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：ex6_922106840127_DMA

学号：922106840127

姓名：刘宇翔

成绩

日期：　　2025 年　　4 月　　8 日

1 题目要求

1. 题目设计要求

（1） 实验内容

串口 UART1 以 DMA 方式接收数据，当收到“*LED ON*”时，点亮绿灯，收到“*LED-OFF*”时，熄灭绿灯；

串口 UART1 以中断方式接收数据，当收到“*LED ON*”时，点亮红灯，收到“*LED-OFF*”时，熄灭红灯；

当轻触一下 LEFT 按键时，切换串口 UART1 的中断 或 DMA 接收方式，并将当前的接收方式打印到 PC 串口上。

串口配置：115200,8,N,1;

（2）完成要求

工程名称命名：ex6_学号_DMA，并打包成：ex6_学号_DMA.rar 压缩文件夹

实验报告 PDF 格式：ex6_学号_DMA.pdf

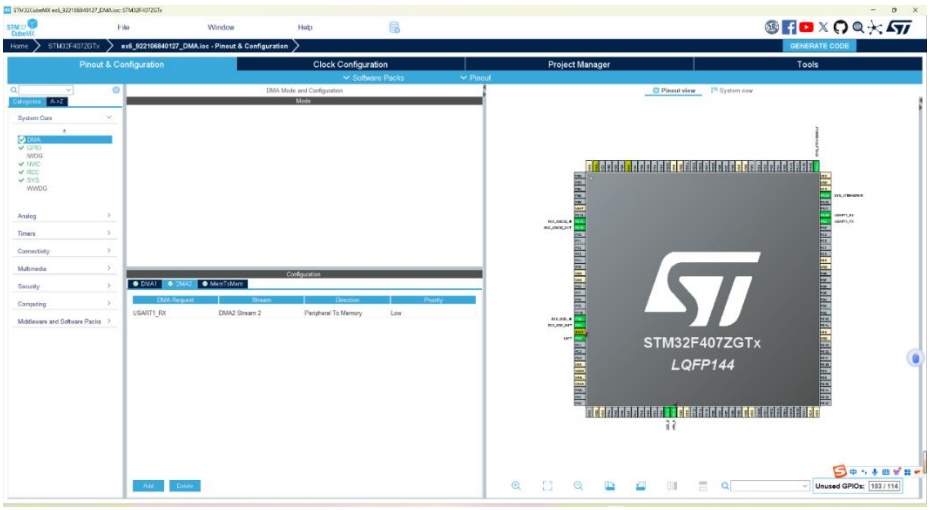
2. 拟实现的具体功能

本次实验拟实现一个基于 STM32CubeMX 硬件平台的 UART 通信控制系统，通过 UART1 接口以两种接收模式——DMA 方式和中断方式，分别实现对 PC 端发送的命令字符串的接收处理，从而控制 LED 灯的状态。在 DMA 模式下，当接收到命令“*LED ON*”时，点亮绿灯；接收到“*LED-OFF*”时，熄灭绿灯；而在中断模式下，对应命令将分别点亮或熄灭红灯。实验中，系统还通过轻触 LEFT 按键切换 UART1 的接收模式，并将当前的模式信息（即“Switched to DMA receive mode”或“Switched to Interrupt receive mode”）通过 PC 串口打印输出。UART 参数严格设置为 115200,8,N,1，系统采用 HAL 库驱动函数确保数据传输的可靠性和实时性，为后续功能扩展奠定坚实基础。

