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

1. [摘要 4](#_Toc28296763)
2. [设计任务要求 4](#_Toc28296764)
3. [系统设计 5](#_Toc28296765)
4. [设计思路 5](#_Toc28296766)
5. [总体框图 6](#_Toc28296767)
6. [分块设计 6](#_Toc28296768)
7. [分频模块 6](#_Toc28296769)
8. [线性反馈移位寄存器模块 6](#_Toc28296770)
9. [按键消抖模块 7](#_Toc28296771)
10. [拨码开关消抖模块 7](#_Toc28296772)
11. [LED控制模块 7](#_Toc28296773)
12. [LCD控制模块 8](#_Toc28296774)
13. [数码管显示模块 10](#_Toc28296775)
14. [点阵显示模块 11](#_Toc28296776)
15. [主模块 11](#_Toc28296777)
16. [仿真波形及波形分析 11](#_Toc28296778)
17. [分频模块 12](#_Toc28296779)
18. [线性反馈移位寄存器模块 12](#_Toc28296780)
19. [按键消抖模块 12](#_Toc28296781)
20. [拨码开关消抖模块 12](#_Toc28296782)
21. [LED控制模块 13](#_Toc28296783)
22. [LCD控制模块 13](#_Toc28296784)
23. [数码管显示模块 13](#_Toc28296785)
24. [点阵显示模块 14](#_Toc28296786)
25. [主模块 14](#_Toc28296787)
26. [代码 14](#_Toc28296788)
27. [分频模块 14](#_Toc28296789)
28. [线性反馈移位寄存器模块 15](#_Toc28296790)
29. [按键消抖模块 15](#_Toc28296791)
30. [拨码开关消抖模块 16](#_Toc28296792)
31. [LED控制模块 16](#_Toc28296793)
32. [LCD控制模块 17](#_Toc28296794)
33. [数码管显示模块 21](#_Toc28296795)
34. [点阵显示模块 22](#_Toc28296796)
35. [主模块 24](#_Toc28296797)
36. [功能说明及资源利用情况 28](#_Toc28296798)
37. [功能说明 28](#_Toc28296799)
38. [资源利用情况 29](#_Toc28296800)
39. [管脚分配情况 29](#_Toc28296801)
40. [故障及问题分析 31](#_Toc28296802)
41. [总结与结论 31](#_Toc28296803)
42. [参考文献 32](#_Toc28296804)

# 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：
     + 1. DISP7和DISP6显示0~23秒循环计时，模拟一天二十四小时；
       2. 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）；
       3. 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：
     + 1. 停车按时间计费，前2秒不计费，以后每秒2元；
       2. 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。

