# 第8章　串行通信接口

## 第1节　串行通信及接口基础

一、有关概念

二、典型异步串行通信接口简介

三、PC机的串行通信接口

## 第2节　MSP430通用串行通信模块

一、MSP430 USART 模块的两种工作方 式

二、MSP430 异步串行方式编程结构

三、利用MSP430 异步串行方式通信

# 第1节　串行通信基础

一、有关概念

二、典型异步串行通信接口简介

三、PC机的串行通信接口

## 一、有关概念

1．并行通信和串行通信

2. 信息传输的检错和纠错

3. 串行数据传送方式

4. 波特率

5. 通信协议

6. 串行通信基本方式

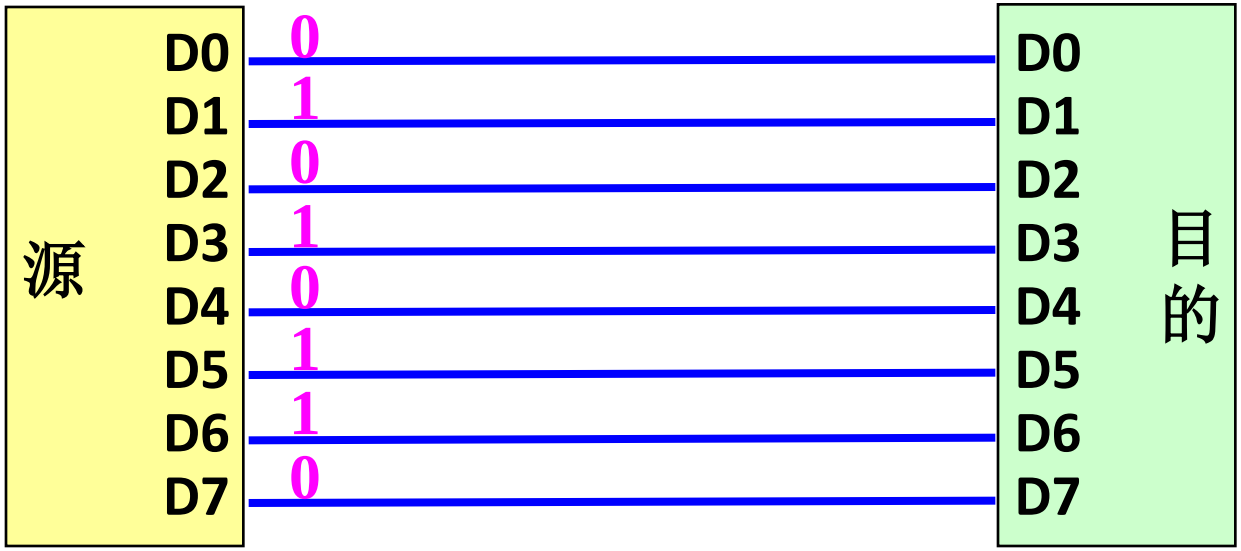
7. 异步串行通信

8. 接收/发送时钟和波特率因子

# 1. 并行通信和串行通信

## ●并行通信

将数据的各位同时在多根并行传输线上进行传输。




数据的各位同时由源到达目的地 → 快

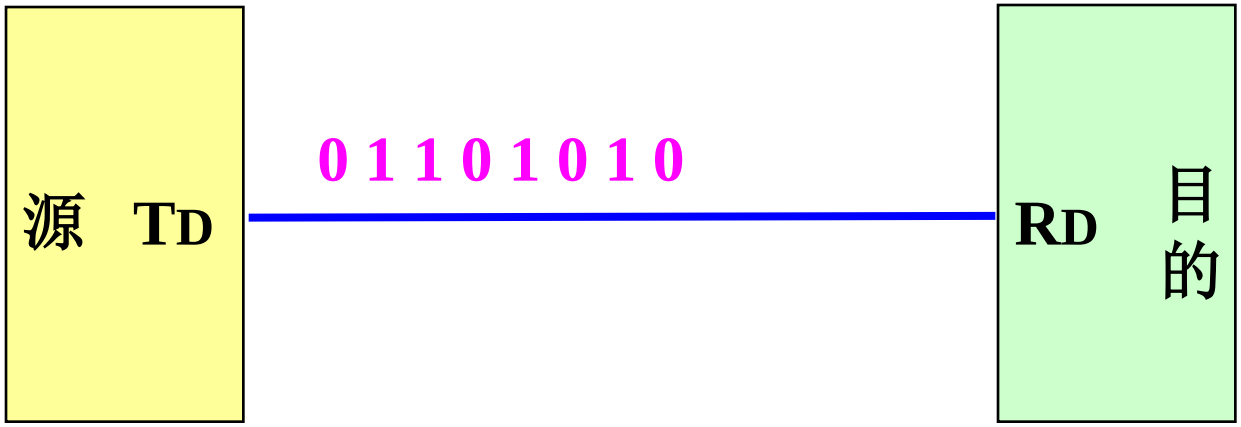
多根数据线 → 远程费用高

并行通信适于短距离、高速通信

*例如：* **PC机的标准并行接口LPT1 (打印机接口)、PCI总线、MSP430的P1~P6均可做8位并行通信**

## ●串行通信

将数据的各位按时间顺序依次在同一传输线上传输。

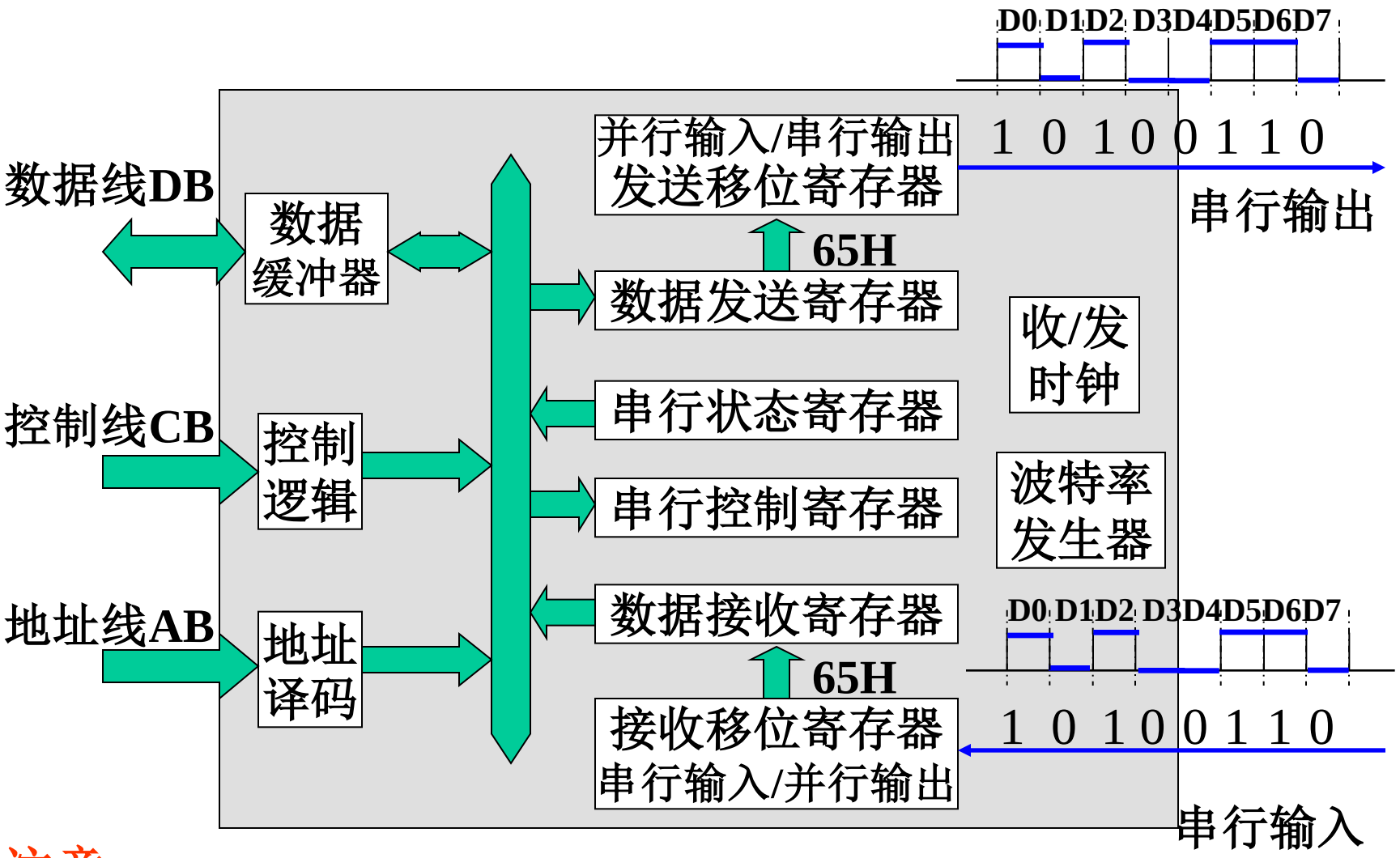



数据的各位依次由源到达目的地

数据线少 → 远程, 费用低

串行通信适于中、长距离通信

*例如：* **PC机的标准串行通信接口COM1, COM2**
**USB、以太网**




注意:

“串行”是外设与接口电路之间的数据传送方式，在CPU与接口之间仍是并行工作方式。

## 2. 信息传输的检错和纠错

- 串行数据在传输过程中，由于干扰可能引起信息的出错，

  如何发现传输中的错误，叫检错；

  发现错误后，如何消除错误，叫纠错

- 最简单的检错方法是奇偶校验，

  即在传送字符的各位之外，再传送1位奇/偶校验位。

  可采用奇校验或偶校验：

  奇校验：使所有传送的数位(含校验位)中1的个数为奇数

  偶校验：使所有传送的数位(含校验位)中1的个数为偶数

- 奇偶校验能够检测出1位误码，但是不能纠错。

## 3. 串行数据传送方式

按照数据流的方向，

分成三种基本的传送方式：

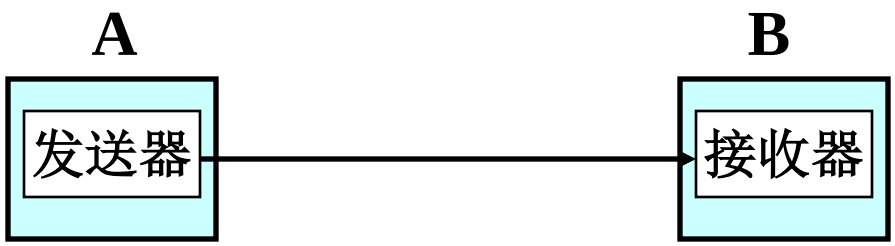
单工方式

全双工方式

半双工方式

# 单工方式

只允许数据按照一个固定的方向传送.



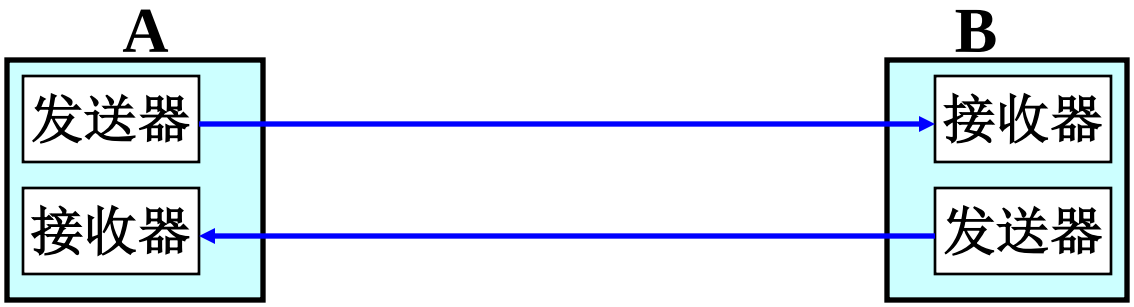

A方只能发送，称发送器

B方只能接收，称接收器

如：鼠标与主机的通信采用单工方式.

# 全双工方式

允许通讯双方同时进行发送和接收操作

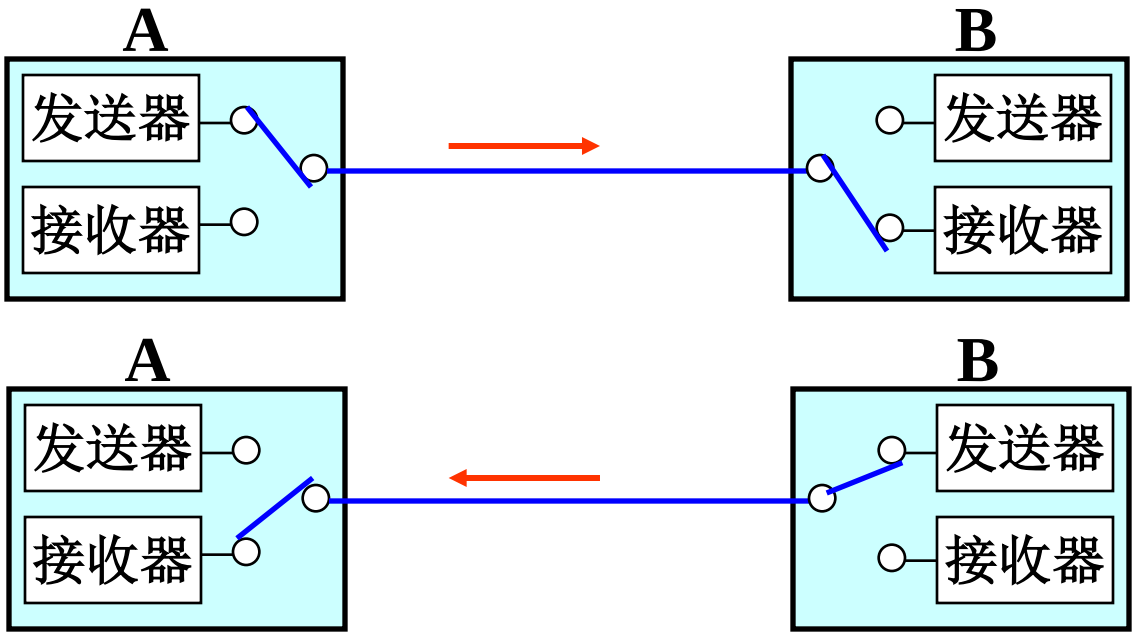



相当于把两个方向相反的单工方式组合在一起，需要两条传输线。

全双工方式主要应用于实时性较强的交互式应用中，如计算机之间的通信等。

# 半双工方式

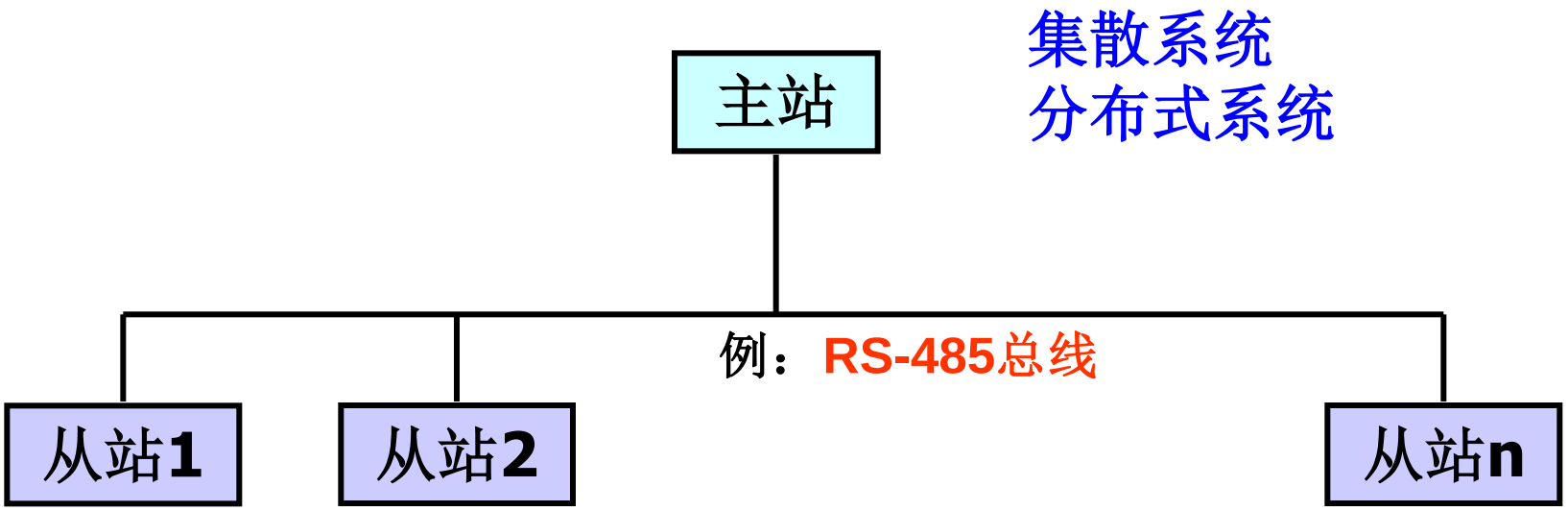
数据能从A方传送到B方，也能从B方传送到A方，
但是不能同时在两个方向上传送，
每次只能由一方发送，另一方接收。




通信双方通过软件控制的电子开关，进行方向的切换，
轮流地进行发送和接收。

一些简单的外部设备如键盘与主机的通信采用半双工方式.

# 半双工方式广泛地应用于主从结构的系统中




- 每个站均有通信地址(定址原则）
- 平时从站都处于接收状态，等待来自主站的命令
- 当从站接收到来自主站的命令后，若需要响应，
  则将自己置为发送状态，并发送响应，发送完毕后再置为接收状态
- 当主站要发送命令时，置主站为发送状态
  当主站发送命令完毕后，置主站为接收状态，以接收从站的响应

# 4. 波特率 (Baudrate)

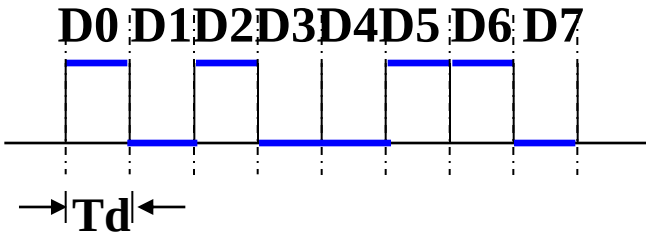
并行通信中，传输速率是以每秒多少字节(B/s)表示。

串行通信中，衡量数据传输速率的单位是波特率，

即每秒传送的二进制数据的位数，以位/秒(bps)表示。

有时也用“位周期”Td表示传输速率，波特率是位周期的倒数

波特率= 1/Td




标准的波特率系列有：

110、300、600、1200、1800、2400、4800、9600和19200等.

