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 CPU 21x

Order No.: VIPA HB103E
Rev. 05/45

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

Ohmstraße 4, D-91074 Herzogenaurach,
Tel.: +49 (91 32) 744 -0
Fax.: +49 (91 32) 744-144
EMail: info@vipa.de
http://www.vipa.de

Hotline: +49 (91 32) 744-114

Disclaimer of liability

The contents of this manual was carefully examined to ensure that it conforms with the described hardware and software.
However, discrepancies can not be avoided. The specifications in this manual are examined regularly and corrections will be included in subsequent editions.
We gratefully accept suggestions for improvement.

Trademarks

VIPA® is a registered trademark of VIPA Gesellschaft für Visualisierung und Prozessautomatisierung mbH

STEP® is a registered trademark 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 operation of the CPU 21x in the System 200V from VIPA. The text provides details on the hardware, the programming and the functions integrated into the unit as well as Profibus and Ethernet applications.

Outline

Chapter 1: Principles

This introduction includes recommendations on the handling of the module as well as information about applications and implementation for CPU modules. You may also read details about the mode of operation of the CPU 21x.

Chapter 2: Hardware description

Different versions of the CPU are available This chapter describes these versions in detail.

Chapter 3: Deployment of the CPU 21x

This chapter describes the deployment of the CPU section together with the peripheral modules of the System 200V that are installed on the same bus rail.

Chapter 4: Deployment of the CPU 21x-2BT10 under TCP/IP

This chapter describes applications of the CPU 21x-2BT10 and the communication of the CP using TCP/IP.
The CP is parameterizable with NetPro from Siemens.

Chapter 5: Deployment of the CPU 21x-2BT02 under H1 / TCP/IP

This chapter describes applications of the CPU 21x-2BT02 and the H1 resp. TCP/IP communication procedure of the CP.
The CP is parameterizable with WinNCS.

Chapter 6: Deployment of the CPU 21xDPM

Content of this chapter is the deployment and project engineering of the CPU 21xDPM with integrated Profibus-DP master. Besides the description of the DP master operating modes you will also find information about the start-up behaviour and the commissioning.

Chapter 7: Deployment of the CPU 21xDP

Topic of this chapter is the CPU 21xDP (intelligent slave). You will find the description of the deployment, configuration and parameter definition for the CPU 21xDP under Profibus.

Chapter 8: Deployment of the CPU 21xCAN

Content of this chapter is the employment of the CPU 21xCAN under CANopen. Here you'll find all information required for the usage of the integrated CAN master.

Chapter 9: Deployment of the CPU 21xSER-1

Content of this chapter is the deployment of the CPU 21xSER-1 with RS232C/RS485 interface..

Chapter 10: Deployment of the CPU 21xSER-2

Content of this chapter is the deployment of the CPU 21x-2BS02 with two RS232C interfaces.

Chapter 11: Integrated OBs, SFBs, SFCs

Here you find the description of the integrated VIPA-specific SFCs, like e.g. the SFCs for the CP communication.

Chapter 12: Command list

This chapter lists all available commands of the CPU in alphabetical order.

Contents

User considerations

Objective and contents

This manual describes the CPU 21x as well as all the versions of the product. It contains a description of the construction, project implementation and the application of the product.

The CPU 21x is compatible with all System 200V components of VIPA.

Target audience

The manual is targeted at users who have a background in automation technology and PLC-programming.

Structure of the manual

This manual consists of 12 chapters. Every chapter provides the description of one specific topic.

Guide to the document

This manual provides the following guides:

- 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

Application specifications

The CPU 21x is constructed and manufactured for

- all VIPA System 200V components
- communication and process control
- general control and automation tasks
- 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

- 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 Principles

Outline

This introduction contains references on the handling and information about application areas and usage of the CPU modules.

It also provides certain suggestions on the approach when programming the module and the CPU specifications that are important in this respect.

Below follows a description of:

- Safety information for users
- Construction and operation of the CPU 21x
- Programming principles

Contents

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 may 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 electrostatically sensitive equipment.

It is possible that electrostatically sensitive equipment is destroyed by energies and voltages that are far less than the human threshold of perception. These voltages may occur where persons do not discharge themselves before handling electrostatically sensitive modules and they may damage components thereby causing the module to become inoperable or unusable. Modules that have been damaged by electrostatic discharge are usually not detected immediately. The respective failure may become apparent after a period of operation.

Components damaged by electrostatic discharges can fail after a temperature change, mechanical shock or changes in the electrical load.

Only the consistent implementation of protective devices and meticulous attention to the applicable rules and regulations for handling the respective equipment is able to prevent failures of electrostatically sensitive modules.

Shipping of modules

Please ship the modules exclusively 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.

You should only use soldering irons with grounded tips when you are making modifications on electrostatically sensitive modules.

Attention!

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

Hints for the deployment of the MPI interface

What is MP²I?

The MP²I jack combines 2 interfaces in 1:

- MP interface
- RS232 interface

Please regard that the RS232 functionality is only available by using the Green Cable from VIPA.

Deployment as MP interface

The MP interface provides the data transfer between CPUs and PCs. In a bus communication you may transfer programs and data between the CPUs interconnected via MPI.

Connecting a common MPI cable, the MPI jack supports the full MPI functionality.

Important notes for the deployment of MPI cables!

Deploying MPI cables at the CPUs from VIPA, you have to make sure that Pin 1 is not connected. This may cause transfer problems and in some cases damage the CPU!

Especially Profibus cables from Siemens, like e.g. the 6XV1 830-1CH30, must not be deployed at MP²I jack.

For damages caused by nonobservance of these notes and at improper deployment, VIPA does not take liability!

Deployment as RS232 interface only via "Green Cable"

For the serial data transfer from your PC, you normally need a MPI transducer. Fortunately you may also use the "Green Cable" from VIPA. You can order this under the order no. VIPA 950-0KB00.

The "Green Cable" supports a serial point-to-point connection for data transfer via the MP²I jack exclusively for VIPA CPUs.

Please regard the hints for the deployment of the "Green Cable" on the following page.

Hints for the Green Cable from VIPA

What is the Green Cable?

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

The Green Cable is a programming and download cable for VIPA CPUs with MP^{2}I jack and VIPA fieldbus masters. The Green Cable from VIPA is available under the order no. VIPA 950-0KB00.

The Green Cable allows you to:

- *transfer projects serial*
  Avoiding high hardware needs (MPI transducer, etc.) you may realize a serial point-to-point connection via the Green Cable and the MP^{2}I jack. This allows you to connect components to your VIPA-CPU that are able to communicate serial via an MPI adapter like e.g. a visualization system.
- *execute firmware updates of the CPUs and fieldbus masters*
  Via the Green Cable and an upload application you may update the firmware of all recent VIPA CPUs with MP^{2}I jack and certain fieldbus masters (see Note).

Important notes for the deployment of the Green Cable

Nonobservance of the following notes may cause damages on system components.

For damages caused by nonobservance of the following notes and at improper deployment, VIPA does not take liability!

Note to the application area

The Green Cable may exclusively deployed directly at the concerning jacks of the VIPA components (in between plugs are not permitted). E.g. a MPI cable has to be disconnected if you want to connect a Green Cable.

At this time, the following components support the Green Cable:

VIPA CPUs with MP^{2}I jack and fieldbus master from VIPA.

Note to the lengthening

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

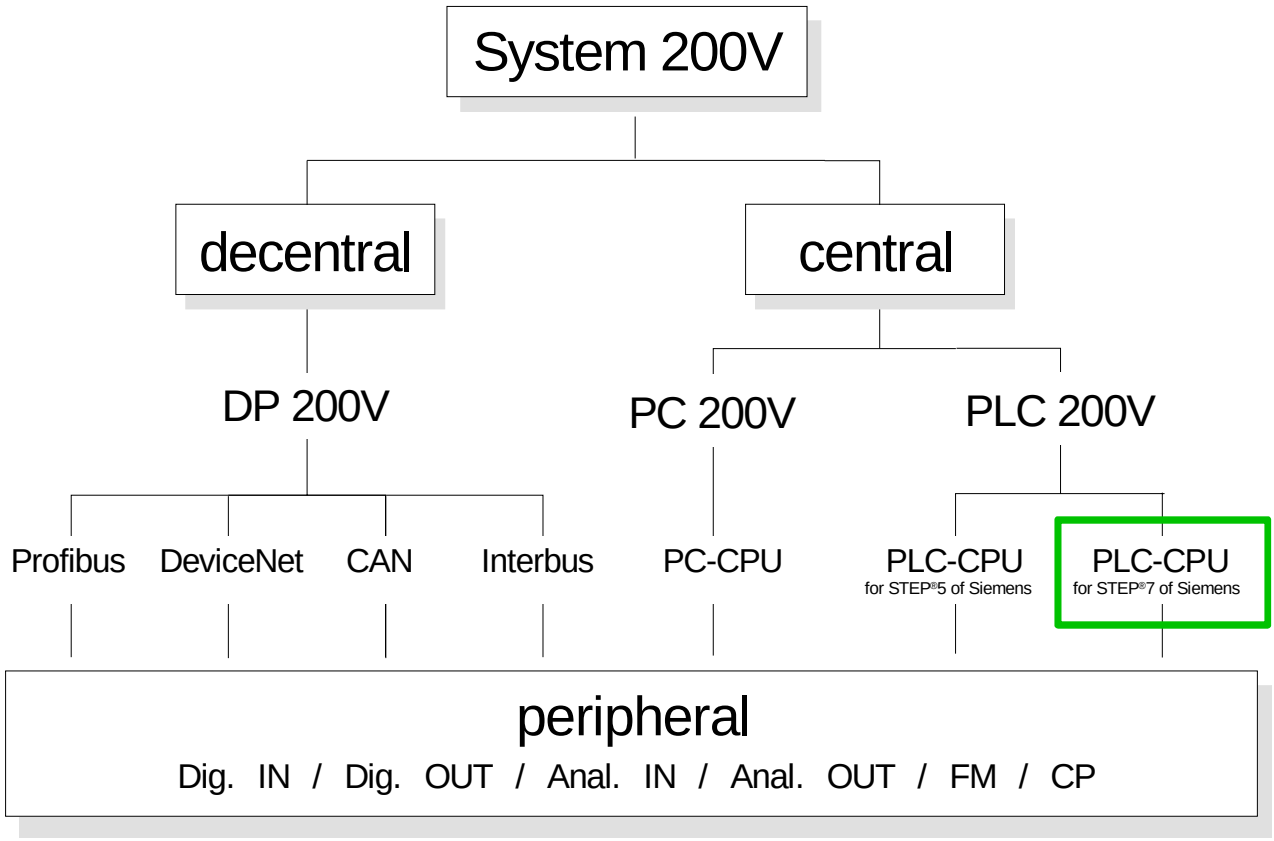
The Green Cable may only be lengthened with a 1:1 cable (all 9 Pins are connected 1:1).

Overview System 200V

The System 200V

The System 200V is a modular automation system for low and middle range of performance that you may use either centralized or decentralized. The single modules are directly clipped to a 35mm DIN rail and are connected together with the help of special bus clips.

The following picture shows the range of performance of the System 200V:



Overview Manuals

The current manual describes the PLC-CPU family CPU 21x compatible to STEP®7 by Siemens.

The description of the PLC-CPU-family CPU 24x compatible to STEP®5 by Siemens is available in the manual No. HB99.

The peripheral modules, PCs and decentralized peripheral equipment are to find in the manual HB97 "System 200V". This manual also contains hints for installation and commissioning of the System 200V.

Function security of the VIPA CPUs

The CPUs include security mechanisms like a Watchdog (100ms) and a parameterizable cycle time surveillance (parameterizable min. 1ms) that stop res. execute a RESET at the CPU in case of an error and set it into a defined STOP state.

The VIPA CPUs are developed function secure and have the following system properties:

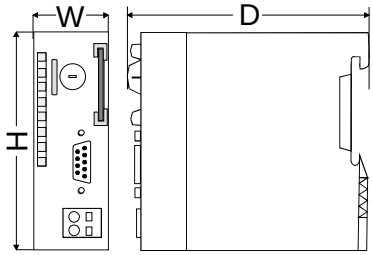
Event	concerns	Effect
RUN → STOP	general	BASP (**B**efehls-**A**usgabe-**Sp**erre, i.e. command output lock) is set.
	central digital outputs	The outputs are set to 0V.
	central analog outputs	The voltage supply for the output channels is switched off.
	decentral outputs	The outputs are set to 0V.
	decentral inputs	The inputs are read constantly from the slave and the recent values are put at disposal.
STOP → RUN res. Power on	general	First the PII is deleted, the call of the OB100 follows. After the execution of the OB, the BASP is set back and the cycle starts with: Delete PIO → Read PII → OB1.
	central analog outputs	The behavior of the outputs at restart can be preset.
	decentral inputs	The inputs are read constantly from the slave and the recent values are put at disposal.
RUN	general	The program execution happens cyclically and can therefore be foreseen: Read PII → OB1 → Write PIO.

PII: = Process image inputs

PIO: = Process image outputs

General description of the System 200V

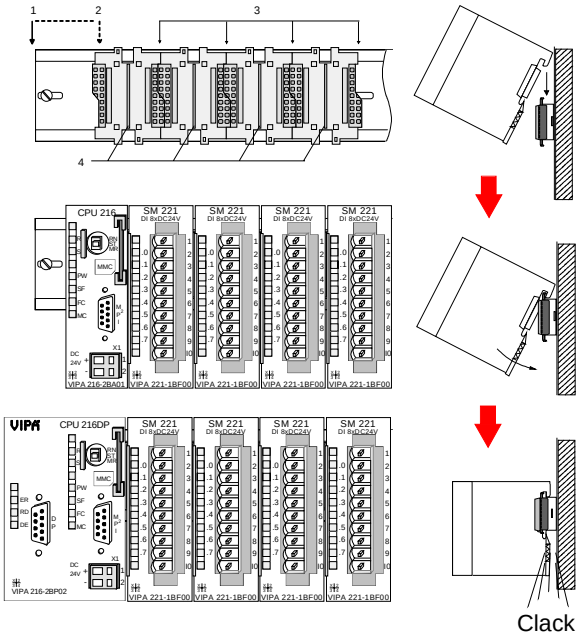
Structure / Dimensions



- Norm profile head rail 35mm
- Peripheral modules with labeling strip
- Measurements basic module:
  Single width: (HxWxD) in mm: 76x25.4x80; in inches: 3x1x3
  Double width: (HxWxD) in mm: 76x50.8x80; in inches: 3x2x3

Installation

Please note, that you have to plug in the CPU only at plug-in location 1 resp. 1 and 2 (if double width).



[1] CPU if double width
[2] CPU single width
[3] peripheral modules
[4] leading bars

Operating security

- Plug in via CageClamps at the front-facing connector, core cross-section 0.08...2.5mm^2 resp. 1.5 mm^2 (18pin plug)
- Total isolation of the cables during module changes
- Potential separation of all modules to the backplane bus
- EMC-proof ESD/Burst acc. EN61000-4-2/EN61000-4-4 to Level 3: 8kV/2.5kV
- Shock resistance acc. IEC 60068-2-6 / IEC 60068-2-27 (1G/12G)

Environmental conditions

- Operating temperature: 0... +60°C
- Storage temperature: -40... +85°C
- Relative humidity: 95% without condensation
- fan-less operation

Overview CPU 21x

General

A CPU is an intelligent module. Your controlling programs are executed here. You are able to select one of three CPUs, depending on the performance required from your system. The higher the performance of the CPU, the more user memory is available.

The CPUs 21x are intended for small to medium applications and have an integrated 24V power supply. The CPUs contain a standard processor with internal program memory for the storage of user-programs. In addition, every CPU 21x is equipped with a socket for a memory module, which is located on the front.

Every CPU has an MPI-interface and is instruction set compatible with STEP®7 from Siemens. The CPU series 214...216 have similar performance as the Siemens STEP®7 CPU series.

This series of CPUs provides access to the peripheral modules of the System 200V. You may query sensors and control actuators by means of standardized commands and programs. The unit can address a maximum of 32 modules. You may parameterize the CPU via the integrated MPI-interface.

Products

The VIPA CPUs 21x are available in three different performance categories with 8 versions each:

- **CPU 21x** PLC-CPU
- **CPU 21x-2BT02** PLC-CPU with Ethernet interface with H1 / TCP-IP
- **CPU 21x-2BT10** PLC-CPU with Ethernet interface with TCP-IP
- **CPU 21xDP** PLC-CPU with Profibus slave
- **CPU 21xDPM** PLC-CPU with Profibus master
- **CPU 21xSER-1** PLC-CPU with 1 serial interface
- **CPU 21xSER-2** PLC-CPU with 2 serial interfaces
- **CPU 21xCAN** PLC-CPU with CANopen master

In all three performance categories, the CPUs 21x are available as CPU 214, 215 and 216. Within these three performance categories, the CPUs are visually on the same lines. With rising number the scope of performance of a CPU 21x is increasing.

Note!

The remainder of this description refers to all CPUs of the VIPA CPU 21x family!

Hints for project engineering

Outline

For the project engineering of the CPU 21x and the other System 200V modules connected to the same VIPA bus, you use the hardware configurator from Siemens.

To address the directly plugged peripheral modules, you have to assign a special address in the CPU to every one.

The address allocation and the parameterization of the modules takes place in the SIMATIC manager from Siemens in form of a virtual Profibus system. For the Profibus interface is standardized software sided, the functionality is guaranteed by including a GSD-file into the SIMATIC manager from Siemens.

Transfer your project into the CPU via the MPI interface.

Preconditions

For the project engineering of your CPU 21x the following requirements have to be fulfilled:

- SIMATIC manager from Siemens is installed at your PC resp. PG.
- The GSD-file for the System 200V is included to the hardware configurator. Actual GSD files can be found at ftp://ftp.vipa.de/support.
- serial connection to the CPU (e.g. via "Green Cable" from VIPA)

Note!

The configuration of the CPU requires a thorough knowledge of the Siemens SIMATIC manager and the hardware configurator!

Compatibility to SIMATIC manager from Siemens via vipa_21x.gsd GSD-file

The project engineering of a CPU 21x takes place via the SIMATIC manager from Siemens in form of a virtual Profibus system with the CPU 315-2DP as basic.

Due to the standardized Profibus interface VIPA is able to support the complete functionality of the System 200V family in the SIMATIC manager from Siemens by including the GSD-file vipa_21x.gsd.

To be compatible with the SIMATIC manager from Siemens, the following steps have to take place:

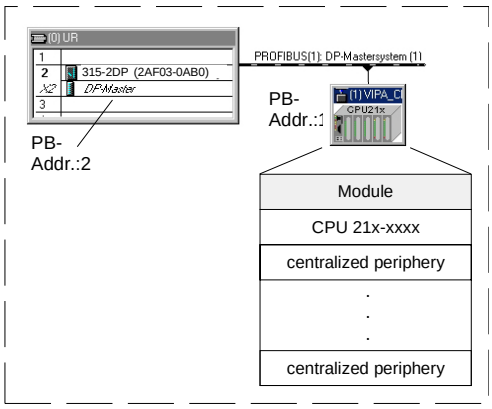
- **Project the Profibus-DP master system with CPU 315-2DP (6ES7 315-2AF03 V1.2).**
- **Add Profibus slave "VIPA_CPU21x" from the HW catalog with address 1.**
- **Include the CPU 21x at the 1st slot of the slave system.**

Project engineering CPU 21x with central periphery

The following steps are necessary to project a CPU 21x in the hardware configurator from Siemens:

- Start the hardware configurator from Siemens.
- Install the GSD-file VIPA_21x.gsd.
- Create a Profibus-DP master system with the CPU 315-2DP.
- Include the master system from the hardware catalog in your slave system "VIPA_CPU21x". You find this slave system in the hardware catalog under *Profibus-DP > Additional field devices > I/O > VIPA_System_200V.*
- Assign the address 1 to your slave system, so that the CPU is able to recognize the system as central periphery system.
- Add your modules to the slave system in the same order you have assembled them. Start with the CPU at the 1st slot.
- Include then your System 200V modules.

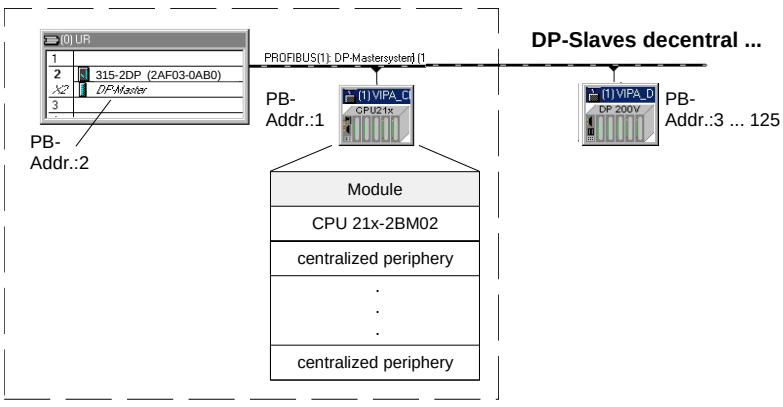
CPU 21x



Master projecting of the CPU 21xDPM

When projecting a CPU 21xDPM you include the central modules like described above. Slave systems that shall be connected to the master are added to the already existing master system:

CPU 21xDPM central



Project engineering of the CPU 21xDP in a master system

When configuring a CPU 21xDP, the central plugged-in modules are parameterized like shown above.

Slave parameterization

As intelligent slave, the Profibus section maps its data areas into the memory area of the CPU 21xDP. The assignment of the areas is fixed via the properties of the CPU 21xDP. These areas have to be provided with an according PLC program.

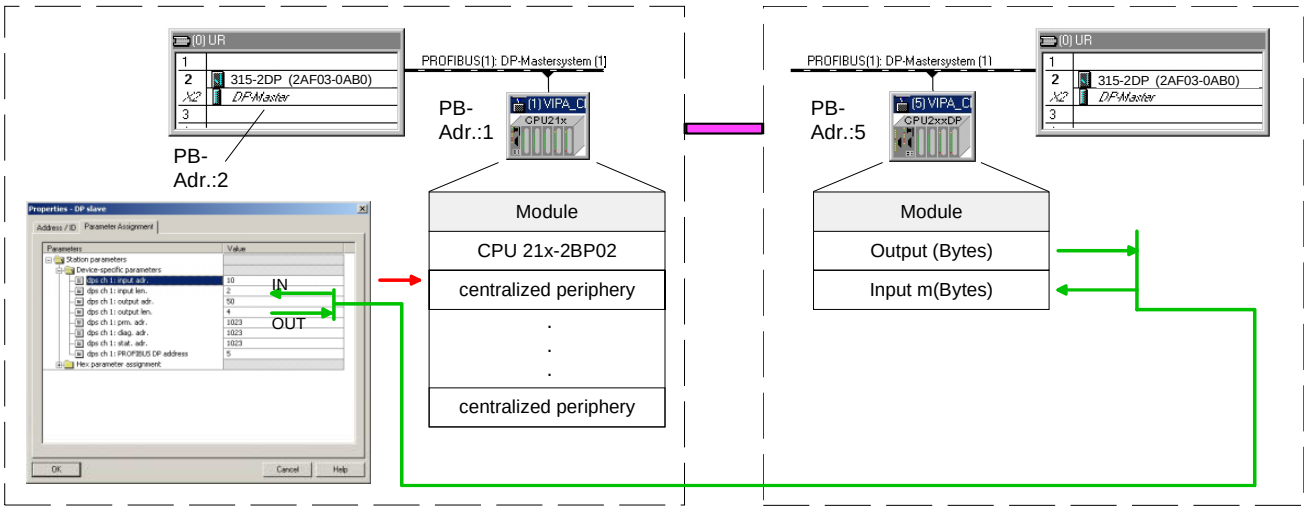
Attention!

The length values of input and output area have to be identical to the byte values of the master project engineering. Otherwise no Profibus communication is possible (slave failure).

Steps of project engineering in the DP master

- Configure the CPU with DP master system (address 2)
- Add Profibus slave VIPA_CPU2xxDP of VIPA04d5.gsd
- Assign Profibus input and output area starting with plug-in location 0

Slave: (VIPA_CPU21x of VIPA_21x.gsd) **Master: (VIPA_CPU21xDP of VIPA04d5.gsd)**



Attention!

When using an IM 208DP master at a CPU 21x with firmware version higher v3.0, you have to make sure that the master itself also has a firmware version higher v3.0; otherwise the deployment of the IM 208DP is not possible.

The according firmware version is to find on the label attached at the backside of each module.

Project engineering CPU 21xNET

The project engineering of network connections via Ethernet for the CPU 21x-2BT02 can be carried out with WinNCS from VIPA and for the CPU21x-2BT10 with SIMATIC Manager using NetPro.

Project engineering CPU 21xSER-1

The CP communication for the CPU 21x-2BS12 or CPU 21x-2BS32 is accomplished via a send buffer and a receive buffer with a size of 256Byte.

Parameterization at runtime has to be carried out with the SFC 216 (SER_CFG). For all protocols except ASCII the parameters have to be stored in a DB. Writing or reading the send buffer respectively receive buffer can be controlled with the SFC 217 (SER_SND) respectively with the SFC 218 (SER_RCV).

Project engineering CPU 21xSER-2

The CP of the CPU 21x-2BS02 is directly connected to the CPU part through a Dual-Port-RAM, also called "page frame". At the CPU this page frame is available as standard CP interface. Communication on the respective protocols is controlled by connection commands which have to be programmed in the user application.

At this the VIPA-SFCs 230 ... 238 are used.

The parameter transfer to the communication processor (CP) is done at runtime using a SEND (SFC 230) with order number 201. The parameters have to be stored in a DB whose set-up depends on the desired protocol.

To activate the parameters a RESET (SFC 234) has to be carried out with order no. 0 after the SEND.

Note!

Please regard that the commands SEND, RECEIVE, FETCH and RESET require a preceding "VKE"=1 otherwise they are not executed.

Project engineering CPU 21xCAN

The project engineering of CANopen master will be done with WinCoCT (**Win**dows **C**AN**o**pen **C**onfiguration **T**ool) from VIPA.

Please export your project as wld file from WinCoCT, which can be imported into the Hardware Configurator from Siemens. Therefor please start a virtual Profibus system "VIPA_CPU21x" and integrate the CPU 21xCAN (VIPA 21x-2CM02) at slot 0.

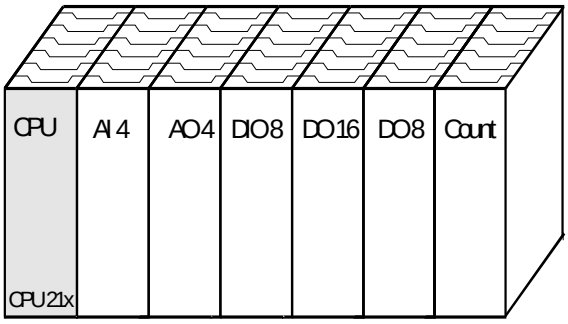
Application fields

Overview

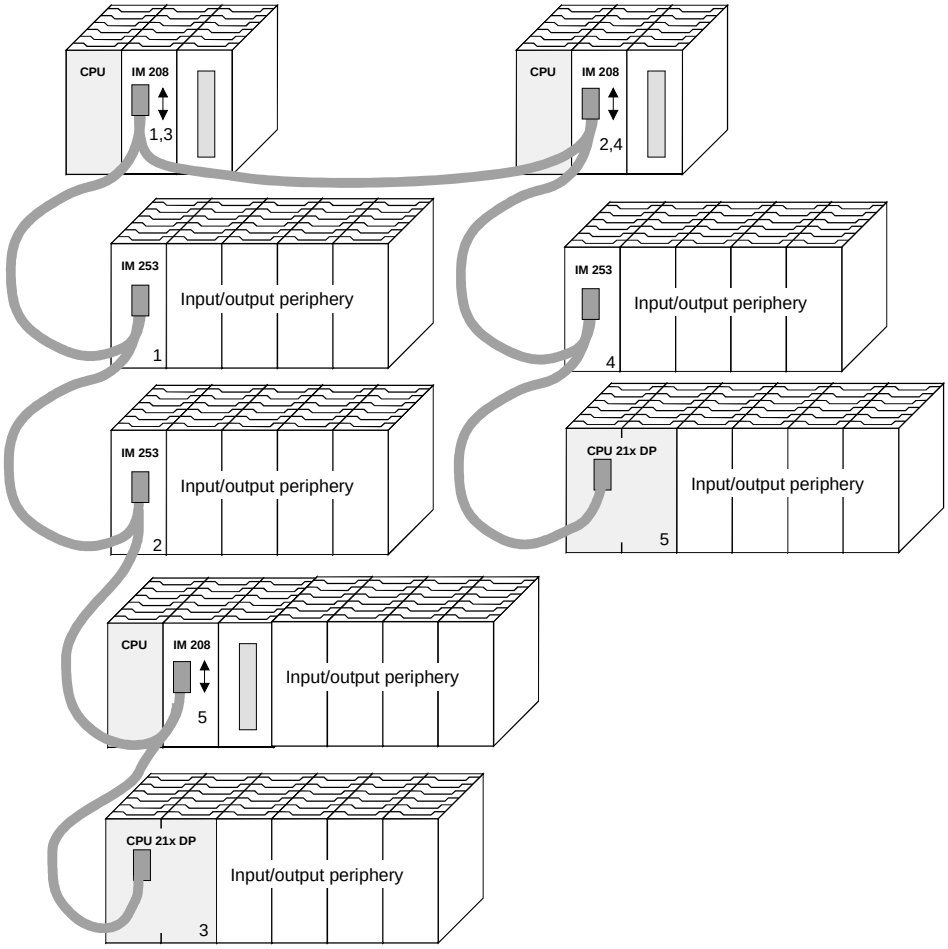
This series of CPU modules provides access to the peripheral modules of the VIPA System 200V. You can use a set of standard commands and programs to interrogate sensors and actuators. One single CPU may address a maximum of central 32 modules.