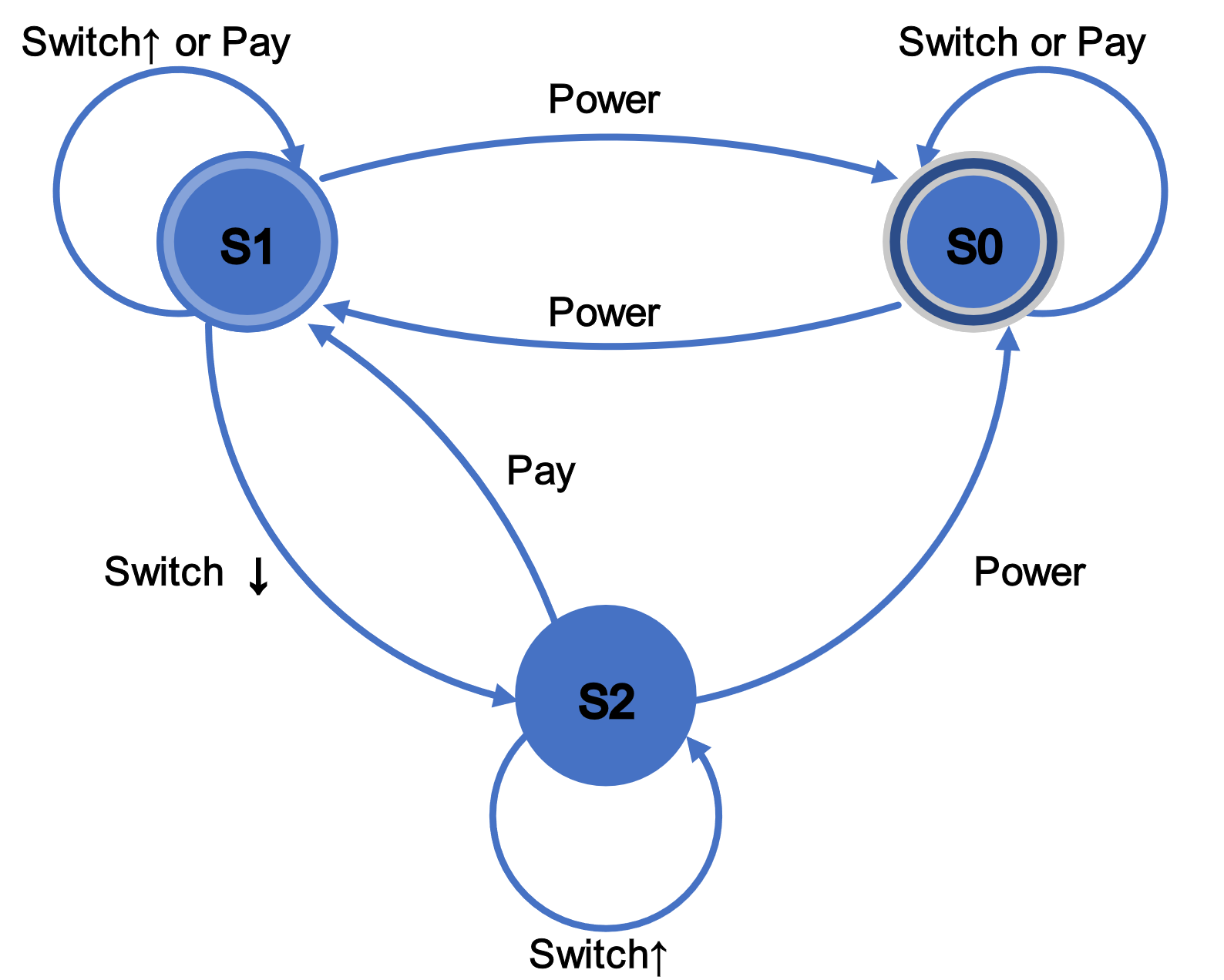


图3-1 状态图

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

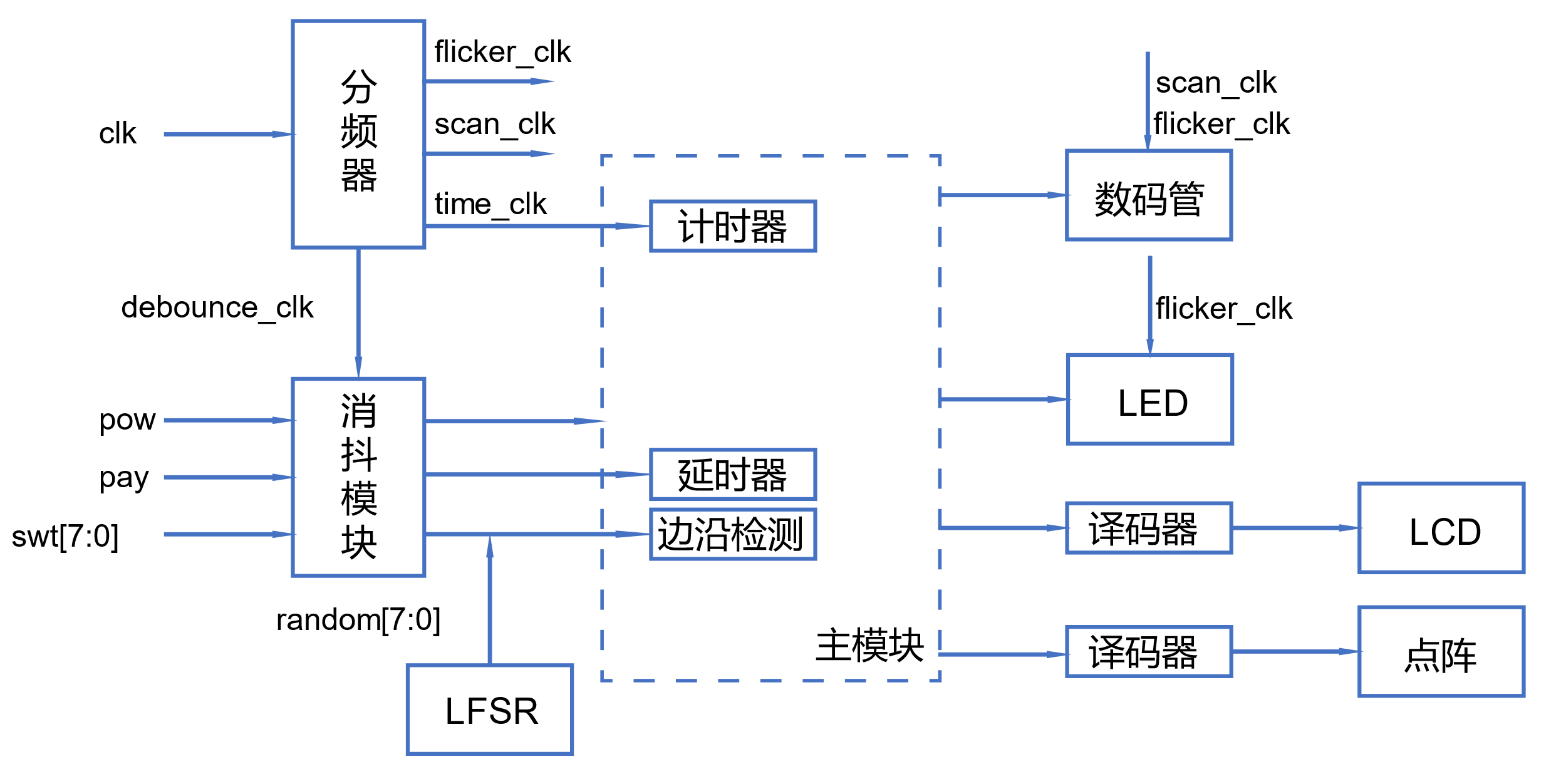


图3-2 总体结构框图

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，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 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[[2]](#参考1) ，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。

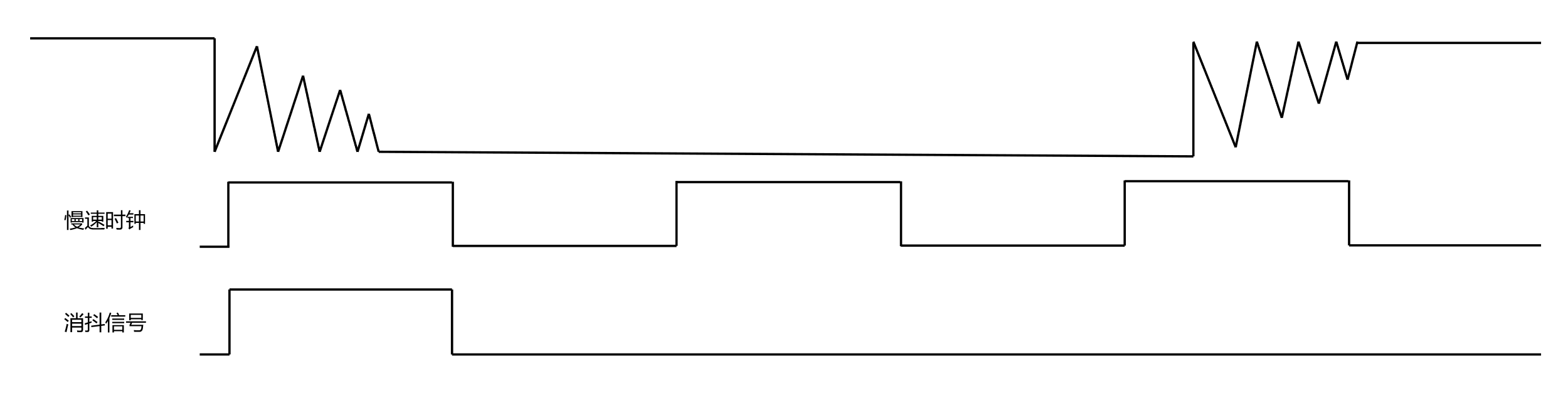


图3-4 消抖示意图

### 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]](#参考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**  **(fOSC=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 | | | | | | | | | | | DDRAM:  CGRAM:  ACG:  ADD:  AC: | 数据显示RAM  字符生成器RAM  CGRAM地址  DDRAM地址(对应于光标地址)  DDRAM和CGRAM的地址计数器 |  |
| D = 1: 开启显示  C = 1: 开启光标  B = 1: 光标闪烁  S/C = 1: 显示内容移动  R/L = 1: 向右移动  DL = 1: 8位  N = 1: 显示两行  F = 1: 5 × 10点阵  BF = 1: 内部运作 | | | | | | D = 0: 关闭显示  C = 0: 关闭光标  B = 0: 光标不闪烁  S/C = 0: 光标左移  R/L = 0: 向左移动  DL = 0: 4位  N = 0: 显示一行  F = 0: 5 × 8点阵  BF = 0: 可接受指示 | | | | |

表3-2 指令表[[4]](#参考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.

