# 实验四 综合电路设计实验报告


| | |
|---|---|
| **组长**：张开元 | **学号**：1120202079 |
| **班级**：07112004 | **手机**：13770509157 |
| **组员**：管德伟 | **学号**：1120202074 |
| **班级**：07112004 | **手机**：18555851991 |
| **组员**：刘博文 | **学号**：1120201883 |
| **班级**：07112004 | **手机**：18510892300 |


注：黑色字体内容不能改动，蓝色字体内容（为示例或说明）需删除和修改。

## 1. 实验题目

- 短跑计时器设计与实现（难度系数：0.9）

短跑计时器描述如下：

- 短跑计时器显示分、秒、毫秒；
- “毫秒”用两位数码管显示：百位、十位；
- “秒”用两位数码管显示：十位、个位；
- “分”用一位 LED 灯显示，LED 灯“亮”为 1 分；
- 最大计时为 1 分 59 秒 99，超限值时应可视或可闻报警；
- 三个按键开关：计时开始/继续（A）、计时停止/暂停（B）、复位/清零（C）。

## 2. 电路设计

下面将从模块的角度分析此电路设计。

第一个模块：key_rebounce，起到的作用是按键消抖。输入为时钟，复位信号和按键信号；输出是消抖过的按键信号。

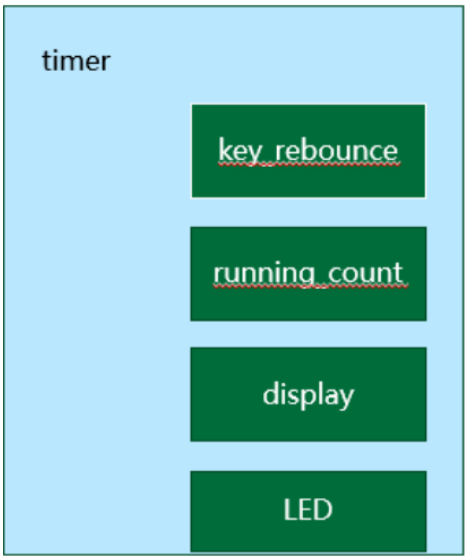
第二个模块：running_count，也是此电路的核心模块，起到计时的作用。输

入为时钟、复位信号和多种按键信号；输出为分、秒、毫秒时间和是否超时。

第三个模块：display，作用是把分、秒与毫秒转换成能在 LED 灯上显示的信号。输入为时钟、复位信号和分、秒、毫秒的时间信息；输出为数码管选择信号和 LED 控制信号。

第四个模块：LED，作用是控制两个 LED 灯的亮灭。输入为时钟、复位信号和超时信号，输出为 LED 是否亮起。

第五个模块，也是总模块：timer，作用是将上面的四个模块合成一个总的电路。




## 3. 电路实现

### running_timer 模块：

```
module running_timer(
    input clk,
    input rst_n,
    input start_continue,
    input pause_stop,
    output wire[3:0] select_digit,
    output wire[7:0] show_led,
    output wire l1,
```

```
        output wire l2
);

//给 start_continue 消抖
wire start_continue_out;

key_rebounce s_key_rebounce(
        .clk            (clk),
        .rst_n          (rst_n),
        .key_in         (start_continue),
        .key_out        (start_continue_out)
);

//给 pause_stop 消抖
wire pause_stop_out;

key_rebounce p_key_rebounce(
        .clk            (clk),
        .rst_n          (rst_n),
        .key_in         (pause_stop),
        .key_out        (pause_stop_out)
);



wire    min;
wire [7:0] sec;
wire [7:0] milisec;
wire timeout;

running_count u_running_count(
        .clk                    (clk),
        .rst_n                  (rst_n),
        .start_continue         (start_continue_out),
        .pause_stop             (pause_stop_out),
        .min                    (min),
        .sec                    (sec),
        .milisec                (milisec),
        .timeout                (timeout)
);



//数字显示模块
```

```
display u_display(
        .clk                    (clk),
        .sec                    (sec),
        .milisec                (milisec),
        .select_digit           (select_digit),
        .show_led               (show_led)
);


//LED 显示模块
LED u_LED(
        .clk            (clk),
        .rst_n          (rst_n),
        .min            (min),
        .timeout        (timeout),
        .l1             (l1),
        .l2             (l2)
);

Endmodule
```

