**河南科技大学**

**课 程 设 计 说 明 书**

**课程名称 FPGA课程设计**

**题 目 ＿＿＿＿＿电子密码锁设计＿＿＿＿＿＿＿**

**学 院 ＿＿＿＿信息工程学院＿＿＿＿**

**班 级 ＿＿＿＿＿电子221＿＿＿＿＿**

**学生姓名 ＿＿＿＿＿赵文晨＿＿＿＿＿＿**

**学 号 ＿＿＿＿？？？？？？＿＿＿＿**

**指导教师 ＿＿＿张雷鸣＿＿齐晶晶＿＿＿**

**日 期 ＿** 2025年2月17日**＿＿**

**课程设计任务书**

课程设计名称 FPGA课程设计 学生姓名 赵文晨 专业班级 电子221

设计题目 **电子密码锁设计**

1. 课程设计目的

1、综合运用EDA技术，独立完成一个课题的设计，考察运用所学知识，解决实际问题的能力；

2、结合理论知识，考察阅读参考资料、文献、手册的能力；

3、进一步熟悉EDA技术的开发流程，掌握文件编辑、编译、仿真、下载验证等环节的实现方法和应用技巧；

4、锻炼撰写研究报告、研究论文的能力；

5、通过本实践环节，培养科学和严谨的工作作风。

1. 设计内容、技术条件和要求

## 1、设计一个密码锁的控制电路，当输入正确代码时，输出开锁信号以推动执行机构工作，用红灯亮、绿灯熄灭表示关锁，用绿灯亮、红灯熄灭表示开锁；

2、在锁的控制电路中储存一个可以修改的4位二进制代码，当开锁按钮开关的输入代码等于储存代码时，开锁；

1. 从第一个按钮触动后的30秒内若未将锁打开，则电路长报警30S，若输入密码错误1或2次，则每次短报警5S，若输错三次密码则长报警。
2. 时间进度安排

1周：(1) 完成设计准备，确定实施方案；(2) 完成电路文件的输入和编译；(4) 完成功能仿真。

2周：(1) 完成文件至器件的下载，并进行硬件验证；(2) 撰写设计说明书。

1. 主要参考文献

(1)谭会生、瞿遂春，《EDA技术综合应用实例与分析》，西安电子科技大学出版社，2004

(2)高有堂，《EDA技术及应用实践》，清华大学出版社，2006

(3)亿特科技，CPLD/FPGA应用系统设计与产品开发》，人民邮电出版社，2005

(4)曹昕燕、周凤臣等，《EDA技术实验与课程设计》，清华大学出版社，2006

指导教师签字： 2025年2月17日

目录

[一、任务 1](#_Toc6444)

[1.课程设计基本要求 1](#_Toc14120)

[2.拓展 1](#_Toc28803)

[二、思路 1](#_Toc23102)

[三、具体设计方案方法 3](#_Toc26182)

[1.矩阵键盘 3](#_Toc29836)

[1.矩阵键盘工作原理 3](#_Toc28491)

[2.扫描频率分析 3](#_Toc29125)

[3.消抖原理与实现 3](#_Toc8926)

[4.模块设计与功能 3](#_Toc26149)

[5.模块程序流程图和仿真 5](#_Toc712)

[2.密码锁程序 5](#_Toc30378)

[1.程序概述 5](#_Toc7810)

[2.功能介绍 6](#_Toc1203)

[无按键处理 6](#_Toc16478)

[3.报警信号生成设计 6](#_Toc27730)

[4.程序流程图和仿真 7](#_Toc3723)

[3. 数码管显示 7](#_Toc11475)

[1.数码管显示程序概述 7](#_Toc7570)

[2.时钟分频 8](#_Toc6471)

[3.扫描计数器与位选信号控制 8](#_Toc27198)

[4.数据转换与显示数据存储 8](#_Toc24898)

[5.段选信号生成 8](#_Toc23532)

[6.程序流程图和仿真 9](#_Toc13437)

[4.报警 10](#_Toc23057)

[1.程序概述 10](#_Toc174)

[2.计时参数定义 10](#_Toc5946)

[3. LED4报警功能介绍 10](#_Toc796)

[4. LED5报警逻辑设计 10](#_Toc1240)

[5.程序流程图和仿真 11](#_Toc29611)

[5.实测照片和编译结果 12](#_Toc16015)

[四、困难 13](#_Toc32121)

[1.数码管亮度低 13](#_Toc28904)

[2.矩阵键盘消抖 13](#_Toc6472)

[3.各种复杂的计时 13](#_Toc7821)

[4.调试仿真 13](#_Toc16452)

[5.模块之间的调用 14](#_Toc125)

[五、课程设计收获 14](#_Toc25062)

[1.掌握FPGA开发基本流程 14](#_Toc24800)

[2.复杂问题解决能力提升 14](#_Toc7683)

[3.培养工程思维 14](#_Toc3417)

[六、参考文献 15](#_Toc3146)

[七、程序代码 15](#_Toc9483)

# 一、任务

## 1.课程设计基本要求

## 1.设计一个密码锁的控制电路，当输入正确代码时，输出开锁信号以推动执行机构工作，用红灯亮、绿灯熄灭表示关锁，用绿灯亮、红灯熄灭表示开锁；

## 2.在锁的控制电路中储存一个可以修改的4位二进制代码，当开锁按钮开关的输入代码等于储存代码时，开锁；

## 3.从第一个按钮触动后的30秒内若未将锁打开，则电路长报警30S，若输入密码错误1或2次，则每次短报警5S，若输错三次密码则长报警。

## 2.拓展

## 1.密码为0~8位10进制（0位就是无密码）任意设定

## 2.修改密码需要两次输入并且两次一样才能成功

## 3.修改密码成功会有闪烁提示

## 4.增加了一键上锁

## 5.开锁状态下无操作10s可以自动上锁

## 6.开锁状态可以查看当前密码

# 二、思路

## 设计一个密码锁，首先输入设备是必不可少的，根据实验箱上的资源，我选择矩阵键盘作为输入，这个和市场上的密码锁输入方式更为贴近。其次要有输出，总不能让用户凭想象吧，有视觉效果最为直观，于是选用led灯来传达开锁与关锁，光有开关锁的指示感觉还是不够，输入密码到底输入进去了没有应该要能看见吧，那我就用数码管来显示用户的输入。有了输入输出，输入啥？输出啥？怎么处理？这就需要逻辑上的功能，即密码锁程序，用矩阵键盘输入键值，传递给密码锁程序，经过处理，输出led显示和数码管显示，这就是最基本的系统结构。根据题目要求，报警程序（由于没有报警器，实际上用led代替）也需要有，就组成了密码锁系统的四大模块：矩阵键盘、密码锁程序、数码管显示、报警。rtl视图和系统框图如下：

