

**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**
2. 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

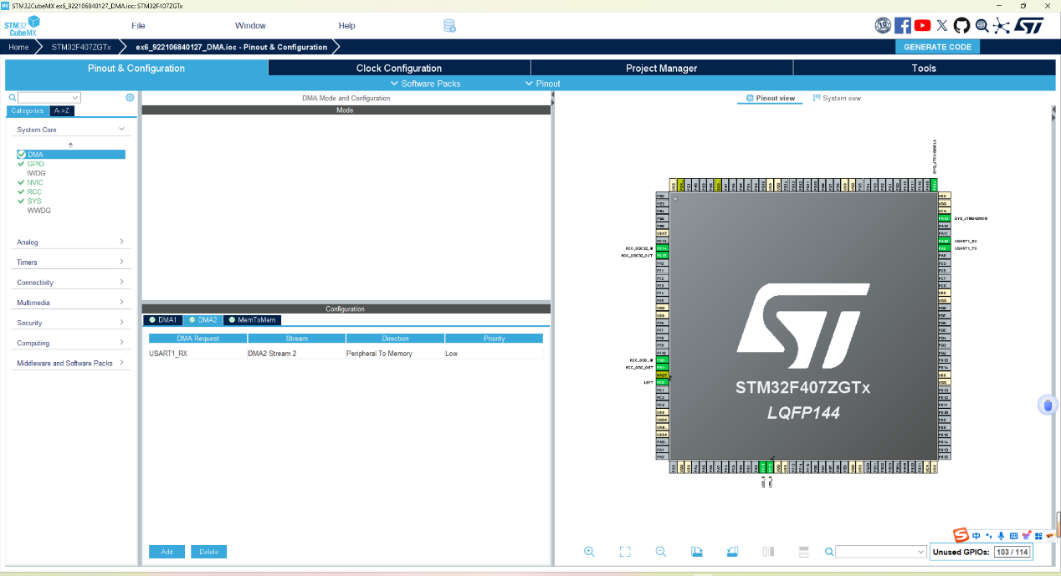
1. **拟实现的具体功能**

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。

