

Modbus通信协议

工业控制已从单机控制走向集中监控、集散控制,如今已进入网络时代,工业控制器连网也为网络管理提供了方便。Modbus就是工业控制器的网络协议中的一种。

一、Modbus 协议简介 Modbus 协议是应用于电子控制器上的一种通用语言。通过此协议,控制器相互之间、控制器经由网络(例如以太网)和其它设备之间可以通信。它已经成为一通用工业标准。有了它,不同厂商生产的控制设备可以连成工业网络,进行集中监控。

此协议定义了一个控制器能认识使用的消息结构,而不管它们是经过何种网络进行通信的。它描述了一控制器请求访问其它设备的过程,如果回应来自其它设备的请求,以及怎样侦测错误并记录。它制定了消息域格局和内容的公共格式。

当在一Modbus网络上通信时,此协议决定了每个控制器须要知道它们的设备地址,识别按地址发来的消息,决定要产生何种行动。如果需要回应,控制器将生成反馈信息并用Modbus协议发出。在其它网络上,包含了Modbus协议的消息转换为在此网络上使用的帧或包结构。这种转换也扩展了根据具体的网络解决节地址、路由路径及错误检测的方法。

1、在Modbus网络上传输

标准的Modbus口是使用一RS-232C兼容串行接口,它定义了连接口的针脚、电缆、信号位、传输波特率、奇偶校验。控制器能直接或经由Modem组网。

控制器通信使用主—从技术,即仅一设备(主设备)能初始化传输(查询)。其它设备(从设备)根据主设备查询提供的数据作出相应反应。典型的主设备:主机和可编程仪表。典型的从设备:可编程控制器。

主设备可单独和从设备通信,也能以广播方式和所有从设备通信。如果单独通信,从设备返回一消息作为回应,如果是广播方式查询的,则不作任何回应。Modbus协议建立了主设备查询的格式:设备(或广播)地址、功能代码、所有要发送的数据、一错误检测域。

从设备回应消息也由Modbus协议构成,包括确认要行动的域、任何要返回的数据、和一错误检测域。如果在消息接收过程中发生一错误,或从设备不能执行其命令,从设备将建立一错误消息并把它作为回应发送出去。

2、在其它类型网络上传输

在其它网络上,控制器使用对等技术通信,故任何控制都能初始和其它控制器的通信。这样在单独的通信过程中,控制器既可作为主设备也可作为从设备。提供的多个内部通道可允许同时发生的传输进程。

在消息位,Modbus协议仍提供了主—从原则,尽管网络通信方法是“对等”。如果一控制器发送一消息,它只是作为主设备,并期望从从设备得到回应。同样,当控制器接收到一消息,它将建立一从设备回应格式并返回给发送的控制器。

3、查询—回应周期

(1) 查询

查询消息中的功能代码告之被选中的从设备要执行何种功能。数据段包含了从设备要执行功能的任何附加信息。例如功能代码03是要求从设备读保持寄存器并返回它们的内容。数据段必须包含要告之从设备的信息:从何寄存器开始读及要读的寄存器数量。错误检测域为从设备提供了一种验证消息内容是否正确的方法。

(2) 回应

如果从设备产生一正常的回应,在回应消息中的功能代码是在查询消息中的功能代码的回应。数据段包括了从设备收集的数据:象寄存器值或状态。如果有错误发生,功能代码将被修改以用于指出回应消息是错误的,同时数据段包含了描述此错误信息的代码。错误检测域允许主设备确认消息内容是否可用。