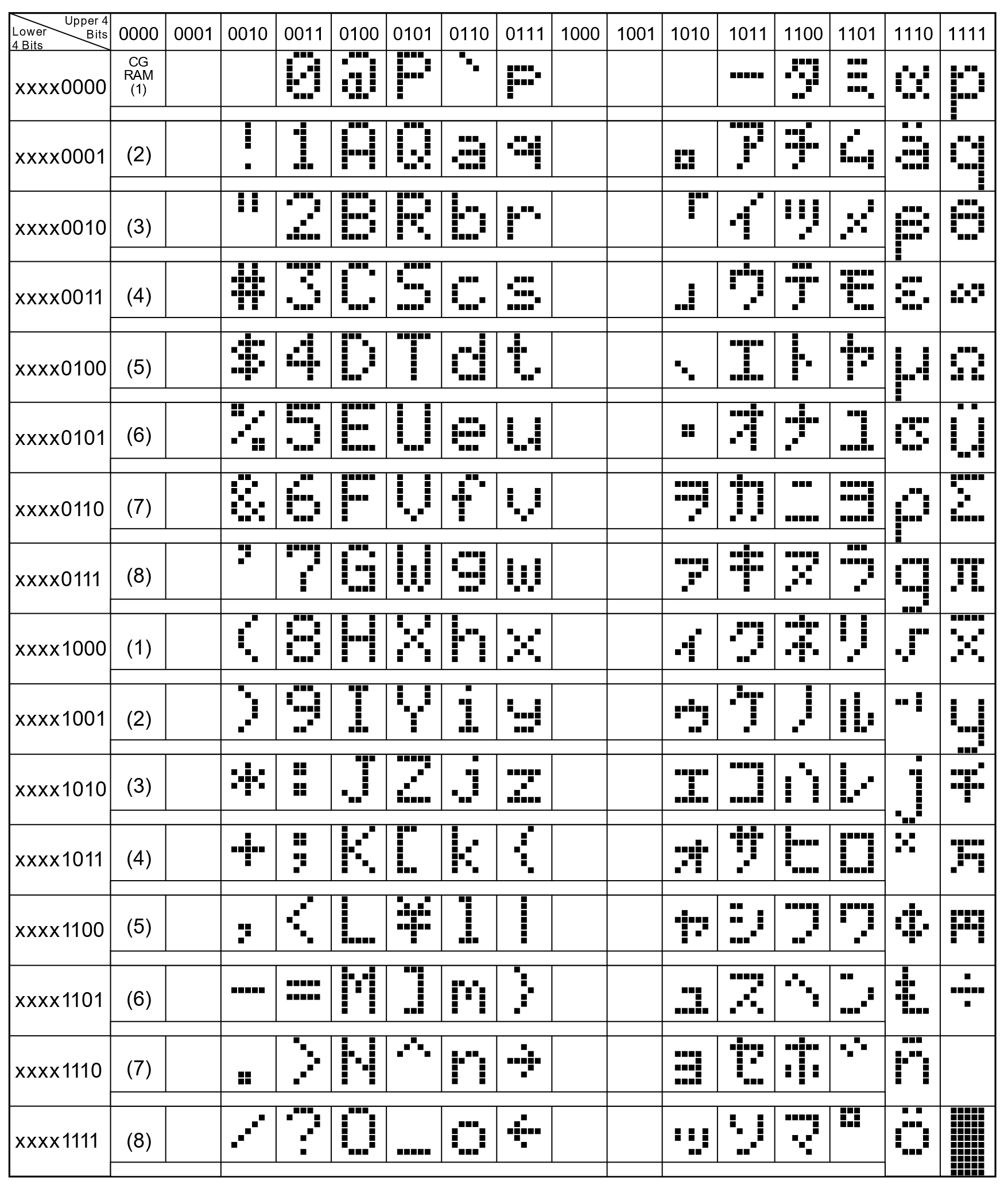


图3-5 字符对照表图[[4]](#参考4)

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）。

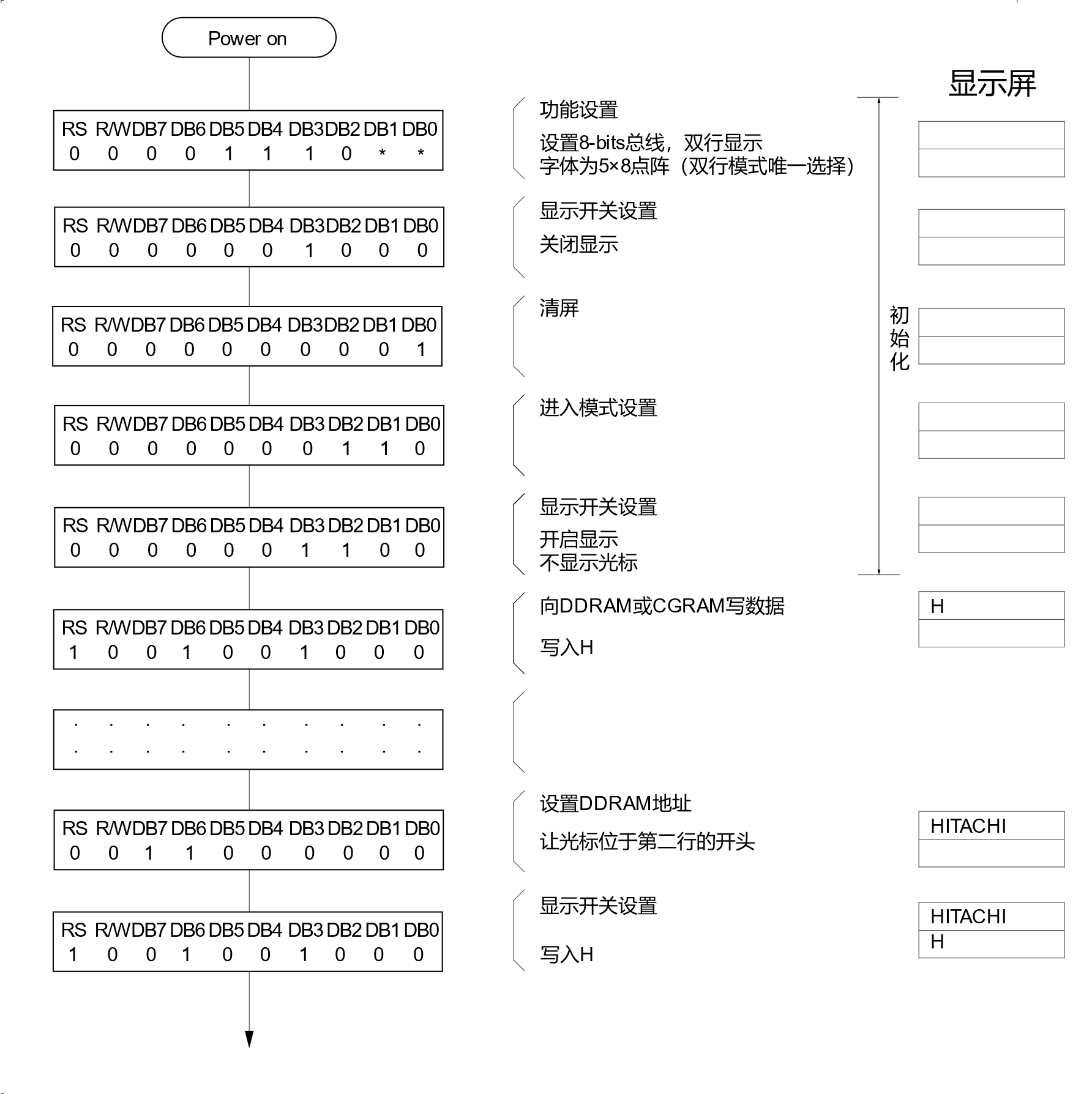


图3-6 LCD模块指令顺序图[[5]](#参考5)

