

第九章 串口通信

主讲人：漆强

电子科技大学

ytqiqiang@163.com

本章内容



串口通信概述



HAL库外设初始化设计思想



轮询方式的串口通信



中断方式的串口通信



DMA方式的串口通信

教学目标



了解通信的基本概念



了解HAL库的外设初始化设计思想



熟练掌握三种方式下的串口通信编程方法



电子科技大学
University of Electronic Science and Technology of China

9.1 串口通信概述

1 计算机通信的基本概念

计算机通信的概念

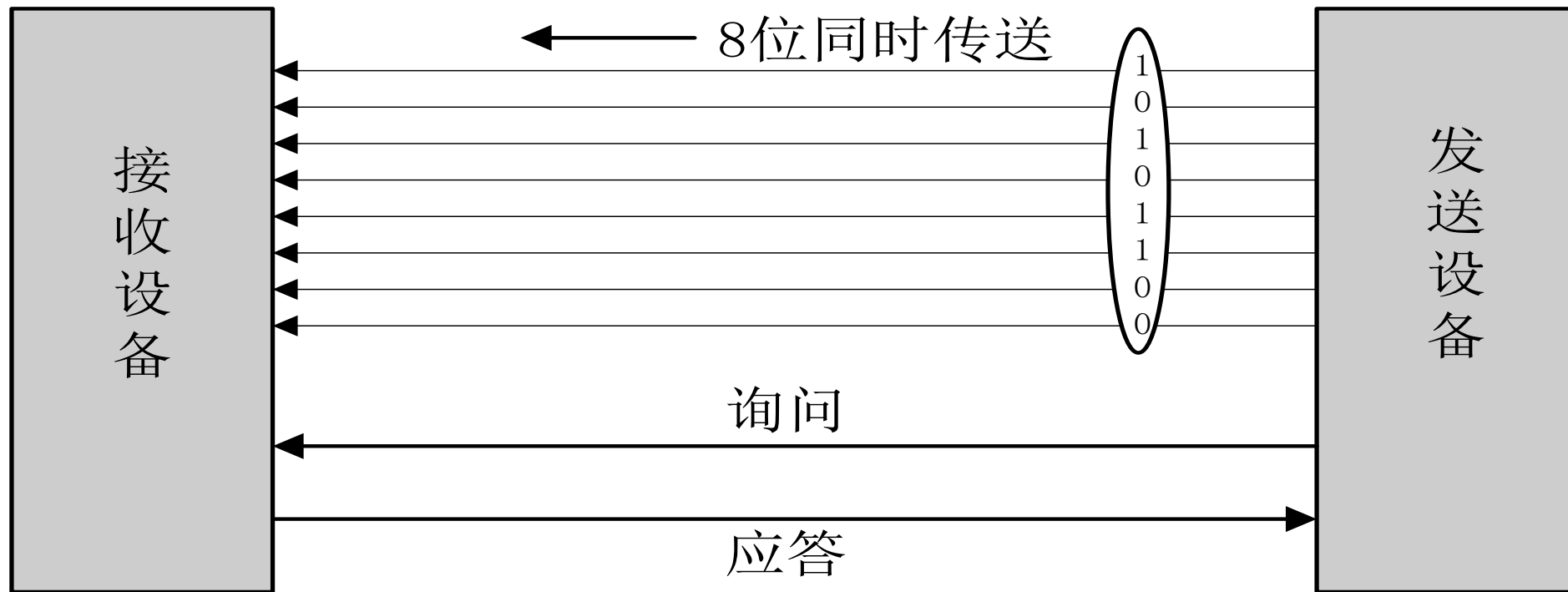
计算机通信

将计算机技术和通信技术相结合，完成计算机与外部设备或计算机与计算机之间的信息交换。按照数据传输方式的不同，可以分为串行通信和并行通信两类。

- ❑ 串行通信：数据逐位传输
- ❑ 并行通信：多位数据同时传输

并行通信

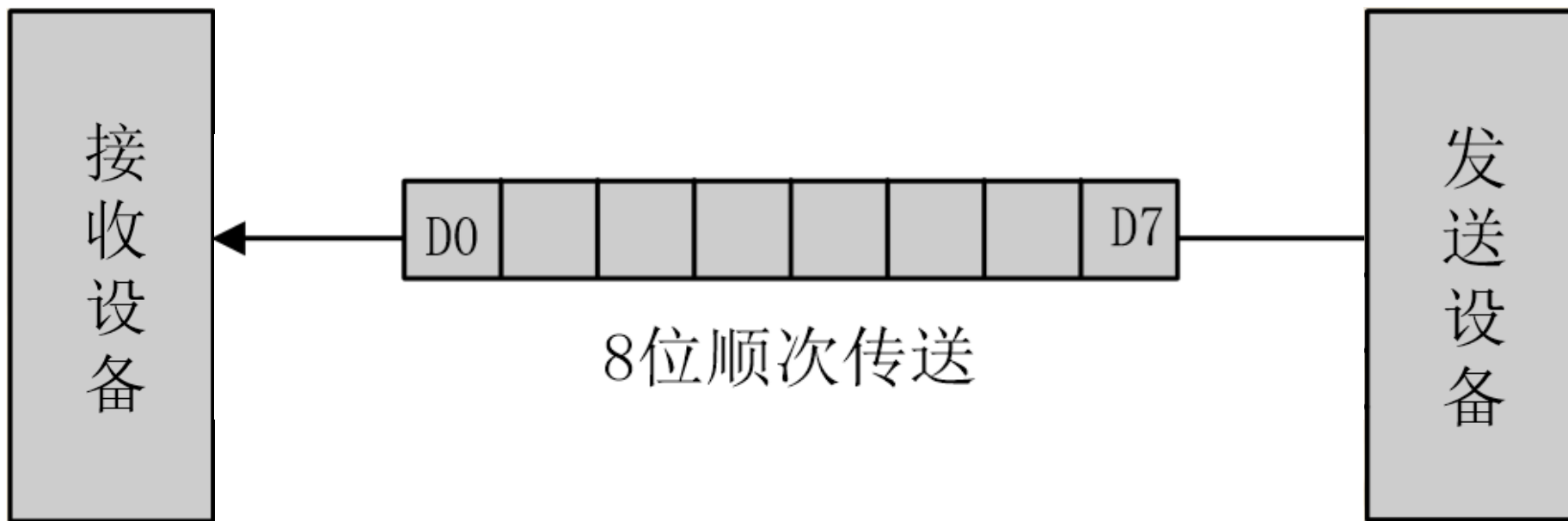
并行通信的特点



特点：多位数据同时传输，传输控制简单，传输速度快，但是在长距离传输时硬件成本较高。

串行通信

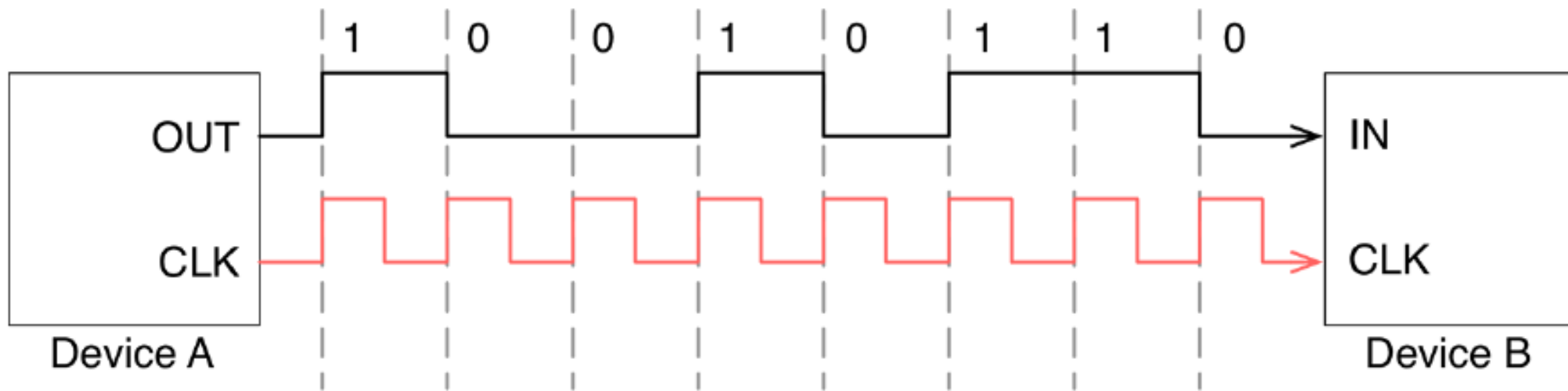
串行通信的特点



特点：数据逐位传输，传输线少，长距离传输时成本低，但数据的传输控制较复杂。按照实现数据同步的方式，可以分为同步串行和异步串行两种。

同步串行

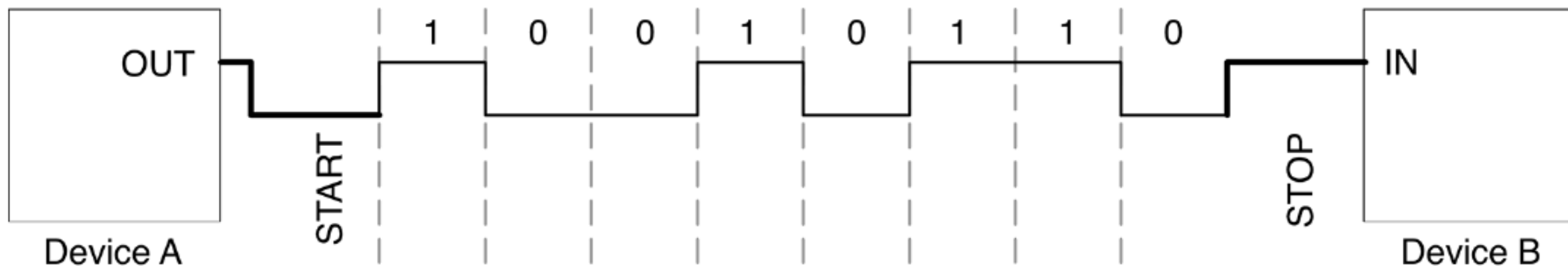
同步串行通信的特点



特点：数据传输以数据块（一组字符）为单位，在一个数据块内，字符与字符间无间隔，收发双方依靠独立的时钟线进行信号的同步。适用于大批量的数据传输。

异步串行

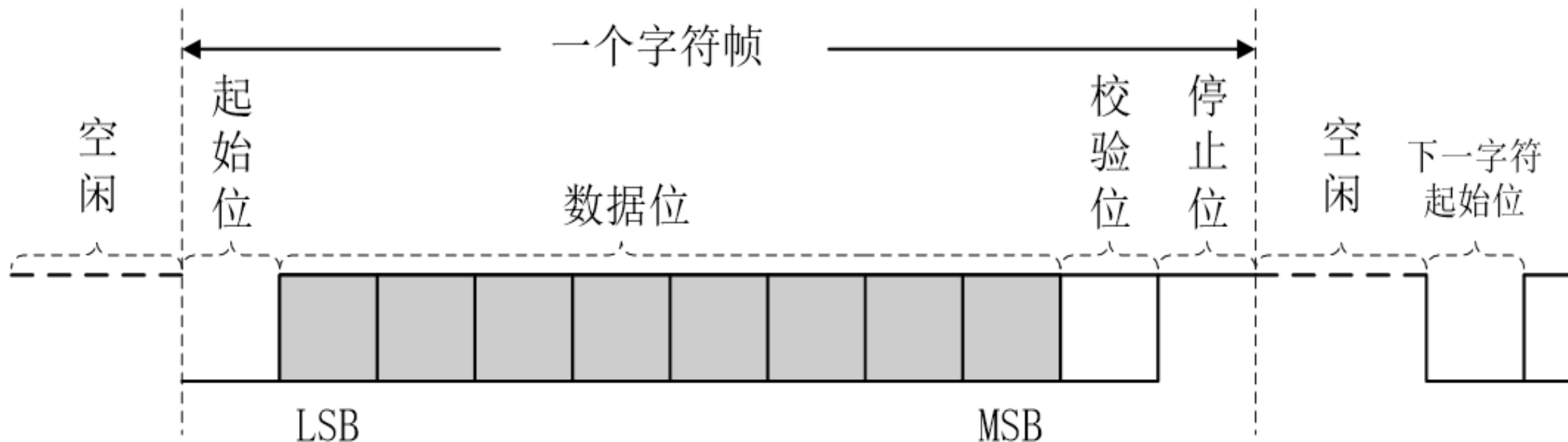
异步串行通信的特点



特点：数据传输以单个字符为单位，字符和字符之间的间隙任意，字符内部每一位持续的时间相同。收发双方没有专门的时钟信号，而是依靠事先约定的字符格式和通信速率来完成通信。

