Motion Control Products

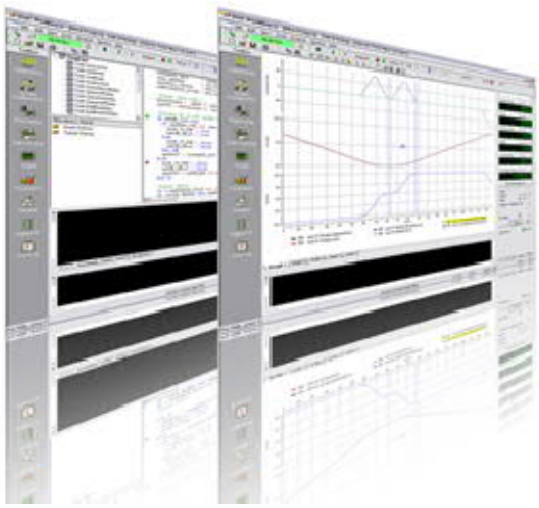
Application note
Mint based Modbus ASCII slave

AN00130

Rev E EN

Modbus is an industry standard protocol that allows a variety of automation devices (such as Programmable Logic Controllers and Human Machine Interfaces) to communicate with each other. The Modbus protocol defines a simple protocol data unit (PDU) that is independent of the underlying communication layers.

Controllers communicate (via RS232/422/485) using a master–slave technique, in which only one device (the master) can initiate transactions (called 'queries'). The other devices (the slaves) respond by supplying the requested data to the master, or by taking the action requested in the query. Typical master devices include host processors and programming panels. Typical slaves include programmable controllers



Introduction

The master can address individual slaves or can initiate a broadcast message to all slaves. Slaves return a message (called a 'response') to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master.

The Modbus protocol establishes the format for the master's query by placing into it the device (or broadcast) address, a function code defining the requested action, any data to be sent, and an error–checking field. The slave's response message is also constructed using the Modbus protocol. It contains fields confirming the action taken, any data to be returned, and an error–checking field. If an error occurred in receipt of the message, or if the slave is unable to perform the requested action, the slave will construct an error message and send it as its response.

A Modbus interpreter has been written in the Mint language. This not only demonstrates the power and flexibility of the Mint programming language, but also allows Modbus functions to be tailored to your application. There are two variants to Modbus, RTU and ASCII. The Mint-based interpreter offers support for the ASCII variant only with the Mint controller operating as a Slave device. The sample Mint program allows a Modbus master to read/write the Mint Comms array, to read digital inputs from the Mint controller and to write to digital outputs on the Mint controller.

Modbus ASCII

Data Format

When controllers are setup to communicate on a Modbus network using ASCII (American Standard Code for Information Interchange) mode, each 8–bit byte in a message is sent as two ASCII characters. The main advantage of this mode is that it allows time intervals of up to one second to occur between characters without causing an error.

The Mint interpreter offers support for master devices with communication settings of either:

- 7 data bits, 1 stop bit, Even Parity
- 7 data bits, 1 stop bit, Odd Parity
- 8 data bits, 1 stop bit, No Parity

Baud rates are limited to those supported by the Mint controller (and of course must match the baud rate setting made on the Modbus master device).

The Mint Startup block defines the controller's serial port setting (the example program illustrates a setting of 19200 baud on terminal channel 1 – e.g. the serial port on a Nextmove ESB controller)…

```
SERIALBAUD(_TERM1) = 19200
```

The Mint task 'ModbusSlave' declares a constant to set the required parity…

```
Const _nParity As Integer   = 0   '0=Even, 1=Odd
```

This task also declares a constant to set the number of data bits being used…

```
Const _nBits As Integer     = 7
```

If this constant is set to 8 then the setting for _nParity becomes irrelevant (i.e. it can be set to either 0 or 1).

Message Framing

In ASCII mode, messages start with a 'colon' (:) character (ASCII 3A hex), and end with a 'carriage return – line feed' (CRLF) pair (ASCII 0D and 0A hex). The allowable characters transmitted for all other fields are hexadecimal 0–9, A–F.

Networked devices monitor the network bus continuously for the 'colon' character. When one is received, each device decodes the next field (the address field) to find out if it is the addressed device.

Intervals of up to one second can elapse between characters within the message. If a greater interval occurs, the receiving device assumes an error has occurred.

A typical message frame is shown below.

