ARM 原理与应用

期末设计报告

安全时钟的设计

|  |  |
| --- | --- |
| 班级 | 通信22-5 |
| 姓名 | 黄彬 |
| 学号 | 2205030510 |
| 报告成绩 |  |
|  | 文献和方案（20）： |
| 硬件设计（30）： |
| 软件设计（30）： |
| 团队和创新（20）： |
| 评阅教师 |  |

1. 功能设计

近年来，卧室火灾频发，其主要原因多为未完全熄灭的烟头、过热的手机充电器或充电宝等引起的安全隐患。传统的室内消防系统通常需要较高的烟雾浓度或室内温度达到触发条件，但往往等到触发时，已经给用户造成了不可忽视的经济损失。为提前预防火灾，降低损失成本，同时兼顾卧室生活场景的实用性，本项目设计了一种集时间管理、计数、报警功能于一体的多功能安全时钟系统。

该系统具有以下主要功能：时钟显示、定时器、倒计时计数器、闹钟及气体检测。用户可以通过按键设置时间、日期及倒计时等功能，实时显示相关信息。系统在计数器或闹钟触发时，通过蜂鸣器发出声光报警提醒用户；同时，气体传感器能够实时监测环境中的气体浓度，当浓度异常时，系统将自动启动报警装置以提醒用户。

本系统的具体设计参数如下：

**检测项目：**

* 时间、日期、分钟的设置和显示；
* 倒计时和计数器功能；
* 闹钟功能
* 环境气体浓度监测（通过MQ-2气体传感器）。

**报警方式：**

* 声光报警：通过蜂鸣器发出高音警报，吸引用户注意。

**定时功能：**

* 支持用户设置时间、日期、倒计时等，并实时在OLED显示屏上显示。

**气体检测功能：**

* 使用MQ-2气体传感器实时监测环境中的气体浓度，当浓度超出安全范围时触发报警。

**自动控制功能：**

* 支持基于时间段的人体红外检测和声控模式切换，优化系统功耗。
* 白天通过红外模块检测人体存在启动显示，夜间通过声音传感器识别异常声音触发显示。

**按键操作：**

* 用户通过按键切换不同功能模块（如时钟、计数器、气体检测等），并设置相关参数。

2. 硬件设计

2.1 系统结构框图

本系统由STM32F103C8主控制器、OLED显示模块、TM1637数码管显示模块、蜂鸣器模块、按键模块、MQ-2烟雾传感器模块、人体红外传感器模块、声音传感器模块7个部分组成，其结构框图如图1所示。



图1 系统结构框图

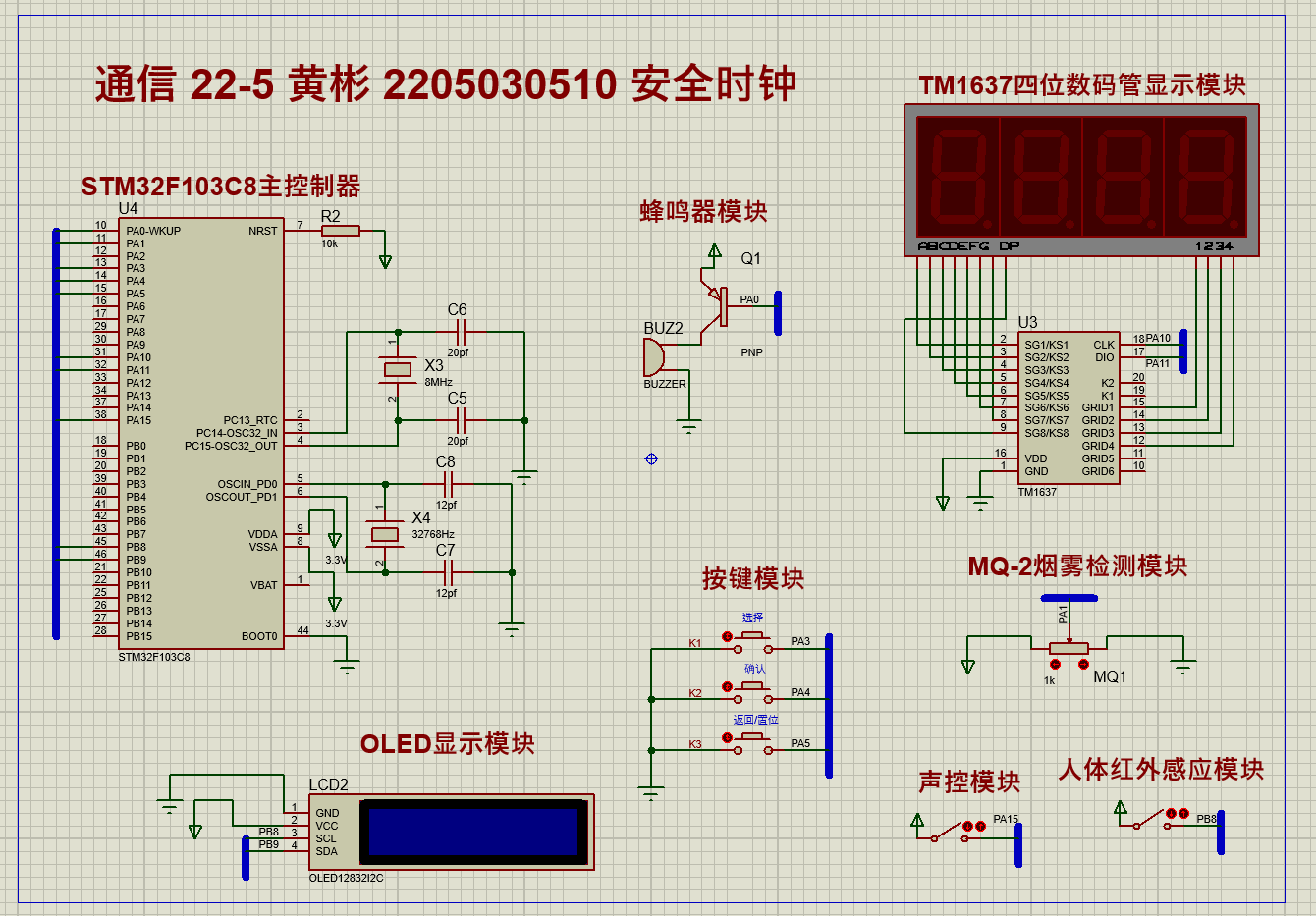


图2 系统电路原理图

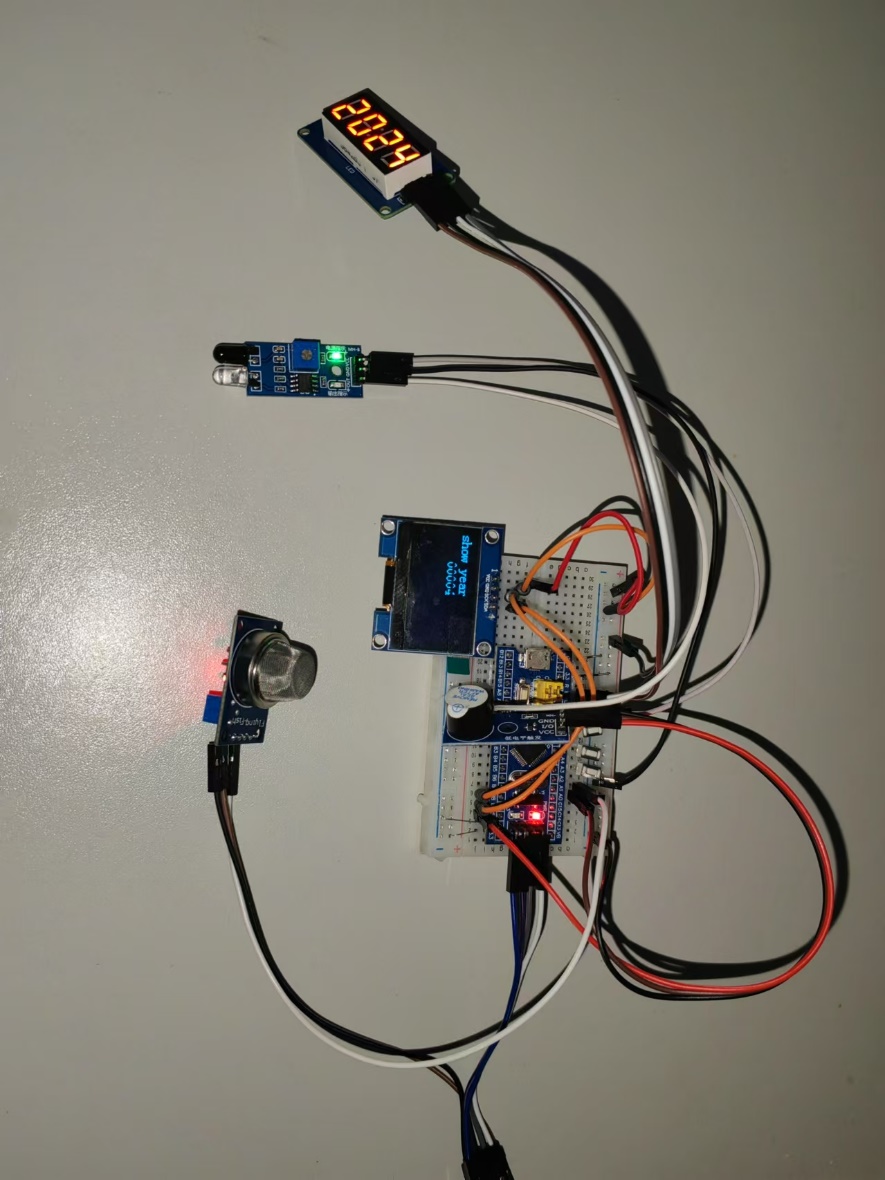


图3 实物图

2.2 主控制器选型

在本系统中，选用了STM32F103C8单片机作为主控制器。该芯片采用高性能的ARM Cortex-M3内核，具有最大72MHz的工作频率。STM32F103C8支持多种外设接口，如I2C、SPI、GPIO等，能够方便地与OLED显示屏、TM1637数码管、按键输入、气体传感器等模块进行连接。此外，该芯片还具有较低的功耗和丰富的中断功能，适合用于实时性要求较高的多功能系统设计。

