



RO-SER-PROTOCOL

Hardware-Description

2011 Oktober

INDEX

<u>1. Introduction</u>	4
<u>2. Accessing registers</u>	6
<u>2.1. What are registers ?</u>	6
<u>2.2. Accessing registers with 8, 16, 32 or 64 bit data width</u>	6
<u>2.3. Register assignment</u>	7
<u>3. The transmission protocol</u>	9
<u>3.1. Speed and interface parameters</u>	9
<u>3.2. Structure of the „sending string“</u>	9
<u>3.2.1. Start character</u>	10
<u>3.2.2. MODUL-NO</u>	10
<u>3.2.3. JOB-ID</u>	10
<u>3.2.4. COMMAND</u>	10
<u>3.2.5. WIDTH</u>	11
<u>3.2.6. ADDRESS</u>	11
<u>3.2.7. DATA</u>	11
<u>3.2.8. CHECKSUM</u>	12
<u>3.2.9. END</u>	12
<u>3.3. Structure of the „response string“</u>	13
<u>3.3.1. Response-string „O“ (OK)</u>	13
<u>3.3.2. Response-string „D“ (DATA)</u>	14
<u>3.3.3. Start-character with response „E“ (ERROR)</u>	15
<u>4. Appendix</u>	17
<u>4.1. Revisions</u>	17
<u>4.2. Copyrights and trademarks</u>	18



Introduction



1. Introduction

Communicating with our modules takes place over a serial interface.

The modules responds to a request (sending packet), process the request and responds with a response (receive packet).

The request:

The sending string consists of a string, beginning with a <SOH> character. The content consists of several ASCII-characters (“0”-“9” and “A”-“F”). The end of the string is indicated with a <CR> character.

The response:

After processing the request, the response is sent back. This can either be a simple “**acknowledge**”, “**data**” or an error message.

The structure of the send and response string is explained in detail in the following chapters.

Accessing registers



2. Accessing registers

2.1. What are registers ?

Registers are little scratch-pad like storages, buffering writing or reading data. The data stays stored until they are overwritten or until their power supply is cut off. They have an address range in order to address them. With the aid of the address, you can read from the registers or write to them. The registry access is therefore addressed.

A special feature are the registers of the input state change flags. If they are read, their data will be reset at the same time.

2.2. Accessing registers with 8, 16, 32 or 64 bit data width

Accessing registers can be done in different data widths. The data can be either 8 (byte), 16 (Word), 32 (long) or 64 (eXtralong) bits wide. With the aid of the address, the desired access range is selected. An address refers to 8 bits.

If for example a 32-bit access to address 0004 hex occurs, then 4 x 8 bytes (as a data block) starting with address 0004 hex until address 0007 hex will be read or written.

Subdivided table in 8, 16, 32, and 64 bit register accesses

Access width	Address	00	01	02	03	04	05	06	07
8 Bit	0000 hex	X							
16 Bit	0000 hex	X	X						
32 Bit	0000 hex	X	X	X	X				
32 Bit	0004 hex					X	X	X	X
64 Bit	0000 hex	X	X	X	X	X	X	X	X

When writing or reading, the data is written or read byte by byte. You need to pay attention to the byte order. The byte order starts with the low-byte order (byte order: Little Endian).

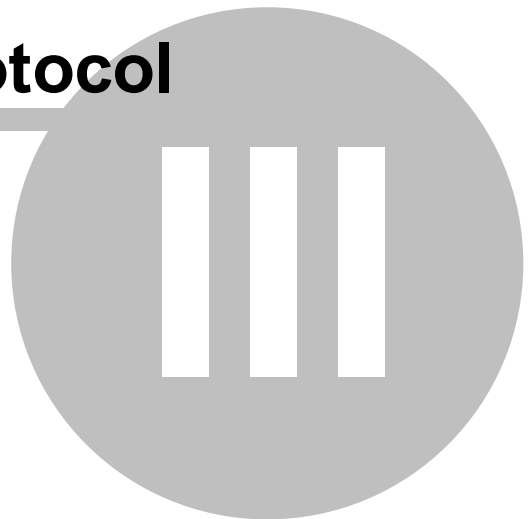
Byte order of the data in the register

Access width	Address	Value	00	01	02	03	04	05	06	07
8 Bit	04	1a					1a			
16 Bit	06	1a1b							1b	1a
32 Bit	00	01020304	04	03	02	01				
32 Bit	04	01020304					04	03	02	01
64 Bit	00	0102030405060708	08	07	06	05	04	03	02	01

2.3. Register assignment

You will find this in the RO-register assignment document.

The transmission protocol



3. The transmission protocol

3.1. Speed and interface parameters

The serial interface is configured with 115200 bauds, 8 data bits, 1 Stop bit and no parity (8,n,1).

3.2. Structure of the „sending string“

The string begins with a <SOH> character (“**Start of Header**”, 01 hex) and ends with a <CR> character (“**Carriage-Return**”, 0D hex).

Before transmission, hexadecimal data is ASCII-encoded. Not encoded are begin and end characters as well as COMMAND and WIDTH. Accessing for example the address 0012 hex, results in transmitting from left to right the 4 ASCII characters “0012” consecutively. ASCII encoded hexadecimal values may only have the numbers from 0..9 and the letters A..F.

