Task:

实现UART 2串口的接收程序，当收到字符时

1. 在电脑的输出窗口显示下一个字符，如收到A显示B,

②亮灯:收到字符G，亮绿灯;收到字符R，亮红灯;收到字符B，亮蓝灯;

收到其他字符，不亮灯。实现方式

1、用构件调用方式实现

2、UART部分用直接地址方式实现(即不调用uart.c中的函数，其他部分如GPIO、中断设置可调用函数)

Main.c:

#define GLOBLE\_VAR

#include "includes.h" // 包含总头文件

**int main**(void)

{

// 关总中断

DISABLE\_INTERRUPTS;

// 用户外设模块初始化

**gpio\_init**(LIGHT\_BLUE, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_RED, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_GREEN, GPIO\_OUTPUT, LIGHT\_OFF);

**uart\_init**(UART\_User, 115200); // 初始化串口模块

// 打开使能模块中断

**uart\_enable\_re\_int**(UART\_User); // 使能UART\_USER模块接收中断功能

// 开总中断

ENABLE\_INTERRUPTS;

// 提示进入串口工具进行操作

**printf**("=====================================\n");

**printf**("CS212 LHL\n");

**printf**("=====================================\n");

return 0;

}

Isr.c:

#include "includes.h"

void **UART\_User\_Handler**(void)

{

// 【1】声明局部变量

uint8\_t ch;

uint8\_t flag;

// 【2】关总中断

DISABLE\_INTERRUPTS;

// 【3】读取接到的一个字节

ch = **uart\_re1**(UART\_User, &flag); // 调用接收一个字节的函数，清接收中断位

// 【4】根据flag判断是否真正收到一个字节的数据

**if** (flag) // 有数据

{

**if** ((ch != 'G') **&&** (ch != 'R') **&&** (ch != 'B'))

{

**uart\_send1**(UART\_User, (ch + 1)); // 在电脑的输出窗口显示下一个字符

**uart\_send\_string**(UART\_User, (uint8\_t \*));

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

}

**else if** (ch == 'G')

{

**uart\_send\_string**(UART\_User, (uint8\_t \*)"绿灯亮");

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_ON);

}

**else if** (ch == 'R')

{

**uart\_send\_string**(UART\_User, (uint8\_t \*)"红灯亮");

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

**gpio\_set**(LIGHT\_RED, LIGHT\_ON);

}

**else if** (ch == 'B')

{

**uart\_send\_string**(UART\_User, (uint8\_t \*)"蓝灯亮");

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_ON);

}

}

// 【5】开总中断

ENABLE\_INTERRUPTS;

}









以下是工作原理：

1.主程序初始化 (main.c):

* + 关闭中断：在初始化阶段，关闭总中断 (DISABLE\_INTERRUPTS)，确保初始化过程不会被中断打断。
  + 初始化GPIO：初始化LED灯所连接的GPIO引脚，设置为输出模式，并初始设为关闭状态 (LIGHT\_OFF)。
  + 初始化UART：初始化UART模块，设置波特率为115200。
  + 启用UART接收中断：使能UART的接收中断功能 (uart\_enable\_re\_int(UART\_User))。
  + 开启中断：完成初始化后，开启总中断 (ENABLE\_INTERRUPTS)。
  + 打印提示信息：通过串口打印提示信息，提示用户进行串口操作。

2.中断服务程序 (isr.c):

* 关闭中断：进入中断服务程序后，首先关闭总中断 (DISABLE\_INTERRUPTS)。
* 读取数据：从UART接收寄存器读取一个字节的数据 (ch = uart\_re1(UART\_User, &flag))。
* 处理数据：
  + 显示下一个字符：如果收到的字符不是'G', 'R', 'B'，则发送收到字符的下一个字符 (ch + 1)到UART，同时关闭所有LED。
  + 控制LED：根据收到的字符，点亮相应的LED：
    - 收到'G'，点亮绿色LED，关闭其他LED。
    - 收到'R'，点亮红色LED，关闭其他LED。
    - 收到'B'，点亮蓝色LED，关闭其他LED。
* 开启中断：完成数据处理后，重新开启总中断 (ENABLE\_INTERRUPTS)。

1. UART部分用直接地址方式实现(即不调用uart.c中的函数，其他部分如GPIO、中断设置可调用函数)

Main.c:

#define GLOBLE\_VAR

#include "includes.h" // 包含总头文件

**int main**(void)

{

// 定义uart寄存器相关地址

**volatile** uint32\_t\* RCC\_AHB2; // GPIO的A口时钟使能寄存器地址

**volatile** uint32\_t\* RCC\_APB1; // UART的2口时钟使能寄存器地址

**volatile** uint32\_t\* gpio\_ptr; // GPIO的A口基地址

**volatile** uint32\_t\* uart\_ptr; // uart2端口的基地址

**volatile** uint32\_t\* gpio\_mode; // 引脚模式寄存器地址=口基地址

**volatile** uint32\_t\* gpio\_afrl; // GPIO复用功能低位寄存器

**volatile** uint32\_t\* uart\_brr; // UART波特率寄存器地址

**volatile** uint32\_t\* uart\_isr; // UART中断和状态寄存器基地址

**volatile** uint32\_t\* uart\_cr1; // UART控制寄存器1基地址

**volatile** uint32\_t\* uart\_cr2; // UART控制寄存器2基地址

**volatile** uint32\_t\* uart\_cr3; // UART控制寄存器3基地址

**volatile** uint32\_t\* uart\_tdr; // UART发送数据寄存器

uint16\_t usartdiv; // BRR寄存器应赋的值

// 变量赋值

RCC\_APB1 = (**volatile** uint32\_t\*)0x40021058UL; // UART时钟使能寄存器地址

RCC\_AHB2 = (**volatile** uint32\_t\*)0x4002104CUL; // GPIO的A口时钟使能寄存器地址

gpio\_ptr = (**volatile** uint32\_t\*)0x48000000UL; // GPIOA端口的基地址

uart\_ptr = (**volatile** uint32\_t\*)0x40004400UL; // UART2端口的基地址

gpio\_mode = (**volatile** uint32\_t\*)0x48000000UL; // 引脚模式寄存器地址=口基地址

gpio\_afrl = (**volatile** uint32\_t\*)0x48000020UL; // GPIO复用功能低位寄存器

uart\_cr1 = (**volatile** uint32\_t\*)0x40004400UL; // UART控制寄存器1基地址

uart\_brr = (**volatile** uint32\_t\*)0x4000440CUL; // UART波特率寄存器地址

uart\_isr = (**volatile** uint32\_t\*)0x4000441CUL; // UART中断和状态寄存器基地址

uart\_tdr = (**volatile** uint32\_t\*)0x40004428UL; // UART发送数据寄存器

uart\_cr2 = (**volatile** uint32\_t\*)0x40004404UL; // UART控制寄存器2基地址

uart\_cr3 = (**volatile** uint32\_t\*)0x40004408UL; // UART控制寄存器3基地址

// 关总中断

DISABLE\_INTERRUPTS;

// 用户外设模块初始化

**gpio\_init**(LIGHT\_BLUE, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_RED, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_GREEN, GPIO\_OUTPUT, LIGHT\_OFF);

// 使能GPIOA和UART2的时钟

\*RCC\_APB1 |= (0x1UL << 17U); // UART2时钟使能

\*RCC\_AHB2 |= (0x1UL << 0U); // GPIOA时钟使能

// 将GPIO端口设置为复用功能

\*gpio\_mode &= ~((0x3UL << 4U) | (0x3UL << 6U));

\*gpio\_mode |= ((0x2UL << 4U) | (0x2UL << 6U));

// 选择引脚的端口复用功能

\*gpio\_afrl &= ~((0xFUL << 8U) | (0xFUL << 12U));

\*gpio\_afrl |= ((0x7UL << 8U) | (0x7UL << 12U));

// 暂时禁用UART功能

\*uart\_cr1 &= ~(0x1UL);

// 暂时关闭串口发送与接收功能

\*uart\_cr1 &= ~((0x1UL << 3U) | (0x1UL << 2U));

// 一位起始位，八位数据位

\*uart\_cr1 &= ~((0x1UL << 28U) | (0x1UL << 12U));

// 过采样因子为16

\*uart\_cr1 &= ~(0x1UL << 15U);

// 配置波特率

usartdiv = (uint16\_t)(SystemCoreClock / 115200);

\*uart\_brr = usartdiv;

// 初始化控制寄存器和中断状态寄存器、清标志位

\*uart\_isr = 0x0UL;

\*uart\_cr2 &= ~((0x1UL << 14U) | (0x1UL << 11U));

\*uart\_cr3 &= ~((0x1UL << 5U) | (0x1UL << 3U) | (0x1UL << 1U));

// 启动串口发送与接收功能

\*uart\_cr1 |= ((0x1UL << 3U) | (0x1UL << 2U));

// 开启UART功能

\*uart\_cr1 |= (0x1UL << 0U);

// 开放UART接收中断

\*uart\_cr1 |= (0x1UL << 5U);

**NVIC\_EnableIRQ**(USART2\_IRQn);

// 开总中断

ENABLE\_INTERRUPTS;

// 提示进入串口工具进行操作

**printf**("=====================================\n");

**printf**("CS212 LHL\n");

**printf**("=====================================\n");

return 0;

}

Isr.c:

#include "includes.h"

