MICROPROCESSOR CONTROLLER

RE15

SERIAL INTERFACE WITH MODBUS PROTOCOL

USER’S MANUAL

CONTENTS

1. PREFACE

The RE15 microprocessor controller destined to measure and control physical quantities is provided with a serial interface in RS-485 standard for the communication with other devices.
The asynchronous communication MODBUS protocol has been implemented on this serial interface.
The configuration of serial interface parameters has been described in the User's Manual of the R15 controller.

Composition of serial interface parameters concerning RE15 controller:

- Controller address - 1 ... 247
- Baud rate - 2400, 4800, 9600 bits/s,
- Working modes - ASCII, RTU,
- Information unit - ASCII: 8N1, 7E1, 7O1;
  and RTU: 8N2, 8E1, 8O1,8N1
- Maximal turnaround time - 1s

Explanation of some abreviations:

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. DESCRIPTION OF THE MODBUS PROTOCOL

The MODBUS interface is a standard adopted by manufacturers of industrial controllers for an asynchronous character exchange of information between different devices of measuring and control 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 realisation and error signalling,
- Effective actions protecting against the system suspension,
- Taking advantage of the asynchronous character transmission.

Programmable controllers working in the **MODBUS** system 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 query by supplying the requested data to 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 realised confirmation of its command. **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 follows:

• **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 realize 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 parameters (baud rate, information unit) during the configuration of any device.

In the **MODBUS** system, transmitted messages are placed into frames that are no related to serial transmission. These frames have a defined beginning and end. This enables for the receiving device to reject incomplete frames and the signalling of 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.

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 (to1sec) 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 a carriage return-line feed" (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 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 bedinning index	Address	Function	Data	CRC check	End index
T1-T2-T3-T4	8 bits	8 bits	n x 8 bits	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) on 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 of 0-247 decimal. The master addresses the slave units 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 of 1-255 decimal. 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, it uses the function code field to indicate and confirm either a normal (error-free) response or that some kind of error occurred and the realization of the command is impossible.
For a formal response the slave simply echoes the original function code. 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 - FF hexadecimal.
These can be made from a pair of ASCII characters or from one RTU character, according to the network's serial transmission mode. The function code range is 1-255. 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 discrete and register addresses, the quantity of items to be handled, and count of actual data bytes in the field, 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 parameters.

Error checking field

Two kinds of error-checking methods are used for standard MODBUS networks. The error checking field contents depends upon the applied transmission mode.

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 (exclusive of the beginning „colon" and terminating CRLF characters). LRC characters are appended to the message as the last frame field preceding the end markers (CR, LF).

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 realised according the following algorythm:

1. Load a 16-bit register with FFFFh. Call this the CRC register.
2. Take the byte from the data block and execute the EXOR operation with the low-order byte of the CRC register. Place the result into the CRC register.
3. Shift the CRC register contents one bit to the right (towards the LSB), write 0 on the most significant bit (MSB=0).
4. Check the state of the lowest order bite (LSB) extracted from the CRC register in the previous step. If its state is equal 0, then follows a return to the step 3 (another shift).

   If the LSB is equal 1, the operation EXOR of the CRC register is executed 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 of the message have been processed.
7. The final contents of the CRC register is the searched 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 which exceeding, 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 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.

In the R15 controller „ maximal response time on the query frame" is equal 1 s.

3. FUNCTION DESCRIPTION

In the R15 controller following protocol functions have been implemented:

Code	Signification
03	Reading of n-register
06	Writing of an individual register
16	Writing of n-registers
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 maximal number of registers read out by one command is 128 for the RE15 controller.

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.

The function is not accessible in the broadcasting mode.

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

Address	Funtion	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

Adres	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 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. The maximal number of registers read out by one command is 128 for RE15 controller.

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

Address	**Function**	**Regist. address Hi**	**Regist. address Lo**	**Number of regist. Hi**	**Number of regist. 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.
The RE15 controller gives information related to additional inputs and outputs.

