EDA大作业二　　投币式手机充电仪

实验报告

班级:　自72
姓名：　吴文绪
学号:　2017010910

2018年 12月 20日

目录

1 实验目的 **1**

2 预习任务 **1**

2.1 电路总体框图和引脚说明 1

2.2 控制电路的状态转换图 3

3 设计思路 **3**

3.1 分频模块 4

3.2 显示模块 4

3.3 键盘输入模块 4

3.4 控制模块 4

4 顶层电路图及模块说明 **5**

4.1 顶层电路图 5

4.2 分频模块 5

4.3 显示模块 6

4.4 键盘输入模块 7

4.5 控制模块 9

5 状态转换图 **11**

5.1 控制模块状态转换图 11

5.2 键盘输入模块状态转换图 12

6 仿真波形图 **13**

6.1 分频模块仿真 13

6.2 显示模块仿真 13

6.3 键盘输入模块仿真 13

6.4 控制模块仿真 15

7 设计和调试中遇到的问题和解决方法 **16**

8 实验总结 **17**

8.1 对verilog在描述硬件的理解 17

8.2 对FPGA的理解 18

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

2.1 电路总体框图和引脚说明

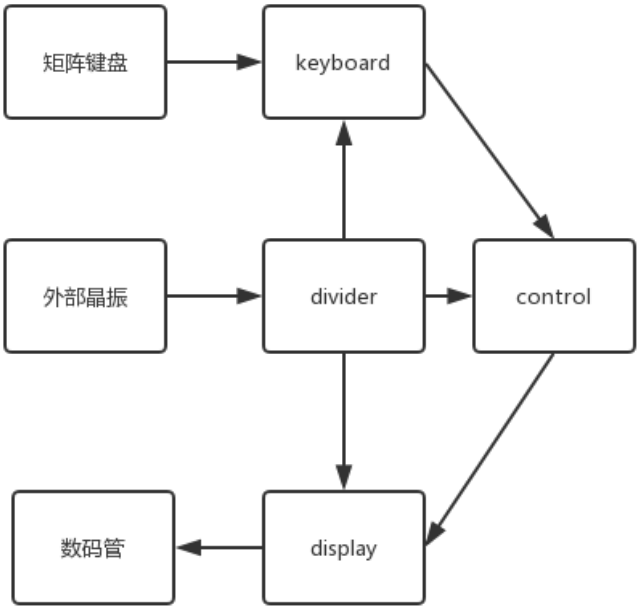


图 2.1.1: 电路总体框图

图2.1.1为抽象出的整体框图，左侧为FPGA实验板上的外设，右侧为用verilog语言编写的模块，个模块的功能如下。

分频模块 本模块对外部晶振的50MHz时钟进行分频，分别给出三个不同频率的时钟，供三个模块调用。输入为晶振时钟OCLK, 输出为三个模块各自的时钟，具体输入输出接口定义如下

```
module divider
(
  input wire OCLK, //50MHz
  output wire ICLK, //3051Hz
  output wire LCLK, //381Hz
```

```
    output wire DCLK, //762HZ
);
```

显示模块 本模块接受四组四位二进制数，按照美化6和9显示后的7448的译码方式，对这四组数据进行扫描显示，故其输入端为扫描时钟和数据，输出为数码管的段选和位选端，具体输入输出接口定义如下

```
module display
(
  input[3:0] D3,
  input[3:0] D2,
  input[3:0] D1,
  input[3:0] D0,
  input CLK, // 762Hz
  output wire A,B,C,D,E,F,G,
  output reg[3:0] DIG
);
```

键盘输入模块 本模块按照讲座ppt所指示，对矩阵键盘进行扫描译码。故其输入为行和扫描时钟，输出为用于控制扫描的列和译码后的数据和控制信号，以及可用于后端模块使能的，键盘是否闲置信号idle，具体输入输出接口定义如下

```
module keyboard
(
  input CLK, // scan f 3051hz
  input [3:0] row,
  output reg[3:0] col,
  output reg[3:0] data,
  output reg start, reset, ok, idle
);
```

控制模块 本模块用于接受键盘得到输入并且对所给数据、控制信号按照状态转换机的模式做处理，输出所需要的数给后级的显示模块。故其输入为数据和控制信号和键盘的idle信号，输出为四组四位二进制数，其中两组表示钱款，两组表示时间，还有一组状态编码，用于查看当前状态机所处状态，方便调试具体输入输出接口定义如下

```
module control
(
  input idle,
  input[3:0] data,
  input start,
```

```
    input reset,
    input ok,
    input CLK, // 381Hz
    output reg[7:0] money,
    output reg[7:0] timer,
    output reg[3:0] state_viewer
);
```

2.2 控制电路的状态转换图

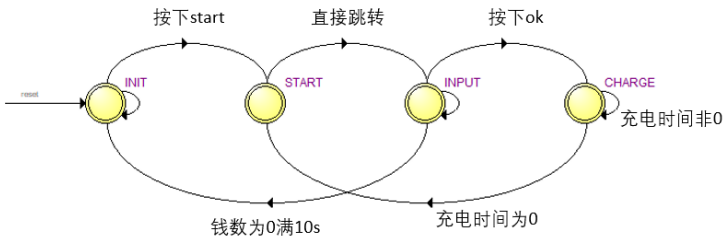


图 2.2.1: 控制电路的状态转换图

控制电路的状态转换图如图2.2.1所示，共有四个状态：初始状态: INIT, 开始状态: START 输入状态： INPUT，充电状态: CHARGE。状态转换条件如下：

* INIT→START: 按下START
* INIT→INIT: 未按下START
* START→INPUT: START状态用于将输出的money, timer置为0，故此次态必为INPUT
* INPUT→INPUT: 没有按下ok，继续输入数字
* INPUT→CHARGE:按下ok，进入充电倒计时
* INPUT→INIT:钱数为0，且持续十秒钟
* CHARGE→CHARGE: 充电时间非0
* CHARGE→START: 充电时间为0

3 设计思路

模块划分在预习任务中已有体现。模块划分要做到符合标准接口设计，任务划分明晰，每一个模块均可独立完成子任务，有较强的复用可能。

3.1 分频模块

通过计数器来进行分频。记待分频信号的频率为f_{in}，分频后信号为f_{out}，则需要f_{out}的一个周期中包含$N = f_{in}/f_{out}$个原脉冲，应当在$N/2$处做时钟信号的翻转，计算中用$mod(N/2)$，来表征。对于多时钟输出的，以最低频的时钟作为计数器重置的基准，对计数器做复位。此为精准的整数倍分频。

