TZA 401

Measuring computer
MODBUS Interface description

Operating manual

42/18-58 EN

Rev. 1.0

Contents

Description

Serial communication of the Measuring Computer TZA 401 can be set to two different types of protocols with the help of the TZAKON2 program:

TZAKONFI Protocol

Configuration and tuning of the TZA 401 with the TZAKON2 program.

MODBUS Protocol

Reading of measured data and writing/reading of constants according to the MODBUS protocol specification.

The measuring computers TA 401 always function as “Slaves”, i.e. they react only when the overlying system, the “Master” (e.g. a PC), gives a due command. Supported are only the RTU process of the TZA 401 and only those functions important for the TZA 401.

For more information on the MODBUS protocol see:

GOULD MODICON MODBUS PROTOCOL
Reference Guide
Gould Inc., Programable Control Division
P.O Box 3083
Andover, Massachussetts, 01810
PI-MBUS -300 Rev A, November 93

Interface module

Technical features

The TZA 401 can be configured by way of a front-panel 9-pin D connector for the following interfaces:

RS-232 Interface for parameter tuning using the TZA-KON2 program

RS-232 for connection of **one** TZA 401 to a MODBUS master (e.g. PC)

RS-485 for connecting a maximum of 32 MODBUS subscribers (including master)

The interface module is electrically isolated from the TZA 401. The type of interface is adjusted by way of two plug-in jumpers (see Configuration Instructions 42/18-51 EN).

Interface		Pinout			
Type	Bridges	Signal A	Signal B	Zero	Shield
RS-232 Front	B522, B523	RXD Pin 3	TXD Pin 2	Pin 5	St.-housing
RS-485 Front	B521, B531	+RX/TX Pin 3	−RX/TX Pin 8	Pin 5	St.-housing
RS-485 Blade connector	B521, B531 B506, B501, B3 B506, B501, B3	+RX/TX Pin 26z Kl. 16	−RX/TX Pin 26d Kl. 17	Pin 28z Kl. 18	Pin 28d PE

Tab. 1 Technical features

RS-485 Interface

The interface can alternatively also be routed through the blade-type terminal at the rear to the front panel connector (alternative to the alarm value output GW2) (see Fig. 1). Permissible are max. 32 bus subscribers (including PC). The bus (line structure, no branching, not longer than 1200 m) has stubs (not longer than 30 cm) leading to the respective subscribers.

At least a three-wire, twisted and shielded bus cable for data transmission and an extra isolated cable for potential balancing between the connections “Module zero” of all electrically separated bus subscribers should be used. In order to operate bus subscribers without electrical isolation, an extra separate cable with a big cross-section should be placed parallel to the data cable.

Block diagram of TZA 401 interface

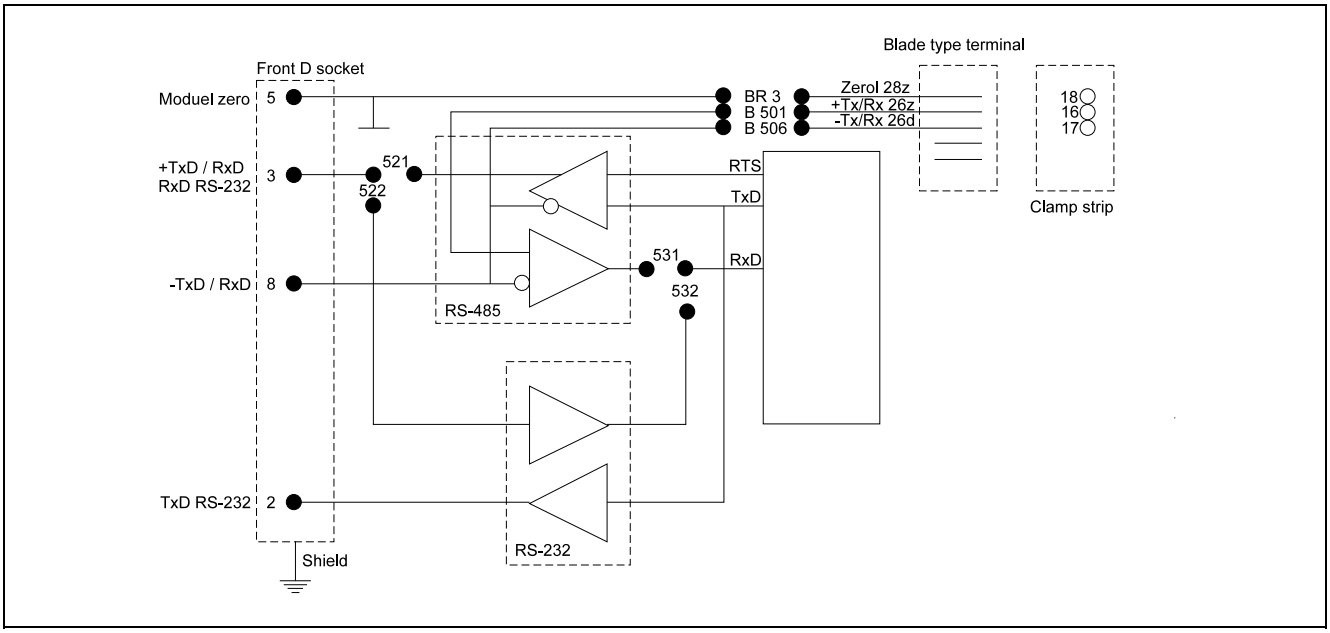

Fig. 1 TZA 401 interface (adjustable: RS-232 or RS-485)
Z-18981 RS-232: Plug-in jumpers B 522, B 532
RS-485: Plug-in jumpers B 521, B 531
RS-485 (MODBUS) Connection to plug receptacle (alternative to alarm value GW2): Soldered jumpers B 506, B 501, BR 3

