Modbus

From Wikipedia, the free encyclopedia

Modbus is a serial communications protocol published by Modicon in 1979 for use with its programmable logic controllers (PLCs). Simple and robust, it has since become one of the de facto standard communications protocols in the industry, and it is now amongst the most commonly available means of connecting industrial electronic devices.[1] The main reasons for the extensive use of Modbus in the industrial environment are:

1. It has been developed with industrial applications in mind
2. It is openly published and royalty-free
3. It is easy to deploy and maintain
4. It moves raw bits or words without placing many restrictions on vendors

Modbus allows for communication between many (approximately 240) devices connected to the same network, for example a system that measures temperature and humidity and communicates the results to a computer. Modbus is often used to connect a supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (SCADA) systems. Many of the data types are named from its use in driving relays: a single-bit physical output is called a *coil*, and a single-bit physical input is called a *discrete input* or a *contact*.

The development and update of Modbus protocols are managed by the Modbus Organization (http://www.modbus.com) , formed of independent users and suppliers of Modbus compliant devices. Some of its prominent members (http://www.modbus.com/about.php) are Precision Digital Corporation (http://www.predig.com) , Motor Protection Electronics and FieldServer Technologies. Other companies, such as SoftDEL Systems and SATEC Ltd. (http://www.satec-global.com/eng/default.aspx) , offer Modbus devices without being formal members of Modbus Organization.

Contents

- 1 Protocol versions
- 2 Communication and devices
- 3 Frame Format
- 4 Supported Function Codes
- 5 Implementations
- 6 Limitations
- 7 Trade group
- 8 References
- 9 External links
  - 9.1 Open-source software

Protocol versions

Versions of the Modbus protocol exist for serial port and for Ethernet and other networks that support the Internet protocol suite. Most Modbus devices communicate over a serial EIA-485 physical layer [1] (http://www.obvius.com/documentation/faq/modbus.html) . There are many variants of Modbus protocols

- *Modbus RTU* — This is used in serial communication & makes use of a compact, binary representation of the data for protocol communication. The RTU format follows the commands/data with a cyclic redundancy check checksum as an error check mechanism to ensure the reliability of data. Modbus RTU is the most common implementation available for Modbus. A Modbus RTU message must be transmitted continuously without inter-character hesitations. Modbus messages are framed (separated) by idle (silent) periods.
- *Modbus ASCII* — This is used in serial communication & makes use of ASCII characters for protocol communication. The ASCII format uses a longitudinal redundancy check checksum. Modbus ASCII messages are framed by leading colon (':') and trailing newline (CR/LF).
- *Modbus TCP/IP or Modbus TCP* — This is a Modbus variant used for communications over TCP/IP networks, connecting over port 502. [2] It does not require a checksum calculation as lower layers already provide checksum protection.
- *Modbus over TCP/IP or Modbus over TCP or Modbus RTU/IP* — This is a Modbus variant that differs from Modbus TCP in that a checksum is included in the payload as with Modbus RTU.
- *Modbus over UDP* — Some have experimented with using Modbus over UDP on IP networks, which removes the overheads required for TCP [3]
- *Modbus Plus (Modbus+, MB+ or MBP)* — An extended version, Modbus Plus (Modbus+ or MB+), also exists, but remains proprietary to SCHNEIDER ELECTRIC. It requires a dedicated co-processor to handle fast HDLC-like token rotation. It uses twisted pair at 1 Mbit/s and includes transformer isolation at each node, which makes it transition/edge triggered instead of voltage/level triggered. Special interfaces are required to connect Modbus Plus to a computer, typically a card made for the ISA (SA85), PCI or PCMCIA bus.

- *Modbus PEMEX*- Modbus PEMEX is an extension of standard Modbus with support for historical and flow data. It is widely used in process automation.

Data model and function calls are identical for the first 4 variants of protocols; only the encapsulation is different. However the variants are not interoperable as the frame formats are different.

Communication and devices

Each device intended to communicate using Modbus is given a unique address. In serial and MB+ networks only the node assigned as the Master may initiate a command, but on Ethernet, any device can send out a Modbus command, although usually only one master device does so. A Modbus command contains the Modbus address of the device it is intended for. Only the intended device will act on the command, even though other devices might receive it (an exception is specific broadcastable commands sent to node 0 which are acted on but not acknowledged). All Modbus commands contain checking information, ensuring that a command arrives undamaged. The basic Modbus commands can instruct an RTU to change a value in one of its registers, control or read an I/O port, as well as commanding the device to send back one or more values contained in its registers.

