《嵌入式系统及应用》课程

综合实验报告

学院　　仪器科学与光电工程学院

专业　　测控技术与仪器

学号　　2023210238

姓名　　李一赫

2025 年 5 月 16 日

摘 要

本实验通过理论与实践结合，使学生掌握 STM32F103 单片机的综合开发技能，理解嵌入式系统的核心设计理念，并为后续复杂项目（如物联网终端、工业控制器）的开发奠定坚实基础。

在硬件调试（如ADC 噪声抑制、PWM 波形验证）和软件调试（如中断冲突、内存溢出）中，积累嵌入式系统调试经验，掌握逻辑分析仪、串口调试工具的使用方法。

培养工程文档编写与团队协作意识。通过撰写实验报告、记录测试用例、维护代码注释，规范技术文档编写习惯；通过差异化任务分工，理解团队协作在复杂项目中的重要性。

目录

1 实验目的

i. **掌握 STM32F103 核心外设的应用** - 通过实际操作”GPIO、ADC、定时器、中断、串口通信” 等模块，深入理解单片机外设的工作原理及配置方法，培 养对硬件资源的直接控制能力。

ii. **培养嵌入式系统全流程开发能力**- 从硬件连接（传感器、LED、按键）到软件编程（驱动开发、协议解析），完成完整的嵌入式系统设计流程，提升系统级工程思维。

iii. **学习多模块协同与系统调试技巧**- 实现”ADC 采集、PWM 输出、串口通信、中断响应” 等任务的协同工作和相关程序调试技巧。

2 实验原理

2.1 STM32 硬件架构基础

STM32f103 是基于 ARM Cortex-M3 的 32 位微控制器，核心特性包括：

- **内核**：负责指令执行、中断处理和内存访问。
- **存储器**：Flash（存储程序代码）和 SRAM（运行数据）。
- **外设**：GPIO、定时器（TIM）、ADC/DAC、USART、SPI、I2C、USB、CAN 等。
- **时钟系统**：通过 HSI（内部高速时钟）、HSE（外部高速时钟）、PLL（锁相 环倍频）等配置系统时钟。
- **电源管理**：支持多种低功耗模式（Sleep/Stop/Standby）。

2.2 温度采集原理

模拟温度传感器会根据环境温度输出相应的电压信号，其输出电压与温度呈线性关系。例如，输出电压为 10mV/℃，当环境温度为 25℃时，输出电压为 250mV。STM32F103 单片机的 ADC 模块将传感器输出的模拟电压信号转换为数字信号，以便单片机进行处理和分析。ADC 具有一定的分辨率，如 12 位分辨率的 ADC 可将输入电压范围（如 0 - 3.3V）量化为 4096 个不同的数字值。通过将传感器输出

电压与 ADC 的量化范围进行映射，单片机就能计算出对应的温度值。

2.3 串口通信原理

单片机的串口模块负责与上位机进行数据通信。在发送端，单片机按照规定的格式（如 “Temp: 25.6° C”）将温度数据打包成字节流，然后通过串口引脚将数据逐位发送出去。在接收端，串口接收中断被触发，当接收到学号末两位（16 进制）时，单片机将当前温度阈值发送回上位机；若接收到的不是自己的学号，则返回 “invalid instruction.”。串口通信需要设置波特率、数据位、停止位和校验位等参数，以确保通信双方能够正确地发送和接收数据。

2.4 LED 状态指示原理

对于绿灯，在上电时，通过控制 PA0 引脚的高低电平，使其闪烁 2 次后常亮，以指示系统工作正常。当温度处于正常范围内时，保持 PA0 引脚为高电平，使绿灯亮起。红灯（PA1）采用控制实现呼吸灯效果。PWM 通过改变输出脉冲的占空比来控制 LED 的平均亮度。当温度超过阈值时，单片机通过定时器产生 PWM 信号，逐渐改变占空比，使红灯的亮度逐渐增加和减小，形成呼吸灯效果，起到警示作用。

2.5 按键控制原理

按键连接到单片机的 PA8 引脚，当按键按下时，会产生一个外部中断信号。单片机检测到该中断后，进入中断服务程序，在程序中实现循环切换三档高温阈值的功能。每次切换阈值后，将最新阈值通过串口发送到上位机，以便用户了解当前的温度阈值设置。同时，利用外部中断可以避免单片机一直查询按键状态，提高系统的效率和实时性。

