信息与机电工程学院

《单片机系统设计》课程设计报告

课题名称	基于 IAP15W4K58S4 的打地鼠游戏机设计
专业班级	2022 级电子信息工程本科 1 班
学　　号	
学生姓名	
指导教师	
评　　分	

2024 年 06 月 20 日至 2022 年 07 月 11 日

答辩记录

1、例举设计过程中遇到的问题及其解决方法（至少两例）。

答：　(1)问题说明：我们在使用 PWM 驱动蜂鸣器的时候，发现开发板引出的接口只有 P0、P1、P2、P3 这四组引脚，其中 P0、P1 被数码管占用，P2 被矩阵键盘占用。只剩下了 P3 接口，在 P3 中唯一支持 PWM 的引脚是 P3.7，但是这个引脚被程序下载串口占用了!!!看起来这个方案陷入僵局，无法使用硬件 PWM 驱动蜂鸣器。

解决方法：我们发现开发板只引出了 32 个引脚，但是 IC 测试座上的 DIP40 封装，一共有 40 个引脚。还剩下 8 根引脚没有被开发板使用。我们在这个位置找到了 P4.4 用来输出 PWM 信号驱动蜂鸣器。

(2)问题说明：我们在扫描矩阵键盘的时候，为了减少单片机性能浪费，同时也为了消除按键抖动产生的干扰信号。设置矩阵键盘的列扫描与行扫描的间隔时间为 10ms。每执行一次列与行的扫描之后，也就是每 20ms 扫描出一次键值。但我们发现，在这种情况下，即使按下同一个键，有时能读出正确的键值，有时只能读取出错误的键值。令人百思不得奇解。

解决方法：我们直接将扫描出来的键值显示在数码管上。发现错误的键值都有一个特征：列扫描得到的值代表没有按键按下，而行扫描得到的值却有按键按下。这是因为在列扫描的时候按键确实没有按下，过了几毫秒后按键按下了，行扫描得到有按键按下的值。这是最终求出的是只有行数据没有列数据的键值。当然是无效的。我们只需要在判断键值没有列数据时重新进行一次按键扫描，即可得到正确的键值。

2、教师现场提的问题记录在此（不少于 2 个问题）。

答：

(1)问题 1：什么是多任务并发？

在单片机中，多任务并发是一个系统具有能够同时执行多个任务的能力。一般的单片机在同一个时刻内只能执行一条指令，即只能执行一个任务。通过多任务技术，可以让单片机在编写代码和使用的时候看起来像是能够同时执行多个任务。由于单片机的设计限制，这种技术并不是真的实现了多个任务同时执行，而是把多个任务拆分为多个小块，并在多个任务的小块之间轮流执行，单片机拥有足够快的执行速度，在人类的感知上看起来就像是具有了同时执行多个任务的能力。

(2)问题 2：为什么使用模块化的形式编写代码？

在编程世界中，模块化编写代码有非常多的好处。模块化代码将功能划分为独立的模块，使得每个模块只关注特定的任务。这样，当需要修改或优化某个功能时，只需关注特定的模块，而不必修改整个程序。这提高了代码的可维护性，减少了出错的机会。在团队开发中，模块化代码使不同的开发人员可以并行工作。每个人负责不同的模块，然后将它们集成到一个完整的系统中。模块化代码允许将底层实现细节隐藏在模块内部。其他部分的代码只需调用模块的接口，而不必关心内部实现。这提高了代码的抽象性和可读性。

摘　　要

打地鼠游戏是一个经典的益智小游戏。本次项目将利用 IAP15W4K58S4 单片机还原这个游戏场景。在项目的开始，分析了项目设计的任务要求。随后阐述了选择的硬件方案和软件方案。在硬件上使用了 IAP15W4K58S4 单片机、8 位动态数码管、4*4 矩阵键盘、无源蜂鸣器。在软件上使用了时分多任务、模块化和状态机的编程思想，将每一个硬件的驱动程序编写到独立的模块中，并且给上层（游戏逻辑层）提供应用程序接口。复杂的游戏逻辑被分为 10 个状态，在每种状态下只需执行简单的逻辑，大大降低游戏的开发难度。系统的设计开发少不了调试阶段，在早期阶段使用仿真调试可以降本增效，在后期阶段使用实机调试可以确保系统功能的真实效果。总的来说，本次系统设计是一个非常成功的项目。

关键词：8 位动态数码管；矩阵键盘；蜂鸣器；脉宽调制信号；多任务；模块化

目　录

第1章　设计任务

1.1　简介

打地鼠是经典的益智小游戏，当游戏开始时，地鼠会随机从多个洞的任意一个洞中冒出。游戏的目标是用锤子打地鼠的头，使地鼠回到洞中，以增加玩家的分数。若玩家未于一定时间内击中地鼠，地鼠会躲回洞里，分数也不会增加。游戏刚开始以慢速进行，大部分的人都能打中所有探出头的地鼠，但速度会随着进级逐渐增加，地鼠每次探出头的时间会变短，不同等级内的进级分数条件也会增加。经过预定的时间限制后，不论玩家技术如何，游戏都会结束。最终的分数取决于玩家击中的地鼠数。

我们可以利用 IAP15W4K58S4 单片机还原这个游戏场景。使用八位动态数码管来显示得分、倒计时、地鼠探出头的位置，并将地鼠探出头的位置一一对应在矩阵键盘上的按钮，通过按下对应按钮来模拟击中地鼠的头部而增加得分。使用蜂鸣器还可以为我们的操作提供声音反馈，提高打地鼠游戏场景的带入感。

即在本次设计任务中，我们需要使用 IAP15W4K58S4 单片机来完成打地鼠游戏机。

1.2　基本要求

(1) 用 12 个 LED 灯代表 12 只地鼠(见图 1.1 要求)
(2) 设计 12 个对应的按键
(3) 12 个灯能随机点亮，随即熄灭
(4) 当对应的按键按下该灯熄灭，并记一分
(5) 如不按也会自动熄灭，但不记分
(6) 设计一个总时间，总分值，由按键开始，倒计时到比赛结束，等待再启动
(7) 显示得分和比赛用时间
(8) 每次得分都有声音提示

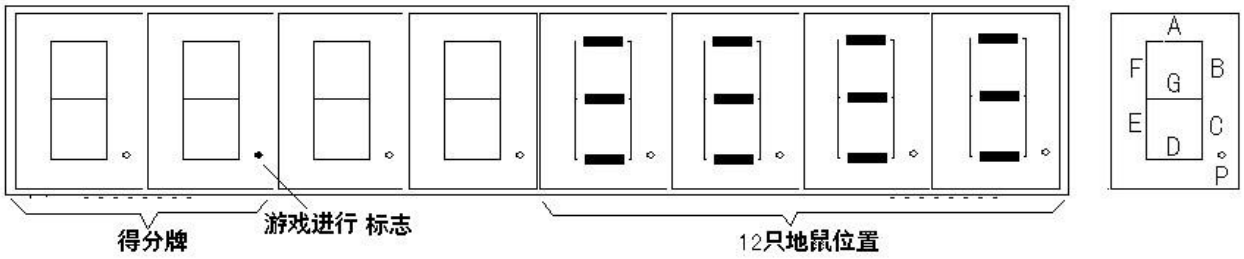


图 1.1 打地鼠显示格式图

1.3　选做

(1) 可以设定随机灯亮的持续时间（分为比赛等级，3-5 级）
(2) 可以设定升级条件（每级得分数门限）

(3) 具有进级比赛功能
(4) 具有最好成绩记录功能

1.4 任务要求之外的功能

(1) 添加游戏菜单，在游戏菜单中可以开始游戏、查看历史最高分、设置游戏难度
(2) 将比赛等级从 3-5 级扩展为 1-9 级
(3) 设置每级亮灯持续时间和升级条件时，可以分别编辑数值的个、十、百位的值
(4) 游戏中按错按键有错误音效
(5) 游戏进级时显示当前游戏等级
(6) 游戏可以暂停，暂停时可以退出、继续游戏、查看总得分和总消耗时间
(7) 游戏结束时有结束音乐
(8) 游戏结束显示得分时有得分结算动画
(9) 游戏中默认使用两位数显示得分，可按下特定按键临时查看四位数得分

第2章 设计方案

2.1 任务分析

IAP15W4K58S4 是一个增强型 8051 内核单片机，它的 SRAM 达到了 4Kbyte，其中 idata 为 256byte，xdata 为 3840byte，flash 储存器容量高达 58Kbyte，并且拥有 62 根 I/O 引脚。硬件支持 ADC,PWM,SPI,CCP,串口等实用外设。

只使用 IAP15W4K58S4 单片机无法完成打地鼠游戏机作品。为了完成设计任务的功能要求，还要为单片机连接如表 2.1 所示的硬件。为了降低代码的开发难度，需要将硬件运行逻辑与游戏逻辑进行解耦，硬件运行逻辑成为驱动程序，并向游戏逻辑提供方便使用的应用程序接口。为了便于小组合作，程序代码使用多文件模块化方式。

表 2.1 需要连接的硬件及其功能

硬件	用途	驱动文件	引脚数
8 位动态数码管	显示游戏菜单和游戏过程的画面	display.c	8 位选线+8 数据线
矩阵键盘	获取用户的输入	keyboard.c	4 行扫描+4 列扫描
无源蜂鸣器	输出声音信号	buzzer.c	1 脉宽调制信号

使用状态机的思想方法，可以降低游戏逻辑的开发难度。状态机（State Machine）是一种数学模型和计算机科学中的抽象概念，它描述了对象在其生命周期中的状态如何根据输入或事件而转换。简单来说，状态机是一个可以表示对象状态及其状态之间转换的框架或图形表示。

在日常生活和软件开发中，我们经常会遇到需要管理和跟踪对象或系统状态的情况。例如，自动售货机、电梯控制系统、游戏角色状态等都可以通过状态机来清晰、高效地描述和管理。

在软件开发中，状态机是一种非常强大和灵活的工具，用于处理复杂的逻辑和状态管理。它在多种应用场景中都有广泛的应用，如游戏开发、嵌入式系统、网络通信、UI 设计等。

状态机的应用能够帮助开发者简化复杂的业务逻辑，使代码结构更加清晰、可维护。通过状态机，我们可以将系统的各种状态和转换规则明确地定义出来，从而更容易地理解和修改代码。

将游戏逻辑分为如表 2.2 所示的 10 个状态，每一个状态在不同的条件下可以切换到另外一个状态，每种状态只需要执行特定的简单逻辑，即可大大简化了游戏的逻辑，降低开发难度。

表 2.2 游戏逻辑状态分类

序号	功能	英文标记	数码管内容
1	开始游戏	GAME RUN	GAME RUN
2	查看历史最高分	TOP	TOP 9999
3	设置每等级亮灯持续时间	TL(time level)	TL2 9999
4	设置每等级晋级分数	SL(score level)	SL2 9999
5	游戏进行中	Playing	2260 -
6	游戏暂停	PAUSE	PAUSE
7	游戏结束	GAME OVER	GAMEOVER
8	显示本次游戏耗时	T(time)	T 009999
9	显示本次游戏得分	S(score)	S 009999
10	显示当前游戏的等级	LEVEL(level)	LEVEL 2

4*4 的矩阵键盘是唯一的输入设备。设计和规划按键功能变得及其重要，在作品制作之前需要事先规划好各个按钮在游戏不同状态中的功能。虽然这个游戏被细分为了十种状态，但是按键的功能划分只被分为三种状态，他们分别是在游戏菜单时(表 2.3)的按键功能、在游戏进行中(表 2.4)的按键功能、在暂停时（表 2.5）的按键功能。以下 4*4 的表格用于模拟 4*4 的矩阵键盘，每个单元格的位置与矩阵键盘上的按钮一一对应，其中空白的单元格代表无功能的按钮。

表 2.3 游戏菜单内按键功能

		数值增大	
	编辑上一位数	确定	编辑下一位数
	上一页	数值减小	下一页

表 2.4 游戏中按键功能

地鼠位置 1	地鼠位置 2	地鼠位置 3	地鼠位置 4
地鼠位置 5	地鼠位置 6	地鼠位置 7	地鼠位置 8
地鼠位置 9	地鼠位置 10	地鼠位置 11	地鼠位置 12
查看当前等级得分			暂停

表 2.5 游戏暂停时按键功能

显示总得分	显示总消耗时间		
显示当前等级得分		退出游戏	继续游戏

2.2 方案设计

2.2.1 硬件方案

根据设计的要求可知，系统的硬件原理框图如图 2.1 所示

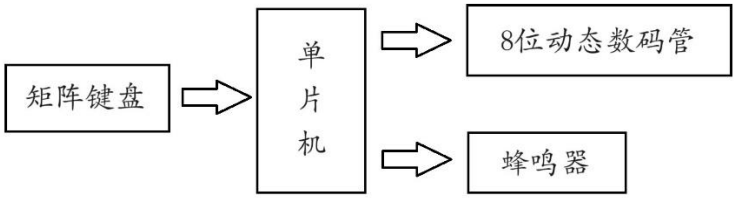


图 2.1 硬件原理框图

8 位动态数码管是一种常见的数字显示设备，广泛应用于各种电子设备中。它具有一并 8 位的引脚，用于连接并联每一个数码管上不同的 LED 灯管，可以控制数码管显示的内

容，一般称之为数据线。还有另外一并 8 为的引脚分别连接着每一个数码管的公共端，可以用于选择工作的数码管，称之为位选线。本次系统设计要求使用的开发板中提供了 P0、P1、P2、P3 共四组八位端口。数据线与位选线与单片机连接的端口可以任意选择。如图图 2.2 所示，本次系统设计将数据线连接到单片机的 P0 端口，位选线连接到单片机的 P1 端口。

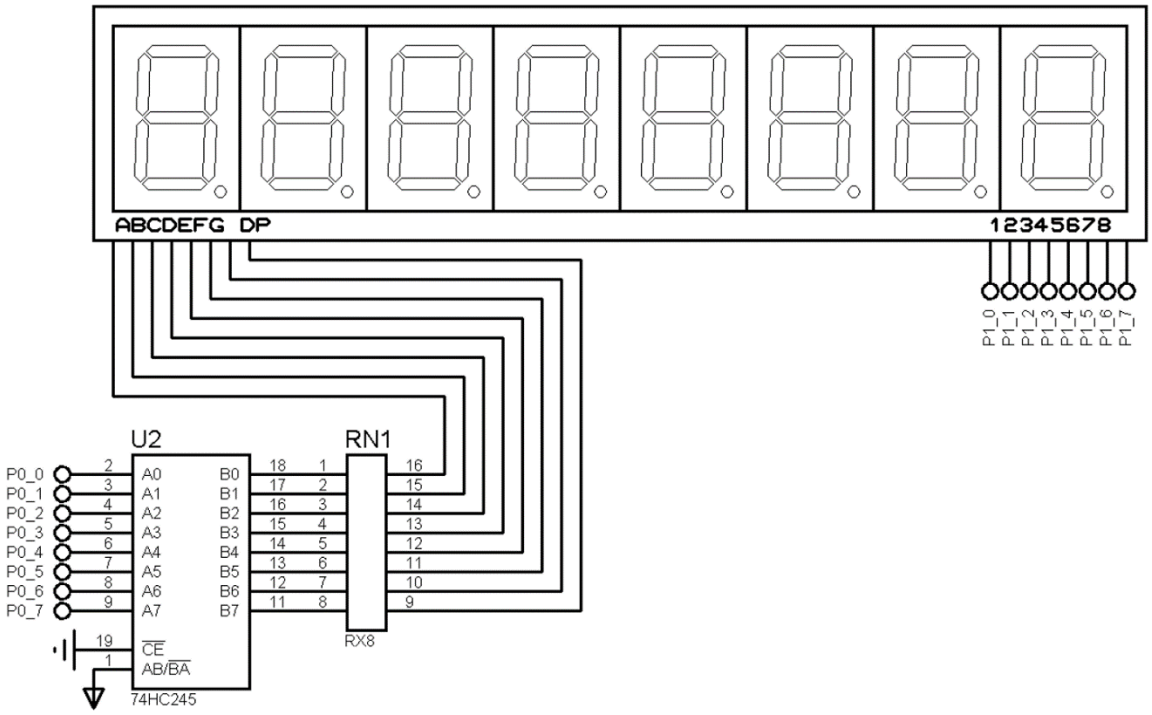


图 2.2 动态数码管接线图

矩阵键盘本质是使用 8 个 I/O 口来进行 16 个按键的控制读取，可以减小 I/O 口的使用，用 4 条 I/O 线作为行线，4 条 I/O 线作为列线组成的键盘。在行线和列线的每个交叉点上，设置一个按键。而这样的按键中按键的个数是 4 * 4 个。

这样的行列式键盘结构能够有效地提高单片机系统中 I/O 口的利用率。节约单片机的资源，其本质和独立按键类似，就是进行逐行扫描和逐列扫描，然后判断是第几行的第几列的按键，进而进行整体按键值得确定，如图图 2.3 所示，我们使用的矩阵键盘是接到了单片机的 P2 口通过读取 P2 口电平变换即可完成矩阵键盘的数值读取。

