Modbus® Register Addressing and Data Type Variations

Chris Bontje

INTRODUCTION

Much of the early development of the Modbus® protocol occurred in the 1970s and early 1980s and so has become shrouded in time. This white paper describes the early days of programmable logic controllers (PLCs), the introduction of Modbus protocol, how it evolved into an industry standard, and some of the challenges today with device interoperability. The paper is not an in-depth analysis of Modbus protocol structure and physical connections. Rather, because Modbus has been adopted by various manufacturers, the paper describes challenges related to register addressing and data type formatting. For additional details on Modbus protocol structure, refer to the Modbus appendixes in the instruction manuals of Modbus-enabled SEL intelligent electronic devices (IEDs).

MODICON PLC ADDRESSING

When the first digital Modicon® PLCs (*xxx*84 series) were released in the late 1970s, the programming environment provided for these controllers included ladder logic rungs for a user to program the PLC to control their particular process. Use of ladder logic for programming allowed some commonality between traditional relay-based control circuits and the new digital controllers (with accompanying I/O modules) that were rapidly replacing them. What was needed within the ladder logic structure was a way to intelligently address the different signals from the I/O modules on a global basis as well as to provide internal memory for calculation and logic results.

Addressing differences for the basic I/O card types were each assigned a base offset for their particular data type. These are summarized in Table 1.

Table 1 Modicon PLC I/O Card Addressing

Card Type	Address Range	Internal Register Data Type	Data Size
Digital output	1–9999	Coil	1 bit
Digital input	10001–19999	Discrete input	1 bit
Analog input	30001–39999	Input register	16 bits
Analog output	40001–49999	Holding register	16 bits

For example, the first digital input card in a PLC rack may have eight individual inputs assigned to discrete input program addresses 10001–10008 (using what is called a 1X offset). The first digital output card in the rack may have four outputs assigned the coil program addresses 1–4 (a 0X offset). With coil addressing syntax*,* leading zeros are occasionally used, so the alternate addresses are 00001–00004. Internally to the PLC, each digital data type (coil or discrete input) is assigned a single bit of memory and each analog data type (input or holding register) is assigned 16 bits.

As PLC usage and programs grew, so did the memory and address requirements, which eventually exhausted the maximum of 9,999 addressable values per data type. To alleviate this pressure, six-digit addressing was introduced to allow access to additional memory locations, with the upper limit of a fixed 16-bit address space (65,536). In six-digit addressing, the offset references stayed the same for the various data types but an extra digit was added (e.g., holding registers became addresses 400001–465536).

In addition to representing actual I/O signals, the two output address ranges (coils for digitals and holding registers for analogs) were also frequently assigned as temporary "scratchpad" variables for use as logic outputs in ladder logic programming. For example, whereas Holding Registers 40001–40004 can represent an actual analog output card with four 4–20 mA outputs, Holding Registers 40101–40201 can represent user logic outputs that contain calculation results.

MODICON LADDER LOGIC PROGRAMMING METHODOLOGY

When programming ladder logic using these addresses, each line of logic code is arranged into rungs with logical inputs on the left-hand side and outputs on the right-hand side. Figure 1 shows a sample ladder program diagram.

In Rung 1, Digital Inputs 10001 and 10002 are connected in series, creating a logical AND. The output of this statement is written to Digital Output 1.

In Rung 2, Input Register 30001 is multiplied (via the MUL calculation block) by a constant of 30 and the output is written to Holding Register 40101 for later use.

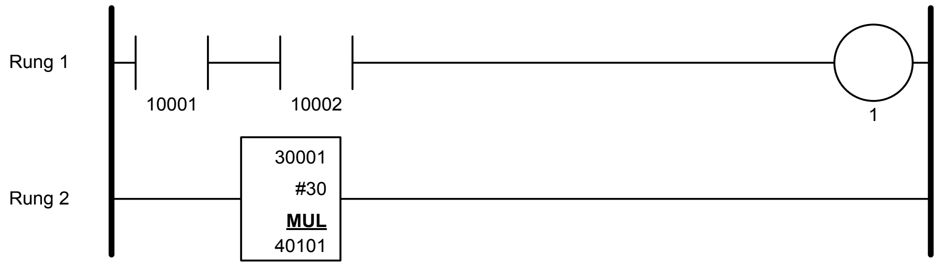


Figure 1 Sample Ladder Logic

MODBUS PROTOCOL

In the early 1980s, the introduction of multiple coexisting PLCs and PC-based hardware to control systems created the need for a standardized interface to connect various pieces of equipment in order to read and write values from and to a given PLC. In response to these changing customer needs, Modicon developed and published an open-standard binary protocol known as Modbus that could enable the exchange of data between two serially connected devices. Modbus was both simple and powerful enough that it soon became widely accepted by many manufacturers as an industry standard to interconnect various pieces of digital industrial hardware. A typical use of Modbus was to use a PC to run a Modbus-enabled human-machine interface (HMI) software package that provided animated mimic screens illustrating the live process that the PLC was controlling.

