附 录 A
（资料性附录）
串行链路诊断计数器的管理

A.1 一般描述

Modbus 串行链路定义了一个诊断计数器列表，进行性能和出错管理。

这些计数值可以通过 Modbus 应用协议及其诊断功能访问（功能码 08）。

可以通过一个带有计数器编号的子功能码得到每个计数值。可以利用子功能码 0x0A 清除所有计数器。

在 Modbus 应用协议规范中描述诊断功能的格式。

表 A.1 是串行链路设备支持的诊断和相应子功能码的列表。

表 A.1 串行链路设备支持的诊断和相应子功能码

子功能码	计数器编号	计数器名称	注释（配合后续图表）
十六进制	十进制		
0x0B	1	返回总线报文计数	在上一次重启动、清除计数器操作或加电之后，远程设备在通信系统中检测到的报文数量。不考虑带有出错 CRC 的报文
0x0C	2	返回总线通信出错计数	在上一次重启动、清除计数器操作或加电之后，远程设备遇到的 CRC 出错数量。在检测到字符出错（超限差错、奇偶校验差错）的情况下，或在报文长度<3 个字节的情况下，接收设备不能计算 CRC。在这种情形下，依然增加计数值
0x0D	3	返回从站异常出错计数	在上一次重启动、清除计数器操作或加电之后，远程设备检测到的 Modbus 异常出错数量。它也包含广播报文中检测到的出错，即使这种情况下不返回异常报文 在 GB/T 19582.1—2008 中描述并列出异常差错
0x0E	4	返回从站报文计数	在上一次重启动、清除计数器操作或加电之后，对远程设备寻址的报文数量，包括远程设备处理的广播报文
0x0F	5	返回从站无响应计数	在上一次重启动、清除计数器操作或加电之后，没有返回响应（既没有正常响应也没有异常响应）的远程设备接收的报文数量。也就是说，这个计数器计算已接收到的广播报文数量
0x10	6	返回从站 NAK 计数	在上一次重启动、清除计数器操作或加电之后，远程设备对接收到的报文返回否定确认（NAK）异常响应报文的数量 在 GB/T 19582.1—2008 中描述并列出异常响应
0x11	7	返回从站忙计数	在上一次重启动、清除计数器操作或加电之后，远程设备对接收到的报文返回从站忙异常响应报文的数量 在 GB/T 19582.1—2008 中描述并列出异常响应
0x12	8	返回总线字符超限计数	在上一次重启动、清除计数器操作或加电之后，由于字符超限状况而无法处理的寻址远程设备的报文数量。由于字符抵达端口的速度高于存储字符的速度，或者由于硬件故障而丢失字符，均产生字符超限

