5.5.3 verilog 语言的描述风格

三种描述风格：

- **行为描述：always语句，使用功能描述**
- **数据流描述：assign语句 ,使用布尔代数式描述**
- **结构化描述：元件例化**

5.5.4 基本逻辑电路的VHDL设计

一、组合电路

原则1：在always中用到的所有输入信号都出现在敏感信号列表中；

原则2：电路的真值表必须在代码中完整的反映出来（使用default），否则会生成锁存器；

原则3：使用if・・ else・・，必须完全互斥；

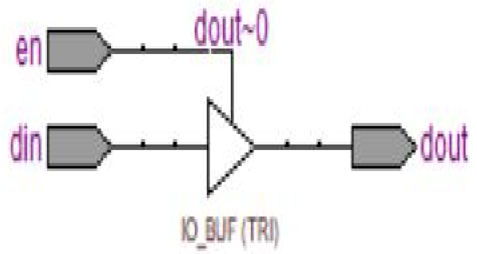
原则4：不能在一个表达式中对某个信号即读又写（不能反馈）；

原则5：不能使用 <= 非阻塞赋值。

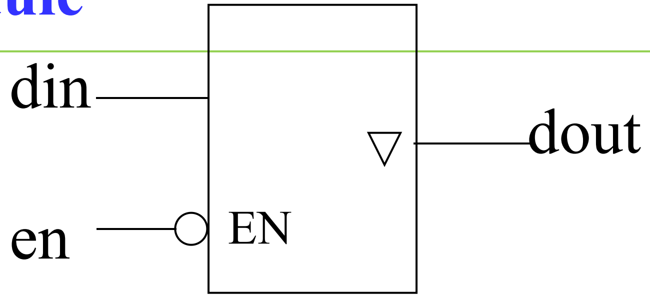
1. 三态门及总线缓冲器

```
module tri  (din , en , dout ) ;

 input din , en ;
output dout ;

 assign dout = en ? din: 1’bz;

endmodule
```

```
module tri  (din , en , dout ) ;

 input din , en ;
 output dout ;
 reg dout ;

 always @ (en , din)
   begin
     if ( !en)
         dout = din ;
     else
         dout = 1'bz ;
   end
endmodule
```



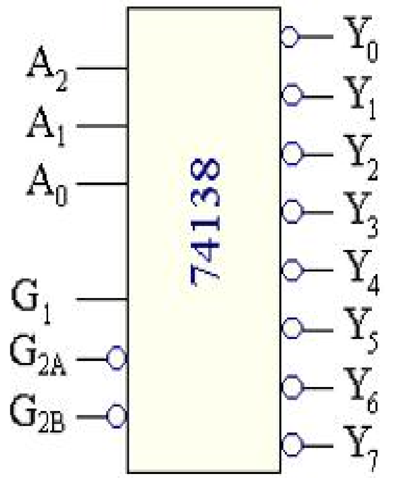
8位数据总线?

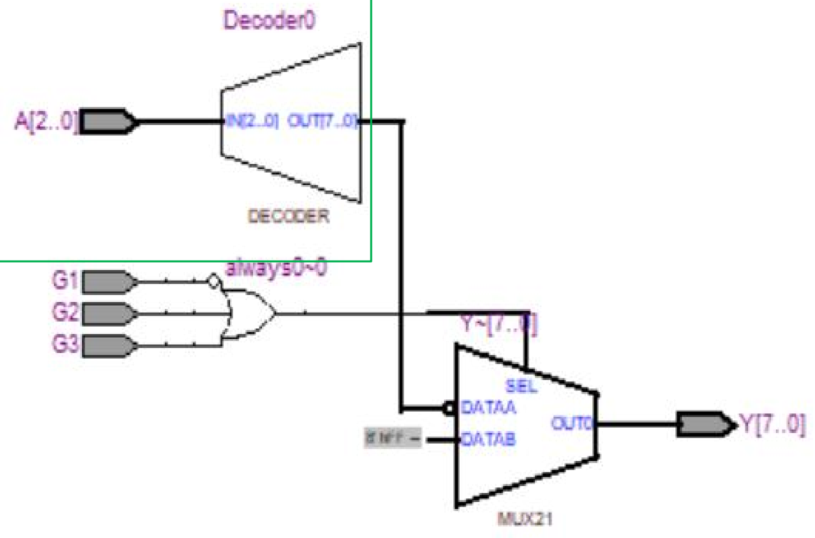


2. 3：8译码器

```
module decode3_8( A , G1,  G2 ,G3 , Y  );
input G1, G2, G3;
input [2:0] A ;
output [7:0]  Y ;
reg [7:0] Y ;
reg s;

always @ (  * )
   begin
      s = G2 | G3 ;
       if (G1 == 0 || s== 1)
           Y = 8'b1111_1111;
        else
           begin
                case ( A )
                     3'b000: Y = 8'b1111_1110;
                     3'b001: Y = 8'b1111_1101;
                     3'b010: Y = 8'b1111_1011;
                     3'b011: Y = 8'b1111_0111;
                     3'b100: Y = 8'b1110_1111;
                     3'b101: Y = 8'b1101_1111;
                     3'b110: Y = 8'b1011_1111;
                     3'b111: Y = 8'b0111_1111;
                     default:Y = 8'b1111_1111;
                endcase
           end
   end
endmodule
```

使能端 if

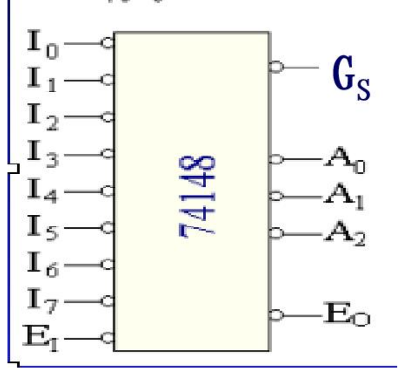


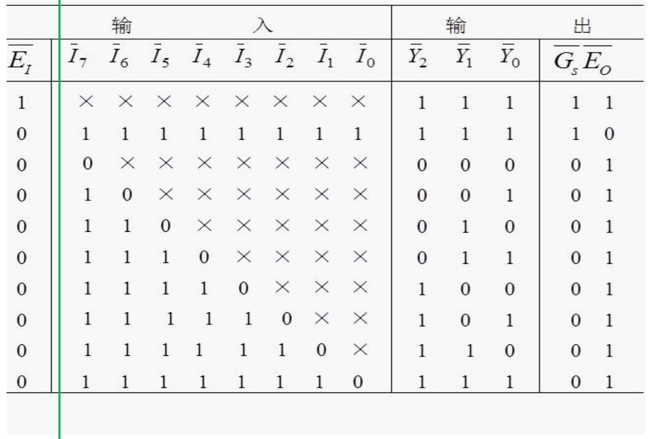


3.　8-3优先编码器

```
module encode_83( I ,EI ,A, GS ,EO );

input [7:0] I ;
input EI ;
output [2:0] A ;
output  GS , EO ;
reg [2:0]  A ;
reg GS, EO ;

always @ ( * )
  begin
    if  (EI )
     begin
       A = 3'b111 ;
       GS = 1’b1 ;
       EO = 1’b1 ;
     end
```

```
    else if (I[7] == 0)
       begin
          A = 3'b000;
         GS = 1’b0;
         EO = 1’b1 ;
       end
......................................

    else if (I[0] == 0)
       begin
         A = 3'b111;
        GS =1’b 0;
         EO = 1’b1;
       end
     else
       begin
         A = 3'b111;
         GS = 1’b1;
         EO = 1’b0;
       end
    end

endmodule
```

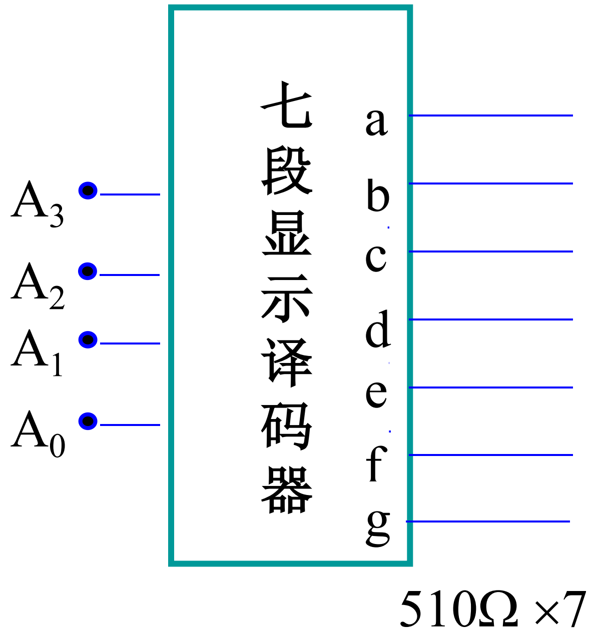


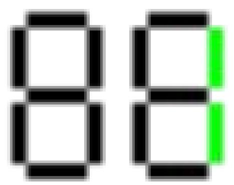


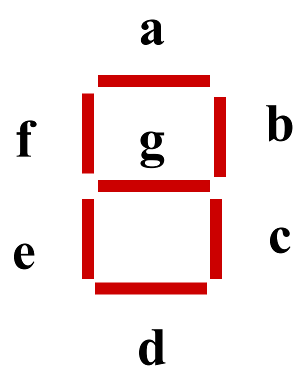
输				入					输		出	
$\overline{E_I}$	$\bar{I}_7$	$\bar{I}_6$	$\bar{I}_5$	$\bar{I}_4$	$\bar{I}_3$	$\bar{I}_2$	$\bar{I}_1$	$\bar{I}_0$	$\bar{Y}_2$	$\bar{Y}_1$	$\bar{Y}_0$	$\overline{G_s}\,\overline{E_O}$
1	×	×	×	×	×	×	×	×	1	1	1	1 1
0	1	1	1	1	1	1	1	1	1	1	1	1 0
0	0	×	×	×	×	×	×	×	0	0	0	0 1
0	1	0	×	×	×	×	×	×	0	0	1	0 1
0	1	1	0	×	×	×	×	×	0	1	0	0 1
0	1	1	1	0	×	×	×	×	0	1	1	0 1
0	1	1	1	1	0	×	×	×	1	0	0	0 1
0	1	1	1	1	1	0	×	×	1	0	1	0 1
0	1	1	1	1	1	1	0	×	1	1	0	0 1
0	1	1	1	1	1	1	1	0	1	1	1	0 1

4. 七段显示译码器

```
module seg_7(  A, seg );

input [3:0] A ;
output [6:0]  seg ;
reg [6:0]  seg ;

always @ (  *  )
   begin
     case (A)
        4'd0:  seg = 7'b111_1110 ;
        4'd1:  seg = 7'b011_0000 ;
        4'd2:  seg = 7'b110_1101 ;
        4'd3:  seg = 7'b011_0000 ;
        4'd4:  seg = 7'b011_0011 ;
        4'd5:  seg = 7'b101_1011 ;
        4'd6:  seg = 7'b101_1111 ;
        4'd7:  seg = 7'b111_0000 ;
        4'd8:  seg = 7'b111_1111 ;
        4'd9:  seg = 7'b111_1011 ;
        default:  seg = 7'b000_0001 ;
     endcase
end

endmodule
```





