



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Embedded System and Microcomputer Principle

---

## LAB4 USART Communication

---

2021 Fall  
wangq9@mail.sustech.edu.cn



# CONTENTS

- 1 USART Function Description
- 2 How to Program
- 3 Practice
- 4 ISP Serial Port Download



01

# USART Function Description



# 1. USART Function Description

## -- Basic Description of Serial Port

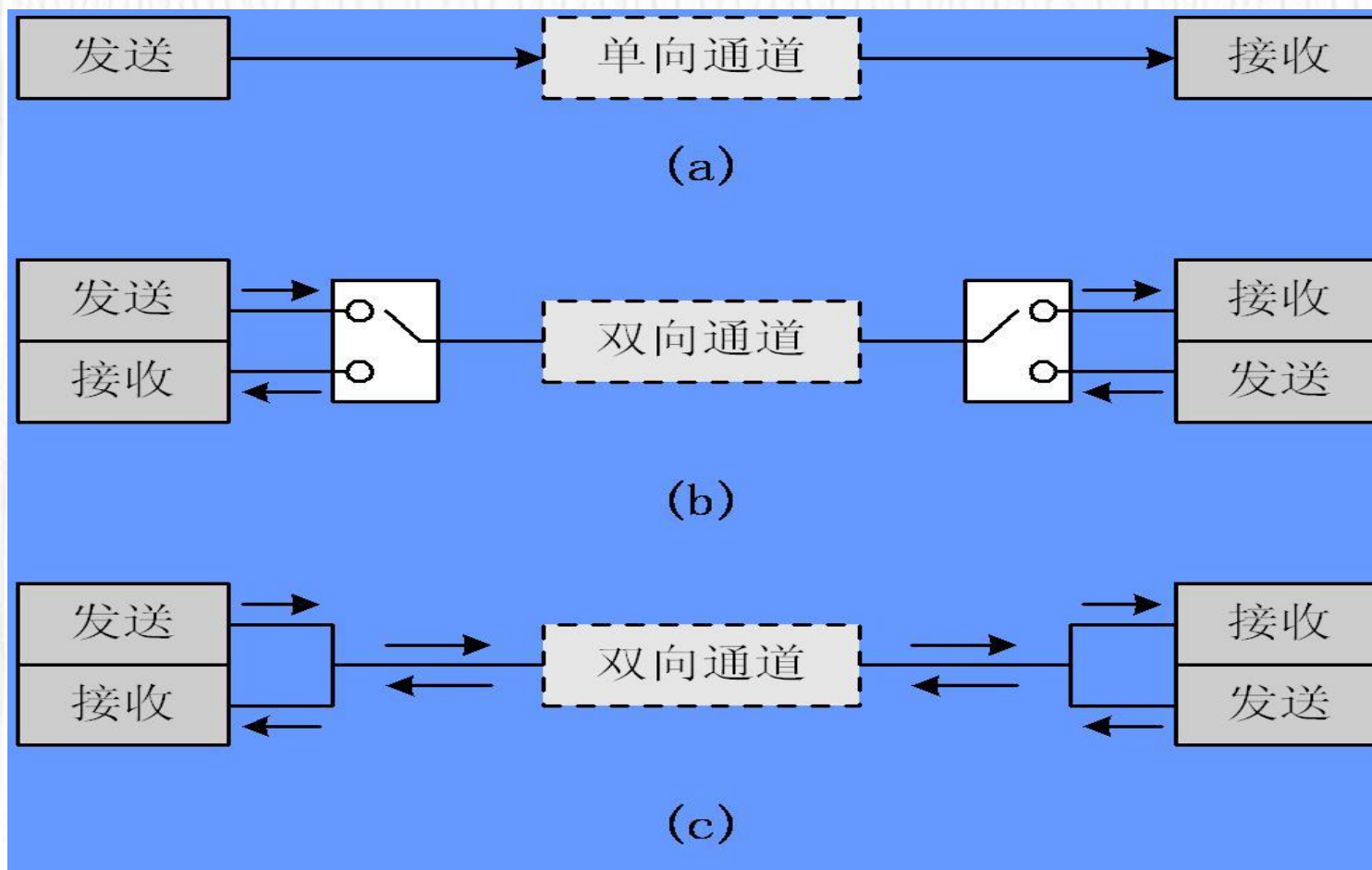
- 按照数据传送方向，分为：
  - **单工**：数据传输只支持数据在一个方向上传输
  - **半双工**：允许数据在两个方向上传输，但是，在某一时刻，只允许数据在一个方向上传输，它实际上是一种切换方向的单工通信
  - **全双工**：允许数据同时在两个方向上传输，因此，全双工通信是两个单工通信方式的结合，它要求发送设备和接收设备都有独立的接收和发送能力。



# 1. USART Function Description

## -- Basic Description of Serial Port

- 串行通信3种数据传送方式





# 1. USART Function Description

## -- Basic Description of Serial Port

- 串行通信的通信方式
  - **同步通信**：带时钟同步信号传输
    - SPI（Serial Peripheral Interface）串行外设接口
    - I<sup>2</sup>C（Inter-Integrated Circuit）集成电路总线
  - **异步通信**：不带时钟同步信号
    - UART（Universal Asynchronous Receiver/Transmitter）通用异步收发传输器

# 1. USART Function Description

## -- Basic Description of Serial Port

- 常见的串行通信接口对比

通信标准	引脚说明	通信方式	通信方向
UART	TXD: 发送端 RXD: 接受端 GND: 公共地	异步通信	全双工
1-WIRE	DQ: 发送/接受端	异步通信	半双工
SPI	SCK: 同步时钟 MISO: 主机输入, 从机输出 MOSI: 主机输出, 从机输入	同步通信	全双工
I <sup>2</sup> C	SCL: 同步时钟 SDA: 数据输入/输出端	同步通信	半双工



# 1. USART Function Description

## -- STM32 serial port

- STM32的串行通信接口

- 3 USART

- USART supports both synchronous and asynchronous transmission. In asynchronous mode, a USART bidirectional communication needs two pins: receive data in(RX) and transmit data out(TX). In synchronous mode, an addition pin(SCLK) for clock synchronous is needed. In most cases, we use USART in asynchronous mode, which is UART, and only UART is involved in the following lab

