MURR ELEKTRONIK

*stay connected*



# CUBE20S Expansion manual

## Communication module RS232

**This document is valid for the following products:**

| Name | Art. no. |
|---|---|
| CUBE20S system<br>Communication module RS232<br>Communication modules incl. base | 57140 |

**Status**
Manual no.: 57140_hdb_en_10
Language: EN
Version: 1.0
Date: 22.4.14

**Contact**
Murrelektronik GmbH
Falkenstraße 3
D-71570 Oppenweiler
Phone +49 (0) 7191 47-0
Fax +49 (0) 7191 47-491000
info@murrelektronik.de

# Table of contents

# 1 Introduction

## 1.1 Service and support

**Sales**

Our sales staff in the company, field service and technicians will support you at all times.

**CONNECTIVITY system consultants**

Our system consultants are your competent contact persons when you want to develop CONNECTIVITY solutions. Together with you, they find the optimum solutions for your electrical installations.

Our CONNECTIVITY consultants find ways that help you to permanently improve the competitiveness of your machines and plants.

**Customer Service Center (CSC)**

Our staff of the Customer Service Center will help you with all questions concerning installation and set-up. They support you, for example, if you have problems when combining hardware and software products of different manufacturers.

There are numerous support tools and possibilities for measurements - both for fieldbus systems and electromagnetic interference.

Please do not hesitate to call us on +49 (0) 7191 47-2050 or send us an e-mail to: csc@murrelektronik.de.

**Service addresses**

Please see our website for your contact person:

www.murrelektronik.com

## 1.2 Introduction / about this document

**Function of this document**

This document describes the use of the module
Communication module RS232
from the Cube20S system of Murrelektronik GmbH. It includes a description of the design, engineering and application.

## 1.3 Applicable documents

**Applicable documents**

| Document | Location |
|---|---|
| Operating manual | Online shop of Murrelektronik GmbH |

## 1.4 Symbols

This document includes information and notes that have to be observed for your own safety and to avoid personal and material damage. They are characterized as follows:

**DANGER!**

**Immediate danger**

➔ Failure to observe this warning involves an imminent risk of death or serious injuries!

**WARNING!**

**Possible danger**

➔ Failure to observe this warning may cause death or serious injuries.

**CAUTION!**

**Low-risk danger**

➔ Failure to observe this warning causes minor to moderate injuries.

***NOTICE***

**Risk of material damage**

➔ Failure to observe this warning causes material damage.

**NOTE**
Other technical information and notes of Murrelektronik GmbH.

**RECOMMENDATION**
Notes with this symbol are recommendations of Murrelektronik GmbH.

**Products and Accessories**
This symbol refers to accessories or product recommendations.

**Instruction for use**

➔ An arrow marks instructions for use.

➔ Read and observe the instructions for use.

1 | If they are numbered, it is absolutely necessary to follow them in the correct order.

2 | Read and observe the instructions for use.

## 1.5 Trademarks

The trademarks of the following companies are used in this documentation:

| | |
|---|---|
| **Adobe Systems Corp.** | Adobe Acrobat Reader |
| **Microsoft Corp.** | Microsoft Windows 7, Windows Vista, Windows 2000, Windows XE/XP and Microsoft Internet Explorer |
| **PROFIBUS International (P.I.)** | PROFIBUS, PROFIBUS DP |
| **PROFIBUS / PROFINET International (P.I.)** | PROFINET, PROFINET IO |
| **ODVA**<br>Open DeviceNet Vendor Association | EtherNet/IP |
| **Beckhoff Automation GmbH** | EtherCAT |
| **CAN in AUTOMATION** - International Users and Manufacturers Group e.V. | CANopen |
| **Gould Inc. Corporation** | Modbus |
| **Siemens AG** | S5-200, S5-300, S5-400<br>S7-200, S7-300, S7-400 |

# 2 For your own safety

## 2.1 Target group

**Users**

This manual is intended for users who have knowledge of automation systems.

**Documentation**

Please give this manual to all employees involved in the following tasks:

- Planning
- Installation
- Set-up
- Operation

## 2.2 Intended purpose

**Designated use**

The Cube20S system has been designed and manufactured for:

- communication and process control
- general control and automation tasks
- industrial use
- operation under the ambient conditions specified under technical data
- installation in a switch cabinet

**Foreseeable misuse**

The device is not approved for being used:

- in potentially explosive atmospheres (EX Zone)
- outside of switch cabinets.

## 2.3 General safety instructions

**Note:**

- the relevant safety and accident prevention regulations;
- the EC Directives or other national regulations;
- generally recognized safety rules;
- the section 2.5 "EMC installation guidelines".

***NOTICE***

**Defective device!**

Improper use of the hardware and software can cause damage to the device.

➔ Only qualified personnel of Murrelektronik GmbH may manipulate the device.

➔ Only use the device to the extent described in the manual.

***Avoid accidents caused by electrical voltage!***

➔ Comply with the 5 safety rules of electrical engineering!

➔ Disconnect the device from the mains.

➔ Then carry out installation or repair work.

***Avoid personal and material damage due to malfunctions!***

➔ Provide external circuit breakers.

➔ The device may only be operated within the specified tolerances.

***Avoid undefined states!***

- ➔ Select and install connection lines so that capacitive and inductive interferences do not have adverse effects on the system.
- ➔ Protect the device against improper and unintended use.

## 2.4 Notes on electrostatically sensitive equipment

***NOTICE***

**Overvoltage due to electrostatic discharge!**

The assemblies might get damaged.

- ➔ Ensure sufficient grounding of persons and working material!

**Handling**

Murrelektronik assemblies include highly integrated MOS components. These components are extremely sensitive to overvoltages occurring, for example, due to electrostatic discharge. Assemblies at risk are marked by the adjacent symbol.

The symbol is fixed to assemblies, sub-racks or packaging and indicates electrostatically sensitive equipment. These assemblies may become irreparably damaged by voltage and energy levels which are far below the perception levels of human beings.

If a person who is not electrostatically discharged handles electrostatically sensitive equipment, voltages may be produced. They may damage components, impair the functioning of the assemblies or render the assembly inoperative. Frequently, assemblies damaged like this cannot directly be recognized as faulty. The fault may show only after longer operation.

Components damaged by electrostatic discharge may produce temporary faults in case of temperature changes, vibrations or load changes.

Only with a consistent use of protective devices and a responsible compliance of the instructions for use can you avoid malfunctions or failures of the electrostatically sensitive equipment.

**Shipping**

- ➔ For shipping electrostatically sensitive equipment, use **always** the original packaging.

**Measurements**

Observe the following notes for measurements on electrostatically sensitive equipment:

- ➔ Discharge potential-free measuring instrument briefly.
- ➔ Ground the measuring instruments used.

**Modifications**

Observe the following in case of modifications on electrostatically sensitive equipment:

- ➔ Use a grounded soldering iron.

## 2.5 EMC installation guidelines

**Industrial use**

The CUBE20S system is an electronic device manufactured according to the current state-of-the-art standards. Both the robust mechanical construction and the design of the electronic components make it ideal for industrial use.

To guarantee a trouble-free operation, observe the following rules when installing the device in systems. Otherwise, the high interference immunity and resistance to damage of the device may become partially ineffective.

The interference immunity of the entire system considerably depends on the correct installation, location and wiring.

1 | For safe operation, check the installation regulations stipulated by the manufacturer of the controller.

2 | Bring them in line with the recommendations for an EMC-compatible design.

3 | Then install CUBE20S system.

## 2.6 Notes on spare parts and accessories

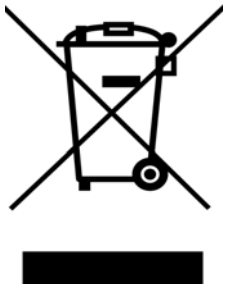
### Spare parts

- Only use the original spare parts or spare parts by other manufacturers expressly authorized by Murrelektronik GmbH.
- Check the function of the device after having replaced a component.

### Accessories

- The use of accessories may alter the device function. Use only accessories authorized by Murrelektronik GmbH.
- Observe the enclosed instructions of the accessories when installing them.

## 2.7 Environmentally friendly disposal



**Disposal**

Do not throw electrical devices, batteries or accumulators in the domestic waste!

If you want to dispose of the product, it may be returned free of charge to Murrelektronik GmbH. This is also valid for original packaging and batteries or accumulators.

***Return***

➔ Label the product and the packaging with ***"For disposal"***.

➔ Pack the product.

➔ Send the package to:

Murrelektronik GmbH

Falkenstraße 3

D-71570 Oppenweiler

We ensure that it is disposed of according to the German legislation. Transport to the place of destination is at the expense of the last owner.

## 2.8 EC Declaration of Conformity

CE

Murrelektronik GmbH herewith declares that the products and systems comply with the basic requirements and other relevant regulations of the following Directives:

- 2004/108/EC Electromagnetic compatibility
- 2011/65/EU RoHS

## 2.9 Warranty and liability

**Warranty and liability claims**

Warranty and liability claims shall be lost if

- the product is not used according to its designated use,
- damage is caused because the manual and the operating instructions have not been observed,
- the staff was/is not qualified.

# 3 System description

## 3.1 System

**Overview**

The Cube20S system is a modular automation system mounted on a 35 mm DIN rail. Using 2, 4 and 8-channel expansion modules, you may adapt this system perfectly to your automation tasks.

You do not need much wiring because the 24 V DC power supply is integrated in the backplane bus. Defective electronic modules can be replaced without having to replace the wiring.

Using power modules with different colors, you may define further voltage ranges for the 24 V DC power supply within the system or add 2 A to the electronic supply.

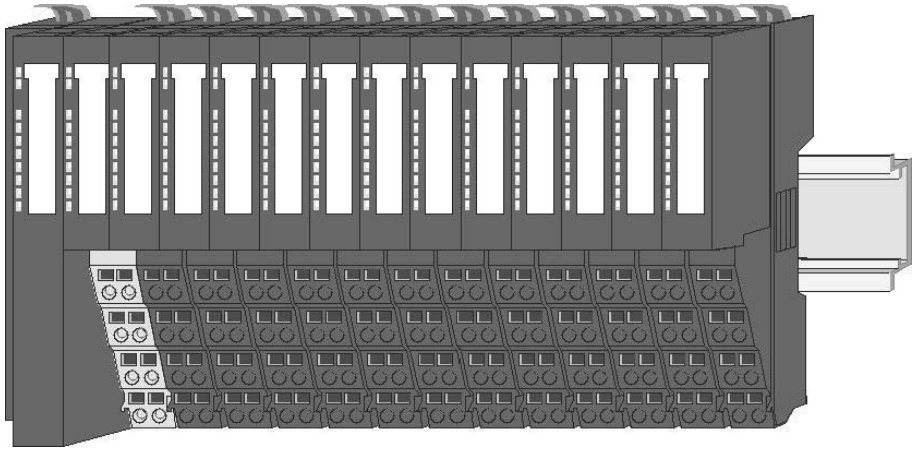


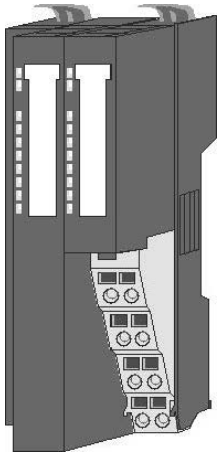
*Fig. 3-1: Cube20S system*

## Components

The Cube20S system consists of the following components:

- Bus node
- Expansion modules
- Power modules
- Accessories

## Bus node



Bus interface and power module of the bus node are incorporated in one housing. The bus interface is used to connect to a parent bus system.

Both bus interface and the electronics of the connected expansion modules are supplied with power over the power module.

There is another connection on the power module for the 24 V DC power supply of the connected expansion modules.

By installing up to 64 expansion modules on the bus node, they will be electrically connected, i.e.

- they are incorporated in the backplane bus,
- the electronic modules are supplied with power,
- each expansion module is connected to the 24 V DC power supply.

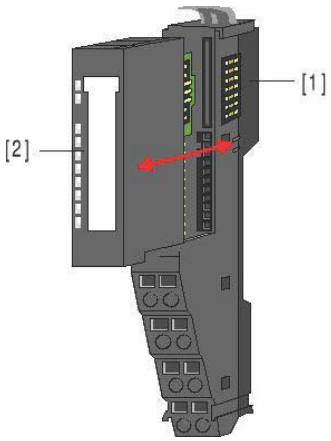
### *Bus cover*

Each bus node has a cover to protect the contacts.

➔ Remove the cover on the bus node before installing CUBE20S modules.

➔ To protect the contacts, mount the bus cover on the outmost module.

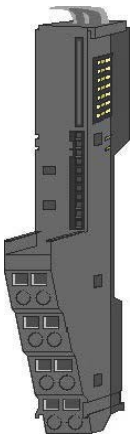
### *Expansion modules*




Each expansion module consists of a terminal and an electronic module.

1 Terminal module
2 Electronic module

### *Terminal module*

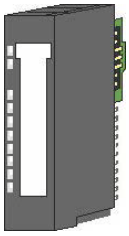


The terminal module consists of the following functional elements:

- a sliding mechanism to fasten the electronic module,
- the backplane bus with power supply for the electronics,
- the connection to the 24 V DC power supply,
- the staircase-shaped terminal block for wiring,
- a safe locking system for fastening on a mounting rail.

This locking mechanism allows you to mount your Cube20S system outside the switch cabinet and fix the complete system later in the switch cabinet.

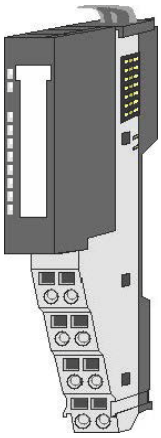
### *Electronic module*



The functionality of an expansion module is defined over the electronic module.

- If the electronic module is defective, it can be replaced while wiring is kept.
- On its front, there are LEDs indicating the status.
- For an easier wiring, there are wiring diagrams on the front and side of each electronic module.

### Power modules



Power modules provide the Cube20S system with power. The power modules are either integrated in the bus node or may be plugged between the expansion module.

Depending on the type of power module, groups of potential can be defined for the 24 V DC power supply, or the electronics supply may be extended by 2 A.

For a better recognition, the power modules have a different color than the expansion modules.

## 3.2 Dimensions

### Dimensions of the bus node

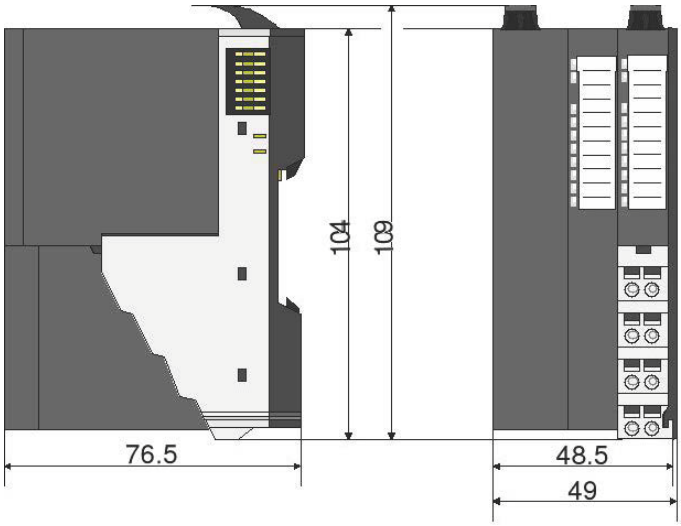



*Fig. 3-2: Dimensions of the bus node*

### Dimensions of the expansion module

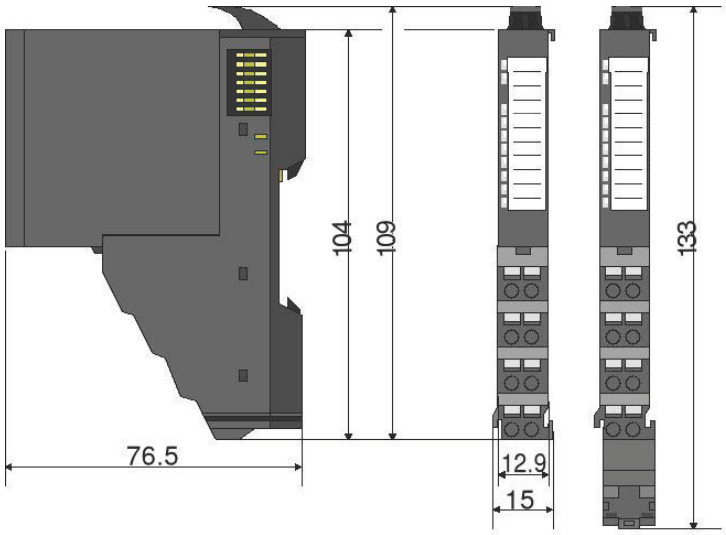



*Fig. 3-3: Dimensions of the expansion module*

### Dimensions of the electronic module

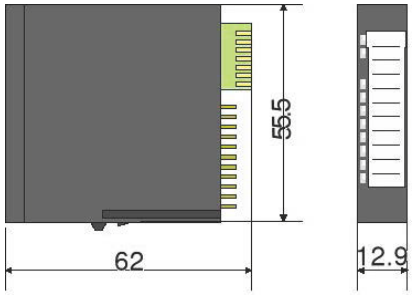



*Fig. 3-4: Dimensions of the electronic module*

## 3.3 Mounting

**NOTE**
You can mount the modules individually or as a whole block on the DIN rail. For block installation, please observe the following: **All** locking levers must be open.

### 3.3.1 General notes

The individual modules are mounted directly on a DIN rail. Electronics and power supply are connected over the backplane bus.

Conditions:

- Max. number of plug-in modules: 64
- Max. total current of the electronics supply: 3 A

A **power module sensor/actuator/bus art. no. 57131** extends the current for the electronics supply by 2 A. For details, refer to section 3.5 "Wiring".

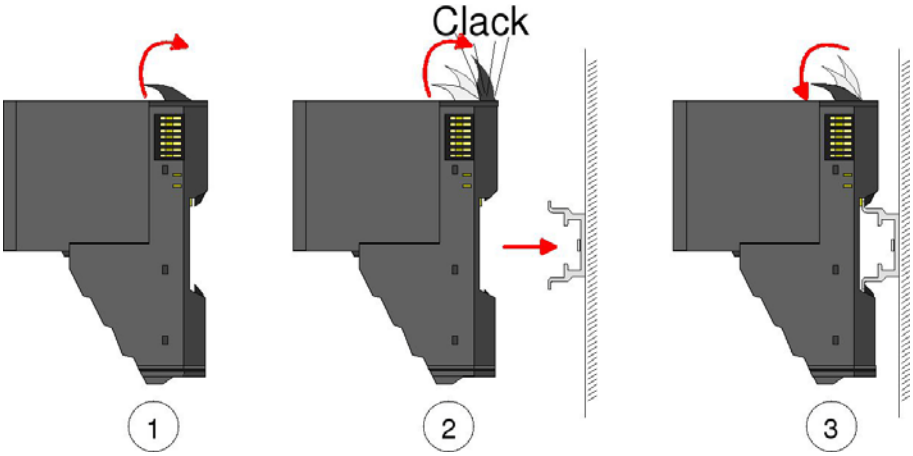



*Fig. 3-5: Installing the module*

### 3.3.2 Functional principle of the locking

***Inserting and locking the module***

🛠 The terminal module has a locking lever at its top.

1 | For installation and disassembly, please press this lever upwards until it engages audibly.

2 | Plug the module to be mounted in the previously plugged-in module.

3 | Slide the module with the help of the guide strips at top and bottom onto the DIN rail.

4 | Flap the locking lever downwards.

*The module is fastened to the DIN rail.*

### 3.3.3 Replacing an electronic module

***Disassembly***

✓ The electronic module has a locking lever at the bottom.

1 | Press the locking lever upwards for disassembly.

2 | To remove the electronic module, pull it out towards the front.

*The electronic module has been removed.*

***Installation***

✓ The electronic module has a locking lever at the bottom.

➔ Slide the electronic module with the help of the guide strip into the terminal module.

*The electronic module engages audibly at the bottom.*

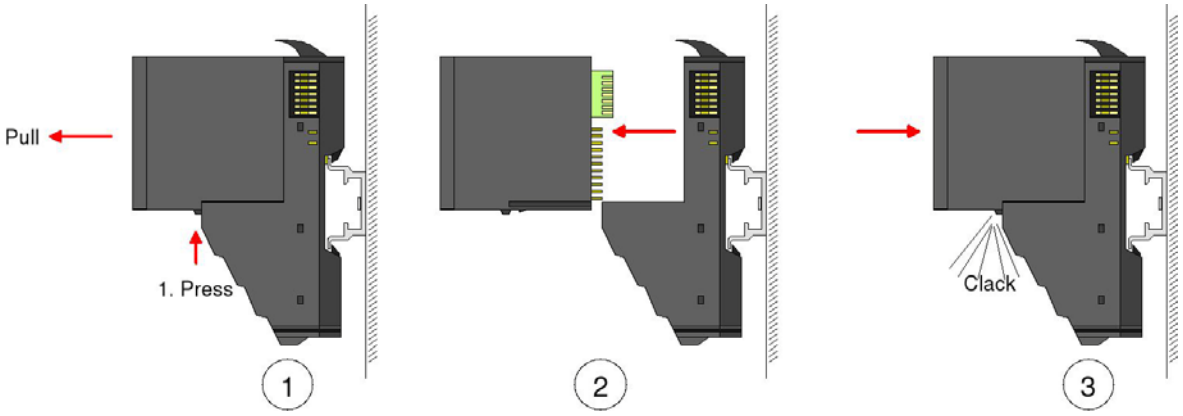



*Fig. 3-6: Disassembling and installing the electronic module*

## 3.3.4 Installing the DIN rail

➔ Install the DIN rail with the necessary distances (see Fig. 3-7: "Installation distances").

