STM32Cube高效开发教程（基础篇）

第12章　USART串口通信

王维波
中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（基础篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2021年9月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc.edu.cn

第12章　USART串口通信

12.1 USART接口功能概述

12.2 示例电路和MX项目设置

12.3 项目初始化代码分析

12.4 编写用户功能代码

12.1 USART接口功能概述

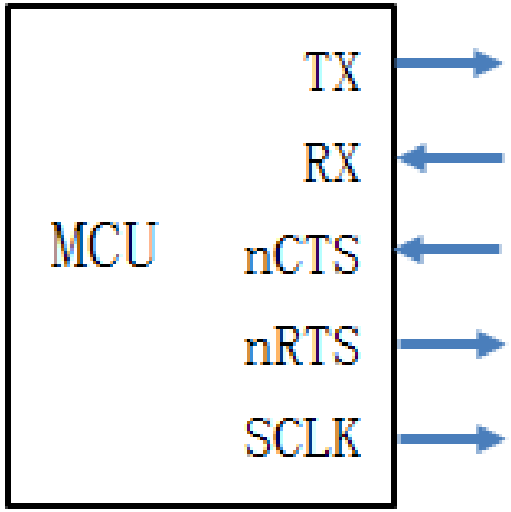
12.1.1 USART接口信号

12.1.2 USART接口通讯参数

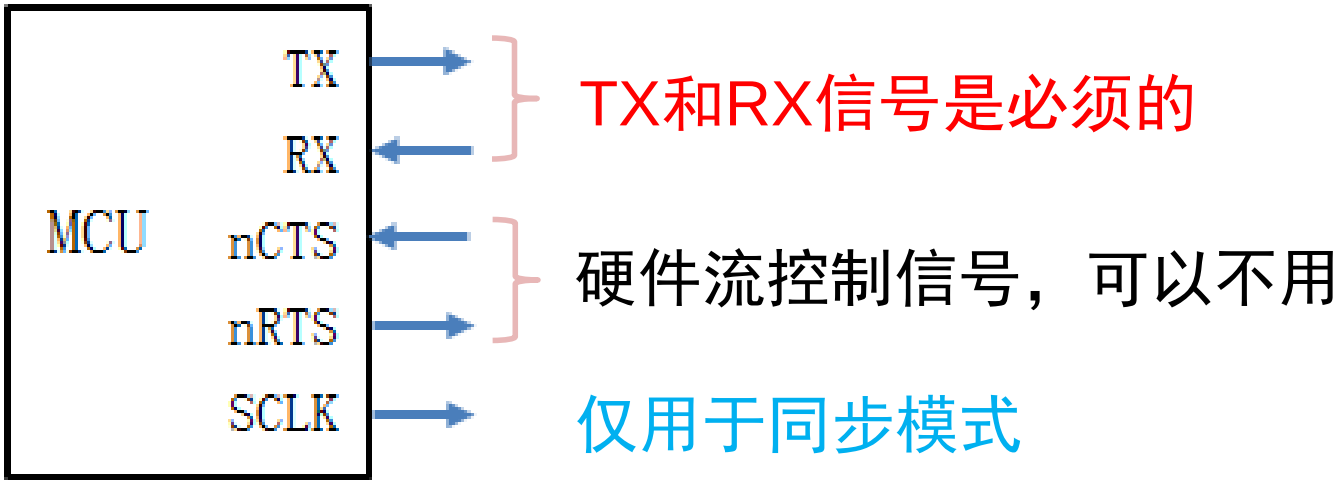
12.1.1 USART接口信号

USART：Universal synchronous asynchronous receiver transmitter，通用同步/异步收发器，一般简称串口

USART接口最多有5个信号线



- TX：串行数据输出
- RX：串行数据输入
- nCTS：清除以发送（clear to send）。当nCTS为低电平时，表示对方设备准备好了接收
- nRTS：请求以发送（request to send。当本机可以接收数据时，将nRTS置为低电平，通知对方设备可以发送数据了
- SCLK：发送器时钟输出信号，仅适用于同步模式

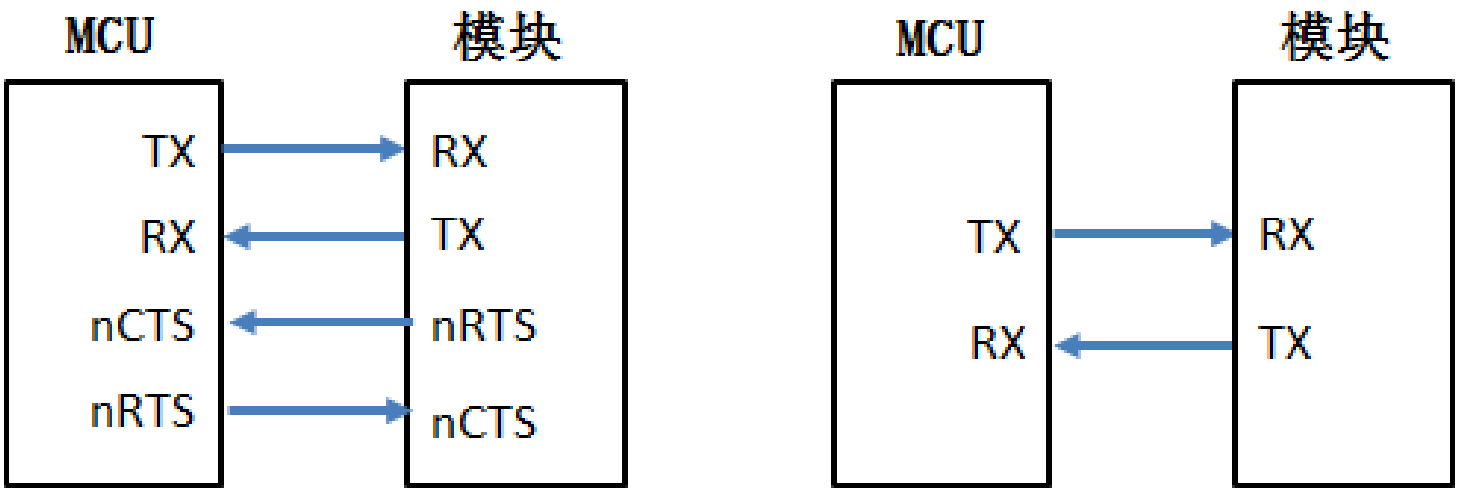


USART 有同步或异步模式工作，一般用异步模式，所以一般不用SCLK信号

还有一种UART（Universal Asynchronous Receiver Transmitter）接口，就是没有同步模式的串口。而且，UART接口一般也没有硬件流控制信号

1. 串口之间的连接

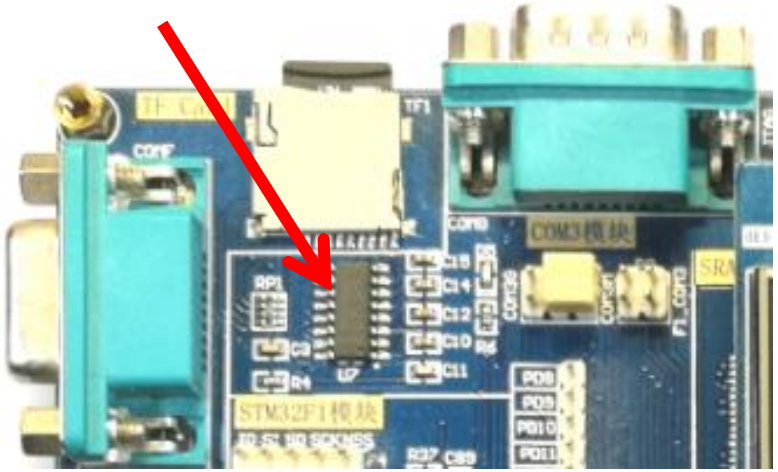
两个设备通过异步串口连接时，两个数据线要交叉连接，硬件流控制信号线也需要交叉连接，如图所示。

