《停车管理系统： FPGA实现》

期末报告

课　　程：	数字逻辑 (CS207)
授课教师：	Georgios Theodoropoulos
姓　　名：	李子南；唐昕宇；何泽安
学　　号：	12011517；12011439；12011323
日　　期：	2021 年 12 月 31 日

开发计划

1 小组选题

停车场收费系统

2 成员分工及贡献百分比

成员	分工	百分比
李子南 12011517	主状态机的实现及debug	33.3%
唐昕宇 12011439	显示模块的实现、键盘输入模块的实现及debug	33.3%
何泽安 12011323	蜂鸣器的实现、键盘输入模块的实现	33.3%

3 进度安排及执行记录

时间	进度安排	执行
第一周	讨论确定总体框架，分配各个模块，确定输入输出端口	完成
第二周	开始推进各个模块	完成，在进行中
第三周	开始推进各个模块	主状态机完成但未测试、蜂鸣器完成并测试成功
第四周	各模块单独测试，并尝试初步进行连接	显示模块完成并测试成功
第五周	连接完成开始debug	键盘输入模块完成并测试成功，连接完成开始总体debug及测试
第六周	debug完成及报告书写，准备并完成答辩	完成

需求分析

停车场系统可大致分为三个部分：基础功能，vip界面和管理员界面

0.1 进场离场

进场可以选择分区，进场之后需要记录用户id和进场时间，所以在进场的时候可以检测用于存储用户ID的寄存器是否全为0，以此判断车位是否为空。

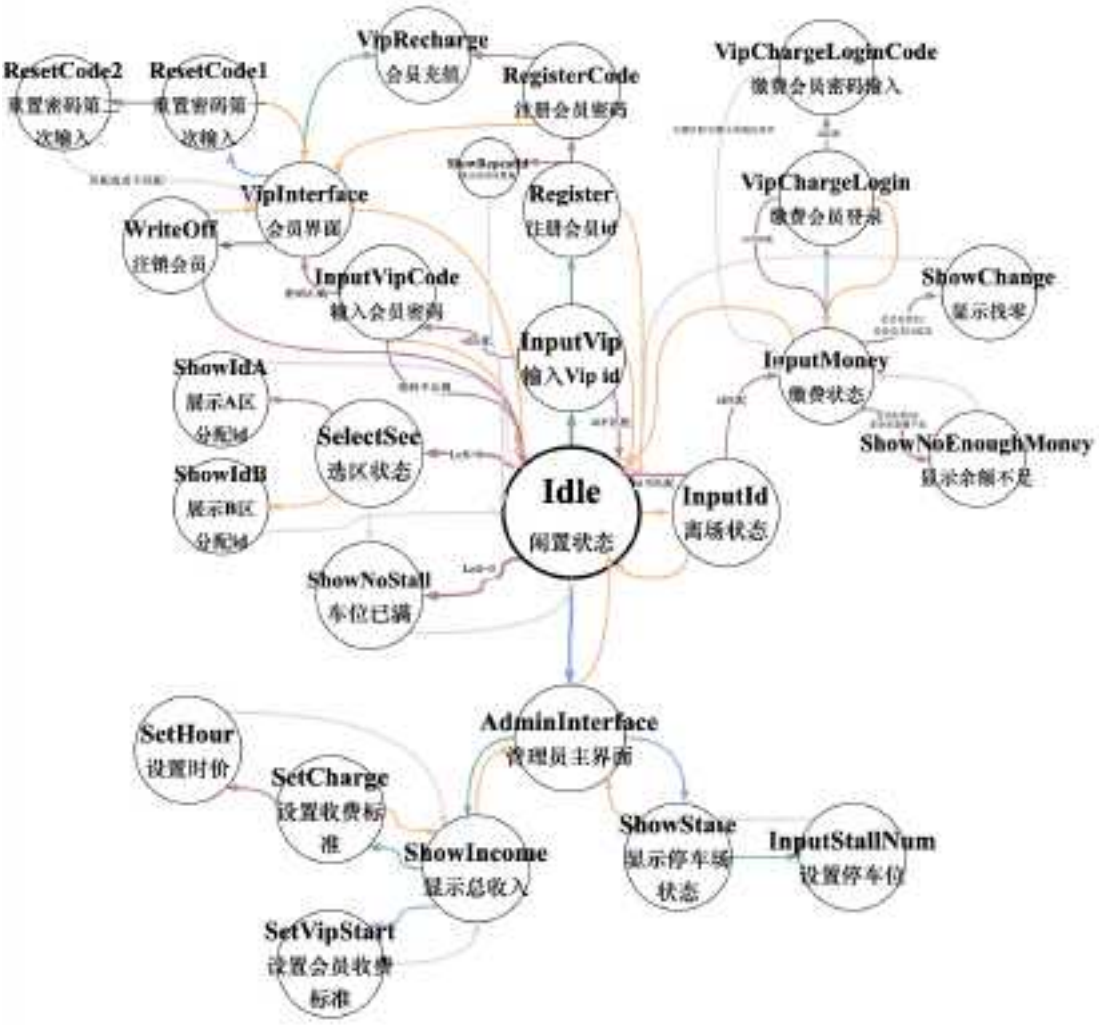
离场的时候，需要匹配输入和用户id，并且依据起始价格，时价和用户停留时间用于计算应付金额。这里需要有一个自增变量储存现在的时间。

0.2 VIP界面：

- 注册：比较输入和已存在vipid，如果存在相同就退出；
- 登录：需要同时匹配vipId和vipCode；
- vip充值，注销：直接对变量进行操作；
- VIP付款：需要判断用户是否是VIP，需要一个标识符。是则将起步价改为vip起步价。