Application example

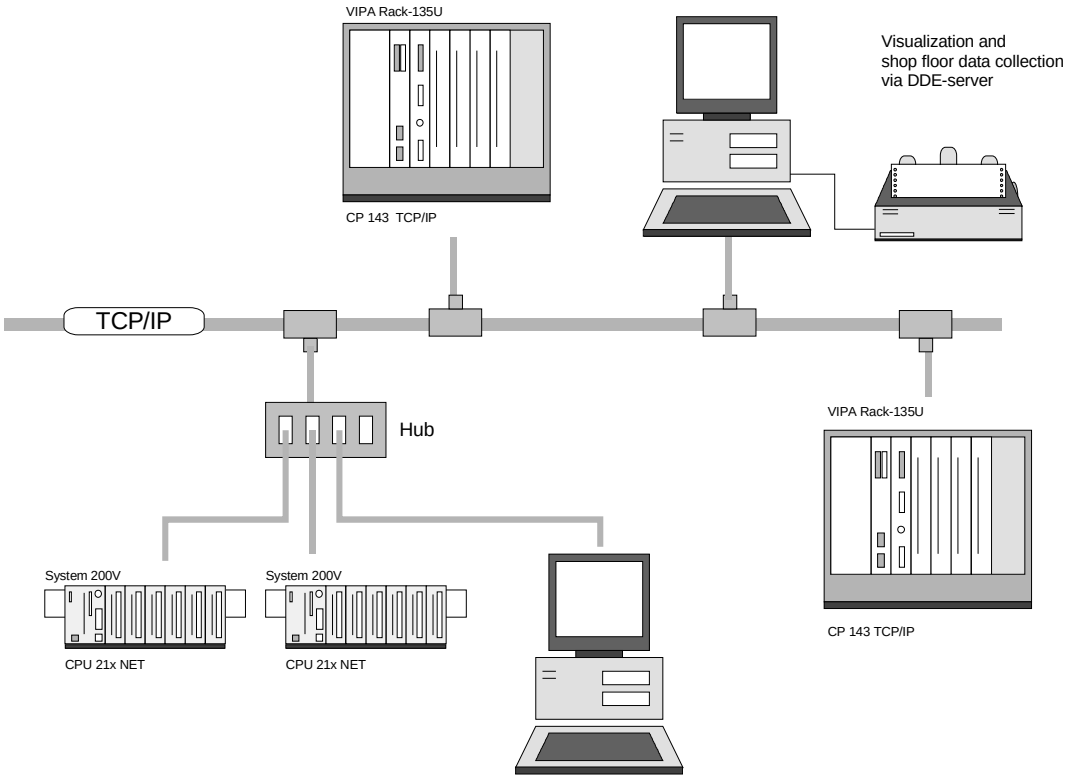
Compact centralized configuration



Decentralized configuration using Profibus



Application based on TCP/IP



Features

The PLC-CPUs are employed as program executor.

They support the input/output modules and process the data from the function modules.

- Quick programming due to the compatibility with STEP®7 from Siemens.
- The compact construction requires less space.
- Enhanced flexibility provided by up to 32 function modules (DIG I/O, ANA I/O, SSI, pulse counter, communication modules, etc.).
- Additional function modules can be added quickly and easily by means of plug-in bus extension options.
- User-friendly maintenance using a PC via MPI.
- Optional Ethernet or Profibus interface.

Operating modes of a CPU

General

These CPUs are intended for small and medium sized applications and are supplied with an integrated 24V power supply. The CPU contains a standard processor with internal program memory. In combination with the System 200V peripherals the unit provides a powerful solution for process automation applications within the System 200V family.

A CPU supports the following modes of operation:

- cyclic processing
- timer processing
- alarm controlled processing
- priority based processing

Cyclic processing

Cyclic processing represents the major portion of all the processes that are executed in the CPU. Identical sequences of operations are repeated in a never ending cycle.

Timer processing

Where a process requires control signals at constant intervals you can initiate certain operations based upon a **timer**, e.g. not critical monitoring functions at one-second intervals.

Alarm controlled operation

If a process signal requires a quick response you would allocate this signal to an **alarm controlled** procedure. An alarm may activate a procedure in your program.

Priority based processing

The above processes are handled by the CPU in accordance with their **priority**. Since a timer or an alarm event requires a quick reaction the CPU will interrupt the cyclic processing when these high-priority events occur to react to the event. Cyclic processing will resume once the reaction has been processed. This means that cyclic processing has the lowest priority.

CPU 21x programs

Overview

The program that is present in every CPU is divided as follows:

- System routine
- User program

System routine

The system routine organizes all those functions and procedures of the CPU that are not related to a specific control application.

User program

This consists of all the functions that are required for the processing of a specific control application. The operating modules provide the interfaces to the system routines.

CPU 21x operands

Overview

The following operands are available for programming the CPU 21x:

- Process image and periphery
- Bit memory/marker
- Timers and counters
- Data blocks

Process image and periphery

The user program can quickly access the process image of the inputs and outputs PII/PIO. You may manipulate the following types of data:

- individual bits
- bytes
- words
- double words

You may also gain direct access to peripheral modules via the bus from your user program. The following types of data are available:

- bytes
- words
- blocks

Bit memory

Bit memory is an area of memory that is accessible to the user program by means of certain operations. Bit memory is intended to store frequently used working data.

You may access the following types of data:

- individual bits
- bytes
- words
- double words

Timer and counter

With your program you may load a time cell with a value between 10ms and 9990s. As soon as the user program executes a start operation the value of this timer is decremented by the interval that you have specified until it reaches zero.

You may load counter cells with an initial value (max. 999) and increment or decrement this when required.

Data blocks

A data block contains constants or variables in form of bytes, words or double words. You may always access the current data block by means of operands.

You may access the following types of data:

- individual bits
- bytes
- words
- double words

Chapter 2 Hardware description

Outline

The CPUs 21x are available in different versions that are described in this chapter. In addition to the hardware description the chapter also contains installation and commissioning instructions and applications for the memory modules.

The technical data conclude the chapter.

The following section contains descriptions of:

- the components of the CPUs along with controls and displays
- the modules integrated into the CPU
- technical data

Contents

System overview

CPU versions

The CPU-21x family of products available from VIPA consists of 3 different models each with 8 versions:

- **CPU 21x** PLC-CPU
- **CPU 21x-2BT10** PLC-CPU with Ethernet interface with TCP-IP
- **CPU 21x-2BT02** PLC-CPU with Ethernet interface with H1 / TCP-IP
- **CPU 21xDPM** PLC-CPU with Profibus master
- **CPU 21xDP** PLC-CPU with Profibus slave
- **CPU 21xCAN** PLC-CPU with CAN master
- **CPU 21xSER-1** PLC-CPU with 1 serial interface
- **CPU 21xSER-2** PLC-CPU with 2 serial interfaces

All CPUs 21x are available as CPU 214, 215 and 216. The CPUs 214, 215 and 216 are functionally identical and their only difference is the memory size.

Note!

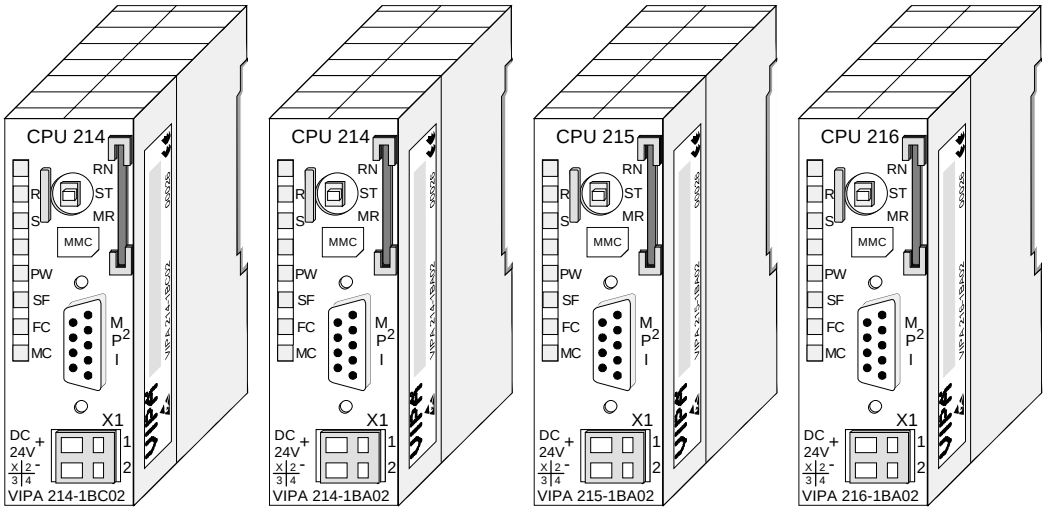
The remainder of this description refers to all CPUs of the CPU 21x family.

Properties

- Instruction set compatible to Siemens STEP®7
- Configuration by means of the Siemens SIMATIC manager
- Integrated 24V power supply
- Total address range:
  1024Byte inputs, 1024Byte outputs (128Byte process image each)
- 48...128kByte of work memory
- 80…192kByte of load memory
- Battery backed clock
- Memory-card slot
- MPI interface
- Integrated V-Bus controller for controlling System 200V peripherals
- User programs can be saved to the external memory card (MMC)
- 256 timer
- 256 counter
- 8192 Bit memories

CPU 21x

- Instruction set compatible with Siemens STEP®7
- MPI interface for data transfer between PC and CPU
- Status LEDs for operating mode and diagnostics
- External memory card (MMC)
- "On board" memory

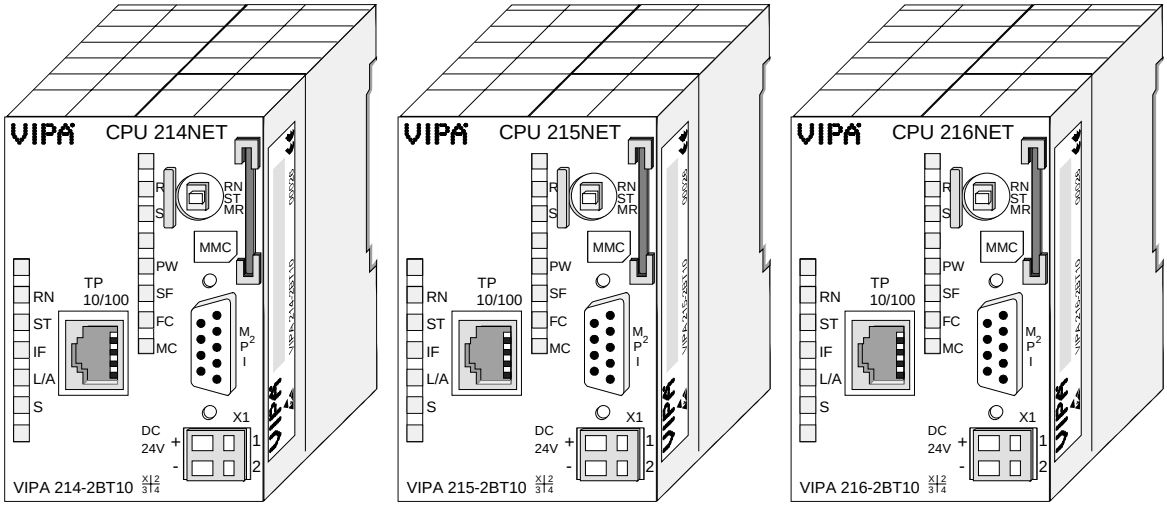


Order data CPU 21x

Type	Order number	Description
CPU 214C	VIPA 214-1BC02	PLC CPU 214 with 32/40KB of work/load memory
CPU 214	VIPA 214-1BA02	PLC CPU 214 with 48/80KB of work/load memory
CPU 215	VIPA 215-1BA02	PLC CPU 215 with 96/144KB of work/load memory
CPU 216	VIPA 216-1BA02	PLC CPU 215 with 128/192KB of work/load memory

CPU 21x-2BT10 Identical to CPU 21x, additionally with:

- Integrated Ethernet CP 243 (compatible to CP 343)
- Direct connection to twisted pair Ethernet via RJ45 jack
- Protocols TCP/IP, UDP and RFC1006
- Transfer rate 10/100MBit/s
- PG/OP channel
- CP configurable with NetPro from Siemens

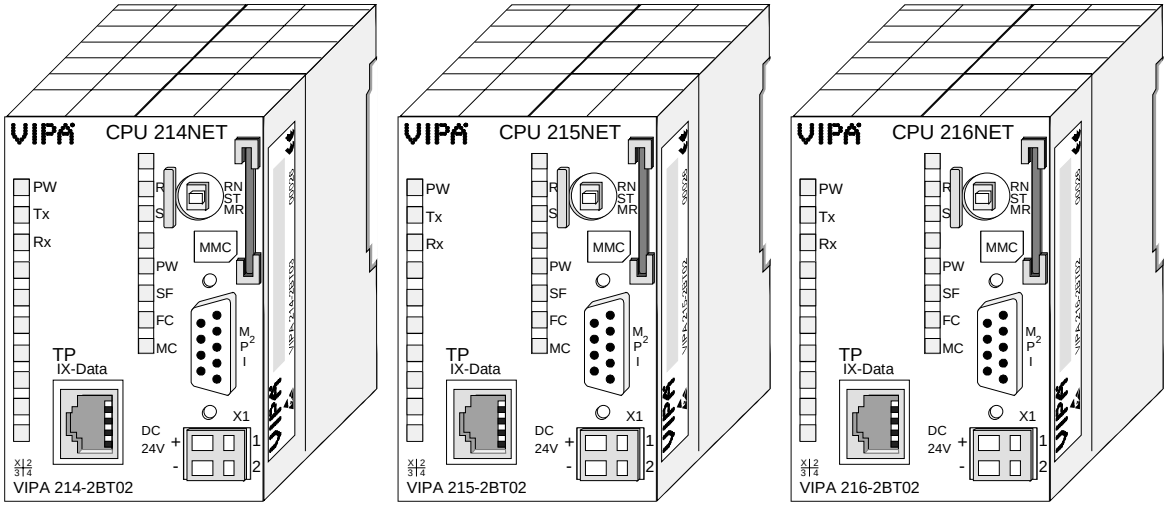


Order data CPU 21xNET

Type	Order number	Description
CPU 214NET	VIPA 214-2BT10	PLC CPU 214 with Ethernet and 48/80KB of work/load memory
CPU 215NET	VIPA 215-2BT10	PLC CPU 215 with Ethernet and 96/144KB of work/load memory
CPU 216NET	VIPA 216-2BT10	PLC CPU 216 with Ethernet and 128/192KB of work/load memory

CPU 21x-2BT02 Identical to CPU 21x, additionally with:

- Integrated Ethernet CP 243
- Direct connection of twisted pair Ethernet via RJ45 jack
- Protocols H1, TCP/IP, UDP
- Transfer rate 10MBit/s
- CP parameterizable with WinNCS

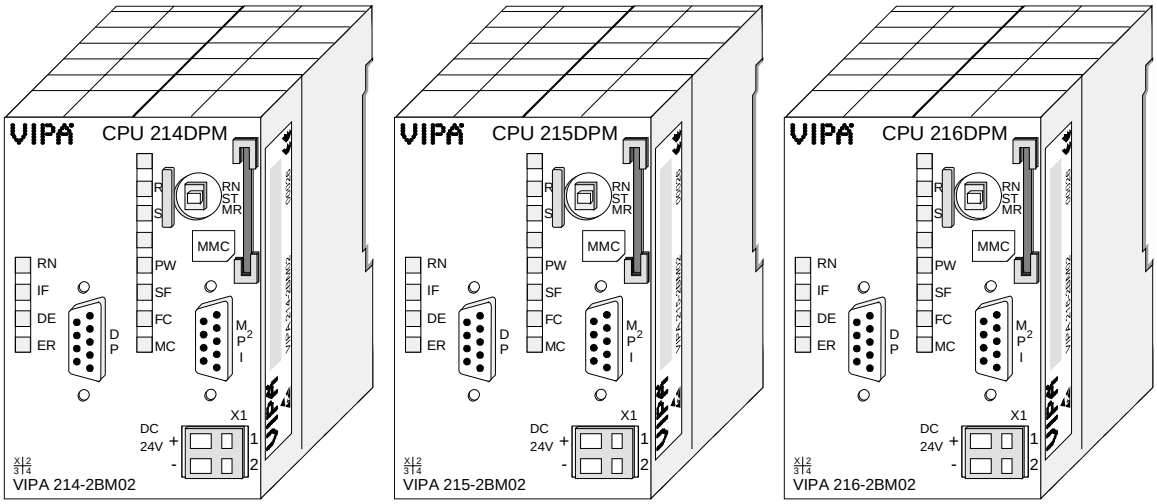


Order data CPU 21xNET

Type	Order number	Description
CPU 214NET	VIPA 214-2BT02	PLC CPU 214 with Ethernet and 48/80KB of work/load memory
CPU 215NET	VIPA 215-2BT02	PLC CPU 215 with Ethernet and 96/144KB of work/load memory
CPU 216NET	VIPA 216-2BT02	PLC CPU 216 with Ethernet and 128/192KB of work/load memory

CPU 21xDPM Identical to CPU 21x, additionally with:

- Integrated Profibus-DP master
- Status-LEDs for Profibus status and diagnostics

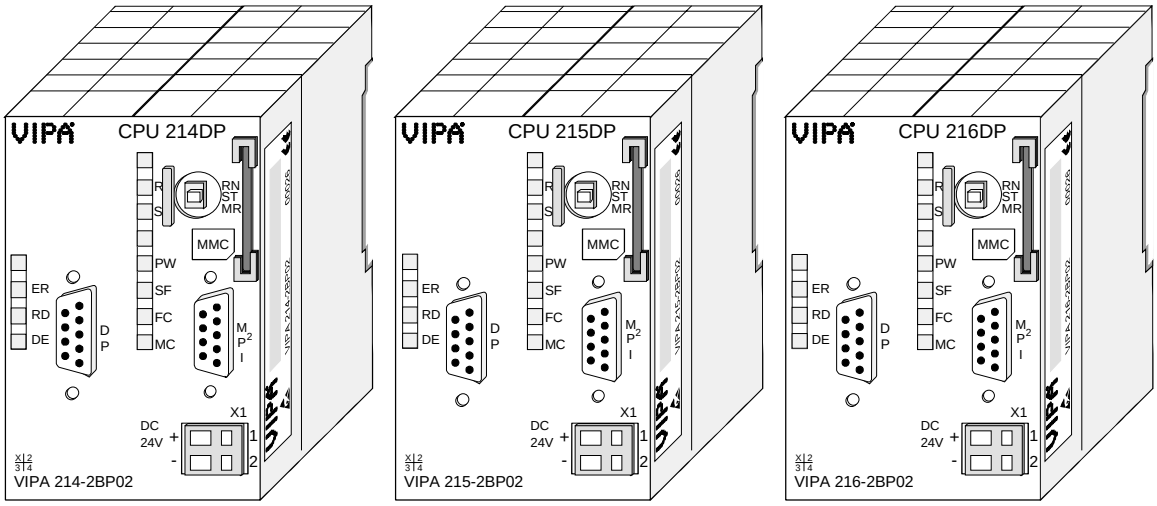


Order data CPU 21xDPM

Type	Order number	Description
CPU 214DPM	VIPA 214-2BM02	PLC CPU 214 with Profibus-DP master and 48/80KB of work/load memory
CPU 215DPM	VIPA 215-2BM02	PLC CPU 215 with Profibus-DP master and 96/144KB of work/load memory
CPU 216DPM	VIPA 216-2BM02	PLC CPU 216 with Profibus-DP master and 128/192KB of work/load memory

CPU 21xDP Identical to CPU 21x, additionally with:

- Integrated Profibus slave
- Status LEDs for Profibus status and diagnostics

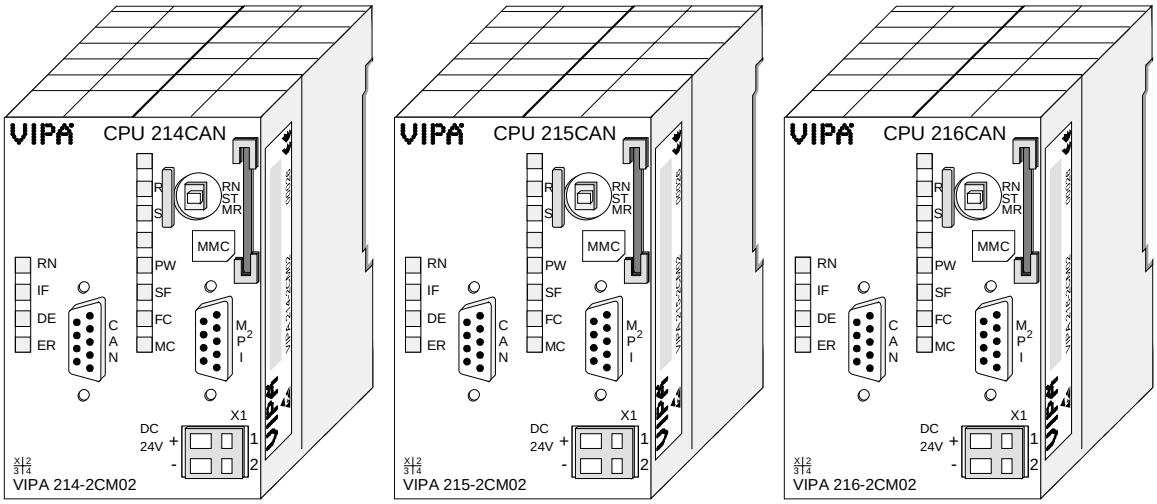


Order data CPU 21xDP

Type	Order number	Description
CPU 214DP	VIPA 214-2BP02	SPS CPU 214 with Profibus slave and 48/80KB of work/load memory
CPU 215DP	VIPA 215-2BP02	SPS CPU 215 with Profibus slave and 96/144KB of work/load memory
CPU 216DP	VIPA 216-2BP02	SPS CPU 216 with Profibus slave and 128/192KB of work/load memory

CPU 21xCAN Identical to CPU 21x, additionally with:

- Integrated CAN master
- Status-LEDs for CAN status and diagnostics

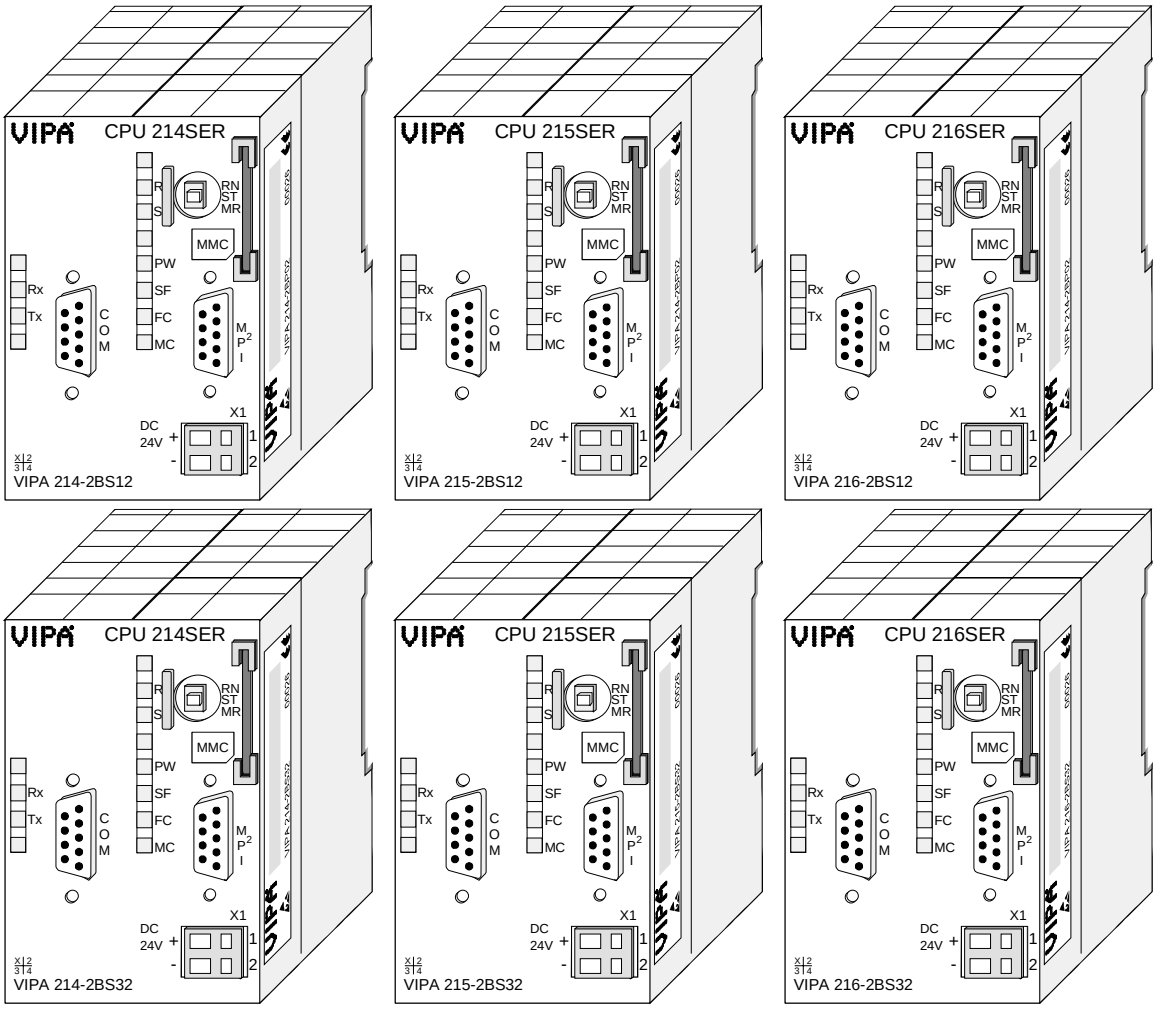


Order data CPU 21xCAN

Type	Order number	Description
CPU 214CAN	VIPA 214-2CM02	PLC CPU 214 with CAN master and 48/80KB of work/load memory
CPU 215CAN	VIPA 215-2CM02	PLC CPU 215 with CAN master and 96/144KB of work/load memory
CPU 216CAN	VIPA 216-2CM02	PLC CPU 216 with CAN master and 128/192KB of work/load memory

CPU 21xSER-1 Identical to CPU 21x, additionally with:

- Serial Communication via two COM interfaces (RS232C or RS485)
- LEDs for communication

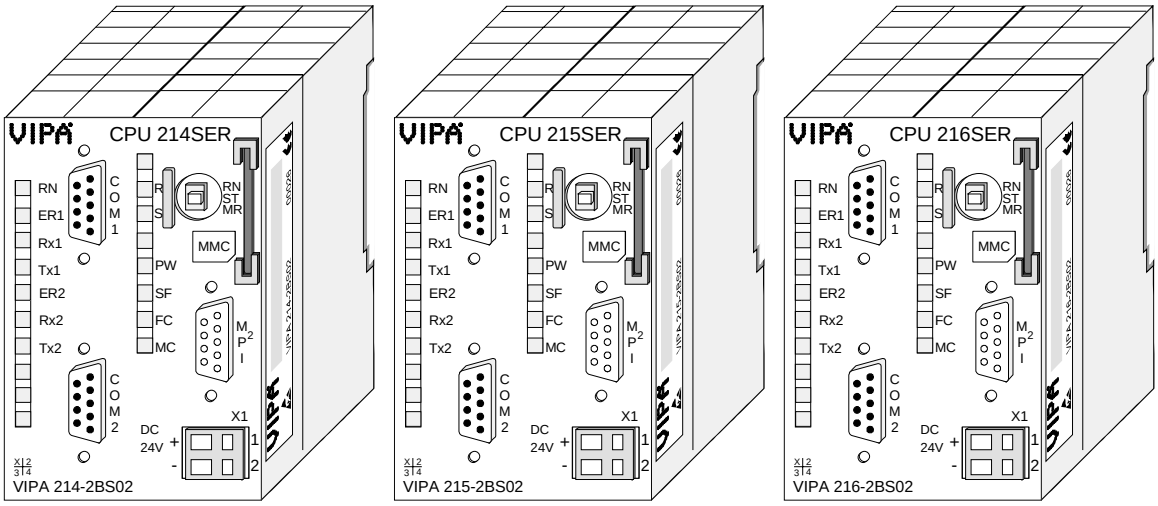


Order data CPU 21xSER-1

Type	Order number	Description
CPU 214SER	VIPA 214-2BS12	SPS CPU 214 with 1xRS232C interface and 48/80kByte of work/load memory
CPU 215SER	VIPA 215-2BS12	SPS CPU 215 with 1xRS232C interface and 96/144kByte of work/load memory
CPU 216SER	VIPA 216-2BS12	SPS CPU 216 with 1xRS232C interface and 128/192kByte of work/load memory
CPU 214SER	VIPA 214-2BS32	SPS CPU 214 with 1xRS485 interface and 48/80kByte of work/load memory
CPU 215SER	VIPA 215-2BS32	SPS CPU 215 with 1xRS485 interface and 96/144kByte of work/load memory
CPU 216SER	VIPA 216-2BS32	SPS CPU 216 with 1xRS485 interface and 128/192kByte of work/load memory