Modbus was designed as a master/slave (also known as client/server) protocol where a single master in the communications network requested a data type with a specific range of addresses from an addressable slave, and the slave responded with the requested data. The range of addresses in a request message is determined by two values: a base address and a quantity. The data types used in Modbus protocol are designed around the four internal data types previously designated by Modicon for use in their PLCs, and each data type is specified by the master with the use of a unique function code.

The primary function codes used by Modbus protocol include those shown in Table 2.

Table 2 Modbus Protocol Function Codes

Function Code Value (Hexadecimal)	Description	Function Type
0x01	Read coil status	Read
0x02	Read discrete input status	Read
0x03	Read holding registers	Read
0x04	Read input registers	Read
0x05	Force single coil	Write
0x06	Preset single holding register	Write

At the binary protocol level, each of the four different data types uses a series of indexed addresses, starting with a register address of 0. For example, if a Modbus master wanted to read the first eight holding registers from Modbus Slave 1, it issued a request to a specific address using the following parameters:

- Slave address: **1**
- Function code: **0x03**
- Register base address: **0**
- Register quantity: **8**

ADDRESSING CONFUSION

Confusion surrounding Modbus protocol register addressing arises from the fact that different manufacturers address registers using different syntax standards. For example, is the first holding register referred to as 40001 or is it 0? Recall that, as originally designed for use in a Modicon PLC, the first holding register actually was address 40001, but Modbus protocol at a binary level reads that same 16-bit memory location using address 0. Conversion between the two syntaxes is straightforward but does require some numeric manipulation. At its simplest, subtracting 40001 from a Modicon holding register address (for example) generates the raw address used in Modbus. The other data types can be converted using similar methods.

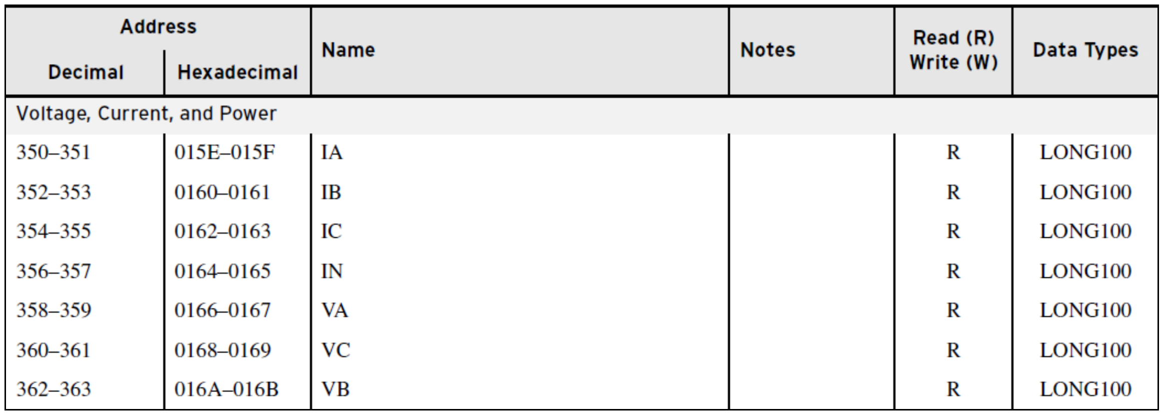
Table 3 shows a mockup of a Modbus register memory map from a non-SEL meter. Note the originally documented addresses in the Address column and the addition of the Raw Address column not present in the non-SEL meter's original table.

Table 3 Non-SEL Meter Modbus Register Memory Map

Label	Address	Raw Address	Number of Registers	Format	Scaling
Vln A	40011	10	1	UINT16	10
Vln B	40012	11	1	UINT16	10
Vln C	40013	12	1	UINT16	10
Vln avg	40014	13	1	UINT16	10

If a master scans these four voltage quantities, it needs to generate a request message using function code 0x03, a register base address of 10, and a register quantity of 4. According to the manufacturer's documentation, the format of each register in the response is an unsigned 16-bit integer (with a range of 0–65535) and has a scaling factor of 10 preapplied to preserve a single decimal place from the original voltage quantity.

As a rule, all Modbus-compatible SEL devices use raw addressing with regards to registers. In Figure 2, a similar set of register data is shown from the SEL-735 Power Quality and Revenue Meter Instruction Manual (Appendix E: Modbus Communications Protocol). In the SEL-735 Instruction Manual, the LONG100 data type is defined as a 32-bit signed integer quantity that occupies two sequential 16-bit register addresses and has a scaling factor of 100 preapplied. All quantities marked with an *R* can be read using function code 0x03.



