**实验四 综合电路设计实验报告**

**组长**：xxx **学号**：xxx

**班级**：xxx **手机**：

**组员：**xxx **学号：**xxx

**班级：**xxx **手机：**

**组员**：xxx  **学号**：xxx

**班级**：xxx **手机**：

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；

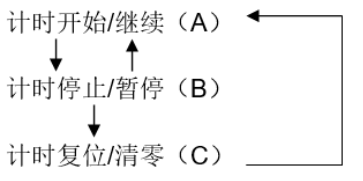
◼ “秒”用两位数码管显示：十位、个位；

◼ “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），

键控流程如下：



1. **电路设计**

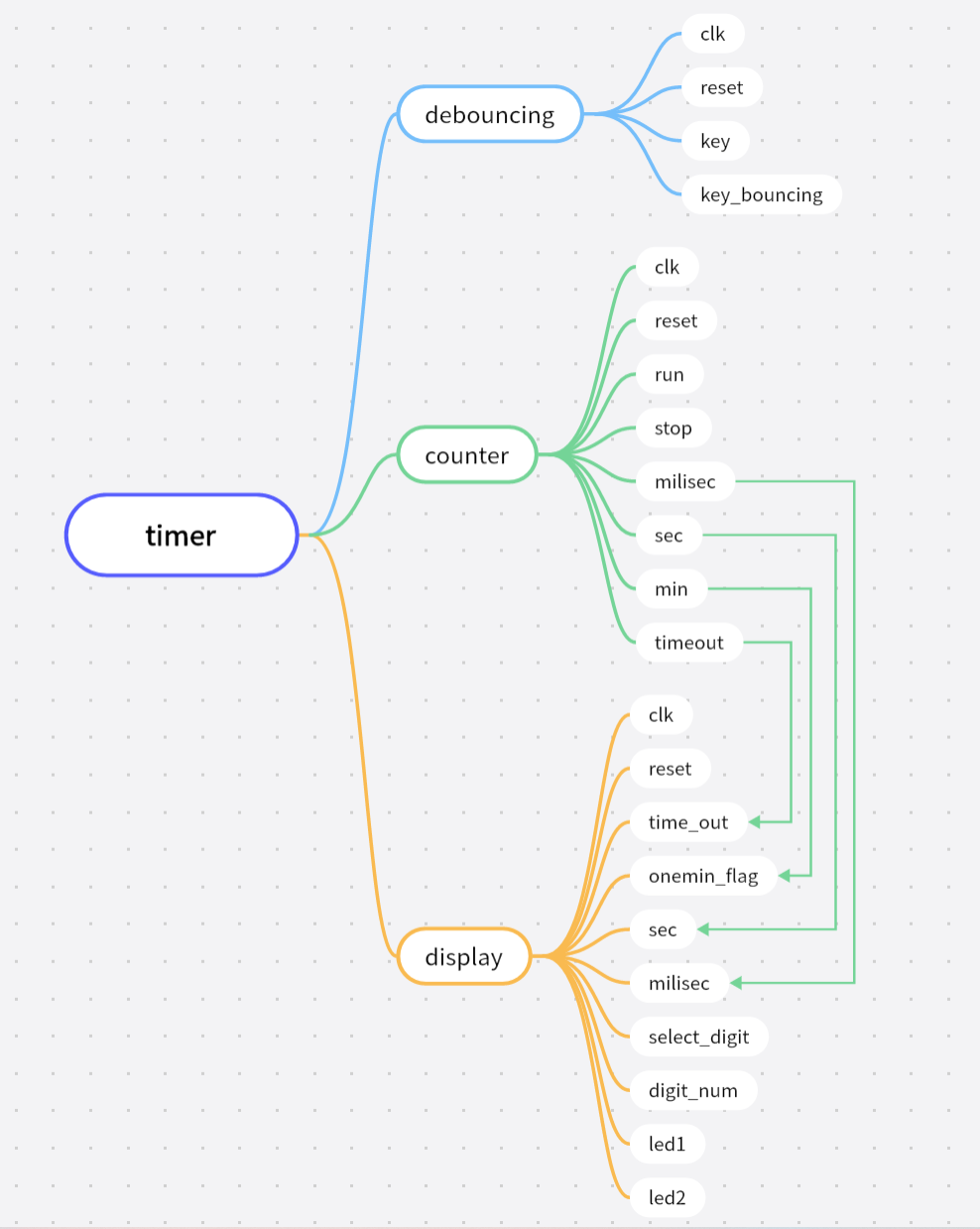
电路设计共分为一大模块、三小模块：

模块一：debouncing模块，主要功能是为按键信号消抖，输入时钟、reset信号以及需要消抖的key（run和stop），输出消抖后的按键信号。

模块二：counter模块，主要功能为实现计时器的计时逻辑，输入时钟、复位信号、消抖后的run、stop信号，输出分、秒、毫秒、超时信号，作为下一展示模块的信号输入。

模块三：display模块，主要功能为对counter模块时间信号处理成对电路板的控制信号，完成数字的显示以及LED灯的亮灭。输入时钟、复位信号以及counter模块的输出信号，输出数码管选择信号、被选中数码管显示的数字信号和两个LED灯控制信号。

总模块：timer模块，对三小模块进行组合，实现短跑计时器的功能。



1. **电路实现**

**timer模块：**

module timer(

input clk, // 时钟信号

input reset, // 重置信号

input run, // 继续计时信号

input stop, // 停止计时信号

output [3:0] select\_digit, // 数码管选择信号

output [7:0] digit\_num, // 选中的数码管显示的数字

output led1, // 1分钟

output led2 // 超时灯

);

wire run\_de,stop\_de;

//按键信号消抖模块

debouncing run\_debouncing(

.reset (reset),

.key (run),

.clk (clk),

.key\_bouncing (run\_de)

);

debouncing stop\_debouncing(

.reset (reset),

.key (stop),

.clk (clk),

.key\_bouncing (stop\_de)

);

wire min;

wire [7:0] sec;

wire [7:0] milisec;

wire timeout;

//计时模块

counter counter(

.clk (clk),

.reset (reset),

.run (run\_de),

.stop (stop\_de),

.timeout (timeout),

.min (min),

.sec (sec),

.milisec (milisec)

);

//展示模块

display display(

.clk (clk),

