WRITING TO MODBUS DEVICES

1. Introduction

Senquip devices can read and write to externally connected Modbus devices. Setup of reads is simply achieved using the Modbus settings on the device setup page. Writing to devices is done within a script.

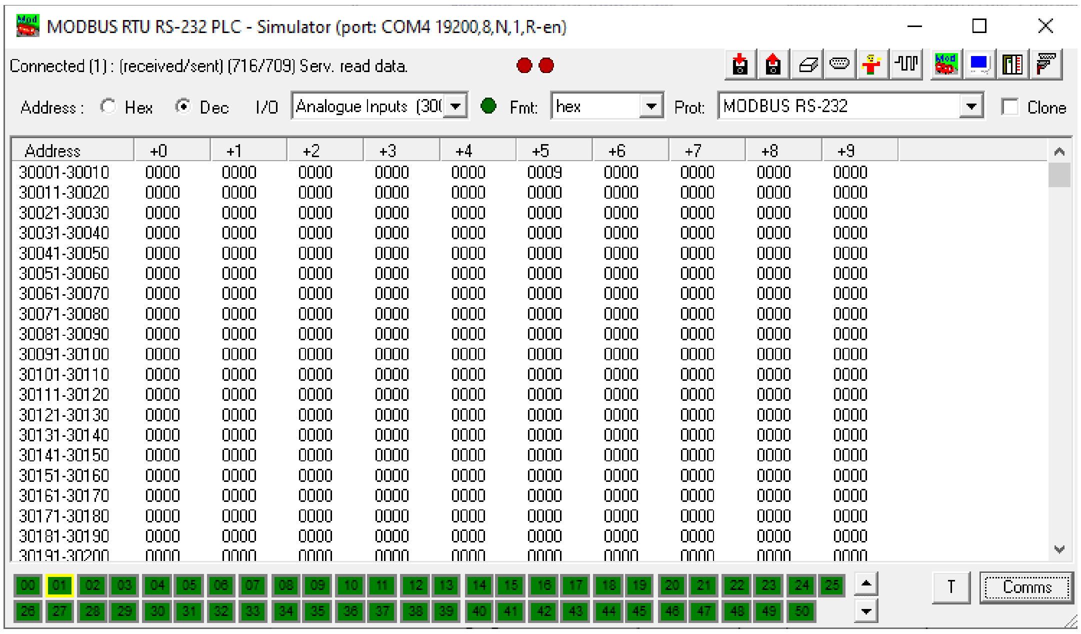
This application note discusses how to perform Modbus writes from within a script. It is assumed that the user has Admin privileges and scripting rights for the device being worked on. To request scripting rights, contact support@senquip.com.

2. References

The following documents were used in compiling this Application Note.

Reference	Document	Document Number
A	Modbus Register Addressing	Modbus Register Addressing, Continental Control Systems
B	Modbus 101 – Introduction to Modbus	Modbus 101 - Introduction to Modbus, Control Solutions, Minnesota
C	Modbus	Modbus, Wikipedia

MOD_RSsim was used as a slave simulator in preparing this application note.



MODBUS RTU RS-232 PLC - Simulator (port: COM4 19200,8,N,1,R-en)

Connected (1) : (received/sent) (716/709) Serv. read data.

Address : Hex [✓ Dec] I/O Analogue Inputs (30(Fmt: hex Prot: MODBUS RS-232 Clone

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
30001-30010	0000	0000	0000	0000	0000	0009	0000	0000	0000	0000
30011-30020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30021-30030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30031-30040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30041-30050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30051-30060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30061-30070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30071-30080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30081-30090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30091-30100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30101-30110	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30111-30120	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30121-30130	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30131-30140	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30141-30150	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30151-30160	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30161-30170	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30171-30180	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30181-30190	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
30191-30200	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 1 - MOD_RSsim Simulation Software

3. Background

Modbus is an industrial protocol standard that was created by Modicon, now Schneider Electric, in the late 1970's for communication among programmable logic controllers (PLCs). Modbus remains the most widely available protocol for connecting industrial devices. The Modbus protocol specification is openly published, and use of the protocol is royalty-free.

The Modbus protocol is defined as a master/slave protocol, meaning a device operating as a master will poll one or more devices operating as slaves. A slave device cannot volunteer information; it must wait to be asked for it. The master will write data to a slave device's registers and read data from a slave device's registers. A register address is always in the context of the slave's registers. Senquip devices are always the master.

