实验四 综合逻辑电路实验报告

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：十位、个位；
- 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：

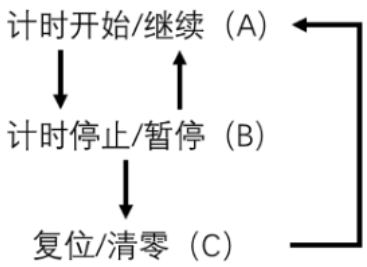


图1 键控流程

2. 电路设计

本实验中，一共设计了7个模块，分别是**防抖模块 、状态机模块、时钟分频计数器模块、位显示模块、LED模块、位选信号模块、段选信号模块。**

防抖模块通过抖动检测→稳定计数→确认稳定，从而实现按键去抖；**状态机模块**包括`START`、`PAUSE`、`RESET`、`OVERTIME`四个状态，分别表示开始计时、暂停计时、计时复位、超时报警，状态转移由三个按键`start`、`stop`、`reset`和`overtime`信号决定；**时钟分频计数器模块**通过将板载时钟频率以不同倍数降频，从而得到四个位选信号的变化频率，然后由四个计数器累计周期次数，并将累计值输出传递给位显示模块；**位显示模块**由4个计数器数值决定各位数字管显示的数值；**LED模块**决定分钟位和超时报警；**位选信号模块**计数时钟变化周期，并以每$2^{16} * 10^{-8}s$ $\left(< \frac{0.01}{4}s\right)$的时间间隔决定4-1多路选择器更新的位置；**段选信号模块**接受4-1多路选择器的输出，根据对应的数值匹配对应数码管的输出显示位置。

具体的电路设计如下：

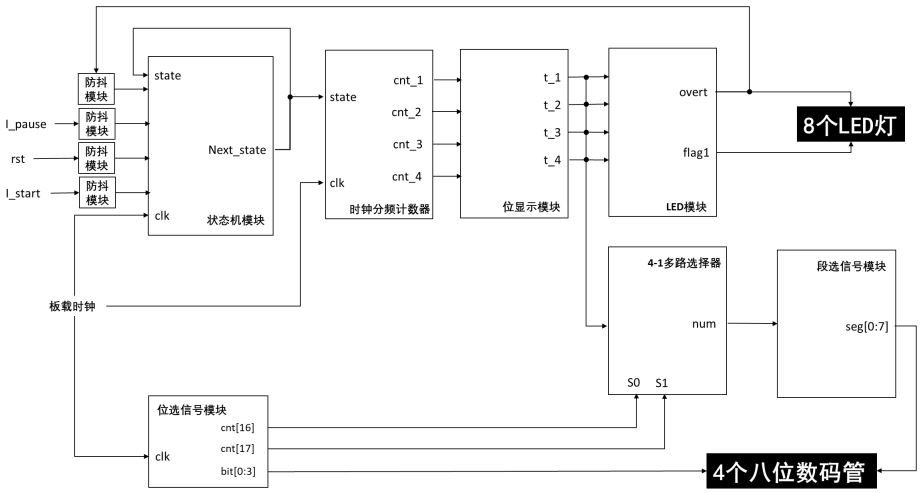


图2 电路设计

3. 电路实现

3.1 防抖模块

由于按键在按下或放开时，存在机械振动，且抖动时间一般在$10ms \sim 20ms$。按键信号不稳定，从而导致输出状态发生剧烈变化。为此，本模块实现了按键去抖：**抖动检测→稳定计数→确认稳定**，使得当按键信号不稳定时（稳定的时钟周期数不足），输出的按键信号不会发生变化。

```
module shake_detect(clk,I_key,O_key);
    input clk,I_key; // I_key为接收到的按键信号
    output O_key; // 输出去抖后的稳定信号

    reg O_key,R_key;
    reg[16:0]cnt;
    wire change;
    parameter MAX_CNT=100;
    always@(posedge clk)
        begin
            R_key<=I_key;
        end
    // 记录在较短的时钟周期内是否按键信号是否发生抖动
    assign change=(I_key&!R_key)|(!I_key&R_key);

    always@(posedge clk)
        begin
            if(change)cnt<=0; // 发生抖动重新进行稳定计数
            else cnt<=cnt+1;
        end

    always@(posedge clk)
        begin
```

```
            if(cnt>MAX_CNT) // 确认稳定
                O_key<=I_key;
        end
endmodule
```

