哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2022 秋季

课程名称：数字逻辑设计（实验）

实验名称：密码锁设计

实验性质：综合设计型

实验学时：6 地点：T2612

学生班级：计算机 4 班

学生学号：210110415

学生姓名：郑瑜杰

评阅教师：

报告成绩：

实验与创新实践教育中心制

2022 年 12 月

注：本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明设计的成果和特色。报告中应该叙述设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

设计的功能描述
概述基本功能、详细描述自行扩展的功能 设计的功能：密码锁 功能 1：未设置密码或者解锁后 按设置密码键可以设置密码 功能 2：已经设置密码后，按下验证密码按键后，可以在小键盘输入 3 位密码进行解锁 功能 3：三次密码输入错误后，将被锁住 功能 4：按复位键可以重新设置密码（即使被锁住） 功能 5：确认输入键，在功能 1 或 2 中输入完三位数字后按下确认键将有效，否则无效（包括输入小于 3 位和输入大于 3 位的情况）

系统功能详细设计

用硬件框图描述系统主要功能及各模块之间的相互关系

要求有信号名、位宽、模块说明，可以参考下面的框图（仅为示例），须有密码处理模块、数码管显示处理模块、按键处理模块，其他模块不限，不可用 vivado 的 RTL 截图。

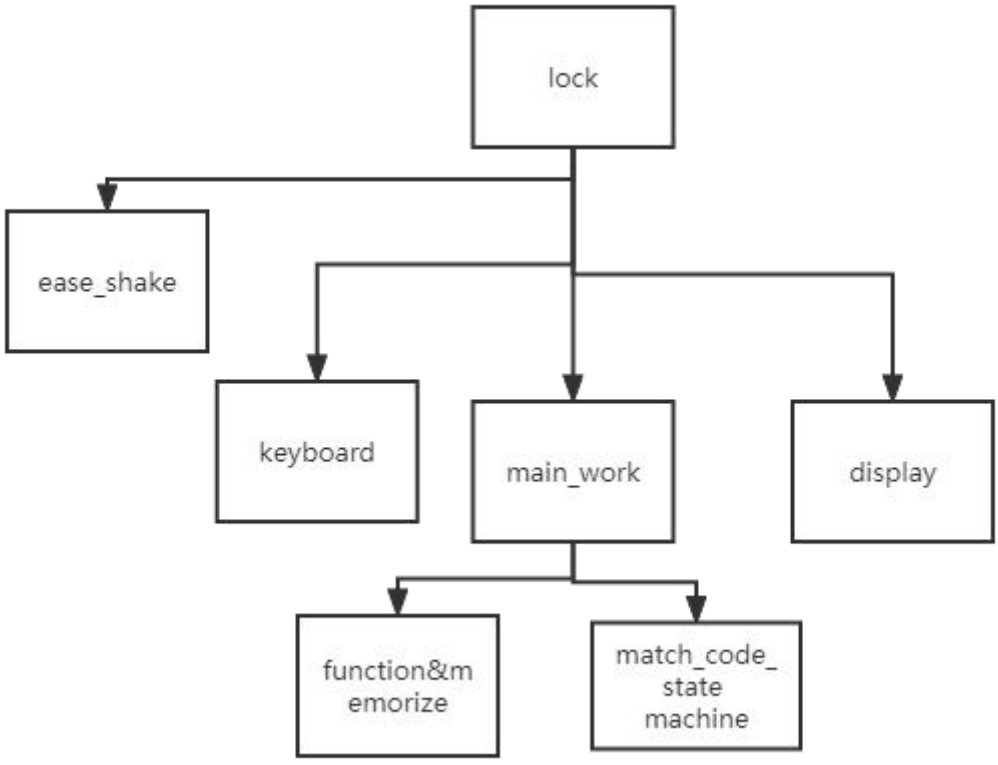
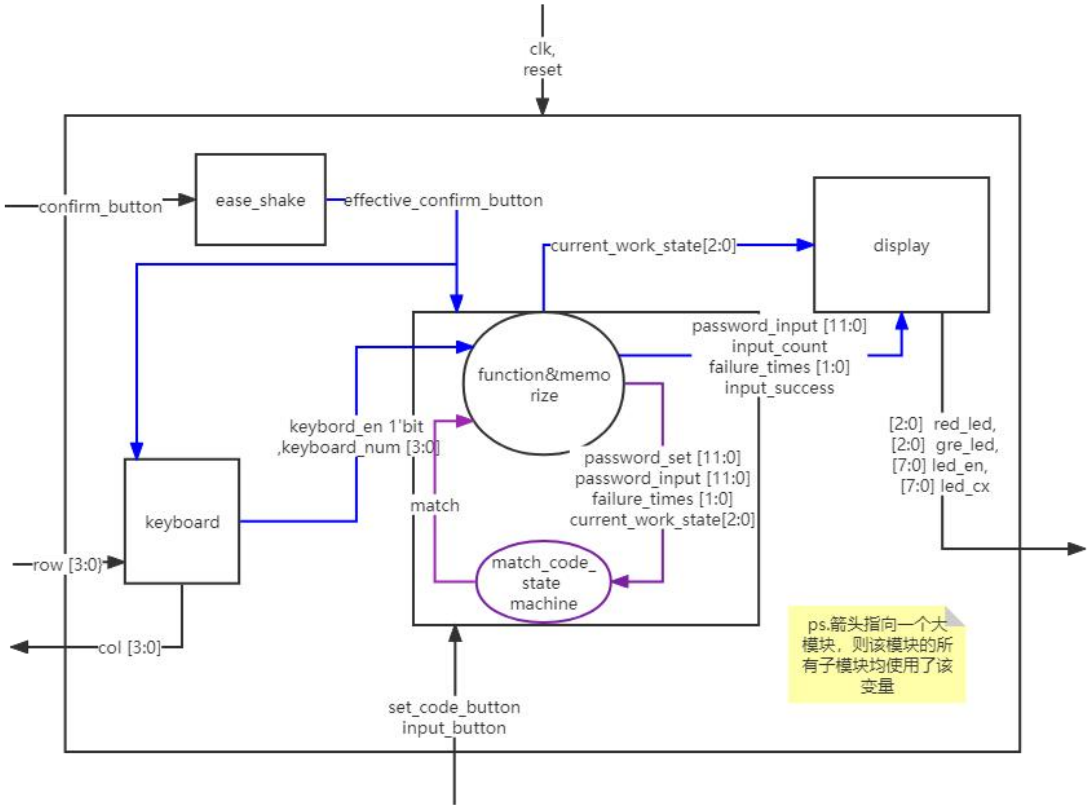


