# 第5章 串口通信

CHAPTER 5

## 5.1 本章导读

串口通信在实际中有着广泛的应用，很多设备的数据传输通过串口和上位机进行通信。在实际应用中，对于需要大量处理的数据，可以通过上位机处理后，通过串口发送给下位机，本章从串行通信的基本概念开始，说明 STM32 的串口通信过程，读者可以获取到以下信息：

(1) 串口通信的基本概念，常用串行接口分类及接线方式。

(2) STM32 串口操作的方式。

(3) 采用寄存器方法和库函数方法操作串口通信的详细步骤。

## 5.2 串口通信基础

视频讲解

串口是计算机中的一种通用设备通信的协议，也是仪器仪表设备通用的通信协议，很多 GPIB 兼容的设备也带有 RS232 口。同时，串口通信协议也可以用来获取远程采集设备的数据。

### 5.2.1 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。例如，打印机与计算机之间的通信一般都采用并行通信方式。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。

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一起称为通信协议。帧的格式

由起始位、数据位、校验位和停止位组成，如图 5-1 所示。起始位都只有一位；数据位长为 1～8 位；校验位只有一位或没有，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 位或 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。

| 起始位<br>(1位) | 数据位<br>(1~8位) | 校验位<br>(0位或1位) | 停止位<br>(1位或2位) |
|---|---|---|---|

图 5-1　一个数据帧

波特率用于描述串行通信的速率，一般单位为“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。

在通信系统中，为了指明两台设备之间的通信协议，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”，由通信协议的定义易知，字长也为 10。

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，不致发生数据的溢出或丢失。流控制一般有两种可选的方式，“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”指用两个特殊的 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”。

## 5.2.2　常用的串行通信接口

串行通信有多种接口方式，通信系统中常用的有 RS232、RS422、RS485 这 3 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，RS232-C 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、DB25 两种形式，有公头（针）、母头（孔）之分，常用的 DB9 接口外形及针脚序号如图 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。

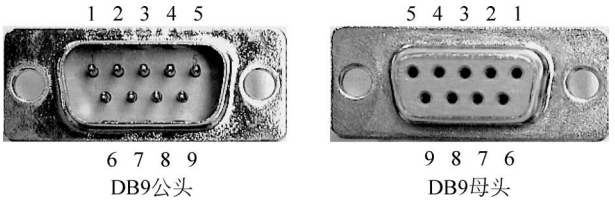


图 5-2　RS232 常用接口

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，简称为 DTE，例如计算机、采集器、智能设备等；另一种是数据电路设备(通信设备)，简称 DCE，例如调制解调器、数据端接设备(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 3 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，传输速率最高可达 115.2kb/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。

以上两种连接方法可以认为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。例如，艾默生网络能源有限公司生产的一体化采集器 IDA 采集模块上的调测接口是按 DCE 接口定义的，当计算机与 IDA 采集模块的调测口连接时，就要采用直通串行电缆。

一般来说，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 3 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：两端设备的串口只连接收、发、地三根线。一般情况下，三线方式即可满足要求，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 Modem 电缆（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。

