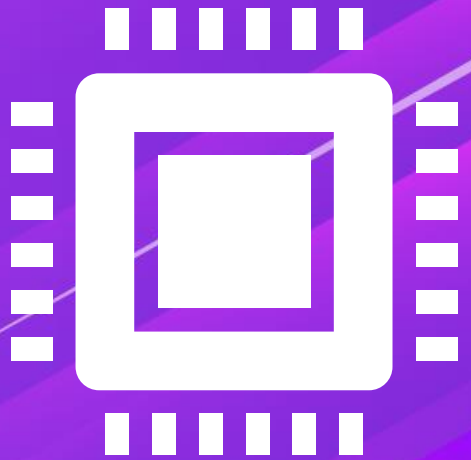
REC

EDA技术

基于Verilog HDL(B)

程序设计 思路速解



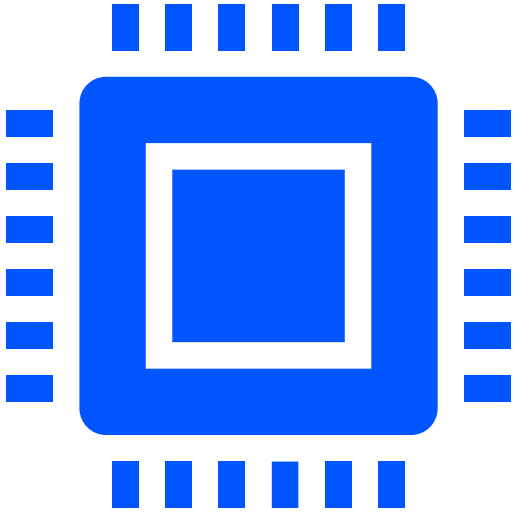
@GhostKING学长

Part 1

- 零散基础语法
- 数据选择器
- 加法器
- 乘法器
- 奇偶校验模块

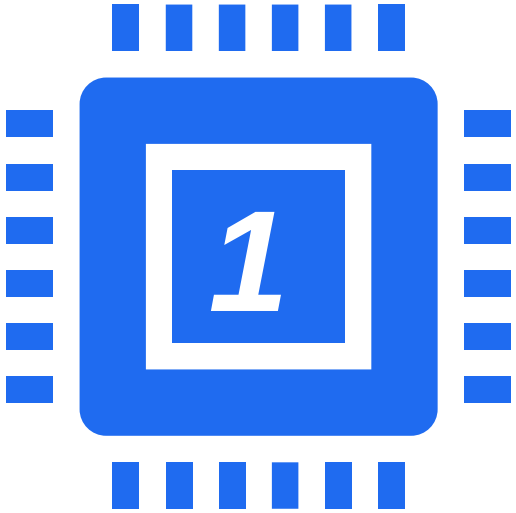
Part 2

- 计数器
- 序列检测器
- 移位寄存器
- 3-8译码器



EDA技术
基于Verilog HDL(B)
程序设计速解

@GhostKING学长



数据选择器

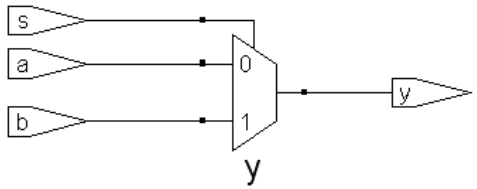
- 二选一数据选择器
  - assign
  - always-if
- 四选一数据选择器
  - assign
  - always-case
  - always-if
- 八选一数据选择器
  - always-case

输入端 | 数据输入

选择端 | 二进制选择

输出端 | 输出

Code 1 2选1数据选择器



输入端 | 2个数据输入

选择端 | 1位二进制选择

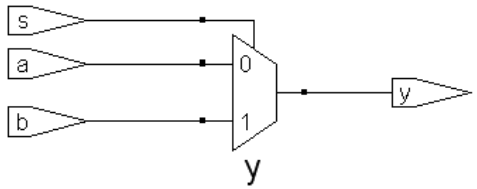
输出端 | 1个输出

设置变量

assign赋值

```
module mux2_1(a, b, s, y); //模块名、模块接口名
    input a, b, s;       // 定义输入端口
    output y;            // 定义输出端口
    /* s为0时，选择a输出；
       s为1时，选择b输出。*/
    assign y = (s == 0) ? a : b;  //输出信号
endmodule
```

Code 1 2选1数据选择器



输入端 | 2个数据输入

选择端 | 1位二进制选择

输出端 | 1个输出

设置变量

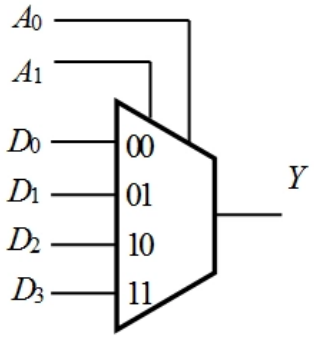
always赋值

If语句块

```
module mux2_1(a, b, s, y);
    input a, b, s;
    output y;
    reg  y; //reg 表示寄存器

    always @(a, b, s)
    begin
        if(!s) y = a;
        else y = b;
    end
endmodule
```

Code 2 4选1数据选择器



输入端 | 4个数据输入

选择端 | 2位二进制选择

输出端 | 1个输出

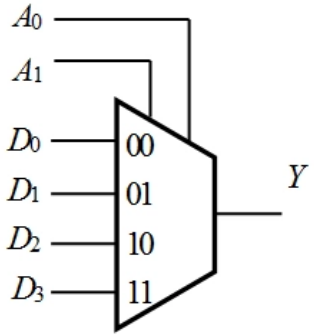
```
module mux4_1 (d0,d1,d2,d3,s0,s1,y);
  input d0,d1,d2,d3,s0,s1;
  output y;
  reg y;
  wire [1:0] SEL; //定义网线型变量用于并位s0,s1
  wire d0T, d1T, d2T, d3T; //定义中间变量，用于接收SEL触发d系列口的变化
  assign SEL={s1,s0}; //并位操作
  assign d0T = (SEL==2'D0); //当SEL十进制值为0时它为1，否则为0，下列一致
  assign d1T = (SEL==2'D1);
  assign d2T = (SEL==2'D2);
  assign d3T = (SEL==2'D3);
  assign y = (d0&d0T) | (d1&d1T) | (d2&d2T) | (d3&d3T)
//d系列端口都有信号，所以与dT系列端口相与，
//四个逻辑信号再相或，实现选择
endmodule
```

