# Chapter 5: Modbus Protocol

The Modbus protocol is a powerful and flexible communications protocol that allows numerous software programs and hardware devices to communicate with each other. Any I-8xx7, I-7188EG/XG & W-8xx7 variable that will be used to communicate through the Modbus protocol **MUST** have a unique network address before it can communicate through a Modbus link (please refer to section 4.1).

## 5.1: Modbus Protocol Format: RTU Serial

The Modbus "RTU Serial" format is supported by the I-8417 and I-8817 controller systems through both COM1 or COM2 communications ports, and the I-8437, I-8837, I-7188EG & I-7188XG controller systems through the COM1 communications port, and the Wincon-8x37 & Wincon-8x47 controller systems through the COM2 (or COM3) communications port.

PC software programs and HMI hardware devices can access data from the variables in the ISaGRAF controller system **ONLY** after that variable is assigned a unique network address (please refer to Chapter 4). For more information regarding connecting a PC to an I-8xx7 controller system, please refer to Section 1.3.3 through 1.3.5 for details on how to properly connect these devices.

It is **CRITICAL** that you must program the Modbus format **EXACTLY** as described to make a proper connection between the Modbus device and the ISaGRAF controller system. The I-8xx7, I-7188EG/XG & W-8xx7 controllers support the following Modbus functions.

| Modbus function | Action |
|---|---|
| 1 | Read N bits (booleans) |
| 2 | Read N bits (booleans) |
| 3 | Read N words<br>(signed short integers) |
| 4 | Read N words<br>(signed short integers) |
| 5 | Write 1 bit (boolean) |
| 6 | Write 1 word<br>(signed short integer) |
| 15 | Write N bits (booleans) |
| 16 | Write N words<br>(signed short integers) |

To read boolean variables, both of function 1 or 3 may be used. If using function 1, values are stored in a bit field while using function 3, variable TRUE means 0xFFFF.

To write boolean variables, both of function 5, 15 could be used. If using function 5, writing bit 0 of byte-vH to 1 will set the Boolean variable to TRUE. For ex, writing vH=1 or 3, or 255 will set Boolean variable to TRUE.

To read analog variables, function 3 should be used.

To write analog variables, both of function 6, 16 could be used.

To read long words (signed long integers, float), function 3 should be used. To write long words, function 16 should be used. Please refer to section 4.2 for the definition of network address of long words.

To assist you with the naming conventions used throughout the Modbus protocol-addressing chapter, the following table describes the notations used in this chapter.

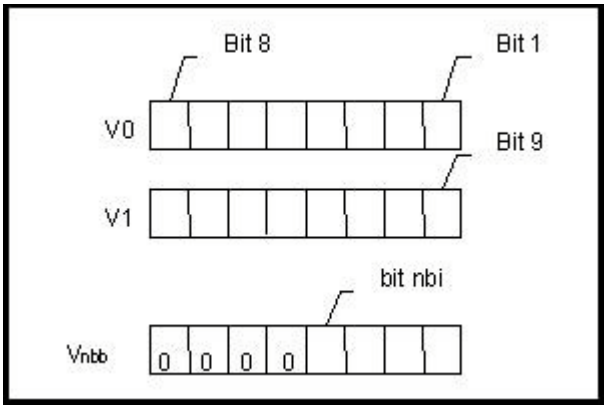
| | |
|---|---|
| Slv | Slave number (Net ID address of the I-8xx7) |
| Nbw | Number of words |
| Nbb | Number of bytes |
| Nbi | Number of bits |
| AddH | **Modbus address**, high byte , 0 ~ 0F |
| AddL | **Modbus address**, low byte , 0 ~ FE |
| VH | Word value, high byte |
| VL | Word Value, low byte |
| V | Byte value |
| CrcH | Checksum, high byte , CRC-16 |
| CrcL | Checksum, low byte , CRC-16 |

**IMPORTANT NOTE**
All of the values used in the request and answer frames are **hexadecimal** values.
Modbus address described in this chapter is equal to Network address of the variable minus one.
For ex., Modbus address 0 is associate with Network address 1. Modbus address FFE (4094) is associate with Network address FFF (4095).

Function 1: Read "N" Bits
Function 1 reads "n" number of bits (nbi) in Boolean starting from Modbus address addH/addL.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Request: | slv | 01 | addH | addL | 00 | nbi | crcH | crcL |
| Answer: | slv | 01 | nbb | V0 | V1 | … | crcH | crcL |

V0, V1 … are the bit fields of number of bytes (nbb) using the following format.




