Introduction to Modbus®

This document is an introduction to Modbus. Although this may not be required, for someone who is familiar with Modbus, it may be helpful to read this before configuring an MTL838C to communicate via Modbus.

What is Modbus?

Modbus is a communication protocol developed by AEG-Modicon, and was devised initially for use with their Programmable Logic Controllers. It has, subsequently, become widely accepted as a communications standard, and many products have now been developed which use this protocol.

The protocol specification is maintained by the originators, AEG Modicon, independently of any professional body or industry association. Consequently, there is no formal process by which a product may be certified to be 'Modbus compatible'. The onus is on the manufacturer of the product to confirm that their products are, and continue to remain, compatible with other Modbus devices.

The protocol defines a message structure and format, and determines how each slave will recognize messages sent to it, and how it should decode the information contained in the message. Standardizing these elements allows Modbus devices from a number of different sources to be interconnected, without the need to write specialized software drivers for each component. This is a significant benefit to the end user and is one of the reasons that Modbus has been so successful and popular.

Modbus Transactions

Modbus controllers communicate using a master-slave technique, in which only one device (the master) can initiate a communication sequence.

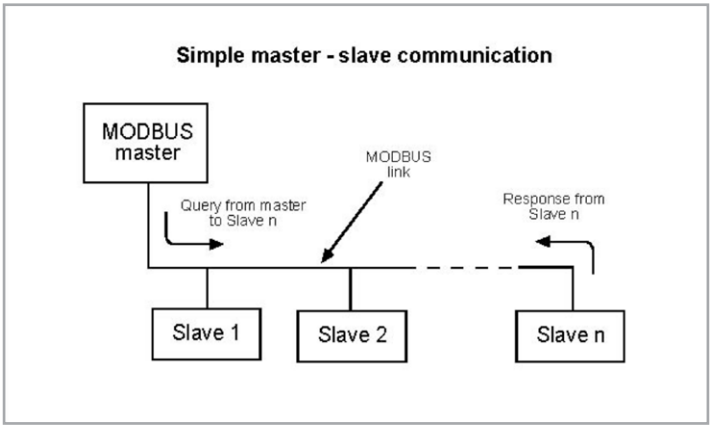

Figure 1 - Master - slave communication

Powering Business Worldwide

Eaton Electric Limited,
Great Marlings, Butterfield, Luton
Beds, LU2 8DL, UK.
Tel: + 44 (0)1582 723633 Fax: + 44 (0)1582 422283
E-mail: mtlenquiry@eaton.com
www.mtl-inst.com

Publication No. TSN838C Modbus Rev 1 140219
February 2019

The sequence begins with the master issuing a request or command on to the bus (a 'query'), which is received by the slaves. The slaves respond by:

a) Taking appropriate action,

b) Supplying requested data to the master, or

c) Informing the master that the required action could not be carried out.

The master can address individual slaves or can transmit a message to be received by all slaves - through a 'broadcast' message (Figure 2).

When a slave receives a message addressed specifically to that slave, it will return a message to the master called a 'response'. The response confirms:

a) that the message was received, understood and acted upon, or

b) informs the master that the action required could not be carried out.

If the 'query' requests data from the slave, this will be returned as part of the response. Messages 'broadcast' to all slaves do not require responses.

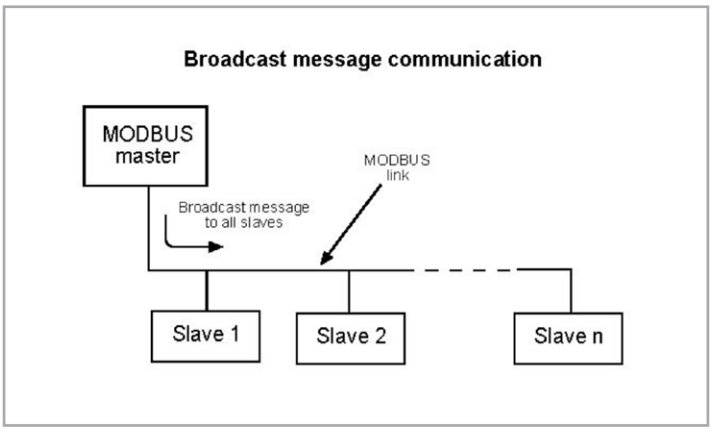

Figure 2 - Broadcast communication

Modbus slaves will only transmit on to the network when required to do so by the master. Slaves never transmit unsolicited messages.

If the slave cannot carry out the requested action, then it will respond with an error message. This error message, known as an exception response, indicates to the master the address of the responding slave, the action it was requested to carry out and an indication of why the action could not be completed.

