Application note AN OM2006 – 005

Protocol MODBUS description of ORBIT MERRET's instrument

Ing. Jiri Antos

26. 10. 2006

Introduction

Implementation of protocol MODBUS is described in this document. At the beginning is necessary to say MODBUS is third protocol installed in ORBIT MERRET instruments. It is intended for connectivity with control or superior system. ORBIT MERRET sell software named OM LINK for easy control of instrument. It is work only by ASCII communication protocol.

Communication parameters - low level protocol

Transfer medium:	RS485, twisted pair, maximum length 1200 m (4000 feet) maximum number of devices on line is 32 for more devices repeater is require
Baud rate:	600, 1 200, 2 400, 4 800, 9 600, 19 200, 38 400, 57 600, 115 200, 230 400 Baud
Start-bits:	1
Data-bits:	8
Start-bits:	1
Parity:	none
Inter-telegram distance:	>3.5characters *)
Inter-symbol distance:	Not monitored

*) Communication is terminated provided no data arrives during 3 1/2 characters. This period is determined with uncertainty of ±250µs. MODBUS has standard rates up to 19 200. For higher rate it is necessary to count with this uncertainty - e.g. 115 200 Baud -> 500±250 µs, 230 400 Baud -> 250 ±250 µs.

MODBUS RTU telegram format

<ADR> <FNC> <DATA> <CRC-L> <CRC-H>
<ADR> field is 1 byte address in range from 1 to 247. Broadcast address 0 is not supported by ORBIT MERRET instruments.
<FNC> field is 1 byte command, which specifies the <DATA> field content. Detailed description of functions and their data is bellow.
<CRC> field is 2 bytes Cyclical Redundancy Check which is described in Appendix A.

Supported data formats

1 byte length	Abreviation	Range	Placing in telegram
index of List	List	0 .. 255	<0x00><Byte>
unsigned char	Char	0 .. 255	<0x00><Byte>

2 byte length		Range	Placing in telegram
unsigned integer	Int	0 .. 65535	<High><Low>
2-characters description	Desc	' ' .. ' ', 0x20 .. 0x7F	<First><Second>

3 byte length		Range	Placing in telegram
3-characters description	Desc3	' ' .. ' ', 0x20 .. 0x7F	<0x00><First><Second><Third>

4 byte length		Range	Placing in telegram
unsigned long integer	Long	0 .. 4 294 967 295	<0x00><First><Second><Third>
IEEE floating point	Float	±6,80564693277E+38	<SE><EM><MM><MM>

Description of Float

first byte	<SE>	SEEE EEE	Z…sign (1(0)/-1(1)); E…Exponent (-127(0x00)…0(0x7F)…128(0xFF))
second byte	<EM>	EMMM MMMM	M…Mantissa (1.0…2.0), highest mantissa bit is always 1 and it is covered by the lowest exponent bit
3., 4. byte	<MM><MM>		rest of mantissa bits e.g.: 0x3F80 0000 = Z*2ˆE*M = 1*2ˆ(0)*1 = 1

DETAILED COMMAND DESCRIPTION

Command 0x01 - Read coils, state of relays

Request: <ADR><0x01><0x00><0x00><0x00><0x08><CRC Lo><CRC Hi>
Response: <ADR><0x01><0x01><RR><CRC Lo><CRC Hi>
<RR> there is binary coded state of relays. Not installed relay return 0

0x80 ... relay 8
0x40 ... relay 7
0x20 ... relay 6
0x10 ... relay 5
0x08 ... relay 4
0x04 ... relay 3
0x02 ... relay 2
0x01 ... relay 1

Command 0x02 - Read external inputs

Request: <ADR><0x02><0x00><0x00><0x00><0x08><CRC Lo><CRC Hi>
Response: <ADR><0x02><0x01><Inputs><CRC Lo><CRC Hi>
<Inputs> there is binary coded state of external inputs. Not installed input return 0

0x80 ... input 8
0x40 ... input 7
0x20 ... input 6
0x10 ... input 5
0x08 ... input 4
0x04 ... input 3
0x02 ... input 2
0x01 ... input 1

Command 0x03 - Read menu items

There is difference depend on menu item data format.

1 or 2 bytes format
Request: <ADR><0x03><AA High><AA Low><0x00><0x01><CRC Lo><CRC Hi>
Response: <ADR><0x03><Data Hi><Data Lo><CRC Lo><CRC Hi>

3 or 4 bytes format
Request: <ADR><0x03><AA High><AA Low><0x00><0x02><CRC Lo><CRC Hi>
Response: <ADR><0x03><MSB of data><data><data><LSB of data><CRC Lo><CRC Hi>

Warning: Multiple menu items reading is not supported.

Command 0x04 - Read measured or evaluated values

This data alwas have been in Float.
Request: <ADR><0x04><AA High><AA Low><0x00><0x02><CRC Lo><CRC Hi>
Response: <ADR><0x04><MSB of data><data><data><LSB of data><CRC Lo><CRC Hi>

Warning: Multiple values reading is not supported.

Command 0x05 - Do action

Request: <ADR><0x05><AA High><AA Low><0xFF><0x00><CRC Lo><CRC Hi>
Response: <ADR><0x05><AA High><AA Low><0xFF><0x00><CRC Lo><CRC Hi>

Warning: Relays are control by instrument. This command start action. Multiple actions doing is not supported.

Command 0x06 - Set menu items
This command is intended for servicing 1 or 2 bytes menu item data.