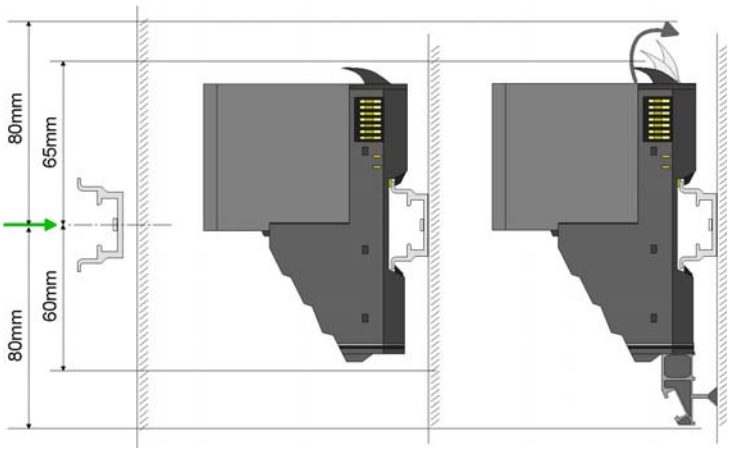



*Fig. 3-7: Installation distances*

## 3.3.5 Installing the bus node

✓ To mount the system, start on the left with the bus node.

1 | Flap the two locking levers of the bus node upwards.

2 | Plug the bus node in the DIN rail.

3 | Flap the two locking levers of the bus node downwards.

4 | To remove the right bus cover, pull it out towards the front.

5 | Store the bus cover to use it as termination of the system.

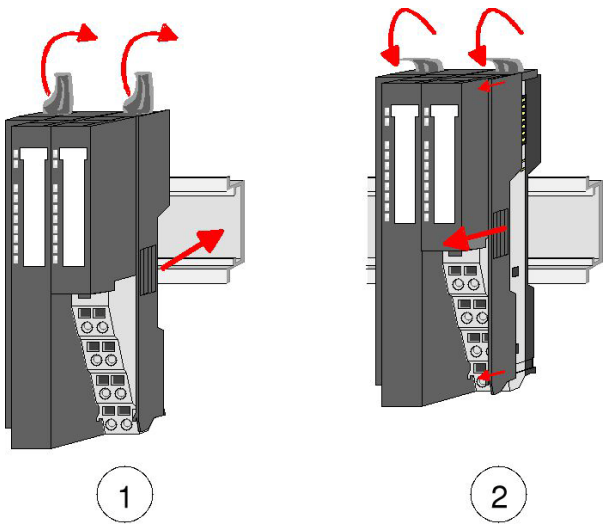



*Fig. 3-8: Installing the bus node*

### 3.3.6 Installing the expansion modules

1 | Flap the locking lever of the expansion module upwards.
2 | Plug the expansion module in the DIN rail.
3 | Push the expansion module towards the bus node or the last expansion module.
4 | Flap the locking lever of the expansion module downwards.
5 | Mount all expansion modules as described.

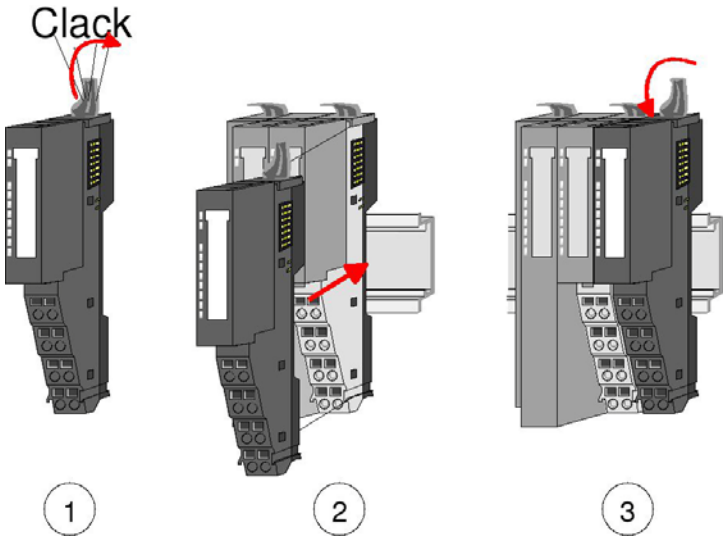



*Fig. 3-9: Installing the expansion module*

### 3.3.7 Installing the bus cover

✓ Prerequisite: The system has been completely mounted.
➔ Plug the bus cover in the outmost module as a protection of the bus contacts.

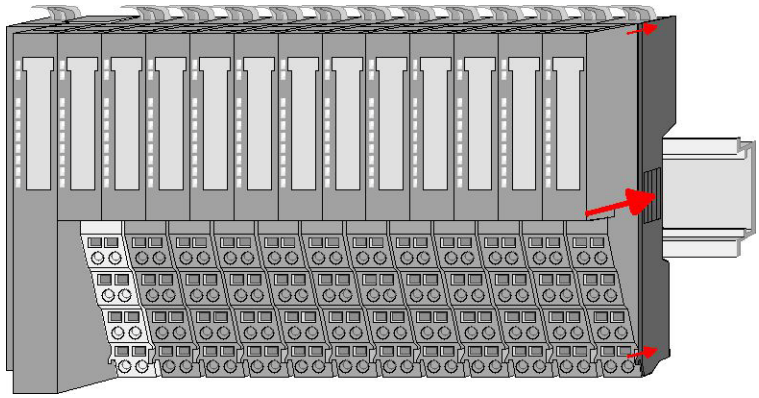


*Fig. 3-10: Installing the bus cover*

# 3.4 Disassembling and replacing modules

## 3.4.1 Procedure

***During disassembly or when replacing a module or module group, please observe the following:***

1 | Remove the electronic module to the right of the module or module group.
2 | Dismount/replace the module or module group.
3 | Plug in the electronic module.

## 3.4.2 Replacing an electronic module

***Disassembly***

✓ The electronic module has a locking lever at the bottom.
1 | Press the locking lever upwards for disassembly.
2 | To remove the electronic module, pull it out towards the front.

*The electronic module has been removed.*

***Installation***

✓ The electronic module has a locking lever at the bottom.
➔ Slide the electronic module with the help of the guide strip into the terminal module.

*The electronic module engages audibly at the bottom.*

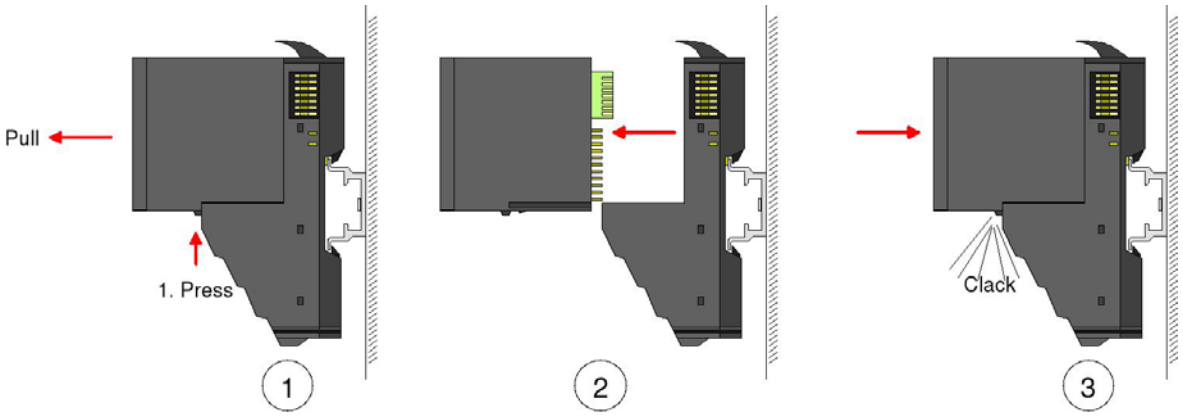



*Fig. 3-11: Disassembling and installing the electronic module*

### 3.4.3 Replacing a module

**Dismounting**

1 | Remove the wiring from the module, if any. For details refer to section ***Wiring***.
2 | Unlock the electronic module to its right at the bottom.
3 | To remove the electronic module, pull it out towards the front.
4 | Flap the locking lever of the module to be replaced upwards.
5 | To remove the module, pull it out towards the front.

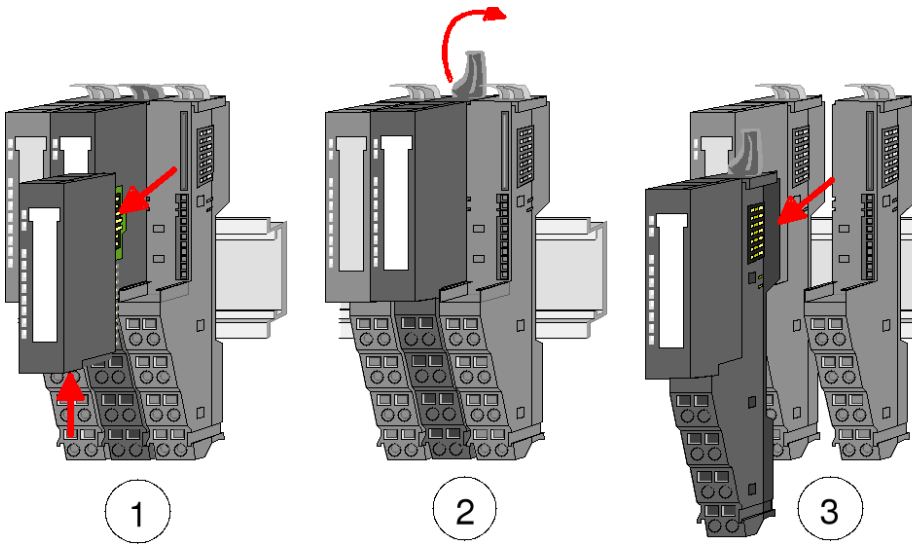



*Fig. 3-12: Disassembling a module*

**Installing the new module**

1 | Flap the locking lever of the module upwards.
2 | Plug the module in the gap between the modules.
3 | Slide the module with the help of the guide strips at both sides onto the DIN rail.
4 | Flap the locking lever of the module downwards.
5 | Plug in the electronic module.

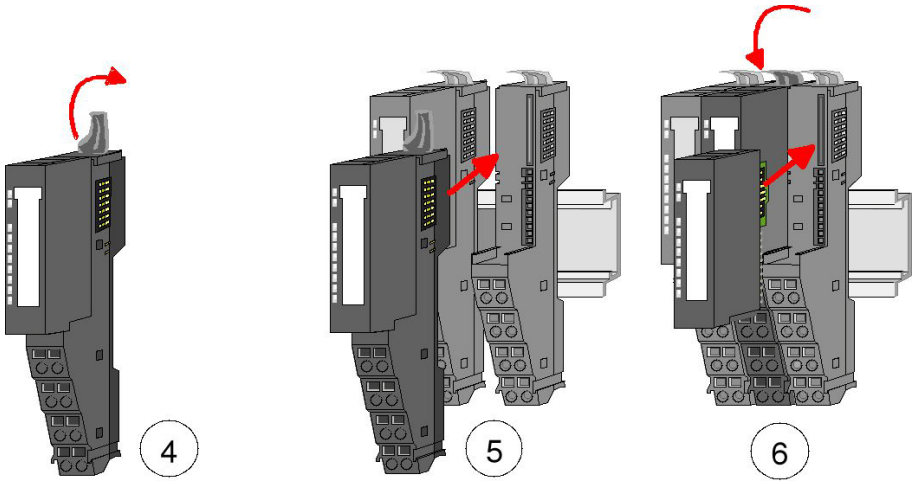



*Fig. 3-13: Installing the new module*

## 3.4.4 Replacing a bus node

### Disassembly

**⚠ CAUTION!**

**Power module and bus interface belong together!**

If separated, the modules get destroyed.

➔ Do not separate power module and bus interface!

1 | Remove the wiring from the bus node, if any. For details, please see section ***Wiring***.

2 | Unlock the electronic module to its right at the bottom.

3 | To remove the electronic module, pull it out towards the front.

4 | Flap the locking lever of the bus node upwards.

5 | To remove the bus node, pull it out towards the front.

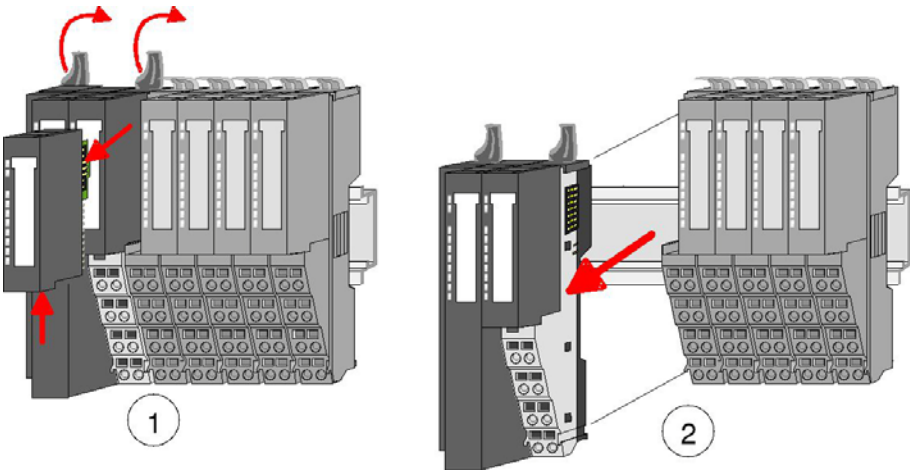



*Fig. 3-14: Disassembling the bus node*

**Installing the new bus node**

1 | Flap the locking levers of the bus node upwards.
2 | Plug the bus node in the left module.
3 | Slide the bus node with the help of the guide strips onto the DIN rail.
4 | Flap the locking levers downwards.
5 | Plug in the electronic module.

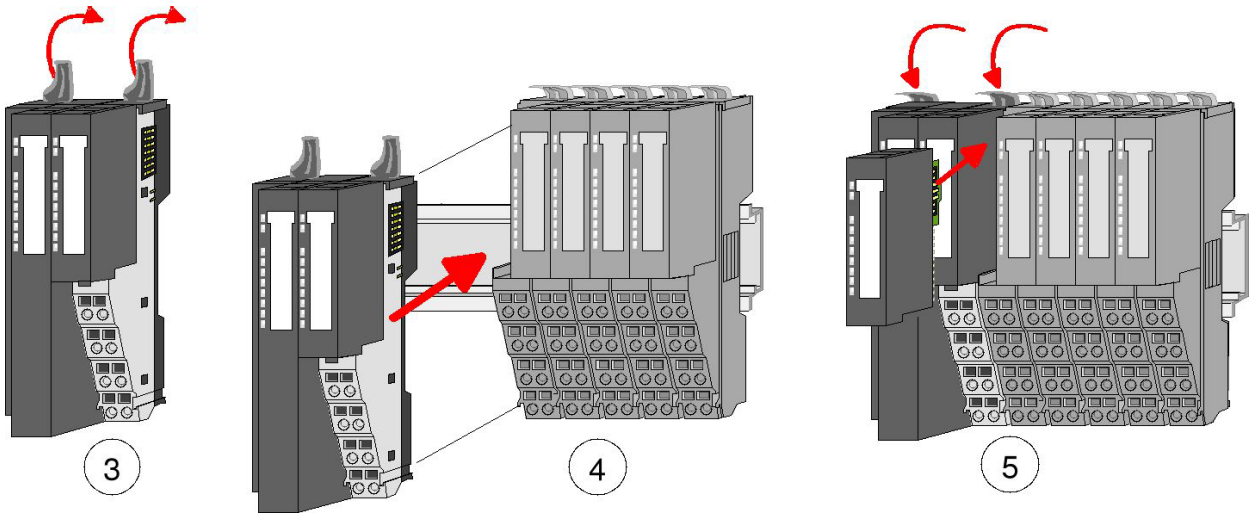



*Fig. 3-15: Installing the new bus node*

### 3.4.5 Replacing a module group

**Disassembly**

1 | Remove the wiring from the module group, if any. For details, please see section ***Wiring***.
2 | Unlock the electronic module to its right at the bottom.
3 | To remove the electronic module, pull it out towards the front.
4 | Flap the locking levers of the module group upwards.
5 | To remove the module group, pull it out towards the front.

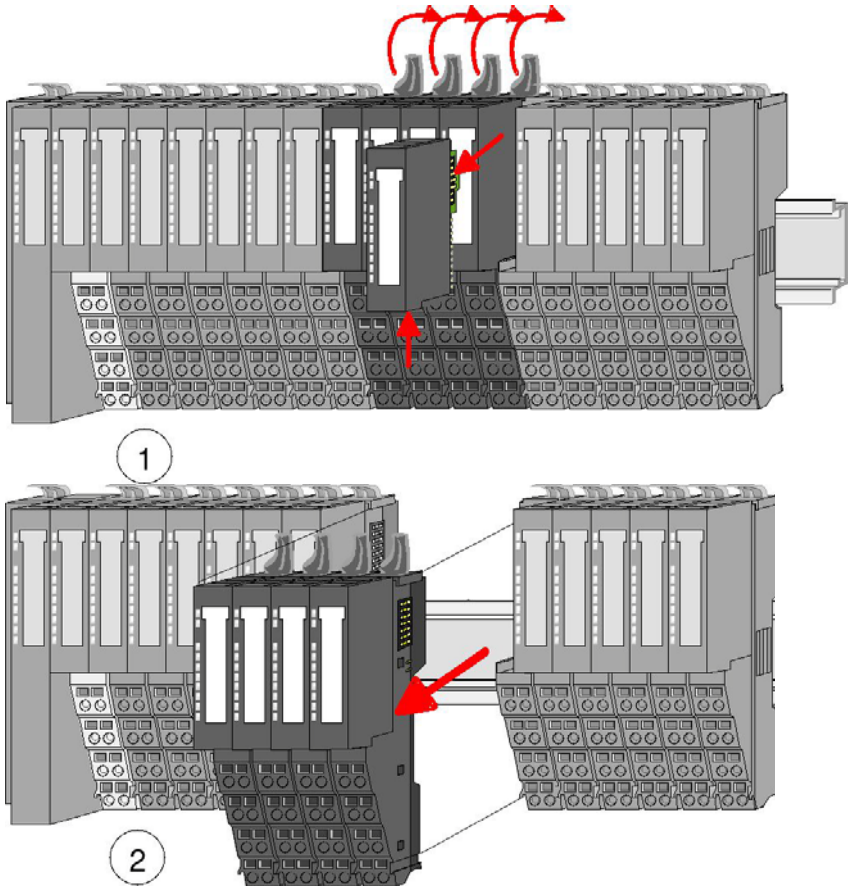



*Fig. 3-16: Disassembling the module group*

**Installing the new module group**

1 | Flap the locking levers of the module group upwards.
2 | Plug the module group in the gap between the modules.
3 | Slide the module group with the help of the guide strips at both sides onto the DIN rail.
4 | Flap the locking levers of the module group downwards.
5 | Plug in the electronic module.

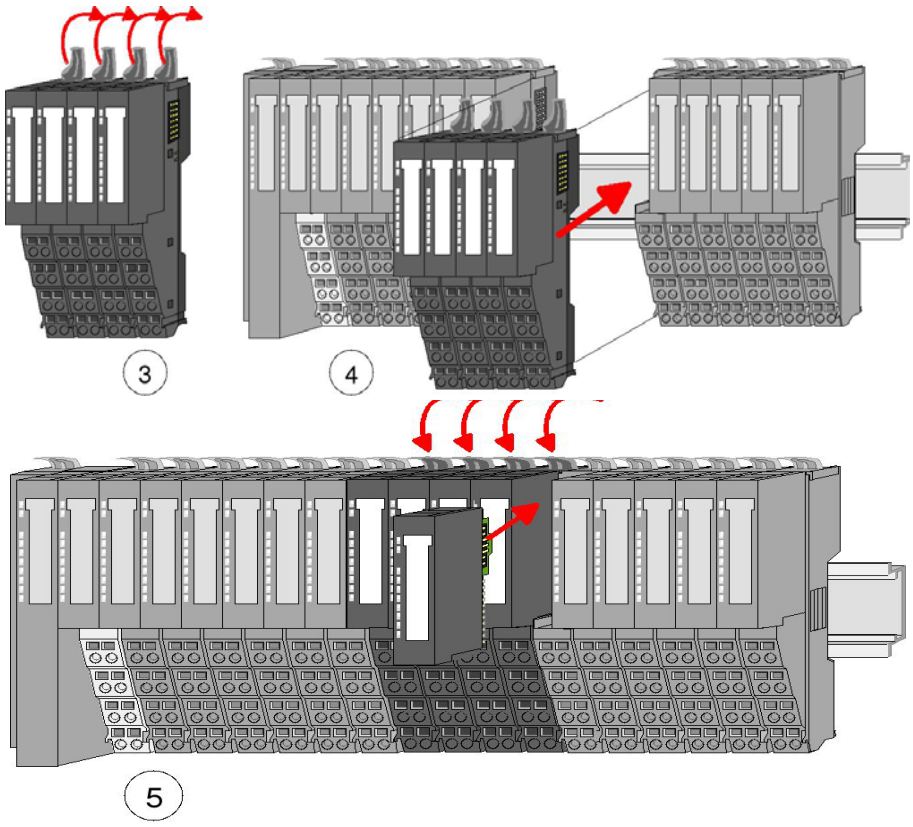



*Fig. 3-17: Mounting of the module group*

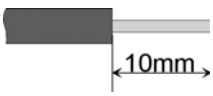
## 3.5 Wiring

### 3.5.1 Spring terminals

**Terminals**

Spring terminals are used for wiring. Spring terminals allow you to connect the signaling lines and power cables fast and easily. This type of connection is resistent to vibrations.