字符格式

异步串行通信的字符格式

☐ 起始位☐ 数据位☐ 校验位☐ 停止位

常用字符格式：1位起始位 8位数据位 无奇偶校验 1位停止位

异步串行通信的通信速率

波特率

每秒钟传送二进制数码的位数，以**bit/s (bps)** 为单位。

- 常用的波特率有：9600、19200、38400、57600和115200；
- 波特率为115200，表示每秒传输115200位，且每一位数据在数据线上持续时间为 $T_{\text{bit}} = 1/115200 \approx 8.68\mu\text{s}$ 。

通信准确性

异步串行通信的两个关键点

决定了字符中数据的
传输形式

字符格式

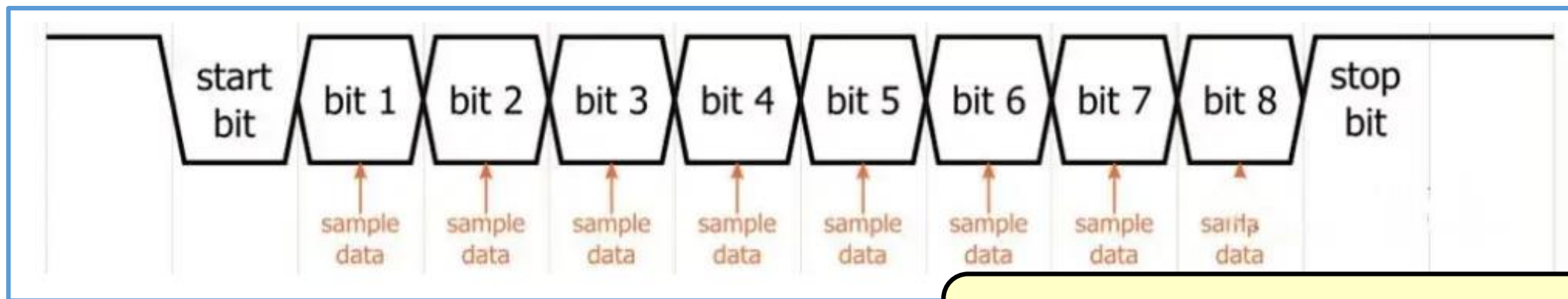
决定了字符中每一
位数据的持续时间

波特率

字符格式实例：1位起始位 8位数据位 无奇偶校验 1位停止位



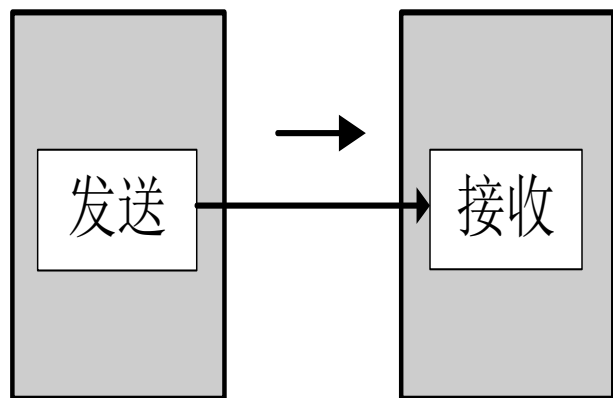
异步串行通信的数据接收过程



接收过程的本质是数据采集，假设接收端的采样时钟是波特率的16倍。

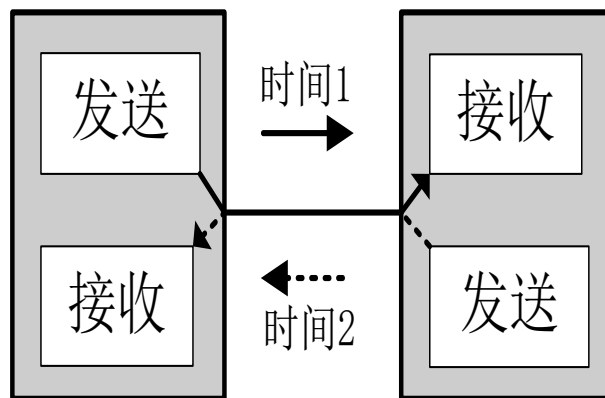
- ① 接收过程由起始位的下降沿启动；
- ② 接收端等待8个时钟周期，以便建立一个接近比特周期中间的采样点；
- ③ 接收端等待16个时钟周期，使其进入第一个数据位周期的中点；
- ④ 第一个数据位被采样并存储在接收寄存器中；
- ⑤ 串口模块在采样第二个数据位之前等待另外16个时钟周期；
- ⑥ 重复此过程，直到所有数据位都被采样和存储；
- ⑦ 由停止位的上升沿使数据线返回到空闲状态。

串口通信的数据传输方向



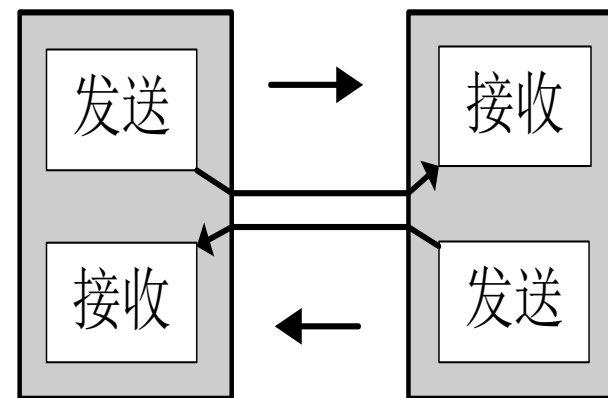
单工

特点：数据传输仅能沿一个方向，不能实现反向传输，只有一条通信线路。



半双工

特点：数据传输可以沿两个方向，但需要分时进行，也只有一条通信线路。



全双工

特点：数据可以同时进行双向传输，具有两条通信线路。典型实例：UART。

错误校验

三种常用的错误校验方式

奇偶
校验

奇校验表示数据中“1”的个数与校验位“1”的个数之和为奇数；偶校验表示数据中“1”的个数与校验位“1”的个数之和为偶数。

错误
校验代码和
校验

发送方将所发数据块求和，产生一个字节的校验字符附加到数据块末尾。接收方采用同样方式进行检测。

循环冗余
校验

通过某种数学运算实现有效信息与校验位之间的循环校验，常用于磁盘信息的传输、存储区的完整性校验等。

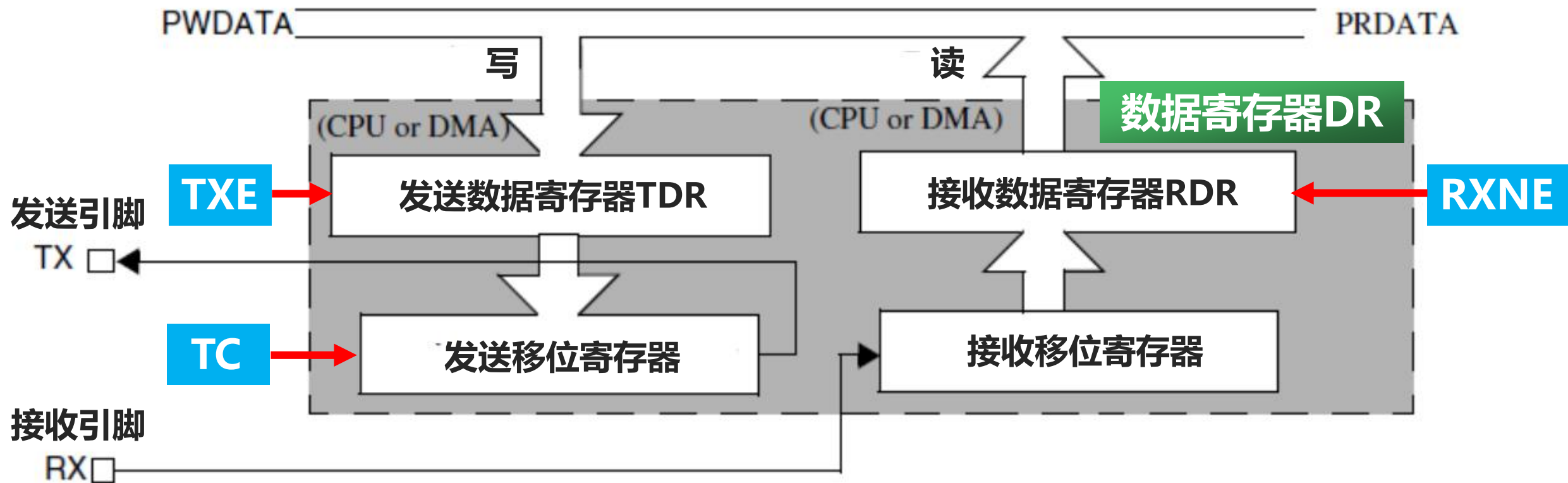


电子科技大学
University of Electronic Science and Technology of China

2 STM32的串口通信

串口收发单元

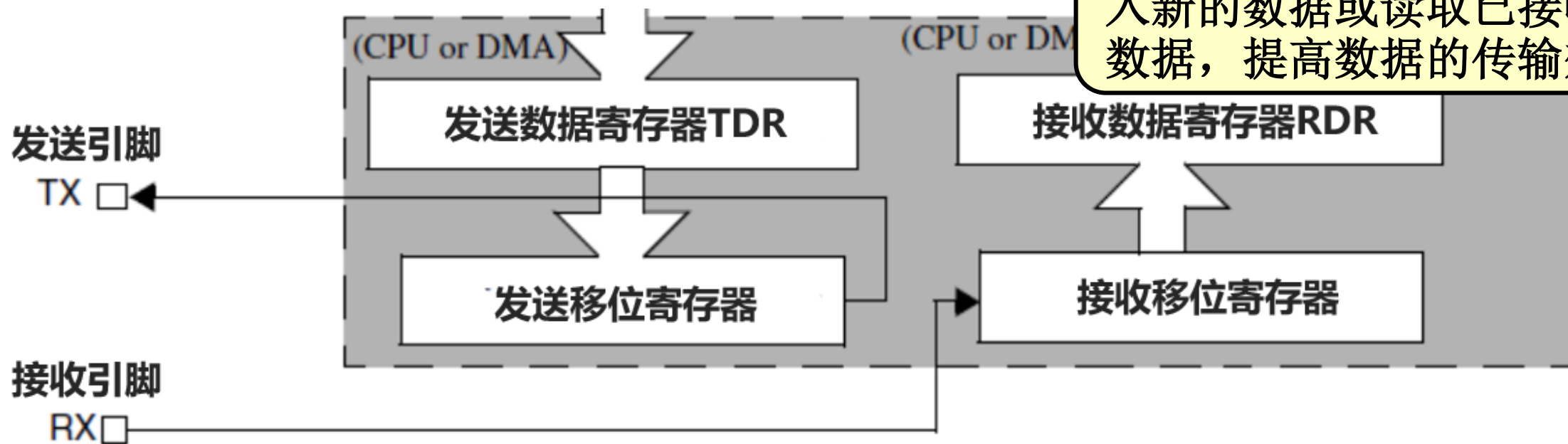
串口收发单元功能框图



串口收发单元主要利用数据寄存器DR，发送引脚TX，接收引脚RX，以及三个通信状态位TXE、TC和RXNE来完成数据的接收和发送。

数据寄存器DR

数据收发过程中，可同时写入新的数据或读取已接收的数据，提高数据的传输效率



- 数据寄存器DR在硬件上分为TDR和RDR两个寄存器，通过数据的流向进行区分，在结构设计上采用了双缓冲结构；
- 发送时，数据通过数据总线送入TDR寄存器，然后传送到发送移位寄存器完成数据转换，从并行数据转为串行数据，最后通过TX引脚发送；
- 接收时，数据通过RX引脚逐位送入接收移位寄存器，8位数据接收完成后，送入RDR寄存器，供用户读取。