A.2 计数器管理流程图

图 A.1～图 A.3 描述了必须将前面每个计数器计数值增加的条件。

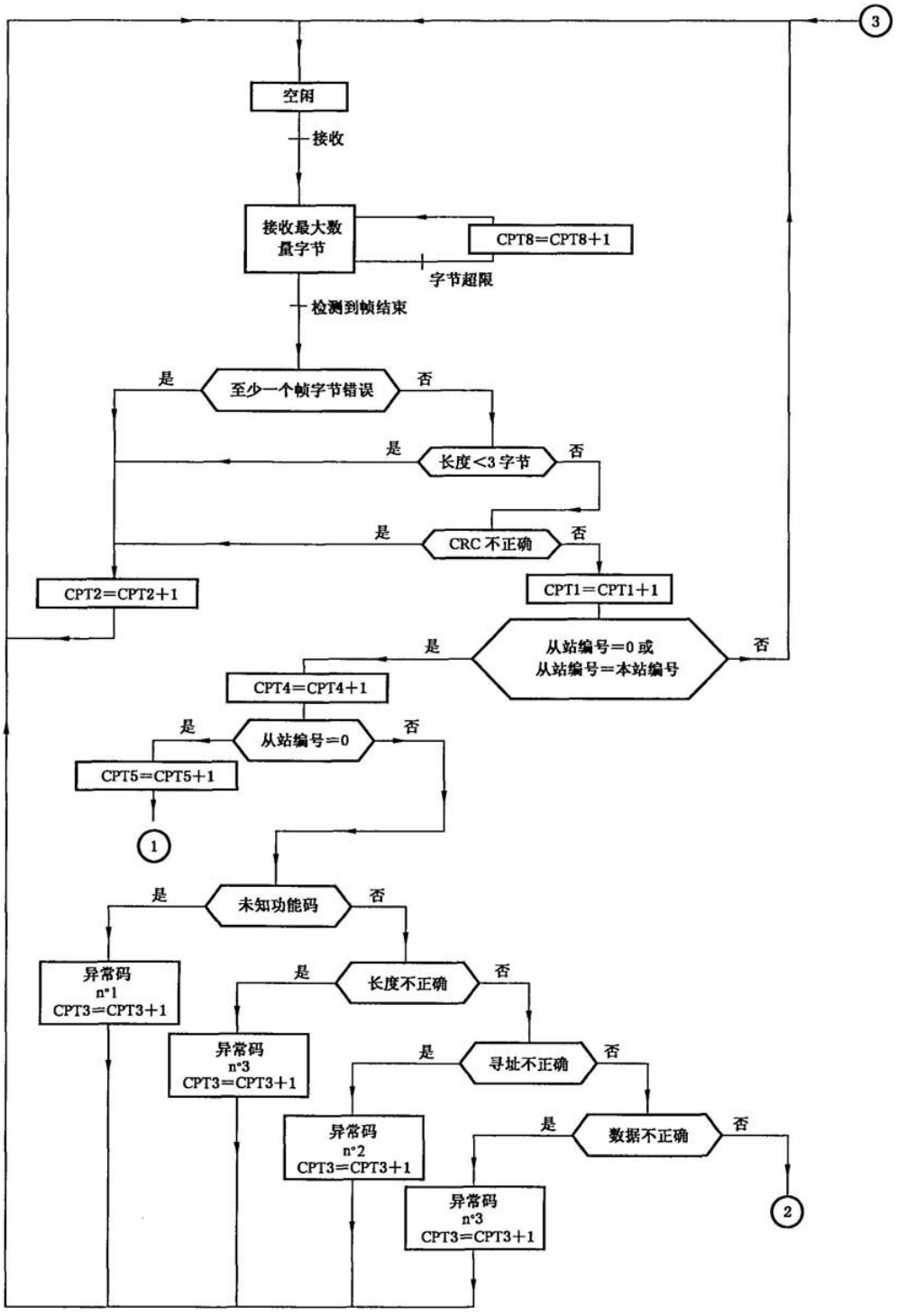

图 A.1 计数器管理流程图-1

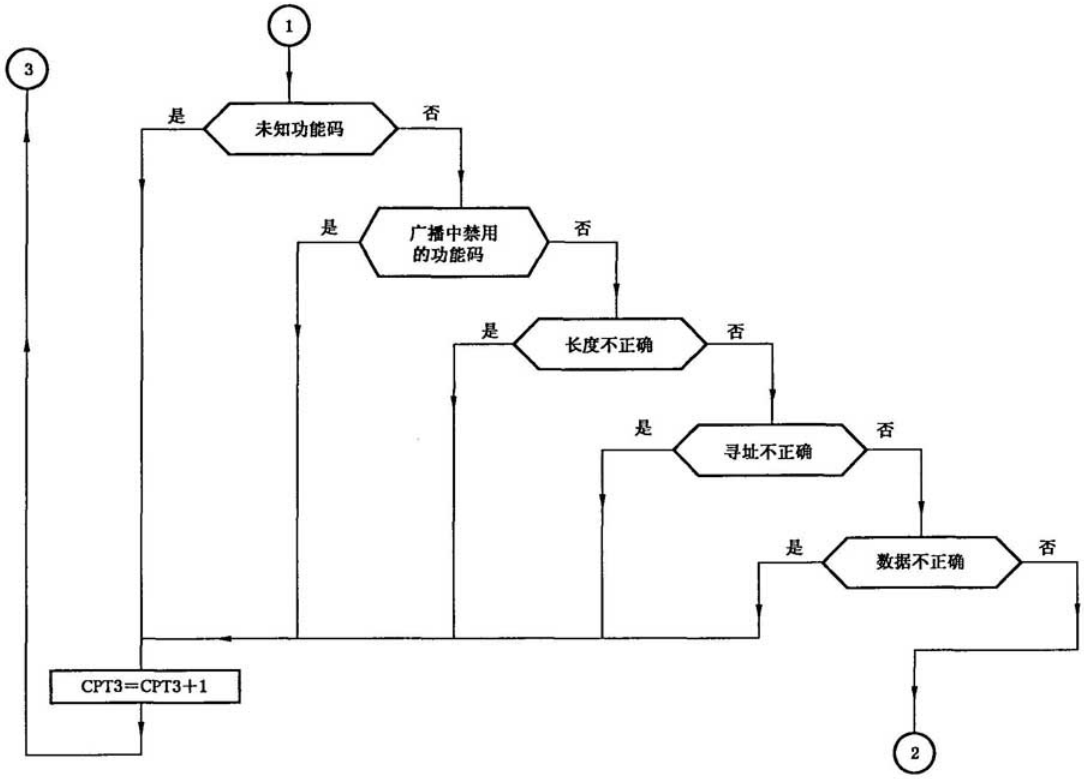

图 A.2 计数器管理流程图-2

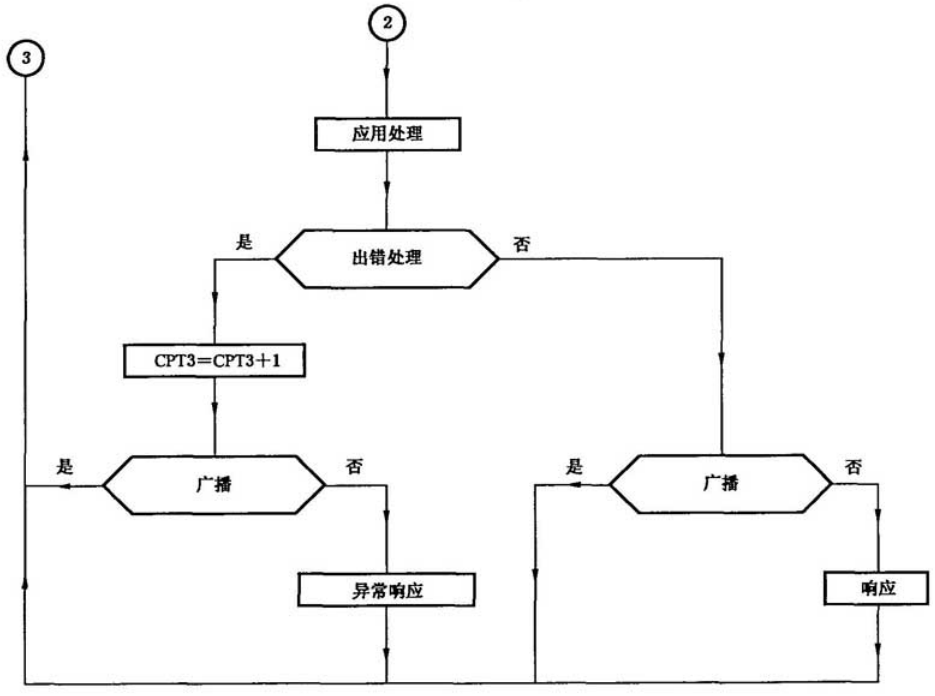

图 A.3 计数器管理流程图-3

附 录 B
（资料性附录）
LRC/CRC 生成

B.1 LRC 生成

纵向冗余校验(LRC)字段是一个字节，包含一个 8 位二进制值。发送设备计算 LRC 值，将 LRC 值附加到报文中。在接收报文过程中，接收设备重新计算 LRC 值，并将计算值与接收到的 LRC 字段中实际值相比较。如果两个值不相等，则说明报文有错误。

