

**课 程 报 告**

年 级 专 业： 2023电子信息

学 生 姓 名： 雷灿曦

学 号： 20233006006

课 程 名 称： 嵌入式系统设计

任 课 教 师： 张永辉 徐博

分 数：

海南大学·信息与通信工程学院

School of Information and Communication Engineering, Hainan University

**具有声光报警功能的温度记录仪**

雷灿曦

电子信息专业，信息与通信工程学院，海南大学，海口，570228

**摘要：本文基于LPC11xx微控制器（LP1114）设计并实现了一个集温度监测、数据存储、实时时钟和用户交互于一体的嵌入式系统。系统通过LM75温度传感器采集环境温度，利用外部Flash存储器实现温度数据的持久化存储，并集成DS1307实时时钟为每条温度记录添加时间戳。系统支持通过UART接口接收用户指令，包括开始/停止记录、显示历史数据、擦除存储器和设置时间等功能。采用魔数标记机制确保Flash数据的有效性，通过定时器中断实现实时数据采集和命令响应。实验结果表明，该系统能够稳定可靠地完成温度监测任务，具备良好的实用性和可扩展性。**

**关键词：LPC11xx微控制器；温度监测；Flash存储；实时时钟；UART通信；嵌入式系统；数据持久化；魔数验证**

**1.引言**

温度监测在工业自动化、环境监测等领域具有重要应用价值。本文设计了一种基于LPC11xx微控制器的智能温度监测系统，创新性地整合了Flash存储管理、实时时钟和UART通信功能。系统通过魔数验证机制确保数据完整性，利用DS1307芯片为温度数据添加时间戳，建立了完整的命令响应体系。与以往课设或设计相比，本系统在数据可靠性、时间精度和用户体验方面具有显著优势，为嵌入式温度监测应用提供了一套实用的解决方案。

**2.需求分析**

##### **2.1 项目背景及目标**

本项目旨在开发一款具有声光报警功能的温度记录仪。该系统能够实时采集环境温度，并按固定时间间隔存储数据，同时具备数据通信与超限报警能力，满足课程设计要求。以ARM Cortex-M0内核的LP1114微控制器为核心，集成温度采集、大容量数据存储、串口通信和报警指示功能的嵌入式系统。该系统应满足课程设计的功能需求，并具备高可靠性、低功耗和易用性。

**2.2 系统总体需求**

|  |  |
| --- | --- |
| 需求分析 | 描述 |
| 功能性需求： | 系统必须具备温度采集、数据存储、数据通信和超温报警四大核心功能。 |
| 可靠性需求 | 至少保存1个小时的温度数据；可掉电保存的存储单元 |
| 实时性需求 | 每1秒记录一次温度,每记录温度一次即通过UART发送至PC进行数据读取。 |
| 可用性需求： | 温度达到设定阈值时能够通过LED或蜂鸣器进行报警；GPIO控制LED闪烁和蜂鸣器 |
| 电路设计需求： | 电路板尺寸：8cm × 3cm以内。 |

**2.3. 功能需求分析**

### 功能需求分析

| 功能模块 | 需求描述 | 实现方案 |
| --- | --- | --- |
| FRQ-001: 温度采集 | 系统应能实时、准确地获取环境温度值。检测温度：范围：-55℃~+125℃，精度：±1.5℃。 | 通过LP114的I²C接口与LM75BD温度传感器通信，读取其内部温度寄存器。 |
| FRQ-002: 定时记录 | 系统必须以1s固定周期，不间断地执行一次温度记录（包含采集与存储）。 | 利用LP114的硬件定时器产生精确的1秒中断，在中断服务程序中触发记录任务。 |
| FRQ-003: 数据存储 | 系统至少能存储1小时（3600个数据点）的温度数据，并保证掉电后数据不丢失。 | 将采集到的温度数据（包括温度值和时间戳）打包后，通过SPI接口写入XT25F02E Serial NOR Flash芯片。 |
| FRQ-004: 数据通信 | 每次记录温度后，需立即将当前温度数据通过串口发送至上位机（PC）。 | 利用LP114的UART外设，配置好波特率、数据位、停止位和校验位，在每次记录完成后调用串口发送函数。 |
| FRQ-005: 超温报警 | 当采集到的温度值超过用户设定的阈值时，系统应触发声光报警。 | 通过LP114的GPIO引脚控制一个LED灯进入亮状态。 |

**2.4. 硬件资源与接口需求分析**

### 硬件资源与接口需求分析

| 硬件组件 | 型号 | 接口 | 需求分析 |
| --- | --- | --- | --- |
| 微控制器 | LP1114 | - | 作为系统核心，负责调度所有任务，处理数据，控制外设。 |
| 温度传感器 | LM75BD | I²C | 1. 拥有I²C外设作为主设备。2. 需知悉LM75BD的I²C设备地址。3. 需编写I²C读写驱动程序，按照LM75BD的时序读取温度数据。 |
| 存储芯片 | XT25F02E | SPI | 1. MCU需配置一个SPI外设作为主设备。  2. 需根据芯片手册编写SPI Flash的读写、擦除驱动。3. 需设计存储数据结构。 |
| 通信接口 | UART to USB | UART | 1. MCU需配置一个UART外设。  2. 需确定通信参数（如115200, 8-N-1）。 |
| 报警装置 | LED | GPIO | 1. GPIO引脚，设置为推挽输出模式。  2. 引脚用于控制LED。 |
| USB  接口 | USB | USB | 1. USB供电。 2. USB转串口。 |

**3.方案设计**

本项目旨在设计并实现一个具备数据记录、远程通信及超限报警功能的嵌入式温度记录仪。系统以LPC1144微控制器为核心，通过模块化设计整合传感器采集、非易失性存储、人机交互及报警指示等单元，构建一个稳定、可靠且功能完备的数据记录系统。