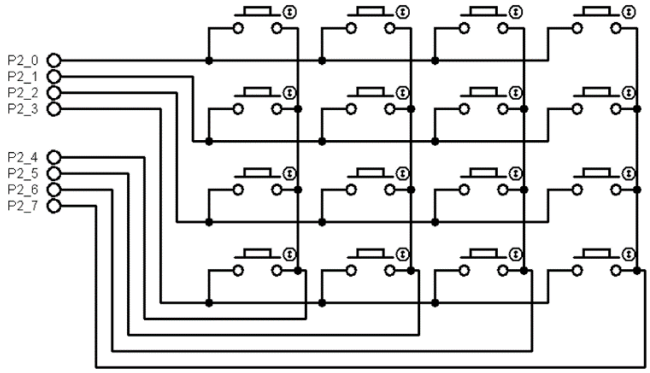


图 2.3 矩阵键盘接线图

无源蜂鸣器内部没有激励源，只有给它一定频率的方波信号，才能让蜂鸣器的振动装置起振，从而实现发声，同时，输入的方波频率不同，发出的声音也不同，无源蜂鸣器可以模拟曲调实现音乐效果。单片机驱动无源蜂鸣器的方式有两种：一种是 PWM 输出口直接驱动，另一种是利用 I/O 定时翻转电平产生驱动波形对蜂鸣器进行驱动。

在本次实验中为了充分利用 IAP15W4K58S4 单片机的硬件功能，选择了使用 PWM 脉宽调制信号来驱动无源蜂鸣器。PWM 输出口直接驱动是利用 PWM 输出口本身可以输出一定的方波来直接驱动蜂鸣器。在单片机的软件设置中有几个系统寄存器是用来设置 PWM 口的输出的，可以设置占空比、周期等等，通过设置这些寄存器产生符合蜂鸣器要求的频率的波形之后，只要打开 PWM 输出，PWM 输出口就能输出该频率的方波，这个时候利用这个波形就可以驱动蜂鸣器了。

IAP15W4K58S4 单片机具有 6 路 PWM 输出，每一路输出还可以通过修改 PWMnCR 中的 PWMn_PS 位独立控制切换到第一引脚或者第二引脚。该单片机共有如表 2.6 所示共 12 个输出引脚。其中 P0~P2 端口已经被其他硬件占用，P3.7 被下载串口占用。剩余 P4.4、P4.2、P4.5 引脚可选择。

表 2.6 IAP15W4K58S4 单片机的 PWM 输出引脚

PWM 通道	PWM2	PWM3	PWM4	PWM5	PWM6	PWM7
PWMn_PS=0	P3.7	P2.1	P2.2	P2.3	P1.6	P1.7
PWMn_PS=1	P2.7	P4.5	P4.4	P4.2	P0.7	P0.7

如图 2.4 所示，我们将选用 P4.4 端口作为无源蜂鸣器 PWM 脉宽调制信号输出端口。

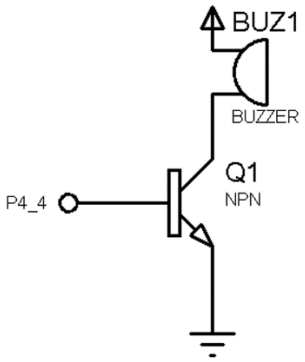


图 2.4 无源蜂鸣器接线图

2.2.2 软件方案

在软件的设计上，如表 2.7 所示，我们采用了模块化的设计方案。模块化是指将一个大的系统或程序分解成独立的模块并按照一定的规则组合在一起的设计方法，每个模块都有自己的功能和接口。在编写代码时，开发人员可以将代码分成多个模块，每个模块只负责特定的任务或功能。

模块化的好处是可以提高代码的可重用性、可维护性和可扩展性。模块化可以将代码划分为相对独立的模块，每个模块都只关注自己的功能。这样，当需要修改某个功能时，只需修改相应的模块，而不需要关注整个系统。这大大提高了代码的可维护性。模块化可以将代码分成多个小块，每个小块都可以独立地被重复被调用。当要添加新的功能时，只需要编写新的模块，而不用修改整个架构，然后将新添加的模块添加即可。

表 2.7 模块化软件

模块	文件	功能
引脚管理	pin.h	管理相关硬件所对应的引脚接口
时分多任务架构	framework.h、framework.c	实现单片机同时运行多个任务
动态数码管	display.h、disaplay.c	实现数码管自动刷新，提供程序接口
矩阵键盘	keyboard.h、keyboard.c	实现按键扫描，提供按键事件接口
无源蜂鸣器	buzzer.h、buzzer,c	实现音乐播放器，提示音和程序接口
随机发生器	random.h、random.c	实现伪随机数生成，提供程序接口
游戏循环	game.h、game.c	实现游戏菜单和游戏逻辑

传统的单片机程序一般采用单任务机制，单任务系统具有简单直观、易于控制的优点。然而由于程序只能按顺序依次执行，缺乏灵活性，在较复杂的应用中使用极为不便。在单片机系统开发中，多任务并发是非常常见的，对于处理复杂的应用场景、提升系统的并发能力、提高系统的实时性等方面都有很大好处。在单片机中实现多任务并发是非常重要的。如图 2.5 所示，我们在底层通过定时器 0 实现了由时间周期分类驱动的伪多任务架构，实现单片机能刷新数码管、刷新蜂鸣器、扫描键值和更新游戏循环的“并发”运行。

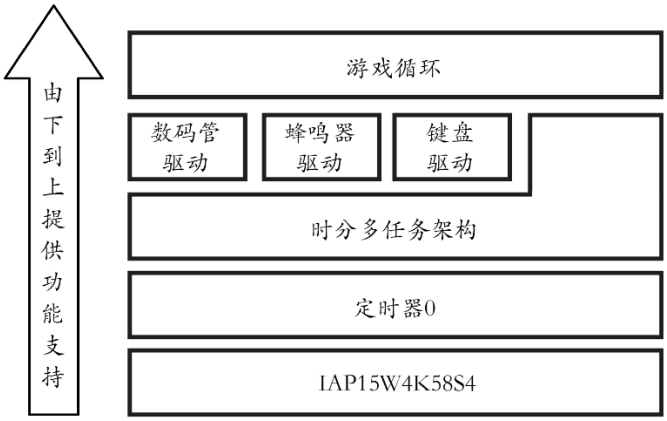


图 2.5 系统架构

驱动程序与硬件相关,同时给应用程序提供 API 函数接口,应用层可以调用这些接口去访问和操控硬件而不必了解硬件底层实现逻辑。我们采用了类似操作系统的思想。将硬件运行的逻辑独立编写为模块化文件，相当于驱动程序，并给上层的游戏循环逻辑提供应用程序接口。在写游戏逻辑时可以简单地快速调用硬件资源。让游戏逻辑更专心于游戏逻辑。降低程序开发难度。

第3章　系统硬件设计

3.1　系统总电路设计

本次系统设计被要求使用特定开发板，系统电路只能使用开发板上预留的硬件电路。可以通过修改跳线连接来修改少部分连接效果。图 3.1 是根据硬件方案的端口设计和开发板预留电路绘制的 Protues 仿真电路图。

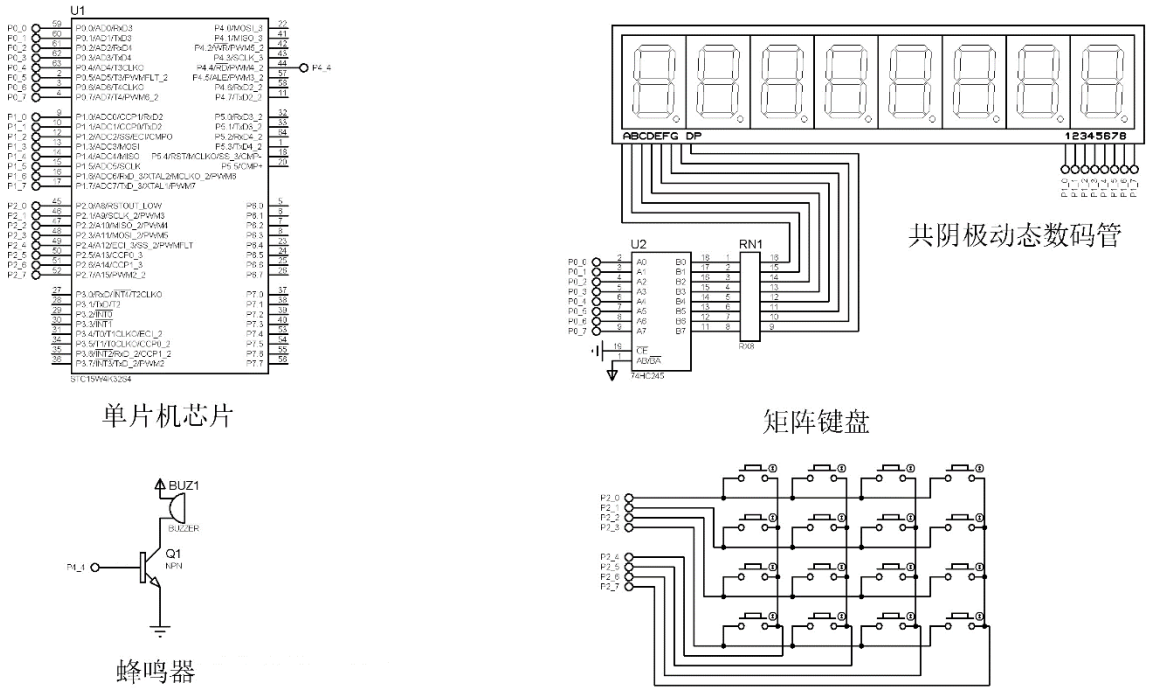


图 3.1 系统总仿真电路图

实物的接线图如图 3.2 所示，将 JP10 与 J12 连接，这是数码管数据线。将 JP8 与 J16 连接，这是数码管位选线。将 JP11 与 JP4 连接，这是矩阵键盘行列扫描线。将 P4.4(单片机封装转接板左下角开始第四根线)接到 J8，这是蜂鸣器的 PWM 输出信号线。

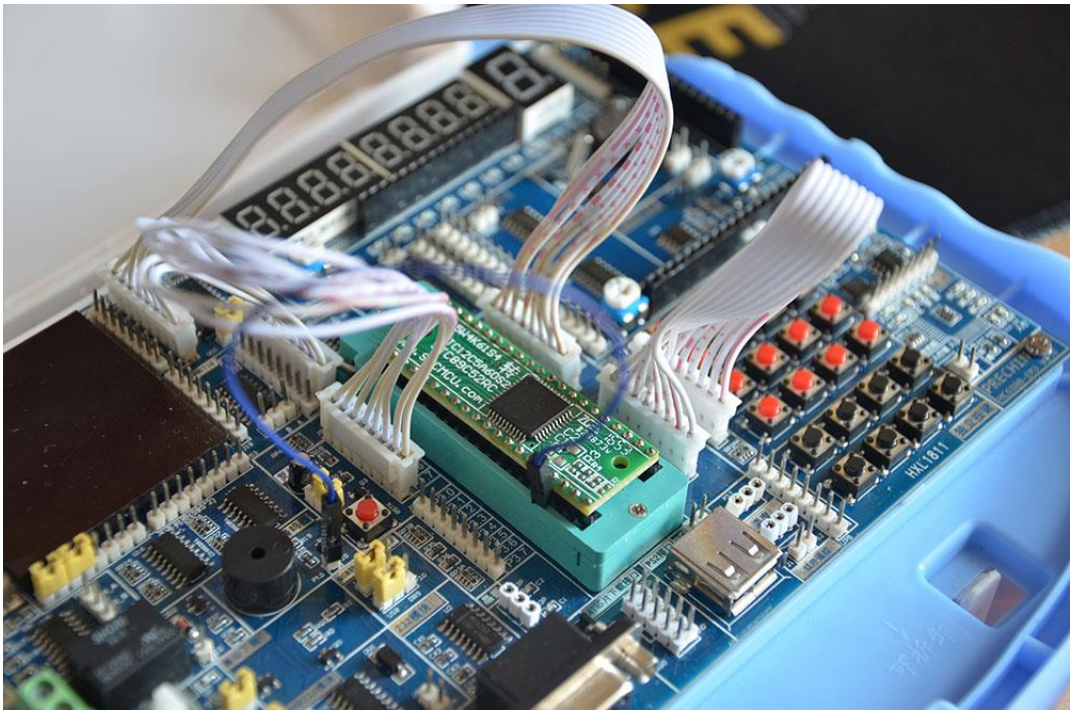


图 3.2 实机接线图

第4章　系统软件设计

4.1　引脚管理

预编译是做些代码文本的替换工作。处理以#开头的指令，比如拷贝#include 包含的文件代码，#define 宏定义的替换，条件编译等，就是为编译做的预备工作的阶段。主要处理#开始的预编译指令，预编译指令指示了在程序正式编译前就由编译器进行的操作，可以放在程序中的任何位置。C 编译系统在对程序进行通常的编译之前，首先进行预处理。

在 pin.h 文件中使用 C 语言预编译命令#define 定义了如表 4.1 所示宏定义，在编译时将代码中的所有 PORT_DISPLAY_DATA 标志替换成 P0，即可实现单片机所用引脚的统一管理。每当需要修改硬件连接引脚时，只需修改 pin.h 内的预编译代码，即可生成对应的单片机程序文件。

表 4.1 引脚管理

名称	引脚	功能
PIN_BUZZER	P4_4	蜂鸣器
PORT_DISPLAY_DATA	P0	数码管数据线端口
PORT_DISPLAY_SELECT	P1	数码管位选线端口
PORT_KEYBOARD	P2	矩阵键盘端口

4.2　时分多任务框架

单片机实现多任务的意义在于提高系统效率、降低程序开发难度、响应速度以及资源利用率。通过多任务技术，单片机可以并行处理多个任务。任务调度是关键概念，它规划任务的顺序和时间，使 CPU 资源充分利用，避免资源争抢和堵塞。自行编写简单的多任务系统，虽不是真正的操作系统，但理解其原理后可扩展为强大的系统。

通常单片机只有一个处理器核心，因此在同一时刻只能执行一个指令流。无法真正实现多个任务同时处理。如果把每一个任务分为多个小块，在不同时间内顺序执行不同任务的小块，单片机的运算速度非常快，在短时间内可以在多个任务的小块中按顺序切换多次，在人类的感知上就好像单片机同时运行了多个任务一样。

我们编写的时分多任务框架代码在 framework.h、framework.c 和 task.c 中，是一款简单的入门多任务处理框架。如图 4.1 所示，使用 IAP15W4K58S4 单片机的定时器 0 作为总驱动，设置每 1 毫秒运行时分多任务框架的中断函数。在该函数中需要判断当前是否达到 2ms 周期、10ms 周期、500ms 周期、1s 周期并按顺序调度相应任务。该框架并没有实现多任务的自动分片，所以要求编写任务函数时应当避免使用死循环和 delay 函数。否则会因为任务无法及时跳出而切换到下一个任务失败。

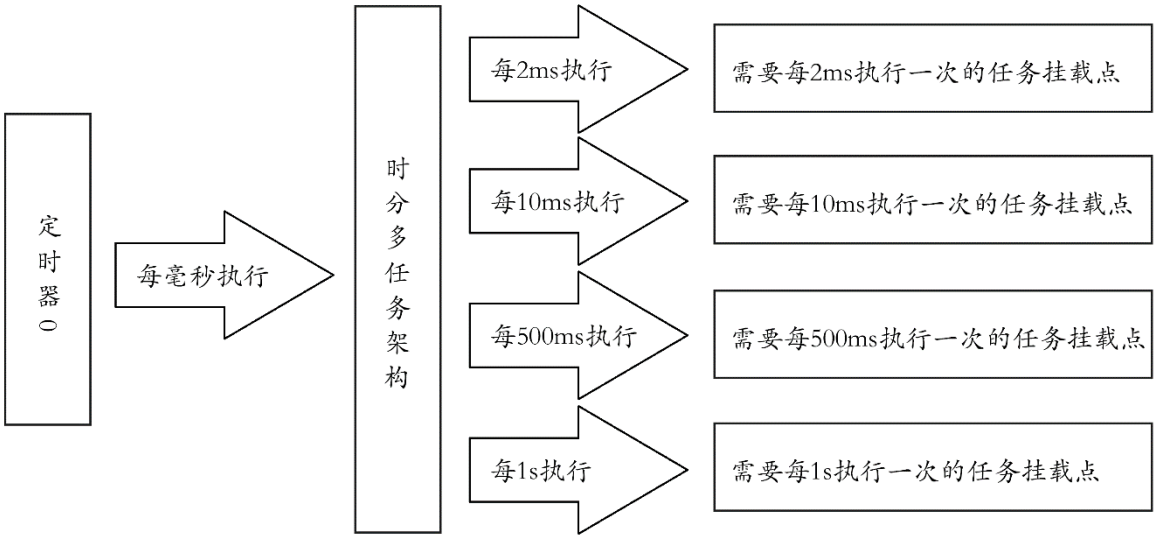


图 4.1 时分多任务架构

Delay 函数的功能是非常必要的，delay 函数允许我们在程序中添加暂停，以便在特定时间间隔内执行操作。例如，我们可以使用 delay(1000) 来等待 1 秒钟。这对于定时、脉冲生成、传感器读取等任务至关重要。但在我们编写的时分多任务框架上应当避免使用 delay 函数，delay 函数会造成任务阻塞从而导致任务无法及时跳出而切换到下一个任务失败。因为该函数的重要性，我们不得不重新设计一个不会阻塞代码运行的 delay 函数。因为它不会阻塞代码运行的特性。我们将他命名为 SETTIMEOUT。为了符合 c89 标准要求，在使用 SETTIMEOUT 功能前必须先在函数开头使用 USE_TIMEOUT 声明延时计数器的计数，等效为使用代码 static uint time_out_count = 0;time_out_count++;声明延时计数器的计数。SETTIMEOUT 等效于 if(time_out_count % num == 0)，所以使用 SETTIMEOUT 的延时时间长度只能是当前所在的函数执行周期的倍数。SETTIMEOUT 的写法为 SETTIMEOUT(延时倍数){延时执行的语句}。

