Volume 9 • Issue 4
JULY-AUGUST 2008

the EXTENSION

A Technical Supplement to Control Network

Introduction to the Modbus Protocol

By George Thomas, Contemporary Controls

This is the first of a two-part series on Modbus. The first issue addresses the protocol itself. The second discusses the Modbus Serial and Modbus TCP implementations allowing Modbus to remain a very popular protocol.

What do ARCNET®, Ethernet and Modbus have in common? They were all developed in the 1970s and are still widely used today. Of course they have evolved over time, but their basic operation remains intact. Why change a good thing?

There is one basic difference in the three technologies. Both ARCNET and Ethernet are data link and physical layer standards without a protocol while Modbus is a protocol that can operate over several data links and physical layers. Originally intended as a point-to-to interface between proprietary Modicon products, the protocol has found use in multi-drop and peer-to-peer networks like TCP/IP. It is no longer restricted to just Modicon equipment.

Modicon Communications Protocol

Modbus was introduced in 1979 by the company "Modicon," a leader in the infant programmable logic controller (PLC) market. It was intended as the internal point-to-point communications protocol between Modicon PLCs and programming panels used to program the controllers. After some acquisitions, Modicon is now part of AEG Schneider Automation with the brand names "Modicon," "Square D," and "Telemecanique." You would think the protocol would have long been forgotten but the group Modbus-IDA now carries the banner at its **http://www.modbus.org** web site. The protocol continues to thrive since it is easy to understand, and many engineers have "cut their protocol teeth" on Modbus. Besides, it is an open system and can be used royalty-free. It is not restricted to just industrial automation. Modbus can be found in numerous diverse automation industries including building automation.

The *Modicon Modbus Protocol Reference Guide* dated June 1996 can be found at the Modbus-IDA site along with other Modbus references. When you read the original Modbus documentation you will notice many mentions of specific Modicon equipment. Only later, did the Modbus-IDA group develop generic standards to assist implementation. The three other references are called the *Modbus Application Protocol Specification, Modbus over Serial Line Specification and Implementation Guide*, and the *Modbus Messaging on TCP/IP Implementation Guide.* All are available for free. The Modbus protocol would operate over several network implementations including Modicon's proprietary peer-to-peer network Modbus Plus. Before we examine the more modern implementations in Part 2, we will concentrate on the protocol itself.

Original Modicon Implementation

It is interesting to note that Modicon did not use Modbus in a multi-drop network but instead used point-to-point connections with EIA-232C interfaces installed on their PLCs. The Modbus protocol is a master-slave protocol and the terms "master" and "slave" continue to be used today. Modbus allows only one master and up to 247 slaves. A slave is typically a Modicon PLC with an EIA-232C interface. Masters are typically programming panels or host computers. Therefore, if one host computer needed to communicate to four PLCs, four serial ports would be required on the host computer. This results in a star topology. EIA-232C cable lengths are short, so if longer distances are required modems can be used. It was not until later that 2-wire and 4-wire EIA-485 multi-drop networks appeared.

With the Modbus protocol, only the master can initiate a message. Slaves cannot. So if a slave notices "that the cooling water pumps to the nuclear reactor have stopped," the slave cannot inform the master until the master happens to send a query to the slave with the effective message "how are things going?" The master has no address, but the slaves are numbered from 1 to 247.

Address "0" is reserved as a broadcast address to all slaves. All slaves will receive the broadcast message but will not respond.

Query—Response Messaging

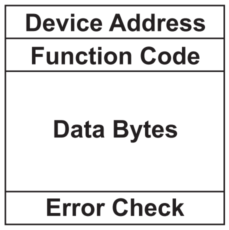

Figure 1. Simplified Modbus message format.

The command issued by the master is called a *Query* and the response from the slave is simply called the *Response*. The format in Figure 1 shows the simplified structure of the messages that can serve as either a query or a response.

The master has no address so the device address is always the intended slave. If it is a query, the query is directed to the slave with the assigned device address. If the message is a response, the response came from the slave with the indicated device address. Commands are issued by function codes such as 03—Read Holding Registers. In this case the master must indicate the range of registers to be queried. The slave responds with the requested data based upon the indicated range. The message format is similar for all function codes, but of course the data changes based upon the code itself. After each message there is an error check appended by the originating station so that the receiving station can check the integrity of the received message.

The above scenario assumes a successful interchange of a query and a response. If the slave wants to communicate an error condition or an exception case, the function code is modified by the slave by setting the most-significant-bit (MSB) of the function code to a 1. The data field will then contain information specific to the exception. The master can still extract the original function code that it sent.

It should be noted that the query-response cycle is completed before the master sends out the next query to either the same slave or another slave. This is unlike protocols such as DeviceNet that can send out one multicast command and then wait for several devices to response in no particular order to this one command. Modbus has no multicast capability so time is lost as each master query requires the directed slave to not only receive the message but act upon it and respond before the master moves on to other communications activity.