0.3 管理员界面：

- 显示平均停留时间：求有车的车位的平均停留时间；
- 设置起步价，时价，vip起步价：直接修改变量；
- 重置系统：对所有变量赋初值；
- 设置车位：需要一个变量标识各个车位是否可用，相关case判断该变量；
- 显示总收入：需要一个变量储存总收入，每次离场增加金额。



主状态机

主要输入输出使用硬件

顶层模块

1 接口和变量描述

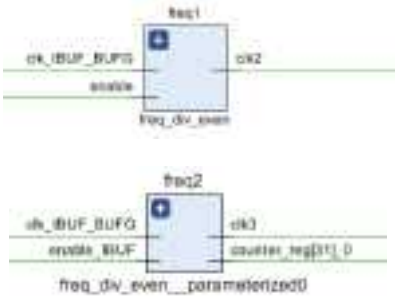
顶层模块接收一个时钟信号 clk (连接 Y18 管脚)，一个使能信号 enable (连接一个拨码开关)，一个音乐使能信号 music_en (链接一个拨码开关)，5位按钮输入 button，4位键盘输入 row，4位键盘输出 col，8位显示输出DIG 和 Y，一位音乐输出 music_frac_ext（连接蜂鸣器）。

2 子模块

2.1 分频器

```
freq_div_even #(100000000,31) freq1(clk, enable, clk2);//1s
freq_div_even #(500000, 31) freq2(clk, enable, clk3);//0.005s
```

clk2, clk3 是一位 wire 类型变量，频率分别为 1Hz 和 200Hz。



2.2 按钮除抖

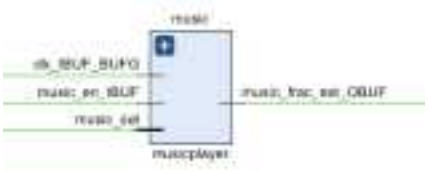
```
ButtonVibration BV(clk, enable, button, buttonVib);
```

buttonVib是5位wire类型变量，代表除抖后的按钮信号。

2.3 音乐播放模块

```
musicplayer music(clk, music_sel,music_en, music_frac_ext);
```

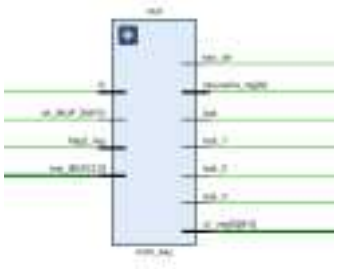
music_sel为3位wire类型变量，代表选择音乐播放模式，music_en为音乐使能信号，music_frac_ext为音乐输出。



2.4 键盘输入

```
multi_key mult(buttonVib[2], clk, 2, row, col, DIG1, Y1, finish, key);
```

buttonVib[2] 为输入模块的使能信号（因为要用输入模块之前必定会按下buttonVib[2]），常数 2 为最多连续输入位数，DIG1，Y1 为键盘显示输出（不调用），finish 为一位 wire 变量，代表输入是否完成，key 为 30 位wire 变量，为键盘输出（最多支持输入 5 位，即每位 6 bit）。



2.5 主状态机

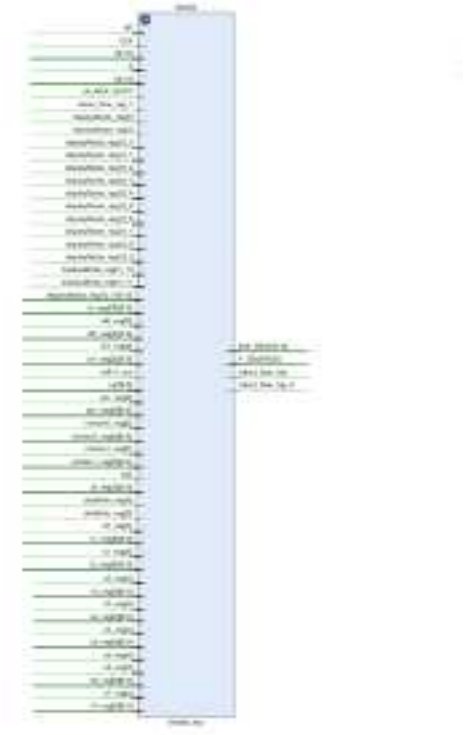
```
FSM fsm(buttonVib, clk2, clk3, enable, key, finish, displayMode, stallLeft,start, per, id0,
id1, remain0, remain1, x0, x1, x2, x3, x4, x5, x6, x7, music_sel);
```

displayMode, stallLeft, per, id0, id1, remain0, remain1, x0, x1, x2, x3, x4, x5, x6, x7, music_sel 均为 FSM 的输出，为 wire 类型变量。

2.6 主显示模块

```
display_top display(displayMode, clk, stallLeft, start, per, id0, id1, remain0, remain1,
x7, x6, x5, x4, x3, x2, x1, x0, DIG, Y);
```

DIG，Y为顶层模块输出。



3 约束文件

3.1 端口绑定描述

```
set_property PACKAGE_PIN Y18 [get_ports clk]//时钟信号

set_property PACKAGE_PIN P5 [get_ports button[0]]//按钮上键
set_property PACKAGE_PIN P2 [get_ports button[1]]//按钮下键
set_property PACKAGE_PIN P1 [get_ports button[2]]//按钮左键
set_property PACKAGE_PIN R1 [get_ports button[3]]//按钮右键
set_property PACKAGE_PIN P4 [get_ports button[4]]//按钮中键

set_property PACKAGE_PIN W4 [get_ports music_en]//音乐开关

set_property PACKAGE_PIN Y9 [get_ports enable]//总开关

set_property PACKAGE_PIN M2 [get_ports {col[3]}]//键盘列扫描
set_property PACKAGE_PIN K6 [get_ports {col[2]}]
set_property PACKAGE_PIN J6 [get_ports {col[1]}]
set_property PACKAGE_PIN L5 [get_ports {col[0]}]
set_property PACKAGE_PIN K4 [get_ports {row[3]}]//键盘行扫描
set_property PACKAGE_PIN J4 [get_ports {row[2]}]
set_property PACKAGE_PIN L3 [get_ports {row[1]}]
set_property PACKAGE_PIN K3 [get_ports {row[0]}]