**Cable data**

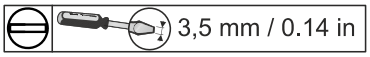



$U_{max.}$: 240 V AC / 30 V DC
$I_{max.}$: 10 A
Cross section: 0.08 ... 1.5 $mm^2$ (AWG 28 ... 16)
Stripping length: 10 mm

### 3.5.2 Procedure

***Wiring***

- Tools: suitable screwdriver




- Wire cross-section: 0.08 $mm^2$ ... 1.5 $mm^2$ (AWG 28 ... 16)

1 | Put the screwdriver slightly inclined in the rectangular opening (Fig. 3-18: 1).

2 | Press and hold the screwdriver away from the round opening. The contact spring is open (Fig. 3-18: 2).

3 | Put the stripped wire in the round opening (Fig. 3-18: 2).

4 | Remove the screwdriver (Fig. 3-18: 3).

*The wire is securely connected with the terminal by means of a spring contact.*

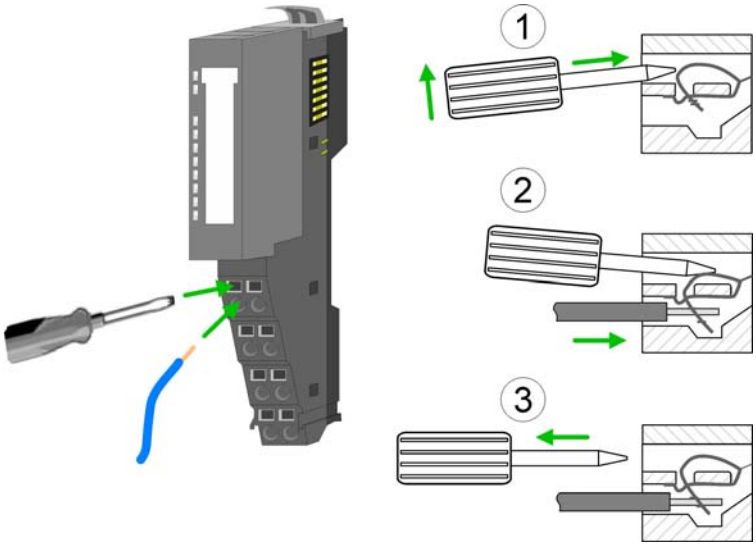



*Fig. 3-18: Spring terminals*

## 3.5.3 Standard wiring

**Standard wiring**

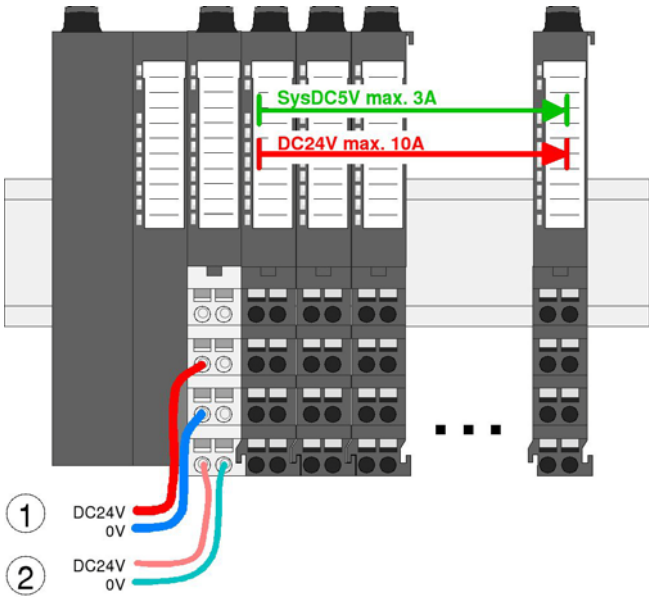



*Fig. 3-19: Standard wiring*

1 24 V DC for power supply of I/O level (max. 10 A)
2 24 V DC for electronics supply, bus node and I/O level

## 3.5.4 Fuse protection

**WARNING!**

**The power supply is not protected internally.**

It can get destroyed by too high currents.

➔ Protect the power supply externally using a fuse or line circuit breaker!

**NOTE**

The electronics supply is internally protected against too high voltages by means of a fuse. The fuse is located inside the power module. After the fuse has tripped, the electronic module has to be replaced!

**External fuse**

| | External Fuse | Circuit breaker (optional) | Comment |
|---|---|---|---|
| Power supply | 10 A (fast) | 10 A characteristic Z | up to max. current 10 A |
| Electronics supply, bus node and I/O level | 2 A (fast) | 2 A characteristic Z | Recommendation! |
| Electronics supply, I/O level, power module art. no. 57131 | 1 A (fast) | 1 A characteristic Z | Recommendation! |

*Tab. 3-1: Fuse protection of power supplies*

## 3.5.5 Using power modules

**Status of the electronics power supply**

After switching on Cube20S, the RUN or MF LED lights up at every module.

If the total current for the electronics supply exceeds 3 A, the LEDs are not activated. In this case, plug in the power module, art. no. 57130, between the expansion modules.

**NOTE**
To guarantee power supply, the power modules can be used in any combination.

**Power module art. no. 57130**

Use this power module if

- 10 A are not longer enough for power supply
- you want to have groups of different potentials

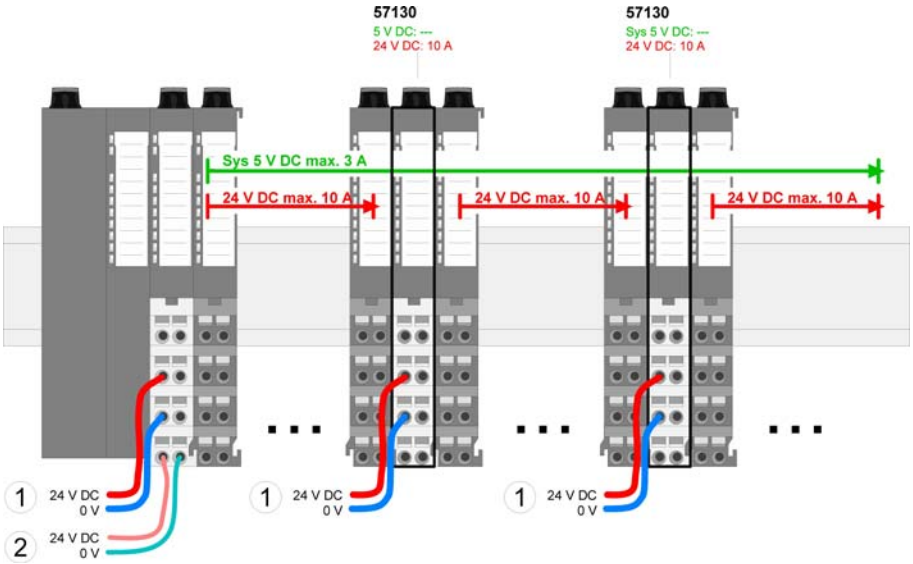



*Fig. 3-20: Power module art. no. 57130*

1 24 V DC for power supply of I/O level (max. 10 A)
2 24 V DC for electronics supply, bus node and I/O level

**Power module art. no. 57131**

Use this power module if 3 A are not enough for electronics supply on backplane bus.

In addition, you will have a new group of potential for 24 V DC power supply with max. 4 A.

Using a power module, you can plug in modules with a maximum total current of 2 A in the following backplane bus. Then you have to plug in another power module.

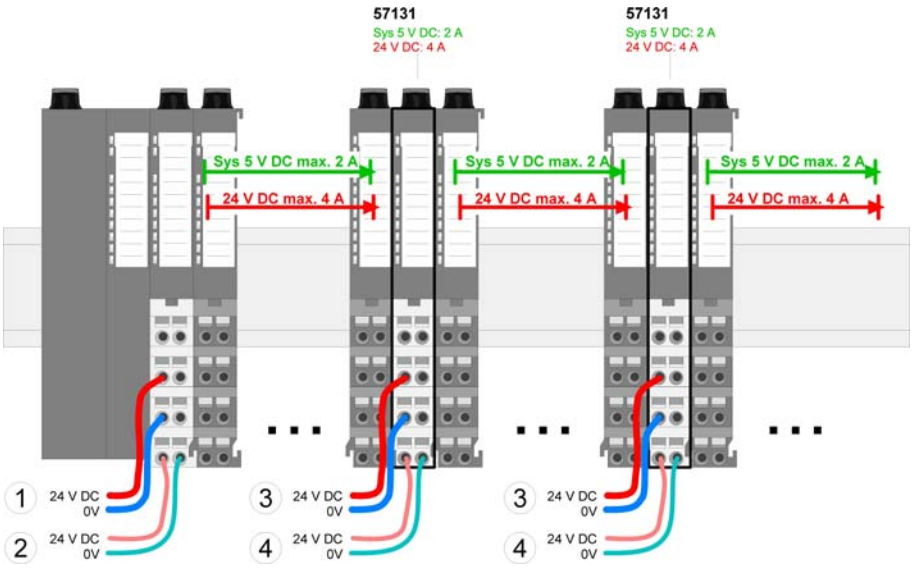



*Fig. 3-21: Power module art. no. 57131*

1 24 V DC for power supply of I/O level (max. 10 A)
2 24 V DC for electronics supply, bus node and I/O level
3 24 V DC for power supply of I/O level (max. 4 A)
4 24 V DC for electronics supply, I/O level

## 3.5.6 Fixing the shield

**Fixing the shield**

**NOTE**
Shield bus carriers are required for installing a shield (see **Accessories**).

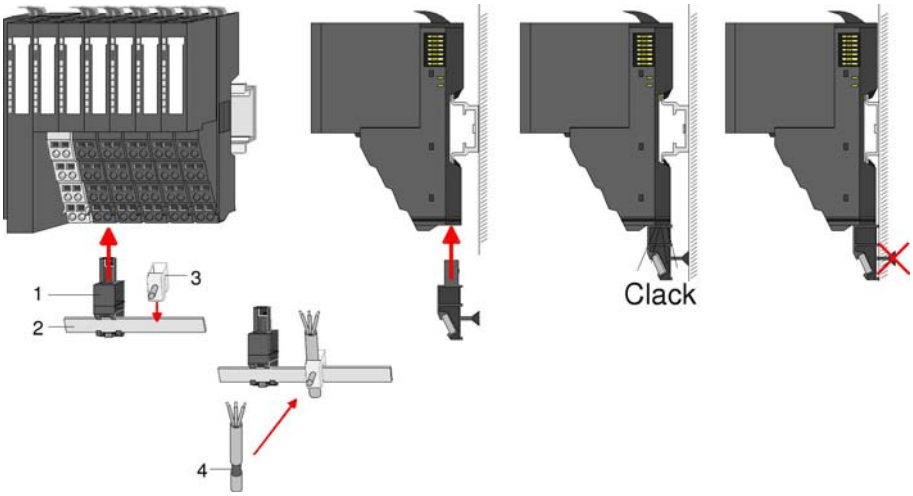



*Fig. 3-22: Fixing the shield*

1 Shield bus carrier
2 Shield bus (10 mm x 3 mm)
3 Shield terminal block
4 Shielding

***Fixing the shielding***

✓ The shield bus carrier and the shield bus have been plugged in.
➔ Fasten the lines with the stripped shield.
➔ Connect the shield terminal blocks to the shield bus.

## 3.6 Troubleshooting - LEDs

**General information**

Each module has two LEDs on the front: **RUN** and **MF**. These LEDs allow you to detect errors in your system or faulty modules.

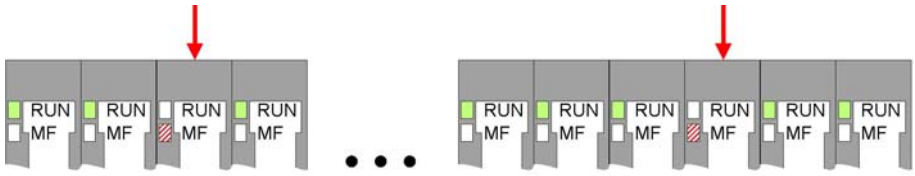
| Designation | Indication | LED status |
|---|---|---|
| RUN LED | | off |
| | | green |
| | | flashing green |
| MF LED | | off |
| | | red |
| | | flashing red |

*Tab. 3-2: Status indications of the LEDs*

**Total current of electronics supply exceeded**

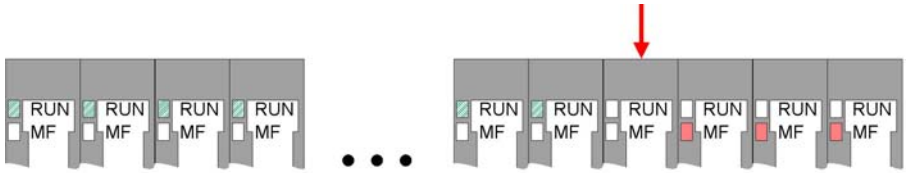

| | |
|---|---|
| Reaction of the LEDs after switching on: | The **RUN LEDs** of all modules are off.<br>The **MF LEDs** are only lighted on some modules. |
| Cause: | The total current for electronics supply exceeds the maximum current. |
| Remedy: | Plug in the power module art. no. 57131.<br>For details, please see section ***Wiring***. |

### Configuration error




| | |
|---|---|
| Reaction of the LEDs after switching on: | The **RUN LEDs** are off on one or several modules.<br>The **MF LEDs** are flashing on these modules. |
| Cause: | The module on which the **MF LED** is flashing, does not match the current configuration. |
| Remedy: | Match configuration and hardware structure. |

### Module failure




| | |
|---|---|
| Reaction of the LEDs after switching on: | The **RUN LEDs** are flashing up to the module to the left of the defective module. On the following modules, the **RUN LED** is off.<br>The **MF LEDs** are off up to the module to the left of the defective module. On the following modules, the **MF LED**is lit. |
| Cause: | The module to the right of the flashing modules is defective. |
| Remedy: | Replace the defective module. |

# 4 Art. no. 57140 Communication module RS232

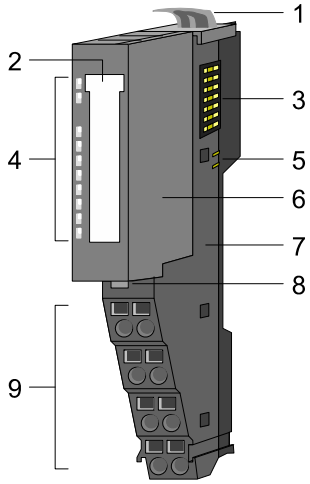
## 4.1 Features

**Features**

- Serial RS232 interface (electrically isolated from the backplane bus)
- Data transfer rate of 150 bit/s up to maximum 115.2 kbit/s
- Serial communication via RS232
- Protocols
  - ASCII
  - STX/ETX
  - 3964(R)
  - Modbus (master/slave with ASCII and RTU short & long) with a telegram length of 250 bytes
- Up to 250 telegrams (1024 bytes of receive and send buffer)
- Character delay time can be parameterized in ms steps
- Parameterization over 17 bytes of parameter data
- Modem signal management DTR-DSR-DCD

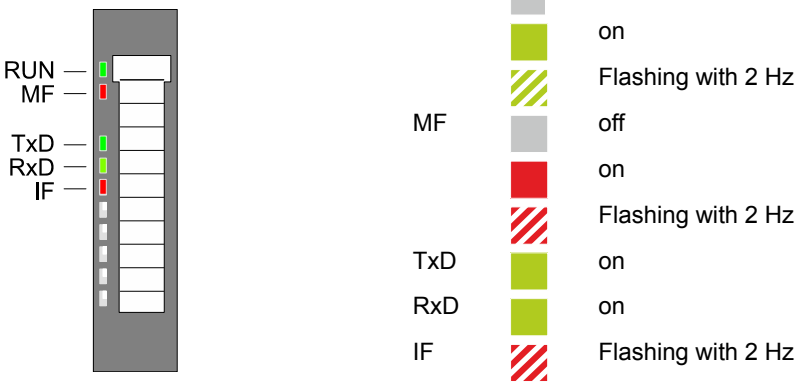
## 4.2 Design

**57140**




1 Spring latch Terminal module
2 Labelling strips
3 Backplane bus
4 LED status indication
5 24 V DC power supply
6 Electronic module
7 Terminal module
8 Locking lever of electronic module
9 Terminals

## Status indication




| RUN | MF | Description |
|---|---|---|
| | | Bus communication is OK<br>Module status is OK |
| | | Bus communication is OK<br>Module status reports error |
| | | Bus communication is not possible<br>Module status reports error |
| | | Error<br>Bus supply voltage |
| | | Configuration error<br>(see 3.6 Troubleshooting - LEDs, Seite 32) |

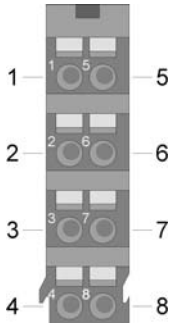
*Tab. 4-1: Status indications of RUN and MF LED*

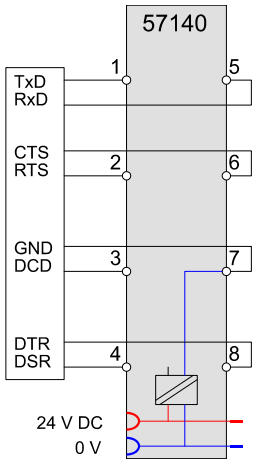
| LED | Color | Description |
|---|---|---|
| TxD | | Transmit data |
| RxD | | Receive data |
| IF | | **Modbus:** internal error<br>**Other protocols:** power interruption, overflow, parity error or character frame error |

*Tab. 4-2: Status indication of the LEDs TxD, RxD, IF*

**Terminal**

➔ Connect the wires with a cross section from 0.08 $mm^2$ to 1.5 $mm^2$.




| Pos. | Function | Type | Description |
|---|---|---|---|
| 1 | TxD | Output | Transmit data |
| 2 | RTS | Output | Request to send<br>RTS at **1**: Communication processor is ready to transmit<br>RTS at **0**: Communication processor is not transmitting |
| 3 | DCD | Input | Data carrier detect<br>Data can be received |
| 4 | DSR | Input | Data set ready<br>Modem indicates readiness for operation |
| 5 | RxD | Input | Receive data |
| 6 | CTS | Input | Clear to send<br>Communication processor may send data |
| 7 | GND_ISO | Output | Zero reference point of signal (isolated) |
| 8 | DTR | Output | Data Terminal Ready<br>Communication processor is ready for operation |

*Tab. 4-3: Terminal assignment*

**NOTE**
**RI (Ring indicator)** - The communication processor does not use the ring tone RI of the modem!

**RS232 Interface**

- Logical states as voltage levels
- Point-to-point link with serial full-duplex transmission
- Data transfer to a distance of up to 15 m
- Data transfer rate up to 115.2 kbit/s

