DIGITAL DESIGN

LAB12   FREQUENCY DIVIDER AND ITS APPLICATION (PROJECT RELATED)

WANGW6@SUSTECH.EDU.CN

LAB12

- Frequency Divider
  - Divide by even
  - Divide by odd

- Structure Design and Frequency Divider
  - Flowing light
  - 7-seg tube

- Memory(1)
  - in Verilog

WANGW6@SUSTECH.EDU.CN

CLOCK ON EGO1/MINISYS BOARD

The **100Mhz** is the most common basic clock frequency. We usually perform frequency division processing based on this frequency to obtain the clock we need.

- **EGO1** board includes a **100MHz** crystal oscillator connected to the main chip **P17** pin.
- **Minisys** board includes a **100MHz** crystal oscillator connected to the main chip **Y18** pin.

WANGW6@SUSTECH.EDU.CN

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.

WANGW6@SUSTECH.EDU.CN

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

FREQUENCY DIVIDER(N:4)

```verilog
`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
```

```verilog
`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
```

2

FREQUENCY DIVIDER(N:5)

```verilog
`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
```

1

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

```verilog
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
```

2

```verilog
`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
```


WANGW6@SUSTECH.EDU.CN

DEMO1:FLOWING LIGHT1(1)

```verilog
`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
```

always@(postedgee clk,negedge rst_n)
if(!rst_n)--
    stata<=s1;
else

        (

    always@(posedge clk)

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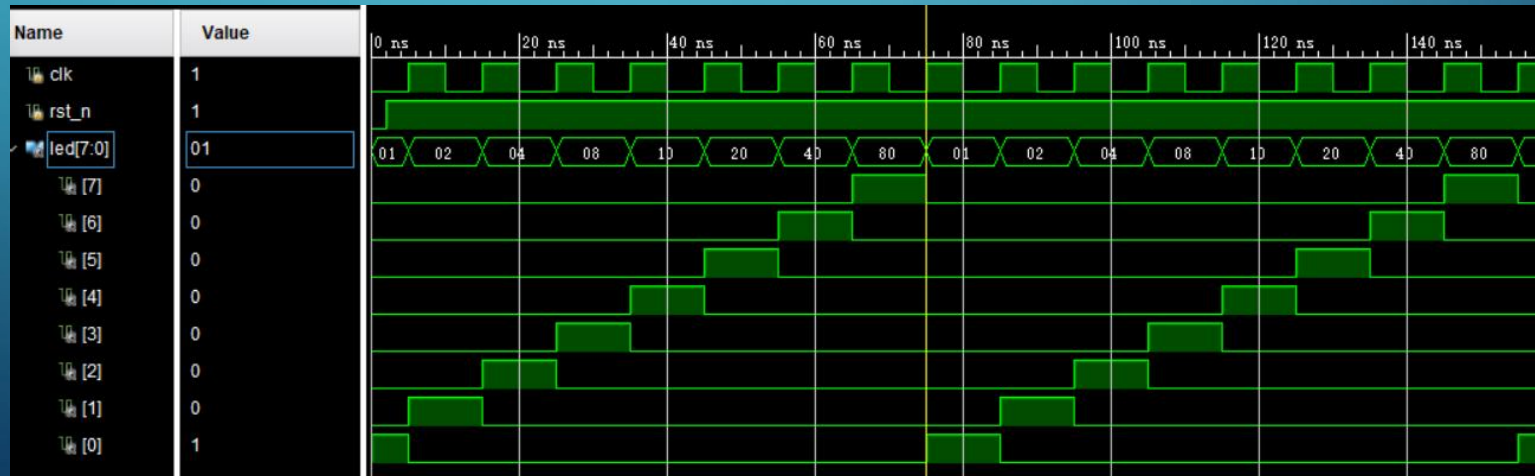
```verilog
`timescale 1ns / 1ps
module flowing_light_lite(input clk, rst_n, output[7:0] led);
    reg [7:0] light_reg;
    always@(posedge clk, negedge rst_n)
    begin
        if (!rst_n)
            light_reg<=8'b0000_0001;
        else if(light_reg == 8'b1000_0000)
            light_reg<=8'b0000_0001;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

Before creat the constraints file with the circuit, generate the bitstream file and program it on the FPGA chip of the board, think about the following question:

1. "**rst_n**" is high effective or low effective?
Is it a synchronous reset signal or an asynchronous reset signal?
Which is more suitable for the "rst_n", a **switch** or a **button** ?

2. If the "**clk**" is binded with pin "p17" of EGO1, How long does a cycle last based on the "clk"?
could it be recognized by human eyes?

3. While replace the operator "**<<**" with "**<<<**" in the code on the left hand, will the updated circuit work same as the orignal one?

DEMO2: FLOWING LIGHT2(1)

10^8

The clock frequency of 100Mhz is too fast to be recognized by human eyes. The 100Mhz clock need frequncy division.