START	ADDRESS	FUNCTION	DATA	LRC CHECK	END
1 CHAR	2 CHARS	2 CHARS	n CHARS	2 CHARS	2 CHARS
:	0 2	0 3	0 0 1 D	D E	<CR> <LF>

Address Field

The address field of an ASCII message frame contains two characters. Valid slave device addresses are in the range of 0 – 247 decimal.

When the slave sends its response, it places its own address in the address field of the response to let the master know which slave is responding.

Address 0 is used for the broadcast address, which all slave devices recognize. When configuring a Mint controller for use as a Modbus Slave it is therefore advisable to avoid setting this controller up as Mint Node 0.

Function Field

The function code field of an ASCII message frame contains two characters. Valid codes (from the master) are in the range of 1 – 127 decimal.

When a message is sent from a master to a slave device the function code field tells the slave what kind of action to perform. Examples are to read the ON/OFF states of a group of discrete inputs; to read the data contents of a group of registers; to read the diagnostic status of the slave or to write to designated registers.

When the slave responds to the master, it uses the function code field to indicate either a normal (error–free) response or that some kind of error occurred (called an exception response). For a normal response, the slave simply echoes the original function code.
For an exception response, the slave returns a code that is equivalent to the original function code with its most–significant bit set to a logic 1.

The Mint Modbus ASCII interpreter supports a subset of the full Modbus function set:-

01 (decimal) – Read outputs

This function reads the ON/OFF status of discrete digital outputs on the slave. Broadcast is not supported (i.e. the function request will be ignored).

The query message from the master specifies the starting input and quantity of outputs to be read from the Mint controller. Outputs are addressed starting at zero (as they are in Mint).

The output status in the response message is packed as one output per bit of the data field. Status is indicated as: 1 = ON; 0 = OFF. The LSB of the first data byte contains the output addressed in the query. The other outputs follow toward the high order end of this byte, and from ‘low order to high order’ in subsequent bytes.

If the returned output quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

02 (decimal) – Read inputs

This function reads the ON/OFF status of discrete digital inputs in the slave. Broadcast is not supported (i.e. the function request will be ignored).

The query message from the master specifies the starting input and quantity of inputs to be read from the Mint controller. Inputs are addressed starting at zero (as they are in Mint).

The input status in the response message is packed as one input per bit of the data field. Status is indicated as: 1 = ON; 0 = OFF. The LSB of the first data byte contains the input addressed in the query. The other inputs follow toward the high order end of this byte, and from ‘low order to high order’ in subsequent bytes.

If the returned input quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

03 (decimal) – Read holding registers
This function reads the binary contents of holding registers (Mint Comms locations in this case) in the slave. Broadcast is not supported.

The query message specifies the starting register and quantity of registers to be read. Registers are addressed starting at zero. However Mint Comms locations are addressed from 1 – 99 (see also Application Note AN00110) so care should be taken not to try and read from register 0 on the Mint controller.

The data is processed as 16–bit values (range +/- 32767).

05 (decimal) – Force single coil

This function forces a single coil (Mint controller digital output) to either ON or OFF. When broadcast by the master (i.e. sent to Node 0), the function forces the same digital output in all attached slaves.

The query message specifies the digital output to be forced. Outputs are addressed starting at zero (as they are in Mint).

The requested ON/OFF state is specified by a constant in the query data field:
- A value of FF00 hex requests the coil to be ON
- A value of 0000 requests it to be OFF

All other values are illegal and will not affect the coil.

The normal response is an echo of the query, returned after the output has been forced.

16 (decimal) – Preset multiple registers

This function presets (writes) values into a sequence of holding registers (Mint Comms locations in this case). When broadcast, the function presets the same register references in all attached slaves.

The query message specifies the register references to be preset. Registers are addressed starting at zero. However Mint Comms locations are addressed from 1 – 99 (see also Application Note AN00110) so care should be taken not to try and write to register 0 on the Mint controller.

The data is processed as 16–bit values (range +/- 32767).

23 (decimal) – Read/Write multiple registers

This function presets (writes) values into a sequence of holding registers (Mint Comms locations in this case) and read back a sequence of holding registers (Mint Comms locations in this case) all in a single transaction (this can be useful where a PLC needs to confirm that the written data has been received/processed). Broadcast is not supported.