计算 LRC，对报文中的所有连续 8 位字节相加，忽略任何进位，然后求出其二进制补码。LRC 是一个 8 位字段，因此导致结果大于 255 的每个新的相加运算，只是简单地将字段值回零“循环”。因为没有第 9 位，自动放弃进位。

生成一个 LRC 的过程是：

a) 将报文中的所有字节相加，不包括起始“:”和结束 CRLF。将相加结果放到 8 位字段中，以便丢弃进位。

b) 从 FF(全 1)十六进制中减去最终的字段值，产生 1 的补码(二进制反码)。

c) 加 1 产生二进制补码。

B.1.1 将 LRC 放置报文中

当在报文中发送 8 位 LRC(2 个 ASCII 字符)时，首先发送高位字符，然后发送低位字符。例如：如果 LRC 值为十六进制 61(0110 0001)，见图 B.1。

冒号	地址	功能码	数据计数	数据	数据	数据	数据	LRC 高位	LRC 低位	CR	LF
								“6” 0x36	“1” 0x31		

图 B.1 LRC 字符序列

例：下面表示了 C 语言函数进行 LRC 生成的示例。

函数带有两个参数：

unsigned char * auchMsg；含有生成 LRC 所使用的二进制数据的报文缓存区指针，

unsigned short usDataLen；报文缓存区中的字节数。

B.1.2 LRC 生成函数

```
static unsigned char LRC(auchMsg,usDataLen)     /* 函数返回 unsigned char 类型的 LRC */
unsigned char *auchMsg;                         /* 用于计算 LRC 的报文 */
unsigned short usDataLen;                       /* 报文的字节数量 */
{
    unsigned char uchLRC=0;                     /* LRC 字节初始化 */
    while (usDataLen--)                         /* 遍历报文缓存区 */
    uchLRC += *auchMsg++;                       /* 缓存区字节相加，无进位 */
```

```
    return ((unsigned char)(-((char)uchLRC)));  /* 返回二进制补码 */
}
```

B.2 CRC 生成

循环冗余校验(CRC)字段为两个字节,包含一个二进制 16 位值。发送设备计算 CRC 值,将 CRC 值附加到报文中。在接收报文过程中,接收设备重新计算 CRC 值,并将计算值与接收到的 CRC 字段中实际值相比较,如果两个值不相等,则说明报文有错误。

通过对一个 16 位寄存器预装载全"1"来启动 CRC 计算,然后开始将报文中的后续 8 位字节与当前寄存器中的内容进行计算,只有每个字符中的 8 个数据位参与生成 CRC 的计算,起始位、停止位和校验位不参与 CRC 计算。

在生成 CRC 过程中,每个 8 位字符与寄存器中的值异或,然后,向最低有效位(LSB)方向移动这个结果,而用零填充最高有效位(MSB),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)移位之后,下一个 8 位字节与寄存器的当前值异或,然后像上述描述的那样重复 8 次这个过程。在已经计算报文中所有字节之后,寄存器的最终值就是 CRC。

生成一个 CRC 的过程是:

a) 将十六进制 FFFF(全 1)装入一个 16 位寄存器。将这个寄存器称作 CRC 寄存器。

b) 将报文的第一个 8 位字节与 16 位 CRC 寄存器的低字节异或,将结果放置在 CRC 寄存器中。

c) 将 CRC 寄存器右移 1 位(向 LSB 方向),MSB 填充零。提取并检测 LSB。

d) (如果 LSB 为 0):重复步骤 c)(进行另一次移位)。
(如果 LSB 为 1):将 CRC 寄存器与多项式值 0xA001(1010 0000 0000 0001)异或。

