# 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.

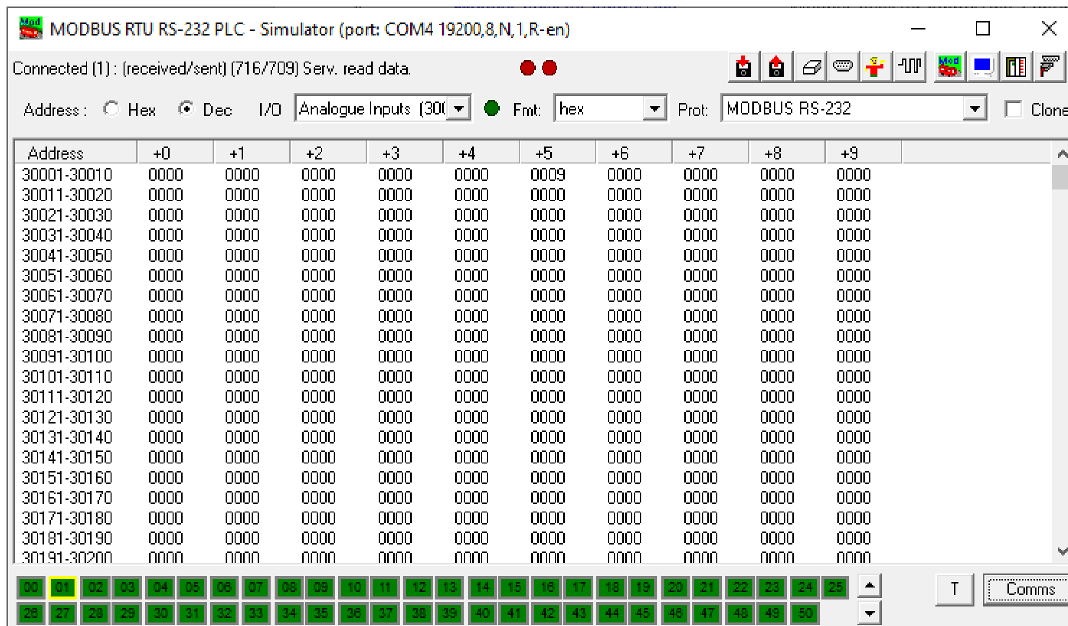


*Figure 1 - MOD_RSsim Simulation Software*


Copyright © 2022 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 holding register read and the response is shown in Figure 3.

| BYTE | REQUEST | BYTE | ANSWER |
|---|---|---|---|
| (Hex) | Field name | (Hex) | Field name |
| 01 | Device address | 01 | Device address |
| 04 | Function code (read unsigned input) | 04 | Function code |

Copyright © 2022 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.

| | | | |
|---|---|---|---|
| 00 | Address of the first register Hi bytes | 02 | Number of bytes more |
| 05 | Address of the first register Lo bytes (Address 5) | 00 | Register value Hi |
| 00 | Number of registers Hi bytes | 09 | Register value Lo |
| 01 | Number of registers Lo bytes (1 register) | 79 | Checksum CRC |
| 21 | Checksum CRC | 36 | Checksum CRC |
| CB | Checksum CRC | | |

