.n(n),.result(result),.reset(reset));

                                              // 调用被测试模块

endmodule
```

### 【例 6.8】顺序执行模块 1

```
module  serial1(q,a,clk);
output  q,a;
input  clk;
reg  q,a;
always  @( posedge  clk)
    begin
        q=~q;
        a=~q;
    end
endmodule
```

### 【例 6.9】顺序执行模块 2

```
module  serial2(q,a,clk);
output  q,a;
```

```
input  clk;
reg  q,a;
always  @( posedge  clk)
    begin
        a=~q;
        q=~q;
    end
endmodule
```

#### 【例 6.10】并行执行模块 1

```
module  paral1(q,a,clk);
output  q,a;
input  clk;
reg  q,a;
always  @( posedge  clk)
    begin
        q=~q;
    end
always  @( posedge  clk)
    begin
        a=~q;
    end
endmodule
```

#### 【例 6.11】并行执行模块 2

```
module  paral2(q,a,clk);
output  q,a;
input  clk;
reg  q,a;
always  @( posedge  clk)
    begin
        a=~q;
    end
always  @( posedge  clk)
    begin
        q=~q;
    end
endmodule
```

#### 【例 7.1】调用门元件实现的 4 选 1 MUX



```

module mux4_1a(out,in1,in2,in3,in4,cntrl1,cntrl2);
output out;
input in1,in2,in3,in4,cntrl1,cntrl2;
wire notcntrl1,notcntrl2,w,x,y,z;
not (notcntrl1,cntrl2),
    (notcntrl2,cntrl2);
and (w,in1,notcntrl1,notcntrl2),
    (x,in2,notcntrl1,cntrl2),
    (y,in3,cntrl1,notcntrl2),
    (z,in4,cntrl1,cntrl2);
or (out,w,x,y,z);
endmodule

```

**【例 7.2】用 case 语句描述的 4 选 1 MUX**

```

module mux4_1b(out,in1,in2,in3,in4,cntrl1,cntrl2);
output out;
input in1,in2,in3,in4,cntrl1,cntrl2;
reg out;
always @(in1 or in2 or in3 or in4 or cntrl1 or cntrl2)
    case ({cntrl1,cntrl2})
        2'b00:out=in1;
        2'b01:out=in2;
        2'b10:out=in3;
        2'b11:out=in4;
        default :out=2'bx;
    endcase
endmodule

```

**【例 7.3】行为描述方式实现的 4 位计数器**

```

module count4(clk,clr,out);
input clk,clr;
output [3:0] out;
reg [3:0] out;
always @(posedge clk or posedge clr)
    begin
        if (clr) out<=0;
        else out<=out+1;
    end
endmodule

```

**【例 7.4】数据流方式描述的 4 选 1 MUX**

```

module mux4_1c(out,in1,in2,in3,in4,cntrl1,cntrl2);
output out;
input in1,in2,in3,in4,cntrl1,cntrl2;
assign out=(in1 & ~cntrl1 & ~cntrl2)|(in2 & ~cntrl1 & cntrl2)|
           (in3 & cntrl1 & ~cntrl2)|(in4 & cntrl1 & cntrl2);
endmodule

```

**【例 7.5】用条件运算符描述的 4 选 1 MUX**

```

module mux4_1d(out,in1,in2,in3,in4,cntrl1,cntrl2);
output out;
input in1,in2,in3,in4,cntrl1,cntrl2;
assign out=cntrl1 ? (cntrl2 ? in4:in3):(cntrl2 ? in2:in1);
endmodule

```

**【例 7.6】门级结构描述的 2 选 1 MUX**

```

module mux2_1a(out,a,b,sel);
output out;
input a,b,sel;
not (sel_,sel);
and (a1,a,sel_),
    (a2,b,sel);
or (out,a1,a2);
endmodule

```

**【例 7.7】行为描述的 2 选 1 MUX**

```

module mux2_1b(out,a,b,sel);
output out;
input a,b,sel;
reg out;
always @(a or b or sel)
begin
    if (sel) out = b;
    else out = a;
end
endmodule

```

**【例 7.8】数据流描述的 2 选 1 MUX**

```

module MUX2_1c(out,a,b,sel);
output out;

```

```

input  a,b,sel;
      assign  out = sel ? b : a;
endmodule

```

**【例 7.9】调用门元件实现的 1 位半加器**

```

module half_add1(a,b,sum,cout);
input  a,b;
output sum,cout;
and  (cout,a,b);
xor  (sum,a,b);
endmodule

```

**【例 7.10】数据流方式描述的 1 位半加器**

```

module half_add2(a,b,sum,cout);
input  a,b;
output sum,cout;
assign sum=a^b;
assign cout=a&b;
endmodule

```

**【例 7.11】采用行为描述的 1 位半加器**

```

module half_add3(a,b,sum,cout);
input  a,b;
output sum,cout;
reg  sum,cout;
always @(a or b)
begin
    case ({a,b}) // 真值表描述
        2'b00:  begin  sum=0; cout=0;  end
        2'b01:  begin  sum=1; cout=0;  end
        2'b10:  begin  sum=1; cout=0;  end
        2'b11:  begin  sum=0; cout=1;  end
    endcase
end
endmodule

```

**【例 7.12】采用行为描述的 1 位半加器**

```

module half_add4(a,b,sum,cout);
input  a,b;
output sum,cout;

```

```

reg sum,cout;
always @(a or b)
    begin
        sum= a^b;
        cout=a&b;
    end
endmodule

```

**【例 7.13】调用门元件实现的 1 位全加器**

```

module full_add1(a,b,cin,sum,cout);
input  a,b,cin;
output sum,cout;
wire  s1,m1,m2,m3;
and  (m1,a,b),
     (m2,b,cin),
     (m3,a,cin);
xor  (s1,a,b),
     (sum,s1,cin);
or   (cout,m1,m2,m3);
endmodule

```

**【例 7.14】数据流描述的 1 位全加器**

```

module full_add2(a,b,cin,sum,cout);
input  a,b,cin;
output sum,cout;
assign sum = a ^ b ^ cin;
assign cout = (a & b)|(b & cin)|(cin & a);
endmodule

```

**【例 7.15】1 位全加器**

```

module full_add3(a,b,cin,sum,cout);
input  a,b,cin;
output sum,cout;
assign {cout,sum}=a+b+cin;
endmodule

```

**【例 7.16】行为描述的 1 位全加器**

```

module full_add4(a,b,cin,sum,cout);
input  a,b,cin;
output sum,cout;

```

```

reg sum,cout;                                // 在 always 块中被赋值的变量应定义为 reg 型
reg m1,m2,m3;
always @(a or b or cin)
begin
    sum = (a ^ b) ^ cin;
    m1 = a & b;
    m2 = b & cin;
    m3 = a & cin;
    cout = (m1|m2)|m3;
end
endmodule

```

### 【例 7.17】混合描述的 1 位全加器

```

module full_add5(a,b,cin,sum,cout);
input  a,b,cin;
output sum,cout;
reg cout,m1,m2,m3;                            // 在 always 块中被赋值的变量应定义为 reg 型
wire s1;
xor x1(s1,a,b);                                // 调用门元件
always @(a or b or cin)                        //always 块语句
begin
    m1 = a & b;
    m2 = b & cin;
    m3 = a & cin;
    cout = (m1| m2) | m3;
end
assign sum = s1 ^ cin;                          //assign 持续赋值语句
endmodule

```

### 【例 7.18】结构描述的 4 位级连全加器

```

`include "full_add1.v"
module add4_1(sum,cout,a,b,cin);
output [3:0] sum;
output  cout;
input  [3:0] a,b;
input  cin;

full_add1 f0(a[0],b[0],cin,sum[0],cin1);        // 级连描述
full_add1 f1(a[1],b[1],cin1,sum[1],cin2);
full_add1 f2(a[2],b[2],cin2,sum[2],cin3);

```

```
full_add1 f3(a[3],b[3],cin3,sum[3],cout);
```

```
endmodule
```

### 【例 7.19】数据流描述的 4 位全加器

```
module add4_2(cout,sum,a,b,cin);
```

```
output [3:0] sum;
```

```
output cout;
```

```
input [3:0] a,b;
```

```
input cin;
```

```
assign {cout,sum}=a+b+cin;
```

```
endmodule
```

### 【例 7.20】行为描述的 4 位全加器

```
module add4_3(cout,sum,a,b,cin);
```

```
output [3:0] sum;
```

```
output cout;
```

```
input [3:0] a,b;
```

```
input cin;
```

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

```
reg cout;
```

```
always @(a or b or cin)
```

```
begin
```

```
    {cout,sum}=a+b+cin;
```

```
end
```

```
endmodule
```

### 【例 8.1】\$time 与 \$realtime 的区别

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

```
module time_dif;
```

```
reg ts;
```

```
parameter delay=2.6;
```

```
initial
```

```
begin
```

```
    #delay ts=1;
```

```
    #delay ts=0;
```

```
    #delay ts=1;
```

```
    #delay ts=0;
```

```
end
```

```
initial $monitor($time,,,"ts=%b",ts);
```

```
// 使用函数 $time
```

```
endmodule
```

### 【例 8.2】\$random 函数的使用

```
`timescale 10ns/1ns
module random_tp;
integer data;
integer i;
parameter delay=10;

initial $monitor($time,,"data=%b",data);
initial begin
    for (i=0; i<=100; i=i+1)
        #delay data=$random;           // 每次产生一个随机数
    end
endmodule
```

### 【例 8.3】1 位全加器进位输出 UDP 元件

```
primitive carry_udp(cout,cin,a,b);
input cin,a,b;
output cout;

table
    //cin a b : cout           // 真值表
    0 0 0 : 0;
    0 1 0 : 0;
    0 0 1 : 0;
    0 1 1 : 1;
    1 0 0 : 0;
    1 0 1 : 1;
    1 1 0 : 1;
    1 1 1 : 1;
endtable
endprimitive
```

### 【例 8.4】包含 x 态输入的 1 位全加器进位输出 UDP 元件

```
primitive carry_udpx1(cout,cin,a,b);
input cin,a,b;
output cout;

table
    // cin a b : cout //           真值表
    0 0 0 : 0;
```

```

        0 1 0 : 0;
        0 0 1 : 0;
        0 1 1 : 1;
        1 0 0 : 0;
        1 0 1 : 1;
        1 1 0 : 1;
        1 1 1 : 1;
        0 0 x : 0; // 只要有两个输入为 0 , 则进位输出肯定为 0
        0 x 0 : 0;
        x 0 0 : 0;
        1 1 x : 1; // 只要有两个输入为 1 , 则进位输出肯定为 1
        1 x 1 : 1;
        x 1 1 : 1;

    endtable
endprimitive
```

【例 8.5】用简缩符“？”表述的 1 位全加器进位输出 UDP 元件

```

primitive    carry_udpx2(cout,cin,a,b);
input  cin,a,b;
output  cout;

    table
        // cin a b : cout                                // 真值表
        ? 0 0 : 0;                                         // 只要有两个输入为 0 , 则进位输出肯定为 0
        0 ? 0 : 0;
        0 0 ? : 0;
        ? 1 1 : 1;                                         // 只要有两个输入为 1 , 则进位输出肯定为 1
        1 ? 1 : 1;
        1 1 ? : 1;

    endtable
endprimitive
```

【例 8.6】3 选 1 多路选择器 UDP 元件

```

primitive    mux31(Y,in0,in1,in2,s2,s1);
input  in0,in1,in2,s2,s1;
output  Y;

    table
        //in0 in1 in2 s2 s1 : Y
        0 ? ? 0 0 : 0;                                     // 当 s2s1=00 时 , Y=in0
        1 ? ? 0 0 : 1;
        ? 0 ? 0 1 : 0;                                     // 当 s2s1=01 时 , Y=in1
    endtable
endprimitive
```