由于本实验的模块对频率精准度要求并不高，只需要频率在合适的量级即可。在通过以上方式找到各个模块合适的工作频率后，可以换用只有2的幂次分频的写法。即通过计数器溢出来自动完成复位，直接引出计数器的第n位来完成2^n分频。这样可以减少很多比较、除法、取模运算，减小资源消耗。

3.2 显示模块

本模块的设计和EDA1中的显示模块设计类似，利用一个两位计数器，根据其所记录的数，把对应的数送入译码器选通对应的位选。当计数频率够快时，利用人体的视觉暂留效应达到同时显示四个数的目的。

对于译码器，用verilog的case语句和花括号{}的将多位数据组合的功能，以列真值表的形式构造即可。这样代码的可读性较高，方便查错修改。

3.3 键盘输入模块

根据讲座的ppt，把该模块需要实现的几个功能和主要思路排列如下

* 防抖:通过读取一定时间内的行输入是否为全为1来进行释放/闭合的防抖
* 扫描：让列线中仅有一个为0，来对列线的下拉端进行控制，判断按键具体位置
* 译码：根据所得的按键位置，进行相应的译码并输出，本模块中，数据按译出的原码在data端给出，控制信号单独拥有接口。并约定，数据端为1111，控制端为0，为这个端口的无效输出信号。

3.4 控制模块

基于状态机电路，按照实验说明抽象出四个状态：初始状态INIT，开始状态START，输入状态INPUT，充电状态CHARGE，和键盘的思路类似，用状态机的输出方程去驱动下位计数器，完成充电状态的倒数、钱为0时10秒后回到初始状态，在输入状态下时间随投入金钱变动等功能。

4 顶层电路图及模块说明

4.1 顶层电路图

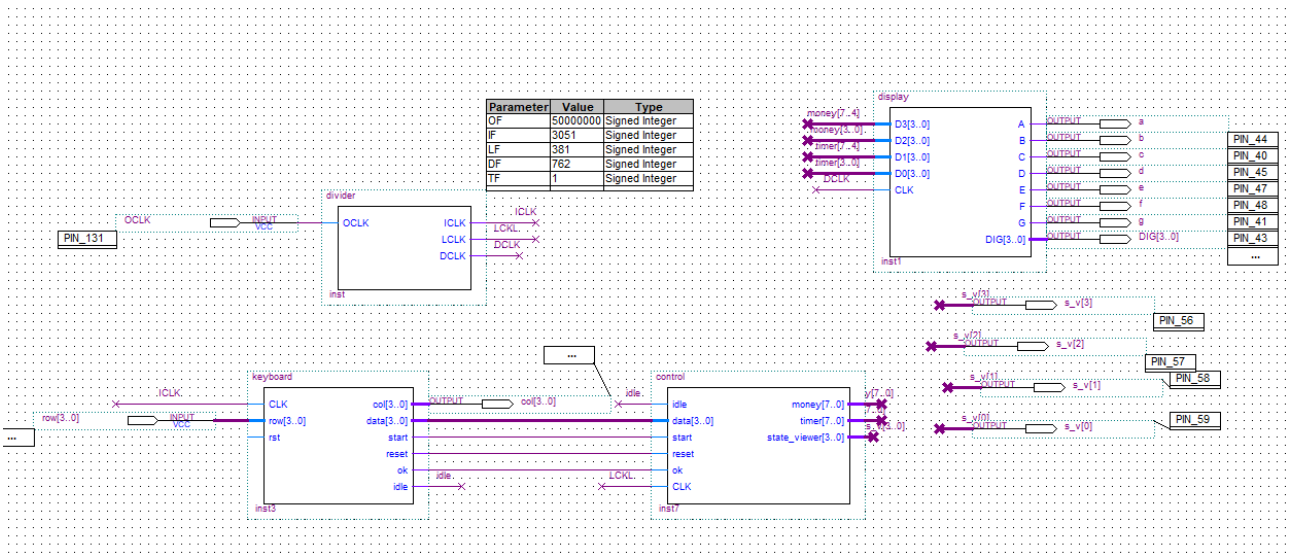


图 4.1.1: 顶层电路图

和预习任务中划分的相同，将四个模块按照原有思路连接。左上角为分频模块，接入外部晶振，输出三个低频时钟给其他三个模块；下侧为键盘模块和控制模块，表现从矩阵键盘读入输出并传递给控制模块处理的信息流；右上角为扫描显示模块，接受控制模块给出的四组四位二进制数并输出。

后续为各个模块的主要代码说明，对应于上一节中设计思路的实现。

4.2 分频模块

这一模块中，利用简单的计数器即可将高频的外接晶振降低为所需要使用频率，值得注意的是，基于计数器的分频可以做精准或非精准分频，精准分频的代码如下

```
1    parameter OF = 50000000, IF = 4000, LF = 500, DF = 1000;
2    reg[17:0] counter;
3    always @(posedge OCLK) begin
4        if (counter >= OF / LF - 1)
5            counter <= 0;
6        else
7            counter <= counter + 18'b1;
8    end
9    always @(posedge OCLK) begin
10       if (counter % (OF / IF / 2) == 0)
11               ICLK <= !ICLK;
12       // ... LCKL, DCLK
```

```
13      end
```

这样能得到精准分频的结果，但是可以看到多位加法除法和模运算都十分消耗资源。而实际任务对于时钟频率并没有那么精准的要求。故在使用这份代码找到各个模块时钟合理的量级后，用以下的模块进行替代。这份代码的优点是利用了计数器的溢出自然的复位，利用计数器的某一为直接作为分频结果，不需要译码电路，节约资源，缺点是分频结果不整，仅能输出2的幂次分频的结果。

两个模块的编译消耗资源的差距可以通过观察Quartus的输出报告得到，其combinational function 从1000多个下降到几十个，优化效果是很可观的。

```
1       always @(posedge OCLK) begin
2           counter <= counter + 17'b1; //利用计数器溢出自然复位
3       end
4       assign ICLK = counter[13];
5       // ... LCKL, DCLK
```