The most used form of Modbus protocol is RTU over RS-485. Data is transmitted in 8-bit bytes, one bit at a time, at baud rates ranging from 1200 bits per second (baud) to 115200 bits per second.

A master's query consists of a slave address, a function code defining the requested action, any required data, and an error checking field. A slave's response consists of fields confirming the action taken, any data to be returned, and an error checking field.

Device Address	Function Code	Data	Checksum
1 byte	1 byte	N x bytes	2 bytes

Figure 2 - Modbus RTU Message Frame

The most used function codes are shown in Table 1 .

Function Code	Function	Size	Access
0x01 =01	Read coil	8 bits	Read
0x02 = 02	Read discrete	8 bits	Read only
0x03 = 03	Read unsigned holding	16 bits	Read
0x04 = 04	Read unsigned input	16 bits	Read only
0x05 = 05	Write coil	8 bits	Write
0x06 = 06	Write unsigned holding	16 bits	Write

Table 1 - Function Codes

An example of a Modbus unsigned holding register read from register 5 of a slave device with address 1, and the response, is shown below. Two byte fields are always sent Most Significant Byte (MSB) first and Least Significant Byte (LSB) second.

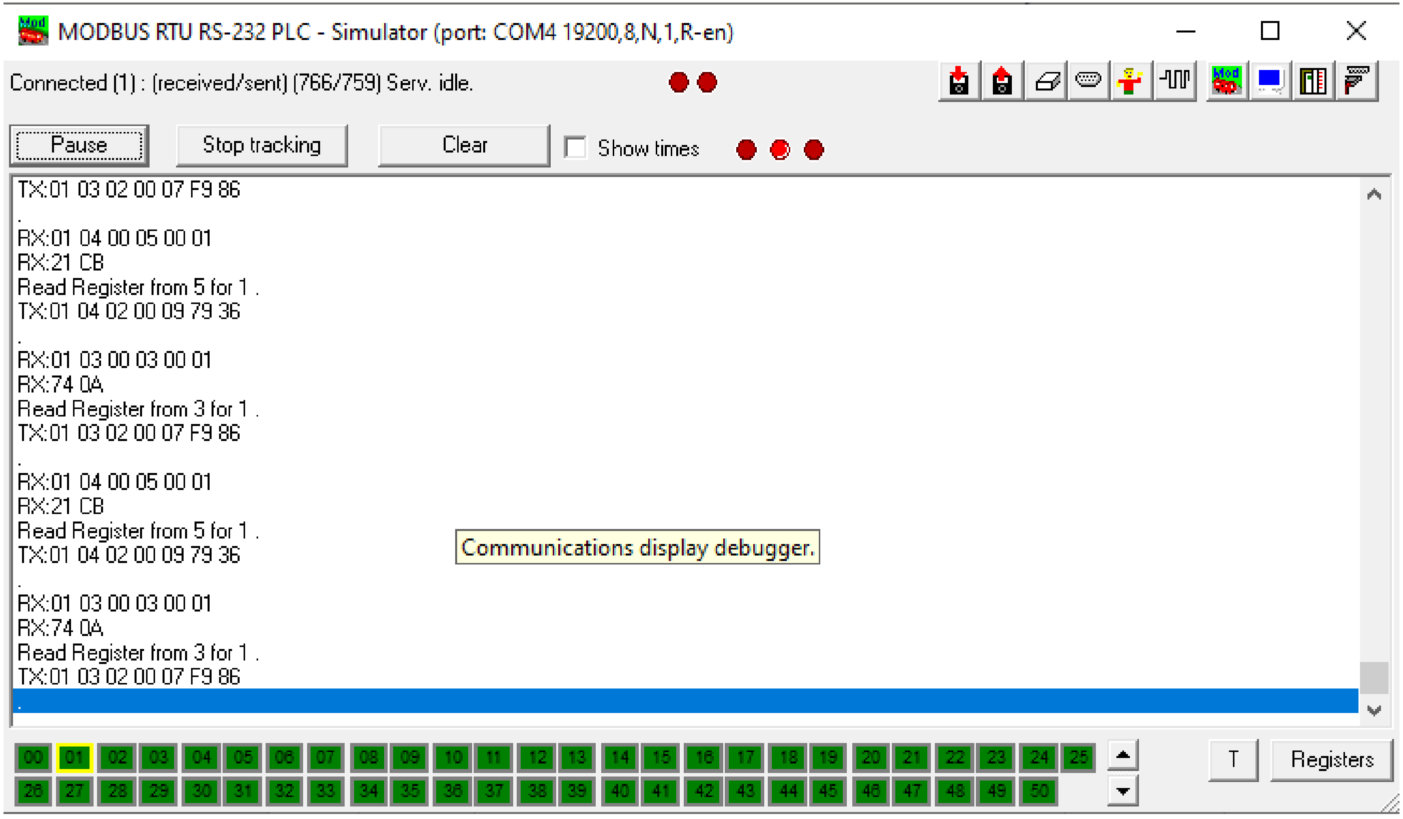
Device Address	Function Code	Data				Checksum	
		Register Address		Number of Registers			
1 byte	1 byte	2 bytes		2 bytes		2 bytes	
0x01	0x04	0x00 (MSB)	0x05 (LSB)	0x00 (MSB)	0x01 (LSB)	0x21 (MSB)	0xCB (LSB)

