SIEB & MEYER

Drive Amplifier SD2x

Modbus RTU Connection

D-00000348.2
2022-02-18

Copyright

Original instructions, Copyright © 2021 SIEB & MEYER AG.

All Rights Reserved.

This manual or extracts thereof may only be copied with the explicit authorization by SIEB & MEYER AG.

Trademarks

All product, font and company names mentioned in this manual may be trademarks or registered trademarks of their respective companies.

SIEB & MEYER Worldwide

For questions regarding our products and technical problems please contact us.

SIEB & MEYER AG
Auf dem Schmaarkamp 21
21339 Lueneburg
Germany

Phone: +49 4131 203 0
Fax: +49 4131 203 2000
info@sieb-meyer.de
http://www.sieb-meyer.com

SIEB & MEYER Shenzhen Trading Co. Ltd.
Room A208 2/F,
Internet Innovation and Creation services base Building (2),
No.126, Wanxia road, Shekou, Nanshan district,
Shenzhen City, 518067
P.R. China

Tel.: +86 755 2681 1417 / +86 755 2681 2487
Fax: +86 755 2681 2967
info@sieb-meyer.cn
http://www.sieb-meyer.cn

SIEB & MEYER Asia Co. Ltd.
4 Fl, No. 532, Sec. 1
Min-Sheng N. Road
Kwei-Shan Hsiang
333 Tao-Yuan Hsien
Taiwan

Tel.: +886 3 311 5560
Fax: +886 3 322 1224
info@sieb-meyer.tw

1 Introduction

This document describes how to communicate with an SD2x drive via Modbus RTU protocol. At present, Modbus ASCII is not implemented for the device series SD2x yet.

For data transfer via Modbus RTU the RS485 interface of the SD2x drive is used. Within the RS485 network, the drive communicates as slave.

For information on the device series SD2x refer to the according hardware and software documentation.

The following chapters provide information on the parameterization of the drive, the function codes for the communication, the process data objects and the drive objects as well as some examples.

More information on Modbus you can find on the website www.modbus.org.

Note

To configure your own I/O data, the Modbus master must support the function “CAN Encapsulation” so that the parameters can be read and written independently of the process image. If the master does not support this function, you can only transmit the default process data objects using the standard Modbus function codes.

2 Parameterization

The following chapters describe how to set the parameters of your SD2x drive in the software *drivemaster2* for the Modbus communication.

2.1 Drive Control

Activate the Modbus communication on the parameter page “Drive control”: Select the control channel and the setpoint channel “Modbus”:

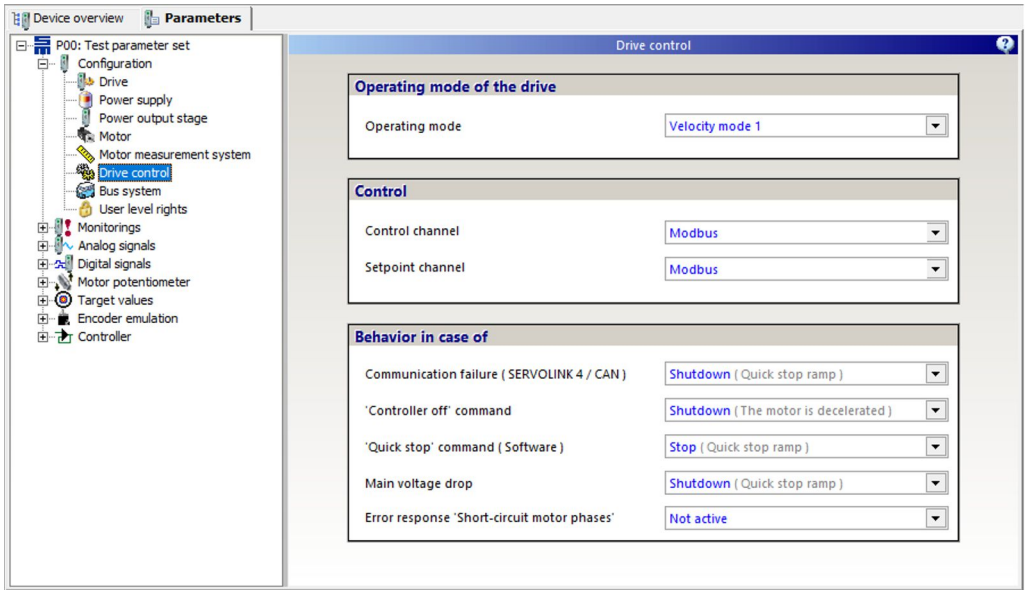


Fig. 1: Parameterization of the drive control

Note

After activating the Modbus communication, you cannot use the RS485 interface of your SD2x drive (connector X74) for drive parameterization anymore. This interface is then used for Modbus communication only.

For information on the pin assignment of the connector X74 refer to the hardware description.

After activating the Modbus communication in the drive control, you can set the baud rate and parity of the interface as described in the following chapters.

2.2 Bus System

On the parameter page “Bus system” you can set the parameters for the Modbus communication:

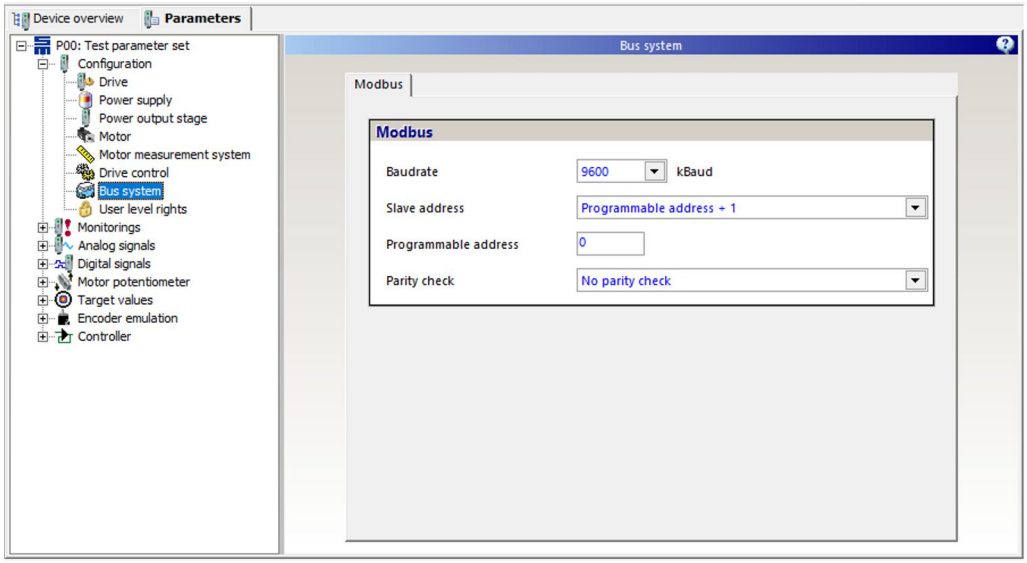


Fig. 2: Parameterization of the Modbus communication

2.2.1 Baud rate

You can set the following values for the baud rate:

- 9600 kBaud
- 14400 kBaud
- 19200 kBaud
- 28800 kBaud
- 38400 kBaud
- 57600 kBaud
- 115200 kBaud

2.2.2 Slave Address

The slave address is the address assigned to the SD2x drive in the bus system.