RS232
Wiring without hardware handshake

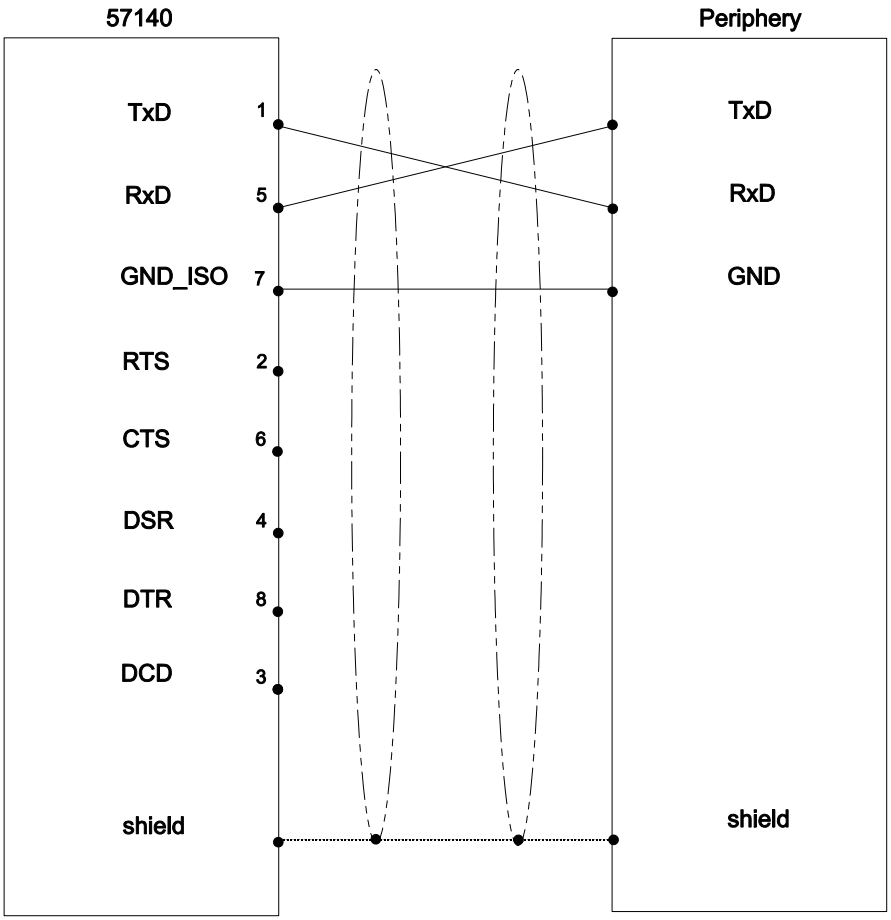



*Fig. 4-1: RS232 wiring without hardware handshake*

RS232
Wiring with
hardware
handshake

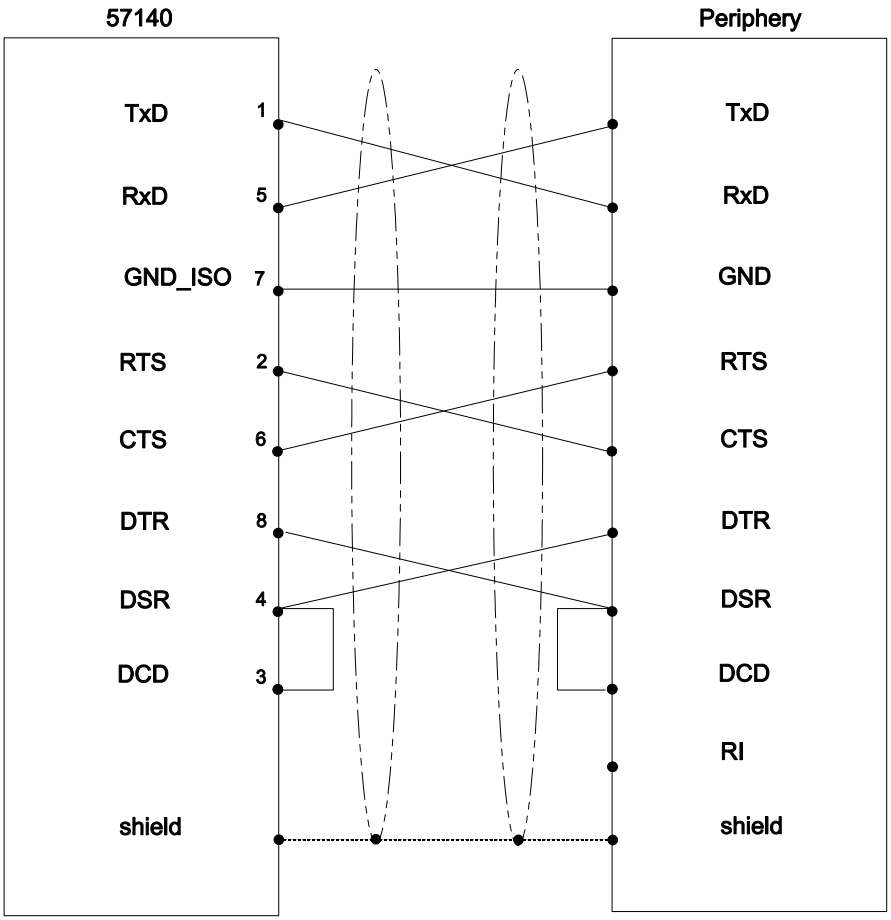



*Fig. 4-2: RS232 wiring with hardware handshake*

## 4.3 Quick start

**Overview**

The communication processor links serial processes at different target or source systems. While doing so, the communication processor is operated as an expansion module and supplied with operating voltage via the backplane bus.

**Parameters**

For parameterization, transfer the parameter data to the communication processor. They are assigned according to the selected protocol.

More detailed information on the assignment of the parameters is given in section 5 Serial communication protocols, Seite 61.

**Protocols**

The communication processor supports the following protocols:

- ASCII
- STX/ETX
- 3964(R)
- Modbus (master, slave)

**Communication**

**Transmitting**

1 | The parent system writes the data via the backplane bus to the output range.
2 | The communication processor writes the data to the send buffer.
3 | The communication processor outputs the data from the send buffer via interface.

**Receiving**

1 | The communication processor receives data via the interface.
2 | The communication processor saves the data in a circular buffer.
3 | The communication processor enters the data in the input range of the parent system via the backplane bus.

The size of the input/output range and, thus, the size of the telegram on the backplane bus depends on the parent system.

The input/output range (I/O range) and communication via the backplane bus are described in sections 4.4 and 4.5 .

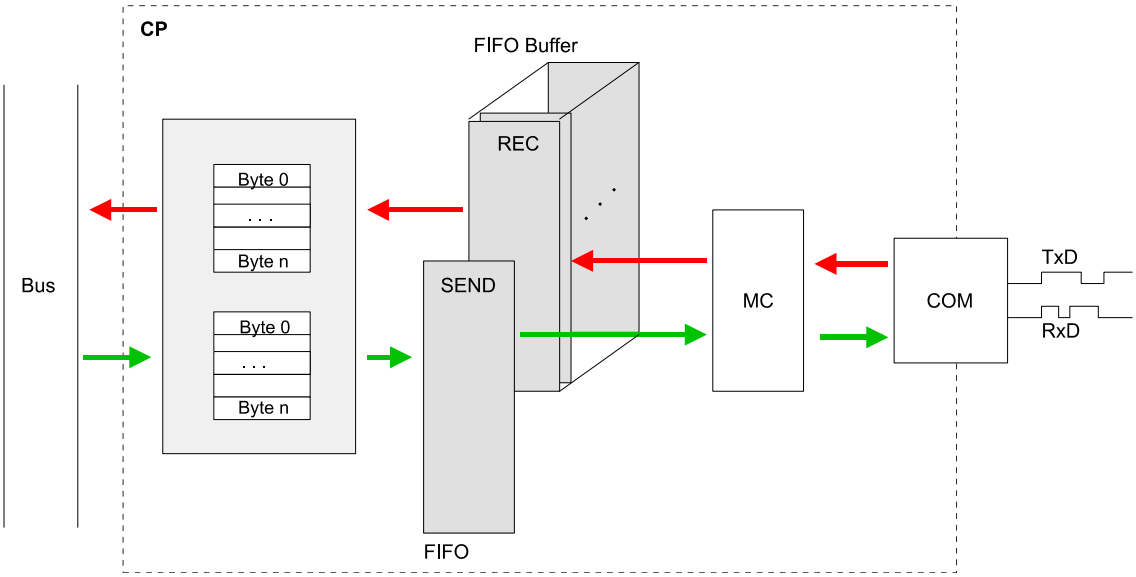



*Fig. 4-3: Communication*

## 4.4 Input/output range

**Overview**

Depending on the parent system, the communication processor assigns the following number of bytes in the address range for the input and output respectively:

| System | Address range [byte] |
|---|---|
| PROFIBUS | 8, 20, 60 (can be selected) |
| PROFINET | 20, 60 (can be selected) |
| CANopen | 8 |
| EtherCAT | 60 |
| DeviceNET | 60 |
| ModbusTCP | 60 |

*Tab. 4-4: Overview of the address range*

In CPU, PROFIBUS and PROFINET the input or output range is displayed in the corresponding address range with n = 8, 20 or 60.

IX = Index for access using CANopen

Use s = subindex to address the corresponding byte

SX = Subindex for access via EtherCAT

**Input range**

| Addr. | Name | Bytes | Function | IX = 0x5450 | SX |
|---|---|---|---|---|---|
| +0 | CP_IN_STS | 1 | Status byte | s = 1 | 0x01 |
| +1 | CP_IN_1 | 1 | Input byte 1 | s = 2 | 0x02 |
| +2 | CP_IN_2 | 1 | Input byte 2 | s = 3 | 0x03 |
| ... | ... | ... | ... | ... | ... |
| +n-1 | CP_IN_n-1 | 1 | Input byte n-1 | s = m | 0xm |

*Tab. 4-5: Input range*

CP_IN_STS — This parameter contains information on the fragmentation of the data in the receive buffer.

CP_IN_x — The contents of this data depends on the structure of the data in the receive buffer. Further information on this topic can be found on the following pages.

**Output range**

| Addr. | Name | Bytes | Function | IX = 0x5650 | SX |
|---|---|---|---|---|---|
| +0 | CP_OUT_STS | 1 | Control byte | s = 1 | 0x01 |
| +1 | CP_OUT_1 | 1 | Output byte 1 | s = 2 | 0x02 |
| +2 | CP_OUT_2 | 1 | Output byte 2 | s = 3 | 0x03 |
| ... | ... | ... | ... | ... | ... |
| +n-1 | CP_OUT_n-1 | 1 | Output byte n-1 | s = m | 0xm |

*Tab. 4-6: Output range*

CP_OUT_CTRL — Here you can control the data transfer by means of the corresponding commands.

CP_OUT_x — The contents of this data depends on the structure of the data in the send buffer. Further information on this topic can be found on the following pages.

## 4.5 Principle of the backplane bus communication

### 4.5.1 Transmitting data

**Overview**

- The parent system enters the data to be output into the output range and transfers them with the ***control byte*** to the communication processor.
- The communication processor acknowledges each telegram:
  - it copies bit 3 ... 0 from byte 0 of the output range to bit 7 ... 4 of byte 0 of the input range **or**
  - it uses this byte to send back a corresponding ***status message***.
- Depending on the length of the data to be transferred, the telegram is transmitted to the communication processor in one or several fragments.
- In case of fragmented transmission, the communication processor acknowledges each fragment.

**Principle of transmission**
without fragmentation

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | Control byte |
| 1 | Telegram info byte |
| 2 | Length high byte |
| 3 | Length low byte |
| 4 ... n-1 | Payload byte 0 ... n-5 |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | Acknowledgement / status |

with n = number of the assigned bytes in the address range (IO size)

Control byte

| Bit | Hex | Function |
|---|---|---|
| 3 ... 0 | 0x08 | Idle run - no data available |
| | 0x0A | Start transmission without fragmentation |
| | 0x0B | Perform a reset of the communication processor |
| 7 ... 4 | | Reserved for receiving |

*Tab. 4-7: Control byte*

Telegram info byte — During transmission 0x00 (fixedly).

Length — Length of the payload for the serial communication in bytes.

Payload byte — The byte contains the payload for the serial communication.

Acknowledgement / status

| Bit | Hex | Function | |
|---|---|---|---|
| 3 ... 0 | | Reserved for receiving | |
| 7 ... 4 | 0x08 | Acknowl-edgement | Idle run |
| | 0x0A | | Receiving data without fragmentation |
| | 0x0C | Status | Reset of communication processor performed |
| | 0x0D | | The specified length is not valid |
| | 0x0E | | Error in the communication of the communication processor - partner does not respond |

*Tab. 4-8: Acknowledgement / status*

**Principle of transmission** with fragmentation

- The parent system transmits the amount of payload and part of the payload in the 1st telegram.
- After that the parent system transmits the fragment telegrams.
- The communication processor acknowledges each fragment telegram:
  - it copies bit 3 ... 0 from byte 0 of the output range to bit 7 ... 4 of byte 0 of the input range **or**
  - it uses this byte to send back a corresponding ***status message***.

Procedure

1 | Write the 1st telegram
2 | Write fragments
3 | Write last fragment

Calculation of the number of fragments

Number of fragments= (length + 3) / (IO_Size -1)

Write the 1st telegram (header)

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | Control byte |
| 1 | Telegram info byte |
| 2 | Length high byte |
| 3 | Length low byte |
| 4 ... n-1 | Payload byte 0 ... n-5 |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | Acknowledgement / status |

with n = number of the assigned bytes in the address range (IO size)

Control byte

| Bit | Hex | Function |
|---|---|---|
| 3 ... 0 | 0x08 | Idle run - no data available |
| | 0x09 | Start fragmented transmission |
| | 0x0A | Transmit the last fragment |
| | 0x0B | Run a reset of the communication processor |
| 7 ... 4 | | Reserved for receiving |

*Tab. 4-9: Control byte*

Telegram info byte

During transmission 0x00 (fixedly).

Length

Length of the payload for the serial communication in bytes.

Payload byte

The byte contains the payload for the serial communication.

Acknowledgement / status

| Bit | Hex | Function | |
|---|---|---|---|
| 3 ... 0 | | Reserved for receiving. | |
| 7 ... 4 | 0x08 | Acknowl-edgement | Idle run |
| | 0x09 | | Fragmented transmission started |
| | 0x0A | | Receiving data without fragmentation |
| | 0x0C | Status | Reset of communication processor performed |
| | 0x0D | | The specified length is not valid |
| | 0x0E | | Error in communication of the communication processor - partner does not respond |

*Tab. 4-10: Acknowledgement / status*

Writing the fragments

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | Control byte |
| 1 ... n-1 | Payload |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| 0 | Acknowledgement / status |

with n = number of the assigned bytes in the address range (IO size)

Control byte

| Bit | Hex | Function |
|---|---|---|
| 3 ... 0 | 0x00 ... 0x07 | Fragment number |
| | 0x08 | Idle run - no data available |
| | 0x0B | Run a reset of the communication processor |
| 7 ... 4 | | Reserved for receiving |

*Tab. 4-11: Control byte*

Payload byte

The byte contains the payload for the serial communication.

Acknowledgement / status

| Bit | Hex | Function | |
|---|---|---|---|
| 3 ... 0 | | Reserved for receiving | |
| 7 ... 4 | 0x00 ... 0x07 | Acknowl-edgement | Fragment number |
| | 0x08 | | Idle run |
| | 0x0C | Status | Reset of communication processor performed |
| | 0x0D | | The specified length is not valid |
| | 0x0E | | Error in the communication of the communication processor - partner does not respond |

*Tab. 4-12: Acknowledgement / status*

Writing the last fragment

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | Control byte |
| 1 ... n-1 | Payload |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| 0 | Acknowledgement / status |

with n = number of the assigned bytes in the address range (IO size)

Control byte

| Bit | Hex | Function |
|---|---|---|
| 3 ... 0 | 0x08 | Idle run - no data available |
| | 0x0A | Transmit the last fragment |
| | 0x0B | Run a reset of the communication processor |
| 7 ... 4 | | Reserved for receiving |

*Tab. 4-13: Control byte*

Payload byte

The byte contains the payload for the serial communication.

Acknowledgement / status

| Bit | Hex | Function | |
|---|---|---|---|
| 3 ... 0 | | Reserved for receiving. | |
| 7 ... 4 | 0x08 | Acknowl-edgement | Idle run |
| | 0x0A | | Last fragment received |
| | 0x0C | Status | Reset of communication processor performed |
| | 0x0D | | The specified length is not valid |
| | 0x0E | | Error in the communication of the communication processor - partner does not respond |

*Tab. 4-14:* Acknowledgement / status

## 4.5.2 Receiving data

**Overview**

- The communication processor enters the received data automatically in the input range of the parent system.
- Depending on the length of the received data, the communication processor transmits the telegram to the parent system:
  - in one fragment **or**
  - in several fragments.
- The fragmented transmission starts after bit 3 ... 0 from byte 0 of the input range to bit 7 ... 4 of byte 0 of the output range.
- The communication processor saves transmission errors in ***RetVal***.

**Principle of the transmission**
without fragmentation

| Parent system | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | Acknowledgement |

◀
▶

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| 0 | Info byte |
| 1 | Telegram info byte |
| 2 | Length high byte |
| 3 | Length low byte |
| [4] | Offset high byte |
| [5] | Offset low byte |
| 6 | RetVal high byte |
| 7 | RetVal low byte |
| 8...n-1 | Payload |
| | |
| 0 | |

with n = number of the assigned bytes in the address range (IO size)

Info byte

| Bit | Hex | Function |
|---|---|---|
| 3 ... 0 | 0x08 | Idle run - no data available |
| | 0x09 | Data are transmitted in fragments |
| | 0x0A | Data are transmitted without fragmentation |
| 7 ... 4 | | Reserved for transmitting |

*Tab. 4-15: Info byte*

Telegram info byte

| Hex | Function |
|---|---|
| 0x00 | The telegram does not contain any additional offset information. |
| 0x04 | The telegram contains additional offset information which is placed as a word after the ***length***. The offset information determines the position of the payload in the input range. |

*Tab. 4-16: Telegram info byte*

Length

Length of the payload for the serial communication in bytes plus 2 bytes for ***RetVal***.

Offset

If the ***telegram info byte*** has the value 0x04, an offset is entered additionally. Otherwise, there is no ***offset*** in the telegram.

RetVal

| Hex | Function |
|---|---|
| 0x0517 | Invalid length (length = 0 or length >1024) |
| 0x080A | No free receive buffer available |
| 0x080C | Incorrect character received (character frame or parity error) |

*Tab. 4-17: RetVal*

Payload

Contains the received payload of the serial communication.

Acknowledgement

After the parent system has processed the data, it has to acknowledge the receipt to the communication processor. Only after that the communication processor provides new receive data.

| Bit | Hex | Function | |
|---|---|---|---|
| 3 ... 0 | | Reserved for transmitting | |
| 7 ... 4 | 0x08 | Acknowl-edgement | Idle run |
| | 0x0A | | Input range free for new data |
| | 0x0B | Command | Perform a reset of the communication processor |

*Tab. 4-18: Acknowledgement*

**Principle of transmission**
with fragmentation

| Parent system | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | Acknowledgement |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| 0 | Info byte |
| 1 | Telegram info byte |
| 2 | Length high byte |
| 3 | Length low byte |
| [4] | Offset high byte |
| [5] | Offset low byte |
| 6...n-1 | Payload |
| | |
| 0 | |

with n = number of the assigned bytes in the address range (IO size)

After the parent system has processed the data, it has to acknowledge the receipt to the communication processor.

For this purpose, it copies bit 3 ... 0 from byte 0 of the input range to bit 7 ... 4 from byte 0 of the output range. Only after that the communication processor provides new receive data.

**Calculation of the number of fragments**

Number of fragments = (length + 7) / (IO_Size-1)

Info byte

| Bit | Hex | Function |
|---|---|---|
| 3 ... 0 | 0x08 | Idle run - no data available |
| | 0x09 | Data are transmitted in fragments |
| | 0x0A | Data are transmitted without fragmentation |
| 7 ... 4 | | Reserved for transmitting |

*Tab. 4-19: Info byte*

Telegram info byte

| Hex | Function |
|---|---|
| 0x00 | The telegram does not contain any additional offset information. |
| 0x04 | The telegram contains additional offset information which is placed as a word after the ***length***. The offset information determines the position of the payload in the input range. |

*Tab. 4-20: Telegram info byte*

Length

Length of payload in bytes plus 2 bytes for ***RetVal***.

Offset

If the ***telegram info byte*** has the value 0x04, an offset is entered additionally. Otherwise, there is no ***offset*** in the telegram.

Calculation of the offset in case of fragmented transmission:

Data_offset = (fragment counter + 1) x (IO_SIZE -1) - 7 + ***offset***

| | | |
|---|---|---|
| with | Data_offset: | Offset of the data in the input range |
| | Fragment counter: | Absolute number of fragments |
| | IO_Size: | Number of the assigned bytes in the address range |
| | Offset: | Offset value in the telegram |

Payload

Contains the received payload of the serial communication.

Acknowledgement

After the parent system has processed the data, it has to acknowledge the receipt to the communication processor. Only after that the communication processor provides new receive data.

| Bit | Hex | Function | |
|---|---|---|---|
| 3 ... 0 | | Reserved for transmitting | |
| 7 ... 4 | 0x08 | Acknowledgement | Idle run |
| | 0x0A | | Input range free for new data |
| | 0x0B | Command | Perform a reset of the communication processor |

*Tab. 4-21: Acknowledgement*

### 4.5.3 Examples

**Transmitting data**
without fragmentation

IO size = 60 bytes, length = 40 bytes

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x0A command |
| 1 | 0x00 telegram info |
| 2 | 0x00 length high byte |
| 3 | 0x28 length low byte |
| 4 ... 43 | Payload byte 0 ... 39 |
| 44 ... 59 | not used |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | 0xA0 acknowledge-ment |

**Transmitting data**
with fragmentation

IO size = 16 bytes, length = 40 bytes

Header

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x09 command |
| 1 | 0x00 telegram info |
| 2 | 0x00 length high byte |
| 3 | 0x28 length low byte |
| 4 ... 15 | Payload byte 0 ... 11 |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | 0x90 Acknowledge-ment |

1st fragment

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x00 fragment |
| 1 ... 15 | Payload byte 12 ... 26 |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| 0 | 0x00 Acknowledge-ment |

2nd fragment

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x01 fragment |
| 1 ... 15 | Payload byte 27 ... 41 |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| 0 | 0x10 Acknowledge-ment |

Last fragment

| Parent system | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x0A command |
| 1 ... 8 | Payload byte 42 ... 49 |
| 11 ... 15 | not used |
| | |
| | |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| 0 | 0xA0 acknowledge-ment |

**Receiving data**
without fragmentation

IO size = 60 bytes, length = 40 bytes

| Parent system | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | 0xA0 acknowl-edgement |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x0A fragment info |
| 1 | 0x00 telegram info byte |
| 2 | 0x00 length high byte |
| 3 | 0x2A length low byte + 2 bytes |
| 4 | 0x00 return value high byte |
| 5 | 0x00 return value low byte |
| 6 ... 45 | Payload byte 0 ... 39 |
| 46 ... 59 | is not used |
| | |
| 0 | |

**Receiving data**
with fragmentation

IO size = 16 bytes, length = 40 bytes

Header

| Parent system | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| 0 | 0x90 Acknowl-edgement |

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x09 fragment info |
| 1 | 0x00 telegram info byte |
| 2 | 0x00 length high byte |
| 3 | 0x2A length low byte + 2 bytes |
| 4 | 0x00 return value high byte |
| 5 | 0x00 return value low byte |
| 6 ... 15 | Payload byte 0 ... 9 |
| | |
| 0 | |

1st fragment

| Parent system | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| 0 | 0x00 Acknowl-edgement |

◀

▶

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x00 fragment info |
| 1 ... 15 | Payload byte 10 ... 24 |
| | |
| 0 | |

Last fragment

| Parent system | |
|---|---|
| **Byte** | **Function** |
| | |
| | |
| | |
| 0 | 0xA0 acknowl-edgement |

◀

▶

| Communication processor | |
|---|---|
| **Byte** | **Function** |
| 0 | 0x0A fragment info |
| 1 ... 15 | Payload byte 25 ... 39 |
| | |
| 0 | |

## 4.6 Backplane bus communication

**Overview**

To process connection jobs on the PLC side, a user program is necessary in the CPU. For this purpose the following specific blocks by Murrelektronik GmbH are used for communication between CPU, communication processor and a communication partner:

| Block | Icon | Comment |
|---|---|---|
| FB 60 | SEND | Block for data transmission to a communication partner |
| FB 61 | RECEIVE | Block for receiving data from a communication partner |

*Tab. 4-22: Blocks for transmitting and receiving*

### 4.6.1 Installing blocks

**Installing blocks**

The specific blocks by Murrelektronik GmbH can be found in the Service area on www.murrelektronik.com under Downloads > ME LIB in form of a library for download.

