第7章 串行通信接口（一）

7.1 串行通信的基础知识

7.2 UART驱动构件及使用方法

7.3 串行通信的编程举例

串行通信接口虽然在PC机上现在使用较少，但在嵌入式开发中却发挥着重要作用，是重要的打桩调试手段。通过串行通信接口的编程，有助于理解MCU与上位机之间的通信原理与方式。本章主要介绍串行通信的基础知识，分析UART驱动构件的要素，最后通过编程举例进一步理解串行通信的原理和UART构件的使用方法。

7.1 串行通信的基础知识

串行通信接口，简称“串口”、UART或SCI。在USB未普及之前，串口是PC机必备的通信接口之一。MCU中的串口通信，在硬件上，一般只需要三根线，分别称为发送线（TxD）、接收线（RxD）和地线（GND）；通信方式上，属于单字节通信，是嵌入式开发中重要的打桩调试手段。实现串口功能的模块在一部分MCU中被称为通用异步收发器（UART），在另一些MCU中被称为串行通信接口（SCI）。

7.1.1 串行通信的基本概念

串行通信的特点是：数据以字节为单位，按位的顺序（例如最高位优先）从一条传输线上发送出去。串行通信分为异步通信与同步通信两种方式，本节主要给出异步串行通信的一些常用概念。正确理解这些概念，对串行通信编程是有益的。主要掌握异步串行通信的格式、波特率、串行通信传输方式等。

1. 异步串行通信的格式

在串口出现初期，异步串行通信采用的是NRZ数据格式，可以译为：“标准不归零传号/空号数据格式”。“不归零”的最初含义是：用负电平表示一种二进制值，正电平表示另一种二进制值，不使用零电平，这是初期规定的（叫做RS-232电平）。“mark/space”即“传号/空号”分别是表示两种状态的物理名称，逻辑名称记为“1/0”。

8位数据、无校验情况的传送格式(逻辑表示)

逻辑上，这种格式的空闲状态为“1”，发送器通过发送一个“0”表示一个字节传输的开始，随后是数据位。最后，发送器发送1到2位的停止位，表示一个字节传送结束。若继续发送下一字节，则重新发送开始位，开始一个新的字节传送。若不发送新的字节，则维持“1”的状态，使发送数据线处于空闲。从开始位到停止位结束的时间间隔称为一字节帧，也称这种格式为字节帧格式。每发送一个字节，都要发送“开始位”与“停止位”，这是影响异步串行通信传送速度的因素之一。

进一步解释：异步就是指每发一个字节重新开始

随堂练习：画出发送字符“B”的时序图（考试要求）

第1步：写出字符“B”的十六进制ascii码

第2步：把十六进制转为二进制

第3步：根据该二进制数，画出时序图

随堂练习：从MCU引脚来看，若逻辑1对应3.3V，逻辑0对应0V，不停地发送字符0，用万用表测量发送线，应该为几伏？

异步串行通信和同步串行通信的区别

1、异步通信方式：

原理： 异步通信是按字符传输的。每传输一个字符就用起始位来进来收、发双方的同步。不会因收发双方的时钟频率的小的偏差导致错误。这种传输方式利用每一帧的起、止信号来建立发送与接收之间的同步。

特点： 每帧内部各位均采用固定的时间间隔，而帧与帧之间的间隔时随即的。接收机完全靠每一帧的起始位和停止位来识别字符时正在进行传输还是传输结束。

案例： USART通信

2、同步通信方式：

原理： 进行数据传输时，发送和接收双方要保持完全的同步，因此，要求接收和发送设备必须使用同一时钟

特点： 可以实现高速度、大容量的数据传送；

缺点： 要求发生时钟和接收时钟保持严格同步，同时硬件复杂。

案例： SPI

3、相似： 都需要进行同步，异步通信是自同步，同步通信是外同步

2. 串行通信的波特率

位长，也称为位的持续时间，其倒数就是单位时间内传送的位数，人们把每秒内传送的位数叫做波特率。波特率的单位是：位/秒，记为bps，是bit per second的缩写。

常用波特率有9600、19200、38400、57600、115200等（注意是翻倍）。

异步串行通信的速度却不能提得很高，因为随着波特率的提高，位长变小，很容易受到电磁源的干扰，通信就不可靠。还有通信距离问题，距离小，可以适当提高波特率，但这样毕竟提高的幅度非常有限，达不到大幅度提高的目的。