CPU 21xSER-2 Identical to CPU 21x, additionally with:

- Serial Communication via 2 RS232C interfaces
- LEDs for Communication

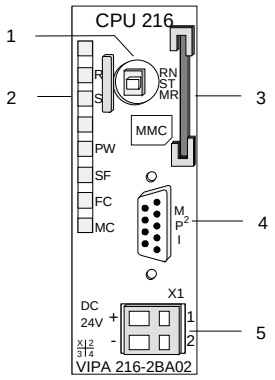


Order data CPU 21xSER-2

Type	Order number	Description
CPU 214SER	VIPA 214-2BS02	SPS CPU 214 with 2xRS232C interface and 48/80kByte of work/load memory
CPU 215SER	VIPA 215-2BS02	SPS CPU 215 with 2xRS232C interface and 96/144kByte of work/load memory
CPU 216SER	VIPA 216-2BS02	SPS CPU 216 with 2xRS232C interface and 128/192kByte of work/load memory

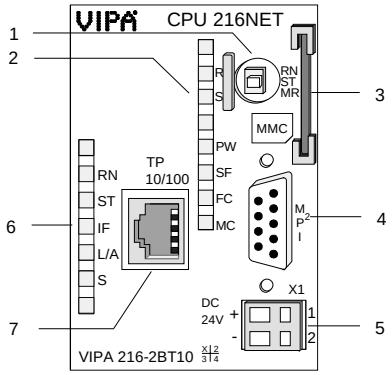
Structure

Front view CPU 21x



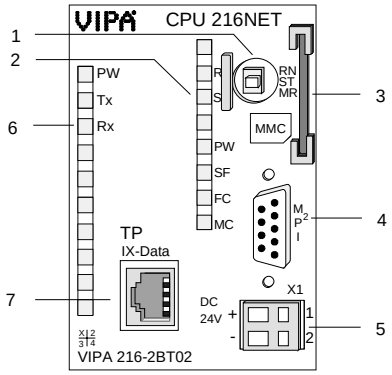
[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP^{2}I interface
[5] Connector for 24V DC power supply

Front view CPU 21x-2BT10



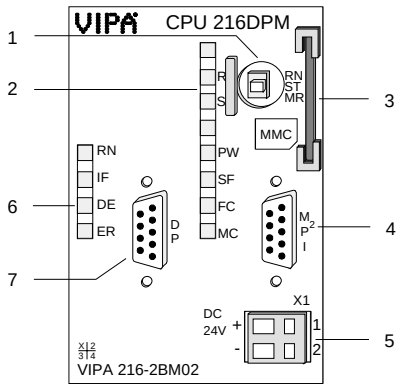
[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP^{2}I interface
[5] Connector for 24V DC power supply
[6] Status indicator LEDs Ethernet
[7] RJ45 jack for twisted pair Ethernet

Front view CPU 21x-2BT02



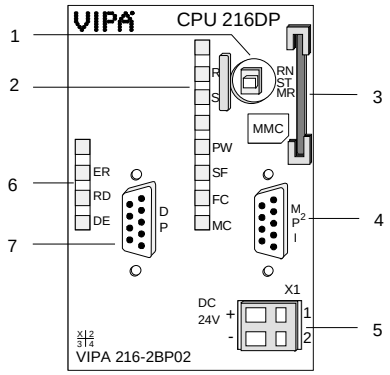
[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP^{2}I interface
[5] Connector for 24V DC power supply
[6] Status indicator LEDs Ethernet
[7] RJ45 jack for twisted pair Ethernet

Front view CPU 21xDPM



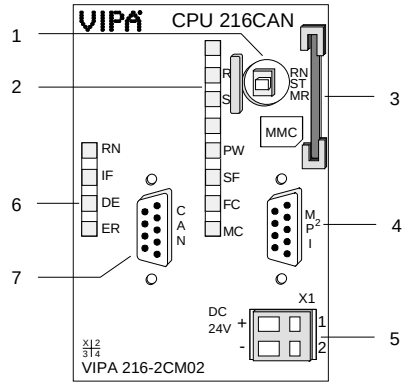
[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP^{2}I interface
[5] Connector for 24V DC power supply
[6] Profibus-DP master status indicator LEDs
[7] Profibus interface

Front view CPU 21xDP



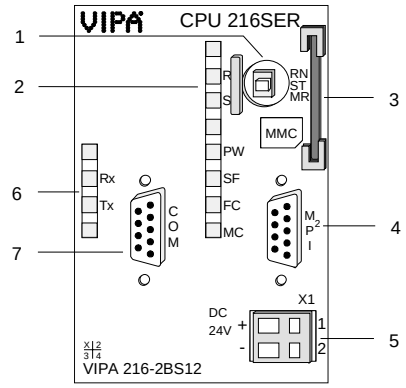
[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP²I interface
[5] Connector for 24V DC power supply
[6] Status indicator LEDs Profibus-DP slave
[7] Profibus DP interface

Front view CPU 21xCAN



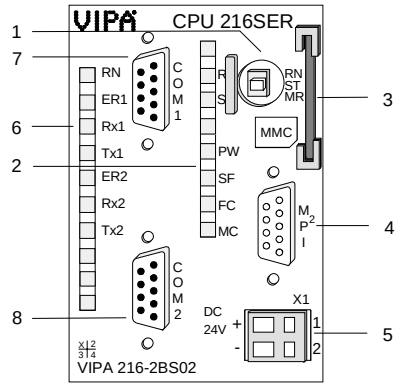
[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP²I interface
[5] Connector for 24V DC power supply
[6] LEDs status indicator CAN master
[7] CAN interface

Front view CPU 21xSER-1



[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP²I interface
[5] Connector for 24V DC power supply
[6] LED status indicator communication
[7] RS232C interface (only 21x-2BS12)
RS485 interface (only 21x-2BS32)

Front view CPU 21xSER-2



[1] RUN/STOP/OVERALL RESET function selector
[2] Status indicator LEDs CPU
[3] Slot for MMC memory card
[4] MP²I interface
[5] Connector for 24V DC power supply
[6] LED status indicator communication
[7] RS232C interface 1
[8] RS232C interface 2

Components

CPU 21x

The components of the CPU 21x that are described here are components of all the CPUs presented in this manual.

LEDs

The CPUs 21x have a number of LEDs that are used to diagnose bus conditions and to display the status of a program. The table below describes the diagnostic LEDs and the according colors.

These LEDs are part of every CPU in this manual.

Name	Color	Description
PW	green	Indicates CPU power on.
R	green	CPU status is RUN.
S	yellow	CPU status is STOP.
SF	red	Is turned on if a system error is detected (hardware defect)
FC	yellow	Is turned on when variables are forced (fixed).
MC	yellow	This LED blinks when the MMC is accessed.

Function selector RN/ST/MR

You can select the operating mode STOP (ST) and RUN (RN) by means of the function selector. The CPU automatically executes the operating mode START-UP when the mode changes from STOP to RUN.

You may issue an overall reset by placing the switch in the Memory Reset (MR) position.

MMC slot memory card

You may install a VIPA MMC memory card in this slot as external storage device (Order No.: VIPA 953-0KX00).

The access to the MMC takes always place after an overall reset.

Power supply

The CPU has an internal power supply. This is connected to an external supply voltage via two terminals located on the front of the unit.

The power supply requires DC 24V (20,4 ... 28,8V). In addition to the electronic circuitry of the CPU this supply voltage is used for the modules connected to the backplane bus.

The electronic circuitry of the CPU is not dc-insulated from the supply voltage. The power supply is protected against reverse polarity and short circuits.

Note!

Please ensure that the polarity of the supply voltage is correct.

Battery backup for clock and RAM

A rechargeable battery is installed on every CPU 21x to safeguard the contents of the RAM when power is removed. This battery is also used to buffer the internal clock.

The rechargeable battery is maintained by a charging circuit that receives its power from the internal power supply and that maintain the clock and RAM for a max. period of 30 days.

Attention!

The CPU will operate only if the battery is healthy.

The CPU will STOP when the battery is faulty. In this case the CPU should be checked. Please contact VIPA!

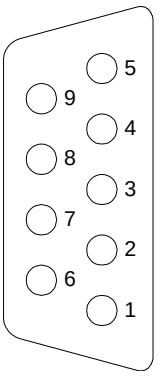
MP²I interface

The MPI unit provides the link for the data transfer between the CPU and the PC. Via bus communication you are able to exchange programs and data between different CPUs that are linked over MPI.

For a serial exchange between the partners you normally need a special MPI-converter. But now you are also able to use the VIPA "Green Cable" (Order-No. VIPA 950-0KB00), which allows you to establish a serial peer-to-peer connection over the MPI interface.

Please regard the notes about the "Green Cable" in chapter "Principles"!

The pin assignment of the MP²I jack is as follows:



9pin jack

Pin	Assignment
1	reserved (must not be connected) See Hints for the deployment of the MPI interface in chapter "Priciples".
2	M24V
3	RxD/TxD-P (Line B)
4	RTS
5	M5V
6	P5V
7	P24V
8	RxD/TxD-N (Line A)
9	n.c.

CPU 21x-2BT10

In addition to the components described in the section on the CPU 21x the CPU 21x-2BT10 module is provided with further LEDs and an Ethernet interface located at the left-hand side of the module.

LEDs

The LEDs are located on the left-hand side of the front panel and indicate communication activities.

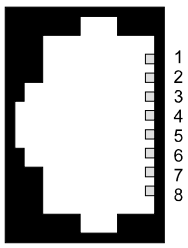
The table below shows the color and the meaning of these LEDs.

Name	Color	Description
RN	green	CP RUN On: CP project is loaded Off: CP is reset (no project)
ST	yellow	CP STOP On: CP status is reset Off: CP Project is transferred
IF	red	On: Internal CP error
L/A	green	Link/Activity: On: physically connected to Ethernet Off: no physical connection to Ethernet Blinks: Ethernet activity
S	green	Transfer speed: On: 100MBit Off: 10MBit

Ethernet interface

An RJ45 jack provides the interface to the twisted pair cable, required for Ethernet. The pin assignment of this jack is as follows:

8pin RJ45 jack:



Pin	Assignment
1	Transmit +
2	Transmit -
3	Receive +
4	-
5	-
6	Receive -
7	-
8	-

Note!

For more detailed information on twisted pair networks refer to chapter "Deployment of the CPU 21x-2BT10".

CPU 21x-2BT02

In addition to the components described in the section on the CPU 21x the CPU 21x-2BT02 module is provided with further LEDs and an Ethernet interface located at the left-hand side of the module.

LEDs

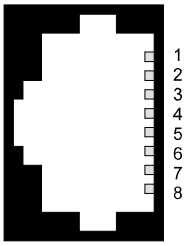
The LEDs are located on the left-hand side of the front panel and indicate communication activities.

The table below shows the color and the meaning of these LEDs.

Name	Color	Description
PW	green	Indicates CPU power on
TxD	green	Transmit data
RxD	green	Receive data

Ethernet interface

An RJ45 jack provides the interface to the twisted pair cable, required for Ethernet. The pin assignment of this jack is as follows:

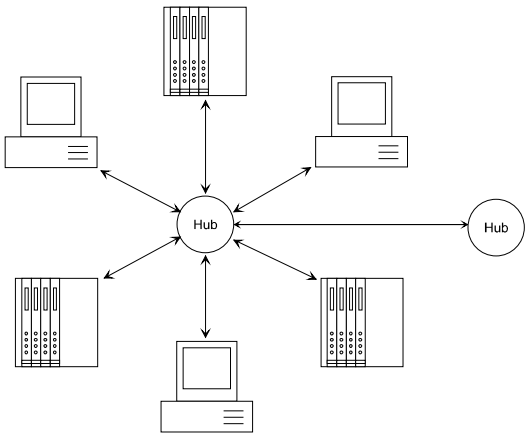


8pin RJ45 jack:

Pin	Assignment
1	Transmit +
2	Transmit -
3	Receive +
4	-
5	-
6	Receive -
7	-
8	-

Star topology

A twisted pair network can only have a star topology. For this purpose a hub is required as the central node:



Note!

For more detailed information on twisted pair networks refer to chapter "Deployment of the CPU 21x-2BT02".

CPU 21xDPM

In addition to the components described in the section on the CPU 21x the CPU 21xDPM module is provided with 4 more LEDs and a Profibus interface.

LEDs

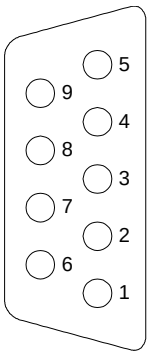
The LEDs are located in the left half of the front panel and they are used for diagnostic purposes. The following table shows the color and the significance of these LEDs.

Name	Color	Description
RN	green	DP-Master-RUN On: Master status is RUN. The slaves are being accessed and the outputs are 0 ("clear" state). On with DE: Master status is "operate". and is communicating with the slaves.
IF	red	Initialization error On: Error in Profibus configuration
DE	yellow	DE (Data exchange) On: Indicates Profibus communication activity.
ER	red	Error On: Slave has failed

Profibus interface

The CPU 21xDPM is connected to the Profibus system by means of a 9pin jack. The pin assignment of this interface is as shown:

9pin Profibus D-type jack:



Pin	Assignment
1	Shield
2	n.c.
3	RxD/TxD-P (Line B)
4	RTS
5	M5V
6	P5V
7	n.c.
8	RxD/TxD-N (Line A)
9	n.c.

Note!

Refer to the chapter "Deployment of the CPU 21xDPM" for details on the Profibus master.

CPU 21xDP

In addition to the components described in the section on the CPU 21x the CPU 21xDP module is provided with 3 more LEDs and a Profibus interface.

LEDs

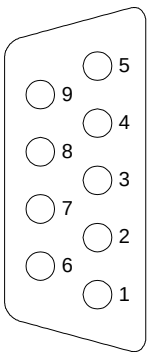
The LEDs are located in the left half of the front panel and they are used for diagnostic purposes. The following table shows the color and the significance of these LEDs.

Name	Color	Description
ER	red	Error On: Error in Profibus part detected respectively CPU has been stopped. Flashing (2Hz): Initialization error Flashing (10Hz): Supply voltage < DC18V Flashing *alternately* with RD: Configuration error (error at Master configuration) Flashing *simultaneously* with RD: Error in parameterization
RD	green	Ready On: Data transfer via back plane bus Flashing: Self-test result is positive (READY) and successful initialization
DE	green	DE (Data exchange) On: Indicates an active Profibus communication.

Profibus interface

The CPU 21xDP is connected to the Profibus system by means of a 9pin jack. The pin assignment of this interface is as shown:

9pin Profibus D-type jack:



Pin	Assignment
1	Shield
2	n.c.
3	RxD/TxD-P (Line B)
4	RTS
5	M5V
6	P5V
7	n.c.
8	RxD/TxD-N (Line A)
9	n.c.

Note!

Refer to the chapter "Deployment of the CPU 21xDP" for details on the Profibus.

CPU 21xCAN

In addition to the components described in the section on the CPU 21x the CPU 21xCAN module is provided with 4 more LEDs and a CAN interface.

LEDs

The LEDs are located in the left half of the front panel and they are used for diagnostic purposes. The following table shows the color and the significance of these LEDs.

Name	Color	Description
RN	green	CAN master RUN ON: CAN master state is RUN OFF: CAN master state is STOP
ER	red	Error ON: During initialization and at slave failure OFF: All slaves are in the state "operational"
BA	yellow	BA (Bus active) On: CAN bus communication respectively state "operational" Blinking (1Hz): State "pre-operational".
IF	red	Initialization ON: Initialization error at wrong parameterization. Off: Initialization is OK.

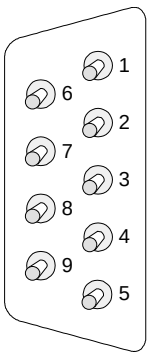
Note!

If all LEDs are blinking with 1Hz, the CAN master awaits valid parameters from the CPU. If the CAN master is not supplied with parameters by the CPU his LEDs get off after 5s.

CAN interface

The CPU 21xCAN is connected to the CAN system by means of a 9pin plug. The pin assignment of this interface is as shown:

9pin CAN D-type plug:



Pin	Assignment
1	reserved
2	CAN low
3	CAN Ground
4	reserved
5	Screen
6	Ground 24V
7	CAN high
8	reserved
9	+DC 24V

Note!

More details on the CAN master see chapter "Deployment CPU 21xCAN"!

CPU 11xSER-1

Additional to the components described before, the CPU 21x with order no. 21x-2BS12 has an RS232C interface and the CPU 21x with order no. 21x-2BS32 an RS485 interface.

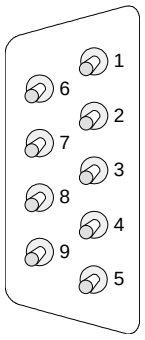
LEDs

The LEDs are located in the left half of the front panel and they are used for diagnostic purposes. The following table shows the color and the significance of these LEDs.

Name	Color	Description
Rx	green	Interface receive data
Tx	green	Interface transmit data

RS232C interface

Via 9pin jack, you may establish a serial RS232C point-to-point connection. The pin assignment of this interface is as shown:

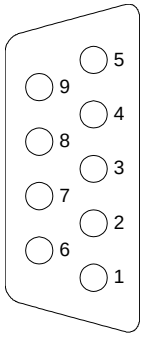


9pin plug (*CPU 21x-2BS12)*

Pin	Assignment
1	CD-
2	RxD
3	TxD
4	DTR-
5	GND
6	DSR-
7	RTS-
8	CTS-
9	RI-

RS485 interface

Via 9pin slot, you may establish a serial RS485 bus connection. The pin assignment of this interface is as shown:



9pin jack (*CPU 21x-2BS32)*

Pin	Assignment
1	n.c.
2	n.c.
3	RxD/TxD-P (Line B)
4	RTS
5	M5V
6	P5V
7	n.c.
8	RxD/TxD-N (Line A)
9	n.c.

CPU 21xSER-2

In addition to the components described in the section on the CPU 21x the CPU 21xSER-2 module is provided with more LEDs and two serial RS232C interfaces.

LEDs

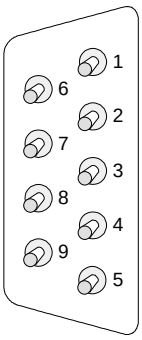
The LEDs are located in the left half of the front panel and they are used for diagnostic purposes. The following table shows the color and the significance of these LEDs.

Name	Color	Description
RN	green	Communication processor runs
ER1	red	Error Interface 1
Rx1	green	Interface 1 receive data
Tx1	green	Interface 1 transmit data
ER2	red	Error Interface 2
Rx2	green	Interface 2 receive data
Tx2	green	Interface 2 transmit data

RS232C interface COM1, COM2

The CPU 21xSER-2 has got a communication processor with 2 RS232C interfaces. The pin assignment of the interfaces is as follows:

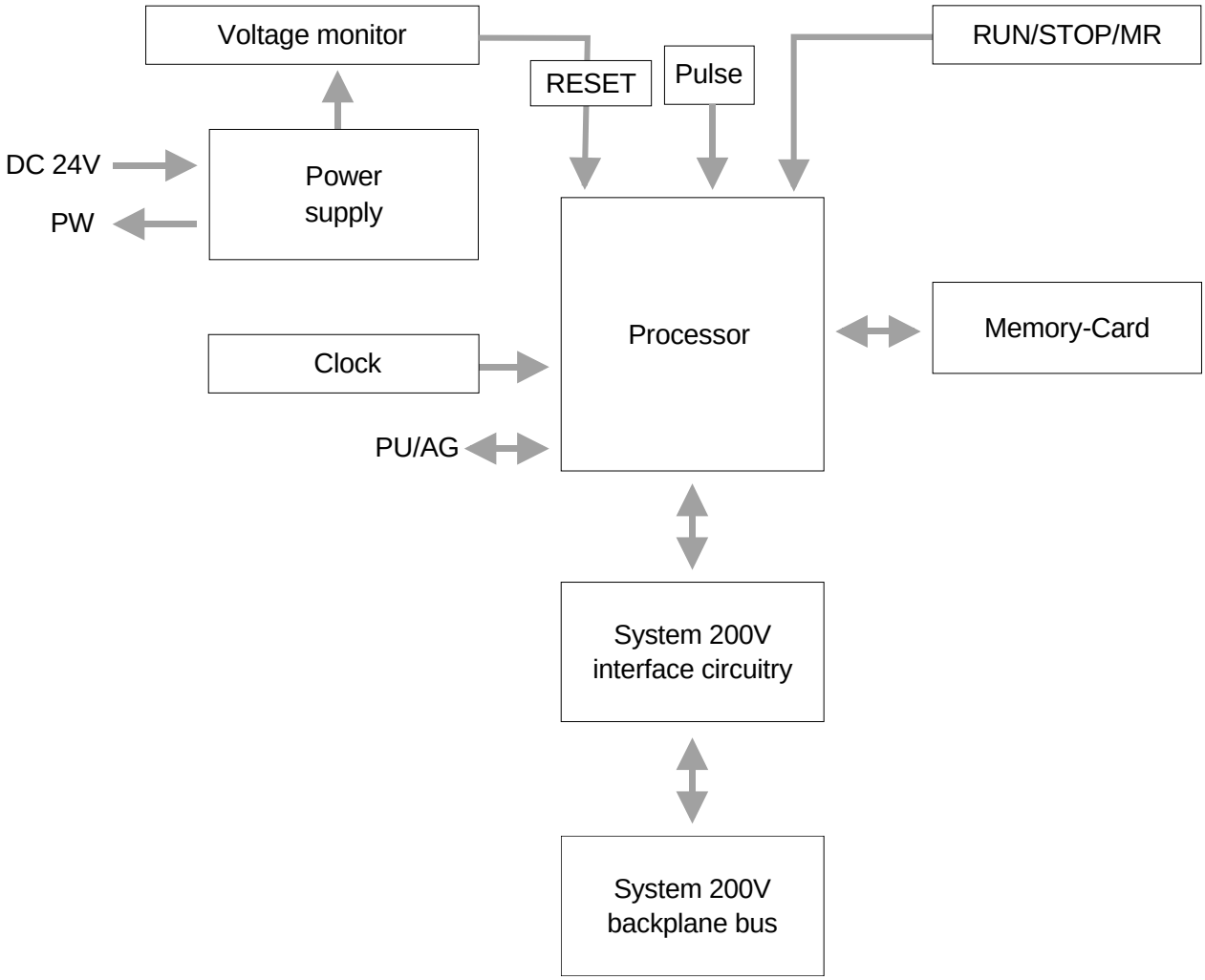
9pin jack



Pin	Assignment
1	CD-
2	RxD
3	TxD
4	DTR-
5	GND
6	DSR-
7	RTS-
8	CTS-
9	RI-

Block diagram

The following block diagram shows the basic hardware construction of the CPU 21x modules:



Technical data

CPU 21x

General

Electrical data		VIPA 214-1BC02, VIPA 214-1BA02 ... VIPA 216-1BA02
Power supply		DC 24V (20.4 ... 28.8V)
Current consumption		max. 1.5A
Dissipation power		max. 3.5W
Status indicators (LEDs)		by means of LEDs located on the front
Connections / interfaces		MP^2I interface for data communication
Clock, memory/clock backup		yes / Lithium accumulator, 30 days backup
Output current to backplane bus		max. 3A
Bit memory		8192 Bit (M0.0…M255.7)
Timer		256 (T0...T255)
Counters		256 (Z0 ... Z255)
Number of Blocks	FB	1024 (FB0...FB1023)
	FC	1024 (FC0...FC1023)
	DB	2047 (DB1...DB2047)
Total addressing space input / output		1024/1024Byte, each 128 Byte for process image (PA)
Process image Inputs		1024Bit (I0.0...I127.7)
Process image Outputs		1024Bit (O0.0...O127.7)
Combination with peripheral modules		
max. no. of modules		32
max. digital I/O		32
max. analog I/O		16
Addressable inputs, outputs		1024 (digital), 128/128 (analog)
Dimensions and weight		
Dimensions (WxHxD) in mm		25.4x76x80
Weight		80g

Module specific

	CPU 214C	CPU 214	CPU 215	CPU 216
Work memory	32kByte	48kByte	96kByte	128kByte
Load memory	40kByte	80kByte	144kByte	192kByte
Cycle time bit operations	0.18µs	0.18µs	0.18µs	0.18µs
Cycle time word operations	0.78µs	0.78µs	0.78µs	0.78µs
Order-No.:	214-1BC02	214-1BA02	215-1BA02	216-1BA02

CPU 21x-2BT10

Electrical data	VIPA 214-2BT10 ... VIPA 216-2BT10
Power supply	DC 24V (20.4 ... 28.8V)
Current consumption	max. 1.5A
Dissipation power	max. 6W
Potential separation	≥ 500V AC to Ethernet
Status indicator (LEDs)	like CPU 21x additionally with LEDs for the Ethernet section
Connections/interfaces	like CPU 21x additionally with RJ45 jack (Ethernet)
Ethernet interface	CP 243
Connector	RJ45
Network topology	Star topology
Medium	Twisted pair
Transfer rate	10/100Mbit
Overall length	max. 100m per segment
PG/OP channel	8 (32 with CP firmware version 1.7.4 and up)
Configurable connections:	16
Dimensions and weight	
Dimensions (WxHxD) in mm	50.8x76x80
Weight	150g

CPU 21x-2BT02

Electrical data	VIPA 214-2BT02 ... VIPA 216-2BT02
Power supply	DC 24V (20.4 ... 28.8V)
Current consumption	max. 1.5A
Dissipation power	max. 6W
Potential separation	≥ 500V AC to Ethernet
Status indicator (LEDs)	like CPU 21x additionally with LEDs for the Ethernet section
Connections/interfaces	like CPU 21x additionally with RJ45 jack (Ethernet)
Ethernet interface	CP 243
Connector	RJ45
Network topology	Star topology
Medium	Twisted pair
Transfer rate	10Mbit
Overall length	max. 100m per segment
PG/OP channel	-
Configurable connections:	32
Dimensions and weight	
Dimensions (WxHxD) in mm	50.8x76x80
Weight	150g

CPU 21xDPM

Electrical Data	VIPA 214-2BM02 ... VIPA 216-2BM02
Power supply	DC 24V (20.4 ... 28.8V)
Current consumption	max. 1.5A
Dissipation power	max. 5W
Potential separation	≥ 500V AC to fieldbus
Status monitoring (LEDs)	like CPU 21x additionally with LEDs for the Profibus section
Adapters/interfaces	like CPU 21x additionally with 9pin D-type jack (Profibus)
Profibus interface	
Connector	9pin D-type jack
Network topology	Linear bus, active bus termination at both ends
Medium	Screened and drilled twisted pair cable. Depending on environment screening may be omitted.
Transfer rate	9.6kBaud up to 12MBaud
Overall length	100m at 12 MBaud without repeater, up to 1000m with repeater
max. no. of stations	32 stations on every segment without repeater. Expandable to 126 stations with repeater.
Combination with peripheral modules	
max. number of slaves	125
max. number of input bytes	1024
max. number of output bytes	1024
Dimensions and Weight	
Dimensions (WxHxD) in mm	50.8x76x80
Weight	150g

CPU 21xDP

Electrical data	VIPA 214-2BP02 ... VIPA 216-2BP02
Power supply	DC 24V (20.4 ... 28.8V)
Current consumption	max. 1.5A
Dissipation power	max. 5W
Potential separation	≥ 500V AC to fieldbus
Status indicator (LEDs)	like CPU 21x additionally with LEDs for the Profibus section
Adapters/interfaces	like CPU 21x additionally with 9pin D-type jack (Profibus)
Profibus interface	
Connector	9pin D-type jack
Network topology	Linear bus, active bus termination at both ends
Medium	Screened and drilled twisted pair cable. Depending on environment screening may be omitted.
Transfer rate	9.6kBaud up to 12MBaud
Overall length	100m at 12 MBaud without repeater, up to 1000m with repeater
max. no. of stations	32 stations on every segment without repeater. Expandable to 126 stations with repeater.
Dimensions and weight	
Dimensions (WxHxD) in mm	50.8x76x80
Weight	150g

CPU 21xCAN

Electrical Data	VIPA 214-2CM02 ... VIPA 216-2CM02
Power supply	DC 24V (20.4 ... 28.8V)
Current consumption	max. 1.5A
Dissipation power	max. 5W
Potential separation	≥ 500V AC to fieldbus
Status monitoring (LEDs)	like CPU 21x additionally with LEDs for the CAN section
Adapters/interfaces	like CPU 21x additionally with 9pin D-type jack (CAN)
CAN interface	
Connector	9pin D-type jack
Network topology	Linear bus, active bus termination at both ends
Medium	Screened and drilled twisted pair cable. Depending on environment screening may be omitted.
Transfer rate	10kBaud up to 1MBaud
Overall length	without repeater 1000m at 50kBaud
max. no. of stations	126 stations
Combination with peripheral modules	
max. number of slaves	125
max. number of TxPDOs	40
max. number of RxPDOs	40
max. number of input bytes	384
max. number of output bytes	384
Dimensions and Weight	
Dimensions (WxHxD) in mm	50.8x76x80
Weight	150g

CPU 21xSER-1

Module specific	VIPA 214-2BS12...216-2BS12	VIPA 214-2BS32...216-2BS32
Power supply	DC 24V (20.4 ... 28.8)	
Current consumption	max. 1.5A	
Dissipation power	max. 5W	
Potential separation	≥ 500V AC to serial interface	
Status indicator (LEDs)	like CPU 21x	
RS232C interface integrated	yes	-
RS485 interface integrated	-	yes
Dimensions and weight		
Dimensions (WxHxD) in mm	50.8x76x80	
Weight	150g	

CPU 21xSER-2

Electrical data	VIPA 214-2BS02 ... VIPA 216-2BS02
Power supply	DC 24V (20.4 ... 28.8V)
Current consumption	max. 1.5A
Dissipation power	max. 5W
Potential separation	-
Status indicator (LEDs)	like CPU 21x additionally with LEDs for serial communication
Adapters/interfaces	like CPU 21x additionally with 2x 9pin D-type jack (serial communication)
Interfaces	
COM 1, COM 2	2 integrated RS232C interfaces
Dimensions and weight	
Dimensions (WxHxD) in mm	50.8x76x80
Weight	150g

Chapter 3 Deployment CPU 21x

Outline

This chapter describes the deployment of the CPU 21x together with the peripheral modules of the System 200V.