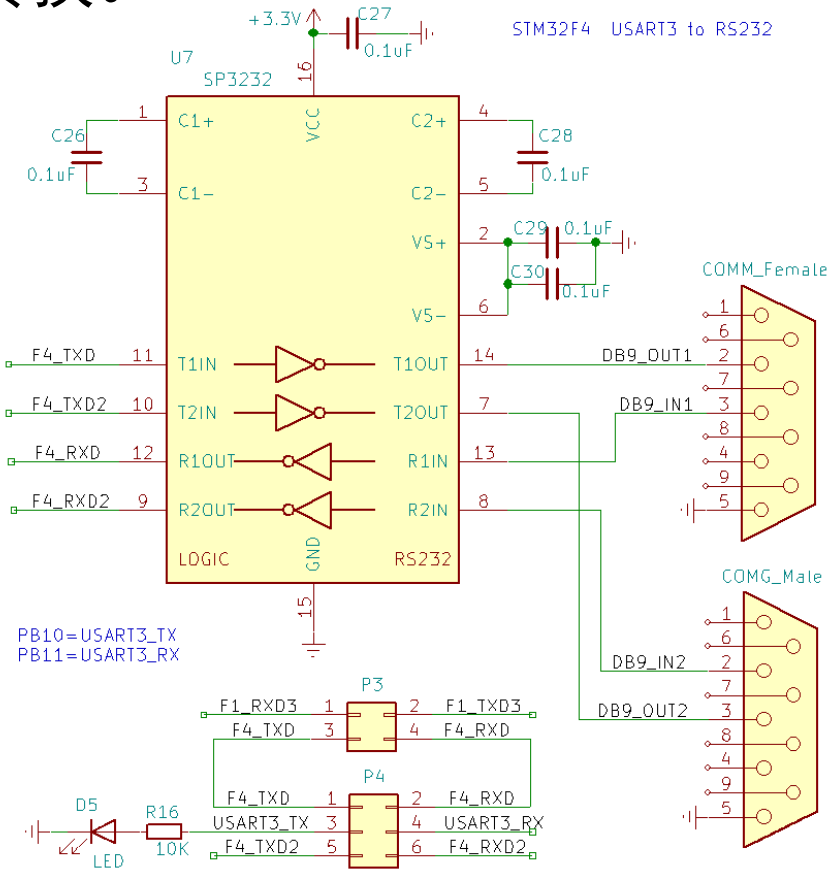


2. 串口与RS232之间的转换

MCU上的USART接口是逻辑电平（TTL或CMOS电平）。RS232接口与USART使用相同的串口通讯协议，但是电平不同，需要进行RS232与逻辑电平之间的转换。

开发板上用一个SP3232芯片实现USART3与RS232之间的转换。





3. USB转RS232

USB转RS232串口的转换线，Win7上需要安装驱动程序。

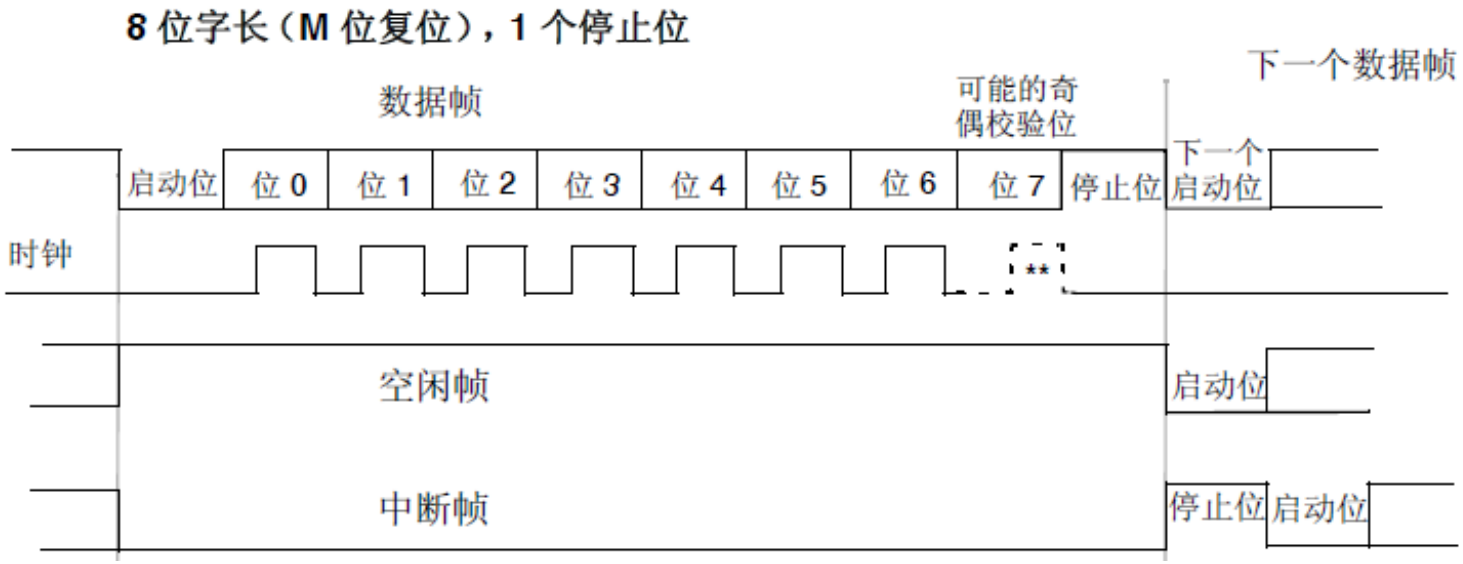
在电脑上作为一个虚拟COM口，可直接连接开发板上的RS232接口

4. USB转USART串口

有一些芯片可以实现USB到USART串口的转换，常用的此类芯片有CH340、PL2303、FT232等。有这种转换线。

- 红线=VCC
- 黑线=GND
- 白线=RXD
- 绿线=TXD

12.1.2 USART接口通讯参数



- 数据位：8位或9位，一般设置为8位。
- 奇偶校验位：奇校验，偶校验，或无校验
- 停止位：1个或2个停止位，一般设置为1个停止位。
- 波特率：就是串行数据传输的速率，单位bps（bits per second），如9600，19200，115200等。

