不讲理论的STM32教程

进阶部分

第八章：串口通信

1.串口的介绍

理论部分：

从本章开始，最近3章应该都是我们的通信部分，我们会依次介绍较为常用的串口通信（USART），IIC通信，SPI通信

接下来，为大家介绍我们的串口通信（USART），相信我们在51的学习中，肯定学过UART，也就是串口通信，全称叫做：通用异步收发传输器（Universal Asynchronous Receiver/Transmitter），是我们最常用的串行通信方式，具体的解释阿熊也不多说了，小伙伴们不懂得可以回头看看我们51的教程

USART是什么？为什么不是UART，为什么多了一个S，S是什么意思？

这里给大家找了一些资料

USART:(Universal Synchronous/Asynchronous Receiver/Transmitter)通用同步/异步串行接收/发送器

对比一下，可以发现S代表的是Synchronous，同步的，也就是说USART就是在UART的基础上升级了同步通信的功能

这里百科也有解释到，在UART上追加同步方式的序列信号变换电路的产品，被称为USART(Universal Synchronous Asynchronous Receiver Transmitter)

USART，他作为UART的升级版，不光支持UART，还支持一些其他的通信模式，这里简单列举一下

USART模式
异步模式
硬件流控制(CTS/RTS)
多缓存通信(DMA)
多处理器通信
同步通信
智能卡（Smartcard）
半双工（单线模式）
IrDA(红外线)
LIN(域互连网络)

