用 户 说 明 书

USER SPECIFICATIONS

PLS 系列 VICO 检测器

WORKING WITH TUNNEL SENSORS

目 录

一、仪器部件及配件清单

1、信号发射机（带电器接线盒） 1 台

2、信号接收机　　1 台

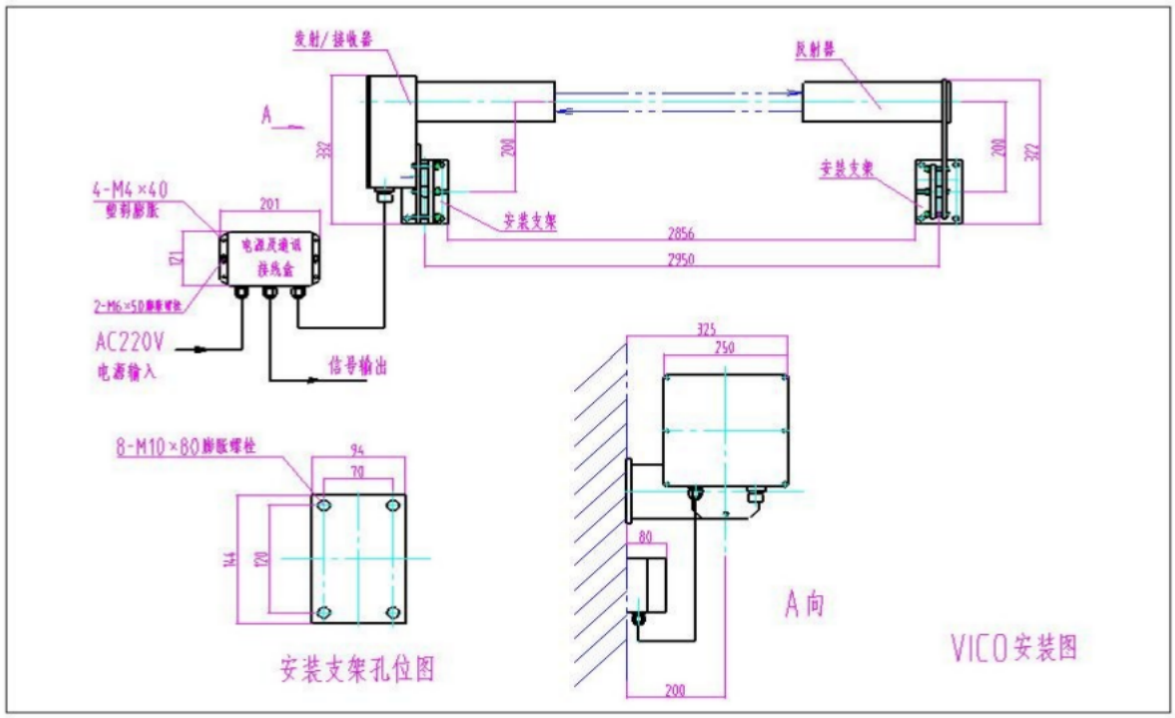
3、壁装支架　　2 支

二、仪器设备安装

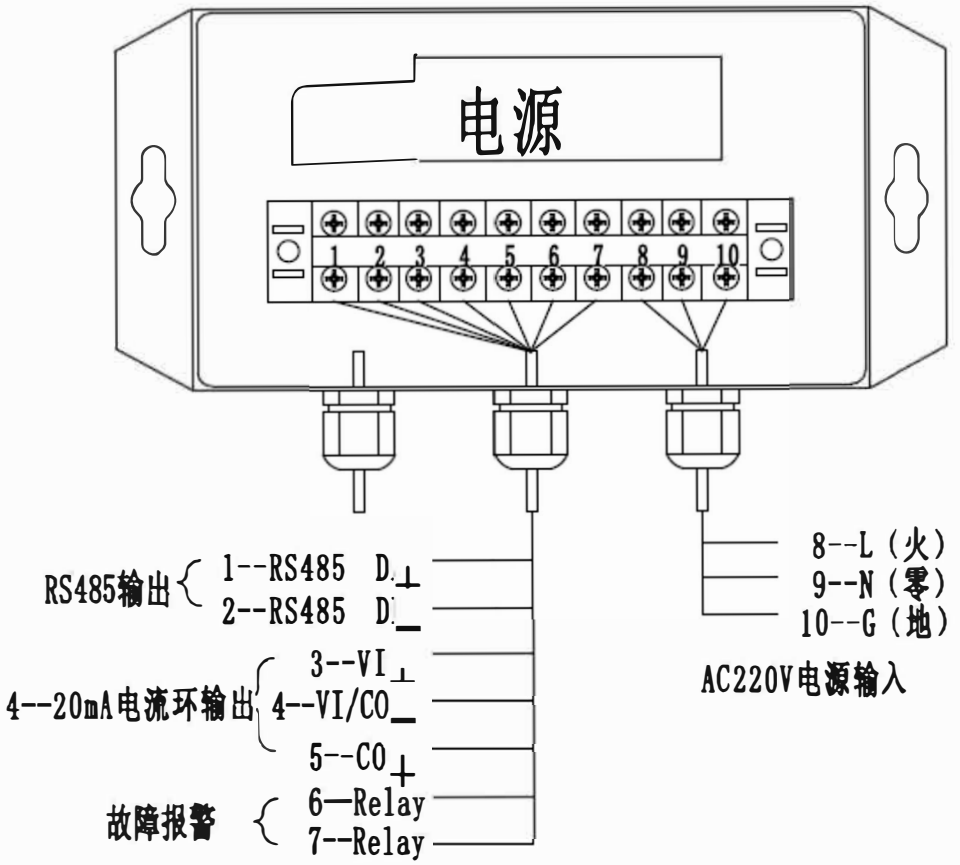
2.1 仪器工作原理简述

透射式隧道烟雾浓度检测仪利用了光在大气中传播受到大气衰减的原理进行测试。通过光发射机发出光信号，在不同的隧道大气条件下，经过大气信道对光信号的衰减后，由光接收机接收，经过 CPU 对信号进行计算处理，给出隧道各种环境下的 VI 值。

2.2 烟雾浓度检测仪外形及连线示意图



2.3 检测器电源接线盒及通信线接口图



三、安装步骤

清单所有部件、线缆、接头均在工厂装配成套，分为接收机、发射机及壁装支架部份。现场只作部件连接及支架打孔安装。

3.1 支架安装

支架安装要求上表面在同一水平面，与地面平行，保证安装后不会产生位移。支架中心距为：基线 L。根据订货要求 L=3 米。支架由 6 颗 M10×100 钢膨胀螺栓固定于隧道墙面，距地面高度 3-4 米。

3.2 发射、接收机安装

在支架安装完成后，取出发射机和接收机相向安放于支架上，分别拧入 4 颗 M10 内六角螺栓和垫圈，先不拧紧。

