Overview

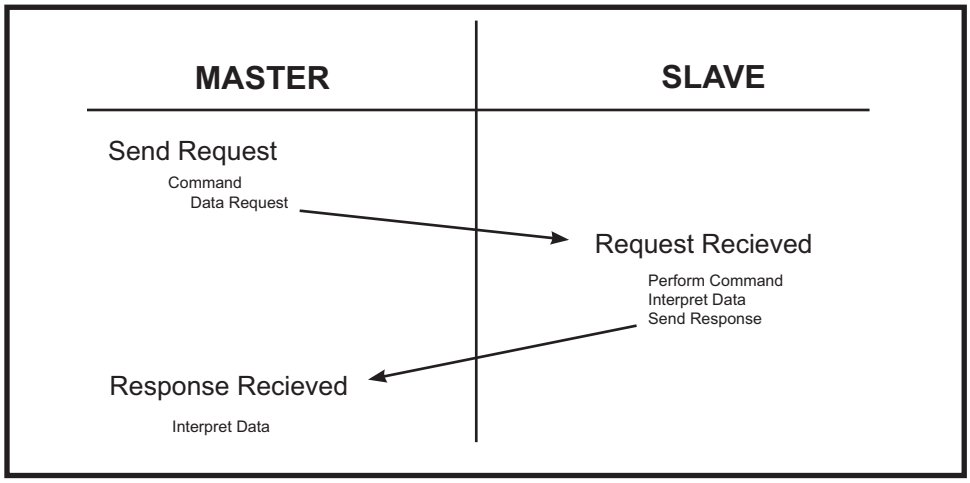
Modbus protocol is a widely used and well-documented communcations method.

It provides a simple and effective means of programming our various products.

A typical Modbus packet looks like this:

Byte1	Device ID, the destination address for a particular message
Byte2	Function
Byte3	Starting address of the particular storage register(s) to be read or written, hi byte,
Byte4	Starting address low byte
Byte5	No of registers to read/write (hi byte)
Byte6	No of registers to read/write (low byte)
Byte7	CRC hi byte
Byte8	CRC low byte

During normal operation, the slave will immediately send a response to the master request.



[Notice]: Most errors during message transfer is normally seen as a timeout errors.
This is because bytes being distorted or missing will not trigger a response resulting in a timeout error.

Software tools can be found at: http://www.modbustools.com/modbus_poll.asp

If your application can read & write bytes to a separate PC running the 'Modbus Slave' application then you can read & write bytes to our MPC´s.

Note: When using the Modbus Poll software, addressing should be set to "Protocol Addresses (Base 0)" under the "Display" menu.

MODBUS EXAMPLES

READ Command (0x03):

This function is used to read the contents of multiple memory registers.

The master to the Modbus must specify, the device ID, it's starting register and quantity of register desired.

By convention if a data were to contain 2 byte, we would first send the Hi byte and then the Lo byte.

The master to the Modbus network will issue a read command:
- Device ID=11
- Read 6 bytes of data
- Starting at register number 107 (6Bh)

Byte #	Field Name (Hex)	Data	Description
Byte1	Slave Address	11	MPC with ID11 will be read
Byte2	Function	03	Read operation
Byte3	Starting Address Hi	00	
Byte4	Starting Address Lo	6B	Reading starting from register #6B
Byte5	No. of Register to read Hi	00	
Byte6	No. of Register to readLo	03	Read a total of 3 registers
Byte7	Error Check (CRC) HI byte	XX	The CRC is calculated using the CRC
Byte8	Error Check (CRC) LO byte	XX	routine described below

The slave device with ID=11 will answer the master within a few milliseconds with the following response.

Byte #	Field Name (Hex)	Data	Description
Byte1	Slave Address	11	Slave with ID11 is responding
Byte2	Function	03	we're responding to a read command
Byte3	Byte Count	06	6 bytes are coming
Byte4	Data1 Hi	02	byte1 of the data
Byte5	Data1 Lo	2B	byte2 of the data
Byte6	Data2 Hi	00	byte3 of the data
Byte7	Data2 Lo	00	byte4 of the data
Byte8	Data3 Hi	00	byte5 of the data
Byte9	Data3 Lo	64	byte6 of the data
Byte10	Error Check (CRC) HI byte	XX	The CRC is calculated using the CRC
Byte11	Error Check (CRC) LO byte	XX	routine described below

Example of the Read Command

The Master sends the Read querie:

Slave Address	Function	Starting Address Hi	Starting Address Lo	No. of Regs Hi	No. of Regs Lo	CRC Hi Byte	CRC Lo Byte
11	3	0	6Bh 107	0	3	xx	xx

The device node sends back the following response:

Slave Address	Function	Byte Count	Data1 Hi	Data1 Lo	Data2 Hi	Data2 Lo
11	3	6	(02h) 2	(2Bh) 43	(00h) 0	(00h) 0

Data3 Hi	Data3 Lo	CRC Hi Byte	CRC Lo Byte			
(00h) 0	(64h) 100	xx	xx			

WRITE command (0x06):

This function is used to write to a single memory register.

The master of the Modbus must specify the device ID, its register address to be written and the data desired.

