简易停车场系统的设计与实现

实验报告

姓名:

兰陈昕

班级:

2018211129

学号:

2018210400

指导教师:

崔岩松

目录

摘要

本实验是用硬件描述语言 Verilog 设计实现简易停车场系统，并在 FPGA 开发板上模拟真实的停车场。其基本功能有车辆的驶入驶出、计时计费以及缴费功能，并通过按键、LED 灯、LCD 显示屏、点阵、数码管、蜂鸣器等器件与用户完成交互。总体电路采用模块化设计，将整个系统为九个模块：分频模块、线性反馈移位寄存器模块、按键消抖模块、拨码开关消抖模块、LED 控制模块、LCD 控制模块、数码管显示模块、点阵显示模块以及主模块。本实验使用的设计环境为 Quartus，仿真软件为 ModelSim，FPGA 芯片的型号为 EPM1270T144C5。

关键词：Verilog、停车场系统、FPGA、数字电路

设计任务要求

- 设计实现一个简易停车场模型，模拟车辆的驶入驶出和计费等。
- 基本要求：

1. BTN7 作为系统总开关，系统关闭时，所有显示器件均不亮，系统打开时：
   a) DISP7 和 DISP6 显示 0~23 秒循环计时，模拟一天二十四小时；
   b) 车场共有 8 个车位，LD1、LD3、LD5、LD7、LD9、LD11、LD13、LD15 八个发光管亮起（其余不用的 LD 应一直保持熄灭状态），表示 8 个车位空闲；同时用点阵显示空闲车位数目（8~0）；
   c) 在各个 LD 右下方对应的 SW0~SW7 八个开关用作车辆驶入控制；
2. 将某个 SW 开关置“1”，对应左上方 LD 熄灭，表示车位被占用，点阵显示的车位数相应减 1。依次操作，可以将所有车位占满，在此期间也允许车辆驶出车位；
3. 有车辆准备驶离车位时应将对应的 SW 开关置“0”，对应左上方 LD 以 2Hz 频率闪烁；点阵显示的车位数也以 2Hz 闪烁；DISP4 和 DISP3 两位数码管显示停车时间；DISP1 和 DISP0 两位数码管显示停车费金额：
   a) 停车按时间计费，前 2 秒不计费，以后每秒 2 元；
   b) 停车时长每 24 秒固定收费 40 元，大于 24 秒后按每秒 2 元收费。例如停车 52 秒，即停车时间为（24*2+4）秒，则停车费为 40*2+2*4=88 元；
4. 点击 BTN0 表示缴费，停车时间和金额显示以 2Hz 频率闪烁两次后消失，同时蜂鸣器响起提示音表示缴费成功、对应车位的 LD 亮起、点阵稳定显示实际的空余车位数（原数值+1）。

- 提高要求：
  1. 系统打开时空余停车位的数目和位置随机；
  2. 停车费计算区分白天夜间，6~22 时按白天每秒 2 元，其余按夜间每秒 1 元计费；
  3. 自拟其他功能。
- 模块电路要求：

  在数码管 DISP7 和 DISP6 显示 0~23 秒循环计时，其他数码管均显示 0。

系统设计

设计思路

首先，根据任务要求，可以把系统分为 3 个状态：关机状态（S0）、正常工作状态（S1）、待付款状态（S2）。系统的输入有电源按键（Power）、付款按键（Pay）、拨码开关（Switch）。各种状态的转换如图 3-1 所示。

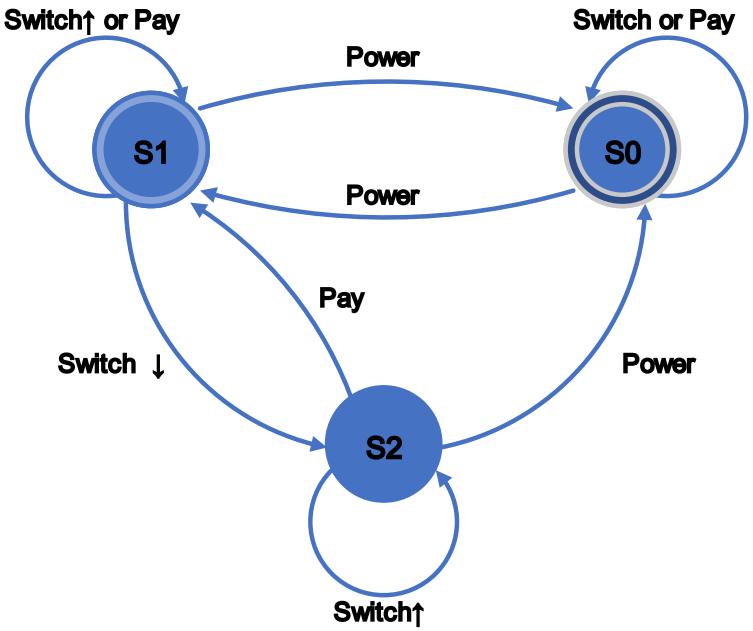

图 3-1 状态图

系统处于关机状态时，LED、LCD、数码管、点阵都不显示。LFSR 模块开始工作，产生伪随机数序列。同时 LCD 进行初始化。

系统进入正常工作状态时，随机数序列生成停止工作，车位是否可用将根据得到的序列决定。同时系统开始计时，并在数码管上显示，同时点阵显示剩余车位数，可用车位对应的 LED 灯亮起，

LCD 屏也将显示时间和车位数，并指示当前是晚上还是晚上。这时如果有拨码开关上拨，那么对应的车位被占用，点阵和 LCD 屏进行信息更新，同时对该车位进行停车时间计时。

系统处于待付款状态时。点阵和驶离车位对应的 LED 灯闪烁。驶离的车位计时停止，并根据之前停靠的时间计算金额，将停车时间和金额显示在数码管上。并在按下付款按键后，数码管对应的数据闪烁两次，系统回到正常状态。

总体框图

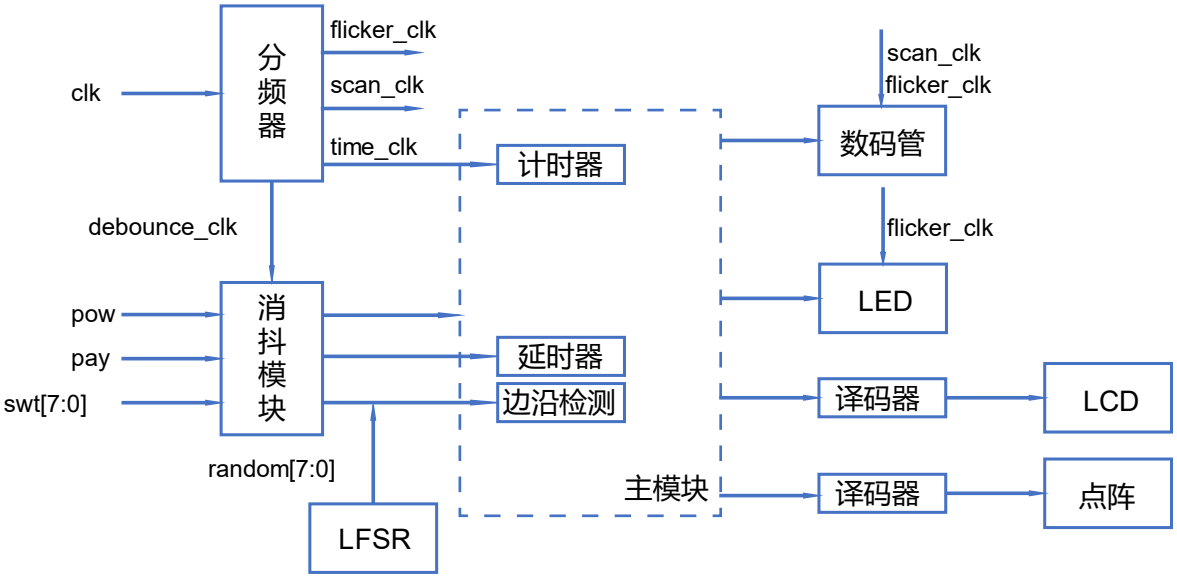

图 3-2 总体结构框图

通过分频器将系统时钟分出几个需要用到的时钟。各个输入信号都先经过消抖模块进行消抖处理后再进入主模块。主模块中进行系统时间以及各个车位的计时，还有检测拨码开关的边缘判断车辆的驶离和计算可用车位的数量，以及在付款后产生一个延时器作用在数码管上。最终主模块中得到的各种数据通过译码器等电路进入 LED、LCD、点阵、数码管进行显示。

分块设计

分频模块

将 FPGA 的始终设定为$50MHz$，经过分析可知，需要的时钟都可以使用偶数分频得到，所以分频模块可以直接使用计时器来实现。对于N倍分频，通过输入时钟触发计数器计数，当计数器从 0 计数到$\frac{N}{2}-1$时，输出时钟进行翻转，以此循环下去。

线性反馈移位寄存器模块

线性反馈移位寄存器（LFSR）是一种特殊的移位寄存器，它的输入位是其先前状态的线性函数。LFSR 产生的序列具有非常长的周期，因此可以看作是伪随机数序列。

LFSR 有多种反馈函数可以选择，考虑到有 8 个车位的输入，所以我们选择 8 位的多项式，它的反馈函数是$x^8+x^6+x^5+x^4+1$[2]，周期为 255，在电路中使用异或操作，种子（初始值）设定为$11111111_{(2)}$。

按键消抖模块

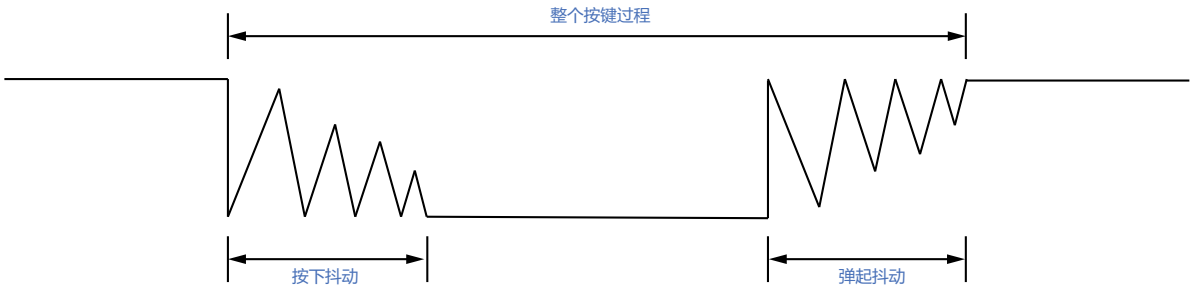

图 3-3 按键抖动示意图

由于机械触点的弹性作用，在按下和释放按钮时，按钮信号在很短的时间内会出现许多意外的上下反弹，被程序读为多次按下，为了防止这种情况的发生就需要按键消抖。

通常来说，发生抖动的时间对比于整个按键周期较短，因此我消抖的策略是：在一个$10Hz$的慢速时钟内检测按键边沿，在边沿产生时生成一个周期为慢速时钟的脉冲，只要慢速时钟的周期大于抖动的时间，就能得到一个稳定的信号，不会出现反弹现象。

