Modbus protocol over RS485

OVERDIGIT srl

innovative automation

Modbus protocol over RS485 - Introduction

Modbus is still one of the most popular communication protocols in the field of industrial automation despite its origins dating back to 1979. Its solid and long-lasting presence in the broad overview of industrial protocols is due to the fact that Modbus is a simple and direct protocol. It is also royalty-free and flexible so it can be freely implemented and adapted to the most varied applications.

The hardware and software resources required by the Modbus protocol are rather small and therefore it can be used also in systems based on small and cheap microprocessor. The simple and essential structure of the data transmitted allows for effective communication with a good ratio between the useful part of the information and the total number of bytes transmitted. The protocol also lends itself to partial implementations of commands, limited to specific requirements, as well as expansion with the addition of custom commands by the device manufacturer.

Very often the manufacturers of automation devices “fall” in the temptation to develop a fully proprietary communication protocol thinking of doing the simplest and most effective thing. However, these protocols are often a “bad copy” of an already simple and direct protocol such as Modbus, but are not compatible with this and with other standard protocols. This can be an advantage if they intend to constrain customers to their protocol but exclude interoperability with many other commercial devices and the possibility that the product itself is more marketable.

The Modbus protocol is based on fieldbus and this means it is suitable for exchanging information between devices physically distinct and positioned even at considerable distances between them. The realization of an automatic machine or a plant by means of a set of interconnected devices has many advantages such as:

- Possibility to realize the automation plant using commercial devices, made by different manufacturers and easily replaceable.
- Distribution of the various devices in strategic locations for the plant, even remote, with significant reduction and savings in I/O cabling.
- Fractionation of the plant resulting in simplification of design development and maintenance/repair operations.

The devices are interconnected by cables with only few conductive poles to form a distributed network even at high distances. The Modbus protocol provides several types of physical connection between the devices. In particular, the most common connection are:

- RS485 network - Serial Modbus Protocol
- Ethernet network - TCP/IP Modbus Protocol

The type of connection adopted defines what is normally called the “Physical layer” of the protocol, that is, the hardware on which the information is exchanged. In this series of articles, only the RS485 serial network Modbus protocol will be explained. Such serial network requires cables with only two poles to send data from any device to another.

The next level of protocol, the “Data Link layer”, is instead of software type and includes the specifications of the data frame (byte sequences) exchange between one device and the other. This part doesn’t address the meaning of the bytes but only deals with sending them on the fieldbus, checking timing and errors by checksum.

The Modbus protocol is of Master/Slave type and therefore there is always a network Master device that manages communication with one or more Slave devices.
Each exchange of information is originated by the Master who sends a bytes frame on the fieldbus containing a particular request, normally a read or write command of the informations contained in one of the Slave. All Slaves are normally receiving and listening to Master’s requests. Only the addressed Slave captures the information sent by the Master, executes the command and responds to the Master by sending its own information on the network.

The modalities according to which the possible requests of the Master and the corresponding Slave responses are encoded within the communication frames are defined by the next level of the protocol, also known as the “Application protocol”. It is with this last level of the protocol that the specific application of the device interacts, in the case of the Master usually the PLC automation program, in the case of the Slave the firmware for handling the specific I/O of the device. This layer constitutes the interface of the protocol with all the rest of the device software.

The description of the protocol, divided in subsequent implementation levels, is defined by the ISO/OSI model frequently used to represent communication protocols:

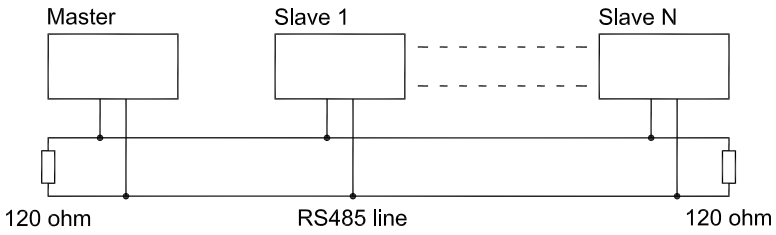
Layer	ISO/OSI Model	
7	Application	Modbus Application Protocol
6	Presentation	Not used
5	Session	Not used
4	Transport	Not used
3	Network	Not used
2	Data Link	Modbus Serial Line Protocol
1	Physical	EIA/TIA-485 standard

Note that in the case of the serial Modbus protocol, levels 3 to 6 are not used while these are part of a communication structured on an Ethernet network.

In the following articles of the “Modbus over RS485” series, the three levels used by the serial protocol will be analyzed in detail by referring to the official documentation of Modbus. In addition, a final article will provide information on the specific implementation of the protocol in the Overdigit Slave modules. This article represents a concrete example of the implementation of Modbus communication over RS485 and how the protocol can be expanded to increase its basic features.