Besides commissioning and start-up behavior you will find here also a description of the project engineering, parameterization, operating modes and test functions.

This information also basically applies to the deployment of the CPU 21x with integrated communication part.

The following section contains a description of:

- Assembly and commissioning
- Principle of address allocation
- Project engineering and parameterization
- Deployment of MPI and MMC
- Operating modes and overall reset and firmware update
- Usage of test functions

Contents

Note!

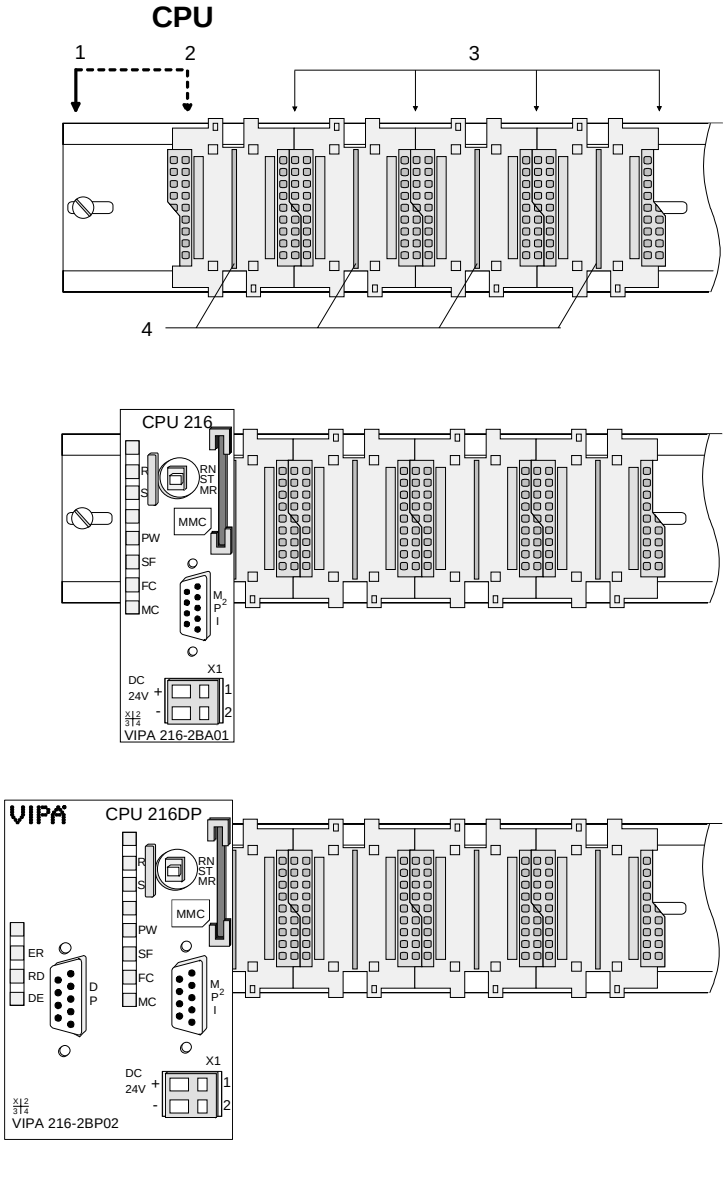
This information is valid for all the CPUs described in this manual since the backplane bus communication between the CPU and the peripheral modules is the same for all models of this CPU!

Assembly

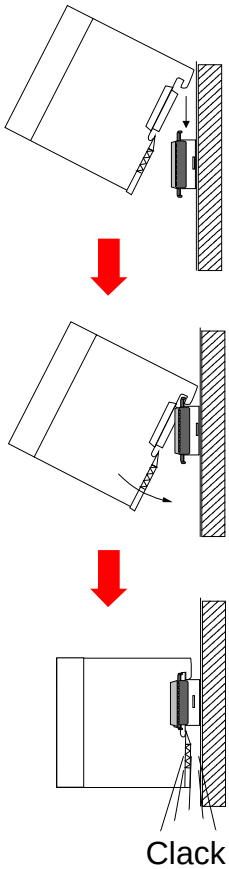
Attention!

It is mandatory to turn off the power supply before you insert or remove any modules!

Please note that the CPU may only be installed into plug-in location 1 or 2 (see figure below).



[1] CPU, double width
[2] CPU, single width
[3] Peripheral modules
[4] Guiding bars



For details on the assembly of System 200V modules please refer to the System 200V manual (Order-No.: VIPA HB97).

Start-up behavior

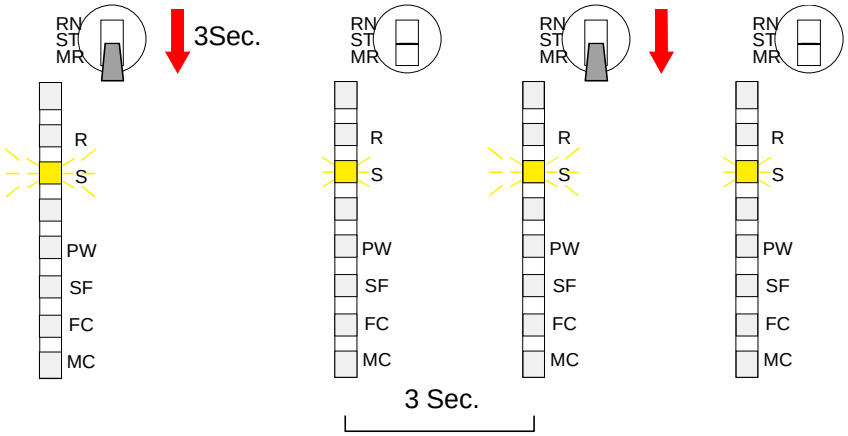
Turn on power supply

After turning on the power supply, the CPU switches to the operating mode that is fixed by the operating mode lever at the CPU.

Now you may transfer your project from your projecting tool into the CPU via MPI resp. plug-in a MMC containing your project and request an OVERALL_RESET.

OVERALL_RESET

The following picture shows the approach:



Note!

The transfer of the user application from the MMC into the CPU takes always place after an OVERALL RESET!

Start-up after delivery

After delivery the CPU is totally clear.

After a STOP→RUN transition, the CPU switches to RUN without application.

Start-up with valid data on the CPU

The CPU switches to RUN with the application located in the battery buffered RAM.

Start-up with empty battery

The battery is loaded directly via the integrated power supply and provides a buffer for up to 30 days. If this time is exceeded, the battery may be totally discharged and the battery buffered RAM is erased.

In this state the CPU starts an overall_reset. If a MMC is plugged in, the program of the MMC is transferred into the RAM.

Depending on the selected operating mode the CPU switches to RUN resp. stays in STOP.

This procedure is fixed in the diagnostic buffer with this entry: "Automatic start OVERALL_RESET (unbuffered POWER-ON)".

Address allocation

Automatic addressing

To provide specific addressing of the installed peripheral modules, certain addresses must be allocated in the CPU.

The CPU contains a peripheral area (addresses 0 ... 1023) and a process image of the inputs and the outputs (for both each address 0 ... 127).

When the CPU is initialized it automatically assigns peripheral addresses to the digital input/output modules starting from 0.

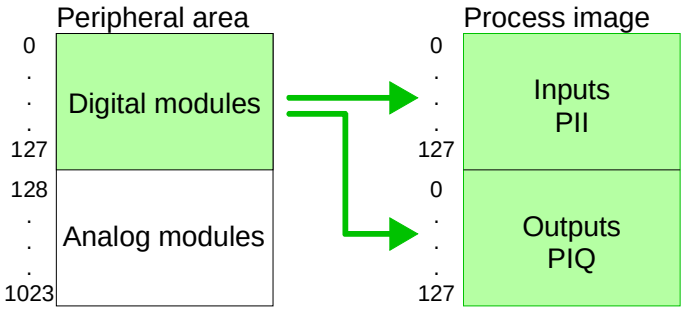
If there is no hardware projecting, analog modules are allocated to even addresses starting from address 128.

Signaling states in the process image

The signaling states of the lower addresses (0 ... 127) are additionally saved in a special memory area called the *process image*.

The process image is divided into two parts:

- process image of the inputs (PII)
- process image of the outputs (PIQ)



The process image is updated automatically when a cycle has been completed.

Read/write access

You may access the modules by means of read or write operations on the peripheral bytes or on the process image.

Note!

Please remember that you may access different modules by means of read and write operations on the same address.

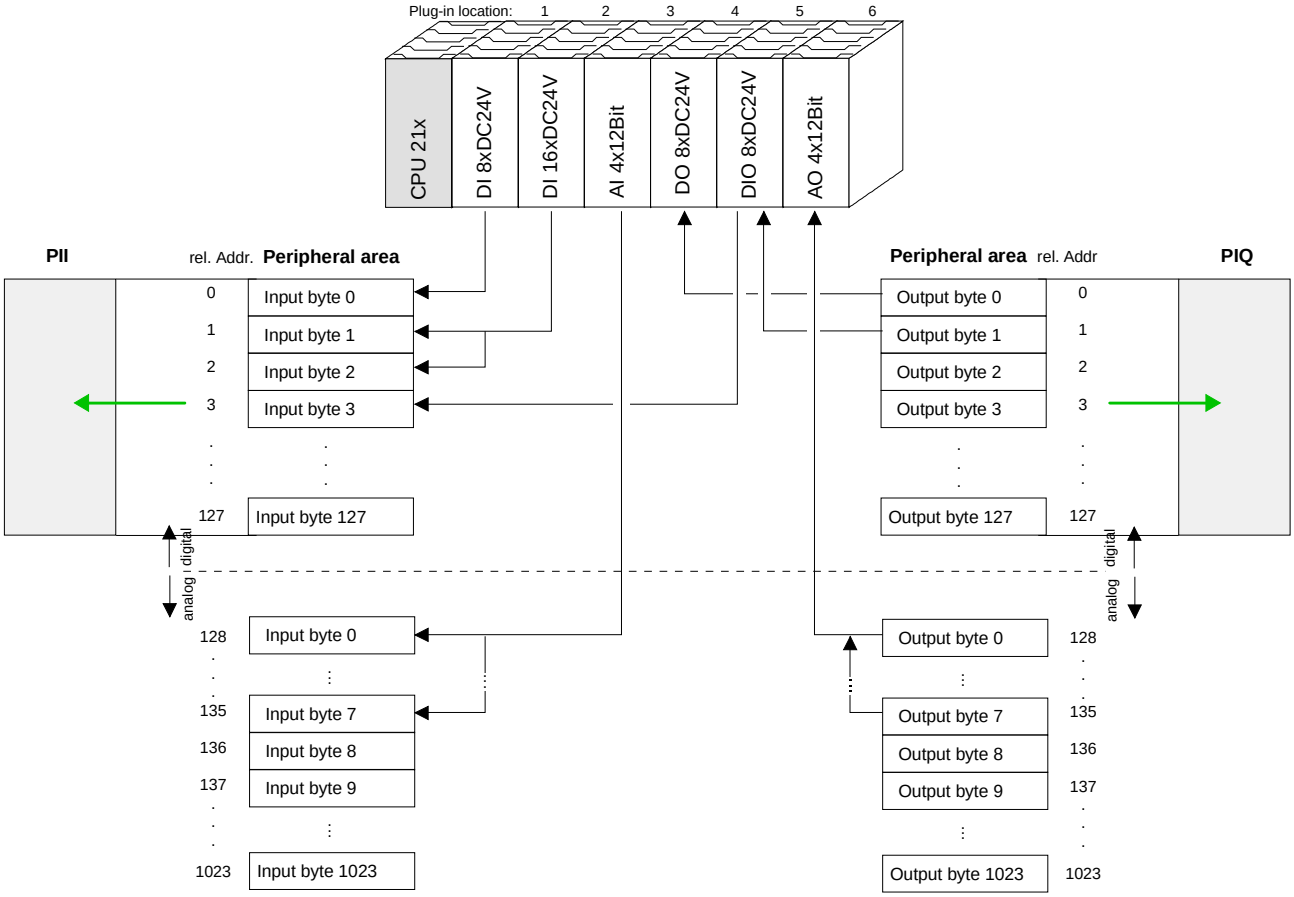
The addressing ranges of digital and analog modules are different when they are addressed automatically.

Digital modules: 0 ... 127

Analog modules: 128 ... 1023

Example for automatic address allocation

The following figure illustrates the automatic allocation of addresses:



Modifying allocated addresses by configuration

You may change the allocated addresses at any time by means of the Siemens SIMATIC Manager. In this way you may also change the addresses of analog modules to the range covered by the process image (0...127) and address digital modules above 127.

The following pages describe the required preparations and the procedure for this type of configuration.

Project engineering

General

The following information always refers to modules that have been installed on the same bus adjacent to the CPU.

In order to address the installed peripheral modules individually, specific addresses in the CPU have to be assigned to them.

The allocation of addresses and the configuration of the installed modules is a function of the Siemens SIMATIC Manager where the modules appear as a virtual Profibus system. For the Profibus interface is software standardized, we are able to guarantee the complete functionality of the System 200V together with the Siemens SIMATIC Manager by including a GSD-file (german: **G**eräte**S**tamm**D**atei = GSD-file).

The project is transferred serial to the CPU via the MPI interface or by use of a MMC.

Fast introduction

For the employment of the CPU 21x from VIPA using the Siemens SIMATIC Manager the inclusion of the System 200V via the vipa GSD file in the hardware catalog is required.

To be compatible with the Siemens SIMATIC Manager you have to execute the following steps:

- Start the hardware configurator from Siemens and include vipa_21x.gsd from VIPA.
- Configure CPU 315-2DP (315-2AF03 0AB00 V1.2) from Siemens and create a new Profibus subnet.
- Attach the System "VIPA_CPU21x" to the subnet with **Profibus-Address 1**, to be found at the hardware catalog at *PROFIBUS DP > Additional field devices > IO > VIPA_System_200V*.
- Place **always at the 1st slot** the corresponding CPU 21x, by taking it from the hardware catalog.
- Include afterwards your System 200V modules in the location sequence.
- Save and transfer you project.

Requirements

The following conditions must be fulfilled for project engineering:

- The Siemens SIMATIC Manager is installed at PC respectively PU
- The GSD files have been included in Siemens hardware configurator
- Serial connection to the CPU (e.g. "Green Cable" from VIPA)

Note!

The configuration of the CPU requires a thorough knowledge of the Siemens SIMATIC Manager and the hardware configurator!

Including the GSD- file

- Go to www.vipa.de > Service > Download > GSD- und EDS- Files > Profibus and download the file Cx000023_Vxxx.
- Extract the file to your work directory. The vipa_21x.gsd (german) respectively vipa_21x.gse (english) can be found at the directory VIPA_System_200V.
- Start the Siemens hardware configurator and close every project.
- Go to **Options** > *Install new GSD file*
- Navigate to the directory *System_200V* and choose the corresponding file **vipa_cpu21x.gsd** (german) or **vipa_cpu21x.gse** (english)

Now the modules of the VIPA System 200V are integrated in the hardware catalog at *Profibus-DP \ Additional field devices \ I/O \ VIPA_System_200V*.

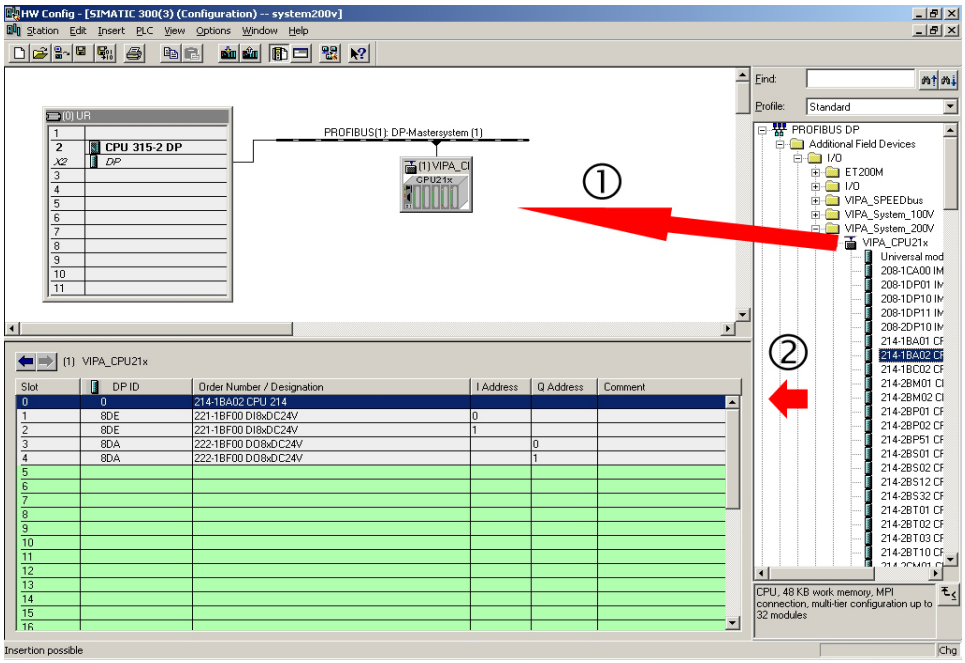
Project engineering CPU 21x

- Start the Siemens hardware configurator with a new project and insert a profil rail from the hardware catalog.
- Place at the 1st possible slot the CPU 315-2DP (6ES7 315-2AF03 V1.2) from Siemens.
- If your CPU 21x provides a DP master, you may now connect it to Profibus and include your DP slaves.

Project engineering System 200V modules

The System 200V modules have to be configured after project engineering of the CPU 21x as described below:

- Create a Profibus subnet (if not yet available).
- Attach the System "VIPA_CPU21x" to the subnet. The respective entries are located in the hardware catalog under *PROFIBUS DP > Additional Field Devices > IO > VIPA_System_200V.* Assign **Profibus address 1** to this slave.
- Place the VIPA CPU 21x that you want to deploy at the 1st plug-in location of the configurator by taking if from the hardware catalog.
- Include your System 200V modules in the location sequence behind it.
- Save your project.

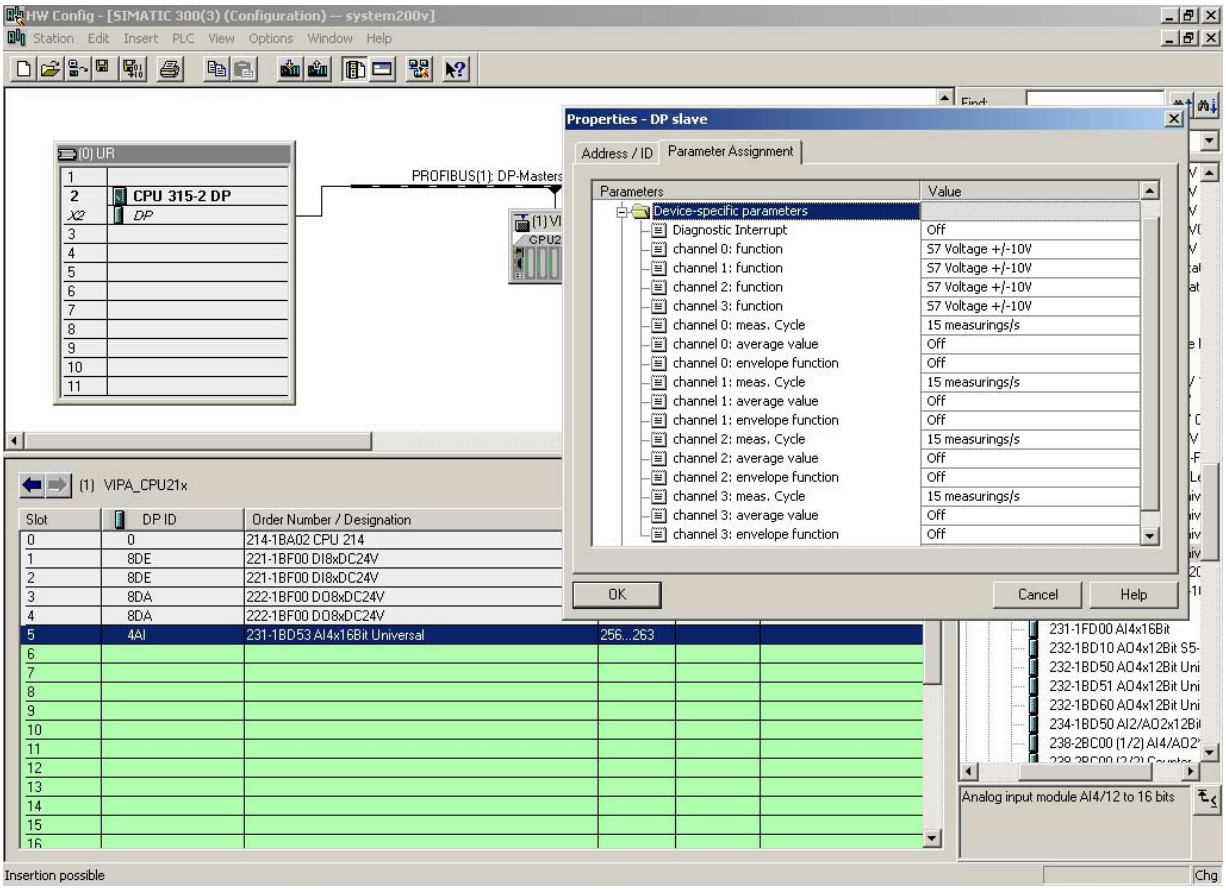


Parameterization System 200V modules

System 200V modules may be supplied with up to 16Byte of configuration data from the CPU. The use of the Siemens SIMATIC Manager provide the option to supply parameters to the configurable System 200V modules at any time.

For this purpose you have to double-click the respective module in the plug-in diagram.

The following figure shows the configuration of a analog input module:



Transferring the project

Data is transferred between the CPU and the PC via MPI. If your PG has no MPI functionality you may use the VIPA "Green Cable" to send the data serial by a peer-to-peer connection. The VIPA "Green Cable" has the OrderNo. VIPA 950-0KB00 and may only be used at MP^{2}I interfaces of VIPA CPUs!

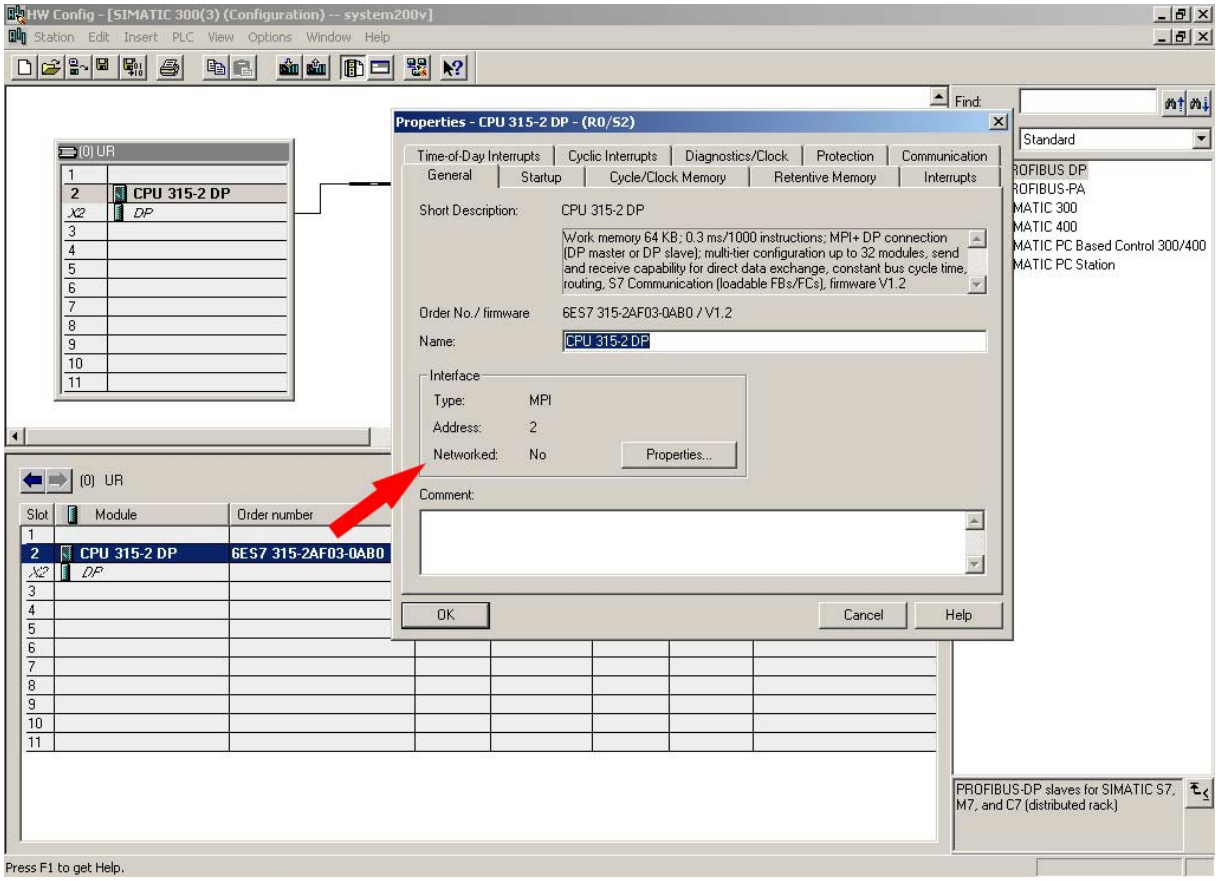
Please regard the notes about the "Green Cable" in chapter 1!

- Connect your PG to the CPU
- Transfer the project into the CPU by means of **PLC** > *Load into module* in your project configuration tool.
- Install an MMC and transfer the application program to the MMC by clicking on **PLC** > *Copy RAM to ROM*.
- During the write operation the "MC"-LED of the CPU blinks. For internal reasons the message signalizing completion of the write operation arrives too soon. The write operation is only complete when the LED gets off.

Configuration of the CPU parameters

Outline

Except of the Profibus parameters of the CPU 21xDP the **CPU parameterization** takes place in the **parameter dialog of the CPU 315-2DP**. The parameterization of the Profibus section of the CPU 21xDP takes place via the parameterization dialog of the CPU 21xDP.



Parameterization CPU 21x under CPU 315-2DP

Per double-click on the CPU 315-2DP you reach the parameterization window of your CPU 21x. Via the different registers you may access all parameters of the CPU 315-2DP. The parameters are those of the CPU 21x from VIPA.

Please regard, that at this time not all the parameters are supported.

Supported parameters

The CPU 21x doesn't use all the parameters that may be defined in the Siemens SIMATIC Manager.

The following parameters are currently employed by the CPU:

General:

- MPI address of the CPU
- Baud rate (19.2kBaud, 187 kBaud)
- maximum MPI address

Startup:

- Startup when expected/actual configuration differ
- "Finished" message by modules
- Transfer of parameters to modules

Retentive memory:

- Number of Memory bytes starting with MB0
- Number of S7 Timers starting with T0
- Number of S7Counters starting with C0

Protection:

- Level of protection removable with password

Time-of-Day Interrupts:

OB10: active, execution, start date, time-of-day

Cyclic Interrupts:

OB35: execution

Cycle Clock Memory:

- Scan Cycle monitoring time
- Scan Cycle load from communication
- OB85: Call up at I/O access error
- Clock bit memory with memory byte no.

Parameterization Profibus section of CPU 21xDP

You may reach the parameterization window for the Profibus section of the CPU 21x by double-clicking on the added System 200V-CPU under VIPA_CPU21x.

Via the different registers you may access all Profibus parameters of the CPU 21x.

More information is to find in the chapter "Deployment CPU 21xDP".

Project transfer

Outline

There are two possibilities to transfer your project into the CPU:

- Transfer via MPI
- Transfer via MMC when using a MMC programmer

Transfer via MPI

The structure of a MPI network is principally the same than the one of a 1.5MBaud Profibus net. This means that the same rules are valid and for both networks you use the same components for building.

Per default the MPI net runs with 187kBaud.

Every bus participant identifies itself at the bus with an unique MPI address.

You link up the single participants via bus connectors and the Profibus bus cable.

Terminating resistor

A line has to be terminated with its ripple resistor. For this you switch on the terminating resistor at the first and the last participant of a network or a segment.

Please take care that those participants with the terminating resistor are supplied with power during start-up and operation.

Approach

- Connect your PG resp. PC with your CPU via MPI.

  If your PG doesn't support MPI, you may use the VIPA "Green Cable" to establish a point-to-point connection.

  The "Green Cable" has the order number VIPA 950-0KB00 and may only be used with MPI interface of VIPA CPUs.
- Configure the MPI interface of your PC.
- Via **PLC** > *Load to module* you transfer your project into the CPU.
- If you want to save your project on MMC additionally, plug-in a MMC and transfer your user application via **PLC** > *Copy RAM to ROM*.

