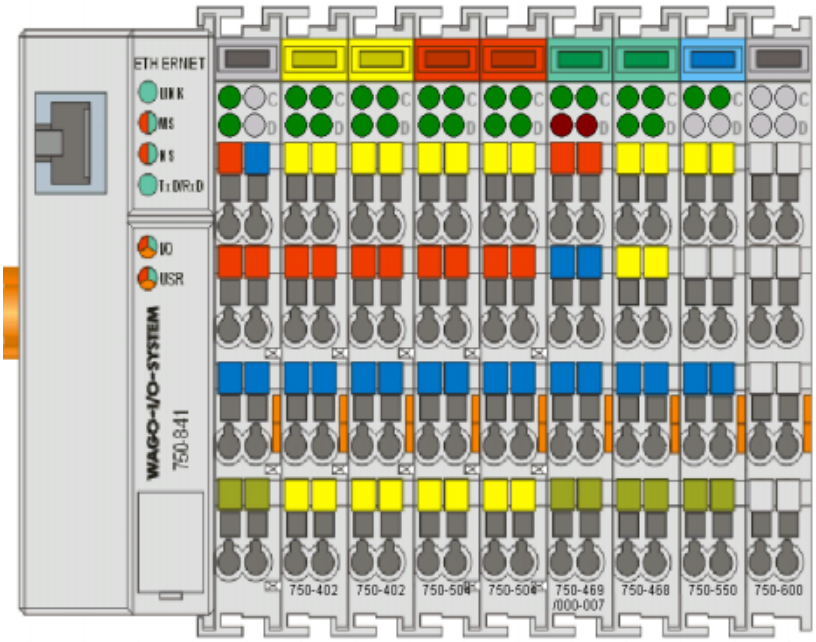
WAGO I/O SYSTEM 750

Configuring WAGO Ethernet with National Instruments LabVIEW via ModbusTCP



Application note

A201602, English
Version 1.1.1

WAGO®
INNOVATIVE CONNECTIONS

WAGO Kontakttechnik GmbH
Hansastraße 27
D-32423 Minden

Phone: +49 (0) 571/8 87 – 0
Fax: +49 (0) 571/8 87 – 1 69

E-Mail: info@wago.com

Web: http://www.wago.com

Technical Support
Phone: +49 (0) 571/8 87 – 5 55
Fax: +49 (0) 571/8 87 – 85 55

E-Mail: support@wago.com

Every conceivable measure has been taken to ensure the correctness and completeness of this documentation. However, as errors can never be fully excluded we would appreciate any information or ideas at any time.

We wish to point out that the software and hardware terms as well as the trademarks of companies used and/or mentioned in the present manual are generally trademark or patent protected.

TABLE OF CONTENTS

1 Important comments

To ensure fast installation and start-up of the units described in this manual, we strongly recommend that the following information and explanation is carefully read and adhered to.

1.1 Legal principles

1.1.1 Copyright

This manual is copyrighted, together with all figures and illustrations contained therein. Any use of this manual which infringes the copyright provisions stipulated herein, is not permitted. Reproduction, translation and electronic and photo-technical archiving and amendments require the written consent of WAGO Kontakttechnik GmbH. Non-observance will entail the right of claims for damages.

1.1.2 Personnel qualification

The use of the product detailed in this manual is exclusively geared to specialists having qualifications in PLC programming, electrical specialists or persons instructed by electrical specialists who are also familiar with the valid standards. WAGO Kontakttechnik GmbH declines all liability resulting from improper action and damage to WAGO products and third party products due to non-observance of the information contained in this manual.

1.1.3 Intended use

For each individual application, the components supplied are to work with a dedicated hardware and software configuration. Modifications are only admitted within the framework of the possibilities documented in the manuals. All other changes to the hardware and/or software and the non-conforming use of the components entail the exclusion of liability on part of WAGO Kontakttechnik GmbH.

Please direct any requirements pertaining to a modified and/or new hardware or software configuration directly to WAGO Kontakttechnik GmbH.

1.2 Range of validity

This application note is based on the stated hardware and software of the specific manufacturer as well as the correspondent documentation. This application note is therefore only valid for the described installation.

New hardware and software versions may need to be handled differently. Please note the detailed description in the specific manuals.

2 Description

The purpose of this document is to provide step-by-step procedures for configuring National Instruments LabVIEW for direct communications with WAGO Ethernet I/O. The procedures that follow illustrate solutions for a direct communication between LabVIEW and the WAGO-I/O without additional protocols or proxis (like OPC, described in appnote “A201601”) using the open standard ModbusTCP.

This document provides two different ways to reach this goal.
1.) Calling the WAGO-Modbus-Driver “MBT.dll” from LabVIEW.
2.) Build your own socket based protocol implementation.

Both ways have its advantages, that mean the comfort supplied by the “MBT.dll” is payed by an increased CPU-usage, but both solutions are up to 20 times faster then an OPC based solution.

Adapting these examples, you need to know on how the buscoupler or controllers build the process image and how to access the process image with the proper Modbus function code and address locations.

When power is applied to the coupler/controller, it automatically detects all connected I/O modules and creates a local process image. This can be analog and digital modules in any order.
The process image is subdivided into inputs data area and outputs data area. The data of the analog modules are mapped first into the process image. The modules are mapped in the order of their position after the coupler/controller. The digital modules are grouped after the analog modules, in the form of words (16 bits per word). When the number of digital I/O.s exceeds 16 bits, the coupler automatically starts another word.

The Modbus/TCP protocol packet is placed in the application layer positioned at level 7 of the OSI model, provides client/server communication between devices using port 502. Modbus use the “big-endian” representation for encoding data items and addresses. Modbus was developed in 1979 and defines the terms as “coils” and “registers”. Coils means digital output, and register could be translated as “word data”.

The procedures in this document have been tested with the following hardware/software configurations:

- Microsoft Operatingsystem NT4(SP6), win2k or XP
- National Instruments LabVIEW 7 and LabVIEW 7.1
- WAGO Modbus-Driver “MBT.dll”
- WAGO 750-x42 16Bit Ethernet TCP/IP Coupler or Controller and
  WAGO 750-x41 32Bit Ethernet TCP/IP Coupler or Controller.
  WAGO 758-870 industrial compact PC

For additional information about WAGO I/O-System visit www.wago.com, contact support@wago.com or call +49/571/887-555

For additional information about LabVIEW visit www.ni.com.