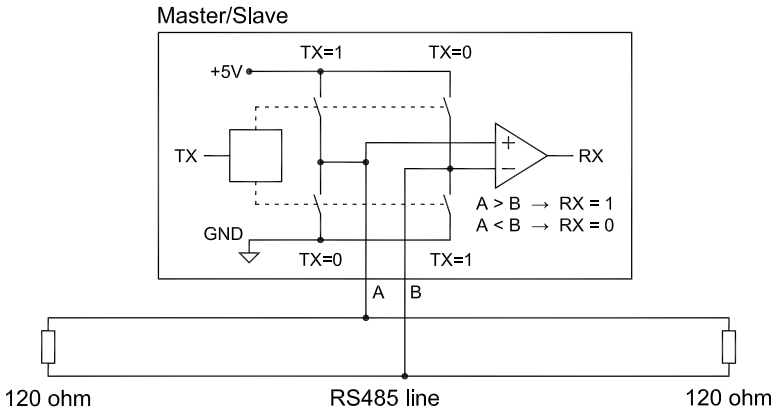
Modbus protocol over RS485 - Physical layer

The Modbus protocol on RS485 serial network is certainly one of the most popular implementation thanks to its simplicity, economy and reliability in the industrial environment. It is realized by means of a two-wire cable that connects all the devices on the network in parallel. As a Master/Slave protocol, only one Master device and one or more Slave devices will always be present:



The two-wire connection is as economical as possible but at the same time it offers excellent performance in terms of communication speed and immunity to any electromagnetic disturbance signals. This is thanks to the particular technique of using the electrical signals applied to the pair of wires that will be described below.

Bits 0/1 communication between the devices occurs by applying on the two wires of the couple a small continuous voltage whose polarity changes according to the 0/1 logic level to be transmitted:



Normally, in the absence of transmission by the device, all four switches of the transmission block are OFF and therefore both RS485 line conductors are not connected to any electrical potential. However, it is preferable not to leave the line floating and therefore a default potential is always applied, positive to A and negative to B. This polarization is obtained by connecting two resistors, one between the power supply + 5V and the signal A and another between signal B and GND reference.

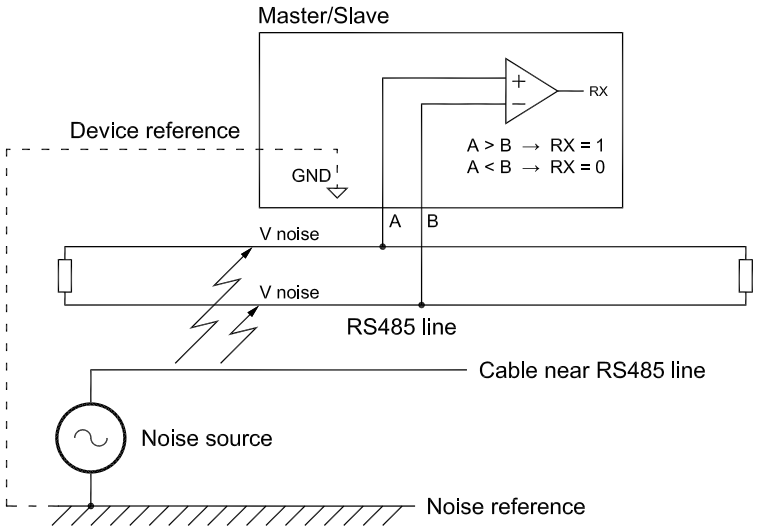
When a device needs to transmit bits, it temporarily takes control of the line by switching its two-to-two diagonal switches. To transmitt a logic 0, the line is forced with A=GND and with B=+5V, while for transmitting a logic 1 the line is forced with A=+5V and with B=GND. In this way, the voltage measured between conductor A and conductor B will be +5V or -5V respectively to transmit bit 1 or bit 0.

Each device also includes a reception block that can determine the polarity of signal A relative to that of signal B. Therefore, the receiving interface is able to measure the voltage difference between pole A and pole B.
If this difference is positive (A > B) reception corresponds to logic state 1, if the difference is negative (A < B), the received logic status is 0. The reception block then allows to perform the inverse process, decoding the 0/1 information previously encoded with the transmitter block.

NOTE: the transmitter block was ideally represented by switches. In reality, these switches are made with transistors capable of switching high-speed signals. A wide range of integrated circuits have been developed as RS485 interface drivers by incorporating both the transmission and receiving parts inside them.

This driving technique of the serial line originates the name “differential signal” referring to that applied to the A/B wires of the line. Note that in a differential connection it is not strictly necessary to connect the common GND reference of a device to that of the others, as it normally happen when connecting signals between two distinct devices. In fact, in the case of the differential signal, it is not important the voltage of A or B with respect to GND, but the relative difference between A and B. Differential signals are very much used, such as in the USB connection, for their high performance associated to the maximum simplicity of the physical support.

A great advantage of the differential signals is achieved realizing the cable with a pair of conductors tightly interconnected, obtaining a high degree of equality of the two wires in terms of physical location in the space. Any interference, caused by the proximity of the line to the cables of other systems, will cause a disturbance voltage on both pair wires, injecting undesired voltages over these with the same polarity (positive or negative) respect to the reference:



The receivers of each device, which perform a voltage difference evaluation between the A and B wires, will then be able to eliminate this disturbance component because of the same sign and entity on both wires. That is why it is important that any disturbance interferes in the same way with both conductors of the line.

In the previous diagrams two 120 ohm resistors at the ends of the RS485 line were also highlighted. The purpose of these resistances is to define the load impedance in the terminal parts of the line in order to avoid that, in the propagation of signals along the wires, reflections are generated at the open ends, resulting in alteration of the signal waveform.
This phenomenon is more evident as the signal frequencies are high, but also at the lower frequencies of fieldbus usage, these effects can be noticed. Then the insertion of line termination resistors is strictly required in all applications.

