南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

计算机科学与工程学院

“嵌入式系统”实验报告书

题目：ex7_922106840127_Thread

学号：922106840127
姓名：刘宇翔
成绩

日期：　　2025 年　　4 月　　11 日

1 题目要求

1. 题目设计要求

（1） 实验内容

基于 RT-Thread 操作系统编写程序，实现如下功能： 1)任务 1，将读取 AHT21 的温湿度信息，小数点后保留 1 位，显示在 LCD 屏上，300ms 刷新一次； 2)任务 2，将串口接收的字符串显示在 LCD 上，字符串长 度不超过 20 个字节，采用 DMA 结合信号量方式接收； 3)编写 PIN 设备应用，实现轻触一次 DOWN 按键，中断 回调函数里翻转红色 LED 灯；

（2） 完成要求

工程名称命名：ex7_学号_Thread，并打 包成：ex7_学号_ Thread.rar 压缩文件夹 实验报告 PDF 格式：ex7_学号_ Thread.pdf

2. 拟实现的具体功能

本次实验拟实现一个基于 RT-Thread 操作系统的嵌入式应用系统，通过任务调度方式实现多个功能模块的协同工作。

在任务 1 中，系统周期性地读取 AHT21 传感器的温湿度信息，温度与湿度数据均保留小数点后一位，并以 300ms 为刷新周期实时显示在 LCD 屏幕上；

在任务 2 中，利用 DMA 结合信号量机制实现串口数据的非阻塞接收，接收到的字符串（长度不超过 20 个字节）随后在 LCD 上进行显示；

在任务 3 中，我编写了 PIN 设备应用，通过轻触 DOWN 按键在中断回调函数中实现红色 LED 灯状态的翻转，添加了对应的消抖函数处理。

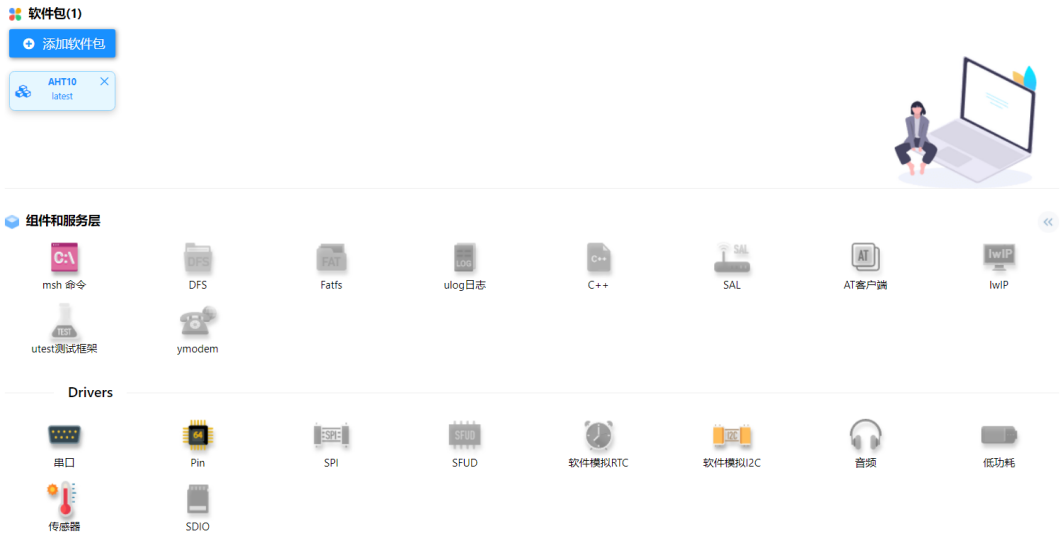
各模块均严格按照 RT-Thread 操作系统的标准 API 及多任务调度方式实现，确保数据采集、通信与外设控制的实时性和系统整体的稳定性，为未来功能扩展奠定坚实基础。

