# 数字逻辑抽油烟机project项目文档

Name: 李家成
StudentID: 12311024
Class: 周三56

Name: 崔子璇
StudentID: 12311007
class: 周三56

Name: 程舒格
StudentID: 12311554
Class: 周三56

March 10, 2025

# Contents

# 1 团队分工

## 1.1 各成员工作简述

以下是团队成员的职责分工和贡献比例:

- 李家成：负责主模块整体框架的构建和各模块的整合工作、bonus功能开发，贡献比例 34%
- 崔子璇：负责项目文档的制作,调节参数模块，led灯显示模块，贡献比例 33%
- 程舒格：负责主模块的逻辑构建和手势开关模块的开发与调试，贡献比例 33%

# 2 开发计划日程安排和实施情况

## 2.1 开发计划日程安排

- 先系统性模块性建立整个project框架，然后再将每个模块分给各个负责人
- 各部分完成之后合并代码，统一到整个project代码中，至此基础功能部分代码完成（未调试）
- 对于整体已完成的代码进行仿真测试
- 上板调试
- 高级功能

## 2.2 实施情况

- 第一步，框架构建，我们团队的leader做的很不错的，把主模块搭建好，将模块理顺并划分给我们三个人来做，并且很快便把所有基础功能部分搞定！
- 第二步，仿真测试，我们选择合并所有代码之后统一仿真，在稍微修改后通过了所有仿真测试。
  - 但是这样写却带来了一个致命的错：忽略延迟的作用——仿真行为和上板行为有极大的差异，我们是一口气合并所有代码，我们最应该的做法是做完一个模块的仿真后立刻进行上板测试然后再进行下一个模块。
- 第三步，上板测试，我们在一周之内de出了大部分bug,并赶在14周周末将所有功能调试成功。

# 3 系统功能列表

## 3.1 基础功能

- 开关机功能
- 模式切换功能（待机模式、抽油烟模式、自清洁模式）
- 数码管显示各参数的功能（包括倒计时）
- 设置模式下修改参数的功能
- 辅助功能（照明等）

## 3.2 高级功能

- 用按键操作模拟手势开关实现开关机功能
- 使用python键盘通过uart端口输入作为输入设备
- 使用python接收uart端口输出并用UI呈现数据

### 3.3 其他功能

- 给开机按键加密码锁(默认为00000001)，只有用键盘输入（用uart输入实现）正确密码，才能激活开机键

## 4 系统使用说明（含系统的输入、输出端口的说明）

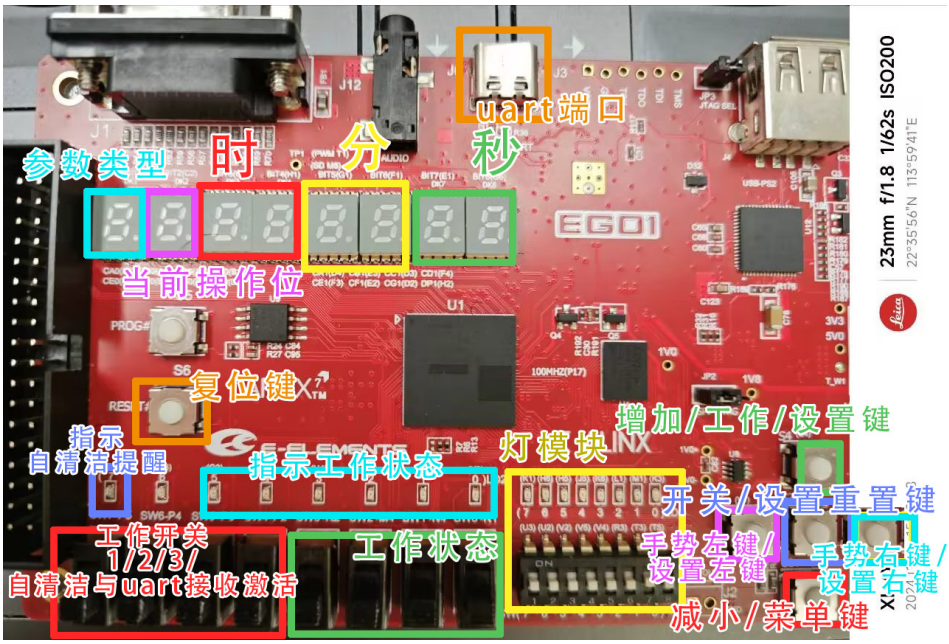


Figure 1: System Info

### 4.1 系统输入说明

- 按键和拨码开关说明:
  1. 关机状态下，中键为开机键，短按开机；长按左键开启倒计时，短按右键进入闲置状态;
  2. 闲置状态下，长按开机键进入关机，长按右键开启倒计时，短按左键进入关机状态；上键为设置键，下键为菜单键，短按设置键进入打开设置状态，或长按菜单键进入打开菜单状态;
  3. 设置状态下，左右键来控制当前操作位（时，分，秒）的变化，上下键是增加和减小键;工作显示状态的四个拨码开关控制数码管显示的参数类型，如0001显示自清洁提醒时间;
  4. 菜单状态下，当自清洁拨码开关上拨进入自清洁状态，若未上拨同时按下工作键，依次检查工作开关3，2，1的拨码开关进入对应工作状态。
- 指拨开关和小led灯：开机状态下，打开几个灯算作几档亮度，显示对应的几盏灯。
- 复位键 rst_n
- uart接收

### 4.2 系统输出功能

系统的输出端口包括:

- 大led灯：左一位显示自清洁提醒，右六位指示工作状态;
- 小led灯：灯模块的输出;

- 数码管：显示参数，左一指示参数类型，如1表示“当前时间”；左二表示当前操作位，如1表示“时”；右六位表示参数数据的时分秒划分；
- uart发送。

# 5 系统结构说明（系统内部各模块及模块间信号线的关系）

## 5.1 模块说明

系统由以下主要模块组成（由verilog代码结构逐一介绍）：

- 模块1：Const 头文件定义默认参数，包括工作参数和区分长按短按的按键Timer参数（不可变）；

```
`define ReminderHours_default 6'b001010
`define SelfCleanMinutes_default 6'b000011
`define HurricaneMinutes_default 6'b000001
`define GestureTime_default 6'b000101
`define CloseTime_default 6'b000011
`define LeftTimer_default 6'b000001
`define RightTimer_default 6'b000001
`define MENUTimer_default 6'b000001
`define WORKTimer_default 6'b000010
`define SETTING_WAINTING_TIMER_default 6'b000011
```

- 模块2：Module KeyBoard 定义整个project需要的输入和输出并且对子模块进行实例化；

```
 module Keyboard(
    input wire clk,
    input wire rst_n,
    input wire [4:0] btn,
    input wire [7:0] B_sw,
    input wire [7:0] S_sw,
    input wire uart_rx,
    output wire uart_tx,
    output wire [7:0] led,
    output wire [7:0] S_led,
    output wire [7:0] seg,
    output wire [7:0] seg1,
    output wire [7:0] an
    wire [3:0] Setting_State = {B_sw[3],B_sw[2],B_sw[1],B_sw[0]}
    always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
        rx_open <= 1'b0;
    end else begin
    case(rx_data_out)
        8'b00110001,8'b00000001: rx_open <= 1'b1;
    endcase
    end
end
);
```

  - input:
    - wire clk:时钟信号
    - wire rst_n: reset键,
    - wire [4:0] Btn：5个按键
    - wire [7:0] B_sw: ego1板上左下一排大的拨码开关
    - wire [7:0] S_sw: ego1板上中下一排较小的拨码开关

    - wire uartrx： uart输出端口

  - output:
    - wire uart_tx
    - wire [7:0]led: ego1板上左边八个大拨码开关上方的8个led灯
    - wire [7:0] s_led: ego1板上按钮左边8个小led灯
    - wire [7:0] seg:第一组数码管
    - wire [7:0] seg1:第二组数码管
    - wire [7:0] an:使能信号，管理一共8个数码管的使能信号

  - wire [3:0] Setting_State = B_sw[3],B_sw[2],B_sw[1],B_sw[0]: 设置状态键，它将与按钮结合共同控制当前状态

  - always语句块:

- 模块3：module Controller 它是module KeyBoard的子模块，它在module KeyBoard中被实例化，信号线绑定关系如下：

```
Controller Controller_init(
    .clk(clk),
    .rst_n(rst_n),
    .OPEN_button(btnOut1 & rx_open),
    .MENU_button(btnOut2),
    .Right_button(btnOut3),
    .WORK_button(btnOut4),
    .Left_button(btnOut5),

    .Light_Witch(S_sw),
    .Light(S_led),
    .led(led),
    .seg(seg),
    .seg1(seg1),
    .btn({btnOut5,btnOut4,btnOut3,btnOut2,btnOut1}),
    .uart_tx(uart_tx),
    .rx_data_out(rx_data_out),
    .WORK1_Witch(B_sw[7]),
    .WORK2_Witch(B_sw[6]),
    .WORK3_Witch(B_sw[5]),
    .SELF_CLEAN_Witch(B_sw[4]),
    .Setting_State(Setting_State),
    .an(an)
);
```

  - .clk(clk): 绑定时钟信号
  - . rst_n (rst_n):绑定reset键
  - . OPEN_button ( btnOut1 & rx_open ):绑定开机按钮
  - . MENU_button ( btnOut2 ) :绑定菜单键
  - . Right_button ( btnOut3 )：右键绑定第三个输出
  - . WORK_button ( btnOut4 )： 工作按钮
  - . Left_button ( btnOut5 )： 左按钮
  - . Light_Witch ( S_sw )： 右按钮:
  - . Light ( S_led )
  - . led ( led )
  - . seg ( seg )
  - . seg1 ( seg1 )

- . btn ( btnOut5 , btnOut4 , btnOut3 , btnOut2 , btnOut1 )
- . uart_tx ( uart_tx )
- . rx_data_out ( rx_data_out )
- . WORK1_Witch ( B_sw [7])
- . WORK2_Witch ( B_sw [6])
- . WORK3_Witch ( B_sw [5])
- . SELF_CLEAN_Witch ( B_sw [4])
- . Setting_State ( Setting_State )
- . an ( an )

- 模块4: module rx_controller 它是module KeyBoard的子模块，它在module KeyBoard中被实例化，信号线绑定关系如下：

```
rx_controller rx_control_init(
    .clk(clk),
    .rst_n(rst_n),
    .btn({btnOut5,btnOut4,btnOut3,btnOut2,btnOut1}),
    .uart_rx(uart_rx),
    .data_out(rx_data_out),
    .data_ready(rx_ready)
);
```

  - .clk ( clk ):绑定时钟信号
  - .rst_n ( rst_n ):绑定reset键
  - .btn ( btnOut5 , btnOut4 , btnOut3 , btnOut2 , btnOut1 ):绑定5个去抖后的按钮
  - .uart_rx ( uart_rx ): 绑定uart接收端口
  - .data_out ( rx_data_out ):绑定uart接收到的数据
  - .data_ready ( rx_ready ):接收数据是否有效