MODBUS Configuration via RS-485

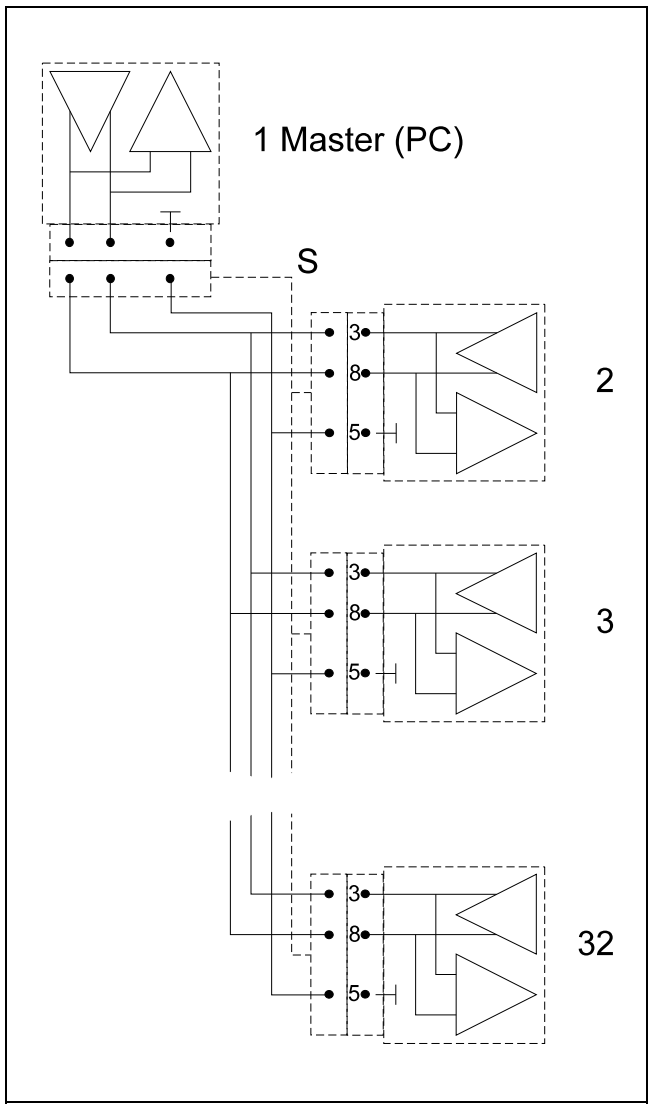

Fig. 2 MODBUS configuration
Z-18982 S Shield

MODBUS Data Transfer

General

Type of Communication

The type of communication (TZAKONFI or MODBUS) is selected in the TZAKON2 program (as of Version 3.2) (Menu item “MODBUS / TZAKONFI COMMUNICATION” of the main menu). The TZA 401 can only be configured or tuned in the TZAKONFI mode. Prior to the changeover, the following conditions must be fulfilled:

- The TZA 401 interface must be set to RS-232 (B 522 and B 532).
- Connection between TZA 401 and PC can only be established with RS232 cable.
- A program must have been loaded onto the measuring computer and started with “Power-On-Start”.
- The program should not execute any PRINT command (Hartmann & Braun’s computation programs inhibit PRINT commands after Power-On-Start).
- In case of calibratable units, the plug-in jumper XB 6 should be removed from the I/O extension card.

When “MODBUS” is selected, a station address from 1...255 must be stated.

MODBUS

Any number of subscribers fulfilling the MODBUS specification can be Modbus participants. The number of subscribers is determined by the type of transmission technique applied:

RS-232 One Master and one Slave.
RS-485 One Master and up to 31 Slaves.

To transfer data, a combination of telegram characters are grouped together to constitute one or several telegrams. These telegrams also take over the “Hand-Shake-function”, whereby every telegram must first be answered from Master to Slave before another one can be sent.

Appropriate monitoring facilities are therefore required in the Master (e.g. PC), in order to keep out non-responsive bus subscribers (“Time-Out-Monitoring”). The Time-out period depends on the type of baud rate applied and the reaction time of the connected subscribers. It should be more than 200 ms to enable smooth communication with the TZA 401.

The TZA 401 operates with a fixed data transmission speed of 9600 baud.

Telegram Characters (Frame)

Telegrams consist of a sequence of I/0 information. The values to be transmitted are apportioned in bytes (= 8 bits). Each of these bytes is supplemented by

1 start bit and
1 stop bit.

There is no parity bit.

In the following description the term “Byte” is used, even if 10 bits are actually transmitted, including the start, stop bits.

Transmission Rules

The quiescent condition (= Pause) of the data cable corresponds to logical “1”.

Before starting the data transmission, at least a time amounting to 3 bytes of the quiescent condition must be available on the data cable.

The space between the bytes of a telegram may not exceed 3 bytes, since a distance of more than 3.5 bytes is interpreted as a separation between two telegrams.

Telegrams

The modbus telegrams have the following structure:

Pause	Address	Function	Data	Checksum	Pause
	1 Byte	1 Byte	n Bytes	2 Bytes	

Permissible station addresses

The figures 1 to 255 are permitted as station addresses of the bus subscribers.

The address 0 is the global address (broadcast address). This address is not accepted by TZA 401 and no acknowledgement is sent to the Master.

CRC Checksum

The checksum is calculated as a total of all the bytes of a telegram (without start, stop bits).

Program examples are provided in the Appendix for calculating the checksum (from page...). For details please refer to the original documentation on MODBUS.

Functions

The TZA 401 supports the following functions:

Code	Name	Function
01H	READ COIL STATUS	Read binary output values and status bits
02H	READ INPUT STATUS	Read the binary input values
03H	READ HOLDING REGISTERS	Read REAL values (IEEE format)
04H	PRESET SINGLE REGISTER	Input unit address
05H	LOOPBACK DIAGNOSTIC TEST	Test telegram for diagnosing the conmmunication cpability of the TZA 401
06H	PRESET MULTIPLE REGISTERS	Input of REAL Floating point constants

Tab. 2 Functions

Function 01

This function is used to query several consecutive binary values from the TZA401. The broadcast address 0 is not permissible.

Example

1. Read telegram from Master to Slave No. 11H

This telegram requests the binary status of the TZA 401 error status register PE_E1 (Register address 22H) up to PE_AX2 (Register address 2FH) (altogether 14 values).

Address	Function	Start address		Quantity		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	01	00	22	00	0E		

(all figures are hexadecimal)

2. Answer telegram from Slave No. 11H to Master

The answer contains binary information packed into a few bytes. The required number of bytes is calculated from: the number of bytes = INT (values / 8) + 1 (in this example: the number of bytes = INT (14 / 8) + 1 = INT (1,75) + 1 = 1 + 1 = 2).

Address	Function	Number of bytes	Status 22H...29H	Status 2AH...2FH	Checksum CRC	
					LByte	Hbyte
11	01	02	8 Bits	8 Bits		

(all figures are hexadecimal)

The number of bytes reveals how many data bytes are going to follow. The example are two bytes:

Data byte No. 1
Status 22H to 29H — Status of the error status register PE_E1 to PE_A1

Bit	7	6	5	4	3	2	1	0
Address	29	28	27	26	25	24	23	22
Reg. PE_	A1	EB	E6	E5	E4	E3	E2	E1
Status	x	x	x	x	x	x	x	x

(all figures are hexadecimal, status x can be 0 or 1)

Data byte No. 2
Status 2AH to 2FH — Status of the binary inputs PE_A2 to PE_AX2

Since only 6 binary pieces of information are transmitted in this byte, bits 6 and 7 are assigned with “0”. Bits 0 to 5 contain the required data.

Bit	7	6	5	4	3	2	1	0
Adresse			2F	2E	2D	2C	2B	2A
Reg. PE_			AX2	AX1	EX2	EX1	AB	A2
Status	0	0	x	x	x	x	x	x

(all firuges are hexadecimal, status x can be 0 or 1)

Function 02

This function is used to query several subsequent binary input values from the TZA 401. The broadcast address 0 is not permissible.

Example

1. Read telegram from Master to Slave No. 11H

This telegram requires the status of the TZA 401 binary inputs EB1 (Register address 0) to EB4 (Register address 3), making 4 values altogether.

Address	Function	Start address		Quantity		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	02	00	00	00	04		

(all figures are hexadecimal)

2. Answer telegramm from Slave No. 11H to Master

In the answer is the binary information packed into a few bytes. The required number of bytes is calculated as follows: the number of bytes = INT (values / 8) + 1 (in this example: the number of bytes = INT (4 / 8) + 1 = INT (0.5) + 1 = 0 + 1 = 1).

