

课程名称：EDA 技术综合设计

设计报告名称：设计四 移位寄存器设计

班级：通信 214

姓名：王峤宇

学号：214022

一、设计内容及原理

设计内容

寄存器是用来存储二进制信息的电路。有存储功能和移位功能，是最基本的时序逻辑电路设计元件。

基础任务

设计任务: 数码管显示时，肉眼能分辨的频率在 30 多赫兹左右最好，对 100MHz 的时钟分频，分频到肉眼可见的频率范围。输出频率时钟由发光二极管显示，同时由数码管显示分频系数。需要给出计算过程。**时钟分频:** 分频采用 DDS 中的频率控制字实现任意分频效果的思路完成，DDS 中的通过频率控制字控制相位累加器的累加步长，实现任意频率相位信号输出。具体计算如下：

假设 FPGA 的基准频率为 100MHz:

$$f_c = 1 * 10^8 (Hz) \quad (1)$$

假定计数器为 32 位计数器，总的频率控制字上限为计数器计数值的一半，也就是最小为 2 分频，最大输出为输入时钟的一半，K 为频率控制字，计算过程如下：

$$N = 2^{32} \quad (2)$$

$$f_o = \frac{f_c * K}{N} = 0.023283 * K \quad (3)$$

可以实现频率分辨率为 0.024 的任意频率输出。最小频率为 0.023283Hz 输出，最大频率为输入时钟的一半，即 50MHz。通过可以得到，为了得到大概 30Hz 的输出，K 值取为：

$$K = \frac{f_o}{0.023283} = 1288 \quad (4)$$

此时的理论输出频率为:29.9886Hz, 实现输入信号的 3,334,601.9 分频。

$$f_o = \frac{f_c * K}{N} = 0.023283 * 1288 = 29.9886 Hz \quad (5)$$

提高内容

设计任务: 设计一个偶分频的通用分频器（通用值 N），输出频率时钟由发光二极管显示，同时由数码管显示分频系数。N 值由外部拨码开关输入。**偶数分频器设计:** 数倍分频是最简单的一种分频模式，可以直接通过计数器计数实现。计数器由输入时钟的上升沿或下降沿驱动，从 0 开始，向上计数当计数值达到 N - 1 时清零，实现一个周期，其中小于 N/2 的部分和小于 N 的部分对称，通过组合逻辑判断计数值就可以实现偶数倍的分频。

拓展任务

任务要求: 设计一个奇分频的通用分频器 (通用值 N)，输出频率时钟由发光二极管显示，同时由数码管显示分频系数。 N 值由外部拨码开关输入。**奇数分频器设计:** 奇数分频器通常由两个计数器实现，将产生的两个时钟进行组合逻辑运算得到相应的 50% 占空比的分频信号输出。

两个计数器输出进行或运算

一个计数器计数信号的上升沿，对信号进行 N 分频，另一个计数器采样信号的下降沿，同样对信号进行 N 分频，产生不为 50% 的波形，以 3 分频为例，计数器包含计数值 0, 1, 2 三个值，通过组合逻辑电路使其在 0, 1 状态时输出 clk 为高电平，值为 2 时输出 clk 为低电平，这样就会导致高电平的时间大于低电平时间一个周期，此时通过另一个下降沿采样计数器同理产生相应的时钟，由于对下降沿采样，在输入时钟严格保证 50% 占空比时，第二个时钟相比第一个是时钟延迟了半个时钟周期，对连个时钟进行或操作得到的结果，使得高电平时间增加半个周期，低电平时间减少半个周期，这样就实现了奇数分频中奇数周期的平分，使得高电平为 1.5 个周期，低电平也为 1.5 个周期的 50% 占空比分频。

设计完整系统

针对基础任务、提高任务和拓展任务的三者进行结合设计，设计得到的该系统中，支持 8 位二进制 0 255 的分频系数设置，当选择 0 或 1 时，为不分频，输出结果为 30Hz 的时钟，数码管显示对应的分频系数，直接进行通过拨码开关设置指定的分频系数，系统对 30Hz 的时钟进行分频，方便观察，实现奇数分频和偶数分频。

二、设计过程

基础任务

通过 DDS 中相位累加器实现的任意分频器源文件如下:

Listing 1: 任意分频器代码

```
1 module clk_div_32bit(  
2     input clk,  
3     input rstn,  
4     input [31:0] step_freq,  
5     output clk_out  
6 );  
7 /* 相位累加器 */  
8 reg [31:0] cnt = 32'b0;  
9 always@(posedge clk or negedge rstn) begin  
10     if(!rstn)  
11         cnt <= 32'b0;
```

```

12         else
13             cnt <= cnt + step_freq;
14     end
15     /* 相位波形输出，由于实现的是分频，输出为50占空比的方波 */
16     reg cnt_equal;
17     always @(posedge clk or negedge rstn) begin
18         if (!rstn) begin
19             cnt_equal <= 1'b0;
20         end else if(cnt < 32'h8000_0000) begin
21             cnt_equal <= 1'b0;
22         end else begin
23             cnt_equal <= 1'b1;
24         end
25     end
26     assign clk_out = cnt_equal;
27 endmodule

