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：ex1_922106840127_GPIO

学号：922106840127

姓名：刘宇翔

成绩

日期：　　2025 年　　4 月　　7 日

1 题目要求

1. 题目设计要求

【1】作业内容

设计一个输入输出的项目实例，使用 BTN LEFT 按键翻转红色 LED R 灯、BTN RIGHT 按键翻转绿色 LEDB 灯，按下 BTN UP 按键发出 5s 救护车报警声,按下 BTN DOWN 键按发出 5s 电动车报警声，按键带有消抖功能。

【2】完成要求：

（1）工程名称命名:ex1_学号_GPIO，并打包成:ex1_学号_GPIO.rar 压缩文件夹

（2）实验报告 PDF 格式:ex1_学号_GPIO.pdf

2. 拟实现的具体功能

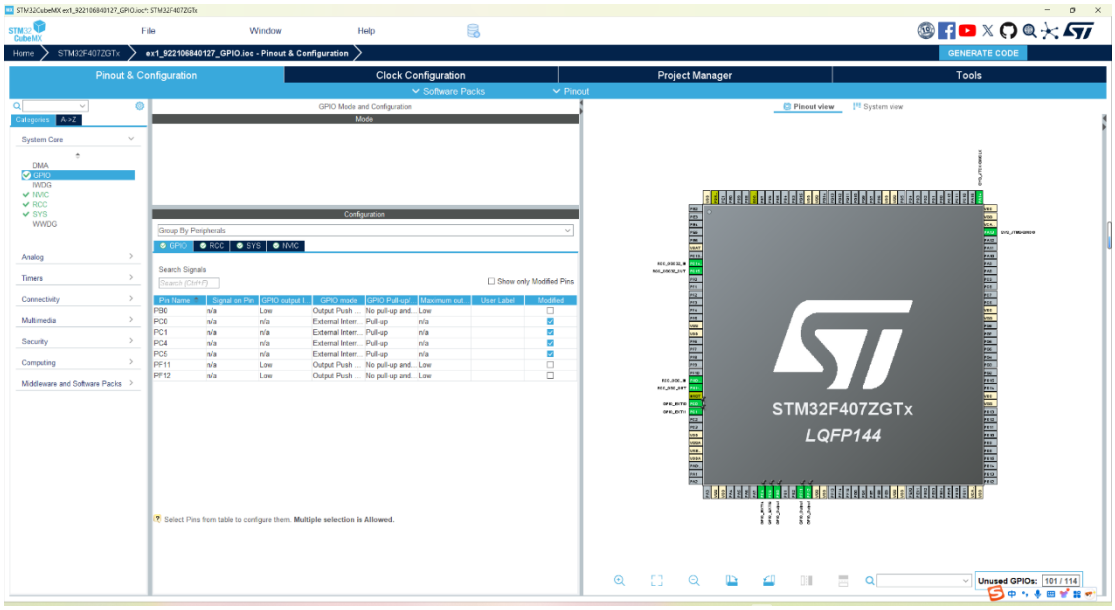
完成基于 STM32CubeMX 硬件设计平台实现的基于方向键加以中断实现对应的 LED 灯输出，使用 BTN_LEFT 按键翻转红色 LED_R 灯、 BTN_RIGHT 按键翻转蓝色 LED_B 灯，按下 BTN_UP 按键发出 5s 救护车报警声，按下 BTN_DOWN 键按发出 5s 电动车报警声。