Via the parameter you can select how to calculate the slave address. The following settings are available:

- Address selector switch + 1
- Address selector switch + programmable address + 1
- Programmable address + 1

Address selector switch + 1

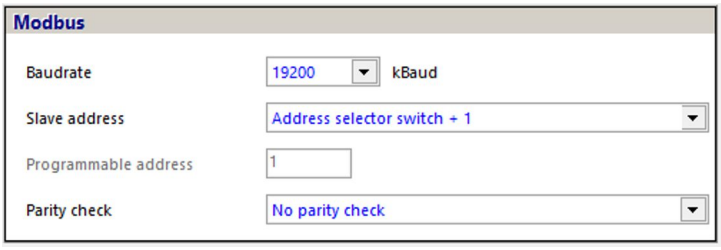
The slave address is set via the ID switch on the front panel of the device. Since you can set the ID switch to a value between 0..15 but the Modbus protocol does not allow the value 0, the value 1 is automatically added. In addition, double-axis devices of the series SD2 require two addresses – one for each axis. For this reason, the value of the ID switch is multiplied by 2 at first. That means: single-axis devices as well as the A-axes of double-axis devices get odd slave addresses *(ID switch × 2 + 1)* and the B-axes of double-axis devices get even slave addresses *(ID switch × 2 + 2)*.

The following table shows the Modbus addresses according to the position of the ID switch:

Position of ID switch	Modbus address	
	Single-axis device	Double-axis device
0	1	1 and 2
1	3	3 and 4
2	5	5 and 6
3	7	7 and 8
4	9	9 and 10
5	11	11 and 12
6	13	13 and 14
7	15	15 and 16
8	17	17 and 18
9	19	19 and 20
A	21	21 and 22
B	23	23 and 24
C	25	25 and 26
D	27	27 and 28
E	29	29 and 30
F	31	31 and 32

Example:

The ID switch of the device is set to position 5.



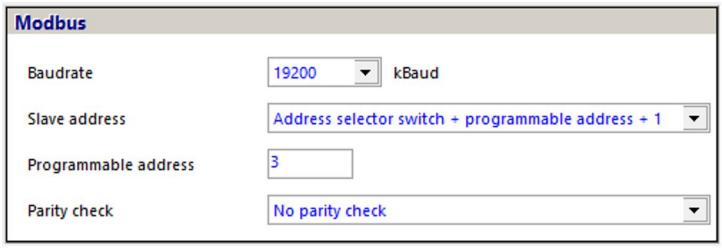
With theses settings a single-axis devices has the Modbus address 11.

Address selector switch + programmable address + 1

The slave address is calculated by the sum of the ID switch value and the value entered in the input field “Programmable address ”. In order to remain within the range of the Modbus ID, you must enter a value between 0..253. Since the number 0 is not allowed in the Modbus protocol, 1 is automatically added to the result.

Example:

The ID switch of the device is set to position 5.

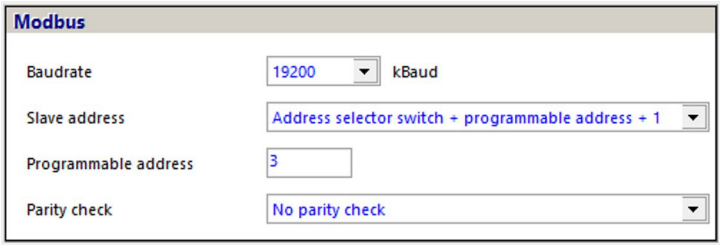


With theses settings a single-axis devices has the Modbus address 11+ 3 = 14.

Programmable address + 1

The slave address is entered in the input field “Programmable address”. Here you can set a value in the range of 0..253. Since the number 0 is not allowed in the Modbus protocol, 1 is automatically added to the result. Thus, the slave address is in the range of 1..254. The ID switch on the front panel of the device has no function

Example:



Regardless of weather the drive is a single-axis device or a double-axis device, it will communicate with the slave address 3 + 1 = 4.

2.2.3 Parity check

The drive supports the following parity settings:

- No parity check (1 stop bit)
- Even parity (1 stop bit)
- Odd parity (1 stop bit)

3 Function Codes

The following function codes are supported:

Function group	Name	Function code
Data access 16 bit	Read Holding Registers	03 (03_h)
	Read Input Register	04 (04_h)
	Write Single Register	06 (06_h)
	Write Multiple Register	16 (10_h)
	Read/Write Multiple Registers	23 (17_h)
Diagnosis	Read Error Counter	08 (08_h)
CAN over Modbus	Encapsulated CAN Message	43 ($2B_h$)

3.1 FC 03 Read Input Register / FC 04 Read Holding Register

These function codes can read one or more 16-bit registers. They are used to read the PDO data from the drive.

Request:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	03_h / 04_h
Register address	1 byte high	0000_h to $FFFF_h$
	1 byte low	
Register count	1 byte high	n = 1 to 48 (30_h)
	1 byte low	
CRC	1 byte low	
	1 byte high	

Response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	03_h / 04_h
Byte count	1 byte	2 × n
Register values	1 byte high	Register (1) value
	1 byte low	
	1 byte high	Register (2) value
	1 byte low	
	⋮	⋮
	1 byte high	Register (n) value
	1 byte low	
CRC	1 byte low	
	1 byte high	

Error response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	83_h / 84_h
Error number	1 byte	See Communication Errors (p. 16)
CRC	1 byte low	
	1 byte high	

Generally, input registers are only readable while holding registers are readable and writable. With the function codes FC-03 and FC-04 the drive does not differentiate between *read only* and *read/write* registers. But both function codes are implemented for compatibility with the Modbus protocol; they are treated identically.

3.2 FC 06 Write Single Register

This function code can write a single 16-bit register. It is used to write the PDO data to the drive.

