**电**

**子**

**测**

**试**

**与**

**试**

**验**

**技**

**术**

姓 名 薛谦

学 号 U202014228

实验名称 集成运算放大器的基本应用

1. **实验名称**

利用FPGA设计电子数字时钟

1. **实验目的**

熟练运用Verilog进行模块编程，在FPGA小板上实现简易数字钟，完成时、分计数。

1. **实验元器件**

高云FPGA实验板，杜邦线，数据传输线

1. **实验任务**

基础要求：

1、在4位数码管上显示小时、分钟，在分钟位的小数点上用闪烁表示走秒

2、用拨码开关实现24/12小时制切换，12小时制时，在小时位小数点上用亮或者不亮表示am/pm

3、通过4个按钮完成小时、分钟的校时调整

扩展要求：

1、实现整点报时提示（可用LED或者显示闪烁完成）

2、实现闹钟功能（设置、到时间LED或闪烁提醒）

3、实现倒计时、秒表功能（实际未编写实现）

1. **实验代码**

1.顶层模块调用

module top(

    input sw2,

    input sw3,

    input sw4,

    input key\_1,

    input key\_2,

    input key\_3,

    input key\_4,

    input alarm\_en,

    output DP,

    output [6:0] data,

    output [3:0] CS,

    output [1:0] led0,

    output [1:0] led1

);

    wire [6:0] data\_clo;

    wire [3:0] CS\_clo;

    wire [6:0] data\_alar;

    wire [3:0] CS\_alar;

    wire [5:0] hou\_clo;

    wire [5:0] min\_clo;

    wire [5:0] sec\_clo;

    wire [5:0] hou\_alar;

    wire [5:0] min\_alar;

    wire [5:0] sec\_alar;

    assign data = sw2?data\_alar:data\_clo;//sw2 = 1时表示闹钟设置，0表示时钟数据

    assign CS = sw2?CS\_alar:CS\_clo;

    wire clk;

    XS\_OSC XS\_OSC(clk);

    clock clock(

        .clk(clk),

        .switch(sw3),

        .rst(sw4),

        .key\_1(key\_1&(!sw2)),

        .key\_2(key\_2&(!sw2)),

        .key\_3(key\_3&(!sw2)),

        .key\_4(key\_4&(!sw2)),

        .hou\_clo(hou\_clo),

        .min\_clo(min\_clo),

        .sec\_clo(sec\_clo),

        .data(data\_clo),

        .sw2(sw2),

        .DP(DP),

        .led0(led0),

        .CS(CS\_clo)

    );

     alarm alarm(

         .clk(clk),

         .rst(sw4),

         .sw2(sw2),

         .key\_1(key\_1&sw2),

         .key\_2(key\_2&sw2),

         .key\_3(key\_3&sw2),

         .key\_4(key\_4&sw2),

         .hou\_alar(hou\_alar),

         .min\_alar(min\_alar),

         .sec\_alar(sec\_alar),

         .data(data\_alar),

         .CS(CS\_alar)

     );

    beep\_ctrl beep\_ctrl(

    .clk(clk),

    .rst(sw4),

    .alarm\_en(!alarm\_en),

    .hou\_clo(hou\_clo),

    .min\_clo(min\_clo),

    .sec\_clo(sec\_clo),

    .hou\_alar(hou\_alar),

    .min\_alar(min\_alar),

    .led1(led1)

     );

endmodule

2.时钟模式

module clock

(

    input clk,

    input rst,

    input key\_1,

    input key\_2,

    input key\_3,

    input key\_4,

    input switch,

    input sw2,

    output [5:0] hou\_clo,

    output [5:0] min\_clo,

    output [5:0] sec\_clo,

    output [6:0] data,

    output [3:0] CS,

    output DP,

    output [1:0] led0

    );

    parameter

        num0 = 5'd0,

        num1 = 5'd1,

        num2 = 5'd2,

        num3 = 5'd3,

        num4 = 5'd4,

        num5 = 5'd5,

        num6 = 5'd6,

        num7 = 5'd7,

        num8 = 5'd8,

        num9 = 5'd9;

    wire [5:0] hour;

    wire [5:0] minute;

    wire [5:0] second;

    wire [4:0] noon;

    wire [1:0] hou\_bias;

    wire [1:0] min\_bias;

    wire [1:0] sec\_bias;

    wire state\_1;

    wire state\_2;

    wire state\_3;

    wire state\_4;

    wire [5:0] D0;

    wire [5:0] D1;

    wire [5:0] D2;

    wire [5:0] D3;

    wire [3:0] CS\_in;

    wire flash\_flag;

    wire [1:0] status;

    wire [1:0] blink\_flag;

    assign hou\_clo = hour;

    assign min\_clo = minute;

    assign sec\_clo = second;

    twinkle twink(

        .clk(clk),

        .rst(rst),

        .CS\_in(CS\_in),

        .flash\_flag(flash\_flag),

        .status(status),

        .CS\_out(CS)

    );

    clock\_ctrl#(//实现时间变换

        .sec\_period(27'd5000000)