- 2 UART



# 1. USART Function Description

-- STM32 serial port

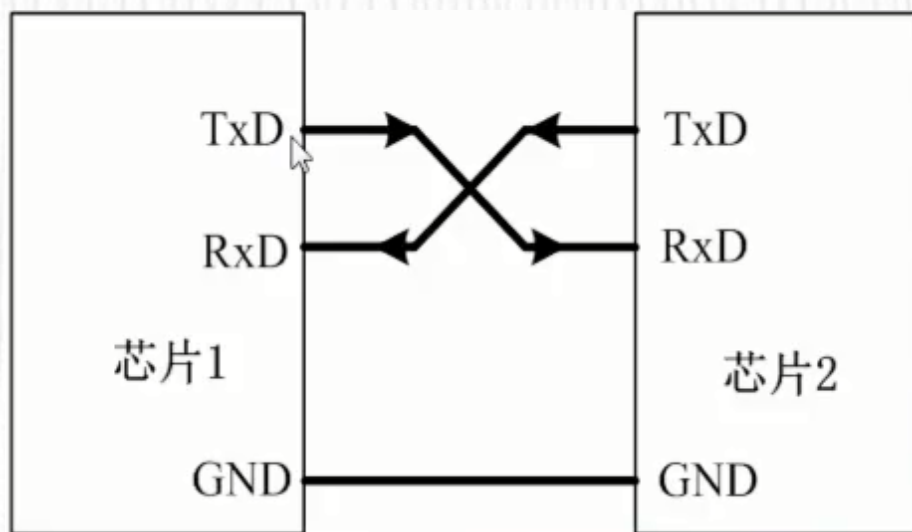
- STM32 UART pin

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

# 1. USART Function Description

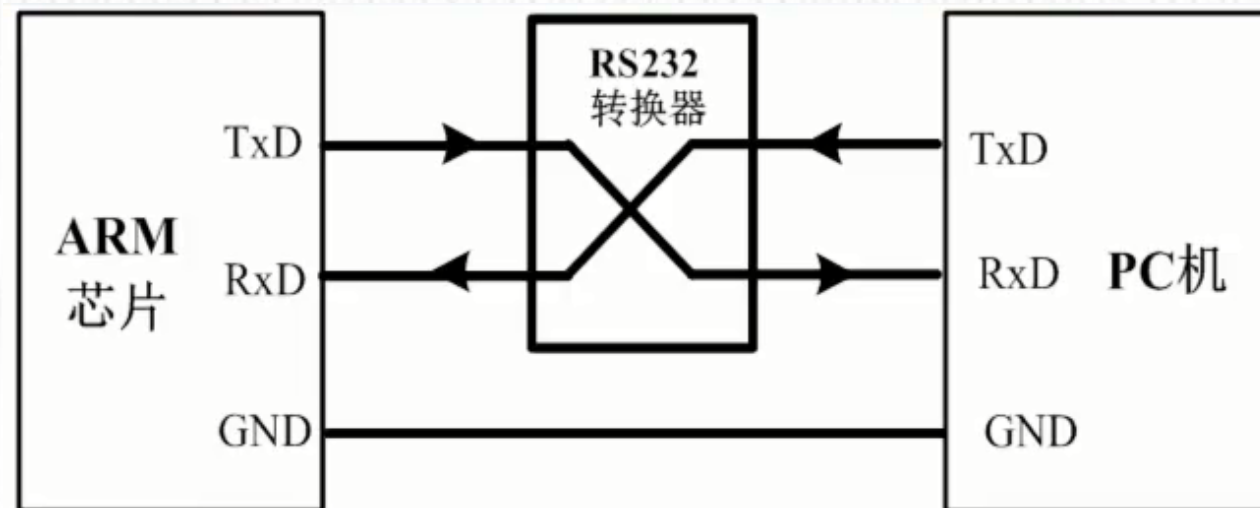
## -- UART

- UART异步通信方式引脚连接方法



(a)

communication between two chips  
with the same electrical characteristics



(b)

communication between computer and chip  
with different electrical characteristics

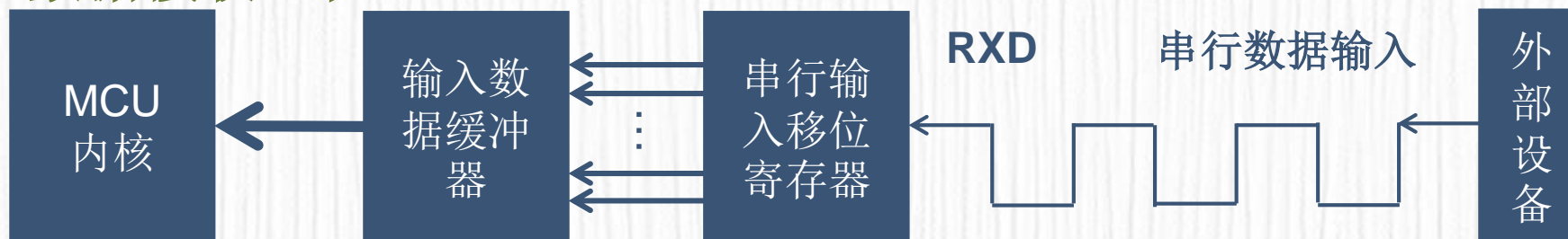


# 1. USART Function Description

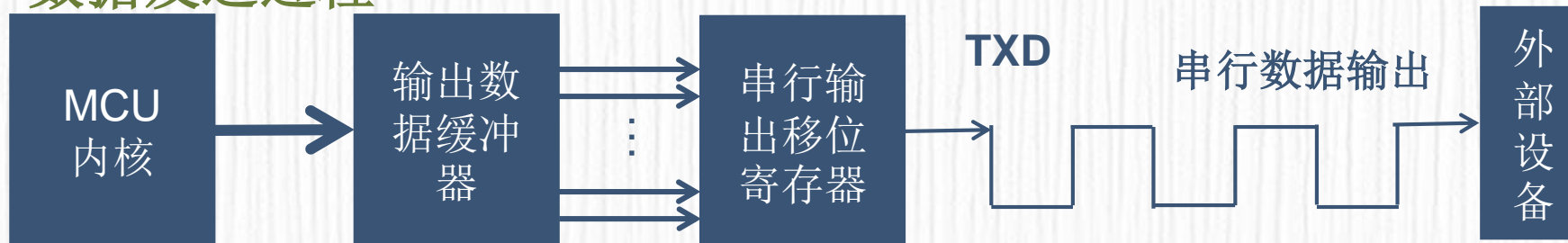
## -- UART

- UART串口异步通信过程

### 数据接收过程



### 数据发送过程



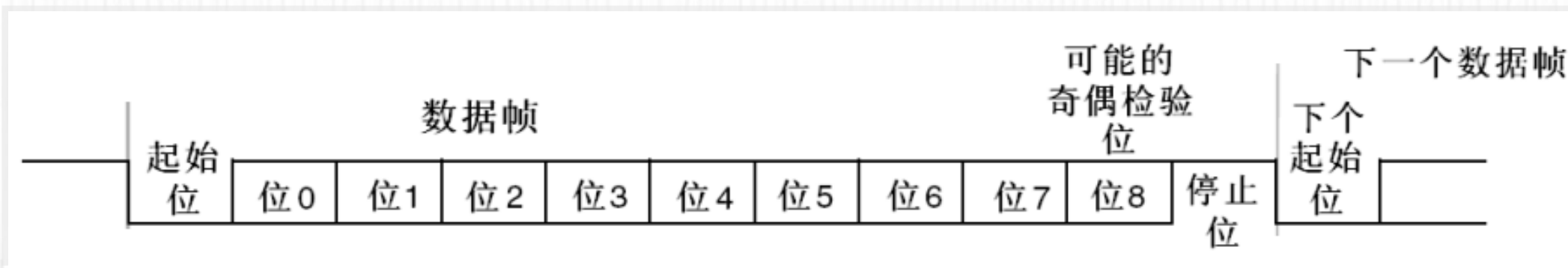


# 1. USART Function Description

## -- UART

- STM32串口异步通信需要定义的参数
  - 起始位
  - 数据位（8位或者9位）
  - 奇偶校验位（奇校验或者偶校验）
  - 停止位
  - 波特率设置

### An example







- # 1. USART Function Description
- STM32 Serial Port Register
    - USART\_SR 状态寄存器
    - USART\_DR 数据寄存器
    - USART\_BRR 波特率寄存器
    - USART\_CR 控制寄存器



# 1. USART Function Description

## -- STM32 USART\_SR

### 状态寄存器(USART\_SR)

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

位31:10	保留位，硬件强制为0
位9	<b>CTS</b> : CTS 标志 (CTS flag) 如果设置了CTSE位，当nCTS输入变化状态时，该位被硬件置高。由软件将其清零。如果USART_CR3中的CTSIE为'1'，则产生中断。 0: nCTS状态线上没有变化; 1: nCTS状态线上发生变化。 注: UART4和UART5上不存在这一位。
位8	<b>LBD</b> : LIN断开检测标志 (LIN break detection flag) 当检测到LIN断开时，该位由硬件置'1'，由软件清'0'(向该位写0)。如果USART_CR3中的LBDIE = 1，则产生中断。 0: 没有检测到LIN断开; 1: 检测到LIN断开。 注意: 若LBDIE=1，当LBD为'1'时要产生中断。



# 1. USART Function Description

## -- STM32 USART\_SR(continued)

位7	<p><b>TXE:</b>发送数据寄存器空 (Transmit data register empty)</p> <p>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。</p> <p>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。</p> <p>注意: 单缓冲器传输中使用该位。</p>
位6	<p><b>TC:</b> 发送完成 (Transmission complete)</p> <p>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, 只有在多缓冲通信中才推荐这种清除程序。</p> <p>0: 发送还未完成; 1: 发送完成。</p>
位5	<p><b>RXNE:</b> 读数据寄存器非空 (Read data register not empty)</p> <p>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, 只有在多缓冲通信中才推荐这种清除程序。</p> <p>0: 数据没有收到; 1: 收到数据, 可以读出。</p>
位4	<p><b>IDLE:</b> 监测到总线空闲 (IDLE line detected)</p> <p>当检测到总线空闲时, 该位被硬件置位。如果USART_CR1中的IDLEIE为'1', 则产生中断。由软件序列清除该位(先读USART_SR, 然后读USART_DR)。</p> <p>0: 没有检测到空闲总线; 1: 检测到空闲总线。</p> <p>注意: IDLE位不会再次被置高直到RXNE位被置起(即又检测到一次空闲总线)</p>