Request:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	06_h
Register address	1 byte high	0000_h to $FFFF_h$
	1 byte low	
Register value	1 byte high	0000_h to $FFFF_h$
	1 byte low	
CRC	1 byte low	
	1 byte high	

Response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	06_h
Register address	1 byte high	0000_h to $FFFF_h$
	1 byte low	
Register value	1 byte high	0000_h to $FFFF_h$
	1 byte low	
CRC	1 byte low	
	1 byte high	

Error response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	86_h
Error number	1 byte	See Communication Errors (p. 16)
CRC	1 byte low	
	1 byte high	

3.3 FC 08 Diagnostics

This function code enables testing and diagnosis of the Modbus communication. The “Diagnostic function” with a length of 2 bytes can trigger diagnostic tests or request values. Some diagnostic functions use data in the request/response telegram. These bytes must be transmitted even when the data are not used so that the diagnostics telegram has always the same size.

Request:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	08_h

Name	Size	Value
Diagnostic function	1 byte high	0, 10, 11, 12, 13, 14, 15
	1 byte low	
Data	1 byte high	0000_h to $FFFF_h$
	1 byte low	
CRC	1 byte low	
	1 byte high	

Response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	08_h
Diagnostic function	1 byte high	0, 10, 11, 12, 13, 14, 15
	1 byte low	
Data	1 byte high	0000_h to $FFFF_h$
	1 byte low	
CRC	1 byte low	
	1 byte high	

Error response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	88_h
Error number	1 byte	See Communication Errors (p. 16)
CRC	1 byte low	
	1 byte high	

Diagnostic functions

The following diagnostic functions are available:

Function code	Function	Request data	Response data
0 (00_h)	Return Query Data The received telegram is returned. Thereby the data bytes received in the data field are returned to the sender.	Any values	The received values
10 ($0A_h$)	Clear Counters and Diagnostic Register All diagnosis registers are reset to 0.	–	–
11 ($0B_h$)	Return Bus Message Count Indicates the number of messages detected by the system.	–	Bus Message Counter
12 ($0C_h$)	Return Bus Communication Error Count Indicates the number of CRC errors.	–	CRC Error Counter
13 ($0D_h$)	Return Bus Exception Error Count Indicates the number of errors reported by the system.	–	Exception Error Counter
14 ($0E_h$)	Return Slave Message Count Indicates the number of messages sent by the system.	–	Message Counter
15 ($0F_h$)	Return Slave No Response Count Indicates the number of messages sent by the system with no response.	–	No Response Counter

3.4 FC 16 Write Multiple Registers

This function code can write several 16-bit registers. It is used to write the PDO data to the drive.

Request:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	10_h
Register address	1 byte high	0000_h to $FFFF_h$
	1 byte low	
Register count	1 byte high	n = 1 to 48 (30_h)
	1 byte low	
Byte count	1 byte	2 × n
Register values	1 byte high	Register (1) value
	1 byte low	
	1 byte high	Register (2) value
	1 byte low	
	⋮	⋮
	1 byte high	Register (n) value
	1 byte low	
CRC	1 byte low	
	1 byte high	

Response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	10_h
Starting address	1 byte high	0000_h to $FFFF_h$
	1 byte low	
Register count	1 byte high	n = 1 to 48 (30_h)
	1 byte low	
CRC	1 byte low	
	1 byte high	

Error response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	90_h
Error number	1 byte	See Communication Errors (p. 16)
CRC	1 byte low	
	1 byte high	

3.5 FC 23 Read/Write Multiple Registers

This function code can write one or more 16-bit registers while reading other registers. It is used to write and read the PDO data of the drive.

Request:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	17_h

Name	Size	Value
Read register address	1 byte high	0000_h to $FFFF_h$
	1 byte low	
Read register count	1 byte high	n = 1 to 48 (30_h)
	1 byte low	
Write register address	1 byte high	0000_h to $FFFF_h$
	1 byte low	
Write register count	1 byte high	m = 1 to 48 (30_h)
	1 byte low	
Byte count	1 byte	2 × m
Register values	1 byte high	Register (1) value
	1 byte low	
	1 byte high	Register (2) value
	1 byte low	
	⋮	⋮
	1 byte high	Register (n) value
	1 byte low	
CRC	1 byte low	
	1 byte high	

Response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	17_h
Byte count	1 byte	2 × n
Register values	1 byte high	Register (1) value
	1 byte low	
	1 byte high	Register (2) value
	1 byte low	
	⋮	⋮
	1 byte high	Register (n) value
	1 byte low	
CRC	1 byte low	
	1 byte high	

Error response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	97_h
Error number	1 byte	See Communication Errors (p. 16)
CRC	1 byte low	
	1 byte high	

3.6 FC 43 Encapsulated Interface Transport

This function block allows easy access to the object dictionary of the drive.

Presently, only 16-bit and 32-bit objects can be accessed. Access to array and string objects is not implemented yet..

Request:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	$2B_h$

Name	Size	Value
MEI type	1 byte	$0D_h$
Control code	1 byte	00_h / 80_h read / write
Reserved field	1 byte	00_h reserved
Node-ID	1 byte	01_h to $7F_h$
Index	1 byte high	0000_h to $FFFF_h$ object number
	1 byte low	
Subindex	1 byte	00_h to FF_h object subindex
Address offset	1 byte high	0000_h data offset
	1 byte low	
Byte count	1 byte high	n = 0000_h to 0004_h bytes
	1 byte low	
Data values	1 byte #0	Byte (1) value
	1 byte #1	
	1 byte #2	
	1 byte #3	Byte (n) value
CRC	1 byte low	
	1 byte high	

Response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	$2B_h$
MEI type	1 byte	$0D_h$
Control code	1 byte	00_h / 80_h read / write
Reserved field	1 byte	00_h reserved
Node-ID	1 byte	01_h to $7F_h$
Index	1 byte high	0000_h to $FFFF_h$ object number
	1 byte low	
Subindex	1 byte	00_h to FF_h object subindex
Address offset	1 byte high	0000_h data offset
	1 byte low	
Byte count	1 byte high	n = 0000_h to 0004_h bytes
	1 byte low	
Data values	1 byte #0	Byte (1) value
	1 byte #1	
	1 byte #2	
	1 byte #3	Byte (n) value
CRC	1 byte low	
	1 byte high	

Error response:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	AB_h
Error number	1 byte	01_h, 02_h, 03_h, 04_h See Communication Errors (p. 16)
CRC	1 byte low	
	1 byte high	