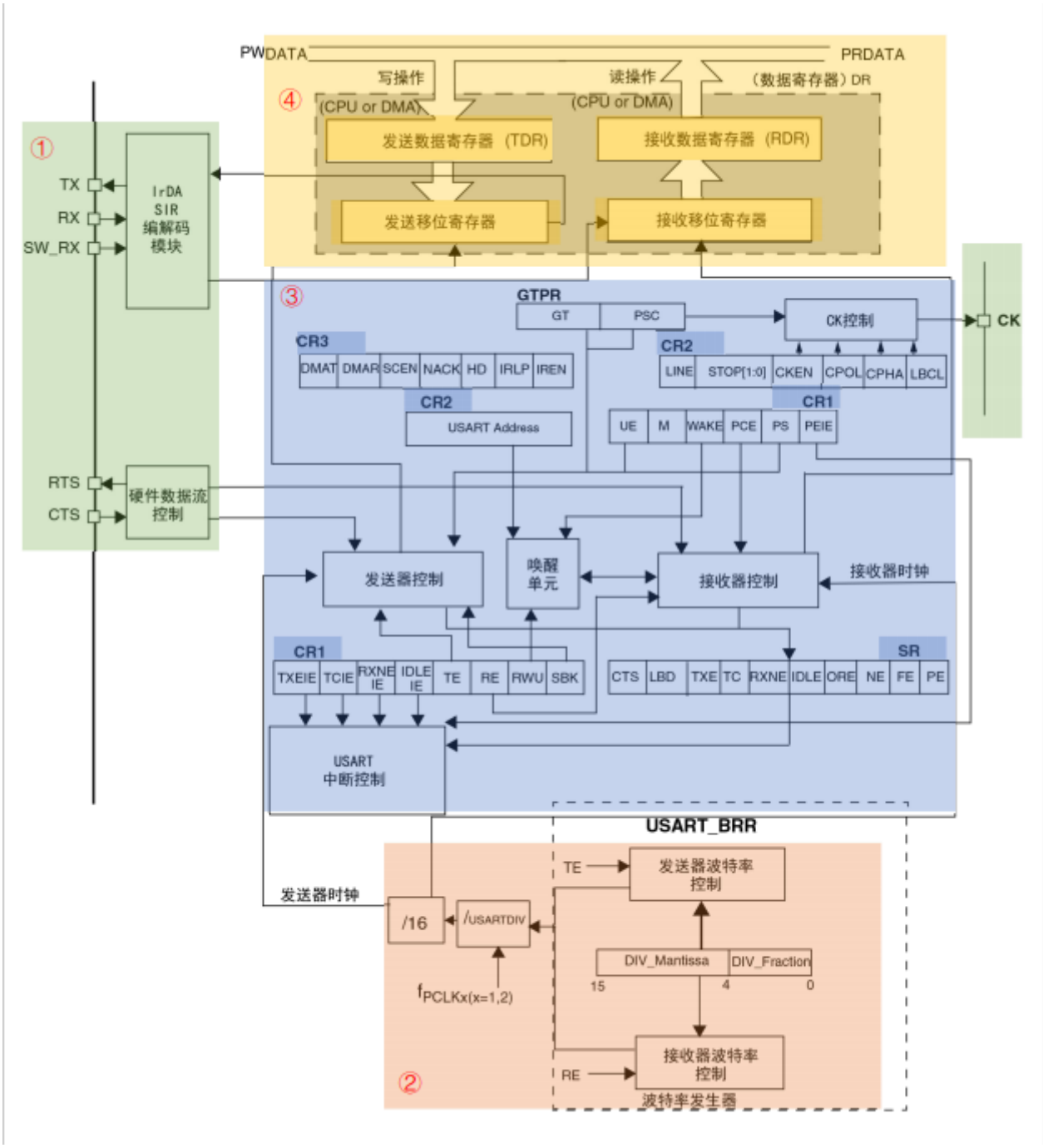
串口在嵌入式中经常使用，一般使用UART就足够了，常见的用途如下：

1/作为调试口，打印程序运行的状态信息

2.连接串口接口的模块（比如GPS模块），传输数据

3.通过电平转换芯片变为RS232/RS485电平，连接工控设备

所以，我们目前支只对我们的UART进行讲解，其他的小伙伴们可以自行去了解哦，如果后期教程涉及到，阿熊也会去讲解哦！



这是他的内部构造，看起来就头疼，所以我们只讲使用不讲理论

首先我们的STMF103C8T6芯片上一共有三组USART(这里只讲UART的TX和RX)分别是：

USART1：

USART2：

USART3：

另外我们既然要通信，波特率，字节长度，停止位，校验位之类的参数肯定少不了，这里关于这些内容，阿熊就不过多介绍，后面项目配置的时候，会给大家细说

常用函数：

我们使用串口一般是接收消息，或者发送消息，而接收消息，一般情况下都是不确定时去接收消息的，所以肯定口不能在我们的主函数里进行判断，而是应该使用我们的中断去接收，这里列举几个较为常用的函数供大家参考

HAL_UART_Transmit(&husart, (uint8_t*)&ch, 1, 10);//串口发送数据（选择串口的地址，发送的字符串，发送字节数，等待时间）

HAL_UART_Receive(&husart, (uint8_t*)&ch, 1, 10); //串口接收数据（选择串口的地址，接收的字符串，接收字节数，等待时间）

前面3个参数相信都很容易看懂，这里重点讲一下第四个，我们的函数是需要执行时间的，这里就是设置一个等待时间（单位是毫秒）当在这个时间过后都还没发送完成，就会停止发送，并且返回发送失败的信号，作为学习一般不太存在发送或者接收失败的 情况，所以通常都是直接设置为最大0xFFFF

前面两个都是阻塞函数，也就是会消耗一定的主函数时间，导致我们的主函数可能会不是那么精确，所以这里就要讲到我们的另外两个函数，这样的话，我们就可以在中断中执行我们的指令而不会影响我们的主函数

HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);//中断发送数据（选择串口的地址，发送的字符串，发送字节数）

HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);//中断接收数据（选择串口的地址，发送的字符串，发送字节数）

可以看到我们的这两个函数都是没有我们第四个参数的，其他的和前面的两个函数没有区别，只是我们这个函数不再是阻塞函数了，而是使用了了我们的中断系统的串口中断，不过一般情况HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);使用的很少，因为一般情况发送不太需要在中断中去发送，反而是接收大多数都是在中断中使用，

