STM32的串口

内容提要

1 串行通信vs并行通信

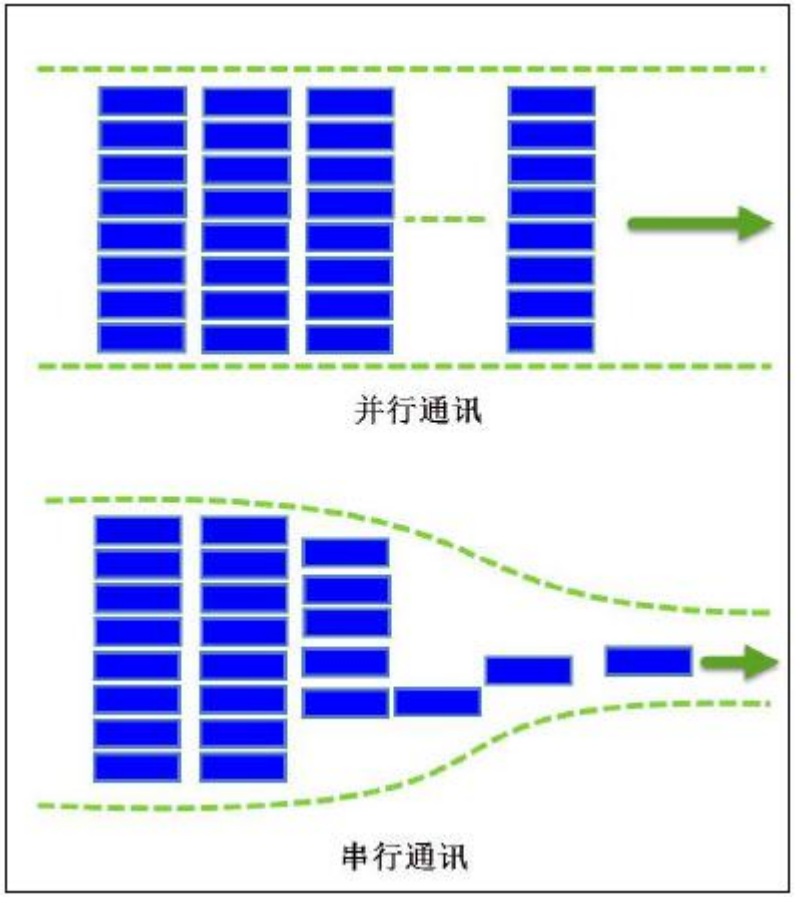
2 串行通信(串口通信）简介

3 STM32串口

4 USART操作相关数据结构和库函数

1 串行通信vs并行通信

串行通信与并行通信



- 并行：使用8根数据线一次传送一个字节（或使用16根数据线一次传送2个字节，…）
- 串行：使用少量数据信号线（8根以下），将数据逐位分时传送
- 并行vs串行：类似于多车道vs单车道

串行通信和并行通信特性对比

特性	串行通信	并行通信
通信距离	较远	较近
抗干扰能力	较强	较弱
传输速率	较低	较高
成本	较低	较高

- 在工作频率相同的情况下，并行通信的数据传输速率明显高于串行通信。（注意前提）
- 由于并行传输对同步要求较高，且随着通信速率的提高，信号干扰的问题会显著影响通信性能。随着技术的发展，越来越多的应用场合采用高速率的串行差分传输。

- 并行接口举例：

Centronics打印接口、PATA硬盘接口、AT键盘接口

- 串行接口举例：

Ps/2接口、usb接口、SATA硬盘接口、I2C接口、SPI接口

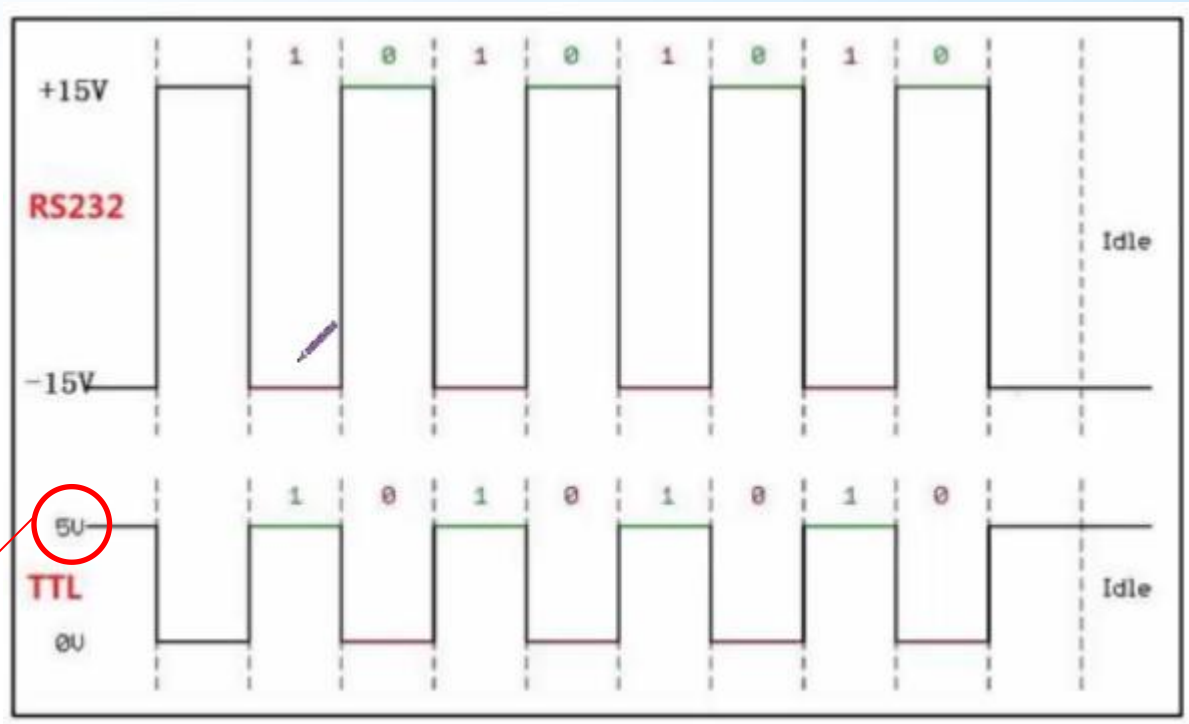
2串口通信简介

- 物理层：规定通讯系统中具有机械、电子功能部分的特性、确保原始数据在物理媒体的传输。其实就是硬件部分。
- 协议层：协议层主要规定通讯逻辑，统一收发双方的数据打包、解包标准。其实就是软件部分。

简单来说，物理层规定我们用嘴巴交流还是用肢体交流；协议层则规定我们用中文还是英文来交流。

串口通信--物理层常用标准

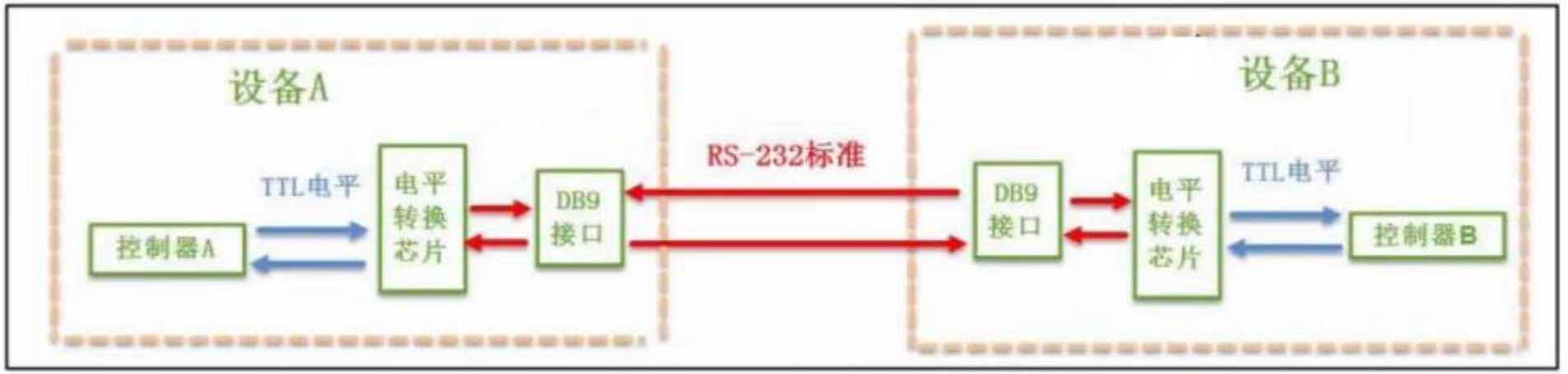
- RS232标准
- USB转串口
- 原生的串口到串口



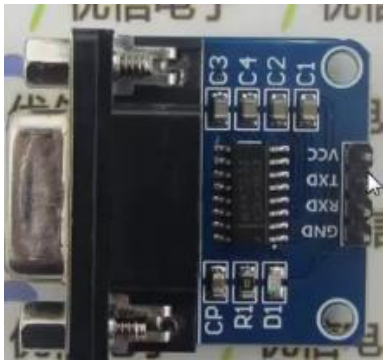
STM32中可以是3.3V

RS-232与TTL电平区别

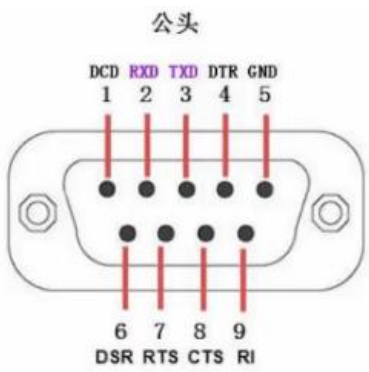
RS232标准串口通讯结构图



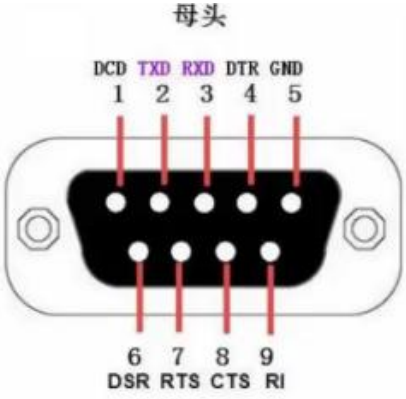
- RS232标准串口主要用于工业设备直接通信
- 电平转换芯片一般有MAX3232、SP3232