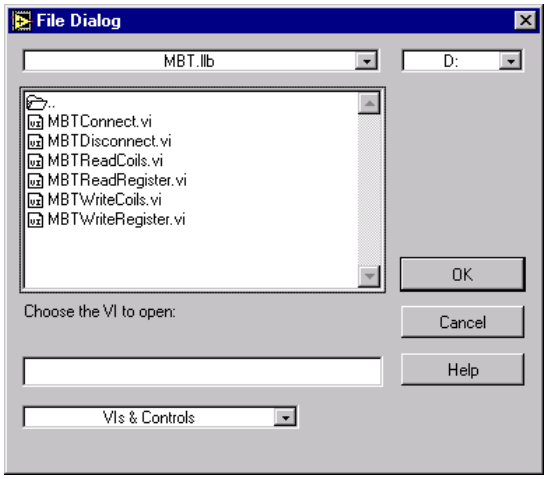
Under www.ethereal.com you find the free network protocol analyser “ethereal”. This software helpful to understand messages on the network.

The Modbus-protocol-specification and more are available on www.modbus.org.

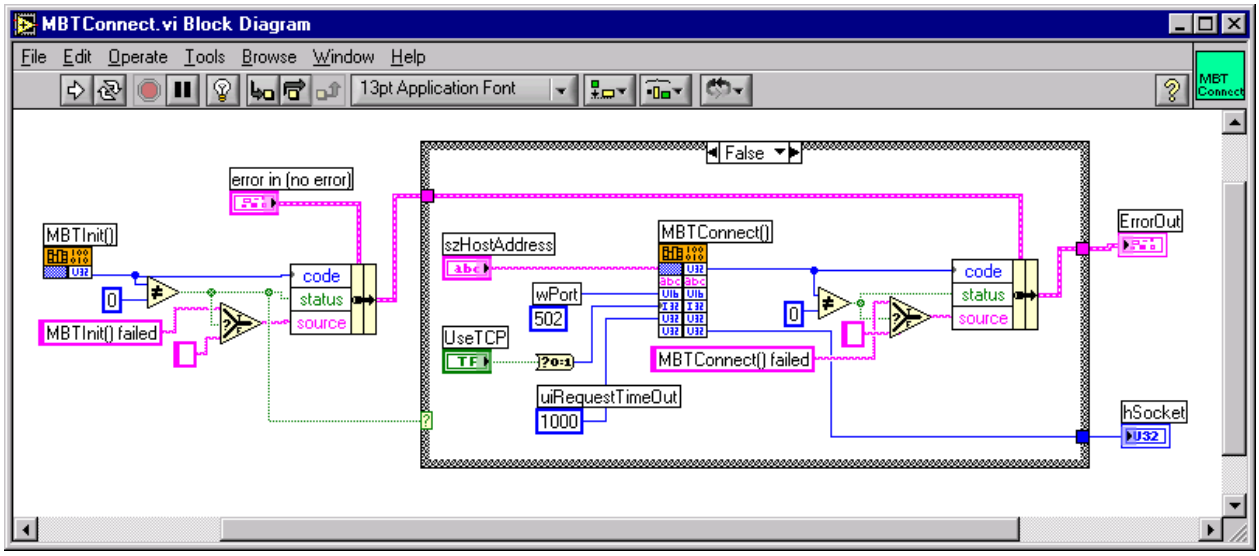
3 Calling the “MBT.dll” from LabVIEW

The communication disscussed in this chapter needs the “MBT.dll” for operation. The easiest way is to copy the DLL in to the windows system folder.

Together with the “MBT.dll”, a LabVIEW VI Library is shipped .
The VI-library “MBT.llb” capsulates the “call library function node” in easys to use subvi’s.



The following image shows the “MBTConnect.vi” block diagram.

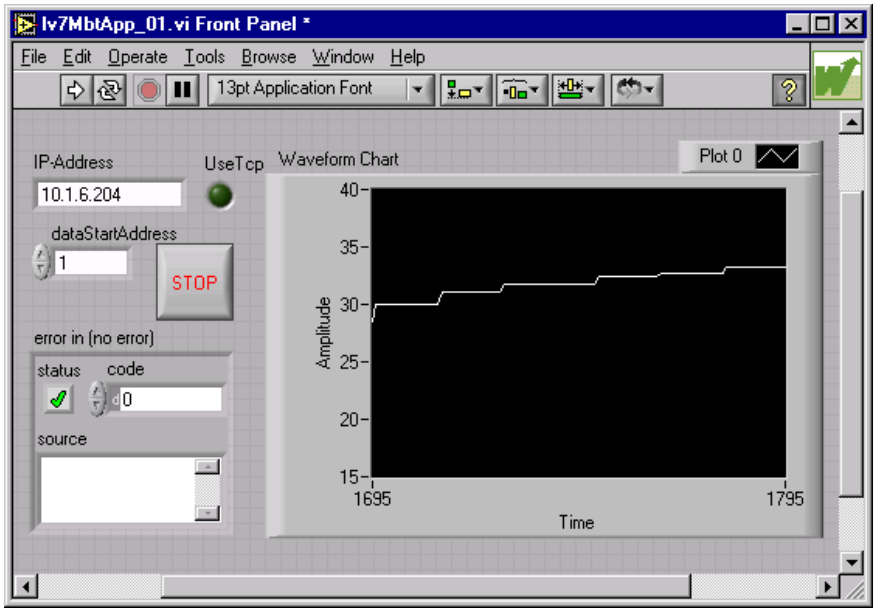


The subvi bundels the calls of the two function MBTInit() and MBTConnect().

