

MODBUS Protocol

The Modbus is a serial communications protocol originally published by Modicon (now Schneider Electric) in **1979** for use with its programmable logic controllers (PLCs). Modbus has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices. It is one of the most popular protocols used in the industrial world. Supporting traditional serial protocols of RS232/422/485 and Ethernet protocols allow industrial devices such a PLCs, HMIs, and meters to use Modbus as their communication method. When communicating with Modbus via serial and [Ethernet](#) networks, a communication gateway is a necessity. Originally it was implemented as an application-level protocol intended to transfer data over a serial layer, Modbus has expanded to include implementations over serial, TCP/IP, and the user datagram protocol (UDP).

The Modbus communication [protocol](#) is a messaging structure used to establish a master-slave or client-server communication between intelligent devices able to support up to **247** slaves connected in a bus or a star network. The intelligent devices can be a PLC, HMI, PC, Distributed Control Systems (DCS), Supervisory Control and Data Acquisition Systems (SCADA), etc.

The Modbus protocol is not industry-specific and can be used in a wide variety of industries such as factory automation, building automation, process control, oil & gas, traffic systems, parking systems, agriculture & irrigation, water & wastewater, pharmaceutical and medical, material handling, etc.

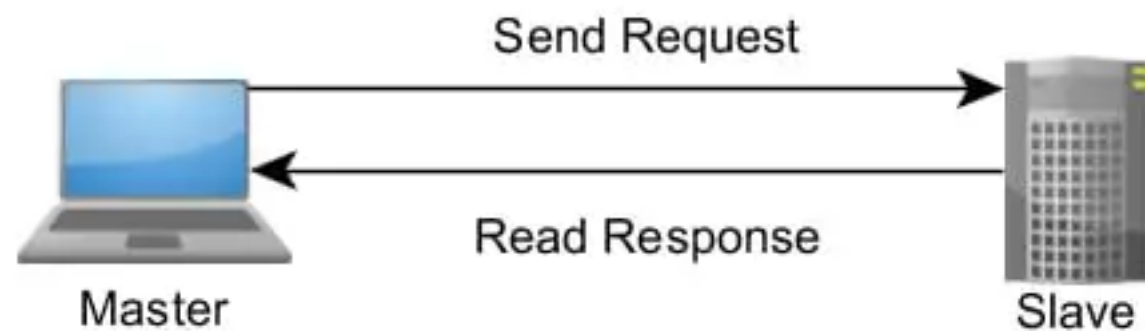
Applications of MODBUS:

The main reasons for the use of Modbus in the industrial environment are:

- It has been developed with industrial applications in mind.
- It is openly published and royalty-free.
- It is easy to deploy and maintain.
- It moves raw bits or words without placing many restrictions on vendors.
- Ease of availability of SCADA systems on the Modbus protocols.

How does MODBUS Protocol Works?

The Modbus is a request-response protocol implemented using a master-slave relationship. In a master-slave relationship, communication always occurs in pairs—one device must initiate a request and then wait for a response—and the initiating device (the master) is responsible for initiating every interaction. Typically, the master is a human-machine interface (HMI) or Supervisory Control and Data Acquisition (SCADA) system and the slave is a sensor, programmable logic controller (PLC), or programmable automation controller (PAC). The content of these requests and responses and the network layers across which these messages are sent are defined by the different layers of the protocol.



MODBUS Master-Slave Communication

The MODBUS is a protocol between a host (master) and devices (slaves) to access the configuration of the devices and to read the measures. MODBUS messages correspond to relatively simple operations to read and write 16-bit words and binary registers (often called “coils”). The host systematically initiates the exchange and the “slave” device answers. The slave doesn’t send any message before the host requests it.

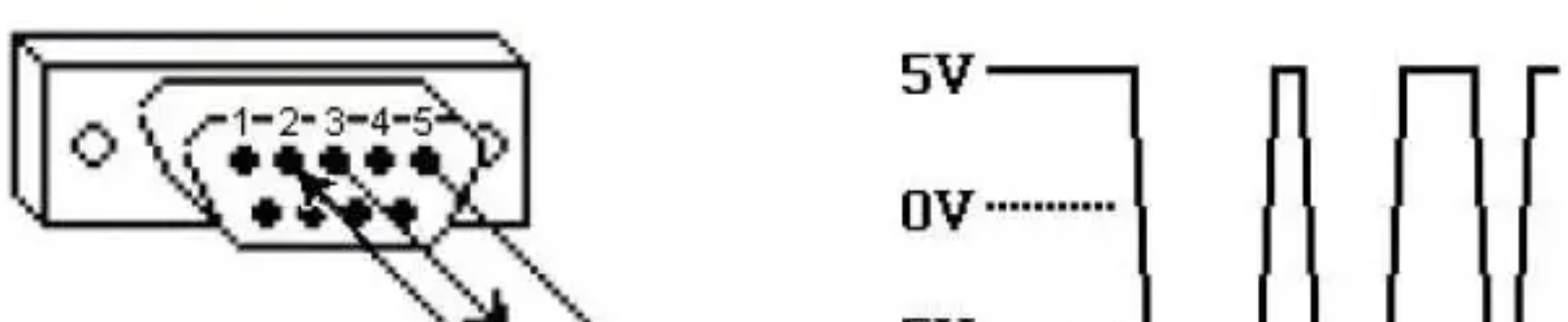
As there can be several devices connected in parallel on the RS485 bus, each slave device must use a unique MODBUS Slave ID on the bus. Each MODBUS request starts with the Slave ID of the intended device, each answer starts with the Slave ID of the slave sending it. So, in order for the MODBUS communication to work, you must check in the device configuration its Slave ID and change it if necessary. On the way, check also communication speed and parity (same principle as RS232).