*Figure 3 - Modbus Holding Register Read and Response*

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

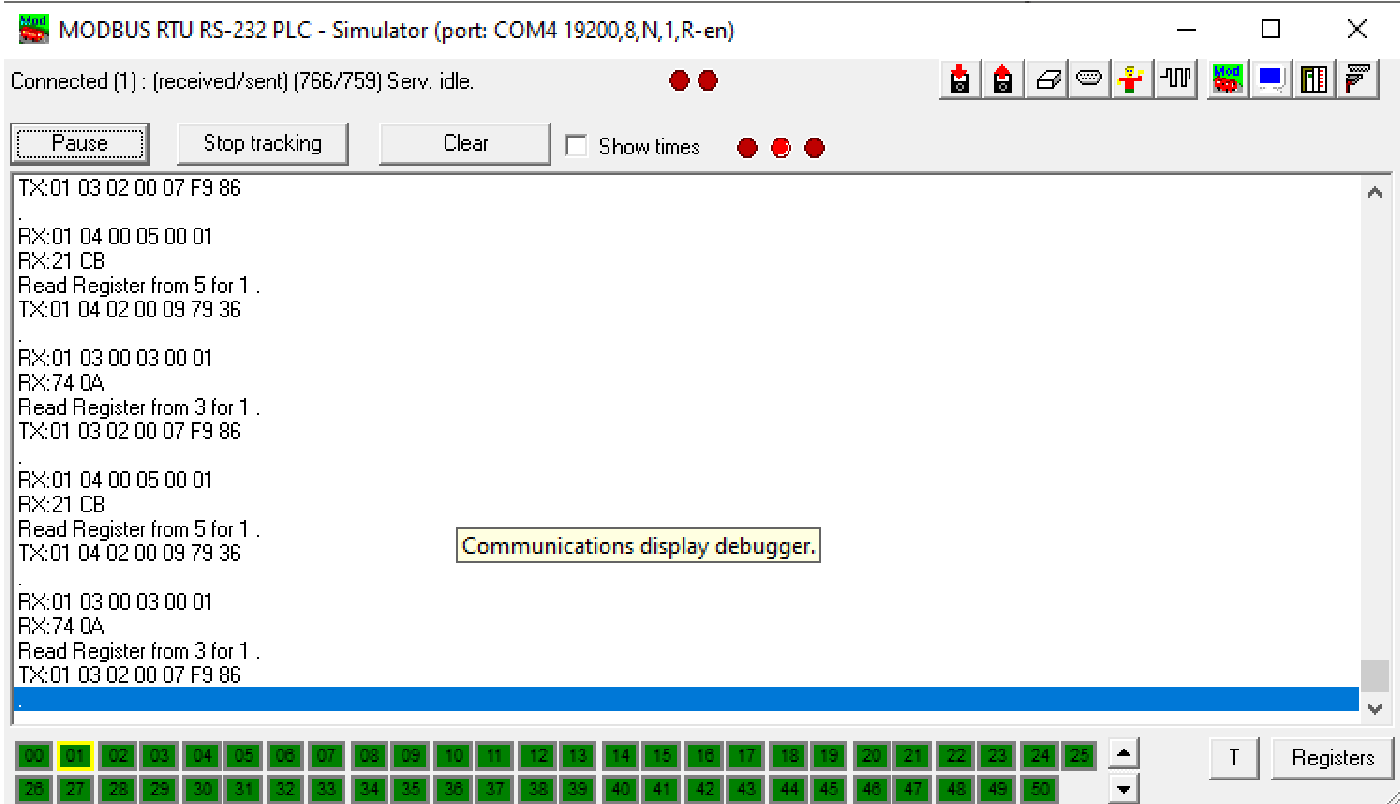


*Figure 4 - MOD_RSsim Message Viewer*

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

Copyright © 2022 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.

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. Modbus 1 was configured to read an unsigned input from slave address 1, register 5. The Modbus peripheral was set to timeout 0.5 seconds after the last Modbus register was requested. A base interval of 10 seconds was selected.

Copyright © 2022 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.

