浅谈使用DMA时需要的不断使能失能的原因

背景：在使用串口+IDLE（空闲中断）+DMA时，对于DMA的不断使能与失能不太明白，花了一点时间明白了其中的一些皮毛，拿来和大家分享。

在介绍时会引用一些本人之前写的一些代码，将写这些代码的文章链接放在下面。

文章一：基于STM32F407标准库串口DMA+空闲中断

文章二：基于STM32F407HAL库串口DMA+空闲中断

HAL库的基础就是标准库，所以我们先拿**标准库**来进行举例说明。

首先看为什么要使能、失能，依据在哪？（下图摘自**STM32F4xx中文参考手册**）

9.5.6 DMA 数据流 x 数据项数寄存器 (DMA_SxNDTR) (x = 0..7)

DMA stream x number of data register

偏移地址：0x14 + 0x18 × *数据流编号*

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:16 保留，必须保持复位值。

位 15:0 **NDT[15:0]**：要传输的数据项数目 (Number of data items to transfer)

要传输的数据项数目（0 到 65535）。只有在禁止数据流时，才能向此寄存器执行写操作。使能数据流后，此寄存器为只读，用于指示要传输的剩余数据项数。每次 DMA 传输后，此寄存器将递减。

传输完成后，此寄存器保持为零（数据流处于正常模式时），或者在以下情况下自动以先前编程的值重载：

- 以循环模式配置数据流时。
- 通过将 EN 位置“1”来重新使能数据流时

如果该寄存器的值为零，则即使使能数据流，也无法完成任何事务。

看了黄色字体标注后，再看摘自**文章一**的代码

```c
#include "stm32f4xx_it.h"
#include "bsp_usart.h"

void USART_IRQHandler(void)
{
    uint16_t temp;
    if(USART_GetFlagStatus(USART, USART_FLAG_IDLE) != RESET)
    {
        temp = USART1->SR;
        temp = USART1->DR;//清除IDLE中断

        temp = DMA_GetCurrDataCounter(USART_RX_DMA_STREAM);//获取剩余的接收数据的量

        DMA_Cmd(USART_RX_DMA_STREAM, DISABLE);

        DMA_SetCurrDataCounter(USART_RX_DMA_STREAM, RX_MAX_LEN);//设置需要接收的量

        DMA_Cmd(USART_RX_DMA_STREAM, ENABLE);


        DMA_Cmd(USART_TX_DMA_STREAM, DISABLE);

        DMA_SetCurrDataCounter(USART_TX_DMA_STREAM, RX_MAX_LEN - temp);//设置需要发送的量

        DMA_Cmd(USART_TX_DMA_STREAM, ENABLE);

        while( USART_GetFlagStatus(USART, USART_FLAG_TC) == RESET );//必须等待，否则最后一位数据错误

        DMA ClearFlag(USART TX DMA STREAM, DMA FLAG TCIF7); //清除DMA传输完成标志
```

```
30
31          USART_ClearFlag(USART,USART_FLAG_TC);     //清除传输完成标志
32      }
33  }
```

以上代码就是老老实实根据手册要求编写的。读DMA 数据流 x 数据项数寄存器 (DMA_SxNDTR)寄存器不需要失能；设置发送多少，下次接收多少需要失能使能，没有问题！

但是，仔细看过**文章一**对DMA的配置的会发现，用于接收数据的DMA配置为循环模式，用于发送数据的DMA为正常模式即一次模式。这样就有疑问了，我用于接收数据的DMA配置为循环模式，我为什么还要在空闲中断中还要给他设置接收多少数据呢，它自己不是会循环吗？