```

```
set_property PACKAGE_PIN A18 [get_ports {DIG[7]}] //单个七段数码显示管的使能信号
set_property PACKAGE_PIN A20 [get_ports {DIG[6]}]
set_property PACKAGE_PIN B20 [get_ports {DIG[5]}]
set_property PACKAGE_PIN E18 [get_ports {DIG[4]}]
set_property PACKAGE_PIN F18 [get_ports {DIG[3]}]
set_property PACKAGE_PIN D19 [get_ports {DIG[2]}]
set_property PACKAGE_PIN E19 [get_ports {DIG[1]}]
set_property PACKAGE_PIN C19 [get_ports {DIG[0]}]
set_property PACKAGE_PIN E13 [get_ports {Y[7]}] //单个七段数码显示管上的引脚
set_property PACKAGE_PIN C15 [get_ports {Y[6]}]
set_property PACKAGE_PIN C14 [get_ports {Y[5]}]
set_property PACKAGE_PIN E17 [get_ports {Y[4]}]
set_property PACKAGE_PIN F16 [get_ports {Y[3]}]
set_property PACKAGE_PIN F14 [get_ports {Y[2]}]
set_property PACKAGE_PIN F13 [get_ports {Y[1]}]
set_property PACKAGE_PIN F15 [get_ports {Y[0]}]
```

主状态机功能描述

1 接口和变量描述

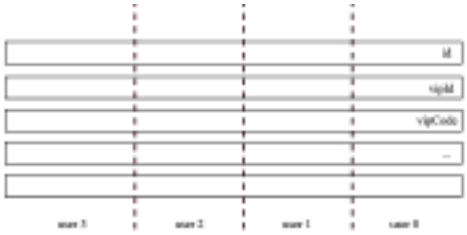
主状态机接收6个输入，分别为 时钟信号 **clk2** (1Hz)， **clk3** (500Hz)，使能信号 **enable**，5-bit按钮 **button**，30-bit键盘输入 **key**，键盘输入结束信号 **finish**。

输出16个信号，分别为显示模式 **displayMode**，剩余车位 **left**，起始价格 **st**，时价 **per**， VIP用户ID **{id1,id0}**，VIP余额 **{remain1,remain0}**，以及七段数码显示管 **{x7,x6,x5,x4,x3,x2,x1,x0}**。

状态机现态和次态使用6位寄存器储存，并且将每个状态设为参数，便于理解。主要状态如下所示：

```
parameter Idle = 6'b000001;
parameter InputVip = 6'b000010;
parameter Register = 6'b000011;
parameter VipRecharge = 6'b000100;
parameter VipInterface = 6'b000101;
```

在设计用户与板上数据的储存关系时，我们使用多个数组分别保管一个“用户”数据结构的成员变量，同一用户在多个数组内对应的数据区下标一致。方便起见，我们将每个成员变量存为8位，由于限制了用户最多为四位， **id, vipId, vipCode, vipBalance, timeStart** 均为32位寄存器。



- **start, vipStart, start_, hour** 为收费标准，均为8位寄存器，分别代表起步价，vip起步价，起步价缓存 (用于计算账单)，时价。

- **Income** 是8位寄存器，用于储存总收入，**counter** 是16位寄存器，用于倒计时。

- **flag** 和 **vipFlag** 均为5位寄存器，用于标记当前用户信息所处位置。

- **stallAble** 为4位寄存器，用于判断车位是否可用(供管理员设置车位数量)。

- **clock** 为8为寄存器，用于储存现在时间，随 clk2 自增，即每秒 +1。**duration** 为8位寄存器，用于储存当前用户停留时间，方便计算账单。

- **change** 为8位寄存器，用于储存找零。

- **isVip** 是1位寄存器，用于在付款界面判断是否为 VIP 用户。

- **codeCache** 是8位寄存器，用于在VIP用户重置密码时判断前后两次是否一致。

- **bill** 为8位wire，即账单，是子模块CountMoney的输出。

2 子模块

核心代码及其例化

```
module CountMoney (
  input  [7:0]   duration, start, hour,
  output [7:0]   bill
);
assign bill = start + ((duration/ 'd10) * hour);
endmodule

CountMoney coun(duration,start_,hour,bill);  // 例化
```

根据输入计算账单, duration 为停车时间，start 为起步价，hour 为时价， bill 为账单。

3 状态机计时

核心代码

```
always @(posedge clk2 or negedge enable) begin
    if (!enable) clock = 8'b0;
    else clock = clock + 1;
end  // 当使能信号 enable 为1时，clock 随 clk2 自增
```

4 状态更新

核心代码

```
always @(posedge clk3 or negedge enable) begin
    if (!enable) currentState <= Idle;
    else currentState <= nextState;
end  // 使能信号 enable 为 1 时，将 nextState 的值赋给 currentState
```