Address	Function	Number of bytes	Status 0 to 4	Checksum CRC	
				LByte	Hbyte
11	02	01	8 Bit		

(all figures are hexadecimal)

The number of bytes reveal how many data bytes are to follow.

Data byte
Status 0 to 4 — Status of the binary inputs EB1, EB2, EB3 and EB4

Since only 4 binary pieces of information are transmitted in this byte, bits 4 to 7 are assigned with “0”. Bits 0 to 3 contain the required data.

Bit	7	6	5	4	3	2	1	0
Address					3	2	1	0
Reg. PE_					EB4	EB3	EB2	EB1
Status	0	0	0	0	x	x	x	x

(all figures are hexadecimal, status x can be 0 or 1)

Function 03

This function is used to query a floating point REAL value from the TZA 401. The broadcast address 0 is not permissible.

REAL Values

In the TZA 401 REAL values are stored in BASIC 52 floating point format (6-byte packed BCD format). In order to read these values, a conversion of the TZA 401 communication firmware into the IEEE floating point format takes place.

The IEEE floating point format is composed of 32 bits and always occupies 2 registers (4 bytes see Chapter “Computation of data”). For this reason, 2 registers must always be stated in the column “Quantity”.

Example

Reading of the analog input signal “Sig_E3” from registers 42H and 43H:

Address	Function	Start address		Quantity		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	03	00	42	00	02		

(all figures are hexadecimal)

The answer has the following structure:

Address	Function	Quantity	IEEE floating point value				Checksum CRC	
		Byte	Date [0] Address 42H		Date [1] Address 43H			
11	03	4	HByte	LByte	HByte	LByte	LByte	HByte

(all figures are hexadecimal)

The conversion of the 4 bytes into REAL values is described in the Chapter on “Calculation of data”.

Function 06

With function 6 only the MODBUS unit address can be set or modified in TZA 401. Writing over other registers is not possible. The target address must always be stated with 0000H, the target data contain the new unit address.

Example

Change the unit address from 11H to 12H:

Address	Function	Target address		Target data		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	06	00	00	00	12		

(all figures are hexadecimal)

The response is structured as follows:

Address	Function	Target address		Target data		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	06	00	00	00	12		

(all figures are hexadecimal)

After this answer telegram, the TZA 401 can now be addressed via the unit address 12.

Function 08

This function is used to diagnose communication, whilst the TZA 401 is featuring the “Loopback” test function. The telegram received is sent back immediately as answer telegram by the TZA 401.

Example

This telegram is directed towards bus subscriber No. 11H. The bus subscriber No. 11H interpretes this telegram, checks the CRC checksum and sends the same telegram back to the Master. The diagnostics code must always be 0000. Otherwise the error will be transmitted back. The data can be of any kind.

Address	Function	Diagnosis code		Data		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	08	00	00	A5	37		

(all figures are hexadecimal)

The answer is identical to the query:

Address	Function	Diagnosis code		Data		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	08	00	00	A5	37		

(all figures are hexadecimal)

Error Message Telegram

Every telegramm sent from the Master to the Slave is checked by the Slave for validity. In case of erroneous telegram, except a CRC error, the Slave generates and sends an error telegram to the Master. To enable this, the 7th bit in the answer telegram is set to “high” in the function bit and an error code No. is supplied.

The following errors are recognized by the communication unit:

- wrong function byte — Code 1
- wrong start address — Code 2
- wrong number of values — Code 2
- target data uninterpretable — Code 3
- CRC checksum error — no answer telegram

Example

Read status of the binary inputs EB1 to EB4. A wrong start address has been stated = 0023H.

Address	Function	Start address		Quantity		Checksum CRC	
		HByte	LByte	HByte	LByte	LByte	HByte
11	02	00	23	00	04	x	x

(all figures hexadecimal)

The answer is:

Address	Function	Error code	Checksum CRC	
			LByte	HByte
11	82	02	x	x

(all figures are hexadecimal)

The error bit 7 in the function byte is set!
error code 2: false target address was sent back!

Function 16 (10H)

Function 16 permits REAL constants to be input or modified in the TZA 401. Only one REAL value can always be transferred. In order to transmit one value, 2 registers (4 bytes) are required. Inputs: in the column “Register Quantity” always “2”; in the column “Byte Quantity” always “4”. Storage space is available in TZA 401 for 71 REAL constants. They occupy register addresses 100....240 (64H...0F0H).

Example

Unit address 11H; Register address = 64H; Floating point REAL value = 1.234567 → IEEE format = 3F9E064B

Address	Function	Register address		Register No.		No. of bytes	1. Data word		2. Data word		CRC checksum	
		HByte	LByte	Hbyte	LByte		Mantissa Bit 8...15	Mantissa Bit 0...7	Exponent Bit 24...31	Mantissa Bit 16...23	LByte	HByte
11	10	00	64	00	02	04	06	4B	3F	9E	40	72

(all figures are hexadecimal)

The answer is structured as follows:

Address	Function	Register address		Number of registers		Number of bytes	Checksum CRC	
		HByte	LByte	HByte	LByte		LByte	HByte
11	10	00	64	00	02	04	07	02

(all figures are hexadecimal)

Value Ranges

REAL

− 1,175495 E−38 ... 0 ... 3,402823 E+38
(stored in two registers (= 4 bytes))

BOOL

0 and 1

Calculation of the Data

REAL Values

The BASIC-52 floating point format used in the TZA 401 is converted into the IEEE format. The IEEE format corresponds to the type of format used in PCs.

The MODBUS protocol requires only 16-bit signed integers as transmission values. In order to also transmit floating point figures with the same accuracy, the TZA 401 has been equipped with a feature for transmitting a 32-bit value in IEEE format: the “Pair of Register” method.

The pair of register method is also featured in Hartmann & Braun’s compact automation system Digimatik. It involves transmitting REAL values (4-byte IEEE format) in two subsequent 16-bit registers. Registers with even addresses transmit the low order byte “Word”; registers with odd addresses (even + 1) transmit the high-order byte “Word” (16-Bit-register). If the consistency of the display is to be maintained, both registers must always transmit a 32-bit value in sequence (one after the other).

```
|-------WORD 2-------|-------WORD1-------|

31.30....23.22.....16|15.................0
+----------+----------------------------+
|s| 8 bits |msb23 bit mantisselsb|
+----------+----------------------------+
| ||
| |+------------ Mantissa
| +-------------------- Exponent (7fh)
+------------- Sign bit (0 = Pos, 1 = Neg)
```

The real exponent value is the exponent minus 7Fh for the IEEE 4-byte REAL format.

Depending on the programming language used in the PC, the respective bytes of the REAL values can be accessed directly or indirectly. Computer systems with a different numerical format or which do not permit access to the respective components of the REAL values are stated in the appendix with Program Examples in C and Quick-Basic for the conversion of REAL values into the byte pattern of the IEEE format.

Reading of a Pair of Registers from a TZA 401

A BASIC 52 floating point value is addressed with a telegram of function 03, converted into IEEE format and stored in two registers (4-byte). The answer telegram is compiled and transmitted to the Master. It is in the Master that the two registers of the floating point value in the IEEE format are brought together again.