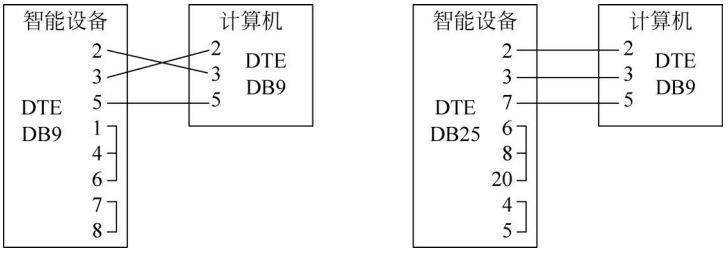


图 5-3　RS232 自握手的接线方式

**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。

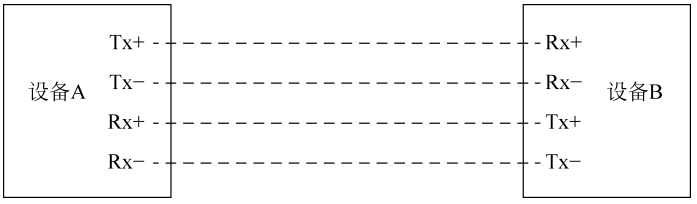


图 5-4　RS422 方式通信接口定义与接线

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。

**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

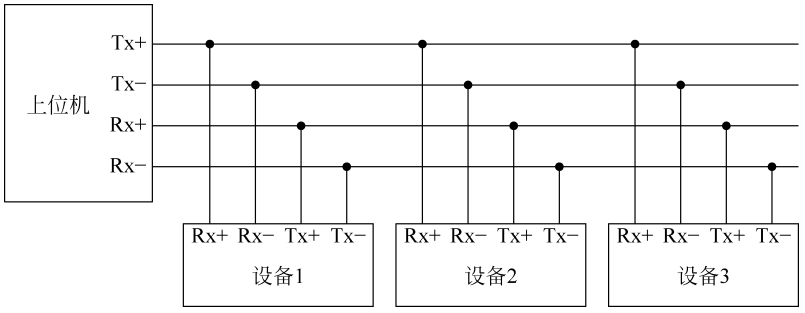


图 5-5　RS422 总线组网示意图

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。

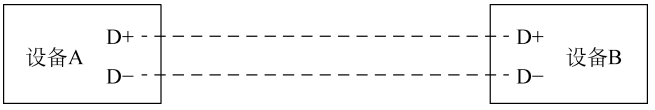


图 5-6　RS485 通信接口定义与接线

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。

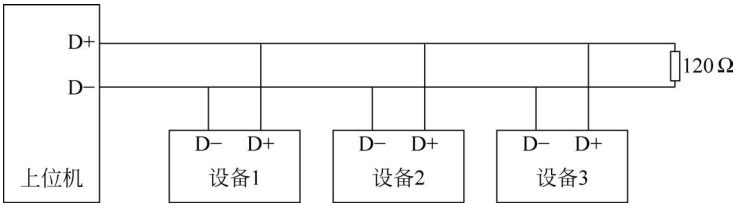


图 5-7　RS485 方式组网

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。

**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)<br>1Mb/s(120m)<br>100kb/s(1200m) | 10Mb/s(12m)<br>1Mb/s(120m)<br>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 |

## 5.3 STM32 串口操作

STM32 的串口资源相当丰富，功能也很强大。STM32Fxxx 一般可提供 5 路串口，有分数波特率发生器、支持同步单线通信和半双工单线通信、支持 LIN、支持调制解调器操作、智能卡协议和 IrDA SIR ENDEC 规范、具有 DMA 等。

下面采用寄存器方式设置串口，实现串口基本的通信功能。本节将实现利用串口 1 不间断地发送信息到计算机，同时接收从串口发过来的数据，把发送过来的数据直接送回给计算机。

串口最基本的设置，就是波特率的设置。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、奇偶校验位等信息，就可以使用。下面简单介绍这几个与串口基本配置直接相关的寄存器。


视频讲解

### 5.3.1 寄存器方式操作串口

**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。除了串口 1 的时钟使能在 APB2ENR 寄存器中，其他串口的时钟使能都在 APB1ENR 寄存器中。

**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 寄存器结构如图 5-8 所示。

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADC3 RST | USART1 RST | TIM8 RST | SPI1 RST | TIM1 RST | ADC2 RST | ADC1 RST | IOPG RST | IOPF RST | IOPE RST | IOPD RST | IOPC RST | IOPB RST | IOPA RST | 保留 | AFIO RST |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

图 5-8 APB2RSTR 寄存器结构

从图 5-9 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达到配置不同波特率的目的。其各位描述如图 5-9 和表 5-3 所示。


图 5-9　USART_BRR 寄存器结构

**表 5-3　寄存器 USART_BRR 各位描述**

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

前面提到 STM32 的分数波特率概念，其实就在这个寄存器(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：

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

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 USART1→BRR 的值；反之，得到 USART1→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

$$USARTDIV = 72000000/(9600 \times 16) = 468.75$$

得到

$$DIV\_Fraction = 16 \times 0.75 = 12 = 0X0C$$

$$DIV\_Mantissa = 468 = 0X1D4$$

这样，就得到了 USART1→BRR 的值为 0X1D4C。只要设置串口 1 的 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 | | $f_{PCLK}$=36MHz | | | $f_{PCLK}$=72MHz | | |
|---|---|---|---|---|---|---|---|
| 序号 | kb/s | 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 |

续表

| 波　特　率 | | $f_{PCLK}$=36MHz | | | $f_{PCLK}$=72MHz | | |
|---|---|---|---|---|---|---|---|
| 序号 | kb/s | 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% |
| 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 |

**4. 串口控制**

STM32 的每个串口都有 3 个控制寄存器 USART_CR1～3，串口的很多配置都是通过这 3 个寄存器来设置。这里只要用到 USART_CR1 就可以实现功能了，该寄存器的各位描述如图 5-10 和表 5-5 所示。


图 5-10　USART_CR 寄存器结构

**表 5-5　USART_CR 寄存器各位描述**

| 位 | 描　　述 |
|---|---|
| 位 31:14 | 保留位，硬件强制为 0 |
| 位 13<br>UE | USART 使能，当该位被清 0，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 0<br>0：USART 分频器和输出被禁止<br>1：USART 模块使能 |
| 位 12<br>M | 字长，该位定义了数据字的长度，由软件对其设置和清 0<br>0：1 个起始位，8 个数据位，n 个停止位<br>1：1 个起始位，9 个数据位，n 个停止位<br>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<br>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 0<br>0：被空闲总线唤醒<br>1：被地址标记唤醒 |
| 位 10<br>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<br>0：禁止校验控制<br>1：使能校验控制 |

