

一、本次任务

了解串口协议和RS-232标准，以及RS232电平与TTL电平的区别；了解"USB/TTL转232"模块（以CH340芯片模块为例）的工作原理。使用HAL库（或标准库）方式，设置USART1 波特率为115200，1位停止位，无校验位，分别采用中断方式、DMA方式完成下列任务：

STM32系统给上位机（win10）连续发送“hello windows! ”；当上位机给stm32发送字符“stop”后，stm32暂停发送“hello windows! ”；发送一个字符“start”后，stm32继续发送；

二、理论

一、串口协议

1. 什么是串口通信

- 1、串口通信属于基层基本性的通信规约，收发双方事先规定好通信参数。
- 2、它自己本身不会去协商通信参数，需要通信前通信双方事先约定好通信参数来进行通信。
- 3、因此，若是收发方的任何一个关键参数设置错误，都会导致通信失败。譬如波特率调错了，发送方发送没问题，接收方也能接收，但是接收到全是乱码。
- 4、信息以二进制流的方式在信道上传输，串口通信的发送方每隔一定时间（时间固定为1/波特率，单位是秒）将有效信息（1或者0）放到通信线上去，逐个二进制位的进行发送。
- 5、接收方通过定时（起始时间由读到起始位标志开始，间隔时间由波特率决定）读取通信线上的电平高低来区分发送给我的是1还是0。依次读取数据位、奇偶校验位、停止位，停止位就表示这一个通信单元（帧）结束，然后中间是不定长短的非通信时间（发送方有可能紧接着就发送第二帧，也可能半天都不发第二帧，这就叫异步通信），下来就是第二帧……
- 6、通过串口不管发数字、还是文本还是命令还是什么，都要先对发送内容进行编码，编码成二进制再进行逐个位的发送。
- 7、串口发送的一般都是字符，一般都是ASCII码编码后的字符，所以一般设置数据位都是8，方便刚好一帧发送1个字符。

2. 串口协议

串口通信指两个或两个以上的设备使用串口按位（bit）发送和接收字节。可以在使用一根线发送数据的同时用另一根线接收数据。串口通信协议就是串口通讯时共同遵循的协议。协议的内容是每一个bit 所代表的意义。常用的串口通信协议 有以下几种

- 1 RS-232（ANSI/EIA-232标准）只支持点对点，最大距离 50英尺。最大速度为128000bit/s，距离越远 速度越慢。支持全双工（发送同时也可接收）。
- 2 RS-422（EIA RS-422-AStandard），支持点对多一条平衡总线上连接最多10个接收器 将传输速率提高到10Mbps，传输距离延长到4000英尺（约1219米），所以在100kbps速率以内，传输距离最大。支持全双工（发送同时也可接收）。
- 3 RS-485（EIA-485标准）是RS-422的改进，支持多对多（2线连接），从10个增加到32个，可以用超过4000英尺的线进行串行通行。速率最大10Mbps。支持全双工（发送同时也可接收）。2线连接时 是半双工状态。

3. RS-232

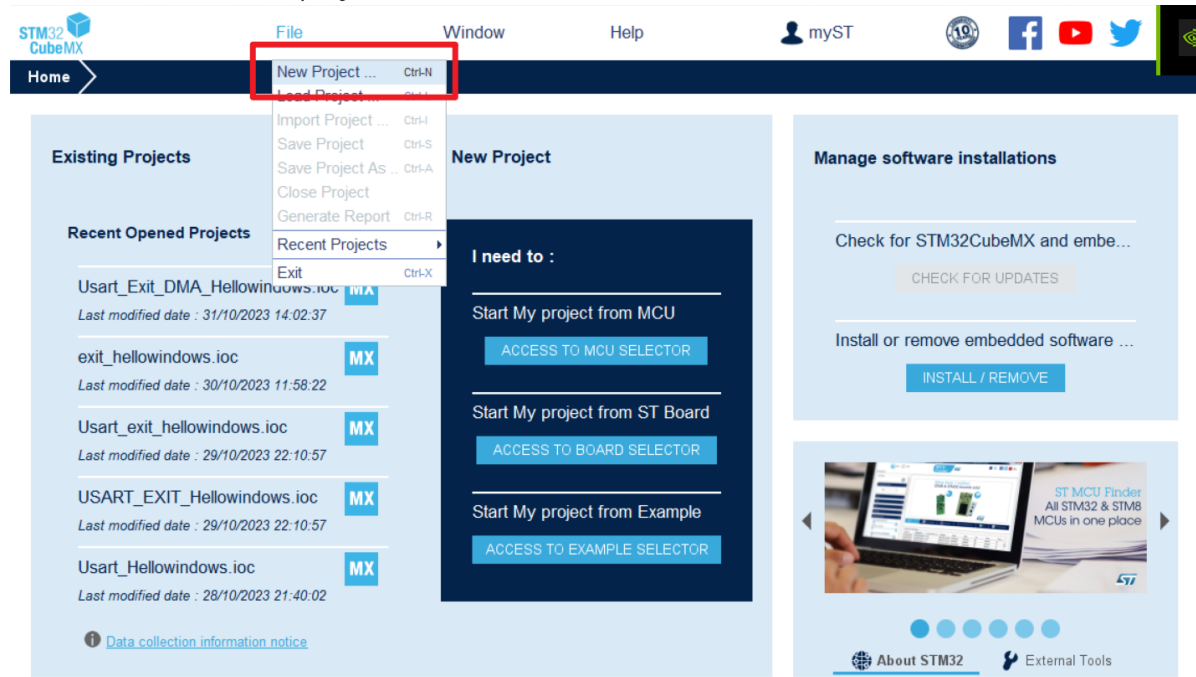
RS-232标准接口（又称EIA RS-232）是常用的串行通信接口标准之一，它是由美国电子工业协会（EIA）联合贝尔系统公司、调制解调厂家及计算机终端生产厂家于1970年共同制定，其全名是“数据终端设备（DTE）和数据通信设备（DCE）之间串行二进制数据交换接口技术标准”。

