实验 5 基于 USART 的串口通信

实验目的：利用 STM32F407 的 USART 模块实现自发自收数据，掌握 USART 的配置流程和串口中断服务程序的编写。

实验内容：

- 配置 USART 模块并编写中断服务程序
- 利用 USART 实现自发自收

5.1 USART 功能描述

USART 即通用同步异步收发器，能够灵活地与外部设备进行全双工数据交换，满足外部设备对工业标准 NRZ 异步串行数据格式的要求。UART 即通用异步收发器，它是在 USART 基础上裁剪掉了同步通信功能，同步和异步主要看其时钟是否需要对外提供。STM32F407 有 2 个 UART（通用异步收发器），4 个 USART（通用异步/同步收发器）。它们都具有串口通信功能，USART 支持同步单向通信和半双工单线通信；还支持 LIN（域互连网络）、智能卡协议，以及调制解调器操作 (CTS/RTS)。同时支持多处理器通信和 DMA 功能，使用 DMA 可实现高速数据通信。

5.2 串口库函数配置流程

串口设置的一般步骤可以总结为如下几个步骤：

（1）串口时钟使能，GPIO 时钟使能。

（2）设置引脚复用器映射：调用 GPIO_PinAFConfig 函数。

（3）GPIO 初始化设置：要设置模式为复用功能。

（4）串口参数初始化：设置波特率，字长，奇偶校验等参数。

（5）开启中断并且初始化 NVIC，使能中断（如果需要开启中断才需要这个步骤）。

（6）使能串口。

（7）编写中断处理函数：函数名格式为 USARTx_IRQHandler(x 对应串口号)。

与串口基本配置直接相关的固件库函数，这些函数定义主要分布在 stm32f4xx_usart.h 和 stm32f4xx_usart.c 文件中。

（1）串口时钟和 GPIO 时钟使能。

串口 1 是挂载在 APB2 下面的外设，所以使能函数为：

RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);

//使能 USART1 时钟

GPIO 时钟使能，因为使用的是串口 1，串口 1 对应着芯片引脚 PA9、PA10，所以只需要使能 GPIOA 时钟即可。

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE); //使能 GPIOA 时钟

（2）设置引脚复用器映射

调用函数为：

GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1);

//PA9 复用为 USART1 TX

GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1);

//PA10 复用为 USART1 RX

因为串口 1 使用 PA9 做为 TX，PA10 为 RX，所以要把 PA9 和 PA10 都映射到串口 1，需要调用两次函数 GPIO_PinAFConfig(XXX)。

（3）GPIO 端口模式设置：PA9 和 PA10 要设置为复用功能。

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 ;//GPIOA9

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用功能

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //速度 50MHz

GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //推挽复用输出

GPIO_Init(GPIOA,&GPIO_InitStructure); //初始化 PA9

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP; //上拉输入

GPIO_Init(GPIOA,&GPIO_InitStructure); //初始化 PA10

（4）串口参数初始化：设置波特率、字长、奇偶校验等参数

串口初始化是调用函数 USART_Init 来实现的，具体设置方法如下：

USART_InitStructure.USART_BaudRate = bound;

USART_InitStructure.USART_WordLength = USART_WordLength_8b;

//字长为 8 位数据格式

USART_InitStructure.USART_StopBits = USART_StopBits_1;//一个停止位

USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位

USART_InitStructure.USART_HardwareFlowControl =

USART_HardwareFlowControl_None;

USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

//收发模式

USART_Init(USART1, &USART_InitStructure); //初始化串口

（5）使能串口

使能串口调用函数 USART_Cmd 来实现，使能串口 1 方法如下：

USART_Cmd(USART1, ENABLE); //使能串口 1

（6）串口数据发送与接收。

STM32F4 的发送与接收都是通过数据寄存器 USART_DR 来实现的，这是一个双寄存器，包含了 TDR 和 RDR。当向该寄存器写数据的时候，串口就会自动发送，当收到数据的时候，也是存在该寄存器内。

STM32 库函数操作 USART_DR 寄存器发送数据的函数是：

void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);

通过该函数向串口寄存器 USART_DR 写入一个数据。

STM32 库函数操作 USART_DR 寄存器读取串口接收到的数据的函数是：

uint16_t USART_ReceiveData(USART_TypeDef* USARTx);

通过该函数可以读取串口接受到的数据。

（7）串口状态

串口的状态可以通过状态寄存器 USART_SR 读取。主要关注两个位，RXNE 和 TXE。

RXNE（读数据寄存器非空），当该位被置 1 的时候，就是提示已经有数据被接收到了，并且可以读出来了。这时要读取 USART_DR 送入内存变量保存，通过读 USART_DR 可以将该位清零，也可以向该位写 0，直接清除。

