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.

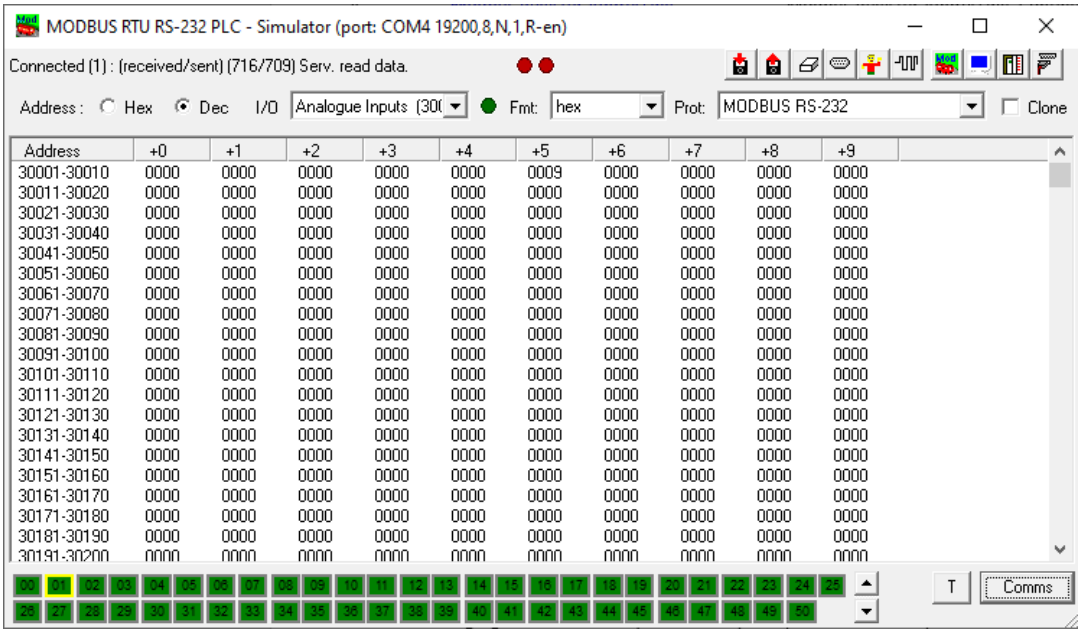

Figure 1 - MOD_RSsim Simulation Software

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

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

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

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.