2 总体设计

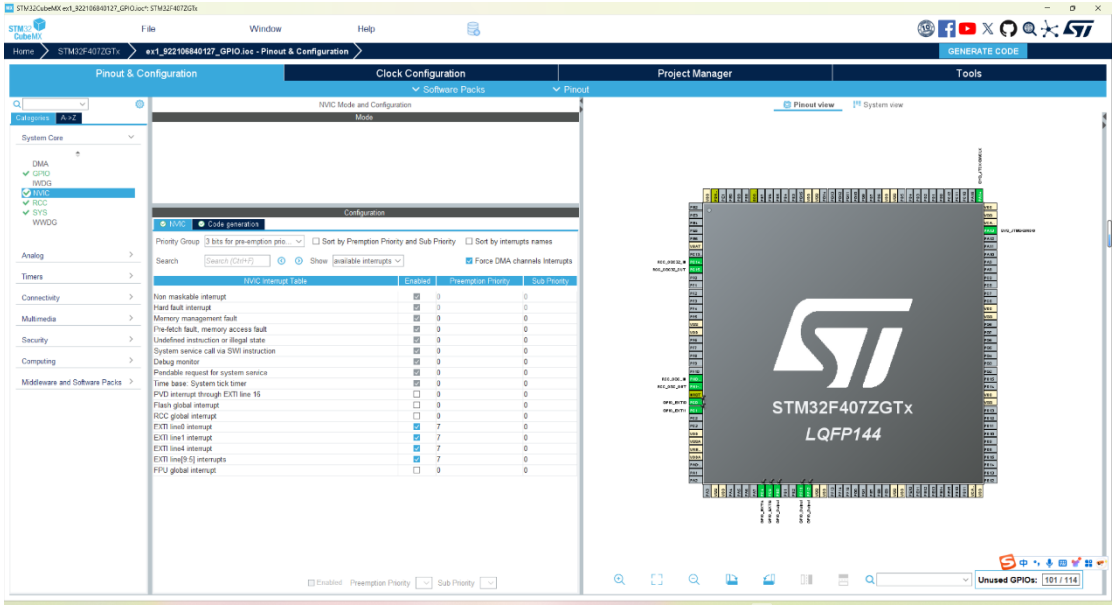
2.1 硬件设计

1. 硬件设计思路

我通过查阅相关开发板原理图确定了本芯片开发板的四个方向键对应的引脚以及两个 LED 灯与开发板实现声音信号输出的相关引脚，具体的代码工程配置硬件设计如下：



如图所示，我成功设置好了对应的引脚，其中四个方向键设置为下降沿触发，即为当按键被按下时实现函数中断相关的内容。



如图所示，我在 NVIC 的相关设置界面对几种中断的相关属性做了具体的规定。其他设置与前几次相同，如 RCC 采用 Crystal/Ceramic Resonator 作为时钟设置，使用 Serial Wire 用于调试对应的 debug 接口，对 Clock Configuration 的时钟配置也做好了相应的规范化，并且在 Project Manager 中选择了 MDK-ARM V5.32 作为编译工具链，其他内容此处省略。

以上配置图是我作为 STM32CUBEMX 进行的配置设置，设置后点击“Generate Code”生成初始化代码。

2.2 软件设计

1. 软件设计概述

本软件主要实现基于外部中断的输入响应功能，通过按键触发控制 LED 状态翻转和报警声音的输出。系统基于 STM32 单片机，利用 HAL 库进行外设初始化，并采用中断机制实现对按键事件的实时响应，整体设计结构清晰，功能模块划分明确。具体设计内容如下：

（1） 系统初始化模块

在 main 函数中，系统首先通过 HAL_Init()完成底层硬件的复位及初始化，然后调用 SystemClock_Config()对系统时钟和 PLL 参数进行配置，确保 MCU 及外设在预定频率下稳定运行；接着通过 MX_GPIO_Init()初始化 GPIO 外设，为后续的按键检测、LED 控制以及报警声音输出提供硬件支持。

（2）外部中断处理模块

通过外部中断技术实现对四个按键（BTN_LEFT、BTN_RIGHT、BTN_UP 和 BTN_DOWN）的实时检测。在 HAL_GPIO_EXTI_Callback 中，通过判断中断触发的 GPIO 引脚，实现不同按键的功能响应：