.reset (reset),

.sec (sec),

.milisec (milisec),

.onemin\_flag (min),

.time\_out (timeout),

.select\_digit (select\_digit),

.digit\_num (digit\_num),

.led1 (led1),

.led2 (led2)

);

Endmodule

**debouncing模块：**

module debouncing(

input clk, // 时钟信号

input reset, // 重置信号

input key, // 输入按键

output reg key\_bouncing // 输出消抖后的按键值

);

reg R\_key\_in0; // 判断是否需要重置

reg [19:0] R\_count; // 计数器

wire W\_change; // 判断是否发生了改变

parameter C\_COUNTER\_NUM = 5; // 设置计数器的检测次数为5

always@(posedge clk or negedge reset)

if(!reset)

R\_key\_in0 <= 0; // 如果按下重置键

else

R\_key\_in0 <= key;

assign W\_change = (key & !R\_key\_in0) | (!key & R\_key\_in0);

// 检测当前按键的值是否与前一时刻相同，即是否发生变化

// 判断计数器是否加一

always@(posedge clk or negedge reset)

if(!reset)

R\_count <= 0; // 按下重置键，计数器清零

else if(W\_change)

R\_count <= 0; // 如果按键值发生了改变，计数器清零

else

R\_count <= R\_count + 1; // 正常情况，计数器加一

always@(posedge clk or negedge reset)

if(!reset)

key\_bouncing <= 0;

else if(R\_count >= C\_COUNTER\_NUM - 1)

key\_bouncing <= key; // 计数器值达到了设定的最大值，判断已经不再抖动，获得输出

endmodule

**count模块：**

module counter(

input clk, //时钟

input reset, //复位

input run, //开始

input stop, //停止

output reg timeout, //是否超时

output reg min, //分

output reg [7:0] sec, //秒

output reg [7:0] milisec //毫秒

);

//分别为不同的状态赋不同的值

parameter state\_initial = 2'b00 ;

parameter state\_run = 2'b01 ;

parameter state\_stop = 2'b10 ;

parameter state\_timeout = 2'b11 ;

//声明

reg [1:0] state = state\_initial, next\_state = state\_initial;

