ELKOR Technologies Inc.
6 Bainard Street
London, Ontario N6P 1A8 CANADA
tel(519)652-9959 fax(519)652-1057

Application Note AN1702

Understanding Modbus Registers

Summary

Modbus is a well-recognized and highly flexible communication protocol. Modbus originated within industrial communications, but has been adapted to many device communications platforms.

Although Modbus is well accepted unfortunately, the standard has not been updated to clarify various implementations and some confusion exists around a number of elements of the protocol.

This document attempts to clarify some confusion on various aspects of Modbus, and discuss how some of these aspects are implemented in Elkor's devices.

Modbus Register Addressing

Modbus registers by definition are associated with a function, and an offset within that function. The various functions define certain types of registers (bits, data registers, etc). Some registers are considered read-only and some may be written to. There are separate functions for reading the various register types, writing the various types, as well as ancillary diagnostic functions. The two common (16-bit) data register types are commonly known as "Holding Registers" and "Input Registers" (function 03 and function 04 respectively).

The specific register within the function is referenced by an offset (starting at 0). This is the actual data which is transmitted during the data query. At some point, certain PLC manufacturers starting using a "3xxxx" or "4xxxx" reference designation in an attempt to provide an absolute address to the register (ie: which would reference both the function and the register). Note however, that it is counter-intuitive since the "3x" references function 04 and the "4x" references function 03, which furthers confusion.

To add more confusion to the scenario, some device manufacturers start their "4xxxx" references at 40001, and some start at 40000. The starting register corresponds to offset "0" within the given function. It is not always evident which reference a manufacturer uses, and it is best to clearly read the documentation to attempt to gain understanding.

The three addressing methodologies are as follows:

Modbus "Standard"	4xxxx (base 1)	4xxxx (base 0)
Function 3, Offset 0	40001	40000
Function 3, Offset 1	40002	40001

It is imperative to understand which addressing methodology is being used by the system so that the correct registers are accessed. This is especially important in the case of 32-bit registers because an "out-by-one" situation (where the wrong register offset is read by one placement) can cause corrupt data being read.

In an attempt to provide some clarity, Elkor Technologies Inc. includes both the "Offset" and "4xxxx (Base 1)" notation in its documentation. For example:

Name	Offset	Address	Size	Type	R/W	Default	Description
Primary CT Ratio (All)	0x500	41281	16	U	RW	*	Used for setting the CT ratios of each phase.

The table shows both the "Offset" notation, as well as the "Address" (4xxxx) notation.

Elkor's devices typically have a "debug" register which can be used to determine the proper addressing methodology. Newer devices contain both 16 and 32 bit debug registers such that testing can be performed on multiple data types. It is important to use these registers when initially setting up a system to avoid issues with incorrect data mapping later.

Register "Endianness"

Per the Modbus Standard, 16-bit registers are defined as "big-endian". This means that the more significant byte is followed by the less significant byte to form a 16-bit word. For instance, when a 16-bit word 0xABCD is transmitted, it is in the order 0xAB, 0xCD.

Unfortunately the Modbus standard does not address larger data types (32-bit or 64-bit), and manufacturers have not consistently implemented these data types. It seems obvious that 32-bit and 64-bit values should also be transferred using big-endian order. However, some manufacturers have chosen to transfer the words in little-endian.

For example:

1) The 32-bit value 0x1234567 could be transferred using "big-endian" order as:
   0x1234, 0x5678
   or specifically, on a byte level as:
   0x12, 0x34, 0x56, 0x78

2) The 32-bit value 0x1234567 could be transferred using "little-endian" order as:
   0x5678, 0x1234
   or more specifically on a byte level as:
   0x56, 0x78, 0x12, 0x34

The endian order of the system is imperative to properly read 32-bit registers. Both the master and the slave must use the same word order.

Elkor's devices are by default configured as big-endian (therefore example #1), but in most cases may be optionally configured (via a Modbus Setting) to be little-endian.

The endian order is critical to read proper data values. Obviously if the 16-bit word data is reversed, then the 32-bit data will not be as expected.

As with the Register Addressing, the word order should be confirmed by using the available debug registers.