Response, when an error occurs during object access:

Name	Size	Value
Slave address	1 byte	
Function code	1 byte	AB_h

Name	Size	Value
Error number	1 byte	FF_h extended exception
Extended exception length	1 byte high	6
	1 byte low	
MEI type	1 byte	$0D_h$
Exception code	1 byte	CE_h
SDO error code	1 byte #0	Byte (1) value
	1 byte #1	
	1 byte #2	
	1 byte #3	Byte (4) value see SDO Errors (p. 16)
CRC	1 byte low	
	1 byte high	

Data description

- **MEI-type:**
  Modbus Encapsulated Interface (MEI) is 0Dh and selects CANopen via Modbus Transport. This is the only MEI-type supported by the drive.
- **Control code:**

Bit	Description	Use
0..6	not used	Set to zero
7	Read or write data	0 = read; 1 = write

- **Node-ID:**
  The Node-ID has the same value as the slave address.
- **Index:**
  The index contains the object number from the drive object dictionary and matches the index in the software *drivemaster2*. The index starts with 0 and is consecutively numbered. You can find the list of objects in the object browser of the *drivemaster2* user interface.
  Objects from the CiA standards DS301 (object number 0x2000 and higher) and DS402 (object number 0x6000 and higher) are not supported.
- **Subindex:**
  The subindex is used to select data from array and string objects. For simple objects, set the subindex to zero.
- **Address offset:**
  The address offset is not used for SD2x drives. Set the address offset to zero.
- **Byte count:**
  These bytes indicate the number of bytes to be read or written.
- **Data values:**
  – Request telegram: data to be written
  – Response telegram: data to be read
  Contrary to all other telegrams in the Modbus protocol, in the Encapsulated Interface Transport telegram the data are transmitted in little-endian format. This results in the following byte orders for 16-bit and 32-bit values:
  16-bit value: example 0x1234

Byte	Byte #0	Byte #1	Byte #2	Byte #3
Value	0x34	0x12	–	–

  32-bit value: example 0x12345678

Byte	Byte #0	Byte #1	Byte #2	Byte #3
Value	0x78	0x56	0x34	0x12

- **Error number:**
  The error number can return one of the values 01_h, 02_h, 03_h, 04_h and FF_h. For the error descriptions refer to the table Communication Errors (p. 16). The value FF_h means that an extended error number is sent.
- **Exception code:**
  Presently, only the code CE_h is implemented. This means, that an object access error code (SDO error code) with a length of 4 bytes follows.
- **SDO error code:**
  SDO error codes are object access error codes and have a length of 4 bytes, see SDO Errors (p. 16).

3.7 Error codes for the communication

The following error codes can be transmitted:

Error code	Name	Meaning
01	Illegal Function	The Modbus function code received in the request is not implemented.
02	Illegal Data Address	The data address received in the request is not valid.
03	Illegal Data Address	The data value received in the request is out of the data range.
04	Slave Device Failure	An unrecoverable error occurred while the device was trying to perform the requested action.
FF_h	SDO error	An object access error occured. Therefore an extended error code was sent, see SDO Errors (p. 16).

3.8 Extended Error Codes (SDO Errors)

The following table shows object errors that may occur during access using the function FC 43 “Encapsulated Interface Transport”:

Error code	Description
$FFFF0003_h$	Client/server command not valid or unknown
$FFFF0007_h$	Out of memory
$FFFF0008_h$	Unsupported access to an object
$FFFF0009_h$	Attempt to read a write-only object
$FFFF000A_h$	Attempt to write in a read-only object
$FFFF000B_h$	Object does not exist in the object dictionary
$FFFF000C_h$	Object cannot be mapped to the PDO
$FFFF000D_h$	Number and length of the objects to be mapped exceed PDO length
$FFFF000E_h$	General incompatibility of the parameters
$FFFF000F_h$	General internal incompatibility in the device
$FFFF0010_h$	Access failed due to hardware error
$FFFF0011_h$	Data type not correct, length of the service parameter not correct
$FFFF0012_h$	Data type not correct, length of the service parameter too long
$FFFF0013_h$	Data type not correct, length of the service parameter too short
$FFFF0014_h$	Subindex does not exist
$FFFF0015_h$	Value range of the parameter exceeded (only for write access)
$FFFF0016_h$	Value of the written parameter too high
$FFFF0017_h$	Value of the written parameter too low
$FFFF0018_h$	Maximum value is less than minimum value
$FFFF0019_h$	General error
$FFFF001A_h$	Data cannot be transmitted or saved in the application

Error code	Description
FFFF001B$_h$	Data cannot be transmitted or saved in the application due to the local control
FFFF001C$_h$	Data cannot be transmitted or saved in the application due to the present device status
FFFF001D$_h$	Dynamic generation of the object dictionary failed or no object directory available (e.g. object directory generated from file and generation fails because of a file error)
FFFF001E$_h$	Requested data object is too large to fit in a single message

4 Process Data

The following process data objects (PDO) are available:

- RxPDO: receive setpoint values
- TxPDO: transmit actual values

The process images start at the following Modbus register addresses:

Modbus register address	Name	Description
2000_d	TxPDO (Input Register)	Actual values
3000_d	RxPDO (Holding Register)	Setpoint values

The following chapters contain the default configuration of the process data objects.

4.1 Default RxPDO

Modbus register	Name	Size	Description
3000_d	Control Word	16 Bit	Control word: start / stop
3001_d	Target Speed	16 Bit	Speed setpoint in 0.1 Hz
3002_d	Torque Limit Iq	16 Bit	Current limitation in 0.1 A
3003_d	Watchdog	16 Bit	Watchdog counter

4.2 Default TxPDO

Modbus register	Name	Size	Description
2000_d	Status Word	16 Bit	Status word: enabled / error
2001_d	Actual Speed	16 Bit	Speed in 0.1 Hz
2002_d	Torque Limit Iq	16 Bit	Current limitation in 0.1 A
2003_d	Act Apparent Current	16 Bit	Current in 0.1 A
2004_d	Act Output Power	16 Bit	Power in 0.1 kW
2005_d	Error Code	16 Bit	Error code
2006_d	Temp Power Stage	16 Bit	Temperature of power output stage in 0.1 °C
2007_d	Temp Motor	16 Bit	Temperature of motor in 0.1 °C
2008_d	Output Voltage	16 Bit	Output voltage in 0.1 V
2009_d	DC Voltage	16 Bit	DC voltage in 0.1 V
2010_d	Hardware ID	32 Bit	Hardware ID
2012_d	Hardware Version	32 Bit	Hardware version
2014_d	Software ID	32 Bit	Software ID
2016_d	Software Version	32 Bit	Software version
2018_d	Max Iq Current	16 Bit	Max. current Iq in 0.1 A
2019_d	Max Speed	16 Bit	Max. speed in 0.1 Hz
2020_d	Speed Scaling	32 Bit	Speed scaling in 0.001 rpm
2022_d	Current Scaling	32 Bit	Current scaling in 0.1 A

4.3 Byte Order of Process data

The process data are transmitted in big-endian format.