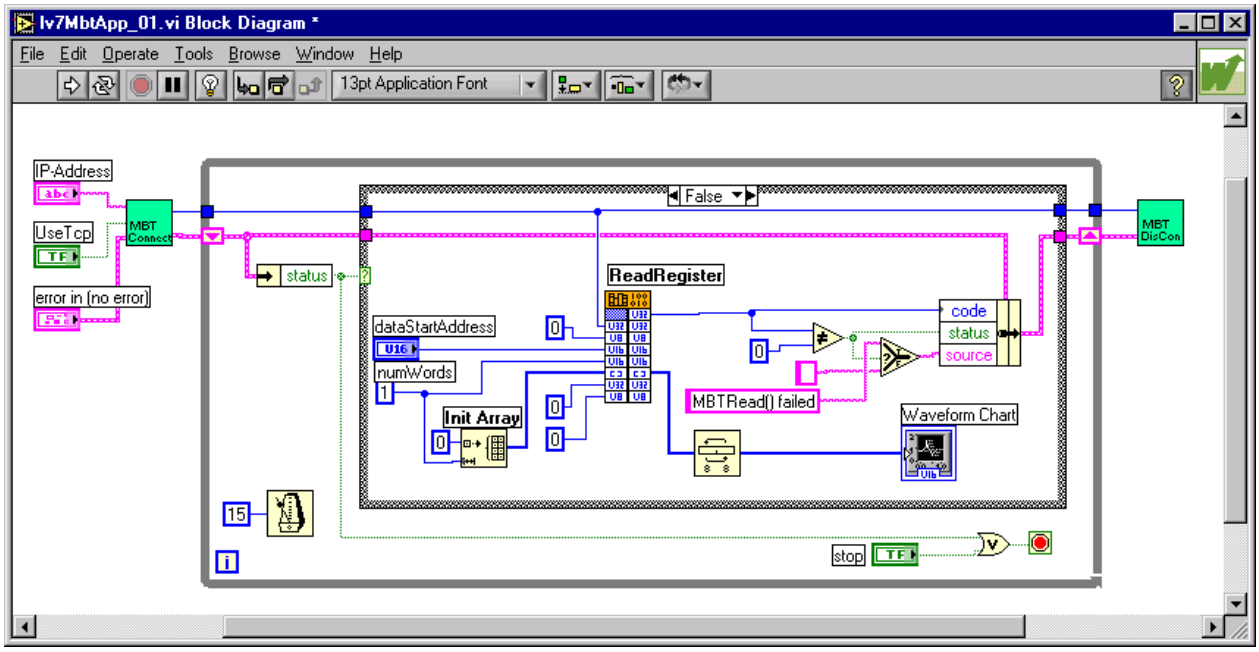
Don’t forget to call “MBTDisconnect.vi” to close the used resources, otherwise the operating system throws an access violation exception error.

3.1 The example “lv7MBTApp_01.vi”

The “lv7MBTApp_01.vi” example is a small temperature writer, the example expects a “750-469” as the first analog input module. The “750-469” is a 2 Channel thermocouples module with diagnostics of sensor types: J, K, B, E, N, R, S, T, U or L.



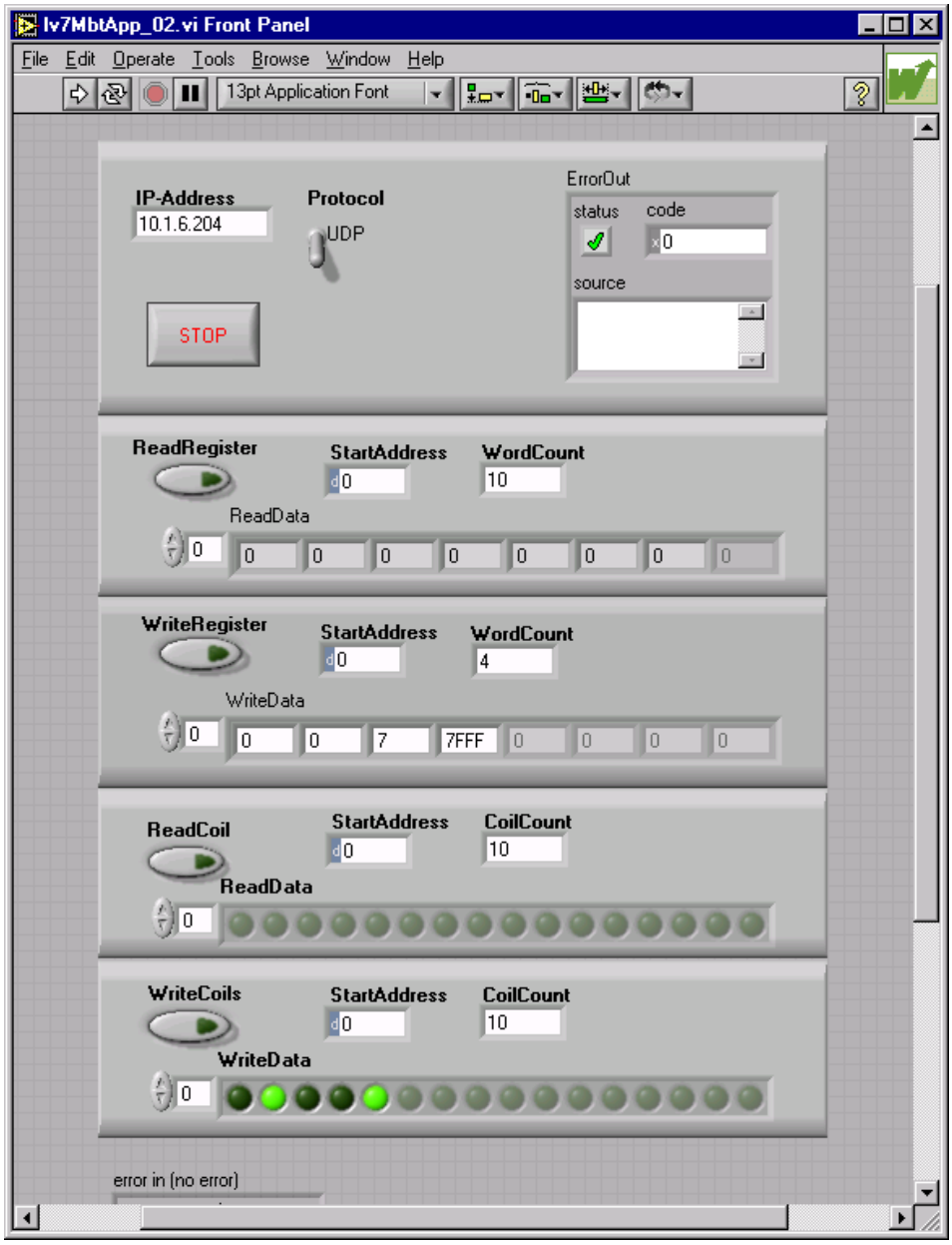
The “lv7MBTApp_01.vi” is a mix of using the VI-lib “MBT.llb” to do direct calls to the dynamic link library “MBT.dll” in one application.



In this example Modbus address 1 cycles through to read one word of data. Address 1 correspond to the second channel of the thermocouples module. WAGO starts at Word Address 0

3.2 The example “lv7MBTApp_02.vi”

The “lv7MBTApp_02.vi” places data in a small modbus monitor, which is able to read and write word and bit data to/from any modbus address.