    )

    ctrl(

        .clk(clk),

        .rst(rst),

        .switch(switch),

        .hou\_bias(hou\_bias),

        .min\_bias(min\_bias),

        .sec\_bias(sec\_bias),

        .hour(hour),

        .minute(minute),

        .second(second),

        .noon(noon),

        .flash\_flag(flash\_flag),

        .blink(blink\_flag)

    );

    led\_flash led\_flash(

        .clk(clk),

        .rst(rst),

        .hour(hour),

        .minute(minute),

        .second(second),

        .state\_1(blink\_flag),

        .led(led0)

    );

    adjust adjust(

        .clk(clk),

        .rst(rst),

        .key\_1(state\_1),

        .key\_2(state\_2),

        .key\_3(state\_3),

        .key\_4(state\_4),

        .sw2(!sw2),

        .hou\_bias(hou\_bias),

        .min\_bias(min\_bias),

        .sec\_bias(sec\_bias),

        .status(status)

    );

    transfer u\_transfer(

        .clk(clk),

        .rst(rst),

        .hour(hour),

        .minute(minute),

        .second(second),

        .D0(D0),            //分钟

        .D1(D1),

        .D2(D2),            //小时

        .D3(D3)

    );

    tube\_driver tube(

        .clk(clk),

        .rst(rst),

        .Din0(D0),

        .Din1(D1),

        .Din2(D2),

        .Din3(D3),

        .data(data),

        .CS(CS\_in)

    );

    blink blink\_u(

        .DP(DP),

        .CS(CS\_in),

        .clk(clk),

        .rst(rst),

        .noon(noon),

        //.CS\_OUT(CS),

        .switch(switch)

        );

    key\_filter key\_filter\_1(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_1),//按键输入

       .sw\_out\_n(state\_1)//按键状态信号

    );

    key\_filter key\_filter\_2(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_2),//按键输入

       .sw\_out\_n(state\_2)//按键状态信号

    );

    key\_filter key\_filter\_3(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_3),//按键输入

       .sw\_out\_n(state\_3)//按键状态信号

    );

    key\_filter key\_filter\_4(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_4),//按键输入

       .sw\_out\_n(state\_4)//按键状态信号

    );

Endmodule

3.计数模块

module counter #(

    parameter M = 100

)(

    input clk, rst, en,

    output reg [$clog2(M) - 1 : 0] cnt,

    output co

);

    assign co = en & (cnt == M - 1);

    always@(posedge clk) begin

        if(rst)

        begin

            cnt <= 1'b0;

        end

        else if(en)

        begin

            if(cnt < M - 1) cnt <= cnt + 1'b1;

            else cnt <= 1'b0;

        end

    end

endmodule

4.数码管解码模块

/\*输入端0~9对应数字0~9\*/

module decode#(

    parameter

        num0 = 7'b011\_1111,

        num1 = 7'b000\_0110,

        num2 = 7'b101\_1011,

        num3 = 7'b100\_1111,

        num4 = 7'b110\_0110,

        num5 = 7'b110\_1101,

        num6 = 7'b111\_1101,

        num7 = 7'b000\_0111,

        num8 = 7'b111\_1111,

        num9 = 7'b110\_1111

)(

    input [5:0] Din,

    output reg [6:0] data

  );

always@(Din)

begin

    case(Din)

        6'd0: data <= num0;

        6'd1: data <= num1;

        6'd2: data <= num2;

        6'd3: data <= num3;

        6'd4: data <= num4;

        6'd5: data <= num5;

        6'd6: data <= num6;

        6'd7: data <= num7;

        6'd8: data <= num8;

        6'd9: data <= num9;

     default: data <= 7'b000\_0000;//全熄灭

    endcase

end

endmodule

5.按键消抖模块（每次按下经过消抖后对应按键信号拉高一个时钟周期）

module key\_filter(

 input clk, //5MHz

 input rst,

 input sw\_in\_n,

 output reg sw\_out\_n

    );

 reg sw\_mid\_r1, sw\_mid\_r2, sw\_valid;

 always@(posedge clk or posedge rst) begin

  if(rst) begin

  sw\_mid\_r1 <= 1;   // synchronize 1 clock

  sw\_mid\_r2 <= 1;   // delay 1 clock

  sw\_valid <= 0;    // gen negedge

  end

  else begin

  sw\_mid\_r1   <=  sw\_in\_n;

  sw\_mid\_r2   <=  sw\_mid\_r1;

  sw\_valid  <=  sw\_mid\_r2 & (~sw\_mid\_r1);

  end

 end

 reg [19:0] key\_cnt;

 always@(posedge clk or posedge rst) begin

  if(rst)

  begin

   key\_cnt <= 0;

  end

  else if(sw\_valid)

  begin

   key\_cnt <= 0;

  end

  else

  begin

   key\_cnt <= key\_cnt + 1; //20ms

  end

 end

 always@(posedge clk or posedge rst) begin

  if(rst) begin

  sw\_out\_n <= 0;

  end

    else if(key\_cnt == 20'd10000)

    sw\_out\_n <= !sw\_in\_n;

  else if (key\_cnt == 20'd10001)

    begin

      sw\_out\_n <= 0;

    end

 end

endmodule

6.按键模块

module filter\_keys(

    input clk,

    input rst,

    input key\_1,

    input key\_2,

    input key\_3,

    input key\_4,

    output flag\_1,

    output flag\_2,

    output flag\_3,

    output flag\_4,

    output state\_1,

    output state\_2,

    output state\_3,

    output state\_4

    );

