课 程 报 告

年 级 专 业：2022 级电子信息工程

学 生 姓 名：刘嘉俊

学　　号：20223004426

课 程 名 称：嵌入式系统设计

任 课 老 师：王庆吉

分　　数：

海南大学·信息与通信工程学院

School of Information and Communication Engineering, Hainan University

基于 LPC1114 的温度记录仪设计

摘 要

本文详细介绍了基于 LPC1114 微控制器的温度记录仪设计。该设计旨在综合运用嵌入式软硬件知识，实现环境温度的实时采集、存储、显示与报警功能。硬件方面，以 LPC1114 为核心，集成了 LM75BD 温度传感器、XT25 闪存、DS1307 时钟、OLED 显示屏、蓝牙串口、蜂鸣器等模块，并采用模块化设计与紧凑 PCB 布局，支持 USB 和电池双供电。软件上，基于 C 语言在 Keil MDK 环境开发，通过定时器中断实现每秒温度采集、显示刷新与数据存储，利用中位数滤波算法处理温度数据，同时具备串口通信、温度超限报警及 RGB 灯指示功能。经测试，该温度记录仪能在 -55℃至 125℃范围内精确采集温度（精度达 ±1.5℃），存储容量不低于 2MB，且在硬件与软件协同工作上稳定可靠，有效解决了温度监测与记录的实际应用需求，同时也针对开发过程中的问题进行了优化与改进，为后续嵌入式系统设计积累了宝贵经验。。

目录

基于 LPC1114 的温度记录仪设计

刘嘉俊
2022 级电子信息工程，20223004426

一、设计任务

1.1 设计目的

本次课程设计的目的是基于嵌入式系统的学习内容，完成一款功能完整的温度记录仪设计，综合运用嵌入式软硬件知识，提高理论与实践结合的能力。通过使用 NXP LPC1114 微控制器，结合多种外围设备（如温度传感器、OLED 显示屏、RTC 时钟模块等），实现环境温度的实时采集、存储和显示功能。该设计目的体现在以下几个方面：

理论实践结合：将课堂知识（如嵌入式系统架构、通信接口）与实际设计需求相结合，锻炼系统设计和问题解决能力。

软硬件协同开发：通过硬件电路设计和嵌入式程序开发，强化对嵌入式系统完整开发流程的理解。

创新与应用：为温度监测和数据记录的实际应用提供解决方案，同时为进一步扩展和优化设计奠定基础。

个人能力提升：提升独立完成项目设计、编程调试及文档编写能力，为后续深入学习和实际工程开发做好准备。

1.2 设计内容

本项目设计基于 NXP LPC1114 微控制器及其开发板，完成一款集温度采集、存储、显示及报警功能为一体的嵌入式温度记录仪，主要设计内容包括以下几个部分：

硬件电路设计：

设计符合项目功能需求的电路原理图，包括温度传感器模块、存储模块、时钟模块、显示模块及电源管理模块等。确保电路设计合理、美观，符合电气连接规范，完成 PCB 布局及布线。

软件功能开发：

基于 C 语言在 Keil MDK 开发环境中进行软件编程，完成温度数据的采集、滤波处理、数据存储及实时显示。

实现蜂鸣器报警功能及温度范围指示灯状态变化。

通过 UART 接口或者蓝牙传输实现与外部设备的数据通信。

功能实现：

实时采集环境温度（范围：-55℃~125℃，精度：±1.5℃）。

将采集到的温度数据与当前时间关联，并周期性存储到外部闪存。

通过 0.96 寸 OLED 显示屏显示温度、时间及日期。

温度超限时启动蜂鸣器报警，并通过 RGB 灯指示当前温度范围。

对设计的硬件电路进行功能验证，确保模块正常运行。对软件功能进行调试，验证采集数据的准确性及系统稳定性。最终设计成果是基于 LPC1114 的温度记录仪系统，能够完成预期的功能需求，并满足技术指标，包括温度采集范围、数据存储空间、电路尺寸及系统稳定性等。

二、项目需求分析

2.1 项目要求

基本功能需求：

（1）每秒检测一次环境温度并存储。

（2）存储的数据与时间相关联，可通过串口导出。

（3）支持通过串口设置和读取当前时间。

（4）提供运行状态指示灯。

（5）具备超温报警功能（声音报警和视觉提示）。

电源供电要求：

（6）支持 USB 供电与电池供电双模式。

（7）在电池供电模式下，确保断电时钟保持正常运行。

接口及模块扩展：

（8）USB 转串口功能，用于与上位机进行通信和数据调试。

（9）预留串口通信和蓝牙通信接口，扩展远程数据传输能力。

（10）预留 OLED 显示屏接口，方便现场查看温度数据和时间。

2.2 项目性能指标说明

本项目的温度记录仪需要达到以下性能指标：

（1）温度检测性能：

检测范围：-55℃~+125℃。

检测精度：±1.5℃。

（2）数据存储性能：

存储空间不低于 2MB，可存储大量温度记录。

（3）硬件设计性能：

电路板尺寸控制在 8cm × 3cm 以内，便于集成。

电路布局合理，符合电气规范。

（4）显示与交互性能：

支持实时显示当前温度、日期和时间。

温度超出阈值时，通过蜂鸣器报警和 RGB 指示灯提示。

（5）通信性能：

支持 UART 串口通信，波特率为 115200，便于数据传输和调试。

数据输出格式清晰，易于外部设备解析。

三、项目方案设计与思路

3.1 设计思路

本项目以 NXP LPC1114 微控制器为核心，结合外部传感器模块、存储模块、显示模块、通信模块和报警模块，构建了一款功能齐全的温度记录仪。整个系统按照功能划分为硬件设计和软件设计两个部分，分别实现环境温度数据的实时采集、存储、显示和报警。

整体设计思路分为硬件设计思路和软件设计思路：

硬件设计思路：模块化设计：硬件部分以功能模块为单位进行设计，包括温度采集、数据存储、时钟管理、显示控制和电源管理模块，各模块通过 I2C 和 SPI 接口与核心微控制器连接，形成整体系统。紧凑的 PCB 设计：电路板布局控制在 8cm × 3cm 的尺寸内，确保紧凑性与功能完整性。双供电设计：支持 USB 和 18650 锂电池两种供电模式，确保系统在断电状态下仍能保持时钟运行。

软件设计思路：分模块开发：通过 C 语言开发，每个功能模块（如温度采集、存储、显示和通信）独立实现，并在主程序或者定时器定时中断中统一调度。实时任务处理：通过定时器中断实现定时采集、显示刷新和数据存储。异常状态处理：实现温度超限报警（蜂鸣器+RGB 指示灯）以及串口通信功能，方便调试和数据导出。

详细功能设计思路：

（1）温度采集模块：使用 LM75BD 温度传感器通过 I2C 接口定时采集环境温度，温度数据范围为 -55℃~+125℃，精度为 ±1.5℃。在软件中对温度数据

进行 中位数平均滤波 处理，去除异常值，提高数据的稳定性。额外使用 DS18B20 温度传感器作为备选传感器，支持单总线通信。

（2）数据存储模块：使用 XT25 SPI 闪存模块存储温度数据，并与 RTC 提供的时间戳关联。实现对闪存的擦除、写入和读取操作，定时将温度数据保存到存储器中，确保数据持久化。

（3）时钟模块：使用 DS1307 RTC 模块 提供日期和时间功能。通过 I2C 接口读取时间，并在断电情况下依赖电池供电保持时钟运行。支持通过 UART 接口设置和校准时间。

（4）显示模块：使用 0.96 寸 OLED 显示屏通过 I2C 接口实时显示当前的时间、日期和温度。软件代码中通过专用显示函数更新屏幕内容，提供清晰的数据信息展示。

（5）报警模块：使用 蜂鸣器和 RGB 指示灯提供视觉和听觉报警提示：当温度超过 30℃ 时，蜂鸣器发出报警声，并点亮 RGB 指示灯。根据温度范围（010℃、1020℃、20~30℃）动态切换指示灯颜色。PWM 控制蜂鸣器实现不同频率的声音报警。

（6）通信模块：使用 CH340 转串口模块实现 USB 转 UART 功能，与上位机进行数据传输。系统通过串口周期性输出当前的时间、日期和温度数据，方便用户获取信息。支持通过串口输入命令设置系统时间或校准温度。

（7）电源管理模块：支持 USB 和 18650 锂电池供电模式，并通过 AMS1117 稳压模块输出稳定的 3.3V 电压。配置电源滤波电路，确保系统稳定运行。

（8）系统控制单元：以 NXP LPC1114 微控制器为核心，通过外设接口（I2C、SPI、UART）连接所有外围模块。主程序通过定时器中断，每秒完成一次温度采集、显示刷新和数据存储，同时检测温度状态并触发报警功能。

3.2 系统框图

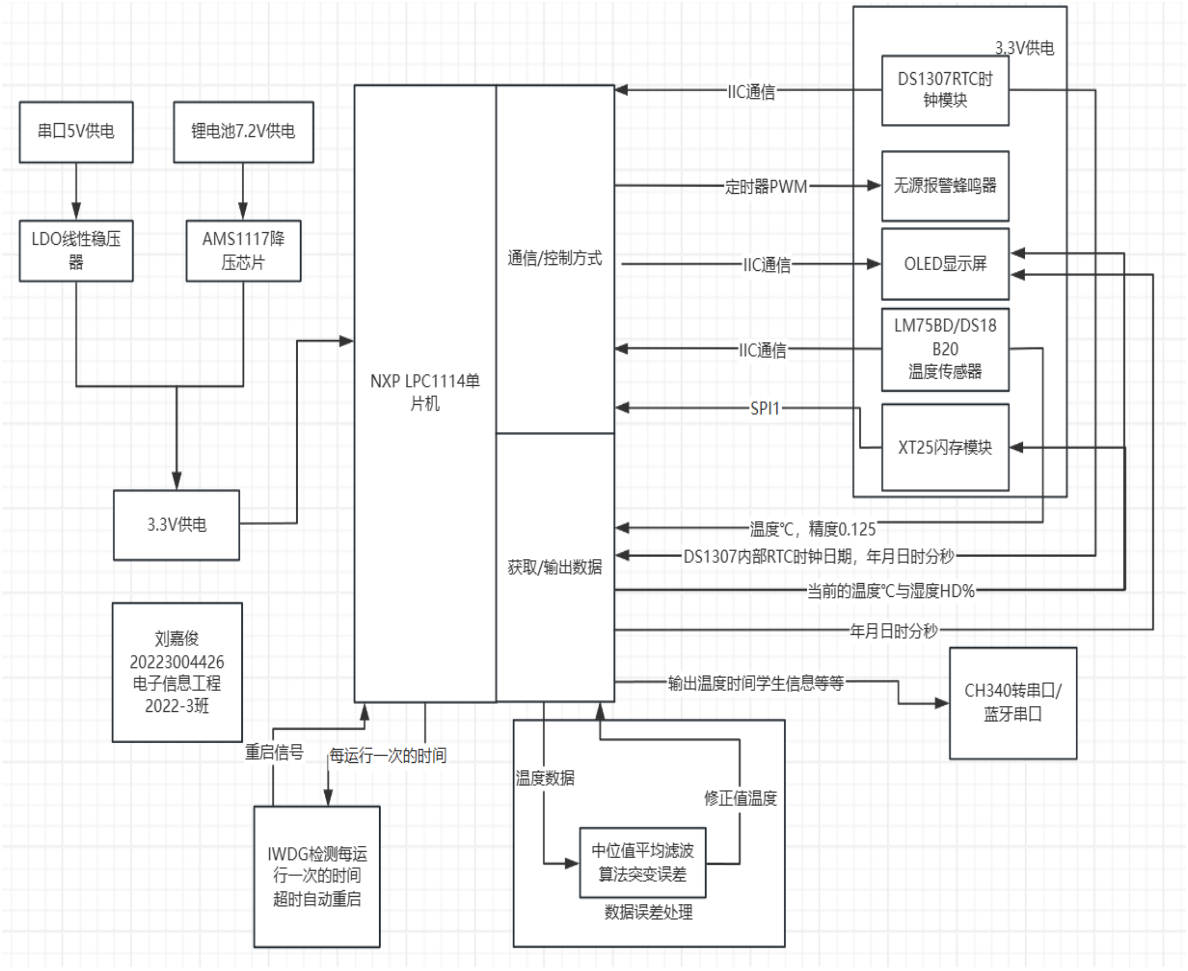


图 1 项目系统框图

3.3 系统控制单元

图 2 NXP LPC1114 芯片

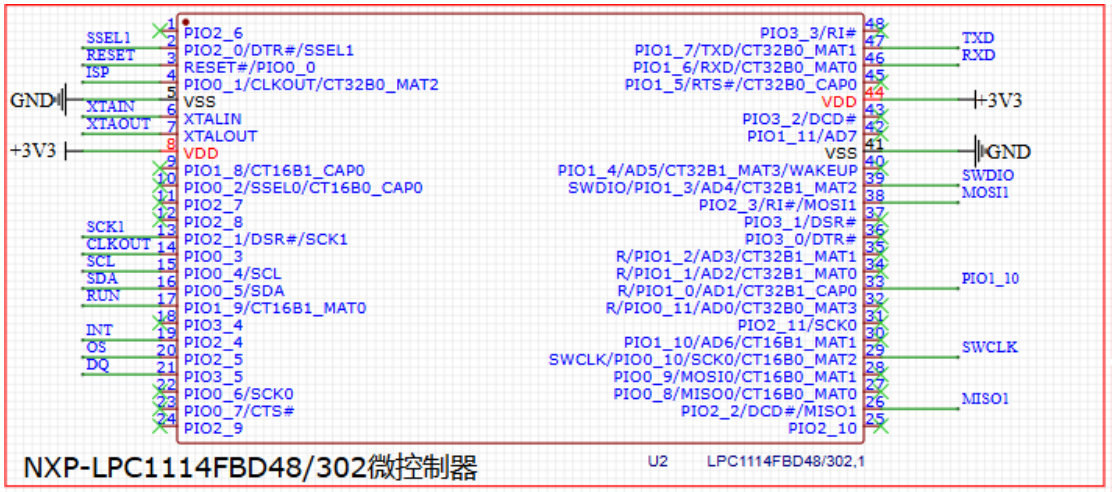


图 3 NXP LPC1114 芯片引脚分布图

LPC1114 是 NXP 公司推出的一款 ARM Cortex-M0 内核的 32 位单片机。它的主频最大可达 50MHz，内部集成时钟产生单元，不用外部晶振也可以工作。内部集成 32KB FALSH 程序存储器、8K SRAM 数据存储器、一个快速 I2C 接口、一个 RS485/EIA485 UART、两个带 SSP 特征的 SPI 接口、4 个通用定时器、1 个系统定时器、1 个带窗口功能的看门狗定时器、功耗管理模块、1 个 ADC 模块和 42 个 GPIO。截至 Ration 写稿时，一片 LPC1114 的零售价只需 3 元。

3.4 温度传感器单元

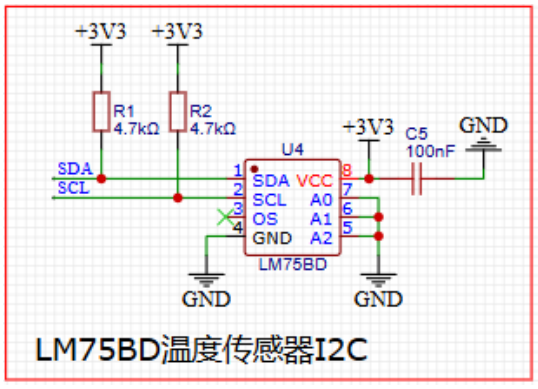


图 4 LM75BD 温度传感器　　图 5 LM75BD 温度传感器硬件设计

LM75BD 是一款由 NXP 半导体公司推出的高精度数字温度传感器，支持 I2C 通信接口，可实现环境温度的实时采集并以数字形式输出。该传感器能够在 -55℃至 +125℃ 的范围内工作，测量精度达到 ±1.5℃，非常适合需要精确温度监测的嵌入式应用场景。LM75BD 内置温度寄存器，可快速完成温度转换，并通过简单的 I2C 协议向微控制器传输温度数据，无需复杂的信号调理电路，具有高集成度和易用性。在本项目中，LM75BD 温度传感器用于采集环境温度数据，数据

通过 I2C 接口传输至 LPC1114 微控制器，经过滤波算法处理后，显示在 OLED 屏幕上，同时用于存储和报警功能。LM75BD 的高精度和快速响应特性，为温度记录仪提供了可靠的基础支持。

Symbol	Pin	Description
SDA	1	Digital I/O. I²C-bus serial bidirectional data line; open-drain.
SCL	2	Digital input. I²C-bus serial clock input.
OS	3	Overtemp Shutdown output; open-drain.
GND	4	Ground. To be connected to the system ground.
A2	5	Digital input. User-defined address bit 2.
A1	6	Digital input. User-defined address bit 1.
A0	7	Digital input. User-defined address bit 0.
V_{CC}	8	Power supply.

图 6 LM75BD 引脚功能图

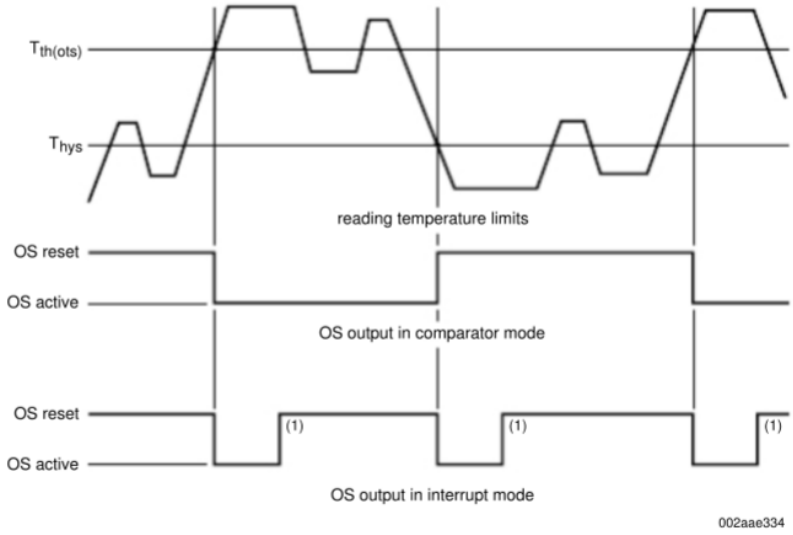


图 7 LM75BD 引脚功能图操作原理图

LM75BD 温度传感器通过 I2C 接口通信，其核心操作是将温度传感器采集的 11 位数据存储到温度寄存器中，并支持实时读取和比较操作。温度传感器在采集到的温度数据后，将其存储为 11 位的二进制补码格式，并通过 I2C 总线传输给微控制器读取。每次读取操作不会影响传感器的正常运行状态。LM75BD 支持用户设置的高温和低温阈值（对应的 Tos 和 Thyst 寄存器），传感器会将实时温度与这些阈值进行比较，以判断是否触发 OS 输出信号。

在正常工作模式下，LM75BD 会定时将温度寄存器中的实时温度数据与高温阈值（Tos）和低温迟滞阈值（Thyst）进行比较。当温度超过 Tos 时，OS 输出信号会根据配置寄存器中的设置触发报警状态；当温度降至低于 Thyst 时，OS 输出信号将恢复到正常状态。这种迟滞机制避免了由于温度波动导致的报警信号频繁切换。Tos 和 Thyst 寄存器的值以 9 位二进制补码形式存储，用户可以通过 I2C 接口对其进行读写操作。

此外，LM75BD 提供两种 OS 输出模式：比较器模式和中断模式。OS 输出响应模式由配置寄存器中的特定位选择，用户还可以通过配置寄存器定义故障队列（Fault Queue），即在触发 OS 输出之前需要满足的连续故障次数。这一机制进一步增强了传感器的抗干扰能力，确保报警信号的可靠性

3.5 FLASH 存储单元

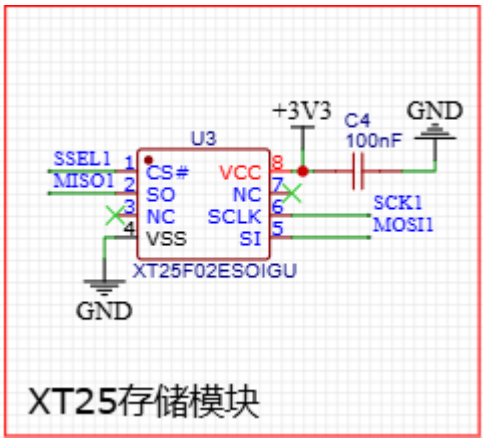
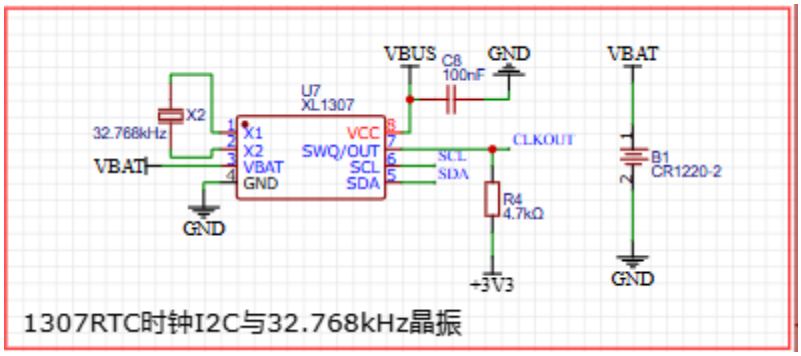


图 8 XT25 闪存模块

本项目的 Flash 存储单元使用 XT25 SPI 闪存模块，主要用于存储实时采集的温度数据并与时间戳关联，实现数据的长期保存。通过 SPI 接口与 LPC1114 微控制器通信，XT25 支持快速的数据写入、读取和擦除操作，容量达到 2MB 以上，能够满足项目对存储空间和操作速度的需求。存储单元通过状态寄存器监控操作状态，并提供写入保护机制，确保数据的完整性和可靠性。该存储单元定期存储温度数据与时间信息，为温度记录仪提供了稳定、高效的数据存储支持。

3.6 DS1307 时钟单元



DS1307 是一款基于 IIC 总线接口的实时时钟芯片，可以独立于 MCU 工作，芯片具有备用电源自动切换功能，可以在主电源掉电或其他一些恶劣环境下保证

系统时钟的准确。

DS1307 具有产生时、分、秒、日、月、年等功能，闰年可自动调整，日历和时钟数据以 BCD 码的方式存放在片内的寄存器上。

片内集成了 56 字节的具有掉电后电池保持的 RAM 数据存储器，可以用来保存一些关键数据，

芯片具有掉电检测和自动切换电池供电功能，当 DS1307 靠后备电池维持工作时，拒绝 CPU 对其的读出和写入操作。

DS1307 片内有多个时间保存寄存器，单片机就是通过读取这些寄存器得到时间和日期相关的数据的，其中有 8 个寄存器专门用来存储时间信息，另外 56 个字节的 RAM 可以供用户自由使用。。

3.7 OLED 显示单元

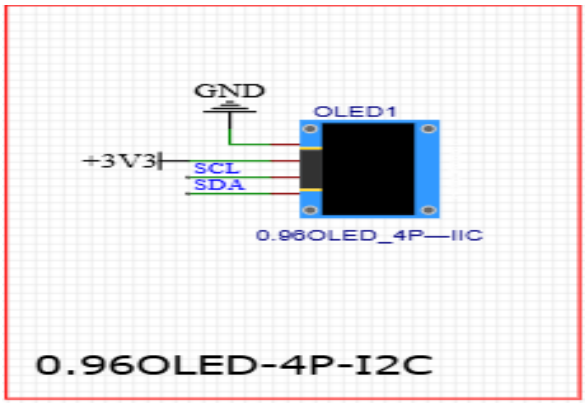
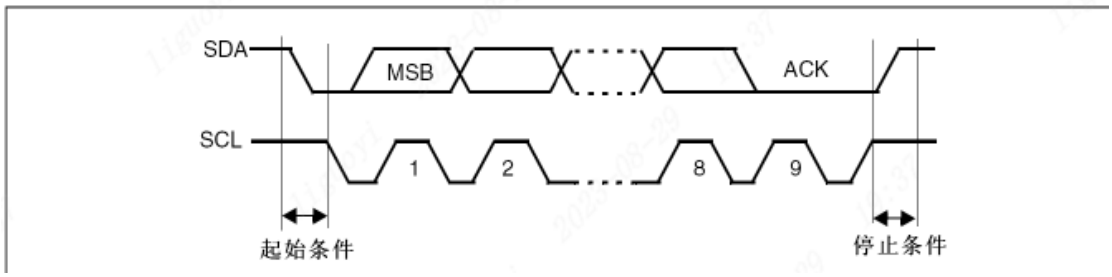


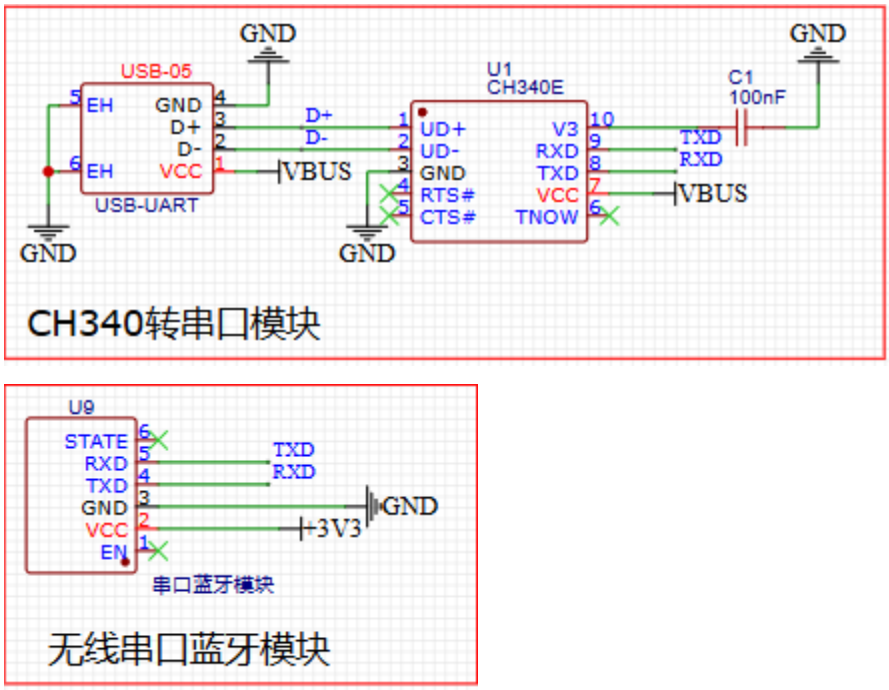
图241　I²C总线协议



IC 总线只需要两条数据线，分别是串行数据线（SDA）和串行时钟线（SCL），在 IIC 协议中，总线上有一个主设备和多个从设备。软件初始化时需配置 I2C 控制器的速率、地址模式、设备地址等参数，接着发送起始信号至 I2C 总线并借由 SBSEND 标志位判断是否发送完毕，10 位地址模式下要清除 SBSEND 位，7 位地址模式则不可清除；若为 10 位地址模式需分两次发送地址并分别清除 ADD10SEND 和 ADDSEND 位，7 位地址模式仅发送一次地址并在 ADDSEND 置一后清除；写入字节数据前要依据 TBE 标志位判断发送寄存器是否为空，发送后借 BTC 标志位判断从机是否应答；最后设置 STOP 发送停止信号以完成整个 I2C

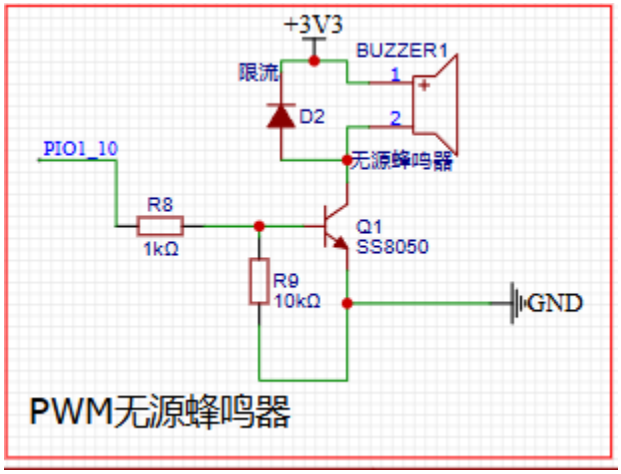
通信过程。

3.8 远程蓝牙串口单元



蓝牙串口是基于 SPP 协议（Serial Port Profile），能在蓝牙设备之间创建串口进行数据传输的一种设备。 蓝牙串口的目的是针对如何在两个不同设备（通信的两端）上的应用之间保证一条完整的通信路径。如 蓝牙模块（BF10-A）和 BF10-A 之间，蓝牙模块和 蓝牙适配器 之间，蓝牙模块和 PDA 蓝牙之间都可以通过 SPP 蓝牙 串行端口 服务来建立蓝牙串口数据传输。

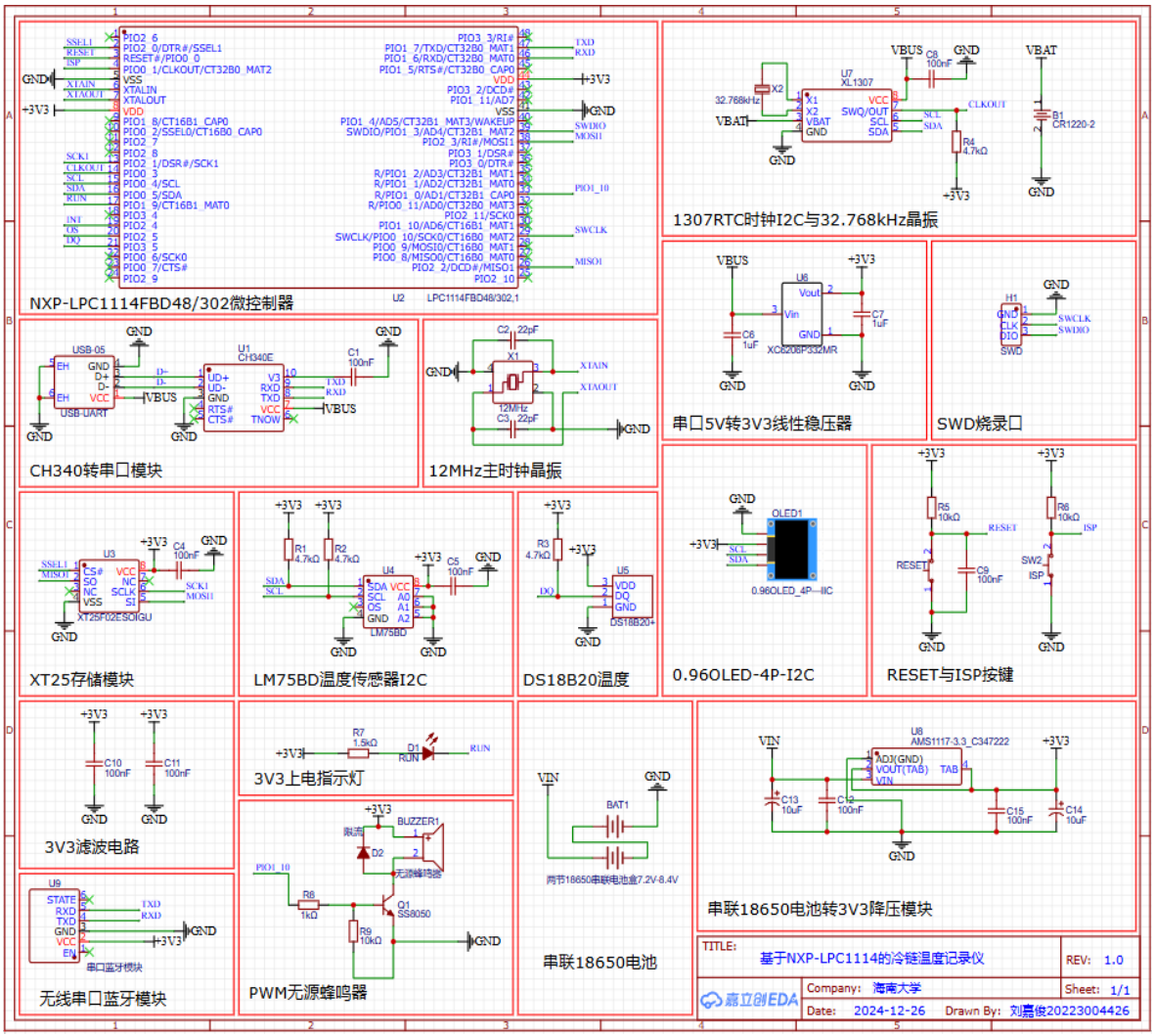
3.9 蜂鸣器报警单元



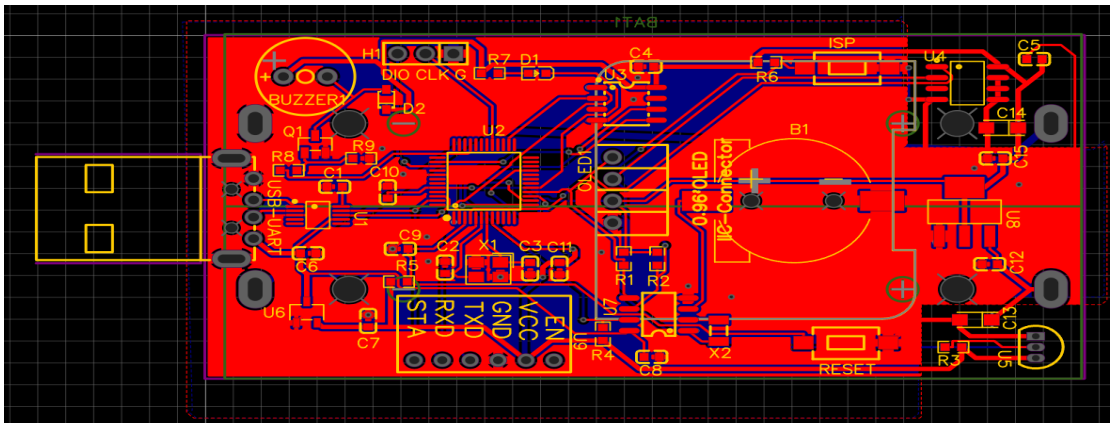
无源蜂鸣器利用电磁感应现象，为音圈接入交变电流后形成的电磁铁与永磁铁相吸或相斥而推动振膜发声，接入直流电只能持续推动振膜而无法产生声音，只能在接通或断开时产生声音。

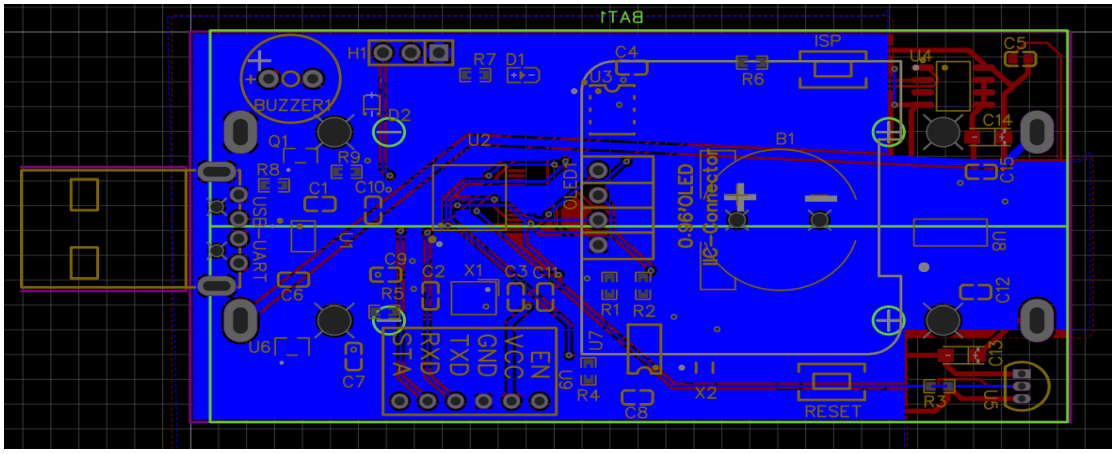
四、项目硬件设计与分析

4.1 项目设计原理图

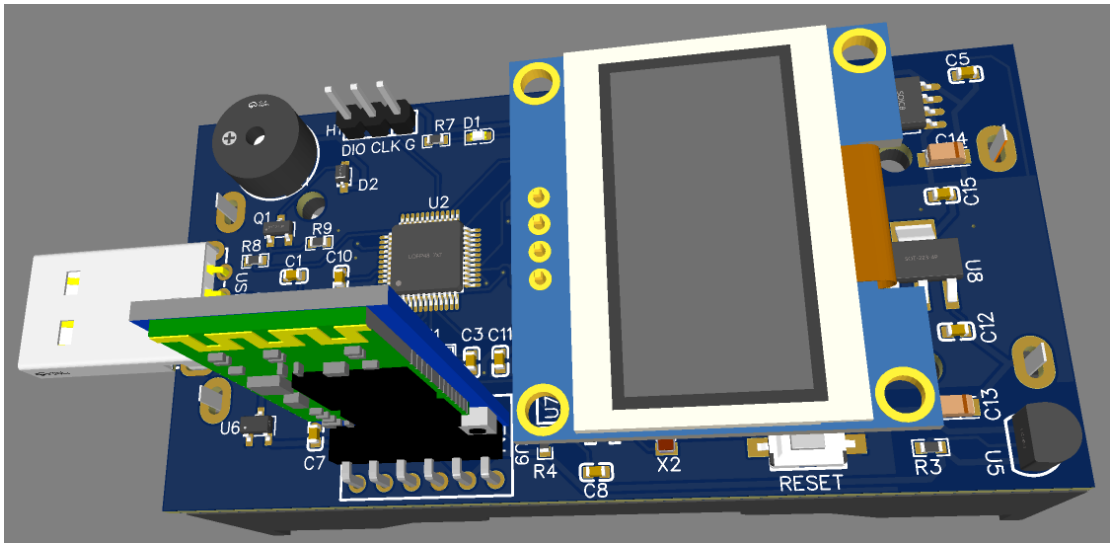


4.2 项目 PCB 图





4.3 项目 3D 图



4.4 项目成本估计与分析

使用嘉立创 EDA 绘制原理图和 PCB 板，领取学生优惠券免费打板，在 PCB 制作上花费 0 元。

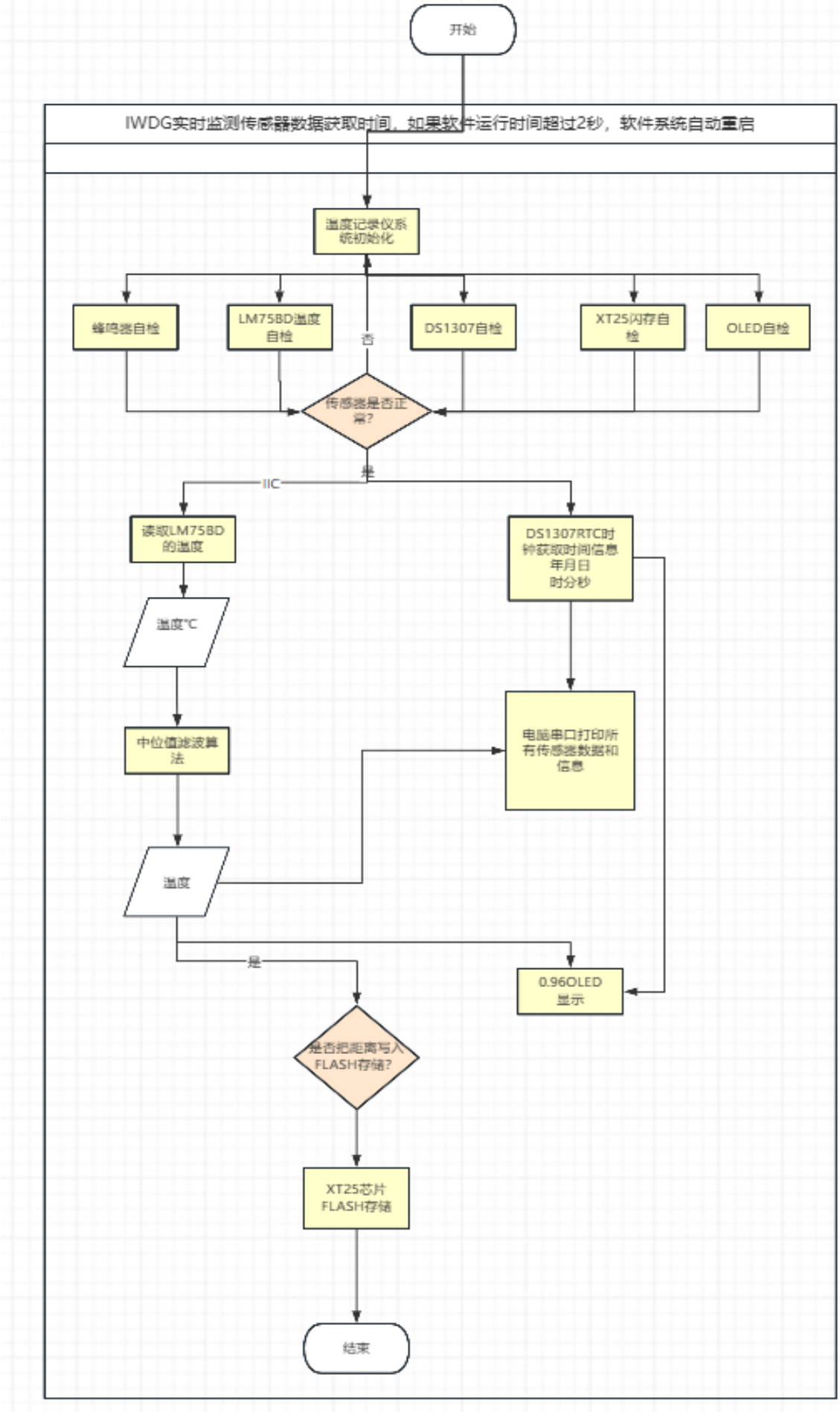
器件	单价	总价
CR1220-2 CR1220-2	1.3778	6.89
无源蜂鸣器 QMB-09B-03	0.6659	3.33
100nF CC0603KRX7R9BB104	0.0141	1.41
22pF CL10C220JB8NNNC	0.0245	2.45
1uF CL10A105KB8NNNC	0.0274	1.37
RUN 19-217/GHC-YR1S2/3T	0.2287	0.00
ISP TS3625A	0.21964	4.39
SS8050 SS8050(RANGE:200-350)	0.0816	4.08
4.7kΩ 0603WAF4701T5E	0.0063	0.03
10kΩ 0603WAF1002T5E	0.0056	0.56

1.5kΩ 0603WAF1501T5E	0.0063	0.63
1kΩ 0603WAF1501T5E	0.0063	0.63
10kΩ 0603WAF1501T5E	0.0063	0.63
RESET TS3625A	0.21964	4.39
CH340E CH340E	3.83	3.83
LPC1114FBD48/302,1 LPC1114FBD48/302,1	27.28	27.28
XT25F02ESOIGU XT25F02ESOIGU	0.77652	3.88
LM75BD LM75BD	2.2147	11.07
DS18B20+ DS18B20+	6.68	6.68
XC6206P332MR XC6206P332MR-G	0.6163	3.08
XL1307 XL1307	1.3164	6.58
AMS1117-3.3_C347222 AMS1117-3.3	0.2394	1.20
32.768kHz Q13FC13500004	1.1957	5.98
		100.37

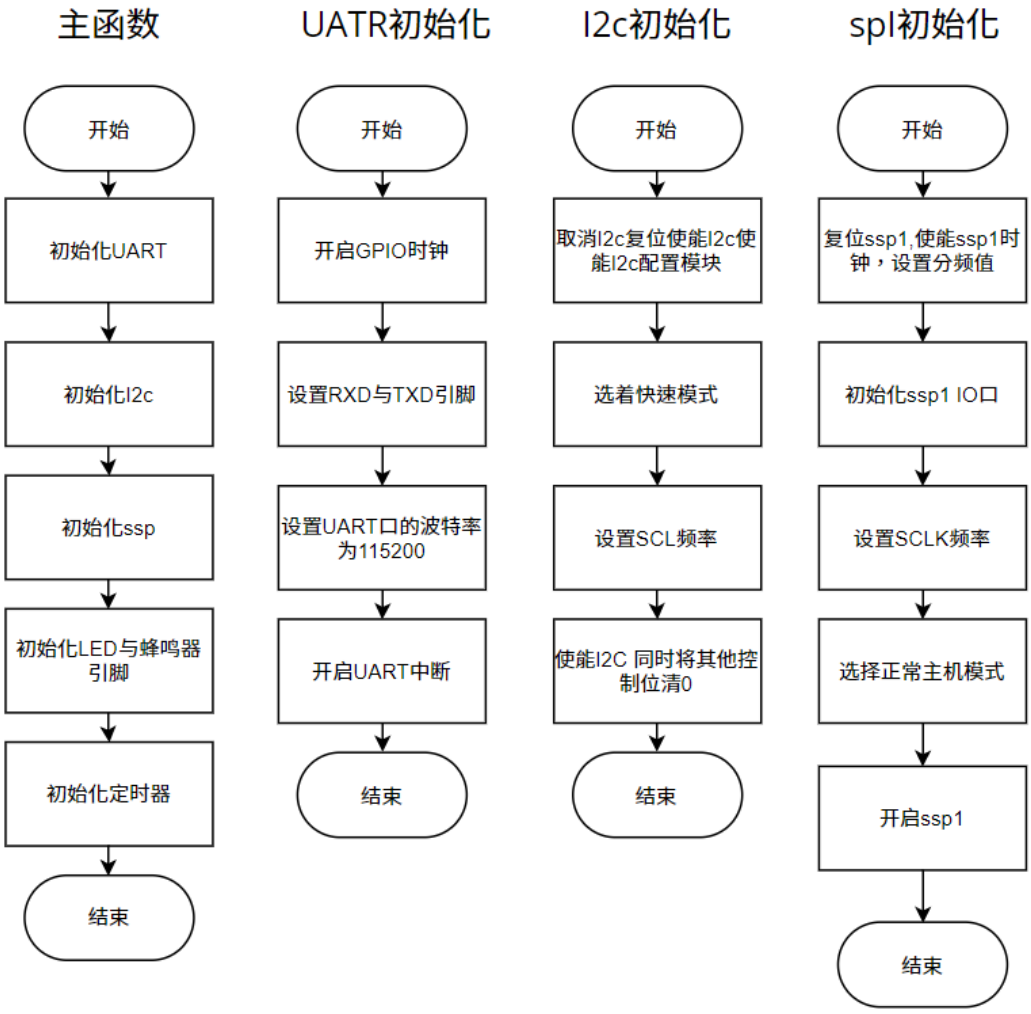
五块 PCB 和五套物料清单如上，平均每套元器件的花费大约 20 元。再额外购买蓝牙模块和两节 18650 锂电池、OLED 显示屏。**平均完整一套温度记录仪花费大约 40 元。**

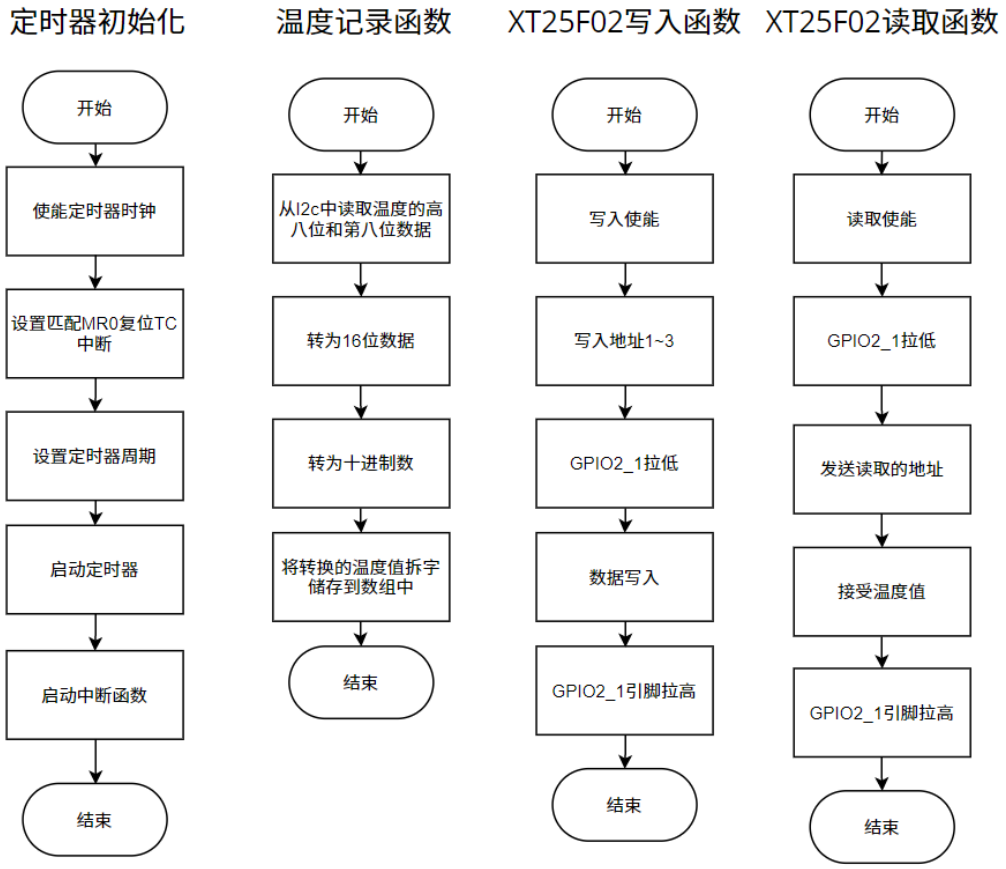
五、项目软件程序设计

5.1 软件总体流程图


开始
IWDG实时监测传感器数据获取时间，如果软件运行时间超过2秒，软件系统自动重启
温度记录仪系统初始化
蜂鸣器自检
LM75BD温度自检
DS1307自检
XT25闪存自检
OLED自检
否
传感器是否正常?
是
IIC
读取LM75BD的温度
温度℃
中位值滤波算法
温度
DS1307RTC时钟获取时间信息
年月日
时分秒
电脑串口打印所有传感器数据和信息
是
0.96OLED显示
是否把距离写入FLASH存储?
XT25芯片FLASH存储
结束

5.2 各个模块单元流程图





5.3 项目软件核心代码介绍

```
/*****************************************************************
 * 课程：基于 NXP LPC1114 的嵌入式系统设计
 * 姓名：刘嘉俊
 * 学号：20223004426
 * 任课教师：王庆吉
 * 嵌入式系统课程设计大作业
*****************************************************************/
/*****************************************************************
 * 编译环境及版本： Keil MDK5 5.39
 * 编译器版本： ARM Compiler version 5.06（960）
 * 硬件版本： NXP LPC1114 DevKit V5.6
 * 调试器： DAPLink
 * 语言： C 语言
 * 文字版本： Chinese GB2312
*****************************************************************/
/*****************************************************************
 * 基于 NXP LPC1114 微控制器的温度监控系统，具备以下主要功能：
 * 温度读取与显示：使用 LM75BD 温度传感器读取环境温度，并通过中位数滤波算法处理数据以获得更稳定的温度值。
```

```
        通过 UART 接口与外部设备进行数据通信，定期发送当前的日期、时间和温度数据。
        同时温度数据可以通过 0.96 寸 OLED 显示屏实时显示，显示内容包括日期、时间、温度等信息。
 * 报警功能：根据实时温度，系统会进行判断并触发相应的报警机制。温度超过设定阈值时，会启动蜂鸣器报警，
        并控制 RGB LED 的颜色变化，以便用户可以通过视觉和听觉感知温度异常。
 * 数据存储：通过 SPI 接口与 XT25 闪存进行数据交互，系统会定期将温度数据存储到闪存中。
        支持对闪存的擦除、写入和读取操作，确保数据的持久存储。
 * 时钟与日期管理：集成了 DS1307 RTC 模块，提供准确的时间和日期功能。
        系统可以读取并显示当前的时间和日期，并在每次操作时同步时间数据。
 * 串口通信与调试：通过 UART 接口与外部设备进行数据通信，定期发送当前的日期、时间和温度数据，便于外部设备获取信息或调试。
**********************************************************************/
/**********************************************************************
 * 编译成功后，烧入代码到口袋开发板 NXP LPC1114 DevKit V5.6
       开发板初始会先滴一声。
 * 声光显示
       当前设置    温度 0-10℃ 开发板绿灯闪烁
                   温度 10-20℃ 开发板蓝灯闪烁
                   温度 20-30℃ 开发板红灯闪烁
                   当温度超过 30℃，开发板所有的灯常量并且蜂鸣器开始报警
 * 通过 IIC 的 SCL 和 SDA 外接一个
       0.96 寸 OLED 显示屏
       OLED 显示内容如下
                     LiuJiajun !
                   DATE:  2025-01-03
                   TIME:   15:13:51
                   Temp:  27.625 ℃
 * UART 串口输出如下
                   Hainan University
                   Teacher: Wang Qingji
                   Liu Jiajun
                   20223004426
                   Date: 2025-01-03
                   Time: 15:13:51
                   Week: 5
                   Temperature: 27.625
**********************************************************************/
/* 引入头文件"LPC11xx.h" */
#include "LPC11xx.h"
#include "stdio.h"  // 使用串口重定向，把 UART_Send（）函数定向给 stdio.h 库中的 printf，方便使用 printf 进行串口调试
```

```
/*****************************************************************
********/
/* I2C 控制寄存器 */
#define I2CONSET_AA         0x00000004  // 是否产生应答信号允许位，即是否设
置为从机模式
#define I2CONSET_SI         0x00000008  // I2C 中断标志位
#define I2CONSET_STO        0x00000010 // 停止标志位
#define I2CONSET_STA        0x00000020  // 开始标志位
#define I2CONSET_I2EN       0x00000040   // I2C 接口允许位
/* I2C “清控制寄存器位”寄存器 */
#define I2CONCLR_AAC        0x00000004    // 清应答信号允许位
#define I2CONCLR_SIC        0x00000008  // 清 I2C 中断标志位
#define I2CONCLR_STAC       0x00000020  // 清开始标志位
#define I2CONCLR_I2ENC      0x00000040  // 清 I2C 接口允许位
/* I2C 通信速率宏定义 */
#define I2SCLH_SCLH         0x00000180  /* I2C SCL Duty Cycle High Reg */
#define I2SCLL_SCLL         0x00000180  /* I2C SCL Duty Cycle Low Reg */
#define I2SCLH_HS_SCLH      0x00000030  /* Fast Plus I2C SCL Duty Cycle High
Reg */
#define I2SCLL_HS_SCLL      0x00000030  /* Fast Plus I2C SCL Duty Cycle Low
Reg */
/* XT25 存储宏定义 */
#define WRITE    0X02
#define WREN    0X06         //写入使能
#define WRDI    0X04
#define RDSR    0X05
#define WRSR    0X01
#define READ    0X03

#define TRUE 1
#define FALSE 0
/*****************************************************************
********/
// 变量声明
/*****************************************************************
********/
float Temp_LM75BD = 0;  // 存储从 LM75BD 温度传感器读取的温度值
uint8_t Temp_Buf[8] = {0};  // 存储温度数据的缓冲区

uint8_t Date[10] = {0};  // 存储当前日期（格式：YYYY-MM-DD）
uint8_t Time_Data[10] = {0};  // 存储当前时间（格式：HH:MM:SS）
uint8_t Week[1] = {0};  // 存储当前星期（1~7，1 表示星期一）
uint8_t Time[7] = {0x50, 0x13, 0x15, 0x05, 0x03, 0x01, 0x25};  // 时间（秒、
分、时、星期、日、月、年）
```

```
uint8_t data[7] = {0};  // 用于存储读取的时间和日期数据

uint8_t Write_Addr[16] = {0};  // 用于 SPI 写操作的地址
uint8_t Read_Addr[16] = {0};   // 用于 SPI 读操作的地址

uint8_t Music_Flag = 0;  // 音乐播放标志，控制音乐播放状态

/*****************************************************************
********/
// OLED 显示相关函数声明
/*****************************************************************
********/
void OLED_Init(void);  // 初始化 OLED 显示屏
void OLED_Clear(void);  // 清空 OLED 显示内容
void OLED_ShowChar(uint8_t Line, uint8_t Column, char Char);  // 在指定行
列显示一个字符
void OLED_ShowString(uint8_t Line, uint8_t Column, char *String);  // 在
指定行列显示字符串
void OLED_ShowNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length);  // 显示数字
void OLED_ShowSignedNum(uint8_t Line, uint8_t Column, int32_t Number,
uint8_t Length);  // 显示带符号的数字
void OLED_ShowHexNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length);  // 显示 16 进制数
void OLED_ShowBinNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length);  // 显示二进制数

/*****************************************************************
********/
// 16 位定时器相关函数声明
/*****************************************************************
********/
void T16B0_init(void);  // 初始化 16 位定时器 0
void T16B0_delay_ms(uint16_t ms);  // 毫秒级延时函数
void T16B0_delay_us(uint16_t us);  // 微秒级延时函数

/*****************************************************************
********/
// 16 位定时器 1 PWM 相关函数声明
/*****************************************************************
********/
void T16B1_PWM_init(void);  // 初始化 16 位定时器 1 为 PWM 模式
void PassiveBuzzer_Init(void);  // 初始化蜂鸣器
void PassiveBuzzer_OFF(void);  // 关闭蜂鸣器
```

```
void PassiveBuzzer_ON(void);  // 启动蜂鸣器

/*************************************************************************
********/
// UART 串口通信相关函数声明
/*************************************************************************
********/
void UART_Init(uint32_t baudrate);  // 初始化 UART，设置波特率
uint8_t UART_Recive(void);  // 接收一个字节数据
void UART_Send_Byte(uint8_t byte);  // 发送一个字节数据
void UART_Send(uint8_t *BufferPtr, uint32_t Length);  // 发送指定长度的数
据

/*************************************************************************
********/
// I2C 总线通信相关函数声明
/*************************************************************************
********/
void I2C_Init(uint8_t Mode);  // 初始化 I2C 通信，选择模式（标准/快速）
void I2C_Start(void);  // 启动 I2C 通信
void I2C_Stop(void);  // 停止 I2C 通信
void I2C_Send_Byte(uint8_t dat);  // 发送一个字节数据
uint8_t I2C_Recieve_Byte(void);  // 接收一个字节数据

/*************************************************************************
********/
// SPI 通信相关函数声明
/*************************************************************************
********/
void SSP1_IOConfig(void);  // 配置 SPI1 接口的 IO
void SSPI1_Init(void);  // 初始化 SPI1 接口
void SSP1_LOW(void);  // 设置 SPI1 片选低电平
void SSP1_HIGH(void);  // 设置 SPI1 片选高电平
uint8_t SPI1_Comunicate(uint8_t TxData);  // 发送一个字节数据，并接收响应
void SSP1_Send(uint8_t *data, uint8_t Length);  // 发送指定长度的数据
void SSP1_Receive(uint8_t *data, int Length);  // 接收指定长度的数据

/*************************************************************************
********/
// XT25 闪存相关函数声明
/*************************************************************************
********/
void XT25_WriteEnable(void);  // 启用 XT25 写入
uint8_t XT25_ReadSR(void);  // 读取 XT25 状态寄存器
```

```
void XT25_Write_Wait(void);  // 等待 XT25 写入完成
void XT25_Read_Wait(void);  // 等待 XT25 读取完成
void XT25_WriteSR(uint8_t sr);  // 写入 XT25 状态寄存器
void XT25_RUID(void);  // 读取 XT25 唯一 ID
void XT25_EraseAll(void);  // 擦除 XT25 闪存
void XT25_EraseSector(void);  // 擦除 XT25 闪存的扇区
void SPI1_Write_FLASH(uint8_t *data, uint8_t Length);  // 向 XT25 写入数据
void SPI1_Read_FLASH(uint8_t *data, uint8_t Length);  // 从 XT25 读取数据

/*************************************************************************
********/
// DS1307 RTC 时钟相关函数声明
/*************************************************************************
********/
void DS1307Init(void);  // 初始化 DS1307 时钟（第一次烧录时使用）
void DS1307_Read(void);  // 读取 DS1307 时间和日期
void DS1307_Write(uint8_t *data);  // 向 DS1307 写入时间和日期
void Ds1307_WriteByte(uint8_t WriteAddr, uint8_t WriteData);  // 向 DS1307
写入一个字节数据
uint8_t Ds1307_ReadByte(void);  // 从 DS1307 读取一个字节数据

/*************************************************************************
********/
// RGB LED 控制函数声明
/*************************************************************************
********/
void RGB_LED_Init(void);  // 初始化 RGB LED
void LED_Toggle(void);  // 切换 LED 状态
void RGB_Blue_Toggle(void);  // 切换蓝色 LED 状态
void RGB_Green_Toggle(void);  // 切换绿色 LED 状态
void RGB_Red_Toggle(void);  // 切换红色 LED 状态

/*************************************************************************
********/
// 温度读取和处理函数声明
/*************************************************************************
********/
float Read_Temp_LM75BD(void);  // 读取 LM75BD 温度传感器的数据
void Menu(void);  // 显示菜单（日期、时间、温度）
void Temperature_Judgment(float T);  // 根据温度判断是否需要报警
void Transfor(uint8_t *Temp_Str, float T);  // 将温度转换为字符串
float MedianFilter(void);  // 中位数滤波算法，去除极端值
void UART_Send_Date(void);  // 通过 UART 发送日期、时间和温度数据
```

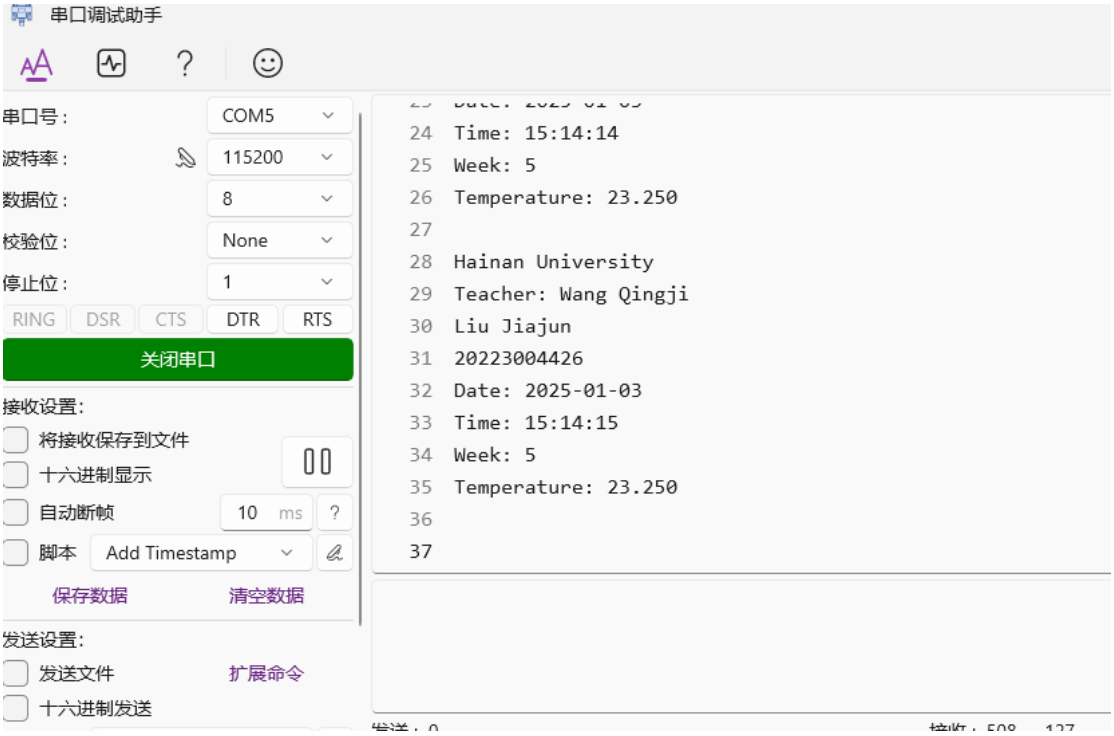
```
/*************************************************************************
********/
// 定时器初始化及中断处理函数声明
/*************************************************************************
********/
void TMR32B0_Init(void);  // 初始化 32 位定时器 0
void TIMER32_0_IRQHandler(void);  // 32 位定时器 0 的中断处理函数

/*************************************************************************
********/
 /**
 * 函数功能：重定向 printf 到 UART
 * 描述：此函数实现了 printf 的底层重定向，使得所有的 printf 输出都通过 UART 进行
发送。
 * 参数：
 *      ch - 要输出的字符
 *      f  - 文件指针（对于此函数不使用，仅作为 printf 的标准接口）
 * 返回值：
 *      返回输出的字符（保持原始格式）
 */
int fputc(int ch, FILE *f)
{
    UART_Send_Byte(ch);  // 将字符通过 UART 发送
    return ch;           // 返回输出字符
}

/*********************************************/
/*              主函数                       */
/*********************************************/
int main()
{
    SystemInit();    // 系统时钟初始化
    I2C_Init(1);     // 初始化 I2C 通信（快速模式）
    SSPI1_Init();    // 初始化 SPI 接口
    UART_Init(115200);  // 初始化 UART，波特率为 115200
    T16B1_PWM_init();  // 初始化 16 位定时器 1 为 PWM 模式，控制蜂鸣器
    T16B0_init();      // 初始化 16 位定时器 0
    TMR32B0_Init();    // 初始化 32 位定时器 0（用于中断处理），1 秒钟在串口打
印一次数据并且在 OLED 屏幕上显示
    RGB_LED_Init();    // 初始化 RGB LED
    DS1307Init();    // 仅第一次烧录时用来校准时间
    DS1307_Read();     // 读取 DS1307 时间
    SPI1_Read_FLASH(Temp_Buf, 8); // 从 SPI 闪存读取温度数据
    OLED_Init();       // 初始化 OLED 显示屏
```

```
    OLED_Clear();       // 清除 OLED 屏幕内容
    PassiveBuzzer_Init();  // 初始化蜂鸣器
    T16B0_delay_ms(200);  // 延时 200ms，等待硬件初始化完成
    Read_Temp_LM75BD();   // 第一次读取温度（过滤掉初始不准确的值）
    PassiveBuzzer_OFF();  // 关闭蜂鸣器
    while(1)
    {

    }
}

/*********************************************************************
 * 函数名: TIMER32_0_IRQHandler
 * 描述: 定时器 0 中断处理函数，触发时进行温度数据读取与处理，1 秒钟在串口打印一次数据并且在 OLED 屏幕上显示
 * 在定时器 0 的匹配事件发生时，启动 ADC 转换，读取 ADC7 的值，处理温度数据并更新显示。
 *********************************************************************/
void TIMER32_0_IRQHandler(void) {
    // 检查定时器 0 的匹配中断标志（MR0）
    if ((LPC_TMR32B0->IR |= 0x01) == 1) {

        // 从 SPI 闪存读取温度数据
        SPI1_Read_FLASH(Temp_Buf, 8);

        // 打印调试信息，输出到串口
        printf("Hainan University\n");
        printf("Teacher: Wang Qingji\n");
        printf("Liu Jiajun\n");
        printf("20223004426\n");

        // 读取 DS1307 时间
        DS1307_Read();

        // 进行温度中位数滤波处理
        Temp_LM75BD = MedianFilter();

        // 将温度数据转换为字符串
        Transfor(Temp_Buf, Temp_LM75BD);

        // 通过 UART 发送日期、时间和温度数据
        UART_Send_Date();

        // 打印当前温度
```

```
        printf("Temperature: %s\n", Temp_Buf);
        printf(" \n");

        // 显示日期、时间和温度在 OLED 屏幕
        Menu();

        // 根据当前温度执行判断操作（例如触发报警或控制 RGB 灯）
        Temperature_Judgment(Temp_LM75BD);

        // 执行 XT25 芯片的扇区擦除操作
        XT25_EraseSector();

        // 将温度数据写入到 XT25 闪存
        SPI1_Write_FLASH(Temp_Buf, 8);

        // 清除定时器 0 的匹配中断标志
        LPC_TMR32B0->IR = 0x01;
    }
```

六、代码调试与系统测试

串口输出结果：



七、完成功能情况与问题分析

7.1 功能介绍

本项目设计的温度记录仪基于 LPC1114 微控制器，结合多种外设模块，具备以下核心功能：

（1）环境温度采集通过 LM75BD 温度传感器实时采集环境温度，支持 -55℃至 +125℃ 的测量范围，精度达 ±1.5℃。温度数据每秒更新一次，保证实时性。

（2）数据存储使用 XT25 SPI 闪存模块存储采集到的温度数据，并将其与 RTC 提供的时间戳关联，形成完整的温度记录。存储容量不低于 2MB，可支持长时间数据保存。

（3）实时显示配备 0.96 英寸 OLED 显示屏，通过 I2C 接口显示当前的时间、日期和温度数据，信息清晰直观，方便用户查看。

（4）报警提示，系统支持温度超限报警功能，当温度超过设定阈值（如 30℃）时，蜂鸣器发出报警声，RGB 指示灯根据温度范围显示不同的颜色，提供视觉和听觉双重提示。

（5）时间管理，集成 DS1307 实时时钟模块，提供精确的时间和日期功能。支持用户通过串口命令校准时间，断电时钟仍可正常运行。

（6）串口通信，通过 CH340 USB 转串口模块实现与外部设备的数据交互，支持温度记录的导出和时间校准等操作，方便与上位机通信。

（7）双供电模式，系统支持 USB 和 18650 锂电池两种供电模式，在断电情况下可切换至电池供电，并保持 RTC 模块的正常运行。

（8）高稳定性与低功耗，采用稳压电路和滤波设计，确保系统运行稳定，同时支持多种低功耗模式，延长电池使用时间。

以上功能相辅相成，使温度记录仪能够满足环境温度实时监测、数据存储、警报提示等多种需求，适用于实际生活与工业环境中的温度监控应用场景。

7.2 困难与问题分析

在本项目的设计与实现过程中，遇到了一些困难和问题，主要包括以下几个方面：

在硬件电路设计阶段，由于多个模块通过 I2C 和 SPI 接口与微控制器连接，容易出现信号干扰和引脚冲突的问题。例如，I2C 总线上同时连接了温度传感器、OLED 显示屏和 RTC 模块，需要合理分配地址和确保总线通信的稳定性。

PCB 布线过程中，由于板子尺寸受限（8cm × 3cm），布局需要非常紧凑，同时还需要考虑电源滤波和信号完整性问题，增加了设计难度。

在软件开发中，串口通信和多任务协调成为一大难点。例如，如何在实现温

度采集、数据存储和报警功能的同时，确保串口通信的数据传输不受影响，需要在中断和主循环之间进行合理分配。

数据存储模块的操作复杂，例如 XT25 SPI 闪存的写入和读取需要严格按照操作时序进行，稍有不慎可能导致数据丢失或操作失败。

硬件调试过程中，遇到了电源稳定性问题，特别是在 USB 和电池供电切换时，系统可能会出现瞬时断电或电压不稳的情况，导致微控制器复位。

温度传感器的数据在初次采集时波动较大，需要通过滤波算法（如中位数滤波）对数据进行稳定化处理，同时优化采样频率以减少噪声干扰。

系统虽然具备基本的温度监测和记录功能，但由于闪存容量有限，长时间存储的大量数据会占用较大存储空间，未来可优化数据压缩存储方法。

针对以上问题，通过以下方法进行了优化：

（1）在硬件设计中，合理规划总线接口的地址分配，增加了信号滤波电路，确保通信稳定。

（2）软件开发中，优化了中断优先级分配和任务调度机制，保证串口通信与其他功能的协调。

（3）对电源切换机制进行了多次调试，并通过添加电容和稳压模块增强了电源的稳定性。

（4）采用中位数滤波算法处理温度数据，成功减少了传感器噪声。

尽管在开发过程中遇到了一些困难，但通过不断的调试与优化，最终实现了温度记录仪的设计目标。未来的改进方向包括增加数据压缩算法、提升电源管理能力，以及优化硬件设计以支持更多功能模块。

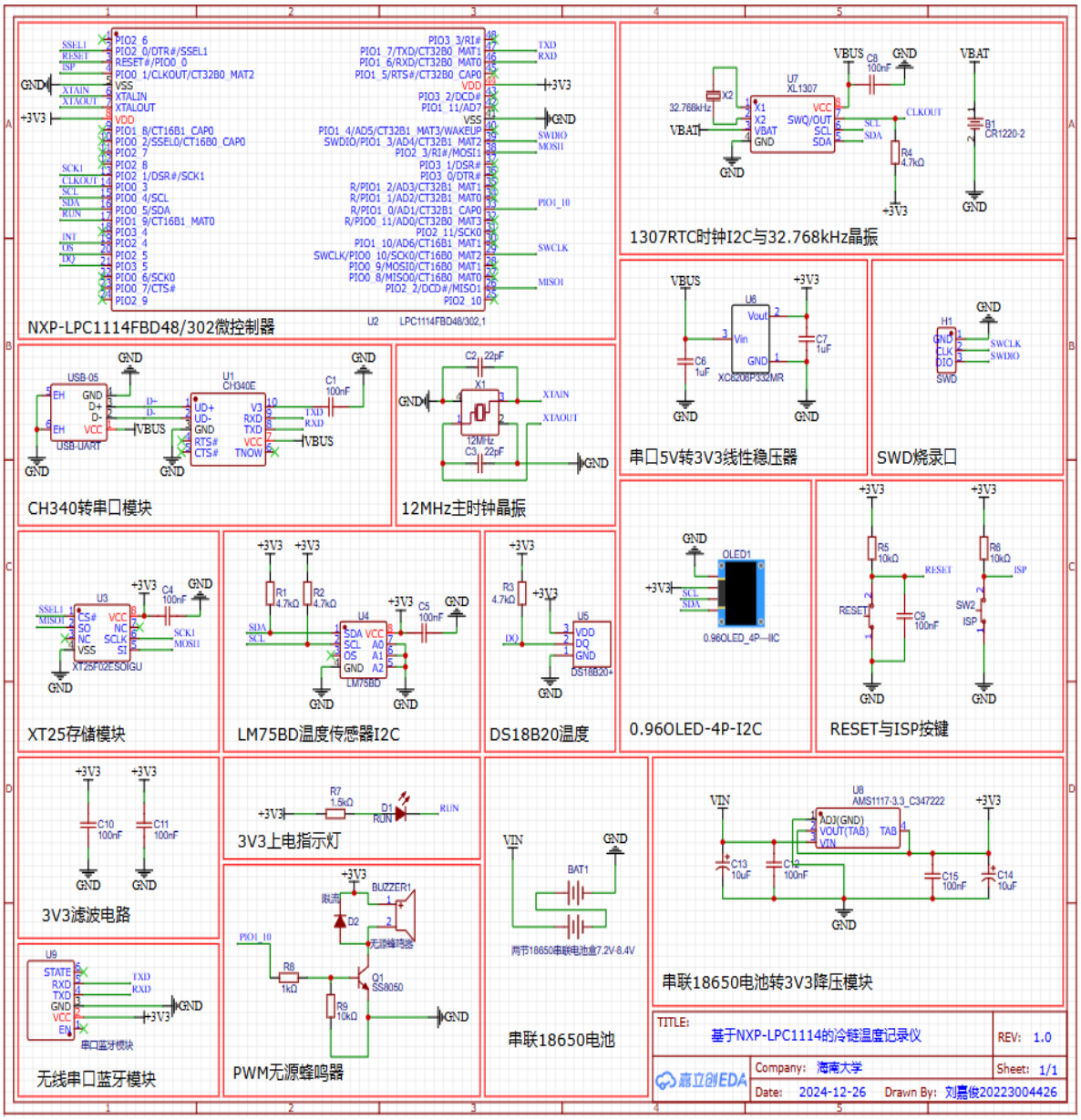
八、自我评价

通过本次基于 LPC1114 的温度记录仪设计，我全面提升了嵌入式系统开发的理论知识和实践能力。在项目过程中，我学会了从硬件设计到软件编写的完整开发流程，特别是在温度传感器、存储模块以及通信模块的功能实现中，积累了丰富的经验。同时，我克服了多次调试中遇到的硬件问题和软件逻辑错误，增强了独立解决问题的能力。

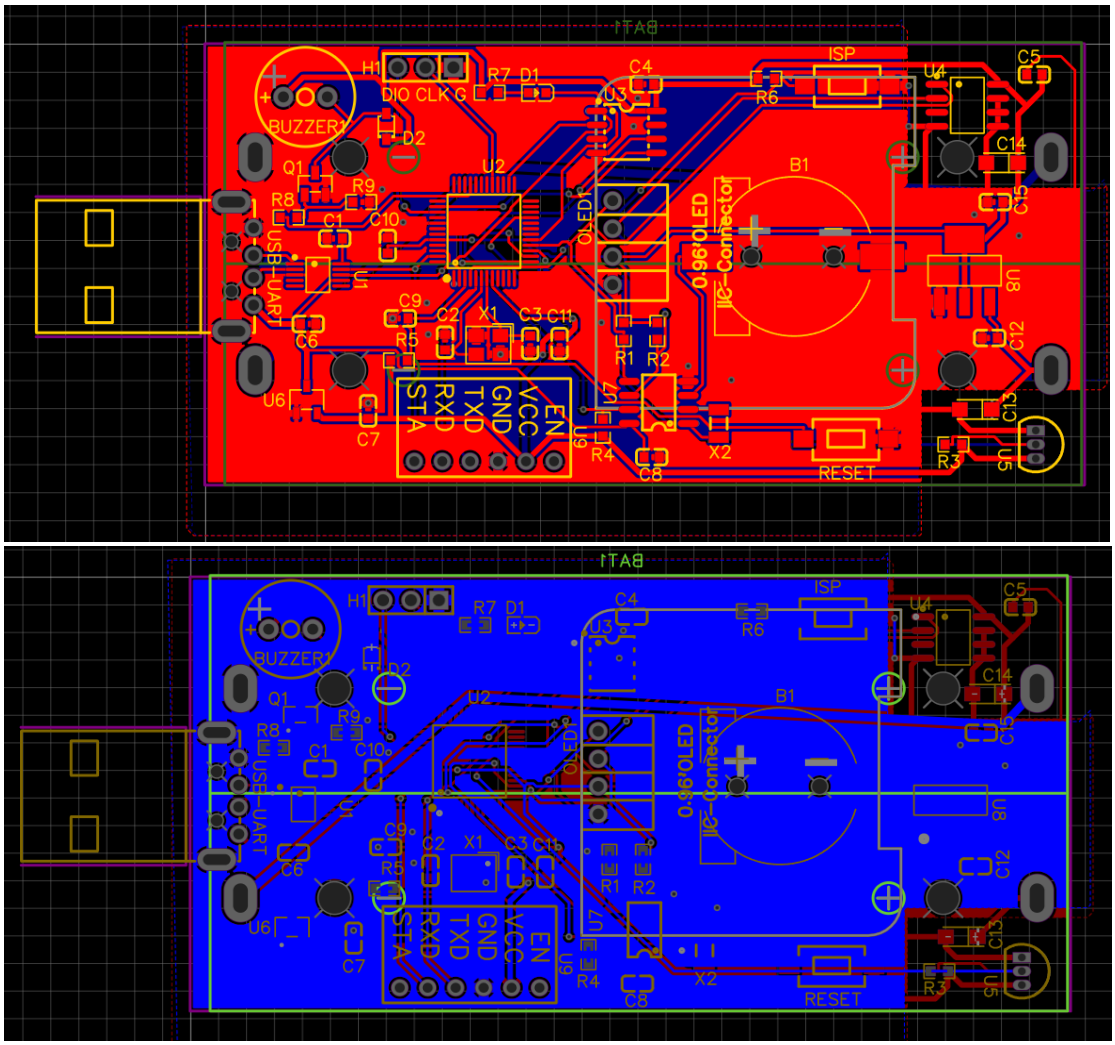
虽然项目最终达到了设计目标，但也发现了一些不足之处，例如在电路设计上可以进一步优化布线和电源管理，在软件编写中可以更好地提高代码的效率和可读性。通过这次项目，我意识到细节的重要性，也为以后更复杂的嵌入式系统设计积累了宝贵的经验，总体来说是一次非常有意义的学习和实践过程。

附件：

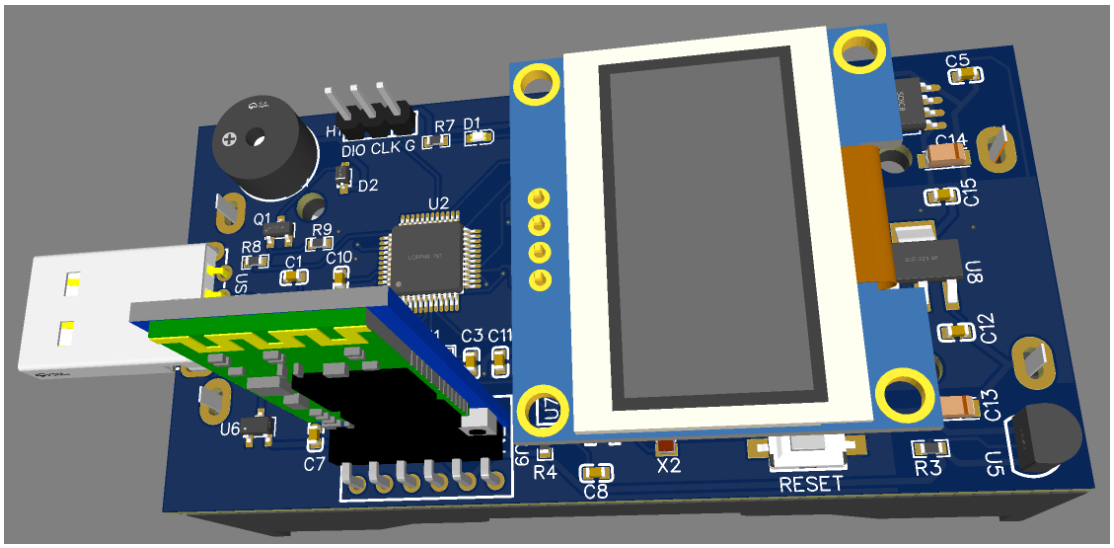
项目硬件原理图：

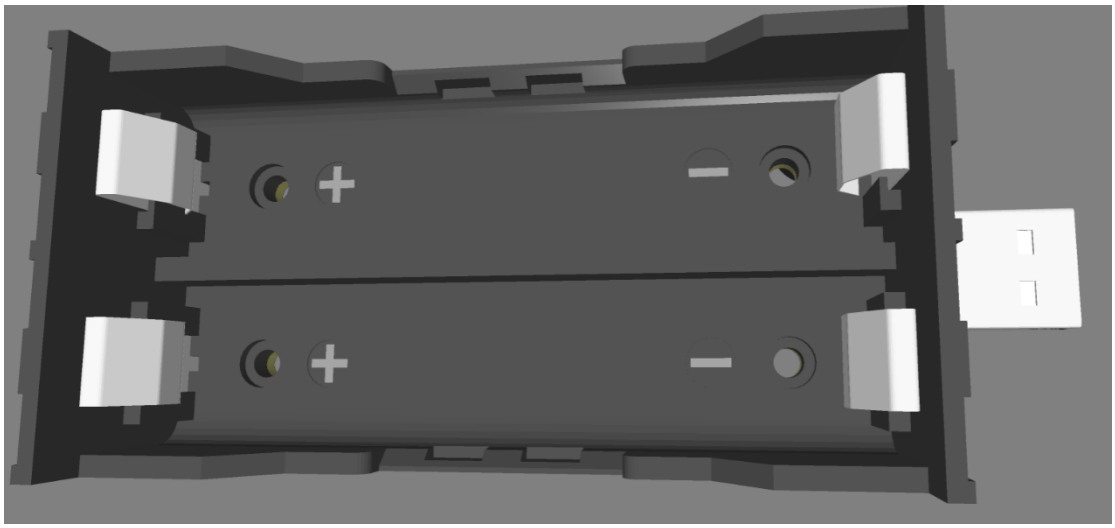

NXP-LPC1114FBD48/302微控制器
1307RTC时钟I2C与32.768kHz晶振
串口5V转3V3线性稳压器
SWD烧录口
CH340转串口模块
12MHz主时钟晶振
XT25存储模块
LM75BD温度传感器I2C
DS18B20温度
0.96OLED-4P-I2C
RESET与ISP按键
3V3滤波电路
3V3上电指示灯
串联18650电池转3V3降压模块
无线串口蓝牙模块
PWM无源蜂鸣器
串联18650电池
TITLE: 基于NXP-LPC1114的冷链温度记录仪
REV: 1.0
Company: 海南大学
Sheet: 1/1
Date: 2024-12-26
Drawn By: 刘嘉俊20223004426

项目 PCB 图：

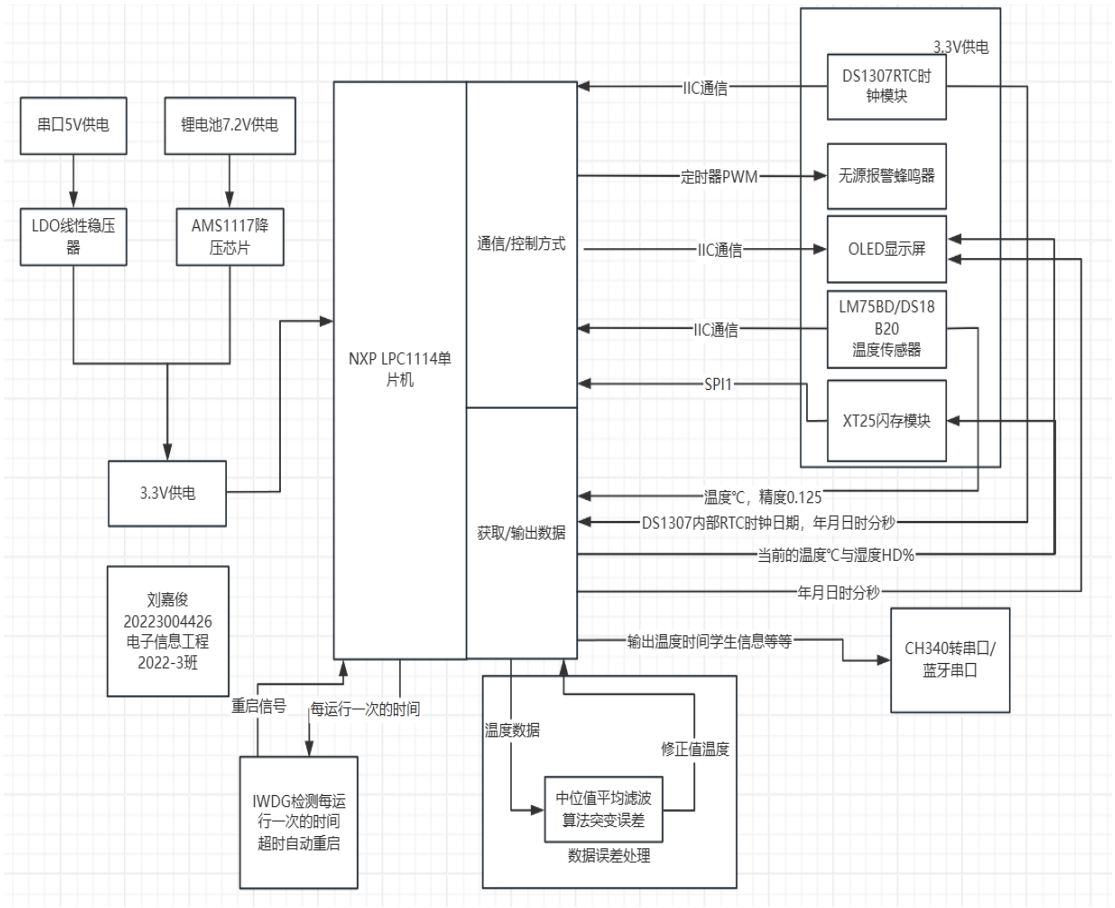


项目 3D 图：

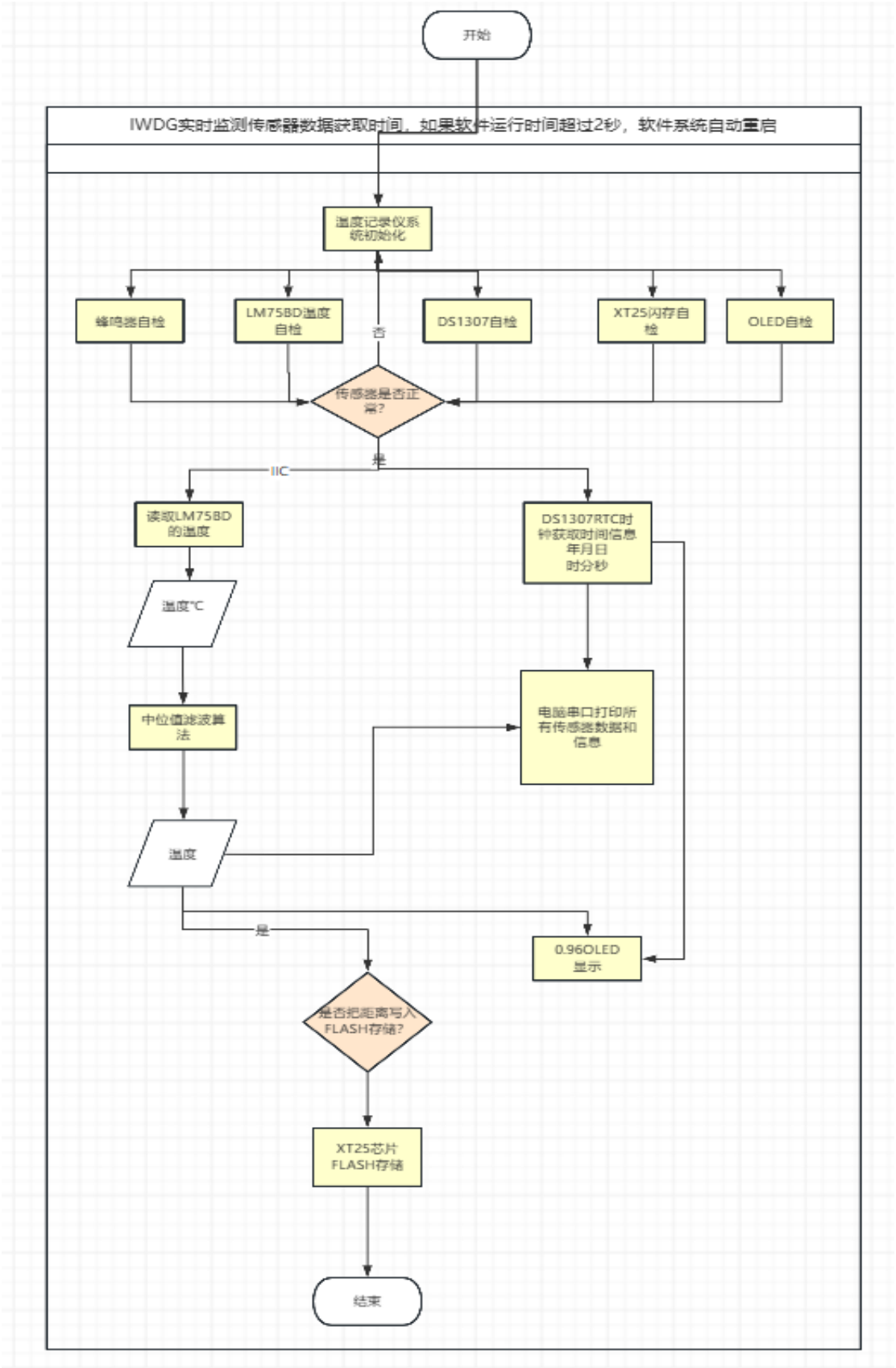




系统流程图：


串口5V供电
锂电池7.2V供电
LDO线性稳压器
AMS1117降压芯片
3.3V供电
NXP LPC1114单片机
通信/控制方式
获取/输出数据
3.3V供电
IIC通信
DS1307RTC时钟模块
定时器PWM
无源报警蜂鸣器
IIC通信
OLED显示屏
IIC通信
LM75BD/DS18B20温度传感器
SPI1
XT25闪存模块
温度℃，精度0.125
DS1307内部RTC时钟日期，年月日时分秒
当前的温度℃与湿度HD%
年月日时分秒
输出温度时间学生信息等等
CH340转串口/蓝牙串口
刘嘉俊
20223004426
电子信息工程
2022-3班
重启信号
每运行一次的时间
IWDG检测每运行一次的时间超时自动重启
温度数据
修正值温度
中位值平均滤波算法突变误差
数据误差处理

软件流程图：


开始
IWDG实时监测传感器数据获取时间，如果软件运行时间超过2秒，软件系统自动重启
温度记录仪系统初始化
蜂鸣器自检
LM75BD温度自检
DS1307自检
XT25闪存自检
OLED自检
否
传感器是否正常?
是
IIC
读取LM75BD的温度
温度℃
中位值滤波算法
温度
DS1307RTC时钟获取时间信息
年月日
时分秒
电脑串口打印所有传感器数据和信息
是
0.96OLED显示
是否把距离写入FLASH存储?
XT25芯片FLASH存储
结束

项目完整软件代码：

```
/*****************************************************************
 * 课程：基于 NXP LPC1114 的嵌入式系统设计
 * 姓名：刘嘉俊
 * 学号：20223004426
 * 任课教师：王庆吉
 * 嵌入式系统课程设计大作业
*****************************************************************/
/*****************************************************************
 * 编译环境及版本： Keil MDK5 5.39
 * 编译器版本： ARM Compiler version 5.06（960）
 * 硬件版本： NXP LPC1114 DevKit V5.6
 * 调试器： DAPLink
 * 语言： C 语言
 * 文字版本： Chinese GB2312
*****************************************************************/
/*****************************************************************
 * 基于 NXP LPC1114 微控制器的温度监控系统，具备以下主要功能：
 * 温度读取与显示：使用 LM75BD 温度传感器读取环境温度，并通过中位数滤波算法处理数据以获得更稳定的温度值。
        通过 UART 接口与外部设备进行数据通信，定期发送当前的日期、时间和温度数据。
        同时温度数据可以通过 0.96 寸 OLED 显示屏实时显示，显示内容包括日期、时间、温度等信息。
 * 报警功能：根据实时温度，系统会进行判断并触发相应的报警机制。温度超过设定阈值时，会启动蜂鸣器报警，
        并控制 RGB LED 的颜色变化，以便用户可以通过视觉和听觉感知温度异常。
 * 数据存储：通过 SPI 接口与 XT25 闪存进行数据交互，系统会定期将温度数据存储到闪存中。
        支持对闪存的擦除、写入和读取操作，确保数据的持久存储。
 * 时钟与日期管理：集成了 DS1307 RTC 模块，提供准确的时间和日期功能。
        系统可以读取并显示当前的时间和日期，并在每次操作时同步时间数据。
 * 串口通信与调试：通过 UART 接口与外部设备进行数据通信，定期发送当前的日期、时间和温度数据，便于外部设备获取信息或调试。
*****************************************************************/
/*****************************************************************
 * 编译成功后，烧入代码到口袋开发板 NXP LPC1114 DevKit V5.6
      开发板初始会先滴一声。
 * 声光显示
      当前设置    温度 0-10℃ 开发板绿灯闪烁
                  温度 10-20℃ 开发板蓝灯闪烁
                  温度 20-30℃ 开发板红灯闪烁
                  当温度超过 30℃，开发板所有的灯常量并且蜂鸣器开始报警
```

```
 * 通过 IIC 的 SCL 和 SDA 外接一个
       0.96 寸 OLED 显示屏
       OLED 显示内容如下
                      LiuJiajun !
                    DATE:  2025-01-03
                    TIME:   15:13:51
                    Temp:  27.625 ℃
 * UART 串口输出如下
                    Hainan University
                    Teacher: Wang Qingji
                    Liu Jiajun
                    20223004426
                    Date: 2025-01-03
                    Time: 15:13:51
                    Week: 5
                    Temperature: 27.625
**************************************************************************/
/* 引入头文件"LPC11xx.h" */
#include "LPC11xx.h"
#include "stdio.h"  // 使用串口重定向，把 UART_Send（）函数定向给 stdio.h 库
中的 printf，方便使用 printf 进行串口调试
/*************************************************************************
********/
/* I2C 控制寄存器 */
#define I2CONSET_AA         0x00000004  // 是否产生应答信号允许位，即是否设
置为从机模式
#define I2CONSET_SI         0x00000008  // I2C 中断标志位
#define I2CONSET_STO        0x00000010 // 停止标志位
#define I2CONSET_STA        0x00000020  // 开始标志位
#define I2CONSET_I2EN       0x00000040   // I2C 接口允许位
/* I2C “清控制寄存器位”寄存器 */
#define I2CONCLR_AAC        0x00000004    // 清应答信号允许位
#define I2CONCLR_SIC        0x00000008  // 清 I2C 中断标志位
#define I2CONCLR_STAC       0x00000020  // 清开始标志位
#define I2CONCLR_I2ENC      0x00000040  // 清 I2C 接口允许位
/* I2C 通信速率宏定义 */
#define I2SCLH_SCLH         0x00000180  /* I2C SCL Duty Cycle High Reg */
#define I2SCLL_SCLL         0x00000180  /* I2C SCL Duty Cycle Low Reg */
#define I2SCLH_HS_SCLH      0x00000030  /* Fast Plus I2C SCL Duty Cycle High
Reg */
#define I2SCLL_HS_SCLL      0x00000030  /* Fast Plus I2C SCL Duty Cycle Low
Reg */
/* XT25 存储宏定义 */
#define WRITE    0X02
```

```
#define WREN    0X06        //写入使能
#define WRDI    0X04
#define RDSR    0X05
#define WRSR    0X01
#define READ    0X03

#define TRUE 1
#define FALSE 0
/*****************************************************************
********/
// 变量声明
/*****************************************************************
********/
float Temp_LM75BD = 0;  // 存储从 LM75BD 温度传感器读取的温度值
uint8_t Temp_Buf[8] = {0};  // 存储温度数据的缓冲区

uint8_t Date[10] = {0};  // 存储当前日期（格式：YYYY-MM-DD）
uint8_t Time_Data[10] = {0};  // 存储当前时间（格式：HH:MM:SS）
uint8_t Week[1] = {0};  // 存储当前星期（1~7，1 表示星期一）
uint8_t Time[7] = {0x50, 0x13, 0x15, 0x05, 0x03, 0x01, 0x25};  // 时间（秒、
分、时、星期、日、月、年）
uint8_t data[7] = {0};  // 用于存储读取的时间和日期数据

uint8_t Write_Addr[16] = {0};  // 用于 SPI 写操作的地址
uint8_t Read_Addr[16] = {0};   // 用于 SPI 读操作的地址

uint8_t Music_Flag = 0;  // 音乐播放标志，控制音乐播放状态

/*****************************************************************
********/
// OLED 显示相关函数声明
/*****************************************************************
********/
void OLED_Init(void);  // 初始化 OLED 显示屏
void OLED_Clear(void);  // 清空 OLED 显示内容
void OLED_ShowChar(uint8_t Line, uint8_t Column, char Char);  // 在指定行
列显示一个字符
void OLED_ShowString(uint8_t Line, uint8_t Column, char *String);  // 在
指定行列显示字符串
void OLED_ShowNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length);  // 显示数字
void OLED_ShowSignedNum(uint8_t Line, uint8_t Column, int32_t Number,
uint8_t Length);  // 显示带符号的数字
```

```
void OLED_ShowHexNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length);  // 显示 16 进制数
void OLED_ShowBinNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length);  // 显示二进制数

/*************************************************************************
********/
// 16 位定时器相关函数声明
/*************************************************************************
********/
void T16B0_init(void);  // 初始化 16 位定时器 0
void T16B0_delay_ms(uint16_t ms);  // 毫秒级延时函数
void T16B0_delay_us(uint16_t us);  // 微秒级延时函数

/*************************************************************************
********/
// 16 位定时器 1 PWM 相关函数声明
/*************************************************************************
********/
void T16B1_PWM_init(void);  // 初始化 16 位定时器 1 为 PWM 模式
void PassiveBuzzer_Init(void);  // 初始化蜂鸣器
void PassiveBuzzer_OFF(void);  // 关闭蜂鸣器
void PassiveBuzzer_ON(void);  // 启动蜂鸣器

/*************************************************************************
********/
// UART 串口通信相关函数声明
/*************************************************************************
********/
void UART_Init(uint32_t baudrate);  // 初始化 UART，设置波特率
uint8_t UART_Recive(void);  // 接收一个字节数据
void UART_Send_Byte(uint8_t byte);  // 发送一个字节数据
void UART_Send(uint8_t *BufferPtr, uint32_t Length);  // 发送指定长度的数
据

/*************************************************************************
********/
// I2C 总线通信相关函数声明
/*************************************************************************
********/
void I2C_Init(uint8_t Mode);  // 初始化 I2C 通信，选择模式（标准/快速）
void I2C_Start(void);  // 启动 I2C 通信
void I2C_Stop(void);  // 停止 I2C 通信
void I2C_Send_Byte(uint8_t dat);  // 发送一个字节数据
```

```
uint8_t I2C_Recieve_Byte(void);  // 接收一个字节数据

/*********************************************************************
********/
// SPI 通信相关函数声明
/*********************************************************************
********/
void SSP1_IOConfig(void);  // 配置 SPI1 接口的 IO
void SSPI1_Init(void);  // 初始化 SPI1 接口
void SSP1_LOW(void);  // 设置 SPI1 片选低电平
void SSP1_HIGH(void);  // 设置 SPI1 片选高电平
uint8_t SPI1_Comunicate(uint8_t TxData);  // 发送一个字节数据，并接收响应
void SSP1_Send(uint8_t *data, uint8_t Length);  // 发送指定长度的数据
void SSP1_Receive(uint8_t *data, int Length);  // 接收指定长度的数据

/*********************************************************************
********/
// XT25 闪存相关函数声明
/*********************************************************************
********/
void XT25_WriteEnable(void);  // 启用 XT25 写入
uint8_t XT25_ReadSR(void);  // 读取 XT25 状态寄存器
void XT25_Write_Wait(void);  // 等待 XT25 写入完成
void XT25_Read_Wait(void);  // 等待 XT25 读取完成
void XT25_WriteSR(uint8_t sr);  // 写入 XT25 状态寄存器
void XT25_RUID(void);  // 读取 XT25 唯一 ID
void XT25_EraseAll(void);  // 擦除 XT25 闪存
void XT25_EraseSector(void);  // 擦除 XT25 闪存的扇区
void SPI1_Write_FLASH(uint8_t *data, uint8_t Length);  // 向 XT25 写入数据
void SPI1_Read_FLASH(uint8_t *data, uint8_t Length);  // 从 XT25 读取数据

/*********************************************************************
********/
// DS1307 RTC 时钟相关函数声明
/*********************************************************************
********/
void DS1307Init(void);  // 初始化 DS1307 时钟（第一次烧录时使用）
void DS1307_Read(void);  // 读取 DS1307 时间和日期
void DS1307_Write(uint8_t *data);  // 向 DS1307 写入时间和日期
void Ds1307_WriteByte(uint8_t WriteAddr, uint8_t WriteData);  // 向 DS1307
写入一个字节数据
uint8_t Ds1307_ReadByte(void);  // 从 DS1307 读取一个字节数据
```

```
/*************************************************************************
********/
// RGB LED 控制函数声明
/*************************************************************************
********/
void RGB_LED_Init(void);  // 初始化 RGB LED
void LED_Toggle(void);  // 切换 LED 状态
void RGB_Blue_Toggle(void);  // 切换蓝色 LED 状态
void RGB_Green_Toggle(void);  // 切换绿色 LED 状态
void RGB_Red_Toggle(void);  // 切换红色 LED 状态

/*************************************************************************
********/
// 温度读取和处理函数声明
/*************************************************************************
********/
float Read_Temp_LM75BD(void);  // 读取 LM75BD 温度传感器的数据
void Menu(void);  // 显示菜单（日期、时间、温度）
void Temperature_Judgment(float T);  // 根据温度判断是否需要报警
void Transfor(uint8_t *Temp_Str, float T);  // 将温度转换为字符串
float MedianFilter(void);  // 中位数滤波算法，去除极端值
void UART_Send_Date(void);  // 通过 UART 发送日期、时间和温度数据

/*************************************************************************
********/
// 定时器初始化及中断处理函数声明
/*************************************************************************
********/
void TMR32B0_Init(void);  // 初始化 32 位定时器 0
void TIMER32_0_IRQHandler(void);  // 32 位定时器 0 的中断处理函数

/*************************************************************************
********/
 /**
 * 函数功能：重定向 printf 到 UART
 * 描述：此函数实现了 printf 的底层重定向，使得所有的 printf 输出都通过 UART 进行
发送。
 * 参数：
 *      ch - 要输出的字符
 *      f  - 文件指针（对于此函数不使用，仅作为 printf 的标准接口）
 * 返回值：
 *      返回输出的字符（保持原始格式）
 */
int fputc(int ch, FILE *f)
```

```
{
    UART_Send_Byte(ch);  // 将字符通过 UART 发送
    return ch;           // 返回输出字符
}

/*********************************************/
/*              主函数                       */
/*********************************************/
int main()
{
    SystemInit();    // 系统时钟初始化
    I2C_Init(1);     // 初始化 I2C 通信（快速模式）
    SSPI1_Init();    // 初始化 SPI 接口
    UART_Init(115200);  // 初始化 UART，波特率为 115200
    T16B1_PWM_init();  // 初始化 16 位定时器 1 为 PWM 模式，控制蜂鸣器
    T16B0_init();      // 初始化 16 位定时器 0
    TMR32B0_Init();    // 初始化 32 位定时器 0（用于中断处理），1 秒钟在串口打印一次数据并且在 OLED 屏幕上显示
    RGB_LED_Init();    // 初始化 RGB LED
    DS1307Init();    // 仅第一次烧录时用来校准时间
    DS1307_Read();     // 读取 DS1307 时间
    SPI1_Read_FLASH(Temp_Buf, 8); // 从 SPI 闪存读取温度数据
    OLED_Init();       // 初始化 OLED 显示屏
    OLED_Clear();      // 清除 OLED 屏幕内容
    PassiveBuzzer_Init();  // 初始化蜂鸣器
    T16B0_delay_ms(200);  // 延时 200ms，等待硬件初始化完成
    Read_Temp_LM75BD();   // 第一次读取温度（过滤掉初始不准确的值）
    PassiveBuzzer_OFF();  // 关闭蜂鸣器
    while(1)
    {

    }
}

/*****************************************************************
 * 函数名: TIMER32_0_IRQHandler
 * 描述: 定时器 0 中断处理函数，触发时进行温度数据读取与处理，1 秒钟在串口打印一次数据并且在 OLED 屏幕上显示
 * 在定时器 0 的匹配事件发生时，启动 ADC 转换，读取 ADC7 的值，处理温度数据并更新显示。
 *****************************************************************/
void TIMER32_0_IRQHandler(void) {
    // 检查定时器 0 的匹配中断标志（MR0）
    if ((LPC_TMR32B0->IR |= 0x01) == 1) {
```

```
            // 从 SPI 闪存读取温度数据
            SPI1_Read_FLASH(Temp_Buf, 8);

            // 打印调试信息，输出到串口
            printf("Hainan University\n");
            printf("Teacher: Wang Qingji\n");
            printf("Liu Jiajun\n");
            printf("20223004426\n");

            // 读取 DS1307 时间
            DS1307_Read();

            // 进行温度中位数滤波处理
            Temp_LM75BD = MedianFilter();

            // 将温度数据转换为字符串
            Transfor(Temp_Buf, Temp_LM75BD);

            // 通过 UART 发送日期、时间和温度数据
            UART_Send_Date();

            // 打印当前温度
            printf("Temperature: %s\n", Temp_Buf);
            printf(" \n");

            // 显示日期、时间和温度在 OLED 屏幕
            Menu();

            // 根据当前温度执行判断操作（例如触发报警或控制 RGB 灯）
            Temperature_Judgment(Temp_LM75BD);

            // 执行 XT25 芯片的扇区擦除操作
            XT25_EraseSector();

            // 将温度数据写入到 XT25 闪存
            SPI1_Write_FLASH(Temp_Buf, 8);

            // 清除定时器 0 的匹配中断标志
            LPC_TMR32B0->IR = 0x01;
        }
    }
```

```
/**
 * 函数功能：显示当前信息到 OLED 屏幕
 * 描述：在 OLED 屏幕上依次显示当前日期、时间和温度。日期、时间和温度的数据通过全局变量传递。
 * OLED 显示的位置和数据格式固定，显示日期、时间和温度值。
 */
void Menu(void)
{
    /*OLED 显示*/
    // OLED_Clear(); // 清除 OLED 显示屏（注释掉了此行）
    OLED_ShowString(1, 4, "LiuJiajun ! ");    // 在第 1 行第 4 列显示字符串 "LiuJiajun !"
    OLED_ShowString(2, 1, "DATE: ");          // 在第 2 行第 1 列显示"DATE: "
    // 显示日期（Date[]数组保存了日期数据）
    OLED_ShowChar(2, 7, Date[0]);
    OLED_ShowChar(2, 8, Date[1]);
    OLED_ShowChar(2, 9, Date[2]);
    OLED_ShowChar(2, 10, Date[3]);
    OLED_ShowChar(2, 11, Date[4]);
    OLED_ShowChar(2, 12, Date[5]);
    OLED_ShowChar(2, 13, Date[6]);
    OLED_ShowChar(2, 14, Date[7]);
    OLED_ShowChar(2, 15, Date[8]);
    OLED_ShowChar(2, 16, Date[9]);

    OLED_ShowString(3, 1, "TIME:  ");         // 在第 3 行第 1 列显示"TIME: "
    // 显示时间（Time_Data[]数组保存了时间数据）
    OLED_ShowChar(3, 8, Time_Data[0]);
    OLED_ShowChar(3, 9, Time_Data[1]);
    OLED_ShowChar(3, 10, Time_Data[2]);
    OLED_ShowChar(3, 11, Time_Data[3]);
    OLED_ShowChar(3, 12, Time_Data[4]);
    OLED_ShowChar(3, 13, Time_Data[5]);
    OLED_ShowChar(3, 14, Time_Data[6]);
    OLED_ShowChar(3, 15, Time_Data[7]);
```

```
    OLED_ShowString(4, 1, "Temp:  ");         // 在第 4 行第 1 列显示"Temp: "
    // 显示温度（Temp_Buf[]数组保存了温度数据）
    OLED_ShowChar(4, 8, Temp_Buf[0]);
    OLED_ShowChar(4, 9, Temp_Buf[1]);
    OLED_ShowChar(4, 10, Temp_Buf[2]);
    OLED_ShowChar(4, 11, Temp_Buf[3]);
    OLED_ShowChar(4, 12, Temp_Buf[4]);
    OLED_ShowChar(4, 13, Temp_Buf[5]);
    OLED_ShowString(4, 15, "C");              // 在温度后显示"℃"
}

/**
 * 函数功能：根据温度判断是否报警，并控制 RGB 灯和蜂鸣器
 * 描述：根据温度 T 值的不同，控制蜂鸣器和 RGB 灯的状态。
 *       温度高于 30℃时启动蜂鸣器，并熄灭 RGB 灯；温度在不同范围内时，控制 RGB
灯的颜色。
 * 参数：
 *      T - 当前温度（单位：摄氏度）
 */
void Temperature_Judgment(float T)
{
    if (T > 30) {
        PassiveBuzzer_ON(); // 启动蜂鸣器
        // 打开所有 RGB 灯（根据 GPIO 控制灯的开关状态）
        LPC_GPIO2->DATA &= ~(1UL << 10);
        LPC_GPIO2->DATA &= ~(1UL << 9);
        LPC_GPIO2->DATA &= ~(1UL << 8);
        LPC_GPIO1->DATA &= ~(1UL << 10);
    }
    else if (0 < T && T <= 10) {
        LPC_GPIO2->DATA |= (1UL << 10);
        LPC_GPIO2->DATA |= (1UL << 9);
        PassiveBuzzer_OFF();  // 关闭蜂鸣器
        RGB_Green_Toggle();   // 切换绿色 RGB 灯
    }
    else if (10 < T && T <= 20) {
        LPC_GPIO2->DATA |= (1UL << 9);
        LPC_GPIO2->DATA |= (1UL << 8);
        PassiveBuzzer_OFF();  // 关闭蜂鸣器
        RGB_Blue_Toggle();    // 切换蓝色 RGB 灯
    }
    else if (20 < T && T <= 30) {
        LPC_GPIO2->DATA |= (1UL << 10);
        LPC_GPIO2->DATA |= (1UL << 8);
```

```
        PassiveBuzzer_OFF();  // 关闭蜂鸣器
        RGB_Red_Toggle();     // 切换红色 RGB 灯
    }
}

/**
 * 函数功能：将浮动温度值转换为字符串
 * 描述：将传入的浮动温度值转换为字符串格式，保留三位小数。
 * 参数：
 *      Temp_Str - 用于保存温度字符串的缓冲区
 *      T - 当前温度（浮动类型）
 */
void Transfor(uint8_t *Temp_Str, float T) {
    sprintf((char *)Temp_Str, "%.3f", T); // 将浮动温度 T 转换为字符串，保留三位小数
}

/**
 * 函数功能：中位数滤波
 * 描述：通过多次温度测量并采用中位数滤波算法，去除极端值，得到更加平稳的温度值。
 * 返回值：
 *      返回去除极端值后的温度值（单位：摄氏度）
 */
float MedianFilter(void) {
    float measurements[7]; // 用于存储 7 次温度测量值
    float temp;

    // 进行 7 次温度测量
    for (unsigned int i = 0; i < 7; i++) {
        measurements[i] = Read_Temp_LM75BD(); // 读取传感器温度值
    }

    // 冒泡排序，找出中位数
    for (unsigned int i = 0; i < 7 - 1; i++) {
        for (unsigned int j = 0; j < 7 - i - 1; j++) {
            if (measurements[j] > measurements[j + 1]) {
                // 交换
                temp = measurements[j];
                measurements[j] = measurements[j + 1];
                measurements[j + 1] = temp;
            }
        }
    }
```

```
    // 去掉最大值和最小值，计算中间 5 个值的平均值
    float sum = 0;
    for (unsigned int i = 1; i < 7 - 1; i++) { // 从 1 到 N-2，即中间 5 个数
        sum += measurements[i];
    }

    // 计算并返回平均值
    return (sum / (7 - 2)); // 去掉 2 个值，所以用(N - 2)
}

/**
 * 函数功能：发送日期、时间和星期数据通过 UART
 * 描述：将日期、时间和星期信息通过 UART 发送到串口，便于外部设备查看。
 */
void UART_Send_Date(void)
{
    UART_Send("Date: ", 6);        // 发送日期前缀
    UART_Send(Date, 10);           // 发送日期数据
    UART_Send(" \n", 2);           // 换行
    UART_Send("Time: ", 6);        // 发送时间前缀
    UART_Send(Time_Data, 8);       // 发送时间数据
    UART_Send(" \n", 2);           // 换行
    UART_Send("Week: ", 6);        // 发送星期前缀
    UART_Send(Week, 1);            // 发送星期数据
    UART_Send(" \n", 2);           // 换行
}
/*****************************************************************
 * 函数名: TMR32B0_Init
 * 描述:初始化 32 位定时器 0，开启定时器匹配中断。
*****************************************************************/
void TMR32B0_Init(void) {
    LPC_SYSCON->SYSAHBCLKCTRL |= (1UL << 9); // 使能定时器 0 时钟
    LPC_TMR32B0->IR = 0x1F; // 清除所有中断标志
    LPC_TMR32B0->PR = 0; // 不分频
    LPC_TMR32B0->MCR = 3; // 当匹配 MR0 时，重置定时器和产生中断
    LPC_TMR32B0->MR0 = SystemCoreClock - 1; // 设置匹配寄存器为系统时钟频率
-1(1 秒)
    LPC_TMR32B0->TCR = 0x01; // 启动定时器
    NVIC_EnableIRQ(TIMER_32_0_IRQn); // 使能定时器 0 中断
}

/*****************************************************************
********/
/*********************************************/
```

```
/*  温度传感器 LM75BD 的数据读取函数  */
/*****************************************/
float Read_Temp_LM75BD(void)
{
  uint16_t Temputerature_8,Temputerature_16;  //温度值，1 次接收 8 位
  float Temputerature;   //存储获得的温度值
  //IIC 启动---默认配置温度模式
      //发送(0x91)1001 0001：1001,硬件布线； 0001，从机地址--000   读模式--1
  I2C_Start();
  I2C_Send_Byte(0x91);
    LPC_I2C->CONSET =(1<<2);//AA=1
  Temputerature_8 = I2C_Recieve_Byte();//读 LM75BD 温度寄存器的高八位数据
LPC_I2C->CONCLR =(1<<2);//AA=0
  Temputerature_16 = (Temputerature_8 <<8)+(I2C_Recieve_Byte());
  I2C_Stop();

  Temputerature_16 = Temputerature_16 >> 5;//1LSB=0.125℃---低五位为无效数
据(移出)
  /* Temputerature_16:温度寄存器的数据 D0--D10:其中 D10 为温度的正负数据*/

  //负温度
  if(Temputerature_16&0x0400)
  Temputerature = (-(~(Temputerature_16&0x03FF)+1)*0.125);//负温度的数据的
转换(二进制的补码+1)
  //正温度
  else
  Temputerature = (0.125*(float)(Temputerature_16));

  //返回温度值 1LSB=0.01℃
  return Temputerature;
}
/*****************************************************************
********/
/**
 * 函数功能：初始化 16 位定时器 0
 * 描述：配置并初始化 16 位定时器 0，设置定时器的时钟源、计数器模式和匹配寄存器，
确保定时器可以正常工作。
 */
void T16B0_init(void)
{
    LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 7);  // 使能 16 位定时器 0 的时钟
（TIM16B0）
    LPC_TMR16B0->IR = 0x01;                 // 清除定时器 0 的中断标志，设置
中断源为 MR0（匹配 0 寄存器）
```

```
    LPC_TMR16B0->MCR = 0x04;                    // 配置匹配控制寄存器（MCR）：匹配 0 后重置 TC，不生成中断（TCR[0]=0）
}

/**
 * 函数功能：延时函数（毫秒级）
 * 描述：使用 16 位定时器 0 生成精确的毫秒级延时。
 * 参数：
 *      ms - 延迟的毫秒数（最大 65535ms，即 65.535 秒）。
 */
void T16B0_delay_ms(uint16_t ms)
{
    LPC_TMR16B0->TCR = 0x02;                    // 重置定时器（TCR[1] = 1），停止定时器
    LPC_TMR16B0->PR = SystemCoreClock / 1000 - 1; // 设置定时器预分频器，确保定时器计数器按 1ms 递增
    LPC_TMR16B0->MR0 = ms;                      // 设置 MR0 为目标延时值（单位：毫秒）
    LPC_TMR16B0->TCR = 0x01;                    // 启动定时器（TCR[0] = 1）
    LPC_TMR16B0->TCR = 0x01;                    // 启动定时器（TCR[0] = 1），重新启动
    while (LPC_TMR16B0->TCR & 0x01);            // 等待定时器计数完成，直到 TCR[0]为 0，表示延时结束
}

/**
 * 函数功能：延时函数（微秒级）
 * 描述：使用 16 位定时器 0 生成精确的微秒级延时。
 * 参数：
 *      us - 延迟的微秒数（最大 65535us，即 65.535ms）。
 */
void T16B0_delay_us(uint16_t us)
{
    LPC_TMR16B0->TCR = 0x02;                    // 重置定时器（TCR[1] = 1），停止定时器
    LPC_TMR16B0->PR = SystemCoreClock / 1000000 - 1; // 设置定时器预分频器，确保定时器计数器按 1us 递增
    LPC_TMR16B0->MR0 = us;                      // 设置 MR0 为目标延时值（单位：微秒）
    LPC_TMR16B0->TCR = 0x01;                    // 启动定时器（TCR[0] = 1）
    while (LPC_TMR16B0->TCR & 0x01);            // 等待定时器计数完成，直到 TCR[0]为 0，表示延时结束
}
```

```
/*****************************************************************
********/
/**
 * 函数功能：初始化 16 位定时器 1 为 PWM 模式
 * 描述：配置定时器 1 的 PWM 功能，使其能够生成 PWM 信号，输出到 PIO1_10 管脚，控
制被动蜂鸣器的频率和占空比。
 */
void T16B1_PWM_init(void)
{
    LPC_SYSCON->SYSAHBCLKCTRL |= (1UL << 8) | (1UL << 16); // 使能 16 位定
时器 1 和 IOCON 的时钟
    LPC_IOCON->PIO1_10 |= 0x02;  // 将 PIO1_10 复用为 MAT1（PWM 输出功能）
    LPC_TMR16B1->TCR = 0x02;  // 复位 16 位定时器 1
    LPC_TMR16B1->PR = 0;  // 设置预分频寄存器为 0，不分频，计数频率等于系统时
钟频率
    LPC_TMR16B1->PWMC = 0x02;  // 配置 MAT1 为 PWM 模式
    LPC_TMR16B1->MCR = (0x02 << 9);  // 设置匹配寄存器 3（MR3）匹配后重置计
数器 TC，且不产生中断
    LPC_TMR16B1->MR3 = SystemCoreClock / 1000;  // 设置 MR3 为 1ms（PWM 周期
为 1ms）
    LPC_TMR16B1->MR1 = LPC_TMR16B1->MR3; // MR1 控制 PWM 的占空比，默认与 MR3
相等，占空比为 100%
    LPC_TMR16B1->TCR = 0x01;  // 使能 16 位定时器 1
}

/**
 * 函数功能：初始化被动蜂鸣器的 GPIO
 * 描述：配置 PIO1_10 为 GPIO 输出模式，默认关闭 GPIO 时钟以降低功耗。
 */
void PassiveBuzzer_Init(void)
{
    LPC_SYSCON->SYSAHBCLKCTRL |= (1UL << 6); // 使能 GPIO 时钟
    LPC_GPIO1->DIR |= (1UL << 10); // 设置 PIO1_10 为输出模式
    LPC_SYSCON->SYSAHBCLKCTRL &= ~(1 << 16);   // 禁用 IOCON 时钟，确保低功
耗
}

/**
 * 函数功能：关闭被动蜂鸣器
 * 描述：关闭定时器 1 的 PWM 功能，复位相关寄存器，关闭 IOCON 和定时器时钟以降低功
耗。
 */
void PassiveBuzzer_OFF(void)
{
```

```
    LPC_SYSCON->SYSAHBCLKCTRL |= (1UL << 16); // 使能 IOCON 时钟
    LPC_IOCON->PIO1_10 &= ~0x3F;  // 清除 PIO1_10 的功能设置
    LPC_GPIO1->DIR &= ~(1UL << 10); // 将 PIO1_10 设置为输入模式
    LPC_TMR16B1->MR1 = 0;  // 将 MR1 设为 0，停止 PWM 输出
    LPC_TMR16B1->TCR = 0x00; // 停止 16 位定时器 1
    LPC_SYSCON->SYSAHBCLKCTRL &= ~(1UL << 8); // 关闭定时器 1 的时钟
    LPC_SYSCON->SYSAHBCLKCTRL &= ~(1 << 16); // 禁用 IOCON 时钟以降低功耗
}

/**
 * 函数功能：开启被动蜂鸣器
 * 描述：配置并启用定时器 1 的 PWM 功能，以 MAT1 输出 PWM 信号控制蜂鸣器发声。
 */
void PassiveBuzzer_ON(void)
{
    LPC_SYSCON->SYSAHBCLKCTRL |= (1UL << 16); // 使能 IOCON 时钟
    LPC_IOCON->PIO1_10 |= 0x02;  // 将 PIO1_10 复用为 MAT1（PWM 功能）
    LPC_TMR16B1->TCR = 0x01;  // 使能 16 位定时器 1
    LPC_SYSCON->SYSAHBCLKCTRL |= (1UL << 8);  // 使能定时器 1 的时钟
    LPC_SYSCON->SYSAHBCLKCTRL &= ~(1 << 16); // 禁用 IOCON 时钟以降低功耗
    LPC_TMR16B1->MR3 = SystemCoreClock / 1000 - 1;  // 设置 MR3 为 1ms（PWM
周期为 1ms）
    LPC_TMR16B1->MR1 = LPC_TMR16B1->MR3 * 50 / 100; // 设置 MR1 为 50%占空
比
}
/**************************************************************************
********/
/*************************************************************
 * 函数名: UARTInit
 * 描述: 初始化 UART0 端口，设置选中引脚、时钟，校验、停止位、FIFO、等等。
 * 参数: UART 波特率
 * 返回值: 无
*************************************************************/
  void UART_Init(uint32_t baudrate)
  {
    uint32_t Fdiv; // 分频值
    uint32_t regVal = regVal; // 临时变量
    LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 16);  // 使能 IOCON 模块的时钟
    LPC_IOCON->PIO1_6 &= ~0x07;  // UART I/O 配置，设置 PIO1_6 为 UART RXD
    LPC_IOCON->PIO1_6 |= 0x01;
    LPC_IOCON->PIO1_7 &= ~0x07; // 设置 PIO1_7 为 UART TXD
    LPC_IOCON->PIO1_7 |= 0x01;
    LPC_SYSCON->SYSAHBCLKCTRL &= ~(1 << 16); // 关闭 IOCON 模块的时钟
    LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 12); // 使能 UART 时钟
```

```
    LPC_SYSCON->UARTCLKDIV = 0x1; // 分频值 1
    LPC_UART->LCR = 0x83; // 设置 UART 线控制寄存器 (LCR)，配置为 8 位数据，
奇校验位，1 停止位
    Fdiv = (SystemCoreClock / 16) / baudrate; // 计算波特率分频值
    LPC_UART->DLM = Fdiv / 256; // 设置波特率低字节和高字节
    LPC_UART->DLL = Fdiv % 256;
    LPC_UART->LCR = 0x03; // 重置 LCR，禁用 DLAB
    LPC_UART->FCR = 0x07; // 使能并复位 TX 和 RX FIFO
    regVal = LPC_UART->LSR; // 读取并清除行状态寄存器 (LSR)
    // 确保正常开始，TX 和 RX FIFO 中没有数据
    while ((LPC_UART->LSR & (0x20 | 0x40)) != (0x20 | 0x40));
    while (LPC_UART->LSR & 0x01)
    {
        regVal = LPC_UART->RBR; // 从 RX FIFO 中转储数据
    }
    LPC_UART->IER |=1<<0; //使能接受中断
    NVIC_EnableIRQ(UART_IRQn); //启动中断
  }


/*************************************************/
/* 函数功能：串口接收字节数据  */
/*************************************************/
uint8_t UART_Recive(void)
{
  while(!(LPC_UART->LSR & (1<<0)));   //等待接收到数据
  return(LPC_UART->RBR);   //读出数据
}

/*************************************************/
/* 函数功能：串口发送字节数据  */
/*************************************************/
void UART_Send_Byte(uint8_t byte)
{
    LPC_UART->THR = byte;
  while ( !(LPC_UART->LSR & (1<<5)) );//等待发送完

}

/*****************************************************************
 * 函数名：UARTSend
 * 描述：根据数据长度发送一串数据到 UART 0 端口。
 * 参数：buffer pointer, and data length
 * 返回值：无
*****************************************************************/
```

```
 void UART_Send(uint8_t *BufferPtr, uint32_t Length)
 {
    while (Length != 0) // 判断数据长度，不为 0 则发送数据
    {
        while (!(LPC_UART->LSR & (1 << 5))); // 等待直到发送数据寄存器空
        LPC_UART->THR = *BufferPtr; // 发送当前字节
        BufferPtr++; // 移动到下一个字节
        Length--; // 递减数据长度
    }
 }
/*********************************************************************
********/
void I2C_Init(uint8_t Mode)
{
  LPC_SYSCON->PRESETCTRL |= (1<<1);   //使 I2C 上电 I2C 模块（在启动 I2C 模块
之前，必须向该位写 1）
  LPC_SYSCON->SYSAHBCLKCTRL |= (1<<5);   //使能 I2C 时钟
  LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16);   // 使能 IOCON 时钟
  LPC_IOCON->PIO0_4 &= ~0x3F;
  LPC_IOCON->PIO0_4 |= 0x01;   //把 P0.4 脚配置为 I2C SCL
  LPC_IOCON->PIO0_5 &= ~0x3F;
  LPC_IOCON->PIO0_5 |= 0x01;   //把 P0.5 脚配置为 I2C SDA

     /*--- Reset registers ---*/
    if(Mode == 1)
    {
     LPC_I2C->SCLL   = I2SCLL_HS_SCLL;
     LPC_I2C->SCLH   = I2SCLH_HS_SCLH;
    }
    else
    {
    LPC_I2C->SCLH = I2SCLH_SCLH;
    LPC_I2C->SCLL = I2SCLL_SCLL;
    }
 /*--- Clear flags ---*/
  LPC_I2C->CONCLR = I2CONCLR_AAC | I2CONCLR_SIC | I2CONCLR_STAC |
I2CONCLR_I2ENC;

  LPC_SYSCON->SYSAHBCLKCTRL &= ~(1<<16);   // 禁能 IOCON 时钟
  LPC_I2C->CONSET |= I2CONSET_I2EN;  // 使能 I2C 接口
}


void I2C_Start(void)
{
```

```
  LPC_I2C->CONSET |= I2CONSET_STA;  // 使能发送开始信号
  while(!(LPC_I2C->CONSET & I2CONSET_SI));  // 等待开始信号发送完成
  LPC_I2C->CONCLR = I2CONCLR_STAC | I2CONCLR_SIC; // 清除开始 START 位和 SI
位
}


void I2C_Stop(void)
{
   LPC_I2C->CONCLR = I2CONCLR_SIC;  // 清 SI 标志位
   LPC_I2C->CONSET |= I2CONSET_STO;  // 发送停止信号
   while( LPC_I2C->CONSET & I2CONSET_STO );  // 等待停止信号发送完成
}

/*****************************************************/
/* 函数功能：I2C 发送一字节数据  */
/* 入口参数：dat : 要发送的字节  */
/*****************************************************/
void I2C_Send_Byte(uint8_t dat)
{
   LPC_I2C->DAT = dat;  // 把字节写入 DAT 寄存器
   LPC_I2C->CONCLR = I2CONSET_SI;  // 清除 SI 标志(1<<3)
   while(!(LPC_I2C->CONSET & I2CONSET_SI));  // 等待数据发送完成
}


/*****************************************************/
/* 函数功能：I2C 接收一字节数据  */
/* 入口参数：rebyte : 要接收的字节  */
/*****************************************************/
uint8_t I2C_Recieve_Byte(void)
{
  uint8_t ReByte;
  LPC_I2C->CONCLR = I2CONCLR_SIC;   // 清 AA 和 SI 标志
  while(!(LPC_I2C->CONSET & I2CONSET_SI));  // 等待接收数据完成
  ReByte = (uint8_t)LPC_I2C->DAT;  // 把接收到的数据给了 rebyte
  return ReByte;
}
/*****************************************************************************
********/
/*
*函数功能：SSP1IO 初始化
*/
void SSP1_IOConfig(void){
    LPC_SYSCON->SYSAHBCLKCTRL |=((1<<6)|(1<<16)); //IO 和 GPIO
```

```
    LPC_IOCON->PIO2_2 &=~(0X07); //
    LPC_IOCON->PIO2_2 |=0X02;// MISO
    LPC_IOCON->PIO2_3 &=~(0X07); //
    LPC_IOCON->PIO2_3 |=0X02;//MOSI  此处可能出现问题
    LPC_IOCON->PIO2_1 &=~(0X07); //
    LPC_IOCON->PIO2_1 |=0X02;//CLK
    LPC_GPIO2->DIR |= (1<<0);
    LPC_GPIO2->DATA |= (1<<0); //拉高
}

/**
*函数功能：SSP1 初始化
*/
void SSPI1_Init(void){
    uint8_t Dummy=Dummy; //随机数
    uint8_t i;

    LPC_SYSCON->PRESETCTRL |=1<<2; //复位 ssp1
    LPC_SYSCON->SYSAHBCLKCTRL |=(1<<18); //ssp1 时钟使能
    LPC_SYSCON->SSP1CLKDIV=0X02 ;  //分频值 2 48/2=24

    SSP1_IOConfig(); //初始化 SSP1 IO 口
    LPC_SSP1->CR0=0X0707;   //CPSR=7 DATAbit= 8 CPOL=0 CPHA=0 SCR=7  选择
spi
    LPC_SSP1->CPSR=0X2;   //SCLK=48/(2*2*8)= 1.5M
    LPC_SSP1->CR1 &=~(1<<0);//LBM=0 :正常模式
    LPC_SSP1->CR1 &=~(1<<2);//MS=0 主机模式
    LPC_SSP1->CR1 |=1<<1; //SSE=1 开启 ssp1
    //清空 RXFIFO
    for(i=0 ; i<8 ;i++){
        Dummy = LPC_SSP1->DR;
    }

    for(i=0;i<16;i++){
        Read_Addr[i]=0;
        Write_Addr[i]=0;
    }

}

void SSP1_LOW(void){
    LPC_GPIO2->DATA &= ~(1<<0); //拉低
}
void SSP1_HIGH(void){
```

```
    LPC_GPIO2->DATA |= (1<<0); //拉高
}

/**
* 函数功能：SSP1 通信
* 接受和发送一个字符
*/
uint8_t SPI1_Comunicate(uint8_t TxData){
    while(((LPC_SSP1->SR)&(1<<4))==(1<<4));//忙时等待
    LPC_SSP1->DR=TxData; // 发送数据到 TxFIFO
    while(((LPC_SSP1->SR)&(1<<2))!=(1<<2));// 等待数据接受完
    return (LPC_SSP1->DR); // 接受返回的数据
}

/**
*函数功能：SSP1 发送
*/
void SSP1_Send(uint8_t *data,uint8_t Length){
    uint8_t i;
    for(i=0;i<Length;i++){
        SPI1_Comunicate(data[i]);
    }
}

/**
* 函数功能：SSP1 接受
*/
void SSP1_Receive(uint8_t *data,int Length){
    uint8_t Dummy=Dummy; // 随机数用于产生时钟
    uint8_t i;
    for(i=0 ; i<Length ;i++){
        data[i]=SPI1_Comunicate(0xff);
    }
}
/*************************************************************************
********/
/**
 * 函数功能：写入使能
 * 描述：向 XT25 芯片发送写入使能命令（WREN），
 *       设置写入使能位（WEL=1），以允许后续的写入操作。
 */
void XT25_WriteEnable(void) {
    SSP1_LOW();               // 拉低片选信号，使能 SPI 通信
    SPI1_Comunicate(WREN);    // 发送写入使能命令（WREN）
```

```
    SSP1_HIGH();                // 拉高片选信号，结束通信
}

/**
 * 函数功能：读取状态寄存器
 * 描述：通过 SPI 读取 XT25 的状态寄存器，获取当前芯片的状态信息。
 * 返回值：
 *      uint8_t - 状态寄存器的值。
 */
uint8_t XT25_ReadSR(void) {
    uint8_t sr;
    SSP1_LOW();                     // 拉低片选信号，使能 SPI 通信
    SPI1_Comunicate(RDSR);          // 发送读取状态寄存器命令（RDSR）
    sr = SPI1_Comunicate(0xFF);     // 接收状态寄存器数据
    SSP1_HIGH();                    // 拉高片选信号，结束通信
    return sr;                      // 返回状态寄存器值
}

/**
 * 函数功能：忙碌等待 - 写入等待
 * 描述：检查 XT25 的状态寄存器，等待写入操作完成（WIP=0，写操作完成）。
 */
void XT25_Write_Wait(void) {
    int stat_code = 0;
    while (1) {
        stat_code = XT25_ReadSR(); // 读取状态寄存器
        if (((stat_code & (1 << 1)) == 0x02) && (stat_code & 1) == 0) { //
检查 WIP=0 且写保护位
            break; // 写入完成，退出等待
        }
    }
}

/**
 * 函数功能：忙碌等待 - 读出等待
 * 描述：检查 XT25 的状态寄存器，等待芯片准备完成（WIP=0）。
 */
void XT25_Read_Wait(void) {
    int stat_code = 0;
    while (1) {
        stat_code = XT25_ReadSR(); // 读取状态寄存器
        if ((stat_code & 1) == 0) { // 检查 WIP=0（芯片不忙）
            break; // 芯片准备完成，退出等待
        }
```

```
    }
}

/**
 * 函数功能：写入状态寄存器
 * 描述：通过 SPI 向 XT25 的状态寄存器写入数据，用于配置芯片的状态。
 * 参数：
 *      sr - 要写入状态寄存器的数据。
 */
void XT25_WriteSR(uint8_t sr) {
    XT25_WriteEnable();       // 使能写入操作
    T16B0_delay_ms(50);       // 延时 50ms，确保稳定
    SSP1_LOW();               // 拉低片选信号，使能 SPI 通信
    SPI1_Comunicate(0x01);    // 发送写入状态寄存器命令
    SPI1_Comunicate(sr);      // 写入状态寄存器数据
    SSP1_HIGH();              // 拉高片选信号，结束通信
}

/**
 * 函数功能：读取唯一 ID
 * 描述：通过 SPI 读取 XT25 芯片的唯一 ID，用于芯片唯一性标识。
 */
void XT25_RUID(void) {
    Write_Addr[0] = 0x4B;     // 设置读取唯一 ID 命令
    Write_Addr[1] = 0x00;     // 保留字节
    Write_Addr[2] = 0x00;     // 保留字节
    Write_Addr[3] = 0x00;     // 保留字节
    SSP1_LOW();               // 拉低片选信号，使能 SPI 通信
    SSP1_Receive((uint8_t *)Read_Addr, 16); // 接收 16 字节唯一 ID 数据
    SSP1_HIGH();              // 拉高片选信号，结束通信
}

/**
 * 函数功能：擦除整个芯片
 * 描述：通过 SPI 向 XT25 发送擦除全部存储区的命令。
 */
void XT25_EraseAll(void) {
    XT25_WriteSR(0x00);       // 清除写保护位（BP1=0, BP0=0）
    T16B0_delay_ms(50);       // 延时 50ms，确保稳定
    XT25_WriteEnable();       // 使能写入操作
    T16B0_delay_ms(50);       // 延时 50ms，确保稳定
    SSP1_LOW();               // 拉低片选信号，使能 SPI 通信
    SPI1_Comunicate(0x60);    // 发送芯片擦除命令
    SSP1_HIGH();              // 拉高片选信号，结束通信
```

```
}

/**
 * 函数功能：扇区擦除
 * 描述：通过 SPI 擦除 XT25 的指定扇区，释放存储空间。
 */
void XT25_EraseSector(void) {
    XT25_WriteSR(0x00);       // 清除写保护位（BP1=0, BP0=0）
    T16B0_delay_ms(50);       // 延时 50ms，确保稳定
    XT25_WriteEnable();       // 使能写入操作
    T16B0_delay_ms(50);       // 延时 50ms，确保稳定
    SSP1_LOW();               // 拉低片选信号，使能 SPI 通信
    SPI1_Comunicate(0x20);    // 发送扇区擦除命令
    SPI1_Comunicate(0x22);    // 指定扇区地址（示例地址 0x222222）
    SPI1_Comunicate(0x22);
    SPI1_Comunicate(0x22);
    SSP1_HIGH();              // 拉高片选信号，结束通信
    T16B0_delay_ms(100);      // 延时 100ms，等待擦除完成
}

/**
 * 函数功能：写入数据到 FLASH
 * 描述：通过 SPI 将数据写入 XT25 芯片的指定地址。
 * 参数：
 *      data - 要写入的数据指针。
 *      Length - 要写入的数据长度。
 */
void SPI1_Write_FLASH(uint8_t *data, uint8_t Length) {
    XT25_WriteEnable();       // 使能写入操作
    T16B0_delay_ms(50);       // 延时 50ms，确保稳定
    Write_Addr[0] = WRITE;    // 页面编程命令
    Write_Addr[1] = 0x22;     // 指定地址（示例地址 0x222222）
    Write_Addr[2] = 0x22;
    Write_Addr[3] = 0x22;

    // 填充要写入的数据
    for (int i = 0; i < Length; i++) {
        Write_Addr[i + 4] = data[i];
    }

    XT25_Write_Wait();        // 等待写入完成
    SSP1_LOW();               // 拉低片选信号，使能 SPI 通信
    SSP1_Send((uint8_t *)Write_Addr, 4 + Length); // 发送写入命令和数据
    SSP1_HIGH();              // 拉高片选信号，结束通信
```

```
}

/**
 * 函数功能：读取 FLASH 中的数据
 * 描述：通过 SPI 从 XT25 芯片的指定地址读取数据。
 * 参数：
 *      data - 用于存储读取数据的指针。
 *      Length - 要读取的数据长度。
 */
void SPI1_Read_FLASH(uint8_t *data, uint8_t Length) {
    Write_Addr[0] = READ;     // 读取命令
    Write_Addr[1] = 0x22;     // 指定地址（示例地址 0x222222）
    Write_Addr[2] = 0x22;
    Write_Addr[3] = 0x22;

    XT25_Read_Wait();         // 等待芯片准备完成
    SSP1_LOW();               // 拉低片选信号，使能 SPI 通信
    SSP1_Send((uint8_t *)Write_Addr, 4);         // 发送读取命令和地址
    SSP1_Receive((uint8_t *)Read_Addr, Length); // 接收读取的数据
    SSP1_HIGH();              // 拉高片选信号，结束通信

    // 将读取的数据拷贝到目标缓冲区
    for (int i = 0; i < Length; i++) {
        data[i] = Read_Addr[i];
    }
}
/*************************************************************************
********/
/**
 * 函数功能：初始化 DS1307 并设置时间
 * 描述：通过调用写函数将初始时间数据写入 DS1307 模块，
 *       从而完成时钟芯片的初始化。
 */
void DS1307Init(void)
{
    DS1307_Write(Time); // 将时间数据写入 DS1307 芯片的寄存器中
}

/**
 * 函数功能：读取 DS1307 的时间和日期
 * 描述：通过 I2C 接口读取 DS1307 的内部寄存器内容，
 *       并提取时间和日期数据，同时进行格式化处理。
 */
void DS1307_Read(void)
```

```
{
    // 定位 DS1307 的内部指针到 RAM 尾部 0x3f 地址
    Ds1307_WriteByte(0x3f, 0x01);

    I2C_Start(); // 发送 I2C 起始信号
    I2C_Send_Byte(0xD1); // 发送设备地址（读操作，地址为 0xD1）
    LPC_I2C->CONSET = (1 << 2); // 设置 ACK（AA=1）以应答下一个字节的数据

    // 从 DS1307 读取 7 个字节数据，依次为秒、分、时、星期、日、月、年
    data[0] = I2C_Recieve_Byte(); // 秒
    data[1] = I2C_Recieve_Byte(); // 分
    data[2] = I2C_Recieve_Byte(); // 时
    data[3] = I2C_Recieve_Byte(); // 星期
    data[4] = I2C_Recieve_Byte(); // 日
    data[5] = I2C_Recieve_Byte(); // 月
    LPC_I2C->CONCLR = (1 << 2);   // 清除 ACK（AA=0）
    data[6] = I2C_Recieve_Byte(); // 年

    // 将读取的时间数据格式化为字符串（例如：12:34:56）
    Time_Data[0] = data[2] / 16 + '0';  // 时的十位
    Time_Data[1] = data[2] % 16 + '0';  // 时的个位
    Time_Data[2] = ':';
    Time_Data[3] = data[1] / 16 + '0';  // 分的十位
    Time_Data[4] = data[1] % 16 + '0';  // 分的个位
    Time_Data[5] = ':';
    Time_Data[6] = data[0] / 16 + '0';  // 秒的十位
    Time_Data[7] = data[0] % 16 + '0';  // 秒的个位

    // 将读取的日期数据格式化为字符串（例如：2023-01-03）
    Date[0] = '2';
    Date[1] = '0';
    Date[2] = data[6] / 16 + '0'; // 年的十位
    Date[3] = data[6] % 16 + '0'; // 年的个位
    Date[4] = '-';
    Date[5] = data[5] / 16 + '0'; // 月的十位
    Date[6] = data[5] % 16 + '0'; // 月的个位
    Date[7] = '-';
    Date[8] = data[4] / 16 + '0'; // 日的十位
    Date[9] = data[4] % 16 + '0'; // 日的个位

    // 星期数据（格式化为数字字符串，例如：1 表示星期一）
    Week[0] = data[3] % 16 + '0';

    I2C_Stop(); // 发送 I2C 停止信号
```

```
}

/**
 * 函数功能：向 DS1307 写入时间和日期数据
 * 描述：通过 I2C 接口将时间和日期数据写入 DS1307 寄存器，
 *       覆盖其当前时间。
 * 参数：
 *      data - 包含时间和日期数据的数组，长度为 7 字节。
 */
void DS1307_Write(uint8_t *data)
{
    I2C_Start(); // 发送 I2C 起始信号
    I2C_Send_Byte(0xD0); // 发送设备地址（写操作，地址为 0xD0）
    LPC_I2C->CONSET = (1 << 2); // 设置 ACK（AA=1）

    I2C_Send_Byte(0x00); // 设置 DS1307 内部指针到地址 0x00
    // 依次写入秒、分、时、星期、日、月、年
    I2C_Send_Byte(data[0]); // 秒
    I2C_Send_Byte(data[1]); // 分
    I2C_Send_Byte(data[2]); // 时
    I2C_Send_Byte(data[3]); // 星期
    I2C_Send_Byte(data[4]); // 日
    I2C_Send_Byte(data[5]); // 月
    LPC_I2C->CONCLR = (1 << 2); // 清除 ACK（AA=0）
    I2C_Send_Byte(data[6]); // 年

    I2C_Stop(); // 发送 I2C 停止信号
}

/**
 * 函数功能：向 DS1307 写入一个字节数据
 * 描述：通过 I2C 接口在指定地址写入一个字节数据。
 * 参数：
 *      WriteAddr - 要写入的寄存器地址。
 *      WriteData - 要写入的数据。
 */
void Ds1307_WriteByte(uint8_t WriteAddr, uint8_t WriteData)
{
    I2C_Start(); // 发送 I2C 起始信号
    I2C_Send_Byte(0xD0); // 发送设备地址（写操作，地址为 0xD0）
    I2C_Send_Byte(WriteAddr); // 指定写入的寄存器地址
    LPC_I2C->CONCLR = (1 << 2); // 清除 ACK（AA=0）
    I2C_Send_Byte(WriteData); // 写入数据
    I2C_Stop(); // 发送 I2C 停止信号
```

```
}

/**
 * 函数功能：从 DS1307 读取一个字节数据
 * 描述：通过 I2C 接口从 DS1307 的当前地址读取一个字节数据。
 * 返回值：
 *      uint8_t - 读取到的数据。
 */
uint8_t Ds1307_ReadByte(void)
{
    uint8_t RevData;

    I2C_Start(); // 发送 I2C 起始信号
    I2C_Send_Byte(0xD1); // 发送设备地址（读操作，地址为 0xD1）
    LPC_I2C->CONCLR = (1 << 2); // 清除 ACK（AA=0）
    RevData = I2C_Recieve_Byte(); // 读取数据
    I2C_Stop(); // 发送 I2C 停止信号

    return RevData; // 返回读取到的数据
}
/*************************************************************************
********/
/*OLED 字模库，宽 8 像素，高 16 像素*/
const uint8_t OLED_F8x16[][16]=
{
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//  0

    0x00,0x00,0x00,0xF8,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x33,0x30,0x00,0x00,0x00,//! 1

    0x00,0x10,0x0C,0x06,0x10,0x0C,0x06,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//" 2

    0x40,0xC0,0x78,0x40,0xC0,0x78,0x40,0x00,
    0x04,0x3F,0x04,0x04,0x3F,0x04,0x04,0x00,//# 3

    0x00,0x70,0x88,0xFC,0x08,0x30,0x00,0x00,
    0x00,0x18,0x20,0xFF,0x21,0x1E,0x00,0x00,//$ 4

    0xF0,0x08,0xF0,0x00,0xE0,0x18,0x00,0x00,
    0x00,0x21,0x1C,0x03,0x1E,0x21,0x1E,0x00,//% 5

    0x00,0xF0,0x08,0x88,0x70,0x00,0x00,0x00,
```

```
0x1E,0x21,0x23,0x24,0x19,0x27,0x21,0x10,//& 6

0x10,0x16,0x0E,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//' 7

0x00,0x00,0x00,0xE0,0x18,0x04,0x02,0x00,
0x00,0x00,0x00,0x07,0x18,0x20,0x40,0x00,//( 8

0x00,0x02,0x04,0x18,0xE0,0x00,0x00,0x00,
0x00,0x40,0x20,0x18,0x07,0x00,0x00,0x00,//) 9

0x40,0x40,0x80,0xF0,0x80,0x40,0x40,0x00,
0x02,0x02,0x01,0x0F,0x01,0x02,0x02,0x00,//* 10

0x00,0x00,0x00,0xF0,0x00,0x00,0x00,0x00,
0x01,0x01,0x01,0x1F,0x01,0x01,0x01,0x00,//+ 11

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x80,0xB0,0x70,0x00,0x00,0x00,0x00,0x00,//, 12

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x01,0x01,0x01,0x01,0x01,0x01,0x01,//- 13

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x30,0x30,0x00,0x00,0x00,0x00,0x00,//. 14

0x00,0x00,0x00,0x00,0x80,0x60,0x18,0x04,
0x00,0x60,0x18,0x06,0x01,0x00,0x00,0x00,/// 15

0x00,0xE0,0x10,0x08,0x08,0x10,0xE0,0x00,
0x00,0x0F,0x10,0x20,0x20,0x10,0x0F,0x00,//0 16

0x00,0x10,0x10,0xF8,0x00,0x00,0x00,0x00,
0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//1 17

0x00,0x70,0x08,0x08,0x08,0x88,0x70,0x00,
0x00,0x30,0x28,0x24,0x22,0x21,0x30,0x00,//2 18

0x00,0x30,0x08,0x88,0x88,0x48,0x30,0x00,
0x00,0x18,0x20,0x20,0x20,0x11,0x0E,0x00,//3 19

0x00,0x00,0xC0,0x20,0x10,0xF8,0x00,0x00,
0x00,0x07,0x04,0x24,0x24,0x3F,0x24,0x00,//4 20
```

```
0x00,0xF8,0x08,0x88,0x88,0x08,0x08,0x00,
0x00,0x19,0x21,0x20,0x20,0x11,0x0E,0x00,//5 21

0x00,0xE0,0x10,0x88,0x88,0x18,0x00,0x00,
0x00,0x0F,0x11,0x20,0x20,0x11,0x0E,0x00,//6 22

0x00,0x38,0x08,0x08,0xC8,0x38,0x08,0x00,
0x00,0x00,0x00,0x3F,0x00,0x00,0x00,0x00,//7 23

0x00,0x70,0x88,0x08,0x08,0x88,0x70,0x00,
0x00,0x1C,0x22,0x21,0x21,0x22,0x1C,0x00,//8 24

0x00,0xE0,0x10,0x08,0x08,0x10,0xE0,0x00,
0x00,0x00,0x31,0x22,0x22,0x11,0x0F,0x00,//9 25

0x00,0x00,0x00,0xC0,0xC0,0x00,0x00,0x00,
0x00,0x00,0x00,0x30,0x30,0x00,0x00,0x00,//: 26

0x00,0x00,0x00,0x80,0x00,0x00,0x00,0x00,
0x00,0x00,0x80,0x60,0x00,0x00,0x00,0x00,//; 27

0x00,0x00,0x80,0x40,0x20,0x10,0x08,0x00,
0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x00,//< 28

0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x00,
0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x00,//= 29

0x00,0x08,0x10,0x20,0x40,0x80,0x00,0x00,
0x00,0x20,0x10,0x08,0x04,0x02,0x01,0x00,//> 30

0x00,0x70,0x48,0x08,0x08,0x08,0xF0,0x00,
0x00,0x00,0x00,0x30,0x36,0x01,0x00,0x00,//? 31

0xC0,0x30,0xC8,0x28,0xE8,0x10,0xE0,0x00,
0x07,0x18,0x27,0x24,0x23,0x14,0x0B,0x00,//@ 32

0x00,0x00,0xC0,0x38,0xE0,0x00,0x00,0x00,
0x20,0x3C,0x23,0x02,0x02,0x27,0x38,0x20,//A 33

0x08,0xF8,0x88,0x88,0x88,0x70,0x00,0x00,
0x20,0x3F,0x20,0x20,0x20,0x11,0x0E,0x00,//B 34

0xC0,0x30,0x08,0x08,0x08,0x08,0x38,0x00,
0x07,0x18,0x20,0x20,0x20,0x10,0x08,0x00,//C 35
```

```
0x08,0xF8,0x08,0x08,0x08,0x10,0xE0,0x00,
0x20,0x3F,0x20,0x20,0x20,0x10,0x0F,0x00,//D 36

0x08,0xF8,0x88,0x88,0xE8,0x08,0x10,0x00,
0x20,0x3F,0x20,0x20,0x23,0x20,0x18,0x00,//E 37

0x08,0xF8,0x88,0x88,0xE8,0x08,0x10,0x00,
0x20,0x3F,0x20,0x00,0x03,0x00,0x00,0x00,//F 38

0xC0,0x30,0x08,0x08,0x08,0x38,0x00,0x00,
0x07,0x18,0x20,0x20,0x22,0x1E,0x02,0x00,//G 39

0x08,0xF8,0x08,0x00,0x00,0x08,0xF8,0x08,
0x20,0x3F,0x21,0x01,0x01,0x21,0x3F,0x20,//H 40

0x00,0x08,0x08,0xF8,0x08,0x08,0x00,0x00,
0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//I 41

0x00,0x00,0x08,0x08,0xF8,0x08,0x08,0x00,
0xC0,0x80,0x80,0x80,0x7F,0x00,0x00,0x00,//J 42

0x08,0xF8,0x88,0xC0,0x28,0x18,0x08,0x00,
0x20,0x3F,0x20,0x01,0x26,0x38,0x20,0x00,//K 43

0x08,0xF8,0x08,0x00,0x00,0x00,0x00,0x00,
0x20,0x3F,0x20,0x20,0x20,0x20,0x30,0x00,//L 44

0x08,0xF8,0xF8,0x00,0xF8,0xF8,0x08,0x00,
0x20,0x3F,0x00,0x3F,0x00,0x3F,0x20,0x00,//M 45

0x08,0xF8,0x30,0xC0,0x00,0x08,0xF8,0x08,
0x20,0x3F,0x20,0x00,0x07,0x18,0x3F,0x00,//N 46

0xE0,0x10,0x08,0x08,0x08,0x10,0xE0,0x00,
0x0F,0x10,0x20,0x20,0x20,0x10,0x0F,0x00,//O 47

0x08,0xF8,0x08,0x08,0x08,0x08,0xF0,0x00,
0x20,0x3F,0x21,0x01,0x01,0x01,0x00,0x00,//P 48

0xE0,0x10,0x08,0x08,0x08,0x10,0xE0,0x00,
0x0F,0x18,0x24,0x24,0x38,0x50,0x4F,0x00,//Q 49

0x08,0xF8,0x88,0x88,0x88,0x88,0x70,0x00,
```

```
0x20,0x3F,0x20,0x00,0x03,0x0C,0x30,0x20,//R 50

0x00,0x70,0x88,0x08,0x08,0x08,0x38,0x00,
0x00,0x38,0x20,0x21,0x21,0x22,0x1C,0x00,//S 51

0x18,0x08,0x08,0xF8,0x08,0x08,0x18,0x00,
0x00,0x00,0x20,0x3F,0x20,0x00,0x00,0x00,//T 52

0x08,0xF8,0x08,0x00,0x00,0x08,0xF8,0x08,
0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,//U 53

0x08,0x78,0x88,0x00,0x00,0xC8,0x38,0x08,
0x00,0x00,0x07,0x38,0x0E,0x01,0x00,0x00,//V 54

0xF8,0x08,0x00,0xF8,0x00,0x08,0xF8,0x00,
0x03,0x3C,0x07,0x00,0x07,0x3C,0x03,0x00,//W 55

0x08,0x18,0x68,0x80,0x80,0x68,0x18,0x08,
0x20,0x30,0x2C,0x03,0x03,0x2C,0x30,0x20,//X 56

0x08,0x38,0xC8,0x00,0xC8,0x38,0x08,0x00,
0x00,0x00,0x20,0x3F,0x20,0x00,0x00,0x00,//Y 57

0x10,0x08,0x08,0x08,0xC8,0x38,0x08,0x00,
0x20,0x38,0x26,0x21,0x20,0x20,0x18,0x00,//Z 58

0x00,0x00,0x00,0xFE,0x02,0x02,0x02,0x00,
0x00,0x00,0x00,0x7F,0x40,0x40,0x40,0x00,//[ 59

0x00,0x0C,0x30,0xC0,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x01,0x06,0x38,0xC0,0x00,//\ 60

0x00,0x02,0x02,0x02,0xFE,0x00,0x00,0x00,
0x00,0x40,0x40,0x40,0x7F,0x00,0x00,0x00,//] 61

0x00,0x00,0x04,0x02,0x02,0x02,0x04,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//^ 62

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,//_ 63

0x00,0x02,0x02,0x04,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//` 64
```

```
0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,
0x00,0x19,0x24,0x22,0x22,0x22,0x3F,0x20,//a 65

0x08,0xF8,0x00,0x80,0x80,0x00,0x00,0x00,
0x00,0x3F,0x11,0x20,0x20,0x11,0x0E,0x00,//b 66

0x00,0x00,0x00,0x80,0x80,0x80,0x00,0x00,
0x00,0x0E,0x11,0x20,0x20,0x20,0x11,0x00,//c 67

0x00,0x00,0x00,0x80,0x80,0x88,0xF8,0x00,
0x00,0x0E,0x11,0x20,0x20,0x10,0x3F,0x20,//d 68

0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,
0x00,0x1F,0x22,0x22,0x22,0x22,0x13,0x00,//e 69

0x00,0x80,0x80,0xF0,0x88,0x88,0x88,0x18,
0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//f 70

0x00,0x00,0x80,0x80,0x80,0x80,0x80,0x00,
0x00,0x6B,0x94,0x94,0x94,0x93,0x60,0x00,//g 71

0x08,0xF8,0x00,0x80,0x80,0x80,0x00,0x00,
0x20,0x3F,0x21,0x00,0x00,0x20,0x3F,0x20,//h 72

0x00,0x80,0x98,0x98,0x00,0x00,0x00,0x00,
0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//i 73

0x00,0x00,0x00,0x80,0x98,0x98,0x00,0x00,
0x00,0xC0,0x80,0x80,0x80,0x7F,0x00,0x00,//j 74

0x08,0xF8,0x00,0x00,0x80,0x80,0x80,0x00,
0x20,0x3F,0x24,0x02,0x2D,0x30,0x20,0x00,//k 75

0x00,0x08,0x08,0xF8,0x00,0x00,0x00,0x00,
0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,//l 76

0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x00,
0x20,0x3F,0x20,0x00,0x3F,0x20,0x00,0x3F,//m 77

0x80,0x80,0x00,0x80,0x80,0x80,0x00,0x00,
0x20,0x3F,0x21,0x00,0x00,0x20,0x3F,0x20,//n 78

0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,
0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,//o 79
```

```
0x80,0x80,0x00,0x80,0x80,0x00,0x00,0x00,
0x80,0xFF,0xA1,0x20,0x20,0x11,0x0E,0x00,//p 80

0x00,0x00,0x00,0x80,0x80,0x80,0x80,0x00,
0x00,0x0E,0x11,0x20,0x20,0xA0,0xFF,0x80,//q 81

0x80,0x80,0x80,0x00,0x80,0x80,0x80,0x00,
0x20,0x20,0x3F,0x21,0x20,0x00,0x01,0x00,//r 82

0x00,0x00,0x80,0x80,0x80,0x80,0x80,0x00,
0x00,0x33,0x24,0x24,0x24,0x24,0x19,0x00,//s 83

0x00,0x80,0x80,0xE0,0x80,0x80,0x00,0x00,
0x00,0x00,0x00,0x1F,0x20,0x20,0x00,0x00,//t 84

0x80,0x80,0x00,0x00,0x00,0x80,0x80,0x00,
0x00,0x1F,0x20,0x20,0x20,0x10,0x3F,0x20,//u 85

0x80,0x80,0x80,0x00,0x00,0x80,0x80,0x80,
0x00,0x01,0x0E,0x30,0x08,0x06,0x01,0x00,//v 86

0x80,0x80,0x00,0x80,0x00,0x80,0x80,0x80,
0x0F,0x30,0x0C,0x03,0x0C,0x30,0x0F,0x00,//w 87

0x00,0x80,0x80,0x00,0x80,0x80,0x80,0x00,
0x00,0x20,0x31,0x2E,0x0E,0x31,0x20,0x00,//x 88

0x80,0x80,0x80,0x00,0x00,0x80,0x80,0x80,
0x80,0x81,0x8E,0x70,0x18,0x06,0x01,0x00,//y 89

0x00,0x80,0x80,0x80,0x80,0x80,0x80,0x00,
0x00,0x21,0x30,0x2C,0x22,0x21,0x30,0x00,//z 90

0x00,0x00,0x00,0x00,0x80,0x7C,0x02,0x02,
0x00,0x00,0x00,0x00,0x00,0x3F,0x40,0x40,//{ 91

0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,//| 92

0x00,0x02,0x02,0x7C,0x80,0x00,0x00,0x00,
0x00,0x40,0x40,0x3F,0x00,0x00,0x00,0x00,//} 93

0x00,0x06,0x01,0x01,0x02,0x02,0x04,0x04,
```

```
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//~ 94
};

void OLED_WriteCommand(uint8_t Command)
{
    I2C_Start();
    I2C_Send_Byte(0x78);        //从机地址
    I2C_Send_Byte(0x00);        //写命令
    I2C_Send_Byte(Command);
    I2C_Stop();
}

void OLED_WriteData(uint8_t Data)
{
    I2C_Start();
    I2C_Send_Byte(0x78);        //从机地址
    I2C_Send_Byte(0x40);        //写数据
    I2C_Send_Byte(Data);
    I2C_Stop();
}

/**
  * @brief  OLED 设置光标位置
  * @param  Y 以左上角为原点，向下方向的坐标，范围：0~7
  * @param  X 以左上角为原点，向右方向的坐标，范围：0~127
  * @retval 无
  */
void OLED_SetCursor(uint8_t Y, uint8_t X)
{
    OLED_WriteCommand(0xB0 | Y);                    //设置 Y 位置
    OLED_WriteCommand(0x10 | ((X & 0xF0) >> 4));    //设置 X 位置高 4 位
    OLED_WriteCommand(0x00 | (X & 0x0F));           //设置 X 位置低 4 位
}

/**
  * @brief  OLED 清屏
  * @param  无
  * @retval 无
  */
void OLED_Clear(void)
{
    uint8_t i, j;
    for (j = 0; j < 8; j++)
    {
```

```
        OLED_SetCursor(j, 0);
        for(i = 0; i < 128; i++)
        {
            OLED_WriteData(0x00);
        }
    }
}

/**
  * @brief  OLED 显示一个字符
  * @param  Line 行位置，范围：1~4
  * @param  Column 列位置，范围：1~16
  * @param  Char 要显示的一个字符，范围：ASCII 可见字符
  * @retval 无
  */
void OLED_ShowChar(uint8_t Line, uint8_t Column, char Char)
{
    uint8_t i;
    OLED_SetCursor((Line - 1) * 2, (Column - 1) * 8);       //设置光标位置
在上半部分
    for (i = 0; i < 8; i++)
    {
        OLED_WriteData(OLED_F8x16[Char - ' '][i]);          //显示上半部分
内容
    }
    OLED_SetCursor((Line - 1) * 2 + 1, (Column - 1) * 8);   //设置光标位置
在下半部分
    for (i = 0; i < 8; i++)
    {
        OLED_WriteData(OLED_F8x16[Char - ' '][i + 8]);      //显示下半部分
内容
    }
}

/**
  * @brief  OLED 显示字符串
  * @param  Line 起始行位置，范围：1~4
  * @param  Column 起始列位置，范围：1~16
  * @param  String 要显示的字符串，范围：ASCII 可见字符
  * @retval 无
  */
void OLED_ShowString(uint8_t Line, uint8_t Column, char *String)
{
    uint8_t i;
```

```
    for (i = 0; String[i] != '\0'; i++)
    {
        OLED_ShowChar(Line, Column + i, String[i]);
    }
}

/**
  * @brief  OLED 次方函数
  * @retval 返回值等于 X 的 Y 次方
  */
uint32_t OLED_Pow(uint32_t X, uint32_t Y)
{
    uint32_t Result = 1;
    while (Y--)
    {
        Result *= X;
    }
    return Result;
}

/**
  * @brief  OLED 显示数字（十进制，正数）
  * @param  Line 起始行位置，范围：1~4
  * @param  Column 起始列位置，范围：1~16
  * @param  Number 要显示的数字，范围：0~4294967295
  * @param  Length 要显示数字的长度，范围：1~10
  * @retval 无
  */
void OLED_ShowNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length)
{
    uint8_t i;
    for (i = 0; i < Length; i++)
    {
        OLED_ShowChar(Line, Column + i, Number / OLED_Pow(10, Length - i -
1) % 10 + '0');
    }
}

/**
  * @brief  OLED 显示数字（十进制，带符号数）
  * @param  Line 起始行位置，范围：1~4
  * @param  Column 起始列位置，范围：1~16
  * @param  Number 要显示的数字，范围：-2147483648~2147483647
```

```
  * @param  Length 要显示数字的长度，范围：1~10
  * @retval 无
  */
void OLED_ShowSignedNum(uint8_t Line, uint8_t Column, int32_t Number,
uint8_t Length)
{
    uint8_t i;
    uint32_t Number1;
    if (Number >= 0)
    {
        OLED_ShowChar(Line, Column, '+');
        Number1 = Number;
    }
    else
    {
        OLED_ShowChar(Line, Column, '-');
        Number1 = -Number;
    }
    for (i = 0; i < Length; i++)
    {
        OLED_ShowChar(Line, Column + i + 1, Number1 / OLED_Pow(10, Length
- i - 1) % 10 + '0');
    }
}

/**
  * @brief  OLED 显示数字（十六进制，正数）
  * @param  Line 起始行位置，范围：1~4
  * @param  Column 起始列位置，范围：1~16
  * @param  Number 要显示的数字，范围：0~0xFFFFFFFF
  * @param  Length 要显示数字的长度，范围：1~8
  * @retval 无
  */
void OLED_ShowHexNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length)
{
    uint8_t i, SingleNumber;
    for (i = 0; i < Length; i++)
    {
        SingleNumber = Number / OLED_Pow(16, Length - i - 1) % 16;
        if (SingleNumber < 10)
        {
            OLED_ShowChar(Line, Column + i, SingleNumber + '0');
        }
```

```
        else
        {
            OLED_ShowChar(Line, Column + i, SingleNumber - 10 + 'A');
        }
    }
}

/**
  * @brief  OLED 显示数字（二进制，正数）
  * @param  Line 起始行位置，范围：1~4
  * @param  Column 起始列位置，范围：1~16
  * @param  Number 要显示的数字，范围：0~1111 1111 1111 1111
  * @param  Length 要显示数字的长度，范围：1~16
  * @retval 无
  */
void OLED_ShowBinNum(uint8_t Line, uint8_t Column, uint32_t Number, uint8_t
Length)
{
    uint8_t i;
    for (i = 0; i < Length; i++)
    {
        OLED_ShowChar(Line, Column + i, Number / OLED_Pow(2, Length - i -
1) % 2 + '0');
    }
}

void OLED_Init(void)
{
    uint32_t i, j;

    for (i = 0; i < 1000; i++)          //上电延时
    {
        for (j = 0; j < 1000; j++);
    }

    I2C_Init(1);            //端口初始化

    OLED_WriteCommand(0xAE);    //关闭显示

    OLED_WriteCommand(0xD5);    //设置显示时钟分频比/振荡器频率
    OLED_WriteCommand(0x80);

    OLED_WriteCommand(0xA8);    //设置多路复用率
    OLED_WriteCommand(0x3F);
```

```
    OLED_WriteCommand(0xD3);    //设置显示偏移
    OLED_WriteCommand(0x00);

    OLED_WriteCommand(0x40);    //设置显示开始行

    OLED_WriteCommand(0xA1);    //设置左右方向，0xA1 正常 0xA0 左右反置

    OLED_WriteCommand(0xC8);    //设置上下方向，0xC8 正常 0xC0 上下反置

    OLED_WriteCommand(0xDA);    //设置 COM 引脚硬件配置
    OLED_WriteCommand(0x12);

    OLED_WriteCommand(0x81);    //设置对比度控制
    OLED_WriteCommand(0xCF);

    OLED_WriteCommand(0xD9);    //设置预充电周期
    OLED_WriteCommand(0xF1);

    OLED_WriteCommand(0xDB);    //设置 VCOMH 取消选择级别
    OLED_WriteCommand(0x30);

    OLED_WriteCommand(0xA4);    //设置整个显示打开/关闭

    OLED_WriteCommand(0xA6);    //设置正常/倒转显示

    OLED_WriteCommand(0x8D);    //设置充电泵
    OLED_WriteCommand(0x14);

    OLED_WriteCommand(0xAF);    //开启显示

    OLED_Clear();               //OLED 清屏
}
/**********************************************************************
********/
void RGB_LED_Init(void)
{
   LPC_SYSCON->SYSAHBCLKCTRL |= (1UL << 6); /* GPIO 时钟使能 */
   LPC_GPIO1->DIR |= (1UL << 9); /*将 GPIO1_9 作为输出口*/
   LPC_GPIO2->DIR |= (1UL << 8); /*配置 GPIO2_8 为输出模式 */  // RGB_Green
   LPC_GPIO2->DIR |= (1UL << 9); /*配置 GPIO2_8 为输出模式 */  // RGB_Red
   LPC_GPIO2->DIR |= (1UL << 10); /*配置 GPIO2_10 为输出模式 */ // RGB_Blue
}
```

```c
void LED_Toggle(void)
{
    LPC_GPIO1->DATA ^= (1UL <<9);/* LED 闪烁切换 */
    T16B0_delay_ms(50);
}

/* 翻转 RGB_Blue 蓝灯 */
void RGB_Blue_Toggle(void)
{
    LPC_GPIO2->DATA  ^=  (1UL <<10); /* LED 闪烁切换 */
    T16B0_delay_ms(50);
}
/* 翻转 RGB_Green 绿灯 */
void RGB_Green_Toggle(void)
{
    LPC_GPIO2->DATA ^= (1UL <<8);/* LED 闪烁切换 */
    T16B0_delay_ms(50);
}

/* 翻转 RGB_Green 绿灯 */
void RGB_Red_Toggle(void)
{
    LPC_GPIO2->DATA ^= (1UL <<9);/* LED 闪烁切换 */
    T16B0_delay_ms(50);
}
/*****************************************************************************
********/
```