#define GLOBLE\_VAR

**volatile** uint32\_t\* uart\_isr = (**volatile** uint32\_t\*)0x4000441CUL; // UART中断和状态寄存器基地址

**volatile** uint32\_t\* uart\_rdr = (**volatile** uint32\_t\*)0x40004424UL; // UART接受数据寄存器

**volatile** uint32\_t\* uart\_tdr = (**volatile** uint32\_t\*)0x40004428UL; // UART发送数据寄存器

void **User\_SysFun**(uint8\_t ch);

// 程序名称：UART\_User\_Handler

// 触发条件：UART\_User串口收到一个字节触发

void **USART2\_IRQHandler**(void)

{

uint8\_t ch;

uint8\_t flag = 0;

uint32\_t t;

uint32\_t t1;

DISABLE\_INTERRUPTS; // 关总中断

// 接收一个字节的数据

**for**(t = 0; t < 0xFBBB; t++) // 一直查询缓冲区是否有数据

{

// 先判断isr状态位，再获取数据

**if**((\*uart\_isr) & (1 << 5U)) // 第五位为1

{

ch = \*uart\_rdr; // 从RDR寄存器取数

flag = 1;

\*uart\_isr &= ~(1 << 5U); // 第五位清零

break;

}

}

**if**(t >= 0xFBBB) // 超过指定次数

{

ch = 0xFF;

flag = 0;

}

**if**(flag) // 有数据

{

**if**((ch != 'G') **&&** (ch != 'R') **&&** (ch != 'B'))

{

**for**(t1 = 0; t1 < 0xFBBB; t1++) // 查询指定次数

{

// 发送缓冲区为空则发送数据

**if**((\*uart\_isr) & (1 << 7U)) // 检查第7位是否为1

{

\*uart\_tdr = (ch + 1); // 放到发送寄存器

break;

}

}

**if**(t1 >= 0xFBBB)

return; // 发送超时，发送失败

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

}

**else if**(ch == 'G')

{

**uart\_send\_string**(UART\_User, (uint8\_t \*)"绿灯亮");

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_ON);

}

**else if**(ch == 'R')

{

**uart\_send\_string**(UART\_User, (uint8\_t \*)"红灯亮");

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

**gpio\_set**(LIGHT\_RED, LIGHT\_ON);

}

**else if**(ch == 'B')

{

**uart\_send\_string**(UART\_User, (uint8\_t \*)"蓝灯亮");

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_ON);

}

}

ENABLE\_INTERRUPTS; // 开总中断

}

// 内部函数

void **User\_SysFun**(uint8\_t ch)

{

// (1) 收到的一个字节参与组帧

**if**(gcRecvLen == 0)

gcRecvLen = **useremuart\_frame**(ch, (uint8\_t\*)gcRecvBuf);

// (2) 字节进入组帧后，判断gcRecvLen=0？若为0，表示组帧尚未完成

**if**(gcRecvLen == 0)

goto User\_SysFun\_Exit;

// (3) 至此，gcRecvLen≠0,表示组帧完成，gcRecvLen为帧的长度

**if**(**strncmp**((char \*)(gcRecvBuf), (char \*)((MCU\_SECTOR\_NUM-1) \* MCU\_SECTORSIZE + MCU\_FLASH\_ADDR\_START), 16) != 0)

{

gcRecvLen = 0; // 恢复接收状态

goto User\_SysFun\_Exit;

}

// (4) 根据命令字节gcRecvBuf[16]进行跳转

**switch**(gcRecvBuf[16])

{

**case** 0:

**SYSTEM\_FUNCTION**((uint8\_t \*)(gcRecvBuf + 17));

gcRecvLen = 0; // 恢复接收状态

break;

**default**:

break;

}

User\_SysFun\_Exit:

return;

}

灯的显示和上面的一样，不多赘述。

工作原理：

1.主程序 main.c:

* 定义与初始化: 定义了多个指向硬件寄存器的指针变量，用于控制GPIO和UART功能。初始化这些指针指向相应的寄存器地址。
* 配置GPIO: 使能GPIOA和UART2的时钟，配置GPIO引脚为复用功能，以用于UART通信。
* 配置UART: 设置UART的控制寄存器、波特率寄存器等，配置波特率为115200，设置为8位数据位，1位起始位，16倍过采样。
* 启动UART功能: 开启UART的发送与接收功能，并使能UART接收中断。

2.中断服务程序 isr.c:

* UART接收中断处理: 在接收到一个字节的数据后，检查该数据是否为特定字符（'G', 'R', 'B'），根据不同的字符控制相应的LED灯。
* 发送响应字符: 如果接收到的字符不是特定字符，则返回接收字符的ASCII值加1，并发送该字符。
* 内部函数 User\_SysFun: 处理接收到的数据帧，进行相应的操作