Figure 1

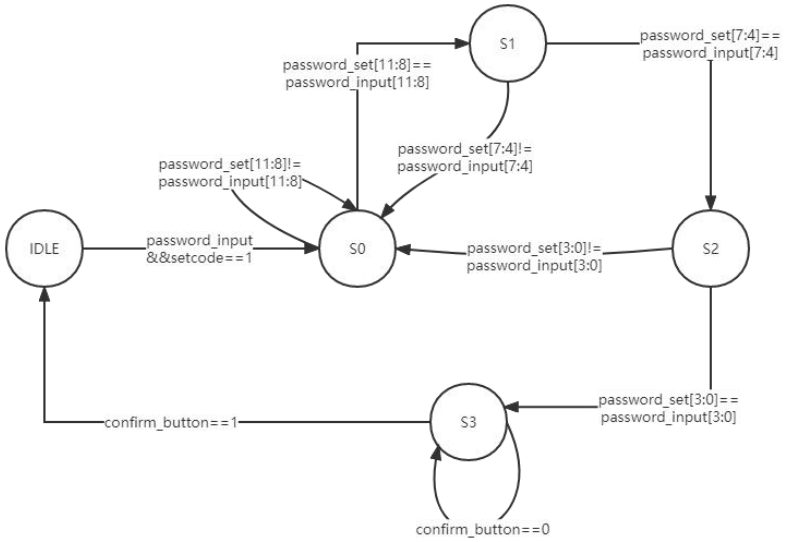
模块划分如上图



状态描述、状态转移图及状态编码，包括功能状态转移图、密码匹配的状态转移图；

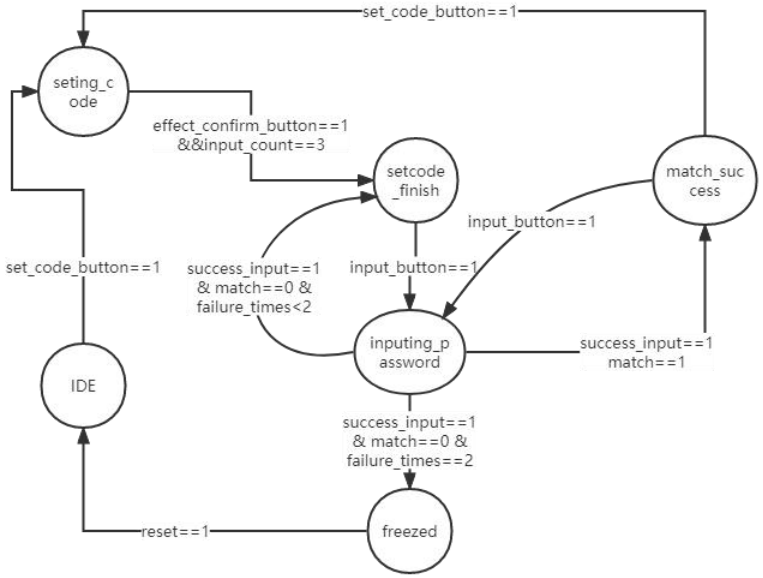
1. 密码匹配

```
//match_code_state machine
parameter IDLE = 3'b000;   //初始状态
parameter s0 =  3'b001;     //开始匹配状态
parameter s1 =  3'b010;     //第一位密码匹配成功
parameter s2 =  3'b011;     //第二位密码匹配成功
parameter s3 = 3'b100;      //第三位密码匹配成功
```



2 功能状态转移

```
//working state machine
parameter IDLE = 3'b000;            //初始状态
parameter seting_code =  3'b001;    //设置密码状态
parameter setcode_finish =  3'b010; //完成密码设置状态
parameter inputing_password =  3'b011;//验证输入密码状态
parameter match_success = 3'b100;       //匹配成功状态
parameter freezed = 3'b101;             //冻结锁住状态
```



各模块描述

包括模块功能，输入、输出端口、变量含义及主要设计代码

Top_lock 模块

功能：连接 ease_shake 消抖模块，连接 keyboard 模块（密码输入输出），function&memorize 模块（核心功能）和 display 显示模块

实现连线

```
module lock#(
    parameter CNT_THRESHOLD=1000000-1
)(
    input         clk,                  //输入：时钟信号
    input         reset,                //输入：复位信号
    input         set_code_button,      //输入：设置密码按键
    input         confirm_button,       //输入：确认输入按键
    input         input_button,         //输入：输入验证密码按键
    input  [3:0]  row,                   //输入：键盘的行信号
    output [3:0]  col,                  //输出：键盘的列信号
    output [2:0]  red_led,              //输出：红色led灯 ：密码匹配错误提示信号
    output [2:0]  gre_led,              //输出：绿色led灯  ：输入有效 及 密码匹配成功信号
    output reg [7:0] led_en,        //输出数码管的使能信号
    output reg [7:0] led_cx         //输出数码管显示数字的信号
);
```

```
  wire  keyboard_en;            //键盘使能
  wire [3:0]keyboard_num;       //键盘输入的数字
keyboard #(CNT_THRESHOLD)u_keyboard(
    .clk(clk),
    .reset(reset),
    .row(row),
    .col(col),
    .keyboard_en(keyboard_en),
    .keyboard_num_buffer(keyboard_num)
);

  wire effect_confirm_button;    //消抖后的上升沿确认键
ease_shake  #(CNT_THRESHOLD)u_ease_shake(
    .clk(clk),
    .reset(reset),
    .button(confirm_button),
    .effect_button(effect_confirm_button)
);
```

```
34  wire [11:0]password_input;       //输入的密码
35  wire   [1:0] failure_times;      //密码匹配错误次数
36  wire         success_input;      //输入是否有效
37  wire          match;             //匹配情况
38  wire  [2:0]input_count;          //输入计数
39  wire [2:0]current_work_state_0;//当前工作状态
40  main_work u_main_work(
41      .clk(clk),
42      .reset(reset),
43      .set_code_button(set_code_button),
44      .input_button(input_button),
45      .confirm_button(effect_confirm_button),
46      .keyboard_en(keyboard_en),
47      .keyboard_num(keyboard_num),
48      .failure_times(failure_times),
49      .success_input(success_input),
50      .password_input(password_input),
51      .match(match),
52      .input_count(input_count),
53      .current_work_state(current_work_state_0)
54  );
```

```
55
56  wire [7:0]led_en_0;
57  wire [7:0]led_cx_0;
58  always@(*)begin
59      led_en<=led_en_0;
60      led_cx<=led_cx_0;
61  end
62  led_display u_led_display(
63      .clk(clk),
64      .reset(reset),
65      .success_input(success_input),
66      .failure_times(failure_times),
67      .red_led(red_led),
68      .gre_led(gre_led),
69      .led_en(led_en_0),
70      .led_cx(led_cx_0),
71      .password_input(password_input),
72      .input_count(input_count),
73      .current_work_state(current_work_state_0)
74  );
75  endmodule
76
```

ease_shake 消抖模块 功能：实现按键消抖，输出按键上升沿

```
module ease_shake#(
    parameter CNT_THRESHOLD=1000000-1
)(
    input         clk,
    input         reset,
    input   wire   button,        //输入所需消抖按键
    output  reg    effect_button     //输出上升沿
);

wire cnt_end;

counter #(CNT_THRESHOLD-2, 24) u_counter(    //计数器
    .clk(clk),
    .reset(reset),
    .cnt_inc(1),
    .cnt_end(cnt_end)
);

reg r_button;
always @(posedge clk ) begin
    if(cnt_end)
        r_button<=button;
end

always @(*) begin
    effect_button <= (~r_button) & button;
end
endmodule

