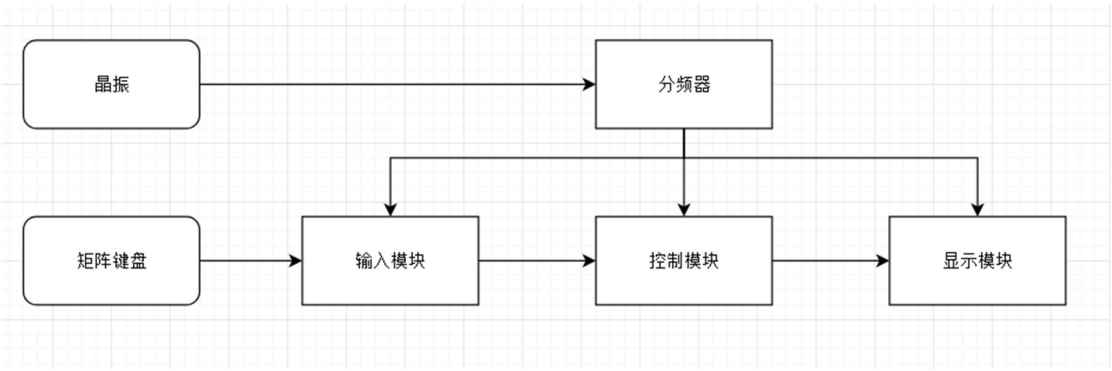
EDA 大作业二 投币式手机充电仪

自 82　何博航　2018010866

一、 预习任务

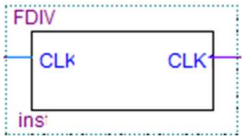
1. 阅读并分析任务要求，画出电路的总体框图，注明各功能模块及其引脚

电路总体框图:



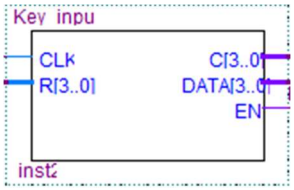
各功能模块及其引脚:

- 分频模块



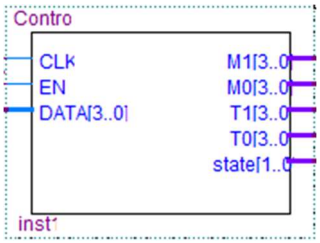
将晶振的 50MHz 频率降下来，为其他电路提供时钟信号。
输入 clk 接晶振，输出 clk 为分频后得到的电路（20000hz）

- 输入模块



从矩阵键盘读取输入的信息（解决了长按健和按键抖动的问题）。
输入时钟信号为分频后得到的信号，R 为矩阵键盘的行线，与 FPGA 板子连接，输出为矩阵键盘列线 C，数据 DATA（包含了此刻输入的值的信息，如输入数字、开始、清零或是确认），数据使能控制端 EN（上升沿时代表数据此刻有效）。加入 EN 是我为了解决长按健问题，这样一来只有 EN 上升沿的时候按键是有效的，就避免了长按键时多次读入。

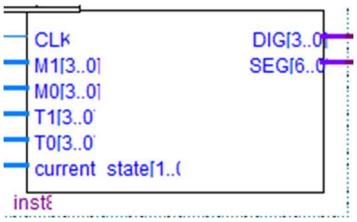
- 控制模块



处理输入模块得到的信息，整理成投币数额和充电时间（即数码管上四位数字）的数，以及当前状态的信息（这是为了解决初始状态要保证数码管不显示从而加入的）。

输入时钟信号为分频后得到的信号，输入 EN、DATA 分别为控制模块的输出 EN、DATA。输出 M1、M0 表示投币数额的高位和低位，T1、T0 表示充电时间的高位和低位，输出 state 表示当前的状态。