  During the write process the "MC"-LED at the CPU is blinking. Due to the system, the completion of the write operation arrives too soon. It is only completed when the LED has been extinguished.

Configure MPI

Hints for configuring a MPI interface are to find in the documentation of your programming software. At this place only the usage of the "Green Cable" from VIPA shall be shown, together with the programming tool from Siemens.

The "Green Cable" establishes a connection between the COM interface of the PC and the MP^{2}I interface of the CPU.

Attention!

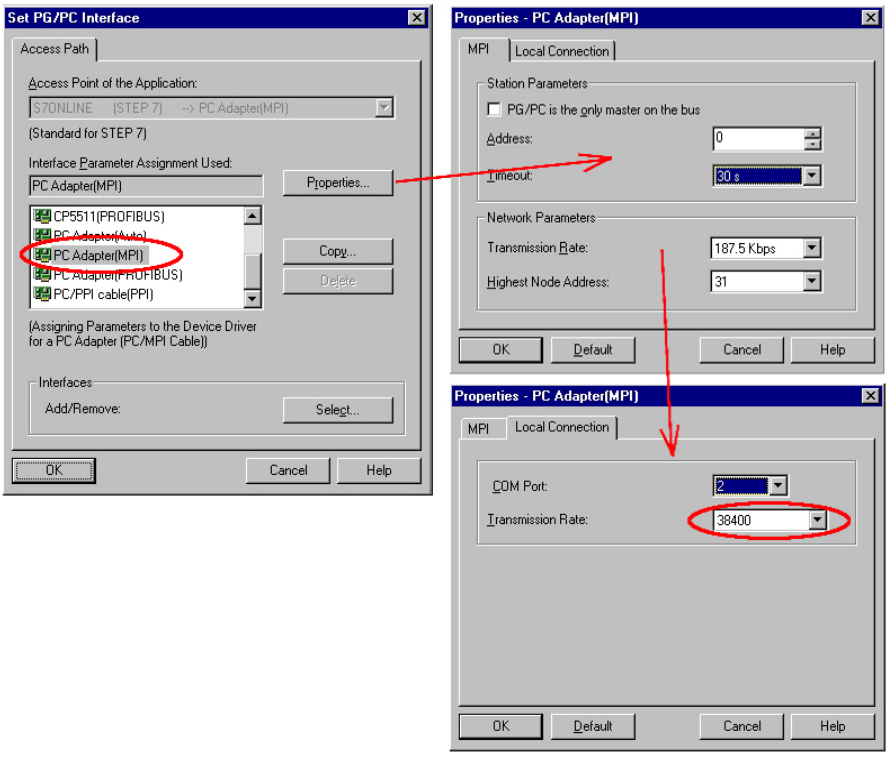
Please regard, that you may use the "Green Cable" exclusively at VIPA CPUs with MP^{2}I-interface!

Please regard the hints for deploying the Green Cable and the MP^{2}I jack in chapter Principles.

Approach

- Start the SIMATIC Manager from Siemens.
- Choose **Options** > *Configure PU/PC interface.*

  → The following dialog window appears, where you may configure the used MPI interface:



- Choose " PC Adapter (MPI) ", possibly you have to add this first.
- Click on [Properties].

  → In the following two dialog windows you may configure your PC adapter like shown in the picture.

Note!

Please make sure to adjust the transfer rate to 38400Baud when using the "Green Cable" from VIPA.

Usage of the MMC

As external storage medium the **M**ulti **M**edia **C**ard (MMC) is used (Order-No.: VIPA 953-0KX00).

The reading of the MMC takes always place after an OVERALL RESET.

You may write on the MMC via a write command from the hardware configurator from Siemens or with a MMC reading device from VIPA (OrderNo.: VIPA 950-0AD00). Thus it is possible to create your applications at the PC, copy them on MMC and transfer them into the CPU by plugging-in the MMC.

The MMC modules are delivered preformatted with the file system FAT16.

Needed files

There may exist several projects and subfolders on one MMC module. You just have to take care, that the recent project is stored in the root directory and has the name: **S7PROG.WLD**.

Transfer CPU → MMC

When the MMC has been installed, the write command stores the application program of the battery buffered RAM at the MMC.

The write command is controlled by means of the Siemens hardware configurator via **PLC** > *Copy RAM to ROM*.

During the write operation the yellow "MC"-LED of the CPU is blinking.

Transfer MMC → CPU

The transfer of the user application from the MMC to the CPU always takes place after an OVERALL_RESET. The blinking of the yellow "MC"-LED indicates the active transfer process.

If there is no valid user application on the MMC or if the transfer fails, an OVERALL_RESET of the CPU takes place and the "STOP"-LED blinks three times.

Note!

If the user application exceeds the user memory of the CPU, the content of the MMC is not transferred into the CPU.

If you initiate a write command and there is no MMC plugged in, an error message about insufficient memory occurs.

It is advisable to compress the application program before transferring it into the CPU, because this is not initialized automatically.

Operating modes

Outline

The CPU can be in one of 3 operating modes:

- STOP
- START-UP
- RUN

Certain conditions in the operating modes START-UP and RUN require a specific reaction from the system program. In this case the application interface is often provided by a call to an organization block that was included specifically for this event.

Operating mode STOP

- Processing of the application program has stopped.
- If the program was being processed, the values of counters, timers, flags and the contents of the process image are retained during the transition to the STOP mode.
- Outputs are inhibited, i.e. all digital outputs are disabled.
- RUN-LED off
- STOP-LED on

Operating mode START-UP

- During the transition from STOP to RUN a call is issued to the start-up organization block OB 100. The length of this OB is not limited. The processing time for this OB is not monitored. The start-up OB may issue calls to other blocks.
- All digital outputs are disabled during the start-up, i.e. outputs are inhibited.
- RUN-LED blinks
- STOP-LED off

When the CPU has completed the start-up OB, it assumes the operating mode RUN.

Operating mode RUN

- The application program in OB 1 is processed in a cycle. Under the control of alarms other program sections can be included in the cycle.
- All timer and counters being started by the program are active and the process image is updated with every cycle.
- The BASP-signal (outputs inhibited) is deactivated, i.e. all digital outputs are enabled.
- RUN-LED on
- STOP-LED off

Overall Reset

Outline

During the OVERALL_RESET the entire user memory (RAM) is erased. Data located in the memory card is not affected.

You have 2 options to initiate an OVERALL RESET:

- initiate the overall reset by means of the function selector switch
- initiate the overall reset by means of the Siemens SIMATIC Manager

Note!

You should always issue an overall reset to your CPU before loading an application program into your CPU to ensure that all blocks have been cleared from the CPU.

Overall reset by means of the function selector

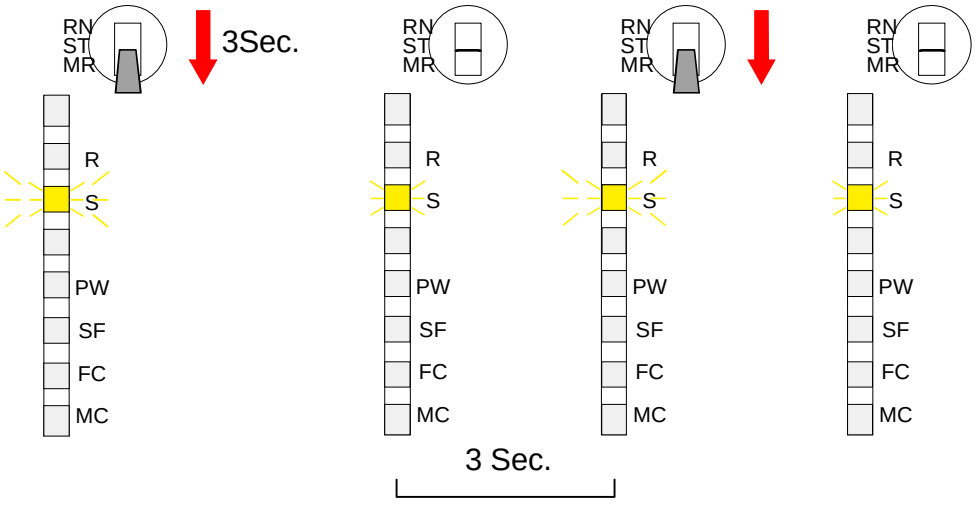
Condition

The operating mode of the CPU is STOP. Place the function selector on the CPU in position "ST" → the S-LED is on.

Overall reset

- Place the function selector in the position MR and hold it in this position for app. 3 seconds. → The S-LED changes from blinking to permanently on.
- Place the function selector in the position ST and switch it to MR and quickly back to ST within a period of less than 3 seconds.
  → The S-LED blinks (overall reset procedure).
- The overall reset has been completed when the S-LED is on permanently. → The S-LED is on.

The following figure illustrates the above procedure:



Automatic reload

At this point the CPU attempts to reload the parameters and the program from the memory card. → The lower LED (MC) blinks.

When the reload has been completed the LED is extinguished. The operating mode of the CPU will be STOP or RUN, depending on the position of the function selector.

Overall reset by means of the Siemens SIMATIC Manager

Condition

The operating mode of the CPU must be STOP.
You may place the CPU in STOP mode by the menu command **PLC** > *Operating mode*.

Overall reset

You may request the OVERALL_RESET by means of the menu command **PLC** > *Clear/Reset*.

In the dialog window you may place your CPU in STOP mode if this has not been done as yet and start the overall reset.

The S-LED blinks during the overall reset procedure.

When the S-LED is on permanently the overall reset procedure has been completed.

Automatic reload

At this point the CPU attempts to reload the parameters and the program from the memory card. → The lower LED (without label) blinks.

When the reload has been completed, the LED is extinguished. The operating mode of the CPU will be STOP or RUN, depending on the position of the function selector.

Set back to factory setting

The following approach deletes the internal RAM of the CPU completely and sets it back to the delivery state.

Please regard that the MPI address is also set back to default 2!

- Push down the reset lever for app. 30 seconds. The ST-LED blinks. After a few seconds the LED turns to static light. Count the number of static light phases because now the LED switches between static light and blinking.
- After the 6th static light you release the reset lever and push it down again shortly. Now the green RUN-LED is on once. This means that the RAM is totally deleted.
- Turn the power supply off and on again.

Firmware update

Outline

At CPU, DP-Master and CP you may execute a firmware update via MMC starting with firmware version 3.3.3.

The latest 2 firmware versions can be found in the service area at www.vipa.de and at the ftp server at ftp.vipa.de.

Attention!

When installing a new firmware you have to be extremely careful. Under certain circumstances you may destroy the CPU, for example if the voltage supply is interrupted during transfer or if the firmware file is defective.

In this case, please call the VIPA-Hotline!

Please regard that the version of the update firmware has to be different from the existing firmware otherwise no update is executed.

Find out CPU firmware version

A label on the rear of the module indicates the firmware version.

You may display the current firmware version of your CPU via the SIMATIC Manager from Siemens. To display the firmware version, you go online with the CPU via your PG or PC and start the SIMATIC Manager from Siemens. Via **PLC** > *Module status,* register "General", the current firmware version is evaluated and displayed.

Note!

The server stores always the latest two firmware versions.

Preconditions for ftp access

For the display of ftp sites in your web browser you may have to execute the following adjustments:

Internet Explorer

ftp access only with version 5.5 or higher

Options > Internet options, Register "Advanced" in the area "Browsing":

- activate: "Enable folder view for ftp sites"
- activate: "Use passive ftp ..."

Netscape

ftp- access only with version 6.0 or higher

No further adjustments are required

If you still have problems with the ftp access, please ask your system operator.

MMC update using reserved file names

By means of reserved file names in the CPU 21x you may transfer the firmware updates per MMC for the following components:

Component	Possible with CPU version	File name at ftp server	New file name at MMC
CPU	V. 3.3.3	order no._release.version	firmware.bin
Profibus master	V. 3.0.5	Bb000089.version	dpm00.bin *)
CANopen master	V. 3.4.8 and CAN master V. 1.0.7	VIPA_order no._version.bin	can00.bin
CP (21x-2BT10)	V. 3.6.7	px000018_version.zip	px000018.pkg

*) When deploying the CPU firmware version V 3.3.3, the file name of the firmware update has to be dpm.bin!

Load firmware and transfer it to MMC with reserved file name

- Go to ftp.vipa.de/support/firmware
- Navigate to *System 200V*. Here the components corresponding firmware can be found.
- Download the corresponding file. The table above can be used for a check of the file name structure.
- Copy the file to your MMC. Rename the file to the corresponding reserved file name of the table.

Transfer firmware from MMC into CPU

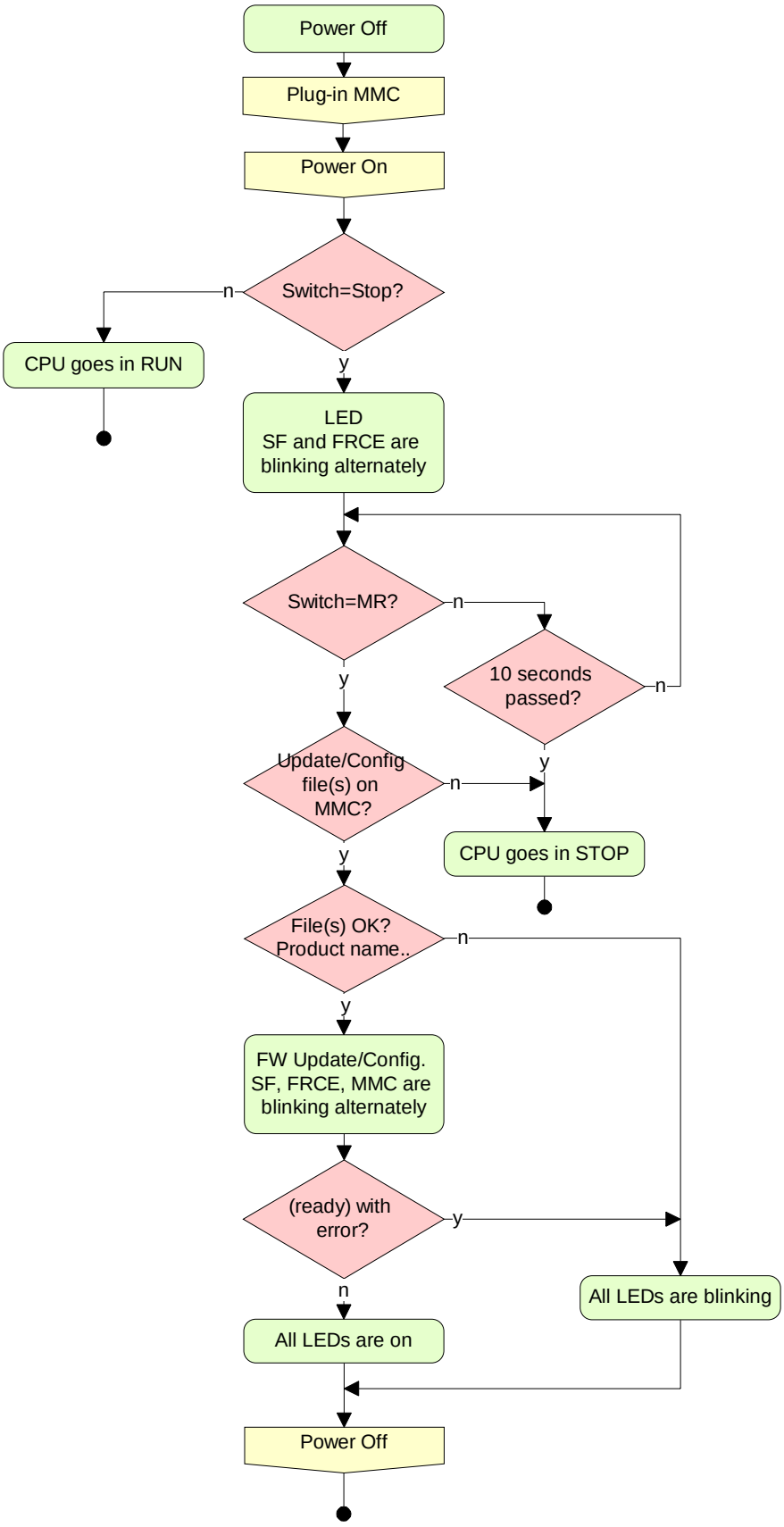
- Get the RUN-STOP lever of your CPU in position STOP.
- Turn off the power supply.
- Plug the MMC with the firmware into the CPU. Please take care of the correct plug-in direction of the MMC.
- Turn on the power supply.
- After a short boot-up time, the alternate blinking of the LEDs SF and FC shows that the firmware file has been found on the MMC.
- You start the transfer of the firmware as soon as you tip the RUN/STOP lever downwards to MRES within 10s. The CPU shows the transfer via a LED blink line.
- During the update process, the LEDs SF, FC and MC are alternately blinking. This may last several minutes.
- The update is successful finished when all CPU-LEDs are on. If they are blinking fast, an error occurred.

Note!

Starting with the WinNCS version 3.1.1 you may update the firmware of the CP online via the parameterization window. The CP portion supports this function from CP firmware version 2.1.2 on.

Flowchart for firmware update

The following flowchart illustrates the CPU behavior at firmware update:



Using test functions for the control and monitoring of variables

Outline

For troubleshooting purposes and to display the status of certain variables you may access certain test functions via the menu item **Test** of the Siemens SIMATIC Manager.

The status of the operands and the VKE are displayed by means of the test function **Test** > *Status*.

You are able to modify and/or display the status of variables by means of the test function **PLC** > *Monitor/control variables*.

Test > *Status*

This test function displays the current signal state and the VKE of the different operands while the program is being executed.

It is also possible to enter corrections to the program.

Note!

When using the test function *Monitoring* the PLC has to be in RUN mode!

The processing of states may be interrupted by means of jump commands or by timer and process-related alarms. At the breakpoint the CPU stops collecting data for the status display and instead of the required data it only provides the PG with data containing the value 0.

For this reason jumps or time and process alarms may result in the value displayed during program execution remaining at 0 for the items below:

- the result of the logical operation VKE
- Status / AKKU 1
- AKKU 2
- Condition byte
- absolute memory address SAZ. In this case SAZ is followed by a "?".

The interruption of the processing of statuses does not change the execution of the program but it only shows that the data displayed is no longer valid after from the point where the interrupt occurred.

PLC >
Monitor/control variables

This test function returns the condition of a selected operand (inputs, outputs, flags, data word, counters or timer) at the end of program execution.

This information is obtained from the process image of the selected operands. During the "processing check" or in operating mode STOP the periphery is read directly from the inputs. Otherwise only the process image of the selected operands is displayed.

Control of outputs

It is possible to check the wiring and proper operation of output modules.

You may set outputs to any desired status with or without a control program. The process image is not modified but outputs are no longer inhibited.

Control of variables

The following variables may be modified:

E, A, M, T, Z, and D.

The process image of binary and digital operands is modified independently of the operating mode of the CPU 21x.

When the operating mode is RUN the program is executed with the modified process variable. When the program continues they may, however, be modified again without notification.

Process variables are controlled asynchronously with respect to the execution sequence of the program.

Chapter 4 Deployment of the CPU 21x-2BT10 with TCP/IP

Outline

The following chapter describes applications of the CPU 21x-2BT10 and the communication using TCP/IP. Please regard the chapter „Fast introduction“ where you find all information compressed required for the project engineering of the CPU 21x-2BT10. After the fast introduction, the mentioned steps are described in detail.

The following text describes:

- Basics about a Twisted-Pair network
- Project engineering of a CP communication
- ORG format for the communication with a foreign system
- Example

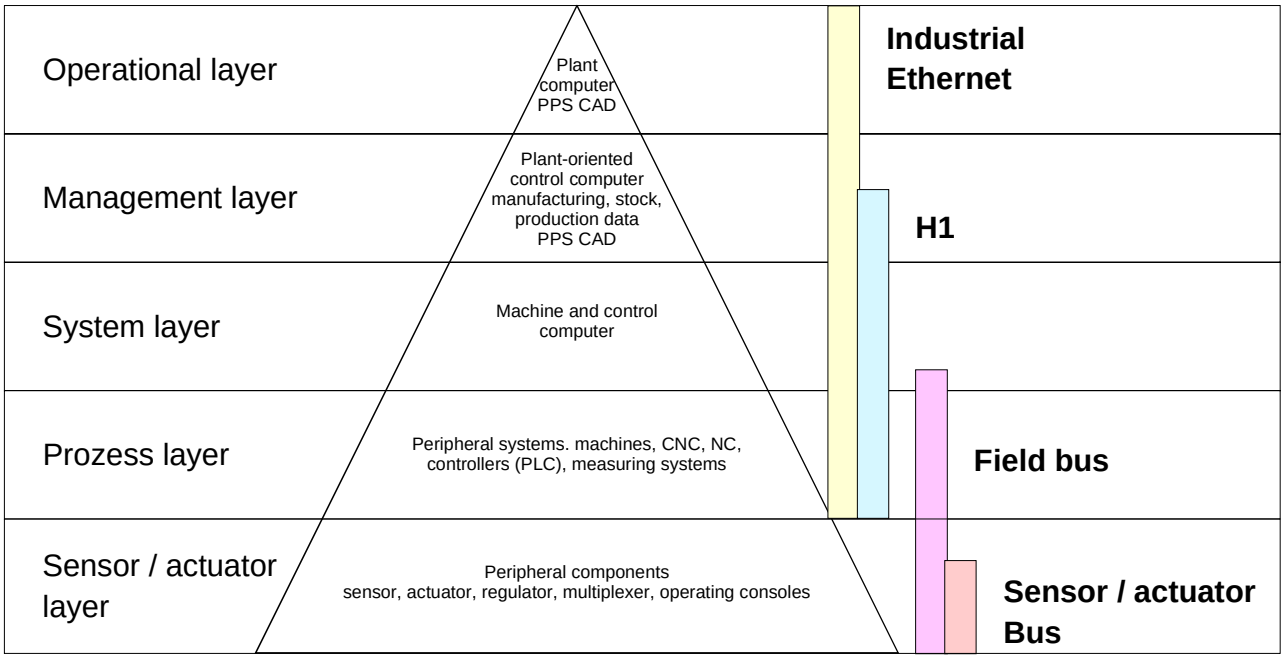
Contents

Industrial Ethernet in automation

Overview

The flow of information in a company presents a vast spectrum of requirements that must be met by the communication systems. Depending on the area of business the bus system or LAN must support a different number of users, different volumes of data must be transferred and the intervals between transfers may vary, etc.

It is for this reason that different bus systems are employed depending on the respective task. These may be subdivided into different classes. The following model depicts the relationship between the different bus systems and the hierarchical structures of a company:



Industrial Ethernet

Industrial Ethernet is an electrical net based on shielded twisted pair cabeling or optical net based on optical fibre.

Industrial Ethernet is defined by the international standard IEEE 802.3. The net access of Industrial Ethernet corresponds to IEEE 802.3 - CSMA/CD (**C**arrier **S**ense **M**ultiple **A**ccess/**C**ollision **D**etection) scheme: every station “listens” on the bus cable and receives communication messages that are addressed to it.

Stations will only initiate a transmission when the line is unoccupied. In the event that two participants should start transmitting simultaneously, they will detect this and stop transmitting to restart after a random delay time has expired.

Using switches there is the possibility for communication without collisions.

ISO/OSI reference model

Overview

The ISO/OSI reference model is based on a proposal that was developed by the International Standards Organization (ISO). This represents the first step towards an international standard for the different protocols. It is referred to as the ISO-OSI layer model. OSI is the abbreviation for **O**pen **S**ystem **I**nterconnection, the communication between open systems. The ISO/OSI reference model does not represent a network architecture as it does not define the services and protocols used by the different layers. The model simply specifies the tasks that the different layers must perform.

All current communication systems are based on the ISO/OSI reference model which is defined by the ISO 7498 standard. The reference model structures communication systems into 7 layers that cover different communication tasks. In this manner the complexity of the communication between different systems is divided amongst different layers to simplify the task.

The following layers have been defined:

Layer	Function
Layer 7	Application Layer
Layer 6	Presentation Layer
Layer 5	Session Layer
Layer 4	Transport Layer
Layer 3	Network Layer
Layer 2	Data Link Layer
Layer 1	Physical Layer

Depending on the complexity and the requirements of the communication mechanisms a communication system may use a subset of these layers.

Layers

Layer 1 Bit communication layer (physical layer)

The bit communication layer (physical layer) is concerned with the transfer of data bits via the communication channel. This layer is therefore responsible for the mechanical, electrical and the procedural interfaces and the physical communication medium located below the bit communication layer:

- Which voltage represents a logical 0 or a 1?
- The minimum time that the voltage be present to be recognized as a bit.
- The pin assignment of the respective interface.

Layer 2 Security layer (data link layer)

This layer performs error-checking functions for bit strings transferred between two communicating partners. This includes the recognition and correction or flagging of communication errors and flow control functions.

The security layer (data link layer) converts raw communication data into a sequence of frames. This is where frame limits are inserted on the transmitting side and where the receiving side detects them. These limits consist of special bit patterns that are inserted at the beginning and at the end of every frame. The security layer often also incorporates flow control and error detection functions.

The data security layer is devided into two sub-levels, the LLC and the MAC level.

The MAC (**M**edia **A**ccess **C**ontrol) is the lower level and controls how senders are sharing a single transmit channel.

The LLC (**L**ogical **L**ink **C**ontrol) is the upper level that establishes the connection for transferring the data frames from one device into the other.

Layer 3 Network layer

The network layer is an agency layer.

Business of this layer is to control the exchange of binary data between stations that are not directly connected. It is responsible for the logical connections of layer 2 communication. Layer 3 supports the identification of the single network addresses and the establishing and disconnecting of logical communication channels.

Additionally, layer 3 manages the prior transfer of data and the error processing of data packets. IP (**I**nternet **P**rotocol) is based on Layer 3.

Layer 4 Transport layer

Layer 4 connects the network structures with the structures of the higher levels by dividing the messages of higher layers into segments and pass them on to the network layer. Hereby, the transport layer converts the transport addresses into network addresses.

Common transport protocols are: TCP, SPX, NWLink and NetBEUI.

Layers continued...

Layer 5 Session layer

The session layer is also called the communication control layer. It relieves the communication between service deliverer and the requestor by establishing and holding the connection if the transport system has a short time fail out.

At this layer, logical users may communicate via several connections at the same time. If the transport system fails, a new connection is established if needed.

Additionally this layer provides methods for control and synchronization tasks.

Layer 6 Presentation layer

This layer manages the presentation of the messages, when different network systems are using different representations of data.

Layer 6 converts the data into a format that is acceptable for both communication partners.

Here compression/decompression and encrypting/decrypting tasks are processed.

This layer is also called interpreter. A typical use of this layer is the terminal emulation.

Layer 7 Application layer

The application layer is the link between the user application and the network. The tasks of the application layer include the network services like file, print, message, data base and application services as well as the according rules.

This layer is composed from a series of protocols that are permanently expanded following the increasing needs of the user.

Principles

Network (LAN)

A network res. LAN (**l**ocal **a**rea **n**etwork) provides a link between different stations that enables them to communicate with each other.

Network stations consist of PCs, IPCs, TCP/IP adapters, etc.

Network stations are separated by a minimum distance and connected by means of a network cable. The combination of network stations and the network cable represent a complete segment.

All the segments of a network form the Ethernet (physics of a network).

Twisted Pair

In the early days of networking the Triaxial- (yellow cable) or thin Ethernet cable (Cheapernet) was used as communication medium. This has been superseded by the twisted-pair network cable due to its immunity to interference. The CPU 21xNET module has a twisted-pair connector.

The twisted-pair cable consists of 8 cores that are twisted together in pairs. Due to these twists this system is provides an increased level of immunity to electrical interference. For linking please use twisted pair cable which at least corresponds to the category 5.

Where the coaxial Ethernet networks are based on a bus topology the twisted-pair network is based on a point-to-point scheme.

The network that may be established by means of this cable has a star topology. Every station is connected to the star coupler (hub/switch) by means of a separate cable. The hub/switch provides the interface to the Ethernet.

Hub (repeater)

The hub is the central element that is required to implement a twisted-pair Ethernet network.

It is the job of the hub to regenerate and to amplify the signals in both directions. At the same time it must have the facility to detect and process segment wide collisions and to relay this information. The hub is not accessible by means of a separate network address since it is not visible to the stations on the network.

A hub has provisions to interface to Ethernet or to another hub res. switch.

Switch

A switch also is a central element for realizing Ethernet on Twisted Pair. Several stations res. hubs are connected via a switch. Afterwards they are able to communicate with each other via the switch without interfering the network. An intelligent hardware analyzes the incoming telegrams of every port of the switch and passes them collision free on to the destination stations of the switch. A switch optimizes the bandwidth in every connected segment of a network. Switches enable exclusive connections between the segments of a network changing at request.

Protocols

Overview

Protocols define a set of instructions or standards that enable computer to establish communication connections and exchange information as error free as possible.

A commonly established protocol for the standardization of the complete computer communication is the so called ISO/OSI layer model, a model based upon seven layers with rules for the usage of hardware and software (see ISO/OSI reference model above).

The CPU 21xNET from VIPA uses the following protocols

- TCP/IP
- UDP
- RFC1006 (ISO-ON-TCP)

The protocols are described in the following:

TCP/IP

TCP/IP protocols are available on all major systems. At the bottom end this applies to simple PCs, through to the typical mini-computer up to mainframes.

For the wide spread of internet accesses and connections, TCP/IP is often used to assemble heterogeneous system pools.