5 状态切换

核心代码

```
always @(*) begin
    begin
        case (currentState)
            Idle: begin  // 闲置状态
                case (button)  // 按钮触发
                    buttonLeft:
                        if (left != 3'b0) begin
                            nextState = SelectSec;  // 选区
                        end
                        else begin
                            nextState = ShowNoStall;  // 显示车位不够
                        end
                    buttonRight:
                        nextState = InputId;  // 离场
                    buttonUp:
                        nextState = InputVip;  // vip 登录
                    buttonDown:
                        nextState = AdminInterface;  // 管理员界面
                    default:
                        nextState = Idle;
                endcase
            end  // more states are omitted here
        endcase
    end
end
```

主状态机为三段式设计，状态切换部分敏感于 * （Vivado自动推断），会根据不同的输入和条件判断下一个状态。

6 各状态输出/变量更改

```
always @(posedge clk3 or negedge enable) begin
    if (!enable) begin  // 一系列标识置零
        displayMode <= 'b0;
        id <= 32'b0;
        vipId <= 32'b0;
        vipCode <= 32'b0;
        vipBalance <= 32'b0;
        timeStart <= 32'b0;
        timeDuration <= 4'b0;
        start <= 8'b00000001;
        vipStart <= 8'b00000001;
        start_ <= 8'b00000001;
        hour <= 8'b00000001;
```

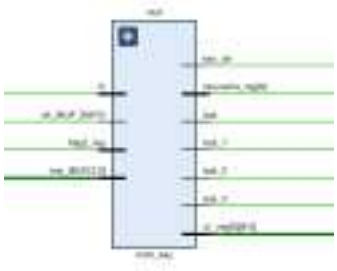
```
        Income <= 8'b0;
        counter <= 16'b0;
        flag <= 5'b0;
        vipFlag <= 5'b0;
        clock <= 8'b0;
        duration <= 8'b0;
        change <= 8'b0;
        isVip <= 1'b0;
        codeCache <= 8'b0;
    end
    else begin
        case (currentState)
            Idle: begin  // 闲置状态
                displayMode <= 'd1;
                st <= start;
                per <= hour;
            end
            InputMoney: begin
                displayMode <= 3;
                x0 <= bill[5:0];
                x1 <= 6'b111_111;
                x2 <= 6'b111_111;
                x3 <= 6'b111_111;
                x4 <= 'd21;
                x5 <= 'd21;
                x6 <= 'd18;
                x7 <= 'd11;
                if (counter == delay) begin  // 倒计时
                    counter <= 16'b0;
                end
                else begin
                    counter <= counter + 1;
                    duration <= clock - timeStart[flag +: 8];
                    if (button == buttonLeft) begin
                        if(!isVip)begin
                            start_ <= start;
                            if ({key[9:6],key[3:0]} >= bill) begin  // 够钱
                                id[flag +: 8] <= 8'b0;
                                Income <= Income + bill;
                                change <= {key[9:6],key[3:0]} - bill;
                            end
                        end
                        else begin
                            start_ <= vipStart;
```

```
58                              if ({key[9:6],key[3:0]} >= bill - vipBalance[vipFlag +: 8])
begin  // 够钱
59                                  id[flag +: 8] <= 8'b0;
60                                  vipBalance[vipFlag +: 8] <= 8'b0;
61                                  Income <= Income + bill;
62                                  change <= {key[9:6],key[3:0]} - bill;
63                              end
64                          end
65                      end
66                  end
67              end
68            // more states......
69          endcase
70      end
71  end
```

更改变量和输出的always块敏感于clk3上升沿和使能信号下降沿，根据不同的状态和输入更改输出或修改变量值。

键盘输入部分功能描述

1 端口及变量描述



键盘输入部分的**输入变量**有：

- reset信号rst、时钟信号clk、控制输入几位数的3bit信号 cnt_tmp、用于键盘上行扫描的4bit信号row。

键盘部分的**输出变量**有：

- 用于键盘上列扫描的4bit信号col、检测输入结束的寄存器finish、储存输入字符的30bit寄存器result、

其他变量：

- 线网：子模块返回监测到的按下的6bit字符信号key_val，子模块返回的监测到的按下信号key_pressed_flag，子模块返回的分频后的时钟信号key_clk。
- 寄存器：除抖后真正有效的按下信号flag，除抖时使用的计数器20bit的count，存输入位数的2bit的cnt；

该模块中还存在测试是用于显示的变量，由于整体功能实现并没有使用，这里不做描述。

2 功能实现

2.1 子模块例化

```
key_top key(clk,rst,row,col,key_val,key_clk,key_pressed_flag);
```

2.2 功能实现

① 对子模块传递上来的key_pressed_flag信号进行除抖。

```
    always@(posedge key_clk )
    begin
           if(key_pressed_flag)
           count=count+1;

           if(count== 20'd20)
           begin
               flag=1;
               count=20'd0;
               end
               else
               flag=0;
   end
```

② 当检测到有效的按下信号时，先判断输入是否结束，若未结束则在判断按下的值之后对储存结果的寄存器result先赋值后移位，计数减一。同时当reset信号生效时给各个变量赋初值。