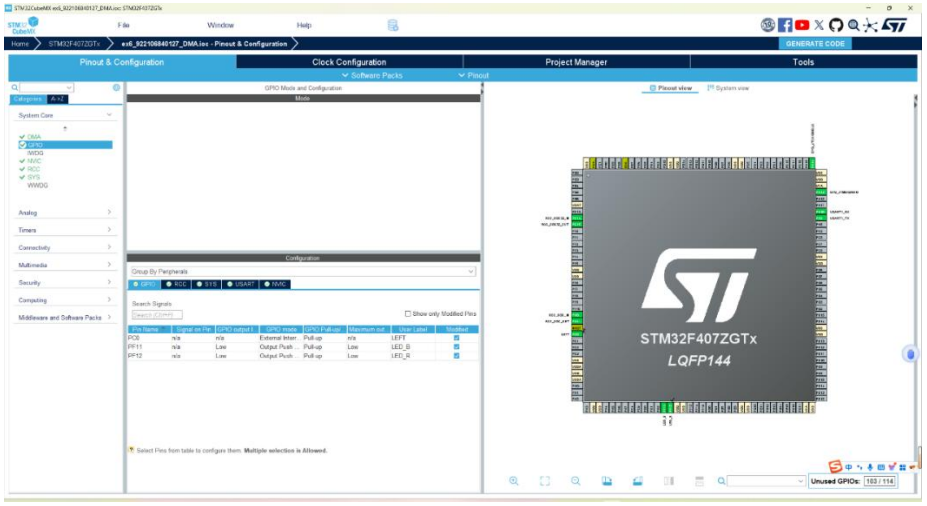
2 总体设计

2.1 硬件设计

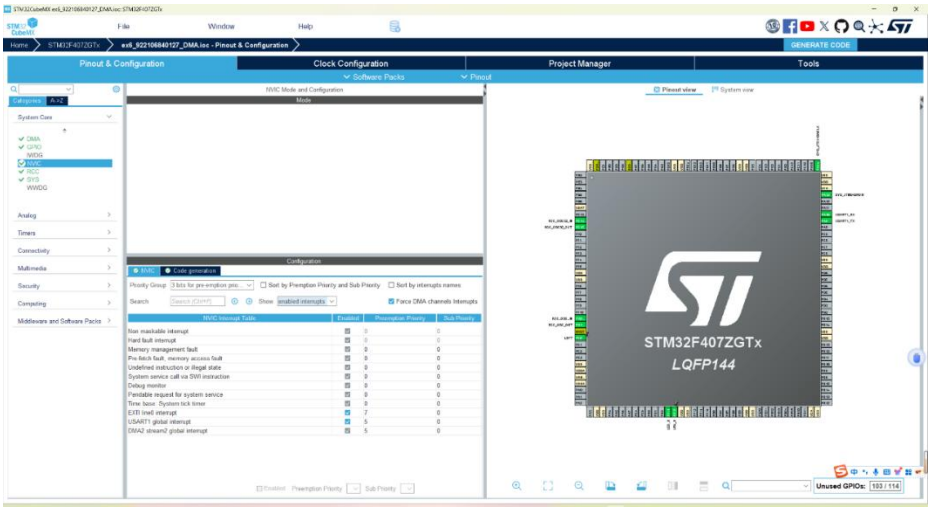
1. 硬件设计思路



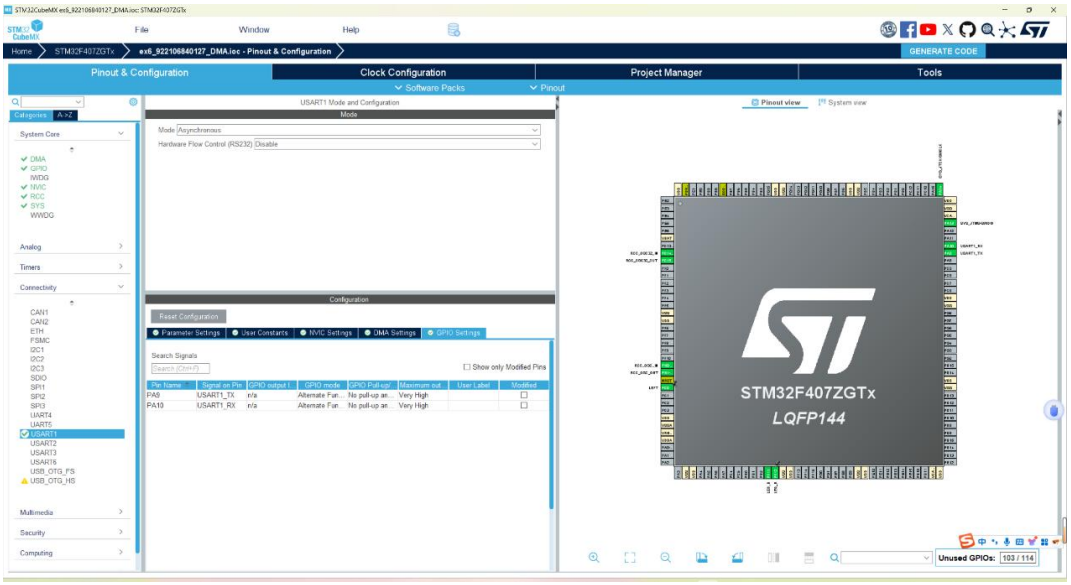
DMA 设置



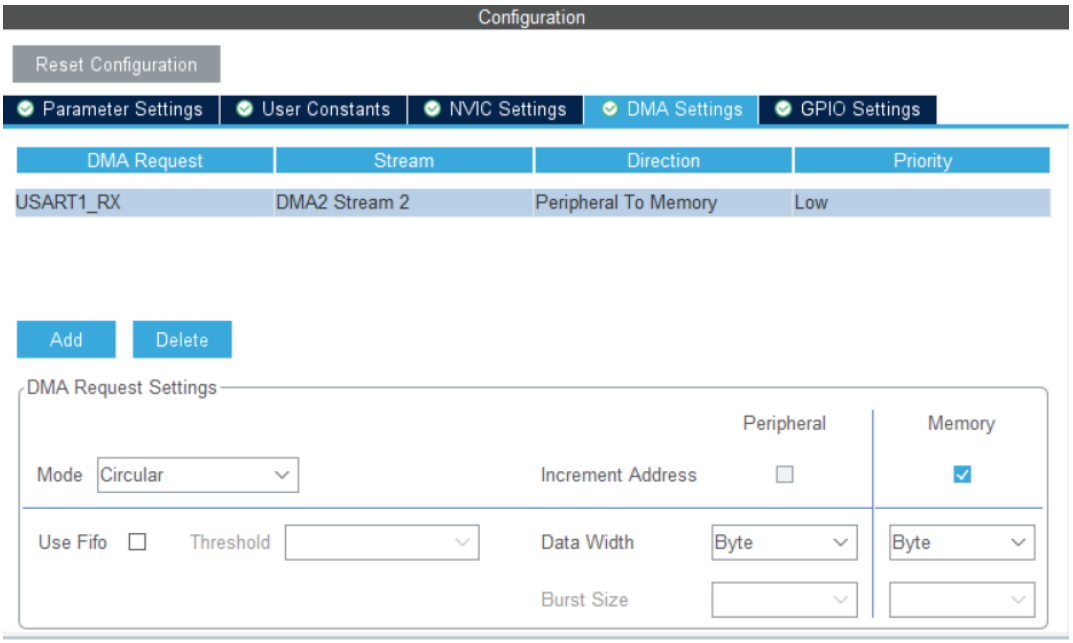
GPIO 设置



NVIC 设置



USART 设置



USART 中 DMA 相关的设置

如图所示，我在 NVIC 的相关设置对几种中断的相关属性做了具体的规定，USART 的通信模式采用的是异步通信，DMA 相关参数与功能的设计参考教材。

其他设置与前几次实验相同，如 RCC 采用 Crystal/Ceramic Resonator，使用 Serial Wire 用于调试对应的 debug 接口，对 Clock Configuration 的时钟配置进行规范，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. 软件流程概述

本软件主要实现基于 UART 通信的 LED 灯控制系统，通过 STM32 单片机利用 UART1 接口在 DMA 和中断两种接收模式下实时解析来自 PC 端的指令，从而分别控制绿灯和红灯的亮灭，并通过按键切换接收模式，将当前模式信息实时反馈给 PC。系统基于 HAL 库进行外设初始化，采用模块化设计方式，整体结构清晰、易于维护且具备良好的响应性。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 HAL_Init()完成底层硬件初始化，并通过 SystemClock_Config()配置系统时钟，确保各模块按预期工作；随后依次初始化 GPIO、DMA 和 USART1，并设置 UART1 参数为 115200,8,N,1，默认启动中断接收模式。

（2）UART 接收与命令解析模块

系统通过 UART1 接收 PC 端发送的命令字符串，在中断模式下利用 HAL_UART_RxCpltCallback() 逐字节接收，当检测到换行符时调用 ProcessReceivedData()函数解析命令，对应执行 LED 灯的开关控制；在 DMA 模式下，则利用 HAL_UARTEx_RxEventCallback()回调完成数据帧的接收及解析，确保完整命令正确处理。

（3）LED 控制模块

在命令解析后，根据指令“*LED-ON*”或“*LED-OFF*”，在中断模式下调用 ControlLED_R()控制红色 LED 的亮灭，在 DMA 模式下调用 ControlLED_B()控制绿色 LED 的亮灭，同时通过 UART 将操作结果反馈给 PC。