The master to the Modbus network will issue a write command:
- Device ID=11
- Write to address 11
- Enter data 3 (03h)

Byte #	Field Name (Hex)	Data	Description
Byte1	Slave Address	11	destination address
Byte2	Function	06	this is a write command
Byte3	Register Address Hi	00	address which will be written to, hi byte
Byte4	Register Address Lo	01	address which will be written to, low byte
Byte5	Data Hi	00	data that we are writing, hi byte
Byte6	Data Lo	03	data we are writing, low byte
Byte7	Error Check (CRC) HI byte	XX	The CRC is calculated using the CRC
Byte8	Error Check (CRC) LO byte	XX	routine described below

The slave device with ID=11 will answer the master within a few milliseconds with the following response.

Byte #	Field Name (Hex)	Data	Description
Byte1	Slave Address	11	destination address
Byte2	Function	06	this is a write command
Byte3	Register Address Hi	00	address which will be written to, hi byte
Byte4	Register Address Lo	01	address which will be written to, low byte
Byte5	Data Hi	00	data that we are writing, hi byte
Byte6	Data Lo	03	data we are writing, low byte
Byte7	Error Check (CRC) HI byte	XX	The CRC is calculated using the CRC
Byte8	Error Check (CRC) LO byte	XX r	outine described below

[Notice]: In this case the Slave device just sends back the message to let the Master know the query has been properly receive.

Example of the Write Command

The Master sends the Write querie:

Slave Address	Function	Register Address Hi	Register Address Lo	Data Hi	Data Lo	CRC Hi Byte	CRC Lo Byte
11	6	0	(01h) 1	0	3	xx	xx

The device node sends back the following response:

Slave Address	Function	Register Address Hi	Register Address Lo	Data Hi	Data Lo	CRC Hi Byte	CRC Lo Byte
11	6	0	(01h) 1	0	3	xx	xx

Multiple-Write Command (0x10):

This function is used to write to multiple memory register. The master of the Modbus must specify the device ID, its starting address register, the amount of register desired and the data.
NOTE: This is used for firmware update only. It is not used to write device registers.

The master to the Modbus network will issue a multiple-write command:
- Device ID=11
- Write to address 291 (123h)
- Number of Registers 3
- Data1 = 10 (000Ah)
- Data2 = 11 (000Bh)
- Data3 = 12 (000Ch)

Byte #	Field Name (Hex)	Data	Description
Byte1	Slave Address	11	destination address ID 11
Byte2	Function	10	this is a multiple write command
Byte3	Register Start Address Hi	01	this is the address we are currently writing to in the code space of the device
Byte4	Register Start Address Lo	23	in this case we want to write to register address 0x0123
Byte5	Quantity of Registers to write HI	00	we will be writing a variables amount of bytes at a time
Byte6	Quantity of Registers to write LOW	10	in this case we want to write to 10H or 16 registers
Byte7	Byte Count	20	if byte count is same as Number of Registers, dealing with 8 bits If byte count is double as Number of Registers, dealing with 16 bits

Byte #	**8 bits**
Byte8	Data 1
Byte9	Data 2
Byte10	Data 3
Byte11	Data 4
[…]	
Byte22	Data 15
Byte23	Data 16
Byte 24	Error Check HI
Byte 25	Error Check LO

Byte #	**16 bits**
Byte8	Data1 Hi
Byte9	Data1 Lo
Byte10	Data2 Hi
Byte11	Data2 Lo
[…]	
Byte38	Data16 Hi
Byte39	Data16 Lo
Byte40	Error Check HI
Byte41	Error Check LO

[Notice]: Byte 7 is used as a byte count.
Thus if the byte count is the same as the number of registers to write then we know we are dealing with 1 byte registers. Similarly, if the byte count is double than the number of registers, we are not dealing with 2 byte registers.

The slave device with ID=11 will answer the master within a few milliseconds with the following response.

Byte #	Field Name (Hex)	**Data**	**Description**
Byte1	Slave Address	11	destination node ID
Byte2	Function	10	this is a multiple write command
Byte3	Register Start Address Hi	00	starting address we are writing to, hi byte
Byte4	Register Start Address Lo	01	start address low byte
Byte5	Quantity of Registers Hi	00	Number of registers to be written to, hi byte
Byte6	Quantity of Registers Lo	0A	Number of registers, low byte
Byte7	Error Check (CRC) HI byte	XX	The CRC is calculated using the CRC
Byte8	Error Check (CRC) LO byte	XX	routine described previously

Example of the Multiple-Write Command

The Master sends the Multiple-Write querie:

Slave Address	Function	Starting Address Hi	Starting Address Lo	Quantity. of Regs Hi	Quantity. of Regs Lo	Byte Count
11	(10h) 16	(01h) 1	(23h) 35	0	3	6

Data1 Hi	Data1 Lo	Data2 Hi	Data2 Lo	Data3 Hi	Data3 Lo	CRC Hi Byte	CRC Lo Byte
(00h) 00	(2Ah) 10	(00h) 00	(0Bh) 12	(00h) 00	(0Ch) 13	xx	xx

Slave Address	Function	Starting Address Hi	Starting Address Lo	Quantity. of Regs Hi	Quantity. of Regs Lo	CRC Hi Byte	CRC Lo Byte
11	10	(01h) 1	(23h) 35	0	3	xx	xx

CRC Error Correcting Details

The following is a collection of code snippets to get your application started.

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 unsigned 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
} ;

For example to calculate the crc of the data in message stored in memory location *puchMsg

```
unsigned short CRC16(unsigned char *puchMsg, unsigned char usDataLen)
{
    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 ^ *puchMsg++ ; /* calculate the CRC */
         uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
          uchCRCLo = auchCRCLo[uIndex] ;
     }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```