三、实操

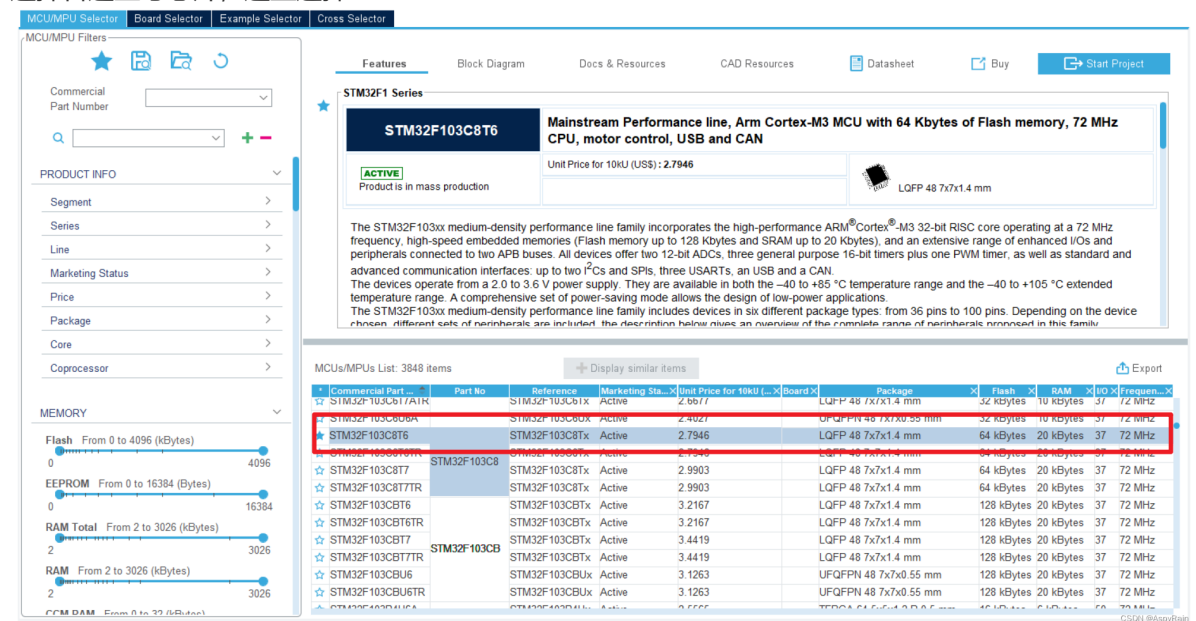
1. 中断方式

I 创建项目

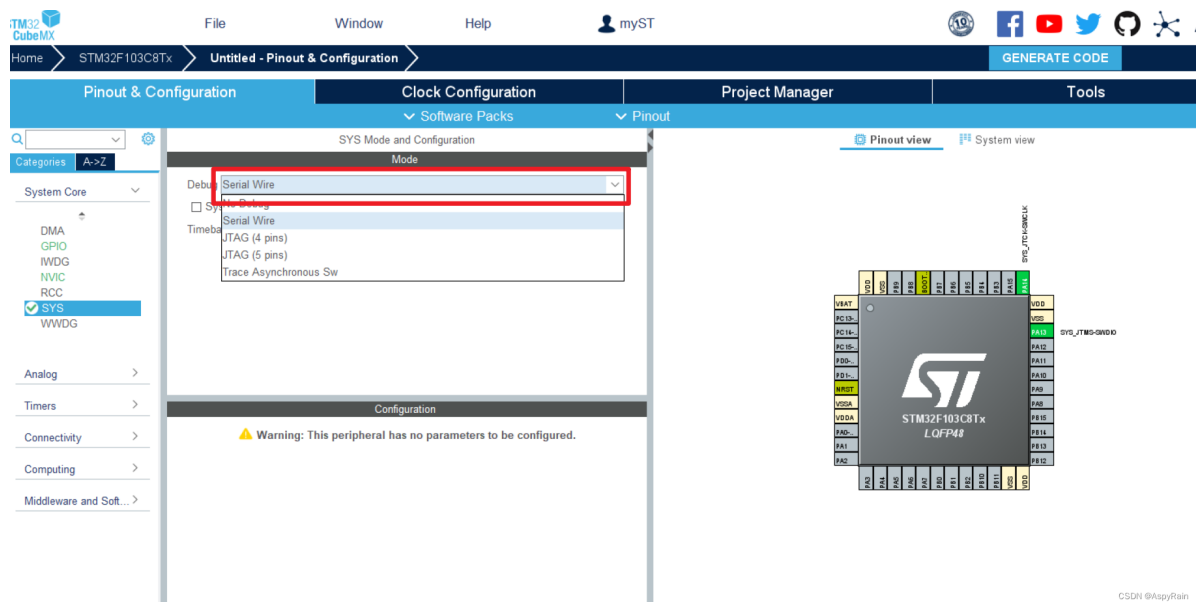
打开CubeMX， 点击new project新建项目



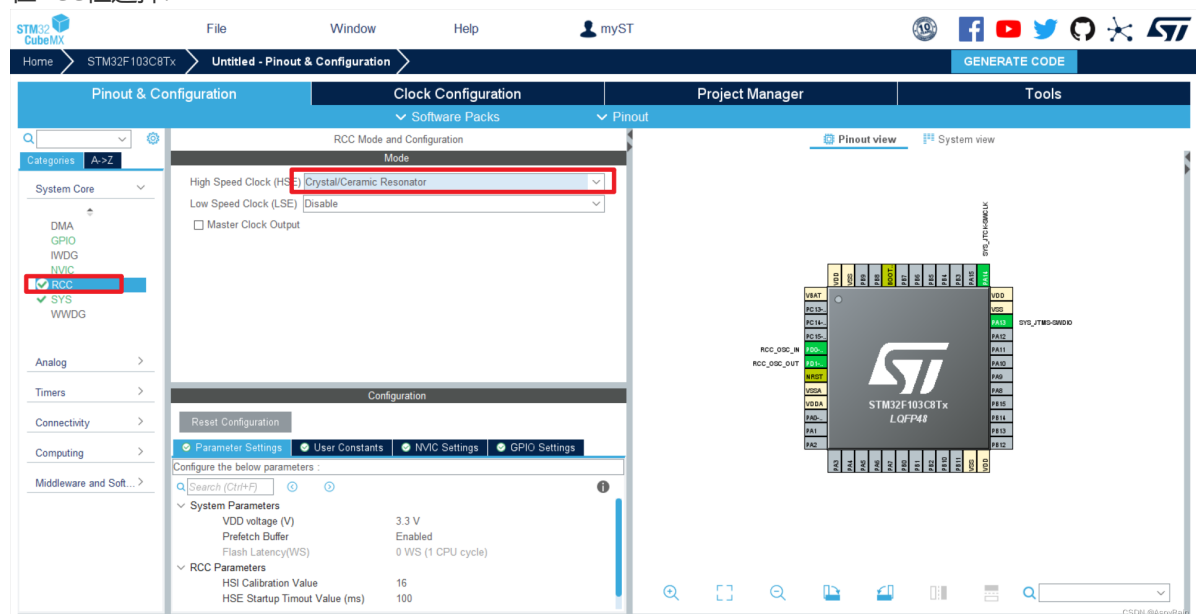
选择合适型号芯片， 这里选择stm32f103c8t6



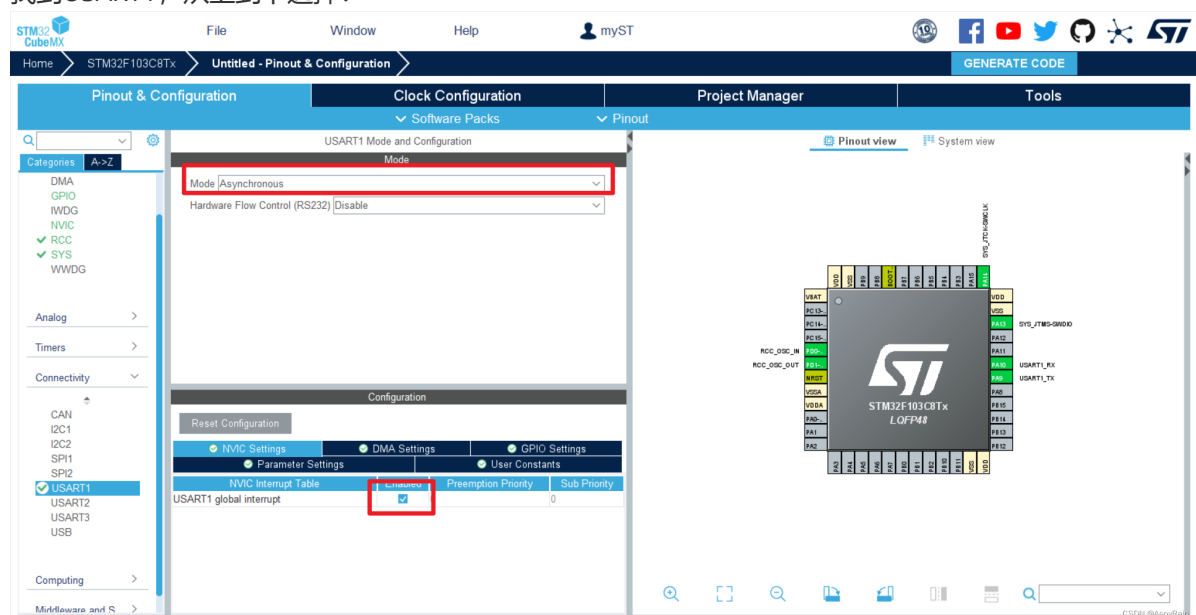
在SYS栏选择:



在RCC栏选择:



找到USART1, 从上到下选择:

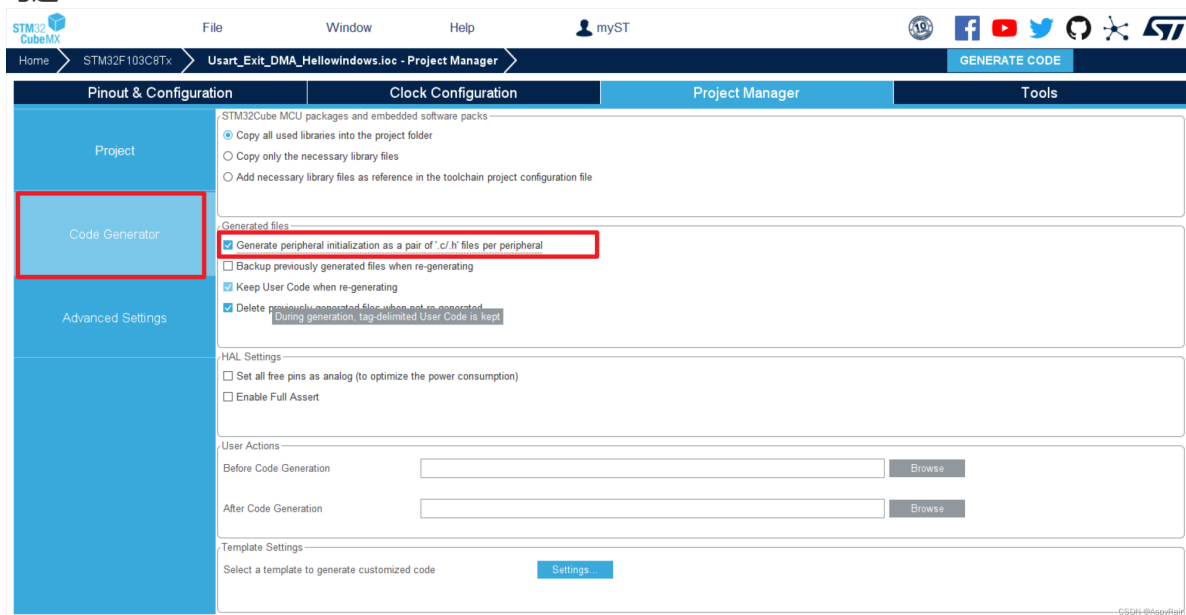


选择mdkIDE:



点击顶部project manager

勾选:



然后就可以点击GENERATE CODE生成项目了

II 编程

找到main.c

在/* USER CODE BEGIN PM */加入全局变量定义

```
char c;
char message[]="hello windows\n";//输出信息
char tips[]="CommandError\n";//提示1
char tips1[]="Start\n";//提示2
char tips2[]="Stop\n";//提示3
char flag='d';
#define MAX_BUFFER_SIZE 255 // 设置最大缓冲区大小

char rx_buffer[MAX_BUFFER_SIZE]; // 接收缓冲区
uint16_t rx_buffer_index = 0; // 缓冲区索引
```

在/* USER CODE BEGIN 2 */里使串口开始接收中断

```
HAL_UART_Receive_IT(&huart1, (uint8_t *)&c, 1);
```

在while里面添加:

```

if(flag=='u'){
    //发送信息
    HAL_UART_Transmit(&huart1, (uint8_t *)&message,
strlen(message),0xFFFF);
    //延时
}
HAL_Delay(1000);