2.6 实验开发工具

开发工具调试工具等

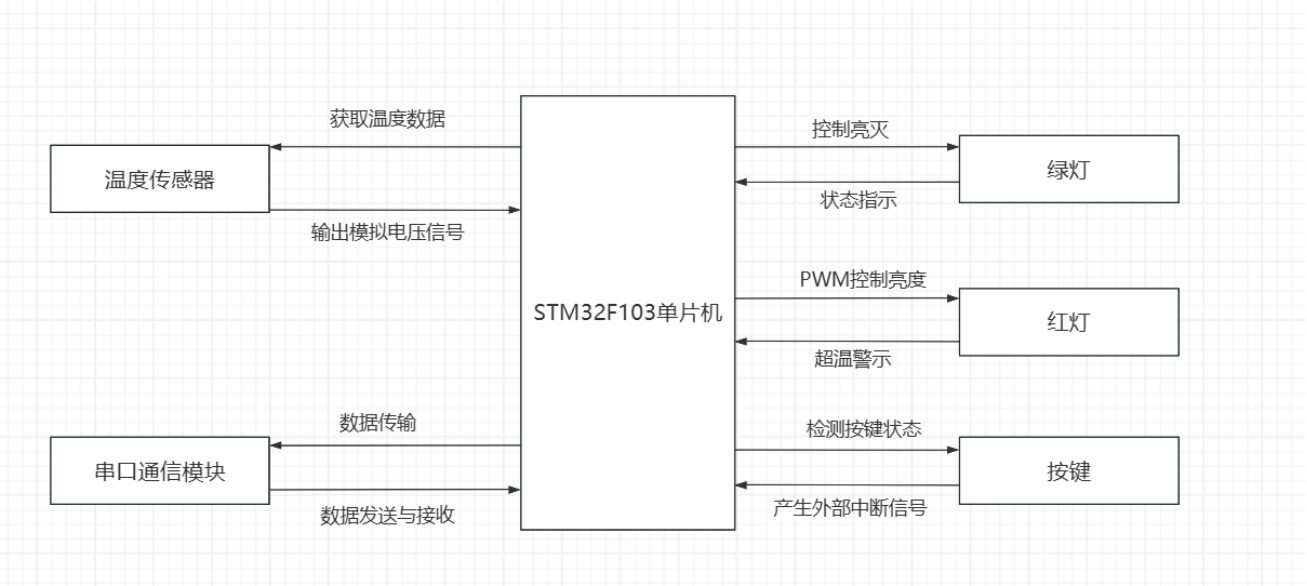


图 1流程图

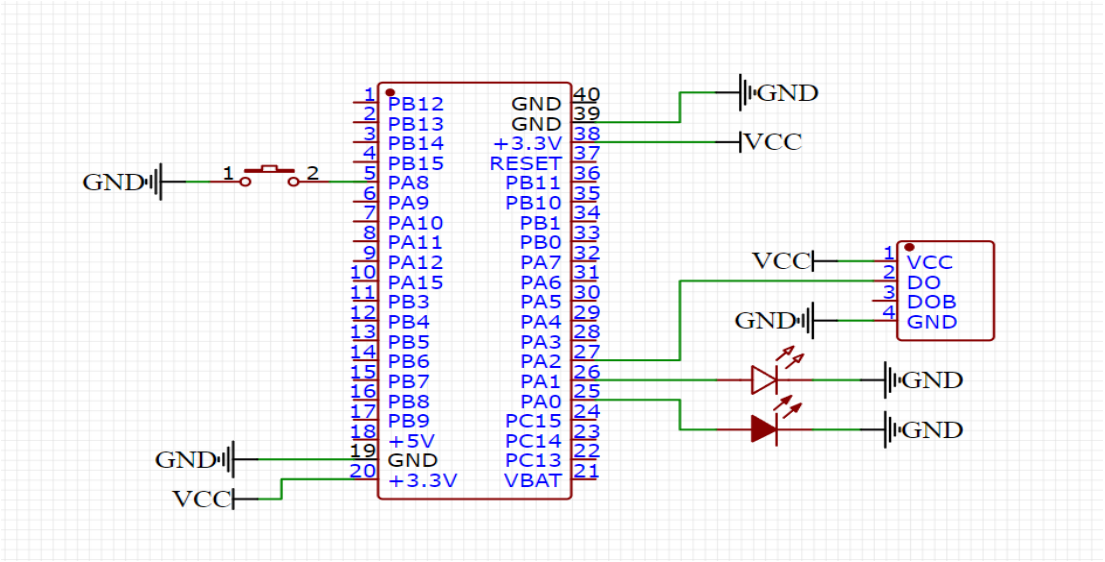
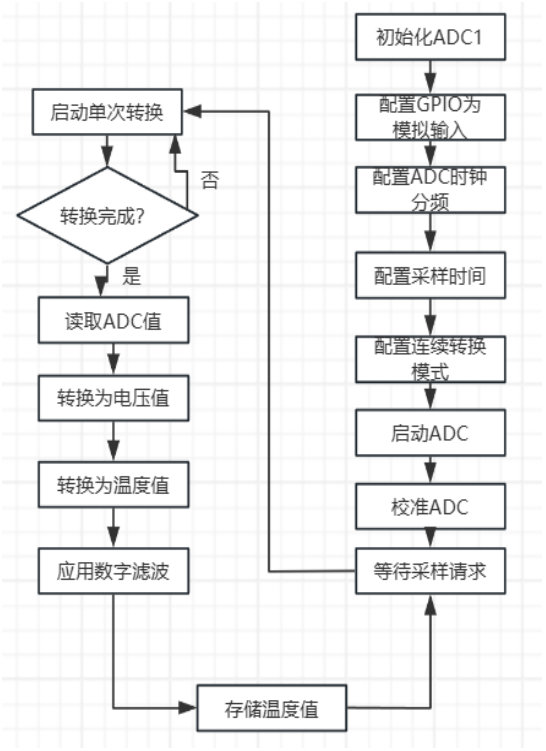