Figure 3 - Read Request

Device Address	Function Code	Data			Checksum	
		Number of Bytes to Follow	Register Value			
1 byte	1 byte	1 bytes	2 bytes		2 bytes	
0x01	0x04	0x02	0x00 (MSB)	0x09 (LSB)	0x79 (MSB)	0x36 (LSB)

Figure 4 - Response to Read Request

MOD_RSsim includes a window that allows you to see the incoming messages and responses from the simulator.



MODBUS RTU RS-232 PLC - Simulator (port: COM4 19200,8,N,1,R-en)

Connected (1) : (received/sent) (766/759) Serv. idle.

```
TX:01 03 02 00 07 F9 86
.
RX:01 04 00 05 00 01
RX:21 CB
Read Register from 5 for 1 .
TX:01 04 02 00 09 79 36
.
RX:01 03 00 03 00 01
RX:74 0A
Read Register from 3 for 1 .
TX:01 03 02 00 07 F9 86
.
RX:01 04 00 05 00 01
RX:21 CB
Read Register from 5 for 1 .
TX:01 04 02 00 09 79 36
.
RX:01 03 00 03 00 01
RX:74 0A
Read Register from 3 for 1 .
TX:01 03 02 00 07 F9 86
.
```

Figure 5 - MOD_RSsim Message Viewer

Modbus RTU mode includes an error–checking field which is based on a Cyclical Redundancy Check (CRC) 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. Senquip provides a function to calculate the CRC that is documented in the Senquip Scripting Guide.

4. Modbus Specification vs Convention

Modbus register addressing can be confusing because of differences between the Modbus specification and common convention. When working with Modbus, the datasheets need to be read carefully, and ultimately experimentation is required to see what each slave requires for specifying register addresses.

4.1. Function Code Prefix

It is a common convention (but not mentioned in the specifications) to describe a register address with a leading digit to indicate the Modbus function code required to access the register. This results in a format like the following:

XNNNN

Where "X" is one of the following Modbus function codes. For example, if you see a Modbus register number 41221, this is really register number 1221 and should be accessed using the Modbus function 04 "Read unsigned Input".

4.2. Zero vs One Based Numbering

The Modbus specification says that registers are addressed starting at zero. Therefore, input registers numbered 1-16 are addressed as 0-15. Unfortunately, that means a register documented with an address of 1221 must be requested by sending an address of 1220 (04C4 hex).

To make things worse, this is not always the case, sometimes register 1 should be addressed at address 0 and sometimes at address 1.

5. Hardware Setup and Device Configuration

A Senquip ORB-C1 was connected to a computer running MOD_RSsim as a Modbus simulator. In this case, RS232 was used with the serial peripheral being placed in MODBUS mode.