If an error occurs in receipt of the message, the message will be ignored. This ensures that a slave does not take action that was really intended for another slave and does not carry out actions other than those that it is required to do. Should the message be ignored, the master will know that its query has not been received correctly as it will not receive a response.

Modbus does not define the encoding of numerical data within the message. This is established by the manufacturer - see Data Encoding and Scaling on page 8.

Modbus ports frequently employ RS232C compatible serial interfaces, though RS422 and RS485 interfaces are also used. The type of interface used defines the connector pinouts, the cabling and the signal levels; these are not defined in Modbus. Similarly, transmission rates and parity checking are not defined in Modbus and will depend on the serial interface used and the options made available by the manufacturer of each Modbus component. Converters are available which allow components employing (say) an RS485 interface to be connected to a Modbus controller with a standard RS232 port. See Data converters on page 16 for more information.

Modbus will support up to 247 slaves from addresses 1 to 247 (JBUS 1 to 255) - address 0 is reserved for broadcast messages. In practice, the number of slave addresses that can be used is determined by the communications link that is chosen. For example, RS485 is limited to a total of 31 slaves.

The query-response cycle

The query-response cycle forms the basis of all communication on a Modbus network. In all situations it is the master that initiates the query and the slave that responds.

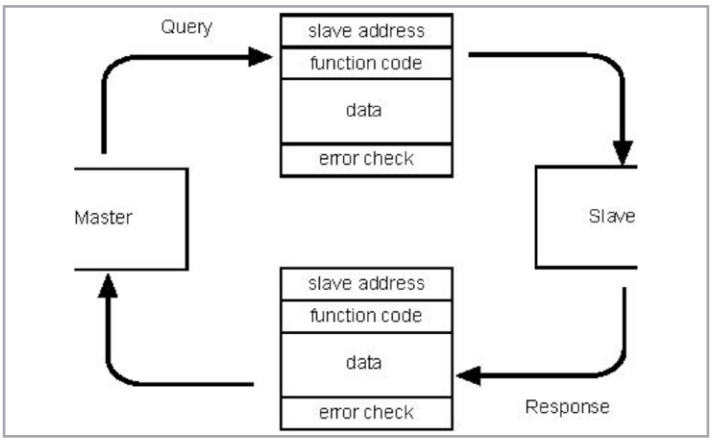

Figure 3 - The query / response cycle

The query

The query is made up of four parts: the device address; the function code; eight bit data bytes; and an error check.

The device address - uniquely identifies a particular slave or indicates that the message is a 'broadcast' addressed to all slaves.

The function code - tells the slave what type of action to perform.

The data - bytes contain any data that the slave will require to carry out the requested function (this may be a register address within the slave, a value to be used by the slave, etc.).

The error check - field allows the slave to confirm the integrity of the message received from the master. If an error is detected, the slave ignores the query and waits for the next query to be addressed to that slave.

The response

A slave will normally be required to provide a response (when a query has been addressed to that slave specifically, and not broadcast to all slaves), which will have the same overall structure format as was used for the query: a device address; a function code; eight bit data bytes; and an error check.

The device address - in the response is that of the addressed slave. This indicates to the master which slave is replying to its query, and allows it to confirm that the correct slave is responding.

The function code - in the response is normally an exact copy of the function code in the query, and will only vary if the slave is unable to carry out the requested function. In such circumstances the function code returned is a modified form of the original code - this then indicates which function the slave was unable to perform.

The data - contain any data requested in the query.

The error check - allows the master to confirm the integrity of the message received from the slave - if the error check is not correct, the response is ignored.

The transmission modes

Two modes are available for transmitting serial data over a Modbus network, but they cannot be used together. The mode that is chosen must be adopted by all the Modbus components throughout the network.

The two transmission modes available are ASCII and RTU, and they differ in a number of ways. The way that the bit contents of the messages are defined, the way that information is packed in to the message fields, the way it will be decoded and the speed of operation at a given baud rate.

ASCII mode – not supported by the MTL838C

When Modbus components are set up to communicate using ASCII (American Standard Code for Information Interchange) mode:

- 8-bit data is encoded in the message as two hex ASCII characters.
- The transmitted characters are printable.
- Each message begins and ends with specified characters.
- Time intervals between characters of up to 1 second are allowed, without causing an error.
- The error checking employed is a Longitudinal Redundancy Check (LRC).

This transmission mode is the slower of the two techniques (as two characters must be transmitted for every 8-bits of data) and it has a less robust error check. The ability to print the transmitted characters and the ability to continue, despite delays in transmission, may only be a significant advantage in a limited number of applications.