续表

| 位 | 描 述 |
| --- | --- |
| 位 9<br>PS | 校验选择，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<br>0：偶校验<br>1：奇校验 |
| 位 8<br>PEIE | PE 中断使能，该位由软件设置或清除<br>0：禁止产生中断<br>1：当 USART_SR 中的 PE 为“1”时，产生 USART 中断 |
| 位 7<br>TXEIE | 发送缓冲区空中断使能，该位由软件设置或清除<br>0：禁止产生中断<br>1：当 USART_SR 中的 TXE 为“1”时，产生 USART 中断 |
| 位 6<br>TCIE | 发送完成中断使能，该位由软件设置或清除<br>0：禁止产生中断<br>1：当 USART_SR 中的 TC 为“1”时，产生 USART 中断 |
| 位 5<br>RXNEIE | 接收缓冲区非空中断使能，该位由软件设置或清除<br>0：禁止产生中断<br>1：当 USART_SR 中的 ORE 或者 RXNE 为“1”时，产生 USART 中断 |
| 位 4<br>IDLEIE | IDLE 中断使能，该位由软件设置或清除<br>0：禁止产生中断<br>1：当 USART_SR 中的 IDLE 为“1”时，产生 USART 中断 |
| 位 3<br>TE | 发送使能，该位使能发送器。该位由软件设置或清除<br>0：禁止发送<br>1：使能发送<br>注意：<br>(1) 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)<br>(2) 当 TE 设置后，在真正发送开始之前，有 1 比特时间的延迟 |
| 位 2<br>RE | 接收使能，该位由软件设置或清除<br>0：禁止接收<br>1：使能接收，并开始搜寻 RX 引脚上的起始位 |
| 位 1<br>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 0<br>0：接收器处于正常工作模式<br>1：接收器处于静默模式<br>注意：<br>(1) 把 USART 置于静默模式(设置 RWU 位)之前，USART 要已经先接收了 1 字节数据；否则在静默模式下，不能被空闲总线检测唤醒<br>(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<br>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<br>0：没有发送断开字符<br>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。该寄存器的结构如图5-11所示。


图5-11 USART_DR寄存器结构

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)，另一个给接收用(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的结构如图5-12所示。


图5-12 USART_SR寄存器结构

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0，也可以向该位写0，直接清除。

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0方式：①读USART_SR，写USART_DR；②直接向该位写0。

通过以上一些寄存器的操作和I/O端口的配置，就可以达到串口最基本的配置。

## 5.3.2 库函数方式操作串口


视频讲解

通过以上寄存器的介绍，了解了 STM32 的 USART 寄存器模式的相关设置，接下来学习库函数操作 USART。表 5-6 给出了操作 USART 的库函数列表，重点介绍几个常用的函数。

**表 5-6 操作 USART 的库函数**

| 函 数 名 | 描 述 |
|---|---|
| USART_DeInit | 将外设 USARTx 寄存器重设为缺省值 |
| USART_Init | 根据 USART_InitStruct 中指定的参数初始化外设 USARTx 寄存器 |
| USART_StructInit | 把 USART_InitStruct 中的每一个参数按缺省值填入 |
| USART_Cmd | 使能或者失能 USART 外设 |
| USART_ITConfig | 使能或者失能指定的 USART 中断 |
| USART_DMACmd | 使能或者失能指定 USART 的 DMA 请求 |
| USART_SetAddress | 设置 USART 节点的地址 |
| USART_WakeUpConfig | 选择 USART 的唤醒方式 |
| USART_ReceiverWakeUpCmd | 检查 USART 是否处于静默模式 |
| USART_LINBreakDetectLengthConfig | 设置 USART LIN 中断检测长度 |
| USART_LINCmd | 使能或者失能 USARTx 的 LIN 模式 |
| USART_SendData | 通过外设 USARTx 发送单个数据 |
| USART_ReceiveData | 返回 USARTx 接收到的数据 |
| USART_SendBreak | 发送中断字 |
| USART_SetGuardTime | 设置指定的 USART 保护时间 |
| USART_SetPrescaler | 设置 USART 时钟预分频 |
| USART_SmartCardCmd | 使能或者失能指定 USART 的智能卡模式 |
| USART_SmartCardNackCmd | 使能或者失能 NACK 传输 |
| USART_HalfDuplexCmd | 使能或者失能 USART 半双工模式 |
| USART_IrDAConfig | 设置 USART IrDA 模式 |
| USART_IrDACmd | 使能或者失能 USART IrDA 模式 |
| USART_GetFlagStatus | 检查指定的 USART 标志位设置与否 |
| USART_ClearFlag | 清除 USARTx 的待处理标志位 |
| USART_GetITStatus | 检查指定的 USART 中断发生与否 |
| USART_ClearITPendingBit | 清除 USARTx 的中断待处理位 |