Example concerning the RE15 controller

Slave address	Function	Number of bytes	Device identifier	Type of additional input	Type of output	Execution number[2]	Checksum
11	11	4	68	xx[1)]	yy[2)]	0[3)]	

[1)] XX - value as in the controller execution code - Item related with the additional input

[2)] YY - four discontinuous output,
1 - one continuous output + three discontinuous outputs
2 - two continuous outputs + two discontinuous outputs

[3)] 0 - for standard execution, different from 0, for custom-made executions.

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.

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.

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: read out 4520 register (11A8h) and the answer of the slave unit: forbidden data address, because the maximal register address in the RE15 controller is 4519.

Data are in the hexadecimal shape.

Example: demand

Slave address	**Function**	**Variable address Hi**	**Variable address Lo**	**Number of variables Hi**	**Number of variables Lo**	**Checksum**	
0A	03	12	BO	00	01	39	LRC

Example: answer

Address	**Function**	**Error**	**Checksum**	
0A	83	02	71	LRC

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

5. TABLE OF REGISTERS FOR THE RE15 CONTROLLER

In the RE15 controller data are placed in 16-bit registers.
Bits in the register are numbered from the lowest to the highest one (b0-b15).
The list of registers are presented in the tabel 1.
Symbols R, W in the Option column signify allowable actions on the controller data: R-readout, W-writing.

Table 1. Contents of 16-bit registers

Regist. address	Option	Symbol	Range	Description
4000	R,W	inPu	0...16	Output type: 0-PT100, 1-PT1000, 2-Ni100, 3-Cu100, 4-J, 5-T, 6-K, 7-S, 8-R, 9-B, 10-E, 11-N, 12-Chromel-copel, 13-resistance 0...400 Ω ,14-0...20 mA, 15-4...20 mA, 16-0...10V, 17- 0...5V
4001	R,W	t.Li	0, 1	0- 2-wire line, 1- 3-wire line
4002	R,W	r.Li	0...200	Resistance of the line*10
4003	R,W	ConP	-1...501	Compensation of cold ends*10, ; values <0 or >500 mean automatic compensation
4004	R,W	L.cPP	0, 1, 2	Number of digits after coma (value 2 for linear inputs)
4005	R,W	S.hiF	-999...999	Shift of measured value$^{1)}$
4006	R,W	SPLL	-999...SPLH	Lower range of the measured value on the main input$^{1)}$
4007	R,W	SPLH	S.PLL ...9999	Upper range of the measured value on the main input$^{1)}$
4008	R,W	i.2Lo	-999...i.2Hi	Lower range of the measured value on the additional input$^{1)}$
4009	R,W	i.2Hi	i.2Lo...9999	Upper range of the measured value on the additional input $^{1)}$
4009	R,W	i.2Hi	i.2Lo...9999	Upper range of the measured value on the additional input$^{1)}$
4010	R,W	i.nP	0...5	Range of the additional input: 0-0...20 mA; 1-4...20 mA; 2-0...10 V ; 3-0...5 V; 4-0...100 Ω ; 5-0...1000 Ω ;
4011	R,W	F.in	0...4	0 - set value (r-SP 8. inP2); 1- extra information measurement; 2- sum of signals from both inputs; 3 - difference with main input; 4- arythmetical mean from both inputs
4012	R,W	F.inb	0...3	Function of the binary input: 0-no used binary input; 1- stops the control (control signal=0); 2- switch over to the manual work, 3- program end; 4- program stopped on the value counted lately