The following byte order for 16-bit and 32-bit values applies to all process data of the function codes FC 03, FC 04, FC 16 and FC 23.

Note

For the function code FC 43, the byte order is different because the data are transmitted in little-endian format (see chapter 3.6 “FC 43 Encapsulated Interface Transport”, page 13).

16-bit register

For a 16-bit register, the following byte order results in the serial data.

Example = 0x1234

	16-bit register			
Byte	**Byte(0)**	**Byte(1)**		
Value	0x12	0x34		

32-bit register

For 32-bit values, two 16-bit registers are combined, whereas register(x) is located at address x and register(x+1) is located at the next address. The following byte order results.

Example = 0x12345678

	16-bit register(x)		16-bit register(x+1)	
Byte	**Byte(0)**	**Byte(1)**	**Byte(0)**	**Byte(1)**
Value	0x12	0x34	0x56	0x78

5 Object Dictionary

Control Word

The Control Word represents the object 0x6040 from the drive profile DS402.

Index	68_d
Name	Control Word
Object code	VAR
Data type	Unsigned 16

Access	RW
PDO mapping	Possible
Unit	Bits coded according to DS402 standard
Value range	0 … 65535
Default value	0
Index acc. to DS402	0x6040

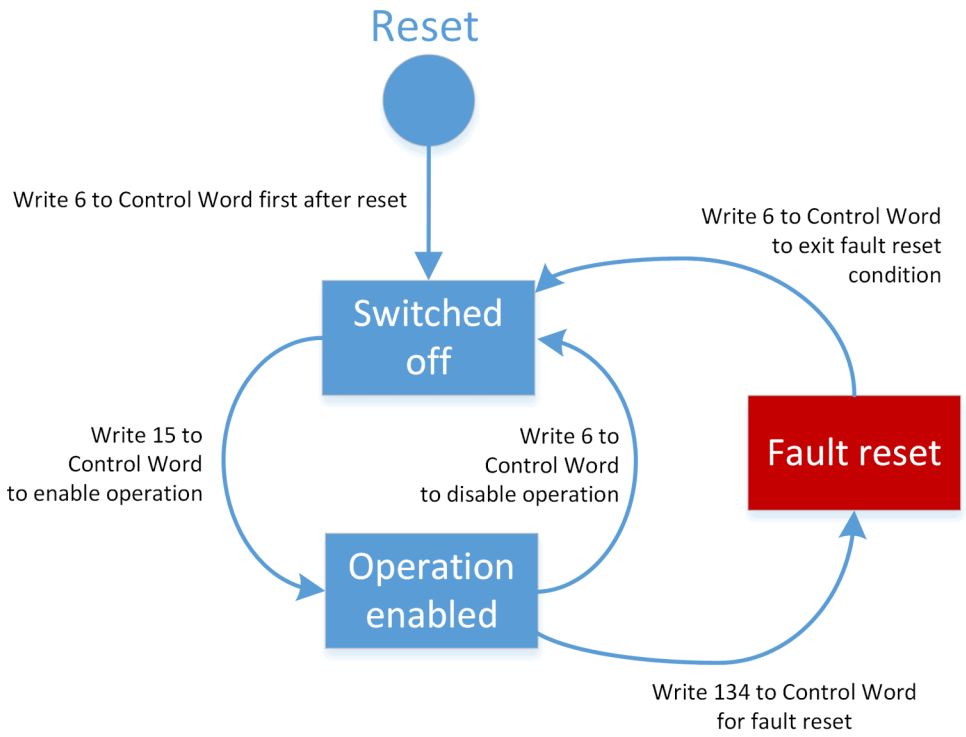


Fig. 3: Drive operation via Control Word

The control word is described in the documentation “Drive System SD2 – Device Control” (see chapter “Structure of the Control Word (Object 68_D)”.

Target Velocity

Index	210_d
Name	Target Velocity
Object code	VAR
Data type	Integer 16

Access	RW

PDO mapping	Possible
Unit	0.1 Hz (10 = 1 Hz)
Value range	(-2^{15}) … $(2^{15}-1)$
Default value	0
Index acc. to DS402	0x60FF

The speed setpoint is indicated in 0.1 Hz.

Torque Limit Iq

Index	209_d
Name	Target Torque Limit
Object code	VAR
Data type	Integer 16

Access	RW
PDO mapping	Possible
Unit	0.1 A (10 = 1 A)
Value range	(-2^{15}) … $(2^{15}-1)$
Default value	0
Index acc. to DS402	–

The current limitation Iq is indicated in 0.1 A.

The current limitation is set by the control. It is limited by the drive peak current. The object represents the object 209 from the *drivemaster2* object list that is described in the documentation “Drive System SD2 – Device Control”.

Watchdog

Index	–
Name	Watchdog Counter
Object code	VAR
Data type	Unsigned 16

Access	RW
PDO mapping	Possible
Unit	1 ms
Value range	0 … 65535
Default value	0
Index acc. to DS402	–

The Watchdog register can deactivate the drive when the communication between control and drive is interrupted. For this purpose, enter a timeout in the register. Within this timeout period the register must be rewritten. If the time expires, the drive stops with the error message "Heartbeat/Watchdog". To deactivate watchdog monitoring, enter the value 0 in the register.

Status Word

The Status Word represents the object 0x6041 from the drive profile DS402.

Index	67_d
Name	Status Word
Object code	VAR
Data type	Unsigned 16

Access	RO
PDO mapping	Possible
Unit	Bits coded according to DS402 standard
Value range	0 … 65535
Default value	0
Index acc. to DS402	0x6041

The status word is described in the documentation “Drive System SD2 – Device Control” (see chapter “Structure of the Status Word (Object 67_D)”.

The most commonly used bits are:

Bit	Name	Description
0	Ready to switch on	Drive is ready to switch on
2	Operation enabled	Drive operation is enabled
3	Fault	An error occurred

Actual Speed

Index	168_d
Name	Actual Speed
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 Hz (10 = 1 Hz)
Value range	(-2^{15}) … $(2^{15}-1)$
Default value	0
Index acc. to DS402	–

The actual speed is indicated in 0.1 Hz.