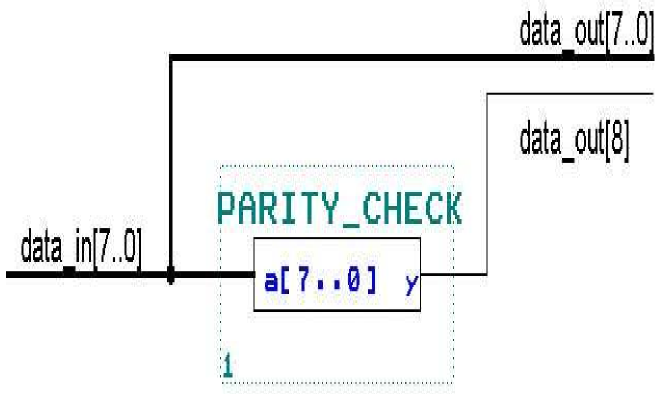


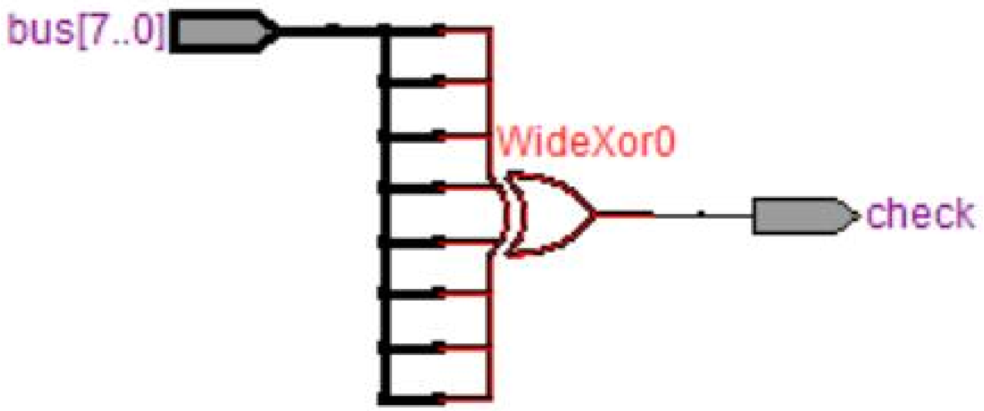
5. 8位奇偶校验位产生电路

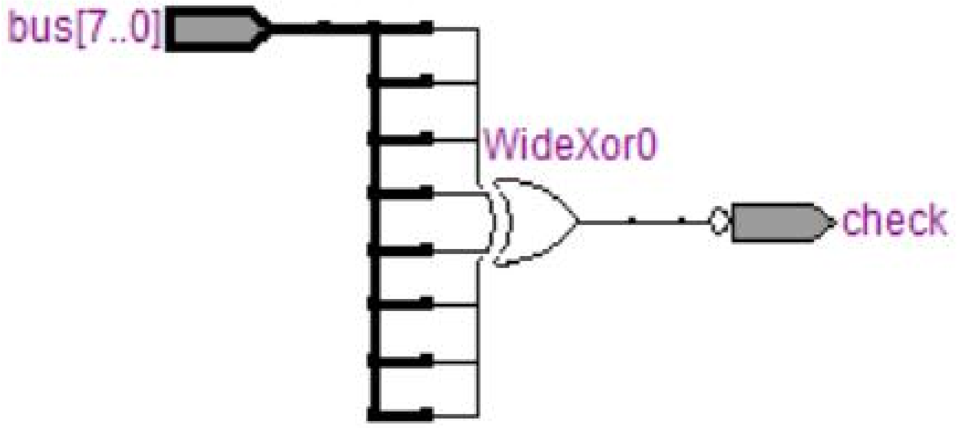
```
module odd_che ( bus , check ) ;

 input [7:0] bus ;
 output check ;

 assign check = ^bus ;

endmodule
```

奇偶？

```
assign  check  =~ ^bus ;
```







二、 时序逻辑电路设计

锁存器、寄存器、计数器、分频器、节拍发生器、状态机等。

时钟上升沿：

```
always @ (posedge clk)
```

时钟下降沿：

```
always @ (negedge clk)
```

原则1：边沿触发信号,不能在always过程中再次出现；

原则2：电平触发信号（或异步控制信号），在always过程中必须用if条件语句对敏感信号表中的信号有匹配的表述，明示信号的逻辑行为。

原则3：同步控制信号，不能出现在敏感信号表中;

原则4：敏感信号列表中不能同时出现边沿触发和电平触发;

原则5：异步敏感信号，异步控制信号在敏感信号表里是边沿敏感信号， 但电路性能上是电平敏感;

原则6：尽量采用同步时序 。

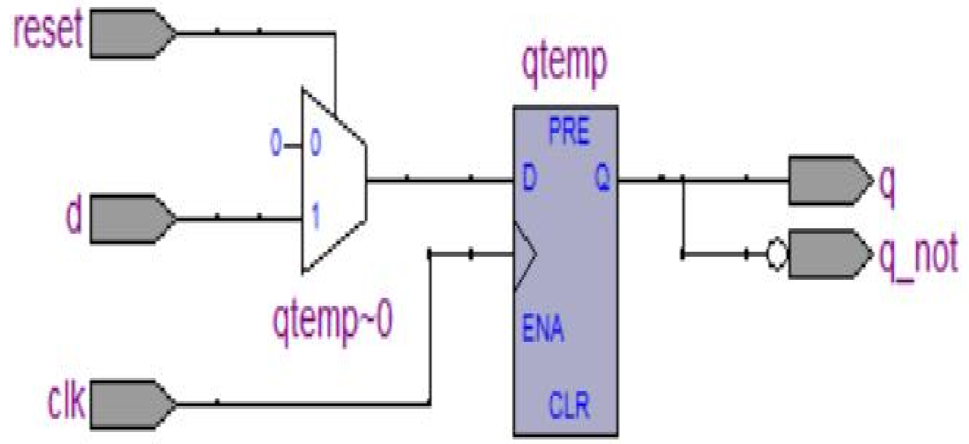
1. 同步复位D触发器(寄存器)

敏感信号表中只有时钟信号——同步复位。

```
module dff ( clk, d , reset , q, q_not  );
 input clk, d , reset;
 output q , q_not ;
 reg qtemp ;

 always @ ( posedge clk )
    begin
      if ( !reset )
         qtemp <= 1’b0 ;
      else
         qtemp <= d ;
     end

   assign q = qtemp ;
   assign q_not = ~qtemp ;

 endmodule
```

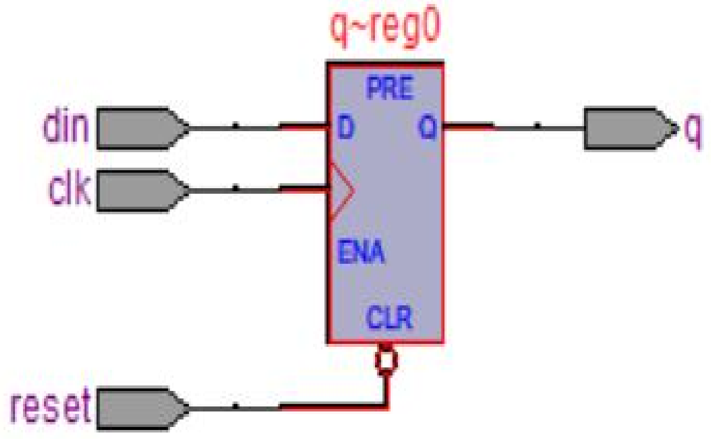
先检查同步信号

if …else都对应时钟沿



2. 异步复位D触发器

敏感信号表中除时钟外，还有其它信号——异步操作



```
always @ ( posedge clk , negedge reset )
   begin
      if ( !reset )
          q <=1'b0;
      else
          q <= din ;
   end
```

- 异步信号必须放在敏感信号表中；
- 必须都是边沿；

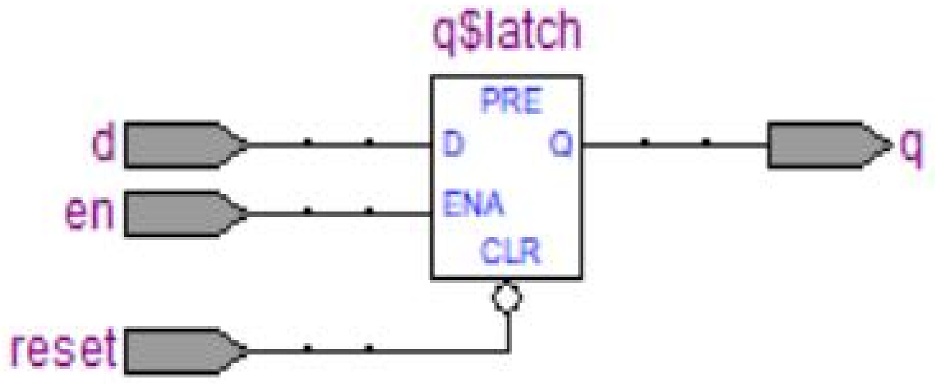
先if描述异步

else分支对应时钟沿

同步、异步区别：敏感信号表

3. 带复位的锁存器

```
always @  (  en ,  d,  reset )
   begin
       if ( !reset )
           q <= 1’b0 ;
      else if ( en )
           q <= d ;
       else
            q <= q ;
  end
```