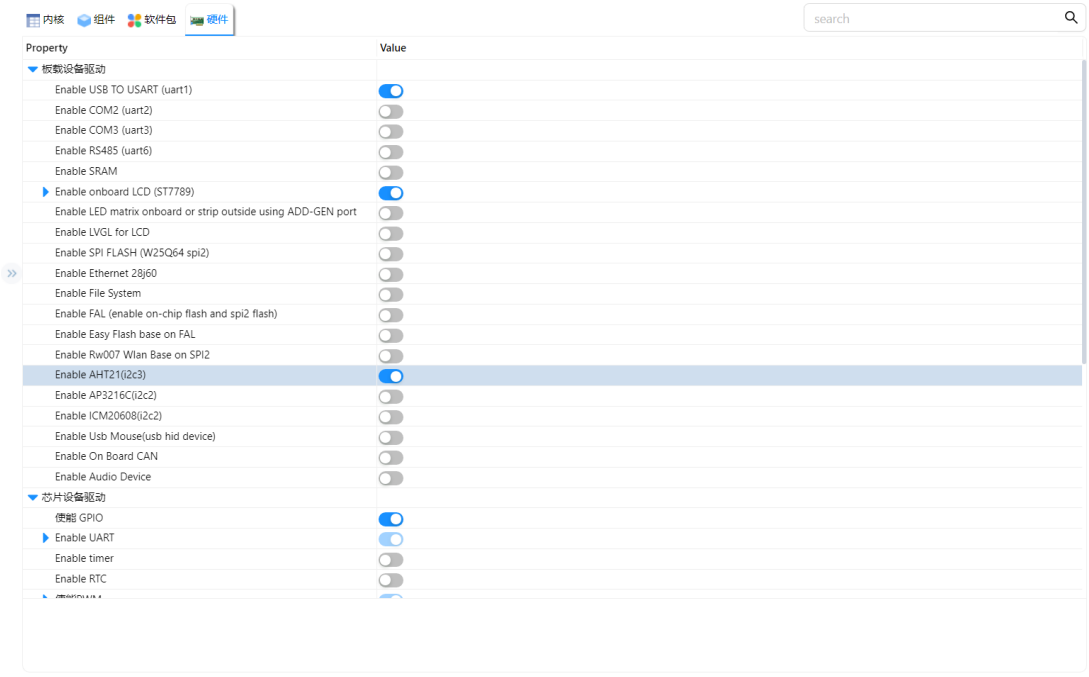
2 总体设计

2.1 硬件设计

1.硬件设计思路

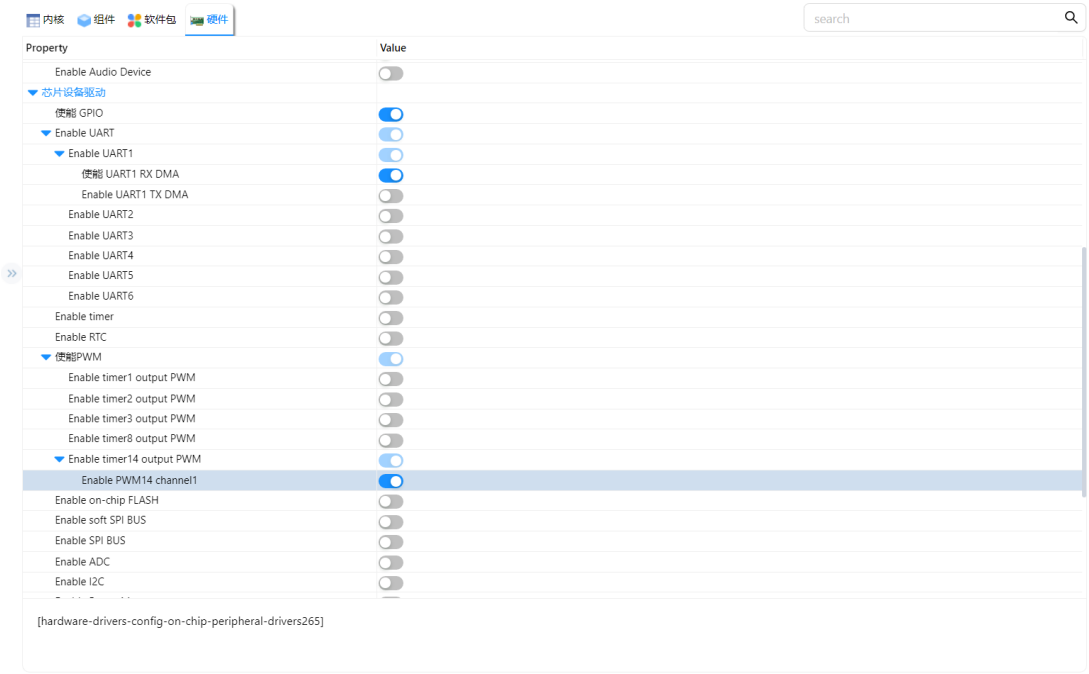
（1） 任务 1：AHT21 温湿度传感器 LCD 显示





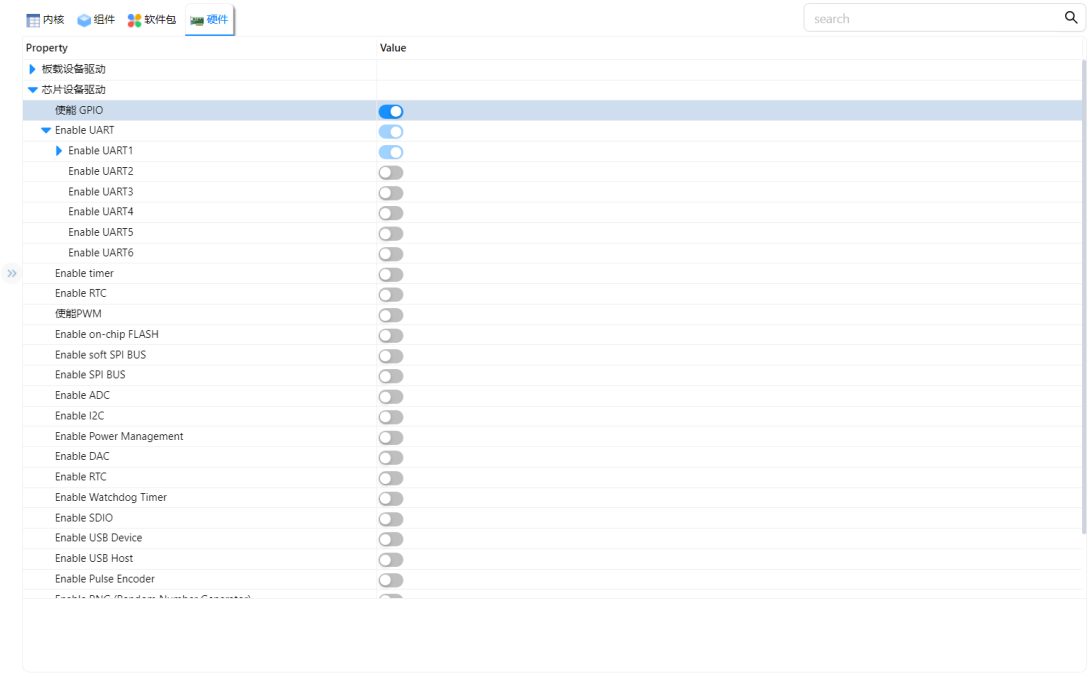
如图所示，我在 RT-Thread 项目设置中添加了有关 AHT10 的软件包，其中的硬件配置即是对 AHT21 传感器打开驱动。LCD 的相关设置在前几个实验中已提及过多次，在此不再展示，基本就是 FSMC 的相关配置。

（2） 任务 2：DMA 数据非阻塞接收 LCD 显示



任务 2 主要用到的硬件配置是使用 UART1 RX DMA 和 TIM14 的 PWM14 CH1 来完成DMA数据非阻塞的接收，完成数据在开发板与PC之间的串口通信，并通过调用 LCD 相关函数（lcd_show_string()与 lcd_clear 等）将接收到的数据缓冲区（rx_buffer）的内容展示到 LCD 上。LCD 的相关设置在前几个实验中已提及过多次，在此不再展示，基本就是 FSMC 的相关配置。

（3） 任务 3：轻触 DOWN 按键红色 LED 翻转



如图所示，任务 3 所需要的硬件设计较少，仅需要打开使能 GPIO 即可，LED 与按键的调用在代码中即可即插即用。

2.2 软件设计

1. 软件设计概述

（1）任务 1：AHT21 温湿度传感器 LCD 显示

本软件主要实现基于 RT-Thread 实时操作系统的温湿度监测与显示系统，通过 AHT21 传感器实时获取环境温度和湿度数据，并将测量结果格式化后显示在 LCD 屏上，实现数据采集与图形显示的无缝衔接。系统整体采用模块化设计，结构清晰、易于维护，并具备良好的实时性和扩展性。具体设计内容如下：