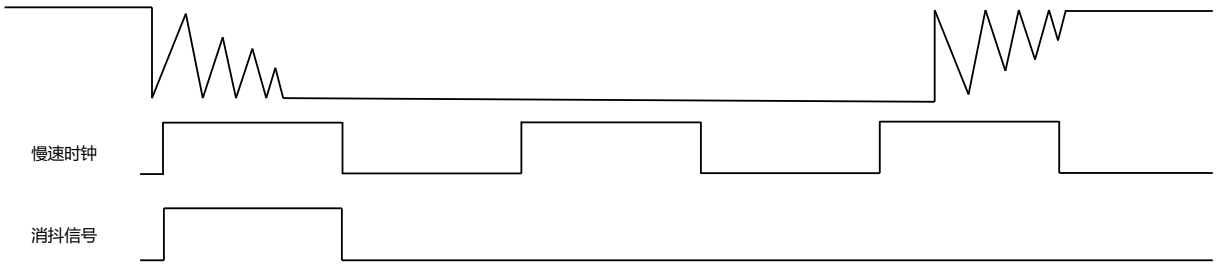

图 3-4 消抖示意图

拨码开关消抖模块

拨码开关的消抖与按键的原理一致，不同的地方在于按键按下后会弹起，而拨码开关不会。因此消抖信号不再是一个脉冲，而是一个稳定不变的信号，直到下一次使用拨码开关时才改变。

LED 控制模块

系统未启动时，所有 LED 灯都熄灭。

系统正常工作时，每个车位对应的 LED 灯状态就与拨码开关的状态一致，上拨时 LED 灯灭，下拨时 LED 灯亮，因此只需要将消抖后的拨码开关信号传递给 LED 灯对应的管脚。

系统处于待付款状态时，当前驶离车位对应的 LED 灯应闪烁。所以将驶离车位的位置和闪烁时钟都传入模块中，把闪烁时钟当作对应 LED 灯的使能信号。

LCD 控制模块

开发板使用的是 LCD1602 模块。它的管脚有 RS(寄存器选择)，R/W（读写选择），E（使能信号），Data bus（数据总线）。模块中有两个 8 位寄存器，一个指令寄存器（IR）和数据寄存器（DR）。IR 存储指令代码。DR 临时存储要写入 DDRAM 或 CGRAM 的数据，并临时存储要从 DDRAM 或 CGRAM 读取的数据。

RS	R/W	操作
0	0	IR 写入内部操作（清屏等）
0	1	读取忙碌标志（DB7）和地址计数器（DB0 至 DB6）
1	0	将数据从 DR 写入 DDRAM 或 CGRAM
1	1	从 DDRAM 或 CGRAM 读取数据到 DR

表 3-1 寄存器选择表[4]

IR 中存储的指令将在表 3-2 中列出。

指令	指令码										描述	执行时间 (f_{osc}=270kHz)
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
清屏	0	0	0	0	0	0	0	0	0	1	清除显示并设置地址计数器为 DDRAM 地址 0。	
光标归位	0	0	0	0	0	0	0	0	1	—	地址计数器设置为 DDRAM 地址 0。将光标返回到初始位置。 DDRAM 内容保持不变。	1.52ms
进入模式设置	0	0	0	0	0	0	0	1	I/D	S	设置光标移动方向并指定显示移位。这些操作在数据写入和读取期间执行。	37μs
显示开关控制	0	0	0	0	0	0	1	D	C	B	设置整个显示的开/关（D），光标的开/关（C）和光标位置字符的闪烁（B）。	37μs
设置显示屏或光标位移	0	0	0	0	0	1	S/C	R/L	—	—	在不更改 DDRAM 的情况下设置光标移动或显示移位控制位，以及移动方向。	37μs
功能设置	0	0	0	0	1	DL	N	F	—	—	设置数据总线长度（DL），显示行数（N）和字符字体（F）。	37μs
设置 CGRAM 地址	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	设置 CGRAM 地址。设置后将发送和接收 CGRAM 数据。	37μs
设置 DDRAM 地址	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	设置 DDRAM 地址。设置后将发送和接收 DDRAM 数据。	37μs
读取忙信号和地址	0	1	BF	AC	AC	AC	AC	AC	AC	AC	读取表示正在执行内部操作的忙标志（BF），并读取地址计数器内容。	0μs
向 DDRAM 或 CGRAM 写数据	1	0	数据								向 DDRAM 或 CGRAM 写数据。	43μs
从 DDRAM 或 CGRAM 读数据	1	1	数据								从 DDRAM 或 CGRAM 读数据。	43μs
	I/D = 1: 光标左移，DDRAM 地址加 1 I/D = 0: 光标右移，DDRAM 地址减 1 S = 1: DDRAM 是读操作（CGRAM 读或写），显示内容不移动 S = 0: DDRAM 是写操作，显示内容移动，移动方向由 I/D 决定 D = 1: 开启显示 D = 0: 关闭显示 C = 1: 开启光标 C = 0: 关闭光标 B = 1: 光标闪烁 B = 0: 光标不闪烁 S/C = 1: 显示内容移动 S/C = 0: 光标左移 R/L = 1: 向右移动 R/L = 0: 向左移动 DL = 1: 8 位 DL = 0: 4 位 N = 1: 显示两行 N = 0: 显示一行 F = 1: 5 × 10 点阵 F = 0: 5 × 8 点阵 BF = 1: 内部运作 BF = 0: 可接受指示										DDRAM: 数据显示 RAM CGRAM: 字符生成器 RAM ACG: CGRAM 地址 ADD: DDRAM 地址(对应于光标地址) AC: DDRAM 和 CGRAM 的地址计数器	

表 3-2 指令表[4]

同时表中还给出了各个指令需要执行的最大时间，这是因为 LCD 控制器一次只能执行一条指令，如果在发送一条指令时上一条还没有执行完成，那么第二条指令将被忽略，所以在每个发送给模块的相邻指令之间都需要加入延时。

CGRAM 是给使用者自行设计显示字符的。但由于我们使用的字符比较常用，已经被存储在 CGROM 中，直接读取即可。对照表如图 3-5.

图 3-5 字符对照表图[4]

因为我们只用预置的字符就可以完成显示，因此 RW 管脚始终为 0。考虑到指令之间需要延迟，因此使用一个500Hz的时钟驱动使能信号。关于指令使用状态机列出了 40 个状态，使用时只用不断循环这些指令即可（图 3-6）。

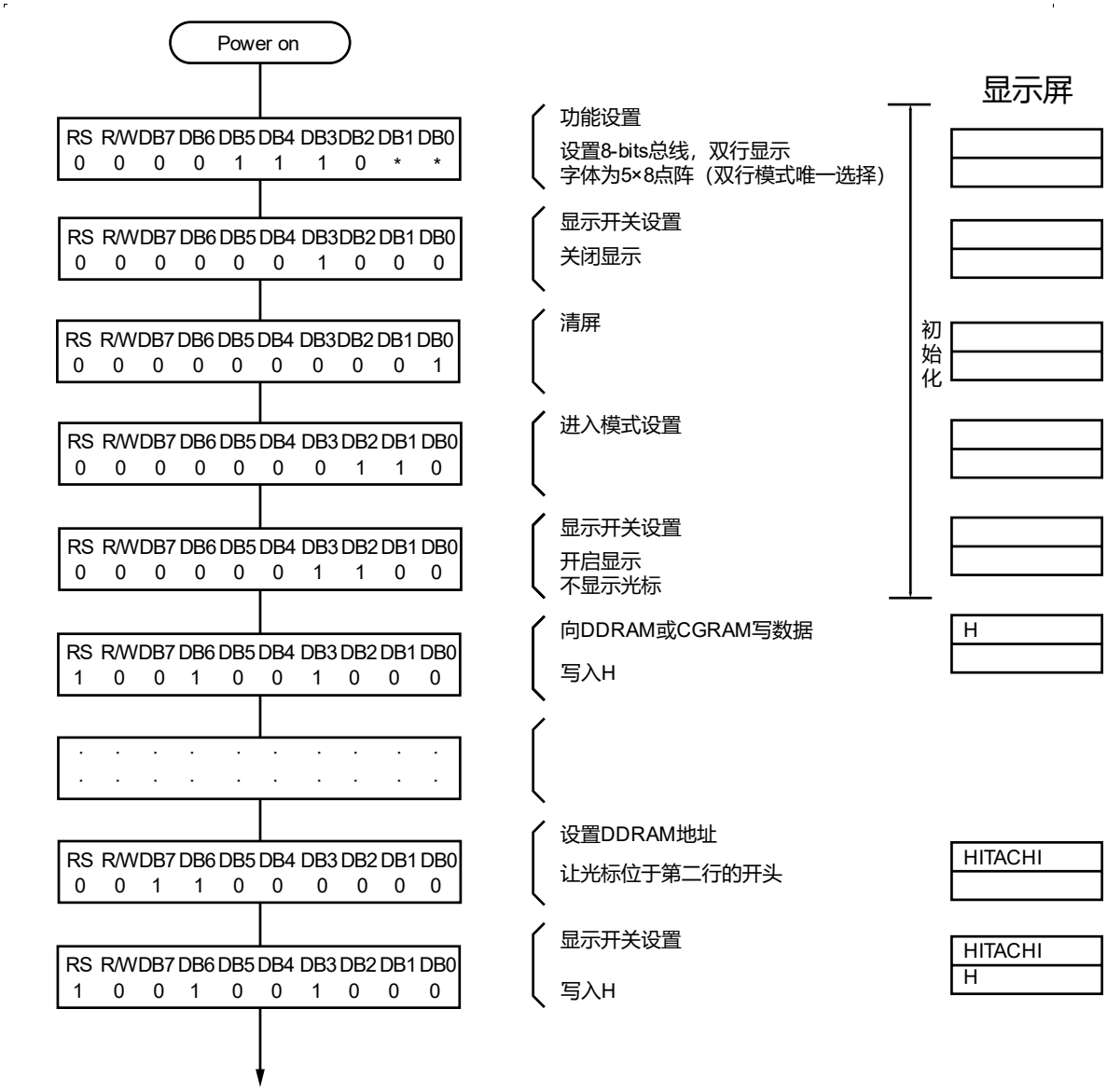

图 3-6 LCD 模块指令顺序图[5]