e) 重复步骤 c)和 d),直到完成 8 次移位。在完成这个操作之后,即完成了对一个完整的 8 位字节的处理。

f) 对报文的下一个 8 位字节重复步骤 b)～e)。继续进行这种操作,直到处理报文中所有字节为止。

g) CRC 寄存器中的最终内容为 CRC 值。

h) 当将 CRC 值放置到报文中时,必须按如下所述交换高位和低位字节。

B.2.1 将 CRC 放置报文中

当在报文中发送 16 位 CRC(2 个 8 位字节)时,首先发送低位字节,然后发送高位字节。

例如:如果 CRC 值为十六进制 1241(0001 0010 0100 0001),见图 B.2。

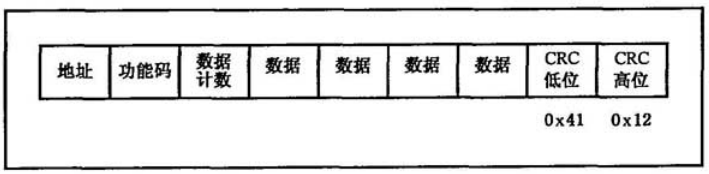

图 B.2 CRC 字节序列

计算 CRC16 的算法见图 B.3。

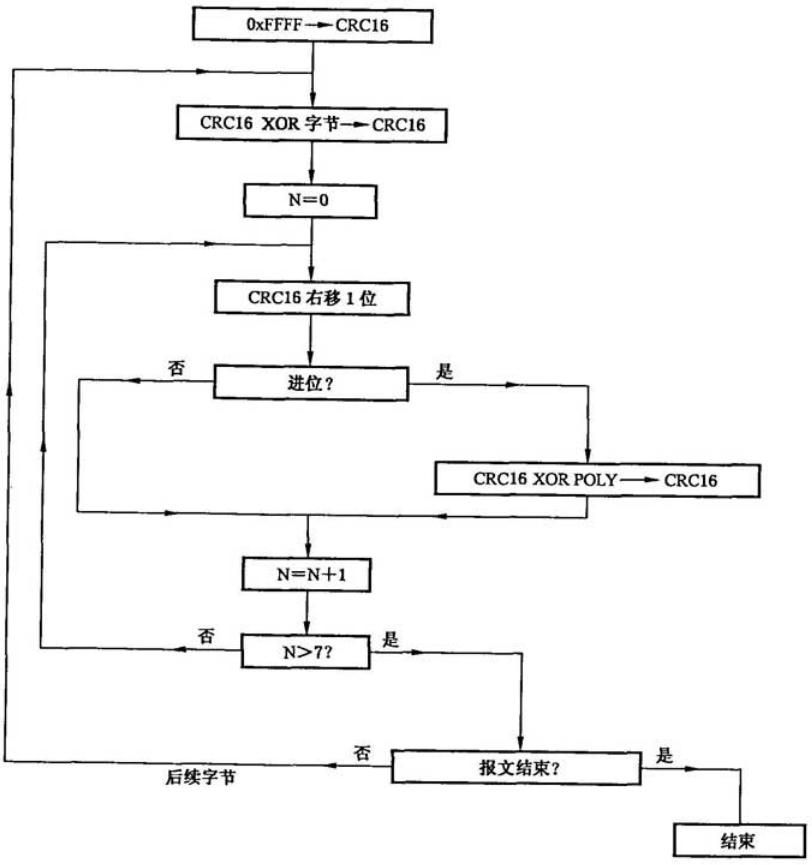

图 B.3　计算 CRC16 的方法

XOR=异或

N=信息位的数量

POLY=计算 CRC16 的多项式=1010 0000 0000 0001

(生成多项式=$1+x^2+x^{15}+x^{16}$)

在 CRC16 中,发送的第一个字节为最低有效字节。

CRC 计算示例(帧 02 07)

CRC 寄存器初始化		1111	1111	1111	1111
XOR 第一个字符		0000	0000	0000	0010
		1111	1111	1111	1101
	移位 1	0111	1111	1111	1110|1
标志 1,XOR 多项式		1010	0000	0000	0001
		1101	1111	1111	1111
	移位 2	0110	1111	1111	1111|1
标志 1,XOR 多项式		1010	0000	0000	0001
		1100	1111	1111	1110
	移位 3	0110	0111	1111	1111|0
	移位 4	0011	0011	1111	1111|1

