Modbus Basics

Camille Bauer AG
CH-5610 Wohlen

CAMILLE BAUER
Auf uns ist Verlass.

Introduction

The Modbus protocol was originally developed by Modicon (nowadays Schneider Electric) for the data transfer with their controllers. Data transfer was organized in terms of 16-Bit registers (integer format) or as status information in terms of data bytes. Over the years the protocol was extended and has been adopted by other manufacturers as well. New data types were added, especially to achieve a higher resolution for values to transmit. The protocol was adopted for new transfer media, dialects like Modbus Plus or Modbus/TCP arised.

But for compatibility reasons the basic structure of the data area or the addressing mechanism of the protocol retained.

The Modbus protocol is in fact a single master protocol. The master controls the complete transmission and monitors if possible timeouts (no answer from the addressed device) occur. The connected devices are slaves and are allowed to send telegrams only on master request.

The following basics are limited to the protocols Modbus/RTU and Modbus/TCP. Also only functions supported by Modbus devices of the company Camille Bauer are described.

Contents

MODBUS® - Modbus is a registered trade mark of Schneider Electric. Detailed protocol specifications are available via the Website http://www.modbus.org

1. Modbus/RTU protocol

1.1 Transmission mode

Character format:	Normally configurable
	1 start, 8 data, even parity, 1 stop bit
	1 start, 8 data, odd parity, 1 stop bit
	1 start, 8 data, no parity, 2 stop bit
	1 start, 8 data, no parity, 1 stop bit (often used but not in accordance with MODBUS specification)
Baudrate:	Normally configurable, often used values are 1200, 2400, 4800, 9600 and 19200 Bd

1.2 General message form

Device address	Function	Data	CRC check
8 bits	8 bits	n * 8 bits	16 bits

The MODBUS® specification defines a silent-interval (Pause) of at least 3.5 chars between two telegrams to transmit. Within a message two chars may be separated for not more than 1.5 chars. A typical data transmission looks like:

Pause | **Telegram 1** → ***Pause*** | **Telegram 2** → ***Pause*** | **Telegram 3** **etc.**

Note: The monitoring of the given interval times is extremely complicated for the master. In particular Windows operating systems are not suited for such circumstances. Therefore in practice often much longer character intervals are accepted. But this may induce problems during device addressing, because the message framing can get lost. The receiver of the message may misinterpret data to be the beginning of a telegram.

Device Address

The device which has to be accessed (Master→Slave communication) or the responding device (Slave→Master communication). Modbus allows addresses in the range 1..247. The address 0 may be used for broadcast messages to all devices, if the selected function supports this.

Function

Defines the purpose of data transmission. The following standard function are supported by Camille Bauer devices:

Code	MODBUS Function	Register	Application examples...
01_H	READ COIL STATUS	0xxxx	- Reading digital output states
02_H	READ INPUT STATUS	1xxxx	- Reading digital input states
03_H	READ HOLDING REGISTERS	4xxxx	- Reading measurands, meters, mean-values - Reading the device configuration
08_H	DIAGNOSTIC		- Device connection test (subfunction 0)
$0F_H$	FORCE MULTIPLE COILS	0xxxx	- Setting / Simulating digital output states
10_H	PRESET MULTIPLE REGISTERS	4xxxx	- Device configuration

Data

Contains the information to transmit. This field is divided into register, number of registers to transmit and, if necessary, read data or information to store. Data is normally transmitted as a multiple of 16 bit registers.

CRC check

The CRC16 checksum is calculated for all byte of a telegram. It is calculated by the receiver of the message as well to detect possible transmission errors. The CRC calculation is shown in chapter 1.5

1.3 Data types

- Standardized data types are **Byte** (8-Bit) and **Register** (16-Bit). According to the Modbus specification registers are transmitted with the high byte first, followed by the low byte.
- Extended data types: **32-Bit-Integer** and **32-Bit-Float** are transmitted as 2 consecutive 16-Bit registers. **64-Bit-Integer** and **64-Bit-Float** are transmitted as 4 consecutive 16-Bit registers. The format of the float numbers is in accordance with IEEE standard 754. But the transmission sequence of the registers is not fixed. In most applications it works as follows:

32-Bit numbers	*Reg_L (Bit 15..0)*		*Reg_H (Bit 31..16)*					
	HByte	LByte	HByte	LByte				
64-Bit numbers	*Reg_L (15..0)*		*Reg_H (31..16)*		*Reg_L (47..32)*		*Reg_H (63..48)*	
	HByte	LByte	HByte	LByte	HByte	LByte	HByte	LByte
Transmission sequence	1.	2.	3.	4.	5.	6.	7.	8.

1.4 Data addressing

Modbus groups different data types as references. The telegram functions 03_H and 10_H e.g. use register addresses starting at 40001. The reference 4xxxx is implicit, i.e. is given by the used telegram function. In addressing therefore the leading 4 is omitted. The reference is also not given in most Modbus descriptions.