设置变量

assign赋值

核心：利用网线型变量存储遍历值

Code 2 4选1数据选择器



输入端 | 4个数据输入

选择端 | 2位二进制选择

输出端 | 1个输出

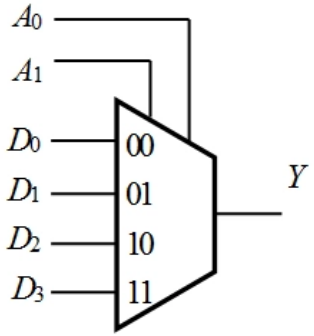
设置变量

always赋值

case语句块

```
module mux4_1 (d0,d1,d2,d3,a0,a1,y);
//定义模块和端口变量
  input d0,d1,d2,d3,a0,a1; //定义输入端
  output y; //定义输出端
  reg y; //定义寄存器存储结果
always@( d0,d1,d2,d3,a0,a1)
//载入发生信号变化的端口
  begin
   case({a1,a0})
    2'b00 : y<=d0;
    2'b01 : y<=d1;
    2'b10 : y<=d2;
    2'b11 : y<=d3;
     default : y<=d0; //可以不写，默认位
    endcase
  end
endmodule
```

Code 2 4选1数据选择器



输入端 | 4个数据输入

选择端 | 2位二进制选择

输出端 | 1个输出

```
module mux4_1 (d0,d1,d2,d3,a0,a1,y); //定义模块和端口变量
  input d0,d1,d2,d3,a0,a1; //定义输入端
  output y; //定义输出端
reg y; //定义寄存器存储结果
always@(d0,d1,d2,d3,a0,a1)
// 也可以用并位简化，就不需要if嵌套，与C语言相同
  if(!s0)
  begin
      if(!s1)
      y <= d0;
      else
      y <= d1;
  end
  else begin
      if(!s1)
      y <= d3;
      else
      y <= d4;
  end
endmodule
```

设置变量

always赋值

If语句块

Code 3 8选1数据选择器

输入端 | 8个数据输入

选择端 | 3位二进制选择

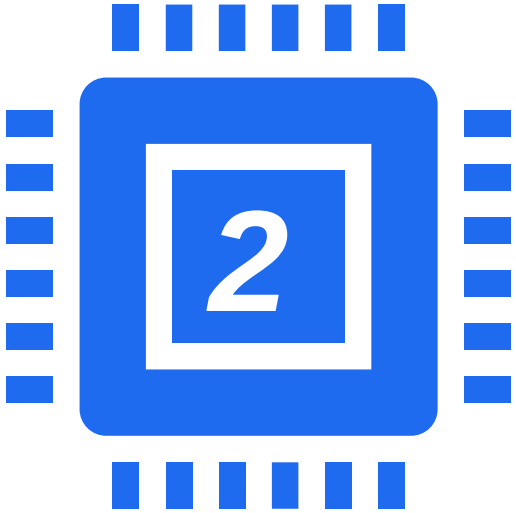
输出端 | 1个输出

```
module Eignth_Select(out,in0,in1,in2,in3,in4,in5,in6,in7,sel);
        output out;
        input in0,in1,in2,in3,in4,in5,in6,in7;
        input[2:0] sel;
        reg out;
        always @(in0,in1,in2, in3 ,in4 , in5 , in6 , in7, sel)
          case(sel)   //根据sel的不同进行选择
            3'b000: out=in0;
            3'b001: out=in1;
            3'b010: out=in2;
            3'b011: out=in3;
            3'b100: out=in4;
            3'b101: out=in5;
            3'b110: out=in6;
            3'b111: out=in7;
            default: out=1'bx;
        endcase
endmodule
```

设置变量

always赋值

case语句块



加法器

- 半加器
- 全加器
  - 非例化
  - 半加器例化
- 八位二进制加法器
  - 非例化的全加器例化
  - 非例化
  - 四位二进制加法器

加数 | 数据输入

结果 | 相加结果

进位 | 进位输入/进位输出

Code 1 半加器(1位二进制加法器)

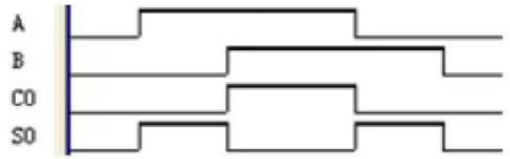


图 3-3　半加器的仿真功能波形图

加数 | 两加数A，B

结果 | 相加结果S0

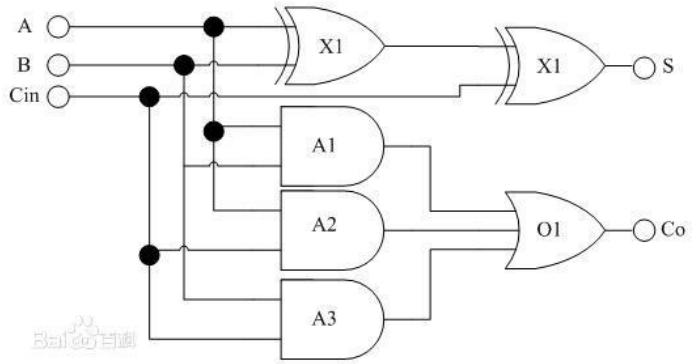
进位 | 进位输出C0

设置变量

assign赋值

```
module half_adder(a,b,S0,C0);
//注意这个模块名，后面例化要用
  input a,b;
  output S0,C0;
  assign S0=A^B; //异或运算
  assign C0=A&B;
endmodule
```

Code 2 全加器(非例化)



加数 | 两加数A，B

结果 | 相加结果S0

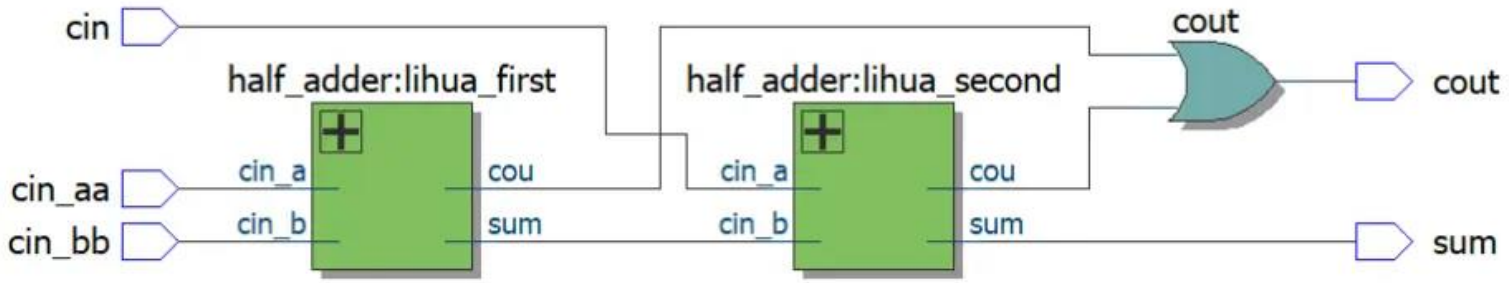
进位 | 进位输入Cin / 进位输出C0

设置变量

always赋值

```
module full_adder(
        input a,b,cin,
        output reg cout,sum );
reg s1,s2,s3; //最后用于判断是否有进位输出
always @(a,b,cin)
begin
        sum=(a^b)^cin;//本位和输出表达式
        s1=a&cin;
        s2=b&cin;
        s3=a&b;
        cout=(s1|s2)|s3;//高位进位输出表达式
end
endmodule
```