```

在/* USER CODE END 4 */里添加flag的翻转函数和重写中断回调函数

```

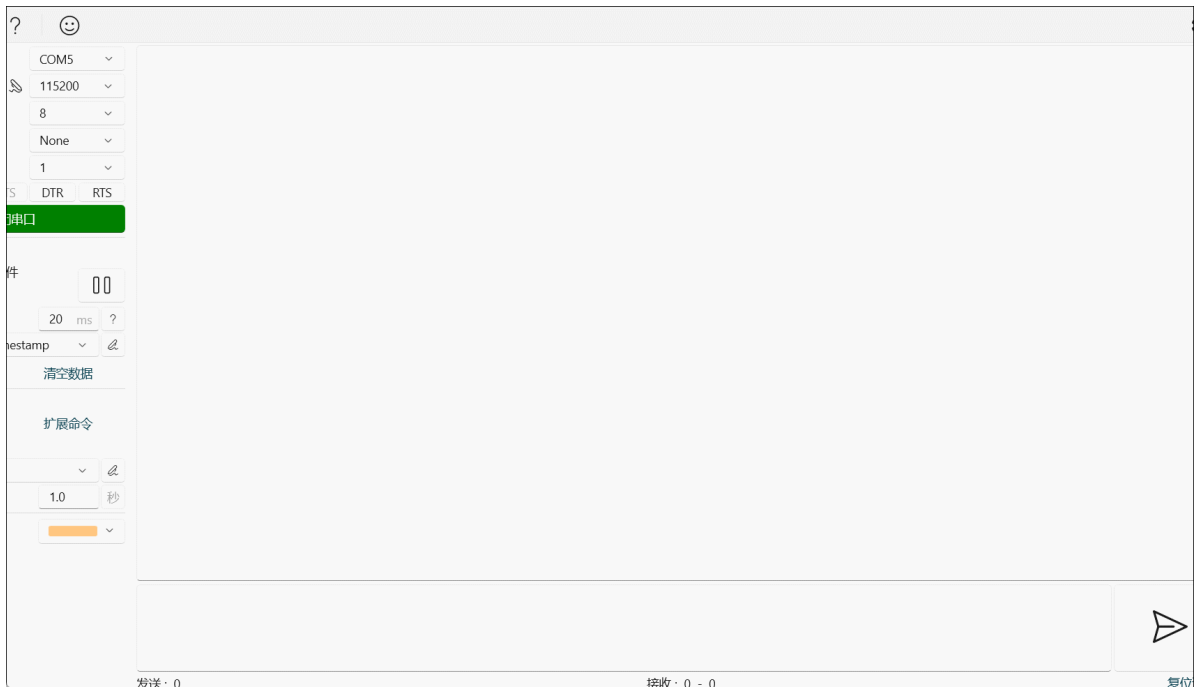
void toggle_flag(char now_flag){
    flag=now_flag;
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (c == '\n') {
        // 收到换行符 '\n' 或回车符 '\r', 表示接收完整的字符串
        if (strcmp(tips1, rx_buffer, strlen(tips1)-2) == 0) {
            toggle_flag('u');
            HAL_UART_Transmit(&huart1, (uint8_t *)&tips1, strlen(tips1), 0xFFFF);
        } else if (strcmp(tips2, rx_buffer, strlen(tips2)-2) == 0) {
            toggle_flag('d');
            HAL_UART_Transmit(&huart1, (uint8_t *)&tips2, strlen(tips2), 0xFFFF);
        } else {
            toggle_flag('d');
            HAL_UART_Transmit(&huart1, (uint8_t *)&tips, strlen(tips), 0xFFFF);
        }
        rx_buffer_index = 0; // 重置缓冲区索引
        memset(rx_buffer, 0, sizeof(rx_buffer));
    }
    else if (rx_buffer_index < MAX_BUFFER_SIZE - 1) {
        // 将字符存储在缓冲区中
        rx_buffer[rx_buffer_index++] = c;
    }

    // 继续接收下一个字符
    HAL_UART_Receive_IT(&huart1, (uint8_t *)&c, 1);
}

```

III 编译烧录



2. DMA模式

I 原理

① DMA

DMA代表Direct Memory Access，直接内存访问，它是计算机系统的一种技术，用于在不干扰中央处理单元（CPU）的情况下，让外部设备（如硬盘驱动器、网络接口卡、图形卡等）直接访问系统内存。DMA的主要作用是提高数据传输的效率和性能，减轻CPU的负担。

DMA的工作原理是通过专门的DMA控制器或硬件来协调数据传输。当外部设备需要读取或写入数据到系统内存时，CPU可以将数据传输的任务交给DMA控制器，然后继续执行其他任务，而不必等待数据传输完成。DMA控制器负责在设备和内存之间传输数据，而无需CPU的干预。