（4）接收模式切换模块

系统通过 LEFT 按键外部中断触发，在 HAL_GPIO_EXTI_Callback()中调用 change_rx_mode()函数切换 UART 接收模式（中断与 DMA 模式互换），并利用 UART 发送切换后的模式信息到 PC，实现灵活的接收方式切换。

总体而言，本软件基于 STM32 和 HAL 库，实现了一个功能完备、响应迅速的 LED 灯控制系统，结构划分明确、逻辑清晰，为后续进一步的通信与控制扩展提供了坚实的基础。

2.软件流程分解

A. 初始阶段

开始 → 系统初始化

程序启动后，首先完成各项硬件与外设的初始化配置，包括：

- **系统底层初始化**：调用 HAL_Init() 对 MCU 进行复位与基础设置；
- **系统时钟配置**：通过 SystemClock_Config() 配置主频及各外设时钟，确保系统稳定运行；
- **外设初始化**：依次初始化 GPIO、DMA 和 USART1（分别调用 MX_GPIO_Init()、MX_DMA_Init() 及 MX_USART1_UART_Init()），为 UART 通信及 LED 控制提供硬件支持；
- **UART 接收启动**：默认以中断模式启动 UART1 接收功能（调用 HAL_UART_Receive_IT()），以接收来自 PC 端的命令数据。

B. 主循环结构

初始化完成 → 进入主循环

系统进入主循环后，所有 UART 数据接收、命令解析、LED 控制和接收模式切换均由中断或空闲中断回调自动完成，主循环内无需额外处理。

C. UART 数据接收（中断模式）流程

当系统处于中断模式时，PC 端通过 UART1 发送数据，每接收到 1 字节数据时便触发 HAL_UART_RxCpltCallback() 回调函数。

在该回调中，数据逐字节存入接收缓冲区，直到检测到换行符（\r 或 \n），随后调用 ProcessReceivedData() 解析命令。如果命令为“LED_ON”或“LED_OFF”，则调用 ControlLED_R() 控制红色 LED 的亮灭，并通过 UART 将操作结果反馈给 PC。

D. UART 数据接收（DMA 模式）流程

当系统处于 DMA 模式下，利用 HAL_UARTEx_RxEventCallback() 回调函数检测 UART 空闲状态。当一整帧数据接收完成后，该回调函数获得实际接收的字节数，并调用 ProcessReceivedData() 对数据进行解析。

在该流程中，如果解析到“LED_ON”或“LED_OFF”命令，则调用 ControlLED_B() 控制绿色 LED 的亮灭，同时通过 UART 发送反馈信息；接收完成后，DMA 重

新启动接收（调用 HAL_UARTEx_ReceiveToIdle_DMA()，接收长度设为MAX_RX_BUFFER_SIZE），确保数据完整性。

E. 接收模式切换流程

通过 LEFT 按键的外部中断触发，在 HAL_GPIO_EXTI_Callback() 中检测到LEFT 按键按下时（结合简单消抖处理），调用 change_rx_mode() 函数完成 UART接收模式的切换。

在此过程中，系统先调用 UART_AbortReceive() 中止当前接收，然后修改全局接收模式变量（在中断模式与 DMA 模式间切换），并通过 UART_RestartReceive()重新启动对应的 UART 接收方式。切换后，系统通过 UART 将当前接收模式提示信息（如“Switched to DMA receive mode”或“Switched to Interrupt receive mode”）发送到 PC 端。

F. LED 控制反馈流程

在解析 UART 接收到的数据后，ProcessReceivedData() 函数根据命令的内容调用相应的 LED 控制函数：

- 在中断模式下，调用 ControlLED_R() 控制红色 LED 的开关；
- 在 DMA 模式下，调用 ControlLED_B() 控制绿色 LED 的开关。

同时，通过 UART 将 LED 操作结果反馈到 PC 端，确保操作过程清晰可追踪。

G. 循环机制

数据接收、命令解析和 LED 控制流程完成后，系统退出中断回调函数，返回主循环并持续等待下一次数据接收或按键触发，从而实现整个系统的连续、稳定运行，确保每条命令都能被实时处理与反馈。