The format for each byte in ASCII mode is:

Coding system:	Hexadecimal, ASCII characters: 0-9, A-F Two ASCII characters per 8 bit data
Bits per byte:	1 start bit, 7 data bits (least significant bit sent first) 1 bit for even/odd parity (no bit for no parity) and 1 or 2 stop bits
Error check field:	Longitudinal redundancy check (LRC)

RTU mode

When Modbus components are set up to communicate in RTU (Remote Terminal Unit) mode:

- 8-bit data is encoded in the message as one 8-bit hex character.
- The transmitted messages are not printable.
- The start and end of each message is signaled by 'gaps' in transmission.
- Transmission of data within the message must be continuous.
- The error checking employed is a Cyclical Redundancy Check (CRC).

RTU mode is faster and more robust than ASCII. The format for each byte in RTU mode is:

Coding system:	8-bit binary, comprised of two 4-bit words Each word equivalent to one hexadecimal character
Bits per byte:	1 start bit, 8 data bits, least significant bit sent first, 1 bit for even/odd parity; no bit for no parity and 1 or 2 stop bits
Error check field:	Cyclical Redundancy Check (CRC)

Comparison of RTU and ASCII modes

The table below shows the encoding of the following message in the two modes:

	Message	ASCII	RTU
Start of Frame	-	3A (:)	> 3 chars
Slave Address	01	30 31	01
Function	04	30 34	04
Data	04 01	30 34 30 31	04 01
Error Check	-	46 37	F1 C9
End of frame	-	0D 0A (CR LF)	-

The message above requests that the slave with address '01' reads a holding register (function '04') -and returns the values to the master. Note that although the data in ASCII mode is shown as hexadecimal characters, it would of course be transmitted as a binary stream. (Further, for simplicity, start, stop and parity bits have been omitted.)

A further difference between ASCII and RTU modes is the possibility of using either 7-bit or 8-bit data. As RTU mode requires the use of 8 bit characters, the 7-bit option cannot be used. ASCII only requires 7-bits for each character of the pair and can therefore use this option. It will work equally well with 8-bit communication, but the most significant bit will always be set to '0'.

Modbus message framing

Modbus messages must be structured (or 'framed') so that the different Modbus components can detect the start, content structure, and end point of a message. It also allows any errors to be detected.

The framing used depends on the transmission mode chosen - ASCII or RTU.

ASCII message framing

In ASCII mode, messages start with a 'colon' (:), which in hex is '3A'. The message end is shown by 'carriage return/line feed' (CRLF) or 'OD OA' in hex.

The allowable characters for all other fields are hexadecimal 0-9, A-F. Networked components monitor the bus continuously for the 'colon' character and when one is received, they decode the next field (the address field) to find out if the address is for that slave. If the address is for another slave, then no action is taken, and the slave returns to monitoring for the 'colon' character. If the field following the colon is the address of the slave in question, then the slave continues to read the message and to act on its contents.

Intervals of up to one second can elapse between characters within the message, but if an interval is greater than this, then the device assumes that an error has occurred. If the delay occurs in the 'query' to a slave, then the addressed slave will discard the message received up to that point and wait till the next message (marked by the colon start character) is received.

RTU message framing

In RTU mode, the message begins with a gap in transmission of at least 3.5 character periods. Network components monitor the bus continuously and when a 'silent' period of more than 3.5 character periods is detected, the first character following the transmission gap is translated to determine if it corresponds to the device's own address. The end of the transmitted message is marked by a further interval of at least 3.5 character periods duration. A new message can only begin after this interval.

The entire message field must be transmitted as a continuous stream. If an interval of more than 1.5 character periods is detected during transmission of the message, then the message is assumed to be incomplete and the device returns to waiting for the next device address. The action taken on receipt of an incomplete message is as for receipt of an incorrect message, and it is ignored.

If a new message begins within 3.5 characters periods of the end of the previous frame, the device again ignores the message.

The message fields

The address field

Slave addresses may be in the range 1 to 247 with Modbus (1 to 255 with JBUS). A slave is addressed by the master placing the relevant address in the address field of the query message. When the slave sends its response, it places its own address in the message field to indicate to the master that the correct slave is replying.

Address '0' is used for 'broadcast' messages. All suitable slaves read them, but do not provide responses to such query messages.

The function field

Function codes may be in the range 1 - 255, though not all functions will be supported by all devices. When a message is sent from a master to a slave, the function code defines the action that is required from the addressed slave. Examples of action requested by the various function codes include: read input status; read register content; change a status within the slave; etc.

When the slave sends its response to the master, it will repeat the function code received, to indicate that the slave has understood the query and acted accordingly. If the query instruction could not be carried out by the slave, an 'exception response' is generated and the function code and data fields are used to inform the master of the reason for the exception.

The exception response is generated by returning the original function code from the master, but with its most significant bit set to '1'. Further information regarding the exception response is passed to the master via the data field of the response message. This tells the master what kind of error occurred and allows it to take the most appropriate action - either to repeat the original message, to try and diagnose what has happened to the slave, to set alarms, or to take whatever action is most appropriate.

