STM32Cube高效开发教程——基础篇

第13章　DMA

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

13.1 DMA功能概述

13.1.1 DMA简介

13.1.2 DMA传输特性

13.2 DMA的HAL驱动程序

13.3 USART的DMA传输示例

13.1.1 DMA简介

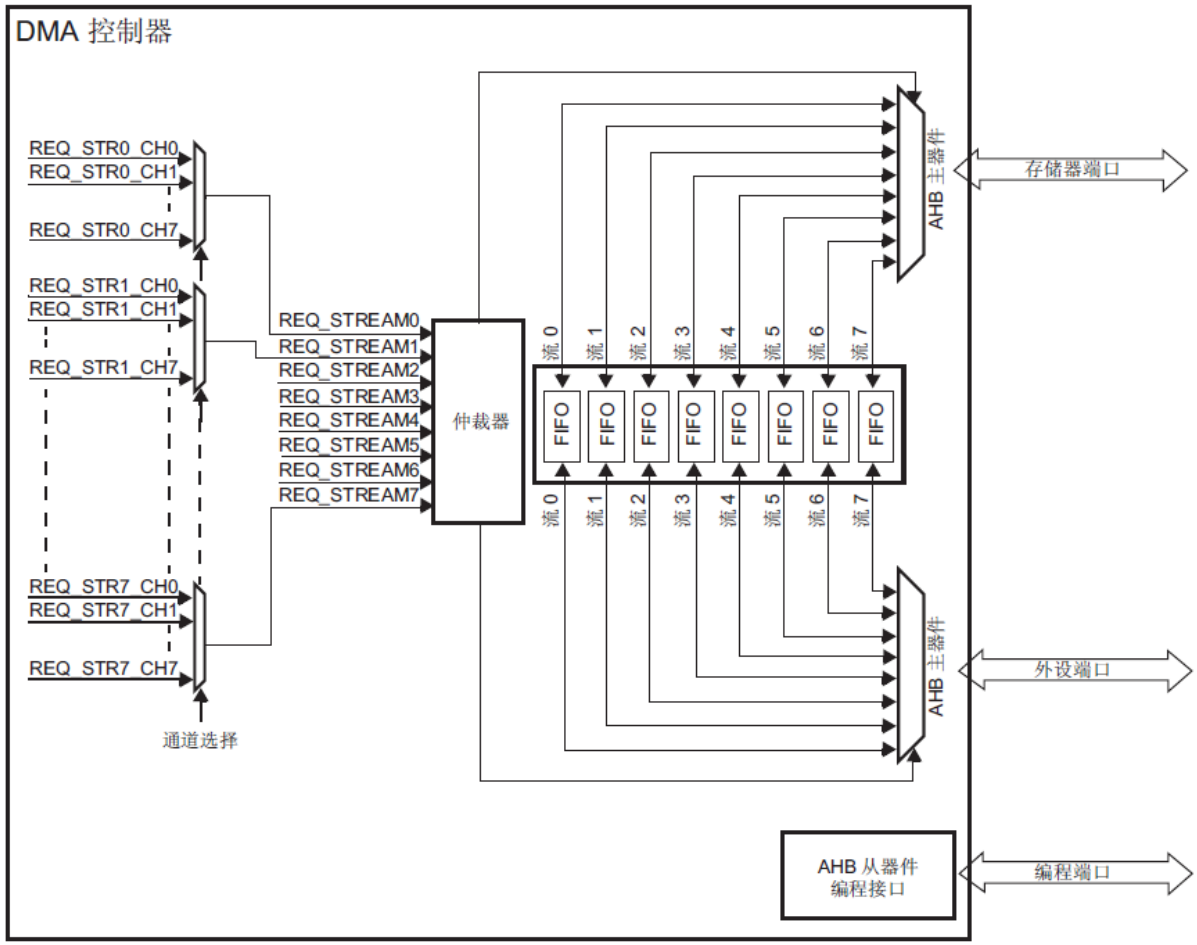
DMA（Direct Memory Access，直接存储器访问）是实现存储器与外设、存储器与存储器之间高效数据传输的方法

