ANALYSER OF 3-PHASE POWER NETWORK PARAMETERS

ND1 TYPE

MODBUS TRANSMISSION PROTOCOL USER'S MANUAL

CONTENTS

1. APPLICATION

In order to obtain the information exchange, when using the serial link, one must choose the interface type and validate the interpretation way of transmitted data. The interface type defines only electrical transmission parameters and the way of the device connection.
Such features, as the possibility to service several devices, check the transmission correctness and the principles of access to the device, depend on the data interpretation.
The task of the protocol is to define which data type is interpreted (permitted) and in which way they are interpreted.
A MODBUS asynchronous character transmission protocol has been implemented on the serial link of the **ND1** analyser. The parameter configuration of the **RS-485** serial link is described in the **ND1** analyser user's manual.
Parameter set of the **ND1** analyser serial link:

• Analyser address	1 ...247
• Baud rate	300, 600, 1200, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200 bit/s
• Working mode	ASCII, RTU
• Information unit	ASCII: 8N1, 7N2, 7E1, 701 RTU:8N2, 8N1, 8E1,8O1
• Maximal turnaround time	100 ms

2. DESCRIPTION OF THE MODBUS PROTOCOL

The MODBUS interface is a standard adopted by manufacturers of industrial controllers for the asynchronous character exchange of information between different devices and measuring systems. It has such features as:
• Simple access rule to the link based on the "master-slave" principle,
• Protection of transmitted messages against errors,
• Confirmation of remote instruction realization and error signaling,
• Effective actions protecting against the system suspension,
• Taking advantage of the asynchronous character transmission.
Device controllers working in the **MODBUS** protocol can communicate with each other, taking advantage of the **master-slave** protocol type, in which only one device (the **master** - superior unit) can originate transactions (called ''queries"), and others (slaves -
subordinate units) respond only to the remote requested data from the **master.** The transaction is composed of the transmitted command from the **master** unit to the **slave** unit and of the response transmitted in the opposite direction. The response includes data demanded by the master or the confirmation of the command realization.
Master can transmit information to individual slaves or broadcast messages destined for all subordinate devices in the system (responses are not returned to broadcast queries from the master)

The format of transmitted information is as following:
• **master => slave:** device address, code representing the required command, data to be sent, control word protecting the transmitted message,
• **slave => master:** sender address, confirmation of the command realization, data required by the master, control word protecting the response against errors.

If the **slave** device detects an error when receiving a message, or cannot realizes the command, it prepares a special message about the error occurrence and transmits it as a response to the **master.**
Devices working in the **MODBUS** protocol can be set into the communication using one of two transmission modes: **ASCII or RTU.** The user chooses the required mode, along with the serial

port communication parameters (baud rate, information unit) during the configuration of any device.
In the **MODBUS** system, transmitted messages are placed into frames that are not related to serial transmission. These frames have a defined beginning and end. This enables for the receiving device to reject incomplete frames and signaling related errors with them.
Taking into consideration the possibility to operate in one of these two different transmission modes (ASCII or RTU), two frames have been defined.

Explanation of some abbreviations:

ASCII = American Standard Code for Information Interchange

RTU = Remote Terminal Unit

LRC = Longitudinal Redundancy Check

CRC = Cyclic Redundancy Check CR = Carriage Return LF = Line-Feed (character)

MSB = Most Significant Bit

Checksum = Control Sum

2.1. ASCII framing

In the ASCII mode, each byte of information is transmitted as two ASCII characters. The basic feature of this mode is that it allows to long intervals between characters within the message (to 1 sec) without causing errors.

A typical message frame is shown below.

Start beginning index	Address	Function	Data	LRC check	End index
1 char /:/	2 chars	2 chars	*n* chars	2 chars	2 chars CR LF

In ASCII mode, messages start with a colon character („:" -ASCII 3Ah) and end with CR and LF characters. The frame information part is protected by the LRC code (Longitudinal Redundancy Check).

2.2. RTU Framing

In RTU mode, messages start and end with an interval lasting minimum 3.5 x (lasting time of a single character), in which a silence reigns on the link.

The simplest implementation of the mentioned time interval character times is a multiple measure of the character duration time at the set baud rate accepted on the link.

The frame format is shown below:

Start beginning index	Address	Function	Data	CRC check	End index
T1-T2-T3-T4	8 bits	8 bits	*n* x 8bits	16 bits	T1-T2-T3-T4

Start and end indexes are marked symbolically as an interval equal to four lengths of the index (information unit). The checking code consists of 16 bits and emerges as the result of CRC calculation (Cyclical Redundancy Check) of the frame contents.