- 锁存器的所有输入都放在敏感信号表中；
- 锁存器敏感信号都是电平。

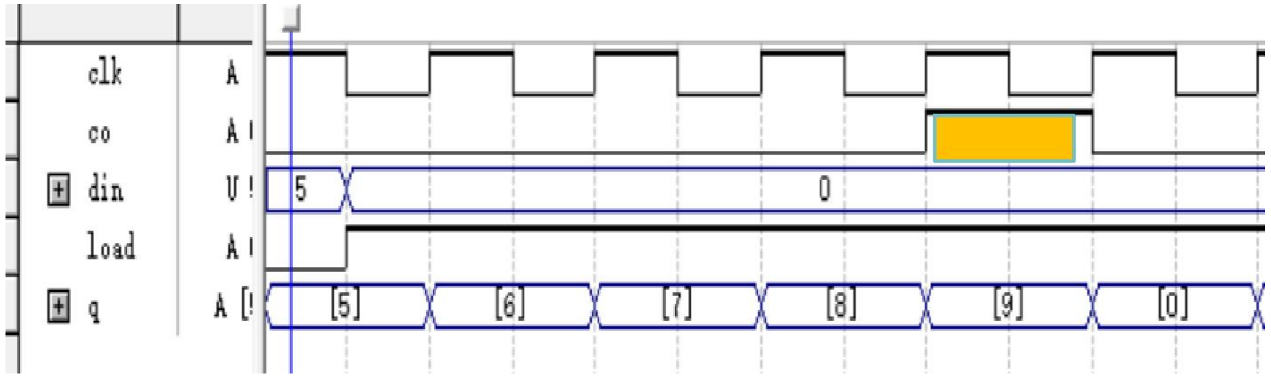
不建议综合成锁存器。

4. 计数器

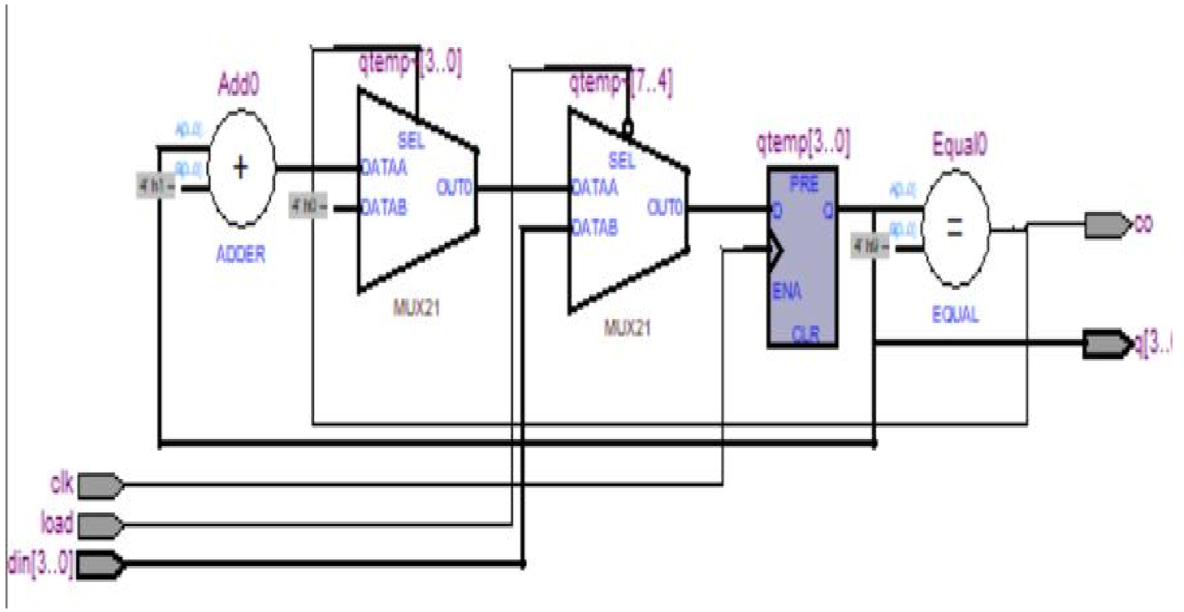
例：同步置数(低有效）的10进制计数器。

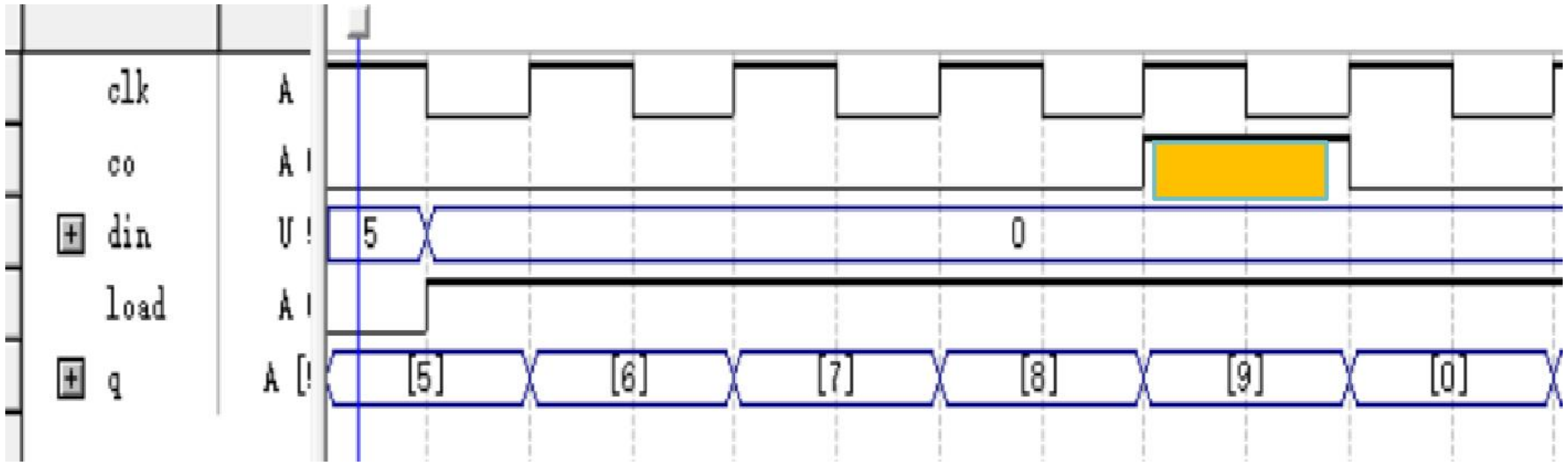
```
module count10 ( clk, load , din , q, co );
input clk , load ;
input [3:0] din ;
output [3:0]  q ;
output co ;
reg [3:0]  qtemp ;

always @ ( posedge clk )
   begin
       if (!load)
             qtemp <= din;
       else if  ( qtemp == 4'd9 )
             qtemp <= 4'd0 ;
       else
             qtemp <= qtemp+4’b1 ;
  end
```

```
 assign q = qtemp ;
 assign co = (qtemp == 4'd9)  ;

endmodule
```



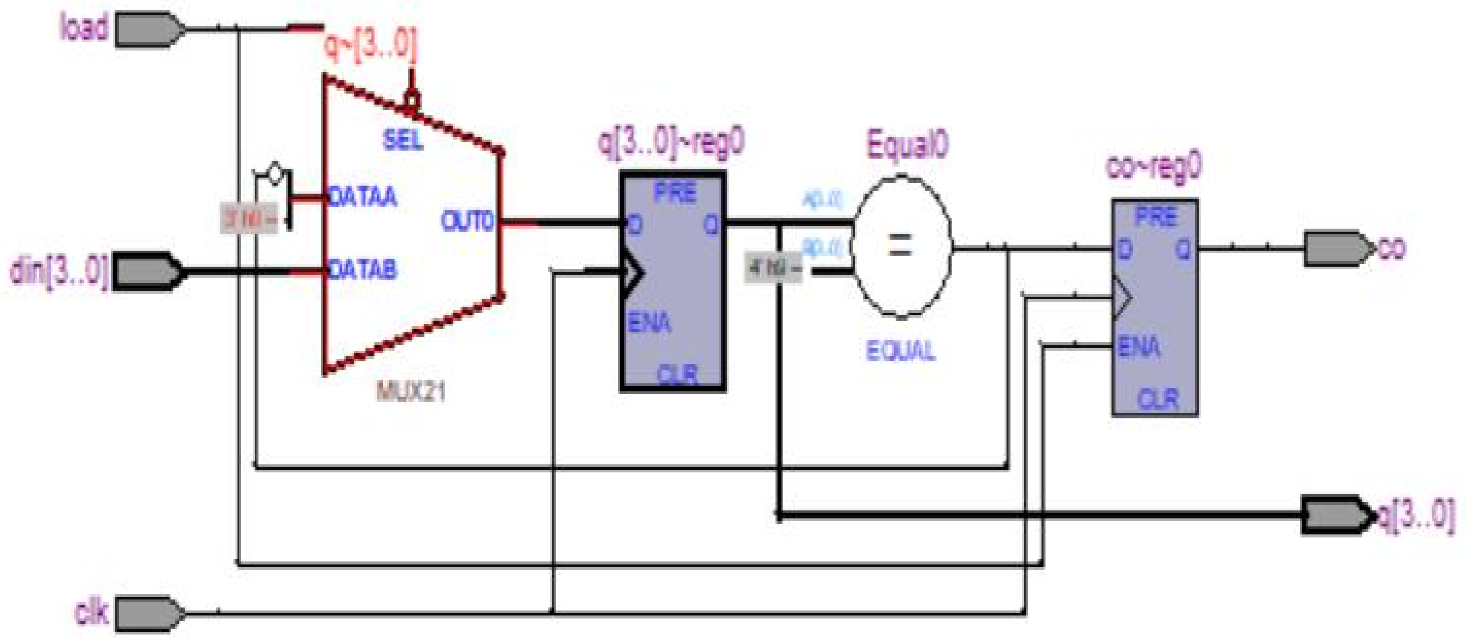
```
always ( posedge clk )
  begin
     if ( qtemp == n'dm-1 )
            qtemp <= n'd0 ;
      else
           qtemp <= qtemp+n'b1;
  end

assign q = qtemp;
assign c = (qtemp == n'dm-1)  ;
```

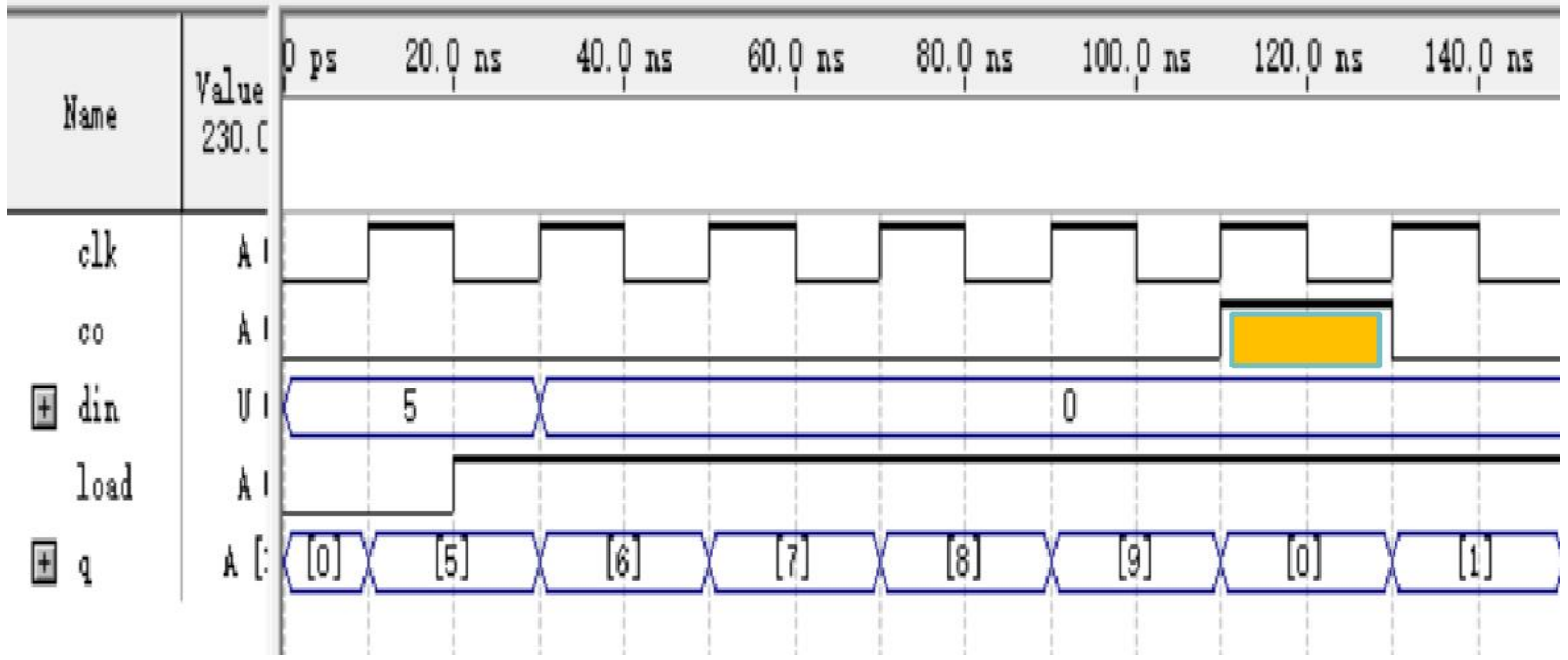




```
module count10(clk, load , din, q, co  );

input clk , load ;
input [3:0] din ;
output [3:0]  q ;
output co ;
reg [3:0]  q ;
reg co ;

always @ (posedge clk)
    begin
        if (!load)
              q <= din;
```

```
   else if  ( q == 4'd9 )
          begin
              q <= 4'd0 ;
              co <= 1'b1 ;
           end
    else
          begin
               q <= q+4’b1 ;
               co <= 1'b0 ;
          end
     end

 endmodule
```


load
q~[3..0]
SEL
DATAA
DATAB
OUT0
MUX21
din[3..0]
q[3..0]~reg0
PRE
D
Q
ENA
CLR
Equal0
=
EQUAL
co~reg0
co
q[3..0]
clk


Name
Value 230.0
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
140.0 ns
clk
co
din
5
0
load
q
[0]
[5]
[6]
[7]
[8]
[9]
[0]
[1]

例：74LS161。

```
module count161 ( clk ,EP ,ET, cr , ld , din , qout ,co );
  input clk ,EP ,ET , cr , ld ;
  input [3:0] din;
  output [3:0] qout ;
  output co ;
  reg [3:0] qtemp;

always @(posedge clk , negedge cr )
  begin
    if (!cr)
      qtemp <= 4'b0000 ;
    else if ( !ld )
      qtemp <= din ;
    else if ( ET && EP )
      qtemp <= qtemp + 1'b1 ;
    else qtemp <= qtemp ;
  end
```

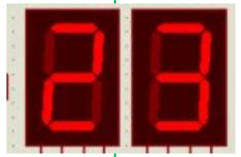
输	入								输	出				CT74LS161
$\overline{CR}$	$\overline{LD}$	EP	ET	CP	D_3	D_2	D_1	D_0	Q_3	Q_2	Q_1	Q_0	CO	
0	×	×	×	×	×	×	×	×	0	0	0	0	0	异步置 0
1	0	×	×	↑	d_3	d_2	d_1	d_0	d_3	d_2	d_1	d_0		$CO = E_T \cdot Q_3Q_2Q_1Q_0$
1	1	1	1	↑	×	×	×	×	计	数				$CO = Q_3 Q_2 Q_1 Q_0$
1	1	0	×	×	×	×	×	×	保	持				$CO = E_T \cdot Q_3Q_2Q_1Q_0$
1	1	×	0	×	×	×	×	×	保	持			0	

