

**计算机科学与工程学院**

**“嵌入式系统”实验报告书**

**题目：嵌入式系统作业4——gpio\_intr**

**学号：922106840127**

**姓名：刘宇翔**

**成绩**

**日期： 2025年 3月 18日**

# 1 题目要求

1. **题目设计要求**

【1】作业目标：

（1）熟悉中断工作原理；

（2）能够熟练外部中断开发编程；

（3）能够使用中断函数。

【2】作业内容：

设计一个输入输出的综合项目实例，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。

【3】完成要求：

（1）工程名称命名为：学号\_gpio\_intr；

（2）采用中断的方式检测按键状态，且按键带有消抖功能；

（3）作业提交：实验报告、工程和运行结果拍摄视频压缩后QQ上提交，压缩文件名称为“学号\_姓名\_gpio\_intr”，不按要求进行命名的，该实验或作业没有分。

**2. 拟实现的具体功能**

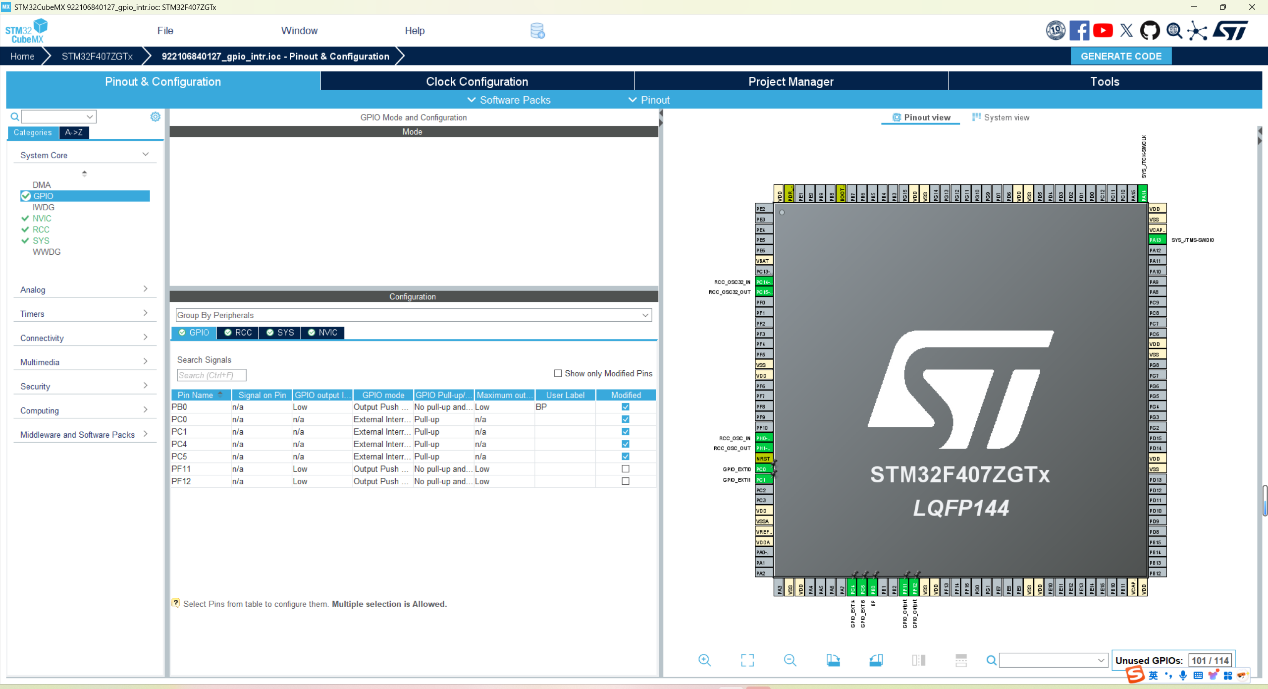
完成基于STM32CubeMX硬件设计平台实现的基于方向键加以中断实现对应的LED灯输出，使用BTN\_LEFT按键翻转红色LED\_R灯、 BTN\_RIGHT按键翻转蓝色LED\_B灯，按下BTN\_UP按键发出5s救护车报警声，按下BTN\_DOWN键按发出5s电动车报警声。