    key\_filter key\_filter\_1(

       .clk(clk),//5M时钟输入

       .rst\_n(~rst),//模块复位

       .key\_in(key\_1),//按键输入

       .key\_flag(flag\_1),//按键标志信号

       .key\_state(state\_1)//按键状态信号

    );

    key\_filter key\_filter\_2(

       .clk(clk),//5M时钟输入

       .rst\_n(~rst),//模块复位

       .key\_in(key\_2),//按键输入

       .key\_flag(flag\_2),//按键标志信号

       .key\_state(state\_2)//按键状态信号

    );

    key\_filter key\_filter\_3(

       .clk(clk),//5M时钟输入

       .rst\_n(~rst),//模块复位

       .key\_in(key\_3),//按键输入

       .key\_flag(flag\_3),//按键标志信号

       .key\_state(state\_3)//按键状态信号

    );

    key\_filter key\_filter\_4(

       .clk(clk),//5M时钟输入

       .rst\_n(~rst),//模块复位

       .key\_in(key\_4),//按键输入

       .key\_flag(flag\_4),//按键标志信号

       .key\_state(state\_4)//按键状态信号

    );

endmodule

7.系统时钟模块

//Copyright (C)2014-2022 Gowin Semiconductor Corporation.

//All rights reserved.

//File Title: IP file

//GOWIN Version: V1.9.8.05

//Part Number: GW1N-LV4QN48C6/I5

//Device: GW1N-4B

//Created Time: Mon May 23 20:44:08 2022

module XS\_OSC (oscout);

output oscout;

OSC osc\_inst (

    .OSCOUT(oscout)

);

defparam osc\_inst.FREQ\_DIV = 42;

defparam osc\_inst.DEVICE = "GW1N-4B";

endmodule

8.闪烁模块

module led\_flash(

    input clk,

    input rst,

    input [5:0] hour,

    input [5:0] minute,

    input [5:0] second,

    input [1:0]state\_1,

    output reg [1:0] led

    );

    reg [5:0] count;

    reg flag;

    wire co;

    always@(posedge clk or posedge rst)begin

        if(rst)

            flag <= 1'd0;

        else if((hour !=6'd0)&&(minute == 6'd0)&&(second == 6'd0))

            flag <= 1'd1;

        else if(count == {hour[4:0],1'b0})

            flag <= 1'd0;

        else

            flag <= flag;

    end

    always@(posedge clk or posedge rst)begin

    if(rst)

    led <= 2'b10;

    else    begin

    case (state\_1)

        2'd00: led <= 2'd00;

        2'd01: led <= 2'd00;

        2'd10: led <= 2'd00;

        2'd11: led <= 2'd11;

        default: begin

            led <= 2'b11;

        end

    endcase

    end

    end

    /\*   always@(posedge clk or posedge rst)begin

        if(rst)

        begin

            led <= 2'b00;

            count <= 6'd0;

        end

        else if(co)

        begin

            led <= ~led;

            count <= count + 6'd1;

        end

        else if(count == {hour[4:0],1'b0})

            count <= 6'd0;

        else;

    end

    \*/用于测试的一段代码

    counter #(1250000)counter(

    .clk(clk),.rst(rst),.en(flag),

    .cnt(),

    .co(co)

);

endmodule

9.数码管显示模块

module transfer(

    input clk,

    input rst,

    input [5:0] hour,

    input [5:0] minute,

    input [5:0] second,

    output reg [5:0] D0,

    output reg [5:0] D1,

    output reg [5:0] D2,

    output reg [5:0] D3

);

always@(posedge clk or posedge rst)begin

    if(rst)begin

        D3 <= 5'd0;

        D2 <= 5'd0;

    end

    else if(hour < 5'd10)begin

        D3 <= 5'd0;

        D2 <= hour;

    end

    else if((hour > 5'd9)&&(hour < 5'd20))begin

        D3 <= 5'd1;

        D2 <= hour - 5'd10;

    end

    else begin

        D3 <= 5'd2;

        D2 <= hour - 5'd20;

    end

end

always@(posedge clk or posedge rst)begin

    if(rst)begin

        D1 <= 6'd0;

        D0 <= 6'd0;

    end

    else if(minute < 6'd10)begin

        D1 <= 6'd0;

        D0 <= minute;

    end

    else if((minute > 6'd9)&&(minute < 6'd20))begin

        D1 <= 6'd1;

        D0 <= minute - 6'd10;

    end

    else if((minute > 6'd19)&&(minute < 6'd30))begin

        D1 <= 6'd2;

        D0 <= minute - 6'd20;

    end

    else if((minute > 6'd29)&&(minute < 6'd40))begin

        D1 <= 6'd3;

        D0 <= minute - 6'd30;

    end

    else if((minute > 6'd39)&&(minute < 6'd50))begin

        D1 <= 6'd4;

        D0 <= minute - 6'd40;

    end

    else begin

        D1 <= 6'd5;

