串口 1（USART1）例程实验

实验平台：奋斗版STM32开发板MINI、V2、V2.1、V3，V5
实验内容：板子加电后，先向串口输出一串测试数据，然后在PC端的串口助手类软件上输入结束符为0x0d 0x0a的一串数据，发送到板子上，板子接收到该串会将该字串输出回PC端串口助手软件，该实验学习了USART1 中断程序的编制及控制流程。

预先需要掌握的知识

1 复用功能I/O和调试配置(AFIO)

为了优化外设数目，可以把一些复用功能重新映射到其他引脚上。设置复用重映射和调试I/O配置寄存器(AFIO_MAPR)(参见0节)实现引脚的重新映射。这时，复用功能不再映射到它们的原始分配上。

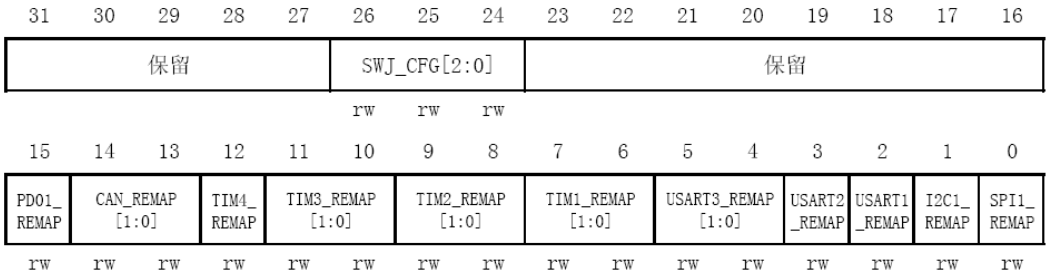
1.1 USART复用功能重映射

参见复用重映射和调试I/O配置寄存器(AFIO_MAPR)
表1 USART1 重映像

复用功能	USART1_REMAP = 0	USART1_REMAP = 1
USART1_TX	PA9	PB6
USART1_RX	PA10	PB7

1.2 复用重映射和调试I/O配置寄存器(AFIO_MAPR)

地址偏移：0x04
复位值：0x0000 0000



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留					SWJ_CFG[2:0]			保留							
					rw	rw	rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD01_REMAP	CAN_REMAP[1:0]		TIM4_REMAP	TIM3_REMAP[1:0]		TIM2_REMAP[1:0]		TIM1_REMAP[1:0]		USART3_REMAP[1:0]		USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:27	保留。
位26:24	SWJ_CFG[2:0]：串行线JTAG配置 这些位可由软件读写，用于配置SWJ和跟踪复用功能的I/O口。SWJ(串行线JTAG)支持JTAG或SWD访问Cortex的调试端口。系统复位后的默认状态是启用SWJ但没有跟踪功能，这种状态下可以通过JTMS/JTCK脚上的特定信号选择JTAG或SW(串行线)模式。 000：完全SWJ(JTAG-DP + SW-DP)：复位状态 001：完全SWJ(JTAG-DP + SW-DP)但没有JNTRST 010：关闭JTAG-DP，启用SW-DP 100：关闭JTAG-DP，关闭SW-DP 其它组合：禁用
位23:16	保留。
位15	PD01_REMAP：端口D0/端口D1映像到OSC_IN/OSC_OUT 该位可由软件读写，它控制PD0和PD1的GPIO功能映像。当不使用主振荡器HSE时(系统运行于内部的8MHz阻容振荡器)PD0和PD1可以映像到OSC_IN和OSC_OUT引脚。此功能只能适用于36、48和64管脚的封装(PD0和PD1出现在TQFP100的封装上，不必重映像)。 0：不进行PD0和PD1的重映像 1：PD0映像到OSC_IN，PD1映像到OSC_OUT。
位14:13	CAN_REMAP[1:0]：CAN复用功能重映像 这些位可由软件读写，控制复用功能CANRX和CANTX的重映像 00：CANRX映像到PA11，CANTX映像到PA12 01：未用组合 10：CANRX映像到PB8，CANTX映像到PB9(不能用于36脚的封装) 11：CANRX映像到PD0，CANTX映像到PD1(只适用于100脚的封装)
位12	TIM4_REMAP：定时器4的重映像 该位可由软件读写，只控制100脚封装中定时器4的通道1至4的映像。 0：没有重映像(TIM4_CH1/PB6，TIM4_CH2/PB7，TIM4_CH3/PB8，TIM4_CH4/PB9) 1：完全映像(TIM4_CH1/PD12，TIM4_CH2/PD13，TIM4_CH3/PD14，TIM4_CH4/PD15) 注：重映像不影响在PE0上的TIM4_ETR。

位11:10	TIM4_REMAP[1:0]：定时器3的重映像 这些位可由软件读写，控制定时器3的通道1至4在GPIO端口的映像。 00：没有重映像(CH1/PA6, CH2/PA7, CH3/PB0, CH4/PB1) 01：未用组合 10：部分映像(CH1/PB4, CH2/PB5, CH3/PB0, CH4/PB1) 11：完全映像(CH1/PC6, CH2/PC7, CH3/PC8, CH4/PC9) 注：重映像不影响在PD2上的TIM3_ETR。
位9:8	TIM2_REMAP[1:0]：定时器2的重映像 这些位可由软件读写，控制定时器2的通道1至4和外部触发(ETR)在GPIO端口的映像。 00：没有重映像(CH1/ETR/PA0, CH2/PA1, CH3/PA2, CH4/PA3) 01：部分映像(CH1/ETR/PA15, CH2/PB3, CH3/PA2, CH4/PA3) 10：部分映像(CH1/ETR/PA0, CH2/PA1, CH3/PB10, CH4/PB11) 11：完全映像(CH1/ETR/PA15, CH2/PB3, CH3/PB10, CH4/PB11)
位7:6	TIM1_REMAP[1:0]：定时器1的重映像 这些位可由软件读写，控制定时器1的通道1至4、1N至3N、外部触发(ETR)和断线输入(BKIN)在GPIO端口的映像。 00：没有重映像(ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PB12, CH1N/PB13, CH2N/PB14, CH3N/PB15) 01：部分映像(ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PA6, CH1N/PA7, CH2N/PB0, CH3N/PB1) 10: 未用组合 11: 完全映像(ETR/PE7, CH1/PE9, CH2/PE11, CH3/PE13, CH4/PE14, BKIN/PE15, CH1N/PE8, CH2N/PE10, CH3N/PE12)
位5:4	USART3_REMAP[1:0]：USART3的重映像 这些位可由软件读写，控制USART3的CTS、RTS、CK、TX和RX复用功能在GPIO端口的映像。 00: 没有重映像(TX/PB10, RX/PB11, CK/PB12, CTS/PB13, RTS/PB14) 01: 部分映像(TX/PC10, RX/PC11, CK/PC12, CTS/PB13, RTS/PB14) 10: 未用组合 11: 完全映像(TX/PD8, RX/PD9, CK/PD10, CTS/PD11, RTS/PD12)
位3	USART2_REMAP：USART2的重映像 该位可由软件读写，控制USART2的CTS、RTS、CK、TX和RX复用功能在GPIO端口的映像。 0: 没有重映像(CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4) 1: 重映像(CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)
位2	USART1_REMAP：USART1的重映像 该位可由软件读写，控制USART1的TX和RX复用功能在GPIO端口的映像。 0: 没有重映像(TX/PA9, RX/PA10) 1: 重映像(TX/PB6, RX/PB7)
位1	I2C1_REMAP：I2C1的重映像 该位可由软件读写，控制I2C1的SCL和SDA复用功能在GPIO端口的映像。 0: 没有重映像(SCL/PB6, SDA/PB7) 1: 重映像(SCL/PB8, SDA/PB9)

位0	SPI1_REMAP：SPI1的重映像 该位可由软件读写，控制SPI1的NSS、SCK、MISO和MOSI复用功能在GPIO端口的映像。 0: 没有重映像(NSS/PA4, SCK/PA5, MISO/PA6, MOSI/PA7) 1: 重映像(NSS/PA15, SCK/PB3, MISO/PB4, MOSI/PB5)

2.1 嵌套向量中断控制器（NVIC）

特性

- 43 个可屏蔽中断通道（不包含16 个Cortex-M3 的中断线）；
- 16 个可编程的优先等级；
- 低延迟的异常和中断处理；
- 电源管理控制；
- 系统控制寄存器的实现；

嵌套向量中断控制器(NVIC)和处理器核的接口紧密相连，可以实现低延迟的中断处理和有效处理地处理晚到的中断。

嵌套向量中断控制器管理着包括核异常等中断。关于更多的异常和NVIC编程的说明请参考ARM《Cortex-M3TM技术参考手册》的第5章的异常和第8章的嵌套向量中断控制器。

2.1.1 系统嘀嗒(SysTick)校准值寄存器

系统嘀嗒校准值固定到9000，当系统嘀嗒时钟设定为9兆赫，产生1ms时基。

2.1.2 中断和异常向量

表1 向量表

位置	优先级	优先级类型	名称	说明	地址
	-	-	-	保留	0x0000_0000
	-3	固定	Reset	复位	0x0000_0004
	-2	固定	NMI	不可屏蔽中断 RCC时钟安全系统(CSS)联接到NMI向量	0x0000_0008
	-1	固定	硬件失效	所有类型的失效	0x0000_000C
	0	可设置	存储管理	存储器管理	0x0000_0010
	1	可设置	总线错误	预取指失败，存储器访问失败	0x0000_0014
	2	可设置	错误应用	未定义的指令或非法状态	0x0000_0018
	-	-	-	保留	0x0000_001C ~0x0000_002B
	3	可设置	SVCall	通过SWI指令的系统服务调用	0x0000_002C
	4	可设置	调试监控	调试监控器	0x0000_0030
	-	-	-	保留	0x0000_0034
	5	可设置	PendSV	可挂起的系统服务	0x0000_0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000_003C
0	7	可设置	WWDG	窗口定时器中断	0x0000_0040

1	8	可设置	PVD	联到EXTI的电源电压检测(PVD)中断	0x0000_0044
2	9	可设置	TAMPER	侵入检测中断	0x0000_0048
3	10	可设置	RTC	实时时钟(RTC)全局中断	0x0000_004C
4	11	可设置	FLASH	闪存全局中断	0x0000_0050
5	12	可设置	RCC	复位和时钟控制(RCC)中断	0x0000_0054
6	13	可设置	EXTI0	EXTI线0中断	0x0000_0058
7	14	可设置	EXTI1	EXTI线1中断	0x0000_005C
8	15	可设置	EXTI2	EXTI线2中断	0x0000_0060
9	16	可设置	EXTI3	EXTI线3中断	0x0000_0064
10	17	可设置	EXTI4	EXTI线4中断	0x0000_0068
11	18	可设置	DMA通道1	DMA通道1全局中断	0x0000_006C
12	19	可设置	DMA通道2	DMA通道2全局中断	0x0000_0070
13	20	可设置	DMA通道3	DMA通道3全局中断	0x0000_0074
14	21	可设置	DMA通道4	DMA通道4全局中断	0x0000_0078
15	22	可设置	DMA通道5	DMA通道5全局中断	0x0000_007C
16	23	可设置	DMA通道6	DMA通道6全局中断	0x0000_0080
17	24	可设置	DMA通道7	DMA通道7全局中断	0x0000_0084
18	25	可设置	ADC	ADC全局中断	0x0000_0088
19	26	可设置	USB_HP_CAN_TX	USB高优先级或CAN发送中断	0x0000_008C
20	27	可设置	USB_LP_CAN_RX0	USB低优先级或CAN接收0中断	0x0000_0090
21	28	可设置	CAN_RX1	CAN接收1中断	0x0000_0094
22	29	可设置	CAN_SCE	CAN SCE中断	0x0000_0098
23	30	可设置	EXTI9_5	EXTI线[9:5]中断	0x0000_009C
24	31	可设置	TIM1_BRK	TIM1断开中断	0x0000_00A0
25	32	可设置	TIM1_UP	TIM1更新中断	0x0000_00A4
26	33	可设置	TIM1_TRG_COM	TIM1触发和通信中断	0x0000_00A8
27	34	可设置	TIM1_CC	TIM1捕获比较中断	0x0000_00AC
28	35	可设置	TIM2	TIM2全局中断	0x0000_00B0
29	36	可设置	TIM3	TIM3全局中断	0x0000_00B4
30	37	可设置	TIM4	TIM4全局中断	0x0000_00B8
31	38	可设置	I2C1_EV	I^2C1事件中断	0x0000_00BC
32	39	可设置	I2C1_ER	I^2C1错误中断	0x0000_00C0
33	40	可设置	I2C2_EV	I^2C2事件中断	0x0000_00C4
34	41	可设置	I2C2_ER	I^2C2错误中断	0x0000_00C8
35	42	可设置	SPI1	SPI1全局中断	0x0000_00CC
36	43	可设置	SPI2	SPI2全局中断	0x0000_00D0
37	44	可设置	USART1	USART1全局中断	0x0000_00D4