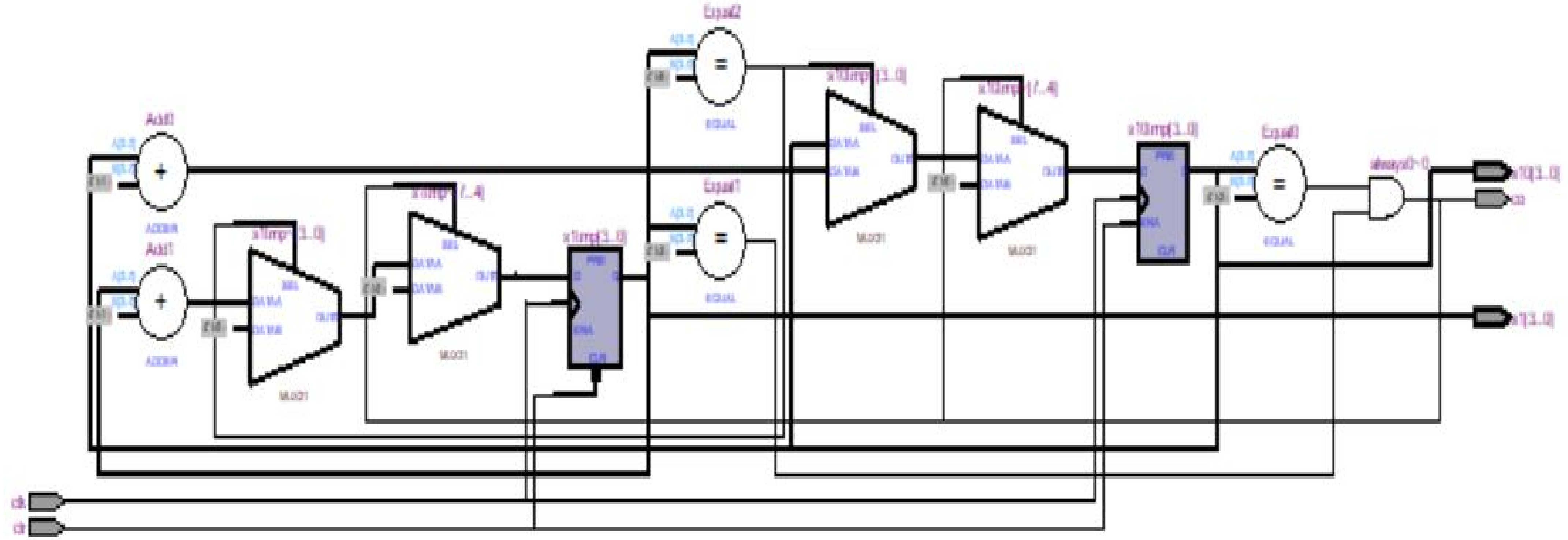
```
 assign co = ET & (qtemp== 4'd15) ;
 assign qout = qtemp ;

endmodule
```

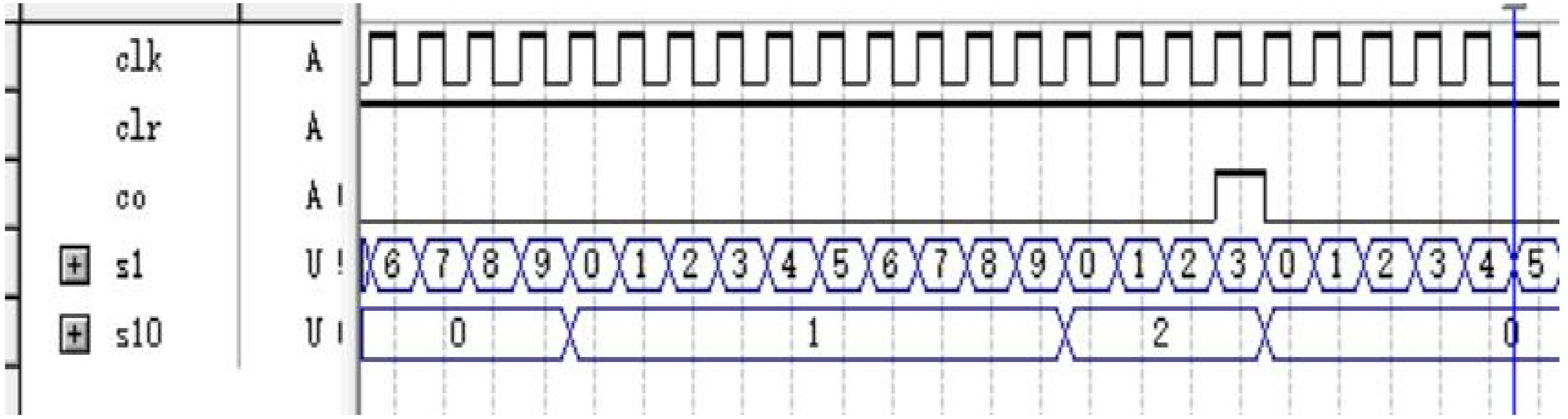
例：异步清零的24进制（小时）计数器

与二进制计数器不同，分别做个位、十位计数

```
module count_24 ( clk , clr, s1 , s10, co );

input clk , clr ;
output [3:0]  s1, s10 ;
output co ;
reg [3:0]  s1tmp , s10tmp ;

always @ ( posedge clk , negedge clr )
    begin
      if ( !clr )
        begin
          s1tmp <= 4'd0 ;
         s10tmp <= 4'd0 ;
        end
      else if (   s10tmp == 4'd2 && s1tmp == 4'd3   )
             begin
                   s10tmp <= 4'd0 ;
                   s1tmp <= 4'd0 ;
              end
       else if ( s1tmp == 4'd9 )
              begin
                  s1tmp <= 4'd0 ;
                  s10tmp <= s10tmp + 4'd1 ;
                end
        else
                  s1tmp <= s1tmp + 4'd1 ;
      end

assign s1 = s1tmp ;
assign s10 = s10tmp ;
assign co = (s10tmp==4'd2) && ( s1tmp==4'd3) ;

endmodule
```




Add0
Add1
ADDER
MUX21
x1tmp[3..0]
Equal0
Equal1
Equal2
EQUAL
x10tmp[3..0]
clk
clr
x10[3..0]
co
x1[3..0]

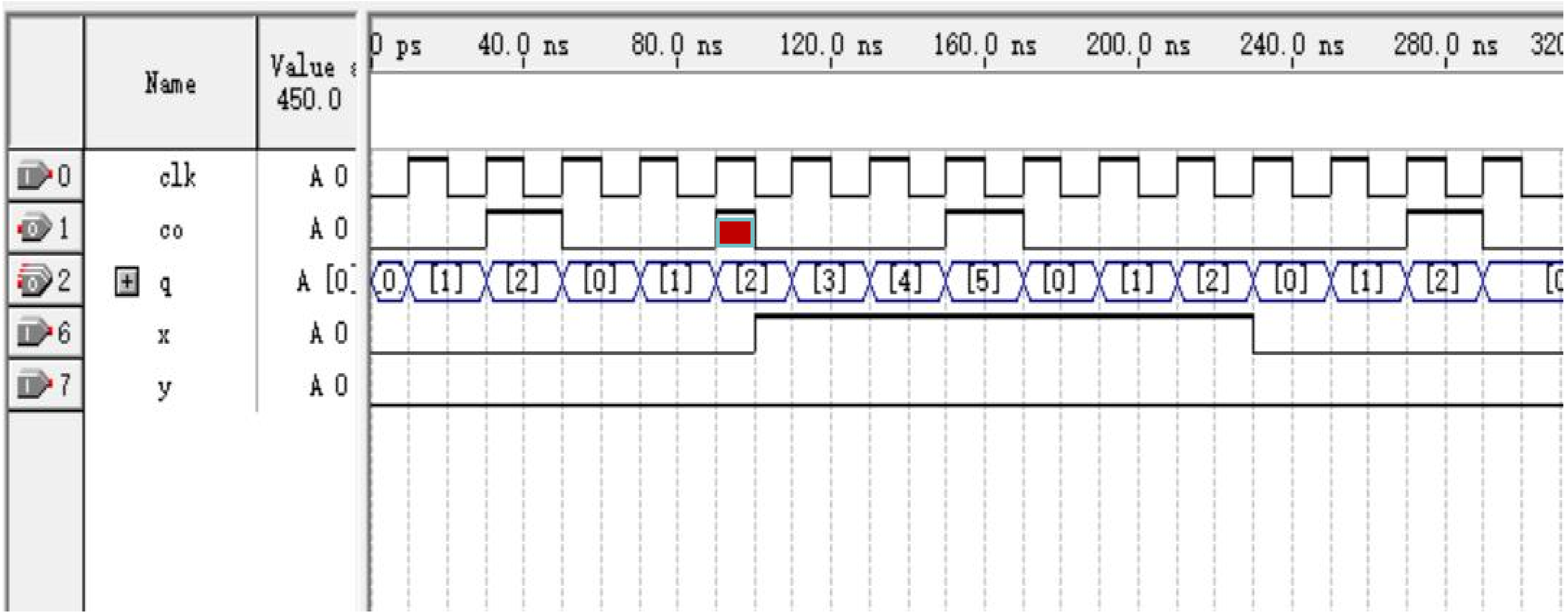

clk
clr
co
s1
s10
6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 0 1 2 3 4 5
0 1 2 0

模可控计数器（由输入x, y控制）

M=3,4,6,7

```
module count(  clk , x , y  , q , co );
input clk , x , y ;
output [2:0]  q;
output co ;
reg [2:0]  qtemp , m ;

always @ ( x ,y )        // 组合过程
 begin
    case ({x , y} )
      2'b00:  m = 3'd3 ;
      2'b01:  m = 3'd4 ;
      2'b10:  m = 3'd6 ;
      default: m = 3'd7 ;
    endcase
 end
```

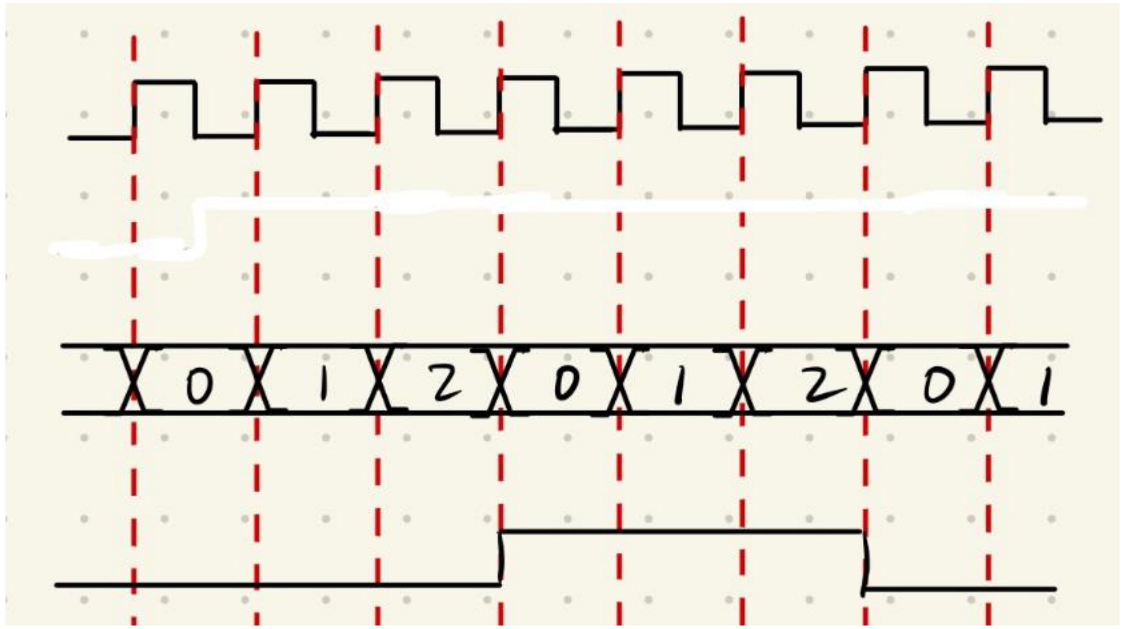
```
always @ ( posedge clk  )   // 时序过程
 begin
     if ( qtemp == m-3’d1 )
        qtemp <= 3'd0 ;
     else
        qtemp <= qtemp + 3'd1
 end

 assign q = qtemp ;
 assign co = (qtemp == m-3’d1)   ;

endmodule
```

