DIGITAL DESIGN

LAB12 FREQUENCY DIVIDER

2020 FALL TERM @ CSE . SUSTECH

LAB12

- Frequency divider
  - Divide by even
  - Divide by odd
- Structure design
  - demo
    - Flowing light
    - 7-seg tube

CLOCK ON MINISYS BOARD

10纳秒

- **Minisys** board includes a **100MHz** crystal oscillator connected to the main chip **Y18** pin. Through requirement design, the input clock can drive MMCMs or PLLs to produce multi-frequency clock and phase changes.

FREQUENCY DIVIDER

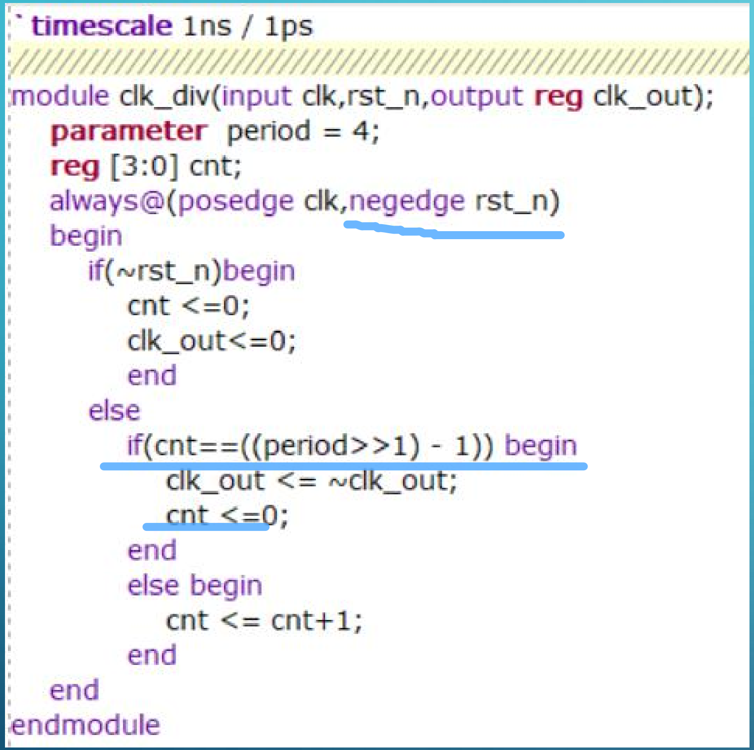
- A **Frequency Divider**, also called a **clock divider** or scaler or pre-scaler, is a circuit that takes an input signal of a frequency fin, and generates an output signal of a frequency fout:

  fout = fin/n (n is an integer).

- For power-of-2 integer division, a simple binary counter can be used, clocked by the input signal. The least-significant output bit alternates at 1/2 the rate of the input clock, the next bit at 1/4 the rate, the third bit at 1/8 the rate, etc.
- An arrangement of flipflops is a classic method for integer-n division. Such division is frequency and phase coherent to the source over environmental variations including temperature. The easiest configuration is a series where each flip-flop is a divide-by-2. For a series of three of these, such system would be a divide-by-8. By adding additional logic gates to the chain of flip flops, other division ratios can be obtained. Integrated circuit logic families can provide a single chip solution for some common division ratios.

https://en.wikipedia.org/wiki/Frequency_divider

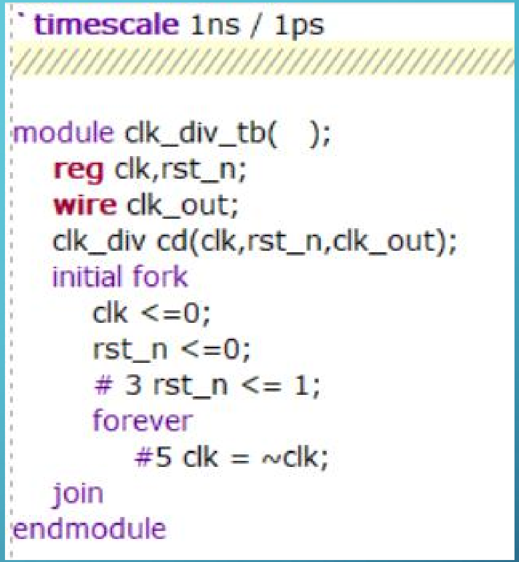
FREQUENCY DIVIDER(N:4)



```
`timescale 1ns / 1ps
module clk_div(input clk,rst_n,output reg clk_out);
  parameter  period = 4;
  reg [3:0] cnt;
  always@(posedge clk,negedge rst_n)
  begin
     if(~rst_n)begin
        cnt <=0;
        clk_out<=0;
        end
     else
        if(cnt==((period>>1) - 1)) begin
           clk_out <= ~clk_out;
           cnt <=0;
        end
        else begin
           cnt <= cnt+1;
        end
  end
endmodule
```

异步复位信号

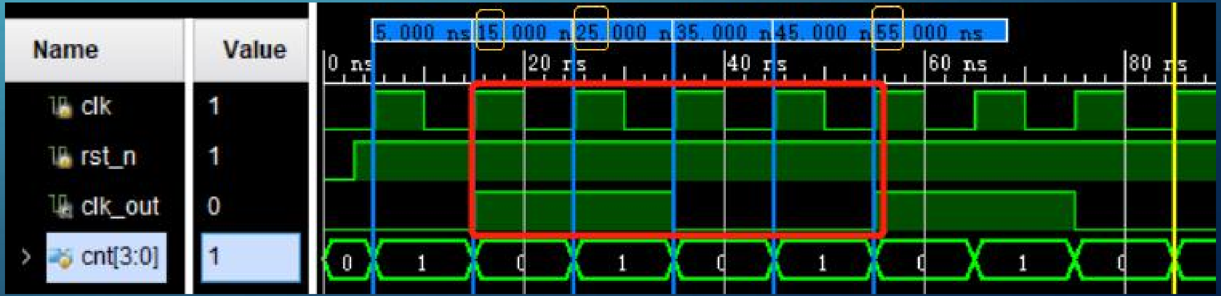
半周期计时一次



```
`timescale 1ns / 1ps
module clk_div_tb(   );
  reg clk,rst_n;
  wire clk_out;
  clk_div cd(clk,rst_n,clk_out);
  initial fork
     clk <=0;
     rst_n <=0;
     # 3 rst_n <= 1;
     forever
        #5 clk = ~clk;
  join
endmodule
```