## running_count 模块：

```
module running_count(
        input clk,                      //时钟
        input rst_n,                    //复位
        input start_continue,           //开始或继续输入
        input pause_stop,               //暂停或停止输入
        output reg min,                 //记录分钟仅需 1 位
        output reg [7:0] sec,           //秒钟，BCD 码，高 4 位是十位，低四位是个位
        output reg [7:0] milisec, //毫秒
        output reg timeout              //超时标记
);


parameter STATE_IDLE        =   "idle" ;
parameter STATE_COUNTING =    "counting" ;
parameter STATE_PAUSE       =   "pause" ;
parameter STATE_TIMEOUT    =   "timeout" ;

reg [100:0] current_state;
reg [100:0] next_state;
```

```
always @ (posedge clk or negedge rst_n) begin
    //复位则回到默认态
    if(~rst_n) begin
        current_state<=STATE_IDLE;
        timeout <= 1'b0;
    end
    //没复位则进入下一状态
    else begin
        current_state <= next_state;
    end
end

//状态机描述
always @ (*) begin
    case(current_state)

    STATE_IDLE : begin
        //默认状态转移到计时状态
        if(start_continue == 1'b1) begin
            next_state = STATE_COUNTING;
        end
        //默认状态不变
        else begin
            next_state = STATE_IDLE;
        end
    end

    STATE_COUNTING : begin
        //计时状态转移到暂停状态
        if(pause_stop==1'b1) begin
            next_state = STATE_PAUSE;
        end
        //计时状态转移到超时状态
        else if (min == 1'b1 && sec == 'h59 && milisec == 'h99 && flag ) begin
            next_state = STATE_TIMEOUT;
        end
        //继续计时
        else begin
            next_state = STATE_COUNTING;
        end
    end

    STATE_PAUSE : begin
        //收到暂停信号，保持暂停
```

```
            if(pause_stop == 1'b1)
                next_state = STATE_PAUSE;
            //收到继续信号，暂停状态转移到计时状态
            else if(start_continue==1'b1)
                next_state = STATE_COUNTING;
            //保持暂停
            else
                next_state = STATE_PAUSE;
        end

        STATE_TIMEOUT : begin
            //复位到默认态
            if(rst_n==1'b0) begin
                next_state = STATE_IDLE;
            end
            //保持超时态
            else begin
                next_state = STATE_TIMEOUT;
            end
        end
        //给冗余态用
        default:
            next_state <= STATE_IDLE;
        endcase
end

//10ms，根据时钟频率决定
parameter MILISEC_10 = 4;
//用于记录是否满足 10ms 进 1
reg [31:0] count;

always @ (posedge clk or negedge rst_n) begin
        if(~rst_n) begin
            count<='b0;
        end
        else if(current_state == STATE_COUNTING) begin
            if(count < MILISEC_10)
                count<=count+1'b1;
            else begin
                count<='b0;
            end
        end
        else if (current_state == STATE_TIMEOUT) begin
            count<='b0;
```

```
        end
        else if(current_state==STATE_PAUSE) begin
            count<=count;
        end
    end

    //flag 是判断条件，当 flag 是 true 的时候 ms 位加一
    wire flag;
    assign flag   =   current_state == STATE_COUNTING && count == MILISEC_10 ;


    //毫秒处理
    always @ (posedge clk or negedge rst_n) begin
        if(~rst_n) begin
            milisec <= 'b0;
        end
        else if(flag) begin
            if(milisec=='h99) begin
                milisec <= 'b0;
            end
            else if(milisec[3:0]=='h9)begin
                milisec[3:0]<='h0;
                milisec[7:4]<= milisec[7:4] +1'b1;
            end
            else begin
                milisec[3:0] <= milisec[3:0] + 1'b1;
            end
        end
    end

    //秒数处理
    always @ (posedge clk or negedge rst_n) begin
        if(~rst_n) begin
            sec <='b0;
        end
        else if (flag&&milisec=='h99) begin
            if(sec == 'h59)begin
                sec<='b0;
            end
            else if (sec[3:0]=='h9) begin
                sec[3:0] <='b0;
                sec[7:4] <= sec[7:4]+1'b1;
            end
            else begin
```

```
                sec[3:0] <= sec[3:0] + 1'b1;
            end
        end
        else begin
            sec <= sec;
        end
end

//分钟处理
always @ (posedge clk or negedge rst_n) begin
        if(~rst_n) begin
            min <= 1'b0;
        end
        else if(flag&&sec == 'h59&&milisec=='h99) begin
            if(min == 1'b0) begin
                min <=1'b1;
            end
            else begin
                min <= min;
                timeout <= 1'b1;
            end
        end
end

endmodule
```