RS232转TTL
（串口模块）



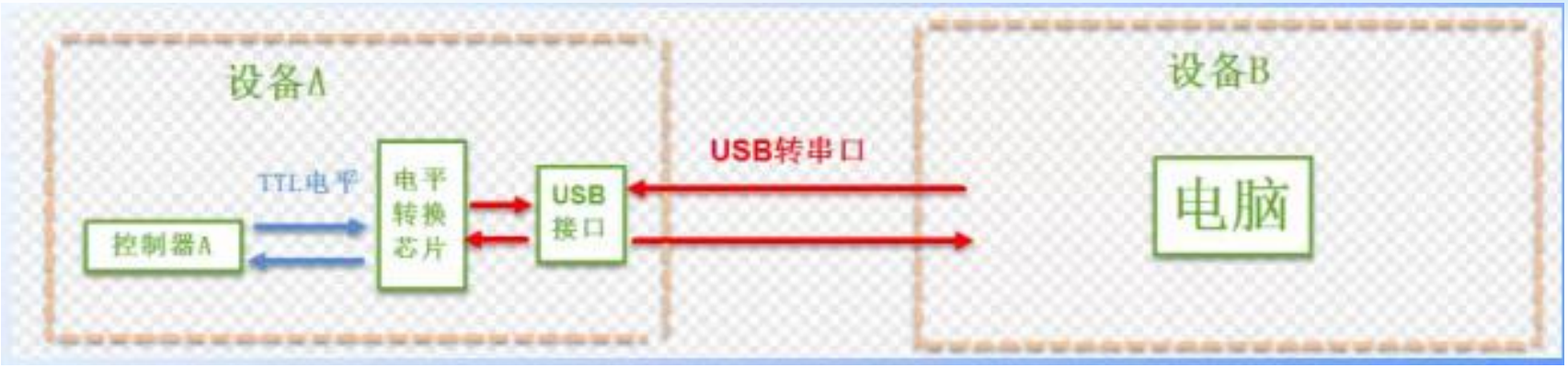
DB9公头



DB9母头

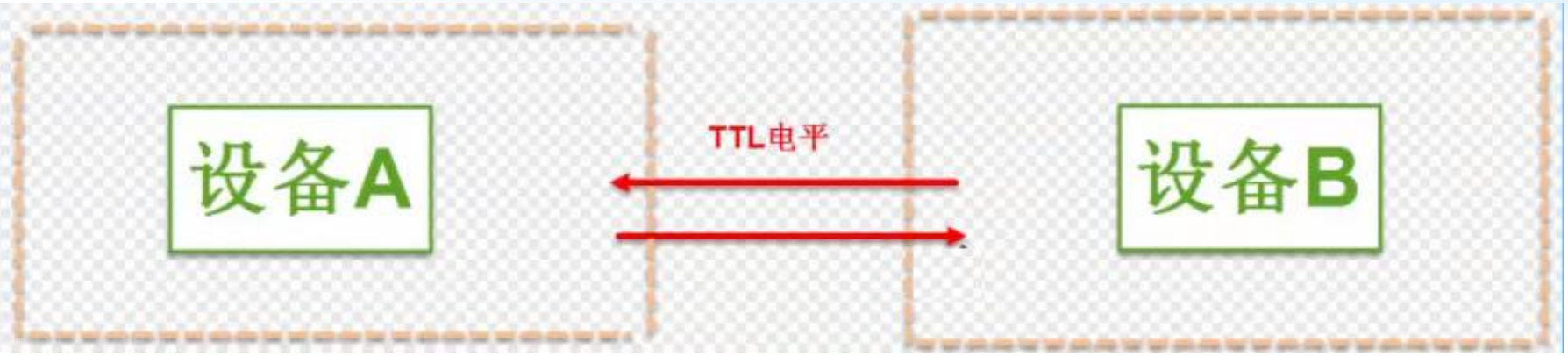
DB9串口线

USB转串口通讯结构图



- USB转串口主要用于设备与电脑通信
- 电平转换芯片一般有CH340、PL2303、CP2101、FT232
- 使用的时候电脑端需要安装电平转换芯片的驱动

原生的串口到串口

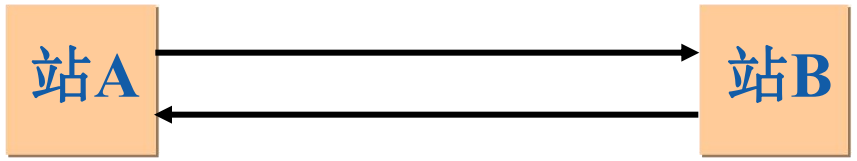


- 原生的串口通信主要是控制器跟串口的设备或者传感器通信，不需要经过电平转换芯片来转换电平，直接就用TTL电平通信
- 如：GPS模块、GSM模块、串口转wifi模块、HC04蓝牙模块

传输制式

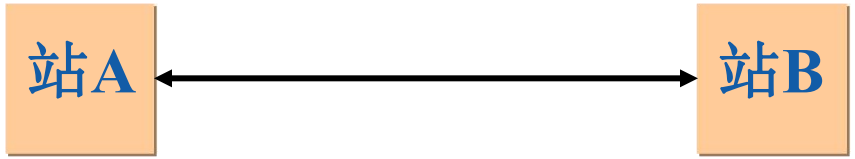
- 全双工通信

同时双向传输



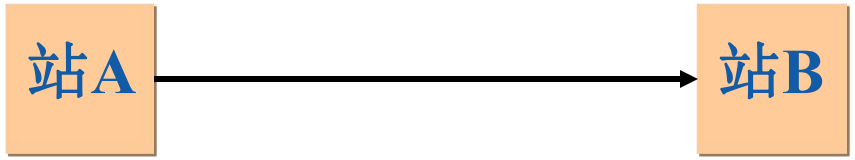
- 半双工通信

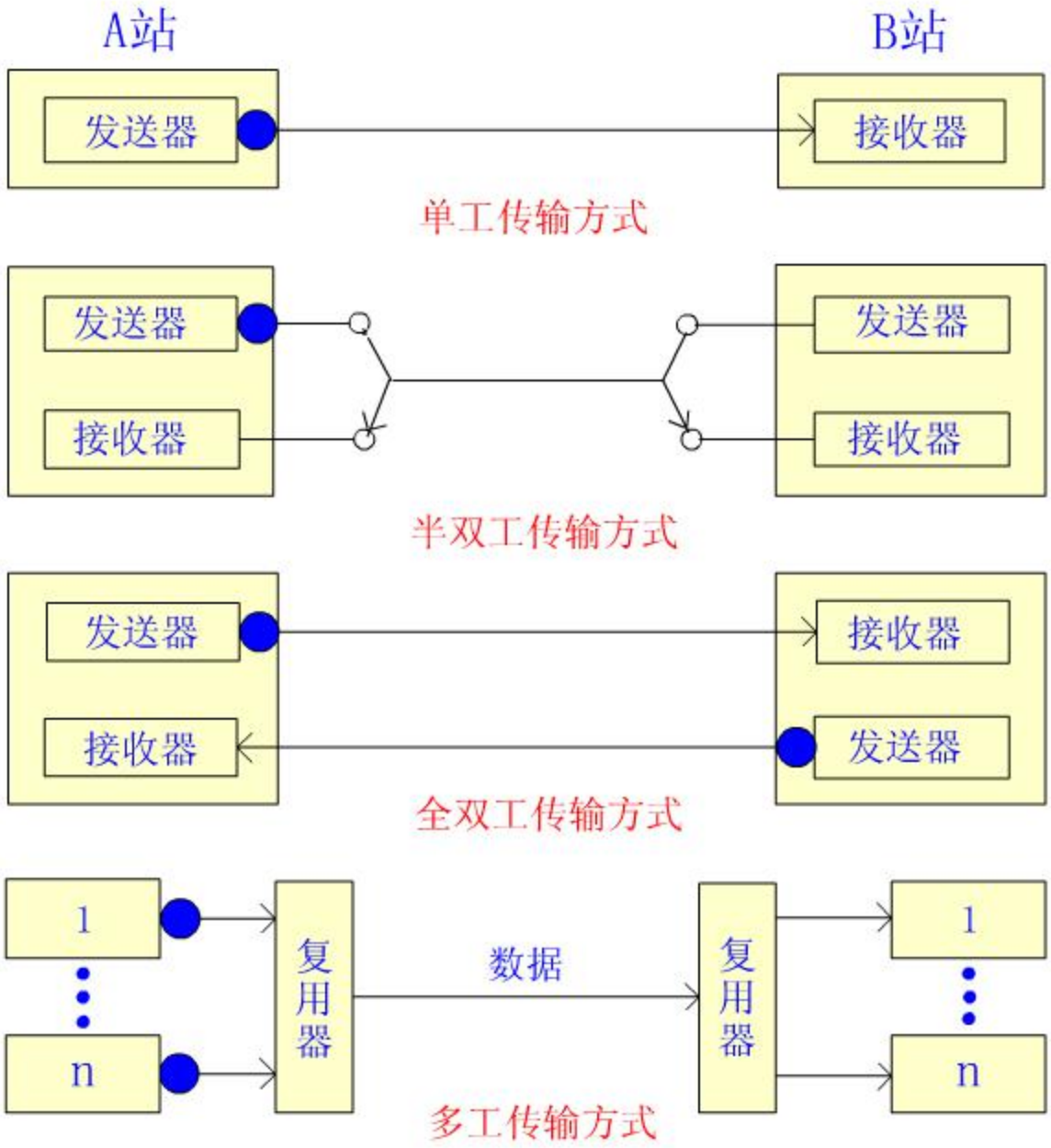
分时双向传输



- 单工通信

单向传输




A站
B站
发送器
接收器
单工传输方式
发送器
接收器
发送器
接收器
半双工传输方式
发送器
接收器
接收器
发送器
全双工传输方式
1
n
复用器
数据
复用器
1
n
多工传输方式