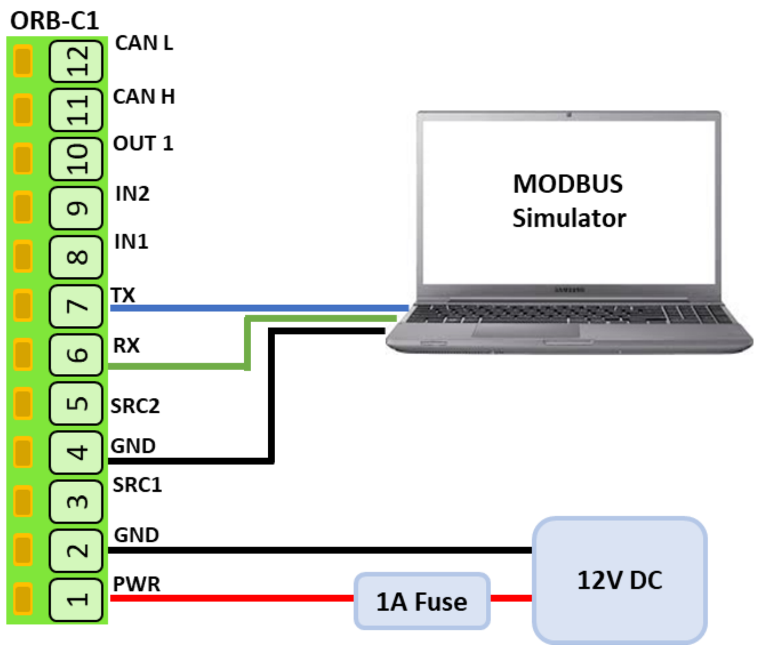
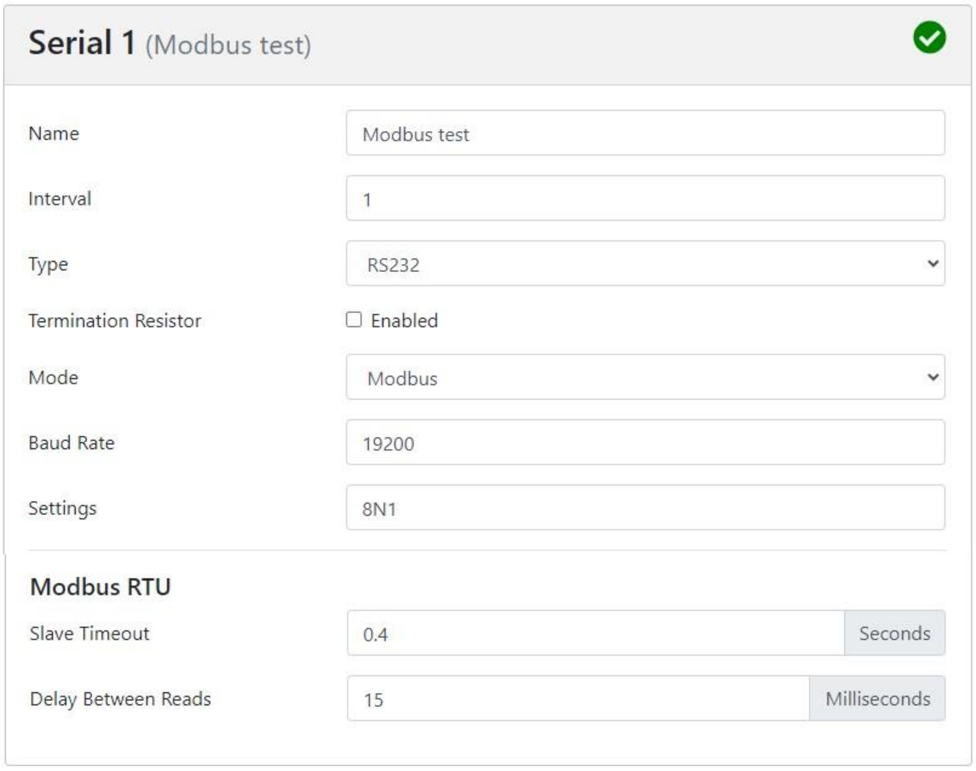


Figure 6 - Hardware Setup

The serial port is configured for RS232 with a baud rate of 19200, 8 data bits, no parity, and 1 stop bit.

Modbus reads are set to timeout if no response is received after 0.5 seconds. A minimum delay of 15 milliseconds is set between a response arriving and the next Modbus read. Although the Modbus standard sets the minimum as 3.5 character periods, we find that many Modbus sensors need a longer time between reads.

A base interval of 10 seconds for the device was selected.



Serial 1 (Modbus test)

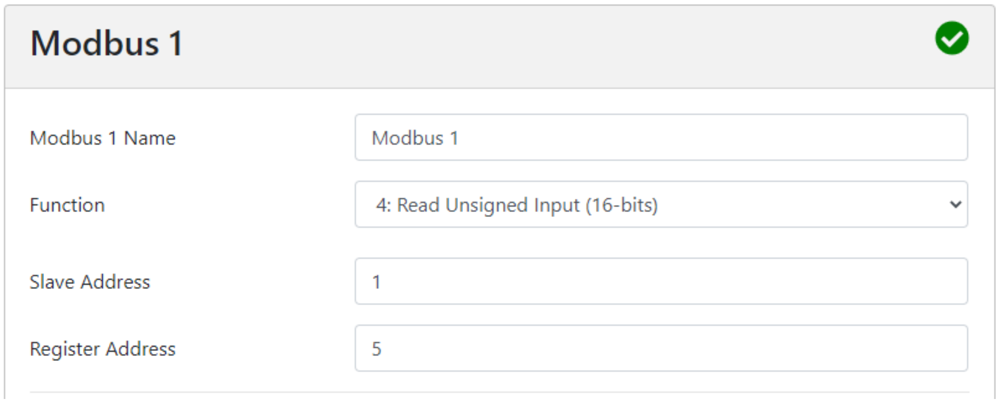
Name	Modbus test
Interval	1
Type	RS232
Termination Resistor	☐ Enabled
Mode	Modbus
Baud Rate	19200
Settings	8N1