Code 2 全加器(半加器例化)

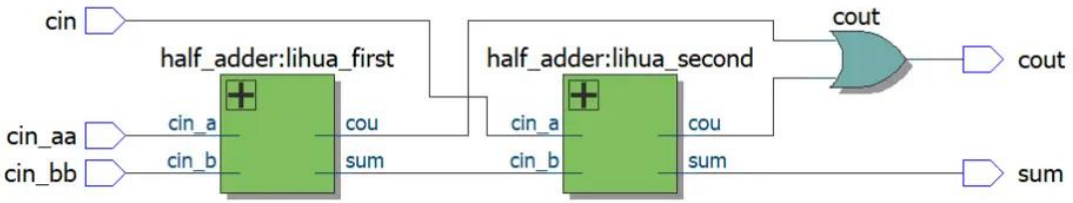


加数 | 两加数A，B

结果 | 相加结果S0

进位 | 进位输入Cin / 进位输出C0

Code 2 全加器(半加器例化)




```
module full_adder(ainF,binF,cin,cout,sum);
  input ainF,binF,cin;
  output cout,sum;
  reg cout,sum;

//定义网线型变量
  wire first_cout, second_cout; //两个半加器的进位输出
  wire first_sum; //第一个半加器的求和输出
```

```
//调用半加器例化模块，端口配对
half_adder first_adder(
    .a(ainF),
    .b(binF),
    .C0(first_cout),
    .S0(first_sum) );

half_adder second_adder(
    .a(cin),
    .b(first_sum),
    .C0(second_cout),
    .S0(sum) );

assign cout = (first_cout | second_cout);

endmodule
```

Code 3 八位加法器(非例化全加器例化)

加数 | 两加数A，B

结果 | 相加结果S0

进位 | 进位输入Cin / 进位输出C0

```
module adder_8bit(a,b,cin,sum,cout);
    input[7:0] a,b; //定义8位的输入数
    input cin;
    output[7:0] sum;
    output cout;
    wire c1,c2,c3,c4,c5,c6,c7;
//每一位依次经过了全加器的进位输出

    full_adder u1(a[0],b[0],cin, c1,sum[0]);
//第一位的cin为当前的进位输入
    full_adder u2(a[1],b[1],c1,c2,sum[1]);
//前一位的进位输出作为后一位的进位输入
    full_adder u3(a[2],b[2],c2,c3,sum[2]);
    full_adder u4(a[3],b[3],c3,c4,sum[3]);
    full_adder u5(a[4],b[4],c4,c5,sum[4]);
    full_adder u6(a[5],b[5],c5,c6,sum[5]);
    full_adder u7(a[6],b[6],c6,c7,sum[6]);
    full_adder u8(a[7],b[7],c7,cout,sum[7]);
//最终的进位输出为当前进位输入

endmodule
```

Code 3 八位加法器(非例化)

加数 | 两加数A，B

结果 | 相加结果S0

进位 | 进位输入Cin / 进位输出C0

```
Module Adder_8bit (A, B, Cin, C0, Out)
    output[7:0] Out;
    output C0;
    input[7:0] A, B;
    input Cin;
    wire[8:0] DATA;
    //定义中间变量存储结果值
    assign DATA = A + B + Cin;
    assign C0 = DATA[8];
    assign Out = DATA[7:0];
endmodule
```

Code 4 四位加法器(非例化全加器例化)

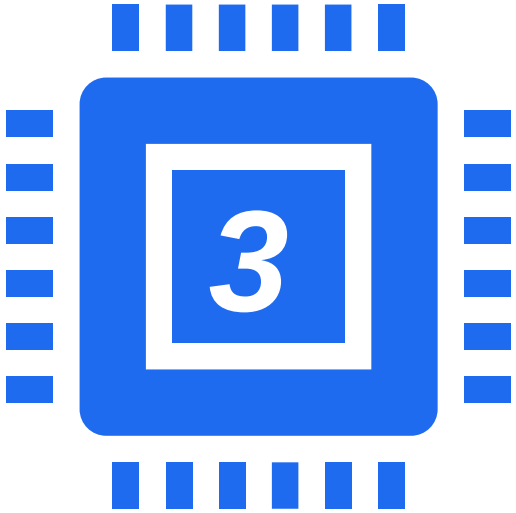
加数 | 两加数A，B

结果 | 相加结果S0

进位 | 进位输入Cin / 进位输出C0

```
module adder_4bit(a,b,cin,sum,cout);
    input[3:0] a,b; //定义4位的输入数
    input cin;
    output[3:0] sum;
    output cout;
    wire c1,c2,c3; //每一位依次经过了全加器的进位输出

    full_adder u1(a[0],b[0],cin, c1,sum[0]);
//第一位的cin为当前的进位输入
    full_adder u2(a[1],b[1],c1,c2,sum[1]);
//前一位的进位输出作为后一位的进位输入
    full_adder u3(a[2],b[2],c2,c3,sum[2]);
    full_adder u4(a[3],b[3],c3,cout,sum[3]);
//最终的进位输出为当前进位输入
endmodule
```



乘法器

- 4位乘法器
  - for增值法
  - for减值法
- 2位乘法器

乘数 | 两数相乘

乘积 | 运算结果

循环 | 循环位移

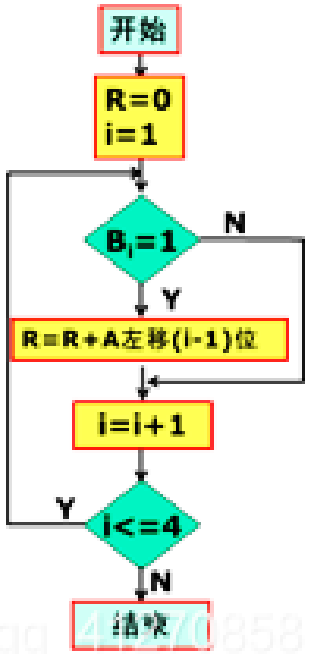
Code 1 4位乘法器(for增值)

