EDA 大作业 2: 投币式手机充电仪

院　　系: 自动化系

班　　级: 自 02 班

学生姓名: 彭程

学　　号: 2020011075

目录

1 实验目的 **2**

2 预习任务 **2**

2.1 画出电路总体框图，注明各功能模块以及引脚 2

2.2 画出控制电路的状态转换图 3

3 设计思路 **4**

4 模块功能 **4**

5 状态转换图 **15**

6 仿真波形图 **15**

6.1 键盘模块仿真 15

6.2 控制模块仿真 16

6.3 显示模块仿真 16

7 实验总结 **17**

1 实验目的

1. 学习面向 FPGA 的简单数字系统的设计流程。
2. 掌握 EDA 软件 Quartus II 的原理图输入方式。
3. 熟悉实验装置——实验板，掌握板上外设的工作原理。

2 预习任务

2.1 画出电路总体框图，注明各功能模块以及引脚

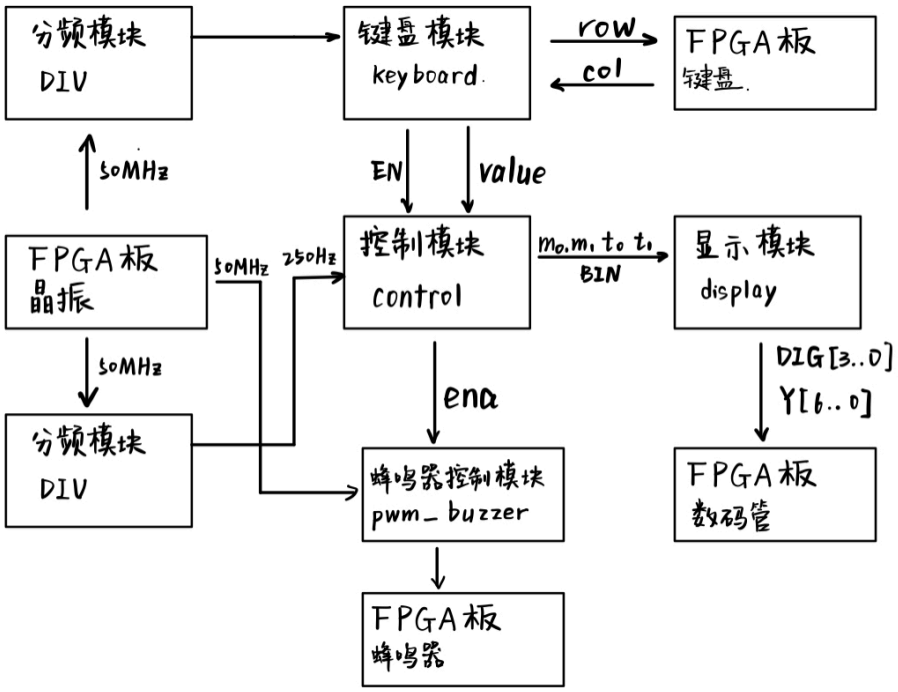


图 1: 电路总体框图

【分频模块】
功能: 将高频率的晶振信号分频至适合电路工作的低频时钟信号
引脚:

input clk: 50MHz 晶振，接 FPGA 板晶振
output clk1: 分频后的信号，接 FPGA 后续模块

【键盘模块】
功能: 识别矩阵键盘按下的位置，输出相应数值
引脚:

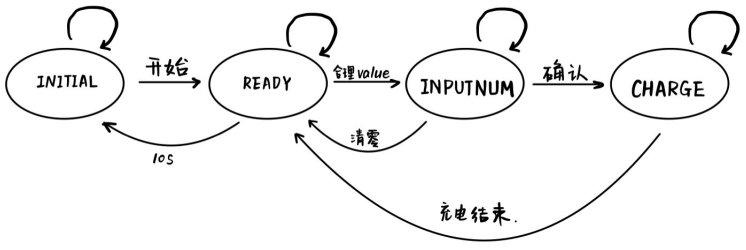
input CLK: 时钟信号
input R: 矩阵键盘列信号
output C: 矩阵键盘行扫描信号，接 FPGA 板矩阵键盘行
output DATA: 按键信息，包括 1-9、开始、清零、充电、无效信息
output EN: 使能信息，标志按键信息是否有效

