MICROVA
electronics & automation

What is Modbus?

Introduction

Modbus is an industrial control "field bus" networking standard developed in 1979. Modbus is a master/slave (client/server) type network used in a wide variety of industries. Modbus networks are reliable, robust and easy to implement. Modbus is one of the most widely supported industrial control networks with tens of millions of devices installed worldwide and with thousands of Modbus device manufacturers. Modbus is an open standard with no licensing fees or certification requirements. The Modbus networking standard defines the physical and electrical wiring for connecting devices on the network as well as the data communication protocol. The Modbus organization oversees the standard and provides technical documents on their website www.modbus.org. Three of the most common Modbus physical and electrical standards are:

1. RS-232

RS-232 defines a single connection between just two devices, such as a computer to a printer. The RS-232 version of Modbus is called "Modbus ASCII". However, due to the fact that the data encoding format (ASCII) is inefficient and the fact that RS-232 allows only one node on the "network", this Modbus version is mostly obsolete.

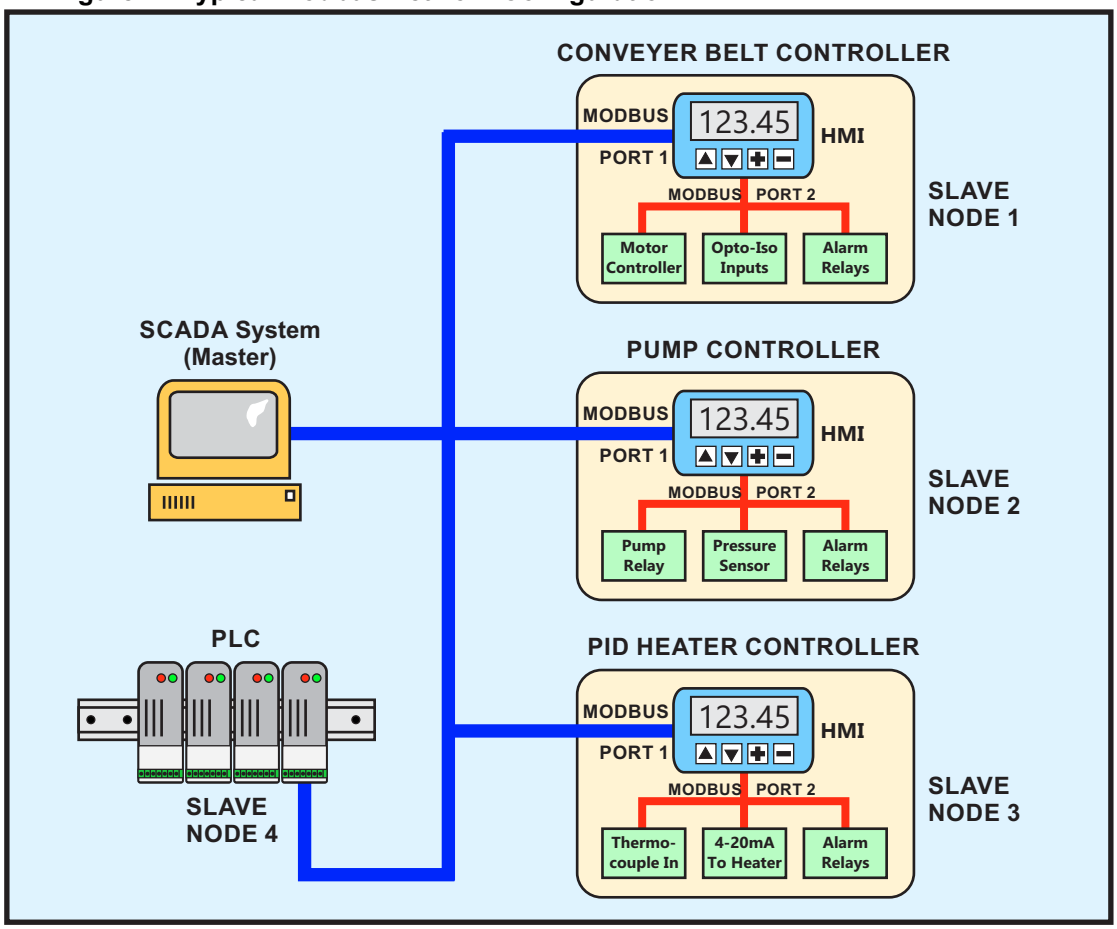
2. RS-485

RS-485 is one of the most widely used industrial networking physical layers. RS-485 uses a "differential balanced line" which provides great noise immunity, high speed (up to 35Mbps) and long distances (up to 4000 feet between nodes). Modbus running on an RS-485 network is called "Modbus RTU" (RTU stands for "remote terminal unit"). Modbus RTU allows one master and up to 247 slave devices on the network. Each slave must be assigned a unique node number. The master has no node number. All Modbus nodes on the RS-485 network must use the same communication settings which consist of the baud rate (eg. 19200) and the character data format (eg. 8/E/1). The figure below shows a typical Modbus network installation. The controllers shown use two separate Modbus networks: the primary network connected to the SCADA (Supervisory Control and Data Acquisition) system and a second internal Modbus network for the controller's I/O boards.