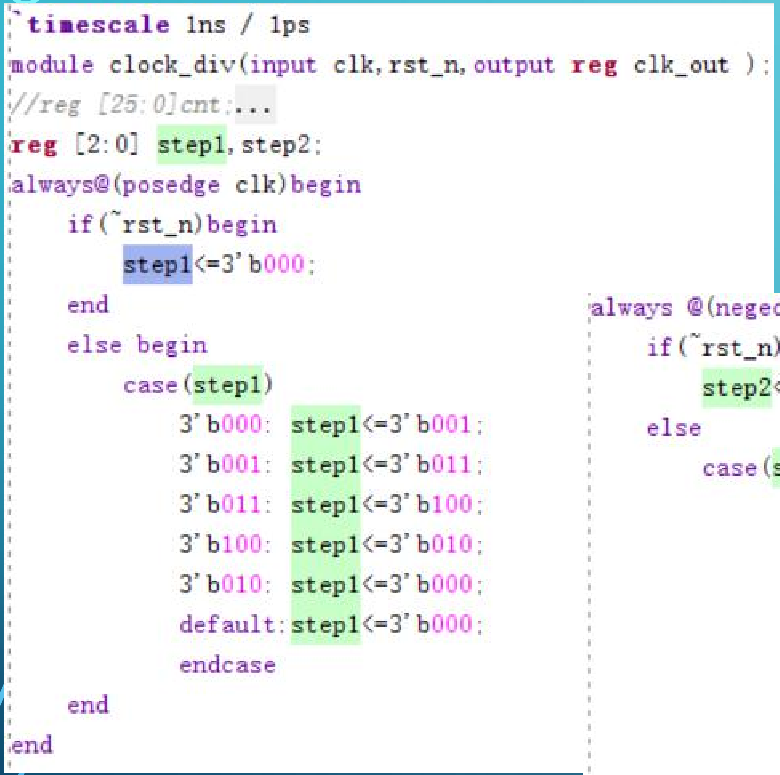
fork join 不能综合

占空比：一个时钟周期里高电平占比

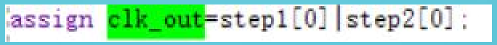


?

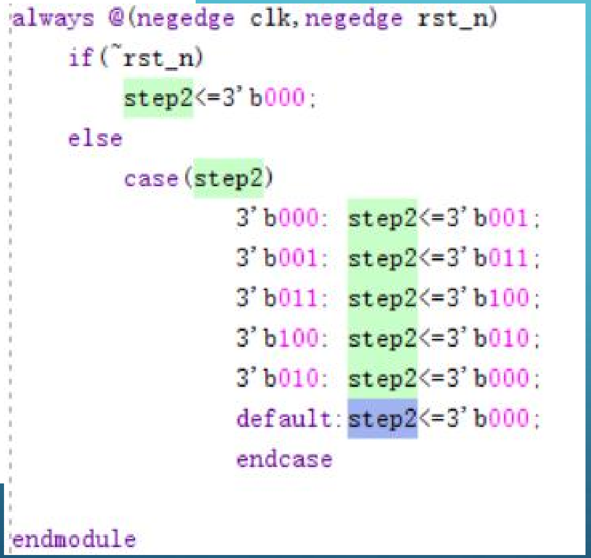
FREQUENCY DIVIDER(N:5)



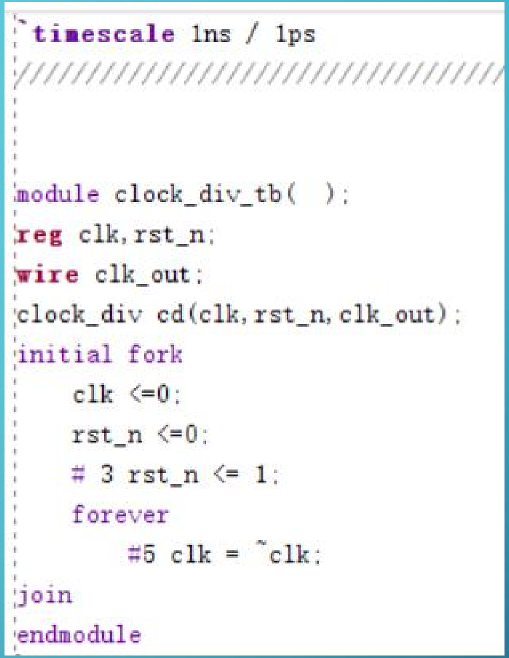
```
`timescale 1ns / 1ps
module clock_div(input clk,rst_n,output reg clk_out );
//reg [25:0]cnt;...
reg [2:0] step1,step2;
always@(posedge clk)begin
    if(~rst_n)begin
        step1<=3'b000;
    end
    else begin
        case(step1)
            3'b000: step1<=3'b001;
            3'b001: step1<=3'b011;
            3'b011: step1<=3'b100;
            3'b100: step1<=3'b010;
            3'b010: step1<=3'b000;
            default:step1<=3'b000;
            endcase
    end
end
```



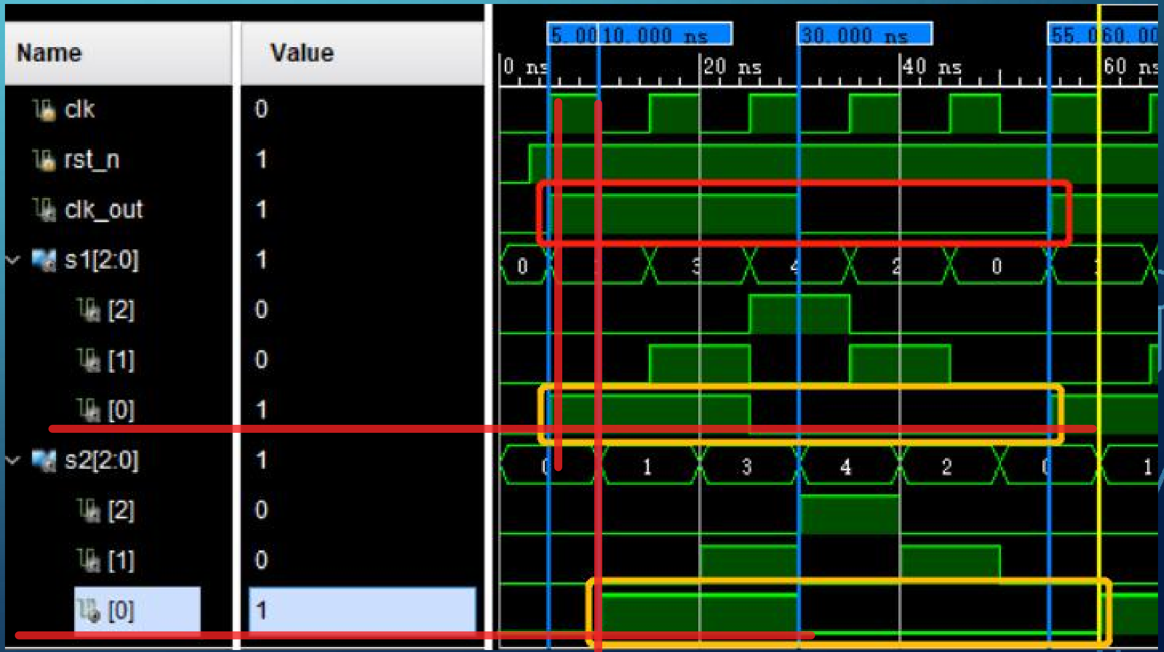
```
assign clk_out=step1[0]|step2[0];
```



```
always @(negedge clk,negedge rst_n)
    if(~rst_n)
        step2<=3'b000;
    else
        case(step2)
            3'b000: step2<=3'b001;
            3'b001: step2<=3'b011;
            3'b011: step2<=3'b100;
            3'b100: step2<=3'b010;
            3'b010: step2<=3'b000;
            default:step2<=3'b000;
            endcase


endmodule
```