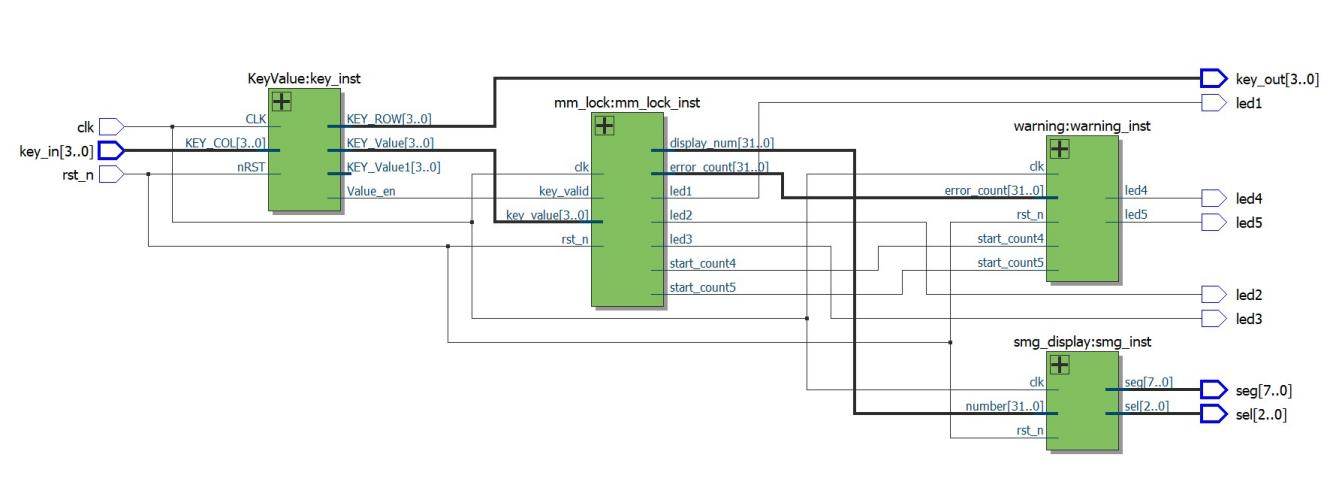


图1

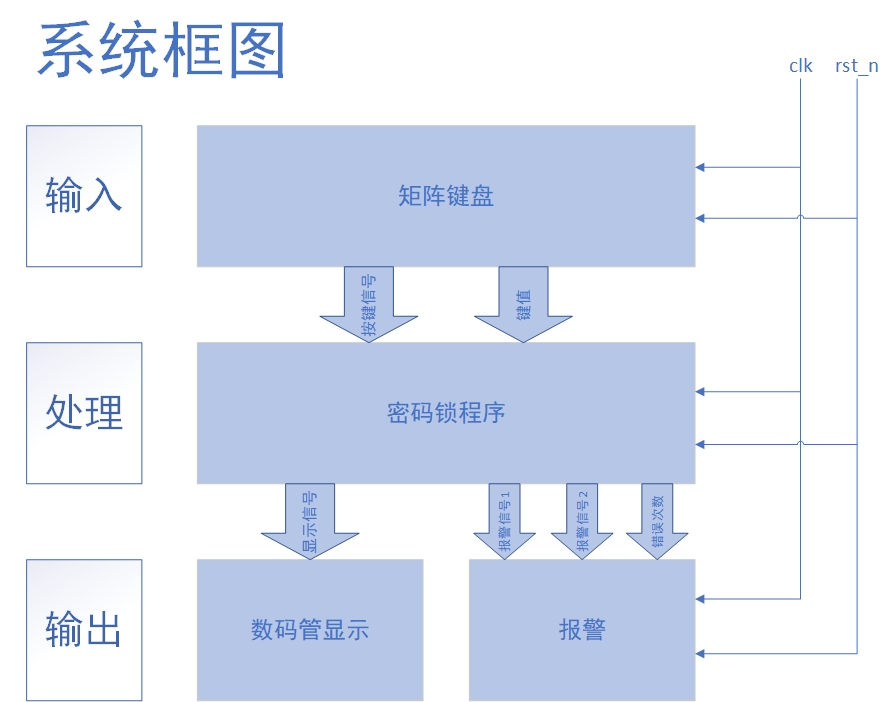


图2

# 三、具体设计方案方法

## 1.矩阵键盘

### 1.矩阵键盘工作原理

## 本设计采用的是 4x4 的矩阵键盘，由行线和列线组成，行线和列线的交叉点即为按键的位置。要识别按键，首先固定输出4行为高电平，然后输出4列为低电平，如果读入的4行有一位为低电平，那么对应的该行肯定有一个按键按下，这样便可以获取到按键的行值。同理，获取列值也是如此，先输出4列为高电平，然后在输出4行为低电平，再读入列值，如果其中有哪一位为低电平，那么肯定对应的那一列有按键按下。

### 2.扫描频率分析

在本设计中，扫描频率由时钟信号CLK决定。在KeyPress模块中，消抖计数使用了 20 位计数器，当计数到 999999 时认为消抖完成。假设时钟频率为 f，则消抖时间为 T = （999999 + 1） / f。为了保证能够及时检测到按键按下，扫描频率应该至少是消抖时间的倒数。时钟频率为50MHz，则消抖时间为 T = （999999 + 1） / 50MHz = 20ms，扫描频率不能低于 1 / 20ms = 50Hz。

### 3.消抖原理与实现

在按键按下或释放的过程中会产生短暂的电平抖动现象，影响系统的稳定性和准确性。为了消除这种抖动，本设计采用了软件消抖的方法。软件消抖的基本原理是通过状态机和计数器来实现。具体过程如下：

****状态机定义****：定义了四个状态，分别为Key\_up（按键未按下）、Filter\_Up2Down（正在按下消抖）、Key\_down（按键按下）和Filter\_Down2Up（正在释放消抖）。状态机根据按键输入信号的变化在这些状态之间进行转换。

****检测跳变沿****：通过对按键输入信号进行两级寄存器缓存，将当前输入信号存储在key\_a中，前一个时钟周期的输入信号存储在key\_b中。通过比较key\_a和key\_b的值，检测按键信号的上升沿（flag\_L2H）和下降沿（flag\_H2L）。

****计数消抖****：当检测到按键信号的跳变沿时，启动计数器开始计数。如果在计数过程中，按键信号再次发生跳变，则认为是抖动，计数器清零，状态机回到原来的状态；如果计数器计数到设定值（本设计为 999999），则认为按键状态稳定，更新按键状态和标志位。

### 4.模块设计与功能

本设计的矩阵键盘模块主要由两个小模块组成：KeyValue模块和KeyPress模块。

##### **KeyValue模块**

该模块的主要功能是进行矩阵键盘的键值扫描判断。它通过状态机依次扫描每一行，检测列线的电平变化，从而确定按键的位置，并计算对应的键值。具体实现过程如下：

****状态机控制行扫描****：使用一个 5 位的状态机，定义了NO\_KEY（无按键按下）、ROW\_ONE（扫描第一行）、ROW\_TWO（扫描第二行）、ROW\_THREE（扫描第三行）和 ROW\_FOUR（扫描第四行）等状态。在每个状态下，将相应的行线拉低，检测列线电平。如果检测到列线电平变化，则记录行索引，并跳转到NO\_KEY状态。

****确定列索引****：根据消抖后的列线状态key\_state，判断按下按键所在的列，确定列索引。

****计算键值****：根据行索引和列索引，通过公式 KEY\_Value1 = 4 \* rowIndex + colIndex 计算出初步的键值，再根据预先设定的编码规则将KEY\_Value1转换为最终的键值KEY\_Value。同时，通过Value\_en信号标志键值是否有效。

##### **KeyPress模块**

该模块负责对单个按键输入信号进行消抖处理。它通过状态机和计数器实现了消抖功能，具体步骤如下：

****跳变沿检测****：通过对按键输入信号进行两级寄存器缓存，检测按键信号的上升沿和下降沿。

****计数使能与计数****：当检测到按键信号的跳变沿时，启动计数器开始计数。计数器的计数范围为 0 到 999999，当计数到 999999 时，认为消抖完成。

****状态机控制****：根据按键信号的变化和计数器的状态，状态机在Key\_up、Filter\_Up2Down、Key\_down和Filter\_Down2Up四个状态之间进行转换，最终输出稳定的按键状态KEY\_STATE和按键按下标志位KEY\_FLAG。

### **5.**模块程序流程图和仿真

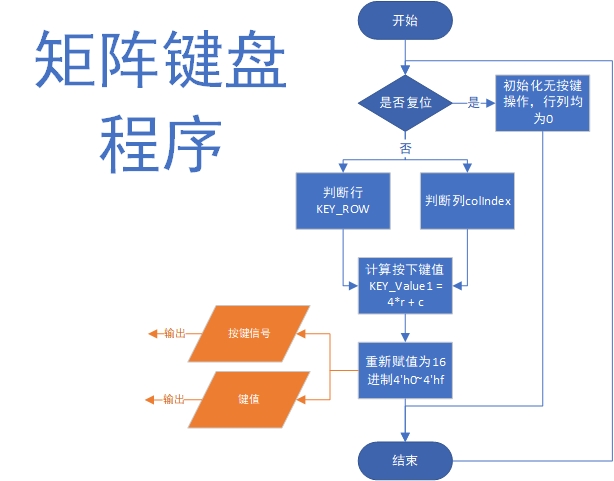


图3



图4

## **2.**密码锁程序

### 1.程序概述

密码锁程序是该系统最核心的处理程序，对矩阵键盘输入的按键值进行处理，控制数码管显示、LED 状态指示以及触发报警信号。系统具备解锁、锁定、修改密码、显示密码等功能，同时支持自动上锁和密码输入错误计数报警。