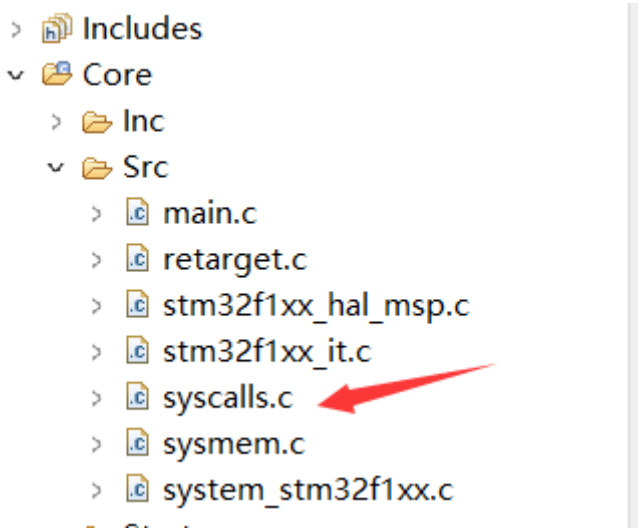
要注意的是我们的后两个函数第一次使用时，是代表开启我们的接收/发送中断，这样才会开启接收中断，并且我们的HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);在接收到数据以后他就会触发我们的对应中断以及对应的中断函数，而且标记我们的接收标志位，这时候我们需要手动给他恢复标志位，也就是再次使用此函数

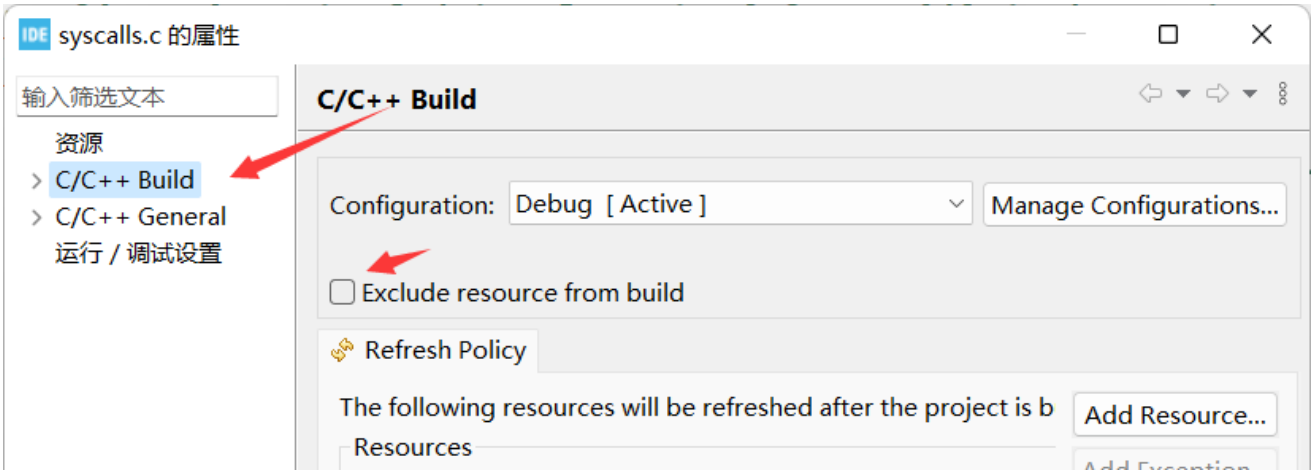
理论就是这样接下来我们讲一下次我们怎么让我们的程序可以printf();以及scanf();发送和接收数据