4.3 显示模块

本模块的设计和EDA1中的显示模块设计类似，利用一个两位计数器，根据其所记录的数，把对应的数送入译码器选通对应的位选。当计数频率够快时，利用人体的视觉暂留效应达到同时显示四个数的目的。在这次作业中用代码重构了这样的思路，代码如下

```
1       always @(posedge CLK)
2           cnt <= cnt + 2'b1;
3       always @(cnt)
4       begin
5       case(cnt)
6           2'b00:
7           begin
8            cur_num = D0;
9            DIG = 4'b0001;
10          end
11          // ... cnt == 01, 10, 11 -> D1, D2, D3
12      endcase
13      end
```

其中my7448为一个美化了6和9显示的译码器，利用verilog的case语句和花括号{}的将多位数据组合的功能，以列真值表的形式构造了此译码器

```
1   // my7448
```

```
    always @(num)
    case(num)
        // ... 0-5
        4'h6: {A, B, C, D, E, F, G} = 7'b1011111;
        // ... 7-8
        4'h9: {A, B, C, D, E, F, G} = 7'b1111011;
        // ... a-f
    endcase
```

4.4 键盘输入模块

先给出相关中间变量的定义与说明

```
    //用于定位行列，由于是列扫描所以列的闲置状态为0000，而行为1111
    localparam CIDLE = 4'b0000, RIDLE = 4'b1111, CR3 = 4'b0111; // CRx...
    //定义键盘模块的状态，分别为闲置、闭合防抖、扫描、输出释放防抖
    localparam IDLE = 5'b00001 // ... PRESS_CHECK, SCAN, OUPUT, FREE_CHECK
    /* 防抖计数器和其置位端，和反映防抖计数满用于键盘状态机跳转的变量*/
    reg[3:0] anti_shake_cnt; reg anti_shake_cnt_reset; wire
        is_anti_shake_cnt_full;
    /* 扫描用变量，用扫描计数器驱动col的扫描，其他为变量为扫描计数器控制端*/
    reg col_scan_hold; reg col_scan_end; reg[2:0] col_scan_cnt;
    /* 译码用变量，reset_out用于把输出的数复位到无效，idle_last用于判断是否第一次进
        入OUTPUT状态并开启译码*/
    reg reset_out;
    reg idle_last;
```

在设计思路节给出的划分下[1]，将键盘模块也用状态机进行实现，有五个状态：闲置IDLE，闭合防抖PRESS_CHECK, 扫描SCAN, 输出OUTPUT, 释放防抖FREE_CHECK。这样仅在扫描状态下，需要对列输出进行修改，在闲置态和闭合防抖态，把列输出置为0000，在输出和释放防抖状态，列输出锁定在扫描时得到的列的状态，在列输出的四个驱动端口，有效降低了功耗。

由于此状态机的状态较多，且状态跳转关系较为复杂，故使用独热码对状态进行编码[2]

防抖功能在PRESS_CHECK和FREE_CHECK态实现，一段时间内行输入均不全为1，则可视为已经经过了按下时的抖动，可以进行扫描；段时间内行输入均全为1，则可视为已经经过了释放时的抖动，可以回到闲置状态。在状态机的输出方程中，驱动防抖计数器的对应端口，并且把计数器慢的状态作为反馈，知道状态机的跳转，即可完成这样的功

[1] 见第3.3节

[2] 详细分析见实验总结第8.2节的相关段落

能。以PRESS_CHECK的次态跳转为例，其相关代码实现如下。

```
// ... anti_shake_cnt++
always @(state) begin
  case(state)
      /*...IDLE*/
      PRESS_CHECK:
          begin
              if (is_anti_shake_cnt_full && row != RIDLE)
                  next_state = SCAN;
              else if (!is_anti_shake_cnt_full && row != RIDLE)
                  next_state = PRESS_CHECK;
              else
                  next_state = IDLE;
          end
      /*...SCAN, OUTPUT PRESS_CHECK */
  endcase
end
```

扫描，即在SCAN状态所需要进行的工作，也是使用了通过状态机输出方程，控制下位的列扫描计数器完成的。由于前期防抖的存在，能进入到扫描态时，按键必定是按下了的，故只需要进行一次扫描，得到具体位置，交由OUTPUT态译码即可。也就是当扫描到row != RIDLE，次态为OUTPUT，否则为SCAN，OUTPUT状态下，不再扫描，通过对col_scan_hold保持扫描计数器得以实现。

在SCAN到OUTPUT的变化中，idle从1变为0，表示键盘非闲置（开始工作）。故在idle的下降沿对数据更新。这样和普通的组合电路相比，防止了按键在时钟上升沿到来之时已经释放，产生一小段不该存在的译码过程，保证数据的可靠性。但为了满足实验要求，设计一个同步电路，使用idle == 0, idle_last == 1判断进入OUTPUT后的第一个时钟周期，达到类似下降沿译码的效果。代码展示如下：

```
// OUTPUT
       /*...IDLE PRESS_CHECK*/
       SCAN:
              idle = 1;
              //...other output
       OUTPUT:
              idle = 0;
              //...other output
       /*..FREE_CHECK*/
/*********decoder*********/
    //... if asynchronous signal
       else if (idle == 0 & idle_last == 1)
    // ... decode start
```

```
        else
            {data, start, reset, ok} <= {data, start, reset, ok};
end
```

4.5 控制模块

先给出相关中间变量的定义与说明

```
    // 定义四个状态初始、开始、输入、充电，开始状态仅用于把输出初始化为0，会无条件跳转到输入状态
    localparam INIT = 2'b00, START = 2'b01, INPUT = 2'b10, CHARGE =2'b11;
    // 输出全1会被译码为数码管全灭，而错误数是和键盘模块约定好的错误数码
    localparam HIDE_NUM = 4'b1111; localparam ERROR_NUM = 4'b1111;
    reg[1:0] current_state, next_state;
    //10s和1s的计数器和他们的相关控制反馈变量，分别用于判断是否需要回到初始状态和充电的倒计时
    reg[11:0] ten_cnt; reg ten_cnt_reset; wire is_ten_full;
    reg[7:0] one_cnt; reg one_cnt_reset; wire is_one_full;
    // 关于显示的一些中间变量，均为正逻辑，为1时实现同步功能：把钱和时间置1，重置钱，重置时间，保持钱，根据钱设置时间，时间是否为0（充电是否结束）
    reg hide_time_money, reset_money, reset_time, hold_money, set_by_money;
    wire is_charge_over;
```