Actual Apparent Current

Index	431_d
Name	Actual Apparent Current
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 A (10 = 1 A)
Value range	(-2^{15}) … $(2^{15}-1)$
Default value	0
Index acc. to DS402	–

The actual apparent current is indicated in 0.1 A.

Actual Output Power

Index	390_d
Name	Actual Output Power
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 kW (10 = 1 kW)
Value range	(-2^{15}) … $(2^{15}-1)$
Default value	0
Index acc. to DS402	–

The actual output power is indicated in 0.1 kW.

Error Code

Index	70_d
Name	Error Code
Object code	VAR
Data type	Unsigned 16

Access	RO
PDO mapping	Possible
Unit	–
Value range	(-2^{15}) ... $(2^{15}-1)$
Default value	0
Index acc. to DS402	0x603F

For a description of the error codes refer to the document “Drive System SD2 – Device Control”, chapter “List of Drive Error Messages”.

Temperature Power Stage

Index	41_d
Name	Temperature Power Stage
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 °C (10 = 1 °C)
Value range	(-2^{15}) … $(2^{15}-1)$
Default value	0
Index acc. to DS402	–

The temperature of the power output stage is indicated in 0.1 °C.

Temperature Motor

Index	63_d
Name	Temperature Motor
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 °C (10 = 1 °C)
Value range	(-2^{15}) … $(2^{15}-1)$
Default value	0

Index acc. to DS402	–

The temperature of the motor is indicated in 0.1 °C.

Output Voltage

Index	434_d
Name	Output Voltage
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 V (10 = 1 V)
Value range	$(-2^{15}) \dots (2^{15}-1)$
Default value	0
Index acc. to DS402	–

The output voltage is indicated in 0.1 V.

DC Voltage

Index	33_d
Name	DC Voltage
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 V (10 = 1 V)
Value range	$(-2^{15}) \dots (2^{15}-1)$
Default value	0
Index acc. to DS402	–

The DC voltage is indicated in 0.1 V.

Hardware ID

Index	2_d
Name	Hardware ID
Object code	VAR
Data type	Unsigned 32

Access	RO
PDO mapping	Possible
Unit	–
Value range	$(-2^{31}) \dots (2^{31}-1)$
Default value	0
Index acc. to DS402	–

The hardware ID contains an identification code for the device type coded as 32-bit value.

Hardware Version

Index	5_d
Name	Hardware Version
Object code	VAR
Data type	Unsigned 32

Access	RO
PDO mapping	Possible
Unit	–
Value range	$(-2^{31}) \dots (2^{31}-1)$
Default value	0
Index acc. to DS402	–

The hardware version is coded as a 32 bit value, divided into a 16-bit part before the decimal point and a 16-bit decimal part.

Example: 0x00020010 is version 2.16

Software ID

Index	16_d
Name	Software ID
Object code	VAR
Data type	Unsigned 32

Access	RO
PDO mapping	Possible
Unit	–
Value range	$(-2^{31}) \dots (2^{31}-1)$
Default value	0
Index acc. to DS402	–

The software ID contains an identification code for the drive software coded as 32-bit value.

Software Version

Index	17_d
Name	Software Version
Object code	VAR
Data type	Unsigned 32

Access	RO
PDO mapping	Possible
Unit	–
Value range	$(-2^{31}) \dots (2^{31}-1)$
Default value	0
Index acc. to DS402	–

The software version is coded as a 32 bit value, divided into a 16-bit part before the decimal point and a 16-bit decimal part.

Example: 0x00020010 is version 2.16

Max Iq Current

Index	171_d
Name	Max Iq Current
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0.1 A (10 = 1 A)
Value range	$(-2^{15}) \dots (2^{15}-1)$
Default value	0
Index acc. to DS402	–

The maximum drive current Iq is indicated in 0.1 A.

This object represents the object 171 from the *drivemaster2* object list and contains the currently permissible maximum current of the current controller. This value includes any settings for the current limitation made in the drive parameters.

Max Speed

Index	120_d
Name	Max Speed
Object code	VAR
Data type	Integer 16

Access	RO
PDO mapping	Possible
Unit	0x3FFF = Speed Scaling
Value range	$(-2^{15}) \dots (2^{15}-1)$
Default value	0
Index acc. to DS402	–

The maximum speed is indicated according to 0x3FFF = Speed Scaling object.

Speed Scaling

Index	177_d
Name	Speed Scaling
Object code	VAR
Data type	Integer 32

Access	RO
PDO mapping	Possible
Unit	0.001 rpm
Value range	$(-2^{31}) \dots (2^{31}-1)$
Default value	0
Index acc. to DS402	–

The speed scaling is indicated in 0.001 rpm.

Current Scaling

Index	182_d

Name	Current Scaling
Object code	VAR
Data type	Integer 32

Access	RO
PDO mapping	Possible
Unit	0.001 A
Value range	$(-2^{31}) \dots (2^{31}-1)$
Default value	0
Index acc. to DS402	–

The current scaling is indicated in 0.001 A.

6 Examples

6.1 Example 1: Drive Operation

The following example shows the sequence of Modbus protocols that is required to operate the drive.

Task: After drive switch-on, the motor shall rotate at a speed of 100 Hz. Then, the motor is to be stopped again. During operation the drive function must be monitored. This task requires writing to the registers 3000 to 3003 of RxPDO.

The module address of the drive is set to 1, see chapter 2.2.2 “Slave Address”, page 6.

Shutdown drive

After booting the drive, the command “shutdown” is written into the control word.

The control sends:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	10_h	Write Multiple Registers (FC 16)
Register address	1 byte high	$0B_h$	Register $0BB8_h = 3000_d$
	1 byte low	$B8_h$	
Register count	1 byte high	00_h	4 registers
	1 byte low	04_h	
Byte count	1 byte	08_h	8 byte
Control Word	1 byte high	00_h	Shutdown command
	1 byte low	06_h	
Target Speed	1 byte high	03_h	Speed setpoint = 100 Hz
	1 byte low	$E8_h$	
Torque Limit Iq	1 byte high	00_h	Current limitation = 10 A
	1 byte low	64_h	
Watchdog	1 byte high	01_h	Watchdog timeout = 500 ms
	1 byte low	$F4_h$	
CRC	1 byte low	48_h	CRC = 0xFC48
	1 byte high	FC_h	

The drive sends:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	10_h	Write Multiple Registers (FC 16)
Register address	1 byte high	$0B_h$	Register $0BB8_h = 3000_d$
	1 byte low	$B8_h$	
Register count	1 byte high	00_h	4 registers
	1 byte low	04_h	
CRC	1 byte low	43_h	CRC = 0xCB43
	1 byte high	CB_h	

The drive signals the status “ready to switch on” now. You can find the status on the diagnosis pages of the *drivemaster2* software. As an alternative, the status register can be read.