- 当检测到对应 BTN_LEFT 的中断（GPIO_PIN_0）时，调用 HAL_GPIO_TogglePin 函数翻转红色 LED_R 状态；
- 当检测到对应 BTN_RIGHT 的中断（GPIO_PIN_4）时，翻转蓝色 LED_B 状态；
- 当检测到对应 BTN_DOWN 的中断（GPIO_PIN_1）时，通过循环调用 sound2 函数产生约 5 秒的电动车报警声；
- 当检测到对应 BTN_UP 的中断（GPIO_PIN_5）时，调用 sound1 函数两次以生成 5 秒的救护车报警声。

在中断服务中还采用了短暂延时（HAL_Delay(15)）来实现简单的按键消抖，确保信号稳定。

（2） 延时模块

设计了基于空循环的延时函数 delay，该函数用于在声音输出函数中产生所需的时间延迟，同时也辅助中断服务中简单的去抖处理，从而保证按键触发后各模块响应的稳定性。

（4）声音输出模块

软件设计了两个声音函数以模拟不同报警器的声音特效：

• sound1 函数用于产生救护车报警声。该函数通过不断地控制指定引脚的电平（先置低电平，再置高电平）并逐步减少延时参数，模拟出变化的音调；

• sound2 函数用于产生电动车报警声，其原理与 sound1 类似，但初始延时和变化速率不同，以达到不同的声音效果。

在中断回调中，根据按键不同，通过循环调用相应的声音函数实现 5 秒左右的报警声输出。

总体而言，本软件在有限的硬件资源下，通过系统初始化、外部中断响应、延时消抖及声音输出四大模块协同工作，实现了对输入按键的精准检测和多种外部信号（LED 指示和报警声）的实时反馈，确保系统具有较高的实时性和可靠性。

2. 软件流程图

A. 初始阶段

开始 → 系统初始化

程序启动后首先完成系统硬件初始化，包括：

- 系统时钟配置（通过 SystemClock_Config() 确保 MCU 及外设稳定运行）
- GPIO 端口初始化（MX_GPIO_Init() 配置各引脚用于按键、LED 及蜂鸣器控制）
- 外部中断设置（配置各按键对应的中断线）

B. 主循环结构

进入主循环 → 等待外部中断触发

由于本系统采用外部中断响应机制，主循环内不执行其他任务，系统处于等待状态，直到有按键事件发生。

C. 外部中断触发流程

按键按下 → 外部中断产生

当用户按下任一按键（BTN_LEFT、BTN_RIGHT、BTN_UP 或 BTN_DOWN）时，相应的 GPIO 引脚电平发生变化，触发外部中断，系统自动进入中断服务函数（HAL_GPIO_EXTI_Callback）。

D. 按键检测与消抖流程

进入中断服务函数 → 软件延时去抖 → 检测按键电平状态

在中断服务函数中，首先执行短暂延时（例如 HAL_Delay(15)），以消除机械按键抖动带来的误触发，再次检测按键状态，确保信号稳定后继续执行相应操作。

E. 数据处理与执行阶段

根据检测到的按键确定相应响应：

- **BTN_LEFT（GPIO_PIN_0）**：翻转红色 LED_R 状态
- **BTN_RIGHT（GPIO_PIN_4）**：翻转蓝色 LED_B 状态
- **BTN_UP（GPIO_PIN_5）**：调用 sound1() 函数两次，通过循环方式产生约 5 秒救护车报警声
- **BTN_DOWN（GPIO_PIN_1）**：循环调用 sound2() 函数，通过循环方式产生约 5 秒电动车报警声