```

keyboard 模块

功能：将开发板的按键信号记录并存储下来，转化为所需的按键使能信号（用于计数）和按键数字信号

```
module keyboard #(
    parameter CNT_THRESHOLD=1000000-1
)(
    input  wire       clk,
    input  wire       reset,
    input  wire [3:0] row,
    output reg  [3:0] col,
    output reg        keyboard_en,    //keyboard的使能信号
    output reg  [3:0] keyboard_num_buffer  //将输入的数字用一个缓冲区存储起来，并作为输出信号
);
reg  [3:0] keyboard_num;//某一使能瞬间键盘的数字
wire cnt_end;
```

```
50  //将按键的数字缓存下来
51  always @(posedge clk, posedge reset) begin
52          if (reset == 1) begin
53          keyboard_num_buffer <= 0;
54          end
55          else begin
56              if(keyboard_en)
57              keyboard_num_buffer <= keyboard_num;
58          end
59  end
```

```
71  //自左向右扫描
72  always @(posedge clk, posedge reset) begin
73      if (reset == 1)           col <= 4'b1111;
74      else if (col ==  4'b1111) col <= 4'b0111;
75      else if (cnt_end)         col <= {col[0], col[3:1]};
76  end
77
```

mainwork 模块:功能：连接 function&memorize 模块（实现功能的状态转移，以及当前信息的存储）和 match_code_state_machine 密码匹配模块

```
module main_work (
    input  wire       clk,
    input  wire       reset,
    input         set_code_button,      //输入：设置密码按键
    input         confirm_button,       //输入：确认输入按键
    input         input_button,         //输入：输入验证密码按键
    input  wire        keyboard_en,   //keyboard的使能信号
    input  wire  [3:0] keyboard_num,  //键盘输入的数字
    output reg   [1:0] failure_times,    //输出：密码匹配失败次数
    output reg         success_input,    //输出：有效输入信号
    output reg      [11:0]password_input,//输出：所输入的密码缓冲区
    output reg          match,           //输出：匹配成功信号
    output reg [2:0]input_count,         //输出：已输入按键的计数
    output reg [2:0]current_work_state      //输出：当前的按键工作状态
);
wire [11:0]password_set;
//连线
wire [11:0]password_input_0;
wire success_input_0;
wire [1:0] failure_times_0;
wire [2:0]input_count_0;
wire match_0;
wire [2:0]current_work_state_0;
always@(*) begin
    password_input <= password_input_0;
    success_input <= success_input_0;
    failure_times <=failure_times_0;
    input_count<=input_count_0;
    match<=match_0;
    current_work_state<=current_work_state_0;
end
memorize u_memorize(
    .clk(clk),
    .reset(reset),
    .set_code_button(set_code_button),
    .confirm_button(confirm_button),
    .input_button(input_button),
    .keyboard_en(keyboard_en),
    .keyboard_num(keyboard_num),
    .password_input(password_input_0),
    .password_set(password_set),
    .match(match_0),
    .success_input(success_input_0),
    .failure_times(failure_times_0),
    .input_count(input_count_0),
    .current_work_state(current_work_state_0)
);
```

```
49   //match code
50   match_code_state_machine u_match_code_state_machine(
51       .clk(clk),
52       .reset(reset),
53       .confirm_button(confirm_button),
54       .input_button(input_button),
55       .set_code_button(set_code_button),
56       .password_input(password_input_0),
57       .password_set(password_set),
58       .match(match_0),
59       .failure_times(failure_times_0)
60   );
61   endmodule
```

function&memorize 模块

实现功能的状态转移，

以及密码设置，密码输入，匹配失败次数，输入计数，有效输入等信号的存储 并将该信号作为输出

```
module memorize(
    input  wire       clk,
    input  wire       reset,
    input  wire      set_code_button,   //输入：设置密码按键
    input  wire       confirm_button,       //输入：确认输入按键
    input  wire       input_button,         //输入：输入验证密码按键
    input wire        keyboard_en,   //keyboard的使能信号
    input wire  [3:0] keyboard_num,  //键盘输入的数字
    output reg [11:0]password_set,   //输出：所设置的密码=
    output reg [11:0]password_input,  //输出：所输入的密码缓冲区
    output reg [1:0]failure_times,   //输出：密码匹配失败次数
    output reg        success_input, //输出：有效输入信号
    output reg [2:0]input_count,    //输出：已输入按键的计数
    output reg [2:0] current_work_state, //输出：当前的按键工作状态
    input wire        match        //输入：匹配成功信号
);
//working state machine
parameter IDLE = 3'b000;            //初始状态
parameter seting_code =  3'b001;    //设置密码状态
parameter setcode_finish =  3'b010; //完成密码设置状态
parameter inputing_password =  3'b011;//验证输入密码状态
parameter match_success = 3'b100;       //匹配成功状态
parameter freezed = 3'b101;             //冻结锁住状态
reg [2:0] next_work_state;
always @(posedge clk or posedge reset) begin
    if(reset) current_work_state <=IDLE;
    else        current_work_state <= next_work_state;
end
//功能状态转移
always @(*) begin
    if(reset) next_work_state <=IDLE;
    case (current_work_state)
        IDLE : if(set_code_button) next_work_state <= seting_code;
            else next_work_state<=next_work_state;
        seting_code : if(confirm_button&input_count==3) next_work_state <= setcode_finish;
           else next_work_state<=next_work_state;
        setcode_finish : if(input_button) next_work_state <= inputing_password;
           else next_work_state<=next_work_state;
        inputing_password : if(match) next_work_state <= match_success;
           else if(success_input&failure_times==2) next_work_state <= freezed;
           else if(success_input&failure_times<2)next_work_state <= setcode_finish;
           else next_work_state<=next_work_state;
        match_success : if(set_code_button) next_work_state<=seting_code;
           else if(input_button) next_work_state<=inputing_password;
           else next_work_state<=next_work_state;
        freezed : if(reset) next_work_state<=IDLE;
           else next_work_state<=next_work_state;
        default:next_work_state <= IDLE;
    endcase
```

```
    endcase
end
//密码错误失败计数
always @(posedge clk or posedge reset) begin
    if(reset) failure_times<=0;
    else if(current_work_state==inputing_password &success_input )begin
        if(match) failure_times<=0;
        else failure_times<=failure_times + 2'b01;
    end
end
reg setcode;//是否设置好密码了
//deal with input and memorize
always @(posedge clk or posedge reset) begin
    if(reset) setcode<=1'b0;
    else if(current_work_state==seting_code)
        begin
            if((~setcode | match) & confirm_button)begin
            setcode <=1'b1;
            end
        end
end
//输入的计数
always @ (posedge clk or posedge reset) begin
    if(reset)   input_count<=0;
    else begin
        if(input_button|confirm_button|set_code_button) input_count<=0;
        else if(keyboard_en)begin
            input_count <=input_count+ 1;
        end
    end
end
```

```
//密码的输入
always @ (posedge clk or posedge reset) begin
    if(reset) password_input<=0;
    else begin
    if(set_code_button|input_button) password_input<=0; //清除上一次输入
    else begin
            case(input_count)
            3'b001 : password_input[11:8]<= keyboard_num;
            3'b010 : password_input[7:4] <= keyboard_num;
            3'b011 : password_input[3:0] <= keyboard_num;
            default: password_input <= password_input;
            endcase
          end
    end
end
```

```
95   //密码设置
96   always @ (posedge clk or posedge reset) begin
97       if(reset)   password_set<=0;
98       else begin
99           if(current_work_state==seting_code| current_work_state==match_success)
100          begin
101              if((~setcode | match) & confirm_button)begin
102              password_set <=password_input;
103              end
104          end
105      end
106  end
107
108  //输入有效信号
109  always @(posedge clk or posedge reset) begin
110      if(reset) success_input<=0 ;
111      else if (confirm_button )begin
112          if(input_count ==3'b011 &(failure_times!=2'b11))
113          success_input<=1'b1;
114          else success_input<=1'b0;
115      end
116  end
117
118  endmodule
```