2.3. Characteristic of frame fields.

Address field

The address field of a message frame contains two characters (in ASCII mode) or eight bits (in RTU mode).
Valid slave device addresses are in the range from 0 -247 . The master addresses the slave unit by placing the slave address in the frame address field. When the slave sends its response, it places its own address in the frame address field what enables the master to check which slave is responding.
The 0 address is used as a broadcast address recognized by all slave units connected to the bus.

Function field

The function code field of a message frame contains two characters in ASCII mode or eight bits in RTU mode. Valid codes are in the range from 1 - 255.
When a message is sent from a master to a slave device, the function code field tells the slave what kind of action to perform.
When the slave responds to the master, the function field is uses to confirm the command execution or error signaling if the function code field cannot realize the command for any reason. to indicate either a normal (error-free) response or that some kind of error occurred.
The positive confirmation is realized through the placement of the command execution code on the function field.
In case of an error assertion, the slave returns a special code that is equivalent to the original function code with its most significant logic 1.
The error code is placed on the data field of the response frame.

Data field

The data field is constructed using sets of two hexadecimal digits, in the range of 00 to FF.
These numbers can be made from a pair of ASCII characters or from one RTU character, according to the network's serial transmission mode. The data field of messages sent from a master to slave devices contains additional information which the slave must use to take the action defined by the function code. This can include items like register addresses, number of bytes in data field, data, a.s.o. The data field can be non-existent (of zero length) in certain kinds of frames. That occurs always, when the operation defined by the code does not require any parameters.

Error checking field

Two kinds of error-checking methods are used for standard MODBUS networks. The error checking field contents depends upon the method that is being used.

ASCII

When ASCII mode is used for character framing, the error checking field contains two ASCII characters. The error check characters are the result of a Longitudinal Redundancy Check (LRC) calculation that is performed on the message contents (without the beginning ’’colon" and terminating CRLF characters). LRC characters are appended to the message, as the last field preceding the CR, LF characters.

RTU

When RTU mode is used for character framing, the error checking field contains a 16-bit value implemented as two 8-bit bytes. The error check value is the result of a Cyclical Redundancy Check Calculation (CRC) performed on a message contents. The CRC field is appended to the message as

the last field in the message. When this is done, the low-order byte of the field is appended first, followed by the high-order byte. The CRC high-order byte is the last byte to be sent in the message.

2.4. LRC checking

The LRC is calculated by adding together successive 8-bit bytes of the message, discarding any carries, and then two is complementing the result. It is performed on the ASCII message field contents excluding the,,colon" character that begins the message, and excluding the CR, LF pair at the end of the message. The 8-bit value of the LRC sum is placed at the frame end as two ASCII characters, first the character containing the higher tetrad, and after it, the character containing the lower LRC tetrad.

2.5. CRC checking

The generating procedure of CRC is realized according to the following algorythm:

1. Load a 16-bit register with FFFFh. Call this the CRC register.
2. Exclusive EXOR the first 8-bit byte of the message with the low-order byte of the 16 bit CRC register, putting the result in the CRC register.
3. Shift the CRC register contents one bit to the right (towards the LSB), zero-filling the MSB. Extract and examine the LSB.
4. (If the LSB was O): Repeat step 3 (another shift) (If the LSB was 1): Exclusive EXOR the CRC register with the polynomial value A001h.
5. Repeat steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8-bit byte will have been processed.
6. Repeat steps 2 through 5 for the next 8-bit byte of the message. Continue doing this until all bytes have been processed.
7. The final contents of the CRC register is the CRC value.
8. When the CRC is placed into the message, its upper and lower bytes must be swapped as described below.

2.6. Character format during serial transmission

In the **MODBUS** protocol, characters are transmitted from the lowest to the highest bit.
Organization of the information unit in the ASCII mode:

- 1 start bit,
- 7 data field bits,
- 1 even parity check bit (odd) or lack of even parity check bit,
- 1 stop bit at even parity check or 2 stop bits when lack of even parity check.

Organization of the information unit in the RTU mode:

- 1 start bit,
- 8 data field bits,
- 1 even parity check bit (odd) or lack of even parity check bit,
- 1 stop bit at even parity check or 2 stop bits when lack of even parity check.