- 模块5： module debounce 消抖 它是module KeyBoard的子模块，它在module KeyBoard 中被实例化，信号线绑定关系如下：

  说明：我们主要对于5个button进行消抖，所以在KeyBoard模块下将debounce模块实例化5次

```
debounce debounce1(
    .clk(clk),
    .rst_n(rst_n),
    .btn(btn[0]),
    .btnOut(btnOut1)
);

debounce debounce2(
    .clk(clk),
    .rst_n(rst_n),
    .btn(btn[1]),
    .btnOut(btnOut2)
);

debounce debounce3(
    .clk(clk),
    .rst_n(rst_n),
    .btn(btn[2]),
    .btnOut(btnOut3)
);

debounce debounce4(
```

```
    .clk(clk),
    .rst_n(rst_n),
    .btn(btn[3]),
    .btnOut(btnOut4)
);

debounce debounce5(
    .clk(clk),
    .rst_n(rst_n),
    .btn(btn[4]),
    .btnOut(btnOut5)
);
```

- .clk(clk)： 绑定时钟信号
- .rst_n(res_n):绑定reset键
- .btn(btn[0-4]) :绑定5个按钮
- .btnOut(btnOut 1-5) :绑定输出5个按钮的状态

## 5.2 schematic diagram

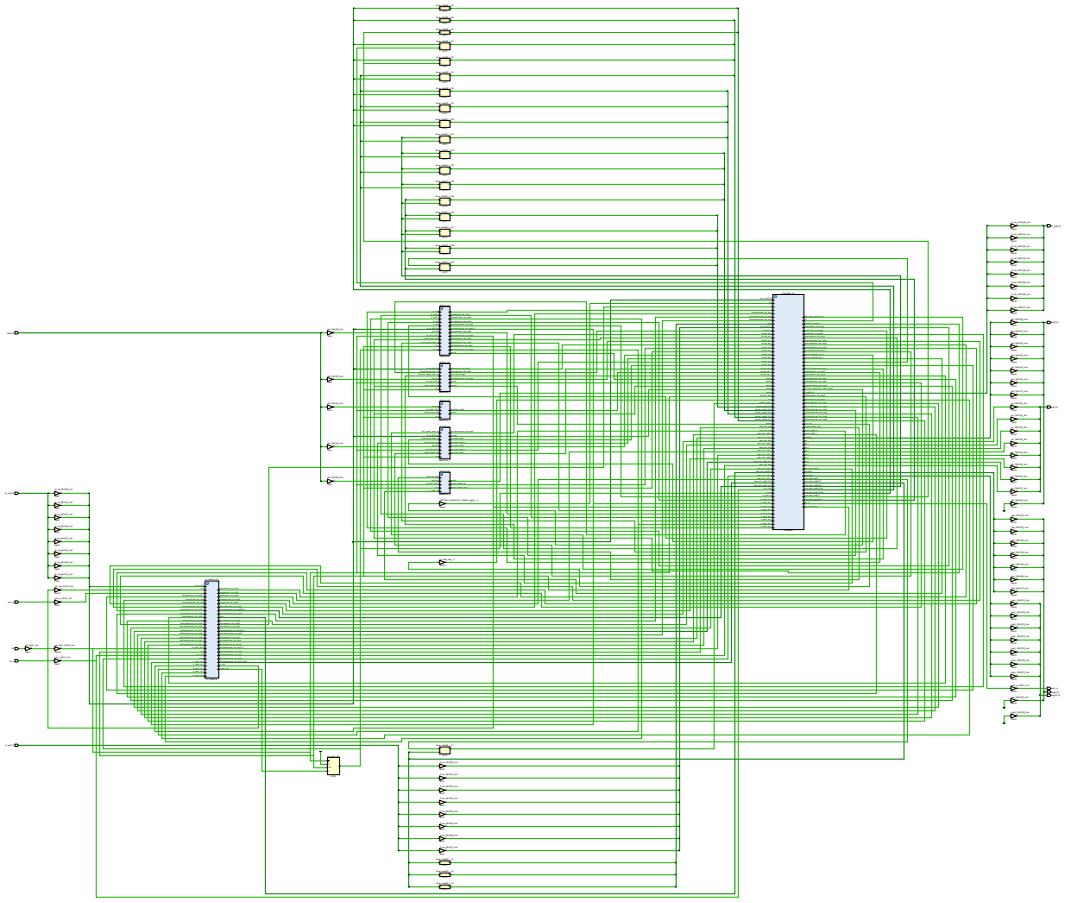

Figure 2: Schematic photo

# 6 子模块功能说明：输入、输出端口说明、子模块功能说明

- **模块A: module Controller**:
  - 输入：描述输入信号
  - 输出：描述输出信号
  - 功能说明：
    - 负责状态机的模式转换；
    - 负责各种时间的计时（当前时间，工作时间和倒计时，以及各按键按下后的Timer计时），以及当前时间和工作时间的设置和清空，详见下方(设置方面代码与SETTING模块用法一致，详见SETTING)：

```
        if (current_state == STANDBY && OPEN_button == 1) begin
            CloseTimer <= CloseTimer + 1;
        end else if (~OPEN_button) begin
            CloseTimer <= 0;
        end

        if (current_state==OFF && Left_button == 1) begin
            LeftTimer <= LeftTimer + 1;
        end else if (~Left_button) begin
            LeftTimer <= 0;
        end

        if (current_state==STANDBY && Right_button == 1) begin
            RightTimer <= RightTimer + 1;
        end else if (~Right_button) begin
            RightTimer <= 0;
        end

        if (current_state==STANDBY && MENU_button == 1) begin
            MENUTimer <= MENUTimer + 1;
        end else if (~MENU_button) begin
            MENUTimer <= 0;
        end

        if (current_state==SETTING && WORK_button == 1) begin
            WORKTimer <= WORKTimer + 1;
        end else if (~WORK_button) begin
            WORKTimer <= 0;
        end
```