3.2 状态机模块

本模块包括`START`、`PAUSE`、`RESET`、`OVERTIME`四个状态，分别表示**开始计时、暂停计时、计时复位、超时报警**；并通过三个按键`I_start`、`I_pause`、`rst`和信号`overt`确定状态的转移状况。具体的状态机图如下所示：

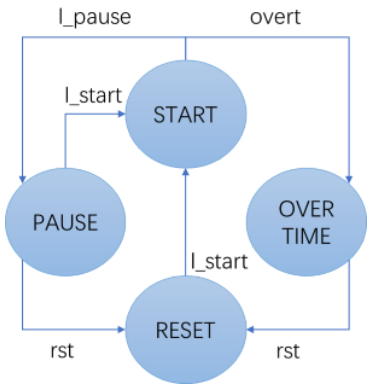


图3 状态机图

状态机模块的代码如下：

```
module the_state(clk,rst,I_start,I_pause,overt,state);
    input clk,rst,I_start,I_pause,overt;
    output reg state;
    reg next_state;
    parameter RESET = 2'b00,START = 2'b01,PAUSE =
2'b10,OVERTIME = 2'b11;
    always @(posedge clk or negedge rst)
    // 在rst的下降沿实现复位（即复位按钮松开时）
    begin
        if (~rst)
            begin
                state = RESET;
            end
        else
            begin
                state = next_state;
            end
    end

    always @(posedge clk or negedge rst)
        begin
            case(state)// 依据当前状态和输入信号选择下一状态
```

```
                    RESET: if(I_start)   next_state <= START;
                           else  next_state <= RESET;
                    START: if(I_pause)  next_state <= PAUSE;
                           else if (overt) next_state <=
OVERTIME;
                           else next_state <= START;
                    PAUSE: if(I_start)   next_state <= START;
                           else if (~rst) next_state <= RESET;
                           else next_state <= PAUSE;
                OVERTIME: if(~rst) next_state <= RESET;
                           else next_state <= OVERTIME;
            endcase
        end
endmodule
```

3.3 时钟分频计数器模块

本模块通过对输入的时钟信号进行分频处理，获得计时器四个数码管的变化周期。已知电路板的板载频率为$100MHz$，对应的时钟周期为10^{-8} s，则对应秒的十位、个位，以及毫秒的百位、十位，分别为时钟周期的10^9、10^8、10^7、10^6倍。

时钟分频计数器模块代码具体如下：

```
module divider(
    input clk,rst,state,
    output reg cnt_1,cnt_2,cnt_3,cnt_4);
    parameter RESET = 2'b00,START = 2'b01,PAUSE =
2'b10,OVERTIME = 2'b11;

    // 十秒位 10s
    always @(posedge clk or negedge rst)
        begin
            // 复位
            if(~rst)  cnt_1 <= 30'd0;
            // cnt_1计数器信号记到10^9(10s)后归零，重新计数
            else if(cnt_1 == 30'd9_9999_9999) cnt_1 <= 30'd0;
            // 计数递增
            else if(state ==START)  cnt_1 <= cnt_1 + 30'd1;
        end
    // 个秒位 1s
    always @(posedge clk or negedge rst)
        begin
            if(~rst)  cnt_2 <= 27'd0;
            //cnt_2计数器信号记到10^8(1s)后归零，重新计数
```

```
            else if(cnt_2 == 27'd9999_9999) cnt_2 <= 27'd0;
            else if(state ==START)  cnt_2 <= cnt_2 + 27'd1;
        end
    // 百毫秒位 0.1s
    always @(posedge clk or negedge rst)
        begin
            if(~rst)  cnt_3 <= 24'd0;
            //cnt_3计数器信号记到10^7(0.1s)后归零，重新计数
            else if(cnt_3 == 24'd999_9999) cnt_3 <= 24'd0;
            else if(state ==START)  cnt_3 <= cnt_3 + 24'd1;
        end
    // 十毫秒位 0.01s
    always @(posedge clk or negedge rst)
        begin
            if(~rst)  cnt_4 <= 20'd0;
            //cnt_4计数器信号记到10^6(0.01s)后归零，重新计数
            else if(cnt_4 == 20'd99_9999) cnt_4 <= 20'd0;
            else if(state == START)  cnt_4 <= cnt_4 + 20'd1;
        end
endmodule
```

