第5讲

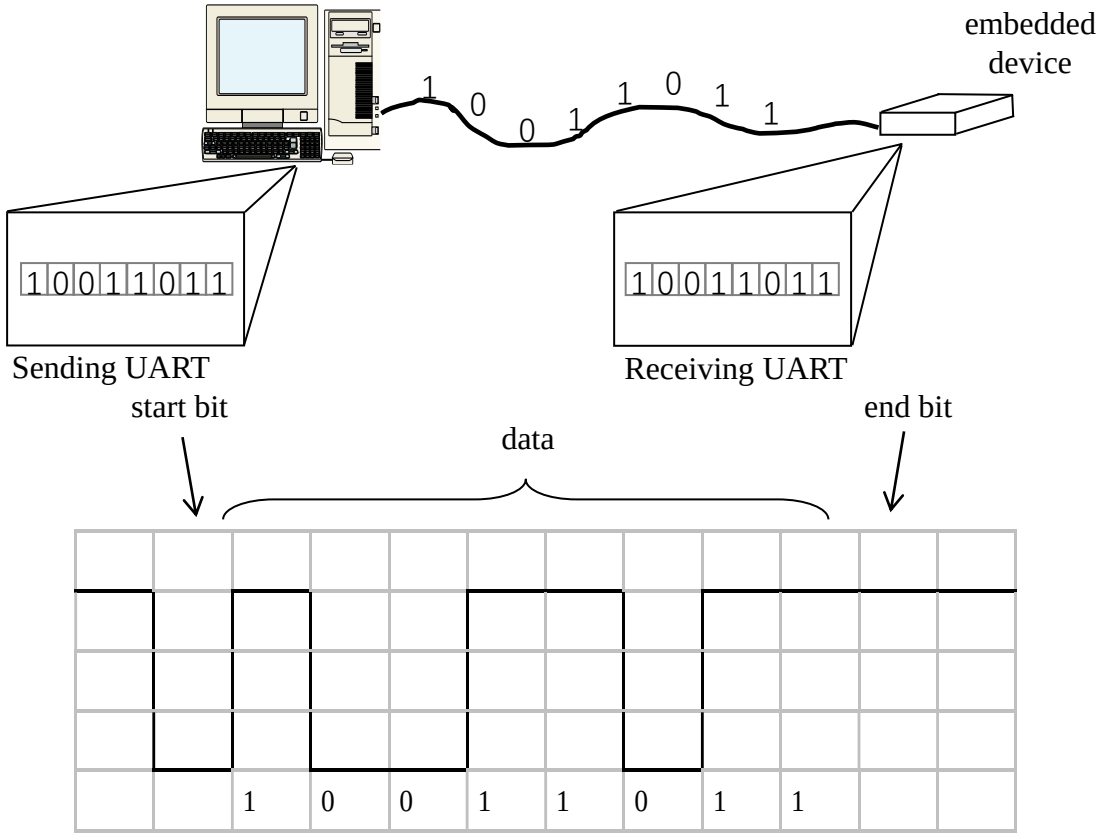
串行通信

主要内容

- **STM32G4的串口**
- **串行通信的编程实现方式**
  - ✓ 查询、中断和DMA
- **动手练习5**

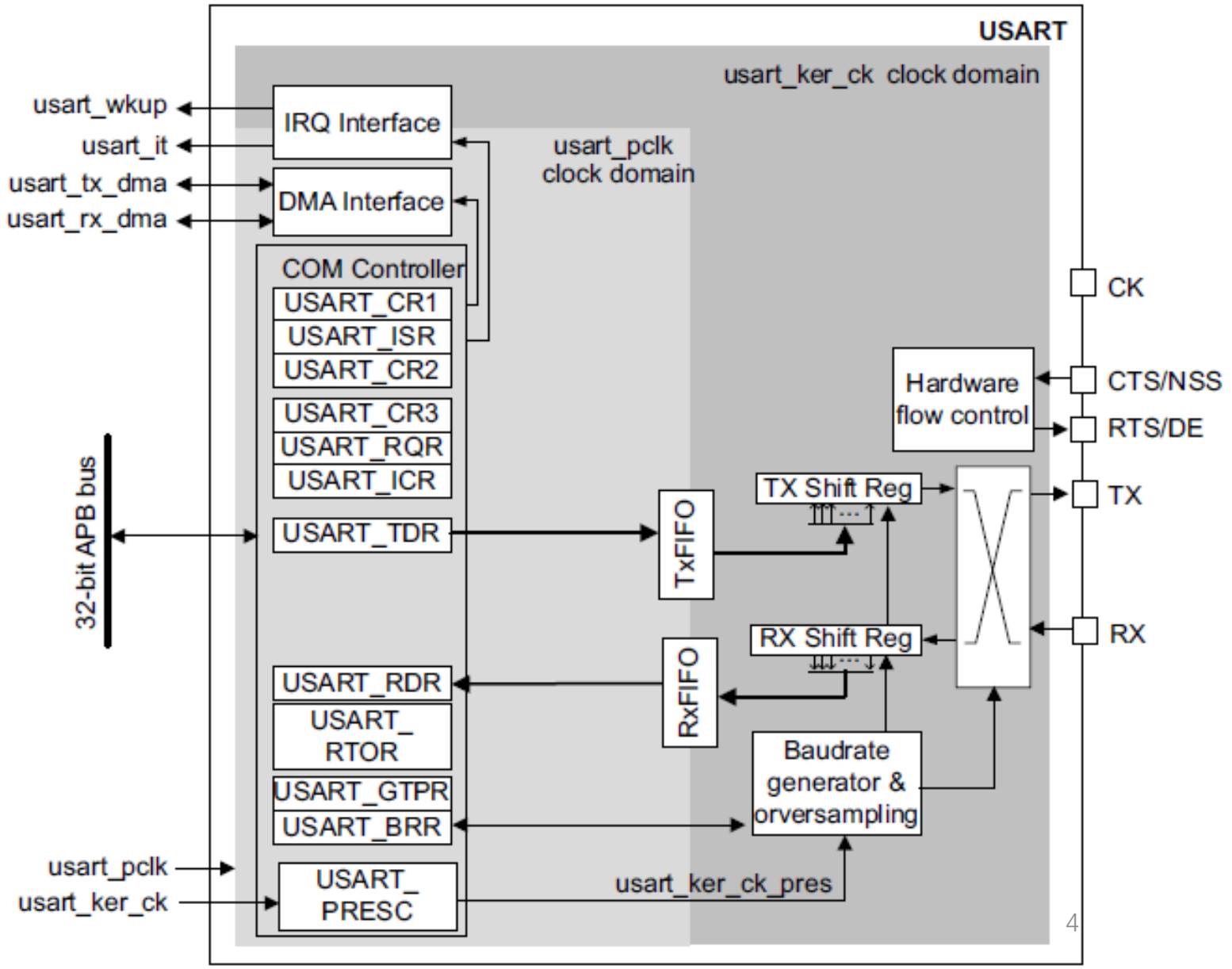
Serial Transmission Using UARTs

- UART: Universal Asynchronous Receiver Transmitter
  - Takes parallel data and transmits serially
  - Receives serial data and converts to parallel
- Parity: extra bit for simple error checking
- Start bit, stop bit
- Baud rate
  - signal changes per second
  - bit rate usually higher