通信状态标志位

标志位名称	含义
TXE	发送数据寄存器空标志。当TDR寄存器的内容已经传送到发送移位寄存器时，该位由硬件置1。如果串口控制寄存器CR1中的TXEIE位为1，将会触发发送数据寄存器空中断。 注意：当TXE置1时，数据有可能还在发送。
TC	发送完成标志。当发送移位寄存器的内容发送完成，同时TDR寄存器也为空时，该位由硬件置1，表示本次数据传输已经完成。如果串口控制寄存器CR1中的TCIE位为1，将会触发发送完成中断。 注意：当TC置1时，数据才是真正地发送完成。
RXNE	接收数据寄存器不为空标志。当移位寄存器的内容已经传送到接收数据寄存器RDR时，该位由硬件置1。如果串口控制寄存器CR1中的RXNEIE位为1，将会触发接收数据寄存器不为空中断。

在轮询方式下可以直接检测标志位；在中断方式下，需要在中断服务程序中通过检测不同的中断标志位，来判断出中断类型，然后执行后续的任务处理。

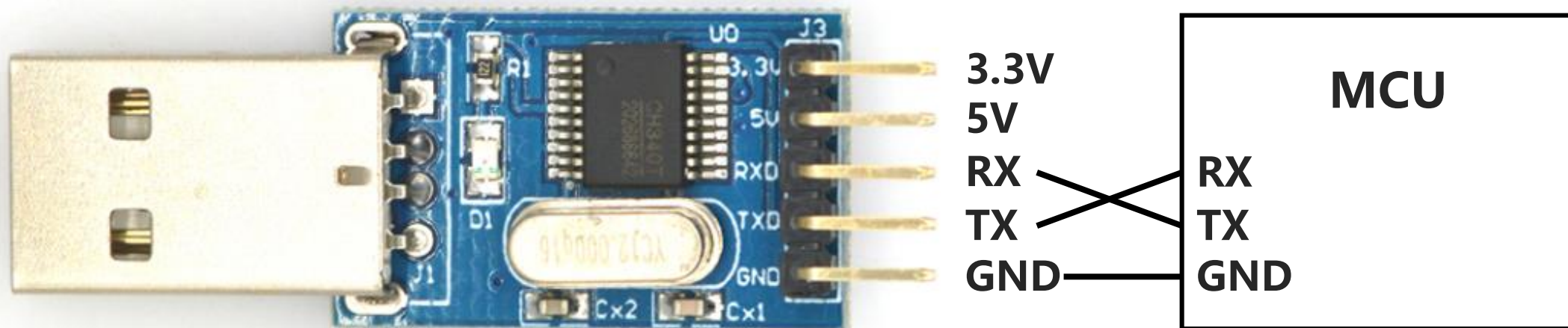
STM32F411芯片的UART引脚

查询芯片的数据手册
Datasheet

串口号	TX引脚	RX引脚
UART1	PA9/PA15/PB6	PA10/PB3/PB7
UART2	PA2	PA3
UART6	PA11/PC6	PA12/PC7

在Nucleo开发板上，默认使用UART2和PC通信，对应的TX引脚是PA2，RX引脚是PA3。利用板载的ST-Link仿真器，将USB接口转换为TTL串口，进而和PC通信。

USB转TTL串口模块



- MCU的TX引脚和串口转换模块的RX引脚连接
- MCU的RX引脚和串口转换模块的TX引脚连接
- MCU的地和串口转换模块的地连接，以确保电平的一致

ST-Link仿真器的虚拟串口功能

即USB转TTL串口功能





电子科技大学
University of Electronic Science and Technology of China

9.2 HAL库外设初始化设计思想



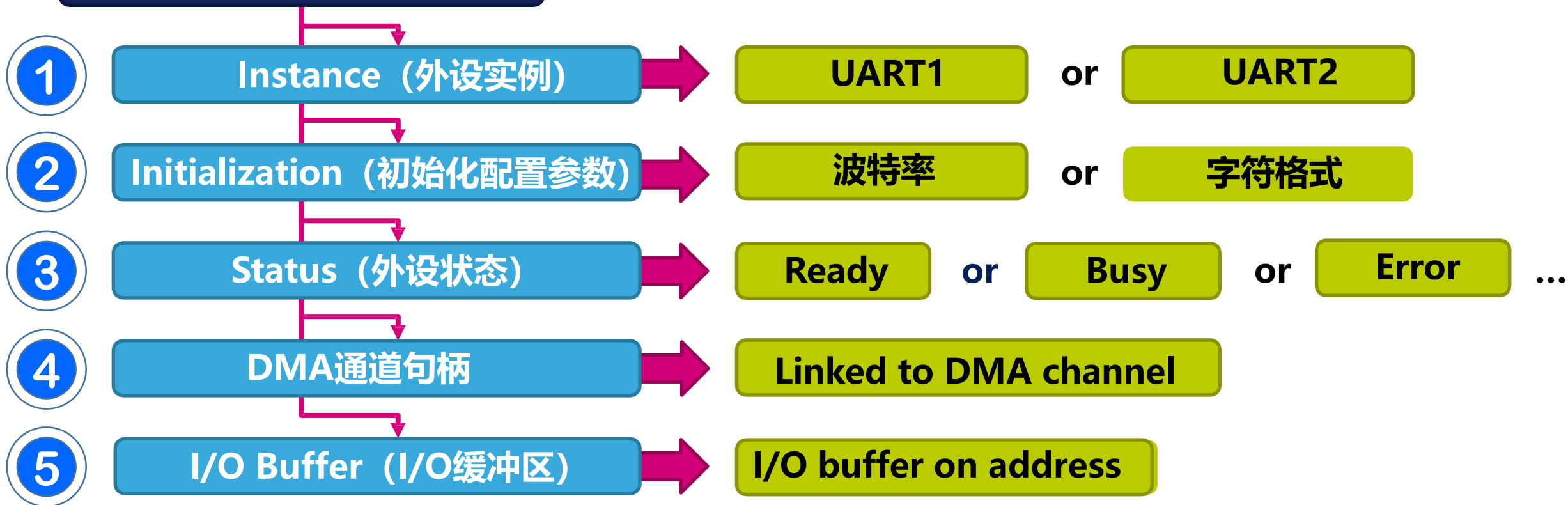
电子科技大学
University of Electronic Science and Technology of China

1 串口的数据类型定义

外设句柄

外设句柄数据类型的组成

Handle structure



1. **typedef struct**

串口实例，如UART1和UART2等

串口句柄定义

2. {

3. USART_TypeDef	*Instance;	// 串口寄存器的基地址定义
4. UART_InitTypeDef	Init;	// 串口初始化数据类型
5. uint8_t	*pTxBuffPtr;	// 串口发送缓冲区首地址
6. uint16_t	TxXferSize;	// 串口待发送数据个数
7. __IO uint16_t	TxXferCount;	// 串口发送数据计数器
8. uint8_t	*pRxBuffPtr;	// 串口接收缓冲区首地址
9. uint16_t	RxXferSize;	// 串口待接收数据个数
10. __IO uint16_t	RxXferCount;	// 串口接收数据计数器
11. DMA_HandleTypeDef	*hdmatx;	// 串口发送的 DMA 通道句柄定义
12. DMA_HandleTypeDef	*hdmarx;	// 串口接收的 DMA 通道句柄定义
13. HAL_LockTypeDef	Lock;	// 保护锁类型定义
14. __IO HAL_UART_StateTypeDef	gState;	// 串口全局状态和发送状态信息
15. __IO HAL_UART_StateTypeDef	RxState;	// 串口接收状态信息
16. __IO uint32_t	ErrorCode;	// 串口错误代码
17. }	UART_HandleTypeDef;	

```
1. typedef struct __UART_HandleTypeDef
2. {
3.     USART_TypeDef *Instance;           // 串口寄存器的基地址定义
4.     UART_InitTypeDef Init;             // 串口初始化数据类型
5.     uint8_t *pTxBuffPtr;               // 串口发送缓冲区首地址
6.     uint16_t TxXferSize;               // 串口待发送数据个数
7.     __IO uint16_t TxXferCount;         // 串口发送数据计数器
8.     uint8_t *pRxBuffPtr;              // 串口接收缓冲区首地址
9.     uint16_t RxXferSize;               // 串口待接收数据个数
10.    __IO uint16_t RxXferCount;          // 串口接收数据计数器
11.    DMA_HandleTypeDef *hdmatx;          // 串口发送的 DMA 通道句柄定义
12.    DMA_HandleTypeDef *hdmarx;          // 串口接收的 DMA 通道句柄定义
13.    HAL_LockTypeDef Lock;               // 保护锁类型定义
14.    __IO HAL_UART_StateTypeDef gState;  // 串口全局状态和发送状态信息
15.    __IO HAL_UART_StateTypeDef RxState; // 串口接收状态信息
16.    __IO uint32_t ErrorCode;            // 串口错误代码
17. } UART_HandleTypeDef;
```

串口初始化数据类型，它的成员变量为串口通信参数

串口句柄定义

```
1. typedef struct __UART_HandleTypeDef
2. {
3.     USART_TypeDef          USART_TypeDef; // 串口寄存器的基地址定义
4.     UART_InitTypeDef       UART_InitTypeDef; // 串口初始化数据类型
5.     uint8_t                *pTxBuffPtr; // 串口发送缓冲区首地址
6.     uint16_t               TxXferSize; // 串口待发送数据个数
7.     __IO uint16_t          TxXferCount; // 串口发送数据计数器
8.     uint8_t                *pRxBuffPtr; // 串口接收缓冲区首地址
9.     uint16_t               RxXferSize; // 串口待接收数据个数
10.    __IO uint16_t           RxXferCount; // 串口接收数据计数器
11.    DMA_HandleTypeDef       *hdmatx; // 串口发送的 DMA 通道句柄定义
12.    DMA_HandleTypeDef       *hdmarx; // 串口接收的 DMA 通道句柄定义
13.    HAL_LockTypeDef         Lock; // 保护锁类型定义
14.    __IO HAL_UART_StateTypeDef gState; // 串口全局状态和发送状态信息
15.    __IO HAL_UART_StateTypeDef RxState; // 串口接收状态信息
16.    __IO uint32_t           ErrorCode; // 串口错误代码
17. } UART_HandleTypeDef;
```

串口的I/O缓冲区

串口句柄定义

```
1. typedef struct __UART_HandleTypeDef
2. {
3.     USART_TypeDef          *Instance;      // 串口寄存器的基地址定义
4.     UART_InitTypeDef       Init;           // 串口初始化数据类型
5.     uint8_t                *pTxBuffPtr;    // 串口发送缓冲区首地址
6.     uint16_t               TxXferSize;     // 串口待发送数据个数
7.     __IO uint16_t          TxXferCount;    // 串口发送数据计数器
8.     uint8_t                *pRxBuffPtr;    // 串口接收缓冲区首地址
9.     uint16_t               // 串口待接收数据个数
10.    __IO uint16_t           RxXferCount;    // 串口接收数据计数器
11.    DMA_HandleTypeDef        *hdmatx;       // 串口发送的 DMA 通道句柄定义
12.    DMA_HandleTypeDef        *hdmarx;       // 串口接收的 DMA 通道句柄定义
13.    HAL_LockTypeDef          Lock;          // 保护锁类型定义
14.    __IO HAL_UART_StateTypeDef gState;      // 串口全局状态和发送状态信息
15.    __IO HAL_UART_StateTypeDef RxState;     // 串口接收状态信息
16.    __IO uint32_t            ErrorCode;     // 串口错误代码
17. } UART_HandleTypeDef;
```