4013	R,W	o.uc	0...3	Range of the continuous output no I: 0-0-20 mA for current outputs; 1-4-20 mA ; 2-0-10 V ; 3-0-5 V
4014	R,W	o.uc	0...3	Range of the continuous output no II: as above
4015	R,W	o.ut	0...18	Output function no 1: 0-no used, 1-Y. 1, 2-Y.2-c, 3-Y.2-S., 4-Ah i, 5-ALo, 6-dbh i, 7-dbLo, 8-dbhL, 9-db in, 10-A.h i2, 11-A.Lo2, 12-E.out, 13-E.oP., 14-E.rr 1, 15-E.rr2, 16-tr i1, 17-tr i2, 18-trSP
4016	R,W	o.ut	0...18	Output function no 2: as above
4017	R,W	o.ut	0...18	Output function no 3: as above
4018	R,W	o.ut	0...18	Output function no 4: as above
4019	R,W	b.Ar	0...4	Bargraph function no 1: 0- control signal Y1 0...100%; 1-control signal Y2 0...100%; 2-signal from the main input SPLL...SPLh; 3-signal from the additional input i2L.o... i2H i; 4-set value SPLL...SPLh
4020	R,W	b.Ar	0...4	Bargraph function no 2: as above
4021	R,W	rSP	0, 1, 2	Kind of set value, 0- constant value, 1- programmed, 2- from the additional input
4022	R,W	SP	Depends on the input	Set value for the constant control1)
4023	R,W	nAro	0...999	Change speed of the set value during the soft-start*10
4024	R,W	n.rPr	1...15	Program number executed for the programmed control
4025	W R		0, 1, 2 0, 1	Control of the program work:0-stop, 1-continue, 2-start from the beginning
4026	R,W	P.b	0...9999	Proportional band for the main line*10
4027	R,W	t.i	0...3600	Integration time-constant
4028	R,W	t.d	0...1000	Differentiation time-constant
4029	R,W	t.o	1...250	Cycle time
4030	R,W	H	0...999	Hysteresis for on-off control1)
4031	R,W	Pb-c	0...9999	Proportional band for the auxiliary line*10
4032	R,W	t.i -c	0...3600	Integration time-constant for the auxiliary line
4033	R,W	t.d-c	0...1000	Differentiation time-constant for the auxiliary line
4034	R,W	to-c	1...250	Cycle time for the auxiliary line
4035	R,W	Hi -c	0...999	Hysteresis for on-off control of the auxiliary control 1)
4036	R,W	Hn	0...999	Dead band for three-state control1)
4037	R,W	tYPr	0,1	Kind of control: 0: inverse, 1- direct
4038	R,W	Y-oF	0...1000	Correction of the control signal *10 (for the integration time-constant ti=0)

4039	R,W	1ASP	for SP	Set value for the alarm on the output 1$^{1)}$
4040	R,W	1AHi	0...999	Hysteresis for the alarm on output 1$^{1)}$
4041	R,W	1APA	0,1	Storage of alarm no 1: 0-off, 1-on
4042	R,W	2ASP	for SP	Set value for the alarm on the output 2$^{1)}$
4043	R,W	2AHi	0...999	Hysteresis for the alarm on output 2$^{1)}$
4044	R,W	2APA	0,1	Storage of alarm no 2: 0-off, 1-on
4045	R,W	3ASP	for SP	Set value for the alarm on the output 3$^{1)}$
4046	R,W	3AHi	0...999	Hysteresis for the alarm on output 3$^{1)}$
4047	R,W	3APA	0,1	Storage of alarm no 3: 0-off, 1-on
4048	R,W	4ASP	for SP	Set value for the alarm on the output 4$^{1)}$
4049	R,W	4AHi	0...999	Hysteresis for the alarm on output 4$^{1)}$
4050	R,W	4APA	0,1	Storage of alarm no 4: 0-off, 1-on
4052	R,W	cont	0,1	Index of the constant control continuation after the supply switching on, 0-control off, 1- control on
4053	R,W	Auto	0,1,2	Self-adaptation algorythm: 0-without self-adaptation, 1-identyfying method, 2-oscillation method
4054... 4773 address of each	R,W	LcYc	1...99	Number of program cycles p Register number r for the program p is equal: r =(p-1)*48+4054
	R,W	bLoh	0...999	Blocking value in the program p$^{1)}$ Register number r for the program p is equal: r =(p-1)*48+4055
	R,W	Cont	0,1	Continuation of the program p after the supply decay: 0-stop, 1-continue, the register number r for the program p is equal: r =(p-1)*48+4056
	R,W	nAxx,	0...999	Change rate of the set value on the segment xx *10 The register number r for program p of the segment x is equal: r =(p-1)*48+(x-1)*3+4057
	R,W	SPxx or tixx	Range depends on the input 0...999	Set value on the end of the segment$^{1)}$, when the change rate > 0 or hold time , when the change rate=0; The register number r for program p of the segment x is equal: r =(p-1)*48+(x-1)*3+4058
	R,W	Eoxx bLxx,	0,1 0,1	Output y in the segment xx (bit y-1): 1-output on, 0-output off. Index for the active deadlock in the segment xx (bit 4) xxx1xxx-1, 1-on, 0-off The number of the register r for the program p of the segment x is equal: r = (p-1)*48+(x-1)*3+4059