ASCII and RTU Modes

The simple Modbus protocol becomes a bit more confusing since there are two serial transmission modes. One is called ASCII for *American Standard Code for Information Interchange* and the other RTU for *Remote Terminal Unit*. In this case RTU does not mean "roof-top unit." The RTU term comes from the *Supervisor Control and Data Acquisition* (SCADA) industry where the master, called a *Central Terminal Unit* (CTU), communicates to several RTUs at distant locations. This configuration is similar to that of the original Modicon implementation with one CTU communicating to RTUs using modems in a star topology. The use of either ASCII or RTU modes has nothing to do with topology, but it impacts the framing and timing of the messages. When operating over serial communication links, both modes utilize asynchronous communications with one character sent at a time with defined framing.

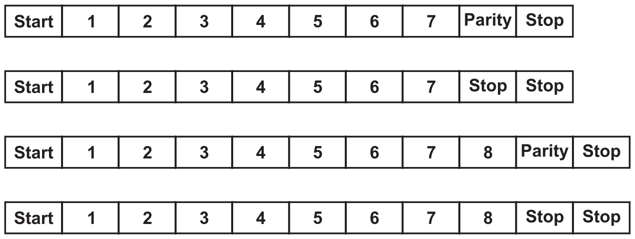

Figure 2. Character framing for 7-bit ASCII and 8-bit RTU with or without parity.

Figure 2 shows how a character is sent using asynchronous serial communications. Each character is sent as a series of bits with a bit time equal to the reciprocal of the baud rate. For example, at 9600 baud the bit time is 104.1 µs. When no messages are being sent, the line is said to be marking. The opposite of the marking state is the spacing state. Each character begins with a start bit, as the line drops to the spacing state for one bit time, and ends with one or more stop bits as the line returns to the marking state.

Either a 7-bit (ASCII mode) or 8-bit (RTU mode) character is sent in between—with the least-significant-bit (LSB) sent first. After the character comes either a parity bit or another stop bit. Odd, even, or no parity can be selected by the user. In ASCII mode it takes ten bits to send one character while in RTU mode it takes eleven. With asynchronous communications, characters can be sent either back-to-back or with some delay between characters. A series of characters form messages having different structures depending whether ASCII or RTU mode is intended.

ASCII Message Framing

The seven-bit ASCII code was developed in the early 1960s as a uniform code for displaying English characters on teleprinters such as the Teletype Model ASR-33. As CRT terminals (glass teletypes) began to replace electromechanical teleprinters, the ASCII standard was retained, making the migration easier. ASCII is a U.S. standard for displaying English characters and for sending control codes such as Carriage Return (CR) and Line Feed (LF) which are carryovers from the teleprinter days. The reason a CR precedes a LF was to give the teleprinter more time to physically move the carriage from the end of a line to the beginning of the line.

The line feed would advance a line vertically, but that activity was faster than moving the carriage so this command was sent second in the sequence. Electromechanical teleprinters had no buffering so if an LFCR was executed instead of a CRLF you could have printing in the middle of the page instead of at the beginning because the carriage failed to return in time before the next printable character was received. The CRLF sequence was very important for 10 characters per second teleprinters but not for CRT terminals. For Modbus ASCII mode, the CRLF sequence simply indicates the end of a frame. The advantage of ASCII mode is that if you attached a CRT terminal in place of a slave device, you can observe the nicely formatted human readable code sent by the master on the CRT screen.

Start of Frame	Device Address	Function Code	Data	LRC Check	End of Frame
1 character (:)	2 characters	2 characters	n characters	2 characters	2 characters (CRLF)

Figure 3. ASCII framing of a Modbus message.

Figure 3 shows the ASCII framing for Modbus messages. The start of frame is simply a colon (:) and the end of frame is the CRLF sequence requiring two ASCII characters. ASCII characters are each 7-bits. All other characters in the other fields must be either the numbers 0–9 or the letters A–F since the data is going to be represented in hexadecimal format but displayed as ASCII characters. For example, function code 03 would be displayed as two ASCII characters "0" and "3." The same applies to the data. One advantage of ASCII mode is timing. As much as one second can elapse between characters without a timeout error. Therefore, a good typist could simulate a master by typing out a character string on a CRT terminal to a slave and observing the slave's response.

RTU Message Framing

When operating in RTU mode the timing is much more critical. There is no specific Start of Frame character. Instead, the message frame begins with four character times of marking. After this interval, the device address is sent followed by the function code and data. There are other differences from that of the ASCII message frame as noted in Figure 4. Instead of a *Longitudinal Redundancy Check* (LRC) check in the ASCII frame, a more robust *Cyclic Redundancy Check* (CRC) check of the data is applied in the RTU frame. The End of Frame indication is strictly based upon four character times of marking.