3. 串行通信传输方式术语（了解）

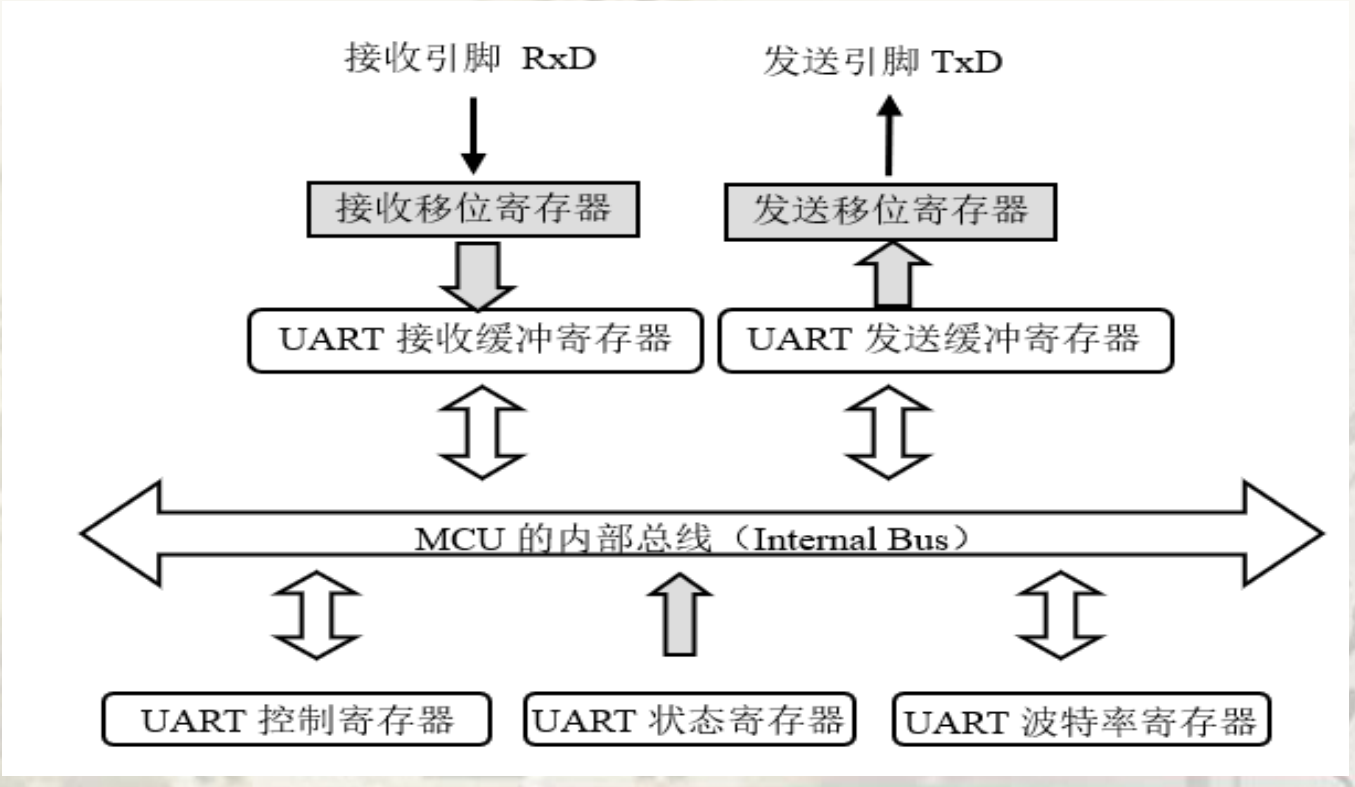
（1）全双工：数据传送是双向的，同时接收与发送数据。除地线之外，需两根数据线，站在任何一端角度看，一根为发送线，另一根为接收线。一般情况下，MCU的异步串行通信接口均是全双工的。

（2）半双工：数据传送是双向的。除地线之外，一般只有一根数据线。任何时刻，只能由一方发送数据，另一方接收数据，不能同时收发。

（3）单工：数据传送是单向的，一端为发送端，另一端为接收端。这种传输方式中，除了地线之外，只要一根数据线就可以了，如有线广播就是采用单工传输的。

7.1.2 串行通信编程模型

串行通信接口UART的主要功能是：接收时，把外部单线输入的数据变成一个字节的并行数据送入MCU内部；发送时，把需要发送的一个字节的并行数据转换为单线输出。



7.1.2 串行通信编程模型

- 波特率寄存器：用于设置UART的波特率
- 控制寄存器
  - 设置通信格式、是否校验、是否允许中断等
- 状态寄存器
  - 串口是否有数据可收、数据是否发送出去等
- 数据寄存器
  - 存放要发送的数据，也存放接收的数据
  - 发送与接收的实际工作是通过“发送移位寄存器”和“接收移位寄存器”完成的
  - 驱动程序设计时，并不直接操作“发送移位寄存器”和“接收移位寄存器”
  - MCU中没有设置“发送移位寄存器”和“接收移位寄存器”的映像地址
  - STM32L431RCT：UART保护发送数据寄存器(TDR)和接收数据寄存器(RDR)