- 显示模块



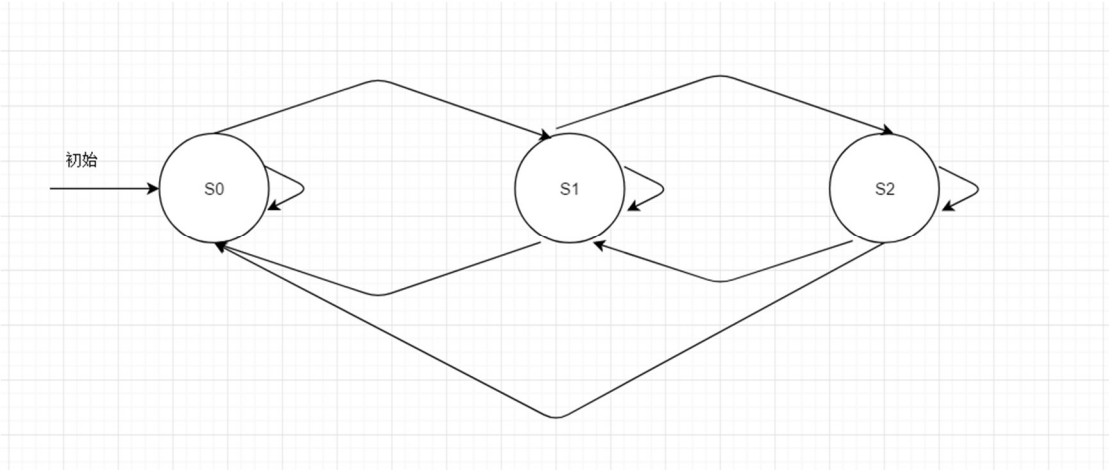
从控制模块接受信息，从而显示在数码管上。

注：为了解决显示功能，我还特意手写了 my7448 代码

输入时钟信号为分频后得到的信号，输入 M1、M0、T1、T0、current_state 分别与控制模块的输出 M1、M0、T1、T0、state 相连。输出 DIG 接外设数码管位选线 DIG0~DIG3，输出 SEG 接外设数码管段选线(A~G)。

2. 根据任务要求画出控制电路的状态转换图

控制电路状态转换图如下：

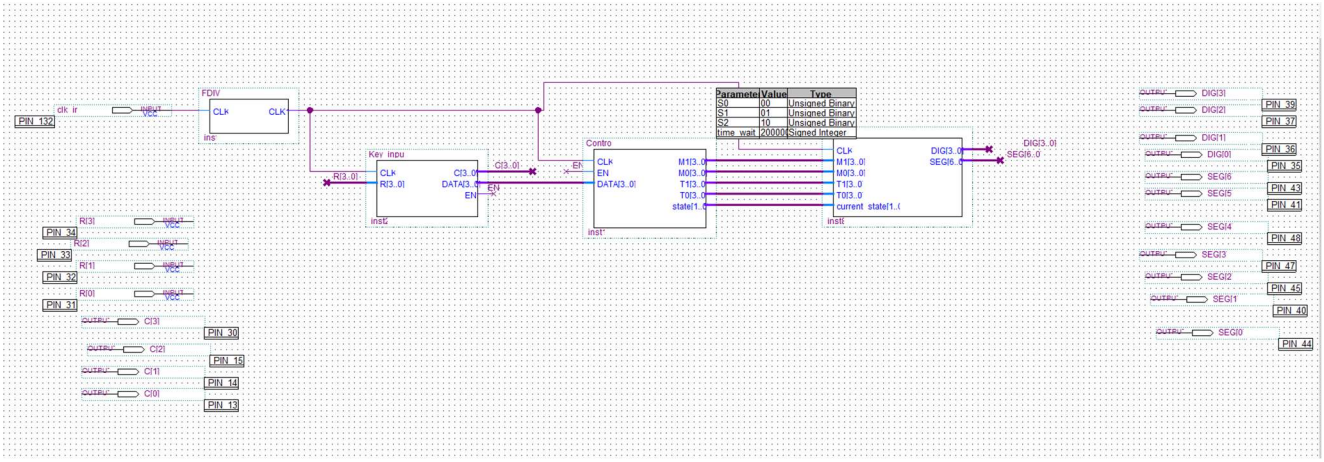


S0 代表初始状态，S2 代表倒计时状态，S1 代表其他情况的状态（包括开始状态、输入状态等）。

状态之间转换关系如下：

- S0→S1：按下“开始”键。这时数码管显示为“0000”，为开始状态，可以按下各种按键。
- S1→S2：投币金额不为 0 时按下“确认”键。
- S2→S1：充电倒计时结束。这时数码管显示为“0000”，即为开始状态。
- S1→S0：任意时刻，10s 无操作即回到 S0 状态（数码管全灭）
- S0→S0：未按下“开始”键。
- S1→S1：投币金额不为 0 时未按下“确认”键，且任意 10s 内有操作。
- S2→S2：保持状态直到充电倒计时结束。

二、 顶层电路图



三、 设计思路（包含各模块功能）

利用模块化方法，“自顶向下”划分功能模块。

- 分频模块

将晶振的 50MHz 频率降下来，为其他电路提供时钟信号。

思路很简单，用一个 count 来计数，每次 CLK 上升沿到达时，如果 count 没达到 1249，就加 1，要是到了 1249，就把信号 FIN 取反，并且把 count 清 0。这样，再让输出信号 CLK1 恒等于 FIN，就达到了 CLK1 由 CLK 分了 2*（1249+1）=2500 分频的效果。

分频得到频率太高，我是因为要避免使能控制端频率太高，（clk 频率据分析应该比 EN 频率高很多才行），并且要避免数码管有重影，我就取 20000 试试，事实上只要改个 count 上限就行。得出来 20000 可行也就不管其他的了。

源代码如下：

```
module FDIV(CLK, CLK1);
    input CLK;
    output CLK1;
    reg FIN;
    reg [10:0] count ;
    always @(posedge CLK)
    begin
        if (count == 11'b10011100001)//为 1249 的 2 进制，用来 2500 分频，得到 20000hz
        begin
            count <= 11'b0;
            FIN = ~FIN;
        end
        else
        begin
            count <= count + 11'b1;
        end
    end
    assign CLK1 = FIN;
endmodule
```