In addition to the correct termination of the differential line it is very important that the connection of this to the various devices takes place in a single linear path, ie without branches. Attention to this rule must be adopted also in the point of line connection to the two terminals of each device. The cable line must be interrupted only at the terminal block of each device, tightening the incoming cable with the outcoming cable directly in the same terminals. Only the first and last device, whether Master or Slave, should also have connection to termination resistance. This resistance is usually incorporated into the devices and can be enabled by a specific switch.

The RS485 serial network makes it possible to create distributed systems by placing the various devices at considerable distances between them. The maximum length of the connection depends on many factors such as communication speed (typically from 9600 to 115200b/s for the Modbus protocol), cable quality and features of the device hardware driver.
Using specific cables for differential lines such as those produced by Belden, the maximum reachable distance is 1200m with communication speeds in the range of 10Kb/s and 100Kb/s, while, with increasing speeds up to 1Mb/s, the distance drops proportionally to 120m. These are typical specifications for cable only, but there are many other factors that depend on the particular application, such as the near environment and the accuracy of the installation, which can lead to a significant reduction in the maximum distance reached for a given communication speed.

The maximum number of devices that can be connected to the RS485 network depends primarily on the characteristics of the driver integrated circuits used in the interfaces of the individual nodes.
The EIA/TIA-485 specification requires that the transmitter circuit be capable of driving up to 32 load units where a load unit (UL) equals an impedance of about 12Kohm. The ever-evolving technologies of driver chips has introduced in the market devices capable of reaching fractions of the load unit so that the maximum number of devices can be greater than 32. For example, components with 1/2UL allow to connect up to 64 devices, while with 1/8UL up to 256 nodes can be reached. Note, however, that this data is based solely on the evaluation of the unit load, without considering the polarization resistances of the bus and many other elements of the specific application, in particular the quality and length of the connections and also the speed of communication that can substantially reduce the maximum number of devices.

Modbus protocol over RS485 - Data Link layer

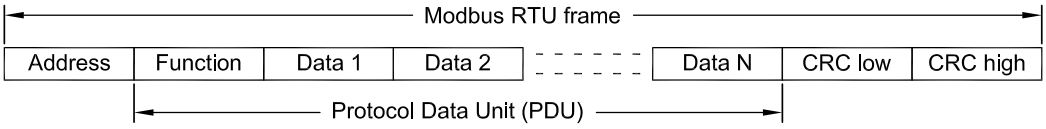
The “Data Link layer” is the first of the software type levels of the Modbus protocol and defines how the bytes are exchanged between one device and the other, however, regardless of their specific meaning. This level provides two different methods for transmitting information.

Transmission of RTU type involves the exchange of binary frames where the bytes can assume all the possible values 0÷255. Since the start and end of each communication frame can not identified by specific bytes values (synchronization characters), timers that control the exact timing of the transmission must be used. In some systems, managing these timings, which can be very short at the highest communication speeds, can be heavy for processing and problematic.

The ASCII transmission is obtained by encoding the value of each byte of the information with the sequence of two hexadecimal characters (digits 0÷9, A÷F). The ASCII protocol is less efficient than the RTU protocol since each byte of information is encoded by two characters. However, this coding allows the reservation of special characters for the purpose of being used as start and end frame indicators (sync characters). This greatly simplifies transmission management especially in slower systems or not equipped with particular hardware devices to implement the timing control needed instead to the RTU protocol.
In this series of articles, only the Modbus RTU serial protocol will be described.

RS485 serial network communication is half duplex, meaning that only one device at a time can transmit bus information. This requires the transmission of the various network nodes to be coordinated in such a way that there are no overlapping of the driving of differential line by more than one device at a time.
Communication coordination is obtained by assigning the Master role to one of the nodes, while the other ones have the Slave role. Typically, the Master function is assumed by the system logic controller, such as the PLC, while all peripheral and I/O expansion devices work as Slave.

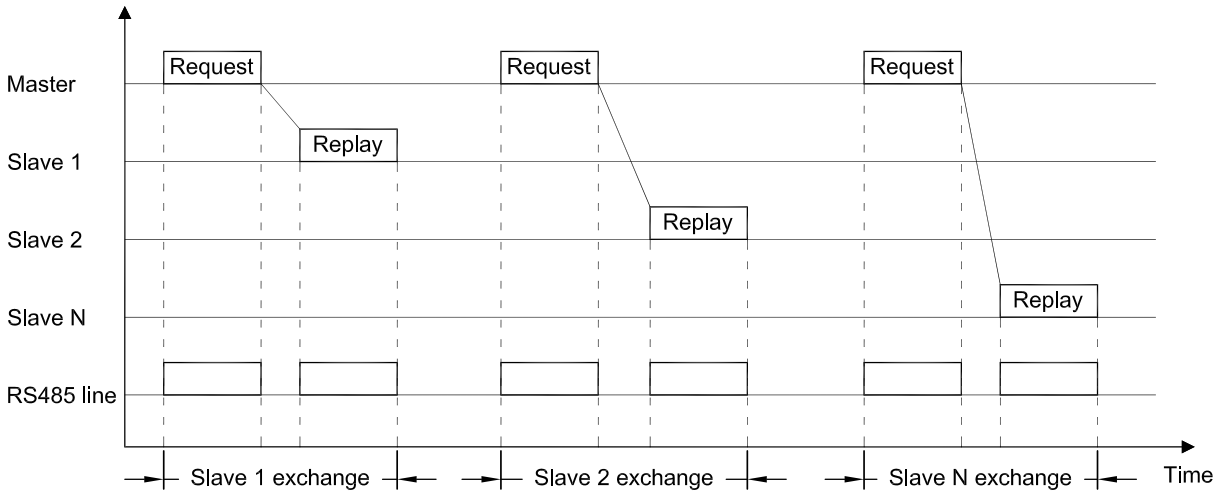
The Master controls communication in the sense that it is the only one authorized to initiate an exchange of information. Periodically or by necessity, the Master starts communication with one of the Slave by sending a bytes packet on the bus containing the address of the Slave concerned, the function to be executed, any data associated with the function and the checksum for packet control:



The Address field consists of a single byte containing the address of the selected Slave. The values allowed by the protocol are in the range 0÷247.
Also, for the Function field, only one byte is used and this contains the command code of the specific request to the Slave. Some command-specific Data bytes, up to a maximum of 252 bytes, can be associated with the function code. The function code values and the meaning of

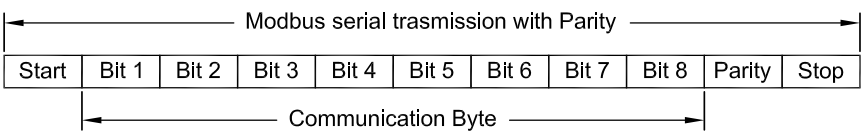
the data associated to the command will be defined in the next protocol layer (Application layer). The set of function field and data field is called Protocol Data Unit (PDU) and represents the useful part of communication.
Two bytes containing the lower and upper part of the checksum word (CRC), calculated on all previous bytes, are added at the end of the frame. This word is a signature about the content of the frame in order to identify any errors in the transmission of the bytes.

All Slaves, usually in “listen mode” on the bus, will receive the frame transmitted by the Master but only the addressed one will continue the communication while the others will ignore the request.
The selected Slave will respond to the Master by sending a frame with the same structure as the one described above.
After completing the exchange of frames with a Slave, the Master will be able to start a new communication sequence with the same Slave or with one of the others in the same manner:



The addressed Slave must replay within the shortest possible time, even in order not to slow down the communication operations with all the others Slave. Additionally, it is important to manage in the Master a predetermined time (Timeout) within which the Slave must replay to avoid blocking the communication in the case of missing or faulty Slave.
The Modbus protocol does not specify how to update the information of the Slave so that the Master may execute requests arbitrarily, as needed, or periodically (polling) to keep the status of the Slave resources constantly updated.

Each byte of the frame is transmitted on the RS485 fieldbus by serialization of a total of 11 bits, including the start bit, parity bit, and stop bit:



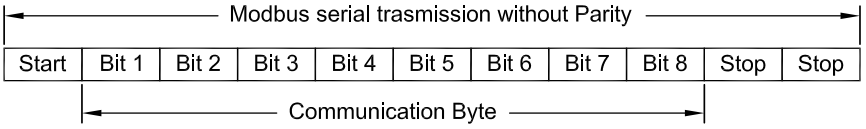
The start bit always applies one 0 level and constitutes a synchronization reference to initialize the reception of all character bits. After the start bit, the 8 bits of the byte are sent starting with

the least significant bit. Subsequently, a parity bit is transmitted for error control over one data bit and finally a stop bit (fixed value to 1).
The Modbus protocol allows one of the following modes to be used for the parity bit:

Parity	Rule for define the parity bit
No	Fixed to 1
Even	Total of bits (data + parity) equal to 1 is even
Odd	Total of bits (data + parity) equal to 1 is odd

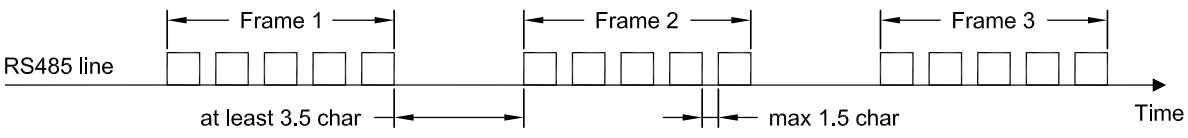
The Modbus specification, even though it gives full freedom, to the device's user, in choosing the parity, forces the manufacturer to implement at least Even parity in the device, while recommends the availability of the "No parity" configuration.

Note that in the case of "No parity" the parity bit is still in tenth position and fixed at logic 1 value. Also considering the stop bit in eleventh position, the transmitted character then always has a total of 11 bits, even in the absence of parity, as if the "No parity" configuration corresponds to the presence of two stop bits:



It is quite common, in various implementations of the Modbus protocol, to find a character bits configuration outside the official specification. In fact, in the case of "No parity", some devices consider the total absence of the parity bit, bringing to 10 the total number of character bits. This configuration corresponds, for example, to the classic "9600,N,8,1" (in case of 9600b/s communication speeds) that is particularly used on systems that do not allow parity management or on PC systems.

In the Modbus RTU protocol, all bytes of the same frame must be sent under strict temporal specifications, because they do not have special characters to use as synchronism and the packets are recognized and isolated from each other based solely on their timings:



Each frame must necessarily have all its bytes close to each other with a maximum time between the end of a byte transmission and the start of the next, equal to 1.5 times the duration of the character itself.

In addition, to distinguish a frame from the adjacent one, after the transmission of a frame, there is a pause of 3.5 times the duration of a character. This end frame pause, combined with the “compactness” of the frame bytes, guarantees the ability to distinguish and separate all the frames between them.

A transmitting device must then ensure that the frame bytes are contiguous, for example by preparing all the bytes up to the checksum in a transmission buffer and subsequently activating the entire buffer trasmission on the serial line.
Receiving devices will “capture” immediately all the bytes of a frame, continuosly reactivating a timer (1.5 character) on each byte received to control bytes contiguity and a timer (3.5 characters) to verify the effective end of a frame. After receipt of the entire frame, checksum value and match of Slave address will be checked.
Only the next level of the protocol (Application layer) will analyze the correctness of the received PDU and in case of success it will process it.

Regarding the address field of the communication frame, the Modbus protocol uses values in the range 1÷247 to communicate with only one specific Slave at a time (Unicast mode) while the special address 0 is used for Broadcast mode. A Master request with address 0 is processed by all Slaves but none of them will respond to the Master to avoid transmission conflicts on the bus. This mode is used to send the same command at the same time to all Slaves without receiving a response and then without any confirmation of the command execution by the Slave.
Finally, the addresses from 248 to 255 are “reserved” and allow the manufacturer to implement specific features.

The terminal field of frame consists of two bytes corresponding to the lower and upper part of the checksum word (CRC). This value allows the verification of the integrity of the packet bytes as it is a signature for the bytes values that precede the CRC field. In transmission, the checksum is calculated on all the bytes from the first (address) to the last of the Data field and added at the end of the frame. In receiving, the checksum is calculated on all bytes of the received frame (CRC field included). In this case, only a zero result indicates the correctness of the received frame.
Two different approaches can be used to calculate the CRC. The first is a complete algorithm that processes the values of all bytes considered and determines the 16 bit value of the CRC. The second is simpler and faster from the calculation point of view but requires the use of a table of 256 bytes pre-calculated constants and this, on microprocessors with little memory program, can be more onerous. For more details on these algorithms, refer to the official document of the serial Modbus protocol:

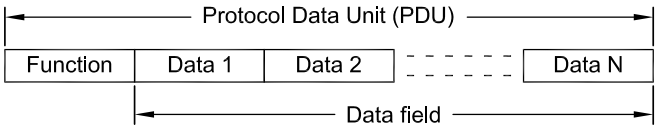
http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

Abnormal situations may occur during frame communication such as the receiving an incomplete frame or incorrect checksum. In this cases, Slave devices must discard these frames and not replay, while the Master, if it founds errors in a Slave response, may decide to retry the frame exchanging.

Modbus protocol over RS485 - Application layer

The second and last level of the Modbus protocol, known as the “Application layer”, deals with the Protocol Data Unit (PDU), that is, the useful part of the information contained in the frames exchanged between the Master and the Slave. In fact, in the previous layer (Data Link layer), only the parts of the protocol relating to the transmission of PDUs are defined without considering their significance.

The PDU contains all the information necessary for the Master to perform operations on a Slave, such as writing or reading a parameter.
For this reason, the bytes of a PDU are subdivided into the first field (1 byte length) defining the particular operation to be performed (function code) and a possible second Data field specific of the command to execute:



The function field contains a default value for each implemented command. The Modbus protocol provides that only values from 1 to 127 are used to encode the command, leaving values from 128 to 255 for encoding the same command (but with the highest bit to 1) in order to report, in the Slave response, an “exception” that is an error detected during its execution.
The Modbus protocol divides the set of numeric codes into two groups. The codes included in the 1÷64, 73÷99 and 111÷127 fields are used for “public” functions, ie officially documented and therefore unique for the protocol. The remaining codes in the fields 65÷72 and 100÷110 are free for the implementation of custom protocol functions.

The meaning and length of the Data field depend on the specific command and this part may also not be present. The maximum Data field length is 252 bytes because the total sum of bytes of a frame, according to the specification, is 256 bytes.
The protocol manages 16bit-word values using the “big-Endian” notation, which requires that the most significant byte of the word be first transmitted. Note that this is not the case for sending the CRC that is transmitted with the least significant byte first.

The standard commands of the Modbus protocol, used for reading and writing the Slave resource values, refer to the concept of “Object” instead of the real memory address of the device. This allows the protocol to be free from the effective physical locations of the resources within the Slave’s internal memory.
For this reason, each device must define the objects that are accessible to the protocol by assigning it to a Modbus address and consequently have to deal with objects association with the effective internal variables in own memory.