**1. USART_Init 函数**

初始化外设 USART 的函数为 USART_Init，具体的含义如表 5-7 所示。

**表 5-7 USART_Init 函数**

| 函 数 名 | USART_Init |
|---|---|
| 函数原形 | void USART_Init(USART_TypeDef * USARTx, USART_InitTypeDef * USART_InitStruct) |
| 功能描述 | 根据 USART_InitStruct 中指定的参数初始化外设 USARTx 寄存器 |
| 输入参数 1 | USARTx：选择 USART 外设，x 可以是 1、2 或者 3 |

续表

| 函　数　名 | USART_Init |
|---|---|
| 输入参数 2 | USART_InitStruct：指向结构 USART_InitTypeDef 的指针，包含了外设 USART 的配置信息 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

USART_InitTypeDef 的结构体定义于文件 stm32f10x_usart.h 中：

```
typedef struct
{
u32 USART_BaudRate;
u16 USART_WordLength;
u16 USART_StopBits;
u16 USART_Parity;
u16 USART_HardwareFlowControl;
u16 USART_Mode;
u16 USART_Clock;
u16 USART_CPOL;
u16 USART_CPHA;
u16 USART_LastBit;
} USART_InitTypeDef;
```

表 5-8 描述了结构 USART_InitTypeDef 在同步和异步模式下使用的不同成员。

**表 5-8　USART_InitTypeDef 成员 USART 模式对比**

| 成　　员 | 异 步 模 式 | 同 步 模 式 |
|---|---|---|
| USART_BaudRate | X | X |
| USART_WordLength | X | X |
| USART_StopBits | X | X |
| USART_Parity | X | X |
| USART_HardwareFlowControl | X | X |
| USART_Mode | X | X |
| USART_Clock | | X |
| USART_CPOL | | X |
| USART_CPHA | | X |
| USART_LastBit | | X |

（1）USART_BaudRate：该成员设置了 USART 传输的波特率。

（2）USART_WordLength：表示在一个帧中传输或者接收到的数据位数。表 5-9 给出了该参数可取的值。

**表 5-9　USART_WordLength 定义**

| USART_WordLength | 描　　述 |
|---|---|
| USART_WordLength_8b | 8 位数据 |
| USART_WordLength_9b | 9 位数据 |

（3）USART_StopBits：定义了发送的停止位数目。表 5-10 给出了该参数可取的值。

表 5-10 USART_StopBits 定义

| USART_StopBits | 描 述 |
|---|---|
| USART_StopBits_1 | 在帧结尾传输 1 个停止位 |
| USART_StopBits_0.5 | 在帧结尾传输 0.5 个停止位 |
| USART_StopBits_2 | 在帧结尾传输 2 个停止位 |
| USART_StopBits_1.5 | 在帧结尾传输 1.5 个停止位 |

(4) USART_Parity：定义了奇偶模式。表 5-11 给出了该参数可取的值。

表 5-11 USART_Parity 定义

| USART_Parity | 描 述 |
|---|---|
| USART_Parity_No | 奇偶失能 |
| USART_Parity_Even | 偶模式 |
| USART_Parity_Odd | 奇模式 |

**注意** 奇偶校验一旦使能，在发送数据的 MSB 位插入经计算的奇偶位(字长 9 位时的第 9 位，字长 8 位时的第 8 位)。

(5) USART_HardwareFlowControl：指定了硬件流控制模式使能还是失能。表 5-12 给出了该参数可取的值。

表 5-12 USART_HardwareFlowControl 定义

| USART_HardwareFlowControl | 描 述 |
|---|---|
| USART_HardwareFlowControl_None | 硬件流控制失能 |
| USART_HardwareFlowControl_RTS | 发送请求 RTS 使能 |
| USART_HardwareFlowControl_CTS | 清除发送 CTS 使能 |
| USART_HardwareFlowControl_RTS_CTS | RTS 和 CTS 使能 |

(6) USART_Mode：指定了使能或者失能发送和接收模式。表 5-13 给出了该参数可取的值。

表 5-13 USART_Mode 定义

| USART_Mode | 描 述 |
|---|---|
| USART_Mode_Tx | 发送使能 |
| USART_Mode_Rx | 接收使能 |

(7) USART_CLOCK：表示 USART 时钟使能还是失能。表 5-14 给出了该参数可取的值。

表 5-14 USART_CLOCK 定义

| USART_CLOCK | 描 述 |
|---|---|
| USART_Clock_Enable | 时钟高电平活动 |
| USART_Clock_Disable | 时钟低电平活动 |