```

实现任意分频, 同时显示分频系数的 top 文件代码如下, 分频系数固定, 直接采用数码管输出。

Listing 2: 基础任务 top 源文件

```

1 module clk_div_top(
2     input clk,                /* 100MHz */
3     input rst,                /* 低电平异步复位 */
4     input [7:0] div_coe,      /* 输入的四位整数加四位小数的二进制实数 */
5     output [7:0] sseg1, sseg2, /* 八段数码管 */
6     output [3:0] an1, an2,    /* 片选信号 */
7     output led_out           /* 闪烁的LED灯 */
8 );
9 /* 分频, */
10 clk_div_32bit  clk_div_32bit_inst (
11     .clk(clk),
12     .rstn(!rst),
13     .step_freq(32'd1288),
14     .clk_out(led_out)
15 );
16 /* 第一个四位数码管 */
17 scan_led_hex_disp_4  scan_led_hex_disp_4_inst_0 (
18     .clk(clk), .rst(rst),
19     .hex0(4'd3), .hex1(4'd3), .hex2(4'd3), .hex3(4'd4), .dp(4'b0000),
20     .an(an1), .sseg(sseg1)
21 );
22
23 /* 第二个四位数码管 */
24 scan_led_hex_disp_4  scan_led_hex_disp_4_inst_1 (
25     .clk(clk), .rst(rst),
26     .hex0(4'd6), .hex1(4'd0), .hex2(4'd1), .hex3(4'd9), .dp(4'b0010),
27     .an(an2), .sseg(sseg2)
28 );
29 endmodule

```

Top 系统仿真文件如下:

Listing 3: top 仿真文件

```

1 module clk_div_top_tb_1;
2     reg clk;
3     reg rst;
4     reg [7:0] div_coe;
5     wire [7:0] sseg1, sseg2;
6     wire [3:0] an1, an2;
7     wire led_out;
8
9     initial begin
10         div_coe = 8'b0;
11         clk = 0;
12         rst = 0;
13         #100;
14         rst = 1;
15         #10;
16         rst = 0;
17     end
18
19     clk_div_top clk_div_top_inst (
20         .clk(clk), .rst(rst),
21         .div_coe(div_coe),
22         .sseg1(sseg1), .sseg2(sseg2),
23         .an1(an1), .an2(an2),
24         .led_out(led_out)
25     );
26
27     always #5 clk = ! clk ;
28 endmodule

```

基础任务、提高任务、拓展任务共用一套约束文件，都使用到八个拨码开关和数码管以及 LED 灯，约束文件如下所示：

Listing 4: 基础任务、提高任务、拓展任务约束文件

```

1 set_property -dict{PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports clk]
2 set_property -dict{PACKAGE_PIN R11 IOSTANDARD LVCMOS33} [get_ports rst]
3 set_property -dict{PACKAGE_PIN K2 IOSTANDARD LVCMOS33} [get_ports led_out]
4 set_property -dict{PACKAGE_PIN P5 IOSTANDARD LVCMOS33} [get_ports {div_coe[7]}]
5 set_property -dict{PACKAGE_PIN P4 IOSTANDARD LVCMOS33} [get_ports {div_coe[6]}]
6 set_property -dict{PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports {div_coe[5]}]
7 set_property -dict{PACKAGE_PIN P2 IOSTANDARD LVCMOS33} [get_ports {div_coe[4]}]
8 set_property -dict{PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports {div_coe[3]}]
9 set_property -dict{PACKAGE_PIN M4 IOSTANDARD LVCMOS33} [get_ports {div_coe[2]}]
10 set_property -dict{PACKAGE_PIN N4 IOSTANDARD LVCMOS33} [get_ports {div_coe[1]}]
11 set_property -dict{PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports {div_coe[0]}]
12 set_property -dict{PACKAGE_PIN G2 IOSTANDARD LVCMOS33} [get_ports {an1[0]}]
13 set_property -dict{PACKAGE_PIN C2 IOSTANDARD LVCMOS33} [get_ports {an1[1]}]
14 set_property -dict{PACKAGE_PIN C1 IOSTANDARD LVCMOS33} [get_ports {an1[2]}]
15 set_property -dict{PACKAGE_PIN H1 IOSTANDARD LVCMOS33} [get_ports {an1[3]}]
16 set_property -dict{PACKAGE_PIN G6 IOSTANDARD LVCMOS33} [get_ports {an2[3]}]
17 set_property -dict{PACKAGE_PIN E1 IOSTANDARD LVCMOS33} [get_ports {an2[2]}]