- 输入模块

从矩阵键盘读取输入的信息（解决了长按健和按键抖动的问题）。

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。

重点是防抖和长按健的问题。

我觉得我的思路很清晰，**首先为了解决防抖问题**，首先想明白抖动，就是按下的时候和释放的时候两个时候有抖动，那么我们只要取稳定时候的按键信息不就可以了吗！那么我就设置一个数，再给他一个上限，（这里为了辨别何时“完全”按下和“完全”释放，我取了 release_p 和 press_p 两个信号），每次检测到按下时就让它加 1，检测到没有按下的时候就清 0，这样达到上限的时候就必然是连续加 1 的过程，也就是连续检测到按下的过程，这就说明此刻已经“完全”按下，亦即进入稳定时候。**而为了解决长按健问题**，长按健问题，也就是要避免一直按下时多次读入信息，如一直按下 3 而输出金额为 33 这类情况。为此，我加入了数据使能控制端 EN（上升沿时代表数据此刻有效）。这样一来只有 EN 上升沿的时候按键是有效的，就避免了长按键时多次读入。

源代码如下：

```
module Key_input(CLK, R, C, DATA, EN);
    input [3:0] R;
    input CLK;
    output reg [3:0] C;
    output reg [3:0] DATA;
    output reg EN;
    reg judge;//用来判断按键是否按下，judge==0 表示没有按键按下
    reg [11:0] release_p;//release_prevent，释放防止抖动
    reg [11:0] press_p;//press_prevent，按下防止抖动
    reg [1:0] A;//用来遍历列值
    always @(posedge CLK)
    begin
        if   (!judge)//没有按下的时候,需要防止按下的抖动
        begin
            if   (R == 4'b1111)
                begin
                    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;
                     endcase;
                end
            else//这时候遍历了列值，行值又不全为 1，说明有一个按键按下
```

```
            begin
                release_p <= 0;
                if    (press_p != 12'b111111111111)
                    press_p <= press_p + 12'b1;
                else//这个状态表明，press_p 已经到达 32，表明按键处于按下的稳定时
刻
                begin
                    press_p <= 0;
                    EN <= 1;
                    judge <= 1;
                    //用来输出 DATA
                    if (R == 4'b0111 && C == 4'b0111)
                        DATA <= 4'b0001;
                    if (R == 4'b0111 && C == 4'b1011)
                        DATA <= 4'b0010;
                    if (R == 4'b0111 && C == 4'b1101)
                        DATA <= 4'b0011;
                    if (R == 4'b0111 && C == 4'b1110)
                        DATA <= 4'b0100;
                    if (R == 4'b1011 && C == 4'b0111)
                        DATA <= 4'b0101;
                    if (R == 4'b1011 && C == 4'b1011)
                        DATA <= 4'b0110;
                    if (R == 4'b1011 && C == 4'b1101)
                        DATA <= 4'b0111;
                    if (R == 4'b1011 && C == 4'b1110)
                        DATA <= 4'b1000;
                    if (R == 4'b1101 && C == 4'b0111)
                        DATA <= 4'b1001;
                    if (R == 4'b1101 && C == 4'b1011)
                        DATA <= 4'b0000;
                    if (R == 4'b1101 && C == 4'b1101)
                        DATA <= 4'b1010;
                    if (R == 4'b1101 && C == 4'b1110)
                        DATA <= 4'b1011;
                    if (R == 4'b1110 && C == 4'b0111)
                        DATA <= 4'b1100;
                end
            end
        end
        if (judge)
        begin
            if (R == 4'b1111)//这个状态表明，按键已经开始释放
            begin
```

```
                press_p <= 0;
                if (release_p != 12'b111111111111)
                    release_p <= release_p + 12'b1;
                else//这个状态表明，release_p 已经到达 32，表明按键释放完毕
                begin
                    release_p <= 0;
                    judge <= 0;
                    EN <= 0;//完全释放才变化使能端
                end
            end
            else
            begin
                release_p <= 0;
            end
        end
    end
endmodule
```

- 控制模块

处理输入模块得到的信息，整理成投币数额和充电时间（即数码管上四位数字）的数，以及当前状态的信息（这是为了解决初始状态要保证数码管不显示从而加入的）。

由于要求里写了，**在涉及到时序的 always 语句只能由 clk 或者 reset 触发，不能出现其他的逻辑量**。但是为了解决长按健问题，我需要在 EN 的上升沿才取数据有效，那么怎么办呢？于是我设了一个“reg [1:0] EN_pre;//CLK 上升沿到来时上一个 EN 的值”，这样一来，只要 EN_pre 为 0，EN 是 1，不就说明到达上升沿了吗？至于取了两位是因为有 clk 脉冲，每次用 EN_pre[1]或者 EN_pre[1]充当真正的前一个 EN 值。