传输制式

同步与异步

数据同步方式

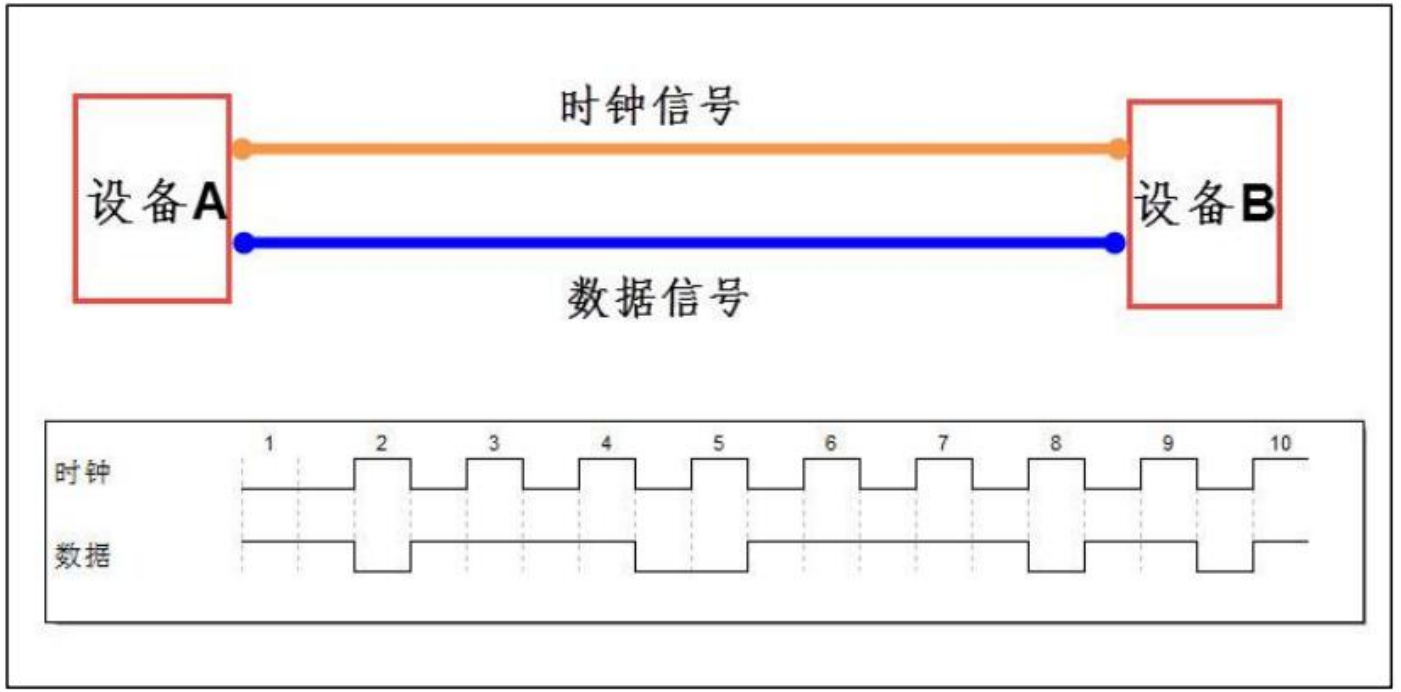


图 19-3 同步通讯

同步与异步

数据同步方式

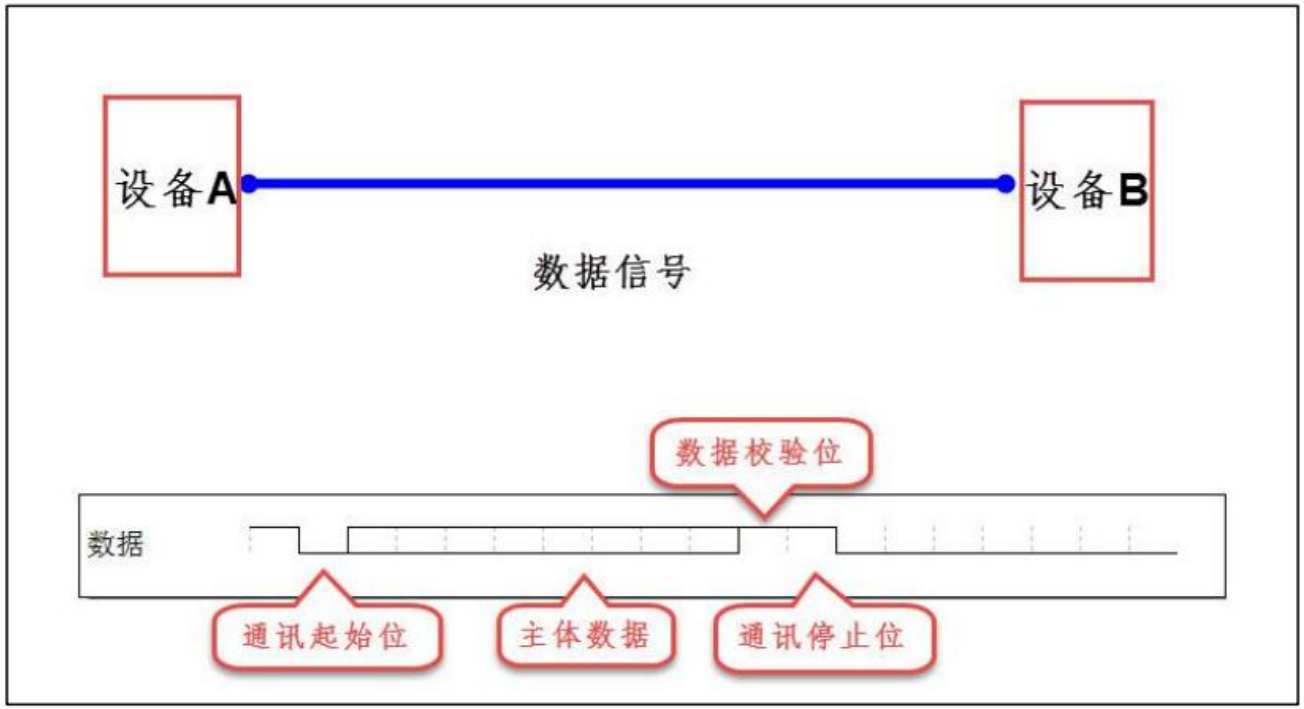


图 19-4 某种异步通讯

同步与异步

在同步通讯中，数据信号所传输的内容绝大部分就是有效数据，而异步通讯中会包含有帧的各种标识符，所以同步通讯的效率更高，但是同步通讯双方的时钟允许误差较小，而异步通讯双方的时钟允许误差较大。

- 串口数据包的基本组成

- 起始位：由1个逻辑0的数据位表示
- 结束位：由0.5、1、1.5或2个逻辑1的数据位表示

有效数据：在起始位后紧接着的就是有效数据，有效数据的长度常被约定为5、6、7或8位长

- 校验位：可选，为的是数据抗干扰，校验方法分为：

1-奇校验（odd），2-偶校验（even），3—0校验（space），4-1校验（mark），5-无校验（noparity）

- 奇校验（odd）：有效数据和校验位中“1”的个数为奇数

比如：一个8位长的有效数据位：01101001，此时总共有4个“1”，为达到奇校验效果，校验位为“1”，最后传输的数据将是8位的有效数据加上1位的校验位共9位。

串口通信协议简介

偶校验(even)： 有效数据和校验位中 “ 1” 的个数为偶数

比如一个 8 位长的有效数据为： 01101001，此时总共有 4 个 “ 1” ，为达到偶校验效果，校验位为 “ 0” ，最后传输的数据将是 8 位的有效数据加上 1 位的校验位总共 9 位

串口通信协议简介

0 校验是不管有效数据中的内容是什么，校验位总为“0”。

1 校验是校验位总为“1”。

无校验就是数据包中不包含校验位。

嵌入式系统中异步串行通信的意义

- 消耗硬件资源少，实现原理简单，为嵌入式系统提供了一种简单易行的连接开发板和上位PC机的方式
- 可以连接PC机串口，用于程序下载
- 借助pc端的串口调试工具可以实现开发板的调试、输出
- 开发板也可以通过串口接收上位机命令，并作出响应

3 STM32的串行通信接口USART

- STM32具有多达5个USART接口
  - USART1连接高速APB2总线、运行速度为72MHz（支持高达4.5Mbps的传输速率）
  - 其他位于APB1总线、36MHz（2.25Mbps传输速率）
- STM32的USART接口具有多种操作模式
  - 异步全双工通信、同步单路通信和半双工单线通信
  - 支持LIN、智能卡、IrDA、多处理器通信
- 每个USART接口具有两个DMA通道
  - 用于接收Rx和发送Tx数据与存储器之间的高速传送

Universal Synchronous Asynchronous Receiver/Transmitter

USART的增强功能

- 局部互联网络LIN
  （Local Interconnection network）
  - 主要针对车辆中低成本的LIN总线
- 智能卡（Smart Card）
  - 内嵌芯片的集成电路（IC）卡
  - 兼容ISO 7816-3标准的异步智能卡协议
- 红外线接口IrDA
  （Infrared Data Association）
  - 短距离、点对点直线数据传输
  - 支持SIR ENDEC传输编码解码协议