**3.1硬件方案设计**

结合系统温度采集、数据存储、通信与报警的核心需求，基于LPC11xx微控制器完成硬件选型与接口匹配，具体设备选型及接口规划如下表所示：

| 外设模块 | 使用的外设/协议 | MCU引脚 | 连接目标 | 引脚功能描述 |
| --- | --- | --- | --- | --- |
| 温度传感器​ | I2C总线​ | PIO0\_4​ | LM75的SDA | I2C数据线 |
|  |  | PIO0\_5​ | LM75的SCL | I2C时钟线 |
| 实时时钟（RTC）​ | I2C总线（与LM75共享）​ | PIO0\_4​ | DS1307的SDA | I2C数据线（与LM75共用） |
|  |  | PIO0\_5​ | DS1307的SCL | I2C时钟线（与LM75共用） |
| 外部存储器​ | SPI总线​ | PIO2\_1​ | Flash的SCK | SPI串行时钟 |
|  |  | PIO2\_2​ | Flash的MISO | SPI主设备输入从设备输出 |
|  |  | PIO2\_3​ | Flash的MOSI | SPI主设备输出从设备输入 |
|  | GPIO​ | PIO2\_0​ | Flash的CS | Flash片选信号 |
| 串口通信​ | UART​ | PIO1\_6​ | USB转串口芯片的RXD | UART数据接收 |
|  |  | PIO1\_7​ | USB转串口芯片的TXD | UART数据发送 |
| 报警指示​ | GPIO​ | PIO1\_9​ | LED阳极（通过限流电阻） | 报警指示灯控制 |

**3.2系统总体软件架构设计**

**3.2.1系统设计思路**

在构思这套温度采集记录报警系统时，核心遵循分层模块化的嵌入式设计思路，以1秒为核心定时周期，串联温度采集、存储、传输、报警四大核心功能，同时兼顾掉电保存与实时性要求。

1.温度采集模块设计

首先需要温度采集，选用I2C接口的LM7BD数字温度传感器，硬件层面完成传感器与微控制器的引脚连接后，软件上编写传感器驱动函数实现原始温度数据的读取，针对传感器输出的数字量（如LM7BD的16位补码数据），设计专属的转换函数，将原始数据按传感器手册的换算规则转化为直观的摄氏度数值。

2. 数据存储与管理方案设计

接着针对“1小时掉电保存”的需求做存储设计，1秒记录一次则1小时需存储3600条数据，选用SPI接口的Flash ，先定义轻量化的温度记录数据结构，包含采集序号（或简易时间戳）与转换后的温度值，计算单条数据的存储字节数并规划存储地址区间，避免数据覆盖；软件上封装存储驱动的读写擦除接口，每次采集到温度后，调用写入函数将数据按地址顺序存入存储单元，同时引入魔数机制，并在系统初始化时检测存储单元的有效性，确保掉电后重启能读取历史数据。

3. 通信与报警联动设计

最后要实现串口通信和报警机制，需要整合UART传输与GPIO报警的逻辑，将UART通信的波特率配置为115200（通用稳定速率），在每次完成温度采集与存储后，立即调用UART发送函数，将温度数据按固定格式（如“采集点：XXX，温度：XX.XX℃\r\n”）封装为字符串发送至PC，保证数据实时回传；报警功能则通过宏定义设置温度阈值，每次采集转换后将温度值与阈值对比，若超过阈值，立即通过GPIO口输出高低电平或脉冲信号，控制LED亮，同时在UART中同步发送报警提示信息，阈值采用宏定义的方式，便于后续灵活修改。

4.系统整体调度设计

为保证1秒周期的精确性与任务执行的实时性，系统的运行调度基于硬件定时器中断。具体而言，将一个高优先级定时中断的周期设置为1秒，在该中断服务程序中，按严格顺序依次执行温度采集与转换、数据存储、串口发送及报警判断这一核心任务链。此种中断驱动模式保证了关键任务的准时触发。而对于全片擦除Flash等非实时性且耗时的操作，则安排在主循环中处理，通过标志位进行触发，从而避免中断被长时间阻塞。