7.1.2 串行通信编程模型

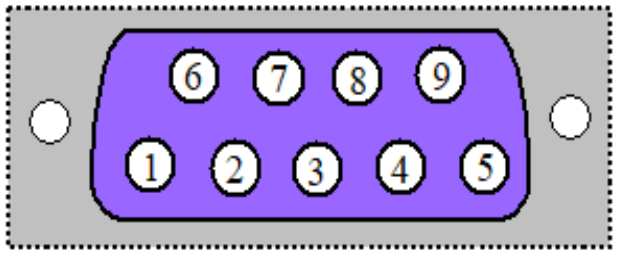
- 数据发送过程：
  - 判断状态寄存器(ISR)： Bit 7 TXE: Transmit data register empty, 1—empty,0—no
  - 如果TXE==1： 将数据保存到TDR中，否则等待
  - 数据保存到TDR之后，就会自动将数据送入“发送移位寄存器”，然后一位一位的发送到引脚TxD。
- 数据接收过程
  - 数据数据一位一位地从接收引脚RxD进入“接收移位寄存器”
  - 当收到一个完整字节时，MCU会自动将数据送入“UART数据寄存器”，并将状态寄存器的相应位改变
  - 程序检测到状态寄存器(ISR):RXNE: Read data register not empty, 1—no, 0--empty
  - 检测到RXNE==1则从数据寄存器中读取数据，否则等待
  - 判断RXNE的方式：
    - 轮询：不断检测该位，编程简单，程序效率低
    - 中断：控制寄存器(CR1):Bit 5 RXNEIE: RXNE interrupt enable可开启接收中断

7.1.3 RS232、RS485总线标准

- **TTL电平**
  - MCU引脚输入/输出一般使用TTL电平，即晶体管-晶体管逻辑电平
  - TTL电平的“1”和“0”的特征电压分别为2.4V和0.4V（目前使用3.3V供电的MCU中，该特征值有所变动），即大于2.4V则识别为“1”，小于0.4V则识别为“0”。
  - 适用于板内数据传输，若用TTL电平将数据传输到5米之外，那么可靠性就会急剧降低。
- **串行物理接口标准RS232C**
  - 目的：为使信号传输得更远
  - 美国电子工业协会EIA制订
  - RS232采用负逻辑，-15V～-3V为逻辑“1”，+3V～+15V为逻辑“0”。
  - RS232最大的传输距离是30m，通信速率一般低于20Kbps。
  - RS232接口：25芯插头和9芯插头(更常用)

7.1.3 RS232、 RS485总线标准

在计算机的串行通信中，25芯线中的大部分并不使用，逐渐改为使用9芯串行接口。一段时间内，市场上还有25芯与9芯的转接头，方便了两种不同类型之间的转换。目前几乎所有计算机上的串行口都是9芯接口。



计算机中常用的 9 芯串行接口引脚含义表

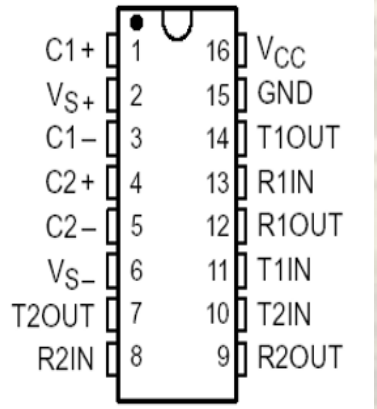
引脚号	功能	引脚号	功能
1	接收线信号检测	6	数据通信设备准备就绪(DSR)
2	接收数据线(RxD)	7	请求发送(RTS)
3	发送数据线(TxD)	8	允许发送(CTS)
4	数据终端准备就绪(DTR)	9	振铃指示
5	信号地(SG)		

7.1.3 RS232、RS485总线标准

- 精简的RS232通信
  - 通信时仅使用3根线：RxD（接收线）、TxD（发送线）和GND（地线）。
  - 其他为进行远程传输时接调制解调器之用，有的也可作为硬件握手信号（如请求发送RTS信号与允许发送CTS信号），初学时可以忽略这些信号的含义。
- RS485标准
  - 方便组网
  - RS485采用差分信号负逻辑，-2V～-6V表示“1”，+2V～+6V表示“0”
  - 硬件连接上，采用两线制接线方式，工业应用较多。
  - 由于PC机默认只带有RS232接口，因此，市场上有RS232-RS485转接头出售，这种转接只是硬件电平信号之间的转换，与MCU编程无关。