Another speciality in Modbus telegrams is, that the register numeration starts at 1, but the addressing starts at 0. So if e.g. you want to read register 40001 the address in the telegram will be 0. This can also be seen in detail in the telegram examples.

1.5 Cyclic redundancy check calculation (crc16*)* (Example in ‘C)’

The calculation is performed on all message characters, except the check bytes itself. The low-order byte (Crc_LByte) is appended to the message first, followed by the high-order byte (Crc_HByte). The receiver of the message calculates the check bytes again and compares them with the received ones.

```
void main()
{
   unsigned char data[NUMDATA+2];                    // Message buffer
   unsigned char Crc_HByte,LByte;                    //
   unsigned int Crc;
   ....
   Crc=0xFFFF;
   for (i=0; i<NUMDATA; i++) {
     Crc = CRC16 (Crc, data[i] );
   }
   Crc_LByte = (Crc & 0x00FF);                       // Low byte calculation
   Crc_HByte = (Crc & 0xFF00) / 256;                 // High byte calculation
}
// CRC16 calculation
// ----------------
unsigned int CRC16(unsigned int crc, unsigned int data)
{
  const unsigned int Poly16=0xA001;
  unsigned int LSB, i;

  crc = ((crc^data) | 0xFF00) & (crc | 0x00FF);
  for (i=0; i<8; i++) {
    LSB=(crc & 0x0001);
    crc=crc/2;
    if (LSB)
      crc=crc^Poly16;
  }
  return(crc);
}
```

1.6 Error handling

If a transmission error occurs, i.e. if the CRC-16 calculated by the recipient doesn't match the received one, no answer will be sent to the master. This way a timeout will be provoked. The same happens if a non-existing or switched-off device will be addressed.

If the recipient of a message detects another error, it sends back a corresponding error message to the master.

Device answer:

Address	Code	Data	Check sum	
			LByte	HByte
`11`$_H$	`Code+80`$_H$	***`Error code`***	`CRC16`	

The function code received will be sent back. However, the most significant bit (MSB) of the function code will be set. The error code indicates an operating or a programming error. The following error codes are supported:

Error code	Meaning
01_H	The used function code is not supported
02_H	The register address used is not allowed. The register address may be invalid or write-protected.
03_H	Some data values used are out of range, i.e. invalid number of registers.
06_H	Device can not handle the request at the moment. Repeat the request.

1.7 Telegram examples

Function 01_H : READ COIL STATUS

Example: Request the (digital) output states 2 to 11 of device 17. These are 10 states, which can be mapped within 2 data bytes.

Request
Master->Slave

Address	Function	Data				CRC check
		Start address		Number of states		
addr	01_H	High-Byte	Low-Byte	High-Byte	Low-Byte	crc16

Answer
Slave->Master

Address	Function	Data			CRC check
		Number of data bytes	States 9..2	States 11..10	
addr	01_H	8 Bit	8 Bit	8 Bit	crc16

```
Example (Hex):  >>>> 11 01 00 01 00 0A crc_l crc_h
                <<<< 11 01 02 11 01 crc_l crc_h
```

`11`$_H$`=00010001`$_B$`: Output 6,2 ON; Output 9,8,7,5,4,3 OFF`

`01`$_H$`=00000001`$_B$`: Output 10 ON; Output 11 OFF`

Note: Start address 2 is accessed as register 1 in accordance with the MODBUS specification

Function 02_H : READ INPUT STATUS

Example: Request the (digital) input states 4 to 17 of device 17. These are 14 states, which can be mapped within 2 data bytes.

Request
Master->Slave

Address	Function	Data				CRC check
		Start address		Number of states		
addr	02_H	High-Byte	Low-Byte	High-Byte	Low-Byte	crc16

Answer
Slave->Master

Address	Function	Data			CRC check
		Number of data bytes	States 11..4	States 17..12	
addr	02_H	8 Bit	8 Bit	8 Bit	crc16

```
Beispiel (Hex): >>>> 11 02 00 03 00 0D crc_l crc_h
                <<<< 11 02 02 2D 3C crc_l crc_h
                2DH=00101110B: Input 9,7,6,5 ON; Input 11,10,8,4 OFF
                3CH=00111100B: Input 17,16,15,14 ON; Input 13,12 OFF
```

Note: Start address 4 is accessed as register 3 in accordance with the MODBUS specification

Function 03_H : READ HOLDING REGISTERS

Example: Request a float number(32-Bit) on register addresses 108 and 109 of device 17

Request
Master->Slave

Address	Function	Data				CRC check
		Start address		Number of registers		
addr	03_H	High-Byte	Low-Byte	High-Byte	Low-Byte	crc16

Answer
Slave->Master

Address	Function	Data		CRC check
		Number of data bytes	Information	
addr	03_H	n (8 Bit)	n/2 registers	crc16

```
Example (Hex):    >>>> 11 03 00 6B 00 02 crc_l crc_h
                  <<<< 11 03 04 CC CD 42 8D crc_l crc_h
```

Note: Start address 108 is accessed as register 107 in accordance with the MODBUS specification

Function 08_H : DIAGNOSTICS

Example: Using Subfunction 00 (Diagnostic) a test is performed if device 17 is connected. The telegram sent will be sent back 1:1.