The data field

The data field transmits a number of hexadecimal values, each in the range 00 to FF. In ASCII transmission mode this is made up of a pair of characters, in RTU it is a single character.

A significant aspect of the communication between the master and its slaves, that is not defined by Modbus, is the encoding of numerical data. Modbus allows the manufacturers of devices to determine which data encoding techniques are available to users of the device. The encoding of data for the MTL838C is in document INM838C.

The data field is used to provide the slave with any additional information needed to perform the function requested in the query. This would typically be a register address, a register range, or a value. With some functions, the data field is not required and will not be included in the query.

If no errors occur, the data field of the response is used by the slave to pass data back to the master.

If an error occurs, the data field is used to pass more information to the master relating to the nature of the fault detected.

Byte count data

The responses to a number of queries require the slave to inform the master of the number of data bytes that are being returned in the response, and this requires a special implementation within the data field.

A typical example of this would be when the master has requested the slave to communicate the status of a range of registers. The slave responds by repeating the function code and its own address, followed by the data field. The first byte of the data field identifies the number of bytes that are being returned that contain the register status information.

As was mentioned earlier, ASCII mode requires two 8-bit bytes to communicate a single register content, compared to RTU which only requires a single 8-bit byte. This difference is ignored when the byte count field is calculated, and the number of bytes indicated is identical to the number of bytes communicated in RTU mode, but is half the actual number of bytes communicated in ASCII mode.

The error check field

The error checking technique employed on the Modbus network depends on the transmission mode selected. With ASCII the technique used is based on an LRC (Longitudinal Redundancy Check) and with RTU a CRC (Cyclical Redundancy Check).

In both cases, the characters transmitted in the error check field are calculated by the transmitting device and included in the resulting transmission. The receiving device calculates what the error check field should contain, on receipt of the message, and compares it with the error check field in the received message. If these two values do not match exactly, then the receiving device knows that it has not received the message correctly, and disregards it.

In both modes, parity checking can be optionally selected.

A fuller explanation of the error checking techniques used by Modbus is given in Appendix A.

Data Encoding and Scaling

As has been mentioned earlier, an important area of the communication along the network, which is not defined by the Modbus protocol, is the encoding of numerical data. A related problem is the adoption of a scaling system for the data once it has been encoded. (Note: this is an area which requires careful consideration by users of the MTL838C.)

There is no problem here for manufacturers who are supplying complete systems, based on the Modbus network, as they can select a data encoding and scaling system appropriate to their needs. However, for manufacturers who are supplying products for general use, there is no possibility that they will be able to determine which data encoding system will be used by their customers, and they must allow the data encoding technique to be user selectable.

Three data encoding techniques are the most popular - IEEE, 16-bit unsigned, and 16¬bit offset.

A further area of difficulty associated with the encoding of data is the way in which the data is scaled - to provide a resolution of the measured value appropriate to the requirements of each application.

Users attempting to configure and scale analog units via the Modbus master and network may find considerable difficulty with this issue. If specifically designed software is available for configuration and scaling of Modbus devices, this may be the simplest and most convenient method of scaling and encoding data. An example of this is the MTL83xC Configurator software, which is available from Eaton for configuring the MTL838C. This makes encoding and scaling decisions transparent to the user.

Modbus concepts and nomenclature

Modbus was originally written as a communication protocol for use with Modicon's own PLCs, and the nomenclature and concepts within Modbus reflect this early intention.

The register concept

The Modbus protocol is based on the concept that Modbus slaves hold their data in a series of defined status flags and registers, with defined addresses. A number of flags and registers are set aside for a number of purposes: single bit input/output; single bit output; multi-bit input/output; and multi-bit output.

Register and flag organization and addressing

The registers and flags within Modbus devices are normally grouped as below:

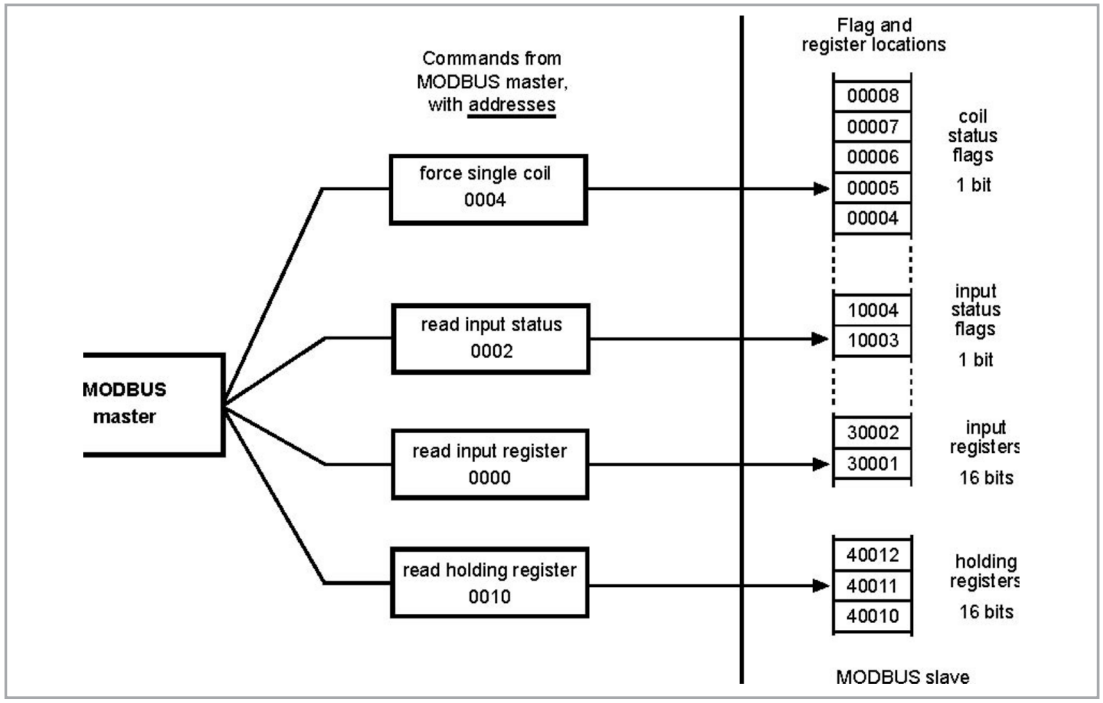

Figure 4 - Addressing Modbus registers

It is not necessary for manufacturers of Modbus devices to follow the register numbering conventions adopted by Modicon's PLCs, but it is normal to do so wherever possible.

COIL STATUS FLAGS

The single bit 'COIL STATUS' flags conventionally have an address of the format 0XXXX, and are used to store digital output information (outputs being from the slave to the field). These flags can be read from and written to by the Modbus master. Though the Modbus slave may not be capable of driving a digital output, these flags may still be used - for example, to request particular functions, such as to reset to original configuration.

NOTE
The term 'coil' comes from the original PLC application, in which the outputs from the PLC were set by controlling the coils of the output relays.

INPUT STATUS FLAGS

The single bit 'INPUT STATUS' flags conventionally have an address of the format 1XXXX. They are used to store the status of digital inputs to the Modbus slave and can only be read by the master. There is no facility, nor any need, for the master to write to these locations.

INPUT REGISTERS

The 16-bit 'INPUT REGISTERS', conventionally, have addresses of the format 3XXXX. They are used to store data which has been collected by the Modbus slave. These registers can only be read by the Modbus master.

HOLDING REGISTERS

The 16-bit 'HOLDING REGISTERS' conventionally have an address of the format 4XXXX. They are used to hold general purpose data within the slave. The master can both read from and write to these registers. A typical application for these registers would be to hold configuration data.

The addressing of flag and register locations within a Modbus slave is simplified by the organization outlined above. Because the Modbus master assumes that data will be stored in a suitable area, the most significant bit of the address is not sent - it is implicitly given by the function code. For example the instruction to 'FORCE COIL STATUS' would not be followed by an address of the format '1XXXX' as the '1' is presumed by way of the function itself.

IMPORTANT NOTE
The locations of a slave's flags and registers begin from X0001, but the addresses given out by the Modbus master begin at X0000.

This means that to force the coil status in flag 10001, the master must send the "force the coil" instruction to address 0000 NOT 0001. To avoid confusion, throughout this manual a distinction is always made between the 'location' and the 'address' of flags and registers.

The addressing of the flags and registers of Modbus slaves may be better understood by reference to Figure 4, where the location number in the slave can be seen to be 'one higher' than the address issued by the master.

When the master wishes to read from or write to a number of flags or registers, Modbus requires that this information is passed to the slave by specifying the address of the first register to be read from, or written to, followed by the number of subsequent registers that must be read. (i.e. To read the values in registers '02' to '04', the master would supply the address for register '02' and ask for the status of three registers.) It follows that if the master wishes to read or write to a number of registers, they must either be sequential - or the master must send more than one query.

Modbus functions and exception responses

Modbus functions

A total of 24 function codes are defined in revision 'E' of the Modbus specification. Modbus slaves will frequently not support all of the functions defined, and the MTL838C is typical of this. The following table summarizes all of the functions that are available in revision 'E'.