### 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。

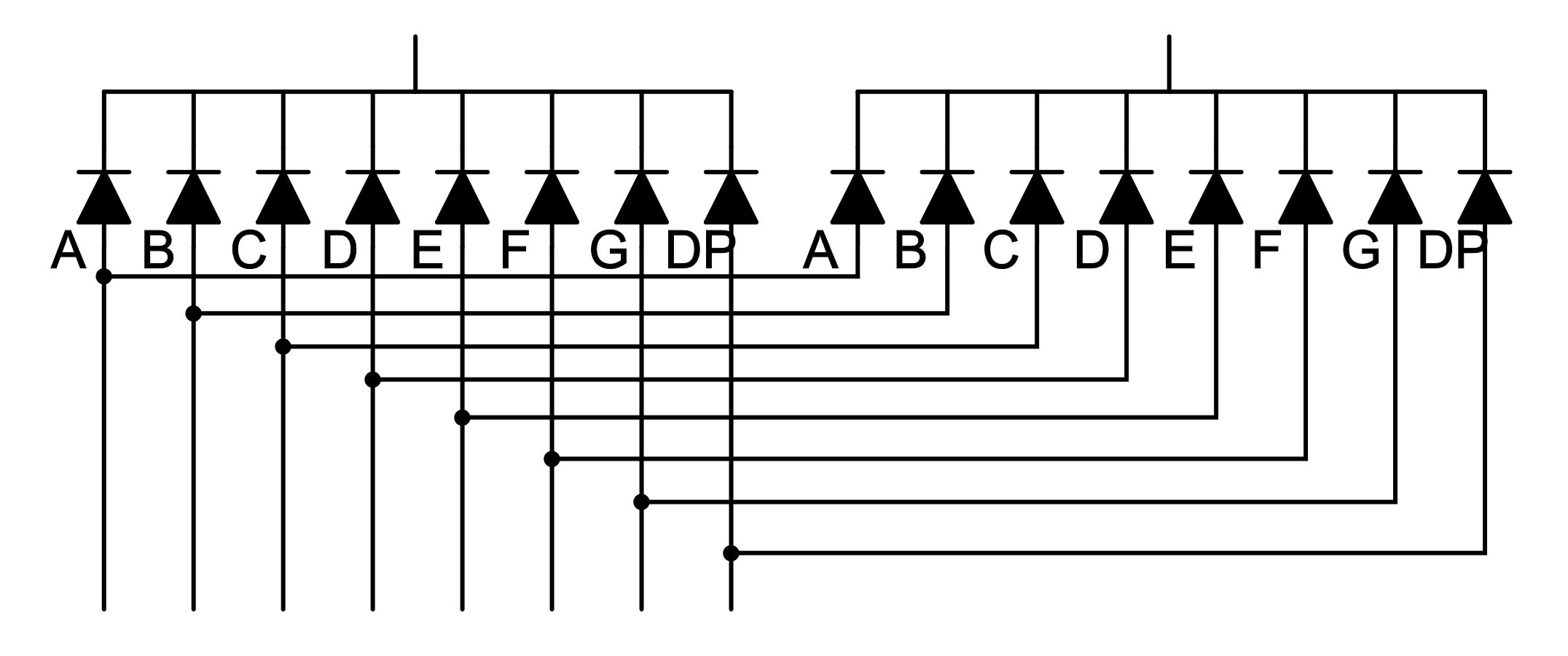


图3-7 数码管电路示意图

### 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 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.

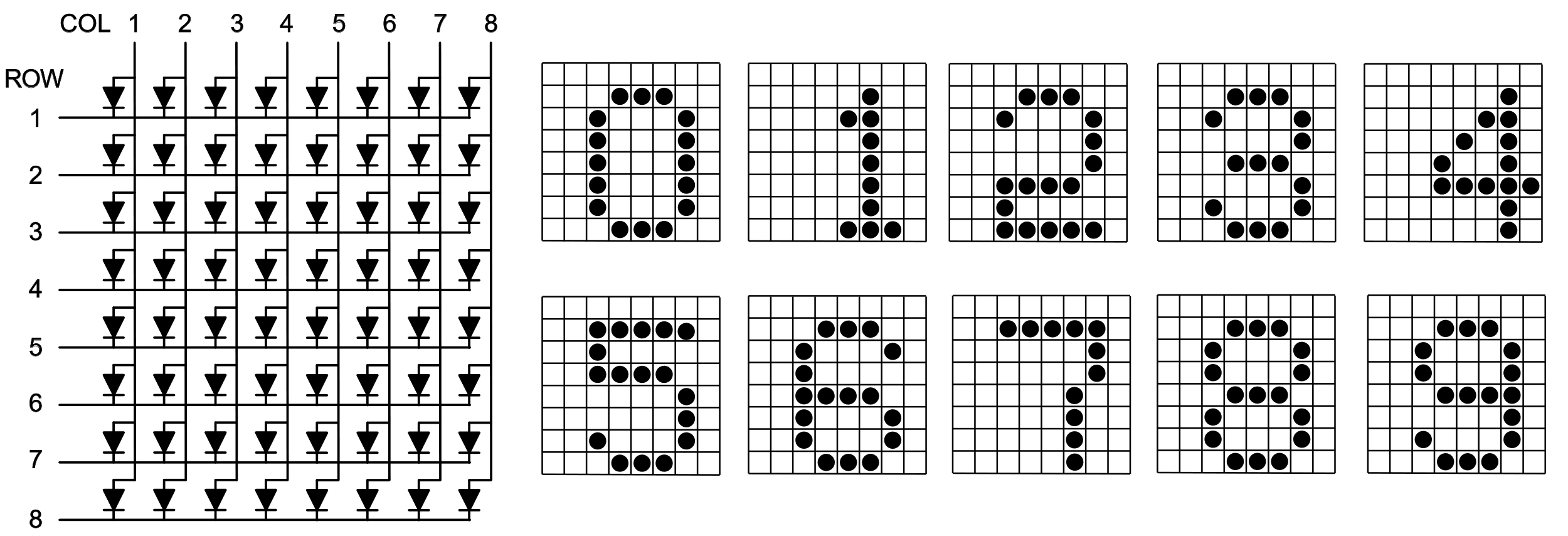


图3-8 点阵电路图以及数字显示图

### 主模块

在主模块中我用一个的时钟进行计时。用一个的时钟驱动蜂鸣器，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