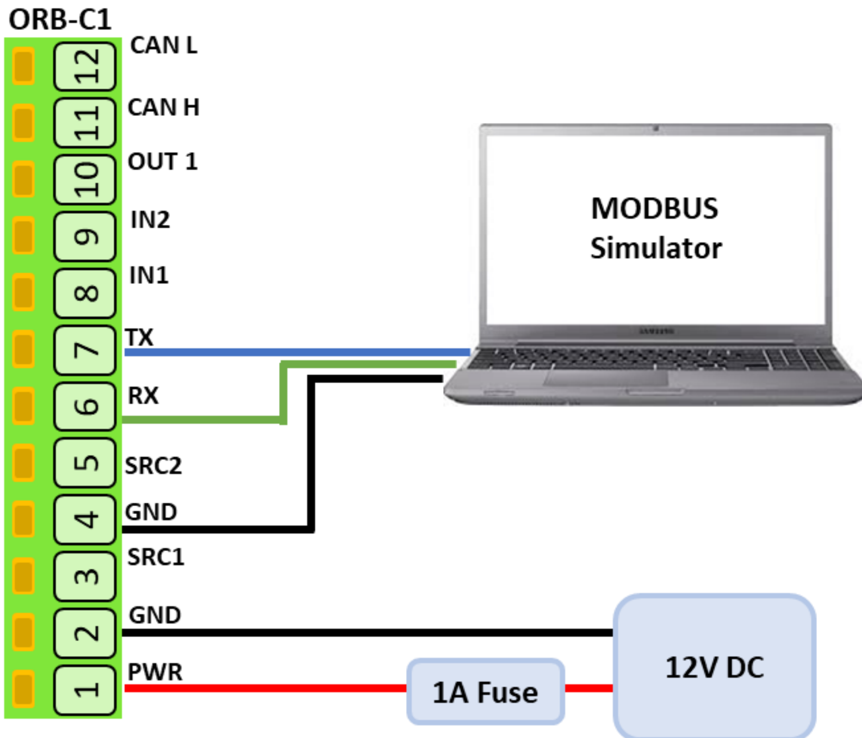


*Figure 5 - Hardware Setup*

**Serial 1** (MODBUS Test)

| | |
| --- | --- |
| Name | MODBUS Test |
| Interval | 1 |
| Type | (●) RS232<br>( ) RS485 |
| Termination Resistor | [ ] Enabled |
| Mode | ( ) Capture<br>(●) Modbus<br>( ) Scripted |
| Baud Rate | 19200 |

**Modbus RTU**

| | |
| --- | --- |
| Slave Timeout | 0.5 Seconds |

*Figure 6 - Serial Peripheral Setup*

Copyright © 2022 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 1**

Modbus 1 Name: Modbus 1

Function: 4: Read Unsigned Input (16-bits)

Slave Address: 1

Register Address: 5

*Figure 7 - 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.