【1】 系统初始化模块

在 main 函数中，系统首先调用 lcd_clear() 和 lcd_set_color() 对 LCD 屏进行初始化设置，确保显示区域背景和文字颜色正确；随后，通过 rt_thread_mdelay() 延时等待传感器稳定工作。接着，利用 aht10_init() 通过指定总线名 "i2c3" 初始化 AHT21（注：代码中的 aht10 驱动与 AHT21 工作原理相似）传感器，并对初始化返回值进行检测，以防传感器初始化失败后导致系统不可预期的后续行为。

【2】 温湿度数据采集与显示模块

系统采用轮询方式，通过循环机制持续获取温度和湿度数据。具体流程为：

- 利用 aht10_read_humidity() 读取湿度数据，将采集的数据进行格式化处理后，通过 sprintf() 生成字符串，并调用 lcd_show_string() 在指定位置显示湿度值；
- 随后，调用 aht10_read_temperature() 获取温度值，同样进行格式化后利用 lcd_show_string() 显示温度信息。

  通过上述操作，系统能够实时更新 LCD 屏幕上的温湿度数据，为用户提供直观的数据反馈。

【3】日志调试模块

为了方便测试与调试，系统在温湿度及温度数据读取后分别调用 LOG_D() 打印调试信息。这一设计不仅有助于开发过程中问题的定位，同时也为后续系统功能扩展提供了有效的运行状态监控手段。

总体而言，本软件基于 RT-Thread 及相关硬件驱动实现了一个实时、准确的温湿度显示系统，模块划分明确、逻辑严谨，为进一步功能拓展（如报警、数据记录等）奠定了良好的技术基础。

（2）任务 2：DMA 数据非阻塞接收 LCD 显示

本任务主要实现基于 RT-Thread 的串口数据非阻塞接收和 LCD 显示系统，采用 DMA 与消息队列（信号量）相结合的方式完成串口数据的异步接收，并在接收线程中将不超过 20 字节的字符串数据实时显示在 LCD 上。系统整体采用模块化设计，逻辑清晰、响应迅速，为后续功能扩展（如数据存储、复杂命令解析等）打下良好基础。具体设计内容如下：

【1】 系统初始化模块

在 main 函数中，系统首先对 LCD 进行初始化，通过 lcd_clear() 清屏，并利用 lcd_set_color() 设置显示文字及背景颜色；随后，使用 rt_mq_init() 初始化消息队列，用于在串口接收回调函数与数据处理线程之间传递接收事件及数据长度。此外，通过 rt_device_find() 查找指定名称的串口设备，并以 DMA 接收模式打开串口，为后续串口数据的异步接收做准备。

【2】串口数据异步接收及消息传递模块

系统采用 DMA 方式进行串口数据的非阻塞接收，通过注册回调函数 uart_input() 实现数据接收通知。每当串口接收到数据时，UART 中断会触发回调，在 uart_input() 中构造包含接收设备和数据长度信息的消息，并利用 rt_mq_send() 将消息发送到消息队列中。此设计确保在串口数据到达时能迅速完成消息传递，不阻塞主程序的其他操作，并通过打印接收长度进行初步调试。

【3】 串口数据处理与 LCD 显示模块

在独立的串口数据处理线程中，通过 rt_mq_recv() 不间断地从消息队列中获取接收消息。收到消息后，线程利用 rt_device_read() 根据消息中的数据长度读取串口缓冲区的数据，并以字符串形式处理。接着，通过 rt_device_write() 将接收到的数据原样回显至串口，同时调用 lcd_clear() 和 lcd_show_string() 先清屏再在指定位置显示接收到的字符串，从而实现串口数据与 LCD 显示的联动。该模块采用线程调度机制，确保数据处理及时且不会影响其它任务的运行。

总体而言，本任务利用 DMA 方式进行串口非阻塞数据接收，结合消息队列

实现数据同步传递，并通过独立线程完成数据解析与 LCD 显示，形成一个实时响应、逻辑严谨的串口数据处理与显示系统，具备良好的扩展性与调试便捷性。

（3）任务 3：轻触 DOWN 按键红色 LED 翻转

本任务主要实现基于 RT-Thread 的 PIN 设备中断控制功能，用户通过轻触按键触发外部中断，实现对红色 LED 灯的状态翻转。系统结合中断机制和定时消抖策略，确保响应灵敏且误触发率低，适用于对输入设备响应精度要求较高的场景。整体设计简洁高效，结构清晰，便于后续功能拓展。具体设计内容如下：

【1】 GPIO 初始化模块

在主函数中，系统首先通过 rt_pin_mode() 将红色 LED 对应的引脚 PF12 配置为输出模式，并将按键引脚 PC1 配置为输入上拉模式，以确保在按键松开时保持为高电平状态，从而在按键按下时产生下降沿中断。此配置确保中断触发条件稳定、逻辑状态清晰。

【2】 按键中断与回调处理模块

系统通过 rt_pin_attach_irq() 将按键 PC1 的下降沿事件绑定至自定义回调函数 btn_irq_callback()，并通过 rt_pin_irq_enable() 启用该中断。按键每次被按下即触发中断，调用对应回调函数处理逻辑。

为避免按键物理抖动导致的重复触发，回调函数内添加了基于系统节拍 rt_tick_get() 的软件消抖机制，限定两次有效中断的最小间隔为 200ms，从而有效抑制误触发，提高系统稳定性。

【3】LED 控制逻辑模块

在中断回调函数中，首先通过 rt_pin_read() 读取当前红色 LED 的电平状态，随后进行状态翻转：若当前为低电平（LED 灭），则通过 rt_pin_write() 设置为高电平（LED 亮），反之亦然。此逻辑保证每次按键按下均可完成 LED 亮灭状态的切换，操作直观，反馈明确。

总体而言，本任务充分利用 RT-Thread 提供的 PIN 中断机制与系统节拍管理功能，实现了一个高效、抗干扰能力强的 LED 控制应用，具有良好的实用性和可维护性，适合作为嵌入式项目中外部事件响应与控制的典型参考范式。