        D0 <= minute - 6'd50;

    end

end

endmodule

10.数码管驱动模块

/\*输入端0~9对应数字0~9\*/

module tube\_driver(

    input clk,

    input rst,

    input [5:0] Din0,

    input [5:0] Din1,

    input [5:0] Din2,

    input [5:0] Din3,

    output reg [6:0] data,

    output reg [3:0] CS

    );

//中间变量

wire [6:0] data0;

wire [6:0] data1;

wire [6:0] data2;

wire [6:0] data3;

reg [1:0] cnt;//计数器，从0计数到3

reg [26:0] timer;

reg fullflag;

always@(posedge clk or posedge rst)begin

    if(rst)

        timer <= 27'd0;

    else if(timer == 27'd1\_9999)

        timer <= 27'd0;

    else

        timer <= timer + 27'd1;

end

always@(posedge clk or posedge rst)begin

    if(rst)

        fullflag <= 1'b0;

    else if(timer == 27'd999)

        fullflag <= 1'b1;

    else

        fullflag <= 1'd0;

end

    decode Data0(

        .Din(Din0),

        .data(data0)

    );

    decode Data1(

        .Din(Din1),

        .data(data1)

    );

    decode Data2(

        .Din(Din2),

        .data(data2)

    );

    decode Data3(

        .Din(Din3),

        .data(data3)

    );

always@(posedge clk or posedge rst)begin

    if(rst)

        cnt <= 2'd0;

    else if(fullflag)

        cnt <= cnt + 2'd1;

    else

        cnt <= cnt;

end

always@(posedge clk or posedge rst)begin

    if(rst)begin

        CS <= 4'b0000;

        data <= 7'b000\_0000;

    end

    else begin

        case(cnt)

            2'd0:begin CS <= 4'b1110; data <= data0; end

            2'd1:begin CS <= 4'b1101; data <= data1; end

            2'd2:begin CS <= 4'b1011; data <= data2; end

            2'd3:begin CS <= 4'b0111; data <= data3; end

            default:begin CS <= 4'b0000; data <= 7'b111\_1111; end

        endcase

    end

end

endmodule

11.掩码生成模块

module twinkle(

    input clk,

    input rst,

    input [3:0] CS\_in,

    input flash\_flag,

    input [1:0] status,

    output [3:0] CS\_out

    );

    reg [3:0] mask;

    assign CS\_out = CS\_in|mask;

    always@(posedge clk or posedge rst)begin

        if(rst)

            mask <= 4'd0;

        else

        case(status)

            2'd1:begin

                    if(flash\_flag)

                        mask <= 4'b0011;

                    else

                        mask <= 4'd0;

                    end

            2'd2:begin

                    if(flash\_flag)

                        mask <= 4'b1100;

                    else

                        mask <= 4'd0;

                    end

            default: begin

                mask <= 4'd0;

            end

        endcase

    end

endmodule

12.时间调节模块

module adjust(

    input clk,

    input rst,

    input key\_1,

    input key\_2,

    input key\_3,

    input key\_4,

    input sw2,

    output reg [1:0] hou\_bias,

    output reg [1:0] min\_bias,

    output reg [1:0] sec\_bias,

    output [1:0] status

    );

    reg [1:0] r\_status;

    assign status = r\_status;

    always@(posedge clk or posedge rst)begin

        if(rst)

            r\_status <= 2'd0;

        else if(key\_4 & sw2)

            r\_status <= !r\_status;

        else if(key\_3 & sw2)

            r\_status <= r\_status + 2'd1;

        else

            r\_status <= r\_status;

    end

    always@(posedge clk or posedge rst )begin

            if(rst)begin

            sec\_bias <= 6'd0;

            min\_bias <= 6'd0;

            hou\_bias <= 5'd0;

            end

            else if(key\_1)begin

                case(r\_status)

                    2'd1:min\_bias <= 2'b01;

                    2'd2:hou\_bias <= 2'b01;

                    default:begin

                        sec\_bias <= 2'b00;

                        min\_bias <= 2'b00;

                        hou\_bias <= 2'b00;

                    end

                endcase

            end

            else if(key\_2)begin

                case(r\_status)

                    2'd1:min\_bias <= 2'b11;

                    2'd2:hou\_bias <= 2'b11;

                    default:begin

                        sec\_bias <= 2'b00;

                        min\_bias <= 2'b00;

                        hou\_bias <= 2'b00;

                    end

                endcase

            end

        else begin

            sec\_bias <= 2'b00;

            min\_bias <= 2'b00;

            hou\_bias <= 2'b00;

        end

    end

endmodule

13.闹钟控制模块

module alarm\_ctrl#(

    parameter   sec\_period = 27'd499\_9999,

                 A = 5'd10,

                 P = 5'd19

)(

    input clk,

    input rst,

    input [1:0] hou\_bias,

    input [1:0] min\_bias,

    input [1:0] sec\_bias,

    output reg [5:0] hour,

    output reg [5:0] minute,

    output reg [5:0] second,

    output flash\_flag

    );

    reg [26:0] cnt;

    //中间变量寄存器

    reg [5:0] r\_hour;

    reg [5:0] r\_minute;

    reg [5:0] r\_second;