（1）DMA控制器

（2）DMA流

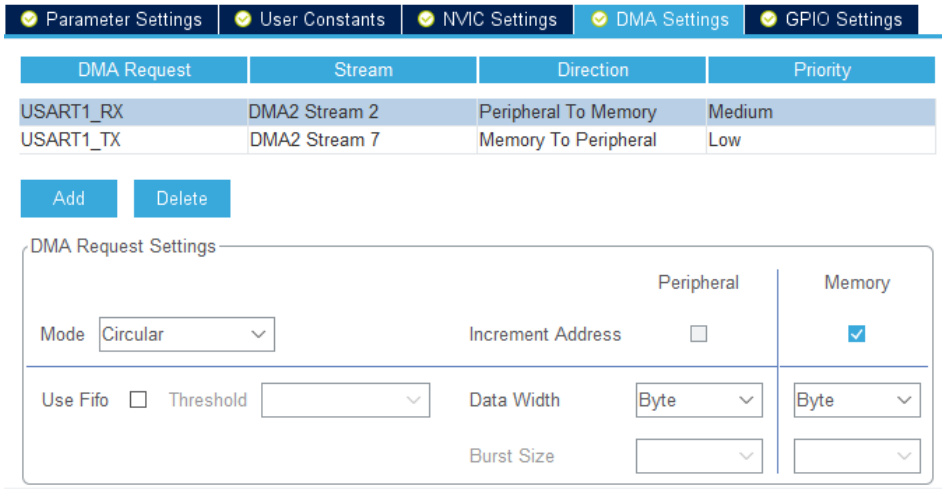
（3）DMA请求

（4）仲裁器



13.1.2 DMA传输属性

一个DMA流配置一个DMA请求后，就构成一个单方向的DMA数据传输链路，DMA传输属性就由DMA流的参数配置决定



- DMA流和通道
- DMA流的优先级别
- 源和目标的数据宽度
- 传输数据量大小
- 源和目标地址指针是否自增加
- DMA工作模式：Normal或Circular
- DMA传输方向
- 是否使用FIFO

DMA传输模式

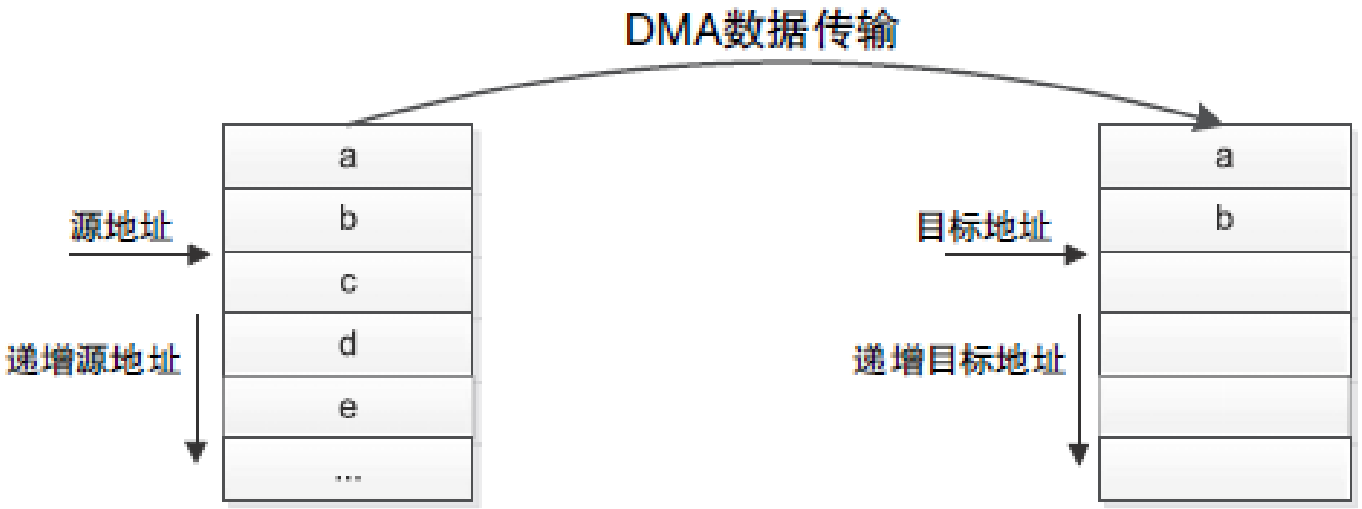
- 外设到存储器（Peripheral To Memory），例如ADC采集的数据存入内存中的缓存区
- 存储器到外设（Memory To Peripheral），例如将内存中的数据通过USART接口发出
- 存储器到存储器（Memory To Memory），例如外部SRAM中的数据复制到内存中，只有DMA2控制器有这种传输模式