【控制模块】
功能: 完成任务核心功能，包括状态跳转、清零、充电倒计时、10s 灭灯等
引脚:
input clk: 时钟信号
input value: 键盘值，来自键盘模块
input EN: 键盘值有效标志，来自键盘模块
output m: 投入的钱
output t: 应有的充电时间
output BIN: 是否回到初始状态
output ena: 音乐释放信号

【显示模块】
功能: 驱动四位数码管，将信息显示在数码管上
引脚:
input clk: 时钟信号
input m: 钱，0-20
input t: 时间，0-40
input BIN: 是否回到初始状态
output Y: 位选信号
output DIG: 段选信号

【蜂鸣器控制模块】
功能: 驱动蜂鸣器，在充电结束时放出音乐，在重新输入使停止音乐
引脚:
input clk: 时钟信号
input enable: 音乐释放信号
output buzzer: 音乐信号

2.2 画出控制电路的状态转换图



3 设计思路

电路设计与 FPGA 板通过两部分连接，一是矩阵键盘，也是电路的输入部分，二是数码管，也就是整个设计的输出部分。这两部分需要通过中间的控制模块进行连接，此外为保证电路为同步时序电路，还需设计分频模块。

依据此思路可以自然地划分出各个模块，包括核心功能控制模块 (control.v)、实现由 FPGA 向核心模块输入的矩阵键盘模块 (keyboard.v)、实现由核心模块向 FPGA 输出的显示模块 (display.v) 以及分频模块 (DIV.v)。其中矩阵键盘模块在电子技术实验课上已经设计过，本次实验只对其输出数值进行修改后便可重新使用，实现了其复用。