位3	<p><b>ORE:</b> 过载错误 (Overrun error)</p> <p>当RXNE仍然是'1'的时候, 当前被接收在移位寄存器中的数据, 需要传送至RDR寄存器时, 硬件将该位置位。如果USART_CR1中的RXNEIE为'1'的话, 则产生中断。由软件序列将其清零(先读USART_SR, 然后读USART_CR)。</p> <p>0: 没有过载错误; 1: 检测到过载错误。</p> <p>注意: 该位被置位时, RDR寄存器中的值不会丢失, 但是移位寄存器中的数据会被覆盖。如果设置了EIE位, 在多缓冲器通信模式下, ORE标志置位会产生中断的。</p>
位2	<p><b>NE:</b> 噪声错误标志 (Noise error flag)</p> <p>在接收到的帧检测到噪音时, 由硬件对该位置位。由软件序列对其清零(先读USART_SR, 再读USART_DR)。</p> <p>0: 没有检测到噪声; 1: 检测到噪声。</p> <p>注意: 该位不会产生中断, 因为它和RXNE一起出现, 硬件会在设置RXNE标志时产生中断。在多缓冲区通信模式下, 如果设置了EIE位, 则设置NE标志时会产生中断。</p>
位1	<p><b>FE:</b> 帧错误 (Framing error)</p> <p>当检测到同步错位, 过多的噪声或者检测到断开符, 该位被硬件置位。由软件序列将其清零(先读USART_SR, 再读USART_DR)。</p> <p>0: 没有检测到帧错误; 1: 检测到帧错误或者break符。</p> <p>注意: 该位不会产生中断, 因为它和RXNE一起出现, 硬件会在设置RXNE标志时产生中断。如果当前传输的数据既产生了帧错误, 又产生了过载错误, 硬件还是会继续该数据的传输, 并且只设置ORE标志位。</p> <p>在多缓冲区通信模式下, 如果设置了EIE位, 则设置FE标志时会产生中断。</p>
位0	<p><b>PE:</b> 校验错误 (Parity error)</p> <p>在接收模式下, 如果出现奇偶校验错误, 硬件对该位置位。由软件序列对其清零(依次读USART_SR和USART_DR)。在清除PE位前, 软件必须等待RXNE标志位被置'1'。如果USART_CR1中的PEIE为'1', 则产生中断。</p> <p>0: 没有奇偶校验错误; 1: 奇偶校验错误。</p>

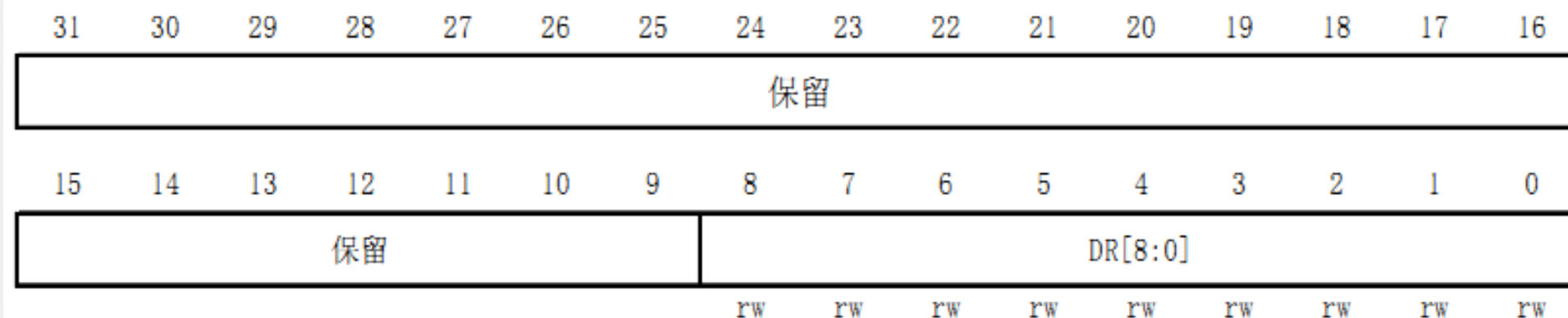




# 1. USART Function Description

## -- STM32 USART\_DR

### 数据寄存器(USART\_DR)



位31:9	保留位，硬件强制为0
位8:0	<b>DR[8:0]: 数据值 (Data value)</b> 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。



### 波特比率寄存器(USART\_BRR)



位31:16	保留位，硬件强制为0
位15:4	<b>DIV_Mantissa[11:0]:</b> 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	<b>DIV_Fraction[3:0]:</b> 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。



# 1. USART Function Description

## -- STM32 USART\_CR1

### 控制寄存器 1(USART\_CR1)

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	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	
res	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

位31:14	保留位，硬件强制为0。
位13	<b>UE</b> : USART使能 (USART enable) 当该位被清零，在当前字节传输完成后USART的分频器和输出停止工作，以减少功耗。该位由软件设置和清零。 0: USART分频器和输出被禁止; 1: USART模块使能。
位12	<b>M</b> : 字长 (Word length) 该位定义了数据字的长度，由软件对其设置和清零 0: 一个起始位，8个数据位，n个停止位; 1: 一个起始位，9个数据位，n个停止位。 注意：在数据传输过程中(发送或者接收时)，不能修改这个位。
位11	<b>WAKE</b> : 唤醒的方法 (Wakeup method) 这位决定了把USART唤醒的方法，由软件对该位设置和清零。 0: 被空闲总线唤醒; 1: 被地址标记唤醒。



# 1. USART Function Description

## -- STM32 USART\_CR1(continued)

位10	<b>PCE:</b> 检验控制使能 (Parity control enable) 用该位选择是否进行硬件校验控制(对于发送来说就是校验位的产生; 对于接收来说就是校验位的检测)。当使能了该位, 在发送数据的最高位(如果M=1, 最高位就是第9位; 如果M=0, 最高位就是第8位)插入校验位; 对接收到的数据检查其校验位。软件对它置'1'或清'0'。一旦设置了该位, 当前字节传输完成后, 校验控制才生效。 0: 禁止校验控制; 1: 使能校验控制。
位9	<b>PS:</b> 校验选择 (Parity selection) 当校验控制使能后, 该位用来选择是采用偶校验还是奇校验。软件对它置'1'或清'0'。当前字节传输完成后, 该选择生效。 0: 偶校验; 1: 奇校验。
位8	<b>PEIE:</b> PE中断使能 (PE interrupt enable) 该位由软件设置或清除。 0: 禁止产生中断; 1: 当USART_SR中的PE为'1'时, 产生USART中断。
位7	<b>TXEIE:</b> 发送缓冲区空中断使能 (TXE interrupt enable) 该位由软件设置或清除。 0: 禁止产生中断; 1: 当USART_SR中的TXE为'1'时, 产生USART中断。
位6	<b>TCIE:</b> 发送完成中断使能 (Transmission complete interrupt enable) 该位由软件设置或清除。 0: 禁止产生中断; 1: 当USART_SR中的TC为'1'时, 产生USART中断。
位5	<b>RXNEIE:</b> 接收缓冲区非空中断使能 (RXNE interrupt enable) 该位由软件设置或清除。 0: 禁止产生中断; 1: 当USART_SR中的ORE或者RXNE为'1'时, 产生USART中断。