串口发送和接收的DMA通道句柄

串口句柄定义

```
1. typedef struct __UART_HandleTypeDef
2. {
3.     USART_TypeDef          *Instance;      // 串口寄存器的基地址定义
4.     UART_InitTypeDef       Init;           // 串口初始化数据类型
5.     uint8_t                *pTxBuffPtr;    // 串口发送缓冲区首地址
6.     uint16_t               TxXferSize;     // 串口待发送数据个数
7.     __IO uint16_t          TxXferCount;    // 串口发送数据计数器
8.     uint8_t                *pRxBuffPtr;    // 串口接收缓冲区首地址
9.     uint16_t               RxXferSize;     // 串口待接收数据个数
10.    __IO uint16_t           RxXferCount;    // 串口接收数据计数器
11.    DMA_HandleTypeDef        ;              // 串口发送的 DMA 通道句柄定义
12.    DMA_HandleTypeDef        *hdmartx;     // 串口接收的 DMA 通道句柄定义
13.    HAL_LockTypeDef          Lock;          // 保护锁类型定义
14.    __IO HAL_UART_StateTypeDef gState;      // 串口全局状态和发送状态信息
15.    __IO HAL_UART_StateTypeDef RxState;     // 串口接收状态信息
16.    __IO uint32_t            ErrorCode;     // 串口错误代码
17. } UART_HandleTypeDef;
```

串口工作状态

串口初始化数据类型

结构体类型，包括7个成员变量

```
1. typedef struct
2. {
3.     uint32_t BaudRate;           // 设置通信波特率
4.     uint32_t WordLength;         // 设置通信字符中数据位的位数
5.     uint32_t StopBits;           // 设置通信字符中停止位的位数
6.     uint32_t Parity;             // 设置奇偶校验模式
7.     uint32_t Mode;               // 设置接收或发送模式是否使能或禁能
8.     uint32_t HwFlowCtl;          // 设置硬件流控是否使能或禁能
9.     uint32_t OverSampling;       // 设置采样频率和信号传输频率的比例
10. } UART_InitTypeDef;
```


成员变量WordLength的取值范围

宏常量定义	含义
UART_WORDLENGTH_8B	数据位长度为8位
UART_WORDLENGTH_9B	数据位长度为9位

成员变量StopBits的取值范围

宏常量定义	含义
UART_STOPBITS_1	停止位长度为1位
UART_STOPBITS_2	停止位长度为2位

成员变量Parity的取值范围

宏常量定义	含义
UART_PARITY_NONE	无奇偶校验
UART_PARITY_EVEN	偶校验
UART_PARITY_ODD	奇校验

成员变量Mode的取值范围

宏常量定义	含义
UART_MODE_RX	串口仅处于接收模式，只能接收数据，不能发送数据
UART_MODE_TX	串口仅处于发送模式，只能发送数据，不能接收数据
UART_MODE_TX_RX	串口处于接收和发送模式，可以同时收发数据

成员变量HwFlowCtrl的取值范围

硬件流控可以控制数据传输的进程，防止数据丢失，该功能主要在收发双方传输速度不匹配的时候使用。

宏常量定义	
UART_HWCONTROL_NONE	无硬件流控
UART_HWCONTROL_RTS	使能“请求发送 (RTS)” 引脚
UART_HWCONTROL_CTS	使能“允许发送 (CTS)” 引脚
UART_HWCONTROL_RTS_CTS	使能“请求发送 (RTS)” 和 “允许发送 (CTS)” 引脚

成员变量OverSampling的取值范围

宏常量定义	含义
UART_OVERSAMPLING_16	采样频率是信号传输频率的16倍
UART_OVERSAMPLING_8	采样频率是信号传输频率的8倍

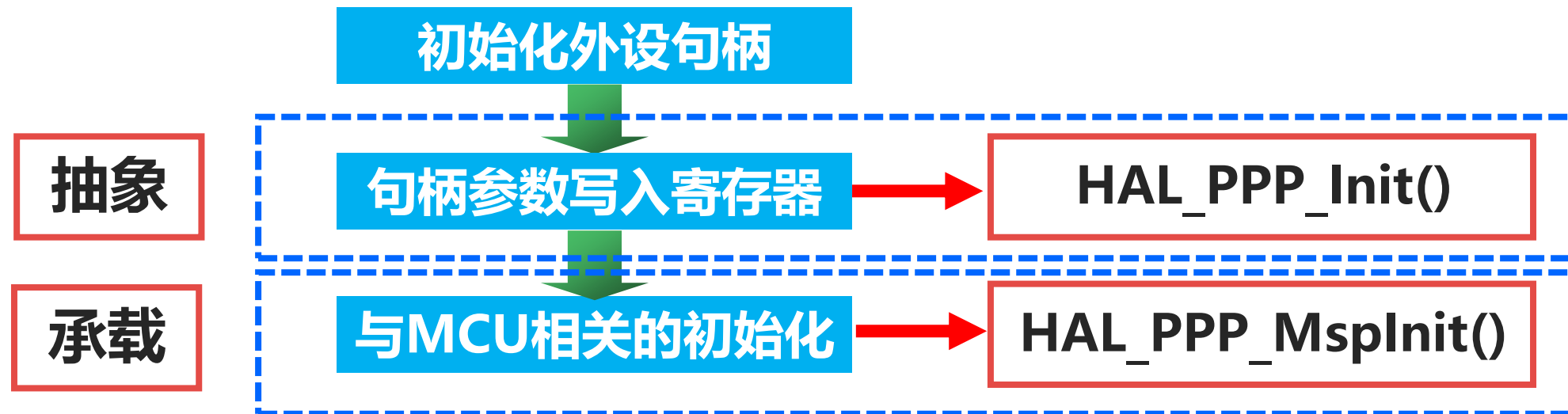


电子科技大学
University of Electronic Science and Technology of China

2 外设初始化设计思想

设计思想

从抽象到承载



抽象：指与MCU无关的参数，比如串口通信中的数据位数、波特率。这些参数属于通用配置，与使用什么样的MCU无关。

承载：指与具体MCU相关的初始化，将抽象的串口在具体的MCU上实现，完成时钟、引脚、DAM数据流和中断等底层硬件的初始化。

串口初始化过程

HAL_UART_Init() Start



将句柄结构中的初始化参数存入寄存器



HAL_UART_MspInit() Start



完成时钟、引脚等初始化



HAL_UART_MspInit() End



HAL_UART_Init() End



串口初始化函数

```
1. void MX_USART2_UART_Init (void)
2. {
3.     huart2.Instance          = USART2;           // 配置串口 2
4.     huart2.Init.BaudRate     = 115200;          // 波特率 115200
5.     huart2.Init.WordLength   = UART_WORDLENGTH_8B; // 数据位为 8 位
6.     huart2.Init.StopBits     = UART_STOPBITS_1;  // 停止位为 1 位
7.     huart2.Init.Parity       = UART_PARITY_NONE; // 无奇偶校验
8.     huart2.Init.Mode         = UART_MODE_TX_RX;  // 使能接收和发送模式
9.     huart2.Init.HwFlowCtl    = UART_HWCONTROL_NONE; // 无硬件流控
10.    huart2.Init.OverSampling = UART_OVERSAMPLING_16; // 16 倍过采样
11.    if (HAL_UART_Init(&huart2) != HAL_OK)
12.    {
13.        Error_Handler();
14.    }
15. }
```

将通信参数存入句柄结构

执行串口初始化操作

与MCU相关的初始化函数

```
1. void HAL_UART_MspInit(UART_HandleTypeDef* huart)
2. {
3.     GPIO_InitTypeDef GPIO_InitStructure = {0};    // 定义引脚初始化结构体，并赋值为0
4.     if(huart->Instance==USART2)                  // 判断串口号
5.     {
6.         __HAL_RCC_USART2_CLK_ENABLE();            // 使能串口2时钟
7.         __HAL_RCC_GPIOA_CLK_ENABLE();             // 使能端口GPIOA时钟
8.         // 串口引脚复用，设置PA2为USART2_TX引脚，PA3为USART2_RX引脚
9.         GPIO_InitStructure.Pin = GPIO_PIN_2|GPIO_PIN_3;
10.        GPIO_InitStructure.Mode = GPIO_MODE_AF_PP;
11.        GPIO_InitStructure.Pull = GPIO_PULLUP;
12.        GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
13.        GPIO_InitStructure.Alternate = GPIO_AF7_USART2;
14.        HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
15.    }
16. }
```

时钟初始化

引脚初始化



电子科技大学
University of Electronic Science and Technology of China

9.3 轮询方式的串口通信



电子科技大学
University of Electronic Science and Technology of China

1 轮询方式的接口函数

1 串口初始化函数：HAL_UART_Init

接口函数：HAL_UART_Init	
函数原型	HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef *huart)
功能描述	按照串口句柄中指定的参数初始化串口
入口参数	huart: 串口句柄的地址
返回值	HAL状态值：HAL_OK表示初始化成功，HAL_ERROR表示初始化失败
注意事项	<div><div>1. 该函数将调用与MCU相关的初始化函数HAL_UART_MspInit完成时钟、引脚和中断等底层硬件的初始化操作</div><div>2. 该函数由CubeMX自动生成</div></div>

接口函数：HAL_UART_Transmit	
函数原型	HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
功能描述	在轮询方式下发送一定数量的数据
入口参数1	huart: 串口句柄的地址
入口参数2	pData: 待发送数据的首地址
入口参数3	Size: 待发送数据的个数
入口参数4	Timeout: 超时等待时间，以ms为单位，HAL_MAX_DELAY表示无限等待
返回值	HAL状态值：HAL_OK表示发送成功；HAL_ERROR表示参数错误；HAL_BUSY表示串口被占用；HAL_TIMEOUT表示发送超时
注意事项	<div>1. 该函数连续发送数据，发送过程中通过判断TXE标志来发送下一个数据，通过判断TC标志来结束数据的发送</div> <div>2. 如果在等待时间内没有完成发送，则不再发送，返回超时标志</div> <div>3. 该函数由用户调用</div>

接口函数：HAL_UART_Receive

函数原型	HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
功能描述	在轮询方式下接收一定数量的数据
入口参数1	huart: 串口句柄的地址
入口参数2	pData: 存放接收数据的首地址
入口参数3	Size: 待接收数据的个数
入口参数4	Timeout: 超时等待时间，以ms为单位，HAL_MAX_DELAY表示无限等待
返回值	HAL状态值：HAL_OK表示接收成功；HAL_ERROR表示参数错误；HAL_BUSY表示串口被占用；HAL_TIMEOUT表示接收超时
注意事项	<ol style="list-style-type: none">1. 该函数连续接收数据，在接收过程中通过判断RXNE标志来接收新的数据2. 如果在超时时间内没有完成接收，则不再接收数据，返回超时标志3. 该函数由用户调用



电子科技大学
University of Electronic Science and Technology of China

2 基础任务：固定长度的数据收发

基础任务**固定长度的数据收发****01****任务目标**

掌握CubeMX软件配置串口实现轮询方式通信的方法。

02**任务内容**

从PC上发送5个字符到Nucleo开发板，Nucleo开发板收到后将字符原样发回到PC。

串口外设配置

Search:

Categories **A->Z**

Connectivity

- I2C1
- I2C2
- I2C3
- SDIO
- SPI1
- SPI2
- SPI3
- SPI4
- SPI5
- USART1
- ✓ USART2**
- USART6
- USB_OTG_FS

Multimedia >

USART2 Mode and Configuration

Mode

Mode: **Asynchronous**

Hardware Flow Control (RS232): **Disable**

Configuration

Reset Configuration

- ☒ DMA Settings
- ☒ GPIO Settings
- ☒ User Constants
- ☒ NVIC Settings
- ☒ Parameter Settings**

Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Trans
Over Sampling	16 Samples

异步模式，无硬件流控

设置通信参数：
波特率**115200**
8位数据位
无奇偶校验
1位停止位
使能接收和发送
16倍过采样

1. `/* USER CODE BEGIN PM */`

用户定义

2. `uint8_t RecBuf[10];`

用户变量定义

`// 接收缓冲区定义`

3. `/* USER CODE END PM */`

CubeMX生成

4. `UART_HandleTypeDef huart2;`

`// 串口 2 句柄定义`

1. `while (1)`

步骤六：程序编写

用户应用代码

2. `{`

3. `/* USER CODE BEGIN 3 */`

4. `// 接收 5 个字符完成`

5. `if(HAL_UART_Receive(&huart2, RecBuf, 5, 100) == HAL_OK)`

6. `{`

7. `HAL_UART_Transmit(&huart2, RecBuf, 5, 100);` `// 把接收的字符原样发回`

8. `}`

9. `}`

10. `/* USER CODE END 3 */`

串口调试助手

串口号:  COM15  

波特率:  115200 

数据位: 8 

校验位: None 

停止位: One 

 关闭串口



接收区设置.



