

EDA 大作业二 投币式手机充电仪

实验报告

姓名：____ 赵文亮 _____

学号：____ 2016011452 _____

班级：____ 自 64 _____

日期：____ 2017 年 12 月 30 日 _____

目录

1	实验目的	1
2	预习任务	1
3	设计思路	2
4	顶层电路图	9
5	状态转换图及其说明	9
6	仿真波形图及其分析说明	11
6.1	分频器仿真	11
6.2	输入模块仿真	11
6.3	控制模块仿真波形	13
6.4	显示模块仿真波形	14
7	设计和调试中遇到的问题及其解决方法	14

1 实验目的

1. 学习自顶向下、分模块的数字系统分析、设计与调试方法。
2. 编写测试文件对设计电路进行仿真验证。
3. 掌握规范使用硬件描述语言描述状态机电路的方法。

2 预习任务

1. 阅读并分析任务要求，画出电路的总体框图，注明各功能模块及其引脚

通过分析可知，本次实验的电路总体框图如图 1 所示。其中红色部分的模块由外设提供，蓝色部分

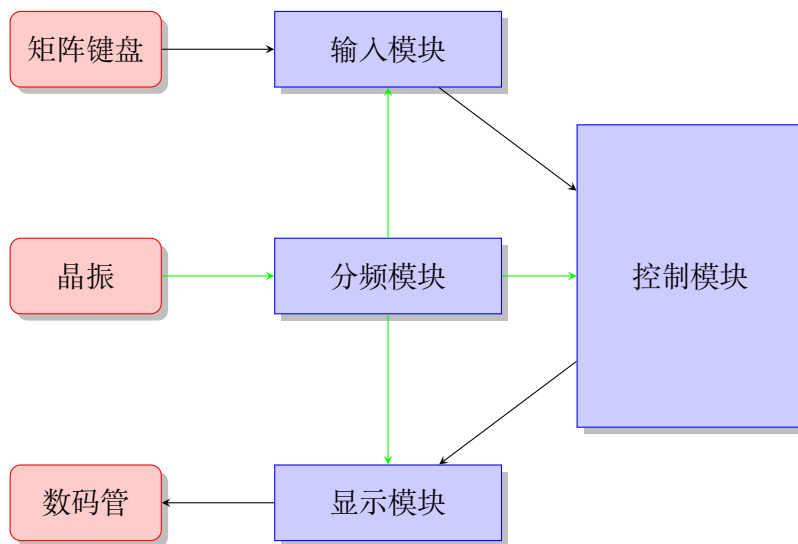


图 1: 模块示意图

是本次实验的设计内容。下面分模块进行说明。

分频模块 用于将晶振的 50MHz 频率降下来，为其他电路提供时钟信号。有 CLK、CLK1 两个引脚，从晶振获取 CLK 信号，经过分频之后得到 CLK1 输出。本次实验中分频器设计为 2500 分频。

输入模块 用于读取矩阵键盘从而获得数据，并传递给控制模块。输入变量为时钟信号 CLK，矩阵键盘列线 C；输出变量为矩阵键盘行线 R、数据 DATA、数据使能控制 EN。这里采用 EN 的原因是，DATA 不一定在任何时候都有效，如果没有这一设置，控制模块会读入无效的输入。输入模块的关键是对矩阵键盘的输入做好防抖处理，以得到正确的 DATA 数据。

控制模块 由于实现电路的主要逻辑。输入变量为从输入模块获得的按键数据 DATA、数据使能 EN、时钟信号 CLK；输出变量为投币金额 M 和充电时间 T。控制模块根据按键数据在三个不同的状态之间跳转，并实施将当前应该显示的钱数与充电时间传递给后续的显示模块。

显示模块 显示模块由数据选择器和 BCD—七段显示译码器组成。输入变量为时钟 CLK、钱数 M、充电时间 T；输出变量为数码管的段选信号 SEG 和位选信号 DIG。