Write command example:

The following example writes the data word 0F hex to the address 0012 hex (the sending string is build up of 16 ASCII-characters)

Byte no.	Description	ASCII-character	Hexadecimal
1	Start Of Header (marks begin of string)	<SOH>	01
2, 3	MODULE-NO (=34 hex)	34	33, 34
4, 5	JOB-ID (= 12 hex)	12	31, 32
6	COMMAND	W	57
7	WIDTH (data width)	B	42
8, 9, 10, 11	ADDRESS (= 0012 hex)	0012	30, 30, 31, 32
12, 13	DATA (= 0F hex)	0F	30, 46
14, 15	CHECKSUM (= 29D hex, the lower 8 bits will be only transmitted)	9D	39, 44
16	END (Carriage Return)	<CR>	0D

3.2.1. Start character

The <SOH> character indicates the begin of a data transfer.

3.2.2. MODUL-NO

The MODULE-NO indicates the selected module (the module numbers can be modified using the DIP switches on the modules itself).

3.2.3. JOB-ID

The JOB-ID indicates the actual sending string. Two consecutive sending strings may not have the same JOB-ID.

HINT:

After a successful data transfer, the sending routine should increase the JOB-ID by "1". That way, the successive numbers "0", then "1" .. "255" and again "0" are sent.

3.2.4. COMMAND

With COMMAND, the characters "W" or "R" indicates writing to or reading from registers.

3.2.5. WIDTH

In dependence of the used “**COMMAND**” and data width, a different data length is transferred.

COMMAND	WIDTH	Data width (bits)	Amount of ASCII characters
W	B	8	2
W	W	16	4
W	L	32	8
W	X	64	16
R		0	0

The COMMAND = “**R**” has no influence on the transferred data length. It controls the data length of the receive string

3.2.6. ADDRESS

The registers are 16 bits wide and are represented with 4 hexadecimal values. As the transfer of every hexadecimal value is done with an ASCII character, 4 ASCII characters are needed.

3.2.7. DATA

When a write command is used, the data field present. A read command does not have any data field.

3.2.8. CHECKSUM

CHECKSUM is a simple checksum. It is a hexadecimal summation of all characters (including <SOH>) until the checksum itself. The receiver compares the summation with the transferred checksum. If both are equal, the transfer is seen as error-free.

In hexadecimal:

$01 + 33 + 34 + 31 + 32 + 57 + 42 + 30 + 30 + 31 + 32 + 30 + 46 = 29D$

→ CHECKSUM = 9D hex (2 is the overflow and is therefore left out).

3.2.9. END

The end of the sending string is indicated with a <CR> (Carriage Return, 0D hex).

3.3. Structure of the „response string“

The response string begins with a character (“O”, “D” oder “E”) followed by ASCII characters and ends with a <CR> (Carriage Return, 0d hex).

3.3.1. Response-string „O“ (OK)

This means the response is “ok” and informs the sender that the write command is successfully executed.

“O” - format

Byte no.	Description
1	ASCII-“O” for OK
2	JOB-ID (higher byte)
3	JOB-ID (lower byte)
4	CHECKSUM (higher byte)
5	CHECKSUM (lower byte)
6	END (Carriage Return)

OK-response of the sending string example of chapter 3.2

Byte no.	Description	ASCII-character	Hexadecimal
1	OK	O	4F
2, 3	JOB-ID (= 12 hex)	12	31, 32
4, 5	CHECKSUM (= B2 hex)	B2	42, 32
6	END (Carriage Return)	<CR>	0D

3.3.2. Response-string „D“ (DATA)

A response-string beginning with a “D” indicates response data, resulting from a read command.

“D” - format

Byte no.	Description
1	“D” for DATA (response to a read command)
2	JOB-ID (higher byte)
3	JOB-ID (lower byte)
4,5 ...	n-data
4 + n	CHECKSUM (higher byte)
5 + n	CHECKSUM (lower byte)
6 + n	END (Carriage Return)

With n = 2, 4, 8 or 16 characters (depending on the WIDTH of the sending string)

Command + WIDTH of the sending string and the response data

Read-request		Response-data		
Command	WIDTH	Data width (bits)	Amount charact.	ASCII
R	B	8	2	
R	W	16	4	
R	L	32	8	
R	X	64	16	

3.3.3. Start-character with response „E“ (ERROR)

The last sent sending string was incorrectly received.

“E” - format

Byte no.	Description
1	“E” for error
2	Error code
3	END (Carriage Return)

Error code

Hexadecimal	Description
31	Incorrect command; an invalid COMMAND was used
32	Invalid data length; the length of the sending-string is false
33	Checksum error

Appendix



4. Appendix

4.1. Revisions

Rev 1.00	First issue
Rev 2.00	Design change

4.2. Copyrights and trademarks

Linux is registered trade-mark of Linus Torvalds.

Windows CE is registered trade-mark of Microsoft Corporation.

USB is registered trade-mark of USB Implementers Forum Inc.

LabVIEW is registered trade-mark of National Instruments.

Intel is registered trade-mark of Intel Corporation

AMD is registered trade-mark of Advanced Micro Devices, Inc.