**MODBUS PROTOCOL**

Introduction

Modbus is a Modicon systems-developed communication protocol. In simple words, it is a mechanism for sending data between electrical devices across serial lines. The Modbus Master is the device that requests information, and the Modbus Slaves are the devices that deliver it. In a standard Modbus network, there is one Master and up to 247 Slaves, each with a unique Slave Address from 1 to 247. The Master can also write information to the Slaves.

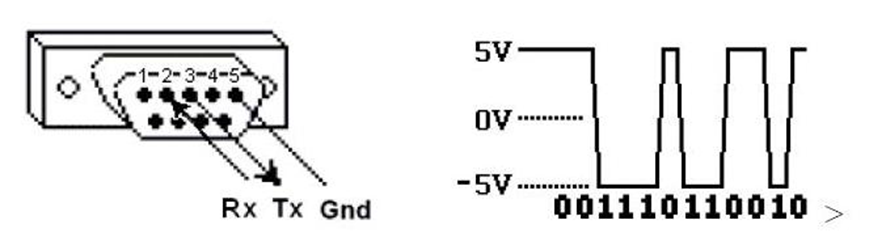
Purpose

Modbus is an open protocol, which means that manufacturers can integrate it into their products without paying royalties. It has been the most widely used method of linking industrial electronic devices and has established a standard communications protocol in the industry.

It is widely used by various producers in a variety of industries. Modbus is commonly used to send signals from instrumentation and control devices back to a main controller or data gathering system, such as a system that measures temperature and humidity and sends the data to a computer. In supervisory control and data acquisition (SCADA) systems, Modbus is frequently used to connect a supervisory computer to a remote terminal unit (RTU). Modbus protocol versions for serial lines (Modbus RTU and Modbus ASCII) and Ethernet exist (Modbus TCP).

How does it work?

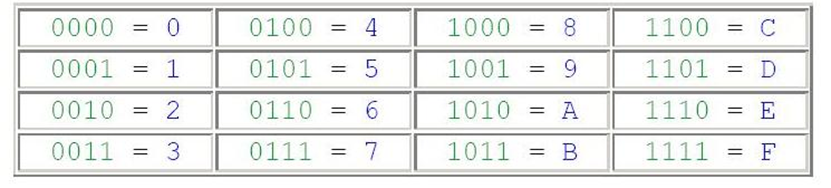
Between devices, Modbus is transmitted using serial lines. A single serial cable connecting the serial ports of two machines, a Master and a Slave, would be the simplest setup. The data is delivered in bits, which are a series of ones and zeros. Each bit is supplied in the form of a voltage. Positive voltages are conveyed as zeros, and negative voltages are sent as ones. The bits are transferred at a rapid speed. 9600 baud is a common communication speed (bits per second).

****

*Figure: Bits – Ones and Zeroes*

Hexadecimal Concept

It can be useful to see the actual raw data being transferred when debugging problems. Because reading long sequences of ones and zeros is challenging, the bits are concatenated and displayed in hexadecimal. Each of the sixteen characters from 0 to F represents a block of four bits. One of the 256-character pairs from 00 to FF represents each block of 8 bits (called a byte).

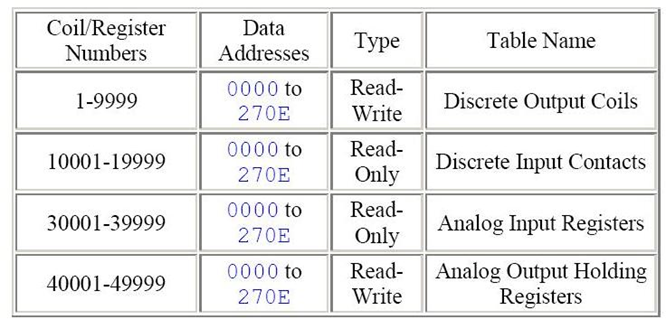


*Figure: Hexadecimal Concept*

Storage of Data

The Slave device stores information in four separate tables. Two tables store discrete values (coils) that turn on and off, while two tables contain numerical values (registers). A read-only table and a read-write table are present in each of the coils and registers. There are 9999 values in each table. Each coil or contact is 1 bit, with a data address ranging from 0000 to 270E. Each register has a data address between 0000 and 270E and is 1 word = 16 bits = 2 bytes.

Because they do not appear in the actual messages, Coil/Register Numbers can be thought of as location names. In the messages, the Data Addresses are used. For example, the first Holding Register, number 40001, has the Data Address 0000. The offset is the difference between these two values. Each table is offset in a distinct way. 10001, 30001, and 40001 are the first three numbers in the alphabet.



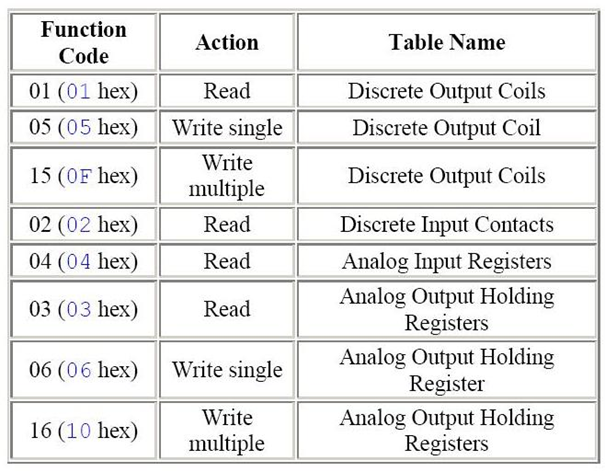
*Figure: Data Storage Tables*

Slave ID

A unique unit address is assigned to each slave in a network, ranging from 1 to 247. When the master requests data, the first byte it sends is the Slave address. This manner, after the first byte, each slave knows whether or not to ignore the message.