(8) USART_CPOL：表示 SLCK 引脚上时钟输出的极性。表 5-15 给出了该参数可取的值。

表 5-15 USART_CPOL 定义

| USART_CPOL | 描 述 |
|---|---|
| USART_CPOL_High | 时钟高电平 |
| USART_CPOL_Low | 时钟低电平 |

(9) USART_CPHA：表示 SLCK 引脚上时钟输出的相位，和 CPOL 位一起配合产生用户希望的时钟/数据的采样关系。表 5-16 给出了该参数可取的值。

表 5-16 USART_CPHA 定义

| USART_CPHA | 描 述 |
|---|---|
| USART_CPHA_1Edge | 时钟第一个边沿进行数据捕获 |
| USART_CPHA_2Edge | 时钟第二个边沿进行数据捕获 |

(10) USART_LastBit：控制是否在同步模式下，在 SCLK 引脚上输出后发送的那个数据字(MSB)对应的时钟脉冲。表 5-17 给出了该参数可取的值。

表 5-17 USART_LastBit 定义

| USART_LastBit | 描 述 |
|---|---|
| USART_LastBit_Disable | 后一位数据的时钟脉冲不从 SCLK 输出 |
| USART_LastBit_Enable | 后一位数据的时钟脉冲从 SCLK 输出 |

**2. 串口复位函数 USART_DeInit**

表 5-18 描述了函数 USART_DeInit 的具体含义。

表 5-18 USART_DeInit 函数

| 函 数 名 | USART_DeInit |
|---|---|
| 函数原形 | void USART_DeInit(USART_TypeDef * USARTx) |
| 功能描述 | 将外设 USARTx 寄存器重设为缺省值 |
| 输入参数 | USARTx：x 可以是 1、2 或 3，来选择 USART 外设 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | RCC_APB2PeriphResetCmd()，RCC_APB1PeriphResetCmd() |

**3. 使能或者失能 USART 外设函数 USART_Cmd**

描述使能或者失能 USART 外设的函数为 USART_Cmd，具体含义如表 5-19 所示。

表 5-19 USART_Cmd 函数

| 函 数 名 | USART_Cmd |
|---|---|
| 函数原形 | void USART_Cmd(USART_TypeDef * USARTx, FunctionalState NewState) |
| 功能描述 | 使能或者失能 USART 外设 |
| 输入参数 1 | USARTx：x 可以是 1、2 或 3，用来选择 USART 外设 |
| 输入参数 2 | NewState：外设 USARTx 新状态参数可以取 ENABLE 或者 DISABLE |
| 输出参数 | 无 |
| 返回值 | 无 |

续表

| 函 数 名 | USART_Cmd |
| --- | --- |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**4. 使能或者失能指定的 USART 中断函数 USART_ITConfig**

使能或者失能指定的 USART 中断函数为 USART_ITConfig,具体含义如表 5-20 所示。

**表 5-20 USART_ITConfig 函数**

| 函 数 名 | USART_ITConfig |
| --- | --- |
| 函数原形 | void USART _ ITConfig (USART _ TypeDef * USARTx, u16 USART _ IT, FunctionalState NewState) |
| 功能描述 | 使能或者失能指定的 USART 中断 |
| 输入参数 1 | USARTx: x 可以是 1、2 或 3,用来选择 USART 外设 |
| 输入参数 2 | USART_IT: 待使能或者失能的 USART 中断源参阅表 5-21: USART_IT 查阅更多该参数允许取值范围 |
| 输入参数 3 | NewState: USARTx 中断的新状态参数可以取 ENABLE 或者 DISABLE |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

输入参数 USART_IT 含义是使能或者失能 USART 的中断。可以把表 5-21 中的一个或者多个取值的组合作为该参数的值。

**表 5-21 USART_IT 值**

| USART_IT | 描 述 |
| --- | --- |
| USART_IT_PE | 奇偶错误中断 |
| USART_IT_TXE | 发送中断 |
| USART_IT_TC | 传输完成中断 |
| USART_IT_RXNE | 接收中断 |
| USART_IT_IDLE | 空闲总线中断 |
| USART_IT_LBD | LIN 中断检测中断 |
| USART_IT_CTS | CTS 中断 |
| USART_IT_ERR | 错误中断 |

**5. 使能或者失能指定 USART 的 DMA 请求函数 USART_DMACmd**

使能或者失能指定 USART 的 DMA 请求函数为 USART_DMACmd,具体含义如表 5-22 所示。

**表 5-22 函数 USART_DMACmd**

| 函 数 名 | USART_DMACmd |
| --- | --- |
| 函数原形 | void USART_DMACmd (USART_TypeDef * USARTx, uint16_t USART_DMAReq, FunctionalState NewState) |

续表

| 函　数　名 | USART_DMACmd |
|---|---|
| 功能描述 | 使能或者失能指定 USART 的 DMA 请求 |
| 输入参数 1 | USARTx：选择 USART 外设，x 可以是 1、2 或 3 |
| 输入参数 2 | USART_DMAreq：指定 DMA 请求 |
| 输入参数 3 | NewState：USARTx DMA 请求源的新状态，这个参数可以取 ENABLE 或者 DISABLE |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