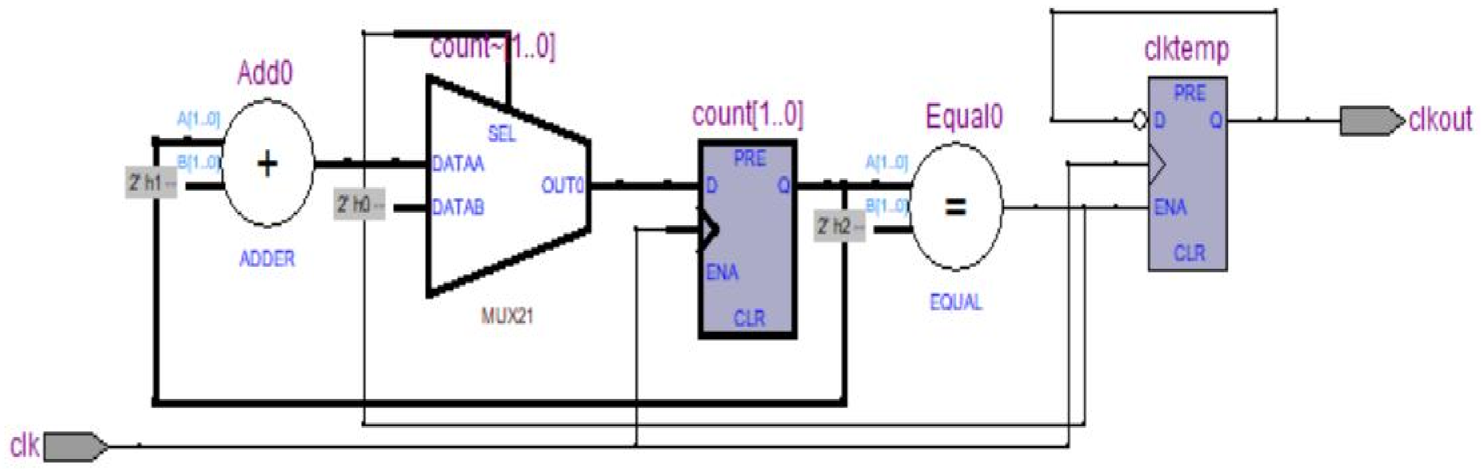


Name
Value 450.0
0 ps
40.0 ns
80.0 ns
120.0 ns
160.0 ns
200.0 ns
240.0 ns
280.0 ns
320
0 clk A 0
1 co A 0
2 q A [0]
6 x A 0
7 y A 0
0
[1]
[2]
[0]
[1]
[2]
[3]
[4]
[5]
[0]
[1]
[2]
[0]
[1]
[2]

5.　设计6分频器，占空比50%

```
module devider ( clk , clkout   );
input clk  ;
output   clkout ;
reg [1:0]  count ;
reg clktemp;
always @ ( posedge clk )
 begin
    if ( count == 2'd2 )
       count <= 2'd0 ;
    else
       count <= count+2'd1 ;
 end

always @ ( posedge clk  )
 begin
     if (count == 2'd2 )
        clktemp <= ~clktemp ;
     else
        clktemp <= clktemp;
  end
assign clkout = clkoutemp ;
  endmodule
```

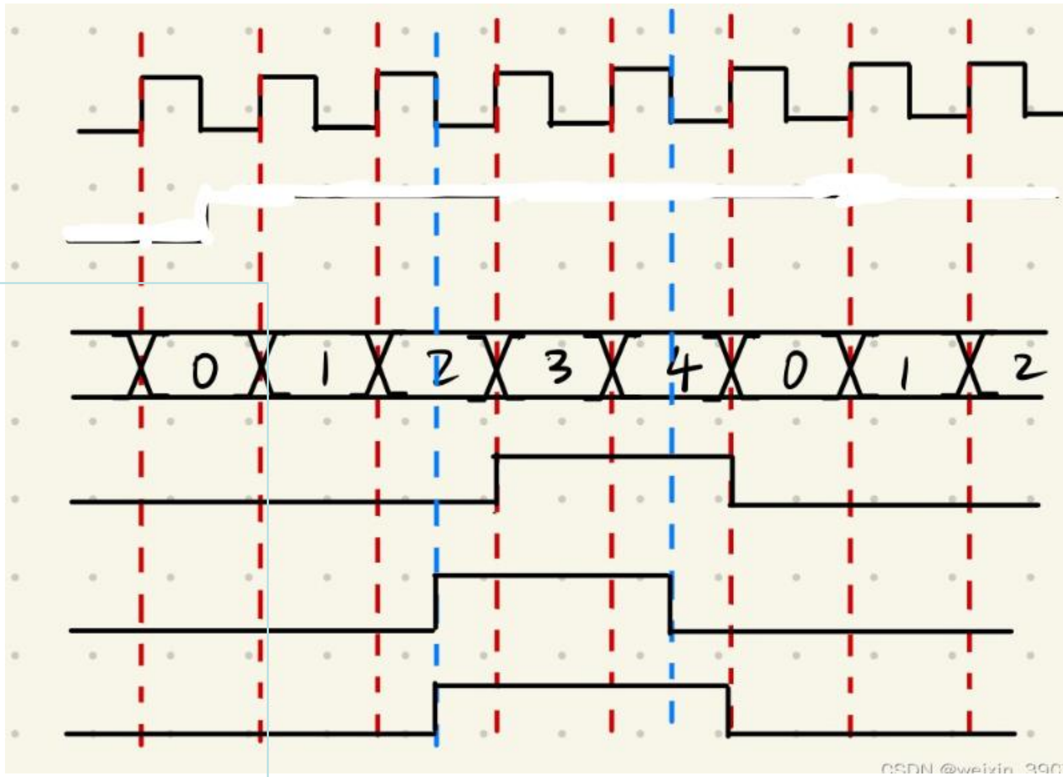


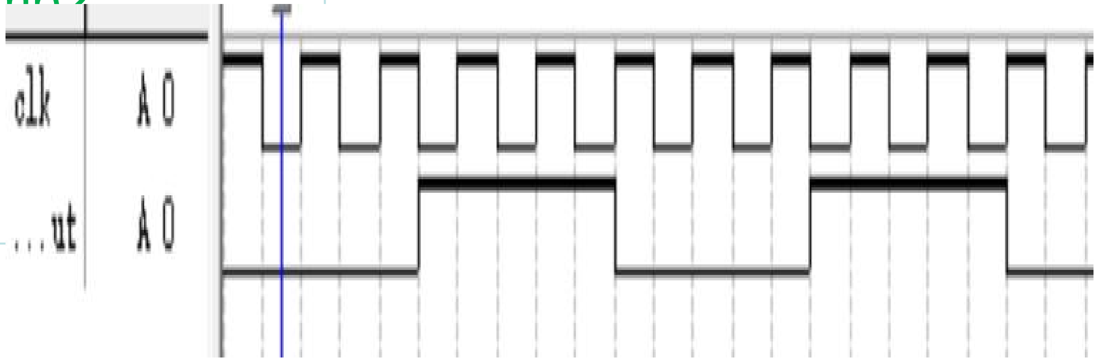


5.　设计5分频器，占空比50%

```
module devider (  clk , clkout );
input clk  ;
output   clkout ;
reg [2:0]  count ;
reg temp1 , temp2;

always @ ( posedge clk )
 begin
    if ( count == 3'd4 )
       count <= 3'd0 ;
    else
       count <= count + 3'd1 ;
 end

always @ ( posedge clk  )
 begin
     if ( count == 3'd2 )
         temp1 <= 1'b1 ;
     else if ( count == 3'd4 )
         temp1 <= 1'b0 ;
     else  temp1 <= temp1 ;
  end
```

```
always @ (negedge clk )
  begin
       if ( count == 3'd2 )
          temp2 <= 1'b1 ;
       else if ( cout == 3’b4 )
          temp2 <= 1'b0 ;
       else  temp2 <= temp2 ;
  end

assign clkout = temp1 | temp2 ;

endmodule
```





6.　移存型计数器

例：4位环形移位寄存器（左、右移、异步置数1000）

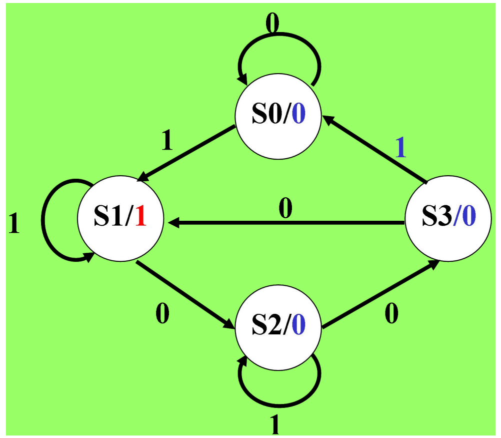
```
module shift ( clk , con , rst , qout  );
input clk , con , rst  ;
output [3:0]  qout ;
reg [3:0]  qtemp;

always @ ( posedge clk , negedge rst )
 begin
     if ( ! rst )
        qtemp <= 4'b1000 ;
     else  if ( con )
            qtemp <= {qtemp[2:0],qtemp[3]} ;       //左移
     else
            qtemp <= {qtemp[0],qtemp[3:1]} ;      // 右移
 end

 assign  qout = qtemp ;

endmodule
```

7. 状态机

1.) 使用两段式描述:

描述时序逻辑（状态改变）

组合逻辑（次态、输出）



2）状态机说明语句：

parameter ： 各状态编码

```
parameter s0=2'b00 , s1=2'b01, s2=2'b10,s3=2'b11 ;

parameter s0=2'b00 , s1=2'b01, s2=2'b11,s3=2'b10 ;

parameter s0=4'b0001 , s1=2'b0010 ;
parameter s2=4'b0100 , s3=2'b1000 ;
```