各模块的设计思路见预习任务中的总体框图和模块功能。

4 模块功能

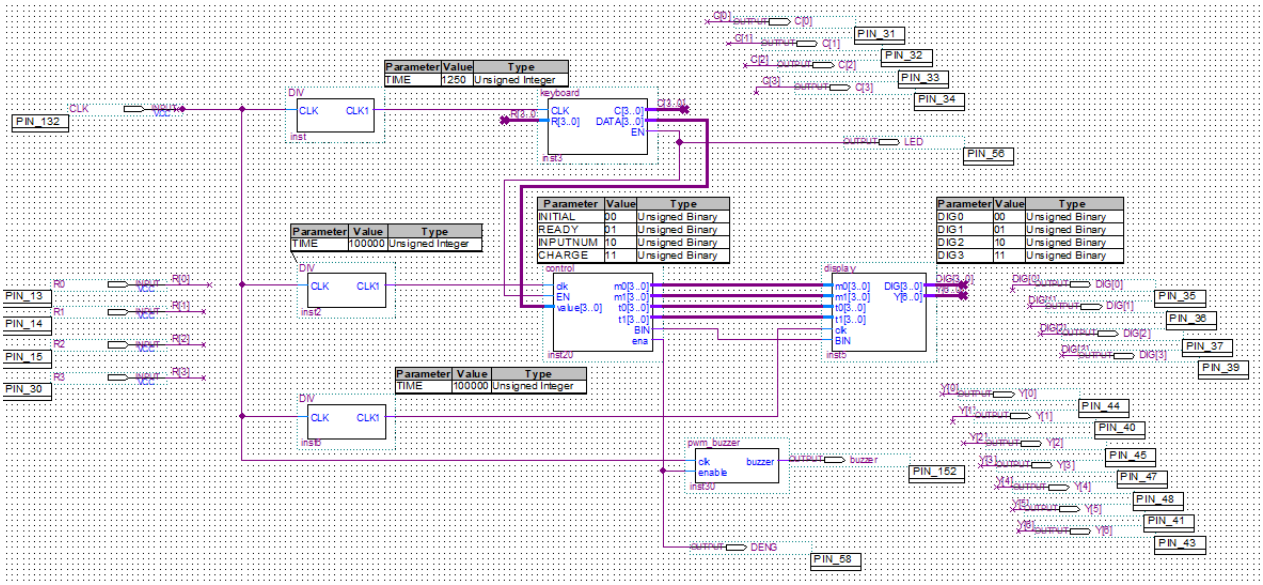


图 2: 顶层电路图

顶层电路图如上图所示。各模块功能以及部分代码说明如下：

【DIV】

分频，将 50MHz 晶振信号分频至 250Hz

```
1  module DIV (CLK , CLK1 ) ;
2   input CLK ;
3   output CLK1 ;
4   reg [24:0] count ;
5   reg CLK1 ;
6   parameter TIME = 24'd1250;
7   always @ ( posedge CLK) begin
```

```
        if (count==TIME) begin
            CLK1 <= ~CLK1 ;
            count <= 0;
       end
       else count <= count +1;
       end
   endmodule
```

【keyboard】

1. 提供扫描信号、同时防抖

```
        always @ (posedge CLK)
        begin
                if (flag)//如果当前有按键按下
                begin
                        if (R == 4'b1111)//有键松开，进行松开消抖
                        begin
                                EN <= 0; //数据无效
                                press <= 0; //未按下
                                if (loose != 6'b111111) //每个CLK信号确认当前状态
                                        loose <= loose + 5'b1;
                                else //连续63个时钟信号都确认发生该转变
                                begin
                                        loose <= 0;
                                        flag <= 0; //确认该按键已经松开
                                end
                        end
                        else//即R!=4'b1111,即又检测到了按键信号
                        begin
                                loose <= 0;//认为有抖动，从0重新计数
                        end
                end

                else//如果当前处于释放状态,flag=0
                begin
                        if (R == 4'b1111)//没有键按下，进行扫描
                        begin
                press <= 0;//有抖动则按下计数器清零
                                A <= A + 2'b1;
                                case (A)
                                        2'b00:C <= 4'b1110;
                                        2'b01:C <= 4'b1101;
                                        2'b10:C <= 4'b1011;
                                        2'b11:C <= 4'b0111;
                                        default:C <= 4'b1110;
                                endcase;
                        end
                        else//有键按下，进行按下消抖，同时使列扫描信号停止在当前列
```

```
38                      begin
39                              loose <= 0;
40                              if (press != 6'b111111)
41                                      press <= press + 5'b1;
42                              else
43                              begin
44                                      press <= 0;         //计数完成，确认按键已经
                                            按下
45                                      EN <= 1;
46                                      flag <= 1;
47                              end
48                      end
49              end
50      end
```

2. 输出按键对应的数值

```
1       always @ (posedge CLK) begin      //读取输出数值
2       if (EN==1) begin  //EN=1表示此时处于按下状态且按键信息有效
3       case(R)
4       4'b0111:case(C)
5               4'b1110:DATA[3:0] = 4'b0100;
6               4'b1101:DATA[3:0] = 4'b0011;
7               4'b1011:DATA[3:0] = 4'b0010;
8               4'b0111:DATA[3:0] = 4'b0001;
9               endcase
10      4'b1011:case(C)
11              4'b1110:DATA[3:0] = 4'b1000;
12              4'b1101:DATA[3:0] = 4'b0111;
13              4'b1011:DATA[3:0] = 4'b0110;
14              4'b0111:DATA[3:0] = 4'b0101;
15              endcase
16      4'b1101:case(C)
17              4'b1110:DATA[3:0] = 4'b1011;
18              4'b1101:DATA[3:0] = 4'b1010;
19              4'b1011:DATA[3:0] = 4'b0000;
20              4'b0111:DATA[3:0] = 4'b1001;
21              endcase
22      4'b1110:case(C)
23              4'b1110:DATA[3:0] = 4'b1111;
24              4'b1101:DATA[3:0] = 4'b1110;
25              4'b1011:DATA[3:0] = 4'b1101;
26              4'b0111:DATA[3:0] = 4'b1100;
27              endcase
28      default: DATA[3:0] = 4'b0000;
29      endcase
30      end
31      else
```

```
32          DATA[3:0]=4'b1111;
33          end
```