38	45	可设置	USART2	USART2全局中断	0x0000_00D8
39	46	可设置	USART3	USART3全局中断	0x0000_00DC
40	47	可设置	EXTI15_10	EXTI线[15:10]中断	0x0000_00E0
41	48	可设置	RTCAlarm	联到EXTI的RTC闹钟中断	0x0000_00E4
42	49	可设置	USB唤醒	联到EXTI的从USB待机唤醒中断	0x0000_00E8

3 USART 通用同步异步收发器(USART)

3.1 介绍

通用同步异步收发器（USART）提供了一种灵活的方法来与使用工业标准NRZ异步串行数据格式的外部设备之间进行全双工数据交换。USART 利用分数波特率发生器提供宽范围的波特率选择。它支持同步单向通信和半双工单线通信。它也支持LIN(局部互连网)，智能卡协议和IrDA(红外数据组织)SIR ENDEC 规范，以及调制解调器(CTS/RTS)操作。它还允许多处理器通信。使用多缓冲器配置的DMA 方式，可以实现高速数据通信。

3.2 主要特性

- 全双工的，异步通信
- NRZ 标准格式
- 分数波特率发生器系统
  - 发送和接收共用的可编程波特率，最高到4.5Mbits/s
- 可编程数据字长度（8 位或9 位）
- 可配置的停止位-支持1 或2 个停止位
- LIN 主发送同步断开符的能力以及LIN 从检测断开符的能力
  - 当USART 硬件配置成LIN 时，生成13 位断开符；检测10/11 位断开符
- 发送方为同步传输提供时钟
- IRDA SIR 编码器解码器
  - 在正常模式下支持3/16 位的持续时间
- 智能卡模拟功能
  - 智能卡接口支持ISO7816-3 标准里定义的异步协议智能卡
  - 智能卡用到的0.5 和1.5 个停止位
- 单线半双工通信
- 可配置的使用DMA 的多缓冲器通信
  - 在SRAM 里利用集中式DMA 缓冲接收/发送字节
- 单独的发送器和接收器使能位
- 检测标志
  - 接收缓冲器满
  - 发送缓冲器空
  - 传输结束标志
- 校验控制
  - 发送校验位
  - 对接收数据进行校验

● 四个错误检测标志
– 溢出错误
– 噪音错误
– 帧错误
– 校验错误
● 10 个带标志的中断源
– CTS 改变
– LIN 断开符检测
– 发送数据寄存器空
– 发送完成
– 接收数据寄存器满
– 检测到总线为空闲
– 溢出错误
– 帧错误
– 噪音错误
– 校验错误
● 多处理器通信 -- 如果地址不匹配，则进入静默模式
● 从静默模式中唤醒（通过空闲总线检测或地址标志检测）
● 两种唤醒接收器的方式：地址位(MSB，第9 位)，总线空闲

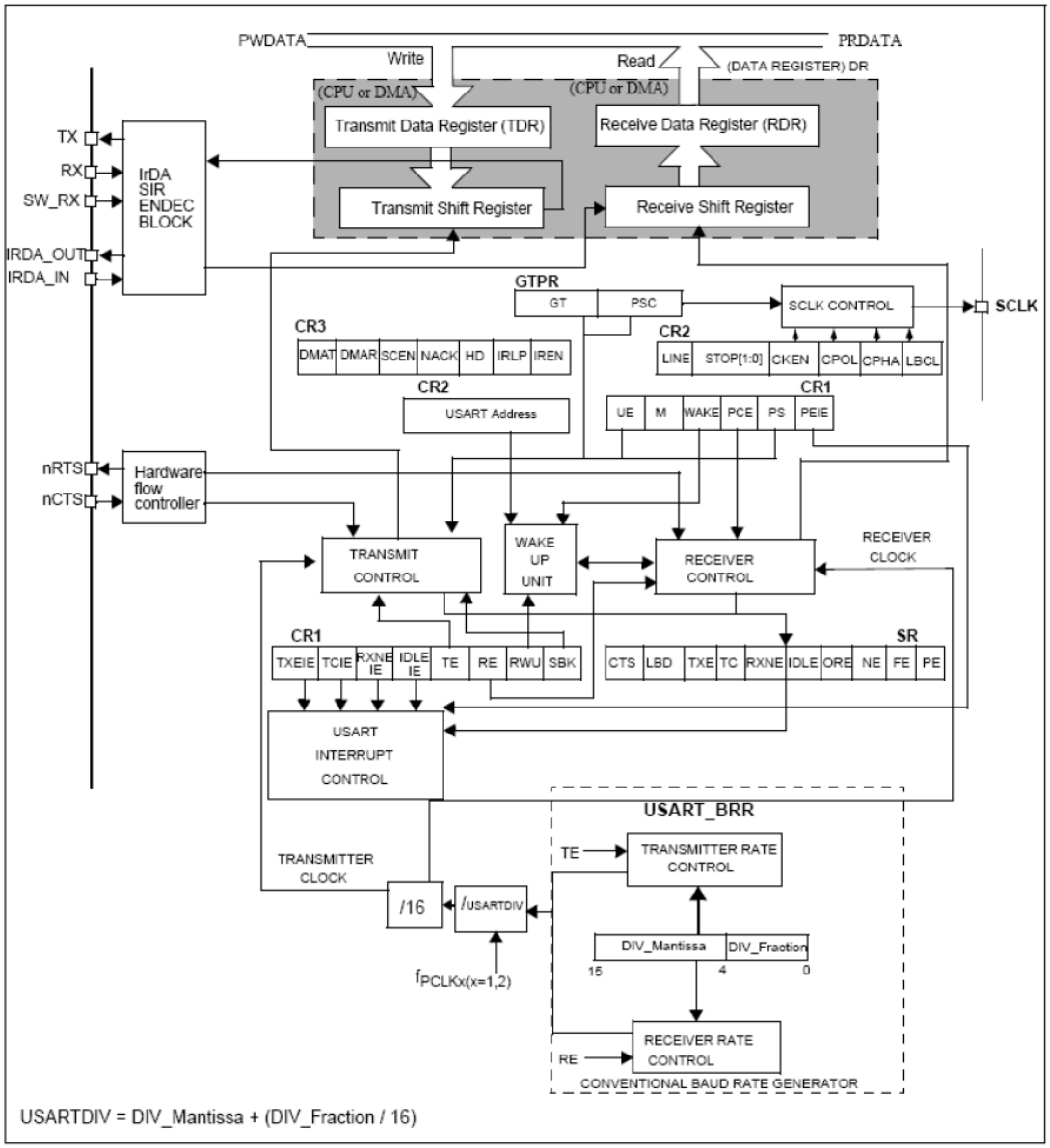
3.3 概述

接口通过三个引脚与其他设备连接在一起(见图1)。任何USART双向通信至少需要两个脚：接收数据输入(RX)和发送数据输出(TX)。
RX：接收数据串行输。通过过采样技术来区别数据和噪音，从而恢复数据。
TX：发送数据输出。当发送器被禁止时，输出引脚恢复到它的I/O 端口配置。当发送器被激活，并且没东西发送时，TX 引脚处于高电平。在单线和智能卡模式里，此I/O 口被用来发送和接收数据。
● 总线在发送或接收前应处于空闲状态
● 一个起始位
● 一个数据字（8 或9 位），最低有效位在前
● 0.5，1.5，2 个的停止位，由此表明数据帧的结束
● 使用分数波特率发生器 —— 12 位整数和4 位小数的表示方法。
● 一个状态寄存器(USART_SR)
● 数据寄存器(USART_DR)
● 一个波特率寄存器(USART_BRR)，12 位的整数和4 位小数
● 一个智能卡模式下的保护时间寄存器(USART_GTPR)
关于以上寄存器中每个位的具体定义，请参考寄存器描述。
在同步模式中需要下列引脚：
SCLK：发送器时钟输出。此引脚输出用于同步传输的 时钟， (在Start 位和Stop位上没有时钟脉冲，软件可选地，可以在最后一个数据位送出一个时钟脉冲)。数据可以在RX 上同步被接收。这可以用来控制带有移位寄存器的外部设备(例如LCD 驱动器)。时钟相位和极性都是软件可编程的。在智能卡模式里，SCLK 可以为智能卡提供时钟。
在IrDA 模式里需要下列引脚：
IrDA_RDI：IrDA 模式下的数据输入。

IrDA_TDO：IrDA 模式下的数据输出。
下列引脚在硬件流控模式中需要：
nCTS：清除发送，若是高电平，在当前数据传输结束时阻断下一次的数据发送。
nRTS：发送请求，若是低电平，表明USART 准备好接收数据

3.3.1 框图

图1 USART 框图

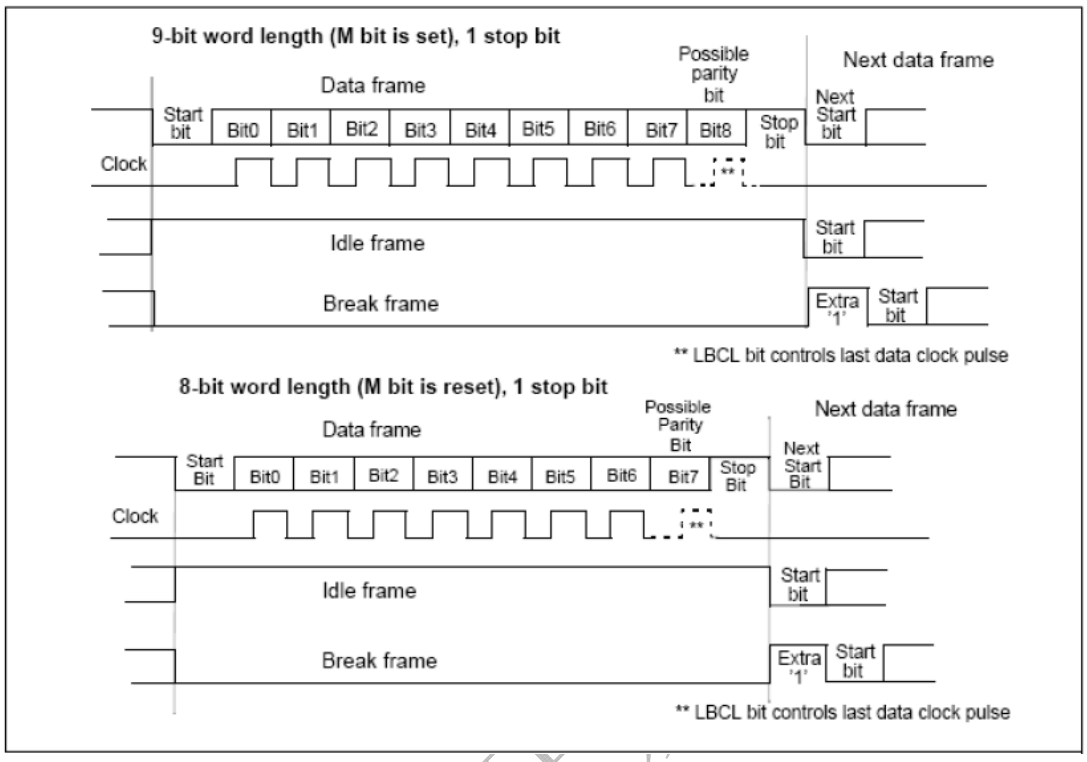


3.3.2 USART 特征描述

字长可以通过编程USART_CR1 寄存器中的M位，选择成8 或9 位(见图2)。在起始位期间，TX 脚处于低电平，在停止位期间处于高电平。空闲符号被视为完全由“1 " 组成的一个完整的数据帧，后面跟着包含了数据的下一帧的开始位。断开符号 被视为在一个帧周期内全部收到“0 "(包括停止位期间，也是“0 ")。在断开帧结束时，发送器再插入1 或2 个停止位来

