MICROVA electronics & automation

Microva Modbus RTU Implementation

Introduction

Modbus RTU (**R**emote **T**erminal **U**nit) is a master/slave (client/server) type network which uses the RS-485 physical layer. Modbus RTU allows one master at a time on the network and up to 247 slaves. Modbus RTU ("Modbus" for the remainder of this document) data packets contain error check codes which the slave device uses to determine if the received data from the master was corrupted. The slave acknowledges the master request or responds with an error code if the data was corrupted or if the master tries to read or write to slave addresses that do not exist. The slave must respond within the "maximum response time" window as described below. This document includes detailed information on how to connect and use Modbus RTU compatible devices from any manufacturer with Microva devices. For a more general introduction to Modbus networking see the Microva application note "*AN130 What is Modbus?*".

Function Codes and Data Format

Each Modbus data packet begins with the node number that the master wishes to communicate with followed by a function code which describes the type of data in the packet. Table 4 shows the eight most common Modbus function codes (shown in hexadecimal) which are supported by Modbus device manufacturers. These function codes are used to read or write single bits, called "coils" or "discrete inputs", or integer or float data, called "registers". Integer data can be 16 or 32 bits. Float data is 32 bits. Each variable in the slave node is assigned one or two Modbus addresses. One Modbus address is used for 16 bit data. Two addresses are needed for 32 bit data. The address range is 0 to 65,535. The Modbus address is a virtual address which is unrelated to the actual (physical) address of the variable in RAM memory in the slave device. Modbus can also read or write text (character) strings such as the node identification string using these same function codes.

The Modbus standard specifies that each 16 bit data value, i.e. the data for one address, is sent in the data packet in the order of most significant byte followed by least significant byte (known as "big endian" format). However, for 32 bit integer and float data the standard does not specify the order of the two 16 bit values (two addresses) which make up the 32 bit data. The default method for Microva devices is to send both 16 bit and 32 bit data in the full big endian data format, i.e. the first byte sent is the most significant byte followed by the less and less significant bytes. Therefore, for Microva devices, Modbus address n in the packet is the upper 16 bits of the 32 bit data and Modbus address n + 1 is the lower 16 bits. However, the order can be swapped in the Microva BASIC application program in order to read or write to Modbus devices made by other manufacturers which do not follow the big endian data format. See the "Microva BASIC Programming Reference Manual" and the examples at the end of this document for more information.

The Modbus master "request" data packet includes the slave node number, the function code, the first variable address, the number of variables to read or write and the error check code (CRC). The address is incremented for each variable in the request packet. The slave "response" packet includes the data requested or an acknowledgment if the request was for a data write. The response packet also includes an error check code. In this way Modbus ensures reliable communication.

Microva Modbus RTU Hardware

Microva Modbus RTU devices support "two wire" half duplex serial communication using RS-485. The two wire designation refers to the method of driving the line differentially on the A and B data lines which greatly improves noise immunity. Actual RS-485 communication requires at least three wires: A, B and V- and in most cases power is also supplied over the bus so there are four wires: A, B, V- and V+. Microva devices may include one or two Modbus RTU ports. The main Modbus port is port 1 and will be labeled V+, A, B, V-. In Microva BASIC port 1 can be opened in Modbus master or slave modes. When port 2 is included it will be labeled V+, A2, B2, V-. Port 2 can only be opened in Modbus master mode. Note that some manufacturers use different names for the RS-485 wires. Table 1 shows the names used on all Microva devices as well as some alternate names. There is also some confusion about whether signal A is the inverting or non-inverting signal. All Microva devices treat A as non-inverting and B as inverting and will communicate properly when A is connected to A and B to B. For other manufacturers' devices it may be necessary to swap the A and B signals. Swapping the data lines will not damage any device, however, be careful not to swap the power supply wires. Microva devices are designed to be easily "daisy chained" using low cost ribbon cable or Cat 5 networking cable. See Microva application note "*AN136 RS-485 Network Wiring*" for more information.

Table 1: Modbus RTU Signal Names

Name	Description	Other Names
A	non-inverting	DATA+, D+, D0, COM+
B	inverting	DATA-, D-, D1, COM-
V+	power plus	POWER, SUPPLY, DC
V-	power minus	GROUND, COMMON, GND, C

Table 2: Baud Rates

Value	Baud Rate
0	2400
1	4800
2	9600
3	19200
4	38400
5	57600
6	115200
7	250000
8	500000
9	1000000

Table 3: Data Formats

Value	Data Bits	Parity	Stop Bits
0	8	None	1
1	8	None	2
2	8	Even	1
3	8	Odd	1

All nodes on the Modbus network must use the same baud rate and data format and each slave node must have a unique node number (from 1 to 247). Modbus master nodes do not have a node number. Tables 2 and 3 show the baud rates and data formats supported by Microva Devices. The factory set communication values are: node number=1, data format=8/N/1 and baud rate=250K or 1M (depending on the device). The devices' node number, baud rate and data format can be changed using the Microva BASIC Compiler software running on the computer or by using the PGM switch on the PCB for devices which do not have a display. The display and keypad or touch screen can be used to change the communication settings on devices that include a display. See the device data sheet.