The query message specifies the register references to be preset and the register references to be read. Registers are addressed starting at zero. However Mint Comms locations are addressed from 1 – 99 (see also Application Note AN00110) so care should be taken not to try to read or write register 0 on the Mint controller.

The data is processed as 16–bit values (range +/- 32767).

Data Field

The data field is constructed using sets of two hexadecimal digits, in the range of 00 to FF hexadecimal. These are made from a pair of ASCII characters. The data field of messages sent from a master to slave devices contains additional information which the slave must use to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

For example, if the master requests a slave to read a group of holding registers (function code 03), the data field specifies the starting register and how many registers are to be read. If the master writes to a group of registers in the slave (function code 16 decimal), the data field specifies the starting register, how many registers to write, the count of data bytes to follow in the data field, and the data to be written into the registers.

If no error occurs, the data field of a response from a slave to a master contains the data requested. If an error occurs, the field contains an exception code that the master application can use to determine the next action to be taken.

Error Checking Field

When ASCII mode is used for character framing, the error checking field contains two ASCII characters. The error check characters are the result of a Longitudinal Redundancy Check (LRC) calculation that is performed on the message contents, exclusive of the beginning 'colon' and terminating <CR><LF> characters.

The LRC characters are appended to the message as the last field preceding the <CR><LF> characters.

Exception Responses

Except for broadcast messages, when a master device sends a query to a slave device it expects a normal response. One of four possible events can occur from the master's query:

- If the slave device receives the query without a communication error, and can handle the query normally, it returns a normal response.
- If the slave does not receive the query due to a communication error, no response is returned. The master program will eventually process a timeout condition for the query.
- If the slave receives the query, but detects a communication error (parity, LRC), no response is returned. The master program will eventually process a timeout condition for the query.
- If the slave receives the query without a communication error, but cannot handle it (for example, if the request is to read a non–existent register), the slave will return an exception response informing the master of the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

Function Code Field: In a normal response, the slave echoes the function code of the original query in the function code field of the response. All function codes have a most–significant bit (MSB) of 0 (their values are all below 80 hexadecimal).

In an exception response, the slave sets the MSB of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response.

With the function code's MSB set, the master's application program can recognize the exception response and can examine the data field for the exception code.

Data Field: In a normal response, the slave may return data or statistics in the data field (any information that was requested in the query). In an exception response, the slave returns an exception code in the data field. This defines the slave condition that caused the exception.

Only three exception responses are provided by the Mint interpreter program…

```
Const _nIllegalFunction As Integer = 1
Const _nIllegalAddress As Integer  = 2
Const _nIllegalValue As Integer    = 3
```

An illegal function response is generated if the master requests a function other than those supported by the interpreter.

An illegal address response is generated if the master tries to access a Comms location outside of the range 1 – 99, tries to read an input outside of the controller's input range for Bank 0 or tries to write to an output outside of the controller's output range for Bank 0.

An illegal value response is generated if the master attempts to write an illegal value to a digital output (i.e. something other than 0xff00 or 0x0000).

For more detailed information about the Modbus ASCII protocol visit www.modbus.org

Debugging

The variable bDebug has been defined to allow some simple debugging. If this is set to _TRUE, various messages will be displayed via terminal channel 2 (e.g. the USB connection on a NextMove ESB) during the execution of the interpreter. For Mint controllers that only support a single terminal channel (e.g. MintDrive II which only has a single serial interface) the example program should be edited to remove references to terminal channel 2 (#2).

Controller Specifics

The sample code that accompanies this Application Note is specifically targeted at non Ethernet Powerlink controllers. Please refer to comments within the code for minor keyword differences when using a NextMove e100 controller (e.g. BUSNODE(6) replaces NODE).

Timing

Operating at 19200 baud with only the ModbusSlave task running on a Nextmove ESB, timing for functions 01, 02, 03, 05, 16 and 23 (decimal) was found to be as follows:

01 – Read outputs
1 Output : 10ms
16 Outputs : 10ms

02 – Read inputs
1 Input : 10ms
16 Inputs : 10ms

03 – Read register
1 location : 10ms
99 locations : 420ms

05 – Write output
1 Output : 10ms

16 – Write register
1 location : 10ms
99 locations : 240ms

23 – Read/Write registers
1 location : 14ms
99 locations : 500ms

This information may be useful if configuration of timeout settings is possible on the master device.

Contact us

For more information please contact your
local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC