

1、Modbus 协议简介

Modbus协议是一种已广泛应用于当今工业控制领域的通用通讯协议。通过此协议，控制器相互之间、或控制器经由网络（如以太网）可以和其它设备之间进行通信。Modbus协议使用的是主从通讯技术，即由主设备主动查询和操作从设备。一般将主控设备方所使用的协议称为Modbus Master，从设备方使用的协议称为Modbus Slave。典型的主设备包括工控机和工业控制器等；典型的从设备如PLC可编程控制器等。

Modbus通讯物理接口可以选用串口（包括RS232、RS485和RS422），也可以选择以太网口。其通信遵循以下的过程：

- 主设备向从设备发送请求
- 从设备分析并处理主设备的请求，然后向主设备发送结果
- 如果出现任何差错，从设备将返回一个异常功能码

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，如何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。

Modbus的工作方式是请求/应答，每次通讯都是主站先发送指令，可以是广播，或是向特定从站的单播；从站响应指令，并按要求应答，或者报告异常。当主站不发送请求时，从站不会自己发出数据，从站和从站之间不能直接通讯。

Modbus协议是应用层（协议层）报文传输协议，它定义了一个与物理层无关的协议数据单元（PDU），即PDU=功能码+数据域，功能码1byte，数据域不确定。

Modbus协议能够应用在不同类型的总线或网络。对应不同的总线或网络，Modbus协议引入一些附加域映射成应用数据单元（ADU），即ADU=附加域+PDU，例如modbus tcp/ip----- ADU=MBAP+ADU。

2、Modbus 通讯方式

2.1、Modbus三种通讯方式

Modbus有下列三种通信方式：

- (1)、以太网：对应的通信模式是**Modbus TCP/IP**
- (2)、异步串行传输（各种介质如有线RS-232-/422/485/；光纤、无线等）：对应的通信模式是**Modbus RTU**或**Modbus ASCII**
- (3)、高速令牌传递网络：对应的通信模式是**Modbus PLUS**

Modbus RTU和Modbus ASCII协议应用于串口链接（RS232、RS485、RS422），Modbus tcp/ip协议应用于以太网链接。

2.2、在Modbus网络上传输

标准的Modbus口是使用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，如果是以广播方式查询的，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。

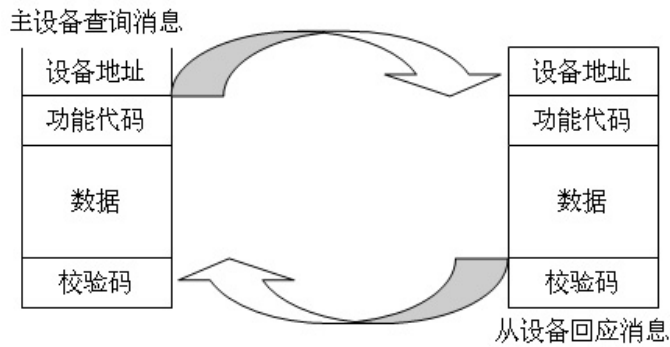
从设备回应消息也由Modbus协议构成，包括确认要行动的域、任何要返回的数据、和一错误检测域。如果在消息接收过程中发生一错误，或从设备不能执行其命令，从设备将建立一错误消息并把它作为回应发送出去。

2.3、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。

2.4、查询---回应



(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。

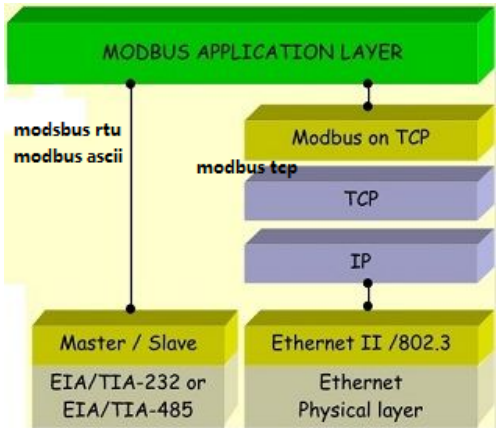
3、三种通讯方式的报文格式

Modbus协议的报文（或帧）的基本格式是：**表头 + 功能码 + 数据区 + 校验码**

功能码和数据区在不同类型的网络都是固定不变的，表头和校验码则因网络底层的实现方式不同而有所区别。表头包含了从站的地址，功能码告诉从站要执行何种功能，数据区是具体的信息。