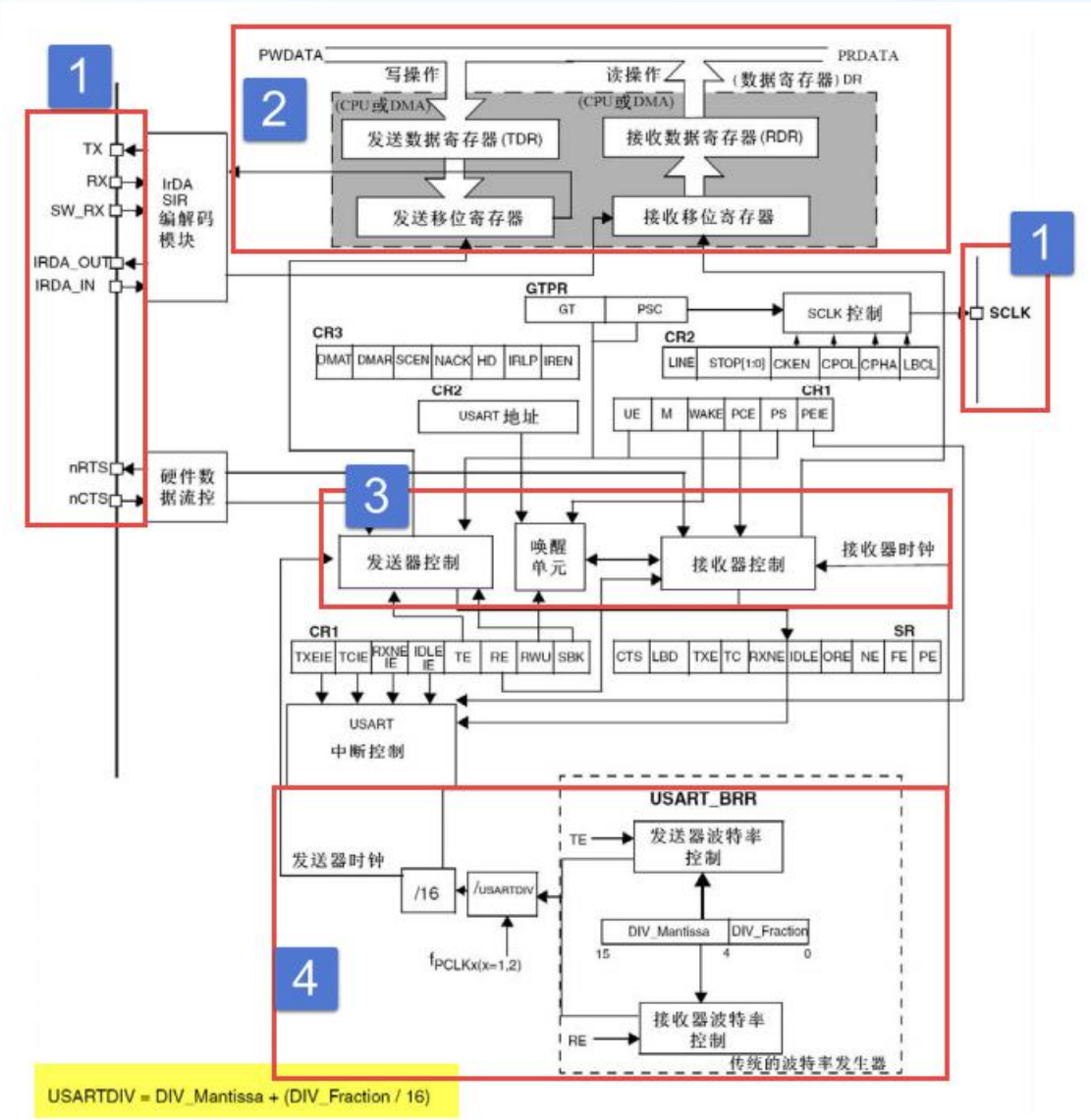
串口功能框图讲解

1-引脚

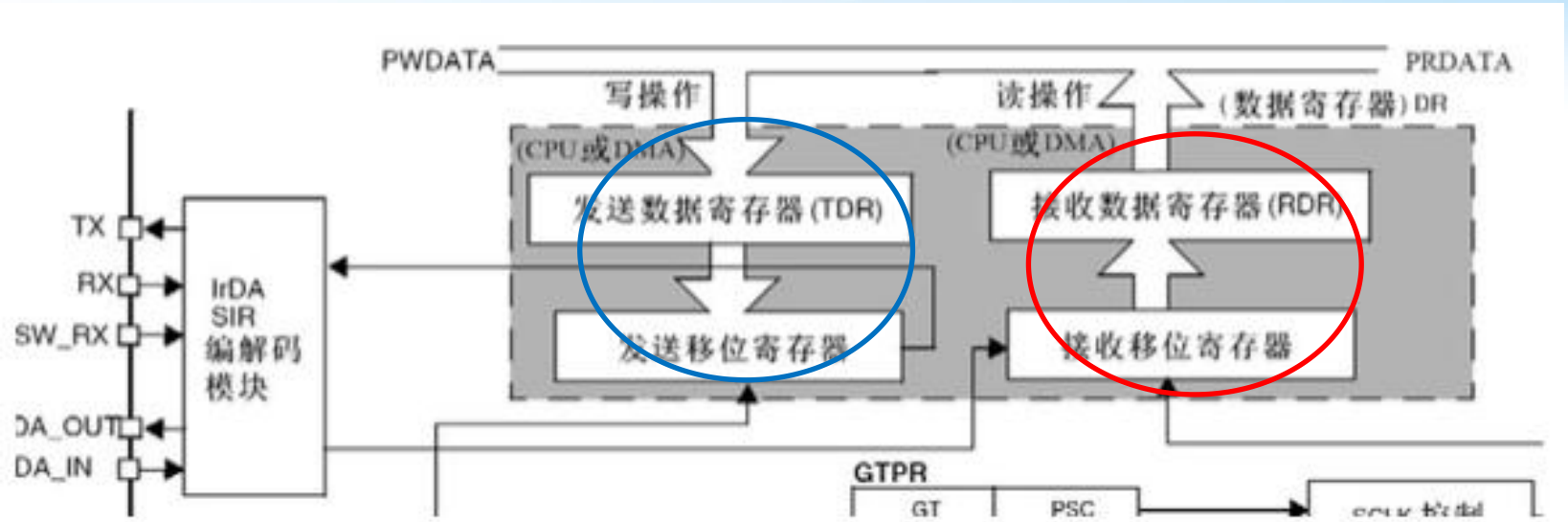
2-数据寄存器

3-控制器

4-波特率



接收发送数据过程



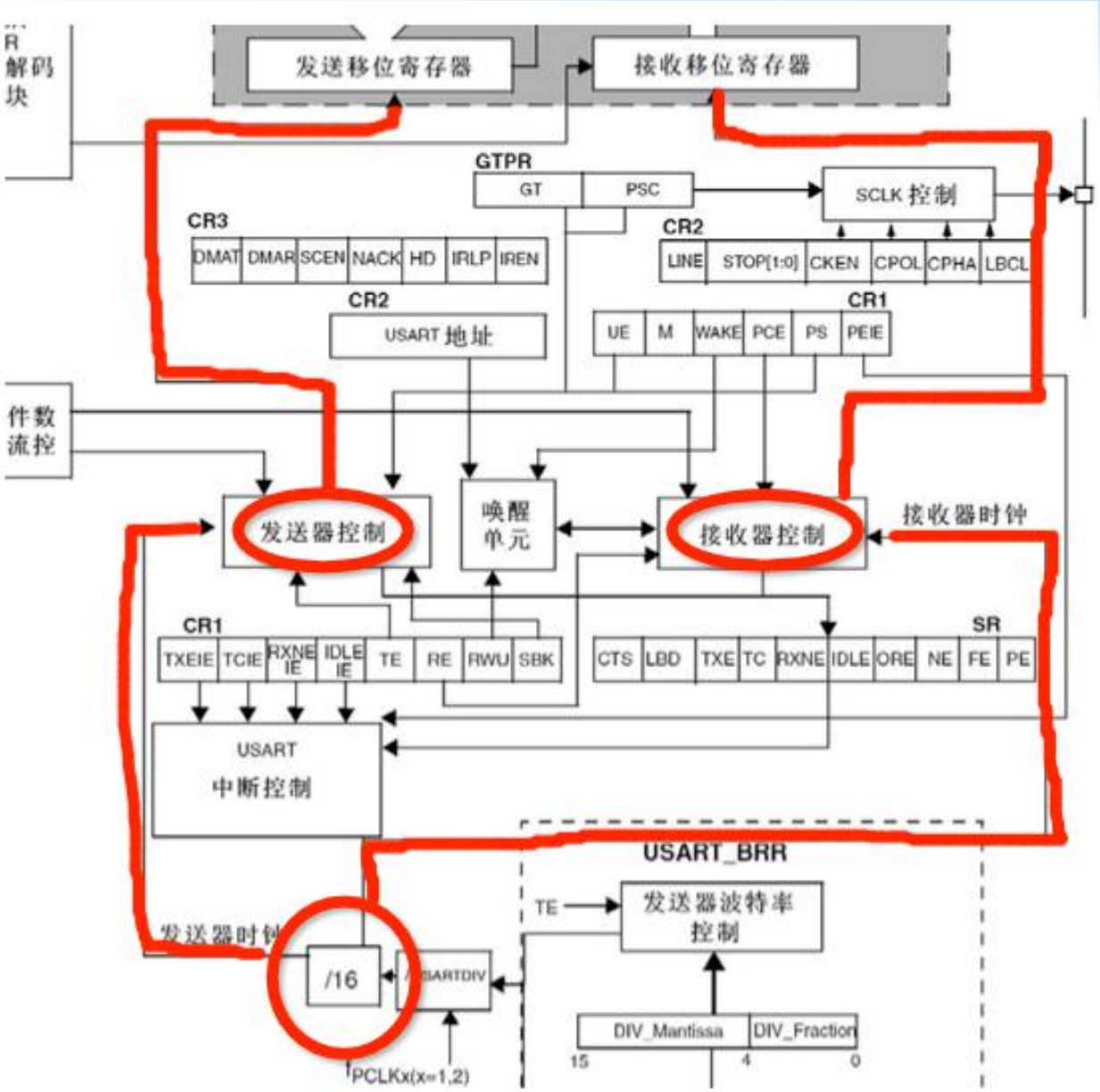
接收数据过程:

由Rx接收数据,根据波特率按位写入”接收移位寄存器”,待接收完毕后一次性写入”接收数据寄存器(RDR)”,CPU读取寄存器获取传输的数据

发送数据过程:

由CPU写数据到”发送数据寄存器(TDR)”,再由TDR一次性将要发送的数据写入”发送移位寄存器”,按照波特率逐位移出

波特率控制



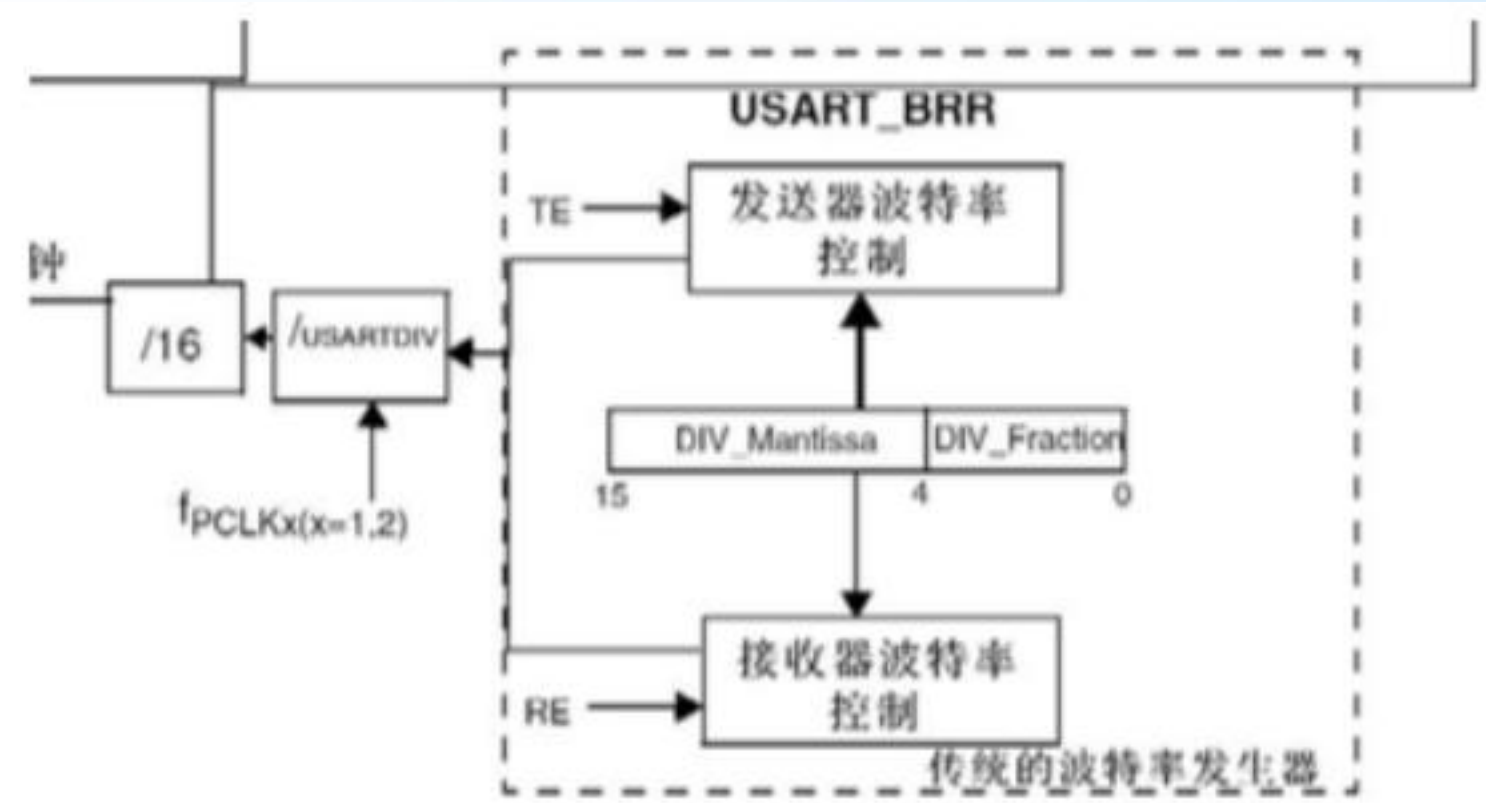
如图:

发送控制器和接收控制器分别控制发送移位寄存器和接收移位寄存器

发送器时钟和接收器时来自同一个时钟单元

发送和接收控制器的波特率公用波特率发生器

接收器时钟产生



STM32F103共5个串口:

串口1时钟来自PLCK2

串口2-5时钟来自PLCK1

例如：串口1的时钟由PCLK2经过/USARTDIV(分频)得到

而分频由波特率发生器控制，通过相关寄存器进行配置

USART接口的引脚

- USART接口通过3个引脚连接外设
  - Tx　发送数据输出（Transmit Data Out）
  - Rx　接收数据输入（Receive Data In）
  - CK 发送时钟输出，用于同步传输模式
    （CK引脚早期版本被称为SCLK）
- 实现硬件流程控制需要如下引脚
  - nCTS 清除发送（Clear To Send）
  - nRTS 发送请求（Request To Send）

硬件流控具体表现为：当串口已经准备好接收新数据时，由硬件自动把RTS脚拉低（向外表示可接收数据）；在发送数据前，由硬件自动检查CTS脚是否为低（表示是否可以发送数据），再进行发送。

表 21-3 STM32F103ZET6 芯片的 USART 引脚

引脚	APB2 总线	APB1 总线			
	USART1	USART2	USART3	UART4	UART5
TX	PA9	PA2	PB10	PC10	PC12
RX	PA10	PA3	PB11	PC11	PD2
SCLK	PA8	PA4	PB12		
nCTS	PA11	PA0	PB13		
nRTS	PA12	PA1	PB14		

STM32F10x数据手册—Pinouts and pin description。

ST每个系列的芯片都有一个数据手册，里面有引脚的详细功能。

USART寄存器

寄存器缩写	寄存器中文名称
USART_SR	状态寄存器
USART_DR	数据寄存器
USART_BRR	波特率寄存器
USART_CR1	控制寄存器1
USART_CR2	控制寄存器2
USART_CR3	控制寄存器3
USART_GTPR	时间保护和预分频寄存器

USART_SR—状态寄存器

25.6.1 状态寄存器(USART_SR)

地址偏移：0x00

复位值：0x00C0

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						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

状态寄存器USART_SR，描述串口寄存器的一些状态:

如位5:读数据寄存器非空

位5	**RXNE**：读数据寄存器非空 (Read data register not empty) 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，则产生中断。对USART_DR的读操作可以将该位清零。RXNE位也可以通过写入0来清除，只有在多缓存通讯中才推荐这种清除程序。 0：数据没有收到； 1：收到数据，可以读出。

通过读取这个位的值,判断是否收到了完整的数据
串口已经接收到了数据,并且已经写入到了USART_DR寄存器

USART_SR几个比较常用的bit

位7	TXE:发送数据寄存器空 (Transmit data register empty) 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 (Transmission complete) 当包含有数据的一帧发送完成后，并且TXE=1时，由硬件将该位置'1'。如果USART_CR1中的TCIE为'1'，则产生中断。由软件序列清除该位(先读USART_SR，然后写入USART_DR)。TC位也可以通过写入'0'来清除，只有在多缓存通讯中才推荐这种清除程序。 0：发送还未完成； 1：发送完成。
位5	RXNE：读数据寄存器非空 (Read data register not empty) 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，则产生中断。对USART_DR的读操作可以将该位清零。RXNE位也可以通过写入0来清除，只有在多缓存通讯中才推荐这种清除程序。 0：数据没有收到； 1：收到数据，可以读出。

USART_DR—数据寄存器

数据寄存器USART_DR，只使用了位0-8，其他位保留

位8:0	**DR[8:0]**：数据值 (Data value) 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248)。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中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。

读寄存器：读取该寄存器获取接收到的数据值
写寄存器：向该寄存器写入发送的数据对数据进行发送

USART_BRR—波特率寄存器

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

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**
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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 这12位定义了USART分频器除法因子(USARTDIV)的整数部分。
位3:0	**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。

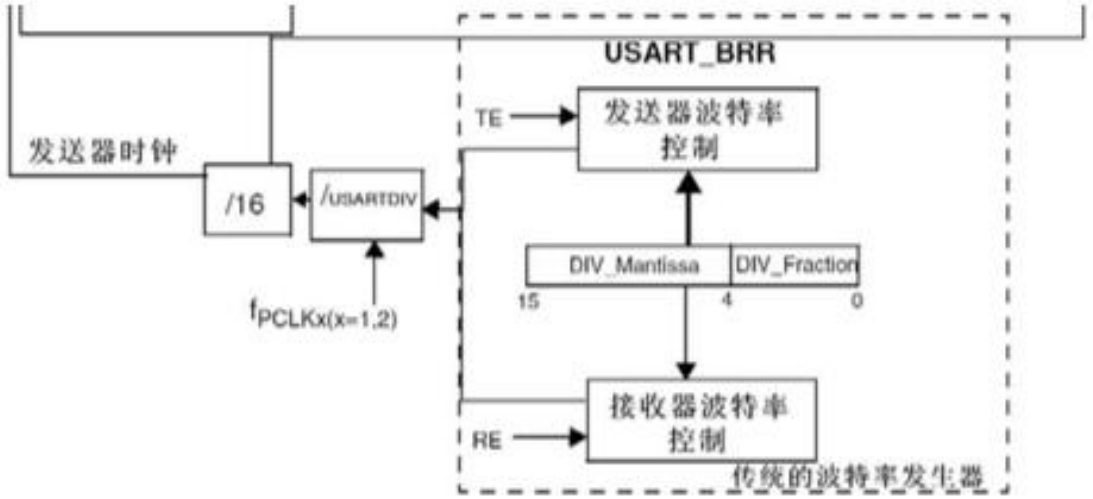
波特率寄存器USART_BRR，只用到了低16位，高16位保留

0-3位[3:0]　：　USART分频器的小数部分DIV_Fraction

4-15位[15:4]：　USART分频器的整数部分DIV_Mantissa

波特率的计算方法

波特率发生器：



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

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。

如图： 波特率由波特率发生器和PCLKx共同产生 PCLKx的值由串口本身决定 通过配置USART_BRR寄存器确定波特率发生器的值 经过USARTDIV分频器除以16得到最终的波特率

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

USARTDIV：无符号的定点数

FCK：串口的时钟，注意区分APB2和APB1两条总线

波特率计算举例

设置串口1波特率为115200

串口1的时钟来自PCLK2=72MHz
由公式得到:
USARTDIV=72000000/(115200*16)=39.0625
整数部分DIV_Mantissa=39=0x27
小数部分DIV_Fraction=16*0.0625=1=0x01

所以设置USART->BRR=0x0271,就可以实现设置串口1的波特率为115200

USART_CR1—控制寄存器

25.6.4 控制寄存器 1(USART_CR1)

地址偏移：0x0C

复位值：0x0000

USART_BRR波特率寄存器，设置串口寄存器使能位

如：接收使能，发送使能

位3	**TE**：发送使能 (Transmitter enable) 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. 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 (Receiver enable) 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。

4 USART操作相关数据结构和库函数

USART初始化结构体

```
typedef struct
{
  uint32_t USART_BaudRate;            //波特率 BRR
  uint16_t USART_WordLength;          //字长 CR1_M
  uint16_t USART_StopBits;            //停止位 CR2_STOP
  uint16_t USART_Parity;              //校验控制 CR1_PCE、CR1_PS
  uint16_t USART_Mode;                //模式选择CR1_TE、CR1_RE
  // 硬件流选择 CR3_CTSE、CR3_RTSE
  uint16_t USART_HardwareFlowControl;
} USART_InitTypeDef;
```

USART编程常用固件库函数

1-串口初始化函数

void USART_Init

(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)

USART初始化

- USART初始化函数

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

- USARTx（要配置的串口）：USART1～UART5
- USART_InitStruct指向USART_InitTypeDef结构变量的指针

```
typedef struct
{ uint32_t USART_BaudRate;    /* 通信波特率 */
  uint16_t USART_WordLength; /* 数据位数 */
  uint16_t USART_StopBits;    /* 停止位数 */
  uint16_t USART_Parity;      /* 校验模式 */
  uint16_t USART_Mode;        /* 接收发送模式 */
  uint16_t USART_HardwareFlowControl;/* 硬件流控制 */
} USART_InitTypeDef;
```

帮助文档

2-中断配置函数

void USART_ITConfig

(USART_TypeDef* USARTx, uint16_t USART_IT,

FunctionalState NewState)

3-串口使能函数

void USART_Cmd(USART_TypeDef* USARTx,

FunctionalState NewState)

USART编程常用固件库函数

4-数据发送函数

void USART_SendData

(USART_TypeDef* USARTx, uint16_t Data)

5-数据接收函数

uint16_t USART_ReceiveData(USART_TypeDef* USARTx)

6-中断状态位获取函数

ITStatus USART_GetITStatus

(USART_TypeDef* USARTx, uint16_t USART_IT)

USART库函数列表

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 最近接收到的数据
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

串口配置的典型步骤

1,串口时钟使能,GPIO时钟使能

RCC_APB2PeriphClockCmd()

2,串口复位

USART_DeInit();

3,GPIO端口模式设置

GPIO_Init();

4,串口参数初始化

USART_Init()

5,开启中断并初始化NVIC

NVIC_Init();

USART_ITConfig();

6,使能串口

USART_Cmd();

7,中断函数逻辑

USARTx_IRQHandler();

8,串口数据发送

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

9,串口传输状态获取

```
ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT);
void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);
```

串口测试程序1

- 串口硬件连接
  - PA9-Tx
  - PA10-Rx
  - 电脑通过USB线连接开发板（将USB虚拟为串口使用）
- 程序功能：
  - STM32向pc发送数据，pc通过串口调试助手查看

1、使能GPIOA和串口1时钟

```
//使能GPIOA时钟源
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
//使能串口1时钟源
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```

2、初始化GPIOA的工作模式

- 通过查找STM32中文参考手册确定串口 1 引脚工作模式配置：

USART引脚	通信配置	GPIO配置
USARTx_TX	全双工	复用推挽输出
	半双工同步模式	复用推挽输出
USARTx_RX	全双工	浮空输入 / 上拉输入
	半双工同步模式	未用。可用于通用I/O

串口 1 接收发送引脚配置 发送端PA9配置为推挽复用输出 接收端PA10配置为浮空输入或上拉输入