7.1.4 TTL电平到RS232电平转换电路

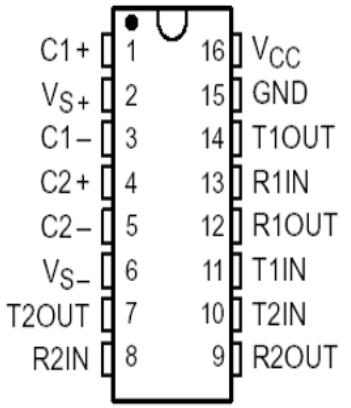
- 在MCU中，若用RS232总线进行串行通信，则需外接电路实现电平转换在发送端，需要用驱动电路将TTL电平转换成RS232电平；在接收端，需要用接收电路将RS232电平转换为TTL电平。
- 电平转换器：由晶体管分立元件构成；直接使用集成电路，引脚含义简要说明如下：Vcc（16脚）：正电源端，一般接+5V；GND（15脚）：地；VS+（2脚）：VS+=2Vcc-1.5V=8.5V；VS-（6脚）：VS-=-2Vcc-1.5V=-11.5V；C2+、C2-（4、5脚）：一般接1μF的电解电容；C1+、C1-（1、3脚）：一般接1μF的电解电容。输入输出引脚分两组，基本含义见表。在实际使用时，若只需要一路串行通信接口，可以使用其中的任何一组。



组别	TTL 电平引脚	方向	典型接口	232 电平引脚	方向	典型接口
1	11（T1IN）	输入	接 MCU 的 TxD	13（R1IN）	输入	接到 9 芯接口的 2 脚 RxD
	12（R1OUT）	输出	接 MCU 的 RxD	14（T1OUT）	输出	接到 9 芯接口的 3 脚 TxD
2	10（T2IN）	输入	接 MCU 的 TxD	8（R2IN）	输入	接到 9 芯接口的 2 脚 RxD
	9（R2OUT）	输出	接 MCU 的 RxD	7（T2OUT）	输出	接到 9 芯接口的 3 脚 TxD

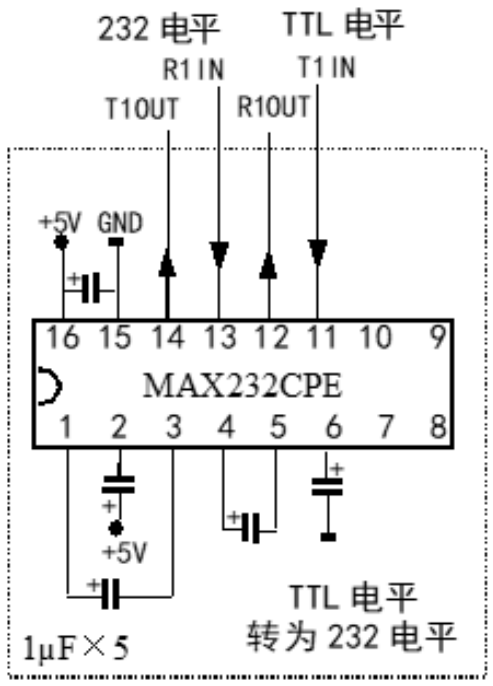
7.1.4 TTL电平到RS232电平转换电路

- 焊接到PCB板上的MAX232芯片检测方法：
  - 正常情况下
  - （1）T1IN=5V，则T1OUT=-9V；T1IN=0V，则T1OUT=9V。
  - （2）将R1IN与T1OUT相连，令T1IN=5V，则R1OUT=5V；令T1IN=0V，则R1OUT=0V。



组别	TTL 电平引脚	方向	典型接口	232 电平引脚	方向	典型接口
1	11（T1IN）	输入	接 MCU 的 TxD	13（R1IN）	输入	接到 9 芯接口的 2 脚 RxD
	12（R1OUT）	输出	接 MCU 的 RxD	14（T1OUT）	输出	接到 9 芯接口的 3 脚 TxD
2	10（T2IN）	输入	接 MCU 的 TxD	8（R2IN）	输入	接到 9 芯接口的 2 脚 RxD
	9（R2OUT）	输出	接 MCU 的 RxD	7（T2OUT）	输出	接到 9 芯接口的 3 脚 TxD

具有串行通信接口的MCU，一般具有发送引脚（TxD）与接收引脚（RxD），不同公司或不同系列的MCU，使用的引脚缩写名可能不一致，但含义相同。串行通信接口的外围硬件电路，主要目的是将MCU的发送引脚TxD与接收引脚RxD的TTL电平，通过RS232电平转换芯片转换为RS232电平，图7-5给出了基本串行通信接口的电平转换电路。进行MCU的串行通信接口编程时，只针对MCU的发送与接收引脚，与MAX232无关，MAX232只是起到电平转换作用。