类似键盘模块的设计，模块的主体是一个状态机，状态机的输出方程用于控制下位的这些计数器和其他时序电路的工作与停止，并根据计数器的反馈，做状态的跳转。

状态机输出方程对其他时序模块的控制体现如下，在INIT态，把hide_time_money置1，timer和money在hide_time_money为1时，输出1111，即全灭；在START态，将两个reset端置为为1，实现timer和money的清零，为输入做好准备；在INPUT态set_by_money为1，此时tiemr和money都根据读入的数据做设置，reset端根据输入的reset信号置为0或1，实现清零功能；在CHARGE态hold_money为1保证timer在倒数时自己不变。在书写时，要注意输出方程模块为一组合电路，为了避免编译时出现锁存器，在每一个case语句中都需要把这些涉及的变量的取值，无论01都编写进去，在行文中只是重点关注需要驱动为1的那些主要功能端。

故由此，以CHARGE态为例，展示控制状态机的输出方程展示如下：

```
    always @(current_state) begin
    case (current_state)
        // ...INIT START INPUT
        CHARGE:
            begin
                ten_cnt_reset = 1;
                one_cnt_reset = 0;
```

```
                hide_time_money = 0;
                hold_money = 1;
                reset_money = 0;
                set_by_money = 0;
            state_viewer = 4'b1000;
            end
    endcase
end
```

而由其驱动的两个简单计数器的代码不赘述，重点关注timer和money这两个复杂时序电路的变化.

关于money电路，其代码为

```
    always @(posedge CLK) begin
        if (hide_time_money)
            money <= {HIDE_NUM, HIDE_NUM};
        else if (reset_money)
            money <= 0;
        else if (hold_money)
            money <= money;
        else if (data != ERROR_NUM && idle_last == 1 && idle == 0)
        // ...输入新数判断大小并做对应的位移寄存或者保持
    end
```

前面都是较高优先级的判断项，表示money应该被消除或者重置为0或者保持；最后是INPUT的时从上级读数并判断新数应该如何表现的过程。读入新数据的判定和键盘模块更新输出的判定类似，都用到判定idle下降沿后第一个周期的方法[3]，不再赘述。

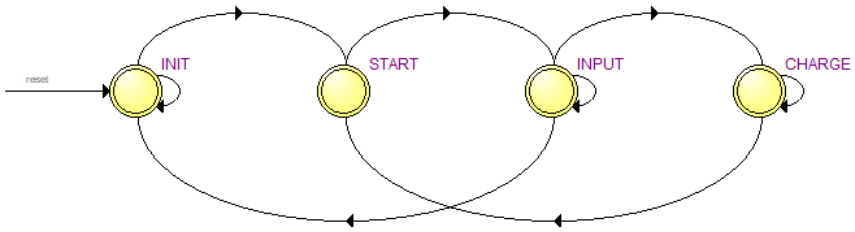
timer的配置和其类似，只是新增一个多一个在一秒计数器满时减一，表现充电的过程。并且读数并判断新数应该如何表现的判定也有一定的修改。

在由计数器反馈的一侧is_ten_full为1作为INPUT态到INIT态的条件；is_one_full，作为timer-1的条件；is_charge_over作为CHARGE态到START态的条件。

[3] 见4.4

5 状态转换图

5.1 控制模块状态转换图



	Source State	Destination State	
1	CHARGE	CHARGE	(!timer[4]~reg0).(!timer[3]~reg0).(!timer[2]~reg0).(!timer[1]~reg0).(!timer[0]~reg0).(!timer[5]~reg0).(!timer[6]~reg0).(timer[7]~reg0) + (!timer[4]
2	CHARGE	START	(!timer[4]~reg0).(!timer[3]~reg0).(!timer[2]~reg0).(!timer[1]~reg0).(!timer[0]~reg0).(!timer[5]~reg0).(!timer[6]~reg0).(!timer[7]~reg0)
3	INIT	INIT	(!start)
4	INIT	START	(start)
5	INPUT	CHARGE	(ok).(!WideAnd0)
6	INPUT	INIT	(WideAnd0)
7	INPUT	INPUT	(!ok).(!WideAnd0)
8	START	INPUT	

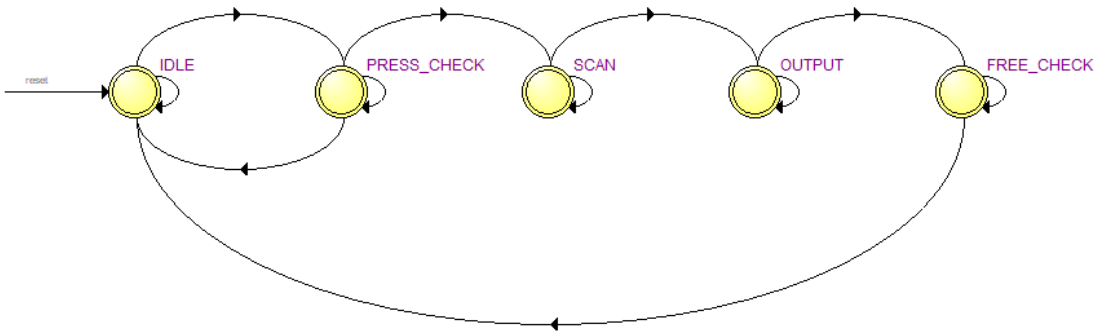
图 5.1.1: 由Quartus State Machine Viewer 生成的控制模块状态转换图

由Quartus的State Machine Viewer生成的状态转换图和其条件见上，可以看到符合预习任务中的设计方法[4]。而在condition中列出的跳转条件有些是指其外电路的运算模块，不对应变量名，但可以在RTL view中查到，在这里把一些命名不够有意义的条件整理如下：

* START→INPUT: 和预习设计一致，START态仅用于把timer和money清零，无条件跳转到INPUT态接受输入。
* INPUT→X:WideAnd0即 &ten_cnt, 表示十秒计数器满。
* CHARGE→CHARGE: 所列长逻辑式即|timer,即timer非0，充电时间非0
* CHARGE→START: 所列长逻辑式即!(|timer),即timer为0，充电时间为0