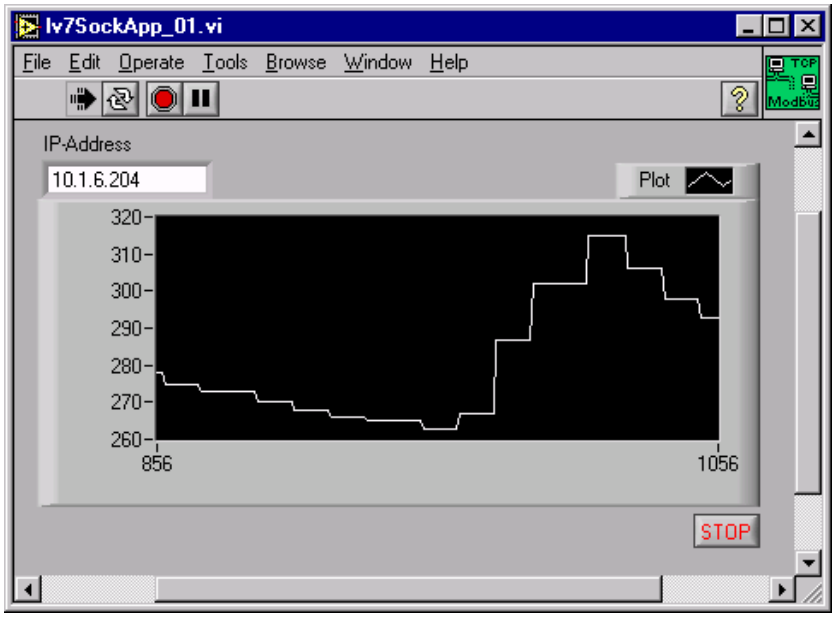
This example shows using all subvis in the “MBT.llb”

4 Build a socket based application

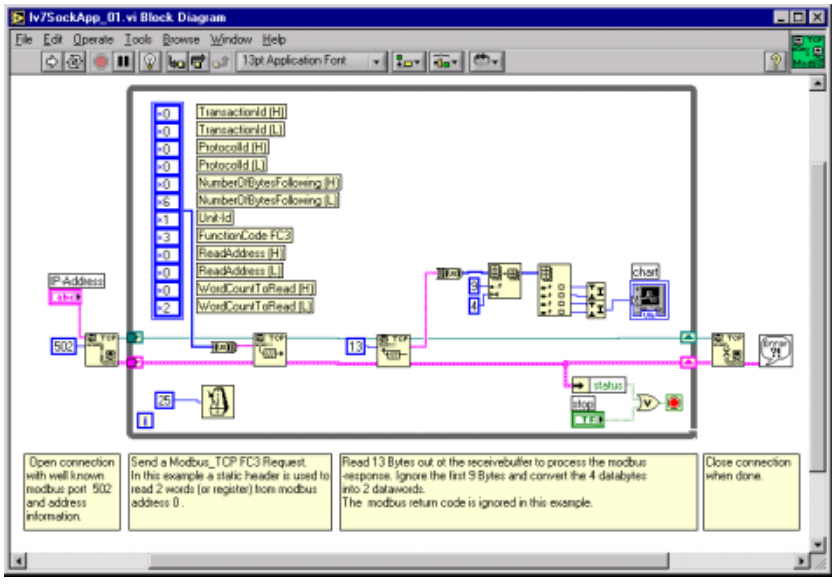
The advantage of your own modbus implementation is the reduced overhead to make it easier to distribute your application because there are no additional libraries needed.

4.1 The example “lv7SockApp_01.vi”

The “lv7SockApp_01.vi” places data in a small temperature writer, the example expects a “750-469” as the first analog input module, but it will work with other configuration.



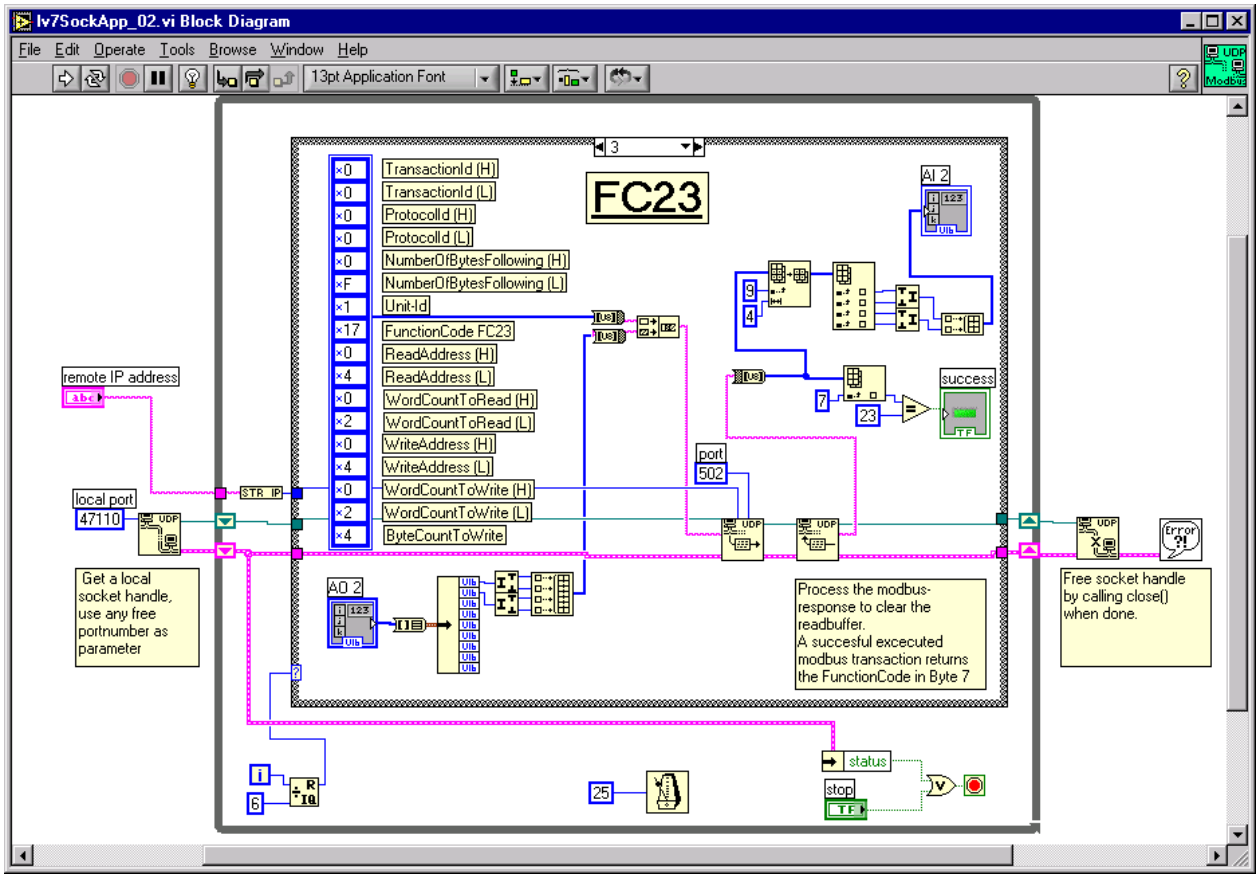
Below you see the hardcoded modbus telegram



Take into account that this example has no modbus error handling.

4.2 The example “lv7SockApp_02.vi”

The “lv7SockApp_02.vi” does not display the classical example character where a small project shows a solution for given task. The “lv7SockApp_02.vi” is more of a code base that could be the starting point for the application.



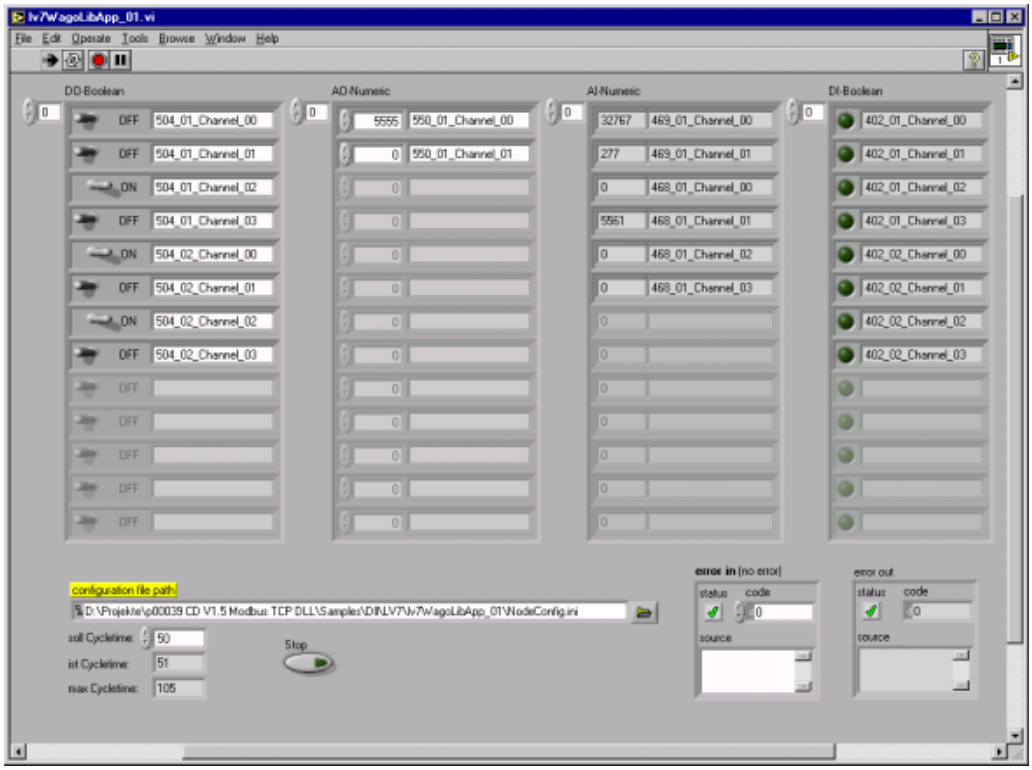
The block diagram shows a hard coded functioncode 23 that reads two (input) words from address 4 and writes two words to the (output) address 4.
The application runs a while loop and executes the following list of function-codes in each loop.

case	function-code	Description
0	--	IDLE – do nothing
1	FC3	Read multiple register
2	FC16	Write multiple registers
3	FC23	Read/Write multiple registers
4	FC2	Read discret inputss
5	FC15	Write coils

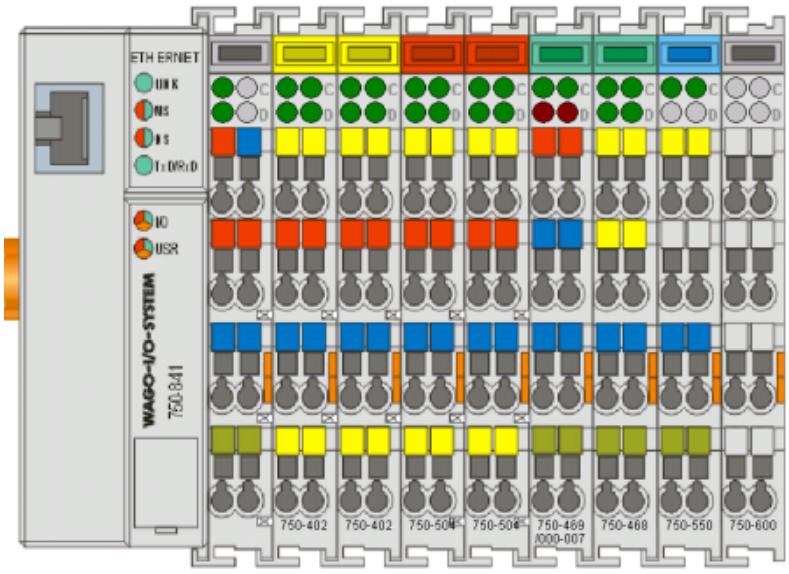
Realized functioncodes in this example

4.3 The application “lv7WagoLibApp_01.vi”

The “lv7WagoLibApp_01.vi” is a more sophisticated example that displays a configurable process image monitor that could be a useful tool.



The configuration is done by an INI-file described below, where the assembled modules described in detail, together with additional information such as channel names and format information to modify the raw process data.



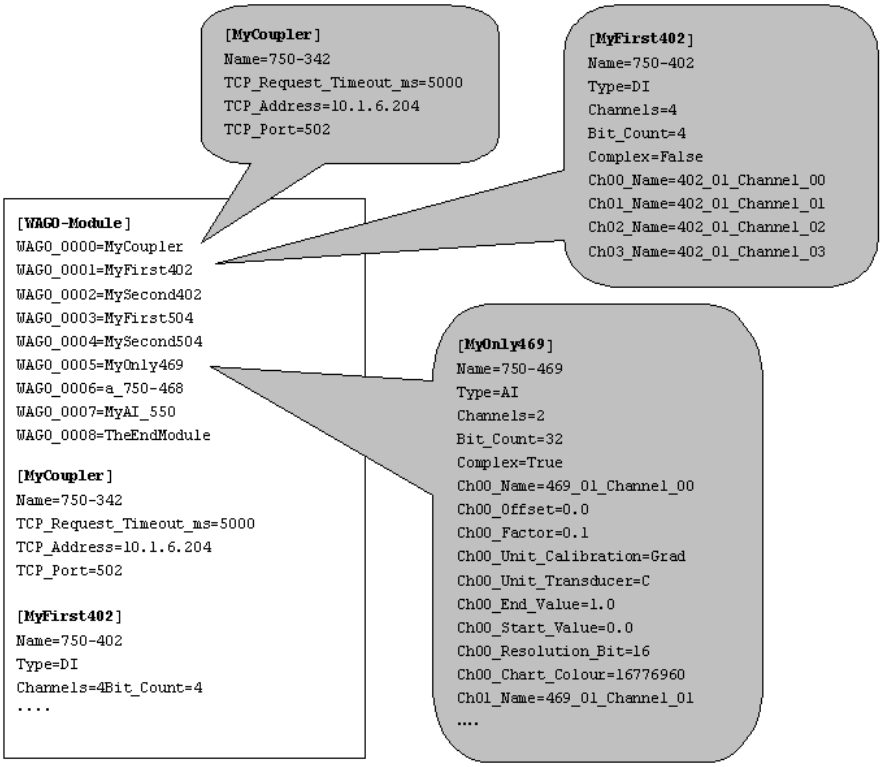
This example expects a “digital_offset” of zero for inputs and outputs.