printf()和scanf()：

第一步：

我们需要禁用一下我们的





然后点击应用就完成了我们的第一步

第二部：

分别创建retarget.c以及retarget.h,我们将我们的代码复制粘贴进去

retarget.c

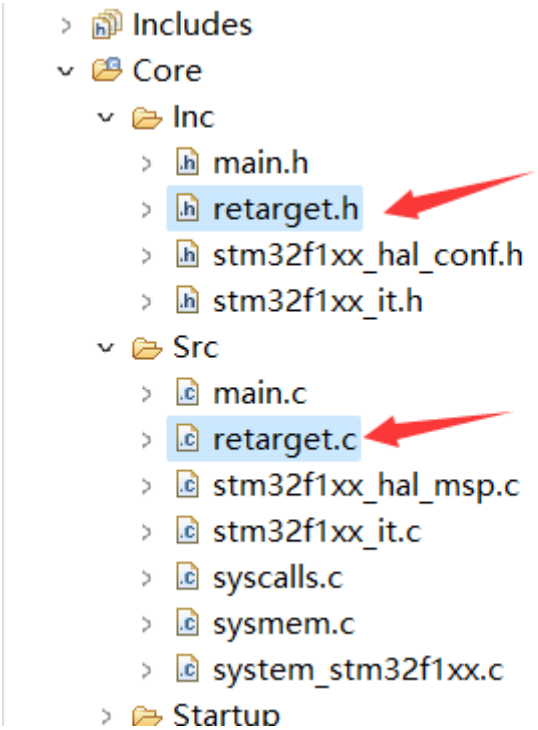
```
#include <_ansi.h>
#include <_syslist.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/times.h>
#include <limits.h>
#include <signal.h>
#include <../Inc/retarget.h>
#include <stdint.h>
#include <stdio.h>
#if !defined(OS_USE_SEMIHOSTING)
#define STDIN_FILENO  0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2

UART_HandleTypeDef *gHuart;
```

```
void RetargetInit(UART_HandleTypeDef *huart)  {
  gHuart = huart;
  /* Disable I/O buffering for STDOUT  stream, so that
   * chars are sent out as soon as they are  printed. */
  setvbuf(stdout, NULL, _IONBF, 0);
}
int _isatty(int fd) {
  if (fd >= STDIN_FILENO && fd <=  STDERR_FILENO)
    return 1;
  errno = EBADF;
  return 0;
}
int _write(int fd, char* ptr, int len) {
  HAL_StatusTypeDef hstatus;
  if (fd == STDOUT_FILENO || fd ==  STDERR_FILENO) {
    hstatus = HAL_UART_Transmit(gHuart,  (uint8_t *) ptr, len,
HAL_MAX_DELAY);
    if (hstatus == HAL_OK)
      return len;
    else
      return EIO;
  }
  errno = EBADF;
  return -1;
}
int _close(int fd) {
  if (fd >= STDIN_FILENO && fd <=  STDERR_FILENO)
    return 0;
  errno = EBADF;
  return -1;
}
int _lseek(int fd, int ptr, int dir) {
  (void) fd;
  (void) ptr;
  (void) dir;
  errno = EBADF;
  return -1;
}
int _read(int fd, char* ptr, int len) {
  HAL_StatusTypeDef hstatus;
  if (fd == STDIN_FILENO) {
    hstatus = HAL_UART_Receive(gHuart,  (uint8_t *) ptr, 1,
HAL_MAX_DELAY);
    if (hstatus == HAL_OK)
      return 1;
    else
      return EIO;
  }
  errno = EBADF;
```

```
  return -1;
}
int _fstat(int fd, struct stat* st) {
  if (fd >= STDIN_FILENO && fd <=  STDERR_FILENO) {
    st->st_mode = S_IFCHR;
    return 0;
  }
  errno = EBADF;
  return 0;
}

#endif //#if !defined(OS_USE_SEMIHOSTING)
```

