EDA 复习

考试说明

- **分值：**
  - 选择题 10 题，**20 分**
  - 程序阅读（补齐代码）4-5 段代码，15 空，共 **30 分**
  - 编程题 4 题，共 **50 分**
- **重点：**
  - 基础语法
    - 关键字、标识符格式、常量定义格式；√
    - 变量定义类型，net 和 variable 两大类；wire、reg 和 integer 3种常用的数据类型；√
    - 数据类型的判断，always 或 initial 中被赋值的变量一定是 variable（reg 或 integer）类型。其他 assign、门原件调用的输出、模块调用的输出都是 wire类型；√
    - parameter 常量定义的两种格式和调用的格式；√
    - 向量和标量基本操作；运算符的基本功能，特别注意 {} ；√
    - always 和 intial 的适用范围和区别；√
    - always 敏感列表格式，同步/异步、高低电平、上升沿/下降沿关系；√
    - 阻塞和非阻塞赋值的区别，<=和=适用范围；√
    - 函数/任务/模块定义和调用；√
  - 程序设计
    - 组合逻辑设计（基本逻辑、条件完备性、优先级）√
    - 时序逻辑（二进制计数器、BCD 码计数器）√
    - 流水线 √
    - 状态机（一段、两段、三段式）√
    - 测试代码（时钟信号、输入信号、输出）√

1、关键词、标识符、常量 √

- **关键词**
  - 关键字都是小写
  - 变量定义不能和关键词冲突
- 例：always、assign、case、module
- **标识符**
  - 是任意一组字母、数字以及符号 $ 和 _ 的组合
  - 标识符的第一个字符必须是字母或者下划线
  - 标识符小于1023字符
  - 标识符是区分大小写的
  - 例：COUNT、count、_A1、Begin、R56_1、qout、clk、clr
- **常量**
  - 常量的值不能被改变
  - Verilog 中的常量主要有如下3种类型：
    - 整数（可综合）20、37
    - 实数（不可综合）12.45

- 字符串（不可综合）"hello"
- 对于整数：

+/- '
+/- <位宽> ' <进制> <数字>
size：为对应**二进制数**的宽度
base：为进制
value：是基于进制的数字序列

进制	举例
二进制（b / B）	10’b1010
八进制（o / O）	10’o57
十进制（d / D）	10’d23 / 23
十六进制（h / H）	10’hB5F

  - 较长的数字间可用下划线分开，例：8’b1100_0101
  - 数字不说位宽时，默认值为32位，例：25
  - ‘O27没位宽，宽度为相应数值中定义的位数
  - 如果定义的位宽比实际的位数长，通常在左边填0补位，如果是x或z则相用x或z左边补齐[10’d10、10’hx5、5’h8A]
  - 如果定义的位宽比实际的位数小，左边的位截掉
  - 整数可带符号，负数通常表示为二进制补码形式 [-4’D2]
  - 5’Hx=5’bxxxxx；4'hZ =4’bzzzz；4'B1x_01； x或z代表的位宽取决于所用的进制
  - 非法案例：3’□b001、4’d-4、(3+2)’b10

2、变量、数据类型 √

- Verilog有下面**四种基本的逻辑状态**：
  - 0：低电平、逻辑0或逻辑非
  - 1：高电平、逻辑1或“真”
  - x或X：不确定或未知的逻辑状态
  - z或Z：高阻态
  - Verilog中的所有数据类型都在上述4类逻辑状态中取值；其中x和z都不区分大小写，值0x1z与值0X1Z是等同的
- **变量两种数据类型**：
  - net型：常用的有wire、tri
  - variable型：包括reg、integer

wire	reg	integer
最常用的net类型数据	最常用的variable型变量	属于一种variable型变量
例：wire [m:n] a,b,c;	例：reg [m:n] a,b,c;	例：integer a,b,c;
定义无符号数	定义无符号数	表示符号数
多位宽无符号数需定义	多位宽无符号数需要定义	不用定义位宽 默认为32位有符号数
input、output、内部信号	output、内部信号	output
		不能作为位向量访问，i[0]、i[20:15]错误

> input、output 默认为 wire
>
> assign 赋值输出一定是 **wire** 等 net 类型；
>
> 门元件调用、模块调用的输出端输出一定是 **wire** 等 net 类型；
>
> **always** 或 **initial** 过程块中赋值语句（表达式左侧被赋值变量） 输出一定是**reg**或**integer**等variable类型。

- 信号可以分为端口信号和内部信号
  - 端口信号:
    - 输入端口只能是 net 类型
    - 输出端口可以是 net 或 variable 类型
      - 若输出端口always、initial中被赋值则为variable类型
      - 在过程块外赋值（assign、门调用、模块实例化）则为net类型
  - 内部信号：
    - 内部信号类型可以是 net 或 variable 类型
      - 若在过程块中赋值，则为 reg 或 integer 类型
      - 若在过程块外赋值，则为 net 类型（wire）
    - 若信号既需要在过程块中赋值，又需要在过程块外赋值（不允许，但有可能出现，这时需要一个中间信号转换）

3、参数、标量、向量 √

- **参数**
  - paramater定义符号常量，不能对其修改赋值
  - 例1：

```
parameter WIDTH=4,sel=8,code=8'ha3;
reg[WIDTH:0] reg1;
```

  - 例2：

```
// 也可以采用下面的定义形式【含参数定义的子模块】
module johnson #(parameter WIDTH=4, WIDTH1=8)(clk,clr,qout);
    //...略
endmodule

// top_m为主模块，调用johnson子模块
module top_m();
    johnson #(3,6) johnson_inst(...);   // 模块调用形式1
    johnson #(WIDTH(6),WIDTH(3)) johnson_inst1(...);    // 模块调用形式2
endmodule
```

- **标量**
  - 线宽为1位的变量
  - 例：wire a；reg clk；
  - 如果在变量声明中没有指定位宽，则默认为标量（1位）
- **向量**
  - 线宽大于1位的变量（包括net型和variable型）
  - 例：wire / reg [7:0] a；reg[0:7] rc；(rc[0]为最高有效位，rc[7]为最低有效位)
  - 向量的操作：
    - 位选择：可任意选中向量中的一位　A=mybyte[6]；
    - 域选择 ： 可任意选中向量中的相邻几位　B=mybyte[5:2]；
- 存储器
  - reg bm [5:1]; //容量为5，字长1的存储器
  - reg [3:0] am [63:0];　//容量64，字长4的存储器