```
1  GPIO_InitTypeDef GPIO_InitStrue;
2
3  //发送端PA9配置
4  GPIO_InitStrue.GPIO_Pin=GPIO_Pin_9;              //发送端-TXD
5  GPIO_InitStrue.GPIO_Mode=GPIO_Mode_AF_PP;        //推挽输出
6  GPIO_InitStrue.GPIO_Speed=GPIO_Speed_10MHz;
7  GPIO_Init(GPIOA, &GPIO_InitStrue);
8
9  //接收端PA10配置
10 GPIO_InitStrue.GPIO_Pin=GPIO_Pin_10;             //接收端-RXD
11 GPIO_InitStrue.GPIO_Mode=GPIO_Mode_IN_FLOATING;//浮空输入
12 GPIO_InitStrue.GPIO_Speed=GPIO_Speed_10MHz;
13 GPIO_Init(GPIOA, &GPIO_InitStrue);
```

3、串口初始化（115200-8-N-1）

```
1  USART_InitTypeDef     USART_InitStrue;
2
3  USART_InitStrue.USART_BaudRate=115200;                              //设置波特率-115200MHz
4  USART_InitStrue.USART_HardwareFlowControl=USART_HardwareFlowControl_None;//硬件流控制-不使用
5  USART_InitStrue.USART_Mode=USART_Mode_Rx| USART_Mode_Tx;     //使能设置-发送接收都使能
6  USART_InitStrue.USART_Parity=USART_Parity_No;                       //奇偶校验-不使用奇偶校验
7  USART_InitStrue.USART_StopBits=USART_StopBits_1;                    //停止位-一个停止位
8  USART_InitStrue.USART_WordLength=USART_WordLength_8b                //字长-8位字长
9
10 USART_Init(USART1, &USART_InitStrue);
```

4, 使能串口1:

- `USART_Cmd(USART1, ENABLE);`

5、清除发送完成标志

```
USART_ClearFlag(USART1,USART_FLAG_TC);
```

在发送第一位数据之前需要如此操作，否则容易造成第一个数据发送不出去。

位6	TC：发送完成 (Transmission complete) 当包含有数据的一帧发送完成后，并且TXE=1时，由硬件将该位置'1'。如果USART_CR1中的TCIE为'1'，则产生中断。由软件序列清除该位(先读USART_SR，然后写入USART_DR)。TC位也可以通过写入'0'来清除，只有在多缓存通讯中才推荐这种清除程序。 0：发送还未完成； 1：发送完成。

6、发送数据并确保发送已完成

```
USART_SendData(USART1,'A');

while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
```

位6	TC：发送完成 (Transmission complete) 当包含有数据的一帧发送完成后，并且TXE=1时，由硬件将该位置'1'。如果USART_CR1中的TCIE为'1'，则产生中断。由软件序列清除该位(先读USART_SR，然后写入USART_DR)。TC位也可以通过写入'0'来清除，只有在多缓存通讯中才推荐这种清除程序。 0：发送还未完成； 1：发送完成。

串口测试程序2

- 串口硬件连接
  - PA9-Tx
  - PA10-Rx
  - 电脑通过USB线连接开发板（将USB虚拟为串口使用）
- 程序功能：
  - 通过可变参数函数自定义一个类似于printf（）的函数Uart_Printf(),实现将指定格式的数据通过UART1输出。

Main函数

```
15  int main()
16  {
17      RCC_Config();
18      GPIO_Config();
19      USART_Config();
20      USART_ClearFlag(USART1,USART_FLAG_TC);
21      Uart_Printf("hello,world 安徽工业大学计算机学院欢迎你\n");
22      Uart_Printf("a=%d\n",10);
23  }
```

Rcc_config函数

```
25  void RCC_Config(void)
26  {
27    //SystemInit();
28    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
29    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
30  }
```

GPIO_Config函数

```
32  void GPIO_Config(void)
33  {
34    GPIO_InitTypeDef  GPIO_InitStructure;
35
36    //GPIO_StructInit(&GPIO_InitStructure);
37    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;          //USART1_TX
38    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
39    GPIO_Init(GPIOA,&GPIO_InitStructure);
40
41    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;         //USART1_RX
42    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
43    GPIO_Init(GPIOA,&GPIO_InitStructure);
44  }
```

USART_Config函数

```
46  void USART_Config(void)
47  {
48    USART_InitTypeDef USART_InitStructure;
49
50    USART_InitStructure.USART_BaudRate=115200;
51    USART_InitStructure.USART_WordLength=USART_WordLength_8b;
52    USART_InitStructure.USART_StopBits=USART_StopBits_1;
53    USART_InitStructure.USART_Parity=USART_Parity_No;
54    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
55    USART_InitStructure.USART_Mode=USART_Mode_Tx;
56    USART_Init(USART1,&USART_InitStructure);
57
58    USART_Cmd(USART1,ENABLE);
59  }
```

单字节发送函数Uart_SendByte

```
61  void Uart_SendByte(int data)
62  {
63    if(data=='\n')
64    {
65      while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
66      USART_SendData(USART1,'\r');
67    }
68    while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
69      USART_SendData(USART1,data);
70  }
```

字符串发送函数Uart_SendString

```
72  void Uart_SendString(char *pt)
73  {
74      char *dst=pt;
75      while(dst< pt+strlen(pt))
76      {
77          Uart_SendByte(*dst++);
78      }
79  }
80
```

利用可变参数函数封装字符串发送函数

```
81  void Uart_Printf(const char *fmt,...)
82  {
83      va_list ap;
84      char string[50];
85      va_start(ap,fmt);
86      vsprintf(string,fmt,ap);
87      va_end(ap);
88      Uart_SendString(string);
89  }
```

- 可变参数函数的参数列表分为两部分：固定参数和个数可变的可变参数，函数中至少有一个固定参数；可变参数由于个数不确定，声明时用“…”表示。

```
81  void Uart_Printf(const char *fmt,...)
82  {
83      va_list ap;
84      char string[50];
85      va_start(ap,fmt);
86      vsprintf(string,fmt,ap);
87      va_end(ap);
88      Uart_SendString(string);
89  }
```

- Va_list ap:定义一个指向可变参数列表的指针
- Va_start(ap,argN):使参数列表指针ap指向第一个可变参数，argN是最后一个固定参数。
- Va_end(ap):清空参数列表，并置参数指针ap无效，结束可变参数的获取。
- Vsprintf(string,fmt,ap)的作用是将ap按格式fmt写入字符串string中

```
81  void Uart_Printf(const char *fmt,...)
82  {
83      va_list ap;
84      char string[50];
85      va_start(ap,fmt);
86      vsprintf(string,fmt,ap);
87      va_end(ap);
88      Uart_SendString(string);
89  }
```

- 函数的基本流程：
  - 先开辟一块区域存储可变参数
  - 调用vsprintf将可变参数按照指定格式复制到缓冲区中
  - 调用uart_sendstring将缓冲区的数据打印到串口

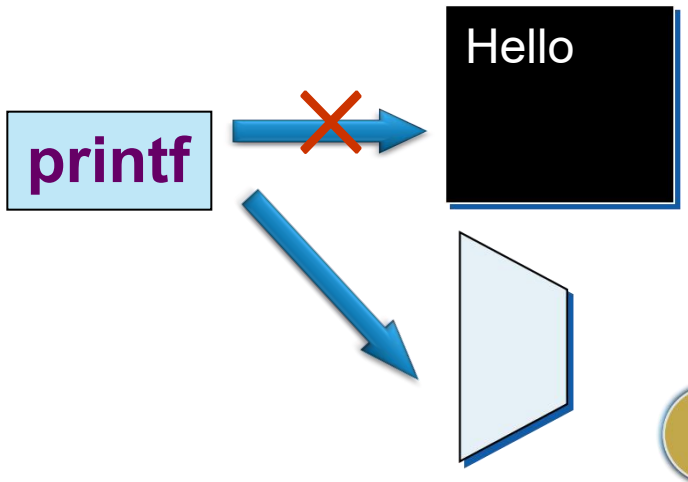
串口测试程序3

- 串口硬件连接
  - PA9-Tx
  - PA10-Rx
  - 电脑通过USB线连接开发板（将USB虚拟为串口使用）
- 程序功能：
  - 将printf（）函数的输出从显示器输出重定向至串口输出。

重定向（Retarget）

- C语言输出函数默认设备：键盘和显示器
- 要使用printf函数，需要重定向
  - 将输出的信息重新定向去到外设（USART1端口）
- 用户可以重新编写C语言的库函数
  - 当C编译器检查到与C库函数相同名称的函数时
  - 优先采用用户编写的函数，实现重定向

1. 编写fputc函数
2. 使用微库



USART的数据发送函数

- 发送数据的函数

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

  - 参数Data就是要发送的数据
  - 虽然是一个16位数据，实际上只使用其低8位

1. 编写fputc函数

- 在C标准库函数中，printf函数实质是一个宏
- 需要调用fputc实现一个字符输出