Microva Modbus Software

Microva BASIC does not support read or write of individual bits over Modbus since the smallest variable in Microva BASIC is 32 bits. Therefore, a single Microva BASIC variable can represent from 1 to 32 individual bits or a 16 or 32 bit integer value (signed or unsigned) or a 32 bit floating point value. Microva BASIC supports function codes 0x03, 0x04, 0x06 and 0x10 as shown in Table 5. Function codes 0x03 and 0x10 can be used to read or write 16 bit or 32 bit data, i.e. one or two addresses per variable, as shown in the table. Microva BASIC can also read or write text data (strings) as shown in the examples.

Table 5 includes the names for the Modbus functions used in Microva BASIC. For example, the statement below reads three 16 bit variables from port 2, node number 5: MyVarA (addr 22), MyVarB (addr 23) and MyVarC (addr 24). The slave Modbus variable addresses are assigned when the variables are created in Microva BASIC using the "Integer", "Float" or "String" commands.

```
J = Modbus2(RdVars, 5, 22, MyVarA, MyVarB, MyVarC)
```

If MyVarA, B and C were float variables the same command can be used to read the data except the function name would be changed to "RdVars32" and now MyVarA uses addresses 22/23, MyVarB uses 24/25 and MyVarC uses 26/27. As explained above, Microva BASIC treats address 22 as the upper 16 bits of MyVarA and address 23 as the lower 16 bits but this can be changed in the software (see the examples).

Microva Modbus slave devices that do not have a display include a red and green (and often blue) LED in order to assist with setting up the network. In RUN mode (the normal operating mode) the green LED will blink each time a valid Modbus packet, sent to the node, is received. The LED will not blink if the node receives a Modbus packet addressed to another node. When the device is in program mode (PGM mode) the red LED will remain on and the green LED will remain off.

Read or Write Arrays and Strings

Microva BASIC arrays can be read or written over the Modbus by using the array name in the command. For example, if the float array "DX" included four elements then the command to write the entire array to the Modbus slave device might be:

```
J = Modbus2(WrVars32, 5, 10, DX)
```

Which is the same as:

```
J = Modbus2(WrVars32, 5, 10, DX[0], DX[1], DX[2], DX[3])
```

The array size determines the number of data bytes read or written. The RdVars32 command can read a maximum of 62 vars (248 bytes). The WrVars32 command can write a maximum of 61 vars (244 bytes).

The Node Identification String

Strings are stored as ordinary 32 bit integer arrays in Microva BASIC, therefore, strings can be read or written using the string name such as: J = Modbus2(RdVars32, 2, 5, MyString).
Modbus devices made by Microva reserve address 0 for the node identification string. Therefore, the command to read the node ID string from any Microva Modbus device is:

```
Dim  NodeID(48)  As String
J = Modbus2(RdVars32, NodeNum, 0, NodeID)
```

The variable address must equal 0 and the destination string (NodeID in the example but any name can be used) must have a size of 48 bytes. Note that normally a 48 byte string array would require 24 Modbus addresses, however, the Microva BASIC firmware kernel treats Modbus address 0 as a special case reserved for the node ID string. Therefore, Modbus slave devices should assign variable addresses starting with address 1. The format of the returned node ID string is:

```
Firmware Version + "," + Application Program Name
```

For example, a typical node ID string read from a Microva model MB122 device is: `"2.3.1,MB122  1.8"` The length of this example string is 16 bytes: one byte for the string length plus 15 bytes for the character data. That is out of the 48 bytes possible for the node ID string. However, the RdVars32 command reads the entire array of 48 bytes. The remaining 32 bytes in the array are ignored. If the node ID string is longer than 48 bytes it will be truncated to fit (actually 47 bytes because the first byte in the string indicates the string length). The node ID can be read from devices in Program mode or Run mode.