数据宽度

数据宽度（Data width）是源和目标传输的基本数据单元的大小，有字节（Byte）、半字（Half word）和字（word）三种大小。

地址指针递增

外设和存储器指针是否递增



DMA工作模式

- 正常（Normal）模式是指传输完一个缓存区的数据后，DMA传输就停止了。
- 循环（Circular）模式是指启动一个缓存区的数据传输后，会循环执行这个DMA数据传输任务。

例如，HAL_UART_Receive_DMA()，正常模式只传输一次，循环模式就能连续接收

DMA流的优先级别

每个DMA流有一个可设置的软件优先级别，有4种：

- Very high（非常高）
- High（高）
- Medium（中等）
- Low（低）

如果两个DMA流的软件优先级别相同，则流编号更小的优先级别更高，流编号就是DMA流的硬件优先级。

13.1 DMA功能概述

13.2 DMA的HAL驱动程序

13.3 USART的DMA传输示例

13.2.1 DMA的HAL函数概述

分组	函数名	功能描述
初始化	HAL_DMA_Init()	DMA传输初始化配置
轮询方式	HAL_DMA_Start()	启动DMA传输，不开启DMA中断
	HAL_DMA_PollForTransfer()	轮询方式等待DMA传输结束，可设置一个超时等待时间
	HAL_DMA_Abort()	终止以轮询模式启动的DMA传输
中断方式	HAL_DMA_Start_IT()	启动DMA传输，开启DMA的中断
	HAL_DMA_Abort_IT()	终止以中断方式启动的DMA传输
	HAL_DMA_GetState()	获取DMA当前状态
	HAL_DMA_IRQHandler()	DMA中断ISR函数里调用的通用处理函数
双缓存区模式	HAL_DMAEx_MultiBufferStart()	启动双缓存区DMA传输，不开启DMA中断
	HAL_DMAEx_MultiBufferStart_IT()	启动双缓存区DMA传输，开启DMA中断
	HAL_DMAEx_ChangeMemory()	传输过程中改变缓存区地址

13.2.2 DMA传输的初始化设置

在CubeMX中可视化设计，自动生成初始化代码

函数HAL_DMA_Init()用于DMA传输初始化配置

```
HAL_StatusTypeDef HAL_DMA_Init(DMA_HandleTypeDef *hdma);
```

结构体DMA_HandleTypeDef定义DMA流对象，包含大量函数指针，用于指向DMA流中断的具体回调函数。

定义DMA传输属性的结构体类型

```
typedef struct
{
 uint32_t Channel;              //DMA通道，也就是外设的DMA请求
 uint32_t Direction;            //DMA传输方向
 uint32_t PeriphInc;            //外设地址指针是否自增
 uint32_t MemInc;               //存储器地址指针是否自增
 uint32_t PeriphDataAlignment;  //外设数据宽度
 uint32_t MemDataAlignment;     //存储器数据宽度
 uint32_t Mode;                 //传输模式，即循环模式或正常模式
 uint32_t Priority;             //DMA流的软件优先级别
 uint32_t FIFOMode;             //FIFO模式，是否使用FIFO
 uint32_t FIFOThreshold;        //FIFO阈值，1/4， 1/2， 3/4或1
 uint32_t MemBurst;             //存储器突发传输数据量
 uint32_t PeriphBurst;          //外设突发传输数据量
}DMA_InitTypeDef;
```

13.2.3 启动DMA传输

函数HAL_DMA_Start_IT()以中断方式启动DMA数据传输，其函数原型定义如下：

```
HAL_StatusTypeDef HAL_DMA_Start_IT(DMA_HandleTypeDef *hdma,
uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)
```

但一般由外设的函数启动DMA传输，如

```
HAL_StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef
    *huart, uint8_t *pData, uint16_t Size)

HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef
    *huart, uint8_t *pData, uint16_t Size)
```

13.2.4 DMA的中断

DMA的中断实际就是DMA流的中断。每个DMA流有独立的中断号，有对应的ISR函数。

```
#define DMA_IT_TC   ((uint32_t)DMA_SxCR_TCIE) //DMA传输完成中断事件
#define DMA_IT_HT   ((uint32_t)DMA_SxCR_HTIE) //DMA传输半完成中断事件
#define DMA_IT_TE   ((uint32_t)DMA_SxCR_TEIE)    //DMA传输错误中断事件


#define DMA_IT_DME  ((uint32_t)DMA_SxCR_DMEIE) //DMA直接模式错误中断事件
#define DMA_IT_FE   0x00000080U              //DMA FIFO上溢/下溢中断事件
```