Modbus RTU

Slave Timeout	0.4 Seconds
Delay Between Reads	15 Milliseconds

Figure 7 - Serial Peripheral Setup

Modbus 1 was configured to read an unsigned input from slave address 1, register 5.

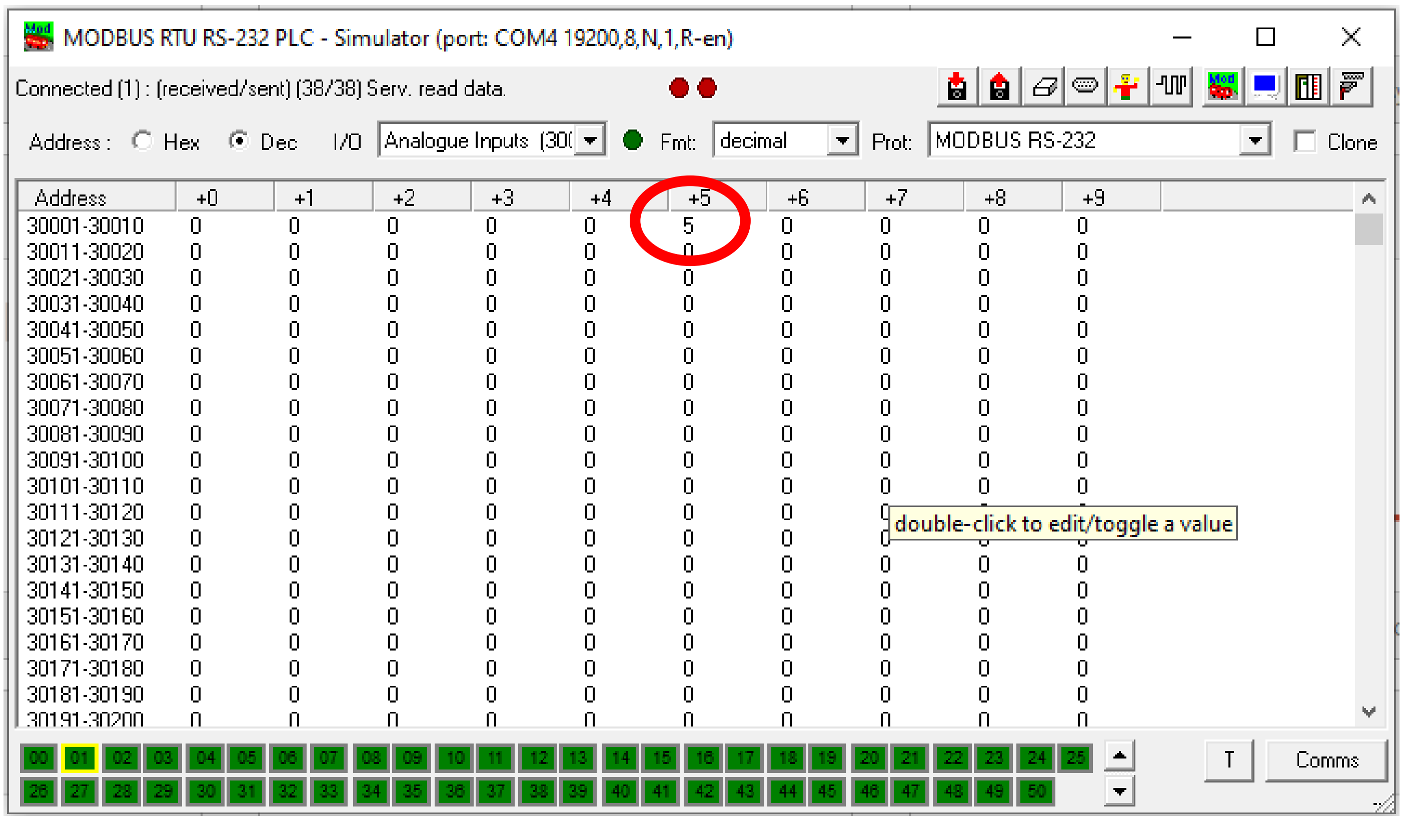


Modbus 1

Modbus 1 Name	Modbus 1
Function	4: Read Unsigned Input (16-bits)
Slave Address	1
Register Address	5