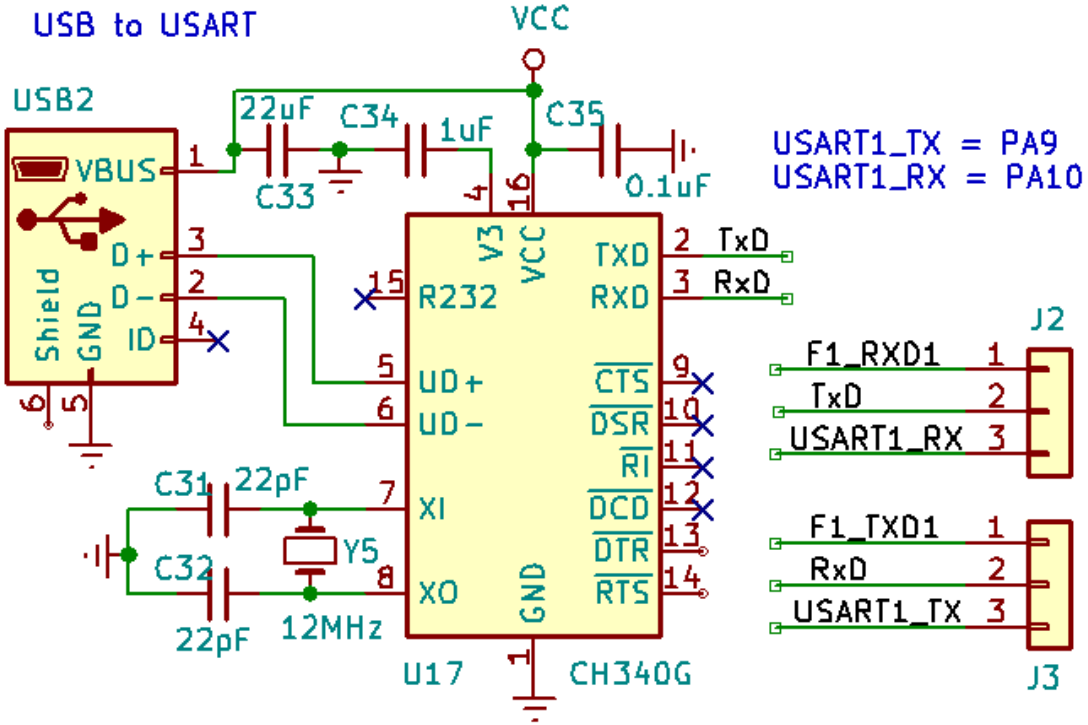
12.2 示例电路和STM32CubeMX项目设置

12.2.1 硬件电路

12.2.2 STM32CubeMX项目设置

12.2.1 硬件电路

开发板上有个CH340芯片，实现USB到USART的转换，直接使用一个MicroUSB线就可以连接电脑和开发板，进行串口通讯，还可以给开发板供电。



需要设置跳线J2和J3，使CH340与F4处理器的USART1连接

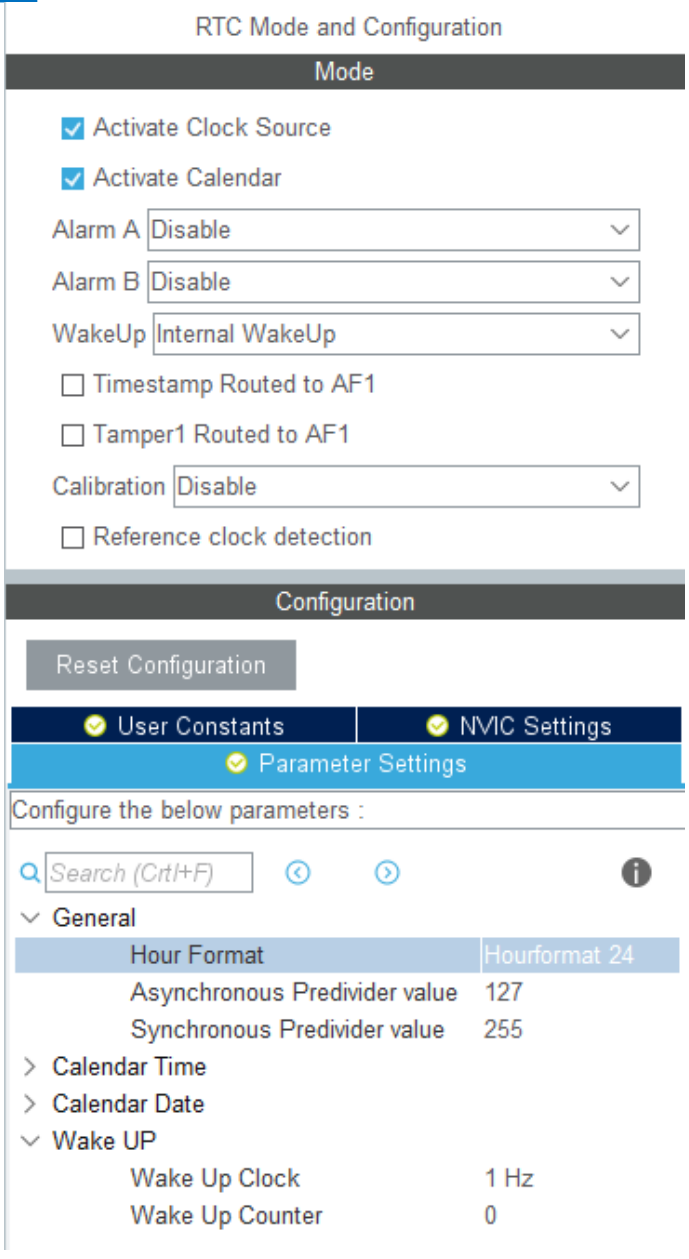
12.2.2 STM32CubeMX项目设置

示例功能：

- 在RTC周期唤醒中断里读取RTC当前时间，通过USART1上传到电脑
- 通过PC上的串口监视软件向开发板发送指令，MCU接收到指令后进行解析，相应地修改RTC的时间