    //进位信号

    reg min\_flag;

    reg sec\_flag;

    assign flash\_flag = cnt[19];

always@(posedge clk or posedge rst)begin

    if(rst)

        cnt <= 27'd0;

    else if(cnt == sec\_period)

        cnt <=27'd0;

    else

        cnt <= cnt + 27'd1;

end

 always@(posedge clk or posedge rst)begin

     if(rst)

     begin

        hour    <= 6'd0;

        minute  <= 6'd0;

        second  <= 6'd0;

     end

     else

     begin

        hour <= r\_hour;

        minute <= r\_minute;

        second <= r\_second;

     end

 end

always@(posedge clk or posedge rst)begin

    if(rst)

        r\_hour <= 5'd0;

    else

        case(hou\_bias)

            2'b00:

                r\_hour <= r\_hour;

            2'b01:begin

                if(r\_hour == 5'd23)

                    r\_hour <= 5'd0;

                else

                    r\_hour <= r\_hour + 5'd1;

            end

            2'b11:begin

                if(r\_hour == 5'd0)

                    r\_hour <= 5'd23;

                else

                    r\_hour <= r\_hour - 5'd1;

            end

        endcase

end

always@(posedge clk or posedge rst)begin

    if(rst)

        r\_minute <= 6'd1;

    else

        case(min\_bias)

            2'b00:

                r\_minute <= r\_minute;

            2'b01:begin

                if(r\_minute == 6'd59)

                    r\_minute <= 6'd0;

                else

                    r\_minute <= r\_minute + 6'd1;

            end

            2'b11:begin

                if(r\_minute == 6'd0)

                    r\_minute <= 6'd59;

                else

                    r\_minute <= r\_minute - 6'd1;

            end

        endcase

end

always@(posedge clk or posedge rst)begin

    if(rst)

        r\_second <= 6'd0;

    else

        case(sec\_bias)

            2'b00:

                r\_second <= r\_second;

            2'b01:begin

                if(r\_second == 6'd59)

                    r\_second <= 6'd0;

                else

                    r\_second <= r\_second + 6'd1;

            end

            2'b11:begin

                if(r\_second == 6'd0)

                    r\_second <= 6'd59;

                else

                    r\_second <= r\_second - 6'd1;

            end

            default:r\_second <= r\_second;

        endcase

end

endmodule

14.闹钟模块

module alarm

(

    input clk,

    input rst,

    input key\_1,

    input key\_2,

    input key\_3,

    input key\_4,

    input sw2,

    output [5:0] hou\_alar,

    output [5:0] min\_alar,

    output [5:0] sec\_alar,

    output [6:0] data,

    output [3:0] CS

    //output [1:0] led0

    );

    parameter

        num0 = 5'd0,

        num1 = 5'd1,

        num2 = 5'd2,

        num3 = 5'd3,

        num4 = 5'd4,

        num5 = 5'd5,

        num6 = 5'd6,

        num7 = 5'd7,

        num8 = 5'd8,

        num9 = 5'd9;

    wire [5:0] hour;

    wire [5:0] minute;

    wire [5:0] second;

    wire [1:0] hou\_bias;

    wire [1:0] min\_bias;

    wire [1:0] sec\_bias;

    wire state\_1;

    wire state\_2;

    wire state\_3;

    wire state\_4;

    wire [5:0] D0;

    wire [5:0] D1;

    wire [5:0] D2;

    wire [5:0] D3;

    wire [3:0] CS\_in;

    wire flash\_flag;

    wire [1:0] status;

    assign hou\_alar = hour;

    assign min\_alar = minute;

    assign sec\_alar = second;

    twinkle twink(

        .clk(clk),

        .rst(rst),

        .CS\_in(CS\_in),

        .flash\_flag(flash\_flag),

        .status(status),

        .CS\_out(CS)

    );

    alarm\_ctrl#(//实现时间变换

        .sec\_period(27'd5000000)

    )

    ctrl(

        .clk(clk),

        .rst(rst),

        .hou\_bias(hou\_bias),

        .min\_bias(min\_bias),

        .sec\_bias(sec\_bias),

        .hour(hour),

        .minute(minute),

        .second(second),

        .flash\_flag(flash\_flag)

    );

    // led\_flash led\_flash(

    //     .clk(clk),

    //     .rst(rst),

    //     .hour(hour),

    //     .minute(minute),

    //     .second(second),

    //     .state\_1(status),

    //     .led(led0)

    // );

    adjust adjust(

        .clk(clk),

        .rst(rst),

        .key\_1(state\_1),

        .key\_2(state\_2),

        .key\_3(state\_3),

        .key\_4(state\_4),

        .sw2(sw2),

        .hou\_bias(hou\_bias),

        .min\_bias(min\_bias),

        .sec\_bias(sec\_bias),

        .status(status)

    );

    transfer u\_transfer(

        .clk(clk),

        .rst(rst),

        .hour(hour),

        .minute(minute),

        .second(second),

        .D0(D0),            //分钟

        .D1(D1),

        .D2(D2),            //小时

        .D3(D3)

    );

    tube\_driver tube(

        .clk(clk),

        .rst(rst),

        .Din0(D0),

        .Din1(D1),

        .Din2(D2),

        .Din3(D3),

        .data(data),

        .CS(CS\_in)

    );

    key\_filter key\_filter\_1(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_1),//按键输入