Bit 1 corresponds to the Boolean value of the variables with the Modbus address addH/addL. Bit nbi corresponds to the Boolean value of the variable with the Modbus address addH/addL + nbi – 1. If the value of the Boolean variable is "True", then the corresponding bit will be set to a "1". If the value is "False", the corresponding bit will be set to a "0".

Function 2: Read N Bits
Function 2 has the same exact same format as function 1.

Function 3: Read N Words
Function 3 reads the number of words (nbw), in signed 16-bit integer format, starting from the Modbus address addH/addL.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Request: | slv | 03 | addH | addL | 00 | nbw | crcH | crcL |
| Answer: | slv | 03 | nbb | vH | vL | ... | crcH | crcL |

The number of bytes (nbb) is the total number of bytes from word value high byte (vH) to word value low byte (vL) inclusive.

**IMPORTANT NOTE About Function 3**
Integer values can be read by function 3. A word in the modbus protocol is a 16-bit value (signed short integer), and an integer variable is a 32-bit value, so only the lower 16 bits of the integer variable are returned. If users would like to read a 32-bit integer (signed long integer) of I-8xx7 controller, the proper network address of the variable should be set as described in section 4.2.

Function 4: Read N Words
Function 4 has the same exact format as function 3.

Function 5: Write 1 Bit
Function 5 writes one (1) bit to the Boolean variable with the Modbus address addH/addL.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Request: | slv | 05 | addH | addL | V | 0 | crcH | crcL |
| Answer: | slv | 05 | addH | addL | V | 0 | crcH | crcL |

Writing a 0xFF value to the byte value (V) will set the Boolean variable to "True". Writing a zero to the byte value (V) is set the Boolean variable to "False".

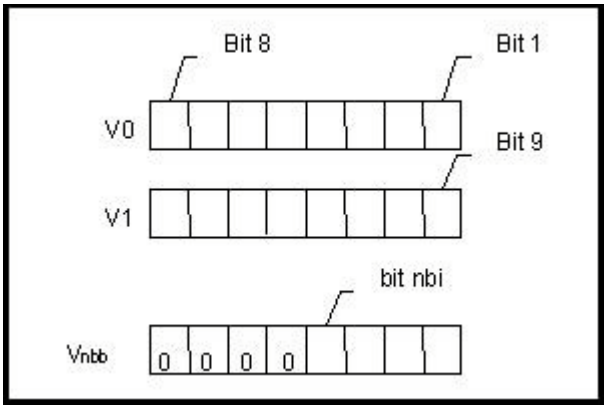
Function 6: Write 1 Word
Function 6 writes one (1) word (16 bits) to the integer variable with the Modbus address addH/addL.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Request: | slv | 06 | addH | addL | vH | vL | crcH | crcL |
| Answer: | slv | 06 | addH | addL | vH | vL | crcH | crcL |

Function 15: Write N Bits
Function 15 writes a number of bits (nbi) to the Boolean variables starting from the Modbus address addH/addL to addH/addL + nbi – 1. The total number of bytes (nbb) is the total amount of bytes occupied by nbi bits, that means nbb = (nbi+7)/8. For ex. nbi=1~8, nbb=1; nbi=9~16, nbb=2.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Request: | slv | 0F | addH | addL | 00 | nbi | nbb | V0 | V1 | ... | crcH | crcL |
| Answer: | slv | 0F | addH | addL | 00 | nbi | crcH | crcL | | | | |

V0, V1 … are the bit fields of number of bytes (nbb) using the following format.




Bit 1 corresponds to the Boolean value of the variables with the Modbus address addH/addL. Bit nbi corresponds to the Boolean value of the variable with the Modbus address addH/addL + nbi – 1. Writing a 1 to a bit will set the value of the corresponding Boolean variable to "True", and writing a 0 to a bit will set the corresponding Boolean variable to "False".

Function 16: Write N Words

Function 16 writes a number of words (nbw) to the integer variables starting from the Modbus address addH/AddL to addH/addL + nbw – 1. The number of bytes (nbb) is the total amount of bytes occupied by number of words (nbw), that is nbb = 2 * nbw.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Request: | slv | 10 | addH | addL | 00 | nbw | nbb | vH | vL | ... | crcH | crcL |
| Answer: | slv | 10 | addH | addL | 00 | nbw | crcH | crcL | | | | |

Examples Of Modbus Function Formats

**Function 1:** Read 15 bits starting from **Modbus address 0**x1020. The NET ID address is 1.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Request: | 01 | 01 | 10 | 20 | 00 | 0F | 79 | 04 |
| Answer: | 01 | 01 | 02 | 00 | 12 | 39 | F1 | |

In this example function 1 returns 2 bytes, the value is 0x0012. This means variables with a **network address** of 0x102A and 0x102D are "True" (**Modbus address** is 0x1029 and 0x102C), the rest of the variables are set to "False".