4、运算符 √

- **算数运算符**
  - +（加、正数）
  - -（减、负数）
  - *（乘）
  - /（除，结果取整数位）
  - %（模运算 or 取余运算，两侧都为整数，余数符号位与第一个操作数相同）
  - 注意：
    - 算数运算符中出现z或x时，整个运算为x‘b10x1+‘b01111=‘bxxxxx
    - 运算结果长度按表达式中（左、右端）最大长度运算，多丢弃、少补零
- **逻辑运算符**（一般用于条件判断）
  - &&（逻辑与）
  - ||（逻辑或）
  - !　（逻辑非）
  - 注意：
    - 操作数不只1位，则应将操作数作为一个整体来对待
      - 如果操作数全0，则相当于逻辑0；如果某一位为1，则整体看做逻辑1
      - 例：(100 && 0) = 0；(4‘b1001 || 0) = 1；! 3‘b100 = 0；

        - 操作数出现x，结果为x，!x=x
            - 例：4 && 3'b1x0 = x；!(5'd4x) = x；
- **关系运算符**
    - <（小于）、<=（小于等于）、>（大于）、>=（大于等于）
    - <= 也表示信号的一种赋值操作
    - 如果操作数的值不定，则关系结果模糊，返回值是不定值x
- **等式运算符**
    - ==（等于）、!=（不等于）、===（全等）、!==（不全等）
    - == 参与比较的两个操作数必须逐位相等，其相等比较的结果才为1，如果某些位不定态或高阻值，其相等比较得到的结果是不定值x
    - === 不定态或高阻值也必须全一致才为1
- **缩位运算符**
    - ~（按位取反）
    - &（按位与）、~&（与非）
    - |（按位或）、~|（或非）
    - ^（按位异或）
    - ^~，~^（按位同或）
    - 注意：
        - 缩位运算符将一个矢量缩减为一个标量
        - 两个长度不同的数据进行位运算，自动右端对其，高位用0补齐
        - 除了~，x或z出现都为x， ~x=x；~z=z；

a	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
b	1	1	1	1	1	0	1	0
~a	0	1	0	0	1	1	0	0
a & b	1	0	1	1	0	0	1	0
a | b	1	1	1	1	1	0	1	1
a ^ b	0	1	0	0	1	0	0	1
a ^~ b	1	0	1	1	0	1	1	0

> &a = 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 = 0
>
> |a = 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 = 1
>
> ^a = 1 ^ 0 ^ 1 ^ 1 ^ 0 ^ 0 ^ 1 ^ 1 = 1 【偶数个1输出0，奇数个1输出1】
>
> ^~a = 1 ^~ 0 ^~ 1 ^~ 1 ^~ 0 ^~ 0 ^~ 1 ^~ 1 = 0

- 移位运算符
    - <<（左移，×2）、>>（右移，÷2）
    - 该移位为逻辑移位，移出的位用0添补
- 条件运算符
    - 信号 = 条件？表达式1：表达式2；
        - 当条件成立时，信号取表达式1的值，反之取表达式2的值
        - 若为x或z，计算两个表达式，若两个表达式某一位为1，则这一位结果为1，同时为0则为0，否则为x

- 位拼接运算符
  - {}，将两个或多个信号的某些位拼接起来
  - 不允许连接非定长常数，如 {db,5}

```
reg [3:0] a,b,sum;
reg [5:0] c;
reg count=0;
a={a,count};         // 单向左移
a={count,a[3:1]};    // 单向右移
a={a[0],a[3:1]};     // 循环右移
a={a,a[3]};          // 循环左移
```

5、always、initial √

always	initial
用于仿真和可综合电路	只用于仿真中的初始化
always块内的语句则是**不断重复执行**的	initial过程块中的语句**仅执行一次**

- always

> always 过程语句通常是带有触发条件的，触发条件写在**敏感信号表达式**中，只有当触发条件满足时， begin-end 块语句才能被执行。

  - 出现在敏感列表中的使能信号，都是异步信号，不出现是同步
  - 异步使能信号，posedge对应高电平有效，negedge对应低电平有效

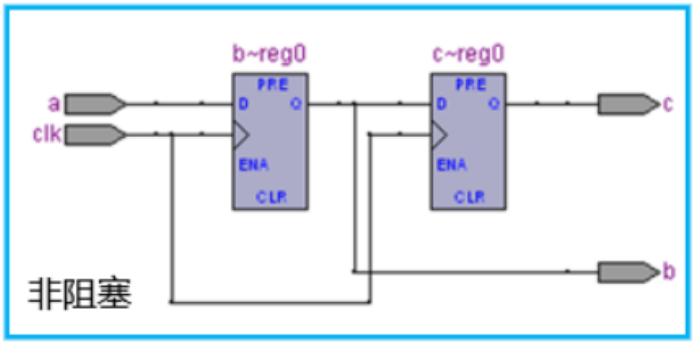
```
always@(<敏感信号表达式>)  // 信号形式要统一（电平信号、边沿信号）
    begin   // 过程赋值
        // if-else，case，casex，casez选择语句
        // while，repeat，for循环
        // task，function调用
    end
// 例：
always @(a) begin end       // 当信号a的值发生改变
always @(a,b) begin end     // 当信号a或信号b的值发生改变
always @(*) begin end       // 所有驱动信号

always @(posedge clock) begin end       //当clock 的上升沿到来时
always @(negedge clock) begin end       //当clock 的下降沿到来时
always @(posedge clk, negedge reset)
```

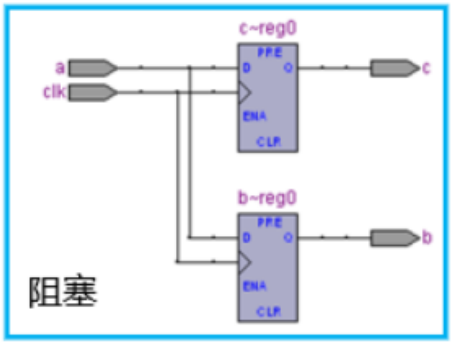
```
always @(posedge clk)   // clk上升沿触发
    begin
        if(!reset) out=8'h00;   // 同步清0，低电平有效
        else if(load) out=data; // 同步预置
        else        out=out+1;  // 计数
    end
```

```
always @(posedge clk, posedge reset, negedge load)
    begin
        if(reset) out=8'h00;     // 异步清0，高电平有效
        else if(!load) out=data;// 异步预置，低电平有效
        else        out=out+1;  // 计数
    end
```