应答起始位。发送和接收由一共用的波特率发生器驱动，当发送器和接收器的使能位分别置位时，分别为其产生时钟。每个功能块的详细资料如下给出。

图2 字长设置



3.3.3 发送器

发送器根据M 位的状态发送8 位或9 位的数据字。当发送使能位（TE）被设置时，发送移位寄存器中的数据在TX 脚上输出，相应的时钟脉冲在SCLK 脚上输出。

字符发送

在USART发送期间，在TX引脚上首先移出数据的最低有效位。在此模式里，USART_DR寄存器包含了一个内部总线和发送移位寄存器之间的缓冲器(见图3)。每个字符之前都有一个低电平的起始位；之后跟着的停止位，其数目可配置。

注意： *1*. 在数据传输期间不能复位*TE* 位，否则将破坏*TX* 脚上的数据，因为波特率计数器停止计数。正在传输的当前数据将丢失。

2. *TE* 位被激活后将发送一个空闲帧。

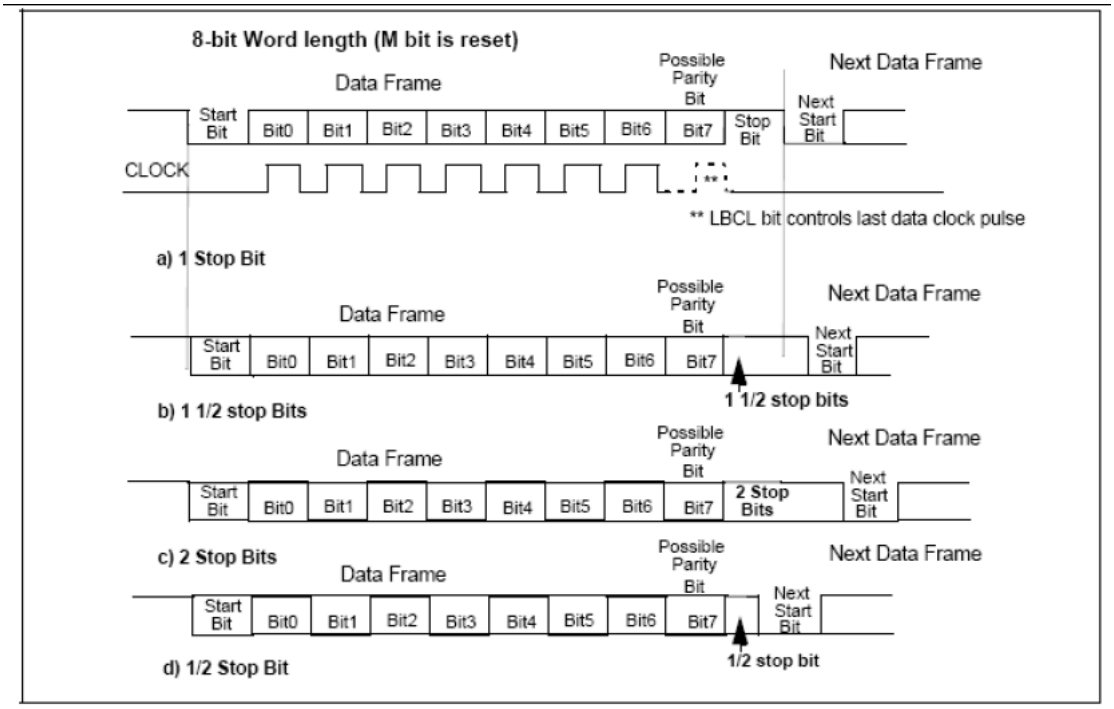
可配置的停止位

随每个字符发送的停止位的位数可以通过控制寄存器2 的位13、12 进行编程。

1. 1个停止位：停止位位数的默认值。
2. 2个停止位：可用于常规USART模式、单线模式以及调制解调器模式。
3. 0.5个停止位：在智能卡模式下接收数据时使用。
4. 1.5个停止位：在智能卡模式下发送数据时使用。

空闲帧包括了停止位。断开帧是10 位低电平，后跟停止位（当m=0 时）；或者11 位低电平，后跟停止位（m=1 时）。不可能传输更长的断开帧（长度大于10 或者11 位）。

图3 配置停止位



配置步骤：

1. 通过在USART_CR1寄存器上置位UE位来激活USART
2. 编程USART_CR1的M位来定义字长。
3. 在USART_CR2中编程停止位的位数。
4. 如果采用多缓冲器通信，配置USART_CR3中的DMA使能位(DMAT)。按多缓冲器通信中的描述配置DMA寄存器。
5. 设置USART_CR1中的TE位，发送一个空闲帧作为第一次数据发送。
6. 利用USART_BRR寄存器选择要求的波特率。
7. 把要发送的数据写进USART_DR寄存器(此动作清除TXE位)。在只有一个缓冲器的情况下，对每个待发送的数据重复步骤7。

单字节通信

清零TXE 位总是通过对数据寄存器的写操作来完成的。TXE 位由硬件来设置，它表明：

- 数据已经从TDR 移送到移位寄存器，数据发送已经开始
- TDR 寄存器被清空
- 下一个数据可以被写进USART_DR 寄存器而不会覆盖先前的数据如果TXEIE 位被设置，此标志将产生一个中断。如果此时USART 正在发送数据，对USART_DR 寄存器的写操作把数据存进TDR 寄存器，并在当前传输结束时把该数据复制进移位寄存器。如果此时USART 没有在发送数据，处于空闲状态，对USART_DR 寄存器的写操作直接把数据放进移位寄存器，数据传输开始，TXE 位立即被置起。当一帧发送完成时（停止位发送后），TC 位被置起，并且如果USART_CR1 寄存器中的TCIE 位被置起时，中断产生。先读一下USART_SR 寄存器，再写一下USART_DR 寄存器，可以完成对TC 位的清零。

注意： *TC* 位也可以通过对它软件写*0* 来清除。此清零方式只在多缓冲器通信模式下推荐使用。

断开符号

置位SBK位可发送一个断开符号。断开帧长度取决M位(见图166)。如果SBK位被置1，在完成当前数据发送后，将在TX线上发送一个断开符号。断开字符发送完成时（在断开符号的停止位时）SBK被硬件复位。USART在最后一个断开帧的结束处插入一逻辑1 位，以保证能识别下一帧的起始位。
注意：如果在开始发送断开帧之前，软件又复位了SBK 位，断开符号将不被发送。如果要发送两个连续的断开帧，SBK 位应该在前一个断开符号的停止位之后置起。

空闲符号

置位TE 将使得USART 在第一个数据帧前发送一空闲帧。

3.3.4 接收器

USART 可以根据USART_CR1 的M 位接收8 位或9 位的数据字

字符接收

在USART 接收期间，数据的最低有效位首先从RX 脚移进。在此模式里，USART_DR 寄存器包含的缓冲器位于内部总线和接收移位寄存器之间。
配置步骤：
1. 将USART_CR1寄存器的UE置1来激活USART。
2. 编程USART_CR1的M位定义字长
3. 在USART_CR2中编写停止位的个数
4. 如果需多缓冲器通信，选择USART_CR3中的DMA使能位（DMAT）。按多缓冲器通信所要求的配置DMA寄存器。
5. 利用波特率寄存器USART_BRR选择希望的波特率。
6. 设置USART_CR1的RE位。激活接收器，使它开始寻找起始位。

当一字符被接收到时，
● RXNE 位被置位。它表明移位寄存器的内容被转移到RDR。换句话说，数据已经被接收并且可以被读出（包括与之有关的错误标志）。
● 如果RXNEIE 位被设置，产生中断。
● 在接收期间如果检测到帧错误，噪音或溢出错误，错误标志将被置起，
● 在多缓冲器通信时，RXNE 在每个字节接收后被置起，并由DMA 对数据寄存器的读操作而清零。
● 在单缓冲器模式里，由软件读USART_DR 寄存器完成对RXNE 位清除。RXNE 标志也可以通过对它写0 来清除。RXNE 位必须在下一字符接收结束前被清零，以避免溢出错误。
注意：在接收数据时，*RE* 位不应该被复位。如果*RE* 位在接收时被清零，当前字节的接收被丢失。

断开符号

当接收到一个断开帧时，USART 像处理帧错误一样处理它。

空闲符号

当一空闲帧被检测到时，其处理步骤和接收到普通数据帧一样，但如果IDLEIE位被设置将产生一个中断。

溢出错误

如果RXNE 还没有被复位，又接收到一个字符，则发生溢出错误。数据只有当RXNE 位被清零后才能从移位寄存器转移到RDR 寄存器。RXNE 标记是接收到每个字节后被置位的。如果下一个数据已被收到或先前DMA 请求还没被服务时，RXNE 标志仍是置起的，溢出错误产生。当溢出错误产生时：

- ORE 位被置位。
- RDR 内容将不会丢失。读USART_DR 寄存器仍能得到先前的数据。
- 移位寄存器中以前的内容将被覆盖。随后接收到的数据都将丢失。
- 如果RXNEIE 位被设置或EIE 和DMAR 位都被设置，中断产生。
- 顺序执行对USART_SR 和USART_DR 寄存器的读操作，可复位ORE 位注意： 当*ORE* 位置位时，表明至少有*1* 个数据已经丢失。有两种可能性：
- 如果RXNE=1，上一个有效数据还在接收寄存器RDR 上，可以被读出。
- 如果RXNE=0，这意味着上一个有效数据已经被读走，RDR 已经没有东西可读。当上一个有效数据在RDR 中被读取的同时又接收到新的（也就是丢失的）数据时，此种情况可能发生。在读序列期间（在USART_SR 寄存器读访问和USART_DR 读访问之间）接收到新的数据，此种情况也可能发生。

噪音错误

使用过采样技术（同步模式除外），通过区别有效输入数据和噪音来进行数据恢复。

图4 检测噪声的数据采样

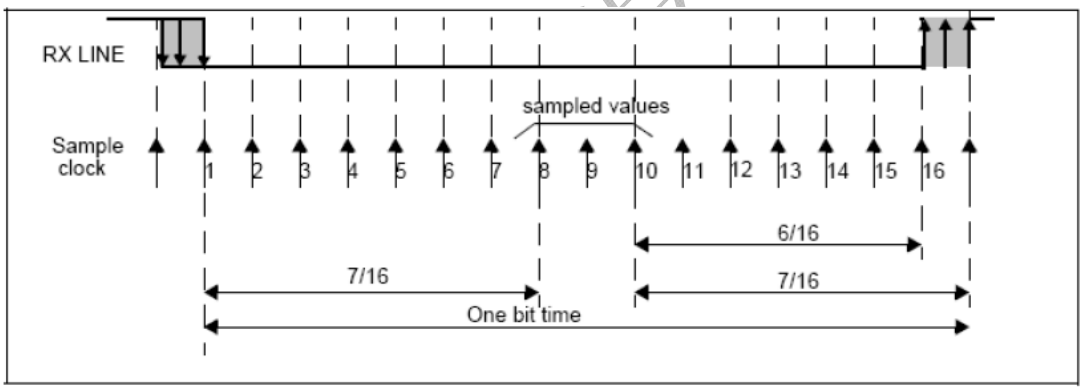


表2 检测噪声的数据采样

采样值	NE状态	接收的位值	数据有效性
000	0	0	有效
001	1	0	无效
010	1	0	无效
011	1	1	无效
100	1	0	无效
101	1	1	无效

110	1	1	无效
111	0	1	有效

当在接收帧中检测到噪音时：

- NE 在RXNE 位的上升沿被置起。
- 无效数据从移位寄存器移送到USART_DR 寄存器。
- 在单个字节通信情况下，没有中断产生。然而，NE 这个位和RXNE 位同时置起，后者自己产生中断。在多缓冲器通信情况下，如果USART_CR3 寄存器中EIE 位被置位的话，将产生一中断顺序执行对USART_SR 和USART_DR 寄存器的读操作，可复位NE 位

帧错误

当以下情况发生时检测到帧错误：

由于没有同步上或大量噪音的原因，停止位没有在预期的时间上接和收识别出来。

当帧错误被检测到时：

- FE 位被硬件置起
- 无效数据从移位寄存器传送到USART_DR 寄存器。
- 在单个字节通信情况下，没有中断产生。然而，这个位和RXNE 位同时置起，后者自己产生中断。在多缓冲器通信情况下，如果USART_CR3 寄存器中EIE 位被置位的话，将产生一中断。顺序执行对USART_SR 和USART_DR 寄存器的读操作，可复位FE 位。