TCP/IP, standing for **T**ransmission **C**ontrol **P**rotocol and **I**nternet **P**rotocol, collects a various range of protocols and functions.

TCP and IP are only two of the protocols required for the assembly of a complete architecture. The application layer provides programs like "FTP" and "Telnet" for the PC.

The application layer of the Ethernet part of the CPU 21xNET is defined with the user application using the standard handling blocks.

These user applications use the transport layer with the protocols TCP and UDP for the data transfer which themselves communicate via the IP protocol with the internet layer.

IP

The internet protocol covers the network layer (Layer 3) of the ISO/OSI layer model.

The purpose of IP is to send data packages from on PC to another passing several other PCs. These data packages are referred to as datagrams. The IP doesn't guarantee the correct sequence of the datagrams nor the delivery at the receiver.

For the unambiguous identification between sender and receiver 32Bit addresses (IP addresses) are used that are normally written as four octets (exactly 8Bit), e.g. 172.16.192.11.

These internet addresses are defined and assigned worldwide from the DDN network (Defense Department Network), thus every user may communicate with all other TCP/IP users.

One part of the address specifies the network, the rest serves the identification of the participants inside the network. The boarder between the network and the host part is variable and depends on the size of the network.

To save IP addresses, so called *NAT router* are used that have one official IP address and cover the network. Then the network can use any IP address.

TCP

The TCP (**T**ransmission **C**ontrol **P**rotocol) bases directly on the IP and thus covers the transport layer (layer 4) of the OSI layer model. TCP is a connection orientated end-to-end protocol and serves the logic connection between two partners.

TCP guarantees the correct sequence and reliability of the data transfer. Therefore you need a relatively large protocol overhead that slows down the transfer speed.

Every datagram gets a header of at least 20Byte. This header also contains a sequence number identifying the series. This has the consequence that the single datagrams may reach the destination on different ways through the network.

Using TCP connections, the telegram length is not transmitted. This means that the recipient has to know how many bytes belong to a message. To transfer data with variable length you may begin the user data with the length information and evaluate this at the counter station.

Properties

- Besides of the IP address ports are used for the addressing. A port address should be within the range of 2000...65535. Partner and local ports may only be identical at one connection.
- Not depending on the used protocol, the PLC needs the VIPA handling blocks AG_SEND (FC5) and AG_RECV (FC6) for data transfer.

UDP

The UDP (**U**ser **D**atagram **P**rotocol) is a connection free transport protocol. It has been defined in the RFC768 (Request for Comment). Compared to TCP, it has much fewer characteristics.

The addressing happens via port numbers.

UDP is a fast unsafe protocol for it doesn't care about missing data packages nor about their sequence.

ISO-on-TCP RFC1006

The TCP transport service works stream orientated. This means that data packages assembled by the user not necessarily have to receive the partner in the same packaging. Depending on the data amount, packages may though come in in the correct sequence but differently packed. This causes that the recipient may not recognize the package borders anymore. For example you may send 2x 10Byte packages but the counter station receives them as 20Byte package. But for most of the applications the correct packaging is important.

Due to this you need another protocol above TCP. This purpose is defined in the protocol RFC1006. The protocol definition describes the function of an ISO transport interface (ISO 8072) basing upon the transport interface TCP (RFC793).

The basic protocol of RFC1006 is nearly identical to TP0 (Transport Protocol, Class 0) in ISO 8073.

For RFC1006 is run as protocol for TCP, the decoding takes place in the data section of the TCP package.

Properties

- In contrast to TCP here the receipt of one telegram is confirmed.
- Instead of ports TSAPs are used for the addressing besides of the IP address. The TSAP length may be 1 ... 16Byte. The entry may happen in ASCII or Hex format. Foreign and local TSAPs may only be identical at 1 connection.
- Independently of the used protocol the VIPA handling blocks AG_SEND (FC5) and AG_RECEIVE (FC6) are necessary for data transfer.
- Contrary to TCP different telegram lengths can be received using RFC1006.

IP address and subnet

IP address structure

The IP address is a 32Bit address that must be unique within the network. The IP address consists of 4 numbers that are separated by a full stop.

Every IP address is a combination of a **Net-ID** and a **Host-ID** and its structure is as follows: XXX.XXX.XXX.XXX

Range: 000.000.000.000 to 255.255.255.255

The network administrator also defines IP addresses.

Net-ID
Host-ID

The **Net**work-ID identifies a network res. a network controller that administrates the network.

The Host-ID marks the network connections of a participant (host) to this network.

Subnet mask

The Host-ID can be further divided into a **Subnet-ID** and a *new* **Host-ID** by using an bit for bit AND assignment with the **Subnet mask**.

The area of the original Host-ID that is overwritten by 1 of the Subnet mask becomes the Subnet-ID, the rest is the new Host-ID.

Subnet mask	binary all "1"		binary all "0"
IPv4 address	Net-ID	Host-ID	
Subnet mask and IPv4 address	Net-ID	Subnet-ID	*new* Host-ID

Subnet

A TCP-based communication via point-to-point, hub or switch connection is only possible between stations with identical Network-ID and Subnet-ID! Different area must be connected with a router.

The subnet mask allows you to sort the resources following your needs. This means e.g. that every department gets an own subnet and thus does not interfere another department.

Address at first start-up

At the first start-up the CP does not have any IP address. The assignment takes place using the following possibilities:

- Using Siemens SIMATIC Manager switch PG/PC interface to "TCP/IP... RFC1006". Via "Assign Ethernet address" search the CP and assign IP parameters. After that the CP is directly assigned to the new IP parameters without any restart of the CPU.
- You may assign an IP address and a subnet mask to your CP with the help of a "minimum project" and transfer this via MMC or MPI into the CPU. After a reboot of the CPU and after switching the PG/PC interface to "TCP/IP... RFC1006" you may now configure your CPU online via the CP.

Address classes

For IPv4 addresses there are five address formats (class A to class E) that are all of a length of 4 Byte = 32 Bit.

Class A	0	Network-ID (1+7 bit)	Host-ID (24 bit)
Class B	10	Network-ID (2+14 bit)	Host-ID (16 bit)
Class C	110	Network-ID (3+21 bit)	Host-ID (8 bit)
Class D	1110	Multicast group	
Class E	11110	Reserved	

The classes A, B and C are used for individual addresses, class D for multicast addresses and class E is reserved for special purposes.

The address formats of the 3 classes A, B, C are only differing in the length of Network-ID and Host-ID.

Private IP networks

To build up private IP-Networks within the internet, RFC1597/1918 reserves the following address areas:

Network class	Start IP	End IP	Standard subnet mask
A	10.0.0.0	10.255.255.255	255.0.0.0
B	172.16.0.0	172.31.255.255	255.255.0.0
C	192.168.0.0	192.168.255.255	255.255.255.0

(The Host-ID is underlined.)

These addresses can be used as net-ID by several organizations without causing conflicts, for these IP addresses are neither assigned in the internet nor are routed in the internet.

Reserved Host-Ids

Some Host-IDs are reserved for special purposes.

Host-ID = 0	Identifier of this network, reserved!
Host-ID = maximum (binary complete 1)	Broadcast address of this network

Note!

Never choose an IP address with Host-ID=0 or Host-ID=maximum!

(e.g. for class B with subnet mask = 255.255.0.0, the "172.16.0.0" is reserved and the "172.16.255.255" is occupied as local broadcast address for this network.)

Network planning

Standards and guidelines

The applicable rules and regulations have to be satisfied in order to establish reliable communications between the different stations.

These agreements define the form of the data protocol, the method of bus access and other principles that are important for reliable communications.

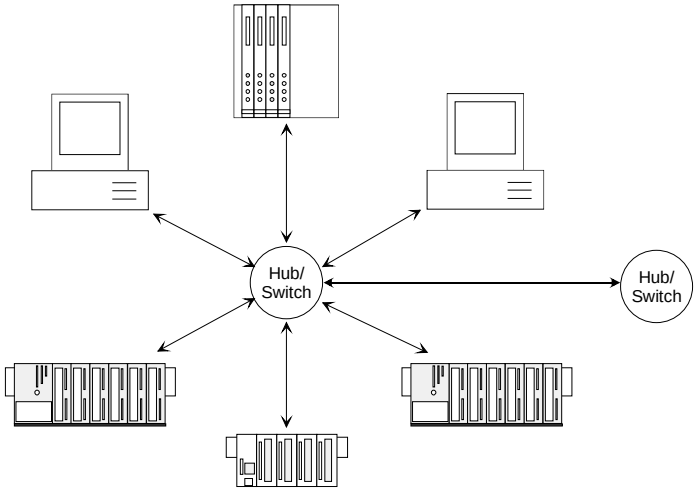
The VIPA CPU 21xNET was developed in accordance with the standards defined by ISO.

International and national committees have defined the following standards and guidelines for networking technologies:

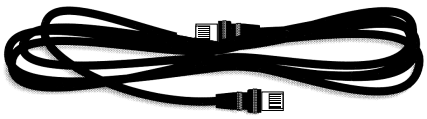
ANSI	American National Standards Institute The ANSI X3T9.5 standard currently defines the provisions for high-speed LANs (100 MB/s) based on fiber optic technology. (FDDI) Fiber Distributed Data Interface.
CCITT	Committee Consultative Internationale de Telephone et Telegraph. Amongst others, this advisory committee has produced the provisions for the connection of industrial networks (MAP) to office networks (TOP) on Wide Area Networks (WAN).
ECMA	European Computer Manufacturers Association. Has produced various MAP and TOP standards.
EIA	Electrical Industries Association (USA) This committee has issued standard definitions like RS-232 (V.24) and RS-511.
IEC	International Electrotechnical Commission. Defines certain special standards, e.g. for the Field Bus.
ISO	International Organization for Standardization. This association of national standards organizations developed the OSI-model (ISO/TC97/SC16). It provides the framework for the standardization of data communications. ISO standards are included in different national standards like for example UL and DIN.
IEEE	Institute of Electrical and Electronic Engineers (USA). The project-group 802 determines LAN-standards for transfer rates of 1 to 1000MB/s. IEEE standards often form the basis for ISO-standards, e.g. IEEE 802.3 = ISO 8802.3.

Overview of components

The CP is exclusively used for employment in a Twisted-Pair network. Within a Twisted-Pair network all participating stations are connected in star topology via a Twisted-Pair cable to a hub/switch which is also able to communicate with another hub/switch. Two connected stations are building a segment where the length of the Twisted-Pair cable between two stations must be max. 100m.



Twisted Pair cable



At twisted pair cable has 8 conductors twisted together in pairs. The different conductors have a diameter of 0.4 to 0.6mm.

For linking please use twisted pair cable which at least corresponds to the category 5.

Analyzing the requirements

- What is the size of the area that must be served by the network?
- How many network segments provide the best solution for the physical (space, interference related) conditions encountered on site?
- How many network stations (SPS, IPC, PC, transceiver, bridges if required) must be connected to the cable?
- What is the distance between the different stations on the network?
- What is the expected “growth rate” and the expected number of connections that must be catered for by the system?
- What data amount has to be handled (band width, accesses/sec.)?

Drawing a network diagram

Draw a diagram of the network. Identify every hardware item (i.e. station cable, hub, switch). Observe the applicable rules and restrictions.

Measure the distance between all components to ensure that the maximum length is not exceeded.

Linking with NetPro

Please regard that the following software packages must be installed for the project engineering:

- Siemens SIMATIC Manager V. 5.1 and vipa_21x.gsd (is included).
- Siemens SIMATIC NET

To enable the stations to communicate with each other you have to configure the required (sub)nets in the Siemens SIMATIC Manager res. NetPro following this approach:

- Create one or more subnets of the wanted type in your project.
- Adjust the properties of the subnets.
- Connect your participants logically to the subnet.
- Establish communication connections between the single stations.

Net-Project variants

You may administrate several subnets in one project. Every station has to be created once. A station may be assigned to several subnets by assigning the CPs accordingly.

In the following typical project variants for networks are listed:

1 subnet - 1 project

The simplest case is a plant with stations that have to be connected via one subnet of the type Industrial Ethernet.

For this you create an object "Ethernet". Stations that are created in the same project refer to this object when they are configured as net knots. They may then be selected directly. Foreign devices are listed in this subnet as "Other station" during project engineering.

2 or more subnets - 1 project

Due to different tasks of the stations or due to the expansion of your plant it may be necessary to create several nets. Here you may create several subnets in one project and configure the stations easily for communication.

1 or more subnets – several part projects

At complex linked plants it is sensible to administrate plant parts in several part projects. Here it may be necessary to create project exceeding connections. For this the Siemens SIMATIC Manager starting with V5.2 provides the multi project function. This function allows you to split projects and join them again. A more detailed description is to be found in the manual of the Siemens SIMATIC Manager.

Subnet exceeding connections

These are connections that long into another subnet due to the complexity of the plant. The subnets are connected via a router. By setting a router address during the hardware configuration of your CP you may instruct the CP to include the according subnet via this router for communication.

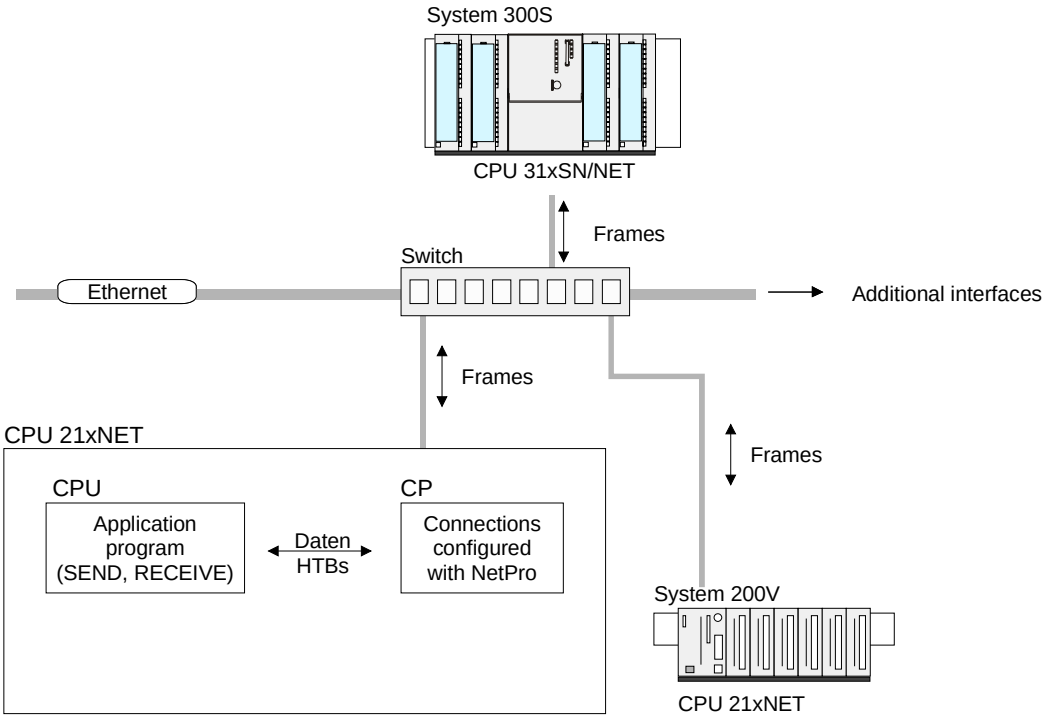
Communication possibilities of the CP

Communication between CP 243 and CPU

The internal CP of the CPU 21x-2BT10 is directly connected to the CPU via a Dual-Port-RAM. The CPU manages the data exchange with the VIPA handling blocks AG_SEND (FC5) and AG_RECV (FC6).

The communication via the according protocols are controlled by connections that are parameterized in the Siemens project engineering tool NetPro and that may be transferred into the CPU via MMC, MPI or directly via Ethernet.

For the transfer via Ethernet, your CP must be connected to Ethernet with valid IP parameters. The assignment takes place either using the corresponding menu item of the Siemens SIMATIC Manager or via a minimum project where the IP parameters are defined. This project can be transferred to the CPU via MMC or MPI.



Communication types

The CP supports the following communication types:

- PG/OP communication
- Configurable connections

PG/OP communication

The PG/OP communication serves the loading of programs and configuration data, for test and diagnostic functions as well as for operating and monitoring a plant. Here you may access the CPU online via the CP (Ethernet).

With CP firmware version 1.7.4 and up a simultaneous access of up to 32 participants is possible. Please note for each PG and OP communication 1 connection is reserved.

Configurable connections

Configurable connections are connections for the communication between PLC stations. The connections may be configured with the Siemens project engineering tool NetPro.

The following table shows the combination option with the different operating modes:

Combination options

Connection partner	Connection type	Conn. Establ.	Connection	Operating mode
Specified in NetPro (in recent project)	TCP / ISO-on-TCP	active/passive	specified	SEND/RECEIVE
	UDP	-		
Unspecified in NetPro (in recent project)	TCP / ISO-on-TCP	active	specified	SEND/RECEIVE
		passive	part specified (Port)	SEND/RECEIVE FETCH PASSIV WRITE PASSIV
			unspecified	
	UDP	-	specified	SEND/RECEIVE
Unspecified in NetPro (in "unknown project")	TCP / ISO-on-TCP	active	unspecified (connection name)	SEND/RECEIVE
		passive	unspecified (connection name)	SEND/RECEIVE FETCH PASSIV WRITE PASSIV
	UDP	-	unspecified (connection name)	SEND/RECEIVE
All Broadcast stations	UDP	-	specified (Port, Broadcast addr)	SEND
All Multicast stations	UDP	-	specified (Port, Multicast group)	SEND/RECEIVE

Connection partner

Connection partner are stations at the counter side.

Specified connection partner

Every station configured in the Siemens SIMATIC Manager is entered in the list of connection partners. By setting an IP address and a subnet mask these stations are uniquely *specified*.

Unspecified connection partner

You may also set an *unspecified* connection partner. Here the connection partner may be within the *recent project* or within an *unknown project*. Connection commands to an *unknown project* must be defined via an unique connection name that has to be used in the projects of both stations. Due to the assignment via a connection name, the connection itself remains *unspecified*.

All broadcast participants

Only with UDP connections you may here send messages to all available broadcast participants. The reception is not possible. The broadcast participants are specified via one port and one broadcast address at sender and receiver.

All multicast participants

This setting allows you to send and receive multicast telegrams between the multicast participants. By setting of one port and one multicast group for sender and receiver the multicast participants are specified.

Connection types

For the communication the following connection types are available:

- **TCP** res**. ISO-on-TCP** for the secured data transfer of related data blocks between two Ethernet participants.
- **UDP** for the unsecured data transfer of related data blocks between two Ethernet.

Connection establishment

Using configurable connections there is always one station that *actively* establishes a connection. The counter station waits *passively* for the active connection. Only then productive data can be transferred.

Connection

By setting IP address and port/TSAP of the counter station, a connection is *specified*. Active connections must always be set specified. An *unspecified* connection which is only possible for passive connection establishment, IP address and port/TSAP of the counter station are not required for telegram evaluation.

There is also an option for *part specified* connections. The part specifications happens via the setting of the port. An IP address is not required.

Operating modes

Depending on the connection, the following operating modes are available:

SEND/RECEIVE

The SEND/RECEIVE interface allows the program controlled communication to any partner station via a configured connection. Here the data transfer happens by call from your user application. The FC5 and FC6 that are part of the VIPA block library are serving as interface.

This enables your control to send messages depending on process events.

FETCH/WRITE PASSIVE

With the help of FETCH/WRITE services partner systems have the direct access to memory areas. This are "passive" communication connections that have to be configured. The connections are "actively" established by the connection partner (e.g. Siemens-S5).

FETCH PASSIVE (request data)

FETCH allows a partner system to request data.

WRITE PASSIVE (write data)

This allows a partner system to write data in the data area of the CPU.

Function overview

Outline

In the following the functions are listed that are supported by the CP of the CPU 21x-2BT10 starting with CP firmware version 1.7.4:

Configurable connections

Function	Property
Maximum number of productive connections	16
TCP connections	SEND, RECEIVE, FETCH PASSIVE, WRITE PASSIVE Connection establishment active and passive, supports unspecified connection partner
ISO-on-TCP connections (RFC1006)	SEND, RECEIVE, FETCH PASSIVE, WRITE PASSIVE Connection establishment active and passive, supports unspecified connection partner
UDP connections	SEND and RECEIVE The transfer of the telegrams is not acknowledged, i.e. the loss of messages is not recognized by the send block.
UDP Broadcast connection	SEND
UDP Multicast connection	SEND and RECEIVE
Data block length	max. 64kByte (max. 2KByte at UDP)
VIPA handling blocks	For connection commands at the PLC: AG_SEND (FC5) / AG_RECEIVE (FC6) Any call without lock in all OBs

PG connections and diagnostic

Function	Property
Maximum number of PG/OP connections	32 (each 1 connection is reserved for PG and OP)
Diagnostic	Supports NCM diagnostic via Ethernet
Search within network	Supports Siemens SIMATIC Manager search
10/100MBit	Switch happens automatically

Fast introduction

Overview

At the first start-up of a CPU 21x-2BT10 the CP of the CPU 21xNET does not have any IP address. The assignment takes place directly via the hardware configuration of the Siemens SIMATIC Manager. For the project engineering of a CPU 21xNET please follow this approach:

- **Assembly and commissioning**
- **Hardware configuration** (Inclusion of CP in CPU)
- **CP project engineering** via NetPro (connection to Ethernet)
- **PLC programming** via user application (connection to PLC)
- **Transfer of the complete project to CPU**

Note!

To be compatible to the Siemens SIMATIC Manager, the CPU 21x from VIPA has to be configured as

CPU 315-2DP (6ES7 315-2AF03-0AB0) V1.2

The CP of the CPU 21xNET is always configured virtually as 4th module at the standard bus as CP343-1 (343-1EX11) from Siemens.

To be able to address the modules they have to be projected in the hardware configurator from Siemens in form of a virtual Profibus system. The full functionality of the System 200V modules is provided by inclusion of a GSD-file from VIPA.

Assembly and commissioning

- Install your System 200V with the CPU 21xNET.
- Wire the system by connecting cables for voltage supply, signals and Ethernet. A detailed description is to be found in the chapter "Assembly and installation guidelines" of the Manual HB97.
- Switch on the voltage supply. → After a short boot time, the CP is in idle.
  At the first commissioning res. after an overall reset of the CPU, the CP has no IP address. For control purposes you may now reach the CP via the MAC address. The MAC address is to be found at a small label on the module.

Assign IP parameters

For the assignment of the IP parameters such as IP address, Subnet mask etc. you have the following possibilities:

- Online using Siemens SIMATIC Manager via "Assign Ethernet Address" (at least CP-Firmware 1.7.4)
- with the help of a "minimum project" and transfer this via MMC or MPI into the CPU. After a reboot of the CPU and after switching the PG/PC interface to "TCP/IP... RFC1006" you may now configure your CPU online via the CP.

Address assignment with "Assign Ethernet Address"

Please regard this functionality is available with firmware version 1.7.4 and up.

- Start Siemens SIMATIC Manager
- Switch to "TCP/IP... RFC1006" using **Options** > *Set PG/PC interface*.
- The dialog for initialization of a station opens by **PLC** > *Assign Ethernet-Address*.
- To get the stations and their MAC address use the [Browse] button or type in the MAC Address. The Mac address can be found at a label at the side of the CPU.
- Choose if necessary the known MAC address of the list of found stations.
- Either type in the IP configuration like IP address, subnet mask and gateway. Or your station is automatically provided with IP parameters by means of a DHCP server. Depending of the chosen option the DHCP server is to be supplied with MAC address, equipment name or client ID. The client ID is a numerical order of max. 63 characters. The following characters are allowed: "hyphen", 0-9, a-z, A-Z
- Confirm with [Assign ...]

Directly after the assignment the CP is online reachable using the set IP parameters.

Address assignment with minimal project

- Start Siemens SIMATIC Manager with new project.
- Place a new System 300 station with **Insert** > *Station* > *SIMATIC 300 station*
- Activate the station "SIMATIC 300" and open the hardware configurator by clicking on "Hardware".
- Configure a rack (SIMATIC 300 \ Rack-300 \ Profile rail).
- Engineer in deputy of your CPU 21xNET the Siemens CPU 315-2DP with the order no. 6ES7 315-2AF03-0AB0 V1.2. which is to be found at SIMATIC 300 \ CPU 300 \ CPU 315-2 DP \ 6ES7 315-2AF03-0AB0. If needed, parameterize the CPU 315-2DP.
- Configure in deputy of your CP the **CP 343-1 (343-1EX11)** from Siemens at slot 4, to be found at SIMATIC 300 / CP 300 / Industrial Ethernet / CP 343-1.
- Set IP address, subnet mask and gateway at CP properties. This is the end of the *Minimal project*. After the *Minimal project* is transferred to CPU, the CPU can be accesses by means of IP address and Subnet mask of the project.

Hardware configuration

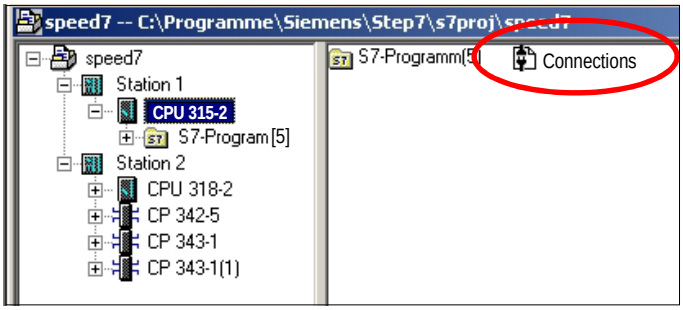
The hardware configuration takes place by the described steps "Address assignment with minimal project" whereas the project is not transferred to the CPU yet.

For project engineering of the System 200V modules you have to continue as follows:

- You have to create a new Profibus subnet with **Profibus address > 1**
- Attach the System "VIPA_CPU21x" (VIPA_21x.GSD necessary) to the subnet. The respective entries are located in the hardware catalog at *PROFIBUS DP \ Additional Field Devices \ IO \ VIPA_System_200V.* Assign **Profibus address 1** to this slave.
- Place the VIPA CPU 21xNET that you want to deploy at the **1st slot**.
- Include your System 200V modules in the location sequence starting from slot 1.
- Save your project.

Configure connections with NetPro

The link-up between the stations happens with the graphical interface NetPro. Start NetPro by clicking on a network in your project res. on connections in the CPU directory.

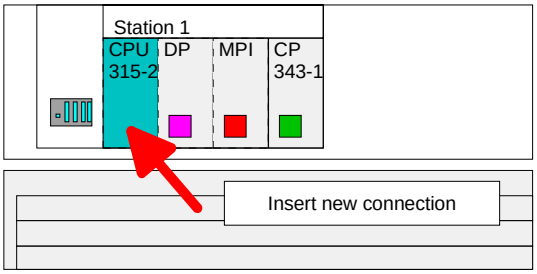


Link-up stations

For the project engineering of connections, connected stations are presumed. To link-up stations, point on the colored net mark of the according CP with the mouse and drag it to the network you want to assign. The connection is displayed graphically by a line.

Configure connections

For the project engineering of new connections click on the according CPU and choose "Insert new connection" from the context menu.



Via the dialog window you may set the parameters for a connection. The parameters ID and LADDR are required for the usage on the blocks AG_SEND res. AG_RECV (FC5 bzw. FC6).

Save and compile connections

Save and compile your project and close NetPro.

To store the CP project engineering data in the system data, you have to activate the option "Save configuration data on the CPU" (default setting) at *object properties* area *Options* in the hardware configuration of the CP.

PLC user application

For the execution of connection commands at the PLC, your CPU requires an user application. For this, exclusively the VIPA handling blocks AG_SEND (FC5) and AG_RECV (FC6) are used. The blocks are part of the VIPA library that is included in the consignment as CD (SW830).

Specify the according CP via the parameters *ID* and *LADDR* by calling FC5 res. FC6.

Transfer project

There are 3 possibilities to transfer your project into the CPU:

- Transfer via MPI
- Transfer via MMC using a card reader
- Transfer via CP (Minimal project necessary)

More detailed information about this is to be found below at "Transfer project".

The following pages provide a more detailed description of the steps of the fast introduction.

Hardware configuration

Overview

For the hardware configuration you use the hardware configurator from Siemens. Among others, you assign the IP address of the CP and configure the hardware components of your PLC.

For the deployment of the System 200V modules you have to include the modules into the hardware catalog via the GSD-file vipa_21x.gsd from VIPA.

Preconditions

Please regard that the following software packages have to be installed for the hardware configuration:

- SIMATIC Manager from Siemens V. 5.1 or higher and VIPA_21x.gsd
- SIMATIC NET

Note!

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

Include VIPA_21x-GSD-file

- Copy the delivered VIPA-GSD-file VIPA_21x.gsd into your GSD directory ... \siemens\step7\s7data\gsd
- Start the hardware configurator from Siemens
- Close all projects
- Choose **Options** > *Install new GSD-file*
- Type **VIPA_21x.gsd**

The modules of the System 200V from VIPA are now integrated in the hardware catalog and may be configured.

Note

To be compatible to Siemens SIMATIC Manager, the CPU 21x from VIPA have to be projected as

CPU 315-2DP (6ES7 315-2AF03) V1.2

The CP part from the CPU 21xNET is virtually projected as CP343-1 (343-1EX11) from Siemens at slot 4.