在每个操作结束后，通过适当控制蜂鸣器（关闭声音输出）确保操作完整。

F. 循环机制

操作执行完毕 → 退出中断服务函数 → 返回主循环

中断服务函数执行完毕后，系统返回主循环，继续等待下一次外部中断触发，实现对按键操作的持续响应。

3. μ vision 详细代码

由于 μ vision IDE 不支持中文字符显示，故使用英文在关键函数处写了相关注释，完成了对应的实验内容。

```
#ifndef __MAIN_H
#define __MAIN_H

#include "stm32f4xx_hal.h"

// Pin definitions for buttons, LEDs, and buzzer
#define BTN_LEFT_PIN            GPIO_PIN_0
#define BTN_RIGHT_PIN           GPIO_PIN_4
#define BTN_UP_PIN              GPIO_PIN_5
#define BTN_DOWN_PIN            GPIO_PIN_1
```

```
#define LED_R_PIN                 GPIO_PIN_12
#define LED_B_PIN                 GPIO_PIN_11
#define BUZZER_PIN                GPIO_PIN_0

// Port definitions for buttons, LEDs, and buzzer
#define BTN_GPIO_PORT             GPIOC
#define LED_GPIO_PORT             GPIOF
#define BUZZER_PORT               GPIOB

// Frequency definitions (used in sound generation)
#define SIREN_FREQ_1              800
#define SIREN_FREQ_2              1200
#define EV_FREQ                   1000

// Alarm duration in milliseconds
#define ALARM_DURATION            5000

#endif /* __MAIN_H */

#include "main.h"
#include "gpio.h"

// Function prototypes
void SystemClock_Config(void);

volatile uint8_t btn_left_flag = 0;
volatile uint8_t btn_right_flag = 0;
volatile uint8_t btn_up_flag = 0;
volatile uint8_t btn_down_flag = 0;
```

```
void sound1(void);
void sound2(void);
void delay(uint32_t);

//-------------------------------------------------------------------------
// @brief    Simple software delay loop
// @param    i: Number of iterations for delay
// @note     This delay is not precise and is dependent on the system clock.
//-------------------------------------------------------------------------
void delay(uint32_t i) {
    while(i--);
}

//-------------------------------------------------------------------------
// @brief    Generate a dynamic sound pattern using a buzzer.
// @note     The function toggles the buzzer pin with variable delay to produce a varying tone.
//           The tone changes frequency by decrementing the delay value (i) gradually.
//           It runs for 5000 milliseconds (ALARM_DURATION).
//-------------------------------------------------------------------------
void sound1(void)
{
    uint32_t i = 30000;                                  // Initial delay value for sound frequency control
    uint32_t sound1_start = HAL_GetTick();           // Record start time
    while((HAL_GetTick() - sound1_start) <= 5000) // Loop for 5000 ms
    {
        HAL_GPIO_WritePin(BUZZER_PORT,                BUZZER_PIN, GPIO_PIN_RESET); // Activate buzzer (start tone)
        delay(i);                                                                  //
```

```
Delay controls tone duration
        HAL_GPIO_WritePin(BUZZER_PORT, BUZZER_PIN, GPIO_PIN_SET);
// Deactivate buzzer (stop tone)
        delay(i);
        i = i - 6;                                  // Gradually decrease delay to change
frequency
        if(i <= 0)
            i = 30000;                              // Reset delay value if it becomes
too small
    }
    HAL_GPIO_WritePin(BUZZER_PORT, BUZZER_PIN, GPIO_PIN_RESET); //
Ensure buzzer is off after sound
}

//-----------------------------------------------------------------------
// @brief    Generate another dynamic sound pattern using a buzzer.
// @note     Similar to sound1(), but with different initial delay value, resulting in a
different tone.
//           Runs for 5000 milliseconds.
//-----------------------------------------------------------------------
void sound2(void)
{
    uint32_t i = 6000;                              // Initial delay value for sound
frequency control
    uint32_t sound2_start = HAL_GetTick();          // Record start time
    while((HAL_GetTick() - sound2_start) <= 5000) // Loop for 5000 ms
    {
        HAL_GPIO_WritePin(BUZZER_PORT,                  BUZZER_PIN,
GPIO_PIN_RESET); // Activate buzzer
        delay(i);                                                          //
```

```
Delay to control tone frequency
        HAL_GPIO_WritePin(BUZZER_PORT, BUZZER_PIN, GPIO_PIN_SET);
// Deactivate buzzer
        delay(i);
        i = i - 6;                              // Decrease delay to modify the tone
frequency gradually
        if(i <= 0)
            i = 6000;                           // Reset delay value when it reaches
zero
    }
    HAL_GPIO_WritePin(BUZZER_PORT, BUZZER_PIN, GPIO_PIN_RESET); //
Ensure buzzer is off after sound
}

//-----------------------------------------------------------------------
// @brief    Simulate an ambulance siren using the buzzer.
// @note     This function toggles the buzzer with specific delays to mimic an
ambulance siren sound.
//-----------------------------------------------------------------------
void AmbulanceSiren(void) {
    for (int i = 0; i < 40; i++) {
        HAL_GPIO_WritePin(BUZZER_PORT, BUZZER_PIN, GPIO_PIN_SET);
// Activate buzzer
        HAL_Delay(30);                                                  //
Delay for 30 ms
        HAL_GPIO_WritePin(BUZZER_PORT,              BUZZER_PIN,
GPIO_PIN_RESET);// Deactivate buzzer
        HAL_Delay(30);                                                  //
Delay for 30 ms
```

```
            HAL_GPIO_WritePin(BUZZER_PORT, BUZZER_PIN, GPIO_PIN_SET);
// Activate buzzer again