1001*1011

```
   1001
  *1011
  ------
   1001
  1001
 0000
1001
------
1100011
```

设A为被乘数，
B为乘数
$A=A_4A_3A_2A_1$
$B=B_4B_3B_2B_1$
R为乘积

R既是部分积也是最终乘积，循环中是部分积，循环结束是最终乘积。



乘数 | 两数相乘

乘积 | 运算结果

循环 | 循环位移

```
module MULT4B(R,A,B);  //4位乘法器
parameter  S  =  4;   //参数定义关键词parameter
（将常数用字符表示称为参数）
input [S:1] A,B; //乘数
output [2*S:1] R; //乘积
integer i; //i为循环变量
reg [2*S:1] R;
always @ (A,B)
   begin
      R = 0;
      for(i=1;i<=S;i=i+1) //循环4次
         if(B[i]) R = R + (A<<(i-1));
//被乘数左移，与部分积相加
         else R = R;
   end
endmodule
```

Code 1 4位乘法器(for减值)

乘数 | 两数相乘

乘积 | 运算结果

循环 | 循环位移

```
module MULT4B(R,A,B);
parameter S = 4;
input [S:1] A,B;
output [2*S:1] R;
reg [2*S:1] TA,R;//TA为A的2S位扩展
reg [S:1] TB,TC;
always @ (A , B)
   begin
      R = 0;
      TA = {{S{1'b0}},A};//将A扩展成2S位
      TB = B;
      TC=S //换成integer i=4也行
      for(TC=S; TC>0 ; TC=TC-1)
         begin
            if(TB[1]) R = R + TA;
            //如果乘数右移后的最低位为1
            else R = R;
            TA = TA << 1;//被乘数左移
            TB = TB >> 1;//乘数右移
         end
   end
endmodule
```

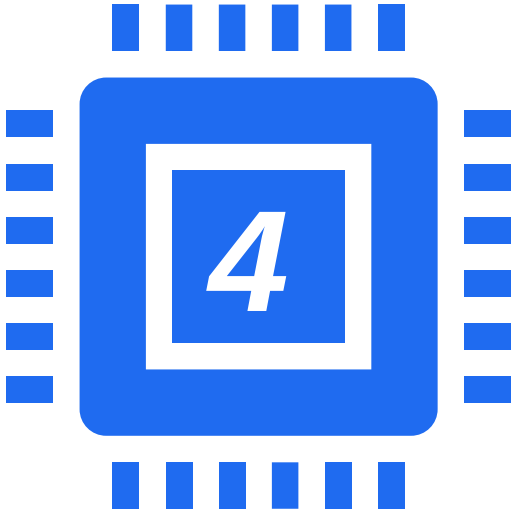
Code 2 2位乘法器(for增值)

乘数 | 两数相乘

乘积 | 运算结果

循环 | 循环位移

```
module MULT2B(R,A,B); //2位乘法器
parameter S = 2;  //参数定义关键词parameter
（将常数用字符表示称为参数）
input [S:1] A,B; //乘数
output [2*S:1] R; //乘积
integer i; //i为循环变量
reg [2*S:1] R;
always @ (A,B)
   begin
      R = 0;
      for(i=1;i<=S;i=i+1) //循环2次
         if(B[i]) R = R + (A<<(i-1));
//被乘数左移，与部分积相加
         else R = R;
   end
endmodule
```



奇偶校验模块

数据输入 | 数据按位异或

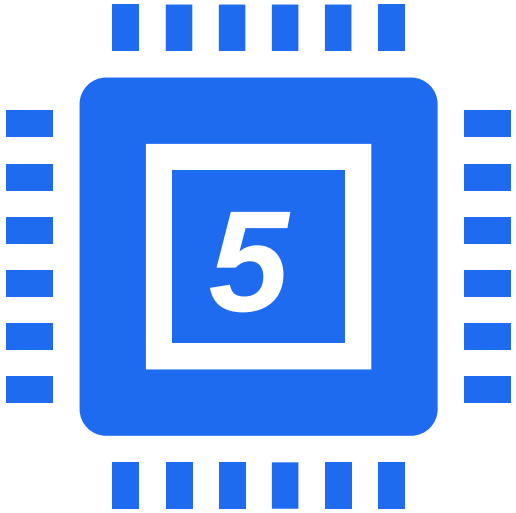
校验输出 | 1输出奇校验，0输出偶校验

```
module odd_sel(
input [31:0] bus,
input sel,
output check
);

/* 若没有说是几位的
input [width-1:0] bus;
parameter width = 8; //设置位数
*/

wire odd;
assign odd = ^bus;
assign check = sel?odd:~odd;

endmodule
```



计数器

- 十进制计数器
  - 无使能同步清零
  - 使能同步清零
  - 使能可预置异步清零
- 60进制计数器
  - 例化
  - 非例化
- 100进制计数器
  - 例化
  - 999计数器