2. 根据任务要求画出控制电路的状态转换图

控制电路的状态转换图如图 2 所示。

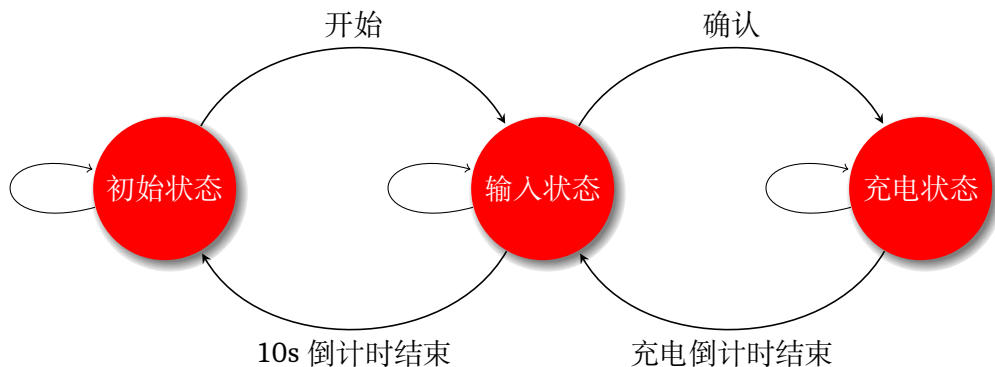


图 2: 控制电路的状态转换图

共有三个状态：初始状态、输入状态、充电状态。状态之间的转换关系如下：

- 初始状态 → 输入状态：按下“开始”键。这时数码管显示为“0000”，为开始状态，可以输入投币金额。
- 输入状态 → 充电状态：按下“确认”键。
- 充电状态 → 输入状态：充电倒计时结束。这时数码管显示为“0000”，即为开始状态。
- 输入状态 → 初始状态：
 - (1) 充电倒计时结束回到输入状态（开始状态），若 10s 无按键，则回到初始状态。
 - (2) 输入状态中按下“清零”键后，若 10s 无按键，则回到初始状态。
- 初始状态 → 初始状态：初始状态下没有按下“开始”键，则保持状态。
- 输入状态 → 输入状态：输入状态下没有按下“确认”键，且没有进入倒计时或倒计时未结束，则保持状态。
- 充电状态 → 充电状态：保持状态直到充电倒计时结束。

3 设计思路

最主要的设计思路是自顶向下，分模块进行设计。每次将一个模块调好后，便将其封装，便于调用。下面阐述每一个子模块的设计思路。

分频模块 将来自晶振的时钟信号作为输入，输出分频结果的信号 CLK1。内部使用 count 变量来计数，当 CLK 上升沿到达的时候，如果 count 未达到 1250，则将 count 加 1；否则将 count 置零，并将 CLK1 翻转。这样，便生成了 2500 分频的时钟信号 CLK1。分频后的信号为 20000Hz。这个频率似乎有点高，但事实上，频率越高，系统的灵敏度越好。在不影响系统工作的情况下（例如本实验中需保证数码管无重影），频率取高一些比较好。事实证明，在这个频率下系统完全可以正常工作。

输入模块 输入模块从矩阵键盘读入按键，并传给控制模块。输出数据 DATA 为当前按下的键的编码，数据使能 EN 用来决定 DATA 是否有效。EN 的设置为控制模块判断当前数据是否有效提供了方便。控制模块检测 EN 的上升沿，并在此时读入数据。输入模块通过对行扫描，依次将每个行线置零，并读取列线的值。如果 4 条列线中的某一个值为 0，则停止扫描行线。如果防抖条件判断成功，则将 EN 置为 1；如果 4 条列线全为 1，则将 EN 置为 0。当 EN 由 0 变为 1 时，根据当前的行列值为 DATA 赋值。

```
1      always @ (posedge CLK)
2      begin
3          if (flag)//press, flag == 1
4          begin
5              if (C == 4'b1111)//release now
6              begin
7                  EN <= 0;
8                  press <= 0;
9                  if (loose != 5'b11111)//
10                     loose <= loose + 5'b1;
11                 else
12                 begin
13                     loose <= 0;
14                     flag <= 0;
15                 end
16             end
17             else
18             begin
19                 loose <= 0;
20             end
21         end
22         else//release, flag == 0
23         begin
24             if (C == 4'b1111)//not press, scan the column
25             begin
26                 A <= A + 2'b1;
27                 case (A)
```

```

28         2'b00 :
29             R <= 4'b1110;
30         2'b01 :
31             R <= 4'b1101;
32         2'b10 :
33             R <= 4'b1011;
34         2'b11 :
35             R <= 4'b0111;
36         default :
37             R <= 4'b1110;
38     endcase;
39 end
40 else//press any key
41 begin
42     loose <= 0;
43     if (press != 5'b11111)
44         press <= press + 5'b1;
45     else
46     begin
47         press <= 0;
48         EN <= 1;
49         flag <= 1;
50     end
51 end
52 end
53 end

```

下面说明防抖的原理。使用 **flag** 来标志当前状态为按下还是释放。如果当前状态为按下，则需要对释放进行防抖：当检测到释放时（所有列线电平均为高），对 **loose** 变量进行计数，记到 31 时给出释放信号。如果期间有按键信号，则将 **loose** 变量清零。如果当前状态为释放，则需要对按下进行防抖：同理使用 **press** 变量对计数，只有连续的 31 个时钟上升沿都有按键（列线不全为高电平），才将 **flag** 置为 1，并将按键使能信号 **EN** 置为 1。当给出按键信号时，**DATA** 变量读取键盘的数据。并以 4 位二进制数的形式保存按键的编码。

控制模块 控制模块根据输入模块提供的按键信息给出当前的投币 **M** 和时间 **T**。首先，我在一个进程中将 **DATA** 转换成数字 **NUM**。若按键对应数字为 0~9，则 **NUM** 即为相应数字；否则若按键为“开始”、“清零”、“确认”或无效按键，则将 **NUM** 置为 4'hf 用于标志。我设计的控制电路具有循环输入的功能。例如，输入“1”，则数码管显示“0102”；再输入“1”，则数码管显示“1122”；再输入“2”，则数码管显示“1224”；再输入“3”，则数码管显示“2040”（钱数 23 超过了 20）。为了实现这个功能，我使用 **M0** 来保

存 M 的个位，使用 next_M 来计算下一时刻应该显示的 M 数值：

```
1    wire [3:0] M0;
2    wire [7:0] next_M;
3    assign M0 = M % 10;
4    assign next_M = M0 * 10 + NUM;
```

为了读取到有效的数据，我使用检测 EN 的上升沿的办法。结构良好的时序电路的模块的 always 的敏感变量中尽量只有一个 CLK，最多加一个异步置零，所以我使用时钟来检测 EN 的上升沿：我利用 last_EN 变量保存上一个 CLK 上升沿到来时的 EN 值，并与这一时刻的 EN 比较。如果两者不同且当前 EN 为 1，则说明 EN 上升沿到来。为了方便，我利用 keyPress 来表示这一判断值：

```
1    wire keyPress = (last_EN != EN && EN);
```