            HAL_Delay(20);                                                  //
Shorter delay of 20 ms
            HAL_GPIO_WritePin(BUZZER_PORT,                BUZZER_PIN,
GPIO_PIN_RESET);// Deactivate buzzer
            HAL_Delay(20);                                                  //
Shorter delay of 20 ms
        }
}

//-----------------------------------------------------------------------------
// @brief    Simulate an electric siren using the buzzer.
// @note     This function toggles the buzzer with equal delays to create a periodic on-
off sound.
//-----------------------------------------------------------------------------
void ElectricSiren(void) {
    for (int i = 0; i < 20; i++) {
            HAL_GPIO_WritePin(BUZZER_PORT, BUZZER_PIN, GPIO_PIN_SET);
// Turn on buzzer
            HAL_Delay(100);                                                 //
Keep buzzer on for 100 ms
            HAL_GPIO_WritePin(BUZZER_PORT,                BUZZER_PIN,
GPIO_PIN_RESET);// Turn off buzzer
            HAL_Delay(100);                                                 //
Delay for 100 ms before next cycle
        }
}

//-----------------------------------------------------------------------------
```

```
// @brief    Main program entry point.
// @note     Initializes the HAL, system clock, and GPIOs. In the main loop, it checks
for button flags
//                and toggles LEDs or plays sounds accordingly.
//-----------------------------------------------------------------------------
int main(void)
{
    HAL_Init();                          // Initialize the HAL Library
    SystemClock_Config();               // Configure the system clock
    MX_GPIO_Init();                      // Initialize all configured peripherals (GPIO)

    while (1)
    {
        // Check if left button flag is set and toggle the red LED
        if (btn_left_flag)
        {
            HAL_GPIO_TogglePin(LED_GPIO_PORT, LED_R_PIN);
            btn_left_flag = 0;
        }
        // Check if right button flag is set and toggle the blue LED
        if (btn_right_flag)
        {
            HAL_GPIO_TogglePin(LED_GPIO_PORT, LED_B_PIN);
            btn_right_flag = 0;
        }
        // Check if up button flag is set and play sound pattern 1
        if (btn_up_flag)
        {
            sound1();
            btn_up_flag = 0;
```

```
            }
            // Check if down button flag is set and play sound pattern 2
            if (btn_down_flag)
            {
                sound2();
                btn_down_flag = 0;
            }
        }
}

//-----------------------------------------------------------------------------
// @brief    Callback function for external GPIO interrupts.
// @param    GPIO_Pin: Specifies the pin connected to the external interrupt.
// @note     This function debounces the button press (using a delay) and sets the corresponding flag
//                based on which button was pressed.
//-----------------------------------------------------------------------------
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    HAL_Delay(20);    // Simple debounce delay (20 ms)
    if (HAL_GPIO_ReadPin(BTN_GPIO_PORT, GPIO_Pin) == GPIO_PIN_RESET)
    {
        switch (GPIO_Pin)
        {
        case BTN_LEFT_PIN:
            btn_left_flag = 1;
            break;
        case BTN_RIGHT_PIN:
            btn_right_flag = 1;
            break;
```

```
            case BTN_UP_PIN:
                btn_up_flag = 1;
                break;
            case BTN_DOWN_PIN:
                btn_down_flag = 1;
                break;
            default:
                break;
            }
        }
}

//-----------------------------------------------------------------------------
// @brief    EXTI line interrupt handler for BTN_UP_PIN.
// @note      Calls the HAL GPIO EXTI IRQ handler to process the up button interrupt.
//-----------------------------------------------------------------------------
void EXTI2_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(BTN_UP_PIN);
}

//-----------------------------------------------------------------------------
// @brief    EXTI line interrupt handler for BTN_DOWN_PIN.
// @note      Calls the HAL GPIO EXTI IRQ handler to process the down button interrupt.
//-----------------------------------------------------------------------------
void EXTI3_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(BTN_DOWN_PIN);
}
```

```
//-----------------------------------------------------------------------------
// @brief    Configure the system clock for STM32F407.
// @note     Sets up the PLL and clock dividers to achieve the desired system clock frequency.
//           This configuration uses the HSE oscillator as the clock source.
//-----------------------------------------------------------------------------
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    __HAL_RCC_PWR_CLK_ENABLE();                                  // Enable power control clock

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1); // Configure voltage scaling

    // Configure the main oscillator (HSE) and PLL
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 168;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    // Select PLL as system clock source and configure bus clocks dividers
```