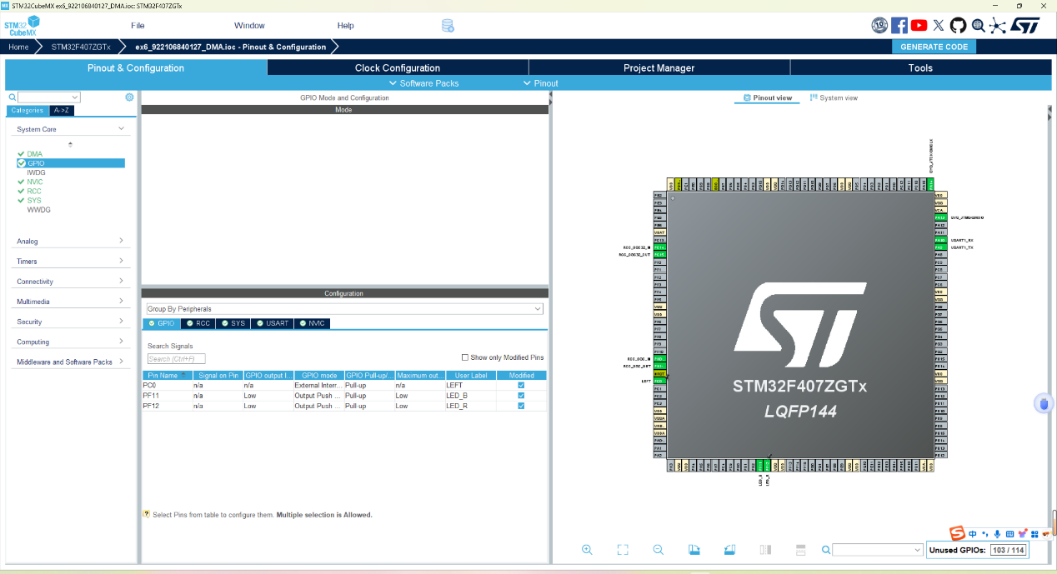
# 2 总体设计

## 2.1硬件设计

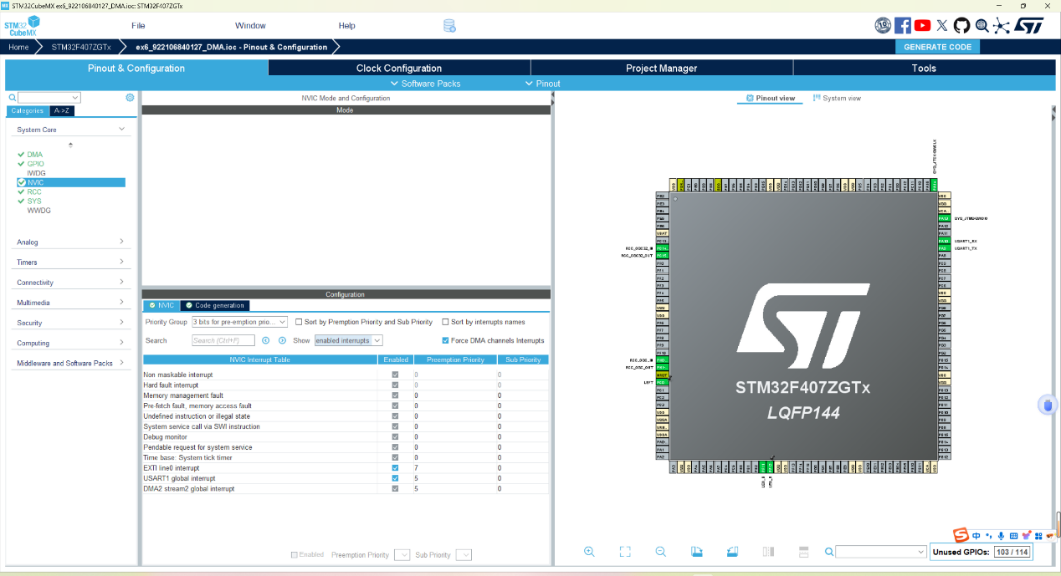
**1. 硬件设计思路**



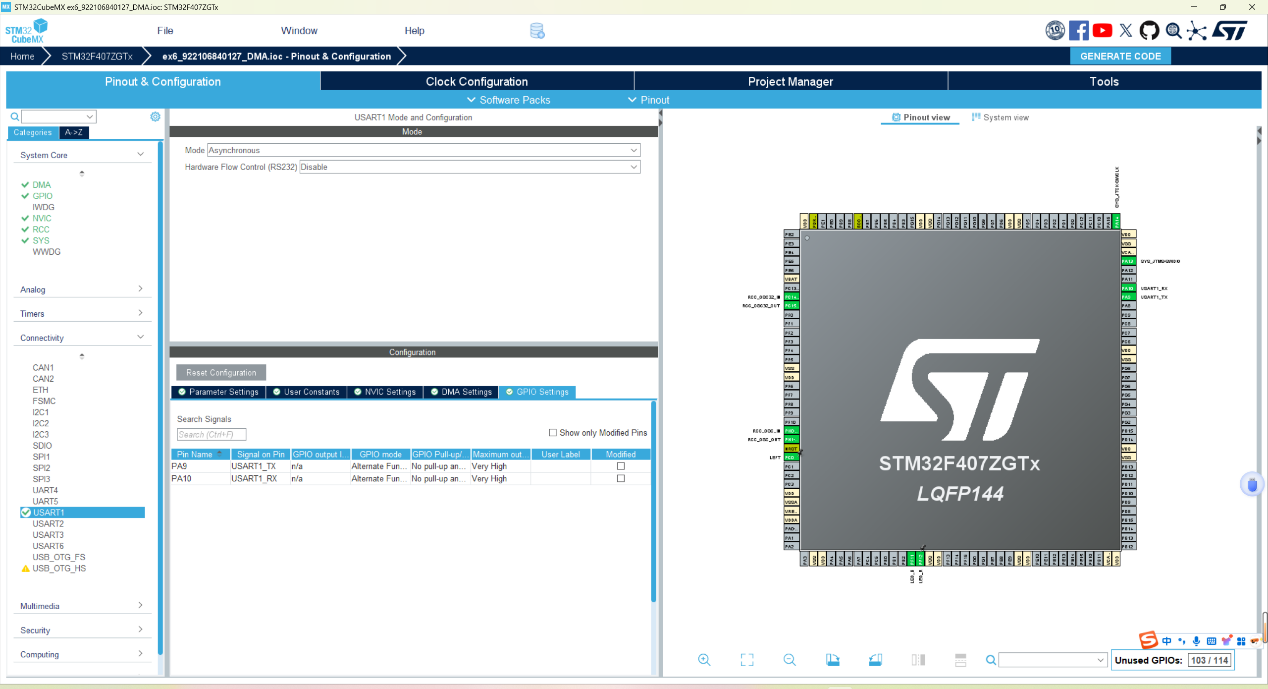
DMA设置



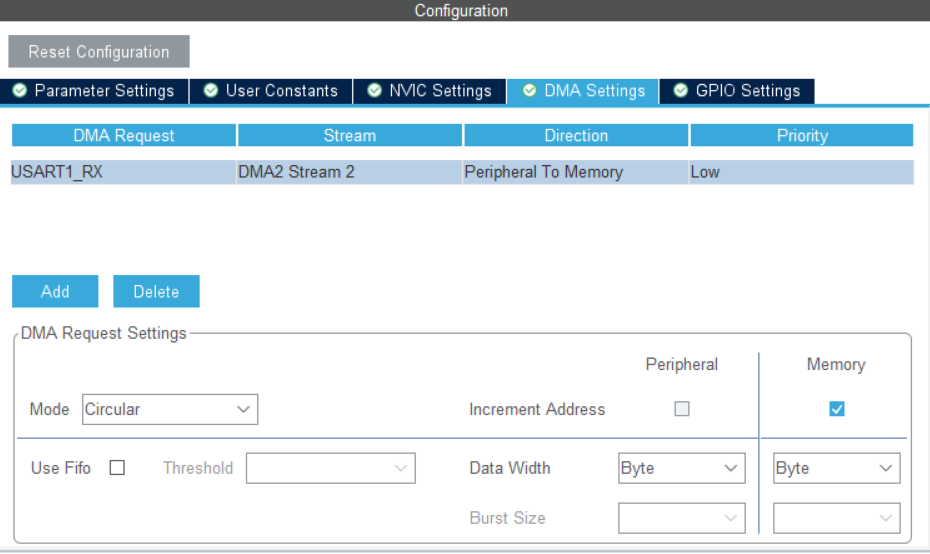
GPIO设置