二进制码（binary）

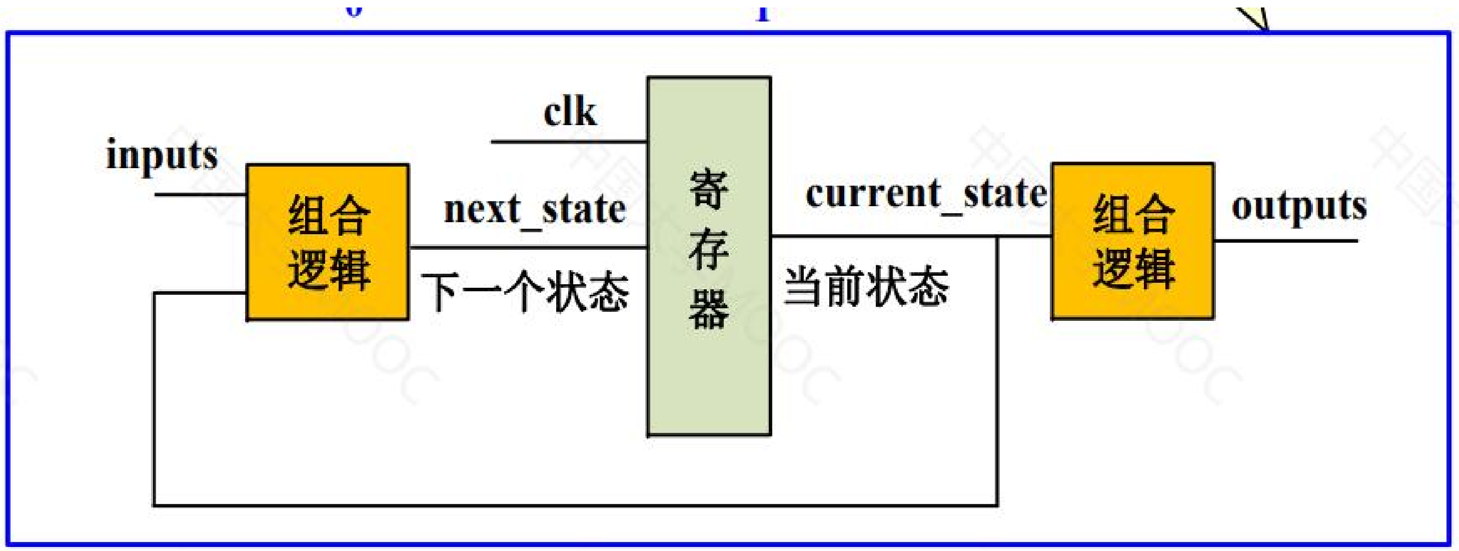
格雷码（Gray）

独热码（one-hot）

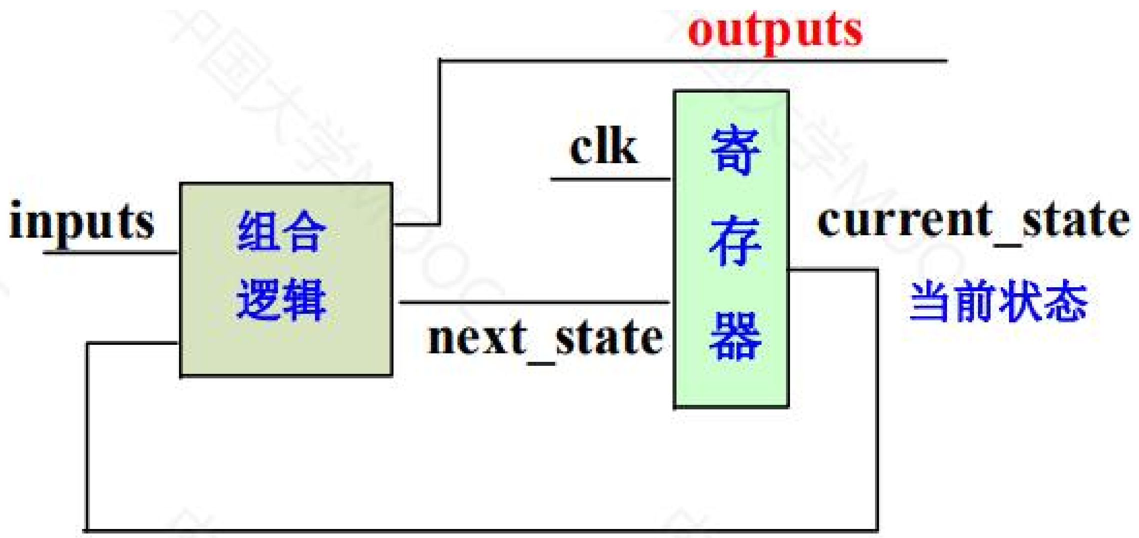
（大型）

3）状态机要复位：避免进入挂起状态。

4）Moore 型状态机:



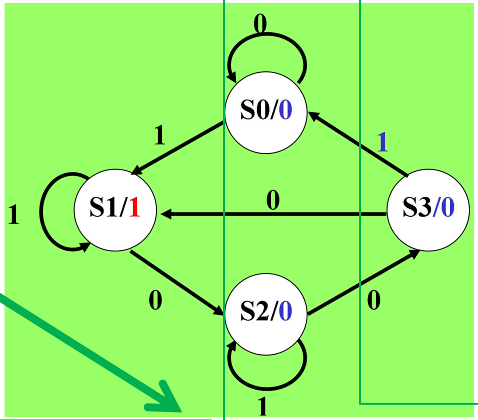
5）Mealy 型状态机:



Moore 型状态机的描述

```
module moore ( clk ,x ,reset , y );
input clk , x , reset ;
output   y ;
parameter s0=2'b00 , s1=2'b01 , s2=2'b11 ,
s3=2'b10 ;
reg [1:0]  current_state , next_state;
reg y ;

always @ ( posedge clk , negedge reset ) ;
//时序逻辑
 begin
    if ( ! reset )
      current_state <= s0 ;
    else
       current_state <= next_state ;
 end
```

```
always @ (current_state , x )  ;
 // 组合逻辑
   begin
     case (current_state)
       s0: begin
              y = 1'b0 ;
              if ( x == 1'b0)
                  next_state = s0;
              else
                  next_state = s1;
           end
       s1: begin
              y = 1'b1 ;
              next_state  =  ( x == 1'b0)? s2:s1 ;
           end
```

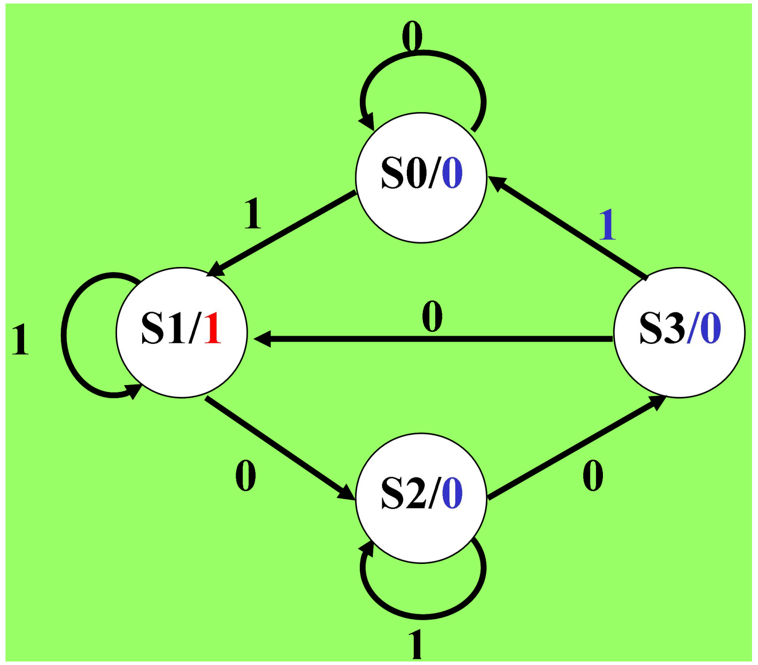


第一进程，完成状态转换，必须饱含时钟敏感信号

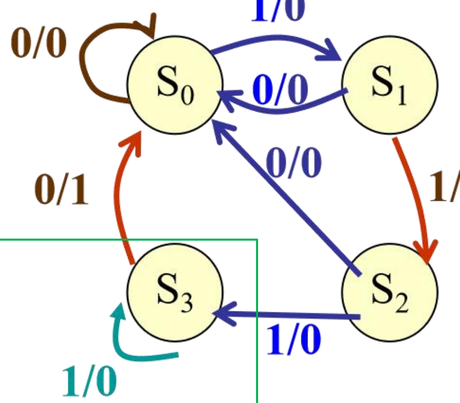
第二进程：输出和状态逻辑间关系，必须包含现态、输入等敏感信号

```
      s2: begin
              y  = 1'b0 ;
             next_state = ( x == 1'b0)?  s3:s2 ;
           end
      s3: begin
              y  = 1'b0 ;
             next_state = ( x == 1'b0)?  s1:s0 ;
          end
       default:  begin
                    y = 1'b0 ;
                   next_state = s0;
                 end
     endcase
  end

endmodule
```

摩尔型



Mealy型有限状态机的设计

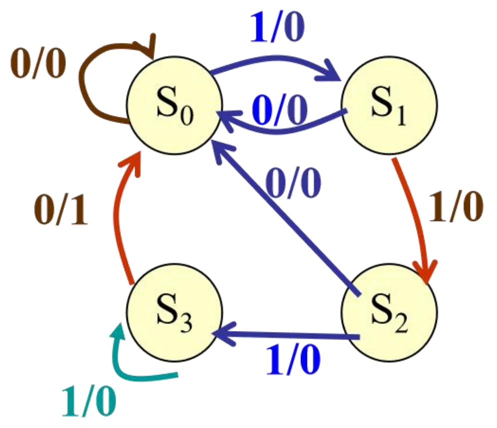


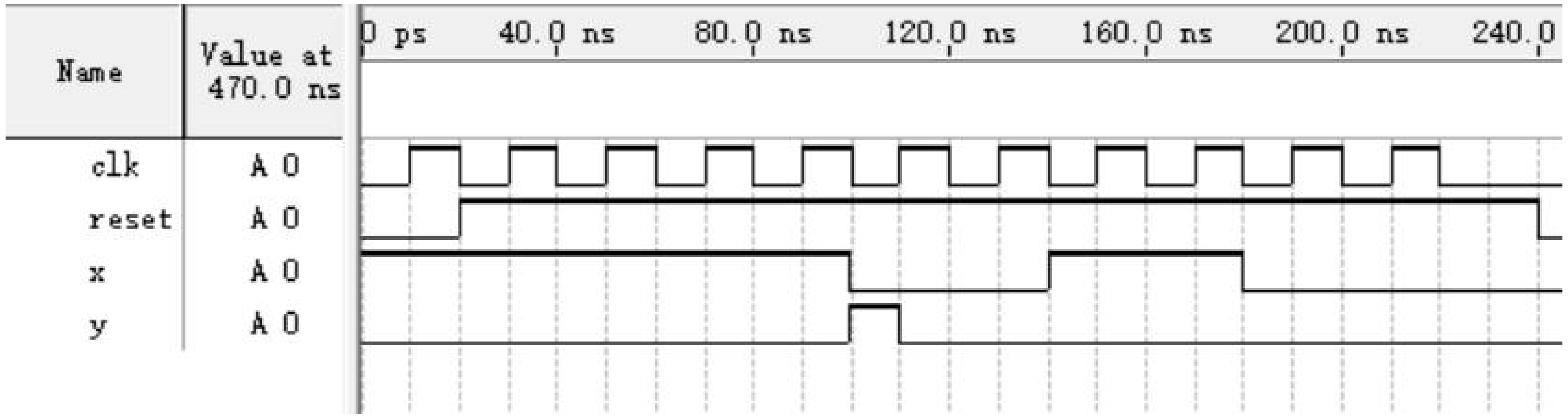
```
module mealy ( clk ,x ,reset ,  y , );
input clk , x , reset;
output   y ;
parameter s0=2'b00 , s1=2'b01 , s2=2'b11 ,
s3=2'b10 ;
reg [1:0]  current_state , next_state;
reg y ;

always @ ( posedge clk , negedge reset )
 begin
    if ( ! reset )
       current_state <= s0 ;
    else
        current_state <= next_state ;
 end
```

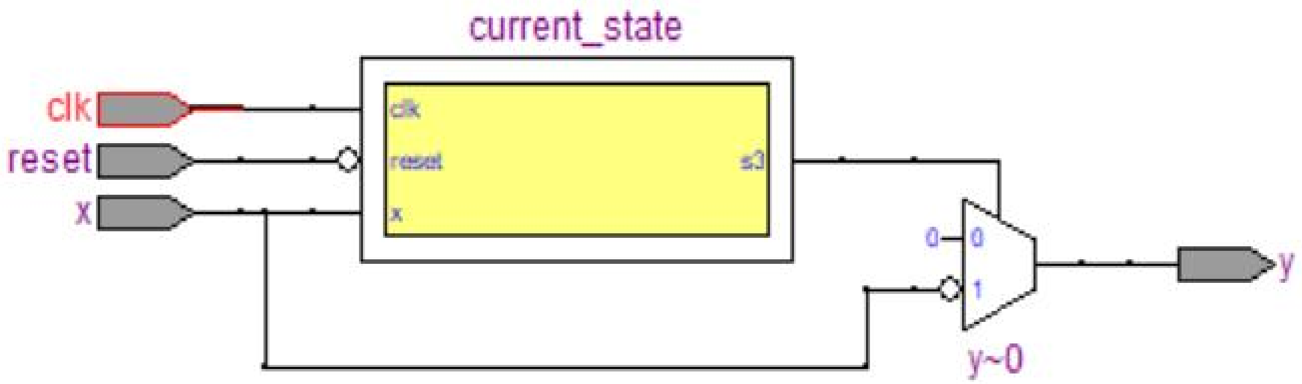
```
always @ (current_state , x)
   begin
     case (current_state)
       s0: begin
             if ( x == 1'b0 )
                 begin
                    next_state = s0;
                    y = 1'b0;
                 end
              else
                  begin
                    next_state = s1;
                    y = 1'b0;
                 end
            end
```

```
  s1: begin
           if ( x == 1'b0 )
              begin
                next_state = s0;
                y = 1'b0;
              end
          else
              begin
                next_state = s2;
                y = 1'b0;
              end
       end
 s2: begin
           if ( x == 1'b0 )
              begin
                next_state = s0;
                y = 1'b0;
              end
          else
              begin
                next_state = s3;
                y = 1'b0;
              end
       end
```

```
  s3: begin
              if ( x == 1'b0 )
                 begin
                   next_state = s0;
                   y = 1'b1;
                 end
             else
                 begin
                   next_state = s3;
                   y = 1'b0;
                 end
          end
    default:  begin
              y = 1'b0 ;
              next_state = s0;
            end
    endcase
  end

endmodule
```