2.软件流程分解

（1）任务 1：AHT21 温湿度传感器 LCD 显示

A. 初始阶段

开始 → 系统初始化

程序启动后，完成以下初始化操作：

- **LCD 显示初始化**：调用 lcd_clear(WHITE) 清屏，并通过 lcd_set_color(WHITE, BLACK) 设置背景与字体颜色；
- **延时等待传感器上电**：调用 rt_thread_mdelay(2000) 延时 2 秒，确保 AHT10 传感器正常启动；
- **I2C 总线配置**：指定 AHT10 使用的 I2C 总线为 "i2c3"；
- **AHT10 传感器初始化**：调用 aht10_init() 初始化温湿度传感器，获取传感器设备句柄；
    - 若初始化失败（返回 RT_NULL），输出错误日志并退出程序；
    - 若成功，进入数据采集循环。

B. 主循环结构

初始化完成 → 进入主循环

在主循环中，程序连续运行 100 次，定期采集温湿度信息并进行数据显示：

- 每轮循环间隔 300ms（rt_thread_mdelay(300)）；
- 每轮循环内完成一次完整的温湿度读取与 LCD 显示。

C. 数据采集与显示流程

每次循环内执行以下流程：

- **湿度读取与显示：**
    - 使用 aht10_read_humidity() 获取当前湿度值；
    - 使用 LOG_D 打印湿度调试信息；
    - 使用 sprintf() 格式化湿度字符串（如 "Humidity: 54.2%"）；
    - 调用 lcd_show_string() 在指定位置显示湿度值。
- **温度读取与显示：**
    - 使用 aht10_read_temperature() 获取当前温度值；
    - 使用 LOG_D 打印温度调试信息；

- 使用 sprintf() 格式化温度字符串（如 "Temperature: 25.7C"）；
- 调用 lcd_show_string() 在指定位置显示温度值。

D. 循环机制

系统通过固定间隔定时采样，每次读取温湿度后立即在 LCD 上更新显示内容，连续执行 100 轮后自动退出。整体流程清晰、响应及时，适用于基本环境监测应用。

（2）任务 2：DMA 数据非阻塞接收 LCD 显示

A. 初始阶段

开始 → 系统初始化

程序启动后，完成以下初始化步骤：

- **LCD 显示初始化**：调用 lcd_clear(WHITE) 清屏，并通过 lcd_set_color(WHITE, BLACK) 设置背景与字体颜色；
- **消息队列初始化**：使用 rt_mq_init() 创建名为 "rx_mq" 的消息队列，用于接收串口数据通知，分配固定长度的缓冲区 msg_pool[256]；
- **串口设备查找与打开**：
  - 通过 rt_device_find("uart1") 查找串口设备；
  - 若查找失败，输出提示信息并退出；
  - 若成功，使用 rt_device_open() 以 DMA 接收模式打开串口设备；
- **设置串口接收回调函数**：通过 rt_device_set_rx_indicate() 将串口数据到达的处理函数设置为 uart_input()；
- **串口初始化测试输出**：通过 rt_device_write() 发送测试字符串 "hello RT-Thread!\r\n"；
- **创建并启动串口数据处理线程**：调用 rt_thread_create() 创建串口处理线程 serial_thread_entry，并启动线程。

B. 主线程结构

初始化完成 → 主线程退出（不含主循环）

主线程在完成串口和线程的初始化工作后直接返回。数据接收与处理逻辑完全由接收中断和后台线程负责，主线程不再参与后续业务处理。

C. UART 接收回调流程

串口收到数据 → 触发回调函数 uart_input()

- 当串口 uart1 接收到数据时，系统自动调用回调函数 uart_input()；
- 在回调中：
  - 构造接收消息结构体 rx_msg，记录设备句柄及接收长度；
  - 将消息发送至 "rx_mq" 消息队列中；
  - 若队列已满，则提示 "message queue full!"，丢弃本次数据；
  - 打印本次接收的数据长度作为调试输出。

D. 串口数据处理线程流程

后台线程循环处理串口消息队列：

- 线程函数 serial_thread_entry() 内部为无限循环结构；
- 每次循环中：
  - 从消息队列 rx_mq 中阻塞式读取消息；
  - 读取指定长度的串口数据至缓冲区 rx_buffer，并添加结束符 '\0'；
  - 将接收内容通过串口回写至终端；
  - 同时在 LCD 上显示接收到的字符串，更新界面信息（使用 lcd_show_string()）。

E. 循环机制

串口接收 → 回调函数通知 → 消息队列中转 → 后台线程处理

系统以串口接收中断为入口，结合消息队列与线程调度机制，形成一套异步、解耦的数据接收与显示流程。主线程退出后，所有逻辑均通过回调与线程协作完成，实现稳定、高效的串口通信与 LCD 显示联动。

（3）任务 3：轻触 DOWN 按键红色 LED 翻转

A. 初始阶段

开始 → 系统初始化

程序启动后执行以下初始化步骤：

- **配置 LED 引脚模式：**
  - 调用 rt_pin_mode(PIN_LED_R, PIN_MODE_OUTPUT) 设置红色 LED 引脚（PF12）为输出模式；
- **配置按键引脚模式：**

- 调用 rt_pin_mode(PIN_BTN_DOWN, PIN_MODE_INPUT_PULLUP) 将按键引脚（PC1）配置为输入模式并开启上拉电阻，以确保未按下时为高电平；

- **中断绑定与使能：**
  - 使用 rt_pin_attach_irq() 将按键下降沿中断事件绑定到回调函数 btn_irq_callback；
  - 使用 rt_pin_irq_enable() 使能按键中断功能；
- **打印调试信息：**
  - 使用 LOG_D() 输出初始化完成信息，提示用户等待中断触发。

B. 主线程结构

主线程进入循环 → 保持运行状态

- 系统初始化完成后，主线程进入 while (1) 无限循环；
- 循环中调用 rt_thread_mdelay(50) 进行短暂延时；
- 主线程不承担具体业务处理，仅维持系统运行与节奏控制，可预留位置用于后续扩展其他任务模块。

C. 中断回调流程

用户按下按键 → 触发中断 → 执行 btn_irq_callback() 回调函数

- 中断触发入口为 **按键下降沿**，即按下瞬间；
- 回调函数执行流程如下：

a) **消抖处理：**
   - 利用静态变量 last_tick 记录上一次中断触发时系统节拍；
   - 调用 rt_tick_get() 获取当前节拍；
   - 若本次中断距离上次不足 200ms，则认为是抖动，直接返回；
   - 否则更新 last_tick 为当前时间，继续执行后续逻辑；

b) **读取 LED 当前状态：**
   - 使用 rt_pin_read(PIN_LED_R) 获取红色 LED 当前电平状态；
   - 输出调试日志，记录当前 LED 状态；

c) **翻转 LED 状态：**
   - 若当前为低电平（熄灭），则写入高电平（点亮）；

- 若当前为高电平（点亮），则写入低电平（熄灭）；
- 使用 rt_pin_write() 完成状态翻转操作。

D. 硬件交互机制

- **按键与中断机制**：利用 GPIO 下降沿触发中断，系统响应迅速；
- **LED 状态控制**：直接通过引脚电平高低控制红色 LED 开关状态；
- **消抖逻辑嵌入回调中**：无需额外定时器，使用系统节拍进行简洁有效的软消抖处理。