4.3.1 The INI-file

An INI file is an 8-bit text file divided into sections, each containing zero or more keys. Each key contains zero or more values. Section names are enclosed in square brackets, and must begin at the beginning of a line.

```
[SectionName]
KeyName=value
;Comment
KeyName=Value1, Value2, valueN ;Comment
```

The INI-file of this example can have any name, only the file extension ".ini" is mandatory. The content of the file starts with the section "WAGO-module" followed by a key for each module. The values are section names, where detailed information for each module is defined.



The section "MyCoupler" has a different structure than the rest, this describes the head station and contains the network parameter like Ip-address and port number.

Each module is described by the common key's "Name", "Type", "Channels" and "BitCount" together with additional module dependend keys.
In general, all "name"-values for the headstation, modules and channels are not used for address calculation depending the process image. Names are additional information you can use inside your application.

To work with a module not listet, identify the "Type", "Channels" and "BitCount" for this module and create a new section.

5 Appendix

5.1 The process image

The powered-up controller recognizes all I/O modules connected.

The controller generates an internal local process image from the data width and type of I/O modules, as well as the position of the I/O modules in the node. This image is divided into an input data area and an output data area.

The data of the digital I/O modules are bit-based (i.e., the data exchange is-made by bits). The analog I/O and most specialty modules (e.g., counter modules, encoder modules, and communication modules) are byte-based, in which the data exchange is made by bytes.

The process image is divided into an input data area and an output data area. Each I/O module is assigned a location in the process image, based on the data exchange type (i.e., bit-based or byte-based) and their position after the controller.

All of the byte-based I/O modules are filled in the process image first, then the bit-based modules. The bits of the digital modules are grouped into a word. Once the number of digital I/Os exceeds 16 bits, the controller automatically starts another word.

Changing the physical layout of a node will result in a new structure of the process image. Also, the addresses of the process data will change.
When adding or removing modules, the process data must be verified.

The process image for physical input and output data is stored in the first 256 words of memory (word 0 to 255). This memory actually consists of a separate area for the input and output data.

5.2 The MODBUS TCP DLL “MBT.dll”

The DLL implements the Modbus/TCP Protocol. The Modbus/TCP DLL supports the operating systems Windows NT 4.0 (from version SP5), Windows 2000, Windows 95 (with Windows Socket 2.0 Update) and Windows 98. Internaly the TCP/IP communication uses the Windows Socket 2.0 interface.

The DLL supports synchronous and asynchronous reading and writing of values. TCP or UDP can be selected optionally as a transport protocol. The dll can be used by nearly all programming languages for microsoft operating systems like C, C++, C#, Delphi, LabVIEW, VBA, VB6, vb.net and more.
For an actual list of examples contact support@wago.com or order 759-312 for the “WAGO Modbus TCP DLL CD”.
The complied manual can also be found in the internet.under:
http://www.wago.com/wagoweb/documentation/759/eng_manu/312/m931200e.pdf

Visual Basic and LabVIEW allow only synchronous function calls of the DLL.

This library only supports the commands FC1, FC2, FC3, FC4, FC7, FC15 and FC16 from Open Modbus/TCP protocol V1.0.

All functions of the MBT library have return values corresponding to the HRESULT format. The functions of the socket APIs do not return any return values of this format. The MBT library converts these return values by means of the macro HRESULT_FROM_WIN32. In the following description this is indicated by means of "HR from". The following functions are contained in the Modbus/TCP.DLL:
- MBTInit()
- MBTExit()
- MBTConnect()
- MBTDisconnect()
- MBTReadRegisters()
- MBTReadCoils()
- MBTReadExeptionStatus()
- MBTReadCompleted()
- MBTWriteRegisters()
- MBTWriteCoils()
- MBTWriteCompleted()
- MBTSwapWord()
- MBTSwapDWord()

The DLL was developed with the Microsoft Visual C++ 6.0 development environment. All modules of the DLL are translated as ASCII components with a statically linked C runtime.

5.3 Common MODBUS functions

MODBUS functions from the *OPEN MODBUS / TCP SPECIFICATION* are found in the application layer of the WAGO ETHERNET fieldbus coupler/controller.

These functions allow digital or analog input and output data to be set or directly read out of the fieldbus node.

Function code	hexadecimal	Function	Description
FC1:	0x01	read coils	Reading of several input bits
FC2:	0x02	read input discretes	Reading of several input bits
FC3:	0x03	read multiple registers	Reading of several input registers
FC4:	0x04	read input registers	Reading of several input registers
FC5:	0x05	write coil	Writing of an individual output bit
FC6:	0x06	write single register	Writing of an individual output register
FC7:	0x07	read exception status	Reading of the first 8 input bits
FC11:	0x0B	get comm event counters	Communication event counter
FC15:	0x0F	force multiple coils	Writing of several output bits
FC16:	0x10	write multiple registers	Writing of several output registers
FC23	0x17	read/write multiple registers	Reading and writing of several output registers

Tab. 5-1: List of the MODBUS functions in the fieldbus coupler and controller

To execute a desired function, specify the respective function code and the address of the selected input or output channel.

Attention
The examples listed use the hexadecimal system (i.e.: 0x0000) as their numerical format. Addressing begins with 0.
The format and beginning of the addressing may vary according to the software and the control system. All addresses then need to be converted accordingly.

All MODBUS functions in the WAGO ETHERNET fieldbus coupler and controller are executed as follows:

When a function code is entered, the MODBUS master (i.e. PC) makes a request to the coupler/controller of the fieldbus node.