数码管显示模块

开发板共有 8 个数码管，为了节约管脚，它们的 8 个管脚是并联在一起的（图 3-7）。因此在显示的时候是利用了人眼的视觉暂留特性，用500Hz的时钟扫描数码管逐个显示。利用译码器就可以让数码管显示数字。

在待付款状态，使用一个2Hz的时钟作为使能信号就可以实现闪烁。

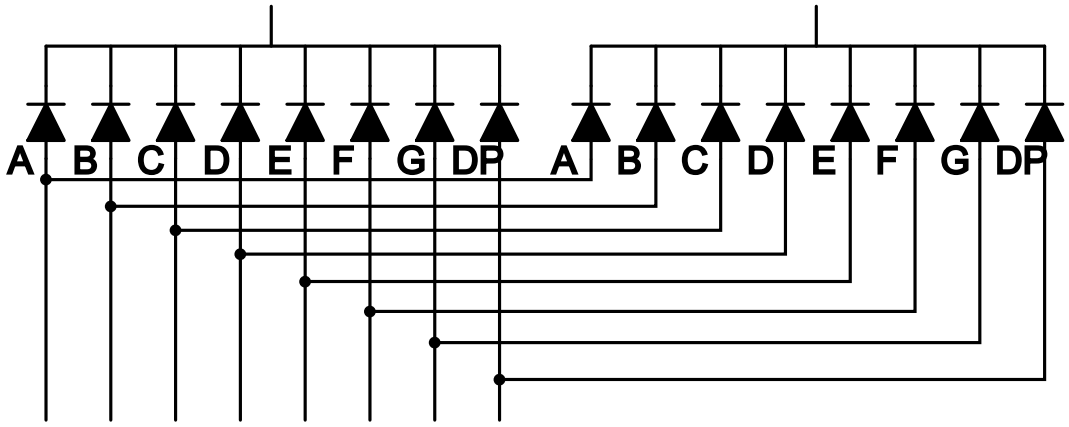

图 3-7 数码管电路示意图

点阵显示模块

点阵也同样用到了扫描显示的方法。点阵的扫描方式主要有三种：按行扫描、按列扫描、按点扫描，我选择了按行扫描。利用 500Hz 的时钟信号对点阵进行逐行扫描，可以得到稳定的点阵显示画面。某一点点亮的条件是 ROW 管脚为 0，COL 管脚为 1.

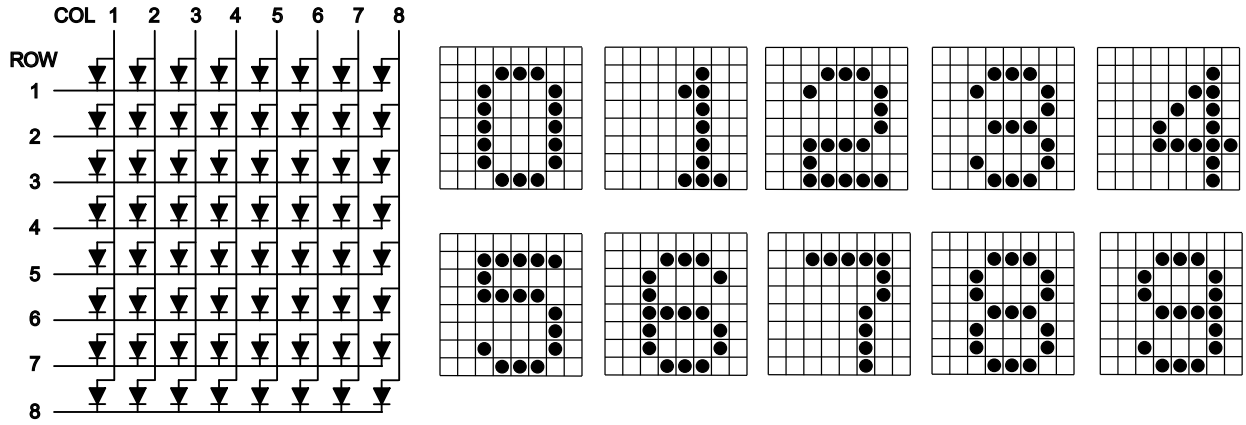

图 3-8 点阵电路图以及数字显示图

主模块

在主模块中我用一个1Hz的时钟进行计时。用一个500Hz的时钟驱动蜂鸣器，使其发出人耳可以听到的声音。还使用寄存器来检测拨码开关的下拨（车辆驶离，进入付款状态）以及付款按键的下降（脱离付款状态）来记录系统的状态。同时使用一个移位寄存器，每遇到2Hz时钟的上升沿时，如果刚刚完成付款则输入位为 1，否则为 0。寄存器总共有两位，这样就实现了一个延时器，使得付款后数码管可以刚好闪烁两次。同时将产生的各种数据传给需要的子模块。

仿真波形及波形分析

所有的仿真波形图都是使用软件 ModelSim 10.5b Intel FPGA Starter Edition 得到。

分频模块

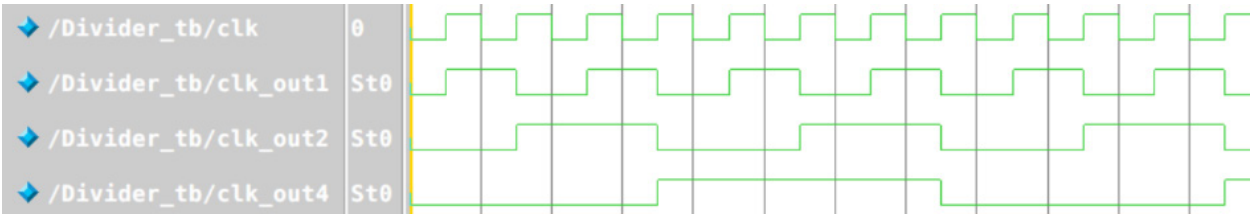

图 4-1 分频模块仿真波形图

分别对系统时钟进行 2、4、8 倍分频，得到的波形符合要求。

线性反馈移位寄存器模块

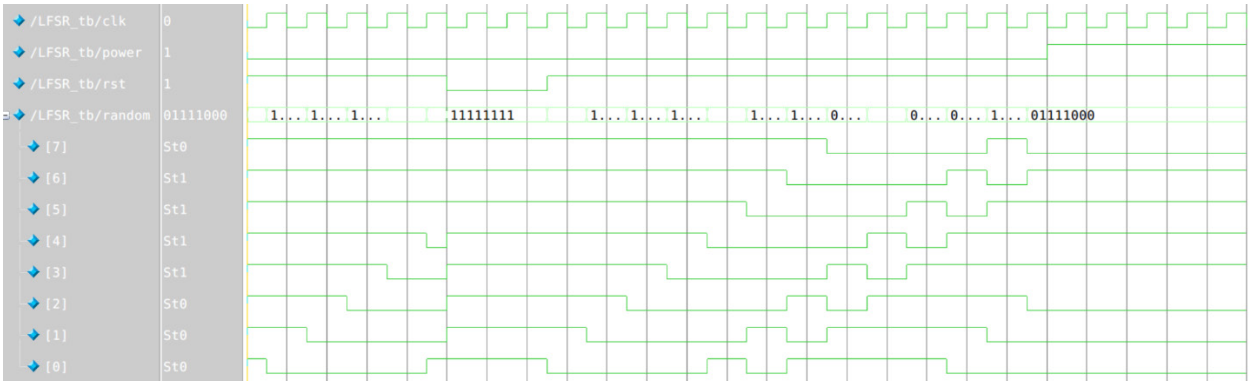

图 4-2 线性反馈移位寄存器模块仿真波形图

随机序列在时钟的驱动下不断生成。遇到 rst 下降沿时（重置）随机序列重置，在 power 置 1（进入工作状态时），序列不再发生变化。

按键消抖模块

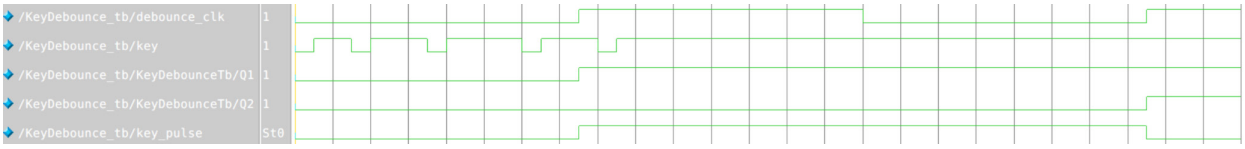

图 4-3 按键消抖模块仿真波形图

在消抖时钟为 0 时，按键发生了多次抖动都不影响输出。消抖时钟变为 1 后检测到按键信号，于是产生了一个消抖时钟周期的脉冲。

拨码开关消抖模块

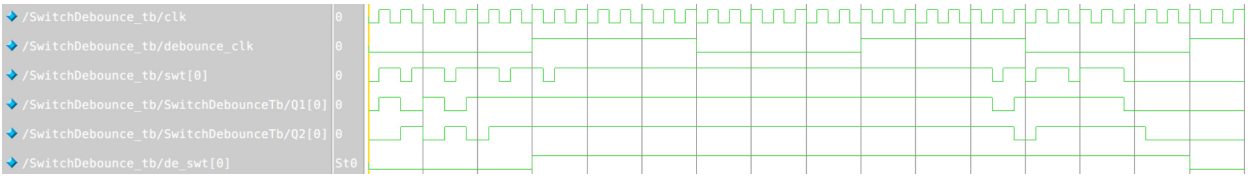

图 4-4 拨码开关消抖模块仿真波形图

只选用一个拨码开关进行仿真。Q1 与拨码开关信号相同，Q2 是 Q1 信号的延迟信号，只有两个信号相同（代表在延时时间内拨码开关稳定）且消抖时钟为 1 时，输出信号才会发生变化。