# 5. 通信协议

要想保证通信成功，通信双方必须有一系列的约定，如：
作为发送方，必须知道什么时候可以发送信息，对方是否收到，
收到的内容有没有错，要不要重发、怎么通知对方结束等
作为接收方，必须知道对方是否发送信息、发的是什么、
收到的信息是否有错、如果有错怎么通知对方、怎么判断结束等

通信协议就是通信双方为了保证通信正确，

事先对数据传送控制规定的必须共同遵守的一种约定，

包括对数据格式、同步方式、传送速率、传送步骤、

检纠错方式以及控制字符定义等问题做出的统一规定。

通信协议的实现可由硬件或软件实现.

## 6. 串行通信基本方式

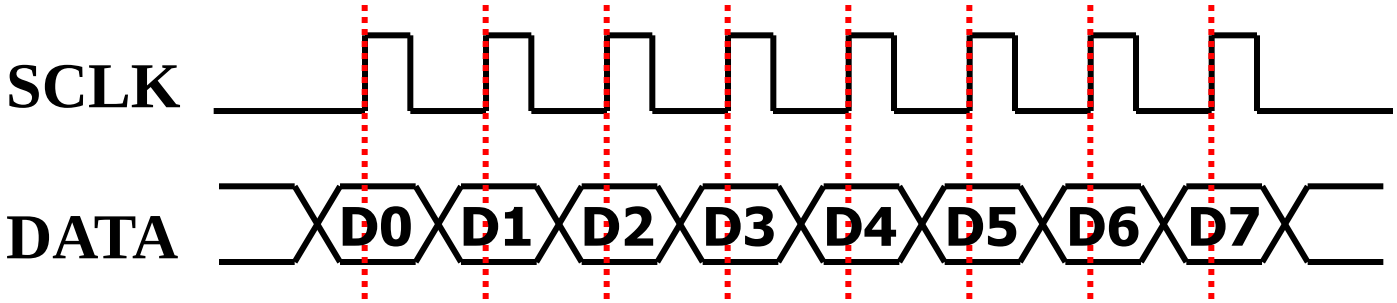
根据对数据流的分界、定时和同步方式的不同，串行通信的基本方式可以分为两种类型：

同步串行通信

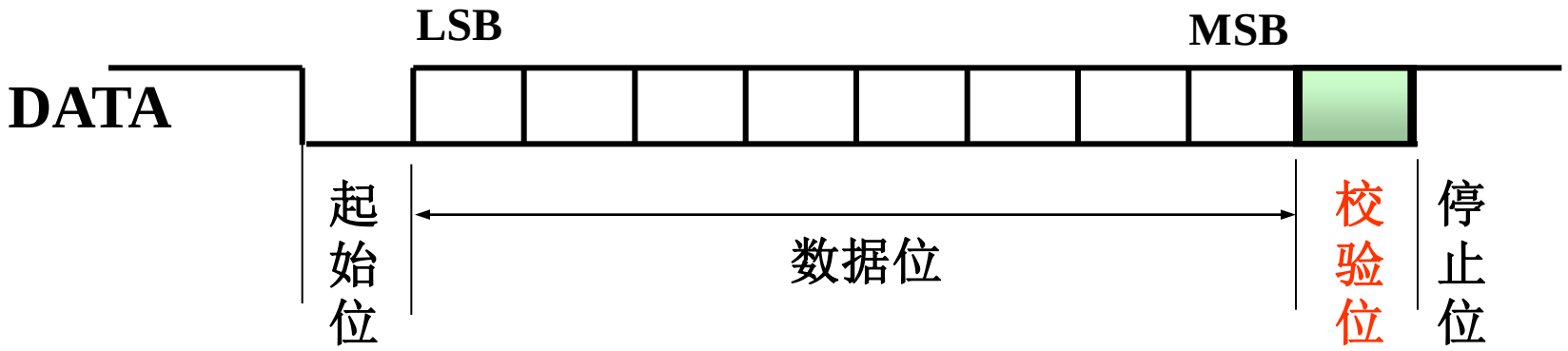
异步串行通信

# 异步串行与同步串行

**同步串行**：使用独立的同步时钟信号线来实现位同步，
用时钟控制数据的传送。




**异步串行**：不使用独立的同步时钟信号线，
位同步是由事先约定好的波特率、
并在传送的信息中设置起始位、停止位来实现



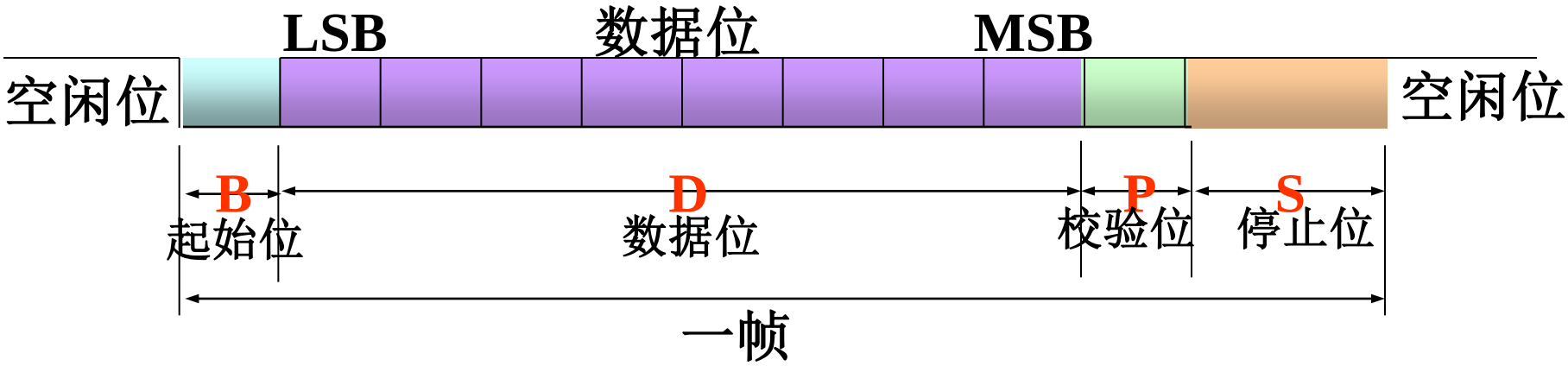
# 7. 异步串行通信

数据是一帧(Frame)一帧传送，

每一帧包含起始位、数据位、校验位、停止位，

帧与帧之间可有任意个空闲位。

异步串行数据格式为：

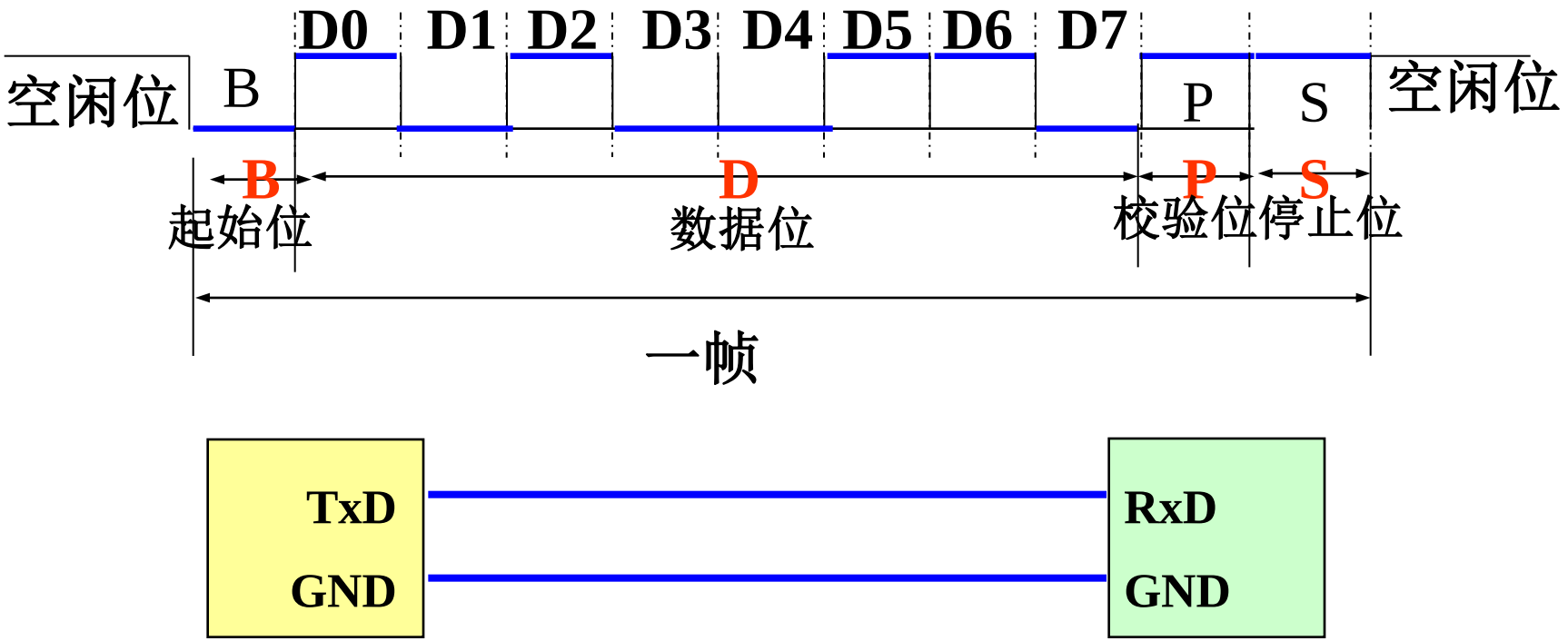



| | | |
|---|---|---|
| 起始位B | 逻辑0 | 1位 |
| 数据位D | 逻辑0或1 | 5位~8位 |
| 校验位P | 逻辑0或1 | 1位或无 |
| 停止位S | 逻辑1 | 1~2位 |
| 空闲位 | 逻辑1 | 任意数量 |

*例*

传送8位数据65H(01100101B)，先低位后高位发送，奇校验，1个停止位，

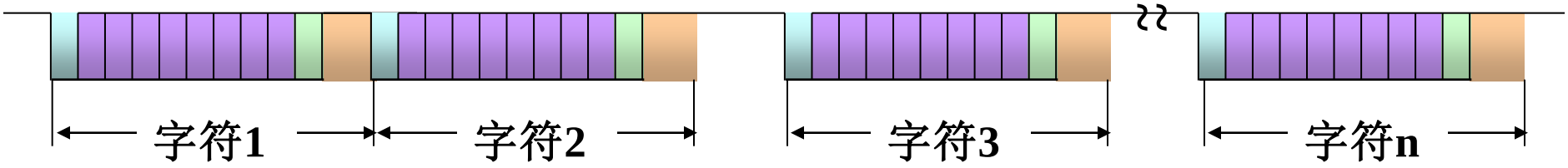
信号线上的波形为：



# 异步串行通信特点

- 异步串行通信以帧为信息单位传送，1帧包含1个字符
- 在数据格式中设置起始位和停止位来协调“同步”
- 异步串行通信中字符与字符之间通信没有严格的定时要求，每个字符作为一个独立的单位，可以随机出现在数据流中，即每个字符在数据流中出现的时间是任意的
- 异步串行通信中位与位之间有严格的定时

  一旦传送开始，收/发双方则按预先约定的传输速率，在时钟的作用下，传送字符中的 每1位
- 异步串行通信适合于发送数据不连续、传送数据量较少，或对传输率要求不高的场合。