       .sw\_out\_n(state\_1)//按键状态信号

    );

    key\_filter key\_filter\_2(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_2),//按键输入

       .sw\_out\_n(state\_2)//按键状态信号

    );

    key\_filter key\_filter\_3(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_3),//按键输入

       .sw\_out\_n(state\_3)//按键状态信号

    );

    key\_filter key\_filter\_4(

       .clk(clk),//5M时钟输入

       .rst(rst),//模块复位

       .sw\_in\_n(key\_4),//按键输入

       .sw\_out\_n(state\_4)//按键状态信号

    );

endmodule

15.闹钟响应模块

module beep\_ctrl(

    input clk,

    input rst,

    input alarm\_en,

    input [5:0] hou\_clo,

    input [5:0] min\_clo,

    input [5:0] sec\_clo,

    input [5:0] hou\_alar,

    input [5:0] min\_alar,

    output [1:0] led1

    );

    reg flag;   //响铃标志

    reg [26:0] cnt ;

    assign led1[0] = flag;

    assign led1[1] = alarm\_en;

    always@(posedge clk or posedge rst)begin

        if(rst)

        begin

            flag <= 1'd0;

            cnt <= 27'd0;

        end

        else if(alarm\_en)

            if((hou\_clo == hou\_alar)&&(min\_clo == min\_alar)&&(sec\_clo == 0))

            begin

                flag <= 1'd1;

                cnt <= 27'd0;

            end

            else if(cnt == 27'd500000)

            begin

                flag <= 1'b0;

                cnt <=27'd0;

            end

        else

        begin

            flag <= flag;

            cnt <= cnt + 27'd1;

        end

    end

endmodule

16．准点报时模块

module blink #(

    parameter

    on = 1'b0,

    off = 1'b1

) (

    output reg [0:0] DP,

    input [3:0] CS,

    //output [3:0] CS\_OUT,

    input clk,

    input rst,

    input [4:0] noon,

    input switch

);

//assign CS\_OUT = CS;

reg [26:0] timer;

reg fullflag;

reg [1:0]cnt\_bl;

always@(posedge clk or posedge rst)begin

    if(rst)

        timer <= 27'd0;

    else if(timer == 27'd5000000)

        timer <= 27'd0;

    else

        timer <= timer + 27'd1;

end

always@(posedge clk or posedge rst)begin

    if(rst)

        fullflag <= 1'b0;

    else if(timer == 27'd5000000)

        fullflag <= fullflag + 1'b1;

    else

        fullflag <= fullflag;

end

// always@(posedge clk or posedge rst)begin

//     if(rst)

//         cnt\_bl <= 2'd0;

//     else if(fullflag)

//         cnt\_bl <= cnt\_bl + 2'd1;

//     else

//         cnt\_bl <= cnt\_bl;

// end

always@(posedge clk or posedge rst)

begin

    if(rst)

    begin

        DP <= 1'b0;

    end

    else begin

        case(CS)

           4'b1110:begin

                   if(fullflag == 1)

                       begin

                            DP <= 1'b1;

                       end

                   end

           4'b1101:begin  DP <= 1'b0; end

           4'b1011:begin

            if(noon == 5'd19 )

            begin

                DP <= 1'b1;

            end

            else

            DP <= 1'b0; end

           4'b0111:begin  DP <= 1'b0; end

           default:begin  DP <= 1'b0; end

        endcase

    end

end

endmodule

17.时钟控制逻辑模块

module clock\_ctrl#(

    parameter   sec\_period = 27'd499\_9999,

                 A = 5'd10,

                 P = 5'd19

)(

    input clk,

    input rst,

    input switch,

    input [1:0] hou\_bias,

    input [1:0] min\_bias,

    input [1:0] sec\_bias,

    output reg [5:0] hour,

    output reg [5:0] minute,

    output reg [5:0] second,

    output [4:0] noon,

    output flash\_flag,

    output [1:0]blink

    );

    reg [26:0] cnt;

    //中间变量寄存器

    reg [5:0] r\_hour;

    reg [5:0] r\_minute;

    reg [5:0] r\_second;

    reg [1:0] blink\_flag;

    //进位信号

    reg min\_flag;

    reg sec\_flag;

    assign blink = blink\_flag;

    assign flash\_flag = cnt[20];

    assign noon = switch?((r\_hour > 5'd12)? P : A ):5'b11111;

always@(posedge clk or posedge rst)begin

    if(rst)

        hour <= 6'd15;

    else if(switch)

        hour <= (r\_hour > 6'd12)?(r\_hour - 6'd12):r\_hour;

    else

        hour <= r\_hour;

end

 always@(posedge clk or posedge rst)begin

     if(rst)

     begin

         minute <= 5'd0;

         second <= 5'd0;

     end

     else    begin

     minute <= r\_minute;

     second <= r\_second;

     end

 end

always@(posedge clk or posedge rst)begin

    if(rst)

        min\_flag <= 1'd0;

    else if(minute != r\_minute)

        min\_flag <= 1'b1;

    else

        min\_flag <= 1'b0;

end

always@(posedge clk or posedge rst)begin

    if(rst)

        sec\_flag <= 1'd0;

    else if(second != r\_second)

        sec\_flag <= 1'b1;

    else

        sec\_flag <= 1'b0;

end

always@(posedge clk or posedge rst)begin

    if(rst)

        cnt <= 27'd0;

    else if(cnt == sec\_period)

        cnt <=27'd0;

    else

        cnt <= cnt + 27'd1;

end

always@(posedge clk or posedge rst)begin

     if(rst)

        r\_hour <= 6'd0;

    else

        case(hou\_bias)