    - 负责各功能模块的激活。
  - 实现步骤重点：
    - 在必要的一些状态基础上，添加了几个特别的状态：

```
    OFF_TO_STANDBY = 6'b000011, // Shutdown to an idle
        intermediate state,used to do count down.
    STANDBY_TO_OFF = 6'b000111, // a intermediate state between
        idle to shutdown.
    MANUAL = 6'b001111, // when you have press the mune button
    SETTING = 6'b111111, // where you can change the working
        parameter.
```

    - 对于含倒计时的状态转换（以从OFF_TO_STANDBY为例，自清洁状态设计同理）：

```
     OFF_TO_STANDBY:
        begin
            if (clk_div) begin // clk_div: 1s clock signal
                if (counter > 0) begin
```

```
                    counter <= counter - 1;
                end
            end
            if(counter>0)
            begin
                if(Right_button)begin
                    next_state <= STANDBY;
                    IsHurricaneEnable <= 1'b1;
            counter_enable <= 1'b0; // count down enable signal
            // not need to set counter = 0;
                end
                else next_state <= OFF_TO_STANDBY;
            end else begin
                counter_enable <= 1'b0;
                next_state <= OFF;
            // not need to set counter = 0;
            end
        end
```

· 状态与模块逻辑分开，模块的激活不取决于状态，而取决于模块的对应的trigger信号，比如：Light模块在开机状态下都能打开，我们没有定义一个开机状态，而是定义了一个WORKING_trigger作为所有开机状态的输出信号，只要将该信号作为激活信号绑定给其他模块就能实现状态管控，并且模块状态分离，方便拓展；

```
reg Working_trigger; // if open
reg WorkTimer_trigger; // if working
reg clean_trigger;
reg Setting_trigger;
```

· 对应trigger输出逻辑：

```
    case(current_state)
        OFF,OFF_TO_STANDBY:
        begin
            led[5:0] <= OFF;
            Working_trigger <= 1'b0;
            WorkTimer_trigger <= 1'b0;
            clean_trigger <= 1'b0;
            Setting_trigger <= 1'b0;
        end
        STANDBY,STANDBY_TO_OFF:
        begin
            led[5:0] <= STANDBY;
            Working_trigger <= 1'b1;
            WorkTimer_trigger <= 1'b0;
            clean_trigger <= 1'b0;
            Setting_trigger <= 1'b0;
        end
        MANUAL:
        begin
            led[5:0] <= MANUAL;
            Working_trigger <= 1'b1;
            WorkTimer_trigger <= 1'b0;
            clean_trigger <= 1'b0;
            Setting_trigger <= 1'b0;
        end
        SELF_CLEAN:
        begin
            led[5:0] <= SELF_CLEAN;
            Working_trigger <= 1'b1;
            WorkTimer_trigger <= 1'b0;
            clean_trigger <= 1'b1;
```

```
            Setting_trigger <= 1'b0;
        end
        SETTING:
        begin
            led[5:0] <= SETTING;
            Working_trigger <= 1'b1;
            WorkTimer_trigger <= 1'b0;
            clean_trigger <= 1'b0;
            Setting_trigger <= 1'b1;
        end
        ONWORK1:
        begin
            led[5:0]<= ONWORK1;
            Working_trigger <= 1'b1;
            WorkTimer_trigger <= 1'b1;
            clean_trigger <= 1'b0;
            Setting_trigger <= 1'b0;
        end
        ONWORK2:
        begin
            led[5:0]<= ONWORK2;
            Working_trigger <= 1'b1;
            WorkTimer_trigger <= 1'b1;
            clean_trigger <= 1'b0;
            Setting_trigger <= 1'b0;
        end
        ONWORK3:
        begin
            led[5:0]<= ONWORK3;
            Working_trigger <= 1'b1;
            WorkTimer_trigger <= 1'b1;
            clean_trigger <= 1'b0;
            Setting_trigger <= 1'b0;
        end
    endcase
```

* 实例关系说明：在该模块下，对以下子模块都进行实例化，用到了模块管控原理和上面谈到的相同，即每个模块都有一个对应的trigger。

– **模块B：module FrequencyDivider**

* 输入：10ns时钟信号；
* 输出：分频好的1s时钟信号；
* 功能说明：分频1s时钟信号；
* 实现步骤：用两个14位宽的寄存器进行计数，完成对10ns的分频。

– **模块C：module divclk**

* 输入：10ns时钟信号；
* 输出：分别实现了四种时钟，1ms，500ms，9600HZ，9600*16HZ；
* 功能说明：实现了一系列时钟的分频，以方便在必要时直接实例化改模块使用。

– **模块D：module SETTING**

* 输入：其trigger被绑于Setting_trigger，左右键（被绑于controller中的左右键）更换操作位，上下键（controller中的工作键和菜单键）调节参数增减，中间键（controller中的开机键）进行回复出厂设置或清零，四位宽设置状态选择当前操作的参数类型；

```
    input wire [3:0] Setting_State,
    input wire ADD_button,
    input wire REDUCE_button,
    input wire Left_button,
    input wire Right_button,
    input wire reset_button,
```