[4]见图2.2.1和其说明

5.2 键盘输入模块状态转换图



	Source State	Destination State	Condition
1	FREE_CHECK	IDLE	(WideAnd0)
2	FREE_CHECK	FREE_CHECK	(!WideAnd0)
3	IDLE	PRESS_CHECK	(!row[0]) + (row[0]).(!row[1]) + (row[0]).(row[1]).(!row[2]) + (row[0]).(row[1]).(row[2]).(!row[3])
4	IDLE	IDLE	(row[0]).(row[1]).(row[2]).(row[3])
5	OUTPUT	OUTPUT	(!row[0]) + (row[0]).(!row[1]) + (row[0]).(row[1]).(!row[2]) + (row[0]).(row[1]).(row[2]).(!row[3])
6	OUTPUT	FREE_CHECK	(row[0]).(row[1]).(row[2]).(row[3])
7	PRESS_CHECK	SCAN	(!row[0]).(WideAnd0) + (row[0]).(!row[1]).(WideAnd0) + (row[0]).(row[1]).(!row[2]).(WideAnd0) + (row[0]).(row[1]).(row[2]).(!row[3]).(WideAnd(
8	PRESS_CHECK	PRESS_CHECK	(always4).(!WideAnd0) + (always4).(WideAnd0).(row[0]).(row[1]).(row[2]).(row[3])
9	PRESS_CHECK	IDLE	(!WideAnd0).(!always4) + (WideAnd0).(!always4).(row[0]).(row[1]).(row[2]).(row[3])
10	SCAN	SCAN	(row[0]).(row[1]).(row[2]).(row[3])
11	SCAN	OUTPUT	(!row[0]) + (row[0]).(!row[1]) + (row[0]).(row[1]).(!row[2]) + (row[0]).(row[1]).(row[2]).(!row[3])

图 5.2.1: 由Quartus State Machine Viewer 生成的键盘输入模块状态转换图

由Quartus的State Machine Viewer生成的状态转换图和其条件见上，可以看到符合4.4节中相关段落的设计。而在condition中，同样列出的跳转条件有些是指其外电路的运算模块，不对应变量名，但可以在RTL view中查到，在这里把一些命名不够有意义的条件整理如下：

* WideAnd0:即 &anti_shake_cnt, 表示防抖计数器满。
* always4:即 !(&anti_shake_cnt)&&!(row == 4'b1111), 表示防抖计数器未满但行有输入。
* IDLE→PRESS_CHECK，OUTPUT→OUTPUT, SCAN→OUTPUT: 所列长逻辑式即row != 4b́1111表示行非空，此行有按键按下。

6 仿真波形图

rst端为所有涉及的模块的异步复位端，在仿真中，modelsim需要利用这个端口把模块内部的时序部分赋确定的初值，否则内部的寄存器会处于x状态，无法计算出结果。但是在实际使用中，上电后寄存器的初态给定，不需要用到这个端口。所以在工程文件中，复制了所有模块的实现，并加入异步复位端，专门供仿真使用。

6.1 分频模块仿真

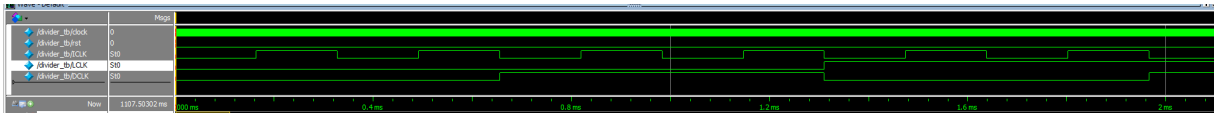


图 6.1.1: 分频模块仿真

这里按照实际时间设置，clock端模拟外部晶振的50MHz输入，并预期输出的分频比为分别为ICLK2^{14}分频，即3051Hz；DCLK2^{16}分频，即762Hz；LCLK2^{17}分频，即381Hz。抽取DCLK读数，其周期大约在大于1ms的量级，符合预期设计。

6.2 显示模块仿真

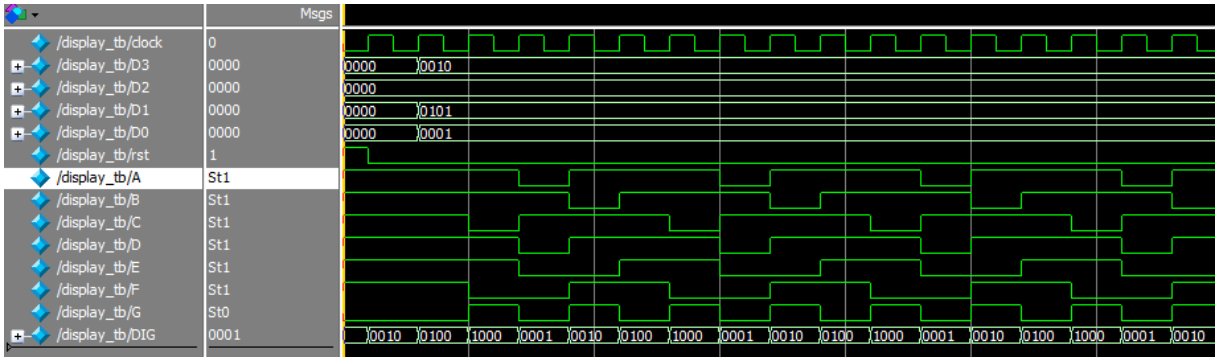


图 6.2.1: 显示模块仿真

可以看到，在D3-D0给出不同的数时，A-G能根据时钟信号选择待译码数据并译码为数码管的段选信号；并且最下方的位选端也会根据时钟信号而周期性变化。

6.3 键盘输入模块仿真

键盘的功能已经被拆分，故仿真验证其功能时也按照防抖、扫描、译码这三个方面逐一验证。同时由于键盘采用状态机电路，在仿真中加入了state_view来观察其状态。

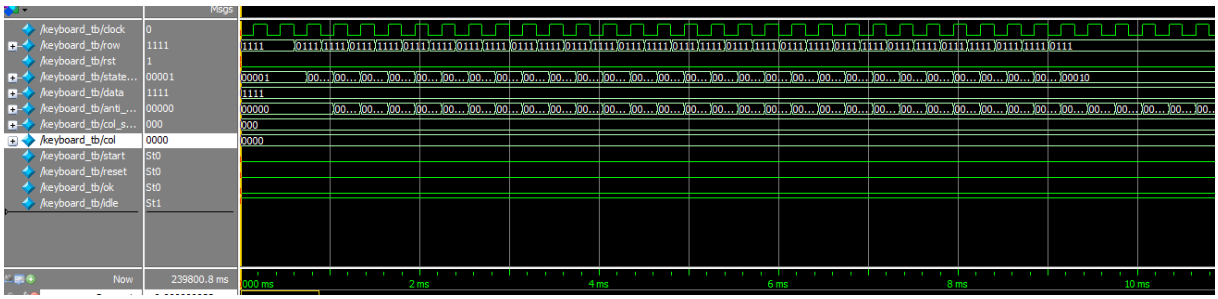


图 6.3.1: 键盘模块仿真-闭合抖动

可以看到行输入在1111和非1111之间跳动，这时从state_view可以看到，其状态也在00001即IDLE和00010即PRESS_CHECK之间跳动，且idle为1表示闲置，data为错误代码1111，其他控制端为000,符合预期。

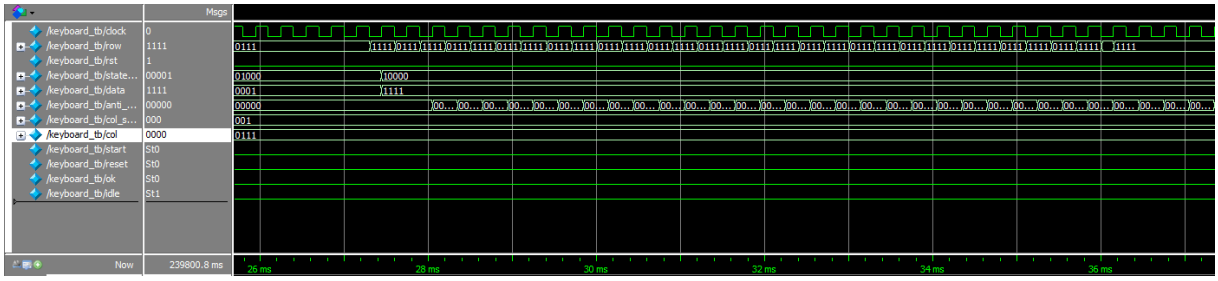


图 6.3.2: 键盘模块仿真-释放抖动

可以看到行输入在1111和非1111之间跳动，这时从state_view可以看到，其状态也在01000即OUTPUT和10000即FREE_CHECK之间跳动，且idle为0表示已经输出；在释放抖动的检查中，data和其他控制端保持之前译码结果0001，控制信号全0，符合预期。

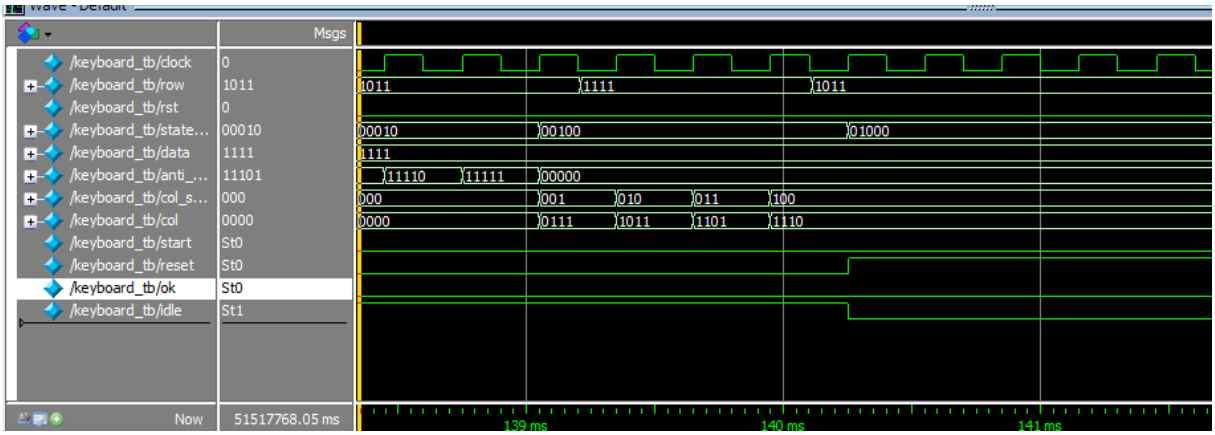


图 6.3.3: 键盘模块仿真-扫描和译码

可以看到，在进入SCAN模式时，前三次扫描（即在对3-1列的扫描）得到的行信号为闲置1111，这时col的次态会是把下一列置为0做进一步扫描；而状态仍然停留在SCAN也

即00100。当扫描第0列时，得到行线输入非1111，转入到OUTPUT状态01000，并译码得到其结果为输出控制信号reset。

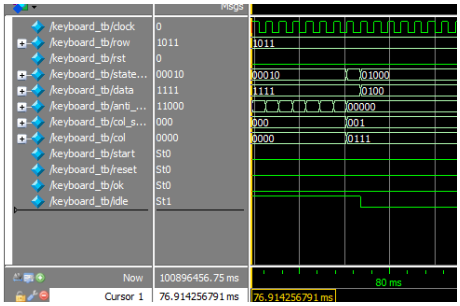


图 6.3.4: 键盘模块仿真-译码数字

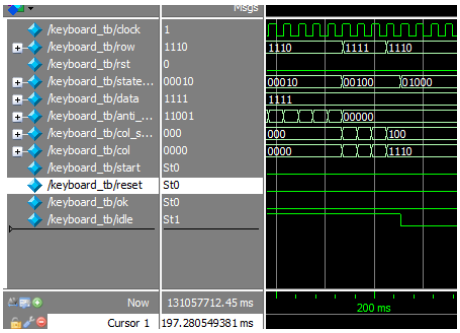


图 6.3.5: 键盘模块仿真-译码无效按键

上两图是对译码的其他验证，分别成功译码数字4和右下角的未定义按键。在译码无效按键时，可以看到，数据端给无效数字1111，控制端全为0，符合模块间的约定。而idle为0表示确实有按键按下，这样可以在需要扩展功能，用到更多的按键时，仅改变代码中译码的部分即可。

6.4 控制模块仿真

此模块仿真中，覆盖核心状态机的各种转换情况即可，同样在仿真中加入了state_view来观察状态机。

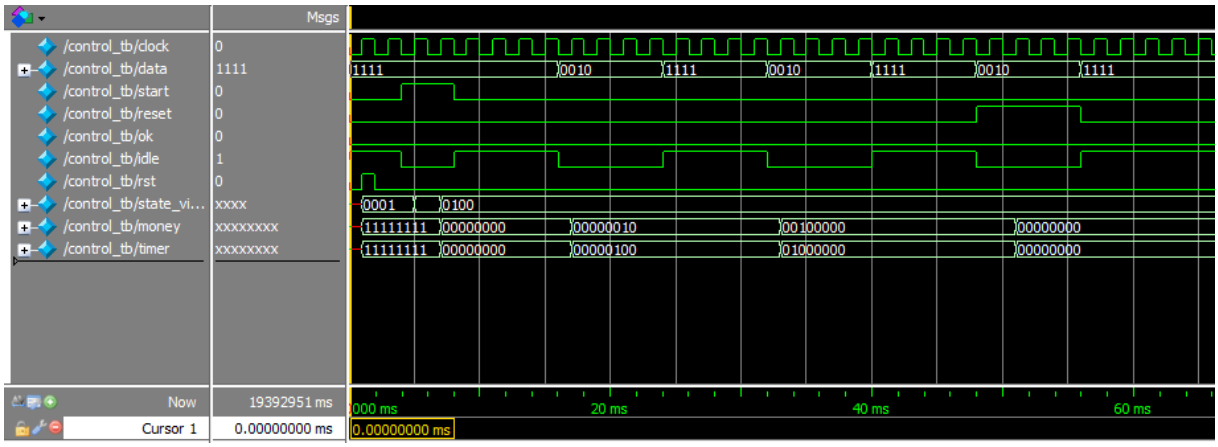


图 6.4.1: 控制模块仿真-开始、输入超额、复位

在按下start按键后，从state_view看到，状态机从INIT态的0001经由START态0010到达INPUT态0100。输出的钱与时间在START态被初始化为0。在INPUT态，连续读入两个数2，第一次，钱数被更新为2时间被更新为4，第二次，总计输入为22，超过20，所以钱数被置为20，充电时间被置为40。然后reset信号到来，二者被清零，INPUT状态的数据读写功能验证完成。

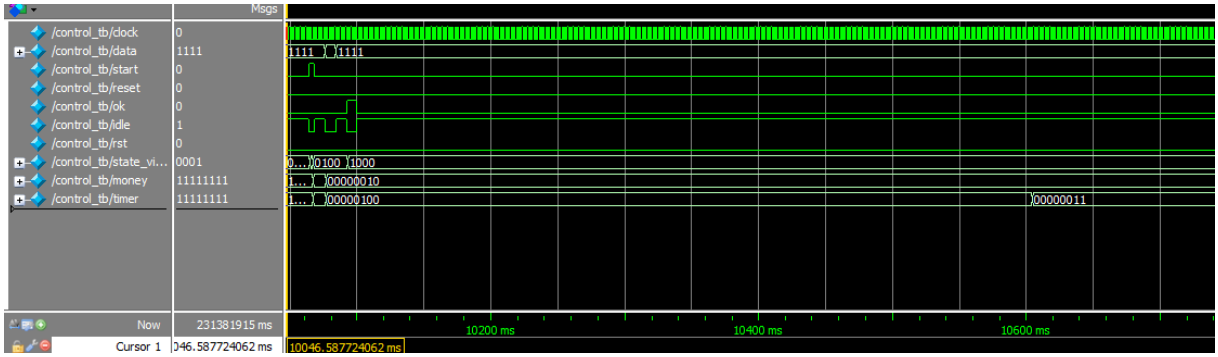


图 6.4.2: 控制模块仿真-充电

这次的情景为按下start，输入数据4，按下ok开始充电。可以从state_view看到，状态机从INIT态的0001经由START态0010到达INPUT态0100再到达CHARGE态1000。同时在仿真的后段，看到timer从4到3，进行了倒数，而money保持不变，符合设计要求。

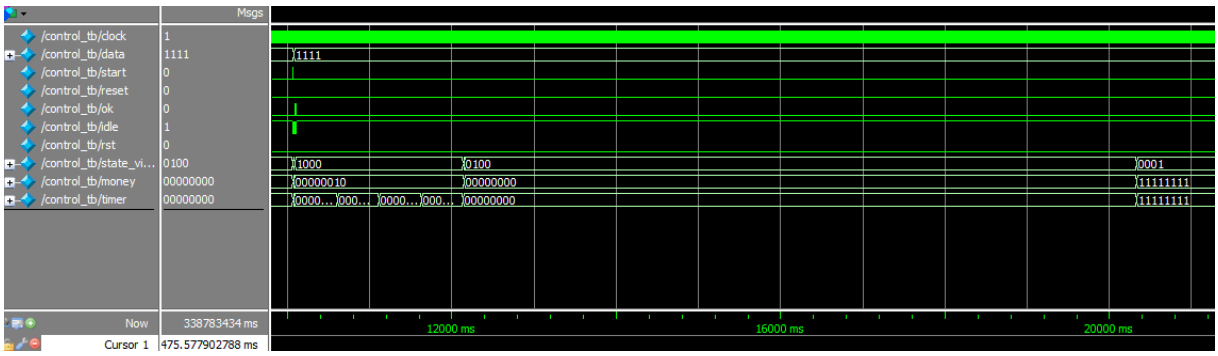


图 6.4.3: 控制模块仿真-结束

接续上次的情景，在充电结束后，可以从state_view看到，状态机从CHARGE态回到了INPUT态，在INPUT态的零金额维持了约10秒后，回到INIT态，money和timer被置为全部为1，对外表示为不显示，符合设计要求。

7 设计和调试中遇到的问题和解决方法

1 对verilog的模板范式理解不够清晰，无法对应到电路结构。在作业开始初期，我仅凭印象记住了异步模块的复位端需要出现在always敏感表中，但是还忘记了要用if语句先行对异步信号进行判断。这样才能综合出来一个正确的异步控制端。我最初代入了一些其他语言的想法，认为if和else交换顺序是无所谓的，绕了很大的弯路。在认真看书看视频学习了这些规范后，才写出了我想要的代码。

此外，在状态机组合电路的书写中，同样是因为代入了其他编程语言的习惯认为值是存在变量中，对于一些上一个状态到下一个状态没有改变输出的量，没有进行赋值。而这样也仅是得到一个latch的warning，但是实际调试中会带来意想不到的麻烦。同样的，这也导致了我有一些ifesle不完整，综合出来的电路并不是自己想要的。在查询相关资料，回

看教学视频，分清楚verilog和其他串行编程语言的不同，认识到自己是在写电路后，这样的问题便解决了。

2 无法用Quartus的state machine viewer调出状态机。在第一次写相关模块时，我always块的划分并不明晰，许多电路都杂糅在一起。在回看了状态机模板和查阅资料后，了解到state machine viewer识别状态机需要满足的条件，除了要用always块分开驱动、状态、输出方程之外，还需要用parameter或者localparam枚举状态并编码，用always @(state)来描述组合电路部分即状态方程和输出方程。并且内部要用case(state)来说明。了解了这个以后，在那些需要记录时间来进行状态转换的部分，我又犯难了，因为计数器的自增又很难写入这些模块中。最后，从硬件接线的角度思考这个问题，我决定用输出方程控制计数器使能或者复位端，计数结果反馈到状态机帮助状态转换。分出上下位关系，解决问题。

3 modelsim仿真时需要一个异步端来把内部的寄存器状态从x变到有限值，且对于我那样用多个always块描述不同时序电路，在通过一些使能变量做交互的流程不太支持，但将时序进程和写又会得不到state machine viewer调出的状态机。所以我复制原有代码，加入异步复位端，整合了进程，用于testbench的调用。当然还存在一定的缺陷，即这个仿真用模块并不在顶层出现，所以拉起仿真时，需要把对应的仿真用模块置为顶层。

8 实验总结

8.1 对verilog在描述硬件的理解

verilog用于描述电路，本质上是并行的，和其他串行的程序语言是有很大区分的。在这次实验中，其具体体现在这些方面

1 在C语言和类似语言中，声明变量是需要开辟空间消耗资源的。但是verilog是通过语言寻找对应的电路结构，多声明变量并不一定代表更多的资源。比如三进程的状态中，通常需要两组reg，state和next_state。看似多声明了一组变量，比双进程消耗了更多的资源，但是在实际的编译报告中看到，并没有出现更多的触发器消耗。同样的还有一些wire型定义的中间条件变量，也不会多消耗板上资源。巧妙利用好这些特性，可以声明更多有意义的变量，提升代码可读性而不多消耗资源。

2 对于always的理解。在一般的教程always后跟随的叫做敏感列表。同样如果把计算机编程语言的一些认知代入，会认为是敏感列表中的一些值发生变化后，启动进程，其他情况下值保持。我初期便是这样理解的，所以在块中对引起变化的事件做相应，导致电路综合时产生了很多不必要也不是功能所想要的锁存器，也导致了在上一节中提到的对异步端口要用if优先判断的误解。

在完成作业，重新审视一下这个功能。我把always的功能总结如下：

* always引导的是一个电路模块

* 电路是时序电路还是组合电路看敏感表中是否出现边沿

* 时序电路的异步端口也需要在敏感表中以边沿的形式声明，但是需要在进入always后马上接if进行判定和执行功能，赋值时用<=。

* 引导组合电路时，把敏感表中的量看做是组合电路的输入，以此进行判定和赋值，要注意在各种情况下的赋值，都应该涵盖所有输出变量，否则有可能得到锁存器。赋值时用=。

* 在这样的理解下，变量不能在两个always块中被赋值的要求就是显然了，因为always中被赋值的要不是组合电路的输出要不是触发器的次态，这样对应在电路连接上就是不能直接对门电路线与。

3 对于阻塞赋值和非阻塞赋值的理解。还是和计算机编程语言的冲突点。“非阻塞”一词，很容易和并行画上等号。而使用硬件的时候，更多是希望是并行高效的，就很容易产生<=的滥用，比如在always引导的组合电路中使用<=。但实际上，通过理解代码规范和查看RTL viewer的电路，我把这两个符号的功能重新理解如下。

阻塞赋值=的含义是在综合电路时，计算机做顺序读取，在有赋值先后顺序时，判定出是否需要加入中间的门电路或者舍弃一些无意义赋值。综合出来的电路还是并行的，赋值的先后仅有电路延时决定。非阻塞赋值<=的含义是，在综合电路时，读取到cnt<=cnt+1之类语句时，综合出触发器这样的能有初态次态区分的电路元件。所以阻塞非阻塞，和电路效率无关，仅和电路功能有关。

4 reg型不一定是reg，它仅代表它是一个可以记忆其原始值的变量，综合出来的是组合电路、锁存器还是触发器，要看具体写法。但wire型一定是wire，是电路的输入输出的那条线。

8.2 对FPGA的理解

在参考网上代码的过程中，我看到许多状态机用独热码进行编写，即n个状态用n位的reg型保存，每一个状态的编码是一位为1其他为0。我以为这样的不仅产生了更多的无关态，还使用了更多的触发器存储变量，浪费资源。但在了解了FPGA的原理后，明白了它是运用D触发器和一些常用逻辑模块来模拟组合电路，触发器是其最底层的资源。所以，独热码的优势就在于不需要去额外综合一些对状态的译码电路，反映到FPGA上触发器的消耗反而可能会更少。在状态多、转换条件复杂的状态机中常使用。在本次作业中，键盘输入模块状态机符合这样的条件，所以采用了独热码进行编码。

此外，在分频模块的修改正，也体现了用触发器模拟其他组合电路有可能产生更多的资源消耗这一点。在作业尾声，我尝试对项目优化，发现近千个combinational functions的消耗均来自分频这样一个相对简单的模块。究其原因，发现是在精准分频的构造下，FPGA需要对17位二进制数进行按位的译码、比较、除法和模，对应到所用的触发器，数量就很庞大了。考虑到作业对频率的精度要求不高，选择在分频精度上做让步，用16位

二进制数计数器的某些位做2的幂次分频。FPGA资源从几千到几十的消耗优化，是非常可观的。

总而言之，这次大作业带我入门了verilog这样一个不同所学其他编程范式的语言，也让我认识到了手中FPGA板的无限可能性，还让我在实践中深化了数电课上所学知识的理解。总而言之，收获颇丰，回味无穷。