TXE（发送寄存器空），当 TDR 寄存器的内容已传输到移位寄存器时，该位由硬件置 1。如果 USART_CR1 寄存器中 TXEIE 位 = 1，则会生成中断。通过对 USART_DR 寄存器执行写入操作将该位清零。0：数据未传输到移位寄存器，1：数据传输到移位寄存器。

（8） 开启中断并且初始化 NVIC，使能相应中断

要开启串口中断必须要配置 NVIC 中断优先级分组，通过调用函数 NVIC_Init 来设置。

NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;//抢占优先级 1

NVIC_InitStructure.NVIC_IRQChannelSubPriority =1;　　//响应优先级 1

NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;　　//IRQ 通道使能

NVIC_Init(&NVIC_InitStructure);　　//根据指定的参数初始化 VIC 寄存器、

同时，我们还需要使能相应中断,使能串口中断的函数是：

void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState)

这个函数的第二个入口参数是标示使能串口的类型，也就是使能哪种中断，因为串口的中断类型有很多种。比如在接收到数据的时候（RXNE 读数据寄存器非空），我们要产生中断，那么开启中断的方法是：

USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//开启中断，接收到数据中断

我们在发送数据结束的时候（TC，发送完成）要产生中断，那么方法是：

USART_ITConfig(USART1，USART_IT_TC，ENABLE);

因为实验开启了串口中断，所以我们在系统初始化的时候需要先设置系统的中断优先级分组，是在 main 函数开头设置的，代码如下：

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置系统中断优先级分组 2

设置为 2 位抢占优先级，2 位响应优先级。

（9）获取相应中断状态

当使能了某个中断的时候，当该中断发生时，就会设置状态寄存器中的某个标志位。经常在中断处理函数中，要判断该中断是哪种中断，使用的函数是：

ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT)

比如使能了串口接收完成中断，那么当中断发生了，可以在中断处理函数中调用这个函数来判断到底是否是串口完成中断，方法是：

USART_GetITStatus(USART1, USART_IT_RXNE) != RESET

（10）中断服务函数

串口 1 中断服务函数为：

void USART1_IRQHandler(void) ;

当发生中断的时候，程序就会执行中断服务函数。然后在中断服务函数中添加相应的逻辑代码即可。

5.3 实验步骤

串口通信工程采用串口 1 自收发流程（需要用杜邦线连接 TX 和 RX 引脚，将 JP3 和 JP4 中间引脚相连即可），接收字符数据通过 LCD 显示，所以可以利用之前的 LCD 工程做为模板，配置工程过程如下：

1. 在 system/usart/usart.h 中添加宏和初始化函数声明。

```
#define USART_REC_LEN  200      //接收 Buffer 长度
·void XXX_Init(u32 bound);
```

2. 在 system/usart/usart.c 中添加接收缓冲区和控制传输标志。

注意：类型为全局变量。

```
u8 USART_RX_BUF[USART_REC_LEN];
u16 USART_TX_EN=1;
u16 USART_RX_STA=0;
```

3. 在 system/usart/usart.c 中添加初始化函数代码和中断服务函数。

```
void XXX_Init(u32 bound)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength=USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl
_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

     USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority =1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
    USART_Cmd(USART1, ENABLE);
 }
中断服务函数
void USART1_IRQHandler(void)
{
    u8 Res;

   if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
   {
        Res =USART_ReceiveData(USART1);
        //？？？补充代码
```

```
            //对应操作 USART_RX_STA++;
            USART_ClearITPendingBit(USART1, USART_IT_RXNE);
        }
    }
```

4. 主程序 main( )函数前添加

```
extern u16 USART_TX_EN;
extern u8    USART_RX_BUF[USART_REC_LEN];
extern u16 USART_RX_STA;
```

在 C 语言中，修饰符 extern 用在变量或者函数的声明前，用来说明“此变量/函数是在别的源文件中定义，要在此处引用”。

5. 在主程序 main( )中添加初始化代码，并添加相关头文件

```
char USART_SendBuf[]="Hello!";
int i;
int Len=strlen(USART_SendBuf);

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
delay_init(168);
LCD_GPIO_Init();
LCD_Init();
usart1_Init(115200);
```

6. 在主框架 while(1)中添加串口发送代码

```
 delay_ms(2000);
 if(USART_TX_EN)
 {
     for(i=0;i<Len;i++)
     {
         USART_SendData(USART1, USART_SendBuf[i]);
         while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)!=SET);
     }
     USART_TX_EN=0;
```

```
}
//？？？补充代码,判断是否发送完毕
LCD_Display_Words(0,0,USART_RX_BUF);
```

5.4 练习

在实验 5.3 的框架中补齐代码，在 LCD 显示通过 USART 自发自收的字符串“Hello!”，并添加代码实现持续发送和接收并在 LCD 换行显示。