这里使用的是 EN 上升沿有效。如果改成

```
1    wire keyPress = (last_EN != EN && !EN);
```

即可实现下降沿有效，两种按键效果有所不同。我使用常量 S0、S1、S2 分别表示三个状态。下面对主进程分块分析。

```
1    if (current_state == S0)
2    begin
3        if (keyPress)
4        begin
5            if (START)
6            begin
7                current_state <= S1;
8                M <= 0;
9                T <= 0;
10           end
11        else
12            current_state <= S0;
13        end
14    else
15    begin
16        current_state <= S0;
17        M <= 8'hff;
18        T <= 8'hff;
19    end
20 end
```

当前状态是 S0 时，如果检测到按键，且为“开始”按键，则跳入 S1 状态，并将 M 和 T 置零。我使用 START 变量用于判断当前的按键是否为开始按键：

```
1      assign START = (DATA == 4'b1101);
```

同理有 RST、CONFIRM 分别代表“清零”和“确认”。如果没有检测到按键，或按键不是开始键，则保持 S0 状态。这里将 M 和 T 置为 8'hff 作为无效值，后续的显示模块识别到 8'hff 会做出灭灯操作。

S1 部分比较复杂。主要的功能有输入数据、确认、清零和倒计时。

- 输入数据部分：

```
1      else if (NUM <10)
2      begin
3          timer <= 0;
4          wait_count <= 0;
5          if (next_M < 21)//money is less than 21
6          begin
7              M <= next_M;
8              T <= next_M * 2;
9          end
10         else
11         begin
12             M <= 8'd20;
13             T <= 8'd40;
14         end
15     end
```

NUM<10 表示当前按键为数字键。这时将定时器 timer 关闭，并判断下一时刻应该显示的钱数是否小于 21。如果小于 21 则将 M 置为 next_M，否则将 M 置为 20，T 对应置为 M 的 2 倍。

- 确认部分

```
1      if (CONFIRM)
2      begin
3          current_state <= S2;
4      end
```

比较简单。直接跳转到 S2 状态，不再赘述。

- 清零部分：

```
1      if (RST)
2      begin
3          M <= 8'b0;
4          T <= 8'b0;
```



```

5         timer <= 1;
6         wait_count <= 0;
7     end

```

按下清零时，将 M 和 T 均置零，并开启定时器（10s 计时），重置计时变量。

- 倒计时部分：

```

1         if (timer)
2         begin
3             if (wait_count < WAIT_TIME)
4                 wait_count <= wait_count + 18'b1;
5             else
6                 begin
7                     wait_count <= WAIT_TIME + 1;
8                     timer <= 0;
9                 end
10            end

```

如果定时器已经开启，则使用 wait_count 变量进行计时。其中 WAIT_TIME 为一个常数：

```

1     /*after finishing charged or after reset,
2     delay for 10s ,
3     WAIT_TIME = 20000 * realtime(s)*/
4     parameter WAIT_TIME = 200000;

```

由于给控制模块的时钟是经分频器 2500 分频后的时钟，故计数 20000 为一秒。这里 WAIT_TIME 的取值保证了倒计时 10s 之后结束。倒计时结束后，将 wait_count 置为 WAIT_TIME + 1 作为标志，并跳转到 S0 状态：

```

1         else if (current_state == S1 && wait_count == WAIT_TIME + 1)
2         begin
3             current_state <= S0;
4             wait_count <= 0;
5         end

```

需要指出的是，对于倒计时功能的设计我完全参考了 EDA2 大作业的作业要求和验收说明，即“计时结束或清零后，若无其他操作 10 秒后跳回‘初始状态’”¹。也就是说，我只在充电倒计时结束或按下清零键之后才开启定时器。但是验收过程中我发现我的理解和助教的理解似乎有所不同。助教的理解是在开始状态下无操作 10s 后也应该回到初始状态。事实上，这两种理解只是在功能上有所

¹见文件 EDA 作业二实验验收说明

不同，在实现难度上并没有本质性的差别，我只需在开始状态下将定时器打开（timer 置为 1）即可实现开始状态下无操作 10s 后回到初始状态的功能。

S2 为充电状态。在此状态下，投币金额保持不变，而充电时间每隔 SEC 减小 1。其中 SEC 是一个常数：

```
1      /* When charging, SEC denotes the number of CLKs in a second
2         here the real time is 0.1 second for saving time
3         SEC = 20000 * realtime(s)
4         */
5      parameter SEC = 2000;
```

SEC 用来表示倒计时中 1s 的时间内有 CLK 的上升沿的个数。为了节省验收时间，我在这里设置 SEC=2000，即每隔 0.1s 充电时间减 1。该值可以根据实际需要设定。

S2 状态下充电倒计时的实现如下：

```
1      if (time_count < SEC)
2          time_count <= time_count + 15'b1;
3      else
4          time_count <= 0;
5      if (time_count == SEC - 1)
6      begin
7          if (T != 0)
8              begin
9                  T <= T - 8'b1;
10             end
11             if (T == 1)
12                 begin
13                     M <= 0;
14                 end
15         end
```

当 time_count 计满一个 SEC 后，且 T 不为 0，则 T 减 1。若 T 已经是 1，则 T 减 1 的同时 M 也要减 1，从而保证同时归零。另外，当计时到零时，需要从 S2 跳转回 S1，并开启、重置计时器：

```
1      if (current_state == S2 && M == 0)
2      begin
3          current_state <= S1;
4          timer <= 1;
5          wait_count <= 0;
6      end
```

显示模块 显示模块分为数据选择器和 BCD—七段显示译码器两个子模块。数据选择器的输入变量有投币金额 M、充电时间 T、时钟信号 CLK。其输出信号包括数据输出四位二进制 DOUT 和数码管位选信号 SOUT。SOUT 遍历 “0001”、“0010”、“0100”、“1000” 四种取值，并分别将 DOUT 置成对应的 T/10、T%10、M/10、M%10。特别的，若 M 为 4'hff，则代表当前为 S0 状态，应该灭灯，此时 DOUT 给出 1111 信号。

BCD—七段显示译码器将输入的四位二进制数（若为 0~9）转成数码管的段选信号。对于大于 9 的输入，输出的段选信号为 7'b0。这样就保证了当状态为 S0 时所有数码管全灭。

4 顶层电路图

顶层电路图如图 3 所示。与图 1 类似，顶层电路主要包括四个模块。左上方为分频模块，用于将晶振

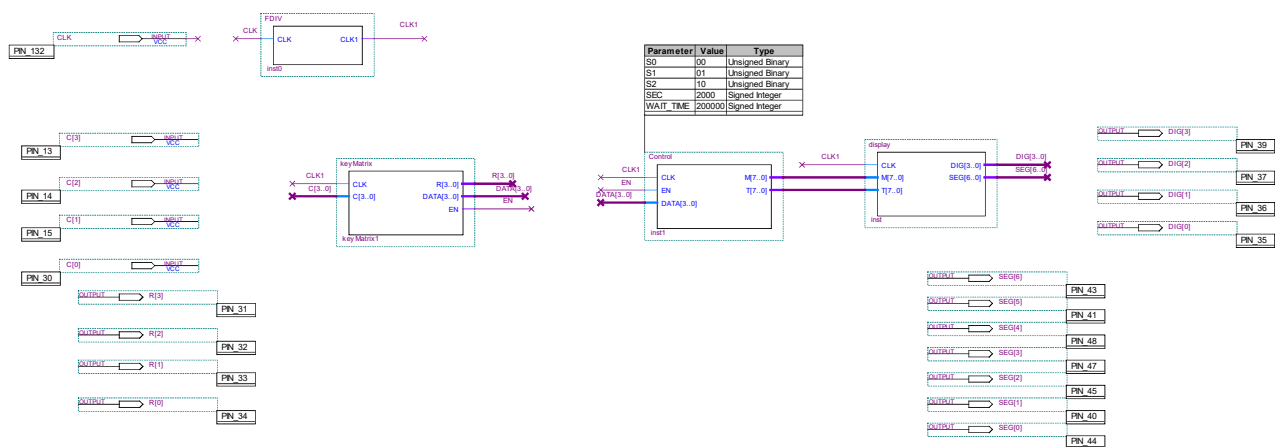


图 3: 顶层电路图

的 50MHz 降到 20kHz；其下方为矩阵键盘输入模块，用于从矩阵键盘中获得数据传给控制模块；中间是控制模块，用于实现电路的主要控制功能；最右边是显示模块，用于将投币金额和充电时间显示在数码管上。

顶层模块的输入输出整体上有三种：

- 时钟输入。位于原理图左上角，将其引脚分配到 FPGA 板上的晶振上。
- 矩阵键盘输入输出。包括列输出和行输入。列输出用于依次将列线置为低电平扫描，行输入用于读取行线状态。将其引脚分配到矩阵键盘对应的行线和列线上。
- 数码管输出。包括位选输出和段选输出。将其连接到数码管的段选和位选控制端。

5 状态转换图及其说明

状态转换图如图 2 所示。利用 Quartus 的 State Machine Viewer 生成的状态转换图如图 4 所示。下面对状态转换图进行说明。S0 是初始状态，S1 是输入状态，S2 是充电状态。各种状态之间的转换关系在预

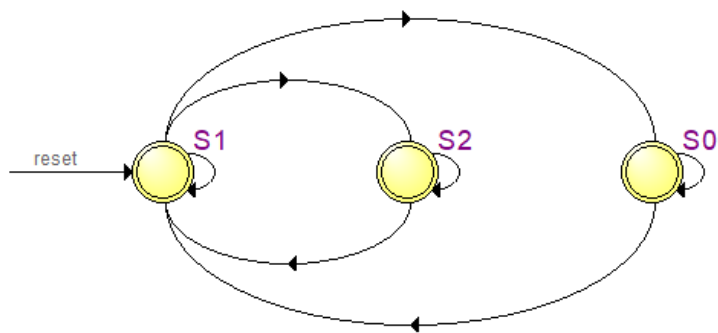


图 4: Quartus 生成的状态转换图

习任务²中已经体现。再次简要整理如下：

- S0→S1：按下“开始”键。
- S1→S2：按下“确认”键。
- S2→S1：充电倒计时结束。
- S1→S0：
 1. 充电倒计时结束回到输入状态（开始状态），若 10s 无按键，则回到初始状态。
 2. 输入状态中按下“清零”键后，若 10s 无按键，则回到初始状态。
- S0→S0：未按下“开始”键。
- S1→ 输入状态：输入状态下没有按下“确认”键，且没有进入倒计时或倒计时未结束，则保持状态。
- 充电状态 → 充电状态：保持状态直到充电倒计时结束。