如表 4.2 所示，时分多任务架构在区分当前时间点所达到的时间周期和调动对应任务使用位标志来标记状态。使用位标志（二进制位数）来表示状态时，我们可以将多个状态存储在一个整数变量中。例如，一个 8 位整数可以表示 8 个不同的状态，而不需要为每个状态单独分配一个变量。这节省了内存空间，特别是在资源受限的嵌入式系统中。位操作（如位设置、清除、翻转）通常比使用布尔变量更高效。我们可以使用位掩码和位运算来设置或检查特定状态，而不需要执行额外的逻辑操作。如果我们有多个相关的开关或标志，使用位标志可以将它们压缩到一个整数中。这样，我们可以在一个变量中同时跟踪多个状态，而不会增加额外的变量数量。使用位标志可以使代码更简洁，因为我们不需要为每个状态声明单独的变量。这有助于提高代码的可读性和维护性。如表 4.2 所示，我们目前只使用了 unsigned char 变量中的四位数据来储存状态，还剩余四位未来扩展新的状态。

表 4.2 时分多任务架构的位标记

名称	16 进制	2 进制	功能
TIME_2MS	0x01	0000 0001	代表当前 2ms 周期已到
TIME_10MS	0x02	0000 0010	代表当前 10ms 周期已到
TIME_500MS	0x04	0000 0100	代表当前 500ms 周期已到

TIME_1S	0x08	0000 1000	代表当前 1s 周期已到

4.3 数码管驱动

我们使用的是共阴极 8 位动态数码管，它有一并 8 位数据输入口（称为数据线）和一并 8 位数码管位选数据口（称为位选线）。如图 4.2 所示，数据线并联着每一个数码管的 LED 灯脚，这些 LED 灯的位置依次被定义为 A、B、C、D、E、F、G、DP。

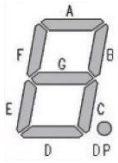


图 4.2 数码管的 LED 灯定义

位选线分别连接着 8 个数码管 LED 端的公共端。因为我们使用的是共阴极的数码管，也就是 LED 灯的公共端是阴极的。只有在公共端接通低电平，LED 灯的另一端接通高电平时 LED 灯才能被正确驱动发光。适当配置数据线的数据，即可让数码管显示多个种内容。适当配置位选线，即可配置数码管的工作状态。如表 4.3 是根据数码管字形计算的数据线数据结果。

表 4.3 数码管字形码

内容	名称	数据线配置（2 进制）	数据线配置（16 进制）
0	NUMBER(0)	0011 1111	0x3f
1	NUMBER(1)	0000 0110	0x06
2	NUMBER(2)	0101 1011	0x5b
3	NUMBER(3)	0100 1111	0x4f
4	NUMBER(4)	0110 0110	0x66
5	NUMBER(5)	0110 1101	0x6d
6	NUMBER(6)	0111 1101	0x7d
7	NUMBER(7)	0000 0111	0x07
8	NUMBER(8)	0111 1111	0x7f
9	NUMBER(9)	0110 1111	0x6f
A	LETTER_A	0111 0111	0x77
B	LETTER_B	0111 1100	0x7c
C	LETTER_C	0011 1001	0x39
D	LETTER_D	0101 1110	0x5e
E	LETTER_E	0111 1001	0x79
F	LETTER_F	0111 0001	0x71
G	LETTER_G	0111 1101	0x7d
T	LETTER_T	0000 0111	0x07
O	LETTER_O	0011 1111	0x3f
U	LETTER_U	0011 1110	0x3e

R	LETTER_R	0011 0001	0x31
N	LETTER_N	0011 0111	0x37
P	LETTER_P	0111 0011	0x73
L	LETTER_L	0011 1000	0x38
S	LETTER_S	0110 1101	0x6d
OFF	DISPLAY_OFF	0000 0000	0x00
ON	DISPLAY_ON	1111 1111	0xff

数码管驱动相关文件在 display.h 和 display.c 内。数码管显示的内容储存在显存数组变量 Display_Memory[]里。显存（display memory），全称显示内存，亦称帧缓存，在本次项目中，它是用来存储被单片机游戏循环逻辑处理过或者即将读取的渲染数据。显存是用来存储数码管显示内容的区域。在八位动态数码管上显示出的画面是由一个个数码管内容构成的，而每个数码管都以 8 位的数据来控制它的内容，这些数码管构成一帧的图形画面。如图 4.3 所示，刷新数码管内容的函数为 Refresh_Display_Hook，该函数每两毫秒执行一次，每执行一次函数，就会操作数码管位选线激活特定的一个数码管工作，从显存数据 Display_Memory[]种获取一个数码管内容，并发送到数码管数据线上。因此，最终八位动态数码管的工作帧率为

$$fps = \frac{1000}{2 \times 8} = 62.5 \qquad (4-1)$$

这个刷新率已经超过 50Hz，并且刷新函数由中断函数驱动，刷新速度始终稳定维持在 62.5Hz，能够保证人眼观察到的是稳定不闪速的显示效果。

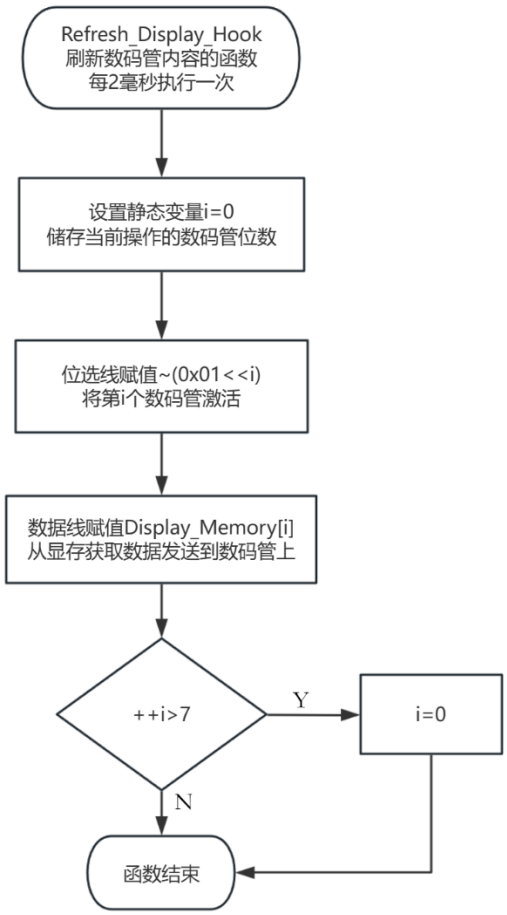


图 4.3 数码管刷新函数流程图

4.4 矩阵键盘驱动

在键盘中按键数量较多时，为了减少 I/O 口的占用，通常将按键排列成矩阵形式。在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。这样，一个端口就可以构成 4*4=16 个按键，比之直接将端口线用于键盘多出了一倍，而且线数越多，区别越明显，比如再多加一条线就可以构成 20 键的键盘，而直接用端口线则只能多出一键（9 键）。由此可见，在需要的键数比较多时，采用矩阵法来做键盘是合理的。

矩阵式结构的键盘显然比直接法要复杂一些，识别也要复杂一些。因为每一个按键所在的行和列不可能和其他按键的相同。所以要区分 16 个按键，只需要检测出按下按键所在的行和所在的列就能求出按下的按键。矩阵键盘的扫描分为两个阶段，其中一个阶段是列扫描，求出按下的按键所在的列位置。剩下的另外一个阶段是行扫描，求出按下按键所在的行位置。这两个阶段的顺序可以随意设置。所以，求出按下按键至少需要经过两次扫描。

接下来将以识别矩阵键盘左上方的第 1 个按键为例子来讲解这个过程。在列扫描的时候，如图 4.4 左侧所示，给四个列引脚设置为高电平，给四个行引脚设置为低电平。如果此时按下按键 1，第一列的线路通过按键 1 的闭合与第一行的线路连接在一起。导致第一列的线路等效于直接接到低电平。矩阵键盘端口使用准双向配置，电流的输出能力远远小于电流的输入能力。导致该列原本的高电平被拉低，出现如图 4.4 右侧所示效果。此时读出四根列输出，即可完成列扫描，确定按下按键所在的列位置。扫描出的列数据是 0111。

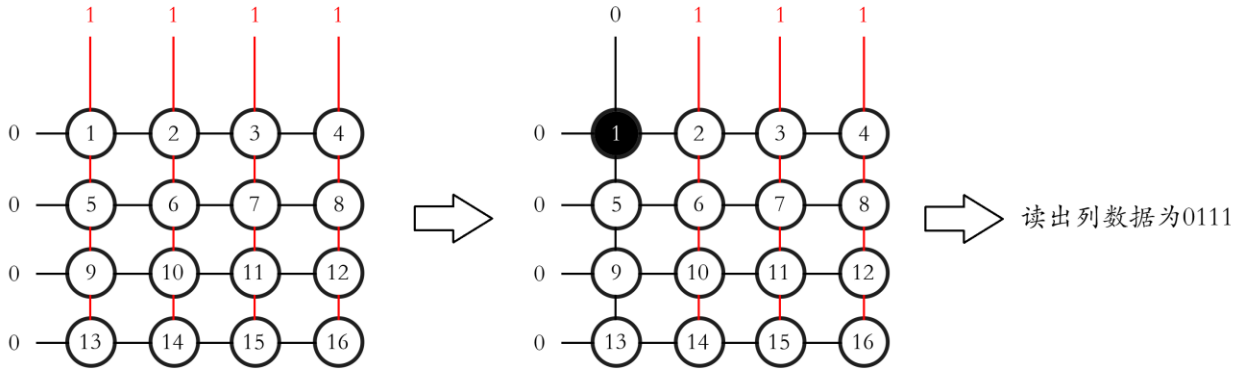


图 4.4 矩阵键盘列扫描

行扫描则是与列扫描相反的做法。在行扫描的时候，如图 4.5 所示，给四个行引脚设置高电平，给四个列引脚设置为低电平。如果此时按下按键 1，第一行的线路通过按键 1 的闭合与第一列的线路连接在一起。导致第一行的线路等效于直接列到低电平。导致该行原本的高电平被拉低，出现如图 4.5 所示效果。此时读出四根行输出，即可完成行扫描，确定按下的按键所在的行位置。扫描出的行数据是 0111。此时行数据与列数据组合在一起变为 0111 0111 用于识别按键 1 按下的键值码。

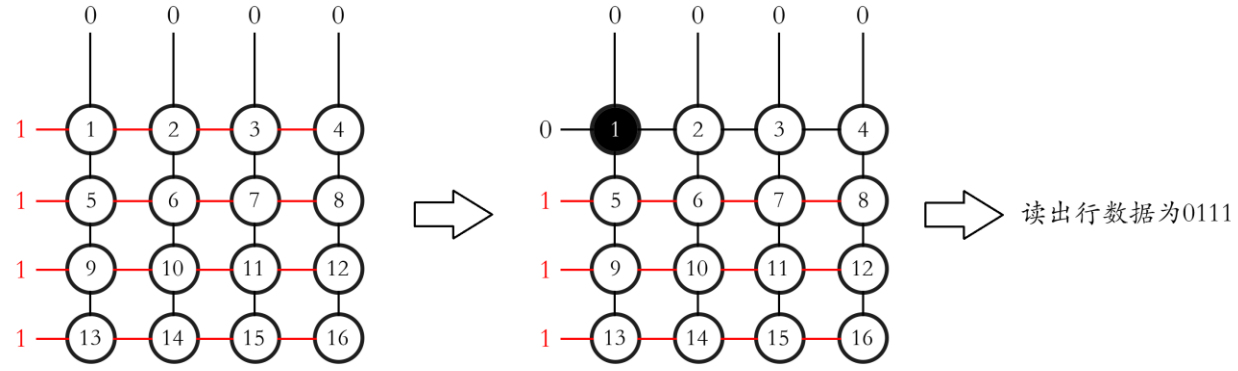


图 4.5 矩阵键盘行扫描

通过上述原理和结合实际线路连接顺序的差异，我们计算出如表 4.4 所示所有按键所对应的键值码。

表 4.4 矩阵键盘键值码

按键名	键值码（2 进制）	键值码（16 进制）
KEY_1	1110 1110	0xEE
KEY_2	1101 1110	0xDE
KEY_3	1011 1110	0xBE
KEY_4	0111 1110	0x7E
KEY_5	1110 1101	0xED
KEY_6	1101 1101	0xDD
KEY_7	1011 1101	0xBD
KEY_8	0111 1101	0x7D
KEY_9	1110 1011	0xEB

KEY_10	1101 1011	0xDB
KEY_11	1011 1011	0xBB
KEY_12	0111 1011	0x7B
KEY_13	1110 0111	0xE7
KEY_14	1101 0111	0xD7
KEY_15	1011 0111	0xB7
KEY_16	0111 0111	0x77
KEY_NULL	1111 1111	0xFF

我们编写的矩阵键盘驱动程序在 keyboard.h 和 keyboard.c 内部。为了向上提供应用程序接口，我们编写了一个函数 Decode_Key_Event 用于解码按键事件，即区分当前按键是按下还是抬起。如图 4.6 所示，程序始终保存着上一次的键值和此时最新扫描得出键值。首先判断上次的键值和最新的键值是否相同，如果相同说明按键状态无变化，无新的按键事件。如果发生了变化则判断这次扫描得到的键值是否代表没有按键按下（即键值为 KEY_NULL）。如果是的话那么当前就是从按下按键变为没有按下按键的过程，即抬起了按键，抬起的键值是保存的上次按下的键值。如果此时扫描得出的键值不为空，那么代表按键按下了，按下按键的键值就是当前扫描出来的键值。

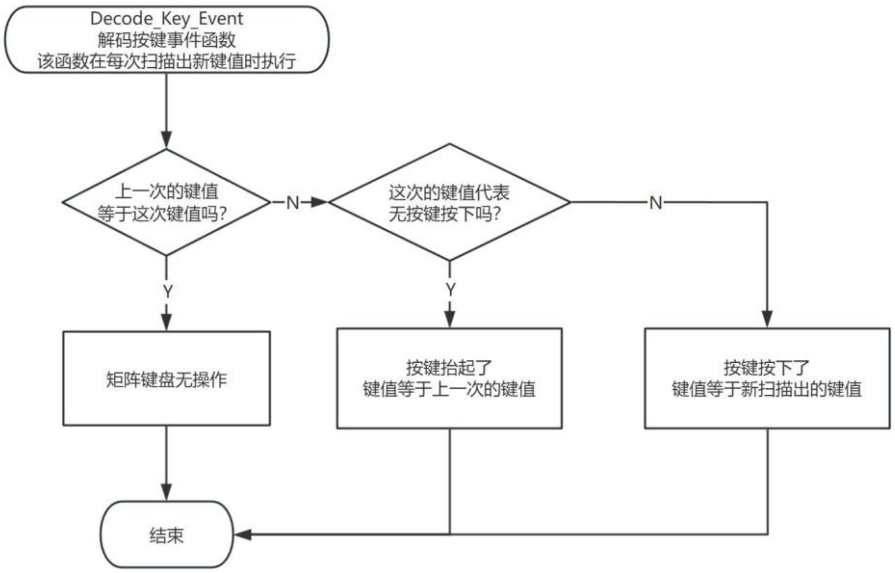


图 4.6 解码按键事件函数流程图

拥有了按键事件，就可以通过构造宏定义来使用这些按键事件。如图 4.7 所示，我们构造了 KEY_DOWN 宏定义用于快速编写按下对应按键时执行对应代码，等效于直接编写代码 for(;Key_Down_Value == value;Key_Down_Value = KEY_NULL)。使用这个功能时，只需要在需要的位置写下 KEY_DOWN(键值码){需要执行的代码}即可实现在按下对应键值按键时执行需要执行的代码。KEY_UP 宏定义用于快速编写抬起对应按键时执行对应的代码，等效于编写代码 for(;Key_Up_Value == value;Key_Up_Value = KEY_NULL)。使用这个功能时，只需要在需要的位置写下 KEY_UP(键值码){需要执行的代码}即可实现在抬起对应键值按键时执行需要执行的代码。ANY_KEY_DOWN 用于快速编写按下任意按键时执

行对应的代码，等效于编写代码 for(;Key_Down_Value != KEY_NULL;Key_Down_Value = KEY_NULL)。

```
KEY_DOWN(KEY_1){
    //KEY_1可替换为其他键值
    //在按下按键时
    //需要执行的代码
}

KEY_UP(KEY_1){
    //KEY_1可替换为其他键值
    //在抬起按键时
    //需要执行的代码
}
```

```
ANY_KEY_DOWN{
    //按下任意键时
    //需要执行代码
    //如果需要读取键值
    //请直接读取变量Key_Value
}
```