位4	<b>IDLEIE:</b> IDLE中断使能 (IDLE interrupt enable) 该位由软件设置或清除。 0: 禁止产生中断; 1: 当USART_SR中的IDLE为'1'时, 产生USART中断。
位3	<b>TE:</b> 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	<b>RE:</b> 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。
位1	<b>RWU:</b> 接收唤醒 (Receiver wakeup) 该位用来决定是否把USART置于静默模式。该位由软件设置或清除。当唤醒序列到来时, 硬件也会将其清零。 0: 接收器处于正常工作模式; 1: 接收器处于静默模式。 注意: 1. 在把USART置于静默模式(设置RWU位)之前, USART要已经先接收了一个数据字节。否则在静默模式下, 不能被空闲总线检测唤醒。 2. 当配置成地址标记检测唤醒(WAKE位=1), 在RXNE位被置位时, 不能用软件修改RWU位。
位0	<b>SBK:</b> 发送断开帧 (Send break) 使用该位来发送断开字符。该位可以由软件设置或清除。操作过程应该是软件设置位它, 然后在断开帧的停止位时, 由硬件将该位置位。 0: 没有发送断开字符; 1: 将要发送断开字符。



# 1. USART Function Description

## -- STM32 USART\_CR2



### 控制寄存器 2(USART\_CR2)

地址偏移: 0x10

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
保留	LINEN	STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	保留	LBDIE	LBDL	保留	ADD[3:0]				
RW	RW	RW	RW	RW	RW	RW	RW		RW	RW		RW	RW	RW	RW

位31:15	保留位，硬件强制为0。
位14	<b>LINEN</b> : LIN模式使能 (LIN mode enable) 该位由软件设置或清除。 0: 禁止LIN模式; 1: 使能LIN模式。 在LIN模式下，可以用USART_CR1寄存器中的SBK位发送LIN同步断开符(低13位)，以及检测LIN同步断开符。
位13:12	<b>STOP</b> : 停止位 (STOP bits) 这2位用来设置停止位的位数 00: 1个停止位; 01: 0.5个停止位; 10: 2个停止位; 11: 1.5个停止位; 注: UART4和UART5不能用0.5停止位和1.5停止位。
位11	<b>CLKEN</b> : 时钟使能 (Clock enable) 该位用来使能CK引脚 0: 禁止CK引脚; 1: 使能CK引脚。 注: UART4和UART5上不存在这一位。





# 1. USART Function Description

## -- STM32 USART\_CR2(continued)

位10	<b>CPOL:</b> 时钟极性 (Clock polarity) 在同步模式下, 可以用该位选择SLCK引脚上时钟输出的极性。和CPHA位一起配合来产生需要的时钟/数据的采样关系 0: 总线空闲时CK引脚上保持低电平; 1: 总线空闲时CK引脚上保持高电平。 注: UART4和UART5上不存在这一位。
位9	<b>CPHA:</b> 时钟相位 (Clock phase) 在同步模式下, 可以用该位选择SLCK引脚上时钟输出的相位。和CPOL位一起配合来产生需要的时钟/数据的采样关系(参见图259和图260)。 0: 在时钟的第一个边沿进行数据捕获; 1: 在时钟的第二个边沿进行数据捕获。 注: UART4和UART5上不存在这一位。
位8	<b>LBCL:</b> 最后一位时钟脉冲 (Last bit clock pulse) 在同步模式下, 使用该位来控制是否在CK引脚上输出最后发送的那个数据字节(MSB)对应的时钟脉冲 0: 最后一位数据的时钟脉冲不从CK输出; 1: 最后一位数据的时钟脉冲会从CK输出。 注意: 1. 最后一个数据位就是第8或者第9个发送的位(根据USART_CR1寄存器中的M位所定义的8或者9位数据帧格式)。 2. UART4和UART5上不存在这一位。

位7	保留位, 硬件强制为0
位6	<b>LBDIE:</b> LIN断开符检测中断使能 (LIN break detection interrupt enable) 断开符中断屏蔽(使用断开分隔符来检测断开符) 0: 禁止中断; 1: 只要USART_SR寄存器中的LBD为'1'就产生中断。
位5	<b>LBDL:</b> LIN断开符检测长度 (LIN break detection length) 该位用来选择是11位还是10位的断开符检测 0: 10位的断开符检测; 1: 11位的断开符检测。
位4	保留位, 硬件强制为0
位3:0	<b>ADD[3:0]:</b> 本设备的USART节点地址 该位域给出本设备USART节点的地址。 这是在多处理器通信下的静默模式中使用的, 使用地址标记来唤醒某个USART设备。



# 1. USART Function Description

## -- STM32 USART\_CR3

### 控制寄存器 3(USART\_CR3)

地址偏移: 0x14

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
保留						CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	EIE
						rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

位31:11	保留位, 硬件强制为0
位10	<b>CTSIE:</b> CTS中断使能 (CTS interrupt enable) 0: 禁止中断; 1: USART_SR寄存器中的CTS为'1'时产生中断。 注: UART4和UART5上不存在这一位。
位9	<b>CTSE:</b> CTS使能 (CTS enable) 0: 禁止CTS硬件流控制; 1: CTS模式使能, 只有nCTS输入信号有效(拉成低电平)时才能发送数据。如果在数据传输的过程中, nCTS信号变成无效, 那么发完这个数据后, 传输就停止下来。如果当nCTS为无效时, 往数据寄存器里写数据, 则要等到nCTS有效时才会发送这个数据。 注: UART4和UART5上不存在这一位。
位8	<b>RTSE:</b> RTS使能 (RTS enable) 0: 禁止RTS硬件流控制; 1: RTS中断使能, 只有接收缓冲区内有空余的空间时才请求下一个数据。当前数据发送完成后, 发送操作就需要暂停下来。如果可以接收数据了, 将nRTS输出置为有效(拉至低电平)。 注: UART4和UART5上不存在这一位。



# 1. USART Function Description

## -- STM32 USART\_CR3(continued)

位7	<b>DMAT:</b> DMA使能发送 (DMA enable transmitter) 该位由软件设置或清除。 0: 禁止发送时的DMA模式。 1: 使能发送时的DMA模式; 注: UART4和UART5上不存在这一位。
位6	<b>DMAR:</b> DMA使能接收 (DMA enable receiver) 该位由软件设置或清除。 0: 禁止接收时的DMA模式。 1: 使能接收时的DMA模式; 注: UART4和UART5上不存在这一位。
位5	<b>SCEN:</b> 智能卡模式使能 (Smartcard mode enable) 该位用来使能智能卡模式 0: 禁止智能卡模式; 1: 使能智能卡模式。 注: UART4和UART5上不存在这一位。
位4	<b>NACK:</b> 智能卡NACK使能 (Smartcard NACK enable) 0: 校验错误出现时, 不发送NACK; 1: 校验错误出现时, 发送NACK。 注: UART4和UART5上不存在这一位。

位3	<b>HDSEL:</b> 半双工选择 (Half-duplex selection) 选择单线半双工模式 0: 不选择半双工模式; 1: 选择半双工模式。
位2	<b>IRLP:</b> 红外低功耗 (IrDA low-power) 该位用来选择普通模式还是低功耗红外模式 0: 通常模式; 1: 低功耗模式。
位1	<b>IREN:</b> 红外模式使能 (IrDA mode enable) 该位由软件设置或清除。 0: 不使能红外模式; 1: 使能红外模式。
位0	<b>EIE:</b> 错误中断使能 (Error interrupt enable) 在多缓冲区通信模式下, 当有帧错误、过载或者噪声错误时(USART_SR中的FE=1, 或者ORE=1, 或者NE=1)产生中断。 0: 禁止中断; 1: 只要USART_CR3中的DMAR=1, 并且USART_SR中的FE=1, 或者ORE=1, 或者NE=1, 则产生中断





# 1. USART Function Description

## -- Serial Port Configuration Steps

- 串口时钟使能, GPIO时钟使能
- 串口复位 (这一步不是必须的)
- GPIO端口模式设置
- 串口参数初始化
- 开启中断并且初始化NVIC
- 使能串口
- 编写中断处理函数
- 串口数据收发
- 串口传输状态获取





02

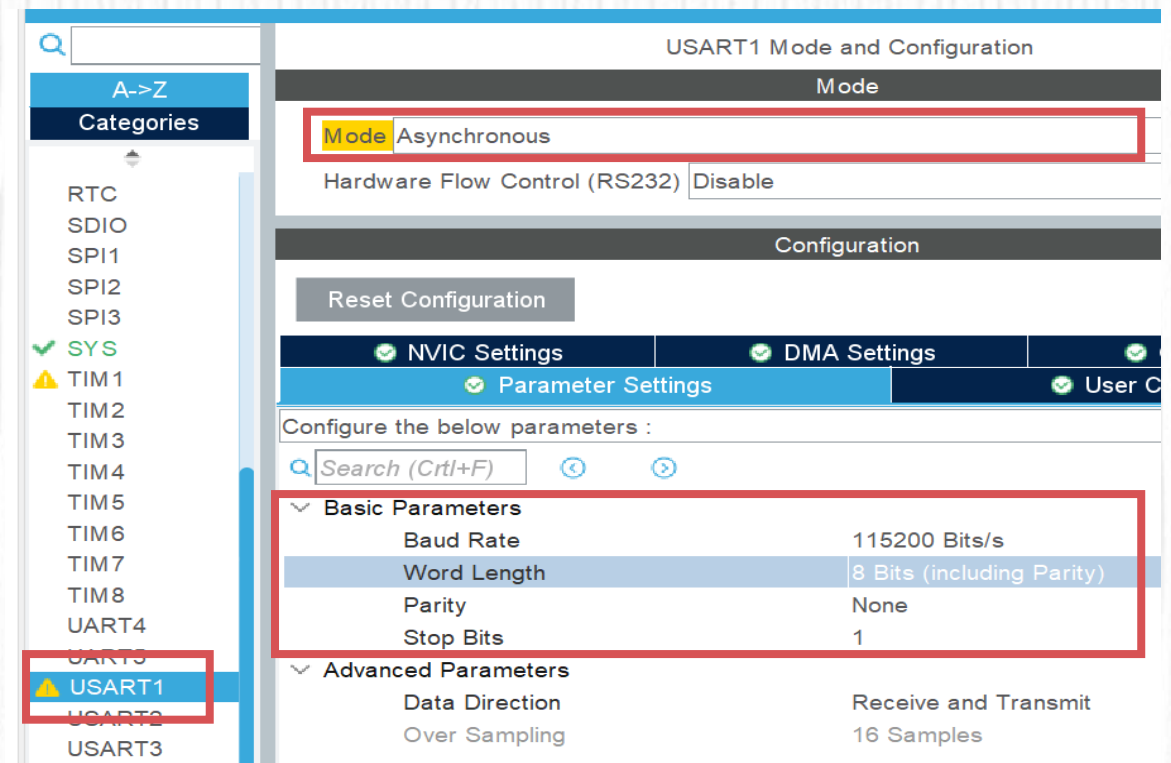
## How to Program



## 2. How to Program

- Our Goal
  - Make MiniSTM32 board communicate with the computer through the serial port
  - PC sends a message to MiniSTM32 board, and then MiniSTM32 board sends the same message back to PC

- 
- Pinout diagram of the STM32F103RCTx LQFP64 package. The diagram shows the chip with its pins and labels. A red box highlights pins PA9 and PA10, which are labeled USART1\_TX and USART1\_RX respectively. Other pins are labeled with their functions, such as VDD, VSS, PB9, PB8, BOOT, PB7, PB6, PB5, PB4, PB3, PD2, PC12, PC11, PC10, PA15, PA14, VBAT, PC13, PC14, PC15, PD0, PD1, NRST, PC0, PC1, PC2, PC3, VSSA, VDDA, PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7, PC4, PC5, PB0, PB1, PB2, PB10, PB11, VSS, and VDD. The chip is labeled STM32F103RCTx LQFP64.





## 2. How to Program

- Receive Data in interruption Mode
  - In most case, we transmit data in **blocking mode**, but receive the data by **interruption**, so we should enable the interruption for the corresponding USART

Configuration

✓ NVIC ✓ Code generation

Priority Group 2 bits for pre-emption prio... ☐ Sort by Preemption Priority and Sub Priority ☐ Sort by interrupts names

Search  Show  ☒ Force DMA channels Interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
USART1 global interrupt	<input checked="" type="checkbox"/>	1	0





## 2. How to Program

- Some structures and functions we used

```
MX_USART1_UART_Init();
```

```
UART_HandleTypeDef huart1;
```

```
huart1.Instance = USART1;  
huart1.Init.BaudRate = 115200;  
huart1.Init.WordLength = UART_WORDLENGTH_8B;  
huart1.Init.StopBits = UART_STOPBITS_1;  
huart1.Init.Parity = UART_PARITY_NONE;  
huart1.Init.Mode = UART_MODE_TX_RX;  
huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;  
huart1.Init.OverSampling = UART_OVERSAMPLING_16;
```



## 2. How to Program

- Some structures and functions we used
  - HAL\_UART\_Transmit() and HAL\_UART\_Receive() functions are used to transmit/receive data in blocking mode. Which means, unless transmit/receive is completed or timeout, the programs will stop here.
  - But in most case, we transmit data in **blocking mode**, but receive the data by **interruption**, so we use HAL\_UART\_Receive\_IT() function instead.
  - These three functions are in stm32f1xx\_hal\_uart.c

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

```
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

```
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)
```



## 2. How to Program

- Some structures and functions we used
  - In main.c, we need to call HAL\_UART\_Receive\_IT() function, which can receive data in non-blocking mode and trigger an interruption

```
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData,
uint16_t Size)
```

```
/* USER CODE BEGIN 0 */
extern UART_HandleTypeDef huart1;
extern uint8_t rxBuffer[20];
/* USER CODE END 0 */
```

```
int main(void)
{
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    /* USER CODE BEGIN 2 */
    HAL_UART_Receive_IT(&huart1, (uint8_t *)rxBuffer, 1);
```





## 2. How to Program

- Some structures and functions we used
  - When there is a interruption of UART coming, the program will go into function USART1\_IRQHandler(), it is in file stm32f1xx\_it.c

```
void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQn 0 */

    /* USER CODE END USART1_IRQn 0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQn 1 */

    /* USER CODE END USART1_IRQn 1 */
}
```

```
void HAL_UART_IRQHandler(UART_HandleTypeDef *huart)
{
    uint32_t isrflags   = READ_REG(huart->Instance->SR);
    uint32_t cr1its     = READ_REG(huart->Instance->CR1);
    uint32_t cr3its     = READ_REG(huart->Instance->CR3);
    uint32_t errorflags = 0x00U;
    uint32_t dmarequest = 0x00U;
    ....
}
```



## 2. How to Program

- Some structures and functions we used
  - When receive interruption is triggered, the program will call the function `UART_Receive_IT()` , which stores the character into the buffer `pRxBuffPtr` and call the receive complete callback function `HAL_UART_RxCpltCallback()` when it receive `RxXferCount` amounts of characters.
  - So we can re-implement the `HAL_UART_RxCpltCallback()` function if we want to receive data in non-blocking mode. For example, we can store the data in a array and sent it out when receive a line



## 2. How to Program

- Re-implement HAL\_UART\_RxCpltCallback() in stm32f1xx\_it.c, and **uint8\_t rxBuffer[20]** is also defined in this file

```
/* USER CODE BEGIN PV */  
uint8_t rxBuffer[20];  
/* USER CODE END PV */
```

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  
{  
    if(huart->Instance==USART1){  
        static unsigned char uRx_Data[1024] = {0};  
        static unsigned char uLength = 0;  
        if(rxBuffer[0] == '\n'){  
            HAL_UART_Transmit(&huart1, uRx_Data, uLength, 0xffff);  
            uLength = 0;  
        }else{  
            uRx_Data[uLength] = rxBuffer[0];  
            uLength++;  
        }  
    }  
}
```