# 2 总体设计

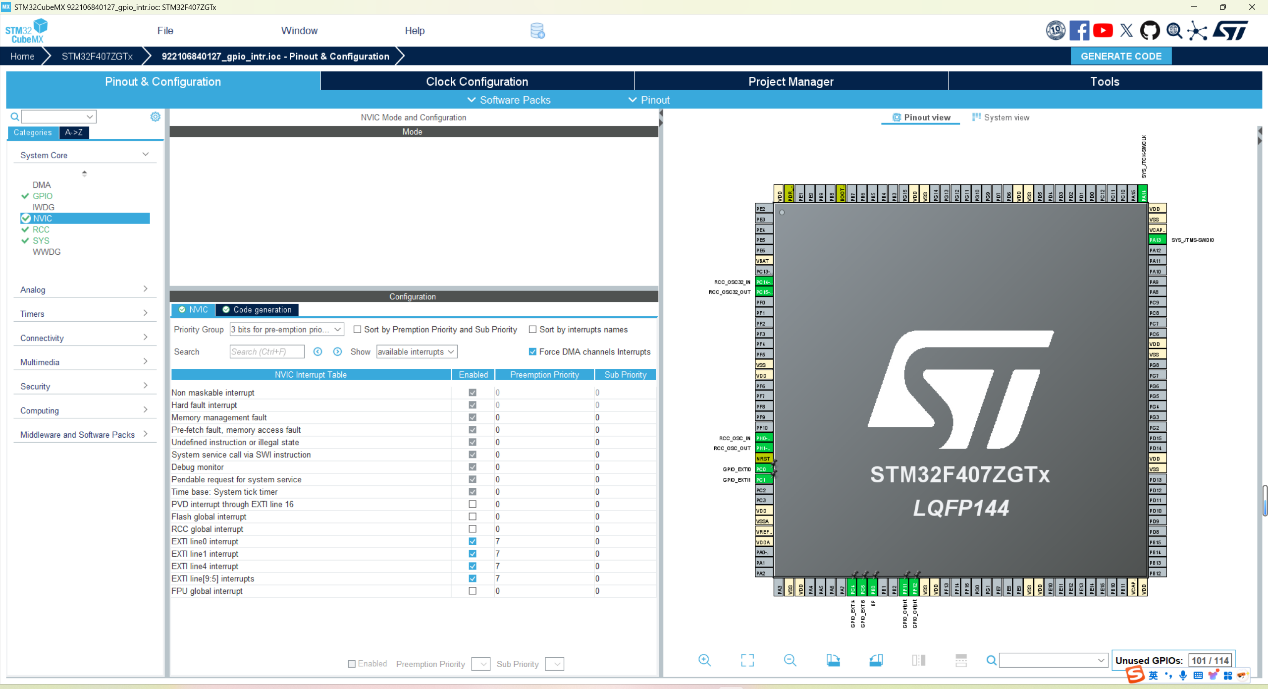
## 2.1硬件设计

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。

1. **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。

1. **软件流程图**
2. **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。

1. **μvision详细代码**

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* 本程序实现基于外部中断的输入响应控制，包括LED灯状态翻转和报警声音输出。

\* 按键功能：

\* - BTN\_LEFT (GPIO\_PIN\_0)：翻转红色LED\_R（连接在GPIOF的PIN\_12）

\* - BTN\_DOWN (GPIO\_PIN\_1)：发出电动车报警声（调用sound2函数约5秒）

\* - BTN\_RIGHT (GPIO\_PIN\_4)：翻转蓝色LED\_B（连接在GPIOF的PIN\_11）

\* - BTN\_UP (GPIO\_PIN\_5)：发出救护车报警声（调用sound1函数两次，约5秒）

\*

\* 系统基于STM32单片机和HAL库，通过外部中断及延时函数实现按键消抖和声音控制。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

#include "gpio.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

// 此处可根据需要添加其他头文件

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

// 可在此处添加自定义数据类型

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

// 可在此处定义宏常量

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

// 可在此处定义私有宏

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

/\* USER CODE BEGIN PV \*/

// 可在此处定义全局变量

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

/\* USER CODE BEGIN PFP \*/

// 声明自定义的延时函数和声音输出函数

void delay(uint32\_t i);

void sound1(void);

void sound2(void);

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\*\*

\* @brief 外部中断回调函数

\* @param GPIO\_Pin: 产生中断的GPIO引脚号

\* @note 该函数根据不同的引脚号实现不同功能：

\* - GPIO\_PIN\_0: 切换红色LED\_R的状态（翻转LED）

\* - GPIO\_PIN\_1: 循环调用sound2()函数产生电动车报警声

\* - GPIO\_PIN\_4: 切换蓝色LED\_B的状态（翻转LED）

\* - GPIO\_PIN\_5: 调用sound1()函数产生救护车报警声

\* 同时通过短暂延时实现简单按键消抖处理。

\*/

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

{

// 延时15ms进行简单的软件消抖

HAL\_Delay(15);

// 检查对应引脚是否仍处于按下状态（低电平有效）

if(HAL\_GPIO\_ReadPin(GPIOC, GPIO\_Pin) == GPIO\_PIN\_RESET)

{

// BTN\_LEFT：按下时翻转红色LED\_R（GPIOF PIN\_12）

if(GPIO\_Pin == GPIO\_PIN\_0)