3.4 位显示模块

本模块通过时钟分频计数器模块的计数器数值，以及输入的`rst`信号，决定四个数码管的显示数值的变化。

位显示模块代码具体如下：

```
module output_num(
    input clk,rst,
    input [29:0]cnt_1,cnt_2,cnt_3,cnt_4,
    output reg [3:0]t_1,t_2,t_3,t_4);

    // 十秒位 10s
    always @(posedge clk or negedge rst)
    begin
        // 复位
        if(~rst) t_1 <= 3'd0;
        // 发生进位，置为0
        else if(t_1 > 3'd5) t_1 <= 3'd0;
        // 计数器达到10^9个时钟周期后，显示数值+1
        else if(cnt_1 == 30'd9_9999_9999) t_1 <= t_1 +3'd1;
    end
    // 个秒位 1s
    always @(posedge clk or negedge rst)
```

```
    begin
        if(~rst) t_2 <= 4'd0;
        else if(t_2 > 4'd9) t_2 <= 1'd0;
        //计数器达到10^8个时钟周期后，显示数值+1
        else if(cnt_2 == 27'd9999_9999) t_2 <= t_2 +4'd1;
    end
    // 百毫秒位 0.1s
    always @(posedge clk or negedge rst)
    begin
        if(~rst) t_3 <= 4'd0;
        else if(t_3 > 4'd9) t_3 <= 1'd0;
        //计数器达到10^7个时钟周期后，显示数值+1
        else if(cnt_3 == 24'd999_9999) t_3 <= t_3 +4'd1;
    end
    // 十毫秒位 0.01s
    always @(posedge clk or negedge rst)
    begin
        if(~rst) t_4 <= 4'd0;
        else if(t_4 > 4'd9) t_4 <= 1'd0;
        //计数器达到10^6个时钟周期后，显示数值+1
        else if(cnt_4 == 20'd99_9999) t_4 <= t_4 +4'd1;
    end
endmodule
```

3.5 LED模块

在本模块设计了分钟位显示和超时报警显示。电路板上共有8个LED灯，其中**分钟位显示**即达到1:00:00时，**1**个LED灯亮起；**超时报警显示**即达到1:59:99时，**8**个LED灯亮起。

```
`timescale 1ns / 1ps
module led(
    input clk,rst,
    input [3:0]t_1,t_2,t_3,t_4,
    output reg overt,flag1);
    reg min_1;
    always @(posedge clk or negedge rst)
    begin
        // 复位
        if(~rst)
            begin
                flag1 <= 1'b0; // 59:99 亮起1个led
                overt <= 1'b0; // 1:59:99 亮起8个led
                min_1 <= 1'b0; // 记录计时器时间超过1min
            end
```

```
        // 计时达到59:99
        else if(flag1 != 1'b1 &&
                t_1 == 3'd5 && t_2 == 4'd9 && t_3 == 4'd9 &&
t_4 == 4'd9)
                    flag1 <= 1'b1;
        // 计时达到1:00:00
        else if(flag1 == 1'b1 &&
                t_1 == 3'd0&&t_2 == 4'd0&&t_3 == 4'd0&&t_4 ==
4'd0&&flag1 == 1'b1)
                    min_1=1;
        // 计时达到1:59:99
        else if(min_1 == 1'b1 &&
                t_1 == 3'd5 && t_2 == 4'd9 && t_3 == 4'd9 &&
t_4 == 4'd9)
                    overt <= 7'b111_1111;
    end
endmodule
```