```

```

18 set_property -dict{PACKAGE_PIN F1 IOSTANDARD LVCMOS33} [get_ports {an2[1]}]
19 set_property -dict{PACKAGE_PIN G1 IOSTANDARD LVCMOS33} [get_ports {an2[0]}]
20 set_property -dict{PACKAGE_PIN D5 IOSTANDARD LVCMOS33} [get_ports {sseg1[7]}]
21 set_property -dict{PACKAGE_PIN B4 IOSTANDARD LVCMOS33} [get_ports {sseg1[6]}]
22 set_property -dict{PACKAGE_PIN A4 IOSTANDARD LVCMOS33} [get_ports {sseg1[5]}]
23 set_property -dict{PACKAGE_PIN A3 IOSTANDARD LVCMOS33} [get_ports {sseg1[4]}]
24 set_property -dict{PACKAGE_PIN B1 IOSTANDARD LVCMOS33} [get_ports {sseg1[3]}]
25 set_property -dict{PACKAGE_PIN A1 IOSTANDARD LVCMOS33} [get_ports {sseg1[2]}]
26 set_property -dict{PACKAGE_PIN B3 IOSTANDARD LVCMOS33} [get_ports {sseg1[1]}]
27 set_property -dict{PACKAGE_PIN B2 IOSTANDARD LVCMOS33} [get_ports {sseg1[0]}]
28 set_property -dict{PACKAGE_PIN H2 IOSTANDARD LVCMOS33} [get_ports {sseg2[7]}]
29 set_property -dict{PACKAGE_PIN D4 IOSTANDARD LVCMOS33} [get_ports {sseg2[6]}]
30 set_property -dict{PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports {sseg2[5]}]
31 set_property -dict{PACKAGE_PIN D3 IOSTANDARD LVCMOS33} [get_ports {sseg2[4]}]
32 set_property -dict{PACKAGE_PIN F4 IOSTANDARD LVCMOS33} [get_ports {sseg2[3]}]
33 set_property -dict{PACKAGE_PIN F3 IOSTANDARD LVCMOS33} [get_ports {sseg2[2]}]
34 set_property -dict{PACKAGE_PIN E2 IOSTANDARD LVCMOS33} [get_ports {sseg2[1]}]
35 set_property -dict{PACKAGE_PIN D2 IOSTANDARD LVCMOS33} [get_ports {sseg2[0]}]

```

Utilization				Post-Synthesis	Post-Implementation
				Graph	Table
Resource	Utilization	Available	Utilization %		
LUT	9	20800	0.04		
FF	48	41600	0.12		
IO	27	210	12.86		
BUFG	1	32	3.13		

图 1: 资源使用

该部分资源使用如图 1所示, 使用资源极少, 但实现了极高分辨率的任意频率生成。

提高任务

偶数分频器设计源文件如下:

Listing 5: 通用偶数分频器源文件

```

1 module clk_div_even(
2     input clk,
3     input rst,
4     input [7:0] div_N,
5     output clk_out
6 );
7 /* 操作 cnt */
8 reg [7:0] cnt;
9 always @(posedge clk, posedge rst) begin
10     if (rst) begin
11         cnt <= 8'b0;
12     end
13     else if (cnt < {div_N[7:1], 1'b0} - 1'b1) begin

```

```

14         cnt <= cnt + 1'b1;
15     end
16     else begin
17         cnt <= 8'b0;
18     end
19 end
20 assign clk_out = (cnt < {1'b0, div_N[7:1]}) ? 1'b1 : 1'b0;
21 endmodule

```

偶数分频器设计的过程中, 首先要保证时序逻辑的复位功能有效, 提供对复位的支持, 实现异步的高电平复位。计数器在计数的过程中, 总计计数分频系数次的输入周期, 对应的计数器值为 0 N-1, 之后利用 assign 设计计数值映射为输出时钟 0 和 1 的逻辑电路即可。

Listing 6: 偶数分频器仿真激励

```

1 initial begin
2     div_coe = 8'd12;
3     clk = 0;
4     rst = 0;
5     #100;
6     rst = 1;
7     #10;
8     rst = 0;
9 end

```

拓展任务

奇数分频器采用两个计数器和或运算实现, 具体原理可见设计内容分析。实现的奇数分频其源文件如下:

Listing 7: 通用奇数分频器设计源文件

```

1 module clk_div_odd(
2     input clk,
3     input rst,
4     input [7:0] div_N,
5     output clk_out
6 );
7
8 /* 保证输入分频系数为奇数 */
9 wire [7:0] div_N_in;
10 assign div_N_in = div_N | 8'b0000_0001;
11
12 /* 第一个计数器, 采样上升沿 */
13 reg [7:0] cnt1;
14 always @(posedge clk, posedge rst) begin
15     if (rst) begin /* 高电平异步复位 */
16         cnt1 <= 8'b0;
17     end
18     else if (cnt1 < div_N_in-1'b1) begin /* 已采样N次上升沿 */
19         cnt1 <= cnt1 + 1'b1;