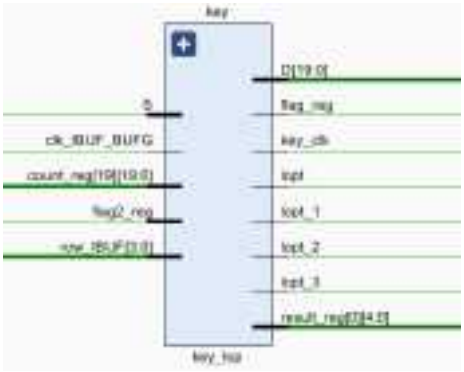
```
always@(posedge key_clk or posedge rst)
    if(rst)
    begin
     cnt=cnt_tmp;
     finish=0;
     result=30'b111111_111111_111111_111111_111111;
     display_rst=2'b11;
    end
    else
      begin
        if(cnt==3'd0)
          begin
          finish=1;
          end
        else
        if(flag)
          begin
            finish=0;
            case(key_val)
             6'h0:begin
                  result[29:24]=6'b000_000;
                  result={ result[23:0], result[29:24] };
                   cnt=cnt-1'b1;
                 end
             6'h1:begin
                  result[29:24]=6'b000_001;
                  result={ result[23:0], result[29:24] };
```

```
                cnt=cnt-1'b1;
                end
            6'h2:begin
                result[29:24]=6'b000_010;
                result={ result[23:0], result[29:24] };
                cnt=cnt-1'b1;
                end
            ......
            endcase
        end
    end
```

3 子模块描述

3.1 key_top 单个键盘输入模块

3.1.1 端口及变量描述



输入包括时钟信号 clk、reset 信号 rst、用于键盘上行扫描的 4bit 信号 row。

输出包括用于键盘上列扫描的4bit信号col、记录按下值的6bit寄存器keyboard_val、分过频的时钟信号key_clk、检测按下键盘的信号key_pressed_flag。

其他变量：

- 寄存器：用于分频计数的20bit的cnt，记录现态和次态的6bit的current_state和next_state，用于锁存行值和列值的4bit的col_val和row_val。
- 参数：用于描述不同状态。

```
NO_KEY_PRESSED = 6'b000_001; // 没有按键按下
SCAN_COL0      = 6'b000_010; // 扫描第0列
SCAN_COL1      = 6'b000_100; // 扫描第1列
SCAN_COL2      = 6'b001_000; // 扫描第2列
SCAN_COL3      = 6'b010_000; // 扫描第3列
KEY_PRESSED    = 6'b100_000; // 有按键按下
```

3.1.2 功能实现

① 对原来时钟信号进行分频，降低为 21ms 监测一次

```
always @ (posedge clk or posedge rst)
  if (rst )
    cnt <= 0;
  else
    cnt <= cnt + 1'b1;

assign key_clk = cnt[19];
```

② 根据分好的频率进行状态切换。

```
always @ (posedge key_clk or posedge rst)
  if (rst)
    current_state <= NO_KEY_PRESSED;
  else
    current_state <= next_state;
```

③ 在状态变化时，根据当前状态以及键盘扫描情况进行状态转移。

```
always @ (*)
  case (current_state)
    NO_KEY_PRESSED :                    // 没有按键按下
        if (row != 4'hF)
          next_state = SCAN_COL0;
        else
          next_state = NO_KEY_PRESSED;
    SCAN_COL0 :                         // 扫描第0列
        if (row != 4'hF)
          next_state = KEY_PRESSED;
        else
          next_state = SCAN_COL1;
    SCAN_COL1 :                         // 扫描第1列
        if (row != 4'hF)
          next_state = KEY_PRESSED;
        else
          next_state = SCAN_COL2;
    SCAN_COL2 :                         // 扫描第2列
        if (row != 4'hF)
          next_state = KEY_PRESSED;
        else
```

```
          next_state = SCAN_COL3;
    SCAN_COL3 :                         // 扫描第3列
        if (row != 4'hF)
          next_state = KEY_PRESSED;
        else
          next_state = NO_KEY_PRESSED;
    KEY_PRESSED :                       // 有按键按下
        if (row != 4'hF)
          next_state = KEY_PRESSED;
        else
          next_state = NO_KEY_PRESSED;
  endcase

```

④ 敏感于分频后的时钟信号，根据次态，给相应寄存器赋值，锁定行值和列值。

```
always @ (posedge key_clk or posedge rst)
  if (rst)
  begin
    col              <= 4'h0;
    key_pressed_flag <=    0;
  end
  else
    case (next_state)
      NO_KEY_PRESSED :                  // 没有按键按下
      begin
        col              <= 4'h0;
        key_pressed_flag <=    0;       // 清键盘按下标志
      end
      SCAN_COL0 :                       // 扫描第0列
        col <= 4'b1110;
      SCAN_COL1 :                       // 扫描第1列
        col <= 4'b1101;
      SCAN_COL2 :                       // 扫描第2列
        col <= 4'b1011;
      SCAN_COL3 :                       // 扫描第3列
        col <= 4'b0111;
      KEY_PRESSED :                     // 有按键按下
      begin
        col_val          <= col;        // 锁存列值
        row_val          <= row;        // 锁存行值
        key_pressed_flag <= 1;          // 置键盘按下标志
      end
endcase
```

⑤ 在分频后的时钟的扫描状态下，根据锁定的行值和列值，确定按下的按键的值。

```
always @ (posedge key_clk or posedge rst)
  if (rst)
    keyboard_val <= 6'b111_111;
  else
    if (key_pressed_flag)
      case ({col_val, row_val})
        8'b1110_1110 : keyboard_val <=6'h1;
        8'b1110_1101 : keyboard_val <=6'h4;
        8'b1110_1011 : keyboard_val <=6'h7;
        8'b1110_0111 : keyboard_val <=6'hE;

        8'b1101_1110 : keyboard_val <=6'h2;
        8'b1101_1101 : keyboard_val <=6'h5;
        8'b1101_1011 : keyboard_val <=6'h8;
        8'b1101_0111 : keyboard_val <=6'h0;

        8'b1011_1110 : keyboard_val <=6'h3;
        8'b1011_1101 : keyboard_val <=6'h6;
        8'b1011_1011 : keyboard_val <=6'h9;
        8'b1011_0111 : keyboard_val <=6'hF;

        8'b0111_1110 : keyboard_val <=6'hA;
        8'b0111_1101 : keyboard_val <=6'hB;
        8'b0111_1011 : keyboard_val <=6'hC;
        8'b0111_0111 : keyboard_val <=6'hD;
        default:   keyboard_val <= 6'b111111;
      endcase
     else  keyboard_val <= 6'b111111;
```