数据块的传输

## 8. 接收/发送时钟和波特率因子

### 接收/发送时钟

在串行通信中，无论发送或接收，

都必须有时钟脉冲信号对传送的数据进行同步和定位控制，

这就需要有接收/发送时钟。

同步，就是确认通信什么时候开始

定位，就是确认每一个的数据位

# 波特率因子

为了提高串行通信的抗干扰能力，

往往用多个时钟调制1位二进制数据，

调制1位二进制数据所用的收/发时钟个数称为波特率因子，

用n表示波特率因子，则：

收/发时钟频率 = 波特率 × 波特率因子n

*例如* 波特率因子为16，则16个时钟脉冲移位1次

n可以是1, 16，32，64等

以n=16为例，

接收器以数据波特率16倍的时钟对所接收的数据进行检测：

首先正确地检测到起始位，然后逐位确定各个数据位。

具体过程如下：

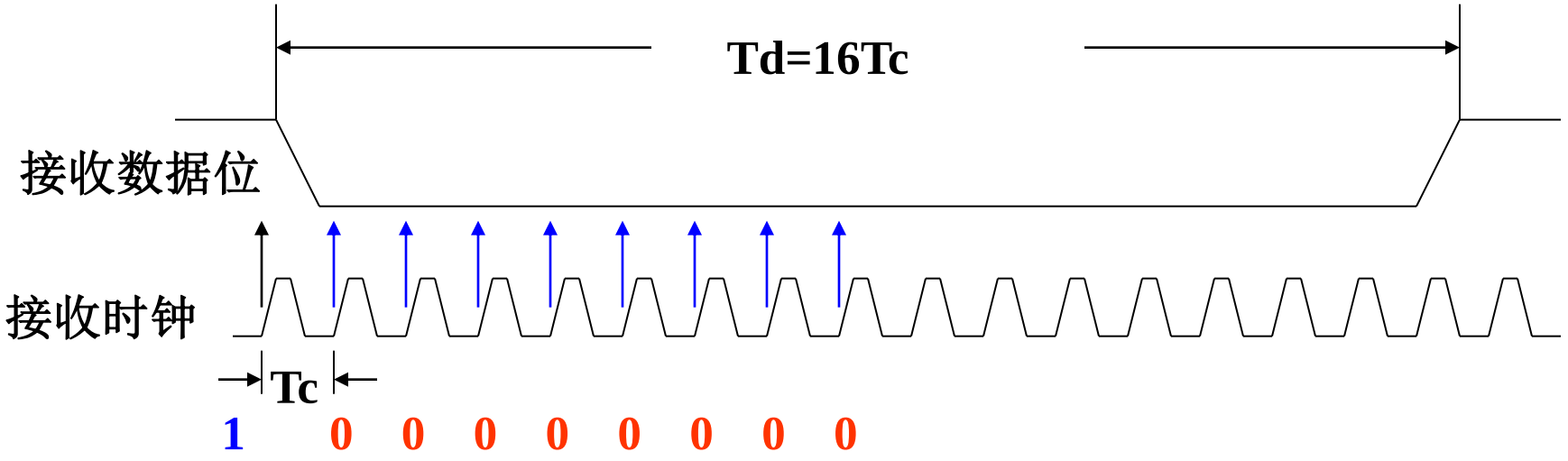
接收器在每个接收时钟的上升沿采样接收数据线，

当发现接收数据线出现低电平时，就认为是起始位的开始，

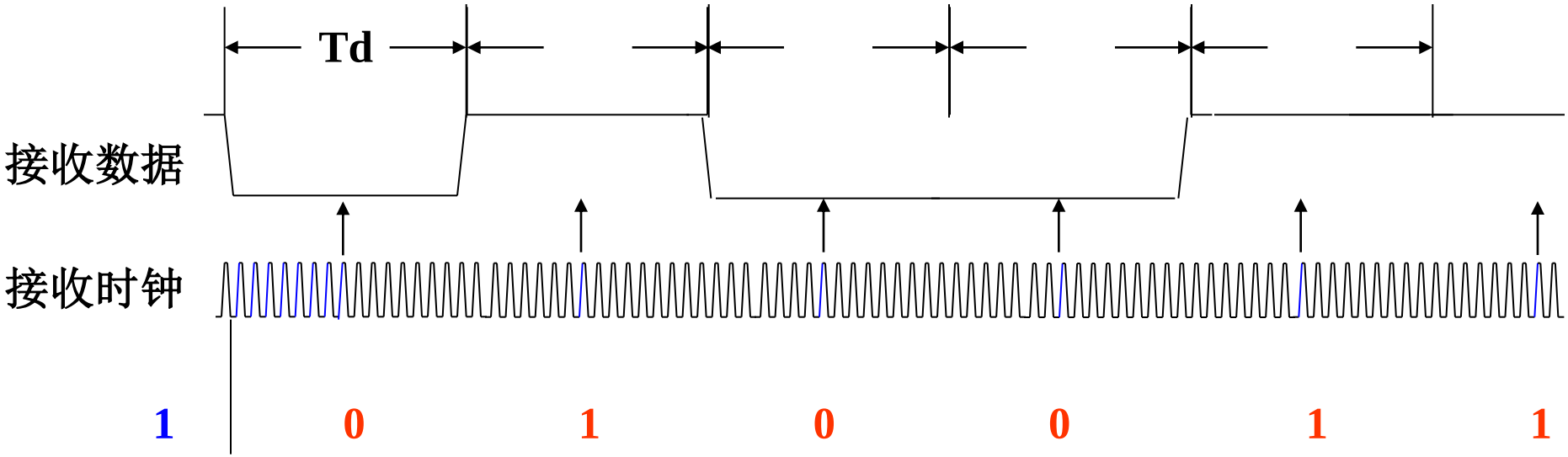
若在连续的8个时钟周期内检测到接收数据线仍保持低电平，

则确认它为起始位，而不是干扰信号。

起始位的确定



## 数据位的采样




通过这种方法，

不仅能够排除接收线上的噪声干扰，识别假起始位，

而且能够相当精确地确定起始位的中间点，

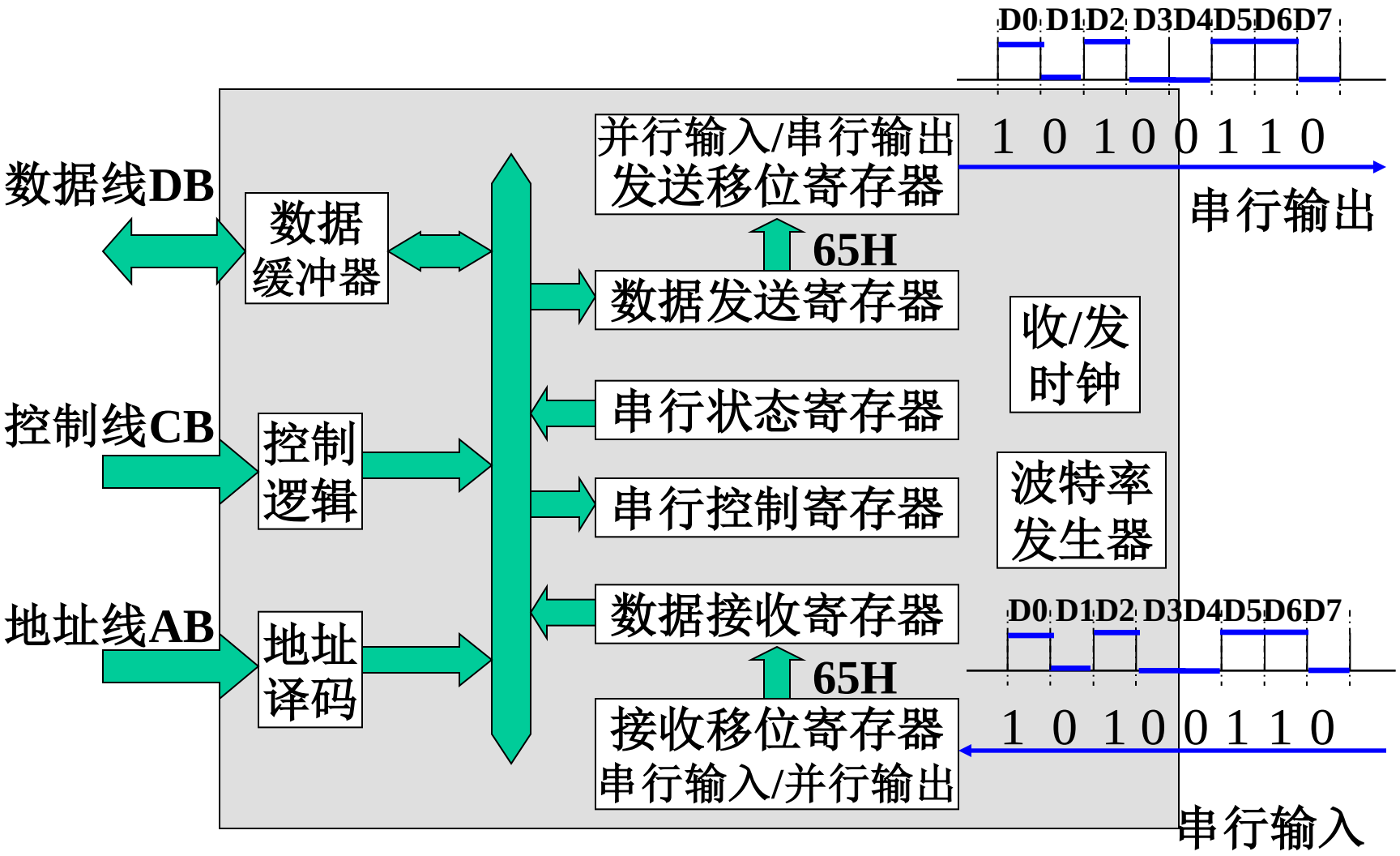
从而提供一个准确的时间基准，

从这个基准算起，每隔16个Tc采样一次数据线，作为输入数据。

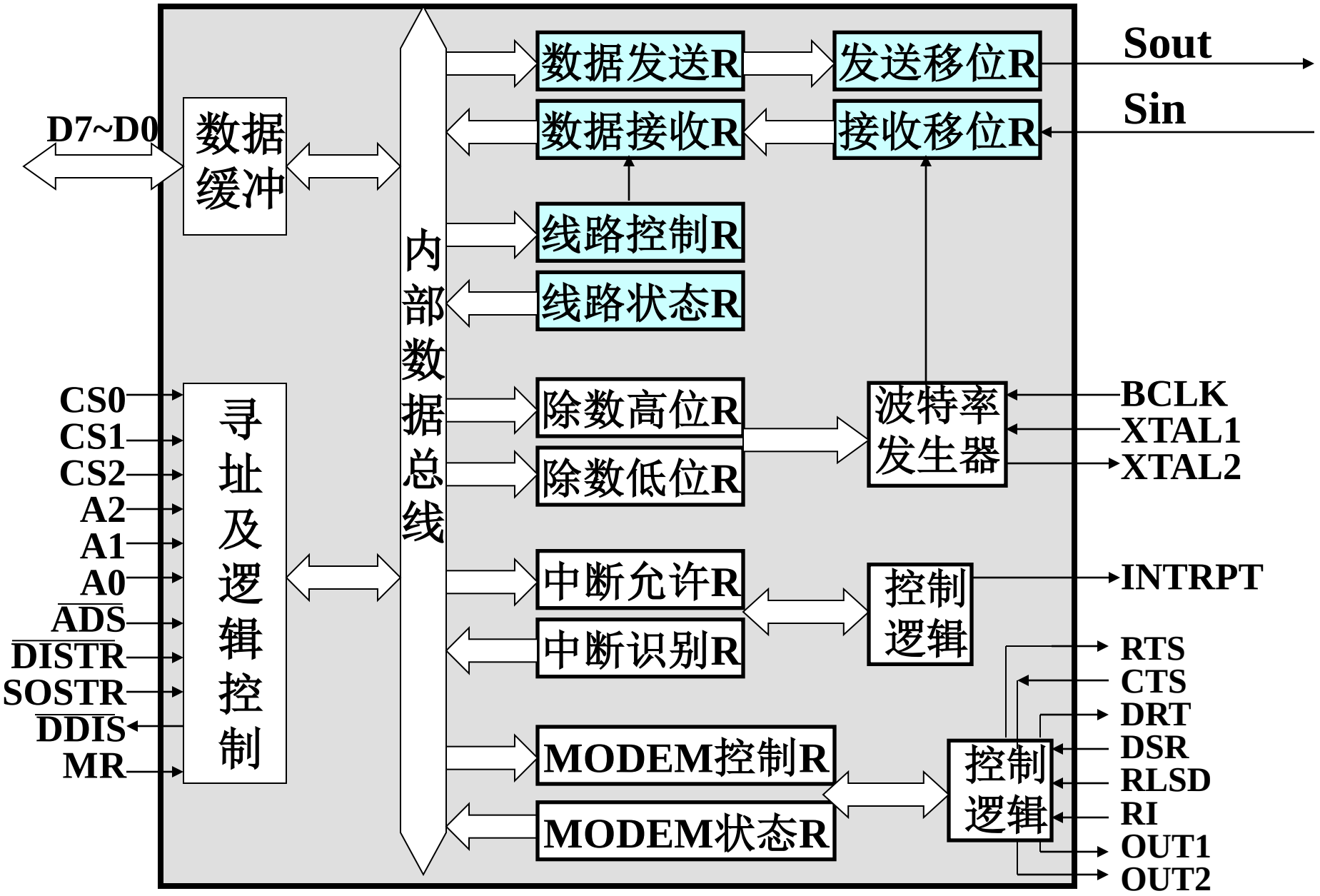
# 二、典型异步串行通信接口简介

1. 串行通信接口典型结构图
2. 串行通信有关寄存器
3. 串行发送数据过程
4. 串行接收数据过程

# 1. 串行通信接口典型结构图



# 例　PC机异步通信接口INS8250结构图



## 2. 串行通信有关寄存器

- 发送数据寄存器
- 发送移位寄存器
- 接收移位寄存器
- 接收数据寄存器
- 串行控制寄存器
- 串行状态寄存器


## 数据发送寄存器

用于存放要发送的并行数据,

如检测到发送移位寄存器为空,

自动将保存的数据传送到发送移位寄存器.

## 发送移位寄存器(并行输入/串行输出)

用于将并行数据转换成串行数据,

并通过串行输出管脚发送数据.

CPU不能直接对发送移位寄存器存取操作.


## 接收移位寄存器(串行输入/并行输出)

用于接收从串行输入管脚传入的数据，并转换成并行数据.

当接收到1字符的串行数据，就自动传送给接收数据寄存器.

CPU不能直接对接收移位寄存器存取操作.

## 接收数据寄存器

用于存放由接收移位寄存器传送来的数据,

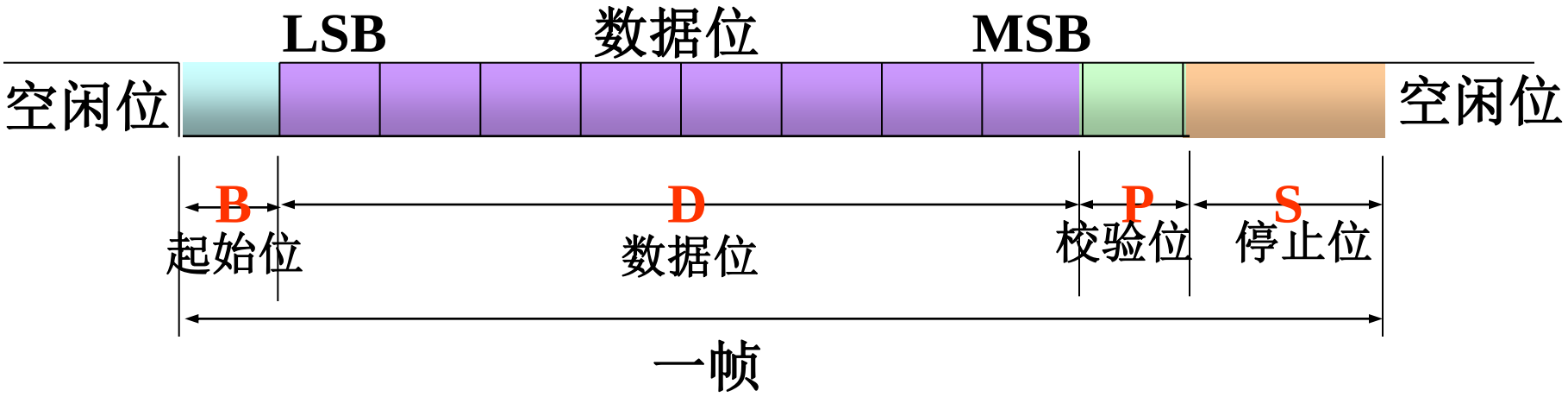
CPU通过对接收数据寄存器进行读操作，获取传入的数据.

# 串行控制寄存器

设置串行通信的数据格式，

包括波特率、停止位长度、数据位长度、奇偶校验方式的设置

异步串行数据格式为：



# 例如　某串行通信接口控制寄存器的含义

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

- **D7~D5:** 波特率设置

  000 /110bps　001/150bps　010/300bps　011/600bps

  100 /1200bps　101/2400bps　110/4800bps　111/9600bps

- **D4~D3:** 奇偶校验设置

  00/无校验　01/奇校验　10/无校验　11/偶校验

- **D2　:** 停止位长度设置

  0: 1位　1:2位

- **D1~D0:** 数据位长度设置

  10: 7位　11:8位

# 串行状态寄存器

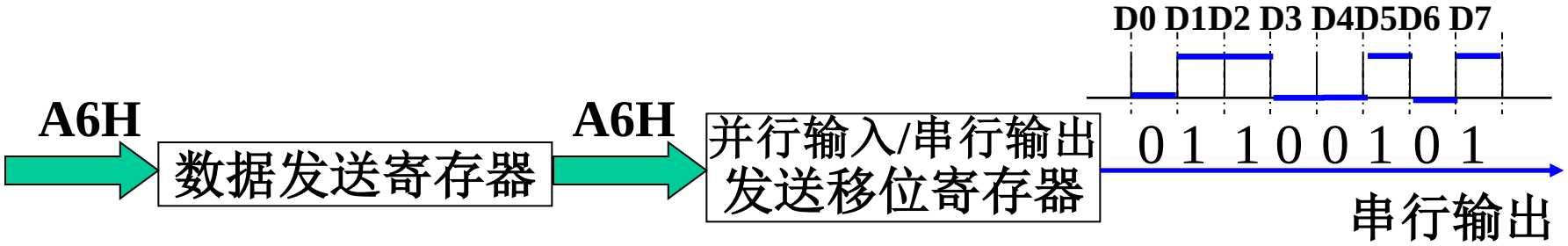
存放串行通信的状态，

包括发送数据寄存器，接收数据寄存器，通信过程的状态等.

通信口状态参数含义

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

- D6 : 1=发送数据寄存器空
- D3 : 1=帧格式错误
- D2 : 1=奇偶校验错误
- D1 : 1=溢出错误
- D0 : 1=接收数据寄存器满




串行状态寄存器

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

## 发送数据寄存器的状态

满:当发送数据寄存器的数据未传送到移位寄存器处于满状态;

空:当发送数据寄存器的数据传送到移位寄存器，

新的数据还未写入发送数据寄存器时，处于空状态

**CPU可通过检测发送数据寄存器是否为空状态，确定是否可以发送新数据.**


串行状态寄存器

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

## 接收数据寄存器的状态

满：当接收移位寄存器从串行输入管脚接收一个字节的数据，并转换成并行数据自动存放到接收数据寄存器后，接收数据寄存器处于满状态.

空：当接收数据寄存器的数据被取走，但新的数据未传入，数据接收寄存器处于空状态，表示没有新的数据到来.

CPU可通过检测接收数据寄存器是否为满状态，确定是否可以接收新数据.

# 接收/发送通信过程状态

## 奇偶校验错误

通信线上的噪音引起某些数据位的改变,
使接收数据位的奇偶数与设定的奇偶校验方式不一致,
从而产生奇偶校验错误。

*例* 传送方发送8位数据65H(01100101B)，奇校验，1个停止位，
数据格式为 **0101001101**1 (含校验位在内1的个数5为奇数)

如果接收方按同样的通讯协议接收数据,
假如收到的数据为 **0101101101**1
由于收到1的个数为6, 为偶数，产生奇偶校验错误。

(注意最后1位“1”是停止位，不在检验数据之内)

串行状态寄存器

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|

## 溢出错误

当上一个存在接收数据寄存器的数据还没有被CPU取走，
又有字符传送到接收数据寄存器时，
产生接收数据寄存器溢出错误。

## 帧格式错误

当接收的数据没有停止位时，产生帧格式错误，
这种错误可能是由于通信线上的噪音引起停止位的丢失，
或是由于接收方和发送方通信协议初始化不匹配引起。

# 3. 串行发送数据过程

CPU采用查询方式检测到发送数据寄存器空状态后，

将待发送的数据写入发送数据寄存器中；

当发送移位寄存器空时，

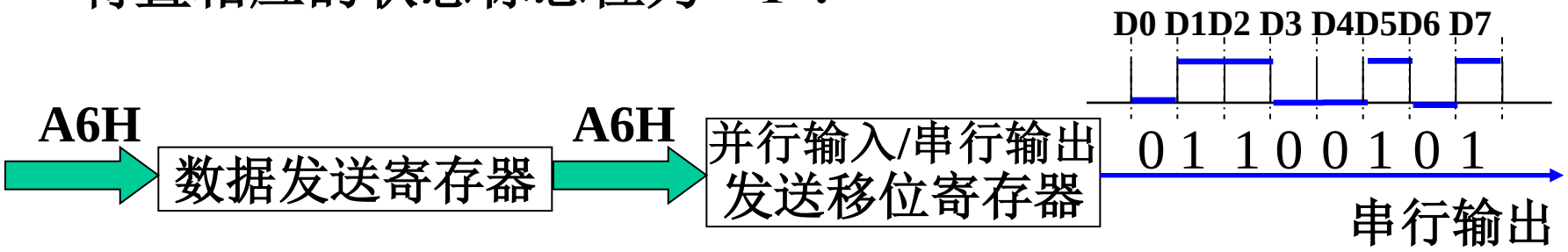
数据发送寄存器会自动将数据传送到移位寄存器，

发送控制逻辑将自动按规定的数据格式构成信息帧，

然后逐位由串行输出管脚输出；

当发送数据寄存器空时，

将置相应的状态标志位为“1”.



# 4. 串行接收数据过程

串行的位信息由串行输入管脚输入，

接收侧监测通信线路，如果检测出一个起始位，就进行内部同步，

开始在接收时钟控制下采样串行输入管脚上的数据，

移位寄存器将接收的串行数据，转换成并行数据，

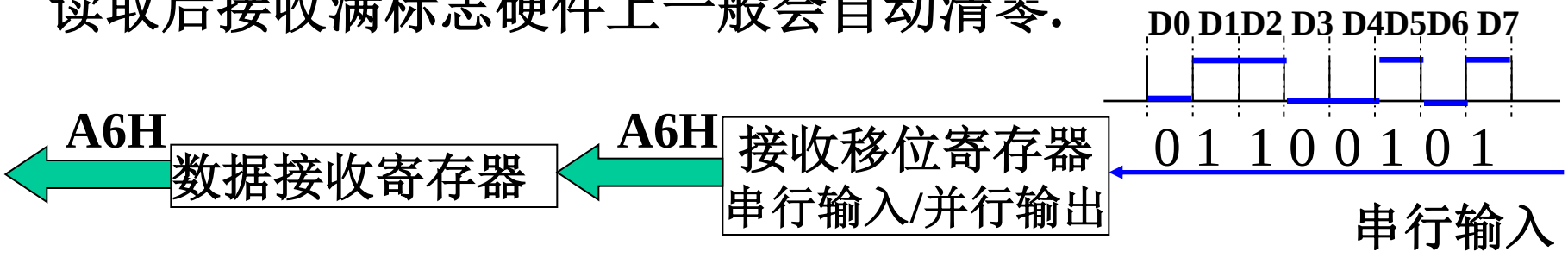
自动存放在接收数据寄存器中，

并置数据接收寄存器满标志位为“1”；

CPU通过查询方式检测到数据接收寄存器满状态后，

可读取接收数据寄存器中的数据，将数据取走；

读取后接收满标志硬件上一般会自动清零.



