第6章 STM32通用同步/异步通信USART

退出

6.1 数据通信基本概念

并行通信与串行通信

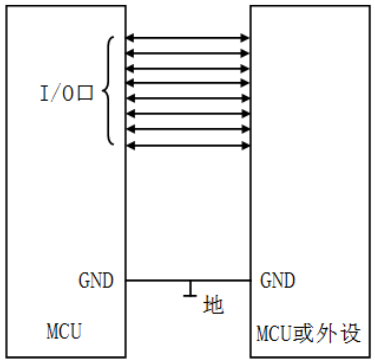
计算机的CPU与外部设备之间的信息交换，以及计算机与计算机之间的信息交换过程称为通信。

并行通信：是指使用多条数据线传输数据，数据各个位同时在不同的数据线上传送。并行通信时，数据可以字或字节为单位并行进行传输。

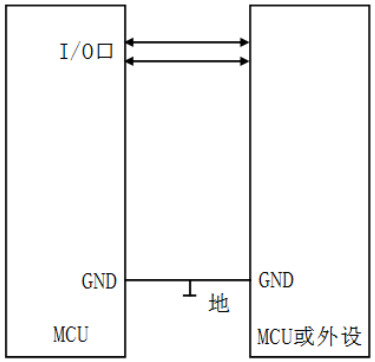
---优点：速度快　　---缺点：占用引脚资源多

串行通信：是指使用一条数据线，数据按位顺序传输。

---优点：占用引脚资源少　　---缺点：速度相对较慢

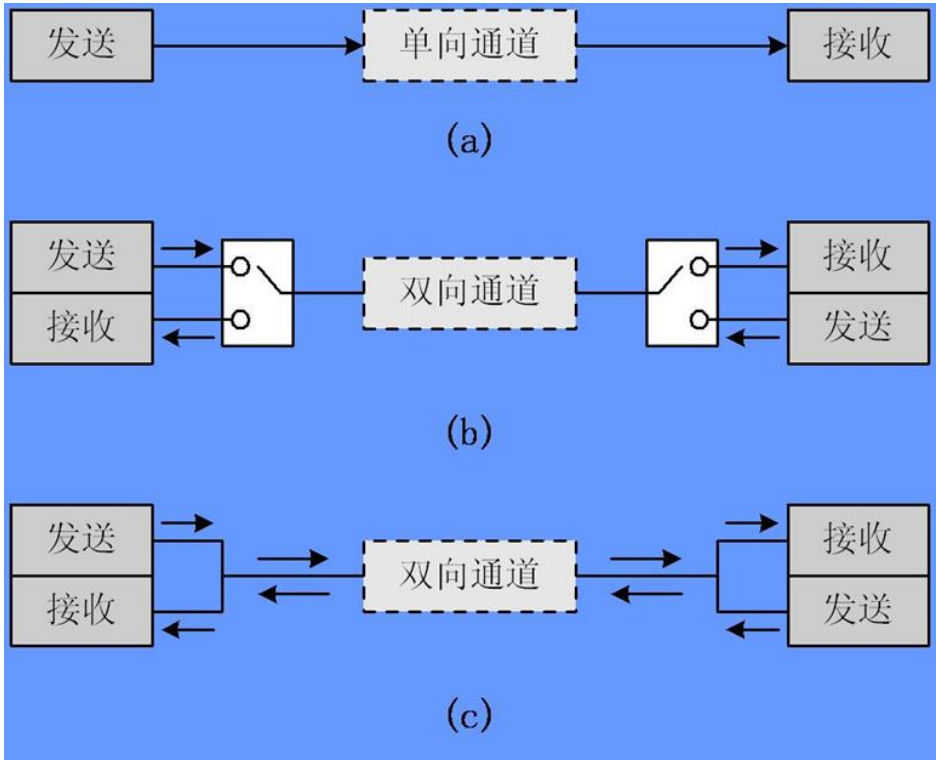


(a) 并行通信



(b) 串行通讯

串行通信的制式



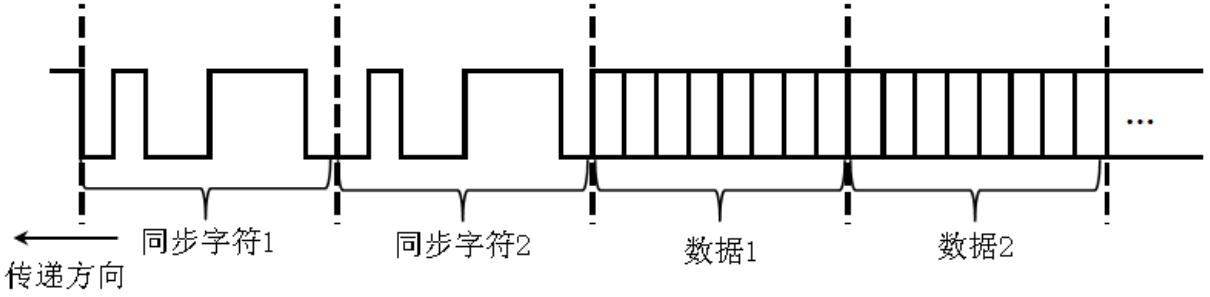
单工制式（Simplex）是指甲乙双方通信时只能单向传送数据。系统组成以后，发送方和接收方固定。

半双工制式（Half Duplex）是指通信双方都具有发送器和接收器，既可发送也可接收，但不能同时接收和发送，发送时不能接收，接收时不能发送。即在某一时刻，只允许数据在一个方向上传输，它实际上是一种切换方向的单工通信。

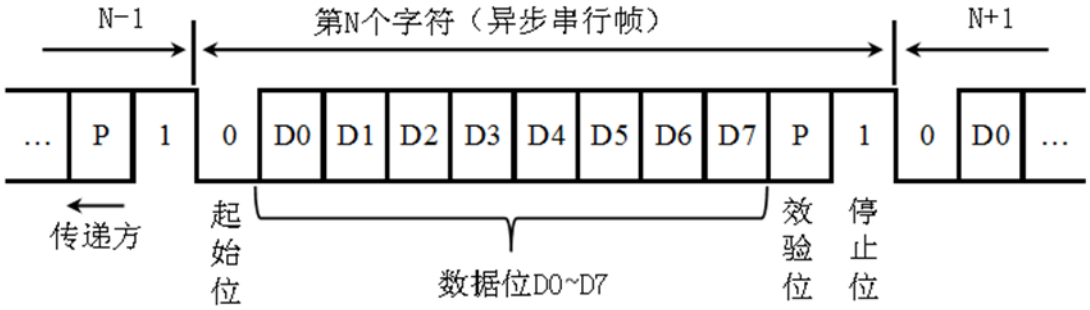
全双工制式（Full Duplex）是指通信双方均设有发送器和接收器，并且信道划分为发送信道和接收信道，因此全双工制式可实现甲方（乙方）同时发送和接收数据，发送时能接收，接收时也能发送。

异步通信与同步通信

同步通信：同步通信是由1~2个同步字符和多字节数据位组成，同步字符作为起始位以触发同步时钟开始发送或接受数据；多字节数据之间不允许有空隙，每位占用的时间相等。通过专用时钟控制线实现时钟同步信号传输。



异步通信：异步通信数据传送按帧传输，一帧数据包含起始位、数据位、校验位和停止位。异步通信不带时钟同步信号。



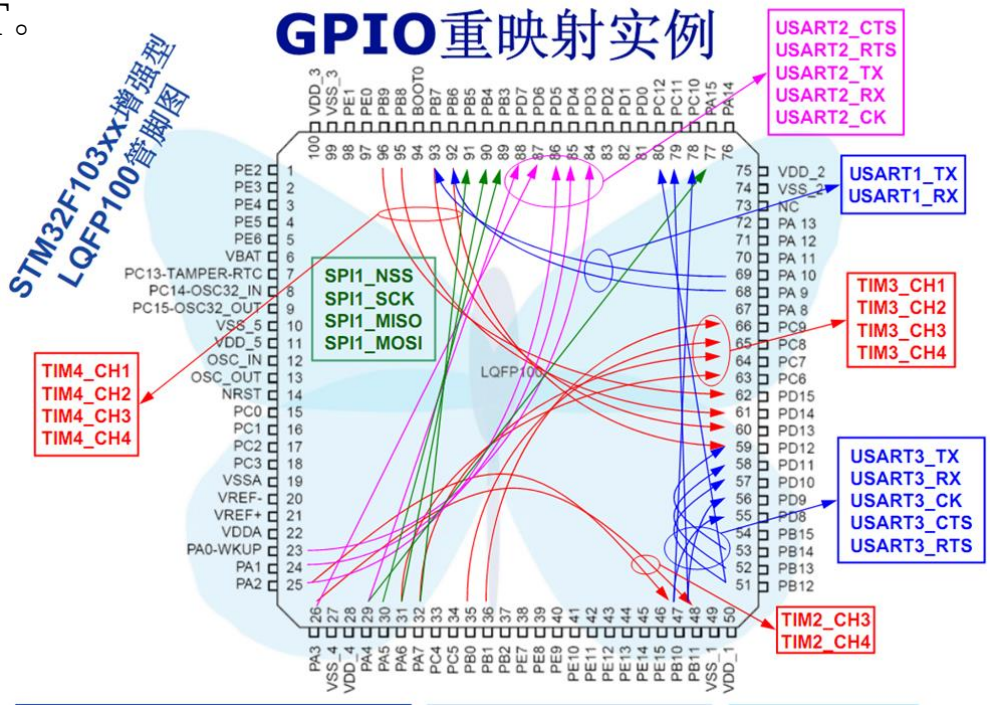
6.2 STM32的USART的结构及工作方式

STM32的串口通信接口

- UART：通用异步收发器
- USART：通用同步异步收发器

STM32F103微控制器的小容量产品有2个USART，中等容量产品有3个USART，大容量产品包含3个USART+2个UART。

串口号	RXD	TXD
1	PA10	PA9
2	PA3	PA2
3	PB11	PB10
4	PC11	PC10
5	PD2	PC12

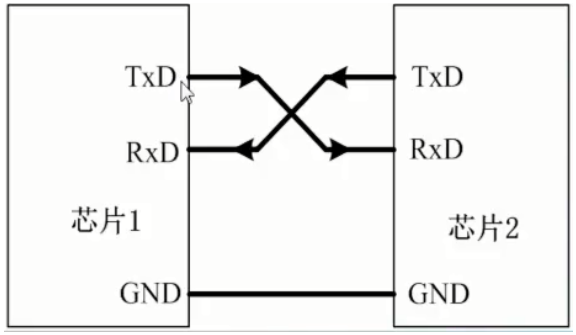


USART介绍

通用同步异步收发器（Universal Synchronous/Asynchronous Receiver/Transmitter，**USART**）是一个全双工通用同步/异步串行收发模块。