The objects are grouped into 4 different typologies with a clear reference to the type of resources that the devices offered during the period in which the protocol had been defined:

Primary tables	Object type	Access type	Comments
Discrete Inputs	Single bit	Read-Only	Provided by an I/O slave
Coils	Single bit	Read-Write	Can be alterable by Master
Input Registers	16-bit word	Read-Only	Provided by an I/O slave
Holding Registers	16-bit word	Read-Write	Can be alterable by Master

“Discrete Inputs” resources refer to digital input (ON/OFF) values while the “Coils” resources to digital output values and therefore are writable but at the same time readable. “Input Registers” and “Holding Registers” resources are analogue versions (continuous values in range 0-65535) of the previous and used for analog inputs/outputs or any other numeric value of the device such as a work parameter.

These objects are associated with both the identification numbers and the specific addresses in order to access these resources through the protocol PDUs. Remember that the address of the resource does not necessarily coincide with the physical memory address inside the device. It is task of the latter to associate, on the fly, the Modbus address with the internal memory variable, using, for example, matching tables.
These tables define what is called “Data Model” of the specific device and constitute a sort of “programmer reference manual” as these list all possible resources available in the Slave and those that are the addresses for the protocol.

Modbus addresses of objects, defined by a word, can extend across the range from 0 to 65535, while the number of objects is numbered from 1 to 65536. However, the numbering of objects is only formal, whereas what is important for the resources access is only the address entered in the PDU.
For each of the 4 subdivision areas, the entire range 0 to 65535 can be used as the function code will define to which of the four areas the address refers.
This extended way, which allows the addressing to 65536 objects for each of the four areas, is alongsided by another convention originally defined by the protocol and still used. In this convention, resources are counted in a narrow range of values from 0001 to 9999 by adding an offset, multiple of 10000, depending on the area concerned:

Primary tables	Type prefix	Object Number	PDU Address
Coils	0x	00001÷09999	0000÷9998
Discrete Inputs	1x	10001÷19999	0000÷9998
Input Registers	3x	30001÷39999	0000÷9998
Holding Registers	4x	40001÷49999	0000÷9998

It is the object number that define to which of the four areas refers and with which function code the related operations must be performed.

The standard function codes predefined by the Modbus specification are few and much of these are used to read or write bit type variables and word type variables:

Function code	Related area	Operation
1	Coils	Read up to 2000 contiguous memory bits
2	Discrete Inputs	Read up to 2000 contiguous input bits
3	Holding Registers	Read up to 125 contiguous memory words
4	Input Registers	Read up to 125 contiguous input words
5	Single Coil	Write one memory bit
6	Single Register	Write one memory word
15	Coils	Write up to 1968 contiguous memory bits
16	Holding Registers	Write up to 123 contiguous memory words
23	Holding Registers	Read up 125 and write up 121 memory words

Function code 1 - Read Coils			
Request	Function code	1 byte	0x01
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Coils	2 bytes	1 to 2000 (0x001 to 0x7D0)
Replay	Function code	1 byte	0x01
	Bytes count	1 byte	N/8 or N/8+1 (N is the Quantity of Coils)
	Coils status	n bytes	n=N/8 or n=N/8+1 if (remainder of N/8) ≠ 0
Notes	The first bit addressed is in the bit 0 position of the first byte of replay. If N is not a multiple of eight the remaining bits in the last byte are 0 padded.		

Function code 2 - Read Discrete Inputs			
Request	Function code	1 byte	0x02
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Inputs	2 bytes	1 to 2000 (0x001 to 0x7D0)
Replay	Function code	1 byte	0x02
	Bytes count	1 byte	N/8 or N/8+1 (N is the Quantity of Inputs)
	Inputs status	n bytes	n=N/8 or n=N/8+1 if (remainder of N/8) ≠ 0
Notes	The first bit addressed is in the bit 0 position of the first byte of replay. If N is not a multiple of eight the remaining bits in the last byte are 0 padded.		

Function code 3 - Read Holding Registers			
Request	Function code	1 byte	0x03
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 125 (0x01 to 0x7D)
Replay	Function code	1 byte	0x03
	Bytes count	1 byte	2 x N
	Registers value	2N bytes	N is the Quantity of Holding Registers

Function code 4 - Read Input Registers			
Request	Function code	1 byte	0x04
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 125 (0x01 to 0x7D)
Replay	Function code	1 byte	0x04
	Bytes count	1 byte	2 x N
	Registers value	2N bytes	N is the Quantity of Input Registers

Function code 5 - Write Single Coil			
Request	Function code	1 byte	0x05
	Coil address	2 bytes	0x0000 to 0xFFFF
	Coil value	2 bytes	0x0000 for OFF or 0xFF00 for ON
Replay	Function code	1 byte	0x05
	Coil address	2 byte	0x0000 to 0xFFFF
	Coil value	2 bytes	0x0000 for OFF or 0xFF00 for ON

Function code 6 - Write Single Register			
Request	Function code	1 byte	0x06
	Register address	2 bytes	0x0000 to 0xFFFF
	Register value	2 bytes	0x0000 to 0xFFFF
Replay	Function code	1 byte	0x06
	Register address	2 byte	0x0000 to 0xFFFF
	Register value	2 bytes	0x0000 to 0xFFFF

Function code 15 - Write Coils			
Request	Function code	1 byte	0x0F
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Coils	2 bytes	1 to 1968 (0x001 to 0x7B0)
	Bytes count	1 byte	N/8 or N/8+1 (N is the Quantity of Coils)
	Coils status	n bytes	n=N/8 or n=N/8+1 if (remainder of N/8) ≠ 0
Replay	Function code	1 byte	0x0F
	Starting address	2 byte	0x0000 to 0xFFFF
	Quantity of Coils	2 bytes	1 to 1968 (0x001 to 0x7B0)
Notes	The first bit addressed is in the bit 0 position of the first byte of request. If N is not a multiple of eight the remaining bits in the last byte are 0 padded.		

