Handbücher/Manuals

art of automation

VIPA
Gesellschaft für Visualisierung
und Prozessautomatisierung mbH

Ohmstraße 4
D-91074 Herzogenaurach
Tel.: +49-9132-744-0
Fax: +49-9132-744-144
Internet: www.vipa.de
E-Mail: Info@vipa.de

Manual

VIPA System 200V

CP

Order No.: VIPA HB97E_CP
Rev. 11/30

The information contained in this manual is supplied without warranties. The information is subject to change without notice.

Ohmstraße 4, D-91074 Herzogenaurach,
Tel.: +49 (91 32) 744 -0
Fax.: +49 (91 32) 744-1864
EMail: info@vipa.de
http://www.vipa.de

Hotline: +49 (91 32) 744-1150

Disclaimer of liability

The contents of this manual were verified with respect to the hard- and software.

However, we assume no responsibility for any discrepancies or errors. The information in this manual is verified on a regular basis and any required corrections will be included in subsequent editions.

Suggestions for improvement are always welcome.

Trademarks

VIPA , System 100V, System 200V, System 300V and System 500V are registered trademarks of VIPA Gesellschaft für Visualisierung und Prozessautomatisierung mbH.

SIMATIC, STEP und S7-300 are registered trademarks of Siemens AG.

Any other trademarks referred to in the text are the trademarks of the respective owner and we acknowledge their registration.

About this manual

This manual describes the System 200V CP modules that are available from VIPA. In addition to the product summary it contains detailed descriptions of the different modules. You are provided with information on the connection and the utilization of the System 200V CP 240 modules. Every chapter is concluded with the technical data of the respective module.

Overview

Chapter 1: Basics

This introduction presents the VIPA System 200V as a centralized as well as decentralized automation system.

The chapter also contains general information about the System 200V, i.e. dimensions, installation and operating conditions.

Chapter 2: Assembly and installation guidelines

This chapter provides all the information required for the installation and the hook-up of a controller using the components of the System 200V.

Chapter 3: Project engineering

In this chapter you will find information about the basic approach with the project engineering of the CP 240. Besides the inclusion of GSD and block library into the Siemens SIMATIC manager it includes the description of all handling blocks that are necessary for the deployment of the CP 240.

Chapter 4: Communication processor CP 240 - serial

This chapter contains information on the construction, the interfacing and the communication protocols of the communication processor CP 240 with RS232 or RS485 interface respectively RS422/485 interface.

Chapter 5: Communication processor CP 240 - EnOcean

The deployment and the project engineering of the CP 240 EnOcean for radio transmission are described in this chapter.

Chapter 6: Communication processor CP 240 - M-Bus

Content of this chapter is the description of the CP 240 M-Bus. M-Bus (Metering Bus) is a standardized field bus for excise data capturing of energy and consumption counters like heating, water, current and gas counter.

Contents

User considerations

Objective and contents

This manual describes the modules that are suitable for use in the System 200V. It contains a description of the construction, project implementation and the technical data.

Target audience

The manual is targeted at users who have a background in automation technology.

Structure of the manual

The manual consists of chapters. Every chapter provides a self-contained description of a specific topic.

Guide to the document

The following guides are available in the manual:

- an overall table of contents at the beginning of the manual
- an overview of the topics for every chapter
- an index at the end of the manual.

Availability

The manual is available in:

- printed form, on paper
- in electronic form as PDF-file (Adobe Acrobat Reader)

Icons Headings

Important passages in the text are highlighted by following icons and headings:

Danger!
Immediate or likely danger.
Personal injury is possible.

Attention!
Damages to property is likely if these warnings are not heeded.

Note!
Supplementary information and useful tips.

Safety information

Applications conforming with specifications

The System 200V is constructed and produced for:

- all VIPA System 200V components
- communication and process control
- general control and automation applications
- industrial applications
- operation within the environmental conditions specified in the technical data
- installation into a cubicle

Danger!

This device is not certified for applications in

- in explosive environments (EX-zone)

Documentation

The manual must be available to all personnel in the

- project design department
- installation department
- commissioning
- operation

The following conditions must be met before using or commissioning the components described in this manual:

- Modification to the process control system should only be carried out when the system has been disconnected from power!
- Installation and modifications only by properly trained personnel
- The national rules and regulations of the respective country must be satisfied (installation, safety, EMC ...)

Disposal

National rules and regulations apply to the disposal of the unit!

Chapter 1 Basics

Overview

The focus of this chapter is on the introduction of the VIPA System 200V. Various options of configuring central and decentral systems are presented in a summary.

The chapter also contains the general specifications of the System 200V, i.e. dimensions, installation and environmental conditions.

Content

Safety information for Users

Handling of electrostatically sensitive modules

VIPA modules make use of highly integrated components in MOS-technology. These components are extremely sensitive to over-voltages that can occur during electrostatic discharges.

The following symbol is attached to modules that can be destroyed by electrostatic discharges:

The symbol is located on the module, the module rack or on packing material and it indicates the presence of electrostatic sensitive equipment.

It is possible that electrostatic sensitive equipment is destroyed by energies and voltages that are far less than the human threshold of perception. These voltages can occur where persons do not discharge themselves before handling electrostatically sensitive modules and they can damage components thereby, causing the module to become inoperable or unusable. Modules that have been damaged by electrostatic discharges may fail after a temperature change, mechanical shock or changes in the electrical load.

Only the consequent implementation of protection devices and meticulous attention to the applicable rules and regulations for handling the respective equipment can prevent failures of electrostatically sensitive modules.

Shipping of electrostatically sensitive modules

Modules have to be shipped in the original packing material.

Measurements and alterations on electrostatically sensitive modules

When you are conducting measurements on electrostatically sensitive modules you should take the following precautions:

- Floating instruments must be discharged before use.
- Instruments must be grounded.

Modifying electrostatically sensitive modules you should only use soldering irons with grounded tips.

Attention!

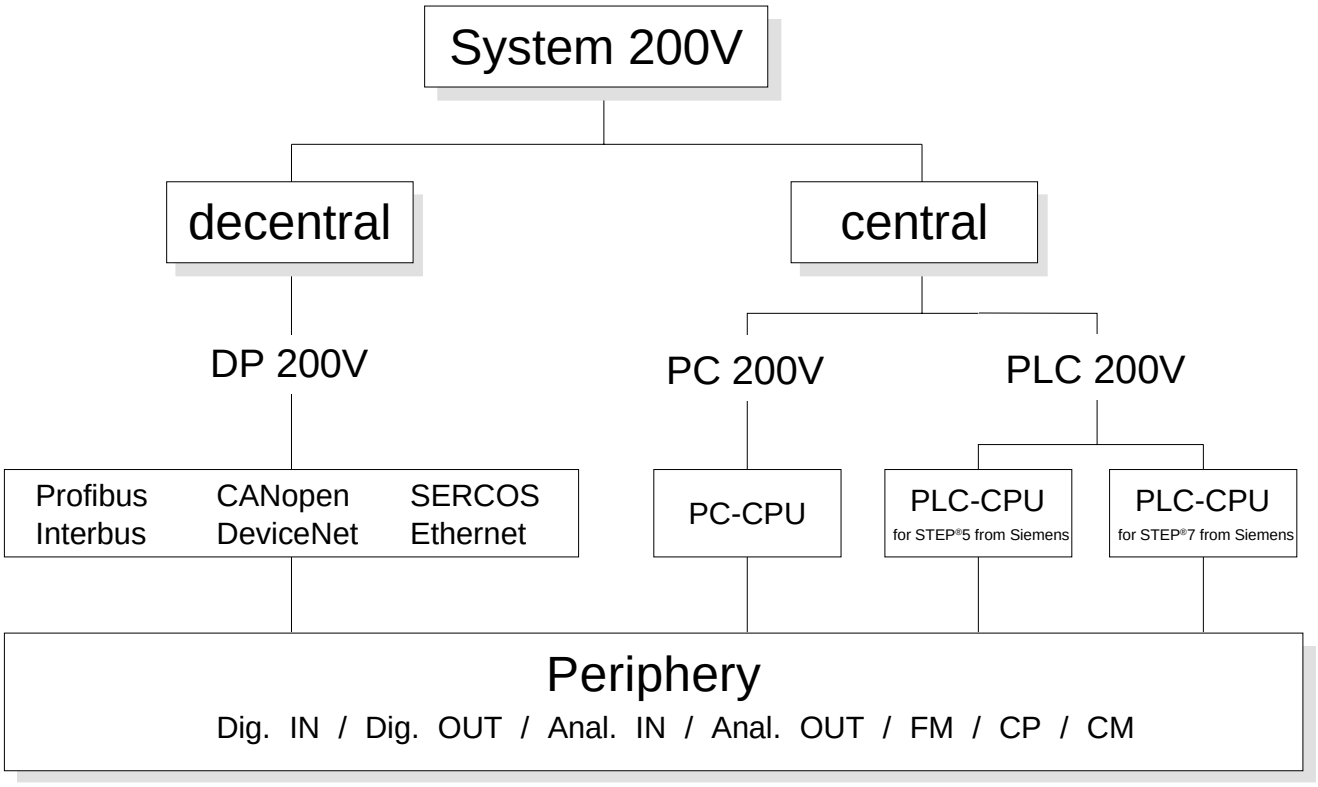
Personnel and instruments should be grounded when working on electrostatically sensitive modules.

Overview

The System 200V

The System 200V is a modular automation system for centralized and decentralized applications requiring low to medium performance specifications. The modules are installed directly on a 35mm mounting rail. Bus connectors inserted into the mounting rail provide the interconnecting bus.

The following figure illustrates the capabilities of the System 200V:



Components

Centralized system

The System 200V series consists of a number of PLC-CPUs. These are programmed in STEP®5 or STEP®7 from Siemens.

CPUs with integrated Ethernet interfaces or additional serial interfaces simplify the integration of the PLC into an existing network or the connection of additional peripheral equipment.

The application program is saved in Flash or an additional plug-in memory module.

The PC based CPU 288 can be used to implement operating/monitoring tasks, control applications or other file processing applications.

The modules are programmed in C++ or Pascal.

The PC 288-CPU provides an active interface to the backplane bus and can therefore be employed as central controller for all peripheral and function modules of the VIPA System 200V.

With the appropriate expansion interface the System 200V can support up to 4 rows.

Decentralized system

In combination with a Profibus DP master and slave the PLC-CPUs or the PC-CPU form the basis for a Profibus-DP network in accordance with DIN 19245-3. The DP network can be configured with WinNCS VIPA configuration tool res. Siemens SIMATIC Manager.

Other fieldbus systems may be connected by means of slaves for Interbus, CANopen, DeviceNet, SERCOS and Ethernet.

Peripheral modules

A large number of peripheral modules are available from VIPA, for example digital as well as analog inputs/outputs, counter functions, displacement sensors, positioners and serial communication modules.

These peripheral modules can be used in centralized as well as decentralized mode.

Integration over GSD File

The functionality of all VIPA system components are available via different GSD-files.

For the Profibus interface is software standardized, we are able to guarantee the full functionality by including a GSD-file using the Siemens SIMATIC Manager.

For every system family there is an own GSD-file. Actual GSD files can be found at ftp.vipa.de/support.

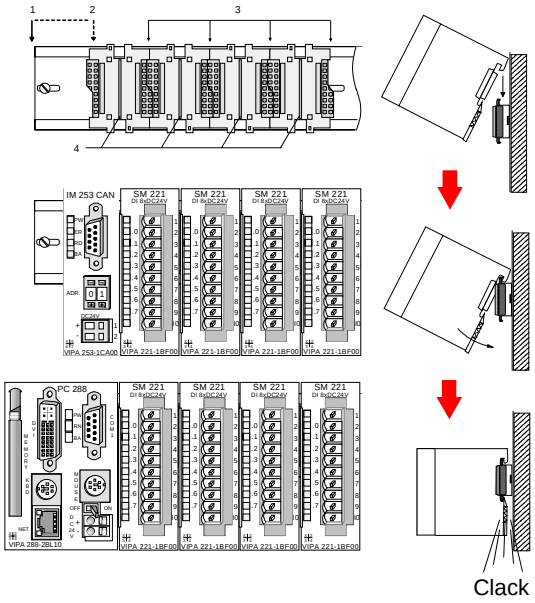
General description System 200V

Structure/ dimensions

- Mounting rail 35mm
- Peripheral modules with recessed labelling
- Dimensions of the basic enclosure:
  1tier width: (HxWxD) in mm: 76x25.4x74 in inches: 3x1x3
  2tier width: (HxWxD) in mm: 76x50.8x74 in inches: 3x2x3

Installation

Please note that you can only install header modules, like the CPU, the PC and couplers into plug-in location 1 or 1 and 2 (for double width modules).



[1] Header modules, like PC, CPU, bus couplers (double width)
[2] Header module (single width)
[3] Peripheral module
[4] Guide rails

Note

A maximum of 32 modules can be connected at the back plane bus. Take attention that here the **maximum sum current** of **3.5A** is not exeeded.

Please install modules with a high current consumption directly beside the header module.

Reliability

- Wiring by means of spring pressure connections (CageClamps) at the front-facing connector, core cross-section 0.08...2.5mm^2 or 1.5 mm^2 (18pole plug)
- Complete isolation of the wiring when modules are exchanged
- Every module is isolated from the backplane bus
- ESD/Burst acc. IEC 61000-4-2 / IEC 61000-4-4 (to level 3)
- Shock resistance acc. IEC 60068-2-6 / IEC 60068-2-27 (1G/12G)
- Class of protection IP20

Environmental conditions

- Operating temperature: 0 ... +60°C
- Storage temperature: -25 ... +70°C
- Relative humidity: 5 ... 95% without condensation
- Ventilation by means of a fan is not required

Chapter 2 Assembly and installation guidelines

Overview

This chapter contains the information required to assemble and wire a controller consisting of Systems 200V components.

Content

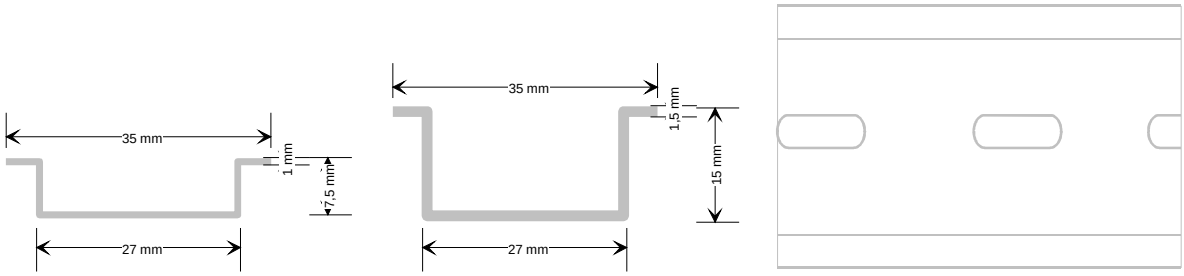
Overview

General

The modules are installed on a carrier rail. A bus connector provides interconnections between the modules. This bus connector links the modules via the backplane bus of the modules and it is placed into the mounting rail that carries the modules.

Mounting rail

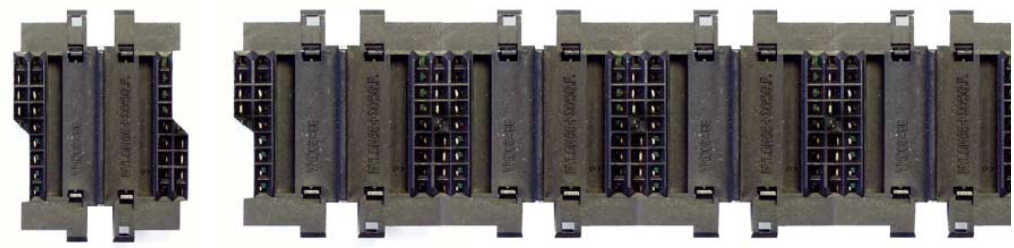
You may use the following standard 35mm mounting rails to mount the System 200V modules:



Bus connector

System 200V modules communicate via a backplane bus connector. The backplane bus connector is isolated and available from VIPA in of 1-, 2-, 4- or 8tier width.

The following figure shows a 1tier connector and a 4tier connector bus:

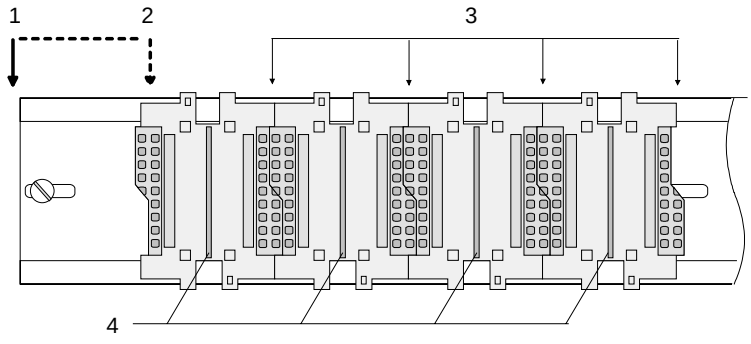


The bus connector is isolated and has to be inserted into the mounting rail until it clips in its place and the bus connections protrude from the rail.

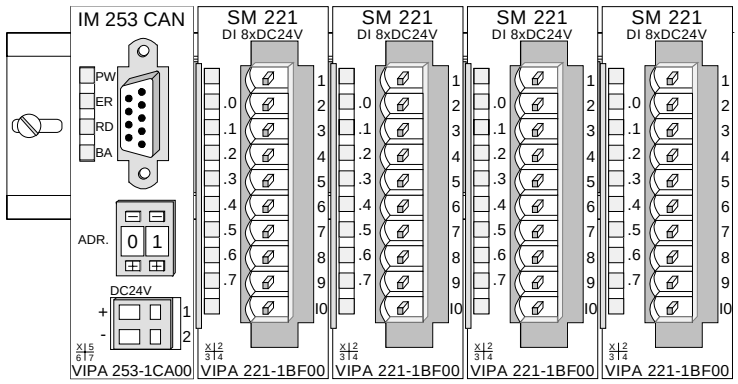
Mounting rail installation

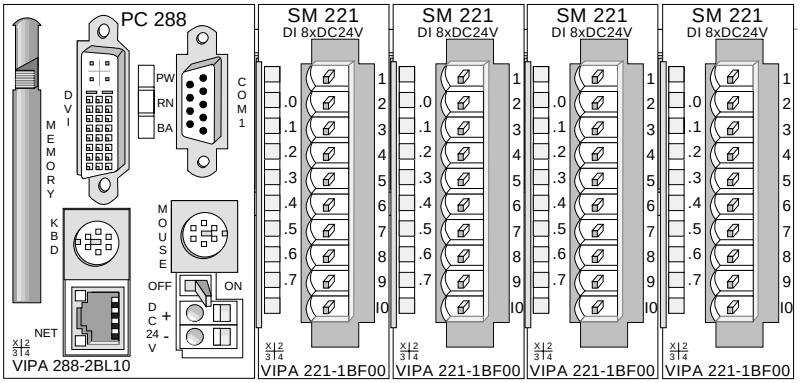
The following figure shows the installation of a 4tier width bus connector in a mounting rail and the plug-in locations for the modules.

The different plug-in locations are defined by guide rails.



[1] Header module, like PC, CPU, bus coupler, if double width
[2] Header module (single width)
[3] Peripheral module
[4] Guide rails





Note

A maximum of 32 modules can be connected at the back plane bus.

Take attention that here the **maximum sum current** of **3.5A** is not exceeded.

Assembly regarding the current consumption

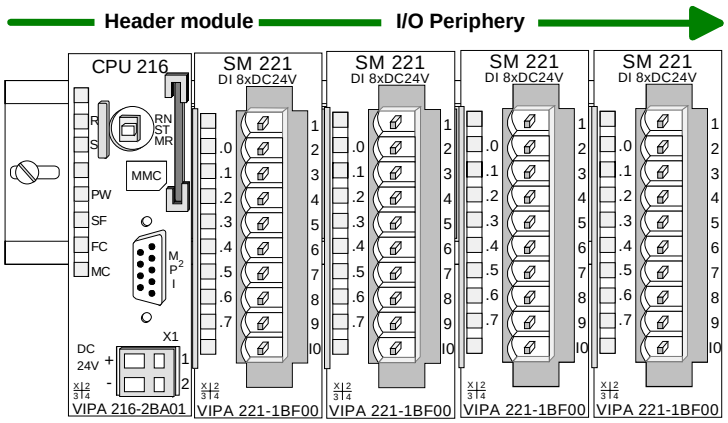
- Use bus connectors as long as possible.
- Sort the modules with a high current consumption right beside the header module. At ftp.vipa.de/manuals/system200v a list of current consumption of every System 200V module can be found.

Assembly horizontal respectively vertical

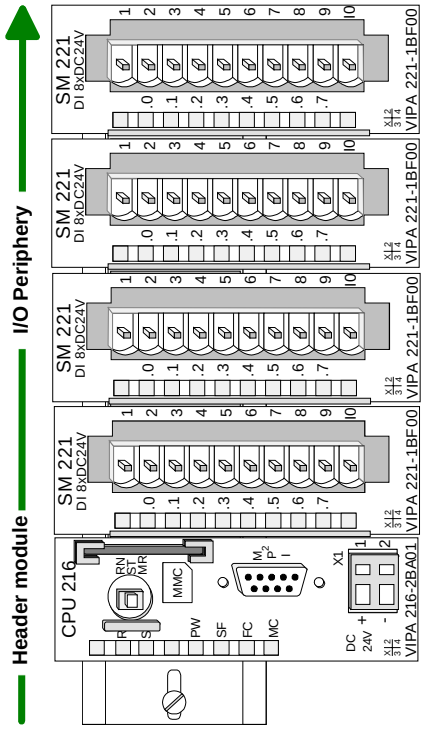
You may install the System 200V as well horizontal as vertical. Please regard the allowed environment temperatures:

- horizontal structure: from 0 to 60°
- vertical structure: from 0 to 40°

The horizontal structure always starts at the left side with a header module (CPU, bus coupler, PC), then you plug-in the peripheral modules beside to the right. You may plug-in maximum 32 peripheral modules.



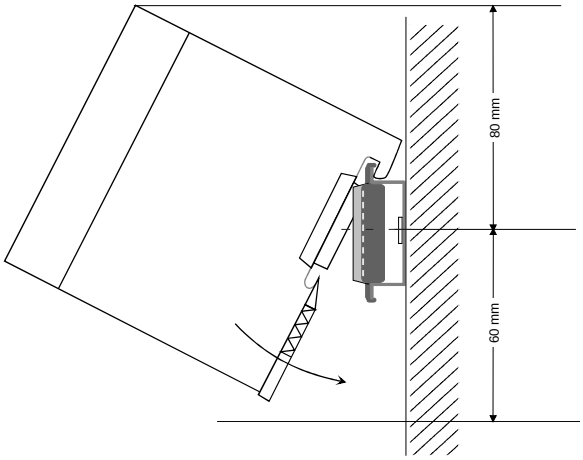
The vertical structure is turned for 90° against the clock.



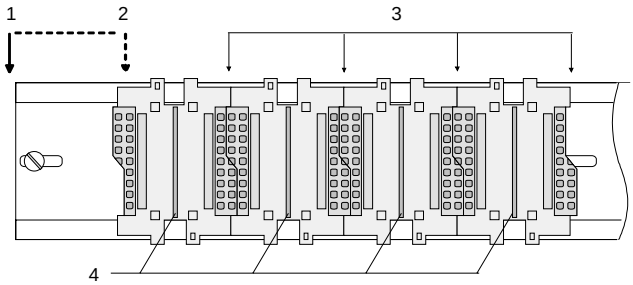
Assembly

Please follow these rules during the assembly!

- Turn off the power supply before you insert or remove any modules!
- Make sure that a clearance of at least 60mm exists above and 80mm below the middle of the bus rail.



- Every row must be completed from left to right and it has to start with a header module (PC, CPU, and bus coupler).



[1] Header module, like PC, CPU, bus coupler, if double width
[2] Header module (single width)
[3] Peripheral module
[4] Guide rails

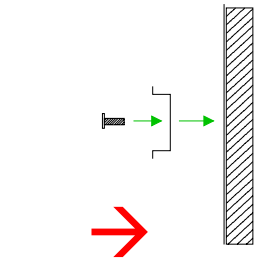
- Modules are to install adjacent to each other. Gaps are not permitted between the modules since this would interrupt the backplane bus.
- A module is only installed properly and connected electrically when it has clicked into place with an audible click.
- Plug-in locations after the last module may remain unoccupied.

Note!

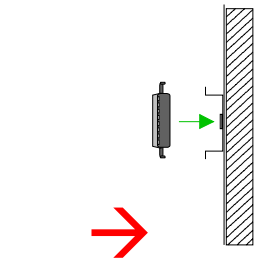
A maximum of 32 modules can be connected at the back plane bus. Take attention that here the maximum **sum current** of **3.5A** is not exceeded.

Assembly procedure

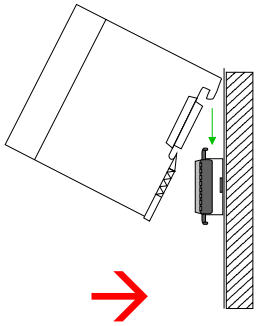
The following sequence represents the assembly procedure as viewed from the side.



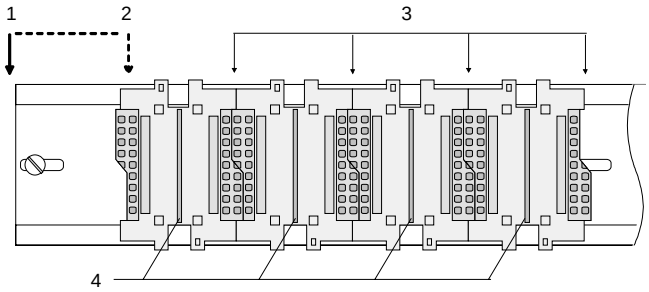
- Install the mounting rail. Make sure that a clearance of at least 60mm exists above and 80mm below the middle of the bus rail.



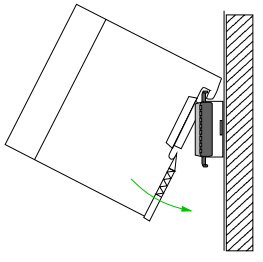
- Press the bus connector into the rail until it clips securely into place and the bus-connectors protrude from the mounting rail. This provides the basis for the installation of your modules.



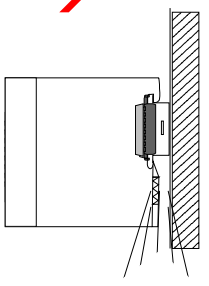
- Start at the outer left location with the installation of your header module like CPU, PC or bus coupler and install the peripheral modules to the right of this.



[1] Header module like PC, CPU, bus coupler
[2] Header module when this is a double width or a peripheral module
[3] Peripheral module
[4] Guide rails



- Insert the module that you are installing into the mounting rail at an angle of 45 degrees from the top and rotate the module into place until it clicks into the mounting rail with an audible click. The proper connection to the backplane bus can only be guaranteed when the module has properly clicked into place.

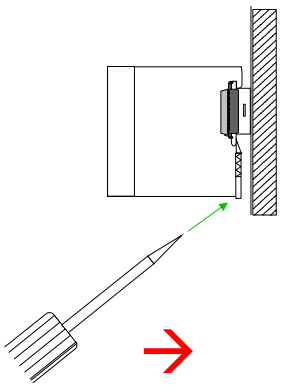


Attention!

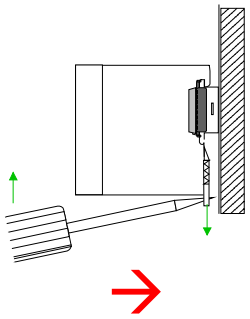
Power must be turned off before modules are installed or removed!

Removal procedure

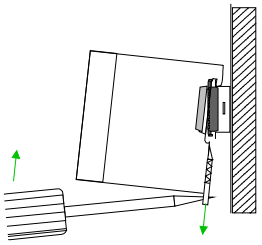
The following sequence shows the steps required for the removal of modules in a side view.



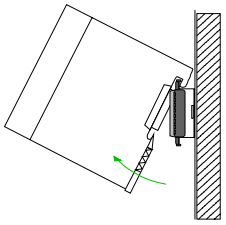
- The enclosure of the module has a spring-loaded clip at the bottom by which the module can be removed from the rail.
- Insert a screwdriver into the slot as shown.

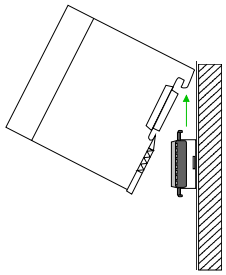


- The clip is unlocked by pressing the screwdriver in an upward direction.



- Withdraw the module with a slight rotation to the top.





Attention!

Power must be turned off before modules are installed or removed!

Please remember that the backplane bus is interrupted at the point where the module was removed!

Wiring

Outline

Most peripheral modules are equipped with a 10pole or an 18pole connector. This connector provides the electrical interface for the signaling and supply lines of the modules.

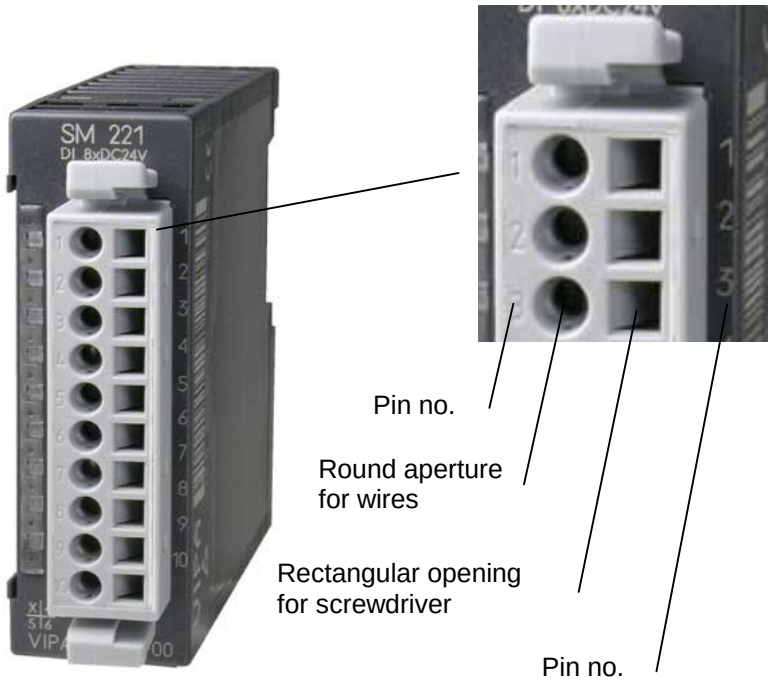
The modules carry spring-clip connectors for the interconnections and wiring.

The spring-clip connector technology simplifies the wiring requirements for signaling and power cables.

In contrast to screw terminal connections, spring-clip wiring is vibration proof. The assignment of the terminals is contained in the description of the respective modules.

You may connect conductors with a wire cross-section from 0.08mm^2 up to 2.5mm^2 (max. 1.5mm^2 for 18pole connectors).

The following figure shows a module with a 10pole connector.

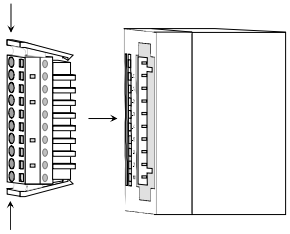


Note!