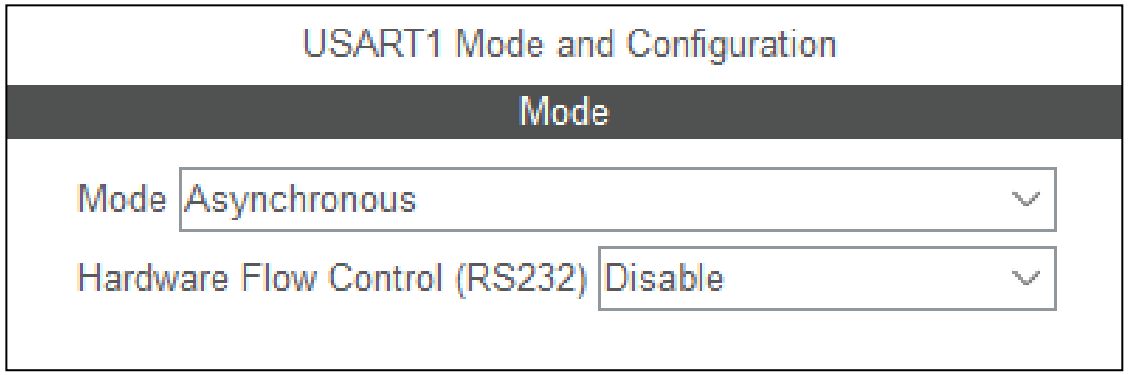
（1）RTC的设置

启用周期唤醒功能，周期为1Hz，1秒钟唤醒一次，开启周期唤醒中断



（2）USART1的设置

USART1接口的模式设置



（1）Mode：工作模式，设置为Asynchronous（异步）

（2）Hardware Flow Control(RS232)：硬件流控制。开发板的USART接口并没有使用硬件流信号，所以设置为Disable。

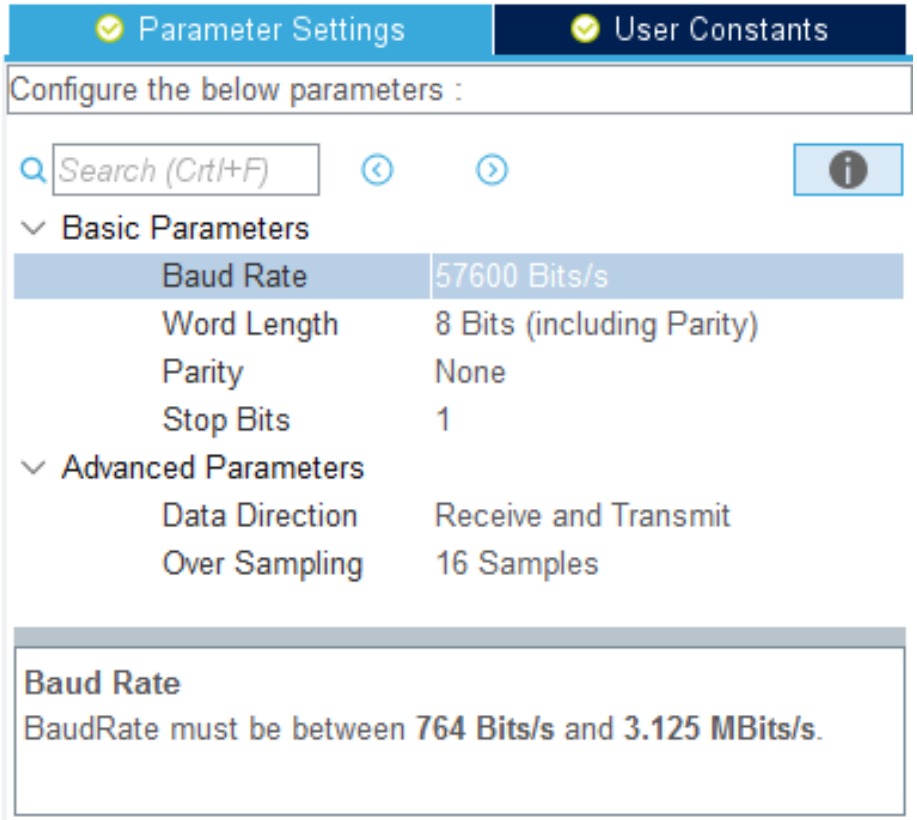
USART1接口的参数设置

（1）Baud Rate：波特率，设置为57600 bps

（2）Word Length：字长（包括奇偶校验位），可选8位或9位，设置为8位

（3）Parity：奇偶校验位。可选None（无）、Even（偶校验）、Odd（奇校验）。这里设置为None。如果设置有奇偶校验，字长应该设置为9

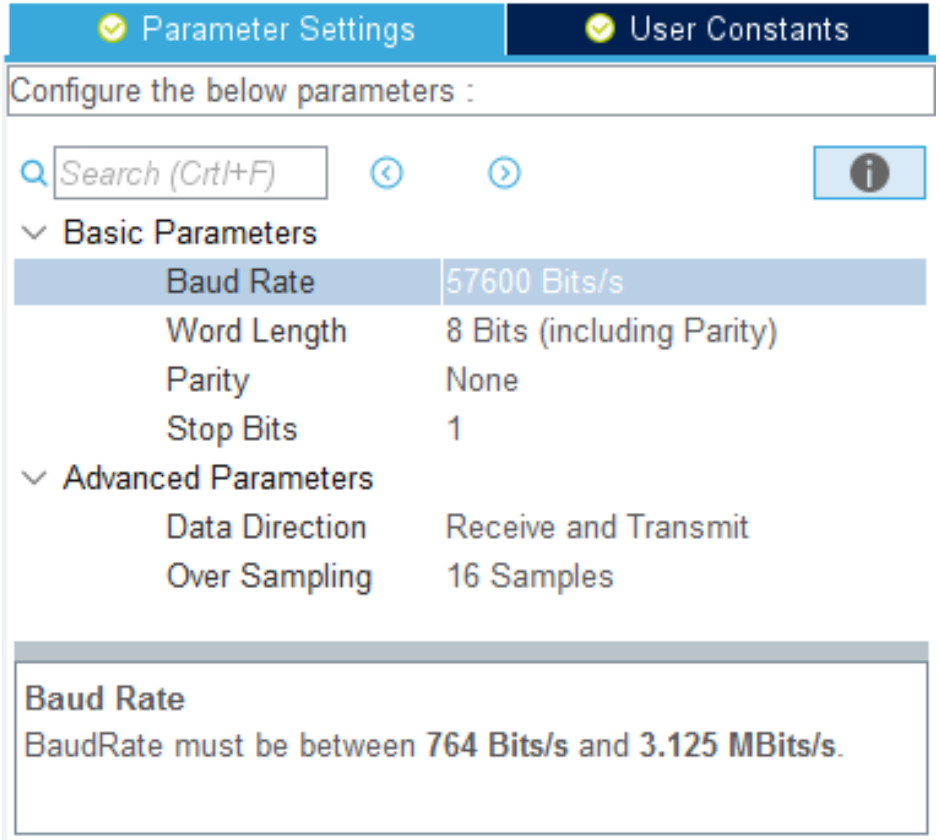
（4）Stop Bits：停止位个数。可选1或2，这里设置为1。



STM32处理器扩展的2个参数

（1）Data Direction：数据方向，这里设置为Receive and Transmit，也就是收发双向

（2）Over Sampling：过采样，可选16或8采样点，这里设置为16



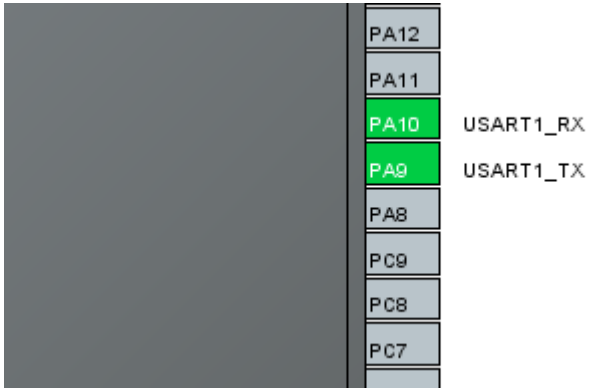
这样设置USART1接口后，CubeMX会自动配置PA9和PA10作为USART1_TX和USART1_RX信号复用引脚，这与电路上是一致的，无需再做任何GPIO设置。

FSMC | RCC | SYS | USART1

Search Signals

Search (Crtl+F)

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/...	Maximum out...
PA9	USART1_TX	n/a	Alternate Function Push Pull	Pull-up	Very High
PA10	USART1_RX	n/a	Alternate Function Push Pull	Pull-up	Very High



打开USART1的全局中断

由于RTC唤醒中断和串口中断的程序中都可能用到延时函数HAL_Delay()，所以设置这两个中断的抢占优先级为1，即要低于System tick timer的抢占优先级。

NVIC Mode and Configuration

Configuration

NVIC | Code generation

Priority Group: 2 bits for pre-emption priority 2 bits for subpriority

☐ Sort by Premption Priority and Sub Priority

Search: Search (Crtl+F)

☑ Show only enabled interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
RTC wake-up interrupt through EXTI line 22	☑	1	0
USART1 global interrupt	☑	1	0

12.3 项目初始化代码分析

12.3.1 主程序

12.3.2 USART1接口初始化

12.3.3 USART接口的中断

12.3.4 USART接口HAL驱动常用函数

12.3.1 主程序

函数MX_USART1_UART_Init()用于USART1接口的初始化。

```
int main(void)
{
    HAL_Init();                    //HAL初始化
    SystemClock_Config();          //系统时钟配置
    /* 初始化所有已配置外设*/
    MX_GPIO_Init();                //GPIO初始化
    MX_FSMC_Init();                //TFT LCD接口初始化
    MX_RTC_Init();                 //RTC初始化
    MX_USART1_UART_Init();         //USART1接口初始化
    while (1)
    {

    }
}
```