* 输出：输出端口为各参数的结果和操作位（在该模块内使用以及输出到controller模块中使用）；

```
    output reg [5:0] ReminderHours_reg,
    output reg [5:0] SelfCleanMinutes_reg,
    output reg [5:0] HurricaneSeconds_reg,
    output reg [5:0] GestureTime_reg,
    output reg [5:0] ReminderMinutes_reg,
    output reg [5:0] SelfCleanSeconds_reg,
    output reg [5:0] HurricaneMinutes_reg,
    output reg [5:0] GestureMinutes_reg,
    output reg [5:0] ReminderSeconds_reg,
    output reg [5:0] SelfCleanHours_reg,
    output reg [5:0] HurricaneHours_reg,
    output reg [5:0] GestureHours_reg,
    output reg [2:0] current_pos // operating position
```

* 功能说明：实现能通过左右键调控操作时分秒，对不同的参数从时或分或分进行增减或重置，具体代码详见下方。
  · 操作位：

```
        case(current_pos)
    hour_pos: begin
       if(Right_button && !Right_button_state) begin
           Right_button_state <= 1'b1;
           next_pos <= minute_pos;
       end else if (!Right_button) begin
           Right_button_state <= 1'b0;
       end
    end
    minute_pos: begin
        // if(Setting_State == 4'b1001)
        // begin
        if(Right_button && !Right_button_state) begin
           Right_button_state <= 1'b1;
           next_pos <= second_pos;
       end else if (!Right_button) begin
           Right_button_state <= 1'b0;
       end
        // end
        // else if(Setting_State == 4'b0110)
        // begin
        if(Left_button && !Left_button_state) begin
            Left_button_state <= 1'b1;
            next_pos <= hour_pos;
        end else if (!Left_button) begin
            Left_button_state <= 1'b0;
        end
        // end else begin
            // next_pos <= minute_pos;
        // end
    end
    second_pos: begin
        if(Left_button && !Left_button_state) begin
           Left_button_state <= 1'b1;
           next_pos <= minute_pos;
       end else if (!Left_button) begin
           Left_button_state <= 1'b0;
       end
    end
    default: next_pos <= second_pos;
    endcase
```

· 增加逻辑（减小逻辑和重置逻辑一致）：

```
        if (ADD_button && !ADD_button_state) begin
        ADD_button_state <= 1'b1;
        case(Setting_State)
        4'b0001: begin
            case(current_pos)
            hour_pos: ReminderHours_reg <= ReminderHours_reg + 1;
            minute_pos: begin if(ReminderMinutes_reg == 59) ReminderMinutes_reg <= 0; else ReminderMinutes_reg <= ReminderMinutes_reg + 1; end
            second_pos: begin if(ReminderSeconds_reg == 59) ReminderSeconds_reg <= 0; else ReminderSeconds_reg <= ReminderSeconds_reg + 1; end
            endcase
        end
        4'b0010: begin
            case(current_pos)
            hour_pos: SelfCleanHours_reg <= SelfCleanHours_reg + 1;
            minute_pos: begin if(SelfCleanMinutes_reg == 59) SelfCleanMinutes_reg <= 0; else SelfCleanMinutes_reg <= SelfCleanMinutes_reg + 1; end
            second_pos: begin if(SelfCleanSeconds_reg == 59) SelfCleanSeconds_reg <= 0; else SelfCleanSeconds_reg <= SelfCleanSeconds_reg + 1; end
            endcase
        end
        4'b0100: begin
            case(current_pos)
            hour_pos: HurricaneHours_reg <= HurricaneHours_reg + 1;
            minute_pos: begin if(HurricaneMinutes_reg == 59) HurricaneMinutes_reg <= 0; else HurricaneMinutes_reg <= HurricaneMinutes_reg + 1; end
            second_pos: begin if(HurricaneSeconds_reg == 59) HurricaneSeconds_reg <= 0; else HurricaneSeconds_reg <= HurricaneSeconds_reg + 1; end
            endcase
        end
        4'b1000: begin case(current_pos)
            hour_pos: GestureHours_reg <= GestureHours_reg + 1;
            minute_pos: begin if(GestureMinutes_reg == 59) GestureMinutes_reg <= 0; else GestureMinutes_reg <= GestureMinutes_reg + 1; end
            second_pos: begin if(GestureTime_reg == 59) GestureTime_reg <= 0; else GestureTime_reg <= GestureTime_reg + 1; end
            endcase
        end
        endcase
    end else if (!ADD_button) begin
        ADD_button_state <= 1'b0;
    end
```

– **模块E：ShowLED**

* 输入1：其trigger被绑于Working_trigger;
* 输入2：一系列六位宽的显示参数和24位宽倒计时计数器以及使能信号;

```
    input[5:0] Hour,
    input[5:0] Minute,
    input[5:0] Second,
    input[5:0] WorkTimeHour,
    input[5:0] WorkTimeMinute,
    input[5:0] WorkTimeSecond,
    input[5:0] ReminderHours,
    input[5:0] SelfCleanMinutes,
    input[5:0] HurricaneSeconds,
    input[5:0] GestureTime,
    input[5:0] ReminderMinutes,
    input[5:0] SelfCleanSeconds,
    input[5:0] HurricaneMinutes,
    input[5:0] GestureMinutes,
    input[5:0] ReminderSeconds,
    input[5:0] SelfCleanHours,
    input[5:0] HurricaneHours,
    input[5:0] GestureHours,
    input[23:0] counter,
    input counter_enable,
```

