实验 10　232 串口通信实验

前面几章介绍了 STM32 的 IO 口操作及中断。这一章我们将学习 STM32 的串口，教大家如何使用 STM32 的串口来发送和接收数据。本章将实现如下功能：STM32 通过串口和上位机的对话，STM32 在收到上位机发过来的字符串后，进行加 1 返回给上位机显示。 本章分为以下学习目标：

1、学会操作 STM32 的串口。

1.1 串口的操作步骤

串口做为单片机的重要外部接口，同时也是软件开发的重要调试手段。对于单片机学习来说，非常重要。而我们开发板使用的 STM32F103ZET6 最多可以提供 5 路串口。那么 STM32 的串口操作步骤是怎么样的呢？

1) 打开 GPIO 的时钟使能和 USART 的时钟使能。

2) 设置串口 IO 的 IO 口模式。 （一般输入是模拟输入， 输出是复用推挽输出）

3) 初始化 USART。 （包括设置波特率、数据长度、停止位、效验位等）

4) 如果使用中断接收的话，那么还要设置 NVIC 并打开中断使能。（即设置 设置它的中断优先级。 ）

1.2 V.35 库函数说明

1) RCC_APB2PeriphClockCmd()函数

开启时钟函数。相信大家对这个函数已经很熟悉了，我们这就不讲了。 我们要打开的时钟有两个一个 GPIO 口的时钟和 USART 的时钟。

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE); 两个函数分别打开了 GPIOA 和 USART1 的时钟(USART1 使用的是 PA9、PA10)

2) GPIO_Init()函数

这个函数相信大家也很熟悉了，我们设置的好的代码如下：

```
/*  配置 GPIO 的模式和 IO 口 */
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX              //串口输出
PA9
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;     //复用推挽输出
    GPIO_Init(GPIOA,&GPIO_InitStructure);  /* 初始化串口输入 IO */
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX          //串口输入
PA10
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;         //模拟
输入
    GPIO_Init(GPIOA,&GPIO_InitStructure); /* 初始化 GPIO */
```

3) USART_Init()函数

这个函数用于配置 USART 的设置，它拥有两个输入参数：

第一个参数是用来设置要选择的串口，我们要使用的是 USART1，所以 我们设置为：USART1。

第二个参数是传递一个结构体的指针，这个结构有 6 个成员

1．第一个成员是：USART_BaudRate，表示要设置的串口波特率，我们 可以设置我们想要的波特率，比如你要使用 9600 的时候，就设置为 9600。（在我们的例程中，波特率是通过初始化函数传递设置的，所 以等于传递的参数 baudRate）。

2．第二个成员是：USART_WordLength，表示要传送数据的长度，一般 是 8 位数据长度，所以我们设置为：USART_WordLength_8b。

3．第三个成员是：USART_StopBits，表示停止位的长度，我们设置为：USART_StopBits_1。

4．第四个成员是：USART_Parity，表示是否需要效验，我们设置为不需 要：USART_Parity_No。

5．第五个成员是： USART_HardwareFlowControl，表示是否需要硬件流， 所谓硬件流就是使用 DMA，我们这里不适用，所以我们设置为硬件 流失能：USART_HardwareFlowControl_None

6．第六个成员是：USART_Mode，表示你要设置的模式，我们要设置既 能接收又能发送， 所以设置为： USART_Mode_Tx | USART_Mode_Rx。

所以最后设置代码为：

```
USART_InitStructure.USART_BaudRate=9600;     //波特率设置为 9600   //波特率
    USART_InitStructure.USART_WordLength=USART_WordLength_8b;        //数据长 8 位
    USART_InitStructure.USART_StopBits=USART_StopBits_1;             //1 位停止位
    USART_InitStructure.USART_Parity=USART_Parity_No;                //无效验
    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None; //失能硬件流
    USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;  //开启发送和接受模式
    USART_Init(USART1,&USART_InitStructure);     /* 初始化 USART1 */
```

4) USART_Cmd()函数：

串口使能函数，它有两个个输入参数。 第一个参数是用来设置要设置的 USART，我们要打开的是 USART1， 所以我们设置为 USART1。 第二个参数是用来选择设置的状态，所以我们设置为：ENABLE。 所以设置的代码为：

```
/* 使能 USART1 */
USART_Cmd(USART1, ENABLE);
```

5) NVIC_Init(&NVIC_InitStructure)函数

用来设置中断的优先级和打开总中断。这个要输入一个结构体指针。 这个结构体的参数分别有四个成员：

第一个成员是 NVIC_IRQChannelPreemptionPriority，表示抢占优先 级的等级，我们设置为 0。

第二个成员是 NVIC_IRQChannelSubPriority，表示响应优先级的等 级，我们也设置为 0。

第三个成员是 NVIC_IRQChannel，表示选择你要设置的全局中断， 我们要设置的中断是 USART1 的中断， 所以我们设置为： USART1_IRQn。