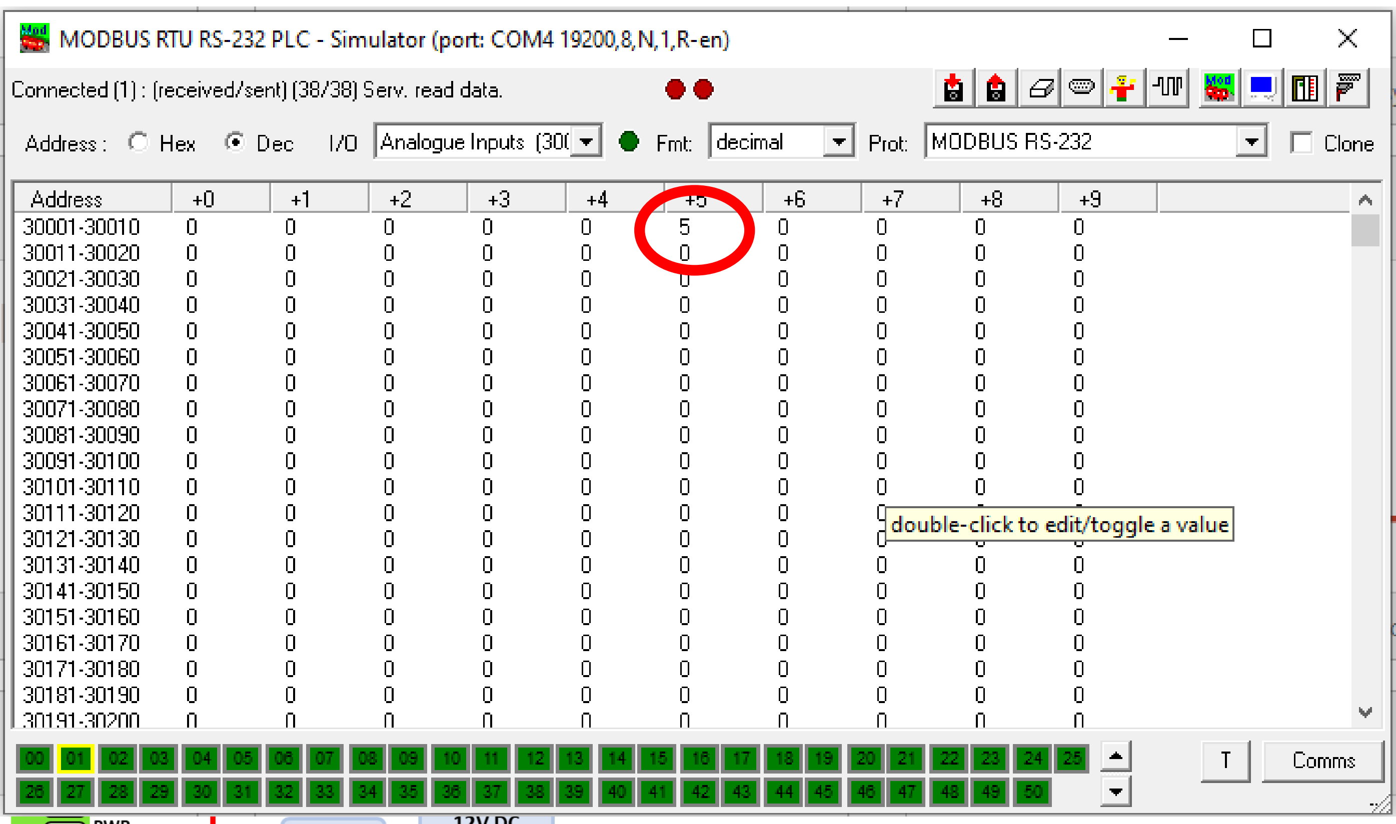


*Figure 8 – MOD_RSsim setup*

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

Copyright © 2022 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.

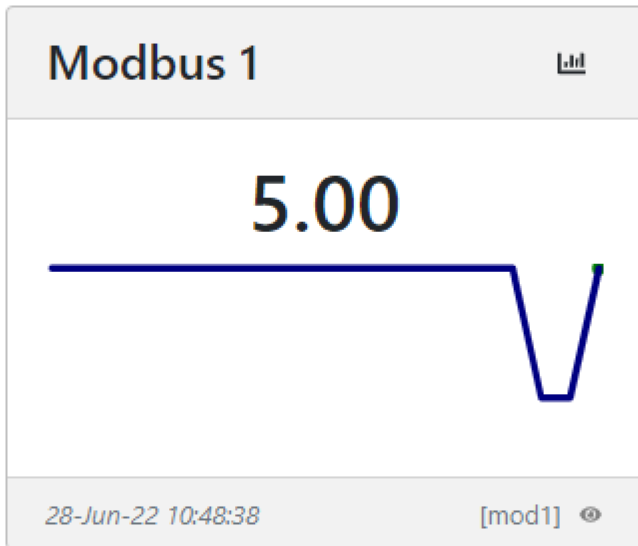


*Figure 9 - Modbus 1 Peripheral on Senquip Portal*

## 7. Writing to a Modbus Peripheral

A JavaScript function will be used to write to a Modbus register. The function is given in Appendix 1 and will be described below.

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 10.

| Device Address | Function Code | Data | | | | Checksum | |
|---|---|---|---|---|---|---|---|
| 1 byte | 6 | Register (2 bytes) | | Data (2 bytes) | | 2 bytes | |
| 01 (hex) | 06 (hex) | 00 (hex) | 07 (hex) | 00 (hex) | 37 (hex) | 79 (hex) | DD (hex) |

*Figure 10 - Writing 55 to Register 7 of Device 1*

A function sendVal which writes an unsigned 16-bit number to a holding register is created. The function is called from the main data handler function that runs after measurement collection is complete. SendVal receives takes as input a JSON object containing the address, register and value to be written as decimal numbers.

The function converts the decimal address to an 8-bit hex number, and the register and value to 16-bit hex numbers, both represented in string format, using the Senquip encode function. A Modbus message is then created by adding the address, function code 6, the register and value. A CRC 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 negative in the encode function. The CRC is appended to complete the Modbus message.

SendVal finally sends the Modbus message to serial port 1 using the serial write function.


Copyright © 2022 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 completed write is shown in Figure 11.