4774	R			Device state: bit 0: 1- measured value on the main input is below the lower input range or input short-circuiting bit 1: 1- measured value on the main input is over the upper input range or input opening bit 2: 1- measured value on the additional input is below the lower input range bit 3: 1- measured value on the additional input is over the upper input range bit 4: output state no 1: 0-off, 1-on bit 5: output state no 2: 0-off, 1-on bit 6: output state no 3: 0-off, 1-on bit 7: output state no 4: 0-off, 1-on bit 8: 1- manual control, 0- automatic control bit 9: 1-change of the set value, i.e. Soft-start bit 10: 1- programmed control, 0- constant control bit 11: 1- program execution, 0- program stopped bit 12: 1- program blocking because of a too high deviation bit 13: binary input state: 0- open, 1- short-circuited bit 14 and 15: position of decimal point: 00- without decimal point, 01- decimal on the position 1 10- decimal on the position 2 (see ref. 1))
4775	R			Measured value on the input no 1 1)
4776	R			Measured value on the input no 2 1)
4777	R			Controlled value 1)
4778	R			Set value (actual value) 1)
4779	R	h	0...1000	Control value of the line I *10
4780	R	c	0...1000	Control value of the line II *10
4781	R	n	0...15	Number of the segment currently executed for the programmed control
4782	R	t		Time remaining to the segment end
4783	R	L.		Number of program cycles remaining to the end

1) the value is multiplied by the multiplier depending on the parameter value LcPP (position of the decimal point), i.e.:

when LcPP=0, then the multiplier =1;
when LcPP=1, then the multiplier = 10;
when LcPP=2, then the multiplier = 100.

In the table 2 all register addresses including data of 15 programs are given. To each register address one must add 4000.

Table 2. ***Register addresses concerning programs of the set value.***

Parameter		**Program**														
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
LCYC	Register numbers	54	102	150	198	246	294	342	390	438	486	534	582	630	678	726
Bloh		55	103	151	199	247	295	343	391	439	487	535	583	631	679	727
cont		56	104	152	200	248	296	344	392	440	488	536	584	632	680	728
nA 1		57	105	153	201	249	297	345	393	441	489	537	585	633	681	729
SP2 or ti1		58	106	154	202	250	298	346	394	442	490	538	586	634	682	730
Eout+blok		59	107	155	203	251	299	347	395	443	491	539	587	635	683	731
nA 2		60	108	156	204	252	300	348	396	444	492	540	588	636	684	732
SP2 or ti2		61	109	157	205	253	301	349	397	445	493	541	589	637	685	733
Eout+blok		62	110	158	206	254	302	350	398	446	494	542	590	638	686	734
nA 3		63	111	159	207	255	303	351	399	447	495	543	591	639	687	735
SP3 or ti3		64	112	160	208	256	304	352	400	448	496	544	592	640	688	736
Eout+blok		65	113	161	209	257	305	353	401	449	497	545	593	641	689	737
nA 4		66	114	162	210	258	306	354	402	450	498	546	594	642	690	738
SP4 or ti4		67	115	163	211	259	307	355	403	451	499	547	595	643	691	739
Eout+blok		68	116	164	212	260	308	356	404	452	500	548	596	644	692	740
nA 5		69	117	165	213	261	309	357	405	453	501	549	597	645	693	741
SP5 or ti5		70	118	166	214	262	310	358	406	454	502	550	598	646	694	742
Eout+blok		71	119	167	215	263	311	359	407	455	503	551	599	647	695	743
nA 6		72	120	168	216	264	312	360	408	456	504	552	600	648	696	744
SP6 or ti6		73	121	169	217	265	313	361	409	457	505	553	601	649	697	745
Eout+blok		74	122	170	218	266	314	362	410	458	506	554	602	650	698	746
nA 7		75	123	171	219	267	315	363	411	459	507	555	603	651	699	747
SP7 or ti7		76	124	172	220	268	316	364	412	460	508	556	604	652	700	748
Eout+blok		77	125	173	221	269	317	365	413	461	509	557	605	653	701	749
nA 8		78	126	174	222	270	318	366	414	462	510	558	606	654	702	750
SP8 or ti8		79	127	175	223	271	319	367	415	463	511	559	607	655	703	751
Eout+blok		80	128	176	224	272	320	368	416	464	512	560	608	656	704	752