2.7. Transaction interruption

In the master unit the user sets up the important parameter which is the ’’maximal response time on the query frame" after exceeding of which, the transaction is interrupted. This time is chosen such that each slave unit working in the system (even the slowest) normally will have the time to answer to the frame query. An exceeding of this time attests therefore about an error and such is treated by the master unit.
If the unit slave will find out a transmission error it does not accomplish the order and does not send any answer. That causes an exceeding of the waiting time after the query frame and the transaction interruption.

3. FUNCTION DESCRIPTION

In the ND1 analyser following protocol functions has been implemented:

Code	Signification
03	Reading of n-register
17	Slave device identification

3.1. Reading of n-registers (code 03)

Demand:
The function enables the reading of values included in registers in being addressed slave device. **Registers are 16 or 32-bit units, which can include numerical values bounded with changeable processes, and the like.** The demand frame defines the 16-bit start address and the number of registers to read-out.
The signification of the register contents with address data can be different for different device types.
The function is not accessible in the broadcast mode.

Example: Reading of 3 registers beginning by the register with the 6Bh address.

Address	Function	Register address Hi	Register address Lo	Number of registers Hi	Number of registers Lo	Checksum	
11	03	00	6B	00	03	7E	LRC

Answer:
Register data are packing beginning from the smallest address: first the higher byte, then the lower register byte.

Example: the answer frame

Address	Function	Number of bits	Value in the regist 107 Hi	Value in the regist 107 Lo	Value in the regist 108 Hi	Value in the regist 108 Lo	Value in the regist 109 Hi	Value in the regist 109 Lo	Checksum	
11	03	06	02	2B	00	00	00	64	55	LRC

3.2 Writing of values in the register (code 06)

Demand:
The function enables the modification of the register contents. It is accessible in broadcast mode.

Example:

Address	Function	Register address Hi	Register address Lo	Value Hi	Value Lo	Checksum	
11	06	00	87	03	9E	C1	LRC

Answer:
The correct answer to a value record demand in the register is the retransmission of the message after accomplishing the operation.

Example:

Address	Function	Register address Hi	Register address Lo	Value Hi	Value Lo	Checksum	
11	06	00	87	03	9E	C1	LRC

3.3 Writing in n-registers (code 16)

Demand:

The function is accessible in broadcast mode. It enables the modification of the register contents.

Example: Writing of two registers beginning from the register addressed 136.

Address	Function	Register address Hi	Register address Lo	Number of registers Hi	Number of registers Lo	Number of bytes	Data Hi	Data Lo	Data Hi	Data Lo	Checksum	
11	10	00	87	00	02	04	00	0A	01	02	45	LRC

Answer:

The correct answer includes the unit slave address, function code, starting address and the number of recorded registers.

Example:

Address	Function	Register address Hi	Register address Lo	Number of registers Hi	Number of registers Lo	Checksum	
11	10	00	87	00	02	56	LRC

3.4. Report identifying the device (code 17)

Demand:

This function enables the user to obtain information about the device type, status and configuration depending on this.

Example

Address	Function	Checksum	
11	11	DE	LRC

Answer:
The field ,,Device identifier" in the answer frame means the unique identifier of this class of device, however the other fields include parameters depended on the device type.

Example concerning the ND1 analyser

Slave address	Function	Number of bytes	Device identifier	Device state	Checksum	
11	11	02	BD	FF	4D	EF

4. ERROR CODES

When the master device is broadcasting a demand to the slave device then, except for messages in the broadcast mode, it expects a correct answer. After sending the demand of the master unit, one of the four possibilities can occur:

- If the slave unit receives the demand without a transmission error and can execute it correctly, then it returns a correct answer,
- If the slave unit does not receive the demand, no answer is returned. Timeout conditions for the demand will be fulfilled in the master device program.
- If the slave unit receives the demand, but with transmission errors (even parity error of checking sum LRC or CRC), no answer is returned. Timeout condition for the demand will be fulfilled in the master device program.
- If the slave unit receives the demand without a transmission error but cannot execute it correctly (e.g. if the demand is, the reading-out of a non-existent bit output or register), then it returns the answer including the error code, informing the master device about the error reason.

A message with an incorrect answer includes two fields distinguishing it from the correct answer.

1. The function code field:

In the correct answer, the slave unit retransmits the function code from the demand message in the field of the answer function code. All function codes have the most-significant bit (MSB) equal zero (code values are under 80h). In the incorrect answer, the slave unit sets up the MSB bit of the function code at 1. This causes that the function code value in the incorrect answer is exactly of 80h greater than it would be in a correct answer.
On the base of the function code with a set up MSB bit the program of the master device can recognize an incorrect answer and can check the error code on the data field.