### 2.功能介绍

主状态机逻辑根据复位信号和按键输入信号进行状态转换和相应的操作处理，具体如下：

****复位操作****：当复位信号rst\_n有效时，系统进行初始化操作，包括清空数码管显示、重置输入计数、将状态设置为锁定状态、退出修改密码模式、清零错误次数和暂存密码等。

****无按键处理****

****自动上锁计时器控制****：在解锁状态下，空闲计时器idle\_timer开始计时。若无任何操作达到10秒（500000000个时钟周期），则触发自动上锁，清空显示和输入计数，退出修改密码模式。

****修改密码成功状态计时****：当密码修改成功标志password\_success置位时，启动计时器success\_timer，led3以250ms为周期开始闪烁1秒。计时结束后，清除password\_success标志。

****按键处理****：当按键有效信号key\_valid有效时，根据不同的按键值进行相应的操作。

****数字键（0 - 9）****：若输入计数小于8，则将按键值添加到数码管显示数据中，并更新输入计数。同时，设置未开锁倒计时标志count5\_\_begin为 1。

****A键（确认）：****在上锁状态下，比较输入的密码与存储的密码。若相同，则解锁成功，清空显示和输入计数，重置错误次数；若不同，则清空显示和输入计数，错误次数加 1，并显示错误次数。在修改密码模式下，若为第一次输入，则将输入的密码暂存到temp\_password中；若为第二次输入，比较两次输入的密码。若相同，则更新存储的密码，设置密码修改成功标志；无论是否成功，都退出修改密码模式。

****B键（锁定）****：将状态切换为锁定状态，清空显示和输入计数，退出修改密码模式。

****C键（清空）****：清空数码管显示和输入计数。

****D键（切换修改模式）**：**在解锁状态下，切换修改密码模式标志，清空显示和输入计数。

**\*/E键（键显示密码）：**在解锁状态下，显示存储的密码，并清空输入计数。

### 3.报警信号生成设计

stat\_count4：当密码锁处于锁定状态且错误次数大于等于1时，触发 LED4 报警信号。

start\_count5：当密码锁处于锁定状态且有按键输入时，触发 LED5 报警信号。

### **4.**程序流程图和仿真

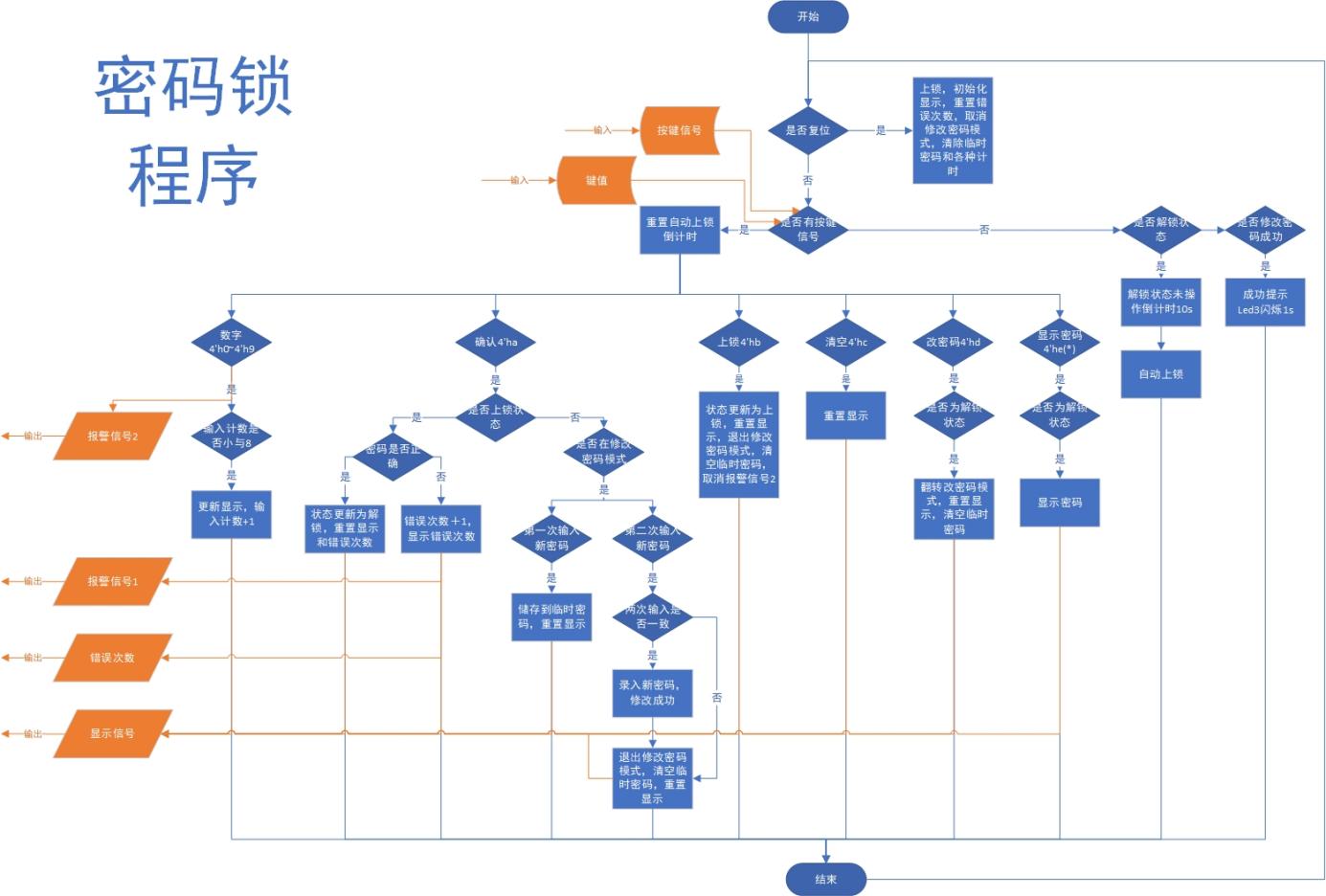


图5

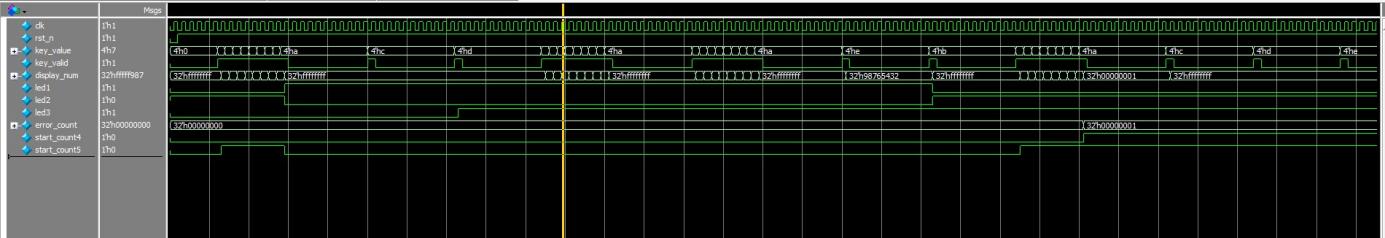


图6

## 数码管显示

### 1.数码管显示程序概述

数码管显示模块将输入的32位数据在数码管上进行动态显示。该模块接收系统时钟信号、复位信号以及要显示的32位数字，通过内部的逻辑处理，输出数码管的段选信号和位选信号，从而实现数字的显示。设计采用动态扫描的方式，利用人眼的视觉暂留效应，依次点亮各个数码管，以达到同时显示多位数字的效果。

****输入信号：****

clk：系统时钟信号。

rst\_n：复位信号，低电平有效。当该信号有效时，模块将进行初始化操作，使数码管显示全“-”。

number：要显示的32位数字，范围为0到99999999。该数字将被拆分为8个4位的数字，分别对应8个数码管的显示内容。

****输出信号：****

seg：段选信号，8位信号，用于控制数码管的a、b、c、d、e、f、g、dp段的点亮状态，从而显示不同的数字或字符。

sel：位选信号，3位信号，用于选择要点亮的数码管，通过38译码器可以控制8个数码管的依次显示。

### 2.时钟分频

为了实现动态扫描，需要一个较低频率的扫描时钟信号。模块内部使用一个8位的计数器div\_cnt对系统时钟进行分频。在每个时钟上升沿，div\_cnt计数器的值加 1。通过取div\_cnt的第4位作为扫描时钟信号scan\_clk，实现了对系统时钟的分频。