3.6 位选信号模块

由于数码管不能同时显示所有位的值，某一时刻只能显示一个数码管的值，所以本模块使用**4-1多路选择器**，通过在**短时间内循环变换选择输出**4个位置的数码管值，利用人眼视觉暂留的特征，从而得到4个数码管的显示输出。

位选信号模块代码如下：

```
`timescale 1ns / 1ps
module bit_selection_signal(
    input clk,rst,t_1,t_2,t_3,t_4,
    output reg [3:0]num,
           wire [3:0]bit );
    reg [17:0]cnt;
    // 使用连续赋值语句实现动态点亮数码管
    assign bit[0] = (!cnt[17])&&(!cnt[16]);
    assign bit[1] = (!cnt[17])&&(cnt[16]);
    assign bit[2] = (cnt[17])&&(!cnt[16]);
    assign bit[3] = (cnt[17])&&(cnt[16]);

    // cnt[17:16]用于位选,对应4个数码管
    // bit[3:0]为四位位选信号，有且只有一个为1，表示当前选择显示输出的数
码管

    // 每2^16个时钟周期，显示输出1个数码管
    always @(posedge clk or negedge rst)
        begin
```

```
            if(~rst) cnt <= 18'd0;
            else cnt <= cnt +1'b1;
        end
    // 确定当前显示输出的数码管及显示的数值
    always @( * )
        case(cnt[17:16])
            2'b00:num = t_4;
            2'b01:num = t_3;
            2'b10:num = t_2;
            2'b11:num = t_1;
            default:num = 4'b0000;
        endcase
endmodule
```

3.7 段选信号模块

已知当前显示输出的数码管和显示数值，将对应的数值对应到数码管的具体显示位置。用8位二进制`seg`表示单个数码管中的8位，`seg`从高位到低位用$8\sim1$表示，对应数码管位置如下图所示：

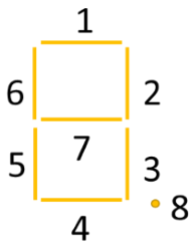


图4 数码管

段选信号模块代码如下：

```
`timescale 1ns / 1ps
module segment_selection_signal(
        input rst,
        input overt,
        input [3:0] bin,//四位表示当前时刻需要被输出的值
        output reg [7:0] seg //八位数码管显示信号
        );

        always @ (*)
            begin
                // 复位
                if(~rst)
                    seg = 8'b0000_0000;
                // 超时报警：4位数码管都显示8
                else if(overt == 1)
```

```
                    seg = 8'b0111_1111;
                // 计时器正常显示输出
                else
                    begin
                        case(bin)
                            4'b0000:seg = 8'b0011_1111; //0
                            4'b0001:seg = 8'b0000_0110; //1
                            4'b0010:seg = 8'b0101_1011; //2
                            4'b0011:seg = 8'b0100_1111; //3
                            4'b0100:seg = 8'b0110_0110; //4
                            4'b0101:seg = 8'b0110_1101; //5
                            4'b0110:seg = 8'b0111_1101; //6
                            4'b0111:seg = 8'b0000_0111; //7
                            4'b1000:seg = 8'b0111_1111; //8
                            4'b1001:seg = 8'b0110_1111; //9
                            default:seg = 8'b0000_0000;
                        endcase
                    end
            end
endmodule
```