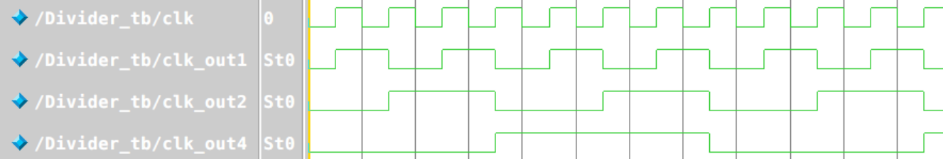


图4-1 分频模块仿真波形图

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

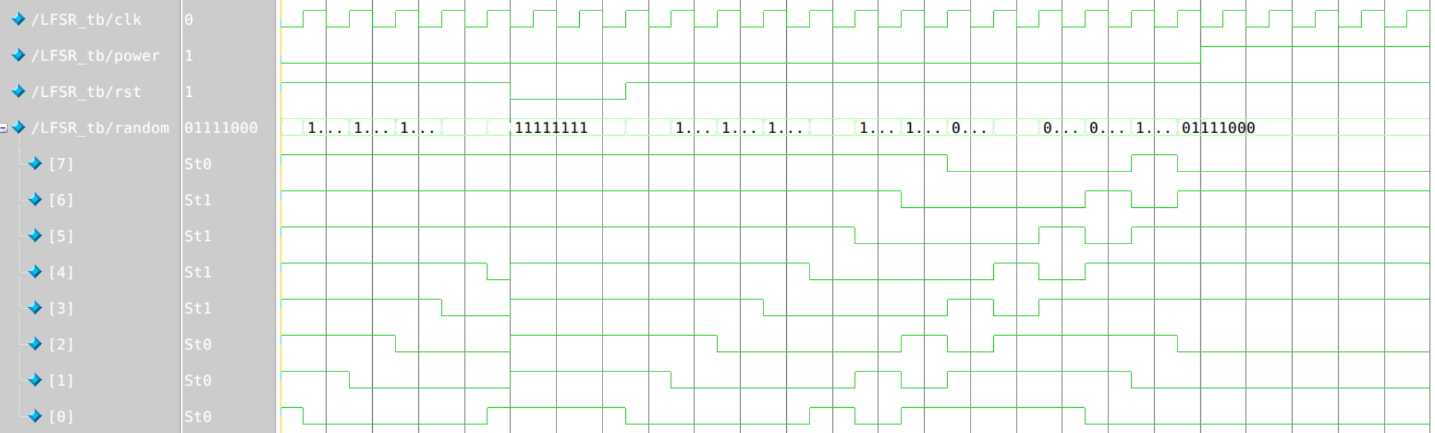


图4-2 线性反馈移位寄存器模块仿真波形图

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

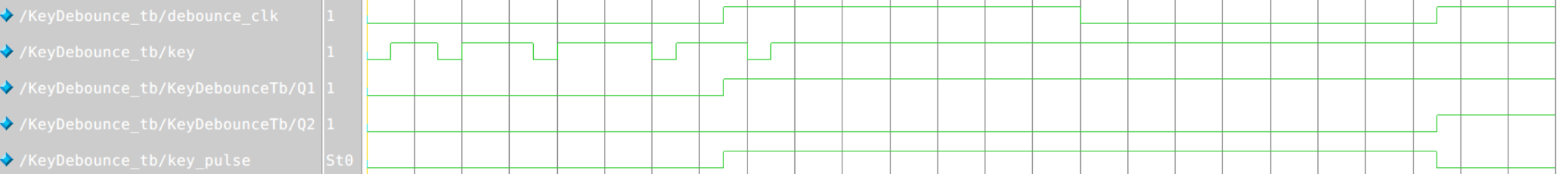


图4-3 按键消抖模块仿真波形图

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

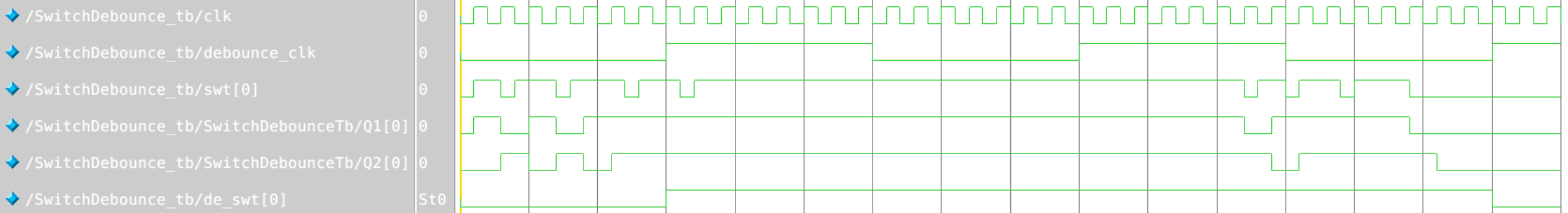


图4-4 拨码开关消抖模块仿真波形图

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

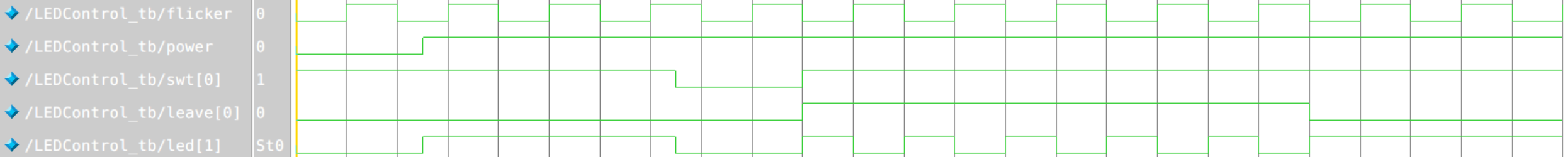


图4-5 LED控制模块仿真波形图

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

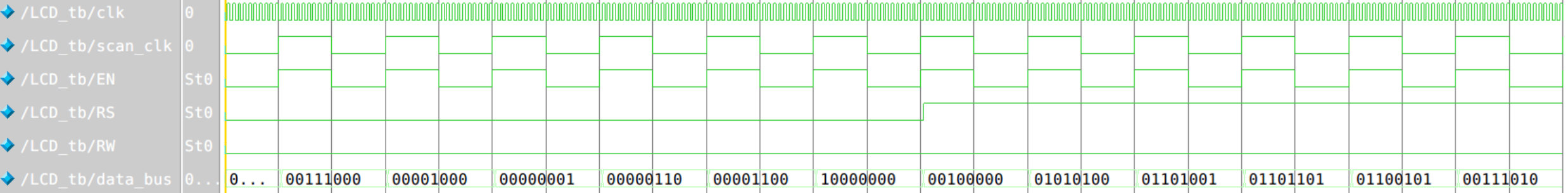


图4-6 LCD控制模块仿真波形图

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

A picture containing text, wall

Description automatically generated

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

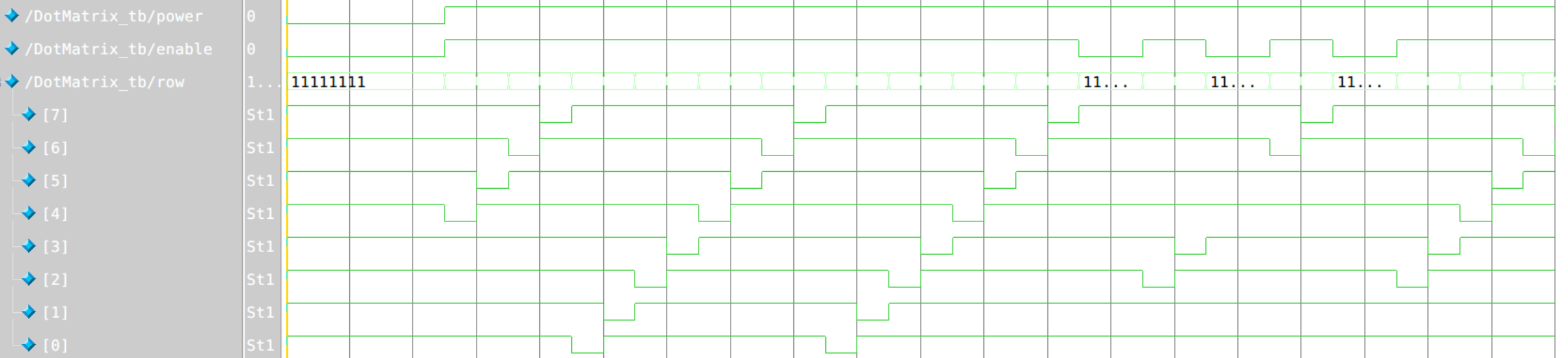


图4-8 点阵显示模块仿真波形图

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

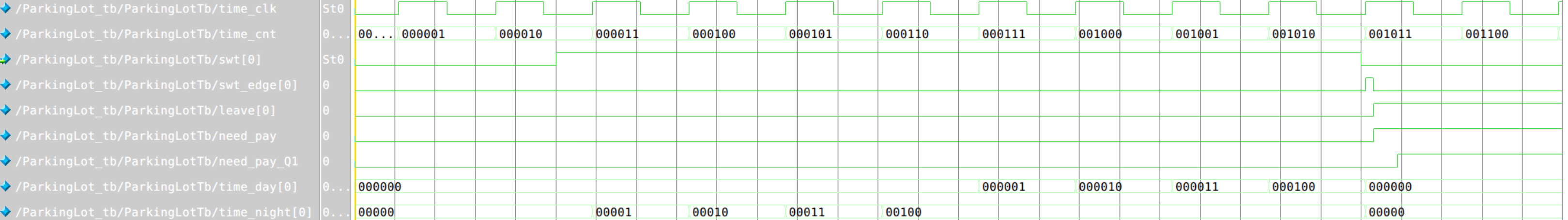


图4-9 主模块仿真波形图

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

*/\*\**

1

2

3

4

5

*\* @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*

*\*/*

6

7

8

9

10

*module* Divider (

*input* clk,

*output* *reg* clk\_out = 1'b0

);

*reg* [25:0] cnt = 26'b0;

11

12

13

14

15

16

17

18

19

20

21

22

23

*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

*/\*\**

1

2

3

4

5

6

7

*\* @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 (

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

*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

1

2

3

4

5

6

*/\*\**

*\* @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 (

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

*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;

22

23

24

*endmodule*

## 拨码开关消抖模块

*/\*\**

1

2

3

4

5

6

*\* @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 (

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

*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

*/\*\**

1

2

3

4

5

6

7

8

*\* @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 (

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

*input* power,

*input* flicker,

*input* [7:0] swt,

*input* [7:0] leave,

*output* *reg* [15:0] led = 0

);

*integer* i;

*always* *@*(\*) *begin*

        f*or* (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

*/\*\**

1

2

3

4

5

6

7

8

9

10

11

12

13

*\* @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 (

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

*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*

*end*

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

*always* *@*(*posedge* clk *or* *negedge* power) *begin*

*if* (!power) row\_2 <= "                ";

        else case (car)

            0 :     row\_2 <= " Space: 0       ";

            1 :     row\_2 <= " Space: 1       ";

            2 :     row\_2 <= " Space: 2       ";

            3 :     row\_2 <= " Space: 3       ";

            4 :     row\_2 <= " Space: 4       ";