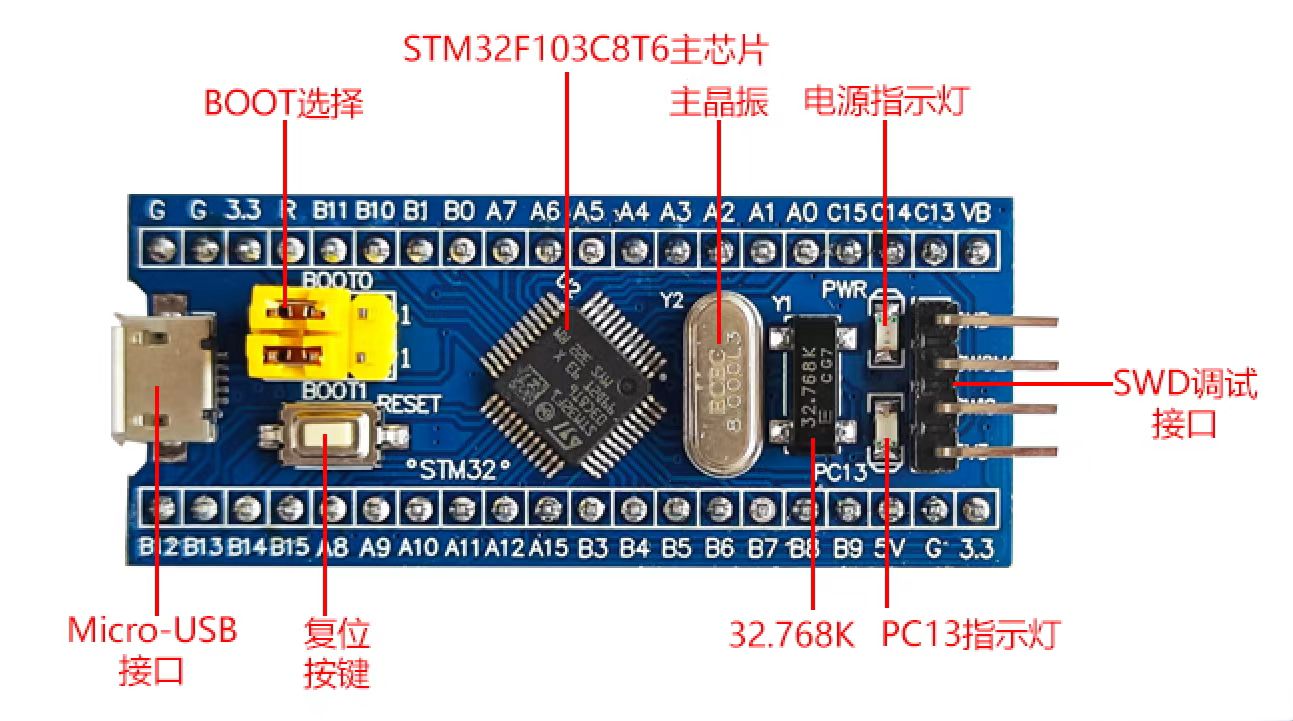


图4 主控制器实物图

2.3 MQ-2烟雾传感器选型

本系统选用MQ-2型气体传感器来进行气体浓度检测。MQ-2传感器能够检测多种气体，如甲烷、烟雾、液化气等。其测量范围为200-10000ppm，精度高，适合用于检测室内空气质量。该传感器通过模拟输出和数字输出两种方式传递数据，与STM32控制器通过ADC（模拟到数字转换）接口连接。

|  |  |
| --- | --- |
| **检测物理量** | **气体浓度** |
| **测量范围** | 200ppm-10000ppm |
| **精度** | ±10% |
| **输出信号类型** | 模拟信号（电压） |
| **接口形式** | 模拟输出，接入STM32的ADC通道 |

图片包含 服装, 游戏机, 帽子

描述已自动生成

图5 MQ-2模块实物图

2.4数码管显示模块选型

数码管显示模块选用了TM1637驱动的四位七段数码管。该模块适用于显示数字信息，具有较高的亮度和较低的功耗，广泛应用于各类显示系统中。TM1637模块支持通过I2C协议与主控芯片进行通信，简化了电路连接，且支持调节显示亮度。每个模块包含四个数码管，每个数码管能够显示0到9的数字，模块内部集成了驱动电路，可以直接控制数字的显示。该模块适合用于显示时间、倒计时或其他数字数据。



图6 TM1637数码管显示模块实物图

2.5人体红外传感器模块选型

SR501模块介绍

引脚 ： VCC，OUT, GND。

功能 ：一种常见的人体红外传感器模块，用于检测人体的活动。

红外感应原理 ：SR501 模块内部包含一个红外传感器探测单元，该单元可以检测环境中的红外辐射变化。即使在黑暗中，人体也会通过辐射红外能量来体现温度差异。

* 静止状态（无人靠近）时，OUT 引脚通常为低电平。
* 当有人体靠近时，OUT 引脚通常会从低电平变为高电平。

图片包含 室内, 小, 水槽, 桌子

描述已自动生成电子器材

中度可信度描述已自动生成

图7 人体红外传感器模块实物图

2.6声音传感器模块选型

声音传感器-（KY-037）

本系统选用KY-037型声音传感器来进行声音强度检测。KY-037传感器能够检测环境中的声音变化，其设计包含一个电容式麦克风和信号放大电路，能够检测并放大声音信号，适合用于环境噪声监测和声音触发控制。该传感器通过模拟输出和数字输出两种方式传递数据，与STM32控制器通过通用接口连接。

|  |  |
| --- | --- |
| 检测物理量 | 声音强度（音量） |
| 测量范围 | **低至高音量范围** |
| 精度 | **依赖于使用场景** |
| 输出信号类型 | **数字信号（高低电平）** |
| 接口形式 | **数字输出接口** |

卡通人物

中度可信度描述已自动生成

图8 声音传感器模块实物图

2.7主控制器外围电路设计

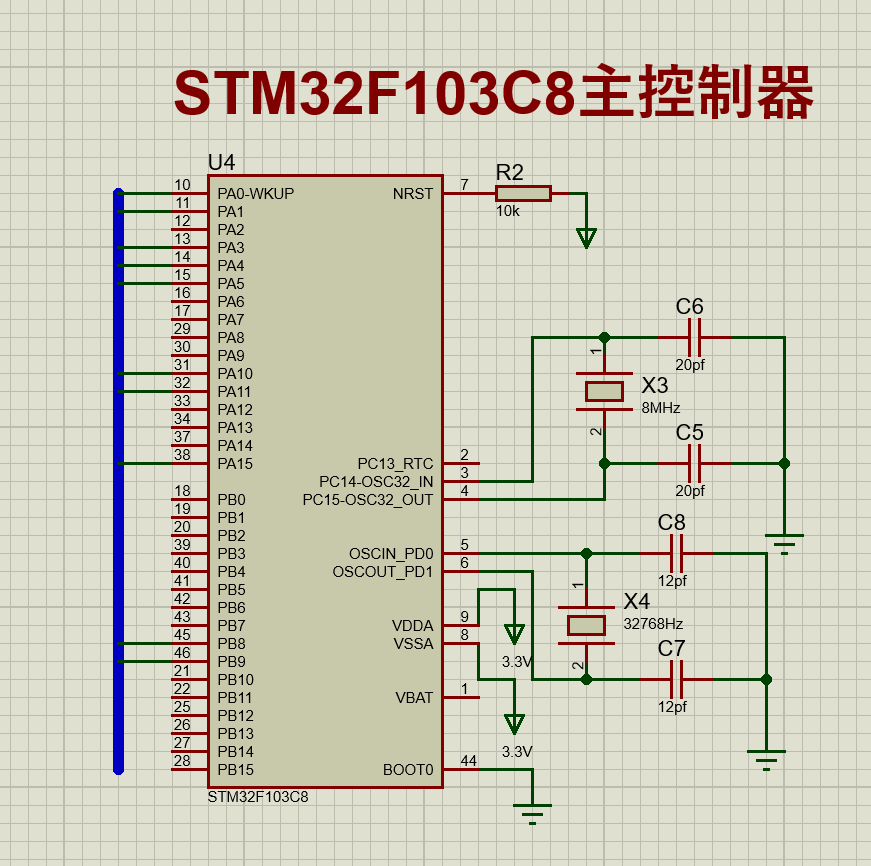


图9 主控制器外围电路原理图

主控制器外围电路如图所示。本设计采用了8 MHz主频的晶振（X3）作为系统时钟源，通过外部负载电容C6、C5（均为20pF）连接至STM32的OSC\_IN和OSC\_OUT引脚，为主时钟提供稳定的振荡信号。此外，电路中还包含一个32.768 kHz的晶振（X4），通过外部负载电容C7和C8（均为12pF）连接到PC14（OSC32\_IN）和PC15（OSC32\_OUT）引脚，用于实时时钟（RTC）模块的低频时钟输入。