图2 硬件电路设计

3　实验内容

3.1　ADC：模数转换

ADC软件流程图

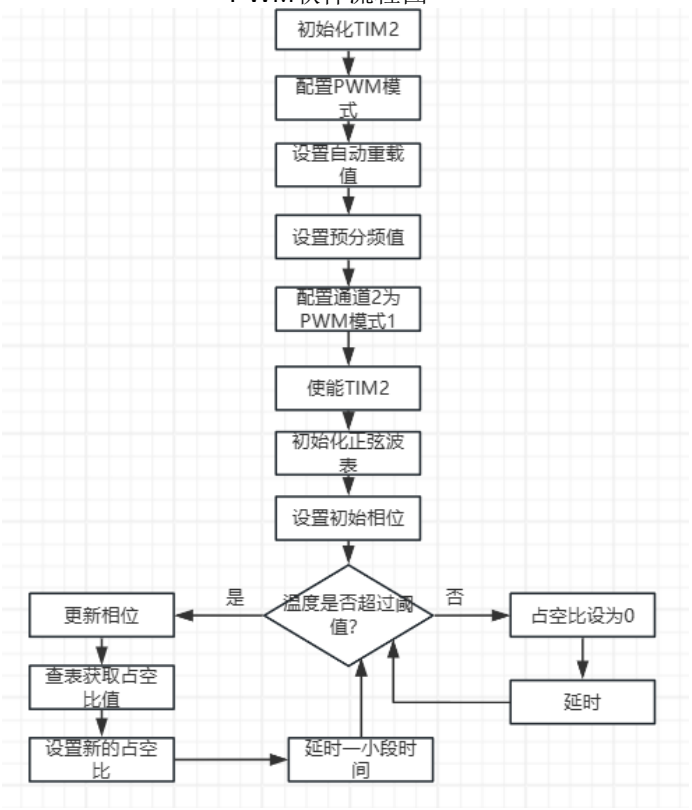


ADC关键代码

```
//    ADC1GPIO
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE)
    ;
//   ADC
RCC_ADCCLKConfig(RCC_PCLK2_Div6);
//     GPIO
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOC, &GPIO_InitStructure);
//   ADC
ADC_InitTypeDef ADC_InitStructure;
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_DataAlign =
    ADC_DataAlign_Right;
ADC_InitStructure.ADC_ExternalTrigConv =
    ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
//  ADC
ADC_Cmd(ADC1, ENABLE);
//
ADC_ResetCalibration(ADC1);
while (ADC_GetResetCalibrationStatus(ADC1) == SET);
//
ADC_StartCalibration(ADC1);
while (ADC_GetCalibrationStatus(ADC1) == SET);
```