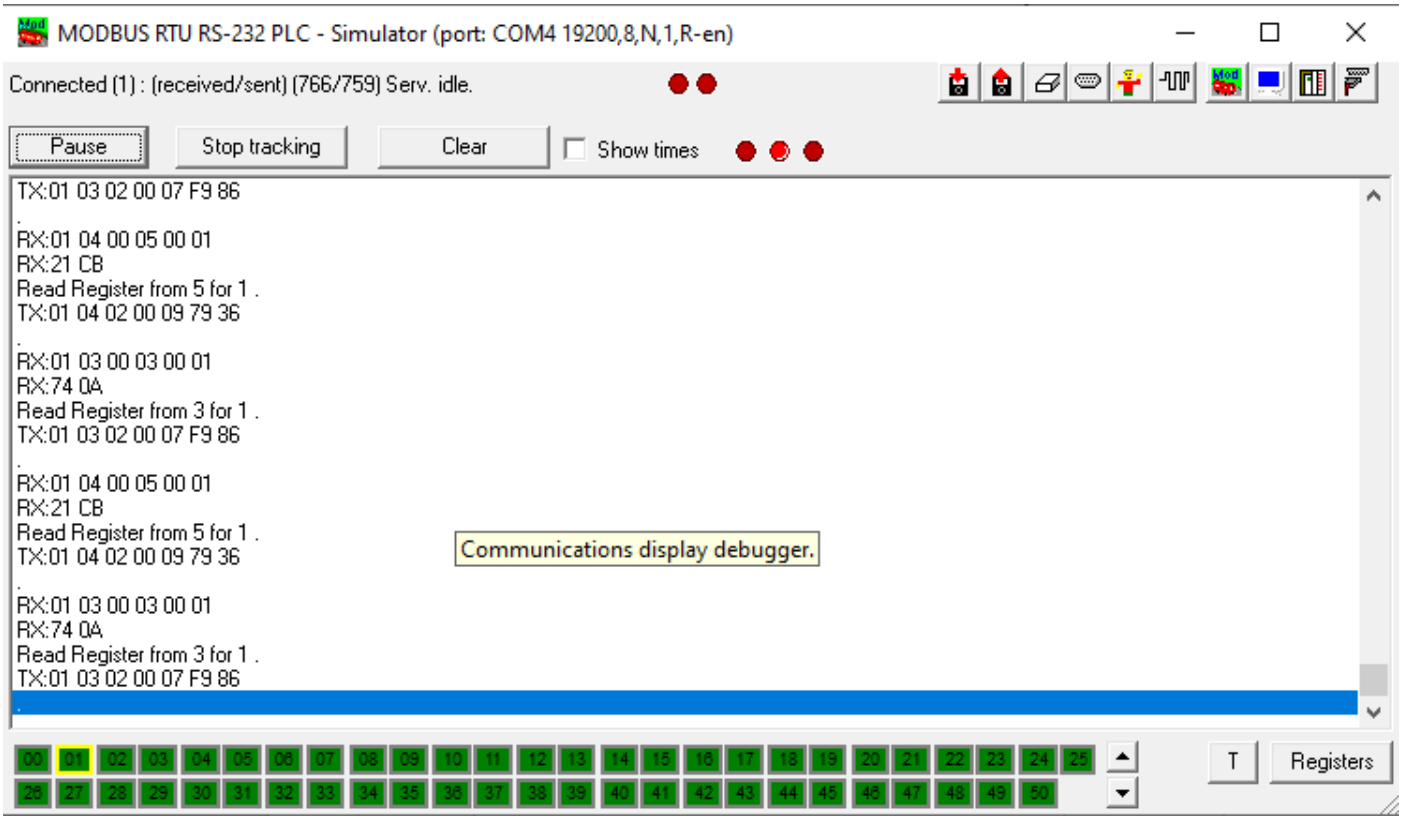

Figure 5 - MOD_RSsim Message Viewer

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.

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

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.

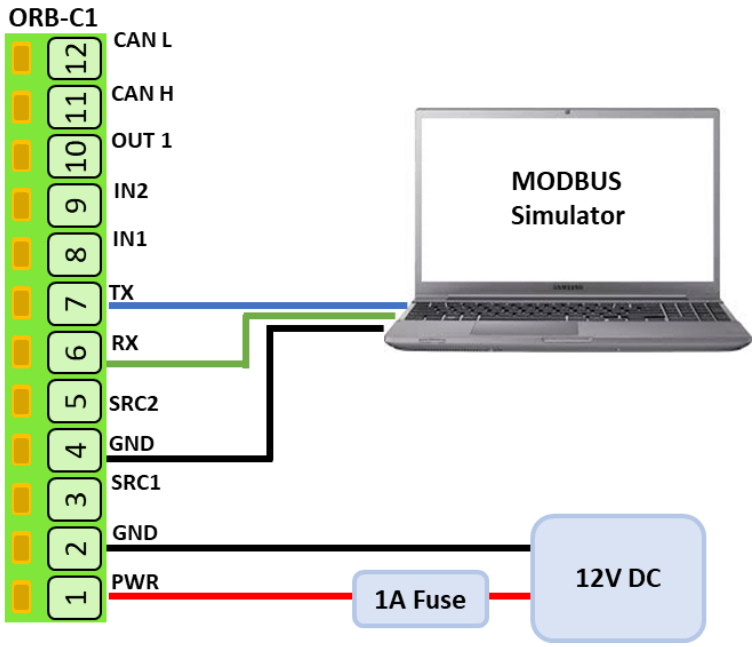

Figure 6 - Hardware Setup

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

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

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.

