第 5 章 串 口 通 信

串口通信作为一种基础而重要的通信方式，在工业控制、仪器仪表、嵌入式系统等领域有着广泛的应用。在实际工程应用中，大量设备通过串口实现与上位机的数据传输与交互。对于需要复杂处理的数据，通常采用上位机进行预处理后，再通过串口传输至下位机执行具体操作。本章将从串行通信的基本原理入手，详细阐述 STM32 微控制器的串口通信机制及其实现过程，为读者深入理解串口通信技术提供系统的理论指导和实践参考。


视频讲解

5.1 串口通信基础

串口通信协议是计算机与外部设备进行数据交换的一种标准化通信方式。在计算机领域，大多数设备都配备了两个符合 RS-232 标准的串行通信接口。这种通信协议不仅广泛应用于计算机外围设备的连接，更是仪器仪表领域中的通用通信标准——众多支持 GPIB（通用接口总线）的设备同样集成了 RS-232 串口接口。此外，串口通信协议在远程数据采集系统中也发挥着重要作用，为远程设备的数据传输提供了可靠且高效的解决方案。

5.1.1 基本概念

1. 计算机通信方式

并行通信与串行通信是计算机常用的两种通信方式。并行通信指数据的各位同时进行传送（发送或接收）的通信方式。其优点是通信速度快，缺点是设备之间的数据线多，通信距离短。串行通信指数据是一位一位按顺序传送的通信方式。虽然通信速率低，但实现的方法及连线简单，适合远距离通信。

串行通信有同步和异步两种方式，同步通信时相互通信的设备之间需要时钟同步，必须有同步信号，实现复杂。异步通信时相互通信的设备之间不需要同步，只要求通信的接口方式及速率相同，以起始位和停止位为标志表示数据发送的开始和结束。监控系统中常采用串行异步通信方式实现智能设备或采集器与监控主机（前置机）之间的通信。

2. 通信协议

异步通信时数据一帧一帧地传送，帧的格式和通信速率共同构成了通信协议。帧的格

式由起始位、数据位、奇偶校验位和停止位组成，如图 5-1 所示。起始位都只有 1 位；数据位常为 4～8 位；校验位只有 1 位或没有，常用的校验方式为 3 种，偶校验记为 e，奇校验记为 o，无校验位记为 n；停止位为 1～2 位。一个数据帧的长度称为字长，字长＝起始位＋数据位＋校验位＋停止位。



图 5-1　一个数据帧

波特率是描述串行通信的速率的主要参数，单位为“位/秒”，记为 b/s。常用的异步串行通信波特率有 1200b/s、2400b/s、4800b/s、9600b/s、19200b/s 等。

在监控系统中，为了确保两台设备之间的正常通信，需要对通信端口进行设置，端口设置的格式为“波特率，校验位，数据位位数，停止位位数”。例如，某一个串口的端口设置为“9600，n，8，1”，表示该串口的通信速率为 9600b/s，没有校验位，数据位的长度为 8，停止位为 1 位，字长为 10。又如“2400，e，7，1”，由通信协议的定义可知，字长也为 10。

在设置具体的通信协议时，常遇到“流控制”这一概念，设置了流控制，设备串口的通信速率可以自动调整，防止数据的溢出或丢失。流控制有两种可选的方式，“硬件流控制”指用串口的两个引脚之间的电压差做流控信号，需要硬件设备的支持，监控系统中遇到的采集器和智能设备一般不支持硬件流控制；“软件流控制”用两个特殊的 ASCII 字符 Xon 和 Xoff 做流控信号，由于监控系统中涉及的设备之间传输时大多为二进制数据，里面很有可能刚好含有字符 Xon 和 Xoff，为了不至于引起设备的误解而导致传输错误，不能使用软件流控制。因此在设置通信串口时，一般不设置流控制，即选择“无流控制信号”。

5.1.2　常用的串行通信接口

串行通信有多种接口方式，监控系统中常用的有 RS232、RS422、RS485 三种接口方式，下面分别对它们进行简要的介绍。

1. RS232 串行通信接口

RS232 是 RS232-C 接口的简称，是一种广泛使用的串行通信标准接口，例如，计算机上的串行接口（简称串口）COM1、COM2。

1）RS232 定义

RS232 的机械接口有 DB9 和 DB25 两种形式，均有公头（针）、母头（孔）之分，RS232 常用接口如图 5-2 所示。DB9 及 DB25 两种串行接口的引脚信号定义如表 5-1 所示。



图 5-2　RS232 常用接口

表 5-1 RS232 接口中 DB9、DB25 引脚信号定义

9 针	25 针	信号名称	信号流向	简称	信 号 功 能
3	2	发送数据	DTE→DCE	TxD	DTE 发送串行数据
2	3	接收数据	DTE←DCE	RxD	DTE 接收串行数据
7	4	请求发送	DTE→DCE	RTS	DTE 请求切换到发送方式
8	5	清除发送	DTE←DCE	CTS	DCE 已切换到准备接收
6	6	数据设备就绪	DTE←DCE	DSR	DCE 准备就绪可以接收
5	7	信号地		GND	公共信号地
1	8	载波检测	DTE←DCE	DCD	DCE 已接收到远程载波
4	20	数据终端就绪	DTE→DCE	DTR	DTE 准备就绪可以接收
9	22	振铃指示	DTE←DCE	RI	通知 DTE,通信线路已接通

设备分为两种,一种是数据终端设备(DTE),例如计算机、采集器、智能设备等。另一种是数据通信设备(DCE),例如调制解调器、数据传输单元(DTU)、数据服务单元/通道服务单元(DCU/DSU)等。对于大多数设备,通常只用到 TxD、RxD、GND 三个针脚。注意表 5-1 中的信号流向,对于 DTE,TxD 是 DTE 向对方发送数据;而对于 DCE,TxD 是对方向自己发送数据。RS232 电气标准中采用负逻辑,逻辑"1"电平为－3～－15V,逻辑"0"电平为 3～15V,可以通过测量 DTE 的 TxD(或 DCE 的 RxD)和 GND 之间的电压了解串口的状态,空载状态下,它们之间应有－10V 左右(－5～－15V)的电压,否则该串口可能已损坏或驱动能力弱。

按照 RS232 标准,传输速率一般不超过 20kb/s,传输距离一般不超过 15m。实际使用时,通信速率最高可达 115200b/s。

2) RS232 串行接口基本接线原则

设备之间的串行通信接线方法,取决于设备接口的定义。设备间采用 RS232 串行电缆连接时有两类连接方式。

(1) 直通线:相同信号(RxD 对 RxD、TxD 对 TxD)相连,用于 DTE(数据终端设备)与 DCE(数据通信设备)相连,例如计算机与 MODEM(或 DTU)相连。

(2) 交叉线:不同信号(RxD 对 TxD、TxD 对 RxD)相连,用于 DTE 与 DTE 相连,例如计算机与计算机、计算机与采集器之间相连。

以上两种连接方法同种设备相连采用交叉线连接,不同种设备相连采用直通线连接。少数情况下会出现两台具有 DCE 接口的设备需要串行通信的情况,此时也用交叉方式连接。当一台设备本身是 DTE,但它的串行接口按 DCE 接口定义时,应按 DCE 接线。

通常,RS232 接口若为公头,则该接口按 DTE 接口定义;若为母头,则该接口按 DCE 接口定义。但也有反例,不能一概而论。一些 DTE 设备上的串行接口按 DCE 接口定义,采用 DB9 或 DB25 母接口,主要因为 DTE 接口一般都采用公头,当用手接触时易接触到针脚;采用母头时因不易碰到针脚,可避免人体静电对设备的影响。

对于某些设备上的非标准 RS232 接口,需要根据设备的说明书确定针脚的定义。如果已知 TxD、RxD 和 GND 三个针脚,但不清楚哪一个针脚是 TxD,哪一个针脚是 RxD,可以

通过万用表测量它们与 GND 之间的电压来判别，如果有一个电压为－10V 左右，则万用表红表笔所接的是 DTE 的 TxD 或 DCE 的 RxD。

3）RS232 的 3 种接线方式

（1）三线方式：即两端设备的串口只连接收、发、地三根线。适用于大多数简单通信场景，例如监控主机与采集器及大部分智能设备之间相连。

（2）简易接口方式：两端设备的串口除了连接收、发、地三根线外，另外增加一对握手信号（一般是 DSR 和 DTR）。具体需要哪对握手信号，需查阅设备接口说明。

（3）完全口线方式：两端设备的串口 9 线全接，例如计算机与外置 Modem 的连接电缆。

此外，有些设备虽然需要握手信号，但并不需要真正的握手信号，可以采用自握手的方式，连接方法如图 5-3 所示。



图 5-3　RS232 自握手的接线方式

2. RS422 串行通信接口

RS422 接口的定义很复杂，一般只使用 4 个端子，其针脚定义分别为 TX＋、TX－、RX＋、RX－，其中 TX＋和 TX－为一对数据发送端子，RX＋和 RX－为一对数据接收端子，如图 5-4 所示。RS422 采用了平衡差分电路，差分电路可在受干扰的线路上拾取有效信号，由于差分接收器可以分辨 0.2V 以上的电位差，因此可大大减弱地线干扰和电磁干扰的影响，有利于抑制共模干扰，传输距离可达 1200m。



