MODBUS

1. INTRODUCTION

Modbus is a serial communications protocol originally publishedby Modicon (now Schneider Electric) in 1979 for use with its programmable logic controllers (PLCs). Simple and robust, it has since become a *de facto* standard communication protocol, and it is now a commonly available means of connecting industrial electronic devices. The main reasons for the use of Modbus in the industrial environment are:

- developed with industrial applications in mind
- openly published and royalty-free
- easy to deploy and maintain
- moves raw bits or words without placing many restrictions on vendors

Modbus enables communication among many (**approximately 247**) 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*.

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**. 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 (1 to 247). 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).

Frame format

A Modbus frame is composed of an **Application Data Unit (ADU)** which encloses a **Protocol Data Unit (PDU)**:

- ADU = Address + PDU + Error check
- PDU = Function code + Data

IMPLEMENTATION OF MODBUS

The MODBUS standard defines an application layer messaging protocol, positioned at **level 7 of the OSI model** that provides "**client/server**" communications between devices connected on different types of buses or networks.

At the physical level, MODBUS over Serial Line systems may use different physical interfaces (**RS485, RS232**). TIA/EIA-485 (RS485) Two-Wire interface is the most common. As an add-on option, RS485 Four-Wire interface may also be implemented. A TIA/EIA-232-E (RS232) serial interface may also be used as an interface, when only short point to point communication is required.

The following figure gives a general representation of MODBUS serial communication stack compared to the 7 layers of the OSI model.

Layer	ISO/OSI Model	
7	Application	MODBUS Application Protocol
6	Presentation	Empty
5	Session	Empty
4	Transport	Empty
3	Network	Empty
2	Data Link	MODBUS Serial Line Protocol
1	Physical	EIA/TIA-485 (or EIA/TIA-232)

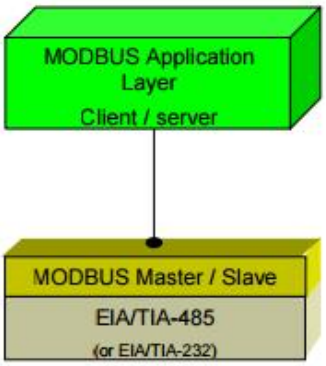

Figure 2: MODBUS Protocols and ISO/OSI Model

MODBUS application layer messaging protocol, positioned at level 7 of the OSI model, provides client/server communication between devices connected on buses or networks. On MODBUS serial line the client role is provided by the Master of the serial bus and the Slaves nodes act as servers.

2. MODBUS Data Link Layer

The MODBUS Serial Line protocol is a Master-Slaves protocol. Only one master (at the same time) is connected to the bus, and one or several (247 maximum number) slaves nodes are also connected to the same serial bus. A MODBUS communication is always initiated by the master. The slave nodes will never transmit data without receiving a request from the master node. The **slave nodes will never communicate with each other**. The master node initiates only one MODBUS transaction at the same time.

2.1 Modes

The master node issues a MODBUS request to the slave nodes in two modes :

In **unicast mode**, the master addresses an individual slave. After receiving and processing the request, the slave returns a message (a 'reply') to the master . In that mode, a MODBUS transaction consists of 2 messages : a request from the master, and a reply from the slave. Each slave must have an unique address (from 1 to 247) so that it can be addressed independently from other nodes.

In **broadcast mode**, the master can send a request to all slaves. No response is returned to broadcast requests sent by the master. The broadcast requests are necessarily writing commands. All devices must accept the broadcast for writing function. The address 0 is reserved to identify a broadcast exchange.

2.2 MODBUS Addressing rules

The MODBUS addressing space comprises 256 different addresses.

0	From 1 to 247	From 248 to 255
Broadcast address	Slave individual addresses	Reserved

The Address 0 is reserved as the broadcast address. All slave nodes must recognise the broadcast address. The MODBUS Master node has no specific address, only the slave nodes must have an address. This address must be unique on a MODBUS serial bus.

2.3 MODBUS frame description

The MODBUS application protocol [1] defines a simple **P**rotocol **D**ata **U**nit (**PDU**) independent of the underlying communication layers:

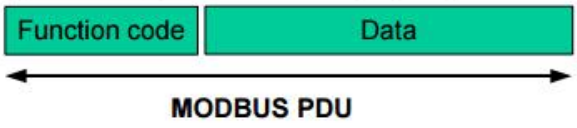

Figure 5: MODBUS Protocol Data Unit

The mapping of MODBUS protocol on a specific bus or network introduces some additional fields on the **P**rotocol **D**ata **U**nit. The client that initiates a MODBUS transaction builds the MODBUS PDU, and then adds fields in order to build the appropriate communication PDU.