```

```

20     end
21     else begin                                     /* 采样自增 */
22         cnt1 <= 8'b0;
23     end
24 end
25 assign clk1 = (cnt1 < {1'b0, div_N_in[7:1]}) ? 1'b1 : 1'b0; /* 高电平比低电平多1个周期 */
26
27 /* 第二个计数器，采样下降沿 */
28 reg [7:0] cnt2;
29 always @(negedge clk, posedge rst) begin
30     if (rst) begin                                 /* 高电平异步复位 */
31         cnt2 <= 8'b0;
32     end
33     else if (cnt1 < div_N_in-1'b1) begin           /* 已采样N次下降沿 */
34         cnt2 <= cnt2 + 1'b1;
35     end
36     else begin                                     /* 采样自增 */
37         cnt2 <= 8'b0;
38     end
39 end
40 assign clk2 = (cnt2 < {1'b0, div_N_in[7:1]}) ? 1'b1 : 1'b0; /* 高电平比低电平多一个周期 */
41 assign clk_out = clk1 | clk2;                     /* 通过或运算输出.5周期结果 */
42 endmodule

```

二进制转 BCD 码源文件如下:

Listing 8: 二进制转 BCD 码

```

1 module bin2bcd_8bit(
2     input [7:0] bin,
3     output [3:0] bcd_12bit /* 高中低BCD结果 */
4 );
5
6 integer i;
7 reg [19:0] bcd_temp;
8 always @(*) begin
9     // 初始化，直接移位三次
10    bcd_temp = {9'b0, bin, 3'b0};
11    // 逐位移位法
12    for (i = 0; i < 5; i = i + 1'b1) begin
13        // 如果BCD的每一部分 > 4，加3
14        if (bcd_temp[11:8] > 4)
15            bcd_temp[11:8] = bcd_temp[11:8] + 3;
16        if (bcd_temp[15:12] > 4)
17            bcd_temp[15:12] = bcd_temp[15:12] + 3;
18        if (bcd_temp[19:16] > 4)
19            bcd_temp[19:16] = bcd_temp[19:16] + 3;
20        // 左移1位
21        bcd_temp = {bcd_temp[18:0], 1'b0};
22    end
23 end
24 assign bcd_12bit = bcd_temp[19 -: 12];
25 endmodule

```


数码管显示源文件如下:

Listing 9: 数码管显示源文件

```
1 module scan_led_hex_disp_4(  
2     input clk, rst,  
3     input [3:0] hex0, hex1, hex2, hex3, /* 显存 */  
4     input [3:0] dp,  
5     output reg [3:0] an,  
6     output reg [7:0] sseg  
7 );  
8  
9 localparam N = 16 + 2;          /* 100MHz时钟分频, 100Mhz/ 2^16 */  
10 reg [N-1:0] regN;  
11  
12 always @(posedge clk, posedge rst) begin  
13     if (rst)  
14         regN <= 0;  
15     else  
16         regN <= regN + 1;  
17 end  
18  
19 always @(*) begin  
20     case (regN[N-1:N-2])  
21         2'b00: begin  
22             an <= 4'b0001;  
23             sseg[6:0] <= dt_translate(hex0);  
24             sseg[7] <= dp[3];  
25         end  
26         2'b01: begin  
27             an <= 4'b0010;  
28             sseg[6:0] <= dt_translate(hex1);  
29             sseg[7] <= dp[2];  
30         end  
31         2'b10: begin  
32             an <= 4'b0100;  
33             sseg[6:0] <= dt_translate(hex2);  
34             sseg[7] <= dp[1];  
35         end  
36         2'b11: begin  
37             an <= 4'b1000;  
38             sseg[6:0] <= dt_translate(hex3);  
39             sseg[7] <= dp[0];  
40         end  
41     endcase  
42 end  
43  
44 function [6:0] dt_translate;  
45     input [3:0] data;  
46     begin  
47         case(data)
```

```

48         4'd0: dt_translate = 7'b1111110;    //number 0 -> 0x7e
49         4'd1: dt_translate = 7'b0110000;    //number 1 -> 0x30
50         4'd2: dt_translate = 7'b1101101;    //number 2 -> 0x6d
51         4'd3: dt_translate = 7'b1111001;    //number 3 -> 0x79
52         4'd4: dt_translate = 7'b0110011;    //number 4 -> 0x33
53         4'd5: dt_translate = 7'b1011011;    //number 5 -> 0x5b
54         4'd6: dt_translate = 7'b1011111;    //number 6 -> 0x5f
55         4'd7: dt_translate = 7'b1110000;    //number 7 -> 0x70
56         4'd8: dt_translate = 7'b1111111;    //number 8 -> 0x7f
57         4'd9: dt_translate = 7'b1111011;    //number 9 -> 0x7b
58     endcase
59 end
60 endfunction
61 endmodule