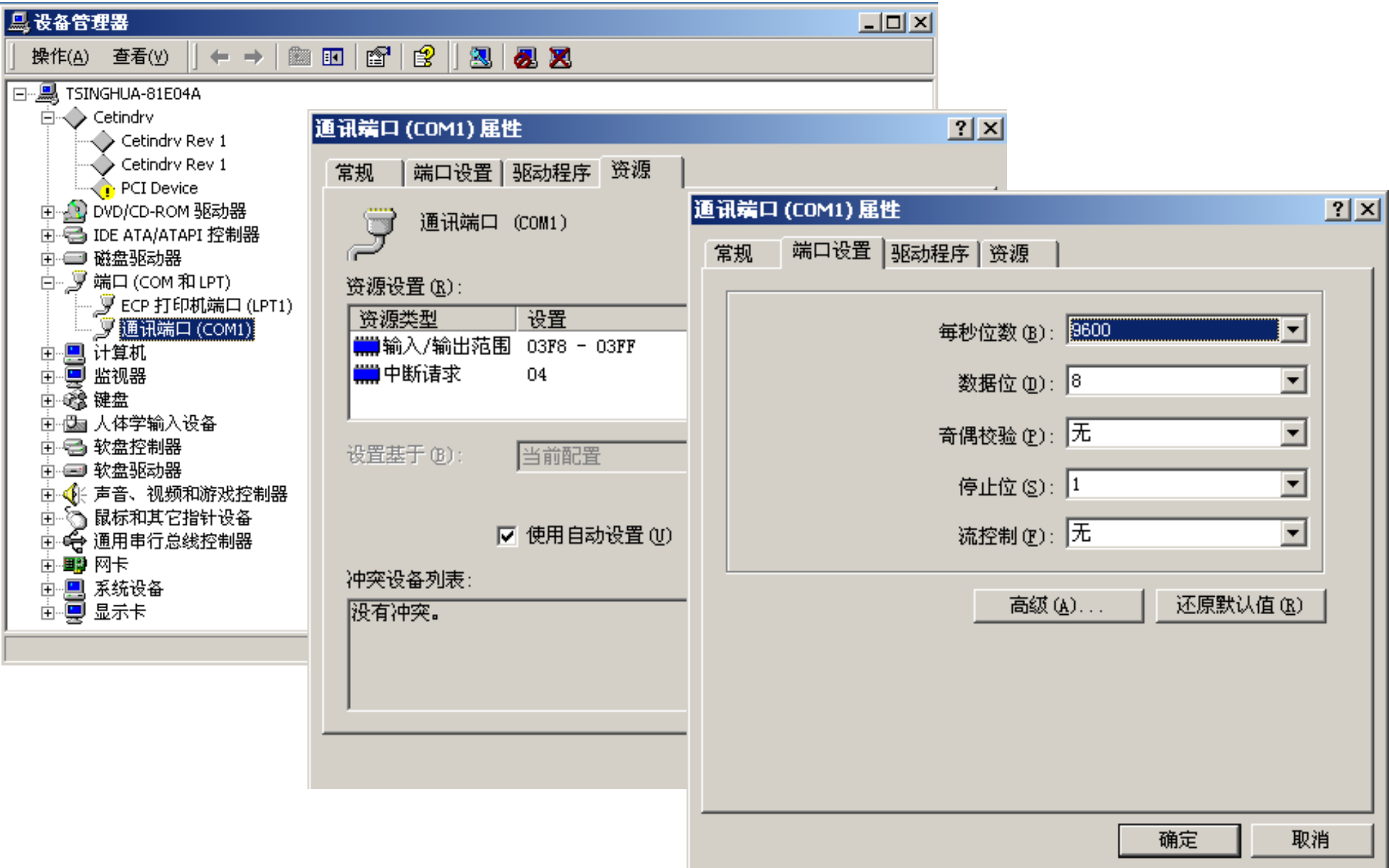
# 三、PC机的标准串口

1. 早期的PC机一般有1或2个异步串行通信接口，
   组装在主板上，称为COM1和COM2
2. 使用的串行通信接口芯片是INS8250,
   或与之兼容的串口芯片(如16550等).
   8250是一种可编程的异步串行通信接口芯片，
   支持异步通信协议，可通过编程改变传送数据的波特率，
   提供与MODEM连接的联络信号，实现远程通信。
3. 两串口的结构相同，只是占用的系统资源不同，
   COM1：3F8~3FFh, 中断IRQ4
   COM2：2F8~2FFh, 中断IRQ3
4. 现在的PC机一般配置有USB口，未配置串口，
   可利用USB/串口转换器，转换出一个串口

串口1　　并口

# 在Windows2000下查看串口硬件资源

## 我的电脑/属性/硬件/设备管理/端口/通信端口COM1



# 4. 串行通信接口标准RS-232

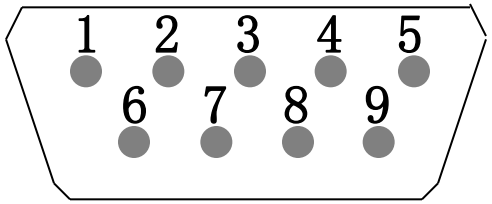
- 在串行通信中，设备之间的连接要符合接口标准
- 计算机通信中使用最普遍的是：

  美国电子工业协会(EIA)推荐使用的RS-232C标准

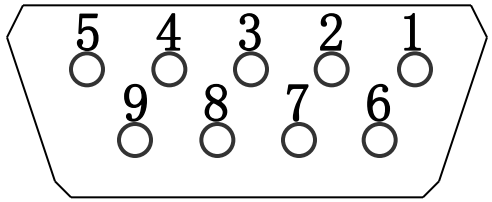
  EIA——Eliectronic Industries Association
  美国电子工业协会

  RS——Recommend Standard　推荐标准

  C表示第三版本，之前有过A、B版
- PC机上的COM1、COM2连接器，符合RS-232C接口




针型DB-9M




孔型DB-9F

| 引脚号 | 符号 | 信号名称 | 方向 |
|---|---|---|---|
| 1 | DCD | 载波检测 | 输入 |
| 2 | RxD | 接收数据 | 输入 |
| 3 | TxD | 发送数据 | 输出 |
| 4 | DTR | 数据终端就绪 | 输出 |
| 5 | GND | 信号地 | |
| 6 | DSR | 数据设备就绪 | 输入 |
| 7 | RTS | 请求发送 | 输出 |
| 8 | CTS | 允许发送 | 输入 |
| 9 | RI | 振铃指示 | 输出 |

**RS-232C采用负逻辑，TTL采用正逻辑，**

**RS-232C信号电平与TTL不兼容**

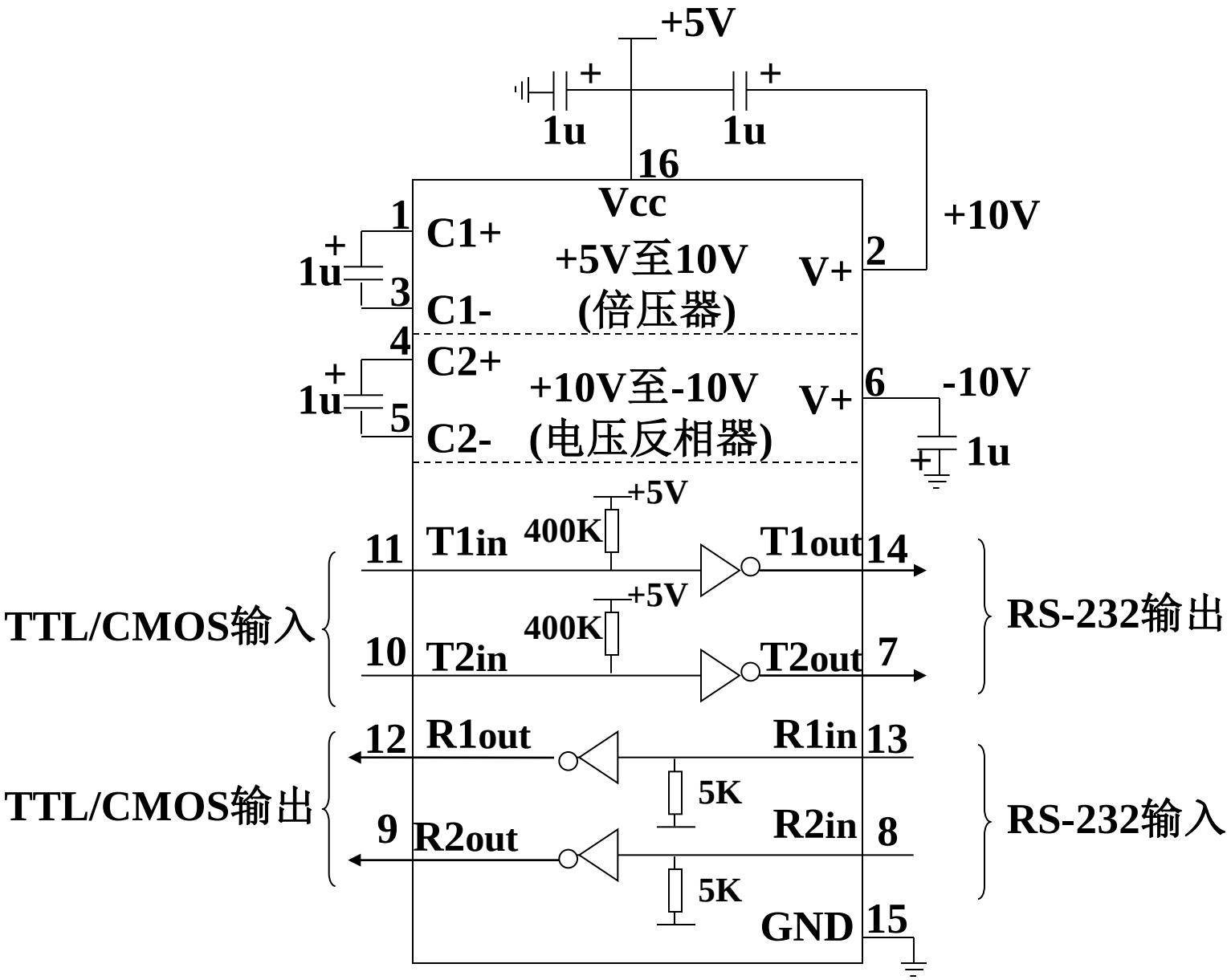
| | RS-232C标准 | TTL标准 |
|---|---|---|
| 逻辑0 | +5 ~ +15V | 0.3 ~ 1.0V |
| 逻辑1 | -5 ~ -15V | 2.5 ~ 5.0V |

**一般串行接口芯片如8250、8251、**

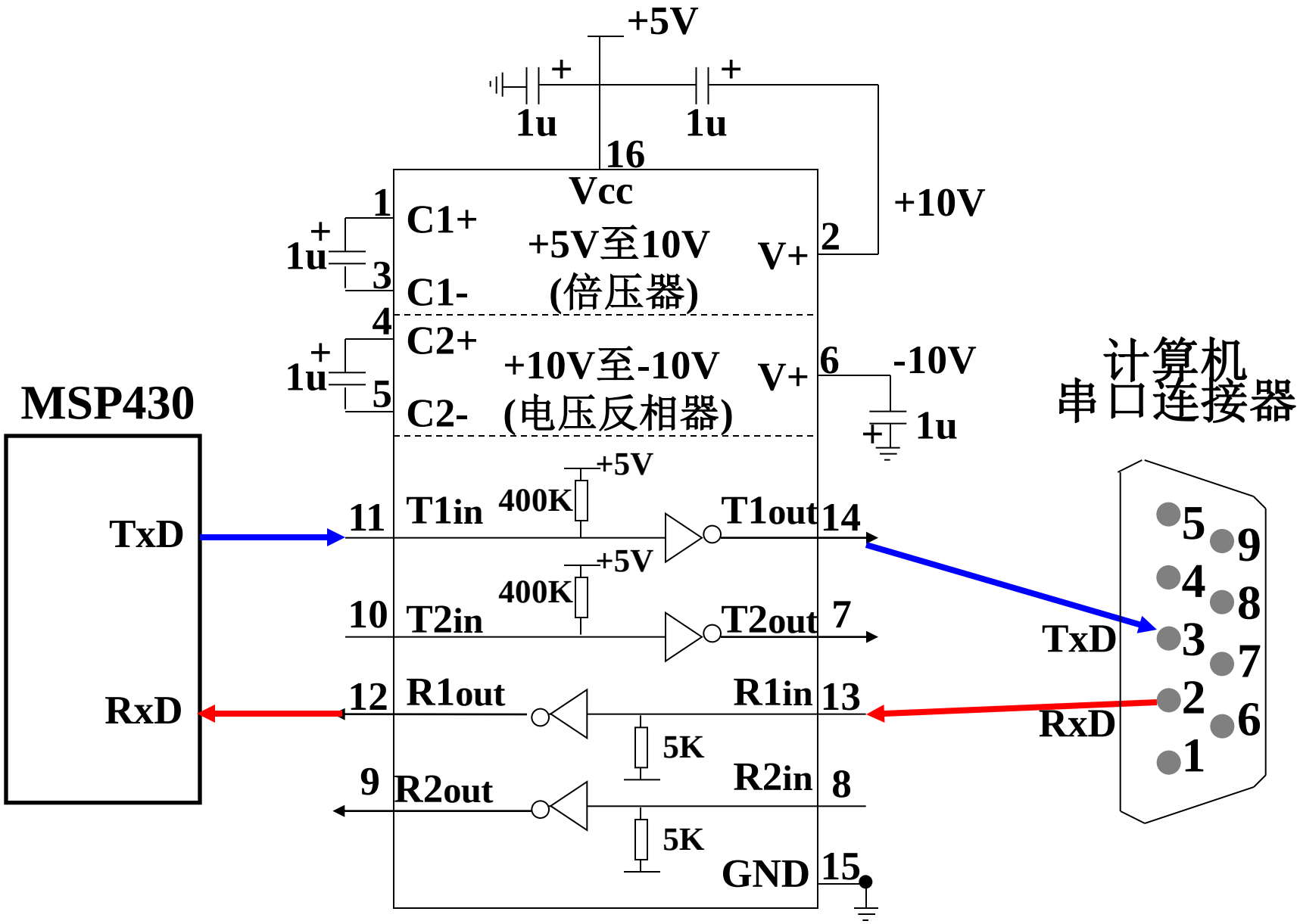
**单片机内（如msp430、串行接口均使用TTL电平，**

**使用电平转换电路才能与RS-232C连接器连接。**

# MAX232电平转换器



# 利用MAX232电平转换器进行电平转换



- 两台PC机或设备进行近距离通信时，

  可直接将串口的信号线对接。

- 当进行远距离通信时，

  要是用调制解调器(MODEM)连接到电话线上。

  因为RS-232标准串行接口输出的是电压信号，

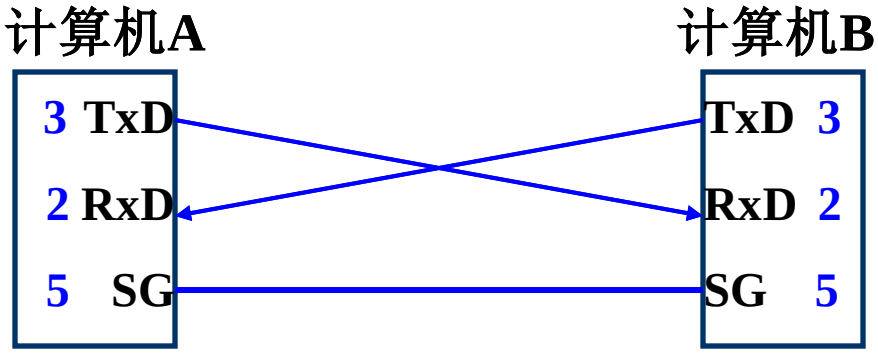
  不能直接接到电话线上，

  调制解调器把代表逻辑1和逻辑0的电压信号转换成

  能在电话线上传输的不同频率的信号。

  电话线另一端的调制解调器把不同频率的信号转换成接口要求的电压信号。

# 无Modem的最简单连接




孔-孔的串行通信线

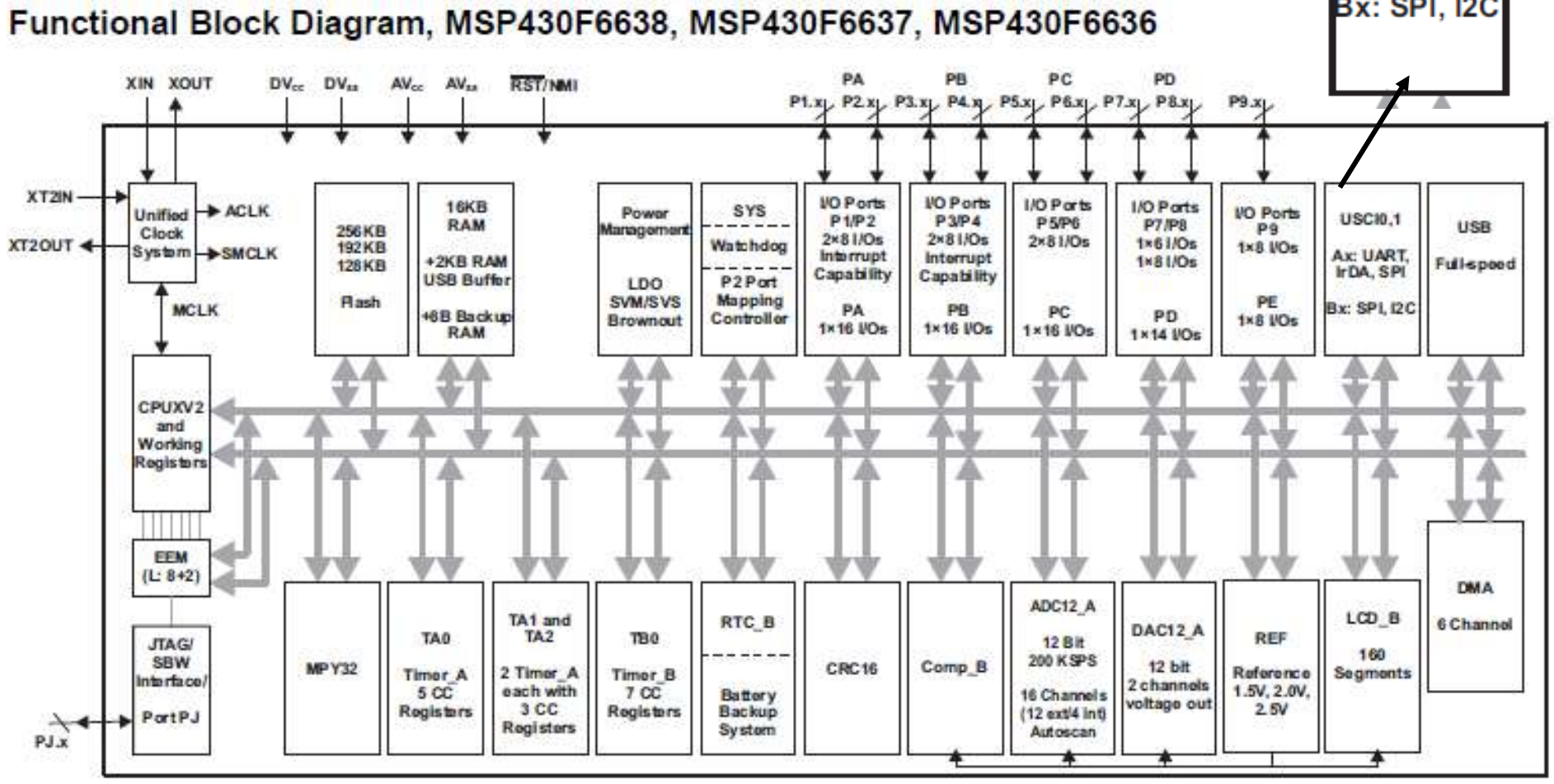
# 第2节　MSP430串行通信模块USCI

一、MSP430 USCI 模块概述

二、MSP430F6638 异步串行方式编程结构

三、利用MSP430F6638 异步串行方式通信

# 一、 MSP430 USCI 模块概述



## Universal Serial Communication Interface (USCI) Overview

The USCI modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on which devices.

USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud-rate detection for LIN communications
- SPI mode

USCI_Bx modules support:

- $I^2C$ mode
- SPI mode

**msp6638内含两个串口：USCI_A1, USCI_B1**

# msp430F6638

# 与USCI的串行通信有关引脚

# The universal serial communication interface (USCI)

## 串口UCA1

P8.1/UCA1CLK

P8.2/UCA1TXD/ UCA1SIMO

P8.3/UCA1RXD/UCA1SOMI

P8.4/UCA1STE

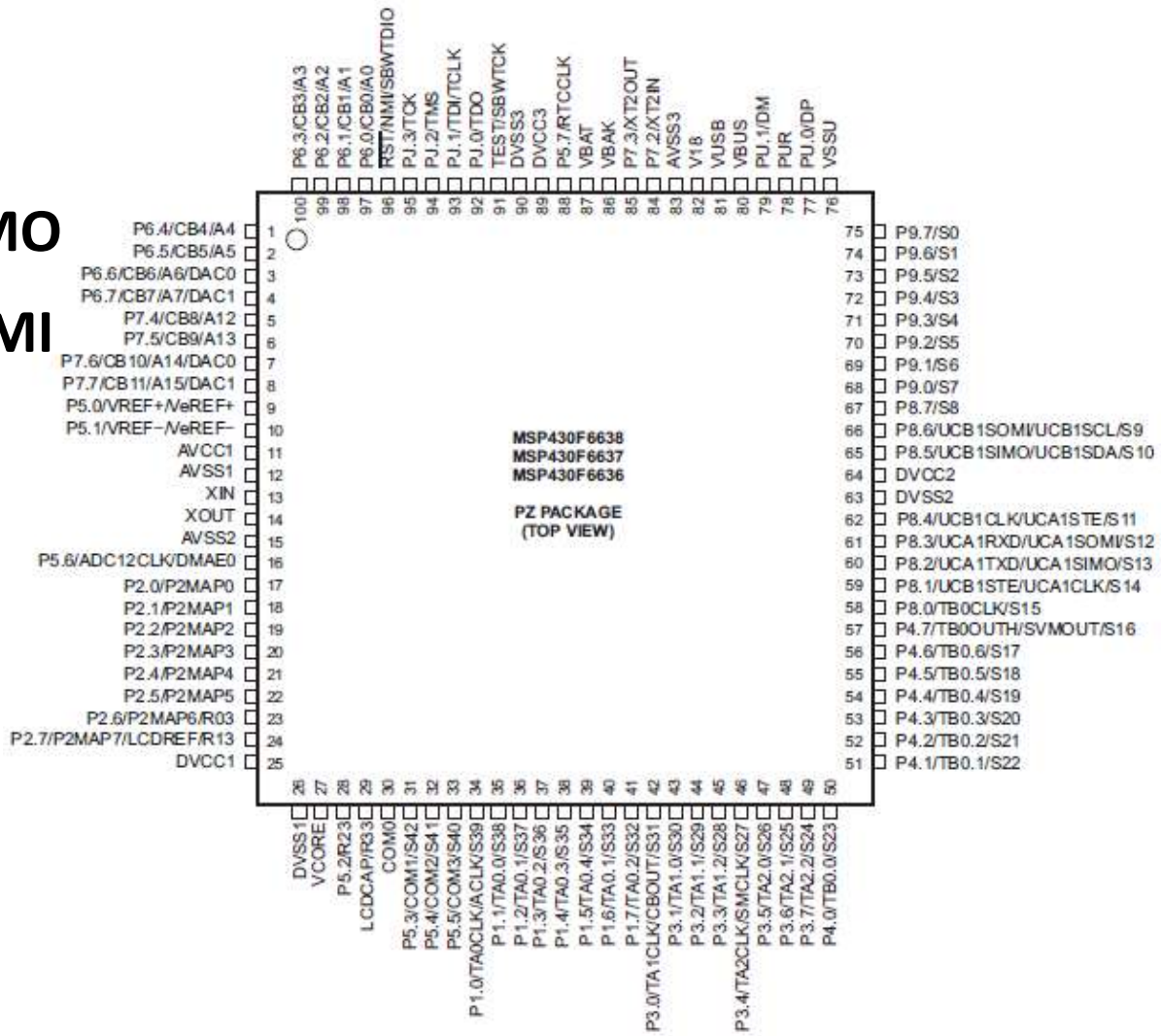
## 串口UCB1

P8.1/UCB1STE

P8.4/UCB1CLK

P8.5/UCB1SIMO

P8.6/UCB1SOMI



MSP430F6xx单片机的USCI模块有4种串行通信模式：

- UART普通模式(UART Mode)
- 线路模式(Idle-Line Multiprocessor Format)(暂缺)
- 地址模式(Address-Bit Multiprocessor Format)(暂缺)
- 自适应波特率模式(*Automatic Baud-Rate Detection*)(暂缺)

## 32.2 USCI Introduction – UART Mode

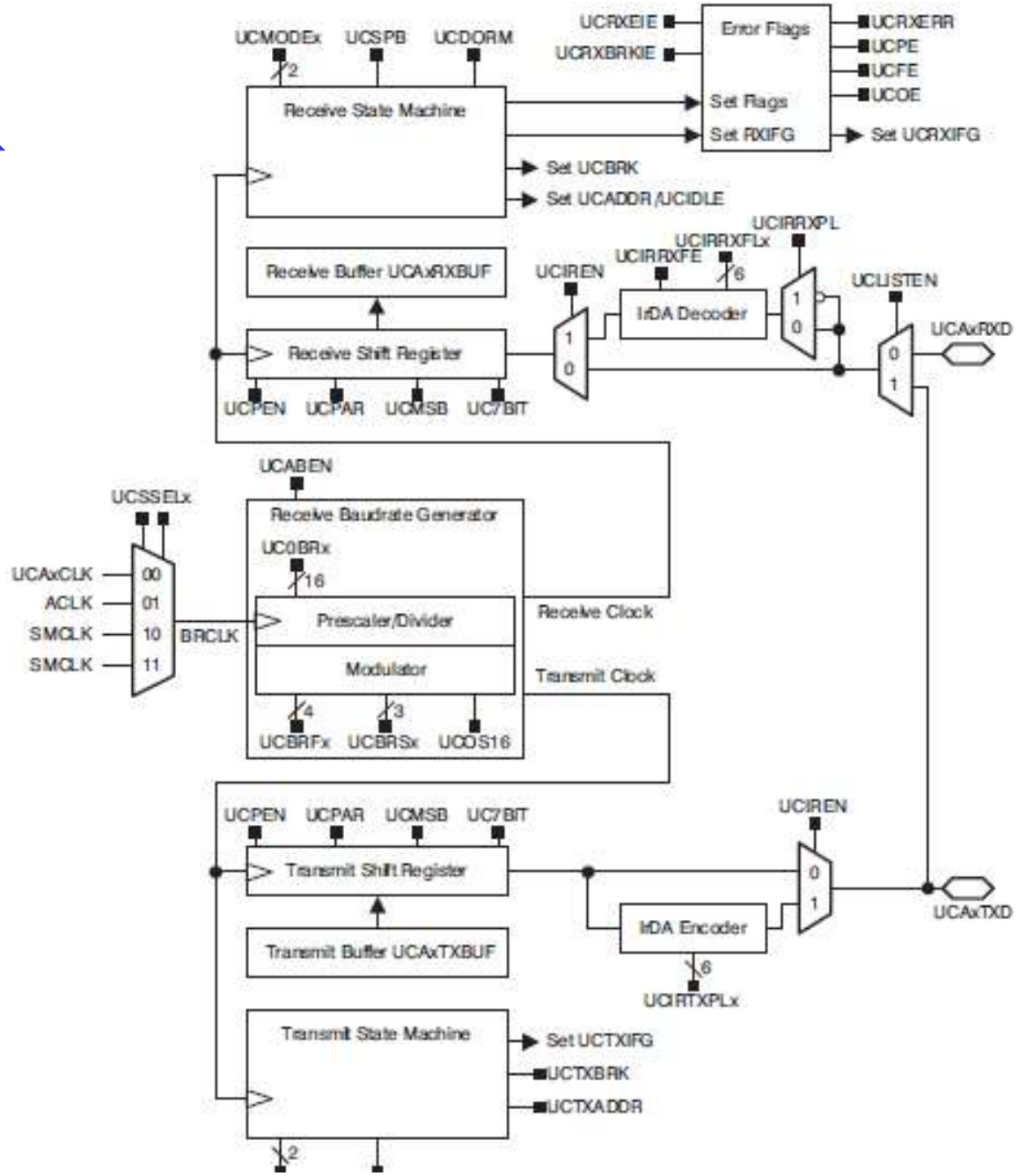
In asynchronous mode, the USCI_Ax modules connect the device to an external system via two external pins, UCAxRXD and UCAxTXD. UART mode is selected when the UCSYNC bit is cleared.

UART mode features include:

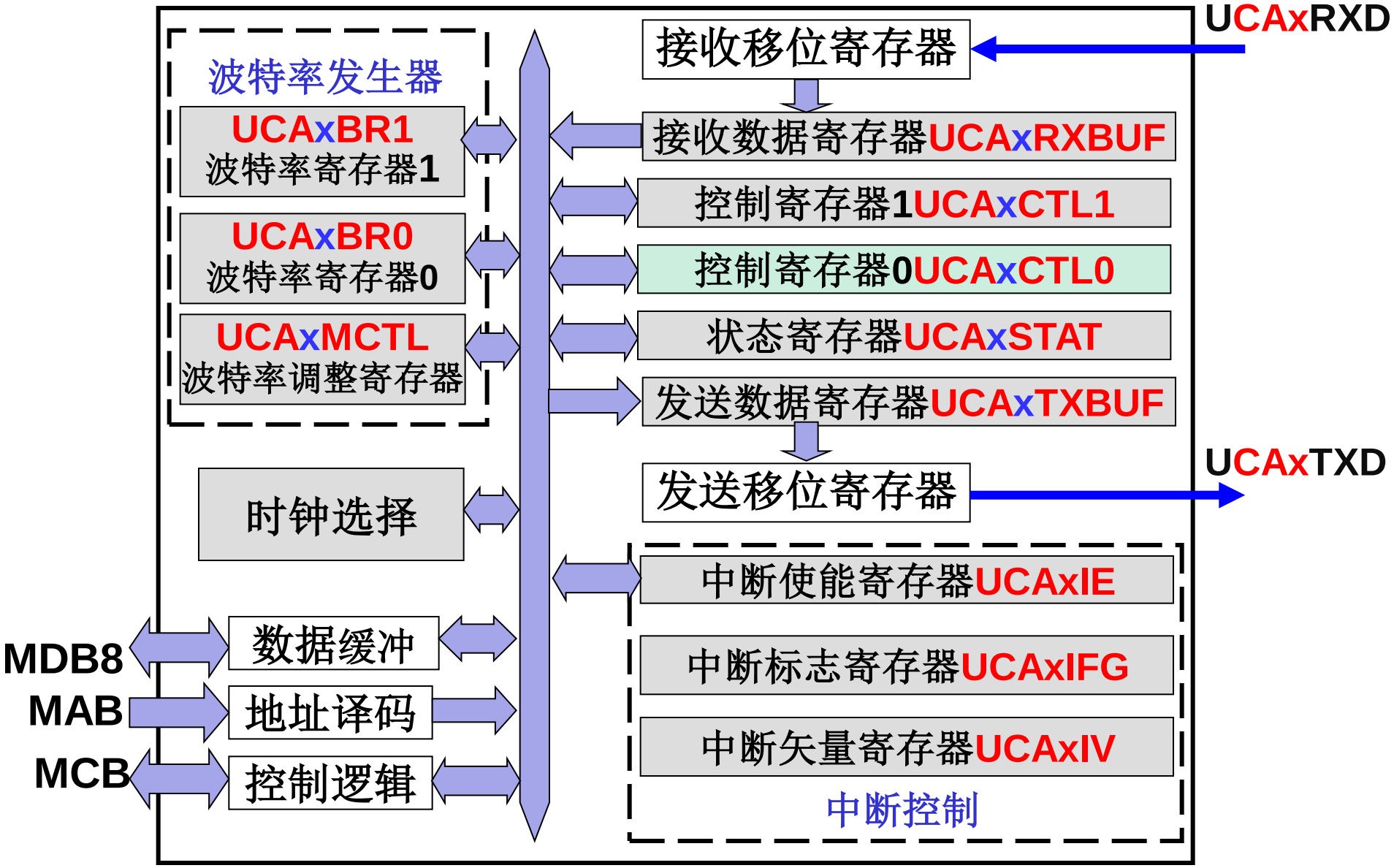
- 7- or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first or MSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud-rate support
- Status flags for error detection and suppression
- Status flags for address detection
- Independent interrupt capability for receive and transmit

# MSP430F6XX

## 异步串行方式下的结构图



# 二、 MSP430F6xx USCI异步串行方式的编程结构




小写x=0或1，分别表示串口0和串口1

红外收发(暂缺)

# MSP430F6xx UCA串行通信有关的寄存器

| 寄存器符号 | 寄存器名称 | 可用字操作 |
|---|---|---|
| UCAxCTL1 | 控制寄存器 | UCAxCTLW0 |
| UCAxCTL0 | 控制寄存器 | |
| UCAxSTAT | 状态寄存器 | |
| UCAxRXBUF | 接收数据缓冲寄存器 | |
| UCAxTXBUF | 发送数据缓冲寄存器 | |
| UCAxBR1 | 波特率寄存器1 | UCAxBRW |
| UCAxBR0 | 波特率寄存器0 | |
| UCAxMCTL | 波特率调整寄存器 | |
| UCAxIE | 中断使能寄存器 | UCAxICTL |
| UCAxIFG | 中断标志寄存器 | |
| UCAxIV | 中断矢量寄存器 | |

红外收发控制(暂缺)

# msp430F663x 串行通信有关的寄存器

| Register | Short Form | Register Type | Register Access | Address Offset | Initial State |
|---|---|---|---|---|---|
| USCI_Ax Control Word 0 | UCAxCTLW0 | Read/write | Word | 00h | 0001h |
| USCI_Ax Control 1 | UCAxCTL1 | Read/write | Byte | 00h | 01h |
| USCI_Ax Control 0 | UCAxCTL0 | Read/write | Byte | 01h | 00h |
| USCI_Ax Baud Rate Control Word | UCAxBRW | Read/write | Word | 06h | 0000h |
| USCI_Ax Baud Rate Control 0 | UCAxBR0 | Read/write | Byte | 06h | 00h |
| USCI_Ax Baud Rate Control 1 | UCAxBR1 | Read/write | Byte | 07h | 00h |
| USCI_Ax Modulation Control | UCAxMCTL | Read/write | Byte | 08h | 00h |
| Reserved - reads zero | | Read | Byte | 09h | 00h |
| USCI_Ax Status | UCAxSTAT | Read/write | Byte | 0Ah | 00h |
| Reserved - reads zero | | Read | Byte | 0Bh | 00h |
| USCI_Ax Receive Buffer | UCAxRXBUF | Read/write | Byte | 0Ch | 00h |
| Reserved - reads zero | | Read | Byte | 0Dh | 00h |
| USCI_Ax Transmit Buffer | UCAxTXBUF | Read/write | Byte | 0Eh | 00h |
| Reserved - reads zero | | Read | Byte | 0Fh | 00h |
| USCI_Ax Auto Baud Rate Control | UCAxABCTL | Read/write | Byte | 10h | 00h |
| Reserved - reads zero | | Read | Byte | 11h | 00h |
| USCI_Ax IrDA Control | UCAxIRCTL | Read/write | Word | 12h | 0000h |
| USCI_Ax IrDA Transmit Control | UCAxIRTCTL | Read/write | Byte | 12h | 00h |
| USCI_Ax IrDA Receive Control | UCAxIRRCTL | Read/write | Byte | 13h | 00h |
| USCI_Ax Interrupt Control | UCAxICTL | Read/write | Word | 1Ch | 0000h |
| USCI_Ax Interrupt Enable | UCAxIE | Read/write | Byte | 1Ch | 00h |
| USCI_Ax Interrupt Flag | UCAxIFG | Read/write | Byte | 1Dh | 00h |
| USCI_Ax Interrupt Vector | UCAxIV | Read | Word | 1Eh | 0000h |

## ● UCAxTXBUF发送数据寄存器

用于存放要发送的并行数据，

如检测到发送移位寄存器为空，

自动将保存的数据传送到发送移位寄存器，

并通过UTXDx引脚按设定的异步串行数据帧格式发出.

*例* 将字节类型变量Tdata的内容通过UCAxTXBUF 发送出去

```
UCAxTXBUF=Tdata;
```

## ● UCAxRXBUF接收数据寄存器

用于存放接收移位寄存器接收URXDx引脚传送来的数据，

CPU通过对接收数据寄存器进行读操作，获取传入的数据.