```
int fputc(int ch, FILE *f)
{
  USART_SendData(USART1, (uint8_t) ch);
  while( USART_GetFlagStatus(USART1,
 USART_FLAG_TC) == RESET);
    return ch;
}
```

- USART_GetFlagStatus用于检测发送是否完成

USART_GetFlagStatus函数

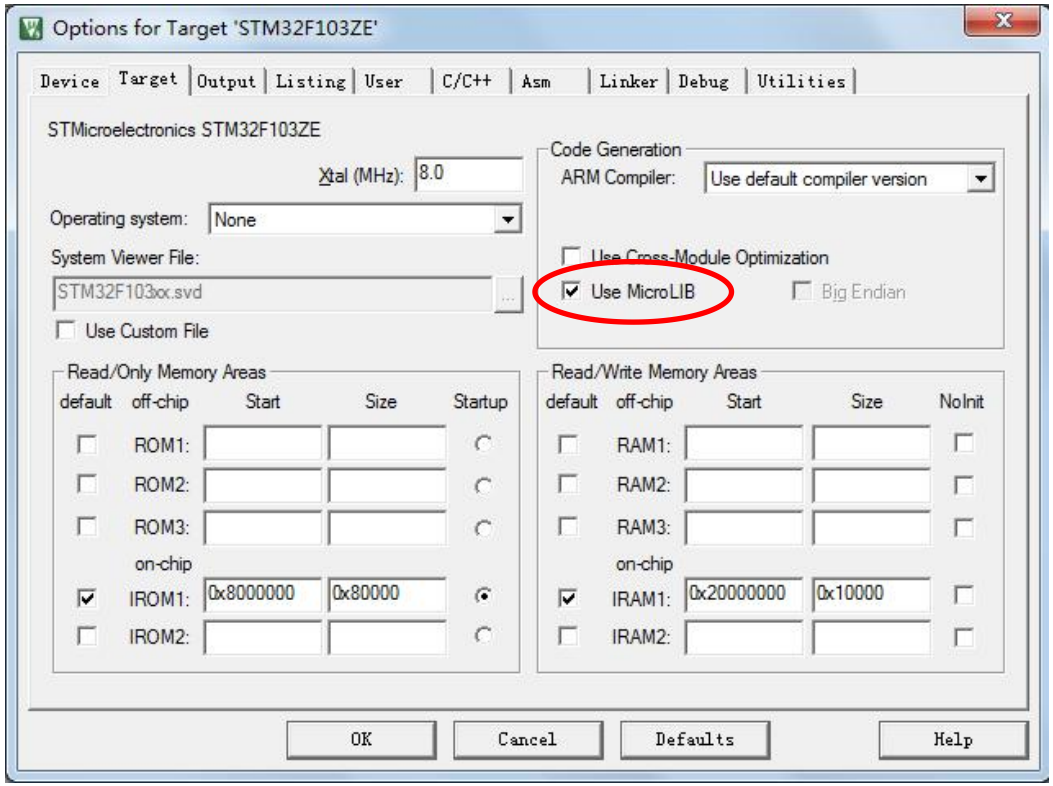
- 用于检测发送完成

  FlagStatus USART_GetFlagStatus (USART_TypeDef * USARTx, uint16_t USART_FLAG)

- 要检测的状态是参数USART_FLAG
  - USART_FLAG_CTS（CTS改变标志，UART4和UART5上不可用）
  - USART_FLAG_LBD（LIN中止检测标志）
  - USART_FLAG_TXE（发送数据寄存器空标志）
  - USART_FLAG_TC（发送完成标志）
  - USART_FLAG_RXNE（接收数据寄存器非空标志）
  - USART_FLAG_IDLE（空闲线检测标志）
  - USART_FLAG_ORE（溢出错误标志）
  - USART_FLAG_NE（噪声错误标志）
  - USART_FLAG_FE（帧错误标志）
  - USART_FLAG_PE（校验错标志）
- 返回值USART_FLAG说明置位（SET）或复位（RESET）

2. 使用微库（MicroLib）

- 将printf函数重定向到USART端口
  需要使用微库（要先具有库，才能重定向）



在MDK集成环境的目标选项中，从代码生成栏（Code Generation），选择Use MicroLIB

使用C标准输入输出函数，需包含stdio.h

MicroLIB微库

- 微库是Keil MDK特别为嵌入式应用编写的小型C库
  - 仅实现了基本的、简单的函数，例如printf
  - 不能使用高级的fprintf、fopen等
- 如果仍使用标准C库
  - 需要用户重新编写使用半主机模式的函数
  - 在MDK安装目录中有一个文件retarget.c，为用户编写自己的函数提供的模板，用户可以将文件复制到工程目录中，添加到项目中，并进行修改

Uartdemo3

串口测试程序5

- 串口硬件连接
  - PA9-Tx
  - PA10-Rx
  - PB5—LED1
  - 电脑通过USB线连接开发板（将USB虚拟为串口使用）
- 程序功能：
  - STM32开发板通过UART1接收上位机（pc）命令，并执行相应动作。
  - 当pc通过串口调试助手发送“LEDON”命令（以回车作为结束符）时，开发板LED1（PB5控制）点亮；
  - 当pc通过串口调试助手发送“LEDOFF”命令（以回车作为结束符）时，开发板LED1（PB5控制）熄灭；

Main（）函数

```
12  unsigned char CmdBuffer[10];
13
14  int main()
15  {
16    RCC_Config();
17    GPIO_Config();
18    USART_Config();
19    NVIC_Config();
20    memset(CmdBuffer,10,0);
21
22    while(1)
23    {
24      if(strstr(CmdBuffer,"LEDON"))
25      {
26        GPIO_ResetBits(GPIOB,GPIO_Pin_5);
27        memset(CmdBuffer,10,0);
28      }
29      if(strstr(CmdBuffer,"LEDOFF"))
30      {
31        GPIO_SetBits(GPIOB,GPIO_Pin_5);
32        memset(CmdBuffer,10,0);
33      }
34    }
35  }
```

- 全局缓冲区CmdBuffer，实现主程序和中断服务程序之间数据传递
- 主程序不断检测CmdBuffer的内容是否是约定好的操纵命令
- 因为使用了串口1中断，所以需要配置NVIC，NVIC_Config()内容暂不深究（讲过NVIC后回头再看）

Stm32f10x_it.c中增加相应的中断服务程序

```
26  #include "stm32f10x_usart.h"   //新增
27  //用户新增
28  unsigned char Buffer[10];
29  extern unsigned char CmdBuffer[10];
30  unsigned char RxCounter=0;
31
32  void USART1_IRQHandler(void)
33  {
34    unsigned int i=0;
35    if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
36    {
37      Buffer[RxCounter++]=USART_ReceiveData(USART1);
38      if((Buffer[RxCounter-2]==0x0d)&&(Buffer[RxCounter-1]==0x0a))
39      {
40        for(i=0;i<RxCounter;i++)
41        {
42          CmdBuffer[i]=Buffer[i];
43        }
44        CmdBuffer[RxCounter]=0;
45      RxCounter=0;
46      }
47    }
48  }
49  //用户新增结束
```

startup_stm32f10x_hd.s中中断服务程序名称约定

```
62  __Vectors       DCD     __initial_sp               ; Top of Stack
63                  DCD     Reset_Handler              ; Reset Handler
64                  DCD     NMI_Handler                ; NMI Handler
65                  DCD     HardFault_Handler          ; Hard Fault Handler
66                  DCD     MemManage_Handler          ; MPU Fault Handler
67                  DCD     BusFault_Handler           ; Bus Fault Handler
68                  DCD     UsageFault_Handler         ; Usage Fault Handler
69                  DCD     0                          ; Reserved
70                  DCD     0                          ; Reserved
71                  DCD     0                          ; Reserved
72                  DCD     0                          ; Reserved
73                  DCD     SVC_Handler                ; SVCall Handler
74                  DCD     DebugMon_Handler           ; Debug Monitor Handler
75                  DCD     0                          ; Reserved
76                  DCD     PendSV_Handler             ; PendSV Handler
77                  DCD     SysTick_Handler            ; SysTick Handler
```

- 这些是Cortex-M3规定的异常处理程序

```
; External Interrupts
DCD     WWDG_IRQHandler             ; Window Watchdog
DCD     PVD_IRQHandler              ; PVD through EXTI Line detect
DCD     TAMPER_IRQHandler           ; Tamper
DCD     RTC_IRQHandler              ; RTC
DCD     FLASH_IRQHandler            ; Flash
DCD     RCC_IRQHandler              ; RCC
DCD     EXTI0_IRQHandler            ; EXTI Line 0
DCD     EXTI1_IRQHandler            ; EXTI Line 1
DCD     EXTI2_IRQHandler            ; EXTI Line 2
DCD     EXTI3_IRQHandler            ; EXTI Line 3
DCD     EXTI4_IRQHandler            ; EXTI Line 4
DCD     DMA1_Channel1_IRQHandler    ; DMA1 Channel 1
DCD     DMA1_Channel2_IRQHandler    ; DMA1 Channel 2
DCD     DMA1_Channel3_IRQHandler    ; DMA1 Channel 3
DCD     DMA1_Channel4_IRQHandler    ; DMA1 Channel 4
DCD     DMA1_Channel5_IRQHandler    ; DMA1 Channel 5
DCD     DMA1_Channel6_IRQHandler    ; DMA1 Channel 6
DCD     DMA1_Channel7_IRQHandler    ; DMA1 Channel 7
DCD     ADC1_2_IRQHandler           ; ADC1 & ADC2
DCD     USB_HP_CAN1_TX_IRQHandler   ; USB High Priority or CAN1 TX
DCD     USB_LP_CAN1_RX0_IRQHandler  ; USB Low  Priority or CAN1 RX0
DCD     CAN1_RX1_IRQHandler         ; CAN1 RX1
DCD     CAN1_SCE_IRQHandler         ; CAN1 SCE
DCD     EXTI9_5_IRQHandler          ; EXTI Line 9..5
DCD     TIM1_BRK_IRQHandler         ; TIM1 Break
DCD     TIM1_UP_IRQHandler          ; TIM1 Update
DCD     TIM1_TRG_COM_IRQHandler     ; TIM1 Trigger and Commutation
DCD     TIM1_CC_IRQHandler          ; TIM1 Capture Compare
DCD     TIM2_IRQHandler             ; TIM2
DCD     TIM3_IRQHandler             ; TIM3
DCD     TIM4_IRQHandler             ; TIM4
DCD     I2C1_EV_IRQHandler          ; I2C1 Event
```

- 这些与外部中断、片上外设相对应

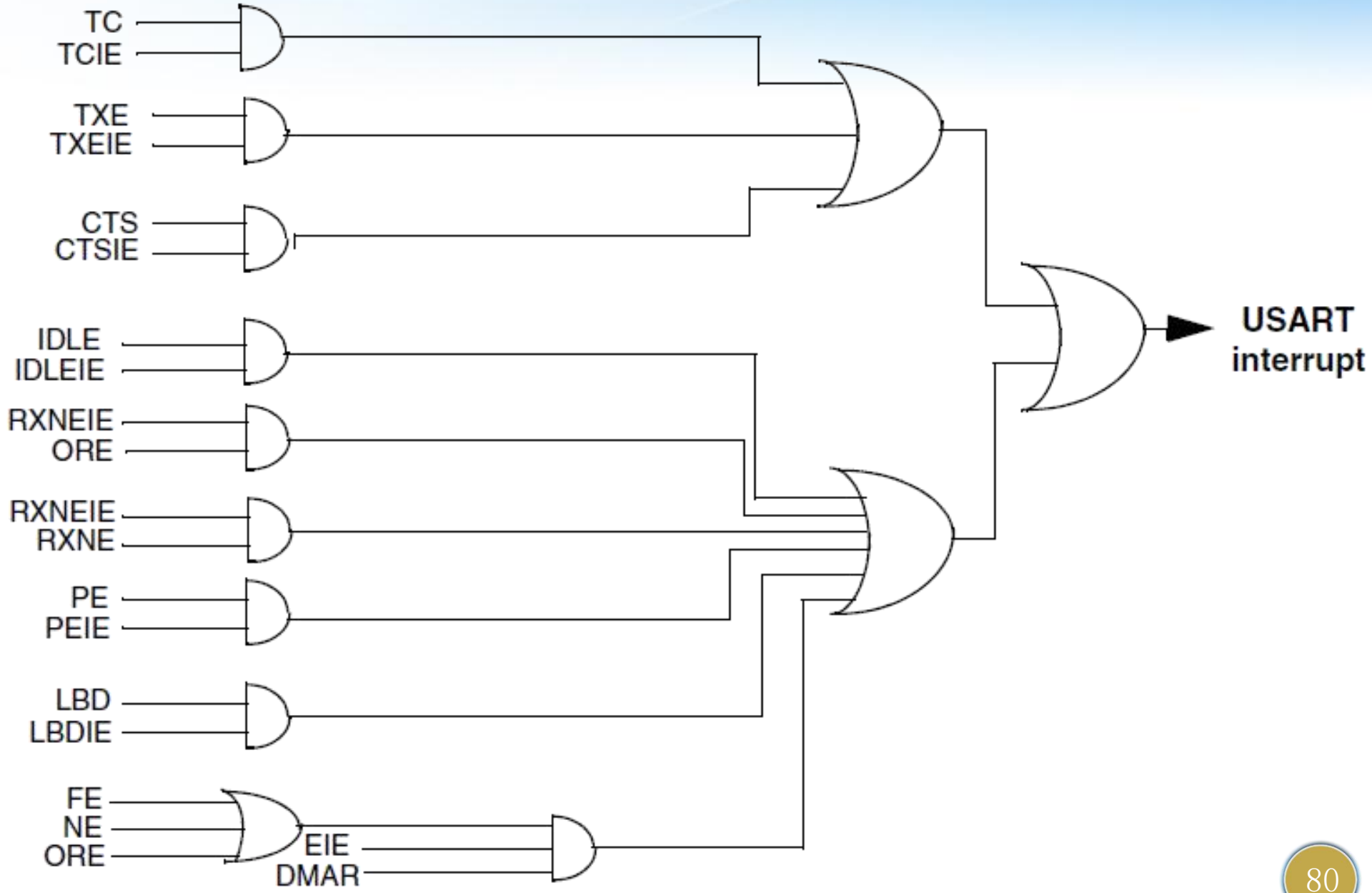
```
112                 DCD     I2C1_ER_IRQHandler          ; I2C1 Error
113                 DCD     I2C2_EV_IRQHandler          ; I2C2 Event
114                 DCD     I2C2_ER_IRQHandler          ; I2C2 Error
115                 DCD     SPI1_IRQHandler             ; SPI1
116                 DCD     SPI2_IRQHandler             ; SPI2
117                 DCD     USART1_IRQHandler           ; USART1
118                 DCD     USART2_IRQHandler           ; USART2
119                 DCD     USART3_IRQHandler           ; USART3
120                 DCD     EXTI15_10_IRQHandler        ; EXTI Line 15..10
121                 DCD     RTCAlarm_IRQHandler         ; RTC Alarm through EXTI Line
122                 DCD     USBWakeUp_IRQHandler        ; USB Wakeup from suspend
123                 DCD     TIM8_BRK_IRQHandler         ; TIM8 Break
124                 DCD     TIM8_UP_IRQHandler          ; TIM8 Update
125                 DCD     TIM8_TRG_COM_IRQHandler     ; TIM8 Trigger and Commutation
126                 DCD     TIM8_CC_IRQHandler          ; TIM8 Capture Compare
127                 DCD     ADC3_IRQHandler             ; ADC3
128                 DCD     FSMC_IRQHandler             ; FSMC
129                 DCD     SDIO_IRQHandler             ; SDIO
130                 DCD     TIM5_IRQHandler             ; TIM5
131                 DCD     SPI3_IRQHandler             ; SPI3
132                 DCD     UART4_IRQHandler            ; UART4
133                 DCD     UART5_IRQHandler            ; UART5
134                 DCD     TIM6_IRQHandler             ; TIM6
135                 DCD     TIM7_IRQHandler             ; TIM7
136                 DCD     DMA2_Channel1_IRQHandler    ; DMA2 Channel1
137                 DCD     DMA2_Channel2_IRQHandler    ; DMA2 Channel2
138                 DCD     DMA2_Channel3_IRQHandler    ; DMA2 Channel3
139                 DCD     DMA2_Channel4_5_IRQHandler ; DMA2 Channel4 & Channel5
140 __Vectors_End
```

Uartdemo5

【例】接收中断驱动LED灯点亮

- USART接口的中断应用、实现功能
  - 从PC机键盘输入数字1、2或3，
  - 从PC机串口发送给嵌入式系统的USART1接口
  - USART1接收数字后，触发中断
  - 中断服务程序获取具体的数字，
    并相应控制LED1、LED2或LED3灯点亮

USART中断连接


TC
TCIE
TXE
TXEIE
CTS
CTSIE
IDLE
IDLEIE
RXNEIE
ORE
RXNEIE
RXNE
PE
PEIE
LBD
LBDIE
FE
NE
ORE
EIE
DMAR
USART
interrupt

USART中断相关函数

- USART中断配置函数

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

- 获取中断状态函数

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

- 清除中断标志函数