You cannot easily craft MODBUS messages “by hand”, as you would have done with ASCII protocols used on RS232: each MODBUS message ends with a checksum code, computed from the full content of the message. To exchange MODBUS messages, you must therefore use:

- A specific program provided by the device vendor, with a compatible interface;
- A simple RS485 interface with a programming library which encodes and decodes MODBUS messages;
- A smart RS485 interface able to encode and decode by itself the MODBUS messages, such as the Yocto-RS485.

MODBUS Physical Layer:

The Modbus is transmitted over serial lines between devices. The simplest setup would be a single serial cable connecting the serial ports on two devices, a Master and a Slave. The data is sent as a series of ones and zeroes called bits. Each bit is sent as a voltage. Zeroes are sent as positive voltages and ones as negative. The bits are sent very quickly. The typical transmission speed is 9600 baud (bits per second).



MODBUS Physical Layer Communication

Modbus message structure:

The Modbus communication interface is built around the messages. The format of these Modbus messages is independent of the type of physical interface used. On plain old RS232 are the same messages used as on Modbus/TCP over ethernet. This gives the Modbus interface definition a very long lifetime. The same protocol can be used regardless of the connection type. Because of this, Modbus gives the possibility to easily upgrade the hardware structure of an industrial network, without the need for large changes in the software. A device can also communicate with several Modbus nodes at once, even if they are connected with different interface types, without the need to use a different protocol for every connection.

There are seven types of MODBUS Communication Protocol as per this standard which designed for different communication purposes as per their requirement communication message structure.

- Modbus ASCII.
- Modbus RTU.
- Modbus TCP.
- Modbus Plus.
- Modbus Daniels.
- Modbus Tek-Air.
- Modbus Omnicflow.

MODBUS ASCII Frame Structure:

Modbus ASCII is an older implementation that contains all of the elements of an RTU packet but expressed entirely in printable ASCII characters. Modbus ASCII is considered deprecated, is rarely used any more, and is not included in the formal Modbus protocol specification. The ASCII mode of the Modbus frame is having each byte is encoded in the serial link as 2 ASCII characters. Each ASCII character is transmitted as 1 start bit, 7 data bits, zero or 1 parity bit, one or two stop bits.

Modbus Register Types:

There are 4 types of MODBUS registers:

Modbus Function Codes:

The Modbus protocol defines several function codes for accessing Modbus [registers](#). There are four different data blocks defined by Modbus, and the addresses or registration numbers in each of those overlap. Therefore, a complete definition of where to find a piece of data requires both the address (or register number) and function code (or register type).

The function codes most commonly recognized by Modbus devices are indicated in the table below. This is only a subset of the codes available – several of the codes have special applications that most often do not apply.

Function Code	Register Type
1	Read Coil status
2	Read Discrete Input status
3	Read Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Holding Register
15	Write Multiple Coils
16	Write Multiple Holding Registers
17	Report slave ID

[Piest Forum - Android App Link](#)[Gift A Cup Of Coffee To PiEmbSysTech](#)[JOIN TELEGRAM FOR TECH. DISCUSSION](#)

Recent Posts

- [Understanding Inheritance in CPP Programming Language](#)
- [Objects of a Class in CPP Language](#)
- [Classes in CPP Programming Language](#)
- [How Crystal Oscillators Work in Microcontrollers: Ensuring Precise Timing and Stable Performance](#)
- [Download and Install Turbo C++ Compiler](#)

Archives

Select Month

Embedded Research Forum

Table Of Contents


- [8051 Microcontroller](#)
- [8085 Microprocessor](#)
- [8086 Microprocessor](#)
- [About Us](#)
- [Account](#)
- [Adaptive AUTOSAR](#)
- [Advanced driver assistance systems \(ADAS\)](#)
- [Arduino](#)
- [ARINC Protocol](#)
- [ARM Microcontroller](#)
- [Artificial Intelligence](#)
- [Assembly Language](#)
- [Automotive Architecture](#)
- [Automotive BAP Protocol](#)
- [Automotive ECU](#)
- [Automotive Protocols](#)
- [Automotive Safety](#)
- [AUTOSAR](#)
- [AUTOSAR DCM](#)
- [AVR Microcontroller](#)
- [Basic Electronics](#)
- [Basic Understanding Of VLSI](#)
- [Bluetooth Protocol](#)
- [Boot Loader](#)
- [ByteFlight Protocol](#)
- [C Plus Plus \(CPP\) Tutorial](#)
- [C-Language](#)
- [CAN Protocol](#)
- [CAN-FD Protocol](#)
- [CAN-TP Protocol](#)
- [Canalyzer](#)
- [Canoë](#)
- [CAPL Language](#)
- [ChibiOS/RT](#)
- [CMSIS-RTOS](#)
- [Contact Us](#)
- [Contiki RTOS](#)
- [Cookie Policy](#)
- [CPU Design](#)
- [Dashboard](#)
- [Disclaimer](#)
- [DMC](#)
- [DoCAN Protocol](#)
- [DoIP Protocol](#)
- [DSRC Protocol](#)
- [eCos RTOS](#)
- [Edit](#)
- [Embedded Linux](#)
- [EtherNet Protocol](#)
- [FlexRay Protocol](#)
- [FlexRay Transport Protocol \(ISO 10681-2\)](#)
- [Free RTOS](#)
- [Guest Post](#)
- [Home](#)
- [HTTP \(Hypertext Transfer Protocol\): An Overview of the Internet's Most Widely Used Protocol](#)
- [Hw/Sw Interface](#)
- [I2C Protocol](#)
- [INTEGRITY Operating System](#)
- [IoT](#)
- [ISO-15031 Protocol](#)
- [K-Line Protocol](#)
- [KWP-2000 Protocol](#)
- [LIN Protocol](#)
- [Linux Basics](#)
- [Linux Device Driver](#)
- [Linux IPC](#)
- [Linux Kernel](#)
- [Linux System Architecture](#)
- [Login](#)
- [Long Range Wide Area Network \(LoRaWAN\) Protocol](#)
- [Mastering MIPI I3C Protocol: A Comprehensive Guide to Efficient Communication Between Devices](#)
- [Mbed OS: Arm's Open-Source OS for IoT Devices](#)
- [Message Queuing Telemetry Transport \(MQTT\) Protocol](#)
- [Microcontroller](#)
- [MODBUS Protocol](#)
- [MOST Protocol](#)
- [Motor Design](#)
- [Nucleus RTOS](#)
- [OBD-II](#)
- [Operating System](#)
- [Order Received](#)
- [OSAL](#)
- [OSEK](#)
- [Payment](#)
- [Power Electronics](#)
- [PowerPC Processor](#)
- [Privacy Policy](#)
- [QNX RTOS](#)
- [Raspberry-Pi](#)
- [Register](#)
- [Resistor](#)
- [RIOT Operating System](#)
- [Robotics](#)
- [RT-Thread RTOS](#)
- [RTLlinux RTOS](#)
- [RTOS Concept](#)
- [SAE J1708 Protocol](#)
- [SAE-J1939 Protocol](#)
- [SENT Protocol](#)
- [SPI Communication Protocol: A Comprehensive Guide to Serial Peripheral Interface](#)
- [Subscription](#)
- [Terms and Conditions](#)
- [Thank You](#)
- [ThreadX RTOS: A Lightweight and Scalable RTOS for Embedded Devices](#)
- [TinyOS](#)
- [Transmission Control Protocol \(TCP/IP\)](#)
- [UART Protocol](#)
- [uC/OS RTOS](#)
- [UDS Protocol](#)
- [Understanding of Internet Protocol \(IP\) - The Backbone of the Internet](#)
- [USB Protocol](#)
- [User Datagram Protocol \(UDP\)](#)
- [V2X Communication](#)
- [VxWorks RTOS: A High-Performance Real-Time Operating System for Embedded Systems](#)
- [Wi-Fi Protocol](#)
- [Windows OS: An Evolution in GUI Based OS](#)
- [XBEE Protocol](#)
- [XCP Protocol](#)
- [Zephyr RTOS](#)

- [Automotive Electronics](#)
- [Computer Science](#)
- [Electronics Technology](#)
- [Linux System](#)
- [Programming Language](#)

[C](#)[C++](#)[Python](#)[Robotics Technology](#)[VLSI](#)

☒ Subscribe


Connect with



Login


Please login to comment

2 COMMENTS


 Most Voted

swarup kumar nath

Very Good Explanation for MODBUS protocol.



0



Veerendra N n

Very nice Explanation for MODBUS protocol



0