STM32G4的串口

- USART1, USART2, USART3
- UART4, UART5
- LPUART1



USART block diagram

练习5：串行通信

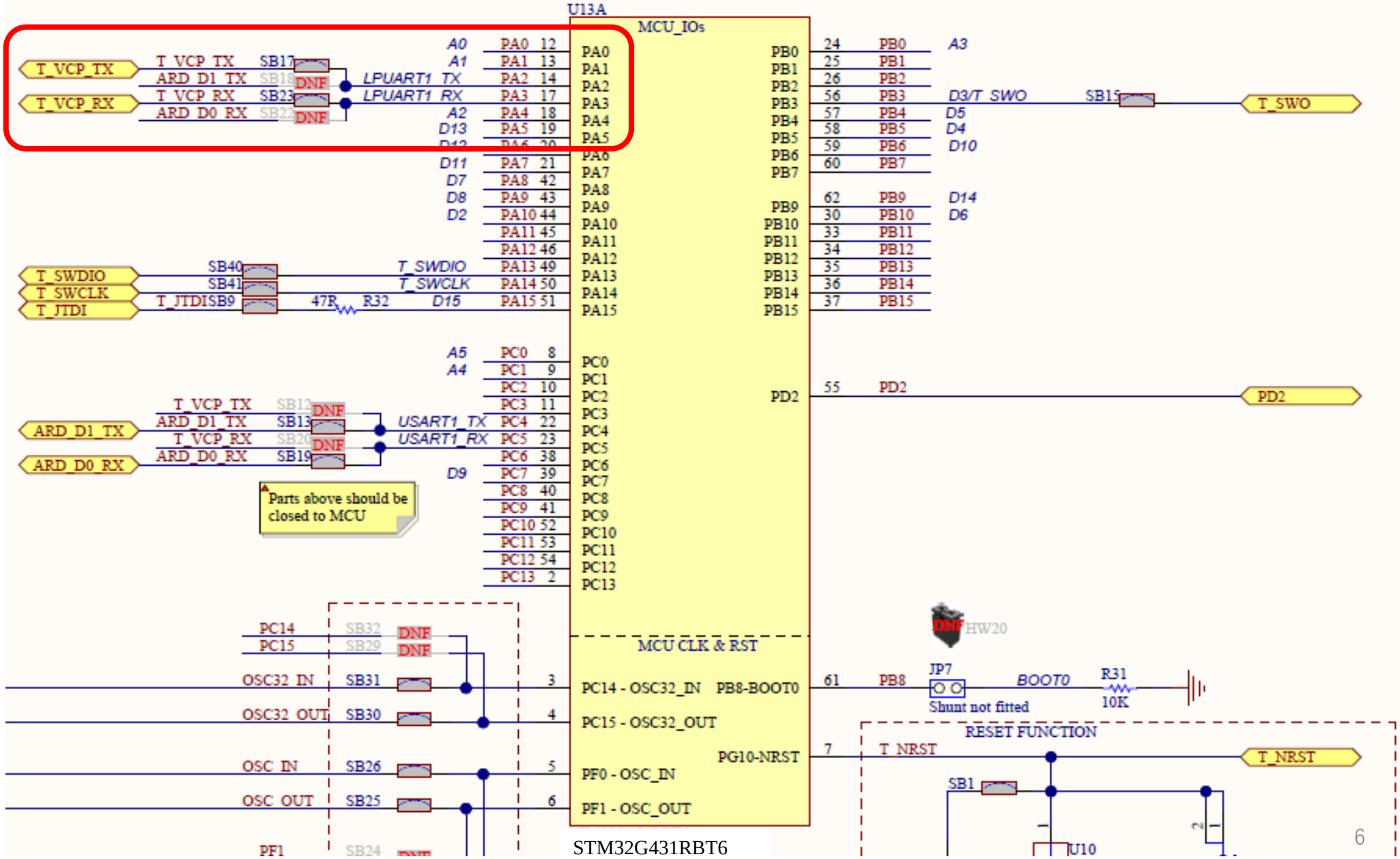
■ 完成功能:

✓ 实现NUCLEO板与PC的串行通信；在PC机上，使用串口助手工具，给MCU发送命令，实现点亮发光二极管，让蜂鸣器响的功能

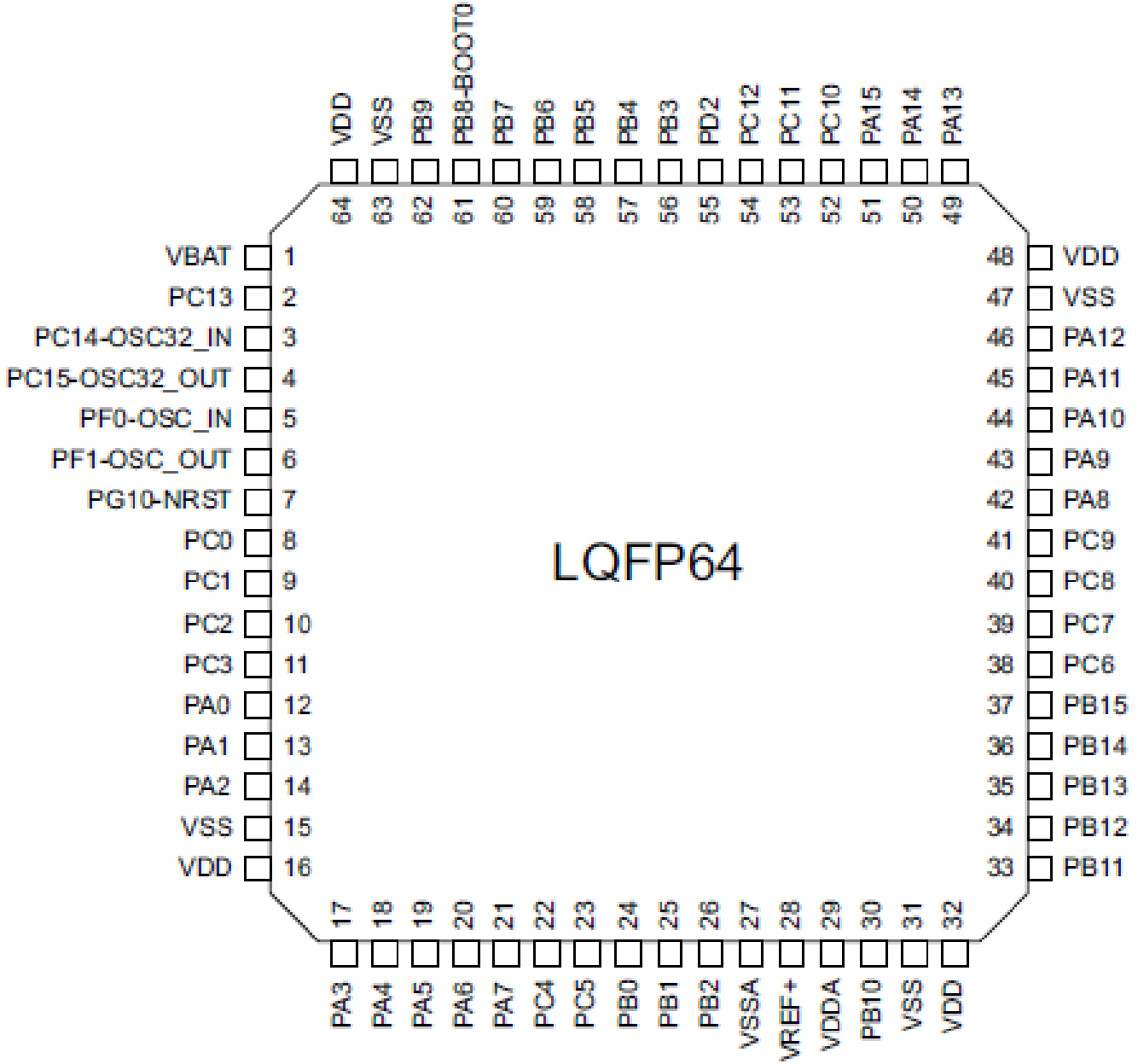
✓ PA5 -> LD2； PC13 <- B1； PA4 ->蜂鸣器

✓ NUCLEO-G431RB板上用ST-Link实现了一个虚拟串口，使用的是STM32G431RB上的USART2模块。USART2默认引脚是PA2（USART2_TX）和PA3（USART2_RX）

NUCLEO64 STM32G4



STM32G431xB/xC/xE LQFP64 pinout



STM32G431xB/xC/xE pin definition

Pin Number								Pin name (function after reset)(1)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
WLCSP81	UFQFPN48	LQFP48	LQFP64	LQFP80	TFBGA100	LPQF100	LPQF128						
G8	10	10	14	14	H1	22	29	PA2	I/O	FT_a	-	TIM2_CH3, TIM5_CH3, USART2_TX, COMP2_OUT, TIM15_CH1, QUADSPI1_BK1_NCS, LPUART1_TX, UCPD1_FRSTX, EVENTOUT	ADC1_IN3, COMP2_INM, OPAMP1_VOUT, WKUP4/LSCO
H9	-	-	15	15	D6	23	30	VSS	S	-	-	-	-
J9	-	-	16	16	D7	24	31	VDD	S	-	-	-	-
H8	11	11	17	17	H3	25	32	PA3	I/O	TT_a	-	TIM2_CH4, TIM5_CH4, SAI1_CK1, USART2_RX, TIM15_CH2, QUADSPI1_CLK, LPUART1_RX, SAI1_MCLK_A, EVENTOUT	ADC1_IN4, COMP2_INP, OPAMP1_VINM/ OPAMP 1_VINP, OPAMP5_VINM

投票

最多可选4项

设置

电路板上MCU某引脚（端口）的最终功能与哪些因素相关?