接收期间的可配置的停止位

被接收的停止位的个数可以通过控制寄存器2 的控制位来配置，在正常模式时，可以是1 或2 个，在智能卡模式里可能是0.5 或1.5 个。

1. 0.5个停止位（智能卡模式里的接收）：不对0.5个停止位进行采样。因此，如果选择0.5个停止位则不能检测帧错误和断开帧。
2. 1个停止位：对1个停止位的采样在第8，第9和第10采样点上进行。
3. 1.5 个停止位(智能卡模式里的发送)：当以智能卡模式发送时，器件必须检查数据是否被正确的发送出去。所以接收器功能块必须被激活(USART_CR1寄存器中的RE =1)，并且在停止位的发送期间采样数据线上的信号。如果出现校验错误，智能卡会在发送方采样NACK信号时，即总线上停止位对应的时间内时，拉低数据线，以此表示出现了帧错误。FE在1.5个停止位结束时和RXNE一起被置起。对1.5个停止位的采样是在第16，第17和第18采样点进行的。1.5个的停止位可以被分成2部分：一个是0.5个时钟周期，期间不做任何事情 。随后是1个时钟周期的停止位，在这段时间的中点处采样。参考19.3.11智能卡，以得到更多详细资料。
4. 2个停止位：对2个停止位的采样是在第一停止位的第8，第9和第10个采样点完成的。如果第一个停止位期间检测到一个帧错误，帧错误标志将被设置。第二个停止位不再检查帧错误。在第一个停止位结束时RXNE标志将被设置。

3.3.5 分数波特率的产生

接收器和发送器(Rx 和Tx)的波特率在USARTDIV 的整数和小数寄存器中的值应设置成相同。

$$\text{Tx / Rx 波特率} = \frac{f_{PCLKx}}{(16 * USARTDIV)}$$

这里的f_{PCLKx}(x=1 、2) 是给外设的时钟 (PCLK1 用于USART2 、3 ， PCLK2用于USART1)

USARTDIV 是一个无符号的定点数。这12 位的值设置在USART_BRR 寄存器。

如何从USART_BRR寄存器值得到USARTDIV

例1：如果 DIV_Mantissa = 27d ， DIV_Fraction = 12d (USART_BRR=1BCh),于是Mantissa (USARTDIV) = 27d Fraction (USARTDIV) = 12/16 = 0.75d 所以 USARTDIV = 27.75d

例2：要求 USARTDIV = 25.62d,

就有：

DIV_Fraction = 16*0.62d = 9.92d, 近似等于10d = 0x0A

DIV_Mantissa = mantissa (25.620d) = 25d = 0x19

于是, USART_BRR = 0x19A

例3：要求 USARTDIV = 50.99d

就有：

DIV_Fraction = 16*0.99d = 15.84d =>近似等于16d = 0x10

DIV_Mantissa = mantissa (50.990d) = 50d = 0x32

注意： 更新波特率寄存器USART_BRR 后，波特率计数器中的值也立刻随之更新。所以在通信进行时不应改变USART_BRR 中的值。

表3 设置波特率时的误差计算

波特率		f_{PCLK} = 36MHz			f_{PCLK} = 72MHz		
序号	Kbps	实际	置于波特率寄存器中的值	误差%	实际	置于波特率寄存器中的值	误差%
1	2.4	2.400	937.5	0%	2.4	1875	0%
2	9.6	9.600	234.375	0%	9.6	468.75	0%
3	19.2	19.2	117.1875	0%	19.2	234.375	0%
4	57.6	57.6	39.0625	0%	57.6	78.125	0%
5	115.2	115.384	19.5	0.15%	115.2	39.0625	0%
6	230.4	230.769	9.75	0.16%	230.769	19.5	0.16%
7	460.8	461.538	4.875	0.16%	461.538	9.75	0.16%
8	921.6	923.076	2.4375	0.16%	923.076	4.875	0.16%
9	2250	2250	1	0%	2250	2	0%
10	4500	不可能	不可能	不可能	4500	1	0%

注： 1. CPU 的时钟频率越低某一特定波特率的误差也越低

2. 只有 USART1 使用 PCLK2(最高 72MHz)。其它 USART 使用 PCLK1(最高 36MHz)。

3.3.6 多处理器通信

通过USART 可以实现多处理器通信（将几个USART 连在一个网络里）。例如某个USART 设备可以是主，它的TX 输出和其他USART 从设备的RX 输入相连接；USART 从设备各自的TX 输出逻辑地与在一起，并且和主设备的RX 输入相连接。在多处理器配置中，我们通常希望只有被寻址的接收者才被激活，来接收随后的数据，这样就可以减少由未被寻址的接收器的参与带来的多余的USART 服务开销。

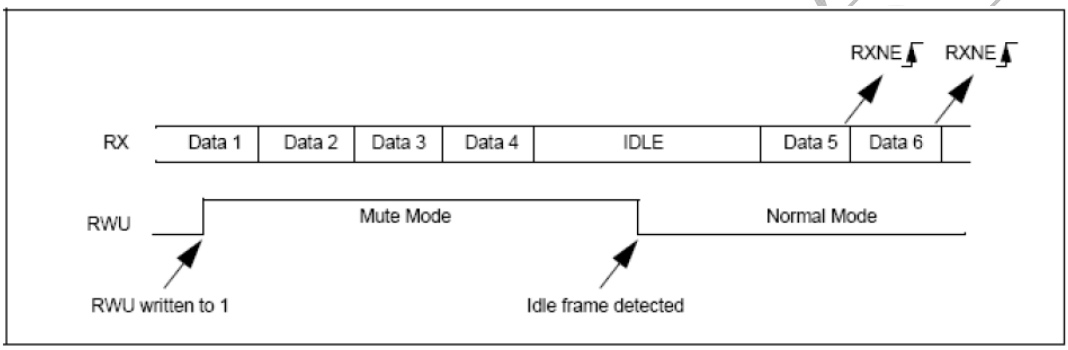
未被寻址的设备可启用其静默功能置于静默模式。在静默模式里：

- 任何接收状态位都不会被设置。
- 所有接收中断被禁止。
- USART_CR1 寄存器中的RWU 位被置1。RWU 可以被硬件自动控制或在某个条件下由软件写。根据USART_CR1 寄存器中的WAKE 位状态，USART 可以用二种方法进入或退出静默模式。
- 如果WAKE 位被复位：进行空闲总线检测。
- 如果WAKE 位被设置：进行地址标记检测。

空闲总线检测(WAKE=0)

当RWU位被写1 时，USART进入静默模式。当检测到一空闲帧时，它被唤醒。然后RWU被硬件清零，但是USART_SR寄存器中的IDLE位并不置起。RWU还可以被软件写0。图169 给出利用空闲总线检测来唤醒和进入静默模式的一个例子

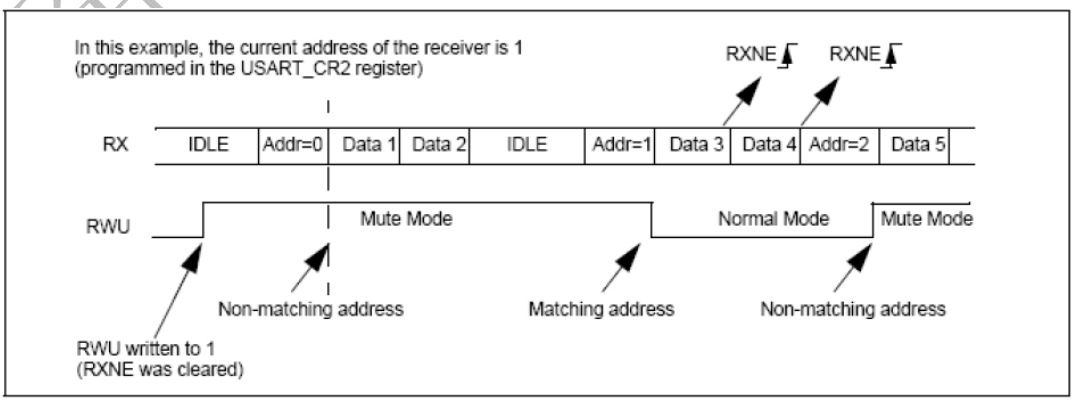
图5 利用空闲总线检测的静默模式



地址标记（address mark）检测(WAKE=1)

在这个模式里，如果MSB 是1，该字节被认为是地址，否则被认为是数据。在一个地址字节中，目标接收器的地址被放在4 个LSB 中。这个4 位地址被接收器同它自己地址做比较，接收器的地址被编程在USART_CR2 寄存器的ADD。如果接收到的字节与它的编程地址不匹配时，USART 进入静默模式。该字节的接收既不会置起RXNE 标志也不会产生中断或发出DMA 请求，因为USART 已经在静默模式。当接收到的字节与接收器内编程地址匹配时，USART 退出静默模式。然后RWU位被清零，随后的字节被正常接收。匹配的地址字节将置位RXNE 位，因为RWU 位已被清零。当接收缓冲器不包含数据时（USART_SR的RXNE=0），RWU位可以被写0 或1。否则，该次写操作被忽略。图170 给出利用地址标记检测来唤醒和进入静默模式的例子。

图6 利用地址标记检测的静默模式



3.3.7 校验控制

奇偶控制（发送时生成一个奇偶位，接收时进行奇偶校验）可以通过设置USART_CR1 寄存器上的PCE位而激活。根据M位定义的帧长度，可能的USART帧格式列在表4 中。

表4 帧格式

M位	PCE位	USART帧
0	0	| 起始位 | 8位数据 | 停止位 |
0	1	| 起始位 | 7位数据 | 奇偶检验位 | 停止位 |
1	0	| 起始位 | 9位数据 | 停止位 |
1	1	| 起始位 | 8位数据 | 奇偶检验位 | 停止位 |

注意： 在用地址标记唤醒设备时，地址的匹配只考虑到数据的*MSB* 位，而不用关心校验位。（*MSB* 是数据位中最后发出的，后面紧跟校验位或者停止位）偶校验：校验位使得一帧中的7 或8 个LSB 数据以及校验位中“1 ”的个数为偶数。

例如：data=00110101，有4 个“1 ”，如果选择偶校验（在USART_CR1 中的PS＝0），校验位将是0。奇校验：此校验位使得一帧中的7 或8 个LSB 数据以及校验位中“1 ”的个数为奇数。例如：data=00110101，有4 个“1 ”，如果选择奇校验（在USART_CR1 中的PS＝1），校验位将是1。

传输模式：如果USART_CR1 的PCE 位被置位，写进数据寄存器的数据的MSB位被校验位替换后发送出去（如果选择偶校验偶数个1，如果选择奇校验奇数个1）。如果奇偶校验失败，USART_SR 寄存器中的PE 标志被置1，并且如果USART_CR1 寄存器的PEIE 在被预先设置的话，中断产生。

3.3.8 USART 同步模式

通过在USART_CR2 寄存器上写CLKEN 位选择同步模式在同步模式里，下列位必须保持清零状态：

- USART_CR2 寄存器中的LINEN 位
- USART_CR3 寄存器中的SCEN,HDSEL 和IREN 位

USART允许用户以主模式方式控制双向同步串行通信。SCLK脚是USART发送器时钟的输出。在起始位和停止位期间，SCLK脚上没有时钟脉冲。根据USART_CR2 寄存器中LBCL位的状态，决定在最后一个有效数据位期间产生或不产生时钟脉冲。USART_CR2 寄存器的CPOL位允许用户选择时钟极性，USART_CR2 寄存器上的CPHA位允许用户选择外部时钟的相位(见图8，图9 和图10)。在总线空闲期间，实际数据到来之前以及发送断开符号的时候，外部SCLK 时钟不被激活。同步模式时，USART 发送器和异步模式里工作一模一样。但是因为SCLK 是与TX 同步的(根据CPOL 和CPHA)，所以TX 上的数据是随SCLK 同步发出的。同步模式的USART 接收器工作方式与异步模式不同。如果RE=1，数据在SCLK上采样（根据CPOL 和CPHA 决定在上升沿还是下降沿），不需要任何的过采样。但必须考虑建立时间和持续时间（取决于波特率，1/16位时间）。