12.3.2 USART1接口初始化

函数MX_USART1_UART_Init()用于USART1接口的初始化

```
/* 文件：usart.c    ----------------------------------------*/
#include "usart.h"
UART_HandleTypeDef huart1;              //表示USART1的句柄变量

void MX_USART1_UART_Init(void)  // USART1 初始化函数
{
      huart1.Instance = USART1;                          //USART1外设地址
      huart1.Init.BaudRate = 57600;                      //波特率
      huart1.Init.WordLength = UART_WORDLENGTH_8B;     //字长1位
      huart1.Init.StopBits = UART_STOPBITS_1;            //1个停止位
      huart1.Init.Parity = UART_PARITY_NONE;             //无奇偶校验
      huart1.Init.Mode = UART_MODE_TX_RX;                //TX-RX模式
      huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;    //无硬件流控制
      huart1.Init.OverSampling = UART_OVERSAMPLING_16;  //过采样
      if (HAL_UART_Init(&huart1) != HAL_OK)
      {
        Error_Handler();
      }
}
```

HAL_UART_MspInit()在HAL_UART_Init()函数内被调用，完成USART1的GPIO复用引脚配置和中断配置

```
void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    if(uartHandle->Instance==USART1)
    {
      __HAL_RCC_USART1_CLK_ENABLE();        // USART1 时钟使能
      __HAL_RCC_GPIOA_CLK_ENABLE();
      /**USART1 GPIO引脚配置
      PA9  作为 USART1_TX，  PA10  作为 USART1_RX      */
      GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10;
      GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
      GPIO_InitStruct.Pull = GPIO_PULLUP;
      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
      GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
      HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

      /* USART1 中断初始化 */
      HAL_NVIC_SetPriority(USART1_IRQn, 1, 0);
      HAL_NVIC_EnableIRQ(USART1_IRQn);
    }
}
```

USART接口初始化中用到两个结构体， 一个用于定义串口对象

```
UART_HandleTypeDef  huart1;      //表示USART1的变量
```

结构体UART_HandleTypeDef的部分定义如下

```
typedef struct __UART_HandleTypeDef
{
  USART_TypeDef     *Instance;     //UART寄存器基址
  UART_InitTypeDef  Init;          //UART通讯参数
........//其他成员定义见讲义
} UART_HandleTypeDef;
```

其中，Instance用于指定一个串口的基址，如程序中有如下的语句，使得变了huart1表示USART1。

```
huart1.Instance = USART1;          //USART1外设地址
```

成员变量Init是结构体类型UART_InitTypeDef，用于设置串口通讯参数。结构体类型UART_InitTypeDef的定义如下

```
typedef struct
{
    uint32_t BaudRate;       //波特率
    uint32_t WordLength;     //字长
    uint32_t StopBits;       //停止位个数
    uint32_t Parity;         //是否有奇偶校验
    uint32_t Mode;           //工作模式
    uint32_t HwFlowCtl;      //硬件流控制
    uint32_t OverSampling;  //过采样
} UART_InitTypeDef;
```

函数MX_USART1_UART_Init()中设置串口通讯参数的代码与CubeMX中的设置是对应的。

12.3.3 USART接口的中断

USART1接口的全局中断ISR函数是USART1_IRQHandler()，在文件stm32f4xx_it.c中生成了其代码框架，同时还有RTC中断的ISR函数框架。

```
/* 文件： stm32f4xx_it.c-----------------------------------*/

void RTC_WKUP_IRQHandler(void)      //RTC WKUP中断 ISR函数
{
    HAL_RTCEx_WakeUpTimerIRQHandler(&hrtc);
}

void USART1_IRQHandler(void)        //USART1中断ISR函数
{
    HAL_UART_IRQHandler(&huart1);  //USART中断通用处理函数
}
```

一个USART接口只有1个ISR函数，ISR函数中调用通用处理函数HAL_UART_IRQHandler()。但是一个USART有多个中断事件，有些中断事件有回调函数。

中断事件宏定义	中断事件描述	回调函数
UART_IT_CTS	CTS 信号变化中断	无
UART_IT_LBD	LIN 打断检测中断	无
UART_IT_TXE	发送数据寄存器非空中断	无
UART_IT_TC	传输完成中断，用于发送完成	HAL_UART_TxCpltCallback(huart)
UART_IT_RXNE	接收数据寄存器非空中断	HAL_UART_RxCpltCallback(huart)
UART_IT_IDLE	线路空闲状态中断	无
UART_IT_PE	奇偶校验中断	HAL_UART_ErrorCallback(huart)
UART_IT_ERR	发生帧错误、噪声错误、溢出错误的中断	HAL_UART_ErrorCallback(huart)

12.3.4 USART接口HAL驱动常用函数

假设已经定义了一个串口外设对象变量huart1，即：

```
UART_HandleTypeDef  huart1;
```

1. 常用宏函数

- __HAL_UART_ENABLE(__HANDLE__)，使能一个UART口，如

```
__HAL_UART_ENABLE(&huart1);
```

- __HAL_UART_ENABLE_IT(__HANDLE__, __INTERRUPT__)，使能某个中断事件源，如

```
__HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);   //使能接收中断
__HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE);    //使能空闲中断
```

- __HAL_UART_DISABLE_IT(__HANDLE__, __INTERRUPT__)，禁止某个中断事件源

- __HAL_UART_GET_FLAG(__HANDLE__, __FLAG__)，检查一个串口的某个中断标志是否被置位，返回值为宏定义常量SET或RESET。中断事件标志的宏定义如下：

UART_FLAG_CTS:	CTS 信号变化标志
UART_FLAG_LBD:	LIN 打断检测标志
UART_FLAG_TXE:	发送数据寄存器空标志
UART_FLAG_TC:	发送完成标志
UART_FLAG_RXNE:	接收数据寄存器非空标志
UART_FLAG_IDLE:	线路空闲标志
UART_FLAG_ORE:	出错误标志
UART_FLAG_NE:	声错误标志
UART_FLAG_FE:	帧错误标志
UART_FLAG_PE:	奇偶校验错误标志

- __HAL_UART_CLEAR_FLAG(__HANDLE__, __FLAG__)，清除一个UART口的某个事件标志位

2. 数据传输模式

串口数据传输有两种模式：

- 阻塞模式（Blocking mode）就是轮询模式，例如使用函数HAL_UART_Transmit()发送一个缓冲区的数据时，这个函数会一直执行直到数据传输完成之后函数才返回。

- 非阻塞模式（Non-blocking mode）是使用中断或DMA模式进行数据传输，例如使用函数HAL_UART_Transmit_IT()启动一个缓冲区的数据传输后，该函数立刻返回。数据传输的过程引发各种中断，用户在相应的回调函数里去处理。

3. 阻塞式数据传输函数

函数HAL_UART_Transmit()和HAL_UART_Receive()用于阻塞模式数据发送和接收。

- HAL_StatusTypeDef **HAL_UART_Transmit**(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)

以阻塞模式发送一个缓冲区的数据，若返回值为HAL_OK表示传输成功。参数pData是缓冲区指针；参数Size是需要传输的字节数；参数Timeout是超时限制节拍数，表示超过这个时间时函数无条件返回。例如：

```
uint8_t  timeStr[]="15:32:06\n";
HAL_UART_Transmit(&huart1,timeStr,sizeof(timeStr),200);
```

- HAL_StatusTypeDef **HAL_UART_Receive**(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)

以阻塞模式接收指定长度的数据道缓冲区，若返回值为HAL_OK表示接收成功。参数pData是用于存放接收数据的缓冲区的指针；参数Size是需要接收的字节数；参数Timeout是超时限制节拍数。例如：

```
uint8_t  recvStr[10];
HAL_UART_Receive(&huart1, recvStr,10 ,200);
```