复位电路由一个10kΩ电阻（R2）和NRST引脚构成，通过将NRST引脚上拉到3.3V确保系统正常工作，并提供手动复位功能。

STM32芯片的基本电源连接包括：

* VCC引脚连接到3.3V电源。
* VSS（或GND）引脚连接到地。

此外，与外部电路连接的主要引脚通过信号线标注反映在电路图中，例如GPIO引脚用于与外设的交互。

此设计保证了STM32F103C8芯片在外部晶振和供电电路支持下正常运行，并为其RTC功能提供了独立的低频时钟源。

2.8 TM1637数码管电路设计

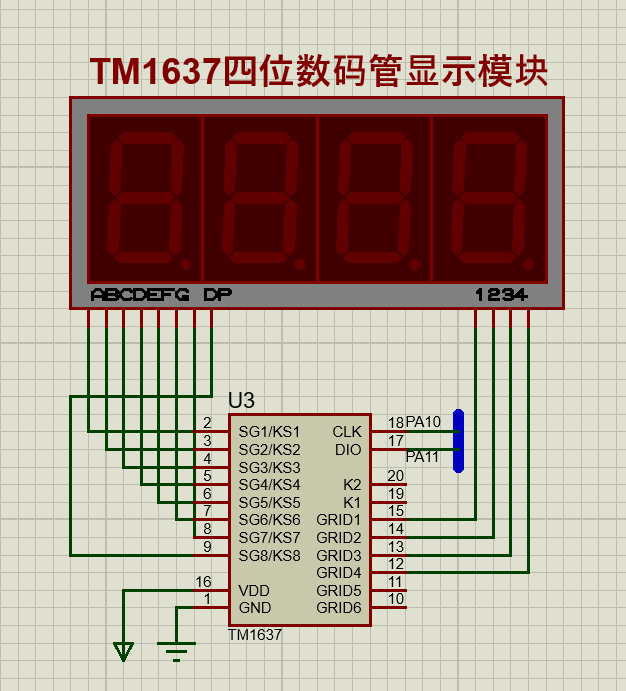


图10 TM1637数码管电路原理图

TM1637是一种常见的四位数码管显示驱动模块。其电路设计如图所示，主要由TM1637芯片和四位数码管组成，通过时钟（CLK）和数据（DIO）两根线与主控制器进行通信。

电路设计说明

1. 主要元件：
   * TM1637芯片（U3）：负责驱动四位共阳极数码管显示。
   * 四位数码管：显示数字和简单符号。
   * STM32控制器：通过GPIO引脚（PA10: CLK, PA11: DIO）与TM1637通信，控制显示内容。
2. 接口连接：
   * VDD和GND：VDD接3.3V电源，GND接地。
   * CLK：连接到STM32的PA10，用于传输时钟信号。
   * DIO：连接到STM32的PA11，用于传输数据信号。
   * GRID1~GRID4：分别连接数码管的4位位选引脚。
   * SG1~SG8：分别连接数码管的8个段选引脚（A-G段和小数点DP）。
3. 工作原理：
   * TM1637通过时钟（CLK）和数据（DIO）接口接收STM32的控制信号，驱动四位数码管显示内容。
   * 位选（GRID1~GRID4）控制数码管的各位显示状态，段选（SG1~SG8）控制对应段的点亮状态。

功能特点

* 简化接口：只需两根控制线即可实现显示功能。
* 灵活显示：支持显示数字和部分符号。
* 低功耗：适合嵌入式低功耗系统。

电路原理图

如图所示，TM1637的接口与STM32控制器及四位数码管合理连接，构成完整的显示模块。通过STM32的GPIO端口控制TM1637芯片的时钟和数据接口，实现数码管内容的动态刷新。

2.9 按键模块电路设计

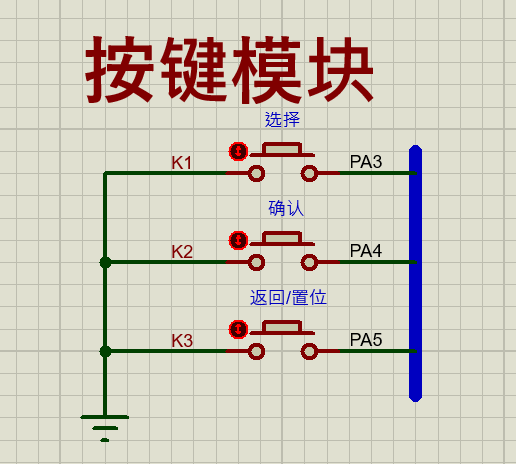


图11 按键模块电路原理图

按键模块电路设计如图所示，包含三个按键（K1、K2、K3），分别连接到STM32的GPIO引脚（PA3、PA4、PA5）。通过简单的电路设计实现按键检测功能，用于用户输入操作。

电路设计说明

1. 主要元件：
   * 按键开关（K1、K2、K3）：
     + K1功能为“选择”，对应GPIO引脚PA3。
     + K2功能为“确认”，对应GPIO引脚PA4。
     + K3功能为“返回/置位”，对应GPIO引脚PA5。
   * 电阻接地：每个按键连接到地，确保按下时信号稳定。
2. 接口连接：
   * 每个按键的一端直接连接到STM32的对应GPIO引脚，另一端接地（GND）。
3. 工作原理：
   * 按键模块采用拉低检测逻辑，按键未按下时，GPIO引脚维持高电平状态。
   * 当按键被按下时，对应引脚通过按键与地直接导通，电平被拉低，检测到低电平信号。
4. 功能特点：
   * 简单可靠：通过拉低电平检测按键状态，硬件设计简单。
   * 灵活配置：可通过软件定义按键功能。
   * 低功耗：未按下时几乎不消耗电流。

电路原理图

如图所示：

* 按键模块中每个按键的一端直接连接到STM32的GPIO引脚，另一端接地，形成完整的按键检测电路。
* STM32通过对PA3、PA4、PA5引脚电平状态的检测来判断按键是否被按下。
* 此模块适用于各种嵌入式控制系统，用于实现用户交互功能。

2.10 蜂鸣器模块电路设计

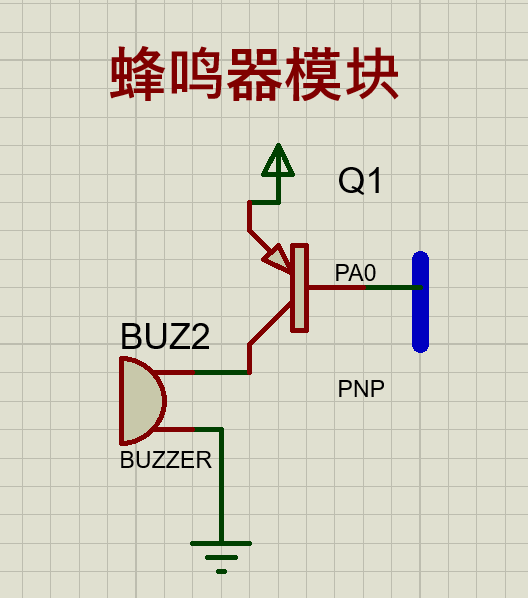


图12 蜂鸣器模块电路原理图

蜂鸣器模块电路设计如图所示，采用PNP型三极管作为开关控制，通过STM32的PA0引脚驱动蜂鸣器（BUZZER）工作，实现声音提示功能。

电路设计说明

1. 主要元件：
   * PNP型三极管（Q1）：用于控制蜂鸣器的通断。
   * 蜂鸣器（BUZZER）：发出声响的执行元件。
   * STM32 PA0引脚：提供控制信号，决定蜂鸣器是否工作。
2. 接口连接：
   * 蜂鸣器正极连接到电源（+3.3V）。
   * 蜂鸣器负极通过PNP型三极管（Q1）连接到地。
   * 三极管基极通过STM32的PA0引脚连接控制信号。
3. 工作原理：
   * PNP三极管的基极接低电平时导通，使蜂鸣器电路形成闭合，蜂鸣器发出声响。
   * PNP三极管的基极接高电平时截止，蜂鸣器电路断开，停止工作。
   * STM32通过控制PA0的输出高低电平来实现蜂鸣器的开启和关闭。
4. 功能特点：
   * 控制简单：通过单片机的GPIO引脚直接控制。
   * 声光提示：可用于报警、状态提示等场景。
   * 低功耗：在不工作时功耗极低。

电路原理图

如图所示，蜂鸣器模块由PNP型三极管和蜂鸣器组成。三极管的基极连接到STM32的PA0引脚，通过高低电平控制蜂鸣器的工作状态。此模块具有设计简单、控制方便的特点，适用于嵌入式系统中的报警提示功能。