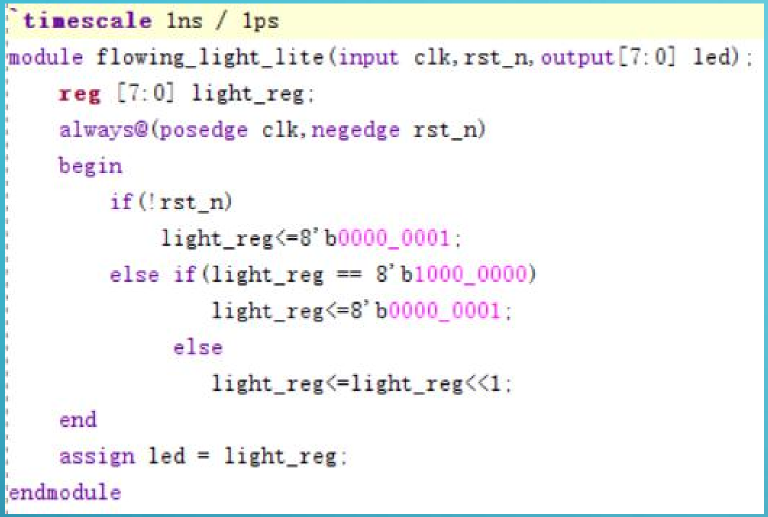


```
`timescale 1ns / 1ps
/////////////////////////////////


module clock_div_tb( );
reg clk,rst_n;
wire clk_out;
clock_div cd(clk,rst_n,clk_out);
initial fork
    clk <=0;
    rst_n <=0;
    # 3 rst_n <= 1;
    forever
        #5 clk = ~clk;
join
endmodule
```

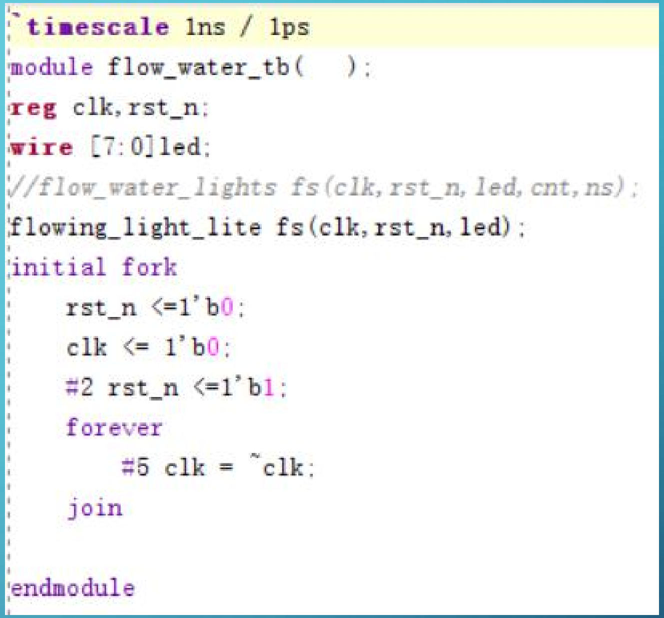


Name	Value
clk	0
rst_n	1
clk_out	1
s1[2:0]	1
[2]	0
[1]	0
[0]	1
s2[2:0]	1
[2]	0
[1]	0
[0]	1

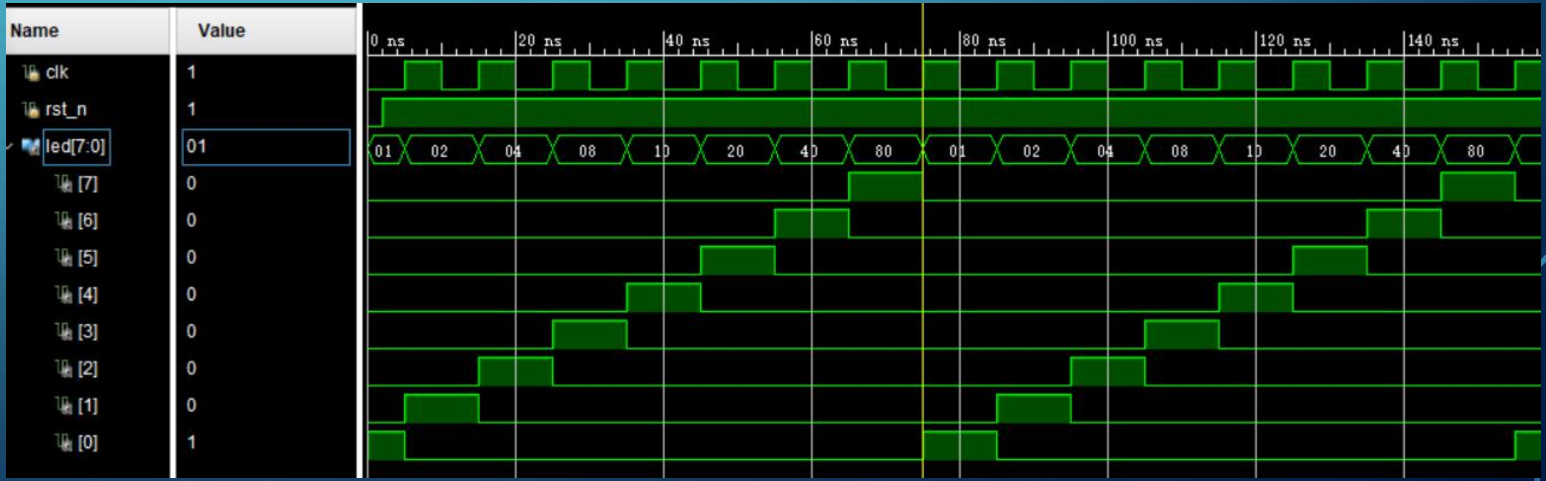
DEMO1 （FLOWING LIGHT1）



```
`timescale 1ns / 1ps
module flowing_light_lite(input clk,rst_n,output[7:0] led);
    reg [7:0] light_reg;
    always@(posedge clk,negedge rst_n)
    begin
        if(!rst_n)
            light_reg<=8'b0000_0001;
        else if(light_reg == 8'b1000_0000)
                light_reg<=8'b0000_0001;
             else
                light_reg<=light_reg<<1;
    end
    assign led = light_reg;
endmodule
```