{

HAL\_GPIO\_TogglePin(GPIOF, GPIO\_PIN\_12);

}

// BTN\_DOWN：按下时发出电动车报警声，通过循环调用sound2函数产生声音

if(GPIO\_Pin == GPIO\_PIN\_1)

{

for(int i = 0; i < 38; i++)

{

sound2();

}

// 声音播放完毕后，关闭蜂鸣器输出

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_RESET);

}

// BTN\_RIGHT：按下时翻转蓝色LED\_B（GPIOF PIN\_11）

if(GPIO\_Pin == GPIO\_PIN\_4)

{

HAL\_GPIO\_TogglePin(GPIOF, GPIO\_PIN\_11);

}

// BTN\_UP：按下时发出救护车报警声，通过调用sound1函数两次实现

if(GPIO\_Pin == GPIO\_PIN\_5)

{

sound1();

sound1();

// 声音播放完毕后，关闭蜂鸣器输出

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_RESET);

}

}

}

/\* USER CODE END 0 \*/

/\*\*

\* @brief 主函数入口

\* @retval int

\*/

int main(void)

{

/\* USER CODE BEGIN 1 \*/

// 用户可以在此处添加初始化前的代码

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* 重置所有外设，初始化Flash接口和SysTick定时器 \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

// 此处可添加其他初始化代码

/\* USER CODE END Init \*/

/\* 配置系统时钟 \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

// 系统时钟配置后，可进行其他系统相关初始化

/\* USER CODE END SysInit \*/

/\* 初始化所有配置的外设 \*/

MX\_GPIO\_Init();

/\* USER CODE BEGIN 2 \*/

// 可在此处添加其他初始化代码（如串口、定时器等初始化）

/\* USER CODE END 2 \*/

/\* 无限循环 \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

// 主循环中无其他任务，所有操作均由中断处理完成

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

// 可在此处添加其他后台任务或低功耗处理代码

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief 系统时钟配置函数

\* @note 配置PLL和各总线时钟，确保MCU及外设运行在预定频率下

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* 配置内部电压调节器输出电压

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

/\*\* 初始化RCC振荡器，根据RCC\_OscInitTypeDef结构体指定的参数配置

\*/

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;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* 初始化CPU、AHB和APB总线时钟

\*/

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;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_5) != HAL\_OK)

{

Error\_Handler();

}

}

/\* USER CODE BEGIN 4 \*/

/\*\*

\* @brief 简单延时函数

\* @param i: 延时计数值，根据MCU频率调整实际延时

\* @retval None

\* @note 该函数采用空循环方式实现延时，精度不高，仅适用于简单延时需求

\*/

void delay(uint32\_t i)

{

while(i--);

}

/\*\*

\* @brief 产生救护车报警声

\* @note 通过控制蜂鸣器引脚电平的周期性变化，并逐步减小延时参数，模拟出声音频率变化

\* @retval None

\*/

void sound1(void)

{

uint32\_t i = 30000; // 初始延时参数

while(i)

{

// 输出低电平使蜂鸣器发声

HAL\_GPIO\_WritePin(GPIOB, BP\_Pin, GPIO\_PIN\_RESET);

delay(i);

// 输出高电平使蜂鸣器停止发声

HAL\_GPIO\_WritePin(GPIOB, BP\_Pin, GPIO\_PIN\_SET);

delay(i);

i = i - 6; // 逐步减小延时，改变声音频率

}

}

/\*\*

\* @brief 产生电动车报警声

\* @note 与sound1函数类似，但初始延时参数较小，从而产生不同的声音效果

\* @retval None

\*/

void sound2(void)

{

uint32\_t i = 6000; // 初始延时参数

while(i)

{

// 输出低电平使蜂鸣器发声

HAL\_GPIO\_WritePin(GPIOB, BP\_Pin, GPIO\_PIN\_RESET);

delay(i);

// 输出高电平使蜂鸣器停止发声

HAL\_GPIO\_WritePin(GPIOB, BP\_Pin, GPIO\_PIN\_SET);

delay(i);

i = i - 6; // 逐步减小延时，改变声音频率

}

}

/\* USER CODE END 4 \*/

/\*\*

\* @brief 错误处理函数

\* @note 当程序发生错误时调用，禁止中断并进入死循环

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

// 禁止所有中断，防止系统继续运行

\_\_disable\_irq();

while (1)

{

// 可在此处添加错误指示或重启代码

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief 断言失败处理函数

\* @param file: 出错的源文件名称指针

\* @param line: 出错的行号

\* @retval None

\* @note 用户可以在此处添加自定义的错误处理代码

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

// 例如，可通过printf打印错误信息：

// printf("Wrong parameters value: file %s on line %d\r\n", file, line);

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

# 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。