```

? 1 ? 0 1 : 1;
? ? 0 1 ? : 0;           // 当 s2s1=1 ? 时, Y=in2
? ? 1 1 ? : 1;
0 0 ? 0 ? : 0;
1 1 ? 0 ? : 1;
0 ? 0 ? 0 : 0;
1 ? 1 ? 0 : 1;
? 0 0 ? 1 : 0;
? 1 1 ? 1 : 1;
endtable
endprimitive

```

### 【例 8.7】电平敏感的 1 位数据锁存器 UDP 元件

```

primitive    latch(Q,clk,reset,D);
input  clk,reset,D;
output  Q;
reg  Q;
initial    Q = 1'b1; //      初始化
    table
        // clk reset D : state : Q
        ? 1 ? : ? : 0; //reset=1           , 则不管其他端口为什么值, 输出都为 0
        0 0 0 : ? : 0; //clk=0             , 锁存器把 D 端的输入值输出
        0 0 1 : ? : 1;
        1 0 ? : ? : -; //clk=1             , 锁存器的输出保持原值, 用符号 “ - ” 表示
    endtable
endprimitive

```

### 【例 8.8】上升沿触发的 D 触发器 UDP 元件

```

primitive    DFF(Q,D,clk);
output  Q;
input  D,clk;
reg  Q;
    table
        //clk D : state : Q
        (01) 0 : ? : 0;           // 上升沿到来, 输出 Q=D
        (01) 1 : ? : 1;
        (0x) 1 : 1 : 1;
        (0x) 0 : 0 : 0;
        (?0) ? : ? : -;           // 没有上升沿到来, 输出 Q 保持原值
        ? (??) : ? : -;           // 时钟不变, 输出也不变
    endtable
endprimitive

```

```
endtable
endprimitive
```

【例 8.9】带异步置 1 和异步清零的上升沿触发的 D 触发器 UDP 元件

```
primitive DFF_UDP(Q,D,clk,clr,set);
output Q;
input D,clk,clr,set;
reg Q;

table
// clk D      clr  set  : state : Q
(01)  1    0    0    : ? :    0;
(01)  1    0    x    : ? :    0;
?     ?    0    x    : 0 :    0;
(01)  0    0    0    : ? :    1;
(01)  0    x    0    : ? :    1;
?     ?    x    0    : 1 :    1;
(x1)  1    0    0    : 0 :    0;
(x1)  0    0    0    : 1 :    1;
(0x)  1    0    0    : 0 :    0;
(0x)  0    0    0    : 1 :    1;
?     ?    1    ?    : ? :    1;           // 异步复位
?     ?    0    1    : ? :    0;           // 异步置 1
n     ?    0    0    : ? :    -;
?     *    ?    ?    : ? :    -;
?     ?    (?0) ? : ? :    -;
?     ?    ?    (?0): ? :    -;
?     ?    ?    ?    : ? :    x;

endtable
endprimitive
```

【例 8.12】延迟定义块举例

```
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
```

### 【例 8.13】激励波形的描述

```
'timescale 1ns/1ns
module test1;
reg A,B,C;
initial
begin                                // 激励波形描述
    A = 0; B = 1; C = 0;
    #100 C = 1;
    #100 A = 1; B = 0;
    #100 A = 0;
    #100 C = 0;
    #100 $finish;
end
initial $monitor($time,,"A=%d B=%d C=%d",A,B,C); // 显示
endmodule
```

### 【例 8.15】用 always 过程块产生两个时钟信号

```
module test2;
reg clk1,clk2;
parameter CYCLE = 100;
always
begin
    {clk1,clk2} = 2'b10;
    #(CYCLE/4) {clk1,clk2} = 2'b01;
    #(CYCLE/4) {clk1,clk2} = 2'b11;
    #(CYCLE/4) {clk1,clk2} = 2'b00;
    #(CYCLE/4) {clk1,clk2} = 2'b10;
end
initial $monitor($time,,"clk1=%b clk2=%b",clk1,clk2);
endmodule
```

### 【例 8.17】存储器在仿真程序中的应用

```
module ROM(addr,data,oe);
output [7:0] data;                // 数据信号
input [14:0] addr;                // 地址信号
input oe;                         // 读使能信号，低电平有效
```

```

reg [7:0] mem[0:255];                // 存储器定义
parameter    DELAY = 100;
assign    #DELAY data=(oe==0) ? mem[addr] : 8'hzz;

initial      $readmemh("rom.hex",mem);        // 从文件中读入数据
endmodule

```

### 【例 8.18】8 位乘法器的仿真程序

```

`timescale    10ns/1ns
module  mult_tp;                // 测试模块的名字
reg [7:0] a,b;                  // 测试输入信号定义为 reg 型
wire  [15:0] out;               // 测试输出信号定义为 wire 型
integer    i,j;

mult8 m1(out,a,b);              // 调用测试对象
                                // 激励波形设定

initial
    begin
        a=0;b=0;
        for (i=1;i<255;i=i+1)
            #10 a=i;
        end
initial
    begin
        for (j=1;j<255;j=j+1)
            #10 b=j;
        end

initial                          // 定义结果显示格式
    begin
        $monitor($time,,,"%d * %d= %d",a,b,out);
        #2560 $finish;
    end
endmodule

module  mult8(out, a, b);        //8 位乘法器源代码
parameter    size=8;
input  [size:1] a,b;            // 两个操作数
output  [2*size:1] out;         // 结果
assign  out=a*b;                // 乘法运算符

```

```
endmodule
```

### 【例 8.19】8 位加法器的仿真程序

```
`timescale 1ns/1ns

module add8_tp;                                // 仿真模块无端口列表
reg [7:0] A,B;                                  // 输入激励信号定义为 reg 型
reg cin;
wire [7:0] SUM;                                // 输出信号定义为 wire 型
wire cout;
parameter DELY = 100;

add8 AD1(SUM,cout,A,B,cin);                    // 调用测试对象

initial begin                                    // 激励波形设定
    A= 8'd0;      B= 8'd0;      cin=1'b0;
#DELY A= 8'd100;  B= 8'd200;    cin=1'b1;
#DELY A= 8'd200;  B= 8'd88;
#DELY A= 8'd210;  B= 8'd18;     cin=1'b0;
#DELY A= 8'd12;   B= 8'd12;
#DELY A= 8'd100;  B= 8'd154;
#DELY A= 8'd255;  B= 8'd255;    cin=1'b1;
#DELY $finish;
end

// 输出格式定义
initial $monitor($time,,"%d + %d + %b = {%b, %d}",A,B,cin,cout,SUM);

endmodule


module add8(SUM,cout,A,B,cin);                  // 待测试的 8 位加法器模块
output [7:0] SUM;
output cout;
input [7:0] A,B;
input cin;
assign {cout,SUM}=A+B+cin;
endmodule
```

### 【例 8.20】2 选 1 多路选择器的仿真

```
`timescale 1ns/1ns

module mux_tp;
reg a,b,sel;
wire out;
```

```

MUX2_1 m1(out,a,b,sel);                                // 调用待测试模块

initial
begin
    a=1'b0; b=1'b0; sel=1'b0;
    #5 sel=1'b1;
    #5 a=1'b1;      sel=1'b0;
    #5 sel=1'b1;
    #5 a=1'b0; b=1'b1;      sel=1'b0;
    #5 sel=1'b1;
    #5 a=1'b1; b=1'b1; sel=1'b0;
    #5 sel=1'b1;
end
initial      $monitor($time,,,"a=%b b=%b sel=%b out=%b",a,b,sel,out);
endmodule

module  MUX2_1(out,a,b,sel);                            // 待测试的 2 选 1MUX 模块
input  a,b,sel;
output out;
not  #(0.4,0.3) (sel_,sel);                            // #(0.4,0.3) 为门延时
and  #(0.7,0.6) (a1,a,sel_);
and  #(0.7,0.6) (a2,b,sel);
or   #(0.7,0.6) (out,a1,a2);
endmodule

【例 8.21】8 位计数器的仿真
`timescale 10ns/1ns
module count8_tp;
reg  clk,reset;                                         // 输入激励信号定义为 reg 型
wire [7:0] qout;                                       // 输出信号定义为 wire 型
parameter  DELY=100;

counter C1(qout,reset,clk);                            // 调用测试对象

always  #(DELY/2) clk = ~clk;                          // 产生时钟波形

initial
begin                                                    // 激励波形定义
    clk =0; reset=0;

```

```
#DELY    reset=1;
#DELY    reset=0;
#(DELY*300) $finish;
end

// 结果显示

initial    $monitor($time,,,"clk=%d reset=%d qout=%d",clk,reset,qout);
endmodule
```

```
module counter(qout,reset,clk);           // 待测试的 8 位计数器模块
output [7:0] qout;
input  clk,reset;
reg [7:0] qout;
always @(posedge clk)
    begin
        if (reset)    qout<=0;
        else          qout<=qout+1;
    end
endmodule
```

### 【例 9.1】基本门电路的几种描述方法

#### ( 1 ) 门级结构描述

```
module gate1(F,A,B,C,D);
input  A,B,C,D;
output F;
nand (F1,A,B);           // 调用门元件
and (F2,B,C,D);
or (F,F1,F2);
endmodule
```

#### ( 2 ) 数据流描述

```
module gate2(F,A,B,C,D);
input  A,B,C,D;
output F;
assign F=(A&B)|(B&C&D);    //assign    持续赋值
endmodule
```

#### ( 3 ) 行为描述

```
module gate3(F,A,B,C,D);
input  A,B,C,D;
output F;
```

```

reg F;
always @(A or B or C or D)           // 过程赋值
begin
    F=(A&B)|(B&C&D);
end
endmodule

```

**【例 9.2】用 bufif1 关键字描述的三态门**

```

module tri_1(in,en,out);
input  in,en;
output out;
tri out;
bufif1 b1(out,in,en);                // 注意三态门端口的排列顺序
endmodule

```

**【例 9.3】用 assign 语句描述的三态门**

```

module tri_2(out,in,en);
output out;
input  in,en;
assign out = en ? in : 'bz;
        // 若 en=1 , 则 out=in ; 若 en=0 , 则 out 为高阻态
endmodule

```

**【例 9.4】三态双向驱动器**

```

module bidir(tri_inout,out,in,en,b);
inout tri_inout;
output out;
input  in,en,b;
assign tri_inout = en ? in : 'bz;
assign out = tri_inout ^ b;
endmodule

```

**【例 9.5】三态双向驱动器**

```

module bidir2(bidir,en,clk);
inout [7:0] bidir;
input  en,clk;
reg [7:0] temp;
assign bidir= en ? temp : 8'bz;
always @( posedge clk)
begin

```



```

        if (en) temp=bidir;
        else    temp=temp+1;
    end
endmodule

```

### 【例 9.6】3-8 译码器

```

module decoder_38(out,in);
output [7:0] out;
input [2:0] in;
reg [7:0] out;
always @(in)
begin
    case (in)
        3'd0: out=8'b11111110;
        3'd1: out=8'b11111101;
        3'd2: out=8'b11111011;
        3'd3: out=8'b11110111;
        3'd4: out=8'b11101111;
        3'd5: out=8'b11011111;
        3'd6: out=8'b10111111;
        3'd7: out=8'b01111111;
    endcase
end
endmodule

```

### 【例 9.7】8-3 优先编码器

```

module encoder8_3(none_on,outcode,a,b,c,d,e,f,g,h);
output none_on;
output [2:0] outcode;
input a,b,c,d,e,f,g,h;
reg [3:0] outtemp;
assign {none_on,outcode}=outtemp;
always @(a or b or c or d or e or f or g or h)
begin
    if (h) outtemp=4'b0111;
    else if (g) outtemp=4'b0110;
    else if (f) outtemp=4'b0101;
    else if (e) outtemp=4'b0100;
    else if (d) outtemp=4'b0011;
    else if (c) outtemp=4'b0010;

```