为了管理如此多功能块，将软件分为三层：

硬件驱动层：负责底层SPI、I2C、UART、GPIO等外设的直接操作。

功能模块层：基于驱动层，实现温度读取、时间管理、Flash存储管理等具体的业务功能。

应用逻辑层：作为系统中枢，负责业务规则的调度，如定时任务序列、命令解析和状态机管理。

这种结构确保了各模块间的低耦合性与高内聚性，便于开发、测试与维护。

**3.2.2系统设计框图**

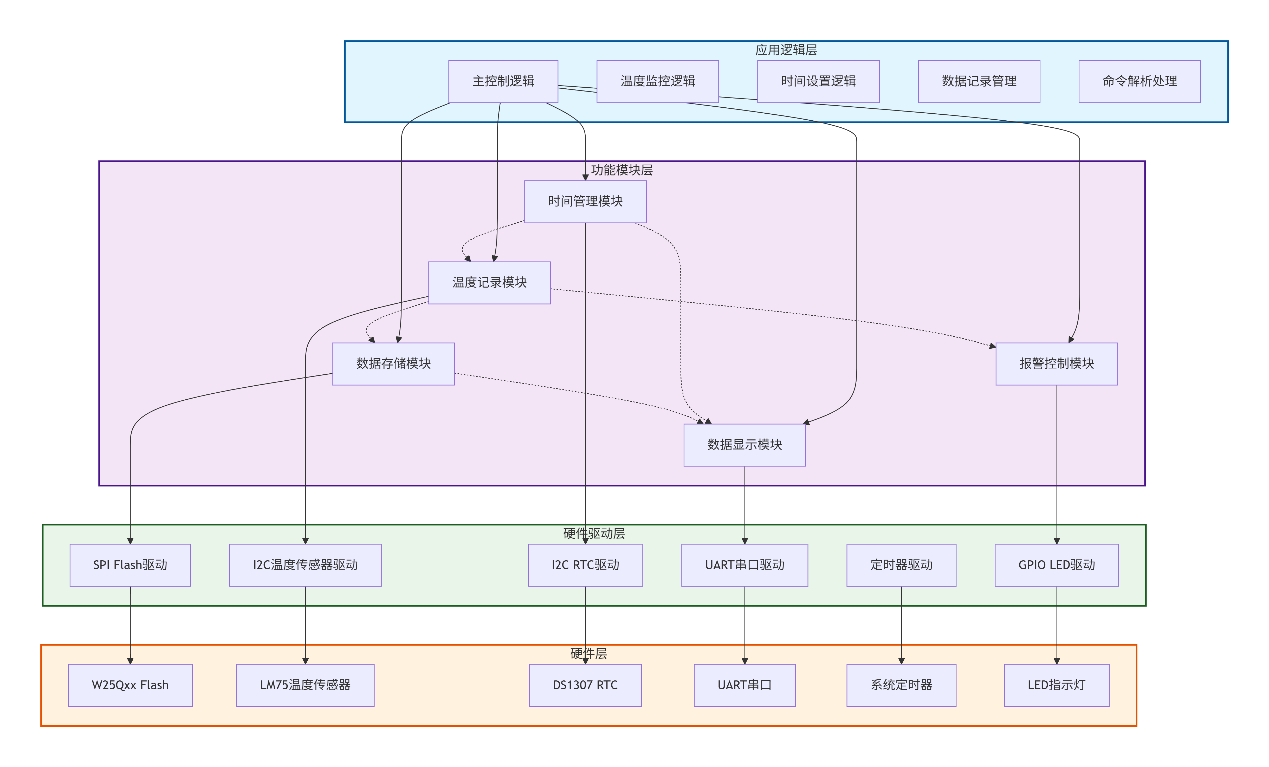


图1 系统设计框图

**3.2.4系统运行逻辑**

1. 初始化阶段

系统上电后，按序完成时钟、串口、总线、外设等硬件的初始化；同时校验Flash数据有效性，读取历史记录信息或完成Flash初始化，为系统运行做好准备。

2. 运行阶段

定时器按固定周期触发中断，依次执行时间读取、温度采集、串口数据显示、温度报警检测及串口命令解析等核心流程；当系统启动记录功能后，按周期将温度与时间数据写入Flash并更新记录信息；若Flash空间不足或接收到擦除指令，系统暂停记录并在主循环中执行Flash擦除，完成后重置记录参数。

3. 交互逻辑

用户通过串口命令实现系统记录启停、数据查看、Flash擦除等状态控制；系统支持自定义时间设置，通过指令解析更新RTC时钟，实现人机交互的灵活控制。

**3.3.程序设计**

**3.3.1关键参数以及函数变量**

// 引脚定义

#define FLASH\_CS\_HIGH() LPC\_GPIO2->DATA |= (1<<0)

#define FLASH\_CS\_LOW() LPC\_GPIO2->DATA &= ~(1<<0) // 直接寄存器操作控制Flash片选，SPI通信的起始和结束信号，时序要求严格。

// Flash指令

#define FLASH\_WriteEnable 0x06 // 关键：在进行写/擦除操作前必须发送此指令，是硬件安全流程，容易遗漏。

#define FLASH\_PageProgram 0x02

#define FLASH\_ChipErase 0xC7

// 全局变量

uint32\_t flash\_address = 0; // 当前Flash写入地址，与record\_count共同实现存储空间的顺序管理。

uint8\_t recording\_enabled = 0; // 系统核心状态标志，控制是否进行温度记录。

uint16\_t record\_count = 0;

uint8\_t flash\_erase\_requested = 0; // 关键异步操作标志。在中断中置位，在主循环中处理，避免阻塞实时任务。

uint32\_t erase\_start\_address = 0;

// 温度报警阈值

#define TEMP\_THRESHOLD\_RAW 0x1E00 // 使用原始数据进行比较，避免在中断服务程序中进行浮点运算，提高效率。

// 魔数标记

#define FLASH\_MAGIC\_NUMBER 0x544D5020 // 用于标记Flash数据有效性，实现系统"断点续存"功能。

#define DATA\_START\_ADDRESS 0x00000A // 用户数据区起始地址，前部空间用于存储元数据（魔数、地址、计数）。

#define TEMP\_RECORD\_SIZE 5 // 定义单条记录结构大小（2字节温度+3字节时间），是存储地址计算的基础。

// DS1307时间相关全局变量

uint8\_t current\_seconds, current\_minutes, current\_hours, current\_date, current\_month, current\_year; // 存储从RTC读取的当前时间，作为数据记录的时间戳来源。

// 时间设置相关变量

uint8\_t time\_set\_buffer[20]; // 用于时间设置模式的输入缓冲区，实现多字节输入处理。

uint8\_t time\_set\_index = 0;

uint8\_t time\_set\_mode = 0; // 时间设置模式的状态标志，与正常命令模式切换，构成一个简单状态机。

// 函数内部关键临时变量说明：

// SaveTemperatureData函数:

// - uint8\_t temp\_data[TEMP\_RECORD\_SIZE] // 临时数据包，体现记录结构(2字节温度+3字节时间)

//

// Flash\_ReadRecordInfo/Flash\_SaveRecordInfo函数:

// - uint8\_t info\_data[INFO\_DATA\_SIZE] //元数据临时存储(魔数+地址+计数)

//

// Process\_Time\_Input函数:

// - uint8\_t hours, minutes... // 时间参数临时解析变量，用于字符串到数值的转换校验

//

// DisplayAllRecords函数:

// - uint8\_t record\_hours, record\_minutes, record\_seconds // 从记录包中提取的时间戳临时变量

// - int16\_t raw\_temp //临时存储从Flash组合的原始温度数据

//

// SPI\_ExchangeByte函数:

// - volatile uint8\_t clear // 用于清空SPI FIFO的临时变量，volatile防止优化

**3.3.2函数设计**

## 1. 硬件驱动层设计

该层封装了所有最底层的寄存器配置与操作，旨在为上两层提供稳定、透明的硬件服务。

### 微控制器核心驱动

| 函数名称 | 功能描述 |
| --- | --- |
| SystemClock\_Config | 配置系统内核时钟，为整个系统提供稳定的运行基础 |
| LED\_Init/On/Off | 实现对LED指示灯GPIO的初始化和开关控制，是报警指示的执行基础 |
| Timer\_Init | 配置32位定时器，使其产生精确的1秒周期性中断，为系统提供心跳信号 |

### 通信接口驱动

| 函数名称 | 功能描述 |
| --- | --- |
| SPI\_Init, SPI\_ExchangeByte | 初始化SPI控制器并实现基本的字节收发功能，为Flash存储提供底层通信支持 |
| I2C\_Init, I2C\_Start/Stop, I2C\_SendByte/ReceiveByte, I2C\_Ack/NAck | 初始化I2C控制器并实现完整的协议时序，用于与LM75温度传感器和DS1307实时时钟通信 |
| UART\_Init, UART\_SendByte | 初始化UART控制器并实现单字节发送功能，构建了与PC通信的基础 |

### 外部存储器驱动

| 函数名称 | 功能描述 |
| --- | --- |
| Flash\_Init, Flash\_WriteEnable, Flash\_WaitBusy, Flash\_WritePage, Flash\_EraseChip | 在SPI驱动之上，实现针对具体Flash芯片的指令集操作（如写使能、页编程、芯片擦除），完成了对存储介质的物理读写控制 |

## 2. 功能模块层设计

本层将底层驱动组合成面向具体业务的功能模块，隐藏硬件细节，提供简洁接口。

### 温度传感模块

| 函数名称 | 功能描述 |
| --- | --- |
| ReadRawTemperature | 通过I2C驱动读取LM75传感器的原始16位温度数据 |
| ConvertToCelsius | 提供将原始数据转换为实际摄氏度值的算法，包含正负温度处理 |

### 实时时钟模块

| 函数名称 | 功能描述 |
| --- | --- |
| DS1307\_Init/Init\_With\_Time | 通过I2C驱动初始化或设置DS1307时钟芯片的时间 |
| Read\_DS1307\_Time | 读取当前时间并转换为十进制数，存储于全局变量中 |
| BCD\_to\_Decimal, Decimal\_to\_BCD | 提供时间数据格式的转换工具函数 |

### 数据存储管理模块

这是系统的核心模块之一，实现了Flash存储的逻辑管理。

| 函数名称 | 功能描述 |
| --- | --- |
| Flash\_IsValidData | 通过校验预设的魔数（Magic Number），判断Flash中是否存在有效的历史数据 |
| Flash\_SaveRecordInfo / Flash\_ReadRecordInfo | 在Flash固定位置保存和加载记录指针（flash\_address）和记录总数（record\_count），实现掉电后记录的续存 |
| SaveTemperatureData | 将16位温度数据和3字节时间戳打包成5字节的记录，并写入Flash，同时更新记录信息。此函数还实现了存储空间监控，在即将写满时触发擦除请求 |
| DisplayAllRecords | 读取Flash中的所有历史记录，进行格式化后通过串口输出，实现数据回溯功能 |

### 人机交互模块

| 函数名称 | 功能描述 |
| --- | --- |
| UART\_SendString | 基于单字节发送功能，实现字符串输出，方便信息展示 |
| Display\_Current\_Time | 格式化输出当前时间日期 |
| Start\_Time\_Set\_Mode, Process\_Time\_Input | 实现了一个简单的交互式命令行界面，用于通过串口设置RTC时间 |

## 3. 应用逻辑层设计

此层是系统的"大脑"，它不关心具体硬件如何工作，只负责根据业务规则调度各项功能。

### 系统状态机管理

通过全局变量（如recording\_enabled, flash\_erase\_requested, time\_set\_mode）定义和管理系统的不同工作状态（如空闲、记录、擦除、时间设置等），控制流程走向。

### 定时中断服务程序

这是核心调度逻辑。在1秒定时中断中，依次执行以下关键任务：

1. **时间同步**：调用Read\_DS1307\_Time获取最新时间
2. **温度采集与上报**：调用ReadRawTemperature和ConvertToCelsius获取温度，并立即格式化后通过串口发送至PC，满足实时传输需求
3. **报警判断与执行**：将温度原始值与阈值比较，若超限则调用LED\_On进行视觉报警
4. **条件记录**：检查记录使能标志，若允许则调用SaveTemperatureData保存带时间戳的数据
5. **命令响应**：检查串口接收缓冲区，解析用户命令（s, t, r, e, c）并改变系统状态

### 主循环后台任务

处理非实时或耗时操作：

| 任务类型 | 功能描述 |
| --- | --- |
| 命令处理 | 对在中断中接收到的命令进行进一步处理，如调用DisplayAllRecords显示数据 |
| 存储维护 | 当检测到flash\_erase\_requested标志时，执行Flash\_EraseChip和重新初始化的耗时操作，确保中断服务程序的及时响应 |
| 低功耗管理 | 在空闲时执行\_\_WFI()指令，使CPU进入低功耗等待模式 |

**3.4.关键函数流程图**

程序运行逻辑流程图

|  |
| --- |
| 程序运行逻辑流程图（见下一页） |
|  |

功能模块层流程图

|  |  |
| --- | --- |
| 数据存储管理(SaveTemperatureData) | 数据显示模块 (DisplayAllRecords) |
|  |  |

|  |
| --- |
| 时间设置模块 (Process\_Time\_Input) |

用户逻辑层

|  |  |
| --- | --- |
| 系统初始化与状态恢复流程 | 定时中断服务流程 |
|  |  |

|  |
| --- |
| 用户命令解析与状态控制流程 |
|  |

|  |
| --- |
| 存储空间管理流程 |
|  |

**4. 软件实现与核心代码分析**

前文已对系统各功能模块的函数及其用途进行了梳理。本节将在此基础上，深入分析几个关键的业务流程，展示这些函数是如何被调用、协同工作以实现系统核心需求的。重点在于阐述代码的执行流程和组织逻辑，而非罗列全部代码。

**4.1 主函数：系统启动与主循环调度**

主函数 main()是系统的总入口，其逻辑清晰地体现了分层初始化、状态恢复和后台任务调度的设计思想。程序运行逻辑如下：

**硬件与模块初始化**：代码严格遵循自底向上的初始化顺序，依次配置系统时钟、GPIO、UART、SPI Flash、I2C总线、LED以及DS1307实时时钟。

**系统状态恢复**：这是满足**掉电保存**需求的关键逻辑。通过 Flash\_IsValidData()函数判断Flash中是否存在有效历史数据。若有效，则调用 Flash\_ReadRecordInfo读取之前保存的记录指针和计数，实现“断点续传”。若无效，则执行 Flash\_EraseChip并初始化存储区，从头开始记录。

**启动核心引擎**：调用 Timer\_Init()启动定时器，开启1秒定时中断，系统的“心跳”开始跳动。

**主循环调度**：while(1)循环作为后台任务调度器，主要职责是检测并处理 flash\_erase\_requested标志。当该标志被置位时，执行耗时的全片擦除操作。完成后，通过 \_\_WFI()指令使CPU进入低功耗等待模式，直到被下一次定时中断唤醒。

int main(void) {

// 1. 硬件层初始化

SystemClock\_Config();

UART\_Init();

Flash\_Init();

I2C\_Init();

LED\_Init();

DS1307\_Init();

// 2. 功能层调用：读取初始时间

Read\_DS1307\_Time();

// 3. 应用逻辑层：系统状态恢复与用户提示

UART\_SendString("\r\nTemperature Monitor with RTC Time Display\r\n");

if (Flash\_IsValidData()) { // 关键决策点

Flash\_ReadRecordInfo(&flash\_address, &record\_count); // 恢复现场

UART\_SendString("Valid temperature data found, resuming...\r\n");

}

else {

UART\_SendString("No valid temperature data found, initializing...\r\n");

Flash\_EraseChip(); // 初始化存储区

flash\_address = DATA\_START\_ADDRESS;

record\_count = 0;

Flash\_SaveRecordInfo(flash\_address, record\_count); // 保存初始状态

}

// 4. 启动核心定时中断

Timer\_Init();

// 5. 应用逻辑层：主循环（后台任务调度器）

while (1) {

if (flash\_erase\_requested) { // 处理异步擦除请求

UART\_SendString("Performing flash erase...\r\n");

Flash\_EraseChip();

... // 复位系统状态

flash\_erase\_requested = 0; // 清除标志

}

\_\_WFI(); // 进入低功耗模式，等待中断唤醒

}

}

主函数将实时性要求高的任务（如数据采集）交由中断处理，而将非实时性**任务**（如Flash擦除）放在主循环，通过状态标志进行通信，合理分配了CPU资源。

**4.2 中断服务程序：1秒任务链的执行**

定时中断服务程序 TIMER32\_0\_IRQHandler()是系统的核心，以精确的1秒为周期，串联起温度采集、存储、传输、报警四大功能。 核心任务链如下：

// 1. 数据采集与实时上报

Read\_DS1307\_Time(); // 从DS1307读取当前时间

int16\_t raw\_temp = ReadRawTemperature(); // 从LM75读取原始温度值

// ... 将时间和温度值格式化后，立即通过 UART\_SendString 发送至PC

// 2. 报警判断与执行

if (raw\_temp >= TEMP\_THRESHOLD\_RAW) {

LED\_On(); // 硬件报警：点亮LED

UART\_SendString(" [ALARM!]"); // 软件报警：串口提示

}

else {

LED\_Off();

}

// 3. 条件存储

if (recording\_enabled && !flash\_erase\_requested) {

SaveTemperatureData(raw\_temp); // 保存带时间戳的数据

}

// 4. 用户命令响应

// 检查串口接收缓冲区，解析's','t'等命令，改变系统状态（如 recording\_enabled）

为了保证程序设计可靠，在中断函数有以下设计：

**固定序列**：任务链顺序固定，确保了“每记录一次即发送一次”的实时性。

**高效报警**：直接比较原始数据 TEMP\_THRESHOLD\_RAW，避免在中断内进行浮点数计算。

**状态驱动**：存储操作由全局标志 recording\_enabled控制，命令解析可以异步改变此状态，实现灵活控制。

**4.3 关键算法函数示例**

**A. 数据存储函数 SaveTemperatureData** 此函数完整展示了带时标数据的打包、存储和存储空间管理的逻辑。

// 数据打包：将2字节温度数据和3字节时间戳组合成5字节的记录包

temp\_data[0] = (raw\_temp >> 8) & 0xFF; // 温度高字节

temp\_data[1] = raw\_temp & 0xFF;

temp\_data[2] = current\_hours; // 时间戳：时

temp\_data[3] = current\_minutes; // 分

temp\_data[4] = current\_seconds; // 秒

Flash\_WritePage(temp\_data, flash\_address, TEMP\_RECORD\_SIZE); // 写入Flash

// ... 更新地址和计数，并保存元数据

// 存储空间监控：当地址接近Flash末尾时，置位擦除请求标志

if (flash\_address >= (0x20000 - 100 \* TEMP\_RECORD\_SIZE)) {

flash\_erase\_requested = 1;

recording\_enabled = 0;

}

**B. 数据回溯函数 DisplayAllRecords** 此函数通过循环读取，展示了如何将Flash中的二进制数据还原为用户可读的信息。

for (uint16\_t i = 0; i < record\_count; i++) {

Flash\_ReadData(temp\_data, address, TEMP\_RECORD\_SIZE); // 读取一条记录

// 解析数据并格式化输出

int16\_t raw\_temp = (temp\_data[0] << 8) | temp\_data[1];

float temp\_c = ConvertToCelsius(raw\_temp);

// ... 使用UART\_SendString输出记录序号、温度值、时间等信息

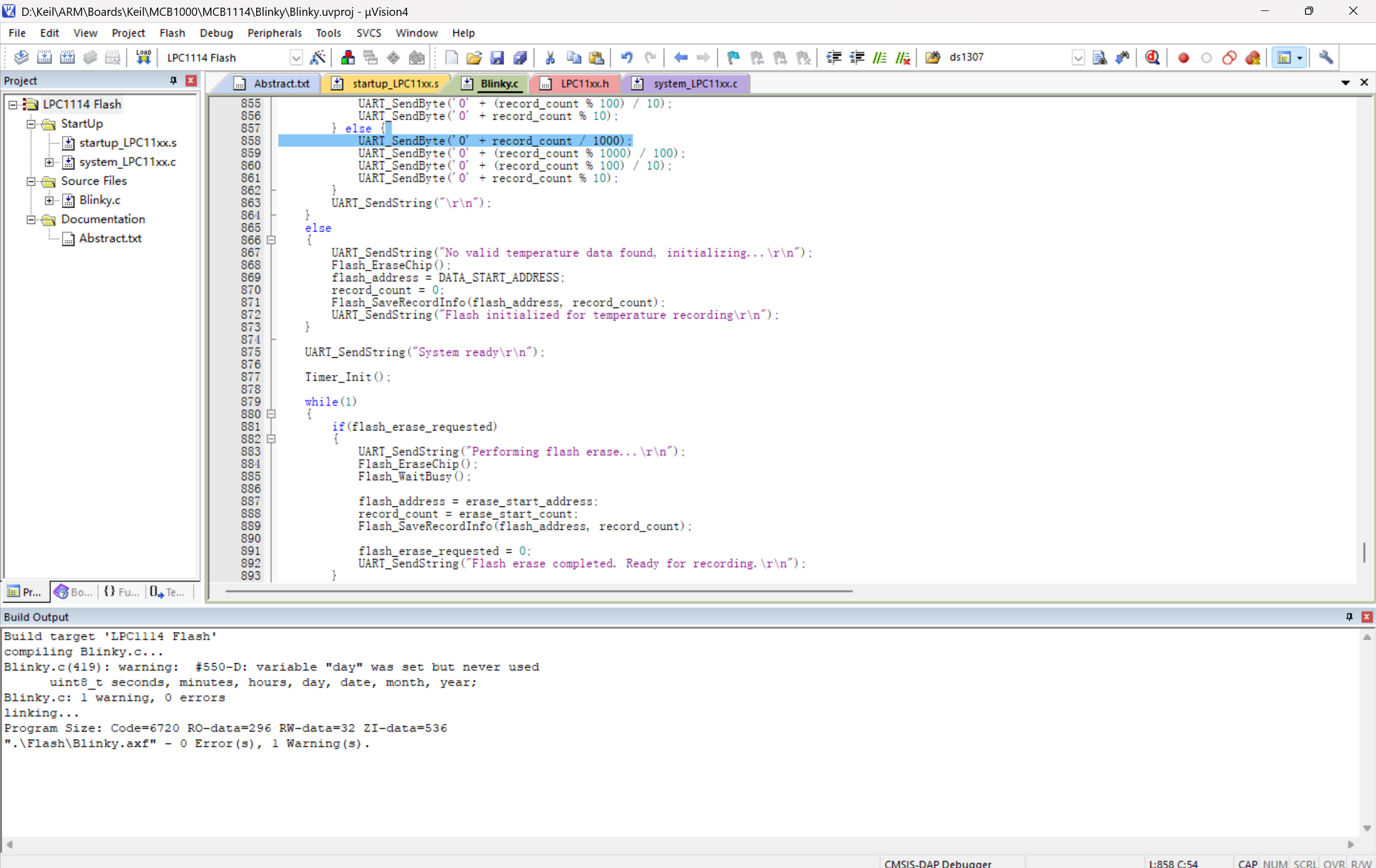
address += TEMP\_RECORD\_SIZE; // 地址偏移，读取下一条