```

top 源文件如下:

Listing 10: top 源文件

```

1  module clk_div_top(
2      input clk,                /* 100MHz */
3      input rst,                /* 低电平异步复位 */
4      input [7:0] div_coe,      /* 输入的四位整数加四位小数的二进制实数 */
5      output [7:0] sseg1, sseg2, /* 八段数码管 */
6      output [3:0] an1, an2,    /* 片选信号 */
7      output led_out           /* 闪烁的LED灯 */
8  );
9
10 /* 分频产生30Hz */
11 wire clk_30Hz;
12 clk_div_32bit clk_div_32bit_inst (
13     .clk(clk),
14     .rstn(!rst),
15     .step_freq(32'd1288),
16     .clk_out(clk_30Hz)
17 );
18
19 wire clk_out_even;
20 clk_div_even clk_div_even_inst (
21     .clk(clk_30Hz),
22     .rst(rst),
23     .div_N(div_coe),
24     .clk_out(clk_out_even)
25 );
26
27 wire clk_out_odd;
28 clk_div_odd clk_div_odd_inst (
29     .clk(clk_30Hz),
30     .rst(rst),
31     .div_N(div_coe),
32     .clk_out(clk_out_odd)
33 );
34

```

```

35  /* 检测输入状态，当分频系数输入为0或1，认为不分频，显示30Hz的分频系数 */
36  assign div_none = div_coe < 8'd2 ? 1'b1 : 1'b0;
37
38  /* 根据输入系数切换输出 */
39  assign led_out = div_none ? clk_30Hz :
40      (div_coe[0]) ? clk_out_odd : clk_out_even;
41
42  wire [11:0] bcd_12bit;
43  bin2bcd_8bit bin2bcd_8bit_inst (
44      .bin(div_coe),
45      .bcd_12bit(bcd_12bit)
46  );
47
48  /* 修改显示内容 */
49  reg [3:0] disp_buffer[0:7];
50  always @(*) begin
51      if (div_none) begin
52          disp_buffer[0] <= 4'd3;
53          disp_buffer[1] <= 4'd3;
54          disp_buffer[2] <= 4'd3;
55          disp_buffer[3] <= 4'd4;
56          disp_buffer[4] <= 4'd6;
57          disp_buffer[5] <= 4'd0;
58          disp_buffer[6] <= 4'd1;
59          disp_buffer[7] <= 4'd9;
60      end
61      else begin
62          disp_buffer[0] <= 4'd0;
63          disp_buffer[1] <= 4'd0;
64          disp_buffer[2] <= 4'd0;
65          disp_buffer[3] <= 4'd0;
66          disp_buffer[4] <= 4'd0;
67          disp_buffer[5] <= bcd_12bit[11:8];
68          disp_buffer[6] <= bcd_12bit[7:4];
69          disp_buffer[7] <= bcd_12bit[3:0];
70      end
71  end
72
73  /* 修改小数点 */
74  reg [3:0] dot_buffer;
75  always @(*) begin
76      if (div_none) begin
77          dot_buffer <= 4'b0010;
78      end
79      else begin
80          dot_buffer <= 4'b0000;
81      end
82  end
83
84  /* 第一个四位数码管 */
85  scan_led_hex_disp_4 scan_led_hex_disp_4_inst_0 (
86      .clk(clk), .rst(rst),

```

```

87     .hex0(displ_buffer[0]), .hex1(displ_buffer[1]), .hex2(displ_buffer[2]), .hex3(displ_buffer[3]), .dp(4'b0000),
88     .an(an1), .sseg(sseg1)
89 );
90 /* 第二个四位数码管 */
91 scan_led_hex_disp_4 scan_led_hex_disp_4_inst_1 (
92     .clk(clk), .rst(rst),
93     .hex0(displ_buffer[4]), .hex1(displ_buffer[5]), .hex2(displ_buffer[6]), .hex3(displ_buffer[7]), .dp(dot_buffer),
94     .an(an2), .sseg(sseg2)
95 );
96 endmodule

```

奇数分频器设计过程中的仿真激励如下:

Listing 11: 奇数分频器仿真激励

```

1  initial begin
2      div_coe = 8'd7;
3      clk = 0;
4      rst = 0;
5      #100;
6      rst = 1;
7      #10;
8      rst = 0;
9  end

```

整个系统的仿真文件如下:

Listing 12: Top 系统仿真文件如下

```

1  module clk_div_top_tb_1;
2      reg  clk;
3      reg  rst;
4      reg [7:0] div_coe;
5      wire [7:0] sseg1, sseg2;
6      wire [3:0] an1, an2;
7      wire  led_out;
8
9      initial begin
10         clk = 0;
11         rst = 0;
12         #10;
13         rst = 1;
14         #10;
15         rst = 0;
16
17         div_coe = 8'd0;
18         #300_000_000;
19         div_coe = 8'd4;
20         #300_000_000;
21         div_coe = 8'd5;
22         #300_000_000;
23         $finish;
24     end