3.2 PWM模块

PWM软件流程图

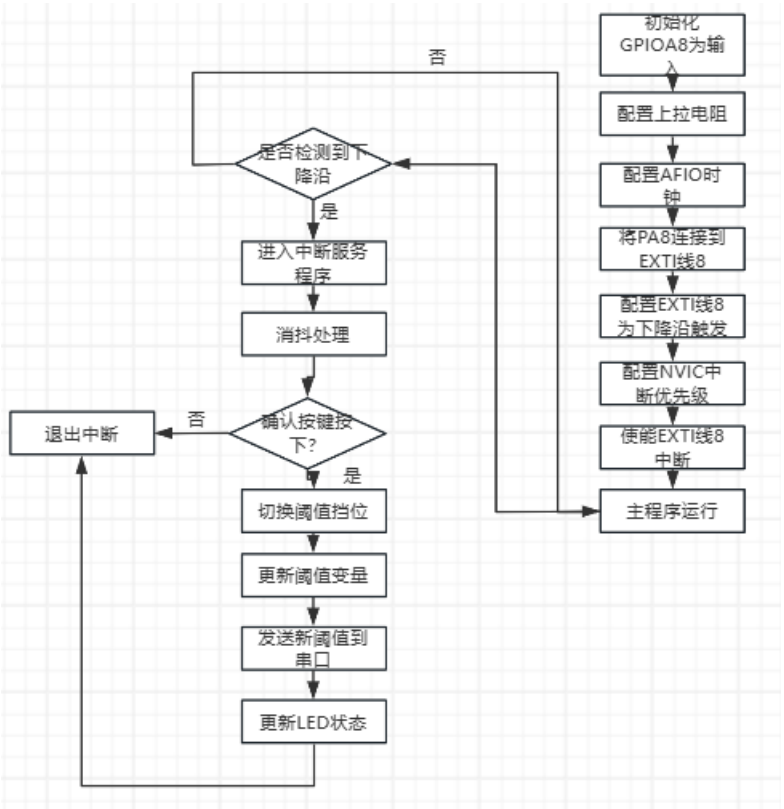


PWM 关键代码

```
//    TIM2GPIOA
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
//      GPIO
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
//    TIM2
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInitStructure.TIM_CounterMode =
    TIM_CounterMode_Up;
TIM_TimeBaseInitStructure.TIM_Period = 1000 - 1;
TIM_TimeBaseInitStructure.TIM_Prescaler = 7200 - 1;
TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStructure);
//      TIM2OC2PWM1
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCStructInit(&TIM_OCInitStructure);
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable
    ;
TIM_OCInitStructure.TIM_Pulse = 0;
TIM_OC2Init(TIM2, &TIM_OCInitStructure);
//  TIM2
TIM_Cmd(TIM2, ENABLE);
```

3.3 中断模块

中断模块软件流程图

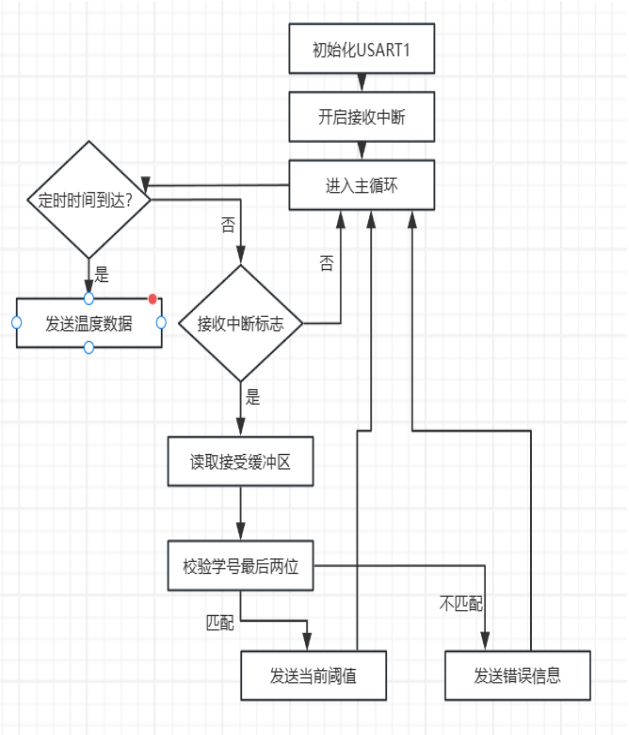


中断模块关键代码

```
//          NVICUSART1_IRQn00
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
//          NVICTIM3_IRQn10
NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
//           EXTIPA8
EXTI_InitTypeDef EXTI_InitStructure;
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource8);
EXTI_InitStructure.EXTI_Line = EXTI_Line8;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
//          NVICEXTI9_5_IRQn20
NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
//          EXTI
void EXTI9_5_IRQHandler(void) {
        if (EXTI_GetITStatus(EXTI_Line8) != RESET) {
                switch (Threshold) {
                        case 25:
                        Threshold = 30;
                        break;
                        case 30:
                        Threshold = 15;
                        break;
                        case 15:
                        Threshold = 25;
                        break;
                }
                Send_Threshold();
                EXTI_ClearITPendingBit(EXTI_Line8);
        }
}
```