*例* 将UCAxRXBUF接收的数据保存到字节类型变量Rdata中

```
Rdata=UCAxRXBUF;
```

# UCAxCTL0串口控制寄存器0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCPENA | UCPAR | UCMSB | UC7BIT | UCSPB | UCMODEx | | UCSYNC |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-1 |

校验允许位
0: 禁止
1:允许

奇偶校验位
0: 奇校验
1: 偶校验

移动发向
0:先低位
1:先高位

数据位长度
0: 8位
1: 7位

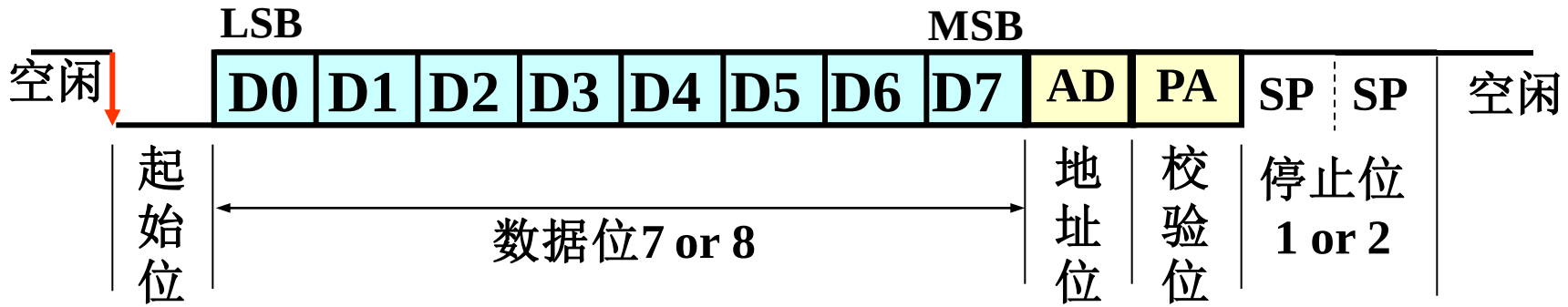
停止位长度
0: 1位
1: 2位

多机模式控制
00: 普通异步方式
01: 多处理器方式
10: 地址方式
11： 异步自适应波特率

串行方式
0: 异步
1: 同步

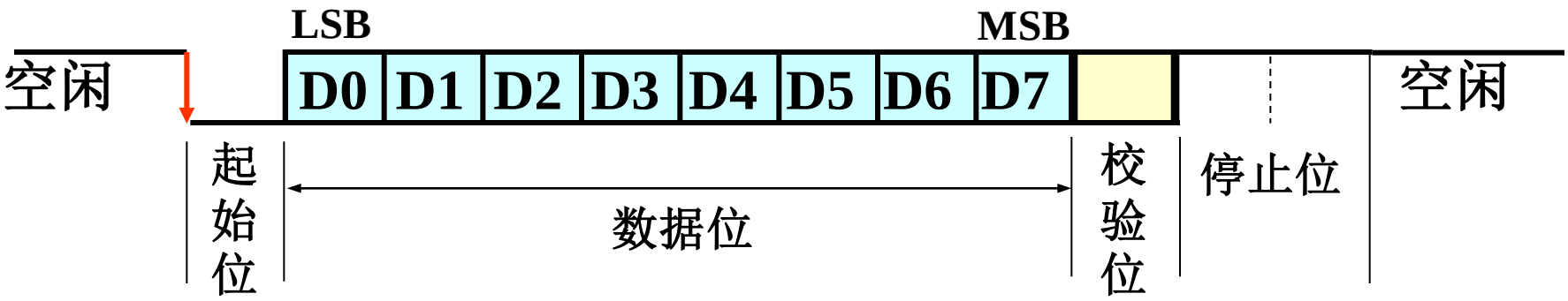
UCAxCTL0复位设置：无校验，1停止位，先发低位，8数据位，普通异步方式

## 异步串行通信数据格式



## *例* 编程设置串口1异步串行数据格式为：

异步方式，数据8位、偶校验、 2位停止位、无地址位格式

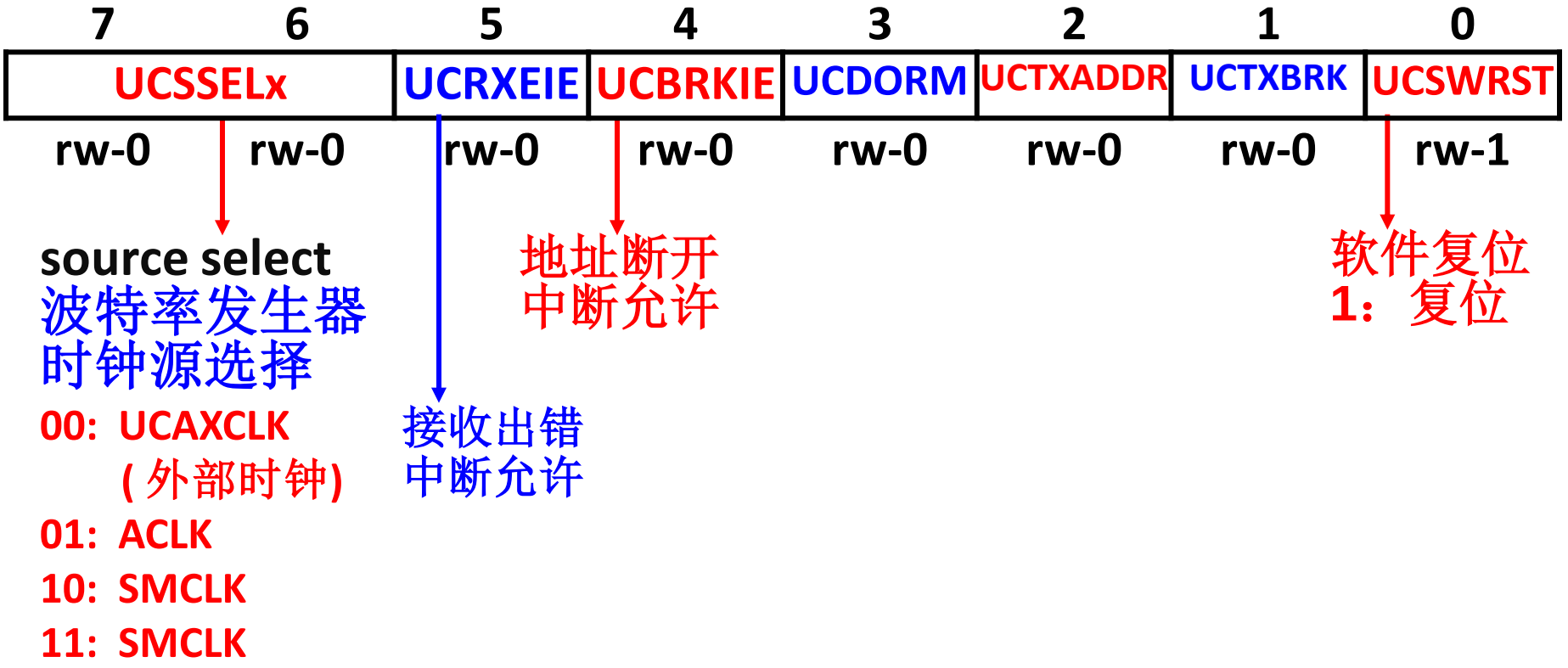



**UxCTL**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCPENA | UCPAR | UCMSB | UC7BIT | UCSPB | UCMODEx | | UCSYNC |
| 复位值 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 欲设置值 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

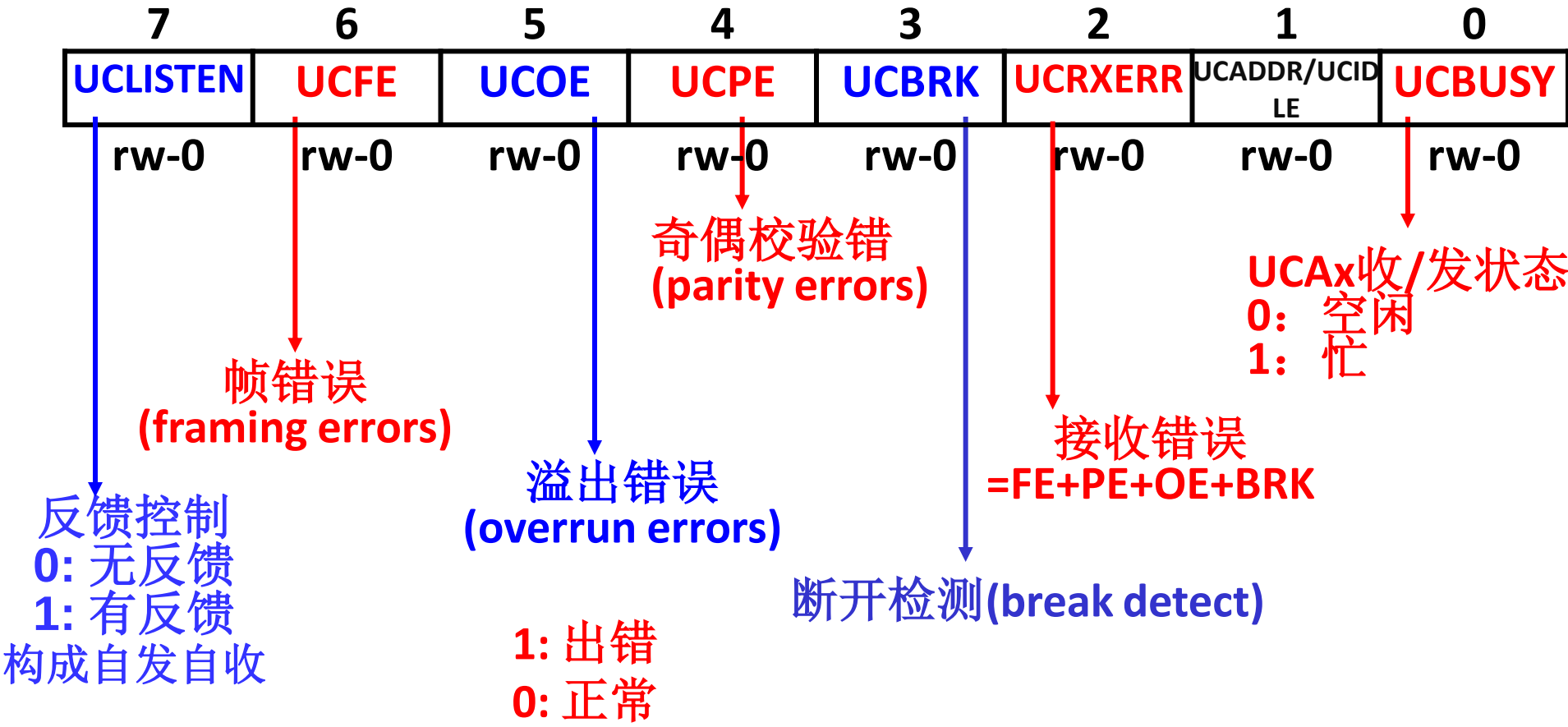
```
UCA1CTL0 |= UCPENA+UCPAR+UCSPB;
```

# UCAxCTL1串口控制寄存器1




URXEIE=
- 0: 若接收到错误字符，则不存入 UxRXBUF，并且不置位 URXIFGx
- 1: 接收到的字符无论错对，都存入 UCAxRXBUF，并置位 UCRXIFG

# ● UCAxSTAT 串口状态寄存器




读**UCAxRXBUF**操作将清零**FE,PE,OE,BRK,RXERR**等标志位

因此错误标志位的判断应在读**UxRXBUF**操作前进行。

错误标志位也可由软件清零。

## ● UCAxIE UCAx中断使能寄存器

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 保留 | | | | | | UCTXIE | UCRXIE |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | rw-0 | rw-0 |

UCTXIE: 1=发送中断允许
UCRXIE: 1=接收中断允许

## ● UCAxIFG UCAx中断标志寄存器

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 保留 | | | | | | UCTXIFG | UCRXIFG |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | rw-0 | rw-0 |

UCTXIFG: 1=当发送寄存器UCAxTXBUF空时置位
UCRXIFG: 1=当接收寄存器UCAxRXBUF满时置位

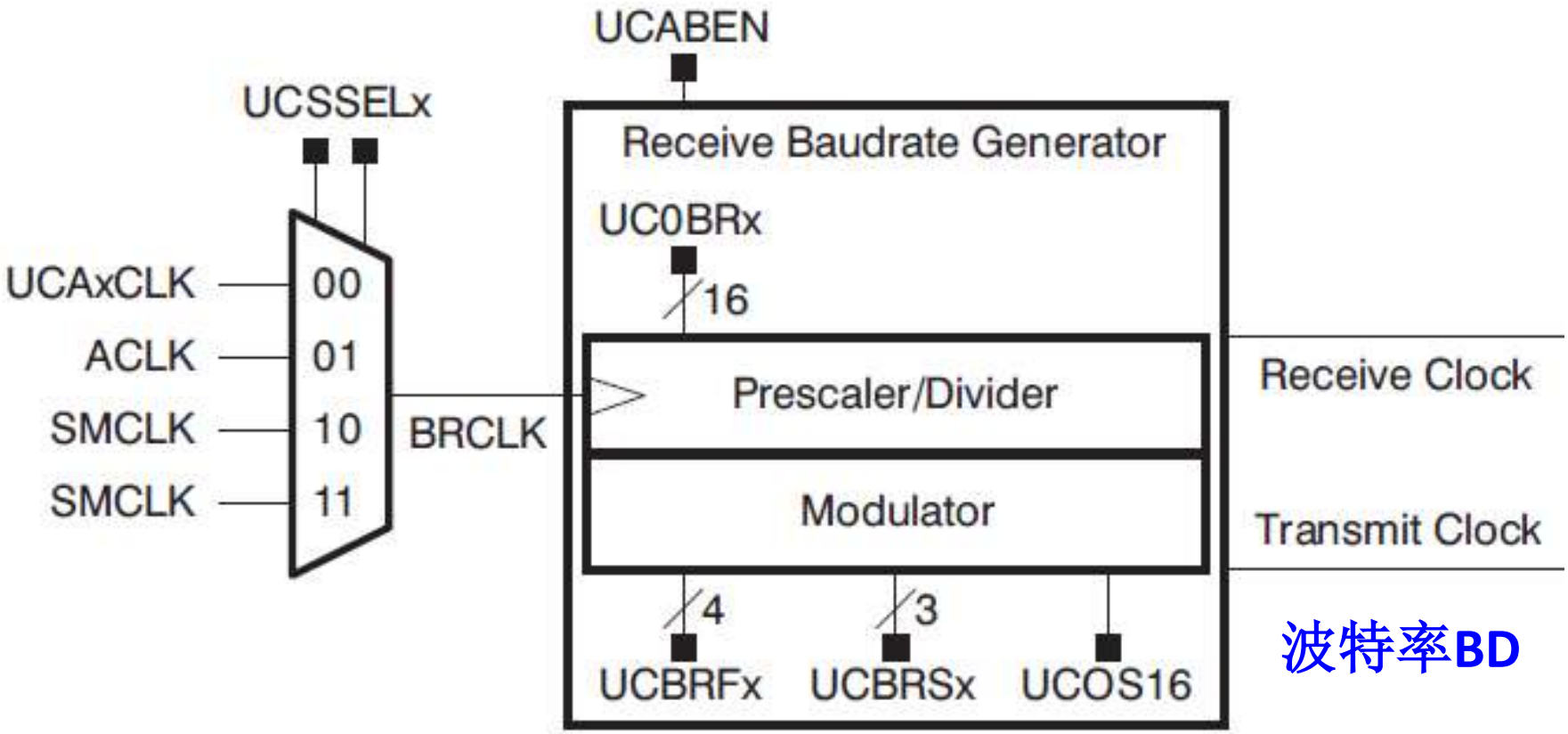
# ● UCAxIV 中断向量寄存器

USCI的收/发共享一个中断向量，可用UCAxIV来区分

| UCAxIV 内容 | 中断源 | 中断标志 | 优先级 |
|---|---|---|---|
| 0000h | 无中断 | - | |
| 0002h | 接收满 | UCAxRXIFG | 高<br>低 |
| 0004h | 发送空 | UCAxTXIFG | |

| 中断源 | 中断标志 | 向量地址 | 优先级,类型号 |
|---|---|---|---|
| … | … | … | … |
| USCI_A1<br>串口收/发 | UCA1RXIFG<br>UCA1TXIFG | 0FFDCh | 46 |
| … | … | … | … |

# MSP430 串口波特率发生器




波特率因子 **N = BRCLK/BD**

**BRCLK：** 进入波特率波特率发生器的输入时钟频率

**BD：** 产生所需要的波特率

# 波特率设置相关寄存器

### USCI_Ax Baud Rate Control Register 0 (UCAxBR0)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRx - low byte | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

### USCI_Ax Baud Rate Control Register 1 (UCAxBR1)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRx - high byte | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | |
|---|---|
| UCBRx | Clock prescaler setting of the baud-rate generator. The 16-bit value of (UCAxBR0 + UCAxBR1 × 256) forms the prescaler value UCBRx. |

### USCI_Ax Modulation Control Register (UCAxMCTL)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRFx | | | | UCBRSx | | | UCOS16 |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

| | | |
|---|---|---|
| UCBRFx | Bits 7-4 | First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 32-3 shows the modulation pattern. |
| UCBRSx | Bits 3-1 | Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 32-2 shows the modulation pattern. |
| UCOS16 | Bit 0 | Oversampling mode enabled<br>0 Disabled<br>1 Enabled |

# 波特率设置

波特率因子 N = BRCLK/BD

BRCLK：进入波特率波特率发生器的输入时钟频率

BD：　　产生所需要的波特率

- 当N的整数部分小于16，则令寄存器中的各位：

```
UCOS16 = 0;                                  //低频模式
UCBRx = INT(N);                  //取整
UCBRSx = round ( ( N-INT(N) ) x 8 );        //四舍五入后取整
```

例如 BRCLK=32768Hz， BD=9600

```
UCOS16 = 0;                                  //低频模式
UCBRx = INT(32768/9600)= 3;                  //取整
UCBRSx = round ( (32768/9600-3 ) x 8 ) = 3;   //四舍五入后取整
```

# 波特率设置(续)

波特率因子 N = BRCLK/BD

BRCLK：进入波特率波特率发生器的输入时钟频率

BD：　　产生所需要的波特率

- 当N的整数部分大于16，则令则令寄存器中的各位：

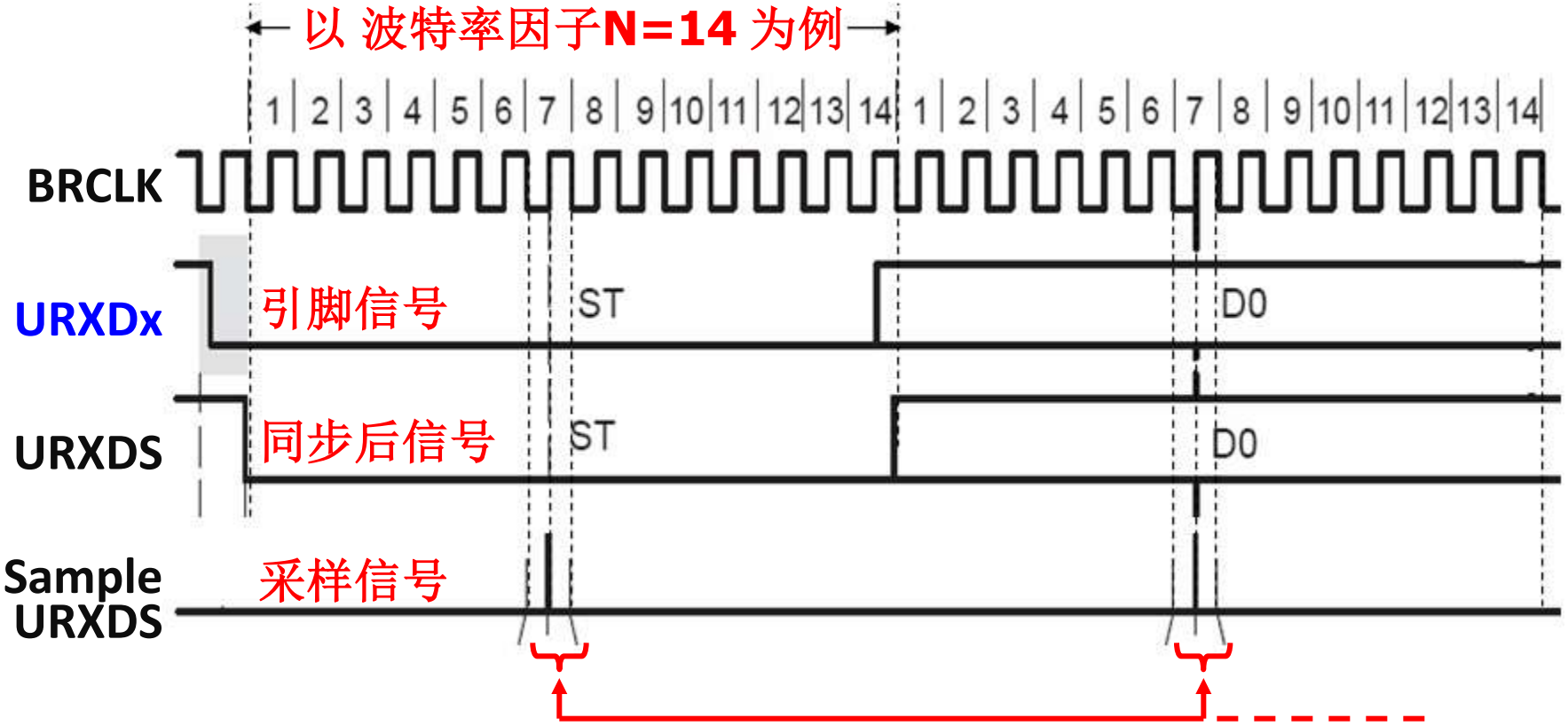
```
UCOS16=1;                                   //高频模式
UCBRx=INT(N/16);                            //取整
UCBRFx=round( ( ( N/16) - INT(N/16) ) ) X16 )   //四舍五入后取整
```