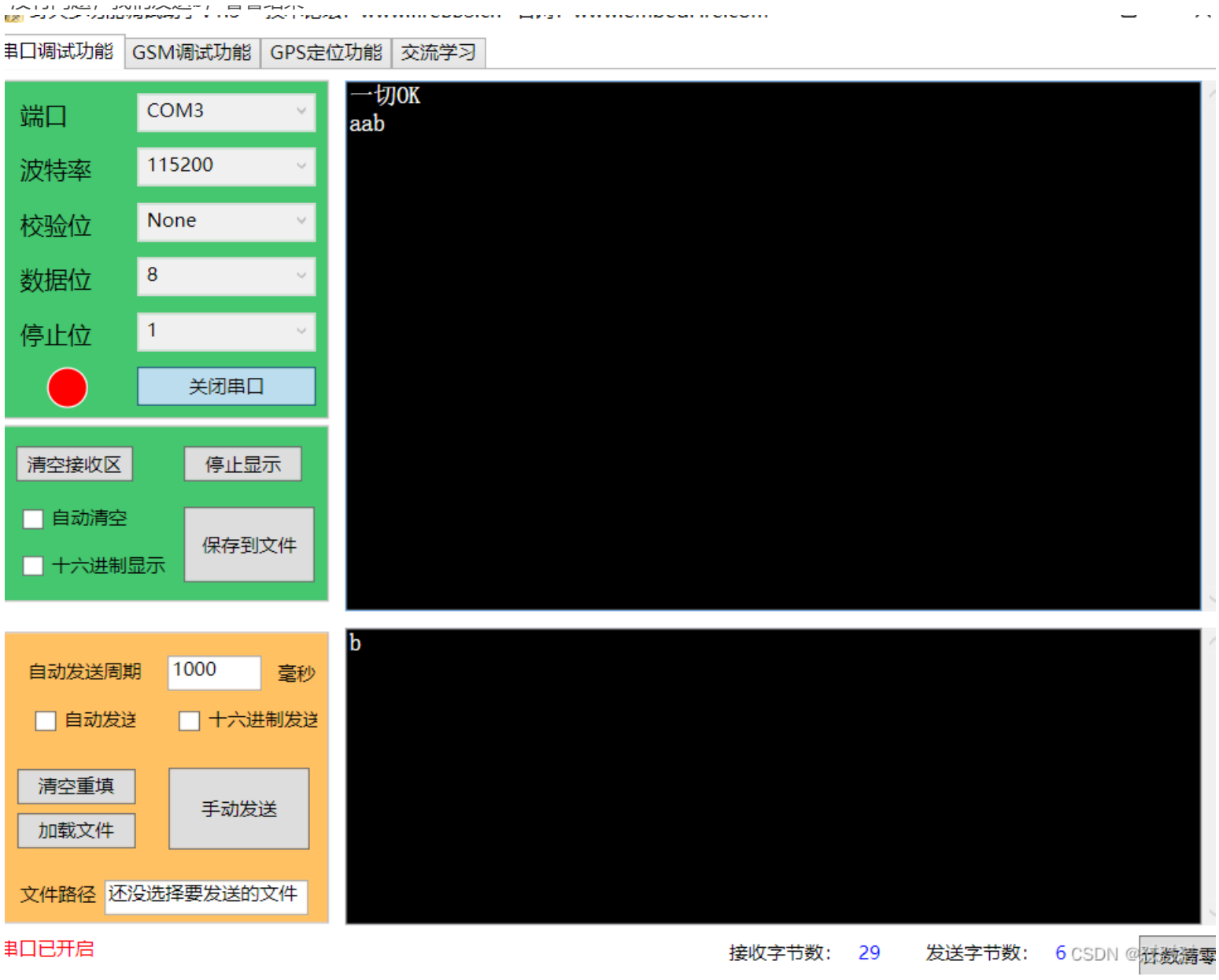
那么有了如下代码

```c
#include "stm32f4xx_it.h"
#include "bsp_usart.h"

void USART_IRQHandler(void)
{
    uint16_t temp;
    if(USART_GetFlagStatus(USART, USART_FLAG_IDLE) != RESET)
    {
        temp = USART1->SR;
        temp = USART1->DR;//清除IDLE中断

        temp = DMA_GetCurrDataCounter(USART_RX_DMA_STREAM);//获取剩余的接收数据的量

        //DMA_Cmd(USART_RX_DMA_STREAM, DISABLE);
        //
        //DMA_SetCurrDataCounter(USART_RX_DMA_STREAM, RX_MAX_LEN);//设置需要接收的量
        //
        //DMA_Cmd(USART_RX_DMA_STREAM, ENABLE);


        DMA_Cmd(USART_TX_DMA_STREAM, DISABLE);

        DMA_SetCurrDataCounter(USART_TX_DMA_STREAM, RX_MAX_LEN - temp);//设置需要发送的量

        DMA_Cmd(USART_TX_DMA_STREAM, ENABLE);

        while( USART_GetFlagStatus(USART, USART_FLAG_TC) == RESET );//必须等待，否则最后一位数据错误

        DMA_ClearFlag(USART_TX_DMA_STREAM, DMA_FLAG_TCIF7); //清除DMA传输完成标志

        USART_ClearFlag(USART,USART_FLAG_TC);     //清除传输完成标志
    }
}
```