2. The data field:

In a correct answer the slave device can return data to the data field (certain information required by the master unit). In the incorrect answer the slave unit returns the error code to the data field. It defines conditions of the slave device which had produced the error. An example considering a demand of a master device and the incorrect answer of the slave unit has been shown below. Data are in the hexadecimal shape.

Example: demand

Slave address	Function	Variable address H1	Variable address Lo	Number Of Variables Hi	Number Of Variables Lo	Checksum	
OA	01	04	A1	00	01	4F	LRC

Example: incorrect answer

Slave	Function	Error	Checksum	
OA	81	02	73	LRC

In this example the master device addresses the demand to the slave unit with No 10 (OAh). The function code (01) serves to the read-out operation of the bit input state. Then, this frame means the demand of the status read-out of a one bit input with the address number: 1245 (04A1h).
If in the slave device there is no bit input with the given address, then the device returns the incorrect answer with the No 02 error code. This means a forbidden data address in the slave device.

Possible error codes and their meanings are shown in the table below.

Code	Meaning
01	Forbidden function
02	Forbidden data address
03	Forbidden data value
04	Damage in the connected device
05	Confirmation
06	Occupied, message removed
07	Negative confirmation
08	Error of memory parity

5. Table of registers

- ND1 analyser identifier (set as a response to the identification function) : 0xBD
- Register types:
  float – floating point number (see the description below),
  sfloat – floating point number (see the description below),
- Access modes to register:
  RO – Read Only,
- Representation of floating point numbers (float IEEE 754)

byte 4	byte 3	byte 2	Byte 1
SEEEEEEE	EMMMMMMM	MMMMMMMM	MMMMMMMM

S – Sign bit
E – Exponent
M – mantissa

Register bytes of **float** type are sent in 4321 sequence
Register bytes of s**float** type are sent in 2143 sequence

Table 1 ND1 registers table

Address	**Type**	**Access**	**Description**
2000	int	RO	Alarms – successive bits are successive alarms
			Registers of float type, 16 bit addressed
4000..4237	float	RO	Network parameters (see Table 2)
4300..4347	float	RO	Modbus Master Inputs
4500..4523	float	RO	Binary Inputs
10000..10357	float	RO	Voltage Harmonics
10000..10101	float	RO	Voltage Harmonics L1
10128..10229	float	RO	Voltage Harmonics L2
10256..10357	float	RO	Voltage Harmonics L3
10400..10757	float	RO	Current Harmonics
10400..10501	float	RO	Current Harmonics L1
10528..10629	float	RO	Current Harmonics L2
10656..10757	float	RO	Current Harmonics L3
			Registers of sfloat type, 16 bit addressed
5000..5237	sfloat	RO	See registers 4000..4237
5300..5347	sfloat	RO	See registers 4300..4347

5500..5523	sfloat	RO	See registers 4500..4523
15000..15357	sfloat	RO	See registers 10000..10357
15400..15757	sfloat	RO	See registers 10400..10757
			Registers of float type, 32 bit addressed
7000..7118	float	RO	See registers 4000..4237
7200..7223	float	RO	See registers 4300..4347
7300..7311	float	RO	See registers 4500..4523
20000..20178	float	RO	Voltage Harmonics (see registers 10000..10357)
20000..20050	float	RO	Voltage Harmonics L1
20064..20114	float	RO	Voltage Harmonics L2
20128..20178	float	RO	Voltage Harmonics L3
20200..20378	float	RO	Current Harmonics (see registers 10400..10757)
20200..20250	float	RO	Current Harmonics L1
20264..20314	float	RO	Current Harmonics L2
20328..20378	float	RO	Current Harmonics L3
			Registers of sfloat type, 32 bit addressed
8000..8118	sfloat	RO	See registers 4000..4237
8200..8223	sfloat	RO	See registers 4300..4347
8300..8311	sfloat	RO	See registers 4500..4523
25000..25178	sfloat	RO	See registers 20000..20178
25200..25378	sfloat	RO	See registers 20200..20378