The library is available as a packed zip filed. It must be imported into your project.

The following steps are necessary for this:

- Unpack file FX000011_Vxxx.zip
- Retrieve ***library***
- Open the library and transfer blocks to project

**FX000011_Vxxx.zip**

***Unpacking FX000011_Vxxx.zip***

➔ Start your unzip tool by double-clicking ***FX000011_Vxxx.zip***.

➔ Copy the Murrelektronik.ZIP file to your working directory.

*It is not necessary to continue unpacking this file.*

**Retrieving the library**

***Retrieve the library for the SPEED7 CPUs.***

➔ Start the SIMATIC Manager by Siemens.

➔ Go to ***File*** > ***Retrieve*** to open a dialog window and select the archive.

➔ Browse to your working directory.

➔ Click Murrelektronik.ZIP.

➔ Click ***Open***.

➔ Specify a target directory where the block has to be saved.

➔ Click ***OK***.

*The unpacking process starts.*

**Opening the library and transferring blocks to the project**

***Opening the library***

➔ Open the library after unpacking.

➔ Open your project.

➔ Copy the required blocks from the library into the ***Blocks*** directory of your project.

*In your user program, you have access to the specific blocks by Murrelektronik GmbH.*

**Communication principle**

The communication processor calls FB 60 and FB 61 cyclically and receives the data cyclically. The communication processor implements the transmission protocols for the communication partner. The transmission protocols can be parameterized in the hardware configuration.

Sending telegram

The CPU divides a telegram depending on the IO size in blocks and transfers it via the data channel to the communication processor. The communication processor combines the blocks in the send buffer and sends the entire telegram using the serial interface.

Receiving telegram

An entire telegram is divided into data blocks depending on the parameterized IO size and then transmitted to the backplane bus. The CPU combines the data blocks.

The communication processor receives the telegrams from the CPU asynchronously via the backplane bus.

If an entire telegram has been received via the serial interface, it is saved in a circular buffer (size of the circular buffer: 1024 bytes). The length of still available circular buffer gives the max. length of a telegram. Depending on the parameterization, up to 250 telegrams can be buffered, whose total length, however, may not exceed 1024 bytes. If the buffer is full, new telegrams are rejected.

Software handshake

Since the data exchange via backplane bus runs asynchronously, a software handshake between the communication processor and the CPU is used. For this purpose, the blocks FB 60 and FB 61 have a common parameter ***CONTROL***. Both blocks use for ***CONTROL*** different bits in the same flag byte.

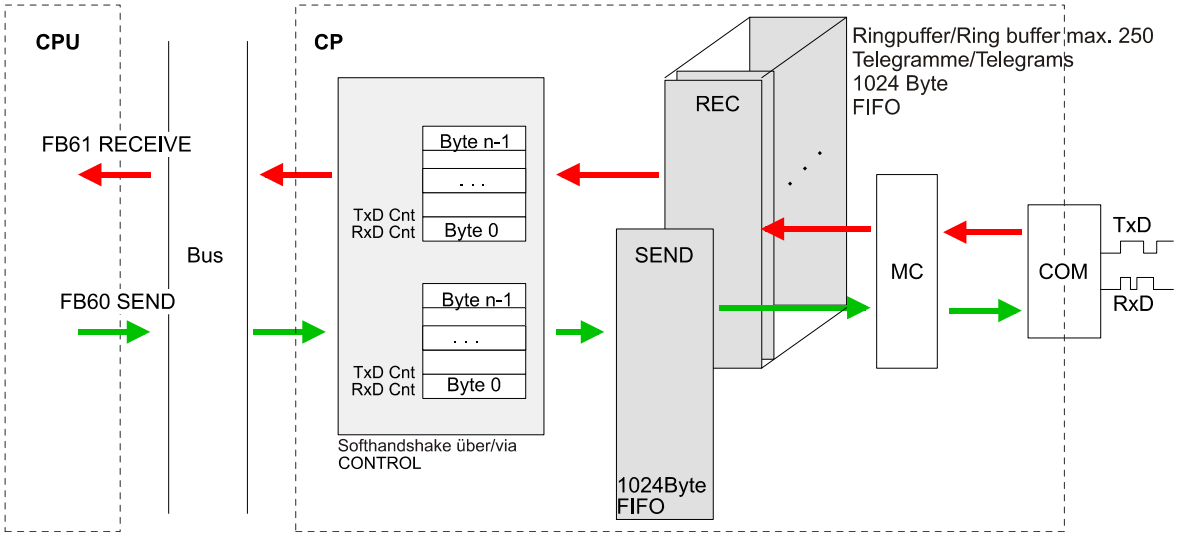



*Fig. 4-4: Communication processor CPU-57140*

**NOTE**
A minimum pulse duration is required to detect a signal change.
The decisive factors are:

- ➔ CPU cycle time,
- ➔ Refresh time on the communication processor,
- ➔ Reaction time of the communication partner.

## 4.6.2 FB 60 - SEND

**FB 60 - SEND Transmitting to the communication processor**

The FB 60 block outputs the data from the CPU to the communication processor. Use the identifiers ***DB_NO***, ***DBB_NO*** and ***LEN*** to specify the transmission range. A positive edge at ***REQ*** triggers the data transmission.

### Parameters

| Name | Declaration | Type | Description |
|---|---|---|---|
| REQ | IN | BOOL | Transmission release at positive edge |
| R | IN | BOOL | Triggers synchronous reset. |
| LADDR | IN | INT | Logical base address of the communication processor |
| DB_NO | IN | INT | Number of the data block of transmit data |
| DBB_NO | IN | INT | Number of the data byte - transmit data starting from data byte |
| LEN | IN | INT | Length of the send telegram in bytes |
| IO_SIZE | IN | WORD | Parameterized IO size of the module |
| DONE * | OUT | BOOL | Send job completed without errors |
| ERROR * | OUT | BOOL | Send job completed with errors<br>***STATUS*** contains the error information |
| STATUS * | OUT | WORD | Specification of the error if ERROR = 1 |
| CONTROL | IN_OUT | BYTE | Divided byte with ***RECEIVE*** block:<br>***SEND*** (bit 0 … 3)<br>***RECEIVE*** (bit 4 … 7) |

*Tab. 4-23: Parameters*

* The parameter is available until the next call by FB 60.

REQ

**Request - Transmission release**

A positive edge at the ***REQ*** input triggers the data transmission. Depending on the data volume, data transfer may continue over several program cycles.

R

**Synchronous reset**

| | |
|---|---|
| Initialization | Call SEND in the start-up OB once with all parameters and set ***R***. |
| Termination | Termination of the current job anytime with the signal state **1** at ***R***.<br>FB is reset to the initial state.<br>**NOTE**<br>Data which have already been received by the communication processor are still being sent to the communication partner. |
| Transmitting switched off | Static signal state **1** at ***R***. |

LADDR

**Peripheral address**

Use ***LADDR*** to enter the address of the communication processor to be addressed. It is the address which you have assigned to the communication processor in the hardware configuration.

DB_NO

**Data block number**

Number of the data block which contains the data to be transmitted. Zero is not allowed.

DBB_NO

**Data byte number**

Number of the data byte in the data block starting from which the transmit data are saved.

LEN

**Length**

Length of the payload to be transmitted. The following applies: 1 ≤ LEN ≤ 1024.

IO_SIZE

**Size I/O range**

Enter the size of the I/O range here. Depending on the parent system, the communication processor assigns the following optionally selectable number of bytes in the address range for the input and output respectively:

| System | Address range [byte] |
|---|---|
| PROFIBUS | 8, 20, 60 (can be selected) |
| PROFINET | 20, 60 (can be selected) |
| CANopen | 8 |
| EtherCAT | 60 |
| DeviceNET | 60 |
| ModbusTCP | 60 |

*Tab. 4-24: IO_SIZE: Size I/O range*

DONE

**Job done without errors**

DONE is set if the job has been completed without errors and STATUS = 0x0000.

ERROR

**Job done with errors**

ERROR is set. STATUS contains the error information.

STATUS

In case of error-free function, STATUS is 0x0000. As long as ERROR is set, the value in STATUS remains unchanged.

| Error code | Function |
|---|---|
| 0x0000 | No error available |
| 0x0202 | FB 60 block and communication processor are not synchronous<br>(Remedy: Trigger synchronous reset) |
| 0x0301 | DB is not valid |
| 0x070A | Transmission failed, partner does not respond or has acknowledged the job negatively |
| 0x0816 | Invalid LEN parameter (LEN = 0 or LEN > 1024) |
| 0x8181 | Job running (status and no error message) |

*Tab. 4-25: Error messages*

CONTROL

The SEND and RECEIVE blocks use the common ***CONTROL*** parameter for the handshake.

➔ Assign a common flag byte to the ***CONTROL***.

**Error indication**

The ***DONE*** output indicates ***Job completed without errors***, the value of ***STATUS*** is ***0***.

For ***ERROR***, the event number is displayed in ***STATUS***.

***DONE***, ***ERROR*** and ***STATUS*** are also output in case of reset of the block. In case of an error, the binary result BIE is reset. If the block is completed without errors, BIE has the state ***1***.

The parameters ***DONE***, ***ERROR*** and ***STATUS*** are always available only for one call of the block.

➔ Copy ***DONE***, ***ERROR*** and ***STATUS*** for further evaluation into a free data range.

### 4.6.3 FB 61 - RECEIVE

**FB 61 - RECEIVE Receiving from communication processor**

The FB 61 block supports data receiving of the CPU from the communication processor. Use the identifiers ***DB_NO***, ***DBB_NO*** and ***LEN*** to specify the receiving range. The length of read-in telegrams is saved in ***LEN***.

**Parameters**

| Name | Declaration | Type | Description |
|---|---|---|---|
| EN_R | IN | BOOL | Release for reading of data |
| R | IN | BOOL | Triggers synchronous reset. |
| LADDR | IN | INT | Logical base address of the communication processor |
| DB_NO | IN | INT | Number of the data block of transmit data |
| DBB_NO | IN | INT | Number of the data byte - receive data starting from data byte |
| IO_SIZE | IN | WORD | Configured IO size of the module |
| LEN | IN | INT | Length of the send telegram in bytes |
| NDR * | OUT | BOOL | Receive job done without errors |
| ERROR * | OUT | BOOL | Receive job done with errors<br>***STATUS*** contains the error information |
| STATUS * | OUT | WORD | Specification of the error if ERROR = 1 |
| CONTROL | IN_OUT | BYTE | Divided byte with ***SEND*** block:<br>***SEND*** (bit 0 … 3)<br>***RECEIVE*** (bit 4 … 7) |

*Tab. 4-26: Parameters*

* The parameter is available until the next call by FB 61.

EN_R

**Enable Receive - Enable reading**

Signal state ***1*** at EN_R enables the check aiming at finding out whether there are data from the communication processor to be read. Depending on the data volume, data transfer may continue over several program cycles.

The signal state ***0*** at EN_R can be used to cancel the running transmission. The canceled receive job is completed with an error message (***STATUS***). As long as ***0*** is active at EN_R, receiving is switched off.

R

**Synchronous reset**

| | |
|---|---|
| Initialization | Call RECEIVE in the start-up OB once with all parameters and set ***R***. |
| Termination | Termination of the current job anytime with the signal state **1** at ***R***.<br>FB is reset to the initial state. |
| Receiving disabled | Static signal state **1** at ***R***. |

LADDR

**Peripheral address**

Use ***LADDR*** to enter the address of the communication processor to be addressed. It is the address which you have assigned to the communication processor in the hardware configuration.

DB_NO

**Data block number**

Number of the data block which contains the data to be transmitted. Zero is not allowed.

DBB_NO

**Data byte number**

Number of the data byte in the data block starting from which the received data should be saved.

LEN

**Length**

Length of the payload to be transmitted. The following applies: 1 ≤ LEN ≤ 1024.

IO_SIZE

**Size I/O range**

Enter the size of the I/O range here. Depending on the parent system, the communication processor assigns the following optionally selectable number of bytes in the address range for the input and output respectively:

| System | Address range [byte] |
|---|---|
| PROFIBUS | 8, 20, 60 (can be selected) |
| PROFINET | 20, 60 (can be selected) |
| CANopen | 8 |
| EtherCAT | 60 |
| DeviceNET | 60 |
| ModbusTCP | 60 |

*Tab. 4-27: IO_SIZE: Size I/O range*

NDR

**New data ready**

New data are ready for the CPU in the communication processor.

ERROR

**Job done with errors**

ERROR is set. STATUS contains the error information.

STATUS

In case of error-free function, STATUS is 0x0000. As long as ERROR is set, the value in STATUS remains unchanged.

| Error code | Function |
|---|---|
| 0x0000 | No error available |
| 0x0202 | Block and communication processor are not synchronous<br>(Remedy: Trigger synchronous reset) |
| 0x0301 | DB is not valid |
| 0x070A | Transmission failed, partner does not respond or has acknowledged the job negatively |
| 0x0816 | Invalid LEN parameter (LEN = 0 or LEN > 1024) |
| 0x080A | No free receive buffer is available |
| 0x080C | Incorrect character received<br>(character frame or parity error) |
| 0x8181 | Job running (status and no error message) |

*Tab. 4-28: Error messages*

CONTROL

The SEND and RECEIVE blocks use the common ***CONTROL*** parameter for the handshake.

➔ Assign a common flag byte to the ***CONTROL***.

**Error indication**

The ***NDR*** output indicates ***Job done without errors / Data adopted***. For ***ERROR***, the corresponding event number is displayed in ***STATUS***. If no error has occurred, the value of ***STATUS*** is ***0***. NDR and ***ERROR***/***STATUS*** are also output in case of reset of FB. If an error has occurred, the binary result ***BIE*** is reset. If the block is completed without errors, ***BIE*** has the state ***1***.

The parameters NDR, ERROR and STATUS are always available only for the calling of block.

➔ Copy these parameters for further evaluation into a free data range.

## 4.7 Diagnostic data

**Overview**

Using parameterization you can enable a diagnostic interrupt for the module.

- Upon triggering of a diagnostic interrupt, the module provides $_{incoming}$ diagnostic data for diagnostics.
- As soon as the reasons for the triggering of a diagnostic interrupt are not present anymore, you receive automatically an$_{outgoing}$ diagnostic interrupt.
- During this period of time (1st diagnostic interrupt$_{incoming}$ until the last diagnostic interrupt$_{outgoing}$) the MF-LED of the module is lit.

DS Data record for access using CPU, PROFIBUS and PROFINET
Access using DS 0x01
Additional access to the first 4 bytes using DS 0x00

IX Index for access using CANopen
Access using IX 0x2F01.
Additional access to the first 4 bytes using IX 0x2F00

SX Subindex for access via EtherCAT

| Name | Bytes | Function | Default | DS | IX | SX |
|---|---|---|---|---|---|---|
| ERR_A | 1 | Diagnostics | 0x00 | 0x01 | 0x2F01 | 0x02 |
| MODTYP | 1 | Module information | 0x10 | | | 0x03 |
| ERR_C | 1 | reserved | 0x00 | | | 0x04 |
| ERR_D | 1 | Diagnostics | 0x00 | | | 0x05 |
| CHTYP | 1 | Channel type | 0x60 | | | 0x06 |
| NUMBIT | 1 | No. of diagnostic bits per channel | 0x08 | | | 0x07 |
| NUMCH | 1 | Number of channels of the module | 0x01 | | | 0x08 |
| CHERR | 1 | reserved | 0x00 | | | 0x09 |
| CH0ERR | 1 | reserved | 0x00 | | | 0x0A |
| CH1ERR ... CH7ERR | 7 | reserved | 0x00 | | | 0x0B ... 0x11 |
| DIAG_US | 4 | µs ticker | 0x00 | | | 0x12 |

*Tab. 4-29: Diagnostic data*

ERR_A
Diagnostics

| Byte | Bit 7 ... 0 | Description |
|---|---|---|
| 0 | 0 | set in case of ***Assembly fault*** |
| | 1 | set in case of ***Internal error*** |
| | 2 | reserved |
| | 3 | reserved |
| | 4 | set in case of ***Missing external power supply*** |
| | 5, 6 | reserved |
| | 7 | set in case of ***Parameter error*** |

*Tab. 4-30: Diagnostics*

MODTYP
Module information

| Byte | Bit 7 ... 0 | Description | |
|---|---|---|---|
| 0 | Bit 3 ... 0 | Module class | |
| | | 1111b | Communication processor |
| | Bit 4 | set if ***Channel information available*** | |
| | Bit 7 ... 5 | reserved | |

*Tab. 4-31: Module information*

ERR_D
Diagnosis

| Byte | Bit 7 ... 0 | Description |
|---|---|---|
| 0 | Bit 2 ... 0 | reserved |
| | Bit 3 | set in case of ***Internal diagnosis buffer overflow*** |
| | Bit 4 | set in case of ***Internal communication error*** |
| | Bit 7 ... 5 | reserved |

*Tab. 4-32: Diagnosis*

CHTYP
Channel type

| Byte | Bit 7 ... 0 | Description |
|---|---|---|
| 0 | 6 ... 0 | Channel type |
| | | 0x60 Communication processor |
| | 7 | reserved |

*Tab. 4-33: Channel type*

NUMBIT
Diagnostic bits

| Byte | Bit 7 ... 0 | Description |
|---|---|---|
| 0 | 7 ... 0 | Number of diagnostic bits of the module per channel (here 0x08) |

*Tab. 4-34: Diagnostic bits*

NUMCH
Channels

| Byte | Bit 7 ... 0 | Description |
|---|---|---|
| 0 | 7 ... 0 | Number of channels of a module (here 0x01) |

*Tab. 4-35: Channels*

CHERR, CH0ERR ...
CH7ERRreserved

| Byte | Bit 7 ... 0 | Description |
|---|---|---|
| 0 | 7 ... 0 | reserved |

*Tab. 4-36: CHERR, CH0ERR ... CH7ERR*

DIAG_US
µs ticker

| Byte | Bit 7 ... 0 | Description |
|---|---|---|
| 0 ... 3 | 7 ... 0 | Value of the µs ticker when generating diagnostic data |

*Tab. 4-37: µs ticker*

***µs ticker***

There is a timer (µs ticker) in the module, it is started by means of PowerON and starts counting from 0 after $2^{32}$-1 µs.

# 5 Serial communication protocols

## 5.1 Overview

**Serial transmission of a character**

The point-to-point connection between two communication partners is the simplest form of information exchange. For this procedure, the communication processor acts as the interface between a parent system and a communication partner with serial connection.

The data are transmitted serially. During serial data transmission, the individual bits of a byte of the information to be transmitted are transmitted in a defined sequence.

Character frame

During the bidirectional data traffic, a distinction is made between ***half-duplex*** and ***full-duplex*** mode. In the ***half-duplex*** mode the data are either transmitted or received at a time. A simultaneous data exchange can only happen in the ***full-duplex*** mode.

Each character to be transmitted is preceded by a synchronizing pulse acting as a ***start bit***. The end of the character transfer is formed by the ***stop bit***.

Beside ***start bit*** and ***stop bit***, further programmable agreements are necessary for a serial data transmission between the communication partners. This character frame consists of the following elements:

- Transfer rate (baud rate)
- Character and acknowledgement delay time
- Parity
- Number of data bits
- Number of stop bits

**Protocols**

The communication processor performs serial data transmission independently. For this purpose, the communication processor is equipped with drivers for the protocols.

The following protocols are described below:

- ASCII
- STEX/ETX
- 3964(R)
- Modbus (master, slave)

## 5.2 ASCII

### 5.2.1 Function

**Mode of operation**

The data communication via ASCII is a simple form of the data exchange. It can be compared with a multicast/broadcast function.