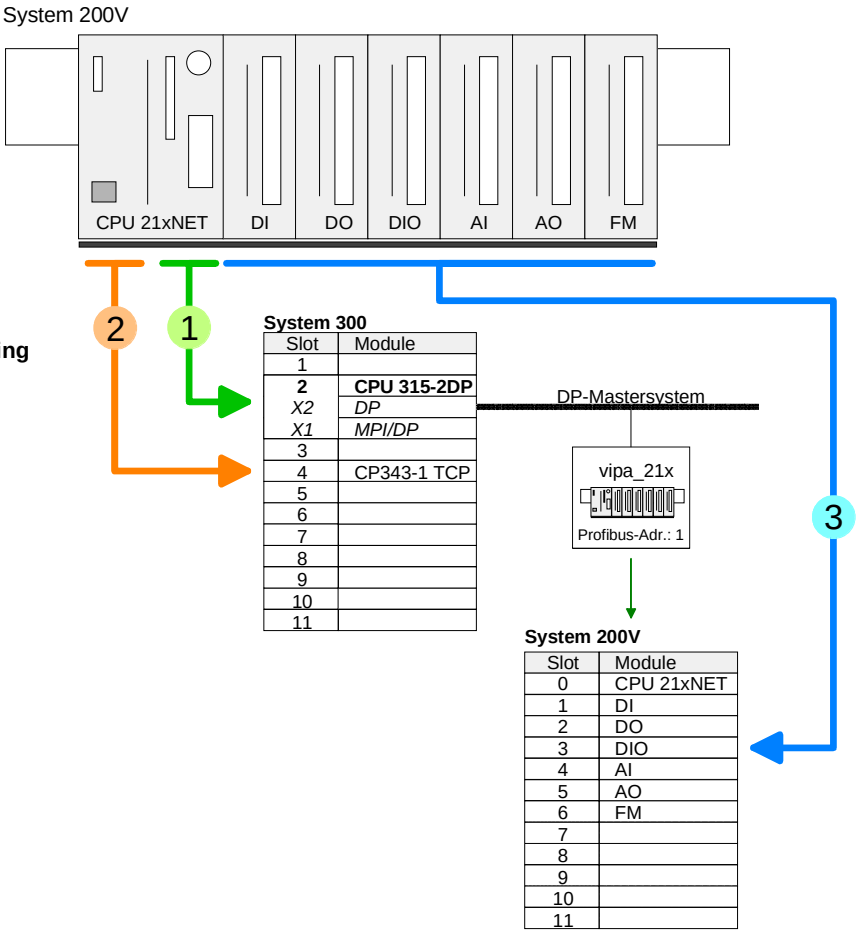
To be able to address the modules they have to be projected in the hardware configurator from Siemens in form of a virtual Profibus system. The full functionality of the System 200V modules is provided by inclusion of a GSD-file from VIPA.

Steps of project engineering

The following section describes the approach of the project engineering in the hardware configurator from Siemens using an abstract example.

project engineering

Hardware



to 1

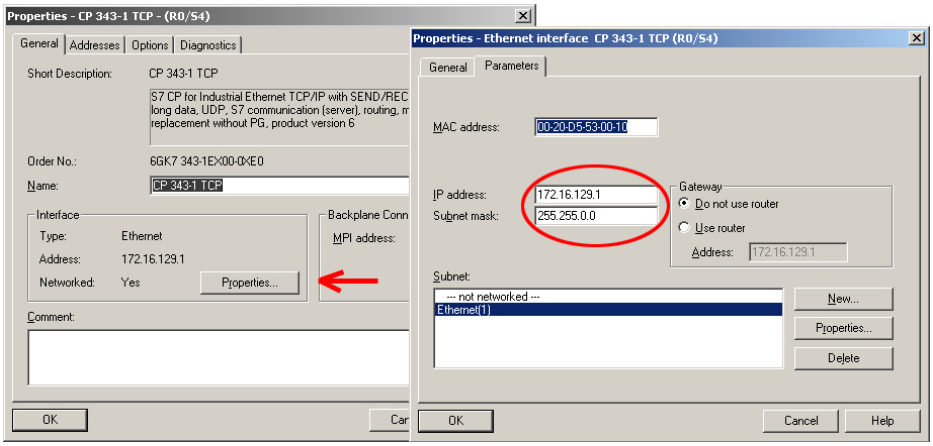
- Start the SIMATIC manager from Siemens and create a new project.
- Insert a new System 300 station via **Insert** > *Station* > *SIMATIC 300 station*.
- Activate the station "SIMATIC 300" and open the hardware configurator by clicking on "Hardware".
- Configure a rack (Simatic300 \ Rack-300 \ Rail).
- For all CPUs 21x from VIPA are configured as CPU 315-2DP, you select the CPU 315-2DP in the hardware catalog. This is to find at:

  SIMATIC 300 \ CPU-300 \ CPU 315-2 DP \ 6ES7 315-2AF03-0AB0

  V1.2

to 2

- The CP part of the CPU 21xNET is projected as CP343-1 from Siemens. Place a virtual CP343-1 (Simatic300 \ CP-300 \ Industrial Ethernet \ CP 343-1 \ 6GK7 343-1EX11 0XE0) at slot 4.
- Via a double click on the CP 343-1 you open the "properties" window. Type in the IP address and Subnet mask and choose your Subnet.



to 3

- if wished parameterize the CPU 315-2DP. Now you have to create a new Profibus subnet with **Profibus address > 1**.
- Attach the System "VIPA_CPU21x" to the subnet. The respective entries are located in the hardware catalog at *PROFIBUS DP* \ *Additional Field Devices* \ *IO* \ *VIPA_System_200V.* Assign **Profibus address 1** to this slave.
- Place the VIPA CPU 21xNET that you want to deploy at **1st slot**.
- Include your System 200V modules in the location sequence starting from plug-in location 1.
- Save and translate your project.

Configure connections

Outline

The project engineering of connections i.e. the "link-up" between stations happens in NetPro from Siemens. NetPro is a graphical user interface for the link-up of stations.

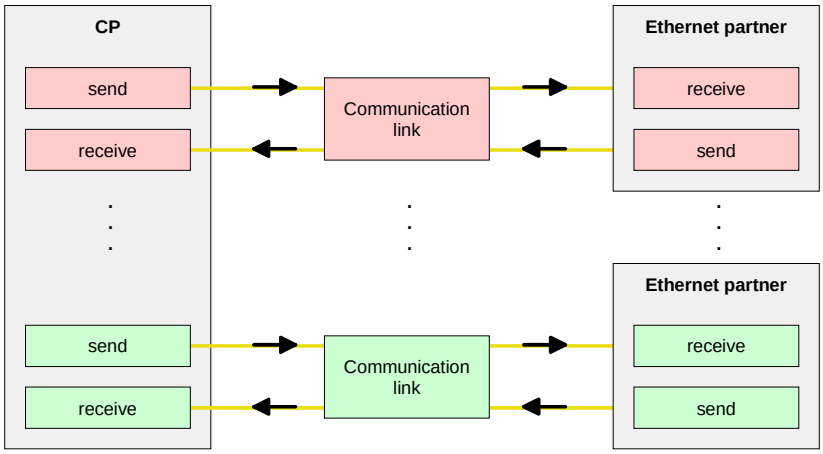
A communication connection enables the program controlled communication between two participants at the Industrial Ethernet. The communication partners may here be part of the same project or - at multi projects - separated within related part projects.

Communication connections to partners outside of a project are configured via the object "In unknown project" or via deputy objects like "Other stations" or Siemens "SIMATIC S5 Station".

Properties

The following properties are characterizing a communication connection:

- Bi-directional data transfer (Send and receive on one connection)
- Both participant have equal rights, i.e. every participant may initialize the send res. receive process event controlled.
- Except of the UDP connection, at a communication connection the address of the communication partner is set via the project engineering. Here the connection is active established by one station.



Requirements

- Siemens SIMATIC Manager V. 5.1 or higher and SIMATIC NET are installed.
- The CP has been engineered at the hardware configuration, entered into the hardware configuration and linked-up to the Ethernet subnet.
- The CP as bus participant has an IP address.

Note!

All stations outside of the recent project must be configured as replacement objects like e.g. Siemens "SIMATIC S5" or "other station" or with the object "In unknown project".

When creating a connection you may also choose the partner type "unspecified" and set the required remote parameter directly in the connection dialog.

Work environment of NetPro

For the project engineering of connections, a thorough knowledge with NetPro from Siemens is required! The following passage only describes the basic usage of NetPro. More detailed information about NetPro is to be found in the according online manual res. documentation.

Start NetPro by clicking on a "net" in the Siemens SIMATIC Manager or on "connections" within the CPU.

The environment of NetPro has the following structure:

1 *Graphic net view*

All stations and networks are displayed in a graphic view. By clicking on the according component you may access and alter the concerning properties.

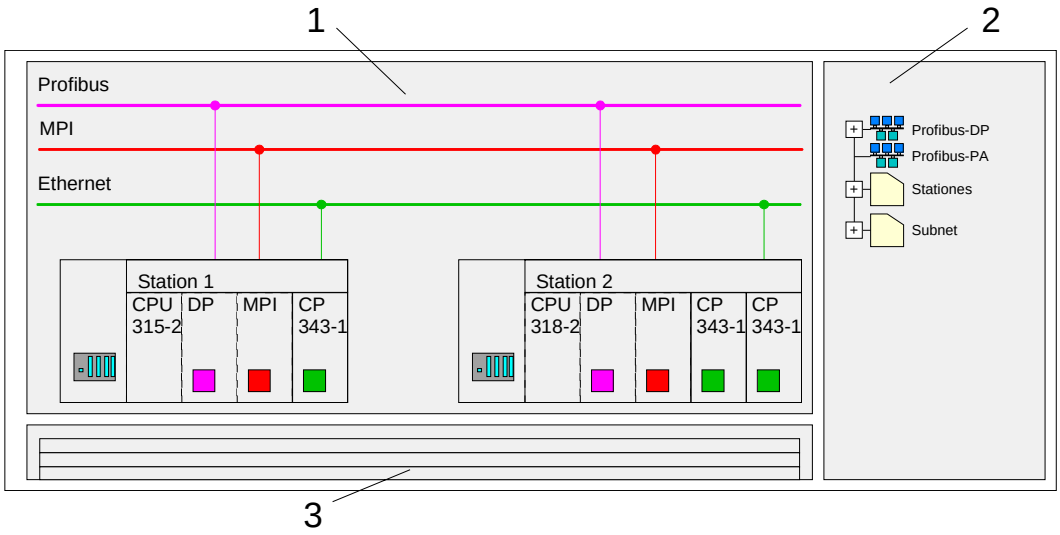
2 *Net objects*

This area displays all available net objects in a directory view. By dragging a wanted object to the net view you may include further net objects and open them in the hardware configurator.

3 *Connection table*

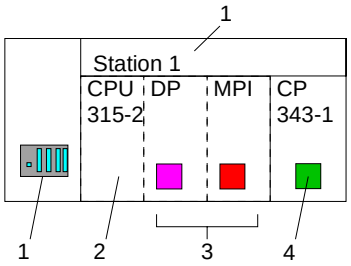
The connection table lists all connections in a table. This list is only shown when you highlighted a connectable module like e.g. a CPU.

You may insert new connections into this table with the according command.



PLC stations

You receive the following graphical display for every PLC station and their component. By selecting the single components, the context menu offers you several functions:



1 *Station*

This includes a PLC station with rack, CPU and communication components. Via the context menu you may configure a station added from the *net objects* and its concerning components in the hardware configurator. After returning to NetPro, the new configured components are shown.

2 *CPU*

A click onto the CPU shows the connection table. The connection table shows all connections that are configured for the CPU.

3 *Internal communication components*

This displays the communication components that are available in your CPU. For the 21xNET-CPUs are configured as CPU 315-2DP the internal components do not show the CP.

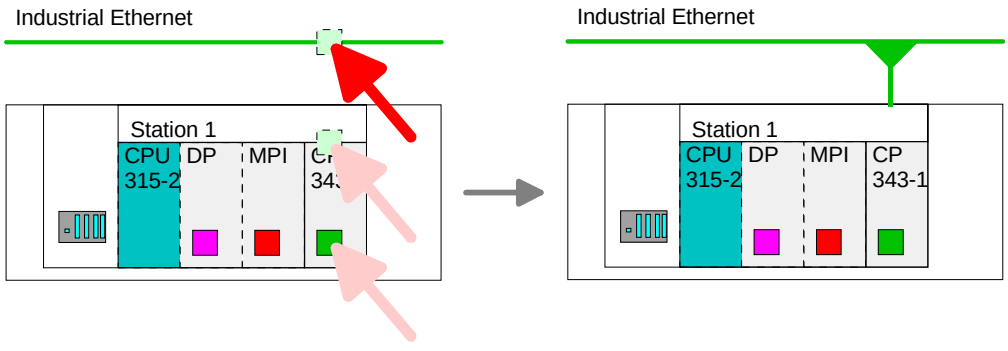
Due to this, the CP that is included in the 21xNET-CPU must be configured as external CP at slot 4. The CP is then also shown in NetPro as external CP in the station.

4 *CP*

The CP must always be configured as Siemens CP 343-1 in the hardware configuration.

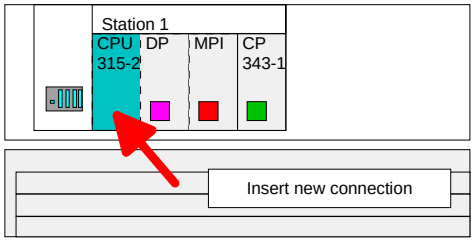
Link up stations

NetPro offers you the option to link-up the communicating stations. You may link-up the stations via the properties in the hardware configuration or graphically via NetPro. For this you point the mouse on the colored net mark of the according CP and drag and drop it to the net you want to link. Now the CP is linked up to the wanted net by means of a line.



Projecting connections

For the project engineering of connections, open the connection list by selecting the according CPU. Choose *Insert new connection* in the context menu:



A dialog window opens where you may choose the connection partner and the type of the connection.

Highlight the partner station to which you would like to establish a connection.

Choose at "Type" the connection type to be used. The following connections are supported by the CP at this time:

ISO-ON-TCP (SEND-RECEIVE, FETCH-WRITE PASSIVE)

TCP (SEND-RECEIVE, FETCH-WRITE PASSIVE)

UDP (SEND-RECEIVE)

General information

If activated, a properties dialog for the according connection opens. This dialog window is the link to your PLC program. Here you may adjust the *Local ID* and evaluate the *LADDR*.

ID
LADDR

Both are parameters that must be given to your PLC application when using the FC 5 and 6 (AG_SEND, AG_RECEIVE). Please do always use the VIPA FCs that are delivered with the SW830 as a library.

Note!

Please regard that a CP depending ID is assigned to the connections of the SEND/RECEIVE interface. This may cause alterations of the ID at changes of the project. In this case you also have to adjust the interface supply of AG_SEND res. AG_RECV in the user application.

If a CP is exchanged by another one, this must at least provide the same services and must at least have the same version level. Only this can guarantee the connections configured via the CP to remain consistent and useable.

Route

The *route* allows you to access the concerning CPs which should be used for the connection.

Addresses

The register addresses shows the relevant local and partner address information as suggestion values. Depending on the communication type you may leave the address information unspecified.

The following table shows the combination options with the different operating modes:

Connection partner	Connection type	Conn. Establ.	Connection	Operating mode
Specified in NetPro (in recent project)	TCP / ISO-on-TCP	active/passive	specified	SEND/RECEIVE
	UDP	-		
Unspecified in NetPro (in recent project)	TCP / ISO-on-TCP	active	specified	SEND/RECEIVE
		passive	part specified (Port)	SEND/RECEIVE FETCH PASSIV WRITE PASSIV
			unspecified	
	UDP	-	specified	SEND/RECEIVE
Unspecified in NetPro (in "unknown project")	TCP / ISO-on-TCP	active	unspecified (connection name)	SEND/RECEIVE
		passive	unspecified (connection name)	SEND/RECEIVE FETCH PASSIV WRITE PASSIV
	UDP	-	unspecified (connection name)	SEND/RECEIVE
All Broadcast stations	UDP	-	specified (Port, Broadcast addr)	SEND
All Multicast stations	UDP	-	specified (Port, Multicast group)	SEND/RECEIVE

Address parameter

A connection is specified by the *local* and *partner* connection end point. At the project engineering of connections ports/TSAPs must be congruent crosswise. Depending on the protocol the following parameters define a connection end point:

IP address Station A			IP address Station B	
partner TSAP	→	*ISO-on-TCP*	→	local TSAP
local TSAP	←	*connection*	←	partner TSAP
IP address Station A			IP address Station B	
partner Port	→	*TCP*	→	local Port
local Port	←	*connection*	←	partner Port
IP address Station A			IP address Station B	
partner Port	→	*UDP*	→	local Port
local Port	←	*connection*	←	partner Port

TSAP

ISO-on-TCP supports TSAP lengths (**T**ransport **S**ervice **A**ccess **P**oint) of 1...16Byte. You may enter the TSAP in ASCII or hexadecimal format. The calculation of the length happens automatically.

Port

Ports res. port addresses are defining the access point to the user application within the station/CPU. These must be unambiguous. A port address should be within the range of 2000...65535. Foreign and local ports may only be identical with one connection.

Save and compile connections

After you configured all connections this way, you may save and compile your project and exit NetPro.

To store the CP project engineering data in the system data, you have to activate the option "Store project data in the CPU" (default setting) at *object properties* area *Options* in the hardware configuration of the CP.

Broadcast-/ Multicast-connections

The expression "connection" is also used at UDP although there is no explicit connection establishment between the communication partners during runtime of the stations.

But during the project engineering like at with e.g. TCP the communication partners are assigned to each other and therefore logical linked-up.

Only at UDP the following options are additionally available at the selection of the connection partner:

- All broadcast stations
- All multicast stations

Broadcast stations

By selecting *All broadcast stations* as connection partner, you define that UDP telegrams are to be send to all available broadcast participants. Please regard that the CP may exclusively receive broadcast telegrams. The reception of user data via broadcast is not possible. Per default, broadcasts that are only serving the Ethernet communication, like e.g. ARP-Requests (Search MAC <> IP address), are received and accordingly processed.

For the identification of the broadcast participants within the net, you have to define a valid broadcast address as partner IP during project engineering of a broadcast connection. Additionally to the broadcast address you have to set a common port for sender and receiver.

Multicast stations

By selecting *All Multicast stations* you define that UDP telegrams have to be send res. received by all participants of a multicast group. In opposite to broadcast here a reception is possible.

For the identification of the multicast participants within the net, you have to define a valid multicast group address as partner IP during project engineering of a multicast connection. Additionally to this address you have to set a common port for sender and receiver.

SEND/RECEIVE with PLC program

Overview

For the execution of connection commands at the PLC, your CPU requires an user application. For this, exclusively the VIPA handling blocks AG_SEND (FC5) and AG_RECV (FC6) are used. By including these blocks into the cycle block OB1 you may send and receive data cyclic.

The two FCs are part of the VIPA library, that is included in the consignment as CD (SW830).

Note!

Please regard that you may only use the SEND/RECV-FCs from VIPA in your user application for the communication with VIPA-CPs. At a change to VIPA-CPs in an already existing project, the present AG_SEND/ AG_LSEND res. AG_RECV/AG_LRECV may be replaced by AG_SEND res. AG_RECV from VIPA without adjustment. Due to the fact that the CP automatically adjusts itself to the length of the data to transfer, the L and F variants of SEND res. RECV are not required for VIPA CPs.

Communication blocks

For the communication between CPU and CP, the following FCs are available:

AG_SEND (FC5)

This block transfers the user data from the data area given in *SEND* to the CP specified via *ID* and *LADDR*. As data area you may set a PA, bit memory or data block area. When the data area has been transferred without errors, "order ready without error" is returned.

AG_RECV (FC6)

The block transfers the user data from the CP into a data area defined via *RECV*. As data area you may set a PA, bit memory or data block area. When the data area has been transferred without errors, "order ready without error" is returned.

Status displays

The CP processes send and receive commands independently from the CPU cycle and needs for this transfer time. The interface with the FC blocks to the user application is here synchronized by means of acknowledgements/receipts.

For status evaluation the communication blocks return parameters that may be evaluated directly in the user application.

These status displays are updated at every block call.

Deployment at high communication load

Do not use cyclic calls of the communication blocks in OB1. This causes a permanent communication between CPU and CP. Program instead the communication blocks within a time OB where the cycle time is higher res. event controlled.

FC call is faster than CP transfer time

If a block is called a second time in the user application before the data of the last time is already completely send res. received, the FC block interface reacts like this:

AG_SEND

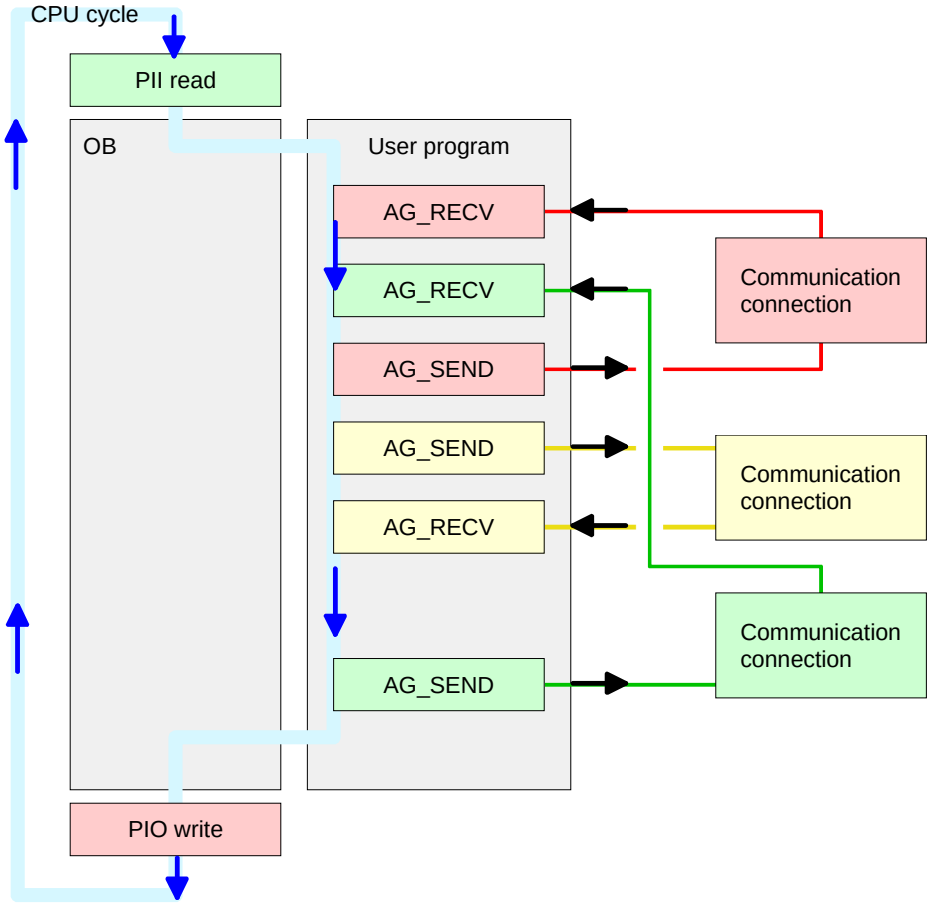
No command is accepted until the data transfer has been acknowledged from the partner via the connection. Until this you receive the message "Order running" before the CP is able to receive a new command for this connection.

AG_RECV

The order is acknowledged with the message "No data available yet" as long as the CP has not received the receive data completely.

AG_SEND, AG_RECV in the user application

The following illustration shows a possible sequence for the FC blocks together with the organizations and program blocks in the CPU cycle:



The FC blocks with concerning communication connection are summed up by color. Here you may also see that your user application may consist of any number of blocks. This allows you to send or receive data (with AG_SEND res. AG_RECV) event or program driven at any wanted point within the CPU cycle.

You may also call the blocks for **one** communication connection several times within one cycle.

AG_SEND (FC5) By means of AG_SEND the data to send are transferred to the CP.

Parameter

Parameter	Declaration	Type	Description
ACT	Input	BOOL	Activation of the sender 0: Updates DONE, ERROR and STATUS 1: The data area defined in SEND with the length LEN is send
ID	Input	INT	Connection number 1 ... 16 (identical with ID of NetPro)
LADDR	Input	WORD	Logical basic address of the CP (identical with LADDR of NetPro)
SEND	Input	ANY	Data area
LEN	Input	INT	Number of bytes from data area to transfer
DONE	Output	BOOL	Status parameter for the order 0: Order running 1: Order ready without error
ERROR	Output	BOOL	Error message 0: Order running (at DONE = 0) 0: Order ready without error (at DONE = 1) 1: Order ready with error
STATUS	Output	WORD	Status message returned with DONE and ERROR. More details are to be found in the following table.

AG_RECV (FC6) By means of AG_RECV the data received from the CP are transferred to the CPU.

Parameter

Parameter	Declaration	Type	Description
ID	Input	INT	Connection number 1 ... 16 (identical with ID of NetPro)
LADDR	Input	WORD	Logical basic address of the CP (identical with LADDR of NetPro)
RECV	Input	ANY	Data area for the received data
NDR	Output	BOOL	Status parameter for the order 0: Order running 1: Order ready data received without error
ERROR	Output	BOOL	Error message 0: Order running (at NDR = 0) 0: Order ready without error (at NDR = 1) 1: Order ready with error
STATUS	Output	WORD	Status message returned with NDR and ERROR. More details are to be found in the following table.
LEN	Output	INT	Number of bytes that have been received

DONE, ERROR, STATUS

The following table shows all messages that can be returned by the CP after a SEND res. RECV command.

A "-" means that this message is not available for the concerning SEND res. RECV command.

DONE (SEND)	NDR (RECV)	ERROR	STATUS	Description
1	-	0	0000h	Order ready without error
-	1	0	0000h	New data received without error
0	-	0	0000h	No order present
-	0	0	8180h	No data available yet
0	0	0	8181h	Order running
0	0	1	8183h	No CP project engineering for this order
0	-	1	8184h	System error
-	0	1	8184h	System error (destination data area failure)
0	-	1	8185h	Parameter LEN exceeds source area SEND
	0	1	8185h	Destination buffer (RECV) too small
0	0	1	8186h	Parameter ID invalid (not within 1 ...16)
0	-	1	8302h	No receive resources at destination station, receive station is not able to process received data fast enough res. has no receive resources reserved.
0	-	1	8304h	The connection is not established. The send command shouldn't be send again before a delay time of >100 ms.
-	0	1	8304h	The connection is not established. The receive command shouldn't be send again after a delay time of >100 ms.
0	-	1	8311h	Destination station not available with the defined Ethernet address.
0	-	1	8312h	Ethernet error in the CP
0		1	8F22h	Source area invalid, e.g. when area in DB not present Parameter LEN < 0
-	0	1	8F23h	Source area invalid, e.g. when area in DB not present Parameter LEN < 0
0	-	1	8F24h	Range error at reading a parameter.
-	0	1	8F25h	Range error at writing a parameter.
0	-	1	8F28h	Orientation error at reading a parameter.
-	0	1	8F29h	Orientation error at writing a parameter.
-	0	1	8F30h	Parameter is within write protected 1st recent data block
-	0	1	8F31h	Parameter is within write protected 2nd recent data block
0	0	1	8F32h	Parameter contains oversized DB number.
0	0	1	8F33h	DB number error
0	0	1	8F3Ah	Area not loaded (DB)

continued...

... continue DONE, ERROR, STATUS

DONE (SEND)	NDR (RECV)	ERROR	STATUS	Description
0	-	1	8F42h	Acknowledgement delay at reading a parameter from peripheral area.
-	0	1	8F43h	Acknowledgement delay at writing a parameter from peripheral area.
0	-	1	8F44h	Address of the parameter to read locked in access track
-	0	1	8F45h	Address of the parameter to write locked in access track
0	0	1	8F7Fh	Internal error e.g. invalid ANY reference e.g. parameter LEN = 0 .
0	0	1	8090h	Module with this module start address not present or CPU in STOP.
0	0	1	8091h	Module start address not within double word grid.
0	0	1	8092h	ANY reference contains type setting unequal BYTE.
-	0	1	80A0h	Negative acknowledgement at reading the module
0	0	1	80A4h	reserved
0	0	1	80B0h	Module doesn't recognize record set.
0	0	1	80B1h	The length setting (in parameter LEN) is invalid.
0	0	1	80B2h	reserved
0	0	1	80C0h	Record set not readable.
0	0	1	80C1h	The set record set is still in process.
0	0	1	80C2h	Order accumulation.
0	0	1	80C3h	The operating sources (memory) of the CPU are temporarily occupied.
0	0	1	80C4h	Communication error (occurs temporarily; a repetition in the user application is reasonable.)
0	0	1	80D2h	Module start address is wrong.

Status parameter at reboot

At a reboot of the CP, the output parameter are set back as follows:

- DONE = 0
- NDR = 0
- ERROR = 8180h (at AG_RECV)
  ERROR = 8181h (at AG_SEND)

Project transfer

Overview

There are 3 possibilities to transfer your project into the CPU:

- Transfer via MPI
- Transfer via MMC by using a card reader
- Transfer via CP

Note!