☐ 接收并保存到文件

☐ 十六进制显示


☐ 暂停接收显示

☐ 自动断帧 ? 20



☐ 接收脚本  Add Timesta 

 保存数据  清空数据

发送区设置.


☐ 发送文件  扩展命令


☐ 十六进制发送

☐ 发送脚本  ADD8 

☐ 定时发送 1.0 秒

☐ DTR ☐ RTS

换行符  \r\n (CRLF)

☒ 显示发送字符串 


1

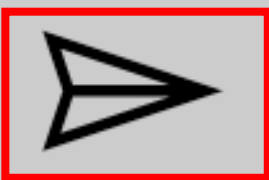
设置串口通信参数

2

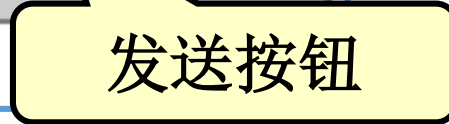
设置接收区和发送区

UESTC

 输入5个字符



发送: 10 接收: 10

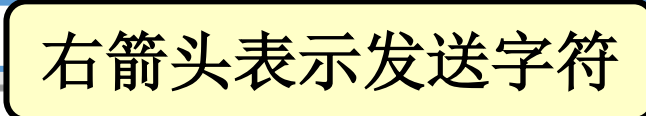
 发送按钮

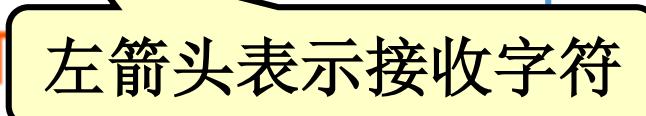
» UESTC

« UESTC

» UEST

« UESTC

 右箭头表示发送字符

 左箭头表示接收字符



电子科技大学
University of Electronic Science and Technology of China

3 进阶任务：实现串口重定向

进阶任务

实现串口重定向

01

任务目标

利用串口实现printf函数和scanf函数。

02

任务内容

在PC上利用串口调试助手发送数据到MCU，MCU调用scanf函数读取数据，然后调用printf函数发送应答信息到PC。

串口重定向设计思路

1. 在C语言中，printf函数是将数据格式化输出到屏幕，scanf函数是从键盘格式化输入数据；
2. 在嵌入式系统中，一般采用串口进行数据的输入和输出；
3. 重定向是指用户改写C语言的库函数，当链接器检查到用户编写了与C库函数同名的函数时，将优先使用用户编写的函数，从而实现对库函数的修改；
4. printf函数内部通过调用fputc函数来实现数据输出，scanf函数内部通过调用fgetc函数来实现数据输入，因此用户需要改写这两个函数实现串口重定向。

添加头文件

在main.h文件中添加标准输入输出头文件“stdio.h”

```
1.  /*----- main.h -----*/
2.  /* USER CODE BEGIN Includes */
3.  #include "stdio.h"
4.  /* USER CODE END Includes */
```

头文件添加位置

USER CODE BEGIN Includes
USER CODE END Includes

重定义fputc函数

```
1.  /* USER CODE BEGIN 4 */
2.  /*
3.   * @brief Retargets the C library printf function to the USART.
4.   * @param None
5.   * @retval None
6.   */
7.  int fputc(int ch, FILE *f)
8.  {
9.      // 采用轮询方式发送 1 字节数据，超时时间设置为无限等待
10.     HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
11.     return ch;
12. }
13. /* USER CODE END 4 */
```

代码添加位置

USER CODE BEGIN	4
USER CODE END	4

HAL_MAX_DELAY表示
超时时间为无限等待

重定义fgetc函数

```
1.  /* USER CODE BEGIN 4 */
2.  /*
3.   * @brief Retargets the C library scanf function to the UART.
4.   * @param None
5.   * @retval None
6.   */
7.  int fgetc(FILE *f)
8.  {
9.      uint8_t ch;
10.     // 采用轮询方式接收 1 字节数据，超时时间设置为无限等待
11.     HAL_UART_Receive( &huart2, (uint8_t *)&ch, 1, HAL_MAX_DELAY );
12.     return ch;
13. }
14. /* USER CODE END 4 */
```

代码添加位置

USER CODE BEGIN	4
USER CODE END	4

```
1.  /* USER CODE BEGIN PV */
```

用户变量定义

```
2.  uint8_t RecData;
```

// 存放接收数据

```
3.  /* USER CODE END PV */
```

```
1.  /* USER CODE BEGIN 2 */
```

用户初始化代码

```
2.  printf ("UART Retarget:\r\n");
```

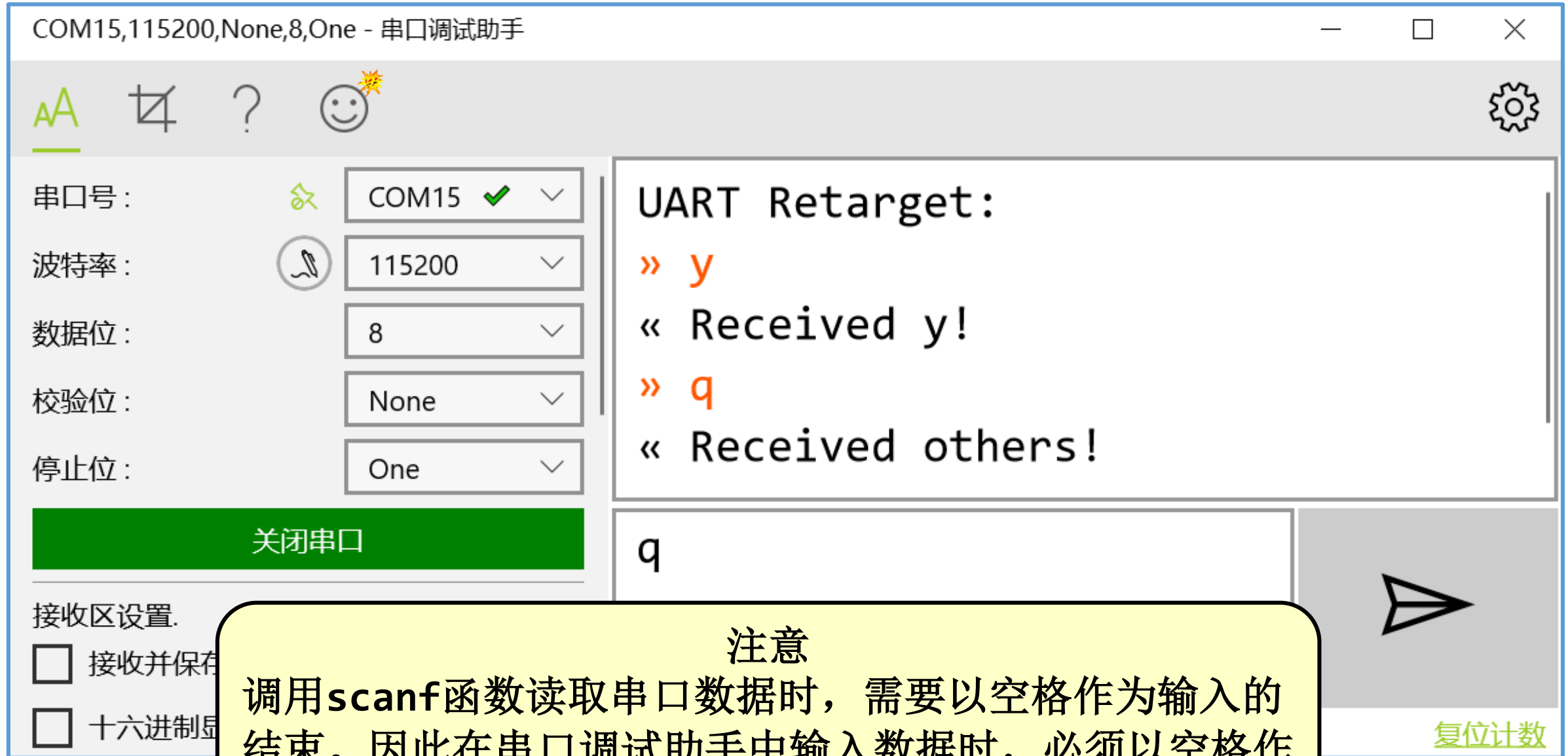
// 发送提示信息

```
3.  /* USER CODE END 2 */
```

用户应用代码

```
1. while (1)
2. {
3.     /* USER CODE BEGIN 3 */
4.     if( scanf("%c",&RecData) == 1)           // 正确接收到数据
5.     {
6.         if( RecData == 'y' )
7.         {
8.             printf ("Received y!\r\n");       // 发送回复信息
9.         }
10.        else
11.        {
12.            printf ("Received others!\r\n");   // 发送回复信息
13.        }
14.    }
15. }
16.     /* USER CODE END 3 */
```

串口调试助手



注意

调用scanf函数读取串口数据时，需要以空格作为输入的结束。因此在串口调试助手中输入数据时，必须以空格作为结束，然后再点击发送按钮，否则无法正确接收数据。



电子科技大学
University of Electronic Science and Technology of China

9.4 中断方式的串口通信

1 中断方式的接口函数

串口中断方式的特点

1. 发送数据时，将一字节数据放入数据寄存器DR；接收数据时，将DR的内容存放到用户存储区；
2. 中断方式不必等待数据的传输过程，只需要在每字节数据收发完成后，由中断标志位触发中断，在中断服务程序中放入新的一字节数据或者读取接收到的一字节数据；
3. 在传输数据量较大，且通信波特率较高（大于38400）时，如果采用中断方式，每收发一个字节的数据，CPU都会被打断，造成CPU无法处理其他事务。因此在批量数据传输，通信波特率较高时，建议采用DMA方式。

1 串口中断方式发送函数：HAL_UART_Transmit_IT

函数原型	HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)	
功能描述	在中断方式下发送一定数量的数据	<div>发送过程</div> <div>每发送一个数据进入一次中断，在中断中根据发送数据的个数来判断数据是否发送完成</div>
入口参数1	huart：串口句柄的地址	
入口参数2	pData：待发送数据的首地址	
入口参数3	Size：待发送数据的个数	
返回值	HAL状态值：HAL_OK表示发送成功；HAL_ERROR表示参数错误；HAL_BUSY表示串口被占用	
注意事项	1. 函数将使能串口发送中断 2. 函数将置位TXEIE和TCIE，使能发送数据寄存器空中断和发送完成中断。完成指定数量的数据发送后，将会关闭发送中断，即清零TXEIE和TCIE。因此用户采用中断方式连续发送数据时，需要重复调用该函数，以便重新开启发送中断 3. 当指定数量的数据发送完成后，将调用发送中断回调函数 HAL_UART_TxCpltCallback进行后续处理 4. 该函数由用户调用。	

2 串口中断方式接收函数：HAL_UART_Receive_IT		
函数原型	HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)	
功能描述	在中断方式下接收一定数量的数据	<div>接收过程</div> <p>每接收一个数据进入一次中断，在中断中根据接收数据的个数来判断数据是否接收完成</p>
入口参数1	huart：串口句柄的地址	
入口参数2	pData：存放接收数据的首地址	
入口参数3	Size：待接收数据的个数	
返回值	HAL状态值：HAL_OK表示接收成功；HAL_ERROR表示参数错误；HAL_BUSY表示串口被占用	
注意事项	1. 函数将使能串口接收中断 2. 函数将置位RXNEIE，使能接收数据寄存器非空中断RXNE。完成指定数量的数据接收后，将会关闭接收中断，即清零RXNEIE。因此用户采用中断方式连续接收数据时，要重复调用该函数，以重新开启接收中断 3. 当指定数量的数据接收完成后，将调用接收中断回调函数HAL_UART_RxCpltCallback进行后续处理 4. 该函数由用户调用	

3 串口中断通用处理函数：HAL_UART_IRQHandler

接口函数：HAL_UART_IRQHandler	
函数原型	void HAL_UART_IRQHandler(UART_HandleTypeDef *huart)
功能描述	作为所有串口中断发生后的通用处理函数
入口参数	huart: 串口句柄的地址
返回值	无
注意事项	<ol style="list-style-type: none">1. 函数内部先判断中断类型，并清除对应的中断标志，最后调用回调函数完成对应的中断处理2. 该函数由CubeMX自动生成

