

实验 12 printf 输出实验

前面介绍了 STM32 的串口通信。这一章我们将学习利用串口进行 printf 重定向，教大家如何使用开发板串口实现 C 语言内的 printf 输出。本章将实现如下功能：STM32 通过串口和上位机的对话，STM32 向上位机发送字符串显示。本章分为以下学习目标：

1、学会 printf 重定向。

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、校验位等）
- 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_HardwareFlowContr
ol_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 产生中断的是不是你想要的中断。

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 USART1， 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 在程序中使用 C 语言中的 printf()函数

大家一开始学习 C 语言的时候， 都接触过 printf()函数， 使用起来很方便， 我们是否可以在我们的程序使用这个函数来作为串口的输出呢？ 其实是可以的。要是用 printf()函数， 就需要将它重新定向一下。怎么做呢：

1) 第一步。我们需要重新写 int fputc(int ch, FILE *f)这么个函数， 我们将它 转化为 STM32 串口输出的函数如下：

```
int fputc(int ch, FILE *p) //函数默认的， 在使用 printf 函数时自动调用
{
    USART_SendData(USART1, (u8)ch);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    return ch;
}
```

通过上面库函数的学习， 我们可以知道， 这个函数其实就是用来设置通 过 USART1 发送一个 8 位的数据。

2) 第二步。添加它的头文件， 也就是 stdio.h, 接下来的设置我们有两个方 法：

1. 第一个方法是： 打开 KEIL 的 Target Options, 选择里面的 Target 位置 选择 Use MicroLIB。 然后编译就可以了。

2. 第二个方法的呢， 再添加一下代码

```
#pragma import(__use_no_semihosting)
/* 标准库需要的支持函数 */
struct __FILE
{
    int handle;
```

```

/* Whatever you require here. If the only file you are using is */
/* standard output using printf() for debugging, no file handling */
/* is required. */
};

/* FILE is typedef' d in stdio.h. */
FILE __stdout;

/* 定义_sys_exit() 以避免使用半主机模式 */
_sys_exit(int x)
{
x = x;
}

```

然 后编译，就可以使用了。

1.4 printf 初始化函数

```

/*****
*****

* 函 数 名      : printf_init
* 函数功能      : IO 端口及串口 1，时钟初始化函数
* 输    入      : 无
* 输    出      : 无
*****
*****/

void printf_init() //printf 初始化
{
    GPIO_InitTypeDef GPIO_InitStructure; //声明一个结构体变量，用来初始化
GPIO
    NVIC_InitTypeDef NVIC_InitStructure; //中断结构体定义
    USART_InitTypeDef USART_InitStructure; //串口结构体定义

```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_USART1|  
RCC_APB2Periph_AFIO,ENABLE);
```

```
GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX  
GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;  
GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;  
GPIO_Init(GPIOA,&GPIO_InitStructure);  
GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX  
GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;  
GPIO_Init(GPIOA,&GPIO_InitStructure);
```

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);  
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;  
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;  
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;  
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;  
NVIC_Init(&NVIC_InitStructure);
```

```
USART_InitStructure.USART_BaudRate=9600;    //波特率设置为 9600  
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_HardwareFlowContr  
ol_None;  
USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;  
USART_Init(USART1,&USART_InitStructure);  
USART_Cmd(USART1, ENABLE);
```



```

        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
    }

```

1.5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);

    }
}

```

1.6程序主函数

```

/*****
* Function Name   : main
* Description     : Main program.
* Input          : None
* Output         : None
* Return         : None
*****/

int main()

```

```

{
    u32 d1=123;

    float d2=12.34;

    printf_init(); //printf 串口初始化
    while(1)
    {
        printf("深圳普中科技有限公司\r\n");

        printf("www.prechin.com\r\n");

        printf("to d1 a int number %d\r\n",d1); //整型数
        printf("to d2 a float number %f\r\n",d2); //浮点数

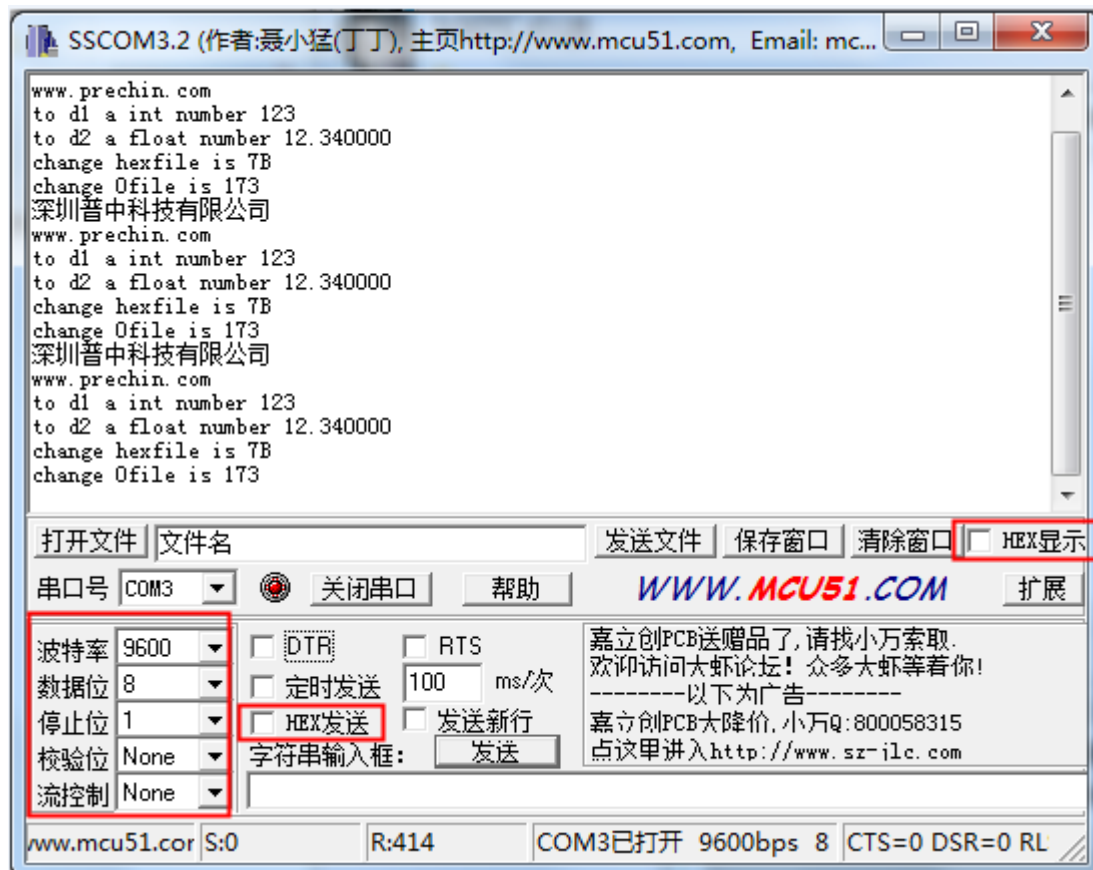
        printf("change hexfile is %X\r\n",d1); //16 进制
        printf("change Ofile is %o\r\n",d1); //八进制

        delay_ms(3000);
    }
}

```

这个主函数的效果是，设置串口波特率为 9600，在串口助手取消设置 HEX 发送和显示。

1.7 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。