            2'b00:begin

                if(min\_flag&&(r\_minute == 6'd0))

                begin

                    if(r\_hour == 6'd23)

                        r\_hour <= 6'd0;

                    else

                        r\_hour <= r\_hour + 6'd1;

                end

                else

                    r\_hour <= r\_hour;

            end

            2'b01:begin

                if(r\_hour == 6'd23)

                    r\_hour <= 6'd0;

                else

                    r\_hour <= r\_hour + 6'd1;

            end

            2'b11:begin

                if(r\_hour == 6'd0)

                    r\_hour <= 6'd23;

                else

                    r\_hour <= r\_hour - 6'd1;

            end

        endcase

end

always@(posedge clk or posedge rst)begin

    if(rst)

        r\_minute <= 6'd2;

    else

        case(min\_bias)

            2'b00:

            begin

                if(sec\_flag&&(r\_second == 6'd0))

                begin

                    if(r\_minute == 6'd59)begin

                        r\_minute <= 6'd0;

                        blink\_flag <= blink\_flag | 2'b01;

                    end

                    else begin

                        r\_minute <= r\_minute + 6'd1;

                        blink\_flag <= blink\_flag & 2'b10;

                    end

                end

                else

                    r\_minute <= r\_minute;

            end

            2'b01:

            begin

                if(r\_minute == 6'd59)

                    r\_minute <= 6'd0;

                else

                    r\_minute <= r\_minute + 6'd1;

            end

            2'b11:

            begin

                if(r\_minute == 6'd0)

                    r\_minute <= 6'd59;

                else

                    r\_minute <= r\_minute - 6'd1;

            end

        endcase

end

always@(posedge clk or posedge rst)begin

    if(rst)

        r\_second <= 6'd58;

    else

        case(sec\_bias)

            2'b00:begin

                if(cnt == sec\_period)

                begin

                    if(r\_second == 6'd59)begin

                        r\_second <= 6'd0;

                        blink\_flag <= blink\_flag | 2'b10;

                    end

                    else    begin

                        r\_second <= r\_second + 6'd1;

                        blink\_flag <= blink\_flag & 2'b01;

                    end

                end

            end

            2'b01:begin

                if(r\_second == 6'd59)

                    r\_second <= 6'd0;

                else

                    r\_second <= r\_second + 6'd1;

            end

            2'b11:begin

                if(r\_second == 6'd0)

                    r\_second <= 6'd59;

                else

                    r\_second <= r\_second - 6'd1;

            end

            default:r\_second <= r\_second;

        endcase

end

endmodule

18.管脚约束

IO\_LOC "led1[1]" 13;

IO\_PORT "led1[1]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "led1[0]" 15;

IO\_PORT "led1[0]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "led0[1]" 17;

IO\_PORT "led0[1]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "led0[0]" 19;

IO\_PORT "led0[0]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "CS[3]" 44;

IO\_PORT "CS[3]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "CS[2]" 45;

IO\_PORT "CS[2]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "CS[1]" 46;

IO\_PORT "CS[1]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "CS[0]" 47;

IO\_PORT "CS[0]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "data[6]" 35;

IO\_PORT "data[6]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "data[5]" 38;

IO\_PORT "data[5]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "data[4]" 39;

IO\_PORT "data[4]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "data[3]" 40;

IO\_PORT "data[3]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "data[2]" 41;

IO\_PORT "data[2]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "data[1]" 42;

IO\_PORT "data[1]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "data[0]" 43;

IO\_PORT "data[0]" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "DP" 34;

IO\_PORT "DP" IO\_TYPE=LVCMOS33 PULL\_MODE=UP DRIVE=8;

IO\_LOC "alarm\_en" 33;

IO\_PORT "alarm\_en" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

IO\_LOC "key\_4" 11;

IO\_PORT "key\_4" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

IO\_LOC "key\_3" 14;

IO\_PORT "key\_3" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

IO\_LOC "key\_2" 16;

IO\_PORT "key\_2" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

IO\_LOC "key\_1" 18;

IO\_PORT "key\_1" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

IO\_LOC "sw4" 30;

IO\_PORT "sw4" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

IO\_LOC "sw3" 31;

IO\_PORT "sw3" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

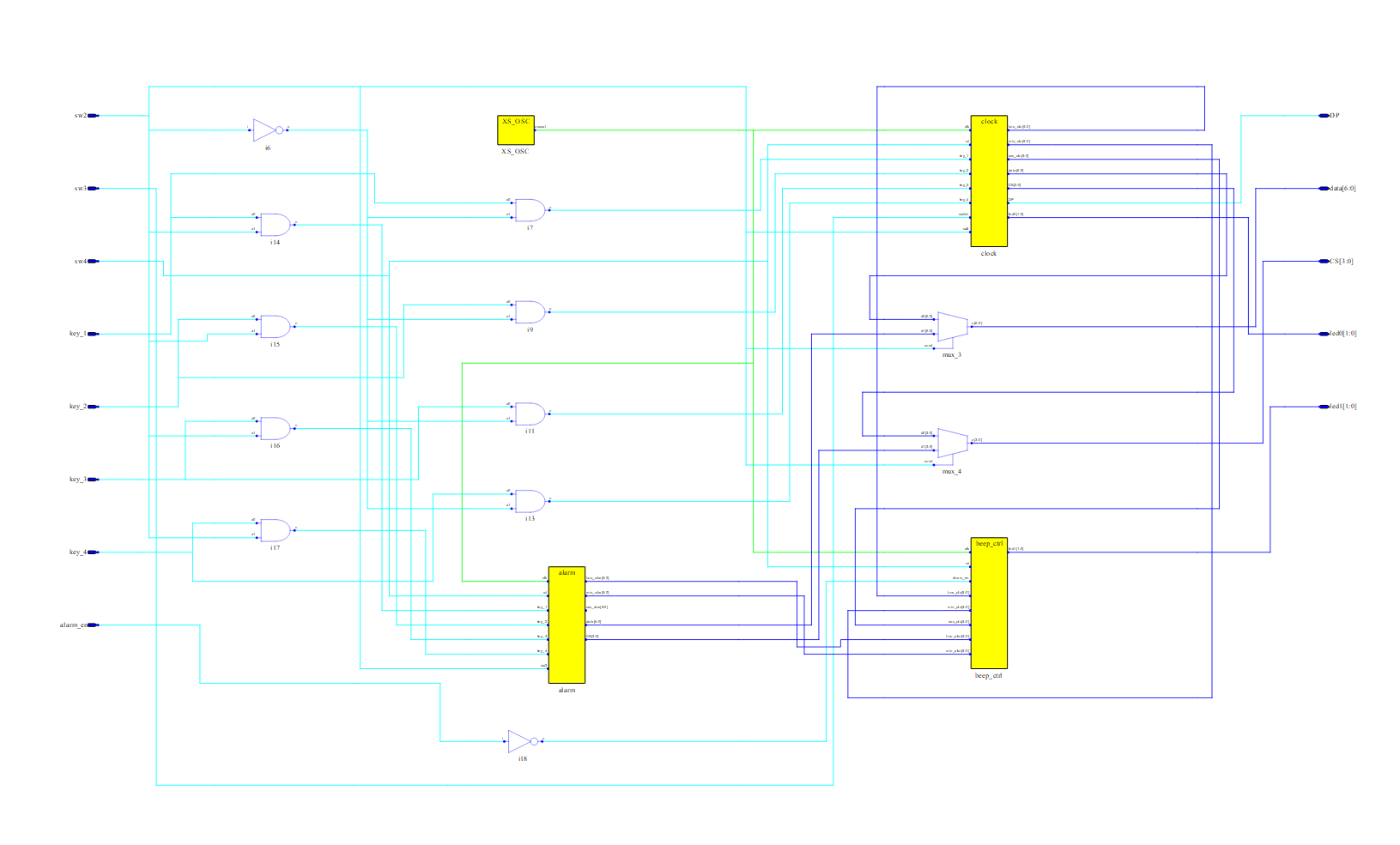
IO\_LOC "sw2" 32;

IO\_PORT "sw2" IO\_TYPE=LVCMOS33 PULL\_MODE=UP;

1. **实验过程**

将上述模块整理加载到Gowin FPGA Designer中，进行整体的编码操作。待无错误警告时，进行管脚的配置并用杜邦线将相应的接口连接。

数字钟结构如下图所示



在编译完成后，用数据传输线将FPGA与电脑相连，进行代码的输送，输送完毕后检验其运行结果即可。

1. **实验结果**

能够稳定进行分屏数字显示，并用第四位数字的小数点的闪烁表示秒数，整个FPGA可以正常进行时间的计数，秒数每过60分钟数加一，分钟每过60小时数加一，在24小时进制下，小时位计数到24时会自动回零。

可以通过调节SW3进行24/12小时的转换，12小时制下，第三个数码管右下角有“.”表示下午，否则表示上午。通过按下KEY3，可以进入调时模式，选择调整小时数还是分钟数，被调节对象会进行闪烁。KEY1进行每按一下对应值加一，KEY2每按一下对应数值减一。

实现了所有的基础功能。

闹钟功能：上拨sw2，进入小时设置模式，与调节时间原理一样设置闹钟时间。到达对应时间后，会有两个LED灯闪烁报时。可通过sw1选择打开还是关闭闹钟。

整点报时功能：当时间达到整点时，LED灯闪烁。默认打开，需要关闭可将对应引脚接地。

1. **实验总结**

这次Verilog实验让我进一步学习了这门语言的使用，同时在模块化层次化设计上给了我更多的压力。同时，由于Verilog没有十分便捷的debug手段，在实际寻找代码问题中，让我充分意识到自己代码风格仍然又过于随意的特点，并给实际编程过程带来了许多麻烦。同时，我也进一步体会到Verilog在处理这种问题时的巨大劣势，以及其在逻辑上更加严格的需求。

总之是很有收获的一次实验。