            5 :     row\_2 <= " Space: 5       ";

            6 :     row\_2 <= " Space: 6       ";

            7 :     row\_2 <= " Space: 7       ";

            8 :     row\_2 <= " Space: 8       ";

*endcase*

*end*

*reg* Q = 0;

*always* *@*(*posedge* clk) *begin*

        Q <= scan\_clk;

*end*

*assign* RW = 1'b0;   *// 始终只进行写入*

*assign* EN = scan\_clk;

*wire* write\_flag = ~Q & scan\_clk;

*// 40个状态*

*localparam*        IDLE = 8'h00;

*// 初始化*

*localparam*    DISP\_SET = 8'h01;    *// 清屏*

*localparam*    DISP\_OFF = 8'h03;    *// 关闭显示*

*localparam*     CLR\_SCR = 8'h02;    *// 光标归位*

*localparam* CURSOR\_SET1 = 8'h06;    *// 设置光标*

*localparam* CURSOR\_SET2 = 8'h07;    *// 开启显示*

*// 显示第一行*

*localparam*   ROW1\_ADDR = 8'h05;

*localparam*      ROW1\_0 = 8'h04;

*localparam*      ROW1\_1 = 8'h0C;

*localparam*      ROW1\_2 = 8'h0D;

*localparam*      ROW1\_3 = 8'h0F;

*localparam*      ROW1\_4 = 8'h0E;

*localparam*      ROW1\_5 = 8'h0A;

*localparam*      ROW1\_6 = 8'h0B;

*localparam*      ROW1\_7 = 8'h09;

*localparam*      ROW1\_8 = 8'h08;

*localparam*      ROW1\_9 = 8'h18;

*localparam*      ROW1\_A = 8'h19;

*localparam*      ROW1\_B = 8'h1B;

*localparam*      ROW1\_C = 8'h1A;

*localparam*      ROW1\_D = 8'h1E;

*localparam*      ROW1\_E = 8'h1F;

*localparam*      ROW1\_F = 8'h1D;

*// 显示第二行*

*localparam*   ROW2\_ADDR = 8'h1C;

*localparam*      ROW2\_0 = 8'h14;

*localparam*      ROW2\_1 = 8'h15;

*localparam*      ROW2\_2 = 8'h17;

*localparam*      ROW2\_3 = 8'h16;

*localparam*      ROW2\_4 = 8'h12;

*localparam*      ROW2\_5 = 8'h13;

*localparam*      ROW2\_6 = 8'h11;

*localparam*      ROW2\_7 = 8'h10;

*localparam*      ROW2\_8 = 8'h30;

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

*localparam*      ROW2\_9 = 8'h31;

*localparam*      ROW2\_A = 8'h33;

*localparam*      ROW2\_B = 8'h32;

*localparam*      ROW2\_C = 8'h36;

*localparam*      ROW2\_D = 8'h37;

*localparam*      ROW2\_E = 8'h35;

*localparam*      ROW2\_F = 8'h34;

*reg* [7:0] current\_state = IDLE;

*reg* [7:0] next\_state = IDLE;

*always* *@*(*posedge* clk *or* *negedge* rst) *begin*

*if* (!rst)  current\_state <= IDLE;

*else* *if* (write\_flag)  current\_state <= next\_state;

*end*

*always* *@*(\*) *begin*

*case* (current\_state)

*// 初始化*

            IDLE :        next\_state = DISP\_SET;

            DISP\_SET :    next\_state = DISP\_OFF;

            DISP\_OFF :    next\_state = CLR\_SCR;

            CLR\_SCR :     next\_state = CURSOR\_SET1;

            CURSOR\_SET1 : next\_state = CURSOR\_SET2;

            CURSOR\_SET2 : next\_state = ROW1\_ADDR;

*// 显示第一行*

            ROW1\_ADDR :   next\_state = ROW1\_0;

            ROW1\_0 :      next\_state = ROW1\_1;

            ROW1\_1 :      next\_state = ROW1\_2;

            ROW1\_2 :      next\_state = ROW1\_3;

            ROW1\_3 :      next\_state = ROW1\_4;

            ROW1\_4 :      next\_state = ROW1\_5;

            ROW1\_5 :      next\_state = ROW1\_6;

            ROW1\_6 :      next\_state = ROW1\_7;

            ROW1\_7 :      next\_state = ROW1\_8;