Address		Name	Notes	Read (R) Write (W)	Data Types
Decimal	Hexadecimal				
Voltage, Current, and Power					
350–351	015E–015F	IA		R	LONG100
352–353	0160–0161	IB		R	LONG100
354–355	0162–0163	IC		R	LONG100
356–357	0164–0165	IN		R	LONG100
358–359	0166–0167	VA		R	LONG100
360–361	0168–0169	VC		R	LONG100
362–363	016A–016B	VB		R	LONG100

Figure 2 SEL-735 Modbus Registers

If a Modbus master wants to scan the three voltage quantities shown in Figure 2, it generates a request using function code 0x03, starting at register address 358 with a register quantity of 6. Each incoming pair of 16-bit registers in the response needs to be reassembled as a 32-bit signed integer (with a range of –2147483647 to 2147483647) and has two decimal points of precision preserved from the original quantity because of the multiplication factor of 100. For reference, the hexadecimal addresses shown in Figure 2 represent the actual hexadecimal digits present in the binary Modbus scan.

The two register addressing approaches illustrated by these examples are used in most cases, but there are a few specific cases where manufacturers have deviated from these standards. Some examples of these deviations for holding registers include the following:

- 1-based raw addressing where the data type offset (e.g., 4X) is implied elsewhere. In this case, subtract 1 from each address.
- Raw addressing that uses a Modicon address (e.g., 40001) as an actual base address. In this case, the raw base address of holding registers is actually 40001, which equates to 0x9C41 in the protocol message. This format is especially confusing because it combines the two standards.

In these cases, it is important to remember that the underlying Modbus protocol itself does not change or deviate, and these are all just syntax variations for register documentation.

Address Conversions

A quick reference for the various data type address conversions is provided by Table 4 through Table 7.

Table 4 Coils

Raw	Modicon Five-Digit	Modicon Six-Digit
0	00001	000001
100	00101	000101
9998	09999	009999
9999	No addressing possible	010000
10000	No addressing possible	010001

Table 5 Discrete Inputs

Raw	Modicon Five-Digit	Modicon Six-Digit
0	10001	100001
100	10101	100101
9998	19999	109999
9999	No addressing possible	110000
10000	No addressing possible	110001

Table 6 Input Registers

Raw	Modicon Five-Digit	Modicon Six-Digit
0	30001	300001
100	30101	300101
9998	39999	309999
9999	No addressing possible	310000
10000	No addressing possible	310001

Table 7 Holding Registers

Raw	Modicon Five-Digit	Modicon Six-Digit
0	40001	400001
100	40101	400101
9998	49999	409999
9999	No addressing possible	410000
10000	No addressing possible	410001

Holding and Input Register Data Type Variations

Recall that Modbus protocol only defines analog register data (holding or input) as a simple 16-bit value. What is not specified about that 16-bit data value is how it should be represented as a real-world quantity. Each particular interpretation of the raw 16-bit register data type can be referred to as a variation.

The most popular variations used in SEL and common third-party devices are documented in this section using the following holding registers and their contents:

- Register 0 contains the 16-bit value of 0x1234 (00010010 00110100 in binary).
- Register 1 contains the 16-bit value of 0xABCD (10101011 11001101 in binary).

16-Bit Signed Integer

For the 16-bit signed integer variation, the quantity is interpreted directly from the 16-bit register contents with two's complement conversion applied to generate negative values (see Table 8). The most significant bit is considered the sign bit, representing negative values if set. The real-time automation controller (RTAC) platform refers to this variation as 16-bit signed MSB (most significant byte). A rarely used reverse-byte-order interpretation uses the corresponding least significant byte (LSB) variant.

Table 8 16-Bit Signed Integer Register Formatting

Minimum Value	Maximum Value	Register 0 Quantity	Register 1 Quantity
–32768	32767	4660	–21555

16-Bit Unsigned Integer

For the 16-bit unsigned integer variation, the quantity is interpreted directly from the 16-bit register contents without any conversion applied, and only positive quantities are supported (see Table 9). The RTAC platform refers to this variation as 16-bit unsigned MSB. A rarely used reverse-byte-order interpretation uses the corresponding LSB variant.

Table 9 16-Bit Unsigned Integer Register Formatting

Minimum Value	Maximum Value	Register 0 Quantity	Register 1 Quantity
0	65535	4660	43981

32-Bit Signed Integer

For the 32-bit signed integer variation, the quantity is interpreted by combining the two 16-bit registers' contents sequentially to create a 32-bit value (0x1234ABCD) with two's complement conversion applied to generate negative values (see Table 10). The most significant bit is considered the sign bit, representing negative values if set. The RTAC platform refers to this variation as 32-bit signed MSR (most significant register). If the two 16-bit registers are interpreted in reverse order (0xABCD1234), the variant used is 32-bit signed LSR (least significant register).