Figure 8 - Modbus 1 Setup

MOD_RSsim is configured to have address 1 and register 5 is given the value 5. Serial port settings are set to match the Senquip serial peripheral.



MODBUS RTU RS-232 PLC - Simulator (port: COM4 19200,8,N,1,R-en)

Connected (1) : (received/sent) (38/38) Serv. read data.

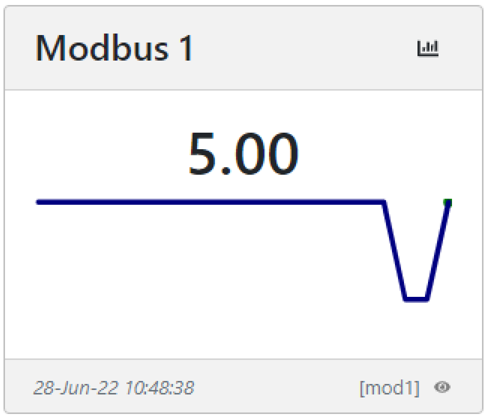
Address : Hex [✓ Dec] I/O: Analogue Inputs (30(Fmt: decimal Prot: MODBUS RS-232 Clone

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
30001-30010	0	0	0	0	0	5	0	0	0	0
30011-30020	0	0	0	0	0	0	0	0	0	0
30021-30030	0	0	0	0	0	0	0	0	0	0
30031-30040	0	0	0	0	0	0	0	0	0	0
30041-30050	0	0	0	0	0	0	0	0	0	0
30051-30060	0	0	0	0	0	0	0	0	0	0
30061-30070	0	0	0	0	0	0	0	0	0	0
30071-30080	0	0	0	0	0	0	0	0	0	0
30081-30090	0	0	0	0	0	0	0	0	0	0
30091-30100	0	0	0	0	0	0	0	0	0	0
30101-30110	0	0	0	0	0	0	0	0	0	0
30111-30120	0	0	0	0	0	0	0	0	[illegible]	[illegible]
30121-30130	0	0	0	0	0	0	0	0	[illegible]	[illegible]
30131-30140	0	0	0	0	0	0	0	0	0	0
30141-30150	0	0	0	0	0	0	0	0	0	0
30151-30160	0	0	0	0	0	0	0	0	0	0
30161-30170	0	0	0	0	0	0	0	0	0	0
30171-30180	0	0	0	0	0	0	0	0	0	0
30181-30190	0	0	0	0	0	0	0	0	0	0
30191-30200	0	0	0	0	0	0	0	0	0	0

double-click to edit/toggle a value

Figure 9 – MOD_RSsim setup

It is confirmed that the Senquip Portal is reading a value of 5 on the Modbus 1 peripheral.



Modbus 1

5.00

28-Jun-22 10:48:38 [mod1]

Figure 10 - Modbus 1 Peripheral on Senquip Portal

6. Writing to a Modbus Peripheral

A script will be used to write to a Modbus register. The script is given in Appendix 1 and will be described below. For more information on scripting for Senquip devices, see the Senquip Scripting Guide.

The standard structure of a Modbus message is given in Figure 2. A Modus message to write to a 16-bit register will have a function code of 6, and will specify the register to be written to, and the data to be written. Specifically, to write the value 55 (37 hex) to register 7, the required message is shown in Figure 11.

Device Address	Function Code	Data				Checksum	
		Register Address		Data to be Written			
1 byte	1 byte	2 bytes		2 bytes		2 bytes	
0x01	0x06	0x00 (MSB)	0x07 (LSB)	0x00 (MSB)	0x37 (LSB)	0x79 (MSB)	0xDD (LSB)

Figure 11 - Writing 55 to Register 7 of Device 1

First, libraries used in the script need to be loaded.

```
load('senquip.js');
load('api_serial.js');