图 5-4　RS422 方式通信接口定义与接线

和 RS232 不同的是，在 RS422 总线上可以挂接多台设备组网，总线上连接的设备 RS422 串行接口同名端相接，与上位机则收发交叉，可以实现点到多点的通信，如图 5-5 所示。RS232 只能点到点通信，不能组成串行总线。

通过 RS422 总线与计算机某一串口通信时，要求各设备的通信协议相同。为了在总线上区分各设备，各设备需要设置不同的地址。上位机发送的数据，所有的设备都能接收到，但只有地址符合上位机要求的设备响应。

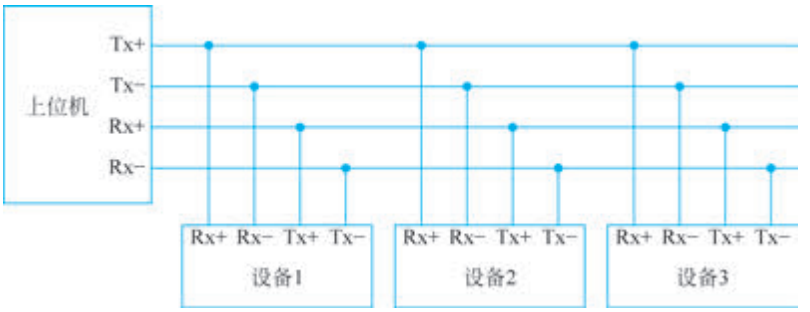

图 5-5 RS422 总线组网示意图

3. RS485 串行通信接口

RS485 是 RS422 的子集，只需要 DATA＋(D＋)、DATA－(D－)两根线。RS485 与 RS422 的不同之处在于 RS422 为全双工结构，可以在接收数据的同时发送数据；而 RS485 为半双工结构，在同一时刻只能接收或发送数据，如图 5-6 所示。



图 5-6 RS485 通信接口定义与接线

RS485 总线上也可以挂接多台设备，用于组网，实现点到多点及多点到多点的通信(多点到多点指总线上接的所有设备及上位机任意两台之间均能通信)，如图 5-7 所示。

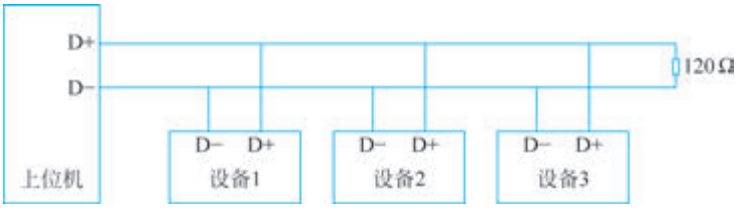

图 5-7 RS485 方式组网

连接在 RS485 总线上的设备也要求具有相同的通信协议，且地址不能相同。不通信时，所有的设备处于接收状态，当需要发送数据时，串口才翻转为发送状态，以避免冲突。为了抑制干扰，RS485 总线常在最后一台设备之后接入一个 120Ω 的电阻。

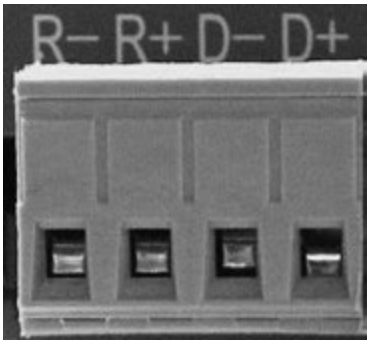

图 5-8 RS422/485 共用接口

很多设备同时有 RS485 接口方式和 RS422 接口方式，常共用一个物理接口，如图 5-8 所示。图中，RS485 的 D＋和 D－与 RS422 的 T＋和 T－共用。

4. 3 种串行通信接口方式比较

RS232、RS422、RS485 串行通信接口性能比较，如表 5-2 所示。

表 5-2　RS232、RS422、RS485 串行通信接口性能比较

接口性能		RS232	RS422	RS485
操作方式		电平	差分	差分
最大传输速率		20kb/s(15m)	10Mb/s(12m) 1Mb/s(120m) 100kb/s(1200m)	10Mb/s(12m) 1Mb/s(120m) 100kb/s(1200m)
驱动器输出电压	无负载	±5～±15V	±5V	±5V
	有负载时		±2V	±1.5V
驱动器负载阻抗		3～7kΩ	100Ω(min)	54Ω(min)
接收输入阻抗		3～7kΩ	4kΩ	12kΩ
接收器灵敏度		±3V	±200mV	±200mV
工作方式		全双工	全双工	半双工
连接方式		点到点	点到多点	多点到多点