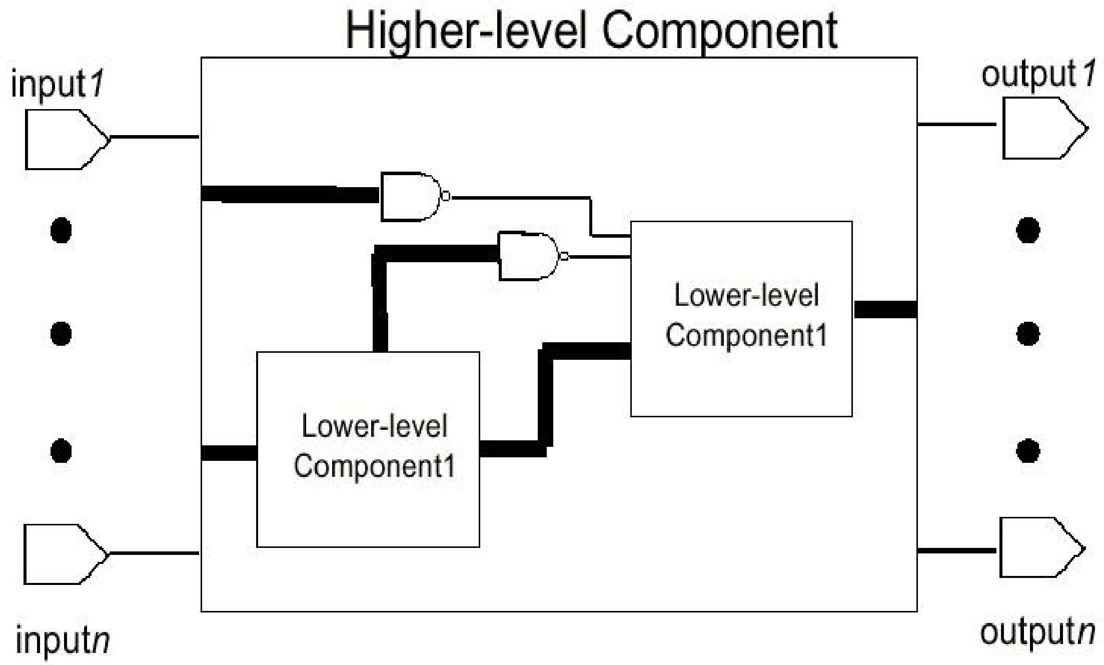

Name
Value at 470.0 ns
0 ps
40.0 ns
80.0 ns
120.0 ns
160.0 ns
200.0 ns
240.0
clk
A 0
reset
A 0
x
A 0
y
A 0


current_state
clk
reset
x
clk
reset
x
s3
0
0
1
y~0
y

5.5.4 verilog的层次化（结构化）设计

在多层次的设计中，高层次的设计模块调用低层次的设计模块，构成模块化的设计。重点描述模块间的连接关系。

子模块名　例化名　（父子模块端口关联表）；



```
AND  u1 ( a, b, and_out );
AND  u1 ( .a(a),  .b(b),  .o(and_out) );
```

子

父

端口列表：位置关联

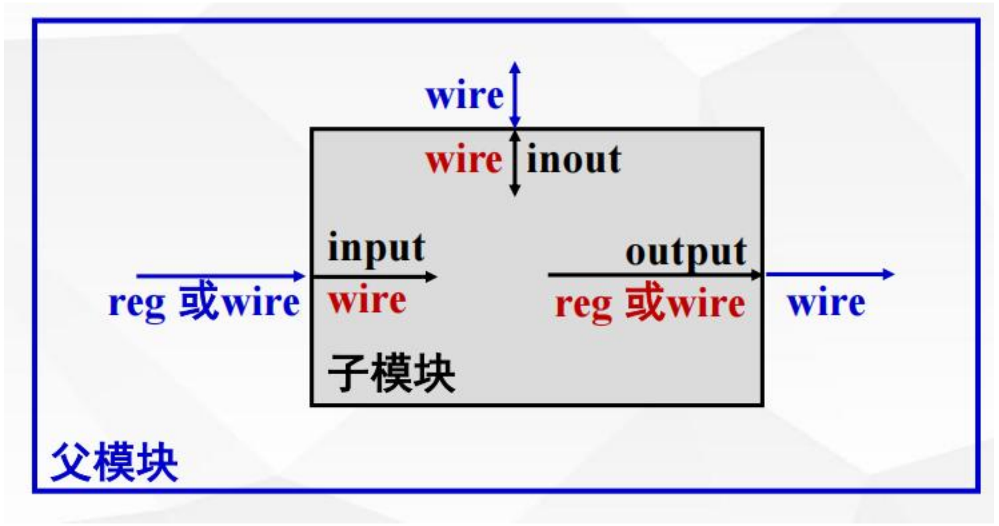
或名字关联

被例化元件的来源:

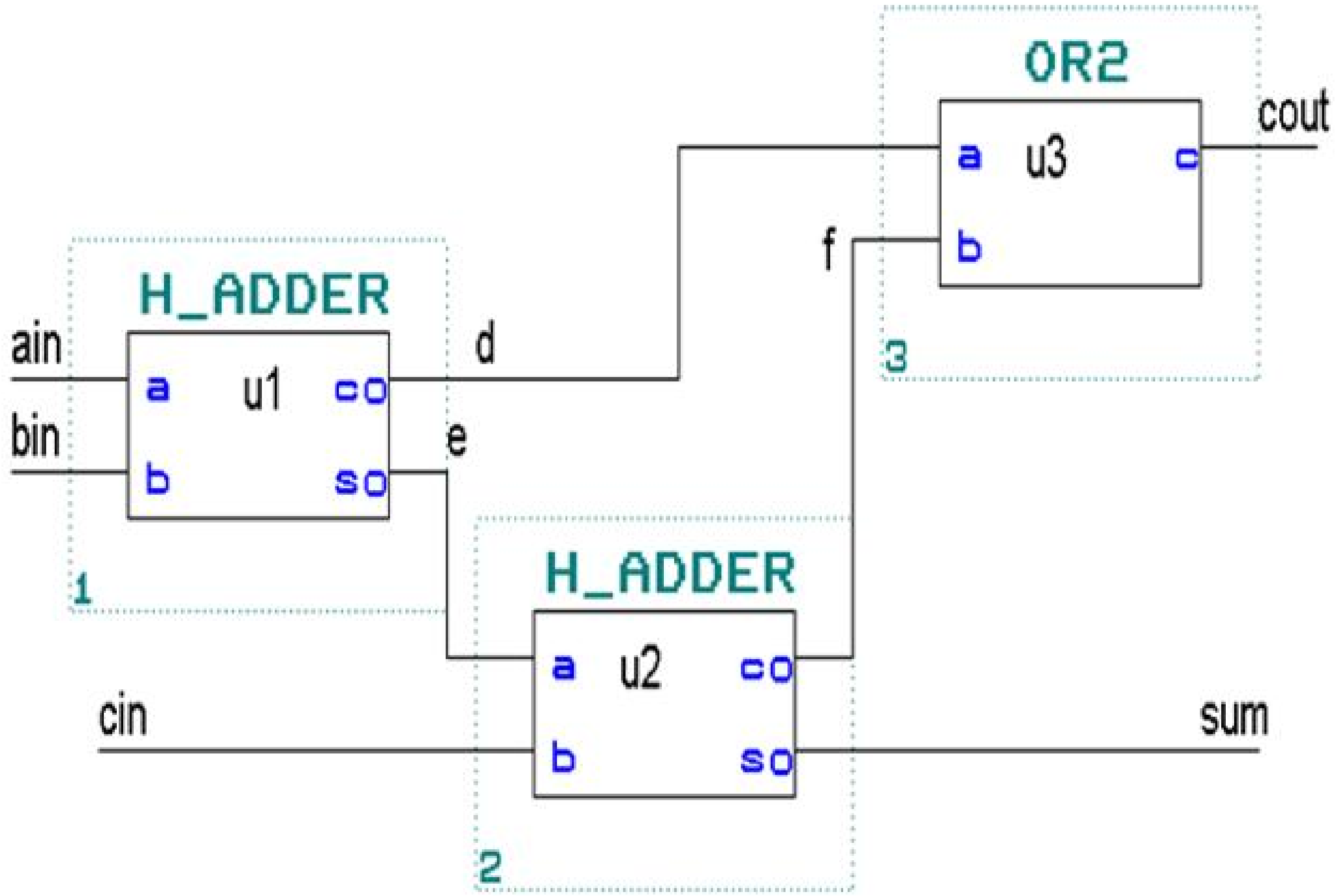
- verilog 设计模块;
- 其它HDL设计实体;
- 厂商提供的工艺库中的元件、IP核。

注意:

- ➢ 一个元件是一段结构完整的 module 模块。
- ➢ 不能在always语句内部引用子模块。
- ➢ 有关模块端口数据类型的规定



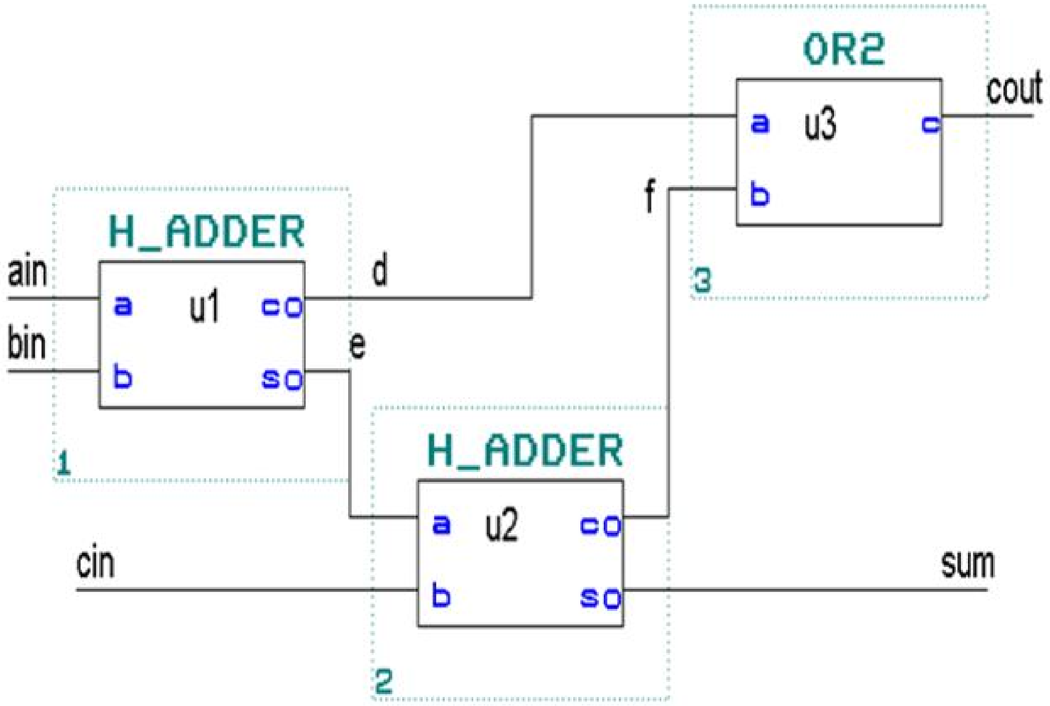
一位全加器原理图


OR2
u3
a
b
c
cout
f
3
H_ADDER
u1
a
b
co
so
ain
bin
d
e
1
H_ADDER
u2
a
b
co
so
cin
sum
2

```
module h_adder (  a , b, so ,co  ) ;  //半加器
input a , b;
output so , co ;

assign so = a ^ b ;
assign co = a & b ;

endmodule
```

```
module adde( ain , bin, cin , sum , cout  );    顶层逻辑
input ain, bin , cin ;
output   sum , cout ;
wire d , e , f ;

h_adder   u1 ( ain , bin, e, d  ) ;   //元件例化
h_adder   u2 (.a(e) , .b(cin) , .so(sum) , .co(f )) ;
or_2   u3 ( d , f , cout ) ;

endmodule
```

```
module or_2 ( a , b, c ) ;    //或门
input a , b;
output c ;

assign c = a | b ;

endmodule
```



```
module lampion(  clk , clr ,y );
input clk , clr ;
output [7:0]   y ;
reg [4:0] count ;
reg [7:0] y ;

always @( posedge clk , negedge clr)
  begin
    if ( !clr )
        count <= 5'd0 ;
     else
        count <= count+5’b1 ;
  end

always @ ( count )
   begin
      case ( count )
       5'd0:  y = 8'b0000_0001 ;
       5'd1:  y = 8'b0000_0010 ;
```