3.3 线缆连接

3.3.1 通讯电缆由现场调试技术人员连接

3.3.2 电源输入按 2.3 图所示接入 AC220V 50Hz 交流电源。

3.3.3 数据输出：该仪器具有 3 种数据输出接口：

1、VI 1 路 4－20mA 电流环输出；

2、RS485 输出；

3、1 组继电器输出。

数据输出按要求接入控制中心。

四、设备调试

4.1 初调

目测，使接收端/反射端，大致对中。

4.2 精调

核对线缆连接无误后，接通电源。肉眼观察当发射端有光束发出后（该光束不会对人眼造成任何损伤），先固定发射端不动，将光束导入反射端，拧紧发射端上下两颗螺栓，调节左右两颗螺栓，调到显示最大值，然后拧紧发射端左右两颗螺栓，调节左右两颗螺栓至新的最大值后。这时，固定发射端，按上述步骤调节反射端，两端反复几次，确认最大值后锁紧所有螺栓。光线较暗时对中比较方便，观察反射端光斑完全进入接收端光学镜头，然后观察数据进行微调，直到出现最大值，然后锁紧相关螺钉（具体细节由现场调试技术人员进行调试）。

五、维护

为保证测量的精度，需要监测仪器的工作状态及对设备进行例行维护

1、设备具有自检功能，若镜头污染，在允许范围内，仪器具备自动补偿功能，超过警戒值，仪器将停止工作，并发出镜头清洁指令；

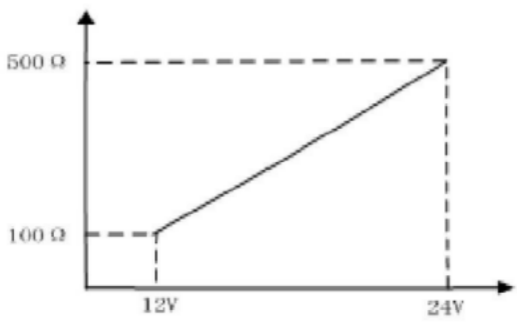
2、若长时间无数据接受，请观察机器是否死机或通信故障，可以尝试将机器重新启动，若状态依旧向设备维护部反映；