显示部分功能描述

1 端口和变量描述

显示模块顶层的**输入变量**组成为：

- 2bit的rst：用于选择显示的子模块；
- 1bit的clk：即开发板Y18所带的时钟信号；
- 4bit的 left / start / each / id1 / id0 / remain1 / remain0 ： 分别为显示的剩余车辆、起始价、时价、会员帐号的第0位、会员帐号的第1位、余额的第1位、余额的第0位；
- 6bit的 x7 / x6 / x5 / x4 / x3 / x2 / x1 / x0：分别为静态显示时从左到右每一个七段数码显示管应该显示的字符的编码。

显示模块顶层的**输出变量**组成为：

- 8bit的DIG：通过自己的编码控制8个七段数码显示管的亮暗；
- 8bit的Y：通过自己的编码控制七段数码显示管显示什么字符。

显示模块顶层的**其他变量**组成为：

- 1bit的寄存器 rst_tmp1\ rst_tmp2 \ rst_tmp3：分别作为三个子模块的reset信号；
- 8bit的线网 DIG1 \ DIG2 \ DIG3 ： 分别接受三个子模块的DIG输出 ， 同样为控制8个七段数码显示管亮暗的变量；
- 8bit的线网 Y1 \ Y2 \ Y3 ： 分别接受三个子模块的Y输出， 同样为控制七段数码管显示什么字符的变量。

2 功能实现

2.1 例化

```
flowinglight u1(rst_tmp1,clk,DIG1,Y1,left,start,each);
flowinglight_vip u2(rst_tmp2,clk,DIG2,Y2,id1,id0,remain1,remain0);
static_light u3(rst_tmp3,clk,DIG3,Y3,x7,x6,x5,x4,x3,x2,x1,x0);
```

2.2 模块选择

根据顶层rst的不同，将有效的rst_tmp赋为1，其余为0。顶层DIG\Y取有效模块传上来的对应线网。

```
always@*
begin
if(rst == 2'b01)
   begin
     rst_tmp1=1;
     rst_tmp2=0;
     rst_tmp3=0;
     DIG=DIG1;
     Y=Y1;
   end
 else if(rst==2'b10)
    begin
      rst_tmp1=0;
      rst_tmp2=1;
      rst_tmp3=0;
      DIG=DIG2;
      Y=Y2;
    end
 else if(rst==2'b11)
    begin
      rst_tmp1=0;
      rst_tmp2=0;
      rst_tmp3=1;
      DIG=DIG3;
      Y=Y3;
    end
 else
    begin
      rst_tmp1=0;
      rst_tmp2=0;
      rst_tmp3=0;
      DIG=8'b1111_1111;
      Y=8'b1111_1111;                                                    end

end
```

```
35 endmodule
```

3 子模块描述

由于之后的滚动显示是在静态显示的基础上实现的，因此先描述静态显示模块。

3.1 static_light 静态显示模块

3.1.1 端口及变量描述

该模块输入包括reset信号rst，时钟信号clk，控制8个七段数码显示管显示内容的8个6bit的 x0_in、x1_in、x2_in、x3_in、x4_in、x5_in、x6_in,、x7_in。

输出包括顶层提到过的控制七段数码显示管的两个8bit变量：DIG和Y。

其他变量还有：

- 寄存器：分过频的时钟信号clkout，32bit的分频时使用的计数器cnt， 5bit 的scan_cnt标记显示哪个七段数码显示管，7bit的x7, x6, x5, x4, x3, x2, x1记录显示字符的编码，使高电平有效的8bit的DIG_r, 使高电平有效且小数点不显示的8bit的Y_r。
- 参数period，用于分频。

3.1.2 功能实现

① 由于七段数码显示管低电平有效，且我们不需要小数点的显示，为了编码方便。

```
1 assign Y= {1'b1,(~Y_r[6:0])};
2 assign DIG= ~DIG_r;
```

② period=200000，将时钟分频为500Hz，用作之后七段数码显示管交替亮暗的频率（即敏感信号）。

```
1 always @(posedge clk or negedge rst)
2  begin
3    if(!rst) begin
4      cnt<=0;
```

```
    clkout<=0;
   end
   else begin
      if(cnt == (period>>1)-1)
      begin
       clkout<=~clkout;
       cnt<=0;
   end
   else
     cnt<=cnt+1;
  end
 end
```

③ 通过改变DIG_r 让8个七段数码显示管按频率依次亮起，并用scan_cnt记录亮的是哪个七段数码显示管，由于频率较高，达到人眼看起来全亮的状态。

```
always @ (posedge clkout or negedge rst)
begin
 if(!rst)
   scan_cnt<=0;
 else begin
   scan_cnt <= scan_cnt + 1;
   if(scan_cnt == 3'd7) scan_cnt <= 0;
 end
end

always @ (scan_cnt)
begin
  case(scan_cnt)
    3'b000: DIG_r=8'b0000_0001;
    3'b001: DIG_r=8'b0000_0010;
    3'b010: DIG_r=8'b0000_0100;
    3'b011: DIG_r=8'b0000_1000;
    3'b100: DIG_r=8'b0001_0000;
    3'b101: DIG_r=8'b0010_0000;
    3'b110: DIG_r=8'b0100_0000;
    3'b111: DIG_r=8'b1000_0000;
    default: DIG_r = 8'b0000_0000;
    endcase
end
```