【control】

1.10s 内无有效输入灭灯

```
1  //倒计时，实现10s灭灯
2      reg [12:0] time_count;
3      always @ (posedge clk)
4      begin
5              case(current)
6              INITIAL:time_count <= 0;
7              READY:
8              begin
9                      if(EN == 4'h0)
10                             time_count <= time_count + 1;
11                     else
12                             time_count <= 0;
13             end
14             INPUTNUM:time_count <= 0;
15             CHARGE: time_count <= 0;
16             endcase
17     end
```

2. 充电时倒计时

```
1      //充电计时模块
2      reg [8:0] time_count1 = 0;
3      always @ (posedge clk)
4      begin
5          if(current == CHARGE)
6          begin
7              time_count1 <= (time_count1 + 1) % 250;//记录250下为1s，用于充电倒计时
8          end
9          else
10             time_count1 = 0;
11     end
```

3. 长按键处理

```
1      reg ischanged = 0;//若为1，表示有新的输入
2      reg [3:0] oldvalue;
3
4      always @ (posedge clk)
5      begin
6          if(oldvalue == value)
7          begin
8              ischanged <= 0;
9          end
```

```
        else
        begin
            ischanged <= 1;
            oldvalue <= value;
        end
    end
```

4. 充电结束后给音乐模块发送一使能信号使其放 10s 音乐

```
        CHARGE:
        begin
                BIN = 1;
                oldt0 = t0;
                oldt1 = t1;
                if(time_count1 == 249)
                begin
                        t0 = (oldt1 * 10 + oldt0 - 1) % 10;
                        t1 = (oldt1 * 10 + oldt0 - 1) / 10;
                end
                if(t0==0&&t1==0) begin
                ena=1;//ena赋1，控制后级释放音乐
                end
        end
```