```

```

25
26     clk_div_top  clk_div_top_inst (
27         .clk(clk),
28         .rst(rst),
29         .div_coe(div_coe),
30         .sseg1(sseg1),
31         .sseg2(sseg2),
32         .an1(an1),
33         .an2(an2),
34         .led_out(led_out)
35     );
36
37     always #5 clk = ! clk ;
38 endmodule

```

三、仿真结果

基础任务

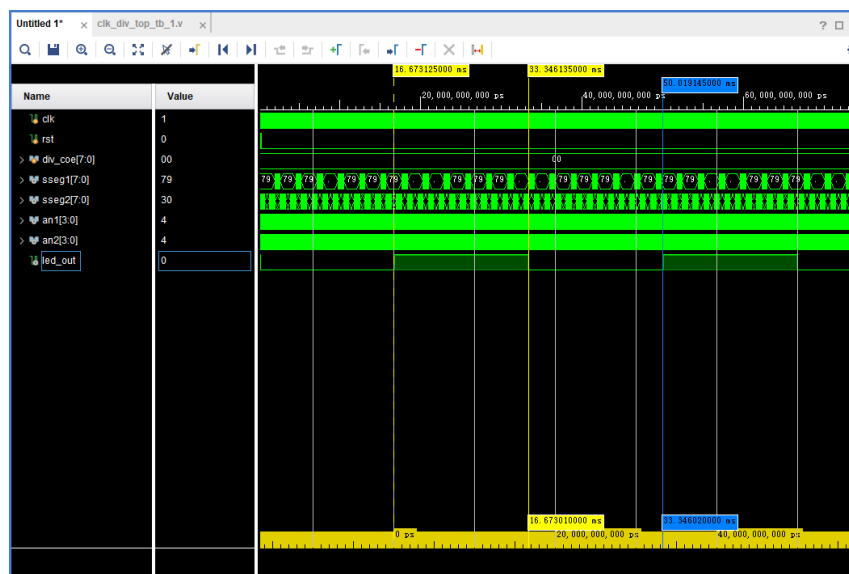


图 2: 30Hz 输出仿真结果

输出 30Hz 的仿真结果见图 2, 首先可以通过 Marker 看到, 在一个输出时钟周期内, 信号的时钟周期为 50% 占空比, 周期为 33.346ms, 计算得到其频率为 29.9866Hz, 与理论计算结果一致。

$$f = \frac{1}{33.346 * 10^{-3}} = 29.9886Hz \quad (6)$$

提高任务

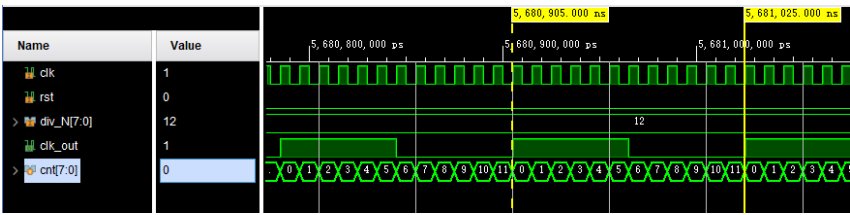


图 3: 偶数分频仿真结果

对偶数分频器的仿真如图 3所示, tb 文件中设定了输入的分频系数为 12, 仿真结果可以清晰地看到, 分频产生的输出时钟的周期为 12 个输入时钟, 并且其占空比为%50。

拓展任务

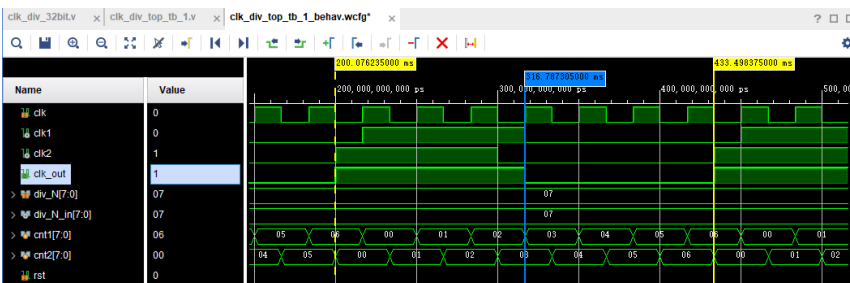


图 4: 奇数分频器仿真结果

对奇数分频器的仿真如图 4所示, tb 文件中设定了输入的分频系数为 7, 对输入时钟进行 7 分频, 结果应该是输出时钟的高低电平各包含 3.5 周期, 通过仿真时序图可以清晰地看到, 输出波形 clk_out 高低电平各为 3.5 周期, 达到了 50% 占空比的分频输出要求。top 系统的仿真结果如图 5所示, 前