例如 BRCLK=16MHz， BD=9600

```
UCOS16=1;                                   //高频模式
UCBRx=INT(16000000/9600/16) =104 ;              //取整
UCBRFx=round(16000000/9600/16- 104 ) X16 ) =3 ;
                                            //四舍五入后取整
```

# MSP430 异步串行通信的同步与定位原理

## 波特率因子 N=BRCLK/波特率




在中间位置 3 次采样，
并按少数服从多数的表决机制确定每位的值
（majority vote taken）

## Table 32-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

| BRCLK Frequency (Hz) | Baud Rate (baud) | UCBRx | UCBRSx | UCBRFx | Maximum TX Error (%) | | Maximum RX Error (%) | |
|---|---|---|---|---|---|---|---|---|
| 32,768 | 1200 | 27 | 2 | 0 | -2.8 | 1.4 | -5.9 | 2.0 |
| 32,768 | 2400 | 13 | 6 | 0 | -4.8 | 6.0 | -9.7 | 8.3 |
| 32,768 | 4800 | 6 | 7 | 0 | -12.1 | 5.7 | -13.4 | 19.0 |
| 32,768 | 9600 | 3 | 3 | 0 | -21.1 | 15.2 | -44.3 | 21.3 |
| 1,000,000 | 9600 | 104 | 1 | 0 | -0.5 | 0.6 | -0.9 | 1.2 |
| 1,000,000 | 19200 | 52 | 0 | 0 | -1.8 | 0 | -2.6 | 0.9 |
| 1,000,000 | 38400 | 26 | 0 | 0 | -1.8 | 0 | -3.6 | 1.8 |
| 1,000,000 | 57600 | 17 | 3 | 0 | -2.1 | 4.8 | -6.8 | 5.8 |
| 1,000,000 | 115200 | 8 | 6 | 0 | -7.8 | 6.4 | -9.7 | 16.1 |
| 1,048,576 | 9600 | 109 | 2 | 0 | -0.2 | 0.7 | -1.0 | 0.8 |
| 1,048,576 | 19200 | 54 | 5 | 0 | -1.1 | 1.0 | -1.5 | 2.5 |
| 1,048,576 | 38400 | 27 | 2 | 0 | -2.8 | 1.4 | -5.9 | 2.0 |
| 1,048,576 | 57600 | 18 | 1 | 0 | -4.6 | 3.3 | -6.8 | 6.6 |
| 1,048,576 | 115200 | 9 | 1 | 0 | -1.1 | 10.7 | -11.5 | 11.3 |
| 4,000,000 | 9600 | 416 | 6 | 0 | -0.2 | 0.2 | -0.2 | 0.4 |
| 4,000,000 | 19200 | 208 | 3 | 0 | -0.2 | 0.5 | -0.3 | 0.8 |
| 4,000,000 | 38400 | 104 | 1 | 0 | -0.5 | 0.6 | -0.9 | 1.2 |
| 4,000,000 | 57600 | 69 | 4 | 0 | -0.6 | 0.8 | -1.8 | 1.1 |
| 4,000,000 | 115200 | 34 | 6 | 0 | -2.1 | 0.6 | -2.5 | 3.1 |
| 4,000,000 | 230400 | 17 | 3 | 0 | -2.1 | 4.8 | -6.8 | 5.8 |
| 4,194,304 | 9600 | 436 | 7 | 0 | -0.3 | 0 | -0.3 | 0.2 |

**Table 32-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 (continued)**

| BRCLK Frequency (Hz) | Baud Rate (baud) | UCBRx | UCBRSx | UCBRFx | Maximum TX Error (%) | | Maximum RX Error (%) | |
|---|---|---|---|---|---|---|---|---|
| 4,194,304 | 19200 | 218 | 4 | 0 | -0.2 | 0.2 | -0.3 | 0.6 |
| 4,194,304 | 57600 | 72 | 7 | 0 | -1.1 | 0.6 | -1.3 | 1.9 |
| 4,194,304 | 115200 | 36 | 3 | 0 | -1.9 | 1.5 | -2.7 | 3.4 |
| 8,000,000 | 9600 | 833 | 2 | 0 | -0.1 | 0 | -0.2 | 0.1 |
| 8,000,000 | 19200 | 416 | 6 | 0 | -0.2 | 0.2 | -0.2 | 0.4 |
| 8,000,000 | 38400 | 208 | 3 | 0 | -0.2 | 0.5 | -0.3 | 0.8 |
| 8,000,000 | 57600 | 138 | 7 | 0 | -0.7 | 0 | -0.8 | 0.6 |
| 8,000,000 | 115200 | 69 | 4 | 0 | -0.6 | 0.8 | -1.8 | 1.1 |
| 8,000,000 | 230400 | 34 | 6 | 0 | -2.1 | 0.6 | -2.5 | 3.1 |
| 8,000,000 | 460800 | 17 | 3 | 0 | -2.1 | 4.8 | -6.8 | 5.8 |
| 8,388,608 | 9600 | 873 | 7 | 0 | -0.1 | 0.06 | -0.2 | 0,1 |
| 8,388,608 | 19200 | 436 | 7 | 0 | -0.3 | 0 | -0.3 | 0.2 |
| 8,388,608 | 57600 | 145 | 5 | 0 | -0.5 | 0.3 | -1.0 | 0.5 |
| 8,388,608 | 115200 | 72 | 7 | 0 | -1.1 | 0.6 | -1.3 | 1.9 |
| 12,000,000 | 9600 | 1250 | 0 | 0 | 0 | 0 | -0.05 | 0.05 |
| 12,000,000 | 19200 | 625 | 0 | 0 | 0 | 0 | -0.2 | 0 |
| 12,000,000 | 38400 | 312 | 4 | 0 | -0.2 | 0 | -0.2 | 0.2 |
| 12,000,000 | 57600 | 208 | 2 | 0 | -0.5 | 0.2 | -0.6 | 0.5 |
| 12,000,000 | 115200 | 104 | 1 | 0 | -0.5 | 0.6 | -0.9 | 1.2 |
| 12,000,000 | 230400 | 52 | 0 | 0 | -1.8 | 0 | -2.6 | 0.9 |
| 12,000,000 | 460800 | 26 | 0 | 0 | -1.8 | 0 | -3.6 | 1.8 |
| 16,000,000 | 9600 | 1666 | 6 | 0 | -0.05 | 0.05 | -0.05 | 0.1 |
| 16,000,000 | 19200 | 833 | 2 | 0 | -0.1 | 0.05 | -0.2 | 0.1 |
| 16,000,000 | 38400 | 416 | 6 | 0 | -0.2 | 0.2 | -0.2 | 0.4 |
| 16,000,000 | 57600 | 277 | 7 | 0 | -0.3 | 0.3 | -0.5 | 0.4 |
| 16,000,000 | 115200 | 138 | 7 | 0 | -0.7 | 0 | -0.8 | 0.6 |
| 16,000,000 | 230400 | 69 | 4 | 0 | -0.6 | 0.8 | -1.8 | 1.1 |
| 16,000,000 | 460800 | 34 | 6 | 0 | -2.1 | 0.6 | -2.5 | 3.1 |
| 16,777,216 | 9600 | 1747 | 5 | 0 | -0.04 | 0.03 | -0.08 | 0.05 |
| 16,777,216 | 19200 | 873 | 7 | 0 | -0.09 | 0.06 | -0.2 | 0.1 |
| 16,777,216 | 57600 | 291 | 2 | 0 | -0.2 | 0.2 | -0.5 | 0.2 |
| 16,777,216 | 115200 | 145 | 5 | 0 | -0.5 | 0.3 | -1.0 | 0.5 |
| 20,000,000 | 9600 | 2083 | 2 | 0 | -0.05 | 0.02 | -0.09 | 0.02 |
| 20,000,000 | 19200 | 1041 | 6 | 0 | -0.06 | 0.06 | -0.1 | 0.1 |
| 20,000,000 | 38400 | 520 | 7 | 0 | -0.2 | 0.06 | -0.2 | 0.2 |
| 20,000,000 | 57600 | 347 | 2 | 0 | -0.06 | 0.2 | -0.3 | 0.3 |
| 20,000,000 | 115200 | 173 | 5 | 0 | -0.4 | 0.3 | -0.8 | 0.5 |
| 20,000,000 | 230400 | 86 | 7 | 0 | -1.0 | 0.6 | -1.0 | 1.7 |
| 20,000,000 | 460800 | 43 | 3 | 0 | -1.4 | 1.3 | -3.3 | 1.8 |

**Table 32-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1**

| BRCLK Frequency (Hz) | Baud Rate (baud) | UCBRx | UCBRSx | UCBRFx | Maximum TX Error (%) | | Maximum RX Error (%) | |
|---|---|---|---|---|---|---|---|---|
| 1,000,000 | 9600 | 6 | 0 | 8 | -1.8 | 0 | -2.2 | 0.4 |
| 1,000,000 | 19200 | 3 | 0 | 4 | -1.8 | 0 | -2.6 | 0.9 |
| 1,048,576 | 9600 | 6 | 0 | 13 | -2.3 | 0 | -2.2 | 0.8 |
| 1,048,576 | 19200 | 3 | 1 | 6 | -4.6 | 3.2 | -5.0 | 4.7 |
| 4,000,000 | 9600 | 26 | 0 | 1 | 0 | 0.9 | 0 | 1.1 |
| 4,000,000 | 19200 | 13 | 0 | 0 | -1.8 | 0 | -1.9 | 0.2 |
| 4,000,000 | 38400 | 6 | 0 | 8 | -1.8 | 0 | -2.2 | 0.4 |
| 4,000,000 | 57600 | 4 | 5 | 3 | -3.5 | 3.2 | -1.8 | 6.4 |
| 4,000,000 | 115200 | 2 | 3 | 2 | -2.1 | 4.8 | -2.5 | 7.3 |
| 4,194,304 | 9600 | 27 | 0 | 5 | 0 | 0.2 | 0 | 0.5 |
| 4,194,304 | 19200 | 13 | 0 | 10 | -2.3 | 0 | -2.4 | 0.1 |
| 4,194,304 | 57600 | 4 | 4 | 7 | -2.5 | 2.5 | -1.3 | 5.1 |
| 4,194,304 | 115200 | 2 | 6 | 3 | -3.9 | 2.0 | -1.9 | 6.7 |
| 8,000,000 | 9600 | 52 | 0 | 1 | -0.4 | 0 | -0.4 | 0.1 |
| 8,000,000 | 19200 | 26 | 0 | 1 | 0 | 0.9 | 0 | 1.1 |
| 8,000,000 | 38400 | 13 | 0 | 0 | -1.8 | 0 | -1.9 | 0.2 |
| 8,000,000 | 57600 | 8 | 0 | 11 | 0 | 0.88 | 0 | 1.6 |
| 8,000,000 | 115200 | 4 | 5 | 3 | -3.5 | 3.2 | -1.8 | 6.4 |
| 8,000,000 | 230400 | 2 | 3 | 2 | -2.1 | 4.8 | -2.5 | 7.3 |
| 8,388,608 | 9600 | 54 | 0 | 10 | 0 | 0.2 | -0.05 | 0.3 |
| 8,388,608 | 19200 | 27 | 0 | 5 | 0 | 0.2 | 0 | 0.5 |
| 8,388,608 | 57600 | 9 | 0 | 2 | 0 | 2.8 | -0.2 | 3.0 |
| 8,388,608 | 115200 | 4 | 4 | 7 | -2.5 | 2.5 | -1.3 | 5.1 |
| 12,000,000 | 9600 | 78 | 0 | 2 | 0 | 0 | -0.05 | 0.05 |
| 12,000,000 | 19200 | 39 | 0 | 1 | 0 | 0 | 0 | 0.2 |
| 12,000,000 | 38400 | 19 | 0 | 8 | -1.8 | 0 | -1.8 | 0.1 |
| 12,000,000 | 57600 | 13 | 0 | 0 | -1.8 | 0 | -1.9 | 0.2 |
| 12,000,000 | 115200 | 6 | 0 | 8 | -1.8 | 0 | -2.2 | 0.4 |
| 12,000,000 | 230400 | 3 | 0 | 4 | -1.8 | 0 | -2.6 | 0.9 |
| 16,000,000 | 9600 | 104 | 0 | 3 | 0 | 0.2 | 0 | 0.3 |
| 16,000,000 | 19200 | 52 | 0 | 1 | -0.4 | 0 | -0.4 | 0.1 |
| 16,000,000 | 38400 | 26 | 0 | 1 | 0 | 0.9 | 0 | 1.1 |
| 16,000,000 | 57600 | 17 | 0 | 6 | 0 | 0.9 | -0.1 | 1.0 |
| 16,000,000 | 115200 | 8 | 0 | 11 | 0 | 0.9 | 0 | 1.6 |
| 16,000,000 | 230400 | 4 | 5 | 3 | -3.5 | 3.2 | -1.8 | 6.4 |
| 16,000,000 | 460800 | 2 | 3 | 2 | -2.1 | 4.8 | -2.5 | 7.3 |
| 16,777,216 | 9600 | 109 | 0 | 4 | 0 | 0.2 | -0.02 | 0.3 |
| 16,777,216 | 19200 | 54 | 0 | 10 | 0 | 0.2 | -0.05 | 0.3 |
| 16,777,216 | 57600 | 18 | 0 | 3 | -1.0 | 0 | -1.0 | 0.3 |
| 16,777,216 | 115200 | 9 | 0 | 2 | 0 | 2.8 | -0.2 | 3.0 |
| 20,000,000 | 9600 | 130 | 0 | 3 | -0.2 | 0 | -0.2 | 0.04 |
| 20,000,000 | 19200 | 65 | 0 | 2 | 0 | 0.4 | -0.03 | 0.4 |
| 20,000,000 | 38400 | 32 | 0 | 9 | 0 | 0.4 | 0 | 0.5 |
| 20,000,000 | 57600 | 21 | 0 | 11 | -0.7 | 0 | -0.7 | 0.3 |
| 20,000,000 | 115200 | 10 | 0 | 14 | 0 | 2.5 | -0.2 | 2.6 |
| 20,000,000 | 230400 | 5 | 0 | 7 | 0 | 2.5 | 0 | 3.5 |

# 三、利用MSP430 异步串行方式通信

1. 查询方式

2. 中断方式

3. 简单通信协议举例

# 查询方式 USCI 初始化

1). 设置UCAxCTL1 中 SWRST=1 ( UCA1CTL1 |= UCSWRST; )

复位USCI模块, 硬件自动完成对模块内寄存器的下面设置:

UCRXIE=0， UCTXIE=0， 禁止USART的中断请求；

使UCRXIFG=0， UCRXERR=0， UCBRK=0， UCPE=0，

UCOE=0， UCFE=0；

使UTCXIFGx=1；

3). 设置USARTx的3个控制寄存器和3个波特率寄存器

控制寄存器: UCAxCTL0, UCAxCTL1,

波特率控制寄存器: UCAxBRW 或 UCAxBR1， UCAxBR0

UCAxMCTL，

2). 设置相关引脚的端口功能选择

将相关引脚设置为USCI模块功能

5). 设置UxCTL 中 SWRST=0 (UCA1CTL1 &= ~UCSWRST; )

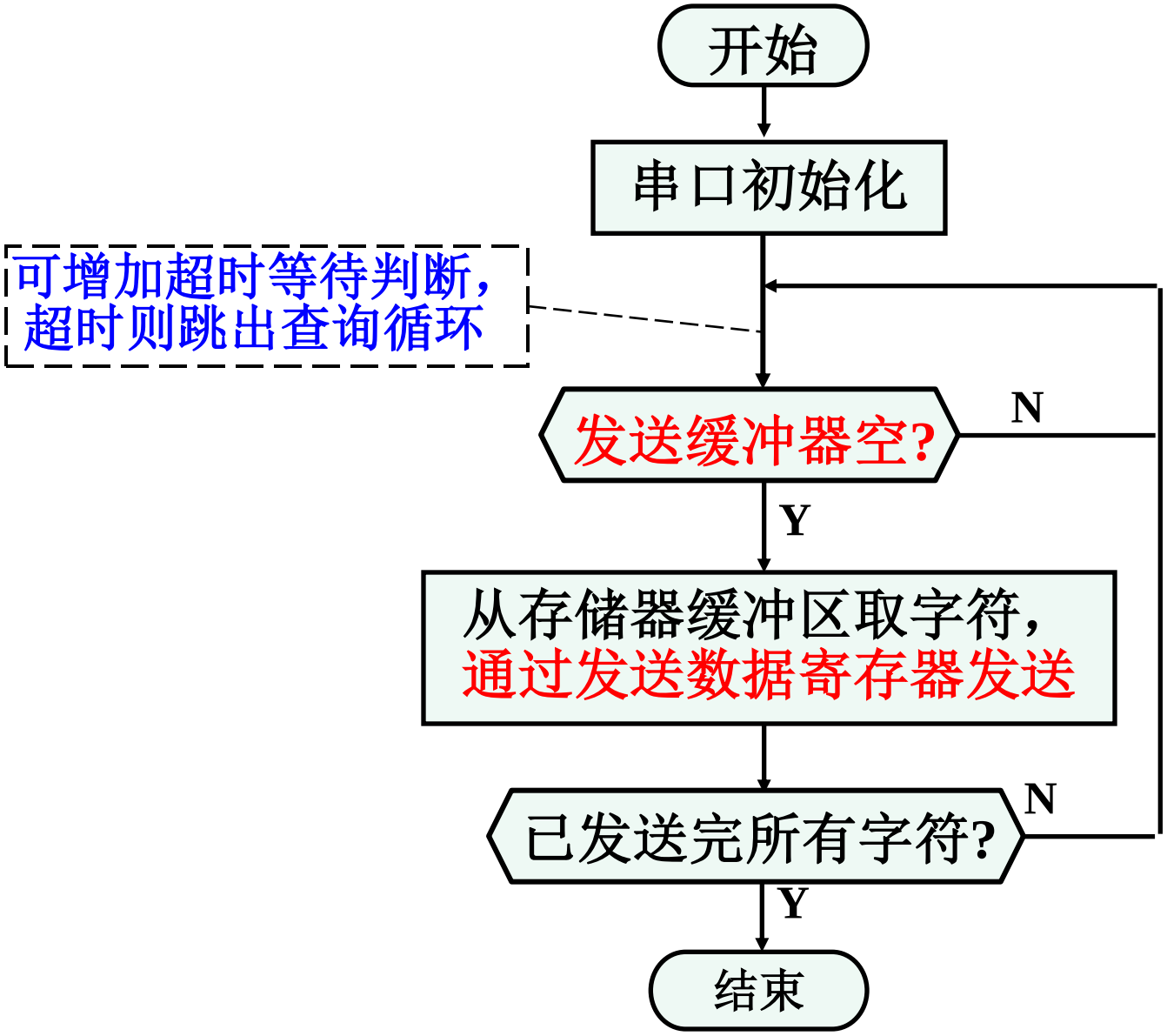
允许USART运行

*注：* PUC(上电清零 )时自动置位 SWRST=1

## *例* 初始化USCI_A1 子程序

```
void initUART( )
{  P8SEL |= BIT2 + BIT3;                    //配置IO引脚
   UCA1CTL1 |= UCSWRST;     //配置串口模块
   //无校验，1停止位，先发低位，8数据位，普通异步方式
   UCA1CTL0 = 0;
   UCA1CTL1 = UCSWRST + UCSSEL__SMCLK + UCRXEIE;
  //假设#define SMCLK_FREQ 16000000，设置波特率 =9600
  UCA1BRW = SMCLK_FREQ / 9600;
  UCA1MCTL= UCBRF_3 + UCOS16;
  UCA1CTL1 &= ~UCSWRST;
}
```