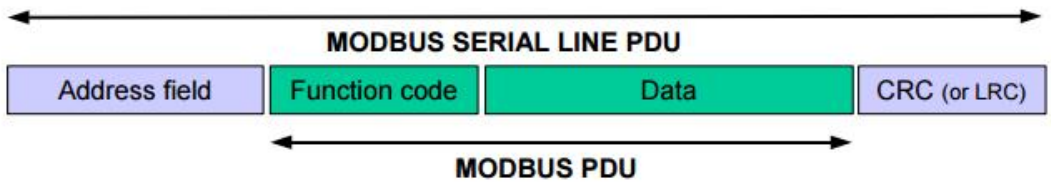

Figure 6: MODBUS frame over Serial Line

On MODBUS Serial Line, the Address field only contains the slave address. As described in the previous section the valid slave nodes addresses are in the range of 0 – 247 decimal. The individual slave devices are assigned addresses in the range of 1 – 247. A master addresses a slave by placing the slave address in the address field of the message. When the slave returns its response, it places its own address in the response address field to let the master know which slave is responding.

The function code indicates to the server what kind of action to perform. The function code can be followed by a data field that contains request and response parameters.

Error checking field is the result of a "Redundancy Checking" calculation that is performed on the message contents. Two kinds of calculation methods are used depending on the transmission mode that is being used (RTU or ASCII). (see 2.5 section, "The two serial Transmission Modes")

2.5 The two serial Transmission Modes Two different serial transmission modes are defined :

The **RTU mode and the ASCII mode**. It defines the bit contents of message fields transmitted serially on the line. It determines how information is packed into the message fields and decoded. The transmission mode (and serial port parameters) must be the same for all devices on a MODBUS Serial Line. Although the ASCII mode is required in some specific applications, interoperability between MODBUS devices can be reached only if each device has the same transmission mode : All devices must implement the RTU Mode. The ASCII transmission mode is an option. Devices should be set up by the users to the desired transmission mode, RTU or ASCII. Default setup must be the RTU mode.

RTU comes from SCADA, where the master is called a Central Terminal Unit (CTU). A CTU communicates to several RTUs at distant locations.

RTU Transmission Mode

When devices communicate on a MODBUS serial line using the RTU (Remote Terminal Unit) mode, each 8–bit byte in a message contains two 4–bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII mode for the same baud rate. Each message must be transmitted in a continuous stream of characters.

Frame Checking Field : Cyclical Redundancy Checking (CRC)

Frame description :

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low, CRC Hi

Figure 12: RTU Message Frame

➔ The maximum size of a MODBUS RTU frame is 256 bytes.

In RTU mode, message frames are separated by a silent interval of at least 3.5 character times.

CRC Checking

The RTU mode includes an error–checking field that is based on a **Cyclical Redundancy Checking (CRC)** method performed on the message contents. The CRC field checks the contents of the entire message. It is applied regardless of any parity checking method used for the individual characters of the message. The CRC field contains a 16–bit value implemented as two 8–bit bytes. **The CRC field is appended to the message as the last field in the message.** When this is done, the low–order byte of the field is appended first, followed by the high–order byte. The CRC high–order byte is the last byte to be sent in the message. The CRC value is calculated by the sending device, which appends the CRC to the message. The receiving device recalculates a CRC during receipt of the message, and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results. The CRC calculation is started by first pre-loading a 16–bit register to all 1's. Then a process begins of applying successive 8–bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and stop bits and the parity bit, do not apply to the CRC.

Calculating CRC value

During generation of the CRC, each 8–bit character is exclusive ORed with the register contents. Then the result is shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position.

The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value. If the LSB was a 0, no exclusive OR takes place. This process is repeated until eight shifts have been performed. After the last (eight) shift, the next 8–bit byte is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above.

The final content of the register, after all the bytes of the message have been applied, is the CRC value. When the CRC is appended to the message, the low-order byte is appended first, followed by the high-order byte.

A procedure for generating a CRC is:

1. Load a 16–bit register with FFFF hex (all 1's). Call this the CRC register.

2. Exclusive OR the first 8–bit byte of the message with the low–order byte of the 16–bit CRC register, putting the result in the CRC register.