对于不同类型的网络，Modbus的协议层实现是一样的，区别在于下层的实现方式，常见的有TCP/IP和串行通讯两种。

Modbus TCP基于以太网和TCP/IP协议，Modbus RTU和Modbus ASCII则是使用异步串行传输（通常是RS-232/422/485）。



如图所示，**串行传输**的物理层是RS-485或RS-232，数据链路层是Modbus的串行传输协议；**Modbus TCP传输**的1、2、3、4层实现和日常所见的以太网、因特网一样，Modbus默认采用的TCP端口号是502。

3.1、以太网 (modbus tcp/ip)

对于Modbus TCP而言，主站通常称为Client，从站称为Server；而对于Modbus RTU和Modbus ASCII来说，主站是Master，从站是Slave。

ModbusTCP的数据帧可分为两部分：**ADU=MBAP+PDU = MBAP + 功能码 + 数据域**，**MBAP 7byte，功能码1byte，数据域不确定，由具体功能决定。**

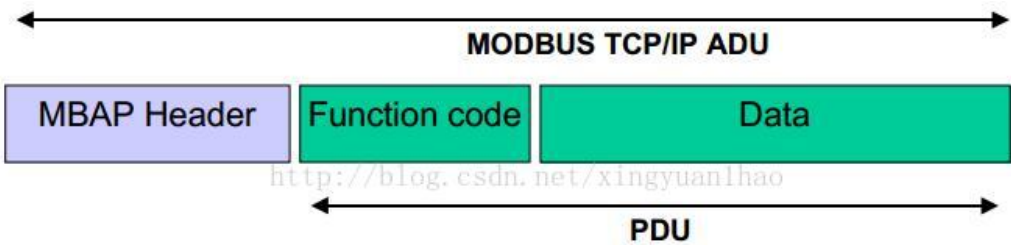


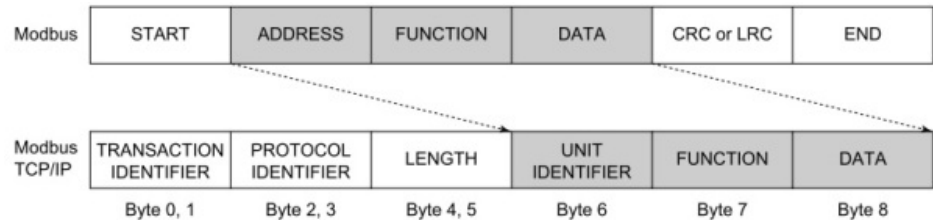
Figure 3: MODBUS request/response over TCP/IP

MBAP为报文头，长度为7字节，组成如下：

事务处理标识	协议标识	长度	单元标识符
2byte	2byte	2byte	1byte

内容	解释
事务处理标识	可以理解为报文的序列号，一般每次通信之后就要加1以区别不同的通信数据报文
协议标识符	00 00表示ModbusTCP协议
长度	表示接下来的数据长度，单位为字节
单元标识符	可以理解的设备地址

下图说明了Modbus TCP的改动：



- (1)、取消了校验位。数据链路层上就进行了CRC-32的校验，TCP/IP是面向连接的可靠性的协议，因此没必要再加上校验位。
- (2)、Slave地址换成了Unit Identifier。当网络里的设备全是使用TCP/IP，这个地址是没有意义的，因为IP就能进行路由寻址。如果网络里还有串行通讯的设备，则需要网关来实现Modbus TCP到Modbus RTU或ASCII之间的协议转换，这时用Unit Identifier来标识网关后面的每个串行通讯设备。
- (3)、Length是指后面的字节总数。实际上数据区的长度是能确定的，有的功能码就可以确定数据区的长度，有的功能码虽不能确定数据区长度，但是数据区有字节计数，参见上文举的从站应答的例子。表头增加的Length是为了应对有些情况下TCP/IP协议会将应用层的数据拆包传输。
- (4)、Transaction Identifier和Protocol Identifier由Client生成，Server的响应将复制这些参数。