```verilog
`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
```

```verilog
21  module counter(
22      input clk,
23      input rst_n,
24      output clk_bps
25      );
                                        50%
26      reg [13:0] cnt_first, cnt_second;
27      always@(posedge clk, negedge rst_n)
28          if( !rst_n )
29              cnt_first <= 14'd0;
30          else if (cnt_first == 14'd10000)
31              cnt_first <= 14'd0;
32          else
33              cnt_first <= cnt_first + 1'b1;
34      always@(posedge clk, negedge rst_n)
35          if( !rst_n )
36              cnt_second <= 14'd0;
37          else if (cnt_second == 14'd10000)
38              cnt_second <= 14'd0;
39          else if ( cnt_first == 14'd10000)
40              cnt_second <= cnt_second + 1'b1;
41          else
42              cnt_second <= cnt_second;
43      assign clk_bps = cnt_second == 14'd10000;
44  endmodule  cnt_second==14'd10000  clk_bps  1
```

```verilog
21  module flash_led_ctrl(
22      input clk,
23      input rst_n,
24      input dir,          dir:
25      input clk_bps,        0          1
26      output reg[7:0] led
27      );
28      always@(posedge clk or negedge rst_n)
29          if( !rst_n )
30              led <= 8'h80;    b10000000
31          else
32              case( dir )
33                  1'b0:
34                      if(clk_bps)
35                          if( led != 8'h01 )
36                              led <= led >>1'b1;  //shift right
37                          else
38                              led <= 8'h80;
39                  1'b1:
40                      if(clk_bps)
41                          if( led != 8'h80 )
42                              led <= led << 1'b1;  //shift left
43                          else
44                              led <= 8'h01;
45              endcase
46  endmodule
```

DEMO2: FLOWING LIGHT2(2)

NOTES:

1. While test the circuit by simulation, the defalut simulation time(1000ns) need to be longger.

2. While test the circuit by the board, Different constraint files are required according to the type of board(EGO1 or Minisys).
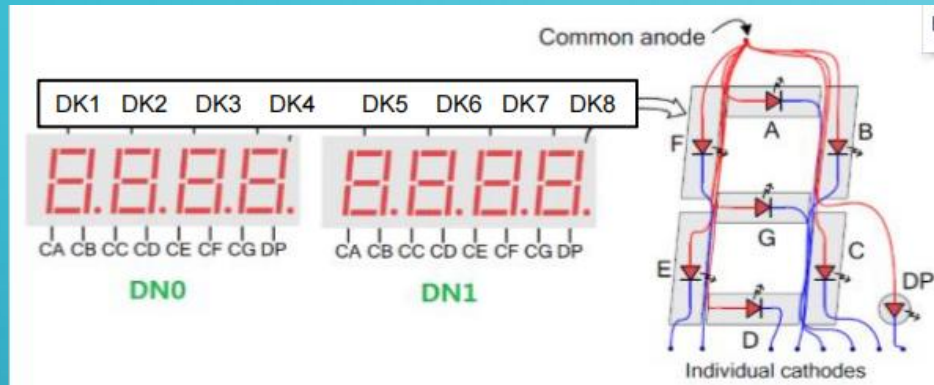
The pin of clock(100MHZ) on EGO1 board

The pin of clock(100MHZ) on Minisys board