The spring-clip is destroyed if you insert the screwdriver into the opening for the hook-up wire!

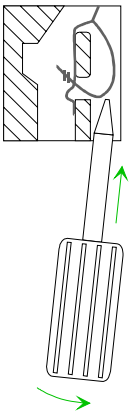
Make sure that you only insert the screwdriver into the square hole of the connector!

Wiring procedure

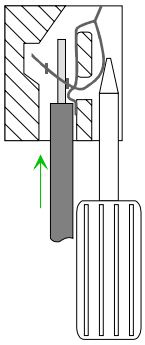


- Install the connector on the module until it locks with an audible click. For this purpose you press the two clips together as shown.
  The connector is now in a permanent position and can easily be wired.

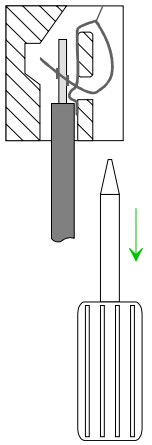
The following section shows the wiring procedure from above.



- Insert a screwdriver at an angel into the square opening as shown.
- Press and hold the screwdriver in the opposite direction to open the contact spring.



- Insert the stripped end of the hook-up wire into the round opening. You can use wires with a diameter of 0.08mm^2 to 2.5mm^2 (1.5mm^2 for 18pole connectors).



- When you remove the screwdriver, the wire is clipped securely.

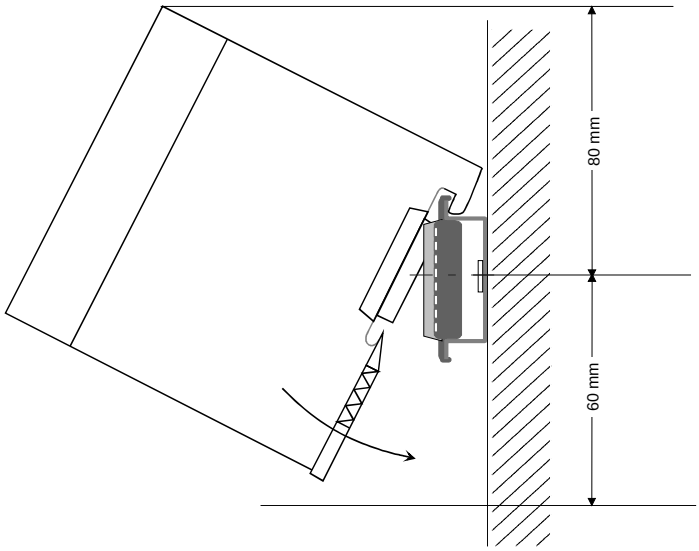
Wire the power supply connections first
followed by the signal cables (inputs and outputs).

Assembly dimensions

Overview Here follow all the important dimensions of the System 200V.

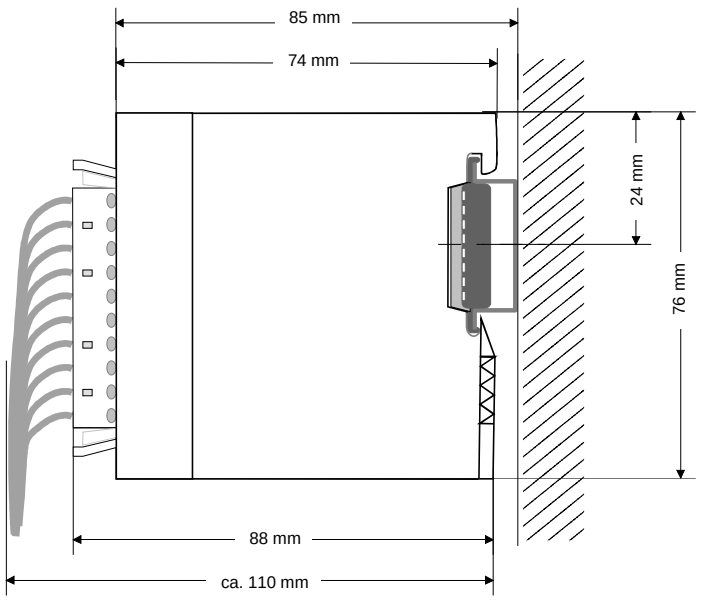
Dimensions Basic enclosure
1tier width (HxWxD) in mm: 76 x 25.4 x 74
2tier width (HxWxD) in mm: 76 x 50.8 x 74

Installation dimensions

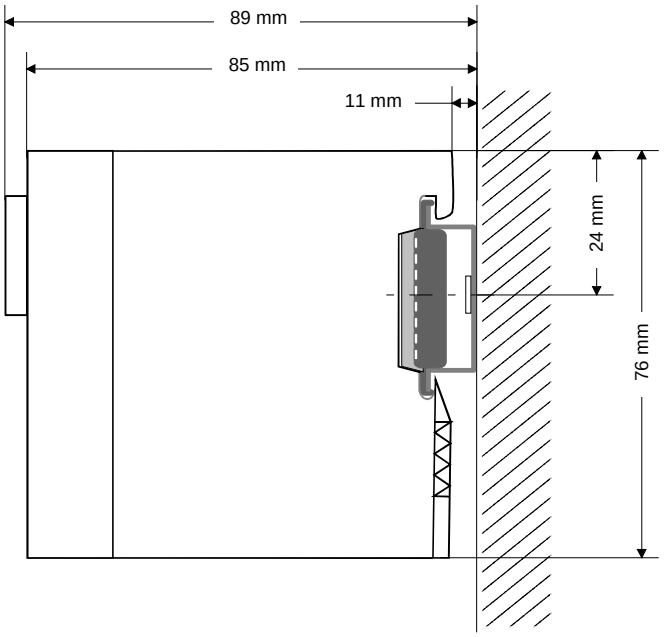


Installed and wired dimensions

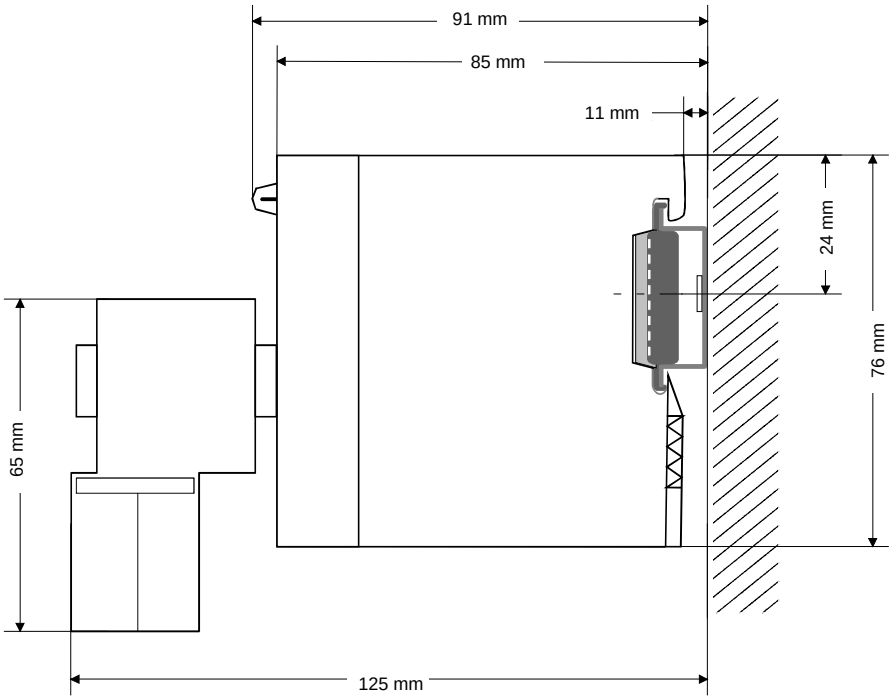
In- / Output modules



Function modules



CPUs here with EasyConn from VIPA



Installation guidelines

General

The installation guidelines contain information on the proper assembly of System 200V. Here we describe possible ways of interference that may disturb the controlling system and how you have to approach shielding and screening issues to ensure the electromagnetic compatibility (EMC).

What is EMC?

The term "electromagnetic compatibility" (EMC) refers to the ability of an electrical device to operate properly in an electromagnetic environment without interference from the environment or without the device causing illegal interference to the environment.

All System 200V components were developed for applications in harsh industrial environments and they comply with EMC requirements to a large degree. In spite of this you should implement an EMC strategy before installing any components which should include any possible source of interference.

Possible sources for disturbances

Electromagnetic interference can enter your system in many different ways:

- Fields
- I/O signal lines
- Bus system
- Power supply
- Protective conductor

Interference is coupled into your system in different ways, depending in the propagation medium (conducted or not) and the distance to the source of the interference.
We differentiate between:

- galvanic coupling
- capacitive coupling
- inductive coupling
- radiated power coupling

The most important rules for ensuring EMC

In many cases, adherence to a set of very elementary rules is sufficient to ensure EMC. For this reason we wish to advise you to heed the following rules when you are installing your controllers.

- During the installation of your components you have to ensure that any inactive metal components are grounded via a proper large-surface earth.
  - Install a central connection between the chassis ground and the earthing/protection system.
  - Interconnect any inactive metal components via low-impedance conductors with a large cross-sectional area.
  - Avoid aluminum components. Aluminum oxidizes easily and is therefore not suitable for grounding purposes.
- Ensure that wiring is routed properly during installation.
  - Divide the cabling into different types of cable. (Heavy current, power supply, signal and data lines).
  - Install heavy current lines and signal or data lines in separate channeling or cabling trusses.
  - Install signaling and data lines as close as possible to any metallic ground surfaces (e.g. frames, metal rails, sheet metal).
- Ensure that the screening of lines is grounded properly.
  - Data lines must be screened.
  - Analog lines must be screened. Where low-amplitude signals are transferred, it may be advisable to connect the screen on one side of the cable only.
  - Attach the screening of cables to the ground rail by means of large surface connectors located as close as possible to the point of entry. Clamp cables mechanically by means of cable clamps.
  - Ensure that the ground rail has a low-impedance connection to the cabinet/cubicle.
  - Use only metallic or metallized covers for the plugs of screened data lines.
- In critical cases you should implement special EMC measures.
  - Connect snubber networks to all inductive loads that are controlled by System 200V modules.
  - Use incandescent lamps for illumination purposes inside cabinets or cubicles, do not use fluorescent lamps.
- Create a single reference potential and ensure that all electrical equipment is grounded wherever possible.
  - Ensure that earthing measures are implemented effectively. The controllers are earthed to provide protection and for functional reasons.
  - Provide a star-shaped connection between the plant, cabinets/cubicles of the System 200V and the earthing/protection system. In this way you avoid ground loops.
  - Where potential differences exist you must install sufficiently large equipotential bonding conductors between the different parts of the plant.

Screening of cables

The screening of cables reduces the influence of electrical, magnetic or electromagnetic fields; we talk of attenuation.

The earthing rail that is connected conductively to the cabinet diverts interfering currents from screen conductors to ground. It is essential that the connection to the protective conductor is of low-impedance as the interfering currents could otherwise become a source of trouble in themselves.

The following should be noted when cables are screened:

- Use cables with braided screens wherever possible.
- The coverage of the screen should exceed 80%.
- Screens should always be grounded at both ends of cables. High frequency interference can only be suppressed by grounding cables on both ends.

  Grounding at one end may become necessary under exceptional circumstances. However, this only provides attenuation to low frequency interference. One-sided earthing may be of advantage where:
  - it is not possible to install equipotential bonding conductors.
  - analog signals (in the mV or µA range) are transferred.
  - foil-type shields (static shields) are used.
- Always use metallic or metallized covers for the plugs on data lines for serial links. Connect the screen of the data line to the cover. Do **not** connect the screen to PIN 1 of the plug!
- In a stationary environment it is recommended that the insulation is stripped from the screened cable interruption-free and to attach the screen to the screening/protective ground rail.
- Connect screening braids by means of metallic cable clamps. These clamps need a good electrical and large surface contact with the screen.
- Attach the screen of a cable to the grounding rail directly where the cable enters the cabinet/cubicle. Continue the screen right up to the System 200V module but do **not** connect the screen to ground at this point!

Please heed the following when you assemble the system!

Where potential differences exist between earthing connections it is possible that an equalizing current could be established where the screen of a cable is connected at both ends.

Remedy: install equipotential bonding conductors

Chapter 3 Project engineering

Overview

This chapter deals with the project engineering and the programming of a CP 240 in general. Detailed information about the project engineering of a special CP 240 is to be found as sample project in the according chapter.

After a fast introduction you will get information about how to include GSD files and block libraries into the Siemens SIMATIC Manager.

The chapter closes with a description of the standard handling blocks for the CP communication.

Content

Fast introduction

Overview

The address allocation and he parameterization of the CP 240 happens by means of the Siemens SIMATIC Manager in form of a virtual Profibus system. For this the inclusion of the VIPA_21x.gsd (V. 1.67 or higher) is required.

For the communication between your CPU and the CP 240 there are handling blocks available, collected in form a library that you may include into your Siemens SIMATIC Manager.

Approach

Preparation

- Start the Siemens SIMATIC Manager with a new project.
- Include the VIPA_21x.gsd. For this, use a GSD version V. 1.67 or higher.
- Include the block library by extracting *FX000011_Vxxx.zip* and de-archiving VIPA.ZIP.
- Open the library and transfer the corresponding FCs into your project.

Hardware configuration

Please follow for the hardware configuration the steps described in the manual HB97 - CPU:

- Configure a Profibus-DP master system with the Siemens CPU 315-2DP (6ES7 315-2AF03 V1.2) and create a Profibus subnet.
- Add to the master system the slave system "VIPA_CPU21x" from the hardware catalog. This is listed in the hardware catalog under *Profibus-DP > Additional field devices > I/O > VIPA_System_200V.*
- Assign the address 1 to the slave system. With this, the VIPA CPU identifies the system as central periphery system.
- Within this slave system, you place your modules in the plugged sequence. Start with the CPU at the first plug-in location.
- Then include your System 200V modules and at the correct place the CP 240.
- If necessary parameterize your CP 240.

PLC program

For the communication a PLC application is required. For this you may use the following handling blocks:

FC 0	SEND	Data output CPU to CP 240
FC 1	RECEIVE	Receive data from CP 240
FC 8	STEUERBIT	Access at serial modem lines
FC 9	SYNCHRON_RESET	Synchronization between CPU and CP 240
FC 11	ASCII_FRAGMENT	Fragmented ASCII data receipt
FC 2	FETCH_RK512	RK512 - FETCH data from remote station
FC 3	SEND RK512	RK512 - SEND data to remote station
FC 4	S/R_ALL_RK512	RK512 - SEND/RECEIVE for passive station

Include GSD and FCs

Project engineering via GSD

The address allocation and he parameterization of the CP 240 happens by means of the Siemens SIMATIC Manager in form of a virtual Profibus system. Since the Profibus interface is software standardized, the inclusion of a GSD file enables the guaranteed functionality of running in the SIMATIC Manager from Siemens at any time. Transfer your project via MPI into CPU.

Include GSD

The following steps are required for the installation of the GSD:

- Copy the delivered VIPA GSD-file **`VIPA_21X.GSD`** (V. 1.31 or higher) into your GSD directory ... \siemens\step7\s7data\gsd
- Start the hardware configurator from Siemens
- Close all projects
- Select **Options** > *Install new GSD-file*
- Set here **VIPA_21X.gsd**

Now the modules of the System 200V from VIPA are integrated into the hardware catalog and may be used.

Installing blocks

The VIPA specific blocks may be found at www.vipa.de as downloadable library at the service area with *Downloads > VIPA LIB*. The library is available as packed zip-file.

If you want to use VIPA specific blocks, you have to import the library into your project.

Retrieve library

Start your un-zip application with a double click on the file FX000011_Vxxx.zip and copy the file vipa.zip to your work directory. It is not necessary to extract this file, too.

To retrieve your library for the SPEED7-CPUs, start the SIMATIC manager from Siemens. Open the dialog window for archive selection via **File** > *Retrieve*. Navigate to your work directory.

Choose VIPA.ZIP and click at [Open].

Select a destination folder where the blocks are to be stored. [OK] starts the extraction.

Open library and transfer blocks to project

After the extraction open the library.

Open your project and copy the necessary blocks from the library into the directory "blocks" of your project.

Now you have access to the VIPA specific blocks via your user application.

Project engineering

General

The address allocation and he parameterization of the directly plugged System 200V modules happens by means of the Siemens SIMATIC Manager in form of a virtual Profibus system. You transfer your project into the CPU serial via the MPI interface or directly via MMC.

Requirements

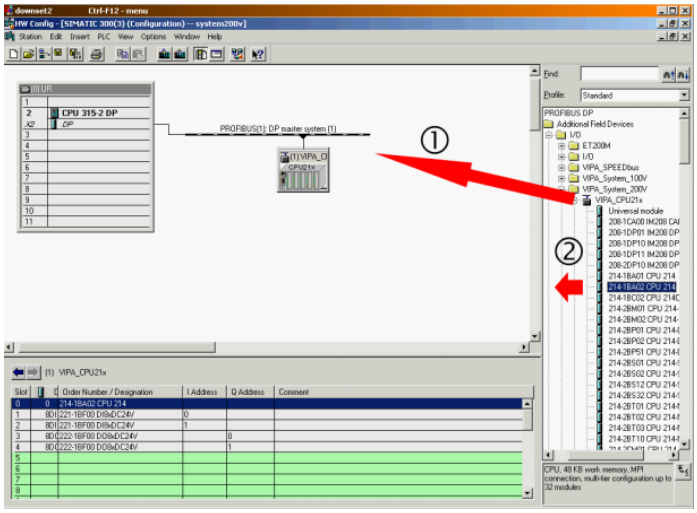
For the project engineering of the CPU a thorough knowledge of the SIMATIC Manager and the hardware configurator from Siemens is required!

For the project engineering the following preconditions must be fulfilled:

- SIMATIC Manager from Siemens is installed at PC res. PG
- GSD files are included into hardware configurator from Siemens
- The project can be transferred into CPU (serial e.g. "Green Cable" or MMC)

Hardware configuration

- Start the hardware configurator from Siemens with a new project and insert a profile rail from the hardware catalog.
- At the first available slot you place the CPU 315-2DP (6ES7 315-2AF03 V1.2) from Siemens.
- If your CPU 21x has an integrated Profibus-DP master, you may now connect it to Profibus and include your DP slaves.
- Create a Profibus subnet (if not present yet).
- Add the system "VIPA_CPU21x" to the subnet. You will find this in the hardware catalog under *PROFIBUS DP* > *Additional field devices* > *IO* > VIPA_System_200V. Assign the **Profibus address 1** to this slave.
- In your configurator, place the CPU 21x, which you are using, **always on the 1. slot** by taking it from the hardware catalog.
- Then you include your System 200V modules in the plugged sequence and your CP 240 at the according place.
- If necessary parameterize your CP 240.
- Save your project.



PLC program

For the communication between CPU and CP 240 shown in the text below, the following handling blocks are used:

FC 0	SEND	Data output CPU to CP 240
FC 1	RECEIVE	Receive data from CP 240
FC 9	SYNCHRON_RESET	Synchronization between CPU and CP 240

The handling blocks are available as library and may be integrated into the Siemens SIMATIC Manager like shown above.

A more detailed description of the handling blocks is to be found on the following pages.

Your PLC program should be build-up with the following structure:

```
OB1:

      CALL  FC     9             //Call Synchron
       ADR       :=0             //1st DW in SEND/EMPF_DB
       TIMER_NR  :=T2            //Delay time Synchron
       ANL       :=M3.0          //Start-up running
       NULL      :=M3.1          //Interim flag
       RESET     :=M3.2          //Execute module reset
       STEUERB_S :=MB2           //Control bits Send_FC
       STEUERB_R :=MB1           //Control bits Receive_FC
      U     M      3.0           //as long as no start-up no
                                 //SEND/RECEIVE processing
      BEB

      CALL  FC     1             //Receive data
       ADR          :=0          //1st DW in SEND/RECEIVE_DB
       _DB          :=DB11       //Receive_DB telegram
       ABD          :=W#16#14    //1st DW receive buffer (DW20)
       ANZ          :=MW10       //Amount of received data
       EMFR         :=M1.0       //Reception ready
       PAFE         :=MB12       //Error byte
       GEEM         :=MW100      //Internal data
       ANZ_INT      :=MW102      //Internal data
       empf_laeuft  :=M1.1       //Internal data
       letzter_block:=M1.2       //Internal data
       fehl_empf    :=M1.3       //Internal data
      U     M      1.0           //Reception ready
      R     M      1.0           //delete reception ready
      CALL  FC     0             //Send data
       ADR          :=0          //1st DW in SEND/RECEIVE_DB
       _DB          :=DB10       //Send_DB telegram
       ABD          :=W#16#14    //1st DW send buffer (DW20)
       ANZ          :=MW14       //Amount of data to send
       FRG          :=M2.0       //Set send ready
       PAFE         :=MB16       //Error byte
       GESE         :=MW104      //Internal data
       ANZ_INT      :=MW106      //Internal data
       ende_kom     :=M2.1       //Internal data
       letzter_block:=M2.2       //Internal data
       senden_laeuft:=M2.3       //Internal data
       fehler_kom   :=M2.4       //Internal data


OB100:

      UN    M      3.0
      S     M      3.0           //Start-up CPU running
```

Transfer project

The data transfer happens via MPI. If your programming device is not provided with a MPI interface you may also use a serial point-to-point transfer from your PC to MPI with the help of the "Green Cable" from VIPA.

The "Green Cable" has the order no. VIPA 950-0KB00 and may only be used with the VIPA CPUs with MP^{2}I interface.

Please regard for this also the hints for the usage of the Green Cable in the basics!

- Connect your PG with the CPU.
- Via **PLC** > *Load to module* in your project engineering tools you transfer the project into the CPU.
- Plug-in a MMC and transfer your user application to the MMC by means of **PLC** > *Copy RAM to ROM*.
- During the write process the "MC"-LED at the CPU is blinking. Due to system reasons a successful write process is announced too early. Please wait until the LED extinguishes.

What is the Green Cable?

The Green Cable is a green connection cable made exclusively for the deployment at VIPA System components.

The Green Cable allows you to:

- transfer project serially from point-to-point
- execute firmware updates of the CPUs and field bus master

Important hints for the deployment of the Green Cable

Non-observance of the following hints may cause damages to the system components.

For damages caused by non-observance of these hints and at incorrect usage, VIPA does not assume liability!

Hints for the operating range

The Green Cable may exclusively be deployed directly at the supposed jacks of the VIPA components (adapter plugs are not permissible). For example you have to pull a plugged MPI cable before connecting a Green Cable.

At this moment the following components supports the Green Cable:

VIPA CPUs with MP^{2}I jack as well as the field bus master from VIPA.

Notes to the lengthening

The lengthening of the Green Cable with another Green Cable res. the combination with other MPI cables is not permissible and causes damages to the connected components!

The Green Cable may only be lengthened with a 1:1 cable (all 9 pins are connected 1:1).

Standard handling blocks for CPU 21x

SEND (FC 0)

This FC serves the data output from the CPU to the CP 240. Here you define the send range via the identifiers _DB, ADB and ANZ.

Via the bit FRG the send initialization is set and the data is send. After the data transfer the handling block sets the bit FRG back again.

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	_DB	BLOCK_DB	DB No. of DB containing data to send
in	ABD	WORD	No. of 1. data word to send
in	ANZ	WORD	No of bytes to send
in_out	FRG	BOOL	Start bit of the function
in_out	GESE	WORD	internal use
in_out	ANZ_INT	WORD	internal use
in_out	ENDE_KOMM	BOOL	internal use
in_out	LETZTER_BLOCK	BOOL	internal use
in_out	SENDEN_LAEUFT	BOOL	Status of function
in_out	FEHLER_KOM	BOOL	internal use
out	PAFE	BYTE	Return Code (00=OK)

ADR

Periphery address with which you may call the CP 240. Via the hardware configuration you may set the periphery address.

_DB

Number of the data block, which contains the data to send.

ABD

Word variable that contains the number of the data word from where on the characters for output are stored.

ANZ

Number of the bytes that are to be transferred.

FRG enable send

At FRG = "1" the data defined via _DB, ADB and ANZ are transferred once to the CP addresses by ADR. After the transmission the FRG is set back again. When FRG = "0" at call of the block, it is left immediately!

PAFE

At proper function, all bits of this bit memory byte are "0". At errors an error code is entered. The error setting is self-acknowledging, i.e. after elimination of the error cause, the byte is set back to "0" again. The following errors may occur:

1 = Data block not present

2 = Data block too short

3 = Data block number outside valid range

GESE, ANZ_INT
ENDE_KOM
LETZTER_BLOCK
SENDEN_LAEUFT
FEHLER_KOM

These parameters are internally used. They serve the information exchange between the handling blocks. For the deployment of the SYNCHRON_RESET (FC9) the control bits ENDE_KOM, LETZTER _BLOCK, SENDEN_LAEUFT and FEHLER_KOM must always be stored in a bit memory byte.

RECEIVE (FC 1)

This FC serves the data reception of the CP 240. Here you set the reception range via the identifiers _DB and ADB.

When the output EMFR is set, a new telegram has been read completely. The length of the telegram is stored in ANZ. After the evaluation of the telegram this bit has to be set back by the user, otherwise no further telegram may be taken over by the CPU.

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	_DB	BLOCK_DB	DB No. of DB containing received data
in	ABD	WORD	No. of 1. data word received
out	ANZ	WORD	No of bytes received
out	EMFR	BOOL	1=data received, reset by user
in_out	GEEM	WORD	internal use
in_out	ANZ_INT	WORD	internal use
in_out	EMPF_LAEUFT	BOOL	Status of function
in_out	LETZTER_BLOCK	BOOL	internal use
in_out	FEHLER_EMPF	BOOL	internal use
out	PAFE	BYTE	Return Code (00=OK)

ADR Periphery address for calling the CP 240. You define the periphery address via the hardware configuration.

_DB Number of the data block, which contains the data.

ABD Word variable that contains the number of the data word from where on the received characters are stored.

ANZ Word variable that contains the amount of received bytes.

EMFR By setting of EMFR the handling block shows that data has been received. Not until setting back EMFR in the user application new data can be received.

PAFE At proper function, all bits of this bit memory byte are "0". At errors an error code is entered. The error setting is self-acknowledging, i.e. after elimination of the error cause, the byte is set back to "0" again. The following errors may occur:

1 = Data block not present

2 = Data block too short

3 = Data block number outside valid range

GEEM, ANZ_INT
LETZTER_BLOCK
EMPF_LAEUFT
FEHLER_EMPF

These parameters are internally used. They serve the information exchange between the handling blocks. For the deployment of the SYNCHRON_RESET (FC9) the control bits LETZTER_BLOCK, EMPF_LAEUFT and FEHLER_EMPF must always be stored in a bit memory byte.

STEUERBIT (FC 8) This block allows you the following access to the serial modem lines:

Read: DTR, RTS, DSR, RI, CTS, CD

Write: DTR, RTS

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	RTS	BOOL	New state RTS
in	DTR	BOOL	New state DTR
in	MASKE_RTS	BOOL	0: do nothing 1: set state RTS
in	MASKE_DTR	BOOL	0: do nothing 1: set state DTR
out	STATUS	BYTE	Status flags
out	DELTA_STATUS	BYTE	Status flags of change between 2 accesses
in_out	START	BOOL	Start bit of the function
in_out	AUFTRAG_LAEU	BOOL	Status of function
out	RET_VAL	WORD	Return Code (00=OK)

Note!

This block must not be called as long as a transmit command is running otherwise you risk a data loss.

ADR Periphery address with which you may call the CP 240. Via the hardware configuration you may set the periphery address.

RTS, DTR This parameter presets the status of RTS res. DTR, which you may activate via MASK_RTS res. MASK_DTR.

MASK_RTS, MASK_DTR With 1, the status of the according parameter is taken over when you set START to 1.

STATUS, DELTA_STATUS STATUS returns the actual status of the modem lines. DELTA_STATUS returns the state of the modem lines that have changed since the last access.

The bytes have the following structure:

Bit no.	7	6	5	4	3	2	1	0
STATUS	x	x	RTS	DTR	CD	RI	DSR	CTS
DELTA_STATUS	x	x	x	x	CD	RI	DSR	CTS

START By setting of START, the state, which has been activated via the mask, is taken over.

AUFTRAG_LAEU As long as the function is executed, this bit remains set.

RET_VAL At this time, this parameter always returns 00h and is reserved for future error messages.

SYNCHRON_ RESET
Synchronization and reset (FC 9)

The block must be called within the cyclic program section. This function is used to acknowledge the start-up ID of the CP 240 and thus the synchronization between CPU and CP. Furthermore it allows to set back the CP in case of a communication interruption to enable a synchronous start-up.

Note!

A communication with SEND and RECEIVE blocks is only possible when the parameter ANL of the SYNCHRON block has been set in the start-up OB before.

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	TIMER_NR	WORD	No of timer for idle time
in_out	ANL	BOOL	restart progressed
in_out	NULL	BOOL	internal use
in_out	RESET	BOOL	1 = Reset the CP
in_out	STEUERB_S	BYTE	internal use
in_out	STEUERB_R	BYTE	internal use

ADR Periphery address with which you may call the CP 240. Via the hardware configuration you may set the periphery address.

TIMER_NR Number of the timer for the delay time.

ANL With ANL = 1 the handling block is informed that a STOP/START res. NETZ-AUS/NETZ-EIN has been executed at the CPU and now a synchronization is required. After the synchronization, ANL is automatically set back.

NULL Parameter is used internally.

RESET RESET = 1 allows you to set back the CP out of your user application.

STEUERB_S Here you have to set the bit memory byte where the control bits ENDE_KOM, LETZTER_BLOCK, SENDEN_LAEUFT and FEHLER_KOM for the SEND-FC are stored.

STEUERB_R Here you have to set the bit memory byte where the control bits LETZTER_BLOCK, EMPF_LAEUFT and FEHLER_EMPF for the RECEIVE-FC are stored.

ASCII_FRAGMENT (FC 11)

This FC serves the fragmented ASCII data reception. This allows you to handle on large telegrams in 12Byte blocks to the CPU directly after the reception. Here the CP does not wait until the complete telegram has been received. The usage of the FC 11 presumes that you've parameterized "ASCII-fragmented" at the receiver.

In the FC 11, you define the reception range via the identifiers _DB and ADB. When the output EMFR is set, a new telegram has been read completely. The length of the read telegram is stored in ANZ. After the evaluation of the telegram this bit has to be set back by the user, otherwise no further telegram may be taken over by the CPU.

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	_DB	BLOCK_DB	DB No. of DB containing received data
in	ABD	WORD	No. of 1. data word received
out	ANZ	WORD	No of bytes received
in_out	EMFR	BOOL	1=data received, reset by user
in_out	GEEM	WORD	internal use
in_out	ANZ_INT	WORD	internal use
in_out	EMPF_LAEUFT	BOOL	internal use
in_out	LETZTER_BLOCK	BOOL	internal use
in_out	FEHLER_EMPF	BOOL	internal use
out	PAFE	BYTE	Return Code (00=OK)

ADR Periphery address with which you may call the CP 240. Via the hardware configuration you may set the periphery address.

_DB Number of the data block, which contains the data to receive.