3.1.1、modbus tcp/ip通信方式

Modbus设备可分为主站(poll)和从站(slave)。主站只有一个，从站有多个，主站向各从站发送请求帧，从站给予响应。在使用TCP通信时，主站为client端，主动建立连接；从站为server端，等待连接。

- 主站请求：功能码+数据
- 从站正常响应：请求功能码+响应数据

- 从站异常响应：异常功能码+异常码，其中异常功能码即将请求功能码的最高有效位置1，异常码指示差错类型
- 注意：需要超时管理机制，避免无期限的等待可能不出现的应答

IANA（Internet Assigned Numbers Authority，互联网编号分配管理机构）给Modbus协议赋予TCP端口号为**502**，这是目前在仪表与自动化行业中唯一分配到的端口号。

3.1.2、通信过程

- A、connect 建立TCP连接
- B、准备Modbus报文
- C、使用send命令发送报文
- D、在同一连接下等待应答
- E、使用recv命令读取报文，完成一次数据交换
- F、通信任务结束时，关闭TCP连接

3.2、异步串行传输的两种传输方式（modbus RTU和modbus ASCII）

异步串行传输时， 控制器可以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：

起始位	设备地址	功能代码	数据数量	数据	LRC高字节	LRC低字节	结束符
:	2*8bit	2*8bit	n	n*8bit	8bit	8bit	CR、LF（回车、换行）

RTU模式：

起始位	设备地址	功能代码	数据数量	数据	CRC低字节	CRC高字节	结束符
无	8bit	8bit	n	n*8bit	8bit	8bit	无

3.2.1、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)

3.2.2、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)

3.2.3、RTU和ASCII的区别

- (1)、RTU模式下，一个字节的数 据，传输的就是一个字节。ASCII模式下，同样一个字节数据用了两个字节来传输。例如，要传输数字0x5B，RTU传输的是0101 1011（二进制），而ASCII传输的是00110101和01000010。可见，ASCII传输的速率是RTU的一半。
- (2)、ASCII模式采用LRC校验，RTU模式采用16位CRC校验。
- (3)、ASCII有开始标记和结束标记，RTU没有

4、Modbus RTU和Modbus ASCII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。

4.1、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

4.2、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传输。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

4.3、两种帧的分析

(1)、地址域

消息帧的地址域包含两个字节（ASCII）或一个字节（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。

(2)、如何处理功能域

消息帧中的功能代码域包含了两个字节（ASCII）或一个字节（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 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 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。

(3)、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。

(4)、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将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。

(5)、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
-----	---	---	---	---	---	---	---	-----	-----

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
-----	---	---	---	---	---	---	---	---	-----	-----

5、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。

5.1、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，使得整个“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，使得整个“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。


5.2、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：



```
1  ///C代码
2  static unsigned char LRC(auchMsg,usDataLen)
3
4  unsigned char *auchMsg ; /* 要进行计算的消息 */
5  unsigned short usDataLen ; /* LRC 要处理的字节的数量*/
6
7  {
8      unsigned char uchLRC = 0 ; /* LRC 字节初始化 */
9      while (usDataLen--) /* 传送消息 */
10         uchLRC += *auchMsg++ ; /* 累加*/
11
12     return ((unsigned char) (~((char_uchLRC))) ) ;
13 }
```