Subsequently, the coupler/controller sends a datagram to the master as a response.

If the coupler receives an incorrect request, it sends an error datagram (Exception) to the master.

The exception code contained in the exception has the following meaning:

Exception Code	Meaning
0x01	Illegal Function
0x02	Illegal Data Address
0x03	Illegal Data Value
0x04	Slave Device Failure

The following chapters describe the datagram architecture of request, response and exception with examples for each function code.

Note
In the case of the read functions (FC1 – FC 4) the outputs can be additionally written and read back by adding an offset of 200_{hex} (0x0200) to the MODBUS address.

5.4 Use of the MODBUS functions

The graphical overview uses a fieldbus node as an example to show which MODBUS functions can be used to access data of the process image.

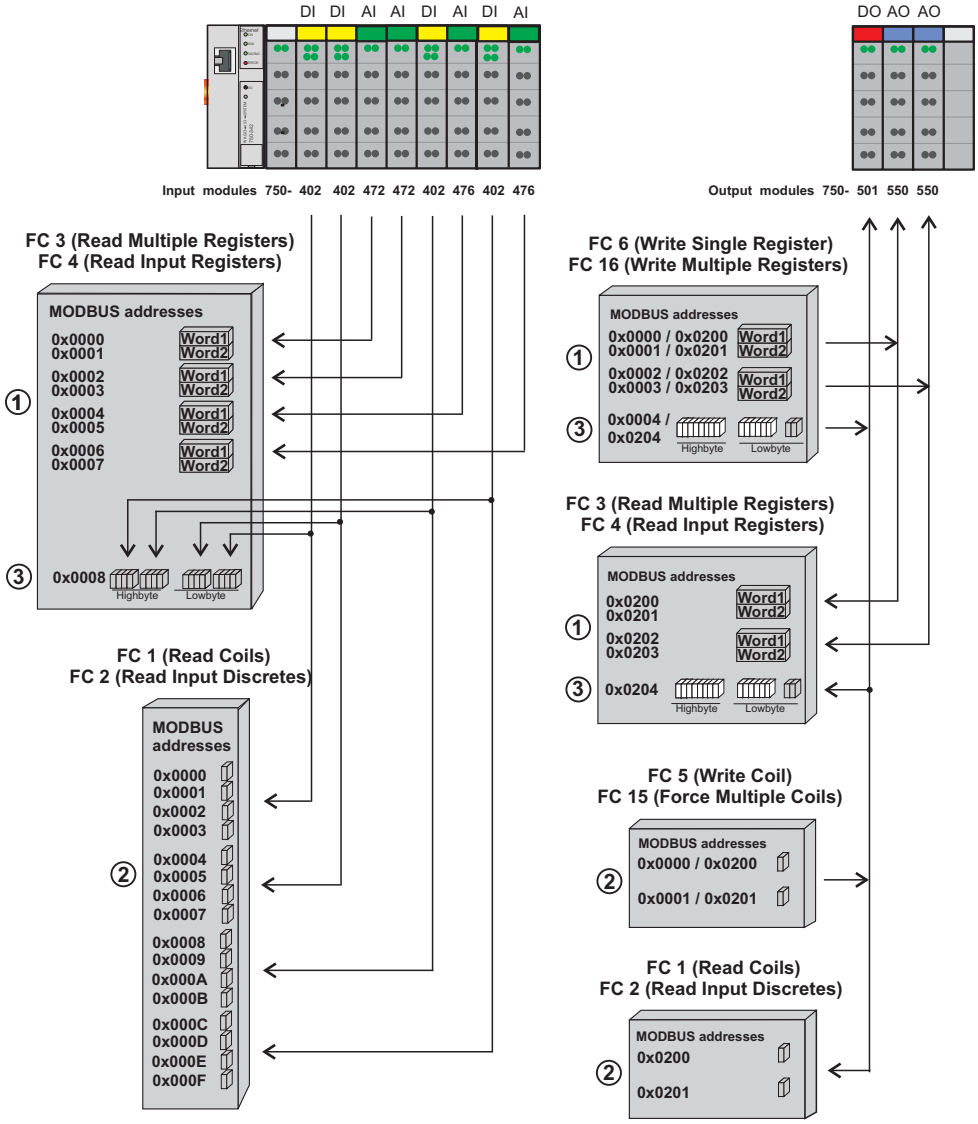


Fig. 5-1: Use of the MODBUS functions G012918e

5.4.1 Function code FC1 (Read Coils)

The function reads the status of the input and output bits (coils) in slave.

Request

The request determines the starting address and the number of bits to be read.
Example: An inquiry, with which the bits 0 to 7 are to be read.

Byte	Field name	Example
Byte 0, 1	Transaction identifier	0x0000
Byte 2, 3	protocol identifier	0x0000
Byte 4, 5	length field	0x0006
Byte 6	unit identifier	0x01 not used
Byte 7	MODBUS function code	0x01
Byte 8, 9	reference number	0x0000
Byte 10, 11	Bit count	0x0008

Response

The current values of the inquired bits are packed in the data field. A 1 corresponds to the ON status and a 0 to the OFF status. The lowest value bit of the first data byte contains the first bit of the inquiry. The others follow in ascending order. If the number of inputs is not a multiple of 8, the remaining bits of the last data byte are filled with zeroes (truncated).

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x01
Byte 8	Byte count	0x01
Byte 9	Bit values	0x12

The status of the inputs 7 to 0 is shown as byte value 0x12 or binary 0001 0010.
Input 7 is the bit having the highest significance of this byte and input 0 the lowest value.
The assignment is thus made from 7 to 0 with OFF-OFF-OFF-ON-OFF-OFF-ON-OFF.

Exception

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x81
Byte 8	Exception code	0x01 or 0x02

5.4.2 Function code FC2 (Read Discrete Inputs)

This function reads the input bits in the slave.

Requests

The request determines the starting address and the number of bits to be read.
Example: An inquiry with which the bits 0 to 7 are to be read:

Byte	Field name	Example
Byte 0, 1	Transaction identifier	0x0000
Byte 2, 3	protocol identifier	0x0000
Byte 4, 5	Length field	0x0006
Byte 6	unit identifier	0x01 not used
Byte 7	MODBUS function code	0x02
Byte 8, 9	reference number	0x0000
Byte 10, 11	Bit count	0x0008

Response

The current value of the inquired bit is packed into the data field. A 1 corresponds to the ON status and a 0 the OFF status. The lowest value bit of the first data byte contains the first bit of the inquiry. The others follow in an ascending order. If the number of inputs is not a multiple of 8, the remaining bits of the last data byte are filled with zeroes (truncated).