## 2. How to Program

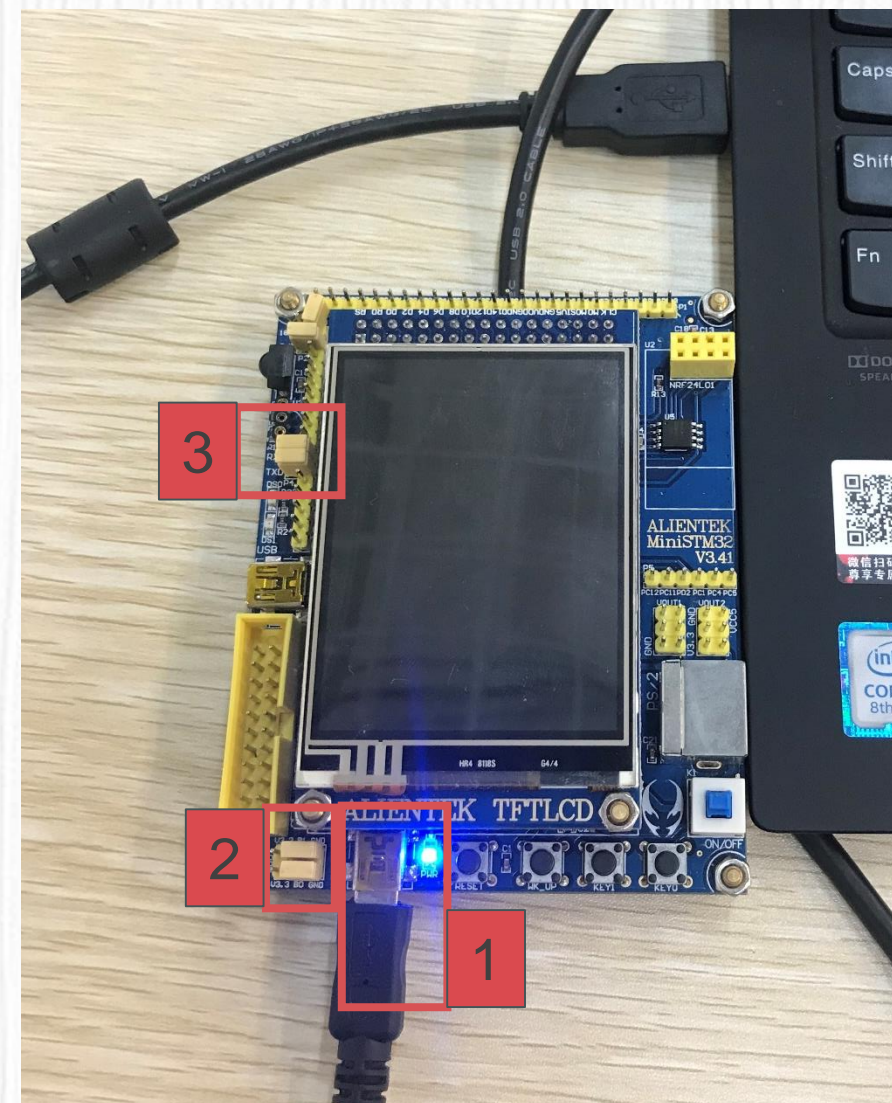
- Re-call HAL\_UART\_Receive\_IT() to receive data continuously in USART1\_IRQHandler() function

```
void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQn 0 */

    /* USER CODE END USART1_IRQn 0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQn 1 */
    HAL_UART_Receive_IT(&huart1, (uint8_t *)rxBuffer, 1);
    /* USER CODE END USART1_IRQn 1 */
}
```

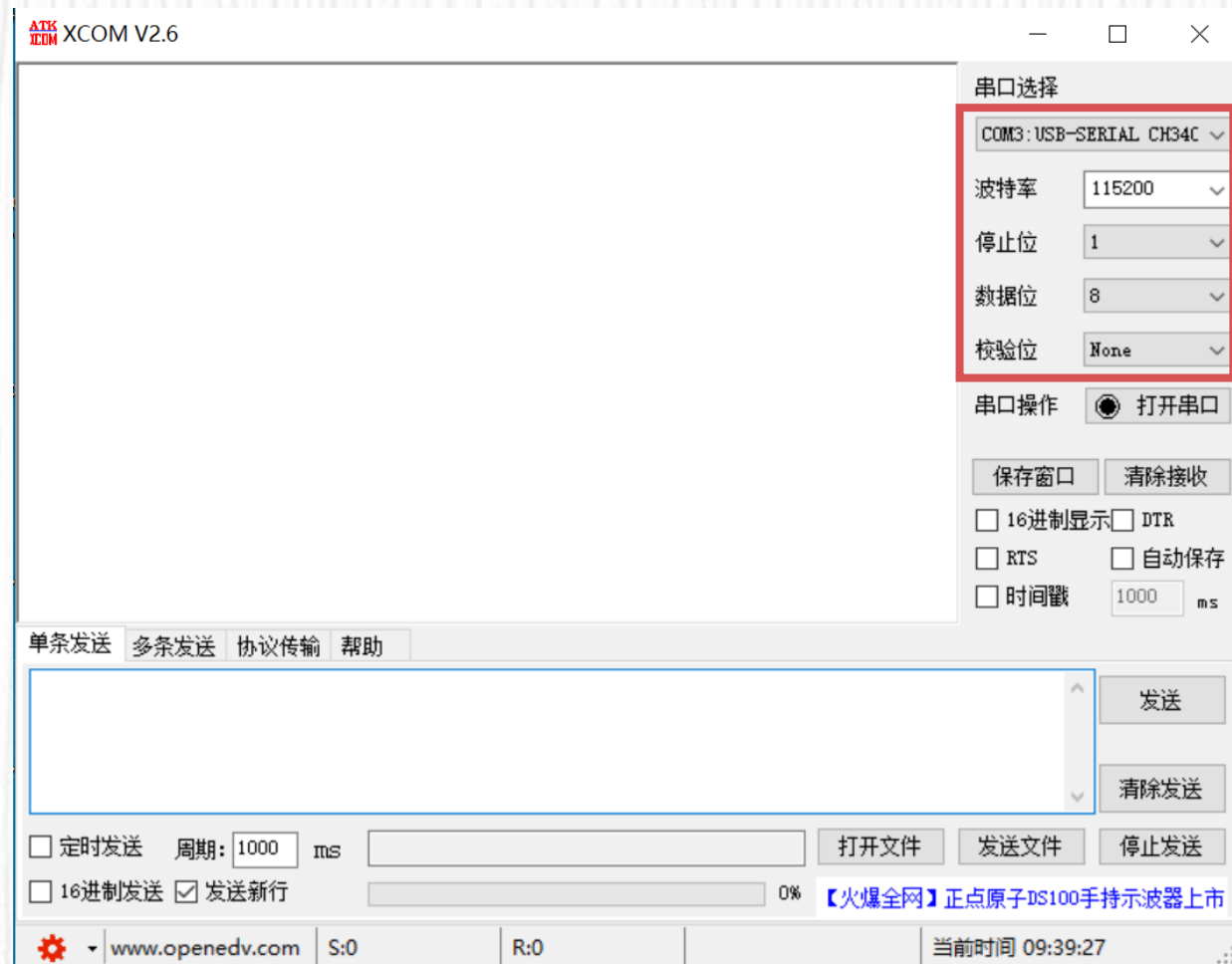
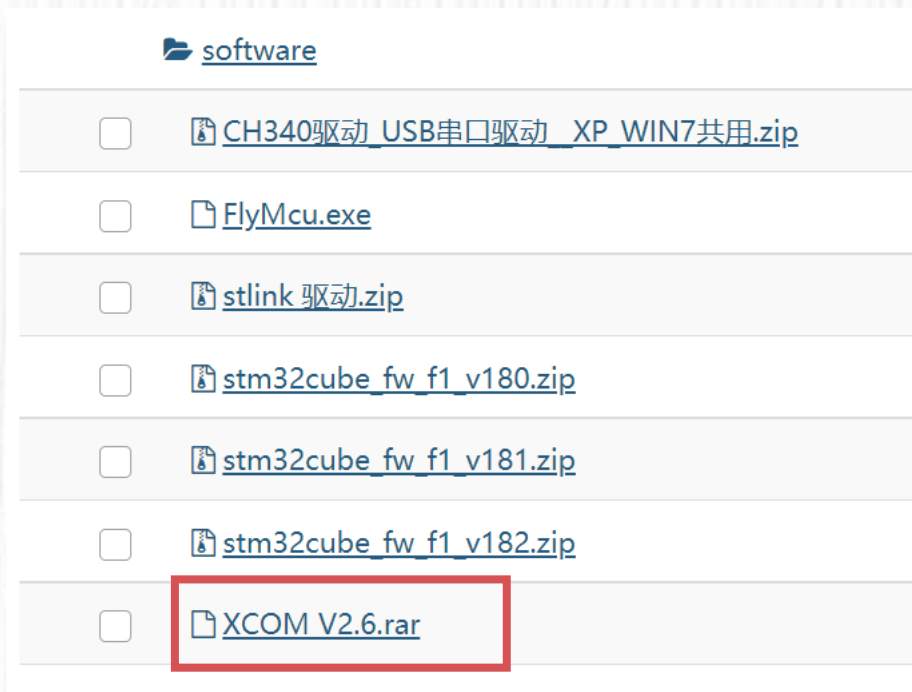
## 2. How to Program