5.3、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。

计算CRC码的步骤为：

- 预置16位寄存器为十六进制FFFF（即全为1）。称此寄存器为CRC寄存器；
- 把第一个8位数据与16位CRC寄存器的低位相异或，把结果放于CRC寄存器；
- 把寄存器的内容右移一位(朝低位)，用0填补最高位，检查最低位；
- 如果最低位为0：重复第3步(再次移位)；如果最低位为1：CRC寄存器与多项式A001（1010 0000 0000 0001）进行异或；
- 重复步骤3和4，直到右移8次，这样整个8位数据全部进行了处理；
- 重复步骤2到步骤5，进行下一个8位数据的处理；
- 最后得到的CRC寄存器即为CRC码。



```
1 /**
2  * @brief modbus rtu校验
3  * @param p_data: 要校验的数据的地址
4  *       data_len: 要校验数据的长度(字节)
5  *       data_crc: 数据的校验码
6  * @retval 无
7  */
8 void CRC_Checkout_16(uint8_t *p_data,uint32_t data_len,uint8_t *data_crc)
9 {
10     uint16_t wcrcl = 0xFFFF;
11     uint8_t temp;
12     uint32_t i=0,j=0;
13     for(i=0;i<data_len;i++)
14     {
15         temp = *p_data & 0X00FF;
16         p_data++;
17         wcrcl = wcrcl^temp;
18         for(j=0;j<8;j++)
19         {
20             if(wcrcl & 0X0001)
21             {
22                 wcrcl>>=1;
23                 wcrcl^=0XA001;
24             }
25             else
26             {
27                 wcrcl>>=1;
28             }
29         }
30     }
31     temp=wcrcl;
32
33     data_crc[0]=wcrcl;
34     data_crc[1]=wcrcl>>8;
35
36     return ;
37 }
```



```
1 /* CRC 高位字节值表 */
2 static unsigned char auchCRCHi[] =
3 {
4     0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
5     0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
6     0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
7     0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
8     0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
9     0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
10    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
11    0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
12    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
13    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
14    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
15    0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
16    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
17    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
18    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
19    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
20    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
21    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
22    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
23    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
24    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
25    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
26    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
27    0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
28    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
29    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
30 } ;
31
32 /* CRC低位字节值表*/
33 static char auchCRCLo[] =
34 {
35     0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
36     0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
37     0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
38     0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
39     0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
40     0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
41     0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
42     0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
43     0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
44     0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
45     0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
46     0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
```



```
47 0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
48 0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
49 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
50 0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
51 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
52 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
53 0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
54 0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
55 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
56 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
57 0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
58 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
59 0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
60 0x43, 0x83, 0x41, 0x81, 0x80, 0x40
61 } ;
```



CRC冗余码在线计算网址：<http://www.ip33.com/crc.html>

6、Modbus的四种数据类型

输出线圈：大小只有1bit，ON或OFF，可读可写，既可以是一个输出量输出点，也可以是数字量输入点，有效的地址范围是1-9999。

输入离散量：大小只有1bit，ON或OFF，只读，即数字量输出点，有效地址范围是10001-19999。

输入寄存器：16bit的寄存器，只读，可以用作模拟量或16位打包输入点，有效地址范围是30001-39999。

保持寄存器：16bit的寄存器，可读可写，既可以是一个模拟量或16位打包输入点，也可以是模拟量或16位打包输出点，有效地址范围是40001-49999。

在PLC或DCS上用点名标记不同的变量，在Modbus则以数据地址来标记每个点。以上所说的地址都是参考地址，而不是实际的物理地址。上述的地址是在设备中的地址，按照PLC的习惯从1开始递增，而Modbus报文中是从0开始递增。例如地址偏移为4、5、6的Holding Register，其实是指参考地址是40005、40006、40007的寄存器。

(1)、Coil和Register

Modbus中定义的两​​种数据类型：Coil是位（bit）变量；Register是整型（Word，即16-bit）变量。

(2)、Slave和Master与Server和Client

同一种设备在不同领域的不同叫法。

Slave：工业自动化用语；响应请求；

Master：工业自动化用语；发送请求；

Server：IT用语；响应请求；

Client：IT用语；发送请求；

在Modbus中，Slave和Server意思相同，Master和Client意思相同。

(3) Modbus数据模型

Modbus中，数据可以分为两大类，分别为位变量(Coil)和整形变量(Register)，每一种数据，根据读写方式的不同，又可细分为两种（只读，读写）。

Modbus四种数据类型：

Discretes Input	位变量	只读
Coils	位变量	读写
Input Registers	16-bit整型	只读
Holding Registers	16-bit整型	读写