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x02
Byte 8	Byte count	0x01
Byte 9	Bit values	0x12

The status of the inputs 7 to 0 is shown as a byte value 0x12 or binary 0001 0010.
Input 7 is the bit having the highest significance of this byte and input 0 the lowest value.
The assignment is thus made from 7 to 0 with OFF-OFF-OFF-ON-OFF-OFF-ON-OFF.

Exception

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x82
Byte 8	Exception code	0x01 or 0x02

5.4.3 Function code FC3 (Read multiple registers)

The binary contents of holding registers are read from the slave using this function.

Request

The request determines the start word address (start register) and the number the register to be read. The addressing starts with 0.
Example: An inquiry of the registers 0 and 1:

Byte	Field name	Example
Byte 0, 1	Transaction identifier	0x0000
Byte 2, 3	protocol identifier	0x0000
Byte 4, 5	length field	0x0006
Byte 6	unit identifier	0x01 not used
Byte 7	MODBUS function code	0x03
Byte 8, 9	reference number	0x0000
Byte 10, 11	Word count	0x0002

Response

The reply register data is packed as 2 bytes per register. The first byte contains the higher value bits, the second the lower values.

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x03
Byte 8	Byte count	0x04
Byte 9, 10	Value Register 0	0x1234
Byte 11, 12	Value Register 1	0x2345

The contents of register 0 are displayed by the value 0x1234 and the contents of register 1 is 0x2345.

Exception

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x83
Byte 8	Exception code	0x01 or 0x02

5.4.4 Function code FC4 (Read input registers)

This function serves to read a number of input words (also ”input register”).

Request

The request determines the address of the start word (start register) and the quantity of the registers to be read. Addressing starts with 0.

Example: An inquiry of the registers 0 and 1:

Byte	Field name	Example
Byte 0, 1	Transaction identifier	0x0000
Byte 2, 3	protocol identifier	0x0000
Byte 4, 5	length field	0x0006
Byte 6	unit identifier	0x01 not used
Byte 7	MODBUS function code	0x04
Byte 8, 9	reference number	0x0000
Byte 10, 11	Word count	0x0002

Response

The register data of the answer is packed as 2 bytes per register. The first byte has the higher value bits, the second the lower values.

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x04
Byte 8	Byte count	0x04
Byte 9, 10	Value Register 0	0x1234
Byte 11, 12	Value Register 1	0x2345

The contents of register 0 are shown by the value 0x1234 and the contents of register 1 is 0x2345.

Exception

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x84
Byte 8	Exception code	0x01 or 0x02

5.4.5 Function code FC15 (Force Multiple Coils)

Using this function a number of output bits are set to 1 or 0. The maximum number is 256 bits.

Request

The first point is addressed with 0.
The inquiry message specifies the bits to be set. The requested 1 or 0 states are determined by the contents of the inquiry data field.

In this example 16 bits are set, starting with the address 0. The inquiry contains 2 bytes with the value 0xA5F0 or 1010 0101 1111 0000 in binary format.

The first byte transmits the 0xA5 to the addresses 7 to 0, whereby 0 is the lowest value bit. The next byte transmits 0xF0 to the addresses 15 to 8, whereby the lowest value bit is 8.

Byte	Field name	Example
Byte 0, 1	Transaction identifier	0x0000
Byte 2, 3	protocol identifier	0x0000
Byte 4, 5	Length field	0x0009
Byte 6	unit identifier	0x01 not used
Byte 7	MODBUS function code	0x0F
Byte 8, 9	reference number	0x0000
Byte 10, 11	Bit Count	0x0010
Byte 12	Byte Count	0x02
Byte 13	Data Byte1	0xA5
Byte 14	Data Byte2	0xF0

Response

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x0F
Byte 8, 9	Reference number	0x0000
Byte 10, 11	Bit Count	0x0010

Exception

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x8F
Byte 8	Exception code	0x01 or 0x02

5.4.6 Function code FC16 (Write multiple registers)

This function writes values in a number of output words (also ”Output register”).

Request

The first point is addressed with 0.

The inquiry message determines the registers to be set. The data is sent as 2 bytes per register.

The example shows how data is set in the two registers 0 and 1:

Byte	Field name	Example
Byte 0, 1	Transaction identifier	0x0000
Byte 2, 3	protocol identifier	0x0000
Byte 4, 5	length field	0x000B
Byte 6	Unit identifier	0x01 not used
Byte 7	MODBUS function code	0x10
Byte 8, 9	reference number	0x0000
Byte 10, 11	Word count	0x0002
Byte 12	Byte Count	0x04
Byte 13, 14	Register Value 1	0x1234
Byte 15, 16	Register Value 2	0x2345

Response

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x10
Byte 8, 9	Reference number	0x0000
Byte 10, 11	Register Value	0x0002

Exception

Byte	Field name	Example
.....		
Byte 7	MODBUS function code	0x85
Byte 8	Exception code	0x01 or 0x02

5.4.7 Function code FC23 (Read/Write multiple registers)

This function reads the register values and writes the values into a number of output words (also ”Output Register“).

Request

The first register is addressed with 0.

The inquiry message determines the registers to be read and set. The data is sent as 2 bytes per register.

Example: The data in register 3 is set to value 0x0123, and values 0x0004 and 0x5678 are read out of the two registers 0 and 1.

Byte	Field name	Example
Byte 0	MODBUS function code	0x17
Byte 1-2	reference number for read	0x0000
Byte 3-4	Word count for read (1-125)	0x0002
Byte 5-6	reference number for write	0x0003
Byte 7-8	Word count for read (1-100)	0x0001
Byte 9	Byte Count (B = 2 x word count for read)	0x02
Byte 10-(B+9)	Register Values	0x0123

Response

Byte	Field name	Example
Byte 0	MODBUS function code	0x17
Byte 1	Byte Count (B = 2 x word count for read)	0x04
Byte 2-(B+1)	Register Values	0x0004 0x5678

Exception

Byte	Field name	Example
Byte 0	MODBUS function code	0x97
Byte 1	Exception code	0x01 or 0x02

WAGO Kontakttechnik GmbH
Postfach 2880 • D-32385 Minden
Hansastraße 27 • D-32423 Minden
Phone: 05 71/8 87 – 0
Telefax: 05 71/8 87 – 1 69
E-Mail: info@wago.com

WAGO Corporation USA
N120W19129 Freistadt Road
PO Box 1015
Germantown, Wi 53022
Phone: 1-262-255-6333
Fax: 1-262-255-3232

Internet: http://www.wago.com

Call Toll Free: 1-800-DIN-RAIL
(346-7245)