6、赋值语句 √

持续赋值语句	过程赋值语句
assign	非阻塞赋值（<=）、阻塞赋值（=）
被赋值变量为net类型（wire）	被赋值变量必须为variable类型（reg、integer）
assign c = a & b;	b <= a; 、b = a;






```
// 非阻塞赋值
always @(posedge clk)
    begin
        b<=a; // b=0, a=1
        c<=b;
    end // 经过一个clk上升沿触发 a=1, b=1, c=0

// 阻塞赋值
always @(posedge clk)
    begin
        b=a; // b=0, a=1
        c=b;
    end // 经过一个clk上升沿触发 a=1, b=1, c=1
```

7、条件语句

- if-else
  - 条件语句必须在always或initial过程块中使用
  - 表达式：一般为逻辑表达式或关系表达式
    - 表达式结果0，x，z为假；否则为真
- case、casez、casex

case	0	1	x	z	casez	0	1	x	z	casex	0	1	x	z
0	1	0	0	0	0	1	0	0	1	0	1	0	1	1
1	0	1	0	0	1	0	1	0	1	1	0	1	1	1
x	0	0	1	0	x	0	0	1	1	x	1	1	1	1
z	0	0	0	1	z	1	1	1	1	z	1	1	1	1

> 严格程度：case (逐位相等) > casez (忽略z对应位) > casex (忽略z、x对应位)

```
module(out, opcode, a, b);
    output reg[7:0] out;
    input[2:0] opcode; input[7:0] a,b;
    always@(opcode or a or b)
        begin
            case(opcode)
                3'd0: out = a + b;
                3'd1: out = a - b;
                3'd2: out = a & b;
                3'd3: out = a | b;
                3'd4: out= ~a;
                default: out = 8'hx;
            endcase
        end
endmodule
```

```
module mux_casez(out,a,b,c,d,sel);
    input a,b,c,d;
    input[3:0] sel;
    output reg out;
    always@(a or b or c or d or sel)
        begin
            casez(sel)
                4'b???1:out=a; 4'b??10:out=b;
                4'b?100:out=c; 4'b1000:out=d;
                default:out=1'bx;
            endcase
        end
endmodule
```

8、循环语句

- forever：连续地执行语句；多用在 initial 块中，以生成时钟等周期性波形
- repeat：连续执行一条语句n次
- while：执行一条语句直到某个条件不满足
- for：有条件的循环语句（绝大多数综合器支持）

```
// forever
initial
    begin
        clk = 0; #20;
        forever #5 clk=~clk; // 连续不断执行，可用disable语句中断
    end
// repeat
initial
```

```
    begin
        repeat(5) out=out+1; // 循环5次
    end
// while
initial
    begin
        i=0;
        while(i<10) i=i+1;
    end
```

9、任务、函数 √

- 任务（task）

```
// 输入:decc; 输出:outt
task dec_out;
    input integer decc;                 // 输入，十进制数
    output reg[6:0] outt;               // 输出，7位二进制数值
    if(decc==0)      outt=7'b1000000;   // 七段管显示0
    else if(decc==1) outt=7'b1111001;   // 七段管显示1
    else if(decc==2) outt=7'b0100100;   // 七段管显示2
    else if(decc==3) outt=7'b0110000;   // 七段管显示3
    else if(decc==4) outt=7'b0011001;   // 七段管显示4
    else if(decc==5) outt=7'b0010010;   // 七段管显示5
    else if(decc==6) outt=7'b0000010;   // 七段管显示6
    else if(decc==7) outt=7'b1111000;   // 七段管显示7
    else if(decc==8) outt=7'b0000000;   // 七段管显示8
    else if(decc==9) outt=7'b0011000;   // 七段管显示9
    else             outt=7'b1111111;   // 七段管不显示
endtask

dec_out(one,out);   // 调用
```

  - 任务的定义与调用须在一个module模块内
  - 定义时，没有端口名列表
  - 任务名调用，调用时，需端口列表，按序匹配
  - 一个任务可以调用别的任务和函数，可以调用的任务和函数个数不限
- 函数（function）

```
// 输入:din; 输出:code
function [2:0] code;
    input[7:0] din;
    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_0001: code=3'h0;
        default:      code=3'hx;
    endcase
endfunction

assign dout=code(din);  // 调用
```

  - 函数的定义与调用须在一个module模块内

    - 函数只允许有输入变量且必须至少有一个输入变量，输出变量由函数名本身担任，在定义函数时，需对函数名说明其类型和位宽
    - 定义函数时，没有端口名列表，但调用函数时，需列出端口名列表，端口名的排序和类型必须与定义时的相一致，这一点与任务相同
- 函数可以出现在持续赋值assign的右端表达式中
    - 函数不能调用任务，而任务可以调用别的任务和函数，且调用任务和函数个数不受限制

比较项目	任务（task）	函数（function）
输入与输出	可有任意个各种类型的参数	至少有一个输入， 不能将 inout 类型作为输出
调用	任务只可在过程语句中调用， 不能在连续赋值语句 assign 中调用	函数可作为表达式中的一个操作数来调用，在过程赋值和连续赋值语句中均可以调用
定时事件控制（#，@和 wait）	任务可以包含定时和事件控制语句	函数不能包含这些语句
调用其它任务和函数	任务可调用其它任务和函数	函数可调用其它函数，但不可以调用其它任务
返回值	任务不向表达式返回值	函数向调用它的表达式返回一个值

10、组合逻辑电路

> 三人表决电路

```
module vote(a,b,c,out);
    input a,b,c;
    output out;
    wire[1:0] sum;
    assign sum=a+b+c;
    assign out=sum[1]?1:0;
endmodule
```

```
module vote(a,out);
    input[2:0] a;
    output reg out;
    wire[1:0] sum;
    summ s1(a,sum); // 模块调用
    always@(a)
        begin
            if(sum>1) out=1;
            else out=0;
        end
endmodule

module summ(a,sum);
    input[2:0] a;
    output[1:0] sum;
    assign sum=a[0]+a[1]+a[2];
endmodule
```