A 该引脚外接的电路

B 该芯片的**Datasheet**

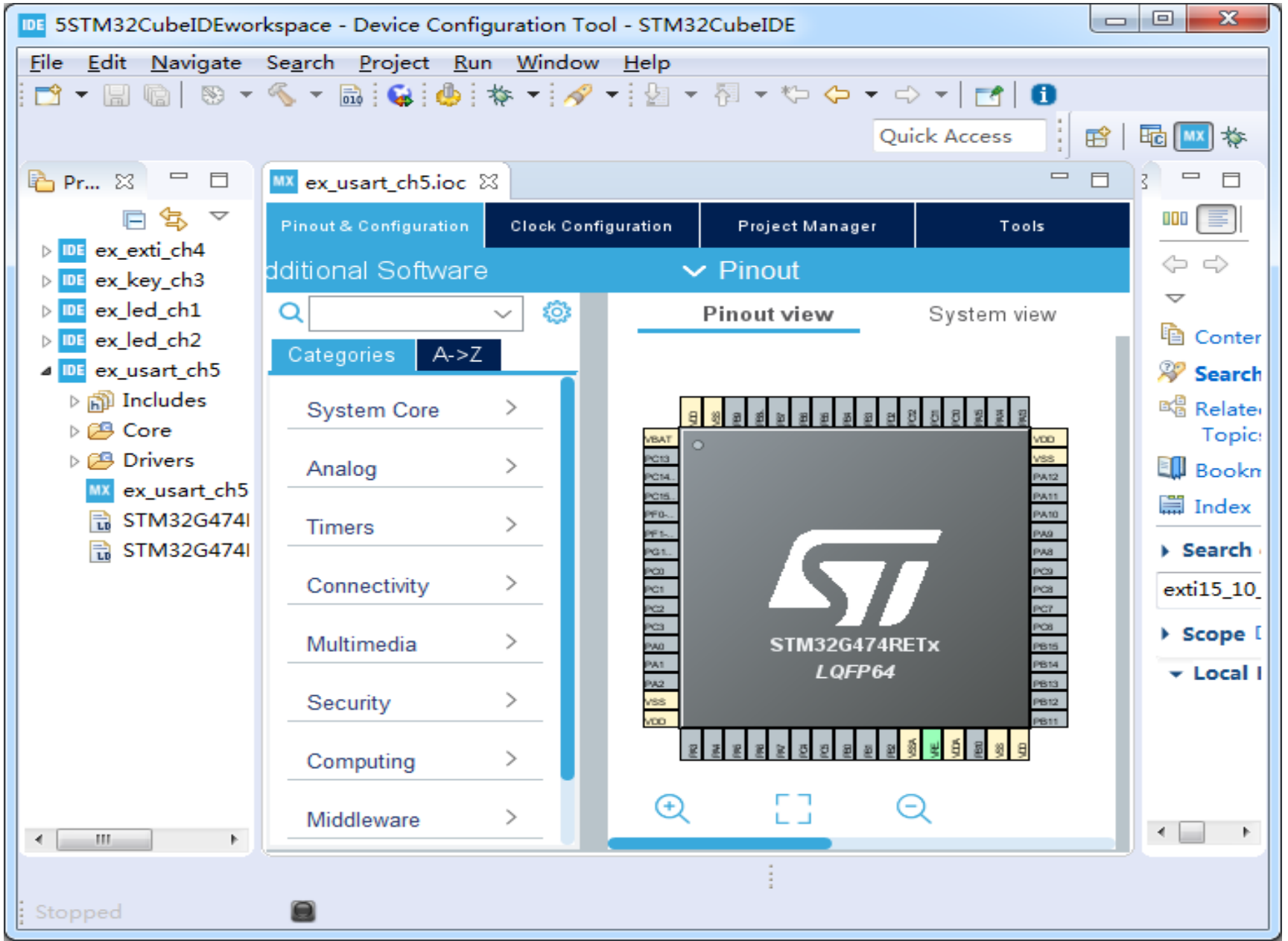
C 对该引脚的寄存器配置

D 用户代码

提交

用中断方式实现串行数据接收

建立新工程


5STM32CubeIDEworkspace - Device Configuration Tool - STM32CubeIDE
File Edit Navigate Search Project Run Window Help
Quick Access
ex_usart_ch5.ioc
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Software
Pinout
Pinout view
System view
Categories
A->Z
System Core
Analog
Timers
Connectivity
Multimedia
Security
Computing
Middleware
ex_exti_ch4
ex_key_ch3
ex_led_ch1
ex_led_ch2
ex_usart_ch5
Includes
Core
Drivers
ex_usart_ch5
STM32G474I
STM32G474I
STM32G474RETx
LQFP64
Search
exti15_10_
Scope
Local
Stopped

配置GPIO

- 配置PA4、PA5为输出（GPIO_Ooutput）
- 配置PC13为GPIO_EXTI13（中断模式）。保留PC13作为外部中断，是为了与串口中断进行对比。PC13用于检测按键B1的状态
- 配置PA2和PA3分别为USART2_TX和USART2_RX。

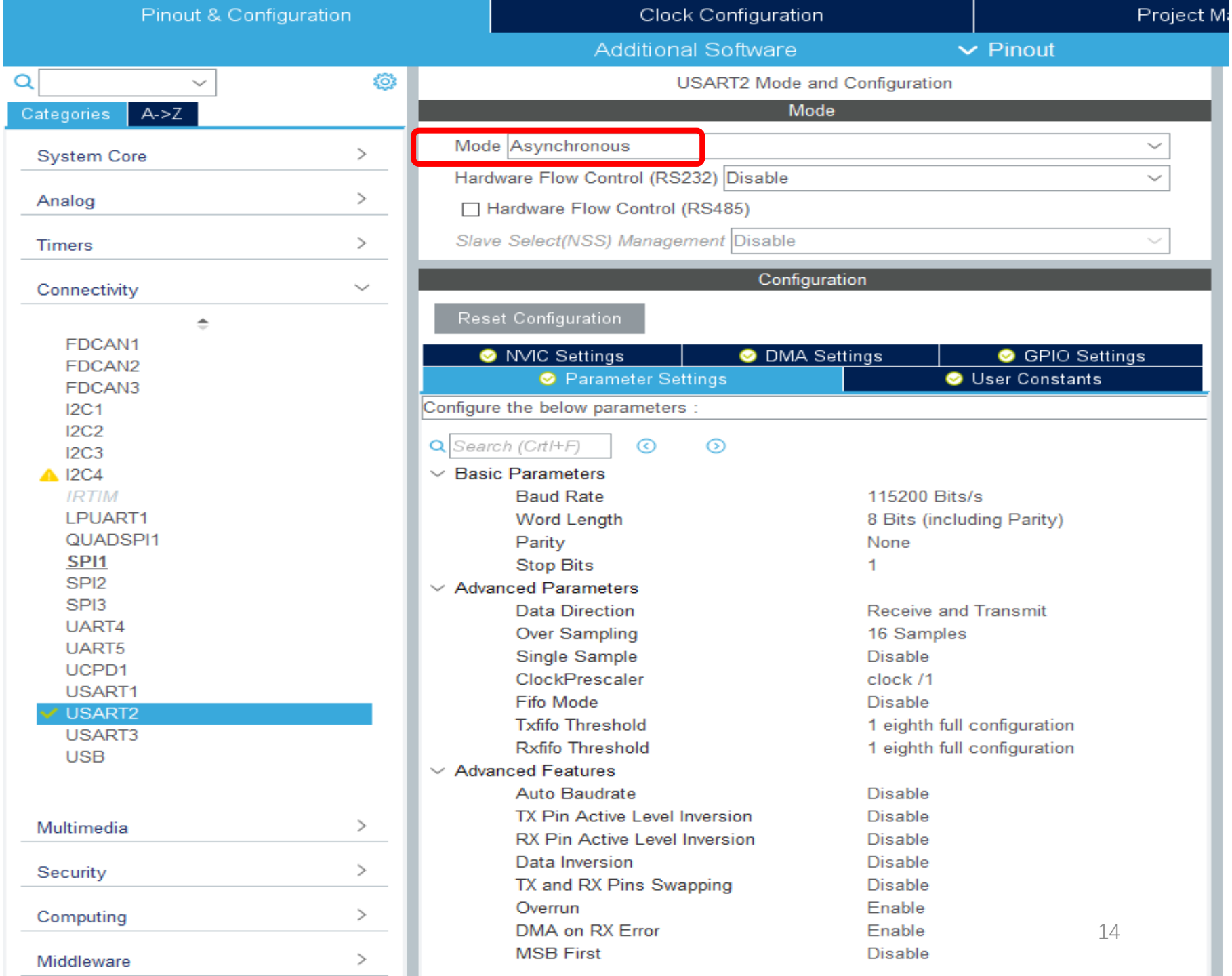
Pin ...	Signal ...	GPIO ...	GPIO mode	GPIO Pul...	Maximu...	Fa...	User La...	Modified
PA4	n/a	High	Output Push Pull	Pull-up	High	n/a	BUZ	☑
PA5	n/a	High	Output Push Pull	Pull-up	High	n/a	LED	☑
PC13	n/a	n/a	External Interrupt Mode with Rising e...	Pull-down	n/a	n/a	KEY	☑