ABD Word variable that contains the number of the data word from where on the received characters are stored.

ANZ Word variable that contains the amount of bytes that have been received.

EMFR By setting of EMFR, the handling block announces that data has been received. Only by setting back EMFR in the user application new data can be received.

PAFE At proper function, all bits of this bit memory byte are "0". At errors an error code is entered. The error setting is self-acknowledging, i.e. after elimination of the error cause, the byte is set back to "0" again. The following errors may occur:

1 = Data block not present

2 = Data block too short

3 = Data block number outside valid range

GEEM, ANZ_INT LETZTER_BLOCK EMPF_LAEUFT FEHLER_EMPF These parameters are internally used. They serve the information exchange between the handling blocks. For the deployment of the SYNCHRON_RESET (FC9) the control bits LETZTER_BLOCK, EMPF_LAEUFT and FEHLER_EMPF must always be stored in a bit memory byte.

RK512 communication - Handling blocks

FETCH_RK512 (FC 2)

This FC serves for an active access to a partner station by means of RK512, which makes passive data available. Here a telegram with source data is sent to the partner station. The partner station collects the data and sends them back to your station.

The received data are stored in the target DB.

Here the source range in the partner station is defined by QDB, QBDW and LANG. The target area in your station is defined by ZDB and ZBDW.

With calling the FC it is checked by means of the check bits if there is an order just running. If the control bits are reset a new FETCH order is released.

Here a frame header is transmitted to the CP after that the system waits for the response message with user data.

The indicator word indicates "Order just running" as long as the message with user data was not receipt. Only after the response message was received by the CP and the user data were transmitted to the PLC, the flag "Order ready" of the indicator word is set and the communication to the CP is finished.

This function is cyclically be called as long as "Order ready with/without error" is set at the indicator word .

With an error during communication the CPU gets an error number from the CP. Then the error number is transferred to the indicator word and the bit "Order ready with error" is set. Then the communication to the CP is finished.

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	QDB	BLOCK_DB	DB No. of DB of the remote station
in	QBDW	WORD	No. of 1. DW of the DB of remote station
in	LANG	INT	Length of data to transfer
in	ZDB	BLOCK_DB	Number target DB of this station
in	ZBDW	INT	No. of 1. data word in target DB
in	KOOR	WORD	Coordination flag
out	ANZW	WORD	Indicator word
out	PAFE	BYTE	Parameterization error byte Return Code (00h=OK)
in_out	ANZ	WORD	internal use
in_out	GESE	WORD	internal use
in_out	KOPF_GESE	BOOL	internal use
in_out	WART_DATEN	BOOL	internal use
in_out	EMPF_LAEUFT	BOOL	internal use
in_out	LETZTER_BLOCK	BOOL	internal use
in_out	FEHL_KOM	BOOL	internal use

ADR	Periphery address with which you may call the CP 240. Via the hardware configuration you may set the periphery address.
QDB	Number of the source data block of the remote station.
QBDW	1. data word of the data block of the remote station.
LANG	Length of the data to send in words.
ZDB	Number of the target block of the own station.
ZBDW	1. data word of the data block of the own station.
KOOR	Coordination flag The coordination flag serves to coordinate the receipt of data. The coordination flag is set by the FETCH order. As long as the flag is set, no other FETCH order may be released. If you want to prevent that data are overwritten after receipt by new data, the deployment of the coordination flag may be useful. With FFFFh the coordination flag is deactivated.
ANZW	Indicator word Information concerning the order commissioning may be accessed by the indicator word. More may be found at "RK512 communication - Indicator word ANZW".
PAFE	At proper function, all bits of this bit memory byte are "0". At errors an error code is entered. The error setting is self-acknowledging, i.e. after elimination of the error cause, the byte is set back to "0" again. The following errors may occur: 1 = Data block not present 2 = Data block too short 3 = Data block number outside valid range
ANZ, GESE, KOPF_GESE, WART_DATEN, EMPF_LAEUFT, LETZTER_BLOCK, FEHL_KOM	These parameters are internally used. They serve the information exchange between the handling blocks.

SEND_RK512 (FC 3)

This FC serves for data transfer from the CPU to a partner station. The target at the partner station is transferred together with the user data Here the source area of the own station is defined by QDB, QBDW and LANG. The target area in the remote station is defined by ZDB and ZBDW.

With calling the FC it is checked by means of the check bits if there is an order just running. If the control bits are reset a new SEND order is released.

Here a frame header with user data is transmitted to the CP after that the system waits for the response message.

The indicator word indicates "Order just running" as long as the response message was not received. Only after the receipt of the response message the flag "Order ready" of the indicator word is set and the communication to the CP is finished.

This function is cyclically called as long as "Order ready with/without error" is set at the indicator word.

With an error during communication the CPU gets an error number from the CP. Then the error number is transferred to the indicator word and the bit "Order ready with error" is set. Then the communication to the CP is finished.

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	QDB	BLOCK_DB	DB No. of DB of this station
in	QBDW	WORD	No. of 1. DW of the DB of this station
in	LANG	INT	Length of data to send
in	ZDB	BLOCK_DB	DB No. of DB of the remote station
in	ZBDW	INT	No of 1. DW of the DB of remote station
in	KOOR	WORD	Coordination flag
out	ANZW	WORD	Indicator word
out	PAFE	BYTE	Parameterization error byte Return Code (00h=OK)
in_out	ANZ	WORD	internal use
in_out	GESE	WORD	internal use
in_out	KOPF_GESENDET	BOOL	internal use
in_out	ERSTER_BLOCK	BOOL	internal use
in_out	SENDEN_LAEUFT	BOOL	internal use
in_out	SENDEN_FERTIG	BOOL	internal use
in_out	LETZTER_BLOCK	BOOL	internal use
in_out	FEHLER	BOOL	internal use

ADR Periphery address with which you may call the CP 240. Via the hardware configuration you may set the periphery address.

QDB Number of the source data block of the own station.

QBDW 1. data word of the data block of the own station.

LANG Length of the data to send in words.

ZDB Number of the target block of the partner station

ZBDW 1. data word of the data block of the partner station

KOOR Coordination flag

The coordination flag serves to coordinate sending data. The coordination flag is set by the SEND order. As long as the flag is set, no other SEND order may be released. With FFFFh the coordination flag is deactivated.

ANZW Indicator word

Information concerning the order commissioning may be accessed by the indicator word. More may be found at "RK512 communication - Indicator word ANZW".

PAFE At proper function, all bits of this bit memory byte are "0". At errors an error code is entered. The error setting is self-acknowledging, i.e. after elimination of the error cause, the byte is set back to "0" again. The following errors may occur:

1 = Data block not present

2 = Data block too short

3 = Data block number outside valid range

ANZ, GESE, KOPF_GESE, ERSTER_BLOCK, SENDEN_LAEUFT, SENDEN_FERTIG, LETZTER_BLOCK, FEHLER These parameters are internally used. They serve the information exchange between the handling blocks.

S/R_ALL_RK512 (FC 4)

These FC serves for to deal with the FETCH and SEND orders in a passive station.

Declaration	Name	Type	Comment
in	ADR	INT	Logical Address
in	ANZW	WORD	Indicator word
out	PAFE	BYTE	Parameterization error byte Return Code (00h=OK)
in_out	GESE	WORD	internal use
in_out	ANZ	WORD	internal use
in_out	DB_KOPF	WORD	internal use
in_out	ABF_KOPF	WORD	internal use
in_out	KOPF_AUSGEW	BOOL	internal use
in_out	LETZTER_BLOCK	BOOL	internal use
in_out	SENDEN_LAEUFT	BOOL	internal use
in_out	EMPF_LAEUFT	BOOL	internal use
in_out	ENDE_KOM	BOOL	internal use
in_out	SEND_ALL	BOOL	internal use
in_out	RECEIV_ALL	BOOL	internal use
in_out	FEHLER	BOOL	internal use

ADR

Periphery address with which you may call the CP 240. Via the hardware configuration you may set the periphery address.

ANZW

Indicator word

Information concerning the order commissioning may be accessed by the indicator word. More may be found at "RK512 communication - Indicator word ANZW".

PAFE

At proper function, all bits of this bit memory byte are "0". At errors an error code is entered. The error setting is self-acknowledging, i.e. after elimination of the error cause, the byte is set back to "0" again. The following errors may occur:

1 = Data block not present

2 = Data block too short

3 = Data block number outside valid range

GESE, ANZ, DB_KOPF, ABF_KOPF, KOPF_AUSGEW, LETZTER_BLOCK, SENDE_LAEUFT, EMPF_LAEUFT, ENDE_KOM, SEND_ALL, RECEIVE_ALL, FEHLER

These parameters are internally used. They serve the information exchange between the handling blocks.

RK512 communication - Indicator word ANZW

Status and error reports

Status and error reports are created by the handling blocks:

- by the indicator word ANZW (information at order commissioning).
- by the parameter error byte PAFE (indication of a wrong order parameterization).

Content and structure of the indicator word ANZW

The "Indicator word" shows the status of a certain order on a CP.

In your PLC program you should keep one indicator word for each defined order at hand.

The indicator word has the following structure:

Byte	Bit 7 ... Bit 0
0	*Error messages CP* 00h: no errors 17h: Message too long 0Ch: Frame error 07h: Acknowledgement delay 0Ah: DBL exceeded
1	*Status management CPU* Bit 0: not used Bit 1: order is running 0: SEND/FETCH released 1: SEND/FETCH blocked Bit 2: Order ready without errors Bit 3: Order ready with errors Bit 7 ... Bit 4: not used

Error message CP Byte 0

In this byte error messages of the CP are entered. The error messages are only valid if the bit "Order ready with error" in the status bit is set simultaneously.

The following error messages may occur:

00h *no error*

If the bit "Order ready with error" is set, the CP had to reinitialize the connection, e.g. after a reboot or RESET.

17h *Message too long*

The received message is too long. Maximally 1024byte user data may be transferred.

07h *Acknowledgement delay*

The message was not acknowledged within the acknowledgement delay time.

0Ah *DBL exceeded*

The number of block repetitions, which may be set at the parameter "Data block length DBL" was exceeded.

Status management CPU Byte 1

Here you may see if an order has already been started, if an error occurred or if this order is blocked, e.g. a virtual connection doesn't exist any longer.

Bit 1: Order running

Set:	Per plug-in: when the CP received the order.
Delete:	Per plug-in: when an order has been commissioned (e.g. receipt received).
Analyze:	Per handling blocks: A new order is only send, when the order before is completely commissioned. Per user: when you want to know, if triggering a new order is convenient.

Bit 2: Order ready without errors

Set:	Per plug-in: when the according order has been commissioned without errors.
Delete:	Per plug-in: when the according order is triggered for a second time.
Analyze:	Per user: to proof that the order has been commissioned without errors.

Bit 3: Order ready with errors

Set:	Per plug-in: when the according order has been commissioned with errors. The cause of the error may be found in byte 0 of the indicator word.
Delete:	Per plug-in: when the according order is triggered for a second time.
Analyze:	Per user: to proof that the order has been commissioned with errors. If set, the error code may be found in byte 0 of the indicator word.

Chapter 4 CP 240 - serial

Overview

This chapter contains a description of the construction and the interfaces of the communication processor CP 240 with RS232- or RS485- interface respectively RS422/485 interface. VIPA distributes the communication processor CP 240 with different communication protocols that are explained in the following.

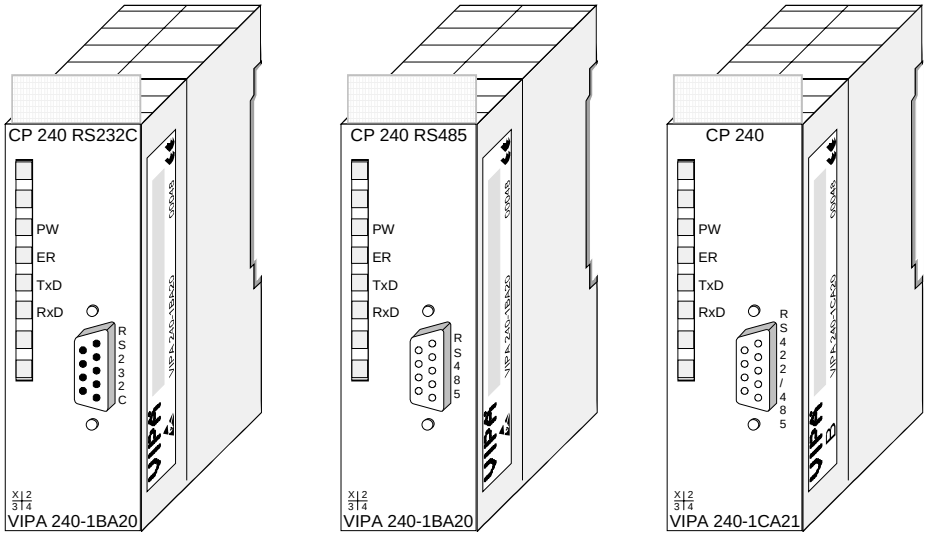
Content

System overview

Properties

- RS232 interface (only VIPA 240-1BA20)
- RS485 interface (only VIPA 240-1CA20)
- RS422/485 interface (only VIPA 240-1CA21)
- The protocols ASCII, STX/ETX, 3964(R), RK512 and Modbus are supported
- Configured by means of 16byte parameter data
- Up to 250 telegrams within the 1024Byte sized receive and send buffer
- Serial interface isolated to back plane bus
- Power supply by back plane bus

CP 240 RS232
CP 240 RS485
CP 240 RS422/485



Order data

Type	Order No.	Description
CP 240	VIPA 240-1BA20	CP 240 with RS232 interface Protocols: ASCII, STX/ETX, 3964(R), RK512, Modbus
CP 240	VIPA 240-1CA20	CP 240 with RS48 -Interface Protocols: ASCII, STX/ETX, 3964(R), RK512, Modbus
CP 240	VIPA 240-1CA21	CP 240 with RS422/485-Schnittstelle Protocols: ASCII, STX/ETX, 3964(R), RK512, Modbus

Fast introduction

Overview

The communication processors CP 240 enable the serial process connection to different destination or source systems. Depending on the module they are provided with an RS232 or an RS485 interface respectively an RS422/485 interface.

The CP 240 modules are supplied with operating voltage via the back plane bus.

For the internal communication the VIPA FCs are to be used. Here the data is transferred with a maximum block size of 12Byte.

For the project engineering of the CP 240 together with a CPU 21x in the Siemens SIMATIC Manager, the inclusion of the GSD VIPA_21x.gsd is required. To enable the CP 240 to communicate with the CPU, a hardware configuration for the system must always be executed.

A general description for the project engineering of the CP 240 is to be found in the chapter "Project engineering".

Parameters

For the parameterization you may send 16Byte parameter data to the CP that are differently assigned depending on the chosen protocol.

The parameterization happens via the hardware configuration in the Siemens SIMATIC Manager by including a protocol specific CP 240.

Protocols

After the inclusion of the GSD the CP 240 is available with the following protocols:

- ASCII
- STX/ETX
- 3964(R) and RK512
- Modbus (master, slave)

Communication

The serial communication happens via the deployment of handling blocks in the PLC user application. These handling blocks may be downloaded from ftp.vipa.de or received as part of the CD-ROM VIPA "ToolDemo".

Depending on the protocol the following handling blocks are used:

ASCII	STX 3964	RK512	Modbus	FC	Name
x	x		x	FC0	SEND_ASCII_STX_3964
x	x		x	FC1	RECEIVE_ASCII_3964
		x		FC2	FETCH_RK512
		x		FC3	SEND_RK512
		x		FC4	S/R_ALL_RK512
x	x	x		FC9	SYNCHRON_RESET
x				FC11	ASCII_FRAGMENT

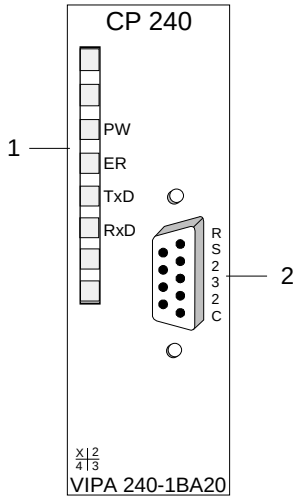
Note!

A communication with SEND and RECEIVE blocks is only possible if the parameter ANL of the SYNCHRON block has been set in the start-up-OB before.

Structure

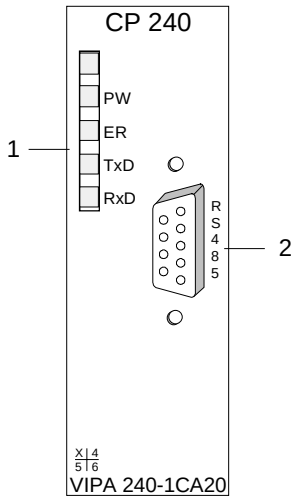
Front view

CP 240 RS232
240-1BA20



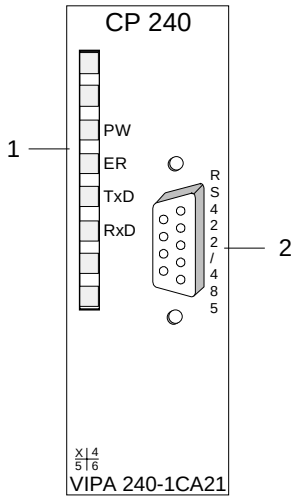
[1] LED status indicator
[2] 9pin serial D-type plug for RS232 communication

CP 240 RS485
240-1CA20



[1] LED status indicator
[2] 9pin serial D-type jack for RS485 communication

CP 240 RS422/485
240-1CA21



[1] LED status indicator
[2] 9pin serial D-type jack for RS422/485 communication

Components

Power supply

The communication processor receives power via the back plane bus.

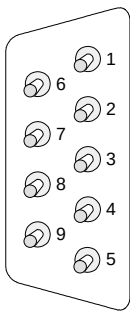
LEDs

The communication processor is provided with 4 LEDs for the purpose of displaying the operating status. The following table depicts the description and the color of these LEDs.

Name	Color	Description
PW	yellow	Indicates that power is available
ER	red	For Modbus this signalizes an internal error other protocols: error indicator for open circuit lines, overflow, parity or framing errors. The error LED is reset automatically after 4s. If diagnostics are enabled the error causes transmission of diagnostic bytes.
TxD	green	Transmit data
RxD	green	Receive data

RS232 interface

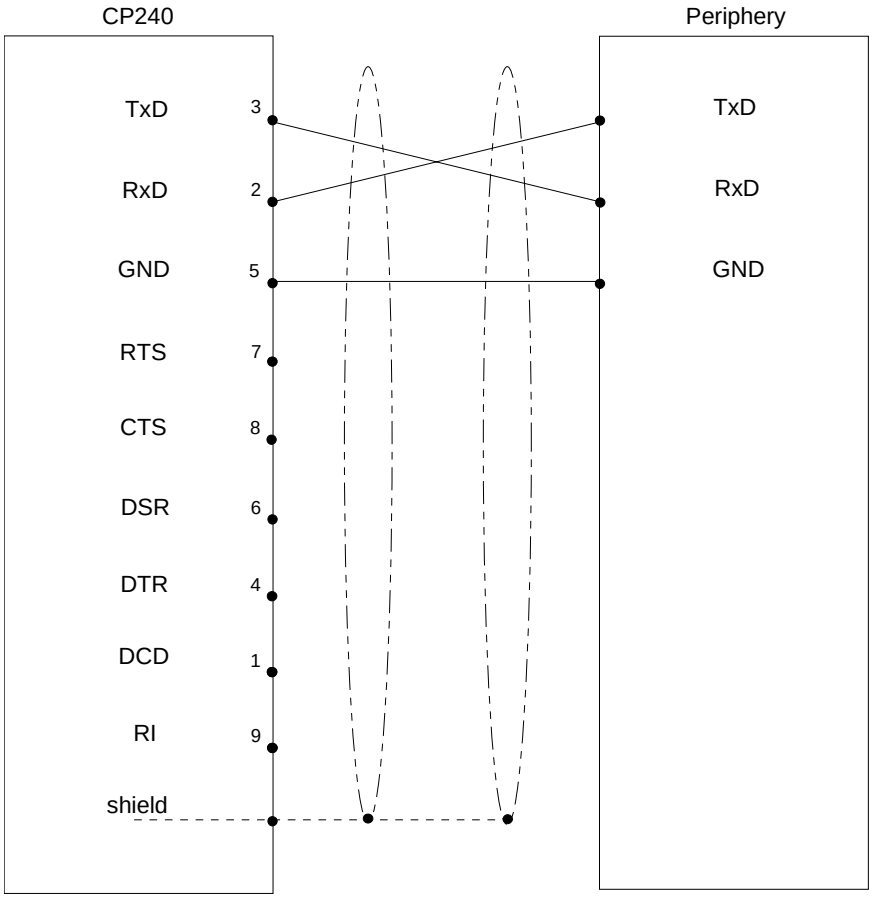
9pin D-type plug



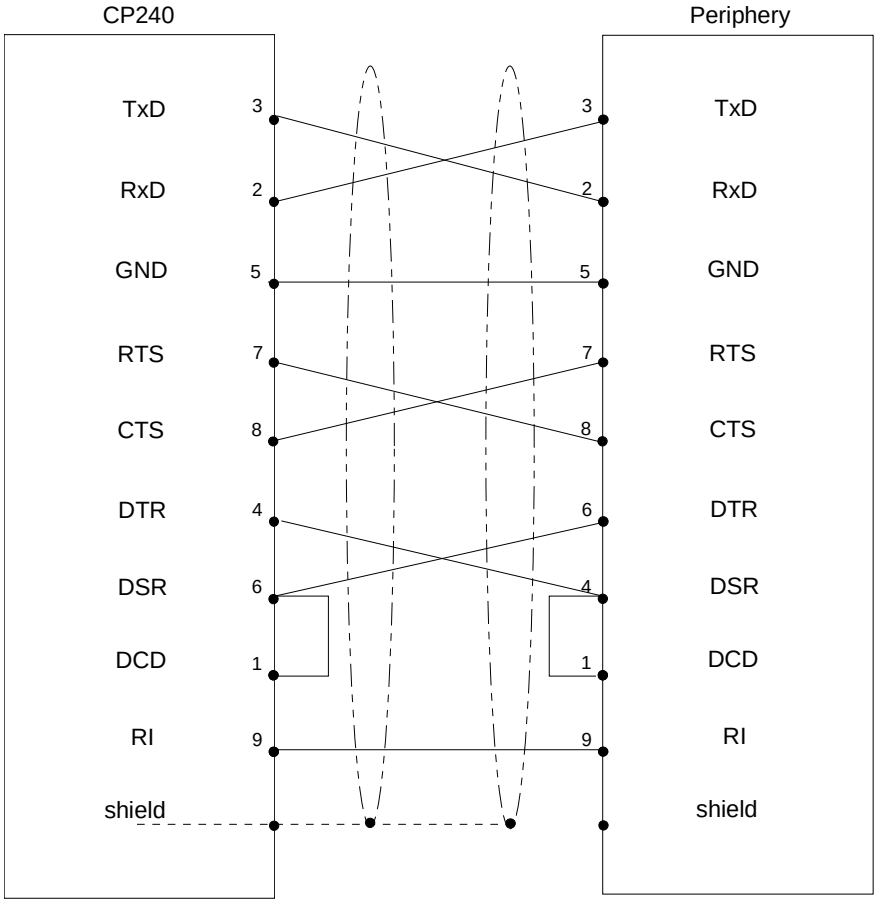
Pin	Designation		Signal description
1	DCD	Data Carrier Detect	Data can be received
2	RxD	Receive Data	Receive data from modem to CP 240
3	TxD	Transmit Data	Send data from CP 240 to modem
4	DTR	Data Terminal Ready	CP 240 indicates data terminal ready
5	GND	Signal Ground	GND Ground
6	DSR	Data Set Ready	Modem indicates data set ready
7	RTS	Request to send	CP 240 indicates request to send
8	CTS	Clear to send	Modem indicates the CP 240 to send
9	RI	Ring indicator	Ring indicator

- Logical conditions as voltage level
- Point-to-point connection with serial full duplex transfer
- Data transfer up to a distance of 15m
- Data transfer rate up to 115.2kbit/s

RS232 cabling without hardware handshake

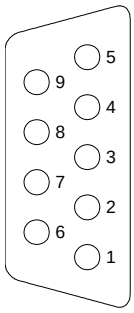


RS232 cabling with hardware handshake



RS485 interface

- Logical states represented by voltage differences between the two cores of a twisted pair cable
- Serial bus connection in two-wire technology using half duplex mode
- High noise immunity
- Connection of up to 32 stations
- Data transfer up to 500m
- Data transfer rate up to 115.2kbit/s

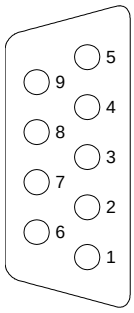


9pin D-type jack

Pin	Designation	Input/Output	Signal description
1	n.c.	---	
2	n.c.	---	
3	RxD/TxD-P (Line B)	Input/Output	Receive/Send data
4	RTS	Output	Request to send
5	M5V	Output	Ground isolated
6	P5V	Output	5V isolated
7	n.c.	---	
8	RxD/TxD-N (Line A)	Input/Output	Receive/Send data
9	n.c.	---	

RS422/485-Schnittstelle

- Logical conditions as voltage difference between 2 twisted lines
- Serial bus connection
  Full-duplex: Four-wire operation (RS422)
  Half-duplex: Two-wire operation (RS485)
- Line length: 250m at 115.2kbit/s ... 1200m at 19.2kbit/s
- Data transfer rate up to 115.2kbit/s



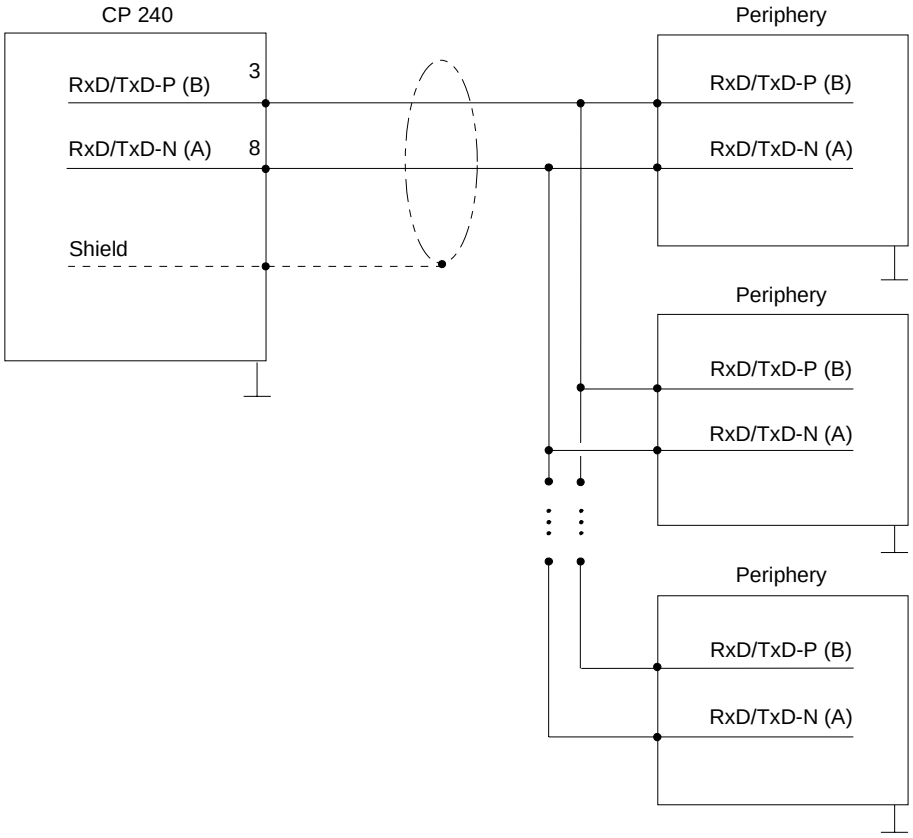
9pin D-type jack

Pin	Designation	Input/Output	Signal description
1	n.c.	---	
2	TxD-P (Line B)	Output	Send data (RS422)
3	RxD-P (Line B) RxD/TxD-P (Line B)	Input Input/Output	Receive data (RS422) Receive/Send data (RS485)
4	RTS	Output	Request to send
5	M5V	Output	Ground isolated
6	P5V	Output	5V isolated
7	TxD-N (Line A)	Output	Send data (RS422)
8	RxD-N (Line A) RxD/TxD-N (Line A)	Input Input/Output	Receive data (RS422) Receive/Send data (RS485)
9	n.c.	---	

Note!

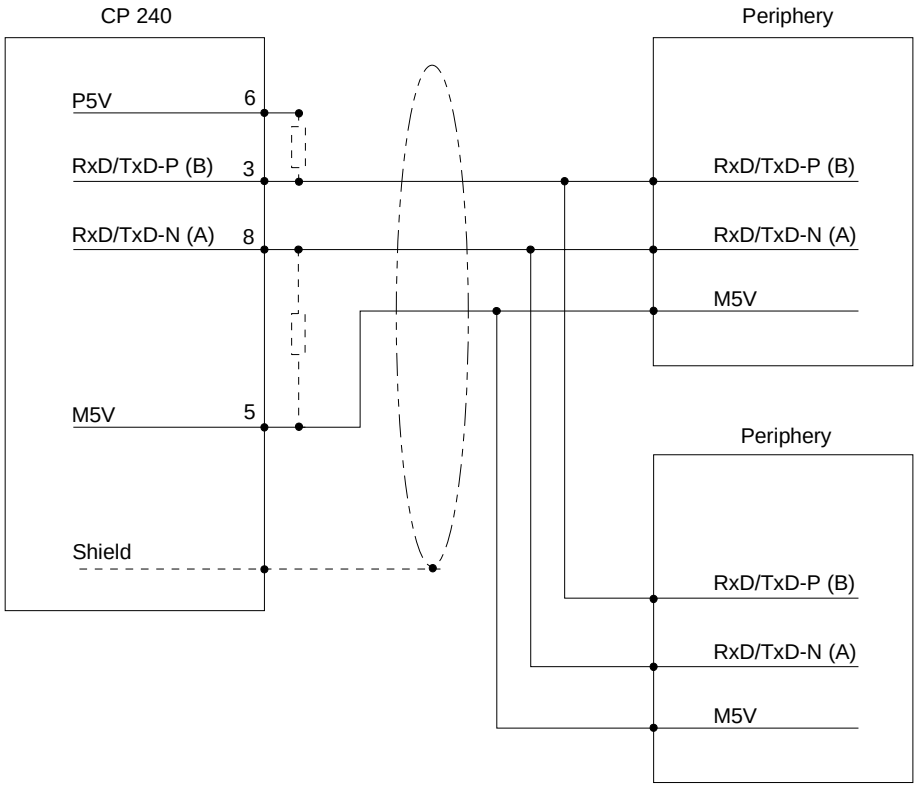
Never connect the shield of the cable with M5V (Pin 5), as this could destroy the interface!