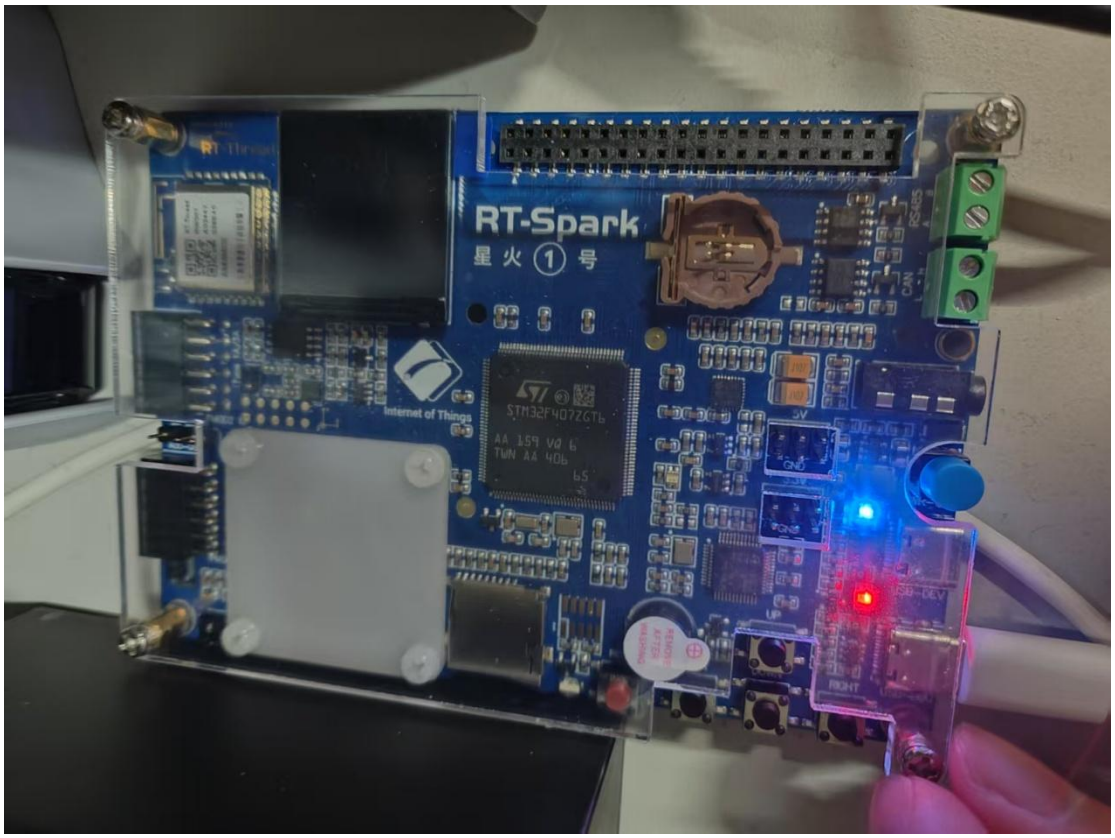
```
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_SYSCLK
                                | RCC_CLOCKTYPE_PCLK1 |
RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
}
```

