**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）。

1. **电路设计**

下面将从模块的角度分析此电路设计。

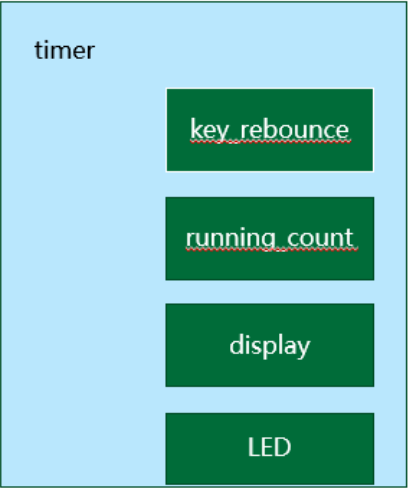
第一个模块：key\_rebounce，起到的作用是按键消抖。输入为时钟，复位信号和按键信号；输出是消抖过的按键信号。

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。



1. **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

1. **电路验证**
   1. **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， l2 输出 1，**

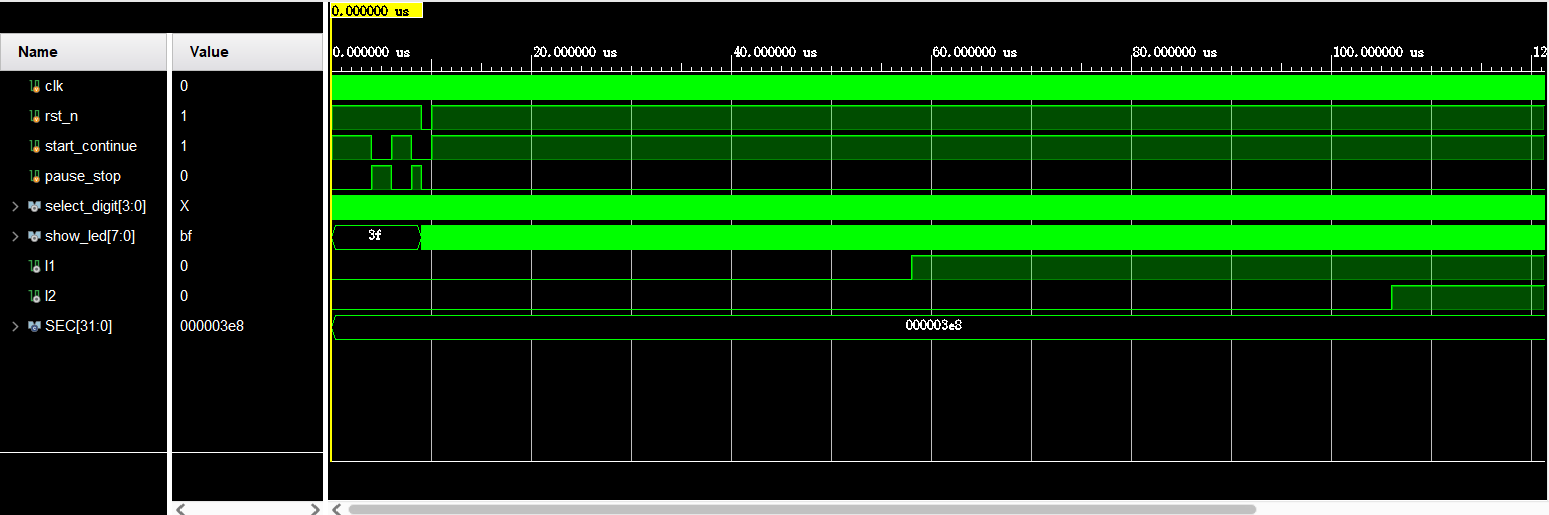
**// 第二个LED灯亮， 表示超时**

**#(125\*SEC) ;**

**$finish;**

**end**

**endmodule**

* 1. **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

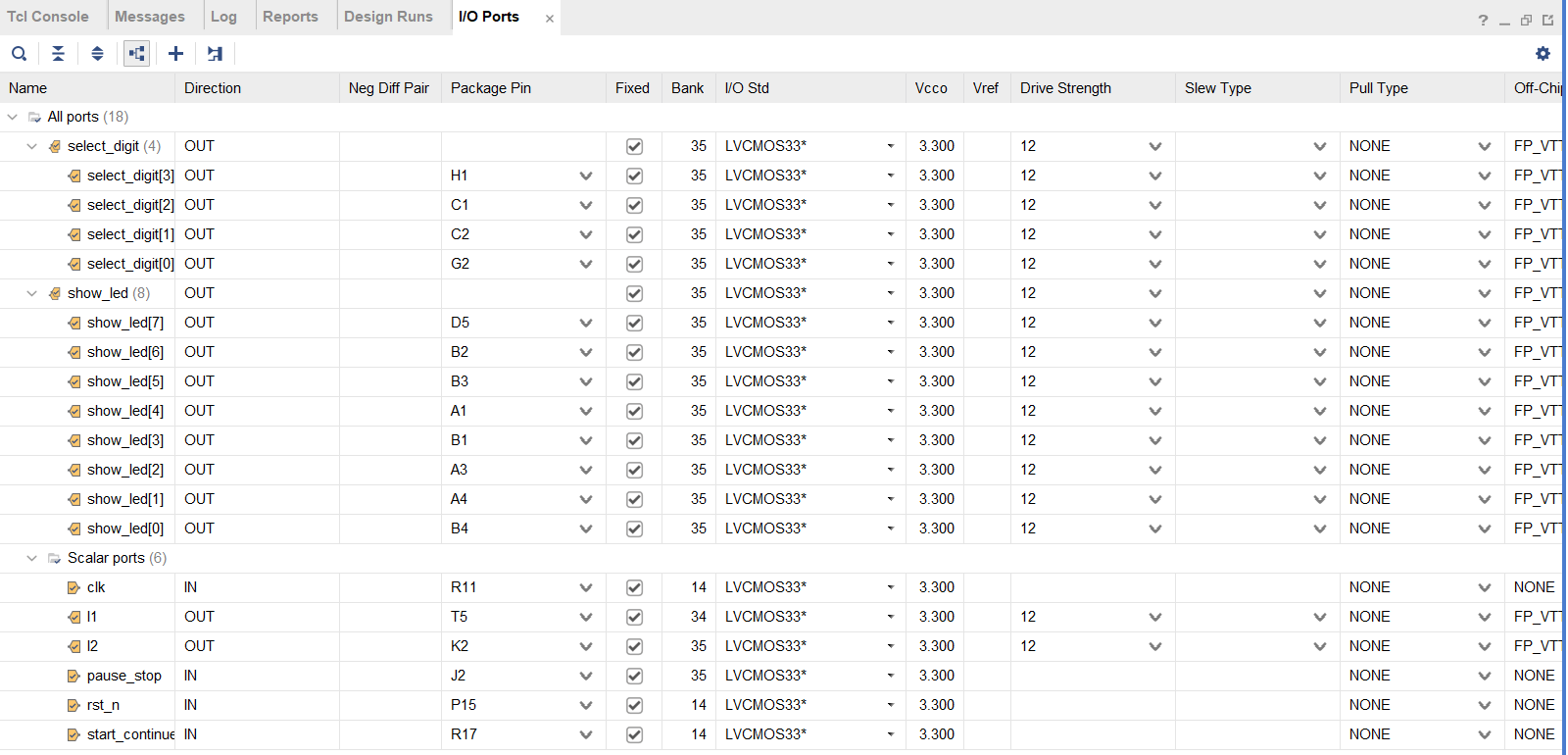
10s时： 电路重新开始计时

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

1. **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。



1. **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。