Regulation for the joining of the two register values each with 16 bits in a (4 byte IEEE) REAL value, whereby data[0] of the read value of the even register and data[1] of odd register contain:

```
float *ptrReal
int data[2]

ptrReal = (float *)&data[0]
```

Assignment of the TZA 401 Variables to the MODBUS Register

The assignment of the respective parameters and dynamic variables to the registers is stated in the following Modbus register tables.

All register numbers are noted in decimals and hexadecimals.

REAL Variables

Register 0 to 96 is reserved for reading REAL variables with the MODBUS function 03. The pair of register method is applied for reading the 32-Bit values. The variables are permanently assigned to the TZA 401 computation programs P731, P735, P751, P761 and P765, and are subdivided into physical and electrical measured and computed values.

REAL Constants

REAL constants can be entered into the TZA 401 with function 10H via the MODBUS and read out with function 03. The Pair of Register method is also applied in this process. Registers 100 (64H) to 240 (0F0H) have been reserved for this purpose. These constants function as parameter data for the TZA 401 and are administered by the respective TZA 401 BASIC program loaded.

BOOLEAN Variables

BOOLEAN variables are divided into four groups:
- Status (high/low) of the binary inputs
- Status (high/low) of the binary outputs
- Error status of the hard and firmware
- Status of the physical Min./Max. values of the calculation programs P731, P735, P751, P761 and P765

Status of Binary Inputs

All binary input states of the TZA 401 can be read out with the MODBUS function 02, see the MODBUS register table of function 02: Input state. In the response telegram, each input state is assigned to a bit in the data byte. See data transmission function 02.

Status of Binary Outputs

With the MODBUS function 01 all binary output states of the TZA 401 can be read out. See the MODBUS register table of function 01: Output status. The addresses 0 to 6 have been reserved for this. In the response telegram, each error status is assigned to a bit in the data byte. See data transmission function 01.

Error Status of Hard and Firmware

With the MODBUS function 01 all error registers of the TZA 401 can be read out. See MODBUS register table of function 01: Error status. The addresses 7 to 29 have been reserved for this. In the response telegram, each error status is assigned to a bit in the data byte. See data transmission function 01.

Status of physical Min./Max. Values

All status register of the Min./Max limit alarms can be read out with the MODBUS function 01. See the MODBUS register table of function 01: Min.-Max.-Status. The addresses 34 to 47 have been reserved for this. In the response telegram, each error status is assigned to a bit in the data byte. See data transmission function 01.

MODBUS Register tables of the TZA 401 Telegram Functions

Register of function 01: “Read output status” High/low level of the binary outputs 0 = Low level, 1 = High level				
Register addresses Dec	Hex	Name	Data type	Description
0	0	ERR	BOOL	Error signal output ERR (basic card)
1	1	AB1	BOOL	Binary output active AB1 5V TTL-Pegel (basic card)
2	2	GW1	BOOL	Alarm value output GW1 (basic card 24 V / 100 mA)
3	3	GW2	BOOL	Alarm value output GW2 (basic card 24 V / 100 mA)
4	4	ABX1	BOOL	Binary output ABX1 (I/O extension 24 V / 440 mA)
5	5	ABX2	BOOL	Binary output ABX2 (I/O extension 24 V / 440 mA)
6	6	ABX3	BOOL	Binary output ABX3 (I/O extension 24 V / 440 mA)

Register of function 02: “Read input status” High/low level of the binary outputs 0 = Low level, 1 = High level				
Register addresses Dec	Hex	Name	Data type	Description
0	0	EB1	BOOL	Binary input EB1 (basic card: 5 V bzw. 24 V signal level)
1	1	EB2	BOOL	Binary input EB2 (basic card: 5 V bzw. 24 V signal level)
2	2	EB3	BOOL	Binary input EB3 (basic card: 5 V bzw. 24 V signal level)
3	3	EB4	BOOL	Binary input EB4 (basic card: 5 V bzw. 24 V signal level)
4	4	ENI	BOOL	Binary input ENI (basic card: NAMUR HF input)
5	5	EBX1	BOOL	Binary input EBX1 (I/O extension card: 24 V signal level, electrically isolated)
6	6	EBX2	BOOL	Binary input EBX2 (I/O extension card: 24 V signal level, electrically isolated)
7	7	EBX3	BOOL	Binary input EBX3 (I/O extension card: 24 V signal level, electrically isolated)
8	8	NN		

Register of function 01: “Read error status” I/O, Hard and firmware error messages 0 = Low level, 1 = High level				
Register addresses Dec	Hex	Name	Data type	Description
7	07	ERR_ALL	BOOL	Global error status, placed when the TZA 401 recognizes an error
8	08	ERR_E1	BOOL	Analog input E1 outside the permissible signal limits
9	09	ERR_E2	BOOL	Analog input E2 outside the permissible signal limits
10	0A	ERR_E3	BOOL	Analog input E3 outside the permissible signal limits
11	0B	ERR_E4	BOOL	Analog input E4 outside the permissible signal limits
12	0C	ERR_E5	BOOL	Analog input E5 outside the permissible signal limits
13	0D	ERR_E6	BOOL	Analog input E6 outside the permissible signal limits
14	0E	ERR_EB	BOOL	Binary einputs EB1, EB4 or ENI: signal frequency outside the permissible limits
15	0F	ERR_A1	BOOL	Analog output A1 outside the permissible signal limits
16	10	ERR_A2	BOOL	Analog output A2 outside the permissible signal limits
17	11	ERR_AB	BOOL	Binary output AB1 outside the permissible limits (not yet available)
18	12	ERR_EX1	BOOL	Analog input EX1 (I/O extension) outside the permissible signal limits
19	13	ERR_EX2	BOOL	Analog input EX2 (I/O extension) outside the permissible signal limits
20	14	ERR_AX1	BOOL	Analog output AX1 (I/O extension) outside the permissible signal limits
21	15	ERR_AX2	BOOL	Analog output AX2 (I/O extension) outside the permissible signal limits
22	16	ERR_ABX	BOOL	Short-circuit or overloading of binary output ABX1, ABX2 or ABX3
23	17	ERR_US1	BOOL	not yet available
24	18	ERR_US2	BOOL	not yet available
25	19	ERR_US3	BOOL	not yet available
26	1A	ERR_US4	BOOL	not yet available
27	1B	ERR_EP1	BOOL	EPROM1 error Checksum error operation system-EPROM
28	1C	ERR_EP2	BOOL	EPROM2 error Checksum error BASIC-EPROM (not yet available)
29	1D	ERR_CNT	BOOL	Redundancy error of the main counter
30	1E	NN		
31	1F	NN		
32	20	NN		
33	21	NN		

Register of function 01: “Read error status” Physical measured value limits overshot!
Calculation programs: P731 and P735
0 = OK,1 = error