```
set_property PACKAGE_PIN P17 [get_ports clk]
```

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

DEMO3： 7-SEG TUBE(1)

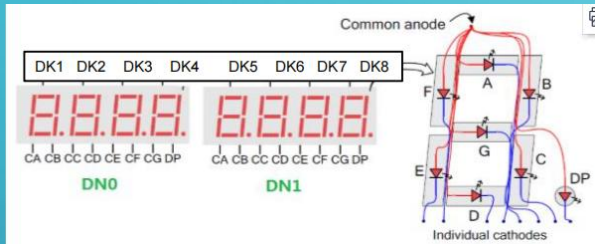7-seg Tube: There are 2 group 7seg tubes on EGO1, each group share the same ctroller bits.

TASK:
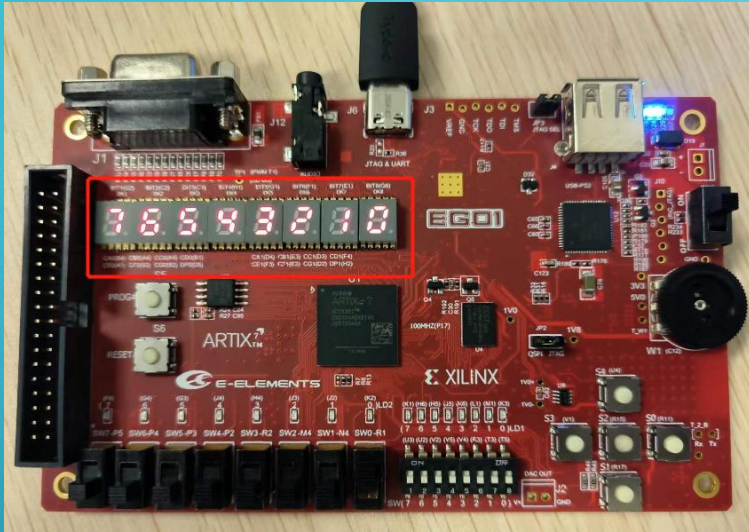Display 8 different numbers on 8 7seg tubes of EGO1 board.
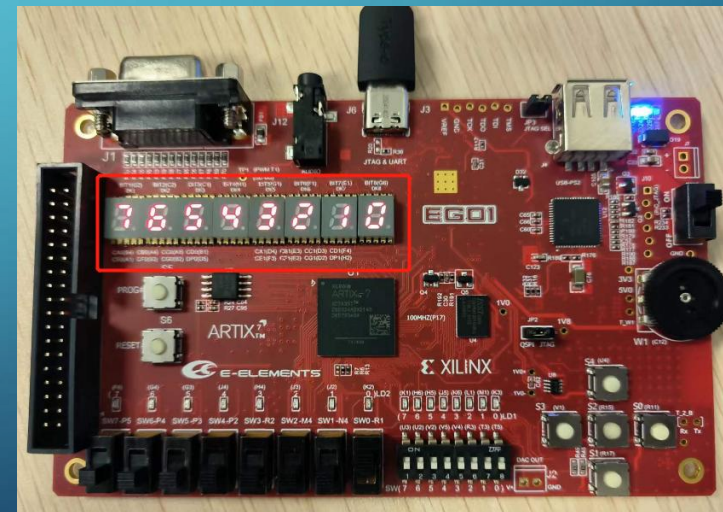
DEMO3： 7-SEG TUBE(2)

The code on the right hand is from lab8, the circuit reads the data from switch and show the number on the 7seg tubes .


```
module light_7seg_ego1(input [3:0]sw, output reg [7:0] seg_out, output [7:0] seg_en);
    assign seg_en = 8'hff;
    always @ *
        case(sw)
            4'h0: seg_out = 8'b1111_1100; //0
            4'h1: seg_out = 8'b0110_0000; //1
            4'h2: seg_out = 8'b1101_1010; //2
            4'h3: seg_out = 8'b1111_0010; //3
            4'h4: seg_out = 8'b0110_0110; //4
            4'h5: seg_out = 8'b1011_0110; //5
            4'h6: seg_out = 8'b1011_1110; //6
            4'h7: seg_out = 8'b1110_0000; //7
            4'h8: seg_out = 8'b1111_1110; //8
            4'h9: seg_out = 8'b1110_0110; //9
            4'ha: seg_out = 8'b1110_1110; //A
            4'hb: seg_out = 8'b0011_1110; //B
            4'hc: seg_out = 8'b1001_1100; //C
            4'hd: seg_out = 8'b0111_1010; //D
            4'he: seg_out = 8'b1001_1110; //E
            4'hf: seg_out = 8'b1000_1110; //F
            default: seg_out = 8'b0000_0001;
        endcase
endmodule
```

DEMO2： 7-SEG TUBE(3)

Display different numbers on 8 7seg tubes of EGO1 board.

Tips:
1. There are 2 group 7seg tubes on EGO1, each group share the same ctroller bits.
2. The code of lab8 could be used in this exercise.

```verilog
module scan_seg( rst_n, clk, seg_en, seg_out0, seg_out1 );
    input rst_n ; //reset: low effective
    input clk ; //system clock 100MHZ
    output reg [7:0] seg_en : // selection on tube 0-7
    output [7:0] seg_out0:    // seg0 selection
    output [7:0] seg_out1:    // seg1 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

    always @(posedge clk, negedge rst_n) //freqency division : clk -> clk_out...
    always @(posedge clkout, negedge rst_n) //change scan_cnt based on clkout...
    always @ ( scan_cnt)  //select tube...
    //module light_7seg_ego1 (input [3:0]sw, output reg [7:0] seg_out, output [7:0] seg_en);
    wire [7:0] useless_seg_en0, useless_seg_en1:
    light_7seg_ego1 u0(. sw({1'b0, scan_cnt}), . seg_out(seg_out0), . seg_en(useless_seg_en0));
    light_7seg_ego1 u1(. sw({1'b0, scan_cnt}), . seg_out(seg_out1), . seg_en(useless_seg_en1));
endmodule
```

DEMO3 : 7-SEG TUBE(4)

In this situation, should the clock frequency be as fast as possible, or as slow as possible ?