5. 根据有效输入给 m、t、BIN 赋值并进行状态转换

```
//状态机
parameter INITIAL = 2'b00;
parameter READY   = 2'b01;
parameter INPUTNUM = 2'b10;
parameter CHARGE  = 2'b11;

reg [1:0] current = INITIAL;
reg [1:0] next    = INITIAL;

//状态方程
always @ (posedge clk)
begin
        current <= next;
end

//驱动方程
always @ *
begin
        case(current)
        INITIAL:
        begin
                if(value == 4'ha)
                begin
```

```
24                          next <= READY;
25                  end
26                  else
27                  begin
28                          next <= INITIAL;
29                  end
30          end
31          READY:
32          begin
33                  if(value != 4'ha && value != 4'hb && value != 4'hc && value != 4'hf
                        && EN==1)
34                  begin
35                          next <= INPUTNUM;
36                  end
37                  else
38                  begin
39                          //time_count==2500说明刚好倒计时10s
40                          if(time_count == 2500)
41                                  next <= INITIAL;
42                          else
43                                  next <= READY;
44                  end
45          end
46          INPUTNUM:
47          begin
48                  if(value == 4'hb)
49                          next <= READY;
50                  else
51                          if(value == 4'hc)
52                                  next <= CHARGE;
53                          else
54                                  next <= INPUTNUM;
55          end
56          CHARGE:
57          begin
58                  //剩余时间为0充电结束，返回INITIAL
59                  if(t1 == 0 && t0 == 0) begin
60                          next <= READY;
61                  end
62                  else
63                          next <= CHARGE;
64          end
65          endcase
66  end
67
68
69  //输出方程
```

```
70  reg [3:0] oldt1;
71  reg [3:0] oldt0;
72  always @ (posedge clk)
73  begin
74          case(current)
75
76                  INITIAL:
77                  begin
78                     ena=0;
79                          m0 = 0;
80                          m1 = 0;
81                          t0 = 0;
82                          t1 = 0;
83                          BIN = 0;
84                  end
85
86                  READY:
87                  begin
88                          m0 = 0;
89                          m1 = 0;
90                          t0 = 0;
91                          t1 = 0;
92                          BIN = 1;
93                  end
94
95                  INPUTNUM:
96                  begin
97                          BIN = 1;
98                          ena=0;
99                          //有效输入1-9
100                         if(ischanged == 1 && value != 4'ha && value != 4'hb &&
                                value != 4'hc && value != 4'hd && value != 4'he &&
                                value != 4'hf)
101                         begin
102                                 m1 = m0;
103                                 m0 = value;
104                         end
105                         else
106                         begin
107                                 m1 = m1;
108                                 m0 = m0;
109                         end
110                         if(m1 >= 2)//充值大于20时置为20
111                         begin
112                                 m1 = 2;
113                                 m0 = 0;
114                         end
```

```
                        t0 = ((m1 * 10 + m0) * 2) % 10;
                        t1 = ((m1 * 10 + m0) * 2) / 10;
                end

                CHARGE:
                begin
                        BIN = 1;
                        oldt0 = t0;
                        oldt1 = t1;
                        if(time_count1 == 249)
                        begin
                                t0 = (oldt1 * 10 + oldt0 - 1) % 10;
                                t1 = (oldt1 * 10 + oldt0 - 1) / 10;
                        end
                        if(t0==0&&t1==0) begin
                        ena=1;
                        end
                end
        endcase
end
```

【display】

1. 实现数码管的位选

```

always @ (current)
begin
        case(current)
        DIG0: next <=DIG1;
        DIG1: next <= DIG2;
        DIG2: next <= DIG3;
        DIG3: next <= DIG0;
        endcase
end
```

2. 实现数码管的段选（只展示了一个数码管的，其余三个同理）

```
always @ (posedge clk)
begin
        case(current)
        DIG0:
        begin
                case(t0)
                //7448
                4'b0000:        Y <= 7'b0111111;
                4'b0001:        Y <= 7'b0000110;
                4'b0010:        Y <= 7'b1011011;
                4'b0011:        Y <= 7'b1001111;
                4'b0100:        Y <= 7'b1100110;
```

```
                4'b0101:        Y <= 7'b1101101;
                4'b0110:        Y <= 7'b1111100;
                4'b0111:        Y <= 7'b0000111;
                4'b1000:        Y <= 7'b1111111;
                4'b1001:        Y <= 7'b1100111;
                default:        Y <= 7'b0000000;
                endcase
                DIG <= 4'b0001;
        end
```