Function code 16 - Write Holding Registers			
Request	Function code	1 byte	0x10
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 123 (0x01 to 0x7B)
	Bytes count	1 byte	2 x N
	Registers value	2N bytes	N is the Quantity of Holding Registers
Replay	Function code	1 byte	0x10
	Starting address	1 byte	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 123 (0x01 to 0x7B)

Function code 23 - Read/Write Holding Registers			
Request	Function code	1 byte	0x17
	Read start address	2 bytes	0x0000 to 0xFFFF
	Quantity to read	2 bytes	1 to 125 (0x01 to 0x7D)
	Write start address	2 bytes	0x0000 to 0xFFFF
	Quantity to write	2 bytes	1 to 121 (0x01 to 0x79)
	Write bytes count	1 byte	2 x N
	Write Registers value	2N bytes	N is the Quantity of writing Registers
Replay	Function code	1 byte	0x17
	Read bytes count	1 byte	2 x n
	Read Registers value	2n bytes	n is the Quantity of reading Registers

The Modbus protocol also provides error handling at the “Application layer” level by checking that Master’s requests are legitimate. If a Slave receives a PDU with values that are not compatible with its resources, it will not execute the requested command and will respond to the Master with a particular PDU (Exception) containing, in the Function field, the same command required but with the most significant bit set to 1. Subsequently to the Function field, it will transmit a data field, with 1 byte length, containing the code of particular error verified:

Exception			
Replay	Function code	1 byte	The function code of request + 128
	Exception code	1 byte	0x00 to 0xFF

Exception code	Name of error	Comments
1	ILLEGAL FUNCTION	Function code is not valid or implemented.
2	ILLEGAL DATA ADDRESS	Object address is not valid for the Slave.
3	ILLEGAL DATA VALUE	Writing value is not valid for the addressed object.
4	SLAVE DEVICE FAILURE	Fatal error occurred during the requested operation.

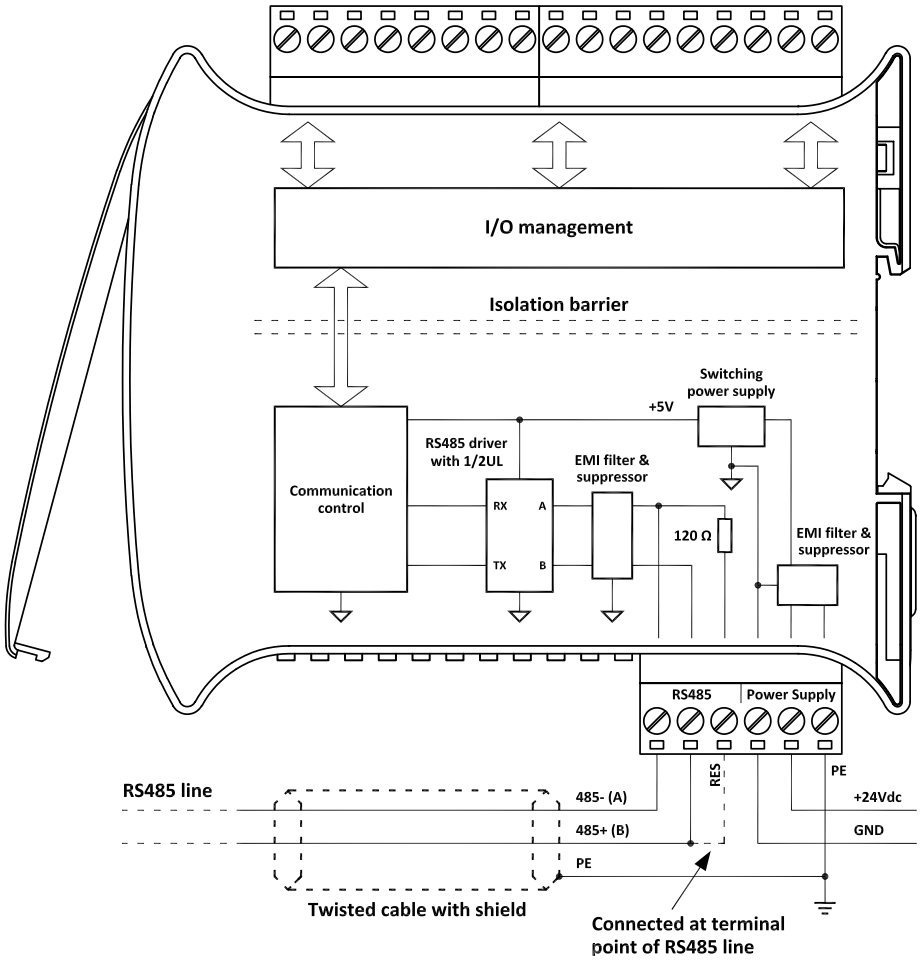
The Modbus protocol also defines other function codes used for service functions, status readings, diagnosis and device identification, as well as other additional error codes. For more details, refer to the official document of the Modbus protocol:

http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

Modbus protocol over RS485 - Overdigit Slave modules

Overdigit Slave modules conform to the official Modbus specification but at the same time take advantage of the protocol flexibility to achieve much higher performance than any other Modbus device on the market.
The performance increase can only be fully used in conjunction with Overdigit Web-PLCs, as the implementation of custom function codes is also required. However, this demonstrates the possibility of using the Modbus protocol also to meet the most specific needs, apparently achievable only with the development of fully proprietary protocols. In this way, it is possible to maintain compatibility, on the same fieldbus, between standard Modbus devices and systems that can achieve higher performance but still using the Modbus protocol as well.