```

A function, sendVal, which writes an unsigned 16-bit number to a holding register is created. SendVal takes as input, an object containing the address and register of the Modbus slave device and the data to be written. It creates a Modbus write message, and initiates a serial write to execute the register update.

```
function sendVal(sendObj){
    let s = SQ.encode(sendObj.sadr,SQ.U8); // encode dec address into hex
    let r = SQ.encode(sendObj.radr,SQ.U16); // encode dec register number into hex
    let v = SQ.encode(sendObj.val,SQ.U16); // encode dec data into hex
    let a = s+'\x06'+r+v;  // 6 is the MODBUS write unsigned 16 function code
    let c = SQ.crc(a);  // use the Senquip CRC function to calculate the Modbus CRC
    c = SQ.encode(c, -SQ.U16); // encode the CRC function in hex + flip byte order
    let t = a+c;  // create the final Modbus write message

    SERIAL.write(1,t,t.length);  // send the message to serial port 1
}
```

The function converts the decimal slave address that has been passed to it, to a string representation of a 1 byte hexadecimal number using the encode function. The encode function convers a number into a string representation using the number type specified. Similarly, the slave register address and data to be sent are encoded into strings representing 2 byte hexadecimal numbers.

The first part of the Modbus write message is assembled by adding the address, function code 6, the register to be written, and the value to write. A checksum for this data is calculated using the Senquip CRC function and is appended as a 16-bit string representation. Note how the byte order in the CRC is swapped by placing a minus in front of the number type in the encode function. The swapping of the bytes is required because the SQ.crc function returns the data LSB first, and the Modbus standard requires the data to be sent MSB first. The CRC is appended to complete the Modbus message.

Finally, the function initiates a serial write to serial port 1.

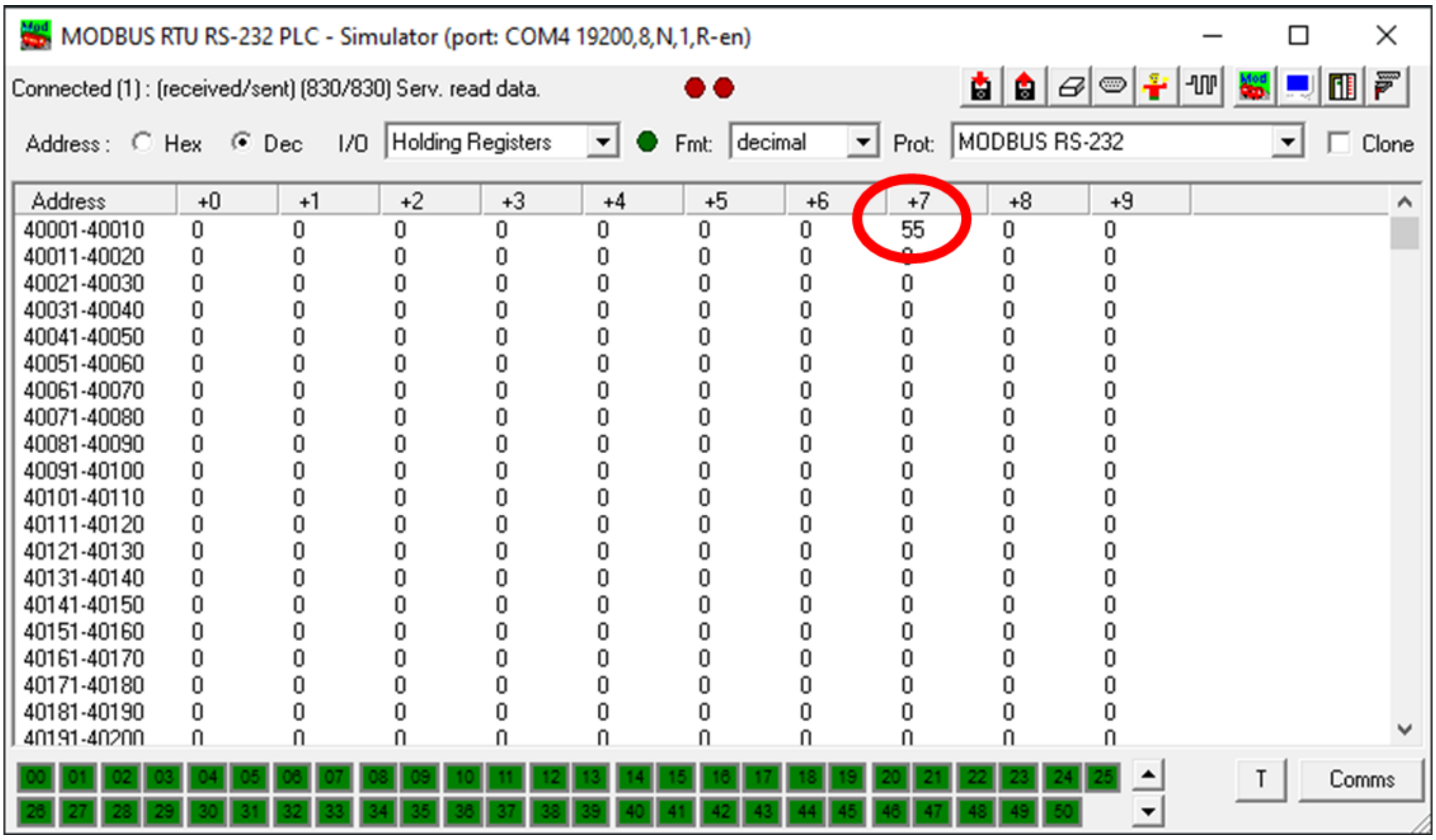
The sendVal function is called from the data handler. The data handler is called on every base interval after measurement collection is complete. In this example, the same write will repeat every time the data handler is called. In a real example, the sendVal would only be called when Modbus data needs to be updated.

```
SQ.set_data_handler(function(data) {

    sendVal({sadr:1, radr:6, val:55});  // call the send Modbus routine

}, null);
```