3.4 串口模块

串口模块软件流程图

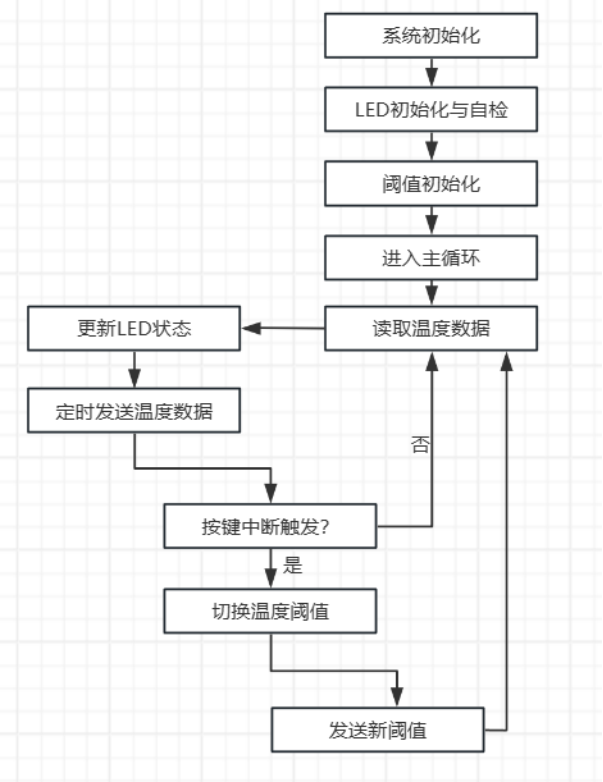


串口模块关键代码

```
//    USART1GPIOA
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA,
    ENABLE);
//       TX
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
//      RX
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);
//    USART1
USART_InitTypeDef USART_InitStructure;
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
    USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART1, &USART_InitStructure);
//    USART1
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
//  USART1
USART_Cmd(USART1, ENABLE);
//       USART1
void USART1_IRQHandler(void) {
        if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
                uint8_t received = USART_ReceiveData(USART1);
                if (UART_RX_Index < 2) {
                        UART_RX_Buffer[UART_RX_Index++] = received;
                }
                if (UART_RX_Index == 2) {
                        uint8_t value = 0;
                        uint8_t valid = 1;
                        for (int i = 0; i < 2; i++) {
                                uint8_t c = UART_RX_Buffer[i];
                                if (c >= '0' && c <= '9') {
                                        value = value * 16 + (c - '0
                                            ');
                                } else if (c >= 'A' && c <= 'F') {
                                        value = value * 16 + (c - 'A
                                            ' + 10);
                                } else if (c >= 'a' && c <= 'f') {
                                        value = value * 16 + (c - 'a
                                            ' + 10);
                                } else {
                                        valid = 0;
                                        break;
                                }
                        }
                        if (valid && value == MyStudentIDLastTwo) {
                                Send_Threshold();
                        } else {
                                char *msg = "Invalid instruction.\r\
```

```
                n";
            for (int i = 0; msg[i] != '\0'; i++)
                {
                    while (USART_GetFlagStatus(
                        USART1, USART_FLAG_TXE)
                        == RESET);
                    USART_SendData(USART1, msg[i
                        ]);
                }
        }
        UART_RX_Index = 0;
    }
    USART_ClearITPendingBit(USART1, USART_IT_RXNE);
}
}
```

3.5 主程序流程



4 实验步骤

4.1 步骤 1：将代码烧录到单片机

1. 点击 Keil 工具栏中的按钮，编译项目代码，确保没有编译错误。
2. 用 ST-Link 仿真器将 STM32F103 开发板与电脑连接，检查连接是否稳固。
3. 在 Keil 中配置 ST-Link 调试器：选择 "Options for Target"，在 "Debug" 选项卡中选择 "ST-Link Debugger"，然后点击 "Settings" 进行 ST-Link 的具体设置，如选择 SWD 接口、设置合适的下载速度等。
4. 点击 Keil 工具栏中的 "Download" 按钮，将编译好的程序下载到 STM32F103 单片机中。