```

        else if    (b)          outtemp=4'b0001;
        else if    (a)          outtemp=4'b0000;
        else              outtemp=4'b1000;
    end
endmodule

```

### 【例 9.8】用函数定义的 8-3 优先编码器

```

module code_83(din, dout);
input  [7:0] din;
output [2:0] dout;

function  [2:0] code;          // 函数定义
input  [7:0] din;              // 函数只有输入端口，输出为函数名本身
if  (din[7])                  code = 3'd7;
else if  (din[6])             code = 3'd6;
else if  (din[5])             code = 3'd5;
else if  (din[4])             code = 3'd4;
else if  (din[3])             code = 3'd3;
else if  (din[2])             code = 3'd2;
else if  (din[1])             code = 3'd1;
else                          code = 3'd0;
endfunction

assign  dout = code(din); //          函数调用
endmodule

```

### 【例 9.9】七段数码管译码器

```

module decode47(a,b,c,d,e,f,g,D3,D2,D1,D0);
output  a,b,c,d,e,f,g;
input   D3,D2,D1,D0;          // 输入的 4 位 BCD 码
reg  a,b,c,d,e,f,g;
always  @(D3  or D2  or D1  or D0)
begin
    case  ({D3,D2,D1,D0})      // 用 case 语句进行译码
        4'd0: {a,b,c,d,e,f,g}=7'b1111110;
        4'd1: {a,b,c,d,e,f,g}=7'b0110000;
        4'd2: {a,b,c,d,e,f,g}=7'b1101101;
        4'd3: {a,b,c,d,e,f,g}=7'b1111001;
        4'd4: {a,b,c,d,e,f,g}=7'b0110011;
        4'd5: {a,b,c,d,e,f,g}=7'b1011011;

```

```

        4'd6: {a,b,c,d,e,f,g}=7'b1011111;
        4'd7: {a,b,c,d,e,f,g}=7'b1110000;
        4'd8: {a,b,c,d,e,f,g}=7'b1111111;
        4'd9: {a,b,c,d,e,f,g}=7'b1111011;
        default : {a,b,c,d,e,f,g}=7'bx;
    endcase
end
endmodule

```

### 【例 9.10】奇偶校验位产生器

```

module parity(even_bit,odd_bit,input_bus);
output  even_bit,odd_bit;
input   [7:0] input_bus;
assign  odd_bit = ^ input_bus;           // 产生奇校验位
assign  even_bit = ~odd_bit;            // 产生偶校验位
endmodule

```

### 【例 9.11】用 if-else 语句描述的 4 选 1 MUX

```

module mux_if(out,in0,in1,in2,in3,sel);
output  out;
input   in0,in1,in2,in3;
input   [1:0] sel;
reg  out;
always @(in0 or in1 or in2 or in3 or sel)
begin
    if (sel==2'b00)          out=in0;
    else if (sel==2'b01)    out=in1;
    else if (sel==2'b10)    out=in2;
    else                    out=in3;
end
endmodule

```

### 【例 9.12】用 case 语句描述的 4 选 1 MUX

```

module mux_case(out,in0,in1,in2,in3,sel);
output  out;
input   in0,in1,in2,in3;
input   [1:0] sel;
reg  out;
always @(in0 or in1 or in2 or in3 or sel)
begin

```

```

        case (sel)
            2'b00: out=in0;
            2'b01: out=in1;
            2'b10: out=in2;
            default : out=in3;
        endcase
    end
endmodule

```

### 【例 9.13】用组合电路实现的 ROM

```

module rom(addr,data);
    input  [3:0] addr;
    output [7:0] data;

    function [7:0] romout;
        input [3:0] addr;
        case (addr)
            0 : romout = 0;
            1 : romout = 1;
            2 : romout = 4;
            3 : romout = 9;
            4 : romout = 16;
            5 : romout = 25;
            6 : romout = 36;
            7 : romout = 49;
            8 : romout = 64;
            9 : romout = 81;
            10 : romout = 100;
            11 : romout = 121;
            12 : romout = 144;
            13 : romout = 169;
            14 : romout = 196;
            15 : romout = 225;
            default : romout = 8'hxx;
        endcase
    endfunction
    assign data = romout(addr);
endmodule

```

### 【例 9.14】基本 D 触发器

```

module DFF(Q,D,CLK);
output  Q;
input   D,CLK;
reg  Q;
always @(posedge CLK)
begin
    Q <= D;
end
endmodule

```

**【例 9.15】带异步清 0、异步置 1 的 D 触发器**

```

module DFF1(q,qn,d,clk,set,reset);
input  d,clk,set,reset;
output q,qn;
reg q,qn;
always @(posedge clk or negedge set or negedge reset)
begin
    if (!reset) begin
        q <= 0;           // 异步清 0 , 低电平有效
        qn <= 1;
    end
    else if (!set) begin
        q <= 1;           // 异步置 1 , 低电平有效
        qn <= 0;
    end
    else begin
        q <= d;
        qn <= ~d;
    end
end
endmodule

```

**【例 9.16】带同步清 0、同步置 1 的 D 触发器**

```

module DFF2(q,qn,d,clk,set,reset);
input  d,clk,set,reset;
output q,qn;
reg q,qn;
always @(posedge clk)
begin
    if (reset) begin

```

```

        q <= 0; qn <= 1;           // 同步清 0 , 高电平有效
    end
    else if (set) begin
        q <= 1;   qn <= 0;           // 同步置 1 , 高电平有效
    end
    else begin
        q <= d;   qn <= ~d;
    end
end
endmodule

```

### 【例 9.17】带异步清 0、异步置 1 的 JK 触发器

```

module JK_FF(CLK,J,K,Q,RS,SET);
input  CLK,J,K,SET,RS;
output Q;
reg Q;
always @(posedge CLK or negedge RS or negedge SET)
begin
    if (!RS) Q <= 1'b0;
    else if (!SET) Q <= 1'b1;
    else case ({J,K})
        2'b00 : Q <= Q;
        2'b01 : Q <= 1'b0;
        2'b10 : Q <= 1'b1;
        2'b11 : Q <= ~Q;
        default : Q <= 1'bx;
    endcase
end
endmodule

```

### 【例 9.18】电平敏感的 1 位数据锁存器

```

module latch_1(q,d,clk);
output q;
input d,clk;
assign q = clk ? d : q;           // 时钟信号为高电平时，将输入端数据锁存
endmodule

```

### 【例 9.19】带置位和复位端的 1 位数据锁存器

```

module latch_2(q,d,clk,set,reset);
output q;

```

```

input  d,clk,set,reset;
assign  q = reset ? 0 : (set ? 1 : (clk ? d : q));
endmodule

```

### 【例 9.20】8 位数据锁存器

```

module latch_8(qout,data,clk);
output  [7:0] qout;
input  [7:0] data;
input  clk;
reg [7:0] qout;
always @(clk or data)
begin
if (clk) qout<=data;
end
endmodule

```

### 【例 9.21】8 位数据寄存器

```

module reg8(out_data,in_data,clk,clr);
output  [7:0] out_data;
input  [7:0] in_data;
input  clk,clr;
reg [7:0] out_data;
always @(posedge clk or posedge clr)
begin
if (clr) out_data <=0;
else out_data <=in_data;
end
endmodule

```

### 【例 9.22】8 位移位寄存器

```

module shifter(din,clk,clr,dout);
input  din,clk,clr;
output [7:0] dout;
reg [7:0] dout;
always @(posedge clk)
begin
if (clr)      dout <= 8'b0;           // 同步清 0 , 高电平有效
else
begin
dout <= dout << 1;           // 输出信号左移一位
end
end

```

```

        dout[0] <= din;                // 输入信号补充到输出信号的最低位
    end
end
endmodule

```

### 【例 9.23】可变模加法 / 减法计数器

```

module updown_count(d,clk,clear,load,up_down,qd);
input  [7:0] d;
input  clk,clear,load;
input  up_down;
output [7:0] qd;
reg [7:0] cnt;
assign  qd = cnt;
always @( posedge  clk)
    begin
        if  (!clear)          cnt = 8'h00;           // 同步清 0 , 低电平有效
        else if  (load)       cnt = d;              // 同步预置
        else if  (up_down)    cnt = cnt + 1;        // 加法计数
        else                  cnt = cnt - 1;        // 减法计数
    end
endmodule

```

### 【例 9.24】4 位 Johnson 计数器 ( 异步复位 )

```

module johnson(clk,clr,out);
input  clk,clr;
output [3:0] out;
reg [3:0] out;
always @( posedge  clk   or posedge  clr)
    begin
        if  (clr)          out<= 4'h0;
        else
            begin          out<= out<< 1;
                           out[0]<= ~out[3];
            end
        end
    end
endmodule

```

### 【例 9.25】256 × 8 RAM 模块

```

module ram256x8(data,address,we,inclock,outclock,q);
input  [7:0] data;

```



```

input  [7:0] address;
input  we,inclock,outclock;
output [7:0] q;
lpm_ram_dq myram(.q(q),.data(data),.address(address),
                 .we(we),.inclock(inclock),.outclock(outclock));
defparam myram.lpm_width=8;           // 定义数据宽度
defparam myram.lpm_widthad=8;         // 定义地址宽度
endmodule

```

### 【例 9.26】256 × 16 RAM 块

```

module map_lpm_ram(dataout,datain,addr,we,inclock,outclock);
input  [15:0] datain;                // 端口定义
input  [7:0] addr;
input  we,inclock,outclock;
output [15:0] dataout;
    //lpm_ram_dq  元件例化
lpm_ram_dq ram(.data(datain),.address(addr),.we(we),.inclock(inclock),
               .outclock(outclock),.q(dataout));
defparam ram.lpm_width=16;           // 参数赋值
defparam ram.lpm_widthad=8;
defparam ram.lpm_indata="REGISTERED";
defparam ram.lpm_outdata="REGISTERED";
defparam ram.lpm_file="map_lpm_ram.mif"; //RAM 块中的内容取自该文件
endmodule

```

### 【例 9.27】4 位串并转换器

```

module serial_pal(clk,reset,en,in,out);
input  clk,reset,en,in;
output [3:0] out;
reg [3:0] out;
always @(posedge clk)
begin
    if (reset)        out<=4'h0;
    else if (en)      out<={out,in}; // 使用连接运算符
end
endmodule

```

### 【例 9.28】用函数实现简单的处理器

```

module mpc(instr,out);
input  [17:0] instr;                //instr  为输入的指令

```

---

```

output[ 8:0] out; // 输出结果
reg [8:0] out;
reg func;
reg [7:0] op1,op2; // 从指令中提取的两个操作数

function [16:0] code_add; // 函数的定义
input [17:0] instr;
reg add_func;
reg [7:0] code,opr1,opr2;
begin
code=instr[17:16]; // 输入指令 instr 的高 2 位是操作码
opr1=instr[7:0]; // 输入指令 instr 的低 8 位是操作数 opr1
case (code)
2'b00:
begin
add_func=1;
opr2=instr[15:8]; // 从 instr 中取第二个操作数
end
2'b01:
begin
add_func=0;
opr2=instr[15:8]; // 从 instr 中取第二个操作数
end
2'b10:
begin
add_func=1;
opr2=8'd1; // 第二个操作数取为 1 , 实现 +1 操作
end
default :
begin
add_func=0;
opr2=8'd1; // 实现 -1 操作
end
endcase
code_add={add_func,opr2,opr1};
end
endfunction

always @(instr)
begin

```

```

        {func,op2,op1}=code_add(instr);
        if (func==1) out=op1+op2;
        else          out=op1-op2;
        end
    endmodule

```

// 调用函数  
// 实现两数相加、操作数 1 加 1 操作  
// 实现两数相减、操作数 1 减 1 操作

### 【例 9.29】微处理器的测试代码