Request:	<ADR><0x06><AA High><AA Low><Data Hi><Data Lo><CRC Lo><CRC Hi>
Response:	<ADR><0x06><AA High><AA Low><Data Hi><Data Lo><CRC Lo><CRC Hi>

Warning: Multiple menu items setting is not supported.

Command 0x10 - Set menu items
This command is intended for servicing 3 or 4 bytes menu item data.

Request:	<ADR><0x10><AA High><AA Low><0x00><0x02> <0x04><MSB of data><data><data><LSB of data><CRC Lo><CRC Hi>
Response:	<ADR><0x10><AA High><AA Low><0x00><0x02><CRC Lo><CRC Hi>

Warning: Multiple menu items setting is not supported.

Command 0x11 - Slave identification
This command is intended for servicing 3 or 4 bytes menu item data.

Request:	<ADR><0x11><CRC Lo><CRC Hi>			
Response:	<ADR><0x11><0x1C><0xFF><0xFF>			
	<data> .. <data>	12 characters of name of instrument	e.g.	`"OM 402UNI    "`
	<'-'><data> .. <data>	6 characters of firmware version	e.g.	`"62-001"`
	<'-'><data> .. <data>	6 characters of measurement mode	e.g.	`"  40 V"`
	<CRC Lo><CRC Hi>			

Supported addresses in <AA High><AA Low> field
List of this address is possible get on demand at orbit@merret.cz or at our website. Address should be used without first from 5 digits, e.g. 40012 should be used as <ADR><0x06>**<0x00><0x0C>**<0x02>...

Exception response
In case of wrong address or CRC there is nothing response.
If case of any error is exception response returned:

<ADR> <FNC & 0x80> 01 <CRC Lo> <CRC Hi>	not supported command was received
<ADR> <FNC & 0x80> 02 <CRC Lo> <CRC Hi>	wrong register address
<ADR> <FNC & 0x80> 03 <CRC Lo> <CRC Hi>	wrong number of coils or registers
<ADR> <FNC & 0x80> 04 <CRC Lo> <CRC Hi>	command have not to be execute. There may be wrong data or menu item is dynamically disabled.

Appendix A - CRC GENERATION

The Cyclical Redundancy Check (CRC) field is two bytes, containing a 16–bit binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The receiving device recalculates a CRC during receipt of the message, and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results. The CRC is started by first preloading a 16–bit register to all 1's. Then a process begins of applying successive 8–bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and stop bits, and the parity bit, do not apply to the CRC. During generation of the CRC, each 8–bit character is exclusive ORed with the register contents. Then the result is shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position. The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value. If the LSB was a 0, no exclusive OR takes place. This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next 8–bit character is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final contents of the register, after all the characters of the message have been applied, is the CRC value.

A procedure for generating a CRC is:
1, Load a 16–bit register with FFFF hex (all 1's). Call this the CRC register.
2. Exclusive OR the first 8–bit byte of the message with the low–order byte of the 16–bit CRC register, putting the result in the CRC register.
3. Shift the CRC register one bit to the right (toward the LSB), zero–filling the MSB. Extract and examine the LSB.
4. (If the LSB was 0): Repeat Step 3 (another shift). (If the LSB was 1): Exclusive OR the CRC register with the polynomial value A001 hex (1010 0000 0000 0001).
5. Repeat Steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8–bit byte will have been processed.
6. Repeat Steps 2 through 5 for the next 8–bit byte of the message. Continue doing this until all bytes have been processed.
7. The final contents of the CRC register is the CRC value.
8. When the CRC is placed into the message, its upper and lower bytes must be swapped as described below.

Placing the CRC into the Message
When the 16–bit CRC (two 8–bit bytes) is transmitted in the message, the low-order byte will be transmitted first, followed by the high-order byte.

Example
An example of a C language function performing CRC generation is shown on the following pages. All of the possible CRC values are preloaded into two arrays, which are simply indexed as the function increments through the message buffer. One array contains all of the 256 possible CRC values for the high byte of the 16–bit CRC field, and the other array contains all of the values for the low byte. Indexing the CRC in this way provides faster execution than would be achieved by calculating a new CRC value with each new character from the message buffer. Note This function performs the swapping of the high/low CRC bytes internally. The bytes are already swapped in the CRC value that is returned from the function. Therefore the CRC value returned from the function can be directly placed into the message for transmission.

The function takes two arguments:
unsigned char *puchMsg ;
A pointer to the message buffer containing binary data to be used for generating the CRC
unsigned short usDataLen ;
The quantity of bytes in the message buffer. The function returns the CRC as a type unsigned short.

CRC Generation Function

```
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg ;                        /* message to calculate CRC upon */
unsigned short usDataLen ;                      /* quantity of bytes in message */
{
        unsigned char uchCRCHi = 0xFF ;         /* high byte of CRC initialized */
        unsigned char uchCRCLo = 0xFF ;         /* low byte of CRC initialized */
        unsigned uIndex ;                       /* will index into CRC lookup table */

        while (usDataLen--)                     /* pass through message buffer */
        {
                uIndex = uchCRCHi ^ *puchMsgg++ ;    /* calculate the CRC */
                uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex} ;
                uchCRCLo = auchCRCLo[uIndex] ;
        }
        return (uchCRCHi << 8 | uchCRCLo) ;
}

/* Table of CRC values for high-order byte */
static unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
} ;

/* Table of CRC values for low-order byte */
static char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
} ;
```