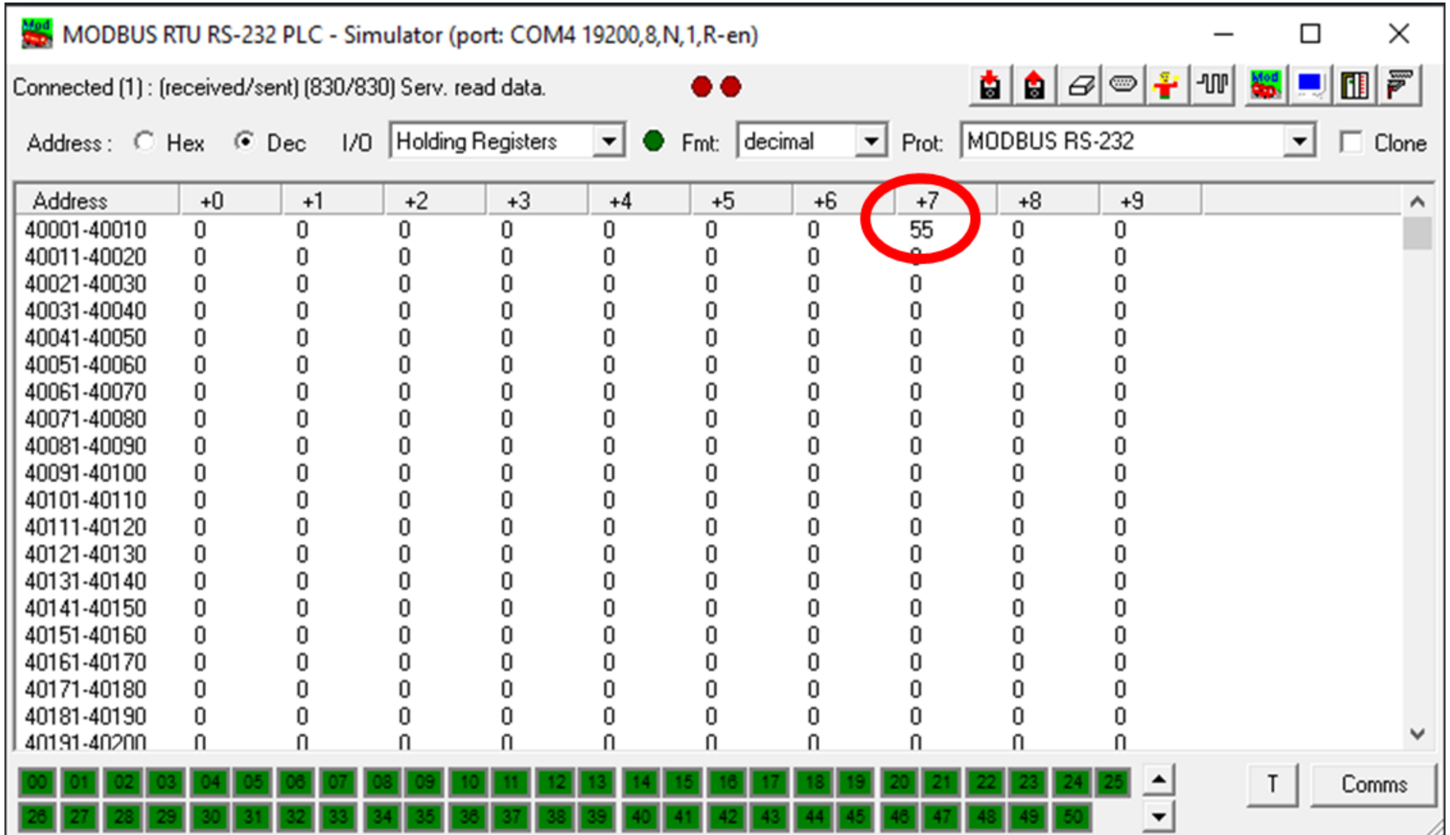


*Figure 11 - Writing 55 to Register 7 of Device 1*

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

## 8. Conclusion

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


Copyright © 2022 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

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

Copyright © 2022 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.