4. 电路验证

4.1 testbench

testbench中的测试代码如下：

```
`timescale 1ns / 1ps
module testbench();
    reg CLK;
    reg rst,start,pause;
    wire overt,min1;
    wire state;
    wire [7:0] seg;
    wire [3:0]t_1,t_2,t_3,t_4;
    integer i;

    initial begin
       CLK <= 1'b0;
       start <= 1'b0;
       pause <= 1'b0;
       rst <= 1'b1;
   end

```

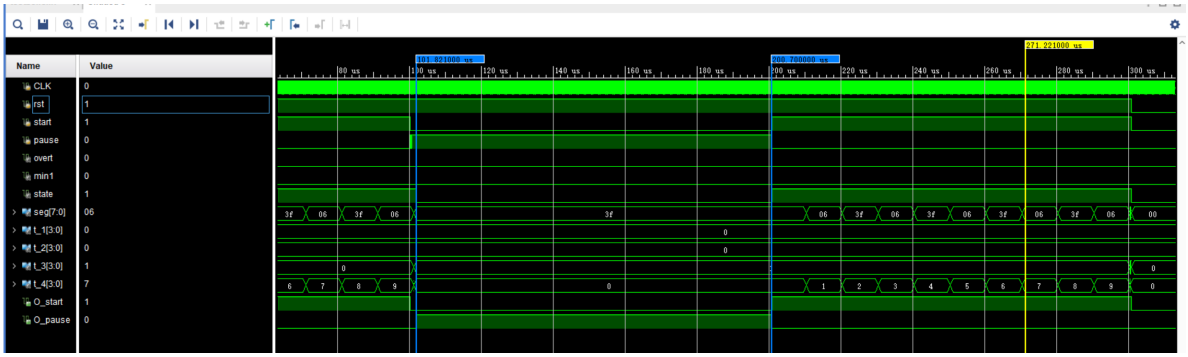
```
    // clock
    always #5
    begin
        CLK=~CLK;
    end

    //仿真
    initial begin
         //1.开始到暂停 暂停到开始
          #100;
          start=1;
          #100000;
          #100;
          start=0;
          pause=1;
             //暂停的防抖模块
             for(i=0;i<20;i=i+1)
             begin
                 #20;
                 if(i%2==0)
                     pause=0;
                 else
                     pause=1;
             end
          #100000;
          #100;
          pause=0;
          start=1;
          #100000;
          //开始到复位，复位到开始
          #100;
          rst=0;
          start=0;
          #100000;
          #100;
          rst=1;
          start=1;
          //跑到1min和超时
          for(i=0;i<20;i=i+1)
              begin
                 #10000000;
              end
          //超时到复位
          #100;
          rst=0;
```

```
63          start=0;
64          #10000;
65          //复位到开始
66          #100;
67          rst=1;
68          start=1;
69          #1000000;
70          $stop;
71      end
72
73      the_config cfig
74      (
75          .CLK(CLK),
76          .I_rst(rst),
77          .I_start(start),
78          .I_pause(pause),
79          .overt(overt),
80          .min1(min1),
81          .seg(seg),
82          .t_1(t_1),
83          .t_2(t_2),
84          .t_3(t_3),
85          .t_4(t_4),
86          .state(state)
87      );
88  endmodule
89
```

4.2 仿真结果

根据图3中的状态机，遍历各状态以实现对结果的检验。

4.2.1 从开始到暂停，从暂停到开始



从开始到暂停：可以看到，暂停信号产生后，计时器保持0.1秒不变。

从暂停到开始：暂停一段时间后，重新按下开始按钮，数字持续跳动，从0.1秒继续增长。图中，`t_1`、`t_2`、`t_3`、`t_4`分别对应信号的十秒、秒、十分秒与百分秒各位。