> 数据选择器（优先级）

```
// 编译时出现未知原因，理解设计思想就行
module select(a,b,c,d,sel,out);
```

```
    input a,b,c,d;
    input[3:0] sel;
    output reg out;
    always@(a,b,c,d,sel)
        begin   // 也可用if-else
            casez(sel)
                4'bxxx1:out=a;
                4'bxx10:out=b;
                4'bx100:out=c;
                4'b1000:out=d;
                default:out=1'bx;
            endcase
        end
endmodule
```

4位二进制加法器

```
module add_4(ina,inb,cin,cout,sum);
    // ina, inb:两个4位无符号二进制数; cin:前级进位
    // sum:4位加法和; cout:1位进位
    input[3:0] ina,inb;
    input cin;
    output cout;
    output[3:0] sum;
    assign {cout,sum}=ina+inb+cin;
endmodule
```

BCD码加法器

```
module add_bcd(ina,inb,cin,cout,sum);
    input[3:0] ina,inb;
    input cin;
    output reg cout;
    output reg[3:0] sum;
    reg[4:0] temp;
    always@(ina,inb,cin)
        begin
            temp=ina+inb+cin;
            // 若大于9，不符合BCD码规范，则强制进位
            if(temp>9) {cout,sum}=temp+6;
            else {cout,sum}=temp;
        end
endmodule
```

for实现2个8位数相乘

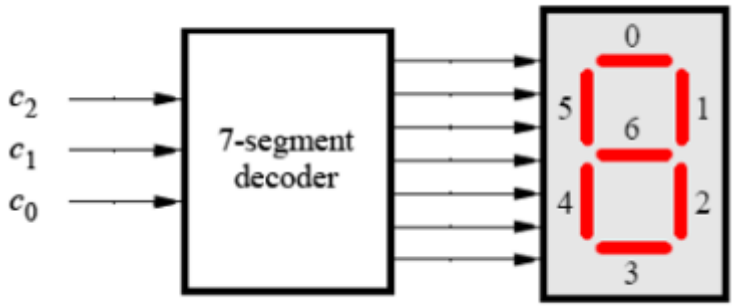
```
module mult(a,b,out);
    parameter size=8;
    input[size:1] a,b;
    output reg[2*size:1] out;
    integer i;
    always @(a,b)
        begin
            out=0;
            for(i=1; i<=size; i=i+1)
                if(b[i]) out=out+(a<<i-1));
        end
endmodule
```

二进制数显示电路

```
// 输入一个八位二进制数，输出对应的十进制数
module test03(in8,out1,out2,out3);
    input [7:0] in8;            // 8路输入，高电平1为有效电平
    output reg[6:0] out1;       // 7路输出，对应第一个七段管十进制数的显示（个位）
    output reg[6:0] out2;       // 7路输出，对应第二个七段管十进制数的显示（十位）
    output reg[6:0] out3;       // 7路输出，对应第三个七段管十进制数的显示（百位）
    wire[3:0] ones;             // 内部变量，记录十进制数~个位
    wire[3:0] tens;             // 内部变量，记录十进制数~十位
    wire[3:0] hundreds;         // 内部变量，记录十进制数~百位
    assign ones=in8%10;         // 个位赋值
    assign tens=(in8/10)%10;    // 十位赋值
    assign hundreds=in8/100;    // 百位赋值
    always@(in8)
        begin
            if(in8<10)          // in8>=0 and in8<10
                begin
                    dec_out(ones,out1);     // 第一个七段管 显示个位
                    out2=7'b1111111;        // 第二个七段管（十位）不显示
                    out3=7'b1111111;        // 第三个七段管（百位）不显示
                end
            else if(in8<100)                // in8>=10 and in8<100
                begin
                    dec_out(ones,out1);     // 第一个七段管 显示个位
                    dec_out(tens,out2);     // 第二个七段管 显示十位
                    out3=7'b1111111;        // 第三个七段管（百位）不显示
                end
            else                            // in8>=100
                begin
                    dec_out(ones,out1);     // 第一个七段管 显示个位
                    dec_out(tens,out2);     // 第二个七段管 显示十位
                    dec_out(hundreds,out3); // 第三个七段管 显示百位
                end
        end
    task dec_out;
        input integer decc;                 // 输入，十进制数
        output reg[6:0] outt;               // 输出，7位二进制数值
        if(decc==0)      outt=7'b1000000;   // 七段管显示0
        else if(decc==1) outt=7'b1111001;   // 七段管显示1
        else if(decc==2) outt=7'b0100100;   // 七段管显示2
        else if(decc==3) outt=7'b0110000;   // 七段管显示3
        else if(decc==4) outt=7'b0011001;   // 七段管显示4
```

```
        else if(decc==5) outt=7'b0010010;   // 七段管显示5
        else if(decc==6) outt=7'b0000010;   // 七段管显示6
        else if(decc==7) outt=7'b1111000;   // 七段管显示7
        else if(decc==8) outt=7'b0000000;   // 七段管显示8
        else if(decc==9) outt=7'b0011000;   // 七段管显示9
        else             outt=7'b1111111;   // 七段管不显示
    endtask
endmodule
```

附：七段管显示原理图



OUT	显示	OUT	显示	OUT	显示
7'b1000000	0	7'b1111001	1	7'b0100100	2
7'b0110000	3	7'b0011001	4	7'b0010010	5
7'b0000010	6	7'b1111000	7	7'b0000000	8
7'b0011000	9	7'b0001000	A	7'b0000011	b
7'b1000110	c	7'b0100001	d	7'b0000110	E
7'b0001110	F	7'b1111111	不显示		

11、时序逻辑电路

> 模16二进制计数器

```
module count16(clk,clr,sum);
    input clk,clr;
    output reg [3:0] sum=0;
    always @(posedge clk)
        begin
            if(clr) sum<=0;
            else sum<=sum+1;
        end
endmodule
```

> 模10二进制计数器 / 4位模10BCD码计数器

```
module count10(clk,clr,sum);
    input clk,clr;
    output reg[3:0] sum=0;
    always @(posedge clk)
        begin
            if(clr) sum<=0;
            else if(sum==9) sum<=0;
            else sum<=sum+1;
        end
endmodule
```

> 8位模100 BCD码计数器（00-99）

```
module BCD100(clk,clr,count);
    input clk,clr;
    output reg [7:0] count=0;
    always @(posedge clk)
        begin
            if(clr) count<=0;
            else if(count[3:0]<9) count[3:0]<=count[3:0]+1;
            else
                begin
                    count[3:0]=0;
                    if(count[7:4]<9) count[7:4]<=count[7:4]+1;
                    else count[7:4]<=0;
                end
        end
endmodule
```