E. 循环与中断协同机制

系统采用 **主循环 + 中断回调协同机制**：

- 主线程保持运行状态，但不承担控制逻辑；
- 按键操作完全由中断驱动，确保实时性与响应性；
- 系统节拍消抖提高稳定性，避免误触发。

3. μ vision 详细代码

（1）任务 1：AHT21 温湿度传感器 LCD 显示

```
/*
 * Copyright (c) 2006-2021, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author       Notes
 * 2023-5-10      ShiHao       first version
 */

#include <rtthread.h>
#include <rtdevice.h>
#include <board.h>
#include "aht10.h"
#include "drv_lcd.h"
```

```
#define DBG_TAG "main"
#define DBG_LVL DBG_LOG
#include <rtdbg.h>

int main(void)
{
    lcd_clear(WHITE);
    lcd_set_color(WHITE,BLACK);
    float humidity, temperature;
    aht10_device_t dev;

    /* 总线名称 */
    const char *i2c_bus_name = "i2c3";
    int count = 0;

    /* 等待传感器正常工作 */
    rt_thread_mdelay(2000);

    /* 初始化 aht10 */
    dev = aht10_init(i2c_bus_name);
    if (dev == RT_NULL)
    {
        LOG_E(" The sensor initializes failure");
        return 0;
    }

    while (count++ < 100)
    {
        /* 读取湿度 */
        humidity = aht10_read_humidity(dev);
```

```
            LOG_D("humidity: %d.%d %%", (int)humidity, (int)(humidity * 10) % 10);

            // 格式化湿度信息为字符串
            char humidity_str[16];
            sprintf(humidity_str, "Humidity: %d.%d%%", (int)humidity, (int)(humidity
* 10) % 10);
            lcd_show_string(10, 100, 24, humidity_str);

            /* 读取温度 */
            temperature = aht10_read_temperature(dev);
            LOG_D("temperature: %d.%d", (int)temperature, (int)(temperature * 10) %
10);

            char temp_str[16];
            sprintf(temp_str, "Temperature: %d.%dC", (int)temperature,
(int)(temperature * 10) % 10);
            lcd_show_string(10, 50, 24, temp_str);

            rt_thread_mdelay(300);
        }
        return 0;
}
```

（2）任务 2：DMA 数据非阻塞接收 LCD 显示

```
/*
 * Copyright (c) 2006-2021, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author       Notes
```

```
 * 2023-5-10           ShiHao             first version
 */

#include <rtthread.h>
#include <rtdevice.h>
#include <board.h>
#include "drv_lcd.h"

#define DBG_TAG "main"
#define DBG_LVL               DBG_LOG
#include <rtdbg.h>

#define SAMPLE_UART_NAME "uart1" /*  串口设备名称*/
static rt_device_t serial;             /*  串口设备句柄*/

/*  串口接收消息结构*/
struct rx_msg
{
    rt_device_t dev;
    rt_size_t size;
};

/*  消息队列控制块*/
static struct rt_messagequeue rx_mq;

/*  接收数据回调函数  */
static rt_err_t uart_input(rt_device_t dev, rt_size_t size)
{
    struct rx_msg msg;
    rt_err_t result;
```

```
    msg.dev = dev;
    msg.size = size;
    result = rt_mq_send(&rx_mq, &msg, sizeof(msg));

        rt_kprintf("\nrx len = %d\n",size);
    if (result == -RT_EFULL)
    {
        /* 消息队列满*/
        rt_kprintf("message queue full！\n");
    }
    return result;
}
/* 串口接收数据处理线程 */
static void serial_thread_entry(void *parameter)
{
    struct rx_msg msg;
    rt_err_t result;
    rt_uint32_t rx_length;
    static char rx_buffer[RT_SERIAL_RB_BUFSZ + 1];
    while (1)
    {
        rt_memset(&msg, 0, sizeof(msg));
        /* 从消息队列中读取消息*/
        result = rt_mq_recv(&rx_mq, &msg, sizeof(msg),
RT_WAITING_FOREVER);
        if (result == RT_EOK)
        {
            /* 从串口读取数据*/
            rx_length = rt_device_read(msg.dev, 0, rx_buffer, msg.size);
            rx_buffer[rx_length] = '\0';
```

```
                /* 通过串口设备 serial 输出读取到的消息*/
                rt_device_write(serial, 0, rx_buffer, rx_length);
                lcd_clear(WHITE);
                lcd_set_color(WHITE,BLACK);
                lcd_show_string(10, 80, 24, rx_buffer);
            }
        }
}

/* 用户线程入口函数 */
int main(void)
{
    lcd_clear(WHITE);
    lcd_set_color(WHITE,BLACK);
    rt_err_t ret = RT_EOK;
    static char msg_pool[256];
    char str[] = "hello RT-Thread!\r\n";

    /* 初始化消息队列*/
    rt_mq_init(&rx_mq, "rx_mq",
                msg_pool, /* 存放消息的缓冲区*/
                sizeof(struct rx_msg), /* 一条消息的最大长度*/
                sizeof(msg_pool),  /* 存放消息的缓冲区大小*/
                RT_IPC_FLAG_FIFO); /* 如果有多个线程等待， 按照先来先得到的方法分配消息*/

    /* 查找串口设备 */
    serial = rt_device_find(SAMPLE_UART_NAME);
    if (!serial)
    {
```

```
        rt_kprintf("find %s failed!\n", SAMPLE_UART_NAME);
        return RT_ERROR;
    }

    rt_device_open(serial, RT_DEVICE_FLAG_DMA_RX);
    /* 设置接收回调函数*/
    rt_device_set_rx_indicate(serial, uart_input);
    /* 发送字符串*/
    rt_device_write(serial, 0, str, (sizeof(str) - 1));
    /* 创建 serial 线程*/
    rt_thread_t thread = rt_thread_create("serial", serial_thread_entry, RT_NULL,
                                          1024, 25, 10);
    /* 创建成功则启动线程*/
    if (thread != RT_NULL)
    {
        rt_thread_startup(thread);
    }
    else
    {
        ret = RT_ERROR;
    }
    return ret;
}
```