注意：*1 ． SCLK* 脚同*TX* 脚一起联合工作。因而，只有在发送器被激活*(TE ＝1)* ，且数据被发送时*(USART_DR* 寄存器被写入）才提供时钟。这意味着在没有发送数据时是不可能接收一个同步数据的。

2 ． LBCL, CPOL 和*CPHA* 位的正确配置，应该在发送器和接收器都被禁止时；当发送器或接收器被激活时，这些位不能被改变

3 ． 建议在同一条指令中设置*TE* 和*RE* ，以减少接收器的建立时间和保持时间。

4． *USART* 只支持主模式：它不能用来自其他设备的输入时钟接收或发送数据*(SCLK* 永远是输出*)*。

图7 USART 同步传输的例子

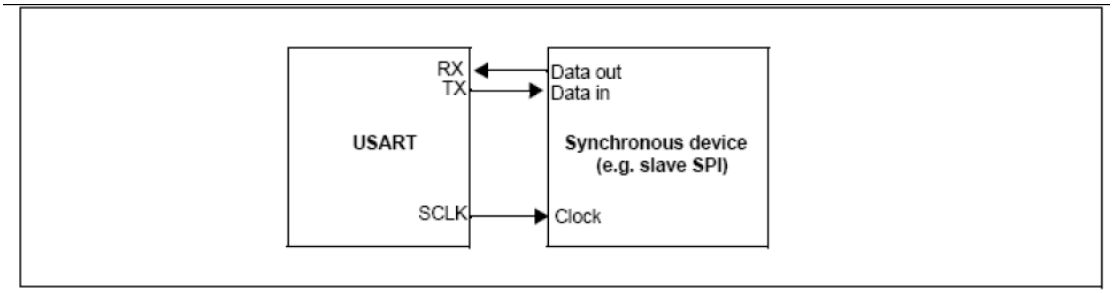


图8 USART 数据时钟时序示例(M=0)

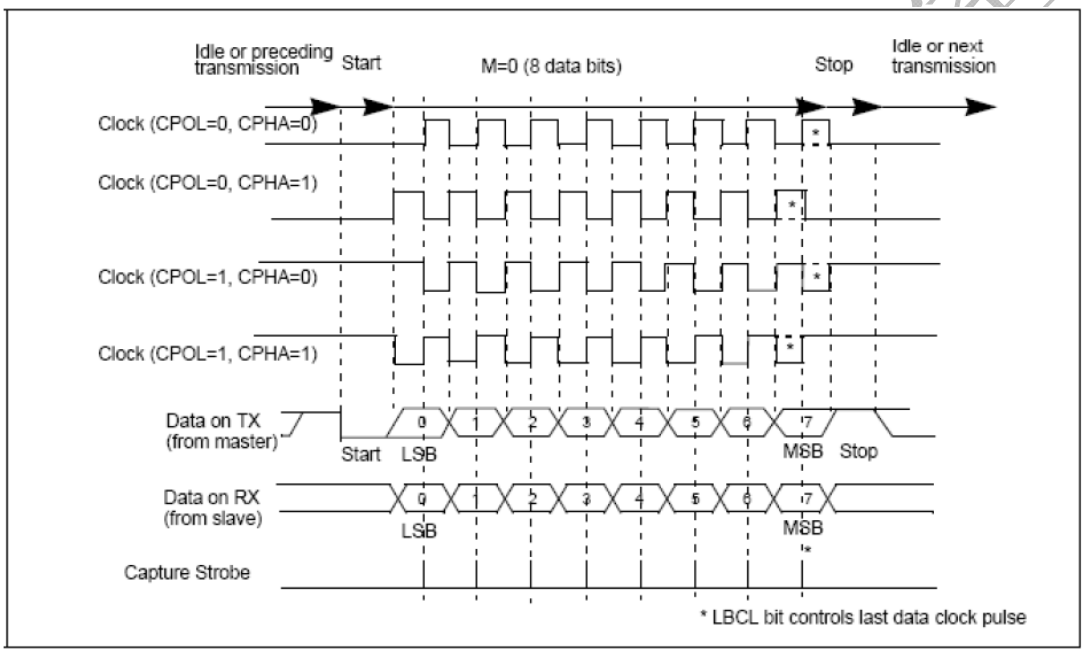


图9 USART 数据时钟时序示例(M=1)

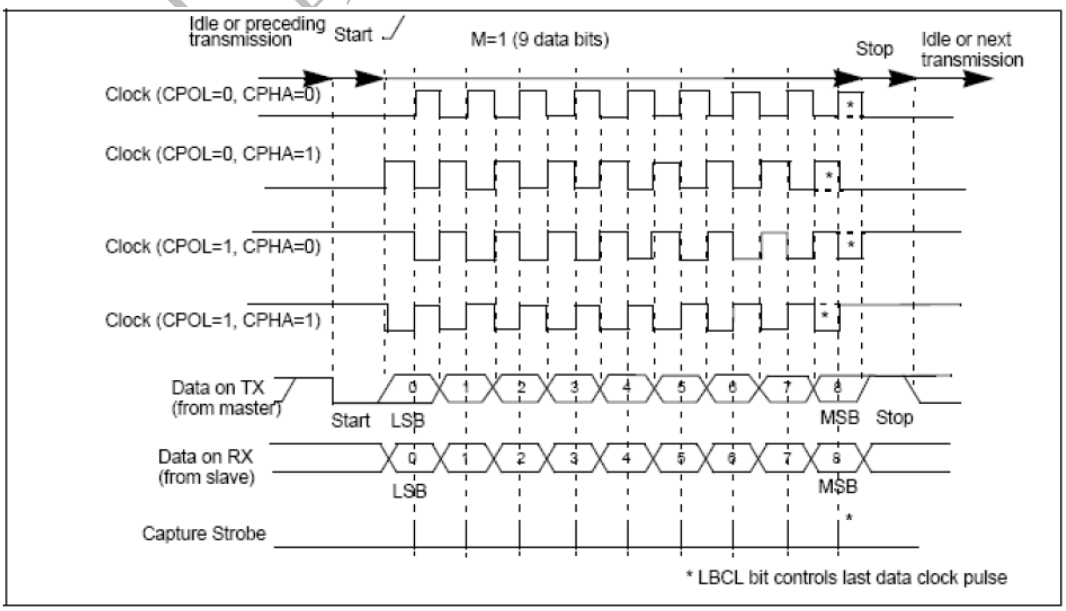
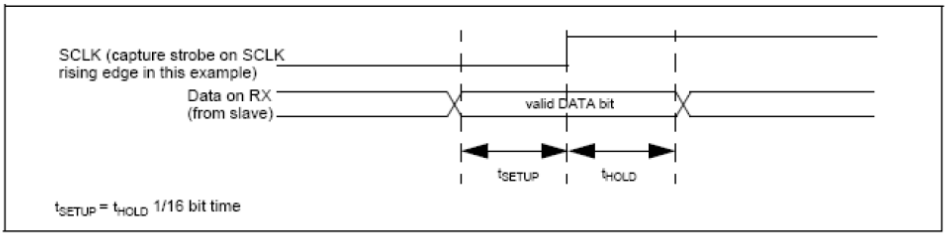


图**10** RX 数据采样/保持时间



注： 在智能卡模式下 SCLK 的功能不同，有关细节请参考智能卡模式部分。

3.3.9 单线半双工通信

单线半双方模式通过设置USART_CR3 寄存器的HDSEL 位选择。在这个模式里，下面的位必须保持清零状态：

● USART_CR2 寄存器的LINEN 和CLKEN 位

● USART_CR3 寄存器的SCEN 和IREN 位USART 可以配置成遵循单线半双工协议。半双工和全双工通信是用控制位“HALF DUPLEX SEL " 选择的。当HDSEL 写“1 " 时

● RX 不再被使用

● 当没有数据传输时，TX 总是被释放。因此，它在空闲状态的或接收状态时表现为一个标准I/O 口。这就意味该I/O 在不被USART 驱动时，必须配置成悬空输入（或开漏的输出高）。除此以外，通信与正常USART 模式类似。由软件来管理线上的冲突（例如通过使用一个中央仲裁器）。特别的是，发送从不会被硬件所阻碍。当TE 位被设置时，只要数据一写到数据寄存器上，发送就继续。

3.3.10 利用DMA连续通信

USART 可以利用DMA 连续通信。Rx 缓冲器和Tx 缓冲器的DMA 请求是分别产生的。

注意： 你应该参考产品技术说明以确定是否可用*DMA* 控制器。如果产品无*DMA* 功能，你应该如*19.3.3* 节或*19.3.4* 里所描述的方法使用*USART* 。在*USART2_SR* 寄存器里，你可以清零*TXE/RXNE* 标志来实现连续通信。

利用DMA发送

使用DMA 进行发送，可以通过设置USART_CR3 寄存器上的DMAT 位激活。只要TXE 位被置起，就从配置成使用DMA 外设的SRAM 区装载数据到USART_DR 寄存器。为USART 的发送分配一个DMA 通道的步骤如下（x 表示通道号）：

1. 在DMA控制寄存器上将USART_DR寄存器地址配置成DMA传输的目的地址。在每个TXE事件后，数据将被传送到这个地址。
2. 在DMA控制寄存器上将存储器地址配置成DMA传输的源地址。在每个TXE事件后，数据将从此存储器区传送到USART_DR寄存器。
3. 在DMA控制寄存器中配置要传输的总的字节数。
4. 在DMA寄存器上配置通道优先级。
5. 根据应用程序的要求配置在传输完成一半还是全部完成时产生DMA中断。
6. 在DMA寄存器上激活该通道。

当DMA 控制器中指定的数据量传输完成时，DMA 控制器在该DMA 通道的中断向量上产生一中断。在中断服务程序里，软件应将USART_CR3 寄存器的DMA
位清零。

注意： 如果*DMA* 被用于发送，不要使能*TXEIE* 位。

利用DMA接收

使用DMA 进行接收，可以通过设置USART_CR3 寄存器的DMAR 位激活。只要接收到一个字节，数据就从USART_DR 寄存器放到配置成使用DMA 的SRAM区(参考DMA 技术说明)。为USART 的接收分配一个DMA 通道步骤如下（x 表示通道号）：

1. 通过DMA控制寄存器把USART_DR寄存器地址配置成传输的源地址。在每个RXNE事件后此地址上的数据将传输到存储器。
2. 通过DMA控制寄存器把存储器地址配置成传输的目的地址。在每个RXNE事件后，数据将从USART_DR传输到此存储器区。
3. 在DMA控制寄存器中配置要传输的总的字节数。
4. 在DMA寄存器上配置通道优先级。。
5. 根据应用程序的要求配置在传输完成一半还是全部完成时产生DMA中断。
6. 在DMA控制寄存器上激活该通道。

当DMA 控制器中指定的传输数据量接收完成时，DMA 控制器在该DMA 通道的中断矢量上产生一中断。在中断程序里， USART_CR3 寄存器的DMAR 位应该被软件清零。

注意： 如果*DMA* 被用来接收，不要使能*RXNEIE* 位。

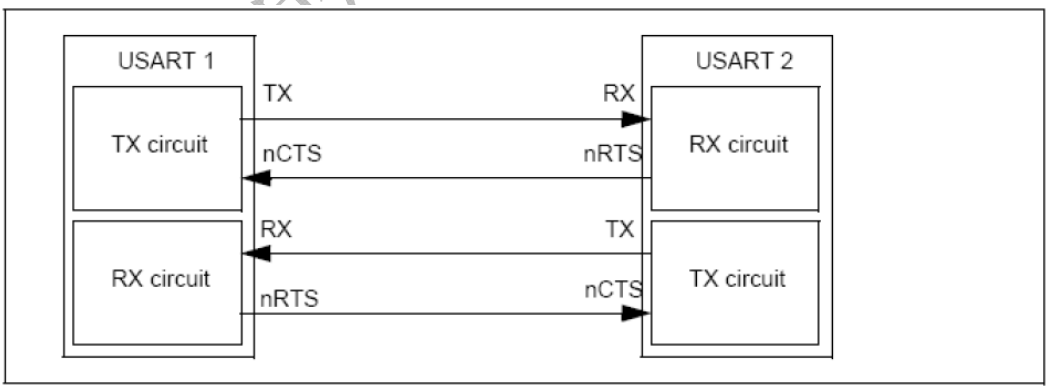
多缓冲器通信中的错误标志和中断产生

在多缓冲器通信的情况下，通信期间如果发生任何错误，在当前字节传输后将置起错误标志。如果中断使能位被设置，将产生中断。在单个字节接收的情况下，和RXNE 一起被置起的帧错误、溢出错误和噪音标志，有单独的错误标志中断使能位；如果设置了，会在当前字节传输结束后，产生中断。