Request
Master->Slave

Address	Function	Data				CRC check
		Subfunktion		Data		
addr	08_H	0	0	High-Byte	Low-Byte	crc16

Answer
Slave->Master

Address	Function	Data				CRC check
		Subfunktion		Data		
addr	08_H	0	0	High-Byte	Low-Byte	crc16

```
Example (Hex):   >>>> 11 08 00 00 AA 55 crc_l crc_h
                 <<<< 11 08 00 00 AA 55 crc_l crc_h
```

Function $0F_H$: FORCE MULTIPLE COILS

Example: Set the (digital) output states 30..46 of device 17. These are 17 states, which fit within 3 data bytes.

<u>Request</u>
Master->Slave

Address	Function	Data						CRC check
		Start address		Number of states		Number of bytes	Information	
addr	$0F_H$	High	Low	High	Low	n	n Bytes	crc16

<u>Answer</u>
Slave->Master

Address	Function	Data				CRC check
		Start address		Number of states		
addr	$0F_H$	High	Low	High	Low	crc16

`Beispiel (Hex):  >>>> 11 0F 00 1D 00 11 03` **`AC 38 01`** `crc_l crc_h`

`<<<< 11 0F 00 1D 00 11 crc_l crc_h`

AC$_H$=10101100$_B$: Output 37,35,33,32 ON; Output 36,34,31,30 OFF

38$_H$=00111000$_B$: Output 43,42,41 ON; Output 45,44,40,39,38 OFF

01$_H$=00000001$_B$: Output 46 ON;

Note: Start address 30 is accessed as register 29 in accordance with the MODBUS specification

Function 10_H : PRESET MULTIPLE REGISTERS

Supports Broadcast. Via Address 0 an action may be performed for all devices at the same time. This kind of telegrams is not acknowledged. Typical application: Setting the display brightness of all devices.

Example: Set a long integer number (32-Bit) on register addresses 302 and 303 of device 17.

<u>Request</u>
Master->Slave

Address	Function	Data						CRC check
		Start address		Number of registers		Number of bytes	Information	
addr	10_H	High	Low	High	Low	n	n Bytes	crc16

<u>Answer</u>
Slave->Master

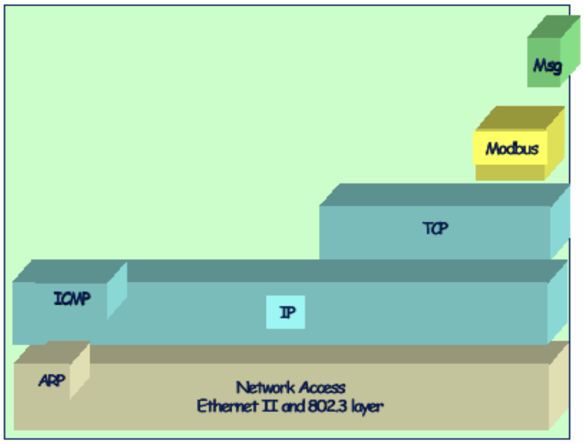
Address	Function	Data				CRC check
		Start address		Number of registers		
addr	10_H	High	Low	High	Low	crc16

`Example (Hex):  >>>> 11 10 01 2D 00 02 04` **`00 0A 01 02`** `crc_l crc_h`

`<<<< 11 10 01 2D 00 02 crc_l crc_h`

Note: Start address 302 is accessed as register 301 in accordance with the MODBUS specification

2. Modbus/TCP protocol



2.1 Generic telegram types

The ADU (Application Data Unit) of the Modbus over TCP/IP protocol is composed of the following parts

MBAP Header	Function code	Data
7 Bytes	1 Byte	n Bytes

MPAP Header (Modbus Application Protocol Header)

Byte 0,1: transaction identifier - Identification number if multiple requests are pending.

Byte 2,3: protocol identifier - always set to 0 (=Modbus protocol)

Byte 4: Number of data bytes following (high byte) - always 0 (because all messages are shorter than 256 bytes)

Byte 5: Number of data bytes following (high byte)

Byte 6: unit identifier (previous ‘device address’). The device is accessed directly via IP address, therefore this parameter has no function and may be set to 0xFF. Exception: If the communication is performed via gateway the device address must be set as before.