发送过程：MCU的TxD（TTL电平）经过MAX232的11脚（T1IN）送到MAX232内部，在内部TTL电平被“提升”为232电平，通过14脚（T1OUT）发送出去。

接收过程：外部232电平经过MAX232的13脚（R1IN）进入到MAX232的内部，在内部232电平被“降低”为TTL电平，经过12脚（R1OUT）送到MCU的RxD，进入MCU内部。

随着USB接口的普及，9芯串口正在逐渐从PC机，特别是从便携式电脑上消失。于是出现232-USB转换线、TTL-USB转换线，在PC机上安装相应的驱动软件，就可在PC机上使用一般的串行通信编程方式，通过USB接口实现与MCU的串行通信。

7.2 UART驱动构件及使用方法

7.2.1 UART驱动构件要素分析

UART具有初始化、发送和接收三种基本操作。串口初始化函数的参数应该有哪些？首先应该有串口号，因为一个MCU有若干串口，必须确定使用哪个串口；其次是波特率，必须确定使用什么速度进行收发，至于波特率使用哪个时钟来产生，这并不重要，这里确定使用系统总线时钟，就不需要传入这个参数了。关于奇偶校验，由于实际使用主要是多字节组成的一个帧，自行定义通信协议，单字节校验意义不大。此外，串口在嵌入式系统中的重要作用是实现类似C语言中printf函数功能，也不宜使用单字节校验，因此就不校验。这样，串口初始化函数就两个参数：串口与波特率。

从知识要素角度，进一步分析UART驱动构件的基本函数，与寄存器直接打交道的有：初始化、发送单个字节、接收单个字节、使能接收中断、禁止接收中断、获取接收中断状态等函数。发送中断不具有实际应用价值，可以忽略。

表7-3 UART驱动构件要素

序号	函数			形参		宏常数	备注
	简明功能	返回	函数名	英文名	中文名		
1	初始化	无	uart_init	uartNo	串口号	用	
				baud_rate	波特率	不用	直接用数字
2	串行发送 1 个字节的数据	函数执行状态： 1=发送成功 0=发送失败	uart_send1	uartNo	串口号	用	
				ch	要发送的字节	不用	
3	串行发送 N 个字节的数据	函数执行状态： 1=发送成功 0=发送失败	uart_sendN	uartNo	串口号	用	
				len	发送长度	不用	
				buff	发送缓冲区	不用	
4	从指定UART端口发送一个以'\0'结束的字符串	函数执行状态： 1=发送成功 0=发送失败	uart_send_string	uartNo	串口号	用	
				buff	要发送的字符串的首地址	不用	
5	串行接收 1 个字节的数据	接收返回字节	uart_re1	uartNo	串口号	用	
				*fp	接收成功标志的指针	不用	1=接收成功 0=接受失败

6	串行接收 n 个字节的数据，放入 buff 中	函数执行状态： 1=接收成功 0=接收失败	uart_reN	uartNo	串口号	用	
				len	接收长度	不用	
				buff	接收缓冲区	不用	
7	开串口接收中断	无	uart_enable_re_int	uartNo	串口号	用	
8	关串口接收中断	无	uart_disable_re_nt	uartNo	串口号	用	
9	获取接收中断标志，同时禁用发送中断	接收中断标志： 1=有接收中断 0=无接收中断	uart_get_re_int	uartNo	串口号	用	
10	UART 反初始化	无	uart_deinit	uartNo	串口号	用	
说明	（1）关于初始化的说明： 初始化 UART 模块，设定使用的串口号和波特率						

7.2.2 UART驱动构件使用方法

1. 包含构件头文件

在设计构件时，已经将uart构件需要使用到的寄存器地址、引脚等信息在uart.inc头文件中进行了宏定义。因此，在使用uart构件时必须在user.inc中包含uart.inc这个头文件，语句如下：

```
.include “uart.inc”          //包含有关串口宏定义的头文件
```

2. 对构件进行初始化

在使用uart构件之前，首先需要在main.s中对构件进行初始化，即调用初始化函数，语句如下：

```
mov r0,#UARTA                //串口号参数放入r0
ldr r1,=UART_BAUD            //波特率参数放放r1
bl  uart_init                //调用串口初始化函数
```

其中“UARTA”代表具体的串口号，“UART_BAUD”代表具体的波特率，均为为了方便而进行的参数宏定义，读者可自行设置。

3．编写使能中断程序

由于uart构件需要使用到中断服务程序，故要在isr.s文件中编写串口接收中断服务程序，此处仍旧以串口UARTA为例，具体程序代码如下：

```
UARTA_Handler:
//（1）屏蔽中断，并且保存现场
        cpsid   i          //屏蔽中断
        push {lr}          //保存现场，将下一条指令地址入栈
//（2）接收字节
        mov r0,#UARTA
        bl  uart_re1                    //收到一个字节数据
//（3）发送字节
        mov r1,r0
        mov r0,#UARTA
        bl  uart_send1     //向原串口回发
//（4）解除屏蔽，并且恢复现场
        cpsie   i          //解除屏蔽中断
        pop {pc}           //恢复现场，返回主程序处继续执行
```