- 全双工操作（相互独立的接收数据和发送数据）；
- 同步操作时，可主机时钟同步，也可从机时钟同步；
- 高精度波特率发生器，发送和接受共用的可编程波特率，最高4.5Mbits/s；
- 可编程的数据字长度（8位或者9位）；
- 可配置的停止位（支持1或者2位停止位）；
- 单独的发送器和接收器使能位；
- 检测标志：① 接受缓冲器 ②发送缓冲器空 ③传输结束标志；
- 多个带标志的中断源。触发中断；
- 其他：校验控制，四个错误检测标志。

STM32 的 USART 结构



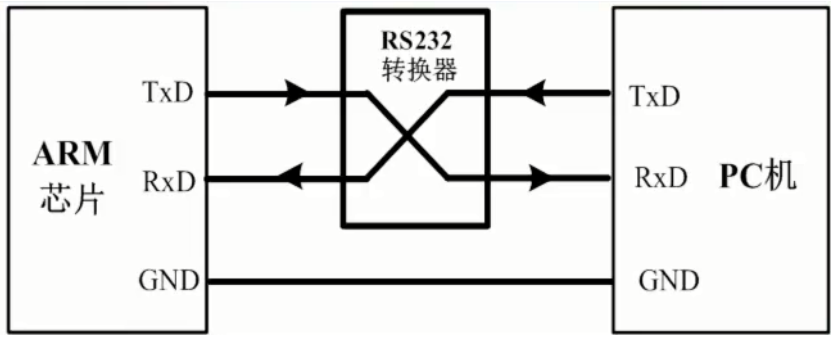
- **外部引脚：**

  接收数据输入（RX）和发送数据输出（TX）

  清除发送（nCTS）

  发送请求（nRTS）

  发送器时钟输出（CK）



- **内部包括：**

  移位寄存器

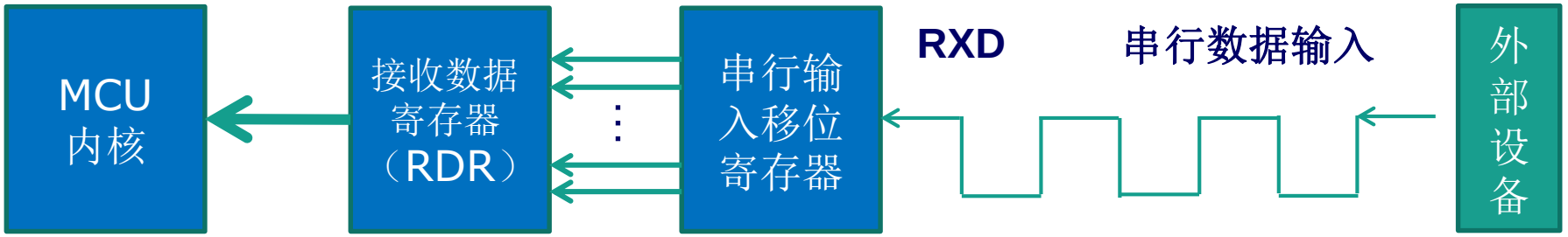
  发送数据寄存器（TDR）和 接收数据寄存器（RDR）

  IrDA 串行红外编解码模块

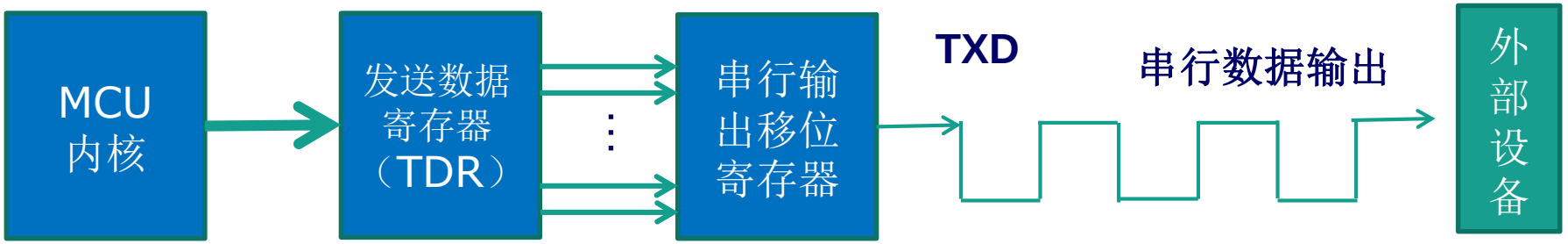
  硬件数据流控制器、时钟控制、发送控制、唤醒单元、接收控制、中断控制和波特率控制等。

USART 通信过程

数据接收过程：



数据发送过程：



STM32串口异步通信需要定义的参数

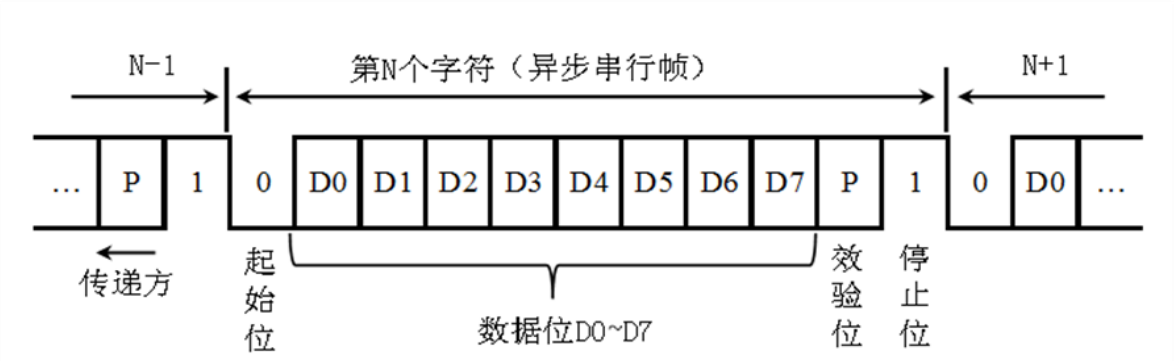
① 起始位

② 数据位（8位或者9位）

③ 奇偶校验位（第9位）

④ 停止位（1,1.5, 2位）

⑤ 波特率设置



串行通信的校验

◆ **奇偶校验**

奇偶校验在发送数据时，数据位尾随的1的位数为奇偶校验位（1或0），当设置为奇校验时，数据中1的个数与校验位1的个数之和应为奇数；当设置为偶校验时，数据中1的个数与校验位中的1的个数之和应为偶数。

◆ **累加和校验**

累加和校验是指发送方将所发送的数据块求和，并将“校验和”附加到数据块末尾。接收方接收数据时也是对数据块求和，将所得结果与发送方的“校验和”进行比较，相符则无差错，否则即出现了差错。

◆ **循环冗余码校验**

循环冗余码校验（Cyclic Redundancy Check，简称CRC）的基本原理是将一个数据块看成一个位数很长的二进制数，然后用一个特定的数去除它，将余数作校验码附在数据块后一起发送。