其他的就很清楚了，通过输入的 DATA 值进行判断状态是否跳转。

但是**还有个难点就是倒计时和 10s 无操作的问题**。

对于 **10s 无操作的问题**，我取了一个定时器 timer，每次输入都先把 timer 置 0，然后通过 data 去判断是否让 timer 变 1，只要 timer 变 1 就开启倒计时模式（方法和分频、防抖动类似，很简单不在阐述）。每次 timer 为 0 的时候就把计数清 0。

```
if (timer == 0)
            count_wait <= 0;
        else
        begin
                if (count_wait < time_wait)
                begin
                    count_wait <= count_wait + 18'b1;
                    change <= 0;
                end
                else
                begin
                    count_wait <= 0;
                    timer <= 0;
                    change <= 1;
```

```
                    //current_state <= S0;
                end
        end
```

而至于倒计时问题，有了前面的经验就更简单了，只要判断一下当前的状态是否为 S2（即倒计时状态）就行了。有个细节就是当 T1 为 0，T0 为 1 时，要把 M1M0T1T0 同时清 0，并且更改状态为 S1，且只有这个时刻才能启动 10s 定时器。

源代码如下：

```
module Control(CLK, EN, DATA, M1, M0, T1, T0, state);
    input CLK;
    input EN;
    input [3:0] DATA;
    output reg [3:0] M1;
    output reg [3:0] M0;
    output reg [3:0] T1;
    output reg [3:0] T0;
    parameter [1:0] S0 = 2'b00;
    parameter [1:0] S1 = 2'b01;
    parameter [1:0] S2 = 2'b10;
    //parameter time_down = 200;//20000hz,此为 1s
    parameter time_wait = 200000;//20000hz,此为 10s
    output reg [1:0] state;
    reg [1:0] current_state;
    reg [17:0] count_down;
    reg [17:0] count_wait;
    reg judge;
    reg [1:0] EN_pre;//CLK 上升沿到来时上一个 EN 的值
    reg flag;
    reg timer;
    reg change;
    always @(posedge CLK)
    begin
        state <= current_state;

        if (timer == 0)
            count_wait <= 0;
        else
        begin
                if (count_wait < time_wait)
                begin
                    count_wait <= count_wait + 18'b1;
                    change <= 0;
                end
                else
                begin
```

```
                count_wait <= 0;
                timer <= 0;
                change <= 1;
                //current_state <= S0;
            end
end
flag <= (flag + 1) % 2;
if(flag)
begin
    EN_pre[1] <= EN;
    judge <= (EN_pre[0] != EN && EN);
end
else
begin
    EN_pre[0] <= EN;
    judge <= (EN_pre[1] != EN && EN);
end
case (current_state)
S0:
    begin
    if (judge)
    begin
        if (DATA == 4'b1010)//输入了开始
        begin
            timer <= 1;
            count_wait <= 0;
            current_state <= S1;
            M1 <= 0;
            M0 <= 0;
            T1 <= 0;
            T0 <= 0;
        end
    end
    end
S1:
    begin
    if (change)
        current_state <= S0;
    if (judge)
    begin
        timer <= 0;
        if (DATA < 4'b1010)//输入了数字
        begin
            timer <= 0;
```

```
                if (M1 == 0)
                begin
                    if (10 * M0 + DATA > 20)
                    begin
                        count_wait <= 0;
                        timer <= 1;
                        M1 <= 2;
                        M0 <= 0;
                        T1 <= 4;
                        T0 <= 0;
                    end
                    else
                    begin
                        count_wait <= 0;
                        timer <= 1;
                        T1 <= (20 * M0 + (2 * DATA))/10;
                        T0 <= (2 * DATA) % 10;
                        M1 <= M0;
                        M0 <= DATA;
                    end
                end
            end
            else if (DATA == 4'b1011)//输入了清零
            begin
                count_wait <= 0;
                timer <= 1;
                T1 <= 0;
                T0 <= 0;
                M1 <= 0;
                M0 <= 0;
            end
            else if (DATA == 4'b1100)//输入了确认
            begin
                timer <= 0;
                count_wait <= 0;
                if (M1 != 0 || M0 != 0)
                begin
                    timer <= 0;
                    current_state <= S2;
                end
            end
        end
        end
    S2://充电状态，下面为倒计时代码
```

```
        begin
            if (change)
                current_state <= S0;
            if (count_down < 2000)
                count_down <= count_down + 18'b1;
            else
                count_down <= 0;
            if (count_down == 2000 - 1)//1 秒钟倒计时结束，数字减一次 1
            begin
                if (T1 == 0 && T0 == 4'b1)//减到最后一下的时候，要回到开始状态
                begin
                    T0 <= 0;
                    M1 <= 0;
                    M0 <= 0;
                    current_state <= S1;
                    timer <= 1;
                    count_wait <= 0;
                end
                else if (T1 != 0 || T0 != 0)
                begin
                    if (T0 != 0)
                        T0 <= T0 - 4'b1;
                    else if (T1 != 0)
                    begin
                        T1 <= T1 - 4'b1;
                        T0 <= 4'b1001;
                    end
                end
            end
        end
        default:
            current_state <= S0;
        endcase
    end
endmodule
```

还有几个需要注明的地方：

i. **多出来一个 state 信号而不直接用 current_state 信号，以及用了 case 语句，都是为了状态机服务的。亲测只有这样才能让 quartus 软件生成状态转换图（即 current_state 不能作为输出信号）**

ii. **定时器以及倒计时的计数的上限的设定都是根据 CLK 的频率计算得到的。频率为 20000hz，那么 10s 也就是 200000hz，如此。**

- 显示模块

  从控制模块接受信息，从而显示在数码管上。

  为了解决显示问题，由于题目要求代码实现，故我手写了 my7448 代码，如下：

```
module my7448(NUM,OA,OB,OC,OD,OE,OF,OG);
input [3:0]NUM;
output OA,OB,OC,OD,OE,OF,OG;
reg OA,OB,OC,OD,OE,OF,OG;
always @(NUM)
      begin
      case(NUM)
            4'h0:
              begin
              OA <= 1'b1;
              OB <= 1'b1;
              OC <= 1'b1;
              OD <= 1'b1;
              OE <= 1'b1;
              OF <= 1'b1;
              OG <= 1'b0;
              end
            4'h1:
              begin
              OA <= 1'b0;
              OB <= 1'b1;
              OC <= 1'b1;
              OD <= 1'b0;
              OE <= 1'b0;
              OF <= 1'b0;
              OG <= 1'b0;
              end
            4'h2:
              begin
              OA <= 1'b1;
              OB <= 1'b1;
              OC <= 1'b0;
              OD <= 1'b1;
              OE <= 1'b1;
              OF <= 1'b0;
              OG <= 1'b1;
              end
            4'h3:
              begin
              OA <= 1'b1;
              OB <= 1'b1;
              OC <= 1'b1;
              OD <= 1'b1;
              OE <= 1'b0;
```

```
  OF <= 1'b0;
  OG <= 1'b1;
  end
4'h4:
  begin
  OA <= 1'b0;
  OB <= 1'b1;
  OC <= 1'b1;
  OD <= 1'b0;
  OE <= 1'b0;
  OF <= 1'b1;
  OG <= 1'b1;
  end
4'h5:
  begin
  OA <= 1'b1;
  OB <= 1'b0;
  OC <= 1'b1;
  OD <= 1'b1;
  OE <= 1'b0;
  OF <= 1'b1;
  OG <= 1'b1;
  end
4'h6:
  begin
  OA <= 1'b1;
  OB <= 1'b0;
  OC <= 1'b1;
  OD <= 1'b1;
  OE <= 1'b1;
  OF <= 1'b1;
  OG <= 1'b1;
  end
4'h7:
  begin
  OA <= 1'b1;
  OB <= 1'b1;
  OC <= 1'b1;
  OD <= 1'b0;
  OE <= 1'b0;
  OF <= 1'b0;
  OG <= 1'b0;
  end
4'h8:
```

```
  begin
  OA <= 1'b1;
  OB <= 1'b1;
  OC <= 1'b1;
  OD <= 1'b1;
  OE <= 1'b1;
  OF <= 1'b1;
  OG <= 1'b1;
  end
 4'h9:
  begin
  OE <= 1'b1;
  OA <= 1'b1;
  OB <= 1'b1;
  OC <= 1'b1;
  OD <= 1'b1;
  OE <= 1'b0;
  OF <= 1'b1;
  OG <= 1'b1;
  end
 4'ha:
  begin
  OA <= 1'b1;
  OB <= 1'b1;
  OC <= 1'b1;
  OD <= 1'b1;
  OE <= 1'b1;
  OF <= 1'b1;
  OG <= 1'b0;
  end
 4'hb:
  begin
  OA <= 1'b1;
  OB <= 1'b1;
  OC <= 1'b1;
  OD <= 1'b1;
  OE <= 1'b1;
  OF <= 1'b1;
  OG <= 1'b0;
  end
 4'hc:
  begin
  OA <= 1'b0;
  OB <= 1'b0;
```

```
            OC <= 1'b0;
            OD <= 1'b0;
            OE <= 1'b0;
            OF <= 1'b0;
            OG <= 1'b0;
            end
          4'hd:
            begin
            OA <= 1'b0;
            OB <= 1'b0;
            OC <= 1'b0;
            OD <= 1'b0;
            OE <= 1'b0;
            OF <= 1'b0;
            OG <= 1'b0;
            end
          4'he:
            begin
            OA <= 1'b1;
            OB <= 1'b0;
            OC <= 1'b0;
            OD <= 1'b1;
            OE <= 1'b1;
            OF <= 1'b1;
            OG <= 1'b1;
            end
          4'hf:
            begin
            OA <= 1'b1;
            OB <= 1'b0;
            OC <= 1'b0;
            OD <= 1'b0;
            OE <= 1'b1;
            OF <= 1'b1;
            OG <= 1'b1;
            end
    endcase
end endmodule
```

其实显示模块很简单，无非是把控制模块得到的 M1、M0、T1、T0 通过 7448 再接入 SEG 上即可。但是**为了解决初始时数码管全灭**，我特意引入了 current_state 这个输入信号，这样当它是 S0（即初始状态）时，就让数码管全灭即可。

源代码如下：

```
module show(CLK, M1, M0, T1, T0, current_state, DIG, SEG);
    input CLK;
```

```
    input [3:0] M1;
    input [3:0] M0;
    input [3:0] T1;
    input [3:0] T0;
    input [1:0] current_state;
    output wire [3:0] DIG;
    output reg [6:0] SEG;
    wire CLK1;
    wire TEMP;
    assign TEMP = CLK;
    FDIV2 fd_1(TEMP, CLK1);
    assign DIG[3] = (CLK == 1 && CLK1 == 1);
    assign DIG[2] = (CLK == 1 && CLK1 == 0);
    assign DIG[1] = (CLK == 0 && CLK1 == 1);
    assign DIG[0] = (CLK == 0 && CLK1 == 0);
    wire [6:0] temp_1;
    wire [6:0] temp_2;
    wire [6:0] temp_3;
    wire [6:0] temp_4;
    my7448 my_1(M1, temp_1[0], temp_1[1], temp_1[2], temp_1[3], temp_1[4], temp_1[5],
temp_1[6]);
    my7448 my_2(M0, temp_2[0], temp_2[1], temp_2[2], temp_2[3], temp_2[4], temp_2[5],
temp_2[6]);
    my7448 my_3(T1, temp_3[0], temp_3[1], temp_3[2], temp_3[3], temp_3[4], temp_3[5],
temp_3[6]);
    my7448 my_4(T0, temp_4[0], temp_4[1], temp_4[2], temp_4[3], temp_4[4], temp_4[5],
temp_4[6]);
    always @(*)
    begin
        if (current_state == 2'b00)//如果状态是初始状态
        begin
            SEG <= 0;
        end
        else
        begin
            if(DIG[3])
            begin
                SEG[0] <= temp_1[0];
                SEG[1] <= temp_1[1];
                SEG[2] <= temp_1[2];
                SEG[3] <= temp_1[3];
                SEG[4] <= temp_1[4];
                SEG[5] <= temp_1[5];
                SEG[6] <= temp_1[6];
```

```
                end
                if(DIG[2])
                begin
                    SEG[0] <= temp_2[0];
                    SEG[1] <= temp_2[1];
                    SEG[2] <= temp_2[2];
                    SEG[3] <= temp_2[3];
                    SEG[4] <= temp_2[4];
                    SEG[5] <= temp_2[5];
                    SEG[6] <= temp_2[6];
                end
                if(DIG[1])
                begin
                    SEG[0] <= temp_3[0];
                    SEG[1] <= temp_3[1];
                    SEG[2] <= temp_3[2];
                    SEG[3] <= temp_3[3];
                    SEG[4] <= temp_3[4];
                    SEG[5] <= temp_3[5];
                    SEG[6] <= temp_3[6];
                end
                if(DIG[0])
                begin
                    SEG[0] <= temp_4[0];
                    SEG[1] <= temp_4[1];
                    SEG[2] <= temp_4[2];
                    SEG[3] <= temp_4[3];
                    SEG[4] <= temp_4[4];
                    SEG[5] <= temp_4[5];
                    SEG[6] <= temp_4[6];
                end
            end
        end
endmodule
```

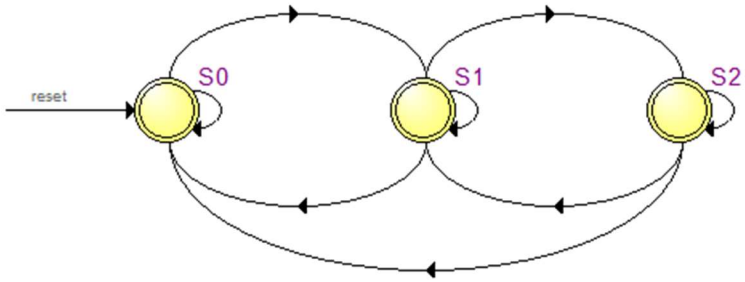
注：我为什么大费周折用了 temp_1~ temp_4 这些信号呢？是因为当我把已写好的外部模块（如 my7448）放入 always 内部时，编译总是报错。经过我很长时间的尝试才发现要写在 always 外部。于是想到使用 temp_1~ temp_4 来代替。

四、 状态转换图及其说明

S0 代表初始状态，S2 代表倒计时状态，S1 代表其他情况的状态（包括开始状态、输入状态等）。

状态之间转换关系如下：

- S0→S1：按下“开始”键。这时数码管显示为“0000”，为开始状态，可以按下各种按键。
- S1→S2：投币金额不为 0 时按下“确认”键。
- S2→S1：充电倒计时结束。这时数码管显示为“0000”，即为开始状态。
- S1→S0：任意时刻，10s 无操作即回到 S0 状态（数码管全灭）
- S0→S0：未按下“开始”键。
- S1→S1：投币金额不为 0 时未按下“确认”键，且任意 10s 内有操作。
- S2→S2：保持状态直到充电倒计时结束

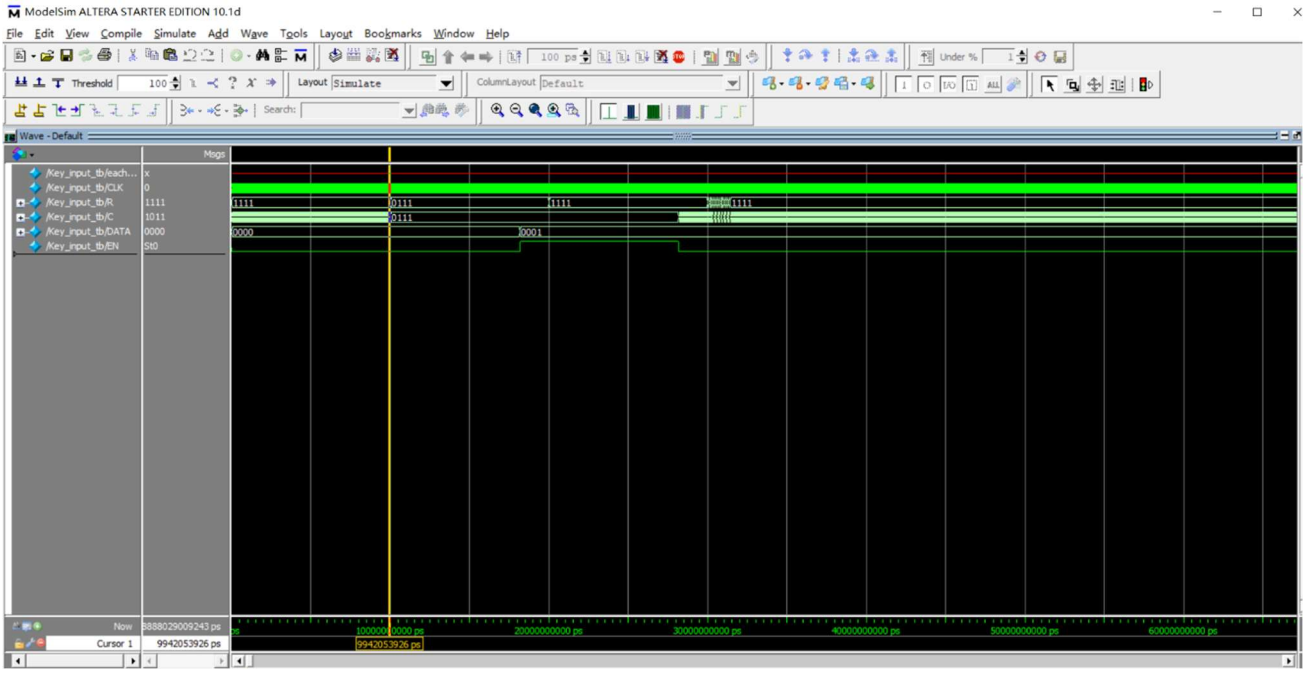


五、 仿真波形图及其分析说明

1. 输入模块仿真

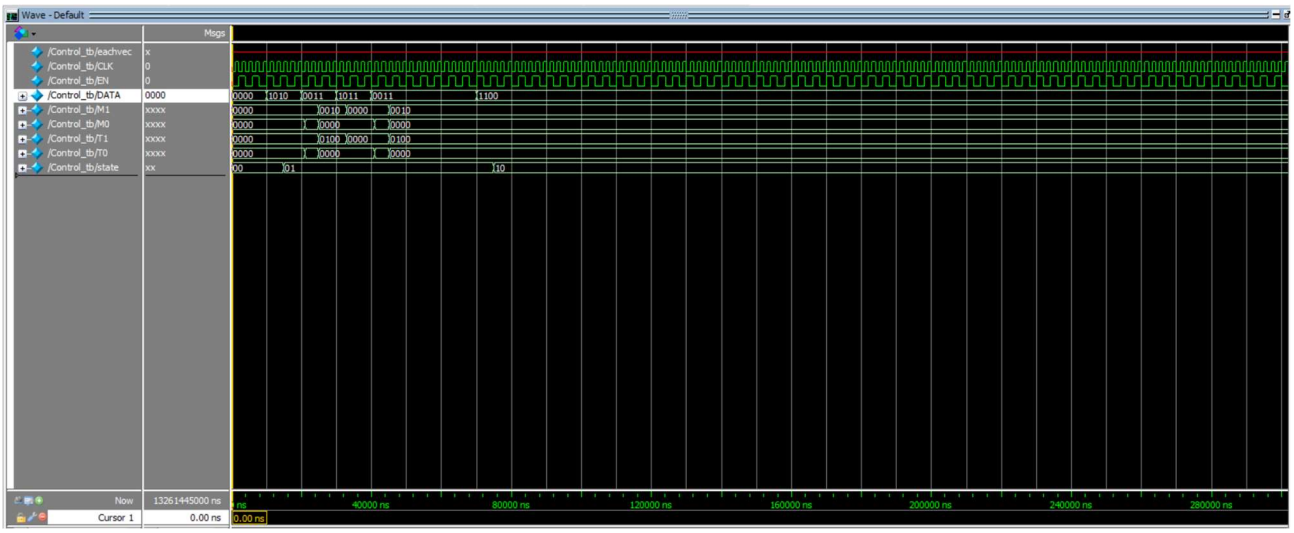
仿真图如下。

从图中可看出，当 R=0111，C=0111 时，data 值为 0001，这是因为第一行第一列的值为 1，而且黄色线所示位置为 R=0111，C=0111 的时刻，但是过了一段时间 data 才变化，这是由于我的防抖动设计，这样也能看出。而从 EN 的变化可以看出，每次只取 EN 的上升沿能十分有效地解决长按键问题。



2. 控制模块仿真

如下图，当 DATA 为 0000 时，四个数字都为 0，且数字的变动是根据 EN 上升沿位置而来的。当 DATA 为 1010 时，代表输入“开始”按键，state 从 00 变成 01，表示当前状态的变化，之后每当 EN 上升沿到来，M1M0T1T0 四个数字的值就发生变化，且超过 20 时变成了 20 和 40，都表明电路设计的成功。当 DATA 为 1011 时，代表输入“清零”按键，也发现数字有效地变成了 0000。最后读取数字后，取 DATA 为 1100，代表输入“确认”按键，也发现了 state 从 01 变成了 10，表明此刻状态变成了倒计时状态。



从下图可看出倒计时状态的变动：

从下图可以看到，减到最后，M1M0T1T0 四个数字同时变成了 0000，表明倒计时结束，并且发现 state 变成了 01，表示进入了开始状态。这些都很成功地验证了电路设计的正确性。

六、 设计和调试中遇到的问题及其解决方法

这次的 EDA 大作业我觉得非常困难，但是正因如此，最终才有了做出来的兴奋感和成就感。