（3）任务 3：轻触 DOWN 按键红色 LED 翻转

```
/*
 * Copyright (c) 2006-2021, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
```

```
 * Date                 Author          Notes
 * 2023-5-10           ShiHao          first version
 * 2025-04-11          YourName        添加按键中断翻转红 LED，并添加中断消抖功能
 */

#include <rtthread.h>
#include <rtdevice.h>
#include <board.h>

#define DBG_TAG "main"
#define DBG_LVL               DBG_LOG
#include <rtdbg.h>

/* 定义 LED 和按键对应的引脚 */
#define PIN_LED_R                   GET_PIN(F, 12)
#define PIN_BTN_DOWN                GET_PIN(C, 1)

/* 中断回调函数，实现 LED 状态翻转，并增加消抖处理 */
static void btn_irq_callback(void *args)
{
    /* 静态变量记录上次中断触发时的系统节拍 */
    static rt_tick_t last_tick = 0;
    rt_tick_t cur_tick = rt_tick_get();

    /* 消抖：两次中断触发的间隔必须大于或等于 50ms */
    if ((cur_tick - last_tick) < rt_tick_from_millisecond(200))
    {
        return;
    }
```

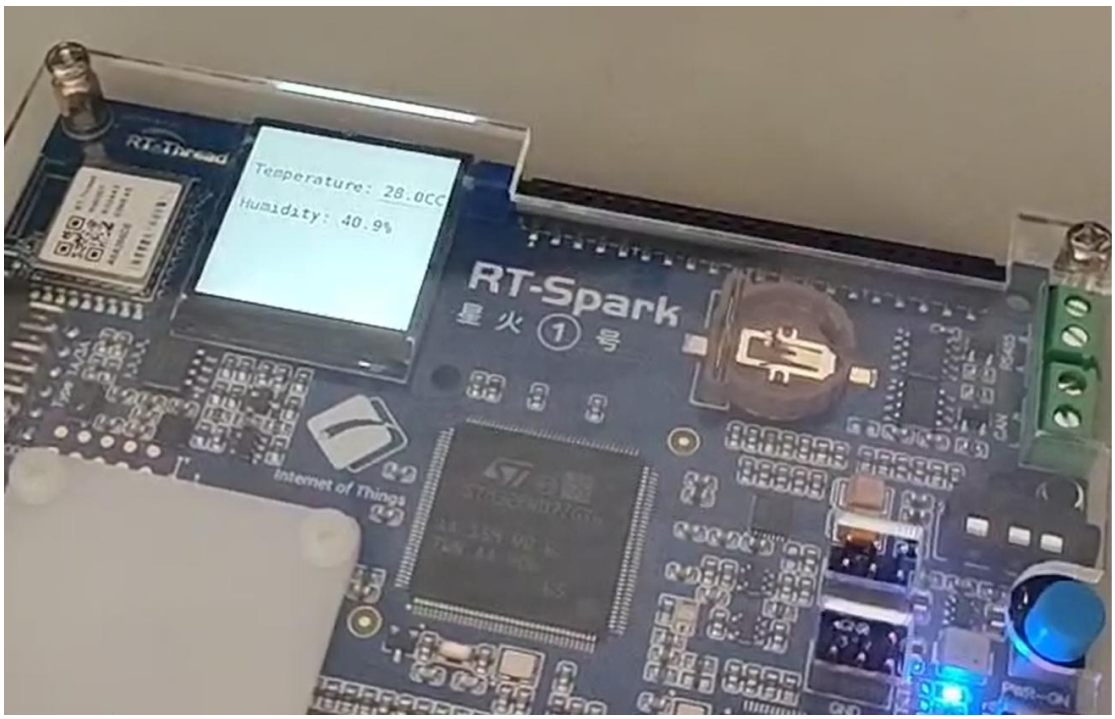
```
    last_tick = cur_tick;

    int state;

    /* 读取当前 LED 状态 */
    state = rt_pin_read(PIN_LED_R);
    LOG_D("按键触发中断，当前 LED 状态：%d", state);

    /* 翻转 LED 状态 */
    if (state == PIN_LOW)
    {
        rt_pin_write(PIN_LED_R, PIN_HIGH);
    }
    else
    {
        rt_pin_write(PIN_LED_R, PIN_LOW);
    }
}

int main(void)
{
    /* 配置 LED 引脚为输出模式 */
    rt_pin_mode(PIN_LED_R, PIN_MODE_OUTPUT);
    /* 配置按键引脚为输入模式，并开启上拉（假设按键按下时为低电平） */
    rt_pin_mode(PIN_BTN_DOWN, PIN_MODE_INPUT_PULLUP);

    /* 绑定中断回调函数，使用下降沿触发 */
    rt_pin_attach_irq(PIN_BTN_DOWN,                PIN_IRQ_MODE_FALLING,
btn_irq_callback, RT_NULL);
```

```
    /* 使能按键中断 */
    rt_pin_irq_enable(PIN_BTN_DOWN, PIN_IRQ_ENABLE);

    LOG_D("系统初始化完成，等待按键触发中断…");

    /* 主线程循环，可以在此添加其他任务 */
    while (1)
    {
        rt_thread_mdelay(50);
    }

    return 0;
}
```