4. 非阻塞式数据传输函数

用中断或DMA方式进行非阻塞模式的数据传输

- HAL_StatusTypeDef **HAL_UART_Transmit_IT**(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)

以中断方式发送一定长度的数据，若返回值为HAL_OK表示启动成功，但并不表示数据发送完成了。参数pData是需要发送数据缓冲区的指针，参数Size是需要发送的字节数。例如：

```
uint8_t  timeStr[]="15:32:06\n";
HAL_UART_Transmit_IT(&huart1,timeStr,sizeof(timeStr));
```

发送结束后会调用回调函数HAL_UART_TxCpltCallback()

- HAL_StatusTypeDef **HAL_UART_Receive_IT**(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)

以中断方式接收一定长度的数据，若返回值为HAL_OK表示启动成功，但并不表示已经接收完数据了。参数pData是存放接收数据的缓冲区的指针，参数Size是需要接收的字节数。例如：

```
uint8_t  rxBuffer[10];          //接收数据缓冲区
HAL_UART_Receive_IT(huart, rxBuffer,10);
```

接收完成后会调用回调函数HAL_UART_RxCpltCallback()

函数HAL_UART_Receive_IT()有一些特性需要注意：

（1）这个函数执行一次只能接收固定长度的数据，即使设置为接收一个字节的数据。

（2）在完成数据接收后会自动关闭接收中断，不会再继续接收下一批数据。若要再接收下一批数据，需要再执行一次这个函数，且不能在回调函数HAL_UART_RxCpltCallback()里调用这个函数。

因为HAL_UART_Receive_IT()的这些特性，使其在处理不确定长度、不确定输入时间的串口数据输入时比较麻烦，需要特殊的处理。

12.4 编写用户功能代码

12.4.1 示例功能

12.4.2 主程序

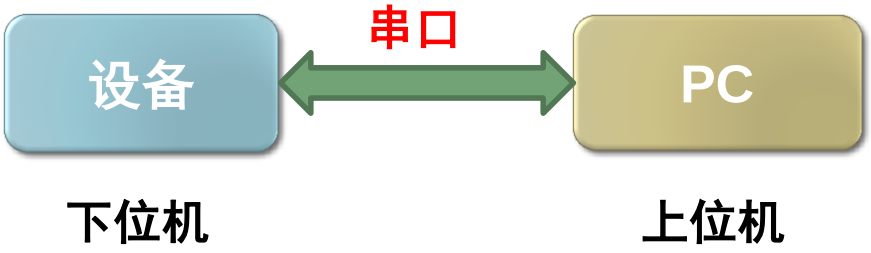
12.4.3 RTC周期唤醒中断的处理

12.4.4 串口中断的处理

12.4.1 示例功能

示例Demo12_1实现如下的一些功能：

- 在RTC周期唤醒中断里读取当前时间，在LCD上显示当前时间，并将时间转换为字符串之后通过串口发送给电脑
- 在电脑上使用串口监视软件查看接收到的数据，并且可以向开发板发送指令数据
- MCU持续以中断方式进行串口数据接收，接收到一条指令后就解析并执行指令的任务，例如修改当前时间

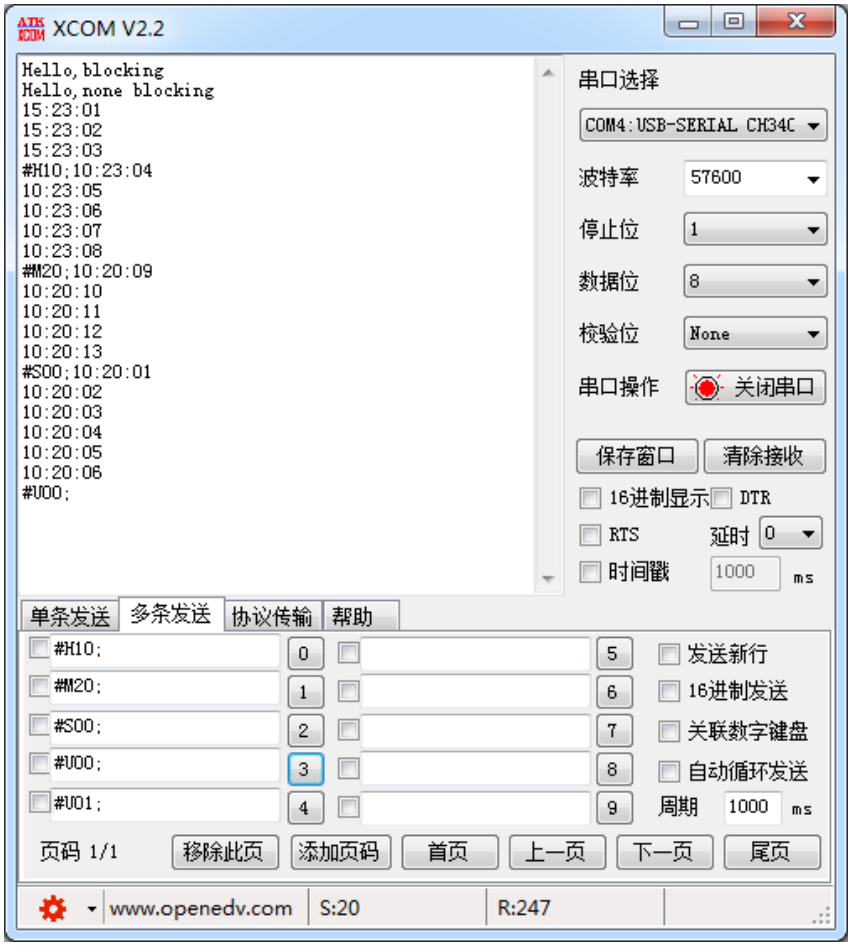


串口的硬件层实现了数据的收发，发送的数据具体是什么意义，需要规定上位机和下位机之间的通讯协议。

串口通讯协议就是传输数据的格式规范及其意义，如本示例中上位机向开发板发送的串口数据的格式定义见表12-2。

上位机发送的指令数据	指令功能
#H13;	设置小时，将RTC时间的小时修改为13
#M32;	设置分钟，将RTC时间的分钟修改为32
#S05;	设置秒，将RTC时间的秒修改为5
#U01; 或 #U00;	上传时间数据，或不上传时间数据

- 上位机发送的指令数据固定为5个字节，每个指令以‘#’开始，以‘;’结束
- 紧跟‘#’后面的一个字母表示指令类型，例如‘H’表示修改小时，‘M’表示修改分钟
- 类型字符后面是两位数字，表示指令的参数，例如“#H13;”表示要将RTC的当前时间的小时数修改为13



12.4.2 主程序

打开源程序文件main.c讲解

```
int main(void)
{
  / *  。 。 。 。 。 。 。 。  初始化部分  */
   /* USER CODE BEGIN 2 */
  TFTLCD_Init();  //LCD软件初始化
  LCD_ShowString(10,10, (uint8_t *)"Demo12_1:USART1+RTC");

   uint8_t hello1[]="Hello,blocking\n";
  HAL_UART_Transmit(&huart1,hello1,sizeof(hello1),500);  //in blocking mode
  HAL_Delay(100);

  uint8_t  hello2[]="Hello,none blocking\n";
  HAL_UART_Transmit_IT(&huart1,hello2,sizeof(hello2));  //in none blocking mode
  HAL_UART_Receive_IT(&huart1, rxBuffer,RX_CMD_LEN);  //中断方式接收5个字节
/* USER CODE END 2 */
  while (1)  {   }
}
```