**display 模块：**

```
module display(
input clk,   //时钟信号
input wire[7:0] sec,   //输入秒数
input wire[7:0] milisec,   //输入毫秒数
output reg[3:0] select_digit,   //数码管选择信号
output reg[7:0] show_led   //LED 控制信号
);
        reg[3:0] show_num;   //当前数码管显示时间
        reg[1:0] count;   //递增从而切换数码管进行显示
        reg point;   //判断小数点是否亮灯
        parameter   //LED 控制信号
         zero = 8'b00111111,   //63
            one = 8'b00000110,   //6
         two = 8'b01011011,   //91
         three = 8'b01001111,   //79
         four = 8'b01100110,   //102
```

```
        five = 8'b01101101,   //109
        six = 8'b01111101,   //125
        seven = 8'b00000111,   //7
        eight = 8'b01111111,   //127
        nine = 8'b01101111;   //111

initial show_num=4'b0000;
      initial count=2'b00;
always @(posedge clk) begin   //切换数码管，并更新显示数字
       case(count)
            2'b00: begin
                  select_digit<=4'b0001;
                       show_num<=milisec[3:0];
                  point=0;
            end

            2'b01: begin
                  select_digit<=4'b0010;
                  show_num<=milisec[7:4];
                  point=0;
            end

            2'b10: begin   //第 3 位数码管需要亮小数点
                  select_digit<=4'b0100;
                  show_num<=sec[3:0];
                  point=1;
            end

            2'b11: begin
                  select_digit<=4'b1000;
                  show_num<=sec[7:4];
                  point=0;
            end
       endcase
            count <= count+1'b1;   //切换数码管
      end

always @(*) begin   //不断重复，当 show_num 改变，更新 LED 控制信号的值
            if(point==0) begin
             case(show_num)
                  4'b0000:show_led=zero;
                  4'b0001:show_led=one;
                  4'b0010:show_led=two;
                  4'b0011:show_led=three;
```

```
                4'b0100:show_led=four;
                4'b0101:show_led=five;
                4'b0110:show_led=six;
                4'b0111:show_led=seven;
                4'b1000:show_led=eight;
                4'b1001:show_led=nine;
            endcase
        end
        else begin   //需要亮小数点，加上 8'b10000000 则 H 为 1，小数点亮
            case(show_num)
                4'b0000:show_led=zero+8'b10000000;
                4'b0001:show_led=one+8'b10000000;
                4'b0010:show_led=two+8'b10000000;
                4'b0011:show_led=three+8'b10000000;
                4'b0100:show_led=four+8'b10000000;
                4'b0101:show_led=five+8'b10000000;
                4'b0110:show_led=six+8'b10000000;
                4'b0111:show_led=seven+8'b10000000;
                4'b1000:show_led=eight+8'b10000000;
                4'b1001:show_led=nine+8'b10000000;
            endcase
        end
    end
endmodule
```

