# DIGITAL DESIGN LAB12 FREQUENCY DIVIDER AND ITS APPLICATION (PROJECT RELATED) 2022 FALL TERM @ CSE . SUSTECH

LAB12

- Frequency Divider
  - Divide by even
  - Divide by odd
- Structure Design and Frequency Divider
  - Flowing light
  - 7-seg tube

# 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.

## 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 <u>flipflops</u> 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.

## 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
```

```
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</pre>
```





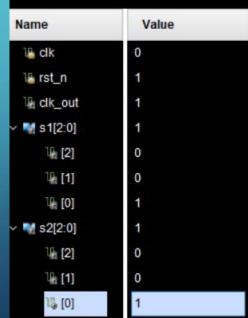
# FREQUENCY DIVIDER(N:5)

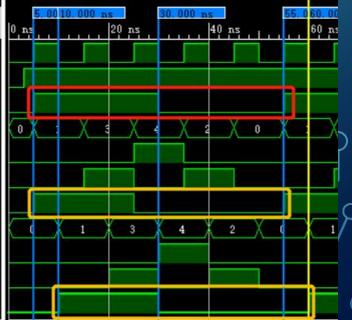
```
//reg [25:0]cnt:...
reg [2:0] step1, step2;
always@(posedge clk)begin
                                                          assign clk_out=step1[0]|step2[0];
    if ("rst_n) begin
        step1 <= 3' b000;
    end
    else begin
        case (step1)
                                      always @(negedge clk, negedge rst_n)
            3'b000: step1<=3'b001;
                                          if (rst n)
            3'b001: step1<=3'b011;
                                               step2<=3' b000;
            3'b011: step1<=3'b100:
                                           else
            3'b100: step1<=3'b010:
                                               case(step2)
            3'b010: step1<=3'b000:
                                                       3'b000: step2<=3'b001;
            default: step1 <= 3' b000;
                                                       3'b001: step2<=3'b011;
            endcase
                                                       3'b011: step2<=3'b100;
    end
                                                       3'b100: step2<=3'b010;
end
                                                       3'b010: step2<=3'b000;
                                                       default: step2<=3' b000;
                                                       endcase
                                      endmodule
```

module clock\_div(input clk, rst\_n, output reg clk\_out );

timescale lns / lps

```
timescale Ins / 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
```

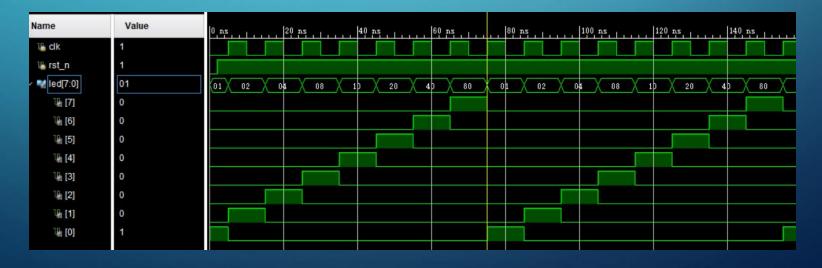




## DEMO1:FLOWING LIGHT1(1)

```
timescale lns / lps
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</pre>
```

```
timescale lns / lps
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
```



# DEMO1:FLOWING LIGHT1(2)

```
rimescale lns / lps
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</pre>
```

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)

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

```
timescale ins / ips
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
```

```
module counter (
      input clk,
      input rst n.
      output clk bps
          reg [13:0] cnt_first, cnt_second;
          always@(posedge clk, negedge rst_n)
              if(!rst n)
                  cnt_first <= 14' d0;
              else if (cnt first = 14' d10000)
                  cnt first <= 14' d0:
31
              else
                  cnt_first <= cnt_first + 1' b1;
          always@(posedge clk, negedge rst_n)
35
              if(!rst n)
                   cnt_second <= 14' d0;
              else if (cnt second = 14' d10000)
                   cnt second <= 14' d0;
38
              else if ( cnt first = 14' d10000)
                   cnt_second <= cnt_second + 1'b1;
              else
                  cnt second <= cnt second;
          assign clk bps = cnt_second = 14' d10000;
      endmodule
```

```
module flash led ctrl(
         input clk,
         input rst n,
         input dir,
24
         input clk bps,
25
         output reg[7:0] led
26
27
28
          always@(posedge clk or negedge rst_n)
             if(!rst n)
29
                 led (= 8' h80:
30
             else
31
                 case (dir )
32
                    1' b0:
33
                        if(clk bps)
34 -
                            if( led != 8' h01 )
35 -
                                led <= led >>1'b1: //shift right
36
37
                             else
                                led (= 8' h80:
38
                     1' b1:
                        if(clk bps)
40 -
                            if( led != 8' h80 )
                                led <= led << 1'b1: //shift left
42
                             else
43
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).

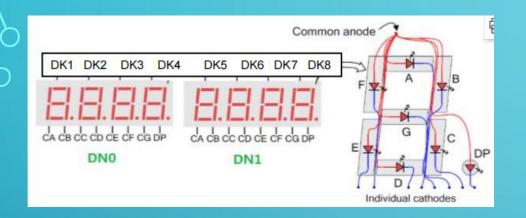
The pin of clock(100MHZ) on EGO1 board

set\_property PACKAGE\_PIN P17 [get\_ports clk]

The pin of clock(100MHZ) on Minisys board

set\_property PACKAGE\_PIN Y18 [get\_ports clk]

## DEMO3: 7-SEG TUBE(1)

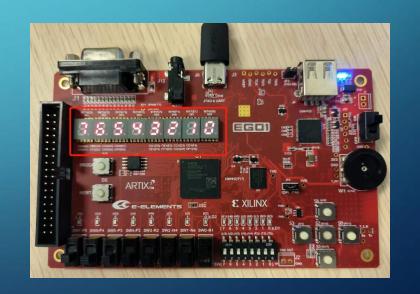


7-seg Tube: There are 2 group 7seg tubes on EGO1, each group share the same ctroller bits.

#### TASK:

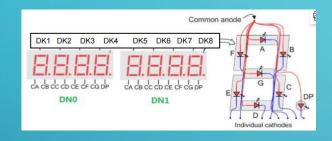
Display 8 different numbers on 8 7seg tubes of EGO1 board.





## DEMO3: 7-SEG TUBE(2)

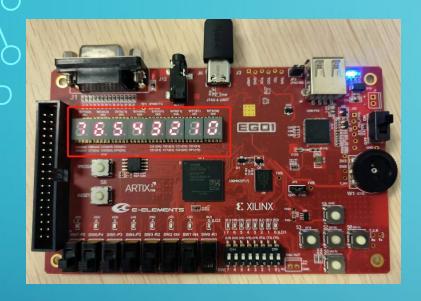
The code on the right hand is from lab8, the circuit reads the data from switch and show the number on the 7seg tubes.





```
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' b00000_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.

```
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) //frequency 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?

```
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) //frequency 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
```

```
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
```

## **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 listenning.

• 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.

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

