4 串口回环实验

4.1. 背景介绍

本章实验将通过串口中断的方式实现数据的发送和接收，首先简单的了解一下串口及串口寄存器映射。

4.1.1. UART 简介

通用异步收发传输器（Universal Asynchronous Receiver/Transmitter，UART）是异步串行通信口的总称，其在数据发送时将并行数据转换成串行数据来传输，在数据接收时将接收到的串行数据转换成并行数据，可以实现全双工传输和接收。

串行接口在数据传输时是一位一位地按照顺序传送的，其特点是通信线路简单，只要一对传输线就可以实现双向通信，从而大大降低了成本，特别适用于远距离通信，但传送速度较慢。一条信息的各位数据被逐位按顺序传送的通讯方式称为串行通讯。串行通讯的特点是：数据位传送，传按位顺序进行，最少只需一根传输线即可完成；成本低但传送速度慢。串行通讯的距离可以从几米到几千米。根据信息的传送方向，串行通讯可以进一步分为单工、半双工和全双工三种。

4.1.2. UART 寄存器映射

下表表 4- 1 显示了 APB UART 的寄存器映射。寄存器地址=基地址+基地址偏移量。其中 UART0 的基地址为 0x4000_4000，UART1 的基地址为 0x4000_5000。

表 4- 1 UART 的寄存器映射表

Name	Base offset	Type	Width	Reset value	Description
DATA	0x000	RW	8	0x--	[7:0] Data value. Read Received data. Write Transmit data.
STATE	0x004	RW	4	0x0	[3] RX buffer overrun, write 1 to clear. [2] TX buffer overrun, write 1 to clear. [1] RX buffer full, read-only. [0] TX buffer full, read-only.
CTRL	0x008	RW	7	0x00	[6] High-speed test mode for TX only. [5] RX overrun interrupt enable. [4] TX overrun interrupt enable. [3] RX interrupt enable. [2] TX interrupt enable. [1] RX enable. [0] TX enable.
INTSTATUS INTCLEAR	0x00C	RW	4	0x0	[3] RX overrun interrupt. Write 1 to clear. [2] TX overrun interrupt. Write 1 to clear. [1] RX interrupt. Write 1 to clear. [0] TX interrupt. Write 1 to clear.

BAUDDIV	0x010	RW	20	0x00000	[19:0] Baud rate divider. The minimum number is 16.
PID4	0xFD0	RO	8	0x04	Peripheral ID Register 4: value = 8'h4
PID5	0xFD4	RO	8	0x00	Peripheral ID Register 5: value = 8'h0
PID6	0xFD8	RO	8	0x00	Peripheral ID Register 6: value = 8'h0
PID7	0xFDC	RO	8	0x00	Peripheral ID Register 7: value = 8'h0
PID0	0xFE0	RO	8	0x21	Peripheral ID Register 0: value = 8'h21
PID1	0xFE4	RO	8	0xB8	Peripheral ID Register 1: value = 8'hb8
PID2	0xFE8	RO	8	0x1B	Peripheral ID Register 2: value = 8'h1b
PID3	0xFEC	RO	8	0x00	Peripheral ID Register 3: value = 8'h0
CID0	0xFE0	RO	8	0x0D	Component ID Register 0: value = 8'hd
CID1	0xFE4	RO	8	0xF0	Component ID Register 1: value = 8'hf0
CID2	0xFE8	RO	8	0x05	Component ID Register 2: value = 8'h5
CID3	0xFEC	RO	8	0xB1	Component ID Register 3: value = 8'hb1