图 4.7 矩阵键盘程序接口

4.5 蜂鸣器驱动

无源蜂鸣器内部没有激励源，只有给它一定频率的方波信号，才能让蜂鸣器的振动装置起振，从而实现发声，同时，输入的方波频率不同，发出的声音也不同，无源蜂鸣器可以模拟曲调实现音乐效果。PWM 输出口直接驱动是利用 PWM 输出口本身可以输出一定的方波来直接驱动蜂鸣器。我们将选用 P4.4 端口作为无源蜂鸣器 PWM 脉宽调制信号输出端口。

关于蜂鸣器的代码在文件 buzzer.h 和 buzzer.c 中。在单片机开始时先要初始化蜂鸣器端口，由表 4.6 可知我们需要将 PWM4 信号配置到第二输出引脚即可使用 P4.4 引脚输出 PWM 信号。如表 4.5 所示，要配置 PWM 在扩展 RAM 区的特殊功能寄存器，必须先将 EAXSFR 位置为 1；B6、B5、B4 为内部测试使用，用户必须填 0。

表 4.5 端口配置寄存器 P_SW2

位号	B7	B6	B5	B4	B3	B2	B1	B0
位名称	EAXSFR	0	0	0	—	S4_S	S3_S	S2_S

通过配置 PWM4CR 中的 PWM4_PS 标志位为 1，即可将 PWM4 输出从第一引脚 P2.2 切换到第二引脚 P4.4。

表 4.6 PWMn 的控制寄存器 PWMnCR

位号	B7	B6	B5	B4	B3	B2	B1	B0
位名称	—	—	—	—	PWMn_PS	EPWMnI	ECnT2SI	ECnT1SI

通过配置 PWM 的周期从而实现蜂鸣器发出不同的音调，PWM 计数器为一个 15 位的寄存器，如表 4.7 和表 4.8 所示，通过配置 PWMCH 和 PWMCL 即可控制 PWM 的波形周期。PWMCH 是 PWM 计数器的高字节，地址是 0xfff0。PWMCL 是 PWM 计数器的低字节，地址是 0xfff1，由此可见 PWMCH 和 PWMCL 是连续的。在单片机的头文件 stc15.h 翻

阅可发现在 268 行有一个宏定义为 PWMC 的地址正好是 0xfff0，通过此处赋值一个 1 到 32767 的值即可一次赋值同时更新 PWM 的高字节和低字节。

表 4.7 PWM 计数器高字节 PWMCH

位号	B7	B6	B5	B4	B3	B2	B1	B0
位名称	—	PWMCH[14:8]						

表 4.8 PWM 计数器低字节 PWMCL

位号	B7	B6	B5	B4	B3	B2	B1	B0
位名称	PWMCL[7:0]							

我们的开发板上使用的晶振频率为 11.0592MHz，如果要蜂鸣器输出 1KHz 的声音，即输出周期为 1ms 的方波信号，需要为 PWM 计数器设置数值 11059。由此为标准，我们就可以得出公式

$$音符PWM计数值 = \frac{1KHz音符PWM计数值 \times 1000}{音符频率} \qquad (4-2)$$

来快速算出每一个音符应当设置的 PWM 计数器值。如表 4.9、表 4.10、表 4.11 是我们通过这个公式求出的中音、低音和高音音符对应的 PWM 计数器值。

表 4.9 中音音符对应的 PWM 计数器值

音符唱名	音符频率	在代码中的名称	PWM 计数器值
Do	523Hz	DO	21145
Do#	554Hz	DOS	19962
Re	587Hz	RE	18840
Re#	622Hz	RES	17780
Mi	659Hz	MI	16781
Fa	698Hz	FA	15844
Fa#	740Hz	FAS	14945
So	784Hz	SO	14106
So#	831Hz	SOS	13308
La	880Hz	LA	12567
La#	932Hz	LAS	11866
Si	988Hz	SI	11193

由表 4.10 可以看到低音 Do 到低音 Mi 的配置中，PWM 计数器的值远远大于 PWM 的 15 位寄存器的最大值 32767，所以这几个低音是无法使用的。我们在编写乐谱时应该避免使用这个几个低音。

表 4.10 低音音符对应的 PWM 计数器值

音符唱名	音符频率	在代码中的名称	PWM 计数器值
Do	262Hz	_DO	42210
Do#	277Hz	_DOS	39924

Re	294Hz	_RE	37616
Re#	311Hz	_RES	35559
Mi	330Hz	_MI	33512
Fa	349Hz	_FA	31688
Fa#	370Hz	_FAS	29889
So	392Hz	_SO	28212
So#	415Hz	_SOS	26648
La	440Hz	_LA	25134
La#	466Hz	_LAS	23732
Si	494Hz	_SI	22387

表 4.11 高音音符对应的 PWM 计数器值

音符唱名	音符频率	在代码中的名称	PWM 计数器值
Do	1046Hz	DO_	10573
Do#	1109Hz	DOS_	9972
Re	1175Hz	RE_	9412
Re#	1245Hz	RES_	8883
Mi	1318Hz	MI_	8391
Fa	1397Hz	FA_	7916
Fa#	1480Hz	FAS_	7472
So	1568Hz	SO_	7053
So#	1661Hz	SOS_	6658
La	1760Hz	LA_	6284
La#	1865Hz	LAS_	5930
Si	1976Hz	SI_	5597

为了蜂鸣器能播放音乐，我们要实现一个蜂鸣器音符播放器来播放乐谱，在实现音符播放器之前，先规划一个乐谱格式。编写乐谱时统一按照如图 4.8 所示格式编写乐谱，音符播放器按照此格式播放音符。

```
uchar code Muisc_Gameover[] = {
    DO_,16,MUTE,30,SO,13,MUTE,30,MI,30,
    LA,20,SI,20,LA,20,
    SO,30,LA,30,SO,30,
    MI,10,RE,13,MI,63,
    FINISH
};
```

图 4.8 乐谱格式

如图 4.8 所示，我们规定了一种乐谱格式。使用数组来储存该乐谱，从索引 0 开始，0 和偶数位是音符的唱名代码，唱名的代码如表 4.9 至表 4.11 所示。音符的下一位是该音符播放的持续时间，单位为 10ms。使用 MUTE 来标记静音的音符，使用 FINISH 来标记乐谱的结束位置。

如图 4.9 右所示，定义了一个 Buzzer_Player_Hook 函数，当系统的 Player_State 变量为 PLAYER_PALY 时，该函数每 10ms 执行一次。执行时设置静态变量 Player_Ps 在第一次执行为 0，用于记录当前播放音符的位置。从当前播放乐谱中获取当前播放的音符。如果该音符是 FINISH 标识，将 Player_State 设置为 PLAYER_STOP，结束音乐的播放。如果该音符不是 FINISH 标识，那么判断当前是否第一次读取这个音符，如果是的话调用 Buzzer_Tone 播放当前音符。判断当前是否达到了该音符规定的时间，是的话给记录播放音符位置的变量 Player_Ps 加 2，用于播放下一个音符。

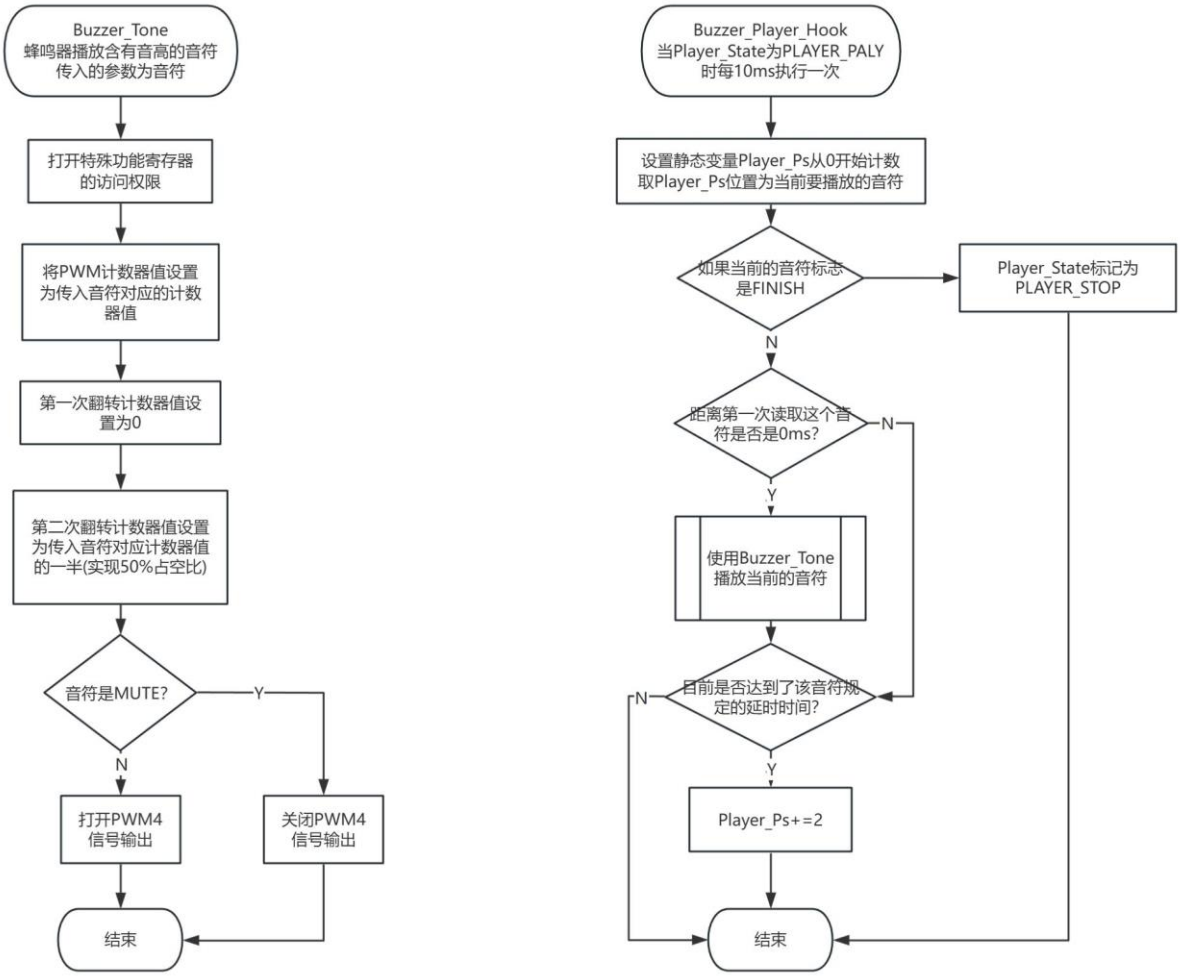


图 4.9 蜂鸣器音符播放器流程图

蜂鸣器驱动为上层应用提供四个应用程序接口，第一个是 uchar Buzzer_Is_Alert，将它置 1 蜂鸣器响起一次错误警报音，播放结束后自动归零。第二个是 uchar Buzzer_Is_Bingo，将它置 1 蜂鸣器响起一次 Bingo 音，播放结束后自动归零。第三个是 void Buzzer_Player_Play(uchar *note_list)函数，参数为乐谱的指针。调用它时会播放该乐谱。如果当前已经有播放的乐谱，调用它会替换当前的乐谱并且重新播放。第四个是 void Buzzer_Player_Stop()函数，调用它会时当前播放的乐谱停止播放。

4.6 游戏循环

根据任务设计方案，如表 4.12 所示，将游戏功能按键映射到 16 个按键中。也就是为矩阵驱动定义的 16 个按键设置别名，可以增加游戏代码的可读性和可维护性。

表 4.12 游戏按键映射

按键别名	功能	原始按键名
KEY_ADD	设置时间或分数时增加数值	KEY_3
KEY_MINUS	设置时间或分数时降低数值	KEY_11
KEY_PREV_BIT	设置时间或分数时编辑上一位	KEY_6
KEY_NEXT_BIT	设置时间或分数时编辑下一位	KEY_8
KEY_OK	确定按键	KEY_7
KEY_PREV_PAGE	游戏菜单翻到上一页	KEY_10
KEY_NEXT_PAGE	游戏菜单翻到下一页	KEY_12
KEY_PAUSE	游戏暂停	KEY_16
KEY_CONTINUE	游戏继续	KEY_16
KEY_EXIT	退出游戏	KEY_15
KEY_LEVEL_SCORE	显示当前等级得分	KEY_13
KEY_TOTAL_SCORE	显示游戏总得分	KEY_9
KEY_TOTAL_TIME	显示游戏总消耗时间	KEY_10

如表 4.13 游戏状态分类所示，使用状态机的思想方法，可以降低游戏逻辑的开发难度。将游戏循环划分为 10 个状态，每一个状态在不同的条件下可以切换到另外一个状态，每种状态只需要执行特定的简单逻辑，即可大大简化了游戏的逻辑，降低开发难度。

表 4.13 游戏状态分类

状态	功能
GAME_STATE_MENU_RUN	显示游戏开始界面
GAME_STATE_MENU_TOP	显示历史最高分
GAME_STATE_MENU_TL	设置各等级小鼠探出头的持续时间
GAME_STATE_MENU_SL	设置各等级的进级分数条件
GAME_STATE_PLAYING	打地鼠进行中的界面和逻辑
GAME_STATE_PAUSE	游戏暂停的界面和逻辑
GAME_STATE_OVER	游戏结束界面
GAME_STATE_SHOW_TOTAL_SCORE	显示游戏当前的总分数
GAME_STATE_SHOW_TOTAL_TIME	显示游戏当前所消耗的总时间
GAME_STATE_NEXT_LEVEL	显示即将开始游戏的等级

在游戏循环中，使用 uchar Game_State 变量储存游戏状态。使用 switch 分支来判断当前的状态，进入不同的分支。

第5章　样机测试与性能分析

5.1　游戏菜单

如图 5.1 和图 5.2 所示，打开开发板电源后，首先显示的是游戏菜单，在菜单界面按上一页或者下一页按钮可以切换菜单选项。菜单选项分别是游戏开始、历史最高分、设置小鼠探头持续时间、设置进级分数条件。每当展示到最后一项菜单时，继续按下一页会从第一页重新开始。（以下图片均为 GIF 动画图片，其动画效果只能在电子版文档且 office 版本大于等于 2021 版的 word 中查看）

图 5.1 仿真切换菜单选项

图 5.2 实机切换菜单选项

5.2　在小鼠探头持续时间设置界面选择等级

如图 5.3 和图 5.4 所示，在小鼠探头持续时间设置界面，也就是标有 TL(time level)字符开头的界面。最左侧的数字在闪动，代表当前编辑的数值是等级数值。按下数值增大或者减小键能够增大或者减小等级，数码管右侧四位数实时显示当前所选等级的时间设置。

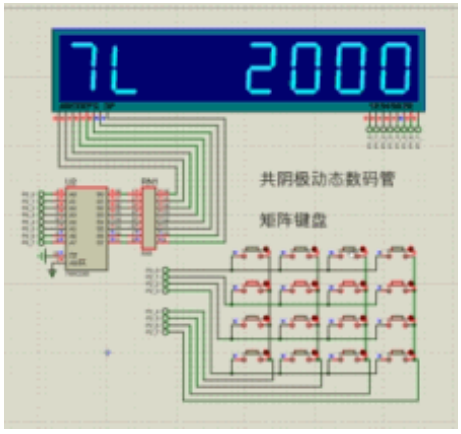


图 5.3 仿真在持续时间设置界面选择等级

图 5.4 实机在持续时间设置界面选择等级

5.3 在小鼠探头持续时间设置界面设置时间

如图 5.5 和图 5.6 所示，在小鼠探头持续时间设置界面，字符 TL 后跟的一位数显示当前正在编辑的等级，按下编辑下（上）一位键可以将当前闪烁的数值位置移动到下（上）一个位置，代表当前编辑的是该数值，使用数值增大（减小）键可以用于给当前闪烁的数值加（减）一。编辑实时保存，无需按确定键。时间的单位是毫秒。每当代表小鼠的 LED 灯亮了之后，必须经过相应等级设置的持续时间才会熄灭。

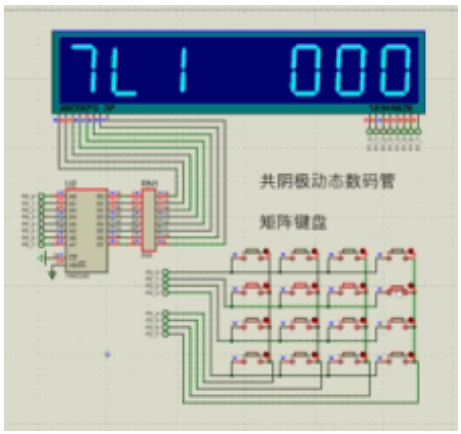


图 5.5 仿真在持续时间设置界面设置时间

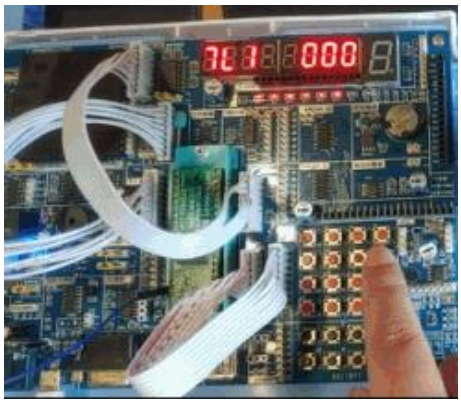


图 5.6 实机在持续时间设置界面设置时间

5.4 在进级分数条件设置界面选择等级

如图 5.7 和图 5.8 所示，在进级分数条件设置界面，也就是标有 SL(score level)字符开头的界面。最左侧的数字在闪动，代表当前编辑的数值是等级数值。按下数值增大或者减小键能够增大或者减小等级，数码管右侧四位数实时显示当前所选等级的时间设置。