DMA的主要作用包括：

提高性能：DMA允许外部设备直接访问内存，减少了CPU的干预，从而提高了数据传输速度和整体系统性能。

减轻CPU负担：CPU不必处理每个数据传输请求，而是将这些任务交给DMA控制器，从而释放了CPU的处理能力，使其能够更好地执行其他任务。

支持高带宽数据传输：DMA对于需要大量数据传输的任务非常有用，如高清视频流、大文件的读写等。

支持多任务处理：DMA可以同时处理多个数据传输请求，这对于多任务操作系统非常有用。

总之，DMA是一项重要的技术，可以显著提高计算机系统的性能和效率，特别是在需要大量数据传输的应用中，如多媒体处理和网络通信。它使CPU能够更有效地利用其处理能力，而不必长时间等待数据传输完成。

② 代码思路：

IDLE 接收空闲中断+DMA

功能：

STM32 IDLE 接收空闲中断

功能：

在使用串口接受字符串时，可以使用空闲中断（IDLEIE置1，即可使能空闲中断），这样在接收完一个字

符串，进入空闲状态时（IDLE置1）便会激发一个空闲中断。在中断处理函数，我们可以解析这个字符串。

接受完一帧数据，触发中断

STM32的IDLE的中断产生条件：

在串口无数据接收的情况下，不会产生，当清除IDLE标志位后，必须有接收到第一个数据后，才开始触发，一旦接收的数据断流，没有接收到数据，即产生IDLE中断

STM32 RXNE接收数据中断

功能：

当串口接收到一个bit的数据时，(读取到一个停止位) 便会触发 RXNE接收数据中断

接收到一个字节的的数据，触发中断

比如给上位机给单片机一次性发送了8个字节，就会产生8次RXNE中断，1次IDLE中断。

串口CR1寄存器

25.7.1 Control register 1 (USART_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	EOBIE	RTOIE	DEAT[4:0]					DED[4:0]				
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER 8	CMIE	MME	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

对bit4写1开启IDLE接受空闲中断

,对bit5写1开启RXNE接收数据中断。

串口ISR寄存器

25.7.8 Interrupt & status register (USART_ISR)

Address offset: 0x1C

Reset value: 0x0000 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	RE ACK	TE ACK	WUF	RWU	SBKF	CMF	BUSY
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

此寄存器为串口状态查询寄存器

当串口接收到数据时，bit5 RXNE就会自动变成1，当接收完一帧数据后，bit4就会变成1。

清除RXNE中断标志位的方法为：

只要把接收到的一个字节读出来，就会清除这个中断

在STM32F1 /STM32F4 系列中 清除IDLE中断标志位的方法为：

1. 先读SR寄存器，

2. 再读DR寄存器。

Bit 4 IDLE: IDLE line detected

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if the IDLEIE=1 in the USART_CR1 register. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No Idle Line is detected

1: Idle Line is detected

Note: The IDLE bit will not be set again until the RXNE bit has been set itself (i.e. a new idle line occurs).

<https://blog.csdn.net/CSDN@AspyRain>

memset()函数

```
extern void *memset(void *buffer, int c, int count)
```

- buffer: 为指针或是数组
- c: 是赋给buffer的值
- count: 是buffer的长度.

USART采用DMA接收时，如何读取当前接收字节数？

```
#define __HAL_DMA_GET_COUNTER(__HANDLE__) ((__HANDLE__)->Instance->CNDTR);
```

DMA接收时该宏将返回当前接收空间剩余字节

实际接受的字节= 预先定义接收总字节 - __HAL_DMA_GET_COUNTER()

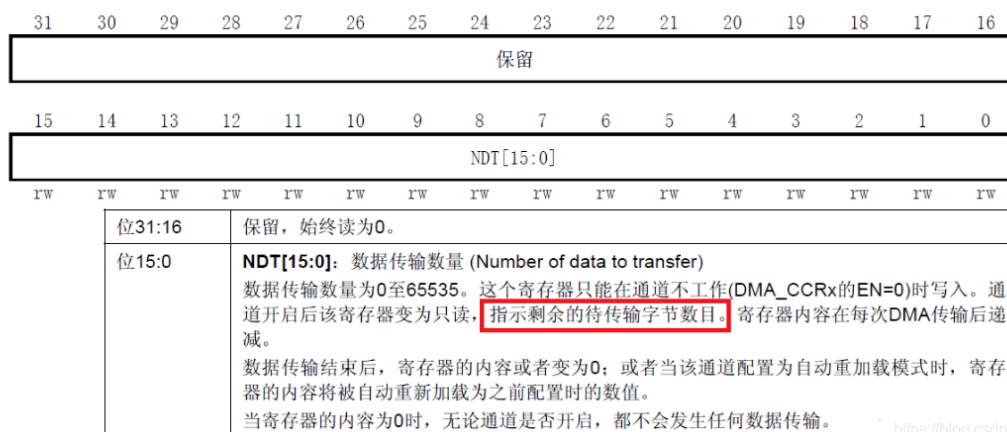
其本质就是读取NDTR寄存器，DMA通道结构体中定义了NDTR寄存器，读取该寄存器即可得到未传输的数据数呢，