配置GPIO外部中断

- 配置PC13的外部中断
- 优先级组（Priority Group）还是选择4bits for pre-emption priority 0 bits for subpriority。随后，将EXTI line[15:10] interrupts使能，并将它的抢占式优先级设为4，响应优先级为0。

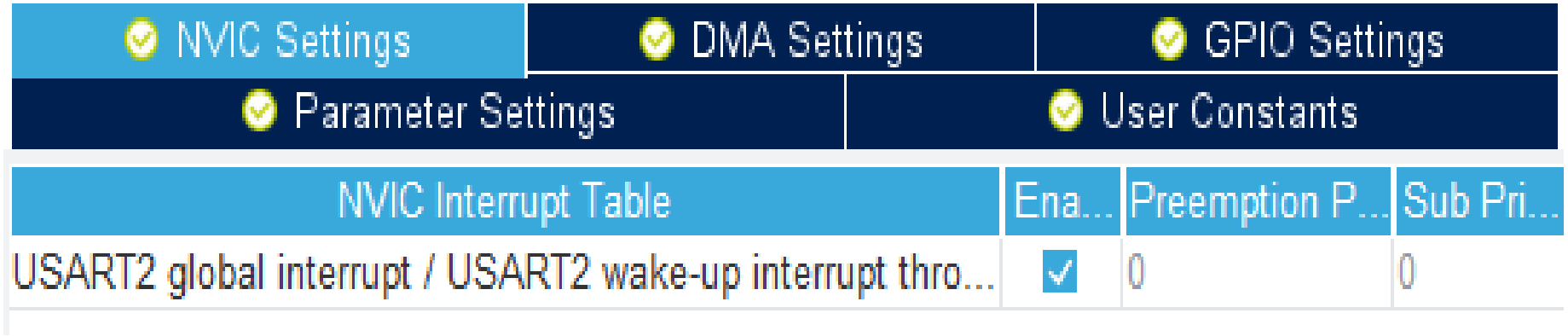
配置串口

- 展开Connectivity，选择其中的USART2
- 将Mode选择为异步Asynchronous



配置串口中断

■ 在NVIC设置中，将USART2的中断使能选上



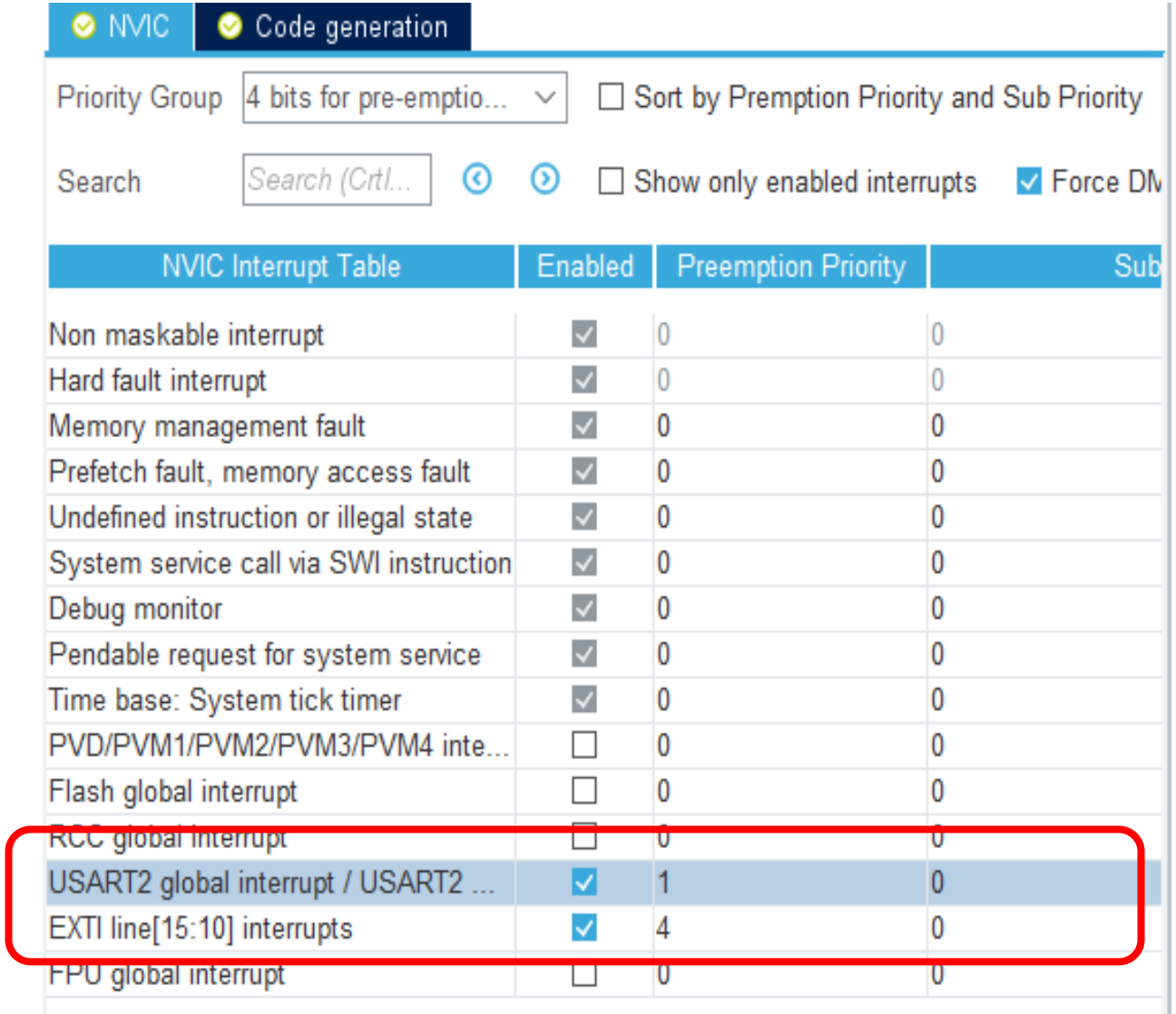
NVIC Interrupt Table	Ena...	Preemption P...	Sub Pri...
USART2 global interrupt / USART2 wake-up interrupt thro...	☑	0	0

■ 在GPIO Settings设置中，将配置PA2和PA3参数

Pin ...	Signal on Pin	GPIO o...	GPIO ...	GPIO P...	Maxim...	Fast M...	User L...	Modified
PA2	USART2_TX	n/a	Alterna...	Pull-up	High	n/a		☑
PA3	USART2_RX	n/a	Alterna...	Pull-up	High	n/a		☑

配置串口中断

■ System Core->NVIC中，将USART2的中断使能选上



其他硬件参数配置

■选择时钟源和Debug模式

✓ System Core->RC->将高速时钟（HSE）选择为Crystal/Ceramic Resonator

✓ SYS->Debug选择为Serial Wire

■配置系统时钟

✓ 在“Clock Configuration”中，将系统时钟（SYSCLK）配置为170Mhz

■ 保存硬件配置界面（*.ioc），启动代码生成

生成代码分析

- Core->Src，打开main.c
- MX_USART2_UART_Init()
  - ✓ 主要完成对USART2的模式和参数配置，如波特率、数据位、停止位等。
  - ✓ 没有对串口引脚的初始化语句
- HAL_UART_MspInit()是对串口硬件的初始化，配置引脚模式，设置中断优先级并使能中断

stm32g4xx_hal_msp.c　　Msp：MCU support package

生成代码分析

MX_USART2_UART_Init()　　main.c

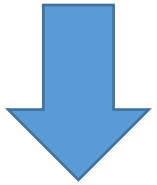
HAL_UART_Init()　　stm32g4xx_hal_uart.c

HAL_UART_MspInit()　　stm32g4xx_hal_msp.c

生成代码分析

PC13 EXTI中断

EXTI15_10_IRQHandler()



```
void EXTI15_10_IRQHandler(void)
{
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
}
```

stm32g4xx_hal_gpio.c

stm32g4xx_it.c

USART2中断

USART2_IRQHandler()

```
void USART2_IRQHandler(void)
{
  HAL_UART_IRQHandler(&huart2);
}
```