```
void USART_ClearITPendingBit ( USART_TypeDef *
  USARTx, uint16_t  USART_IT )
```

USART中断标志和事件标志

中断请求	中断标志	事件标志
CTS改变	USART_IT_CTS	USART_FLAG_CTS
LIN中止检测中断	USART_IT_LBD	USART_FLAG_LBD
发送数据寄存器空中断	USART_IT_TXE	USART_FLAG_TXE
发送完成中断	USART_IT_TC	USART_FLAG_TC
接收数据寄存器非空中断	USART_IT_RXNE	USART_FLAG_RXNE
空闲线检测中断	USART_IT_IDLE	USART_FLAG_IDLE
校验错中断	USART_IT_PE	USART_FLAG_PE
错误中断	USART_IT_ERR	
溢出错误中断	USART_IT_ORE	USART_FLAG_ORE
噪声错误中断	USART_IT_NE	USART_FLAG_NE
帧错误中断	USART_IT_FE	USART_FLAG_FE

USART中断初始化

（1）NVIC 初始化配置

……

```
NVIC_InitStructure.NVIC_IRQChannel=USART1_IRQn;
```

……

（2）USART初始化配置

……

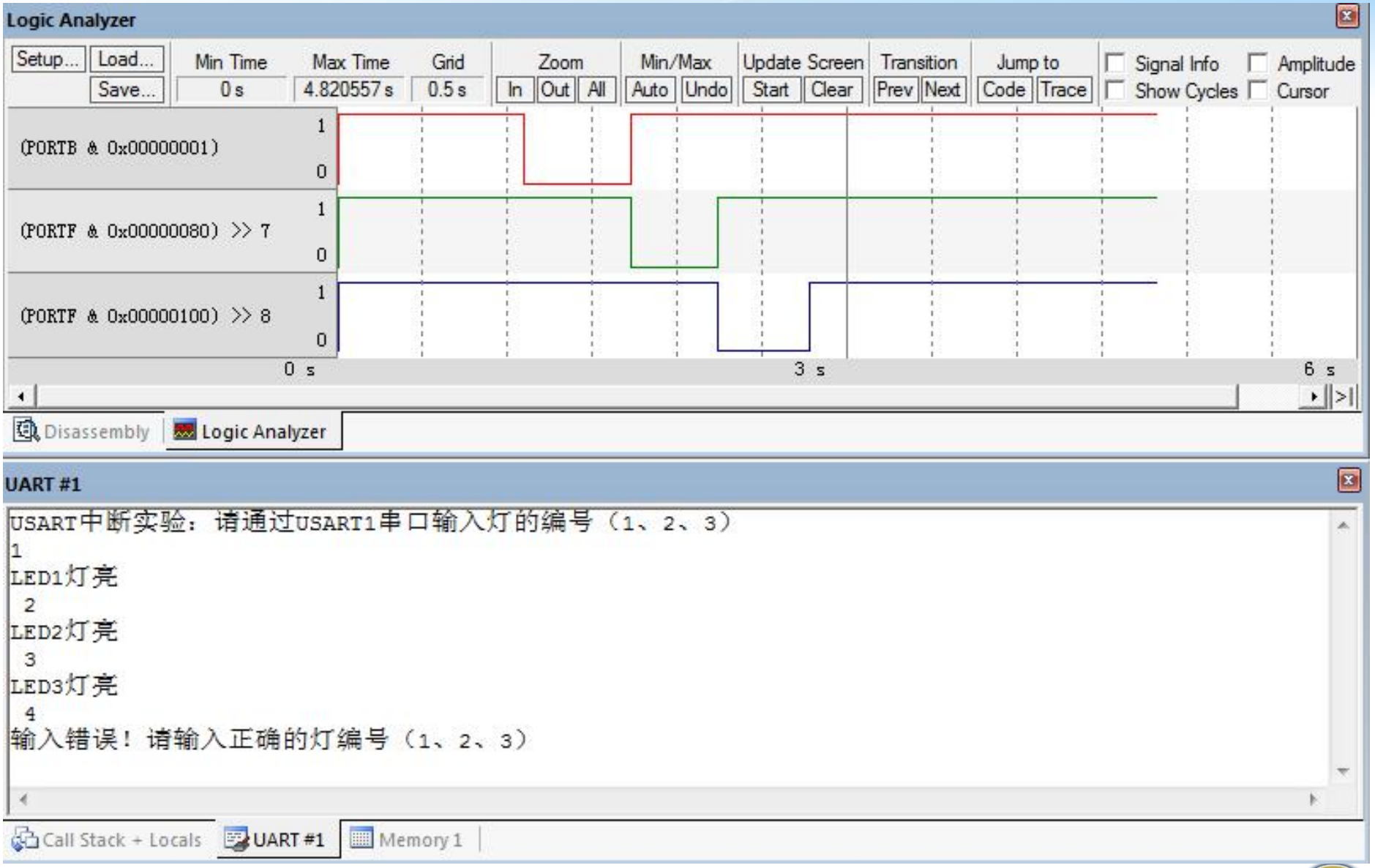
（3）USART中断配置

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

USRAT1中断服务程序

```
void USART1_IRQHandler(void)
{  uint8_t ch;
  LED_Off_all();
  if(USART_GetITStatus(USART1,USART_IT_RXNE)!= RESET)
    { ch= (uint8_t)USART_ReceiveData(USART1);
      printf("%c\n",ch);
      switch(ch)
      {
     case '1':LED_On(1);printf("LED1灯亮\n"); break;
     ……
      }
    }
}
```

软件模拟运行


Logic Analyzer
Setup...
Load...
Save...
Min Time
0 s
Max Time
4.820557 s
Grid
0.5 s
Zoom
In
Out
All
Min/Max
Auto
Undo
Update Screen
Start
Clear
Transition
Prev
Next
Jump to
Code
Trace
Signal Info
Show Cycles
Amplitude
Cursor
(PORTB & 0x00000001)
(PORTF & 0x00000080) >> 7
(PORTF & 0x00000100) >> 8
1
0
0 s
3 s
6 s
Disassembly
Logic Analyzer
UART #1
USART中断实验：请通过USART1串口输入灯的编号（1、2、3）
1
LED1灯亮
2
LED2灯亮
3
LED3灯亮
4
输入错误！请输入正确的灯编号（1、2、3）
Call Stack + Locals
UART #1
Memory 1