Register addresses		Name	Data type	Physical measured value names	
z	Hex			Program P731: Flow, Gas dry/humid	Program P735: Thermal power, Gas dry/humid
34	22	PE_E1	BOOL	p	p
35	23	PE_E2	BOOL	T	T
36	24	PE_E3	BOOL	density	Hu
37	25	PE_E4	BOOL	humidity	humidity
38	26	PE_E5	BOOL	dp1, Qv	dp1, Qv
39	27	PE_E6	BOOL	dp2	dp2
40	28	PE_EB	BOOL	-	-
41	29	PE_A1	BOOL	Q	P
42	2A	PE_A2	BOOL	Q, T, humidity, p	Q, P, T, humidity, p
43	2B	PE_AB	BOOL	-	-
44	2C	PE_EX1	BOOL	T	T
45	2D	PE_EX2	BOOL	-	-
46	2E	PE_AX1	BOOL	Q	Q, P
47	2F	PE_AX2	BOOL	Q, T, humidity, p	Q, P, T, humidity p

Register of function 01: “Read error status” Physical measured value limits overshot!
Calculation programs P751, P761 and P765
0 = OK,1 = error

Register addresses		Name	Data type	Physical measured value names		
Dec	Hex			Program P751: Flow / Thermal power, Water	Program P761: Flow / Thermal power, Heat	Program P765: Flow / Thermal power, Heat minus water
34	22	PE_E1	BOOL	p	p	p
35	23	PE_E2	BOOL	Tw	T	T
36	24	PE_E3	BOOL	Tk	-	Tw
37	25	PE_E4	BOOL	-	-	dpw, Qvw
38	26	PE_E5	BOOL	dp1, Qv	dp1, Qv	dp1, Qvd
39	27	PE_E6	BOOL	dp2	dp2	dp2
40	28	PE_EB	BOOL	-	-	-
41	29	PE_A1	BOOL	Q, P,	Q, P	Qd, Qw, Pd, Pw, dP
42	2A	PE_A2	BOOL	Q, P, Tw, Tk, dT, T	Q, P, T, p	Qd, Qw, Pd, Pw, dP, T, Tw, p
43	2B	PE_AB	BOOL	-	-	-
44	2C	PE_EX1	BOOL	Tw	T	T
45	2D	PE_EX2	BOOL	Tk	-	Tw
46	2E	PE_AX1	BOOL	Q, P, Tw, Tk, dT, T	Q, P, T, p	Qd, Qw, Pd, Pw, dP, T, Tw, p
47	2F	PE_AX2	BOOL	Q, P, Tw, Tk, dT, T	Q, P, T, p	Qd, Qw, Pd, Pw, dP, T, Tw, p

Register of function 03: “Read Floating point values” Physical measured values
Calculation programs P731 and P 735

Register addresses Dec	Hex	Name	Data type	Physical variables (Dimensions according to parameter definition) Program P731: Flow, Gas dry/humid	Program P735: Thermal power, Gas dry/humid
0	00	Count1	REAL	-	P
2	02	Count2	REAL	Q	Q
4	04	W1	REAL	-	-
6	06	W2	REAL	-	-
8	08	W3	REAL	-	-
10	0A	W4	REAL	-	-
12	0C	W5	REAL	-	-
14	0E	W6	REAL	-	-
16	10	W7	REAL	-	-
18	12	W8	REAL	-	-
20	14	W9	REAL	-	-
22	16	W10	REAL	-	-
24	18	W11	REAL	-	-
26	1A	W12	REAL	-	-
28	1C	W13	REAL	-	-
30	1E	W14	REAL	-	-
32	20	Phy_E1	REAL	p	p
34	22	Phy_E2	REAL	T	T
36	24	Phy_E3	REAL	Norm-density	Hu
38	26	Phy_E4	REAL	humidity	humidity
40	28	Phy_E5	REAL	dp1, Qv	dp1, Qv
42	2A	Phy_E6	REAL	dp2	dp2
44	2C	Phy_EB	REAL	f(Qv)	f(Qv)
46	2E	Phy_A1	REAL	Q	P, Q
48	30	Phy_A2	REAL	Q, T, humidity, p	P, Q, T, humidity, p
50	32	Phy_AB	REAL	-	-
52	34	Phy_EX1	REAL	T	T
54	36	Phy_EX2	REAL	-	-
56	38	Phy_AX1	REAL	Q	P, Q
58	3A	Phy_AX2	REAL	Q, T, humidity, p	P, Q, T, humidity, p
60	3C	Phy_ABX	REAL	-	-

Register of function 03: “Read Floating point values” Input/output signal measured values
Programs P731 and P735

Register addresses Dec	Hex	Name	Data type	Signal measuring variable mA, V or Ω depending on the parameter definition Program P731: Flow, Gas dry/humid	Program P735: thermal power, Gas dry/humid
62	3E	Sig_E1	REAL	p	p
64	40	Sig_E2	REAL	T	T
66	42	Sig_E3	REAL	Norm density	Hu
68	44	Sig_E4	REAL	humidity	humidity
70	46	Sig_E5	REAL	dp1, Qv	dp1, Qv
72	48	Sig_E6	REAL	dp2	dp2
74	4A	Sig_EB	REAL	f(Qv)	f(Qv)
76	4C	Sig_A1	REAL	Q	P, Q
78	4E	Sig_A2	REAL	Q, T, humidity, p	P, Q, T, humidity, p
80	50	Sig_AB	REAL	-	-
82	52	Sig_EX1	REAL	T	T
84	54	Sig_EX2	REAL	-	-
86	56	Sig_AX1	REAL	Q	P, Q
88	58	Sig_AX2	REAL	Q, T, humidity, p	P, Q, T, humidity, p
90	5A	Sig_ABX	REAL		
92	5C	NN			
94	5E	NN			
96	60	NN			

Register of function 03: “Read floating point values” Physical measured values
Programs P751, P761 and P765

Register addresses Dec	Hex	Name	Data type	Physical measured values (Dimensions according to para. definition) Program P751: Flow / Thermal power, Water	Program P761: Flow / Thermal power, Heat	Program P765 Flow / Thermal power, Heat minus water
0	00	Count1	REAL	P	P	P
2	02	Count2	REAL	Q	Q	Q
4	04	W1	REAL	H	H	Rho_d
6	06	W2	REAL	Rho	Rho	Rho_w
8	08	W3	REAL	Hw	-	Hd
10	0A	W4	REAL	Hk	-	Hw
12	0C	W5	REAL	-	-	-
14	0E	W6	REAL	-	-	-
16	10	W7	REAL	-	-	-
18	12	W8	REAL	-	-	-
20	14	W9	REAL	-	-	-
22	16	W10	REAL	-	-	-
24	18	W11	REAL	-	-	-
26	1A	W12	REAL	-	-	-
28	1C	W13	REAL	-	-	-
30	1E	W14	REAL	-	-	-
32	20	Phy_E1	REAL	p	p	p
34	22	Phy_E2	REAL	Tw, T	T	T
36	24	Phy_E3	REAL	Tk	-	Tw
38	26	Phy_E4	REAL	-	-	dpw;Qvw
40	28	Phy_E5	REAL	dp1, Qv	dp1, Qv	dp1,Qvd
42	2A	Phy_E6	REAL	dp2	dp2	dp2
44	2C	Phy_EB	REAL	f(Qv)	f(Qv)	f(Qv)
46	2E	Phy_A1	REAL	P, Q	P, Q	Pd, Pw, dP, Qd, Qw
48	30	Phy_A2	REAL	P, Q, , Tw, Tk, dT, T	P, Q, T, p	Pd, Pw, dP, Qd, Qw, T, Tw, p
50	32	Phy_AB	REAL	-	-	-
52	34	Phy_EX1	REAL	Tw	T	T
54	36	Phy_EX2	REAL	Tk	-	Tw
56	38	Phy_AX1	REAL	P, Q, , Tw, Tk, dT, T	P, Q, T, p	Pd, Pw, dP, Qd, Qw, T, Tw, p
58	3A	Phy_AX2	REAL	P, Q, , Tw, Tk, dT, T	P, Q, T, p	Pd, Pw, dP, Qd, Qw, T, Tw, p
60	3C	Phy_ABX	REAL	-	-	-