4.2. 实验介绍

本次实验通过串口中断的方式实现数据的发送和接收，并且通过重写 fputc 函数的方式实现串口打印功能。

4.3. 建立 HQFPGA 工程

将已有的 HQ 工程复制到存放本次实验代码的文件夹下，并且对其进行修改，操作方式参考实验二中 2.3 节中的内容。

4.4. 物理管脚约束

本次实验使用 UART0，在引脚约束文件“ac208.upc”中对其进行引脚分配，分配完成之后保存文件，如下图 4. 1 所示。

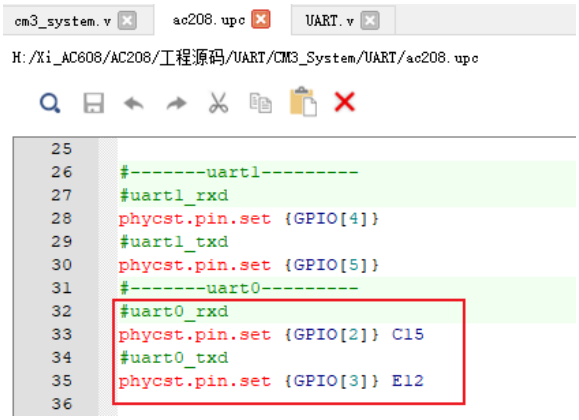

图 4. 1 分配串口引脚

4.5. 编译设计

点击“全部运行”按钮，对设计进行全编译并生成 bin 文件。操作步骤如下

图 4. 2 所示。

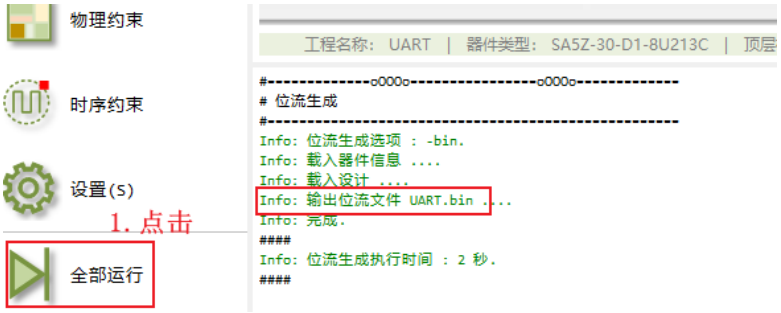

图 4. 2 编译运行整个文件

4.6. 建立 MDK 工程

将已有的 MDK 工程复制至本次工程的文件下，并对其进行修改，操作方式参考实验二中 2.8 节的内容。

4.7. 软件设计

对于串口的控制，官方提供的函数都在“CM3DS_uart”这个文件当中，针对本次实验，我们也提供有对应的应用库“uart”。使用之前，将这两个文件添加至工程当中（“CM3DS_uart”文件添加复制至 CMSIS 文件夹之中，“uart”文件复制至 HARDWARE 文件夹当中），添加方式参考 3.6.1 节中的内容。如果需要使用串口打印（printf）功能，需要将“regtarget.c”这个文件添加至工程当中。

4.7.1. UART 库

4.7.1.1. Uart_Init

初始化串口，函数的变量说明如下表 4- 2 所示。

表 4- 2 Uart_Init 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_UARTx	使用哪一个串口，选择串口 0（CM3DS_MPS2_UART0）或者串口 1（CM3DS_MPS2_UART1）
Uart_BR	设置串口波特率

函数内容：将串口对应的 IO 口进行复用，配置串口波特率，同时使能发送和接收。配置串口波特率就是向 BAUDDIV 寄存器中设置相应的值，例如，串口工作的时钟频率为 100Mhz，需要设置的波特率为 9600 时，那么需要向 BAUDDIV 寄存器写入$(100\times10^6)/9600$。使能发送和接收就是向串口的 CTRL 寄存器的第 0 位和第 1 位写入 1。对于串口 0 的初始化，代码如下所示：

```
if( CM3DS_MPS2_UARTx == CM3DS_MPS2_UART0){//如果使用串口 0
    //对串口的 IO 口进行复用，对应复用 GPIO2 和 GPIO3
```

```
    GPIO_PinRemapConfig(CM3DS_MPS2_GPIO0,GPIO_Remap_USART0_RXD,ENABLE);
    GPIO_PinRemapConfig(CM3DS_MPS2_GPIO0,GPIO_Remap_USART0_TXD,ENABLE);
    //对串口的结构体变量参数进行配置
    UART_InitStruct.UART_BundRate = Uart_BR;//配置波特率
    //同时使能发送和接收
    UART_InitStruct.UART_CTRL = UART_CTRL_TxEnable | UART_CTRL_RxEnable;
    //初始化串口 0
    UART_Init(CM3DS_MPS2_UART0, &UART_InitStruct);
}
```

函数使用示例如下：

```
Uart_Init(CM3DS_MPS2_UART0,9600);//初始化串口,波特率设置为 9600
```

4.7.1.2. UART_EXTI_Init

初始化串口中断，函数的变量说明如下表 4- 3 所示。

表 4- 3 UART_EXTI_Init 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_UARTx	使用哪一个串口，选择串口 0（CM3DS_MPS2_UART0）或者串口 1（CM3DS_MPS2_UART1）
en	中断使能信号；ENABLE:使能；DISABLE：不使能