stm32g4xx_hal_uart.c

生成代码分析

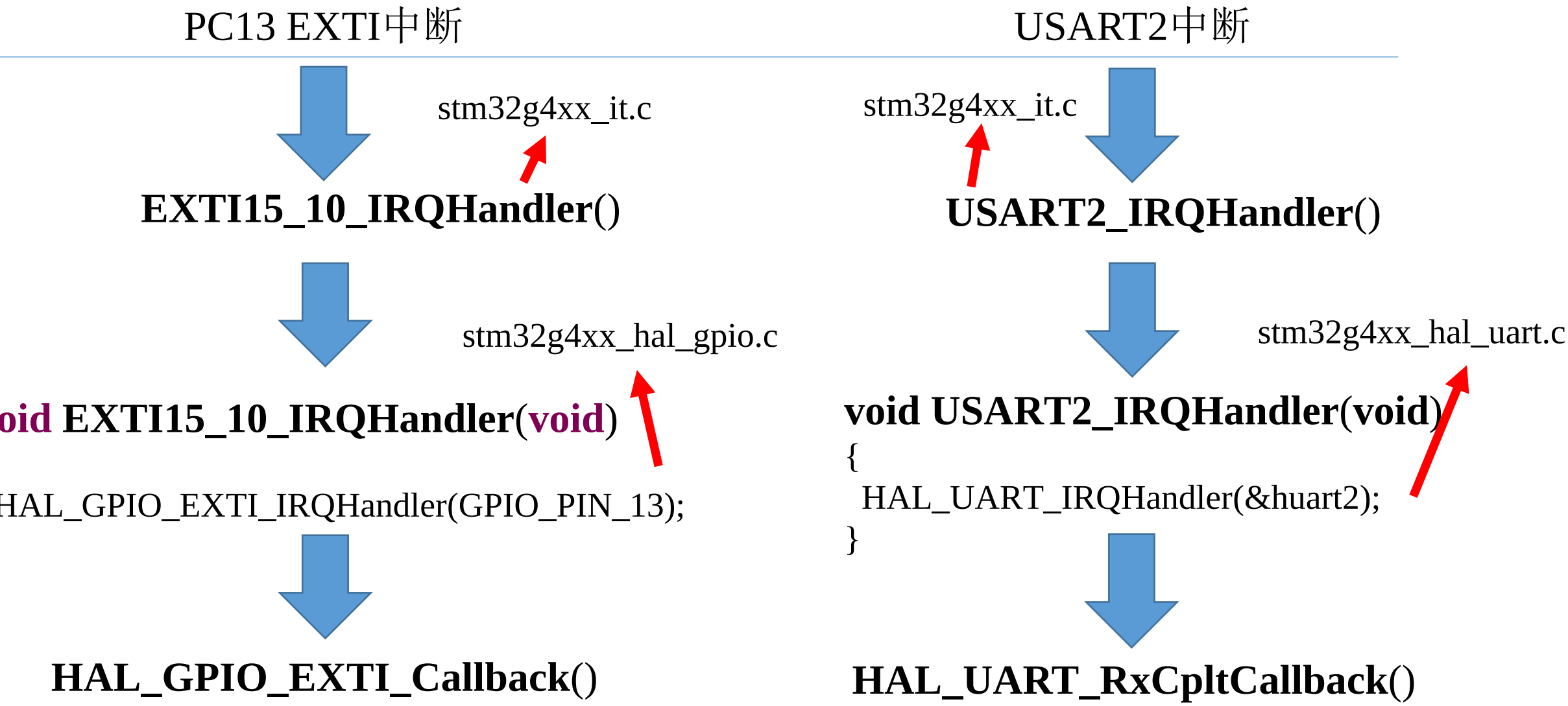
```
void HAL_UART_IRQHandler(UART_HandleTypeDef *huart)
{
  ……
  /* If no error occurs */
errorflags = (isrflags & (uint32_t)(USART_ISR_PE | USART_ISR_FE | USART_ISR_ORE | USART_ISR_NE));
  if (errorflags == 0U)
  {
    /* UART in mode Receiver --------*/
    if (((isrflags & USART_ISR_RXNE_RXFNE) != 0U)
        && (((cr1its & USART_CR1_RXNEIE_RXFNEIE) != 0U)
            || ((cr3its & USART_CR3_RXFTIE) != 0U)))
    {
      if (huart->RxISR != NULL)
      {
        huart->RxISR(huart);
      }
      return;
    }
  }
……
}
```

→ UART_RxISR_8BIT() → HAL_UART_RxCpltCallback()

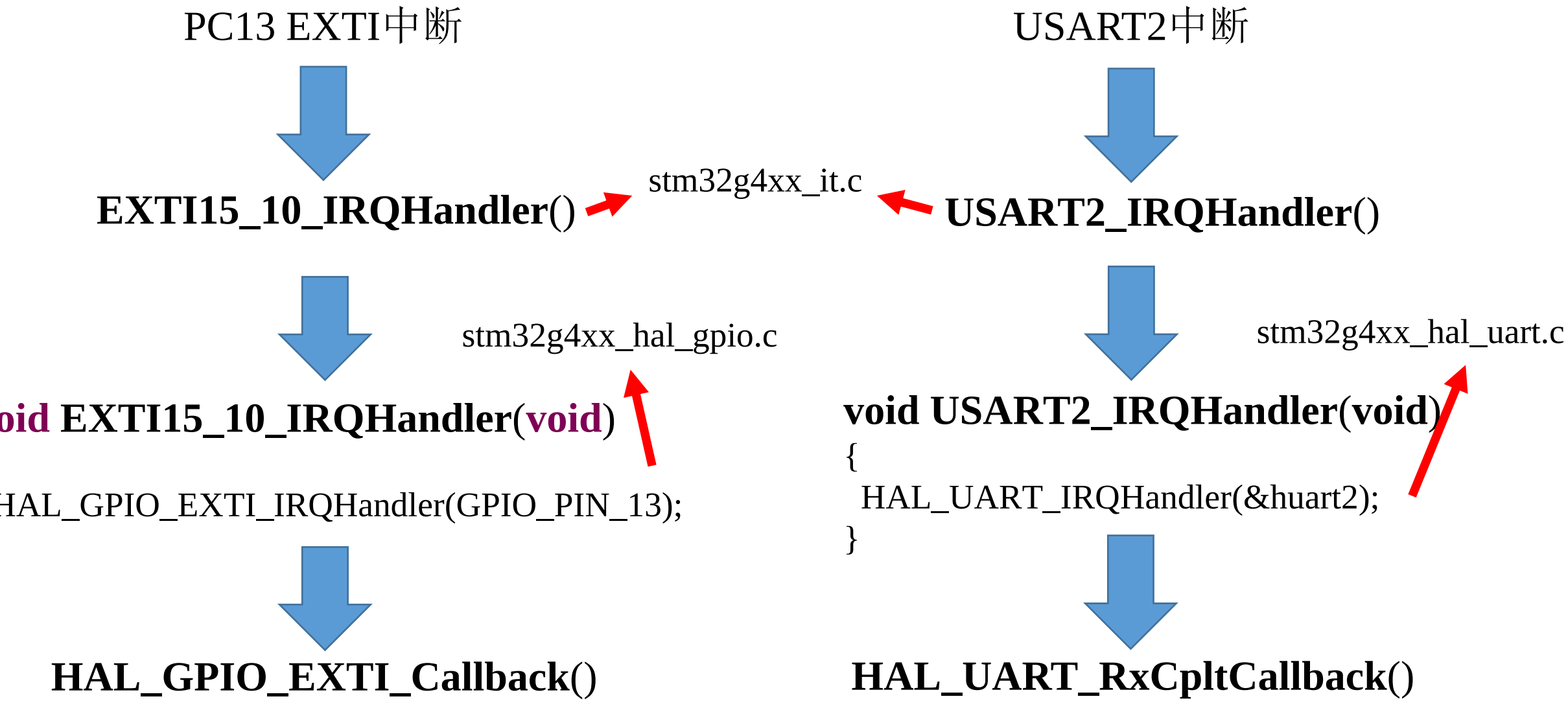
弱函数

stm32g4xx_hal_uart.c

生成代码分析


PC13 EXTI中断
stm32g4xx_it.c
EXTI15_10_IRQHandler()
stm32g4xx_hal_gpio.c
oid EXTI15_10_IRQHandler(void)
HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
HAL_GPIO_EXTI_Callback()
USART2中断
stm32g4xx_it.c
USART2_IRQHandler()
stm32g4xx_hal_uart.c
void USART2_IRQHandler(void)
{
HAL_UART_IRQHandler(&huart2);
}
HAL_UART_RxCpltCallback()

生成代码分析


PC13 EXTI中断
USART2中断
stm32g4xx_it.c
EXTI15_10_IRQHandler()
USART2_IRQHandler()
stm32g4xx_hal_gpio.c
stm32g4xx_hal_uart.c
oid EXTI15_10_IRQHandler(void)
HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
void USART2_IRQHandler(void)
{
HAL_UART_IRQHandler(&huart2);
}
HAL_GPIO_EXTI_Callback()
HAL_UART_RxCpltCallback()

启动串口接收中断

■使用中断之前，还要用到一个函数HAL_UART_Receive_IT()

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

- ✓ 该函数是给将要接收的数据，定义一个缓冲区pData
- ✓ 指定接收数据的长度为Size（也就是要接收的字节数）
- ✓ Size决定调用回调函数的频率。如果Size大于1，则不会每次中断都调用回调函数，而是到Size次之后，才会调用一次回调函数
- ✓ 此外，这个函数还有开启中断接收的功能