2.11 OLED显示模块电路设计

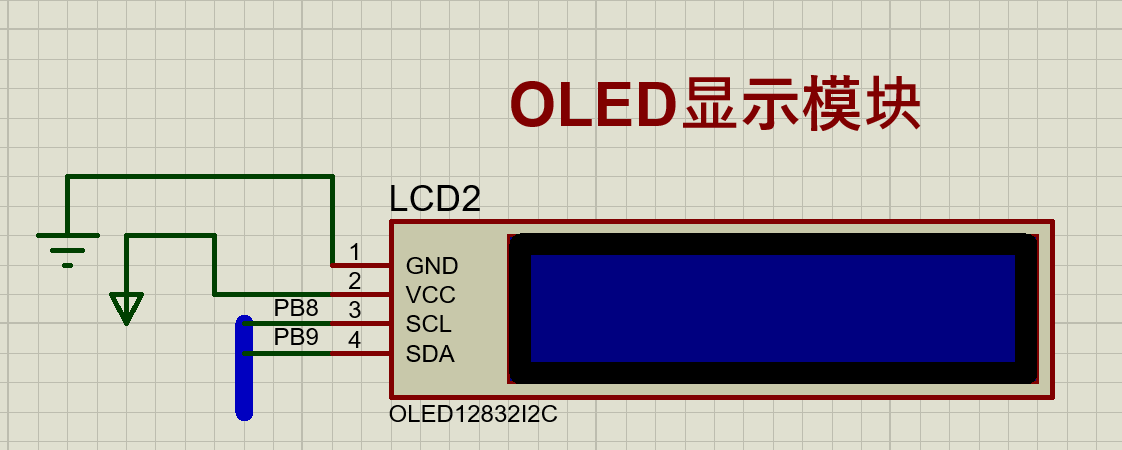


图13 OLED显示模块电路原理图

OLED显示模块电路设计如图所示，采用OLED12832 I2C接口显示屏，通过I2C通信与STM32的PB8和PB9引脚连接，实现显示功能。

电路设计说明

1. 主要元件：
   * OLED显示屏（LCD2）：分辨率为128x32，支持I2C通信协议，用于显示文字和图形信息。
   * STM32 I2C接口：
     + PB8引脚作为I2C时钟信号（SCL）。
     + PB9引脚作为I2C数据信号（SDA）。
2. 接口连接：
   * GND：接系统地。
   * VCC：接系统电源（3.3V）。
   * SCL：连接到STM32的PB8引脚，用于传输时钟信号。
   * SDA：连接到STM32的PB9引脚，用于传输数据信号。
3. 工作原理：
   * STM32通过I2C通信向OLED显示屏发送控制命令和显示数据。
   * SCL引脚提供时钟信号，同步数据传输。
   * SDA引脚传输显示内容数据。
   * OLED显示屏根据接收的数据更新显示内容。
4. 功能特点：
   * 低功耗：OLED显示屏功耗低，适用于便携式设备。
   * 高对比度：OLED显示屏具有高对比度，显示效果清晰。
   * 简单接口：使用I2C通信，只需两根信号线即可完成数据传输。
   * 灵活显示：可显示文字、数字和简单图形。

电路原理图

如图所示，OLED模块电路采用4个引脚：

* GND连接系统地。
* VCC连接系统电源（3.3V）。
* SCL和SDA分别连接STM32的PB8和PB9引脚，形成I2C通信接口。
* 此模块通过I2C通信与主控板实现显示功能，适用于嵌入式显示场景，结构简单且功能强大。

2.12其他简化电路设计

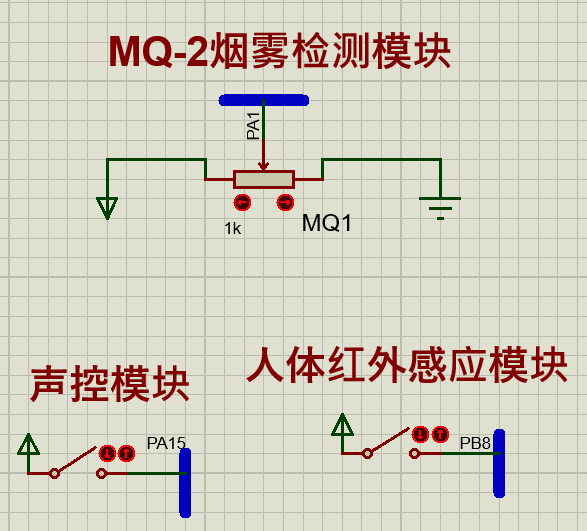


图14 其他模块电路原理图

**MQ-2烟雾检测模块电路设计**

MQ-2烟雾检测模块通过检测空气中可燃气体的浓度，提供一个与浓度成比例的电压输出，具体电路设计如下：

* 主要元件：
  + MQ-2传感器（MQ1）：检测烟雾或可燃气体浓度。
  + 分压电阻（1kΩ）：与MQ-2内部可变电阻形成分压电路，将输出信号提供给STM32的PA1引脚。
* 接口连接：
  + MQ-2模块的信号输出端通过1kΩ电阻连接至STM32的PA1引脚，提供模拟电压信号。
  + 供电端连接3.3V，接地端连接系统地。
* 工作原理：
  + MQ-2根据烟雾浓度改变其内部电阻，形成与浓度相关的电压信号。
  + STM32通过ADC读取PA1引脚的电压，计算烟雾浓度。

**声控模块电路设计**

声控模块用于检测环境声音的强度，并通过信号输出控制相关功能，电路设计如下：

* 主要元件：
  + 声音传感器：检测环境中的声音。
* 接口连接：
  + 声控模块的信号输出端连接至STM32的PA15引脚。
* 工作原理：
  + 声控模块检测到声音信号时，输出高电平。
  + STM32通过读取PA15引脚的电平状态，触发相应控制逻辑。

**人体红外感应模块电路设计**

人体红外感应模块用于检测人体的红外辐射，判断是否有人体存在，电路设计如下：

* 主要元件：
  + 红外感应模块：检测人体红外信号。
* 接口连接：
  + 红外模块的信号输出端连接至STM32的PB8引脚。
* 工作原理：
  + 当检测到人体红外信号时，模块输出高电平。
  + STM32通过读取PB8引脚的电平状态，判断是否有人体存在，并执行相关控制功能。

**功能总结**

以上简化电路模块设计实现了以下功能：

* **MQ-2烟雾检测模块**：监测环境中的烟雾和可燃气体浓度。
* **声控模块**：检测声音变化，用于声音触发控制。
* **人体红外感应模块**：检测人体存在，用于控制系统的自动化功能。

这些模块的设计简单、实用，便于嵌入式系统的集成和功能扩展。

3. 软件设计

3.1 主程序设计

主程序流程图如图15所示。



图15 主程序流程图

本程序是基于 STM32 的嵌入式系统控制程序，集成了多个模块（蜂鸣器、按键、OLED 显示器、TM1637 数码管、MQ-2 气体传感器、红外传感器、语音传感器等），实现了多种功能，包括时间显示、警报提示、模式切换和输入检测。

1）系统初始化

调用以下初始化函数：Beep\_init：初始化蜂鸣器。

Timer\_Init 和 Timer\_Init1：初始化 TIM2 和 TIM3 定时器。

Key\_Init：初始化按键输入。

OLED\_Init：初始化 OLED 显示器。

Infrared\_Init：初始化红外传感器。

VoiceSensor\_Init：初始化语音传感器。

1. 主循环

显示主菜单:通过 OLED 显示当前功能选项，例如 "Key1 select" 和 "Key2 ok"。

按键检测:调用 KEy\_getNUM 检测按键输入，进入不同功能模式：定时器模式（Timer）。

计数器模式（Counter）。

时钟设置模式（Clock）。

警报模式（Alarm）。

时间比较:如果当前时间与设置的时间相等，启动蜂鸣器。

1. 模式功能

定时器模式:显示和设置时间、日期等。

用户通过按键调整年、月、日、小时、分钟。

数码管显示对应的数据。

计数器模式:使用 TIM3 定时器进行计数。

OLED 和数码管显示实时计数值。

时钟设置模式:用户可设置目标时间和分钟。

定时触发蜂鸣器。

警报模式:MQ-2 气体传感器检测到信号时触发蜂鸣器。

红外/语音传感器检测:白天基于红外传感器检测是否有人。

夜晚基于语音传感器检测是否有声音。

检测到信号后，显示对应提示并启动计时。

1. 显示和警报控制

检测显示信号:白天显示 "Person Detected"。

夜晚显示 "Sound Detected"。

清除显示:如果显示已持续 1 分钟，清除显示内容并重置系统。

程序总结

优点

多功能集成:集成蜂鸣器、OLED、数码管、气体传感器、红外和语音传感器，实现多种功能。

模块化设计:每个功能模块独立，便于扩展和调试。

用户交互性强:提供按键输入支持，允许用户设置和切换模式。

缺点

复杂性高:主循环逻辑较复杂，分支过多，可能影响维护和调试。

错误处理不足:缺少硬件故障检测和错误处理机制。

性能问题:某些功能（如清屏和计时）可能影响系统实时性。

3.2 OLED模块子程序设计

OLED模块子程序流程图如图16所示。



图16 OLED显示模块子程序流程图

程序工作流程分析

该程序实现了 OLED 显示模块的初始化、字符显示、字符串显示、清屏等功能，主要通过 I2C 通信实现数据的传输和控制。

1）初始化

GPIO 初始化（OLED\_I2C\_Init）:开启 GPIOB 时钟。

配置 PB8 和 PB9 引脚为推挽输出模式，用于 I2C 的 SCL 和 SDA。

默认将 SCL 和 SDA 设置为高电平。

OLED 初始化（OLED\_Init）:上电延时等待稳定。

配置 OLED 的各项显示参数（如时钟分频比、方向控制、对比度等）。

发送指令开启显示。

清屏以确保 OLED 显示处于初始状态。

1. I2C 通信

起始信号（OLED\_I2C\_Start）:在 SCL 高电平时，将 SDA 拉低以生成起始信号。

停止信号（OLED\_I2C\_Stop）:在 SCL 高电平时，将 SDA 拉高以生成停止信号。

发送字节（OLED\_I2C\_SendByte）:循环发送 8 位数据，依次设置 SDA 电平，并在 SCL 的上升沿表示数据有效。

写命令（OLED\_WriteCommand）:发送 I2C 地址，表示写入命令数据。

发送具体命令。

写数据（OLED\_WriteData）:发送 I2C 地址，表示写入显示数据。

发送具体数据。

1. 显示控制

光标设置（OLED\_SetCursor）:设置 OLED 显示的行（Y 方向）和列（X 方向）位置。

清屏（OLED\_Clear）:遍历 OLED 的所有显示位置，将数据清零。

字符显示（OLED\_ShowChar）:通过光标设置，分两部分（上下）写入字符数据。

字符串显示（OLED\_ShowString）:循环调用字符显示函数，将字符串逐个字符显示在 OLED 上。

数字显示（OLED\_ShowNum）:将数字按十进制拆分成字符，通过字符显示函数逐个显示。

支持其他格式显示:提供十六进制和二进制数字的显示函数。

程序总结

优点

功能齐全:支持字符、字符串、十进制、十六进制、二进制等多种格式显示。

模块化设计:初始化、通信、显示功能分离，便于扩展和维护。

兼容性好:使用 I2C 协议通信，适用于多种硬件平台。

缺点

硬编码问题:显示参数（如地址、指令）硬编码，扩展性有限。

性能优化不足:清屏和复杂显示操作可能占用较多时间。

缺乏错误处理:通信失败或硬件异常未进行检测和处理。

3.3TM1637数码管显示模块子程序设计

TM1637数码管显示模块子程序流程图如图17所示。



图17 TM1637数码管模块子程序流程图

程序工作流程分析

该程序用于通过 I2C 协议控制 TM1637 数码管，实现数码管的显示操作，包括初始化、数据发送、清除显示等功能。以下是详细分析：

1）初始化（TM1637\_Init 函数）

功能: 初始化 GPIO 引脚，为 TM1637 通信准备。

详细步骤:开启 GPIOB 时钟。

将 PB10（SCL）和 PB11（SDA）引脚配置为开漏输出。

设置默认电平（高电平），释放总线。

1. 通信控制（TM1637\_Start 和 TM1637\_Stop 函数）

TM1637\_Start:产生起始信号：SCL 和 SDA 拉高。

在 SCL 高电平期间，拉低 SDA。

TM1637\_Stop:产生停止信号：SCL 和 SDA 拉低。

在 SCL 高电平期间，释放 SDA。

1. 数据发送与接收

发送字节（TM1637\_Write\_Byte）:循环发送 8 位数据，每发送一位：设置 SDA 电平为数据位。

拉高 SCL，表示位数据有效。

拉低 SCL，准备发送下一位。

接收应答（TM1637\_ReceiveAck）:释放 SDA，拉高 SCL 读取从机应答信号。

1. 数码管显示控制

写命令（TM1637\_WriteCmd）:使用起始信号、命令字节和停止信号完成。

写数据（TM1637\_WriteData）:指定地址发送数据字节到 TM1637 数码管。

显示数据（TM1637\_TubeDisplay）:根据段码表 u8NumTab 转换数码管数据并写入对应地址。

设置亮度（TM1637\_SetBrightness）:通过命令字节控制显示亮度。

1. 数码管显示处理

处理单一数字（Display\_TubeDataProcess）:将数字拆分为各个位，填充到段码表对应位置。

处理多数字组合（Display\_TubeDatatwo）:同时处理两个两位数的数据。

程序总结

优点

功能完整:实现了 TM1637 所有基础功能（初始化、通信、显示控制等）。

模块化设计:各功能独立，便于扩展和维护。

高效通信:使用 I2C 协议与 TM1637 通信，时序控制精确。

缺点

硬编码限制:数码管段码表和地址表写死，扩展性不足。

复杂性高:代码量较大，嵌套函数调用较多，逻辑复杂。

缺乏错误处理:对于通信失败（如未接收到应答）没有处理机制。

3.4按键模块子程序设计

按键模块子程序流程图如图18所示。



图18按键模块子程序流程图

程序工作流程分析

该程序通过 STM32 微控制器的 GPIO 模块检测按键输入，主要功能包括初始化按键硬件和获取按键状态。以下是详细的工作流程分析：

1）按键初始化（Key\_Init 函数）

功能: 初始化 GPIO 引脚以检测按键输入。

详细步骤:启用 GPIOA 的时钟 (RCC\_APB2PeriphClockCmd)。

配置 GPIO 引脚（Pin 5、7、3）为上拉输入模式（GPIO\_Mode\_IPU）。

设置 GPIO 引脚的速度为 50 MHz。

1. 获取按键状态

去抖按键检测（KEy\_getNUM 函数）:功能: 获取单个按键的状态，支持按键去抖。

详细步骤:依次检测每个按键是否被按下（GPIO 电平为低）。

如果按下，延时 30 ms 去抖。

等待按键释放（循环检测直到按键释放）。

延时 30 ms，再次去抖。

返回按键编号（1、2 或 3）。

非去抖按键检测（KEy\_getNUM1 函数）:功能: 获取按键状态，直接返回按键编号，无去抖。

详细步骤:检测 GPIO 引脚状态。

同时支持组合按键的检测（如按键 1 和 2 同时按下返回编号 4）。

返回按键编号。

1. 按键计数与显示

按键计数（keycount 函数）:功能: 获取按键编号，调用 KEy\_getNUM 实现。

按键显示编号（shownum 函数）:功能: 占位函数，尚未实现，可能用于后续扩展功能。

程序总结

优点

功能明确:代码实现了按键检测的基础功能，支持去抖处理。

模块化设计:初始化、检测和计数功能分开，易于扩展。

支持组合按键:通过 KEy\_getNUM1 实现多按键检测功能。

缺点

功能不完善:shownum 函数未实现具体功能，代码存在占位部分。

缺乏错误处理:未处理异常情况（如按键检测超时或硬件故障）。

硬编码限制:GPIO 引脚和按键编号硬编码在程序中，灵活性不足。

3.5蜂鸣器模块子程序设计

蜂鸣器模块子程序流程图如图19所示。



图19 蜂鸣器模块子程序流程图

该程序控制一个蜂鸣器的启动和停止，使用 STM32 微控制器的 GPIO 和定时器硬件模块。以下是程序的详细工作流程和功能描述：

1）初始化（Beep\_init 函数）

GPIO 配置:启用 GPIOA 的时钟 (RCC\_APB2PeriphClockCmd)。

配置 GPIOA 的 Pin 0 为推挽输出模式，速度为 50 MHz。

默认将蜂鸣器关闭 (BEEP\_OFF)。

定时器配置:启用 TIM4 定时器的时钟 (RCC\_APB1PeriphClockCmd)。

设置定时器时基单位：计数器频率设为 10 kHz (72 MHz 除以 7200)。

周期设置为 4000（400 ms 中断周期）。

清除 TIM4 的更新中断标志。

配置 TIM4 的中断优先级并启用中断 (NVIC\_Init)。

初始状态:禁用定时器 TIM4 (TIM\_Cmd(DISABLE))。

1. 蜂鸣器启动（BEEP\_Start 函数）

设置蜂鸣器状态为开启（beep\_state = 1）。

将定时器计数器清零 (TIM\_SetCounter)。

启用定时器 (TIM\_Cmd(ENABLE))。

1. 蜂鸣器停止（BEEP\_Stop 函数）

设置蜂鸣器状态为关闭（beep\_state = 0）。

禁用定时器 TIM4 (TIM\_Cmd(DISABLE))。

确保蜂鸣器关闭（BEEP\_OFF）。

1. 定时器中断处理（TIM4\_IRQHandler 函数）

检测 TIM4 的更新中断事件。

如果蜂鸣器处于开启状态（beep\_state = 1）:控制蜂鸣器的开/关状态，每 400 ms 切换一次。

当蜂鸣次数达到 5 次（即 2 秒）后：停止蜂鸣器。

禁用定时器 TIM4。

如果蜂鸣器已关闭或达到次数限制，确保蜂鸣器保持关闭状态。

程序总结

优点

模块化设计:功能分工明确，易于维护和扩展（如 GPIO、定时器初始化和中断处理分开）。

实时性强:使用定时器硬件模块，保证高精度的时间控制。

低功耗和资源占用:在不需要蜂鸣器时，可以禁用定时器以节省资源。

缺点

硬编码限制:蜂鸣器鸣响次数和间隔时间是硬编码的（5 次、400 ms），不够灵活。

缺少错误处理:如果硬件初始化失败或中断处理过程中发生意外，缺乏容错机制。

可移植性较低:依赖 STM32 特定硬件寄存器和库，不易移植到其他平台。

3.6其他模块子程序设计

MQ-2模块、红外感应模块、声音感应模块子程序流程图如图19所示。



图20 MQ-2模块子程序流程图

程序工作流程分析

此程序主要负责初始化 MQ-2 气体传感器的输入引脚，并从传感器读取数据。以下是详细分析：

1）初始化 MQ-2（MQ\_2\_init 函数）

功能: 该函数负责设置 GPIO 的相关配置，以连接 MQ-2 传感器。

详细步骤:启用 GPIOA 的时钟 (RCC\_APB2PeriphClockCmd)。

设置 GPIOA 的 Pin 1 为上拉输入模式（GPIO\_Mode\_IPU），用于读取传感器数据。

将配置写入到指定 GPIO 引脚 (GPIO\_Init)。

1. 获取传感器数据（MQ2\_getData 函数）

功能: 读取连接 MQ-2 传感器的 GPIO 输入数据。

详细步骤:读取 GPIO 引脚的输入电平 (GPIO\_ReadInputDataBit)。

返回读取到的传感器数据（数据类型为 uint16\_t，但实际上是一个 0 或 1 的二进制值）。

程序总结

优点

简单直观:函数功能单一且逻辑清晰，易于理解和维护。

模块化设计:通过分离初始化和数据读取，便于程序扩展或功能更改。

硬件资源使用有效:使用 STM32 的 GPIO 硬件模块，无需额外的外设或复杂操作。

缺点

功能单一:只支持读取单一引脚的数据，无法处理复杂的传感器特性（如模拟信号的采样）。

缺乏错误处理:如果 GPIO 初始化失败或引脚未正确配置，程序无法检测到或纠正错误。

硬编码限制:GPIO 配置参数硬编码在程序中，缺乏灵活性，不适合动态需求。

**其他两个模块的内容高度相似，代码逻辑也相同，这里就不再赘述。**

4. 程序清单

|  |
| --- |
| //main函数 #include "stm32f10x.h" // Device header  #include "beep.h"  #include "delay.h"  #include "timer.h"  #include "KEY.h"  #include "LED.h"  #include "OLED.h"  #include "1637.h"  #include "showtimer.h"  #include "MQ-2.h"  #include "Infrared.h"  #include "Voice.h"  #include <stdbool.h>  #include "1637.h" // 确保声明了 TM1637\_DisplayNumber 和 TM1637\_Clear  // 软件时钟结构体  typedef struct {  uint8\_t hour;  uint8\_t minute;  uint8\_t second;  } SoftwareClock;  // 声明外部软件时钟变量  extern volatile SoftwareClock SysClock;  // 全局变量  uint8\_t Keynum;  uint16\_t shownum(void);  uint8\_t keycount(void);  uint16\_t NUM = 0;  static uint16\_t num = 0;  uint16\_t J = 0;  uint16\_t NUM1 = 0;  uint16\_t num1 = 0;  uint16\_t year, month;  uint16\_t arr1[] = {2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032};  uint16\_t arr2[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0};  uint16\_t arr3[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 0};  uint16\_t arr4[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24};  uint16\_t arr5[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59};  uint16\_t y = 0;  uint16\_t m = 0;  uint16\_t d = 0;  uint16\_t t = 0;  uint16\_t s = 0;  uint16\_t t1 = 0;  uint16\_t s1 = 1;  uint8\_t bFlag = 0;  static uint8\_t Keyn = 0;  uint8\_t SK = 0;  uint8\_t KS = 0;  uint32\_t flag = 0;  uint8\_t S = 0;  uint8\_t S1 = 0;  uint16\_t value;  // 显示保持标志  bool display\_on = false;  // 函数声明  void show(uint16\_t Year, uint16\_t Mon, uint16\_t Day, uint16\_t Tim, uint16\_t sec);  bool TIM3\_HasElapsed(void);  void TIM3\_Start(void);  void TIM3\_Stop(void);  uint8\_t keycount(void);  uint16\_t shownum(void);  int main()  {  // 初始化模块  Beep\_init();  Timer\_Init(); // 初始化 TIM2 作为软件时钟  Key\_Init();  OLED\_Init();  Infrared\_Init(); // 初始化红外传感器  VoiceSensor\_Init(); // 初始化声控传感器  TM1637\_Switch(0);  Timer\_Init1(); // 初始化 TIM3 用于显示保持时间  OLED\_ShowString(1,1,"System Ready");    while(1)  {  // 显示菜单  OLED\_ShowString(1,1,"Key1 select");  OLED\_ShowString(2,1,"Key2 ok");  OLED\_ShowNum(3,1,t,5);  