5.1.3　应用串行通信的数据采集结构

监控系统中监控主机与采集器或智能设备，一般都通过串行接口通信。当一个被监控端的设备较多时，可采用总线方式或多串口方式进行数据采集。

1. 总线方式

一条 RS422/485 总线上可挂接多台设备，连接到上位机的某一个串口，实现上位机与各设备的通信，如图 5-9 所示。总线上的设备要求接口方式和通信协议相同，地址不同。接口方式不相同的设备如果与总线具有相同的通信协议，可通过接口转换器接入总线；如果某设备与总线上的其他设备通信协议不相同，仍可以通过协议转换器将其协议转换成总线的通信协议后，接入总线。

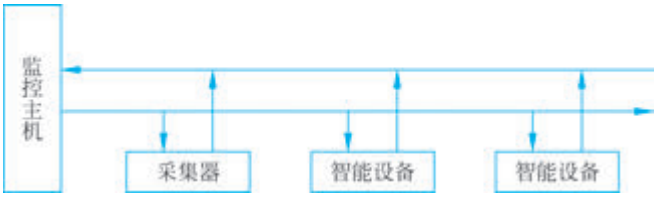

图 5-9　数据采集结构——总线方式

主机对同一条总线上并联的采集器或智能设备采用轮询方式采集数据，因此当连接在同一总线上的设备数量很多时，监控主机对总线上每一台设备采集一遍数据的用时将变长。

2. 多串口方式

设备分别与监控主机的各串口连接，监控主机为每个串口分配一个采集线程，各个串口可以同时采集数据，提高了采集速度。当监控主机的串口数不能满足设备的要求时，一般在监控主机上安装多串口卡增加监控主机的串口数量，如图 5-10 所示。

监控主机通过多个串口分别与各采集器和智能设备通信，此时没有通信协议相同的要求，同样，也不要求设备的地址各不相同。



图 5-10 数据采集结构——多串口方式

3. 多层次主从采集结构

多串口方式中，如果某些串口通过总线方式连接了多台设备，某些设备本身和监控主机一样具有一个或多个串行接口，可以连接多台设备，这种采集结构称为多层次主从采集结构。实际的监控系统数据采集方案中，采用这种复杂的多层次主从采集结构，一台监控主机最多可以连接数百台采集器和智能设备。

5.2 STM32 串口操作

STM32 系列微控制器集成了丰富的串口外设资源，其通信功能强大且灵活。以 STM32Fxxx 系列为例，通常可提供 5 路独立串口(USART/UART)，具有以下核心特性。

(1) 高精度波特率控制：内置分数波特率发生器，支持精确的通信速率配置。

(2) 多模式支持：支持同步/异步单线通信、半双工通信模式、LIN 总线协议、调制解调器控制、智能卡协议(ISO 7816)和 IrDA SIR 红外编解码规范。

(3) 高效数据传输：集成 DMA 控制器，支持无 CPU 干预的数据传输。

下面从寄存器层面设置串口，以实现基本的通信功能。具体实现的目标是利用串口 1 不断地向计算机发送信息，同时接收从串口发过来的数据，把接收过来的数据直接回传给计算机。本实验使用的开发板有 1 个 USB 串口，通过 USB 串口和计算机通信。

串口最基本的设置是波特率的配置。STM32 的串口使用简单、方便。只要开启了串口时钟，并设置相应 I/O 端口的模式，然后配置波特率、数据位长度、奇偶校验位等参数，就可以使用。下面简单介绍几个与串口基本配置直接相关的寄存器。

5.2.1 寄存器方式操作串口


视频讲解

1. 串口时钟使能

串口作为 STM32 的一个外设，其时钟由外设时钟使能寄存器控制，这里使用的串口 1 是在 APB2ENR 寄存器的第 14 位关于 APB2ENR 寄存器的详细介绍已在前面章节中说明，此处不再赘述。除了串口 1 的时钟使能在 APB2ENR 寄存器，其他串口的时钟使能都在 APB1ENR 寄存器。