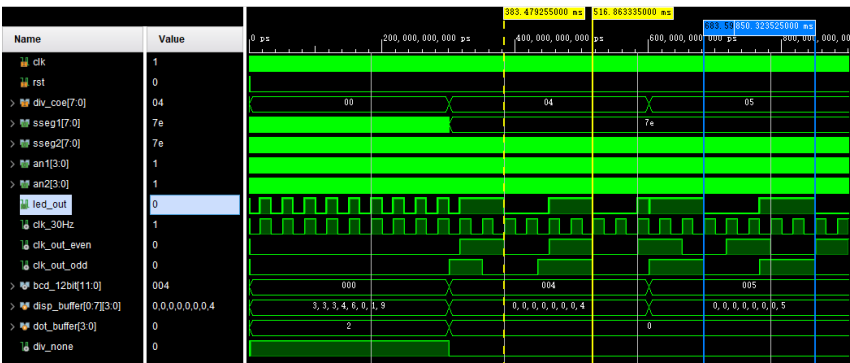


图 5: Top 系统仿真结果

300ms 为输出 30Hz 分频信号, 同时显示其系数, 之后一个 300ms 内将分频系数设置为 4, 系统将输出信号切换为偶分频器输出结果, 最后 300ms 内, 将分频系数设置为 5, 数码管将显示 005, 同时自动输出奇数分频产生 5 分频信号, 频率在 6Hz 左右。

四、硬件验证结果

基础任务

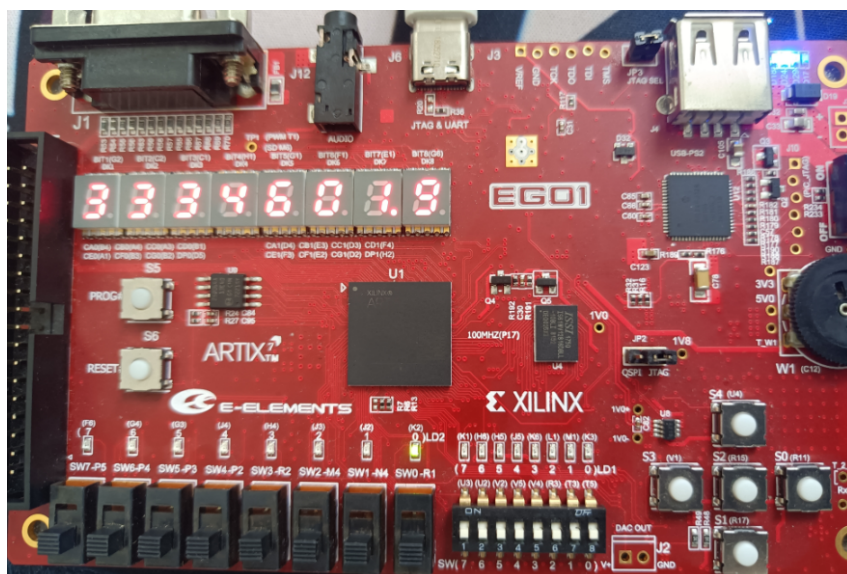


图 6: 30Hz 输出时钟的硬件验证

硬件验证结果如图 6所示, 数码管能够正确显示对应的分频系数, 肉眼能够观察到对应的 LD2_2 频闪。

提高任务

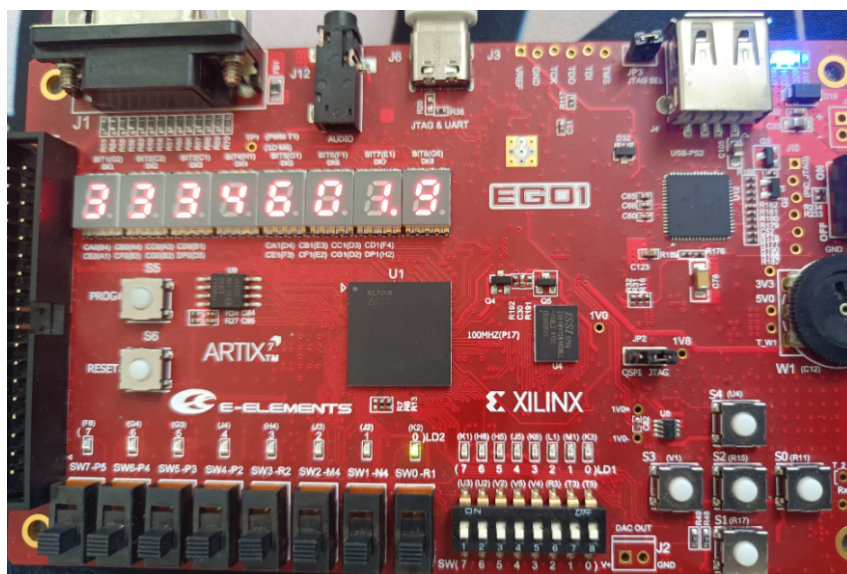


图 7: 偶分频的硬件验证

硬件验证结果如图??所示, 通过拨码开关设置分频系数为 28 进行偶分频, 数码管显示正确, 可以明显通过肉眼观察到 LED 的亮灭变化, 相比于 30Hz 已经进一步分频为