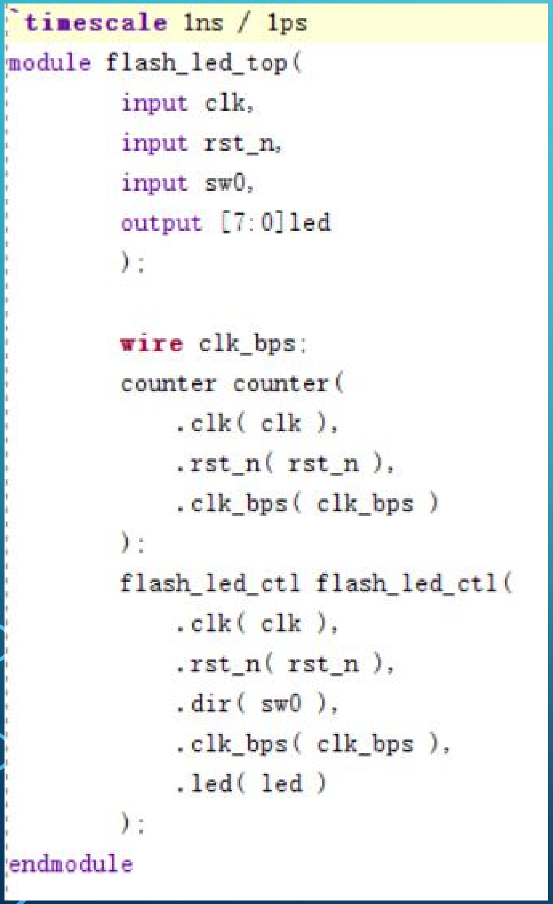
```
`timescale 1ns / 1ps
module flow_water_tb(   );
reg clk,rst_n;
wire [7:0]led;
//flow_water_lights fs(clk,rst_n,led,cnt,ns);
flowing_light_lite fs(clk,rst_n,led);
initial fork
    rst_n <=1'b0;
    clk <= 1'b0;
    #2 rst_n <=1'b1;
    forever
        #5 clk = ~clk;
    join

endmodule
```



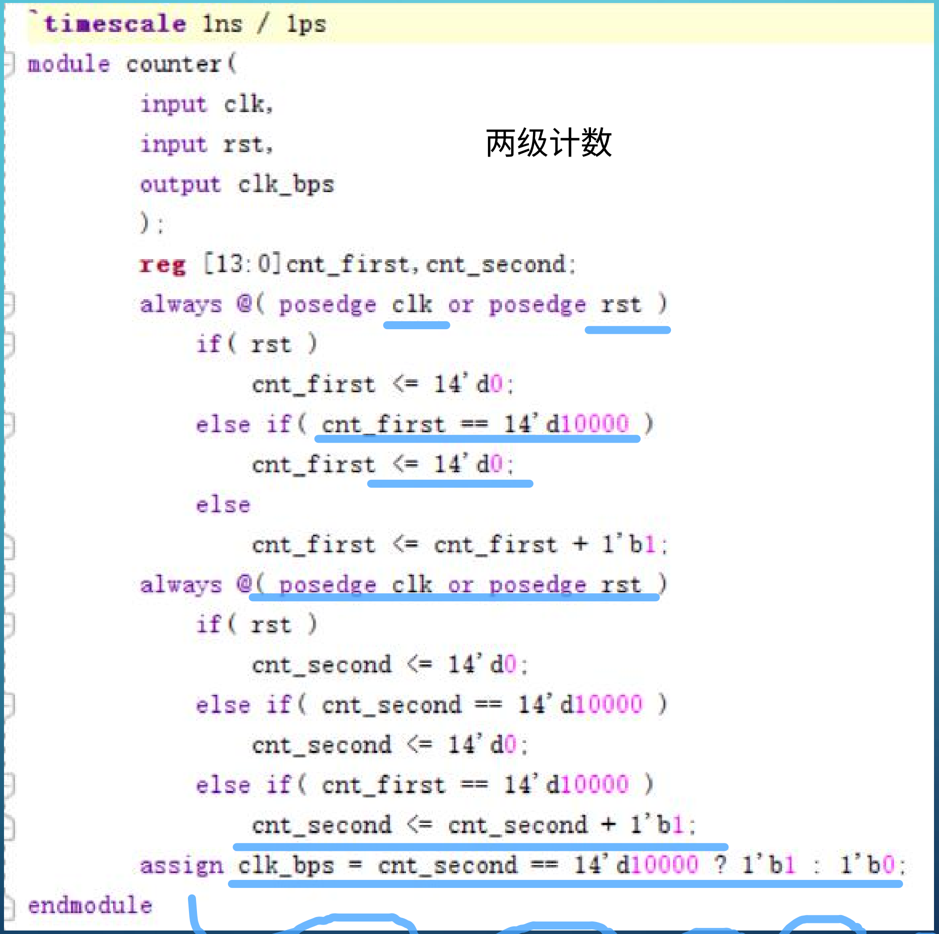
用拨码开关模拟时钟要加上一句话

DEMO1 （FLOWING LIGHT2）

Minisys board includes a 100MHz crystal oscillator connected to the main chip Y18 pin.

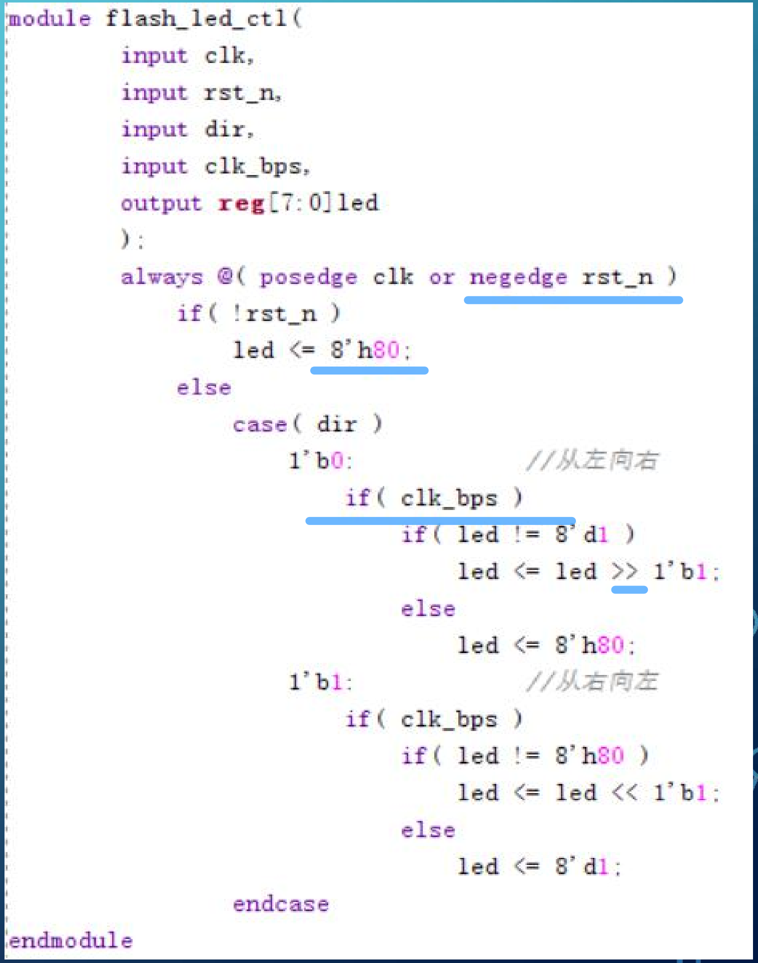


```
`timescale 1ns / 1ps
module flash_led_top(
        input clk,
        input rst_n,
        input sw0,
        output [7:0]led
        );

        wire clk_bps;
        counter counter(
            .clk( clk ),
            .rst_n( rst_n ),
            .clk_bps( clk_bps )
        );
        flash_led_ctl flash_led_ctl(
            .clk( clk ),
            .rst_n( rst_n ),
            .dir( sw0 ),
            .clk_bps( clk_bps ),
            .led( led )
        );
endmodule
```