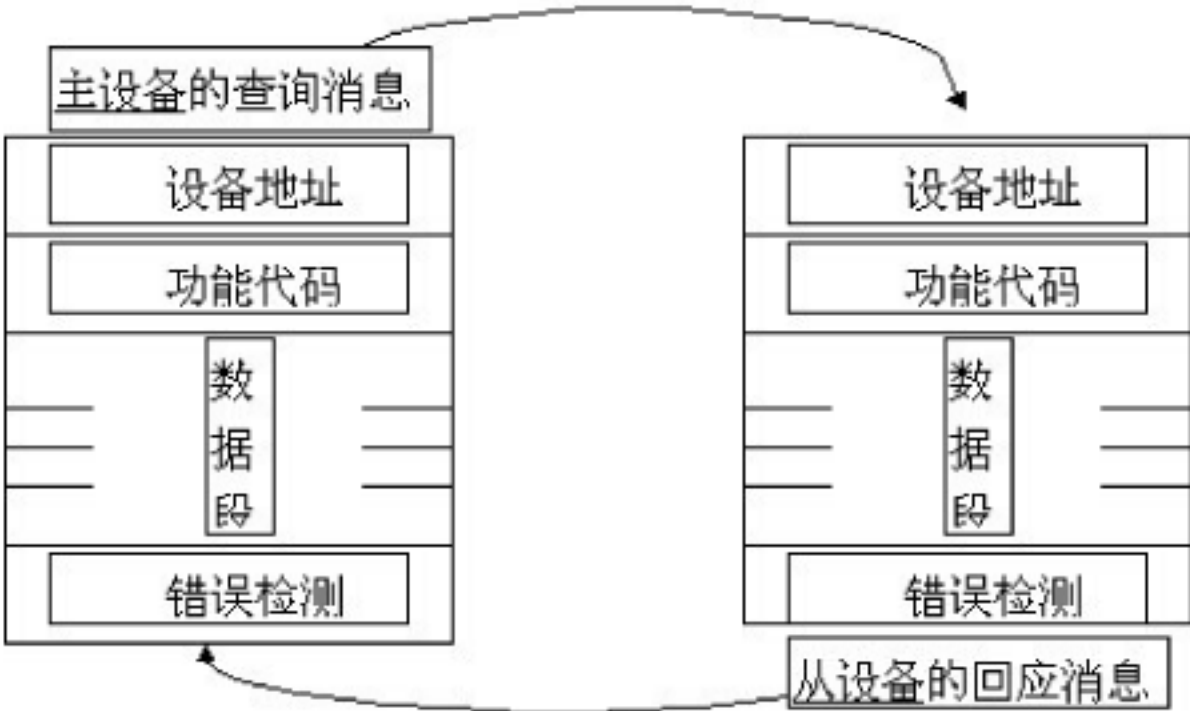


图1 主-从 查询-响应周期表

二、两种传输方式

控制器能设置为两种传输模式（ASCII或RTU）中的任何一种在标准的Modbus网络通信。用户选择想要的模式，包括串口通信参数（波特率、校验方式等），在配置每个控制器的时候，在一个Modbus网络上的所有设备都必须选择相同的传输模式和串口参数。

ASCII模式

...	地址	功能代码	数据数量	数据1	...	数据n	LRC高字节	LRC低字节	回车	换行
-----	----	------	------	-----	-----	-----	--------	--------	----	----

RTU模式

地址	功能代码	数据数量	数据1	...	数据n	CRC高字节	CRC低字节
----	------	------	-----	-----	-----	--------	--------

所选的ASCII或RTU方式仅适用于标准的Modbus网络，它定义了在这些网络上连续传输的消息段的每一位，以及决定怎样将信息打包成消息域和如何解码。

在其它网络上（象MAP和Modbus Plus）Modbus消息被转成与串行传输无关的帧。

1、ASCII模式

当控制器设为在Modbus网络上以ASCII（美国标准信息交换代码）模式通信，在消息中的每个8Bit字节都作为两个ASCII字符发送。这种方式的主要优点是字符发送的时间间隔可达到1秒而不产生错误。

代码系统

- 十六进制，ASCII字符0...9, A...F
- 消息中的每个ASCII字符都是一个十六进制字符组成

每个字节的位

- 1个起始位
- 7个数据位，最小的有效位先发送
- 1个奇偶校验位，无校验则无
- 1个停止位（有校验时），2个Bit（无校验时）

错误检测域

- LRC(纵向冗长检测)

2、RTU模式

当控制器设为在Modbus网络上以RTU（远程终端单元）模式通信，在消息中的每个8Bit字节包含两个4Bit的十六进制字符。这种方式的主要优点是：在同样的波特率下，可比ASCII方式传送更多的数据。

代码系统

- 8位二进制，十六进制数0...9, A...F
- 消息中的每个8位域都是一个两个十六进制字符组成

每个字节的位

- 1个起始位
- 8个数据位，最小的有效位先发送
- 1个奇偶校验位，无校验则无
- 1个停止位（有校验时），2个Bit（无校验时）

错误检测域

- CRC(循环冗长检测)