第四个成员是 NVIC_IRQChannelCmd,表示要设置的状态,我们是 要打开中断的，所以我们设置为：ENABLE。

还有就是我们需要对中断进行一个分组，我们使用的是组 1，如下：

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

所以最后的设置如下：

```
/* 设置 NVIC 参数 */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;       //打开
USART1 的全局中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;    //抢
占优先级为 0
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;          //响应
优先级为 0
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;           //使能
    NVIC_Init(&NVIC_InitStructure);
```

6) USART_SendData()函数

这个函数是用来发送数据的，它有两个参数：

第一个参数是用来选择使用的 USART 我们要使用 USART1，所以选 择 USART1；

第二个参数是用来传递要发送的数据的，一般为一个 8 位数据。

注意：这个发送函数结束之后一定要接一个检测状态函数，用来检 测你的数据是否发送完成，如果不检测的话，传送回产生错误。

7) USART_GetFlagStatus()函数

这个函数用是用来检测状态的函数，它有两个参数：

第一个参数是用来选择要检测的 USART 我们要检测 USART1，所以 选择 USART1；

第二个参数是用来设置要检测的状态的， 我们要检测 USART 是否发 送完成，所以我们设置为：USART_FLAG_TC。 这个函数还有一个返回值，如果发送完成，

那么它返回 SET（SET 也就是非零），如果没有发送完成，那么它返回 RESET（即 0）。

8) USART_ITConfig()函数

是用来打开 USART 中断的函数，它有三个参数：

第一个参数是选择要打开的 USART，我们要使用 USART1，所以选择 USART1。

第二个参数用来选择要打开 USART 中断的哪个中断，我们这里是要打开接收中断，所以选择 USART_IT_RXNE。

最后一个参数用来设置设置的状态， 我们要打开所以选择 ENABLE。 所以设置如下：

USART_ITConfig(USART1, USART_IT_RXNE ,ENABLE);

9) USART 的中断函数

前面我们学习 NVIC 的时候，我们说过，在库函数中，每个中断的 中断函数名字都已经帮我们定义了好，一般放在启动文件中（大家可以 打开 startup_stm32f10x_hd.s 查看 264 行之后，都是帮起好的中断函数）。 而我们要使用 USART1 的中断函数叫做：void USART1_IRQHandler (void)；需要注意的是，因为我们中断函数只有一个，但是中断标志却有 多种，所以在中断函数中，最好确认检测一下相应的中断标志位，看看 产生中断的是否是你想要的中断。

10) USART_GetITStatus()函数

这个函数是获取中断标志状态函数，它有两个参数： 第一个参数是用来选择要读取的串口，我们要读取 USART1，所以 这个参数设置为：USART1。 第二个参数是选择要读取的中断标志位，我们要读取的是接收中断 的标志位，所以这个参数设置为：USART_IT_RXNE。 它还有一个返回值，如果中断标志设置了，那么它返回 SET(SET 也就是非零），如果中断标志没有设置，那么它返回 RESET（即 0） ； 所以我们读取的函数应该写为：

USART_GetITStatus(USART1, USART_IT_RXNE)。

11) USART_ReceiveData()函数

这个函数用来读取 USART 接收到的数据。它有一个参数。这个参数 是用来选择你要读取的 USART， 我们要读取 USATT1， 所以我们设置为：USART1。 这个函数通过返回一个 16 位的数据。 当然如果你是通过 8 位传送的， 那么它就返回一个 8 位的数据。

1.3 串口初始化函数

```
/*******************************************************************************
*
* 函 数 名          : usart_init
* 函数功能          : 串口初始化函数
* 输    入          : 无
* 输    出          : 无
*******************************************************************************
/
void usart_init()
{
    GPIO_InitTypeDef GPIO_InitStructure;  //声明一个结构体变量，用来初始化 GPIO

    USART_InitTypeDef   USART_InitStructure;    //串口结构体定义

    NVIC_InitTypeDef NVIC_InitStructure;//中断结构体定义

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);      //打开时钟

    /*  配置 GPIO 的模式和 IO 口 */
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX                  //串口输出 PA9
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
```

```
GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;        //复用推挽输出
GPIO_Init(GPIOA,&GPIO_InitStructure);   /*  初始化串口输入 IO */
GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX                 //串口输入 PA10
GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;              //模拟输入
GPIO_Init(GPIOA,&GPIO_InitStructure); /*  初始化 GPIO */



USART_InitStructure.USART_BaudRate=9600;     //波特率设置为 9600    //波特率
USART_InitStructure.USART_WordLength=USART_WordLength_8b;         //数据长 8 位
USART_InitStructure.USART_StopBits=USART_StopBits_1;               //1 位停止位
USART_InitStructure.USART_Parity=USART_Parity_No;                        //无效验
USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None; //
失能硬件流
USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;    //开启发送和接
受模式
USART_Init(USART1,&USART_InitStructure); /*  初始化 USART1 */
USART_Cmd(USART1, ENABLE);                  /*  使能 USART1 */
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//使能或者失能指定的 USART 中断  接
收中断
USART_ClearFlag(USART1,USART_FLAG_TC);//清除 USARTx 的待处理标志位



/*  设置 NVIC 参数  */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;            //打开 USART1 的全局中断
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;       //抢占优先级为 0
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;                //响应优先级为 0
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                   //使能
NVIC_Init(&NVIC_InitStructure);
```

```
}
```