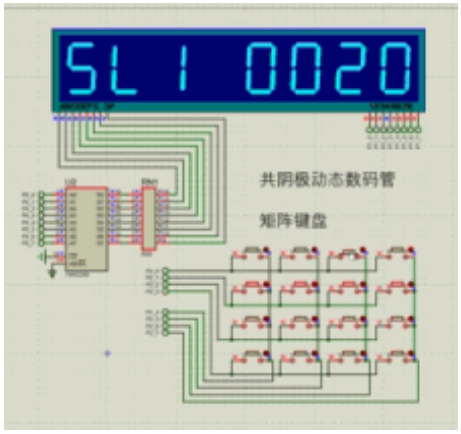


图 5.7 仿真在分数条件设置界面选择等级

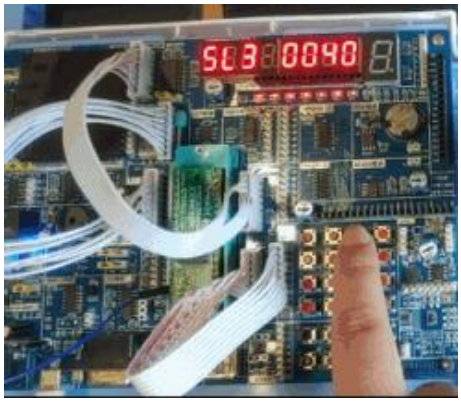


图 5.8 实机分数条件设置界面选择等级

5.5 在进级分数条件设置界面设置分数

如图 5.9 和图 5.10 所示，在进级分数条件设置界面，SL 后跟的一位数显示当前正在编辑的等级，按下编辑下（上）一位键可以将当前闪烁的数值位置移动到下（上）一个位置，代表当前编辑的是该数值，使用数值增大（减小）键可以用于给当前闪烁的数值加（减）一。编辑实时保存，无需按确定键。分数的单位是分。每当在该等级得分达到对应数值，就会进级到下一个等级。

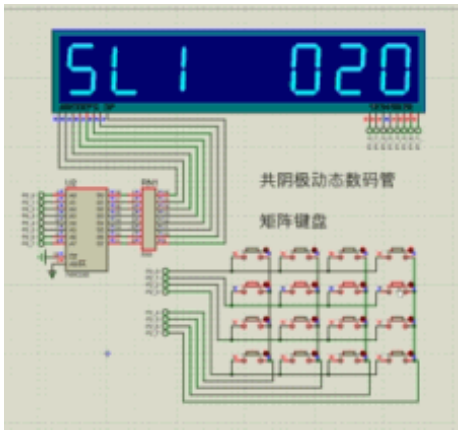


图 5.9 仿真分数条件设置界面设置分数

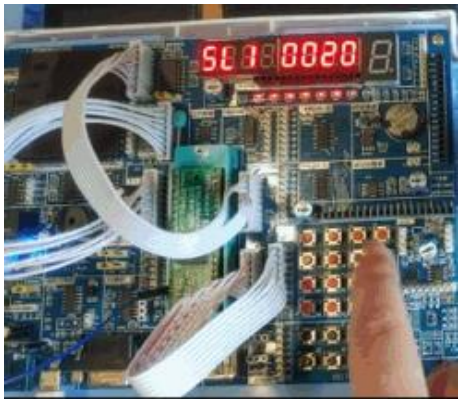


图 5.10 实机分数条件设置界面设置分数

5.6 游戏进行过程

如图 5.11 和图 5.12 所示，在游戏菜单的 GAME RUN 选项，按下确定键（位于数值增大的下面，数值减小的上面）即可开始运行游戏。游戏开始后，首先显示当前游戏等级，然后进入正式打地鼠环节，此时数码管第 0~1 位用于显示当前得分，数码管 2~3 位显示游戏倒计时，倒计时达到 0 结束游戏。数码管 4~7 位显示小鼠的位置。小鼠会随机在 300 毫秒到 700 毫秒内显示在随机 12 个位置之一。当按下对应按键时，蜂鸣器发出 Bingo 声，小鼠消失，玩家计 1 分，如果按下错误按键，蜂鸣器发出 Alert 声，玩家不计分。长达规定时间没有按下小鼠对应的按键，小鼠也会消失，玩家不计分。

图 5.11 时机游戏过程

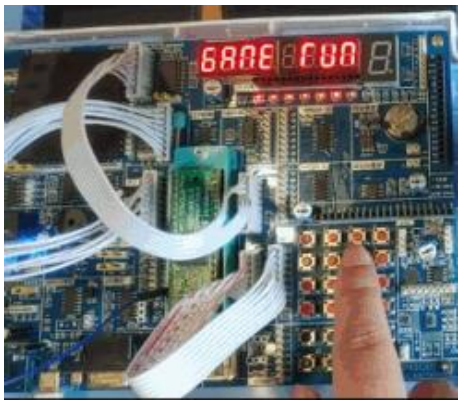


图 5.12 实机游戏进行过程

5.7 在游戏过程中临时查看 4 位数得分

如图 5.13 和图 5.14 所示，由于数码管的显示空间较小，只能容纳两位数用于显示得分，如果此时得分超过 99，就会导致得分数据无法正常显示出来，因此，我们添加了一个功能，在游戏进行过程中，按下查看当前等级得按键，游戏计时器消失，数码管的第 0~3 位共 4 位长度用于显示当前得分。

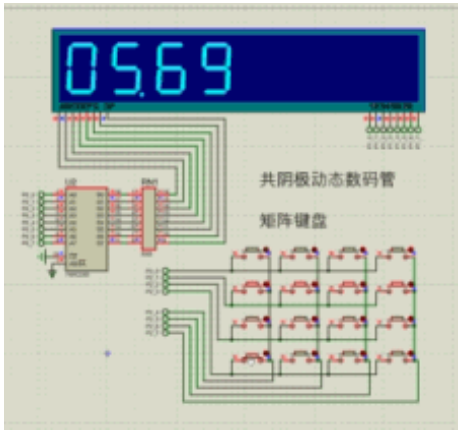


图 5.13 仿真在游戏过程中临时查看 4 位数得分

图 5.14 实机在游戏过程中临时查看 4 位数得分

5.8 游戏暂停

如图 5.15 和图 5.16 所示，在游戏过程中，点击暂停按键即可暂停游戏，此时游戏被暂停，并且数码管上闪烁着 PAUSE 字样代表暂停。在暂停时可以查看当前等级得分、游戏总得分（总得分为各个等级得分之和）、查看游戏总消耗时间继续游戏和退出游戏。

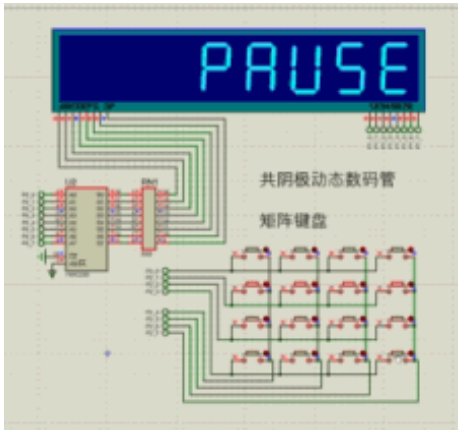


图 5.15 仿真游戏暂停界面

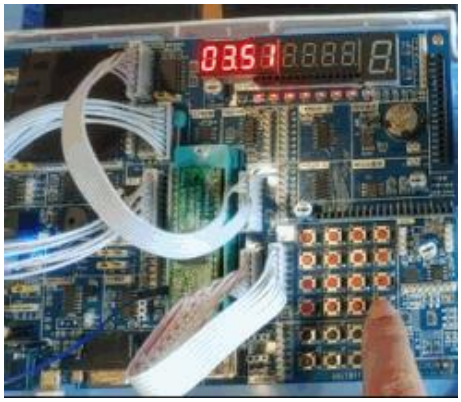


图 5.16 实机游戏暂停界面

如图 5.17 和图 5.18 所示，在暂停时按下退出游戏即可返回到游戏开始菜单，游戏得分清空，不与历史最高分做比较。

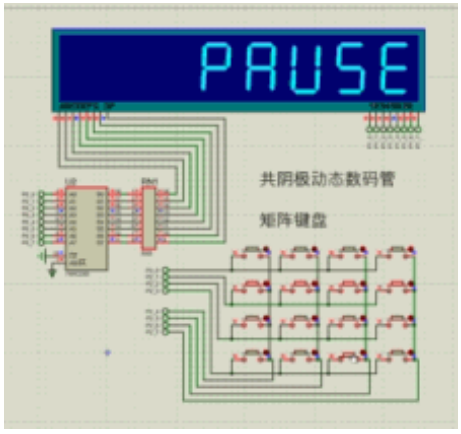


图 5.17 仿真在游戏暂停中退出游戏

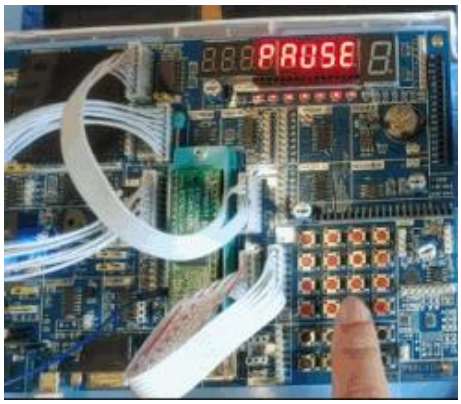


图 5.18 实机在游戏暂停中退出游戏

5.9 游戏进级

如图 5.19 和图 5.20 所示，当玩家得分到达规定的分值后，游戏将进级到下一个等级，此时数码管显示下一个等级值，并且游戏倒计时重新开始计数，小鼠探头持续时间按照规定减少，游戏难度增大。如图 5.20 所示演示的时在第 4 级进级到第 5 级的过程。

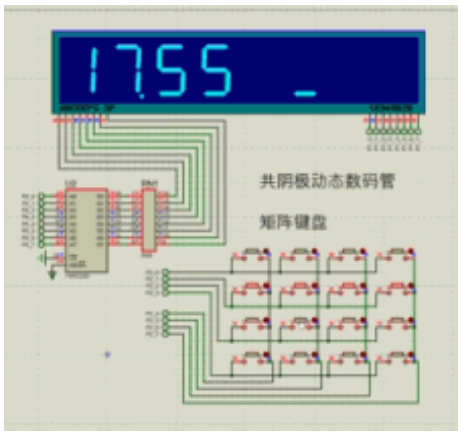


图 5.19 仿真游戏进级

图 5.20 实机游戏进级

5.10 游戏结束

如图 5.21 和图 5.22 所示，游戏倒计时为 0 时会触发游戏结束，此时数码管上显示 GAMEOVER，蜂鸣器响起游戏结束音乐。持续 5.5 秒后使用数字递增动画显示本次游戏获得的总分数。该界面左侧由字符 S 代表 score，在数字递增的过程中蜂鸣器发出金币碰撞的声音。按下任意键可以显示本次游戏总耗时，该界面左侧由字符 T 代表 time。

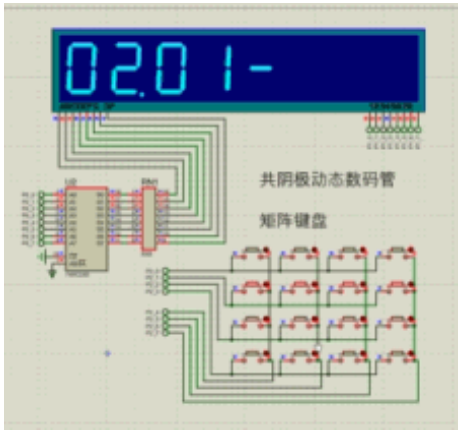


图 5.21 仿真游戏结束

图 5.22 实机游戏结束

5.11 游戏结束时返回游戏菜单

如图 5.23 和图 5.24 所示，在游戏结束的查看总耗时界面，按下任意键即可回到游戏开始菜单 GAME RUN

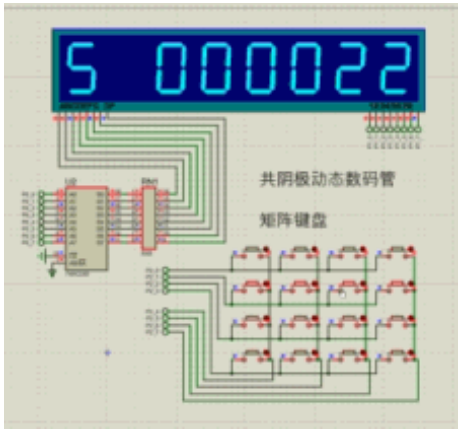


图 5.23 仿真游戏结束时返回游戏菜单

图 5.24 实机游戏结束时返回游戏菜单

5.12 查看历史最高分

如图 5.25 和图 5.26 所示，在游戏菜单界面点击下一页，知道有 TOP 字符标记的界面，右侧显示的是游戏历史最高分。可以看到目前仿真测试为 22 分，实机测试为 198 分。

图 5.25 仿真历史最高分

图 5.26 实机历史最高分

第6章 设计小结

通过本次单片机系统设计项目，我们学会了很多单片机的功能和编程技巧。俗话说兴趣是最好的老师，从立项开始，我们选择了比较有趣的打地鼠游戏设计，这使我们能够在兴趣的驱使下进行学习。按照任务要求，我们罗列了每一个需要完成的功能，幻想着游戏机能够有朝一日在开发板上运行的那一刻，那绝对是工科生最大的幸福。

做单片机系统设计首先遇到的难题是系统硬件究竟该如何选择。在硬件上我们按照题目要求选择了平时实验课上使用过的 8 位动态数码管作为显示输出、平时上课也做过实验的矩阵键盘和无源蜂鸣器。这些硬件我们曾经都把玩过不数一两次，看到此处我们信心大增。不由感慨着真是弱爆了。为了让生活充满未知和挑战性，创造困难迎难而上。在无源蜂鸣器的驱动方式上我们选择了从未尝试过的硬件脉宽调制信号（PWM）驱动。

唯一的 IAP15W4K58S4 开发板经组长从实验室签领后收入囊中，其余三位组员犹如巧妇难为无米之炊，除了在实验室合作的那点可怜的时间能使用开发板进行调试外，想要在课外时间私底下努力编写程序代码竟无处可调试。每当实验室关闭大门后的项目开发生活就像在战场上误入雷区一样举步维艰。为了消灭这个困难，我们使用了先进的计算机仿真技术，在电脑上安装了 Proteus 电路仿真软件，并依据着硬件端口线路设计和开发板的线路绘制了数字版本开发板。实现每一位组员的代码调试自由。

一个优秀的单片机软件需要层层包装、简单明了。在软件方案上我们学习了很多优秀的编程思想，例如多任务、模块化、软编码、状态机、详细注释等等数不胜数。我们胸怀大志，将硬件使用的引脚独立封装到一个头文件内，为将来代码的跨平台移植做好打量。把各个硬件运行的逻辑程序编写为各个模块，并提供简单易用的应用程序接口。这样就形成了驱动程序。不仅能让每一位小组成员可以专心致志实现自己的模块，还可以实现在编写游戏时丝毫不用在乎硬件究竟如何运行。只要调用驱动模块提供的应用程序接口，就可以在单片机上为所欲为，所有的功能终究都会水到渠成。

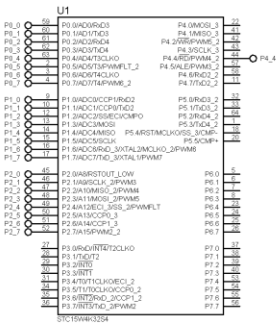
我们设计的游戏有能够令人眼花缭乱的菜单界面，有令人欲罢不能的游戏过程，还有令人叹为观止的动画效果！将游戏循环的流程图挥之而出，不经让人惊掉下巴：这一个个节点如沙哈拉之沙粒，节点之间的连线如魔鬼般布下天罗地网、错综复杂。想要把这种代码写出来简直就是大海捞针。我们不禁想起了数字电子技术课上学习过的状态机，使用这一思想我们就能将混乱不堪的游戏循环归类为 10 大状态，在每种状态只需要执行简单的逻辑，在特定的条件下能够切换到另一个状态。使游戏循环逻辑大大被简化、降低了游戏开发难度。

虽然开发之路道阻且长，我们披荆斩棘、艰难困苦，但在百折不挠的努力和很多先进编程技巧的驱使下，我们奋发图强、永不言弃，逐个击破了单片机系统设计之路上的所有困难，不用几天就彻底完成了程序代码的编写。并且在实机开发板上调试取得了圆满的结果。

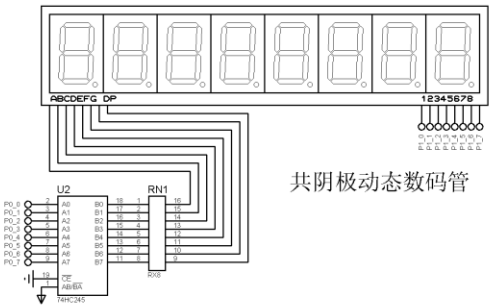
参考文献

[1] 覃丽姗,李宁,王恬灏.基于单片机的打地鼠游戏设计[J].山东工业技术,2018(16):130.
[2] 程崇源. 高刷多行扫 LED 显示控制芯片的设计与实现[D].华中科技大学,2020.
[3] 王苏.直流电机 PWM 调速研究及单片机控制实现[J].机电工程技术,2008(11):82-84+95+110.
[4] 纪瀚涛,王伟.单片机驱动蜂鸣器的程序设计[J].时代农机,2019,46(02):125.
[5] 武志鹏,焦红卫.一种基于矩阵键盘扫描原理的程序设计[J].工业控制计算机,2018,31(07):141-142.
[6] 沈俊慧,朱其祥.高性价比的 LED 驱动及按键扫描电路与算法设计——以 8 段数码管显示驱动控制电路为例[J].福建商学院学报,2023(06):59-67.
[7] 符潇天,黄明,彭召敏,等.基于定时中断的实时按键任务处理设计[J].工业技术创新,2018,05(02):24-28.
[8] 谭同超. 有限状态机及其应用[D].华南理工大学,2014.
[9] 刘春玲,熊馨,董晓庆,等.基于时序的单片机多任务系统驱动策略[J].单片机与嵌入式系统应用,2023,23(03):84-87+91.
[10] 王孝平,张建兵.基于单片机的智能玩具小车控制软件的设计与实现[J].科学技术创新,2024,(14):215-218.
[11] 李红刚,张素萍.基于单片机和 LabVIEW 的多路数据采集系统设计[J].国外电子测量技术,2014,33(04):62-67.DOI:10.19652/j.cnki.femt.2014.04.017.
[12] 曹宇,魏丰,胡士毅.用 51 单片机控制 RTL8019AS 实现以太网通讯[J].电子技术应用,2003,(01):21-23.