RTU messages must be sent as a continuous stream and any significant gaps between characters could result in a dropped message. Unlike ASCII, the RTU messages are not human readable. However, the messages are quite compact and more efficient to send. The RTU mode remains the more popular format.

Start of Frame	Device Address	Function Code	Data	CRC Check	End of Frame
4 character times	8 bits	8 bits	n x 8 bits	16 bits	4 character times

Figure 4. RTU framing of a Modbus message.

Modbus Register Map

Before we discuss function codes, we should study the Modbus register map in Figure 5 since certain function codes imply specific register ranges. Early PLCs were mostly concerned with discrete inputs and discrete outputs each represented by one-bit in a register map. For Modicon PLCs, discrete outputs begin at location 00001 and discrete inputs begin at location 10001. Each requires one-bit of storage. Inputs, called contacts, can only be read and outputs, called coils, can be read or written. Since early PLCs were considered relay panel replacements, the terms coils and contacts were retained to assist electricians trying to understand these new electronic controllers.

As the complexity of PLCs increased, the ability to handle analog input/output (I/O), and to execute math calculations was added. The I/O range of 16-bit register references begins at 30001 for read-only analog inputs or thumbwheel switches, and 40001 for general purpose read/write registers that can also serve as analog outputs. There are really no restrictions above 40001. Depending upon the vendor of the equipment, they can be internal registers, analog inputs, analog outputs, and even discrete inputs and outputs. However, not all function codes reference this range—but enough do.

I/O Range	Description
00001 – 10000	Read/Write discrete output or coils
10001 – 20000	Read discrete inputs
30001 – 40000	Read input registers – 16-bit registers such as analog inputs
40001 – 50000	Read/Write holding registers – 16-bit storage or I/O

Figure 5. Typical Modbus Register Map.

Function Codes

The Modbus function codes are defined in both the *Modicon Modbus Protocol Reference Guide* and the *Modbus Application Protocol Specification*. Because there are differences in the function names and the number of function codes the latter document is recommended. Although the function code range spans from 1 to 127, only about 20 are defined public function codes. User-defined function codes are allowed in specific locations within this range. However, many Modbus devices only support a small subset of the available codes. We will only examine those function codes that involve single-bit and 16-bit data access to get a flavor of how I/O is handled. A list is provided in Figure 6.

Code	1/16-bit	Description	I/O Range
01	1-bit	Read coils	00001 – 10000
02	1-bit	Read contacts	10001 – 20000
05	1-bit	Write a single coil	00001 – 10000
15	1-bit	Write mulitple coils	00001 – 10000
03	16-bit	Read holding registers	40001 – 50000
04	16-bit	Read input registers	30001 – 40000
06	16-bit	Write single register	40001 – 50000
16	16-bit	Write mulitple registers	40001 – 50000
22	16-bit	Mask write register	40001 – 50000
23	16-bit	Read/write mulitple registers	40001 – 50000
24	16-bit	Read FIFO queue	40001 – 50000

Figure 6. Data access function codes.

You will notice from Figure 6 that 1-bit function codes relate to discrete devices such as contacts and coils. The 16-bit function codes relate to input registers and holding registers. Input registers can only be read while holding registers can be either read or written. Also notice there is an implied I/O range depending upon the function code. For example, function code 06—Write single register, only addresses the relevant range of 40001–50000 and no other range. Therefore it is only necessary to reference the offset from the base range when structuring the message. Instead of indicating register location 40001 we simply say 0000.

This is a good time to explain one of the more confusing aspects of Modbus and that is referencing I/O points in Modbus messages. Modicon elected to number physical points within a range beginning with the number 1 instead of 0. Coil 1 is referenced in a message as location 0000 and not 00001. Likewise, Contact 1 is referenced as 0000 instead of 10001. The same applies to holding register 40001. It is also referenced as 0000. The function code always points to the proper I/O range and only the offset from base address of that range is needed to uniquely identify the point.

The offset is a 16-bit word and is displayed in hexadecimal when examining the actual Modbus message. All references in the Modbus Register Map are decimal references. Register 40016 is referenced as 0x000F which is hexadecimal for 40016–40001. Although this is confusing at first, it is only an issue for those writing Modbus drivers.

Summary

Modbus is popular for its simplicity. With so many users in the field with Modbus knowledge and a Modbus-IDA association backing this open standard, it will continue to remain popular.

References

Modbus Application Protocol Specification V1.1b,
http://www.Modbus-IDA.org, December 28, 2006

Modbus over Serial Line Specification and Implementation Guide, V1.02,
http://www.Modbus-IDA.org, December 20, 2006

Modbus Messaging on TCP/IP Implementation Guide V1.0b,
http://www.Modbus-IDA.org, October 24, 2006

Modbus Protocol Reference Guide Rev J,
http://www.Modbus-IDA.org, June 1996

CONTEMPORARY CONTROLS®
www.ccontrols.com