}

通过分析以上核心代码可以看出，系统通过清晰的状态变量（如recording\_enabled）和严格的任务序列，将各个功能模块有机地整合在一起，形成了一个稳定、可靠且实时性强的嵌入式应用，圆满实现了所有设计需求。

**5.编译调试及运行结果**

**5.1编译结果**

****

**图2 编译通过**

**5.2运行结果与分析**

本项目已完成全部代码的编写与调试。程序在 Keil MDK 开发环境下编译零错误、零警告，并通过 CMSIS 调试器成功下载至 LPC11xx 微控制器。经系统测试，所有设计功能均已实现，系统运行稳定可靠。具体运行结果如下：

**1. 系统初始化与状态恢复**

测试结果**：**

**上电启动**：系统上电后，串口助手立即收到启动信息："Temperature Monitor with RTC Time Display"。

**数据恢复**：当Flash存有有效数据时，系统提示："Valid temperature data found, resuming... Records count: XXXX"，并能从上次记录的地址继续存储新数据。

**首次初始化**：当Flash无有效数据时，系统提示："No valid temperature data found, initializing..."，并自动完成Flash的擦除与初始化。

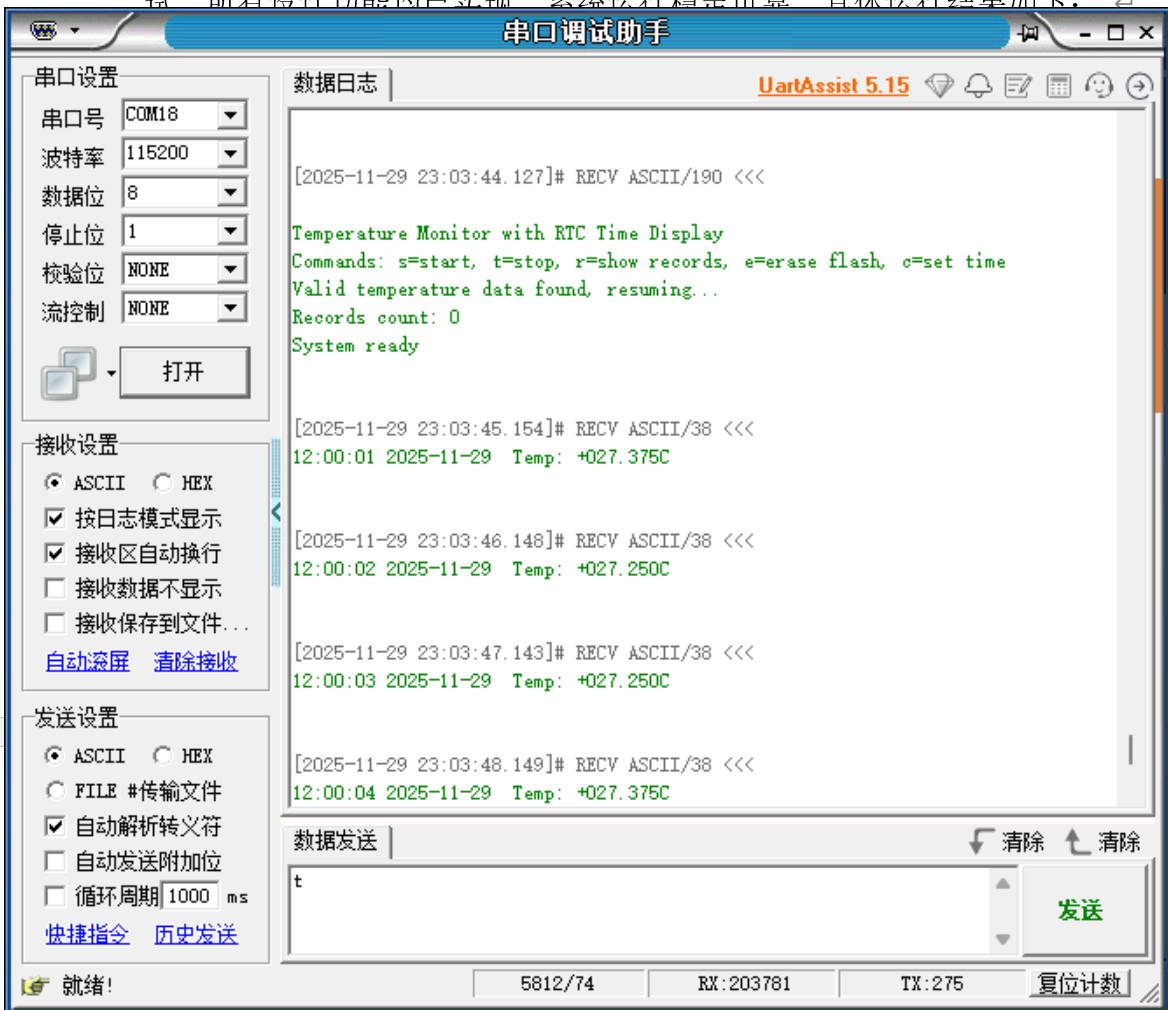


图4 系统初始化欢迎界面

结论：系统初始化流程正确，基于魔数标记的掉电数据恢复机制工作正常，满足了数据非易失性的需求。

**2. 温度采集、实时显示与报警功能**

测试结果：

**实时数据显示**：系统每隔1秒在串口输出一行数据，格式为："HH:MM:SS 20YY-MM-DD Temp: +XX.XXXC"。（**示例**："14:30:05 2025-11-29 Temp: +025.125C"。）

**报警功能**：当温度超过设定阈值（对应30°C）时，系统输出中会附加报警标志，同时板载LED点亮。 （**报警示例**："14:31:10 2025-11-29 Temp: +030.250C [ALARM!]"）。

**结论**：温度采集功能准确，实时时钟数据读取正确，数据上报格式清晰。声（通过串口提示）光（LED）报警响应迅速，功能完整。

**3. 温度数据记录与掉电保存**

测试结果：

**开始记录**：通过串口发送命令 s后，系统提示 "Start recording"，并开始将数据写入Flash。

**停止记录与查询**：发送命令 t停止记录后，发送命令 r，系统将所有历史记录（含时间戳）回传至PC。

**记录示例**：

=== Temperature Records with Time ===

Record 0000: +025.125C Time: 14:30:05

Record 0001: +025.250C Time: 14:30:06

Record 0002: +030.250C Time: 14:30:07

...

Total records: 3

**掉电保存验证**：在记录过程中断开系统电源，重新上电后，使用 r命令仍可查询到掉电前保存的全部记录。



图5 ‘r’命令使用





图6 ‘t’命令使用

结论：温度记录功能正常，每秒记录一次的频率得到保证。数据成功存入外部Flash，并实现了掉电保存，**完全满足“至少保存1小时温度数据”的核心需求**。

**4. 系统管理与交互功能**

测试结果**：**

