Modbus in MAC400/800

The modbus implementation in MAC400/800 is a subset of the Modbus Specification V1.1b. This standard can be downloaded free of charge from the website www.modbus.org.

The serial communications lines normally used for communications between the basic motor and one of the intelligent MAC00-XX modules, can be configured to use the Modbus protocol instead of the standard FastMac protocol.

The MAC400/800 firmware supports the two command types Read Holding Registers (3) and Write Multiple Register (0x10). All other commands will result in Exception replies (exception type 1, Illegal Function).

All registers can be read as well as written over Modbus, but the number of registers per transfer is limited to 16 16-bit registers or 8 32-bit registers. Contact JVL if more registers are needed in a single transfer.

All registers in the MAC400/800 motors are 32-bits. To comply with the clean 16-bit Modbus standard, a 32-bit register must be read or written as two consecutive 16-bit registers.
The register address mapping follows the normal documented register numbers but the address field, but must be multiplied by two, so to read or write register 3, P_SOLL, use the address 6.

The setup of the Modbus protocol is part of the general setup of the UART.

Register 213, UART1_SETUP, supports the following bit-fields:

Bits	Values	Description
3:0	0=9600, 1=19200, 2=38400, 3=57600, 4=115200, 5=230400, 6=444444, 7=1000000 baud	Basic baud rate in bits per second. For regular passive server operations, any of these rates can be used. For Client operation, the bit rate must be 1Mb (value=7). Note that values 6 and 7 are non-standard baud rates, that are intended to be used between two or more MAC400/800 motors only.
7:4	0=FastMac, 1=Modbus	Protocol to use – select 1 for Modbus
9:8	0=5, 1=6, 2=7, 3=8 data bits	Number of data bits in a byte. Modbus always uses 8 bits per byte.
10	Must be zero	
13:11	0=Even, 1=Odd, 2=Space, 3=Mark, 4 or 5=None 6 or 7=Multidrop	Parity scheme. Modbus should use either Even or Odd parity for maximum error checking. Multidrop is not supported by Modbus.

15:14	0=1, 1=1.5, 2=2 stop bits	Number of stop bits to use.
23:16	0..255	Guard-time. Number of idle bit times between bytes during transmission. These can be seen as additional stop bits. Normally this value is set to zero, but with some UARTs that have trouble synchronizing on long telegrams, this value can be set to non-zero. Setting this value non-zero may help with visually separating bytes on an oscilloscope.
25:24	0=Passive server, 1=Active server with timeout monitoring. 2=Client (bus master) operation to transfer requested position and monitor errors.	For normal operation where a PC or PLC talks to one or more motors, set this to zero. If set to One, the motor will set a Communications Error and stop if it has not received a valid write request to P_SOLL within the timeout selected in bits 31:28. If set to two, the motor will write a position and read the error/status register of the client at address 254 once per sample period. If not both write acknowledge and error/status data is received within the timeout specified in bits 31:28, the motor will set a Communications error and stop.
27:26	Reserved	
31:28	Timeout in milliseconds.	

The firmware will use only the power-up value of register 213, so for any changes to take effect, do a Save in Flash operation.

Read Holding operation:

```
Request: <adr>, 0x03,  RegHi, RegLo, CountHi, CountLo, CRC1, CRC2
Offset:    [0]    [1]    [2]    [3]     [4]      [5]     [6]   [7]
Reply:   <adr>, 0x03, #Bytes, Reg0Hi, Reg0Lo, Reg1Hi, Reg1Lo, ..... CRC1, CRC2
```

Example to read P_IST from motor with address 1, values in decimal:
1, 3, 0, 20, 0, 2, NN, MM (NN and MM are the CRC-16 bytes)

Write Multiple Register operation:

```
Request: <adr>, 0x10,  RegHi, RegLo, CountHi, CountLo, NBytes, Val0Hi, Val0Lo, ..., CRC1, CRC2
Offset:    [0]   [1]    [2]    [3]     [4]      [5]      [6]     [7]     [8]
Reply:   <adr>, 0x10,  RegHi, RegLo, CountHi, CountLo, CRC1,   CRC2
```

Example to write P_SOLL to motor with address 1, values in decimal:
1, 16, 0, 6, 0, 2, 4, bb, aa, dd, cc, NN, MM (NN and MM are the CRC-16 bytes)

This would write a 32-bit hexadecimal value of ddccbbaa – note the byte-packing.