3. μvision 详细代码

```
#include "main.h"
#include "usart.h"
#include "gpio.h"
#include "dma.h"
#include "string.h"
#include "stdio.h"

#define MAX_RX_BUFFER_SIZE 200
```

```
#define LED_ON_CMD "*LED-ON*"
#define LED_OFF_CMD "*LED-OFF*"

uint8_t rx_buffer[MAX_RX_BUFFER_SIZE];
uint8_t rx_index = 0;

typedef enum {
    RX_MODE_IT = 0,
    RX_MODE_DMA = 1
} RxMode_t;
RxMode_t rx_mode = RX_MODE_IT;

void SystemClock_Config(void);
uint8_t IsNewLineChar(uint8_t ch);
void ProcessReceivedData(void);
void ControlLED_R(uint8_t state);
void ControlLED_B(uint8_t state);
void UART_AbortReceive(void);
void UART_RestartReceive(void);
void change_rx_mode(void);

uint8_t IsNewLineChar(uint8_t ch)
{
    return (ch == '\r' || ch == '\n');
}

void ProcessLEDOnCommand(void)
{
    if (rx_mode == RX_MODE_IT)
    {
```

```
        ControlLED_R(1);
        HAL_UART_Transmit(&huart1, (uint8_t *)"LED is ON (red)\r\n", strlen("LED is ON (red)\r\n"), HAL_MAX_DELAY);
    }
    else if (rx_mode == RX_MODE_DMA)
    {
        ControlLED_B(1);
        HAL_UART_Transmit(&huart1, (uint8_t *)"LED is ON (green)\r\n", strlen("LED is ON (green)\r\n"), HAL_MAX_DELAY);
    }
}

void ProcessLEDOffCommand(void)
{
    if (rx_mode == RX_MODE_IT)
    {
        ControlLED_R(0);
        HAL_UART_Transmit(&huart1, (uint8_t *)"LED is OFF (red)\r\n", strlen("LED is OFF (red)\r\n"), HAL_MAX_DELAY);
    }
    else if (rx_mode == RX_MODE_DMA)
    {
        ControlLED_B(0);
        HAL_UART_Transmit(&huart1, (uint8_t *)"LED is OFF (green)\r\n", strlen("LED is OFF (green)\r\n"), HAL_MAX_DELAY);
    }
}

void ProcessEchoCommand(void)
{
```

```
    HAL_UART_Transmit(&huart1, rx_buffer, rx_index, HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart1, (uint8_t *)"\r\n", 2, HAL_MAX_DELAY);
}


void ProcessReceivedData(void)
{
    rx_buffer[rx_index] = '\0';
    if (strstr((char *)rx_buffer, LED_ON_CMD) != NULL)
    {
        ProcessLEDOnCommand();
    }
    else if (strstr((char *)rx_buffer, LED_OFF_CMD) != NULL)
    {
        ProcessLEDOffCommand();
    }
    else
    {
        ProcessEchoCommand();
    }
    rx_index = 0;
}


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart == &huart1)
    {
        if (rx_index < MAX_RX_BUFFER_SIZE - 1)
        {
```

```
            if (IsNewLineChar(rx_buffer[rx_index]))
            {
                ProcessReceivedData();
            }
            else
            {
                rx_index++;
            }
        }
        else
        {
            rx_index = 0;
        }

        if (rx_mode == RX_MODE_IT)
        {
            if (HAL_UART_Receive_IT(&huart1, &rx_buffer[rx_index], 1) !=
HAL_OK)
            {
                Error_Handler();
            }
        }
    }
}

void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{
    if ((huart == &huart1) && (rx_mode == RX_MODE_DMA))
    {
        __HAL_UART_CLEAR_IDLEFLAG(huart);
```

```
        if (Size == 0)
        {
            HAL_UART_DMAStop(&huart1);
            if (HAL_UARTEx_ReceiveToIdle_DMA(&huart1, rx_buffer,
MAX_RX_BUFFER_SIZE) != HAL_OK)
            {
                HAL_UART_Transmit(&huart1, (uint8_t *)"Restart DMA
failed!\r\n",
                                  strlen("Restart DMA failed!\r\n"),
HAL_MAX_DELAY);
                Error_Handler();
            }
            return;
        }

        rx_index = Size;
        rx_buffer[rx_index] = '\0';
        ProcessReceivedData();

        HAL_UART_DMAStop(&huart1);
        if (HAL_UARTEx_ReceiveToIdle_DMA(&huart1, rx_buffer,
MAX_RX_BUFFER_SIZE) != HAL_OK)
        {
            HAL_UART_Transmit(&huart1, (uint8_t *)"Restart DMA failed!\r\n",
                              strlen("Restart DMA failed!\r\n"),
HAL_MAX_DELAY);
            Error_Handler();
        }
    }
}
```

```
void UART_AbortReceive(void)
{
    if (rx_mode == RX_MODE_IT)
    {
        HAL_UART_AbortReceive_IT(&huart1);
    }
    else if (rx_mode == RX_MODE_DMA)
    {
        HAL_UART_DMAStop(&huart1);
    }
}

void UART_RestartReceive(void)
{
    if (rx_mode == RX_MODE_IT)
    {
        if (HAL_UART_Receive_IT(&huart1, &rx_buffer[rx_index], 1) !=
HAL_OK)
        {
            Error_Handler();
        }
    }
    else if (rx_mode == RX_MODE_DMA)
    {
        if (HAL_UARTEx_ReceiveToIdle_DMA(&huart1, rx_buffer,
MAX_RX_BUFFER_SIZE) != HAL_OK)
        {
            Error_Handler();
        }
```

```
    }
}

void change_rx_mode(void)
{
    UART_AbortReceive();
    if (rx_mode == RX_MODE_IT)
    {
        rx_mode = RX_MODE_DMA;
        char msg[] = "Switched to DMA receive mode\r\n";
        HAL_UART_Transmit(&huart1, (uint8_t *)msg, strlen(msg),
HAL_MAX_DELAY);
    }
    else
    {
        rx_mode = RX_MODE_IT;
        char msg[] = "Switched to Interrupt receive mode\r\n";
        HAL_UART_Transmit(&huart1, (uint8_t *)msg, strlen(msg),
HAL_MAX_DELAY);
    }
    rx_index = 0;
    UART_RestartReceive();
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == LEFT_Pin)
    {
        static uint32_t last_tick = 0;
        uint32_t current_tick = HAL_GetTick();
```

```
            if (current_tick - last_tick > 500)
            {
                change_rx_mode();
            }
            last_tick = current_tick;
        }
    }

    void ControlLED_R(uint8_t state)
    {
        if (state)
        {
            HAL_GPIO_WritePin(GPIOF, LED_R_Pin, GPIO_PIN_RESET);
        }
        else
        {
            HAL_GPIO_WritePin(GPIOF, LED_R_Pin, GPIO_PIN_SET);
        }
    }

    void ControlLED_B(uint8_t state)
    {
        if (state)
        {
            HAL_GPIO_WritePin(GPIOF, LED_B_Pin, GPIO_PIN_RESET);
        }
        else
        {
            HAL_GPIO_WritePin(GPIOF, LED_B_Pin, GPIO_PIN_SET);
        }
```