3.3.10 硬件流控制

利用nCTS输入和nRTS输出可以控制2 个设备间的串行数据流。图11 表明在这个模式里如何连接2 个设备。

图11 两个USART 间的硬件流控制

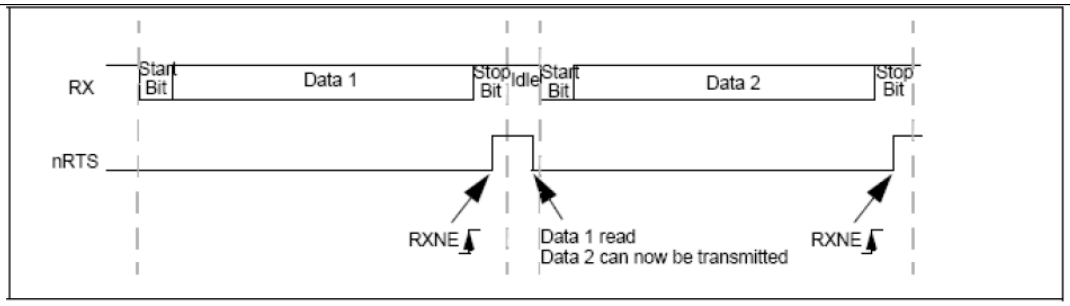


通过将 UASRT_CR3 中的 RTSE 和 CTSE 置位，可以分别独立地使能 RTS 和 CTS 流控制。

RTS流控制

如果RTS流控制被使能(RTSE=1)，只要USART接收器准备好接收新的数据，nRTS就变成有效(接低电平)。当接收寄存器内有数据到达时，nRTS被释放，由此表明希望在当前帧结束时停止数据传输。图12 展示了一个启用RTS流控制的通信的例子。

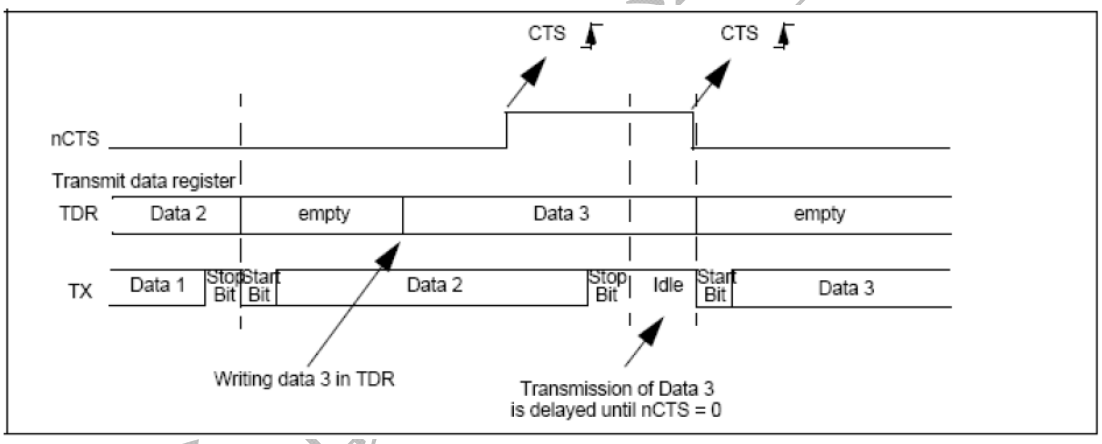
图12 RTS 流控制



CTS流控制

如果CTS 流控制被使能(CTSE=1),发送器在发送下一帧前检查nCTS 输入。如果nCTS 有效(被拉成低电平),则下一个数据被发送(假设那个数据是准备发送的,也就是TXE=0),否则下一帧数据不被发出去。若nCTS 在传输期间被变成无效,当前的传输完成后停止发送。
当CTSE=1 时,只要nCTS 输入一变换状态,CTSIF 状态位就自动被硬件设置。它表明接收器是否准备好进行通信。如果USART_CT3 寄存器的CTSIE 位被设置,中断产生。下图展示了一个CTS 流控制被启用的通信的例子。

图13 CTS 流控制



CTS流控制

如果CTS 流控制被使能(CTSE=1),发送器在发送下一帧前检查nCTS 输入。如果nCTS 有效(被拉成低电平),则下一个数据被发送(假设那个数据是准备发送的,也就是TXE=0),否则下一帧数据不被发出去。若nCTS 在传输期间被变成无效,当前的传输完成后停止发送。

3.4 中断请求

表5 USART 中断请求

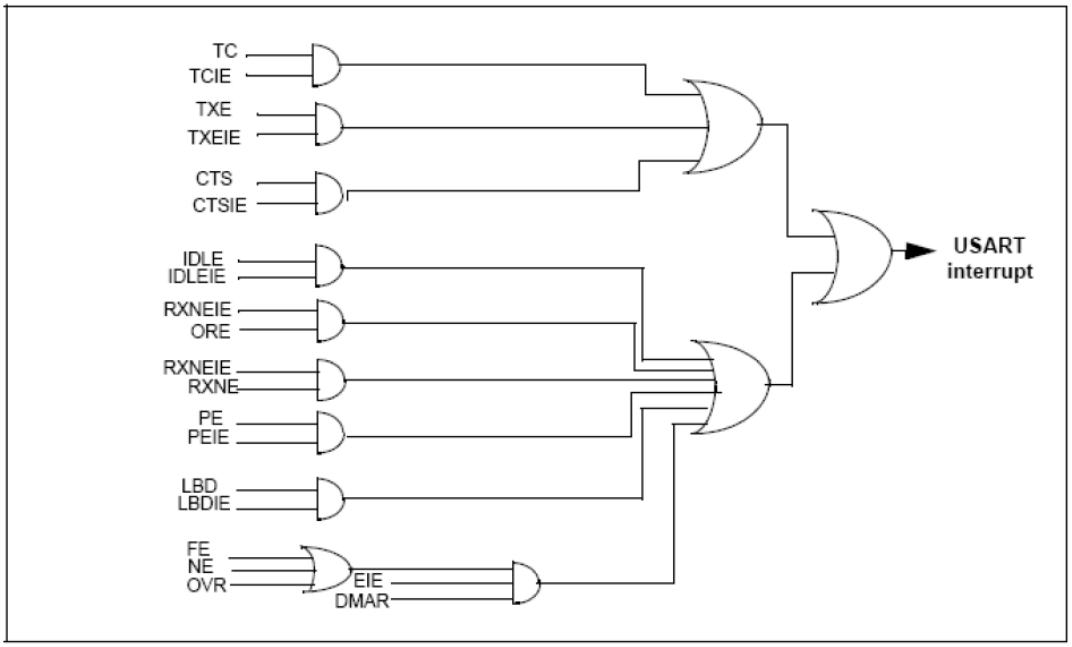
中断事件	事件标志	使能位
发送数据寄存器空	TXE	TXEIE
CTS标志	CTS	CTSIE
发送完成	TC	TCIE
接收数据就绪可读	TXNE	TXNEIE
检测到数据溢出	ORE	
检测到空闲线路	IDLE	IDLEIE
奇偶检验错	PE	PEIE
断开标志	LBD	LBDIE
噪声标志，多缓冲通信中的溢出错误和帧错误	NE或ORT或FE	EIE

USART的各种中断事件被连接到同一个中断向量(见 图 184)，有以下各种中断事件：

- 发送期间：发送完成中断、清除发送中断、发送数据寄存器空中断。
- 接收期间：空闲总线检测中断、溢出错误中断、接收数据寄存器非空中断、校验错误中断、LIN 断开符号检测中断、噪音中断（仅在多缓冲器通信）和帧错误中断（仅在多缓冲器通信）。

如果对应的使能控制位被设置，这些事件就会产生各自的中断。

图**14** USART 中断映像图

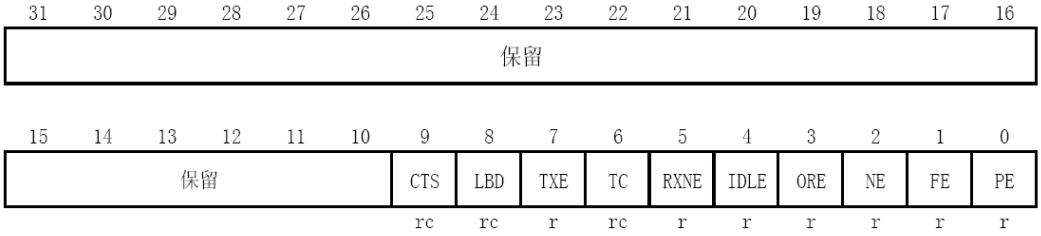


3.5 USART寄存器描述

3.5.1 状态寄存器(USART_SR)

地址偏移：0x00

复位值：0x00C0



位31:10	保留位，硬件强制为0
位9	CTS: CTS 标志 如果CTSE位置位，当nCTS输入变化状态时，该位被硬件置高。由软件将其清零。如果USART_CR3中的CTSIE为一，产生中断 0：nCTS状态线上没有变化 1：nCTS状态线上发生变化

位8	LBD: LIN break检测标志（状态标志） 0：没有检测到LIN break 1：检测到LIN break 注意：若LBDIE=1，当LBD为1时要产生中断
位7	TXE:发送数据寄存器空 当TDR寄存器中的数据被硬件转移到移位寄存器的时候，该位被硬件置位。如果USART_CR1寄存器中的TXEIE为1，则产生中断。对USART_DR的写操作，将该位清零。 0：数据还没有被转移到移位寄存器 1：数据已经被转移到移位寄存器 注意：单缓冲器传输中使用该位
位6	TC: 发送完成 当包含有数据的一帧发送完成后，由硬件将该位置位。如果USART_CR1中的TCIE为1，产生中断。由软件序列清除该位（先对USART_SR进行读操作，然后对USART_DR进行写操作） 0：发送还未完成 1：发送完成成
位5	RXNE:读数据寄存器非空 当RDR移位寄存器中的数据被转移到USART_DR寄存器中，该位被硬件置位。如果USART_CR1寄存器中的RXNEIE为1，中断产生。对USART_DR的读操作可以将改位清零。 0：数据没有收到 1：收到数据，可以读出
位4	IDLE:监测到IDLE总线 当检测到空闲总线时，该位被硬件置位。如果USART_CR1中的IDLEIE为1，产生中断。由软件序列清除该位（先读USART_SR，然后读USART_DR） 0：没有检测到空闲总线 1：检测到空闲总线 注意：IDLE位不会再次被置高直到RXNE位被置起(即又检测到一次空闲总线)
位3	ORE:过载错误 当RXNE还是1的时候，当前被接收在移位寄存器中的数据要往RDR寄存器中传送时，硬件将该位置位。如果USART_CR1中的RXNEIE为1的话，产生中断。由软件序列将其清零（先读USART_SR，然后读USART_CR） 0：没有过载错误 1：检测到过载错误 注意：该位被置位时，RDR寄存器中的值不会丢失，但是移位寄存器中的数据会被覆盖。如果EIE位被设置，在多缓冲器通信模式下，ORE标志置位会产生中断的。
位2	NE: 噪声错误标志 在接收到的帧检测到噪音时，由硬件对该位置位。由软件序列对其清玲（先读USART_SR，再读USART_DR） 0：没有检测到噪声 1：检测到噪声 注意：该位不会产生中断，因为它和RXNE一起出现，后者自己会在RXNE标志置位时产生中断，如果EIE位被设置，并且工作在多缓冲区通信模式下

位1	FE: 帧错误 当检测到同步错位，过多的噪声或者检测到break符，该位被硬件置位。由软件序列将其清零（先读USART_SR，再读USART_DR） 0：没有检测到帧错误 1：检测到帧错误或者break符 注意：该位不会产生中断，因为它和RXNE一起出现，后者自己会在RXNE标志置位时产生中断。如果当前传输的数据既产生了帧错误，又产生了过载错误，还是会继续该数据的传输，并且只有ORE位会被置位。 如果EIE位被置位，在多缓冲区通信模式下，随着FE标志被置位，中断产生。
位0	PE: 校验错误 在接收模式下，如果出现校验错误，硬件对该位置位。由软件序列对其清零（依次读USART_SR和USART_DR）。如果USART_CR1中的PEIE为1，产生中断。 0：没有校验错误