The completed write is shown in Figure 12.



MODBUS RTU RS-232 PLC - Simulator (port: COM4 19200,8,N,1,R-en)

Connected (1) : (received/sent) (830/830) Serv. read data.

Address : ○ Hex ◉ Dec I/O Holding Registers Fmt: decimal Prot: MODBUS RS-232 ☐ Clone

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
40001-40010	0	0	0	0	0	0	0	55	0	0
40011-40020	0	0	0	0	0	0	0	0	0	0
40021-40030	0	0	0	0	0	0	0	0	0	0
40031-40040	0	0	0	0	0	0	0	0	0	0
40041-40050	0	0	0	0	0	0	0	0	0	0
40051-40060	0	0	0	0	0	0	0	0	0	0
40061-40070	0	0	0	0	0	0	0	0	0	0
40071-40080	0	0	0	0	0	0	0	0	0	0
40081-40090	0	0	0	0	0	0	0	0	0	0
40091-40100	0	0	0	0	0	0	0	0	0	0
40101-40110	0	0	0	0	0	0	0	0	0	0
40111-40120	0	0	0	0	0	0	0	0	0	0
40121-40130	0	0	0	0	0	0	0	0	0	0
40131-40140	0	0	0	0	0	0	0	0	0	0
40141-40150	0	0	0	0	0	0	0	0	0	0
40151-40160	0	0	0	0	0	0	0	0	0	0
40161-40170	0	0	0	0	0	0	0	0	0	0
40171-40180	0	0	0	0	0	0	0	0	0	0
40181-40190	0	0	0	0	0	0	0	0	0	0
40191-40200	0	0	0	0	0	0	0	0	0	0

Figure 12 - Writing 55 to Register 7 of Device 1

The function could be enhanced to facilitate writing to 8-bit and 16-bit registers. Range checking and data validation should be implemented.

7. Conclusion

Reading from and writing to Modbus registers with a Senquip device is simple using available settings and a simple script.

Writing to Modbus registers can be useful in configuring sensors, and in the remote control of machines. In the figure below, the pump is remotely controlled by writing to an engine controller using a Modbus write command.

Figure 13 - Remote control and monitoring of diesel pumps by McGregor Diesel

Appendix 1: Source Code

```
load('senquip.js');
load('api_serial.js');

function sendVal(sendObj){
        let s = SQ.encode(sendObj.sadr,SQ.U8); // encode dec address into hex
        let r = SQ.encode(sendObj.radr,SQ.U16); // encode dec register number into hex
        let v = SQ.encode(sendObj.val,SQ.U16); // encode dec data into hex
        let a = s+'\x06'+r+v;  // 6 is the MODBUS write unsigned 16 function code
        let c = SQ.crc(a);  // use the Senquip CRC function to calculate the Modbus CRC
        c = SQ.encode(c, -SQ.U16); // encode the CRC function in hex + flip byte order
        let t = a+c;  // create the final Modbus write message
        SERIAL.write(1,t,t.length);  // send the message to serial port 1
}


SQ.set_data_handler(function(data) {

        sendVal({sadr:1, radr:6, val:55});  // call the send Modbus routine

}, null);
```