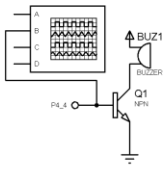
附录 1　系统原理图



单片机芯片

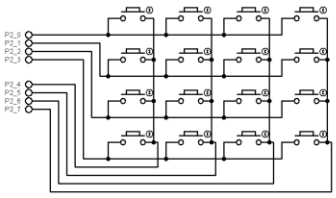


共阴极动态数码管



蜂鸣器(无仿真模型，所以不响)

矩阵键盘



附录 2　程序清单

文件名：pin.h
功能：管理单片机用到的引脚

```
1.  #ifndef PIN_H
2.  #define PIN_H
3.  //定义常用引脚名称
4.
5.  //蜂鸣器
6.  //本次程序操作 PWM4 输出信号到 P4_4 引脚驱动无源蜂鸣器
7.  //不直接操作 io 口，具体声明请看 buzzer.c 中的 Buzzer_Init()
8.  #define PIN_BUZZER            P4_4 //蜂鸣器
9.
10. #define PORT_DISPLAY_DATA     P0   //数码管数据线端口
11. #define PORT_DISPLAY_SELECT   P1   //数码管位选线端口
12. #define PORT_KEYBOARD         P2   //矩阵键盘端口
13.
```

```
#endif
```

文件名：framework.h
功能：时分多任务框架的声明文件

```
//使用 if 判断预定义，防止头文件因为 include 的
//复杂关系偶然出现执行定义函数的情况
#ifndef FRAMEWORK_H
#define FRAMEWORK_H

#include "STC15.h"

//定义这些类型方便使用
#define uint unsigned int
#define uchar unsigned char

extern unsigned long int Uptime_Seconds;
extern uchar Loop_Time;

//Loop_Time 变量存储当前的时间周期
//用于定制操作的依据
//目前只用了低 4 位来储存 4 种情况，还有高四位可以后续拓展
//以下是 Loop_Time 可能的值
#define TIME_2MS   0x01    //二进制是 0000 0001，代表当前 2ms 周期已到
#define TIME_10MS  0x02    //二进制是 0000 0010，代表当前 10ms 周期已到
#define TIME_500MS 0x04    //二进制是 0000 0100，代表当前 500ms 周期已到
#define TIME_1S    0x08    //二进制是 0000 1000，代表当前 1s 周期已到

void TIME_1S_FUNCTION();
void TIME_500MS_FUNCTION();
void TIME_10MS_FUNCTION();
void TIME_2MS_FUNCTION();

//定义一个延时宏定义，不阻塞代码运行
//延时时间等于当前函数执行周期的倍数
//如果在一个函数内要使用这个延时宏定义
//首先按照符合 c89 标准要求在函数开始定义变量
//必须使用 USE_TIMEOUT 宏在函数开始处定义局部变量
//在需要延时处使用如下语句，延时时间以当前函数的调用周期而定
//SETTIMEOUT(延时倍数){延时执行的语句}
//如果当前函数每 2 秒执行一次，那么 SETTIMEOUT(4)就是每 8 秒执行一次
//延时倍数最大值由 time_out_count 类型决定，目前是 uint。延时倍数必须是整数
#define USE_TIMEOUT static uint time_out_count = 0;time_out_count++
#define SETTIMEOUT(num) if(time_out_count % num == 0)
```

```
40.
41. #endif
```

文件名：framework.c
功能：时分多任务框架的实现文件

```
1.  #include "framework.h"
2.  #include "buzzer.h"
3.  #include "display.h"
4.
5.  //定义系统运行时间变量
6.  //系统每一次启动从 0 开始计数
7.  //每秒增加一个数值
8.  //使用 uint 类型只有 2 字节，只能保存 18 小时的秒数大小
9.  //而 ulint 类型有 4 字节，可以保存 136 年的秒数大小
10. //当前把储存秒数改为储存毫秒数，以上计算的时间都除以 1000
11. //储存毫秒数用于给随机函数提供种子值
12. //使用毫秒提供随机数种子能让随机数看起来更随机
13. unsigned long int Uptime_Seconds = 0;
14.
15.
16. //如果要给 Loop_Time 登记 0000 0001，2ms 周期
17. //只需要 Loop_Time 与 TIME_2MS 按位或赋值
18. //需要判断当时 2ms 周期是否已到
19. //只需要 Loop_Time 与 TIME_2MS 按位与，如果结果不为 0 那么说明条件符合
20. uchar Loop_Time = 0x00;
21.
22. //初始化系统运行定时器，每毫秒执行一次中断函数
23. void System_Timer_0_Init() {
24.     AUXR |= 0x80;           //定时器时钟 1T 模式
25.     TMOD &= 0xF0;           //设置定时器模式
26.     TL0 = 0xCD;             //设置定时初始值
27.     TH0 = 0xD4;             //设置定时初始值
28.     TF0 = 0;                //清除 TF0 标志
29.     TR0 = 1;                //定时器 0 开始计时
30.     ET0 = 1;                //使能定时器 0 中断
31.     EA = 1;                 //打开使能总中断
32. }
33.
34. //系统定时器 0 执行的中断函数
35. //每隔一毫秒就给 Uptime_Seconds 加 1
36. //用来表示系统运行的毫秒数
37. //为给 Loop_Time 赋值标记当前周期的功能
38. //Loop_Time 可能的取值请看 framework.h
```

```
39. void System_Timer_0() interrupt 1 {
40.     static uint i=0;
41.     i++;
42.     Uptime_Seconds++;
43.     //把 Refresh_Display_Hook();写在这里是为了
44.     //提高刷新数码管的优先权
45.     //之前写在 task.c 里面
46.     //如果其他任务过多会造成代码阻塞
47.     //数码管刷新速度受到影响
48.     //数码管刷新速度只要轻微降低
49.     //就能让人明显感知到数码管在闪速
50.     if(i%2){
51.         (Loop_Time|=TIME_2MS);
52.         Refresh_Display_Hook();
53.     }
54.     //i%2     || (Loop_Time|=TIME_2MS);
55.     i%10  || (Loop_Time|=TIME_10MS);
56.     i%500 || (Loop_Time|=TIME_500MS);
57.
58.     if(i==1000){
59.         i = 0;
60.         Loop_Time|=TIME_1S;
61.     }
62.
63. }
64. //初始化端口的工作模式
65. void Port_Init(){
66.     P0M1 = 0x00;P0M0 = 0x00;
67.     P1M1 = 0x00;P1M0 = 0x00;
68.     P2M1 = 0x00;P2M0 = 0x00;
69. }
70.
71.
72. void main(){
73.     //开机只会执行一次的操作
74.     Port_Init();
75.     Buzzer_Init();
76.     System_Timer_0_Init();
77.     //开机后重复执行的操作
78.     while(1)
79.     {
80.         if(Loop_Time&TIME_1S)TIME_1S_FUNCTION();//如果 1s 周期已到，那就执行
    TIME_1S_FUNCTION 函数；
```

```
        if(Loop_Time&TIME_500MS)TIME_500MS_FUNCTION();
        if(Loop_Time&TIME_10MS)TIME_10MS_FUNCTION();
        if(Loop_Time&TIME_2MS)TIME_2MS_FUNCTION();//如果 2ms 周期已到，那就执行 TIME_2MS_FUNCTION 函数；
    }
}
```

文件：display.h
功能：数码管驱动的声明文件

```
#ifndef DISPLAY_H
#define DISPLAY_H
#include "framework.h"

//第 0 个数码管显示数字示例(数值可动态变化)：
//Display_Memory[0] = NUMBER(1)
#define NUMBER(num) font_table[num]


//第 0 个数码管显示字母示例：
//Display_Memory[0] = LETTER_C
#define LETTER_A         0x77
#define LETTER_B         0x7c
#define LETTER_C         0x39
#define LETTER_D         0x5e
#define LETTER_E         0x79
#define LETTER_F         0x71
#define LETTER_G         0x7d
#define LETTER_T         0x07
#define LETTER_O         0x3F
#define LETTER_U         0x3E
#define LETTER_R         0x31
#define LETTER_N         0x37
#define LETTER_P         0x73
#define LETTER_L         0x38
#define LETTER_S         0x6d
#define DISPLAY_OFF      0x00
#define DISPLAY_ON       0xFF


extern uchar xdata font_table[];

extern uchar Display_Memory[];

```

```
void Refresh_Display_Hook();
void Display_Show(uchar d0,uchar d1,uchar d2,uchar d3,uchar d4,uchar d5,uchar d6
,uchar d7);
void Display_Show_Number(long int num,uchar begin,uchar bits);
#endif
```

文件：display.c
功能：数码管驱动的实现文件

```
#include "display.h"
#include "pin.h"

//定义数码管显示每一个字的字形码
uchar xdata font_table[]=
    {
    0x3f,0x06,0x5b,0x4f,// 0  1  2  3
    0x66,0x6d,0x7d,0x07,// 4  5  6  7
    0x7f,0x6f,0x77,0x7c,// 8  9  A  B
    0x39,0x5e,0x79,0x71 // C  D  E  F
    };


//定义数码管显示器内存，
//如果需要修改显示内容，只需要改此变量的内容
//示例，修改第 0 个数码管为数字 2:
//Display_Memory[0]=NUMBER(2);
uchar Display_Memory[] =
    {
    DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF, //数码管第 0~3 位初始内容
    DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF  //数码管第 4~7 位初始内容
    };


//刷新数码管内容函数
//该函数每 2ms 执行一次
//从显示器内存中获取当前需要显示的内容
//使用动态刷新的方法显示在数码管上
void Refresh_Display_Hook(){
    static uchar i=0;

    PORT_DISPLAY_SELECT = ~(0x01<<i);//切换当前工作的数码管
    PORT_DISPLAY_DATA=Display_Memory[i];//将显示内存数据依次发送到数码管数据线上
    if(++i>7)i=0;
```

```
}

//数码管显示八个内容
//参数分别是数码管从
//第 0 位到第 7 位的内容
//示例：
//Display_Show(LETTER_G,LETTER_A,LETTER_N,LETTER_E,DISPLAY_OFF,LETTER_R,LETTER_U,LETTER_N);
void Display_Show(uchar d0,uchar d1,uchar d2,uchar d3,uchar d4,uchar d5,uchar d6,uchar d7){
    Display_Memory[0] = d0;
    Display_Memory[1] = d1;
    Display_Memory[2] = d2;
    Display_Memory[3] = d3;
    Display_Memory[4] = d4;
    Display_Memory[5] = d5;
    Display_Memory[6] = d6;
    Display_Memory[7] = d7;
}

//数码管显示数字
//参数：
//num：需要显示的数字
//begin：个位所在数码管的位置
//bits：要显示的数字的位数
void Display_Show_Number(long int num,uchar begin,uchar bits){
    while (bits--)
    {
        Display_Memory[begin--] = NUMBER(num%10);
        num /= 10;
    }

}
```

文件名：keyboard.h
功能：矩阵键盘驱动的声明文件

```
#ifndef KEYBOARD_H
#define KEYBOARD_H

#include "framework.h"

//矩阵键盘的键值码
#define KEY_1       0xEE
```

```
#define KEY_2       0xDE
#define KEY_3       0xBE
#define KEY_4       0x7E

#define KEY_5       0xED
#define KEY_6       0xDD
#define KEY_7       0xBD
#define KEY_8       0x7D

#define KEY_9       0xEB
#define KEY_10      0xDB
#define KEY_11      0xBB
#define KEY_12      0x7B

#define KEY_13      0xE7
#define KEY_14      0xD7
#define KEY_15      0xB7
#define KEY_16      0x77

#define KEY_NULL    0xFF

//按键使用示例：
//按下按键时执行一次代码
//KEY_DOWN(键值){
//    需要执行的代码;
//}
#define KEY_DOWN(value) for(;Key_Down_Value == value;Key_Down_Value = KEY_NULL)

//抬起按键时执行一次代码
//KEY_UP(键值){
//    需要执行的代码;
//}
#define KEY_UP(value) for(;Key_Up_Value == value;Key_Up_Value = KEY_NULL)


//非必要情况下不需要读取以下键值
//只需要按照上方宏定义的用法使用就好

extern uchar Key_Value;
extern uchar Key_Down_Value;
extern uchar Key_Up_Value;

```

```
//按键使用示例：
//按下任意按键时执行一次代码
//如果需要读取键值，请读取全局变量 Key_Value
//ANY_KEY_DOWN{
//    需要执行的代码;
//}
#define ANY_KEY_DOWN for(;Key_Down_Value != KEY_NULL;Key_Down_Value = KEY_NULL)


void Key_Scan();
void Test_Key();
void Decode_Key_Event();
#endif
```