retarget.h

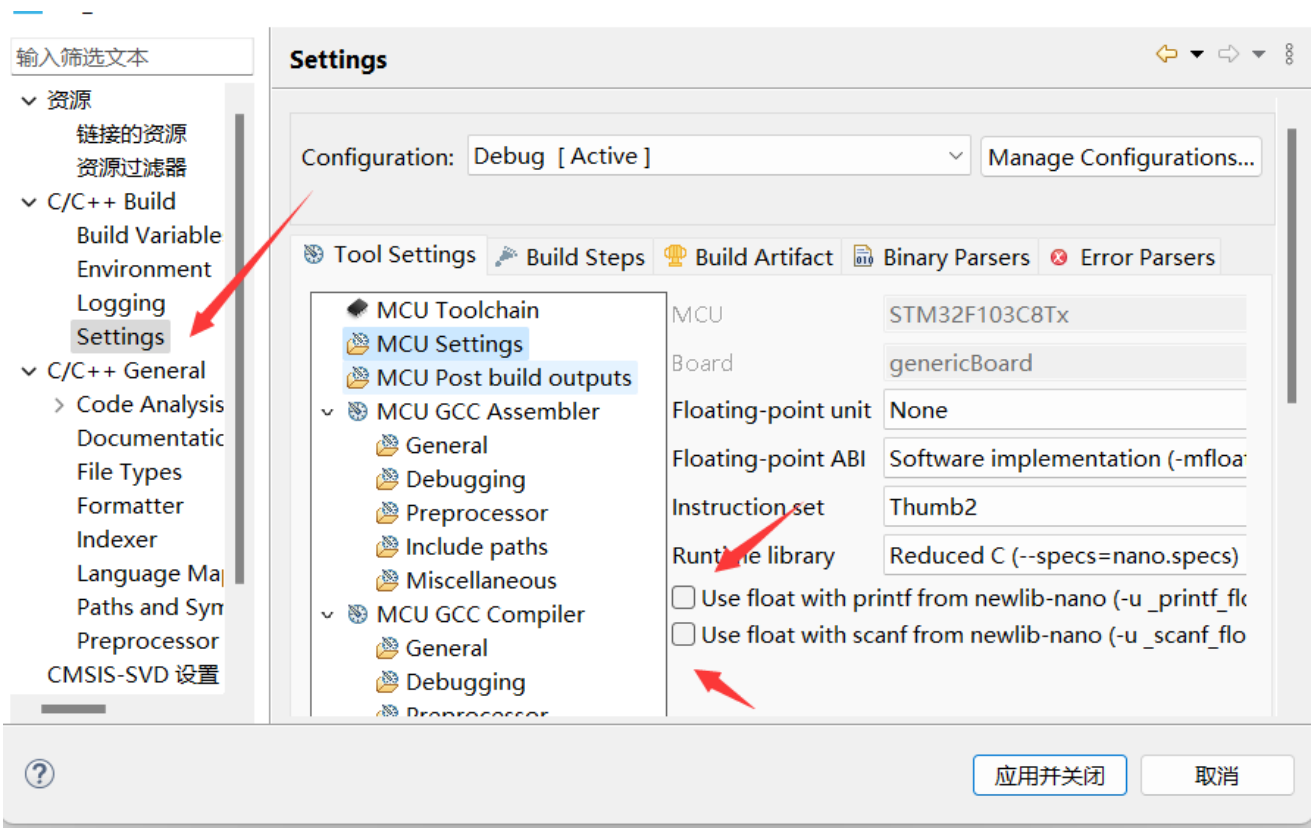
```
#ifndef INC_RETARGET_H_
#define INC_RETARGET_H_

#include "stm32f1xx_hal.h"
#include "stdio.h"//用于printf函数串口重映射
#include <sys/stat.h>

void RetargetInit(UART_HandleTypeDef  *huart);

int _isatty(int fd);
int _write(int fd, char* ptr, int len);
int _close(int fd);
int _lseek(int fd, int ptr, int dir);
int _read(int fd, char* ptr, int len);
int _fstat(int fd, struct stat* st);

#endif /* INC_RETARGET_H_ */
```