Table 2 Registers of power parameters

Index	16-bit addressed registers		32-bit addressed registers		Description
	float	sfloat	float	sfloat	
0	4000	5000	7000	8000	Urms L1
1	4002	5002	7001	8001	Urms L2
2	4004	5004	7002	8002	Urms L3
3	4006	5006	7003	8003	U L1-2
4	4008	5008	7004	8004	U L2-3
5	4010	5010	7005	8005	U L3-1
6	4012	5012	7006	8006	Upeak- L1
7	4014	5014	7007	8007	Upeak- L2
8	4016	5016	7008	8008	Upeak- L3
9	4018	5018	7009	8009	Upeak+ L1
10	4020	5020	7010	8010	Upeak+ L2
11	4022	5022	7011	8011	Upeak+ L3
12	4024	5024	7012	8012	Ucf L1
13	4026	5026	7013	8013	Ucf L2
14	4028	5028	7014	8014	Ucf L3
15	4030	5030	7015	8015	Irms L1
16	4032	5032	7016	8016	Irms L2
17	4034	5034	7017	8017	Irms L3
18	4036	5036	7018	8018	Icf L1
19	4038	5038	7019	8019	Icf L2
20	4040	5040	7020	8020	Icf L3
21	4042	5042	7021	8021	INM L1
22	4044	5044	7022	8022	INC L1
23	4046	5046	7023	8023	Reserved
24	4048	5048	7024	8024	IPeak- L1

25	4050	5050	7025	8025	IPeak- L2
26	4052	5052	7026	8026	IPeak- L3
27	4054	5054	7027	8027	IPeak+ L1
28	4056	5056	7028	8028	IPeak+ L2
29	4058	5058	7029	8029	IPeak+ L3
30	4060	5060	7030	8030	φ U-I L1 [rad]
31	4062	5062	7031	8031	φ U-I L2 [rad]
32	4064	5064	7032	8032	φ U-I L3 [rad]
33	4066	5066	7033	8033	φ U L1-2 [rad]
34	4068	5068	7034	8034	φ U L2-3 [rad]
35	4070	5070	7035	8035	φ U L3-1 [rad]
36	4072	5072	7036	8036	φ I L1-2 [rad]
37	4074	5074	7037	8037	φ I L2-3 [rad]
38	4076	5076	7038	8038	φ I L3-1 [rad]
39	4078	5078	7039	8039	φ U L1 [rad]
40	4080	5080	7040	8040	φ U L2 [rad]
41	4082	5082	7041	8041	φ U L3 [rad]
42	4084	5084	7042	8042	φ I L1 [rad]
43	4086	5086	7043	8043	φ I L2 [rad]
44	4088	5088	7044	8044	φ I L3 [rad]
45	4090	5090	7045	8045	φ U-I L1 [°]
46	4092	5092	7046	8046	φ U-I L2 [°]
47	4094	5094	7047	8047	φ U-I L3 [°]
48	4096	5096	7048	8048	φ U L1-2 [°]
49	4098	5098	7049	8049	φ U L2-3 [°]
50	4100	5100	7050	8050	φ U L3-1 [°]
51	4102	5102	7051	8051	φ I L1-2 [°]
52	4104	5104	7052	8052	φ I L2-3 [°]
53	4106	5106	7053	8053	φ I L3-1 [°]
54	4108	5108	7054	8054	φ U L1 [°]
55	4110	5110	7055	8055	φ U L2 [°]
56	4112	5112	7056	8056	φ U L3 [°]
57	4114	5114	7057	8057	φ I L1 [°]
58	4116	5116	7058	8058	φ I L2 [°]
59	4118	5118	7059	8059	φ I L3 [°]
60	4120	5120	7060	8060	P L1
61	4122	5122	7061	8061	P L2
62	4124	5124	7062	8062	P L3
63	4126	5126	7063	8063	S L1
64	4128	5128	7064	8064	S L2
65	4130	5130	7065	8065	S L3
66	4132	5132	7066	8066	Q L1
67	4134	5134	7067	8067	Q L2
68	4136	5136	7068	8068	Q L3
69	4138	5138	7069	8069	Tgφ L1
70	4140	5140	7070	8070	Tgφ L2
71	4142	5142	7071	8071	Tgφ L3
72	4144	5144	7072	8072	PF L1
73	4146	5146	7073	8073	PF L2
74	4148	5148	7074	8074	PF L3
75	4150	5150	7075	8075	Reserved