**key_rebounce 模块：**

```
module key_rebounce(                    //定义输入输出端口
    input           clk,                //系统时钟
    input           rst_n,              //复位信号
    input           key_in,             //按键输入信号
    output reg      key_out             //按键去抖输出信号
    );

    reg key_in0;                //记录上个时钟周期的按键输入信号
    reg [19:0] count;           //计数寄存器

    wire change;                //按键输入改变信号

    parameter     C_COUNTER_NUM = 5;
//      parameter     C_COUNTER_NUM = 180000;

    always@(posedge clk or negedge rst_n)
        if(!rst_n)//复位处理
```

```
            key_in0 <= 0;
        else            //记录按键输入
            key_in0 <= key_in;
    //如果前后两个时钟按键输入数据不同，将此信号置为 1
    assign change=(key_in & !key_in0)|(!key_in & key_in0);

    always@(posedge clk or negedge rst_n)
        if(!rst_n)      //复位处理
            count <= 0;
        else if(change)//按键输入发生改变，重新开始计数
            count <= 0;
        else
            count <= count + 1;

    always@(posedge clk or negedge rst_n)
        if(!rst_n)      //复位处理
            key_out <= 0;
        else if(count >= C_COUNTER_NUM - 1)//更改输出信号
            key_out <= key_in;
endmodule
```

**LED 模块：**

```
module LED(
    input wire clk,
    input wire rst_n,
    input wire min,
    input wire timeout,   //超时信号
    output wire l1,   //LED 灯，时间>=60s 亮起
    output wire l2   //LED 灯，超时亮起
);
    reg light1,light2;   //对应于 l1,l2
    initial light1=1'b0;
    initial light2=1'b0;

    always @(posedge clk or negedge rst_n) begin
        if(~rst_n) begin
            light1=1'b0;
            light2=1'b0;
        end
        else begin
            light1=min;
            if(timeout==1'b1)
                light2=1'b1;
```

```
            end
        end

        assign l1=light1;
        assign l2=light2;
endmodule
```