3. 实现数码管的灭灯

```
    if(BIN == 0)//初始状态灭灯
    DIG <= 4'b0000;
```

【pwm_buzzer】

接收使能信号 ena 并放音乐，中间 ena 变化则音乐停止

```
    module pwm_buzzer(
        input   clk,            //时钟输入
    input enable,
        output reg buzzer                       //驱动蜂鸣器
        );
        //定义音符时序周期数
        localparam              M0 = 98800,
                                    M1      = 95600,
                                    M2      = 85150,
                                    M3      = 75850,
                                    M4      = 71600,
                                    M5 = 63750,
                                    M6      = 56800,
                                    M7      = 50600;

        //信号定义
        reg [16:0] cnt0;        //计数每个音符对应的时序周期
        reg [10:0] cnt1;        //计数每个音符重复次数
        reg [5:0] cnt2;     //计数曲谱中音符个数
        reg     [16:0] pre_set          ;       //预装载值
        wire[16:0] pre_div              ;       //占空比
        reg [10:0] cishu                ;       //定义不同音符重复不同次数
        wire [10:0] cishu_div   ;       //音符重复次数占空比
        reg [5 :0] YINFU;       //定义曲谱中音符个数

        //设置音符的个数
        always @(posedge clk ) begin
                        YINFU <= 30;
        end

        //计数每个音符的周期，也就是表示音符的一个周期
```

```
32          always @(posedge clk ) begin
33          if(enable) begin
34                          if(cnt0 == pre_set - 1)
35                                  cnt0 <= 0;
36                          else
37                                  cnt0 <= cnt0 + 1;
38          end
39          else cnt0 <= 0;
40          end
41
42          //计数每个音符重复次数，也就是表示一个音符的响鸣持续时长
43          always @(posedge clk ) begin
44          if(enable) begin
45                          if(cnt0 == pre_set - 1)begin
46                                  if(cnt1 == cishu)
47                                          cnt1 <= 0;
48                                  else
49                                          cnt1 <= cnt1 + 1;
50                          end
51          end
52          else cnt1 <= 0;
53          end
54
55          //计数有多少个音符，也就是曲谱中有共多少个音符
56          always @(posedge clk ) begin
57          if(enable) begin
58                          if(cnt1 == cishu && cnt0 == pre_set - 1) begin
59                                  if(cnt2 < YINFU - 1) begin
60                                          cnt2 <= cnt2 + 1;
61                          end
62                  end
63          end
64          else cnt2 <= 0;
65          end
66
67          //定义音符重复次数
68          always @(*) begin
69                  case(pre_set)
70                          M0:cishu = 181;
71                          M1:cishu = 187;
72                          M2:cishu = 210;
73                          M3:cishu = 235;
74                          M4:cishu = 249;
75                          M5:cishu = 280;
76                          M6:cishu = 314;
77                          M7:cishu = 353;
78                  endcase
```

```
79          end
80
81          //曲谱定义
82          always @(*) begin
83                          case(cnt2)      //歌谱
84                                  0 : pre_set = M5;
85                                  1 : pre_set = M5;
86                  \\剩余谱子省略
87
88                          endcase
89          end
90          assign pre_div = pre_set >> 1;  //除以2
91          assign cishu_div = cishu * 9 / 10;
92
93          //向蜂鸣器输出脉冲
94          always @(posedge clk) begin
95          if(enable) begin
96           if(pre_set != M0) begin
97                          if(cnt1 < cishu_div) begin
98                                  if(cnt0 < pre_div) begin
99                                                  buzzer <= 1'b1;
100                                 end
101                                 else begin
102                                                 buzzer <= 1'b0;
103                                 end
104                         end
105                         else begin
106                                 buzzer <= 1'b0;
107                         end
108                 end
109                 else
110                         buzzer <= 1'b0;
111                 end
112         else buzzer <= 1'b0;
113         end
114 endmodule
```