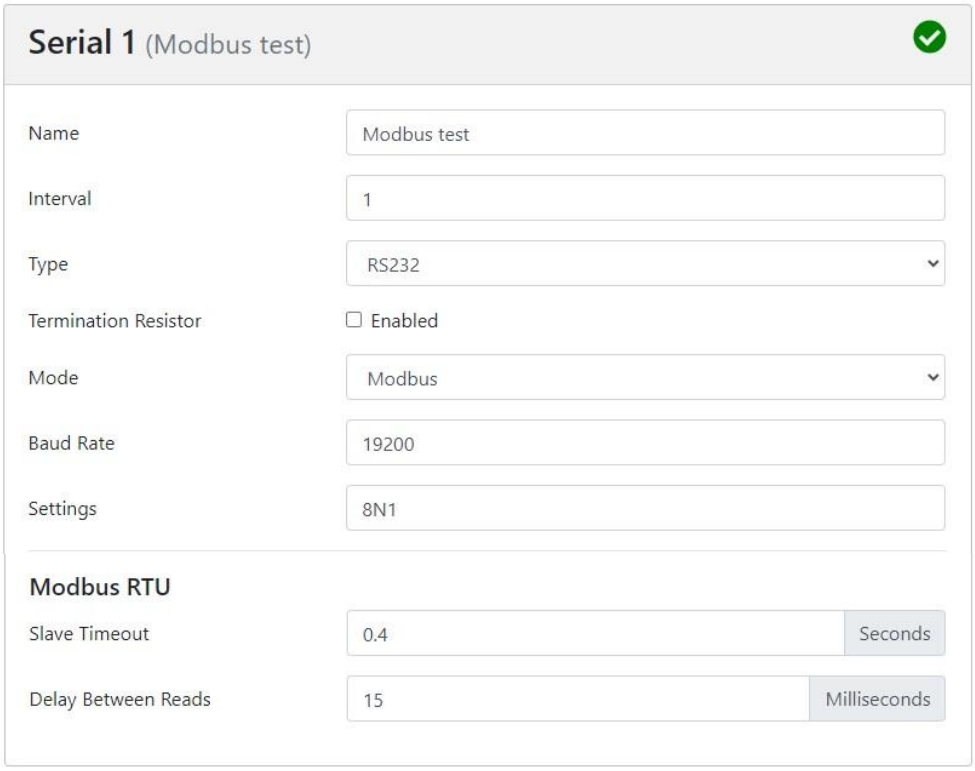

Figure 7 - Serial Peripheral Setup

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

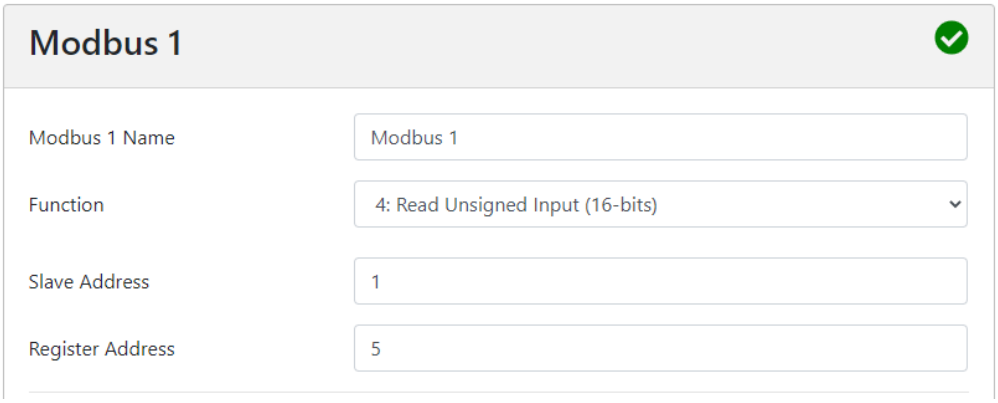

Figure 8 - Modbus 1 Setup

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

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.