```

`timescale 10ns/1ns
`include "mpc.v"
module mpc_tp;
    reg [17:0] instr;
    wire [8:0] out;

    parameter DELY=10;
    mpc m1(instr,out);
    initial begin
        instr=18'd0;
        #DELY instr=18'b00_01001101_00101111;
        #DELY instr=18'b00_11001101_11101111;
        #DELY instr=18'b01_01001101_11101111;
        #DELY instr=18'b01_01001101_00101111;
        #DELY instr=18'b10_01001101_00101111;
        #DELY instr=18'b11_01001101_00101111;
        #DELY instr=18'b00_01001101_00101111;
        #DELY $finish;
    end
    initial $monitor($time,,"instr=%b out=%b",instr,out);
endmodule

```

// 调用待测试模块

### 【例 9.30】乘累加器（MAC）代码

```

module MAC(out,opa,opb,clk,clr);
    output [15:0] out;
    input [7:0] opa,opb;
    input clk,clr;
    wire [15:0] sum;
    reg [15:0] out;

    function [15:0] mult;
        input [7:0] opa,opb;
        reg [15:0] result;

```

// 函数定义，mult 函数完成乘法操作  
// 函数只能定义输入端，输出端口为函数名本身

```

integer    i;

begin
    result = opa[0]? opb : 0;
    for (i= 1; i <= 7; i = i+1)
    begin
        if (opa[i]==1) result=result+(opb<<(i-1));
    end
    mult=result;
end
endfunction

assign    sum=mult(opa,opb)+out;

always    @( posedge  clk    or posedge  clr)
    begin
        if (clr)        out<=0;
        else            out<=sum;
    end

endmodule

```

### 【例 9.31】乘累加器的测试代码

```

'timescale    1ns/1ns
'include    "mac.v"
module  mac_tp;
reg [7:0] opa,opb;                                // 测试输入信号用 reg 型变量
reg  clr,clk;
wire [15:0] out;                                    // 测试输出信号用 wire 型变量
parameter    DELY = 100;
// 测试对象调用
MAC m1(out,opa,opb,clk,clr);

always    #(DELY) clk = ~clk;                        // 产生时钟波形

initial begin                                        // 激励波形定义
    clr=1;clk=0;opa=8'd0; opb=8'd0;
    #DELY clr=0;opa=8'd1; opb=8'd10;
    #DELY opa=8'd2; opb=8'd10;
    #DELY opa=8'd3; opb=8'd10;

```

```
#DELY opa=8'd4; opb=8'd10;
#DELY opa=8'd5; opb=8'd10;
#DELY opa=8'd6; opb=8'd10;
#DELY opa=8'd7; opb=8'd10;
#DELY opa=8'd8; opb=8'd10;
#DELY opa=8'd9; opb=8'd10;
#DELY opa=8'd10; opb=8'd10;
#DELY $finish;
end
```

// 结果显示

```
initial      $monitor($time,,,"clr=%b opa=%d opb=%d out=%d",clr,opa,opb,out);
endmodule
```

### 【例 10.1】非流水线方式 8 位全加器

```
module adder8(cout,sum,ina,inb,cin,clk);
output [7:0] sum;
output cout;
input [7:0] ina,inb;
input cin,clk;
reg [7:0] tempa,tempb,sum;
reg cout;
reg tempc;
always @(posedge clk)
begin
tempa=ina; tempb=inb; tempc=cin;           // 输入数据锁存
end
always @(posedge clk)
begin
{cout,sum}=tempa+tempb+tempc;
end
endmodule
```

### 【例 10.2】4 级流水方式的 8 位全加器

```
module pipeline(cout,sum,ina,inb,cin,clk);
output [7:0] sum;
output cout;
input [7:0] ina,inb;
input cin,clk;
reg [7:0] tempa,tempb,sum;
reg tempci,firstco,secondco,thirdco,cout;
```

```

reg [1:0] firsts,thirda,thirdb;
reg [3:0] seconda,secondb,seconds;
reg [5:0] firsta,firstb,thirds;

always @(posedge clk)
begin
tempa=ina; tempb=inb; tempci=cin;           // 输入数据缓存
end

always @(posedge clk)
begin
{firstco,firsts}=tempa[1:0]+tempb[1:0]+tempci;
                                           // 第一级加（低 2 位）

firsta=tempa[7:2];                       // 未参加计算的数据缓存
firstb=tempb[7:2];

end

always @(posedge clk)
begin
{secondco,seconds}={firsta[1:0]+firstb[1:0]+firstco,firsts};
                                           // 第二级加（第 2、3 位相加）

seconda=firsta[5:2];                     // 数据缓存
secondb=firstb[5:2];

end

always @(posedge clk)
begin
{thirdco,thirds}={seconda[1:0]+secondb[1:0]+secondco,seconds};
                                           // 第三级加（第 4、5 位相加）

thirda=seconda[3:2];                     // 数据缓存
thirdb=secondb[3:2];

end

always @(posedge clk)
begin
{cout,sum}={thirda[1:0]+thirdb[1:0]+thirdco,thirds};
                                           // 第四级加（高两位相加）

end

endmodule

```

### 【例 10.3】两个加法器和一个选择器的实现方式

```

module resource1(sum,a,b,c,d,sel);

parameter size=4;

output [size:0] sum;

```

```

input  sel;
input  [size-1:0] a,b,c,d;
reg [size:0] sum;
always @(a or b or c or d or sel)
    begin
        if (sel)    sum=a+b;
        else       sum=c+d;
    end
endmodule

```

#### 【例 10.4】两个选择器和一个加法器的实现方式

```

module resource2(sum,a,b,c,d,sel);
parameter    size=4;
output  [size-1:0] sum;
input  sel;
input  [size-1:0] a,b,c,d;
reg [size-1:0] atemp,btemp;
reg [size:0] sum;
always @(a or b or c or d or sel)
    begin
        if (sel)    begin atemp=a; btemp=b;          end
        else       begin atemp=c; btemp=d;          end
        sum=atemp+btemp;
    end
endmodule

```

#### 【例 10.5】状态机设计的例子

```

module FSM(clk,clr,out,start,step2,step3);
input  clk,clr,start,step2,step3;
output [2:0] out;
reg [2:0] out;
reg [1:0] state,next_state;

parameter    state0=2'b00,state1=2'b01,
              state2=2'b11,state3=2'b10;
              /* 状态编码，采用格雷（ Gray ）编码方式 */
always @( posedge clk or posedge clr) /* 该进程定义起始状态 */
begin
    if (clr) state <= state0;
    else state <= next_state;
end

```

```

end

always @(state or start or step2 or step3) /* 该进程实现状态的转换 */
begin
case (state)
state0: begin
if (start) next_state <= state1;
else next_state <= state0;
end
state1: begin
next_state <= state2;
end
state2: begin
if (step2) next_state <= state3;
else next_state <= state0;
end
state3: begin
if (step3) next_state <= state0;
else next_state <= state3;
end
default : next_state <= state0; /*default 语句 */
endcase
end

always @(state) /* 该进程定义组合逻辑 ( FSM的输出 ) */
begin
case (state)
state0: out=3'b001;
state1: out=3'b010;
state2: out=3'b100;
state3: out=3'b111;
default :out=3'b001; /*default 语句, 避免锁存器的产生 */
endcase
end

endmodule

```

### 【例 10.6】自动转换量程频率计控制器

/\* 信号定义 :

clk : 输入时钟 ;



```

clear   :      为整个频率计的异步复位信号；
reset   :      用来在量程转换开始时复位计数器；
std_f_sel   :   用来选择标准时基；
cntover  :      代表超量程；
cntlow   :      代表欠量程。
状态 A , B , C , D , E , F 采用一位热码编码  */
module control(std_f_sel,reset,clk,clear,cntover,cntlow);
output [1:0] std_f_sel;
output  reset;
input  clk,clear,cntover,cntlow;
reg [1:0] std_f_sel;
reg reset;
reg [5:0] present,next;           // 用于保存当前状态和次态的中间变量
parameter  start_fl00k=6'b000001,           // 状态 A 编码，采用 1 位热码
            fl00k_cnt=6'b000010,           // 状态 B
            start_fl0k=6'b000100,          // 状态 C
            fl0k_cnt=6'b001000,           // 状态 D
            start_flk=6'b010000,          // 状态 E
            flk_cnt=6'b100000;           // 状态 F

always @( posedge clk   or posedge clear)
begin
    if (clear)      present<=start_fl0k;           //start_fl0k      为起始状态
    else            present<=next;
end

always @(present      or cntover      or cntlow)
begin
    case (present)           // 用 case 语句描述状态转换
start_fl00k:                next<=fl00k_cnt;
fl00k_cnt:
begin
    if (cntlow)      next<=start_fl0k;
    else            next<=fl00k_cnt;
end

start_fl0k:                next<=fl0k_cnt;
fl0k_cnt:
begin
    if (cntlow)                next<=start_flk;
    else if  (cntover)         next<=start_fl00k;
end
end

```

```

        else                                next<=fl0k_cnt;
    end

    start_flk:                            next<=flk_cnt;
    flk_cnt:
        begin
            if (cntover)                    next<=start_fl0k;
            else                            next<=flk_cnt;
        end

    default :next<=start_fl0k;              // 缺省状态为起始状态
endcase
end

always @(present)                        // 该进程产生各状态下的输出
begin
    case (present)
        start_fl00k:                        begin    reset=1; std_f_sel=2'b00;            end
        fl00k_cnt:                          begin    reset=0; std_f_sel=2'b00;            end
        start_fl0k:                         begin    reset=1; std_f_sel=2'b01;            end
        fl0k_cnt:                           begin    reset=0; std_f_sel=2'b01;            end
        start_flk:                          begin    reset=1; std_f_sel=2'b11;            end
        flk_cnt:                            begin    reset=0; std_f_sel=2'b11;            end
        default:                            begin    reset=1; std_f_sel=2'b01;            end
    endcase
end
endmodule

```

### 【例 10.7】8 位全加器

```

module add8(sum,cout,b,a,cin);
output [7:0] sum;
output cout;
input [7:0] a,b;
input cin;
    assign {cout,sum}=a+b+cin;
endmodule

```

### 【例 10.8】8 位寄存器

```

module reg8(qout,in,clk,clear);
output [7:0] qout;
input [7:0] in;
input clk,clear;

```

```
reg [7:0] qout;
always @(posedge clk or posedge clear)
begin
    if (clear)      qout=0;          // 异步清 0
    else            qout=in;
end
endmodule
```

#### 【例 10.9】累加器顶层连接文本描述

```
module acc(accout,cout,accin,cin,clk,clear);
output [7:0] accout;
output  cout;
input  [7:0] accin;
input  cin,clk,clear;
wire [7:0] sum;

add8 accadd8(sum,cout,accout,accin,cin);           // 调用 add8 子模块
reg8 accreg8(accout,sum,clk,clear);                // 调用 reg8 子模块

endmodule
```

#### 【例 10.10】用 `include 描述的累加器

```
`include "add8.v ";
`include "reg8.v ";

module accn(accout,cout,accin,cin,clk,clear);
output [7:0] accout;
output  cout;
input  [7:0] accin;
input  cin,clk,clear;
wire [7:0] sum;

add8 accadd8(sum,cout,accout,accin,cin);           // 调用 add8 子模块
reg8 accreg8(accout,sum,clk,clear);                // 调用 reg8 子模块

endmodule
```

#### 【例 10.11】阻塞赋值方式描述的移位寄存器 1

```
module block1(Q0,Q1,Q2,Q3,din,clk);
output  Q0,Q1,Q2,Q3;
input  clk,din;
```

```

reg Q0,Q1,Q2,Q3;
always @(posedge clk)
begin
    Q3=Q2;           // 注意赋值语句的顺序
    Q2=Q1;
    Q1=Q0;
    Q0=din;
end
endmodule

```

### 【例 10.12】阻塞赋值方式描述的移位寄存器 2