(4)、Modbus地址范围对应表

设备地址	Modbus地址	描述	功能	R/W
1~10000	address-1	Coils (Output)	0	R/W
10001~20000	address-10001	Discrete Inputs	01	R
30001~40000	address-30001	Input Registers	04	R
40001~50000	address-40001	Holding Registers	03	R/W

(5)、Modbus变量地址

映射地址	Function Code	地址类型	R/W	描述

0xxxx	01,05,15	Coil	R/W	---
1xxxx	02	离散输入	R	---
2xxxx	03,04,06,16	浮点寄存器	R/W	两个连续16-bit寄存器表示一个浮点数 (IEEE754)
3xxxx	04	输入寄存器	R	每个寄存器表示一个16-bit无符号整数 (0~65535)
4xxxx	03,06,16	保持寄存器	R/W	---
5xxxx	03,04,06,16	ASCII字符	R/W	每个寄存器表示两个ASCII字符

7、modbus协议的功能码

地址码	功能码	数据区	错误校验码
8bit	8bit	n*8bit	2*8bit

Modbus的操作对象有四种：线圈、离散输入、输入寄存器、保持寄存器。

对象	含义
线圈	PLC的输出位，开关量，在Modbus中可读可写
离散量	PLC的输入位，开关量，在Modbus中只读
输入寄存器	PLC中只能从模拟量输入端改变的寄存器，在Modbus中只读
保持寄存器	PLC中用于输出模拟量信号的寄存器，在Modbus中可读可写

根据对象的不同，Modbus的功能码有：

功能码	名称	功能
01	读线圈状态	读位（读N个bit）---读从机线圈寄存器，位操作
02	读输入离散量	读位（读N个bit）---读离散输入寄存器，位操作
03	读多个寄存器	读整型、字符型、状态字、浮点型（读N个words）---读保持寄存器，字节操作
04	读输入寄存器	读整型、状态字、浮点型（读N个words）---读输入寄存器，字节操作
05	写单个线圈	写位（写一个bit）---写线圈寄存器，位操作
06	写单个保持寄存器	写整型、字符型、状态字、浮点型（写一个word）---写保持寄存器，字节操作
07	读取异常状态	取得8个内部线圈的通断状态，这8个线圈的地址由控制器决定，用户逻辑可以将这些线圈定义，以说明从机状态，短报文适宜于迅速读取状态
08	回送诊断校验	把诊断校验报文送从机，以对通信处理进行评鉴
09	编程（只用于484）	使主机模拟编程器作用，修改PC从机逻辑
0A	控询（只用于484）	可使主机与一台正在执行长程序任务从机通信，探询该从机是否已完成其操作任务，仅在含有功能码9的报文发送后，本功能码才发送
0B	读取事件计数	可使主机发出单询问，并随即判定操作是否成功，尤其是该命令或其他应答产生通信错误时
0C	读取通讯事件记录	可是主机检索每台从机的ModBus事务处理通信事件记录。如果某项事务处理完成，记录会给出有关错误
0D	编程（184/384/484/584）	可使主机模拟编程器功能修改PC从机逻辑
0E	探询（184/384/484/584）	可使主机与正在执行任务的从机通信，定期控询该从机是否已完成其程序操作，仅在含有功能13的报文发送后，本功能码才得发送
0F	写多个线圈	可以写多个线圈---强置一串连续逻辑线圈的通断

10	写多个保持寄存器	写多个保持寄存器---把具体的二进制值装入一串连续的保持寄存器
11	报告从机标识	可使主机判断编址从机的类型及该从机运行指示灯的状态
12	(884和MICRO84)	可使主机模拟编程功能，修改PC状态逻辑
13	重置通信链路	发生非可修改错误后，是从机复位于已知状态，可重置顺序字节
14	读取通用参数(584L)	显示扩展存储文件中的数据信息
15	写入通用参数(584L)	把通用参数写入扩展存储文件
16~40	保留做扩展功能备用	
41~48	保留以备用户功能所用	留作用户功能的扩展编码
49~77	非法功能	
78~7F	保留	留作内部作用
80~FF	保留	用于异常应答