文件名：keyboard.c
功能：矩阵键盘驱动的实现文件

```
#include "keyboard.h"
#include "pin.h"
#include "display.h"
#include "buzzer.h"
#include "random.h"

//当前的实时键值
uchar Key_Value = KEY_NULL;

//当前已按下的键值
uchar Key_Down_Value = KEY_NULL;

//当前已抬起的键值
uchar Key_Up_Value = KEY_NULL;

//扫描矩阵键盘按下的键
//该函数每 10ms 执行一次
//每执行两次求出键值
void Key_Scan(){
    static uchar State = 0;
    static uchar Value;
    uchar temp;
    if(State){
        Value = 0x00;
        Value |= PORT_KEYBOARD;
        State = 0;
```

```
        PORT_KEYBOARD = 0xF0;
    }else{
        Value |= PORT_KEYBOARD;
        //判断当 Value 是一个有效的键值时
        //才将它写入 Key_Value 变量中
        //有效的值包括
        //1、按下时包含两个 0 的八位二进制数
        //2、没有按下按键 KEY_NULL
        temp = ~Value;
        temp &= temp-1;
        //有时候因为按下按键的时机问题获得无效的键值
        //这时候获得的键值是有可能是包含 1 个 0 的八位二进制数
        //temp 给键值取反会得到包含 1 个 1 的八位二进制数
        //使用与运算去掉一个 1，如果此时结果是 0
        //证明该键值是无效的，应舍去
        //否则是有效的，为 Key_Value 赋值该键值
        if(temp!=0 || Value==KEY_NULL)Key_Value = Value;

        //解析按键事件的函数
        //只能在这个 if 分支里面生效
        //所以就放在了这个分支里
        //放在分支外工作正常，但是会多浪费一次算力
        Decode_Key_Event();
        State = 1;
        PORT_KEYBOARD = 0x0F;
    }
}

//解析按键事件
void Decode_Key_Event(){
    static uchar Prev_Value=KEY_NULL;
    //如果上次的键值不等于这次的键值
    //这次键值为空的话，那么就是抬起了按键
    //否则是按下了按键
    if(Prev_Value!=Key_Value){
        if(Key_Value==KEY_NULL){
            Key_Up_Value = Prev_Value;
        }else{
            Key_Down_Value = Key_Value;
        }
        Prev_Value = Key_Value;
    }
}
```

```
70.
71. //测试按键的函数
72. //同时也是按键使用的示例代码
73. void Test_Key(){
74.     static uchar i = 1;
75.     static uchar j = 1;
76.     if(i>15)i=1;
77.     if(j>15)j=1;
78.
79.     KEY_UP(KEY_1){
80.         Display_Memory[0]=NUMBER(i++);
81.     }
82.     KEY_DOWN(KEY_1){
83.         Display_Memory[3]=NUMBER(j++);
84.     }
85.
86.     KEY_DOWN(KEY_5){
87.         //生成一个三位数随机数
88.         uint random_value = random(0,999);
89.         Display_Memory[7]=NUMBER(random_value%10);//个位
90.         random_value /= 10;
91.         Display_Memory[6]=NUMBER(random_value%10);//十位
92.         random_value /= 10;
93.         Display_Memory[5]=NUMBER(random_value%10);//百位
94.     }
95.
96.     KEY_DOWN(KEY_4){
97.         Buzzer_Beep();
98.     }
99.     KEY_UP(KEY_4){
100.         Buzzer_noBeep();
101.       }
102.
103.      KEY_DOWN(KEY_13){
104.          Buzzer_Tone(DO);
105.      }
106.      KEY_UP(KEY_13){
107.          Buzzer_noBeep();
108.      }
109.
110.      KEY_DOWN(KEY_14){
111.          Buzzer_Tone(RE);
112.      }
```

```
113.       KEY_UP(KEY_14){
114.           Buzzer_noBeep();
115.       }
116.
117.       KEY_DOWN(KEY_15){
118.           Buzzer_Tone(MI);
119.       }
120.       KEY_UP(KEY_15){
121.           Buzzer_noBeep();
122.       }
123.
124.       KEY_DOWN(KEY_16){
125.           Buzzer_Tone(FA);
126.       }
127.       KEY_UP(KEY_16){
128.           Buzzer_noBeep();
129.       }
130.
131.       KEY_DOWN(KEY_8){
132.           Buzzer_Tone(_RE);
133.       }
134.       KEY_UP(KEY_8){
135.           Buzzer_noBeep();
136.       }
137.
138.       KEY_DOWN(KEY_12){
139.           Buzzer_Is_Alert=1;
140.       }
141.       KEY_DOWN(KEY_10){
142.           Buzzer_Is_Bingo=1;
143.       }
144.
145.       KEY_DOWN(KEY_9){
146.           Buzzer_Player_Play(&Muisc_Gameover);
147.       }
148.       KEY_DOWN(KEY_11){
149.           Buzzer_Player_Stop();
150.       }
151.   }
```

文件名：buzzer.h
功能：蜂鸣器的驱动声明文件

```
1.  #ifndef BUZZER_H
```

```
2.  #define BUZZER_H
3.  #include "framework.h"
4.
5.  //该频率定义是数字为
6.  //该周期在 Tone_Freq[]数组中的顺序位置
7.  //TONE_1KHZ 的周期在 Tone_Freq[41]内，为 11059
8.  #define TONE_1KHZ   41  //11059us
9.  #define TONE_ALERT  42  //169hz
10.
11. void Buzzer_Init();
12. void Buzzer_Beep();
13. void Buzzer_noBeep();
14. void Buzzer_Tone(uchar tone);
15. void Refresh_Buzzer_Hook();
16. void Buzzer_Player_Hook();
17.
18. //蜂鸣器警告声的开关
19. //当设置 Buzzer_Is_Alert=1 时警报打开
20. extern uchar Buzzer_Is_Alert;
21.
22. extern uchar Buzzer_Is_Bingo;
23.
24. extern uint code Tone_Freq[];
25.
26. //播放地址索引
27. extern long int Player_Ps;
28. //当前播放音符列表索引
29. extern uchar *Player_Note_List;
30. extern uchar Player_State;
31.
32. extern uchar code Muisc_Gameover[];
33.
34. void Buzzer_Alert();
35. void Buzzer_Bingo();
36. void Buzzer_Player_Play(uchar *note_list);
37. void Buzzer_Player_Stop();
38.
39.
40. //宏定义，音调唱名转换为
41. //该音调的周期在 Tone_Freq[]数组中的顺序位置
42. //这样用来编谱只需要 uchar 类型保存顺序，减少储存占用
43. //唱名+S 代表升半阶音节
44.
```

```
45. //中音谱
46. #define DO      0  //1
47. #define DOS     1  //1#
48. #define RE      2  //2
49. #define RES     3  //2#
50. #define MI      4
51. #define FA      5
52. #define FAS     6
53. #define SO      7
54. #define SOS     8
55. #define LA      9
56. #define LAS     10
57. #define SI      11
58. //低音谱
59. #define _DO     12
60. #define _DOS    13
61. #define _RE     14
62. #define _RES    15
63. #define _MI     16
64. #define _FA     17
65. #define _FAS    18
66. #define _SO     19
67. #define _SOS    20
68. #define _LA     21
69. #define _LAS    22
70. #define _SI     23
71. //高音谱
72. #define DO_     24
73. #define DOS_    25
74. #define RE_     26
75. #define RES_    27
76. #define MI_     28
77. #define FA_     29
78. #define FAS_    30
79. #define SO_     31
80. #define SOS_    32
81. #define LA_     33
82. #define LAS_    34
83. #define SI_     35
84. //静音音符
85. #define MUTE    36
86. #define FINISH  254
87.
```

```
#define PLAYER_PALY     1
#define PLAYER_STOP     2

#endif
```

文件名：buzzer.c
功能：蜂鸣器的驱动实现文件

```
#include "buzzer.h"


//data：固定指前面 0x00-0x7F 的 128 个 RAM，可以用 acc 直接读写，速度最快，生成的代码也最小。
//idata：固定指前面 0x00-0xFF 的 256 个 RAM，其中前 128 和 data 的 128 完全相同，只是访问的方式不同。
//xdata：外部扩展 RAM。
//code：ROM。
//================================
//这个变量是不会在运行中改变的，为了节约内存
//所以把这个数组变量直接定义在 code(ROM)内
//音符周期(us) = 1khz 周期*1000/音符频率
uint code Tone_Freq[] = {
    //数组 0~11 位：中音 1、1#、2、2#、3、4、4#、5、5#、6、6#、7
    21145,19962,18840,17780,16781,15844,14945,14106,13308,12567,11866,11193,
    //数组 12~23：低音 1、1#、2、2#、3、4、4#、5、5#、6、6#、7
    42210,39924,37616,35559,33512,31688,29889,28212,26648,25134,23732,22387,
    //数组 24~35：高音 1、1#、2、2#、3、4、4#、5、5#、6、6#、7
    10573,9972,9412,8883,8391,7916,7472,7053,6658,6284,5930,5597,
    //数组 36~40：音符频率和其他类型音频频率的分界线
    0,0,0,0,0,
    //数组 41~42：TONE_1KHZ,TONE_ALERT
    11059,65534
};

uchar code Muisc_Gameover[] = {
    DO_,16,MUTE,30,SO,13,MUTE,30,MI,30,
    LA,20,SI,20,LA,20,
    SO,30,LA,30,SO,30,
    MI,10,RE,13,MI,63,
    FINISH
};

//初始化蜂鸣器引脚和相关 PWM 配置
//默认配置 1khz 频率，50 占空比
```

```
35. void Buzzer_Init(){
36.     P4M1 = 0x00;P4M0 = 0x00;
37.
38.     P_SW2 |= 0x80;                  //打开访问 PWM 特殊功能寄存器的权限
39.     PWMC = 11059;                   //设置 PWM 周期为 1ms
40.     PWM4T1 = 0;
41.     PWM4T2 = 5529;                  //设置 PWM4 第 2 次反转的 PWM 计数，占空比 50
42.     PWM4CR = 0x08;                  //选择 PWM4 输出到 P4.4
43.     PWMCR = 0x00;                   //设置 0x84 可打开 PWM4 信号输出
44.     P_SW2 &= ~0x80;
45. }
46.
47. //蜂鸣器响起嘟嘟声
48. void Buzzer_Beep(){
49.     P_SW2 |= 0x80;
50.     PWMC = Tone_Freq[TONE_1KHZ];
51.     PWM4T1 = 0;
52.     PWM4T2 = Tone_Freq[TONE_1KHZ]/2;
53.     PWMCR = 0x84;
54.     P_SW2 &= ~0x80;
55. }
56.
57. //蜂鸣器静音
58. void Buzzer_noBeep(){
59.     P_SW2 |= 0x80;
60.     PWMCR = 0x00;
61.     P_SW2 &= ~0x80;
62. }
63.
64. //蜂鸣器播放含有音高的声音
65. //参数是音符在 Tone_Freq[]数组中的顺序位置
66. //示例：Buzzer_Tone(DO);
67. //其中 DO 是顺序位置 0
68. void Buzzer_Tone(uchar tone){
69.     P_SW2 |= 0x80;
70.     PWMC = Tone_Freq[tone];
71.     PWM4T1 = 0;
72.     PWM4T2 = Tone_Freq[tone]/2;
73.     if(tone == MUTE){
74.         PWMCR = 0x00;
75.     }else{
76.         PWMCR = 0x84;
77.     }
```

```
    P_SW2 &= ~0x80;
}

//设置为 1，蜂鸣器响起一次警告
//结束自动归零
uchar Buzzer_Is_Alert = 0;
//设置为 1，蜂鸣器响起一次得分提示音
//结束自动归零
uchar Buzzer_Is_Bingo = 0;

//刷新蜂鸣器状态的函数
//该函数每 10ms 执行一次
void Refresh_Buzzer_Hook(){
    if(Buzzer_Is_Alert)Buzzer_Alert();
    if(Buzzer_Is_Bingo)Buzzer_Bingo();
    if(!Buzzer_Is_Alert && !Buzzer_Is_Bingo && Player_State == PLAYER_PALY)Buzzer_Player_Hook();
}

//如果 Buzzer_Is_Alert=1 就会执行此函数
//此函数执行的时间间隔由 Refresh_Buzzer_Hook 决定
void Buzzer_Alert(){
    static uchar i =0;
      if(i==0)Buzzer_Tone(TONE_ALERT);
      if(i==10)Buzzer_noBeep();
      if(i==20)Buzzer_Tone(TONE_ALERT);
      if(i==30){
          Buzzer_noBeep();
          i=0;
          Buzzer_Is_Alert=0;
          return;
      }
      i++;
  }

  //如果 Buzzer_Is_Bingo=1 就会执行此函数
  //此函数执行的时间间隔由 Refresh_Buzzer_Hook 决定
  void Buzzer_Bingo(){
      static uchar i =0;
      if(i==0)Buzzer_Tone(SI);
      if(i==6)Buzzer_Tone(MI_);
      if(i==36){
          Buzzer_noBeep();
```

```
            i=0;
            Buzzer_Is_Bingo=0;
            return;
        }
        i++;
    }

    uchar *Player_Note_List=0;
    uchar Player_State = PLAYER_STOP;
    long int Player_Ps = 0;

    //播放器的刷新函数
    //在播放状态每 10ms 执行一次
    void Buzzer_Player_Hook(){
        static uchar count = 0;
        uchar tone = Player_Note_List[Player_Ps];
        //如果当前的音调是播放完成标志，那么停止播放播放
        if(tone==FINISH){
            Buzzer_Player_Stop();
            return;
        }
        //如果当前的播放计数是 0，那么该音符第一次进入播放器循环
        //播放该音符
        if(count==0){
            Buzzer_Tone(tone);
        }
        //如果当前计数次数大于音符列表限制的循环次数
        //那么结束当前音符，将播放音符位置移动到下一个音符
        if(++count>Player_Note_List[Player_Ps+1]){
            Player_Ps+=2;
            count=0;
            return;
        }
    }

    //播放器开始播放音乐
    //参数是音乐音符数组指针
    //示例：Buzzer_Player_Play(&Muisc_Gameover);
    void Buzzer_Player_Play(uchar *note_list){
        Player_Note_List = note_list;
        Player_Ps = 0;
        Player_State = PLAYER_PALY;
    }
```

```
163.
164.    //停止音乐播放器的播放
165.    void Buzzer_Player_Stop(){
166.        Player_State=PLAYER_STOP;
167.        Player_Ps=0;
168.        Buzzer_noBeep();
169.    }
```

文件名：random.h
功能：随机数发生器的声明文件

```
1.  #ifndef RANDOM_H
2.  #define RANDOM_H
3.
4.  #include "stdlib.h"
5.  #include "framework.h"
6.
7.  uint random(uint min,uint max);
8.
9.  #endif
```

文件名：random.h
功能：随机数发生器的实现文件

```
1.  #include "random.h"
2.
3.  //生成随机数字的函数
4.  //输入参数是最小值和最大值
5.  //random(最小值，最大值);
6.  //返回 uint 类型数字
7.  uint random(uint min,uint max) {
8.      uint value;
9.      srand(Uptime_Seconds);
10.     if(max<200){
11.         //当数值过小时，随机数就看起来不太随机
12.         //这里通过增大数值范围来添加随机的效果
13.         max *=100;
14.         min *=100;
15.         value = rand() % (max + 1 - min) + min;
16.         return value/100;
17.     }
18.     value = rand() % (max + 1 - min) + min;
19.     return value;
20. }
```

文件名：game.h

功能：游戏循环的声明文件

```
#ifndef GAME_H
#define GAME_H
#include "framework.h"
#include "keyboard.h"

extern unsigned long int xdata Game_Time_Level[];
extern uint xdata Game_Score_Level[];


//游戏菜单状态值
#define GAME_STATE_MENU_RUN             1
#define GAME_STATE_MENU_TOP             2
#define GAME_STATE_MENU_TL              3  //设置小鼠持续时间
#define GAME_STATE_MENU_SL              4  //设置晋级分数
//记录着菜单里最大的值
#define MENU_MAX_NUMBER                 4

//游戏非菜单状态值
#define GAME_STATE_PLAYING              41
#define GAME_STATE_PAUSE                42
#define GAME_STATE_OVER                 43
#define GAME_STATE_SHOW_TOTAL_SCORE     44
#define GAME_STATE_SHOW_TOTAL_TIME      45
#define GAME_STATE_NEXT_LEVEL           46


//用于在给某些固定的数组求长度
//规定该数组的第 0 位手动输入数组长度
//有效内容位从 1 开始
#define LENGTH                          0

void Switch_Page_Key_Event();
void Refresh_Game_Hook();
void Game_Menu_Run_Hook();
void Game_Menu_Top_Hook();
void Game_Menu_TL_Hook();
void Game_Menu_SL_Hook();
void Game_Timer_Hook();
void Game_Playing_Hook();
void Show_Mole(uchar location);
void Hide_Mole();
```

```
43. void Game_Over_Hook();
44. void Show_Total_Score_Hook();
45. void Next_Level_Hook();
46. void Show_Total_Time_Hook();
47. void Game_Pause_Hook();
48.
49. //当处于菜单页面的按键映射关系
50. #define KEY_ADD                         KEY_3
51. #define KEY_MINUS                       KEY_11
52. #define KEY_PREV_BIT                    KEY_6
53. #define KEY_NEXT_BIT                    KEY_8
54. #define KEY_OK                          KEY_7
55. #define KEY_PREV_PAGE                   KEY_10
56. #define KEY_NEXT_PAGE                   KEY_12
57.
58. //当处于游戏或者暂停中的按键映射关系
59. #define KEY_PAUSE                       KEY_16
60. #define KEY_CONTINUE                    KEY_16
61. #define KEY_EXIT                        KEY_15
62. #define KEY_LEVEL_SCORE                 KEY_13
63. #define KEY_TOTAL_SCORE                 KEY_9
64. #define KEY_TOTAL_TIME                  KEY_10
65.
66.
67. //游戏每一个等级倒计时的时间
68. #define GAME_TIMER_VALUE                70
69.
70. //小鼠消失状态
71. #define MOLE_STATE_DISAPPEAR            1
72. //小鼠准备好要出现的状态
73. #define MOLE_STATE_READY                2
74. //小鼠出现了！的状态
75. #define MOLE_STATE_APPEAR               3
76.
77. #endif
```