```

module block2(Q0,Q1,Q2,Q3,din,clk);
output Q0,Q1,Q2,Q3;
input clk,din;
reg Q0,Q1,Q2,Q3;
always @(posedge clk)
begin
    Q3=Q2;
    Q1=Q0;           // 该句与下句的顺序与例 10.11 颠倒
    Q2=Q1;
    Q0=din;
end
endmodule

```

### 【例 10.13】阻塞赋值方式描述的移位寄存器 3

```

module block3(Q0,Q1,Q2,Q3,din,clk);
output Q0,Q1,Q2,Q3;
input clk,din;
reg Q0,Q1,Q2,Q3;
always @(posedge clk)
begin
    Q0=din;           //4 条赋值语句的顺序与例 10.11 完全颠倒
    Q1=Q0;
    Q2=Q1;
    Q3=Q2;
end
endmodule

```

### 【例 10.14】非阻塞赋值方式描述的移位寄存器

```

module block4(Q0,Q1,Q2,Q3,din,clk);

```

```

output  Q0,Q1,Q2,Q3;
input   clk,din;
reg     Q0,Q1,Q2,Q3;
always  @(posedge clk)
    begin
        Q3<=Q2;
        Q1<=Q0;
        Q2<=Q1;
        Q0<=din;
    end
endmodule

```

### 【例 10.15】长帧同步时钟的产生

```

module longframe1(clk,strb);
parameter    delay=8;
input        clk;
output       strb;
reg          strb;
reg [7:0] counter;
always @(posedge clk)
    begin
        if (counter==255)      counter=0;
        else                  counter=counter+1;
    end
always @(counter)
    begin
        if (counter<=(delay-1)) strb=1;
        else                  strb=0;
    end
endmodule

```

### 【例 10.16】引入了 D 触发器的长帧同步时钟的产生

```

module longframe2(clk,strb);
parameter    delay=8;
input        clk;
output       strb;
reg [7:0] counter;
reg temp;
reg strb;
always @(posedge clk)

```

```

begin
    if (counter==255)        counter=0;
        else                counter=counter+1;
    end
always @(posedge clk)
begin
    strb=temp;              //      引入一个触发器
end
always @(counter)
begin
    if (counter<=(delay-1)) temp=1;
    else                      temp=0;
end
endmodule

```

### 【例 11.1】数字跑表

/\* 信号定义：

CLK：        CLK 为时钟信号；  
CLR：        为异步复位信号；  
PAUSE：       为暂停信号；  
MSH, MSL：    百分秒的高位和低位；  
SH, SL：      秒信号的高位和低位；  
MH, ML：      分钟信号的高位和低位。    \*/

```
module paobiao(CLK,CLR,PAUSE,MSH,MSL,SH,SL,MH,ML);
```

```
input  CLK,CLR;
```

```
input  PAUSE;
```

```
output [3:0] MSH,MSL,SH,SL,MH,ML;
```

```
reg [3:0] MSH,MSL,SH,SL,MH,ML;
```

```
reg cn1,cn2;                //cn1  为百分秒向秒的进位，  cn2  为秒向分的进位
```

// 百分秒计数进程，每计满 100，cn1 产生一个进位

```
always @(posedge CLK or posedge CLR)
```

```
begin
```

```
    if (CLR) begin          // 异步复位
```

```
        {MSH,MSL}<=8'h00;
```

```
        cn1<=0;
```

```
    end
```

```
    else          if (!PAUSE)          //PAUSE 为 0 时正常计数，为 1 时暂停计数
```

```
        begin
```

```
            if (MSL==9) begin
```

```

        MSL<=0;
        if (MSH==9)
            begin  MSH<=0; cn1<=1;      end
            else  MSH<=MSH+1;
            end
        else
            begin
                MSL<=MSL+1; cn1<=0;
            end
        end
    end
end

```

// 秒计数进程，每计满 60，cn2 产生一个进位

```
always @(posedge cn1 or posedge CLR)
```

```
begin
```

```
    if (CLR) begin          // 异步复位
```

```
        {SH,SL}<=8'h00;
```

```
        cn2<=0;
```

```
    end
```

```
    else if (SL==9)        // 低位是否为 9
```

```
        begin
```

```
            SL<=0;
```

```
            if (SH==5) begin  SH<=0; cn2<=1;      end
```

```
            else          SH<=SH+1;
```

```
        end
```

```
    else
```

```
        begin  SL<=SL+1; cn2<=0;      end
```

```
end
```

// 分钟计数进程，每计满 60，系统自动清零

```
always @(posedge cn2 or posedge CLR)
```

```
begin
```

```
    if (CLR)
```

```
        begin {MH,ML}<=8'h00;      end          // 异步复位
```

```
    else if (ML==9) begin
```

```
        ML<=0;
```

```
        if (MH==5)  MH<=0;
```

```
        else        MH<=MH+1;
```

```
    end
```

```
    else  ML<=ML+1;
```

```
end
```

```
endmodule
```

### 【例 11.2】4 位数字频率计控制模块

```
module fre_ctrl(clk,rst,count_en,count_clr,load);
output count_en,count_clr,load;
input clk,rst;
reg count_en,load;
always @(posedge clk)
begin
    if (rst) begin count_en=0; load=1; end
    else begin
        count_en=~count_en;
        load=~count_en; //load 信号的产生
    end
end
assign count_clr=~clk&load; //count_clr 信号的产生
endmodule
```

### 【例 11.3】4 位数字频率计计数器模块

```
module count10(out,cout,en,clr,clk);
output [3:0] out;
output cout;
input en,clr,clk;
reg [3:0] out;

always @(posedge clk or posedge clr)
begin
    if (clr) out = 0; // 异步清 0
    else if (en)
        begin
            if (out==9) out=0;
            else out = out+1;
        end
    end
    assign cout =((out==9)&en)?1:0; // 产生进位信号
endmodule
```

### 【例 11.4】频率计锁存器模块

```
module latch_16(qo,din,load);
output [15:0] qo;
```



```

input  [15:0] din;
input  load;
reg [15:0] qo;
always @(posedge load)
    begin  qo=din;      end
endmodule

```

### 【例 11.5】交通灯控制器

/\* 信号定义与说明：

CLK： 为同步时钟；

EN： 使能信号，为 1 的话，则控制器开始工作；

LAMPA： 控制 A 方向四盏灯的亮灭；其中， LAMPA0~LAMPA3，分别控制 A 方向的左拐灯、绿灯、黄灯和红灯；

LAMPB： 控制 B 方向四盏灯的亮灭；其中， LAMPB0 ~ LAMPB3，分别控制 B 方向的左拐灯、绿灯、黄灯和红灯；

ACOUNT： 用于 A 方向灯的时间显示， 8 位，可驱动两个数码管；

BCOUNT： 用于 B 方向灯的时间显示， 8 位，可驱动两个数码管。 \*/

```

module traffic(CLK,EN,LAMPA,LAMPB,ACOUNT,BCOUNT);
output  [7:0] ACOUNT,BCOUNT;
output  [3:0] LAMPA,LAMPB;
input  CLK,EN;
reg [7:0] numa,numb;
reg  tempa,tempb;
reg [2:0] counta,countb;
reg [7:0] ared,ayellow,agreen,aleft,bred,byellow,bgreen,bleft;
reg [3:0] LAMPA,LAMPB;

always @(EN)
    if (!EN)
        begin
            // 设置各种灯的计数器的预置数
            ared      <=8'd55;          //55 秒
            ayellow <=8'd5;             //5 秒
            agreen   <=8'd40;          //40 秒
            aleft    <=8'd15;          //15 秒
            bred     <=8'd65;          //65 秒
            byellow  <=8'd5;           //5 秒
            bleft    <=8'd15;          //15 秒
            bgreen   <=8'd30;          //30 秒
        end
end

```

```
assign  ACOUNT=numa;
assign  BCOUNT=numb;

always  @( posedge  CLK)           // 该进程控制  A 方向的四种灯
begin
    if  (EN)
    begin
        if  (!tempa)
        begin
            tempa<=1;
            case  (counta)           // 控制亮灯的顺序
            0:  begin  numa<=agreen;          LAMPA<=2; counta<=1;          end
            1:  begin  numa<=ayellow;         LAMPA<=4; counta<=2;          end
            2:  begin  numa<=aleft;           LAMPA<=1; counta<=3;          end
            3:  begin  numa<=ayellow;         LAMPA<=4; counta<=4;          end
            4:  begin  numa<=ared;            LAMPA<=8; counta<=0;          end
            default  :                       LAMPA<=8;
            endcase
        end
    else begin                        // 倒计时
        if  (numa>1)
            if  (numa[3:0]==0)        begin
                numa[3:0]<=4'b1001;
                numa[7:4]<=numa[7:4]-1;
                end
            else
                numa[3:0]<=numa[3:0]-1;
            if  (numa==2) tempa<=0;
        end
    end
else
    begin
        LAMPA<=4'b1000;
        counta<=0; tempa<=0;
    end
end

always  @( posedge  CLK)           // 该进程控制  B 方向的四种灯
begin
    if  (EN)
    begin
        if  (!tempb)
```

```

begin
tempb<=1;
case (countb)           // 控制亮灯的顺序
    0: begin numb<=bred;      LAMPB<=8; countb<=1;      end
    1: begin numb<=bgreen;    LAMPB<=2; countb<=2;      end
    2: begin numb<=byellow;    LAMPB<=4; countb<=3;      end
    3: begin numb<=bleft;      LAMPB<=1; countb<=4;      end
    4: begin numb<=byellow;    LAMPB<=4; countb<=0;      end
    default :                  LAMPB<=8;
endcase
end
else
begin                    // 倒计时
    if (numb>1)
        if (!numb[3:0])      begin
                                numb[3:0]<=9;
                                numb[7:4]<=numb[7:4]-1;
                                end
        else                  numb[3:0]<=numb[3:0]-1;
        if (numb==2) tempb<=0;
    end
end
else begin
    LAMPB<=4'b1000;
    tempb<=0; countb<=0;
    end
end
endmodule

```

### 【例 11.6】“梁祝”乐曲演奏电路

// 信号定义与说明：

//clk\_4Hz : 用于控制音长（节拍）的时钟频率；

//clk\_6MHz : 用于产生各种音阶频率的基准频率；

//speaker : 用于激励扬声器的输出信号，本例中为方波信号；

//high, med, low : 分别用于显示高音、中音和低音音符，各驱动一个数码管来显示。

```

module song(clk_6MHz,clk_4Hz,speaker,high,med,low);
input  clk_6MHz, clk_4Hz;
output speaker;
output [3:0] high,med,low;

```

```
reg [3:0] high,med,low;
reg [13:0] divider,origin;
reg [7:0] counter;
reg speaker;
wire carry;

assign carry=(divider==16383);

always @(posedge clk_6MHz)
    begin
        if (carry) divider=origin;
        else divider=divider+1;
    end

always @(posedge carry)
    begin
        speaker=~speaker;           //2 分频产生方波信号
    end

always @(posedge clk_4Hz)
    begin
        case ({high,med,low})       // 分频比预置
            'b000000000011: origin=7281;
            'b000000000101: origin=8730;
            'b000000000110: origin=9565;
            'b000000000111: origin=10310;
            'b000000010000: origin=10647;
            'b000000100000: origin=11272;
            'b000000110000: origin=11831;
            'b000001010000: origin=12556;
            'b000001100000: origin=12974;
            'b000100000000: origin=13516;
            'b000000000000: origin=16383;
        endcase
    end

always @(posedge clk_4Hz)
    begin
        if (counter==63) counter=0;           // 计时，以实现循环演奏
        else counter=counter+1;
        case (counter)                         // 记谱
```