时钟域 | 时钟信号/时钟复位

预置域 | 预置数据

输出端 | 输出/进位标识

控制端 | 使能/数据控制

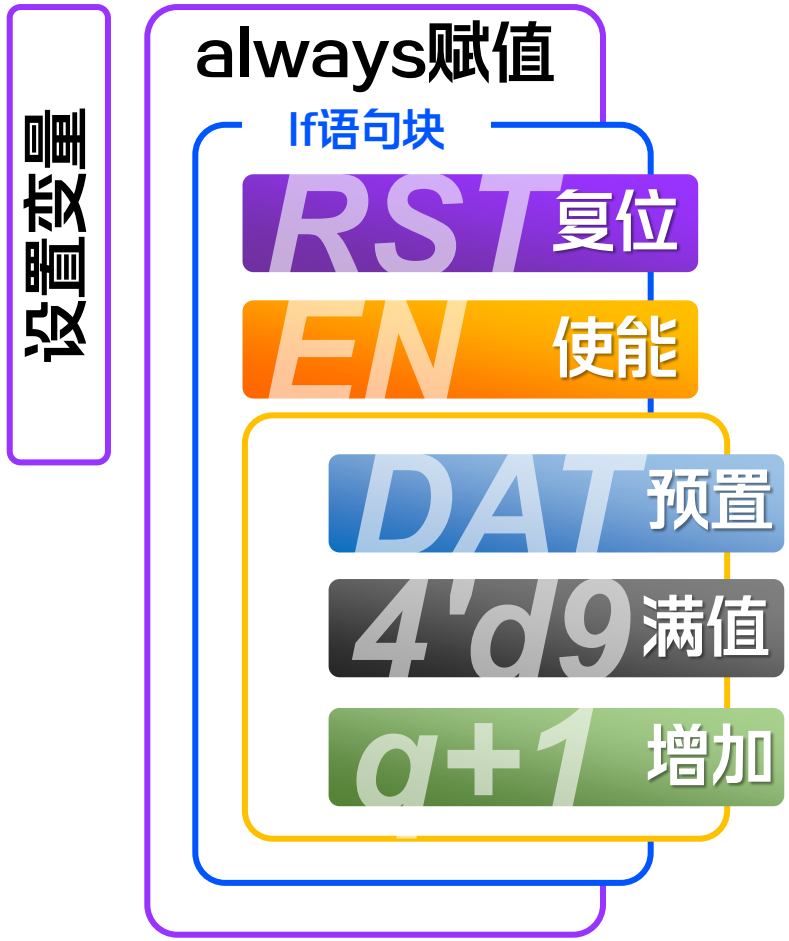
Code 1 10进制计数器(无使能同步)


```
module CNT10(clk,rst,q,cout);
input clk,rst;
output cout;
output [3:0] q; //一枚十进制数需要4个位宽(9为1001)
reg [3:0] q;
always@(posedge clk) //上升沿触发，下降沿为negedge
begin
    if(!rst)
    q <= 0;
  else if(q >= 4'd9)
    q <= 0;
  else
    q <= q + 1;
end
assign cout = (q == 4‘d9 )? 1’b1: 1‘b0; //判断该位是否有进位
endmodule
```

Code 1 10进制计数器(使能同步)


```
module CNT10(clk, clr, en, q, cout);
//CLR是芯片的复位输入，同RES
input clk,clr,en;
output cout;
output[3:0] q;
reg[3:0] q;
always@(posedge clk)
    begin
    if(clr) q <= 0;
    else if (en)
        begin
         if(q == 4'd9) q <= 0;
         else q <= q + 1;
         end
    end
      assign cout = (q == 4'd9 )? 1'b1: 1'b0;
endmodule
```

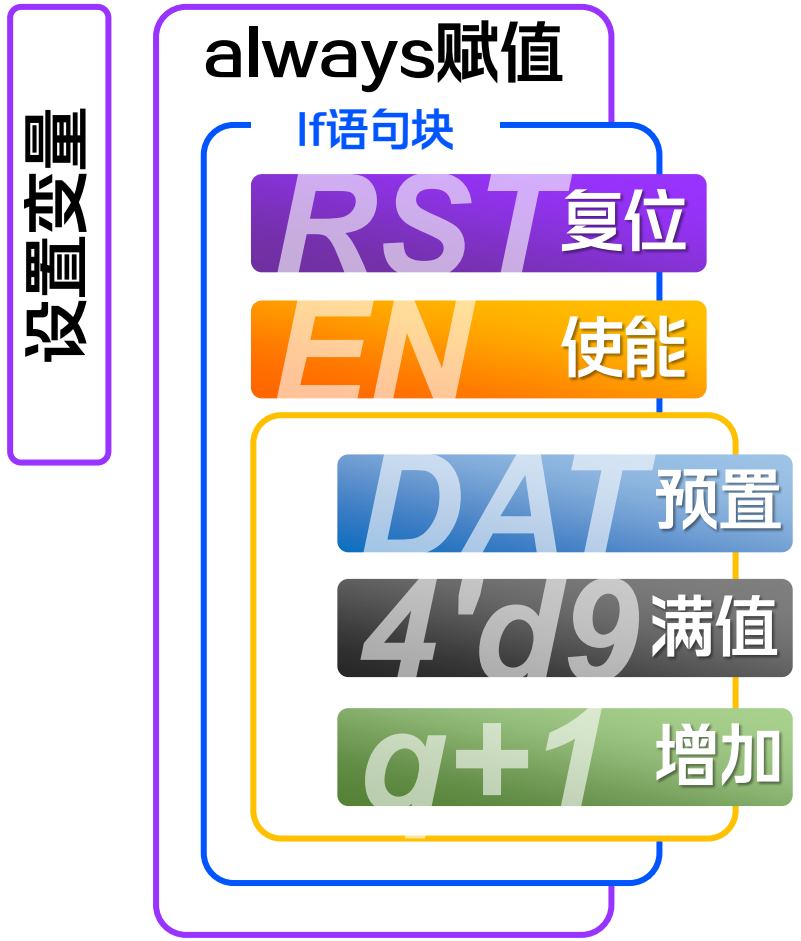
Code 1 10进制计数器(使能预置异步)



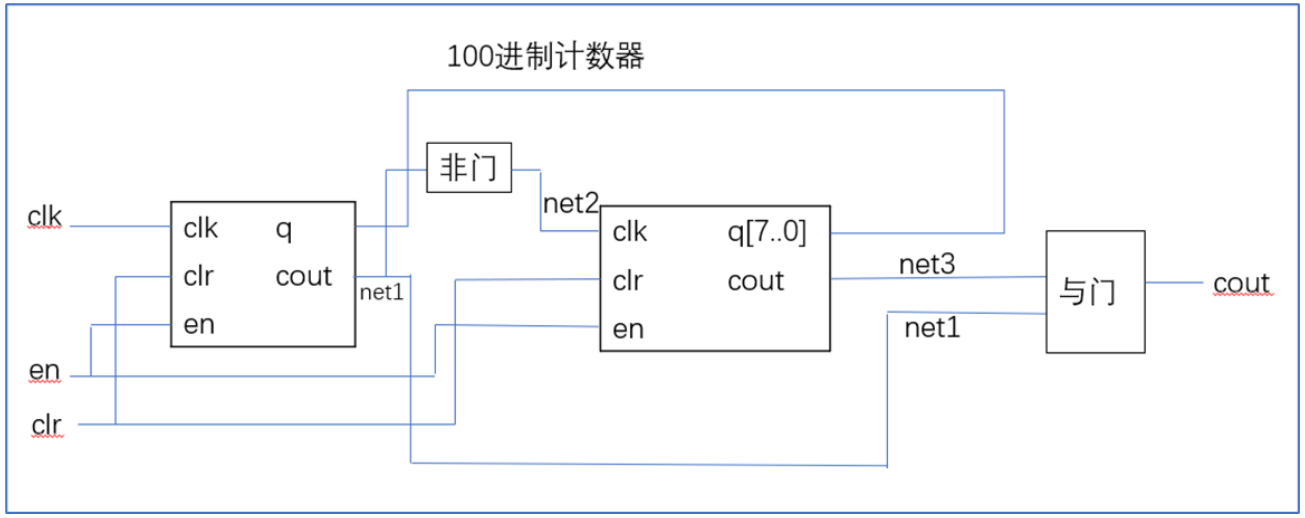

```
module CNT10 (CLK, RST, EN, LOAD, COUT, DOUT, DATA);
  input CLK, EN, RST, LOAD; //时钟，时钟使能，复位，数据加载控制信号
  input [3: 0] DATA;              //4位并行加载数据
  output [3: 0] DOUT;             //4位计数输出
  output COUT;  //进位
reg [3: 0] Q1;  //计数数值
reg  COUT;
  assign  DOUT=Q1;                  //将内部寄存器的计数结果输出至DOUT
  always @ (posedge CLK , negedge RST)        //时序过程
   begin
    if (!RST)  Q1<=0;     //RST=0时，对内部寄存器单元异步清0，看题
    else if (EN)
begin        //同步使能EN=1，则允许加载或计数
                if (!LOAD)  Q1<=DATA;        //当LOAD=0,向内部寄存器加载数据
       else if (Q1<9)  Q1<=Q1+1;   //当Q1小于9时，允许累加
       else Q1<=4`b0000;
end     //否则一个时钟后清0返回初值
    end
   assign COUT = (q == 4'd9 )? 1'b1: 1'b0;
endmodule
```