//是否复位

always @ (posedge clk or negedge reset)

begin

//复位则initial

if(!reset) begin

state <= state\_initial;

end

//不复位

else begin

state <= next\_state;

end

end

//计时10ms

//10ms的时钟次数

parameter ms\_10 = 1000000;

//记录时钟次数

reg [31:0] count = 0;

//记录是否到了10ms

wire flag;

assign flag = count == ms\_10;

//count计数

always @ (posedge clk)

begin

case(state)

state\_initial : begin

count <= 0;

timeout <= 1'b0;

end

state\_run : begin

if(count < ms\_10)

count <= count + 1;

else begin

count <= 0;

end

end

state\_stop : begin

count <= count;

end

state\_timeout : begin

count <= 0;

timeout <= 1'b1;

end

endcase

end

//状态转移

always @ (\*)

begin

case(state)

state\_initial : begin

//默认状态转移到计时状态

if(run == 1'b1) begin

next\_state = state\_run;

end

//默认状态不变

else begin

next\_state = state\_initial;

end

end

state\_run : begin

//转移到暂停状态

if(stop == 1'b1) begin

next\_state = state\_stop;

end

//转移到超时状态

else if (min == 1'b1 && sec == 'h59 && milisec == 'h99 && flag ) begin

next\_state = state\_timeout;

end

//计时状态不变

else begin

next\_state = state\_run;

end

end

state\_stop : begin

//转移到计时状态

if(run == 1'b1)

next\_state = state\_run;

//暂停状态不变

else

next\_state = state\_stop;

end

state\_timeout : begin

//一直保持timeout状态，直到被reset（此时next\_state自动转为initial）

next\_state = state\_timeout;

end

endcase

end

//毫秒处理

always @ (posedge clk or negedge reset) begin

if(!reset) begin

milisec <= 'b0;

end

else if(flag) begin

//990ms归0

if(milisec == 'h99) begin

milisec <= 'h00;

end

//个位为9进位

else if(milisec[3:0] == 'h9)begin

milisec[3:0] <= 'h0;

milisec[7:4]<= milisec[7:4] + 1'b1;

end

//个位加1

else begin

milisec[3:0] <= milisec[3:0] + 1'b1;

end

end

end

//秒处理

always @ (posedge clk or negedge reset) begin

if(!reset) begin

sec <='b0;

end

//1000ms时second加1

else if (flag && milisec == 'h99) begin

//60s归0

if(sec == 'h59)begin

sec <= 'h00;

end

//个位10s进1

else if (sec[3:0]=='h9) begin

sec[3:0] <='b0;

sec[7:4] <= sec[7:4]+1'b1;

end

//个位加1

else begin

sec[3:0] <= sec[3:0] + 1'b1;

end

end

else begin

sec <= sec;

end

end

//分处理

always @ (posedge clk or negedge reset) begin

if(!reset) begin

min <= 1'b0;

end

//60s

else if(flag && sec == 'h59 && milisec=='h99) begin

//min为0

if(min == 1'b0) begin

min <= 1'b1;

end

//min为1，timeout

else begin

min <= min;

end

end

end

Endmodule

**display模块：**

module display(clk, reset, sec, milisec, onemin\_flag, time\_out, select\_digit, digit\_num, led1, led2);

input clk;

input reset;

input [7:0] sec, milisec;

input onemin\_flag; // 大于等于一分钟标志

input time\_out; // 超时信号

output [3:0] select\_digit;

output [7:0] digit\_num;

output wire led1; // 1分钟灯

output wire led2; // 超时灯

reg [3:0] showing\_num;

reg [3:0] showing\_digit = 4'b0001;

parameter period = 50000;

reg [31:0] counter = 0;

assign led1 = onemin\_flag; // 一分钟输入转换为一分钟灯

assign led2 = time\_out; // 超时信号转换为超时灯

assign select\_digit = showing\_digit;

always @ (posedge clk or negedge reset) begin

if(counter >= period) begin

if(!reset) begin // 复位显示矩形

case(showing\_digit)