4. 开放使能中断

在主函数main.s文件中调用使能中断函数打开中断，语句如下：

```
mov r0,#UARTA            //将串口号参数放放r0
bl  uart_enable_re_int   //调用串口使用中断函数
```

5. 调用具体方法

在执行完以上四步以后，即可在需要的位置调用已封装好的具体方法了，此处以发送一个字符串为例，在需要调用的位置增加以下语句：

1）声明字符串

```
string_2:
    .asciz "Assembly call c's uart2!"
```

2）调用函数

```
mov r0,#UARTA                //r0←串口号
ldr r1,=string_2             //r1←字符串
bl  uart_send_string         //调用uart_send_string
```

7.3 串行通信的编程举例

为了实现芯片无关性和可移植性，本节内容仅提供编程的步骤和部分代码内容，对UART构件的汇编语言编程过程进行举例讲解。由于每种芯片的引脚和UART模块不尽相同，因此读者可根据具体的芯片及引脚要求选择需要使用的UART模块。

本节的两个例子中使用的测试工具均为金葫芦IoT-GEC开发套件以及上位机串口调试工具。金葫芦IoT-GEC开发套件串口测试使用底板上的标识为UART_User，在开发套件通电的情况下，用串口线连接，其中白色线连接Tx引脚，绿色线连接Rx引脚，黑色线连接GND引脚。

7.3.1 例1：发送和接收一个字节的数据

本例主要实现的功能是：用户在上位机使用串口调试工具，实现通过串口向开发套件的串口模块发送一个字节的数据，参考程序见“Exam7_1”工程。

1. 定义波特率

汇编语言中的可使用.equ伪指令来表示类似于C中宏定义的方式，定义波特率为115200。例如在工程文件05_UserBoard\user.inc中，有如下语句：

```
.equ UARTA, 2
.equ UART_BAUD,115200
```

2. 实现UARTA中断服务程序

由于已定义UARTA对应的是串口2，因此UARTA中断服务程序的实际名称为USART2_IRQHandler，在中断服务程序文件isr.s中需要添加串 2的接收中断服务程序的实现代码。例如在工程文件07_NosPrg\isr.s中，有如下语句：

```
USART2_IRQHandler:
//（1）屏蔽中断，并且保存现场
        cpsid  i                    //屏蔽中断
        push {lr}                   //保存现场，将下一条指令地址入栈
//（2）接收字节
        mov r0,#UARTA               //r0←串口号
        bl  uart_re1                //收到一个字节数据
//（3）发送字节
        mov r1,r0                   //r1←要发送的一个字节数据
        mov r0,#UARTA               //r0←串口号
        bl  uart_send1              //向原串口回发
//（4）解除屏蔽，并且恢复现场
        cpsie   i                   //解除屏蔽中断
        pop {pc}                    //恢复现场，返回主程序处继续执行
```

4. 主函数文件main.s的工作

（1）定义要发送的字符串。

在主函数文件main.s中，定义要发送的字符串，代码如下：

```
string_2:
    .asciz "Assembly call c's uart2! "
```

（2）UARTA模块初始化。

在主函数文件中，对UARTA模块进行初始化，指明串口号和波特率，代码如下：

```
mov r0,#UARTA              //r0←串口号
ldr r1,=UART_BAUD          //r1←波特率
bl  uart_init              //调用串口初始化函数
```