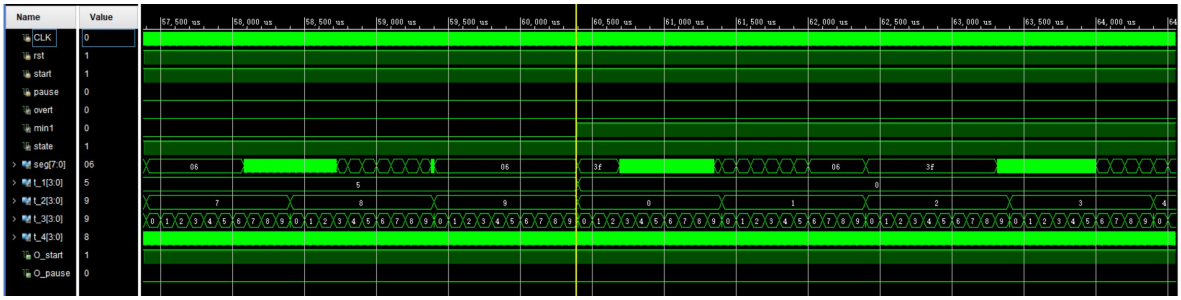
下面以`seg[7:0]`从暂停到开始的变化，阐释数码管的动态显示。

06是数码管的段选信号，对应显示的值为**1**（h'06=2'b0000_0110，对应图4数码管各段是否点亮的选择）

3f是数码管的段选信号，对应显示的值为**0**（h'3f=2'b0011_1111）

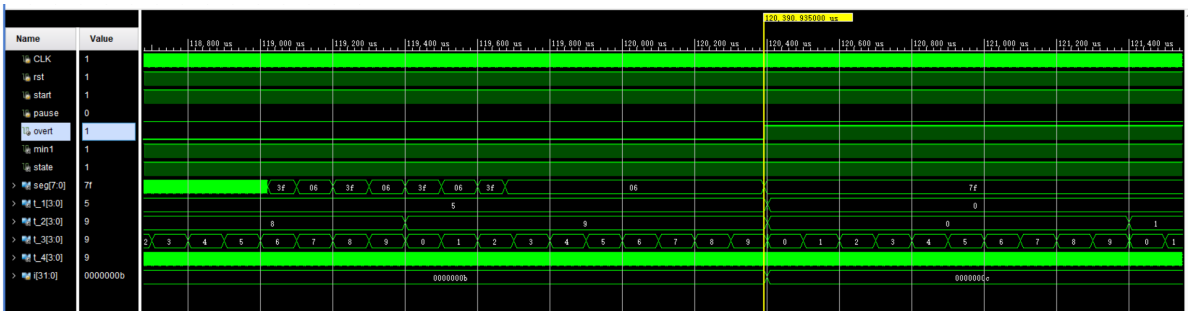
因此，**06->3f->06->3f**对应到数码管即对应十秒、秒、十分秒与百分秒各位的1、0、1、0分别点亮，由于轮流点亮的时间间隔小于0.01秒，数码管成功实现了10.10秒的动态显示。