```

0: {high,med,low}='b000000000011; // 低音“3”
1: {high,med,low}='b000000000011; // 持续4个时钟节拍
2: {high,med,low}='b000000000011;
3: {high,med,low}='b000000000011;
4: {high,med,low}='b000000000101; // 低音“5”
5: {high,med,low}='b000000000101; // 发3个时钟节拍
6: {high,med,low}='b000000000101;
7: {high,med,low}='b000000000110; // 低音“6”
8: {high,med,low}='b000000010000; // 中音“1”
9: {high,med,low}='b000000010000; // 发3个时钟节拍
10: {high,med,low}='b000000010000;
11: {high,med,low}='b000000100000; // 中音“2”
12: {high,med,low}='b000000000110; // 低音“6”
13: {high,med,low}='b000000010000;
14: {high,med,low}='b000000000101;
15: {high,med,low}='b000000000101;

16: {high,med,low}='b000001010000; // 中音“5”
17: {high,med,low}='b000001010000; // 发3个时钟节拍
18: {high,med,low}='b000001010000;
19: {high,med,low}='b000100000000; // 高音“1”
20: {high,med,low}='b000001100000;
21: {high,med,low}='b000001010000;
22: {high,med,low}='b000000110000;
23: {high,med,low}='b000001010000;
24: {high,med,low}='b000000100000; // 中音“2”
25: {high,med,low}='b000000100000; // 持续11个时钟节拍
26: {high,med,low}='b000000100000;
27: {high,med,low}='b000000100000;
28: {high,med,low}='b000000100000;
29: {high,med,low}='b000000100000;
30: {high,med,low}='b000000100000;
31: {high,med,low}='b000000100000;

32: {high,med,low}='b000000100000;
33: {high,med,low}='b000000100000;
34: {high,med,low}='b000000100000;
35: {high,med,low}='b000000110000; // 中音“3”
36: {high,med,low}='b000000000111; // 低音“7”
37: {high,med,low}='b000000000111;

```

```
38: {high,med,low}='b000000000110; // 低音 “ 6 ”
39: {high,med,low}='b000000000110;
40: {high,med,low}='b000000000101; // 低音 “ 5 ”
41: {high,med,low}='b000000000101;
42: {high,med,low}='b000000000101;
43: {high,med,low}='b000000000110; // 低音 “ 6 ”
44: {high,med,low}='b000000010000; // 中音 “ 1 ”
45: {high,med,low}='b000000010000;
46: {high,med,low}='b000000100000; // 中音 “ 2 ”
47: {high,med,low}='b000000100000;

48: {high,med,low}='b000000000011; // 低音 “ 3 ”
49: {high,med,low}='b000000000011;
50: {high,med,low}='b000000010000; // 中音 “ 1 ”
51: {high,med,low}='b000000010000;
52: {high,med,low}='b000000000110;
53: {high,med,low}='b000000000101; // 低音 “ 5 ”
54: {high,med,low}='b000000000110;
55: {high,med,low}='b000000010000; // 中音 “ 1 ”
56: {high,med,low}='b000000000101; // 低音 “ 5 ”
57: {high,med,low}='b000000000101; // 持续 8 个时钟节拍
58: {high,med,low}='b000000000101;
59: {high,med,low}='b000000000101;
60: {high,med,low}='b000000000101;
61: {high,med,low}='b000000000101;
62: {high,med,low}='b000000000101;
63: {high,med,low}='b000000000101;

endcase
end
endmodule
```

【例 11.7】自动售饮料机

```
/* 信号定义：
clk      :      时钟输入；
reset    :      为系统复位信号；
half_dollar :      代表投入 5 角硬币；
one_dollar  :      代表投入 1 元硬币；
half_out   :      表示找零信号；
dispense   :      表示机器售出一瓶饮料；
collect    :      该信号用于提示投币者取走饮料。 */
```

```

module sell(one_dollar,half_dollar,
            collect,half_out,dispense,reset,clk);
parameter  idle=0,one=2,half=1,two=3,three=4;
            //idle,one,half,two,three          为中间状态变量，代表投入币值的几种情况
input  one_dollar,half_dollar,reset,clk;
output collect,half_out,dispense;
reg collect,half_out,dispense;
reg [2:0] D;
always @(posedge clk)
begin
    if (reset)
        begin
            dispense=0;          collect=0;
            half_out=0;          D=idle;
        end
    case (D)
        idle:
            if (half_dollar)      D=half;
            else if (one_dollar)
                D=one;
        half:
            if (half_dollar)      D=one;
            else if (one_dollar)
                D=two;
        one:
            if (half_dollar)      D=two;
            else if (one_dollar)
                D=three;
        two:
            if (half_dollar)      D=three;
            else if (one_dollar)
                begin
                    dispense=1;    //          售出饮料
                    collect=1;      D=idle;
                end
        three:
            if (half_dollar)
                begin
                    dispense=1;    //          售出饮料

```

```

                                collect=1;          D=idle;
                                end
                                else if    (one_dollar)
                                begin
                                dispense=1;          //          售出饮料
                                collect=1;
                                half_out=1; D=idle;
                                end
                                endcase
                                end
                                endmodule

```

### 【例 11.8】多功能数字钟

/\* 信号定义：

clk : 标准时钟信号，本例中，其频率为 4Hz ；

clk\_1k : 产生闹铃音、报时音的时钟信号，本例中其频率为 1024Hz ；

mode : 功能控制信号； 为 0：计时功能；  
           为 1：闹钟功能；  
           为 2：手动校时功能；

turn : 接按键，在手动校时功能时，选择是调整小时，还是分钟；  
       若长时间按住该键，还可使秒信号清零，用于精确调时；

change : 接按键，手动调整时，每按一次，计数器加 1 ；  
       如果长按，则连续快速加 1，用于快速调时和定时；

hour,min,sec : 此三信号分别输出并显示时、分、秒信号，  
               皆采用 BCD码计数，分别驱动 6 个数码管显示时间；

alert : 输出到扬声器的信号，用于产生闹铃音和报时音；  
       闹铃音为持续 20 秒的急促的“滴滴滴”音，若按住“ change ”键，  
       则可屏蔽该音；整点报时音为“滴滴滴滴—嘟”四短一长音；

LD\_alert : 接发光二极管，指示是否设置了闹钟功能；

LD\_hour : 接发光二极管，指示当前调整的是小时信号；

LD\_min : 接发光二极管，指示当前调整的是分钟信号。

```

*/
module clock(clk,clk_1k,mode,change,turn>alert,LD_alert,LD_hour,LD_min);
                                LD_alert,LD_hour,LD_min);
input  clk,clk_1k,mode,change,turn;
output alert,LD_alert,LD_hour,LD_min;
output [7:0] hour,min,sec;
reg [7:0] hour,min,sec,hour1,min1,sec1,ahour,amin;
reg [1:0] m,fm,num1,num2,num3,num4;
reg [1:0] loop1,loop2,loop3,loop4,sound;

```



```

reg LD_hour,LD_min;
reg clk_1Hz,clk_2Hz,minclk,hclk;
reg alert1,alert2,ear;
reg count1,count2,counta,countb;
wire ct1,ct2,cta,ctb,m_clk,h_clk;

always @(posedge clk)
begin
    clk_2Hz<=~clk_2Hz;
    if (sound==3) begin sound<=0; ear<=1; end
                    //ear 信号用于产生或屏蔽声音
    else begin sound<=sound+1; ear<=0; end
end

always @(posedge clk_2Hz) // 由 4Hz 的输入时钟产生 1Hz 的时基信号
    clk_1Hz<=~clk_1Hz;

always @(posedge mode) //mode 信号控制系统在三种功能间转换
begin if (m==2) m<=0; else m<=m+1; end
always @(posedge turn)
    fm<=~fm;

always // 该进程产生 count1,count2,counta,countb 四个信号
begin
    case (m)
    2: begin if (fm)
            begin count1<=change; {LD_min,LD_hour}<=2; end
            else
            begin counta<=change; {LD_min,LD_hour}<=1; end
            {count2,countb}<=0;
        end
    1: begin if (fm)
            begin count2<=change; {LD_min,LD_hour}<=2; end
            else
            begin countb<=change; {LD_min,LD_hour}<=1; end
            {count1,counta}<=2'b00;
        end
    default : {count1,count2,counta,countb,LD_min,LD_hour}<=0;
    endcase
end
end

```

```

always @(negedge clk)
    // 如果长时间按下 “ change ” 键，则生成 “ num1 ” 信号用于连续快速加 1
    if (count2) begin
        if (loop1==3) num1<=1;
        else
            begin loop1<=loop1+1; num1<=0; end
        end
    else begin loop1<=0; num1<=0; end
always @(negedge clk) // 产生 num2 信号
    if (countb) begin
        if (loop2==3) num2<=1;
        else
            begin loop2<=loop2+1; num2<=0; end
        end
    else begin loop2<=0; num2<=0; end
always @(negedge clk)
    if (count1) begin
        if (loop3==3) num3<=1;
        else
            begin loop3<=loop3+1; num3<=0; end
        end
    else begin loop3<=0; num3<=0; end
always @(negedge clk)
    if (counta) begin
        if (loop4==3) num4<=1;
        else
            begin loop4<=loop4+1; num4<=0; end
        end
    else begin loop4<=0; num4<=0; end

assign ct1=(num3&clk)|(!num3&m_clk); //ct1 用于计时、校时中的分钟计数
assign ct2=(num1&clk)|(!num1&count2); //ct2 用于定时状态下调整分钟信号
assign cta=(num4&clk)|(!num4&h_clk); //cta 用于计时、校时中的小时计数
assign ctb=(num2&clk)|(!num2&countb); //ctb 用于定时状态下调整小时信号

always @(posedge clk_1Hz) // 秒计时和秒调整进程
    if (!(sec1^8'h59)|turn&(!m))
        begin
            sec1<=0; if (!(turn&(!m))) minclk<=1;

```

```

end
// 按住“turn”按键一段时间，秒信号可清零，该功能用于手动精确调时
else begin
    if (sec1[3:0]==4'b1001)
    begin    sec1[3:0]<=4'b0000; sec1[7:4]<=sec1[7:4]+1;    end
    else    sec1[3:0]<=sec1[3:0]+1; minclk<=0;
    end
assign    m_clk=minclk||count1;

always    @(posedge ct1)    // 分计时和分调整进程
begin
    if (min1==8'h59)    begin    min1<=0; hclk<=1;    end
    else    begin
        if (min1[3:0]==9)
        begin    min1[3:0]<=0; min1[7:4]<=min1[7:4]+1;    end
        else    min1[3:0]<=min1[3:0]+1; hclk<=0;
        end
    end
assign    h_clk=hclk||counta;

always    @(posedge cta)    // 小时计时和小时调整进程
    if (hour1==8'h23) hour1<=0;
    else    if (hour1[3:0]==9)
        begin    hour1[7:4]<=hour1[7:4]+1; hour1[3:0]<=0;    end
        else    hour1[3:0]<=hour1[3:0]+1;

always    @(posedge ct2)    // 闹钟定时功能中的分钟调节进程
    if (amin==8'h59) amin<=0;
    else    if (amin[3:0]==9)
        begin    amin[3:0]<=0; amin[7:4]<=amin[7:4]+1;    end
        else    amin[3:0]<=amin[3:0]+1;

always    @(posedge ctb)    // 闹钟定时功能中的小时调节进程
    if (ahour==8'h23) ahour<=0;
    else    if (ahour[3:0]==9)
        begin    ahour[3:0]<=0; ahour[7:4]<=ahour[7:4]+1;    end
        else    ahour[3:0]<=ahour[3:0]+1;

always    // 闹铃功能
    if ((min1==amin)&&(hour1==ahour)&&(amin|ahour)&&(!change))

```