DMA流中断事件对应的回调函数不是固定的，用函数指针来动态分配，与具体的外设有关。

DMA流中断事件与DMA流的回调函数指针的关系

DMA流中断事件	DMA流中断事件类型	DMA_HandleTypeDef 结构体中的函数指针
DMA_IT_TC	传输完成中断	XferCpltCallback
DMA_IT_HT	传输半完成中断	XferHalfCpltCallback
DMA_IT_TE	传输错误中断	XferErrorCallback
DMA_IT_FE	FIFO错误中断	无
DMA_IT_DME	直接模式错误中断	无

USART以DMA方式传输数据时，DMA流中断与回调函数的关系

USART的DMA传输函数	DMA流中断事件	DMA流对象的函数指针	DMA流事件中断关联的具体回调函数
HAL_UART_Transmit_DMA()	DMA_IT_TC	XferCpltCallback	HAL_UART_TxCpltCallback()
	DMA_IT_HT	XferHalfCpltCallback	HAL_UART_TxHalfCpltCallback()
HAL_UART_Receive_DMA()	DMA_IT_TC	XferCpltCallback	HAL_UART_RxCpltCallback()
	DMA_IT_HT	XferHalfCpltCallback	HAL_UART_RxHalfCpltCallback()

13.1 DMA功能概述

13.2 DMA的HAL驱动程序

13.3 USART的DMA传输示例

13.3.1示例功能与CubeMX项目设置

与第12章的示例Demo12_1CH340相同，但是串口数据传输采用DMA方式。复制项目。

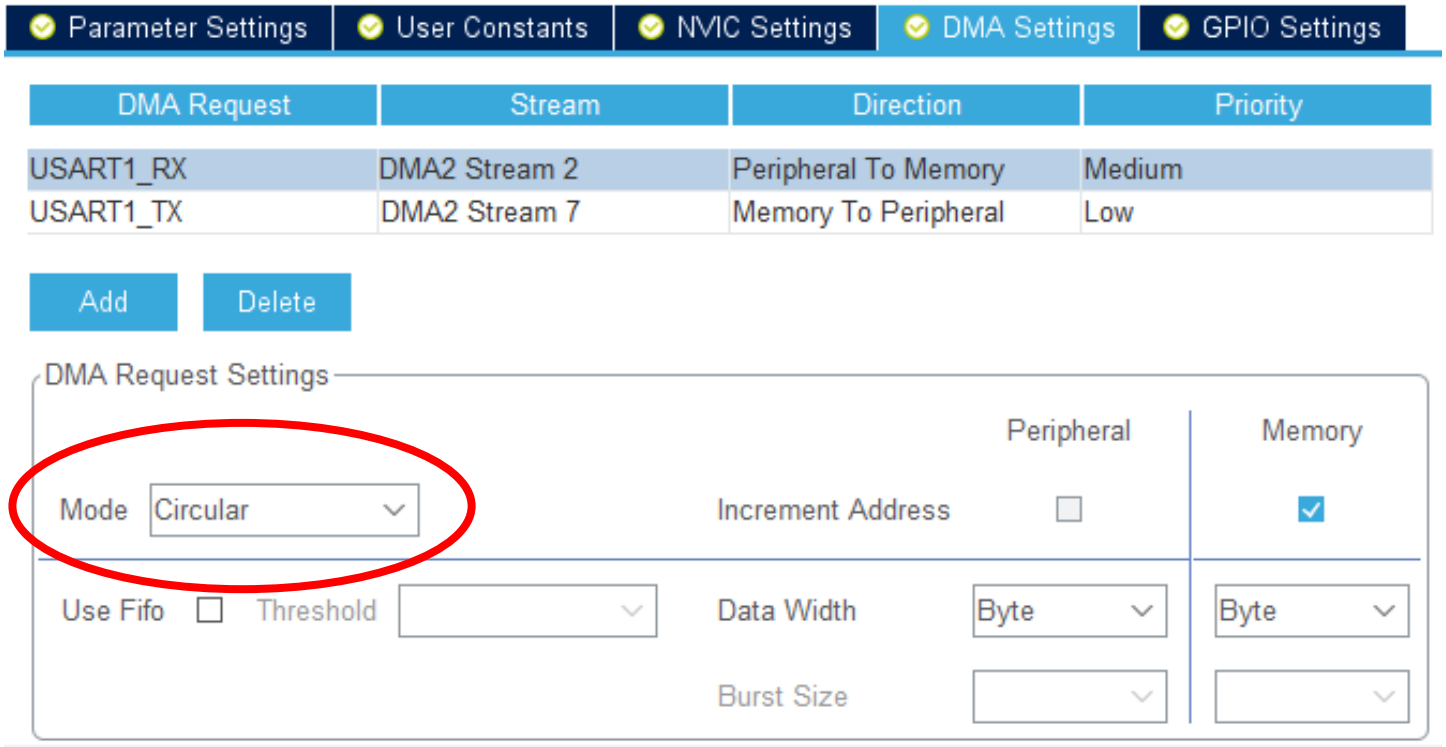


图13-3 DMA请求USART1_RX的DMA设置

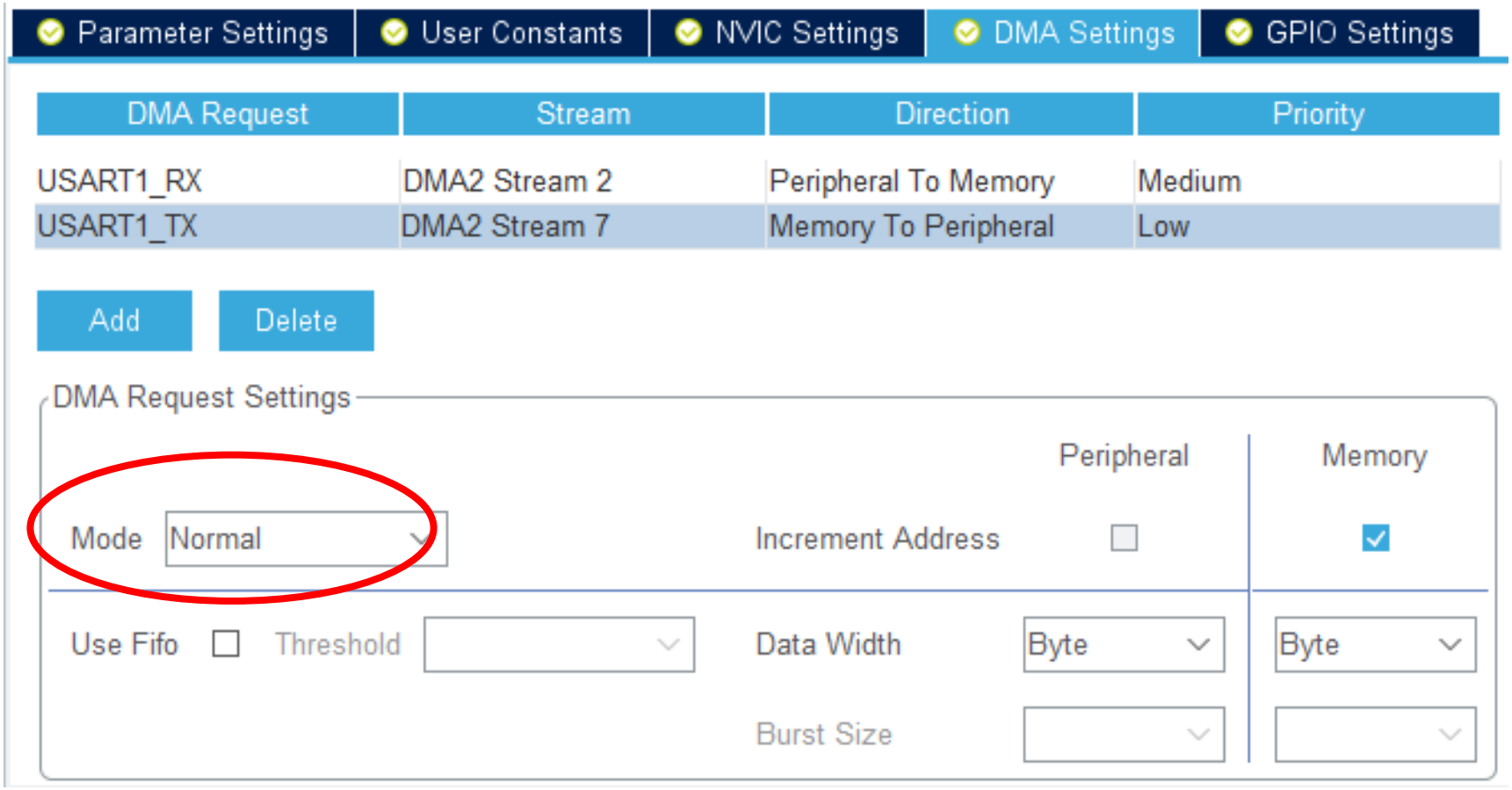


图13-4 DMA请求USART1_TX的DMA设置

DMA流有自己的中断号和ISR函数

NVIC Mode and Configuration

Configuration

NVIC | Code generation

Priority Group: 2 bits for pre... | Sort by Premption Priority and Sub Priority

Search: Sear... | Show only enabled interrupts | Force DMA channels 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
DMA2 stream2 global interrupt	☑	1	0
DMA2 stream7 global interrupt	☑	1	0

图13-5 中断优先级设置

13.3.2 程序实现和运行效果

1. 主程序

```
int main(void)
{
      /* USER CODE BEGIN 2 */
