* **在设置工程时, UART的TX/RX这两个管脚要设成PullUp;**
* 原子的示例程序用的是XCOM往STM板发数据, XCOM在发数据时会在数据后面添加**\r\n**；

# [STM32CubeIDE工程分析]

## 默认工程建好以后的初始化函数

工程建好以后int main()中有如下函数在while(1)之前：

### HAL\_Init();

这个函数按顺序设置了：

1. Configure the Flash prefetch, instruction and Data caches;
2. **HAL\_NVIC\_SetPriorityGrouping(NVIC\_PRIORITYGROUP\_4)**; 这里的**NVIC\_PRIORITYGROUP\_4**指的是：

#define NVIC\_PRIORITYGROUP\_4 0x00000003U /\* 4 bits for pre-emption priority

0 bits for sub-priority \*/

1. Configures the SysTick to generate an interrupt each 1 millisecond;
2. HAL\_MspInit(), 在这里是个空函数;

### SystemClock\_Config();

这个函数的定义就在main.c中, 在main()函数下方, 主要是设置了在.ioc中Clock Configuration的实现;

### MX\_GPIO\_Init();

这个函数的定义就在main.c中, 在main()函数下方, 主要是设置了在.ioc中被指定为GPIO的管脚的初始化;

### MX\_USART1\_UART\_Init();

这个函数比较复杂.

**首先**在**main.c的全局部分定义了UART\_HandleTypeDef** **huart1**; 这里的**UART\_HandleTypeDef**是对UART Controller的一个抽象, 其定义在HAL的**stm32f4xx\_hal\_uart.h**中:

**typedef** **struct** \_\_UART\_HandleTypeDef

{

**USART\_TypeDef** \*Instance; /\* UART registers base address \*/

**UART\_InitTypeDef** Init; /\* UART communication parameters \*/

uint8\_t \*pTxBuffPtr; /\* Pointer to UART Tx transfer Buffer \*/

uint16\_t TxXferSize; /\* UART Tx Transfer size \*/

\_\_IO uint16\_t TxXferCount; /\* UART Tx Transfer Counter \*/

uint8\_t \*pRxBuffPtr; /\* Pointer to UART Rx transfer Buffer \*/

uint16\_t RxXferSize; /\* UART Rx Transfer size \*/

\_\_IO uint16\_t RxXferCount; /\* UART Rx Transfer Counter \*/

DMA\_HandleTypeDef \*hdmatx; /\* UART Tx DMA Handle parameters \*/

DMA\_HandleTypeDef \*hdmarx; /\* UART Rx DMA Handle parameters \*/

HAL\_LockTypeDef Lock; /\* Locking object \*/

\_\_IO HAL\_UART\_StateTypeDef gState;

/\* UART state information related to global Handle management and also related to Tx operations.

This parameter can be a value of @ref HAL\_UART\_StateTypeDef \*/

\_\_IO HAL\_UART\_StateTypeDef RxState;

/\* UART state information related to Rx operations. This parameter can be a value of @ref HAL\_UART\_StateTypeDef \*/

\_\_IO uint32\_t ErrorCode; /\* UART Error code \*/

**#if** (**USE\_HAL\_UART\_REGISTER\_CALLBACKS == 1**)

**void** (\* TxHalfCpltCallback)(**struct** \_\_UART\_HandleTypeDef \*huart);

/\* UART Tx Half Complete Callback \*/

**void** (\* TxCpltCallback)(**struct** \_\_UART\_HandleTypeDef \*huart);

/\* UART Tx Complete Callback \*/

**void** (\* RxHalfCpltCallback)(**struct** \_\_UART\_HandleTypeDef \*huart);

/\* UART Rx Half Complete Callback \*/

**void** (\* RxCpltCallback)(**struct** \_\_UART\_HandleTypeDef \*huart);

/\* UART Rx Complete Callback \*/

**void** (\* ErrorCallback)(**struct** \_\_UART\_HandleTypeDef \*huart);

/\* UART Error Callback \*/

**这里还有一些其他的callback functionz声明**

**#endif** /\* USE\_HAL\_UART\_REGISTER\_CALLBACKS \*/

} **UART\_HandleTypeDef;**

**这里前两个成员比较重要:**

1. **USART\_TypeDef** \*Instance;