* 输入1：操作位和设置显示状态;

```
    input [2:0] current_pos, // operating position
    input[3:0] Setting_State,
```

* 输出：两个八位宽数码数码显示管和八位宽使能信号;
* 内部变量：showNumWithDot指当前显示的参数类型，如1表示当前时间，2表示工作时间等，operationNum表示当前操作位，1表示时，2表示分，3表示秒，show_data是要显示参数的24位BCD码形式，number是要通过uart输出的拼接所有上面变量的数据输出（具体绑定详见下方bonus的uart部分）;

```
    reg[3:0] ShowNumWithDot;
    reg[3:0] operationNum;
    reg[23:0] show_data;
    wire [31:0] number;
    assign number = {ShowNumWithDot, operationNum, show_data};
```

* 功能说明:
  · 在不同的Setting_State输入下，显示不同的数据;

```
    case(Setting_State)
        4'b1111:begin
            show_data <= {bin_to_bcd(WorkTimeHour), bin_to_bcd(
                WorkTimeMinute), bin_to_bcd(WorkTimeSecond)};
            $display("show work time: %d",WorkTimeSecond);
            ShowNumWithDot <= 4'h2;
        end
        4'b0001:begin
            show_data <= {bin_to_bcd(ReminderHours), bin_to_bcd
                (ReminderMinutes), bin_to_bcd(ReminderSeconds)};
            ShowNumWithDot <= 4'h3;
        end
        4'b0010:begin
            show_data <= {bin_to_bcd(SelfCleanHours),
                bin_to_bcd(SelfCleanMinutes), bin_to_bcd(
                SelfCleanSeconds)};
            ShowNumWithDot <= 4'h4;
```

```
        end
        4'b0100:begin
            show_data <= {bin_to_bcd(HurricaneHours),
                bin_to_bcd(HurricaneMinutes), bin_to_bcd(
                HurricaneSeconds)};
            ShowNumWithDot <= 4'h5;
        end
        4'b1000:begin
            show_data <= {bin_to_bcd(GestureHours), bin_to_bcd(
                GestureMinutes), bin_to_bcd(GestureTime)};
            ShowNumWithDot <= 4'h6;
        end
        default:begin
            show_data <= {bin_to_bcd(Hour), bin_to_bcd(Minute),
                 bin_to_bcd(Second)};
            $display("show time: %d",Second);
            ShowNumWithDot <= 4'h1;
        end
    endcase
```

· 对要显示的数据，将每一个两位数参数通过如下定义函数分别转换成八位BCD码，倒计时通过另一个函数将24位二进制数转16位BCD码

```
    function [7:0] bin_to_bcd;
        input [5:0] bin;    // 7-bit binary input
        reg [3:0] bcd_low; // Low 4 bits of BCD output
        reg [3:0] bcd_high; // High 4 bits of BCD output
        // reg [2:0] bcd_high_temp; // Temporary variable for
            high 4 bits of BCD output
        begin
            bcd_high = bin / 10;  // Integer division to get
                the tens place
            bcd_low = bin % 10;   // Modulo operation to get
                the units place
            bin_to_bcd = {bcd_high, bcd_low}; // Concatenate
                the high and low bits to get the 8-bit BCD
                output
        end
    endfunction

    function [15:0] bin24_to_bcd16;
        input [23:0] bin;    // 7-bit binary input
        reg [5:0] bin1;
        reg [5:0] bin2;
        reg [7:0] bcd_low; // Low 4 bits of BCD output
        reg [7:0] bcd_high; // High 4 bits of BCD output
        begin
            bin1 = bin / 60;
            bin2 = bin % 60;
            bcd_high = bin_to_bcd(bin1);
            bcd_low = bin_to_bcd(bin2);
            bin24_to_bcd16 = {bcd_high, bcd_low}; //
                Concatenate the high and low bits to get the 8-
                bit BCD output
        end
    endfunction
```

· 与SETTING模块形成配合，实时显示参数和修改参数。

– **模块F：module Light**

* 输入,输出:

```
    input wire clk,
    input wire rst_n,
    input wire trigger,            // activation signal bound WORKING_trigger
    input wire[7:0] Light_Witch,       // bound direct dial switch
    output reg[7:0] light             // bound to small LED lights
```

* 功能说明：在开机模式下，开几个开关亮几个灯，意味着灯的挡位。

# 7 bonus 的实现说明

在本项目中，我们实现了以下 bonus 功能：

– uart输入相关：

* **模块1：module rx_controller – 实例化于KeyBoard主模块**：

```
module rx_controller (
    input wire clk,
    input wire rst_n,
    // input wire tx_start,
    input wire [4:0] btn,
    // input wire [7:0] tx_data_in,
    // input wire [31:0] number,
    input wire uart_rx,
    // output wire uart_tx,
    // output wire[7:0] led,
    output wire[7:0] data_out,
    output wire data_ready
);


wire clk_ms,clk_20ms,clk_16x,clk_x;
wire[15:0] data_disp;
wire data_error;


divclk my_divclk(
    .clk(clk),
    .clk_16x(clk_16x)
);

uart_rx uart_rx_init(
    .clk_16x(clk_16x),
    // .rst(btnout[0]),
    .rst_n(rst_n),
    .rxd(uart_rx),
    .data_disp(data_disp),
    .data_ready(data_ready),
    .data_error(data_error)
);
assign data_out = data_disp[7:0];
endmodule
```