      TFTLCD_Init();
//  需要打开USART的全局中断，但是可以关闭中断事件
      __HAL_UART_DISABLE_IT(&huart1, UART_IT_TC);
      __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
      uint8_t  hello1[]="Hello,DMA transmit\n";
      HAL_UART_Transmit_DMA(&huart1,hello1,sizeof(hello1)); //DMA方式发送
      HAL_UART_Receive_DMA(&huart1, rxBuffer,RX_CMD_LEN); //DMA方式循环接收
      /* USER CODE END 2 */

      /* Infinite loop */
      while (1)
      {
      }
}
```

3. USART初始化

代码较长，看文件usart.c的源代码

```
#include "usart.h"

UART_HandleTypeDef  huart1;       //USART1外设对象变量
DMA_HandleTypeDef  hdma_usart1_rx;//DMA请求USART1_RX的 DMA流对象变量
DMA_HandleTypeDef  hdma_usart1_tx;//DMA请求USART1_TX的 DMA流对象变量
```

在完成了DMA流的初始化配置后，执行了下面的一行语句：

```
__HAL_LINKDMA(uartHandle, hdmarx, hdma_usart1_rx);
```

它相当于执行了如下的两行语句，就是互相设置了关联对象。

```
(&huart1)->hdmarx=&(hdma_usart1_rx);  //串口的hdmarx指向具体的DMA流对象
(hdma_usart1_rx).Parent=(&huart1);   //DMA流对象的Parent指向具体的串口对象
```

4. RTC周期唤醒中断的处理

读取RTC当前时间，并通过串口发送出去

```
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[20];   //时间字符串
       sprintf(timeStr,"%2d:%2d:%2d\n",sTime.Hours,sTime.Minutes,sTime.Seconds);
       LCD_ShowStr(30,100, timeStr);  //在LCD上显示当前时间
       if (isUploadTime)
       {
          HAL_UART_Transmit_DMA(&huart1,timeStr,strlen(timeStr));
          HAL_Delay(10);  //若要上位机正常显示换行，必须要有这个延时
       }
      }
}
```