**typedef** **struct**

{

\_\_IO uint32\_t SR; /\* USART Status register, Address offset: 0x00 \*/

\_\_IO uint32\_t DR; /\* USART Data register, Address offset: 0x04 \*/

\_\_IO uint32\_t BRR; /\* USART Baud rate register, Address offset: 0x08 \*/

\_\_IO uint32\_t CR1; /\* USART Control register 1, Address offset: 0x0C \*/

\_\_IO uint32\_t CR2; /\* USART Control register 2, Address offset: 0x10 \*/

\_\_IO uint32\_t CR3; /\* USART Control register 3, Address offset: 0x14 \*/

\_\_IO uint32\_t GTPR; /\* USART Guard time and pre-scaler register, Address offset: 0x18 \*/

} **USART\_TypeDef**;

1. **UART\_InitTypeDef** Init; **定义了波特率, 数据长度, 停止位, Parity Check和采样率之类的.**
2. **#if** (**USE\_HAL\_UART\_REGISTER\_CALLBACKS == 1**) 这个宏决定了你的工程要不要用callback function. 在ioc文件里Advanced Settings右边有Register Callback列, 选中UART后Enable, 这个宏就变成1了.

**接着看MX\_USART1\_UART\_Init()本身.**在设置了USART1和基本参数后, 重要的是**HAL\_UART\_Init(&huart1)**. 这个函数定义了:

1. 把Callback function让UART Handler调用;
2. Set the UART Communication parameters;
3. 清除了非UART模式下的一些不用的寄存器;
4. **/\* Initialize the UART state \*/**

huart->ErrorCode = HAL\_UART\_ERROR\_NONE; // 0x00000000U

huart->gState = *HAL\_UART\_STATE\_READY*;

huart->RxState = *HAL\_UART\_STATE\_READY*;

至此, 默认的初始化函数分析结束.

## 默认工程建好以后的在stm32f4xx\_it.c中的中断函数

**void** **USART1\_IRQHandler**(**void**)

{

/\* USER CODE BEGIN USART1\_IRQn 0 \*/

/\* USER CODE END USART1\_IRQn 0 \*/

**HAL\_UART\_IRQHandler(&huart1);**

/\* USER CODE BEGIN USART1\_IRQn 1 \*/

/\* USER CODE END USART1\_IRQn 1 \*/

}

HAL\_UART\_IRQHandler()的定义在**stm32f4xx\_hal\_uart.c**中.

**它的流程是**:

首先读取SR (isrflags)和CR(crits)寄存器的值, 设置一个errorflags = 0x00; **然后是用if区分出来的4个平行的判断逻辑**:

### /\* If UART receives something and no error occurs \*/

/\* If no error occurs \*/

errorflags = (isrflags & (uint32\_t)(USART\_SR\_PE | USART\_SR\_FE | USART\_SR\_ORE | USART\_SR\_NE));

**if** (errorflags == *RESET*)

{

/\* UART in mode Receiver -------------------------------------------------\*/

**if** (((isrflags & **USART\_SR\_RXNE**) != *RESET*) && ((cr1its & **USART\_CR1\_RXNEIE**) != *RESET*))

{

**UART\_Receive\_IT(huart);**

**return**;

}

}

很明确, 当有中断产生(isrflags[9:4])并且没有错误产生(isrflags[3:0]是UART的4个Error)时，如果中断是RX Buffer Not Empty(RXNE), 调用**UART\_Receive\_IT(huart);**

#### UART\_Receive\_IT()

**typedef** **enum**

{

HAL\_OK = 0x00U,

HAL\_ERROR = 0x01U,

HAL\_BUSY = 0x02U,

HAL\_TIMEOUT = 0x03U

} **HAL\_StatusTypeDef**;

**static** HAL\_StatusTypeDef **UART\_Receive\_IT**(UART\_HandleTypeDef \*huart)

对于最常见的, 也是本工程设定的UART\_WORDLENGTH\_8B和UART\_PARITY\_NONE的情况, 其实质:

**\*huart->pRxBuffPtr++ = (uint8\_t)(huart->Instance->DR & (uint8\_t)0x00FF);**

然后紧接着判断

**if** (**--huart->RxXferCount == 0U**)

{

/\* Disable the UART Data Register not empty Interrupt \*/

\_\_HAL\_UART\_DISABLE\_IT(huart, UART\_IT\_RXNE);