需要指出的是，State Machine Viewer 生成的状态转换图中有一个指向 S1 的 reset 箭头。这一箭头表示电路初始会进入到 S1 状态。但事实上我已经在控制模块中将状态初始化为 S0：

```

1  initial
2  begin
3      current_state = S0;
4      M = 8'hff;
5      T = 8'hff;
6  end
  
```

²见第 2 页第 2 节

这可能和 Quartus 综合电路的方式有关，最终得到的电路会在初始时刻进入 S1。为了解决这一问题，我在 S1 的状态中设置了如下代码保证电路自启动到 S0：

```
1      if (M == 8'hff && T==8'hff)
2          current_state <= S0;
```

由于刚上电的时候已经将 M 和 T 初始化为 8'hff，如果当前还处于 S1 状态，则应该马上跳转到 S0 状态。这样就实现了上电进入初始状态，且数码管全灭。

6 仿真波形图及其分析说明

6.1 分频器仿真

分频器的仿真波形如图 5 所示。我在测试文件中设置了时钟 CLK 的周期为 20ns，从仿真波形可以看

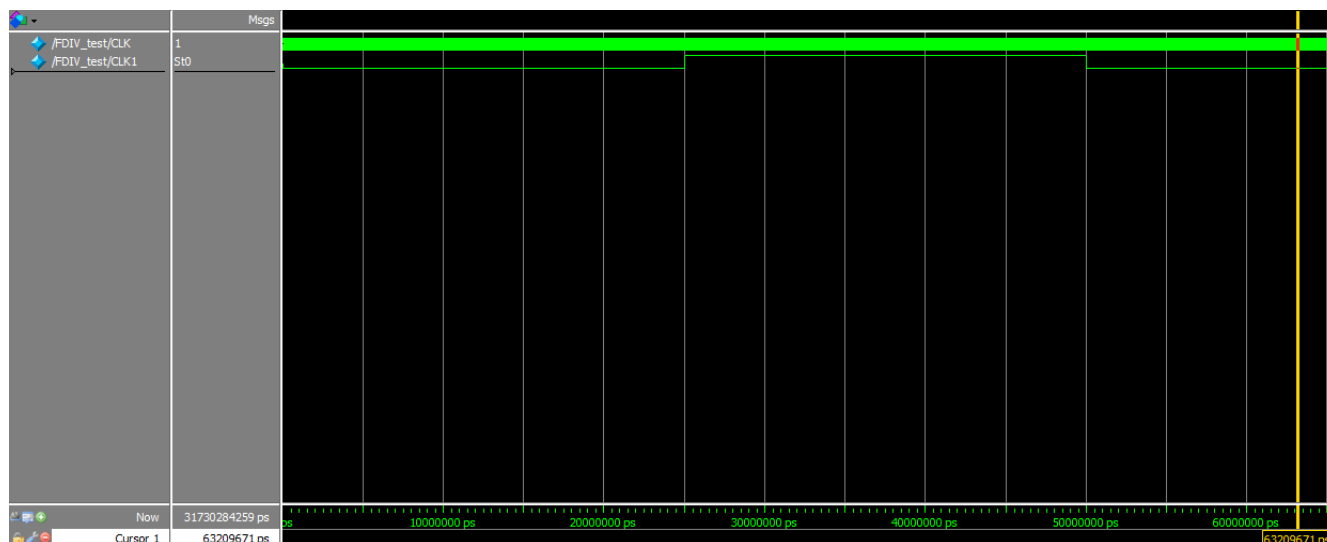


图 5: 分频器仿真波形

出，经过分频后，CLK1 的周期为 50000ns，完全符合预期的 2500 分频。

6.2 输入模块仿真

输入模块的仿真波形如图 6 所示。DATA 表示输出的数据，EN 表示输出数据是否有效，R 为行线扫描信号，R_index 用于表示当前扫描的行线序号，C 为读入的列线数据。en 为测试文件中用于表示按键是否按下（用于控制抖动）的变量。press_key 为测试文件中的变量，表示当前按下的键的二进制编码。k 为测试文件中的循环变量。

图中展示了长按键和防抖的情况。第一个按键（press_key=0001）为长按键，每个按键前后都做了抖动处理。从图中可以看出，当有按键按下时，且已经过了抖动区间一段时间后，EN 的值才被置为 1；而当释放按键时，只要开始抖动，EN 就被置为 0，表示数据无效。事实上，我对防抖的处理对于按键是否为长按键并无区别，故自然可以满足长按键的需求。我使用变量 k 进行循环，遍历了所有按键可能。可以看到对于每一种输入，输入模块都可以有效地读取数据。