There are many modems and gateways that support Modbus, as it is a very simple protocol and often copied. Some of them were specifically designed for this protocol. Different implementations use wireline, wireless communication, such as in the ISM band, and even SMS or GPRS. One of the more common designs of wireless networks makes use of the mesh topology. Typical problems the designers have to overcome include high latency and timing problems.

Frame Format

All modbus variants choose different frame formats.[1]

Modbus RTU Frame Format		
Name	**Length**	**Function**
Start	3.5c idle	*at least 3-1/2 character times of silence (MARK condition)*
Address	8 bits	*Station Address*
Function	8 bits	*Indicates the function codes like read coils / inputs*
Data	n * 8 bits	*Data + length will be filled depending on the message type*
CRC Check	16 bits	*Error checks*
End	3.5c idle	*at least 3-1/2 character times of silence between frames*

Modbus ASCII Frame Format		
Name	**Length**	**Function**
Start	1 char	*starts with colon (:) (ASCII value is 3A hex)*
Address	2 chars	*Station Address*
Function	2 chars	*Indicates the function codes like read coils / inputs*
Data	n chars	*Data +length will be filled depending on the message type*
LRC Check	2 chars	*Error checks*
End	2 chars	*carriage return – line feed(CRLF) pair (ASCII values of 0D & 0A hex)*

Modbus TCP Frame Format		
Name	**Length**	**Function**
Transaction Identifier	2 bytes	*For synchronization between messages of server & client*
Protocol Identifier	2 bytes	*Zero for MODBUS/TCP*
Length Field	2 bytes	*Number of remaining bytes in this frame*
Unit Identifier	1 byte	*Slave Address (255 if not used)*
Function code	1 byte	*Function codes as in other variants*
Data bytes	n bytes	*Data as response or commands*

Unit identifier is used with MODBUS/TCP devices that are composites of several MODBUS devices, e.g. on MODBUS/TCP to MODBUS RTU gateways. In such case, the unit identifier tells the Slave Address of the device behind the gateway. Natively MODBUS/TCP-capable devices usually ignore the Unit Identifier.

Supported Function Codes

The various reading, writing and other operations are categorised as follows.[4] The most primitive reads and writes are shown in bold. A number of sources [5] use alternative terminology, for example *Force Single Coil* where the standard uses *Write Single Coil.*

			Function Name	Function Code
Data Access	Bit access	Physical Discrete Inputs	**Read Discrete Inputs**	2
		Internal Bits or Physical Coils	**Read Coils**	1
			Write Single Coil	5
			Write Multiple Coils	15
	16-bit access	Physical Input Registers	**Read Input Register**	4
		Internal Registers or Physical Output Registers	**Read Holding Registers**	3
			Write Single Register	6
			Write Multiple Registers	16
			Read/Write Multiple Registers	23
			Mask Write Register	22
			Read FIFO Queue	24
	File Record Access		Read File Record	20
			Write File Record	21
Diagnostics			Read Exception Status	7
			Diagnostic	8
			Get Com Event Counter	11
			Get Com Event Log	12
			Report Slave ID	17
			Read Device Identification	43
Other			Encapsulated Interface Transport	43

Implementations

Almost all implementations have variations from the official standard. Different varieties might not communicate correctly between equipment of different suppliers. Some of the most common variations are:

- Data types
  - Floating point IEEE
  - 32-bit integer
  - 8-bit data
  - Mixed data types
  - Bit fields in integers
  - Multipliers to change data to/from integer. 10, 100, 1000, 256 ...
- Protocol extensions
  - 16-bit slave addresses
  - 32-bit data size (1 address = 32 bits of data returned.)
  - Word swapped data

Limitations

- Since Modbus was designed in the late 1970s to communicate to programmable logic controllers, the number of data types is limited to those understood by PLCs at the time. Large binary objects are not supported.
- No standard way exists for a node to find the description of a data object, for example, to determine if a register value represents a temperature between 30 and 175 degrees.
- Since Modbus is a master/slave protocol, there is no way for a field device to "report by exception" (except over Ethernet TCP/IP, called open-mbus)- the master node must routinely poll each field device, and look for changes in the data. This consumes bandwidth and network time in applications where bandwidth may be expensive, such as over a low-bit-rate radio link.
- Modbus is restricted to addressing 247 devices on one data link, which limits the number of field devices that may be connected to a master station (once again Ethernet TCP/IP proving the exception).