函数内容：使能串口的接收中断（CTRL 寄存器的第 3 位置 1），清除挂起的中断，根据 en 的值判断，是否启用外部中断。对于串口 0 的中断初始化代码如下所示：

```
if(CM3DS_MPS2_UARTx == CM3DS_MPS2_UART0){
    //使能串口 0 的接收中断
    UART_ITConfig(CM3DS_MPS2_UART0,UART_CTRL_RxInterruptEnable,ENABLE);
    //清除挂起的中断
    NVIC_ClearPendingIRQ(UART0_IRQn);
    if(en == ENABLE){
        //启用外部中断
        NVIC_EnableIRQ(UART0_IRQn);
    }
    else
        NVIC_DisableIRQ(UART0_IRQn);
    }
```

4.7.1.3. Uart_ReceiveData

读取串口数据寄存器接收到的数据，并将接收到的数据返回。函数只有一个变量：CM3DS_MPS2_UARTx，使用串口 0 或者串口 1。函数代码如下所示：

```
unsigned char Uart_ReceiveData(CM3DS_MPS2_UART_TypeDef *CM3DS_MPS2_UARTx)
{
    uint8_t read_data;
```

```
    read_data = (uint8_t) (CM3DS_MPS2_UARTx->DATA & (uint8_t) 0xff);
    return  read_data;
}
```

函数的使用示例如下：

```
Uart_ReceiveData(CM3DS_MPS2_UART0);//读取串口 0 接收到的数据
```

4.7.1.4. Uart_SendData

将需要发送的数据写入到数据寄存器中，函数的变量说明如下表 4- 4：

表 4- 4 Uart_SendData 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_UARTx	使用哪一个串口，选择串口 0（CM3DS_MPS2_UART0）或者串口 1（CM3DS_MPS2_UART1）
Data	需要发送的数据

函数的代码如下所示：

```
void Uart_SendData(CM3DS_MPS2_UART_TypeDef *CM3DS_MPS2_UARTx, uint8_t Data)
{
    CM3DS_MPS2_UARTx->DATA = (Data & (uint8_t)0xff);
}
```

函数的使用示例如下：

```
Uart_SendData(CM3DS_MPS2_UART0,0x98);//通过串口 0 发送 0x98
```

4.7.1.5. Uart_ClearInt

清除对应的中断，向 INTCLEAR 寄存器中对应位写 1 即可清除对应中断，对于该寄存器的每一位的介绍参看 4.1.2 一节中的表 4- 1 中的内容。函数代码如下所示：

```
void Uart_ClearInt(CM3DS_MPS2_UART_TypeDef *CM3DS_MPS2_UARTx,uint16_t UART_INTFLAG)
{
    CM3DS_MPS2_UARTx->INTCLEAR = UART_INTFLAG;
}
```

函数的使用示例如下，清除发送中断：

```
Uart_ClearInt(CM3DS_MPS2_UART0,UART_INTCLEAR_TxInterrupt);
```

4.7.1.6. uart_tx

串口发送数据函数，函数的变量说明如下表 4- 5 所示。

表 4- 5 uart_tx 函数变量说明表

变量名称	变量意义
CM3DS_MPS2_UARTx	使用哪一个串口，选择串口 0（CM3DS_MPS2_UART0）或者串口 1（CM3DS_MPS2_UART1）

*data	需要发送的数组
len	需要发送的数据的长度

函数内容：将需要发送的数据拷贝到 buffer 中，然后将待发送数据的长度传递给串口发送长度变量中，最后打开发送中断（向 CTRL 寄存器的第 2 位写入 1）。需要注意的是，必须手动将需要发送的数组的第 0 位写入到 buffer 中，触发中断，如果不进行该操作，将无法触发中断，进入到中断服务函数当中。函数代码如下所示：

```
void uart_tx(CM3DS_MPS2_UART_TypeDef *CM3DS_MPS2_UARTx,unsigned char *data,
unsigned char len)
{
        memcpy(uart_txbuffer,data,len);//将需要发送的数据拷贝到 buffer 中
        txbuffer_len = len;//将待发送的数据的长度传递给串口发送长度变量当中
        //打开发送中断
        UART_ITConfig(CM3DS_MPS2_UARTx,UART_CTRL_TxInterruptEnable,ENABLE);
        txbuffer_len = txbuffer_len - 1;
        //将第一个数据写入 buffer 中,触发中断
        Uart_SendData(CM3DS_MPS2_UARTx,uart_txbuffer[0]);
}
```

函数的使用示例如下：

```
uart_tx(CM3DS_MPS2_UART0,(unsigned char *)"hello!\n",sizeof("hello!\n"));
```

4.7.1.7. uart_rx

串口发送数据函数，函数只有一个输入：*data，存放接收到的数据。

函数内容：将接收到的数据拷贝到 data 中，存储接收到的数据长度，清零接收长度，最后返回读到的数据长度，函数代码如下所示：

```
unsigned char uart_rx(unsigned char *data)
{
    unsigned char len;
    memcpy(data,uart_rxbuffer,rxbuffer_len);//将接收到的数据拷贝到 data 中
    len = rxbuffer_len;//存储接收到的数据长度
    rxbuffer_len = 0;//清零接收长度
    return len;//返回读到的数据长度
}
```

函数的使用示例如下：

```
rx_len = uart_rx((unsigned char *)data_tmp);
```

4.7.1.8. UART0_Handler

串口 0 的中断服务函数。当数据接收完成之后，产生接收中断（INTSTATUS 寄存器的第 1 位为 1），将接收的数据存储至接收 buffer 中，每接收一次数据长度加 1，最后清除接收中断（向 INTCLEAR 寄存器的第 1 位写 1），方便下次使

用；当发送 buffer 还未满的时（STATE 寄存器的第 0 位为 0），检测 txbuffer_len 的值：当 txbuffer_len 不为 0 的时，代表 buffer 中有数据，需要将数据发送出去；当 txbuffer_len 为 0 的时，表示 buffer 中已经没有数据了，串口此时处于空闲的状态，清除发送中断（向 INTCLEAR 寄存器的第 0 位写 1），方便下次使用。需要注意的是，由于我们已经将需要发送的数组的第 0 位发送出去了，那么在中断服务函数中就需要从第 1 位开始发送。函数代码如下所示：

```
void UART0_Handler(void)
{
    static unsigned char i = 1;
    unsigned char Rx_BufferOver;
    unsigned char Rx_INTFLAG,TX_BufferFull;
    Rx_INTFLAG = UART_GetITStatus(CM3DS_MPS2_UART0,UART_INTCLEAR_RxInterrupt);
    TX_BufferFull = UART_GetFlagStatus(CM3DS_MPS2_UART0,UART_TxBufferFull_FLAG);
    Rx_BufferOver=UART_GetFlagStatus(CM3DS_MPS2_UART0,UART_RxBufferOverrun_FLAG);
    //如果数据溢出
    if(Rx_BufferOver)
    {
        //清除溢出中断
        Uart_ClearInt(CM3DS_MPS2_UART0,UART_INTCLEAR_RxInterruptOverrun);
    }
    //接收完成产生中断
    if(Rx_INTFLAG){
        //将接收到的数据存储在接收 buffer 中
        uart_rxbuffer[rxbuffer_len] = Uart_ReceiveData(CM3DS_MPS2_UART0);
        rxbuffer_len++;
        Uart_ClearInt(CM3DS_MPS2_UART0,UART_INTCLEAR_RxInterrupt);//清除接收中断
    }
    //发送非满，可发送新数据
    if(TX_BufferFull == 0){
        if(txbuffer_len){
            //buffer 有数据时，将需要发送的数据发送出去
            Uart_SendData(CM3DS_MPS2_UART0,uart_txbuffer[i]);
            txbuffer_len--;
            i++;
        }
        else{
            //当 buffer 里面没有数据的时候
            i = 1;
            uart0_tx_state = UART_TX_IDLE;//串口此时处于空闲状态
            //清除发送中断
            Uart_ClearInt(CM3DS_MPS2_UART0,UART_INTCLEAR_TxInterrupt);
```

```
        }
    }
}
```