系统时钟频率f=50MHz，那么扫描时钟频率fscan=(50×10^6)/32=1.5625×10^6Hz=1.5625MHz。这个扫描频率决定了数码管的扫描速度，确保数码管显示稳定且不会出现闪烁现象。

### 3.扫描计数器与位选信号控制

使用一个3位的扫描计数器scan\_cnt来控制数码管的扫描顺序。在扫描时钟scan\_clk的上升沿，scan\_cnt的值加1。位选信号sel根据scan\_cnt的值进行变化，初始值为3'd1，随着scan\_cnt的增加，sel的值依次递增，从而依次选择8个数码管进行显示。

### 4.数据转换与显示数据存储

在每个系统时钟的上升沿，模块会根据复位信号和输入的数字number对显示数据进行处理。当复位信号rst\_n有效时，将8个数码管的显示数据都设置为 4'hf，即显示 “-”。当输入的数字number为 32'hFFFFFFFF 时，同样将所有显示数据设置为4'hf。对于其他情况，将number按每4位一组进行拆分，分别存储到display\_data数组的 8 个元素中，对应8个数码管的显示内容。

### 5.段选信号生成

根据当前扫描到的数码管对应的显示数据 display\_data[scan\_cnt]，通过一个case语句生成相应的段选信号seg。对于0-9的数字，分别定义了对应的段选信号编码，用于在数码管上显示相应的数字。当显示数据为4'hf时，显示“-”。对于其他未定义的值，段选信号设置为8'b00000000，即不显示。

### 6.程序流程图和仿真

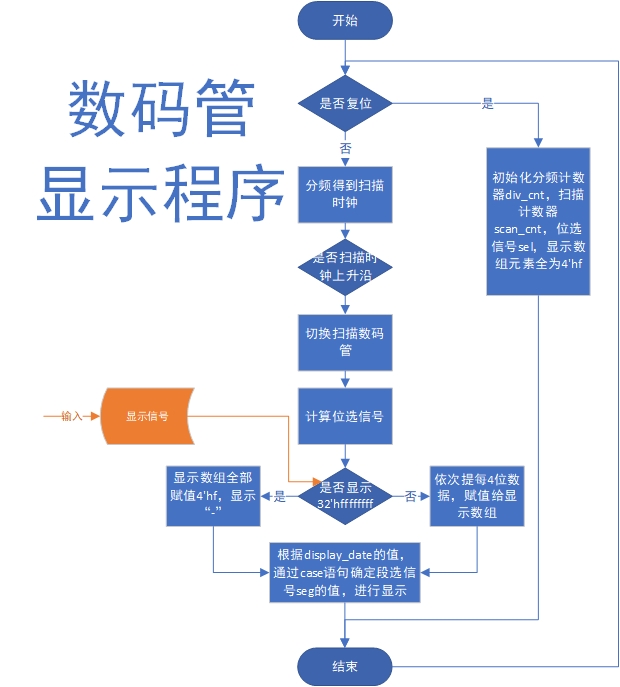


图7

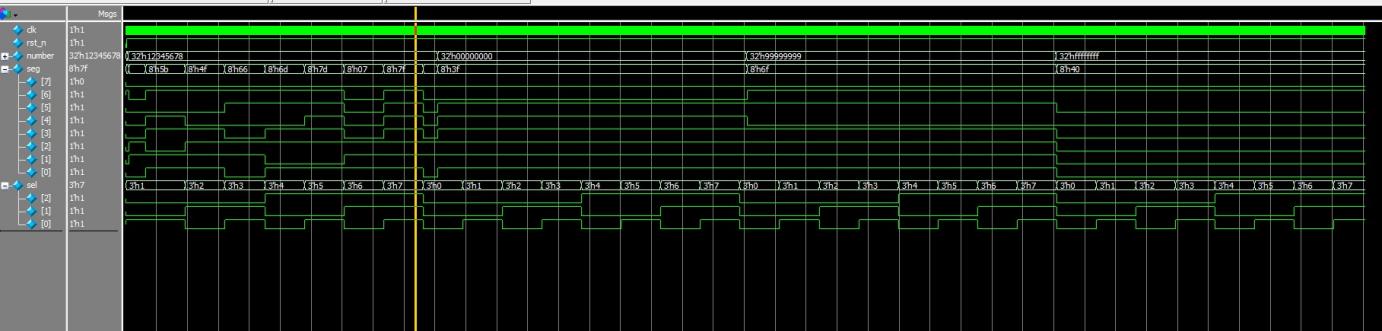


图8

## 4.报警

### 1.程序概述

报警模块是依据密码锁的操作状态驱动相应的报警指示灯。该模块接收系统时钟信号、复位信号、错误次数计数以及两个报警触发信号，通过内部的逻辑处理，控制两个报警指示灯LED4和LED5的亮灭状态，实现不同形式的报警提示。