文件名：game.c
功能：游戏循环的实现文件

```
1.  #include "game.h"
2.  #include "display.h"
3.  #include "random.h"
4.  #include "buzzer.h"
5.
```

```
6.
7.  //不同等级下的小鼠亮灯持续时间(ms)
8.  //等级值为 1~9
9.  unsigned long int xdata Game_Time_Level[10] = {
10.     0,2000,1700,1400,1100,
11.     900,600,400,300,200
12. };
13.
14. //把键值按顺序放在一个数组中
15. //当生成小鼠时候才好方便
16. //把小鼠的位置与键盘对应起来
17. uchar Key_Value_List[] = {
18.     KEY_NULL,KEY_1,KEY_2,KEY_3,KEY_4,KEY_5,KEY_6,
19.     KEY_7,KEY_8,KEY_9,KEY_10,KEY_11,KEY_12
20. };
21.
22. //不同等级下的晋级分数线
23. //等级值为 1~9
24. uint xdata Game_Score_Level[10] = {
25.     0,20,30,40,50,
26.     60,70,80,80,9999
27. };
28.
29. long int xdata Game_Total_Score     =   0;
30. long int xdata Game_Total_Time      =   0;
31. long int xdata Best_Total_Score     =   0;
32. long int xdata Best_Total_Time      =   0;
33.
34. //该游戏目前的等级
35. uchar Game_Level = 1;
36. //该游戏目前等级的分数
37. uint Game_Socre = 0;
38. //该游戏目前等级的倒计时
39. uint Game_Timer = GAME_TIMER_VALUE;
40.
41. //分数从 0 数到总分的动画数值
42.  uint Count_Score = 0;
43.
44.
45. //记录游戏状态的变量
46. //该游戏分为多个状态
47. //在不同的状态内执行不同的代码
48. //所有状态可能的值为 GAME_STATE_开头的值
```

```
49. //详细内容请查看 game.h
50. uchar Game_State = 1;
51. 
52. void Refresh_Game_Hook(){
53.     switch (Game_State)
54.     {
55.     case GAME_STATE_MENU_RUN:
56.         Game_Menu_Run_Hook();
57.         break;
58.     case GAME_STATE_MENU_TOP:
59.         Game_Menu_Top_Hook();
60.         break;
61.     case GAME_STATE_MENU_TL:
62.         Game_Menu_TL_Hook();
63.         break;
64.     case GAME_STATE_MENU_SL:
65.         Game_Menu_SL_Hook();
66.         break;
67.     case GAME_STATE_PLAYING:
68.         Game_Playing_Hook();
69.         break;
70. 
71.     case GAME_STATE_OVER:
72.         Game_Over_Hook();
73.         break;
74.     case GAME_STATE_SHOW_TOTAL_SCORE:
75.         Show_Total_Score_Hook();
76.         break;
77.     case GAME_STATE_NEXT_LEVEL:
78.         Next_Level_Hook();
79.         break;
80.     case GAME_STATE_SHOW_TOTAL_TIME:
81.         Show_Total_Time_Hook();
82.         break;
83.     case GAME_STATE_PAUSE:
84.         Game_Pause_Hook();
85.         break;
86.     default:
87. 
88.         break;
89.     }
90. }
91. 
```

```
92. //一个求次方的函数
93. //因为当前项目的简单要求
94. //该函数的返回值只有正整数，上限为 65535
95. //num：底数
96. //power：指数
97. uint Pow(uchar num,uchar power){
98.     uint value = num;
99.     if(power==0)return 1;
100.       power -= 1;
101.       while (power--)
102.       {
103.           value *= num;
104.       }
105.       return value;
106.   }
107.
108.   //菜单 切换页面的按键事件
109.   //这是一个公共的函数
110.   //提供给所有的菜单 Hook 函数使用
111.   void Switch_Page_Key_Event(){
112.       KEY_DOWN(KEY_NEXT_PAGE){
113.           if(++Game_State>MENU_MAX_NUMBER)Game_State = 1;
114.       }
115.       KEY_DOWN(KEY_PREV_PAGE){
116.           if(--Game_State<1)Game_State = MENU_MAX_NUMBER;
117.       }
118.   }
119.
120.   //菜单 开始游戏界面的钩子函数
121.   void Game_Menu_Run_Hook(){
122.       Switch_Page_Key_Event();
123.       Display_Show(LETTER_G,LETTER_A,LETTER_N,LETTER_E,DISPLAY_OFF,LETTER_R,LET
    TER_U,LETTER_N);
124.       KEY_DOWN(KEY_OK){
125.           Game_Level = 1;
126.           Game_Total_Score = 0;
127.           Game_Total_Time = 0;
128.           //游戏跳到显示下一等级
129.           Game_State = GAME_STATE_NEXT_LEVEL;
130.       }
131.   }
132.   //菜单 显示最好成绩的钩子函数
133.   void Game_Menu_Top_Hook(){
```

```
134.        Switch_Page_Key_Event();
135.
136.        Display_Memory[0] = LETTER_T;
137.        Display_Memory[1] = LETTER_O;
138.        Display_Memory[2] = LETTER_P;
139.        Display_Memory[3] = DISPLAY_OFF;
140.        Display_Show_Number(Best_Total_Score,7,4);
141.    }
142.    //菜单 设置不同等级的地鼠显示持续时间钩子函数
143.    void Game_Menu_TL_Hook(){
144.        static uchar caret = 1;
145.        static uchar level = 1;
146.        static uchar Is_Hide_Caret = 0;
147.        //Variable_Index 数组用于储存当前界面可以增减的
148.        //数字所在的数码管位子
149.        //该数组的第 0 位应该手动储存该储存有效位的内容个数
150.        //有效位内容从第 1 位开始计算
151.        //如果需要获取该数组的有效内推个数
152.        //应写 Variable_Index[0]
153.        //同时也可以写 Variable_Index[LENGTH]
154.        uchar code Variable_Index[]={4,2,4,5,6};
155.        static count = 0;
156.        //每 500ms 显示或者隐藏光标
157.        if(count==0)Is_Hide_Caret=0;
158.        if(count==50)Is_Hide_Caret=1;
159.        if(++count == 100) count=0;
160.        Switch_Page_Key_Event();
161.        Display_Memory[0] = LETTER_T;
162.        Display_Memory[1] = LETTER_L;
163.        Display_Memory[2] = NUMBER(level);
164.        Display_Memory[3] = DISPLAY_OFF;
165.        Display_Show_Number(Game_Time_Level[level],7,4);
166.        if(Is_Hide_Caret)Display_Memory[Variable_Index[caret]] = DISPLAY_OFF;
167.        KEY_DOWN(KEY_ADD){
168.            if(caret==1){
169.                if(++level>9) level=1;
170.            }else{
171.                Game_Time_Level[level] += Pow(10,7-Variable_Index[caret]);
172.            }
173.            count=0;
174.        }
175.        KEY_DOWN(KEY_MINUS){
176.            if(caret==1){
```

```
177.                 if(--level<1) level=9;
178.             }else{
179.                 Game_Time_Level[level] -= Pow(10,7-Variable_Index[caret]);
180.             }
181.             count=0;
182.         }
183.         KEY_DOWN(KEY_PREV_BIT){
184.             if(--caret<1)caret = Variable_Index[LENGTH];
185.             count=50;
186.         }
187.         KEY_DOWN(KEY_NEXT_BIT){
188.             if(++caret>Variable_Index[LENGTH])caret = 1;
189.             count=50;
190.         }
191.     }
192.     //菜单 设置不同等级的晋级分数条件钩子函数
193.     void Game_Menu_SL_Hook(){
194.         static uchar caret = 1;
195.         static uchar level = 1;
196.         static uchar Is_Hide_Caret = 0;
197.         uchar code Variable_Index[]={5,2,4,5,6,7};
198.         static count = 0;
199.         if(count==0)Is_Hide_Caret=0;
200.         if(count==50)Is_Hide_Caret=1;
201.         if(++count == 100) count=0;
202.         Switch_Page_Key_Event();
203.         Display_Memory[0] = LETTER_S;
204.         Display_Memory[1] = LETTER_L;
205.         Display_Memory[2] = NUMBER(level);
206.         Display_Memory[3] = DISPLAY_OFF;
207.         Display_Show_Number(Game_Score_Level[level],7,4);
208.         if(Is_Hide_Caret)Display_Memory[Variable_Index[caret]] = DISPLAY_OFF;
209.         KEY_DOWN(KEY_ADD){
210.             if(caret==1){
211.                 if(++level>9) level=1;
212.             }else{
213.                 Game_Score_Level[level] += Pow(10,7-Variable_Index[caret]);
214.             }
215.             count=0;
216.         }
217.         KEY_DOWN(KEY_MINUS){
218.             if(caret==1){
219.                 if(--level<1) level=9;
```

```
            }else{
                Game_Score_Level[level] -= Pow(10,7-Variable_Index[caret]);
            }
            count=0;
        }
        KEY_DOWN(KEY_PREV_BIT){
            if(--caret<1)caret = Variable_Index[LENGTH];
            count=50;
        }
        KEY_DOWN(KEY_NEXT_BIT){
            if(++caret>Variable_Index[LENGTH])caret = 1;
            count=50;
        }
    }

    void Game_Over_Hook(){
        static uint count = 0;
        Display_Show(LETTER_G,LETTER_A,LETTER_N,LETTER_E,LETTER_O,LETTER_U,LETTER
    _E,LETTER_R);
        if(++count == 550){
            count = 0;
            Game_State = GAME_STATE_SHOW_TOTAL_SCORE;
            //清除在这 5 秒内按键的记录
            //防止对后面代码逻辑的影响
            Key_Down_Value = KEY_NULL;
        }
    }

    void Game_Playing_Hook(){
        static uchar count = 0;
        static uchar Mole_State = MOLE_STATE_DISAPPEAR;
        static uchar random_location = 0;
        static uchar random_time_before_appear = 0;//小鼠出现前的随机时间,单位
    10ms
        static uchar Is_Show_Level_Score_Only = 0;
        if(Is_Show_Level_Score_Only){
            //只显示当前等级得分
            //使用四位数字来显示得分
            Display_Show_Number(Game_Socre,3,4);
        }else{
            //显示当前等级得分
            Display_Show_Number(Game_Socre,1,2);
            Display_Memory[1] |= 0x80;
```

```
261.            //显示当前倒计时
262.            Display_Show_Number(Game_Timer,3,2);
263.        }
264.        if(Key_Value == KEY_LEVEL_SCORE){
265.            Is_Show_Level_Score_Only = 1;
266.        }else{
267.            Is_Show_Level_Score_Only = 0;
268.        }
269.        switch (Mole_State)
270.        {
271.        case MOLE_STATE_DISAPPEAR:
272.            random_time_before_appear = random(30,70);
273.            random_location = random(1,12);
274.            Mole_State = MOLE_STATE_READY;
275.            count = 0;
276.            break;
277.        case MOLE_STATE_READY:
278.            if(++count!=random_time_before_appear)break;
279.            Show_Mole(random_location);
280.            count = 0;
281.            Mole_State = MOLE_STATE_APPEAR;
282.            break;
283.        case MOLE_STATE_APPEAR:
284.            if(++count!=Game_Time_Level[Game_Level]/10)break;
285.            Hide_Mole();
286.            count = 0;
287.            Mole_State = MOLE_STATE_DISAPPEAR;
288.            break;
289.        default:
290.            break;
291.        }
292.        ANY_KEY_DOWN{
293.            if(Key_Value == KEY_PAUSE){
294.                Game_State = GAME_STATE_PAUSE;
295.                //直接清空当前按下的键值并退出这个函数
296.                Key_Down_Value = KEY_NULL;
297.                return;
298.            }
299.            if(Key_Value == Key_Value_List[random_location]){
300.                Game_Socre+=1;
301.                Game_Total_Score+=1;
302.                Buzzer_Is_Bingo = 1;
303.                Hide_Mole();
```

```
304.                 count = 0;
305.                 Mole_State = MOLE_STATE_DISAPPEAR;
306.                 if(Game_Socre >= Game_Score_Level[Game_Level] && Game_Level!=9){

307.                     //升级啦!
308.                     Game_Level += 1;
309.                     Game_State = GAME_STATE_NEXT_LEVEL;
310.                 }
311.                 //如果游戏等级是 9，而且分是 9999，直接结束游戏
312.                 if(Game_Level == 9 && Game_Socre == 9999){
313.                     Game_Timer = 1;
314.                 }
315.             }else{
316.                 if(Key_Value!=KEY_LEVEL_SCORE)Buzzer_Is_Alert = 1;
317.             }
318.         }
319.     }
320.
321.     //刷新游戏计时器函数
322.     void Game_Timer_Hook(){
323.         if(Game_State == GAME_STATE_PLAYING){
324.             Game_Timer--;
325.             Game_Total_Time++;
326.             if(Game_Timer==0){
327.                 Game_State = GAME_STATE_OVER;
328.                 Buzzer_Player_Play(&Muisc_Gameover);
329.                 if(Best_Total_Score < Game_Total_Score){
330.                     Best_Total_Score = Game_Total_Score;
331.                 }
332.                 //清空计数动画数值
333.                 Count_Score = 0;
334.             }
335.         }
336.     }
337.
338.     void Next_Level_Hook(){
339.         static uchar count = 0;
340.         Display_Show(LETTER_L,LETTER_E,LETTER_U,LETTER_E,LETTER_L,DISPLAY_OFF,DIS
    PLAY_OFF,DISPLAY_OFF);
341.         Display_Memory[7] = NUMBER(Game_Level);
342.         if(++count == 200){
343.             Game_Socre = 0;
344.             Game_Timer = GAME_TIMER_VALUE;
```

```
345.             Game_State = GAME_STATE_PLAYING;
346.             Display_Memory[4] = DISPLAY_OFF;
347.             Display_Memory[5] = DISPLAY_OFF;
348.             Display_Memory[6] = DISPLAY_OFF;
349.             Display_Memory[7] = DISPLAY_OFF;
350.             count = 0;
351.         }
352.     }
353.
354.     //显示小鼠
355.     //参数
356.     //location：小鼠的位置，范围为 1 到 12
357.     void Show_Mole(uchar location){
358.         uchar Dispaly_Number = 7;
359.         uchar row;
360.         uchar col;
361.         uchar row_dispaly_data[] = {0x01,0x40,0x08};
362.         //转换为 0 到 11 更方便求行列
363.         location -= 1;
364.         //行：  0 到 2
365.         row = location/4;
366.         //列：  0 到 3
367.         col = location%4;
368.
369.         while (Dispaly_Number>3)
370.         {
371.             if(Dispaly_Number==col+4){
372.                 Display_Memory[Dispaly_Number] =row_dispaly_data[row];
373.             }else{
374.                 Display_Memory[Dispaly_Number] = DISPLAY_OFF;
375.             }
376.             Dispaly_Number--;
377.         }
378.
379.     }
380.
381.     //隐藏小鼠
382.     void Hide_Mole(){
383.         Display_Memory[4] = DISPLAY_OFF;
384.         Display_Memory[5] = DISPLAY_OFF;
385.         Display_Memory[6] = DISPLAY_OFF;
386.         Display_Memory[7] = DISPLAY_OFF;
387.     }
```

```
388.  //显示当前游戏的总得分
389.  void Show_Total_Score_Hook(){
390.      //计时用的
391.      static uint count = 0;
392.
393.      Display_Memory[0]=LETTER_S;
394.      Display_Memory[1]=DISPLAY_OFF;
395.      Display_Show_Number(Count_Score,7,6);
396.
397.      if(Count_Score==Game_Total_Score){
398.          ANY_KEY_DOWN{
399.              Game_State = GAME_STATE_SHOW_TOTAL_TIME;
400.          }
401.          Buzzer_noBeep();
402.          return;
403.      }
404.
405.      if(count==0){
406.          Buzzer_Beep();
407.          Count_Score++;
408.      }
409.      if(count==2){
410.          Buzzer_noBeep();
411.      }
412.      if(++count>3)count = 0;
413.
414.  }
415.
416.  //显示当前游戏消耗总时间的函数
417.  void Show_Total_Time_Hook(){
418.      Display_Memory[0]=LETTER_T;
419.      Display_Memory[1]=DISPLAY_OFF;
420.      Display_Show_Number(Game_Total_Time,7,6);
421.      ANY_KEY_DOWN{
422.          Game_State = GAME_STATE_MENU_RUN;
423.      }
424.  }
425.
426.
427.  //游戏暂停函数
428.  void Game_Pause_Hook(){
429.      static uchar Is_Hide_Caret = 0;
430.      static count = 0;
```

```
431.        if(count==0)Is_Hide_Caret=0;
432.        if(count==50)Is_Hide_Caret=1;
433.        if(++count == 100) count=0;
434.
435.        //当什么键都没按下的时候显示 pause 字样
436.        if(Key_Value == KEY_NULL){
437.            if(Is_Hide_Caret){
438.                Display_Show(DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISP
    LAY_OFF,DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF);
439.            }else{
440.                Display_Show(DISPLAY_OFF,DISPLAY_OFF,DISPLAY_OFF,LETTER_P,LETTER_
    A,LETTER_U,LETTER_S,LETTER_E);
441.            }
442.        }
443.        KEY_DOWN(KEY_CONTINUE){
444.            Game_State = GAME_STATE_PLAYING;
445.            //由于游戏循环的小鼠位置在函数内，外部无法读取
446.            //懒得改，目前就只能让游戏从暂停回来的时候
447.            //隐藏小鼠了，就当是暂停罚时。
448.            Hide_Mole();
449.        }
450.        KEY_DOWN(KEY_EXIT){
451.            Game_State = GAME_STATE_MENU_RUN;
452.        }
453.        KEY_DOWN(KEY_LEVEL_SCORE){
454.            Display_Memory[0] = LETTER_L;
455.            Display_Memory[1] = LETTER_S;
456.            Display_Memory[2] = DISPLAY_OFF;
457.            Display_Show_Number(Game_Socre,7,5);
458.        }
459.
460.        KEY_DOWN(KEY_TOTAL_SCORE){
461.            Display_Memory[0] = LETTER_S;
462.            Display_Memory[1] = DISPLAY_OFF;
463.            Display_Show_Number(Game_Total_Score,7,6);
464.        }
465.        KEY_DOWN(KEY_TOTAL_TIME){
466.            Display_Memory[0] = LETTER_T;
467.            Display_Memory[1] = DISPLAY_OFF;
468.            Display_Show_Number(Game_Total_Time,7,6);
469.        }
470.    }
```