3、镜头被污染后，需取下防护筒，用纸巾蘸纯净水反复擦拭，直到污物去除然后用镜头布擦拭，清洗完成后重新安装防护罩；

4、本仪器具备校准功能，在对镜片进行人工清洁后，重新上电即可。

六、电流输出接口对应关系

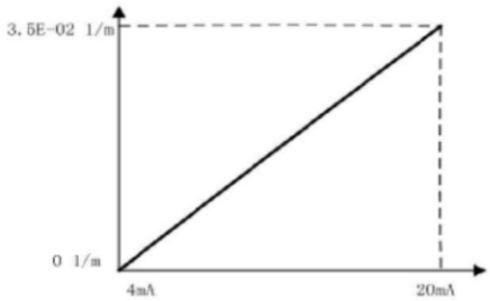
6.1、电压和电流环负载对应曲线图 （如右图）



6.2 电流转换参数对照表

（1） 烟雾浓度与电流成线性关系

正向风速（米/秒）	输出电流（4~20mA）
0	4.0
3.5E-02	20.0
烟雾浓度和输出电流呈线性关系，如右表	



七、BN-VC930 隧道能见度检测仪技术参数

型号	PLS-VICODZ10
测试范围	VI: 035*10-3 1/m
测试精度	VI: ±0.00001 1/m
环境温度	-55 - 65 摄氏度
环境湿度	0 - 100%RH
测试时间	1 - 120 秒可调
通信接口	一路 4-20mA 对应 VI 值，最大阻抗 500 欧姆 一路数字量 RS485 输出 一路继电器输出，可配置为极限报警
供电电压	220V±15%VAC 50HZ；
防护等级	IP67
整机重量	20KG（包括仪器、支架、通讯电缆等）

MODBUS 协议-RTU 模式

一、 通讯端口定义

起始位	1 bit
数据位	8 bit
奇偶校验位	EVEN(偶校验)

停止位	1 bit
波特率	9600

二、 通讯方式

仪器与外部设备通讯方式为 RS485 或 RS232，默认通讯方式为 *RS485*。

三、 通讯协议

MODBUS 协议-RTU 模式。

四、 协议简介

MODBUS 是一个请求/应答协议，并且提供功能码规定的服务。MODBUS 功能码是 MODBUS 请求/应答 PDU 的元素。它是一项应用层报文传输协议，用于在通过不同类型的总线或网络连接的设备之间的客户机/服务器通信。