4.2.2 1分钟时，LED灯变亮



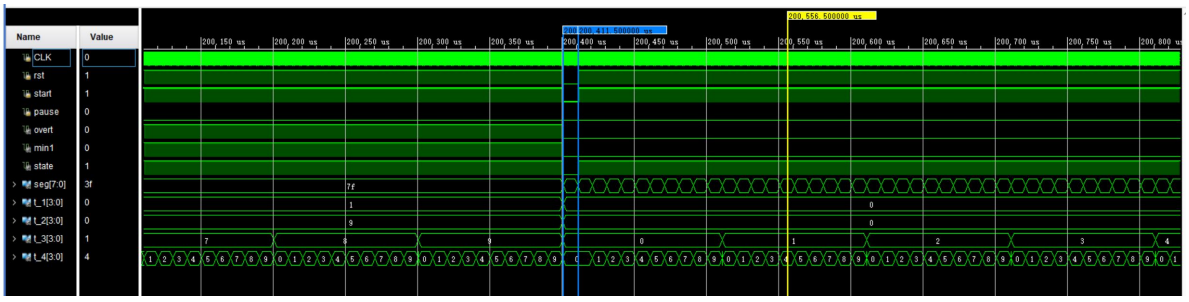
从图中可以看出，1分钟的时候，LED灯变亮，显示分钟位。

4.2.3 2分钟时，超时报警



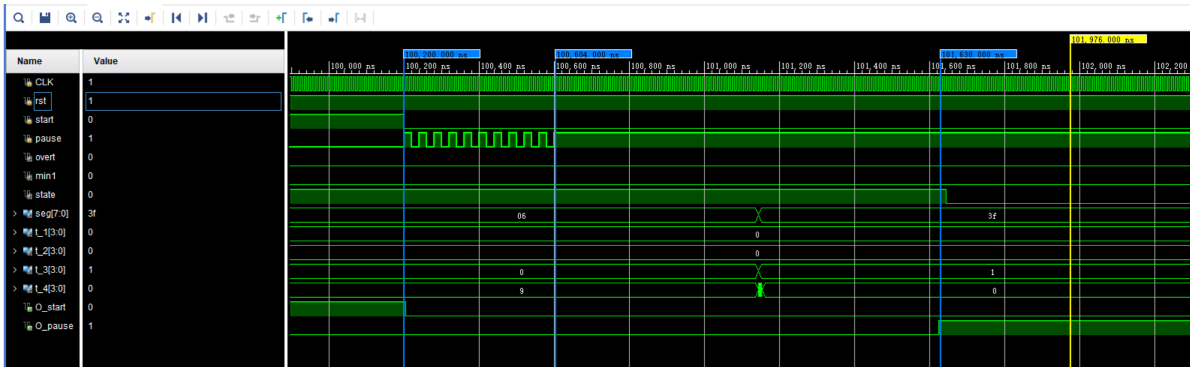
在超过1分59秒99的时候，所有LED灯常亮，数码管各段常亮（即显示为8），以进行超时报警。

4.2.4 从超时到复位，从复位到开始



超时后，复位信号将所有时钟变量归零。开始后，计时器启动，`t_4`由0开始逐渐增大。

4.2.5 按键去抖



在100200ns到100604ns（图中前两条蓝线）引入抖动暂停的信号，可以发现，经过一段时间稳定信号后，电路在第三条蓝线的位置转移至暂停状态，实现了对按键信号的成功去抖。

4. 实验心得

这次实验相较于前两次在难度上有较大提升，在小组合作的过程中我们对verilog语言有了更深的理解。由于我们将问题分解成了多个模块，分块完成了代码的编写，我们也学会了**模块化**的编程思想。

实验的过程中，我们在或大或小的错误中提高了自身**代码编写与调试的能力**。在仿真的时候，我们忘记设置变量的位宽，该变量被4位二进制数直接赋值，并且不断自增1。然而，我们发现该变量实际只在0和1之间跳转，并未实现想象中不断递增的变化：由于该变量的位宽只有1，因此只能容纳1位的二进制数。将位宽设置为`[3:0]`即四位后，便成功实现了自增的显示。

相较前两次实验，本次实验与实践更加贴合，虽然由于疫情原因，无法上板实施，但是我们在编程时也充分考虑了上板后可能出现的**按钮抖动问题**和**多个数码管显示**的问题，并设计了对应的防抖模块和位选信号模块来解决。尤其是位选信号模块，我们了解到多位数码管不能同时显示所有位的值，在同一时刻只能显示一个数码管的值。利用人眼视觉暂留的特性，我们在一个较短的时间间隔内实现了四个数码管的轮流点亮，从而成功实现了数码管的动态显示。这些理论联系实际的问题让我们从代码的视角转到了实际的硬件上，在实践中增强了自身对知识的理解。