3.5.2 数据寄存器(USART_DR)

地址偏移：0x04
复位值：不确定

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留							DR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:9	保留位，硬件强制为0
位8:0	DR[8:0]:数据值 包含了发送或接收的数据。由于它是由两个寄存器组成的，一个给发送用（TDR），一个给接收用（RDR），该寄存器兼具读和写的功能。TDR寄存器提供了内部总线和输出移位寄存器之间的并行接口（参见图1）。RDR寄存器提供了输入移位寄存器和内部总线之间的并行接口。 当使能校验位（USART_CR1种PCE位被置位）进行发送时，写到MSB的值（根据数据的长度不同，MSB是第7位或者第8位）会被后来的校验位该取代。 当使能校验位进行接收时，读到的MSB位是接收到的校验位。

3.5.3 波特比率寄存器(USART_BRR)

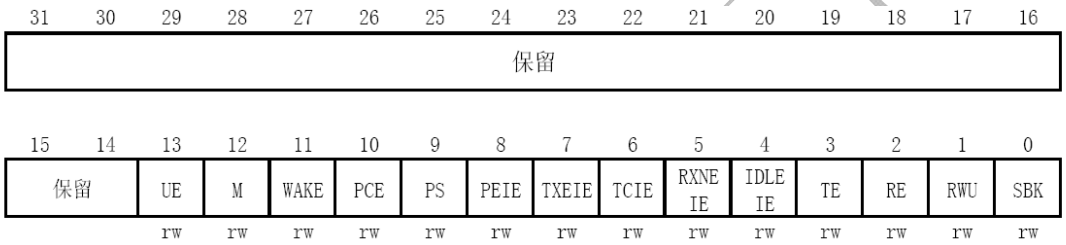
注意： 如果*TE* 或*RE* 被分别禁止，波特计数器停止计数
地址偏移：0x08
复位值：0x0000

位31:16	保留位，硬件强制为0
位15:4	DIV_Mantissa[11:0]:USARTDIV的小数部分 这12位定义了USART分频器除法因子(USARTDIV)的小数部分
位3:0	DIV_Fraction[3:0]:USARTDIV的整数部分 这4位定义了USART分频器除法因子(USARTDIV)的整数部分

3.5.4 控制寄存器1 (USART_CR1)

地址偏移：0x0C

复位值：0x0000



位31:14	保留位，硬件强制为0
位13	UE: USART使能 当该位被清零，USART的分频器和输出在当前字节传输完成后停止工作，以减少功耗。该位的置起和清零，是由软件操作的。 0：USART分频器和输出被禁止 1：USART模块使能
位12	M: 字长 该位定义了数据字的长度，由软件对其置位和清零操作 0：一个起始位，8个数据位，n个停止位 1：一个起始位，9个数据位，一个停止位 注意：在数据传输过程中（发送或者接收时），不能修改这个位

位11	WAKE: 唤醒的方法 这位决定了把USART唤醒的方法，由软件对该位置位或者清零。 0：被空闲总线唤醒 1：被地址标记唤醒
位10	PCE: 检验控制使能 用该位来选择是否进行硬件校验控制（对于发送来说就是校验位的产生；对于接收来说就是校验位的检测）。当使能了该位，在发送数据的MSB（如果M=1，MSB就是第9位；如果M=0,MSB就是第8位）插入校验位；对接收到的数据检查其校验位。软件对它置位或者清零。一旦该位被置位，当前字节传输完成后，校验控制才生效。 0：校验控制被禁止 1：校验控制被使能
位9	PS: 校验选择 该位用来选择当校验控制使能后，是采用偶校验还是奇校验。软件对它置位或者清零。当前字节传输完成后，该选择生效 0：偶校验 1：奇校验
位8	PEIE: PE中断使能 软件对该位置位或者清零 0：中断被禁止 1：当USART_SR中的PE为1时，产生USART中断
位7	TXEIE: 发送缓冲区空中断使能 软件对该位置位或者清零 0：中断被禁止 1：当USART_SR中的TXE为1时，产生USART中断
位6	TCIE: 发送完成中断使能 软件对该位置位或者清零 0：中断被禁止 1：当USART_SR中的TC为1时，产生USART中断
位5	RXNEIE: 接收缓冲区非空中断使能 软件对该位置位或者清零 0：中断被禁止 1：当USART_SR中的ORE或者RXNE为1时，产生USART中断
位4	IDLEIE: IDLE中断使能 软件对该位置位或者清零 0：中断被禁止 1：当USART_SR中的IDLE为1时，产生USART中断
位3	TE: 发送使能 该位使能发送器。软件对该位置位或者清零 0：发送被禁止 1：发送被使能 注意： 在数据传输过程中，除了在智能卡模式下，如果TE位上有个0脉冲（即“0”之后来一个“1”），会在当前数据字传输完成后，发送一个“预备状态”（空闲总线） 当TE被设置后，在真正发送开始之前，有一个比特时间的延迟。

位2	RE: 接收使能 软件对该位置位或者清零 0：接收被禁止 1：接收被使能，开始搜寻RX引脚上的起始位。
位1	RWU: 接收唤醒 该位用来决定是否把USART置于静默模式。软件对该位置位或者清零。当唤醒序列到来时，硬件也会将其清零。 0：接收器处于正常工作模式 1：接收器处于静默模式 注意： 在把USART置于静默模式(设置RWU位)之前，USART要已经先接收了一个数据字节。否则在静默模式下，不能被空闲总线检测唤醒。 当配置成地址标记检测唤醒(WAKE位为1)，在RXNE位被置位时，不能用软件来修改RWU位。
位0	SBK: 发送断开帧 使用该位来发送断开字符。软件可以对该位置位或者清零。应该由软件来置位它，然后在断开帧的停止位时，由硬件将该位复位。 0：没有发送断开字符 1：将要发送断开字符

3.5.5 控制寄存器2(USART_CR2)

地址偏移：0x10

复位值：0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	保留	LBDIE	LBDL	保留	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw		rw	rw	rw	rw

位31:15	保留位，硬件强制为0
位14	LINEN: LIN模式使能 软件对该位置位或者清零。 0：LIN模式被禁止 1：LIN模式被使能 LIN模式可以用USART_CR1寄存器中的SBK位发送LIN同步breaks,以及检测LIN同步break
位13:12	STOP: 停止位 用来设置停止位的位数 00：1个停止位 01：0.5个停止位 10：2个停止位 11：1.5个停止位

位11	CLKEN: 时钟使能 该位用来使能SCLK引脚 0：SCLK引脚被禁止 1：SCLK引脚被使能
位10	CPOL: 时钟极性 用户可以用该位来选择同步模式下SLCK引脚上时钟输出的极性。和CPHA位一起配合来产生用户希望的时钟/数据的采样关系 0：总线空闲时SCLK引脚上保持低电平 1：总线空闲时SCLK引脚上保持高电平
位9	CPHA: 时钟相位 用户可以用该位来选择同步模式下SLCK引脚上时钟输出的相位。和CPOL位一起配合来产生用户希望的时钟/数据的采样关系(参见图174和0) 0：时钟第一个边沿进行数据捕获 1：时钟第二个边沿进行数据捕获
位8	LBCL: 最后一位时钟脉冲 使用该位来控制是否在同步模式下，在SCLK引脚上输出最后发送的那个数据字节(MSB)对应的时钟脉冲 0：最后一位数据的时钟脉冲不从SCLK输出 1：最后一位数据的时钟脉冲会从SCLK输出 注意：最后一个数据位就是第8或者第9个发送的位（根据USART_CR1寄存器中的M位所定义的8或者9位数据帧格式）
位7	保留位，硬件强制为0
位6	LBDIE: LIN break检测中断使能 Break中断掩码（使用break定界符来检测break） 0：中断被禁止 1：只要USART_SR寄存器中的LBD为1就产生中断
位5	LBDL: LIN break检测长度 该位用来选择是11位还是10位的break检测 0：10位的break检测 1：11位的break检测
位4	保留位，硬件强制为0
位3:0	ADD[3:0]：该USART节点的地址 该位域给出这个USART节点的地址 这是在多处理器通信下的静默模式中使用的，使用地址标记来唤醒某个USART设备

注意：在发送被使能后不能写这三个位（CPOL 、CPHA 、LBCL）

3.5.6 控制寄存器3(USART_CR3)

地址偏移：0x14
复位值：0x0000

位31:11	保留位，硬件强制为0
位10	CTSIE: CTS中断使能 0：中断被禁止 1：只要USART_SR寄存器中的CTS为1就产生中断
位9	CTSE: CTS使能 0:CTS硬件流控制被禁止 1:CTS模式使能，只有nCTS输入信号有效（拉成低电平）时才能发送数据。如果在数据传输的过程中，nCTS信号变成无效,那么发完这个数据后，传输就停止下来。如果当nCTS为无效的时候，往数据寄存器里写了数据，那么这个数据要等到nCTS有效的时候才会被发送出去。
位8	RTSE: RTS使能 0：RTS硬件流控制被禁止 1：RTS中断使能，只有接收缓冲区内有空闲的空间时才请求下一个数据。当前数据发送完成后，发送操作就需要暂停下来。如果可以接收数据了，将nRTS输出置为有效（拉至低电平）
位7	DMAT: DMA使能发送 由软件对该位清零或者置位 1：发送时的DMA模式使能 0：发送时的DMA模式被禁止
位6	DMAR: DMA使能接收 由软件对该位清零或者置位 1：接收时的DMA模式使能 0：接收时的DMA模式被禁止
位5	SCEN: 智能卡模式使能 该位用来使能智能卡模式 0：智能卡模式使能 1：智能卡模式被禁止
位4	NACK:智能卡NACK使能 0：校验错误出现时，不发送NACK 1：校验错误出现时，发送NACK

位3	HDSEL：半双工选择 选择单线半双工模式 0：不选择半双工模式 1：选择半双工模式
位2	IRLP：红外低功耗 该位用来选择普通模式还是低功耗红外模式 0：通常模式 1：低功耗模式
位1	IREN：红外模式使能 由软件对该位清零或者置位 0：红外被禁止 1：红外使能
位0	EIE：错误中断使能 在多缓冲区通信模式下，当有帧错误、过载或者噪声错误时(USART_SR中德FE=1，或者ORE=1，或者NE=1)，产生中断。 0：中断被禁止 1：只要USART_CR3中的DMAR=1，并且USART_SR中的FE=1，或者ORE=1,或者NE=1，产生中断

3.6 USART寄存器地址映象

表6 USART 寄存器列表及其复位值

偏移	寄存器	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
000h	USART_SR	保留																						CTS	LBD	TXEIE	TC	RXNE	IDLE	ORE	NE	FE	PE
	复位值																							0	0	1	1	0	0	0	0	0	0
004h	USART_DR	保留																						DR[8:0]									
	复位值																							0	0	0	0	0	0	0	0	0	0
008h	USART_BRR	保留																DIV_Mantissa[15:4]												DIV_Fraction[3:0]			
	复位值																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00Ch	USART_CR1	保留																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	复位值																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
010h	USART_CR2	保留																	LIEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	保留	LBDIE	LBDL	保留	ADD[3:0]			
	复位值																		0	0	0	0	0	0	0		0	0		0	0	0	0
014h	USART_CR3	保留																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	复位值																						0	0	0	0	0	0	0	0	0	0	0
018h	USART_GTPR	保留																GT[7:0]												PSC[7:0]			
	复位值																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