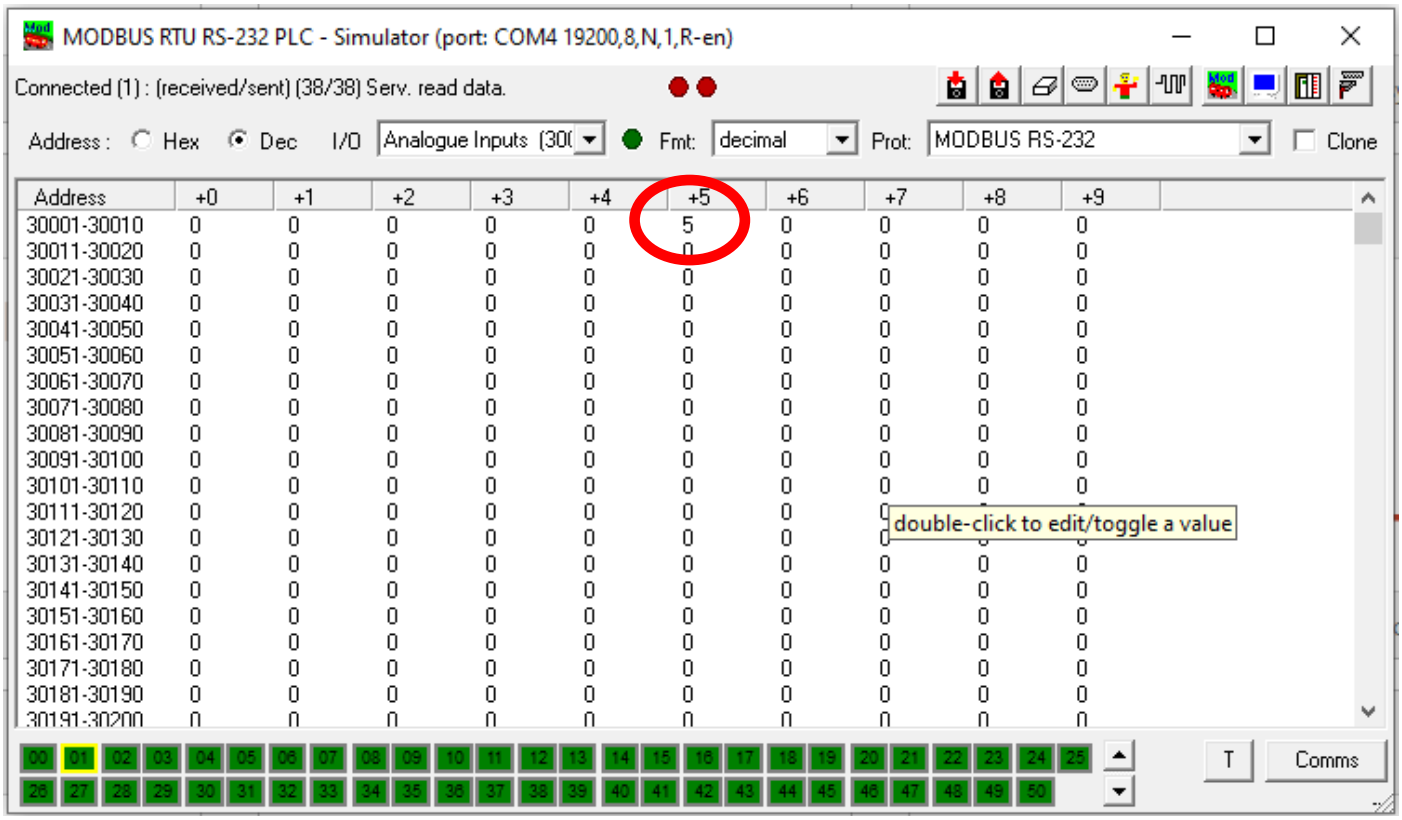

Figure 9 – MOD_RSsim setup

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

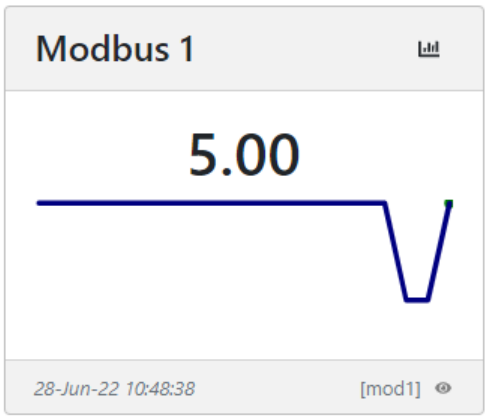

Figure 10 - Modbus 1 Peripheral on Senquip Portal

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

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.

Modbus Message Structure

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.

Field	Size
Device Address	1 byte
Function Code (0x06)	1 byte
Register Address	2 bytes
Data	2 bytes
CRC	2 bytes

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

Example Script

The following script writes a 16-bit value to a Modbus register:

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
}
```

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

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.

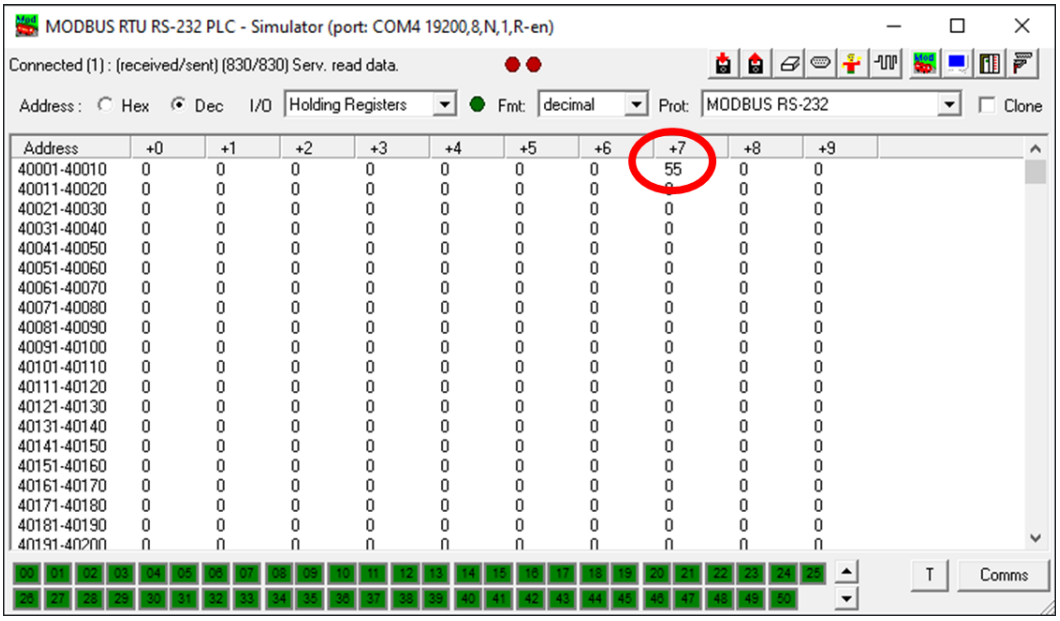

Figure 12 - Writing 55 to Register 7 of Device 1

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

In a real-world implementation, input range checking and data validation should be implemented.

7. Writing a 32-bit Floating Point Value

Many Modbus devices use 32-bit floating point values (IEEE 754) for parameters such as pressure, flow, temperature, or setpoints. These values are stored across two consecutive 16-bit registers.

Unlike writing a single 16-bit register using function code 06, writing a floating point value requires the use of:

- Function Code 16 (0x10) – Write Multiple Registers
- Two registers (4 bytes)

Modbus Message Structure

A Modbus write for a 32-bit floating point value includes:

Field	Size
Device Address	1 byte
Function Code (0x10)	1 byte
Register Address	2 bytes
Number of Registers	2 bytes (value = 2)
Byte Count	1 byte (value = 4)
Data	4 bytes
CRC	2 bytes

Example Script

The following script writes a 32-bit floating point value to a Modbus register:

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

function sendFloat(sendObj){

  let s = SQ.encode(sendObj.sadr, SQ.U8);      // slave address
  let r = SQ.encode(sendObj.radr, SQ.U16);     // register address
  let rc = SQ.encode(2, SQ.U16);               // number of registers (2)
  let bc = SQ.encode(4, SQ.U8);                // byte count (4 bytes)
  let v = SQ.encode(sendObj.val, SQ.FLOAT);    // 32-bit float (IEEE 754)

  let a = s + '\x10' + r + rc + bc + v;        // function 0x10

  let c = SQ.crc(a);                           // CRC calculation
  c = SQ.encode(c, -SQ.U16);                   // swap CRC byte order

  let t = a + c;

  SERIAL.write(1, t, t.length);
}

SQ.set_data_handler(function(data) {

  sendFloat({sadr: 1,radr: 6,val: 55.5});

}, null);
```