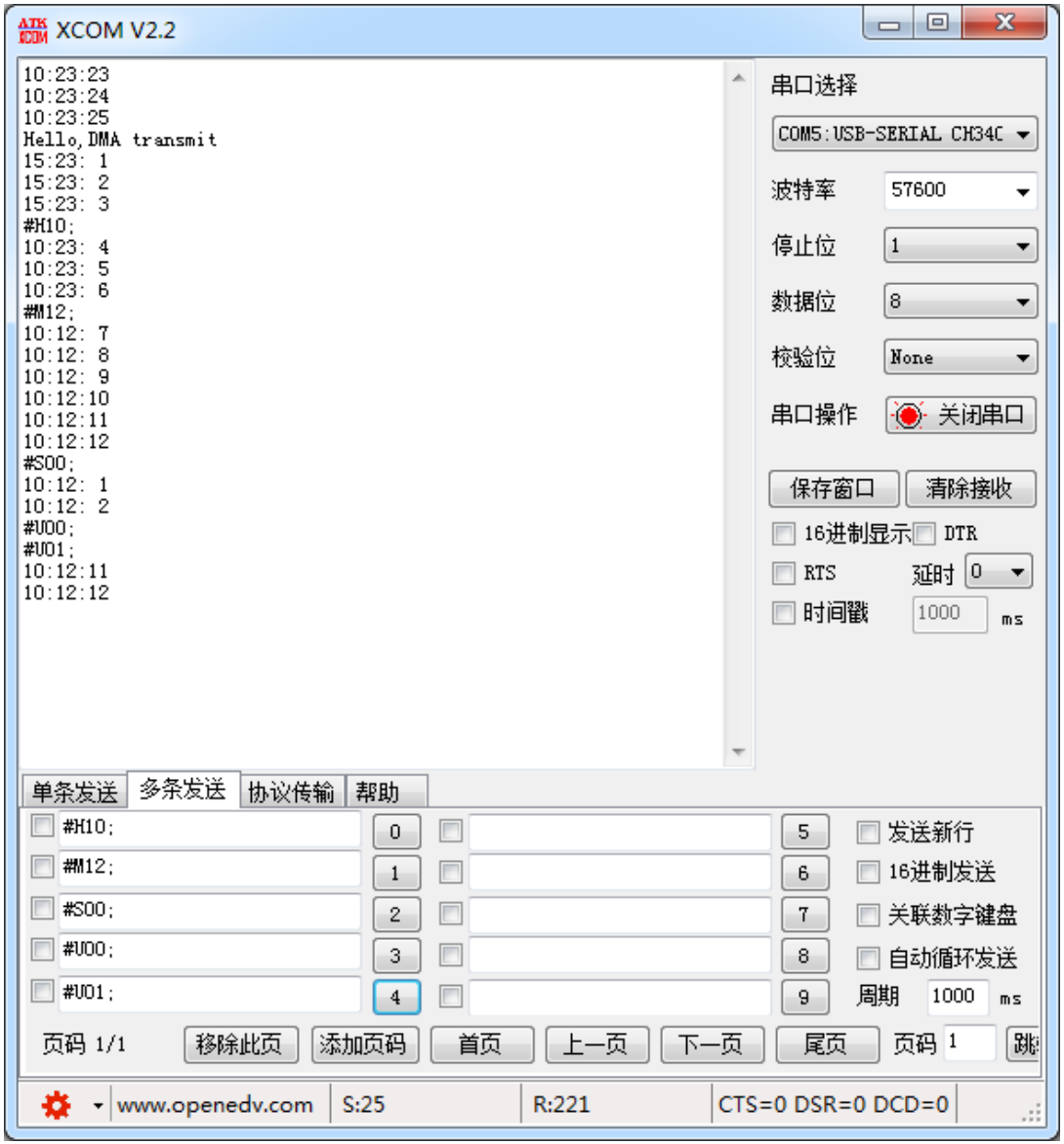
5. DMA中断处理

负责接收的DMA流是循环工作模式，会循环触发回调函数

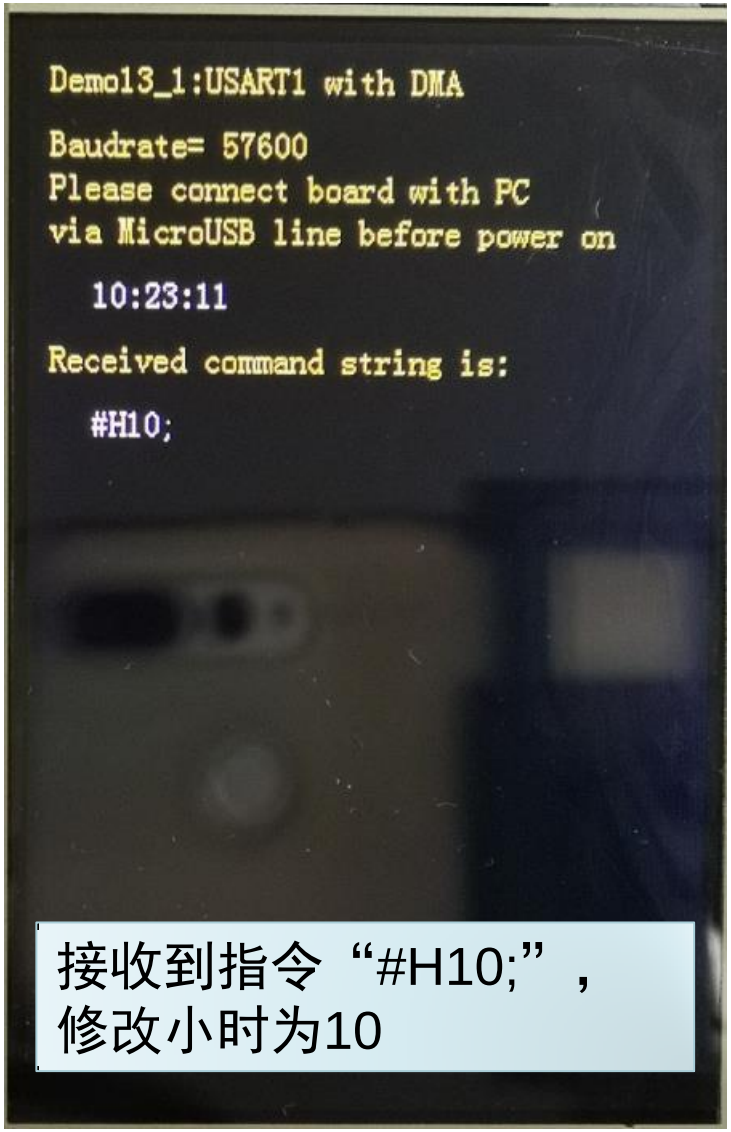
```
/*  hdma_usart1_rx的DMA传输完成事件中断回调函数   */
void HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart)
{
      if (huart->Instance == USART1)
      {
       for(uint16_t i=0;i<RX_CMD_LEN;i++)
          proBuffer[i]=rxBuffer[i];
       HAL_UART_Transmit_DMA(huart,rxBuffer,RX_CMD_LEN+1);//上传,带换行符
       HAL_Delay(10);  //必须加延时, updateRTCTime()才能正常处理
       updateRTCTime();  //指令解析处理
       LCD_ShowStr(30,170, (uint8_t *)rxBuffer);
      }
}
```