With regard to the “Physical layer”, Overdigit modules are equipped with an RS485 driver with 1/2 load units. This means that up to 64 devices can be connected to the bus without using line repeaters. The 6-pin connector, standardized for all modules, allows powering the devices and connecting the serial interface to the fieldbus. Within the modules, the termination resistor is provided, whose insertion is obtained wiring an external jumper in the terminal block between the signal pole B and the RES pole:



The A and B signals are also indicated as 485- and 485+ by highlighting the polarity of the line in the active driving state of the bit=0, as opposed to the inactive state of the transmission. Note that inactive state, for the presence of the default polarization resistors, assumes the same polarity associated with the transmission of bit=1.

The RS485 line driver used in the Overdigit modules allows serial communications up to 1Mb/s maximum speed, even though Modbus systems on the market normally never exceed 115200b/s.

The “Data Link layer” implemented in Overdigit modules provides four modes for the parity bit:

Parity	Total bits	Comments
No	11	2 stop bits, compliant to specification
Even	11	Even parity + 1 stop bit, compliant to specification
Odd	11	Odd parity + 1 stop bit, compliant to specification
No-1stop	10	Parity bit not present (1 stop bit), out of specification

“No-1stop” mode is not included in the Modbus protocol specification, but it has been implemented to achieve compatibility with the many commercial systems that have “deviated” from the specification and have in fact created a rather widespread, but non-official, variant to the serial Modbus protocol.

With regard to the implementation of the Protocol Data Unit (PDU), defined in the “Application layer”, the Overdigit modules provide all the standard command codes required for the resources management of the specific module. To selecting the objects, extended addressing has been adopted so that the address field of the PDU can extend across the range 0 to 65535 for each of the four data areas defined by the protocol. In practice, however, for the operation of the various devices, only few addresses are normally needed starting from the value 0.

In addition, in the Overdigit modules, some custom function codes have been added, as permitted by the Modbus protocol, which allow to greatly speed up the operations for module resources update.
In particular, codes 100, 101, 102 have been added respectively to update all input, output, and input/output bytes of the module. In this case, all the I/O resources of the module are packed into a single bytes stream inside the Data area of the PDU.
The command code that makes the module updating most effectively is that used for reading inputs and writing outputs:

Function code 102 - Read and Write I/O Area			
Request	Function code	1 byte	0x64
	Write Out bytes count	1 byte	N
	Write Out bytes value	N bytes	N is the Quantity of writing Output bytes
Replay	Function code	1 byte	0x64
	Read In bytes count	1 byte	n
	Read In bytes value	n bytes	n is the Quantity of reading Input bytes

Using this command allows the refreshing of the entire image area of the module inputs and outputs in a single frame exchange. To do this in conventional Slave modules with Modbus protocol, multiple queries and word variables, even if the information to be exchanged can be contained in just one byte, must be used. In the case of Overdigit modules, I/O resources are instead inserted within a bytes array as long as necessary to contain the status of all resource variables.

The amount of I/O bytes can be known in advance or determined dynamically, for example at power-on, using the standard command code 17 (Report Slave ID):

Function code 17 - Report Slave ID			
Request	Function code	1 byte	0x11
Replay	Function code	1 byte	0x11
	Bytes count	1 byte	N
	ID bytes value	N bytes	N is the Quantity of Identification bytes

The data field in the response PDU consists of a single 20 bytes array that contains informations about the module and size of the I/O areas at predefined locations:

ID byte offset	Data type	Comments
0÷7	Fixed 8 bytes string	Module name, for example “EX1608DD“
8÷15	Fixed 8 bytes string	Firmware release, for example “r.01.008”
16-17	Word	Number of bytes for Input Area
18-19	Word	Number of bytes for Output Area

The use of command code 102, in conjunction with the maximum communication speed (1Mb/s), allows to update all the I/O area of the modules at considerably reduced times than is normally achieved on other commercial systems.

For example, in the case of the Overdigit EX1608DD module with 16 inputs + 8 digital outputs, the total communication time is only 250µs. Most of the similar modules on the market often limit the maximum speed to 38400b/s and do not include additional commands. For this reason, to update the I/O, a reading of one “Input Register” and a writing of one “Holding Register” are required. Considering also a zero time in frame processing and therefore an immediate response from the slave, a total communication time of about 13ms is necessary, which is 52 times greater.

Obviously such performances request also the implementation of custom functions, as is the case in the CoDeSys Modbus library of Overdigit Web PLCs, but the main advantage over the development of a fully proprietary protocol is that the main structure of the protocol remains compliant with Modbus standard, allowing compatibility of these modules with all the most traditional devices.