串行通信的波特率

波特率是串行通信中一个重要概念，是指传输数据的速率。波特率（bit per second）的定义是每秒传输数据的位数，即：

1波特=1位/秒（1bps）

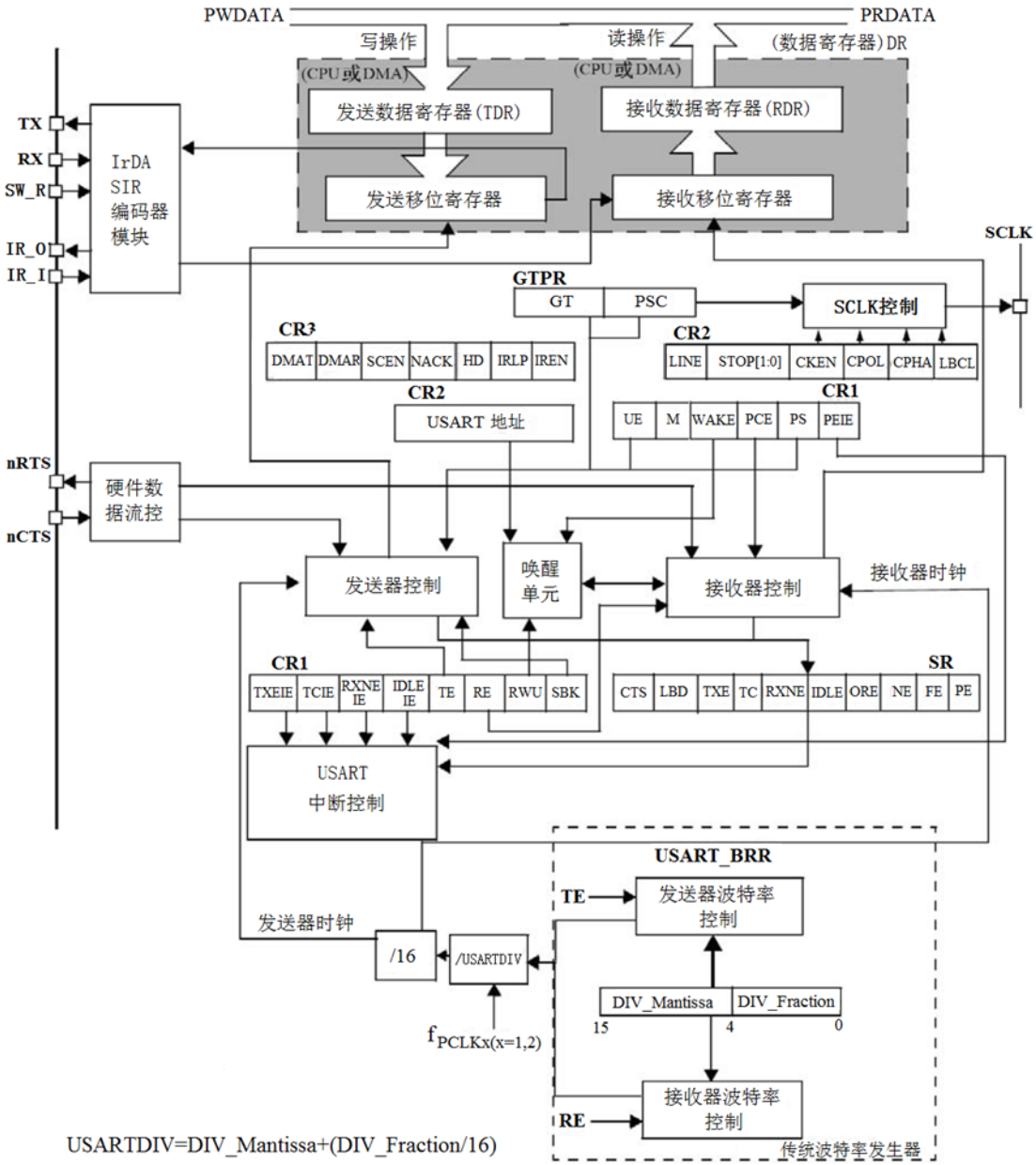
波特率的倒数即为每位传输所需的时间。

由以上串行通信原理可知，**互相通信的甲乙双方必须具有相同的波特率**，否则无法成功地完成串行数据通信。

STM32 的 USART 结构

USART相关寄存器

- **状态寄存器(USART_SR)**
- **数据寄存器(USART_DR)**
- **波特率寄存器(USART_BRR)**
- 控制寄存器 1(USART_CR1)
- 控制寄存器 2(USART_CR2)
- 控制寄存器 3(USART_CR3)
- 保护时间和预分频寄存器(USART_GTPR)

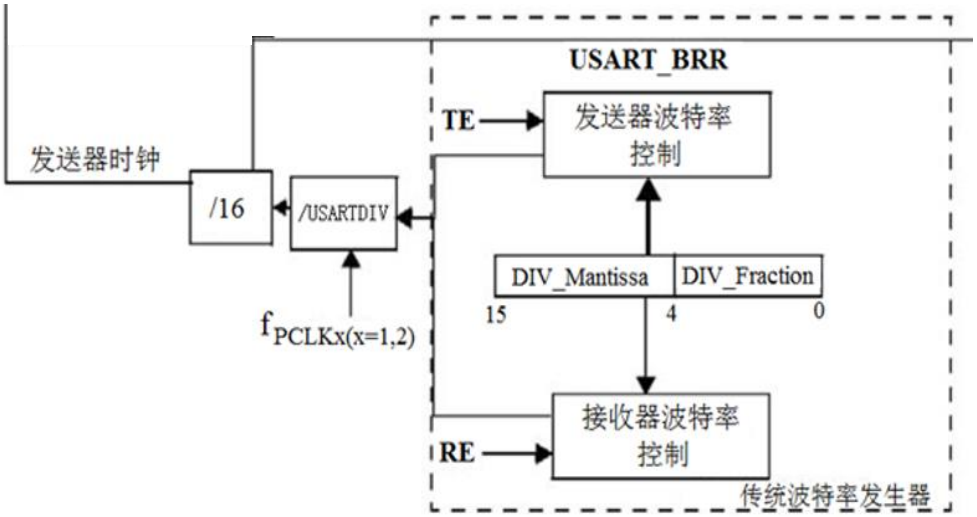


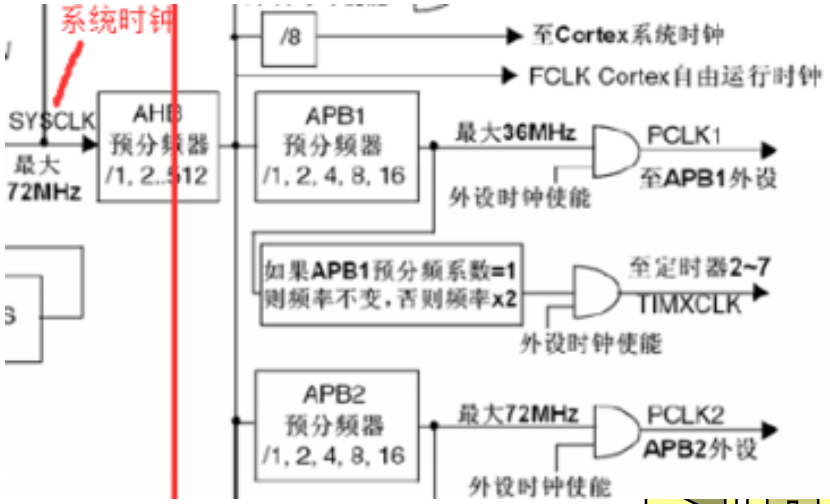
波特率的设置方法

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

上式中，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。我们只要得到 USARTDIV 的值，就可以得到串口波特率寄存器 USART1->BRR 的值，反过来，我们得到 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，因为一般我们知道的是波特率，和 PCLKx 的时钟，要求的就是 USART_BRR 的值。





波特率的设置方法

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

假设USART1波特率要设置为115200bps，而PCLK2的时钟为72MHz，根据上面的公式：

USARTDIV = 72000000/(115200*16) = 39.0625

在USART-BRR中，DIV_Mantissa[15:4] 位定义 USARTDIV 整数部分，DIV_Fraction[3:0]位定义 USARTDIV 的小数部分。那么得到：