nA 9	Register numbers	81	129	177	225	273	321	369	417	465	513	561	609	657	705	753
SP9 or ti9		82	130	178	226	274	322	370	418	466	514	562	610	658	706	754
Eout+blok		83	131	179	227	275	323	371	419	467	515	563	611	659	707	755
nA10		84	132	180	228	276	324	372	420	468	516	564	612	660	708	756
SP10 or ti10		85	133	181	229	277	325	373	421	469	517	565	613	661	709	757
Eout+blok		86	134	182	230	278	326	374	422	470	518	566	614	662	710	758
nA11		87	135	183	231	279	327	375	423	471	519	567	615	663	711	759
SP11 or ti11		88	136	184	232	280	328	376	424	472	520	568	616	664	712	760
Eout+blok		89	137	185	233	281	329	377	425	473	521	569	617	665	713	761
NA12		90	138	186	234	282	330	378	426	474	522	570	618	666	714	762
SP12 or ti12		91	139	187	235	283	331	379	427	475	523	571	619	667	715	763
Eout+blok		92	140	188	236	284	332	380	428	476	524	572	620	668	716	764
nA13		93	141	189	237	285	333	381	429	477	525	573	621	669	717	765
SP13 or ti13		94	142	190	238	286	334	382	430	478	526	574	622	670	718	766
Eout+blok		95	143	191	239	287	335	383	431	479	527	575	623	671	719	767
nA14		96	144	192	240	288	336	384	432	480	528	576	624	672	720	768
SP14 or ti14		97	145	193	241	289	337	385	433	481	529	577	625	673	721	769
Eout+blok		98	146	194	242	290	338	386	434	482	530	578	626	674	722	770
nA15		99	147	195	243	291	339	387	435	483	531	579	627	675	723	771
SP15 or ti15		100	148	196	244	292	340	388	436	484	532	580	628	676	724	772
Eout+blok		101	149	197	245	293	341	389	437	485	533	581	629	677	725	773

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 */
{
  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 = 0xFF;  /* initialisation of the
                                      higher CRC byte CRC */
   unsigned char uchCRClo = 0xFF;  /* initialisation of the
                                      lower CRC byte */
   while (usDataLen- -)
   {    uIndex =uchCRChi ^ *puchMsg++; /* CRC
                                      calculation */
       uchCRChi = uchCRClo ^ crc_hi[uIndex];
       uchCRClo = crc_lo[uIndex];
   }
   return(uchCRChi<<8 | uchCRClo);
}
```

```
//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
Tel.: (48-68) 32 95 100 (exchange)
Fax: (48-68) 32 95 101
www.lumel.com.pl
e-mail:lumel@lumel.com.pl

Export Department:
Tel.: (48-68) 329 53 02 or 53 04
Fax: (48-68) 325 40 91
e-mail: export@lumel.com.pl