The ***character delay time*** (ZVZ) is used to separate the telegrams logically. During the character delay time, the transmitter must send its telegram to the receiver. A telegram is forwarded to the parent system only if it has been received completely.

As long as the ***Time delay after command*** (ZNA) has not elapsed, no new send job is accepted.

With these two time indications, a simple serial communication can be established.

Since no further measures for data back-up are taken for the ASCII transfer except for the use of the parity bit, the data transfer is very efficient, but not reliable. The parity helps to avoid the inversion of a bit within a character. If several bits of a character are inverted, this error can no longer be detected.

## 5.2.2 Parameterization data

DS Data record for access using CPU, PROFIBUS and PROFINET
IX Index for access using CANopen
SX Subindex for access via EtherCAT

| Name | Bytes | Function | Default | DS | IX | SX |
|---|---|---|---|---|---|---|
| PII_L | 1 | Process image length Input data | Values depend on the parent system | 0x02 | 0x3100 | 0x01 |
| PIQ_L | 1 | Process image length Output data | | 0x02 | 0x3101 | 0x02 |
| DIAG_EN | 1 | Diagnostic interrupt | 0x00 | 0x00 | 0x3102 | 0x03 |
| BAUD | 1 | Baud rate | 0x00 | 0x80 | 0x3103 | 0x04 |
| PROTOCOL | 1 | Protocol | 0x01 | 0x80 | 0x3104 | 0x05 |
| OPTION3 | 1 | Character frame | 0x13 | 0x80 | 0x3105 | 0x06 |
| OPTION4, 5 | 2 | ZNA 0 ... 65535 (ms) | 0 | 0x80 | 0x3106 ... 0x3107 | 0x07 ... 0x08 |
| OPTION6, 7 | 2 | ZVZ 0 ... 65535 (ms) | 250 | 0x80 | 0x3108 ... 0x3109 | 0x09 ... 0x0A |
| OPTION8 | 1 | Qty. Receive buffer | 1 | 0x80 | 0x310A | 0x0B |
| OPTION9 ... 14 | 6 | reserved | 0x00 | 0x80 | 0x310B ... 0x3110 | 0x0C ... 0x11 |

*Tab. 5-1: Parameterization data with ASCII*

**DIAG_EN**
**Diagnostic interrupt**

➔ Enable or disable the diagnostic function here.

*Value range* *0x00: lock* *Default: 0x00*
*0x40: release*

**BAUD**
**transfer rate**

➔ Specify the rate of data transmission in bit/s (baud).

The following setting ranges are available; other values are not permitted:

Default: 0x00 (9600 baud)

**Value range:**

| Hex | Baud | Hex | Baud | Hex | Baud |
|---|---|---|---|---|---|
| 0x00 | 9600 | 0x06 | 2400 | 0x0C | 38400 |
| 0x01 | 150 | 0x07 | 4800 | 0x0D | 57600 |
| 0x02 | 300 | 0x08 | 7200 | 0x0E | 76800 |
| 0x03 | 600 | 0x09 | 9600 | 0x0F | 115200 |
| 0x04 | 1200 | 0x0A | 14400 | 0x10 | 109700 |
| 0x05 | 1800 | 0x0B | 19200 | | |

*Tab. 5-2: Transfer rate*

**PROTOCOL**

Contains the selected protocol. This setting affects the further structure.

➔ Specify the value 0x01 for the ASCII protocol.

**OPTION3**
**Character frame**

| Byte | Bit 7 ... 0 | Description | |
|---|---|---|---|
| 0 | 1, 0 | **binary** | ***Data bits*** |
| | | 00 | 5 data bits |
| | | 01 | 6 data bits |
| | | 10 | 7 data bits |
| | | 11 | 8 data bits |
| | 3, 2 | **binary** | ***Parity*** |
| | | 00 | none |
| | | 01 | odd |
| | | 10 | even |
| | | 11 | even |
| | 5, 4 | **binary** | ***Stop bits*** |
| | | 01 | 1 |
| | | 10 | 1.5 |
| | | 11 | 2 |
| | 7, 6 | **binary** | ***Flow control*** |
| | | 00 | none |
| | | 10 | Hardware |
| | | 11 | XON/XOFF |

*Tab. 5-3: Character frame*

Default: 0x13 (data bits: 8, parity: none, stop bit: 1, flow control: none)

Data bits

Number of data bits used to represent a character.

Parity

For the parity check, the parity bit is added to the information bits. Its value (***0*** or ***1***) completes the value of all bits to obtain a specified state.

If no parity has been specified, the parity bit is set to ***1*** but not evaluated.

Stop bits

The stop bits are added after each character to be transmitted, they indicate the end of the character.

Flow control

The mechanism which synchronizes the data transfer if the transmitter sends the data faster than the receiver can process them. The flow control can be performed using hardware or software (XON/XOFF).

**Hardware flow control**

The RTS and CTS lines are used for the hardware flow control.

➔ Wire RTS and CTS correspondingly.

**Software flow control**

The software flow control uses the control characters XON=0x11 and XOFF=0x13 for control.

When using the software flow control, the data must not contain these two control characters.

**OPTION4, 5**
**ZNA**

***Entering waiting time (ZNA)***

➔ Enter ZNA in ms.

*The next send job starts only after this waiting time.*

| | | |
|---|---|---|
| Option4 | ZNA (High byte) | |
| Option5 | ZNA (Low byte) | |
| *Value range* | *0 ... 65535* (ms) | *Default: 0* |

**OPTION6, 7**
**ZVZ**

The character delay time (ZVZ) defines the maximum permitted time interval between two received characters within a telegram.

***Defining ZVZ in ms***

➔ Enter a ZVZ between 1 ... 65535.

***Communication processor calculates ZVZ***

➔ Enter 0 for ZVZ.

*The communication processor calculates the ZVZ on the basis of the baud rate (approx. double character time).*

| | | |
|---|---|---|
| Option6 | ZVZ (High byte) | |
| Option7 | ZVZ (Low byte) | |
| *Value range* | *0 ... 65535* (ms) | *Default: 250* |

**OPTION8**
**Number Receive buffers**

Defines the number of receive buffers.

***Using 1 receive buffer***

➔ Enter the value ***1***.

*If the input buffer is full, no further data can be received.*

***Using 2 ... 250 receive buffers***

➔ Enter a value between 2 ... 250.

*The received data are transferred into still free receive buffer.*

| | | |
|---|---|---|
| *Value range* | *0 ... 250* | *Default: 1* |

# 5.3 STX/ETX

## 5.3.1 Function

**Mode of operation**

STX/ETX is a simple protocol with header and trailer. STX/ETX is used for transmission of ASCII characters (0x20 … 0x7F). It is performed without block check character (BCC). Should data from the peripheral devices be read in, STX (start of text) must be available as a start character. The characters to be transmitted follow after that. ETX (end of text) must be available as the end character.

The payload, i.e. all characters between STX and ETX, is transferred to the parent system once the end character ETX has been received. During data transmission, the payload is transferred to the communication processor and from there transmitted to the communication partner with STX as start character and ETX as end character.

Telegram structure

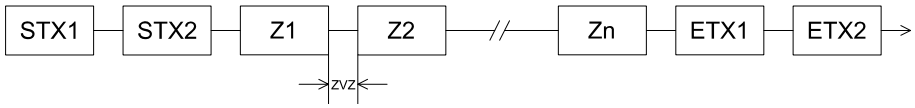



*Fig. 5-1: Telegram structure*

You can arbitrarily define up to 2 start and end characters and a waiting time (ZNA) for the transmitter.

## 5.3.2 Parameterization data

| | |
|---|---|
| DS | Data record for access using CPU, PROFIBUS and PROFINET |
| IX | Index for access using CANopen |
| SX | Subindex for access via EtherCAT |

| Name | Bytes | Function | Default | DS | IX | SX |
|---|---|---|---|---|---|---|
| PII_L | 1 | Process image length Input data | Values depend on the parent system. | 0x02 | 0x3100 | 0x01 |
| PIQ_L | 1 | Process image length Output data | | 0x02 | 0x3101 | 0x02 |
| DIAG_EN | 1 | Diagnostic interrupt | 0x00 | 0x00 | 0x3102 | 0x03 |
| BAUD | 1 | Baud rate | 0x00 | 0x80 | 0x3103 | 0x04 |
| PROTOCOL | 1 | Protocol | 0x02 | 0x80 | 0x3104 | 0x05 |
| OPTION3 | 1 | Character frame | 0x13 | 0x80 | 0x3105 | 0x06 |
| OPTION4, 5 | 2 | ZNA 0 ... 65535 (ms) | 0 | 0x80 | 0x3106 ... 0x3107 | 0x07 ... 0x08 |
| OPTION6, 7 | 2 | TMO 0 ... 65535 (ms) | 250 | 0x80 | 0x3108 ... 0x3109 | 0x09 ... 0x0A |
| OPTION8 | 1 | Number Start flags | 1 | 0x80 | 0x310A | 0x0B |
| OPTION9 | 1 | Start flag 1 | 3 | 0x80 | 0x310B | 0x0C |
| OPTION10 | 1 | Start flag 2 | 0 | 0x80 | 0x310C | 0x0D |
| OPTION11 | 1 | Number of end flags | 1 | 0x80 | 0x310D | 0x0E |
| OPTION12 | 1 | End flag 1 | 3 | 0x80 | 0x310E | 0x0F |
| OPTION13 | 1 | End flag 2 | 0 | 0x80 | 0x310F | 0x10 |
| OPTION14 | 1 | reserved | 0x00 | 0x80 | 0x3110 | 0x11 |

*Tab. 5-4: Parameterization data with STX/ETX*

**DIAG_EN**
**Diagnostic interrupt**

➔ Enable or disable the diagnostic function here.

*Value range* *0x00: lock* *Default: 0x00*
*0x40: release*

**BAUD**
**transfer rate**

➔ Specify the rate of data transmission in bit/s (baud).

The following setting ranges are available; other values are not permitted:

Default: 0x00 (9600 baud)

**Value range:**

| Hex | Baud | Hex | Baud | Hex | Baud |
|---|---|---|---|---|---|
| 0x00 | 9600 | 0x06 | 2400 | 0x0C | 38400 |
| 0x01 | 150 | 0x07 | 4800 | 0x0D | 57600 |
| 0x02 | 300 | 0x08 | 7200 | 0x0E | 76800 |
| 0x03 | 600 | 0x09 | 9600 | 0x0F | 115200 |
| 0x04 | 1200 | 0x0A | 14400 | 0x10 | 109700 |
| 0x05 | 1800 | 0x0B | 19200 | | |

*Tab. 5-5: Transfer rate*

**PROTOCOL**

Contains the selected protocol. This setting affects the further structure.

➔ Specify the value 0x02 for the STX/ETX protocol.

**OPTION3**
**Character frame**

| Byte | Bit 7 ... 0 | Description | |
|---|---|---|---|
| 0 | 1, 0 | **binary** | ***Data bits*** |
| | | 00 | 5 data bits |
| | | 01 | 6 data bits |
| | | 10 | 7 data bits |
| | | 11 | 8 data bits |
| | 3, 2 | **binary** | ***Parity*** |
| | | 00 | none |
| | | 01 | odd |
| | | 10 | even |
| | | 11 | even |
| | 5, 4 | **binary** | ***Stop bits*** |
| | | 01 | 1 |
| | | 10 | 1.5 |
| | | 11 | 2 |
| | 7, 6 | **binary** | ***Flow control*** |
| | | 00 | none |
| | | 10 | Hardware |
| | | 11 | XON/XOFF |

*Tab. 5-6: Character frame*

Default: 0x13 (data bits: 8, parity: none, stop bit: 1, flow control: none)

Data bits — Number of data bits used to represent a character.

Parity — For the parity check, the parity bit is added to the information bits. Its value (***0*** or ***1***) completes the value of all bits to obtain a specified state.

If no parity has been specified, the parity bit is set to ***1*** but not evaluated.

Stop bits — The stop bits are added after each character to be transmitted, they indicate the end of the character.

Flow control

The mechanism which synchronizes the data transfer if the transmitter sends the data faster than the receiver can process them. The flow control can be performed using hardware or software (XON/XOFF).

**Hardware flow control**

The RTS and CTS lines are used for the hardware flow control.

➔ Wire RTS and CTS correspondingly.

**Software flow control**

The software flow control uses the control characters XON=0x11 and XOFF=0x13 for control.

When using the software flow control, the data must not contain these two control characters.

**OPTION4, 5 ZNA**

***Entering waiting time (ZNA)***

➔ Enter ZNA in ms.

*The next send job starts only after this waiting time.*

| | | |
|---|---|---|
| Option4 | ZNA (High byte) | |
| Option5 | ZNA (Low byte) | |
| *Value range* | *0 ... 65535* (ms) | *Default: 0* |

**OPTION6, 7 TMO**

***Maximum permitted time interval between two telegrams (TMO)***

➔ Enter TMO in ms.

*The next telegram must start within the entered time.*

| | | |
|---|---|---|
| Option6 | TMO (High byte) | |
| Option7 | TMO (Low byte) | |
| *Value range* | *0 ... 65535* (ms) | *Default: 250* |

**OPTION8**
**Number Start flags**

***Set the number of start flags***

➔ Set 1 or 2 start flags.

*If **1** is set as the number of start flags, the contents of the 2nd start flag is ignored.*

*Range* *0 ... 2* *Default: 1*

**OPTION9, 10**
**Start flag 1**
**Start flag 2**

The start flag is sent before a telegram and indicates the start of transmission.

*Start flag 1, 2* *Range: 0 ... 255* *Default: 3 (start flag 1)*
*0 (start flag 2)*

***Entering ASCII value of the start character: 1 start flag***

✓ The preset value for start flag 2 is 0.

➔ Enter for ***Start flag 1*** (option9) a value in the range 0 ... 255.

*1 start flag is sent before the telegram.*

***Entering ASCII value of the start character: 2 start flags***

➔ Enter 2 for ***Number of start flags*** (Option8).

➔ Enter for ***Start flag 1*** (option9) a value in the range 0 ... 255.

➔ Enter for ***Start flag 2*** (option10) a value in the range 0 ... 255.

*2 start flags are sent before the telegram.*

**OPTION11**
**Number of end flags**

***Set the number of end flags***

➔ Set 1 or 2 end flags.

*If 1 is set as the number of end flags, the contents of the 2nd end flag is ignored.*

*Range* *0 ... 2* *Default: 1*

**OPTION12, 13**
**End flag 1**
**End flag 2**

The end flag is sent after a telegram and indicates the end of transmission.

*End flag 1, 2* *Range: 0 ... 255* *Default: 3 (end flag 1)*
*0 (end flag 2)*

***Entering the ASCII value of the end character: 1 end flag***

✓ The preset value for end flag 2 is 0.

➔ Enter for ***End flag 1*** (option12) a value in the range 0 ... 255.

*1 end flag is sent after the telegram.*

***Entering ASCII value of the end character: 2 end flags***

➔ Enter 2 for ***Number of end flags*** (Option11).

➔ Enter for ***End flag 1*** (option12) a value in the range 0 ... 255.

➔ Enter for ***End flag 2*** (option13) a value in the range 0 ... 255.

*2 end flags are sent after the telegram.*

# 5.4 3964(R)

## 5.4.1 Function

3964(R) controls the data transmission for a point-to-point connection between the communication processor and a communication partner. During data transmission, control characters are added to the payload. The communication partner can use these control characters to check whether the data received are complete and free of errors.

The 3964 protocol differs from the 3964R protocol only in the absence of Cyclic_Redundancy_Check (CRC). CRC allows more reliable transmission.

The following control characters are analyzed:

- STX Start of Text
- DLE Data Link Escape
- ETX End of Text
- BCC Block Check Character (only 3964R, BCC is the checksum of the 3964R protocol)
- NAK Negative Acknowledge

**DLE duplication**
If a DLE is transmitted as information, then it is sent twice to distinguish between the control character DLE and the data characters when establishing and terminating connection. The receiver restores the original state by removing the DLE duplication.

**Priority of the communication partners**
With 3964(R) a lower priority must be assigned to one communication partner. If both communication partners issue send jobs simultaneously, the partner with lower priority delays its send job.

**Procedure**

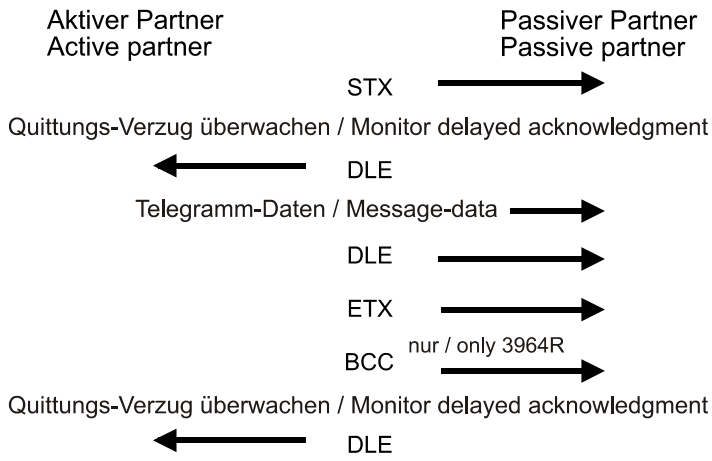



*Fig. 5-2: Procedure*

The communication processor transmits max. 250 bytes per telegram.

**Time-out periods**

Character delay time (ZVZ)

- ZVZ is monitored during the entire telegram receiving procedure.

Acknowledgement delay time (QVZ)

- QVZ is monitored between STX and DLE as well as between BCC and DLE.
- After STX: After the QVZ has elapsed after STX, STX is sent again.
  After 5 attempts a NAK is sent and the attempt to establish connection is terminated. The same happens if a NAK or any optional character is received after an STX.
- After the telegram: After the QVZ has elapsed after the telegram (after BCC byte) or if a character other than DLE has been received, the connection is established anew and the telegram is sent once again. After 5 attempts a NAK is sent and the attempt to establish connection is terminated.

**Passive operation**

If the driver waits for the connection to be established and a character other than STX is received, it sends NAK. If a NAK character is received, the driver does not send any response.

If the ZVZ is exceeded when receiving data, a NAK is sent and a new attempt to establish connection is awaited.

If the driver is not ready yet when the STX is received, it sends a NAK.

**3964R: Block Check Character (BCC byte)**

With 3964R a **B**lock **C**heck **C**haracter is added at the end of the telegram for further data back-up. The BCC byte is formed using an XOR connection for the data of the entire telegram including DLE/ETX.

When receiving a BCC byte which is different from the one that has been determined, a NAK is sent instead of the DLE.

**Initialization conflict**

If both communication partners are trying to establish connection within QVZ simultaneously, the partner with the lower priority sends the DLE and switches to receiving.

**Data Link Escape (DLE character)**

The DLE character in a telegram is duplicated by the driver, i.e. DLE/DLE is sent. During receiving procedure, duplicated DLEs are saved as one DLE in the buffer. In 3964R, the end of the telegram is always the combination DLE/ETX/BCC.

Control codes

| Command | Code |
|---|---|
| STX | 0x02 |
| ETX | 0x03 |
| DLE | 0x10 |
| NAK | 0x15 |

## 5.4.2 Parameterization data

DS Data record for access using CPU, PROFIBUS and PROFINET
IX Index for access using CANopen
SX Subindex for access via EtherCAT

| Name | Bytes | Function | Default | DS | IX | SX |
|---|---|---|---|---|---|---|
| PII_L | 1 | Process image length Input data | Values depend on the parent system | 0x02 | 0x3100 | 0x01 |
| PIQ_L | 1 | Process image length Output data | | 0x02 | 0x3101 | 0x02 |
| DIAG_EN | 1 | Diagnostic interrupt | 0x00 | 0x00 | 0x3102 | 0x03 |
| BAUD | 1 | Baud rate | 0x00 | 0x80 | 0x3103 | 0x04 |
| PROTOCOL | 1 | Protocol | 0x01 | 0x80 | 0x3104 | 0x05 |
| OPTION3 | 1 | Character frame | 0x13 | 0x80 | 0x3105 | 0x06 |
| OPTION4 | 2 | ZNA (x 20 ms) | 0 | 0x80 | 0x3106 | 0x07 |
| OPTION5 | 2 | ZVZ (x 20 ms) | 10 | 0x80 | 0x3107 | 0x08 |
| OPTION6 | 2 | QVZ (x 20 ms) | 10 | 0x80 | 0x3108 | 0x09 |
| OPTION7 | 2 | BWZ (x 20 ms) | 10 | 0x80 | 0x3109 | 0x0A |
| OPTION8 | 1 | STX repetitions | 5 | 0x80 | 0x310A | 0x0B |
| OPTION9 | 1 | DBL | 6 | 0x80 | 0x310B | 0x0C |
| OPTION10 | 1 | Priority | 0 | 0x80 | 0x310C | 0x0D |
| OPTION11 ... 14 | 4 | reserved | 0x00 | 0x80 | 0x310D ... 0x3110 | 0x0E ... 0x11 |

*Tab. 5-7: Parameterization data with 3964(R)*

**DIAG_EN**
**Diagnostic interrupt**

➔ Enable or disable the diagnostic function here.

*Value range* *0x00: lock* *Default: 0x00*
*0x40: release*

**BAUD**
**transfer rate**

➔ Specify the rate of data transmission in bit/s (baud).