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\_SCALE1);

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)

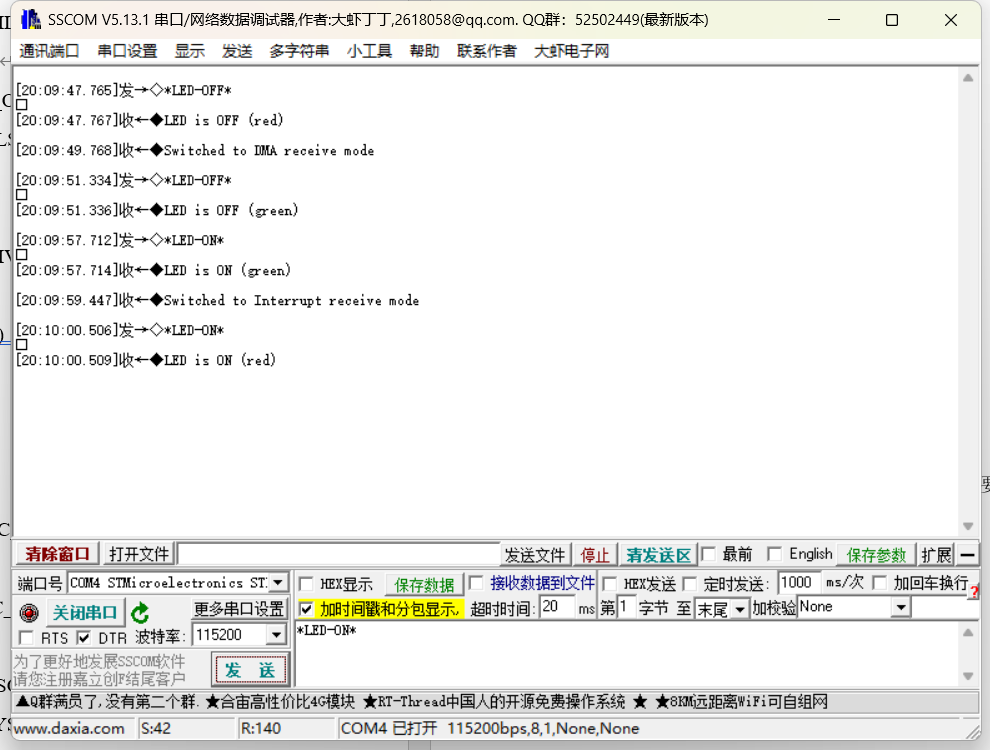
{

}

#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