****输入信号：****

clk：系统时钟信号频率为50MHz。

rst\_n：复位信号，低电平有效。当该信号有效时，报警模块将进行初始化，所有报警指示灯熄灭，计数器清零。

error\_count：错误次数计数信号。

start\_count4：触发LED4报警的信号。

start\_count5：触发 LED5 报警的信号。

****输出信号：****

led4：LED4报警提示。

led5：LED5报警提示。

### 2.计时参数定义

为了实现精确的计时和不同时长的报警效果，模块内部定义了多个计时参数：

**COUNT\_1S：**表示1秒的计时周期，根据系统时钟频率50MHz计算得出，值为 50000000 - 1，即50MHz 时钟下1秒内的时钟周期数。

**COUNT\_5S：**表示5秒的计时周期，5 \* (COUNT\_1S + 1) - 1。

**COUNT\_30S：**表示30秒的计时周期，3 \*(COUNT\_1S + 1) - 1。

**FLASH\_PERIOD：**表示闪烁周期，值为 (COUNT\_1S + 1) / 8 - 1，对应125ms的时长，用于控制报警指示灯的闪烁频率，以实现闪烁报警效果。

### 3. LED4报警功能介绍

当接收到报警信号4时，判断当前错误次数，前2次错误报警5s（闪烁5s,下同），第3次错误报警30s，超过3次错误常报警。当报警信号4置低时报警停止。

### 4. LED5报警逻辑设计

当接收到报警信号5时，开启30s计时，结束后开始30s报警，如此循环。当报警信号5置低时报警停止。

### 5.程序流程图和仿真

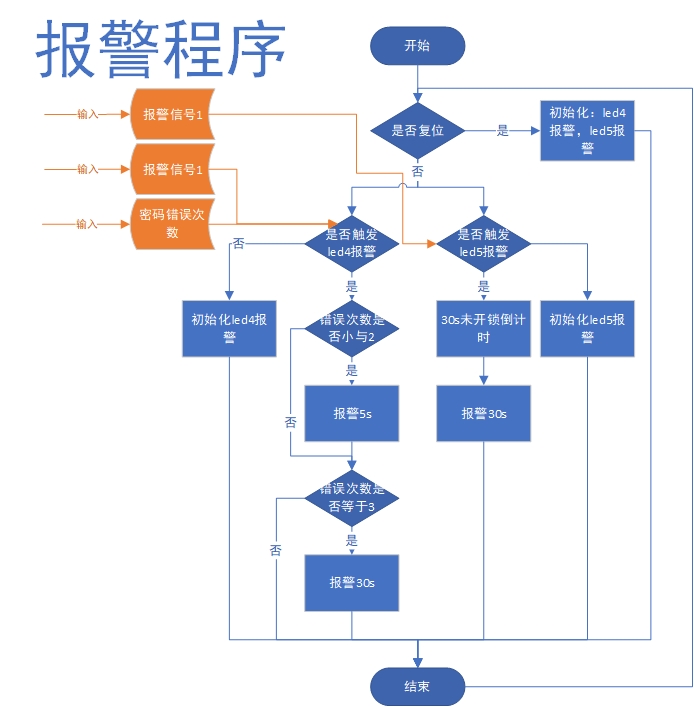


图9

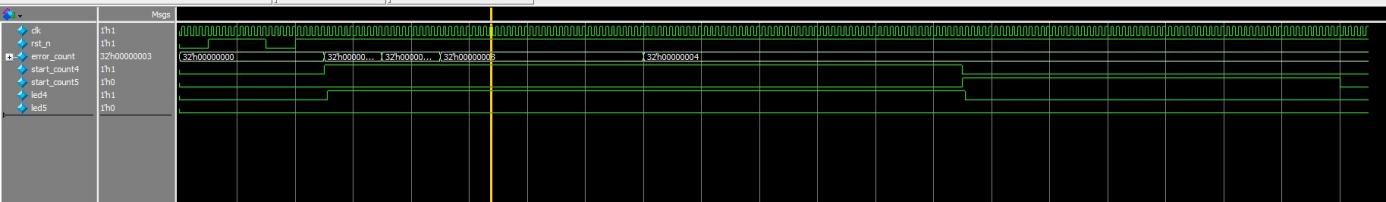


图10

## 5.实测照片和编译结果

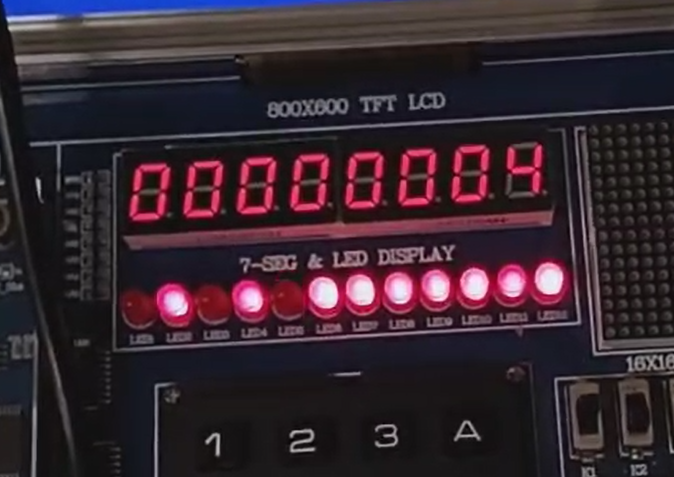


图11

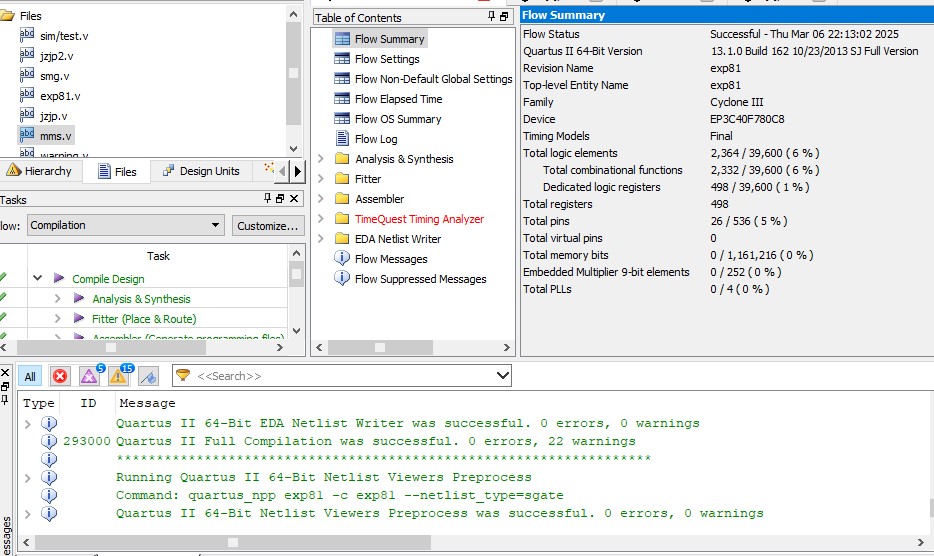


图12

# 四、困难

## 1.数码管亮度低

在编写数码管程序时发现，数码管静态显示相同的数字亮度是正常的，但是动态扫描显示，亮度就会明显变暗。网上查找资料，主要有2个解决方法，第一个方法是调整限流电阻，增加电流以提高驱动能力，增强亮度，但是实验箱是固定的，没法更换电阻，也没有找到调节亮度的旋钮或者其他装置，所以这个方法不可行。第二个方法是增加单个数码管的显示时间，看起来就会变亮，但是也不能时间太长，否则就会有单个比较亮其他比较暗，闪烁的感觉。但是我实测第二种方法效果并不明显，更换了几个实验箱都试试，有的亮一些有的暗一些，应该也和实验箱有关。

## 2.矩阵键盘消抖

借助数码管程序来检测矩阵键盘，发现按下哪个按键就会直接出现一大串这个键值，我才发现没有消抖。但是矩阵键盘行列扫描，消抖没有普通的单个按键消抖那么简单，尝试着写消抖程序，却又很不好用，有时候按键按下去松开居然会检测不到，有时候还会检测到按了2次，不好用。于是就去csdn上找verilog矩阵键盘代码，发现有个大神写的挺好用，分了2个模块，一个就是矩阵键盘键值判断，另一个专门用来消抖，实测非常好用，就借过来了。

## 3.各种复杂的计时

做第3个要求报警程序的时候，由于之前单片机使用习惯了，先想到的就是延时函数delay，可是delay的时候，其他程序能正常执行吗？如果我去执行其他程序，那这个实际计时时间是不是就延长了？反正挺迷糊的。想了个办法，延时实际上是计数，只不过函数里只有计数，我把计数放在主程序相应的地方，应该就可以实现计时和其他程序的执行互不干扰了。问题又来了，仿真发现实际时间总是多出来20ns，刚好多出来1个周期，一般都是从0开始计数，假如计100个周期，总时间应该是2000ns，但是计数截止应该是99，计到100是101个周期，需要减1，才是正确时间。

## 4.调试仿真

个人习惯来说，喜欢编译0错误就直接上机验证，但是编译只能检验逻辑错误，检查不了功能错误，实际上对了没有，只是我以为他对了，上机验证他不对，那些寄存器数据，怎么变化的，数码管就那么几位也无法完全显示，找bug就挺痛苦的，又拐回去看程序，到底哪里出问题了？烦死了。网上找的野火FPGA教程，发现几乎每一章都是先仿真后上机验证，跟着学吧，发现仿真真好用，特别是找bug比较方便，直接看某个数据什么时候什么状态，直接上机验证是看不见这些东西的，只能看宏观的现象。

## 5.模块之间的调用

这纯粹就是verilog代码以及fpga开发流程不熟悉，遇到的困难就是，比如密码锁程序，要调用数码管显示，直接在密码锁程序里面实例化，能调用，但是导致问题是，只能在这个模块里面用，后来发现，把所有模块都在顶层模块实例化，这样模块之间就可以相互调用了，不需要那么多实例化，只需要写明输入输出就可以。

# 五、课程设计收获

## 1.掌握FPGA开发基本流程

通过本次课程设计，对FPGA有了更为深入的理解。认识了逻辑单元、存储单元和 I/O 引脚资源分配。深入研究了 FPGA 的时钟资源和时序约束机制，掌握了通过合理设置时钟分频器来满足不同功能模块对时钟频率的要求。学会了利用异步复位来解决时序冲突问题，确保系统中各模块能够在统一的时钟架构下稳定运行。同时提升了对Verilog的运用能力。

## 2.复杂问题解决能力提升

在设计过程中遭遇了诸多复杂问题，如数码管亮度暗、矩阵键盘消抖困难、复杂计时任务实现以及模块间调用协同等问题。针对数码管亮度暗的问题，通过研究电路原理，在处理矩阵键盘消抖时，通过从网上寻求帮助，成功实现了稳定的按键检测。面对复杂的计时任务，深入学习了FPGA的时钟管理单元，合理分配时钟资源，解决了不同计时任务之间的干扰问题，锻炼了在复杂系统中协调多种任务的能力。在模块间调用方面，通过精心设计接口规范和同步机制，确保了各模块之间数据的正确传输和协同工作，提高了系统集成和调试能力。这些经历使得在面对复杂工程问题时，能够更加有条不紊地进行分析、尝试解决方案并最终实现问题的解决。

## 3.培养工程思维

整个设计过程遵循了从任务分析、思路构建、具体设计方案实施到调试优化的工程流程，培养了严谨的工程思维。在任务分析阶段，能够准确梳理出密码锁设计所需的各项功能和性能指标，为后续设计奠定基础。在思路构建过程中，学会了从整体架构出发，将系统分解为多个功能模块，并考虑各模块之间的相互关系和接口设计。在具体设计实施时，注重代码的规范性和可读性，采用模块化编程方法，使代码结构清晰，易于维护和扩展。在调试优化阶段，通过合理运用开发工具的调试功能，如波形仿真、逻辑分析等，快速定位和解决问题。这种工程思维和方法的养成，对于今后从事其他工程项目具有重要的指导意义，能够更加高效、规范地完成复杂的工程设计任务。