CODE	DESCRIPTION
01	Read coil status
02	Read input status
03	Read holding registers
04	Read input registers
05	Force single coil
06	Preset single register
07	Read exception status
08	Diagnostics
09	Program 484
10	Poll 484
11	Fetch comm. Event ctr.
12	Fetch comm. Event log
13	Program controller
14	Poll controller
15	Force multiple coils
16	Preset multiple registers
17	Report slave ID
18	Program 884/M84
19	Reset comm. Link
20	Read general reference
21	Write general reference
22	Mask write 4X register
23	Read/write 4X registers
24	Read FIFO queue

When a slave is unable to carry out a command that has been read correctly (i.e. the received message passes the error and parity checks), the slave will return a response which indicates the function it could not carry out and it will also transmit one of a number of 'exception responses' in the data field.

Modbus exception responses

The exception responses defined in revision 'E' are listed below. Those exception responses supported by the MTL838C are documented in the MTL838C Modbus Implementation Manual (INM838C-MBF).

CODE	DESCRIPTION
01	**ILLEGAL FUNCTION** The function code received in the query is not an allowed action for the slave.
02	**ILLEGAL DATA ADDRESS** The data address received in the query is not an allowable value for the slave.
03	**ILLEGAL DATA VALUE** A value contained in the query data field is not an allowable value for the slave.
04	**SLAVE DEVICE FAILURE** An unrecoverable error occurred while the slave was attempting to perform the requested action.
05	**ACKNOWLEDGE** The slave has received and accepted a request, but it will require a long period of time to complete the task. This prevents a time-out error occurring in the master.
06	**SLAVE DEVICE BUSY** The slave is engaged on a long-duration task. The master should re-transmit the request.
07	**NEGATIVE ACKNOWLEDGE** The slave cannot perform the program function requested.
08	**MEMORY PARITY ERROR** The slave detected a parity error when reading extended memory.

The exception responses issued by a slave may be tailored by the component manufacturer to suit each slave, and events other than those defined in the table above can be used to trigger exception responses. The MTL838C is typical of this and uses the exception response '04' to indicate that its configuration database has become corrupted.

The Modbus physical layer

The serial interface used by most Modbus masters is RS232C. This interface does not allow the Modbus network to extend beyond 10 to 20 meters in length, hence, many manufacturers of Modbus slaves have used other serial interfaces- with longer network capabilities.
The MTL838C uses the RS485 serial interface standard for its Modbus multiplexers. This includes tri-state operation, allows network lengths of up to 1000m and operates with data rates between 300 baud and 19.2 kbaud. RS485 also allows simple parallel connection of a number of units.

NOTE
When a non-RS232 interface is used with an RS232 master, a data converter must be inserted in to the network.

The RS485 serial interface standard

The RS485 standard defines the characteristics of the drivers and receivers that are connected to the bus. It does not define the cabling or connectors used, nor does it specify a particular data rate or signal format.

RS485 employs differential signaling and therefore requires at least two connectors per signal and a 'common' line connected between all devices.

RS485 specifies that the two differential signal lines should be marked 'A' and 'B', but this is not always followed. The MTL838C is marked with '+' and '-', which describes the relative voltages of the signaling lines in their quiescent state. Note: with RS485, no damage will occur if the signaling lines are connected with the wrong polarity - but the system will not operate.

RS485 effectively limits the number of addresses supported to 32 units, which in Modbus corresponds to 31 slaves (with 1 master).

Terminations

RS485 interfaces should ideally be provided with a ‘matched’ termination to prevent reflections and 'ringing' of the signal on the bus cabling. The termination will normally be a simple resistive terminator, with an impedance that matches the characteristic of the bus - this will normally be close to 100Ω. In practice, with low data rates and relatively short networks, it is often unnecessary to terminate the bus.

Biasing

When no communication is taking place, the bus is in an undefined, floating state - so that noise on the bus may be decoded as real characters. Well written software should discard most of these characters, but the system may be further protected by biasing the bus to a known state and thereby prevent reception of ‘false’ characters.

IMPORTANT NOTE
The MTL838C does not feature any facilities for terminating or biasing the network, which must be provided by the user.

2- and 4-wire interconnection

When using RS485, two wiring methods are used - two-wire or four-wire interconnection. The two systems are shown in the diagrams below.