LED 控制模块

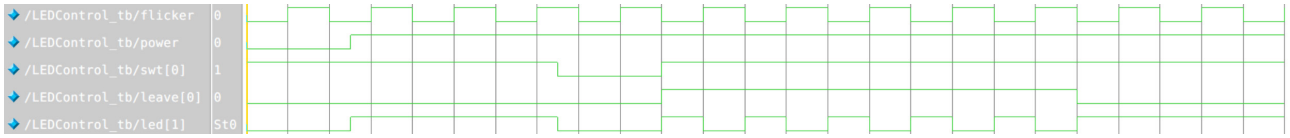

图 4-5 LED 控制模块仿真波形图

只选用一个 LED 灯进行仿真。在 power 为 1（开启）后，LED 灯与开关状态相同。在拨码开关拨下后，leave[0]为 1（待付款状态），此使 LED 的亮暗与 flicker（闪烁使能）相同，在 leave[0]为 0 后（离开待付款状态），LED 灯常亮。

LCD 控制模块

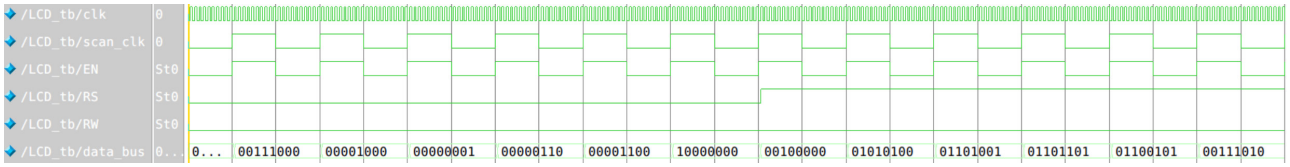

图 4-6 LCD 控制模块仿真波形图

观察 RW、RS 和 data_bus，与我们指定的运行指令一致。使能信号也能够与慢时钟同步。

数码管显示模块

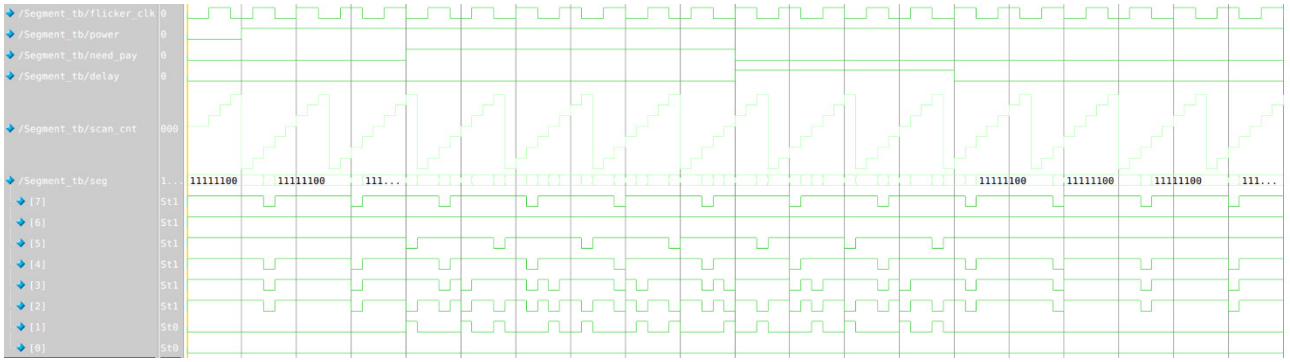

图 4-7 数码管显示模块仿真波形图

在 power 为 1（开启）后，数码管开始工作。这个时候只用显示时间的两个位在工作（仿真中时间不变），所以其他位不变。在 need_pay 为 1（待付款状态）时，显示停车时间和金额的数码管也开始工作。need_pay 变回 0 时 delay 变为 1（延时器），在这个时候可以观察到数码管还受到 flicker_clk（闪烁时钟）的控制。闪烁两次后，数码管又回到正常状态。

点阵显示模块

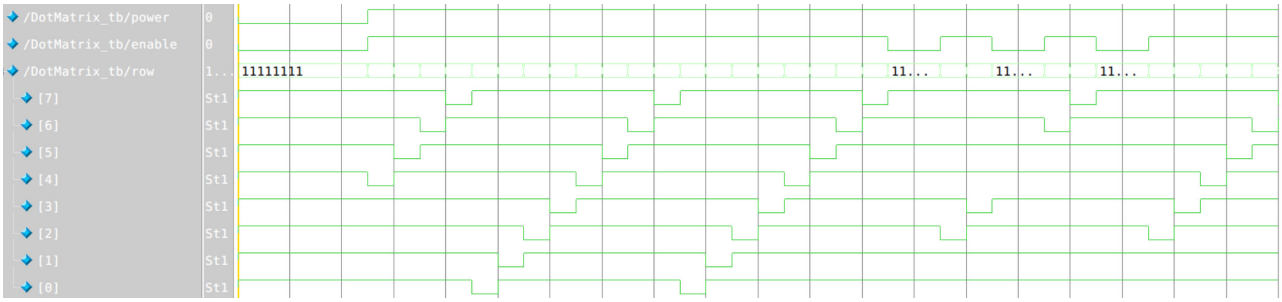

图 4-8 点阵显示模块仿真波形图

在 power 和 enable 为 1 时，row 的变化规律符合按行扫描操作。而在 enable 为 0 时，row 始终置 1，因此能通过控制使能信号 enable 来使点阵闪烁。

主模块

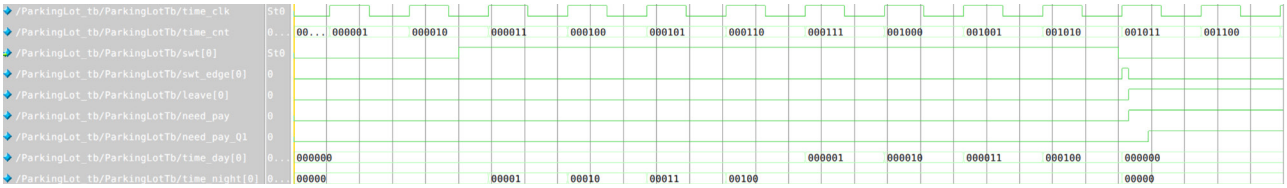

图 4-9 主模块仿真波形图

为了便于仿真，我将调快分频器得到的时钟，同时不进行按键消抖。观察 time_clk 和 time_cnt，得到计时器可以正常工作。在 swt[0]置 1 后（停入车辆），对应车位的计时器开始工作，并且按照白天和晚上分别计时。在 swt[0]置 0 后（车辆驶离），swt_edge 产生了一个脉冲，说明负边沿检测正常；同时 need_pay 变为 1 表明系统进入了待付款状态。按下 pay 按键后，系统又重新回到正常工作状态。

代码

所有的代码都在 Quartus Prime 18.1.1 Lite Edition 中编译通过。

分频模块

```
/**
  * @desc 分频模块 - 使用计数器，在一个计数周期里输出信号发生两次翻转
  * @input wire clk - 系统时钟，f = 50MHz
  * @ouput reg clk_out - 新时钟，f = 50000000 / (kPeriod * 2) Hz
*/
module Divider (
    input clk,
    output reg clk_out = 1'b0
);

```

```
    reg [25:0] cnt = 26'b0;

    parameter kPeriod = 25000000;

    always @(posedge clk) begin
        if (cnt == kPeriod-1) begin
            cnt <= 1'b0;
            clk_out <= ~clk_out;
        end
        else    cnt <= cnt + 1'b1;
    end

endmodule
```

线性反馈移位寄存器模块

```
/**
  * @desc 线性反馈移位寄存器模块 - 生成伪随机数
  * @input wire rst - 系统复位
  * @input wire clk - 系统时钟
  * @input wire power - 系统开关
  * @ouput reg [7:0] random - 伪随机序列
*/
module LFSR (
    input rst,
    input clk,
    input power,
    output reg [7:0] random = 8'b1111_1111
);

    wire feedback = random[7] ^ random[5] ^ random[4] ^ random[3];  // 反馈函数

    always @(posedge clk or negedge rst) begin
        if (!rst)   random <= 8'b1111_1111;     // 种子
        else if (!power)    random <= {random[6:0], feedback};
    end

endmodule
```

按键消抖模块

```
/**
  * @desc 按键消抖模块 - 使用两个D 触发器，检测到按键信号在消抖时钟周期发生翻转才产生脉冲
  * @input wire debounce_clk - 消抖时钟
  * @input wire key - 消抖按键
  * @ouput wire key_pulse - 脉冲
*/
module KeyDebounce (
    input debounce_clk,
    input key,
    output key_pulse
);

    reg Q1 = 1'b0;
    reg Q2 = 1'b0;

    // D 触发器
    always @(posedge debounce_clk) begin
        Q1 <= key;
        Q2 <= Q1;
    end

```

```
    assign key_pulse = Q1 & ~Q2;

endmodule
```

拨码开关消抖模块

```
/**
  * @desc 拨码开关消抖模块 - 与按键消抖模块类似，不过把脉冲信号变为了持续信号
  * @input wire debounce_clk - 消抖时钟
  * @input wire [7:0] swt - 拨码开关
  * @ouput reg [7:0] de_swt - 消抖后的拨码开关状态
*/
module SwitchDebounce (
    input clk,
    input debounce_clk,
    input [7:0] swt,
    output reg [7:0] de_swt = 0
);

    reg [7:0] Q1 = 0;
    reg [7:0] Q2 = 0;

    // D 触发器
    always @(posedge clk) begin
        Q1 <= swt;
        Q2 <= Q1;
        if (de_swt != Q2 && debounce_clk)    de_swt <= Q2;
    end

endmodule
```

LED 控制模块

```
/**
  * @desc LED 控制模块 - 由信号控制LED 的亮暗
  * @input wire power - 系统开关
  * @input wire flicker - 闪烁使能
  * @input wire [7:0] swt - 拨码按键状态
  * @input wire [7:0] leave - 当前驶离车辆
  * @ouput reg [15:0] led - LED 灯
*/
module LEDControl (
    input power,
    input flicker,
    input [7:0] swt,
    input [7:0] leave,
    output reg [15:0] led = 0
);

    integer i;
    always @(*) begin
        for (i = 0; i < 8; i = i+1) begin
            led[i << 1] <= 0;
            // 驶离车位对应的LED 灯亮暗由闪烁使能控制
            if (!power || (leave[i] && flicker))  led[(i << 1)+1] <= 0;
            else    led[(i << 1)+1] <= swt[i];
        end
    end

endmodule
```