2. 串口复位

当外设出现异常时，可以通过复位寄存器里面的对应位设置，实现该外设的复位，然后重新配置这个外设达到让其重新工作的目的。一般在系统刚开始配置外设的时候，都会先

执行复位该外设的操作。串口 1 的复位通过配置 APB2RSTR 寄存器的第 14 位来实现。APB2RSTR 寄存器的各位描述如图 5-11 所示。

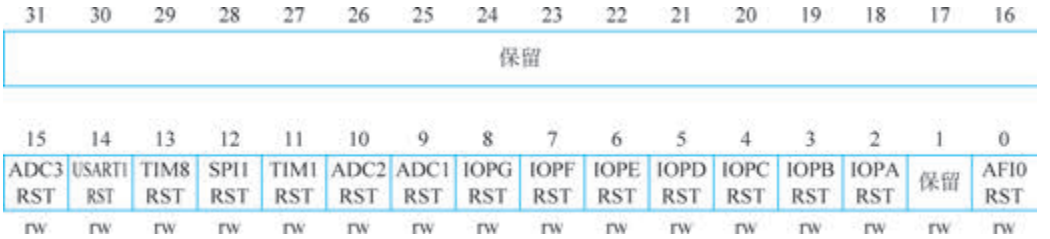

图 5-11　APB2RSTR 寄存器的各位描述

从图 5-11 可知，串口 1 的复位设置位在 APB2RSTR 的第 14 位。通过向该位写 1 复位串口 1，写 0 结束复位。其他串口的复位在 APB1RSTR 里面。

3. 串口波特率设置

每个串口都有一个独立的波特率寄存器 USART_BRR，通过设置该寄存器可以配置不同波特率。其各位描述如图 5-12 和表 5-3 所示。



图 5-12　寄存器 USART_BRR 结构

表 5-3　寄存器 USART_BRR 各位描述

位 31:16	保留位，硬件强制为 0
位 15:4 DIV_Mantissa[11:0]	USARTDIV 的整数部分，定义分频器除法因子的整数位，取值范围为 0x000-0xFFF
位 3:0 DIV_Fraction[3:0]	USARTDIV 的小数部分，定义了分频器除法因子的小数位，取值范围为 0x0-0xF

前面提到 STM32 的分数波特率概念，就在这个寄存器(USART_BRR)里面体现。USART_BRR 的最低 4 位(位[3:0])用来存放小数部分 DIV_Fraction，紧接着的 12 位(位[15:4])用来存放整数部分 DIV_Mantissa，最高 16 位未使用。STM32 的串口波特率计算公式如下：

$$\text{波特率} = \frac{f_{\text{PCLKx}}}{16 \times \text{USARTDIV}}$$

式中，f_{PCLKx} 是给串口的时钟(PCLK1 用于 USART2、3、4、5，PCLK2 用于 USART1)；USARTDIV 是一个无符号定点数。只要得到 USARTDIV 的值，就可以得到串口波特率寄

存器 USART_BRR 的值；反之，得到 USART_BRR 的值，也可以推导出 USARTDIV 的值。但我们更关心的是如何从 USARTDIV 的值得到 USART_BRR 的值，因为一般知道的是波特率和 PCLKx 的时钟，要求的是 USART_BRR 的值。

下面介绍如何通过 USARTDIV 得到串口 USART_BRR 寄存器的值。假设串口 1 要设置为 9600 的波特率，而 PCLK2 的时钟为 72MHz。这样，根据公式有

$$USARTDIV=72000000/(9600\times16)=468.75$$

得到

$$DIV_Fraction=16\times0.75=12=0X0C$$

$$DIV_Mantissa=468=0X1D4$$

这样，就得到了 USART_BRR 的值为 0X1D4C。只要设置串口 1 的 USART_BRR 寄存器值为 0X1D4C，就可以得到 9600 的波特率。当然，并不是任何条件下都可以随便设置串口波特率，在某些波特率和 PCLK2 频率下，还是会存在误差，设置波特率时的误差计算如表 5-4 所示。

表 5-4 设置波特率时的误差计算

波特率/(kb·s⁻¹)	$f_{PCLK}=36MHz$			$f_{PCLK}=72MHz$		
	实际/(kb·s⁻¹)	置于波特率寄存器中的值	误差/%	实际/(kb·s⁻¹)	置于波特率寄存器中的值	误差/%
2.4	2.400	937.5	0	2.4	1875	0
9.6	9.600	234.375	0	9.6	468.75	0
19.2	19.2	117.1875	0	19.2	234.375	0
57.6	57.6	39.0625	0	57.6	78.125	0
115.2	115.384	19.5	0.15	115.2	39.0625	0
230.4	230.769	9.75	0.16	230.769	19.5	0.16
460.8	461.538	4.875	0.16	461.538	9.75	0.16
921.6	923.076	2.4375	0.16	923.076	4.875	0.16
2250	2250	1	0	2250	2	0
4500	不可行	不可行	不可行	4500	1	0

4. 串口控制

STM32 的每个串口都有 3 个控制寄存器 USART_CR1～3，用于实现通信协议的全面配置。本节重点解析 USART_CR1 寄存器，该寄存器的各位描述如图 5-13 和表 5-5 所示。



图 5-13 寄存器 USART_CR 结构

表 5-5 USART_CR 寄存器各位描述

位 31:14	保留位,硬件强制为 0
位 13 UE	USART 使能,当该位被清零,在当前字节传输完成后,USART 的分频器和输出停止工作,以减少功耗。该位由软件设置和清零 0:关闭 USART 模块,分频器与输出停止,进入低功耗状态 1:使能 USART 模块
位 12 M	字长,该位定义了数据字的长度,由软件对其设置和清零 0:1 个起始位,8 个数据位,n 个停止位 1:1 个起始位,9 个数据位,n 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,该位决定了把 USART 唤醒,由软件对该位设置和清零 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,用该位选择是否进行硬件校验控制(对于发送就是校验位的产生;对于接收就是校验位的检测)。当使能了该位,在发送数据的最高位(如果 M=1,最高位就是第 9 位;如果 M=0,最高位就是第 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“1”或清“0”。一旦设置了该位,当前字节传输完成后,校验控制才生效 0:禁止校验控制 1:使能校验控制
位 9 PS	校验模式选择,当校验控制使能后,该位用来选择是采用偶校验还是奇校验。软件对它置“1”或清“0”。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,该位由软件设置或清除 0:禁止产生中断 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,该位由软件设置或清除 0:禁止产生中断 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,该位由软件设置或清除 0:禁止产生中断 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,该位由软件设置或清除 0:禁止产生中断 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,该位由软件设置或清除 0:禁止产生中断 1:当 USART_SR 中的 IDLE 为“1”时,产生 USART 中断

续表

位 3 TE	发送使能，该位使能发送器。该位由软件设置或清除 0：禁止发送 1：使能发送 注意： (1) 数据传输过程中，除了在智能卡模式下，如果 TE 位上有一个 0 脉冲(即设置为“0”之后再设置为“1”)，会在当前数据字传输完成后发送一个“前导符”(空闲总线) (2) 当 TE 设置后，在真正发送开始之前，有一比特时间的延迟
位 2 RE	接收使能，该位由软件设置或清除 0：禁止接收 1：使能接收，并开始搜寻 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。该位由软件设置或清除。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： (1) 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 (2) 当配置成地址标记检测唤醒(WAKE 位＝1)，RXNE 位被置位时，不能用软件修改 RWU 位
位 0 SBK	发送断开帧，使用该位来发送断开字符。该位可以由软件设置或清除。操作过程应该是软件设置位，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

5. 数据发送与接收

STM32 的发送与接收通过数据寄存器 USART_DR 来实现，这是一个双寄存器，包含了 TDR 和 RDR。当向该寄存器写数据时，串口就会自动发送，当收到收据时，也存在该寄存器内。该寄存器的各位描述如图 5-14 所示。



图 5-14 USART_DR 寄存器各位描述

可以看出，虽然是一个 32 位寄存器，但是只用了低 9 位(DR[8:0])，其他都保留。

DR[8:0]为串口数据，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。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。

6. 串口状态

串口的状态可以通过状态寄存器 USART_SR 读取。USART_SR 寄存器的各位描述,如图 5-15 所示。

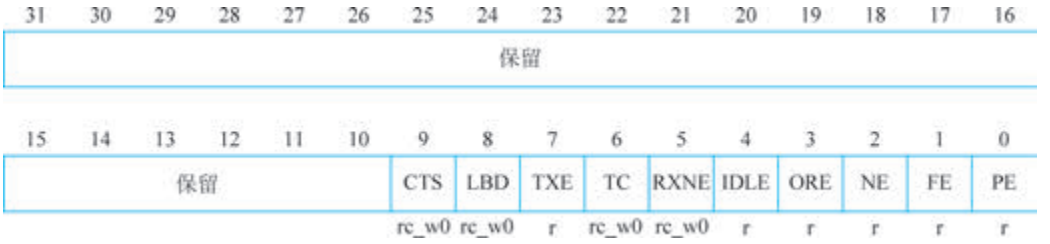

图 5-15 USART_SR 寄存器的各位描述

这里关注两个位,第 5、6 位 RXNE 和 TC。

RXNE(读数据寄存器非空):当该位被置 1 的时候,就是提示已经有数据被接收到,并且可以读出来了。这时候要尽快去读取 USART_DR,通过读 USART_DR 可以将该位清零,也可以向该位写 0,直接清除。


视频讲解

TC(发送完成):当该位被置位的时候,表示 USART_DR 内的数据已经被发送完成。如果设置了这个位的中断,则会产生中断。该位也有两种清零方式:①读 USART_SR,写 USART_DR;②直接向该位写 0。


视频讲解

通过以上一些寄存器的操作外加 I/O 端口的配置,就可以达到串口最基本的配置。

5.2.2 库函数方式操作串口


视频讲解

串口设置可以总结为如下几个步骤:

(1) 串口时钟使能,GPIO 时钟使能;

(2) 串口复位;