match_code_machine

功能：实现密码的状态匹配，并且输出匹配结果

```
module match_code_state_machine(
    input         clk,
    input         reset,
    input         confirm_button,    //输入：确认输入按键
    input         set_code_button, //输入：设置密码按键
    input         input_button,      //输入：输入验证密码按键
    input   [11:0]password_set,    //输入：所设置的密码
    input  [11:0]password_input,    //输入：所输入的密码
    input  [1:0]failure_times,       //输出：密码匹配失败次数
    output reg     match          //输出：匹配成功信号
);
//match_code_state machine
parameter IDLE = 3'b000;    //初始状态
parameter s0 =  3'b001;      //开始匹配状态
parameter s1 =  3'b010;      //第一位密码匹配成功
parameter s2 =  3'b011;      //第二位密码匹配成功
parameter s3 = 3'b100;       //第三位密码匹配成功
reg [2:0] current_state;
reg [2:0] next_state;
reg setcode;

always @(posedge clk or posedge reset) begin
    if(reset) setcode <=0;
    else begin
        if(set_code_button) setcode <=1;
        // else if(confirm_button) setcode <=0;
    end
end
//state machine

always @(posedge clk or posedge reset) begin
    if(reset|input_button) current_state <=IDLE;
    else        current_state <= next_state;
end
```

```
always @(posedge clk or posedge reset) begin
    if(reset|input_button) current_state <=IDLE;
    else        current_state <= next_state;
end

always @(*) begin
    if(reset) next_state <=IDLE;
    else if(failure_times!=2'd3)begin
    case (current_state)
         IDLE : if(input_button) next_state <= s0;
                else next_state<=next_state;
         s0 : if(password_input[11:8]==password_set[11:8]) next_state <= s1;
            else next_state<=s0;
         s1 : if(password_input[7:4]==password_set[7:4]) next_state <= s2;
            else next_state<=s0;
         s2 : if(password_input[3:0]==password_set[3:0]) next_state <= s3;
            else next_state<=s0;
         s3 : if(confirm_button) next_state<=IDLE;
            else next_state<=s3;
         default:next_state <= IDLE;
    endcase
    end
end

always @(posedge clk or posedge reset) begin
    if(reset | input_button) match <= 0;
    else begin
        if(~setcode) match <=0;
        else if(current_state==s3&confirm_button)
            match <= 1'b1;
    end
end

endmodule
```

Led_display 模块

功能：显示错误匹配 led 灯，有效输入和成功匹配 led 灯

显示 数码管（锁住全 F，输入时依次显示输入的数字，不输入时显示全 0）

```
module led_display (
    input wire clk,
    input wire reset,
    input wire  [1:0] failure_times,   //输入：密码匹配失败次数
    input wire success_input,          //输入：有效输入信号
    input wire [11:0]password_input,       //输入：所输入的密码
    input wire [2:0]input_count,            //输入：已输入按键的计数
    input wire [2:0]current_work_state,     //输入：当前的按键工作状态
    output reg [2:0]  red_led,          //输出：红色led灯 ：密码匹配错误提示信号
    output reg [2:0]  gre_led,          //输出：绿色led灯  ：输入有效 及 密码匹配成功信号
    output reg [7:0] led_en,            //输出数码管的使能信号
    output reg [7:0] led_cx             //输出数码管显示数字的信号
);
```

```
  parameter IDLE = 3'b000;
  parameter seting_code =  3'b001;
  parameter setcode_finish =  3'b010;
  parameter inputing_password =  3'b011;
  parameter match_success = 3'b100;
  parameter freezed = 3'b101;

  //count 2ms refresh moudule
  reg[3:0]led_output;
  reg [7:0]switch_led;
  reg [19:0] refresh_cnt;
  wire refresh = (refresh_cnt == 20'd199_999);
  // wire refresh = (refresh_cnt == 20'd2);

always @(posedge clk or posedge reset) begin
      if(reset)    switch_led <=8'b1111_1110;
      else if(refresh)    switch_led <= {switch_led[0],switch_led[7:1]};
  end

always @(posedge clk or posedge reset) begin
      if(reset |refresh) refresh_cnt<=0;
      else  refresh_cnt<=refresh_cnt+ 1;
  end
```

```
38  //display red and green led
39  always @(*) begin
40      case(failure_times)
41          2'b00:red_led<=3'b000;
42          2'b01:red_led<=3'b100;
43          2'b10:red_led<=3'b110;
44          2'b11:red_led<=3'b111;
45          default:red_led<=0;
46      endcase
47  end
48
49  always @(*) begin
50      if(current_work_state==match_success) gre_led <= 3'b111;
51      else if(success_input) gre_led<=3'b001;
52      else gre_led<=3'b000;
53  end
54
```

```
55  //display digital led
56  always @ (*) begin
57      if(current_work_state==freezed)
58          led_output<=4'hf;
59      else if(~((current_work_state==seting_code)|(current_work_state==inputing_password)))
60          led_output <=4'h0;
61      else begin
62      if(input_count>display_place_num)begin
63          case(display_place_num)
64              2'd0:led_output<=password_input[11:8];
65              2'd1:led_output<=password_input[7:4];
66              2'd2:led_output<=password_input[3:0];
67              default led_output<=0;
68          endcase
69           end
70       else  led_output<=0;
71      end
72  end
73
74  always @(*) begin
75      led_en <= switch_led;
76  end
77
78  reg [2:0]display_place_num;
79  always @(*) begin
80      case(switch_led)
81                  8'b1111_1110:display_place_num <=3'd7;
82                  8'b1111_1101:display_place_num <=3'd6;
83                  8'b1111_1011:display_place_num <=3'd5;
84                  8'b1111_0111:display_place_num <=3'd4;
85                  8'b1110_1111:display_place_num <=3'd3;
86                  8'b1101_1111:display_place_num <=3'd2;
87                  8'b1011_1111:display_place_num <=3'd1;
88                  8'b0111_1111:display_place_num <=3'd0;
89      endcase
90  end
```

```
always @(*) begin
        if((current_work_state==seting_code)|(current_work_state==inputing_password))
        begin//正在输入时
            if(input_count>display_place_num)begin  //显示已输入数字
            case(led_output)
            4'h0:led_cx <= 8'b0000_0011;
            4'h1:led_cx <= 8'b1001_1111;
            4'h2:led_cx <= 8'b0010_0101;
            4'h3:led_cx <= 8'b0000_1101;
            4'h4:led_cx <= 8'b1001_1001;
            4'h5:led_cx <= 8'b0100_1001;
            4'h6:led_cx <= 8'b0100_0001;
            4'h7:led_cx <= 8'b0001_1111;
            4'h8:led_cx <= 8'b0000_0001;
            4'h9:led_cx <= 8'b0001_1001;
            4'ha:led_cx <= 8'b0001_0001;
            4'hb:led_cx <= 8'b1100_0001;
            4'hc:led_cx <= 8'b1110_0101;
            4'hd:led_cx <= 8'b1000_0101;
            4'he:led_cx <= 8'b0110_0001;
            4'hf:led_cx <= 8'b0111_0001;
            endcase
            end
            else   led_cx <= 8'b1111_1111;//不显示其他灯
    end
```