4.7.2. 添加用户程序

本次实验需要实现的功能：每隔 1S 打印一次数据，通过串口调试助手手动发送数据，在接收区显示我们手动发送的数据。

实验代码编写：通过定时器的方式定时 1S，当定时时间到了之后，定时器标志位 timer_flag 至 1，表示此时可以通过 printf 打印数据。将串口读取到的数据存放至串口接收数据的临时缓存区 data_tmp 当中，然后通过 uart0_tx_state 的值判断串口的工作状态，等到串口空闲的时候，便可以开始发送数据。函数代码如下所示：

```
#include "CM3DS_rcc.h"
#include "CM3DS_gpio.h"
#include "CM3DS_uart.h"
#include "CM3DS_MPS2.h"
#include "uart.h"
#include "stdio.h"
#include "timer.h"
unsigned char data_tmp[1024] = {0};//串口接收的数据临时缓存
extern unsigned char uart0_tx_state;//标记串口的工作状态
extern unsigned char timer_flag;
void delay(uint32_t t);
int main(void)
{
    unsigned char rx_len = 0;//接收到的数据长度
    uart0_tx_state = UART_TX_IDLE;//初始化串口状态标志为空闲
    Uart_Init(CM3DS_MPS2_UART0,9600);//初始化串口
    UART_EXTI_Init(CM3DS_MPS2_UART0,ENABLE);//初始化串口中断
    Timer_Init(CM3DS_MPS2_TIMER0,SystemCoreClock,SystemCoreClock);//定时 1S
    TIM_EXTI_Init(CM3DS_MPS2_TIMER0,ENABLE);
    while(1){
        if(timer_flag == 1){//定时发送数据
            printf("hello world!\n");
            timer_flag = 0;
        }
        //读取串口接收到的数据
        rx_len = uart_rx((unsigned char *)data_tmp);
        if(rx_len){//如果接收到数据
            //串口 0 忙则一直等待,代表还在发送数据
            while(uart0_tx_state == UART_TX_BUSY);
```

```
            //发送接收到数据
            uart_tx(CM3DS_MPS2_UART0,(unsigned char *)data_tmp,rx_len);
            uart0_tx_state = UART_TX_BUSY;//标记串口 0 状态为忙
        }
       delay_ms(1);//延时 1ms
    }
}
```

4.8. 板级验证

4.8.1. 实验所需硬件

(1) AC208 开发板
(2) FPGA 下载器：XIST USB Cable
(3) CM3 仿真器：DAP Link
(4) 电源线一根

4.8.2. 硬件连接

硬件连接参考实验一。

4.8.3. 下载文件至目标板

下载文件的方式参考实验一。

4.8.4. 功能演示

首先将串口调试助手打开，找到端口号，串口波特率选择 9600，收码区和发码区都设置为字符串。然后下载程序，此时我们可以看到在一直接收到数据“hello world!”，在发码区输入你想要发送的数据，在收码区可以看到我们手动发送的数据，此时表示功能正常，如下图 4. 3 所示。