5. 下载完成后，观察开发板上的 LED 状态，若绿灯闪烁 2 次后常亮，说明程序已成功烧录且系统正常启动。

4.2 步骤 2：上电观察 led 等

1. 确保开发板电源连接正常，按下电源开关给系统上电。
2. 观察绿色 LED（PA0）的状态：正常情况下，上电后绿色 LED 会先闪烁 2 次，然后保持常亮，这表示系统已正常启动且工作状态良好。
3. 观察红色 LED（PA1）的状态：在系统上电初始阶段，若环境温度低于默认阈值（30℃），红色 LED 应保持熄灭状态；若环境温度高于默认阈值，红色 LED 应呈现呼吸灯效果，即亮度会有规律地从暗到亮再到暗循环变化。
4. 若 LED 状态与预期不符，检查硬件连接是否正确，特别是 LED 与单片机引脚的连接；检查代码中 GPIO 的配置是否正确；使用调试工具（如串口打印调试信息）排查问题。

4.3 步骤 3：改变温度

1. 使用温度源改变温度传感器周围的环境温度。
2. 当温度升高超过当前阈值时，观察红色 LED（PA1）应开始呈现呼吸灯效果，同时绿色 LED（PA0）应熄灭，表示系统检测到温度异常。
3. 当温度降低到阈值以下时，观察红色 LED 应停止呼吸灯效果并熄灭，绿色 LED 应重新亮起，表示系统恢复正常状态。
4. 使用串口调试助手，查看单片机定时发送的温度数据，确认显示的温度值与实际环境温度变化趋势一致。
5. 若系统对温度变化的响应不符合预期，检查传感器的连接是否稳固，输出信号是否正常；检查 ADC 采样和温度转换代码是否正确；检查阈值比较和 LED 控制逻辑是否存在问题。

4.4 步骤 4：串口通信测试

1. 打开串口调试助手，设置波特率为 115200bps，数据位 8 位，停止位 1 位，无

校验位，与程序中配置的串口参数一致。

2. 观察串口调试助手的接收窗口，应能定时看到单片机发送的温度数据，格式为 "Temp: XX.X° C"。

3. 在串口调试助手的发送窗口中，输入学号末两位的 16 进制数据。

4. 正常情况下，串口调试助手应收到单片机返回的当前温度阈值信息，格式为 "Threshold: XX.X° C"。

5. 若发送其他非学号末两位的数据，串口调试助手应收到 "invalid instruction." 的响应。

6. 若串口通信不正常，检查串口线连接是否正确；检查串口参数配置是否一致；检查串口接收中断和命令解析代码是否正确。

4.5 步骤 5：按键功能测试

1. 初始状态下，系统默认的温度阈值为 30℃，可通过串口调试助手发送学号末两位查询确认。

2. 按下开发板上连接到 PA8 引脚的按键，注意按键按下后应立即松开，避免长按导致多次触发。

3. 每按一次按键，系统应循环切换温度阈值，切换顺序为 25℃→30℃→35℃→25℃→...。

4. 每次按下按键后，观察串口调试助手是否收到新的阈值信息，格式为 "Threshold: XX.X° C"。

5. 改变环境温度，验证当温度超过当前阈值时，红色 LED 是否呈现呼吸灯效果，绿色 LED 是否熄灭；当温度低于当前阈值时，红色 LED 是否熄灭，绿色 LED 是否亮起。

6. 若按键功能不正常，检查按键与单片机引脚的连接是否稳固；检查按键消抖代码是否有效；检查阈值切换逻辑是否正确。

5 实验结果与分析

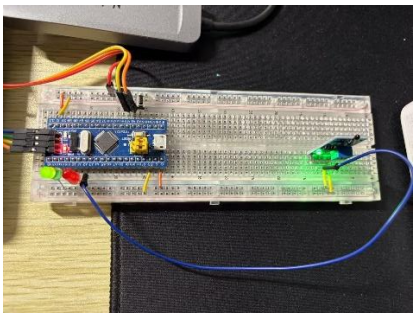
5.1 LED 指示

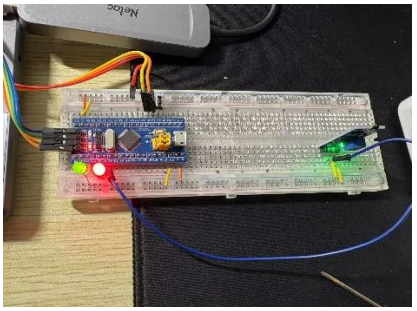
一、关键流程分析

系统初始化后，绿灯（PA0）上电闪烁 2 次后常亮，用于指示系统工作正常。当使用加热设（如热吹风机）对温度传感器（LM35）进行加热，使采集到的温度超过预设阈值（默认 30℃）时，单片机通过定时器产生 PWM 信号控制红灯（PA1），改变其占空比实现呼吸灯效果，以警示温度异常；当温度降至阈值以下，红灯熄灭，绿灯重新亮起。