启动串口接收中断

■使用中断之前，还要用到一个函数HAL_UART_Receive_IT()

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

✓ 在main函数的初始化代码中，调用一次HAL_UART_Receive_IT()函数。这样就可以确保开启接收中断

✓ 在执行一次回调函数时，接收中断会关闭，所以还需要再次开启接收中断。这个再次开启中断的动作，可以在回调函数中通过调用HAL_UART_Receive_IT()函数来实现（也可以放到中断服务函数中调用）

启动串口接收中断

完成上面的硬件配置，并自动生成代码后，

接下来下的工作就是在main.c中的初始化部分调用

HAL_UART_Receive_IT()函数，设置参数及开启接收中断，

并写回调函数HAL_UART_RxCpltCallback()，以对接收的数据进行处理。

自动生成的main函数

```
#include "main.h"
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
```

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART2_UART_Init();
   while (1)
   {
   }
}
```

MX_USART2_UART_Init()

■ 在main.c中，首先定义了一个全局变量huart2，类型为UART_HandleTypeDef；huart2是一个结构体变量，通常也称为串口句柄。

```
static void MX_USART2_UART_Init(void)
{
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    ……
   if (HAL_UART_Init(&huart2) != HAL_OK)
   ……
}
```

HAL_UART_Receive_IT()

- 需要在主程序的初始化代码中调用HAL_UART_Receive_IT()函数

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

- 放到调用MX_USART2_UART_Init()函数之后的注释对中
- 第二个参数是放置接收到数据的缓冲区，可以定义一个长度为RXBUFFERSIZE的数组RxBuffer[RXBUFFERSIZE]，定义为全局变量
- 第三个参数用于指定接收数据的长度，这个数据长度可以与接收缓冲区的长度相同，即RXBUFFERSIZE。
- 将对HAL_UART_Receive_IT()函数的调用放置到 MX_USART2_UART_Init()语句之后的两个注释对/* USER CODE BEGIN 2 */与/* USER CODE END 2 */之间

修改后的main函数

```
#include "main.h"
UART_HandleTypeDef huart2;
/* USER CODE BEGIN PV */
uint8_t RxBuffer[RXBUFFERSIZE];
/* USER CODE END PV */
/* Private function prototypes */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
```

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  HAL_UART_Receive_IT(&huart2, (uint8_t *)RxBuffer, RXBUFFERSIZE);
  /* USER CODE END 2 */
  while (1)
  {
  }
}
```

单选题 1分

设置

串口模块的使用与GPIO不同之处是什么?

A 需要在CubeMX中配置

B 用户要重定义回调函数

C 需要编程者调用初始化库函数

D 无不同，过程类似

提交

定义RXBUFFERSIZE

- 对RXBUFFERSIZE的定义可以放到main.h头文件中，用define宏定义声明（也需放置到两个注释对之间）

```
/* USER CODE BEGIN Private defines */
#define RXBUFFERSIZE  1 //接收缓冲区的长度
/* USER CODE END Private defines */
```

串口接收中断执行过程

- 串口有数据送来，会执行中断服务函数USART2_IRQHandler()；该函数又会调用函数HAL_UART_IRQHandler()；调用“一定次数”的HAL_UART_IRQHandler()后，就会自动执行回调函数HAL_UART_RxCpltCallback()。
- 这里的“一定次数”是由HAL_UART_Receive_IT()函数的第3个参数决定的，也就是前面在主程序中定义的常量RXBUFFERSIZE。
- 如RXBUFFERSIZE定义为1，则串口收到1个字节的数据后，会调用一次回调函数HAL_UART_RxCpltCallback()。调用回调函数之时，1个字节的数据已经放到了接收缓冲区中，也就是放到数组RxBuffer中。

重定义回调函数

- 重定义GPIO外部中断及串口中断接收的回调函数

```
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        HAL_UART_Receive_IT(&huart2, (uint8_t *)RxBuffer, RXBUFFERSIZE);
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        HAL_GPIO_WritePin(BUZ_GPIO_Port, BUZ_Pin, GPIO_PIN_RESET);
        HAL_Delay(100);//延时
        HAL_GPIO_WritePin(BUZ_GPIO_Port, BUZ_Pin, GPIO_PIN_SET);
}
/* USER CODE END 4 */
```

修改main函数中的while循环

- 根据串口送来的数据，控制发光二极管的亮灭；当接收到的数据为0x10（十六进制）时，点亮LD2亮，不是0x10时，熄灭LD2

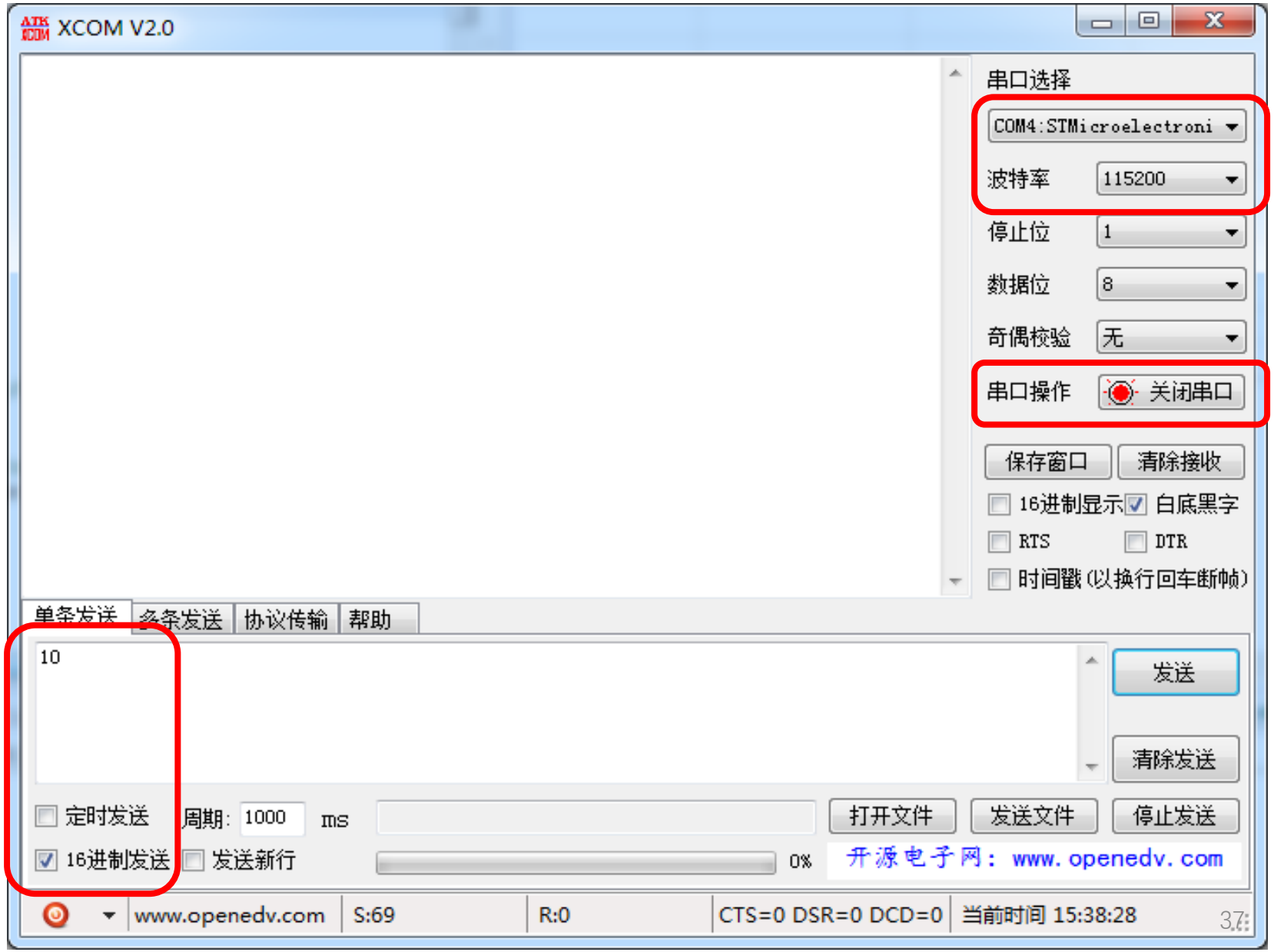
```
/* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE BEGIN 3 */
        if (RxBuffer[0]==0x10)
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
        else
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
   }
  /* USER CODE END 3 */
```