LCD 控制模块

```
/**
  * @desc LCD 显示模块 - 控制LCD 显示的内容
  * @input wire rst - 系统复位
  * @input wire clk - 系统时钟，f = 50MHz
  * @input wire scan_clk - 慢速时钟，f = 500Hz
  * @input wire power - 系统开关
  * @input wire [3:0] car - 可用车位数
  * @input wire [5:0] time_cnt - 系统时间
  * @ouput wire RW - 读写选择
  * @ouput wire EN - 使能
  * @ouput reg RS - 寄存器选择
  * @ouput reg [7 : 0] data_bus - 数据总线
*/
module LCD (
    input rst,
    input clk,
    input scan_clk,
    input power,
    input [3:0] car,
    input [5:0] time_cnt,
    output RW,
    output EN,
    output reg RS = 0,
    output reg [7:0] data_bus = 0
);

    // 译码部分
    // 根据可用车位数和系统时间更改显示的内容
    reg [127:0] row_1 = "                ";
    reg [127:0] row_2 = "                ";

    always @(posedge clk or negedge power) begin
        if (!power) row_1 = "                ";
        else case (time_cnt)
            0 :     row_1 <= " Time: 00 Night ";
            1 :     row_1 <= " Time: 01 Night ";
            2 :     row_1 <= " Time: 02 Night ";
            3 :     row_1 <= " Time: 03 Night ";
            4 :     row_1 <= " Time: 04 Night ";
            5 :     row_1 <= " Time: 05 Night ";
            6 :     row_1 <= " Time: 06 Day   ";
            7 :     row_1 <= " Time: 07 Day   ";
            8 :     row_1 <= " Time: 08 Day   ";
            9 :     row_1 <= " Time: 09 Day   ";
            10 :    row_1 <= " Time: 10 Day   ";
            11 :    row_1 <= " Time: 11 Day   ";
            12 :    row_1 <= " Time: 12 Day   ";
            13 :    row_1 <= " Time: 13 Day   ";
            14 :    row_1 <= " Time: 14 Day   ";
            15 :    row_1 <= " Time: 15 Day   ";
            16 :    row_1 <= " Time: 16 Day   ";
            17 :    row_1 <= " Time: 17 Day   ";
            18 :    row_1 <= " Time: 18 Day   ";
            19 :    row_1 <= " Time: 19 Day   ";
            20 :    row_1 <= " Time: 20 Day   ";
            21 :    row_1 <= " Time: 21 Day   ";
            22 :    row_1 <= " Time: 22 Day   ";
            23 :    row_1 <= " Time: 23 Night ";
        endcase
```

```
60      end
61
62      always @(posedge clk or negedge power) begin
63          if (!power) row_2 <= "                ";
64          else case (car)
65              0 :     row_2 <= " Space: 0       ";
66              1 :     row_2 <= " Space: 1       ";
67              2 :     row_2 <= " Space: 2       ";
68              3 :     row_2 <= " Space: 3       ";
69              4 :     row_2 <= " Space: 4       ";
70              5 :     row_2 <= " Space: 5       ";
71              6 :     row_2 <= " Space: 6       ";
72              7 :     row_2 <= " Space: 7       ";
73              8 :     row_2 <= " Space: 8       ";
74          endcase
75      end
76
77      reg Q = 0;
78      always @(posedge clk) begin
79          Q <= scan_clk;
80      end
81
82      assign RW = 1'b0;   // 始终只进行写入
83      assign EN = scan_clk;
84      wire write_flag = ~Q & scan_clk;
85
86      // 40 个状态
87      localparam        IDLE = 8'h00;
88      // 初始化
89      localparam    DISP_SET = 8'h01;    // 清屏
90      localparam    DISP_OFF = 8'h03;    // 关闭显示
91      localparam     CLR_SCR = 8'h02;    // 光标归位
92      localparam CURSOR_SET1 = 8'h06;    // 设置光标
93      localparam CURSOR_SET2 = 8'h07;    // 开启显示
94      // 显示第一行
95      localparam   ROW1_ADDR = 8'h05;
96      localparam      ROW1_0 = 8'h04;
97      localparam      ROW1_1 = 8'h0C;
98      localparam      ROW1_2 = 8'h0D;
99      localparam      ROW1_3 = 8'h0F;
100     localparam      ROW1_4 = 8'h0E;
101     localparam      ROW1_5 = 8'h0A;
102     localparam      ROW1_6 = 8'h0B;
103     localparam      ROW1_7 = 8'h09;
104     localparam      ROW1_8 = 8'h08;
105     localparam      ROW1_9 = 8'h18;
106     localparam      ROW1_A = 8'h19;
107     localparam      ROW1_B = 8'h1B;
108     localparam      ROW1_C = 8'h1A;
109     localparam      ROW1_D = 8'h1E;
110     localparam      ROW1_E = 8'h1F;
111     localparam      ROW1_F = 8'h1D;
112     // 显示第二行
113     localparam   ROW2_ADDR = 8'h1C;
114     localparam      ROW2_0 = 8'h14;
115     localparam      ROW2_1 = 8'h15;
116     localparam      ROW2_2 = 8'h17;
117     localparam      ROW2_3 = 8'h16;
118     localparam      ROW2_4 = 8'h12;
119     localparam      ROW2_5 = 8'h13;
120     localparam      ROW2_6 = 8'h11;
121     localparam      ROW2_7 = 8'h10;
```

```
122     localparam      ROW2_8 = 8'h30;
123     localparam      ROW2_9 = 8'h31;
124     localparam      ROW2_A = 8'h33;
125     localparam      ROW2_B = 8'h32;
126     localparam      ROW2_C = 8'h36;
127     localparam      ROW2_D = 8'h37;
128     localparam      ROW2_E = 8'h35;
129     localparam      ROW2_F = 8'h34;
130
131     reg [7:0] current_state = IDLE;
132     reg [7:0] next_state = IDLE;
133     always @(posedge clk or negedge rst) begin
134         if (!rst)  current_state <= IDLE;
135         else if (write_flag)  current_state <= next_state;
136     end
137
138     always @(*) begin
139         case (current_state)
140             // 初始化
141             IDLE :         next_state = DISP_SET;
142             DISP_SET :     next_state = DISP_OFF;
143             DISP_OFF :     next_state = CLR_SCR;
144             CLR_SCR :      next_state = CURSOR_SET1;
145             CURSOR_SET1 :  next_state = CURSOR_SET2;
146             CURSOR_SET2 :  next_state = ROW1_ADDR;
147             // 显示第一行
148             ROW1_ADDR :    next_state = ROW1_0;
149             ROW1_0 :       next_state = ROW1_1;
150             ROW1_1 :       next_state = ROW1_2;
151             ROW1_2 :       next_state = ROW1_3;
152             ROW1_3 :       next_state = ROW1_4;
153             ROW1_4 :       next_state = ROW1_5;
154             ROW1_5 :       next_state = ROW1_6;
155             ROW1_6 :       next_state = ROW1_7;
156             ROW1_7 :       next_state = ROW1_8;
157             ROW1_8 :       next_state = ROW1_9;
158             ROW1_9 :       next_state = ROW1_A;
159             ROW1_A :       next_state = ROW1_B;
160             ROW1_B :       next_state = ROW1_C;
161             ROW1_C :       next_state = ROW1_D;
162             ROW1_D :       next_state = ROW1_E;
163             ROW1_E :       next_state = ROW1_F;
164             ROW1_F :       next_state = ROW2_ADDR;
165             // 显示第二行
166             ROW2_ADDR :    next_state = ROW2_0;
167             ROW2_0 :       next_state = ROW2_1;
168             ROW2_1 :       next_state = ROW2_2;
169             ROW2_2 :       next_state = ROW2_3;
170             ROW2_3 :       next_state = ROW2_4;
171             ROW2_4 :       next_state = ROW2_5;
172             ROW2_5 :       next_state = ROW2_6;
173             ROW2_6 :       next_state = ROW2_7;
174             ROW2_7 :       next_state = ROW2_8;
175             ROW2_8 :       next_state = ROW2_9;
176             ROW2_9 :       next_state = ROW2_A;
177             ROW2_A :       next_state = ROW2_B;
178             ROW2_B :       next_state = ROW2_C;
179             ROW2_C :       next_state = ROW2_D;
180             ROW2_D :       next_state = ROW2_E;
181             ROW2_E :       next_state = ROW2_F;
182             ROW2_F :       next_state = ROW1_ADDR;
183             default :      next_state = IDLE;
184         endcase
```

```
185     end
186
187     always @(posedge clk or negedge rst) begin
188         if (!rst) RS <= 1'b0;
189         else if (write_flag) begin
190             if (next_state == IDLE        || next_state == DISP_SET    ||
191                 next_state == DISP_OFF    || next_state == CLR_SCR     ||
192                 next_state == CURSOR_SET1 || next_state == CURSOR_SET2 ||
193                 next_state == ROW1_ADDR   || next_state == ROW2_ADDR)
194                     RS <= 1'b0;     //写入指令
195             else    RS <= 1'b1;     //写入数据
196         end
197     end
198
199     always @(posedge clk or negedge rst) begin
200         if (!rst)             data_bus <= 8'hxx;
201         else if (write_flag)
202             case (next_state)
203                 IDLE :        data_bus <= 8'hxx;
204                 // 初始化
205                 DISP_SET :    data_bus <= 8'h38;
206                 DISP_OFF :    data_bus <= 8'h08;
207                 CLR_SCR :     data_bus <= 8'h01;
208                 CURSOR_SET1 : data_bus <= 8'h06;
209                 CURSOR_SET2 : data_bus <= 8'h0c;
210                 // 显示第一行
211                 ROW1_ADDR :   data_bus <= 8'h80;
212                 ROW1_0 :      data_bus <= row_1[127:120];
213                 ROW1_1 :      data_bus <= row_1[119:112];
214                 ROW1_2 :      data_bus <= row_1[111:104];
215                 ROW1_3 :      data_bus <= row_1[103:96 ];
216                 ROW1_4 :      data_bus <= row_1[95 :88 ];
217                 ROW1_5 :      data_bus <= row_1[87 :80 ];
218                 ROW1_6 :      data_bus <= row_1[79 :72 ];
219                 ROW1_7 :      data_bus <= row_1[71 :64 ];
220                 ROW1_8 :      data_bus <= row_1[63 :56 ];
221                 ROW1_9 :      data_bus <= row_1[55 :48 ];
222                 ROW1_A :      data_bus <= row_1[47 :40 ];
223                 ROW1_B :      data_bus <= row_1[39 :32 ];
224                 ROW1_C :      data_bus <= row_1[31 :24 ];
225                 ROW1_D :      data_bus <= row_1[23 :16 ];
226                 ROW1_E :      data_bus <= row_1[15 :8  ];
227                 ROW1_F :      data_bus <= row_1[7  :0  ];
228                 // 显示第二行
229                 ROW2_ADDR :   data_bus <= 8'hC0;
230                 ROW2_0 :      data_bus <= row_2[127:120];
231                 ROW2_1 :      data_bus <= row_2[119:112];
232                 ROW2_2 :      data_bus <= row_2[111:104];
233                 ROW2_3 :      data_bus <= row_2[103:96 ];
234                 ROW2_4 :      data_bus <= row_2[95 :88 ];
235                 ROW2_5 :      data_bus <= row_2[87 :80 ];
236                 ROW2_6 :      data_bus <= row_2[79 :72 ];
237                 ROW2_7 :      data_bus <= row_2[71 :64 ];
238                 ROW2_8 :      data_bus <= row_2[63 :56 ];
239                 ROW2_9 :      data_bus <= row_2[55 :48 ];
240                 ROW2_A :      data_bus <= row_2[47 :40 ];
241                 ROW2_B :      data_bus <= row_2[39 :32 ];
242                 ROW2_C :      data_bus <= row_2[31 :24 ];
243                 ROW2_D :      data_bus <= row_2[23 :16 ];
244                 ROW2_E :      data_bus <= row_2[15 :8  ];
245                 ROW2_F :      data_bus <= row_2[7  :0  ];
246             endcase
```

```
    end

endmodule
```