RS485 cabling with Profibus cable

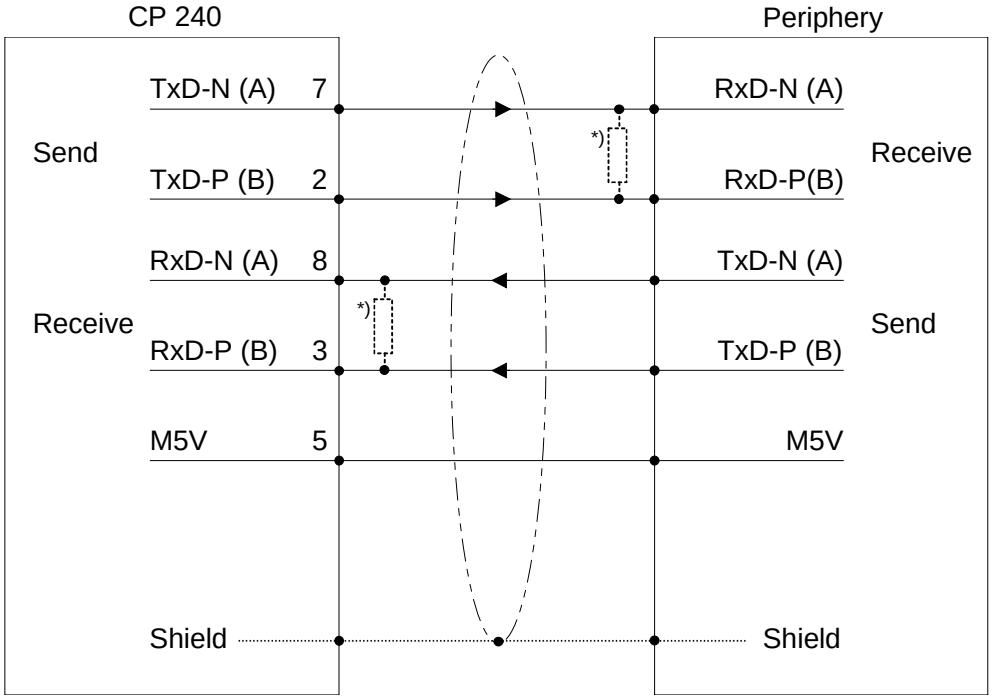


RS485 cabling with defined static levels

Pin 6 (P5V) of the isolated interfaces carries the isolated 5V supply with the respective ground on pin 5 (M5V). You may use this isolated voltage to provide defined static voltage levels on the signaling lines by means of resistors and ensure that reflections are reduced to a minimum.



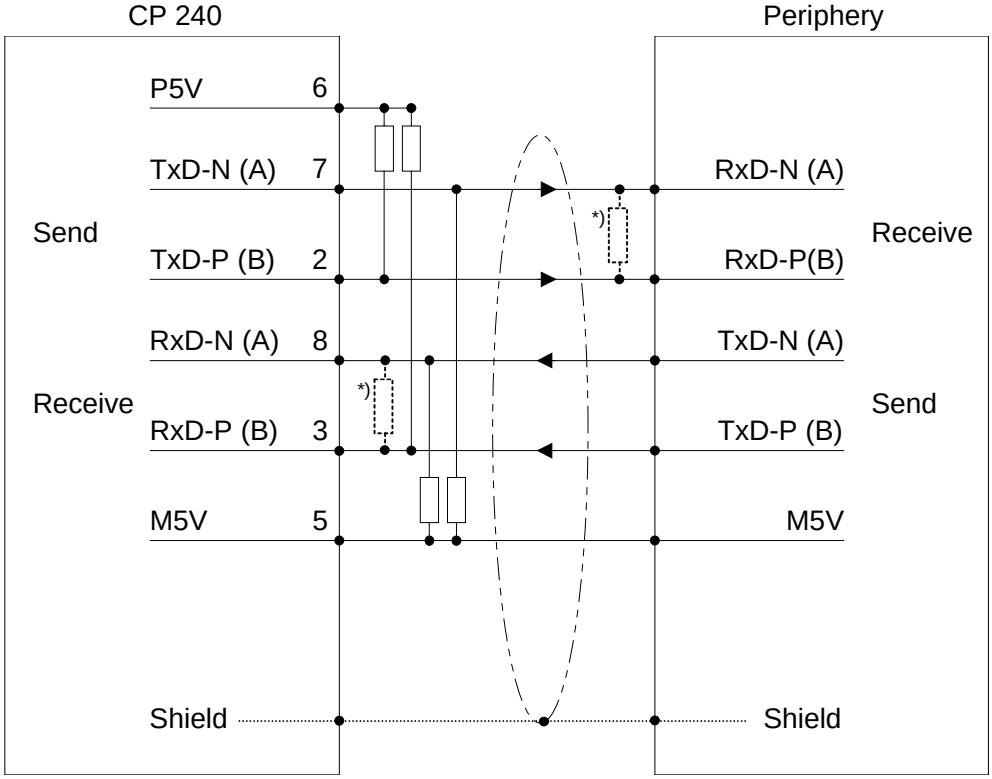
RS422 cabling



*) In the case of cables >50m you have to solder in a terminating resistor of approx. 330Ω on the receiver for data free traffic.

RS422 cabling with defined static levels

Pin 6 (P5V) of the isolated interfaces carries the isolated 5V supply with the respective ground on pin 5 (M5V). You may use this isolated voltage to provide defined static voltage levels on the signaling lines by means of resistors and ensure that reflections are reduced to a minimum.



*) In the case of cables >50m you have to solder in a terminating resistor of approx. 330Ω on the receiver for data free traffic.

ASCII / STX/ETX / 3964(R) / RK512 - Basics

ASCII

ASCII data communication is one of the simple forms of data exchange that can be compared to a multicast/broadcast function.

Individual messages are separated by means of 2 windows in time. The sending station has to transmit data messages within the character delay time (ZVZ) or receive window that was defined in the receiving station.

The receiving station must acknowledge the receipt of the message within the "time delay after command" (ZNA) or command window that was defined in the sending station.

These time stamps can be used to establish a simple serial communication link between PLC and PLC.

The Bit FRG is only reset when the data has been transferred and the ZNA has expired.

ASCII-fragmented

Using ASCII a telegram is only handled over to the CPU when it has been received completely. ASCII-fragmented allows you by means of the usage of the Receive block FC11 (ASCII_FRAGMENT) to handle on big telegrams to the CPU in blocks as soon as they have been received. For this the block length is 12Byte. With ASCII-fragmented the CP doesn't wait until the telegram has been received completely.

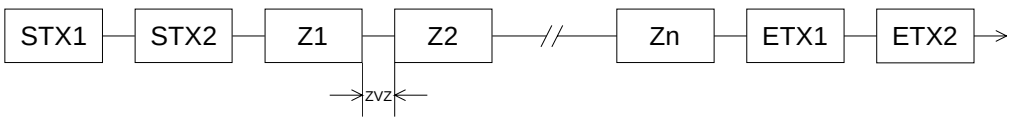
STX/ETX

STX/ETX is a simple protocol employing headers and trailers. The STX/ETX procedure is suitable for the transfer of ASCII characters (20h…7Fh). It does not use block checks (BCC). Any data transferred from the periphery must be preceded by an STX (Start of Text) followed by the data characters. An ETX (End of Text) must be inserted as the terminating character.

The effective data, which includes all the characters between STX and ETX, are transferred to the CPU when the ETX has been received.

When data is sent from the CPU to a peripheral device, any user data is handed to the CP 240 where it is enclosed with an STX start character and an ETX termination character and transferred to the communication partner.

Message structure:



You may define up to 2 start and end characters. It is also possible to specify a ZNA for the sending station.

3964(R)

The 3964(R) procedure controls the data transfer of a point-to-point link between the CP 240 and a communication partner. The procedure adds control characters to the message data during data transfer. These control characters may be used by the communication partner to verify the complete and error free receipt.

The procedure employs the following control characters:

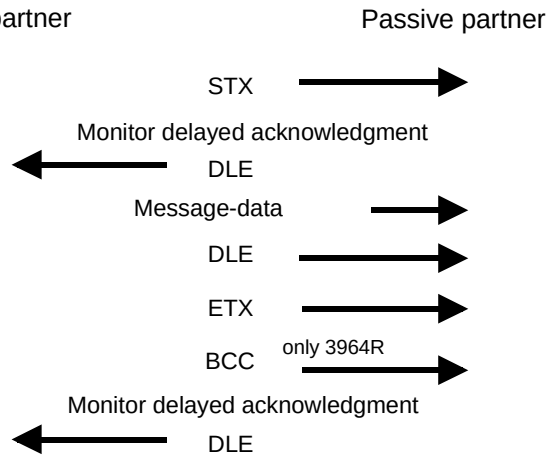
- STX Start of Text
- DLE Data Link Escape
- ETX End of Text
- BCC Block Check Character (only for 3964R)
- NAK Negative Acknowledge

Note!

When a DLE is transferred as part of the information it is repeated to distinguish between data characters and DLE control characters that are used to establish and to terminate the connection (DLE duplication). The DLE duplication is reversed in the receiving station.

The 3964(R) procedure requires that a lower priority is assigned to the communication partner. When communication partners issue simultaneous send commands the station with the lower priority will delay its send command.

Procedure

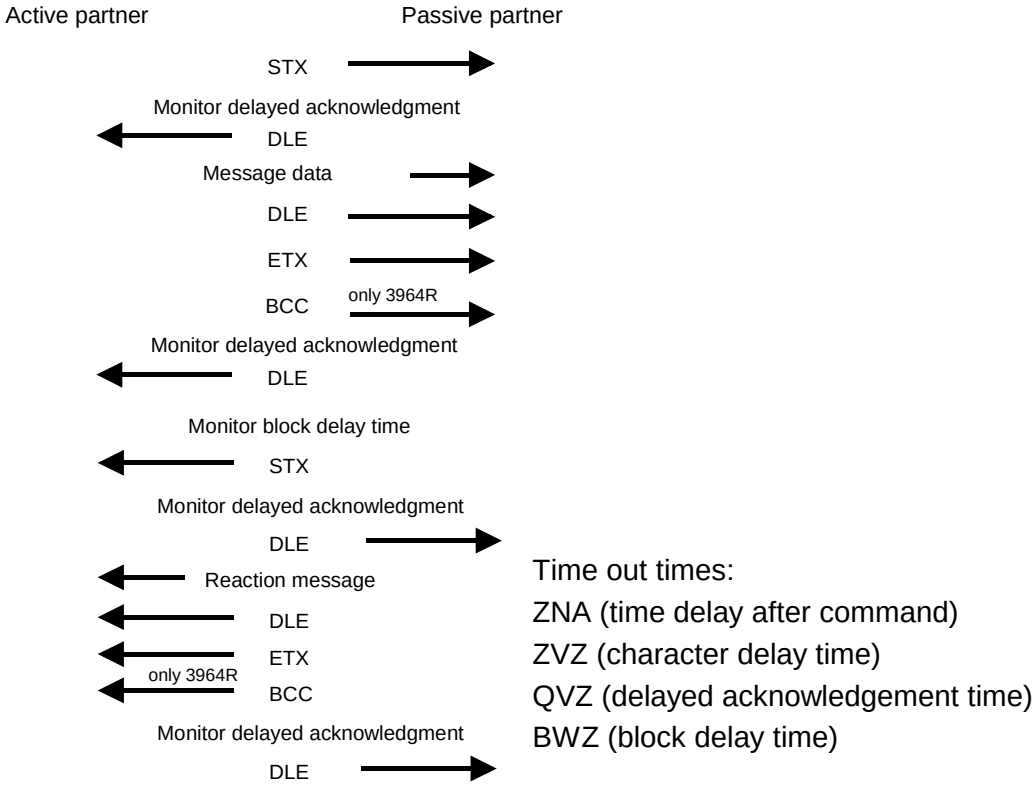


You may transfer a maximum of 250Byte per message.

3964(R) with RK512

The RK512 is an extended form of the 3964(R) procedure. The difference is that a message header is sent ahead of the message data. The header contains data about the size, type and length of the message data.

Procedure



Timeout times

The QVZ is monitored between STX and DLE and between BCC and DLE. ZVZ is monitored for the entire period of receiving the message.

When the QVZ expires after an STX, the STX is repeated. This process is repeated 5 times after which the attempt to establish a connection is terminated by the transmission of a NAK. The same sequence is completed when a NAK or any other character follows an STX.

When the QVZ expires after a message (following the BCC-byte) or when a character other than DLE is received the attempt to establish the connection and the message are repeated. This process is also repeated 5 times after which a NAK is transmitted and the attempt is terminated.

BWZ is the max. time between acknowledgement of a request telegram (DLE) and STX of the answer telegram. When exceeding the BWZ it is repeatedly attempted (parameterizable by DBL) to send the request telegram. If these attempts are unsuccessful, the transmission is interrupted.

Passive operation

When the procedure driver is expecting a connection request and it receives a character that is not equal to STX it will transmit a NAK. The driver does not respond with an answer to the reception of a NAK.

When the ZVZ is exceeded at reception, a NAK is send and it is waited for a new connection.

When the driver is not ready yet at reception of the STX, it sends a NAK.

Block check character (BCC-Byte)

3964R appends a **B**lock **c**heck **c**haracter to safeguard the transmitted data. The BCC-Byte is calculated by means of an XOR function over the entire data of the message, including the DLE/ETX.

When a BCC-Byte is received that differs from the calculated BCC, a NAK is transmitted instead of the DLE.

Initialization conflict

If two stations should simultaneously attempt to issue a connection request within the QVZ then the station with the lower priority will transmit the DLE and change to receive mode.

Data Link Escape (DLE-character)

The driver duplicates any DLE-character that is contained in a message, i.e. the sequence DLE/DLE is sent. During the reception, the duplicated DLEs are saved as a single DLE in the buffer. The message always terminates with the sequence DLE/ETX/BCC (only for 3964R).

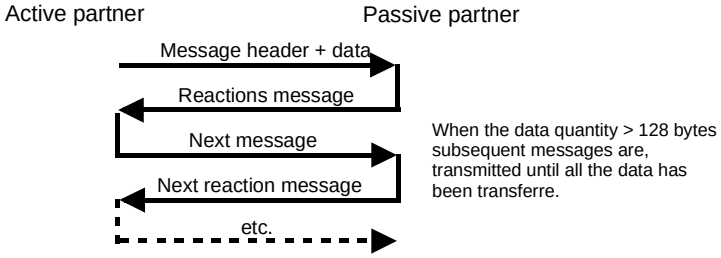
The control codes : 02h = STX
03h = ETX
10h = DLE
15h = NAK

When ZVZ expires during the reception, the driver will send a NAK and wait for another connection request.

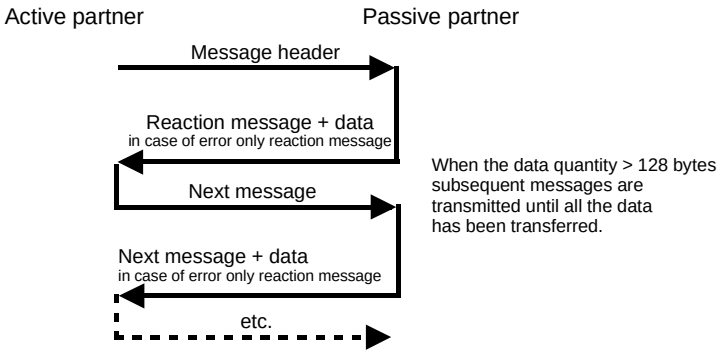
The driver also sends a NAK when it receives an STX while it is not ready.

Logical message sequence

SEND (transmission of data)



FETCH (retrieving data)

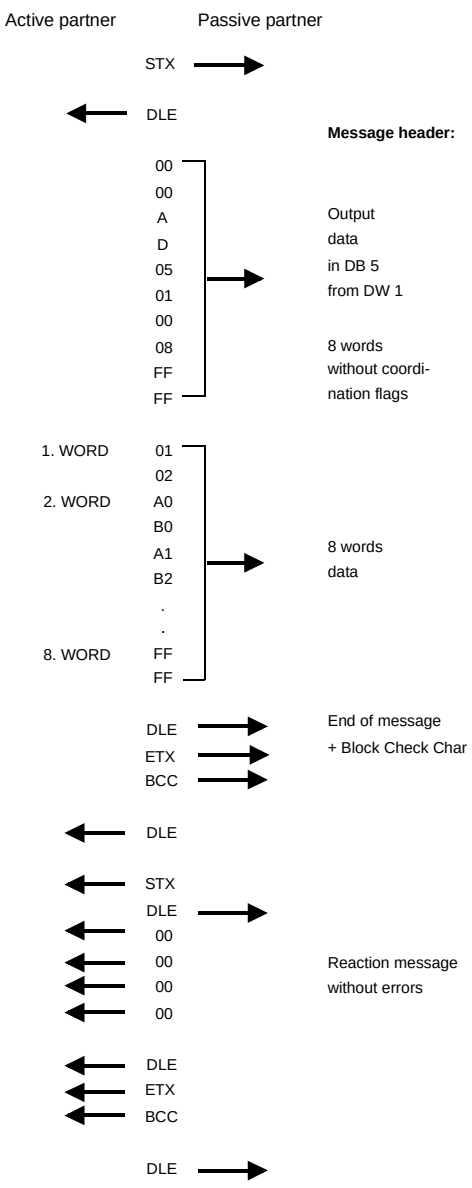


In both cases the procedures will time out after a maximum period of 5s during which a reaction must be received, else the reception is terminated.

Message contents Every message has a header. Depending on the history of the message traffic, this header will contain all the required information.

Structure of the output message

Sample output message



Normal message

Byte		
0	00	Message
1	00	flag
2	A	Output command
3	X	type of data
4	xx	Parameter 1
5	xx	Destination
6	yy	Parameter 2
7	yy	Quantity
8	zz	Parameter 3
9	zz	Coordination flag
10	aa	
-	bb	Data
N	xy	

with N = 10 ... 127

Reaction message

Byte		
0	00	Flag for
1	00	reaction
2	00	message
3	xx	Error code

When the data exceeds 128Byte, additional messages will be sent.

Structure of additional messages

Next message

Byte		
0	FF	Flag for
1	00	next message
2	A	Output command
3	X	Data type
4	aa	
-	bb	Data
N	xy	

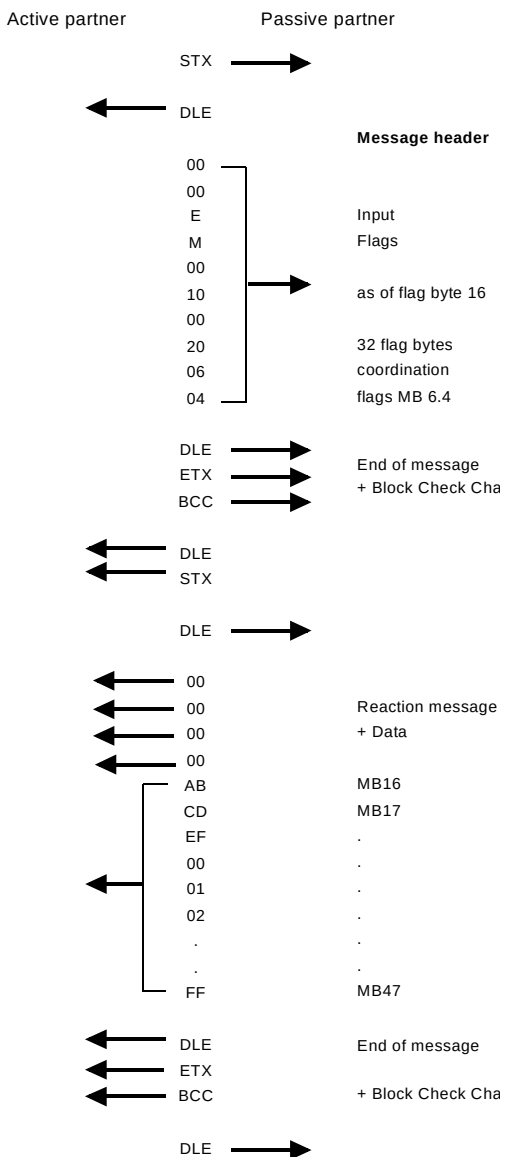
with N = 4 ... 127

Next reaction message

Byte		
0	FF	Flag for
1	00	next reaction
2	00	message
3	xx	Error code

Structure of the input message

Sample input message



Normal message

Byte

0	00	Message
1	00	identifier
2	E	Input command
3	X	Data type
4	xx	Parameter 1
5	xx	Destination
6	yy	Parameter 2
7	yy	Quantity
8	zz	Parameter 3
9	zz	Coordinnation flag

Reaction message

Byte

0	00	Reaction
1	00	message
2	00	flag
3	xx	Error code
4	aa	
-	bb	Data
N	xy	

with N = 4 ... 127

When the data exceeds 128Byte, additional messages will be sent.

Structure of additional messages

Next message

Byte

0	FF	Flag for
1	00	next message
2	E	Input command
3	X	data type

Next reaction message

Byte

0	FF	Flag for next
1	00	reaction
2	00	message
3	xx	Error code
4	aa	
-	bb	Data
N	xy	

with N = 4 ... 127

Coordination flags The coordination flag is set in the partner PLC in active-mode when a message is being received. This occurs for input as well as for output commands. When the coordination flag has been set and a message with this flag is received, then the respective data is not accepted (or transferred) and a reject message is sent (error code 32h). In this case the user has to reset the coordination flag in the partner PLC.

ASCII / STX/ETX / 3964(R) / RK512 - Communication principle

Communication via handling blocks

The serial communication happens via the deployment of handling blocks in the PLC user application. These handling blocks may be downloaded from ftp.vipa.de or received as part of the CD-ROM VIPA "ToolDemo".

Depending on the protocol the following handling blocks are used:

ASCII	STX 3964	RK512	Modbus	FC	Name
x	x		x	FC0	SEND_ASCII_STX_3964
x	x		x	FC1	RECEIVE_ASCII_3964
		x		FC2	FETCH_RK512
		x		FC3	SEND_RK512
		x		FC4	S/R_ALL_RK512
x	x	x		FC9	SYNCHRON_RESET
x				FC11	ASCII_FRAGMENT

Note!

A communication with SEND and RECEIVE blocks is only possible if the parameter ANL of the SYNCHRON block has been set in the start-up-OB before.

Send and receive data

Data that is written into the according data channel by the CPU via the back plane bus are written into the according send buffer (1024Byte) by the communication processor and from here put out via the interface.

When the communication processor receives data via the interface, the data is stored in a ring buffer (1024Byte). The CPU via the data channel may read the received data.

Communication via back plane bus

The exchange of received telegrams via the back plane bus happens asynchronously. When a complete telegram has arrived via the serial interface (expiration of the ZVZ), this is stored in a ring buffer of 1024Byte. The length of the ring buffer determines the max. length of a telegram. There may be stored up to 250 telegrams according to the parameterization whereby their overall length may not exceed 1024.

When the buffer is filled up, new telegrams are rejected. A complete telegram is divided into blocks of 12Byte and transferred to the back plane bus. The reassembly of the data blocks has to take place inside the CPU.

Communication with ASCII-fragmented

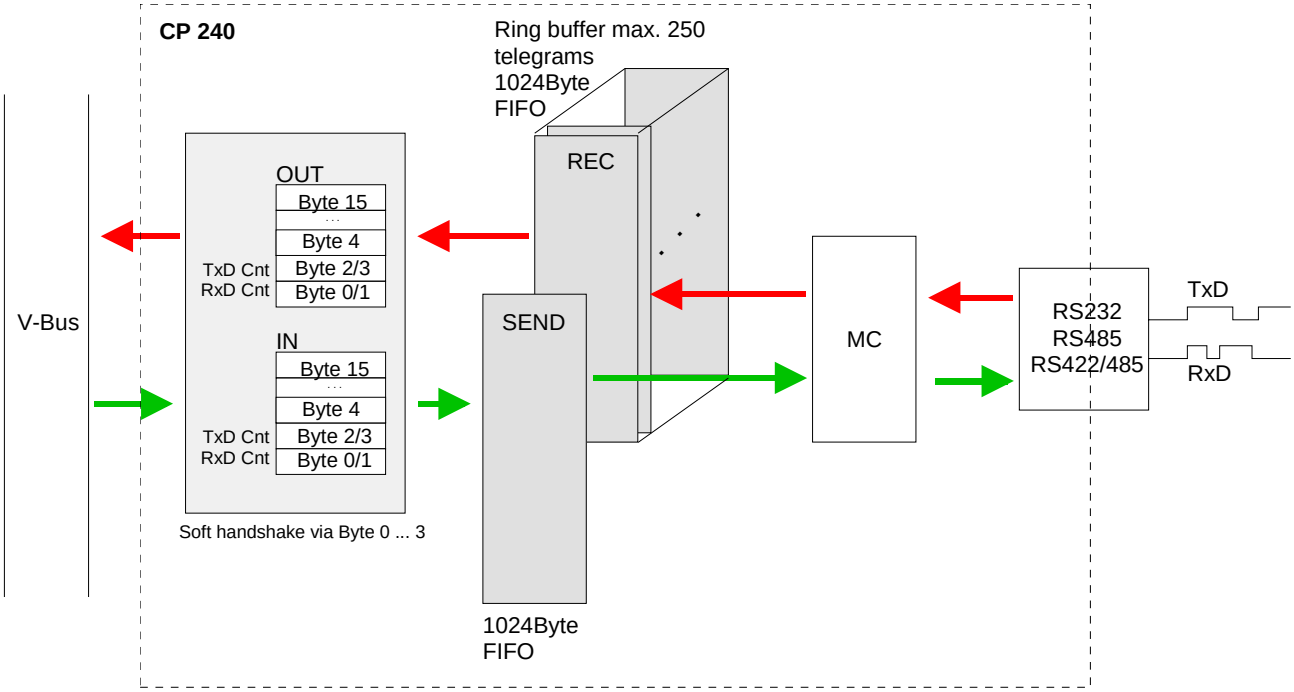
With ASCII-fragmented incoming data of a telegram is in blocks immediately transferred to the CPU. Here the block length is at least 12Byte. At ASCII-fragmented the CP doesn't wait until the complete telegram has been received.

Tasks of the CPU

The CPU has to split the telegram to send into blocks of 12Byte and transfer them via the back plane bus to the CP 240. In the CP 240 these blocks are assembled in the send buffer, proofed for completeness and then sent to the serial interface.

For the data transfer via the back plane bus is asynchronous, a "software handshake" is used between the CP 240 and the CPU. The register for the data transfer from the CP 240 has a width of 16Byte. The bytes 0 to 3 (word 0 and 2) are reserved for the handshake.

The following picture illustrates this:



Software handshake

For the deployment of the CP 240 together with a System 200V CPU VIPA offers you a series of standard handler blocks that provide the software handshake comfortable and easy.

At deployment of the CP 240 without handler blocks, the functionality is elucidated with an example of data send and receive.

Example SEND data

For example, a telegram with 30Byte length is to send. The CPU writes the first 12Byte user data of the telegram into the Bytes 4 to 15. Byte 2/3 contain the telegram length, i.e. "30". The CP 240 receives the data via the back plane bus and copies the 12Byte user data into the send buffer. For the acknowledgement of the telegram the CP 240 writes the value "30" back to Byte 2/3 (length of the telegram).

At reception of the "30", the CPU can send further 12Byte user data to Byte 4 to 15 and the rest length of the telegram ("18" Byte) to Byte 2/3 to the CP 240. Again, this stores the user data in the send buffer and sends back the length information ("18") in Byte 2/3 to the CPU.

The CPU receives the "18" and sends the remaining 6Byte user data in the Bytes 4 to 9 and the according rest length ("6") in Byte 2/3 to the CP 240. The user data is stored in the send buffer and the value "6" is send back to the CPU via Byte 2/3.

The CPU receives the "6" and sends back a "0" via Byte 2/3. The CP 240 now initializes the sending of the telegram via the serial interface. After data transfer is completed, the CP 240 sends back a "0" to the CPU via Byte 2/3.

At reception of the "0", the CPU is able to send a new telegram to the CP 240.

Example RECEIVE data

The interface of the CP 240 has e.g. received a telegram with a length of 18Byte via the serial interface. The CP 240 writes the 12Byte user data into the Bytes 4 to 15 of the receive buffer and the telegram length (i.e. "18") into Byte 0/1. The data is transferred to the CPU via the back plane bus. The CPU stores the 12Byte user data and sends back the length value "18" to the CP 240.

At reception of the "18", the CP 240 writes the remaining 6Byte user data into the Bytes 4 to 9 of the receive buffer and the received length of user data ("6") in Bytes 0/1. The user data are stored by the CPU and "6" in Byte 0/1 is returned to the CP 240.

Having received the "6", the CP 240 returns the value "0" via Byte 0/1, i.e. the telegram has been completed. The CPU acknowledges with another "0" in Byte 0/1 to the CP 240.

Receiving "0" the CP 240 may send another telegram to the CPU.

ASCII / STX/ETX / 3964(R) / RK512 - Parameterization

General

You may configure the CP 240 by means of 16Byte of configuration data. The structure of the parameter data depends on the selected protocol or.

Please regard at the hardware configuration to use the CP 240 according to the chosen protocol.

Below follows a list of the parameter bytes with the respective default values.