12.4.3 RTC周期唤醒中断的处理

功能是读取RTC当前时间，转换为字符串timeStr，通过串口向上位机上传。

打开源程序文件rtc.c讲解

```
void HAL_RTCEx_WakeUpTimerEventCallback(RTC_HandleTypeDef *hrtc)
{
    RTC_TimeTypeDef sTime;
    RTC_DateTypeDef sDate;
    if (HAL_RTC_GetTime(hrtc, &sTime,  RTC_FORMAT_BIN) == HAL_OK)
    {
      HAL_RTC_GetDate(hrtc, &sDate,  RTC_FORMAT_BIN); //必须读取日期
      uint8_t  timeStr[]="15:32:06\n";  //时间字符串
      /*  。。。。。。。。。。。。。。  */
      ..........  //计算分钟、秒的代码略
      LCD_ShowString( 30, 50, timeStr);
      if (isUploadTime)   //变量isUploadTime在文件usart.c中定义
        HAL_UART_Transmit(&huart1,timeStr,sizeof(timeStr),200);
    }
}
```

12.4.4串口中断的处理

对USART1中断ISR函数USART1_IRQHandler()稍微做了修改，代码如下：

```
void USART1_IRQHandler(void)
{
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQn 1 */
    on_UART_IDLE(&huart1);          //检测空闲中断并处理
    /* USER CODE END USART1_IRQn 1 */
}
```

执行了on_UART_IDLE(&huart1)，用于检测空闲中断并做相应处理，函数on_UART_IDLE()在usart.h文件中定义。

增加用户代码后的文件usart.h的内容如下：

打开源程序讲解

```
/* 文件： usart.h  ----------------------------------------*/
#include "main.h“

/* USER CODE BEGIN Includes */
#define  RX_CMD_LEN  5     //指令长度，5字节
extern  uint8_t  rxBuffer[];      //5字节的输入缓冲区，如“#H15;”,
extern  uint8_t  isUploadTime;  //是否上传时间数据
/* USER CODE END Includes */

extern UART_HandleTypeDef huart1;
void MX_USART1_UART_Init(void);

/* USER CODE BEGIN Prototypes */
void  HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);  //接收完成回调函数
void  on_UART_IDLE(UART_HandleTypeDef *huart);          //IDLE中断检测与处理
void  updateRTCTime();        //对接收指令的处理
/* USER CODE END Prototypes */
```

增加用户代码后的文件usart.c的内容如下

打开源程序讲解

```
/* 文件：usart.c   ----------------------------------------*/
#include "usart.h"
/* USER CODE BEGIN 0 */
#include      "rtc.h"
uint8_t  proBuffer[10]="#S45;";   //用于处理的数据
uint8_t  rxBuffer[10]="#H12;";    //接收数据缓冲区
uint8_t  rxCompleted=RESET;   //HAL_UART_Receive_IT()接收是否完成
uint8_t  isUploadTime=1;          //控制RTC周期唤醒中断里是否上传时间数据
/* USER CODE END 0 */
UART_HandleTypeDef huart1;

/* USER CODE BEGIN 1 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  //串口接收完成回调函数
{
        if (huart->Instance == USART1)
        {
            rxCompleted=SET;  //接收完成
            for(uint16_t i=0; i<RX_CMD_LEN; i++)
                proBuffer[i]=rxBuffer[i];   //复制接收到的指令字符串
            __HAL_UART_ENABLE_IT(huart, UART_IT_IDLE);  //开启IDLE中断
        }
}
```

1. 回调函数HAL_UART_RxCpltCallback()的功能

- 将接收缓冲区rxBuffer[ ]的数据复制到proBuffer[ ]
- 这个回调函数里最后使能UART_IT_IDLE中断

注意，HAL_UART_Receive_IT()完成一次数据接收后就关闭了串口接收中断，不会自动进行下一次的接收，需要再次调用HAL_UART_Receive_IT()以启动下一次的接收，但不能在回调函数HAL_UART_RxCpltCallback()里调用HAL_UART_Receive_IT()。

为了能连续进行中断方式的串口接收，程序的处理方法是：在完成一次接收，并且串口状态为空闲，也就是发生UART_IT_IDLE中断时，对接收到的指令数据进行处理，然后再次调用HAL_UART_Receive_IT()以启动下一次的接收。

2. 函数on_UART_IDLE()的功能

```
void  on_UART_IDLE(UART_HandleTypeDef *huart)
{
        if(__HAL_UART_GET_IT_SOURCE(huart,UART_IT_IDLE) == RESET)
                return;

        __HAL_UART_CLEAR_IDLEFLAG(huart);   //清除IDLE标志
        __HAL_UART_DISABLE_IT(huart, UART_IT_IDLE);  //禁止IDLE中断
        if (rxCompleted)  //接收到了一条指令
        {
                HAL_UART_Transmit(huart,proBuffer,RX_CMD_LEN,200);  //指令字符串传回PC
                updateRTCTime();
                rxCompleted=RESET;
                HAL_UART_Receive_IT(huart, rxBuffer,RX_CMD_LEN);  //再次启动接收
        }
}
```

如果rxCompleted被置位，就表示上次接收一个缓冲区的数据已经完成，调用updateRTCTime()函数对接收到的指令数据进行解析处理，并再次调用HAL_UART_Receive_IT()开启下一次串口中断方式接收。

3. 函数updateRTCTime()的功能

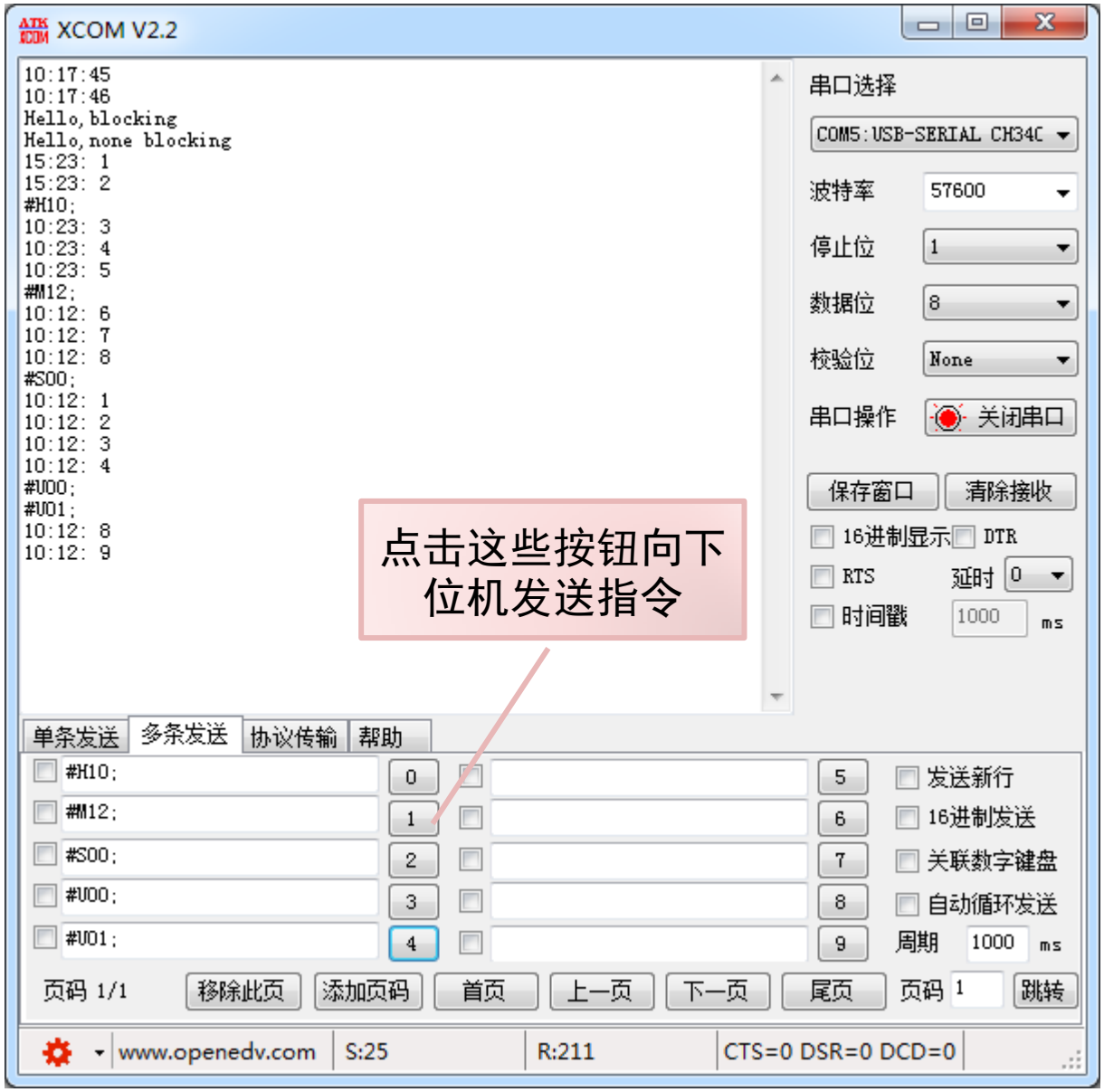
对接收到的指令进行解析和执行。例如，接收到的指令字符串是“#H10”，就将RTC时间的小时修改为10