Register der function 03: “Read Floating-Point-Values” Input/Output signal measured values
programs P751, P761 and P765

Register addresses Dec	Hex	Name	Data type	Signal variables in mA, V or Ω depending on param. definition Program P751: Flow / Thermal power, Water	Program P761: Flow / Thermal power, Heat	Program P765: Flow / Thermal power, Heat minus water
62	3E	Sig_E1	REAL	p	p	p
64	40	Sig_E2	REAL	Tw	T	T
66	42	Sig_E3	REAL	Tk	-	Tw
68	44	Sig_E4	REAL	-	-	dpw;Qvw
70	46	Sig_E5	REAL	dp1, Qv	dp1, Qv	dp1,Qvd
72	48	Sig_E6	REAL	dp2	dp2	dp2
74	4A	Sig_EB	REAL	f(Qv)	f(Qv)	f(Qv)
76	4C	Sig_A1	REAL	P, Q	P, Q	Pd, Pw, dP, Qd, Qw
78	4E	Sig_A2	REAL	P, Q, ,Tw, Tk, dT, T	P, Q, T, p	Pd, Pw, dP, Qd, Qw, T, Tw, p
80	50	Sig_AB	REAL	-	-	-
82	52	Sig_EX1	REAL	Tw	T	T
84	54	Sig_EX2	REAL	Tk	-	Tw
86	56	Sig_AX1	REAL	P, Q, , Tw, Tk, dT, T	P, Q, T, p	Pd, Pw, dP, Qd, Qw, T, Tw, p
88	58	Sig_AX2	REAL	P, Q, , Tw, Tk, dT, T	P, Q, T, p	Pd, Pw, dP, Qd, Qw, T, Tw, p
90	5A	Sig_ABX	REAL			
92	5C	NN				
94	5E	NN				
96	60	NN				

Register of function 16 (10H): “Write Floating Point Constants”					
Register addresses		Name	Data type	TZA 401 - RAM	Names of constants are dependent on the BASIC program and can be freely defined
Dec	Hex			Memory address	
100	0064H	Konst01	REAL	6400H	
102	0066H	Konst02	REAL	6406H	
104	0068H	Konst03	REAL	640CH	
106	006AH	Konst04	REAL	6412H	
108	006CH	Konst05	REAL	6418H	
110	006EH	Konst06	REAL	641EH	
112	0070H	Konst07	REAL	6424H	
114	0072H	Konst08	REAL	642AH	
116	0074H	Konst09	REAL	6430H	
118	0076H	Konst10	REAL	6436H	
120	0078H	Konst11	REAL	643CH	
122	007AH	Konst12	REAL	6442H	
124	007CH	Konst13	REAL	6448H	
126	007EH	Konst14	REAL	644EH	
128	0080H	Konst15	REAL	6454H	
130	0082H	Konst16	REAL	645AH	
132	0084H	Konst17	REAL	6460H	
134	0086H	Konst18	REAL	6466H	
136	0088H	Konst19	REAL	646CH	
138	008AH	Konst20	REAL	6472H	
140	008CH	Konst21	REAL	6478H	
142	008EH	Konst22	REAL	647EH	
144	0090H	Konst23	REAL	6484H	
146	0092H	Konst24	REAL	648AH	
148	0094H	Konst25	REAL	6490H	
150	0096H	Konst26	REAL	6496H	
152	0098H	Konst27	REAL	649CH	
154	009AH	Konst28	REAL	64A2H	
156	009CH	Konst29	REAL	64A8H	
158	009EH	Konst30	REAL	64AEH	
160	00A0H	Konst31	REAL	64B4H	
162	00A2H	Konst32	REAL	64BAH	
164	00A4H	Konst33	REAL	64B0H	
166	00A6H	Konst34	REAL	64C6H	
168	00A8H	Konst35	REAL	64CCH	
170	00AAH	Konst36	REAL	64D2H	
172	00ACH	Konst37	REAL	64D8H	
174	00AEH	Konst38	REAL	64DEH	
176	00B0H	Konst39	REAL	64E4H	
178	00B2H	Konst40	REAL	64EAH	
180	00B4H	Konst41	REAL	64F0H	
182	00B6H	Konst42	REAL	64F6H	
184	00B8H	Konst43	REAL	64FCH	
186	00BAH	Konst44	REAL	6502H	
188	00BCH	Konst45	REAL	6508H	
190	00BEH	Konst46	REAL	650EH	
192	00C0H	Konst47	REAL	6514H	
194	00C2H	Konst48	REAL	651AH	
196	00C4H	Konst49	REAL	6520H	
198	00C6H	Konst50	REAL	6526H	
200	00C8H	Konst51	REAL	652CH	
202	00CAH	Konst52	REAL	6532H	
204	00CCH	Konst53	REAL	6538H	
206	00CEH	Konst54	REAL	653EH	
208	00D0H	Konst55	REAL	6544H	
210	00D2H	Konst56	REAL	654AH	
212	00D4H	Konst57	REAL	6550H	
214	00D6H	Konst58	REAL	6556H	
216	00D8H	Konst59	REAL	655CH	
218	00DAH	Konst60	REAL	6562H	
220	00DCH	Konst61	REAL	6568H	
222	00DEH	Konst62	REAL	656EH	
224	00D0H	Konst63	REAL	6574H	
226	00D2H	Konst64	REAL	657AH	
228	00E4H	Konst65	REAL	6580H	
230	00E6H	Konst66	REAL	6486H	
232	00E8H	Konst67	REAL	648CH	
234	00EAH	Konst68	REAL	6592H	
236	00ECH	Konst69	REAL	6598H	
238	00EEH	Konst70	REAL	659EH	
240	00F0H	Konst71	REAL	65A4H	

Program Examples in C

All subsequent examples for access to register are in C, in order to obtain an error-free example. The transmission takes place here in the RTU protocol. The functions *modbus_read()* and *modbus_write()* show how a telegram is geneated, all other functions show how to deal with various data formats.