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

Byte and Word Ordering

While Modbus specifies that bytes within a register are sent MSB first, the ordering of multiple registers is not standardised. This means that different devices may expect the 32-bit floating point value in different formats.

Common formats include:

Format	Description	Order
Standard	No swapping	ABCD
Word Swap	Swap 16-bit registers	CDAB
Byte Swap	Swap bytes within words	BADC
Word + Byte Swap	Full reversal	DCBA

Where:

- A = Most significant byte
- D = Least significant byte

The encoded floating point string can be rearranged using string slicing before transmission. For example, a word swap can be achieved with v.slice(2,4) + v.slice(0,2). More complex byte reordering can also be handled in the same way by concatenating individual byte slices in the required order.

```
// ABCD = as encoded
let abcd = v;

// CDAB = word swap
let cdab = v.slice(2,4) + v.slice(0,2);

// BADC = byte swap within each word
let badc = v.slice(1,2) + v.slice(0,1) + v.slice(3,4) + v.slice(2,3);

// DCBA = reverse all bytes
let dcba = v.slice(3,4) + v.slice(2,3) + v.slice(1,2) + v.slice(0,1);
```

If the value written to the Modbus device appears incorrect, the word order should be checked against the device documentation. Testing with a known value such as 1.0 (0x3F800000) can help identify ordering issues

8. Conclusion

Writing to Modbus registers from a Senquip device is straightforward and flexible using a simple script. By constructing the Modbus message and using the built-in encoding and CRC functions, reliable communication with external devices can be achieved.

For single register writes, function code **0x06** provides a simple method for updating 16-bit values. For more complex data types, such as floating point values, function code **0x10** allows multiple registers to be written in a single transaction.

In practice, successful Modbus integration depends on careful attention to:

- Register addressing (zero-based vs one-based)
- Data format and size (16-bit vs 32-bit)
- Byte and word ordering for multi-register values
- Device-specific behaviour and timing requirements

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

Modbus is widely used but not always implemented consistently. Testing with known values and validating behaviour against the target device is essential.

Writing to Modbus registers enables remote configuration and control of connected equipment, making it a powerful tool for applications such as sensor setup, parameter adjustment, and machine control.

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

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

Appendix 1: Source Code – 16 bit write

```
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);
```

Appendix 2: Source Code – Floating Point Write

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

function sendFloat(sendObj){

  let s = SQ.encode(sendObj.sadr, SQ.U8);      // slave address
  let r = SQ.encode(sendObj.radr, SQ.U16);     // register address
  let rc = SQ.encode(2, SQ.U16);               // number of registers (2)
  let bc = SQ.encode(4, SQ.U8);                // byte count (4 bytes)
  let v = SQ.encode(sendObj.val, SQ.FLOAT);    // 32-bit float (IEEE 754)

  let a = s + '\x10' + r + rc + bc + v;        // function 0x10

  let c = SQ.crc(a);                           // CRC calculation
  c = SQ.encode(c, -SQ.U16);                   // swap CRC byte order

  let t = a + c;

  SERIAL.write(1, t, t.length);
}

SQ.set_data_handler(function(data) {

  sendFloat({sadr: 1,radr: 6,val: 55.5});

}, null);
```

Copyright © 2026 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.