- Communicate with PC - hardware wire connection
  - 1: use USB\_232 port
  - 2: both BOOT0 and BOOT1 connect to GND
  - 3: if choose USART1, connect PA9 and PA10 to TXD and RXD



## 2. How to Program

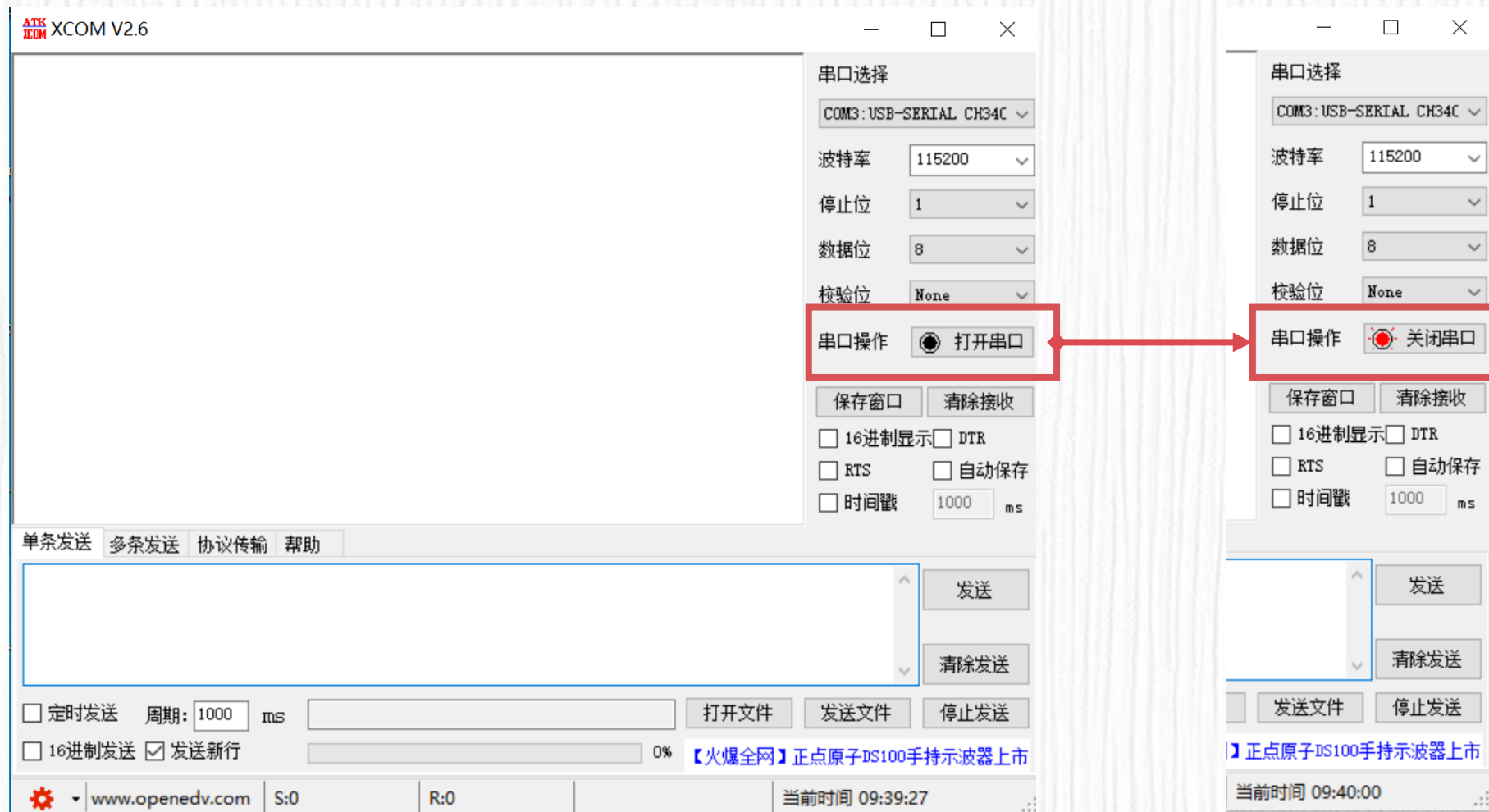
- Communicate with PC – PC serial port configuration





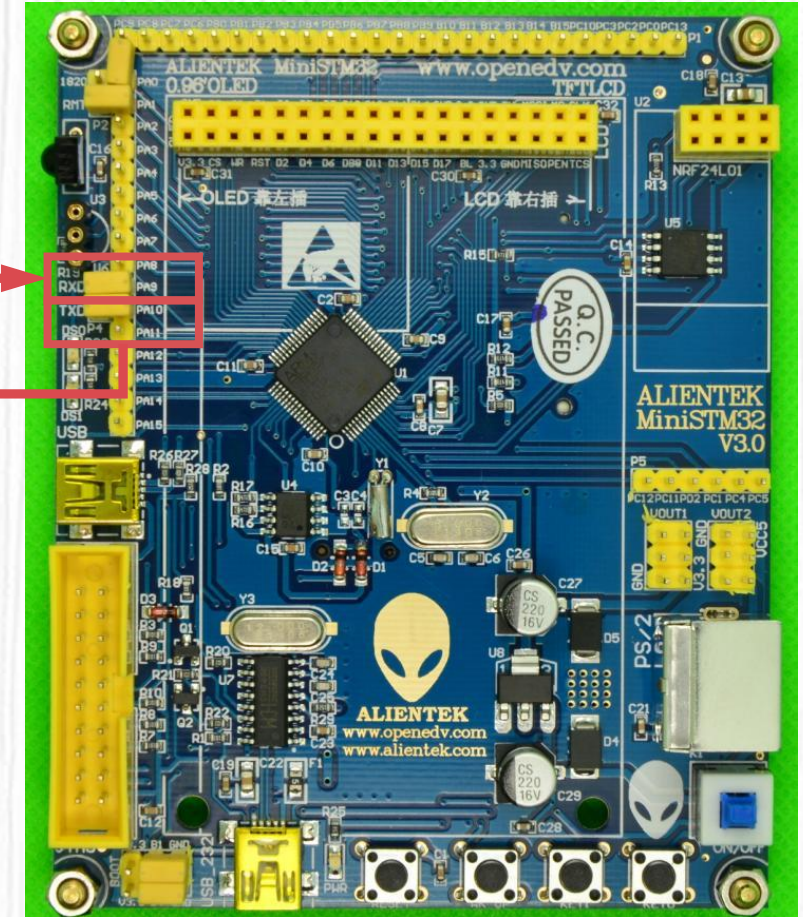
## 2. How to Program

- Communicate with PC – open serial port of PC



## 2. How to Program

- Communicate with PC – send messages





03

Practice



### 3. Practice



- Run the demo on MiniSTM32 board
- Complete the quiz on Sakai site(it will start after the holiday)