> 8位模16 BCD码计数器（00-15）

```
module BCD16(clk,clr,count);
    input clk,clr;
    output reg [7:0] count=0;
    always @(posedge clk)
        begin
            if(clr) count<=0;
            else if(count[7:4]==0)
                begin
                    if(count[3:0]<9)  count[3:0]<=count[3:0]+1;
                    else
                        begin
                            count[3:0]<=0;
                            count[7:4]<=count[7:4]+1;
                        end
                end
            else
                begin
                    if(count[3:0]<5) count[3:0]<=count[3:0]+1;
                    else
                        begin
                            count[3:0]<=0;
                            count[7:4]<=0;
                        end
                end
        end
```

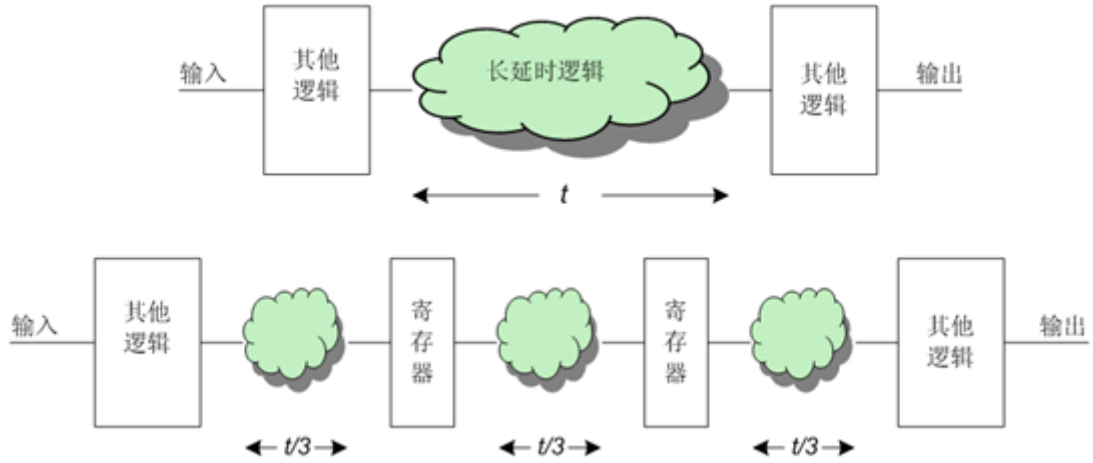
```
endmodule
```

模60的BCD码加法计数器

```
module BCD60(data,reset,load,cin,clk,qout,cout);
    input[7:0] data;            // 预置数
    input reset,load,cin,clk;   // 控制信号
    output reg[7:0] qout=0;     // 计数值
    output cout;                // 进位信号
    always @(posedge clk)
        begin
            if(reset) qout<=0;          // 同步清零
            else if(load) qout<=data;   // 同步置数
            else if(cin)    // 计数
                begin
                    if(qout[3:0]==9)
                        begin
                            qout[3:0]<=0;
                            if(qout[7:4]==5) qout[7:4]<=0;
                            else qout[7:4]<=qout[7:4]+1;
                        end
                    else
                        qout[3:0]<=qout[3:0]+1;
                end
        end
    assign cout=((qout==8'h59)&cin)?1:0;    // 进位
endmodule
```

12、流水线

设计理念：将其复杂的逻辑功能分解为多个小的步骤来实现，以减小每个部分的延时，在各步间加入寄存器，以暂存中间结果，从而提高整个系统的工作频率。代价是增加了寄存器逻辑，增加了芯片资源的耗用。

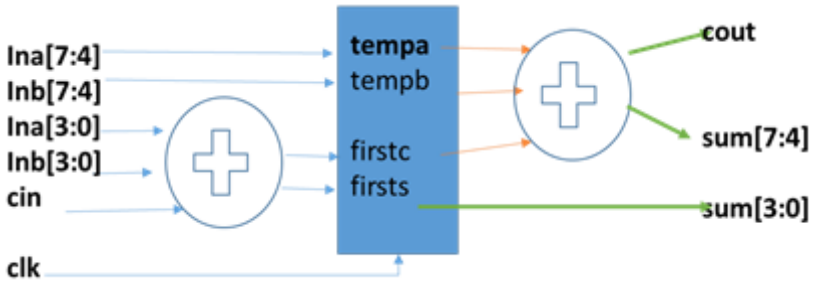


非流水线方式8位全加器

```
module adder8(cout,sum,ina,inb,cin,clk);
    input[7:0] ina,inb;     // 输入数(8位)
    input cin,clk;          // cin:前级进位
    output reg[7:0] sum;    // 输出和
    output reg cout;        // 输出进位
```

```
    reg[7:0] tempa,tempb;    // 缓存数据ina,inb
    reg tempc;               // 缓存数据cin

    always @(posedge clk)
        begin   // 输入数据锁存
            tempa=ina;
            tempb=inb;
            tempc=cin;
        end
    always @(posedge clk)
        begin   // 输出
            {cout,sum}=tempa+tempb+tempc;
        end
endmodule
```

两级流水实现的8位加法器




```
module adder_pipe2(cout,sum,ina,inb,cin,clk);
    input[7:0] ina,inb;
    input cin,clk;
    output reg[7:0] sum;
    output reg cout;
    reg[3:0] tempa,tempb,firsts;
    reg firstc;
    always @(posedge clk)
        begin
            {firstc,firsts}=ina[3:0]+inb[3:0]+cin;
            tempa=ina[7:4];
            tempb=inb[7:4];
        end
    always @(posedge clk)
        begin
            {cout,sum[7:4]}=tempa+tempb+firstc;
            sum[3:0]=firsts;
        end
endmodule
```

四级流水线实现的8位加法器

```
// 理解原理!!!
module  pipeline(cout,sum,ina,inb,cin,clk);
    input [7:0] ina,inb;
    input cin,clk;
    output reg[7:0] sum;
    output reg cout;
    reg[7:0] tempa,tempb;
    reg tempcin,firstco,secondco,thirdco;
```