/\* Disable the UART Parity Error Interrupt \*/

\_\_HAL\_UART\_DISABLE\_IT(huart, UART\_IT\_PE);

/\* Disable the UART Error Interrupt: (Frame error, noise error, overrun error) \*/

\_\_HAL\_UART\_DISABLE\_IT(huart, UART\_IT\_ERR);

/\* Rx process is completed, restore huart->RxState to Ready \*/

huart->RxState = *HAL\_UART\_STATE\_READY*;

**#if** (**USE\_HAL\_UART\_REGISTER\_CALLBACKS == 1**)

/\*Call registered Rx complete callback\*/

**huart->RxCpltCallback(huart);**

**#else**

/\*Call legacy weak Rx complete callback\*/

**HAL\_UART\_RxCpltCallback(huart);**

**#endif** /\* USE\_HAL\_UART\_REGISTER\_CALLBACKS \*/

**return** *HAL\_OK*;

}

所以只有当初始RxXferCount == 1时上面的接收完的处理才会执行. **所以UART\_Receive\_IT()最好用于单Byte的接收;**

在USE\_HAL\_UART\_REGISTER\_CALLBACKS == 1时, MX\_USART1\_UART\_Init()中已经把huart->RxCpltCallback(huart)指向了HAL\_UART\_RxCpltCallback(huart)，**所以前面的在ioc中特意使能CallBack并无多大意义, 因为总是调用HAL\_UART\_RxCpltCallback(huart)**;

**所以，中断函数HAL\_UART\_IRQHandler(&huart1)本身用于接收时最好单Byte接受, 而且这个函数本身不返回数据, 数据在其参数huart1的成员指针指向的地方;**

### /\* If some errors occur \*/

**首先根据error的类型设置huart->ErrorCode;**

然后:

1. 如果RXNE被set了, 用UART\_Receive\_IT()完成本次RX;
2. 如果是DMA接收的, error会blocking进一步的接受. 所以中断DMA, **然后调用HAL\_UART\_ErrorCallback(huart)**;
3. 如果不是DMA接收的, Non-Blocking error and transfer could go on, 调用HAL\_UART\_ErrorCallback(huart), **然后清除了huart->ErrorCode；**
4. **然后 return;**

### /\* UART in mode Transmitter \*/

**if** (((isrflags & USART\_SR\_TXE) != *RESET*) && ((cr1its & USART\_CR1\_TXEIE) != *RESET*))

{

**UART\_Transmit\_IT(huart);**

**return**;

}

#### UART\_Transmit\_IT(huart);

对于最常见的, 也是本工程设定的UART\_WORDLENGTH\_8B的情况, 其实质:

**huart->Instance->DR = (uint8\_t)(\*huart->pTxBuffPtr++ & (uint8\_t)0x00FF);**

**这里, 往DR里写入数据就能引起发送了, 因为：**When **no transmission is taking place, a write instruction to the USART\_DR register** places the data directly in the shift register, the data transmission starts, and the **TXE bit is immediately set**.

**随后：**

**if** (--huart->TxXferCount == 0U)

{

/\* Disable the UART Transmit Complete Interrupt \*/

\_\_HAL\_UART\_DISABLE\_IT(huart, UART\_IT\_TXE);

/\* Enable the UART Transmit Complete Interrupt \*/

\_\_HAL\_UART\_ENABLE\_IT(huart, UART\_IT\_TC);

}

**return** *HAL\_OK*;

**--huart->TxXferCount == 0U代表本次传输完成, 所以把TXE给disable, 但把TC给使能. 这里也能看出, 只有在指定的数据数目被发送完后TC才会出现. 但如果指定每次都是单Byte发送, 那就是每Byte发送完出现TC.**

### /\* UART in mode Transmitter end \*/

**if** (((isrflags & USART\_SR\_TC) != *RESET*) && ((cr1its & USART\_CR1\_TCIE) != *RESET*))

{

**UART\_EndTransmit\_IT(huart);**

**return**;

}

#### UART\_EndTransmit\_IT(huart);

**static** HAL\_StatusTypeDef **UART\_EndTransmit\_IT**(UART\_HandleTypeDef \*huart)

{

/\* Disable the UART Transmit Complete Interrupt \*/

**\_\_HAL\_UART\_DISABLE\_IT(huart, UART\_IT\_TC);**

/\* Tx process is ended, restore huart->gState to Ready \*/

huart->gState = *HAL\_UART\_STATE\_READY*;