常用功能码如下：

功能码	名称	功能	对应的地址类型
01	读线圈状态	读位（读N个bit）---读从机线圈寄存器，位操作	0x
02	读输入离散量	读位（读N个bit）---读离散输入寄存器，位操作	1x
03	读多个寄存器	读整型、字符型、状态字、浮点型（读N个words）---读保持寄存器，字节操作	4X
04	读输入寄存器	读整型、状态字、浮点型（读N个words）---读输入寄存器，字节操作	3x
05	写单个线圈	写位（写一个bit）---写线圈寄存器，位操作	0x
06	写单个保持寄存器	写整型、字符型、状态字、浮点型（写一个word）---写保持寄存器，字节操作	4x
0F	写多个线圈	写位（写n个bit）---强置一串连续逻辑线圈的通断	0x
10	写多个保持寄存器	写整形、字符型、状态字、浮点型（写n个word）---把具体的二进制值装入一串连续的保持寄存器	4x

8、PDU详细结构

(1)、0x01：读线圈

在从站中读1~2000个连续线圈状态，ON=1,OFF=0

- 请求：MBAP 功能码 起始地址H 起始地址L 数量H 数量L（共12字节）
- 响应：MBAP 功能码 数据长度 数据（一个地址的数据为1位）
- 如：在从站0x01中，读取开始地址为0x0002的线圈数据，读0x0008位-----00 01 00 00 00 06 01 01 00 02 00 08
- 回：数据长度为0x01个字节，数据为0x01，第一个线圈为ON，其余为OFF-----00 01 00 00 00 04 01 01 01 01

(2)、0x05：写单个线圈

将从站中的一个输出写成ON或OFF，0xFF00请求输出为ON,0x000请求输出为OFF

- 请求：MBAP 功能码 输出地址H 输出地址L 输出值H 输出值L（共12字节）
- 响应：MBAP 功能码 输出地址H 输出地址L 输出值H 输出值L（共12字节）
- 如：将地址为0x0003的线圈设为ON-----00 01 00 00 00 06 01 05 00 03 FF 00
- 回：写入成功-----00 01 00 00 00 06 01 05 00 03 FF 00

(3)、0x0F：写多个线圈

将一个从站中的一个线圈序列的每个线圈都强制为ON或OFF，数据域中置1的位请求相应输出位ON，置0的位请求响应输出为OFF

- 请求：MBAP 功能码 起始地址H 起始地址L 输出数量H 输出数量L 字节长度 输出值H 输出值L
- 响应：MBAP 功能码 起始地址H 起始地址L 输出数量H 输出数量L

(4)、0x02：读离散量输入

从一个从站中读1~2000个连续的离散量输入状态

- 请求：MBAP 功能码 起始地址H 起始地址L 数量H 数量L（共12字节）
- 响应：MBAP 功能码 数据长度 数据（长度：9+ceil（数量/8））

- 如：从地址0x0000开始读0x0012个离散量输入-----
---00 01 00 00 00 06 01 02 00 00 00 12
- 回：数据长度为0x03个字节，数据为0x01 04 00，表示第一个离散量输入和第11个离散量输入为ON，其余为OFF-----00 01 00 00 00 06 01 02 03 01 04 00

(5)、0x04：读输入寄存器

从一个远程设备中读1~2000个连续输入寄存器

- 请求：MBAP 功能码 起始地址H 起始地址L 寄存器数量H 寄存器数量L（共12字节）
- 响应：MBAP 功能码 数据长度 寄存器数据(长度：9+寄存器数量×2)
- 如：读起始地址为0x0002，数量为0x0005的寄存器数据-----00 01 00 00 00 06 01 04 00 02 00 05
- 回：数据长度为0x0A，第一个寄存器的数据为0x0c，其余为0x00-----00 01 00 00 00 0D 01 04 0A 00 0C 00 00 00 00 00 00 0 0 00