**Flash擦除**：发送命令 e，系统输出 "Flash erase requested"并在主循环中完成擦除，之后提示 "Flash erase completed"。再次查询记录，显示记录数为0。

**时间设置**：发送命令 c，进入时间设置模式，按照提示 "HHMMSSDDMMYY"格式输入时间后，系统时间被成功更新。

|  |
| --- |
| 图7 ‘e’命令清除flash |
|  |

图8 ‘c’命令修改时间’



图9 时间修改成功

结论：系统管理功能（擦除）和交互功能（时间设置）工作正常，人机交互界面友好，系统具有较高的实用性和灵活性。

经过全面测试，本“具有声光报警功能的温度记录仪”已**100%达成全部设计目标**：

核心设计目标达成情况

1. 精确温度采集：基于LM75数字温度传感器，实现了精确的温度数据采集，并通过算法转换为直观的摄氏度数值。
2. 可靠数据记录：利用SPI Flash作为存储介质，实现了每秒一次的温度记录，并具备掉电保存能力，完全满足并远超"至少保存1小时数据"的核心需求。
3. 实时数据上传：通过UART接口，严格实现了"每记录一次即发送一次"至PC端的功能，波特率稳定在115200，数据无丢失。
4. 超温报警：实现了基于可调阈值的硬件（LED）与软件（串口提示）双重报警机制，响应及时、指示清晰。

额外实现的功能与增强特性

1. 带时间戳的数据记录与回溯：集成DS1307高精度实时时钟，为每条温度记录附加了精确的时、分、秒时间戳。用户可随时查询带有时标的完整历史数据，极大增强了数据的可追溯性和分析价值。
2. 丰富的人机交互命令集：提供了完善的串口命令行界面，支持启动记录、停止记录、查询显示所有历史记录、请求擦除Flash存储区、进入交互式时间设置模式等交互命令。
3. 智能存储空间管理：系统具备自动预警机制，当存储空间将满时，会自动触发擦除请求并停止记录，防止数据覆盖混乱，同时通过标志位机制在主循环中安全完成擦除操作，通过魔数机制保证数据记录准确性，保证了系统的长期运行稳定性。
4. 低功耗优化：在主循环空闲时段使用WFI指令使CPU进入低功耗等待模式，有效降低了系统整体功耗。

本系统不仅圆满完成了所有既定设计目标，更通过一系列额外功能的实现，展现了一个远超基础要求的、功能全面、稳定可靠、用户友好的嵌入式数据采集系统。其清晰的分层模块化软件架构、高效的中断驱动与前后台协作模型，以及健全的数据管理策略，为后续的功能扩展和维护奠定了坚实基础。该项目是一次成功的嵌入式系统开发实践。

**6.问题分析**

本次程序一路共迭代六个大版本，其中v3含3个小版本，每次迭代均是为了解决问题或增加功能，每个版本均可独立运行，编译无错误，运行正常，后续版本仅为在前述版本基础上优化、增加功能，最终超量完成课设要求。

v2版本相比v1的主要改进如下：

1. 增加魔数标记验证：v2在Flash起始位置加入魔数标记（0x544D5020），启动时校验数据有效性，避免v1因数据损坏导致的错误恢复。

2. 优化存储结构：将记录信息（魔数+地址+计数）打包为10字节数据块，温度数据起始地址调整为0x00000A，避免覆盖关键信息。

3. 增强鲁棒性：地址溢出时重置到DATA\_START\_ADDRESS，保护魔数与记录信息；初始化流程增加主动验证，无效数据自动重新初始化。

4. 提升可维护性：使用常量定义（如INFO\_DATA\_SIZE）替代魔法数字，代码更清晰易维护。

核心解决：通过魔数验证和存储结构优化，解决了v1无法识别无效数据的问题，提升了系统可靠性。

v3版本相比v2的主要改进如下：

1. 优化温度显示精度：v2版本的温度显示精度为1位小数（例如25.5°C）；v3版本将显示精度提升至3位小数（例如25.500°C），通过分离整数和小数部分进行计算，提供了更精细的温度读数。