**#if** (USE\_HAL\_UART\_REGISTER\_CALLBACKS == 1)

/\*Call registered Tx complete callback\*/

**huart->TxCpltCallback(huart);**

**#else**

/\*Call legacy weak Tx complete callback\*/

**HAL\_UART\_TxCpltCallback(huart);**

**#endif** /\* USE\_HAL\_UART\_REGISTER\_CALLBACKS \*/

**return** *HAL\_OK*;

}

**至此, 工程默认的中断函数分析完毕. 注意，UART的中断函数是既负责发送又负责接收的.**

## UART HAL Function归纳

在中断函数中出现的三个函数:

**static** HAL\_StatusTypeDef **UART\_Transmit\_IT**(UART\_HandleTypeDef \*huart);

**static** HAL\_StatusTypeDef **UART\_EndTransmit\_IT**(UART\_HandleTypeDef \*huart);

**static** HAL\_StatusTypeDef **UART\_Receive\_IT**(UART\_HandleTypeDef \*huart);

**注意他们都是static的**, 意味着他们只在他们定义所在的文件, **stm32f4xx\_hal\_uart.c**中可见. 实际上, **他们三个都只在中断函数void USART1\_IRQHandler(void)中被使用.**

**UART的API:**

**Blocking mode API's are :**

(+) HAL\_UART\_Transmit()

(+) HAL\_UART\_Receive()

**Non-Blocking mode API's with Interrupt are :**

(+) HAL\_UART\_Transmit\_IT()

(+) HAL\_UART\_Receive\_IT()

(+) HAL\_UART\_IRQHandler()

The **HAL\_UART\_TxCpltCallback()**, **HAL\_UART\_RxCpltCallback()** user callbacks will be executed respectively at the end of the transmit or receive process. The HAL\_UART\_ErrorCallback()user callback will be executed when a communication error is detected.

### HAL\_UART\_Transmit\_IT()

**HAL\_StatusTypeDef HAL\_UART\_Transmit\_IT(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size)**

{

/\* Check that a Tx process is not already ongoing \*/

**if** (huart->gState == *HAL\_UART\_STATE\_READY*)

{

**if** ((pData == NULL) || (Size == 0U))

{

**return** *HAL\_ERROR*;

}

/\* Process Locked \*/

\_\_HAL\_LOCK(huart);

huart->pTxBuffPtr = pData; **// 在中断函数里搬进DR寄存器**

huart->TxXferSize = Size;

huart->TxXferCount = Size;

huart->ErrorCode = HAL\_UART\_ERROR\_NONE;

huart->gState = *HAL\_UART\_STATE\_BUSY\_TX*;

/\* Process Unlocked \*/

\_\_HAL\_UNLOCK(huart);

/\* Enable the UART Transmit data register empty Interrupt \*/

**\_\_HAL\_UART\_ENABLE\_IT(huart, UART\_IT\_TXE)**;

**return** *HAL\_OK*;

}

**else**

{

**return** *HAL\_BUSY*;

}

}

这里, **\_\_HAL\_UART\_ENABLE\_IT(huart, UART\_IT\_TXE)中开启了TXEIE中断使能:**

**UART\_IT\_TXE是(1 << 28U | 1 << 7U);**

**UART\_CR1\_REG\_INDEX == 1;**

**UART\_IT\_MASK == 0x0000FFFFU**

**#define** \_\_HAL\_UART\_ENABLE\_IT(\_\_HANDLE\_\_, \_\_INTERRUPT\_\_)

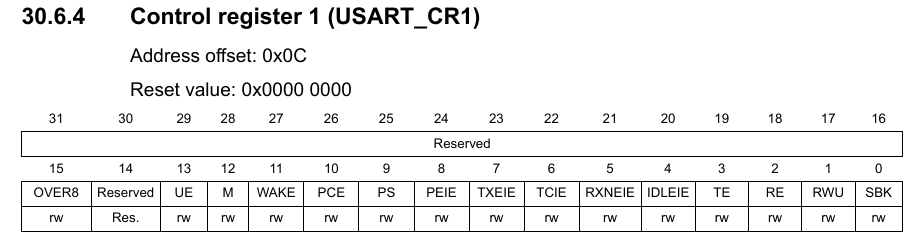
((((\_\_INTERRUPT\_\_) >> 28U) == UART\_CR1\_REG\_INDEX)?