编译、下载与运行

- 编译
- 在下载之前，先进入主菜单Run->Debug Configurations，在弹出的“创建、管理和运行配置”（Create, manage, and configurations）界面中，用鼠标右键点击左侧栏目中的“STM32 Cortex-M C/C++ Application”，新建一个新的配置（New Configuration），.....
- 配置完毕后，点击配置界面右下角的“Debug”，即可以自动完成下载
- 下载完成后，点击主菜单上的运行（Resume）按钮，就可以运行程序

串口助手

- 打开串口助手，在串口选择栏选择NULEO-G431RB板在计算机上虚拟的串口号，选择波特率等参数
- 在串口助手的发送栏，以十六进制数的形式发送数据到MCU中



练习5：串行通信

任务5.1、用中断方式实现串口数据接收

（1）通过发送命令控制LD2的亮灭和蜂鸣器响。

（2）通过串口发送数据，控制LD2的闪烁次数；譬如发送的数为8，则LD2闪烁8次后，停止闪烁（闪烁频率为1Hz）。

串口数据发送

■ 实现串口发送的库函数

HAL_UART_Transmit(*huart, pData, Size, Timeout)

HAL_UART_Transmit_IT(*huart, pData, Size)

使用HAL_UART_Transmit()发送数据

- 如收到0x10后，MCU送出一串字符：Everything is OK；收到的数据不是0x10时，则MCU送出字符串：Received Error Data。
- 首先在main函数前定义两个放置这些字符的数组，可以与前面定义的RxBuffer放到相同的注释对中：

```
/* USER CODE BEGIN PV */
uint8_t CommOkMessage[] = "Everything is OK\r\n";
uint8_t CommErrMessage[] = "Received Error Data\r\n";
uint8_t CommFlag = 0;
uint8_t RxBuffer[RXBUFFERSIZE] = {0};
/* USER CODE END PV */
```

\r和\n：转义字符，分别指回车和换行

- 发送CommOkMessage字符串可以用如下语句：

```
HAL_UART_Transmit(&huart2,CommOkMessage,19,1000);
```

修改while循环

■ 当接收到的是0x10时，发送字符串CommOkMessage，不是0x10时，发送字符串CommErrMessage。为了避免MCU一直往外送数据，利用标志变量CommFlag进行控制。

```
while (1)
 {
  /* USER CODE BEGIN 3 */
        if ((RxBuffer[0] == 0x10)&&( CommFlag == 1))  {
              CommFlag = 0;
              HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
              HAL_UART_Transmit(&huart2,CommOkMessage,19,1000);
        }
        else if ((RxBuffer[0] != 0x10)&&( CommFlag == 1))  {
              CommFlag = 0;
              HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
              HAL_UART_Transmit(&huart2,CommErrMessage,22,1000);
        }
 }
 /* USER CODE END 3 */
```

修改回调函数

- 在回调函数中置位CommFlag

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        CommFlag = 1;
        HAL_UART_Receive_IT(&huart2, (uint8_t *)RxBuffer, RXBUFFERSIZE);
}
```

使用HAL_UART_Transmit_IT()函数实现数据发送

- 直接将while(1)循环中的HAL_UART_Transmit()改成HAL_UART_Transmit_IT()，不过需要把超时时间参数去掉。
- 发送字符串CommOkMessage的函数就可以改为

```
HAL_UART_Transmit_IT(&huart2,CommOkMessage,19);
```

练习5：串行通信

任务5.2、串口数据的发送与接收

（1）实现上述串口数据的发送与接收；

（2）修改任务5.1(2)，回送LD2闪烁的次数；譬如，要求闪烁8次，当第一次闪烁时，发送第一次闪烁的提示（具体内容自定义），第二次时，发送第2次闪烁的提示….

修改回调函数的调用模式

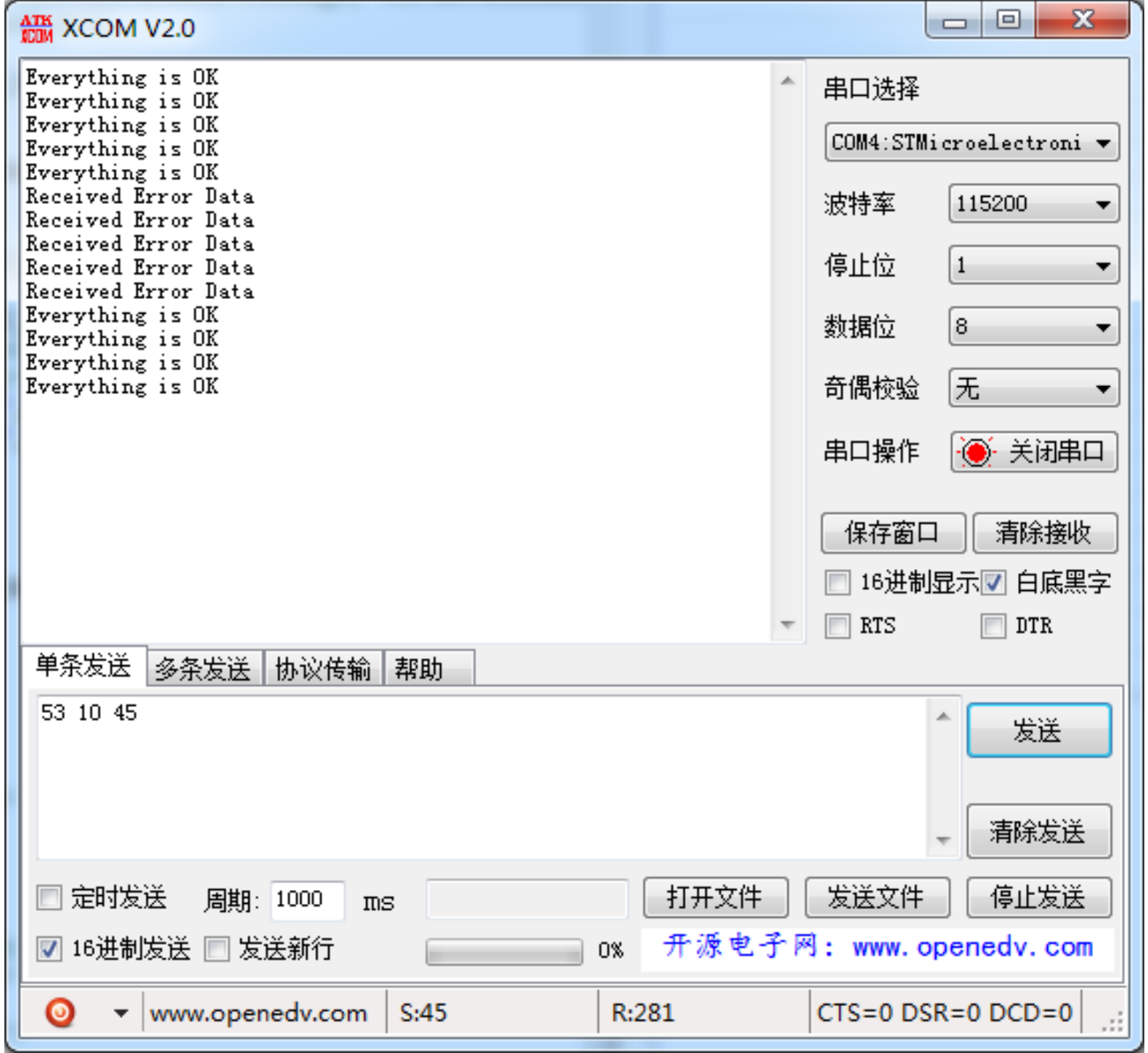
- 在上面的实现中，串口每接收到一个字节的数据，就调用一次回调函数。
- 可以通过修改参数，在接收了一定长度的数据之后，再调用回调函数。
- 下面尝试在串口接收到3个字节数据后，再调用一次回调函数。
- 需要将HAL_UART_Receive_IT()函数中的Size参数设置为3，也就是把在main.h中声明的RXBUFFERSIZE设置为3。可以直接在main.h中修改。

数据帧

- 假如串口助手向MCU一次发送三个字节的数据，其中第2个字节的数据是有意义的，前后两个字节是编码，作为帧头和帧尾（可以任意设置，此处分别用字符“S”和“E”来表示，它们在ASC II码表中对应的16进制数分别为0x53和0x45）
- 当MCU通过串口接收到3个字节的数后，可以首先判断帧头和帧尾是否正确，如果正确，再进行下一步的处理。

修改while循环中代码