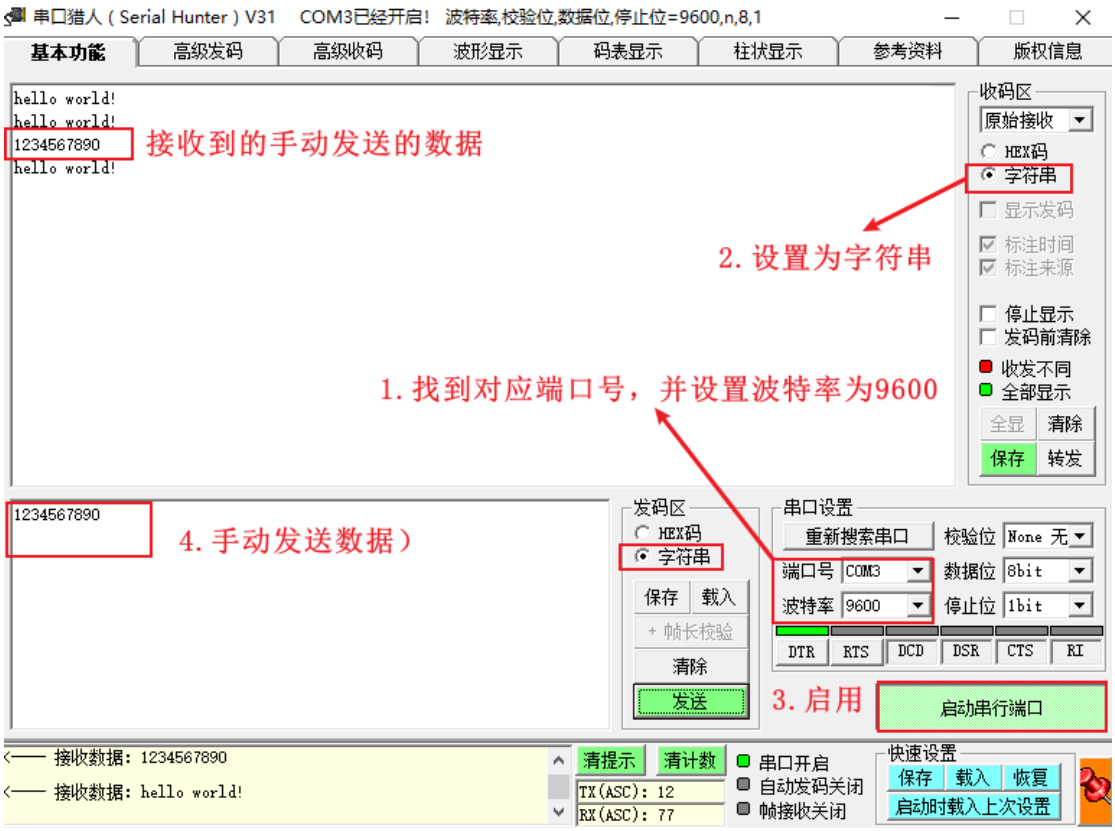

图 4. 3 串口实验结果显示

当打开串口助手之后，没有接收到数据，从如下几个方面去排查问题。

1. 程序和串口助手的波特率是否设置一致。

2. 打开设备管理器，检查是否检测到了串口，这里提供有两种方式连接串口和电脑之间进行数据传输：

(1) 通过两根杜邦线，将 DAP Link 模块上的 RXD 和 TXD 接口连接至开发板上，连接方式如下图 4. 4 所示。

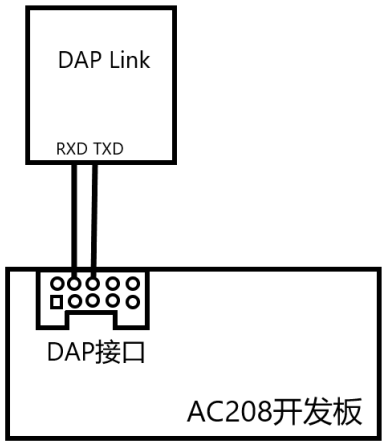

图 4. 4 DAP Link 模块上的串口连接图

此时，打开设备管理器，应该检测到如下图 4. 5 所示的端口

图 4. 5 DAP Link 串口端口检测示意图

(2) 用一根 Type-C 的线连接开发板和电脑，开发板上的 Type-C 接口如下图 4. 6 所示。

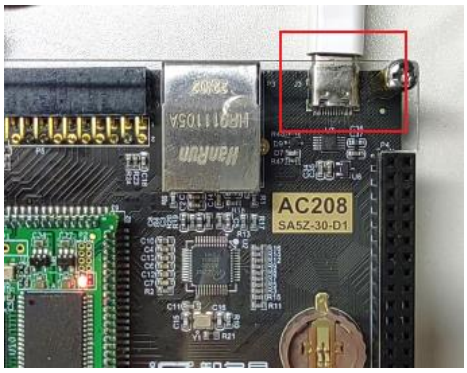

图 4. 6 开发板上 Type-C 接口示意图

此时，打开设备管理器，应该检测到如下图 4. 7 所示的端口。

图 4. 7 Type-C 接口连接串口端口检测示意图

4.9. 思考与总结

本次实验通过串口中断的方式，实现了数据的发送和接收，并且通过重写 fputc 函数实现了 printf 的功能。本次实验需要注意以下几点：

1. 数据和接收和发送之后，需要增加一个延时，然后再进行下一次的传输。这是由于串口产生中断的时间非常短，当我们连续发送的时候，可能会导致中断

冲突，使得数据接收出错。

2. 在发送数据的时候，首先要将接收到的第一个数据发送出去，以此来触发中断，然后程序将进入中断服务函数中，继续发送接收到的数据。

3. 重写的 fputc 函数位于 retarget.c 文件中，需要使用 printf 功能的时候，需要将其添加至工程当中。