DIV_Mantissa = 39 = 0X27

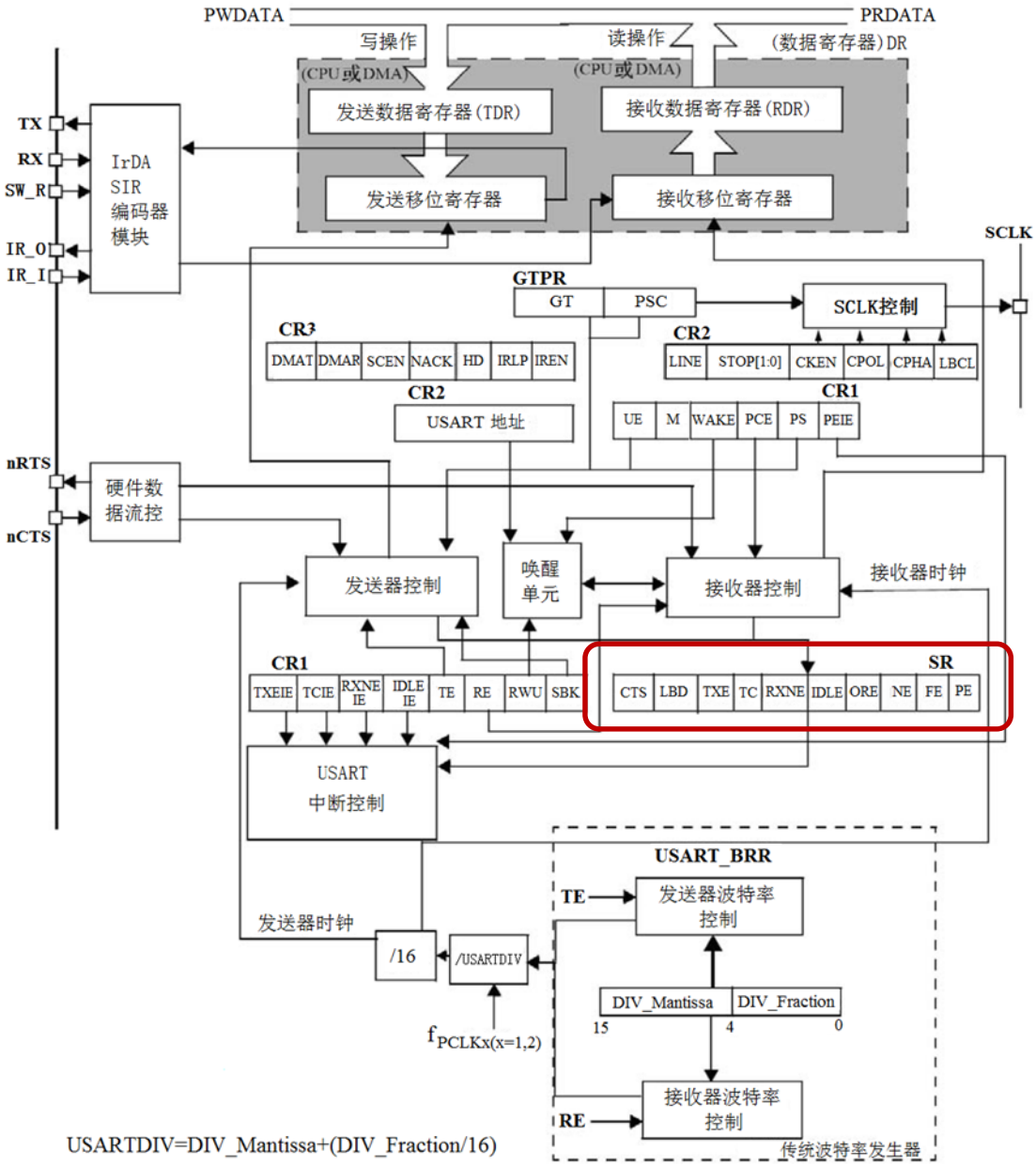
DIV_Fraction = 16*0.0625 = 1 = 0X01

这样，可得USART-BRR的值为 0X0271。即：只要将USART-BRR的值为 0X0271，就可以得到115200的波特率。

STM32 的 USART 结构

USART相关寄存器

- **状态寄存器(USART_SR)**
- **数据寄存器(USART_DR)**
- **波特率寄存器(USART_BRR)**
- 控制寄存器 1(USART_CR1)
- 控制寄存器 2(USART_CR2)
- 控制寄存器 3(USART_CR3)
- 保护时间和预分频寄存器(USART_GTPR)

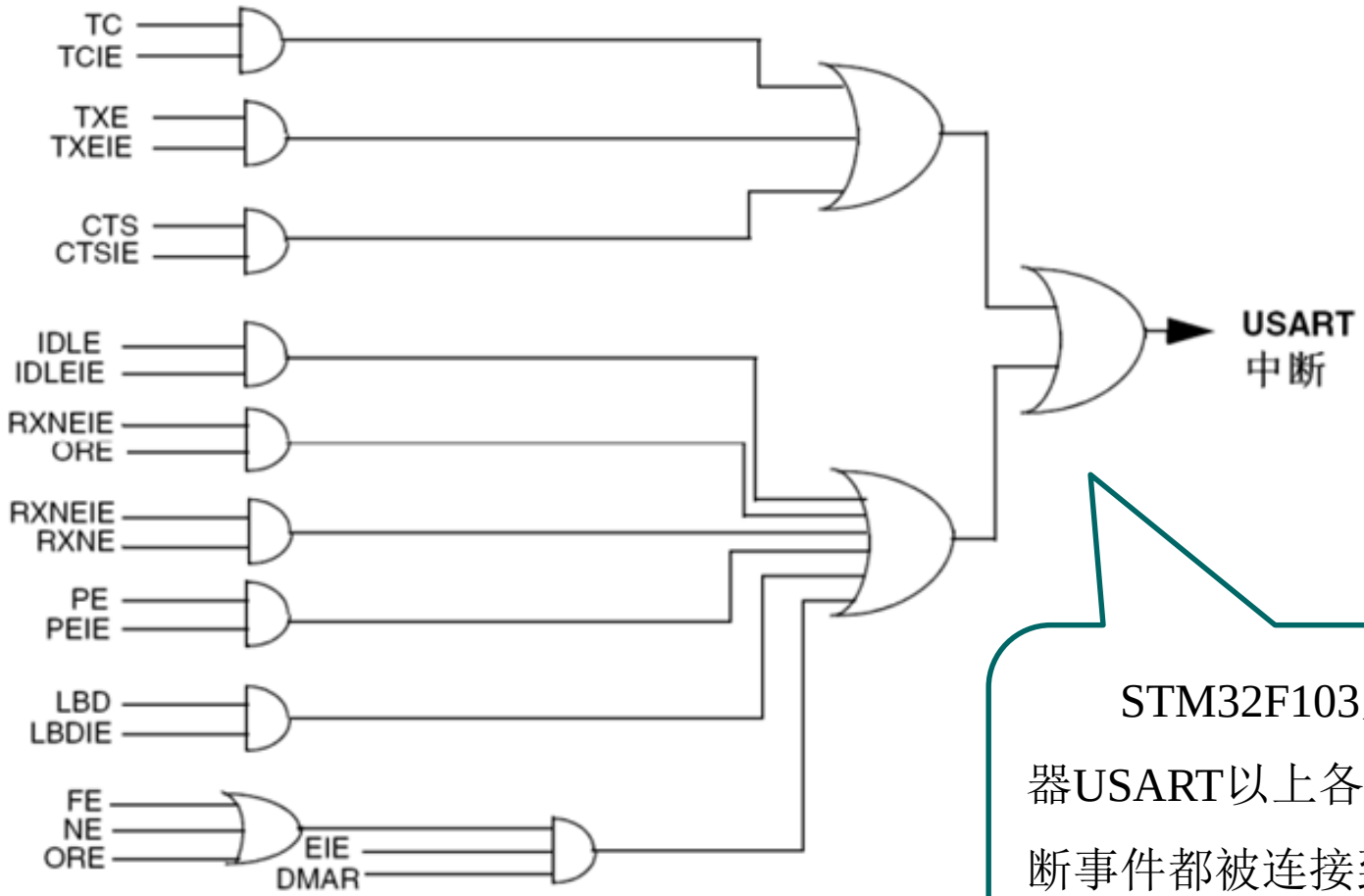


USART中断

发送期间：发送完成（TC）、清除发送（CTS）、发送数据寄存器空（TXE）；

接收期间：空闲总线检测（IDLE）、溢出错误（ORE）、接收数据寄存器非空（RXNE）、校验错误（PE）、LIN断开检测（LBD）、噪声错误（NE，仅在多缓冲器通信）和帧错误（FE，仅在多缓冲器通信）。

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	RXNE	RXNEIE
检测到数据溢出	ORE	RXNEIE
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、溢出错误和帧错误	NE或ORT或FE	EIE



STM32F103系列微控制器USART以上各种不同的中断事件都被连接到同一个中断向量。

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	连到EXTI的RTC闹钟中断	0x0000_00E4
42	49	可设置	USB唤醒	连到EXTI的从USB待机唤醒中断	0x0000_00E8
43	50	可设置	TIM8_BRK	TIM8刹车中断	0x0000_00EC
44	51	可设置	TIM8_UP	TIM8更新中断	0x0000_00F0
45	52	可设置	TIM8_TRG_COM	TIM8触发和通信中断	0x0000_00F4
46	53	可设置	TIM8_CC	TIM8捕获比较中断	0x0000_00F8
47	54	可设置	ADC3	ADC3全局中断	0x0000_00FC
48	55	可设置	FSMC	FSMC全局中断	0x0000_0100

49	56	可设置	SDIO	SDIO全局中断	0x0000_0104
50	57	可设置	TIM5	TIM5全局中断	0x0000_0108
51	58	可设置	SPI3	SPI3全局中断	0x0000_010C
52	59	可设置	UART4	UART4全局中断	0x0000_0110
53	60	可设置	UART5	UART5全局中断	0x0000_0114
54	61	可设置	TIM6	TIM6全局中断	0x0000_0118
55	62	可设置	TIM7	TIM7全局中断	0x0000_011C
56	63	可设置	DMA2通道1	DMA2通道1全局中断	0x0000_0120
57	64	可设置	DMA2通道2	DMA2通道2全局中断	0x0000_0124
58	65	可设置	DMA2通道3	DMA2通道3全局中断	0x0000_0128
59	66	可设置	DMA2通道4_5	DMA2通道4和DMA2通道5全局中断	0x0000_012C

6.3 USART常用库函数

函数 USART_Init

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： x 可以是 1， 2 或者 3，来选择 USART 外设
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

函数 USART_Init

USART_InitTypeDef 定义于文件“*stm32f10x_usart.h*”：

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

USART_WordLength	描述
USART_WordLength_8b	8 位数据
USART_WordLength_9b	9 位数据

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

USART_Parity	描述
USART_Parity_No	奇偶失能
USART_Parity_Even	偶模式
USART_Parity_Odd	奇模式

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

USART_Mode	描述
USART_Mode_Tx	发送使能
USART_Mode_Rx	接收使能

函数 USART_Init 使用方法

```
USART_InitTypeDef USART_InitStructure;

/* 配置USART波特率、数据位、停止位、奇偶校验、硬件流控制和模式 */
USART_InitStructure.USART_BaudRate = 9600;  //波特率9600bps
USART_InitStructure.USART_WordLength = USART_WordLength_8b; //8数据位
USART_InitStructure.USART_StopBits = USART_StopBits_1;  //1停止位
USART_InitStructure.USART_Parity = USART_Parity_No;   //无奇偶校验
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
//无硬件流控制
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;   //接收和发送模式

USART_Init (USART1, &USART_InitStructure);
```

函数 USART_Cmd

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 或者 3，来选择 USART 外设
输入参数 2	NewState: 外设 USARTx 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