04

## ISP Serial Port Download



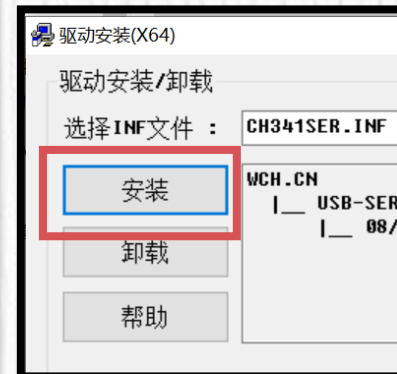
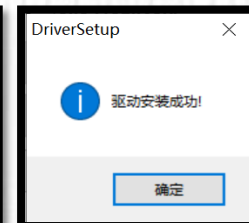
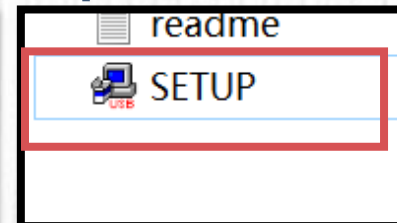
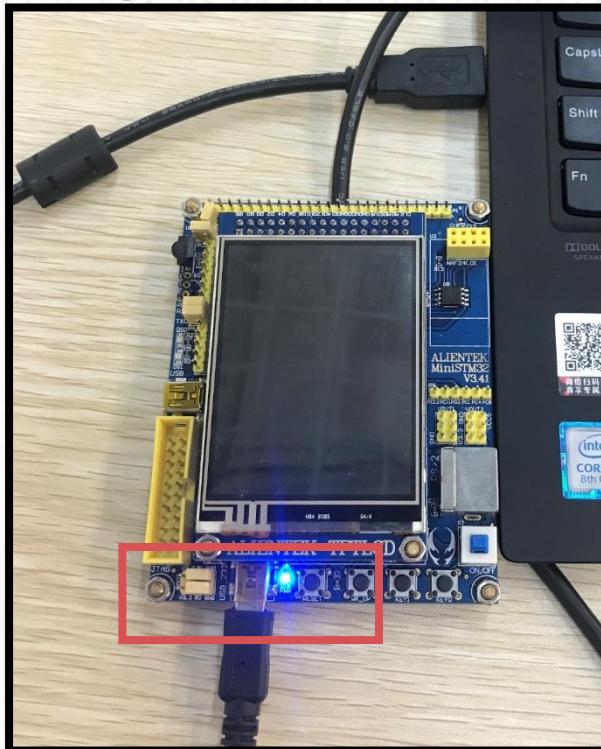
## 4. ISP Serial Port Download

- ISP(In-System Programming) is another method to download our program into MiniSTM32 board, you can use either ISP download or SW (ST-Link and J-Link) download method.
- NOTE: We can only use UART1 (PA9, PA10) to implete ISP download, while other UARTs, such as UART2 (PA2, PA3) is not available for ISP downloading



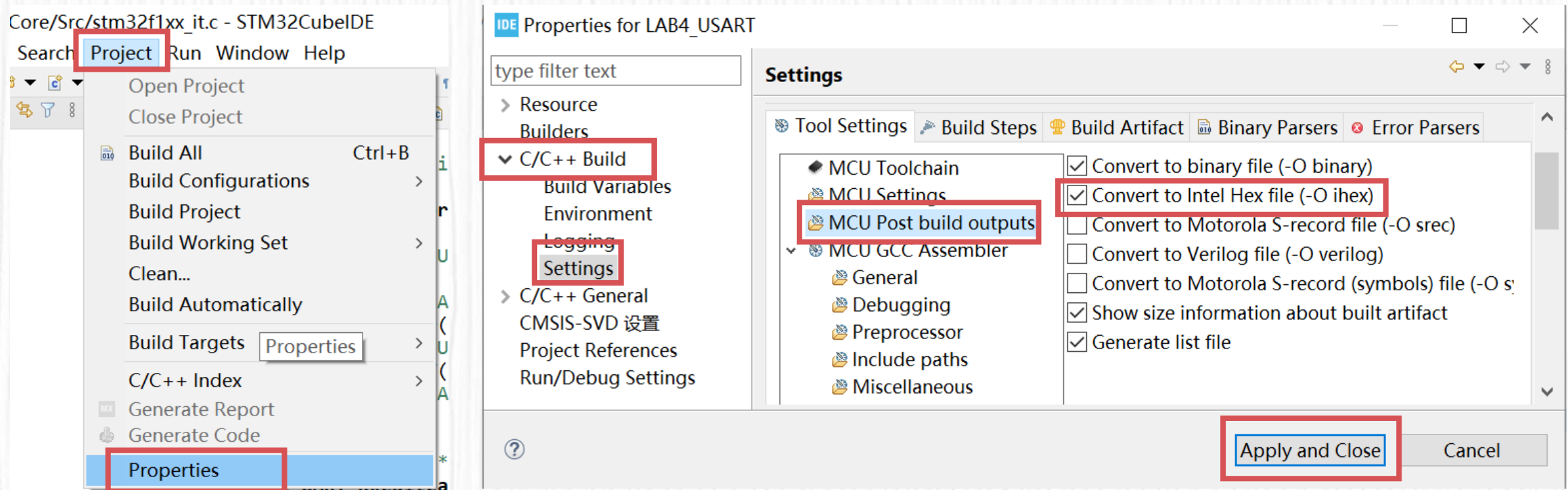
## 4. ISP Serial Port Download

- Download the serial port driver and serial assistant software from Sakai site
- Connect the MiniSTM32 board and PC with USB wire
- Unzip driver package and run setup.exe



## 4. ISP Serial Port Download

- Re-build the project to generate .hex file
- [Project] -> [Properties] -> [C/C++ Build] -> [Settings] -> [MCU Post build outputs] -> [Convert to Intel Hex file(-O ihex)] -> [Apply and Close]



The screenshot illustrates the steps to configure the build output in STM32CubeIDE:

- Project Menu:** The 'Project' menu is open, and 'Properties' is selected.
- Properties for LAB4\_USART:** The 'C/C++ Build' section is expanded, and 'Settings' is selected.
- Settings Dialog:** The 'Settings' dialog is open, showing the 'MCU Post build outputs' section. The 'Convert to Intel Hex file (-O ihex)' checkbox is checked.
- Buttons:** The 'Apply and Close' button is highlighted.



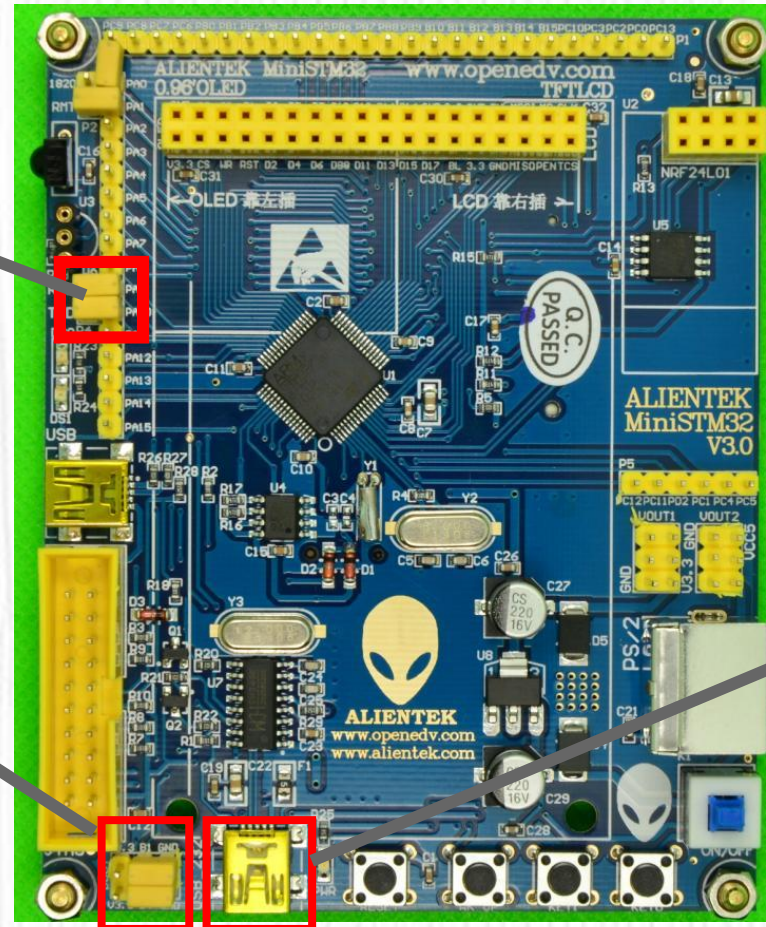
## 4. ISP Serial Port Download



- MiniSTM32 configuration

connect PA9  
and PA10 to  
TXD and  
RXD

both BOOT0  
and BOOT1  
connect to  
GND



use USB\_232 port



## 4. ISP Serial Port Download

- Run serial assistant
- 1: choose serial port
- 2: set bound rate
- 3: check on auto reload
- 4: check on Verify and Run After ISP complete
- 5: choose Reset RTS Low, ISP DTR High
- 6: choose your .hex file
- 7: run the program
- 8: press the RESET key on MiniSTM32 board