- Modbus transmissions must be contiguous which limits the types of remote communications devices to those that can buffer data to avoid gaps in the transmission.
- Modbus protocol provides no security against unauthorized commands or interception of data.[6]

Trade group

The *Modbus organization* is a trade association for the promotion and development of Modbus protocol.

References

1. ^ **a** **b** Bill Drury, *Control Techniques Drives and Controls Handbook (2nd Edition)* . 2009, Institution of Engineering and Technology, Online version available at: http://knovel.com/web/portal/browse/display?_EXT_KNOVEL_DISPLAY_bookid=2995&VerticalID=0, page 508 and following
2. ^ Modbus Messaging on TCP/IP Implementation Guide V1.0b, s3.1.3
3. ^ Java implementation (http://jamod.sourceforge.net/kb/modbus_udp.html)
4. ^ Modbus Application Protocol V1.1b (http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf)
5. ^ Gordon Clarke, Deon Reynders *Practical Modern Scada Protocols: Dnp3, 60870.5 and Related Systems* ,Newnes, 2004 ISBN 0750657995 pages 47-51
6. ^ Charles Palmer, Sujeet Shenoi (ed) *Critical Infrastructure Protection III: Third IFIP WG 11. 10 International Conference, Hanover, New Hampshire, USA, March 23–25, 2009, Revised Selected Papers* Springer, 2009 ISBN 3642047971, page 87

External links

- Modbus Organization site (http://www.modbus.org)
- Detailed Protocol Description (http://www.modbus.org/specs.php)
- Modicon - Modbus Protocol Reference Guide (http://www.modbus.org/docs/PI_MBUS_300.pdf)
- Modbus TCP (http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf)
- Protocol explanation for Java developers (http://jamod.sourceforge.net/kbase/protocol.html)
- Free Modbus Device Testing Software (http://www.globalmultimedia.in/modnet.htm)

Open-source software

- An Open Source Modbus library in C for GNU/Linux, Mac OS X, FreeBSD, and QNX (http://www.libmodbus.org/)
- QModBus (http://qmodbus.sourceforge.net/) is a graphical Modbus master application for Linux and Windows.
- Free Modbus ASCII/RTU and TCP for microcontrollers (http://freemodbus.berlios.de) . In C. New site location is http://www.freemodbus.org (http://www.freemodbus.org/) . A commercially supported version is available at http://www.embedded-solutions.at (http://www.embedded-solutions.at/) .
- Protocol::Modbus in Perl (http://search.cpan.org/~cosimo/Protocol-Modbus-0.05/lib/Protocol/Modbus.pm)
- Modbus::Client in Perl (http://search.cpan.org/~dvklein/Modbus-Client-1.03/lib/Modbus/Client.pm)
- Modbus master in Ruby (http://www.messen-und-deuten.de/modbus.html) . Public domain sample code, can easily be re-implemented in other scripting languages.
- rmodbus (http://rmodbus.heroku.com) . Free implementation of ModBus protocol in pure Ruby.
- jamod (http://jamod.sourceforge.net/) . Implementation of ModBus protocol in Java.
- Modbus4J (http://sourceforge.net/projects/modbus4j/) . Implementation of the ModBus protocol in Java. Part of the Mango M2M project.
- ModBus-Droid (http://www.bencatlin.com/sofware-projects/modbus-droid/) . Modbus Scanner for Android based on a modified version of the Modbus4J library.
- node-modbus-stack (https://github.com/TooTallNate/node-modbus-stack) . Open-source implementation of the Modbus protocol, written in JavaScript for NodeJS.
- pymodbus (http://code.google.com/p/pymodbus/) . Free implementation of ModBus protocol in Python.
- modbus-tk (http://code.google.com/p/modbus-tk/) . Another implementation of Modbus Protocol in Python
- MBLogic - Free implementation of Modbus/TCP in Python (http://mblogic.sourceforge.net)
- DelphiModbus - Free Delphi implementation of Modbus/TCP (http://sourceforge.net/projects/delphimodbus/)