二、实验结果展示

通过对温度传感器加热，当温度超过 30℃时，红灯呈现呼吸灯效果。从截图中可以清晰看到红灯亮度由暗到亮，再由亮到暗循环变化，此时绿灯熄灭。当停止加热，温度下降至阈值以下后，红灯熄灭，绿灯重新亮起，表明系统能够准确根据温度状态控制 LED 指示。





三、结果分析

实验中红灯呼吸灯效果及绿灯状态切换符合预期目标，这得益于 PWM 控制逻辑和温度阈值判断机制的正确实现。定时器产生稳定的 PWM 信号，通过正弦波查表法改变占空比，使得红灯亮度变化平滑自然。不过，在实际测试中发现，呼吸灯的频率可进一步优化，若适当降低频率，呼吸效果可能更明显；另外， 当温度接近阈值时，LED 状态切换存在轻微延迟，后续可通过优化温度采样频率和判断逻辑来改进。

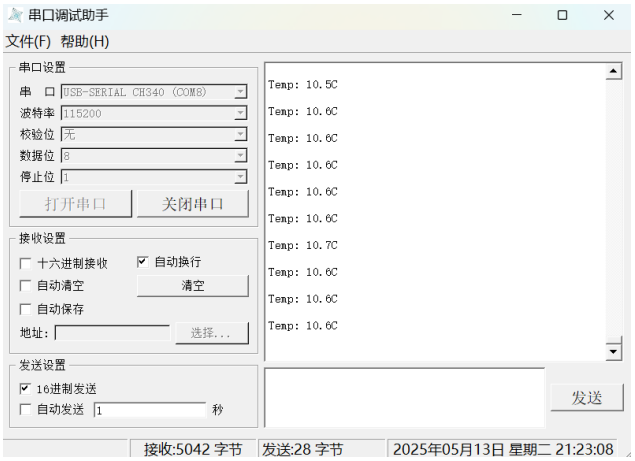
5.2 按键响应

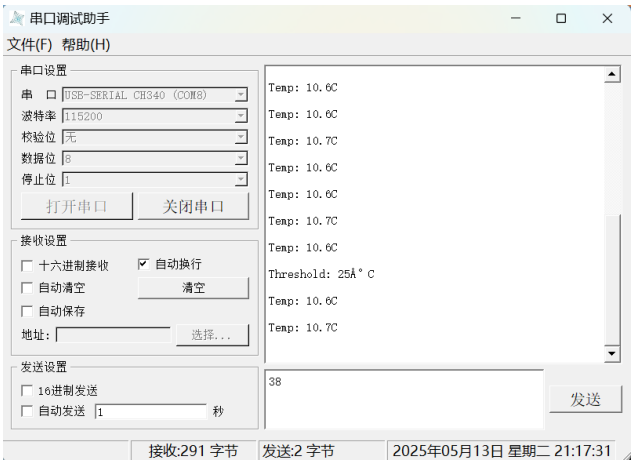
一、关键流程分析

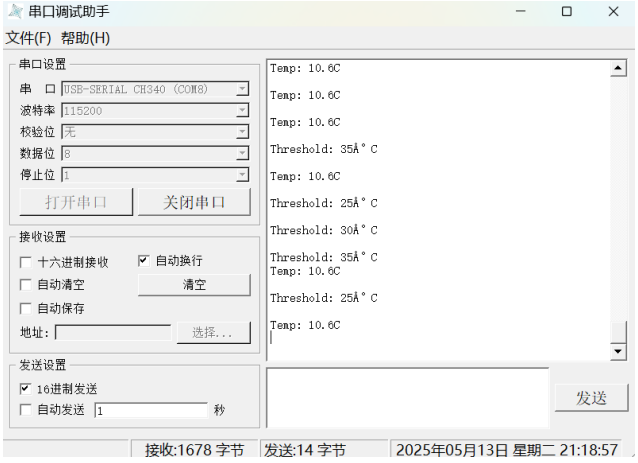
按键连接在单片机 PA8 引脚，按下按键后触发外部中断。在中断服务程序中，系统实现三档高温阈值（25℃、30℃、35℃）的循环切换，并将最新阈值通过串口发送到上位机显示。每按下一次按键，系统执行阈值切换逻辑，同时更新内部阈值变量，并调用串口发送函数输出新阈值。