5 状态转换图

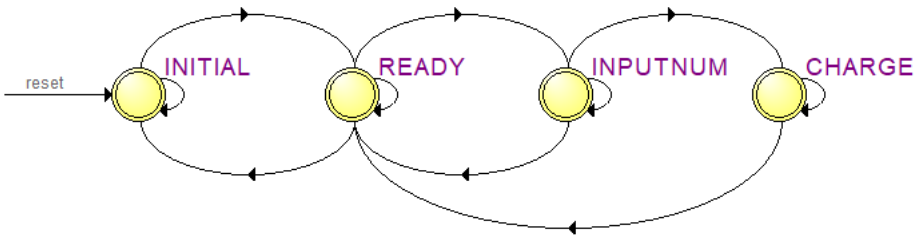
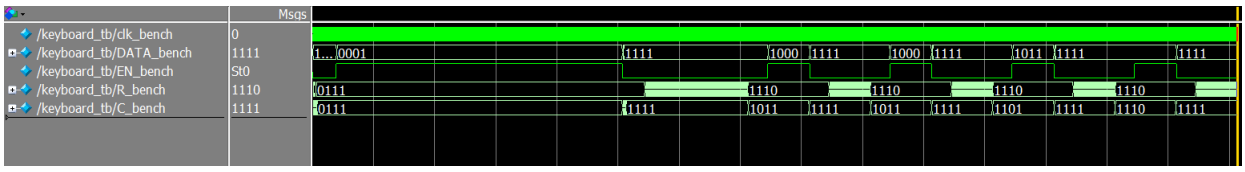


图 3: 控制模块状态转换图

【INITIAL】当且仅当按下“开始”键也就是输入“a”的时候跳入 READY;
【READY】按下数字键也就是输入 1-9 的时候跳入 INPUTNUM，10s 内无有效输入跳入 INITIAL，跳入 READY 时 ena 输出置 0 使音乐停止;
【INPUTNUM】继续按下数字键也就是输入 0-9 的时候仍在 INPUTNUM，按下“清零”键也就是输入“b”的时候跳入 READY，按下“充电”键也就是输入“c”的时候跳入 CHARGE;
【CHARGE】倒计时尚未结束，也就是 t! =0 时仍在 CHARGE, 倒计时结束后，也就是 t==0 时跳回 READY, 同时 ena 输出置 1，释放音乐。

6 仿真波形图

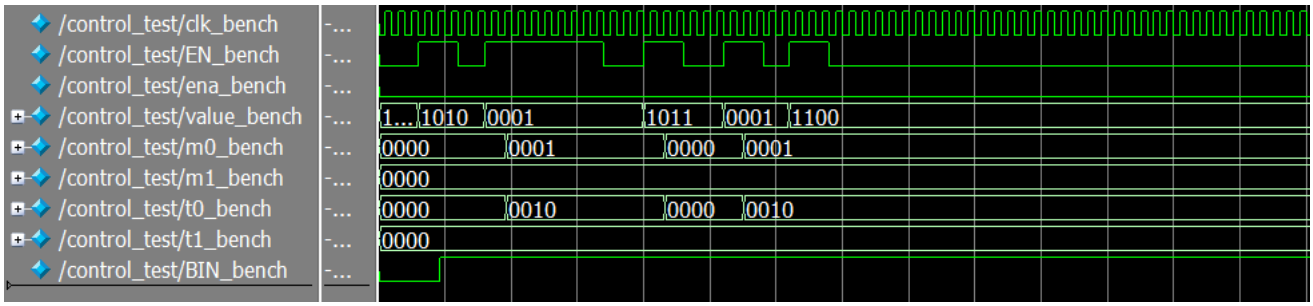
6.1 键盘模块仿真



初始时列扫描信号输出为 0000，检测到列信号存在变化（图中列信号输入变为“0111”）则进行防抖处理，在仿真中我对第一次输出长按键的两端进行了抖动的模拟，可以观察到，在毫秒级的抖动后直到 C_bench 维持稳定了一段时间后，使能端 EN_bench 才变为有效。说明防抖模块有效，同样在松开按键后也同理。档松开按键，EN_bench 变为 0 后，R_bench 出现块状区域代表着扫描信号，当 C_bench 再次变化时才停止扫描，与预期键盘逻辑符合，说明键盘模块仿真正确。