三、Modbus消息帧

两种传输模式中（ASCII或RTU），传输设备以将Modbus消息转为有起点和终点的帧，这就允许接收的设备在消息起始处开始工作，读地址分配信息，判断哪一个设备被选中（广播方式则传给所有设备），判知何时信息已完成。部分的消息也能侦测到并且错误能设置为返回结果。

1、ASCII帧

使用ASCII模式，消息以冒号（:）字符（ASCII码 3AH）开始，以回车换行符结束（ASCII码 0DH, 0AH）。

其它域可以使用的传输字符是十六进制的0...9, A...F。网络上的设备不断侦测“:”字符，当有一个冒号接收到时，每个设备都解码下个域（地址域）来判断是否发给自己的。

消息中字符间发送的时间间隔最长不能超过1秒，否则接收的设备将认为传输错误。一个典型消息帧如下所示：

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

图2 ASCII消息帧

2、RTU帧

使用RTU模式，消息发送至少要以3.5个字符时间的停顿间隔开始。在网络波特率下多样的字符时间，这是最容易实现的(如下图的T1-T2-T3-T4所示)。传输的第一个域是设备地址。可以使用的传输字符是十六进制的0...9, A...F。网络设备不断侦测网络总线，包括停顿间隔时间内。当第一个域（地址域）接收到，每个设备都进行解码以判断是否发往自己的。在最后一个传输字符之后，一个至少3.5个字符时间的停顿标定了消息的结束。一个新的消息可在此停顿后开始。

整个消息帧必须作为一连续的流转输。如果在帧完成之前有超过1.5个字符时间的停顿时间，接收设备将刷新不完整的消息并假定下一字节是一个新消息的地址域。同样地，如果一个新消息在小于3.5个字符时间内接着前个消息开始，接收的设备将认为它是前一消息的延续。这将导致一个错误，因为在最后的CRC域的值不可能是正确的。一典型的消息帧如下所示：

起始位	设备地址	功能代码	数据	CRC校验	结束符
T1-T2-T3-T4	8Bit	8Bit	n个8Bit	16Bit	T1-T2-T3-T4

图3 RTU消息帧

3、地址域

消息帧的地址域包含两个字符（ASCII）或8Bit（RTU）。可能的从设备地址是0...247（十进

制)。单个设备的地址范围是1...247。主设备通过将要联络的从设备的地址放入消息中的地址域来选通从设备。当从设备发送回应消息时,它把自己的地址放入回应的地址域中,以便主设备知道是哪一个设备作出回应。

地址0是用作广播地址,以使所有的从设备都能认识。当Modbus协议用于更高水准的网络,广播可能不允许或以其它方式代替。

4、如何处理功能域

消息帧中的功能代码域包含了两个字符(ASCII)或8Bits(RTU)。可能的代码范围是十进制的1...255。当然,有些代码是适用于所有控制器,有此是应用于某种控制器,还有些保留以备后用。

当消息从主设备发往从设备时,功能代码域将告之从设备需要执行哪些行为。例如去读取输入的开关状态,读一组寄存器的数据内容,读从设备的诊断状态,允许调入、记录、校验在从设备中的程序等。

当从设备回应时,它使用功能代码域来指示是正常回应(无误)还是有某种错误发生(称作异议回应)。对正常回应,从设备仅回应相应的功能代码。对异议回应,从设备返回一等同于正常代码的代码,但最重要的位置为逻辑1。

例如:一从主设备发往从设备的消息要求读一组保持寄存器,将产生如下功能代码:

0 0 0 0 0 0 1 1 (十六进制03H)

对正常回应,从设备仅回应同样的功能代码。对异议回应,它返回:

1 0 0 0 0 0 1 1 (十六进制83H)

除功能代码因异议错误作了修改外,从设备将一独特的代码放到回应消息的数据域中,这能告诉主设备发生了什么错误。

主设备应用程序得到异议的回应后,典型的处理过程是重发消息,或者诊断发给从设备的消息并报告给操作员。

5、数据域

数据域是由两个十六进制数集合构成的,范围00...FF。根据网络传输模式,这可以由一对ASCII字符组成或由一RTU字符组成。

从主设备发给从设备消息的数据域包含附加的信息:从设备必须用于进行执行由功能代码所定义的所为。这包括了象不连续的寄存器地址,要处理项的数目,域中实际数据字节数。

例如,如果主设备需要从设备读取一组保持寄存器(功能代码03),数据域指定了起始寄存器以及要读的寄存器数量。如果主设备写一组从设备的寄存器(功能代码10十六进制),数据域则指明了要写的起始寄存器以及要写的寄存器数量,数据域的数据字节数,要写入寄存器的数据。

如果没有错误发生,从从设备返回的数据域包含请求的数据。如果有错误发生,此域包含一异议代码,主设备应用程序可以用来判断采取下一步行动。

在某种消息中数据域可以是不存在的(0长度)。例如,主设备要求从设备回应通信事件记录(功能代码0B十六进制),从设备不需任何附加的信息。

6、错误检测域

标准的Modbus网络有两种错误检测方法。错误检测域的内容视所选的检测方法而定。

ASCII

当选用ASCII模式作字符帧,错误检测域包含两个ASCII字符。这是使用LRC(纵向冗长检测)方法对消息内容计算得出的,不包括开始的冒号符及回车换行符。LRC字符附加在回车换行符前面。

RTU

当选用RTU模式作字符帧,错误检测域包含一16Bits值(用两个8位的字符来实现)。错误检测域的内容是通过消息内容进行循环冗长检测方法得出的。CRC域附加在消息的最后,添加时先是低字节然后是高字节。故CRC的高位字节是发送消息的最后一个字节。

7、字符的连续传输

当消息在标准的Modbus系列网络传输时，每个字符或字节以如下方式发送（从左到右）：

最低有效位... 最高有效位

使用ASCII字符帧时，位的序列是：

有奇偶校验

起始位	1	2	3	4	5	6	7	奇偶位	停止位
-----	---	---	---	---	---	---	---	-----	-----

无奇偶校验

起始位	1	2	3	4	5	6	7	停止位	停止位
-----	---	---	---	---	---	---	---	-----	-----

图4. 位顺序（ASCII）

使用RTU字符帧时，位的序列是：

有奇偶校验

起始位	1	2	3	4	5	6	7	8	奇偶位	停止位
-----	---	---	---	---	---	---	---	---	-----	-----

无奇偶校验

起始位	1	2	3	4	5	6	7	8	停止位	停止位
-----	---	---	---	---	---	---	---	---	-----	-----

图4. 位顺序（RTU）

四、错误检测方法

标准的Modbus串行网络采用两种错误检测方法。奇偶校验对每个字符都可用，帧检测（LRC或CRC）应用于整个消息。它们都是在消息发送前由主设备产生的，从设备在接收过程中检测每个字符和整个消息帧。

用户要给主设备配置一预先定义的超时时间间隔，这个时间间隔要足够长，以使任何从设备都能作为正常反应。如果从设备测到一传输错误，消息将不会接收，也不会向主设备作出回应。这样超时事件将触发主设备来处理错误。发往不存在的从设备的地址也会产生超时。

1、奇偶校验

用户可以配置控制器是奇或偶校验，或无校验。这将决定了每个字符中的奇偶校验位是如何设置的。如果指定了奇或偶校验，“1”的位数将算到每个字符的位数中（ASCII模式7个数据位，RTU中8个数据位）。例如RTU字符帧中包含以下8个数据位：

1 1 0 0 0 1 0 1

整个“1”的数目是4个。如果便用了偶校验，帧的奇偶校验位将是0，便得整个“1”的个数仍是4个。如果便用了奇校验，帧的奇偶校验位将是1，便得整个“1”的个数是5个。

如果没有指定奇偶校验位, 传输时就没有校验位, 也不进行校验检测。代替一附加的停止位填充至要传输的字符帧中。

2、LRC检测

使用ASCII模式, 消息包括了一基于LRC方法的错误检测域。LRC域检测了消息域中除开始的冒号及结束的回车换行号外的内容。

LRC域是一个包含一个8位二进制值的字节。LRC值由传输设备来计算并放到消息帧中, 接收设备在接收消息的过程中计算LRC, 并将它和接收到消息中LRC域中的值比较, 如果两值不等, 说明有错误。