USART_DMAreq 选择待使能或者失能的 DMA 请求。表 5-23 给出了该参数可取的值。

**表 5-23　USART_LastBit 值**

| USART_DMAreq | 描　　述 |
|---|---|
| USART_DMAReq_Tx | 发送 DMA 请求 |
| USART_DMAReq_Rx | 接收 DMA 请求 |

**6. 通过外设 USARTx 发送单个数据函数 USART_SendData**

通过外设 USARTx 发送单个数据函数为 USART_SendData，具体含义如表 5-24 所示。

**表 5-24　USART_SendData 函数**

| 函　数　名 | USART_SendData |
|---|---|
| 函数原形 | void USART_SendData(USART_TypeDef * USARTx, u8 Data) |
| 功能描述 | 通过外设 USARTx 发送单个数据 |
| 输入参数 1 | USARTx：选择 USART 外设，x 可以是 1、2 或 3 |
| 输入参数 2 | Data：待发送的数据 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**7. USART 收到数据函数 USART_ReceiveData**

表 5-25 描述了串口接收函数 USART_ReceiveData。

**表 5-25　USART_ReceiveData 函数**

| 函　数　名 | USART_ReceiveData |
|---|---|
| 函数原形 | u8 USART_ReceiveData(USART_TypeDef * USARTx) |
| 功能描述 | 返回 USARTx 接收到的数据 |
| 输入参数 | USARTx：选择 USART 外设，x 可以是 1、2 或 3 |
| 输出参数 | 无 |
| 返回值 | 接收到的字 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**8. 检查指定的 USART 标志位设置与否函数 USART_GetFlagStatus**

检查指定的 USART 标志位设置与否函数为 USART_GetFlagStatus，具体含义如表 5-26 所示。

**表 5-26　USART_GetFlagStatus 函数**

| 函　数　名 | USART_GetFlagStatus |
|---|---|
| 函数原形 | FlagStatus USART_GetFlagStatus(USART_TypeDef * USARTx, u16 USART_FLAG) |
| 功能描述 | 检查指定的 USART 标志位设置与否 |
| 输入参数 1 | USARTx：选择 USART 外设，x 可以是 1、2 或 3 |
| 输入参数 2 | USART_FLAG：待检查的 USART 标志位 |
| 输出参数 | 无 |
| 返回值 | USART_FLAG 的新状态(SET 或者 RESET) |
| 先决条件 | 无 |
| 被调用函数 | 无 |

表 5-27 给出了所有可以被函数 USART_GetFlagStatus 检查的标志位列表。

**表 5-27　USART_FLAG 值**

| USART_FLAG | 描　　述 |
|---|---|
| USART_FLAG_CTS | CTS 标志位 |
| USART_FLAG_LBD | LIN 中断检测标志位 |
| USART_FLAG_TXE | 发送数据寄存器空标志位 |
| USART_FLAG_TC | 发送完成标志位 |
| USART_FLAG_RXNE | 接收数据寄存器非空标志位 |
| USART_FLAG_IDLE | 空闲总线标志位 |
| USART_FLAG_ORE | 溢出错误标志位 |
| USART_FLAG_NE | 噪声错误标志位 |
| USART_FLAG_FE | 帧错误标志位 |
| USART_FLAG_PE | 奇偶错误标志位 |

**9. 清除 USARTx 的待处理标志位函数 USART_ClearFlag**

清除 USARTx 的待处理标志位函数为 USART_ClearFlag，具体含义如表 5-28 所示。

**表 5-28　USART_ClearFlag 函数**

| 函　数　名 | USART_ClearFlag |
|---|---|
| 函数原形 | void USART_ClearFlag(USART_TypeDef * USARTx, u16 USART_FLAG) |
| 功能描述 | 清除 USARTx 的待处理标志位 |
| 输入参数 1 | USARTx：选择 USART 外设，x 可以是 1、2 或 3 |
| 输入参数 2 | USART_FLAG：待清除的 USART 标志位 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**10. 检查指定的 USART 中断发生与否函数 USART_GetITStatus**

检查指定的 USART 中断发生与否函数为 USART_GetITStatus，具体含义如表 5-29 所示。

表 5-29 USART_GetITStatus 函数

| 函 数 名 | USART_GetITStatus |
|---|---|
| 函数原形 | ITStatus USART_GetITStatus(USART_TypeDef * USARTx, u16 USART_IT) |
| 功能描述 | 检查指定的 USART 中断发生与否 |
| 输入参数 1 | USARTx：选择 USART 外设，x 可以是 1、2 或 3 |
| 输入参数 2 | USART_IT：待检查的 USART 中断源 |
| 输出参数 | 无 |
| 返回值 | USART_IT 的新状态 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