运行测试

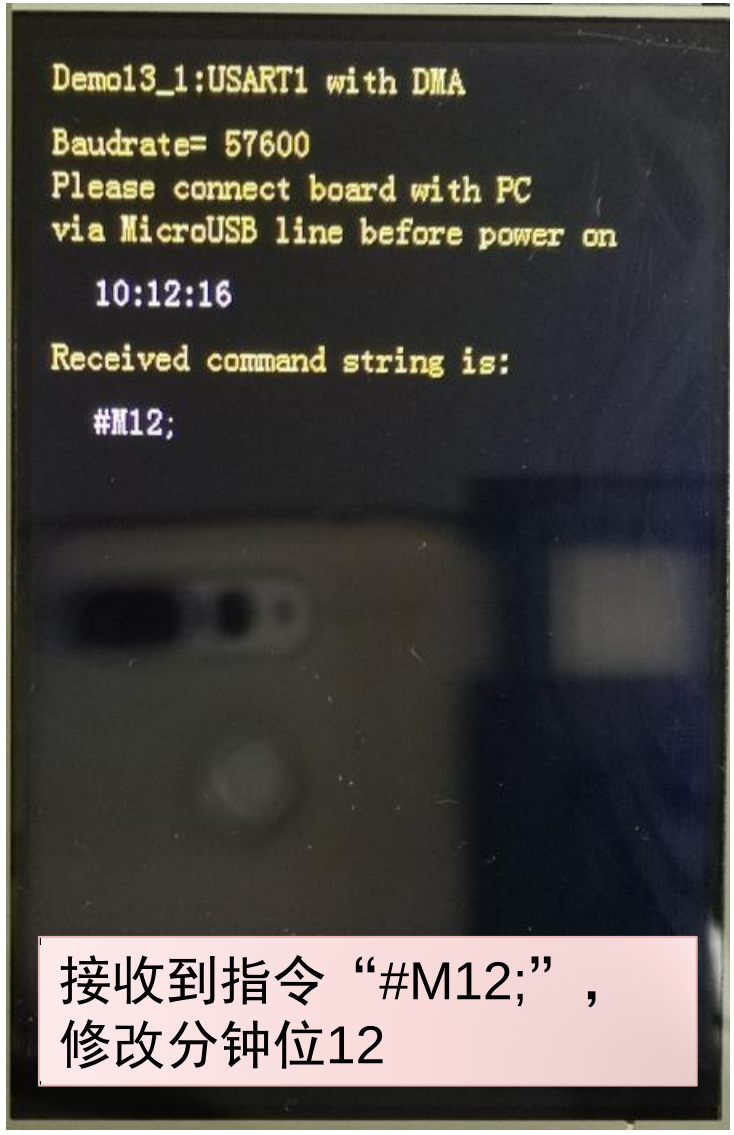
串口监视软件运行画面，接收并显示下位机上传的数据，并可以向下位机发送指令



运行测试



接收到指令“#H10;”，
修改小时为10



接收到指令“#M12;”，
修改分钟位12