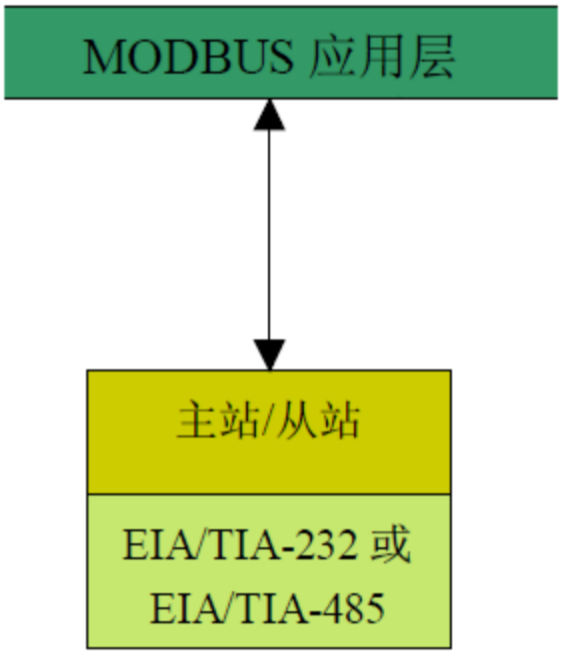


图 1

五、 协议描述

MODBUS 协议定义了一个与基础通信层无关的简单协议数据单元（PDU）。特定总线或网络上的MODBUS 协议映射能够在应用数据单元（ADU）上引入一些附加域。

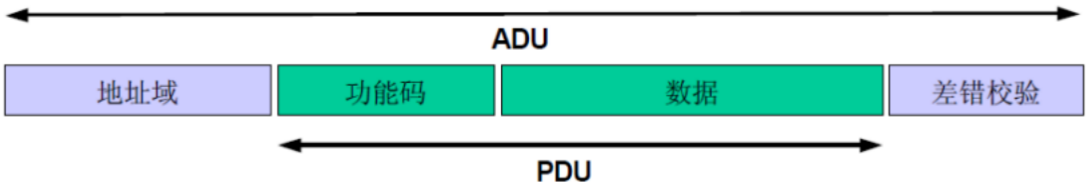


图 2. 通用 MODBUS 帧

MODBUS 有两种传输模式，RTU 和 ASCII。*本仪器采用 RTU 传输模式。*

(一)RTU传输模式

当设备使用RTU (Remote Terminal Unit) 模式在Modbus 串行链路通信，报文中每个8位字节含有两个4 位十六进制字符。这种模式的主要优点是较高的数据密度，在相同的波特率下比ASCII 模式有更高的吞吐率。每个报文必须以连续的字符流传送。

- **RTU 模式串行位系列**

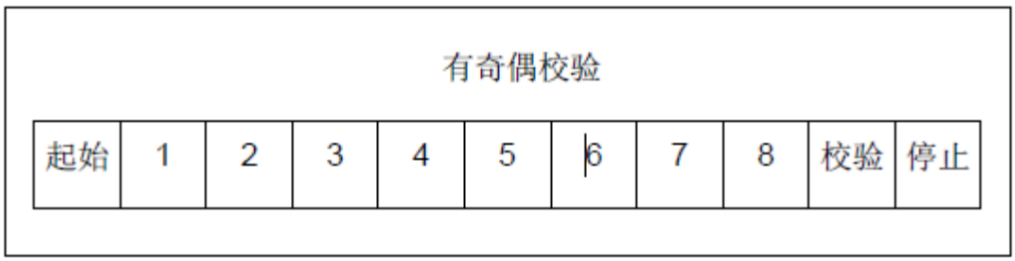


图 3. RTU 模式串行位序列

- **Modbus RTU 报文帧**

子节点地址	功能代码	数据	CRC
1 字节	1 字节	0 到 252 字节	2 字节 CRC 低 | CRC 高

图 4. ASCII 报文帧

- **CRC 校验**

在RTU 模式包含一个对全部报文内容执行的，基于循环冗余校验(CRC – Cyclical Redundancy Checking) 算法的错误检验域。CRC 域检验整个报文的内容。不管报文有无奇偶校验，均执行此检验。CRC 包含由两个8 位字节组成的一个16 位值。CRC 域作为报文的最后的域附加在报文之后。计算后，首先附加低字节，然后是高字节。CRC高字节为报文发送的最后一个子节。

附加在报文后面的CRC 的值由发送设备计算。接收设备在接收报文时重新计算CRC 的值，并将计算结果于实际接收到的CRC 值相比较。如果两个值不相等，则为错误。

附录含有CRC 生成的详细示例。

(二)MODBUS通讯模型

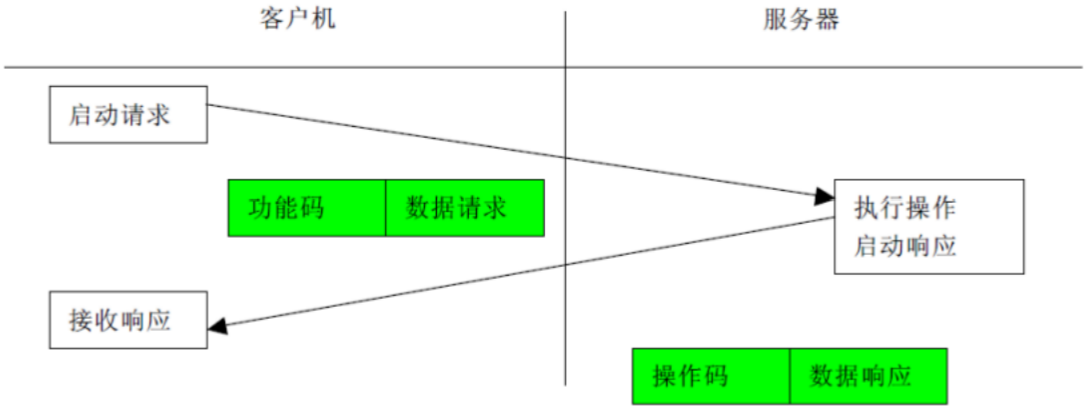


图 5. MODBUS 事务处理（无差错）

图 5 中，此处的客户机指的就是要查询仪器信息的主机。而服务器就是仪器。

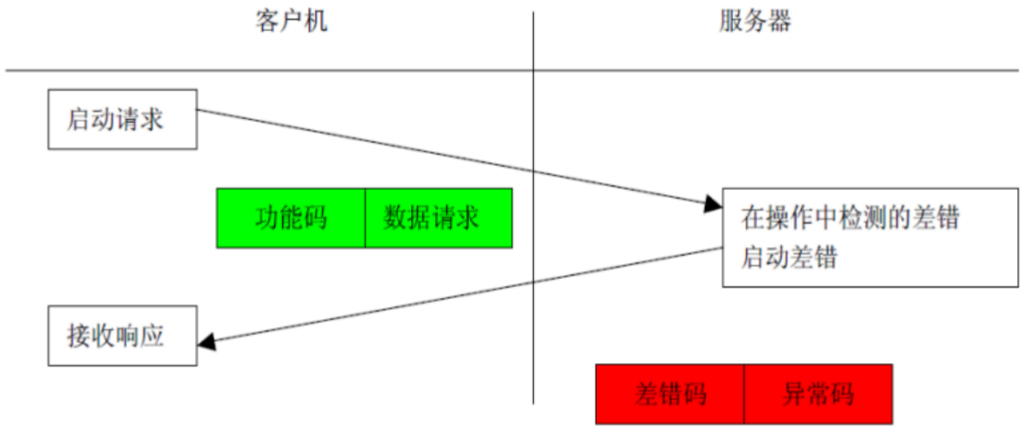


图 6. MODBUS 事务处理（异常响应）

数据编码

MODBUS 使用一个”big-Endian” 表示地址和数据项。这意味着当发射多个字节时，首先发送最高有效位。

例如：

寄存器大小值	值
16bit	0x1234

发送的第一字节为 0x12 然后 0x34

注：具体 MODBUS 协议，请参照 MODBUS 标准。

(三)本仪器协议支持

- 支持的功能码

功能码类型	字长	功能码	描述

数据访问	16 bit	03	内部寄存器读出

- **仪器内部寄存器描述**

本仪器有两个量可供读取：VI(烟雾浓度)和 CO(一氧化碳浓度)。

内部用 3 个寄存器表示这两个量。

寄存器	字长	值数据类型	值定义	值范围
寄存器 1	16 bit	32 bit 浮点型	VI	1.0e-06 至 3.5e-02
寄存器 2	16 bit			
寄存器 3	16 bit	16 bit 整形	CO	0 至 500

- **32 bit 浮点型存储方式(IEEE754 标准)**

D3	D2	D1	D0
高字节	中间字节 1	中间字节 2	低字节

- **寄存器中存储方式**

数值定义	寄存器	Bit	字节位置
烟雾浓度	寄存器 1-高字节	8 bit	D1
	寄存器 1-低字节	8 bit	D0
	寄存器 2-高字节	8 bit	D3
	寄存器 2-低字节	8 bit	D2

数值定义	寄存器	Bit	字节位置
CO	寄存器 3-高字节	8 bit	D1
	寄存器 3-低字节	8 bit	D0

- **功能码(0x03)描述-读保持寄存器**

在一个远程设备中，使用该功能码读取保持寄存器连续块的内容。请求PDU说明了起始寄存器地址和寄存器数量。从零地址开始寻址寄存器。因此，寻址寄存器1-3 对应地址为0-2。将响应报文中的寄存器数据分成每个寄存器有两字节，在每个字节中直接地调整二进制内容。对于每个寄存器，第一个字节包括高位比特，并且第二个字节包括低位比特。

请求

功能码	1 个字节	0x03
起始地址	2 个字节	0x0000 至 0x0002
寄存器数量	2 个字节	1 至 3

响应

功能码	1 个字节	0x03
字节数	1 个字节	N*2
寄存器值	N*2 个字节	

注： N 表示寄存器个数

错误响应

差错码	1 个字节	0x83
异常码	1 个字节	01，02 ，03 ，04 ，06

通讯实例：

读取内部 3 个寄存器的实例

请求		响应	
域名	(十六进制)	域名	(十六进制)
功能码	03	功能码	03
起始地址-高字节	00	字节数	06
起始地址-低字节	00	寄存器 1 值-高字节	23
读取数量-高字节	00	寄存器 1 值-低字节	A1
读取数量-低字节	03	寄存器 2 值-高字节	3C
		寄存器 2 值-低字节	DB
		寄存器 3 值-高字节	00
		寄存器 3 值-低字节	FF

如上，实际通讯系列为：

- 请求系列：

  (HEX)　　01 03 00 00 00 03 05 CB

- 应答系列：

  (HEX)　　01 03 06 23 A1 3C DB 00 FF 3E 5C

- 整型解析

D1	D0
寄存器 3 高字节	寄存器 3 低字节
00	FF
高字节	低字节

转换为整形后，值为：　　255

- 浮点型解析

D3	D2	D1	D0
寄存器 2 高字节	寄存器 2 低字节	寄存器 1 高字节	寄存器 1 低字节
3C	DB	23	A1
高字节	中间字节 1	中间字节 2	低字节

转换为浮点数后，值为：　　0.02675（约等于 0.027）

附录1. －CRC 的生成

循环冗余校验(CRC) 域为两个字节，包含一个二进制16 位值。附加在报文后面的CRC 的值由发送设备计算。接收设备在接收报文时重新计算CRC 的值，并将计算结果于实际接收到的CRC值相比较。如果两个值不相等，则为错误。

CRC 的计算，开始对一个16位寄存器预装全1. 然后将报文中的连续的8位子节对其进行后续的计算。只有字符中的8个数据位参与生成CRC 的运算，起始位，停止位和校验位不参与CRC 计算。

CRC 的生成过程中， 每个 8－位字符与寄存器中的值异或。然后结果向最低有效位(LSB)方向移动(Shift) 1位，而最高有效位(MSB) 位置充零。然后提取并检查LSB：如果LSB 为1， 则寄存器中的值与一个固定的预置值异或；如果LSB 为 0， 则不进行异或操作。

这个过程将重复直到执行完8 次移位。完成最后一次（第8 次）移位及相关操作后，下一个8位字节与寄存器的当前值异或，然后又同上面描述过的一样重复8 次。当所有报文中子节都运算之后得到的寄存器中的最终值，就是CRC.

生成CRC 的过程为：

1. 将一个16 位寄存器装入十六进制FFFF (全1). 将之称作CRC 寄存器.
2. 将报文的第一个8位字节与16 位CRC 寄存器的低字节异或，结果置于CRC 寄存器.
3. 将CRC 寄存器右移1位(向LSB 方向)， MSB 充零. 提取并检测LSB.
4. (如果LSB 为0)：重复步骤3 (另一次移位).
   (如果LSB 为1)：对CRC 寄存器异或多项式值0xA001 (1010 0000 0000 0001).
5. 重复步骤3 和 4，直到完成8 次移位。当做完此操作后，将完成对8位字节的完整操作。
6. 对报文中的下一个字节重复步骤2 到5，继续此操作直至所有报文被处理完毕。
7. CRC 寄存器中的最终内容为CRC 值.
8. 当放置CRC 值于报文时，如下面描述的那样，高低字节必须交换。

将 CRC 放置于报文

当16 位CRC (2 个 8 位字节) 在报文中传送时，低位字节首先发送，然后是高位字节。

例如， 如果 CRC 值为十六进制1241 (0001 0010 0100 0001)：

地址	功能	数据计数	数据	数据	数据	数据	CRC低	CRC高
							0x41	0x12

例

执行CRC 生成的C 语言的函数在下面示出。所有的可能的CRC 值都被预装在两个数组中，当计算报文内容时可以简单的索引即可。一个数组含有16 位CRC 域的所有256个可能的高位字节，另一个数组含有地位字节的值。这种索引访问CRC 的方式提供了比对报文缓冲区的每个新字符都计算新的CRC 更快的方法。

注意：此函数内部执行高/低CRC 字节的交换。此函数返回的是已经经过交换的CRC 值。也就是说，从该函数返回的CRC 值可以直接放置于报文用于发送。(低字节首先发送，接着发送高字节)

函数使用两个参数：

unsigned char *puchMsg; 指向含有用于生成CRC 的二进制数据报文缓冲区的指针

unsigned short usDataLen; 报文缓冲区的字节数.

CRC 生成函数

```
unsigned short CRC16 ( puchMsg,  usDataLen )  /* 函数以unsigned short 类型返回CRC */

unsigned char *puchMsg ;                      /* 用于计算CRC 的报文*/

unsigned short usDataLen ;                    /* 报文中的字节数*/

{
    unsigned char uchCRCHi = 0xFF ;           /* CRC 的高字节初始化*/

    unsigned char uchCRCLo = 0xFF ;           /* CRC 的低字节初始化*/

    unsigned uIndex ;                         /* CRC 查询表索引*/

    while (usDataLen--)                       /* 完成整个报文缓冲区*/
    {
        uIndex = uchCRCLo ^ * puchMsg++ ;     /* 计算CRC */

        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex] ;
        uchCRCHi = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

高字节表

```
/* 高位字节的CRC 值*/

static unsigned char auchCRCHi[] =
{
  0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,0x00, 0xC1,
  0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,0x40, 0x01,
  0xC0,0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,0x81, 0x40,
  0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0, 0x80,
```

```
  0x41,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,0x00, 0xC1,
  0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,0x41, 0x01,
  0xC0,0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,0x80, 0x41,
  0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00,0xC1, 0x81,
  0x40,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,0x00, 0xC1,
  0x81,0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40, 0x01,
  0xC0,0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,0x81, 0x40,
  0x01,0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0, 0x80,
  0x41,0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
  0x00, 0xC1, 0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
  0x81,0x40, 0x01, 0xC0,0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
  0xC0,0x80, 0x41, 0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
  0x01,0xC0, 0x80, 0x41,
  0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,0x00, 0xC1,
  0x81,0x40
} ;
```

低字节表

```
/* 低位字节的CRC 值*/

static char auchCRCLo[] =
{
  0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7,0x05, 0xC5,
  0xC4,0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB,0x0B,
  0xC9, 0x09,0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE,
  0xDF, 0x1F, 0xDD,
  0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2,0x12, 0x13,
  0xD3,
  0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32,0x36, 0xF6, 0xF7,
  0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E,0xFE, 0xFA, 0x3A,
  0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B,0x2A, 0xEA, 0xEE,
  0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27,0xE7, 0xE6, 0x26,
  0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1,0x63, 0xA3, 0xA2,
  0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD,0x6D, 0xAF,
  0x6F,
  0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8,0xB9, 0x79,
  0xBB,
  0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4,0x74, 0x75,
  0xB5,
  0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,0x50, 0x90, 0x91,
  0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94,0x54, 0x9C, 0x5C,
  0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59,0x58, 0x98, 0x88,
  0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D,0x4D, 0x4C, 0x8C,
```

```
    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83,0x41, 0x81,
    0x80,0x40
};
```

附录2.　　4个字节转换为浮点数。（基于C语言）

```
union                                          //共用体
{
    float TestData_Float;                      //浮点数（4 个字节）
    unsigned char TestArray[4];                //数值
}TData;
```

注：在共用体中，上例中的浮点数和四个字节的字符数组共用一段存储空间。

解析：

D3	D2	D1	D0
寄存器 2 高字节	寄存器 2 低字节	寄存器 1 高字节	寄存器 1 低字节
40	AC	19	DF
高字节	中间字节 1	中间字节 2	低字节

转换为浮点数后，值为：　　5.378

程序：

```
float Tempfloat;

TData.TestArray [3]= 0x40;                     //输入高字节
TData.TestArray [2]= 0xac;                     //
TData.TestArray [1]= 0x19;                     //
TData.TestArray [0]= 0xdf;                     //低字节

Tempfloat = TData.TestData_Float;              //得到浮点数 5.378
```

注：以上实例在 VC6.0 测试通过，因 PC 是基于小端模式存储，故低地址字节存储数据低字节，高地址字节存储数据高字节。在实际应用中，根据实际存储模式，决定存储浮点型高低字节的位置。