设置变量
always赋值
If语句块
RST 复位
4'd9 满值
a+1 增加


设置变量
always赋值
If语句块
CLR 复位
EN 使能
4'd9 满值
a+1 增加


设置变量
always赋值
If语句块
RST 复位
EN 使能
DAT 预置
4'd9 满值
a+1 增加

Code 2 60进制计数器(例化)




```
module CNT60(clk, clr, en, q1, q0, cout);
  input clk;
  input clr;
  input en;
  output[3:0] q1;  //高四位，十位
  output[3:0] q0;  //低四位，个位
  output cout;
wire net1, net2, net3; //1是个位的进位，2是1的非，3是十位的进位
  CNT10 U1(.clk(clk), .clr(clr), .en(en), .q(q0), .cout(net1));
    not U2(net2, net1);  //非门（输出，输入）
  CNT6 U3(.clk(net2), .clr(clr), .en(en), .q(q1), .cout(net3));
    and U4(cout, net1, net3); //与门
endmodule
```

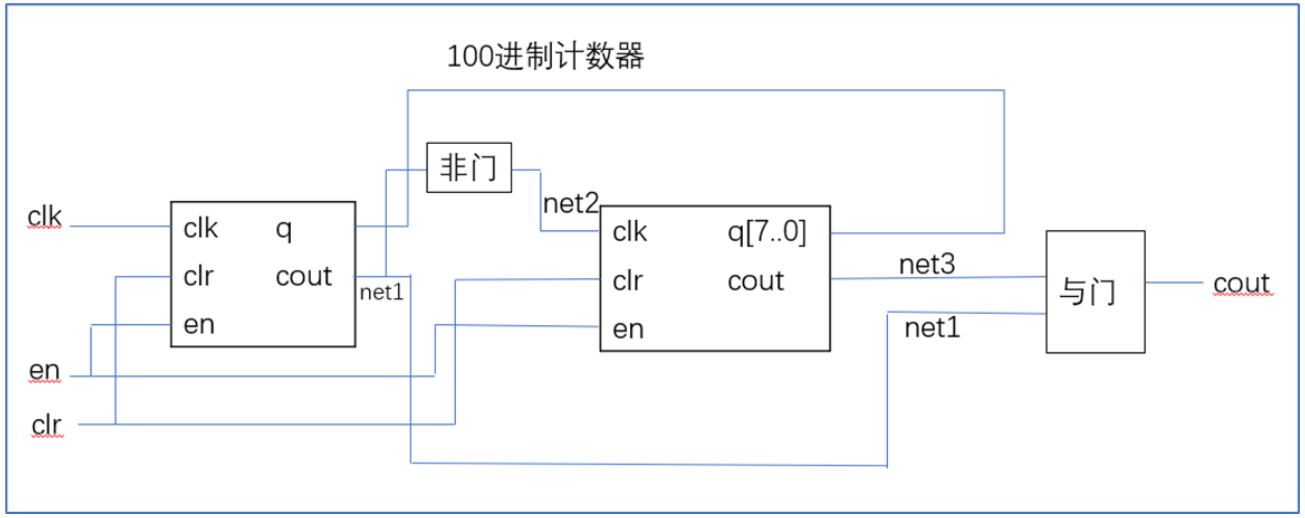
Code 2 60进制计数器(使能异步非例化)


```
always @(posedge in_clk,negedge rest)
begin
  if(!rest)//异步清零
  begin
  out_munb <= 6'b0;
  end
else
  begin
  if(in_en)
      begin
      if(out_munb== 6‘d59)
      begin
       out_munb <= 6'b0;
       end
       else
       begin
        out_munb <= out_munb + 6'b1;
      end
    end
  end
end
endmodule
```