```

// 若按住 “ change ” 键不放，可屏蔽闹铃音
if (sec1<8'h20) alert1<=1;           // 控制闹铃的时间长短
else  alert1<=0;
else  alert1<=0;

always                                // 时、分、秒的显示控制
case (m)
3'b00:  begin  hour<=hour1; min<=min1; sec<=sec1;          end
           // 计时状态下的时、分、秒显示
3'b01:  begin  hour<=ahour; min<=amin; sec<=8'hzz;          end
           // 定时状态下的时、分、秒显示
3'b10:  begin  hour<=hour1; min<=min1; sec<=8'hzz;          end
           // 校时状态下的时、分、秒显示
endcase

assign  LD_alert=(ahour|amin)?1:0;    // 指示是否进行了闹铃定时
assign  alert=((alert1)?clk_1k&clk:0)|alert2;    // 产生闹铃音或整点报时音

always                                // 产生整点报时信号  alert2
begin
if ((min1==8'h59)&&(sec1>8'h54)||(!(min1|sec1)))
if (sec1>8'h54) alert2<=ear&clk_1k;    // 产生短音
else  alert2<=!ear&clk_1k;            // 产生长音
else  alert2<=0;
end
endmodule

```

### 【例 11.9】电话计费器程序

/\* 信号定义：

```

clk      :      时钟信号，本例中其频率值为      1Hz ；
decide   :      电话局反馈回来的信号，代表话务种类，      “ 01 ” 表示市话， “ 10 ” 表示
                  长话， “ 11 ” 表示特话；
dispmoney :      用来显示卡内余额，其单位为角，这里假定能显示的最大数额为      50 元
                  ( 500 角 )；
disptime  :      显示本次通话的时长；
write,read :      当 write  信号下降沿到来时写卡，当话卡插入，      read  信号变高时读卡；
warn      :      余额过少时的告警信号。本例中，当打市话时，余额少于      3 角，打长
                  话时，余额少于 6 角，即会产生告警信号；
cut       :      当告警时间过长时自动切断通话信号。      */

```

```

module account(state,clk,card,decide,disptime,dispmoney,
               write,read,warn,cut);
output  write,read,warn,cut;
input   state,clk,card;
input   [2:1] decide;
output  [10:0] dispmoney;
output  [8:0] disptime;
reg [10:0] money;
reg [8:0] dtime;
reg warn,cut,write,t1m;           //t1m 为分时钟
reg set,reset_ena;
integer num1,temp;

assign dispmoney=card?money:0;
assign disptime=dtime;
assign read=card?1:0;

                                // 产生分时钟
always @(posedge clk)
begin
    if (num1==59)    begin num1<=0; t1m<=1;    end
    else begin
        if (state)    num1<=num1+1;
        else    num1<=0; t1m<=0;
    end
end

always @(negedge clk)           // 该进程完成电话计费功能
begin
    if (!set)
        begin money<=11'h500; set<=1;    end
    if (card&state)
        if (t1m)
            case ({state,decide})
3'b101:    if (money<3)
                begin warn<=1; write<=0; reset_ena<=1;    end
            else
                begin                                // 市话计费
                    if (money[3:0]<4'b0011)
                        begin
                            money[3:0]<=money[3:0]+7;

```

```

if (money[7:4]!=0)
    money[7:4]<=money[7:4]-1;
else
    begin    money[7:4]<=9; money[10:8]<=money[10:8]-1;          end
    end
else    money[3:0]<=money[3:0]-3; write<=1;
        // 市话通话计时

if (dtime[3:0]==9)
    begin
        dtime[3:0]<=0;
        if (dtime[7:4]==9)
            begin    dtime[7:4]<=0; dtime[8]<=dtime[8]+1;          end
        else    dtime[7:4]<=dtime[7:4]+1;
        end
    else
        begin
            dtime[3:0]<=dtime[3:0]+1; warn<=0; reset_ena<=0;
        end
    end
end
3'b110:    if (money<6)
            begin    warn<=1; write<=0; reset_ena<=1;          end
            else begin
                    // 通话计时

                if (dtime[3:0]==9)
                    begin
                        dtime[3:0]<=0; if(dtime[7:4]==9)
                        begin    dtime[7:4]<=0; dtime[8]<=dtime[8]+1;          end
                        else    dtime[7:4]<=dtime[7:4]+1;
                        end
                    else    dtime[3:0]<=dtime[3:0]+1;
                            // 长话计费

                    if (money[3:0]<4'b0110)
                        begin
                            money[3:0]<=money[3:0]+4;
                            if (!money[7:4])
                                begin    money[7:4]<=9; money[10:8]<=money[10:8]-1;          end
                                else    money[7:4]<=money[7:4]-1;
                                end
                            else    money[3:0]<=money[3:0]-6;
                                write<=1; reset_ena<=0; warn<=0;

```

```

        end
    endcase
    else write<=0;
else begin    dtime<=0; warn<=0; write<=0; reset_ena<=0;                end
    // 取卡后对一些信号进行复位
end

always @(posedge clk) // 该进程在告警时间过长的情况下切断本次通话
begin
    if (warn) temp<=temp+1;
    else temp<=0;
    if (temp==15)
        begin cut<=1; temp<=0;    end
    if (!card||!reset_ena)
        begin
            cut<=0;                // 复位 cut 信号
            temp<=0;
        end
end
endmodule

```

### 【例 12.1】8 位级连加法器

```

module add_jl(sum,cout,a,b,cin);
output [7:0] sum;
output cout;
input [7:0] a,b;
input cin;

full_add1 f0(a[0],b[0],cin,sum[0],cin1);                // 级连描述
full_add1 f1(a[1],b[1],cin1,sum[1],cin2);
full_add1 f2(a[2],b[2],cin2,sum[2],cin3);
full_add1 f3(a[3],b[3],cin3,sum[3],cin4);
full_add1 f4(a[4],b[4],cin4,sum[4],cin5);
full_add1 f5(a[5],b[5],cin5,sum[5],cin6);
full_add1 f6(a[6],b[6],cin6,sum[6],cin7);
full_add1 f7(a[7],b[7],cin7,sum[7],cout);
endmodule

module full_add1(a,b,cin,sum,cout);                //1 位全加器
input a,b,cin;

```

```

output  sum,cout;
wire  s1,m1,m2,m3;
and  (m1,a,b),
      (m2,b,cin),
      (m3,a,cin);
xor  (s1,a,b),
      (sum,s1,cin);
or   (cout,m1,m2,m3);
endmodule

```

### 【例 12.2】8 位并行加法器

```

module  add_bx(cout,sum,a,b,cin);
output  [7:0] sum;
output  cout;
input  [7:0] a,b;
input  cin;
      assign  {cout,sum}=a+b+cin;
endmodule

```

### 【例 12.3】8 位超前进位加法器

```

module  add_ahead(sum,cout,a,b,cin);
output  [7:0] sum;
output  cout;
input  [7:0] a,b;
input  cin;
wire  [7:0] G,P;
wire  [7:0] C,sum;

assign  G[0]=a[0]&b[0];           // 产生第 0 位本位值和进位值
assign  P[0]=a[0]|b[0];
assign  C[0]=cin;
assign  sum[0]=G[0]^P[0]^C[0];

assign  G[1]=a[1]&b[1];           // 产生第 1 位本位值和进位值
assign  P[1]=a[1]|b[1];
assign  C[1]=G[0]|(P[0]&cin);
assign  sum[1]=G[1]^P[1]^C[1];

assign  G[2]=a[2]&b[2];           // 产生第 2 位本位值和进位值
assign  P[2]=a[2]|b[2];

```



```

assign  C[2]=G[1]||(P[1]&C[1]);
assign  sum[2]=G[2]^P[2]^C[2];

assign  G[3]=a[3]&b[3];           // 产生第 3 位本位值和进位值
assign  P[3]=a[3]||b[3];
assign  C[3]=G[2]||(P[2]&C[2]);
assign  sum[3]=G[3]^P[3]^C[3];

assign  G[4]=a[4]&b[4];           // 产生第 4 位本位值和进位值
assign  P[4]=a[4]||b[4];
assign  C[4]=G[3]||(P[3]&C[3]);
assign  sum[4]=G[4]^P[4]^C[4];

assign  G[5]=a[5]&b[5];           // 产生第 5 位本位值和进位值
assign  P[5]=a[5]||b[5];
assign  C[5]=G[4]||(P[4]&C[4]);
assign  sum[5]=G[5]^P[5]^C[5];

assign  G[6]=a[6]&b[6];           // 产生第 6 位本位值和进位值
assign  P[6]=a[6]||b[6];
assign  C[6]=G[5]||(P[5]&C[5]);
assign  sum[6]=G[6]^P[6]^C[6];

assign  G[7]=a[7]&b[7];           // 产生第 7 位本位值和进位值
assign  P[7]=a[7]||b[7];
assign  C[7]=G[6]||(P[6]&C[6]);
assign  sum[7]=G[7]^P[7]^C[7];

assign  cout=G[7]||(P[7]&C[7]);    // 产生最高位进位输出
endmodule

```

#### 【例 12.4】8 位并行乘法器

```

module  mult(outcome,a,b);
parameter  size=8;
input  [size:1] a,b;           // 两个操作数
output  [2*size:1] outcome;    // 结果
assign  outcome=a*b;          // 乘法运算符
endmodule

```

#### 【例 12.5】4 × 4 查找表乘法器

```
module mult4x4(out,a,b,clk);
output [7:0] out;
input [3:0] a,b;
input clk;
reg [7:0] out;
reg [1:0] firsta,firstb;
reg [1:0] seconda,secondb;
wire [3:0] outa,outb,outc,outd;
always @( posedge clk)
begin
    firsta = a[3:2]; seconda = a[1:0];
    firstb = b[3:2]; secondb = b[1:0];
end

lookup      m1(outa,firsta,firstb,clk),
             m2(outb,firsta,secondb,clk),
             m3(outc,seconda,firstb,clk),
             m4(outd,seconda,secondb,clk); // 模块调用

always @( posedge clk)
begin
    out = (outa << 4) + (outb << 2) + (outc << 2) + outd;
end
endmodule
```

```
module lookup(out,a,b,clk); // 用查找表方式实现 2 × 2 乘法
output [3:0] out;
input [1:0] a,b;
input clk;
reg [3:0] out;
reg [3:0] address;
always @( posedge clk)
begin
    address = {a,b};
    case (address)
        4'h0 : out = 4'b0000;
        4'h1 : out = 4'b0000;
        4'h2 : out = 4'b0000;
        4'h3 : out = 4'b0000;
        4'h4 : out = 4'b0000;
```

```

        4'h5 : out = 4'b0001;
        4'h6 : out = 4'b0010;
        4'h7 : out = 4'b0011;
        4'h8 : out = 4'b0000;
        4'h9 : out = 4'b0010;
        4'ha : out = 4'b0100;
        4'hb : out = 4'b0110;
        4'hc : out = 4'b0000;
        4'hd : out = 4'b0011;
        4'he : out = 4'b0110;
        4'hf : out = 4'b1001;
        default : out='bx;
    endcase
end
endmodule

```

### 【例 12.6】8 位加法树乘法器

```

module add_tree(out,a,b,clk);
output [15:0] out;
input [7:0] a,b;
input clk;
wire [15:0] out;
wire [14:0] out1,c1;
wire [12:0] out2;
wire [10:0] out3,c2;
wire [8:0] out4;
reg [14:0] temp0;
reg [13:0] temp1;
reg [12:0] temp2;
reg [11:0] temp3;
reg [10:0] temp4;
reg [9:0] temp5;
reg [8:0] temp6;
reg [7:0] temp7;

function [7:0] mult8x1; // 该函数实现 8×1 乘法
input [7:0] operand;
input sel;
begin
    mult8x1= (sel) ? (operand) : 8'b00000000;

```