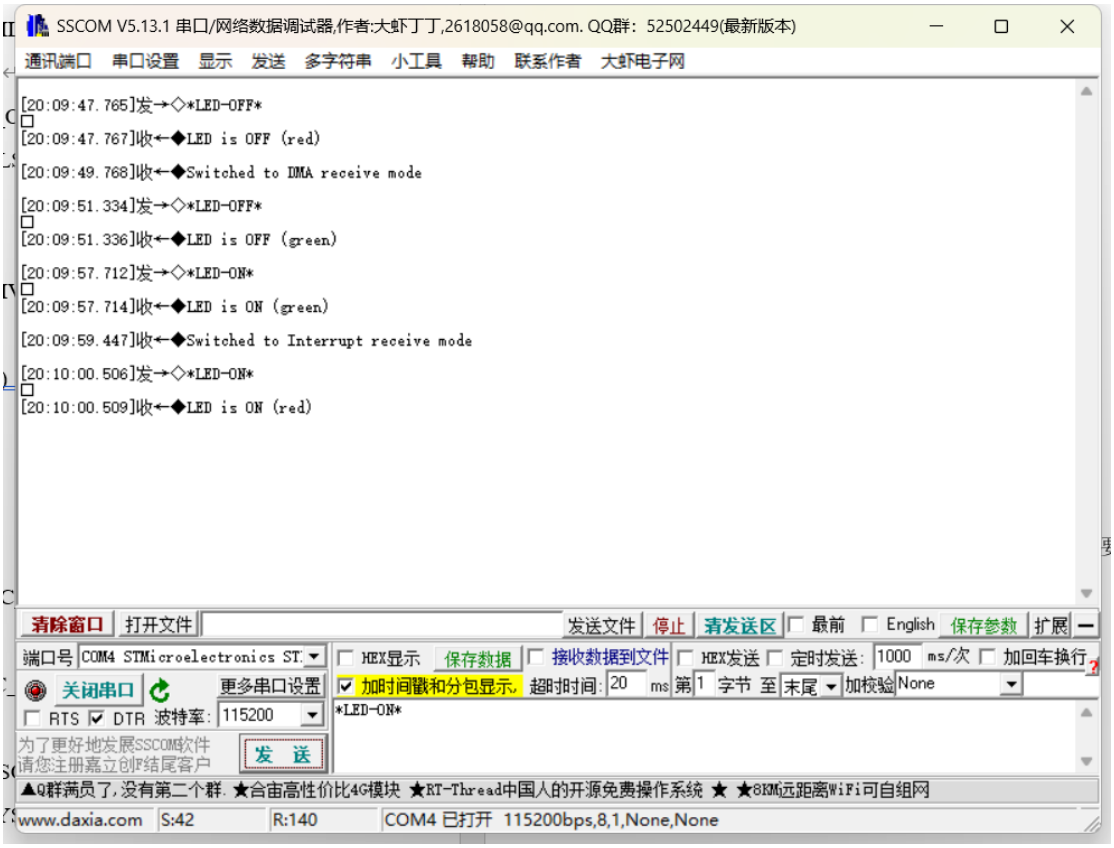
```
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_USART1_UART_Init();

    if (HAL_UART_Receive_IT(&huart1, &rx_buffer[rx_index], 1) != HAL_OK)
    {
        Error_Handler();
    }

    while (1)
    {

    }
}

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_S
CALE1);
```