(6)、0x03：读保持寄存器

从远程设备中读保持寄存器连续块的内容

- 请求：MBAP 功能码 起始地址H 起始地址L 寄存器数量H 寄存器数量L（共12字节）
- 响应：MBAP 功能码 数据长度 寄存器数据(长度：9+寄存器数量×2)
- 如：起始地址是0x0000，寄存器数量是 0x0003-----00 01 00 00 00 06 01 03 00 00 00 03
- 回：数据长度为0x06，第一个寄存器的数据为0x21，其余为0x00-----00 01 00 00 00 09 01 03 06 00 21 00 00 00 00

(7)、0x06：写单个保持寄存器

在一个远程设备中写一个保持寄存器

- 请求：MBAP 功能码 寄存器地址H 寄存器地址L 寄存器值H 寄存器值L（共12字节）
- 响应：MBAP 功能码 寄存器地址H 寄存器地址L 寄存器值H 寄存器值L（共12字节）
- 如：向地址是0x0000的寄存器写入数据0x000A-----00 01 00 00 00 06 01 06 00 00 00 0A
- 回：写入成功-----00 01 00 00 00 06 01 06 00 00 00 0A

(8)、0x10：写多个保持寄存器

在一个远程设备中写连续寄存器块（1~123个寄存器）

- 请求：MBAP 功能码 起始地址H 起始地址L 寄存器数量H 寄存器数量L 字节长度 寄存器值（13+寄存器数量×2）
- 响应：MBAP 功能码 起始地址H 起始地址L 寄存器数量H 寄存器数量L（共12字节）
- 如：向起始地址为0x0000，数量为0x0001的寄存器写入数据，数据长度为0x02，数据为0x000F-----00 01 00 00 00 09 01 10 00 00 00 01 02 00 0F
- 回：写入成功-----00 01 0 0 00 00 06 01 10 00 00 00 01

9、数据解析

ModBus TCP/IP与串行链路Modbus的数据域是一致的，具体数据域可以参考串行modbus。这里给出几个ModbusTcp的链路解析说明，辅助新人分析报文。

(1)、数据请求

97 76 00 00 00 06 04 04 00 7D 00 7D				
	示例	长度	说明	备注
Map报文头	0x97	1	事务处理标识符Hi	客户机发起，服务器复制，用于事务处理配对
	0x96	1	事务处理标识符Lo	
	0x0000	2	协议标识符号	客户机发起，服务器复制 Modbus协议 = 0.
	0x0006	2	长度	从本字节下一个到最后
	0x04	1	单元标识符	客户机发起，服务器复制 串口链路或其他总线上远程终端标识
功能码	0x04	1	功能码，读寄存器	参考标准modbus协议
数据	0x007D	2	起始地址	
	0x 007D	2	寄存器数量	
校验				

(2)、数据请求回复

[illegible]

	示例	长度	说明	备注
Map报文头	0x97	1	事务处理标识符Hi	客户机发起，服务器复制，用于事务处理配对
	0x96	1	事务处理标识符Lo	
	0x0000	2	协议标识符号	客户机发起，服务器复制 Modbus协议 = 0.
	0x00FD	2	长度	从本字节下一个到最后
	0x04	1	单元标识符	客户机发起，服务器复制 串口链路或其他总线上远程终端标识
功能码	0x04	1	功能码，读寄存器	参考标准modbus协议
数据	0x FA	1	字节个数	
	0x----		数据	
校验				

(3)、写多个寄存器

97 79 00 00 00 09 04 10 00 00 00 01 02 00 01				
	示例	长度	说明	备注
Map报文头	0x97	1	事务处理标识符Hi	客户机发起，服务器复制，用于事务处理配对
	0x79	1	事务处理标识符Lo	
	0x0000	2	协议标识符号	客户机发起，服务器复制 Modbus协议 = 0.
	0x0009	2	长度	从本字节下一个到最后
	0x04	1	单元标识符	客户机发起，服务器复制 串口链路或其他总线上远程终端标识
功能码	0x10	1	功能码，读寄存器	参考标准modbus协议
数据	0x0000	2	起始地址	
	0x 0001	2	写寄存器数量	
	0x 02	1	写字节的个数	
	00 01	2	目标值	
校验				