Slave Maximum Response Time

When a request packet is sent from the master to the slave, the master will expect a response. The application program running in the master will be paused while the master waits for the response from the slave. Most often the response from the slave takes just a few milliseconds, however, in some cases the slave is busy or off line (i.e. not in RUN mode) or damaged, etc in which case there is no response. The master will wait for the response at most "max response time". The max response time is specified when the serial port is opened using the Microva BASIC command Open SerialPort. See the "Microva BASIC Programming Reference Manual" for details. If the response is not received from the slave within "max response time", the Modbus master read/write command running in the application program in the master device will return with an error code indicating "no response", and the application program will continue. Therefore, the max response time must be chosen so that it is as short as possible (since the master program is paused while waiting) but long enough to allow the slave time to respond.

In the slave device there is also an application running. Modbus request packets received in the slave are temporarily stored (by way of an interrupt) into a buffer. At the end of every program loop the slave kernel firmware checks the status of the buffer. If a complete request packet has been received the slave will execute the function code command in the request packet and send the response packet. If the master tries to read or write to variable addresses that do not exist in the slave or if the function code is not recognized by the slave, then the slave will return an error "exception" response packet as detailed in the Modbus specification.

In the worst case, the slave will respond to the master request after one entire slave program loop has completed plus the time to execute the request. In most cases the slave program executes in under a millisecond and the time to execute the request takes only a fraction of a millisecond so the response is almost instantaneous. However, if for example the slave program loop took 50 milliseconds to execute, which is possible for devices with a display when the display is being refreshed, then in the worst case the response sent back to the master could take up to 50 milliseconds plus the request execution time. Therefore, the max response time must be long enough so that the master does not "time out", generating an error, during normal master/slave communication. This also illustrates why application programs in the slave devices should be kept as short as possible.

Modbus Command Transaction Time

The Modbus specification requires a minimum of 3.5 characters of "bus idle" time between each request/response data packet. The bus idle time is needed so devices on the network can determine when a packet ends and when a new packet begins. The Microva BASIC kernel firmware provides 4 characters of bus idle time between packets. In Microva BASIC each character is 10 or 11 bits depending on the data format selected in the Open SerialPort command. For example, the 8/N/1 data form = 1 start bit + 8 data bits + 1 stop bit = 10 bits per character. The 8/E/1 data form = 1 start bit + 8 data bits + 1 parity bit + 1 stop bit = 11 bits per character. The time to transmit or receive each character = num bits per character divided by the baud rate. So at 250000 baud rate, a character using 8/N/1 data form would take 40us to transmit or receive (10/250000). This "character time" can be used along with the information in Table 5 to calculate how long the Modbus request and response, called the "transaction time", would take.

For example, from Table 5, the "RdVars" command requires 8 chars in the request packet. If 3 variables were requested the response packet would use 11 chars. But, as mentioned above, there are 4 chars of "bus idle" before and after each packet. Therefore, the total time to request 3 variables using the RdVars command would = 8 + 4 + 11 + 4 = 27 chars. However, the transaction time must also include the time for the slave device to finish the application program loop and to execute the master request since the request packets are not parsed until the end of each application program loop. Therefore, the worst case total "transaction time" for a Modbus command from the master node point of view is:

```
Transaction Time =
   request packet time +
   4 chars bus idle time +
   slave application program loop time +
   time for slave to execute request +
   response packet time +
   4 chars bus idle time
```

In almost all cases the "time for the slave to execute the request" is under 100 microseconds. The "slave loop time" is highly dependent on the application program but for simple slave devices it can also be assumed to be around 100 microseconds. So for the Modbus request example used previously:

```
J = Modbus2 (RdVars, 5, 22, MyVarA, MyVarB, MyVarC)
```

The worst case Modbus command transaction time is:

```
  8 chars  = request packet time
  4 chars  = bus idle time
  100uS    = slave application pgm loop time
  100uS    = time for slave to execute request
  11 chars = response packet time
  4 chars  = bus idle time
-----------------------------------
  27 chars + 0.2ms = Total

At 250000 baud and 8/N/1 the total = 1.28ms
At   9600 baud and 8/E/1 the total = 31.14ms
```

How To Determine The Max Response Time

As mentioned above the maximum response time is the amount of time the master will wait for a response from the slave in the event that the slave is not responding. The max response timer starts immediately after the request packet is sent. Therefore, the max response time equation is similar to the transaction time equation. The equation is:

```
Maximum Response Time =
    slave application program loop time +
    time for slave to execute request +
    response packet time +
    4 chars bus idle time
```

The max response time is set when the serial port is opened in Microva BASIC using the Open SerialPort command. For example, assume the baud rate is 250K and the data format is 8/N/1 so the time per character is 40us. Also assume the slave max program loop time is 2ms and the slave can execute the Modbus request in under 0.5ms (both are good assumptions). If the longest packet is when the master reads ten float variables from the slave. Then using the "RdVars32" function from Table 5:

```
slave app program loop time       = 2.00ms
time to execute Modbus request    = 0.50ms
response packet time (45 chars)   = 1.80ms
bus idle time (4 chars)           = 0.16ms
-----------------------------------------
total response time               = 4.46ms
```

Therefore, for this example, a reasonable max response time value to use in the Open SerialPort command might be 10ms which is around twice the max calculated response time. So the normal response time from the slave will be under 4.46ms, however, if the slave is offline or damaged the response time will max out at 10ms.

