



# DIGITAL DESIGN

LAB11 FREQUENCY DIVIDER

2022 SUMMER TERM

# LAB11

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

# CLOCK ON EGO1 BOARD

- **EGO1** board includes a **100MHz** crystal oscillator connected to the main chip **P17** 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  $f_{in}$ , and generates an output signal of a frequency  $f_{out}$ :

$$f_{out} = f_{in}/n \text{ (n is an integer).}$$

- 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.

# 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
    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
```





# 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
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
    end
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

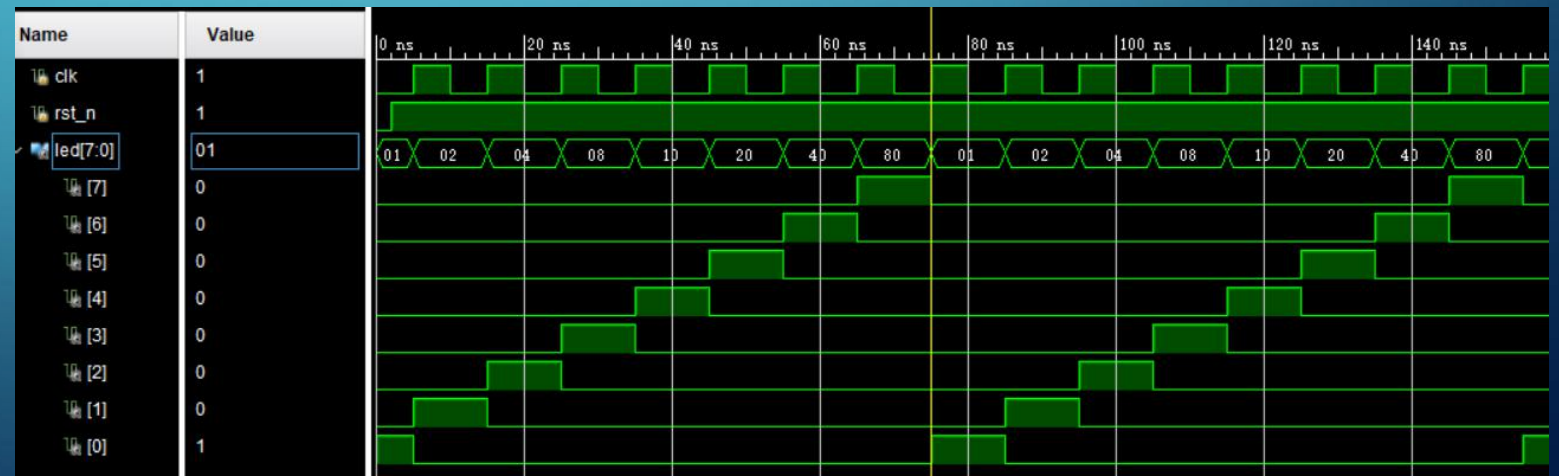
```



# 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
```



# DEMO1 (FLOWING LIGHT2)

```
module flash_led_top(  
    input rst_n,  
    input clk,  
    input dir_sw,  
    output [7:0] led  
);  
  
wire clk_bps;  
counter flash_led_0(  
    .rst_n(rst_n),  
    .clk(clk),  
    .clk_bps(clk_bps)  
);  
  
flash_led_ctrl(  
    .rst_n(rst_n),  
    .clk(clk),  
    .dir(dir_sw),  
    .clk_bps(clk_bps),  
    .led(led)  
);  
endmodule
```

```
module flash_led_ctrl(  
    input rst_n,  
    input clk,  
    input dir,  
    input clk_bps,  
    output reg [7:0] led  
);  
  
always @(posedge clk or negedge rst_n)  
begin  
    if(!rst_n)  
        led <= 8'h30;  
    else  
        case(dir)  
            1'b0://from left to right  
                if(clk_bps)  
                    if(led != 8'd1)  
                        led <= led >> 1;  
                    else  
                        led <= 8'h30;  
            1'b1://from right to left  
                if(clk_bps)  
                    if(led != 8'h30)  
                        led <= led << 1;  
                    else  
                        led <= 8'd01;  
        endcase  
    end  
endmodule
```

```
module counter(  
    input rst_n,  
    input clk,  
    output clk_bps  
);  
  
reg [13:0] cnt_first, cnt_second;  
always @(posedge clk or posedge rst_n)  
begin  
    if(!rst_n)  
        cnt_first <= 0;  
    else  
        if(cnt_first == 14'd10000)  
            cnt_first <= 0;  
        else  
            cnt_first <= cnt_first + 1;  
    end  
  
always @(posedge clk or posedge rst_n)  
begin  
    if(!rst_n)  
        cnt_second <= 0;  
    else  
        if(cnt_second == 14'd10000)  
            cnt_second <= 0;  
        else if(cnt_first == 14'd10000)  
            cnt_second <= cnt_second + 1;  
    end  
  
assign clk_bps = (cnt_second == 14'd10000)?1'b1:1'b0;  
endmodule
```



# DEMO1 (FLOWING LIGHT2)



Test on EGO1 board

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

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

# DEMO2 : 7-SEG TUBE DISPLAY

```
module scan_seg(  
    input rst_n,  
    input clk,  
    output [7:0] bit_sel,  
    output [7:0] Y_0,  
    output [7:0] Y_1  
);  
  
    reg clkout;  
    reg [31:0] cnt;  
    reg [2:0] scan_cnt;  
    reg [6:0] Y_reg;  
    reg [7:0] bit_sel_reg;  
  
    parameter period = 200000; //500HZ stable  
    //parameter period = 250000; //400HZ stable  
    //parameter period = 5000000; //20HZ loop one by one  
    //parameter period = 2500000; //40HZ twinkle  
    //parameter period = 1000000; //100HZ twinkle  
  
    assign Y_0 = { Y_reg, 1'b0 };  
    assign Y_1 = { Y_reg, 1'b0 };  
    assign bit_sel = bit_sel_reg;
```

```
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...
```



# DEMO2 : 7-SEG TUBE

```
always@(posedge clk or negedge rst_n)
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
end
```

```
always @(posedge clkout or negedge rst_n)
begin
    if(!rst_n)
        scan_cnt <= 0;
    else begin
        if(scan_cnt == 3'b111)
            scan_cnt <= 0;
        else
            scan_cnt <= scan_cnt + 1;
        end
    end
end
```

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

```
always @(scan_cnt)
begin
    case(scan_cnt)
        0: Y_reg = 8'b1111_110; //0
        1: Y_reg = 8'b0110_000; //1
        2: Y_reg = 8'b1101_101; //2
        3: Y_reg = 8'b1111_001; //3
        4: Y_reg = 8'b0110_011; //4
        5: Y_reg = 8'b1011_011; //5
        6: Y_reg = 8'b1011_111; //6
        7: Y_reg = 8'b1110_000; //7

        8: Y_reg = 8'b1111_111; //8
        9: Y_reg = 8'b1110_011; //9
        10: Y_reg = 8'b1110_111; //a
        11: Y_reg = 8'b0011_111; //b
        12: Y_reg = 8'b0001_101; //c
        13: Y_reg = 8'b0111_101; //d
        14: Y_reg = 8'b1001_111; //e
        15: Y_reg = 8'b1000_111; //f
        default: Y_reg = 7'b0000000;
    endcase
end
```

# PRACTICE 1

- Implement a Rolling subtitles showing 'CSE' and flowing from right to left on the 7-seg tube of EGO1 board.