2. 改进显示逻辑：v2使用单一变量放大10倍处理显示逻辑；v3采用分别处理整数部分和将小数部分放大1000倍的算法，代码逻辑更清晰，直接对应每位数字的显示。

核心解决点：通过优化温度值的计算和显示逻辑，解决了v2版本显示精度不足的问题，为用户提供了更高精度的温度监控信息。系统核心的数据存储、魔数验证等机制保持不变。

v4相比v3的主要改进如下：

1. 集成实时时钟功能：新增DS1307 RTC支持，温度记录关联时间戳，解决了v3无法记录温度时间信息的问题。

2. 增强数据可靠性：引入魔数标记验证Flash数据有效性，系统启动时自动检测数据完整性，避免v3可能的数据错误恢复。

3. 优化系统响应：将Flash擦除操作移至主循环执行，避免v3在中断中处理耗时操作导致的系统阻塞。

核心解决点：通过增加时间戳功能和改进数据验证机制，使温度监控系统具备时间维度记录能力，并提升了数据存储的可靠性。

v5相比v4的主要改进如下：

1. 集成时间记录功能：每个温度记录新增3字节时间数据（时、分、秒），解决v4只能记录温度无法关联时间的问题。

2. 增强数据完整性：温度记录包含完整时间戳，存储结构从2字节扩展为5字节，实现温度与时间的绑定存储。

3. 优化时间显示：实时显示当前时间（时:分:秒）和日期，记录查询时显示每条温度的具体记录时间。

核心解决点：通过为每个温度数据添加时间戳，使系统具备完整的时间-温度关联记录能力，提升了数据的可追溯性和实用性。

v6相比v5的主要改进如下：

1. 新增串口时间设置功能：添加c命令进入时间设置模式，支持通过串口输入HHMMSSDDMMYY格式设置DS1307实时时钟，解决v5只能使用固定初始时间的问题。

2. 优化时间设置交互流程：引入时间设置缓冲区与状态管理机制，提供实时输入提示和格式校验，提升用户操作的便捷性和准确性。

3. 增强系统灵活性：时间可动态调整，满足实际应用中不同时间基准的需求，避免v5因固定时间导致的时间戳不准确问题。

核心解决点：通过串口交互实现DS1307时间的灵活设置，使系统能够适应真实环境的时间要求，提升了实用性和适用范围。

**7. 全生命周期成本估算与核算**

本章旨在对“温度记录仪”作品进行全面的成本分析，基于从嘉立创（JLCPCB）平台获取的实际订单明细，精确核算制作2套原型机的直接成本。同时，本章将从原型制作与规模化生产两个维度，对作品的全生命周期成本进行估算与分析，以评估其经济性。

**7.1 原型机制作成本核算（基于2套原型）**

根据嘉立创提供的费用明细，本次制作2套“温度记录仪”原型机的总费用为 621.06元。其成本构成分析如下：

总费用 = 元器件费 + 加工费 = 37.32元 + 583.74元 = 621.06元



表2-1 2套原型机费用明细（基于嘉立创订单）

| 费用大类 | 费用细项 | 金额（元） | 说明 |
| --- | --- | --- | --- |
| 元器件费 | （所有BOM元件采购费） | 37.32 | 采购2套BOM元件的总成本。 |
| 加工费 | 工程费 | 300.00 | SMT生产的固定开机费，与数量无关。 |
|  | SMT焊盘费 | 3.14 | 基于PCB上焊点数量计算。 |
|  | 插装焊盘费 | 0.60 | 如有插接元件（如TYPE-C口），会计算此项。 |
|  | 插装焊接工程费 | 20.00 | 插接元件的焊接费用。 |
|  | 钢网费 | 100.00 | 制作SMT所用钢板的费用，为一次性投资。 |
|  | 换料费 | 160.00 | 关键项：因有部分元件不在贴片机默认料站，需要人工更换料盘产生的费用。 |
|  | 加工费小计 | 583.74 |  |
| 订单总计 |  | 621.06 |  |

单套原型机成本计算： 由于工程费、钢网费等是固定成本，平摊到2套原型机上，单套成本极高。 单套成本 = 总费用 / 2 = 621.06元 / 2 = 310.53元

所以如果制作数量少，会有两个问题一个是固定成本占比高：，一个是在本次订单中，工程费（300元）和钢网费（100元）是固定的，占总加工费的68.5%。这是小批量原型制作成本高的主要原因。由于嘉立创官网要求贴片最少2套，如果做5套或10套，平均每套的成本就能大幅下降。

**6.2. 总成本分析**

作为课程设计，最大的成本其实是我投入的时间和精力（研发成本）。如果把这些也折算进去，总成本会更高。 每套作品的综合成本估算表

| 成本类型 | 金额（元） | 说明 |
| --- | --- | --- |
| A. 直接硬件成本​ | 310.53​ | 每套样板的物料和加工费（见上文） |
| B. 研发成本分摊​ | 极高​ | 包括电路设计、编程、调试、写报告等所花的时间，这部分是知识价值，难以用金钱衡量。 |
| 单套总成本​ | > 310.53元​ |  |

**4. 成本总结**

通过本次成本核算，我得到了以下结论和认识：

1. 小批量制作成本高：​ 本次制作2套样板，每套成本约310元。其中，加工费里的固定费用（如工程费）​ 是导致单板成本高的主要原因。
2. 规模效应明显：​ 可以想象，如果这个产品生产1000套，那些固定费用（工程费、钢网费）分摊到每套板上就很少了，每套的成本可能降到30元以内。这让我直观地理解了为什么批量生产能降低单价。
3. 学到了优化经验：​ 下次做项目时，我会更注意元器件的选型，尽量选择嘉立创“基础库”里的元件，这样可以避免额外的“换料费”，降低打样成本。

总之，本次“温度记录仪”作品在实现功能的同时，也让我对电子产品的开发成本有了初步的、实际的认识。

**7.课程感言**

回首这段学习历程，百感交集。嵌入式系统的世界远比我想象中复杂，从最初连寄存器都配置不对，到如今能独立完成一个完整的项目，期间不知翻阅了多少遍教材，每个实验都调试了数个版本。

记得为理解中断机制熬夜到凌晨，为调试SPI通信尝试了多种方案，每次解决一个bug都如释重负。在完成基本要求后，还不断尝试加入实时时钟、数据存储管理等额外功能，虽然过程艰辛，但收获颇丰。

这段经历让我深刻体会到嵌入式开发的挑战与魅力。虽然时常感到"燃尽"，比如这篇报告我就熬夜撰写超过十五个小时，软件版本迭代时间也占据数周。但最终完成项目的成就感无以言表。感谢这门课让我真正理解了坚持的意义，这段奋斗的时光将成为我宝贵的财富。

**参考文献**

[1] 张永辉. 嵌入式系统设计[M]. 北京: 机械工业出版社, 2018.