3 实验结果分析与总结

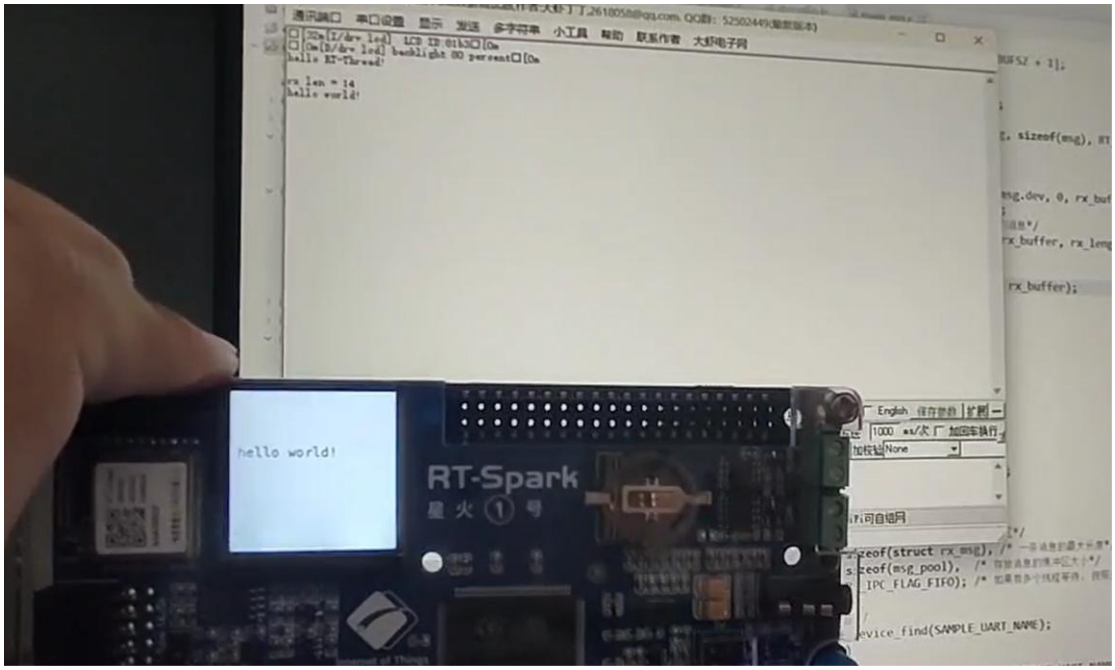
（1）任务 1：AHT21 温湿度传感器 LCD 显示

如图所示，本次任务 1 的代码工程能够很好的结合 AHT21 温湿度传感器与 LCD，完成实验要求，具体的内容见附带的视频。

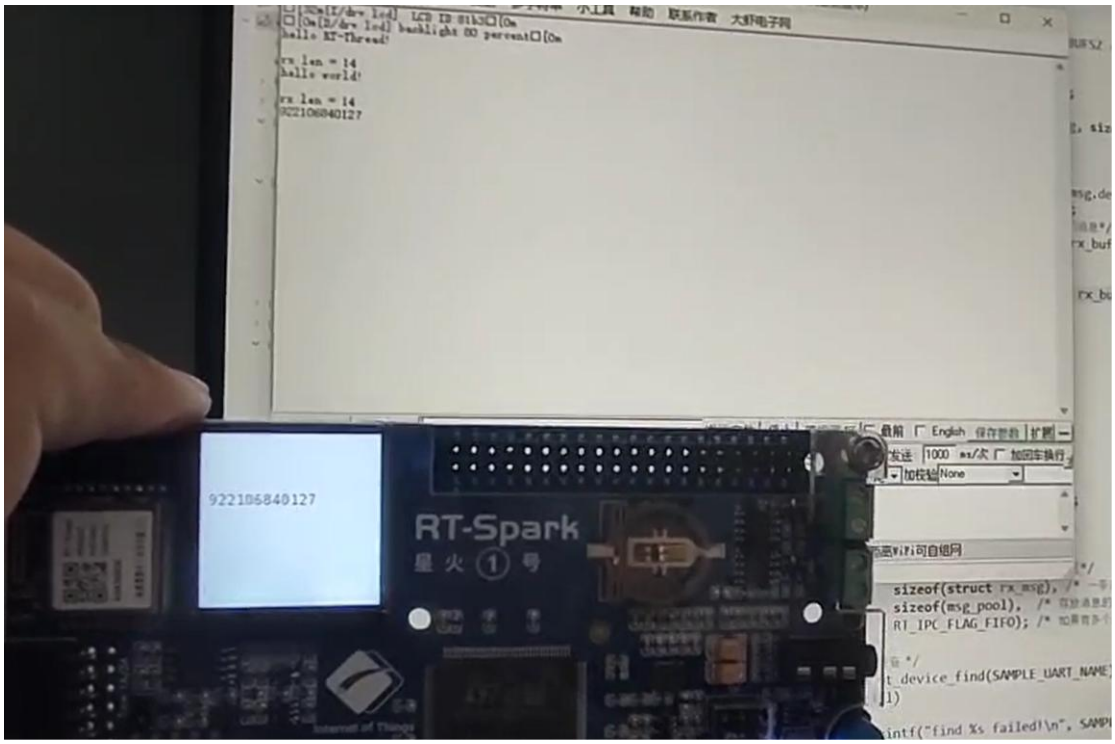


（2）任务 2：DMA 数据非阻塞接收 LCD 显示

如图所示，本次任务 2 的代码工程能很好的结合 DMA 方式消息队列的接收并且结合 LCD 驱动将 PC 串口通信发送的文本内容显示到 LCD 上，具体的内容见附带的视频。