Structure of the parameter bytes of ASCII

Byte	Function	Range	Default parameter
0	Baud rate	00h: Default (9600Baud) 01h: 150Baud 02h: 300Baud 03h: 600Baud 04h: 1200Baud 05h: 1800Baud 06h: 2400Baud 07h: 4800Baud 08h: 7200Baud 09h: 9600Baud 0Ah: 14400Baud 0Bh: 19200Baud 0Ch: 38400Baud 0Dh: 57600Baud 0Fh: 76800Baud 0Eh: 115200Baud	00h: 9600Baud
1	Protocol	01h: ASCII 11h: ASCII fragment	01h: (ASCII)
2	Bit 1/0 Data bits	00b: 5 Data bits 01b: 6 Data bits 10b: 7 Data bits 11b: 8 Data bits	11b: 8 Data bits
	Bit 3/2 Parity	00b: none 01b: odd 10b: even 11b: even	00b: none
	Bit 5/4 Stop bits	01b: 1 10b: 1.5 11b: 2	01b: 1 Stop bit
	Bit 7/6 Flow control	00b: none 01b: Hardware 10b: XON/XOFF	00b: none
3	reserved	0	0
4	ZNA (*20ms)	0..255	0
5	ZVZ (*20ms)	0..255	10
6	No. of receive buffers	1..250	1
7...15	reserved		

Structure of parameter bytes for STX/ETX

Byte	Function	Range of values	Default parameters
0	Baud rate	00h: Default (9600Baud) 01h: 150Baud 02h: 300Baud 03h: 600Baud 04h: 1200Baud 05h: 1800Baud 06h: 2400Baud 07h: 4800Baud 08h: 7200Baud 09h: 9600Baud 0Ah: 14400Baud 0Bh: 19200Baud 0Ch: 38400Baud 0Dh: 57600Baud 0Fh: 76800Baud 0Eh: 115200Baud	00h: 9600Baud
1	Protocol	02h: STX/ETX	02h (STX/ETX)
2	Bit 1/0 Data bits	00b: 5 Data bits 01b: 6 Data bits 10b: 7 Data bits 11b: 8 Data bits	11b: 8 Data bits
	Bit 3/2 Parity	00b: none 01b: odd 10b. even 11b: even	00b: none
	Bit 5/4 Stop bits	01b: 1 10b: 1.5 11b: 2	01b: 1 Stop bit
	Bit 7/6 Flow control	00b: none 01b: Hardware 10b: XON/XOFF	00b: none
3	reserved	0	0
4	ZNA (*20ms)	0..255	0
5	TMO (*20ms)	1..255	10
6	Number of start flags	0..2	01
7	Start flag 1	0..255	02
8	Start flag 2	0..255	0
9	Number of end flags	0..2	01
10	End flag 1	0..255	03
11	End flag 2	0..255	0
12	reserved		
13	reserved		
14	reserved		
15	reserved		

Structure of parameter bytes for 3964(R) / 3964(R) with RK512

Byte	Function	Range of values	Default parameters
0	Baud rate	00h: Default (9600Baud) 01h: 150Baud 02h: 300Baud 03h: 600Baud 04h: 1200Baud 05h: 1800Baud 06h: 2400Baud 07h: 4800Baud 08h: 7200Baud 09h: 9600Baud 0Ah: 14400Baud 0Bh: 19200Baud 0Ch: 38400Baud 0Dh: 57600Baud 0Fh: 76800Baud 0Eh: 115200Baud	00h: 9600Baud
1	Protocol	03h: 3964 04h: 3964R 05h: 3964 + RK512 06h: 3964R + RK512	03h: 3964
2	Bit 1/0 Data bits	00b: 5 Data bits 01b: 6 Data bits 10b: 7 Data bits 11b: 8 Data bits	11b: 8 Data bits
	Bit 3/2 Parity	00b: none 01b: odd 10b .even 11b: even	00b: none
	Bit 5/4 Stop bits	01b: 1 10b: 1.5 11b: 2	01b: 1 Stop bit
	Bit 7/6 Flow control	reserved	-
3	reserved	0	0
4	ZNA (*20ms)	0..255	0
5	ZVZ (*20ms)	0..255	10
6	QVZ (*20ms)	0..255	25
7	BWZ (*20ms)	0..255	100
8	STX repetitions	0..255	5
9	DBL	0..255	6
10	Priority	0: low 1: high	0: low
11	reserved		
12	reserved		
13	reserved		
14	reserved		
15	reserved		

Parameter description

Baud rate

The data communication rate in Bit/s (Baud).
You may select one of the following values:

00h:	Default (9600Baud)
01h:	150Baud
02h:	300Baud
03h:	600Baud
04h:	1200Baud
05h:	1800Baud
06h:	2400Baud
07h:	4800Baud
08h:	7200Baud
09h:	9600Baud
0Ah:	14400Baud
0Bh:	19200Baud
0Ch:	38400Baud
0Dh:	57600Baud
0Fh:	76800Baud
0Eh:	115200Baud

Default: 0 (9600Baud)

Protocol

The protocol to be used. This setting determines the further structure of the parameter data.
The following options are available:

01h:	ASCII
02h:	STX/ETX
03h:	3964
04h:	3964R
05h:	3964 and RK512
06h:	3964R and RK512
11h:	ASCII fragment

Transfer parameter byte

For every character frame there are 3 data formats available. The data formats are different in the number of data bits, with or without parity bit and number of stop bits.

The transfer parameter byte has the following structure:

Byte	Function	Range	Default parameter
2	Bit 1/0 Data bits	00b: 5 Data bits 01b: 6 Data bits 10b: 7 Data bits 11b: 8 Data bits	11b: 8 Data bits
	Bit 3/2 Parity	00b: none 01b: odd 10b: even 11b: even	00b: none
	Bit 5/4 Stop bits	01b: 1 10b: 1,5 11b: 2	01b: 1 Stop bit
	Bit 7/6 Flow control	00b: none 01b: Hardware 10b: XON/XOFF	00b: none

Data bits

Number of *data bits* that represent a character.

Parity

The parity is depending on the value even or odd. For the purposes of the parity check, the information bits are expanded by the parity bit. The value of the parity bit ("0" or "1") completes the value of all the bits to obtain a pre-arranged state. If the parity was not specified, the parity bit is set to "1" but it is not included in the assessment.

Stop bits

The stop bits are appended to each character and signify the end of the character.

Flow control
(at ASCII and STX/ETX)

This is a mechanism that synchronizes the data transfer when the transmitting station sends the data faster than it can be processed by the receiving station. Flow control can be hardware- or software-based (XON/XOFF). Hardware flow control employs the RTS and CTS lines and these must therefore be wired accordingly.

Software flow control employs the control characters XON=11h and XOFF=13h. Please remember that your data must not contain these control characters.

Default: 13h (data bits: 8, parity: none, stop bits: 1, flow control: none)

Time delay after command (ZNA)

The delay time that must expire before a command is executed. The ZNA is specified in units of 20ms.

Range: 0 ... 255 *Default: 0*

Character delay time (ZVZ)
(for ASCII, 3964(R) and RK512)

The character delay time defines the maximum time that may expire between two characters of a single messages during the reception of the message. The ZVZ is defined in units of 20ms.

When the ZVZ=0 the character delay time (ZVZ) will be calculated automatically (about double character time).

Range: 0 ... 255 *Default: 10*

Number of receive buffers
(only for ASCII)

Defines the number of receive buffers. When only 1 receive buffer is available no more data can be received while the receive buffer is occupied. The received data can be redirected into an unused receive buffer when you chain up to a maximum of 250 receive buffers.

Range: 1 ... 250 *Default: 1*

Timeout (TMO)
(only for STX/ETX)

TMO defines the maximum time between two messages. TMO is specified in units of 20ms.

Range: 1 ... 255 *Default: 10*

Number of start flags
(only for STX/ETX)

You can select 1 or 2 start flags. When you select "1" as number of start flags, the contents of the 2nd start flag (byte 8) is ignored.

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

Start flag 1 and 2 (STX)
(only for STX/ETX)

The ASCII value of the start character that precedes a message to signify the start of a data transfer. You may select 1 or 2 start characters. When you are using 2 start characters you have to specify "2" at "Number of start flags ".

Start character 1, 2: *Range: 0 ... 255* *Default: 2 (char. 1)*
0 (char. 2)

Number of end flags
(only for STX/ETX)

You can select 1 or 2 end flags. When you select "1" as number of end flags, the contents of the 2nd end flag (byte 11) is ignored.

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

End flag 1 and 2 (ETX)
(only for STX/ETX)

The ASCII value of the end character that follows a message to signify the end of the data transfer. You may specify 1 or 2 end characters. When you are using 2 end characters you have to enter a "2" for "number of end flags".

End character 1, 2: *Range: 0 ... 255* *Default: 3 (char. 1)*
0 (char. 2)

Delayed acknowledgment time (QVZ)
(for 3964(R), RK512)

The delayed acknowledgment time defines the maximum time for the acknowledgment from the partner when the connection is being established. The QVZ is specified in units of 20ms.

Range: 0 ... 255 *Default: 25*

Block delay time (BWZ)
(for 3964(R), RK512)

BWZ is the max. time between acknowledgement of a request telegram (DLE) and STX of the answer telegram.

The BWZ is specified in units of 20ms.

Range: 0 ... 255 *Default: 100*

STX repetitions
(for 3964(R), RK512)

Maximum number of allowed attempts for a CP 240 to establish a connection.

Range: 0 ... 255 *Default: 3*

Repetitions of data blocks (DBL) if exceeding BWZ
(for 3964(R), RK512)

With exceeding the block waiting time (BWZ) you can set the maximum number of repetitions for the request telegram by means of the parameter DBL. If these attempts are unsuccessful, the transmission is interrupted.

Range: 0 ... 255 *Default: 6*

Priority
(for 3964(R), RK512)

A communication partner has a high priority when its transmit request supersedes the transmit request of a partner. When the priority is lower, it must take second place after the transmit request of the partner.

The priorities of the two partners must be different for the 3964(R) and RK512 protocols.

You may select one of the following settings:

0: low

1: high *Default: 0 (low)*

Modbus - Basics

Overview

The Modbus protocol is a communication protocol that defines a hierarchic structure between a master and several slaves.

Physically, Modbus transmits via a serial half-duplex core as point-to-point connection with RS232 or as multi-point connection with RS485.

Master-Slave-Communication

There are no bus conflicts for the master is only able to communicate with one slave at a time. After the master requested a message, it waits for an answer until an adjustable wait period has expired. During waiting is no other communication possible.

Telegram-structure

The request telegrams of the master and the respond telegrams of a slave has the same structure:

Start ID	Slave address	Function code	Data	Flow control	End ID

Broadcast with slave address = 0

A request may be addressed to a certain slave or send as broadcast message to all slaves. For identifying a broadcast message, the slave address 0 is set.

Only write commands may be sent as broadcast.

ASCII-, RTU-Modus

Modbus supports two different transmission modes:

- ASCII mode: Every Byte is transferred in 2-character ASCII code. A start and an end ID mark the data. This enables high control at the transmission but needs time.
- RTU mode: Every Byte is transferred as character. Thus enables a higher data throughput than the ASCII mode. Instead of start and end ID, RTU uses a time watcher.

The mode selection is at parameterization.

Modbus at the CP 240 from VIPA

The CP 240 Modbus supports several operating modes that are described in the following:

Modbus Master

In *Modbus Master* operation you control the communication via your PLC user application. For this the SEND and RECEIVE handling blocks are required. By using a blockage you here have the option to transfer up to 250Byte user data.

Modbus Slave Short

In *Modbus Slave Short* operation the CP 240 occupies each 16Byte for in- and output data at arbitrary area in the CPU. Via the address parameter you may define this area during the hardware configuration. A PLC program for the data provision is at the slave side not required. This operation mode is especially convenient for the fast transfer of small data amounts via Modbus.

Modbus Slave Long

For data that exceeds the length of 16Byte you should use the operation mode Modbus Slave Long. Here the master transfers at data reception via RECEIVE the area to the CPU where a change has happened. The date transfer happens following this principle:

The reception area of max. 1024Byte is separated into 128 8Byte blocks. At data change by the master only those blocks are transferred to the CPU where changes occurred. During one block cycle of the RECEIVE block up to 16 coherent 8Byte block may be handled on at the back plane bus. If the 8Byte blocks are not coherent, every changes 8Byte block requires one block cycle. The receive DB of the RECEIVE block must always be set as a multiple of 8.

By means of a SEND call a wanted data area is transferred to the CP that may be read by the master. Writing master accesses must not lie outside of the reception area!

Please regard that Modbus Slave Long is supported starting with the block library FX000002_V120 or higher.

Note!

The CP 240 only reports a respond telegram to the master after all data has been received.

Commissioning

After switching on the voltage supply the LEDs ER, TxD and RxD are flashing at the Modbus module. Thus the module signalizes that it hasn't received valid parameters from the CPU yet. As soon as you switch the CPU to RUN, the Modbus parameters are transferred to the module. With valid parameters the LEDs ER, TxD and RxD extinguish. Now the Modbus module is ready for communication.

At deployment in master mode you may now execute according write/read commands in your user application.

If the ER-LED is not extinguishing, an internal error has happened. At a transient error you may set this back by means of a STOP-RUN switch of the CPU.

Modbus - Parameterization

Parameter structure at Modbus

Byte	Function	Range	Default parameter
0	Baud rate	0h: 9600Baud 6h: 2400Baud 7h: 4800Baud 9h: 9600Baud Ah: 14400Baud Bh: 19200Baud Ch: 38400Baud	0h: 9600Baud
1	Protocol	0Ah: Modbus master ASCII short 0Bh: Modbus master RTU short 0Ch: Modbus slave ASCII short 0Dh: Modbus slave RTU short 1Ch: Modbus slave ASCII long 1Dh: Modbus slave RTU long	Bh: Modbus master RTU
2	Bit 1/0 Data bits	00b: 5 Data bits 01b: 6 Data bits 10b: 7 Data bits 11b: 8 Data bits	11b: 8 Data bits
	Bit 3/2 Parity	00b: none 01b: odd 11b: even	00b: none
	Bit 5/4 Stop bits	01b: 1 10b: 1.5 11b: 2	01b: 1 Stop bit
	Bit 7/6	reserved	00b: none
3	reserved	0	0
4	Address	1...255	1
5	Debug	0: Debug off 1: Debug on	0
6...7	Wait period	0: automatic calculation 1 ... 60000: Time in ms	0
8	reserved		
9	reserved		
10	reserved		
11	reserved		
12	reserved		
13	reserved		
14	reserved		
15	reserved		

Note to default parameter!

If no parameterization is present and the CP 240 is linked-up via auto addressing, the CP has the following default parameters:

Baud rate: 9600Baud, Protocol: ASCII, data bits: 8, **Parity: even**, Stop bits: 1, Flow control: no, ZNA: 0, ZVZ: 200ms, Receive buffer: 1

Parameter description

Baud rate

The data communication rate in bit/s (Baud). You may select one of the following values:

00h: Default (9600Baud)
06h: 2400Baud
07h: 4800Baud
09h: 9600Baud
0Ah: 14400Baud
0Bh: 19200Baud
0Ch: 38400Baud

Default: 0 (9600Baud)

Protocol

The protocol to be used. This setting determines the further structure of the parameter data.

0Ah: Modbus master with ASCII
0Bh: Modbus master with RTU
0Ch: Modbus slave short with ASCII
0Dh: Modbus slave short with RTU
1Ch: Modbus slave long with ASCII
1Dh: Modbus slave long with RTU

Transfer parameter byte

For every character frame there are 3 data formats available. The data formats are different in the number of data bits, with or without parity bit and number of stop bits.

The transfer parameter byte has the following structure:

Byte	Function	Range	Default parameter
2	Bit 1/0 Data bits	00b: 5 Data bits 01b: 6 Data bits 10b: 7 Data bits 11b: 8 Data bits	11b: 8 Data bits
	Bit 3/2 Parity	00b: none 01b: odd 10b: even 11b: even	00b: none
	Bit 5/4 Stop bits	01b: 1 10b: 1,5 11b: 2	01b: 1 Stop bit
	Bit 7/6	reserved	-

Data bits — Number of *data bits* that represent a character.

Parity — The parity is depending on the value even or odd. For the purposes of the parity check, the information bits are expanded by the parity bit. The value of the parity bit ("0" or "1") completes the value of all the bits to obtain a pre-arranged state. If the parity was not specified, the parity bit is set to "1" but it is not included in the assessment.

Stop bits — The stop bits are appended to each character and signify the end of the character.

Default: 13h (Data bits: 8, Parity: none, Stop bit: 1)

Address — Set here in slave mode the Modbus slave address.
Range: 1 ... 255 *Default: 1*

Debug — This mode is for internal tests. This function should always be de-activated.
Range: 0, 1 *Default: 0*

Delay time — In master mode here has to be set a delay time in ms. With "0" the delay time is evaluated automatically depending on the protocol with the following formula:

Modbus ASCII: $50ms + \frac{2926000ms}{Baudrate} \cdot s$ with baud rate in Bit/s

Modbus RTU: $50ms + \frac{5190000ms}{Baudrate} \cdot s$ with baud rate in Bit/s

In slave mode this parameter is ignored.

Modbus - Deployment

Overview

You may deploy the CP 240 Modbus either in master or in slave mode. At both modes the module occupies each 16Byte for in- and output data at arbitrary area in the CPU.

For the deployment with Modbus a hardware configuration must always be executed.

Requirements for operation

The following components are required for the deployment of the System 200V Modbus modules:

- Each 1 System 200V consisting of CPU 21x and CP 240
- Siemens SIMATIC Manager
- Programming cable for MPI coupling (e.g. Green Cable from VIPA)
- GSD-file **VIPA_21x.gsd** (V1.67 or higher)
- VIPA handling blocks Fx000002_V120.zip or higher
- Serial connection between both CP 240

Parameterization

The CP 240 always requires a hardware configuration. For this the inclusion of the VIPA_21x.gsd into the hardware catalog is necessary. The parameterization has the following approach:

- Start the Siemens SIMATIC Manager
- Install the GSD-file **VIPA_21x.gsd** in the hardware catalog.
- Create a virtual Profibus system in the hardware configurator with the CPU 315-2DP (6ES7 315-2AF03 V1.2).
- Add to this system the slave system "VIPA_CPU21x" and assign the Profibus address 1.
- Configure your System 200V starting with the CPU 21x. Use a CP labeled with "Modbus".
- Parameterize the CP 240 after your needs. The CP 240 occupies each 16Byte in the CPU for in- and output.
- Transfer your project to the PLC.

PLC application

Except of the "Modbus Slave Short", the communication always requires a PLC application. For this the communication happens via handling blocks that you may include into the Siemens SIMATIC Manager by means of the VIPA library **Fx000002_V120.zip** or higher. The library is available on the VIPA-CD "ToolDemo" or at ftp.vipa.de.

Note!

More detailed information about the installation of the GSD-file and the library is to be found in the chapter "Project engineering".

Communication options

The following text describes the communication options between Modbus master and Modbus slave with the following combination options:

- CP 240 Modbus Master ↔ CP 240 Modbus Slave Short
- CP 240 Modbus Master ↔ CP 240 Modbus Long

Master ↔ Slave Short

Modbus Master

The communication in master mode happens via data blocks deploying the CP 240 SEND-RECEIVE handling blocks. With the usage of a blockage you may transfer up to 250Byte user data.

Modbus Slave Short

The *Modbus Slave Short* mode limits the amount of user data for in- and output to 16Byte. For this you only need a hardware configuration at the slave section.

Approach

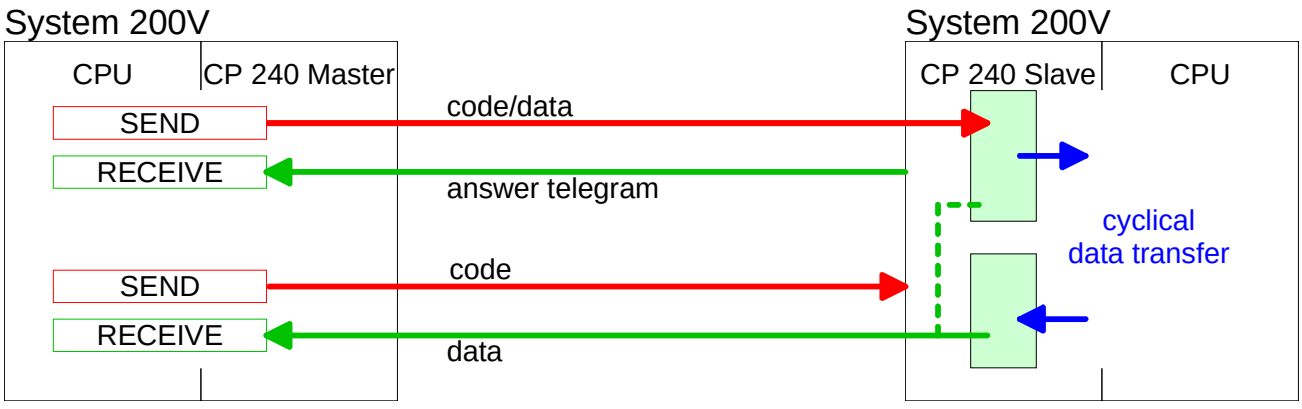
- Build-up each one System 200V for the master and the slave consisting of a CPU 21x and a CP 240 for each and connect both systems via the serial interface.
- Configure the master section.

  The parameterization of the CP 240 as Modbus master happens via the hardware configuration. Additionally you need a PLC user application for the communication, build-up with the following structure:

  OB 1: Call of FC0 (SEND) with error evaluation. For this the telegram has to be stored in the send block according to Modbus rules.

  Call of FC1 (RECEIVE) with error evaluation. The data is stored in the receive block according to Modbus rules.
- Configure the slave section.

  The parameterization of the CP 240 happens via the hardware configuration. Set here the start address for in- and output area from where on the fix amount of each 16Byte for in- and output are stored in the CPU at arbitrary place.



Master → Slave Long

Modbus Master

The communication in master mode happens via data blocks deploying the CP 240 SEND-RECEIVE handling blocks. With the usage of a blockage you may transfer up to 250Byte user data.

Modbus Slave Long

In the *Modbus Slave Long* mode only a changed data area is transferred to the CPU via RECEIVE starting with 0. If the master requests data it has to be made sure that the relevant data are present in the CP. With a SEND call a wanted data area in transferred to the CP starting with 0 .

Approach

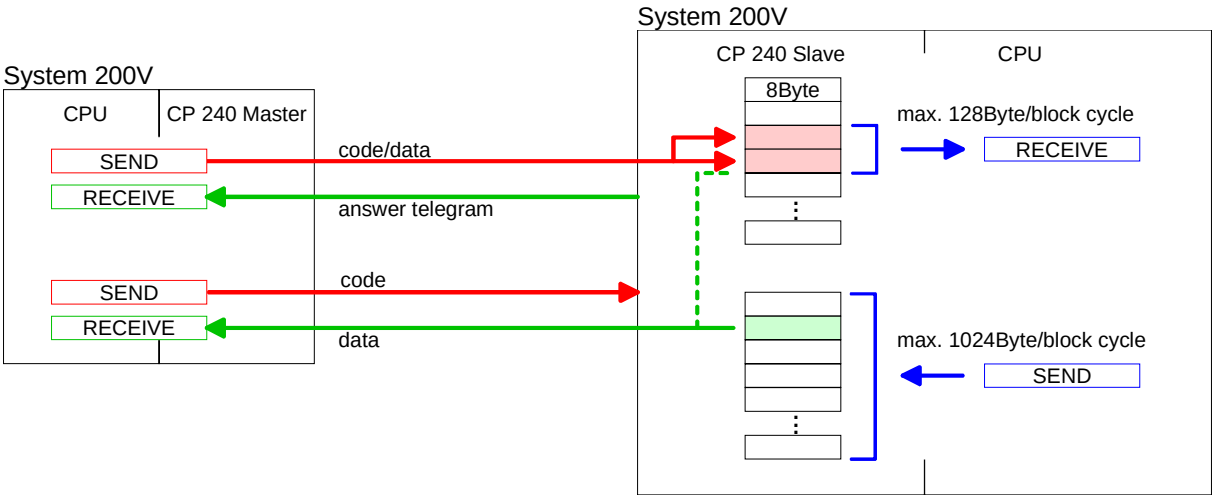
- Build-up each one System 200V for the master and the slave consisting of a CPU 21x and a CP 240 for each and connect both systems via the serial interface.
- Configure the master section.

  The project engineering of the master section happens like shown in the sample above.
- Configure the slave section.

  The parameterization of the CP 240 as Modbus slave happens via the hardware configuration. Additionally you need a PLC user application for the communication, build-up with the following structure:

  OB 1: Call of FC0 (SEND) with error. For this an area starting at 0 is stored in the CP 240 where the master may gain access via Modbus.

  The FC1 (RECEIVE) with error evaluation allows you to transfer a data area into the CPU. The reception area with a max. size of 1024Byte is separated into 128 8Byte blocks. At a data change by the master, only those blocks are transferred to the CPU where changes occurred. During one block cycle of the RECEIVE block a maximum of 16 coherent 8Byte blocks may be handled on at the back plane bus. If the 8Byte blocks are not coherent every changed 8Byte block requires one block cycle. At call of the RECEIVE block, the receive DB has always to be set as a multiple of 8. Writing master accesses may not be outside of the receive area!



Access to multiple slaves

At deployment of multiple slaves with RS485 there cannot occur bus conflict errors because the master may only communicate with one slave at a time. The master sends a command telegram to the save specified via the address and waits for a certain time where within the slave may send its respond telegram. During the latency communication with another slave is not possible.

For the communication with multiple slaves every slave needs a SEND data block for the command telegram and a RECEIVE data block for the respond telegram.

An application with several slaves would be consisting of an according amount of data blocks with commands.

These are executed in sequence:

1. slave: Send command telegram to slave address 1.slave
   Receive respond telegram from slave address 1.slave
   Interpret respond telegram

2. slave: Send command telegram to slave address 2.slave
   Receive respond telegram from slave address 2.slave
   Interpret respond telegram

... etc.

A request may be send to on specified slave or as broadcast message to all slaves. To mark a broadcast message the slave address is set to 0.

Only write commands may be sent as broadcast.

Note!

After a broadcast the master is not waiting for a respond telegram.

Write to master output area

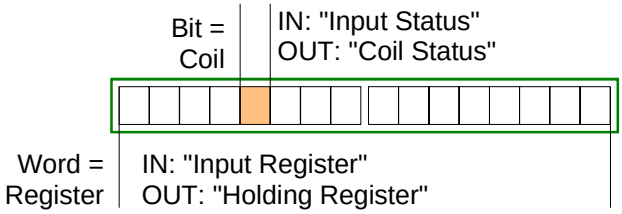
If you "OR" the FC 0 parameter ANZ with 4000h the slave data to send were not transferred to the master input area but to the master output area.

Since this area can be read by the master by means of function codes this functionality can be used for example for the direct error transmission to the master.

Modbus - Function codes

Naming convention

Modbus has some naming conventions:



- Modbus differentiates between bit and word access;
  Bits = "Coils" and Words = "Register".
- Bit inputs are referred to as "Input-Status" and Bit outputs as "Coil-Status".
- Word inputs are referred to as "Input-Register" and Word outputs as "Holding-Register".

Range definitions

Normally the access at Modbus happens by means of the ranges 0x, 1x, 3x and 4x.

0x and 1x gives you access to *digital* Bit areas and 3x and 4x to *analog* word areas.

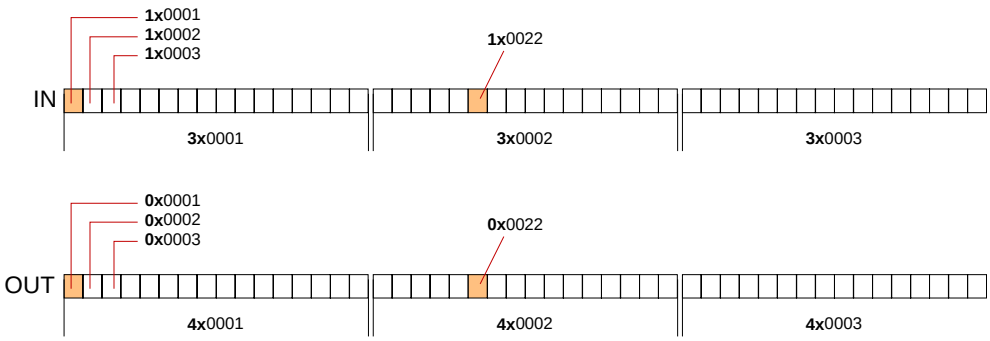
For the CP 240 from VIPA is not differentiating digital and analog data, the following assignment is valid:

0x: Bit area for master output data
Access via function code 01h, 05h, 0Fh

1x: Bit area for master input data
Access via function code 02h

3x: Word area for master input data
Access via function code 04h

4x: Word area for master output data
Access via function code 03h, 06h, 10h



A description of the function codes follows below.

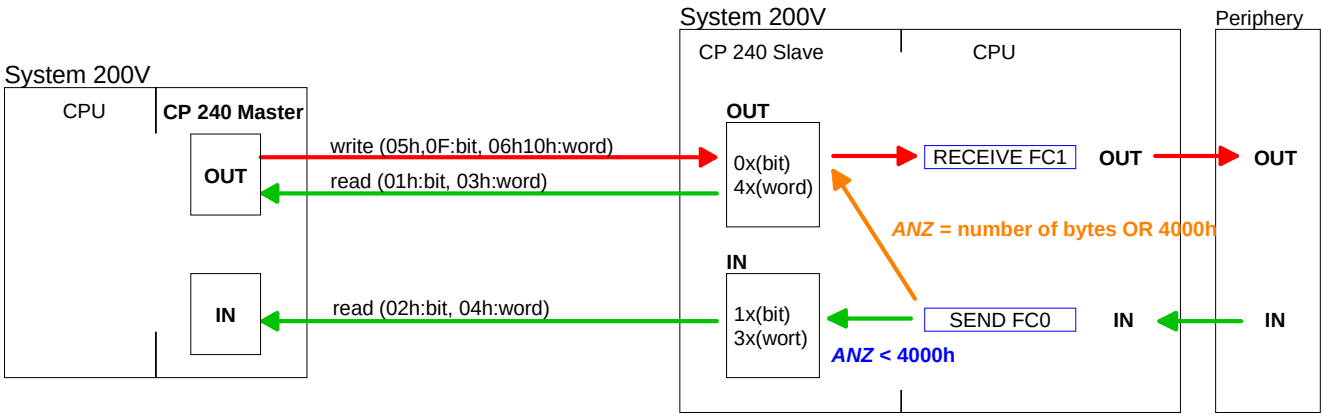
Overview

With the following Modbus function codes a Modbus master can access a Modbus slave: With the following Modbus function codes a Modbus master can access a Modbus slave. The description always takes place from the point of view of the master:

Code	Command	Description
01h	Read n Bits	Read n Bits of master output area 0x
02h	Read n Bits	Read n Bits of master input area 1x
03h	Read n Words	Read n Words of master output area 4x
04h	Read n Words	Read n Words master input area 3x
05h	Write 1 Bit	Write 1 Bit to master output area 0x
06h	Write 1 Word	Write 1 Word to master output area 4x
0Fh	Write n Bits	Write n Bits to master output area 0x
10h	Write n Words	Write n Words to master output area 4x

Point of View of "Input" and "Output" data

The description always takes place from the point of view of the master. Here data, which were sent from master to slave, up to their target are designated as "output" data (OUT) and contrary slave data received by the master were designated as "input" data (IN).



Respond of the slave

If the slave announces an error, the function code is send back with an "ORed" 80h. Without an error, the function code is sent back.

Coupler answer: Function code OR 80h → Error
Function code → OK

Byte sequence in a Word

For the Byte sequence in a Word is always valid:

1 Word

High Byte	Low Byte

Check sum CRC, RTU, LRC

The shown check sums CRC at RTU and LRC at ASCII mode are automatically added to every telegram. They are not shown in the data block.

Read n Bits 01h, 02h

Code 01h: Read n Bits of master output area 0x
Code 02h: Read n Bits of master input area 1x

Command telegram

Slave address	Function code	Address 1. Bit	Number of Bits	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Respond telegram

Slave address	Function code	Number of read Bytes	Data 1. Byte	Data 2. Byte	...	Check sum CRC/LRC
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte max. 250 Byte		1 Word

Read n Words 03h, 04h

03h: Read n Words of master output area 4x
04h: Read n Words master input area 3x

Command telegram

Slave address	Function code	Address 1. Bit	Number of Words	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Respond telegram

Slave address	Function code	Number of read Bytes	Data 1. word	Data 2. word	...	Check sum CRC/LRC
1 Byte	1 Byte	1 Byte	1 Word	1 Word max. 125 Words		1 Word

Write 1 Bit 05h

Code 05h: Write 1 Bit to master output area 0x
A status change is via "Status Bit" with following values:

"Status Bit" = 0000h → Bit = 0
"Status Bit" = FF00h → Bit = 1

Command telegram

Slave address	Function code	Address Bit	Status Bit	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Respond telegram

Slave address	Function code	Address Bit	Status Bit	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Write 1 Word 06h

Code 06h: Write 1 Word to master output area 4x

Command telegram

Slave address	Function code	Address word	Value word	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Respond telegram

Slave address	Function code	Address word	Value word	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Write n Bits 0Fh

Code 0Fh: Write n Bits to master output area 0x

Please regard that the number of Bits has additionally to be set in Byte.