那我们看看结果：

第一次发送a

没有问题，我们发送b，看看结果



会发现明明发送的为b，为什么发过去的是ab。

我们进行更改代码。看看到底是怎么回事。

```
void USART_IRQHandler(void)
{
    uint16_t temp;
    if(USART_GetFlagStatus(USART, USART_FLAG_IDLE) != RESET)
    {
        temp = USART1->SR;
        temp = USART1->DR;//清除IDLE中断

        temp = DMA_GetCurrDataCounter(USART_RX_DMA_STREAM);//获取传输中剩余的单元数

        printf("传输中剩余的单元数=%d\n",temp);

        //DMA_Cmd(USART_RX_DMA_STREAM, DISABLE);

        //DMA_SetCurrDataCounter(USART_RX_DMA_STREAM, RX_MAX_LEN);//设置需要接收的量

        //DMA_Cmd(USART_RX_DMA_STREAM, ENABLE);


        DMA_Cmd(USART_TX_DMA_STREAM, DISABLE);

        DMA_SetCurrDataCounter(USART_TX_DMA_STREAM, RX_MAX_LEN - temp);//设置需要发送的量

        DMA_Cmd(USART_TX_DMA_STREAM, ENABLE);

        while( USART_GetFlagStatus(USART, USART_FLAG_TC) == RESET );//必须等待，否则最后一位数据错误

        DMA_ClearFlag(USART_TX_DMA_STREAM, DMA_FLAG_TCIF7); //清除DMA传输完成标志

        USART_ClearFlag(USART,USART_FLAG_TC);     //清除传输完成标志

        for(int i=0;i<6;i++)
            printf("Rec[%d]=%c ",i,Rec[i]);

        printf("\n");
    }
}
```

老样子，先发送a

然后发送b

会发现b到了数组的第二个而非第一个，这也就解释为什么我们先发送a再发送b的结果是发送了aab。

回过头我们继续看一下手册为什么是这样?

位 31:16 保留，必须保持复位值。

位 15:0 **NDT[15:0]**：要传输的数据项数目 (Number of data items to transfer)

要传输的数据项数目（0 到 65535）。只有在禁止数据流时，才能向此寄存器执行写操作。使能数据流后，此寄存器为只读，用于指示要传输的剩余数据项数。每次 DMA 传输后，此寄存器将递减。

传输完成后，此寄存器保持为零（数据流处于正常模式时），或者在以下情况下自动以先前编程的值重载：

- 以循环模式配置数据流时。
- 通过将 EN 位置“1”来重新使能数据流时

如果该寄存器的值为零，则即使使能数据流，也无法完成任何事务。

看绿色标注的，仔细思考**何为传输完成**？我们设置DMA接收255个数据，当它接收完255个数据，才算接收完成，才可以重装载，如果让它完成重装载数据就正确了吗？**为什么在第二次输入b时Rec的第二位为b而不是Rec的第一位为b**？这是因为我们使能了存储器地址递增，所以b会到Rec的第二个位置，这就是导致我们数据错误的原因，也是为什么我们要失能DMA的原因——当**接收数据的数量小于为我们设定时让指向存储器的指针回到存储器的起始地址 。**当然，失能再使能也可以使得(DMA_SxNDTR）寄存器的值恢复置我们最开始设定的。

用于发送数据的DMA的存储器地址也是递增模式，但是由于我们设置为普通模式即一次模式，因此我们必须失能再使能来改变(DMA_SxNDTR）寄存器的值，所以我们这里不讨论失能再失能对它的存储器地址递增的影响情况。

接下来是**HAl库**

下面代码摘自**文章二**

```
void USER_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart->Instance == USART1)
    {
        if(__HAL_UART_GET_FLAG(&huart1, UART_FLAG_IDLE) != RESET)
        {
             __HAL_UART_CLEAR_IDLEFLAG(&huart1); //清除IDLE标志

            uint8_t Len = 255 - __HAL_DMA_GET_COUNTER(&hdma_usart1_rx);

            HAL_UART_DMAStop(&huart1);//停止DMA，为了重新设置DMA发送多少数据

            HAL_UART_Transmit_DMA(&huart1, (uint8_t *)pData, Len);

            HAL_UART_Receive_DMA(&huart1, (uint8_t *)pData, 255);
        }
    }
}
```

现在在看这个HAL库中的代码就很容易理解了。HAL_UART_DMAStop(&huart1);不仅停止了发送数据的DMA，更停止了接收数据的DMA。当然，这里最重要的是失能接收数据的DMA，**使得接收数据的数量小于为我们设定时可以让指向存储器的指针回到存储器位置的起始地址**。

总之，关闭再开启DMA不仅仅是为了给用于发送数据的DMA设置发送多少数据，而改变DMA 数据流 x 数据项数寄存器 (DMA_SxNDTR)寄存器的值，更是为了使得当接收到的数据小于我们设定的数据大小时用于接收数据的DMA所配置的存储器地址重新开始递增（指针指向存储器的起始地址），否则将会把原来的值也发送出去。