当然遇到了不少问题，如下：

1) 矩阵键盘输入太敏感，数字总是跳变

解决方法：

把防抖的设计中的两个上限值提高，(事实上最终我提高到了 4000 多才有效地解决了，这也是基于我频率过高的原因)。

2) 按下矩阵键盘总是没有反应

解决方法：

琢磨了很久才发现是分频器写错了，刚开始写的代码是：每次达到上限的那个时刻才把输出的信号 clk1 变成 1，这样做导致了占空比非常非常低，导致根本没有用。最后用了

FIN = ~FIN

这样的代码予以解决。

3) 刚连接 FPGA 板子时显示了 0000

解决方法：

正确情况应该是数码管全灭，这是因为 M1M0T1T0 的初值就是 0 导致的。为此我特意引入一个灭零（熄灭）信号 current_state，这样，每当状态为初始状态时，就让数码管灭掉就好了。

4) 仿真时显示 HIZ

解决方法：

这个我弄了老半天，最后靠室友的帮助我才发现是我的 setting 设置没有弄好。最后放弃了生成 vt 模板的方法，转而自己写.v 的测试文件了。

5) 引入了 my7448 等写好的.v 的器件编译报错

解决方法：

这是因为当我把已写好的外部模块（如 my7448）放入 always 内部时，编译总是报错。经过我很长时间的尝试才发现要写在 always 外部。最终想到使用 temp_1~ temp_4 来代替。

6) 生成不了状态转换图

解决方法：

这个又让我伤脑筋。

忙活半天最后发现了，不直接用 current_state 信号做为模块的输出信号，于是引入了一个 state 信号，再让 state 恒等于 current_state。并且使用了 case 语句，而且要在最后加上 default，即使你觉得不加也没问题。

七、 收获和一些感想

毫无疑问，历经这么长时间，完成了一个目前对我来说有些困难的项目，收获是巨大的，成就感也是很强烈的。在最初的写一个很简单的代码都会编译出错（这个的原因是 2'b 的'打错了导致识别不出来而报错，由于这个问题过于简单所以没有在第六项中阐述），到最后写了加起来好几百行代码并且最后完成了大作业项目，真是有一丝感慨。

除了编写 Verilog 代码的能力，还熟悉了矩阵键盘的原理，以及原理图和代码的结合方法，以及第三方仿真。收获颇丰。

我们遇到什么困难都不要怕，有什么困难好好分析好好修改，认真完成就可以了。也写给未来的自己，希望以后再遇大作业也能微笑面对，不畏难。