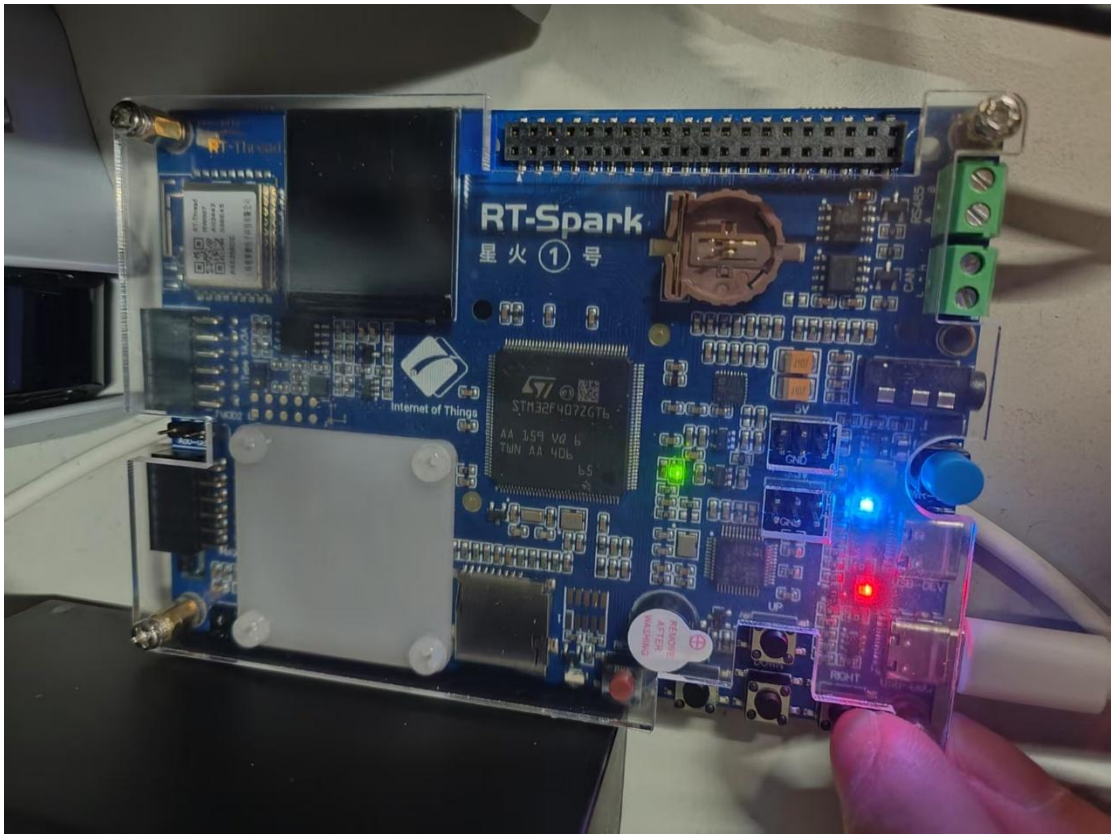
3 实验结果分析与总结

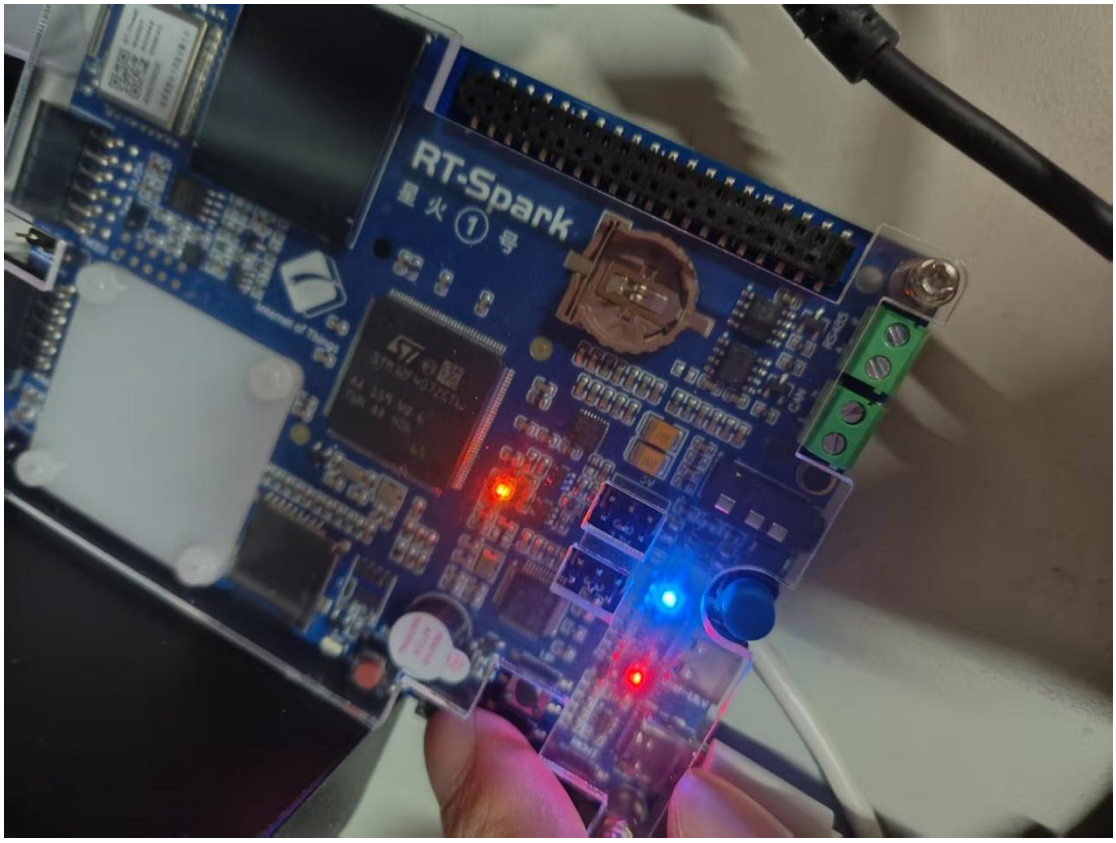
由于本次实验涉及按下按键发出相关报警声的内容，图片形式无法展现，故在附件中添加视频以完全展现。

本小节主要是展示完成按下 BTN_LEFT 按键翻转红色 LED_R 灯、 BTN_RIGHT 按键翻转蓝色 LED_B 灯的实验内容。

如图可知，在开发板刚启动的时候，所有灯都是灭的。当我按下右方向键时亮蓝色 LED 灯，按下左方向键时亮红色 LED 灯。


RT-Spark
星火①号
Internet of Things
STM32F407ZGT6
RS485
CAN
5V
GND
UP
RIGHT


RT-Spark
星火①号
Internet of Things
STM32F407ZGT6
RS485
CAN
5V
GND
UP
RIGHT


RT-Spark
星火①号