# 串口查询方式发送数据流程

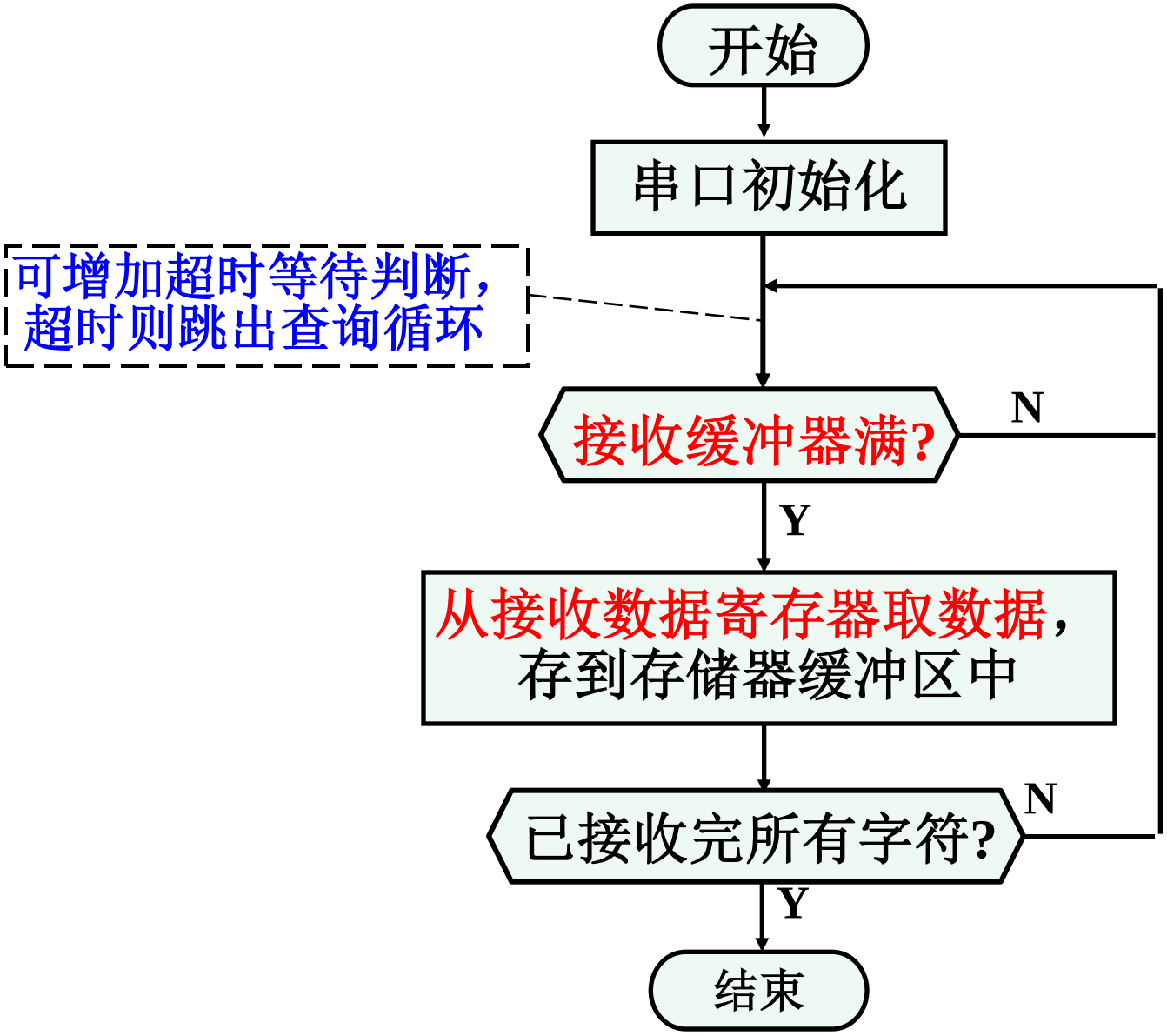


# 查询方式发送程序举例

## ——发送从string 开始的25字节

```
……
 initUART( );
……
for (i=0;i<25;i++)
{while ( (UCA1IFG & UCTXIFG) ==0 );//检测发送缓冲是否空
    UCA1TXBUF=string[i];                    //取一个字符发送
};
……
```

# 串口查询方式接收数据流程



## 查询方式接收程序举例

—— 从**USART0**接收**32**字节存入**buffer** 开始的存储器中。不考虑超时等待

```
……
USART0_init();
……
for (j=0; j<32; j++)
 {   while ((UCA1IFG & UCRXIFG)==0);    //检测接收缓冲器是否满
     buffer[j]= UCA1RXBUF;               //接收一个字符并保存
 };
……
```

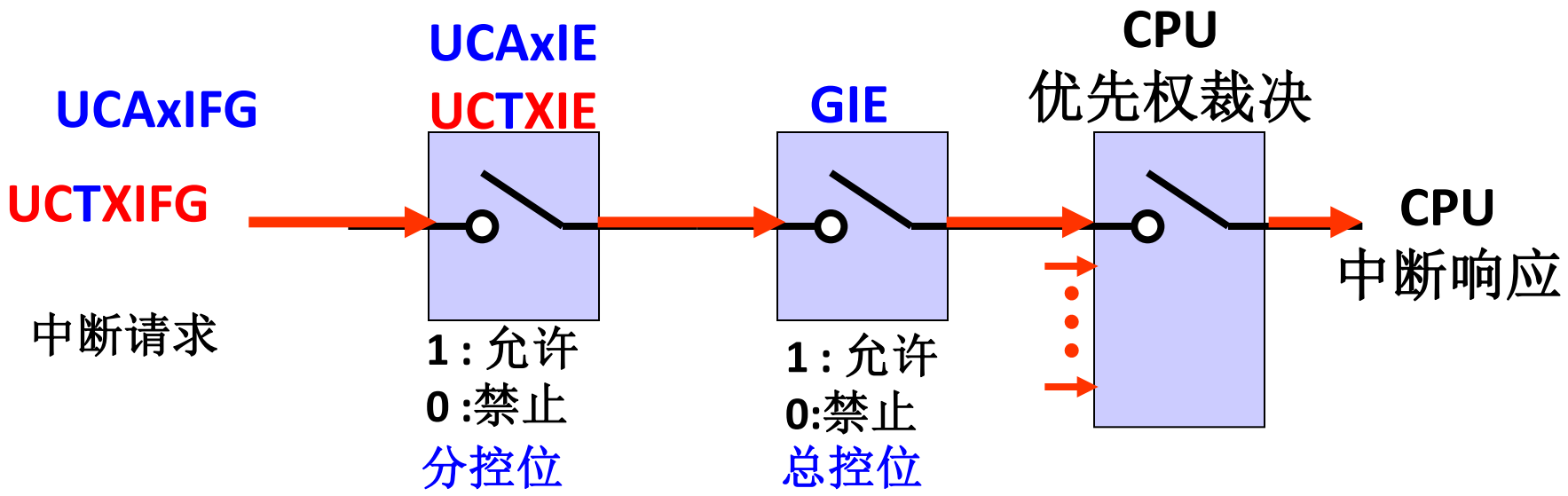
注意：从**UxRXBUF**读取数据后，**URXIFGx**自动清**0**，不用软件清**0**

## 2. 中 断 方 式

- 串口发送、接收中断控制
- 串口发送、接收中断向量
- 串口中断方式初始化
- 串口中断方式程序流程举例
- 串口中断方式程序清单例

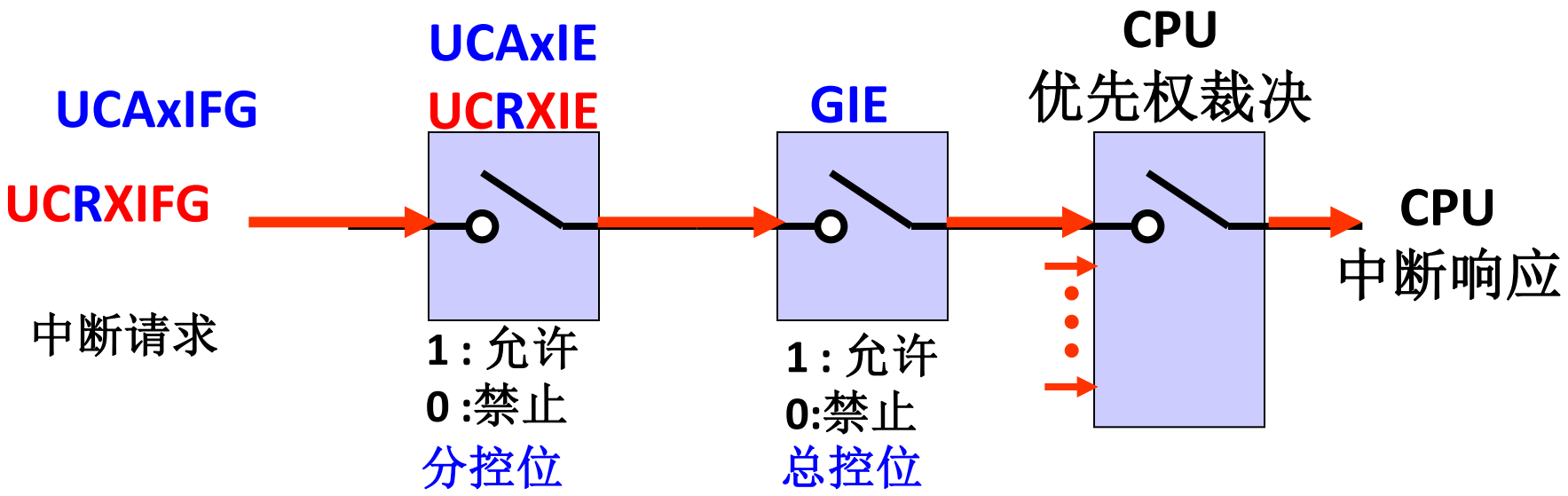
# USART 发送中断控制

- 当被发送的字符从发送数据寄存器UCAxTXBUF移入发送移位寄存器, 发送数据寄存器变空, 置UCAxIFG中的发送寄存器空标志UCTXIFG=1，发中断申请，相当于主动告诉CPU可以发送新数据了，如果UCAxIE中的分中断允许位UCTXIE=1, 且CPU的 GIE=1, 则CPU将响应该中断，转去执行相应类型的中断程序



# USART 接收中断控制

- 当串行接口检测到引脚UCAxRXD上有异步串行格式数据，通过移位寄存器转化成并行数据存放到接收数据寄存器时，置UCAxIFG中的接收寄存器满标志UCRXIFG=1，发中断申请，相当于主动告诉CPU有新数据到了，
  如果UCAxIE中的分中断允许位UCRXIE=1, 且CPU的 GIE=1，则CPU将响应该中断，转去执行相应类型的中断程序



# MSP430f6638 串口通信接收和发送共享一个中断向量

| 中断源 | 中断标志 | 向量地址 | 优先级,类型号 |
|---|---|---|---|
| … | … | … | … |
| USCI_A1<br>串口收/发 | UCA1RXIFG<br>UCA1TXIFG | 0FFDCh | 46 |
| … | … | … | … |

msp430x14x.h中的定义：

```
……
#define USCI_B1_VECTOR      (45 * 1u)          /* 0xFFDA USCI B1 Receive/Transmit */
#define USCI_A1_VECTOR      (46 * 1u)          /* 0xFFDC USCI A1 Receive/Transmit */
#define PORT1_VECTOR        (47 * 1u)          /* 0xFFDE Port 1 */
……
```

# USART中断方式 初始化

1). 设置UCAxCTL1 中 SWRST=1 ( UCA1CTL1 |= UCSWRST; )

2). 设置USARTx的3个控制寄存器和3个波特率寄存器

控制寄存器: UCAxCTL0, UCAxCTL1,

波特率控制寄存器: UCAxBRW 或 UCAxBR1， UCAxBR0

UCAxMCTL

3). 设置相关引脚的端口功能选择

将相关引脚设置为USCI模块功能

4). 设置UxCTL 中 SWRST=0 ( UCA1CTL1 &= ~UCSWRST; )

允许USART运行

同查询方式

开中断

1. 设置编程 中断允许寄存器UCAxIE,

根据需要置UCxRXIE=1 或 UCxTXIE=1, 开分中断允许

2. 置GIE=1, 开总中断允许

*例：* 编程设置允许**USCI_A1** 的接收中断

**UCA1IE |=UARXIE;**

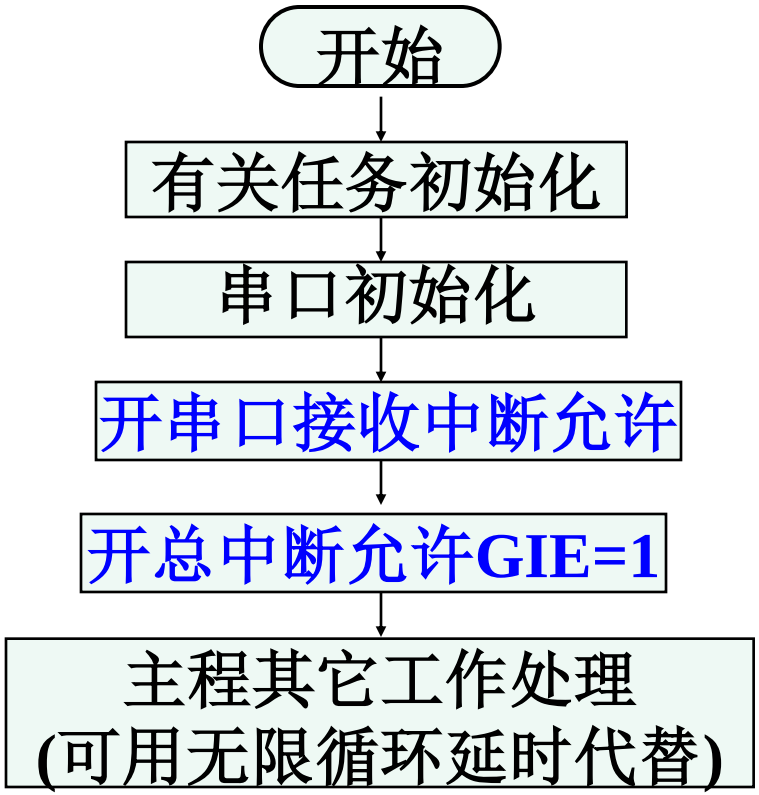
思考：如何打开**USCI_A1**的发送中断?

**UCA1IE**

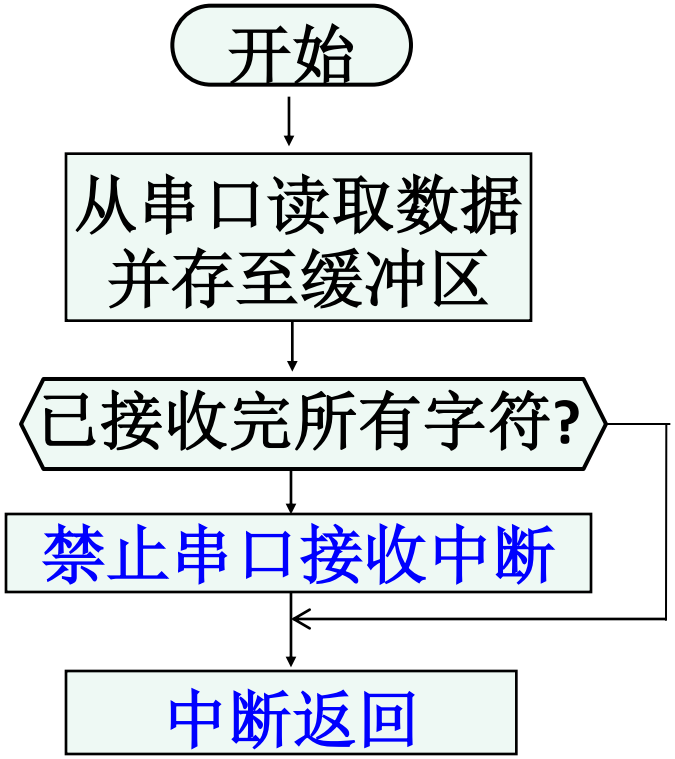
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 保留 | | | | | | UCTXIFG | UCRXIFG |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | rw-0 | rw-0 |

# *例* 采用中断方式接收数据程序流程

## 1. 主程序




## 2. 中断子程




## 3. 设置中断向量

根据中断源在
中断向量表相应位置
设置中断向量

## *例* 采用中断方式从USCI_A1串口接收数据存到buffer中

```
#include “msp430F6638.h“
#define  SMCLK_FREQ  16000000
void  initUART( );
char     buffer[32];
 unsigned int  j;

int  main( void )
{   WDTCTL = WDTPW + WDTHOLD;
      initUART(  );                    //串口初始化
    UCA1IE|=UCRXIE;                          //开串口接收中断允许
    _EINT();                          //开总中断
    while(1);
}

#pragma  vector= USCI_A1_VECTOR
__interrupt  void  UCA1_RX_isr( )
{   buffer[ j ]=UCA1RXBUF;              //将接收到的数据存至存储器
    j++;
    if (j==32) UCA1IE&=UCRXIE;          //关闭串口A1接收中断允许
}

void  intUART( )
{ 略 }
```

思考：1. 编写中断方式的发送程序?

2. 发送中断允许什么时候打开？是否需要关闭？