Command telegram

Slave address	Function code	Address 1. Bit	Number of Bits	Number of Bytes	Data 1. Byte	Data 2. Byte	...	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Byte	1 Byte	1 Byte	1 Byte	1 Word
					max. 250 Byte			

Respond telegram

Slave address	Function code	Address 1. Bit	Number of Bits	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Write n Words 10h

Code 10h: Write n Words to master output area 4x

Command telegram

Slave address	Function code	Address 1st word	Number of words	Number of Bytes	Data 1. word	Data 2. word	...	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Byte	1 Word	1 Word	1 Word	1 Word
					max. 125 Words			

Respond telegram

Slave address	Function code	Address 1. word	Number of words	Check sum CRC/LRC
1 Byte	1 Byte	1 Word	1 Word	1 Word

Modbus - Error messages

Overview

At the communication at Modbus there are 2 error types:

- Master doesn't receive valid data
- Slave responds with error message

Master doesn't receive valid data

If the slave doesn't answer within the specified delay time or if a telegram is defective, the master enters an error message into the receive block in plain text.

The following error messages may occur:

ERROR01 NO DATA	*Error no data* No telegram arrived within the specified delay time.
ERROR02 D LOST	*Error data lost* No data is available because either the receive buffer is full or an error occurred in the receive section.
ERROR03 F OVERF	*Error frame overflow* The telegram end wasn't recognized or maximum telegram length exceeded.
ERROR04 F INCOM	*Error frame incomplete* Only a part telegram has been received.
ERROR05 F FAULT	*Error frame Fault* The check sum of the telegram is faulty.
ERROR06 F START	*Error frame start* The start bit it wrong. this error may only occur with Modbus-ASCII.

Slave answers with error message

If the slave answers with an error, the function code is sent back like shown below, marked as "or" with 80h:

DB11.DBD 0	**DW#16#05900000**	**Respond telegram**
	with 05 →	Slave address 05h
	90 →	Function code 90h (error message, for 10h OR 80h = 90h)
	0000 →	The rest data is not relevant, for an error has been sent.

Modbus - Example

Overview

In the following example a communication between a master and a slave via Modbus is build-up. Furthermore the example shows how you can easily control the communication processes by means of the handling blocks.

At need you may receive the example project from VIPA.

Requirements

The following components are required for the example:

- 2 System 200V with CPU 21x and CP 240
- Programming cable for MPI connection (e.g. Green Cable from VIPA)
- Serial cable to connect both CP 240

Approach

- Assemble a Modbus system, existing of master system, slave system and connect them with the serial cable.
- Engineer the master side! For this open the sample project using your configuration tool. Adjust the transfer parameters accordingly.
  Select "Modbus Master RTU" at *Protocol*.
  Edit the OB1 and coordinate the module addresses with the addresses of the parameterization.
  Transfer your project into the master CPU 21x.
- Engineer the slave side. For this you open the sample project using your configuration tool. Adjust the parameters of the CP 240 accordingly. Select "Modbus Slave RTU" at *Protocol*. Type a slave address in *Address*.
  For the communication with Modbus, the slave doesn't need a PLC application, because the master transfers source and destination.

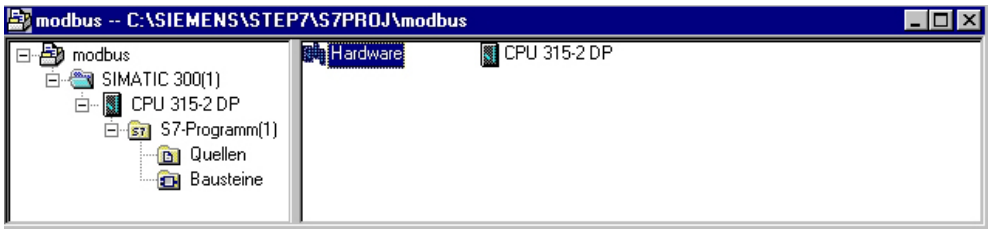
De-archive project

To de-archive your project into the configuration tool follow the described approach:

- Start the Siemens SIMATIC Manager.
- To extract the file Modbus.zip click on **File** > *de-archive*.
- Select the example file Modbus.zip and choose as destination directory "s7proj".
- After the extraction open the project.

Project structure

The project has the following structure:



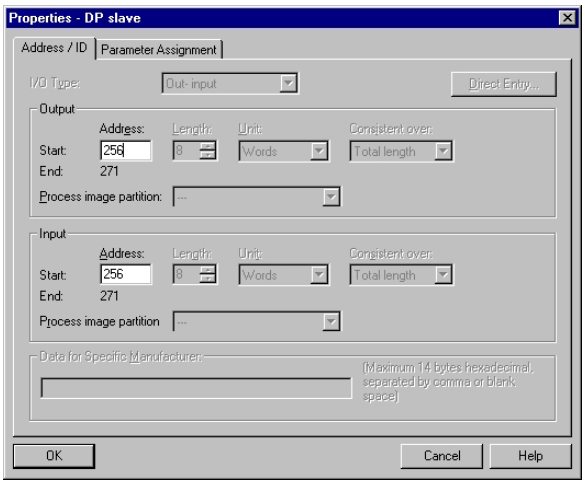
Master project engineering

The sample already contains the PLC program and the parameters for the Modbus master. you only need to adjust the Modbus parameters.

Parameterization

For this start the hardware configurator and choose the module 240-1CA20. Via double-click you reach the parameterization:

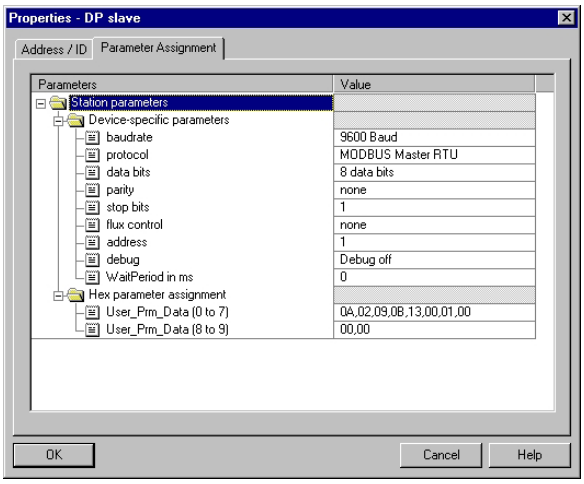
Dialog for address entry



Here you may set from which address on the 16Byte for in- and output are stored in the CPU.

Please regard that you have to change the addresses that you are changing here also in the SEND and RECEIVE blocks.

Dialog for Modbus parameters



In this section of the parameterization you set the Modbus parameters.

The following parameters must be identical for all bus participants: baud rate, data Bits, parity, stop bits and flow control.

Set “Modbus Master RTU” in *Protocol*.

The setting of an address is only required at slave side.

At a master parameterization the address is ignored.

PLC program

The wanted Modbus commands are set via your PLC program. In the present sample the deployment of SEND and RECEIVE in the OB1 is shown.

```
OB 1:


    CALL     FC       0                    //"SEND"
      ADR           :=256                  //Start address of the module
      _DB           :=DB10                 //In this block you create
                                           //the telegram you want to send
      ABD           :=W#16#0               //Starting with this Byte-Offset
                                           //the telegram starts in the _DB
      ANZ           :=MW12                 //Telegram length (length to send) in Byte
      PAFE          :=MB14                 //Error byte
      FRG           :=M1.0                 //Send init (1=init, back to 0
                                           //when send ready)
      GESE          :=MW16                 //required internal
      ANZ_INT       :=MW18                 //required internal
      ENDE_KOM      :=M2.0                 //required internal
      LETZTER_BLOCK:=M2.1                  //required internal
      SENDEN_LAEUFT:=M2.2                  //required internal
      FEHLER_KOM    :=M2.3                 //required internal

    CALL     FC       1                    //"RECEIVE"
      ADR           :=256                  //Input address of the module
      _DB           :=DB11                 //In this data block the
                                           //received telegram is stored
      ABD           :=W#16#0               //Starting with this Byte-Offset the tel. starts in _DB
      ANZ           :=MW22                 //Telegram length (received length) in Byte
      EMFR          :=M1.1                 //Reception status (1=Telegram fully received)
      PAFE          :=MB34                 //Error byte
      GEEM          :=MW36                 //required internal
      ANZ_INT       :=MW38                 //required internal
      EMPF_LAEUFT   :=M3.0                 //required internal
      LETZTER_BLOCK:=M3.1                  //required internal
      FEHL_EMPF     :=M3.2                 //required internal

     U     M      1.1                      //as long as reception status=1 no new
                                           //telegram is entered
     R     M      1.1                      //for this the reception status must be acknowledged
                                           //with 0
```

If necessary also adjust the addresses that the CP occupies in the CPU to the addresses of your parameterization and transfer the hardware configuration to your CPU 21x of the master system.

Note!

Due to the transfer of the data in blocks of 8Byte you have to make sure that the length of the reception data area is a multiple of 8. As well the writing accesses of the master should not be outside of the reception area otherwise the RECEIVE block announces a range error.

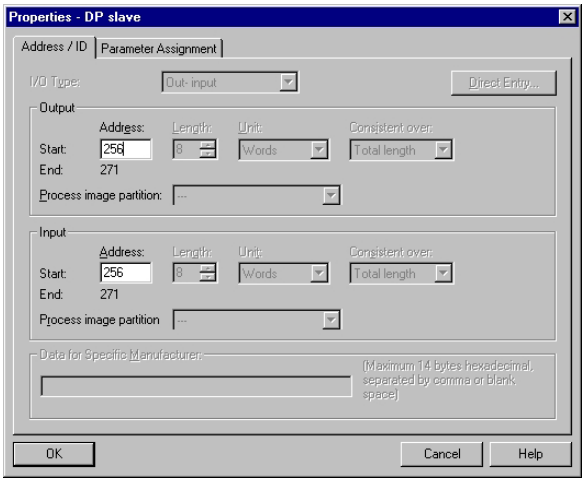
Slave project engineering

For the project engineering of the slave you only have to adjust the Modbus parameters. A PLC application is not required for the source and destination data are delivered in the master telegram.

Parameterization

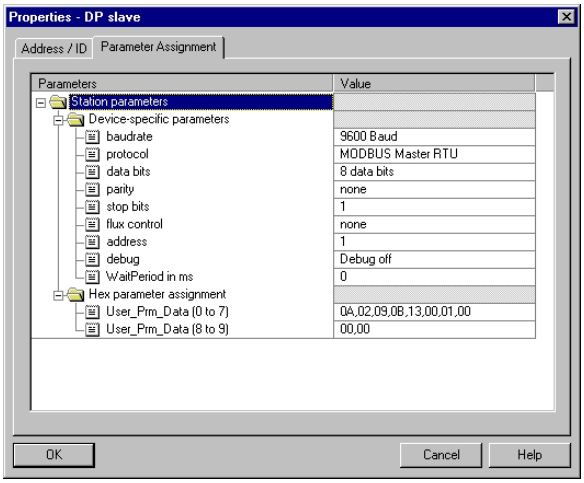
For the parameterization of the slave module open the sample project in your hardware configurator. Select the module 240-1CA20. Via double-click you reach the parameterization.

Dialog for address entry



Here you may set from which address on the 16Byte for in- and output are stored in the CPU.

Dialog for Modbus parameters



In this section of the parameterization you set the Modbus parameters.

The following parameters must be identical for all bus participants: baud rate, data bits, parity, stop bits and flow control.

Enter a valid Modbus address for the slave into *Address*.

Transfer the parameterization into the CPU of the slave system.

Send and receive telegram

Open the variable table **Tabelle1** of the example project and switch to online mode.

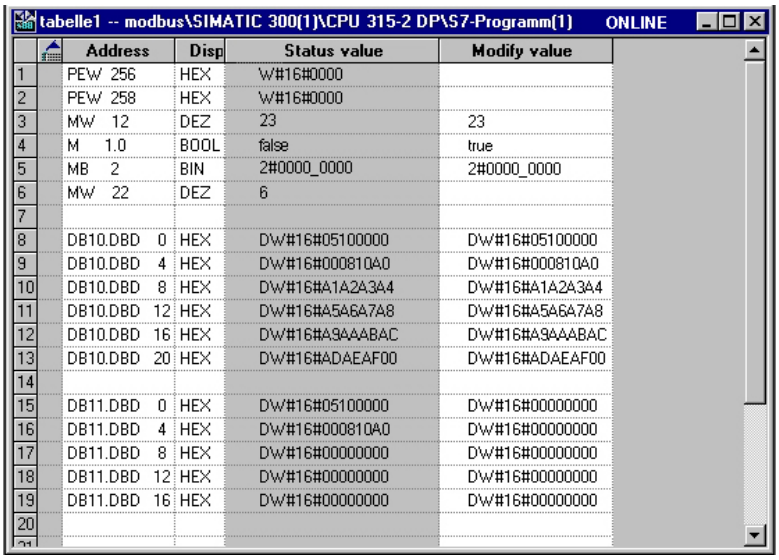


tabelle1 -- modbus\SIMATIC 300(1)\CPU 315-2 DP\S7-Programm(1) ONLINE

	Address	Disp	Status value	Modify value
1	PEW 256	HEX	W#16#0000	
2	PEW 258	HEX	W#16#0000	
3	MW 12	DEZ	23	23
4	M 1.0	BOOL	false	true
5	MB 2	BIN	2#0000_0000	2#0000_0000
6	MW 22	DEZ	6	
7				
8	DB10.DBD 0	HEX	DW#16#05100000	DW#16#05100000
9	DB10.DBD 4	HEX	DW#16#000810A0	DW#16#000810A0
10	DB10.DBD 8	HEX	DW#16#A1A2A3A4	DW#16#A1A2A3A4
11	DB10.DBD 12	HEX	DW#16#A5A6A7A8	DW#16#A5A6A7A8
12	DB10.DBD 16	HEX	DW#16#A9AAABAC	DW#16#A9AAABAC
13	DB10.DBD 20	HEX	DW#16#ADAEAF00	DW#16#ADAEAF00
14				
15	DB11.DBD 0	HEX	DW#16#05100000	DW#16#00000000
16	DB11.DBD 4	HEX	DW#16#000810A0	DW#16#00000000
17	DB11.DBD 8	HEX	DW#16#00000000	DW#16#00000000
18	DB11.DBD 12	HEX	DW#16#00000000	DW#16#00000000
19	DB11.DBD 16	HEX	DW#16#00000000	DW#16#00000000
20				

Send block DB10

DB10.DBD 0	**DW#16#05100000** with 05 → 10 → 0000 →	**Command telegram** Slave address 05h Function code 10h (write n Words) Offset 0000h
DB10.DBD 4	**DW#16#000810A0** with 0008 → 10 → A0 →	**Command telegram + 1 data byte** Word count 0008h Byte count 10h Start 16 byte data with A0h
DB10.DBD 8	**DW#16#A1A2A3A4**	**Data byte 2 ... 5**
DB10.DBD 12	**DW#16#A5A6A7A8**	**Data byte 6 ... 9**
DB10.DBD 16	**DW#16#A9AAABAC**	**Data byte 10 ... 13**
DB10.DBD 20	**DW#16#ADAEAF00** with ADAEAF → 00 →	**Data byte 14 ... 16 + 1 byte not used** Data byte 14 ... 16 not used by the module

Receive block DB11

DB11.DBD 0	**DW#16#05100000** with 05 → 10 → 0000 →	**Response telegram** Slave address 05h Function code 10h (no error) Offset 0000h
DB11.DBD 4	**DW#16#000810A0** with 0008 → 10 → A0 →	**Response telegram + 1 data byte** Word count 0008h Byte count 10h Start 16 byte data with A0h (irrelevant at write command)
DB11.DBD 8	**DW#16#00000000**	**Data byte 2 ... 5**
DB11.DBD 12	**DW#16#00000000**	**Data byte 6 ... 9**
DB11.DBD 16	**DW#16#00000000**	**Data byte 10 ... 13**
DB11.DBD 20	**DW#16#00000000** with 000000 → 00 →	**Data byte 14 ... 16 + 1 byte not used** Data byte 14 ... 16 not used by the module

Receive block with error response

Slave does not answer to the master command

If the slave does not respond within the specified timeout time, the master enters the following error message into the receive block:

ERROR01 NO DATA. The Hex format has the following values:

DB11.DBD 0	**DW#16#4552524F**	**Respond telegram**
	with 45 →	E
	52 →	R
	52 →	R
	4F →	O
DB11.DBD 4	**DW#16#52000120**	**Respond telegram**
	with 52 →	R
	0001 →	0001h:1 (as Word)
	20 →	" "
DB11.DBD 8	**DW#16#4E4F2044**	**Respond telegram**
	with 4E →	N
	4F →	O
	20 →	" "
	44 →	D
DB11.DBD 12	**DW#16#41544100**	**Respond telegram**
	with 41 →	A
	54 →	T
	41 →	A
	00 →	00h: (zero termination)

.
.
.

Slave responds with error message

If the slave responds with an error, the function code is sent back "ORed" with 80h.

DB11.DBD	**DW#16#05900000**	**Respond telegram**
	with 05 →	Slave address 05h
	90 →	Function code 90h (error message since 10h OR 80h = 90h)
	0000 →	Residual data are irrelevant since error returned.

Technical data

CP 240 RS232

Electrical Data	VIPA 240-1BA20
Number of channels	1
Voltage supply	5V via back plane bus
Current consumption	max. 150mA
Status monitor	via LED at the front side
Protocols	ASCII, ASCII fragmented, STX/ETX, 3964(R), 3964(R) with RK512, Modbus (Master, Slave)
Plugs / Interfaces	9pin D-type plug for RS232
RS232 signals	TxD, RxD, RTS, CTS, DTR, DSR, RI, CD, GND
Potential separation	to back plane bus
Transfer size	max. 15m
Baud rate	max. 115.2kbit/s
Programming data	
Input data	16Byte
Output data	16Byte
Parameter data	16Byte
Dimensions and Weight	
Dimensions (WxHxD) in mm	25.4x76x78
Weight	80g

CP 240 RS485

Electrical Data	VIPA 240-1CA20
Number of channels	1
Voltage supply	5V via back plane bus
Current consumption	max. 150mA
Status monitor	via LED at the front side
Protocols	ASCII, ASCII-fragmented, STX/ETX, 3964(R), 3964(R) with RK512, Modbus (Master, Slave)
Plugs / Interfaces	9pin D-type plug for RS485
RS232 signals	RxD/TxD-P, RxD/TxD-N, RTS, M5V, P5V
Potential separation	to back plane bus
Transfer size	max. 1200m
Baud rate	150 ... 115.2kbit/s
Programming data	
Input data	16Byte
Output data	16Byte
Parameter data	16Byte
Dimensions and Weight	
Dimensions (WxHxD) in mm	25.4x76x78
Weight	80g

CP 240 RS422/485

Electrical Data	VIPA 240-1CA21
Number of channels	1
Voltage supply	5V via back plane bus
Current consumption	max. 150mA
Status monitor	via LED at the front side
Protocols	ASCII, ASCII-fragmented, STX/ETX, 3964(R), 3964(R) with RK512, Modbus (Master, Slave)
Plugs / Interfaces	9pin D-type plug for RS422/485
RS485 signals	RxD/TxD-N (A), RxD/TxD-P (B)
RS422 signals	TxD-N (A), RxD-N (A), TxD-P (B), RxD-P (B)
Potential separation	to back plane bus
Transfer size	max. 1200m
Baud rate	150 ... 115.2kbit/s
Programming data	
Input data	16Byte
Output data	16Byte
Parameter data	16Byte
Dimensions and Weight	
Dimensions (WxHxD) in mm	25.4x76x78
Weight	80g

Technical Data
Protocols

ASCII	
Telegram length	max. 1024Byte
Baud rate	150, 300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 76800, 115200
Character delay time ZVZ	0 ... 5.1s in 20ms steps
Flow control	none, hardware, XON/XOFF
Number of telegrams to buffer	max. 250
End recognition of a telegram	after character delay time ZVZ
STX/ETX	
Telegram length	max. 1024Byte
Baud rate	150, 300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 76800, 115200
Character delay time TMO	20ms ... 5.1s in 20ms steps
Flow control	none, hardware, XON/XOFF
Number of telegrams to buffer	max. 250
End recognition of a telegram	after character delay time TMO
Number of start characters	0 ... 2 (characters parameterizable)
Number of end characters	0 ... 2 (characters parameterizable)
3964, 3964R	
Telegram length	max. 1024Byte
Baud rate	150, 300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 76800, 115200
Block proof sign	only 3964R
Priority	low/high
Character delay time ZVZ	0 ... 5.1s in 20ms steps
Acknowledgment delay time QVZ	0 ... 25.5s in 20ms steps
Number of connection attempts	0 ... 255
Number of transfer attempts	1 ... 255
Computer coupling RK512	
Telegram length	max. 1024Byte
Baud rate	150, 300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 76800, 115200
Character delay time ZVZ	0 ... 5.1s in 20ms steps
Acknowledgment delay time QVZ (User)	0 ... 25.5s in 100ms steps
Number of connection attempts	0 ... 255
Number of transfer attempts	1 ... 255
Modbus	
Telegram length	max. 258Byte
Addressable range	each 1024Byte
Baud rate	2400, 4800, 7200, 9600, 14400, 19200, 38400
Mode	Master ASCII, Master RTU, Slave ASCII short, Slave RTU short, Slave ASCII long, Slave, RTU long
Address	1 ... 255
Delay time	automatically, 1 ... 60000ms

Chapter 5 CP 240 - EnOcean

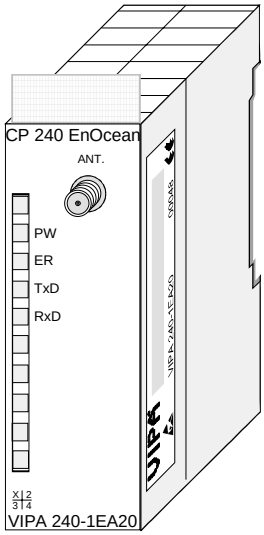
Overview

This chapter contains information about the structure and the inclusion of the communication processor CP 240 with EnOcean transceiver module.

Content

System overview

CP 240 EnOcean



Order data

Type	Order number	Description
CP 240 EnOcean	VIPA 240-1EA20	CP with EnOcean radio transceiver module TCM 120
Portable antenna	VIPA 240-0EA00	Portable antenna with SMA plug
Magnetic socket antenna	VIPA 240-0EA10	Magnetic socket antenna with 150cm cable and SMA plug

Basics

EnOcean

EnOcean is a battery free radio system that has been developed by the company EnOcean in 2001. Due to the short signal length of 0.5ms and 10mW transmitting power the radio system technique has an energy requirement of 50µWs. For this the system uses the energy of smallest changes of pressure and temperature as power supply for the sensors.

The reach of the sensors is up to 300m out of doors. Additionally, every transmitter gets an unique 32Bit address as ID during manufacturing. The modules are using the internationally accredited SRD frequency band an 869 MHz.

Main points of usage of EnOcean are building automation, industrial production and automotive.

Properties

- Minimal energy requirements
- Support of several transmitters in the immediate environs
- Telegram length 0.5ms
- Transfer reach up to 300m
- Mono- and bi-directional communication
- Easy extensibility

Amplitude modulation

As modulation procedure EnOcean uses the incoherent amplitude - modulation (ASK). The error likeliness is nearly the same compared with the frequency modulation at identical interference signal level. The digital amplitude modulation allows the realization of energy saving transmitters because here only the "1"-Bits are transferred.

Security by means of telegram repetition

The transfer of a data telegram takes about 0.5ms. To enhance data security every telegram is repeated two times within 40ms, whereat the time lag between every repetition is perchancely.

This fast multiple sending allows that many neighborhood transmitters may be working parallel together on one radio frequency with a low error ratio.

IDs for addressing

EnOcean uses IDs for the addressing. An ID is an compound of *ID base* and a freely configurable *bit area*. Since the EnOcean modules are delivered by VIPA with a different *ID base* with extensive projects it is recommended to note all *ID base* of the modules. So on error an module can be replaced and the appropriate *ID base* can be taken.

For this details can be found at "Exchange module and set ID base".

Fast introduction

Overview

The communication processor CP 240 EnOcean enables the process coupling to different destination or source systems based upon the wireless EnOcean communication.

The CP 240 EnOcean is supplied with voltage via the back plane bus. For the internal communication the VIPA FCs are used. For the project engineering of the CP 240 EnOcean together with a CPU 21x in the Siemens SIMATIC Manager, the inclusion of the GSD VIPA_21x.gsd is required. To enable the CP 240 EnOcean to communicate with the CPU, a hardware configuration for the system is always necessary.

A general description for the project engineering of the CP 240 is to be found in the chapter "Project engineering".

Parameters

By placing the CP 240 EnOcean in the hardware configuration into the "virtual" Profibus system, the required parameters are automatically created. The parameter area has the following structure:

Byte	Function	Value range	Default parameter
0	reserved		
1	Protocol	E0h: EnOcean	-
2...15	reserved		

You have only to set E0h in Byte 1 as protocol for EnOcean. The other parameters are reserved and not evaluated.

Internal communication

With the help of VIPA-FCs you control the communication between CPU and CP 240. For this, send and receive data have each a reserved 2048Byte buffer which may handle up to 150 telegrams. Together with a CPU 21x the following handling blocks are used:

Label	FCs	Description
SEND	FC0	send block
RECEIVE	FC1	receive block
SYNCHRON_RESET	FC9	reset and synchronization of the CP 240

11Byte telegram for EnOcean communication

Always use telegrams with a length of 11Byte for the communication. At the transmission, the CP 240 EnOcean extends the 11Byte automatically with 2 synchronization bytes and a Checksum to 14Byte res. cuts the 14Byte telegram to 11Byte at reception.

Installation

The recent GSD-files and libraries are to be found either on the CD-ROM "SW-ToolDemo" or under ftp.vipa.de. The installation of the CP 240 EnOcean happens with this approach:

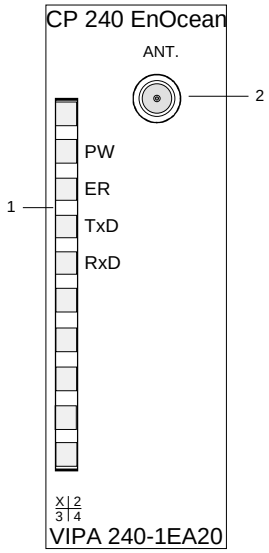
1. Install the GSD-file **VIPA_21X.gsd** in your project-engineering tool.
2. Install the VIPA library **Fx000002_Vxxx.zip** with the handling blocks in your project-engineering tool.
3. Configure your System 200V, parameterize the CP 240 EnOcean and program the communication.
4. Transfer your project into the CPU.

Structure

Properties

- The communication processor has the order number VIPA 240-1EA20
- 16Byte Parameter data
- Voltage supply via back plane bus
- The TCM 120 **T**rans**c**eiver **m**odule works at 868.3MHz.

Front view 240-1EA20



[1] LED Status monitor
[2] SMA antenna jack with male thread and calyx

Components

Power supply

The communication processor gets its voltage supply via the back plane bus.

LEDs

The communication processor is provided with 4 LEDs to monitor the operating status. The meaning and the according colors are shown in the following table.

Label	Color	Description
PW	Green	Signalizes a present operating voltage
ER	Red	Signalizes an error by buffer overflow
TxD	Green	transmit data
RxD	Green	receive data

Antennas

The consignment doesn't include an antenna but you may optional order a portable antenna or a magnetic socket antenna with 150cm cable.

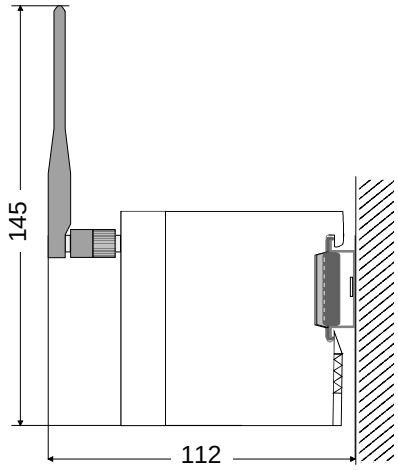
Both antennas are provided with a SMA plug. The coaxial build SMA plug (**s**traight **m**edium **a**daptor) is a miniature HF plug with threaded connector that excels by high HF denseness. In the standard version the plug has a swivel nut with female thread and a pin.

The SMA jack at the CP is with its male thread and the calyx the complement for assembly.

Portable antenna

The portable antenna is a short rod antenna that is mounted without cable directly at the module via the SMA plug.

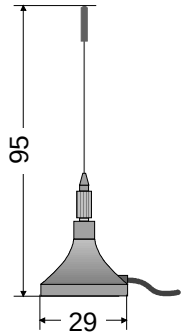
The antenna may be angled and turned into any direction.



all measures in mm

Magnetic socket antenna

The magnetic socket antenna with 150cm cable is convenient for mounting into a cabinet. Due to the magnetic socket you may install the antenna to any steel surface. The connection of the magnetic socket antenna to the CP 240 EnOcean happens via the antenna cable of 150cm with SMA plug.



all measures in mm

Communication principle

Send and receive data

The CPU writes data via the back plane bus, which is to be sent, into the according data channel. The communication processor enters them into a ring buffer (2048Byte) and transmits them then via EnOcean.

When the communication processor receives data via EnOcean, the data is stored in a ring buffer (2048Byte). The received data may now be read telegram by telegram (11Byte) from the CPU via the data channel .

Communication via back plane bus

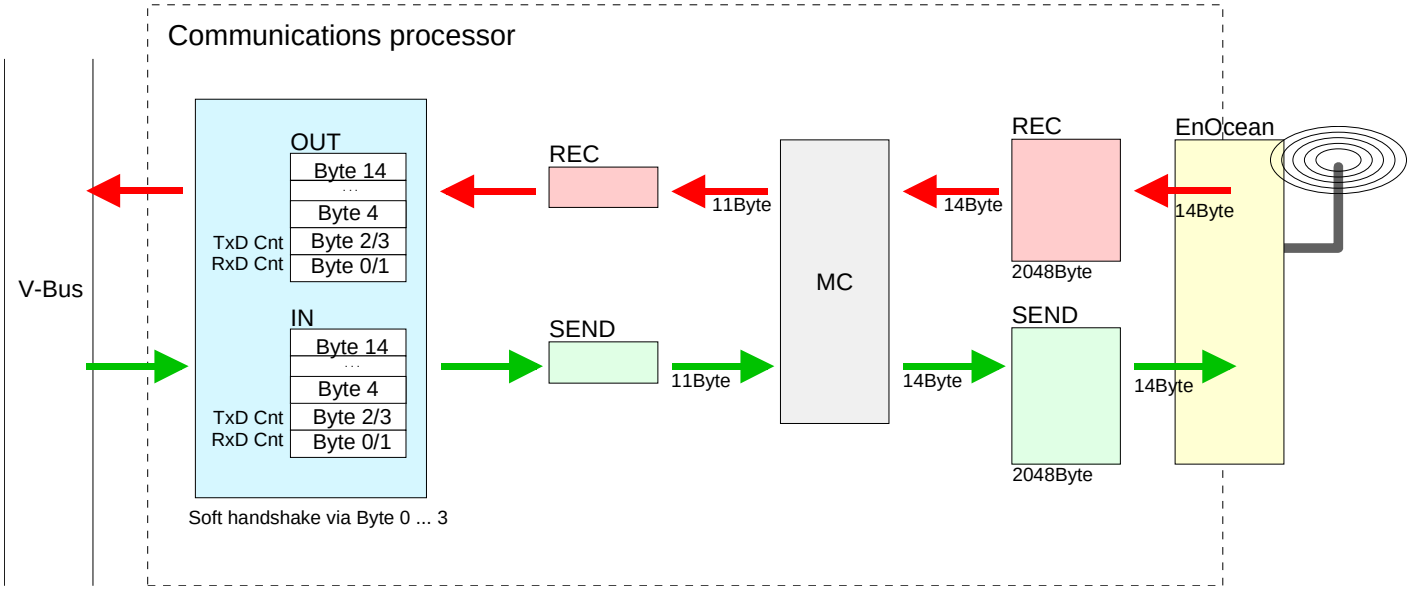
The exchange of received telegrams via back plane bus happens asynchronously. When a complete telegram has been arrived via EnOcean, it is stored in the buffer. The length of the ring buffer limits the maximum number of telegrams. At full buffer new telegrams are ignored.