3. Ethernet/WiFi

Modbus over Ethernet or WiFi is called "Modbus TCP/IP" and allows Modbus devices to communicate using standard computer networking technology. Typically, Modbus TCP/IP serves as a gateway between a standard Modbus RTU network and a computer SCADA system.

Figure 1: Typical Modbus Network Configuration



The Modbus Application Layer

A big reason for the success of Modbus is the relatively simple application layer protocol. Many industrial networking standards are complicated because they explicitly define every detail for every type of node that can exist on the network. For example, the network standard might define a pressure sensor node which includes internal variables that hold data for the full scale pressure value and for the calibration span and offset value and for the temperature at the pressure location and for a table of data used to linearize the sensor, etc. And all of this data can be accessed on the network and that's just one type of device possible on the network. The standard might define hundreds of different sensor and actuator nodes such as thermocouples and isolators and relays and contact sensors and motor drivers, etc. Consequently, the network protocol documentation winds up being hundreds of pages.

Modbus does not work that way. There is no specification for devices. There is no standard Modbus pressure sensor for example. Modbus simply defines different data types and the Modbus device manufacturer uses the data in any way that is appropriate for that device. The Modbus master sends a "request" data packet to the slave which includes the slave node number, the function code, the first variable address, the number of variables to read or write and an error check code (CRC). The variable address is incremented for each variable read or written from or to the slave. The slave "response" packet includes the data requested or an acknowledgment if the request was for a data write or an error code if the data was corrupted or if the master tries to read or write to slave addresses that do not exist. The slave must respond within a "maximum response time" window or the master will assume there is a communication error. Response data packets from the slave back to the master also contain error check codes so the master knows if the returned data was corrupted. In this way Modbus ensures reliable data communication even in electrically noisy industrial environments.

Modbus Standard Function Codes

The table below shows the eight most common standard Modbus function codes which are supported by Modbus device manufacturers. These function codes are used to read or write single bits (called "coils" or "discrete inputs") or integer or float data values (called "registers"). Note that the original Modbus standard separated input registers, like from an analog input, from general purpose memory (hold registers). Likewise "coil" bits were separate from "discrete input" bits. The specification has since been revised so that the function codes are now just used to read or write single bits or data with no distinction between an input register and a hold register or between a coil and a discrete input. That is why some of the function codes are now redundant. For example, function codes 0x03 and 0x04 perform the same function. Modbus is able to read or write 16 bit or 32 bit integer or floating point data. One address is used for 16 bit data. Two addresses are needed for 32 bit data. For 32 bit data the device manufacturer will specify whether the lower address is the upper 16 bits of the 32 bit data and the higher address is the lower 16 bits of the data or vice versa. Modbus can also read or write text (character) strings such as the node identification string using these same function codes.

Modbus Variable Addresses

The original Modbus specification assigned logical addresses to variables based on the different data types, i.e. coils, discrete inputs, input registers and hold registers as shown in Table 1. These logical addresses are no longer used, however, many legacy systems still refer to these addresses which is a source of confusion. The actual addresses used in the Modbus data packets can be calculated from the logical addresses by subtracting the first logical address for the data type being read or written. For example, the variable address used in the Modbus data packet for the variable with logical address 40007 is equal to 6 = 40007 - 40001. And the logical address for variable address 15, when reading input registers, is 30016 = 15 + 30001. Microva devices always use the variable addresses used in the data packets not the logical addresses. For example, if a Microva Modbus device specifies the relay output 1 state variable is at address 5 then the address read or written in the Modbus data packet will be 5.

The device node number can be changed in the field but the variable addresses are fixed by the manufacturer. Sequential variable addresses can be read or written in one Modbus request packet by the master. For example, in Microva BASIC, to read the pressure, temperature and humidity at physical addresses 15, 16 and 17 from node number 2 using serial port 1, the statement would be:

```
J = Modbus1(RdVars, 2, 15, Pressure, Temperature, Humidity)
```

See Microva application note *"AN132 Microva Modbus RTU Implementation"* for more information.

Table 1: Commonly Supported Modbus Standard Function Codes

Function Code	Function Name	Description	Request Packet Size	Response Packet Size	Logical Address
0x01	read coils	read 1 to 2000 bits	8	5 or 6+N/8	00001 to 09999
0x02	read discrete inputs	read 1 to 2000 bits	8	5 or 6+N/8	10001 to 19999
* 0x03	read hold registers	read 1 to 125 regs	8	5+2N	40001 to 49999
* 0x04	read input registers	read 1 to 125 regs	8	5+2N	30001 to 39999
0x05	write single coil	write 1 bit	8	8	00001 to 09999
* 0x06	write single register	write 1 register	8	8	40001 to 49999
0x0F	write multiple coils	write 1 to 2000 bits	9 or 10+N/8	8	00001 to 09999
* 0x10	write multiple regs	write 1 to 123 regs	9+2N	8	40001 to 49999

* registers use 1 address for 16 bit data, 2 addresses for 32 bit data