Modbus_read

Read data from other Modbus subscribers (Read output register: Function 03)

```
void modbus_read(unsigned regnr, int anzahl, int *recdata)
{
inti,anz;
unsigned crc;

sendbuf[0] = mod_adr; /* Modbus target address */
sendbuf[1] = 3; /* function “Read real value from TZA 401” */
sendbuf[2] = regnr>>8;/* Hi register address*/
sendbuf[3] = regnr; /* Lo register address*/
sendbuf[4] = 0; /* Hi numberof registers (for TZA 401 always 0)*/
sendbuf[5] = 2/* Lo number of register for TZA 401 always 2)*/
crc= CRC16(sendbuf,6);
sendbuf[6] = crc;
sendbuf[7] = crc>>8;

ComWrite(sendbuf,8);/* transmit 8 characters */
ComRead(receivebuf);/* receive data */

// receivebuf[0]; contains the Modbus target address
// receivebuf[1]; contains the functions code

anz = receivebuf[2];/*Number of data bytes (for TZA 401 always 4) */

// receivebuf[3+anz]; contains AElt address CRC
// receivebuf[4+anz]; contains AElt address CRC

for (i=0; i < anz; i+=2) {
recdata[i+0] = receivebuf[4+i];
recdata[i+1] = receivebuf[3+i];
}
}
```

Modbus_write

Send REAL data to other Modbus subscribers (write constants in TZA 401: Function 16)

```
void modbus_write(unsigned regnr, int data[0], int data[1])
{
 unsigned crc;

sendbuf[00] = Stat_adr        ;     /* Modbus Station address*/
sendbuf[01] = 16        ;     /* function write Real-Wert */
sendbuf[02] = regnr>>8              ;     /* Hi register number*/
sendbuf[03] = regnr     ;     /* Lo register number */
sendbuf[04] = 00        ;     /* Hi register number (for TZA 401 always 0) */
sendbuf[05] = 02        ;     /* Lo register number (for TZA 401 always 2) */
sendbuf[06] = 04        ;     /* Number of bytes (for TZA 401 always 4)*/
sendbuf[07] = data[0]>>8      ;     /* Load Hi byte register 0 */
sendbuf[08] = data[0]   ;     /* Load Lo byte register 0 */
sendbuf[09] = data[1]>>8      ;     /* Load Hi byte register 1 */
sendbuf[10] = data[1]   ;     /* Load Lo byte register1 */
crc = CRC16(sendbuf,11);      /* Generate CRC check for 11 bytes */
sendbuf[11] = crc       ;     /* Lo Byte CRC load */
sendbuf[12] = crc>>8    ;     /* Hi Byte CRC load */
comWrite(sendbuf,13)          ;     /* transfer 13 character */
comRead(receivebuf)           ;     /* receive acknowledgement */

}
```

Programming Example for determining the CRC Sum of the Modbus RTU Telegram

```
unsigned short CRC16(
void *data_p/* Data range */,
unsigned len/* Data length */
)
/* let data_p calculate 16 Bit CRC (Modbus-RTU)  */
{

# define POLYNOM              0x0A001

int i,j;
unsigned shortcrc = 0xffff;
unsigned char *p = data_p;

for (j=0; j < len; j++) {          /* for entire buffer */
for (crc ^=*p++,i=0;i < 8; i++) {/* for one byte */
if ((crc & 0x0001))
crc = (crc >> 1) ^ POLYNOM;
else
crc >>= 1;
}
}
return (crc);
}
```

Count “Thermal Power P” (Program P761) Determine with Pair of Register

(W, Register 0..1)

```
void read_float_split_merge()
{
float *fval;
int recdata[30];

modbus_read(0, 2, &recdata[0]);
fval = (void *)&recdata[0];
printf("Float-Register 0/1 :float =%6.3f", *fval);
}
```

```
 Determine signal value dp1 input (program P761) with pair of register

(dp1, Register 74..75)

void read_float_split_merge()
{
float *fval;
int recdata[30];

modbus_read(74, 2, &recdata[0]);
fval = (void *)&recdata[0];
printf("Float-Register 74/75 :float =%6.3f", *fval);
}
```

Write REAL Constant1 into Register Address 100

(Register 100/101)

```
void write_float_split_merge()
{
float value;
unsigned long*pdata;
int data[2];

wert= 123.4567;
pdata = (void *)&wert;
data[0] = (unsigned) (*pdata & 0xFFFF);
data[1] = (unsigned) (*pdata >>16);
modbus_write(100,data[0],data[1]);
}
```

Program Examples in Quickbasic 4.5

IEEE Value Calculation by Means of MKS$ and the CSV Function

```
’Demo program for editing the IEEE value display
’in Quick-Basic 4.5
’uses the Quick-Basic functions MKS$ and CVS
’-----------------------------------------------------
DECLARE FUNCTION BIN.AER$ (z$)
DECLARE FUNCTION HEX2$ (x)
CLS
DO
INPUT "Real value (E = End) "; real value$
IF UCASE$(Real value$) = "E" THEN END
Real value! = VAL(Real value$)
’-----------------------------------------------------
’Condition:
’-----------------------------------------------------
’ Real value in IEEE display
IEEE$ = MKS$(Real value!) ’4 Byte string
FOR I = 0 TO 3
Byte(I) = ASC(MID$(IEEE$, I + 1, 1))
NEXT
Date0& = Byte(1) * 256 + Byte(0)
Date1& = Byte(3) * 256 + Byte(2)
’These 2 words must be correctly integrated into the send telegram
’-----------------------------------------------------
’Control displays
IEEE$ = HEX2$(Byte(3)) + HEX2$(Byte(2))
IEEE$ = IEEE$ + HEX2$(Byte(1)) + HEX2$(Byte(0))
PRINT IEEE$; " ="; BIN.AER$(IEEE$)
’=====================================================
’Recalculate
’-----------------------------------------------------
’received were bytes(0) to bytes(3)
’-----------------------------------------------------
IEEEHEX$ = ""
FOR I = 0 TO 3
IEEEHEX$ = IEEEHEX$ + CHR$(Byte(I))
NEXT
Realwert! = CVS(IEEEHEX$)
PRINT "Recalculation = "; Real value!
LOOP
’---------------------------------------
’Conversion of a hex numeral in binary display
’-----------------------------------------------
FUNCTION BIN.AER$ (z$)
DEFINT A-Z
FOR I = 1 TO LEN(z$)
x1$ = ""
x% = VAL("&H" + MID$(z$, I, 1))
DO UNTIL x% = 0
Y$ = LTRIM$(STR$(x% MOD 2))
x% = x% \ 2
x1$ = Y$ + x1$
LOOP
x1$ = RIGHT$("0000" + x1$, 4)
x$ = x$ + " " + x1$
NEXT
BIN.AER$ = x$
END FUNCTION
’---------------------------------------
DEFSNG A-Z
’Display the hex figures in 2 digits
’---------------------------------------
FUNCTION HEX2$ (x)
HEX2$ = RIGHT$("00" + HEX$(x), 2)
END FUNCTION
```