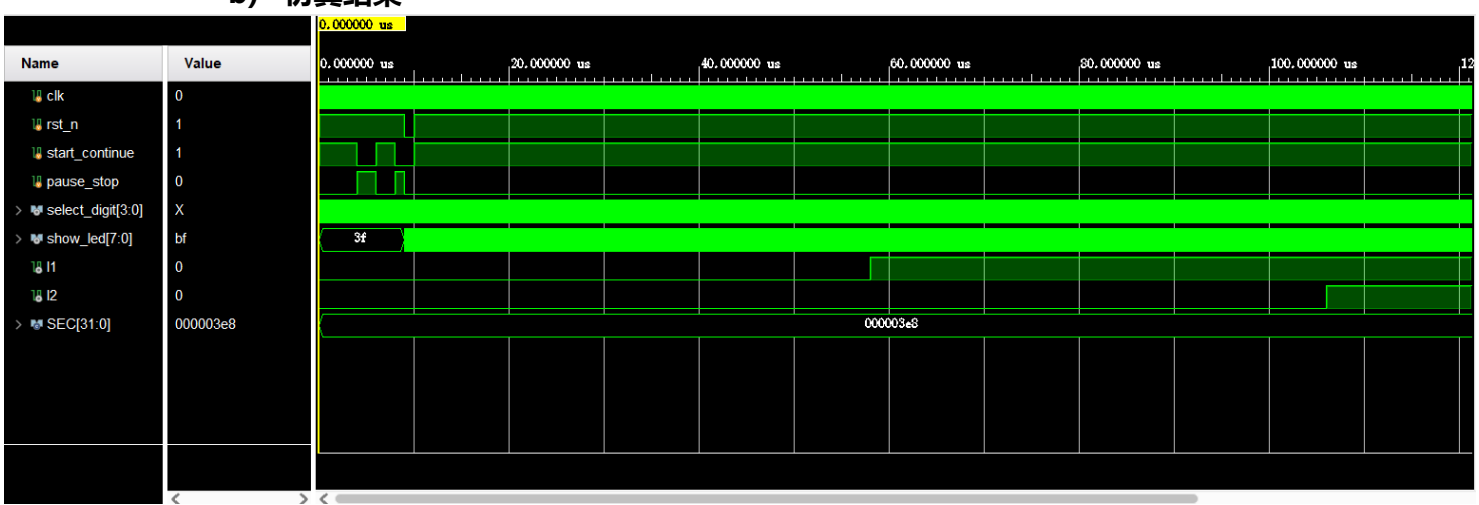
## 4. 电路验证

### a) TestBench

```
`timescale 1ns / 1ns

module test(
    );

reg clk;                    // 时钟信号
reg rst_n;                  // 复位按键
reg start_continue;         // 开始按键
reg pause_stop;             // 暂停按键
wire[3:0] select_digit; // 段选
wire[7:0] show_led;         // 显示
wire l1;                    // 1min 标志
wire l2;                    // 超时标志

running_timer running_timer_instance(
    .clk                    (clk),
    .rst_n                  (rst_n),
    .start_continue         (start_continue),
    .pause_stop             (pause_stop),
    .select_digit           (select_digit),
    .show_led               (show_led),
    .l1                     (l1),
    .l2                     (l2)
);
initial clk = 1'b0 ;
always #1 clk=~clk ;
// SEC 代表 1 S
parameter SEC = 1000 ;
initial begin
    $dumpfile("dumpfile.vcd");
    $dumpvars;
    // 当前时刻 0S ， 开始计时
    start_continue = 1'b1;
    pause_stop = 1'b0;
```

```
        rst_n = 1'b1;

        #(4* SEC) ;
        // 当前时刻 4S，按下暂停
        pause_stop = 'b1;
        start_continue = 0 ;

        #(2 * SEC) ;
        // 当前时刻 6S ， 继续计时
        pause_stop = 'b0;
        start_continue = 1 ;

        #(2*SEC) ;
        // 当前时刻 8S ， 暂停，
        pause_stop = 'b1;
        start_continue = 0 ;

        #(SEC) ;
        // 当前时刻 9S， 清零
        pause_stop=0 ;
        rst_n=0 ;

        #(SEC) ;
        // 当前时刻 10S, 重新开始计时
        rst_n = 1 ;
        start_continue = 'b1;
        // 在 70S 时， L1 输出 1，
        // 第一个 LED 灯亮，表示 1min
        // 在 130S 时， I2 输出 1，
        // 第二个 LED 灯亮， 表示超时
        #(125*SEC) ;

        $finish;
    end

endmodule
```

**b) 仿真结果**



为了方便观察 把 1000ns 近似为 1s

观察仿真图可以发现

0s 时： 电路开始计时

4s 时： 按下暂停键，

6s 时： 继续计时

8s 时： 按下暂停键

9s 时： 按下复位键，计时器变为 0

10s 时： 电路重新开始计时

70s 时： l1 输出 1，表示计时器已经计时一分钟

130s 时： l2 输出为 1， 表示超过 1min59s99ms,处于超时状态

## 5. 电路上板

为 4 个数码管、数码管的 LED 信号、2 个 LED 灯、开始暂停复位按键进行管脚分配。

Tcl Console | Messages | Log | Reports | Design Runs | I/O Ports

| Name | Direction | Neg Diff Pair | Package Pin | Fixed | Bank | I/O Std | Vcco | Vref | Drive Strength | Slew Type | Pull Type | Off-Chi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| All ports (18) | | | | | | | | | | | | |
| select_digit (4) | OUT | | | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| select_digit[3] | OUT | | H1 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| select_digit[2] | OUT | | C1 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| select_digit[1] | OUT | | C2 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| select_digit[0] | OUT | | G2 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led (8) | OUT | | | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[7] | OUT | | D5 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[6] | OUT | | B2 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[5] | OUT | | B3 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[4] | OUT | | A1 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[3] | OUT | | B1 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[2] | OUT | | A3 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[1] | OUT | | A4 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| show_led[0] | OUT | | B4 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| Scalar ports (6) | | | | | | | | | | | | |
| clk | IN | | R11 | ☑ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE |
| l1 | OUT | | T5 | ☑ | 34 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| l2 | OUT | | K2 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VT |
| pause_stop | IN | | J2 | ☑ | 35 | LVCMOS33* | 3.300 | | | | NONE | NONE |
| rst_n | IN | | P15 | ☑ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE |
| start_continue | IN | | R17 | ☑ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE |

## 6. 实验心得

一开始将题目转化为状态图，然后思考如何将状态图转化为代码，并输出分秒毫秒，这个过程中遇到了一些问题。同时，如何将整个计时功能分成不同模块，模块间的组合与调用，也花了一些时间，最终决定用一个主模块调用所有其余模块。