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

    always @(posedge clk)
        begin   // 输入数据缓存
            tempa=ina;
            tempb=inb;
            tempcin=cin;
        end
    always @(posedge clk)
        begin   // 第一级加（0,1）
            {firstco,firsts}=tempa[1:0]+tempb[1:0]+tempci;
            firsta=tempa[7:2];  // 数据缓存
            firstb=tempb[7:2];
        end
    always @(posedge clk)
        begin   // 第二级加（2, 3）
            {secondco,seconds}={firsta[1:0]+firstb[1:0]+firstco,firsts};
            seconda=firsta[5:2];// 数据缓存
            secondb=firstb[5:2];
        end
    always @(posedge clk)
        begin   // 第三级加（4, 5）
            {thirdco,thirds}={seconda[1:0]+secondb[1:0]+secondco,seconds};
            thirda=seconda[3:2];// 数据缓存
            thirdb=secondb[3:2];
        end
    always @(posedge clk)
        begin   //第四级加（6, 7）
            {cout,sum}={thirda[1:0]+thirdb[1:0]+thirdco,thirds};
        end
endmodule
```

13、状态机

- 分类:（输出信号产生方式的不同）
  - 摩尔型：输出只与当前状态有关系
  - 米里型：输出与当前状态和输入有关系
  - 输出变化：摩尔型比米里型晚一个周期
- 表示法：
  - 状态图、状态表、流程图
- **状态编码方式**：
  - 顺序编码：采用顺序的二进制数进行编码
    - 例：2'b00、2'b01、2'b10、2'b11
  - 格雷码:格雷码算法
    - 例：2'b00、2'b01、2'b11、2'b10
  - 约翰逊编码：约翰逊计数器产生，最高位取反循环移位到最低位
    - 例：000、001、011、111、110、100、000
  - 一位热码：采用n位对n个状态编码
    - 例：0001、0010、0100、1000
- **状态编码的定义**：

```
// 方式1: 用 parameter 参数定义
parameter state1=2'b00,state2=2'b01,state3=2'b10,state4=2'b11;

// 方式2: 用 'define 语句定义, 不加 ;
'define state1 2'b00
'define state2 2'b01
'define state3 2'b10
'define state4 2'b11
```

- 几种描述方式
  - 用三个过程描述：现态（CS）、次态（NS）、输出逻辑（OL）各用一个always过程描述
  - 双过程描述1：(CS+NS、OL双过程描述)：使用两个always过程来描述有限状态机，一个过程描述现态和次态时序逻辑（CS+NS）；另一个过程描述输出逻辑（OL）
  - 双过程描述2：(CS、NS+OL双过程描述)：一个过程用来描述现态（CS）；另一个过程描述次态和输出逻辑（NS+OL）
  - 单过程描述：在单过程描述方式中，将状态机的现态、次态和输出逻辑（CS+NS+OL）放在一个always过程中进行描述

> 模5计数器（三段式）

```
module fsm(clk,clr,state,cout);
    input clk,clr;
    output reg cout;
    output reg[2:0] state=3'b000;
    reg[2:0] next_state;
    // 现态
    always@(posedge clk,posedge clr)
        begin
            if(clr) state<=0; // 异步复位
            else state<=next_state;
        end
    // 次态
    always@(state)
        begin
            case(state)
                3'b000: next_state<=3'b001;
                3'b001: next_state<=3'b010;
                3'b010: next_state<=3'b011;
                3'b011: next_state<=3'b100;
                3'b100: next_state<=3'b000;
                default:next_state<=3'b000;
            endcase
        end
    // 输出逻辑
    always@(state)
        begin
            case(state)
                3'b100: cout=1'b1;
                default:cout=1'b0;
            endcase
        end
endmodule
```

> 模5计数器（二段式）

```
module test01(clk,clr,qout,cout);
    input clk,clr;
    output reg cout;
    output reg[2:0] qout=3'b000;
    always@(posedge clk,posedge clr)
        begin
            if(clr) qout<=0; //异步复位
            else
                begin
                    case(qout)
                        3'b000: qout<=3'b001;
                        3'b001: qout<=3'b010;
                        3'b010: qout<=3'b011;
                        3'b011: qout<=3'b100;
                        3'b100: qout<=3'b000;
                        default:qout<=3'b000;
                    endcase
                end
        end
    always@(qout)   // 此过程产生输出逻辑
        begin
            case(qout)
                3'b100: cout=1'b1;
                default:cout=1'b0;
            endcase
        end
endmodule
```

模5计数器（一段式）

```
module fsm(clk,clr,qout,cout);
    input clk,clr;
    output reg[2:0] qout;
    output reg cout;
    always@(posedge clk,posedge clr)
        begin
            if(clr) qout<=0; //异步复位
            else
                case(qout)
                    3'b000: begin qout<=3'b001; cout<=0; end
                    3'b001: begin qout<=3'b010; cout<=0; end
                    3'b010: begin qout<=3'b011; cout<=0; end
                    3'b011: begin qout<=3'b100; cout<=0; end
                    3'b100: begin qout<=3'b000; cout<=1; end
                    default:begin qout<=3'b000; cout<=0; end
                endcase
        end
endmodule
```

101 序列检测器（三段式）

```
module fsm1_seq101(clk,clr,cin,qout);
    input clk,clr,cin;
    output reg qout;
    reg[1:0] state,next_state;
    parameter S0=2'b00,S1=2'b01,S2=2'b11,S3=2'b10;  // 格雷编码
```

```
    always@(posedge clk,posedge clr)// 当前状态
        begin
            if(clr) state<=S0; // 异步复位，s0为起始状态
            else state<=next_state;
        end
    always@(state,cin)  // 定义状态
        begin
            case (state)
                S0:
                    begin
                        if(cin) next_state<=S1;
                        else  next_state<=S0;
                    end
                S1:
                    begin
                        if(cin) next_state<=S1;
                        else  next_state<=S2;
                    end
                S2:
                    begin
                        if(cin) next_state<=S3;
                        else  next_state<=S0;
                    end
                S3:
                    begin
                        if(cin) next_state<=S1;
                        else  next_state<=S2;
                    end
                default: next_state<=S0;
            endcase
        end
    always@(state)  // 该过程产生输出逻辑
        begin
            case(state)
                S3: qout=1'b1;
                default: z=1'b0;
            endcase
        end
endmodule
```

101 序列检测器（二段式）