```verilog
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


always @(posedge clk, negedge rst_n) //freqency division : clk -> clk_out
begin
    if(!rst_n) 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
```

```verilog
always @(posedge clkout, negedge rst_n) //change scan_cnt based on clkout
begin
    if(~rst_n)
        scan_cnt <= 0;
    else begin
        if(scan_cnt==3'd7)
            scan_cnt <= 0;
        else
            scan_cnt <= scan_cnt + 1;
    end
end
always @ ( scan_cnt )  //select tube
begin
    case (scan_cnt)
        3'b000 : seg_en = 8'h01;
        3'b001 : seg_en = 8'h02;
        3'b010 : seg_en = 8'h04;
        3'b011 : seg_en = 8'h08;
        3'b100 : seg_en = 8'h10;
        3'b101 : seg_en = 8'h20;
        3'b110 : seg_en = 8'h40;
        3'b111 : seg_en = 8'h80;
        default: seg_en = 8'h00;
    endcase
end
```

MEMORY(1)

- Memory can be seen as a set of registers with the same bit width.

  Modeling memory by building arrays of reg variables, and addressing each unit of the array by array index

- Definition:

  reg [n-1:0] memory name [m-1:0];   // there are m unit in memory, the size of each unit in the memory is n .

- Notes:

  - A n-bit register can be assigned in an assignment statement, but a full memory CAN NOT.

  - If you need to read and write a storage unit in memory, you must specify the address of the unit in memory.

  reg [2:0] Mema [4:0];   // define a memory named Mema which has 5 memory units, each with a bit width of 3 bits.

  Mema [1]= 3'b101;   // assign 3'b101 to Mema [1] unit in Mema

```verilog
module test(
    A, C0, C1, C2
    );
    input [2:0] A;
    output [1:0] C0, C1, C2;
    reg[1:0] B [2:0];
    assign {C0, C1, C2} = {B[0], B[1], B[2]};
    always @(A)
    if(A)
    begin
        B[0] = 2'b11;
        B[1] = 2'b10;
        B[2] = 2'B01;
    end
    else
    begin
        B[0] = 2'b00;
        B[1] = 2'b00;
        B[2] = 2'B00;
    end
endmodule
```

```verilog
module test(
    A, C0, C1, C2
    );
    input [2:0] A;
    output [1:0] C0, C1, C2;
    reg[1:0] B [2:0];
    assign {C0, C1, C2} = {B[0], B[1], B[2]};
    always @(A)
    if(A)
    begin
        {B[0], B[1], B[2]} = 6'b011011;
        /*B[0] = 2'b11;
        B[1] = 2'b10;
        B[2] = 2'B01;*/
    end
    else
    begin
        {B[0], B[1], B[2]} = 6'b0;
        /*B[0] = 2'b00;
        B[1] = 2'b00;
        B[2] = 2'B00;*/
    end
endmodule
```
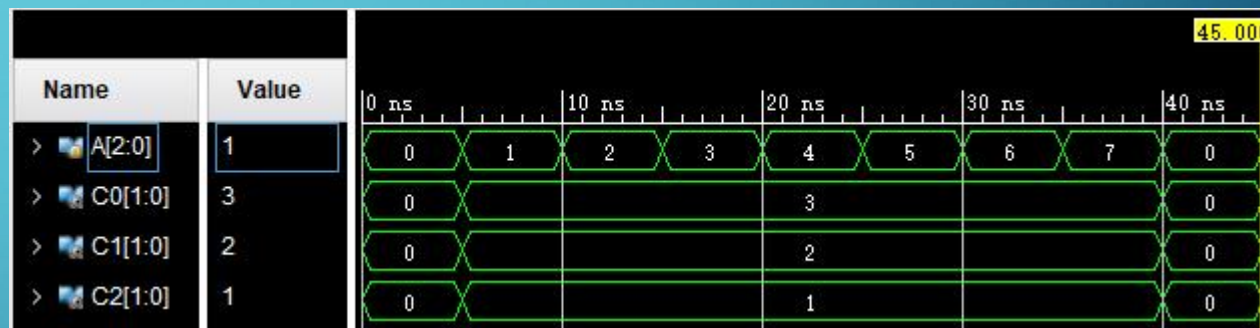
PRACTICES

- 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. There is a buzzer on the EGO1/Minisys board, implement a circuit to Let the sound of buzzer be heard by human beings.

- tips: while using EGO1 board, a headphones is needed to connect with the board for listening.

- 3. The starting switch of the induction cooker is equipped with a child lock button. The button will not take effect until it is pressed continuously for 2 second. Within 1 second of the child lock button taking effect, the induction cooker can be started only after the start switch is turned on, otherwise the induction cooker cannot be started. Implement the circuit to turn on the induction cooker.

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.

```verilog
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
```

```verilog
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
```

```verilog
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
```