8个彩灯，4花样自动切换的彩灯控制器：

第1：从右到左，然后从左到右逐次点亮，再全黑，再全亮；

第2：两边同时亮1个逐次向中间移动再散开；

第3：两边同时亮2个逐次向中间移动再散开；

第4：每三个亮、每四个亮、间隔亮。

共32个种变换，循环往复。

```
        5'd2:  y = 8'b0000_0100 ;
        5'd3:  y = 8'b0000_1000 ;
        5'd4:  y = 8'b0001_0000 ;
        5'd5:  y = 8'b0010_0000 ;
        5'd6:  y = 8'b0100_0000 ;
        5'd7:  y = 8'b1000_0000 ;
        5'd8:  y = 8'b1000_0000 ;
        5'd9:  y = 8'b0100_0000 ;
        5'd10:  y = 8'b0010_0000 ;
        5'd11:  y = 8'b0001_0000 ;
        5'd12:  y = 8'b0000_1000 ;
        5'd13:  y = 8'b0000_0100 ;
        5'd14:  y = 8'b0000_0010 ;
        5'd15:  y = 8'b0000_0001 ;
        5'd16:  y = 8'b0000_0000 ;
        5'd17:  y = 8'b1111_1111 ;
        5'd18:  y = 8'b1000_0001 ;
```

```
       5'd19:  y = 8'b0100_0010 ;
        5'd20:  y = 8'b0010_0100 ;
        5'd21:  y = 8'b0001_1000 ;
        5'd22:  y = 8'b0010_0100 ;
        5'd23:  y = 8'b0100_0010 ;
        5'd24:  y = 8'b1000_0001 ;
        5'd25:  y = 8'b1100_0011 ;
        5'd26:  y = 8'b0011_1100 ;
        5'd27:  y = 8'b1100_0011 ;
        5'd28:  y = 8'b1110_0111 ;
        5'd29:  y = 8'b0111_1110 ;
        5'd30:  y = 8'b1111_0000 ;
        5'd31:  y = 8'b1111_0000 ;
        default: y = 8'b0000_0000 ;
        endcase
    end

endmodule
```

序列信号发生器如何实现？

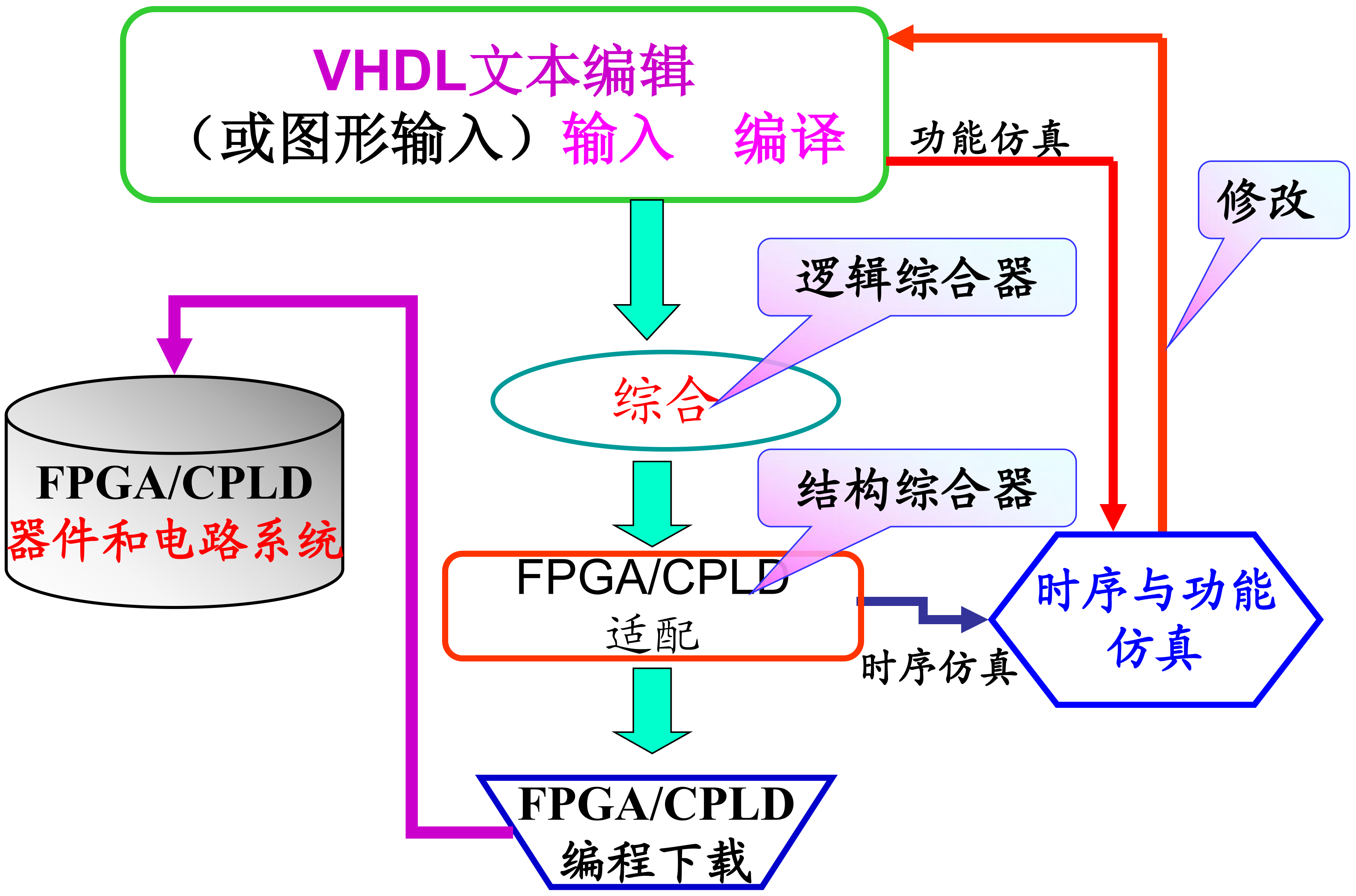
顺序脉冲发生器如何实现？

可编程逻辑器件器件的开发流程

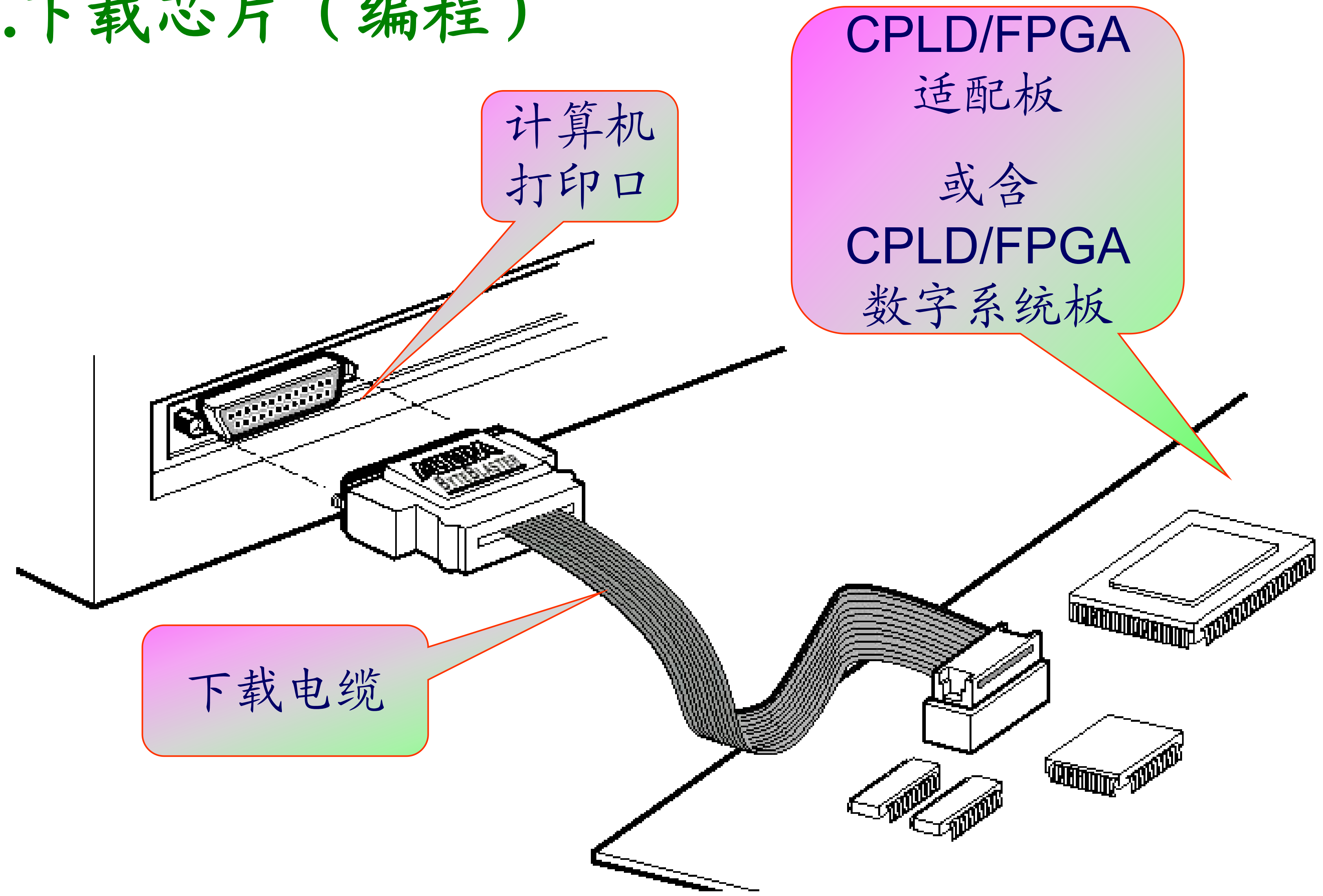
开发工具

- 开发设计
  - 软件设计
  - 硬件设计
    - 下载板（线）制作
    - 用户板制作

三、软件设计流程


VHDL文本编辑
（或图形输入）输入 编译
功能仿真
修改
逻辑综合器
综合
结构综合器
FPGA/CPLD
器件和电路系统
FPGA/CPLD
适配
时序仿真
时序与功能
仿真
FPGA/CPLD
编程下载

四.下载芯片（编程）


计算机
打印口
CPLD/FPGA
适配板
或含
CPLD/FPGA
数字系统板
下载电缆