Out of the telegrams of 14Byte length telegram by telegram 11Byte user data are transferred to the CPU via back plane bus. The first two sync bytes and the Checksum are not handled over.

Tasks of the CPU

A telegram that is to send has to be transferred to the CP 240. This supplements the telegram with the first two sync bytes and the Checksum and handles the telegram on to the send buffer. The CP 240 compiles these blocks in the send buffer and sends it via the EnOcean transceiver as soon as the telegram is complete. Since the data transfer via back plane bus happens asynchronously, a "software handshake" is used between CP 240 and CPU. The registers for the data transfer from the CP 240 have a width of 16Byte. For the handshake, the Bytes 0 to 3 (word 0 and 2) are reserved.

The following picture shall illustrate this:



Software handshake

For the deployment of the CP 240 together with a System 200V CPU VIPA offers handling blocks that enable a comfortable software handshake.

For the deployment of the CP 240 without handling blocks, the following text shows the functionality for transmitting and receiving data with an example.

Example transmitting data w/o handling blocks

An EnOcean telegram contains 11Byte user data. At the transmission the CPU writes for every telegram 11Byte user data into the Bytes 4 to 14 and into Byte 2/3 the length of the telegram (i.e. "11"). The CP 240 receives the data via the back plane bus. To acknowledge the telegram, the CP 240 writes the value "11" (length of the telegram) back to the CPU into Byte 2/3.

At reception of this "11" in Byte 2/3 the CPU sends back a "0" at Byte 2/3. Thereupon the user data in the CP 240 are supplemented to 14Byte with 2 sync bytes at the beginning and the Checksum at the end and stored in the send buffer. After this, the CP responds with a "0" at Byte 2/3. After the CPU received this "0", it may send a new telegram to the CP 240.

The telegrams stored in the send buffer are immediately transmitted via EnOcean.

Example receiving data without handling blocks

Every EnOcean telegram has a size of 14Byte. When the CP 240 receives a telegram this is stored in the receive buffer. For every telegram the 11Byte of user data are handled over to the CPU via the back plane bus into Byte 4 to 14 and the length (i.e. "11") into Byte 0/1. The first two sync bytes and the Checksum are deleted.

The CPU stores the user data and responds with the value "11" at Byte 0/1. The CP acknowledges this with a "0" at Byte 0/1 and thus announces that the transfer has been completed. As soon as new data may be transferred the CPU answers with "0".

With the reception of "0" the CP 240 may send a new telegram to the CPU.

Example for EnOcean deployment

Overview

In the following example an EnOcean communication (send and receive) is build-up. Furthermore the sample illustrates how you may easily establish the control over communication processes by using the handling blocks.

At need you may receive the example project from VIPA.

Requirements

The following components are required for the sample:

1 System 200V consisting of CPU 21x and CP 240 EnOcean

1 switch with EnOcean transmitter

Project engineering tool SIMATIC Manager from Siemens with transmitting cable

Approach

Build-up the System 200V.

Load the example project, if necessary adjust the periphery address and transfer the project into the CPU.

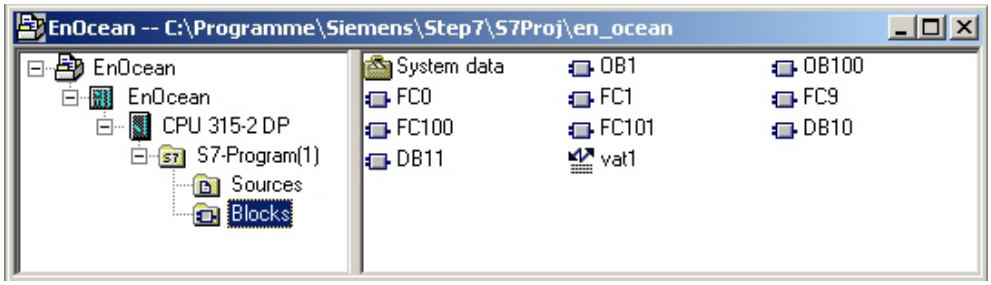
Dearchive the project

Follow these steps in the Siemens SIMATIC Manager:

- Start the Siemens SIMATIC Manager.
- To extract the file Enocean.zip select **File** > *de-archive*.
- Choose the example file Enocean.zip and set "s7proj" as destination directory.
- Open the extracted project.

Project structure

The project already contains the PLC application and the hardware configuration and has the following structure:



Data blocks The example uses the following data blocks:

DB10
Send data block

Addr.	Label	Type	Comment
0.0		STRUCT	
+0.0	Sendefach	STRUCT	Send data block
+0.0	RX_TX_Kennung	BYTE	0B=RX/6B=TX
+1.0	ORG	BYTE	
+2.0	Datenbyte3	BYTE	Data byte 3
+3.0	Datenbyte2	BYTE	Data byte 2
+4.0	Datenbyte1	BYTE	Data byte 1
+5.0	Datenbyte0	BYTE	Data byte 0
+6.0	IDbyte2_3	WORD	ID Byte 2 and 3
+8.0	IDbyte0_1	WORD	ID Byte 0 and 1
+10.0	Status	BYTE	Status
=12.0		END_STRUCT	
+12.0	Reserve	BYTE	
+13.0	SENDEN_LAEUFT	BOOL	Still transmitting
+13.1	LETZTER_BLOCK	BOOL	Last block has been sent
+13.2	FEHL_KOM	BOOL	Error during transmission
+13.3	ENDE_KOM	BOOL	Transfer complete
+14.0	PAFE	BYTE	Parameterization error byte of FC0
+15.0	Res00	BOOL	
+15.1	Res01	BOOL	
+15.2	Res02	BOOL	
+15.3	Res03	BOOL	
+15.4	Res04	BOOL	
+15.5	Res05	BOOL	
+15.6	Res06	BOOL	
+15.7	Senden_start	BOOL	Telegram transmitted completely
+16.0	GESE	WORD	Already sent data
+18.0	ANZ_INT	WORD	Amount of sent data
+20.0	Reserve1	ARRAY[0..50]	
*1.0		BYTE	
=72.0		END_STRUCT	

DB11
Receive data block

Addr.	Label	Type	Comment
0.0		STRUCT	
+0.0	Empfangsfach	STRUCT	Receive data block
+0.0	RX_TX_Kennung	BYTE	0B=RX/6B=TX
+1.0	ORG	BYTE	
+2.0	Datenbyte3	BYTE	Data byte 3
+3.0	Datenbyte2	BYTE	Data byte 2
+4.0	Datenbyte1	BYTE	Data byte 1
+5.0	Datenbyte0	BYTE	Data byte 0
+6.0	IDbyte2_3	WORD	ID byte 2 and 3
+8.0	IDbyte0_1	WORD	ID byte 0 and 1
+10.0	Status	BYTE	Status
=12.0		END_STRUCT	
+12.0	Reserve	BYTE	
+13.0	EMP_LAEUFT	BOOL	Still receiving
+13.1	LETZTER_BLOCK	BOOL	Last block has been received
+13.2	FEHL_EMPF	BOOL	Error during reception
+14.0	PAFE	BYTE	Parameterization error byte of FC1
+15.0	Res00	BOOL	
+15.1	Res01	BOOL	
+15.2	Res02	BOOL	
+15.3	Res03	BOOL	
+15.4	Res04	BOOL	
+15.5	Res05	BOOL	
+15.6	Res06	BOOL	
+15.7	Empfang_fertig	BOOL	Telegram received completely
+16.0	GEEM	WORD	Already received data
+18.0	ANZ_INT	WORD	Amount of received data
+20.0	Reserve1	ARRAY[0..50]	
*1.0		BYTE	
=72.0		END_STRUCT	

PLC program

The example already contains the PLC application and the hardware configuration. The following blocks are used:

OB 1

```
CALL  FC 9                    //Re-boot or Reset
 ADR      :=256               //Address of the module
 TIMER_NR :=T2
 ANL      :=M3.0
 NULL     :=M3.1
 RESET    :=M3.2
 STEUERB_S:=MB4
 STEUERB_R:=MB6
U  M  3.0                     //Reception can only be started
BEB                           //after execution of FC 9 (SYNCHRON_RESET)
CALL FC 100                   //Call of reception-FC
```

OB 100

```
L  0
T  MB 1                       //delete order bit
UN M  3.0                     //Initialize re-boot
S  M  3.0
```

FC 100

```
CALL  FC 1
 ADR  :=256                                      //Address of the module
 _DB  :="EMPFANG_en_ocean"                       //DB with received data
 ABD  :=W#16#0                                   //1. DBB received data
 ANZ  :=W#16#B                                   //reception length always 11
 EMFR :=M7.0                                     //all data received
 PAFE :="EMPFANG_en_ocean".Pafe                  //error byte
 GEEM :="EMPFANG_en_ocean".GEEM                  //received amount (internal)
 ANZ_INT:="EMPFANG_en_ocean".ANZ_INT             //reception length (internal)
 EMPF_LAEUFT:="EMPFANG_en_ocean".EMP_LAEUFT      //receiving data (internal)
 LETZTER_BLOCK:="EMPFANG_en_ocean".LETZTER_BLOCK    //all data received
 FEHL_EMPF:="EMPFANG_en_ocean".FEHL_EMPF         //Error in the
                                                 //reception routine
UN    M         7.0                             //no telegram received
BEB                                             //then end
R     M         7.0                             //delete reception bit
L     "EMPFANG_en_ocean".Empfangsfach.IDbyte0_1 //switch ID
L     W#16#1C7A                                 //Please enter here the ID
==I                                             //of your switch.
SPB   e_a2                                      //This can be taken from
BEA                                             //DB 11.DBW 8

e_a2: NOP   0
L     5                                         //ID switch on
L     "EMPFANG_en_ocean".Empfangsfach.Datenbyte3    //Byte with ID
SRW   4                                         //ID in Low nipple
==I                                             //Proof if switch is pushed
SPB   ein
L     7                                         //ID switch of
==I                                             //Proof if switch is pushed
SPB   aus
BEA

ein:  NOP   0
S     A         0.0                             //Function on
BEA

aus:  NOP   0
R     A         0.0                             //Function off
BEA
```

FC 101

```
L     B#16#6B                                          //allocate send data
T     "SEND_en_ocean".Empfangsfach.RX_TX_Kennung       //send ID
L     B#16#5                                           //ORG-ID
T     "SEND_en_ocean".Empfangsfach.ORG
L     B#16#2
T     "SEND_en_ocean".Empfangsfach.Datenbyte3
L     0
T     "SEND_en_ocean".Empfangsfach.Datenbyte2
T     "SEND_en_ocean".Empfangsfach.Datenbyte1
T     "SEND_en_ocean".Empfangsfach.Datenbyte0
T     "SEND_en_ocean".Empfangsfach.IDbyte2_3
L     W#16#3267                                        //Only the last 7Bit
T     "SEND_en_ocean".Empfangsfach.IDbyte0_1           //are relevant for addr.
L     6                                                //and are "ORed" in the
T     "SEND_en_ocean".Empfangsfach.Status              //CP 240 with the here
                                                       //stored ID-Base

CALL  FC 0
 ADR          :=256
 _DB          :="SEND_en_ocean"
 ABD          :=W#16#0                          //send from data byte 0 on
 ANZ          :=W#16#B                          //always 11Byte
 PAFE         :="SEND_en_ocean".Pafe
 FRG          :="SEND_en_ocean".Senden_start
 GESE         :="SEND_en_ocean".GEEM
 ANZ_INT      :="SEND_en_ocean".ANZ_INT
 ENDE_KOM     :="SEND_en_ocean".ENDE_KOM
 LETZTER_BLOCK:="SEND_en_ocean".LETZTER_BLOCK
 SENDEN_LAEUFT:="SEND_en_ocean".SENDEN_LAEUFT
 FEHLER_KOM   :="SEND_en_ocean".FEHL_KOM
```

Overview of the EnOcean telegrams

General structure

The following table shows the general structure of an EnOcean telegram. Send and receive telegrams have the same structure. They only differ in the ID.

Bit 7 Bit 0

0xA5	These bytes are automatically created at transmission and hidden at reception.
0x5A	
0x0B 0x6B	0x0B: ID for reception telegram 0x06: ID for send telegram
ORG	See table *supported ORG formats*
DataBytes3	Data from a sensor res. to an actuator
DataBytes2	
DataBytes1	
DataBytes0	
IDBytes3*	ID of the transceiver module. With SET_IDBASE you may alter the ID up to 10 times.
IDBytes2*	
IDBytes1*	
IDBytes0*	
Status	Status information of the according sensor
Checksum	Is automatically created at transmission and hidden at reception.

*) During transmission the actual ID base of the module replaces the ID base in the telegram.

General

At the following pages all telegrams are listed that are supported by the CP 240 EnOcean. This description has been taken directly out of the documentation by courtesy of EnOcean.

Note!

Please regard that the first two synchronization bytes and the Checksum of received telegrams are not stored in the CP 240. At transmission, the 11Byte user data are automatically supplemented with these bytes to a total size of 14Byte.

Description of ORG field

The TX_TELEGRAM and RX_TELEGRAM telegrams have the same structure. The only difference is that a TX_TELEGRAM is identified by "3" in H_SEQ instead of "0" for an RX_TELEGRAM.

ORG	Description	RRT / TRT Acronym
0x05	Telegram from a PTM switch module received (original or repeated message)	RPS
0x06	1 byte data telegram from a STM sensor module received (original or repeated message)	1BS
0x07	4 byte data telegram from a STM sensor module received (original or repeated message)	4BS
0x08	Telegram from a CTM module received (original or repeated message)	HRC
0x0A	6byte Modem Telegram (original or repeated)	6DT
0x0B	Modem Acknowledge Telegram	MDA

Serial command encoding for RPS, 1BS, 4BS, HRC

Bit 7 Bit 0

0xA5
0x5A
0x0B (RX_TELEGRAM) 0x6B(TX_TELEGRAM)
ORG
DataBytes3
DataBytes2
DataBytes1
DataBytes0
IDBytes3
IDBytes2
IDBytes1
IDBytes0
Status
ChkSum

DataBytes2= DataBytes1= DataBytes0= 0x00 for RPS,1BS, HRC

Serial command encoding for 6DT

Bit 7 Bit 0

0xA5
0x5A
0x0B (RX_TELEGRAM) 0x6B(TX_TELEGRAM)
0x0A
DataBytes5
DataBytes4
DataBytes3
DataBytes2
DataBytes1
DataBytes0
Address1
Address0
Status
ChkSum

Serial command encoding for MDA

Bit 7 Bit 0

0xA5
0x5A
0x0B (RX_TELEGRAM) 0x6B(TX_TELEGRAM)
0x0B
0xXX
0xXX
0xXX
0xXX
Address1
Address0
0xXX
0xXX
Status
ChkSum

Description of STATUS field

If ORG = 0x05 (Telegram from a PTM switch module)

7			0
Reserved	**T21**	**NU**	**RP_COUNTER**

Reserved (2 bit) Do not care

T21 (1 bit) T21=0 → PTM type 1, T21=1 → PTM type 2

Note: In transmission the TCM 120 always sets T21=1
→ it is only possible to transmit PTM type 2 telegrams!

NU (1 bit) NU=1 → N-message, NU=0 → U-message.

RP_COUNTER (4 bit) =0..15 Repeater level: 0 is original message

IMPORTANT NOTE

Within toggle switch applications using the RCM 120 or TCM 120 serial receiver mode in combination with the TCM 110 repeater module, please ensure that no serial command interpretation error may occur at the connected control unit. A toggle signal means that the same telegram (from e.g. PTM 100, PTM 200 or STM 100) is sent for switching something on and off. If e.g. the light is switched on by means of a RCM 120 receiving the I-button telegram from a PTM 100, the repeated telegram (delay <100ms) may switch off the light again. It is therefore mandatory to interpret the RP_COUNTER field as described in the RCM 120 User Manual. If a repeated telegram (RP_COUNTER>0) is received it has to be verified if the same telegram with a lower RP_COUNTER state has already been received in the previous 100 ms. In this case the repeated message has to be discarded.

PTM Type 1

PTM switch modules of Type 1 (e.g. PTM 100) do not support interpretation of operating more than one rocker at the same time:

N-message received → Only one pushbutton was pressed.

U-message received → No pushbutton was pressed when activating the energy generator, or more than one pushbutton was pressed.

PTM Type 2

PTM switch modules of Type 2 allow interpretation of operating two buttons simultaneously:

N-message received → Only one or two pushbuttons have been pressed.

U-message received → No pushbutton was pressed when activating the energy generator, or more than two pushbuttons have been pressed.

Note for telegrams from PTM 100 piezo transmitters:

Due to the mechanical hysteresis of the piezo energy bow, in most rocker switch device implementations, pressing the rocker sends an N-message and releasing the rocker sends a U-message!

If ORG = 0x06, 0x07, 0x08 or 0x0A:

7	0
Reserved	**RP_COUNTER**

Reserved (4 bit) Do not care

RP_COUNTER (4 bit) Repeater level: 0 original message
1 repeated message

Description of DATA_BYTE 3..0

If ORG = 0x05 and NU = 1 (N-message from a PTM switch module):

DATA_BYTE2..0 always = 0
DATA_BYTE3 as follows:

7					0
RID	**UD**	**PR**	**SRID**	**SUD**	**SA**

RID	(2 bit)	Rocker ID, from left (A) to right (D): 0, 1, 2 and 3 (decimal)
UD	(1 bit)	UD=1 → O-button, UD=0 → I-button
PR	(1 bit)	PR=1 → energy bow pressed PR=0 → energy bow released
SRID	(2 bit)	Second Rocker ID, from left to right: 0, 1, 2 and 3
SUD	(1 bit)	(Second) SUD=1 → O-button, SUD=0 → I-button
SA	(1 bit)	SA=1 → Second action (2 buttons pressed simultaneously), SA=0 → No second action

If ORG = 0x05 and NU = 0 (U-message from a PTM switch module):

DATA_BYTE2..0 always = 0
DATA_BYTE3 as follows:

7		0
BUTTONS	**PR**	**Reserved**

BUTTONS	(3 bit)	Number of simultaneously pressed buttons, as follows: PTM 100 / PTM200 0 = 0 Buttons / 0 = 0 Button 1 = 2 Buttons / 1 = not possible 2 = 3 Buttons / 2 = not possible 3 = 4 Buttons / 3 = 3 or 4 buttons 4 = 5 Buttons / 4 = not possible 5 = 6 Buttons / 5 = not possible 6 = 7 Buttons / 6 = not possible 7 = 8 Buttons / 7 = not possible
PR	(1 bit)	PR = 1 → energy bow pressed PR = 0 → energy bow released
Reserved	(4 bit)	for future use

If ORG = 0x06 (Telegram from a 1 Byte STM sensor):

DATA_BYTE2..0 always = 0
DATA_BYTE3 Sensor data byte.

If ORG = 0x07 (Telegram from a 4 Byte STM sensor):

DATA_BYTE3 Value of third sensor analog input
DATA_BYTE2 Value of second sensor analog input
DATA_BYTE1 Value of first sensor analog input
DATA_BYTE0 Sensor digital inputs as follows:

7				0
Reserved	**DI_3**	**DI_2**	**DI_1**	**DI_0**

If ORG = 0x08 (Telegram from a CTM module set into HRC operation):

DATA_BYTE2..0 always = 0
DATA_BYTE3 as follows:

7				0
RID	**UD**	**PR**	**SR**	**Reserved**

RID	(2 bit)	Rocker ID, from left (A) to right (D): 0, 1, 2 and 3
UD	(1 bit)	UD=1 → O-button, UD=0 → I-button
PR	(1 bit)	PR=1 → Button pushed, PR=0 → Button released
SR	(1 bit)	SR=1 → Store, SR=0 → Recall (see note)
Reserved	(3 bit)	for future use

Note: The SR bit is used only when the lower 3 bits from ID_BYTE0 = B'111' (scene switch), and RID ≠ 0 (indicates that the memory buttons M0-M6 are operated in the handheld remote control).

If ORG = 0x0A (Modem telegram):

Please note the different structure of modem telegrams with 6 data bytes and 2 address bytes for the ID of the receiving modem. See A.1.1.

Command Telegrams and Messages

INF_INIT

After a power-on, a hardware reset or a RESET command the TCM informs the user through several of these telegrams about the current status. The messages have the general syntax as shown. The information contained by the bytes marked as X should be decoded according to ASCII code.

Bit 7	*Bit 0*
0xA5	
0x5A	
0x8B	
0x89	
X	
X	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

In total there are 15 telegrams:

0xA5 0x5A 0x8B 0x89	" "
0xA5 0x5A 0x8B 0x89	"EnOcean"
0xA5 0x5A 0x8B 0x89	"TCM120"
0xA5 0x5A 0x8B 0x89	"Version"
0xA5 0x5A 0x8B 0x89	Version number in ASCII
0xA5 0x5A 0x8B 0x89	"Bdrate"
0xA5 0x5A 0x8B 0x89	*"0x40" (9600 baud)*
0xA5 0x5A 0x8B 0x89	"Modem"
0xA5 0x5A 0x8B 0x89	"ON" or "OFF"
0xA5 0x5A 0x8B 0x89	"RxID"
0xA5 0x5A 0x8B 0x89	modem ID in ASCII
0xA5 0x5A 0x8B 0x89	"Mode"
0xA5 0x5A 0x8B 0x89	"Run"
0xA5 0x5A 0x8B 0x89	"PrgMem"
0xA5 0x5A 0x8B 0x89	"OK" or "CORRUPT"

OK

Standard message used to confirm that an action was performed correctly by the TCM.

Bit 7 *Bit 0*

0xA5
0x5A
0x8B
0x58
X
X
X
X
X
X
X
X
X
ChkSum

ERR

Standard error message response if after a TCT command the operation could not be carried out successfully by the TCM.

Bit 7 Bit 0

0xA5
0x5A
0x8B
0x19
X
X
X
X
X
X
X
X
X
ChkSum

RD_IDBASE

When this command is sent to the TCM, the base ID range number is retrieved though an INF_IDBASE telegram.

Bit 7 Bit 0

0xA5
0x5A
0xAB
0x58
X
X
X
X
X
X
X
X
X
ChkSum

SET_IDBASE

With this command the user can rewrite its ID range base number. The most significant ID byte is IDBaseByte3**.** The information of the 25 most significant bits is stored in EEPROM.

The allowed ID range is from 0xFF800000 to 0xFFFFFFFF.

Bit 7 Bit 0

0xA5
0x5A
0xAB
0x18
IDBaseByte3
IDBaseByte2
IDBaseByte1
IDBaseByte0
X
X
X
X
X
ChkSum

32 **0**

25 most significant bits	**0**	**0**	**0**	**0**	**0**	**0**	**0**

ID range base

This command can only be used a maximum number of 10 times. After successfully ID range reprogramming, the TCM answers with an OK telegram. If reprogramming was not successful, the TCM answers sending an ERR telegram if the maximum number of 10 times is exceeded or an ERR_IDRANGE telegram if the ID range base is not within the allowed range.

INF_IDBASE

This message informs the user about the ID range base number.

Bit 7 Bit 0

0xA5	
0x5A	
0x8B	
0x98	
IDBaseByte3	IDBaseByte3 is the most significant byte.
IDBaseByte2	
IDBaseByte1	
IDBaseByte0	
X	
X	
X	
X	
X	
ChkSum	

SET_RX_SENSITIVITY

This command is used to set the TCM radio sensitivity.

In LOW radio sensitivity, signals from remote transmitters are not detected by the TCM receiver. This feature is useful when only information from transmitters in the vicinity should be processed. An OK confirmation telegram is generated after TCM sensitivity has been changed.

Bit 7 Bit 0

0xA5
0x5A
0xAB
0x08
Sensitivity
X
X
X
X
X
X
X
X
ChkSum

Sensitivity=0x00 Low sensitivity
Sensitivity=0x01 High sensitivity

RD_RX_SENSITIVITY

This command is sent to the TCM to retrieve the current radio sensitivity mode (HIGH or LOW). This information is sent via an INF_RX_ SENSITIVITY command.

Bit 7 Bit 0

0xA5
0x5A
0xAB
0x48
X
X
X
X
X
X
X
X
X
ChkSum

INF_RX_SENSITIVITY

This message informs the user about the current TCM radio sensitivity.

Bit 7 Bit 0

0xA5
0x5A
0x8B
0x88
Sensitivity
X
X
X
X
X
X
X
X
ChkSum

Sensitivity= 0x00 Low sensitivity
Sensitivity= 0x01 High sensitivity

SLEEP

If the TCM receives the SLEEP command, it works in an energy-saving mode. The TCM will not wake up before a hardware reset is made or a WAKE telegram is sent via the serial interface.

Bit 7	Bit 0
0xA5	
0x5A	
0xAB	
0x09	
X	
X	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

WAKE

If the TCM receives the WAKE command, it wakes up from sleep mode. In contrast to all other telegrams this telegram is only one byte long.

Bit 7	Bit 0
0xAA	

RESET

Performs a reset of the TCM micro controller. When the TCM is ready to operate again, it sends an ASCII message (INF_INIT) containing the current settings.

Bit 7	*Bit 0*
0xA5	
0x5A	
0xAB	
0x0A	
X	
X	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

MODEM_ON

Activates TCM modem functionality and sets the modem ID. An OK confirmation telegram is generated. The modem ID is the ID at which the TCM receives messages of type 6DT. The modem ID and modem status (ON/OFF) is stored in EEPROM. The modem ID range is from 0x0001 to 0xFFFF. IF 0x0000 is provided as modem ID, the modem is activated with the ID previously stored in EEPROM.

Bit 7	Bit 0
0xA5	
0x5A	
0xAB	
0x28	
Modem ID (MSB)	
Modem ID (LSB)	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

MODEM_OFF

Deactivates TCM modem functionality. When this command has been sent, an OK command should be received, confirming that the modem status is OFF. The modem ID is not erased.

Bit 7	Bit 0
0xA5	
0x5A	
0xAB	
0x2A	
X	
X	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

RD_MODEM_ STATUS

This command requests the TCM to send information about its current modem current status. The requested information is reported to the user through an INF_MODEM_STATUS telegram.

Bit 7	Bit 0
0xA5	
0x5A	
0xAB	
0x68	
X	
X	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

INF_MODEM_ STATUS

Informs the user about the TCM current modem status. The information provided is the following: Modem status (ON or OFF) and modem ID stored.

Modem state=0x01, modem ON
Modem state=0x00, modem OFF

Modem ID MSB= most significant modem ID byte.
Modem ID LSB=least significant modem ID byte.

Bit 7 Bit 0

0xA5
0x5A
0x8B
0xA8
Modem status
Modem ID MSB
Modem ID LSB
X
X
X
X
X
X
ChkSum

RD_SW_VER

This command requests the TCM to send its current software version number. This information is provided via an INF_SW_VER telegram by the TCM.

Bit 7 Bit 0

0xA5
0x5A
0xAB
0x4B
X
X
X
X
X
X
X
X
X
ChkSum

INF_SW_VER Informs the user about the current software version of the TCM.

Bit 7 Bit 0

0xA5
0x5A
0x8B
0x8C
TCM SW Version Pos.1
TCM SW Version Pos.2
TCM SW Version Pos.3
TCM SW Version Pos.4
X
X
X
X
X
ChkSum

Example: Version 1.0.1.16
TCM SW Version Pos.1 = 1
TCM SW Version Pos.2 = 0
TCM SW Version Pos.3 = 1
TCM SW Version Pos.4 =16

ERR_MODEM_NOTWANTEDACK When a 6DT modem telegram has been sent, the TCM waits for a modem acknowledge (MDA) telegram. This error message is generated if an MDA with the right modem ID is received after the timeout (100ms) or if there is more than one MDA received.

Bit 7 Bit 0

0xA5
0x5A
0x8B
0x28
X
X
X
X
X
X
X
X
X
ChkSum

ERR_MODEM_NOTACK When a 6DT modem telegram has been sent, the TCM waits for a modem acknowledge (MDA) telegram. This error message is generated if no acknowledge was received before the timeout (100ms).

Bit 7 Bit 0

0xA5
0x5A
0x8B
0x29
X
X
X
X
X
X
X
X
X
ChkSum

ERR_MODEM_ DUP_ID

When the TCM receives an original (not repeated) MDA telegram with the same modem ID as its own, it sends this message through the serial port and informs that at least 2 TCMs have the same modem ID. This is not necessarily a problem and may even be intended. On the other hand it may also indicate that there is another installation/building in the vicinity where the same modem ID is in use.

Bit 7	*Bit 0*
0xA5	
0x5A	
0x8B	
0x0C	
X	
X	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

ERR_SYNTAX

This telegram is sent automatically through the serial port after the TCM has detected a syntax error in a TCT telegram.

Errors can occur in the H_SEQ, LENGTH, ORG or CHKSUM fields/bytes.

Bit 7	Bit 0
0xA5	
0x5A	
0x8B	
Field	
X	
X	
X	
X	
X	
X	
X	
X	
X	
ChkSum	

Field code:
H_SEQ=0x08 ORG=0x0B
LENGTH=0x09 CHKSUM=0x0A

ERR_TX_ IDRANGE

When a radio telegram intended to be sent has an ID number outside the ID range, this error message is generated. The radio telegram is not delivered.

Bit 7 Bit 0

0xA5
0x5A
0x8B
0x22
X
X
X
X
X
X
X
X
X
ChkSum

ERR_ IDRANGE

This message is generated when the user tries to change the ID range base using the SET_IDBASE command to a value outside the allowed range from 0xFF800000 to 0xFFFFFFFF.

Bit 7 Bit 0

0xA5
0x5A
0x8B
0x1A
X
X
X
X
X
X
X
X
X
ChkSum

Exchange module and set ID base

Overview

Since the ID base of every module is different, you have in case of replacement the option to change the ID base of a module for up to 10 times by means of a SET_IDBASE telegram. Consequently the newly adjustment of the actuators to the replacement module is no longer necessary. After successful transfer of the ID base you have either to reboot your CPU or reset it via FC 9.

Please regard that only the upper 25 bits are taken over as ID base. The remaining 7 bits you may specify via your user application during runtime and herewith address multiple actuators.

ID base request

With RD_IDBASE the current ID base of your module may be requested.