```
module fsm1_seq101(clk,clr,cin,qout);
    input clk,clr,cin;
    output reg qout;
    reg[1:0] state;
    parameter S0=2'b00,S1=2'b01,S2=2'b11,S3=2'b10;
    always@(posedge clk,posedge clr)
        begin
            if(clr) state<=S0;
            else begin
                case (state)
                    S0:begin if(cin) state<=S1; else state<=S0; end
                    S1:begin if(cin) state<=S1; else state<=S2; end
                    S2:begin if(cin) state<=S3; else state<=S0; end
                    S3:begin if(cin) state<=S1; else state<=S2; end
```

```
                    default: state<=S0;
                endcase
            end
        end
    always@(state)
        begin
            case(state)
                S3: qout=1'b1;
                default: qout=1'b0;
            endcase
        end
endmodule
```

101 序列检测器（一段式）

```
module fsm1_seq101(clk,clr,cin,qout);
    input clk,clr,cin;
    output reg qout;
    reg[1:0] state;
    parameter S0=2'b00,S1=2'b01,S2=2'b11,S3=2'b10;
    always@(posedge clk,posedge clr)
        begin
            if(clr) state<=S0;
            else case(state)
                S0:
                    begin
                        if(cin) begin state<=S1; qout=1'b0; end
                        else begin state<=S0; qout=1'b0; end
                    end
                S1:
                    begin
                        if(cin) begin state<=S1; qout=1'b0; end
                        else begin state<=S2; qout=1'b0; end
                    end
                S2:
                    begin
                        if(cin) begin state<=S3; qout=1'b0; end
                        else begin state<=S0; qout=1'b0; end
                    end
                S3:
                    begin
                        if(cin) begin state<=S1; qout=1'b1; end
                        else begin state<=S2; qout=1'b1; end
                    end
                default:
                    begin
                        state<=S0; qout=1'b0;
                    end
            endcase
        end
endmodule
```

4连续0或4连续1的序列检测FSM

```
module test01(clk,clr,in,out0,out1);
    input clk;              // 时钟信号clk
```

```
    input clr;              // 复位信号clr
    input in;               // 输入信号
    output reg[6:0] out0=7'b1111111;    // 4个连续0的个数
    output reg[6:0] out1=7'b1111111;    // 4个连续1的个数
    reg[6:0] zero=0;        // 计数，4个连续0的个数
    reg[6:0] one=0;         // 计数，4个连续1的个数
    reg[2:0] num0=0;        // 当前0的个数
    reg[2:0] num1=0;        // 当前1的个数

    always@(posedge clk or posedge clr)
        begin
            if(clr) begin num0<=0; num1<=0; end
            else // clr=0
                begin
                    if(in)  // 输入为1
                        begin
                            num0<=3'b000;               // 将当前0的个数清零
                            case(num1)
                                3'b000: num1<=3'b001;   // 0->1
                                3'b001: num1<=3'b010;   // 1->2
                                3'b010: num1<=3'b011;   // 2->3
                                3'b011: num1<=3'b100;   // 3->4
                                3'b100: num1<=3'b001;   // 4->1
                                default:num1<=3'b000;
                            endcase
                        end
                    else    // 输入为0
                        begin
                            num1<=3'b000;               // 将当前1的个数清零
                            case(num0)
                                3'b000: num0<=3'b001;   // 0->1
                                3'b001: num0<=3'b010;   // 1->2
                                3'b010: num0<=3'b011;   // 2->3
                                3'b011: num0<=3'b100;   // 3->4
                                3'b100: num0<=3'b001;   // 4->1
                                default:num0<=3'b000;
                            endcase
                        end
                end
        end
    always@(num0) // 此过程产生输出逻辑zero（4个连续0的个数）
        begin
            if(num0==3'b100)// 连续出现4个1
                begin zero=zero+1; dec_out(zero,out0); end
            else
                begin zero=zero; dec_out(zero,out0); end
        end
    always@(num1) // 此过程产生输出逻辑one（4个连续1的个数）
        begin
            case(num1)
                3'b100: // 连续出现4个1
                    begin one=one+1; dec_out(one,out1); end
                default:
                    begin one=one; dec_out(one,out1); end
            endcase
        end
    // 任务：七段管十进制数显示
    task dec_out;
```

```
        input integer decc;                 // 输入，十进制数
        output reg[6:0] outt;               // 输出，7位二进制数值
        if(decc==0)      outt=7'b1000000;   // 七段管显示0
        else if(decc==1) outt=7'b1111001;   // 七段管显示1
        else if(decc==2) outt=7'b0100100;   // 七段管显示2
        else if(decc==3) outt=7'b0110000;   // 七段管显示3
        else if(decc==4) outt=7'b0011001;   // 七段管显示4
        else if(decc==5) outt=7'b0010010;   // 七段管显示5
        else if(decc==6) outt=7'b0000010;   // 七段管显示6
        else if(decc==7) outt=7'b1111000;   // 七段管显示7
        else if(decc==8) outt=7'b0000000;   // 七段管显示8
        else if(decc==9) outt=7'b0011000;   // 七段管显示9
        else             outt=7'b1111111;   // 七段管不显示
    endtask
    // 函数：七段管十进制数显示
    function[6:0] outf;
        input integer decc;                 // 输入，十进制数
        if(decc==0)      outf=7'b1000000;   // 七段管显示0
        else if(decc==1) outf=7'b1111001;   // 七段管显示1
        else if(decc==2) outf=7'b0100100;   // 七段管显示2
        else if(decc==3) outf=7'b0110000;   // 七段管显示3
        else if(decc==4) outf=7'b0011001;   // 七段管显示4
        else if(decc==5) outf=7'b0010010;   // 七段管显示5
        else if(decc==6) outf=7'b0000010;   // 七段管显示6
        else if(decc==7) outf=7'b1111000;   // 七段管显示7
        else if(decc==8) outf=7'b0000000;   // 七段管显示8
        else if(decc==9) outf=7'b0011000;   // 七段管显示9
        else             outf=7'b1111111;   // 七段管不显示
    endfunction
endmodule
```

速度可控的HELLO自动循环显示