Function code

Byte 7: Function code of the standard MODBUS protocol. See chapter 1.2

Data

Byte 8..n: The data area corresponds to the standard MODBUS protocol (see chapter 1). The CRC checksum is no longer necessary because this part is implemented on TCP/IP protocol level.

2.2 Communication management

The Modbus communication requires to establish a TCP connection between a client (e.g. PC) and a server (device). Normally **TCP-Port 502** is used, which is reserved for Modbus communication. However, the user is free to set another port number. A server normally accepts an additional connection via port 502, besides the configured port.

If a firewall is arranged between server and client you have to ensure that the configured TCP port is released.

It is also possible to use a Modbus RTU/TCP gateway as server to which up to 32 devices can be serially connected. This allows to connect Modbus/RTU devices directly to the Ethernet without the need to modify the firmware. However, this cost-effective solution reduces the transmission speed to the baudrate of the serial bus.

2.3 Error handling

If a transmission error occurs, i.e. if the CRC-16 calculated by the recipient doesn't match the received one, no answer will be sent to the master. This way a timeout will be provoked. The same happens if a non-existing or switched-off device will be addressed. If you use an interconnected Modbus RTU/TCP gateway you will receive an error message if the accessed device gives no response.

If the recipient of a message detects another error, it sends back a corresponding error message to the master.

Device answer:

MBAP Header	Function code	Data
Copy of the request	Code+80_H	error code

The function code received will be sent back. However, the most significant bit (MSB) of the function code will be set. The error code indicates an operating or a programming error. The following error codes are supported:

Error code	Meaning
01_H	The used function code is not supported
02_H	The register address used is not allowed. The register address may be invalid or write-protected.
03_H	Some data values used are out of range, i.e. invalid number of registers.
06_H	Device can not handle the request at the moment. Repeat the request.
$0B_H$	Error message of the interconnected gateway: No response of the accessed device.

2.4 Telegram examples

Function 03_H : READ HOLDING REGISTERS

Request: Read a float number (32-Bit) on register addresses 108 and 109 of device 17

Request Client->Server	Transact. identifier		Protocol identifier		Number of Data bytes		unit identifier	Function	Data			
									Start address		Number of registers	
	0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	03_H	High-Byte	Low-Byte	High-Byte	Low-Byte

Answer Server->Client	Transact. identifier		Protocol identifier		Number of Data bytes		unit identifier	Function	Daten	
									Number of data bytes	Information
	0x00	***tno***	0x00	0x00	0x00	n+3	0xFF	03_H	n	n/2 Register

```
Example (Hex)  >>> 00 00 00 00 00 06 FF 03 00 6B 00 02
               <<< 00 00 00 00 00 07 FF 03 04 CC CD 42 8D
```

Note: Start address 108 is accessed as register 107 in accordance with the MODBUS specification. If communication is performed via gateway the unit identifier must be set to the device address (17).

tno = Identifikation number if more than request is pending

Function 08_H : DIAGNOSTICS

Example: Using Subfunction 00 (Diagnostic) a test is performed if device 17 is connected. The telegram sent will be sent back 1:1.

Request
Client->Server

Transact. identifier		Protocol identifier		Number of Data bytes		unit identifier	Function	Data			
								Subfunction		Data	
0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	08_H	0x00	0x00	High-Byte	Low-Byte

Answer
Server->Client

Transact. identifier		Protocol identifier		Number of Data bytes		unit identifier	Function	Data		
								Subfunction	Data	
0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	03_H	n	High-Byte	Low-Byte

```
Example (Hex)  >>> 00 00 00 00 00 06 FF 08 00 00 AA 55
               <<< 00 00 00 00 00 06 FF 08 00 00 AA 55
```

Note: If communication is performed via gateway the unit identifier must be set to the device address (17).

Function 10_H : PRESET MULTIPLE REGISTERS

Example: Set a long integer number (32-Bit) on register addresses 400 and 401 of device 17.

Request
Client->Server

Transact. identifier		Protocol identifier		Number of Data bytes		unit identifier	Function	Data					
								Start addr.		#Reg.		#Bytes	Info
0x00	***tno***	0x00	0x00	0x00	n+7	0xFF	10_H	High	Low	High	Low	n	n Bytes

Answer
Server->Client

Transact. identifier		Protocol identifier		Number of Data bytes		unit identifier	Function	Data			
								Start address		Numb. registers	
0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	10_H	High-Byte	Low-Byte	High-Byte	Low-Byte

Example (Hex) >>> 00 00 00 00 00 0B FF 10 01 8F 00 02 04 **d2 d1 d4 d3**
               <<< 00 00 00 00 00 06 FF 10 01 8F 00 02

Note: Start address 400 is accessed as register 399 in accordance with the MODBUS specification. If communication is performed via gateway the unit identifier must be set to the device address (17).

tno = Identifikation number if more than request is pending