打开源程序讲解

```
void updateRTCTime()            //根据串口接收的数据进行处理
{
        if (proBuffer[0] != '#')        //非有效指令
                return;
        uint8_t  timeSection=proBuffer[1];   //类型字符
        uint8_t  tmp10=proBuffer[2]-0x30;   //十位数
        uint8_t  tmp1 =proBuffer[3]-0x30;    //个位数
        uint8_t  val=10*tmp10+tmp1;
        if (timeSection=='U')        //是否上传RTC时间
        {
                isUploadTime=val;
                return;
        }
  。。。。。。。。。。。//中间代码略，见源程序
                HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN);  //设置RTC时间
}
```

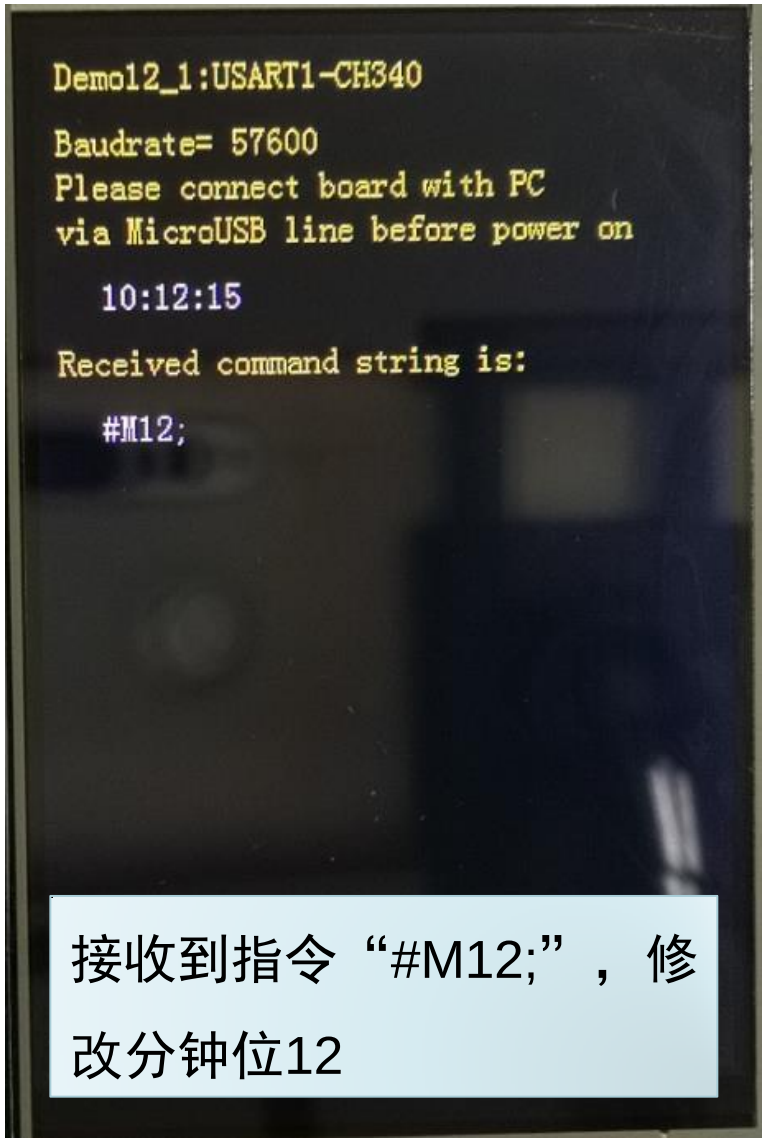
示例运行效果

串口监视软件运行画面，接收并显示下位机上传的数据，并可以向下位机发送指令



示例运行效果

接收到指令“#H10;”，修改小时为10



接收到指令“#M12;”，修改分钟位12

12.4.5 测试和讨论

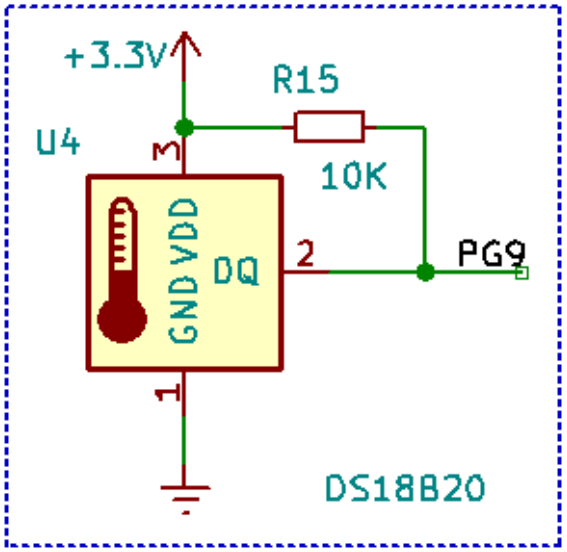
- 在使用上位机软件XCOM向开发板发送数据时需要注意：不要在指令后面添加额外的数据，也就是不要勾选“发送新行”复选框。
- 函数HAL_UART_Receive_IT()存在的一个问题就是每次只能接收固定长度的数据，且不能自动重复接收。

使用DMA传输模式可以解决串口自动重复接收的问题，在下一章介绍

综合性作业

开发板上的【2-5】位置可以插入一个DS18B20温度传感器进行温度测量，DS18B20的电路如下图所示，数据线接MCU的PG9引脚。

- 查阅DS18B20的数据手册，搞清楚其工作原理，编写或改写DS18B20驱动程序。
- 使用RTC周期唤醒，每2秒采集一次温度，在LCD上显示，并通过串口上传



【提示】用一个定时器实现微秒级延时函数