```

    end
endfunction

always @(posedge clk) // 调用函数实现操作数 b 各位与操作数 a 的相乘
begin
    temp7<=mult8x1(a,b[0]);
    temp6<=((mult8x1(a,b[1]))<<1);
    temp5<=((mult8x1(a,b[2]))<<2);
    temp4<=((mult8x1(a,b[3]))<<3);
    temp3<=((mult8x1(a,b[4]))<<4);
    temp2<=((mult8x1(a,b[5]))<<5);
    temp1<=((mult8x1(a,b[6]))<<6);
    temp0<=((mult8x1(a,b[7]))<<7);
end

assign out1 = temp0 + temp1; // 加法器树运算
assign out2 = temp2 + temp3;
assign out3 = temp4 + temp5;
assign out4 = temp6 + temp7;
assign c1 = out1 + out2;
assign c2 = out3 + out4;
assign out = c1 + c2;

endmodule

```

### 【例 12.7】11 阶 FIR 数字滤波器

```

module fir(clk,x,y);
input [7:0] x;
input clk;
output [15:0] y;
reg [15:0] y;
reg [7:0] tap0,tap1,tap2,tap3,tap4,tap5,tap6,tap7,tap8,tap9,tap10;
reg [7:0] t0,t1,t2,t3,t4,t5;
reg [15:0] sum;

always @(posedge clk)
begin
    t0<=tap5;
    t1<=tap4+tap6;
    t2<=tap3+tap7;

```

```

        t3<=tap2+tap8;
        t4<=tap1+tap9;
        t5<=tap0+tap10;           // 利用对称性
        sum<=(t1<<4)+{t1[7],t1[7:1]}+{t1[7],t1[7],t1[7:2]}+
            {t1[7],t1[7],t1[7],
            t1[7:3]}-(t2<<3)-(t2<<2)+t2-{t2[7],t2[7],t2[7:2]}
        +(t3<<2)+t3+{t3[7],t3[7],t3[7:2]}+{t3[7],t3[7],t3[7],t3[7],t3[7:4]}
        +{t3[7],t3[7],t3[7],t3[7],t3[7],t3[7:5]}
        -t4-{t4[7],t4[7:1]}-{t4[7],t4[7],t4[7],t4[7:3]}
        +{t5[7],t5[7:1]}-{t5[7],t5[7],t5[7],t5[7],t5[7],t5[7:5]}
        +(t0<<7)-((t0<<2)<<2)-(t0<<2)+{t0[7],t0[7:1]}
        +{t0[7],t0[7],t0[7:2]}+{t0[7],t0[7],t0[7],t0[7],t0[7:4]};

//16+0.5+0.25+0.125=16.875
//8+4-1+0.25=11.25
//4+1+0.25+0.0625+0.03125=5.34375
//1+0.5+0.125=1.625
//0.5-0.03125=0.46875
//128-4*4-4+0.5+0.25+0.0625=108.8125
/* 0.0036    , -0.0127    , 0.0417    , -0.0878    , 0.1318    , 0.8500    , 0.1318    , -0.0878    ,
   0.0417    , -0.0127    , 0.0036    , 0.4608    , -1.6256    , 5.3376    , -11.2384    , 16.8704    ,
   108.800    , 16.8704    , -11.238    , 5.3376    , -1.6256    , 0.4608 */

tap10<=tap9;
tap9<=tap8;
tap8<=tap7;
tap7<=tap6;
tap6<=tap5;
tap5<=tap4;
tap4<=tap3;
tap3<=tap2;
tap2<=tap1;
tap1<=tap0;
tap0<=x;

y<={sum[15],sum[15],sum[15],sum[15],sum[15],sum[15],sum[15],sum[15:7]}
;
end
endmodule

```

### 【例 12.8】16 位高速数字相关器

```

module correlator(out,a,b,clk);
output  [4:0] out;
input   [15:0] a,b;
input   clk;
wire    [2:0] sum1,sum2,sum3,sum4;
wire    [3:0] temp1,temp2;

detect    u1(sum1,a[3:0],b[3:0],clk),           // 模块调用
          u2(sum2,a[7:4],b[7:4],clk),
          u3(sum3,a[11:8],b[11:8],clk),
          u4(sum4,a[15:12],b[15:12],clk);
add3      u5(temp1,sum1,sum2,clk),
          u6(temp2,sum3,sum4,clk);
add4      u7(out,temp1,temp2,clk);
endmodule


module detect(sum,a,b,clk);                     // 该模块实现 4 位相关器
output    [2:0] sum;
input     clk;
input     [3:0] a,b;
wire      [3:0] ab;
reg        [2:0] sum;

assign    ab = a ^ b;
always    @( posedge  clk)
begin
    case  (ab)
        'd0: sum = 4;
        'd1,'d2,'d4,'d8:          sum = 3;
        'd3,'d5,'d6,'d9,'d10,'d12: sum = 2;
        'd7,'d11,'d13,'d14: sum = 1;
        'd15: sum = 0;
    endcase
end
endmodule


module add3(add,a,b,clk);                       //3 位加法器
output    [3:0] add;
input     [2:0] a,b;
input     clk;

```

```
reg [3:0] add;
always @(posedge clk)
    begin add = a + b;    end
endmodule
```

```
module add4(add,a,b,clk);           //4 位加法器
output [4:0] add;
input [3:0] a,b;
input clk;
reg [4:0] add;
always @(posedge clk)
    begin add = a + b;    end
endmodule
```

#### 【例 12.9】( 7 , 4 ) 线性分组码编码器

```
module linear(c,u,clk);
output [6:0] c;           //c 为编码输出码字
input [3:0] u;
input clk;
reg [6:0] c;
always @(posedge clk)
    begin
        c[6] = u[3];
        c[5] = u[2];
        c[4] = u[1];
        c[3] = u[0];
        c[2] = u[1] ^ u[2] ^ u[3];
        c[1] = u[0] ^ u[1] ^ u[2];
        c[0] = u[0] ^ u[2] ^ u[3];
    end
endmodule
```

#### 【例 12.10】( 7 , 4 ) 线性分组码译码器

```
module decoder1(c,y,clk);
output [6:0] c;
input [6:0] y;
input clk;
reg [2:0] s;
reg [6:0] e,c;
always @(posedge clk)
```

```

begin
    s[0] = y[0] ^ y[3] ^ y[5] ^ y[6];
    s[1] = y[1] ^ y[3] ^ y[4] ^ y[5];
    s[2] = y[2] ^ y[4] ^ y[5] ^ y[6];           //s[0]~ s[2]      为伴随子
    e[0] = s[0] & (~s[1]) & (~s[2]);
    e[1] = (~s[0]) & s[1] & (~s[2]);
    e[2] = (~s[0]) & (~s[1]) & s[2];
    e[3] = s[0] & s[1] & (~s[2]);
    e[4] = (~s[0]) & s[1] & s[2];
    e[5] = s[0] & s[1] & s[2];
    e[6] = s[0] & (~s[1]) & s[2];           //e[0]~ e[6]      为错误图样
    c = e ^ y;                             //c   为输出码字
end
endmodule

```

### 【例 12.11】( 7 , 4 ) 循环码编码器

```

module cycle(c,u,clk);
output [6:0] c;
input [3:0] u;
input clk;
reg [2:0] i;
reg d0,d1,d2,temp;
reg [6:0] c;

always @(posedge clk)
begin
    d0=0; d1=0; d2=0;           // 初始化
    for (i=0;i<4;i=i+1)         // 该 for 循环计算码组的前 4 个码元
    begin
        temp = d2 ^ c[i];
        d2 = d1; d1 = d0 ^ temp;
        d0 = temp; c[i] = u[i];
    end
    for (i=4;i<7;i=i+1)         // 该 for 循环计算码组的后 3 个码元
    begin
        temp = d2;
        d2 = d1; d1 = d0 ^ temp;
        d0 = temp; c[i] = temp;
    end
end
end

```



```
endmodule
```

【例 12.12】( 7 , 4 ) 循环码纠错译码器

```
module decoder2(c,y,clk);
output [6:0] c;           //c 为输出码字 , c[6] 为高次项
input [6:0] y;           //y 为接收码字 , y[6] 为高次项
input clk;
reg [6:0] c,c_buf,buffer;
reg temp;
reg s0,s1,s2;           // 伴随式电路寄存器
reg e;                  // 错误检测输出信号
integer i;

always @( posedge clk)
begin
    s0=0;    s1=0;    s2=0;           // 初始化
    temp=0;
    buffer=y;                         // 接收码字移入缓存

    for (i=6;i>=0;i=i-1)             // 接收码字进入除法电路
    begin
        e=s0&(~s1)&temp;
        temp=s2;
        s2=s1;
        s1=s0^temp;
        s0=y[i]^temp^e;
    end

    for (i=6;i>=0;i=i-1)             // 输出纠错译码后的码字
    begin
        e=s0&(~s1)&temp;
        temp=s2;
        s2=s1;
        s1=s0^temp;
        s0=temp^e;
        c_buf[i]=buffer[i]^e;
        if (e==1)                   // 若出错 , 对缓存进行清零
        begin
            s0=0;    s1=0;    s2=0;
        end
    end
end
```

```

        end
    end
end

```

```

always  @(posedge  clk)
    begin
        c=c_buf;
    end
endmodule

```

### 【例 12.13】CRC 编码

```

module  crc(crc_reg,crc,d,calc,init,d_valid,clk,reset);
output  [15:0] crc_reg;
output  [7:0] crc;
input   [7:0] d;
input   calc;
input   init;
input   d_valid;
input   clk;
input   reset;

reg [15:0] crc_reg;
reg [7:0] crc;
wire [15:0] next_crc;

always  @(  posedge  clk   or posedge  reset)
    begin
        if  (reset)
            begin
                crc_reg <= 16'h0000;
                crc <= 8'h00;
            end

        else if  (init)
            begin
                crc_reg <= 16'h0000;
                crc <= 8'h00;
            end

        else if  (calc & d_valid)
            begin

```

```

    crc_reg <= next_crc;
    crc <= ~{next_crc[8], next_crc[9], next_crc[10], next_crc[11],
            next_crc[12], next_crc[13], next_crc[14], next_crc[15]};
    end

    else if    (~calc & d_valid)
    begin
        crc_reg <= {crc_reg[7:0], 8'h00};
        crc <= ~{crc_reg[0], crc_reg[1], crc_reg[2], crc_reg[3],
                crc_reg[4], crc_reg[5], crc_reg[6], crc_reg[7]};
    end
end

assign  next_crc[0] = crc_reg[12] ^ d[7] ^ crc_reg[8] ^ d[3];
assign  next_crc[1] = crc_reg[13] ^ d[6] ^ d[2] ^ crc_reg[9];
assign  next_crc[2] = d[5] ^ crc_reg[14] ^ d[1] ^ crc_reg[10];
assign  next_crc[3] = d[4] ^ crc_reg[15] ^ d[0] ^ crc_reg[11];
assign  next_crc[4] = crc_reg[12] ^ d[3];
assign  next_crc[5] = crc_reg[12] ^ crc_reg[13] ^ d[7] ^ crc_reg[8] ^ d[2] ^ d[3];
assign  next_crc[6] = crc_reg[13] ^ d[6] ^ crc_reg[14] ^ d[1] ^ d[2] ^
crc_reg[9];
    assign  next_crc[7] = d[5] ^ crc_reg[14] ^ crc_reg[15] ^ d[0] ^ d[1] ^
crc_reg[10];
    assign  next_crc[8] = d[4] ^ crc_reg[15] ^ d[0] ^ crc_reg[0] ^ crc_reg[11];
    assign  next_crc[9] = crc_reg[12] ^ crc_reg[1] ^ d[3];
    assign  next_crc[10] = crc_reg[13] ^ d[2] ^ crc_reg[2];
    assign  next_crc[11] = crc_reg[3] ^ crc_reg[14] ^ d[1];
    assign  next_crc[12] = crc_reg[12] ^ crc_reg[4] ^ d[7] ^ crc_reg[15]
                        ^ d[0] ^ crc_reg[8] ^ d[3];
    assign  next_crc[13] = crc_reg[13] ^ d[6] ^ crc_reg[5] ^ d[2] ^ crc_reg[9];
    assign  next_crc[14] = d[5] ^ crc_reg[14] ^ crc_reg[6] ^ d[1] ^ crc_reg[10];
    assign  next_crc[15] = d[4] ^ crc_reg[15] ^ d[0] ^ crc_reg[7] ^ crc_reg[11];
endmodule

```