IEEE Value Calculation Without Special Functions

```
'Demo program for editing the IEEE value display
'in Quick-Basic 4.5 Version 1.0
'-------------------------------------------------------
DECLARE FUNCTION BIN.AER$ (z$)
CLS
DO UNTIL i = 127
INPUT "Real value (e = ende) "; Real value$
IF UCASE$(Real value$) = "E" THEN END
RealWert! = VAL(Real value$)
'-------------------------------------------------------
'Conditioning:
'=======================================================
'Separate preceding sign
Preceding sign = 0
IF RealWert! < 0 THEN
RealWert! = Real valuet! * (-1)
Preceding singn = -1
END IF
'-------------------------------------------------------
'Determine exponents
Exponent% = 0
X! = Real value!
IF X! > 1 THEN
DO UNTIL X! < 1
X! = X! / 2
Exponent% = Exponent% + 1
LOOP
Exponent% = Exponent% - 1
ELSE
DO UNTIL X! > 1
X! = X! * 2
Exponent% = Exponent% - 1
LOOP
PRINT Exponent%
END IF
'-------------------------------------------------------
'Determine mantissa
Mantissa = Real value! * (2 ^ (23 - Exponent%))
Mantissa = Mantissa AND &H7FFFFF
'-------------------------------------------------------
'Determine words and bytes for telegram
Exponent% = (Exponent% + &H7F) * 128
Date0 = Mantissa MOD &H10000
Date1 = Mantissa \ &H10000 + Exponent%
Byte(0) = Date0 MOD 256
Byte(1) = Date0 \ 256
Byte(2) = Date1 MOD 256
Byte(3) = Date1 \ 256 + ((-1) * preceding sign) * &H80
'-------------------------------------------------------
'Control display
PRINT "IEEE value: ";
FOR i = 3 TO 0 STEP -1
PRINT BIN.AER$(HEX$(Byte(i)));
NEXT
PRINT
'=======================================================
'Recalculate
'-------------------------------------------------------
'Received were bytes(0) to byte(3)
'-------------------------------------------------------
'Preceding sign is coded in bit 7 of byte(3)
Preceding sign = 1
IF (Byte(3) AND &H80) = &H80 THEN preceding sign = -1
'-------------------------------------------------------
'Determine exponent from bit 6 to 0 from byte(3)
'and bit 8 from byte(2)
Exponent = (Byte(3) AND &H7F) * 2 + (Byte(2) \ 128)
'-------------------------------------------------------
'Determine mantissa:
'Set bit 7 of byte(3),
```

```
’Calculate mantissa from byte(0) to Byte(3)
Mantissa = (Byte(2) OR &H80) * &H10000
Mantissa = Mantissa + Byte(1) * &H100 + Byte(0)
’-------------------------------------------------
Real value! = Sign * Mantissa / (2 ^ (23 - (Exponent - &H7F)))
PRINT "Recalculation = "; Real value!
LOOP
’-------------------------------------------------
’Conversion of a hex numeral in binary display
’-------------------------------------------------
FUNCTION BIN.AER$ (z$)
DEFINT A-Z
FOR i = 1 TO LEN(z$)
x1$ = ""
X% = VAL("&H" + MID$(z$, i, 1))
DO UNTIL X% = 0
Y$ = LTRIM$(STR$(X% MOD 2))
X% = X% \ 2
x1$ = Y$ + x1$
LOOP
x1$ = RIGHT$("0000" + x1$, 4)
X$ = X$ + " " + x1$
NEXT
BIN.AER$ = X$
END FUNCTION
```

Calculated Examples

Exponent on Basis 2 is calculated by repeatedly multiplying with 2 or by dividing by 2 until a 24-digit binary value with 1 as topmost (left) is produced. This 1 is suppressed in the IEEE display.

```
Decim. hexadecimal binary
s/Exponent /Value /
-1,0BF 80 00 00 1011 1111 1000 0000 0000 0000 0000 0000
-0,5BF 00 00 00 1011 1111 0000 0000 0000 0000 0000 0000
-0,4BE CC CC CD 1011 1110 1100 1100 1100 1100 1100 1101
-0,3BE 99 99 9A 1011 1110 1001 1001 1001 1001 1001 1010
-0,2BE 4C CC CD 1011 1110 0100 1100 1100 1100 1100 1101
-0,1BD CC CC CD 1011 1101 1100 1100 1100 1100 1100 1101
 0,000 00 00 00 0000 0000 0000 0000 0000 0000 0000 0000
 0,13D CC CC CD 0011 1101 1100 1100 1100 1100 1100 1101
 0,23E 4C CC CD 0011 1110 0100 1100 1100 1100 1100 1101
 0,33E 99 99 9A 0011 1110 1001 1001 1001 1001 1001 1010
 0,43E CC CC CD 0011 1110 1100 1100 1100 1100 1100 1101
 0,53F 00 00 00 0011 1111 0000 0000 0000 0000 0000 0000
 1,03F 80 00 00 0011 1111 1000 0000 0000 0000 0000 0000
10,041 20 00 00 0100 0001 0010 0000 0000 0000 0000 0000
```

Calculation of the CRC checksum

```
’Basic program for calculating the CRC checksum for Modbus-RTU telegrams
’Quickbasic 4.5 Version 1.0
’-----------------------------------------------------------
DECLARE FUNCTION HEX2$ (x!)
CLS
MaxI = 2
PRINT "input of the telegram bytes in Hex 05H or decimal 5"
PRINT "uniformly separated by blanks or decimal points"
DO
INPUT Tel$
i = 1
L = LEN(Tel$)
Tel$ = UCASE$(Tel$)
DO UNTIL Tel$ = ""
Tel$ = LTRIM$(Tel$)
```

```
x = INSTR(Tel$, " ") + INSTR(Tel$, ",")
IF x > 4 THEN Error = 1: EXIT DO
IF x > 0 THEN
Byte$(i) = LEFT$(Tel$, x)
Tel$ = RIGHT$(Tel$, LEN(Tel$) - x + 1)
ELSE
Byte$(i) = Tel$
Tel$ = ""
END IF
Byte$(i) = RTRIM$(Byte$(i))
IF RIGHT$(Byte$(i), 1) <> "H" THEN
Byte$(i) = HEX2$(VAL(Byte$(i)))
ELSE
Byte$(i) = LEFT$(Byte$(i), 2)
END IF
IF HEX2$(VAL("&H" + Byte$(i))) <> Byte$(i) THEN error = 1: EXIT DO
i = i + 1
LOOP
IF error = 0 THEN EXIT DO
SOUND 1000, .03
LOOP
MaxI = i - 1
x& = 65535
FOR i = 1 TO MaxI
y& = (VAL("&H" + Byte$(i)) XOR x&)
n = 1
DO
DO
r = y& MOD 2
y& = y& - r
y& = y& / 2
IF ABS(r) = 1 THEN EXIT DO
n = n + 1
IF (n = 9) AND (i = MaxI) THEN EXIT FOR
IF n = 9 THEN EXIT DO
LOOP
IF n < 9 THEN
y& = y& XOR 40961
n = n + 1
END IF
IF n = 9 THEN
IF (i = MaxI) THEN EXIT FOR
EXIT DO
END IF
LOOP
x& = y&
NEXT
PRINT "CRC ="; HEX$(y&); " Hex"
PRINT " must be in serial sequence "; HEX2$(y& MOD 256);
" "; HEX2$(y& \ 256);
PRINT " incorporate in the telegram !"
FUNCTION HEX2$ (x)
HEX2$ = RIGHT$("00" + HEX$(x), 2)
END FUNCTION
```

Subject to technical changes.

ABB Automation Products GmbH
Borsigstrasse 2
D-63755 Alzenau
Tel. +49(0)60 23 92 - 0
Fax +49(0)60 23 92 - 33 00
http://www.abb.com

Subject to technical changes.
Printed in the Fed. Rep. of Germany
42/18-58 EN Rev. 1.0
Edition 04.01