(3) GPIO 端口模式设置;

(4) 串口参数初始化;

(5) 开启中断并且初始化 NVIC(如果需要开启中断,才需要这个步骤);

(6) 使能串口;

(7) 编写中断处理函数。

下面简单介绍这几个与串口基本配置直接相关的几个固件库函数。这些函数和定义主要分布在 stm32f10x_usart.h 和 stm32f10x_usart.c 文件中。

1. 串口时钟及对应 GPIO 端口时钟使能

串口与 GPIOA 是挂载在 APB2 下面的外设，使能函数为：

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1|RCC_APB2Periph_GPIOA,ENABLE);
```

2. 串口复位

当外设出现异常的时候，可以通过复位设置实现该外设的复位，然后重新配置这个外设达到让其重新工作的目的。通常，在系统刚开始配置外设的时候，都会先执行复位该外设的操作。复位在函数 USART_DeInit()中完成。

```
void USART_DeInit(USART_TypeDef * USARTx); //串口复位函数原型
```

例如要复位串口 1，方法为：

```
USART_DeInit(USART1); //复位串口 1
```

3. 串口参数初始化

串口初始化通过 USART_Init()函数实现，函数原型如下：

```
void USART_Init(USART_TypeDef * USARTx,USART_InitTypeDef * USART_InitStruct);
```

这个函数的第一个参数是指定初始化的串口标号，这里选择 USART1。第二参数是一个 USART_InitTypeDef 类型的结构体指针，这个结构体指针的成员变量用来设置串口的一些参数。一般的实现格式为：

```
//波特率
USART_InitStructure.USART_BaudRate = bound;
//字长为 8 位数据格式
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
//一位停止位
USART_InitStructure.USART_StopBits = USART_StopBits_1;
//无奇偶校验位
USART_InitStructure.USART_Parity = USART_Parity_No;
//无硬件数据流控制
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
//收发模式
USART_InitStructure.USART_Mode = USART_Mode_Rx|USART_Mode_Tx;
//初始化串口
USART_Init(USART1,&USART_InitStructure);
```

从上面的初始化格式可以看出，初始化需要设置的参数为波特率、字长、停止位、奇偶校验位、硬件数据流控制、模式(收、发)。可以根据需要设置这些参数。

4. 数据发送与接收

STM32 的发送与接收通过数据寄存器 USART_DR 来实现，这是一个双寄存器，包含 TDR 和 RDR。当向该寄存器写数据的时候，串口就会自动发送，当收到数据的时候，也存在该寄存器内。

STM32 库函数操作 USART_DR 寄存器发送数据的函数是：

```
void USART_SendData(USART_TypeDef * USARTx,uint16_t Data);
```

通过该函数向串口寄存器 USART_DR 写入一个数据。

STM32 库函数操作 USART_DR 寄存器读取串口接收到的数据的函数是：

```
uint16_t USART_ReceiveData(USART_TypeDef * USARTx);
```

通过该函数可以读取串口接收到的数据。

5. 串口状态

串口的状态可以通过状态寄存器 USART_SR 读取。USART_SR 的各位描述如图 5-12 所示，5.2.1 节已有介绍。

固件库函数里面，读取串口状态的函数是：

```
FlagStatus USART_GetFlagStatus(USART_TypeDef * USARTx,uint16_t USART_FLAG);
```

这个函数的第 2 个参数非常关键，它标示要查看串口的哪种状态，如上面讲解的 RXNE（读数据寄存器非空）及 TC（发送完成）。例如要判断读寄存器是否非空（RXNE），操作库函数的方法是：

```
USART_GetFlagStatus(USART1,USART_FLAG_RXNE);
```

判断发送是否完成（TC），操作库函数的方法是：

```
USART_GetFlagStatus(USART1,USART_FLAG_TC);
```

6. 串口使能

串口使能是通过函数 USART_Cmd()来实现的，这个很容易理解，使用方法是：

```
USART_Cmd(USART1,ENABLE); //使能串口
```

7. 开启串口响应中断

使能串口中断的函数是：

```
void USART_ITConfig(USART_TypeDef * USARTx,uint16_t USART_IT,FunctionalState NewState)
```

这个函数的第 2 个参数是标示使能串口的类型，也就是使能哪种中断，因为串口的中断类型有很多种。例如在接收到数据时（RXNE 读数据寄存器非空），要产生中断，开启中断的方法是：

```
//开启中断,接收到数据中断
USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
```

在发送数据结束时（TC，发送完成）要产生中断，方法是：

```
USART_ITConfig(USART1,USART_IT_TC,ENABLE);
```

8. 获取相应中断状态

当使能某个中断且该中断发生时，状态寄存器中相应的标志位会被置位。中断处理函数中，要判断该中断是哪种类型，使用的函数如下：