4'b0001: begin

showing\_digit = 4'b1000;

showing\_num = 4'b1010;

counter <= 0;

end

4'b0010: begin

showing\_digit = 4'b0001;

showing\_num = 4'b1100;

counter <= 0;

end

4'b0100: begin

showing\_digit = 4'b0010;

showing\_num = 4'b1011;

counter <= 0;

end

4'b1000: begin

showing\_digit = 4'b0100;

showing\_num = 4'b1011;

counter <= 0;

end

endcase

end

else if(time\_out) begin // 超时显示OUT

case(showing\_digit)

4'b0001: begin

showing\_digit = 4'b1000;

showing\_num = 4'b0000;

counter <= 0;

end

4'b0010: begin

showing\_digit = 4'b0001;

showing\_num = 4'b1110;

counter <= 0;

end

4'b0100: begin

showing\_digit = 4'b0010;

showing\_num = 4'b0111;

counter <= 0;

end

4'b1000: begin

showing\_digit = 4'b0100;

showing\_num = 4'b1101;

counter <= 0;

end

endcase

end

else begin // 否则显示输入的秒和毫秒

case(showing\_digit)

4'b0001: begin

showing\_digit = 4'b1000;

showing\_num = sec[7:4];

counter <= 0;

end

4'b0010: begin

showing\_digit = 4'b0001;

showing\_num = milisec[3:0];

counter <= 0;

end

4'b0100: begin

showing\_digit = 4'b0010;

showing\_num = milisec[7:4];

counter <= 0;

end

4'b1000: begin

showing\_digit = 4'b0100;

showing\_num = sec[3:0];

counter <= 0;

end

endcase

end

end

else begin

counter <= counter + 1;

end

end

num2led num2led(showing\_num, digit\_num, showing\_digit);

endmodule

module num2led(num, led, showing\_digit); // 该模块将数字转换成数码管的显示

input [3:0] num;

input [3:0] showing\_digit;

output [7:0] led;

// 10为[，11为二，12为],13为U，14为-

assign led[0] = (num == 4'b0000 || num == 4'b0001 || num == 4'b1110

|| num == 4'b0111 || num == 4'b1010 || num == 4'b1011

|| num == 4'b1100 || num == 4'b1101) ? 0 : 1;

assign led[1] = (num == 4'b0001 || num == 4'b0010 || num == 4'b0011

|| num == 4'b0111 || num == 4'b1011 || num == 4'b1100

|| num == 4'b1110) ? 0 : 1;

assign led[2] = (num == 4'b0001 || num == 4'b0011 || num == 4'b0100

|| num == 4'b0101 || num == 4'b0111 || num == 4'b1001

|| num == 4'b1011 || num == 4'b1100 || num == 4'b1110) ? 0 : 1;

assign led[3] = (num == 4'b0001 || num == 4'b0100 || num == 4'b0111

|| num == 4'b1110) ? 0 : 1;

assign led[4] = (num == 4'b0010 || num == 4'b1010 || num == 4'b1011

|| num == 4'b1110) ? 0 : 1;

assign led[5] = (num == 4'b0101 || num == 4'b0110 || num == 4'b1010

|| num == 4'b1011 || num == 4'b1110) ? 0 : 1;

assign led[6] = (num == 4'b0001 || num == 4'b0100

|| num == 4'b1101) ? 0 : 1;

assign led[7] = showing\_digit == 4'b0100;

endmodule

1. **电路验证**
   1. **TestBench**

module timer\_testbench;

// 时钟信号和其他输入信号

reg clk;

reg reset;

reg run;

reg stop;

// 输出信号

wire [3:0] select\_digit;

wire [7:0] digit\_num;

wire led1;

wire led2;

parameter PERIOD = 100000;

// 实例化顶层模块

timer dut (

.clk(clk),

.reset(reset),

.run(run),

.stop(stop),

.select\_digit(select\_digit),

.digit\_num(digit\_num),

.led1(led1),

.led2(led2)

);

// 时钟生成模块

always #5 clk = ~clk;

// 初始化输入信号

initial begin

clk = 0;

reset = 0;

run = 0;

stop = 0;

// 等待一段时间，然后释放复位信号

#(20\*PERIOD) reset = 1;

// 模拟计时器的操作

// 情况1：计时器启动，然后停止

#(20 \* PERIOD) run = 1; // 开始计时

#(50 \* PERIOD)

run = 0;

stop = 1; // 停止计时

#(20 \* PERIOD) stop = 0;