图 6: 输入模块仿真波形

下面详细地分析防抖的波形，将波形放大如图 7。

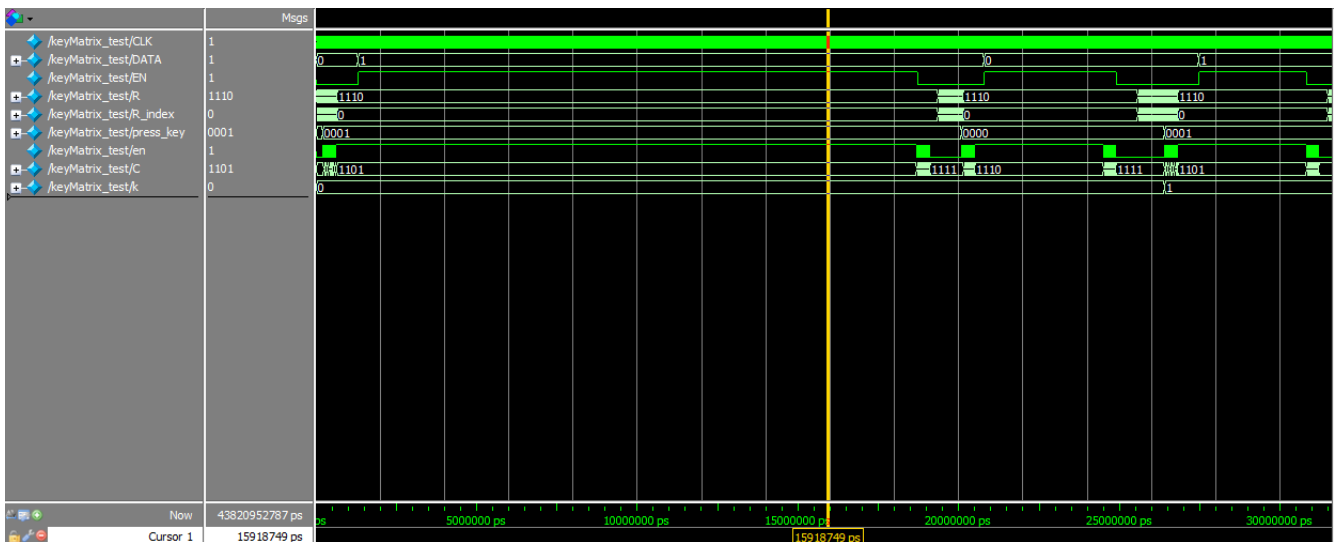


图 7: 防抖仿真波形

不妨观察最长的按键，即 $\text{press_key}=1$ 。可以看到，在没有按键的时候，行线在不断扫描；按键信号开始时的抖动中会出现 $C=1111$ 的情况，而只要当前状态为未按下，且 $C=1111$ ，行线就会不断扫描，故在开始阶段直到按键抖动结束的时间内， R 都在扫描状态。当按键信号稳定后， R 停止扫描，经过一段时间后，判断案件信号稳定， EN 被置为 1，同时读入 $DATA$ 数据。在释放按键时，只要一开始抖动，就将 EN 置为 0，完全符合设计的思路。

需要指出的是，我的输入模块的仿真波形如此符合实际得益于测试文件的编写。我的理解是，输入模块的测试文件的作用就是模拟矩阵键盘。于是我需要根据矩阵键盘的实际情况，给出对应的 C 值。我经过分析，得出列线信号和三个因素有关：

- 按键编号。即哪个键被按下，使用 press_key 变量来表示。

- 行线扫描信号。即当前哪一行被置零。我使用 `R_index` 来表示当前被置零的行线编号。`R_index` 的值由行扫描信号 `R` 获得。
- 是否按下。这就是抖动的来源。实际情况中，由于机械按键的不稳定，按下一个键时会发生抖动，即被认为在一段时间内不断重复“按下——释放”，直到按键稳定。我用变量 `en` 来表示是否按下，故只需要让 `en` 在一段时间内不断变化即可模拟出抖动的效果。