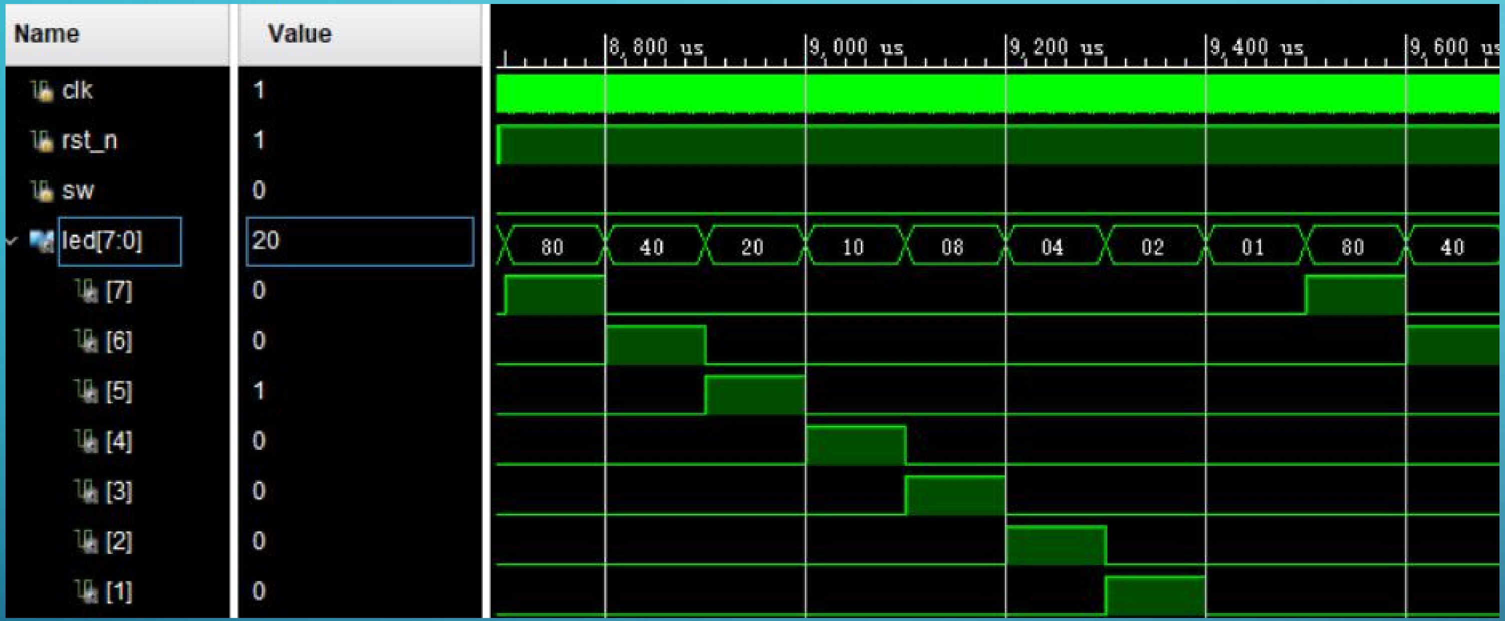
两级计数

```
`timescale 1ns / 1ps
module counter(
        input clk,
        input rst,
        output clk_bps
        );
        reg [13:0]cnt_first,cnt_second;
        always @( posedge clk or posedge rst )
            if( rst )
                cnt_first <= 14'd0;
            else if( cnt_first == 14'd10000 )
                cnt_first <= 14'd0;
            else
                cnt_first <= cnt_first + 1'b1;
        always @( posedge clk or posedge rst )
            if( rst )
                cnt_second <= 14'd0;
            else if( cnt_second == 14'd10000 )
                cnt_second <= 14'd0;
            else if( cnt_first == 14'd10000 )
                cnt_second <= cnt_second + 1'b1;
        assign clk_bps = cnt_second == 14'd10000 ? 1'b1 : 1'b0;
endmodule
```



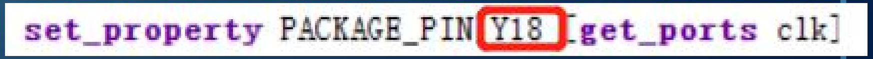
```
module flash_led_ctl(
        input clk,
        input rst_n,
        input dir,
        input clk_bps,
        output reg[7:0]led
        );
        always @( posedge clk or negedge rst_n )
            if( !rst_n )
                led <= 8'h80;
            else
                case( dir )
                    1'b0:               //从左向右
                        if( clk_bps )
                            if( led != 8'd1 )
                                led <= led >> 1'b1;
                            else
                                led <= 8'h80;
                    1'b1:               //从右向左
                        if( clk_bps )
                            if( led != 8'h80 )
                                led <= led << 1'b1;
                            else
                                led <= 8'd1;
                endcase
endmodule
```

DEMO1 （FLOWING LIGHT2）



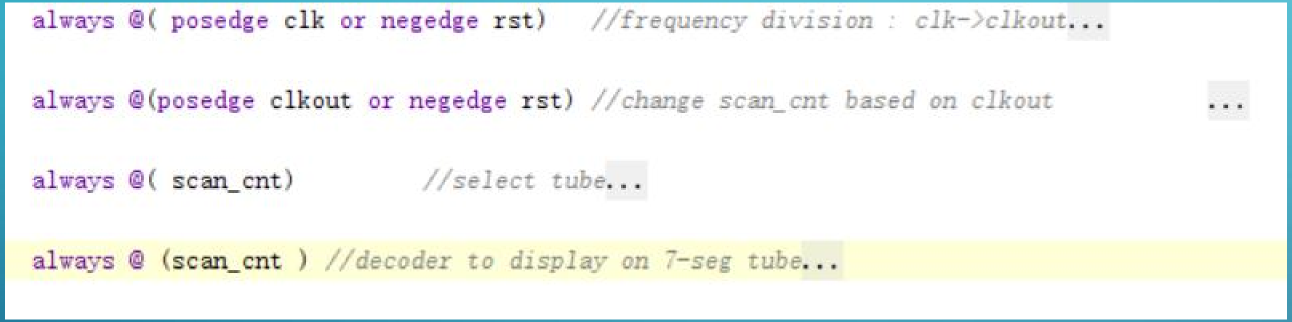
Name	Value
clk	1
rst_n	1
sw	0
led[7:0]	20
[7]	0
[6]	0
[5]	1
[4]	0
[3]	0
[2]	0
[1]	0