```
module test02(clk50,clr,clk1,key0,key1,out0,out1,out2,out3,out4);
    input clk50, clr;
    input key0, key1;       // 按键信号，key0=1 -> 4”移动一次
    output clk1;            // clk1：新产生的1Hz信号
    output reg[6:0] out0,out1,out2,out3,out4;
    reg [1:0] state=0;      // key按键状态，0-key0=1，1-key1=1
    reg [2:0] S=3'b000;     // 状态，'HELLO' 'ELLOH' 'LLOHE' 'LOHEL' 'OHELL'
    integer i=0;            // 延时计数用
    div_clk dc(clk50,clk1); // 模块调用，50MHz -> 1Hz

    always@(posedge clr,posedge key0,posedge key1)
        begin
            if(clr)       state=0;  // 复位
            else if(key0) state=0;  // KEY0被按下
            else if(key1) state=1;  // KEY1被按下
        end
    always @(posedge clk1,posedge clr)
        begin
            if(clr) S=3'b000;
            else
                begin
                    if(state)   // KEY1被按下(4”)
                        begin
                            if(i==3)     // 延时4秒
```

```
                                begin
                                    i=0;
                                    case(S)
                                        3'b000: S<=3'b001;
                                        3'b001: S<=3'b010;
                                        3'b010: S<=3'b011;
                                        3'b011: S<=3'b100;
                                        3'b100: S<=3'b000;
                                        default:S<=3'b000;
                                    endcase
                                end
                            else i=i+1;
                        end
                    else        // KEY0被按下(1")
                        begin
                            case(S)
                                3'b000: S<=3'b001;  // 'HELLO'->'ELLOH'
                                3'b001: S<=3'b010;  // 'ELLOH'->'LLOHE'
                                3'b010: S<=3'b011;  // 'LLOHE'->'LOHEL'
                                3'b011: S<=3'b100;  // 'LOHEL'->'OHELL'
                                3'b100: S<=3'b000;  // 'OHELL'->'HELLO'
                                default:S<=3'b000;  // 'HELLO'
                            endcase
                        end
                end
        end
    always@(S)
        begin
            case(S)
                3'b000: // 状态'HELLO'
                    begin
                        out0=7'b0001001; out1=7'b0000110;
                        out2=7'b1000111; out3=7'b1000111;
                        out4=7'b1000000;
                    end
                3'b001: // 状态'ELLOH'
                    begin
                        out0=7'b0000110; out1=7'b1000111;
                        out2=7'b1000111; out3=7'b1000000;
                        out4=7'b0001001;
                    end
                3'b010: // 状态'LLOHE'
                    begin
                        out0=7'b1000111; out1=7'b1000111;
                        out2=7'b1000000; out3=7'b0001001;
                        out4=7'b0000110;
                    end
                3'b011: // 状态'LOHEL'
                    begin
                        out0=7'b1000111; out1=7'b1000000;
                        out2=7'b0001001; out3=7'b0000110;
                        out4=7'b1000111;
                    end
                3'b100: // 状态'OHELL'
                    begin
                        out0=7'b1000000; out1=7'b0001001;
                        out2=7'b0000110; out3=7'b1000111;
                        out4=7'b1000111;
```

```
                    end
                default:
                    begin
                        out0=7'b0001001; out1=7'b0000110;
                        out2=7'b1000111; out3=7'b1000111;
                        out4=7'b1000000;
                    end
            endcase
        end

endmodule
// **分频电路模块 50MHz -> 1Hz**
module div_clk(clk50,clk1);
    input clk50;                // clk50：输入的50MHz信号
    output reg clk1=1;          // clk1:  产生的1Hz信号，赋初始值为1
    integer i=0;                // 50MHz频率下，周期计数器
    always@(posedge clk50)      // clk50上升沿触发
        begin
            if(i==25000000)     // 每过25000000个周期
                begin
                    i=0; clk1=~clk1; // clk1翻转
                end
            else i=i+1;
        end
endmodule
```

14、测试代码

> $display()、$monitor()

```
$display()  // 显示当前变量的值
$monitor()  // 用于持续监测指定变量,只要变量发生变化，就会立即显示对应的输出语句

$display("%b+%b=%b",a,b,sum);
$display("Running testbench");
$monitor($realtime,"%d %d:%d %d",out3,out2,out1,out0);
```

> $finish（结束）、$stop（暂停）

```
$finish; $stop;
```

> $random（随机数函数）

```
rand=$random%60;    //生成-59~59之间的随机数
```

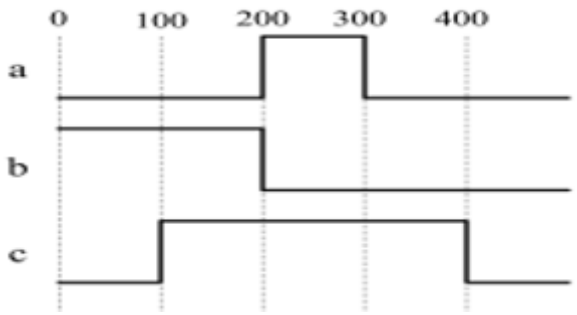
> `timescale（时间标尺定义）

```
`timescale <时间单位>/<时间精度>
// 其中时间度量的符号有：s、ms、us、ns、ps和fs
`timescale  1ns/100ps   // 时间延迟单位为1ns，时间精度为100ps
```

> **激励波形的描述**

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

```
module test1;
    reg a,b,c;
    initial
        begin
            a=0;b=1;c=0;
            #100 c=1;
            #100 a=1;b=0;
            #100 a=0;
            #100 c=0;
            #100 $stop;
        end
    initial $monitor($time,"a=%d b=%d c=%d",a,b,c); //显示
endmodule
```



时钟信号

```
reg clk50;
parameter DELAY=20; // 时钟周期
always #(DELAY/2) clk50=~clk50; // 半个周期翻转一次
initial clk50=0;    // 初始化
```

8位乘法器的仿真（组合逻辑电路）

```
module mult8(a,b,out);       // 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
endmodule
```

```
`timescale 10ns/1ns
module mult8_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(a,b,out);  // 调用测试对象

    initial              // 激励波形设定a
        begin
            a=0;
            for(i=1;i<255;i=i+1)
                #10 a=i;
        end
    initial              // 激励波形设定b
        begin
```

```
            b=0;
            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
```

8位计数器的仿真（时序逻辑电路）

```
module count8(clk,reset,qout);
    input clk,reset;
    output reg[7:0] qout;
    always@(posedge clk)
        begin
            if(reset) qout<=0;
            else qout<=qout+1;
        end
endmodule
```

```
`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

    count8 C1(clk,reset,qout);  // 调用测试对象

    parameter DELY=100;
    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
    initial $monitor($realtime,"clk=%d reset=%d qout=%d",clk,reset,qout);
endmodule
```