		1010	0000	0000	0001
		1001	0011	1111	1110
	移位 5	0100	1001	1111	1111|0
	移位 6	0010	0100	1111	1111|1
		1010	0000	0000	0001
		1000	0100	1111	1110
	移位 7	0100	0010	0111	1111|0
	移位 8	0010	0001	0011	1111|1
		1010	0000	0000	0001
XOR 第二个字符		1000	0001	0011	1110
		0000	0000	0000	0111
		1000	0001	0011	1001
	移位 1	0100	0000	1001	1100|1
		1010	0000	0000	0001
		1110	0000	1001	1101
	移位 2	0111	0000	0100	1110|1
		1010	0000	0000	0001
		1101	0000	0100	1111
	移位 3	0110	1000	0010	0111|1
		1010	0000	0000	0001
		1100	1000	0010	0110
	移位 4	0110	0100	0001	0011|0
	移位 5	0011	0010	0000	1001|1
		1010	0000	0000	0001
		1001	0010	0000	1000
	移位 6	0100	1001	0000	0100|0
	移位 7	0010	0100	1000	0010|0
	移位 8	0001	0010	0100	0001|0
				最高有效位	最低有效位

则帧的 CRC16 为:4112

示例:

下面是一个用 C 语言函数进行 CRC 生成的示例。将所有的可能 CRC 值都预先装入两个数组中,伴随着函数对报文缓存区的处理来简单地索引这些数组。一个数组含有 16 位 CRC 字段高位字节的所有可能的 256 个 CRC 值,另一个数组含有低位字节的所有可能的 CRC 值。

这种索引 CRC 的方式比对报文缓存区的每个新字符都计算新的 CRC 值的方法更快捷。

注:此函数内部执行高/低 CRC 字节的交换。此函数返回的 CRC 值是已经交换了的 CRC 值。

因此,从该函数返回的 CRC 值可以直接地放置于报文中,用于发送。

函数带有两个参数:

unsigned char * puchMsg;含有生成 CRC 所使用的二进制数据的报文缓存区指针,

unsigned short usDataLen;报文缓存区中的字节数。

B.2.2 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;                  /* 报文中的字节数量                     */
{

    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--)                    /*遍历报文缓存区                        */
{
    uIndex=uchCRCLo ^ * puchMsg++;         /* 计算 CRC                             */
    uchCRCLo=uchCRCHi ^ auchCRCHi[uIndex];
    uchCRCHi=auchCRCLo[uIndex];
}
    return (uchCRCHi << 8 | uchCRCLo);
}
```

B.2.3 高位字节表

```
/* 高位字节的 CRC 值表 */
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
};
```

B.2.4 低位字节表

```
/* 低位字节的 CRC 值表 */
static unsigned 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
};
```

参 考 文 献

[1] ANSI/ TIA/ EIA-232-E-1997 Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange.

[2] ANSI/ TIA/ EIA-485-A-1998 Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems.

[3] AWG(American Wire Gauge)("AWG"(美国线规)是在美国及其他国家中使用的表示线径的标准方法,参见 1993 年 McGraw-Hill 出版的第 13 期电气工程师标准手册 D. G. Fink and H. W. Beaty).

[4] Modbus. org Modbus application protocol specification.

中 华 人 民 共 和 国
国 家 标 准
基于 Modbus 协议的工业自动化网络规范
第 2 部分：Modbus 协议在串行链路上的
实现指南
GB/T 19582.2—2008
*
中国标准出版社出版发行
北京复兴门外三里河北街 16 号
邮政编码：100045
网址 www.spc.net.cn
电话：68523946 68517548
中国标准出版社秦皇岛印刷厂印刷
各地新华书店经销
*
开本 880×1230 1/16 印张 2.5 字数 66 千字
2008 年 5 月第一版 2008 年 5 月第一次印刷
*
书号：155066·1-31329

如有印装差错 由本社发行中心调换
版权专有 侵权必究
举报电话：(010)68533533

GB/T 19582.2-2008