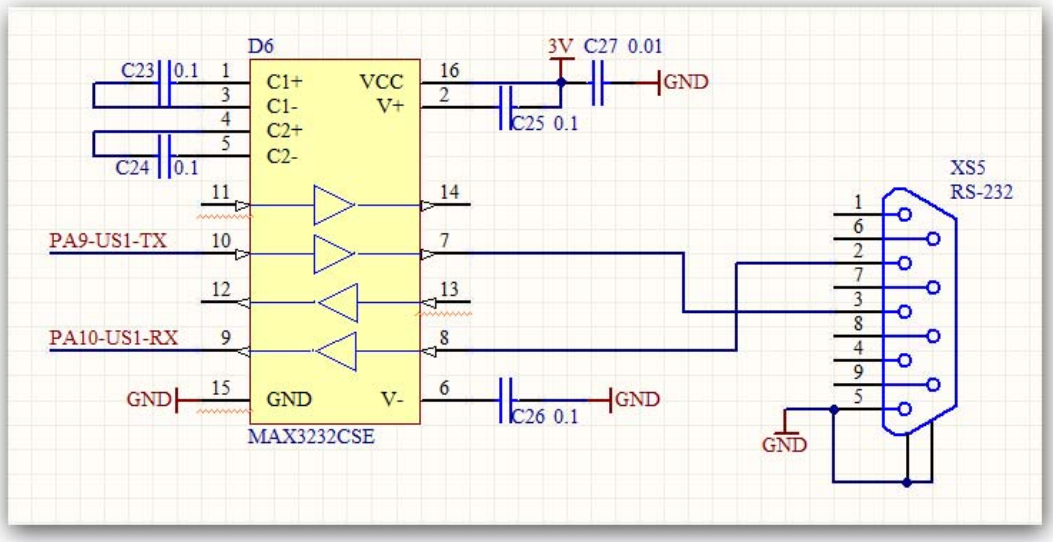
4. 应用实例

4.1. 设计要求

板子加电后，先向串口输出一串测试数据，然后在PC端的串口助手类软件上输入结束符为0x0d 0x0a的一串数据，发送到板子上，板子接收到该字串会将该字串输出回PC端串口助手软件。

4.2 硬件电路设计

如下图所示，只需要将RS232串口线连接板子的串口1到PC的串口上，如果PC上没有串口，可以用USB转串口线配合随板赠送的串口线，连接到板子上。

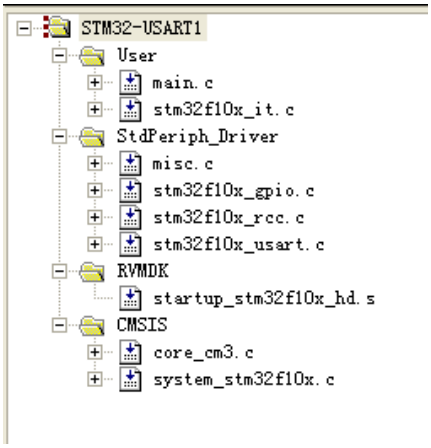


4.3 软件程序设计

根据任务要求，程序内容主要包括：

1. 初始化串口，使能串口中断
2. 通过串口中断服务程序检测串口。
3. 将收到的字符发送回PC。然后回到步骤2

整个工程包含4类源文件：



ASM--startup_stm32f10x_hd.s 由于奋斗板采用的是STM32F103大存储器芯片，因此采用STM32标准库自带的大存储器芯片启动代码，这个文件已经配置好了初始状态，以及中断向量表。可以直接在工程里使用，如果你在以后的应用中采用了中存储器或者小存储器STM32芯片，可以将启动代码换为startup_stm32f10x_md.s 或者 startup_stm32f10x_ld.s。

FWLIB--stm32f10x_gpio.c ST公司的标准库，包含了关于对通用IO口设置的函数。

stm32f10x_rcc.c ST公司的标准库，包含了关于对系统时钟设置的函数。

stm32f10x_USART.c ST公司的标准库，包含了关于对USART设置的函数。

Misc.c ST公司的标准库，包含了关于中断设置的函数。

CMSYS—是关于CORETEX-M3平台的系统函数及定义

USER—main.c 例程的主函数。

USER--stm32f10x_it.c 中断服务程序

//_____

```
int main(void)
{
  uint8_t a=0;
  /* System Clocks Configuration */
  RCC_Configuration();                                                    //系统时钟设置

  /*嵌套向量中断控制器
     说明了USART1抢占优先级级别0（最多1位） ，和子优先级级别0（最多7位） */
  NVIC_Configuration();                                                   //中断源配置

  /* 对控制LED指示灯的IO口进行了初始化，将端口配置为推挽上拉输出，口线速度为50Mhz。PA9,PA10端口复用为串口1的TX，RX。
  在配置某个口线时，首先应对它所在的端口的时钟进行使能。否则无法配置成功，由于用到了端口A,B， 因此要对这两个个端口的时钟
  进行使能，同时由于用到复用IO口功能用于配置串口。因此还要使能AFIO（复用功能IO）时钟和USART1时钟。*/
  GPIO_Configuration();                                                   //端口初始化

/* USART1 configuration ------------------------------------------------------*/
  USART_Config(USART1);                                                   //串口1初始化

  USART_OUT(USART1,"****(C) COPYRIGHT 2011 奋斗嵌入式开发工作室 *******\r\n");          //向串口1发送开机字符。
  USART_OUT(USART1,"*                                                 *\r\n");
  USART_OUT(USART1,"*         奋斗版STM32开发板 USART1 实验           *\r\n");
  USART_OUT(USART1,"*                                                 *\r\n");
  USART_OUT(USART1,"*  以HEX模式输入一串数据，以16进制0d 0a作为结束   *\r\n");
  USART_OUT(USART1,"*                                                 *\r\n");
  USART_OUT(USART1,"*  奋斗STM32论坛：www.ourstm.com                  *\r\n");
  USART_OUT(USART1,"*                                                 *\r\n");
  USART_OUT(USART1,"*****************************************************\r\n");
  USART_OUT(USART1,"\r\n");
  USART_OUT(USART1,"\r\n");
   while (1)
  {
      if(rec_f==1){                                                       //判断是否收到一帧有效数据
```

```
            rec_f=0;
            USART_OUT(USART1,"\r\n您发送的信息为: \r\n");
            USART_OUT(USART1,&TxBuffer1[0]);
            if(a==0) {GPIO_SetBits(GPIOB, GPIO_Pin_5); a=1;}                    //LED1  V6（V3板） V2（MINI板） 明暗闪烁
            else {GPIO_ResetBits(GPIOB, GPIO_Pin_5);a=0;  }
      }
  }
}


/*************************************************
            格式化串口输出函数
       "\r" 回车符    USART_OUT(USART1, "abcdefg\r")
       "\n" 换行符    USART_OUT(USART1, "abcdefg\r\n")
       "%s" 字符串    USART_OUT(USART1, "字符串是：%s","abcdefg")
       "%d" 十进制    USART_OUT(USART1, "a=%d",10)
*************************************************/
void USART_OUT(USART_TypeDef* USARTx, uint8_t *Data,...){
    const char *s;
    int d;
    char buf[16];
    va_list ap;
    va_start(ap, Data);
    while(*Data!=0){                                                        //判断是否到达字符串结束符
            if(*Data==0x5c){                                                //  判断控制符是否为'\'
                  switch (*++Data){
                        case 'r':                                           //回车符
                              USART_SendData(USARTx, 0x0d);
                              Data++;
                              break;
                        case 'n':                                           //换行符
                              USART_SendData(USARTx, 0x0a);
                              Data++;
                              break;
                        default:
                              Data++;
                            break;
                  }
            }
            else if(*Data=='%'){                                                    // 判断控制符是否为'%'
                  switch (*++Data){
                  case 's':                                                         //字符串
                        s = va_arg(ap, const char *);
                        for ( ; *s; s++) {
                              USART_SendData(USARTx,*s);
                              while(USART_GetFlagStatus(USARTx, USART_FLAG_TC)==RESET);   //发送是否完成
                        }
                        Data++;
                        break;
                  case 'd':                                                         //十进制
                        d = va_arg(ap, int);
                        itoa(d, buf, 10);
                        for (s = buf; *s; s++) {
                              USART_SendData(USARTx,*s);
                              while(USART_GetFlagStatus(USARTx, USART_FLAG_TC)==RESET); //发送是否完成
                        }
                        Data++;
                        break;
                        default:
                              Data++;
                              break;
                  }
            }
            else USART_SendData(USARTx, *Data++);                                   //发送单个字节
            while(USART_GetFlagStatus(USARTx, USART_FLAG_TC)==RESET);               //发送是否完成
      }
}
/*************************************************
            整形数据转字符串函数
        char *itoa(int value, char *string, int radix)
```

```
            radix=10 标示是10进制     非十进制，转换结果为0;

          例：d=-379;
            执行   itoa(d, buf, 10); 后

            buf="-379"
**********************************************************/
char *itoa(int value, char *string, int radix)
{
    int     i, d;
    int     flag = 0;
    char    *ptr = string;

    /* This implementation only works for decimal numbers. */
    if (radix != 10)
    {
        *ptr = 0;
        return string;
    }

    if (!value)
    {
        *ptr++ = 0x30;
        *ptr = 0;
        return string;
    }

    /* if this is a negative value insert the minus sign. */
    if (value < 0)
    {
        *ptr++ = '-';

        /* Make the value positive. */
        value *= -1;
    }

    for (i = 10000; i > 0; i /= 10)
    {
        d = value / i;

        if (d || flag)
        {
            *ptr++ = (char)(d + 0x30);
            value -= (d * i);
            flag = 1;
        }
    }

    /* Null terminate the string. */
    *ptr = 0;

    return string;

} /* NCL_Itoa */

//_____________________________________________________

void USART_Config(USART_TypeDef* USARTx){
  USART_InitStructure.USART_BaudRate = 115200;                                    //速率115200bps
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;                     //数据位8位
  USART_InitStructure.USART_StopBits = USART_StopBits_1;                          //停止位1位
  USART_InitStructure.USART_Parity = USART_Parity_No;                             //无校验位
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;   //无硬件流控
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;                 //收发模式

  /* Configure USART1 */
  USART_Init(USARTx, &USART_InitStructure);                                       //配置串口参数函数


  /* Enable USART1 Receive and Transmit interrupts */
  USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);                                  //使能接收中断
  USART_ITConfig(USART1, USART_IT_TXE, ENABLE);                                   //使能发送缓冲空中断
  //USART_ITConfig(USART1, USART_IT_TC, ENABLE);                                  //使能发送完成中断

  /* Enable the USART1 */
  USART_Cmd(USART1, ENABLE);
}
```

```
//-------------------------------------------------------------------------------
void USART1_IRQHandler(void)                                    //串口1 中断服务程序
{
  unsigned int i;
  if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)          //判断读寄存器是否非空
  {
    /* Read one byte from the receive data register */
    RxBuffer1[RxCounter1++] = USART_ReceiveData(USART1);         //将读寄存器的数据缓存到接收缓冲区里

    if(RxBuffer1[RxCounter1-2]==0x0d&&RxBuffer1[RxCounter1-1]==0x0a)   //判断结束标志是否是0x0d 0x0a
    {
        for(i=0; i< RxCounter1; i++) TxBuffer1[i]  = RxBuffer1[i];     //将接收缓冲器的数据转到发送缓冲区，准备转发
        rec_f=1;                                                       //接收成功标志
        TxBuffer1[RxCounter1]=0;                                       //发送缓冲区结束符
        RxCounter1=0;

    }
  }

  if(USART_GetITStatus(USART1, USART_IT_TXE) != RESET)           //这段是为了避免STM32 USART 第一个字节发不出去的BUG
  {
     USART_ITConfig(USART1, USART_IT_TXE, DISABLE);              //禁止发缓冲器空中断，
  }

}
```

5 运行过程

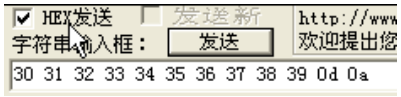
按  编译工程， 完成后会提示如下。

```
Build target 'STM32-USART1'
linking...
Program Size: Code=1868 RO-data=336 RW-data=76 ZI-data=532
FromELF: creating hex file...
".\Obj\STM32_FD_USART1.axf" - 0 Error(s), 0 Warning(s).
```

1. 通过JLINK V8或者串口将代码写入板子，具体的烧写步骤，参考奋斗板文档目录下的《奋斗版STM32开发板JTAG下载步骤》或者《奋斗版STM32开发板串口下载步骤》。
2. 使用串口线将板子与PC连接起来， 运行串口助手软件。 设置如下：

3. 通过USB线给板子加电。串口助手软件会收到如下的信息。

4．点击串口助手软件的HEX发送点选框，在字符输入框中输入需要发送的16进制数据，以0d 0a 作为结束符，这两个结束符，是串口例程用于判断接收帧结束的标志位。



5．点击发送键，将刚输入的字符发送给板子， 板子串口收到后，会将该帧数据回送回PC。显示如下。

该例程实验完成。

用户可根据例程编写自己的应用例程。