$$f_o = \frac{29.9886}{28}(Hz) = 1.0710(Hz)$$

拓展任务

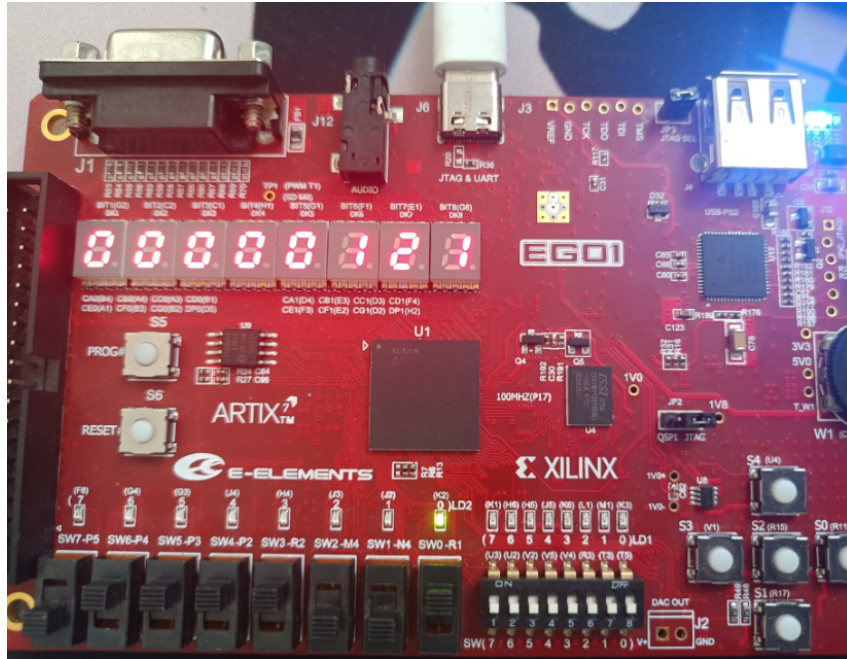


图 8: 奇分频硬件验证

硬件验证结果如图??所示, 通过拨码开关设置分频系数为 121 进行奇分频, 数码管显示正确, 可以明显观察到 LED 的缓慢的进行亮灭变化, 相比于 30Hz 已经进一步分频为

$$f_o = \frac{29.9886}{121}(Hz) = 0.24784(Hz)$$

五、问题解决

最后系统设计中,偶分频不输出

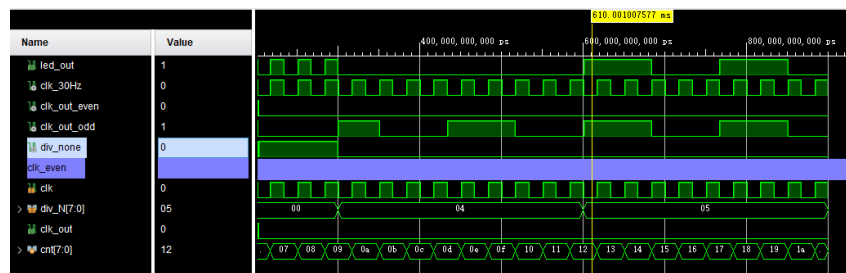


图 9: 问题仿真

通过系统仿真, 观察通用偶分频器模块内部计数器变化解决问题, 仿真结果如图 7 所示, 仿真文件中给了 300ms 时间用于偶分频输出, 通过观察内部计数器的变化可以得知, 源文件设计为到达目标值时复位, 但是 cnt 没有进行使能管理, 导致 cnt 在进行有效分频前就进行了采样计数, 导致其计数值大于了设定的溢出值, 计数器来不及溢出清零。将分频器中计数器的清零判断由相等改为小于则增判断。

使用等于号进行清零的计数器占用资源多余小于号

发现该问题时,源文件已经较为复杂,综合过程缓慢,为了提高效率创建单独项目研究该问题。

六、写出心得体会

时序逻辑设计是复杂系统设计过程中所必需的, 通过分频器的设计流程, 同时掌握了 DDS 数字电路部分的技术原理和计数器的应用, 在理解和掌握这些原理的基础, 将其用 Verilog 语言描述并实现自己的想法, 实现任意分频模块, 并完成偶分频和奇分频的通用模块这些时序电路中基础, 对之后更加复杂的电路设计奠定了基础。并通过适当扩展, 将偶分频和奇分频模块结合后, 实现任意整数分频, DDS 的相位累加器则可以实现小数分频。

随着设计越来越复杂,所使用的模块也越来越多,体现出了层次化设计的强烈需求,可以适当通过 IP 封装,提高模块的复用效率。