OLED\_ShowNum(3,7,s,5);  OLED\_ShowNum(4,1,t1,5);  OLED\_ShowNum(4,7,s1,5);  TM1637\_Switch(0);    // 现有逻辑  if((uint32\_t)t == (uint32\_t)t1 && (uint32\_t)s == (uint32\_t)s1)  {  // 使用蜂鸣器中断而不是阻塞式 BEEP\_work  BEEP\_Start();  }    Keynum = KEy\_getNUM();  if(Keynum == 2)  {  OLED\_Clear();  OLED\_ShowString(2,1,"1.timer");  OLED\_ShowString(2,8,"2.counter");  OLED\_ShowString(3,1,"3.clock");  OLED\_ShowString(3,8,"4.alarm");  uint8\_t Keyn = keycount();  switch(Keyn)  {  case 1:  {  OLED\_Clear();  while(1)  {  Keynum = KEy\_getNUM();  OLED\_ShowString(1,1,"1.show");  OLED\_ShowString(2,1,"2.set");  if(Keynum == 1)  {  while(1)  {  OLED\_Clear();  OLED\_ShowString(2,1,"1.year");  OLED\_ShowString(2,10,"2.month");  OLED\_ShowString(3,1,"3.day");  OLED\_ShowString(3,10,"4.time");  OLED\_ShowString(4,1,"5.minute");  OLED\_ShowString(4,10,"6.back");  if(S == 1)  {  OLED\_Clear();  break;  }  uint8\_t Km = keycount();  bFlag = 0;  flag = 0;  switch(Km)  {  case 1:  {  OLED\_Clear();  while(1)  {  TM1637\_Init();  Display\_Init();  bFlag = 1;  uint8\_t K = KEy\_getNUM();  OLED\_ShowString(1,1,"show year");  if(flag == 0)  {  flag = 1;  y = shownum();  }  Display\_TubeDataProcess(arr1[y-1]);  if(K == 3)  {  TM1637\_Switch(0);  OLED\_Clear();  break;  }  }  break;  }  case 2:  {  OLED\_Clear();  while(1)  {  TM1637\_Init();  Display\_Init();  bFlag = 2;  uint8\_t K = KEy\_getNUM();  OLED\_ShowString(1,1,"show month");  if(flag == 0)  {  flag = 1;  m = shownum();  }  Display\_TubeDataProcess(arr2[m-1]);  if(K == 3)  {  TM1637\_Switch(0);  OLED\_Clear();  break;  }  }  break;  }  case 3:  {  OLED\_Clear();  while(1)  {  TM1637\_Init();  Display\_Init();  bFlag = 3;  uint8\_t K = KEy\_getNUM();  OLED\_ShowString(1,1,"show day");  if(flag == 0)  {  flag = 1;  d = shownum();  }  Display\_TubeDataProcess(arr3[d-1]);  if(K == 3)  {  TM1637\_Switch(0);  OLED\_Clear();  break;  }  }  break;  }  case 4:  {  OLED\_Clear();  while(1)  {  TM1637\_Init();  Display\_Init();  bFlag = 4;  uint8\_t K = KEy\_getNUM();  OLED\_ShowString(1,1,"show time");  if(flag == 0)  {  flag = 1;  t = shownum();  }  Display\_TubeDataProcess(arr4[t-1]);  if(K == 3)  {  TM1637\_Switch(0);  OLED\_Clear();  break;  }  }  break;  }  case 5:  {  OLED\_Clear();  while(1)  {  TM1637\_Init();  Display\_Init();  bFlag = 5;  uint8\_t K = KEy\_getNUM();  OLED\_ShowString(1,1,"show minute");  if(flag == 0)  {  flag = 1;  s = shownum();  }  Display\_TubeDataProcess(arr5[s-1]);  if(K == 3)  {  TM1637\_Switch(0);  OLED\_Clear();  break;  }    }  break;  }  case 6:  {  OLED\_Clear();  S = 1;  break;  }  }  }  }  if(Keynum == 2)  {  OLED\_Clear();  TM1637\_Init();  Display\_Init();  while(1)  {  Keynum = KEy\_getNUM();  OLED\_ShowString(1,1,"1.show");    OLED\_ShowNum(2,5,J,5);  if(J > 0 && J < 6)  {  Display\_TubeDataProcess(arr1[y-1]);  }  else if(J >= 6 && J < 12)  {  Display\_TubeDatatwo(arr2[m-1], arr3[d-1]);  }  else  {  Display\_TubeDatatwo(arr4[t-1], arr5[s-1]);  }  if(Keynum == 3)  {  OLED\_Clear();  break;  }  }    //show(s,m,d,t,num);  }  if(Keynum == 3)  {  break;  }  }  break;  }  case 2:  {  OLED\_Clear();  Timer\_Init1();  TM1637\_Init();  Display\_Init();  while(1)  {  uint8\_t Km = KEy\_getNUM1();  OLED\_ShowString(2,1,"Num:");  Display\_TubeDatatwo(num1, NUM1);  OLED\_ShowNum(2,5,NUM1,5);  OLED\_ShowNum(3,5,num1,5);  if(Km == 1)  {  TIM\_Cmd(TIM3, ENABLE);  }  if(Km == 2)  {  TIM\_Cmd(TIM3, DISABLE);  }  if(Km == 4)  {  NUM1 = 0;  num1 = 0;  }  if(Km == 3)  {  OLED\_Clear();  TM1637\_Switch(0);  break;  }  }  break;  }  case 3:  {  OLED\_Clear();  while(1)  {  uint8\_t Km = KEy\_getNUM();    OLED\_ShowString(2,1,"1.set time");  OLED\_ShowString(3,1,"2.set minute");  OLED\_ShowString(4,1,"2.back");  bFlag = 0;  flag = 0;  if(S1 == 1)  {  OLED\_Clear();  break;  }  uint8\_t k = keycount();  switch(k)  {  case 1:  {  OLED\_Clear();  while(1)  {  TM1637\_Init();  Display\_Init();  bFlag = 4;    uint8\_t K = KEy\_getNUM();  if(flag == 0)  {  flag = 1;  t1 = shownum();  //O = t1;  }  Display\_TubeDataProcess(arr4[t1-1]);  if(K == 3)  {  TM1637\_Switch(0);  OLED\_Clear();  break;  }  }  break;  }  case 2:  {  OLED\_Clear();  while(1)  {  TM1637\_Init();  Display\_Init();  bFlag = 5;  uint8\_t K = KEy\_getNUM();  if(flag == 0)  {  flag = 1;  s1 = shownum();  //P = s1;  }  Display\_TubeDataProcess(arr5[s1-1]);  if(K == 3)  {  TM1637\_Switch(0);  OLED\_Clear();  break;  }  }  break;  }  case 3:  {  OLED\_Clear();  S1 = 1;  break;    }  }  if(t == t1 && s == s1)  {  BEEP\_Start(); // 使用非阻塞的蜂鸣器  }  if(Km == 3)  {  OLED\_Clear();  break;  }  }  break;  }  case 4:  {  OLED\_Clear();  MQ\_2\_init();  value = MQ2\_getData();  while(1)  {  OLED\_ShowString(3,8,"alarm");  if(value)  {  BEEP\_Start(); // 使用非阻塞的蜂鸣器  }  }    break;  }  case 5:  {  OLED\_Clear();  break;  }  }  }  // 时间管理和传感器检测  uint8\_t current\_hour = SysClock.hour;  bool is\_daytime = (current\_hour >= 6 && current\_hour < 18);    if (is\_daytime)  {  // 白天：启用红外传感器  if (Read\_STATE()) // 检测到人体  {  if (!display\_on)  {  OLED\_Clear();  OLED\_ShowString(1, 1, "Person Detected");  TM1637\_DisplayNumber(1); // 假设 1 代表红外触发  display\_on = true;  TIM3\_Start(); // 启动显示保持计时器  }  }  }  else  {  // 夜晚：启用声控传感器  if (VoiceSensor\_Get() == 1) // 假设高电平表示声控达到阈值  {  if (!display\_on)  {  OLED\_Clear();  OLED\_ShowString(1, 1, "Sound Detected");  TM1637\_DisplayNumber(2); // 假设 2 代表声控触发  display\_on = true;  TIM3\_Start(); // 启动显示保持计时器  }  }  }    // 控制显示屏和数码管保持1分钟  if (display\_on && TIM3\_HasElapsed())  {  OLED\_Clear();  TM1637\_Clear();  display\_on = false;  OLED\_ShowString(1, 1, "System Ready");  }  Delay\_ms(100); // 稍作延时，降低CPU占用  }  }  // 不要在 main.c 中重新定义 TIM4\_IRQHandler  // TIM2\_IRQHandler 已在 timer.c 中定义  // TIM4\_IRQHandler 已在 beep.c 中定义  // key.c  #include "KEY.h"  #include "delay.h"  // 初始化按键引脚  void Key\_Init(void)  {  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_gpiox, ENABLE); // 开启GPIOA的时钟  GPIO\_InitTypeDef GPIO\_InitStructure;  GPIO\_InitStructure.GPIO\_Mode = GPIO\_KEY\_mode;  GPIO\_InitStructure.GPIO\_Pin = GPIO\_PINC | GPIO\_PIND | GPIO\_PINE;  GPIO\_InitStructure.GPIO\_Speed = GPIO\_speed;  GPIO\_Init(GPIO\_PORTa, &GPIO\_InitStructure);  }  // 获取按键编号（带防抖）  uint8\_t KEy\_getNUM(void)  {  uint8\_t Num = 0;  if(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PINC) == 0)  {  Delay\_ms(30); // 防抖延时  while(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PINC) == 0);  Delay\_ms(30);  Num = 1;  }  if(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PIND) == 0)  {  Delay\_ms(30);  while(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PIND) == 0);  Delay\_ms(30);  Num = 2;  }  if(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PINE) == 0)  {  Delay\_ms(30);  while(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PINE) == 0);  Delay\_ms(30);  Num = 3;  }  return Num;  }  // 获取按键编号（不带防抖）  uint8\_t KEy\_getNUM1(void)  {  uint8\_t Num = 0;  if(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PINC) == 0)  {  Num = 1;  }  if(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PIND) == 0)  {  Num = 2;  }  if(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PINE) == 0)  {  Num = 3;  }  if(GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PINC) == 0 && GPIO\_ReadInputDataBit(GPIO\_PORTa, GPIO\_PIND) == 0)  {  Num = 4;  }  return Num;  }  // 获取按键计数（示例实现）  uint8\_t keycount(void)  {  return KEy\_getNUM(); // 示例：直接返回 KEy\_getNUM 的结果  }  // 显示数字（示例实现）  uint16\_t shownum(void)  {  // 实现 shownum 的逻辑  // 这里提供一个示例实现，具体根据您的需求调整  // 例如，可以从某个输入获取一个数字，或基于按键输入进行计数  // 目前返回一个固定值，请根据需要修改  return 0; // 示例返回值  }  // 1637.c  #include <string.h> // 提供了一些用于处理字符串和内存块的函数。  #include <stdint.h> // 定义了具有确定大小的整数类型。  #include <stdbool.h> // 提供布尔类型支持。  #define TUBE\_DISPLAY\_NULL 26  #define TUBE\_DISPLAY\_DECIMAL\_PIONT\_OFFSET 16  #include "stm32f10x.h"  #include "1637.h"  #include "delay.h"  TM1637Tube\_ts sDisplayData;  const uint8\_t u8NumTab[] =  {  //0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F,  0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,  //0., 1., 2., 3., 4., 5., 6., 7., 8., 9. Null  0xBF,0x86,0xDB,0xCF,0xE6,0xED,0xFD,0x87,0xFF,0xEF,0x00  };  // 最左至最右数码管 ，依次为0-3号，对应的显示寄存器地址  const uint8\_t u8TubeAddrTab[] =  {  0xC0,0xC1,0xC2,0xC3  };  void TM1637\_Init(void)  {  /\*开启时钟\*/  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE); //开启GPIOB的时钟  /\*GPIO初始化\*/  GPIO\_InitTypeDef GPIO\_InitStructure;  GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_OD;  GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_10 | GPIO\_Pin\_11;  GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOB, &GPIO\_InitStructure); //将PB10和PB11引脚初始化为开漏输出  /\*设置默认电平\*/  GPIO\_SetBits(GPIOB, GPIO\_Pin\_10 | GPIO\_Pin\_11); //设置PB10和PB11引脚初始化后默认为高电平（释放总线状态）  }  /\*\*  \* 函数：I2C写SCL引脚电平  \* 参数：BitValue 协议层传入的当前需要写入SCL的电平，范围0~1  \* 返回值：无  \* 注意事项：当BitValue为0时，置SCL为低电平；当BitValue为1时，置SCL为高电平  \*/  void TM1637\_W\_SCL(uint8\_t BitValue)  {  GPIO\_WriteBit(GPIOB, GPIO\_Pin\_10, (BitAction)BitValue); //根据BitValue，设置SCL引脚的电平  Delay\_us(10); //延时10us，防止时序频率超过要求  }  /\*\*  \* 函数：I2C写SDA引脚电平  \* 参数：BitValue 协议层传入的当前需要写入SDA的电平，范围0~1  \* 返回值：无  \* 注意事项：当BitValue为0时，置SDA为低电平；当BitValue为1时，置SDA为高电平  \*/  void TM1637\_W\_SDA(uint8\_t BitValue)  {  GPIO\_WriteBit(GPIOB, GPIO\_Pin\_11, (BitAction)BitValue); //根据BitValue，设置SDA引脚的电平，BitValue要实现非0即1的特性  Delay\_us(10); //延时10us，防止时序频率超过要求  }  /\*\*  \* 函数：I2C读SDA引脚电平  \* 参数：无  \* 返回值：协议层需要得到的当前SDA的电平，范围0~1  \* 注意事项：当前SDA为低电平时，返回0；当前SDA为高电平时，返回1  \*/  uint8\_t TM1637\_R\_SDA(void)  {  uint8\_t BitValue;  BitValue = GPIO\_ReadInputDataBit(GPIOB, GPIO\_Pin\_11); //读取SDA电平  Delay\_us(10); //延时10us，防止时序频率超过要求  return BitValue; //返回SDA电平  }  void TM1637\_Start(void)  {  TM1637\_W\_SDA(1); //释放SDA，确保SDA为高电平  TM1637\_W\_SCL(1); //释放SCL，确保SCL为高电平  TM1637\_W\_SDA(0); //在SCL高电平期间，拉低SDA，产生起始信号  TM1637\_W\_SCL(0); //起始后把SCL也拉低，即为了占用总线，也为了方便总线时序的拼接  }  void TM1637\_Stop(void)  {  TM1637\_W\_SCL(0); //拉低SCL，确保SCL为低电平  TM1637\_W\_SDA(0); //拉低SDA，确保SDA为低电平  TM1637\_W\_SCL(1); //释放SCL，使SCL呈现高电平  TM1637\_W\_SDA(1); //在SCL高电平期间，释放SDA，产生终止信号  }  // 应答信号  uint8\_t TM1637\_ReceiveAck(void)  {  uint8\_t AckBit; //定义应答位变量  TM1637\_W\_SDA(1); //接收前，主机先确保释放SDA，避免干扰从机的数据发送  TM1637\_W\_SCL(1); //释放SCL，主机在SCL高电平期间读取SDA  AckBit = TM1637\_R\_SDA(); //将应答位存储到变量里  TM1637\_W\_SCL(0); //拉低SCL，开始下一个时序模块  return AckBit; //返回应答位变量  }  // 向 TM1637 发送一个字节数据  void TM1637\_Write\_Byte(uint8\_t data)  {  uint8\_t i;  TM1637\_W\_SCL(0); // 设置时钟线 SCL 为低电平，准备开始发送数据  // 循环发送 8 位数据  for (i = 0; i < 8; i++)  {  // 根据数据的最低位决定数据线 SDA 的状态  if(data & 0x01)  {  TM1637\_W\_SDA(1); // 发送 1  }  else  {  TM1637\_W\_SDA(0); // 发送 0  }    data = data >> 1; // 右移数据，处理下一个位    TM1637\_W\_SCL(1); // 拉高时钟线，表示位数据有效  TM1637\_W\_SCL(0); // 拉低时钟线，准备发送下一个位  }  }  // 向 TM1637 数码管写入命令  void TM1637\_WriteCmd(uint8\_t u8Cmd)  {  // 开始信号，表示通信开始  TM1637\_Start();    // 发送命令字节  TM1637\_Write\_Byte(u8Cmd);    // 接收确认应答  TM1637\_ReceiveAck();    // 停止信号，表示通信结束  TM1637\_Stop();  }  // 向指定地址写入数据  // u8Addr: 要写入的地址  // u8Data: 要写入的数据  void TM1637\_WriteData(uint8\_t u8Addr, uint8\_t u8Data)  {  // 开始信号，表示通信开始  TM1637\_Start();    // 发送地址字节  TM1637\_Write\_Byte(u8Addr);    // 接收确认应答  TM1637\_ReceiveAck();    // 发送数据字节  TM1637\_Write\_Byte(u8Data);    // 接收确认应答  TM1637\_ReceiveAck();    // 停止信号，表示通信结束  TM1637\_Stop();  }  // 在四个数码管上显示数据  // sData: 显示数据结构体，包含四个数码管的显示值  void TM1637\_TubeDisplay(TM1637Tube\_ts sData)  {  uint8\_t temp[4], i;    // 根据 sData 中的值，从 u8NumTab 数组中获取对应的段码  temp[0] = u8NumTab[sData.tube0]; // 获取第一个数码管的段码  temp[1] = u8NumTab[sData.tube1]; // 获取第二个数码管的段码  temp[2] = u8NumTab[sData.tube2]; // 获取第三个数码管的段码  temp[3] = u8NumTab[sData.tube3]; // 获取第四个数码管的段码    // 遍历四个数码管，逐个写入数据  for (i = 0; i < 4; i++)  {  // 向每个数码管的地址写入对应的段码  TM1637\_WriteData(u8TubeAddrTab[i], temp[i]);  }  }  // 设置亮度  // 0x88为开显示，u8Brt亮度  void TM1637\_SetBrightness(uint8\_t u8Brt)  {  TM1637\_WriteCmd(0x88 | u8Brt);  }  // 显示开关  // 0x88为开显示，0x80关显示  void TM1637\_Switch(bool bState)  {  bState ? TM1637\_WriteCmd(0x88) : TM1637\_WriteCmd(0x80);  }  // 显示数字  void TM1637\_DisplayNumber(uint8\_t num)  {  // 将数字转换为显示编码并发送到数码管  // 这里假设显示数字在最低四位  Display\_TubeDataProcess(num);  }  // 清除数码管显示  void TM1637\_Clear(void)  {  // 清除数码管显示的代码  Display\_TubeDataProcess(0); // 显示0  }  void Display\_Init(void)  {  TM1637\_Switch(0); // 关显示  TM1637\_SetBrightness(0x87); // 设置亮度，开显示  TM1637\_WriteCmd(0x44); // 写数据到寄存器，固定地址模式  memset(&sDisplayData, 0xFF, sizeof(sDisplayData));  }  void Display\_TubeDataProcess(uint16\_t u16Data)  {  memset(&sDisplayData, 0xFF, sizeof(sDisplayData));  if (u16Data > 9999)  {  u16Data = 9999; // 最多四位数  }  if (u16Data > 999) // 四位数  {  sDisplayData.tube0 = (uint8\_t)(u16Data / 1000); // 千位  sDisplayData.tube1 = (uint8\_t)(u16Data / 100 % 10); // 百位  sDisplayData.tube2 = (uint8\_t)(u16Data / 10 % 10); // 十位  sDisplayData.tube3 = (uint8\_t)(u16Data % 10); // 个位  }  else if (u16Data > 99) // 三位数  {  sDisplayData.tube0 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube1 = (uint8\_t)(u16Data / 100); // 百位  sDisplayData.tube2 = (uint8\_t)(u16Data / 10 % 10); // 十位  sDisplayData.tube3 = (uint8\_t)(u16Data % 10); // 个位  }  else if (u16Data > 9) // 两位数  {  sDisplayData.tube0 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube1 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube2 = (uint8\_t)(u16Data / 10); // 十位  sDisplayData.tube3 = (uint8\_t)(u16Data % 10); // 个位  }  else // 一位数  {  sDisplayData.tube0 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube1 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube2 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube3 = (uint8\_t)u16Data; // 个位  }  TM1637\_TubeDisplay(sDisplayData);  }  void Display\_TubeDatafirst(uint16\_t u16Data)  {  memset(&sDisplayData, 0xFF, sizeof(sDisplayData));  if (u16Data > 99)  {  u16Data = 99; // 最多两位数  }    if (u16Data > 9) // 两位数  {  sDisplayData.tube0 = (uint8\_t)(u16Data / 10); // 十位  sDisplayData.tube1 = (uint8\_t)(u16Data % 10); // 个位  sDisplayData.tube2 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube3 = TUBE\_DISPLAY\_NULL; // 不显示  }  else // 一位数  {  sDisplayData.tube0 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube1 = (uint8\_t)u16Data; // 个位  sDisplayData.tube2 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube3 = TUBE\_DISPLAY\_NULL; // 不显示  }  TM1637\_TubeDisplay(sDisplayData);  }  void Display\_TubeDatalast(uint16\_t u16Data)  {  memset(&sDisplayData, 0xFF, sizeof(sDisplayData));  if (u16Data > 99)  {  u16Data = 99; // 最多两位数  }    if (u16Data > 9) // 两位数  {  sDisplayData.tube0 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube1 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube2 = (uint8\_t)(u16Data / 10); // 十位  sDisplayData.tube3 = (uint8\_t)(u16Data % 10); // 个位  }  else // 一位数  {  sDisplayData.tube0 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube1 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube2 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube3 = (uint8\_t)u16Data; // 个位  }  TM1637\_TubeDisplay(sDisplayData);  }  void Display\_TubeDatatwo(uint16\_t u16Data1, uint16\_t u16Data2)  {  // Display\_TubeDatalast(u16Data2);  // Display\_TubeDatafirst(u16Data1);  memset(&sDisplayData, 0xFF, sizeof(sDisplayData));  if (u16Data1 > 99 || u16Data2 > 99)  {  u16Data1 = 99; // 最多两位数  u16Data2 = 99;  }    if (u16Data1 > 9 || u16Data2 > 9) // 两位数  {  sDisplayData.tube0 = (uint8\_t)(u16Data1 / 10); // 十位  sDisplayData.tube1 = (uint8\_t)(u16Data1 % 10); // 个位  sDisplayData.tube2 = (uint8\_t)(u16Data2 / 10); // 十位  sDisplayData.tube3 = (uint8\_t)(u16Data2 % 10); // 个位  }  else // 一位数  {  sDisplayData.tube0 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube1 = (uint8\_t)u16Data1; // 个位  sDisplayData.tube2 = TUBE\_DISPLAY\_NULL; // 不显示  sDisplayData.tube3 = (uint8\_t)u16Data2; // 个位  }  TM1637\_TubeDisplay(sDisplayData);  } |