(4)、写多个寄存器响应

97 79 00 00 00 06 04 10 00 00 00 01				
	示例	长度	说明	备注
Map报文头	0x97	1	事务处理标识符Hi	客户机发起，服务器复制，用于事务处理配对
	0x79	1	事务处理标识符Lo	
	0x0000	2	协议标识符号	客户机发起，服务器复制 Modbus协议 = 0.
	0x0006	2	长度	从本字节下一个到最后

	0x04	1	单元标识符	客户机发起，服务器复制 串口链路或其他总线上远程终端标识
功能码	0x10	1	功能码，读寄存器	参考标准modbus协议
数据	0x0000	2	起始地址	
	0x 0001	2	寄存器个数	
校验				

10、五种量的概念

在工业自动化控制中，经常会遇到开关量，数字量，模拟量，离散量，脉冲量等各种概念，而人们在实际应用中，对于这些概念又很容易混淆。现将各种概念罗列如下：

(1)、开关量

一般指的是触点的“开”与“关”的状态，一般在计算机设备中也会用“0”或“1”来表示开关量的状态。开关量分为有源开关量信号和无源开关量信号，有源开关量信号指的是“开”与“关”的状态是带电源的信号，专业叫法为跃阶信号，可以理解为脉冲量，一般的都有220VAC, 110VAC,24VDC,12VDC等信号，无源开关量信号指的是“开”和“关”的状态时不带电源的信号，一般又称之为干接点。电阻测试法为电阻0或无穷大。

(2)、数字量

很多人会将数字量与开关量混淆，也将其与模拟量混淆。数字量在时间和数量上都是离散的物理量，其表示的信号则为数字信号。数字量是由0和1组成的信号，经过编码形成有规律的信号，量化后的模拟量就是数字量。

(3)、模拟量

模拟量的概念与数字量相对应，但是经过量化之后又可以转化为数字量。模拟量是在时间和数量上都是连续的物理量，其表示的信号则为模拟信号。模拟量在连续的变化过程中任何一个取值都是一个具有意义的物理量，如温度，电压，电流等。

(4)、离散量

离散量是将模拟量离散化之后得到的物理量。即任何仪器设备对于模拟量都不可能有个完全精确的表示，因为他们都有一个采样周期，在该采样周期内，其物理量的数值都是不变的，而实际上的模拟量则是变化的。这样就将模拟量离散化，成为了离散量。

(5)、脉冲量

脉冲量就是瞬间电压或电流由某一值跃变到另一值的信号量。在量化后，其变化持续有规律就是数字量，如果其由0变成某一固定值并保持不变，其就是开关量。

综上所述，模拟量就是在某个过程中时间和数量连续变化的物理量，由于在实际的应用中，所有的仪器设备对于外界数据的采集都有一个采样周期，其采集的数据只有在下一个采样周期开始时才有变动，采样周期内其数值并不随模拟量的变化而变动。

这样就将模拟量离散化了，例如：某设备的采样周期为1秒，其在第五秒的时间采集的温度为35度，而第六秒的温度为36度，该设备就只能标称第五秒时间温度35度，第六秒时间温度36度，而第五点五秒的时间其标称也只是35度，但是其实际的模拟量是35.5度。这样就将模拟信号离散化。其采集的数据就是离散化了，不再是连续的模拟量信号。

由于计算机只识别0和1两个信号，即开关量信号，用其来表示数值都是使用数字串来表示，由于计算能力的问题，其数字串不能无限长，即其表达的精度也是有限的，同样的以温度为例，由于数字串限制，其表达温度的精度只能达到0.1度，小于该单位的数值则不能被标称，这样就必须将离散量进行量化，将其变为数字量。即35.68度的温度则表示为35.6度。

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。