            ROW1\_8 :      next\_state = ROW1\_9;

            ROW1\_9 :      next\_state = ROW1\_A;

            ROW1\_A :      next\_state = ROW1\_B;

            ROW1\_B :      next\_state = ROW1\_C;

            ROW1\_C :      next\_state = ROW1\_D;

            ROW1\_D :      next\_state = ROW1\_E;

            ROW1\_E :      next\_state = ROW1\_F;

            ROW1\_F :      next\_state = ROW2\_ADDR;

*// 显示第二行*

            ROW2\_ADDR :   next\_state = ROW2\_0;

            ROW2\_0 :      next\_state = ROW2\_1;

            ROW2\_1 :      next\_state = ROW2\_2;

            ROW2\_2 :      next\_state = ROW2\_3;

            ROW2\_3 :      next\_state = ROW2\_4;

            ROW2\_4 :      next\_state = ROW2\_5;

            ROW2\_5 :      next\_state = ROW2\_6;

            ROW2\_6 :      next\_state = ROW2\_7;

            ROW2\_7 :      next\_state = ROW2\_8;

            ROW2\_8 :      next\_state = ROW2\_9;

            ROW2\_9 :      next\_state = ROW2\_A;

            ROW2\_A :      next\_state = ROW2\_B;

            ROW2\_B :      next\_state = ROW2\_C;

            ROW2\_C :      next\_state = ROW2\_D;

            ROW2\_D :      next\_state = ROW2\_E;

            ROW2\_E :      next\_state = ROW2\_F;

            ROW2\_F :      next\_state = ROW1\_ADDR;

*default* :     next\_state = IDLE;

*endcase*

*end*

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

*always* *@*(*posedge* clk *or* *negedge* rst) *begin*

*if* (!rst) RS <= 1'b0;

*else* *if* (write\_flag) *begin*

*if* (next\_state == IDLE        || next\_state == DISP\_SET    ||

                next\_state == DISP\_OFF    || next\_state == CLR\_SCR     ||

                next\_state == CURSOR\_SET1 || next\_state == CURSOR\_SET2 ||

                next\_state == ROW1\_ADDR   || next\_state == ROW2\_ADDR)

                    RS <= 1'b0;     *//写入指令*

*else*    RS <= 1'b1;     *//写入数据*

*end*

*end*

*always* *@*(*posedge* clk *or* *negedge* rst) *begin*

*if* (!rst)             data\_bus <= 8'hxx;

*else* *if* (write\_flag)

*case* (next\_state)

                IDLE :        data\_bus <= 8'hxx;

*// 初始化*

                DISP\_SET :    data\_bus <= 8'h38;

                DISP\_OFF :    data\_bus <= 8'h08;

                CLR\_SCR :     data\_bus <= 8'h01;

                CURSOR\_SET1 : data\_bus <= 8'h06;

                CURSOR\_SET2 : data\_bus <= 8'h0c;

*// 显示第一行*

                ROW1\_ADDR :   data\_bus <= 8'h80;

                ROW1\_0 :      data\_bus <= row\_1[127:120];

                ROW1\_1 :      data\_bus <= row\_1[119:112];

                ROW1\_2 :      data\_bus <= row\_1[111:104];

                ROW1\_3 :      data\_bus <= row\_1[103:96 ];

                ROW1\_4 :      data\_bus <= row\_1[95 :88 ];

                ROW1\_5 :      data\_bus <= row\_1[87 :80 ];

                ROW1\_6 :      data\_bus <= row\_1[79 :72 ];

                ROW1\_7 :      data\_bus <= row\_1[71 :64 ];

                ROW1\_8 :      data\_bus <= row\_1[63 :56 ];

                ROW1\_9 :      data\_bus <= row\_1[55 :48 ];

                ROW1\_A :      data\_bus <= row\_1[47 :40 ];

                ROW1\_B :      data\_bus <= row\_1[39 :32 ];

                ROW1\_C :      data\_bus <= row\_1[31 :24 ];

                ROW1\_D :      data\_bus <= row\_1[23 :16 ];

                ROW1\_E :      data\_bus <= row\_1[15 :8  ];

                ROW1\_F :      data\_bus <= row\_1[7  :0  ];

*// 显示第二行*

                ROW2\_ADDR :   data\_bus <= 8'hC0;

                ROW2\_0 :      data\_bus <= row\_2[127:120];

                ROW2\_1 :      data\_bus <= row\_2[119:112];

                ROW2\_2 :      data\_bus <= row\_2[111:104];

                ROW2\_3 :      data\_bus <= row\_2[103:96 ];

                ROW2\_4 :      data\_bus <= row\_2[95 :88 ];

                ROW2\_5 :      data\_bus <= row\_2[87 :80 ];

                ROW2\_6 :      data\_bus <= row\_2[79 :72 ];

                ROW2\_7 :      data\_bus <= row\_2[71 :64 ];

                ROW2\_8 :      data\_bus <= row\_2[63 :56 ];

                ROW2\_9 :      data\_bus <= row\_2[55 :48 ];

                ROW2\_A :      data\_bus <= row\_2[47 :40 ];

                ROW2\_B :      data\_bus <= row\_2[39 :32 ];

                ROW2\_C :      data\_bus <= row\_2[31 :24 ];

                ROW2\_D :      data\_bus <= row\_2[23 :16 ];

                ROW2\_E :      data\_bus <= row\_2[15 :8  ];

                ROW2\_F :      data\_bus <= row\_2[7  :0  ];

*endcase*

*end*

247

248

249

*endmodule*

## 数码管显示模块

*/\*\**

1

2

3

4

5

6

7

8

9

10

11

12

13

*\* @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 (

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

*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;

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

        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

*/\*\**

1

2

3

4

5

6

7

8

9

10

*\* @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 (

11

12

13

14

15

16

17

18

19

20

21

22

23

24

*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;

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

                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*

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

                    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

*/\*\**

1

2

3

4

5

6

7

8

*\* @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*

9

10

11

12

13

14

15

16

17

18

19

*\* @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 (

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

*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);

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

*// 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*

        f*or* (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*

*end*

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

*end*

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

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    缴费部分    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// 多位正边沿检测检测，判断是否有车辆驶离*

*reg* [7:0] swt\_dly = 8'b1111\_1111;

*reg* [7:0] swt\_edge = 0;

*always* *@*(*posedge* clk) *begin*

        swt\_dly <= power ? de\_swt\_bar : 8'b1111\_1111;

        swt\_edge <= power ?  (~swt\_dly) & de\_swt\_bar : 8'b0;

*end*

*reg* need\_pay = 0;       *// need\_pay为1时系统处于待缴费状态*

*reg* [7:0] leave = 0;    *// leave为1的位置代表该车位处于待缴费状态*

*always* *@*(*posedge* clk) *begin*

*// 当系统启动并且不处于待缴费状态时，检测车辆驶离*

*if* (power && !need\_pay && |swt\_edge) *begin*

            need\_pay <= 1; leave <= swt\_edge;

*// 把驶离车辆的停车信息存到time\_day[8]和time\_night[8]中*

*case* (swt\_edge)

                8'b0000\_0001 : *begin* time\_day[8]   <= time\_day[0];

                                     time\_night[8] <= time\_night[0]; *end*

                8'b0000\_0010 : *begin* time\_day[8]   <= time\_day[1];