# 六、参考文献

(1)谭会生、瞿遂春，《EDA技术综合应用实例与分析》，西安电子科技大学出版社，2004

(2)高有堂，《EDA技术及应用实践》，清华大学出版社，2006

(3)亿特科技，CPLD/FPGA应用系统设计与产品开发》，人民邮电出版社，2005

(4)曹昕燕、周凤臣等，《EDA技术实验与课程设计》，清华大学出版社，2006

# 七、程序代码：

module exp81(

input wire clk, // 外部时钟输入

input wire rst\_n, // 外部复位信号输入

input wire [3:0] key\_in, // 外部按键输入

output wire [3:0] key\_out, // 输出到外部的按键信号

output wire [7:0] seg, // 输出到数码管的段选信号

output wire [2:0] sel, // 输出到数码管的位选信号

output wire led1, // 输出到 LED 的信号

output wire led2,

output wire led3,

output wire led4,

output wire led5

);

// 中间信号

wire [3:0] key\_value;

wire key\_valid;

wire [31:0] display\_num;

wire [31:0] error\_count;

wire start\_count4, start\_count5;

// 实例化按键扫描模块

KeyValue key\_inst(

.CLK(clk),

.nRST(rst\_n),

.KEY\_COL(key\_in), // 按键列

.KEY\_ROW(key\_out), // 按键行

.KEY\_Value(key\_value), // 按键值

.Value\_en(key\_valid) // 按键有效信号

);

// 实例化数码管显示模块

smg\_display smg\_inst(

.clk(clk),

.rst\_n(rst\_n),

.number(display\_num),

.seg(seg),

.sel(sel)

);

// 实例化密码锁逻辑模块

mm\_lock mm\_lock\_inst (

.clk(clk),

.rst\_n(rst\_n),

.key\_value(key\_value),

.key\_valid(key\_valid),

.display\_num(display\_num),

.led1(led1),

.led2(led2),

.led3(led3),

.error\_count(error\_count),

.start\_count4(start\_count4),

.start\_count5(start\_count5)

);

// 实例化报警控制模块

warning warning\_inst(

.clk(clk),

.rst\_n(rst\_n),

.error\_count(error\_count),

.start\_count4(start\_count4),

.start\_count5(start\_count5),

.led4(led4),

.led5(led5)

);

endmodule

module mm\_lock(

input wire clk, // 系统时钟 (50MHz)

input wire rst\_n, // 复位信号 (低有效)

input wire [3:0] key\_value, // 按键值 (0-F)

input wire key\_valid, // 按键有效信号

output reg [31:0] display\_num, // 数码管显示数据

output wire led1, // 状态LED1 (解锁亮)

output wire led2, // 状态LED2 (锁定亮)

output reg led3, // 状态LED3 (修改模式/闪烁)

output reg [31:0] error\_count, // 错误次数计数

output wire start\_count4, // 触发LED4报警

output wire start\_count5 // 触发LED5报警

);

// 状态定义

reg [3:0] input\_count = 0; // 当前输入计数

reg current\_state; // 当前状态

localparam LOCKED = 1'b0; // 锁定状态

localparam UNLOCKED = 1'b1; // 解锁状态

reg count5\_\_begin = 0; // 开锁倒计时，开锁重置

// 密码存储

reg [31:0] password = 32'h12345678; // 存储的密码

// 密码修改相关信号

reg change\_password\_mode; // 修改模式标志

reg [31:0] temp\_password; // 第一次输入暂存

reg password\_success; // 修改成功标志

// 闪烁控制

reg [24:0] flash\_counter; // 125ms闪烁计数器 (50MHz时钟下6,250,000 cycles)

wire led3\_flash; // 闪烁信号

// 1秒计时器

reg [26:0] success\_timer; // 1秒计时 (50,000,000 cycles)

// 自动上锁信号声明

reg [28:0] idle\_timer; // 10秒空闲计时器 (50MHz下500,000,000 cycles)

wire idle\_timeout; // 超时信号

localparam IDLE\_TIMEOUT = 500\_000\_000 - 1; // 实际值需按时钟频率计算

// 主状态机逻辑

always @(posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

// 复位初始化

display\_num <= 32'hFFFFFFFF; // 清空显示

input\_count <= 0; // 清空输入计数

current\_state <= LOCKED; // 初始化状态为锁定

change\_password\_mode <= 0; // 默认不在修改密码模式

error\_count <= 0; // 初始化错误次数

temp\_password <= 0;

password\_success <= 0;

success\_timer <= 0;

idle\_timer <= 0;

end else begin

// 无按键处理

if(!key\_valid) begin

// 自动上锁计时器控制

if(current\_state == UNLOCKED) begin

idle\_timer <= (idle\_timer >= IDLE\_TIMEOUT) ? 0 : idle\_timer + 1;

end else begin

idle\_timer <= 0; // 非解锁状态清零

end

// 自动上锁触发

if(idle\_timeout) begin

current\_state = LOCKED; // 锁定

display\_num = 32'hFFFFFFFF; // 清空显示

input\_count = 0; // 清空输入计数

change\_password\_mode = 0; // 退出修改密码模式

temp\_password = 0;

count5\_\_begin = 0;

end

// 修改密码成功状态计时

if(password\_success) begin

if(success\_timer < 27'd50\_000\_000) begin

success\_timer <= success\_timer + 1;

end else begin

password\_success <= 0;

success\_timer <= 0;

end

end

end

// 按键处理

if(key\_valid) begin

idle\_timer <= 0; // 重置计时器

case(key\_value)

4'h0, 4'h1, 4'h2, 4'h3, 4'h4, 4'h5, 4'h6, 4'h7, 4'h8, 4'h9: begin

count5\_\_begin <= 1;

if(input\_count < 8) begin

display\_num <= {display\_num[27:0], key\_value};

input\_count <= input\_count + 1; // 更新显示

end

end

4'hA: begin // A键确认输入

if(current\_state == LOCKED) begin

if(display\_num == password) begin

current\_state <= UNLOCKED; // 解锁成功

display\_num <= 32'hFFFFFFFF; // 清空显示

error\_count <= 0; // 重置错误次数

end else begin // 密码输入错误

input\_count = 0; // 清空输入计数

display\_num = 32'hFFFFFFFF; // 清空显示

error\_count = error\_count + 1; // 错误次数+1

display\_num = error\_count; // 显示错误次数

end

end else if(change\_password\_mode) begin // 修改密码流程

if(temp\_password == 0) begin // 第一次输入

temp\_password = display\_num;

display\_num = 32'hFFFFFFFF;

input\_count = 0;

end else begin // 第二次输入

if(display\_num == temp\_password) begin

password <= display\_num;

password\_success <= 1; // 修改成功闪烁

end

// 无论是否成功都退出修改模式

change\_password\_mode <= 0;

temp\_password <= 0;

display\_num <= 32'hFFFFFFFF;

input\_count <= 0;

end

end

end

4'hB: begin // B键锁定

current\_state <= LOCKED; // 锁定

display\_num <= 32'hFFFFFFFF; // 清空显示

input\_count <= 0; // 清空输入计数

change\_password\_mode <= 0; // 退出修改密码模式

temp\_password <= 0;

count5\_\_begin <= 0;

end

4'hC: begin // C键清空

display\_num <= 32'hFFFFFFFF; // 清空显示

input\_count <= 0; // 清空输入计数

end

4'hD: begin // D键切换修改模式

if(current\_state == UNLOCKED) begin

change\_password\_mode <= ~change\_password\_mode;

input\_count <= 0; // 清空输入计数

display\_num <= 32'hFFFFFFFF; // 清空显示

temp\_password <= 0;

end

end

4'hE: begin // \*键显示密码

if(current\_state == UNLOCKED) begin

input\_count <= 0; // 清空输入计数

display\_num <= password; // 显示密码

end

end

default: ; // 其他按键无操作

endcase

end

end

end

// 闪烁信号生成

always @(posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

flash\_counter <= 0;

end else begin

flash\_counter <= (flash\_counter == 25'd6\_250\_000) ? 0 : flash\_counter + 1;

end

end

assign led3\_flash = (flash\_counter < 25'd3\_125\_000); // 125ms周期

// LED3输出控制

always @(\*) begin

if(password\_success) begin

led3 = led3\_flash; // 成功时闪烁

end else begin

led3 = change\_password\_mode; // 修改模式常亮

end

end

// 触发信号生成

assign start\_count4 = (current\_state == LOCKED) && (error\_count >= 1);

assign start\_count5 = (current\_state == LOCKED) && (count5\_\_begin > 0);