((\_\_HANDLE\_\_)->Instance->CR1 |= ((\_\_INTERRUPT\_\_) & UART\_IT\_MASK)):

(((\_\_INTERRUPT\_\_) >> 28U) == UART\_CR2\_REG\_INDEX)?

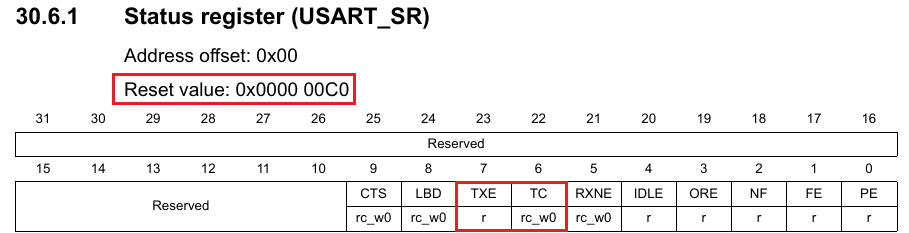
((\_\_HANDLE\_\_)->Instance->CR2 |= ((\_\_INTERRUPT\_\_) & UART\_IT\_MASK)): ((\_\_HANDLE\_\_)->Instance->CR3 |= ((\_\_INTERRUPT\_\_) & UART\_IT\_MASK)))

**所以最终结果是把CR1设为(1 << 7U), 使能了TXEIE.**

****

**在第一次发送时:**

1. 在开启了TXEIE中断使能之后，程序就陷入了中断, **因为TXE初始是1.**

****

1. **进入中断后在Section 2.3 /\* UART in mode Transmitter \*/**中, 调用UART\_Transmit\_IT(huart);
2. UART\_Transmit\_IT(huart)中真正执行的是:

huart->Instance->DR = (uint8\_t)(\*huart->pTxBuffPtr++ & (uint8\_t)0x00FF);

**if** (--huart->TxXferCount == 0U)

{

/\* Disable the UART Transmit Complete Interrupt \*/

\_\_HAL\_UART\_DISABLE\_IT(huart, UART\_IT\_TXE); **// 把TXEIE清0**

/\* Enable the UART Transmit Complete Interrupt \*/

\_\_HAL\_UART\_ENABLE\_IT(huart, UART\_IT\_TC); **// 置TCIE为1**

}

**return** *HAL\_OK*;

1. **这时Section 2.4 /\* UART in mode Transmitter end \*/的条件满足了, 进入，执行UART\_EndTransmit\_IT(huart)：**

\_\_HAL\_UART\_DISABLE\_IT(huart, UART\_IT\_TC);

再调用HAL\_UART\_TxCpltCallback(huart);

### HAL\_UART\_Receive\_IT()

流程几乎一样, 最后\_\_HAL\_UART\_ENABLE\_IT(huart, UART\_IT\_RXNE); **等待接收数据的到来**. 等到1Byte数据到来, RXNE中断被触发, 进入 Section 2.1 /\* If UART receives something and no error occurs \*/.

## HUART状态变量的追踪

按照顺序来:

### MX\_USART1\_UART\_Init(void)里最后:

/\* Initialize the UART state \*/

huart->ErrorCode = HAL\_UART\_ERROR\_NONE;

huart->gState = *HAL\_UART\_STATE\_READY*;

huart->RxState = *HAL\_UART\_STATE\_READY*;

### 然后进入main(), issue一个HAL\_UART\_Receive\_IT().

**HAL\_UART\_Receive\_IT()**会先判断if (huart->RxState == HAL\_UART\_STATE\_READY), 如果成立才执行; 在使能中断前会huart->RxState = HAL\_UART\_STATE\_BUSY\_RX;

然后进入中断函数. 以下几个分支:

1. **RX并且没有错误:**

**if** (huart->RxState == HAL\_UART\_STATE\_BUSY\_RX)是这个分支执行的前提条件; 这个分支执行结束后会有huart->RxState = HAL\_UART\_STATE\_READY; 然后是callback function.

所以RxCpltCallback()里不需要重置huart->RxState;

1. **RX并且有错误:**

流程和上面基本一样, 只是多了ErrorCode的设置;