```
ITStatus USART_GetITStatus(USART_TypeDef * USARTx, uint16_t USART_IT)
```

例如，使能串口发送完成中断后，若该中断发生，便可以在中断处理函数中调用此函数，以判断是否为串口发送完成中断，方法如下：

```
USART_GetITStatus(USART1,USART_IT_TC)
```

返回值是 SET，说明是串口发送完成中断发生。

下面两段代码，简单地完成了串口通信功能，主程序完成串口的初始化，然后打印两条信息，代码如下：

```
#include "stm32f10x.h"
#include "usart1.h"
int main(void)
{
    /* USART1 config 115200 8-N-1 */
    USART1_Config();
    printf("\r\n this is a usart printf test \r\n");
    printf("\r\n 欢迎您来到哈尔滨! \r\n");
    for(; ; )
    {
    }
 }
```

串口初始化代码如下：

```
#include "usart1.h"
void USART1_Config(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   USART_InitTypeDef USART_InitStructure;
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1| RCC_APB2Periph_GPIOA,ENABLE);
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOA,&GPIO_InitStructure);
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
   GPIO_Init(GPIOA,&GPIO_InitStructure);
   USART_InitStructure.USART_BaudRate = 115200;
   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
   USART_InitStructure.USART_StopBits = USART_StopBits_1;
   USART_InitStructure.USART_Parity = USART_Parity_No;
   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
   USART_InitStructure.USART_Mode = USART_Mode_Rx| USART_Mode_Tx;
   USART_Init(USART1,&USART_InitStructure);
   USART_Cmd(USART1,ENABLE);
```

```
22.     }
23.     ///重定向 c 库函数 printf 到 USART1
24.     int fputc(int ch, FILE * f)
25.     {
26.         /* 发送一字节数据到 USART1 */
27.         USART_SendData(USART1,(uint8_t) ch);
28.         /* 等待发送完毕 */
29.         while (USART_GetFlagStatus(USART1,USART_FLAG_TC) == RESET);
30.         return(ch);
31.     }
32.     ///重定向 c 库函数 scanf 到 USART1
33.     int fgetc(FILE * f)
34.     {
35.         /* 等待串口 1 输入数据 */
36.         while (USART_GetFlagStatus(USART1,USART_FLAG_RXNE) == RESET);
37.         return (int)USART_ReceiveData(USART1);
38.     }
```

第 6 行代码,配置 USART1 和 GPIOA 端口时钟。

第 7～10 行代码,配置 USART1 的 Tx (PA.09)引脚为复用推挽输出,速度为 50MHz。

第 11～13 行代码,配置 USART1 的 Rx (PA.10)引脚为浮空输入。

第 14～21 行代码,USART1 通信参数配置。

第 24～31 行 fputc 函数重定向 printf 到 USART1,将标准库函数 printf 的输出重定向到串口 USART1,实现通过串口发送数据到上位机。通过调用 USART_SendData()函数将字符写入 USART1 的数据寄存器(USART_DR),触发串口发送。通过 USART_GetFlagStatus()函数循环检测发送完成标志位(TC)。当 TC=1 时,表示当前字节已从移位寄存器完全发送到 TX 引脚,可以安全发送下一字节。

第 33～37 行 fgetc 函数重定向 scanf 到 SART1,将标准库函数 scanf 的输入重定向到串口 USART1,实现通过串口接收上位机发送的数据。通过 USART_GetFlagStatus()函数循环检测接收缓冲区非空标志(RXNE)。当 RXNE=1 时,表示 USART1 的接收缓冲区(RDR)中有新数据。调用 USART_ReceiveData( )函数读取接收缓冲区中的数据。

使用串口调试时应注意硬件依赖条件:

(1) 需提前配置 USART1 的时钟、GPIO 引脚(PA9/TX 和 PA10/RX)、波特率、数据位、停止位等参数;

(2) 启用微库(MicroLIB):在 Keil MDK 中,需勾选 Use MicroLIB 选项以支持标准输入输出重定向。微库是专为嵌入式系统优化的轻量级 C 库,支持 printf 和 scanf 的简化实现。

串口代码实验现象如图 5-16 所示。



图 5-16 串口代码实验现象

习题

(1) RS232 通信中,DTE 与 DCE 设备之间应采用哪种接线方式?若两台 DTE 设备直接通信,应如何连接?

(2) 简述 STM32 中 USART_BRR 寄存器的作用及波特率计算公式。

(3) 使用 STM32 库函数配置串口时,一般需要哪几个步骤?

(4) 编程实现基于 USART2 的中断接收数据并回显。

(5) 编程实现 DMA 串口发送,提高数据传输效率。