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

函数 USART_SendData

函数名	USART_ SendData
函数原形	void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
功能描述	通过外设 USARTx 发送单个数据
输入参数 1	USARTx： x 可以是 1， 2 或者 3，来选择 USART 外设
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

```
/* Send one HalfWord on USART3 */
USART_SendData(USART3, 0x26);
```

函数 USART_ReceiveData

函数名	USART_ ReceiveData
函数原形	u8 USART_ReceiveData(USART_TypeDef* USARTx)
功能描述	返回 USARTx 最近接收到的数据
输入参数	USARTx： x 可以是 1， 2 或者 3，来选择 USART 外设
输出参数	无
返回值	接收到的字
先决条件	无
被调用函数	无

```
/* Receive one halfword on USART2 */
u16 RxData;
RxData = USART_ReceiveData(USART2);
```

函数 USART_GetFlagStatus

函数名	USART_ GetFlagStatus
函数原形	FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t 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： x 可以是 1， 2 或者 3，来选择 USART 外设
输入参数 2	USART_FLAG：待检查的 USART 标志位 参阅 Section： USART_FLAG 查阅更多该参数允许取值范围
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

USART_FLAG：

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

```
FlagStatus Status;
Status = USART_GetFlagStatus(USART1, USART_FLAG_TXE);
```

函数 USART_ClearFlag

函数名	USART_ ClearFlag
函数原形	void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG)
功能描述	清除 USARTx 的待处理标志位
输入参数 1	USARTx： x 可以是 1， 2 或者 3，来选择 USART 外设
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

```
/* 清除 USART1 溢出错误标志位 */
USART_ClearFlag ( USART1, USART_FLAG_ORE );
```

函数 USART_GetITStatus

函数名	USART_ GetITStatus
函数原形	ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT)
功能描述	检查指定的 USART 中断发生与否
输入参数 1	USARTx： x 可以是 1， 2 或者 3，来选择 USART 外设
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

```
/* Get the USART1 Overrun Error interrupt status */
ITStatus ErrorITStatus;
ErrorITStatus = USART_GetITStatus(USART1, USART_IT_OverrunError);
```

USART_IT：

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	噪音错误中断
USART_IT_FE	帧错误中断

函数 USART_ITConfig

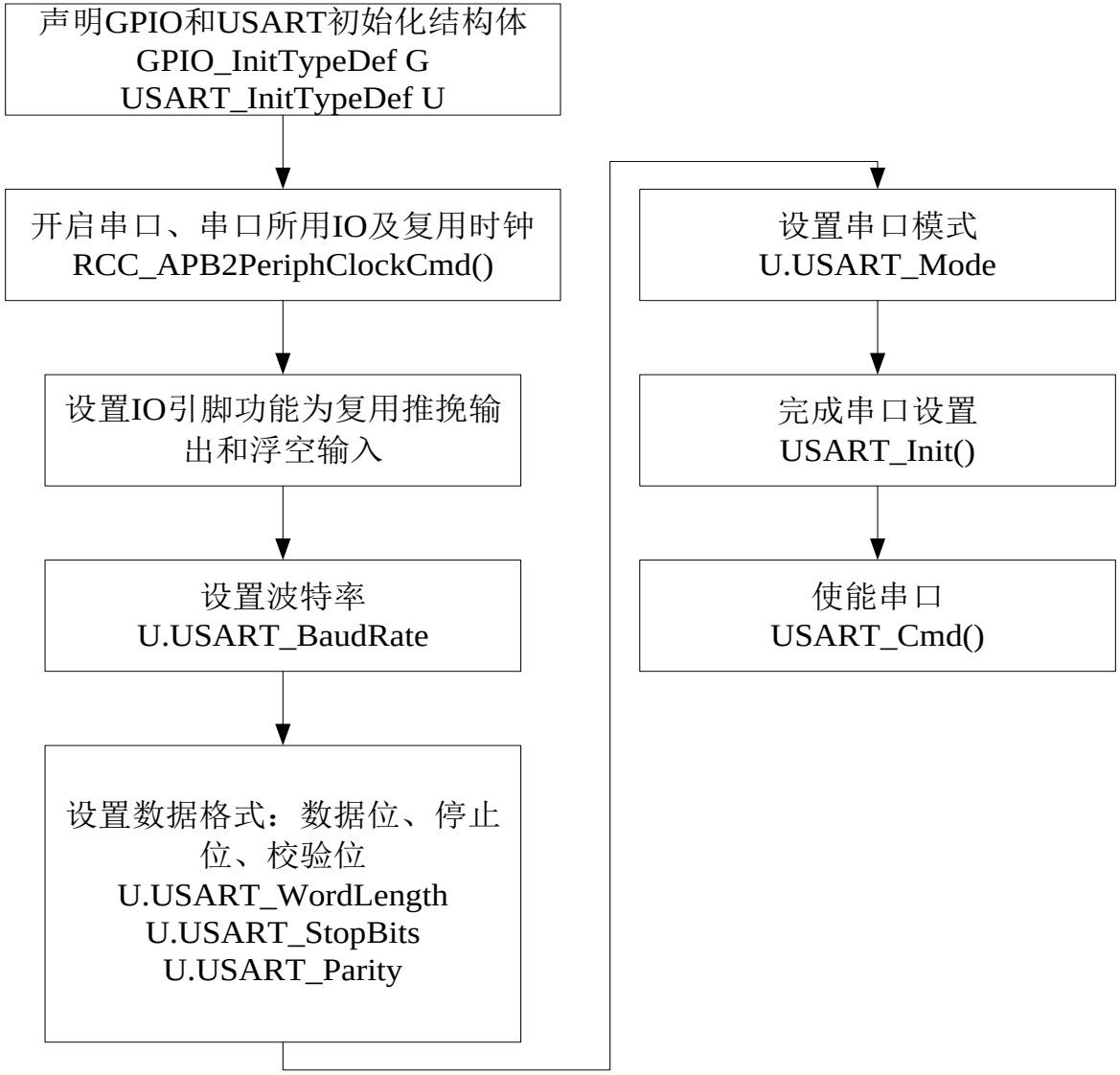
函数名	USART_ITConfig
函数原形	void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 USART 中断
输入参数 1	USARTx： x 可以是 1， 2 或者 3，来选择 USART 外设
输入参数 2	USART_IT：待使能或者失能的 USART 中断源 参阅 Section： USART_IT 查阅更多该参数允许取值范围
输入参数 3	NewState： USARTx 中断的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_IT：

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

```
/* 使能 USART1 接收中断 */
USART_ITConfig ( USART1, USART_IT_RXNE, ENABLE );
```

6.4 USART 使用流程



USART 中断方式流程

① 串口时钟使能，GPIO时钟使能: *RCC_APB2PeriphClockCmd();*

② 串口复位: *USART_DeInit(); 这一步不是必须的*

③ GPIO端口模式设置: *GPIO_Init(); 模式设置为 GPIO_Mode_AF_PP，GPIO_Mode_IN_FLOATING*

④ 串口参数初始化：*USART_Init();*

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）：

NVIC_Init();

USART_ITConfig();

⑥ 使能串口: *USART_Cmd();*

⑦ 编写中断处理函数：*USARTx_IRQHandler();*

⑧ 串口数据收发：

*void USART_SendData(); //*发送数据到串口，DR

*uint16_t USART_ReceiveData(); //*接受数据，从DR读取接受到的数据

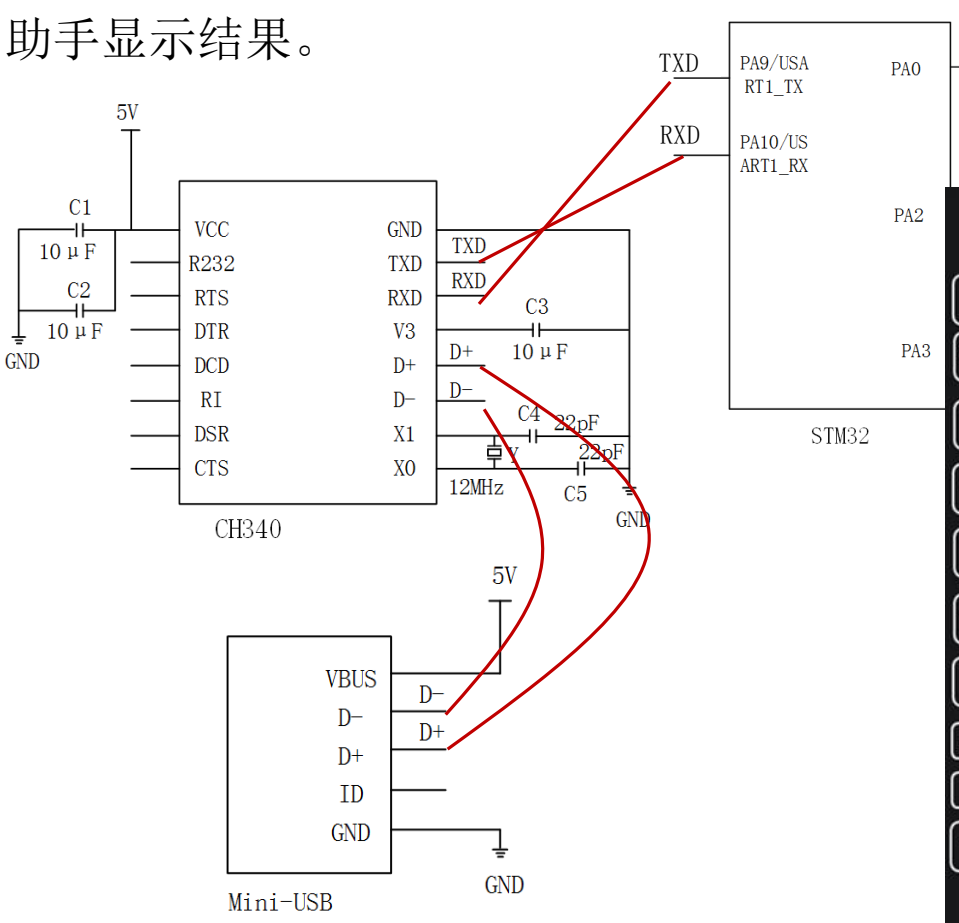
⑨ 串口传输状态获取：

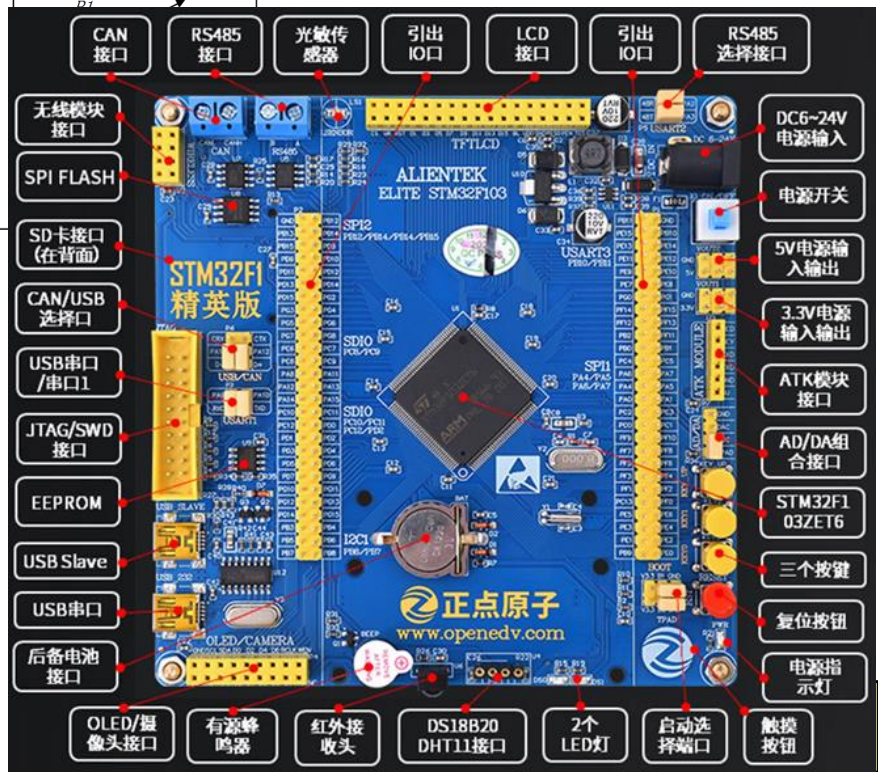
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);

void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);

6.5 USART应用设计实例

实例1：利用上位机串口与 STM32 的 USART1 通信。上位机通过键盘给 STM32 的串口发送字符，STM32 将接受到的字符再传回上位机。在上位机上通过串口调试助手显示结果。





方法 1

```
int  main (void)
{
   uint16_t dat;
   USART1_Configure ();   // USART1初始化
   while (1) {
       /* 判断接收寄存器是否非空 */
       if ( USART_GetFlagStatus (USART1, USART_FLAG_RXNE) == SET) {
           dat = USART_ReceiveData (USART1);    // 接收数据
           USART_SentData (USART1, dat);      //  发送数据
             /* 等待发送完成 */
           while (USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET)
           ;
        }
    }
}
```

RCC_AHB2Periph	描述
RCC_APB2Periph_AFIO	功能复用IO时钟
RCC_APB2Periph_GPIOA	GPIOA时钟
RCC_APB2Periph_GPIOB	GPIOB时钟
RCC_APB2Periph_GPIOC	GPIOC时钟
RCC_APB2Periph_GPIOD	GPIOD时钟
RCC_APB2Periph_GPIOE	GPIOE时钟
RCC_APB2Periph_ADC1	ADC1时钟
RCC_APB2Periph_ADC2	ADC2时钟
RCC_APB2Periph_TIM1	TIM1时钟
RCC_APB2Periph_SPI1	SPI1时钟
RCC_APB2Periph_USART1	USART1时钟
RCC_APB2Periph_ALL	全部APB2外设时钟

```
void USART1_Configure(void)
{
        GPIO_InitTypeDef  GPIO_InitStructure;
        USART_InitTypeDef  USART_InitStructure;
          /* 打开串口、串口所用I/O口以及端口复用时钟 */
          RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
                                 RCC_APB2Periph_AFIO, ENABLE);

          /* 配置PA9(USART_Tx)为复用推挽输出，IO速度50MHz */
          GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
          GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
          GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
          /* 完成配置 */
          GPIO_Init(GPIOA, &GPIO_InitStructure);

          /* 配置PA10(USART1_Rx)为浮空输入 */
          GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
          GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
          /* 完成配置 */
          GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
    /* 配置USART波特率、数据位、停止位、奇偶校验、硬件流控制和模式 */
    USART_InitStructure.USART_BaudRate = 9600;  //波特率9600 bps或 115200bps
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;  //8数据位
    USART_InitStructure.USART_StopBits = USART_StopBits_1;   //1停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;   //无奇偶校验
    USART_InitStructure.USART_HardwareFlowControl =
                                        USART_HardwareFlowControl_None;//无硬件流控制
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;//收、发模式

    /* 完成配置 */
    USART_Init (USART1, &USART_InitStructure);

    /* 使能USART1 */
    USART_Cmd(USART1, ENABLE);
}
```

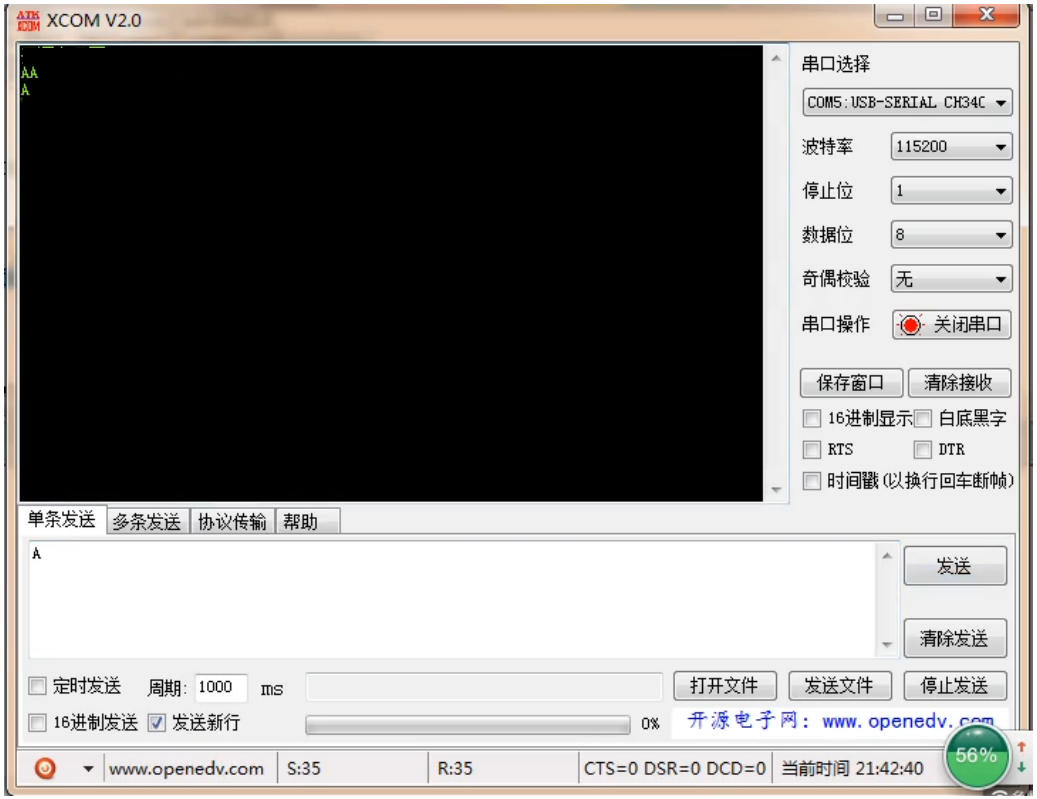
方法 2

```
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    My_USART1_Init();
    while(1)
        ;
}
```

```
void My_USART1_Init(void)
{
        GPIO_InitTypeDef GPIO_InitStrue;
        USART_InitTypeDef USART_InitStrue;
        NVIC_InitTypeDef NVIC_InitStrue;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
                                RCC_APB2Periph_AFIO, ENABLE);

        GPIO_InitStrue.GPIO_Pin=GPIO_Pin_9;
        GPIO_InitStrue.GPIO_Mode=GPIO_Mode_AF_PP;
        GPIO_InitStrue.GPIO_Speed=GPIO_Speed_10MHz;
        GPIO_Init(GPIOA,&GPIO_InitStrue);     // PA9，TX

        GPIO_InitStrue.GPIO_Pin=GPIO_Pin_10;
        GPIO_InitStrue.GPIO_Mode=GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOA,&GPIO_InitStrue);      // PA10，RX
```