The following setting ranges are available; other values are not permitted:

Default: 0x00 (9600 baud)

**Value range:**

| Hex | Baud | Hex | Baud | Hex | Baud |
|---|---|---|---|---|---|
| 0x00 | 9600 | 0x06 | 2400 | 0x0C | 38400 |
| 0x01 | 150 | 0x07 | 4800 | 0x0D | 57600 |
| 0x02 | 300 | 0x08 | 7200 | 0x0E | 76800 |
| 0x03 | 600 | 0x09 | 9600 | 0x0F | 115200 |
| 0x04 | 1200 | 0x0A | 14400 | 0x10 | 109700 |
| 0x05 | 1800 | 0x0B | 19200 | | |

*Tab. 5-8: Transfer rate*

**PROTOCOL**

Contains the selected protocol. This setting affects the further structure.

➔ Specify the value 0x03 for the 3964 protocol.

➔ Specify the value 0x04 for the 3964R protocol.

*Value range* *0x03: 3964*
*0x04: 3964R*
*Default: 0x03*

**OPTION3**
**Character frame**

| Byte | Bit 7 ... 0 | Description | |
|---|---|---|---|
| 0 | 1, 0 | **binary** | ***Data bits*** |
| | | 00 | 5 data bits |
| | | 01 | 6 data bits |
| | | 10 | 7 data bits |
| | | 11 | 8 data bits |
| | 3, 2 | **binary** | ***Parity*** |
| | | 00 | none |
| | | 01 | odd |
| | | 10 | even |
| | | 11 | even |
| | 5, 4 | **binary** | ***Stop bits*** |
| | | 01 | 1 |
| | | 10 | 1.5 |
| | | 11 | 2 |
| | 7, 6 | | ***reserved*** |

*Tab. 5-9: Character frame*

Default: 0x13 (data bits: 8, parity: none, stop bit: 1)

Data bits — Number of data bits used to represent a character.

Parity — For the parity check, the parity bit is added to the information bits. Its value (***0*** or ***1***) completes the value of all bits to obtain a specified state.

If no parity has been specified, the parity bit is set to ***1*** but not evaluated.

Stop bits — The stop bits are added after each character to be transmitted, they indicate the end of the character.

**OPTION4**
**ZNA**

***Entering waiting time (ZNA)***

➔ Enter ZNA as a factor of 20-ms steps.

*The next send job starts only after this waiting time.*

*Value range* *0 ... 255* *Default: 0*

**OPTION5**
**ZVZ**

The character delay time (ZVZ) defines the maximum permitted time interval between two received characters within a telegram.

***Defining ZVZ***

➔ Enter a ZVZ between 1 ... 255 as a factor of 20-ms steps.

***Communication processor calculates ZVZ***

➔ Enter 0 for ZVZ.

*The communication processor calculates the ZVZ on the basis of the baud rate (approx. double character time).*

*Value range* *0 ... 255* *Default: 10*

**OPTION6**
**QVZ**

The acknowledgment delay time (QVZ) defines the maximum permitted time interval up to the acknowledgment by the partner when establishing or terminating connection.

***Entering acknowledgement delay time (QVZ)***

➔ Enter QVZ as a factor of 20-ms steps.

| | | |
|---|---|---|
| *Value range* | *0 ... 255* | *Default: 10* |

**OPTION7**
**BWZ**

The block waiting time (BWZ) is the maximum time between the acknowledgment of a request telegram (DLE) and STX of the reaction telegram.

***Entering block waiting time (BWZ)***

➔ Enter BWZ as a factor of 20-ms steps.

| | | |
|---|---|---|
| *Value range* | *0 ... 255* | *Default: 10* |

**OPTION8**
**STX repetitions**

Maximum number of attempts performed by the communication processor when trying to establish connection.

***STX repetitions***

➔ Enter the max. number of STX repetitions.

| | | |
|---|---|---|
| *Value range* | *0 ... 255* | *Default: 5* |

**OPTION9**
**DBL**

When the block waiting time (BWZ) is exceeded, you can define the maximum number of repetitions for the request telegram using the DBL parameter.

***Entering repetitions for the request telegram (DBL)***

➔ Enter for DBL a value between 0 ... 255.

*If the attempts are not successful, the transmission is aborted.*

| | | |
|---|---|---|
| *Value range* | *0 ... 255* | *Default: 6* |

**OPTION10**
**Priority**

A communication partner has high priority if its transmission attempt has priority over the send request of the communication partner.

In 3964(R) the priorities of the communication partners must be different.

***Setting priority***

➔ Enter 0x00 for the low priority (low).

➔ Enter 0x01 for the high priority (high).

| | | |
|---|---|---|
| *Value range* | *0x00: low*<br>*0x01: high* | *Default: 0* |

# 5.5 Modbus

## 5.5.1 Overview

Modbus protocol is a communication protocol with hierarchical structure that specifies one master and several slaves.

Modbus functions physically over a serial half-duplex connection as a point-to-point connection with RS232 or as a multipoint connection with RS485.

**Master - Slave communication**

Bus conflicts do not occur since the master can always communicate with only one slave. After a request from the master, it waits for the slave response until a configurable waiting time has elapsed. During waiting, communication with another slave is not possible.

**Telegram structure**

The request telegrams sent by the master and the response telegrams of a slave have the same structure:

| Start character | Slave Address | Function Code | Data | flow control | End character |
|---|---|---|---|---|---|

**Broadcast with Slave address = 0**

A request can be addressed to a certain slave or sent to all slaves as a broadcast message. The slave address 0 is entered to identify a broadcast message.

Only writing jobs may be sent as broadcast.

**ASCII- RTU Mode**

There are two different transmission modes:

- ***ASCII mode:*** Each byte is transmitted in 2-sign ASCII code. The data are marked by the start sign and end sign. This makes the transmission transparent but slow.
- ***RTU mode:*** Each byte is transferred as one character. This leads to higher data throughput than in the ASCII mode. Time monitoring is used instead of start sign and end sign.

You can select the mode during parameterization.

## 5.5.2 Modbus on the communication processor

**Modbus operating modes**

The communication processor supports the following Modbus operating modes:

- Modbus Master
- Modbus Slave short
- Modbus Slave long

Modbus Master

In the ***Modbus Master*** mode, the Modbus communication is controlled by the PLC user program in your parent system.

Use the Modbus function code to obtain reading or writing access to your Modbus slaves by means of the Modbus master.

In the ***Modbus Master*** mode Modbus transmits up to 250 bytes of payload per telegram.

Modbus Slave short

In the ***Modbus Slave short*** mode the communication processor communicates with a Modbus master. Depending on the function code, the communication processor receives data from Modbus master or provides data. The data handling on the slave side is performed automatically.

In the ***Modbus Slave short*** mode Modbus transmits up to 250 bytes of payload per telegram.

Modbus Slave long

In the ***Modbus Slave long*** mode the communication processor transfers to the parent system only the changed data range starting with 0.

If the Modbus master requests data, a user program must be used to ensure that the relevant data are in the communication processor.

Writing master accesses may not be outside the receiving range.

**NOTE**
Only after all data are available in the communication processor, it sends a response telegram to the master!

## 5.5.3 Parameterization data

DS Data record for access using CPU, PROFIBUS and PROFINET
IX Index for access using CANopen
SX Subindex for access via EtherCAT

| Name | Bytes | Function | Default | DS | IX | SX |
|---|---|---|---|---|---|---|
| PII_L | 1 | Process image length Input data | Values depend on the parent system | 0x02 | 0x3100 | 0x01 |
| PIQ_L | 1 | Process image length Output data | | 0x02 | 0x3101 | 0x02 |
| DIAG_EN | 1 | Diagnostic interrupt | 0x00 | 0x00 | 0x3102 | 0x03 |
| BAUD | 1 | Baud rate | 0x00 | 0x80 | 0x3103 | 0x04 |
| PROTOCOL | 1 | Protocol | 0x0B | 0x80 | 0x3104 | 0x05 |
| OPTION3 | 1 | Character frame | 0x13 | 0x80 | 0x3105 | 0x06 |
| OPTION4 | 1 | Slave address | 1 | 0x80 | 0x3106 | 0x07 |
| OPTION5, 6 | 2 | Waiting time | 0 | 0x80 | 0x3107 ... 0x3108 | 0x08 ... 0x09 |
| OPTION7 ... 14 | 8 | reserved | 0x00 | 0x80 | 0x3107 ... 0x3110 | 0x0A ... 0x11 |

*Tab. 5-10: Parameterization data*

**DIAG_EN**
**Diagnostic interrupt**

➔ Enable or disable the diagnostic function here.

*Value range* *0x00: lock* *Default: 0x00*
*0x40: release*

**BAUD**
**transfer rate**

➔ Specify the rate of data transmission in bit/s (baud).

The following setting ranges are available; other values are not permitted:

Default: 0x00 (9600 baud)

**Value range:**

| Hex | Baud | Hex | Baud | Hex | Baud |
|---|---|---|---|---|---|
| 0x00 | 9600 | 0x06 | 2400 | 0x0C | 38400 |
| 0x01 | 150 | 0x07 | 4800 | 0x0D | 57600 |
| 0x02 | 300 | 0x08 | 7200 | 0x0E | 76800 |
| 0x03 | 600 | 0x09 | 9600 | 0x0F | 115200 |
| 0x04 | 1200 | 0x0A | 14400 | 0x10 | 109700 |
| 0x05 | 1800 | 0x0B | 19200 | | |

*Tab. 5-11: Transfer rate*

**PROTOCOL**

Contains the selected protocol. This setting affects the further structure. Default value: 0x0B (Modbus Master).

➔ Enter the value from the table for the required Modbus protocol.

| ASCII mode | Value | RTU mode | Value |
|---|---|---|---|
| Modbus Master | 0x0A | Modbus Master | 0x0B |
| Modbus Slave short | 0x0C | Modbus Slave short | 0x0D |
| Modbus Slave long | 0x1C | Modbus Slave long | 0x1D |

*Tab. 5-12: PROTOCOL*

**OPTION3**
**Character frame**

| Byte | Bit 7 ... 0 | Description | |
|---|---|---|---|
| 0 | 1, 0 | **binary** | ***Data bits*** |
| | | 00 | 5 data bits |
| | | 01 | 6 data bits |
| | | 10 | 7 data bits |
| | | 11 | 8 data bits |
| | 3, 2 | **binary** | ***Parity*** |
| | | 00 | none |
| | | 01 | odd |
| | | 10 | even |
| | | 11 | even |
| | 5, 4 | **binary** | ***Stop bits*** |
| | | 01 | 1 |
| | | 10 | 1.5 |
| | | 11 | 2 |
| | 7, 6 | | ***reserved*** |

*Tab. 5-13: Character frame*

Default: 0x13 (data bits: 8, parity: none, stop bit: 1)

Data bits — Number of data bits used to represent a character.

Parity — For the parity check, the parity bit is added to the information bits. Its value (***0*** or ***1***) completes the value of all bits to obtain a specified state.

If no parity has been specified, the parity bit is set to ***1*** but not evaluated.

Stop bits — The stop bits are added after each character to be transmitted, they indicate the end of the character.

**OPTION4**
**Slave address**

In Modbus Master the parameter is ignored.

***Entering the address for the Modbus slave***

➔ Specify an address for the Modbus slave in the Slave protocol.

*With this address Modbus accesses the function code.*

*Value range* *0 ... 255* *Default: 1*

**OPTION5, 6 Waiting time**

In Modbus slave the parameter is ignored.

***Specifying waiting time for the Modbus master***

➔ Specify a waiting time in ms for the Modbus master. With 0 the waiting time is determined automatically depending on the protocol according to the following formula:

| | | |
|---|---|---|
| Modbus ASCII | 50 ms + (2926000 ms / baud rate) x bit/s | with baud rate in bit/s |
| Modbus RTU | 50 ms + (5190000 ms / baud rate) x bit/s | with baud rate in bit/s |

Option5: waiting time (High byte)

Option6: waiting time (Low byte)

*Value range: 0...60000 (ms)* *Default: 0*

## 5.6 Modbus use

### 5.6.1 Overview

**IO size**

The number of input and output data depends on the IO size. For the communication processor it can be parameterized using GSD.

**Prerequisite for the operation**

The following components are required for the use of Modbus modules:

- Master system consisting of the Cube20S system with communication processor
- Slave system consisting of the Cube20S system with communication processor
- Siemens SIMATIC manager or WinPLC7 by Murrelektronik
- GSD file
- Murrelektronik blocks Fx000011_Vxxx.zip
- Serial connection between both communication processors

**Parameterization**

Always perform a hardware configuration for the Modbus communication processor. For this purpose, the GSD file by Murrelektronik must be integrated in the hardware catalog.

***Procedure for parameterization***

1 | Plan a Cube20S system.
2 | Install the selected GSD file in the hardware catalog.
3 | Start the Siemens SIMATIC manager or WinPLC7 by Murrelektronik.
4 | Embed a communication processor marked with **Modbus**.
5 | Configure the communication processor according to your specifications.
6 | Transfer your project to the PLC.

**PLC program**

Beside ***Modbus Slave short***, a PLC program is necessary for the communication. The communication is performed by means of blocks which are embedded in form of Murrelektronik library Fx000011-Vxxx.zip in the configuration tool.
You can find them on our homepage under www.murrelektronik in the Service area.

## 5.6.2 Communication

**Communication options**

Below you will find the communication options between Modbus master and Modbus slave.

Combinations:

- Modbus Master ↔ Modbus Slave short
- Modbus Master ↔ Modbus long

**Modbus Master ↔ Modbus Slave short**

**Modbus Master**

In the Master mode the communication processor communicates using the blocks FB 60 - SEND and FB 61 - RECEIVE. It transmits up to 250 bytes of payload per telegram.

**Modbus Slave short**

In the Modbus Slave short mode the amount of payload for input and output is limited by the IO size. Only hardware configuration is required on the slave side for deployment.

Procedure

***System structure***

➔ Install one Cube20S system for master and slave side respectively, it should have a communication processor on each side.

➔ Connect both systems using the serial interface.

***Planning Modbus Master***

➔ Plan the master side.

➔ Parameterize the communication processor in the hardware configuration.

A PLC user program is required for the communication, the structure of the program should be planned according to the structure in the table (Tab. 5-14: ).

| | |
|---|---|
| OB 100 | Single call of the blocks ***FB 60 - SEND and FB 61 - RECEIVE*** with all parameters and set ***R*** for initialization. |
| OB 1 | Call of ***FB 60 - SEND*** with error evaluation.<br>The PLC user program creates the telegram according to the Modbus specifications in the send block. |
| | Call of ***FB 61 - RECEIVE*** with error evaluation.<br>The PLC user program saves the data according to the Modbus specifications in the receive block. |

*Tab. 5-14: Structure of the PLC user program*

***Planning Modbus Slave short***

➔ Plan the slave side.

➔ Configure the communication processor in the hardware configuration.

➔ Define for the input and output range the start address starting from which the data are saved (depending on the IO size).

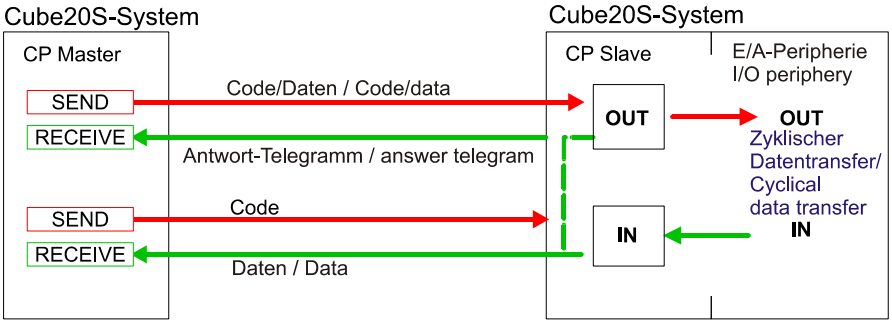



*Fig. 5-3: System Cube20S Master <-> Slave short*

**Master ↔ Slave long**

**Modbus Master**

In the Master mode the communication processor communicates using the blocks FB 60 - SEND and FB 61 - RECEIVE. It transmits up to 250 bytes of payload per telegram.

**Modbus Slave long**

In the Modbus Slave long mode the communication processor transmits with FB 61 - RECEIVE only the changed data range starting with 0 to the CPU. If the master requests data, the relevant data must be available in the communication processor. With FB 60 - SEND, the CPU transmits the required data range starting with 0 to the communication processor.

Procedure

***System structure***

➔ Install one Cube20S system for master and slave side respectively, it should have a communication processor on each side.

➔ Connect both systems using the serial interface.

### *Planning Modbus Master*

➔ Plan the master side.

➔ Configure the communication processor in the hardware configuration.

A PLC user program is required for the communication, the structure of the program should be planned according to the structure in the table (Tab. 5-15: ).

| | |
|---|---|
| OB 100 | Single call of the blocks ***FB 60 - SEND and FB 61 - RECEIVE*** with all parameters and set ***R*** for initialization. |
| OB 1 | Call of ***FB 60 - SEND*** with error evaluation.<br>The PLC user program creates the telegram according to the Modbus specifications in the send block. |
| | Call of ***FB 61 - RECEIVE*** with error evaluation.<br>The PLC user program saves the data according to the Modbus specifications in the receive block. |

*Tab. 5-15: Structure of the PLC user program*

### *Planning Modbus Slave long*

➔ Plan the slave side.

➔ Configure the communication processor in the hardware configuration.

A PLC user program is required for the communication, the structure of the program should be planned according to the structure in the table (Tab. 5-16: ).

| | |
|---|---|
| OB 100 | Single call of the blocks ***FB 60 - SEND and FB 61 - RECEIVE*** with all parameters and set ***R*** for initialization. |
| OB 1 | Call of ***FB 60 - SEND*** with error evaluation.<br>The PLC user program saves a range starting with 0 in the communication processor which can be accessed by the master via Modbus. |
| | Call of ***FB 61 - RECEIVE*** with error evaluation.<br>The communication processor transfers a data range into the CPU. |
| | If the data are changed by the master, the CPU receives only the data which have been changed. |

*Tab. 5-16: Structure of the PLC user program*

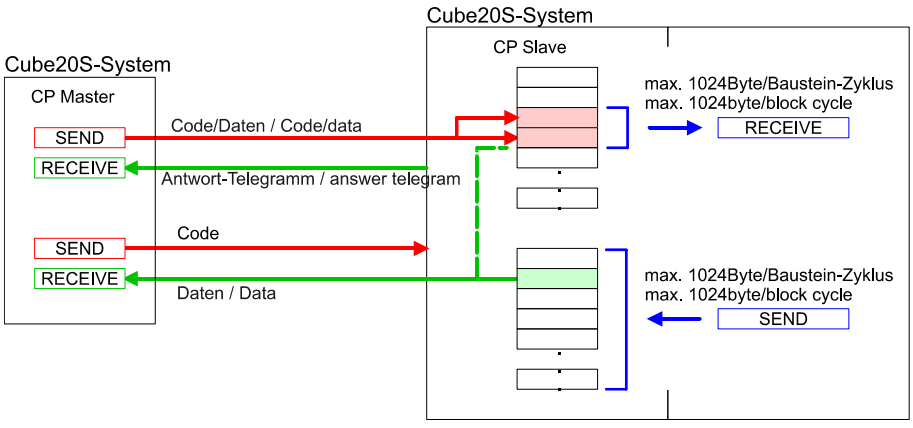



*Fig. 5-4: System Cube20S Master ↔ Slave long*

# 5.7 Modbus function codes

## 5.7.1 Definitions

**Naming conventions for Modbus**

Modbus distinguishes between bit and word access

- Bit = **Coil**
- Word = **Register**
- Bit input = **Input status**
- Bit output = **Coil status**
- Word input = **Input register**
- Word output = **Holding register**

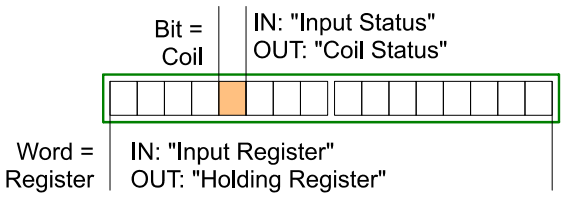



*Fig. 5-5: Naming conventions*

**Range definitions**

Usually Modbus accesses using the ranges 0x, 1x, 3x and 4x.

- 0x and 1x: Access to digital bit ranges
- 3x and 4x: Access to analog word ranges

**NOTE**
The communication processor does not distinguish between digital and analog data, therefore, the assignment in the table (Tab. 5-17: ) applies.

| Range | | Access using function code |
|---|---|---|
| 0x | Bit range for the output data of the master | 0x01, 0x05, 0x0F |
| 1x | Bit range for the input data of the master | 0x02 |
| 3x | Word range for the input data of the master | 0x04 |
| 4x | Word range for the output data of the master | 0x03, 0x06, 0x10 |

*Tab. 5-17: Assignment of the bit and word ranges*

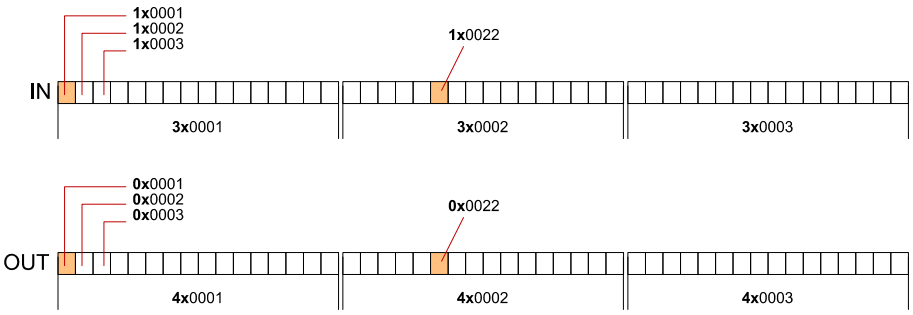



*Fig. 5-6: Range definitions*

## 5.7.2 Overview

**Access**

Using the following function codes you can access a slave from a Modbus master. The function codes are described from the point of view of the master.

| Code | Command | Description |
|---|---|---|
| 0x01 | Read n bits | Read n bits of master output range 0x |
| 0x02 | Read n bits | Read n bits of master input range 1x |
| 0x03 | Read n words | Read n words of master output range 4x |
| 0x04 | Read n words | Read n words of master input range 3x |
| 0x05 | Write 1 bit | Write 1 bit in master output range 0x |
| 0x06 | Write 1 word | Write 1 word in master output range 4x |
| 0x0F | Write n bits | Write n bits in master output range 0x |
| 0x10 | Write n words | Write n words in master output range 4x |

*Tab. 5-18: Access*

View for **input** and **output** data

The function codes are described from the point of view of the master.