// LED状态指示

assign led1 = (current\_state == UNLOCKED); // 解锁时LED1亮

assign led2 = (current\_state == LOCKED); // 锁定时LED2亮

// 超时信号生成

assign idle\_timeout = (idle\_timer == IDLE\_TIMEOUT);

endmodule

module smg\_display(

input wire clk, // ϵͳʱ��

input wire rst\_n, // ��λ�źţ��͵�ƽ��Ч

input wire [31:0] number, // Ҫ��ʾ�����֣���Χ0~99999999

output reg [7:0] seg, // ��ѡ�ź� (a,b,c,d,e,f,g,dp)

output reg [2:0] sel // λѡ�źţ�����3-8������

);

// ��Ƶ������

reg [7:0] div\_cnt;

wire scan\_clk;

// ��ʾ���ݴ洢

reg [3:0] display\_data [7:0]; // 8���������ʾ������

reg [2:0] scan\_cnt; // ɨ�������

// ��Ƶ����ɨ��ʱ��

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n)

div\_cnt <= 8'd0;

else

div\_cnt <= div\_cnt + 8'd1;

end

assign scan\_clk = div\_cnt[4]; // ɨ��ʱ�ӷ�Ƶ

// ɨ�����������

always @(posedge scan\_clk or negedge rst\_n) begin

if (!rst\_n)

scan\_cnt <= 3'd0;

else

scan\_cnt <= scan\_cnt + 3'd1;

end

// λѡ�ź�����

always @(posedge scan\_clk or negedge rst\_n) begin

if (!rst\_n)

sel <= 3'd1;

else

sel <= 3'd1 + scan\_cnt;

end

// ����ת����ʾ����

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

// ��ʼ����ʾȫ��"-"

display\_data[0] <= 4'hf;

display\_data[1] <= 4'hf;

display\_data[2] <= 4'hf;

display\_data[3] <= 4'hf;

display\_data[4] <= 4'hf;

display\_data[5] <= 4'hf;

display\_data[6] <= 4'hf;

display\_data[7] <= 4'hf;

end else begin

if (number == 32'hFFFFFFFF) begin

// ��ʾȫ��"-"

display\_data[0] <= 4'hf;

display\_data[1] <= 4'hf;

display\_data[2] <= 4'hf;

display\_data[3] <= 4'hf;

display\_data[4] <= 4'hf;

display\_data[5] <= 4'hf;

display\_data[6] <= 4'hf;

display\_data[7] <= 4'hf;

end else begin

// ֱ����ȡÿ��4λ��

display\_data[0] <= number[31:28];

display\_data[1] <= number[27:24];

display\_data[2] <= number[23:20];

display\_data[3] <= number[19:16];

display\_data[4] <= number[15:12];

display\_data[5] <= number[11:8];

display\_data[6] <= number[7:4];

display\_data[7] <= number[3:0];

end

end

end

// �߶����������

always @(\*) begin

case(display\_data[scan\_cnt])

4'h0: seg = 8'b00111111; // 0

4'h1: seg = 8'b00000110; // 1

4'h2: seg = 8'b01011011; // 2

4'h3: seg = 8'b01001111; // 3

4'h4: seg = 8'b01100110; // 4

4'h5: seg = 8'b01101101; // 5

4'h6: seg = 8'b01111101; // 6

4'h7: seg = 8'b00000111; // 7

4'h8: seg = 8'b01111111; // 8

4'h9: seg = 8'b01101111; // 9

4'hf: seg = 8'b01000000; // "-"

default: seg = 8'b00000000;

endcase

end

endmodule

// warning.v

module warning(

input wire clk,

input wire rst\_n,

input wire [31:0] error\_count, // 闁挎瑨顕ゅ▎鈩冩殶

input wire start\_count4, // 鐟欙箑褰侺ED4閹躲儴顒

input wire start\_count5, // 鐟欙箑褰侺ED5閹躲儴顒

output reg led4, // LED4鏉堟挸鍤

output reg led5 // LED5鏉堟挸鍤

);

// 閸欏倹鏆熺€规矮绠熼敍鍫熺壌閹诡喖鐤勯梽鍛闁界喖顣堕悳鍥殶閺佽揪绱

parameter COUNT\_1S = 50\_000\_000 - 1; // 50MHz閺冨爼鎸撴稉缁夋帟顓搁弫鏉库偓

parameter COUNT\_5S = 5 \* (COUNT\_1S + 1) - 1;

parameter COUNT\_30S = 30 \* (COUNT\_1S + 1) - 1;

parameter FLASH\_PERIOD = (COUNT\_1S + 1) / 8 -1; // 125ms闂傤亞鍎婇崨銊︽埂

// LED4閹貉冨煑闁槒绶

reg [31:0] counter4;

reg [31:0] prev\_error\_count; // 新增：记录上一次错误次数

reg counter4\_reset\_flag; // 新增：计数器重置标志

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

counter4 <= 0;

led4 <= 0;

prev\_error\_count <= 0;

counter4\_reset\_flag <= 0;

end else begin

// 检测error\_count变化

if (error\_count != prev\_error\_count) begin

counter4 <= 0; // 错误次数变化时重置计数器

prev\_error\_count <= error\_count;

counter4\_reset\_flag <= 1; // 设置重置标志

end else begin

counter4\_reset\_flag <= 0;

end

if (start\_count4) begin

if (counter4\_reset\_flag) begin

counter4 <= 0; // 确保新周期从0开始

end

case (error\_count)

32'h1, 32'h2: begin // 合并1-2次错误逻辑

if (counter4 < COUNT\_5S) begin

counter4 <= counter4 + 1;

led4 <= (counter4 % FLASH\_PERIOD < FLASH\_PERIOD/2);

end else begin

led4 <= 0;

end

end

32'h3: begin // 3次错误

if (counter4 < COUNT\_30S) begin

counter4 <= counter4 + 1;

led4 <= (counter4 % FLASH\_PERIOD < FLASH\_PERIOD/2);

end else begin

led4 <= 0;

end

end

default: begin // 3次以上错误

if (error\_count > 32'h3) begin

led4 <= 1; // 常亮报警

counter4 <= 0;

end

end

endcase

end else begin

counter4 <= 0;

led4 <= 0;

end

end

end

// LED5閹貉冨煑闁槒绶

reg [31:0] counter5;

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

counter5 <= 0;

led5 <= 0;

end else if (start\_count5) begin // 鐡掑懏妞傞張顏呮惙娴ｆ粍濮ょ拃

counter5 <= counter5 + 1;

if (counter5 < COUNT\_30S) begin

led5 <= 0;

end else if (counter5 >= COUNT\_30S && counter5 < 2 \* COUNT\_30S) begin

led5 <= (counter5 % FLASH\_PERIOD < FLASH\_PERIOD/2);

end else begin

counter5 <= 0;

led5 <= 0;

end

end else begin

counter5 <= 0;

led5 <= 0;

end

end

endmodule

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//���ܣ�������̼��

//

//

//����:Ray

//ʱ��:2021-4-24

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module KeyValue(

CLK,

nRST,

KEY\_ROW,

KEY\_COL,

KEY\_Value,

KEY\_Value1,

Value\_en

);

input CLK;

input nRST;

input [3:0]KEY\_COL; //��

output reg Value\_en;

output reg [3:0]KEY\_ROW; //��

output reg [3:0]KEY\_Value; //������������ֵ

output reg [3:0]KEY\_Value1; //������̰��µ�ֵ

wire [3:0]key\_flag; //������־λ

wire [3:0]key\_state;

reg [4:0]state;

reg row\_flag; //��ʶ�Ѷ�λ����

reg [1:0]rowIndex; //������

reg [1:0]colIndex; //������

localparam

NO\_KEY = 5'b00001,

ROW\_ONE = 5'b00010,

ROW\_TWO = 5'b00100,

ROW\_THREE = 5'b01000,

ROW\_FOUR = 5'b10000;

KeyPress u0(

.CLK(CLK),

.nRST(nRST),

.KEY\_IN(KEY\_COL[0]),

.KEY\_FLAG(key\_flag[0]),

.KEY\_STATE(key\_state[0])