Test on Minisys board

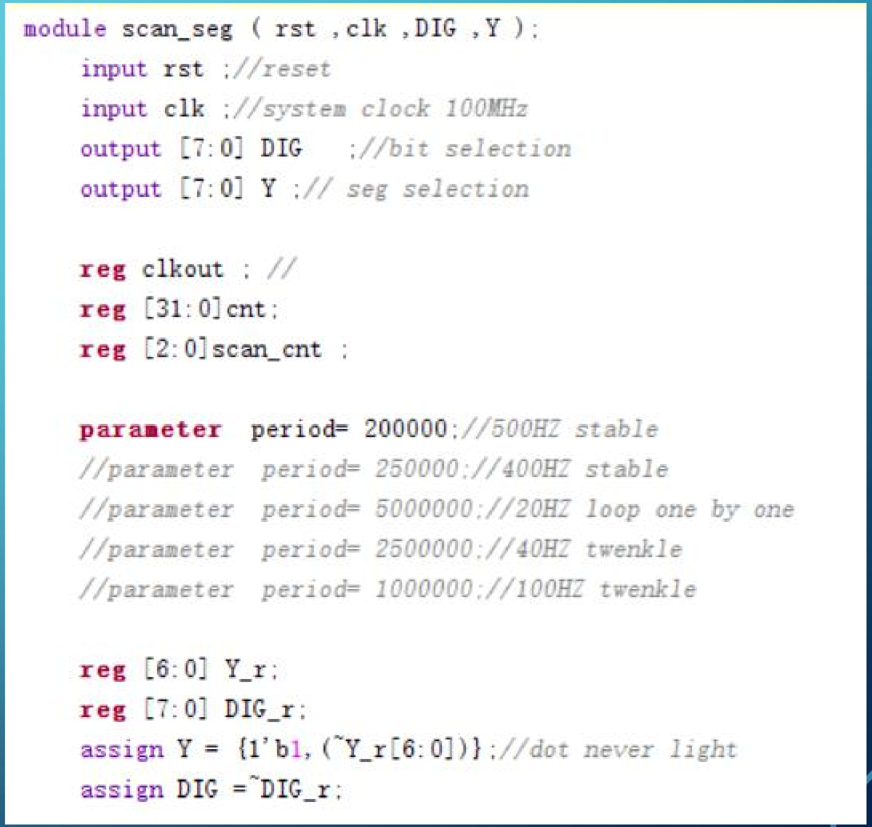


```
set_property PACKAGE_PIN Y18 [get_ports clk]
```

DEMO2： 7-SEG TUBE



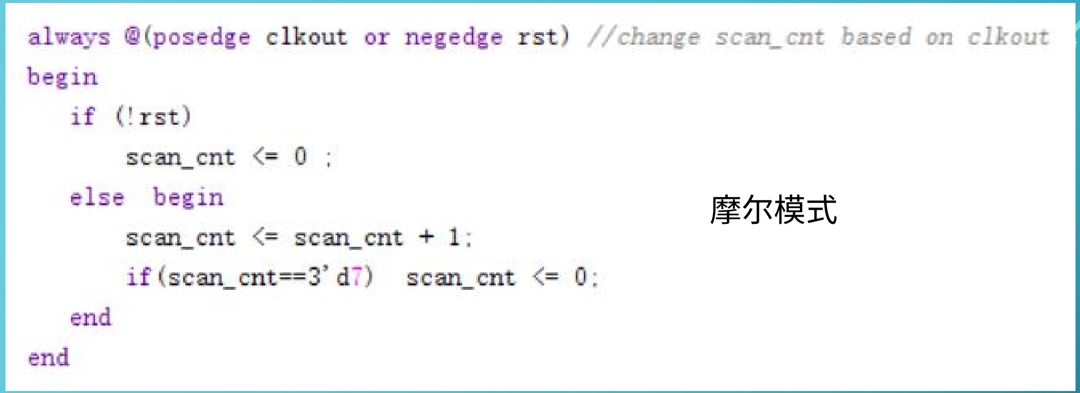
```
always @( posedge clk or negedge rst)   //frequency division : clk->clkout...

always @(posedge clkout or negedge rst) //change scan_cnt based on clkout      ...

always @( scan_cnt)         //select tube...

always @ (scan_cnt ) //decoder to display on 7-seg tube...
```



```
module scan_seg ( rst ,clk ,DIG ,Y );
    input rst ;//reset
    input clk ;//system clock 100MHz
    output [7:0] DIG    ;//bit selection
    output [7:0] Y ;// seg selection

    reg clkout ; //
    reg [31:0]cnt;
    reg [2:0]scan_cnt ;

    parameter  period= 200000;//500HZ stable
    //parameter  period= 250000;//400HZ stable
    //parameter  period= 5000000;//20HZ loop one by one
    //parameter  period= 2500000;//40HZ twenkle
    //parameter  period= 1000000;//100HZ twenkle

    reg [6:0] Y_r;
    reg [7:0] DIG_r;
    assign Y = {1'b1,(~Y_r[6:0])};//dot never light
    assign DIG =~DIG_r;
```