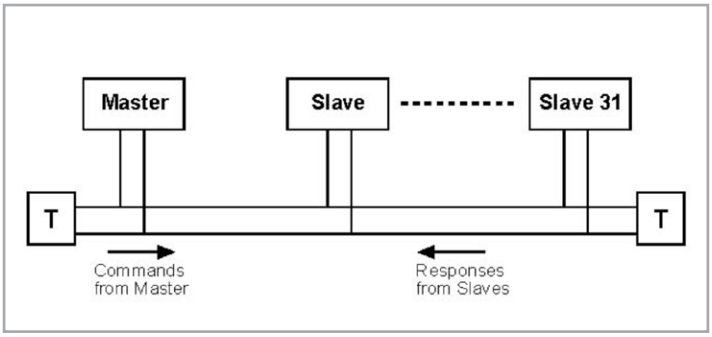

Figure 5 - 2 wire interconnection

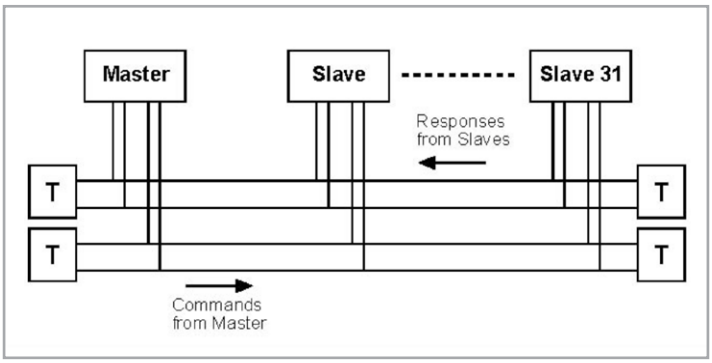

Figure 6 - 4 wire interconnection

In simple terms, two-wire uses the same pair of wires to transmit queries from the master and responses from the slave. With four-wire, queries are sent out on one pair of wires and responses are returned on another set. Note, the nomenclature 'two-' and 'four-wire' ignores the common wire, which is normally used to connect to all devices, but it is not generally shown on interconnection diagrams.

With some protocols, with the adoption of 'four-wire' interconnection, the system can be made to run more quickly, by allowing simultaneous communication of commands and responses. This is not possible with Modbus, as the 'Query-Response' cycle ensures that simultaneous communication by both the master and the addressed slave can never occur. When using Modbus, the decision to use 'two-' or 'four-wire' interconnection is generally made according to the type of serial data interface that is made available on the host. If an RS232 interface is used, then it is simplest to implement 'four-wire' interconnection, if the interface is RS422, then 'two-wire' interconnection is the simplest to implement. As four-wire interfaces have become uncommon, the MTL838C only provides for a two-wire interface.

Point-to-point and multi-drop connection

"Point-to-point" and "multi-drop" are methods of forming the network link between the master and its slave or slaves. Point-to-point describes the connection between a master and a single slave, whereas multi-drop allows a number of slaves to be connected to one master.

Linear BUS and daisy chain connection are multi-drop techniques for connecting a number of slaves to a single master. The two methods are shown below, where the difference between the two methods is apparent.

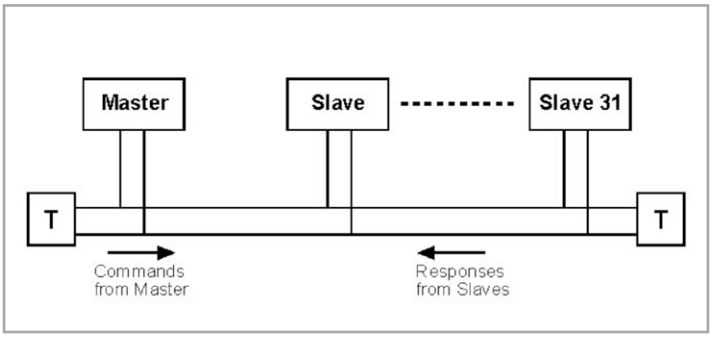

Figure 7 - Linear bus connection

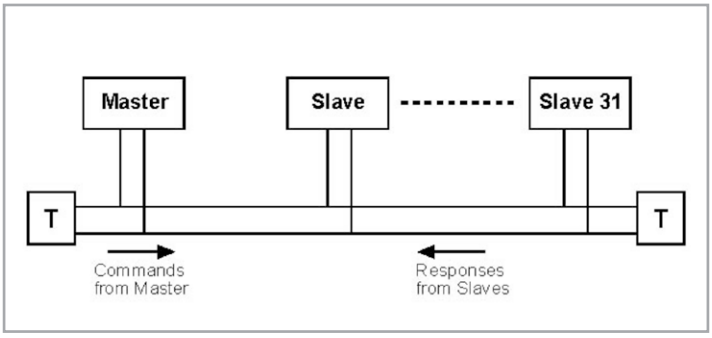

Figure 8 - Daisy-chain connection

Linear BUS connection has two advantages over daisy chain. Firstly, if any particular slave fails, then only that slave is lost, and secondly, slaves can be added to the system at any point along the network. With daisy chain connection, it is possible to lose communication with all slaves connected 'downstream' of any failed device, and it is necessary to gain access to at least one other slave in order to connect a new unit.