NDTR寄存器

10.4.4 DMA通道x传输数量寄存器(DMA_CNDTRx)(x = 1...7)

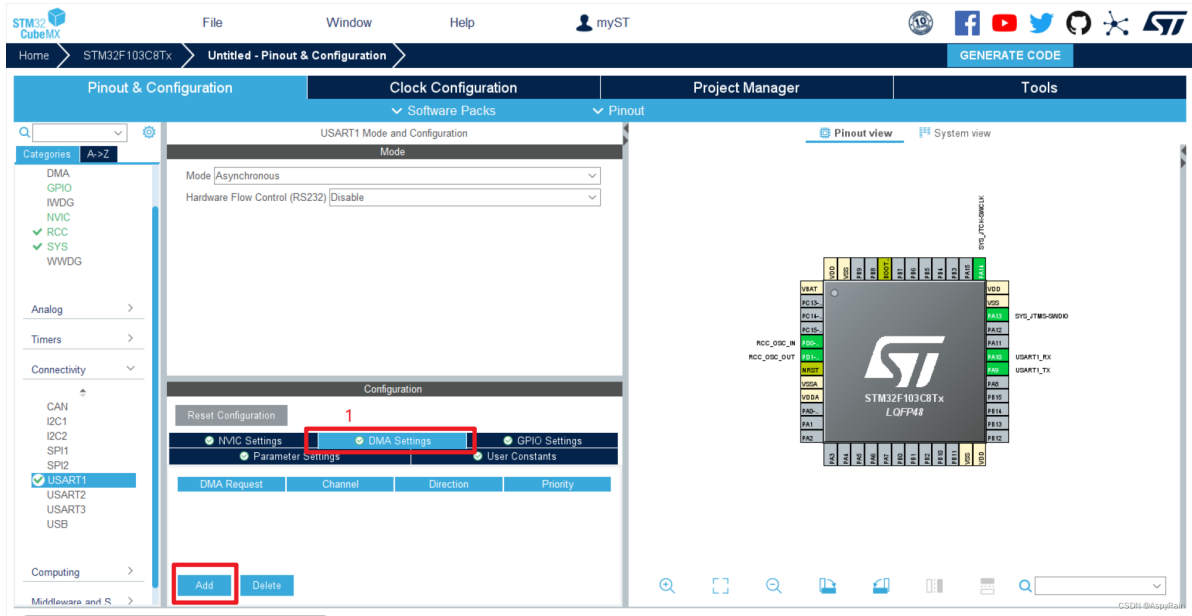
偏移地址: 0x0C + 20 x (通道编号 - 1)

复位值: 0x0000 0000



II 创建项目

在前一个项目创建的基础上，点击USART栏中的ADD,添加如下：



The screenshot shows the STM32CubeMX Pinout & Configuration window. The USART1 Mode and Configuration section is active, showing Mode: Asynchronous and Hardware Flow Control (RS232): Disable. The Configuration section shows a table with DMA Request, Channel, Direction, and Priority. The DMA Settings tab is selected, and the DMA Request is set to USART1_RX, Channel to DMA1 Channel 5, Direction to Peripheral To Memory, and Priority to Low. The USART1_TX is also configured with DMA1 Channel 4, Memory To Peripheral, and Low priority. The Add button is highlighted with a red box.

DMA Request	Channel	Direction	Priority
USART1_RX	DMA1 Channel 5	Peripheral To Memory	Low
USART1_TX	DMA1 Channel 4	Memory To Peripheral	Low

III 编程

在usart.c中添加：

```
volatile uint8_t rx_len = 0; //接收一帧数据的长度
volatile uint8_t recv_end_flag = 0; //一帧数据接收完成标志
uint8_t rx_buffer[100]={0}; //接收数据缓存数组

void MX_USART1_UART_Init(void)
{
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart1) != HAL_OK)
    {
        Error_Handler();
    }

    __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE); //使能IDLE中断
    __HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE); //使能IDLE中断

    //DMA接收函数，此句一定要加，不加接收不到第一次传进来的实数据，是空的，且此时接收到的数据长度为
    //缓存器的数据长度
    HAL_UART_Receive_DMA(&huart1, rx_buffer, BUFFER_SIZE);
}
```

```
}
```

在usart.h中添加

```
extern UART_HandleTypeDef huart1;
extern DMA_HandleTypeDef hdma_usart1_rx;
extern DMA_HandleTypeDef hdma_usart1_tx;
/* USER CODE BEGIN Private defines */

#define BUFFER_SIZE 100
extern volatile uint8_t rx_len ; //接收一帧数据的长度
extern volatile uint8_t recv_end_flag; //一帧数据接收完成标志
extern uint8_t rx_buffer[100]; //接收数据缓存数组
```