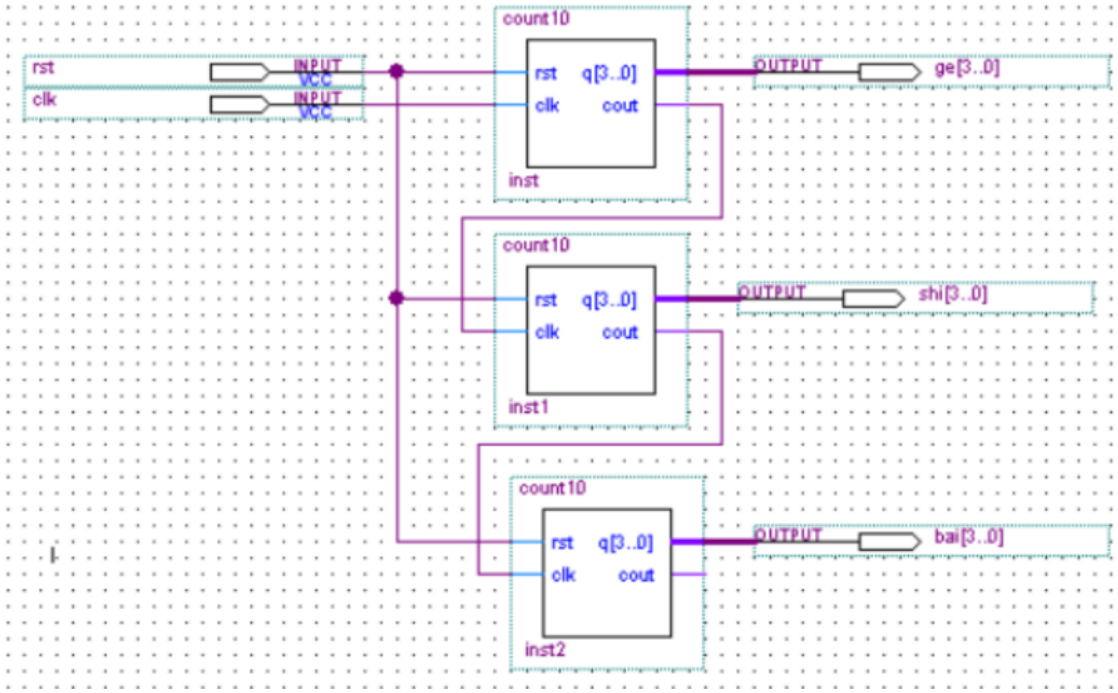
Code 2 60进制计数器(使能异步非例化)

```
module counter60(in_en,in_clk,rest,out_munb);
input in_clk;//输入时钟
input rest;//清零
input in_en;//使能计数
output [5:0] out_munb;//输出计数
reg [5:0] out_munb;
always @(posedge in_clk,negedge rest)
begin
  if(!rest)//异步清零
  begin
  out_munb <= 6'b0;
  end
else
  begin
  if(in_en)
      begin
      if(out_munb== 6‘d59)
      begin
       out_munb <= 6'b0;
       end
       else
       begin
        out_munb <= out_munb + 6'b1;
      end
    end
  end
end
endmodule
```

Code 3 100进制计数器(例化)

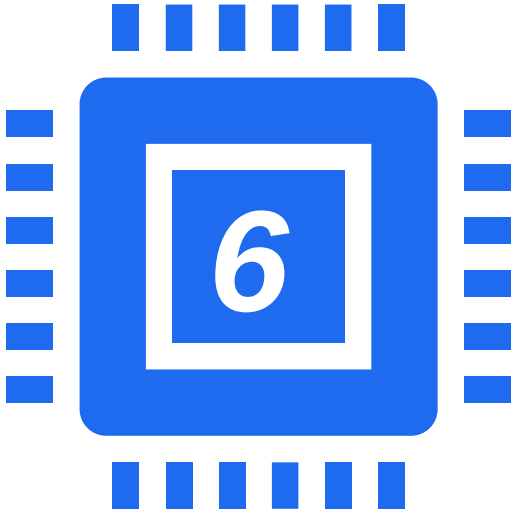



```
module CNT100(clk, clr, en, q1, q0, cout);
input clk;
input clr;
input en;
output[3:0] q1;  //高四位，十位
output[3:0] q0;  //低四位，个位
output cout;
wire net1, net2, net3; //1是个位的进位，2是1的非，3是十位的进位
CNT10 U1(.clk(clk), .clr(clr), .en(en), .q(q0), .cout(net1));
not U2(net2, net1);  //非门（输出，输入）
//这里默认低电平触发clk，所以要用非门，否则直接连接，net2就是q0
CNT10 U3(.clk(net2), .clr(clr), .en(en), .q(q1), .cout(net3));
and U4(cout, net1, net3); //与门（输出，输入，输入…）
endmodule
```



Code 4 999(1000)进制计数器(例化)

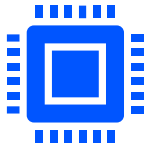
```
module CNT1000(clk, clr, en, q2,q1, q0, cout);
input clk;
input clr;
input en;
output[3:0] q2;  //百位
output[3:0] q1;  //十位
output[3:0] q0;  //个位
output cout;
wire net1, net2, net3,net4,net5;
//1是个位的进位，2是1的非，3是十位的进位，4是2的非，5是百的进位
CNT10 U1(.clk(clk), .clr(clr), .en(en), .q(q0), .cout(net1));
not U2(net2, net1);  //非门（输出，输入）
//这里默认低电平触发clk，所以要用非门，否则直接连接，net2就是q0
CNT10 U3(.clk(net2), .clr(clr), .en(en), .q(q1), .cout(net3));
not U4(net4, net2);
CNT10 U5(.clk(net4), .clr(clr), .en(en) .q(q2), .cout(net5));
and(cout,net1,net3,net5); //与门（输出，输入，输入…）
endmodule
```



序列检测器

参数 | 状态参数

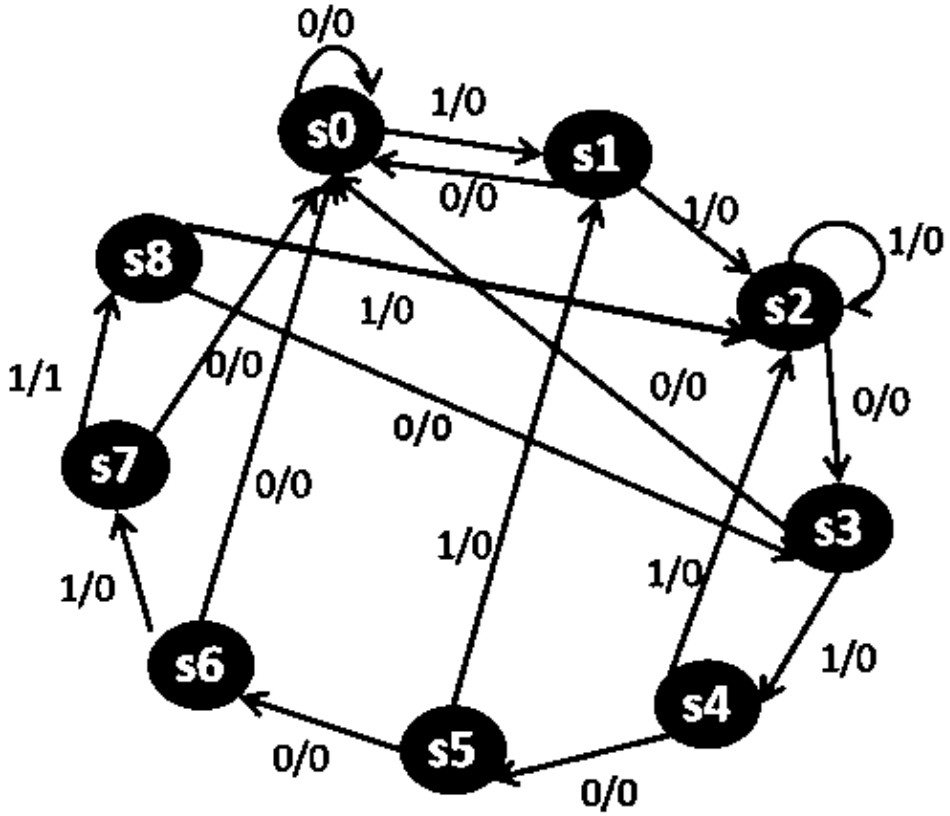
变量 | 现态/次态



程序设计速解(上)