Since the watchdog is activated, the watchdog register must be rewritten at least every 500 ms. Else the drive switches to the error status. You can realize that by sending the telegram above repeatedly. It is advantageous to send the telegram every 200 ms so that the watchdog is not triggered when a single transmission failed.

Enable operation

Then, the command “enable operation” is sent.

The control sends:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	10_h	Write Multiple Registers (FC 16)
Register address	1 byte high	$0B_h$	Register $0BB8_h = 3000_d$
	1 byte low	$B8_h$	
Register count	1 byte high	00_h	4 registers
	1 byte low	04_h	
Byte count	1 byte	08_h	8 byte
Control Word	1 byte high	00_h	Enable operation command
	1 byte low	$0F_h$	
Target Speed	1 byte high	03_h	Speed setpoint = 100 Hz
	1 byte low	$E8_h$	
Torque Limit Iq	1 byte high	00_h	Current limitation = 10 A
	1 byte low	64_h	
Watchdog	1 byte high	01_h	Watchdog timeout = 500 ms
	1 byte low	$F4_h$	
CRC	1 byte low	48_h	CRC = 0xFC48
	1 byte high	FC_h	

The drive responds:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	10_h	Write Multiple Registers (FC 16)
Register address	1 byte high	$0B_h$	Register $0BB8_h = 3000_d$
	1 byte low	$B8_h$	
Register count	1 byte high	00_h	4 registers
	1 byte low	04_h	
CRC	1 byte low	43_h	CRC = 0xCE43
	1 byte high	CE_h	

The drive operates the motor with a speed of 100 Hz now.

6.2 Example 2: Encapsulated Interface Transport

The following example shows how to read the output stage temperature via Modbus.

The output stage temperature is also displayed in the *drivemaster2* software on the diagnosis page “Actual drive values”:

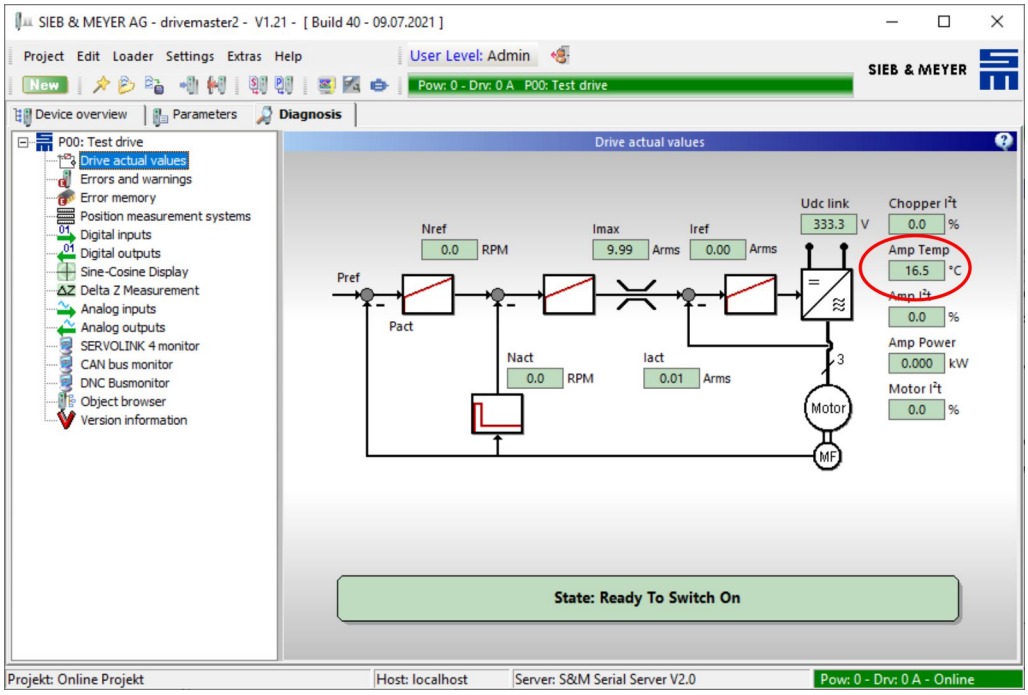


Fig. 4: Output stage temperature on the page “Actual drive values”

In addition, you can find the output stage temperature in the object browser of the *drivemaster2* software. For this purpose you must load the object “POWER_STAGE_TEMPERATURE_ACTUAL” in the object browser:

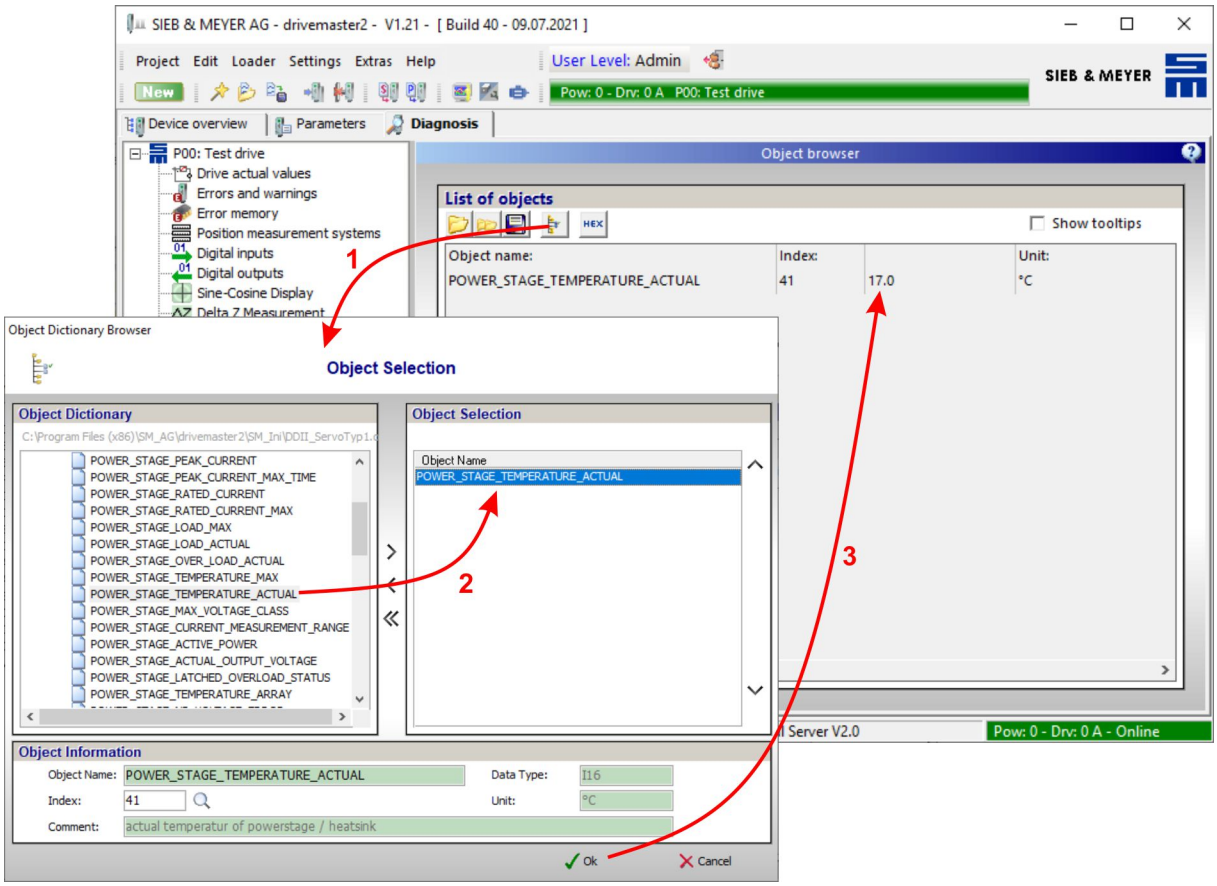


Fig. 5: Display output stage temperature in object browser

In the object browser you can find the properties of the object:

- object index: 41
- data type: 16 bit
- subindex: 0 (The subindex is only used for array objects. Therefore you must set it to 0.)

Task: Using function code 43 “Encapsulated Interface Transport” the object with the properties listed above shall be read.

The module address of the drive is set to 1, see chapter 2.2.2 “Slave Address”, page 6.

To read the object, the control sends the following telegram:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	$2B_h$	Encapsulated Interface Transport
MEI type	1 byte	$0D_h$	Modbus encapsulated Interface type
Control code	1 byte	00_h	Read
Node-ID	1 byte	01_h	Node-ID 1
Index	1 byte high	00_h	0029_h = object 41_d
	1 byte low	29_h	
Subindex	1 byte	00_h	0

Name	Size	Value	Description
Address offset	1 byte high	00_h	0
	1 byte low	00_h	
Byte count	1 byte high	00_h	Object 41_d is a 16-bit object
	1 byte low	02_h	
CRC	1 byte low	$C1_h$	CRC = 0x2BC1
	1 byte high	$2B_h$	

The drive sends:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	$2B_h$	Encapsulated Interface Transport
MEI type	1 byte	$0D_h$	Modbus encapsulated Interface type
Control code	1 byte	00_h	Read
Node-ID	1 byte	01_h	Node-ID 1
Index	1 byte high	00_h	0029_h = object 41_d
	1 byte low	29_h	
Subindex	1 byte	00_h	0
Address offset	1 byte high	00_h	0
	1 byte low	00_h	
Byte count	1 byte high	00_h	Object 41_d is a 16-bit object
	1 byte low	02_h	
Data values	1 byte #0	00_h	$00AA_h$ = 170_d = 17.0 °C
	1 byte #1	AA_h	
CRC	1 byte low	$D1_h$	CRC = 0xDFD1
	1 byte high	DF_h	

The output stage temperature 17.0 °C is returned.

6.3 Example 3: Calculate CRC

In the following example the status register of the drive shall be read via the function “Read Input Register”. How to calculate the CRC checksum is shown for this telegram.

The module address of the drive is set to 1, see chapter 2.2.2 “Slave Address”, page 6.

The control sends:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	03_h	Read Input Register
Register address	1 byte high	07_h	$07D0_h$ = 2000_d = status register
	1 byte low	$D0_h$	
Register count	1 byte high	00_h	1 register
	1 byte low	01_h	
CRC	1 byte low	84_h	CRC = 0x8784
	1 byte high	87_h	

You can calculate the CRC value for 1 byte using the following C function. The transfer parameter “input” transfers the data byte. The parameter “last” transfers the value of the last calculation or the start value. In the Modbus protocol, CRC calculation starts with the value 0xFFFF.

```
uint16_t ModbusCRC(uint8_t input, uint16_t last)
{
    uint16_t crc     = last ^ (uint16_t)input;      // crc = start value XOR input
    uint16_t polynom = 0xA001;
    for(int i = 0; i < 8; i++)                      // Loop is run 8 times
    {
        if(crc & 0x0001)                            // When bit 0 is set, then...
        {
            crc = (crc >> 1) ^ polynom;             // crc = (crc shift left 1) XOR polynom
        }
        else                                        // When bit 0 is not set, then...
        {
            crc = (crc >> 1);                       // crc = (crc shift left 1)
        };
     }
     return crc;                                    // Result after 8 runs
}
```

To calculate the CRC value for the complete telegram, you can use the function as follows:

```
    uint16_t crc;
    crc = ModbusCRC(0x01, 0xFFFF);
    crc = ModbusCRC(0x03, crc);
    crc = ModbusCRC(0x07, crc);
    crc = ModbusCRC(0xD0, crc);
    crc = ModbusCRC(0x00, crc);
    crc = ModbusCRC(0x01, crc);
    printf("CRC is 0x%04x\r\n",crc);
```

The program output is: CRC is 0x8784.

The drive responds:

Name	Size	Value	Description
Slave address	1 byte	01_h	
Function code	1 byte	03_h	Read Input Register
Byte count	1 byte	02_h	2 byte data
Register values	1 byte high	70_h	Status register value: 0x7038
	1 byte low	38_h	
CRC	1 byte low	$9C_h$	CRC = 0x569C
	1 byte high	56_h	

For the response, you can calculate the CRC checksum as follows:

```
uint16_t crc;
crc = ModbusCRC(0x01, 0xFFFF);
crc = ModbusCRC(0x03, crc);
crc = ModbusCRC(0x02, crc);
crc = ModbusCRC(0x70, crc);
crc = ModbusCRC(0x38, crc);
printf("CRC is 0x%04x\r\n",crc);
```

The program output is: CRC is 0x569C.