  · 输入：详见前文Keyboard模块；
  · 输出：详见前文； 内部绑定：实例化divclk，得到clk_16x时钟（9600*16HZ），用这个时钟作为uart_rx模块的输入时钟，以便实现16倍采样，保证数据接收的稳定性；
  · 功能描述：rx接收的调控器，负责将程序与rx联系起来，并方便调控rx行为。

* **模块2：module uart_rx**

· 实现步骤:

· 定义四个状态:

```
            parameter idle=1,one=2,two=3,stop=4;
                    // one to deal with the 8 bit data;
                    // two to check Parity bits(even).
```

· 状态输出逻辑和状态逻辑:

```
                    always@(clk_16x_cnt)
begin
    if(clk_16x_cnt<='d8)
        next_state=idle;
    if(clk_16x_cnt>'d8 && clk_16x_cnt <= 'd136)
        next_state=one;
    if(clk_16x_cnt>'d136 && clk_16x_cnt <= 'd152)
        next_state=two;
    if(clk_16x_cnt>'d152 && clk_16x_cnt <= 'd168)
        next_state=stop;
    if(clk_16x_cnt > 'd168)
        next_state=idle;
end
always@(posedge clk_16x)
begin
    if(~rst_n)
    begin
        rxd1<=1'd1;
        rxd2<=1'd1;
        data_ready<='d0;
        clk_16x_cnt<='d0;
        start_flag<=0;
    end
    else begin
        case(present_state)
        idle: begin
              rxd1<=rxd;
              rxd2<=rxd1;
              if((~rxd1)&&rxd2)
                    start_flag<='d1;
              else if(start_flag==1)
                clk_16x_cnt<=clk_16x_cnt+'d1;
            end
         one: begin
                clk_16x_cnt<=clk_16x_cnt + 'd1;
                if(clk_16x_cnt=='d24)data_out[0]<=rxd;
                else if(clk_16x_cnt=='d40)data_out[1]<=rxd;
                else if(clk_16x_cnt=='d56)data_out[2]<=rxd;
                else if(clk_16x_cnt=='d72)data_out[3]<=rxd;
                else if(clk_16x_cnt=='d88)data_out[4]<=rxd;
                else if(clk_16x_cnt=='d104)data_out[5]<=rxd;
                else if(clk_16x_cnt=='d120)data_out[6]<=rxd;
                else if(clk_16x_cnt=='d136)data_out[7]<=rxd;
            end
         two: begin
                if(clk_16x_cnt=='d152)
                begin
                    if(rxd_buf==rxd) data_error<=1'd0;// not
                        error
                    else data_error<=1'd1;//error
                end
                clk_16x_cnt <= clk_16x_cnt+'d1;
            end
```

```
53          stop: begin
54                  if(clk_16x_cnt=='d168)
55                  begin
56                      if(1'd1==rxd)
57                      begin
58                          data_error<=1'd0;
59                          data_ready<='d1;
60                      end
61                      else begin
62                          data_error<=1'd1;
63                          data_ready<='d0;
64                      end
65                  end
66                  data_out1<=data_out;
67                  if(clk_16x_cnt>168)
68                  begin
69                      clk_16x_cnt <= 0;
70                      start_flag <= 0;
71                  end
72                  else
73                      clk_16x_cnt <= clk_16x_cnt + 'd1;
74              end
75          endcase
76      end
77  end
```

– uart输出相关:

* **模块1：module uart_send_decimal – 在ShowLED进行实例化**
  · 绑定关系:

```
uart_send_decimal uart_send_decimal_init(
    .clk_x(clk_x),
    .clk_500ms(clk_500ms),
    .rst_n(rst_n),
    .number(number),
    .btn(btn),
    .uart_tx(uart_tx)
);
```

  · 输入：number为32位BCD码（即在基础功能在数码管显示的数据），clk_500ms为经过分频的0.5s的信号，btn是5位宽的所有按键;
  · 输出：uart_tx为uart输出端口;
  · 功能描述：用按键按下和0.5s时钟信号同时（取或）作为触发发送的信号，（按键按下在于可以让修改参数时立刻能反应到电脑输出上，0.5时钟是让当前时间等时间变化能及时反应到电脑输出）每次发送都会发送执行4位宽BCD码八次，即32位宽BCD（十进制8位数）一位一位发送。

* **模块2：module uart_tx – 在uart_send_decimal进行实例化**
  · 绑定关系:

```
    uart_tx uart_tx_init(
        .clk_x(clk_x),
        .data_in(tx_data),
        .btn({4'b0,tx_start}),
        .txd(uart_tx),
        .rst_n(rst_n),
        .uart_tx_done(uart_tx_done)
    );
```

  · 输入：使用9600HZ时钟信号;

- 功能说明：一个基本的uart_tx模块，每次发送一个8位宽数据（一位数），与uart_rx实现类似，故在此略过。

* **python代码实现：具体代码详见UI文件夹的UI.py文件**

  - 功能说明：代码实现了一个基于Tkinter的图形用户界面(GUI)应用程序，用于从串行端口接收数据，并在GUI中显示。代码定义了数据处理函数analyzeData，将类型和操作数以及时，分，秒分割出来，显示简单明了。同时，它还监听键盘事件，并且模拟了命令行输入数字，点击回车删除一个数，点击Enter键进行数据发送，十进制数据会转换成BCD码后发送。

```python
def on_key_event(event):
    global inputValue
    if isdigit(event.name):
        inputValue.append(event.name)
    elif event.name == 'backspace':
        if inputValue:
            inputValue.pop()
    elif event.name == 'enter':
        if inputValue:
            num_str = ''.join(inputValue)
            print(f"Sending: {num_str}")
            send_serial_dataInBin(f"{int(num_str):08b}")
            inputValue.clear()
    elif event.name.lower() == 'q':
        print("Exiting...")
        ser.close()
        exit(0)
    else:
        print(f"Invalid key: {event.name}")

keyboard.on_press(on_key_event)
```