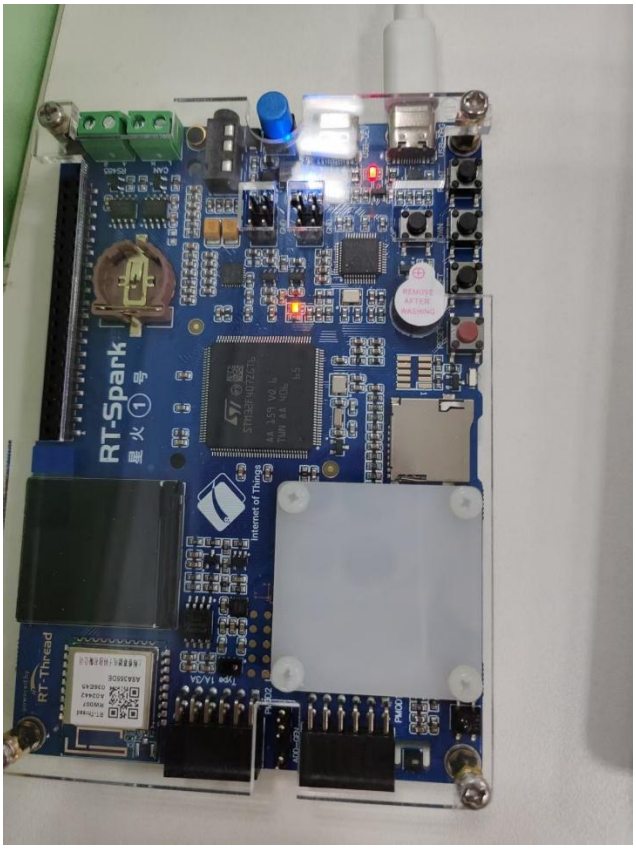
```
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 168;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    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();
    }
}
void Error_Handler(void)
{
```

```
    __disable_irq();
    while (1)
    {
    }
}

#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif
```

3 实验结果分析与总结

我将写好的代码工程经过编译无误后上板验证，并对 DMA 和外部中断两种模式下的 LED 灯的控制进行测试，运行过程中的串口通信与板子验证图片如下：





最初始的时候开机，此时默认中断模式，两个 LED 全亮

发送*LED-OFF*指令，红灯灭

切换到 DMA 模式，发送*LED-OFF*指令，绿灯灭

在 DMA 模式下，发送*LED-ON*指令，绿灯亮

切换到中断模式，发送*LED-ON*指令，红灯亮