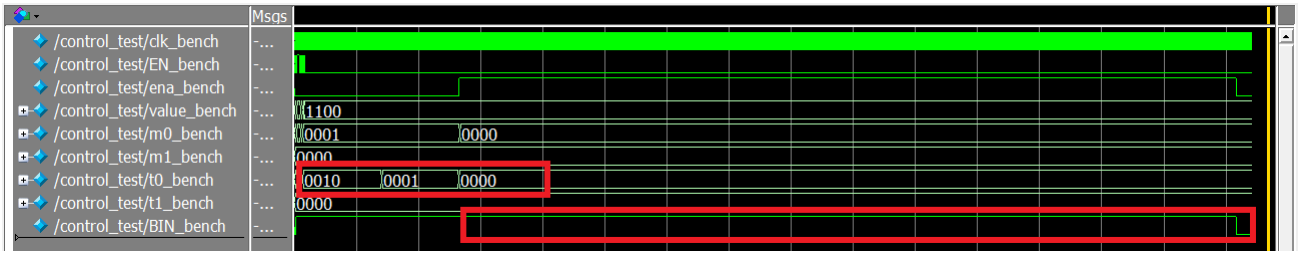
6.2 控制模块仿真

根据实验任务对控制模块进行了全流程仿真：
首先是局部放大图：



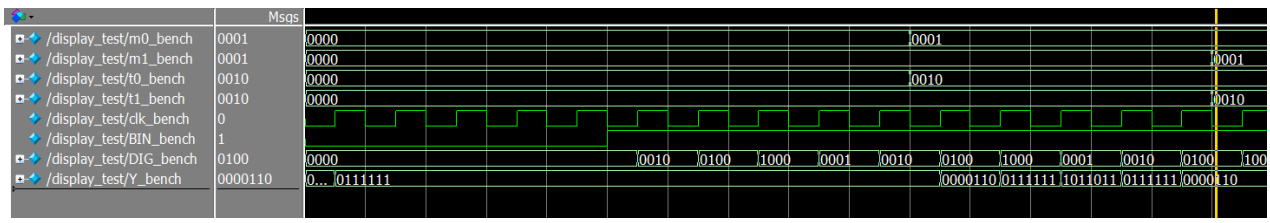
图中 value_bench 和 EN_bench 的输出依次代表开始、长按 1、清零、短按 1、确认。刚上电时默认为 INITIAL 状态，BIN 为 0。当按下开始键（图中输入 “1010”），BIN 变为 1。长按数字键（图中输入 “0001” 代表按下按键 1），输出相应的金额和时间（如图中 m0=1，t0=2）；按下清零键（图中输入 “1011”），金额时间均清零；再按下数字键（图中输入 “0001”）重新输出金额和时间。可以看到相关 t 和 m 的输出符合我们预期的需求。

其次是整体仿真图：



图中框选的两处分别为倒计时两秒后清零和清零后 10s 灭灯，可以看到仿真结果符合我们的预期。

6.3 显示模块仿真



BIN 为 0 时 DIG 输出 “0000” 数码管全灭，BIN 为 1 后位选端开始循环扫描四位数码管，数码管亮起。根据输入的 m1、m0、t1、t0 的值，该模块将按照数码管位选端地址将正确的数字显示在数码管上。如图中 m1=0,m0=1,t1=0,t0=2 时，DIG=0001 是右边第一位数码管，显示 2，即数码管段选端 a-g 分别为 1101101，以此类推，波形无误。

7 实验总结

遇到的问题及解决方法

1. 键盘模块防抖一直无效，出现奇怪的静电问题，后经过一番研究发现，FPGA 板说明文件中，键盘的行列接法和实际情况相反，将行列颠倒后问题消失。
2. verilog 代码编写不太熟悉，通过自学基本语法大致掌握了基本编写方法。
3. 一开始蜂鸣器有杂音，后发现是设置出现错位，误将代码参数写错，导致在非音乐时间输出高电平。

收获

1. 掌握了用 verilog 设计状态机电路的方法流程。
2. 学会了 Quartus II 的状态转换图查看方法。
3. 掌握了蜂鸣器放音乐的原理及控制方法。
4. 掌握了矩阵键盘的原理和控制方法。