1. 参考文献

[1]孙官正,张晓新.基于STM32单片机的舒适家居系统[J].电子元器件与信息技术,2024,8(06):4-7+11.DOI:10.19772/j.cnki.2096-4455.2024.6.002.

[2]查启明.基于STM32的非接触式红外体温检测系统设计[J].现代信息科技,2024,8(16):157-161+167.DOI:10.19850/j.cnki.2096-4706.2024.16.033.

[3]李金成,刘楠.基于STM32的人体红外测温及火灾预警系统的设计[J].工业控制计算机,2021,34(09):134-136.

[4]梁成功,崔梦瑜,边鑫荣,等.基于STM32单片机的智能手环设计与实现[J].中国新技术新产品,2024,(22):32-35.DOI:10.13612/j.cnki.cntp.2024.22.006.

[5]引用的CSDN文章：

1. 标题：STM32单片机开发入门(一)STM32F103C8T6小系统板电路原理图分析

原文链接：https://blog.csdn.net/zy2232652/article/details/139349843

1. STM32学习记录——声音传感器的使用

原文链接：<https://blog.csdn.net/m0_73629902/article/details/130231446>

1. 标题：SR501人体红外模块

原文链接：https://blog.csdn.net/wuyiyu\_/article/details/131880151

1. 标题：【HC-SR501人体红外传感器】

原文链接：https://blog.csdn.net/boybs/article/details/124576299

1. 标题：MQ-2烟雾浓度传感器

原文链接：https://blog.csdn.net/Dustinthewine/article/details/126543605