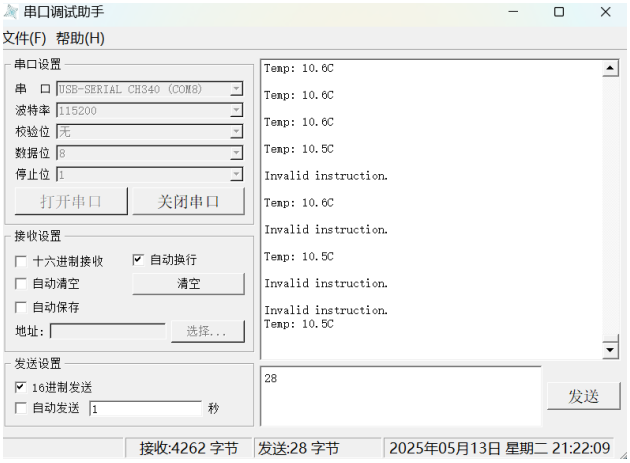
二、实验结果展示

多次按下按键，串口调试助手实时显示阈值切换信息。从截图可知，初始阈值为 30℃，按下一次按键后，阈值切换为 35℃，再次按下切换为 25℃，实现了三档阈值的循环切换功能，且串口打印的阈值信息准确无误。









三、结果分析

按键触发阈值切换功能运行稳定，串口能够及时准确地输出新阈值，说明外部中断配置和串口通信模块工作正常。然而，在快速连续按下按键时，偶尔会出现阈值跳过某一档位的情况，这可能是由于按键消抖时间设置不合理或中断处理程序执行速度不足导致。后续可调整消抖算法，延长消抖时间，同时优化中断服务程序代码，提高执行效率，确保每次按键操作都能准确响应 。

6 总结及心得体会

通过本次基于 STM32F103 单片机的综合温度监测系统实验，我系统掌握了嵌入式系统开发的全流程。从硬件连接到软件编程，每个环节都充满挑战与收获。在硬件搭建时，需严谨确认 温度 传感器、LED 及串口的连接，任何一处疏忽都可能导致系统异常；软件编程中，ADC 采样、PWM 控制、中断处理等功能模块的实现，让我深入理解了单片机各外设的工作原理和编程要点。

实验过程中，我遭遇过温度数据波动大、串口通信乱码等问题。通过查阅资料、添加数字滤波算法和仔细核对串口参数配置，最终成功解决，这极大提升了我的问题排查与解决能力。同时，模块化编程思想的应用，使程序结构更清晰，也增强了代码的可维护性。

此次实验不仅巩固了理论知识，更让我认识到实践的重要性。未来，我将进一步探索嵌入式系统优化与功能拓展，如添加无线通信模块实现远程监控，让所学知识在更多实际场景中发挥价值。

7 对本实验过程及方法、手段的改进建议

7.1 实验改进建议

在完成本次温度监测系统实验后，结合实际操作中的体验，我对实验过程及方

法、手段有以下改进想法。

硬件方面，当前系统的抗干扰能力较弱，环境中的电磁干扰可能影响温度传感器的输出。后续可以尝试在传感器和单片机之间增加滤波电路，减少信号噪声，提高数据采集的准确性。此外，实验使用的普通 LED 亮度和可视角度有限，若换成高亮度 LED，能让状态指示效果更清晰。

软件部分，虽然实现了基本功能，但代码的模块化程度还能进一步提升。目前部分函数耦合度较高，后续可以将功能划分得更细致，增加函数的复用性。在数据处理上，仅使用简单的计算转换温度值，未来可以考虑引入中值滤波等算法，让温度数据更加稳定。

在调试过程中，主要依靠串口打印信息排查问题，效率较低。如果能使用调试器的单步执行、断点设置功能，就能更精准地定位代码问题。总的来说，本次实验让我发现了很多不足，这些改进方向也为我后续学习提供了目标。

7.2 发布地址

- Github:
  https://github.com/langsunny/HFUT_Report_LaTeX_Template

直接使用\cite{}即可[1]

7.3 文献引用方法

参考文献

[1] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]// Advances in Neural Information Processing Systems 30. Long Beach: NeurIPS Foundation, 2017: 5998-6008.