数码管显示模块

```
/**
  * @desc 数码管显示模块
  * @input wire power - 系统开关
  * @input wire delay - 延时状态
  * @input wire need_pay - 缴费状态
  * @input wire flicker_clk - 闪烁时钟
  * @input wire [5:0] count - 点阵显示的数字
  * @input wire [2:0] scan_cnt - 扫描计数
  * @input wire [5:0] time_day - 白天停车时间
  * @input wire [4:0] time_night - 夜晚停车时间
  * @ouput reg [7:0] dis - 数码管位置
  * @ouput reg [7:0] seg - 数码管管脚
*/
module Segment (
    input power,
    input delay,
    input need_pay,
    input flicker_clk,
    input [5:0] count,
    input [2:0] scan_cnt,
    input [5:0] time_day,
    input [4:0] time_night,
    output reg [7:0] dis = 0,
    output reg [7:0] seg = 0
);

    always @(*) begin
        if (power) begin
            if (delay && flicker_clk) begin     // 当系统处于延时状态时，闪烁
                case (scan_cnt)
                    6 :       dis <= ~8'b0100_0000;
                    7 :       dis <= ~8'b1000_0000;
                    default : dis <= ~8'b0000_0000;
                endcase
            end
            else begin
                case (scan_cnt)
                    0 :       dis <= ~8'b0000_0001;
                    1 :       dis <= ~8'b0000_0010;
                    3 :       dis <= ~8'b0000_1000;
                    4 :       dis <= ~8'b0001_0000;
                    6 :       dis <= ~8'b0100_0000;
                    7 :       dis <= ~8'b1000_0000;
                    default : dis <= ~8'b0000_0000;
                endcase
            end
        end
        else                  dis <= ~8'b0000_0000;
    end

    reg [7:0] kSeg [9:0];
    initial begin
        kSeg[0] = 8'b1111_1100;
        kSeg[1] = 8'b0110_0000;
        kSeg[2] = 8'b1101_1010;
        kSeg[3] = 8'b1111_0010;
```

```
        kSeg[4] = 8'b0110_0110;
        kSeg[5] = 8'b1011_0110;
        kSeg[6] = 8'b1011_1110;
        kSeg[7] = 8'b1110_0000;
        kSeg[8] = 8'b1111_1110;
        kSeg[9] = 8'b1111_0110;
    end

    // 计算总时间和总金额
    wire [5:0] whole_time = time_day + time_night;
    wire [6:0] whole_money = time_day * 7'd2 + time_night;

    always @(*) begin
        // 当系统处于待缴费状态时以及延时状态时，34 显示时间，01 显示金额
        if (delay || need_pay) begin
            case (scan_cnt)
                7 :       seg <= kSeg[count / 10];
                6 :       seg <= kSeg[count % 10];
                4 :       seg <= kSeg[whole_time / 10];
                3 :       seg <= kSeg[whole_time % 10];
                1 :       seg <= kSeg[whole_money / 10];
                0 :       seg <= kSeg[whole_money % 10];
                default : seg <= kSeg[0];
            endcase
        end
        else begin
            case (scan_cnt)
                7 :       seg <= kSeg[count / 10];
                6 :       seg <= kSeg[count % 10];
                default : seg <= kSeg[0];
            endcase
        end
    end

endmodule
```

点阵显示模块

```
/**
  * @desc 点阵显示模块 - 点阵在信号的控制下显示数字
  * @input wire power - 系统开关
  * @input wire enable - 使能
  * @input wire [3:0] num - 点阵显示的数字
  * @input wire [2:0] scan_cnt - 扫描计数
  * @ouput reg [7 : 0] row - 点阵行
  * @ouput reg [7 : 0] col_r - 点阵列（红色）
  * @ouput reg [7 : 0] col_g - 点阵列（绿色）
*/
module DotMatrix (
    input power,
    input enable,
    input [3:0] num,
    input [2:0] scan_cnt,
    output reg [7 : 0] row,
    output reg [7 : 0] col_r,
    output reg [7 : 0] col_g
);

    // 扫描行
    always @(*) begin
        if (enable && power)       // 只有当系统开启和使能为1时显示
            case (scan_cnt)
```

```
                0 :         row <= 8'b1111_1110;
                1 :         row <= 8'b1111_1101;
                2 :         row <= 8'b1111_1011;
                3 :         row <= 8'b1111_0111;
                4 :         row <= 8'b1110_1111;
                5 :         row <= 8'b1101_1111;
                6 :         row <= 8'b1011_1111;
                7 :         row <= 8'b0111_1111;
                default :   row <= 8'b1111_1111;
            endcase
        else                row <= 8'b1111_1111;
    end

    // 根据要显示的数字扫描列
    always @(*) begin
        if (enable && power) begin // 只有当系统开启和使能为1时显示
        case (num)
            0 : case (scan_cnt)
                1 :         begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                2 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                3 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                4 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                5 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                6 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                7 :         begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                default :   begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            1 :   case (scan_cnt)
                1 :         begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                2 :         begin col_r <= 8'b0000_1100; col_g <= 8'b0000_1100; end
                3 :         begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                4 :         begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                5 :         begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                6 :         begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                7 :         begin col_r <= 8'b0000_1110; col_g <= 8'b0000_1110; end
                default :   begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            2 : case (scan_cnt)
                1 :         begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                2 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                3 :         begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                4 :         begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                5 :         begin col_r <= 8'b0011_1100; col_g <= 8'b0011_1100; end
                6 :         begin col_r <= 8'b0010_0000; col_g <= 8'b0010_0000; end
                7 :         begin col_r <= 8'b0011_1110; col_g <= 8'b0011_1110; end
                default :   begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            3 : case (scan_cnt)
                1 :         begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                2 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                3 :         begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                4 :         begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                5 :         begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                6 :         begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                7 :         begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                default :   begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            4 : case (scan_cnt)
                1 :         begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                2 :         begin col_r <= 8'b0000_1100; col_g <= 8'b0000_1100; end
                3 :         begin col_r <= 8'b0001_0100; col_g <= 8'b0001_0100; end
                4 :         begin col_r <= 8'b0010_0100; col_g <= 8'b0010_0100; end
                5 :         begin col_r <= 8'b0011_1110; col_g <= 8'b0011_1110; end
```

```
                6 :       begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                7 :       begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                default : begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            5 : case (scan_cnt)
                1 :       begin col_r <= 8'b0011_1110; col_g <= 8'b0011_1110; end
                2 :       begin col_r <= 8'b0010_0000; col_g <= 8'b0010_0000; end
                3 :       begin col_r <= 8'b0011_1100; col_g <= 8'b0011_1100; end
                4 :       begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                5 :       begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                6 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                7 :       begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                default : begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            6 : case (scan_cnt)
                1 :       begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                2 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                3 :       begin col_r <= 8'b0010_0000; col_g <= 8'b0010_0000; end
                4 :       begin col_r <= 8'b0011_1100; col_g <= 8'b0011_1100; end
                5 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                6 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                7 :       begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                default : begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            7 : case (scan_cnt)
                1 :       begin col_r <= 8'b0011_1110; col_g <= 8'b0011_1110; end
                2 :       begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                3 :       begin col_r <= 8'b0000_0010; col_g <= 8'b0000_0010; end
                4 :       begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                5 :       begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                6 :       begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                7 :       begin col_r <= 8'b0000_0100; col_g <= 8'b0000_0100; end
                default : begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            8 : case (scan_cnt)
                1 :       begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                2 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                3 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                4 :       begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                5 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                6 :       begin col_r <= 8'b0010_0010; col_g <= 8'b0010_0010; end
                7 :       begin col_r <= 8'b0001_1100; col_g <= 8'b0001_1100; end
                default : begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
            endcase
            default :     begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
        endcase
        end
        else              begin col_r <= 8'b0000_0000; col_g <= 8'b0000_0000; end
    end

endmodule
```