Table 10 32-Bit Signed Integer Register Formatting

Minimum Value	Maximum Value	Register Data	MSR Quantity	LSR Quantity
–2147483648	2147483647	0x1234ABCD	305441741	–1412623820

32-Bit Unsigned Integer

For the 32-bit unsigned integer variation, the quantity is interpreted by combining the two 16-bit registers' contents sequentially to create a 32-bit value (0x1234ABCD) without any conversion applied, and only positive quantities are supported (see Table 11). The RTAC platform refers to this variation as 32-bit unsigned MSR. If the two 16-bit registers are interpreted in reverse-order (0xABCD1234), the variant is 32-bit unsigned LSR.

Table 11 32-Bit Unsigned Integer Register Formatting

Minimum Value	Maximum Value	Register Data	MSR Quantity	LSR Quantity
0	4294967295	0x1234ABCD	305441741	2882343476

32-Bit Floating Point

For the 32-bit floating point variation, the quantity is interpreted by combining the two 16-bit registers' contents sequentially to create a 32-bit value (0x1234ABCD) and applying IEEE 754 single-precision floating-point formatting (see Table 12). The formulas defined by IEEE 754 split the 32-bit value into three components: the sign (1 bit), the exponent (8 bits), and the fraction or mantissa (23 bits). The format itself supports negative and positive values with decimal-point precision, where precision decreases as the quantity itself increases. The RTAC platform refers to this variation as 32-bit float MSR. If the two 16-bit registers are interpreted in reverse order (0xABCD1234), the variant is 32-bit float LSR.

Table 12 32-Bit Floating Point Register Formatting

Minimum Value	Maximum Value	Register Data	MSR Quantity	LSR Quantity
–3.4028E+38	3.4028E+38	0x1234ABCD	5.7009E–28	–1.4571E–12

Bit-Packed 16-Bit Register

The real-world quantities determined from the 16-bit register contents in this case must be considered on a bit-by-bit basis, and the register contents as a whole generally have little meaning. A Boolean value (0 or 1) is assigned to a particular position within the 16-bit register so that each bit position indicates an on/off status for some given signal.

For example, Register 1780 in the SEL-710 Motor Protection Relay is a bit-packed register (Trip Status LO). If the value of Register 0 (0x1234 or 00010010 00110100) is applied to this particular trip status, the SEL-710 will report Jam, Overcurrent, RTD Winding Bearing, Underpower, and Phase Reversal, as shown in Table 13.

Table 13 SEL-710 Trip Status Bit-Packed Register

Bit Position	Value	Description
0	0 (off)	Overload
1	0 (off)	Undercurrent
2	1 (on)	Jam
3	0 (off)	Current underbalance
4	1 (on)	Overcurrent
5	1 (on)	Resistance temperature detector (RTD) winding or bearing trip
6	0 (off)	Positive temperature coefficient switching thermistor
7	0 (off)	Ground current
8	0 (off)	VAR
9	1 (on)	Underpower
10	0 (off)	Undervoltage
11	0 (off)	Overvoltage
12	1 (on)	Phase reversal
13	0 (off)	Power factor
14	0 (off)	Speed switch
15	0 (off)	Neutral current

Conclusion

This paper documents Modbus protocol and many of the common quirks and intricacies that occasionally change a very simple protocol into something much more complicated. Because a core Modbus protocol standard has been maintained that is very rudimentary, much of the more advanced functionality present in other protocols today (advanced numeric types, file transfers, and so on) has been determined by manufacturer-specific design choices, which creates problems with manufacturer interoperability. The paper does not explain every possible variation in addressing or register formatting, but it does provide a good start to understanding the reasoning behind how Modbus is implemented in modern devices.

Biography

Chris Bontje has been involved in the integration, supervisory control and data acquisition (SCADA), and automation disciplines since 2000. He graduated from the Southern Alberta Institute of Technology in 2000 and has worked for a variety of firms in the United States and Canada performing remote terminal unit, programmable logic controller, and SCADA programming and system design. He has been with Schweitzer Engineering Laboratories, Inc. (SEL) since 2011 and works as an application specialist in automation for the south-central region, which includes Colorado, Oklahoma, Texas, Kansas, and Missouri. He has written numerous technical documents at SEL concerning protocol decoding and real-time automation controllers.

SCHWEITZER ENGINEERING LABORATORIES, INC.
2350 NE Hopkins Court • Pullman, WA 99163-5603 USA
Tel: +1.509.332.1890 • Fax: +1.509.332.7990
www.selinc.com • info@selinc.com

LWP0020-01