Function Code

The second byte sent by the Master is the Function code. This number tells the slave which table to access and whether to read from or write to the table.

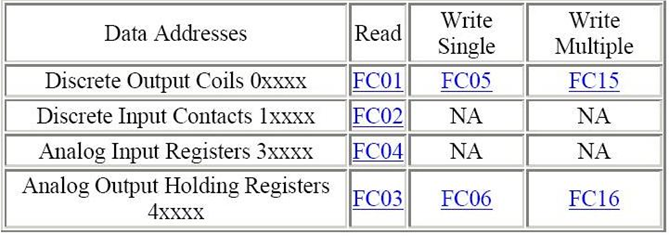


*Figure: Function Code*

Cyclic Redundancy Check (CRC)

It is two bytes added to the end of every ModBus message for error detection. Every byte in the message is used to calculate the CRC. The receiving device also calculates the CRC and compares it to the CRC from the sending device. If even one bit in the message is received incorrectly, the CRCs will be different, and an error will result.

ModBus Formats for Commands and Responses



*Figure: ModBus Formats*

ModBus Data Types

The example for FC03 shows that register 40108 contains AE41 which converts to the 16 bits 1010 1110 0100 0001.

Register 40108 could be defined as any of these 16-bit data types:

* A 16-bit unsigned integer (a whole number between 0 and 65535) register 40108 contains AE41 = 44,609 (hex to decimal conversion)
* A 16-bit signed integer (a whole number between -32768 and 32767) AE41 = -20,927 (hex to decimal conversion that wraps, if it is over 32767 then subtract 65536)
* A two-character ASCII string (2 typed letters) AE41 = ® A
* A discrete on/off value (this works the same as 16-bit integers with a value of 0 or 1. The hex data would be 0000 or 0001)

Register 40108 could also be combined with:

* 40109 to form any of these 32-bit data types:
* A 32-bit unsigned integer (a number between 0 and 4,294,967,295) 40108,40109 = AE41 5652 = 2,923,517,522
* A 32-bit signed integer (a number between -2,147,483,648 and 2,147,483,647) AE41 5652 = -1,371,449,774
* A 32-bit double precision IEEE floating point number. This is a mathematical formula that allows any real number (a number with decimal points) to be represented by 32 bits with an accuracy of about seven digits. AE41 5652 = -4.395978 E-11
* A four-character ASCII string (4 typed letters) AE41 5652 = ® A V R More registers can be combined to form longer ASCII strings. Each register being used to store two ASCII characters (two bytes).

Byte and Word Ordering

The Modbus specification does not define exactly how the data is stored in the registers. Therefore, some manufacturers implemented ModBus in their equipment to store and transmit the higher byte first followed by the lower byte. (AE before 41). Alternatively, others store and transmit the lower byte first (41 before AE). Similarly, when registers are combined to represent 32-bit data types, some devices store the higher 16 bits (high word) in the first register and the remaining low word in the second (AE41 before 5652) while others do the opposite (5652 before AE41). It does not matter which order the bytes or words are sent in, as long as the receiving device knows which way to expect it. For example, if the number 29,235,175,522 were to be sent as a 32-bit unsigned integer, it could be arranged any of these four ways.

* AE41 5652 high byte first high word first
* 5652 AE41 high byte first low word first
* 41AE 5256 low byte first high word first
* 5256 41AE low byte first low word first

Modbus Map

A modbus map is simply a list for an individual slave device that defines:

* What the data is (eg. pressure or temperature readings)
* Where the data is stored (which tables and data addresses)
* How the data is stored (data types, byte and word ordering)

Some devices are built with a fixed map that is defined by the manufacturer. While other devices allow the operator to configure or program a custom map to fit their needs.

Extended Register Address

Since the range of the analog output holding registers is 40001 to 49999, it implies that there cannot be more than 9999 registers. Although this is usually enough for most applications, there are cases where more registers would be beneficial.

Registers 40001 to 49999 correspond to data addresses 0000 to 270E. If we utilize the remaining data addresses 270F to FFFF, over six times as many registers can be available, 65536 in total. This would correspond to register numbers from 40001 to 105536.

Many modbus software drivers (for Master PCs) were written with the 40001 to 49999 limits and cannot access extended registers in slave devices. And many slave devices do not support maps using the extended registers. But on the other hand, some slave devices do support these registers and some Master software can access it, especially if custom software is written.

Difference between Modbus ASCII and Modbus RTU

Modbus RTU and Modbus ASCII talk the same protocol. The only difference is that the bytes being transmitted over the wire are presented as binary with RTU and as readable ASCII with Modbus RTU. important to note about RTU is that the RTU message does not have a Start\_of\_text indication. The receiving party in the communications uses a "silent" time to determine the start of a new message. ASCII does have a start-of-text token. Binary messages are shorter than ASCII and therefore theoretically faster to transmit/receive. You may be happy to see update rates of about 100ms in your HMI/SCADA and could choose either communication.

Summary:

* Use RTU if possible
* Use ASCII in case RTU is giving timeout problems on WinNT or when using slow communications media like 300 bps or dial up modems Most OPC Servers for Modbus support ASCII as well as RTU communications.

2-byte Slave Addressing

Since a single byte is normally used to define the slave address and each slave on a network requires a unique address, the number of slaves on a network is limited to 256. The limit defined in the ModBus specification is even lower at 247. To get beyond this limit, a modification can be made to the protocol to use two bytes for the address. The master and the slaves would all be required to support this modification. Two-byte addressing extends the limit on the number of slaves in a network to 65535. By default, the Simply Modbus software uses 1 byte addressing. When an address greater than 255 is entered, the software automatically switches to 2-byte addressing and stays in this mode for all addresses until the 2-byte addressing is manually turned off.