LRC方法是将消息中的8Bit的字节连续累加, 丢弃了进位。

LRC简单函数如下:

```
static unsigned char LRC(auchMsg, usDataLen)
unsigned char *auchMsg ; /* 要进行计算的消息 */
unsigned short usDataLen ; /* LRC 要处理的字节的数量*/
{ unsigned char uchLRC = 0 ; /* LRC 字节初始化 */
while (usDataLen--) /* 传送消息 */
    uchLRC += *auchMsg++ ; /* 累加*/
return ((unsigned char) (~((char_uchLRC))) ) ;
}
```

3、CRC检测

使用RTU模式, 消息包括了一基于CRC方法的错误检测域。CRC域检测了整个消息的内容。

CRC域是两个字节, 包含一16位的二进制值。它由传输设备计算后加入到消息中。接收设备重新计算收到消息的CRC, 并与接收到的CRC域中的值比较, 如果两值不同, 则有误。

CRC是先调入一值是全“1”的16位寄存器, 然后调用一过程将消息中连续的8位字节各当前寄存器中的值进行处理。仅每个字符中的8Bit数据对CRC有效, 起始位和停止位以及奇偶校验位均无效。

CRC产生过程中, 每个8位字符都单独和寄存器内容相或(OR), 结果向最低有效位方向移动, 最高有效位以0填充。LSB被提取出来检测, 如果LSB为1, 寄存器单独和预置的值或一下, 如果LSB为0, 则不进行。整个过程要重复8次。在最后一位(第8位)完成后, 下一个8位字节又单独和寄存器的当前值相或。最终寄存器中的值, 是消息中所有的字节都执行之后的CRC值。