Data sent by the master to the slave are referred to as **output** data (OUT) up to the target.

Data received by the master from the slave, are referred to as **input** data (IN).

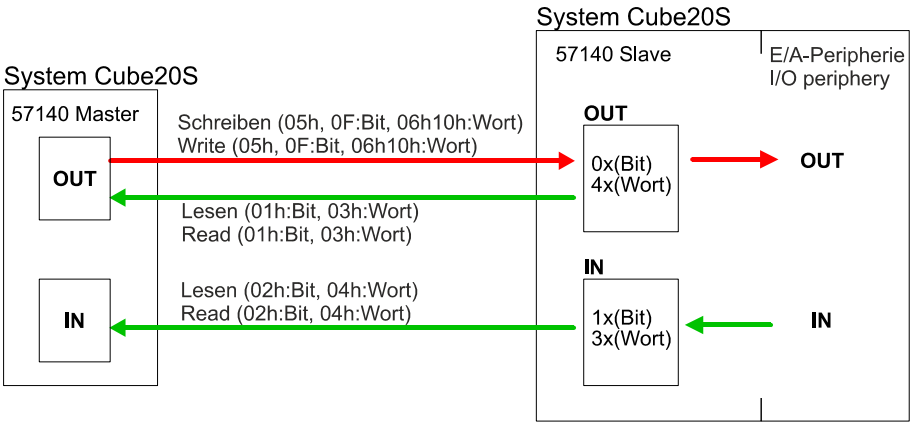



*Fig. 5-7: View for input and output data*

**Response of the slave**

If the slave responds with an error, the function code is returned with **OR 0x80**.

If there is no error, the function code is returned.

| | | | |
|---|---|---|---|
| Slave response: | Function code OR 0x80 | → | Error |
| | Function code | → | OK |

**NOTE**

The following applies to the order of bytes in a word:

| 1 word | |
|---|---|
| High byte | Low byte |

**Checksum CRC or LRC**

**CRC in the RTU mode, LRC in the ASCII mode**

The checksums are automatically attached to each telegram. They are not displayed in the data block.

**Slave address**

The slave address must be identical with the parameterized ***slave address*** (OPTION4).

## 5.7.3 Function codes

**Read n bits 0x01, 0x02**

Code 0x01: Read n bits of master output range 0x

Code 0x02: Read n bits of master input range 1x

| Slave Address | Function code | Address 1st bit | No. of bits | Checksum CRC/LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-19: Command telegram*

| Slave address | Function code | Number of the read bytes | Data 1st byte | Data 2nd byte | ... | Checksum CRC/LRC |
|---|---|---|---|---|---|---|
| 1 byte | 1 byte | 1 byte | 1 byte | 1 byte<br>max. 250 bytes | | 1 word |

*Tab. 5-20: Response telegram*

**Read n words 0x03, 0x04**

0x03: Read n words of master output range 4x

00x4: Read n words of master input range 3x

| Slave address | Function code | Address 1st bit | No. of words | Checksum CRC/LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-21: Command telegram*

| Slave address | Function code | Number of the read bytes | Data 1st word | Data 2nd word | ... | Checksum CRC/LRC |
|---|---|---|---|---|---|---|
| 1 byte | 1 byte | 1 byte | 1 word | 1 word<br>max. 125 words | | 1 word |

*Tab. 5-22: Response telegram*

**Write 1 bit 0x05**

Code 0x05: Write 1 bit in master output range 0x

The status is changed in **Statusbit** with the following values:

***Status bit*** = 0x0000 → bit = 0

***Status bit*** = 0xFF00 → bit = 1

| Slave address | Function code | Address Bit | Bit status | Checksum CRC/LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-23: Command telegram*

| Slave address | Function code | Address Bit | Bit status | Checksum CRC/LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-24: Response telegram*

**Write 1 word 0x06**

Code 0x06: Write 1 word in master output range 4x

| Slave address | Function code | Address Word | Value Word | Checksum CRC/LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-25: Command telegram*

| Slave address | Function code | Address Word | Value Word | Checksum CRC/LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-26: Response telegram*

**Write n bits 0x0F**

Code 0x0F: Write n bits in master output range 0x

The number of bits has also to be specified in bytes.

| Slave address | Function code | Address 1st bit | No. of bits | No. of bytes | Data 1st byte | Data 2nd byte | ... | Checksum CRC/LRC |
|---|---|---|---|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 byte | 1 byte | 1 byte | 1 byte | 1 word |
| | | | | | max. 250 bytes | | | |

*Tab. 5-27: Command telegram*

| Slave address | Function code | Address 1st bit | No. of bits | Checksum CRC/ LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-28: Response telegram*

**Write n words 0x10**

Code 0x10: Write n words in master output range

| Slave address | Function code | Address 1st word | No. of words | No. of bytes | Data 1st word | Data 2nd word | ... | Checksum CRC/LRC |
|---|---|---|---|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 byte | 1 word | 1 word | 1 word | 1 word |
| | | | | | max. 125 words | | | |

*Tab. 5-29: Command telegram*

| Slave address | Function code | Address 1st word | No. of words | Checksum CRC/LRC |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 word | 1 word | 1 word |

*Tab. 5-30: Response telegram*

## 5.8 Modbus - Error messages

**Overview**

For the communication in Modbus, there are 2 error types:

- Master does not receive any valid data
- Slave responds with an error message

**Master does not receive any valid data**

If the slave does not respond within the predefined waiting time or if a telegram is faulty, the master enters an error message in plain text in the receiving block.

Error messages

| | |
|---|---|
| ERROR01 NO DATA | ***Error no data***<br>The master has not received any telegram within the waiting time. |
| ERROR02 D LOST | ***Error data lost***<br>No data are available.<br>The receive buffer is either full or an error occurred in the reception buffer. |
| ERROR03 F OVERF | ***Error frame overflow***<br>Master has not recognized the telegram end and the maximum telegram length has been exceeded. |
| ERROR04 F INCOM | ***Error frame incomplete***<br>Master has only received a partial telegram. |
| ERROR05 F FAULT | ***Error frame fault***<br>The checksum within a telegram is not correct. |
| ERROR06 F START | ***Error frame start***<br>The start character is incorrect.<br>**NOTE:** The error can only occur with Modbus-ASCII. |

**Slave responds with an error message**

If the slave responds with an error, the function code is returned with 0x80 **logical OR connection**.

**Example:**

| | | |
|---|---|---|
| DB11.DBD 0 | DW#16#05900000 | **Response telegram** |
| | with 05 → | Slave address 0x05 |
| | 90 → | Function code 0x90 |
| | | (error message, because 0x10 OR 0x80 = 0x90) |
| | 0000 → | The residual data are irrelevant because an error has been reported back |

# 6 General data

| | | | |
|---|---|---|---|
| **Conformity** | | | |
| | CE | 2004/108/EC | EMC Directive |
| | | 2011/65/EU | RoHS |
| **Personal and device protection** | | | |
| | Ingress protection | EN 60529 | IP20 |
| | **Electric isolation** | | |
| | To fieldbus | - | DC-isolated |
| | To process level | - | DC-isolated |
| | Dielectric strength | EN 61131-2 | - |
| | **Insulation voltage to ground** | | |
| | Inputs / outputs | - | 50 V AC/DC,<br>with test voltage 500 V AC |
| | Protective measures | - | against short-circuit |
| **Ambient conditions** | | | |
| | **Climatic** | | |
| | Storage / transport | EN 60068-2-14 | -25 … +70 °C |
| | **Operation** | | |
| | Horizontal installation | EN 61131-2 | 0 … +60 °C |
| | Vertical installation | EN 61131-2 | 0 … +60 °C |
| | Humidity | EN 60068-2-30 | RH1 (without condensation, relative humidity 10 … 95 %) |
| | Pollution | EN 61131-2 | Pollution degree 2 |
| | **Mechanical** | | |
| | Vibration | EN 60068-2-6 | 1 g, 9 Hz ... 150 Hz |
| | Shock | EN 60068-2-27 | 15 g, 11 ms |
| **Installation conditions** | | | |
| | Place of installation | - | Inside the switch cabinet |
| | Installation position | - | Horizontal and vertical |
| | Fastening | - | 35 mm DIN rail |

| | | |
|---|---|---|
| **Mechanical data** | **Housing** | |
| | Material | PPE / PPE GF10 |
| | Dimensions (W x H x D) | 12.9 x 109 x 76.5 mm |
| | Weight | 60 g |
| **Ambient conditions** | | |
| | Operating temperature | 0 °C to 60 °C |
| | Storage temperature | -25 °C to 70 °C |
| **Certifications** | | |
| | Certification according to UL 508 | yes |

| EMC / Standard | | | Notes |
|---|---|---|---|
| | Emitted interference | EN 61000-6-4 | Class A (industrial environments) |
| | Immunity Zone B | EN 61000-6-2 | Industrial environments |
| | | EN 61000-4-2 | ESD<br>8 kV with air discharge (severity grade 3),<br>4 kV with contact discharge (severity grade 2) |
| | | EN 61000-4-3 | HF irradiation (housing)<br>80 MHz ... 1000 MHz, 10 V/m, 80 % AM (1 kHz)<br>1.4 GHz ... 2.0 GHz, 3 V/m, 80 % AM (1 kHz)<br>2 GHz ... 2.7 GHz, 1 V/m, 80 % AM (1 kHz) |
| | | EN 61000-4-6 | conducted 150 kHz … 80 MHz,<br>10 V, 80 % AM (1 kHz) |
| | | EN 61000-4-4 | Burst, severity grade 3 |
| | | EN 61000-4-5 | Surge, installation class 3 *) |

*) Due to single high-energy impulses, a suitable external wiring with lightning protection elements is required for surge, e.g. lightning arresters and surge arrester.

# 7 Technical data

| Power consumption / power dissipation | | |
|---|---|---|
| | Power consumption from the backplane bus | 100 mA |
| | Power consumption from load voltage L+ (without load) | 10 mA |
| | Power dissipation | 1 W |

| Status, interrupt, diagnoses | | |
|---|---|---|
| | Status indication | yes |
| | Alarms | yes, programmable |
| | Process interrupt | no |
| | Diagnostic interrupt | yes, programmable |
| | Diagnostic function | yes, programmable |
| | Diagnostic information readable | possible |
| | Supply voltage indication | Green LED |
| | Collective error indication | Red LED |
| | Channel error | Red LED |

| Point-to-Point communication | | |
|---|---|---|
| | PtP communication | yes |
| | Interface, electrically isolated | yes |
| | RS232 interface | yes |
| | RS422 interface | - |
| | RS485 interface | - |
| | Connection | Terminal module |
| | Min. transfer rate | 150 bit/s |
| | Max. transfer rate | 115.2 kbit/s |
| | Cable length, max. | 15 m |

| Point-to-Point protocols | | |
|---|---|---|
| | Protocol ASCII | yes |
| | Protocol STX/ETX | yes |
| | Protocol 3964(R) | yes |
| | Protocol RK512 | - |
| | Protocol USS Master | - |
| | Protocol Modbus Master | yes |
| | Protocol Modbus Slave | yes |
| | Special protocols | - |

| Data sizes | | |
|---|---|---|
| | Input bytes | 8 / 20 / 60 |
| | Output bytes | 8 / 20 / 60 |
| | Parameter bytes | 21 |
| | Diagnostic bytes | 20 |

## 7.1 Protocols

| ASCII | |
|---|---|
| Telegram length | max. 1024 bytes |
| Baud rate | 150, 300, 600, 1 200, 1 800, 2.400, 4 800, 7 200, 9 600, 14 400, 19 200, 38 400, 57 600, 76 800, 109 700, 115 200 baud |
| Character delay time (ZVZ) | 0 ... 65 535 (in ms steps), with ZVZ = 0 the 3-fold character time is used. |
| Flow control | no hardware, XON/XOFF |
| Number of telegrams which can be buffered | max. 250 |
| End flag of a telegram | after the character delay time (ZVZ) has elapsed |

| STX/ETX | |
|---|---|
| Telegram length | max. 1,024 bytes |
| Baud rate | 150, 300, 600, 1 200, 1 800, 2.400, 4 800, 7 200, 9 600, 14 400, 19 200, 38 400, 57 600, 76 800, 109 700, 115 200 baud |
| Character delay time (TMO) | 0 ... 65 535 (in ms steps), with TMO = 0 the 3-fold character time is used. |
| Flow control | no hardware, XON/XOFF |
| Number of telegrams which can be buffered | max. 250 |
| End flag of a telegram | using parameterized end character |
| Number of start characters | 0 ... 2 (characters, programmable) |
| Number of end characters | 0 ... 2 (characters, programmable) |

| 3964, 3964R | |
|---|---|
| Telegram length | max. 1,024 bytes |
| Baud rate | 150, 300, 600, 1 200, 1 800, 2.400, 4 800, 7 200, 9 600, 14 400, 19 200, 38 400, 57 600, 76 800, 109 700, 115 200 baud |
| Block check character | only 3964R |
| Priority | low/high |
| Character delay time (ZVZ) | 0 ... 255 (in 20-ms steps), with ZVZ = 0 the 3-fold character time is used. |
| Acknowledgement delay time (QVZ) | 0 ... 255 (in 20-ms steps), with QVZ = 0 the 3-fold character time is used. |
| Number of connection attempts | 0 ... 255 |
| Number of transfer attempts | 1 ... 255 |

| Modbus | |
|---|---|
| Telegram length | max. 258 bytes |
| Baud rate | 150, 300, 600, 1 200, 1 800, 2.400, 4 800, 7 200, 9 600, 14 400, 19 200, 38 400, 57 600, 76 800, 109 700, 115 200 baud |
| Mode | Master ASCII, Master RTU,<br>Slave ASCII short, Slave RTU short,<br>Slave ASCII long, Slave, RTU long |
| Address | 1 ... 255 |
| Waiting time | automatic, 1 ... 60,000 ms |

# 8 Annex

## 8.1 Accessories

***Bus cover***
***Art. no. 57190***

*Fig. 8-1: Bus cover*

**Carrier for shield busses**

The shield busses (10 mm x 3 mm) to connect cable shields are fastened on the carrier.

**NOTE**
Carriers for shield busses, shield busses and cable shield fasteners are not included in the delivery.

**Installing the carrier**

✓ Prerequisite: The Cube20S system has been completely mounted.
➔ If the DIN rail is flat, break the spacer off the carrier.
➔ Plug the carrier in the terminal module below the terminal block until it engages.

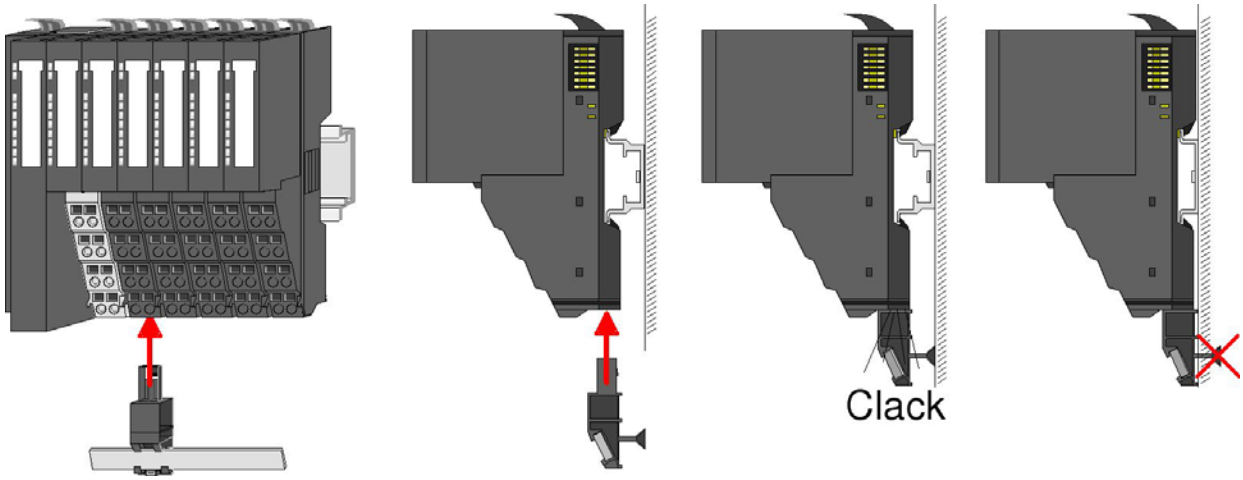



*Fig. 8-2: Installing the carriers for shield busses*

## 8.2 Glossary

**General terms:**

| Term | Meaning |
|---|---|
| Intended purpose | Use of a product, process, or feature according to the specifications, instructions, and information supplied by the MANUFACTURER. |
| Bit | Binary digit |
| Byte | 1 byte corresponds to 8 bit |
| DI | Digital inputs |
| DIN | Deutsches Institut für Normung (German Institute for Standardization) |
| I/O | Input/Output |
| Directive 2004/108/CE | EMC Directive |
| EMC | Electromagnetic compatibility |
| EN | European standard |
| ESD | Electrostatic discharges |
| FE | Functional earth |
| I | Current |
| IEC | International Electrotechnical Commission, international standardization institute |
| IN | Input |
| IP20 | Ingress Protection, protection class according to DIN EN 60529<br>1st code digit = Protection against accidental contact and solid foreign objects<br>2nd code digit = Protection against ingress of water<br>2: protected against: solid foreign objects with diameter starting from 12.5 mm and contact with a finger.<br>0: No protection |
| IP67 | 6: Dustproof, protection against contact with a wire<br>7: Protection against the effects of temporary submersion in water |
| ISO | International Standard Organization |
| LED | Light Emitting Diode |
| n. c. | not connected |
| OUT | Output |
| PELV | Protective Extra Low Voltage |
| SELV | Safety Extra Low Voltage |
| U | Voltage |
| U/I | Voltage / current |

## 8.3 Legal information

### Exclusion of liability

Murrelektronik GmbH checked the contents of this technical documentation for conformity with the described hardware and software. Changes on an individual case basis cannot be excluded. For this reason, we shall not assume any responsibility for errors or omissions, in particular for a complete conformity. This exclusion of liability shall not apply in case the damage is caused deliberately and/or due to gross negligence as well as for all claims based on the German Product Liability Act. Should a major contractual obligation negligently be violated, the liability of Murrelektronik GmbH shall be limited to damages that typically arise.

Subject to technical modifications or changes with regard to contents. We recommend to regularly check whether this documentation has been updated because corrections that might be required due to technical modifications will be included by Murrelektronik GmbH at regular intervals. We are always grateful for any proposals of improvement.

### Copyright



### Rights of use

Murrelektronik GmbH grants its customers a non-exclusive right revocable at any time and for an indefinite period of time to use this technical documentation to produce their own technical documentation. For this purpose, the documentation produced by Murrelektronik GmbH may be changed in parts, or amended, or copied, and transferred to the customer's users as part of the customer's own technical documentation on paper or on electronic media. In this case, the customers shall bear sole responsibility for the correctness of the contents of the technical documentation produced by them.

If the technical documentation is integrated in part, or in full in the customer's technical documentation, the customer shall refer to the copyright of Murrelektronik GmbH. Furthermore, special attention shall

be paid to compliance with the safety instructions.

Although the customers are obliged to make reference to the copyright of Murrelektronik GmbH, provided the technical documentation of Murrelektronik GmbH is used, the customers shall market and/or use the technical documentation on their sole responsibility. The reason is that Murrelektronik GmbH has no influence on changes or applications of the technical documentation and even minor changes to the starting product or deviations in the intended applications may render incorrect the specifications contained in the technical documentation. For this reason, the customers are obliged to identify the technical documentation originating from Murrelektronik GmbH if and inasmuch as the documentation is changed by the customers. The customers shall undertake to exempt Murrelektronik from the damage claims of third parties if the latter are attributable to any deficits in the documentation. This shall not apply to damages to the rights of third parties caused deliberately or by gross negligence.

The customers shall be entitled to use the company brands of Murrelektronik GmbH exclusively for their product advertising, but only inasmuch as the products of Murrelektronik GmbH are integrated in the products marketed by the customers. When using Murrelektronik GmbH brands, the customers shall mention this in an adequate manner.

Murrelektronik GmbH | Falkenstraße 3, D-71570 Oppenweiler
+49 7191 47-0 | +49 7191 47-491 000 | info@murrelektronik.com
www.murrelektronik.com