于是，我根据上述三个变量推导出实际情况下列线对应的值，作为矩阵键盘的输入，得到了较好的仿真波形。

6.3 控制模块仿真波形

控制模块的仿真波形如图 8 所示。为了便于仿真，我将倒计时计数变量改为较小的值。这里我令 `SEC=1`,

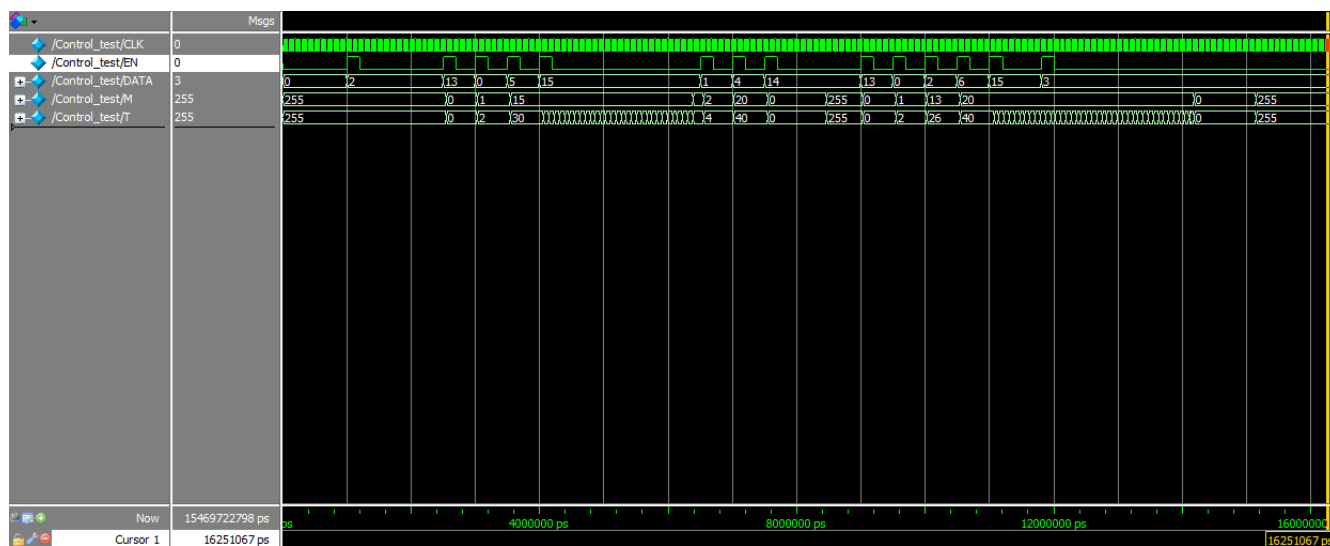


图 8: 控制模块仿真波形

`WAIT_TIME=20`。为了便于观察变量的数值，我将 `DATA`、`M`、`T` 用无符号整数来表示。

首先为初始状态，这时 `M` 和 `T` 均为 255 (8'hff)，对应数码管全灭。在初始状态下，我给出一个按键“2”，输出没有任何变化；接着按下“13”（开始），这时 `M` 和 `T` 全被置为 0。下面依次按下“0”、“5”，对应的数字为“1”、“5”，可以看到 `M` 依次为 1、15，`T` 为 `M` 的二倍。下面按下“15”（确认），则开始充电倒计时，`T` 递减而 `M` 保持不变，`T` 回零的时候 `M` 也回零。

这时回到了开始状态，在倒计时未结束时，进行了新一轮的按键。这次依次按下了“1”、“4”，对应数字为“2”、“3”。由于 $23 > 20$ ，故 `M` 会显示 20 而 `T` 会显示 40。下面按下了“14”（清零），`M` 和 `T` 都被置零，经过 `WAIT_TIME` 个 `CLK` 上升沿倒计时后，回到初始状态，`M` 和 `T` 又被置为 255。

重新开始新一轮按键。按下开始后，依次按下“0”、“2”、“6”，分别对应了数字“1”、“3”、“6”。从这里可以看出我的循环输入功能。`M` 的值依次为：“1”、“13”、“20”（因为 $36 > 20$ ）。按下确认之后，进入充电倒计时，`T` 递减，`M` 不变。在充电倒计时期间，我按下了“3”号键，可见对输出没有任何影响。`T` 充电倒计时回零时，`M` 也同时归零，再经过 `WAIT_TIME` 个 `CLK` 上升沿后，回到了初始状态。

上述波形完整地展示了各种情况，输出结果均符合预期。

6.4 显示模块仿真波形

显示模块仿真波形如图 9 所示。图中遍历了 M 和 T 的各种取值可能。SED 和 DIG 分别是四位数码管

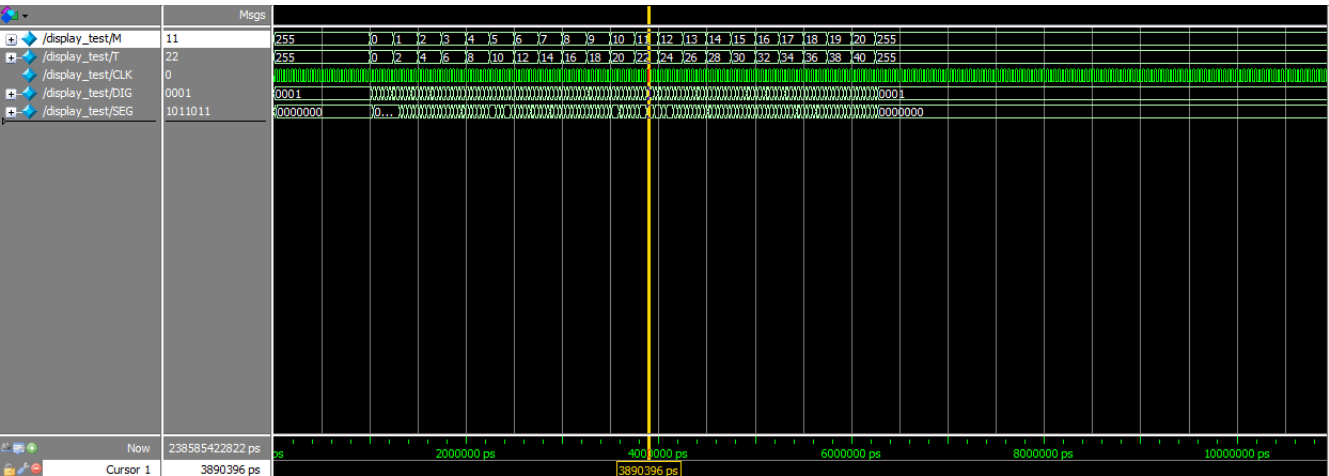


图 9: 显示模块仿真波形

的段选信号和位选信号。例如图中标光所示位置，M=11，T=22，DIG=0001，说明第四位数码管被选中，应该显示 T 的个位“2”。此时段选信号为“1011011”（从左到右依次为 gfedcba），结合七段数码管的段选编码可知该信号对应“2”，符合要求。

另外，在 M 和 T 为 255（8'hff）时，段选信号的每一位都为 0，对应着初始状态下的数码管全灭。

7 设计和调试中遇到的问题及其解决方法

1. 输入模块的设计及其防抖处理。一开始,我按照 EDA 讲座二中 PPT 的原理图设计输入模块。如图 10 所示。我按照 PPT 中讲解的列线依次置低电平,检测行线状态设计输入模块。但是当我准备进行引脚