76	4152	5152	7076	8076	Reserved
77	4154	5154	7077	8077	Reserved
78	4156	5156	7078	8078	Reserved
79	4158	5158	7079	8079	Reserved
80	4160	5160	7080	8080	Reserved
81	4162	5162	7081	8081	Reserved
82	4164	5164	7082	8082	Reserved
83	4166	5166	7083	8083	Reserved
84	4168	5168	7084	8084	Uavg
85	4170	5170	7085	8085	Iavg
86	4172	5172	7086	8086	Reserved
87	4174	5174	7087	8087	Reserved
88	4176	5176	7088	8088	Reserved
89	4178	5178	7089	8089	Reserved
90	4180	5180	7090	8090	∑P
91	4182	5182	7091	8091	∑S
92	4184	5184	7092	8092	∑Q
93	4186	5186	7093	8093	Tgφav
94	4188	5188	7094	8094	PFav
95	4190	5190	7095	8095	Reserved
96	4192	5192	7096	8096	Reserved
97	4194	5194	7097	8097	Reserved
98	4196	5196	7098	8098	f
99	4198	5198	7099	8099	Pav
100	4200	5200	7100	8100	Uav
101	4202	5202	7101	8101	UavQTotal
102	4204	5204	7102	8102	UavQPart
103	4206	5206	7103	8103	fav
104	4208	5208	7104	8104	favQTotal
105	4210	5210	7105	8105	favQPart
106	4212	5212	7106	8106	Reserved
107	4214	5214	7107	8107	Reserved
108	4216	5216	7108	8108	Reserved
109	4218	5218	7109	8109	Reserved
110	4220	5220	7110	8110	Reserved
111	4222	5222	7111	8111	Reserved
112	4224	5224	7112	8112	Reserved
113	4226	5226	7113	8113	THD U L1
114	4228	5228	7114	8114	THD U L2
115	4230	5230	7115	8115	THD U L3
116	4232	5232	7116	8116	THD I L1
117	4234	5234	7117	8117	THD I L2
118	4236	5236	7118	8118	THD I L3

APPENDIX A. CALCULATION OF THE CHECKSUM

In this appendix some examples of function in the C language calculating the LRC checksum for ASCII mode and the CRC checksum for the RTU mode have been shown

The function for LRC calculation has two arguments:

*unsigned char *outMsg;* - Pointer for the communication buffer, including binary data from which one must calculate LRC.
unsigned short usDataLen; - Number of bytes in the communication buffer.

The function returns LRC of *unsigned char type.*

```
static unsigned char LRC(outMsg, usDataLen)
unsigned char *outMsg;                    /* buffer to calculate LRC */
unsigned short usDataLen;                 /* number of bytes in the buffer 7
{
unsigned char uchLRC = 0;                 /* initialization of LRC */
 while (usDataLen--)
uchLRC += *outMsg++;                      /* add the buffer byte without transfer*
return ((unsigned char)(-(char uchLRC)));  /* return the sum in the completion code up two*/
}
```

An example of function in C language calculating the CRC sum is presented below. All possible values of CRC sum are placed in two tables.
The first table includes the highest byte of all 256 possible values of the 16-bit CRC field, however the second table includes the lowest byte.
The assignment of the CRC sum through table indexing is further more rapid than the calculation of a new CRC value for each sign of the communication buffer.

Note: The below function represents bytes of the sum CRC higher/lower, and this way the CRC value returned by the function can be directly placed in the communication buffer.
The function serving to calculate CRC has two arguments:

*unsigned char *puchMsg;* - Pointer for the communication buffer, including binary data from which one must calculate LRC.
unsigned short usDataLen; - Number of bytes in the communication buffer.

The function returns CRC of *unsigned short* type.

```
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg;                   /* buffer to calculate CRC */
 unsigned short usDataLen;                /*Number of bytes in the buffer */
{
unsigned char uchCRChi = OxFF;            /* initialisation of the higher CRC byte*/
 unsigned char uchCRCIo = OxFF;           /* initialisation of the lower CRC byte */
 while (usDataLen--)
{        uindex = uchCRChiA *puchMsg++;   /* CRC calculation*/
         uchCRChi = uchCRCIo A crc_hi[ulndex];
         uchCRCIo = crc_lo[ulndex];
}
retum(uchCRChi«8 \ uchCRCIo);
}
```

//table of the older CRC byte /
const unsigned char crc_hi[]={

```
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
};
```

//table of the lower CRC byte /

```
const unsigned char crc_lo[]={
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
};
```

Lubuskie Zakłady Aparatów Elektrycznych LUMEL S.A.
ul. Sulechowska 1, 65-022 Zielona Góra, Poland (EU)
http://www.lumel.com.pl

Export department
Tel.: (48-68) 329 53 02
Fax: (48-68) 325 40 91
e-mail: export@lumel.com.pl

ND1-09/2 (16.11.09)