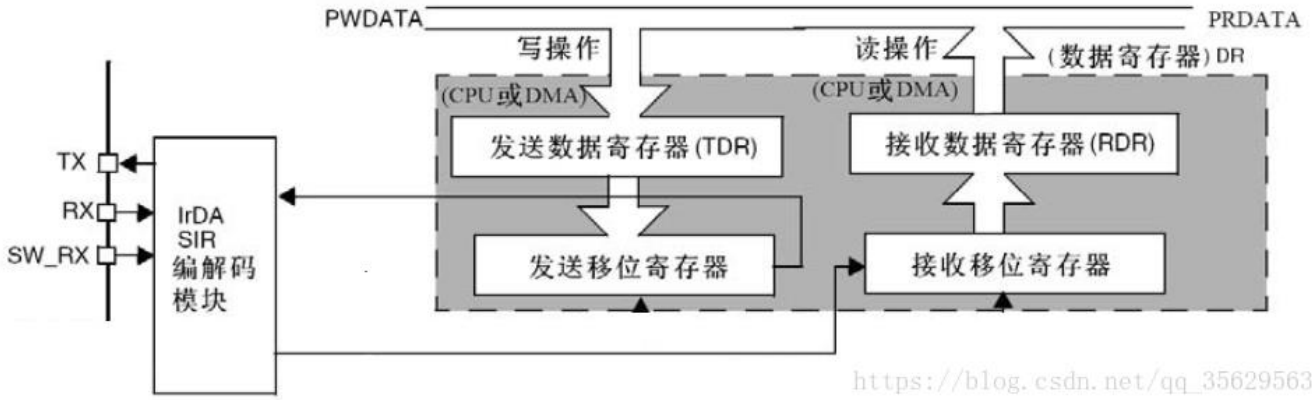
```
USART_InitStrue.USART_BaudRate=115200;
USART_InitStrue.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
USART_InitStrue.USART_Mode=USART_Mode_Tx|USART_Mode_Rx;
USART_InitStrue.USART_Parity=USART_Parity_No;
USART_InitStrue.USART_StopBits=USART_StopBits_1;
USART_InitStrue.USART_WordLength=USART_WordLength_8b;

USART_Init(USART1,&USART_InitStrue);//③

USART_Cmd(USART1,ENABLE);     //使能串口1

NVIC_InitStrue.NVIC_IRQChannel=USART1_IRQn;
NVIC_InitStrue.NVIC_IRQChannelPreemptionPriority=1;
NVIC_InitStrue.NVIC_IRQChannelSubPriority=1;
NVIC_InitStrue.NVIC_IRQChannelCmd=ENABLE;
NVIC_Init(&NVIC_InitStrue);

USART_ITConfig(USART1,USART_IT_RXNE, ENABLE);    //开启接收中断
}
```

```
void USART1_IRQHandler(void)
{
        u8 res;
        if(USART_GetITStatus(USART1,USART_IT_RXNE) != RESET) {
                USART_ClearITPendingBit(USART1,USART_IT_RXNE);
                res= USART_ReceiveData(USART1);
                USART_SendData(USART1, res);
         }
}
```



实例 2：发送程序的实现

在USART的发送端有2个寄存器，一个是程序可以看到的寄存器——发送数据寄存器（通过USART_DR查看）；另一个是程序看不到的寄存器——发送移位寄存器。对应的有两个USART数据发送标志，一个是TXE=发送数据寄存器空，另一个是TC=发送移位寄存器空。

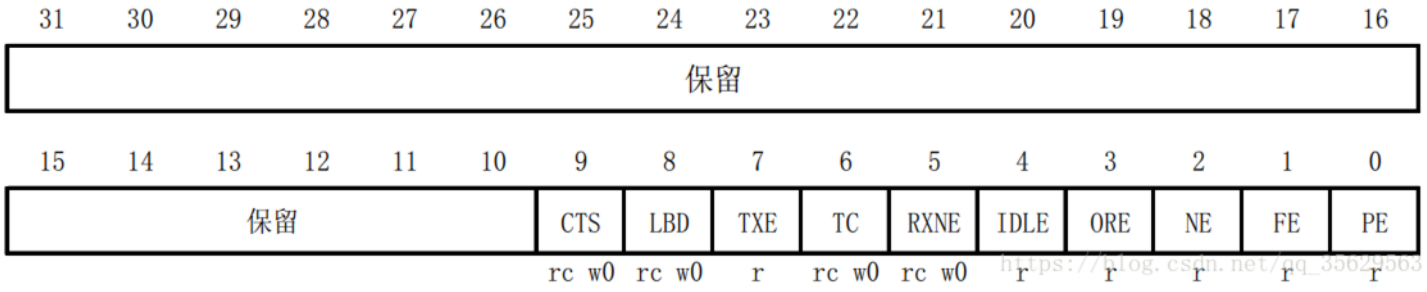


当TDR中的数据传送到移位寄存器后，TXE被设置，此时移位寄存器开始向TX信号线按位传输数据，但因为TDR已经变空，所以程序可以把下一个要发送的字节写入TDR中，而不必等到移位寄存器中所有位发送结束，**所有位发送结束时（送出停止位后）硬件会设置TC标志**。另一方面，**在刚刚初始化好USART还没有发送任何数据时，也会有TXE标志，因为这时发送数据寄存器是空的。**

USART_FLAG：

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

状态寄存器（USART_SR）



发送方法 1

```
/*******************************************************************************
* 函数名  : UART1_SendString
* 描述    : USART1发送字符串
* 输入    : *s字符串指针
* 注释    : 0==RESET,表示发送还未完成
USART_FLAG_TC!=RESET，就是=SET，表示发送完成，此时执行USART_GetFlagStatus会把USART_FLAG_TC清零
*******************************************************************************/
void UART1_SendString(u8* s)
{
  while(*s)//检测字符串结束符
  {
   //USART_FLAG_TC==RESET时，表示发送还未完成。
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC)==RESET);
    USART_SendData(USART1 ,*s++);//发送当前字符
  }
}
```

发送方法 2

```c
u8 FLAG_TC=0;//定义全局变量
/*******************************************************************************
* 函数名  : UART1_SendString
* 描述    : USART1发送字符串
* 输入    : *s字符串指针
*******************************************************************************/
void UART1_SendString(char* s)
{
  FLAG_TC=0;//提前准备一下
  while(*s)//检测字符串结束符
  {
    USART_SendData(USART1 ,*s++);//发送当前字符
    while( FLAG_TC==0); //0：发送还未完成；1：发送完成
    FLAG_TC=0;
  }
}


void USART1_IRQHandler(void)
{
  if (USART_GetITStatus(USART1, USART_IT_TC) != RESET)//发送完成中断,= SET
  {
    USART_ClearITPendingBit(USART1,USART_IT_TC);
    FLAG_TC=1;
  }
}
```

发送方法 3

```
//定义全局变量。也可以为了简化，把这四个参数结合起来包含在一个结构体里
u8 TxLength;   //发送数据长度
u8 TxIndicator;//发送指示器，表示目前发送完成哪一位了，下面要发送的是第（TxIndicator+1）位
u8 TxBuff[256];//Data
u8 TxFnd;      //发送完成标志

/*****************************************************************************
*程序名称     :   SendFirstByte
*功能         :   启动发送第一个字节
*@Notes      :            剩下的都放在USART_FLAG_TC中断里面，因为这个中断是
                 发送完成中断，是利用“发送第一个字节”来“启动发送”
*@Notes      :   需要注意，如果发送结果是乱码的话，一种供参考的解决方案是
                 把数据直接赋值给TxBuff，而不要通过函数的形参传递
*输入参数     :   u8 txbuf[]: 需要发送的数据,u8 len : 数据中的字节数
*返回值       :   无
*****************************************************************************/
void SendFirstByte( u8 txbuf[], u8 len )
{
    TxBuff      = txbuf;//需要发送的数据
    TxLength    = len  ;//发送数据长度
    TxIndicator = 1    ;//0已经发送，也是用来启动发送的

    USART_SendData(USART1, txbuf[0]);  /**@Notes：只发送了txbuf的第一个字节*/
}
```

发送方法 3

```
/*********************************************************************************
*程序名称     :   USART1_IRQHandler
*功能         :   完成发送数据
*********************************************************************************/
void USART1_IRQHandler(void)
{
    if (USART_GetITStatus(USART1, USART_IT_TC) != RESET)
    {
        USART_ClearITPendingBit(USART1,USART_IT_TC);
        if( TxIndicator < TxLength  )//数组的索引max永远小于数组元素的个数
        {
            USART_SendData(USART1, TxBuff[TxIndicator++]);
        }
        else//最后一字节数据发送完成
        {
            TxFnd = 0;
            TxIndicator = 0;
        }
    }
}
```

6.6 串行通信接口抗干扰设计

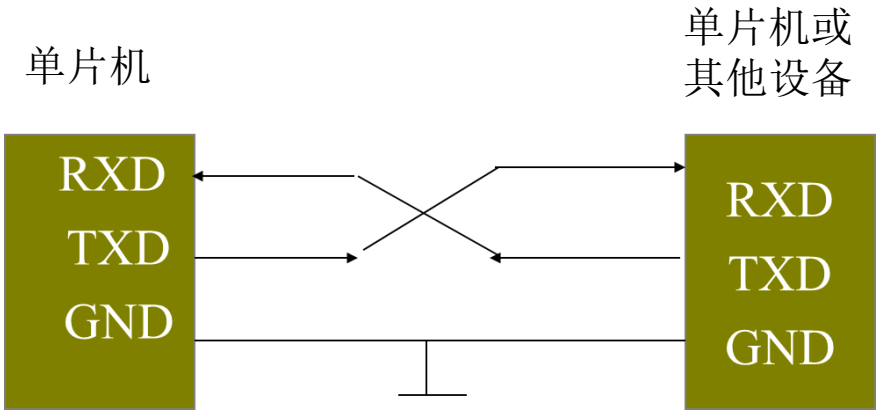
STM32串行口的输入、输出均为TTL电平。这种以TTL电平串行传输数据的方式抗干扰性差，传输距离短。为了提高串行通信的可靠性，增大串行通信的距离，一般都采用标准串行接口，如RS-232、RS-422A、RS-485等来实现串行通信。