@GhostKING学长

		1	1	0	1	0	0	1	1
	s0	s1	s2	s3	s4	s5	s6	s7	s8
Din:	0	1	1						
	0	0							
	0	1	0						
	0	1	1	1					



Code 序列检测器

状态参数

状态变量

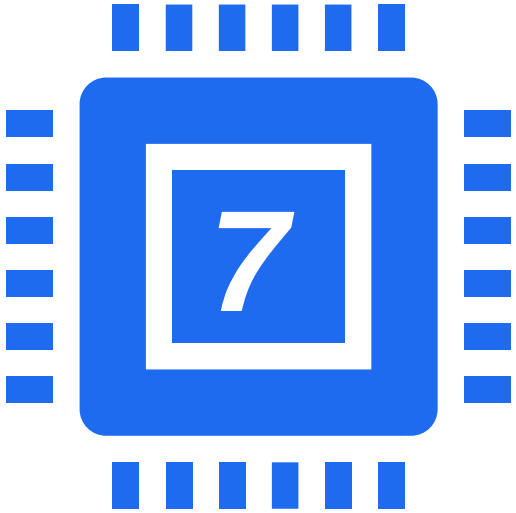
触发条件

RST 复位

序列传递

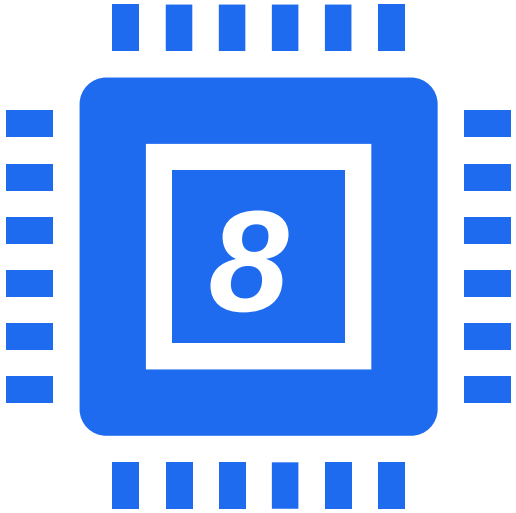
结果传出

```
module SCHK(input CLK,DIN,RST,output SOUT);
parameter s0=40,s1=41,s2=42, s3=43, s4=44,s5=45, s6=46,s7=47,s8=48;
reg[8:0] ST, NST; //设定现态变量和次态变量
always @(posedge CLK or posedge RST)
if (RST) ST<=s0;
else ST<=NST;
always @(ST,DIN)
begin
case(ST)  //11010011串行输人，高位在前
s0 : if (DIN==1'b1) NST<=s1; else NST<=s0;
s1 : if (DIN==1'b1) NST<=s2; else NST<=s0;
s2 : if (DIN==1'b0) NST<=s3; else NST<=s2;
s3 : if (DIN==1'b1) NST<=s4; else NST<=s0;
s4 : if (DIN==1'b0) NST<=s5; else NST<=s2;
s5 : if (DIN==1'b0) NST<=s6; else NST<=s1;
s6 : if (DIN==1'b1) NST<=s7; else NST<=s0;
s7 : if (DIN==1'b1) NST<=s8; else NST<=s0;
s8 : if (DIN==1'b0) NST<=s3; else NST<=s2;
default : NST<=s0;  //写回s0
endcase
end
assign SOUT=(ST==s8);
endmodule
```



移位寄存器

```
module Regist(clk,load,din,qb);
  input clk,load; //时钟信号，移位信号
input[7:0]din; //8位数据
output qb;
reg[7:0] r; //用于运算
always@(posedge clk)
  if(load) r<=din;
else r[6:0]<=r[7:1]; //位移信号为低电平时执行位移
assign qb=r[0]; //检查移位
endmodule
```



3-8译码器

不含控制器

```
module decoder(
        input wire [2:0] a,//输入信号，3位
        output reg [7:0] b//输出信号，8位
);
        //译码器组合逻辑
        always@(a)begin
                case(a)
                        3'b000: b=8'b11111110;
                        3'b001: b=8'b11111101;
                        3'b010: b=8'b11111011;
                        3'b011:b=8'b11110111;
                        3'b100: b=8'b11101111;
                        3'b101:b=8'b11011111;
                        3'b110:b=8'b10111111;
                        3'b111:b=8'b01111111;
                        default: b=8'b00000000;
                endcase
        end
endmodule
```

Code 3-8译码器(使能译码控制)

```
module yi_ma_qi138(INA,Y_01, EN,Y_DOUT);
input [2:0] INA;//三位  输入
input Y_01;//输出有效 电平控制
input EN;//使能控制端
output [7:0] Y_DOUT;//输出
reg [7:0] Y_DOUT;//输出
always @(INA)
begin
if(EN)
begin
if(Y_01==0）//译码电平 0（否则是1）
begin
case(INA)
3'b000: Y_DOUT = 8'b1111_1110;
3'b001: Y_DOUT = 8'b1111_1101;
3'b010: Y_DOUT = 8'b1111_1011;
3'b011: Y_DOUT = 8'b1111_0111;
3'b100: Y_DOUT = 8'b1110_1111;
3'b101: Y_DOUT = 8'b1101_1111;
3'b110: Y_DOUT = 8'b1011_1111;
3'b111: Y_DOUT = 8'b0111_1111;
endcase
end
else //译码电平 1
begin
case(INA)
3'b000: Y_DOUT = 8'b0000_0001;
3'b001: Y_DOUT = 8'b0000_0010;
3'b010: Y_DOUT = 8'b0000_0100;
3'b011: Y_DOUT = 8'b0000_1000;
3'b100: Y_DOUT = 8'b0001_0000;
3'b101: Y_DOUT = 8'b0010_0000;
3'b110: Y_DOUT = 8'b0100_0000;
3'b111: Y_DOUT = 8'b1000_0000;
endcase
end
end
end
endmodule
```