主模块

```
/**
  * @desc 主模块
  * @input wire rst - 系统复位
  * @input wire clk - 系统时钟，f = 50MHz
  * @input wire pow - 开关按键
  * @input wire pay - 缴费按键
  * @input wire [7:0] swt - 拨码开关
  * @ouput wire rs - lcd 寄存器选择
```

```
 * @ouput wire rw - lcd 读写选择
 * @ouput wire en - lcd 使能
 * @ouput wire buz - 蜂鸣器
 * @ouput wire [7:0] row - 点阵行
 * @ouput wire [7:0] col_r - 点阵列（红色）
 * @ouput wire [7:0] col_g - 点阵列（绿色）
 * @ouput wire [7:0] dis - 数码管位置
 * @ouput wire [7:0] seg - 数码管管脚
 * @ouput wire [7:0] lcd - lcd 数据总线
 * @ouput wire [15:0] led - LED 灯
*/
module ParkingLot (
    input rst,
    input clk,
    input pow,
    input pay,
    input [7:0] swt,
    output rs,
    output rw,
    output en,
    output buz,
    output [7:0] row,
    output [7:0] col_r,
    output [7:0] col_g,
    output [7:0] dis,
    output [7:0] seg,
    output [7:0] lcd,
    output [15:0] led
);

    // 仿真用
    /*wire pow_pulse = pow;
    wire pay_pulse = pay;
    wire [7:0] de_swt = swt;
    wire [7:0] de_swt_bar = ~swt;*/

    /*************************    系统初始化部分    *************************/

    // 系统启动部分 按下开关后系统启动，按下复位按键系统关闭复位
    reg power = 0;      // power 为1 代表系统开启，为0 代表系统关闭
    always @(posedge pow_pulse or negedge rst) begin
        if (!rst)   power <= 0;
        else    power <= pow_pulse ? ~power : power;
    end

    // 生成伪随机数
    wire [7:0] random;
    LFSR EightBitRandom(rst, clk, power, random);

    /*************************    系统初始化部分    *************************/

    /*************************    分频部分    *************************/

    wire scan_clk;          // 扫描时钟
    wire debounce_clk;      // 消抖时钟
    wire flicker_clk;       // 闪烁时钟
    wire time_clk;          // 计时时钟
    // f = 50000000 / 100000 = 500Hz
    Divider#(.kPeriod(50000)) ScanDivider(clk, scan_clk);
    // f = 50000000 / 50000000 = 10Hz
```

```
    Divider#(.kPeriod(2500000)) DebounceDivider(clk, debounce_clk);
    // f = 50000000 / 25000000 = 2Hz
    Divider#(.kPeriod(12500000)) FlickerDivider(clk, flicker_clk);
    //Divider#(.kPeriod(2)) FlickerDividerTest(clk, flicker_clk);
    // f = 50000000 / 50000000 = 1Hz
    Divider#(.kPeriod(25000000)) TimeDivider(clk, time_clk);
    // Divider#(.kPeriod(6)) TimeDividerTest(clk, time_clk);

    /*****************************    分频部分    *****************************/

    /*****************************    消抖部分    *****************************/

    wire pow_pulse;         // 开关按键脉冲
    wire pay_pulse;         // 缴费按键脉冲
    wire [7:0] de_swt;      // 拨码开关状态
    KeyDebounce PowDebounce(debounce_clk, pow, pow_pulse);
    KeyDebounce PayDebounce(debounce_clk, pay, pay_pulse);
    SwitchDebounce SwtDebounce(clk, debounce_clk, swt, de_swt);

    // 拨码开关状态取反同时与 random 相与
    // random 为0 的位置对应的车位将无效，只有为1 的位置才可以正常操作
    wire [7:0] de_swt_bar = ~de_swt & random;

    /*****************************    消抖部分    *****************************/

    /*****************************    计时部分    *****************************/

    reg [5:0] time_cnt = 0;     // 系统时间
    always @(posedge time_clk or negedge power) begin
        if (!power)  time_cnt <= 0;
        else    time_cnt <= (time_cnt > 22) ? 1'b0 : time_cnt + 1'b1;
    end

    // time_day 代表白天停车的时间，time_night 代表夜晚停车时间
    // 0~7 对应8 个车位，8 储存当前驶离车辆的停车时间
    reg [5:0] time_day [8:0];
    reg [4:0] time_night [8:0];

    initial begin
        time_day[0] = 0;    time_night[0] = 0;
        time_day[1] = 0;    time_night[1] = 0;
        time_day[2] = 0;    time_night[2] = 0;
        time_day[3] = 0;    time_night[3] = 0;
        time_day[4] = 0;    time_night[4] = 0;
        time_day[5] = 0;    time_night[5] = 0;
        time_day[6] = 0;    time_night[6] = 0;
        time_day[7] = 0;    time_night[7] = 0;
        time_day[8] = 0;    time_night[8] = 0;
    end

    integer i;
    always @(posedge time_clk or negedge power) begin
        for (i = 0; i < 8; i = i+1) begin
            if (!power) begin time_day[i] <= 0; time_night[i] <= 0; end
            else begin
                if (de_swt[i]) begin
                    if (time_cnt >= 6 && time_cnt <= 22)
                            time_day[i] <= time_day[i] + 1'b1;
                    else    time_night[i] <= time_night[i] + 1'b1;
                end
                else    begin time_day[i] <= 0; time_night[i] <= 0; end
            end
```

```
131         end
132     end
133 
134     /*****************************     计时部分     *****************************/
135 
136     /*****************************     缴费部分     *****************************/
137 
138     // 多位正边沿检测检测，判断是否有车辆驶离
139     reg [7:0] swt_dly = 8'b1111_1111;
140     reg [7:0] swt_edge = 0;
141 
142     always @(posedge clk) begin
143         swt_dly <= power ? de_swt_bar : 8'b1111_1111;
144         swt_edge <= power ?  (~swt_dly) & de_swt_bar : 8'b0;
145     end
146 
147     reg need_pay = 0;       // need_pay 为1 时系统处于待缴费状态
148     reg [7:0] leave = 0;    // leave 为1 的位置代表该车位处于待缴费状态
149 
150     always @(posedge clk) begin
151         // 当系统启动并且不处于待缴费状态时，检测车辆驶离
152         if (power && !need_pay && |swt_edge) begin
153             need_pay <= 1; leave <= swt_edge;
154             // 把驶离车辆的停车信息存到time_day[8]和time_night[8]中
155             case (swt_edge)
156                 8'b0000_0001 : begin time_day[8]   <= time_day[0];
157                                      time_night[8] <= time_night[0]; end
158                 8'b0000_0010 : begin time_day[8]   <= time_day[1];
159                                      time_night[8] <= time_night[1]; end
160                 8'b0000_0100 : begin time_day[8]   <= time_day[2];
161                                      time_night[8] <= time_night[2]; end
162                 8'b0000_1000 : begin time_day[8]   <= time_day[3];
163                                      time_night[8] <= time_night[3]; end
164                 8'b0001_0000 : begin time_day[8]   <= time_day[4];
165                                      time_night[8] <= time_night[4]; end
166                 8'b0010_0000 : begin time_day[8]   <= time_day[5];
167                                      time_night[8] <= time_night[5]; end
168                 8'b0100_0000 : begin time_day[8]   <= time_day[6];
169                                      time_night[8] <= time_night[6]; end
170                 8'b1000_0000 : begin time_day[8]   <= time_day[7];
171                                      time_night[8] <= time_night[7]; end
172             endcase
173         end
174         // 当系统未启动或者按下缴费按键时，系统结束待缴费状态
175         if (!power || pay_pulse)    begin need_pay <= 0; leave <= 0; end
176     end
177 
178     // 1 位移位寄存器，以2Hz 的速度左移
179     // 在need_pay 上升后，往移位寄存器加入一个1，其余时刻加0
180     // 从而产生一个的延时器
181     reg need_pay_Q1 = 0;
182     reg [1:0] delay = 0;
183 
184     always @(posedge flicker_clk) begin
185         need_pay_Q1 <= need_pay;
186         if (need_pay_Q1 & ~need_pay)  delay <= {delay[0:0], 1'b1};
187         else    delay <= {delay[0:0], 1'b0};
188     end
189 
190     /*****************************     缴费部分     *****************************/
191 
192     /*****************************     视图部分     *****************************/
```

```
193
194     LEDControl CarPort(power, (!flicker_clk || !need_pay), de_swt_bar, leave, led
195 );
196
197     // 因为数码管的个数和点阵行列数都是8，所以计数的范围为0～7
198     reg [2:0] scan_cnt = 0;
199     always @(posedge scan_clk) begin
200         scan_cnt <= (scan_cnt > 6) ? 1'b0 : scan_cnt + 1'b1;
201     end
202
203     // 当前可用的车位数可直接由拨码开关的状态相加得到
204     // 同时要考虑到待缴费状态，所以再减去 need_pay
205     wire [3:0] car = de_swt_bar[0] + de_swt_bar[1] + de_swt_bar[2] +
206                      de_swt_bar[3] + de_swt_bar[4] + de_swt_bar[5] +
207                      de_swt_bar[6] + de_swt_bar[7] - need_pay;
208
209     DotMatrix CarDotMatrix(power, (flicker_clk || !need_pay),
210                            car, scan_cnt, row, col_r, col_g);
211
212     Segment TimeCounter(power, |delay, need_pay | need_pay_Q1, flicker_clk,
213                         time_cnt, scan_cnt, time_day[8], time_night[8], dis, seg)
214 ;
215
216     // 蜂鸣器在延时器工作时发生，频率与scan_clk(500Hz)相同
217     assign buz = (|delay && power) ? scan_clk : 1'b0;
218
219     LCD LCDDisplay(rst, clk, scan_clk, power, car, time_cnt, rw, en, rs, lcd);
220
221     /*****************************     视图部分     *****************************/
222
223 endmodule
```

功能说明及资源利用情况

功能说明

1. 接通电源时，LCD 开始初始化，同时开始生成伪随机数序列，其他模块都处于不工作状态。
2. 按下电源按键后，系统启动，停止生成伪随机数序列，并将当前序列作为车位是否有效的依据。有效的车位对应的 LED 灯亮起，表示对应车位可用；点阵显示当前所有可用车位的个数；数码管 DISP7 和 DISP6 显示 0~23 秒循环计时，表示一天二十四小时；同时 LCD 也将显示时间和可用车位，同时还会指示当前是处于白天还是晚上。
3. 当某个车位对应的拨码开关上拨表示有车辆停入，这时 LED 灯熄灭，可用车位数减 1，同时开始记录此车位的停靠时间。
4. 当某个车位对应的拨码开关下拨表示有车辆驶离，这时 LED 灯和点阵以$2Hz$闪烁，DISP4 和 DISP3 两位数码管显示停车时间；DISP1 和 DISP0 两位数码管显示停车费金额。其中金额计算为白天每小时 2 元，夜间每小时 1 元。
5. 之后按下付款按键进行付款，这时 LED 灯重新亮起，可用车位数加 1，停车时间和金额显示以$2Hz$频率闪烁两次后消失，同时蜂鸣器响起提示音表示缴费成功。