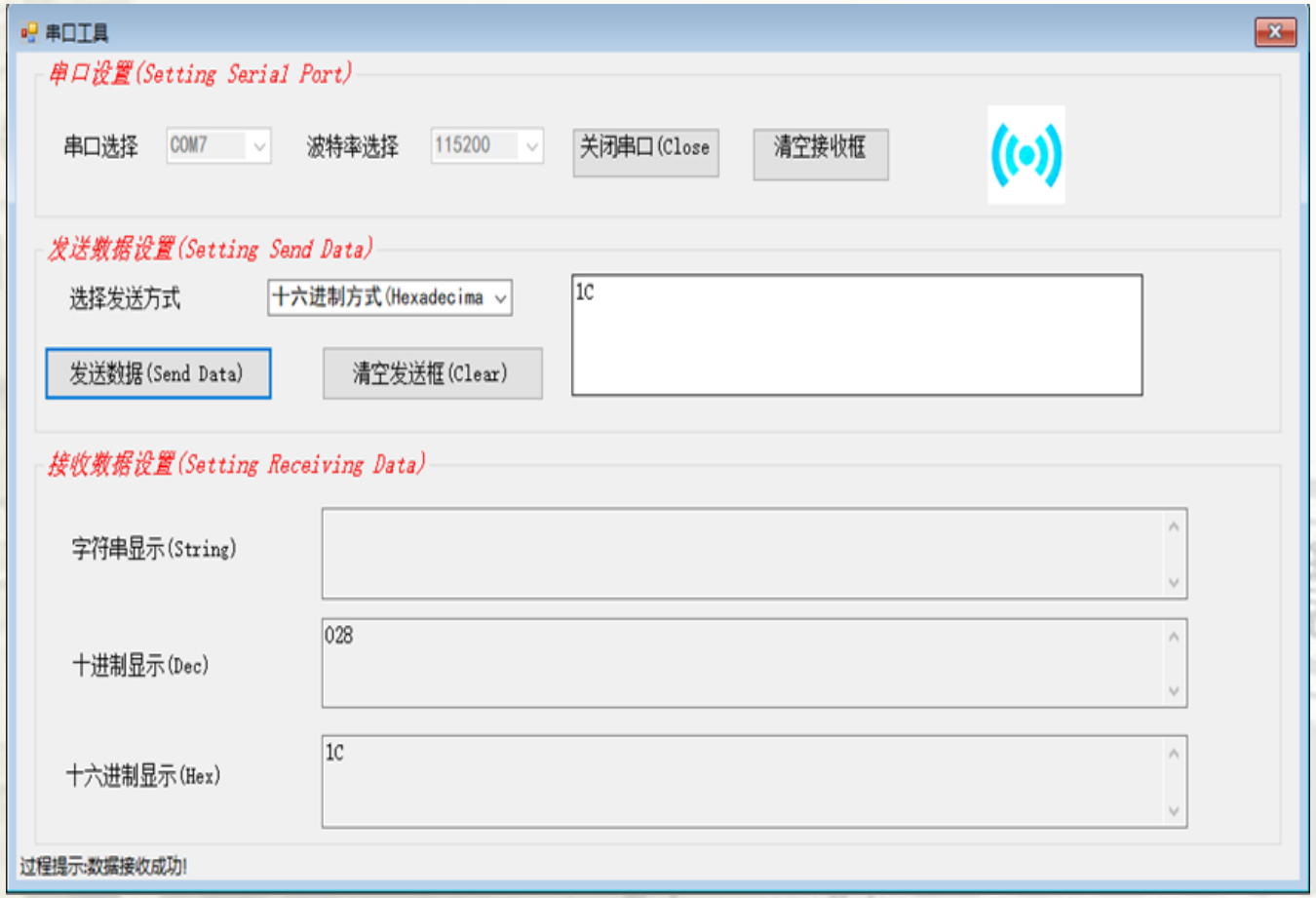
（3）使能模块中断。

在主函数文件中开放UARTA模块的使能模块终端，代码如下：

```
mov r0,# UARTA             //r0←串口号
bl  uart_enable_re_int     //调用串口中断使能函数
```

5. 下载程序机器码到目标板，观察运行情况

用SWD连接目标套件和PC机的USB端口，经过编译后生成机器码（Hex文件），通过AHL-GEC-IDE软件下载到目标开发套件中，并可在AHL-GEC-IDE的串口调试工具（单击“工具”菜单，选择“串口工具”可打开串口调试工具）中选择好串口，设置波特率为115200，选择发送方式为“十六进制方式”，单击“打开串口”按钮，在发送框中输入任意十六进制数如“1C”，接收数据显示框中也会立即显示出对应的数据。



7.3.2 例2：发送和接收一帧数据

数据帧格式由于使用方法的不同而各有不同，本节例子中使用的帧格式为：2字节帧头+1至2字节有效数据长度（与数据长度大小有关）+有效数据+2字节帧尾。其中为了读者方便理解，此处将帧头简易的设置为两个美元符号“$$”，将帧尾设置为两个井号“##”。本例主要实现的功能是：用户在上位机使用串口调试工具，通过串口实现对串口模块发送的数据进行封装成帧并接收，参考程序见“Exam7_2”工程。包含文件以及初始化等过程与例一相同。