CRC添加到消息中时, 低字节先加入, 然后高字节。

CRC简单函数如下:

```
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg ; /* 要进行CRC校验的消息 */
unsigned short usDataLen ; /* 消息中字节数 */
{
    unsigned char uchCRCHi = 0xFF ; /* 高CRC字节初始化 */
```



```

unsigned char uchCRCLo = 0xFF ; /* 低CRC 字节初始化 */
unsigned uIndex ; /* CRC循环中的索引 */
while (usDataLen--) /* 传输消息缓冲区 */
{
    uIndex = uchCRCHi ^ *puchMsgg++ ; /* 计算CRC */
    uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
    uchCRCLo = auchCRCLo[uIndex] ;
}
return (uchCRCHi << 8 | uchCRCLo) ;
}

/* CRC 高位字节值表 */
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,
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, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;

/* CRC低位字节值表*/
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
} ;

ModBus网络是一个工业通信系统，由带智能终端的可编程序控制器和计算机通过公用线路或局部专用线路连接而成。其系统结构既包括硬件、亦包括软件。它可应用于各种数据采集和过程监控。下表1是ModBus的功能码定义。

表1 ModBus功能码
01 READ COIL STATUS
02 READ INPUT STATUS
03 READ HOLDING REGISTER
04 READ INPUT REGISTER
05 WRITE SINGLE COIL
06 WRITE SINGLE REGISTER
15 WRITE MULTIPLE COIL
16 WRITE MULTIPLE REGISTER

ModBus网络只是一个主机，所有通信都由他发出。网络可支持247个之多的远程从属控制器，但实际所支持的从机数要由所用通信设备决定。采用这个系统，各PC可以和中心主机交换信息而不影响各PC执行本身的控制任务。表2是ModBus各功能码对应的数据类型。

表2 Modbus功能与数据类型对应表

代码	功能	数据类型
01	读	位
02	读	位
03	读	整型、字符型、状态字、浮点型
04	读	整型、状态字、浮点型
05	写	位
06	写	整型、字符型、状态字、浮点型
08	N/A	重复“回路反馈”信息
15	写	位
16	写	整型、字符型、状态字、浮点型
17	读	字符型

(1) ModBus的传输方式

在ModBus系统中有2种传输模式可选择。这2种传输模式与从机PC通信的能力是同等的。选择时应视所用ModBus主机而定，每个ModBus系统只能使用一种模式，不允许2种模式混用。一种模式是ASCII（美国信息交换码），另一种模式是RTU（远程终端设备）这两种模式的定义见表3

表3 ASCII和RTU传输模式的特性

特性		ASCII(位)	RTU(位)
编码系统		十六进制（使用ASCII可打印字符：0~9，A~F）	二进制
每一个字符的位数	开始位	1位	1位
	数据位（最低有效位第一位）	7位	8位
	奇偶校验（任选）	1位（此位用于奇偶校验，无校验则无该位）	1位（此位用于奇偶校验，无校验则无该位）
	停止位	1或2位	1或2位
	错误校验	LRC（即纵向冗余校验）	CRC（即循环冗余校验）

ASCII可打印字符便于故障检测，而且对于用高级语言（如Fortan）编程的主计算机及主PC很适宜。RTU则适用于机器语言编程的计算机和PC主机。

用RTU模式传输的数据是8位二进制字符。如欲转换为ASCII模式，则每个RTU字符首先应分为高位和低位两部分，这两部分各含4位，然后转换成十六进制等量值。用以构成报文的ASCII字符都是十六进制字符。ASCII模式使用的字符虽是RTU模式的两倍，但ASCII数据的译码和处理更为容易一些，此外，用RTU模式时报文字符必须以连续数据流的形式传送，用ASCII模式，字符之间可产生长达1s的间隔，以适应速度较快的机器。

(2) ModBus的数据校验方式

CRC-16（循环冗余错误校验）

CRC-16错误校验程序如下：报文（此处只涉及数据位，不指起始位、停止位和任选的奇偶校验位）被看作是一个连续的二进制，其最高有效位（MSB）首选发送。报文先与X↑16相乘（左移16位），然后看X↑16+X↑15+X↑2+1除，X↑16+X↑15+X↑2+1可以表示为二进制数11000000000000101。整数商位忽略不记，16位余数加入该报文（MSB先发送），成为2个CRC校验字节。余数中的1全部初始化，以免所有的零成为一条报文被接收。经上述处理而含有CRC字节的

报文, 若无错误, 到接收设备后再被同一多项式 ($X^{16}+X^{15}+X^2+1$) 除, 会得到一个零余数 (接收设备核验这个CRC字节, 并将其与被传送的CRC比较)。全部运算以2为模 (无进位)。

习惯于成串发送数据的设备会首选送出字符的最右位 (LSB-最低有效位)。而在生成CRC情况下, 发送首位应是被除数的最高有效位MSB。由于在运算中不用进位, 为便于操作起见, 计算CRC时设MSB在最右位。生成多项式的位序也必须反过来, 保持一致。多项式的MSB略去不记, 因其只对商有影响而不影响余数。

生成CRC-16校验字节的步骤如下:

- ①装如一个16位寄存器, 所有数位均为1。
- ②该16位寄存器的高位字节与开始8位字节进行“异或”运算。运算结果放入这个16位寄存器。
- ③把这个16寄存器向右移一位。
- ④若向右 (标记位) 移出的数位是1, 则生成多项式1010000000000001和这个寄存器进行“异或”运算; 若向右移出的数位是0, 则返回③。
- ⑤重复③和④, 直至移出8位。
- ⑥另外8位与该十六位寄存器进行“异或”运算。
- ⑦重复③~⑥, 直至该报文所有字节均与16位寄存器进行“异或”运算, 并移位8次。
- ⑧这个16位寄存器的内容即2字节CRC错误校验, 被加到报文的最高有效位。

另外, 在某些非ModBus通信协议中也经常使用CRC16作为校验手段, 而且产生了一些CRC16的变种, 他们是使用CRC16多项式 $X^{16}+X^{15}+X^2+1$, 单首次装入的16位寄存器为0000; 使用CRC16的反序 $X^{16}+X^{14}+X^1+1$, 首次装入寄存器值为0000或FFFFH。

LRC (纵向冗余错误校验)

LRC错误校验用于ASCII模式。这个错误校验是一个8位二进制数, 可作为2个ASCII十六进制字节传送。把十六进制字符转换成二进制, 加上无循环进位的二进制字符和二进制补码结果生成LRC错误校验 (参见图)。这个LRC在接收设备进行核验, 并与被传送的LRC进行比较, 冒号 (:)、回车符号 (CR)、换行字符 (LF) 和置入的其他任何非ASCII十六进制字符在运算时忽略不计。

版权所有 (C) 1999 [珠海市水业科技有限公司]。保留所有权利。

修订日期: 2003年05月26日。