RD_IDBASE

0xAB	ID for transmission telegram
0x58	ORG ID for RD_IDBASE
X	Irrelevant
...	...
X	Irrelevant

INF_IDBASE

RD_IDBASE reports back the current ID base of the module in form on an INF_IDBASE telegram. The telegram has the following structure:

0x8B	ID for reception telegram
0x98	ORG-ID for INF_IDBASE
IDBaseByte3	Byte 3 current ID base
IDBaseByte2	Byte 2 current ID base
IDBaseByte1	Byte 1 current ID base
IDBaseByte0	Byte 0 current ID base (Bit 6...0 irrelevant)
X	irrelevant
...	...
X	irrelevant

SET_IDBASE

In case of replacement send a SET_IDBASE telegram with the following structure from your CPU to the module (transceiver). Use the address of the module you want to replace as new ID base:

0xAB	ID for transmission telegram
0x18	ORG ID for SET_IDBASE
IDBaseByte3	Byte 3 new ID base
IDBaseByte2	Byte 2 new ID base
IDBaseByte1	Byte 1 new ID base
IDBaseByte0	Byte 0 new ID base (Bit 6...0 irrelevant)
X	irrelevant
...	...
X	irrelevant

Possible respond telegram

0x8B **0x58**	←	**OK** – ID base was set

To take over the ID base during runtime you have to execute a reset via FC 9. Otherwise the new ID base is available after a reboot of the CPU.

In case of an error you receive one of the following messages. Here the old ID base remains valid.

0x8B **0x19**	←	**ERR** More than 10 ID base changes are not permitted
0x8B **0x1A**	←	**ERR_IDRANGE** ID base is outside the valid range (0xFF800000 ... 0xFFFFFFFF).

Control your ID settings and send the telegram once more.

Technical data

CP 240 EnOcean

Electrical data	VIPA 240-1EA20
Number of channels	1
Power supply	5V via back plane bus
Current consumption	max. 120mA
Power loss	0.75W
External power supply	-
Status indicator (LEDs)	via LED on the front
Radio transceiver module	
Type	EnOcean TCM120
Frequency / modulation type / transmitter power	868.3MHz / ASK / max.10mW
Transmission range	300m free area
Aerial	external 50Ω-aerial mountable
Type	portable or with 150cm cable and magnetic socket
Connection	SMA jack
Programming data	
Input data	16Byte
Output data	16Byte
Parameter data	16Byte
Diagnostic data	-
Dimensions and weight	
Dimensions (WxHxD) in mm	25.4x76x78
Weight	80g

Note!

Please regard that for the usage of the module national guidelines must be kept!

The adherence of these guidelines is incumbent on the user!

Chapter 6 CP 240 - M-Bus

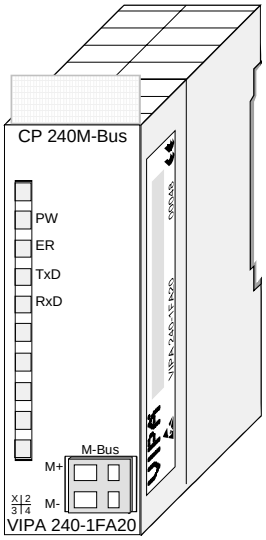
Overview

This chapter contains information about the assembly and the inclusion of the CP 240 M-Bus for communication with energy and excise counters.

Content

System overview

CP 240 M-Bus



Order data

Type	Order No.	Description
CP 240 M-Bus	VPA 240-1FA20	CP with M-Bus interface

Basics

M-Bus

The M-Bus system (**M**etering **Bus**) is an European standardized (DIN EN 1434-3) two-wire field bus for the excise data capturing. At this the data transfer happens serial via a polarity inversion secure two-wire core from slave systems (measuring devices) to a master system.

The M-Bus has been developed in Germany by Prof. Dr. Horst Ziegler (Uni Paderborn) together with the companies Techem and Texas Instruments.

The main advantage of the M-Bus technique is its high flexibility. Due to the standardization you may easily combine devices of several manufacturers at one bus. Also the inclusion of emitter counter is possible via special M-Bus adapters. Up to 250 counters may be connected at one bus.

Properties

- Polarity inversion secure standardized bus system acc. DIN EN 1434-3
- Short-circuit resistant M-Bus interface
- Current, gas, water and heat counter can be integrated
- Data is read electronically
- Connection of up to 250 counter at one bus
- Counter are individually addressable via unique addresses
- Remote reading possible in dense read intervals
- Extraction of statistic data as base for net optimization
- No special requirements for bus cables or cabling topologies
- Striking distance up to several km

Transfer principle

The data transfer from master to slaves happens via the modulation of the voltage supply. Here at 36V account for the state "1" and 24V for state "0".

The slave system responds the master via the modulation (increase) of its current consumption. Herat 1.5mA account for the state "1" and 11-20mA for state "0".

Due to the voltage modulation and thus the present M-Bus voltage of 24V the terminal equipment can be supplied with the necessary operating voltage.

Fast introduction

Overview

The communication processor CP 240 M-Bus allows the process coupling to different destination or source systems based upon the M-Bus communication.

The CP 240 M-Bus is power supplied via the back plane bus. For the internal communication the VIPA FCs are used. For the project engineering of the CP 240 M-Bus together with a CPU 21x in the Siemens SIMATIC Manager, the inclusion of the GSD VIPA_21x.gsd is required. To enable the CP 240 M-Bus to communicate with the CPU, the system always requires a hardware configuration.

A general description of the project engineering of the CP 240 is to be found in chapter "Project engineering".

Parameters

By placing the CP 240 M-Bus into the "virtual" Profibus system in the hardware configuration, the required parameters are created automatically. The parameter area has the following structure:

Byte	Function	Values	Default parameter
0	reserved		
1	Protocol	F0h: M-Bus	-
2...15	reserved		

You have only to set F0h in Byte 1 as protocol for M-Bus. The other parameters are reserved and not evaluated.

Internal communication

With the help of VIPA-FCs you control the communication between CPU and CP 240 M-Bus. For this, send and receive data have each a reserved 2048Byte buffer. Together with a CPU 21x the following handling blocks are used:

Name	FCs	Short description
SEND	FC0	Send block
RECEIVE	FC1	Receive block
SYNCHRON_RESET	FC9	Reset and synchronize the CP 240

Telegram structure

For M-Bus telegrams that are send from the CPU to the CP 240, you must prefix every telegram with one byte, which contains the baud rate. This procedure allows you to communicate with different bus participants during runtime by using different baud rates.

As a countermove the CP 240 announces back the state of the M-Bus data transfer via this byte (0: OK - valid answer, ≠0:error).

Installation

The recent GSD-files and libraries are to be found either on the CD-ROM "SW-ToolDemo" or under ftp.vipa.de. The installation of the CP 240 M-Bus happens with this approach:

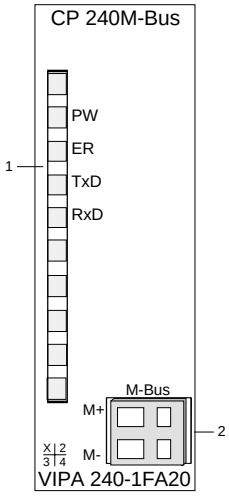
1. Install the GSD-file **VIPA_21X.gsd** in your project-engineering tool.
2. Install the VIPA library **Fx000002_Vxxx.zip** with the handling blocks in your project-engineering tool.
3. Configure your System 200V, parameterize the CP 240 M-Bus and program the communication.
4. Transfer your project into the CPU.

Structure

Properties

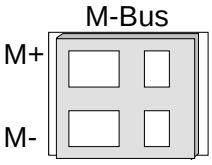
- The communication processor has the order number VIPA 240-1FA20
- Voltage supply via back plane bus
- Up to 6 slaves may be connected
- Standardized bus system acc. DIN EN 1434-3

Front view 240-1FA20



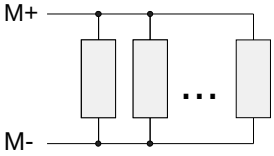
[1] LED Status monitor
[2] M-Bus interface

M-Bus interface



The labels M+ and M- serve the distinction of the M-Bus wires. The polarization is irrelevant for M-Bus installations.

Since the CP gets its voltage supply via the back plane bus and thus supplies the connected M-Bus modules, up to 6 slaves may be connected. The slaves must be connected in parallel.



LEDs

The communication processor is provided with 4 LEDs to monitor the operating status. The meaning and the according colors are shown in the following table.

Name.	Color	Description
PW	Green	Signalizes a present operating voltage
ER	Red	Signalizes an error e.g. by buffer overflow, wrong baud rate or missing stop bit.
TxD	Green	Transmit data
RxD	Green	Receive data

Communication principle

Send and receive data

The CPU writes data via the back plane bus, which is to be sent into the according data channel.

The communication processor enters them into a ring buffer (2048Byte) and transmits them via M-Bus.

When the communication processor receives data via M-Bus, the data is stored in a ring buffer (2048Byte). The received data may now be read telegram by telegram from the CPU via the data channel.

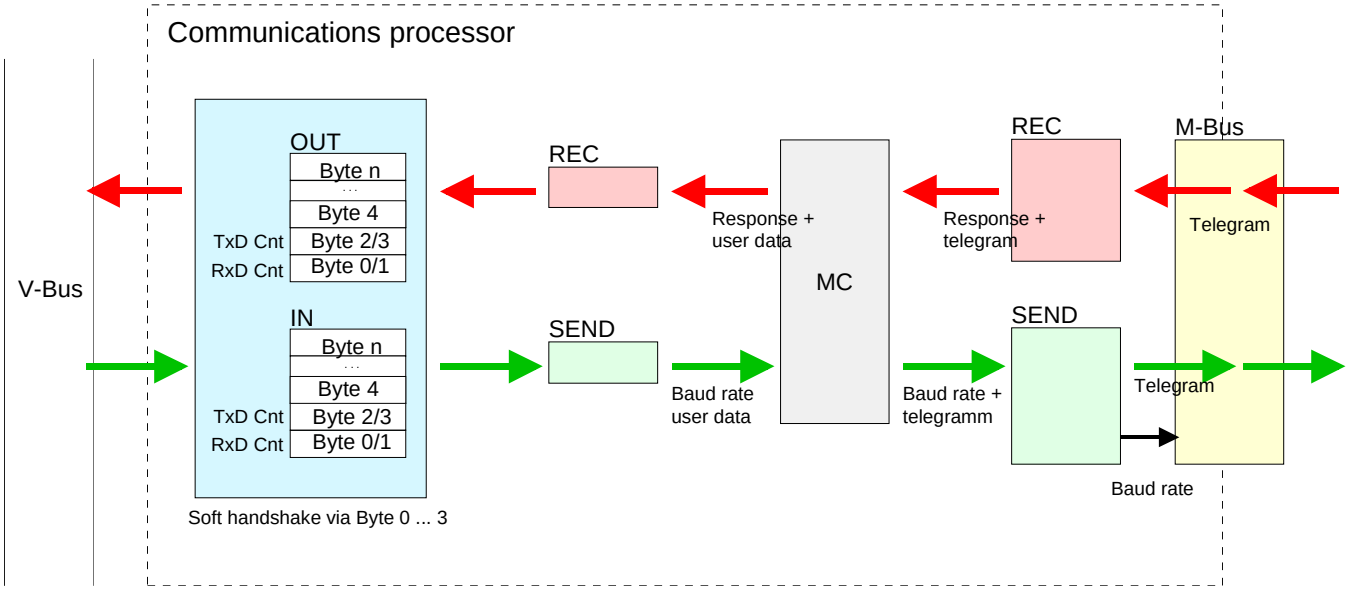
Communication via back plane bus

The exchange of received telegrams via the back plane bus happens asynchronously. When a complete telegram has been arrived via M-Bus, it is stored in the buffer. The user data are extracted from the M-Bus telegram and transferred to the CPU via back plane bus.

Tasks of the CPU

A telegram that is to send has to be transferred to the CP 240. This recognizes the telegram type due to the length definition, supplements it with the according telegram bytes and handles the telegram on to the send buffer. The CP 240 compiles these blocks in the send buffer and sends it via M-Bus with the specified baud rate as soon as the telegram is complete. Since the data transfer via back plane bus happens asynchronously, a "software handshake" is used between CP 240 and CPU. The registers for the data transfer from the CP 240 have a width of 16Byte. For the handshake, the Bytes 0 to 3 (word 0 and 2) are reserved.

The following picture shall illustrate this:



Software handshake

For the deployment of the CP 240 together with a System 200V CPU VIPA offers you a series of standard handling blocks that provide the software handshake comfortable and easy.

At deployment of the CP 240 without handling blocks, the functionality is elucidated with an example of data send and receive.

Example SEND data

For example, a telegram with 30Byte length is to send. Please regard that the CP 240 takes the 1st byte of the telegram as baud rate. The CPU writes the first 12Byte user data of the telegram into the Bytes 4 to 15. Byte 2/3 contain the telegram length, i.e. "30". The CP 240 receives the data via the back plane bus and copies the 12Byte user data into the send buffer. For the acknowledgement of the telegram the CP 240 writes the value "30" back to Byte 2/3 (length of the telegram).

At reception of the "30", the CPU can send further 12Byte user data to Byte 4 to 15 and the rest length of the telegram ("18" Byte) to Byte 2/3 to the CP 240. Again, this stores the user data in the send buffer and sends back the length information ("18") in Byte 2/3 to the CPU.

The CPU receives the "18" and sends the remaining 6Byte user data in the Bytes 4 to 9 and the according rest length ("6") in Byte 2/3 to the CP 240. The user data is stored in the send buffer and the value "6" is send back to the CPU via Byte 2/3.

The CPU receives the "6" and sends back a "0" via Byte 2/3. The CP 240 now initializes the sending of the telegram via the M-Bus interface. After data transfer is completed, the CP 240 sends back a "0" to the CPU via Byte 2/3.

At reception of the "0", the CPU is able to send a new telegram to the CP 240.

Example RECEIVE data

For example, the CP 240 received a telegram with 18Byte user data via M-Bus. Out of this telegram the first 11Byte user data and the prefixed respond byte are taken over into the Bytes 4 to 15 of the receive buffer and the length of the telegram (i.e. "18") into Byte 0/1. The data is transferred to the CPU via the back plane bus. The CPU stores the 12Byte user data and sends back the length value "18" to the CP 240.

At reception of the "18", the CP 240 writes the remaining 7Byte user data into Byte 4 to 10 of the receive buffer and the length ("7") of the transferred user data into Byte 0/1. The CPU stores the user data and announces back to the CP 240 the value "7" into Byte 0/1.

At reception of the "7", the CP 240 sends back the value "0" into Byte 0/1, which means telegram complete, to the CPU. The CPU responds a "0" into Byte 0/1 to the CP 240.

Receiving "0" the CP 240 may send another telegram to the CPU.

Overview of M-Bus telegrams

Overview

M-Bus differentiates the following 4 telegrams:

- Single character
  The single character serves the acknowledgment of correctly received telegrams (syntax and check sum)
- Short frame
  For a short frame telegram you always have to define 3 bytes. The M-Bus takes them as telegrams of the CP 240 to a slave, e.g.:
  - SND_NKE: initialize counter
  - REQ_UD2: request counter data
- Control frame
  The control frame requires 4 defined bytes. Herewith you may send M-Bus control commands like e.g.:
  - set baud rate of the slave
  - execute slave reset
- Long frame
  The long frame contains the user data and has consequently a variable length. Here the user data may be sent in both directions, e.g.:
  - send user data to the slave
  - select slave via secondary address
  - set date and time of a slave
  - select data area to read from

Please regard that every M-Bus telegram has to be prefixed by one byte that specifies the baud rate to use. At error-free reception, the Byte 0 of the receive DB contains 00h. A value <> 00h indicates an error.

At the transfer of a M-Bus telegram to the CP 240, the telegram type is automatically detected; the data is automatically included into the according telegram structure and put out via M-Bus. With the call of the VIPA handling blocks exclusively the following data (marked green) must be transferred, depending on the telegram. Here the sum of these bytes has to be set as length value.

Single charact.	**Short frame**	**Control frame**	**Long frame**
Baud rate	Baud rate	Baud rate	Baud rate
E5h	10h Start	68h Start	68h Start
	C field	L field = 3	L field
	A field	L field = 3	L field
	Check sum	68h Start	68h Start
	Stop 16h	C field	C field
		A field	A field
		CI field	CI field
		Check sum	User Data (0...252Byte)
		Stop 16h	
			Check sum
			Stop 16h
Length for handling block: 2	Length for handling block: 3	Length for handling block: 4	Length for handling block: 5...n

Baud rate

Every M-Bus telegram has to be prefixed with one byte that specifies the baud rate. The following baud rates are supported:

Hex value	Baud
B8h	300
BBh	2400
BDh	9600

If none of the mentioned values is entered in the 1st byte, 2400Baud are automatically used.

C field

Via the C field the function of a telegram is defined. It enables also to differentiate between the call and response direction on connection level.

Depending on the direction, the C field has the following structure:

Send	0	1	FCB	FCV	F3	F2	F1	F0
Receive	0	0	ACD	DFC	F3	F2	F1	F0

Functions

With M-Bus, the following functions are defined:

Name	**C field binary**	**C field hex**	**Telegram**	**Description**
SND_NKE	0100 0000	40	short frame	This causes an initialization of the slaves (terminal equipment) and correlates a deletion of the FCB bits and an acknowledgment by the single character E5h.
SND_UD	01*F*1 0011	53/73	long / control frame	With that user data may be send to slaves.
REQ_UD2	01*F*1 1011	5B/7B	short frame	This function summons a slave to answer with data class 2 (e.g. counter values). If the slave has no such data it responds with a single character. Otherwise it sends a RSP_UD. At a defective transfer an answer lacks.
REQ_UD1	01*F*1 1010	5A/7A	short frame	This allows you to summon a slave to answer with data class 1 (e.g. alarm protocols). If the slave has no such data it responds with a single character. Otherwise it sends a RSP_UD. At a defective transfer an answer lacks.
RSP_UD	00*AD* 1000	08/18/28/38	long / control frame	Data transfer after request (slave respond)

F: FCB bit, A: ACD bit, D: DFC bit

FCB bit

The FCB bit alternates at successful communication. An unchanging FCB summons the terminal device to send the last telegram again. The lack of an answer from a slave is accounted after 330 bit times plus 50ms. The master firstly assumes that an error occurred in the connection level. It repeats the transfer of the same telegram for up to two times. If there is still no answer from the slave, the bus receives a pause of 33 bit times.

The same procedure starts when the master receives a defective respond from the slave.

Baud rate	33x	330x
300	110	1100
2400	13.8	137.5
9600	3.4	34.4

Bit time in ms

A field

For the addressing of the slaves the values 1 to 250 are available. Not configured (new) slaves have the primary address 0.

The addresses 254 and 255 are to be used as broadcast address. Using 255, the master sends information to all participants, but does not receive an answer. Using 254, every slave answers with its address. If you are having more than one slave this causes collisions. Therefore the address 254 should exclusively be used for test purposes.

The address 253 shows a secondary addressing. The addresses 251 and 252 are reserved for future extensions.

CI field

The CI field defines the purpose of the transmitted telegram. The data fields are always sent with the lowest value byte first (LSB first).

Slave → Master

CI field	Application	Defined at
70h	Sending of an error state	User group March ´94
71h	Sending of an alarm state	User group March ´94
72h	Respond with variable data structure	EN1434-3
73h	Respond with fix data structure	EN1434-3

Master → Slave

CI field	Application	Defined at
51h	Sending data	EN1434-3
52h	Select slave	User group July ´93
50h	Reset on application level	User group March ´94
54h	Synchronize slave	-
B8h	Set baud rate 300	User group July ´93
B9h	Set baud rate 600	User group July ´93
BAh	Set baud rate 1200	User group July ´93
BBh	Set baud rate 2400	User group July ´93
BCh	Set baud rate 4800	User group July ´93
BDh	Set baud rate 9600	User group July ´93
BEh	Set baud rate 19200	-
BFh	Set baud rate 38400	-
B1h	Output RAM content	Techem suggestion
B2h	Write RAM content	Techem suggestion
B3h	Start calibration test mode	User group July ´93
B4h	Read EEPROM	Techem suggestion
B6h	Start software test	Techem suggestion
90h ... 97h	Reserved	

Check sum

Check sum serves the recognition of transfer and synchronization errors. Whereat the check sum is evaluated over the following data bytes: C field, A field, CI field (if present) and user data (if present).

Examples

The following examples show how a telegram is build-up from predefined data and how the received telegram is stored in the data block.

Send data

Assignment via DB

BBh	Baud rate
7Bh	C field
FEh	A field

→

Telegram via M-Bus

10h	Start
7Bh	C field: REQ_UD2
FEh	A field: PtP Broadcast
79h	Check sum of C and A field
16h	Stop

ANZ for handling block FC0: 3

Baud rate

Please take care to prefix one byte to every M-Bus telegram that specifies the baud rate to use.

Receive data

Telegram via M-Bus

68h	68h Start
00h	L field
03	L field
68h	68h Start
08h	C field: RSP_UD
02h	A field: address 02h
72h	CI field: resp. variable
01h	Data field
02h	
03h	
75h	Check sum of C,A,CI, data
16h	Stop

→

storage in DB:

00h	Response
08h	C field: RSP_UD
02h	A field: address 02h
72h	CI field: response variable length
01h	Data field
02h	
03h	

ANZ for handling block FC1: 4

Response byte

At error-free reception, Byte 0 of the receive-DB contains the value 00h. A value <> 00h indicates an error.

Here the bits have the following allocation:

Bit 0: set if no answer was received

Bit 1: set in the case of short-circuit at the bus

Bit 2... 7: reserved

Example for M-Bus deployment

Overview

In the following example a M-Bus communication (send and receive) is build-up. Furthermore the example shows how you may easily control the communication processes by using the handling blocks.

At need you may receive the example project from VIPA.

Requirements

For the example, the following components are required:

1 System 200V consisting of CPU 21x and CP 240 M-Bus

1 acquisition device with M-Bus interface

Project engineering tool SIMATIC Manager from Siemens with transfer cable

Approach

Build-up the System 200V.

Load the example project, adjust the periphery address if necessary and transfer the project to the CPU.

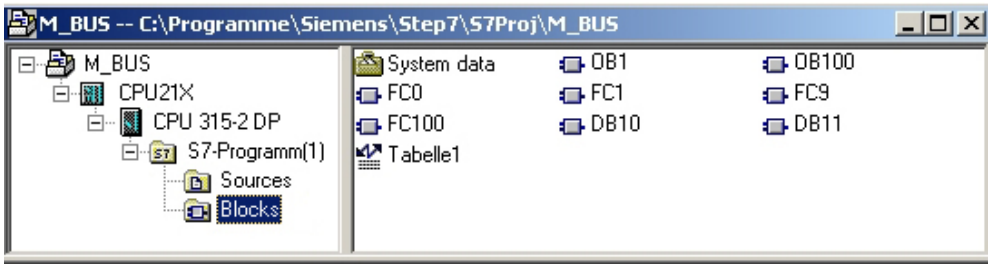
Dearchive the project

Follow these steps in the Siemens SIMATIC Manager:

- Start the Siemens SIMATIC Manager.
- To extract the file MBUS.zip select **File** > *de-archive*.
- Choose the example file MBUS.zip and set "s7proj" as destination directory.
- Open the extracted project.

Project structure

The project already contains the PLC application and the hardware configuration and has the following structure:



Data blocks The example uses the following data blocks:

DB10
SEND data block

Addr.	Name	Type	Comment
0.0		STRUCT	
+0.0	Sendefach	STRUCT	Send data block
+0.0	Baudrate	BYTE	B8h=300, BBh=2400, BDh=9600
+1.0	OK / C-Feld	BYTE	E5h=OK / C field
+2.0	A-Feld	BYTE	A field
+3.0	CI-Feld	BYTE	CI field
+4.0	User-Data Byte 0	BYTE	

...

+252.0	User-Data Byte 247	BYTE	Transfer complete
+253.0	Reserve	BYTE	
+254.0	Anzahl	WORD	Transmitting length
+256.0	gesendet	WORD	Already sent data
+258.0	Byte_Zaehler	WORD	Transmitting length (internal)
+260.0	Kom_Ende	BOOL	Telegram transmitted completely
+260.1	LB	BOOL	Last block has been sent
+260.2	SL	BOOL	Still transmitting
+260.3	Fehl	BOOL	Error during transmission
+260.4	Senden_Start	BOOL	Start bit
+261.0	PAFE	BYTE	Parameterization error
=262.0		END_STRUCT	

DB11
RECEIVE data block

Addr.	Name	Type	Comment
0.0		STRUCT	
+0.0	Data	ARRAY [0..100]	
*1.0		Byte	
+102.0	Anzahl	WORD	Amount of received data
+104.0	empfangen	WORD	Already received data
+106.0	Byte_Zaehler	WORD	Amount of received bytes (internal)
+108.0	Empf_laeuft	BOOL	Still receiving
+108.1	LB	BOOL	Last block has been received
+108.2	Fehl	BOOL	Error during reception
+108.3	Reserve	BOOL	
+108.4	Empfang fertig	BOOL	Reception ready
+109.0	PAFE	BYTE	Parameterization error
=110.0		END_STRUCT	

PLC program The PLC application has the following structure:

OB1

```
CALL       FC    9              //SYNCHRON_RESET
 ADR       :=256                //Module address
 TIMER_NR  :=T8                 //Delay time for CP
 ANL       :=M3.0               //CPU start complete
 NULL      :=M3.1               //Internal flag
 RESET     :=M3.2               //Initialize reset at CP
 STEUERB_S:=DB10.DBB260         //Control bits for Send
 STEUERB_R:=DB11.DBB108         //Control bits for Receive

U          M     3.0            //as long as Synchron active
BEB                             //no processing of CP

CALL       FC    100            //M-Bus communication
 Adr_CP    :=256                //Module address
 Baud      :=MB100              //Transfer baud rate
 C_Field   :=MB101              //Transfer value C field
 A_Field   :=MB102              //Transfer value A field
 CI_Field  :=MB103              //Transfer value CI field
 Data      :=MB104              //Transfer telegram length
 RET_VAL   :=MW106              //Return value
 Senden_Start:=M99.0            //Start order

U          M     99.0           //Order running
BEB

L          MW    106            //Return of send function
L          W#16#2000            //Ready without error
==I
SPB        copy

NOP        0                    //Error evaluation
BEA

copy:  L          0             //Delete ready without error
       T          MW    106
       L          MB    102     //Participant address
       L          20            //Basic No. for the data-DBs
       +I
       T          MW    50      //Data block no. for data storage
       L          0             //1st byte to copy
       T          MW    188     //Predefine byte counter

loo:   L          MW    188     //Load byte counter
       SLW        3             //x8 is byte address in DB
       T          MD    184     //Save address
       AUF        DB    11      //Open receive buffer
       L          DBB   [MD 184] //Value from receive buffer
       AUF        DB    [MW 50]
       T          DBB   [MD 184] //Store in data-DB

       L          MW    188
       +1                       //Increase byte number
       T          MW    188
       L          DB11.DBW  102 //Last byte to copy
       <=I                      //not all bytes copied yet
       SPB        loo           //then resume
```

OB 100

```
UN         M     3.0
S          M     3.0            //Set start-up ID
```

FC 100

This function sends a request to a M-Bus participant and receives the answer. The transmit data has to be entered into DB 10 starting with data byte 4 before calling the function.

```
      UN        #Senden_Start
      BEB
      U         DB11.DBX    108.7          //Waiting for acknowledgment
      SPB   REC
      NOP       0                          //Enter transmit data into send buffer
      L         #Baud                      //1st send byte is baud rate
      T         DB10.DBB    0
      L         #C_Field                   //2nd send byte is C_Field
      T         DB10.DBB    1
      L         #A_Field                   //3rd send byte is A_Field
      T         DB10.DBB    2
      L         #CI_Field                  //4th send byte is CI_Field
      T         DB10.DBB    3              //At Long Frame data must be entered
      NOP   0                              //from user data on before FC call
      SET                                  //
      S         DB10.DBX    260.4          //Set send release
      L         0
      L         #Data                      //Telegram length at Long Frame
      <>I
      +4
      T         DB10.DBW    254            //Telegram length for Long Frame
      SPB       send
      L         0
      L         #CI_Field                  //ID for Control Frame
      <>I
      L         4                          //Telegram length for Control Frame
      SPB       sen1
      L         3                          //Telegram length for Short Frame
sen1: T         DB10.DBW    254            //Telegram length
send: CALL      FC    0                    //Block send
       ADR      :=#Adr_CP                  //Module address
       _DB      :=DB10                     //DB send buffer
       ABD      :=W#16#0                   //1st data byte to send
       ANZ      :=DB10.DBW254              //Amount of send data
       PAFE     :=DB10.DBB261              //Error byte
       FRG      :=DB10.DBX260.4            //Send release
       GESE     :=DB10.DBW256              //Internal variable
       ANZ_INT  :=DB10.DBW258              //Internal variable
       ENDE_KOM :=DB10.DBX260.0            //Internal variable
       LETZTER_BLOCK:=DB10.DBX260.1        //Internal variable
       SENDEN_LAEUFT:=DB10.DBX260.2        //Internal variable
       FEHLER_KOM   :=DB10.DBX260.3        //Internal variable
      U         DB10.DBX    260.4          //Transmission still running
      BEB                                  //then end
      S         DB11.DBX    108.7          //Waiting for acknowledgment
REC:  NOP       0
      CALL      FC    1
       ADR      :=#Adr_CP                  //Module address
       _DB      :=DB11                     //DB receive buffer
       ABD      :=W#16#0                   //1st data byte receive buffer
       ANZ      :=DB11.DBW102              //Amount of received bytes
       EMFR     :=DB11.DBX108.4            //Telegram received completely
       PAFE     :=DB11.DBB109              //Error byte
       GEEM     :=DB11.DBW104              //Internal variable
       ANZ_INT  :=DB11.DBW106              //Internal variable
       EMPF_LAEUFT    :=DB11.DBX108.0      //Internal variable
       LETZTER_BLOCK  :=DB11.DBX108.1      //Internal variable
       FEHL_EMPF      :=DB11.DBX108.2      //Internal variable
      UN        DB11.DBX    108.4          //No new value received yet
      BEB
      R         DB11.DBX  108.4
      R         DB11.DBX  108.7
      R         #Senden_Start
      L         DB11.DBW  102
      L         1                          //If received only 1Byte -> error
      ==I
      SPB       Fehl
      L         W#16#2000                  //After respond reception, delete
Ende: T         #RET_VAL                   //start bit and return ID to RET_VAL
      BEA
Fehl: L         DB11.DBB    0              //Received byte
      L         1                          //No response from M-Bus slave
      ==I
      L         W#16#8001                  //Error ID for no response
      SPB       Ende
      L         W#16#80FF                  //Undefined response from CP
      SPA       Ende
```

Technical data

CP 240 with M-Bus interface

Electrical Data	VIPA 240-1FA20
Number of channels	1
Number of connectable slaves	6
Power supply	5V via back plane bus
Current consumption	max. 300mA
External power supply	-
Status monitor	via LED at the front side
Bus voltage pause level	30V
Max. bus negative bias	9mA
Rigid bit threshold	11mA
Short circuit firmness	permanent
Shutdown level at overcurrent	50mA
Minimum shutdown time	50ms
Thevenin resistance	<100Ω
Galvanical separation to M-Bus	yes
Connectors/interfaces	2pin jack for M-Bus
Transfer rate	300, 2400, 9600Baud
Programming data	
Input data	16Byte
Output data	16Byte
Parameter data	16Byte
Diagnosis data	-
Dimensions and Weight	
Dimensions (WxHxD) in mm	25.4x76x78
Weight	80g

Appendix

A Index

M.Stich