3. Shift the CRC register one bit to the right (toward the LSB), zero–filling the MSB. Extract and examine the LSB.

4. (If the LSB was 0): Repeat Step 3 (another shift). (If the LSB was 1): Exclusive OR the CRC register with the polynomial value 0xA001 (1010 0000 0000 0001).

5. Repeat Steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8–bit byte will have been processed.

6. Repeat Steps 2 through 5 for the next 8–bit byte of the message. Continue doing this until all bytes have been processed.

7. The final content of the CRC register is the CRC value.

8. When the CRC is placed into the message, its upper and lower bytes must be swapped as described below.

The security of standard MODBUS Serial Line is based on two kinds of error checking :

1. Parity checking (even or odd) should be applied to each character.

2. Frame checking (LRC or CRC) must be applied to the entire message.

3. Physical Layer

3.1 Preamble

A new MODBUS solution over serial line should implement an electrical interface in accordance with EIA/TIA-485 standard (also known as **RS485 standard**). This standard allows **point to point** and **multipoint** systems, in a "**two-wire configuration**". In addition, some devices may implement a "**Four-Wire" RS485-Interface**. A device may also implement an **RS232-Interface**. In such a MODBUS system, a Master Device and one or several Slave Devices communicate on a passive serial line. On standard MODBUS system, all the devices are connected (in parallel) on a trunk cable constituted by 3 conductors.

Two of those conductors (the "Two-Wire" configuration) form a balanced twisted pair, on which bi-directional data are transmitted, typically at the **bit rate of 9600 bits per second**.

Each device may be connected (see figure 19): - either directly on the trunk cable, forming a daisy-chain, - either on a passive Tap with a derivation cable, - either on an active Tap with a specific cable.

Screw Terminals, RJ45, or D-shell 9 connectors may be used on devices to connect cables (see the chapter “Mechanical Interfaces”).

3.2 Data Signaling Rates

9600 bps and 19.2 Kbps are required and 19.2 is the required default Other baud rates may optionally be implemented : 1200, 2400, 4800, … 38400 bps, 56 Kbps, 115 Kbps, …

A MODBUS over Serial Line Cable must be shielded. At one end of each cable its shield must be connected to protective ground. If a connector is used at this end, the shell of the connector is connected to the shield of the cable.

An RS485-MODBUS must use a balanced pair (for D0-D1) and a third wire (for the Common). In addition to that a second balanced pair must be used in a 4W-MODBUS system (for RXD0-RXD1).

3.3.4 RS232-MODBUS Definition

Some devices may implement an RS232-Interface between a DCE and a DTE.

Optional RS232-MODBUS Circuits Definition

Signal	For DCE	Required on DCE (1)	Required on DTE (1)	Description
Common	--	X	X	**Signal Common**
CTS	In			Clear to Send
DCD	--			Data Carrier Detected (from DCE to DTE)
DSR	In			Data Set Ready
DTR	Out			Data Terminal Ready
RTS	Out			Request to Send
RXD	In	X	X	**Received Data**
TXD	Out	X	X	**Transmitted Data**

Notes :

- “X” marked signals are required only if an RS232-MODBUS option is implemented.
- Signals are in accordance with EIA/ TIA-232.
- Each TXD must be wired with RXD of the other device ;
- RTS may be wired with CTS of the other device,
- DTR may be wired with DSR of the other device.
- Optional electrical interfaces may be added, for example :
  - **Power Supply :** 5..24 V D.C.
  - **PMC circuit :** See above (In 2W-MODBUS Circuits Definition) the note about this optional circuit.

DCE and DTE

The RS232 spec references two kinds of device, DTE (Data Terminal Equipment) and DCE (Data Computing Equipment).

When RS232 was developed the idea was that there would be 2 kinds of devices, DTE (Data Terminal Equipment) and DCE (Data Communications Equipment). Everything would use DB25 connectors and everyone would always connect a DTE to a DCE using a straight-through cable and everything would be easy.

However, as time went by folks wanted to connect two DTEs (or two DCEs) to each other, and DB9* connectors started being used, so alternative cable wirings were required.

There are no hard and fast rules but in general a DTE will have a male DB25 or possibly a male DB9 connector and a DCE will have a female DB25 or DB9. Other connectors may be used but these are the most common.

A typical DTE is a serial port on a terminal, a Com port on a PC or the serial port on an MSS100.

A typical DCE is the serial port on a modem or on a UDS-10, UDS100 or UDS1100.

The connector for DCE equipment is male for the connector housing and female for the connection pins. Likewise, the DTE connector is a female housing with male connection pins