                                     time\_night[8] <= time\_night[1]; *end*

                8'b0000\_0100 : *begin* time\_day[8]   <= time\_day[2];

                                     time\_night[8] <= time\_night[2]; *end*

                8'b0000\_1000 : *begin* time\_day[8]   <= time\_day[3];

                                     time\_night[8] <= time\_night[3]; *end*

                8'b0001\_0000 : *begin* time\_day[8]   <= time\_day[4];

                                     time\_night[8] <= time\_night[4]; *end*

                8'b0010\_0000 : *begin* time\_day[8]   <= time\_day[5];

                                     time\_night[8] <= time\_night[5]; *end*

                8'b0100\_0000 : *begin* time\_day[8]   <= time\_day[6];

                                     time\_night[8] <= time\_night[6]; *end*

                8'b1000\_0000 : *begin* time\_day[8]   <= time\_day[7];

                                     time\_night[8] <= time\_night[7]; *end*

*endcase*

*end*

*// 当系统未启动或者按下缴费按键时，系统结束待缴费状态*

*if* (!power || pay\_pulse)    *begin* need\_pay <= 0; leave <= 0; *end*

*end*

*// 1位移位寄存器，以2Hz的速度左移*

*// 在need\_pay上升后，往移位寄存器加入一个1，其余时刻加0*

*// 从而产生一个的延时器*

*reg* need\_pay\_Q1 = 0;

*reg* [1:0] delay = 0;

*always* *@*(*posedge* flicker\_clk) *begin*

        need\_pay\_Q1 <= need\_pay;

*if* (need\_pay\_Q1 & ~need\_pay)  delay <= {delay[0:0], 1'b1};

*else*    delay <= {delay[0:0], 1'b0};

*end*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    缴费部分    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    视图部分    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

    LEDControl CarPort(power, (!flicker\_clk || !need\_pay), de\_swt\_bar, leave, led);

*// 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*

*reg* [2:0] scan\_cnt = 0;

*always* *@*(*posedge* scan\_clk) *begin*

        scan\_cnt <= (scan\_cnt > 6) ? 1'b0 : scan\_cnt + 1'b1;

*end*

*// 当前可用的车位数可直接由拨码开关的状态相加得到*

*// 同时要考虑到待缴费状态，所以再减去 need\_pay*

*wire* [3:0] car = de\_swt\_bar[0] + de\_swt\_bar[1] + de\_swt\_bar[2] +

                     de\_swt\_bar[3] + de\_swt\_bar[4] + de\_swt\_bar[5] +

                     de\_swt\_bar[6] + de\_swt\_bar[7] - need\_pay;

    DotMatrix CarDotMatrix(power, (flicker\_clk || !need\_pay),

                           car, scan\_cnt, row, col\_r, col\_g);

    Segment TimeCounter(power, |delay, need\_pay | need\_pay\_Q1, flicker\_clk,

                        time\_cnt, scan\_cnt, time\_day[8], time\_night[8], dis, seg);

*// 蜂鸣器在延时器工作时发生，频率与scan\_clk(500Hz)相同*

*assign* buz = (|delay && power) ? scan\_clk : 1'b0;

    LCD LCDDisplay(rst, clk, scan\_clk, power, car, time\_cnt, rw, en, rs, lcd);

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    视图部分    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*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

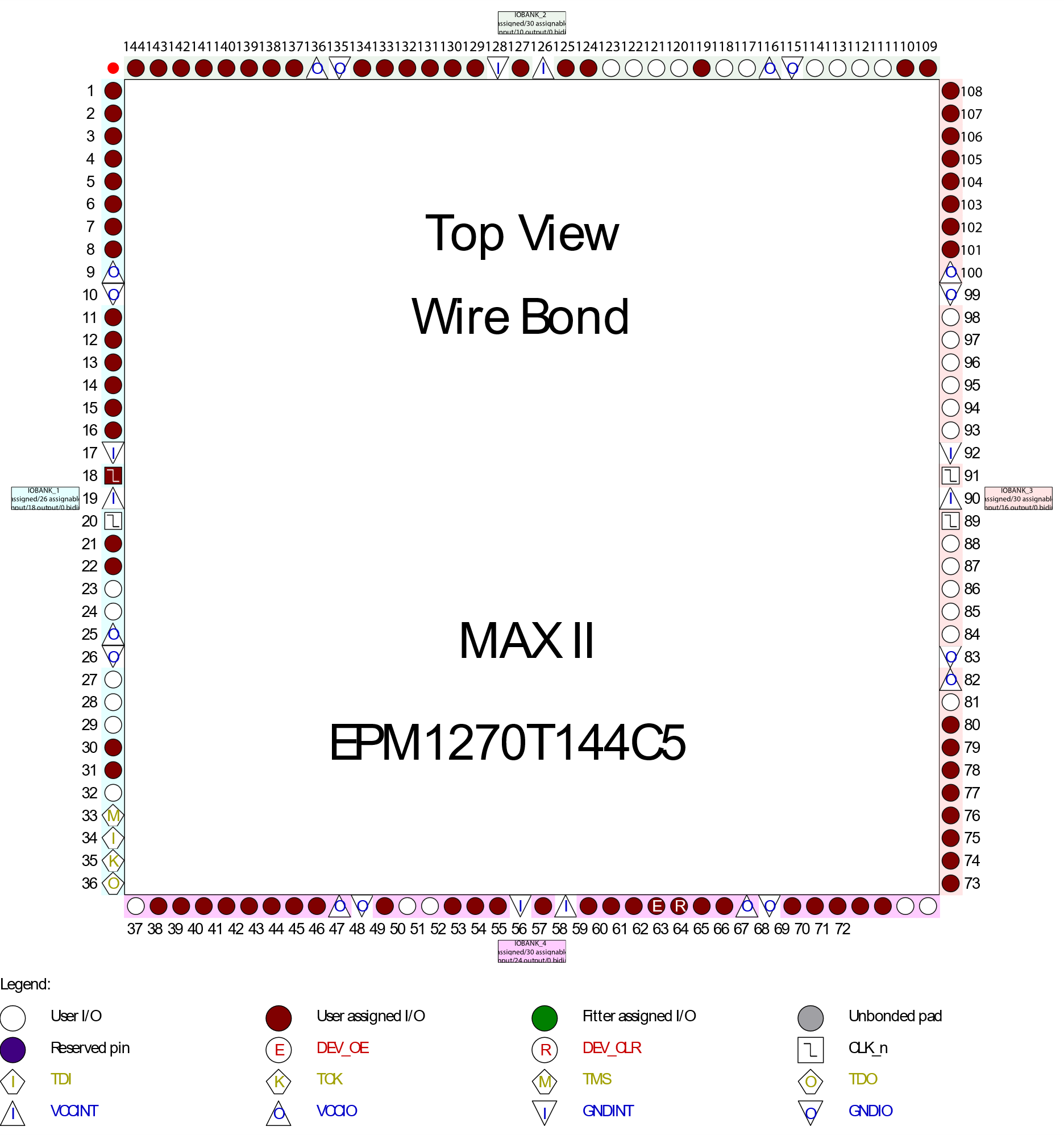


图6-1 管脚分配图

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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

1. 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。
2. 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。
3. 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.