**Function 5:** Write 1 bit to the Boolean variable with the **Modbus address** 0x0006. The NET ID address is 1. The value to write to is 0xFF.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Request: | 01 | 05 | 00 | 06 | FF | 00 | 6C | 3B |
| Answer: | 01 | 05 | 00 | 06 | FF | 00 | 6C | 3B |

In this example of function 5 the Boolean variable is set to "True".

**Function 16:** Write 2 words (4 bytes) to the integer variables with the **Modbus address** starting from 0x2100. The first word value to write to is 0x1234. The second word value to write to is 0x5678. The NET ID address is 1.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Request: | 01 | 10 | 21 | 00 | 00 | 02 | 04 | 12 | 34 | 56 | 78 | 1C | CA |
| Answer: | 01 | 10 | 21 | 00 | 00 | 02 | 4B | F4 | | | | | |

# 5.2: Modbus Protocol Format: TCP/IP

The I-8437 / I-8837 , I-7188EG, W-8x37 and W-8x47 controller systems support the Modbus "TCP/IP" communications protocol.

ALL requests are sent via TCP on port number **502**.

The Modbus TCP/IP protocol adds 6 extra bytes before the Modbus RTU serial protocol, and these 6 extra bytes and the Modbus RTU serial protocol are all packed inside the TCP/IP protocol.

| TCP/IP | Extra 6 Bytes | Modbus RTU serial | TCP/IP |
|---|---|---|---|

The request and responses are prefixed by the six bytes as follows:

Byte 0: transaction identifier - copied by server
Byte 1: transaction identifier - copied by server
Byte 2: protocol identifier = 0
Byte 3: protocol identifier = 0
Byte 4: length field (upper byte) = 0 (since all messages are smaller than 256)
Byte 5: length field (lower byte) = number of following bytes

The rest of the Modbus TCP/IP protocol is the same as the Modbus RTU Serial protocol after byte No. of 6 except that the CRC-16 is not need for the Modbus TCP/IP protocol.

Example TCP/IP Transactions
The first example of a TCP/IP transaction is reading one (1) word at Modbus address 4 from slave number 9 returning a value of 8; the transaction would be as follows:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Request: | 01 | 02 | 00 | 00 | 00 | 06 | 09 | 03 | 00 | 04 | 00 | 01 |
| Response: | 01 | 02 | 00 | 00 | 00 | 05 | 09 | 03 | 02 | 00 | 08 | |

The second example of a TCP/IP transaction is reading 8 bits starting from Modbus address 2 from slave number 7, returning a value of 0x49 (bit field: 01001001) would be as follows:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Request: | 03 | 29 | 00 | 00 | 00 | 06 | 07 | 01 | 00 | 02 | 00 | 08 |
| Response: | 03 | 29 | 00 | 00 | 00 | 04 | 07 | 01 | 01 | 49 | | |

# 5.3: Algorithm For CRC-16 Check

**The following C language algorithm is for Modbus RTU Serial ONLY!!** This CRC (Cyclic Redundancy Check) program provides a checksum that can be used to validate information being passed through Modbus RTU Serial protocol.
This CRC-16 check program first calls "crc_init()" one time at the beginning of the communication to initialize the checksum table. Then you can call "crc_make()" to calculate a checksum whenever you want to.

```
#define POLY_CRC16 0xA001
static BYTE TABLE1[256];
static BYTE TABLE2[256];

void crc_init(void)   /* set crc table */
{
  WORD mask,bit,crc,mem;
  for(mask=0;mask<0x100;mask++)
  {
    crc=mask;
    for(bit=0;bit<8;bit++)
    {
      mem=crc & 0x0001;
      crc/=2;
      if(mem!=0) crc ^= POLY_CRC16;
    }
    TABLE2[mask]=crc & 0xff;
    TABLE1[mask]=crc >> 8;
  }
}

void crc_make(WORD size, BYTE *buff, BYTE *hi, BYTE *lo)   /* calculate crc */
{
  BYTE car,i;
  BYTE crc[2];
  crc[0]=0xff;
  crc[1]=0xff;
  for(i=0;i<size;i++)
  {
    car = buff[i];
    car ^= crc[0];
    crc[0]=crc[1] ^ TABLE2[car];
    crc[1]=TABLE1[car];
  }
  *hi=crc[0];
  *lo=crc[1];
}
```