1. 编写获取有效数据长度程序

在uart.s中编写一个内部函数，用于计算发送的字符串的长度，即帧中的有效数据长度，具体代码如下：

```
get_data_length:
//（1）保存现场，pc(lr)入栈
        push {lr}
//（2）初始字符串长度为0
        mov r5,#0
```

```
//（3）主循环
data_length_loop:
        ldrb r3,[r0,r5]                          //将当前地址中的内容加载到r3
//（3.1）判断当前是否为字符串终止符
        mov r4,#0
        cmp r3,r4
        beq data_length_end                      //是则循环结束，跳转到结尾
//（3.2）继续循环
        add r5,#1
        b data_length_loop
//（4）函数结尾，恢复现场，lr出栈到pc（即子程序返回），将字符串长度放入r0中返回
data_length_end:
        mov r0,r5
        pop {pc}
```

2. 编写封装成帧处理程序

在uart.s中编写一个内部函数，用于对要发送的字符串进行封装成帧处理，具体代码如下：

```
uart_make_frame:
//（1）保存现场，pc(lr)入栈
        push {lr}
        mov r4,r0                       //将有效数据长度暂存到r4
//（2）添加帧头
        ldr r2,=0x20003000              //使用RAM区USER可用地址作为帧头首地址
        mov r3,#36                      //r3←帧头数据“$”
        strb r3,[r2]                    //将“$”存入帧头首地址
        add r2,#1                       //地址后移一位
        strb r3,[r2]                    //将“$”存入帧头第二个地址
//（3）在帧头后添加表示有效数据长度的一至两个字节数据
        add r2,#1                       //地址后移一位
//（3.1）判断数据长度是否超过255
        cmp r0,#255
        bgt make_frame_data2            //超过则分配两个字节来表示数据长度
```

```
//（3.2）否则分配一个字节
        mov r3,#1                       //r3表示分配一个字节
        strb r0,[r2]                    //将数据长度存放到表示数据长度的字节地址中
        b make_frame_going              //跳转到make_frame_going继续执行
//（4）为有效数据长度分配两个字节
make_frame_data2:
//（4.1）将有效数据长度的高八位存放到数据长度的第一个字节处的地址中
        lsr r0,#8
        strb r0,[r2]
        mov r0,r4                       //将有效数据长度放回r0
//（4.2）将有效数据长度的低八位存放到数据长度的第二个字节处的地址中
        add r2,#1                       //地址后移一位
        mov r5,#0x0f
        and r0,r0,r5                    //进行与运算获取有效数据长度的低八位
        strb r0,[r2]
        mov r3,#2                       //r3表示分配两个字节
```

```
//（5）设置循环，将有效数据逐个字节存入数据长度后
make_frame_going:
//（5.1）获取有效数据存入位置的首地址
        add r2,#1
        mov r0,#0                          //设置初始计数值为0
make_frame_loop:
//（5.2）判断当前计数是否等于字符串长度
        cmp r0,r4
        beq make_frame_tail                //相等则跳转拼接帧尾
//（5.3）当前字节数据存入帧中
        ldrb r6,[r1,r0]
        strb r6,[r2,r0]
//（5.4）继续循环
        add r0,#1
        b make_frame_loop
```

```
//（6）添加帧尾
make_frame_tail:
        add r2,r0                   //r2←帧尾的首地址
        mov r5,#35                  //r5←帧尾的第一个字符“#”
        strb r5,[r2]                //将帧尾的第一个字符“#”存入帧尾的首地址
        add r2,#1                   //地址后移一位
        strb r5,[r2]                //将帧尾的第二个字符“#”存入帧尾的第二个地址
        add r2,#1                   //地址后移一位
//（7）组帧完成，返回帧头首地址
make_frame_end:
        mov r0,r7                   //保存帧头首地址到r0
//（8）统计帧长度
        add r4,#4
        add r4,r3
        mov r1,r4                   //保存帧长度到r1
//（9）恢复现场
        pop {pc}
```

3. 主函数文件main.s中的工作

1）声明要发送的有效数据

在头部声明要发送的有效数据如“abc”，代码如下：

```
string_test:
        .asciz "abc"
```

2）调用相关函数，发送有效数据

调用步骤1中的获取有效数据长度内部函数，获取有效数据长度，代码如下：

```
ldr r0,=string_test                    //r0←字符串
bl get_data_length                     //调用get_data_length获取有效数据长度，存入r0
ldr r1,=string_test                    //r1←字符串
bl uart_make_frame                     //调用uart_make_frame进行组帧
ldr r2,=0x20002000                     //r2←帧的首地址
mov r0,#UARTA                          //r0←串口号
bl  uart_sendN                         //调用uart_sendN
```

4. 下载程序机器码到目标板，观察运行情况

类似例1，打开串口调试工具，单击“打开串口”，接收框中收到已封装成帧的数据，在十六进制显示框中“0x24 0x24”表示帧头、“0x03”表示有效数据长度、“0x61 0x 62 0x63”表示有效数据、“0x23 0x23”表示帧尾。

作业：1~5