表 5-30 给出了所有可以被函数 USART_GetITStatus 检查的中断标志位列表。

表 5-30 USART_IT 值

| USART_IT | 描 述 |
|---|---|
| USART_IT_PE | 奇偶错误中断 |
| USART_IT_TXE | 发送中断 |
| USART_IT_TC | 发送完成中断 |
| USART_IT_RXNE | 接收中断 |
| USART_IT_IDLE | 空闲总线中断 |
| USART_IT_LBD | LIN 中断探测中断 |
| USART_IT_CTS | CTS 中断 |
| USART_IT_ORE | 溢出错误中断 |
| USART_IT_NE | 噪声错误中断 |
| USART_IT_FE | 帧错误中断 |

**11. 清除 USARTx 的中断待处理位函数 USART_ClearITPendingBit**

清除 USARTx 的中断待处理位函数为 USART_ClearITPendingBit，具体含义如表 5-31 所示。

表 5-31 USART_ClearITPendingBit 函数

| 函 数 名 | USART_ClearITPendingBit |
|---|---|
| 函数原形 | void USART_ClearITPendingBit(USART_TypeDef * USARTx, u16 USART_IT) |
| 功能描述 | 清除 USARTx 的中断待处理位 |
| 输入参数 1 | USARTx：选择 USART 外设，x 可以是 1、2 或 3 |
| 输入参数 2 | USART_IT：待检查的 USART 中断源 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |


视频讲解

## 5.3.3 串口设置步骤

串口设置的一般步骤如下。

(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) 开启串口中断并且初始化 NVIC(如果需要开启中断才需要这个步骤)。

(6) 使能串口。

(7) 编写串口中断处理函数。

## 5.4　串口通信操作实例


视频讲解

下面两段代码,简单地完成了串口通信功能,主程序主要完成串口的初始化然后打印两条信息,编写代码的方式仍然是先写主函数,然后调用子函数,这么编写代码含义清晰,容易理解和分层设计。

### 5.4.1　主程序

主程序主要完成串口代码初始化(第 6 行),然后打印两条信息说明串口配置正确(第 7、8 行),串口主程序代码如下:

```
1.  #include "stm32f10x.h"
2.  #include "usart1.h"
3.  int main(void)
4.  {
5.              /*USART1 config 115200 8-N-1 */
6.              USART1_Config();
7.              printf("\r\n this is a usart printf test \r\n");
8.              printf("\r\n 欢迎您来到哈尔滨! \r\n");
9.              for(;;)
10.             {
11.             }
12. }
```

### 5.4.2　串口初始化代码

串口初始化代码如下:

```
1.  #include "usart1.h"
2.  void USART1_Config(void)
3.  {
4.          GPIO_InitTypeDef GPIO_InitStructure;
5.          USART_InitTypeDef USART_InitStructure;
6.          RCC_APB2PeriphClockCmd( RCC_APB2Periph_USART1|
                                    RCC_APB2Periph_GPIOA,ENABLE);
7.          GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
8.          GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
9.          GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
10.         GPIO_Init(GPIOA,&GPIO_InitStructure);
11.         GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
```

```
12.            GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
13.            GPIO_Init(GPIOA,&GPIO_InitStructure);
14.            USART_InitStructure.USART_BaudRate = 115200;
15.            USART_InitStructure.USART_WordLength = USART_WordLength_8b;
16.            USART_InitStructure.USART_StopBits = USART_StopBits_1;
17.            USART_InitStructure.USART_Parity = USART_Parity_No;
18.            USART_InitStructure.USART_HardwareFlowControl =
                                  USART_HardwareFlowControl_None;
19.            USART_InitStructure.USART_Mode = USART_Mode_Rx|
                                                  USART_Mode_Tx;
20.            USART_Init(USART1,&USART_InitStructure);
21.            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 1 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 行配置 USART1 时钟。

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

第 11～13 行配置 USART1 Rx(PA.10)为浮空输入。

第 14～21 行配置 USART1 模式波特率 115200，数据传输 8 位，停止位 1，无奇偶校验，无硬件流控制。

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

**1. 串口时钟使能**

串口是挂载在 APB2 总线下面的外设，所以使能函数为

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

**2. 串口复位**

当外设出现异常时可以通过复位设置，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。一般在系统刚开始配置外设时，都会先执行复位该外设的操作。复位是在函数 USART_DeInit()中完成：

```
void USART_DeInit(USART_TypeDef * USARTx);
```

比如要复位串口 1，方法为

```
USART_DeInit(USART1);
```

**3. 串口参数初始化**

串口初始化是通过 USART_Init()函数实现的：

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

这个函数的第一个入口参数是指定初始化的串口标号，这里选择 USART1。

第二个入口参数是一个 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

(1) 设置波特率：