Figure 3: KeyBoardInput

```python
def analyzeData(data):
    type = data[0]
    pos = data[1]
    pos_type = ''
    match pos:
        case '0':
            pos_type = '无'
        case '1':
            pos_type = '时'
        case '2':
            pos_type = "分"
        case '3':
            pos_type = "秒"
        case _:
            pos_type = "Invalid"
            return None
    hours = int(data[2:4])
    minutes = int(data[4:6])
    seconds = int(data[6:8])
    info = f'{hours}小时, {minutes}分钟, {seconds}秒! '
    match type:
        case '1':
            return "当前时间: " + info +'\n'+ f"当前位置: {pos_type}"
        case '2':
            return "工作时间: " + info + '\n'+ f"当前位置: {pos_type}"
        case '3':
            return "自清洁提醒时间: " + info +'\n'+ f"当前位置: {pos_type}"
        case '4':
            return "自清洁时间: " + info + '\n'+f"当前位置: {pos_type}"
        case '5':
            return "暴风模式时间: " + info +'\n'+ f"当前位置: {pos_type}"
        case '6':
            return "手势识别时间: " + info +'\n'+ f"当前位置: {pos_type}"
        case '8':
            return "倒计时: " + info +'\n'+ f"当前位置: {pos_type}"
        case _:
            # return "Invalid number, Something went wrong"
            return None
```

Figure 4: AnalyzeData

# 8 项目总结

通过本次项目，我们总结出以下经验:

## 8.1 团队领导的重要性

– 团队的leader在项目中起着至关重要的作用，需要全面了解团队进度，及时应对不可控情况。例如，我们第一次上板测试时，由于没有充分准备，导致一个功能都无法实现，团队leader迅速调整方案，帮助团队重新进入正轨。

## 8.2 团队合作的重要性

– 团队合作是项目成功的关键。每个成员都需要确保自己负责的模块功能正确且稳定，模块之间无缝衔接，才能保证整体系统的正常运行。

## 8.3 实践应用的教训

– 我们第一周主要集中在仿真阶段，每个模块仿真通过后就进入下一个模块开发，然而忽略了实际应用的重要性。最终我们发现，仿真通过并不代表上板测试也能顺利通过。

### 8.4 开发与测试过程总结

- 项目开发中，我们总结出有效的工作流程：每个模块开发完成后必须立即进行测试，测试通过后才进入下一个模块的开发。这种方法有效避免了问题累积，提高了开发效率与系统稳定性。

## 9 假如你们团队负责project出题，请给出基于ego1可实现的project的想法和建议

### 9.1 项目名称：基于FPGA的高级数字密码锁系统

### 9.2 项目概述

- 设计一个基于FPGA的多功能数字密码锁系统，不仅要实现基本的密码保护功能，还要加入复杂的加密算法、用户管理功能、联网功能和高级交互界面等。该项目展示FPGA在安全系统设计中的应用，以及复杂硬件功能的整合。

### 9.3 具体功能模块

- FPGA配置:
  - 多重密码加密：采用更复杂的加密算法，如AES（高级加密标准）等，实现多重密码保护
  - 联网功能：支持与外部系统联网，用户可以通过智能手机或计算机远程管理和设置密码
  - 计时器和锁定机制：在错误密码输入后，系统不仅锁定，还可以通过FPGA的计时器限制解锁尝试次数。
- 通用IO接口
  - 按键和触摸屏：除了按键输入外，系统还可以通过触摸屏进行设置和管理。
  - 开关：用来切换锁的状态（锁定/解锁/远程解锁）。
  - LED灯和七段数码管:
    - LED灯：显示密码输入状态（正确或错误）和系统状态（如锁定、远程解锁）
    - 七段数码管：显示当前输入的密码、锁定时间以及远程管理的状态。
  - 
- 音频接口
  - 提供错误或成功的音效反馈，并支持语音提示。
- SRAM和Flash存储
  - 用于存储多个用户的密码信息和设置参数，支持不同用户的独立管理。
  - Flash存储可以保存系统的远程管理和联网记录。
- 蓝牙模块
  - 支持无线远程管理和解锁功能，用户可以通过智能手机应用与系统进行交互。

## 9.4 高级功能

- 多用户管理:
  - 系统支持多个用户，每个用户可以设置独立的密码。
  - 通过FPGA实现不同用户的管理功能，每个用户的密码和权限信息存储在SRAM和Flash中。
- 联网功能
  - 通过蓝牙模块或Wi-Fi模块，用户可以远程管理密码和系统状态。
  - 实现远程解锁和设置功能，用户可以通过手机应用进行操作。
- 错误记录和统计
  - 系统会记录错误输入次数和时间，可以通过FPGA进行分析，帮助提高安全性。
- 锁定时间的动态管理:
  - 错误输入后，不同的锁定时间可以根据错误次数自动调整，增加安全性。

## 9.5 项目目标

- 展示FPGA如何实现多重加密和联网功能，提高密码锁系统的安全性和实用性。
- 实现更复杂的用户交互功能，如远程解锁、密码管理等，增强系统的用户体验。
- 利用FPGA和硬件设计，提升在数字电路和安全系统设计方面的能力。
- 提供一个有实际应用价值的数字密码锁系统，适用于家庭、安全柜等场所的门锁管理。