// 情况2：计时器启动，然后继续计时，再停止

#(10\*PERIOD) reset = 0;

#(10\*PERIOD) reset = 1;

#(20\*PERIOD) run = 1; // 开始计时

#(20\*PERIOD) run = 0;

#(50\*PERIOD) stop = 1; // 停止计时

#(20\*PERIOD) stop = 0; // 停止计时

// 情况3：计时器启动，然后停止，再重新启动

#(20\*PERIOD) reset = 0;

#(20\*PERIOD) reset = 1;

#(20\*PERIOD) run = 1; // 开始计时

#(20\*PERIOD) run = 0; // 开始计时

#(50\*PERIOD) stop = 1; // 停止计时

#(20\*PERIOD) stop = 0; // 停止计时

// 情况4：计时器启动，然后超时

#(20\*PERIOD) run = 1; // 开始计时

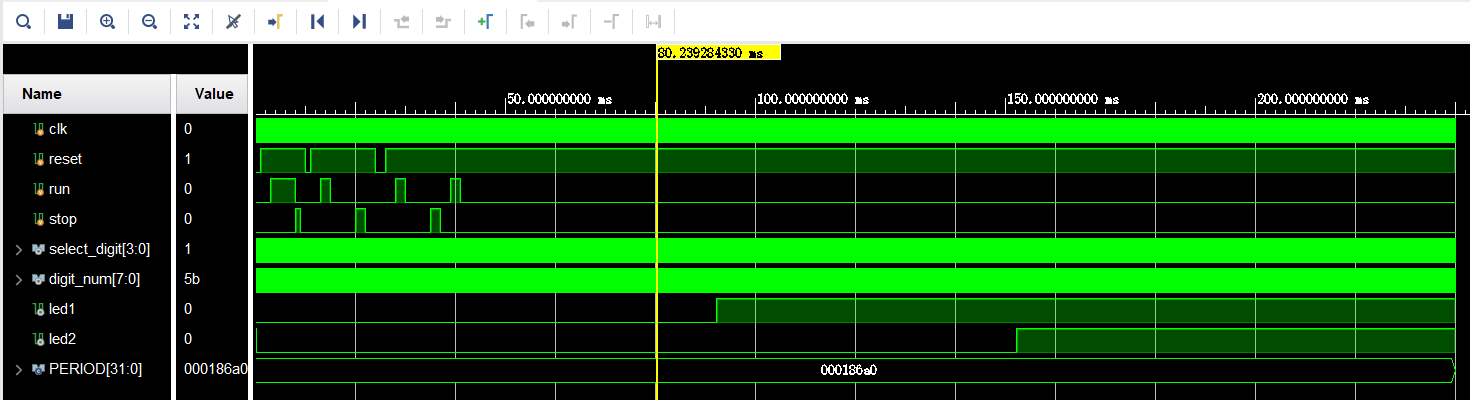
#(20\*PERIOD) run = 0;

#(150000\*PERIOD);

end

endmodule

* 1. **仿真结果**



测试说明:

由于现实时间和仿真模拟能力不同，我们将仿真中的1ms等效成现实中的1秒钟。

整体测试功能流程为：

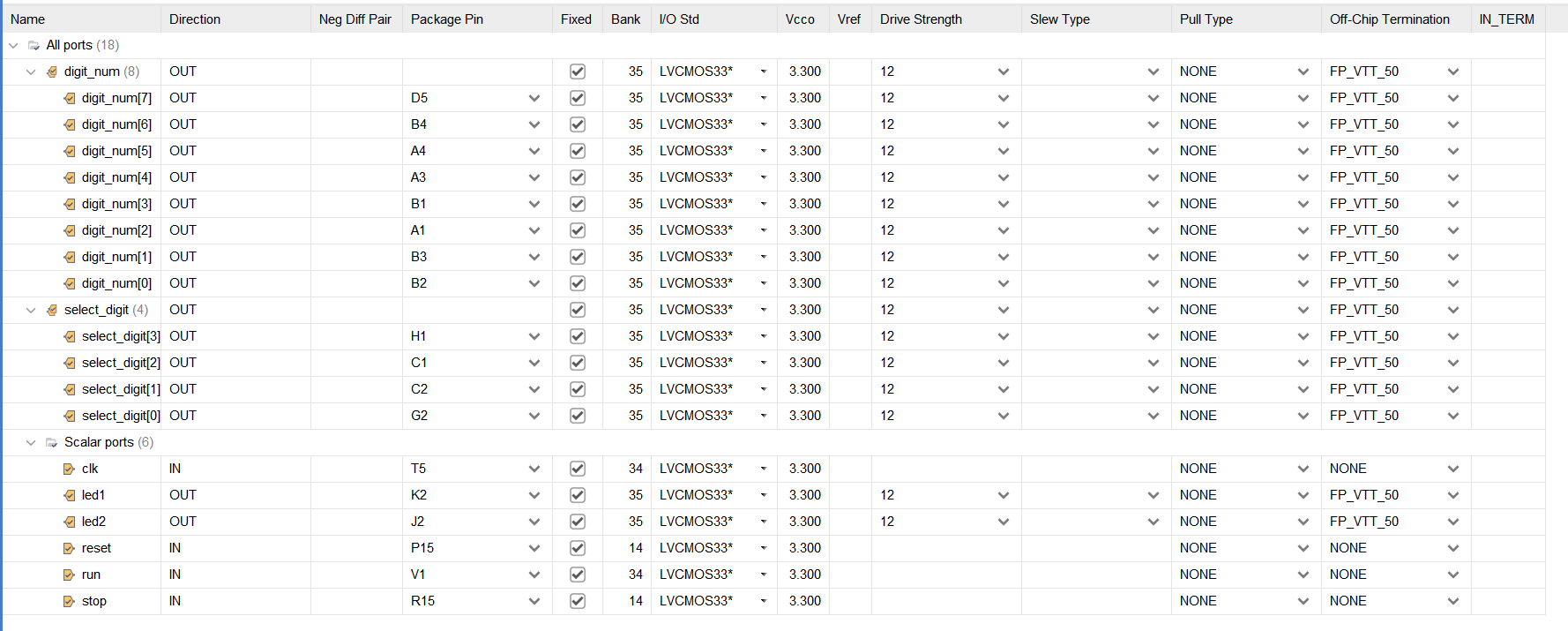
1. 按下Reset键，将状态清空；
2. 等待1秒后，我们按下run键，开始计时；
3. 3秒后，放开run键，迅速按下stop键并放开，停止计时；
4. 停止后迅速按下reset键并放开，再次清空状态；
5. 1秒后，重新按下run键，重新开始计时；
6. 等待3秒后，按下stop停止计时；
7. 1秒后，按下reset，重置状态；
8. 1秒后按下run，再次重新开始计时；
9. 3秒后暂停，1秒后再次按下run重新开始计时；
10. 接下来令系统自动进行计时，观察波形即可。

1-9步中，我们反复测试了reset键、run键、stop键的功能效果，分别进行了单独的功能测试，连续的功能测试，间隔按下run和stop进行断点重新计时；

第10步，我们测试了程序的一分钟提示和2分钟报警功能，我们特意将计时设为从已有一段时间计时并且被暂停的状态开始，并对提示和报警时间进行预测，波形显示，结果完美符合预测。整体通过测试。

1. **电路上板**

管脚配置中，为数码管选择信号、数码管LED信号、超时LED灯、分钟LED灯分配相应的管脚。



1. **实验心得**

实验中碰到比较困难的部分之一是对于计时器不同模块的设计，不仅需要明确地划分不同模块的作用，还要通过统一的变量使得模块之间可以正确地整合在一起，最终实现整体的计时功能。本实验中我们将整个计时器分为3个小模块：debouncing、counter、display，分别处理消抖、计时、展示，让小组的分工与实验思路更加清晰。

模块化处理是极其重要的思维和操作方式。在设计实现方法时，模块化设计能清楚地进行功能分离，使得设计思路更加清晰、有效地降低设计的难度。在对复杂功能进行实现时，将功能模块化可以有效提高代码的可读性、可维护性和复用性。在整个设计和上板过程中，尤其需要细致和耐心。通过逐步调试和测试，我们进行详细的记录和比对，反复钻研细节，提高了程序设计能力和实际应用能力。