在main.c的/* USER CODE BEGIN 0 */添加:

```
uint8_t      Rx_Flag = 0;           //标志位
uint8_t      RecCount = 0;          //个数
uint8_t      Rx_Buf[256];           //接收缓冲区
uint8_t      Tx_Buf[256];
char message[]="Hello windows!\n";
char tips[]="CommandError\n";//提示1
char tips1[]="start\n";//提示2
char tips2[]="stop\n";//提示3
char flag='d';
char rx_buf[100];

void toggle_flag(char now_flag){
    flag=now_flag;
}
void transmit_DMA(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t
Size){
    HAL_UART_Transmit_DMA(huart, pData,Size);
    HAL_Delay(1000);
}
```

在while里面添加:

```
if (flag=='u'){
    transmit_DMA(&huart1,(uint8_t *)message,strlen(message));
}
/* USER CODE BEGIN 3 */
if(recv_end_flag == 1) //接收完成标志
{
    char read=flag;
    rx_len = 0;//清除计数
    recv_end_flag = 0;//清除接收结束标志位
    if (strcmp(tips1, rx_buffer, strlen(tips1)-2) == 0) {
        toggle_flag('u');
        transmit_DMA(&huart1, (uint8_t *)&tips1, strlen(tips1));
    } else if (strcmp(tips2, rx_buffer, strlen(tips2)-2) == 0) {
```

```

        toggle_flag('d');
        transmit_DMA(&huart1, (uint8_t *)&tips2, strlen(tips2));
    } else {
        toggle_flag(read);
        transmit_DMA(&huart1, (uint8_t *)&tips, strlen(tips));
    }

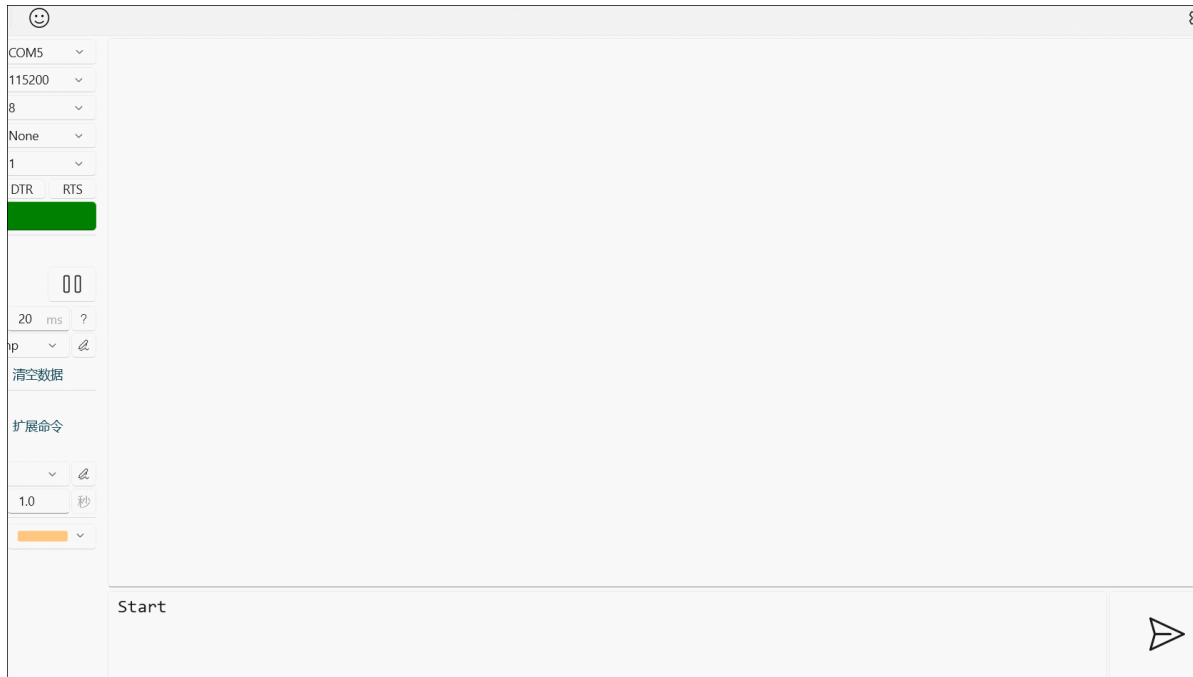
    memset(rx_buffer, 0, rx_len);
}

HAL_UART_Receive_DMA(&huart1, (uint8_t *)rx_buffer, BUFFER_SIZE); //重新打开

```

DMA接收

IV 编译烧录



本部分参考链接:<https://blog.csdn.net/as480133937/article/details/105013368>