1.4 串口接收函数

```
void USART1_IRQHandler(void)     //串口 1 中断函数
{
    static u8 k;
    USART_ClearFlag(USART1,USART_FLAG_TC);
    if(USART_GetITStatus(USART1,USART_IT_RXNE)!=Bit_RESET)//检查指定的 USART 中断发生与否
    {
        k=USART_ReceiveData(USART1);
        k++;
        USART_SendData(USART1,k);//通过外设 USARTx 发送单个数据
        while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==Bit_RESET);
    }
}
```

我们定义了一个 8 位数据变量，当上位机发送一个数据的时候加 1 后在发送给上位机进行显示。

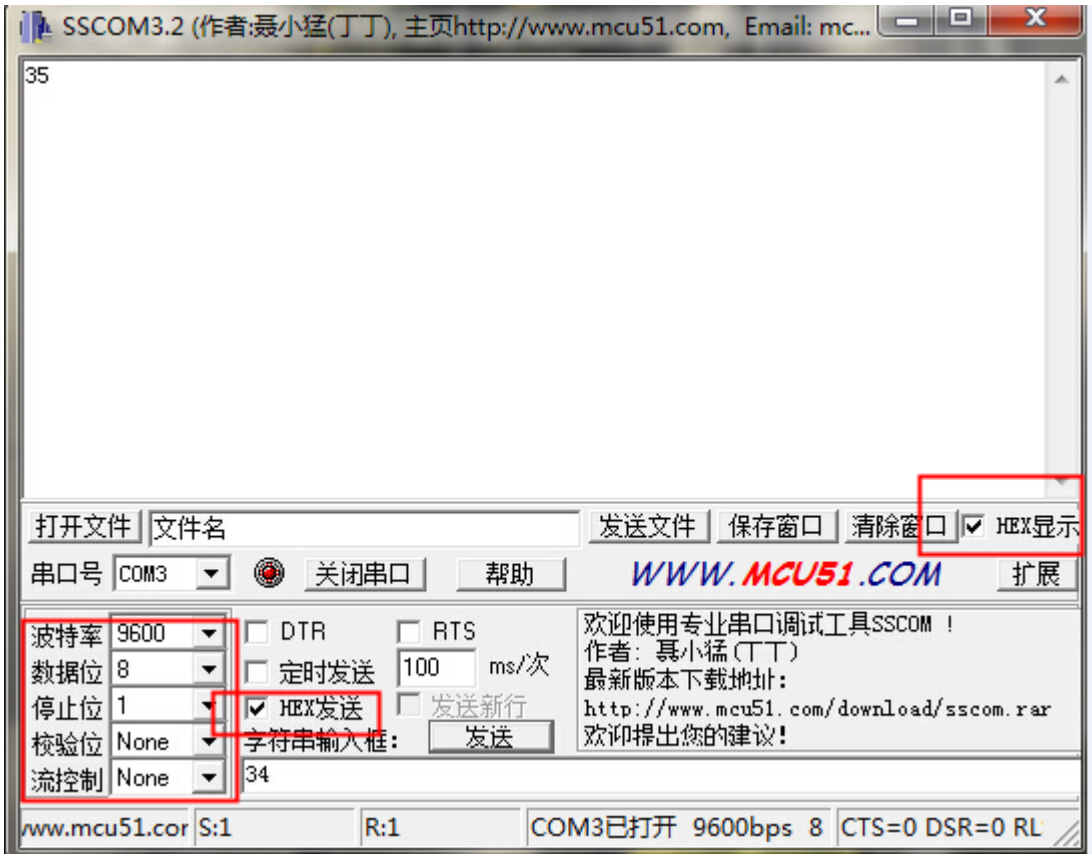
1.5 程序主函数

```
/*******************************************************************************
* Function Name    : main
* Description      : Main program.
* Input            : None
* Output           : None
* Return           : None
*******************************************************************************/
int main()
{
    usart_init();//串口 1 初始化
```

```
    while(1);
}
```

这 个主函数的效果是，设置串口波特率为 9600，在串口助手设置 HEX 发送和显示，发送 16 进制数后加 1 进行显示。

1.6 串口助手设置



当程序下载进去后，打开串口，对 DTR 前进行勾选，然后在取消。再通过发送字符即可以显示。