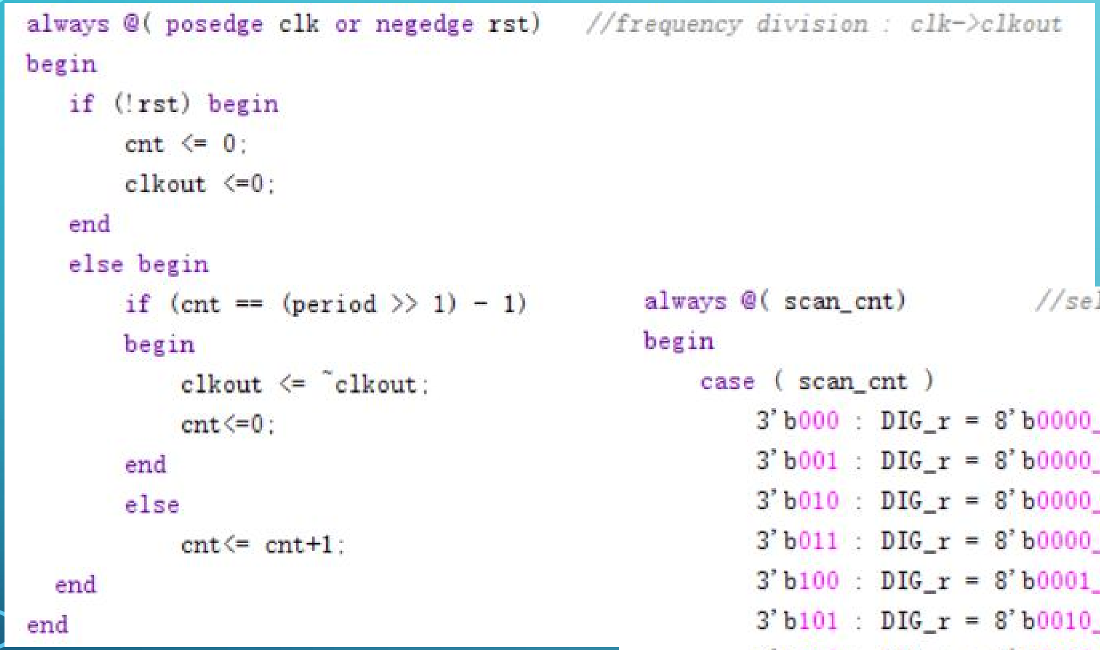
异步复位 复位不受时钟信号控制 随时都可以复位

DEMO2 ： 7-SEG TUBE

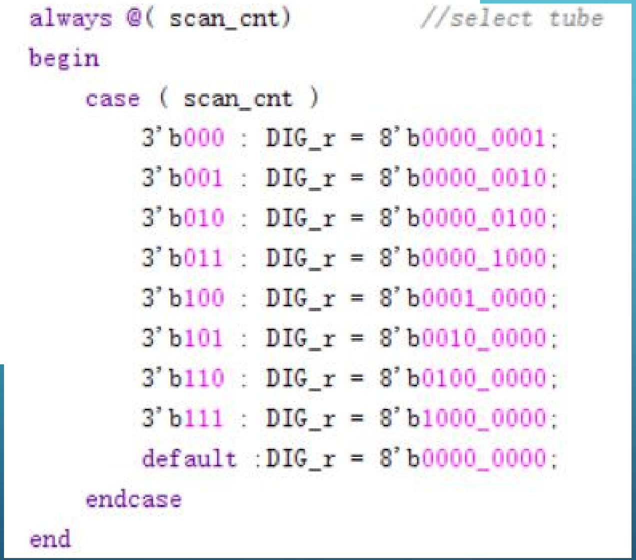


```
always @(posedge clkout or negedge rst) //change scan_cnt based on clkout
begin
   if (!rst)
       scan_cnt <= 0 ;
   else  begin
       scan_cnt <= scan_cnt + 1;
       if(scan_cnt==3'd7)  scan_cnt <= 0;
   end
end
```

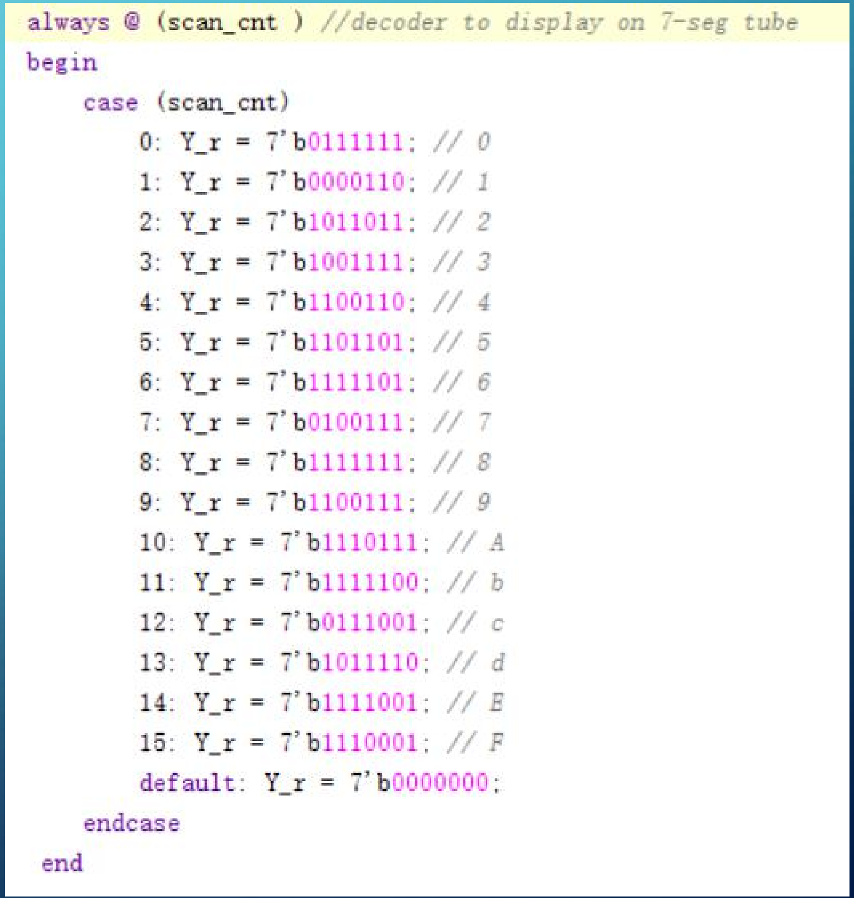
摩尔模式



```
always @( posedge clk or negedge rst)   //frequency division : clk->clkout
begin
   if (!rst) begin
       cnt <= 0;
       clkout <=0;
   end
   else begin
       if (cnt == (period >> 1) - 1)
       begin
           clkout <= ~clkout;
           cnt<=0;
       end
       else
           cnt<= cnt+1;
  end
end
```



```
always @( scan_cnt)          //select tube
begin
    case ( scan_cnt )
        3'b000 : DIG_r = 8'b0000_0001;
        3'b001 : DIG_r = 8'b0000_0010;
        3'b010 : DIG_r = 8'b0000_0100;
        3'b011 : DIG_r = 8'b0000_1000;
        3'b100 : DIG_r = 8'b0001_0000;
        3'b101 : DIG_r = 8'b0010_0000;
        3'b110 : DIG_r = 8'b0100_0000;
        3'b111 : DIG_r = 8'b1000_0000;
        default :DIG_r = 8'b0000_0000;
    endcase
end
```



```
always @ (scan_cnt ) //decoder to display on 7-seg tube
begin
    case (scan_cnt)
        0: Y_r = 7'b0111111; // 0
        1: Y_r = 7'b0000110; // 1
        2: Y_r = 7'b1011011; // 2
        3: Y_r = 7'b1001111; // 3
        4: Y_r = 7'b1100110; // 4
        5: Y_r = 7'b1101101; // 5
        6: Y_r = 7'b1111101; // 6
        7: Y_r = 7'b0100111; // 7
        8: Y_r = 7'b1111111; // 8
        9: Y_r = 7'b1100111; // 9
        10: Y_r = 7'b1110111; // A
        11: Y_r = 7'b1111100; // b
        12: Y_r = 7'b0111001; // c
        13: Y_r = 7'b1011110; // d
        14: Y_r = 7'b1111001; // E
        15: Y_r = 7'b1110001; // F
        default: Y_r = 7'b0000000;
    endcase
 end
```