Keep in mind when reading any slave node ID string the response packet size is always 53 chars (5 + 4N where N is 12 since the node ID string variable is always 48 bytes). That means the response time to read the node ID, using the numbers above, will be 4.78ms. So the max response time of 10ms will work.

Microva Modbus Program Mode Commands

Microva devices are always in either RUN mode or PGM (program) mode. RUN mode is the normal operating condition. PGM mode is used to write new software into the device or to change the communication settings. The device is online is RUN mode and offline in PGM mode. Microva function code 0x44 shown in Table 5 is used to place all slave devices into PGM mode. Microva function code 0x45 is used in PGM mode to write a new application program into a specific slave device (a specific node). Table 6 shows a summary of the PGM mode commands and the data format for each command. These commands are "user defined" functions as allowed by the Modbus specification. These commands will only be recognized when the device is in program mode, except for the "Read Node ID" command which is the only command which executes properly in normal run mode or in program mode. See the "Microva BASIC Programming Reference Manual" for information on device programming.

Table 4: Commonly Supported Modbus Function Codes

Function Code	Function Name	Description	Request Packet Size	Response Packet Size
0x01	read coils	read 1 to 2000 bits	8	5 or 6+N/8
0x02	read discrete inputs	read 1 to 2000 bits	8	5 or 6+N/8
0x03	read hold registers	read 1 to 125 regs	8	5+2N
0x04	read input registers	read 1 to 125 regs	8	5+2N
0x05	write single coil	write 1 bit	8	8
0x06	write single register	write 1 register	8	8
0x0F	write multiple coils	write 1 to 2000 bits	9 or 10+N/8	8
0x10	write multiple regs	write 1 to 123 regs	9+2N	8

Table 5: Microva Supported Modbus Function Codes

	Function Code	Function Name	Description	Request Packet Size	Response Packet Size
Modbus Function Codes	0x03	RdVars	read 16 bit vars (125 max)	8	5+2N
	0x04	RdInputs	read 16 bit vars (125 max)	8	5+2N
	0x06	WrVar	write a single 16 bit var	8	8
	0x10	WrVars	write 16 bit vars (123 max)	9+2N	8
	0x03	RdVars32	read 32 bit vars (62 max)	8	5+4N
	0x10	WrVars32	write 32 bit vars (61 max)	9+4N	8
Microva Specific Codes	0x44	<none>	begin program mode	8	N/A
	0x45	<none>	program mode commands	see below	see below

N = number of vars read/written

Table 6: Microva Modbus Program Mode Commands

Byte Addr	Modbus Name	Read Node ID	Begin PGM Mode	End PGM Mode	Erase App Pgm	Done Pgming	Ack Response	Chg Node Config	File Sector Data 0-239	File Sector Data 240-479	File Sector Data 480-511
0	Node Number	Node Num	0	0	Node Num	Node Num	Node Num	Node Num	Node Num	Node Num	Node Num
1	Function Code	0x03	0x44	0x45	0x45	0x45	0x45	0x45	0x45	0x45	0x45
2/3	Start Address	0x0000	0x55AA	0x55AA	0x55AA	0x55AA	0x55AA	0x55AA	0x55AA	0x55AA	Sector Num
4/5	Num Vars	24	0x55AA	0x55A1	0x55A2	0x55A3	0x55A4	0x55A5	0x55A6	0x55A7	0x55A8
6-N	data bytes	-	-	-	-	-	-	4	240	240	32
N+1/2	CRC	2	2	2	2	2	2	2	2	2	2
	Packet Size	8	8	8	8	8	8	12	248	248	40