④在不同显示管亮起时，通过改变Y_r的编码让七段数码显示管显示不同的值，达到八个七段数码显示管分别显示不同值的效果。

```
always @ (scan_cnt)
begin
 case(scan_cnt)
  0:Y_r= x0;
  1:Y_r= x1;
  2:Y_r= x2;
  3:Y_r= x3;
  4:Y_r= x4;
  5:Y_r= x5;
  6:Y_r= x6;
  7:Y_r= x7;
  default : Y_r= 7'b0000000;
  endcase
 end
endmodule
```

3.2 flowinglight **闲置界面滚动显示** & flowinglight_vip **会员界面滚动显示模块**

3.2.1 *端口及变量描述*

① 在flowinglight中

输入包括reset信号，时钟信号clk，4bit的剩余车辆显示数目、起步价显示数目、时价显示数目。

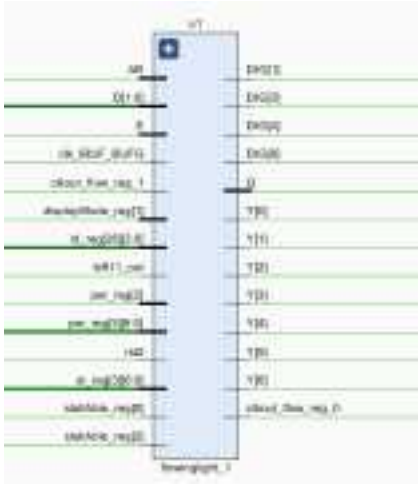
输出包括顶层提到过的控制七段数码显示管的两个8bit变量：DIG和Y。

其他变量：

- 寄存器：分过频的时钟信号clkout，以及分频时用到的计数器32bit的cnt，用于达到静态显示的目的；另外一个分过频的时钟信号clk_out，以及用到的计数器32bit的cnt_flow，使8个七段数码管的显示按这个频率切换，达到滚动显示的目的；
- 5bit 的 scan_cnt 标记显示哪个七段数码显示管；
- 7bit 的 left_flow、start_flow、each_flow为根据剩余车位、起步价、时价的值，对应七段数码显示管显示时的编码；
- 7bit 的 x7、x6、x5、x4、x3、x2、x1、x0 作为变量，记录滚动时每个七段数码显示管的状态迁移。

- 4bit 的 jugde 变量，用来记录现在滚动到什么状态；
- 使高电平有效的 8bit 的 DIG_r；使高电平有效且小数点不显示的 8bit 的 Y_r；
- 参数 period、period_flow，分别用于两个分频；

② 在 flowinglight_vip 中



输入除了显示的参数变为 4bit 的会员帐号第 0 位 id0、会员帐号第 1 位 id1、余额第 0 位 remain0、余额第 1 位 remain1，其余均与滚动显示闲置状态的输入相同，分别为 rst、clk。

输出也为DIG、Y不变。

其他变量：

- 寄存器：除了用于记录显示的数的编码的变量变为id0_flow、id1_flow、remain_0、remain_1，其余均不变，分别为clkout、cnt、clk_flow、cnt_flow、scan_cnt、x7、x6、x5、x4、x3、x2、x1、x0、jugde、DIG_r、Y_r。
- 参数仍然为 period、period_flow 不变。

3.2.2 功能实现

由于两个滚动显示基本相同，核心代码及描述只选择一个模块说明。

在静态显示的基础上，滚动显示还有以下行为：

① period_flow = 100000000，频率为 1Hz，用于控制滚动显示状态改变的频率，是人演可以分辨的状态。

```
always @(posedge clk or negedge rst) begin
   if(!rst) begin
     cnt_flow <= 0;
     clkout_flow <= 0;
   end
   else begin
      if(cnt_flow == (period_flow>>1)-1)
      begin
       clkout_flow <= ~clkout_flow;
```

```
      cnt_flow <= 0;
   end
   else
     cnt_flow <= cnt_flow+1;
  end
end
```

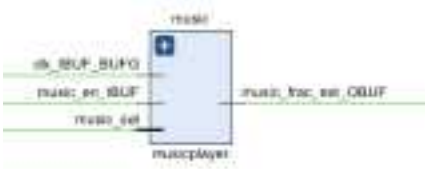
② 敏感与分频后的clk_flow，以这个频率使 x7、x6、x5、x4、x3、x2、x1、x0 相比静态显示时，成为了变量，不同状态的编码不同。

```
always @(posedge clkout_flow or negedge rst)
begin
    if(!rst)
    begin
     x0 <=7'b0000000;
     x1 <=7'b0000000;
     x2 <=7'b0000000;
     x3 <=7'b0000000;
     x4 <=7'b0000000;
     x5 <=7'b0000000;
     x6 <=7'b0000000;
     x7 <=7'b0000000;
     judge<=0;
     end
    else begin
      if(judge==4'd0) begin
                      x0 <=7'b0111000;//L
                      x1 <=7'b0000000;
                      x2 <=7'b0000000;
                      x3 <=7'b0000000;
                      x4 <=7'b0000000;
                      x5 <=7'b0000000;
                      x6 <=7'b0000000;
                      x7 <=7'b0000000;
                      judge  <= judge+1;
                      end
// omitted
else if(judge==4'd15)  begin
                    x0 <=7'b0111000; //L
                    x1 <=7'b0000000;
                    x2 <=7'b0000000;
                    x3 <=7'b0000000;
                    x4 <=7'b0000000;
                    x5 <=7'b0000000;
```

```
35                      x6 <=7'b0000000;
36                      x7 <=7'b0000000;
37                      judge<=0;
38                      end
39      end
40  end
```