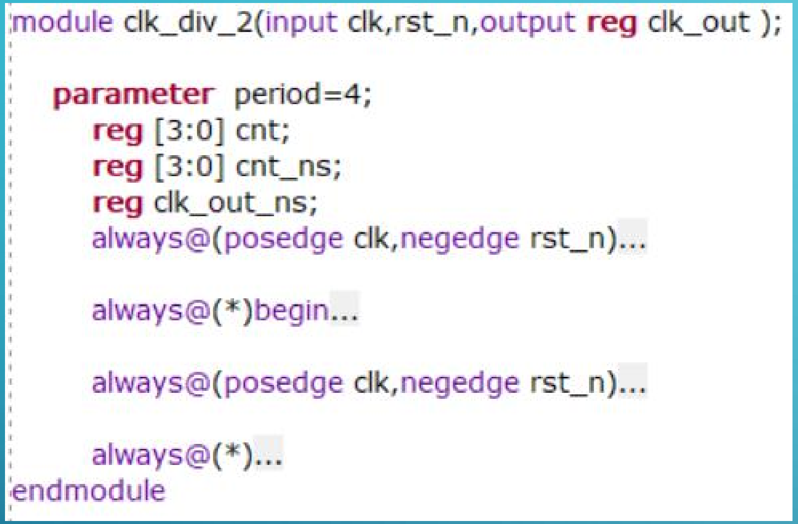
PRACTICE

- 1. Implement a ‘Breathing lamp’ on Minsys board, the ‘Breathing lamp’ here is a led which light on for 1 seconds while light off for 1 seconds.

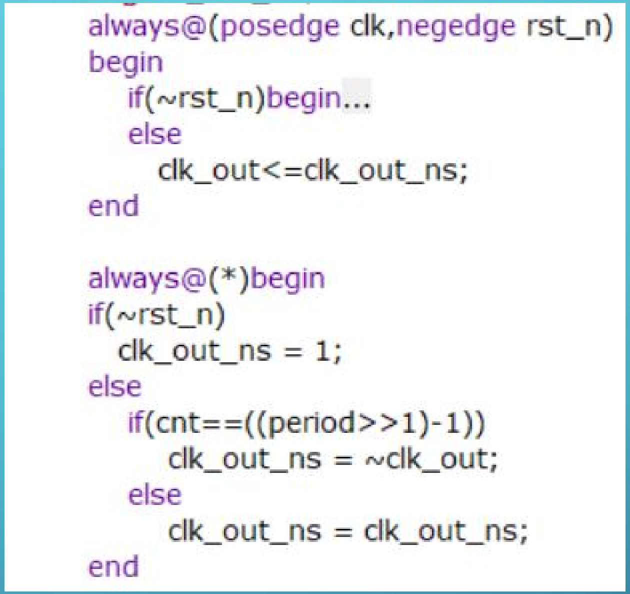
- 2. implement a Rolling subtitles showing ‘CSE’ and flowing from right to left on the tube of Minisys board.

TIPS

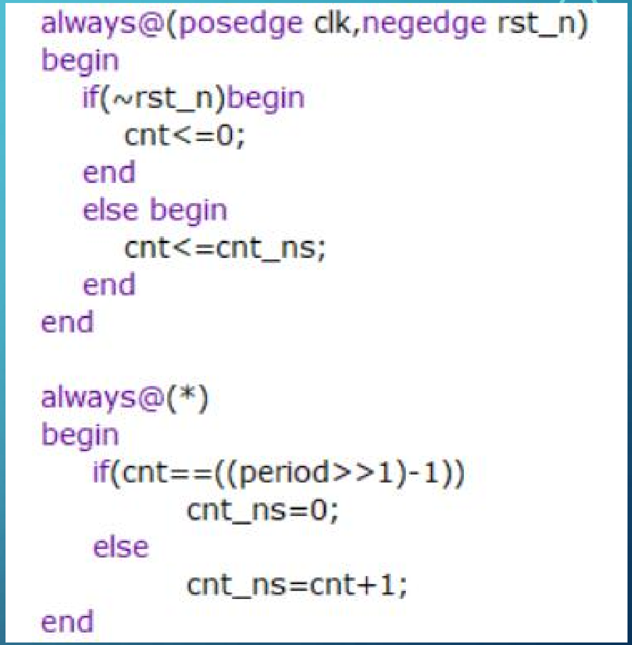
If you have any problem about frequency divider, wish this demo will help you.
2-stages is an easier way for understanding sequential circuit compared to 1-stage.



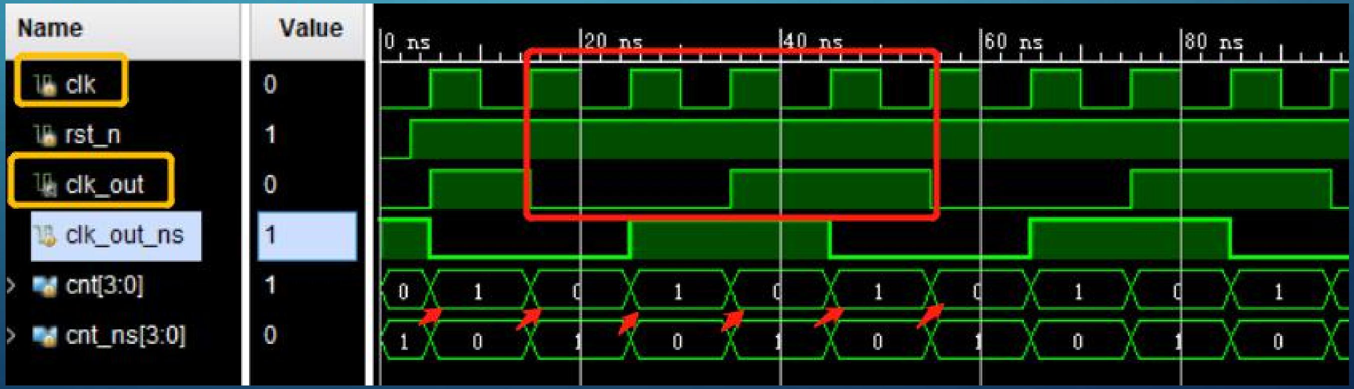
```
module clk_div_2(input clk,rst_n,output reg clk_out );

  parameter  period=4;
    reg [3:0] cnt;
    reg [3:0] cnt_ns;
    reg clk_out_ns;
    always@(posedge clk,negedge rst_n)...

    always@(*)begin...

    always@(posedge clk,negedge rst_n)...

    always@(*)...
endmodule
```



```
always@(posedge clk,negedge rst_n)
begin
   if(~rst_n)begin...
   else
      clk_out<=clk_out_ns;
end

always@(*)begin
if(~rst_n)
   clk_out_ns = 1;
else
   if(cnt==((period>>1)-1))
      clk_out_ns = ~clk_out;
   else
      clk_out_ns = clk_out_ns;
end
```



```
always@(posedge clk,negedge rst_n)
begin
   if(~rst_n)begin
      cnt<=0;
   end
   else begin
      cnt<=cnt_ns;
   end
end

always@(*)
begin
   if(cnt==((period>>1)-1))
         cnt_ns=0;
    else
         cnt_ns=cnt+1;
end
```



```
Name          Value
clk           0
rst_n         1
clk_out       0
clk_out_ns    1
cnt[3:0]      1
cnt_ns[3:0]   0
```