```
USART_InitStructure.USART_BaudRate = bound;
```

(2) 设置字长：

```
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
```

(3) 设置停止位：

```
USART_InitStructure.USART_StopBits = USART_StopBits_1;
```

(4) 设置奇偶校验位：

```
USART_InitStructure.USART_Parity = USART_Parity_No;
```

(5) 设置件数据流控制：

```
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
```

(6) 设置收发模式：

```
USART_InitStructure.USART_Mode = USART_Mode_Rx|USART_Mode_Tx;
```

(7) 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)。读者可以根据需要设置这些参数。

**4. 数据发送与接收**

STM32 的发送与接收是通过数据寄存器 USART_DR 来实现的，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，当收到数据时，也是存在该寄存器内。

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。

在固件库函数里面，读取串口状态的函数是：

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

这个函数的第二个入口参数非常关键，用于标识要查看串口的状态，比如上面讲解的RXNE(读数据寄存器非空)以及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);
```

要判断发送是否完成(TC)，操作库函数的方法是：

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

这些标识号在MDK里面是通过宏定义定义的：

```
#define USART_IT_PE                          ((uint16_t)0x0028)
#define USART_IT_TXE                         ((uint16_t)0x0727)
#define USART_IT_TC                          ((uint16_t)0x0626)
#define USART_IT_RXNE                        ((uint16_t)0x0525)
#define USART_IT_IDLE                        ((uint16_t)0x0424)
#define USART_IT_LBD                         ((uint16_t)0x0846)
#define USART_IT_CTS                         ((uint16_t)0x096A)
#define USART_IT_ERR                         ((uint16_t)0x0060)
#define USART_IT_ORE                         ((uint16_t)0x0360)
#define USART_IT_NE                          ((uint16_t)0x0260)
#define USART_IT_FE                          ((uint16_t)0x0160)
```

**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，使用的方法是：

```
USART_Cmd(USART1,ENABLE);
```

**7. 开启串口响应中断**

当开启串口中断时，还需要使能串口中断，使能串口中断的函数是：

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

这个函数的第二个入口参数是标识使能串口的类型，也就是使能哪种中断，因为串口的中断类型有很多种。比如在接收到数据的时候(RXNE，读数据寄存器非空)，要产生中断，那么开启中断的方法是：

```
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，那么产生中断的方法是：

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

**8. 获取相应中断状态**

当使能了某个中断，该中断发生了的时候，就会设置状态寄存器中的某个标志位。经常在中断处理函数中，要判断该中断是哪种中断，使用的函数是：

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

比如使能了串口发送完成中断，那么当中断发生了，便可以在中断处理函数中调用这个函数来判断到底是否是串口发送完成中断，方法是：

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

返回值是 SET，说明串口发送完成中断发生。

串口代码实验现象如图 5-13 所示。

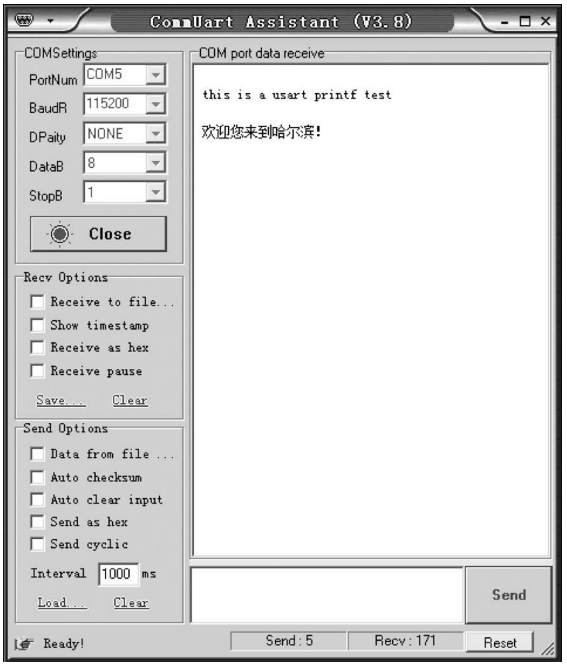


图 5-13　串口代码实验现象

## 5.5　本章小结

本章主要讲述了 STM32 串口通信的基本方法，通过第 4、5 章的学习，对 STM32 的控制方法有了一定了解，注意操作步骤之间的相同点和不同点。

本章例子是通过 printf 输出到串口，实际上读者可以直接使用 USART_SendData (USART1,(uint8_t) ch)函数完成发送数据，读者思考一下，区别在哪里？读者应详细掌握串口配置每个步骤使用的函数和方法，后续其他功能还会用到大量的 STM32 库函数，读者应掌握如何使用官方库函数的方法。

## 5.6　习题

(1) RS232 有哪些接线方式？

(2) 串口波特率的计算方法是什么？

(3) 简述 STM32 采用库函数方式操作串口的步骤。

(4) 如何同时打开两个终端实现串口之间数据通信？