```
while (1)
{
  /* USER CODE BEGIN 3 */
          if ((RxBuffer[0] == 0x53)&&(RxBuffer[2] == 0x45)&&(CommFlag == 1))
           {
               CommFlag = 0;
               if (RxBuffer[1] == 0x10)
               {
                    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
                    HAL_UART_Transmit_IT(&huart2,CommOkMessage,19);
               }
               else
              {
                   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
                   HAL_UART_Transmit_IT(&huart2,CommErrMessage,22);
              }
           }
}
/* USER CODE END 3 */
```


XCOM V2.0
Everything is OK
Everything is OK
Everything is OK
Everything is OK
Everything is OK
Received Error Data
Received Error Data
Received Error Data
Received Error Data
Received Error Data
Everything is OK
Everything is OK
Everything is OK
Everything is OK
串口选择
COM4:STMicroelectroni
波特率
115200
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
16进制显示
白底黑字
RTS
DTR
单条发送
多条发送
协议传输
帮助
53 10 45
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网：www.openedv.com
www.openedv.com
S:45
R:281
CTS=0 DSR=0 DCD=0

练习5：串行通信

任务5.3、多字节串口数据的发送与接收

（1）实现上述多字节串口数据的发送与接收；

（2）修改任务5.2(2)，通过串口助手发送数据，控制LD2的闪烁次数和闪烁频率，闪烁中将当前闪烁次数和频率信息送出来。

提示：如果LED的闪烁频率过快，有可能分辨不出。测试时可以把频率放慢一些。

串口相关库函数

1、串口中断服务函数

HAL_UART_IRQHandler(UART_HandleTypeDef *huart)

2、串口接收回调函数

HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)

3、串口接收中断配置函数

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

4、串口发送函数

非中断发送：HAL_UART_Transmit(*huart, pData, Size, Timeout)

中断发送：HAL_UART_Transmit_IT(*huart, pData, Size)

printf函数

- 在C语言中，一个非常有用的库函数printf()，其声明是在头文件 stdio.h 中，作用是根据指定的格式输出字符串。
- 在C的中，用printf函数，是将输出送到位于界面中某处的控制台上；在MCU中，可以借用这种思想，利用printf函数将信息送到MCU的外设上。比较常用的就是与串口结合起来。
- 在stm32CubeIDE中，如何使用printf函数，实现从串口发送数据呢？

包含头文件

- 要想在代码中使用printf函数，需要在main函数中包含头文件stdio.h。可以将它放到main.c最前面的一个注释对中，如：

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */
```

修改发送接收字符函数

- printf函数需要fputc()函数或putchar()函数的支持，还需要在程序中给出它的实现代码。使用fputc()还是putchar()，取决于编译器的类型。
- 由于STM32CubeIDE使用的是GNU C/C++编译器，所以printf()使用的底层函数是putchar()函数，具体来说就是__io_putchar(int ch)。可在main.c中，给出具体的实现：

```
int __io_putchar(int ch)
{
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
}
```

- 可以放到注释对/* USER CODE BEGIN 4 */ 和/* USER CODE END 4 */之间

使用printf发送数据

■ 实现功能：

✓ 程序执行后，MCU通过串口发送“Please Enter Data:”这串提示字符；

✓ MCU通过中断接收到3个字节数据后，会将它们存储到RxBuffer数组中；

✓ 利用按键B1来查看送来的数据，即在按下按键后，通过串口送出RxBuffer数组中接收到的数据。

使用printf发送数据

- 首先在while(1)循环前，用printf()通过串口送出提示输入数据的信息，譬如：

```
/* USER CODE BEGIN WHILE */
  printf("Please Enter Data:\r\n");
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
          if ((RxBuffer[0] == 0x53)&&(RxBuffer[2] == 0x45)&&(CommFlag == 1))
          ……
  }
  /* USER CODE END 3 */
```

使用printf发送数据

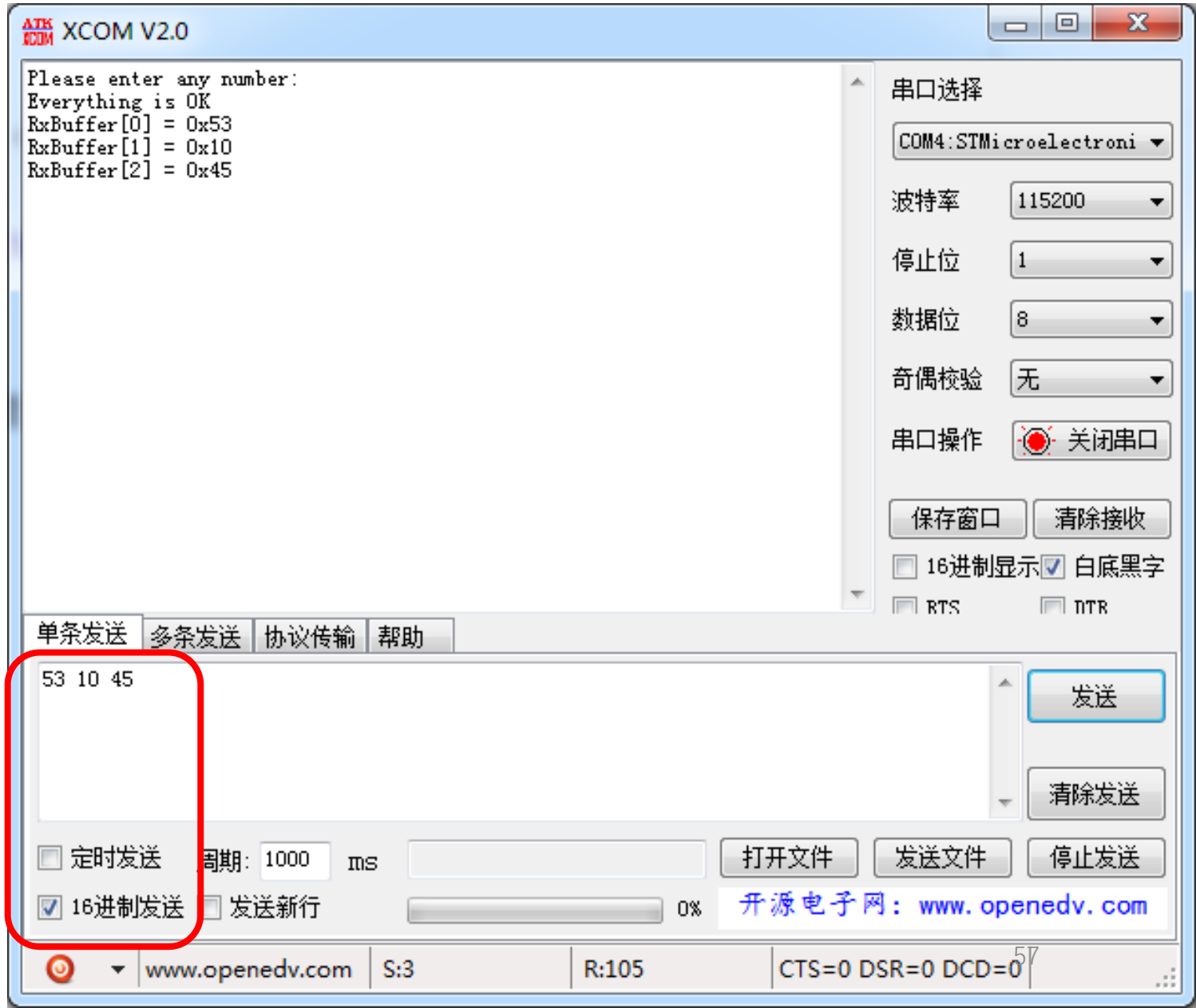
- 随后，修改按键中断处理的回调函数HAL_GPIO_EXTI_Callback()，加入串口发送数据的代码：

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        HAL_GPIO_WritePin(BUZ_GPIO_Port, BUZ_Pin, GPIO_PIN_RESET);
        HAL_Delay(100);
        HAL_GPIO_WritePin(BUZ_GPIO_Port, BUZ_Pin, GPIO_PIN_SET);
        for(uint8_t i = 0; i < RXBUFFERSIZE; i++)
        {
                printf("RxBuffer[%d] = 0x%02x\r\n", i, RxBuffer[i]);
        }
}
```

- 在printf()语句中，%d是按整型数据输出，%x是按16进制格式输出，%02x中的02，表示位宽为2，不够的话前边补0。

使用printf发送数据

- 运行程序后，在串口助手的接收区域会看到提示输入数据的字符“Please Enter Data:”。
- 通过串口助手按16进制格式送出数据53 10 45 后，串口助手会接收到字符“Everything is OK”，表示数据已经正确收到。
- 按NUCLEO-G431RB板上的B1按键，在串口助手上就会显示所接收到的数据。



练习5：串行通信

任务5.4、使用printf发送数据的方式实现任务5.3(2)。

任务5.5、问题描述：

一个班级有120名同学，刚完成了数学测验，成绩出来后，老师想根据成绩排一下名次。请用单片机实现。

成绩输入、排名输出均通过串口。为简化过程，假设仅有**20**名同学，成绩输入通过串口助手采用交互式方式，并且可以判断输入成绩是否有效（0~100有效），如果输入无效，提示重新输入。

提交网络学堂：每个子任务的工程文件（压缩），代码有简单注释

谢　谢!