项目结构就是这样，具体原理我们暂时不解释，这个代码也是官方提供的，所以我们会移植就好了，学会如何使用就可以了

好了，移植完成

最后还有一部，为了我们的printf();以及scanf();可以使用浮点型，我们需要在项目属性里勾选一下两个框框，可以参考下图



好了，以上就完成了全部的移植，我们已经可以使用我们的printf和我们的scanf();了，接下来就是我们的展示了

2.串口的使用：

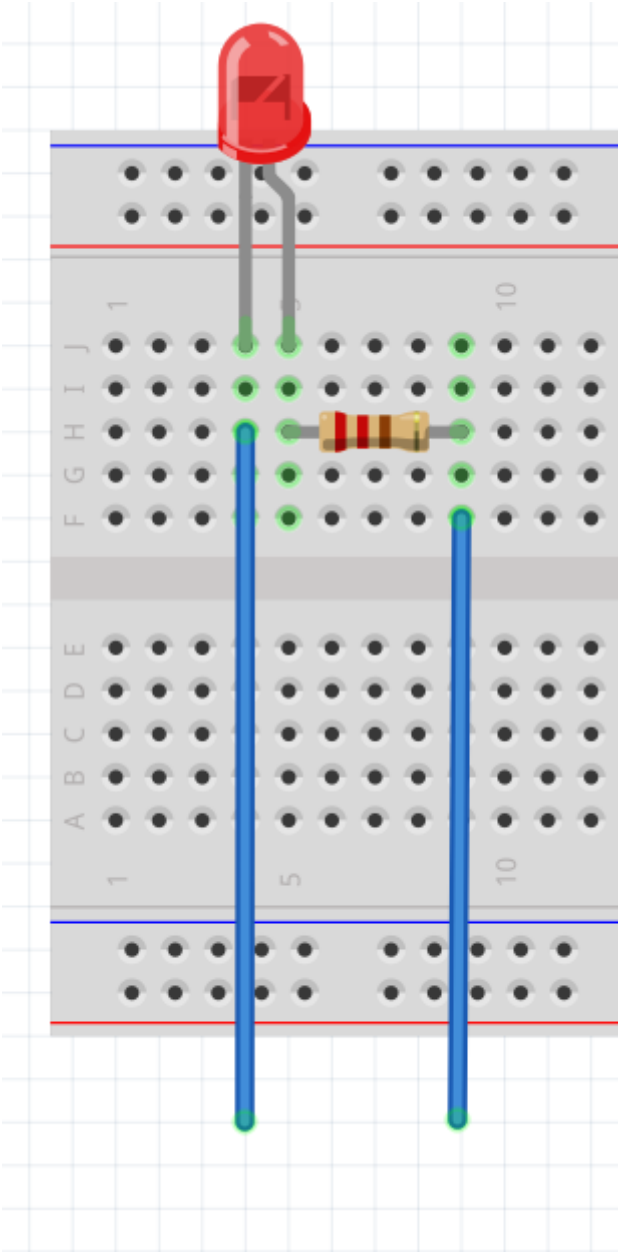
项目：

串口控制小灯泡亮灭

材料：

面包板，LED小灯泡一个，1K电阻一个，导线若干

原理图：



思路分析：

我们本次学习的是UART串口通信，主要带着大家熟悉一下我们的串口发送和我们的串口中断接收用到的函数是我们的HAL_UART_Transmit();//串口发送数据（选择串口的地址，发送的字符串，发送字节数，等待时间）和HAL_UART_Receive_IT();//中断接收数据（选择串口的地址，发送的字符串，发送字节数）

这里我们要知道的这两个函数是不一样的，一个是阻塞函数，一个是触发中断函数，阻塞函数是需要占用主函数运行时间的，而中断触发函数是不会占用我们主函数的执行时间的，并且我们要注意的HAL_UART_Receive_IT();只会触发一次，所以需要我们手动重置，也就是在写一次HAL_UART_Receive_IT();

所以我们的计划的实验效果是：主函数中以每秒一下的频率发送“BUG XIONG”，然后串口接收到我们电脑发送的字符串就让LED状态反转；

大体代码：

while{

HAL_UART_Transmit();//发送“BUG XIONG”

HAL_Delay();//延时一秒

}

中断触发函数{

//反转LED状态

HAL_UART_Receive_IT();//清楚重置接收中断

}

代码书写：

主函数部分：

```
int main(void)
{
  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();
  MX_USART1_UART_Init();

  HAL_UART_Receive_IT(&huart1,buf,3);//开启接收中断

  while (1)
  {
      HAL_UART_Transmit(&huart1,(uint8_t*)"BUG XIONG\r\n",11,0xFFFF);//串口发送"BUG XIONG\r\n"一共是是11位哦!
      HAL_Delay(1000);//延时1000毫秒
  }
  /* USER CODE END 3 */
}
```

串口中断部分：

```
uint8_t buf[10];//定义一个存数据的数组
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)//接收完成中断
{
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, 1-
HAL_GPIO_ReadPin(LED_GPIO_Port, LED_Pin));//LED状态反转
    HAL_UART_Receive_IT(&huart1,buf,3);//重新开启接收中断
}
```

好了我的代码虽然不多，但是需要小伙伴们自己动手操作看效果哦！快动手试试吧！