4 串口发送中断回调函数：HAL_UART_TxCpltCallback

接口函数： HAL_UART_TxCpltCallback	
函数原型	void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
功能描述	回调函数，用于处理所有串口的发送中断，用户在该函数内编写实际的任务处理程序
入口参数	huart：串口句柄的地址
返回值	无
注意事项	<div>1. 函数由串口中断通用处理函数HAL_UART_IRQHandler调用，完成所有串口的发送中断任务处理</div> <div>2. 函数内部需要根据串口句柄的实例来判断是哪一个串口产生的发送中断</div> <div>3. 函数由用户根据具体的处理任务编写</div>

5 串口接收中断回调函数：HAL_UART_RxCpltCallback

接口函数： HAL_UART_RxCpltCallback	
函数原型	void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
功能描述	回调函数，用于处理所有串口的接收中断，用户在该函数内编写实际的任务处理程序
入口参数	huart：串口句柄的地址
返回值	无
注意事项	<div>1. 函数由串口中断通用处理函数HAL_UART_IRQHandler调用，完成所有串口的接收中断任务处理</div> <div>2. 函数内部需要根据串口句柄的实例来判断是哪一个串口产生的接收中断</div> <div>3. 函数由用户根据具体的处理任务编写</div>

6 串口中断使能函数：__HAL_UART_ENABLE_IT

接口函数：__HAL_UART_ENABLE_IT

函数原型	__HAL_UART_ENABLE_IT(__HANDLE__, __INTERRUPT__)
功能描述	使能对应的串口中断类型
参数1	__HANDLE__：串口句柄的地址
参数2	<p>__INTERRUPT__：串口中断类型，该参数几个常用的取值如下：</p> <p>UART_IT_TXE：发送数据寄存器空中断</p> <p>UART_IT_TC：发送完成中断</p> <p>UART_IT_RXNE：接收数据寄存器非空中断</p> <p>UART_IT_IDLE：线路空闲中断</p>
返回值	无
注意事项	<ol style="list-style-type: none">1. 该函数是宏函数，进行宏替换，不发生函数调用2. 函数需要由用户调用，用于使能对应的串口中断类型

7 串口中断标志查询函数：__HAL_UART_GET_FLAG

接口函数：__HAL_UART_GET_FLAG	
函数原型	__HAL_UART_GET_FLAG (__HANDLE__, __INTERRUPT__)
功能描述	查询对应的串口中断标志
参数1	__HANDLE__：串口句柄的地址
参数2	__INTERRUPT__：串口中断类型，该参数几个常用的取值如下： UART_IT_TXE：发送数据寄存器空中断 UART_IT_TC：发送完成中断 UART_IT_RXNE：接收数据寄存器非空中断 UART_IT_IDLE：线路空闲中断
返回值	中断标志的状态值：SET表示中断标志置位；RESET表示中断标志没有置位
注意事项	1. 该函数是宏函数，进行宏替换，不发生函数调用 2. 函数需要由用户调用，用于查询对应的串口中断标志

8 空闲中断标志清除函数：__HAL_UART_CLEAR_IDLEFLAG

接口函数：__HAL_UART_CLEAR_IDLEFLAG	
函数原型	__HAL_UART_CLEAR_IDLEFLAG (__HANDLE__)
功能描述	清除串口的空闲中断标志
参数	__HANDLE__：串口句柄的地址
返回值	无
注意事项	<ol style="list-style-type: none">1. 该函数是宏函数，进行宏替换，不发生函数调用2. 函数需要由用户调用，用于清除对应的串口空闲中断标志



电子科技大学
University of Electronic Science and Technology of China

2 基础任务：固定长度的数据收发

基本任务**固定长度的数据收发****01****任务目标**

掌握CubeMX软件配置串口实现中断方式通信的方法。

02**任务内容**

利用串口调试助手，从PC上发送10个字符到Nucleo开发板，Nucleo开发板收到后原样发回到PC。

前后台编程模式

1. 采用前后台编程模式;
2. 前台程序为中断服务程序，一旦数据接收完成，则设置一个标志位为1;
3. 后台程序为while(1)的死循环，在循环中不断检测标志位是否为1。如果为1，表明数据接收完成，并存放在接收缓冲区中。然后进行后续处理：先清除标志位，再把接收的数据原样发回。

串口外设配置

Search:

Categories **A->Z**

Connectivity

- I2C1
- I2C2
- I2C3
- SDIO
- SPI1
- SPI2
- SPI3
- SPI4
- SPI5
- USART1
- ✓ USART2**
- USART6
- USB_OTG_FS

Multimedia >

USART2 Mode and Configuration

Mode

Mode: **Asynchronous**

Hardware Flow Control (RS232): **Disable**

Configuration

Reset Configuration

- ☒ DMA Settings
- ☒ GPIO Settings
- ☒ User Constants
- ☒ NVIC Settings
- ☒ Parameter Settings**

Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Trans
Over Sampling	16 Samples

异步模式，无硬件流控

设置通信参数：
波特率**115200**
8位数据位
无奇偶校验
1位停止位
使能接收和发送
16倍过采样

使能串口中断

✓ NVIC Settings	✓ DMA Settings	✓ GPIO Settings	
✓ Parameter Settings	✓ User Constants		
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART2 global interrupt	✓	0	0

使能串口2的中断
中断优先级使用默认值

步骤六：程序编写

```
1.  /* USER CODE BEGIN PD */
```

```
2.  #define LENGTH 10
```

// 接收缓冲区的大小

```
3.  /* USER CODE END PD */
```

便于修改缓冲区的大小

```
4.  /* USER CODE BEGIN PV */
```

```
5.  uint8_t RxBuffer[LENGTH];
```

// 接收缓冲区

```
6.  uint8_t RxFlag = 0;
```

// 接收完成标志：0 表示接收未完成，1 表示接收完成

```
7.  /* USER CODE END PV */
```

用户宏定义
及变量定义

```
1.  /* USER CODE BEGIN 2 */
```

```
2.  printf("***** UART commucition using IT *****\r\n");
```

// 发送提示信息

```
3.  printf("Please enter 10 characters:\r\n");
```

```
4.  HAL_UART_Receive_IT(&huart2, (uint8_t*)RxBuffer, LENGTH);
```

// 使能接收中断

```
5.  /* USER CODE END 2 */
```

用户初始化代码

步骤六：程序编写

用户应用代码



```
1. while (1)
2. {
3.     /* USER CODE BEGIN 3 */
4.     if( RxFlag == 1)           // 判断数据是否接收完成
5.     {
6.         RxFlag = 0;           // 清除标志位
7.         printf("Recevie Success!\r\n");
8.         // 将接收的字符原样发回
9.         HAL_UART_Transmit_IT(&huart2, (uint8_t*)RxBuffer, LENGTH);
10.    }
11. }
12.     /* USER CODE END 3 */
```


接收中断回调函数



```
1.  /* USER CODE BEGIN 4 */
2.  /*
3.   * @brief  UART receive interrupt call back function
4.   * @param  None
5.   * @retval None
6.   */
7.  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
8.  {
9.      if(huart ->Instance == USART2)    // 判断发生接收中断的串口
10.     {
11.         RxFlag = 1;                    // 置位接收完成标志
12.         HAL_UART_Receive_IT(&huart2, (uint8_t*)RxBuffer, LENGTH); // 使能接收中断
13.     }
14. }
15. /* USER CODE END 4 */
```

串口调试助手


COM15,115200,None,8,One - 串口调试助手



串口号:

 COM15 

波特率:

 115200

数据位:

8

校验位:

None

停止位:

One

关闭串口

接收区设置.

☐ 接收并保存到文件

☐ 十六进制显示

☐ 暂停接收显示

***** UART commucition using IT *****

Please enter 10 characters:

» UESTCQIANG


« Recevie Success!

UESTCQIANG

UESTCQIANG

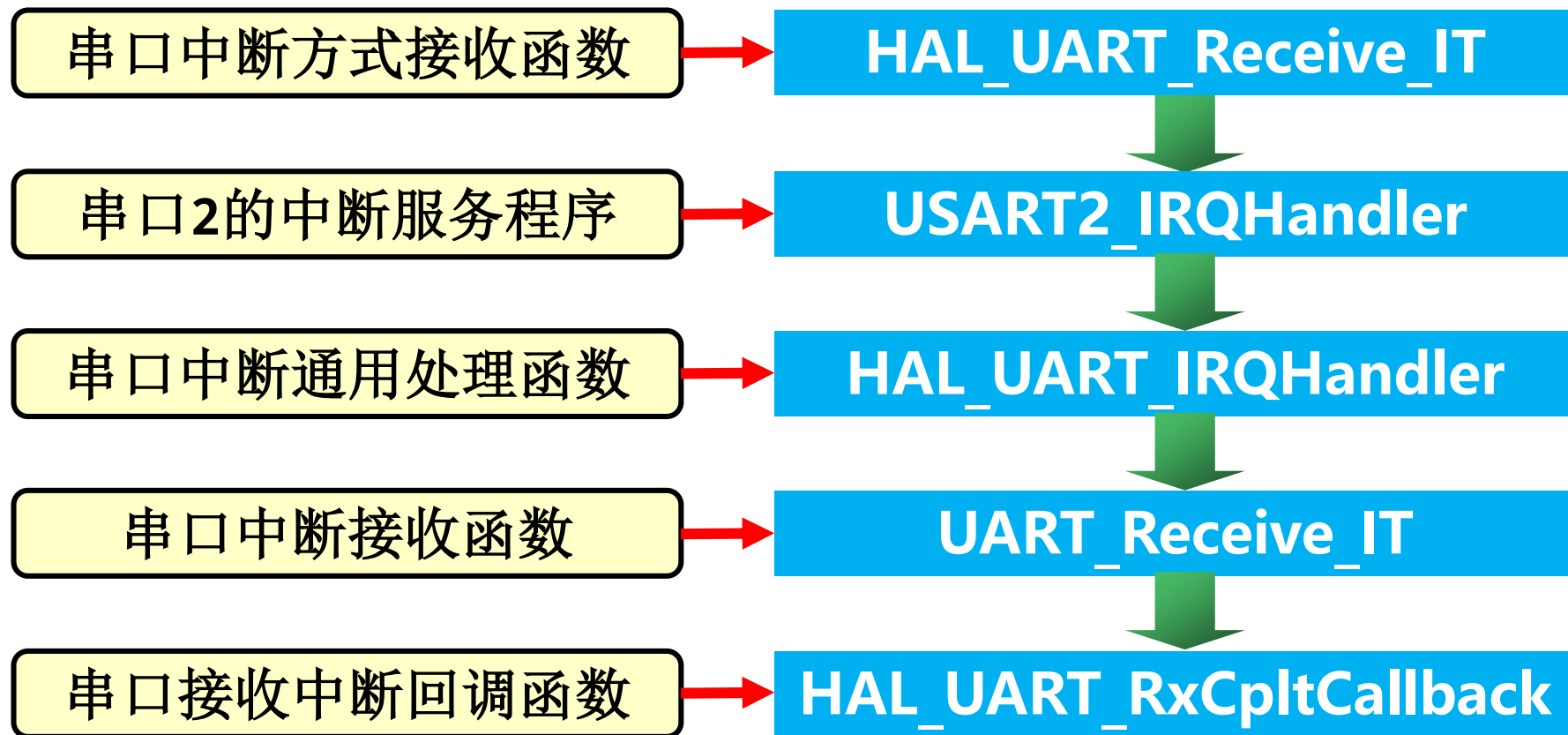
发送: 10

接收: 100

 [复位计数](#)

串口中断过程

串口中断处理过程



3 进阶任务：实现简单的帧格式通信

帧格式的概念

1. 帧 (Frame) 是数据传输的一种单位。一帧数据由多个字符组合而成，不同字段的字符代表不同的含义，执行不同的功能；
2. 在实际的工程应用中，数据的传输常常以帧为单位来进行，如工控领域中最常用的Modbus通信协议中的消息帧；
3. 发送方按照规定的帧格式发送一帧数据，接收方接收下这一帧数据后，再按照帧格式进行解析，最后完成后续的处理。