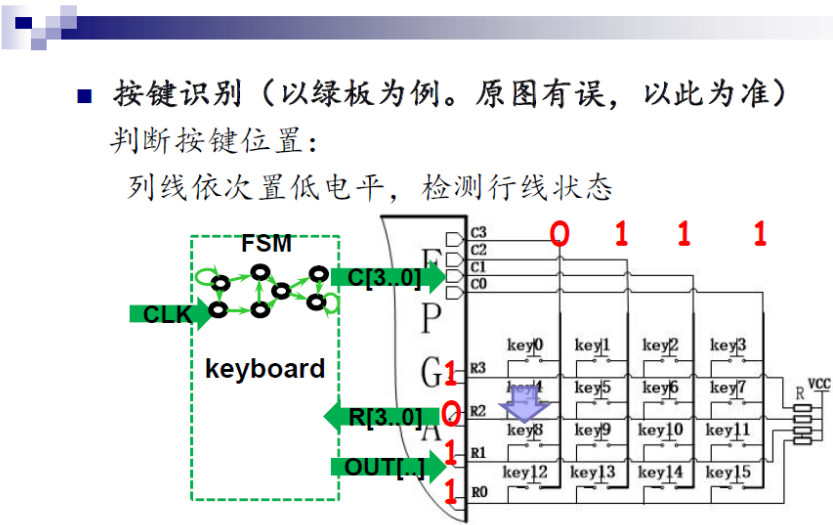


图 10: PPT 中矩阵键盘原理图

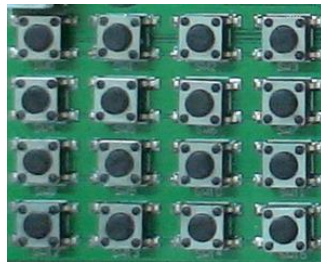
矩阵键盘	C3	PIN_13	行 (Column line) 输入← 低电平有效	
	C2	PIN_14		
	C1	PIN_15		
	C0	PIN_30		
	R3	PIN_31	列 ((Row line) 输出→ 低电平有效	
	R2	PIN_32		
	R1	PIN_33		
	R0	PIN_34		

图 11: 实验板说明书中矩阵键盘引脚表

分配时, 我发现实验板说明书上与 PPT 中有所不同, 如图 11 所示。实验板说明书中的引脚表更加令人疑惑, 首先它混淆了行和列 (Column or Row)。其次, 如果以英文为准, 则列线应该是输入, 行线应该是输出, 这一点又和 PPT 中所示的不同。看到 PPT 中的“以此为准则”字样, 我于是按照 PPT 中的做法进行了引脚分配。但是结果并不理想, 防抖的效果完全不起作用, 我在这上花费了大量的时间。

后来, 我尝试按照实验板说明书上的原理图进行设计, 这次防抖的效果可以实现, 但是数据读入有问题。最后我发现: PPT 上的原理图输入输出是错误的, 行列关系是正确的。所以只需要按照 PPT 的原理图, 并把列线作为输入, 行线作为输出, 即可得到正确的结果。

2. 控制模块的状态机设计。一开始我试图按照三进程的状态机模板来编写控制模块。在 S0 和 S1 状态还算顺利, 但是进入到 S2 状态后涉及到计时的功能, 就不得不使用 CLK 进行计时, 这样的话就需要在 CLK 触发的 always 块中引入计时变量, 则与原来的三进程模式不符。另一方面, 我原来使用 always @(posedge EN) 的方法检测 EN 的上升沿, 这其实不是一种良好的做法。后来我了解到, 在时序电路的 always 块中最好只有一个 CLK 信号, 最多有一个异步复位信号。于是我修改了代码结构, always 块中只包含一个 CLK, 对于 EN 上升沿的检测采用了缓存上一时刻的 EN 为 last_EN, 并与这一时刻的 EN 比对的方式。最终的代码中我只用了一个进程, 这是因为我的 S1 状态中实现了循环输入, 故输出的 M 和 T 并不是和状态对应, 很难单独写出输出方程。后来我了解到, 有的同学将 S1 状态分解为开始状态、输入第一个数、输入第二个数, 这种划分方式可以方便地写出输出方程, 但是却不能实现我希望的循环输入的功能。事实上, 使用几个进程实现并不是主要问题, 关键是能够完整实现出功能; 更高的要求是可以用 Quartus 生成状态转换图, 则一定程度上说明代码结构良好。这两点我都可以实现, 所以满足要求。
3. 初学 Verilog 还是会遇到许多问题。例如阻塞和非阻塞分不清、同一变量不能在两个 always 块中赋值、不同 always 块之间的并行关系、组合逻辑 if 和 else 要写全等等。这些在我编写代码, 以及调试的过程中已经逐渐加深了理解。我认为尤其要注意的一点是不能够望文生义, 而是要用电路的思维去考虑。例如一个组合逻辑的 always 块, 很容易看到 always @(something) 就理解成, 当 something 变化的时候才进入块中。这一点是不对的。应该将其理解成一个综合好的组合电路, 它一直存在着,

`something` 只是这个组合电路的一个输入而已。虽然在一些情况下两种理解方式不影响结果，但是在另一些情况下，错误的理解方式却会导致陷入误区。