标准	逻辑电平0	逻辑电平1	是否全双工	抗干扰能力
TTL	输出低电平<0.4V, 输入低电平<=0.8V	输出高电平>2.4V,输入高电平>=2.0V	全双工	差
RS232	+3～+15V	-3～-15V	全双工	强
RS485	+2V～+6V	- 6V～- 2V	半双工	很强

常见的串口通信电路

通信速率和通信距离这两个方面是相互制约的，降低通信速率，可以提高通信距离。不同的通信距离，串行通信电路有不同的连接方法。

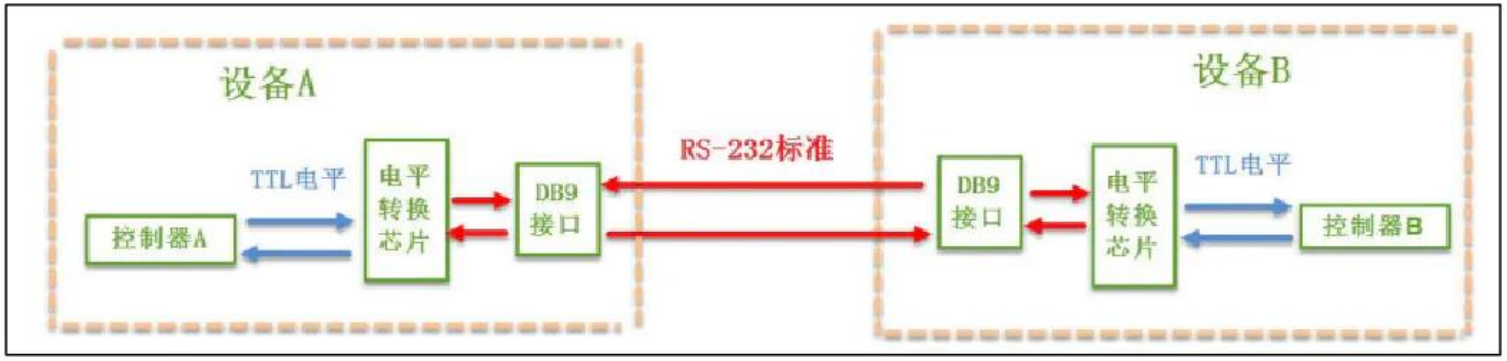
（1）TTL设备之间通讯

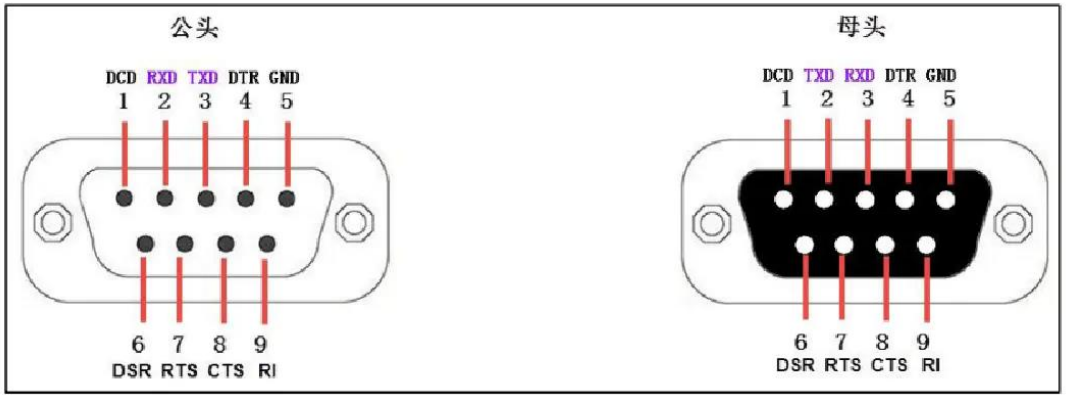


近距离传送电路

（2）两个设备之间采用RS232通讯

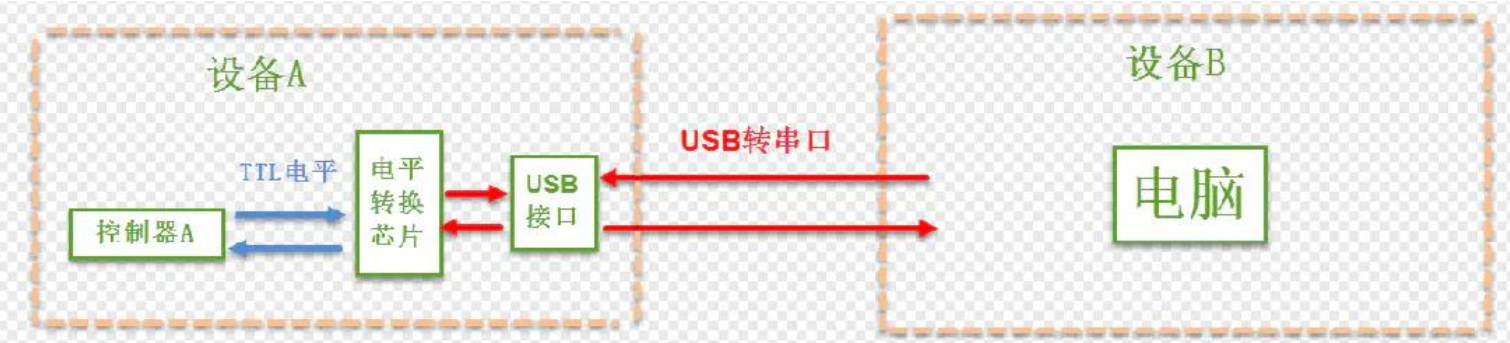
常用的TTL转RS232的电平转换芯片有MAX232、MAX3232。其中MAX232只能用5V供电。而MAX3232可以用5V或3.3V供电。



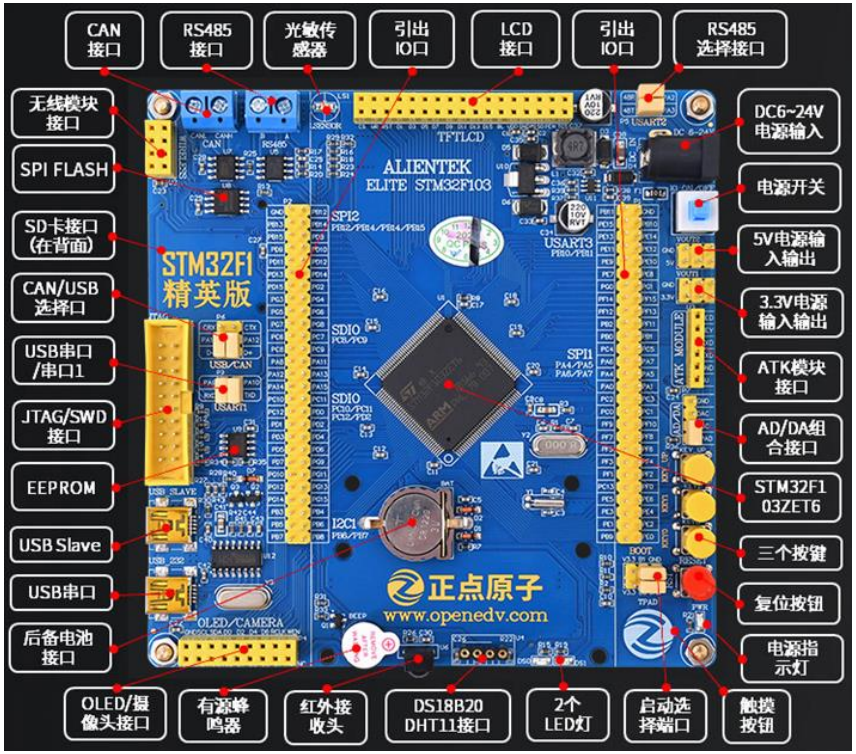


RS232接口图片

（3）TTL转USB，直接和电脑通讯



经常使用的芯片是FT232、PL2303、CH340。这三个都可以实现USB转串口TTL电平，稳定性方面：FT232>CH340>PL2303。 PL2303虽然不稳定，但是很便宜，因此用的也不少。FT232最稳定，但是价格最贵。



测控系统中，计算机通信主要采用异步串行通信方式，常用的异步总线标准：

- **RS-232（RS-232A，RS-232B，RS-232C）**

  RS-232C：速率：20Kbit/S，最大通信距离：15m

  特性：全双工，1对1单站通信

- **RS485： 速率： 10Mbit/s，最大通信距离：3000m**

  特性：半双工，1对128多站通信

✓ 和RS232相比，由于RS485采用了差分传输的方式，因此抗干扰能力强很多。TTL抗干扰能力最差。

✓ 像STM32这类单片机的USART口，出来的电平都是TTL电平，想要和电脑通讯需要电平转换芯片。

作业：

完成课本P94 6.5.2实例的编程，并在实验课上调试通过。