Modbus消息帧格式

起始符	设备地址	功能代码	数据	校验	结束符
1个字符	2个字符	1个字符	n个字符	2个字符	1个字符

起始符 : 表示一帧数据的开始

设备地址 : 用于指定需要进行信息传递的设备

功能代码 : 用于指定需要完成的操作

数据 : 表示需要传输的数据

校验 : 用于通信中的错误校验

结束符 : 表示一帧数据的结束

自定义的帧格式设定

帧头	设备码	功能码	帧尾
0xaa	1个字符 (8bit)	1个字符(8bit)	0x55

帧头 : 0xaa表示一帧数据的开始

设备码: 0x01表示指示灯

功能码: 0x00表示关闭指示灯, 0x01表示开启指示灯

帧尾 : 0x55表示一帧数据的结束

进阶任务

实现简单的帧格式通信

01

任务目标

实现简单的帧格式通信。

02

任务内容

PC按照自定义的帧格式发送指令开启或关闭Nucleo开发板上的LD2。

步骤六：程序编写

```
1.  /* USER CODE BEGIN PV */
2.  uint8_t RxBuffer[4];           // 接收缓冲区
3.  uint8_t RxFlag = 0;           // 接收完成标志：为 0 表示接收未完成，为 1 表示接收完成
4.  uint8_t ErrFlag = 0;          // 指令错误标志：为 0 表示指令正确，为 1 表示指令错误
5.  /* USER CODE END PV */
6.  /*-----*/
7.  /* USER CODE BEGIN 2 */
8.  // 发送提示信息
9.  printf("***** Communication Frame *****\r\n");
10. printf("Please enter instruction:\r\n");
11. printf("Head->0xaa Device->0x01 Operation->0x00/0x01 Tail->0x55.\r\n");
12. HAL_UART_Receive_IT(&huart2, (uint8_t*)RxBuffer, 4); // 使能接收中断
13. /* USER CODE END 2 */
```

添加用户变量定义

添加用户初始化代码

添加用户代码

```
1. while (1)
2. {
3.     /* USER CODE BEGIN 3 */
4.     if( RxFlag == 1) // 判断数据是否接收完成
5.     {
6.         RxFlag = 0; // 清除标志位
7.         // 帧格式解析
8.         if( RxBuffer[0] == 0xaa && RxBuffer[3] == 0x55) // 判断帧头帧尾
9.         {
10.            if( RxBuffer[1] == 0x01 ) // 判断设备码
11.            {
12.                if(RxBuffer[2] == 0x00 ) // 判断功能码
13.                {
14.                    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
15.                    printf("LD2 is close!\r\n");
16.                }
```

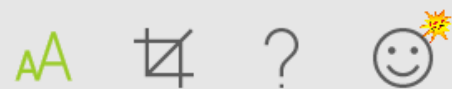
```
17.         else if(RxBuffer[2] == 0x01 )           // 判断功能码
18.         {
19.             HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
20.             printf("LD2 is open!\r\n");
21.         }
22.         else                                     // 功能码错误
23.         {
24.             ErrFlag = 1;                         // 置位错误标志
25.         }
26.     }
27.     else                                         // 设备码错误
28.     {
29.         ErrFlag = 1;                             // 置位错误标志
30.     }
31. }
32. else                                           // 帧头帧尾错误
33. {
34.     ErrFlag = 1;                                 // 置位错误标志
35. }
```

```
36.      if( ErrFlag == 1 )                                // 发送错误提示信息
37.      {
38.          printf("Communication Error! Please send again!\r\n");
39.      }
40.      // 清除接收缓冲区和错误标志, 准备下一次接收
41.      ErrFlag      = 0;
42.      RxBuffer[0] = 0;
43.      RxBuffer[1] = 0;
44.      RxBuffer[2] = 0;
45.      RxBuffer[3] = 0;
46.  }
47. }
48. /* USER CODE END 3 */
```

接收中断回调函数

```
1.  /* USER CODE BEGIN 4 */
2.  /*
3.   * @brief  UART receive interrupt call back function
4.   * @param  None
5.   * @retval None
6.   */
7.  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
8.  {
9.      if(huart ->Instance == USART2)    // 判断发生接收中断的串口
10.     {
11.         RxFlag = 1;                    // 置位接收完成标志
12.         HAL_UART_Receive_IT(&huart2, (uint8_t*)RxBuffer, LENGTH); // 使能接收中断
13.     }
14. }
15. /* USER CODE END 4 */
```

串口调试助手



接收区设置.

☐ 接收并保存到文件☐ 十六进制显示☐ 暂停接收显示☐ 自动断帧☐ 接收脚本

Add TimeSta

保存数据

清空数据

发送区设置.

☐ 发送文件

扩展命令

☒ 十六进制发送☐ 发送脚本

ADD8

☐ 定时发送

1.0

秒

☐ DTR☐ RTS

换行符

\r\n (CRLF)

☒ 显示发送字符串

***** Communication Frame *****

Please enter instruction:

Head->0xaa Device->0x01

Operation->0x00/0x01 Head->0x55.

» aa 01 01 55

« LD2 is open!

» aa 01 00 55

« LD2 is close!

» aa 01 02 55

« Communication Error! Please send again!

aa 01 02 55



发送: 12

接收: 195

复位计数



电子科技大学
University of Electronic Science and Technology of China

9.5 DMA方式的串口通信

1 DMA概述

DMA的概念

直接存储器访问 (DMA)：用于在外设与存储器之间以及存储器与存储器之间进行高速数据传输。DMA传输过程的初始化和启动由CPU完成，传输过程由DMA控制器来执行，无需CPU参与，从而节省CPU资源，提高利用率。

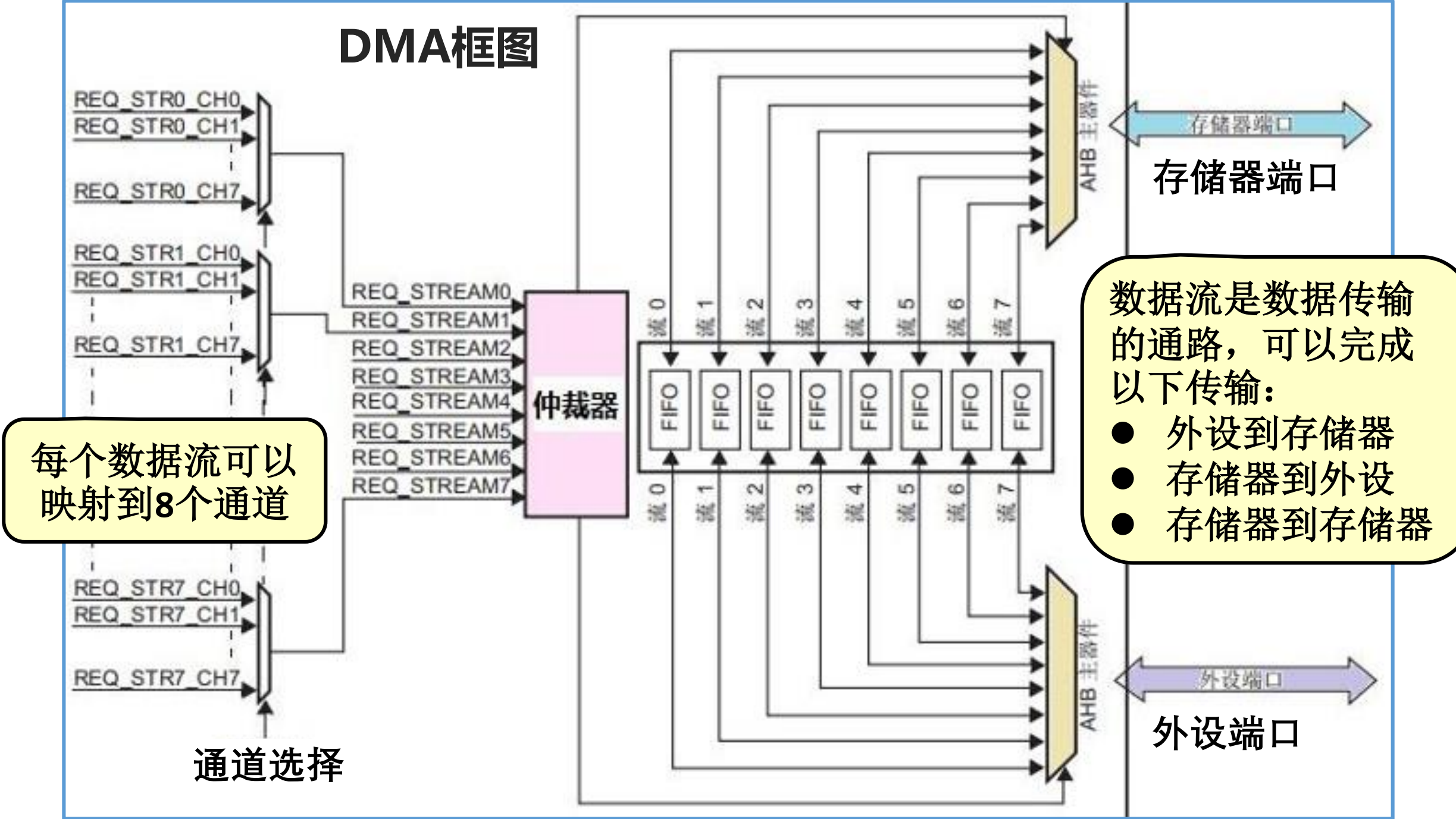
DMA数据传输的四个要素：

- ① 传输源：DMA数据传输的来源
- ② 传输目标：DMA数据传输的目的
- ③ 传输数量：DMA传输数据的数量
- ④ 触发信号：启动一次DMA数据传输的动作

STM32的DMA控制器特点

1. STM32F411微控制器具备两个DMA控制器：DMA1和DMA2，每个控制器有8个数据流，每个数据流可以映射到8个通道（或请求）；
2. 每一个DMA控制器用于管理一个或多个外设的存储器访问请求，并通过总线仲裁器来协调各个DMA请求的优先级；
3. 数据流（stream）是用于连接传输源和传输目标的数据通路，每个数据流可以配置为不同的传输源和传输目标，这些传输源和传输目标称为通道（Channel）；
4. 具备16字节的FIFO。使能FIFO功能后，源数据先送入FIFO，达到FIFO的触发阈值后，再传送到目标地址。

DMA框图



DMA数据传输方式

普通模式

传输结束后（即要传输数据的数量达到零），将不再产生DMA操作。若开始新的DMA传输，需在关闭DMA通道情况下，重新启动DMA传输。

循环模式

可用于处理环形缓冲区和连续数据流（例如ADC扫描模式）。当激活循环模式后，每轮传输结束时，要传输的数据数量将自动用设置的初始值进行加载，并继续响应DMA请求。



电子科技大学
University of Electronic Science and Technology of China

2 DMA方式的接口函数

1 串口DMA方式发送函数：HAL_UART_Transmit_DMA	
函数原型	HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)
功能描述	在DMA方式下发送一定数量的数据
入口参数1	huart: 串口句柄的地址
入口参数2	pData: 待发送数据的首地址
入口参数3	Size: 待发送数据的个数
返回值	HAL状态值：HAL_OK表示发送成功；HAL_ERROR表示参数错误； HAL_BUSY表示串口被占用
注意事项	1. 该函数将启动DMA方式的串口数据发送 2. 完成指定数量的数据发送后，可以触发DMA中断，在中断中将调用发送中断回调函数HAL_UART_TxCpltCallback进行后续处理 3. 该函数由用户调用

2 串口DMA方式接收函数：HAL_UART_Receive_DMA

函数原型	HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)
功能描述	在DMA方式下接收一定数量的数据
入口参数1	huart: 串口句柄的地址
入口参数2	pData: 待接收数据的首地址
入口参数3	Size: 待接收数据的个数
返回值	HAL状态值: HAL_OK表示接收成功; HAL_ERROR表示参数错误; HAL_BUSY表示串口被占用
注意事项	1. 该函数将启动DMA方式的串口数据接收 2. 完成指定数量的数据接收后, 可以触发DMA中断, 在中断中将调用接收中断回调函数HAL_UART_RxCpltCallback进行后续处理 3. 该函数由用户调用