6. 再次按下电源按键，系统关闭，所有模块复位。

资源利用情况

Flow Summary	
Flow Status	Successful - Thu Dec 26 18:42:43 2019
Quartus Prime Version	18.1.1 Build 646 04/11/2019 SJ Lite Edition
Revision Name	ParkingLot
Top-level Entity Name	ParkingLot
Family	MAX II
Device	EPM1270T144C5
Timing Models	Final
Total logic elements	943 / 1,270 (74 %)
Total pins	80 / 116 (69 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

表 6-1 编译报告摘要

管脚分配情况

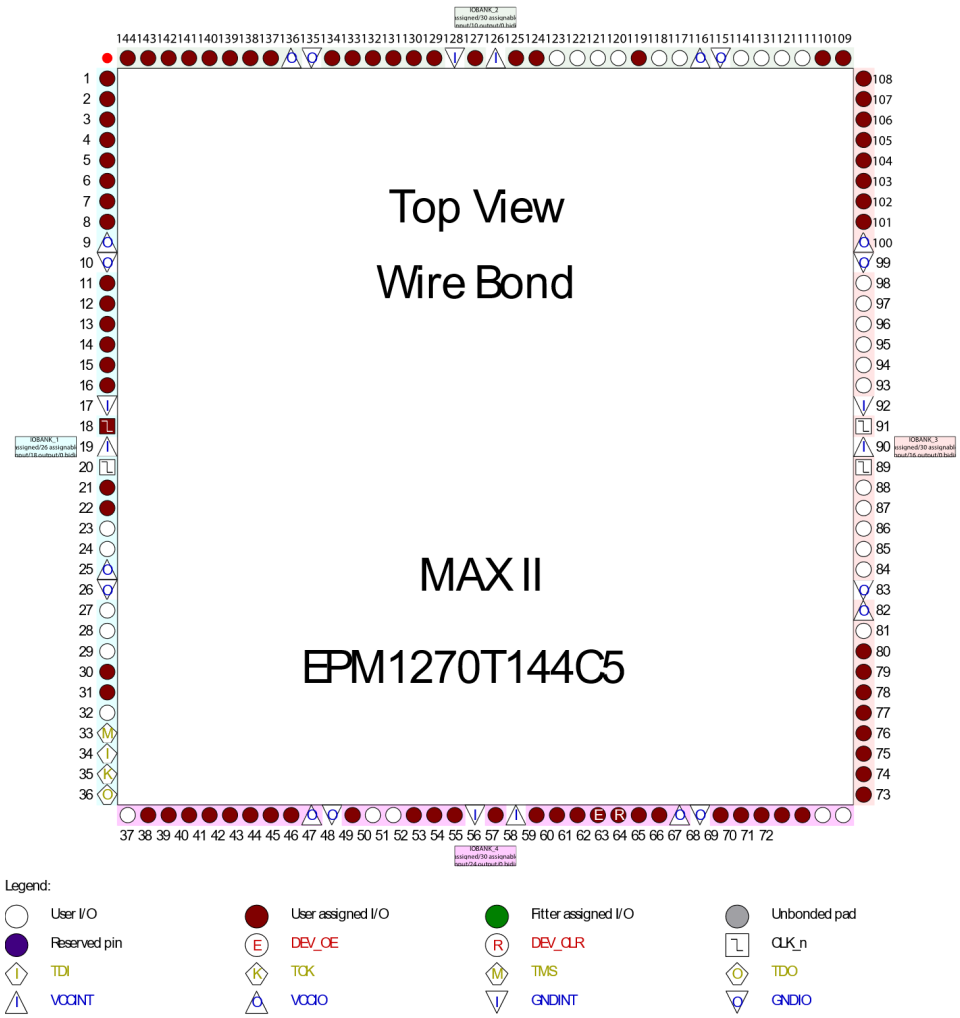

图 6-1 管脚分配图

To	Direction	Location	To	Direction	Location
col_g[7]	Output	PIN_45	rst	Input	PIN_119
col_g[6]	Output	PIN_44	buz	Output	PIN_60
col_g[5]	Output	PIN_43	clk	Input	PIN_18
col_g[4]	Output	PIN_42	pay	Input	PIN_61
col_g[3]	Output	PIN_41	pow	Input	PIN_124
col_g[2]	Output	PIN_40	row[7]	Output	PIN_8
col_g[1]	Output	PIN_39	row[6]	Output	PIN_7
col_g[0]	Output	PIN_38	row[5]	Output	PIN_6
col_r[7]	Output	PIN_22	row[4]	Output	PIN_5
col_r[6]	Output	PIN_21	row[3]	Output	PIN_4
col_r[5]	Output	PIN_16	row[2]	Output	PIN_3
col_r[4]	Output	PIN_15	row[1]	Output	PIN_2
col_r[3]	Output	PIN_14	row[0]	Output	PIN_1
col_r[2]	Output	PIN_13	rs	Output	PIN_48
col_r[1]	Output	PIN_12	en	Output	PIN_108
col_r[0]	Output	PIN_11	rw	Output	PIN_109
lcd[7]	Output	PIN_107	seg[7]	Output	PIN_62
lcd[6]	Output	PIN_106	seg[6]	Output	PIN_59
lcd[5]	Output	PIN_105	seg[5]	Output	PIN_58
lcd[4]	Output	PIN_104	seg[4]	Output	PIN_57
lcd[3]	Output	PIN_103	seg[3]	Output	PIN_55
lcd[2]	Output	PIN_102	seg[2]	Output	PIN_53
lcd[1]	Output	PIN_110	seg[1]	Output	PIN_52
lcd[0]	Output	PIN_101	seg[0]	Output	PIN_51
led[15]	Output	PIN_137	swt[7]	Input	PIN_125
led[14]	Output	PIN_138	swt[6]	Input	PIN_127
led[13]	Output	PIN_139	swt[5]	Input	PIN_129
led[12]	Output	PIN_140	swt[4]	Input	PIN_130
led[11]	Output	PIN_141	swt[3]	Input	PIN_131
led[10]	Output	PIN_142	swt[2]	Input	PIN_132
led[9]	Output	PIN_143	swt[1]	Input	PIN_133
led[8]	Output	PIN_144	swt[0]	Input	PIN_134

led[7]	Output	PIN_73	dis[7]	Output	PIN_31
led[6]	Output	PIN_74	dis[6]	Output	PIN_30
led[5]	Output	PIN_75	dis[5]	Output	PIN_70
led[4]	Output	PIN_76	dis[4]	Output	PIN_69
led[3]	Output	PIN_77	dis[3]	Output	PIN_68
led[2]	Output	PIN_78	dis[2]	Output	PIN_67
led[1]	Output	PIN_79	dis[1]	Output	PIN_66
led[0]	Output	PIN_80	dis[0]	Output	PIN_63

表 6-2 管脚分配表

故障及问题分析

7. 经常会出现系统开启后突然关闭，或者拨码开关上拨后 LED 和点阵进入闪烁状态。原因是因为没有进行消抖，使得系统接收到了“假”信号。因此对于拨码开关和按键进行了消抖，成功解决了问题。

8. 一开始的时候，付款后数码管闪烁两次的效果不一定每次都能做出来，有时候只闪 1 次，有时候闪 3 次。之后发现是因为延时器的时钟域和数码管扫描的时钟域不一致，因此在跨时钟域时不能够同步。我得到的解决方法是在按下付款按键后产生一个延时的脉冲 need_pay_Q1 作为两个时钟域的同步信号，成功解决了问题。

9. 开始的时候写了几个模块就占用了特别多的资源。所以总结了几个节省资源的经验：① 不要定义太多不必要临时变量，有一些可以共用的资源就不必要再使用新的。② 对于像计时器之类的模块，用来计数的寄存器可能会有非常多位，这是我们可以选择较慢的始终来驱动计数器，这样就可以减小计数寄存器的长度，从而节省资源。③ 对于一个功能应考虑设计为时序电路还是组合电路，正确的设计能节省不少资源。④ Quartus 中的综合选项可以选择优化电路资源，但是得到的电路执行速度会比较慢。所以在对速度要求不高时可以使用这个选项。

总结与结论

1. 进行硬件编程，特别是复杂的模块编程时，一定不能一上来就直接写代码，这样经常会陷入困难，或者产生很多冗余设计，最终的成品效果一定不好。所以应该先认真分析系统的功能，有多少种状态，需要什么操作，这种时候可以使用有限状态机（FSM）来分析。然后将具体的事项先抽象，完成顶层设计，再根据不同的功能分模块进行具体设计，确保各个模块之间的耦合良好。同时使用模块化的设计也能使开发过程更加清晰，调试仿真起来更加简单。

2. 硬件开发与软件开发具有很大的不同。首先 Verilog 每一个描述的变量都是一个实际的电路结构，它们在执行的时候都是并行操作。对于计算机程序，如果调试通过了，那么基本就确定开发成功了。但是硬件开发即使仿真成功了也不能代表到实际的电路中就可以正常使用。因为现实中的器件不是完全理想化的，可能会存在各种误差，比如门电路的延迟、按键的抖动等等。因此要确定程序是否正确，一定要在实际器件上进行多次测试。

参考文献

[1] 刘培植. 数字电路与逻辑设计[M]. 北京邮电大学出版社:北京, 2013.

[2] Faraz Khan.Random Number Generator in Verilog | FPGA[EB/OL]. https://simplefpga.blogspot.com/2013/02/random-number-generator-in-verilog-fpga.html ,2013-2-10.

[3] Tan LeTian.How to add debounce switch in Verilog code[EB/OL]. https://getsudocode.com/how-to-add-debounce-switch-in-verilog-code/,2019-3-22.

[4] Hitachi.HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)[Z]. 日本:Hitachi,1998.

[5] Donald Weiman.LCD Initialization[EB/OL]. http://web.alfredstate.edu/faculty/weimandn/lcd/lcd_initialization/lcd_initialization_index.html,2012-9-29.