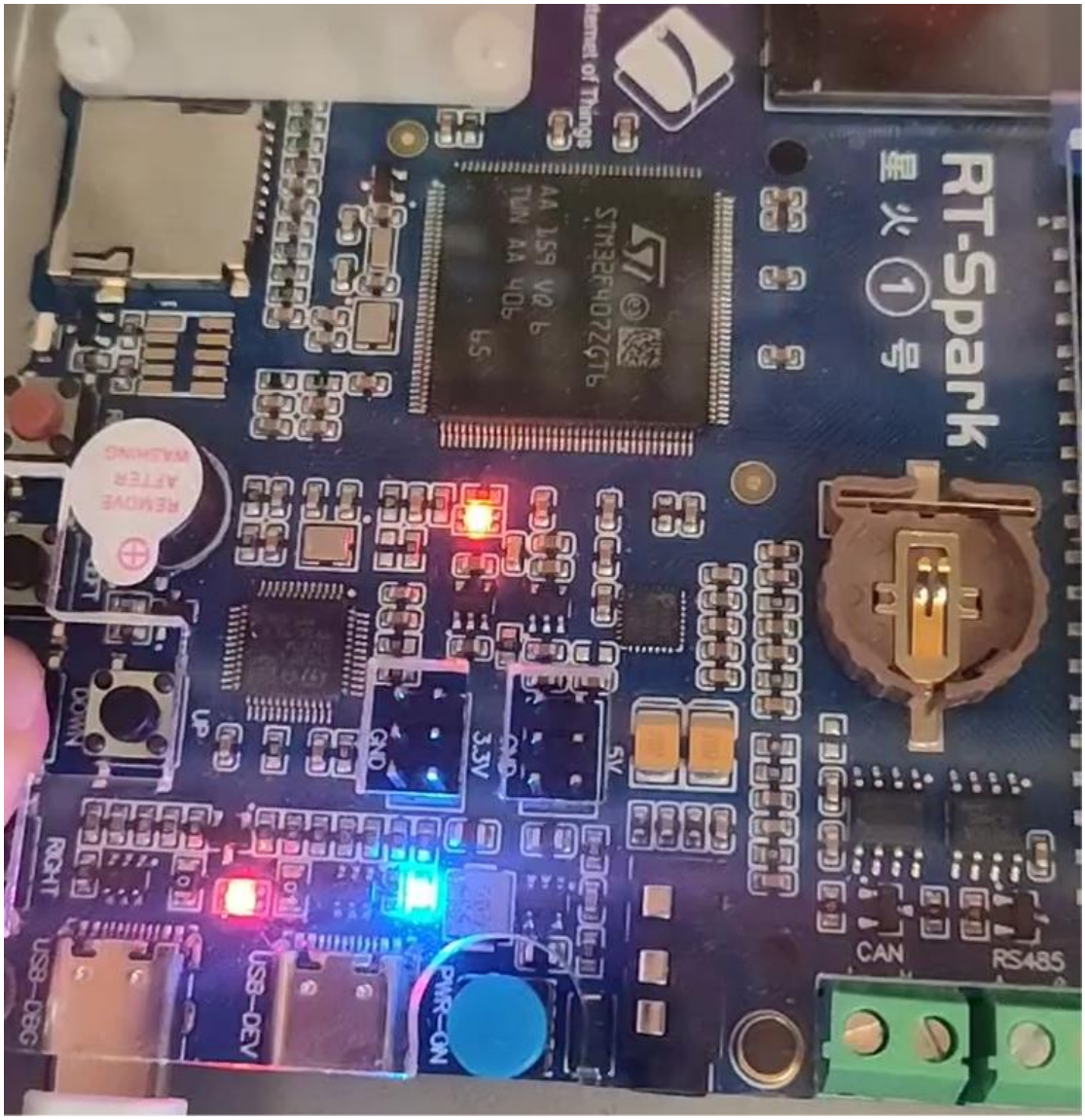
PC 输入 hello world!



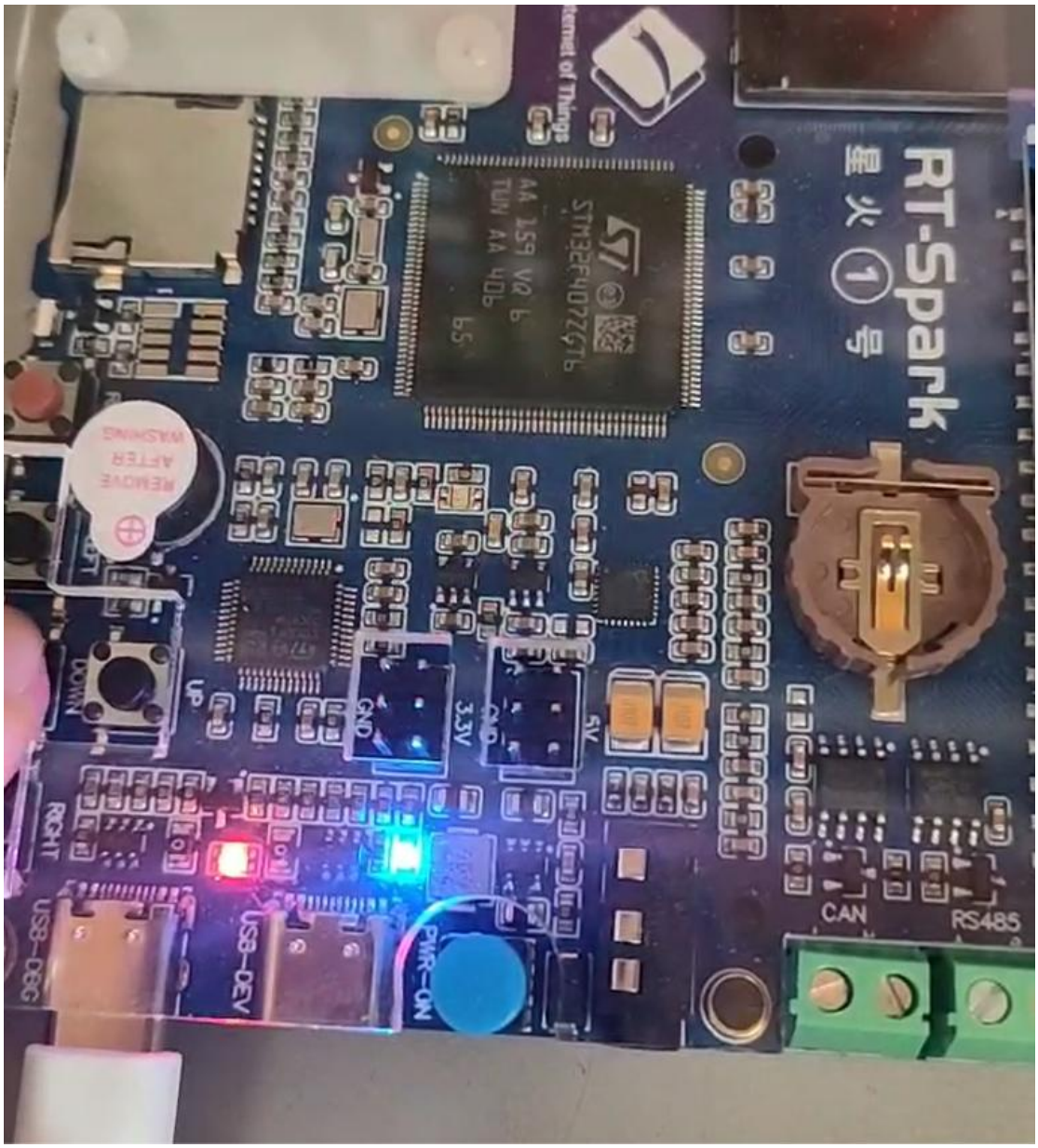
PC 输入 922106840127

（3）任务 3：轻触 DOWN 按键红色 LED 翻转

如图所示，本次任务 3 的代码工程能够很好的结合 GPIO 与中断控制相关的技术，实现了消抖功能，完成了翻转 LED 的实验要求，具体的内容见附带的视频。



开发板启动时默认红色 LED 开



按一下 DOWN 按键，LED 状态翻转，LED 灭