Modbus I/O Examples in Microva BASIC

The following examples show the Microva BASIC code as well as the Modbus packet data read or written and the data stored to the variables in RAM memory. Note that Modbus data is in "big endian" format, however, Microva BASIC stores data to memory in "little endian" data format.

```
--------------------------------------------------------------------------
ex. write a Microva BASIC format string to a Modbus slave device

 Dim MyString(8) As String
 MyString = "Hello"
 J = Modbus2(WrVars32, NodeNum, 6, MyString)

                                  Modbus
  Var Name        RAM Addr    Data  Addr
 -----------   -----------   ----  ------
 MyString[0]   0x8000_0000    5      7
 MyString[0]   0x8000_0001   "H"     7
 MyString[0]   0x8000_0002   "e"     6
 MyString[0]   0x8000_0003   "l"     6

 MyString[1]   0x8000_0004   "l"     9
 MyString[1]   0x8000_0005   "o"     9
 MyString[1]   0x8000_0006    x      8
 MyString[1]   0x8000_0007    x      8

 Modbus Packet Data Sent = "l", "e", "H", 5, x, x, "o", "l"
--------------------------------------------------------------------------
ex. read a null terminated string from a Modbus slave device

 Modbus Packet Data Received = "H", "e", "l", "l", "o", 0, x, x

 Dim NullText(2) As Integer
 Dim MyString(8) As String
 J = Modbus2(RdVars32, NodeNum, 6, NullText)
 NullText[0] = rev(NullText[0])
 NullText[1] = rev(NullText[1])
 MyString = linein(addr(NullText))

  Var Name        RAM Addr    Data
 -----------   -----------   ----
 NullText[0]   0x8000_0000   "H"
 NullText[0]   0x8000_0001   "e"
 NullText[0]   0x8000_0002   "l"
 NullText[0]   0x8000_0003   "l"

 NullText[1]   0x8000_0004   "o"
 NullText[1]   0x8000_0005    0
 NullText[1]   0x8000_0006    x
 NullText[1]   0x8000_0007    x
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
ex. write a null terminated string to a Modbus slave device

 Dim NullText(2) As Integer
 Dim MyString(8) As String
 MyString = "Hello" + chr(0)
 LineOut  MyString, addr(NullText)
 NullText[0] = rev(NullText[0])
 NullText[1] = rev(NullText[1])
 J = Modbus2(WrVars32, NodeNum, 6, NullText)

                                  Modbus
  Var Name        RAM Addr    Data  Addr
 -----------   -----------   ----  ------
 NullText[0]   0x8000_0000   "l"     7
 NullText[0]   0x8000_0001   "l"     7
 NullText[0]   0x8000_0002   "e"     6
 NullText[0]   0x8000_0003   "H"     6

 NullText[1]   0x8000_0004    x      9
 NullText[1]   0x8000_0005    x      9
 NullText[1]   0x8000_0006    0      8
 NullText[1]   0x8000_0007   "o"     8

 Modbus Packet Data Sent = "H", "e", "l", "l", "o", 0, x, x
--------------------------------------------------------------------------
ex. write a 32 bit integer (or float) array to a Modbus slave device

 Dim MyData(2) As Integer
 MyData[0] = 0x01234567
 MyData[1] = 0x89ABCDEF
 J = Modbus2(WrVars32, NodeNum, 6, MyData)

                                  Modbus
 Var Name       RAM Addr    Data  Addr
 ---------   -----------   ----  ------
 MyData[0]   0x8000_0000   0x67    7
 MyData[0]   0x8000_0001   0x45    7
 MyData[0]   0x8000_0002   0x23    6
 MyData[0]   0x8000_0003   0x01    6

 MyData[1]   0x8000_0004   0xEF    9
 MyData[1]   0x8000_0005   0xCD    9
 MyData[1]   0x8000_0006   0xAB    8
 MyData[1]   0x8000_0007   0x89    8

 Modbus Packet Data Sent = 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF
--------------------------------------------------------------------------
ex. write a 16 bit integer array to a Modbus slave device

 Dim MyData(2) As Integer
 MyData[0] = 0x01234567
 MyData[1] = 0x89ABCDEF
 J = Modbus2(WrVars, NodeNum, 6, MyData)

                                  Modbus
 Var Name       RAM Addr    Data  Addr
 ---------   -----------   ----  ------
 MyData[0]   0x8000_0000   0x67    6
 MyData[0]   0x8000_0001   0x45    6
 MyData[0]   0x8000_0002   0x23    -
 MyData[0]   0x8000_0003   0x01    -

 MyData[1]   0x8000_0004   0xEF    7
 MyData[1]   0x8000_0005   0xCD    7
 MyData[1]   0x8000_0006   0xAB    -
 MyData[1]   0x8000_0007   0x89    -

 Modbus Packet Data Sent = 0x45, 0x67, 0xCD, 0xEF
--------------------------------------------------------------------------
```