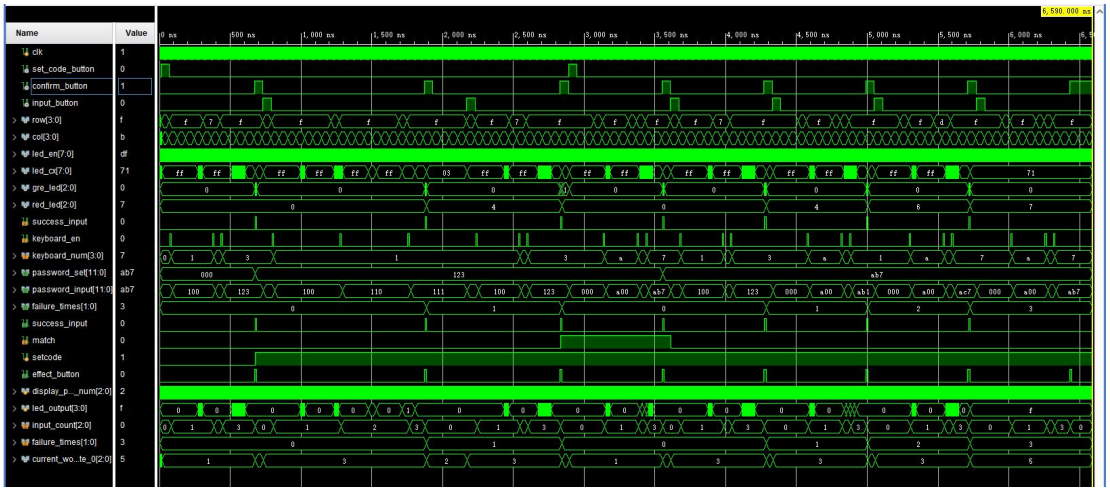
```
            end
            else begin  //非输入数据状态，所有灯都亮
                case(led_output)
                4'h0:led_cx <= 8'b0000_0011;
                4'h1:led_cx <= 8'b1001_1111;
                4'h2:led_cx <= 8'b0010_0101;
                4'h3:led_cx <= 8'b0000_1101;
                4'h4:led_cx <= 8'b1001_1001;
                4'h5:led_cx <= 8'b0100_1001;
                4'h6:led_cx <= 8'b0100_0001;
                4'h7:led_cx <= 8'b0001_1111;
                4'h8:led_cx <= 8'b0000_0001;
                4'h9:led_cx <= 8'b0001_1001;
                4'ha:led_cx <= 8'b0001_0001;
                4'hb:led_cx <= 8'b1100_0001;
                4'hc:led_cx <= 8'b1110_0101;
                4'hd:led_cx <= 8'b1000_0101;
                4'he:led_cx <= 8'b0110_0001;
                4'hf:led_cx <= 8'b0111_0001;
                endcase
            end
    end
    endmodule
```

硬件框图有的模块都需体现

调试报告

仿真波形截图及仿真分析

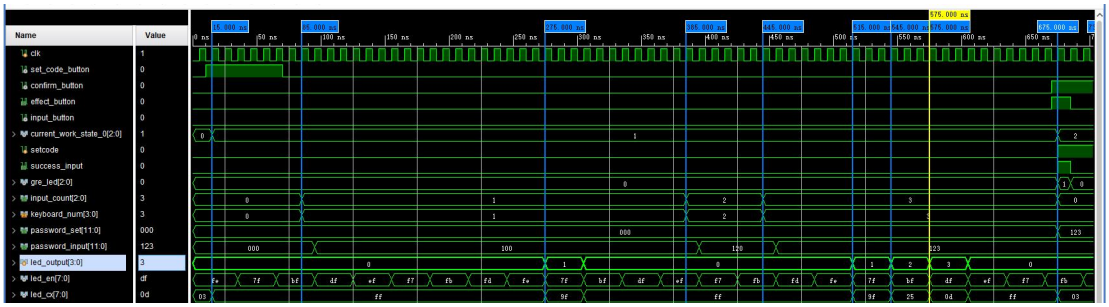
仿真总波形



Current_work_state 表示当前工作处于的状态，0 表示初始状态，1 表示正在设置密码，2 表示设置完密码，3 表示正在验证密码，4 表示密码匹配成功。5 表示锁定状态

Set_code_button 设置密码 confirmbutton 确认输入，inputbutton 验证密码 led_en 显示灯使能，led_cx 显示数字,red_led 错误输入。Gre_led 有效及匹配成功结果，success_input 有效输入。Setcode 表示已设置密码，password_set 设置的密码 password_input 输入的密码

1.密码设定 （未设置密码）（工作状态分析+密码+显示分析）



15ns 时刻 clk 上升沿

状态转变： set_code_button 为 1。Current_work_state 由 0 变为 1，进入正在设置密码状态

密码部分：无

显示部分：led_cx 由 03 变为 ff 即 显示 0 变为不显示。

85ns clk 上升沿

状态转变:。Current_work_state 为 1，正在设置密码状态

密码部分：输入数字 keyboard_num 变为 1。Input_count 变为 1，95ns 输入缓存 password_input 由 000 变为 100

显示部分：85ns led_cx 由 03 变为 ff 即 显示 0 变为不显示。在 275ns 时刻，选择第一个数码管,led_out 变为 1。Led_cx 由 ff 不显示变为 9f 显示数字 1

385ns 时刻

密码部分：输入数字 keyboard_num 变为 2。Input_count 计数变为 2，在 395ns 时刻密码缓存 password_input 100 变为 120，

445ns 时刻

密码部分：输入数字 keyboard_num 变为 3。Input_count 计数变为 3

在 395ns 时刻密码缓存 password_input 120 变为 123，

515,545,575ns 时刻。Led_cx 依次变为 9f,25,0d.Led_out 依次变为 1,2,3，依次显示已经输入的密码。而当 605ns 时刻 led_cx 变为 ff 不显示其他位置的密码。

675ns 时刻 clk 上升沿，confirm_button 确认按钮变为 1，

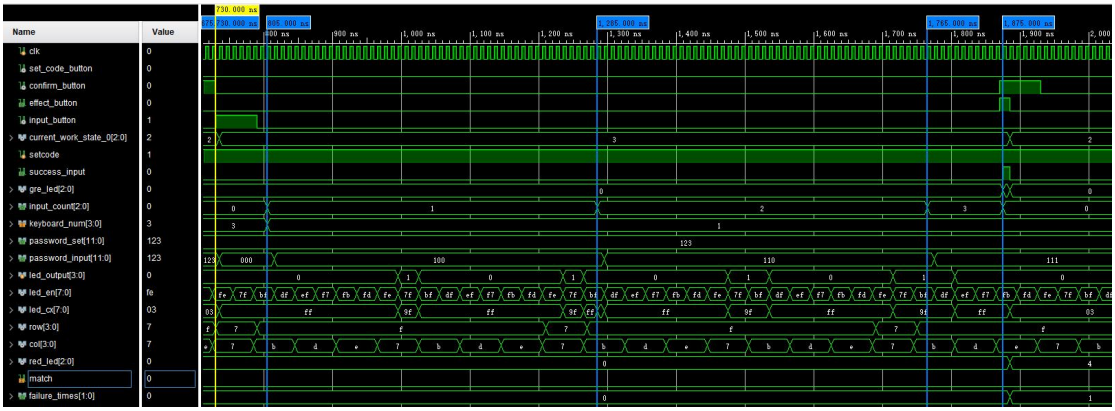
状态转变:。Current_work_state 由 1 变为 2，从正在设置密码状态转变为完成密码设置状态。

密码部分:setcode 由 0 变为 1。 Password_set 由 000 变为 123。

输入计数 input_count 由 3 变回 0

显示部分：success_input 变为 1 。gre_led 由 0 变为 1

2.密码匹配失败（工作状态分析+密码+显示分析）



730ns 时刻 input_button 验证密码按钮为 1，

735ns 上升沿

状态：current_work_stare 由 2 变为 3，由已设置密码状态进入验证密码状态

密码部分：password_input 由 123 变为 000（清除之前的输入）

805,1285,1765ns 时刻均输入 1，input_count 计数分别变为 1,2,3

815,1295,1775ns 时刻 password_input 分别变为 100,110,111;

显示部分：995ns，led_en 为 7f， led_cx 为 9f 显示第一个数字 1

1475 和 1505ns ，led_en 为 7f,bf .led_cx 为 9f 。依次显示 11

1715，1745,1775 ns led_en 为 7f,bf df。.led_cx 为 9f 。依次显示 111，而后面显示 0

1875ns 时刻.确认输入按键 effect_button 为 1

而后 1885ns 时刻。状态：current_work_stare 由 3 变为 2，由验证密码状态进入已设置密码状态

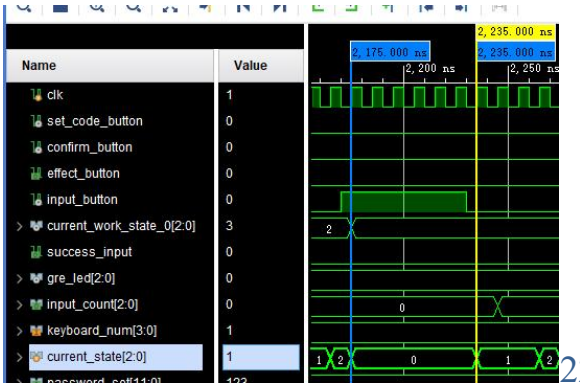
密码部分： failure_times 由 0 变为 1，计数错误匹配次数。

显示部分：gre_led 由 000 变为 100(二进制)显示最左侧 1 个红灯

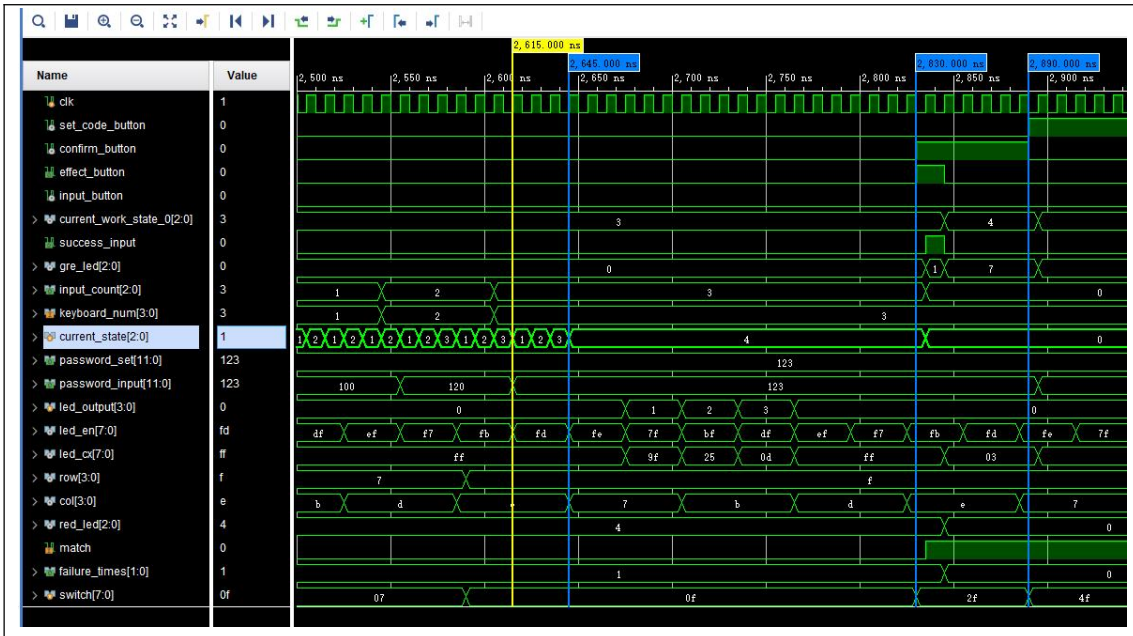
Led_cx 由 ff 变为 03. 即只显示输入的密码，而其他部分不显示 变为所有数字显示 0

3.密码匹配成功（密码匹配状态分析+工作状态分析+显示分析）

密码匹配 current_state 0 表示初始状态，1 表示开始匹配的初始状态 2 表示第一位匹配成功。3 表示前两位匹配成功。4 表示前三位匹配成功。



2235ns 时刻，input_button 已变回 0，current_state 由 0 变为 1



此时设置密码 password_set 为 123

当 3 位密码 password_set 123 均输入完成后。2615,2625,2635,2645ns current_state 依次为 1,2,3,4 完成匹配。

2625ns 时刻 current_state 为 1，进行第一位密码匹配，由于 1=1 ，故 current_state 变为 2

2635ns 时刻 current_state 为 2，进行第二位密码匹配，由于 2=2 ，故 current_state 变为 3

2645ns 时刻 current_state 为 3，进行第三位密码匹配，由于 3=3 ，故 current_state 变为 4

之后一直保持 4

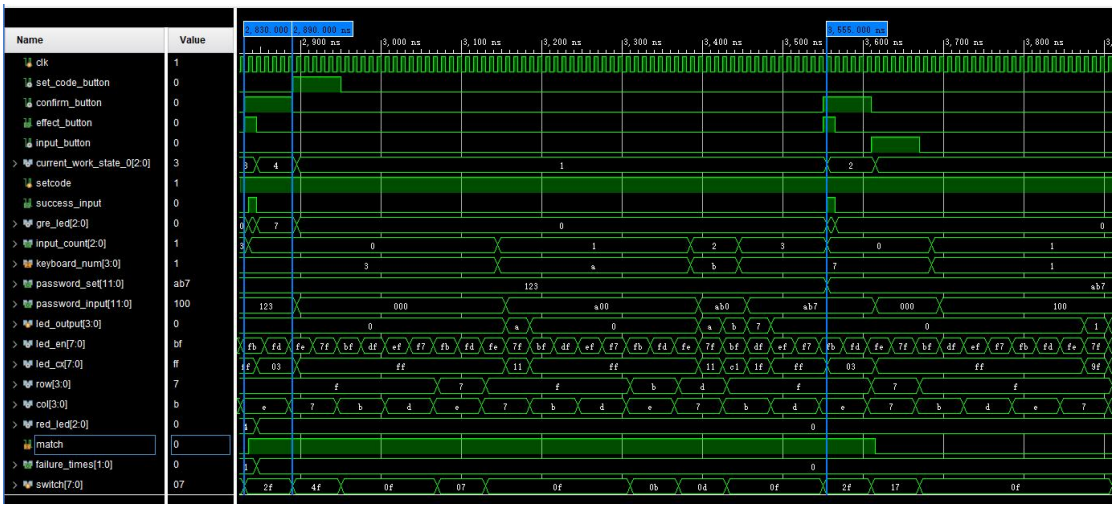
在 2835ns 时刻确认输入完成，current_state 由 4 变回 0

2830ns 时刻 confirm_button 为 1；

2835 时刻 ，success_input 有效输入变为 1.match 变为 1，匹配成功

在 2845ns 时刻。状态 current_work_state 由 3 变为 4.进入匹配成功状态。匹配成功绿灯 gre_led 变为 111。失败次数 Failure_times 由 1 变为 0。失败红灯 red_led 由 100 变为 000.

4.重新设置密码（状态分析 + 键盘实现数字输入分析）



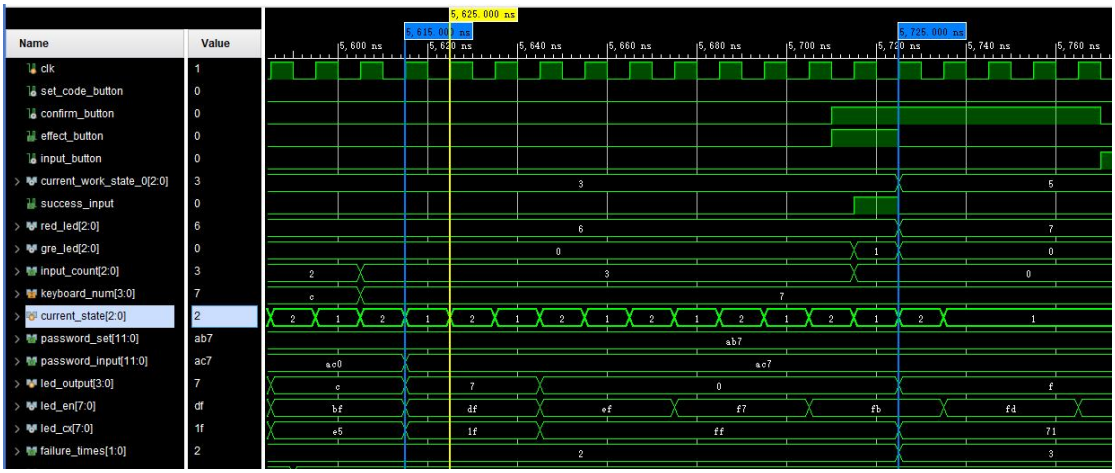
2895ns 时刻 set_code_button 为 1.

状态 current_work_state 由 4 变为 1 即由匹配成功状态变为设置密码状态。

3555ns 时刻 。confirm_button 为 1. 状态 current_work_state 由 1 变为 2 即由正在设置密码状态变为已设置密码状态。

Password_set 由 123 变为 ab7

5.密码匹配失败次数由 2 变为 3（密码匹配状态机分析）



工作状态分析：

5715ns 时刻 success_input 为 1。 Failure_times 由 2 变为 3

Current_work_state 由 3 变为 5（由正在验证密码 进入 密码冻结状态）

密码匹配状态分析：

5605ns 时刻 count 变为 3，密码输入完成。

此时密码 password_set 为 ab7

输入密码 password_set 为 ac7

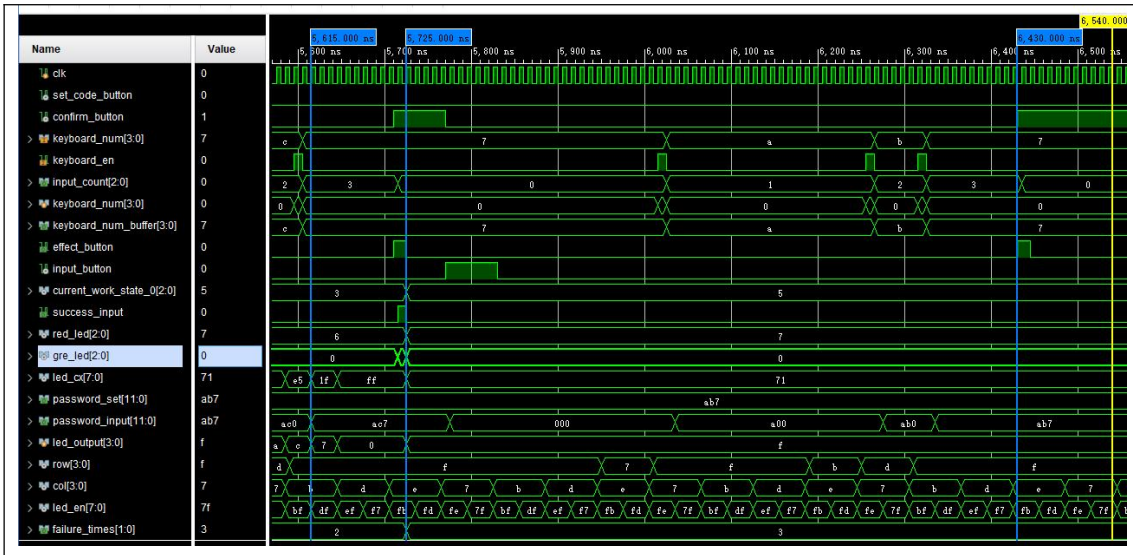
5625ns 时刻进行第一位密码匹配： a = a 故 current_state 由 1 变为 2（一位匹配成功）

2635ns 时刻 进行第二位密码匹配 由于 b!=c 故故 current_state 由 2 变为 1（回到无密码匹配成功状态）

2645ns 时刻，由于 current_state 为 1，故不进行第三位密码匹配，转而进行第一位密码匹配。进行第二位密码匹配故 current_state 由 1 变为 2（一位匹配成功）

匹配失败

6.锁住状态 输入正确密码却无效



如图。6430ns 时刻 confirm_button 输入为 1.

此时 password_set == password_input == ab7

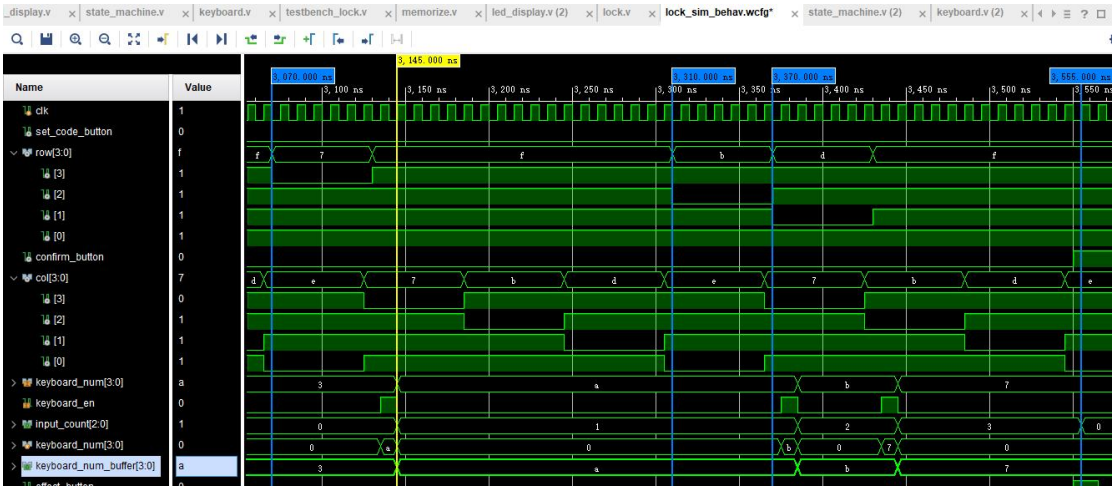
但此时 led_output 一直为 f ， failure_times 依旧为 3

Led_cx 一直为 71 （显示 f）

gre_led 一直为 0 （无有效输入的显示）

red_led 一直为 111 显示三次错误次数

7.键盘输入分析



当 row 为 f 时刻，键盘无输入

键盘输出 col 一直在刷新，低电平有效，每次仅一位为 0

Row[3:0]分别对应键盘第 1-4 行 。 col[3:0]分别对应 1-4 列

3070ns 时刻 row[3]为 0，col[0]为 0。第一行第四列对应输入键盘数字 A。此时 keybard_num 为 a，该局部变量数据存储在 keyboard_num_buffer 中。

设计过程中遇到的问题及解决方法

第一个最大问题：**仿真失败。**

A.代码无明显语法错误，却无法仿真

解决方案： 翻译报错信息，上网搜原因等，不断解决问题。最后排查出某一个变量在两个模块都做 output reg 寄存器时不能直接相连，需要创造 wire 线进行连接。

B.仿真测试成功，输入无问题，绝大多数内部中间变量不怎么变化。

解决方案：从输入开始检查看是否有问题，顺藤摸瓜，发现测试文件 testbench 的时钟和模块时钟不一致，在测试文件实例化执行文件时，需要参数化用同一时间。

第二部分问题：仿真以及上版排查问题

解决方案：仿真检查中间变量的信号传递，上版遇到某些功能未完成时完善代码并利用仿真去排查对应问题

1.数据处理模块

问题 a:密码设置.设置密码时发现密码设置一直为 000

解决方案：通过仿真的中间变量排查密码设置过程的信号传递，发现密码输入后下一时钟周期 input 立即回归到 0，而需要等待确认设置密码直接才能设置密码（密码输入与设置不在相邻时钟周期）。因此需要添加密码输入缓冲区。

问题 b:密码所败三次后输入正确密码可解锁

解决方案：检查开始密码匹配的条件，发现密码匹配模块也需要添加

功能状态。并且排查匹配成功后执行的语句需要添加未锁住的条件。

2 显示模块

问题 a:仿真时，密码输入错误，红灯没有实现预期的亮错误次数。

解决方法：添加显示红灯部分的中间变量，发现失败次数 mistake_times 无数据，但执行模块的 mistake_times 有数据且正确，从而排查出是显示部分的接口未添加 mistake_times.

问题 b:仿真时，测试已输入的密码是否显示成功，发现 led 全部不显示。

解决方案：直接排查输入密码部分的显示条件是否正确。

问题 c: 上版时，输入密码错误按确认键，直接锁住亮 3 个红灯。

解决方法：添加消抖模块，并且检查输入模块：添加正确输入变量，一定要在合适的工作状态下达到 3 次输入才能实现有效密码输入，才能做状态的转移。

第三部分问题：综合失败。

解决方案：根据报错信息依个排查直至无错误。检查各模块接口是否正确有无缺漏（因为不同模块之间所需传递的变量信息一开始并未考虑周全）

课程设计总结
包括设计的总结和还需改进的内容以及收获 设计的总结：本次设计实现了一个密码锁 在功能上：该密码锁实现了可以按键输入功能，设置密码功能，解锁后重置密码功能，连续失败 3 次将冻结锁住功能，显示功能。 在实现上：该密码锁的实现主要分为了按键输入，密码处理和数码管数据显示模块。密码处理又分为了功能状态模块、密码输入存储模块和密码匹配处理模块。 在设计方法上：本次设计采用自顶向下的设计方法设计不同的模块，每个模块再细分需要实现什么功能。然后再逐个模块完成设计。在实验中我体会到这种方法的好处：结构清晰，层次分明，容易排查 bug。在本次实验设计中，我锻炼了自顶向下的设计能力，锻炼了工程实践能力。 还需改进的内容以及收获： 1.实验过程中有很多变量的位宽是相同的，采用位宽参数化可能可以提高代码实现的效率。 2.原本想设计一个可以设置任意位数密码的密码锁，但是由于最初没有使用参数化位宽设计只按照 3 位密码锁所需的最低位宽要求设计，许多变量一个一个去调整位宽显得有点麻烦，因此最后没有实现该功能。但事实上本功能近在咫尺。 3.在 debug 中体会到自顶向下设计的一个非常重要的好处就是可以快

速定位到底是那一个模块的那一部分功能出了问题，debug 效率非常高。

4.我感悟到，最初的设计可能并不完善，但最好要把大体框架搭好。在不断的实践过程中提高对实验的认识，然后再去调整框架，补充细节和完善功能。设计和实践应该需要紧密联系。

5.在实验遇到问题时，跟同学交流讨论一下可以互帮互助，提高解决问题的效率，开拓思路。

6.一边进行代码实现还需一边在关键位置写注释，以免后期遇到问题需要修改时忘记了最初这么实现的原因，从而避免重复劳动。