音乐播放部分功能描述

1 端口及变量描述



音乐模块仅接受三个**输入**：

- clk 系统时钟信号 100 MHz
- music_en 控制模块整体是否工作
- music_sel 选择音乐曲目，一般接主状态机的状态标识变量即可，由music模块内部自动判断。

音乐部分的**输出变量**为：

- music_frac_ext 接蜂鸣器，其通过在一秒钟内多次高速由 0 - 1 震荡，从而驱动蜂鸣器发出不同频率的声音，模拟播放器支持的 21 个音（低中高八度，各七音）。

本模块在顶层实例化并绑定端口，见前页阐述。

此外，音乐模块还带有以下参数，由标准中央C的频率与板载时钟信号频率计算而得：

```
parameter stop = 0;

parameter do = 19111;

parameter re = 17026;

parameter me = 15168;

parameter fa = 14317;

parameter so = 12755;

parameter la = 11363;

parameter si = 10123;

// lo & hi, 14 params are omitted

parameter pai_gap = 25000000; // 控制当前播放速度，这里设为 0.5 sec 一拍
```

音乐在被近似划分为 21 种音的组合后，可以编码为类似下述结构，并供模块识别读取：

```
parameter SONG_1_LEN = 93;  // 21 个音 + 1 个休止符，每个音需要 5bit 编码
```

> parameter SONG_1_MON = 'b0001……0000;

2 功能实现

① 根据主状态机传入的状态编码，识别当前应该播放的乐曲，并自动选择是否重复播放

```
always @ (posedge clk)  // 切换音乐
case (music_sel)
    default:begin
        song_midi <= 0;
        song_len <= 0;
        music_rep <= 0;
    end
    1:begin
        song_midi <= SONG_1_MON;
        song_len <= SONG_1_LEN;
        music_rep <= 1;
    end
    3:begin
        song_midi <= SUCCESS_PAID;
        song_len <= SUCCESS_PAID_LEN;
        music_rep <= 0;
    end
endcase  // some cases are omitted
```

② 在指定乐曲下，不同拍数的音节不同，因此需要不同频率。

```
// @needs: song_midi, paisp
// 控制每个时间下，各音节需要的震荡的频率
always @ (paisp)begin
    if (!music_en)begin
        freq = stop;
    end
    else begin
        case(song_midi[paisp * 5 +:5])
            'd1 :     freq = do_lo;   // other cases are omitted
            default:  freq = stop;
        endcase
    end
end
```

③ 对于某一指定的音节，知道其对应的震荡频率后，控制输出信号 music_frac_ext 震荡。

```
reg [18:0] freq_cnt;

always @ (posedge clk)begin
    if(music_en)begin
        if (freq_cnt >= freq)begin
            freq_cnt = 0;
            music_frac_ext = ~music_frac_ext;
        end
        else freq_cnt = freq_cnt + 1;
    end
end
```

④ 控制乐曲拍数的前进，这里依靠参数，为 0.5 sec 一拍；music_rep 信号控制乐曲播完后是否从头重播。

```
always @ (posedge clk)begin
    if (music_en)begin
        if(paicg >= pai_gap)begin
            paicg <= 0;
            if (paisp == 0)begin
                if(music_rep)  paisp = song_len;
            end
            else  paisp = paisp - 1;
        end
        else paicg = paicg + 1;
    end
end
```

总结及优化

1 遇到的问题

在开发过程中，我们遇到了大大小小的许多问题，为此熬了不少夜，但好在最后都顺利解决了。我们在一开始设计状态机时没有明确设计思路，所以第一版写了一个两段式状态机。但是因为状态数太多，状态转移条件太多非常难debug，所以后面对状态机进行了重构，改为三段式状态机，将输入输出规范化。但后面在状态切换方面还是有很大的问题，后期经过长时间的 debug，我们发现要将变量赋值的语句都放在状态机的输出部分，才能正常执行。在显示方面，我们还遇到了无法显示的问题，后面检查发现是因为在两个 always 块中分别对同一个变量进行了赋值，导致冲突。

2 系统特色

我们的停车场系统**使用5个按钮**实现了所有的状态转移，使用按钮可以简单的切换状态，降低了操作难度，避免了拨码开关繁琐的操作，且更加符合我们平常的使用习惯。我们还使用了键盘作为输入，并且可以**实时显示输入的内容**，增加了系统的易用性，我们的输入还支持数字以外的输入，给了用户更多的选择空间。我们的显示模块可以根据输入的数字和模式信号切换不同的显示效果，有闲置状态滚动显示，vip滚动显示和静态显示3种模式，这样设计使后期的修改成本大大减小。我们还加入了背景音乐并赋予音乐一个单独的开关，提高了用户体验和趣味性。

3 优化方向

我们的停车场系统还有一些可以优化的地方，第一个就是按钮的灵敏程度，我们按钮的灵敏程度较低，有时会出现按下无反应的情况或者是连续识别多次。初步推测是因为按钮的分频没有设置好，后续可以考虑优化。再就是显示的准确性，受限于七段数码管的长度，我们的显示信息都不是特别完整，后续可以考虑应用七段数码管之外的硬件作为辅助显示。

另外，显示模块滚动显示的地方还可以再进行优化，因为滚动显示中的状态变化使用列举的方法，并没有写算法优化逻辑及语句，代码不灵活，当要使用多个滚动显示时代码量很多。、

键盘输入模块可以优化的地方是，并没有设置退格功能，一旦输入之后无法删除，只能返回后重新进入，这其实是会影响用户体验的。