);

KeyPress u1(

.CLK(CLK),

.nRST(nRST),

.KEY\_IN(KEY\_COL[1]),

.KEY\_FLAG(key\_flag[1]),

.KEY\_STATE(key\_state[1])

);

KeyPress u2(

.CLK(CLK),

.nRST(nRST),

.KEY\_IN(KEY\_COL[2]),

.KEY\_FLAG(key\_flag[2]),

.KEY\_STATE(key\_state[2])

);

KeyPress u3(

.CLK(CLK),

.nRST(nRST),

.KEY\_IN(KEY\_COL[3]),

.KEY\_FLAG(key\_flag[3]),

.KEY\_STATE(key\_state[3])

);

//==========ͨ��״̬���ж���===========//

always @(posedge CLK or negedge nRST)

if(!nRST)

begin

state <= NO\_KEY;

row\_flag <= 1'b0;

KEY\_ROW <= 4'b0000;

end

else

case(state)

NO\_KEY: begin

row\_flag <= 1'b0;

KEY\_ROW <= 4'b0000;

if(key\_flag != 4'b0000) begin

state <= ROW\_ONE;

KEY\_ROW <= 4'b1110;

end

else

state <= NO\_KEY;

end

ROW\_ONE: begin

//�������ж�ֻ����KEY\_COL��������key\_state

//��Ϊ��������ģ��ʹ��key\_state���ȶ�

//������ΪKEY\_ROW�Ķ��ڱ仯���仯

//��KEY\_COL������KEY\_ROWʵʱ�仯

if(KEY\_COL != 4'b1111) begin

state <= NO\_KEY;

rowIndex <= 4'd0;

row\_flag <= 1'b1;

end

else begin

state <= ROW\_TWO;

KEY\_ROW <= 4'b1101;

end

end

ROW\_TWO: begin

if(KEY\_COL != 4'b1111) begin

state <= NO\_KEY;

rowIndex <= 4'd1;

row\_flag <= 1'b1;

end

else begin

state <= ROW\_THREE;

KEY\_ROW <= 4'b1011;

end

end

ROW\_THREE: begin

if(KEY\_COL != 4'b1111) begin

state <= NO\_KEY;

rowIndex <= 4'd2;

row\_flag <= 1'b1;

end

else begin

state <= ROW\_FOUR;

KEY\_ROW <= 4'b0111;

end

end

ROW\_FOUR: begin

if(KEY\_COL != 4'b1111) begin

rowIndex <= 4'd3;

row\_flag <= 1'b1;

end

state <= NO\_KEY;

end

endcase

//===========�жϰ���������=============//

always @(posedge CLK or negedge nRST)

if(!nRST)

colIndex <= 2'd0;

else if(key\_state != 4'b1111)

case(key\_state)

4'b1110: colIndex <= 2'd0;

4'b1101: colIndex <= 2'd1;

4'b1011: colIndex <= 2'd2;

4'b0111: colIndex <= 2'd3;

endcase

//===========ͨ�����м����ֵ==========//

always @(posedge CLK or negedge nRST)

if(!nRST)

Value\_en <= 1'b0;

else if(row\_flag)

begin

Value\_en = 1'b1;

KEY\_Value1 = 4\*rowIndex + colIndex;

if(KEY\_Value1 == 4'd0)

begin

KEY\_Value <= 4'h1;

end

else if(KEY\_Value1 == 4'd1)

begin

KEY\_Value <= 4'h2;

end

else if(KEY\_Value1 == 4'd2)

begin

KEY\_Value <= 4'h3;

end

else if(KEY\_Value1 == 4'd3)

begin

KEY\_Value <= 4'ha;

end

else if(KEY\_Value1 == 4'd4)

begin

KEY\_Value <= 4'h4;

end

else if(KEY\_Value1 == 4'd5)

begin

KEY\_Value <= 4'h5;

end

else if(KEY\_Value1 == 4'd6)

begin

KEY\_Value <= 4'h6;

end

else if(KEY\_Value1 == 4'd7)

begin

KEY\_Value <= 4'hb;

end

else if(KEY\_Value1 == 4'd8)

begin

KEY\_Value <= 4'h7;

end

else if(KEY\_Value1 == 4'd9)

begin

KEY\_Value <= 4'h8;

end

else if(KEY\_Value1 == 4'd10)

begin

KEY\_Value <= 4'h9;

end

else if(KEY\_Value1 == 4'd11)

begin

KEY\_Value <= 4'hc;

end

else if(KEY\_Value1 == 4'd12)

begin

KEY\_Value <= 4'he;

end

else if(KEY\_Value1 == 4'd13)

begin

KEY\_Value <= 4'h0;

end

else if(KEY\_Value1 == 4'd14)

begin

KEY\_Value <= 4'hf;

end

else if(KEY\_Value1 == 4'd15)

begin

KEY\_Value <= 4'hd;

end

end

else

Value\_en <= 1'b0;

endmodule

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//功能：定义状态机,方便后期用于矩阵键盘消抖

//

//

//作者:Ray

//时间:2021-4-24

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module KeyPress(

CLK,

nRST,

KEY\_IN,

KEY\_FLAG,

KEY\_STATE

);

input CLK;

input nRST;

input KEY\_IN;

output reg KEY\_FLAG; //按键按下标志位 高电平为按下

output reg KEY\_STATE;

reg key\_a, key\_b;

reg en\_cnt, cnt\_full;

reg [3:0]state;

reg [19:0]cnt;

wire flag\_H2L, flag\_L2H;

//运用状态机对矩阵键盘进行检测

//定义状态

localparam

Key\_up = 4'b0001,

Filter\_Up2Down = 4'b0010,

Key\_down = 4'b0100,

Filter\_Down2Up = 4'b1000;

//======判断按键输入信号跳变沿========//

always @(posedge CLK or negedge nRST)

if(!nRST)

begin

key\_a <= 1'b0;

key\_b <= 1'b0;

end

else

begin

key\_a <= KEY\_IN;

key\_b <= key\_a;

end

assign flag\_H2L = key\_b && (!key\_a);

assign flag\_L2H = (!key\_b) && key\_a;

//============计数使能模块==========//

always @(posedge CLK or negedge nRST)

if(!nRST)

cnt <= 1'b0;

else if(en\_cnt)

cnt <= cnt + 1'b1;

else

cnt <= 1'b0;

//=============计数模块=============//

always @(posedge CLK or negedge nRST)

if(!nRST)

cnt\_full <= 1'b0;

else if(cnt == 20'd999\_999)

cnt\_full <= 1'b1;

else

cnt\_full <= 1'b0;

//=============有限状态机============//

always @(posedge CLK or negedge nRST)

if(!nRST)

begin

en\_cnt <= 1'b0;

state <= Key\_up;

KEY\_FLAG <= 1'b0;

KEY\_STATE <= 1'b1;

end

else

case(state)

//保持没按

Key\_up: begin

KEY\_FLAG <= 1'b0;

if(flag\_H2L) begin

state <= Filter\_Up2Down;

en\_cnt <= 1'b1;

end

else

state <= Key\_up;

end

//正在向下按

Filter\_Up2Down: begin

if(cnt\_full) begin

en\_cnt <= 1'b0;

state <= Key\_down;

KEY\_STATE <= 1'b0;

KEY\_FLAG <= 1'b1;

end

else if(flag\_L2H) begin

en\_cnt <= 1'b0;

state <= Key\_up;

end

else

state <= Filter\_Up2Down;

end

//保持按下状态

Key\_down: begin

KEY\_FLAG <= 1'b0;

if(flag\_L2H) begin

state <= Filter\_Down2Up;

en\_cnt <= 1'b1;

end

else

state <= Key\_down;

end

//正在释放按键

Filter\_Down2Up: begin

if(cnt\_full) begin

en\_cnt <= 1'b0;

state <= Key\_up;

KEY\_FLAG <= 1'b1;

KEY\_STATE <= 1'b1;

end

else if(flag\_H2L) begin

en\_cnt <= 1'b0;

state <= Key\_down;

end

else

state <= Filter\_Down2Up;

end

//其他未定义状态

default: begin

en\_cnt <= 1'b0;

state <= Key\_up;

KEY\_FLAG <= 1'b0;

KEY\_STATE <= 1'b1;

end

endcase

endmodule