With some interfaces, linear BUS connection is not supported, and the only multi-drop connection that is possible is daisy chain. One of the significant advantages of the MTL838C is that it allows multi-drop connections.

NOTE ON RS422:
The only direct connection method that is theoretically allowed when using a Modbus host with an RS422 serial interface is point-to-point. The specification for RS422 indicates that it would be necessary to employ an RS422 to RS485 converter to connect more than one Modbus slave. However, in practice, RS422 is found to perform well when RS485 slaves are linear BUS connected to the RS422 host.

The theory maintains that since RS422 has no tristate facility it cannot be connected to more than one slave. However, since the Tx and Rx lines of the host are always ready to transmit to or receive from a slave, there is no need for the host to adopt a tristate mode. This then allows linear BUS connection on an RS422 host with RS485 slaves.

Data converters

Most Modbus hosts will not provide an RS485 serial interface as standard, with RS232 and RS422 being much more common. Apart from the possibilities of using RS422 as outlined above, in many applications it will therefore be necessary to use a data converter to allow the connection of the Modbus RS485 interface to the host's RS232 or RS422 interface.

Appendix A

ERROR CHECKING TECHNIQUES IN MODBUS

Calculation of the LRC in ASCII transmission mode

The Longitudinal Redundancy Check (LRC) used in the ASCII transmission mode is one byte long and contains an 8-bit binary value. It is calculated by the transmitting device and added to the message. It is also calculated by the receiving device, from the contents of the message, and then compared with the value contained in the message. If the two values are not equal, then an error has occurred in either the transmission or the reception of the message.

The LRC is calculated by adding together the successive 8-bit bytes of the message, discarding any carries and then performing a two's complement operation on the result.

When the sum of the bytes reaches 1111 1111, the addition of any subsequent bytes causes the data to reach a 9-bit value, which cannot be expressed in an 8-bit word. For the purpose of the LRC, the ninth bit is simply discarded so that, for example, the addition of the 8-bit byte 0000 1110, to the value above would give 0000 0111.

The two's complement operation is carried out by subtracting the final sum from 1111 1111, which would give 1111 1000 in our example, and then adding 1. This would give a value of 1111 1001 ('F9' in hex.) to be transmitted as the LRC.

The LRC is actually transmitted as two ASCII characters, equivalent of the LRC value obtained, so for our example, the actual characters transmitted would be '46 39'. The high order character being transmitted first, followed by the low order character.

NOTE
The start and stop characters that are used in ASCII transmission (':' and 'CR/LF') are excluded from the LRC calculation.

Calculation of the CRC in RTU transmission mode

The Cyclical Redundancy Check (CRC) used for RTU transmission mode is 16-bits long and is transmitted as two 8-bit bytes. It is calculated by the transmitting device and added to the message. It is also calculated by the receiving device, from the contents of the message, and then compared with the value contained in the message. If the two values are not equal, then an error has occurred in either the transmission or the reception of the message.

The CRC is calculated by carrying out a process of exclusive OR operations. The first byte is exclusive OR'ed with the value 1111 1111 1111 1111, and the result is used for any subsequent exclusive OR operations that are required. Only the eight bits of the message data are used for this operation. Start and stop bits and the parity bit - if one is selected - are not included in this operation.

Once the first byte has been exclusive OR'ed with the register contents, the least significant bit is removed, and the register contents are shifted towards the least significant bit position, with a zero placed in to the vacant position of the most significant bit.

The least significant bit which is extracted is examined and, depending on its value, one of two operations is carried out. If the extracted LSB is a '1', the register is exclusive OR'ed with the hex value 'A0 01'. If the LSB is a '0', the register contents are left as they are and shifted again. This shifting and extracting process is repeated until eight shifts have occurred.

The second data byte is exclusive OR'ed with the register contents and the process of shifting, examining and exclusive OR'ing is repeated another eight times. This process continues until all the data bytes have been through the operation. The resulting 16-bit CRC is transmitted as two 8-bit bytes, with the high order bytes transmitted first.

Powering Business Worldwide

Eaton Electric Limited,
Great Marlings, Butterfield, Luton
Beds, LU2 8DL, UK.
Tel: + 44 (0)1582 723633 Fax: + 44 (0)1582 422283
E-mail: mtlenquiry@eaton.com
www.mtl-inst.com

Publication No. TSN838C Modbus Rev 1 140219
February 2019

EUROPE (EMEA):
+44 (0)1582 723633
mtlenquiry@eaton.com

THE AMERICAS:
+1 800 835 7075
mtl-us-info@eaton.com

ASIA-PACIFIC:
+65 6645 9864 / 9865
sales.mtlsing@eaton.com