3 获取未传输数据个数函数：__HAL_DMA_GET_COUNTER

接口函数：__HAL_DMA_GET_COUNTER	
函数原型	__HAL_DMA_GET_COUNTER(__HANDLE__)
功能描述	获取DMA数据流中未传输数据的个数
参数	__HANDLE__：串口句柄的地址
返回值	NDTR寄存器的内容，即DMA数据流中无传输数据的个数
注意事项	1. 该函数是宏函数，进行宏替换，不发生函数调用 2. 该函数需要由用户调用，用于获取未传输数据的个数

4 关闭DMA数据流：__HAL_DMA_DISABLE

接口函数：__HAL_DMA_DISABLE	
函数原型	__HAL_DMA_DISABLE(__HANDLE__)
功能描述	关闭指定的DMA数据流
参数	__HANDLE__：串口句柄的地址
返回值	无
注意事项	<ol style="list-style-type: none">1. 该函数是宏函数，进行宏替换，不发生函数调用2. 该函数需要由用户调用，用于关闭指定的DMA数据流3. 关闭DMA数据流后触发DMA中断，最终调用串口收发的回调函数



电子科技大学
University of Electronic Science and Technology of China

3 挑战任务：不定长数据的收发

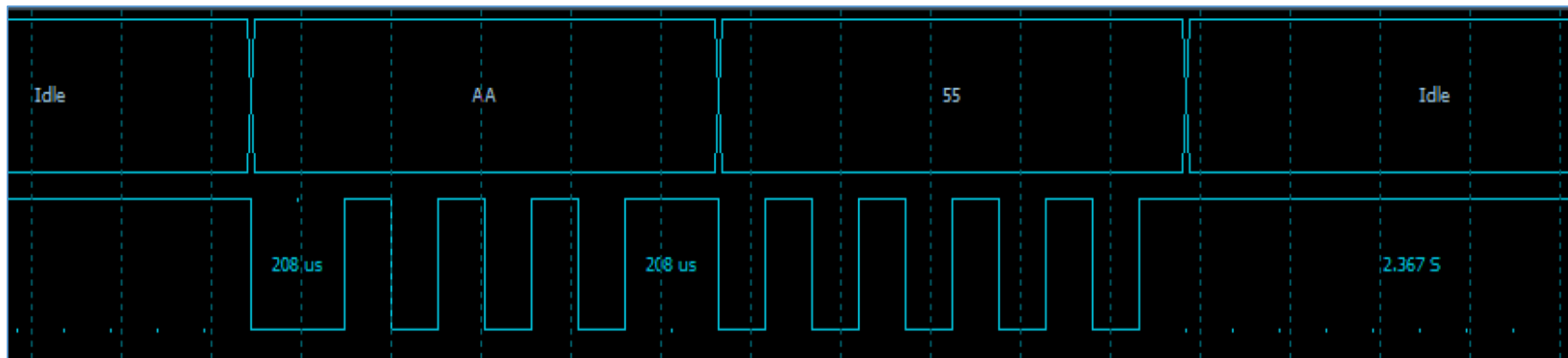
空闲中断的特点

1. 在一帧数据传输结束后，通信线路将会维持高电平，这个状态称为空闲状态；
2. 当CPU检测到通信线路处于空闲状态时，且空闲状态持续时间大于一个字节传输时间时，空闲状态标志IDLE将由硬件置1。如果串口控制寄存器CR1中的IDLEIE位为1，将会触发空闲中断（IDLE中断）；
3. 由于空闲标志是在一帧数据传输完成后才置位，在有效数据传输过程中不会置位，因此借助空闲中断，可以实现不定长数据的收发。

时序图

空闲中断时序图

假设一帧数据由2个字符构成，分别是0xaa、0x55，传输的时序图如下：



从图中可以看到，在发送该帧数据之前和之后，数据线为高电平，处于IDLE状态。在该帧有效数据发送过程中，即0xaa字节与0x55字节之间，没有IDLE状态出现，这样利用IDLE状态就可以判断不定长数据的发送结束。

挑战任务**不定长数据的收发****01****任务目标**

实现不定长数据的收发

02**任务内容**

利用串口调试助手，从PC上发送任意长度的字符到Nucleo开发板，Nucleo开发板收到后原样发回到PC。

设计思路

设计思路

1. 使能IDLE中断，在串口2的中断服务程序USART2_IRQHandler中添加对IDLE中断的判断，该函数位于stm32f4xx_it.c文件；
2. 设置传输模式为普通模式，启动DMA传输。串口一旦接收到数据，则触发DMA操作，将数据存放到用户定义的接收缓冲区；
3. 当一帧数据发送完成后，线路处于IDLE状态，将触发IDLE中断，调用IDLE中断回调函数，设置数据接收完成标志；
4. 主程序检测到接收完成标志置位后，将接收的一帧数据原样发回到PC，并禁能DMA，以触发DMA中断。DMA中断将调用接收中断回调函数，在回调函数中重新启动DMA传输。

Parameter Settings User Constants NVIC Settings **DMA Settings** GPIO Settings

DMA Request	Stream	Direction	Priority
Select			
Select			
USART2_RX			
USART2_TX			

分别选择RX和TX进行设置

2

点击两次Add按钮，分别添加串口接收和发送的DMA数据流

1

串口2的DMA配置

Add

Delete

设置串口接收的DMA参数

USART2_RX	DMA1 Stream 5	Peripheral To Memory	Low
USART2_TX	DMA1 Stream 6	Memory To Peripheral	Low

AddDelete

DMA Request Settings

ModeNormal

Increment Address☐

Use Fifo☐

Threshold

Data WidthByte

Burst Size

Memory☒

Byte

Byte

设置DMA请求参数

传输模式：普通模式

地址递增：外设不递增，存储器递增

FIFO使用：不使能

数据宽度：字节

DMA数据流、传输方向

和优先级自动配置

设置串口发送的DMA参数

USART2_RX	DMA1 Stream 5	Peripheral To Memory	Low
USART2_TX	DMA1 Stream 6	Memory To Peripheral	Low

AddDelete

串口发送的DMA参数与串口接收的DMA参数一致

DMA Request Settings

		Peripheral		Memory	
Mode	Normal	Increment Address	<input type="checkbox"/>		<input checked="" type="checkbox"/>
Use Fifo	<input type="checkbox"/>	Threshold		Data Width	Byte
				Burst Size	

使能串口中断及DMA中断

DMA数据流的中断使能
由CubeMX自动勾选

Parameter Settings	User Constants	NVIC Settings	DMA Settings	GPIO Settings
NVIC Interrupt Table		Enabled	Preemption Priority	Sub Priority
DMA1 stream5 global interrupt		<input checked="" type="checkbox"/>	0	0
DMA1 stream6 global interrupt		<input checked="" type="checkbox"/>	0	0
USART2 global interrupt		<input checked="" type="checkbox"/>	0	0

使能串口2的中断
中断优先级使用默认值

步骤六：程序编写

添加空闲中断的处理

```
1. void USART2_IRQHandler(void)
2. {
3.     /* USER CODE BEGIN USART2_IRQn 0 */
4.     /* USER CODE END USART2_IRQn 0 */
5.     HAL_UART_IRQHandler(&huart2);
6.     /* USER CODE BEGIN USART2_IRQn 1 */
7.     // 添加 IDLE 中断处理
8.     if(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_IDLE) != RESET) // 是否发生 IDLE
9.     {
10.         __HAL_UART_CLEAR_IDLEFLAG(&huart2); // 清除 IDLE 中断标志
11.         HAL_UART_IdleCpltCallback(&huart2); // 用户编写的 IDLE 中断回调函数
12.     }
13.     /* USER CODE END USART2_IRQn 1 */
14. }
```

该函数位于中断服务程序
stm32f4xx_it.c文件中

```

1.  /* USER CODE BEGIN PM */
2.  #define    LENGTH    100    // 接收缓冲区大小, 该值需要大于一帧数据的总字符数
3.  /* USER CODE END PM */
4.  /* USER CODE BEGIN PV */
5.  uint8_t RxBuffer[LENGTH]; // 接收缓冲区
6.  uint8_t RecCount = 0;      // 接收数据个数
7.  uint8_t RxFlag    = 0;     // 接收完成标志: 0 表示接收未完成, 1 表示接收完成
8.  /* USER CODE END PV */
9.  /*-----*/
10. /* USER CODE BEGIN 2 */
11. // 发送提示信息
12. printf("*****  UART communication using IDLE IT + DMA  *****\r\n");
13. printf("Please enter arbitrary length characters:\r\n");
14. _HAL_UART_ENABLE_IT(&huart2, UART_IT_IDLE); // 使能 IDLE 中断
15. HAL_UART_Receive_DMA(&huart2, (uint8_t*)RxBuffer, LENGTH); // 启动 DMA 接收
16. /* USER CODE END 2 */

```

该值需大于一帧数据的总字符数

添加用户宏定义及变量定义

添加用户初始化代码

用户应用代码

```
1.  /* USER CODE BEGIN 3 */
2.  if( RxFlag == 1)      // 判断数据是否接收完成
3.  {
4.      RxFlag = 0;      // 清除标志位
5.      // 发生空闲中断时, 已接收数据个数等于数据总量减去 DMA 数据流中待接收的数据个数
6.      RecCount = LENGTH - __HAL_DMA_GET_COUNTER(&hdma_usart2_rx);
7.      // 采用 DMA 方式将接收的数据原样发回到 PC
8.      HAL_UART_Transmit_DMA(&huart2, (uint8_t*)RxBuffer, RecCount);
9.      RecCount = 0;
10. // 设置 DMA Disable, 触发 DMA 中断, 调用接收中断回调函数来重新启动下一次 DMA 接收
11. __HAL_DMA_DISABLE(&hdma_usart2_rx);
12. }
13. }
14. /* USER CODE END 3 */
```

```
1.  /* USER CODE BEGIN 4 */
```

```
2.  /*----- 接收中断回调函数 -----
```

接收中断回调函数

```
3.  // DMA 中断将调用接收中断回调函数
```

```
4.  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
```

```
5.  {
```

```
6.      if(huart ->Instance == USART2)
```

```
7.      {
```

```
8.          // 重新启动 DMA 接收，准备下一次的数据传输
```

```
9.          HAL_UART_Receive_DMA(&huart2, (uint8_t*)RxBuffer, LENGTH);
```

```
10.     }
```

```
11. }
```

```
12. /*----- 空闲中断回调函数 -----
```

空闲中断回调函数

```
13. void HAL_UART_IdleCpltCallback(UART_HandleTypeDef *huart)
```

```
14. {
```

```
15.     RxFlag = 1;           // 设置接收完成标志
```

```
16. }
```

```
17. /* USER CODE END 4 */
```

函数声明

添加函数声明

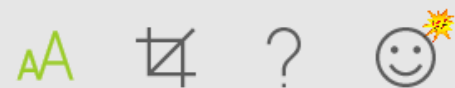
在main.h文件中添加对空闲中断回调函数的声明

```
1.  /* USER CODE BEGIN EFP */  
2.  // 添加用户自定义的空闲中断回调函数声明  
3.  void HAL_UART_IdleCpltCallback(UART_HandleTypeDef *huart);  
4.  /* USER CODE END EFP */
```

函数声明添加位置

```
USER CODE BEGIN EFP  
USER CODE END EFP
```

串口调试助手



接收区设置.

☐ 接收并保存到文件☐ 十六进制显示☐ 暂停接收显示☐ 自动断帧

? 20

☐ 接收脚本

Add Timesta ▾

保存数据

清空数据

发送区设置.

☐ 发送文件

扩展命令

☐ 十六进制发送☐ 发送脚本

ADD8 ▾

☐ 定时发送

1.0 秒

☐ DTR☐ RTS

换行符

\r\n (CRLF) ▾

☒ 显示发送字符串 ▾

```
*****  UART communication using IDLE IT
+ DMA  *****
Please enter arbitrary length characters:
» UESTC
« UESTC
» QIQIANG
« QIQIANG
» 2020/03/19
« 2020/03/19
```

2020/03/19



发送 : 22

接收 : 121

复位计数

本章结束

本章结束