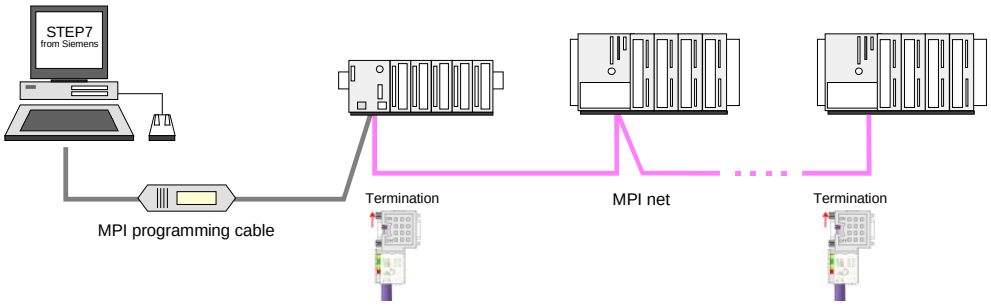
The following text contains a short description of the transfer methods. A more detailed information is to be found at "Employment CPU 21x" in the chapter "Project transfer".

Transfer via MPI

Transfer with MPI programming cable (MPI communication)

The MPI programming cables from VIPA provide a bus enabled RS485 plug for the MP^{2}I jack of the CPU and a RS232 res. USB plug for the PC. Per default your CPU has the MPI address 2.

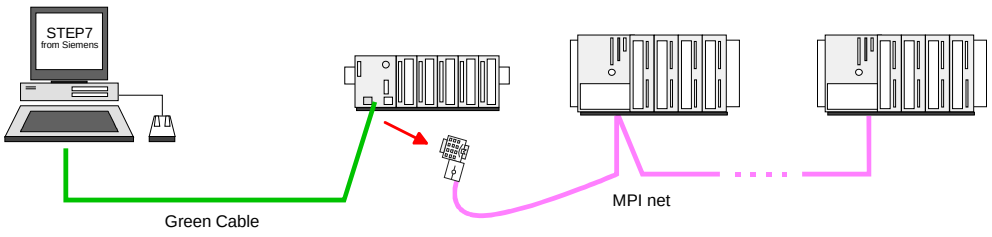
- Choose **Options** > *Set PG/PC interface* in the menu,
- adjust *MPI transfer parameter* and *address*
- Set the PC-COM port and the transfer rate 38400Baud at *Local port*.



Transfer via Green Cable (serial communication)

Via exclusively direct plugging of the Green Cable to a MP^{2}I jack you may establish a serial connection between PC and CPU.

- Set the PC-COM port and the transfer rate 38400Baud at *Local port*. The settings of the register *MPI* are ignored at employment of the Green Cable.



Transfer via MMC

The MMC (**Me**m**o**ry **C**ard) serves as external storage medium and has the FAT16 file system.

Your project engineering must be stored in the root directory and requires the file name: **S7PROG.WLD.**

With an OVERALL_RESET the MMC is automatically read (if plugged).

Transfer via Ethernet

The access via Ethernet requires a hardware project engineering (Minimum project) in the CPU where IP address and subnet mask are defined via a virtual CP.

This project is transferred per MMC or MPI into the CPU. After an overall reset you may access the CP via the PLC functions.

Approach (Minimal project)

- Configure a CPU 315-2 with the order no. 6ES7 315-2AF03-0AB0 V1.2 in the hardware configurator.
- Include the CP 343-1 (343-1EX11) at slot 4.
- Set the wanted IP address and subnet mask in the according dialog window and connect the CP with "Ethernet".
- Transfer your project to your CPU via MPI or MMC.

Access to CP

The following approach requires that the CP is available online, i.e. you assigned an IP address and subnet mask via a hardware configuration and the project engineering PC is in the same IP number circle.

Establish a connection between project PC and CPU via Ethernet using the twisted-pair slot. Adjust the following setting in the SIMATIC Manager at **Options** > *PG/PC interface*...:

TCP/IP -> Network card...Protocol RFC 1006

Now you may access the CP using e.g. the PLC functions.

CPU access via CP

- Change to your project in the hardware configurator and start the transfer with **PLC** > *Download to module*
- Select the wanted module and type as "address" the IP address of the CP. Before the transfer takes place you receive an error message that the "Online module" is different from the "Offline module". Ignore this message and start the transfer with [OK].

Now you may access the PG/OP channel via your project and configure the wanted connections for the CP 343 with NetPro.

Provided that no new hardware configuration is transferred to the CPU, the given CP is permanently stored in the project as transfer channel.

NCM diagnostic – Help for error diagnostic

Check list for error search

This page shall help you with the error diagnostic. The following page lists a number of typical problems and their probable causes:

Question	Solution with "no"
CPU in Run?	Control DC 24V voltage supply. Set RUN/STOP lever in position RUN. Check PLC program and transfer it again.
AG_SEND, AG_RECV in user application?	These 2 blocks are required in the user application for the data transfer between CP and CPU. Both blocks must also be called with a passive connection.
Is CP able to connect?	Check Ethernet cable (at a point-to-point connection a crossed Ethernet cable is to be used). Check IP address.
Can data be transferred?	Check Port no. for read and write. Check source and destination areas. Check if the right CP is selected in the route. Enlarge the receive res. send buffer defined via the ANY pointer
Is the complete data block send at ISO-on-TCP?	Check the LEN parameter at AG_SEND. Set the receive res. send buffer defined via the ANY pointer to the required size.

Siemens NCM S7 diagnostic

The CP supports the Siemens NCM diagnostic tool. The NCM diagnostic tool is part of the Siemens SIMATIC Manager. This tool delivers information about the operating state of the communication functions of the online CPs dynamically.

The following diagnostic functions are available:

- Check operating state at Ethernet
- Read the diagnostic buffer of the CP
- Diagnostic of connections

The following pages contain a short description of the NCM diagnostic. More details about the function range and for the deployment of the Siemens NCM diagnostic tool is to be found in the according online help res. the manual from Siemens.

Start NCM diagnostic

There are two options to start the diagnostic tool:

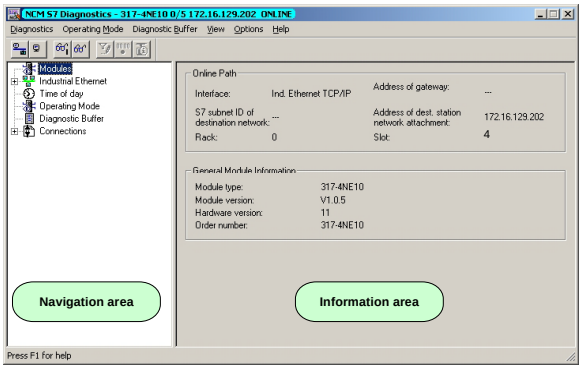
- Via *Windows-START menu > SIMATIC ... NCM S7 > Diagnostic*
- Within the project engineering res. the hardware configuration via the register "Diagnostic" in the "Property" dialog with [Execute].

Structure

The working surface of the diagnostic tool has the following structure:

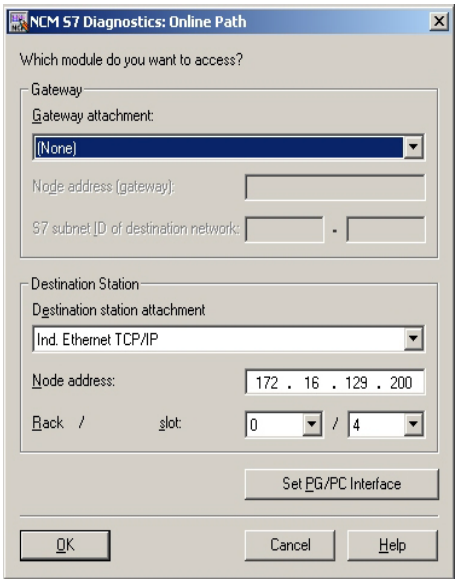
The *navigation area* at the left side contains the hierarchical listed diagnostic objects. Depending on CP type and configured connections there is an adjusted object structure in the navigation area.

The *information area* at the right side always shows the result of the navigation function you chose in the *navigation area*.



No diagnostic without connection

A diagnostic always requires a online connection to the CP you want to control. For this click on  at the symbol bar. The following dialog window appears:



Set the following parameters at *destination station*:

Connection...: Ind. Ethernet TCP/IP

Station addr.: Enter the IP address of the CP

Module rack/slot:

Always choose *rack* 0 and *slot* 4 when using System 200V.

Set your PG/PC interface to TCP/IP...RFC1006. [OK] starts the online diagnostic.

Read diagnostic buffer

The CP has a diagnostic buffer. This has the architecture of a ring memory and may store up to 100 diagnostic messages. The NCM diagnostic allows you to monitor and evaluate the CP diagnostic messages via the diagnostic object *Diagnostic buffer*.

Via a double click on a diagnostic message the NCM diagnostic shows further information.

Approach for diagnostic

You execute a diagnostic by clicking on a diagnostic object in the navigation area. More functions are available via the menu and the symbol bar.

Note!

Please always control the preconditions for an operative communication using the check at the beginning of this chapter.

For the aimed diagnostic deployment the following approach is convenient:

- Start diagnostic.
- Open the dialog for the online connection with  , enter connection parameters and establish the online connection with [OK].
- Identify the CP and check the recent state of the CP via module status.
- Check the connections for particularities like:
  - Connection status
  - Receive status
  - Send status
- Control and evaluate the diagnostic buffer of the CP via *diagnostic buffer*.
- As needed, alter project engineering res. programming and restart diagnostic.

Coupling to other systems

Outline

The operating mode FETCH/WRITE supported at TCP res. ISO-on-TCP can be used for accesses of partner devices to the PLC system memory. To be able to use this access also for example for implementation in PC applications you have to know the telegram structure for orders. The specific headers for request and acknowledgement telegrams have per default a length of 16Byte and are described at the following pages.

ORG format

The organization format is the abbreviated description of a data source or a data destination in a PLC environment. The available ORG formats are listed in the following table.

The ERW-identifier is used for the addressing of data blocks. In this case the data block number is entered into this identifier. The start address and quantity provide the address for the memory area and they are stored in HIGH-/LOW- format (Motorola-formatted addresses)

Description	Type	Range
ORG identifier	BYTE	1..x
ERW identifier	BYTE	1..255
Start address	HILOWORD	0..y
Length	HILOWORD	1..z

The following table contains a list of available ORG-formats. The "length" must not be entered as -1 (FFFFh).

ORG identifier 01h-04h

CPU area	DB	MB	EB	AB
ORG identifier	01h	02h	03h	04h
Description	Source/destination data from/into data Block in main memory.	Source/destination data from/into flag memory area	Source/destination data from/into process image of the inputs (PII).	Source/destination data from/into process image of the outputs (PIO).
ERW identifier (DBNO)	DB, from where the source data is retrieved or to where the destination data is transferred.	irrelevant	irrelevant	irrelevant
Start address significance	DBB-No., from where the data is retrieved or where the data is saved.	MB-No., from where the data is retrieved or where the data is saved.	IB-No., from where the data is retrieved or where the data is saved.	QB-No., from where the data is retrieved or where the data is saved.
Length significance	Length of the source/destination data block in words	Length of the source/destination data block in bytes	Length of the source/destination data block in bytes	Length of the source/destination data block in bytes

Note!

Information about the valid range can be found at Chapter "Hardware description of the CPU".

ORG identifier 05h-0Ah

CPU area	PB	ZB	TB
ORG identifier	05h	06h	07h
Description	source/destination data from/into peripheral modules. Input module for source data, output module for destination data.	source/destination data from/into counter cells.	Source/destination data from/into timer cells.
ERW identifier (DBNO)	irrelevant	irrelevant	irrelevant
Start address Significance	PB-No., from where the data can be retrieved or where it is saved.	ZB-No., from where the data can be retrieved or where it is saved.	TB-No., from where the data can be retrieved or where it is saved.
Length Significance	Length of the source/destination data block in bytes.	Length of the source/destination data block in words (counter cell = 1 word).	Length of the source/destination data block in words (counter cell = 1 word).

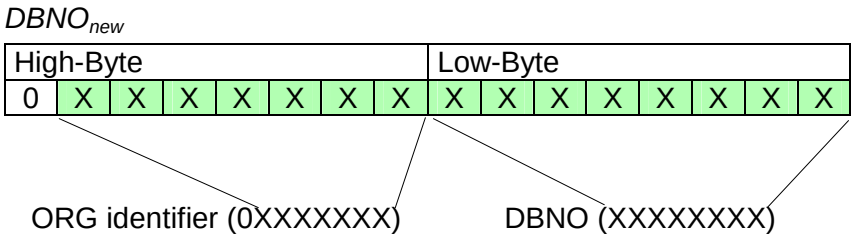
Transfer of blocks with numbers >255

ORG identifier 81h-FFh

To transfer data blocks of the number range 256 ... 32768 you may use the ORG identifier 81h-FFh.

For the setting of a DB No. >255 needs a length of one word, the $DBNO_{new}$ is assembled from the content of the ORG identifier and the DBNO.

$DBNO_{new}$ is created as word as follows:



If the highest bit of the ORG identifier is set, the Low-Byte of $DBNO_{new}$ is defined via DBNO and the High-Byte of $DBNO_{new}$ via ORG identifier, where the highest bit of the ORG identifier is eliminated.

The following formula illustrates this:

$DBNO_{new}$ =256 x (ORG-identifier AND 7Fh) + DBNO

Structure of PLC-Header

For every READ and WRITE the CP generates PLC header for request and acknowledgment messages. Normally the length of these headers is 16Bytes and have the following structure:

WRITE

Request telegram Remote Station

System ID	="S5"	(Word)
Length.Header	=10h	(Byte)
ID OP-Code	=01h	(Byte)
Length OP-Code	=03h	(Byte)
OP-Code	**=03h**	(Byte)
ORG block	=03h	(Byte)
Length ORG block	=08h	(Byte)
*ORG identifier**		*(Byte)*
ERW identifier		*(Byte)*
Start address		*(Word)*
Length		*(Word)*
Empty block	=FFh	(Byte)
Length empty block	=02h	(Byte)
Data up to 64kByte (only if error no.=0)		

Acknowledgement telegram CP

System ID	="S5"	(Word)
Length.Header	=10h	(Byte)
ID OP-Code	=01h	(Byte)
Length OP-Code	=03h	(Byte)
OP-Code	**=04h**	(Byte)
Ackn. block	=0Fh	(Byte)
Length Ack. block	=03h	(Byte)
Error no.		(Byte)
Empty block	=FFh	(Byte)
Length empty block	=07h	(Byte)
5 empty bytes attached		

FETCH

Request telegram Remote Station

System ID	="S5"	(Word)
Length.Header	=10h	(Byte)
ID OP-Code	=01h	(Byte)
Length OP-Code	=03h	(Byte)
OP-Code	**=05h**	(Byte)
ORG block	=03h	(Byte)
Length ORG block	=08h	(Byte)
*ORG identifier**		*(Byte)*
ERW identifier		*(Byte)*
Start address		*(Word)*
Length		*(Word)*
Empty block	=FFh	(Byte)
Length empty block	=02h	(Byte)

Acknowledgement telegram CP

System ID	="S5"	(Word)
Length.Header	=10h	(Byte)
ID OP-Code	=01h	(Byte)
Length OP-Code	=03h	(Byte)
OP-Code	**=06h**	(Byte)
Ackn. block	=0Fh	(Byte)
Length Ackn. block	=03h	(Byte)
Error no.		(Byte)
Empty block	=FFh	(Byte)
Length empty block	=07h	(Byte)
5 empty bytes attached		
Data up to 64kByte (only if error no.=0)		

*) More details to the data area is to be found at "ORG-Format" above.

Note!

Please regard that in opposite to Siemens-S5 systems, the block addressing of these CPUs takes the start address as byte number and the length as number of words.

Messages of error no.

The following messages can be returned via *error no.*:

Error no.	Message
00h	No error occurred
01h	The defined area cannot be read res. written

Example communication CPU21x-2BT10

Overview

This chapter provides an introduction to use the TCP/IP bus system for the System 200V. The object of this chapter is to create a small communication system between two VIPA CPUs 21xNET that provides a simple approach to the control of the communication processes.

Preconditions

Knowledge of the VIPA CP handling blocks AG_SEND and AG_RECV is required. CP handling blocks provide the options required to utilize the communication functions in the programs of the PLCs.

The minimum technical equipment required for the example is as follows:

Hardware

- 2 CPUs 21x-2BT10 from VIPA
- 1 PC or PG with Twisted Pair Ethernet connection

Communication line

- 3 bus cables
- 1 Swich/Hub

Addresses

- 2 IP Addresses and subnet masks for 2 CPs

Software package

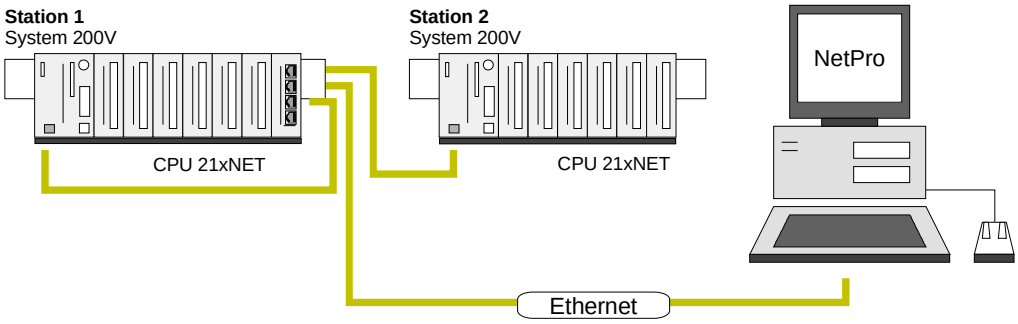
- SIMATIC Manager from Siemens V. 5.1 or higher
- SIMATIC NET

The implementation of the example requires that the two CPUs be programmed as well as the configuration of the CPs by means of NetPro from Siemens.

Note!

The complete example is to e found as zip at ftp.vipa.de/support/demofiles. You may transfer the PLC program directly to both CPUs.

Structure



Station tasks

The example for the application is based upon a communication task that is described in detail in the following passage:

Both of the CPUs contain the same PLC program, only the configuration of the CPs have to be adjusted.

Both stations are sending and receiving 16 data words per second.

- Data block DB11 transfers the data bytes DBB 0 to DBB 32 at an interval of 1s. Data byte DBB 0 in DB 11 is used as message counter. It is only incremented if the preceding transmit command was processed correctly (completed without error). The remaining data words (DBB 2 to DBB 32) can be used for the transfer of user data.
- The receiving station stores the data in DB12 (DBB 0 to DBB 31).
- Using NetPro an active SEND/RECEIVE connection with ID 1 is to be configured for the CP. This Connection is established at station 2 as a passive SEND/RECEIVE connection.
- The source and destination parameters must be configured directly.

At this point the purpose and the required settings have been outlined. The programs provide additional details of the configuration of the handler blocks. A detailed description follows.

Steps of project engineering

The project engineering is divided into the following steps:

- Hardware configuration
- CP Project engineering with NetPro
- PLC user application
- Transfer project

Hardware configuration Station 1

- Start Siemens SIMATIC Manager with new project.
- Place a new System 300 station with **Insert** > *Station* > *SIMATIC 300 station* and rename it to "Station 1".
- Activate the station "SIMATIC 300" and open the hardware configurator by clicking on "Hardware".
- Configure a rack (SIMATIC 300 \ Rack-300 \ Profile rail).
- Engineer in deputy of your CPU 21xNET the Siemens CPU 315-2DP with the order no. 6ES7 315-2AF03-0AB0 V1.2. which is to be found at SIMATIC 300 \ CPU 300 \ CPU 315-2 DP \ 6ES7 315-2AF03-0AB0. If needed, parameterize the CPU 315-2DP.
- Configure in deputy of your CP the **CP 343-1 (343-1EX11)** from Siemens at slot 4, to be found at SIMATIC 300 / CP 300 / Industrial Ethernet / CP 343-1.
- Set IP address, subnet mask and gateway at CP properties.

Here it is not necessary to configure the System 200V modules by means of a virtual Profibus system.

Hardware configuration Station 2

Create, following the approach above, a hardware configuration for the destination CPU and assign the name "Station 2".

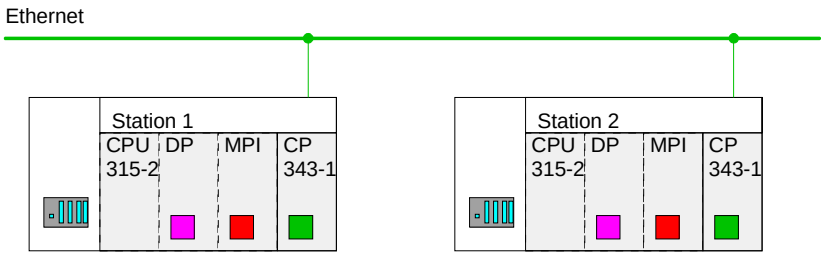
For the CP, use the IP addresses, subnet masks and gateways assigned to Station 2.

Save and compile your project.

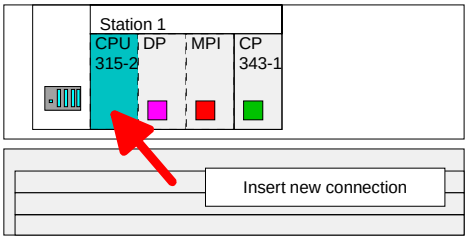
Project engineering with NetPro

Start NetPro by selecting the CPU below *Station 1* and clicking on the object "connections".

In NetPro "Station 1" and "Station 2" are listed together with Ethernet.



To configure the connection open the connection list. For this you choose the CPU of Station 1 and call *Insert new connection* via the context menu:



A dialog window appears where you can select the connection partner and the type of the connection.

Configure the following connection:

New connection

Connection: TCP connection

Connection partner: Station 2 > CPU 315-2

Properties TCP connection

ID: 1

ID and *LADDR* are parameters that you have to define in your PLC program if using FC5 (AG_SEND) and FC6 (AG_RECEIVE).

Route: The *Route* allows you to choose the CP that has to manage the connection.

For the communication between the CPU 21xNET the route "CP 343-1 - (R0/S4)" is just right preset.

Active connection establishment: activated

Save and compile your connection.

PLC user program

For the processing of connection commands at the PLC, a PLC user program is necessary in the concerning CPU. For this only the handling blocks AG_SEND (FC5) and AG_RECV (FC6) are used. By including this blocks into the cycle block OB1 with the parameters *ID* and *LADDR* you may cyclically send and receive data.

The two FCs are part of the VIPA library that is included in the consignment of the CPU as CD.

OB 1 Cycle

Via the cycle OB OB1 the sending and receiving of the data is controlled. The OB1 that you may transfer into both CPUs has the following structure:

```
      UN    T      1              // Timer 1 triggered sending
      L     S5T#1S                // Send initiation every 1 sec
      SV    T      1
      S     M     10.0            // Init bit memory
      CALL  "AG_SEND"
      ACT   :=M10.0               // Init bit memory
      ID    :=1                   // Connection number
      LADDR :=W#16#110            // Module address
      SEND  :=P#DB11.DBX0.0 BYTE 100   // Send buffer area DB11
      LEN   :=32                  // send 32 Byte (16 Words) from DB11
      DONE  :=M10.1
      ERROR :=#Senderror          // Temporary error bit memory
      STATUS:=MW12                // Order res. connection state
      U     M     10.1            // Send ready?
      SPBN  nDon
      U     M     10.1            // Send ready?
      R     M     10.0            // Set back init
      U     #Senderror            // At send error
      SPB   nDon                  // Don't raise send counter
      L     DB11.DBW    0         // Send counter in user data (DBW0)
      L     1                     // increment for 1 and
      +I                          // store again in send buffer
      T     DB11.DBW    0

nDon: NOP   0                     // Send not ready yet

      // Cyclic call of the receive block

      CALL  "AG_RECV"
      ID    :=1                   // Connection number
      LADDR :=W#16#110            // Module address
      RECV  :=P#DB12.DBX100.0 BYTE 32  //Receive buffer
      NDR   :=#Newdata            // NewDataReceived?
      ERROR :=M0.1                // RecError
      STATUS:=MW2                 // Order res. connection state
      LEN   :=#Reclen             // Really received length
      NOP   0                     // Reclen can be at IsoOnTCP < 32
      U     #Newdata              // when new data received
      ZV    Z      1              // Increment Receive counter Counter1
      L     Z      1              // reset counter 1 at overflow
      L     999
      ==I
      R     Z      1
```

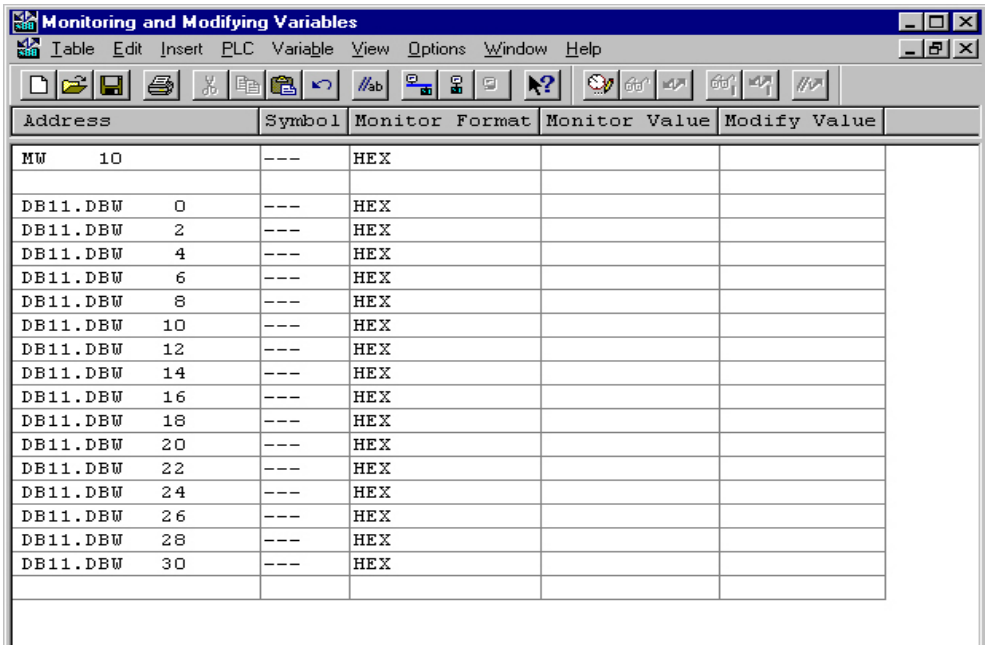
Monitoring the data transfer in the Siemens SIMATIC Manager

It is assumed, that the CPs are programmed and that an overall reset was issued to the CPUs, where the RUN/STOP switch must be located in STOP position.

Now load the above PLC programs into both CPUs and switch them into RUN.

Start the Siemens SIMATIC manager and execute the following steps to monitor the transmit job:

- **PLC** > *Monitor/Modify Variables*
- In the column "Operand" you have to enter the respective data block number and the data word (DB11.DBB 0-31).
- Establish a connection and click "monitor"  .



Entering User data

You may enter user data starting with DBB2. Place the cursor on *modify value* and enter the value you wish to transfer, e.g. W#16#1111.

The  button transfers the modify value in every cycle and the  button initiates a single transfer.

Chapter 5 Deployment CPU 21x-2BT02 with H1 / TCP/IP

Outline

The following chapter describes applications of the CPU 21x-2BT02 and the H1 resp. TCP/IP communication procedure. It also contains an introduction to the configuration of the module by means of WinNCS along with a real-world example.

The chapter contains a description of:

- the principles of the twisted-pair network
- configuration by means of WinNCS along with an example
- a test program for TCP/IP connections

Contents

Principles

Network

A network provides a link between different stations that enables them to communicate with each other.

Network stations consist of PCs, IPCs, H1/TCP/IP adapters, etc.

Network stations are separated by a minimum distance and connected by means of a network cable. The combination of network stations and the network cable represent a complete segment.

All the segments of a network form the Ethernet (physics of a network).

Twisted Pair

In the early days of networking the Triaxial- (yellow cable) or thin Ethernet cable (Cheapernet) was used as communication medium. This has been superseded by the twisted-pair network cable due to its immunity to interference. The CPU 21xNET module has a twisted-pair connector.

The twisted-pair cable consists of 4 cores that are twisted together in pairs. Due to these twists this system is provides an increased level of immunity to electrical interference.

Where the coaxial Ethernet networks are based on a bus topology the twisted-pair network is based on a point-to-point scheme.

The network that may be established by means of this cable has a star topology. Every station is connected to the star coupler (hub/switch) by means of a separate cable. The hub/switch provides the interface to the Ethernet.

Star coupler (Hub)

The hub is the central element that is required to implement a twisted-pair Ethernet network.

It is the job of the hub to regenerate and to amplify the signals in both directions. At the same time it must have the facility to detect and process segment wide collisions and to relay this information. The hub is not accessible by means of a separate network address since it is not visible to the stations on the network.

A hub has provisions to interface to Ethernet or to another hub.

Switch

A switch also is a central element for realizing Ethernet on Twisted Pair. Several stations res. hubs are connected via a switch. Afterwards they are able to communicate with each other via the switch without interfering the network. An intelligent hardware analyzes the incoming telegrams of every port of the switch and passes them collision free on to the destination stations of the switch. A switch optimizes the bandwidth in every connected segment of a network. Switches enable exclusive connections between the segments of a network changing at request.

Access control

Ethernet supports the principle of random bus accesses: every station on the network accesses the bus independently as and when required. These accesses are coordinated by a CSMA/CD (Carrier Sense Multiple Access/Collision Detection) scheme: every station “listens” on the bus cable and receives communication messages that are addressed to it.

Stations will only initiate a transmission when the line is unoccupied. In the event that two participants should start transmitting simultaneously, they will detect this and stop transmitting to restart after a random delay time has expired.

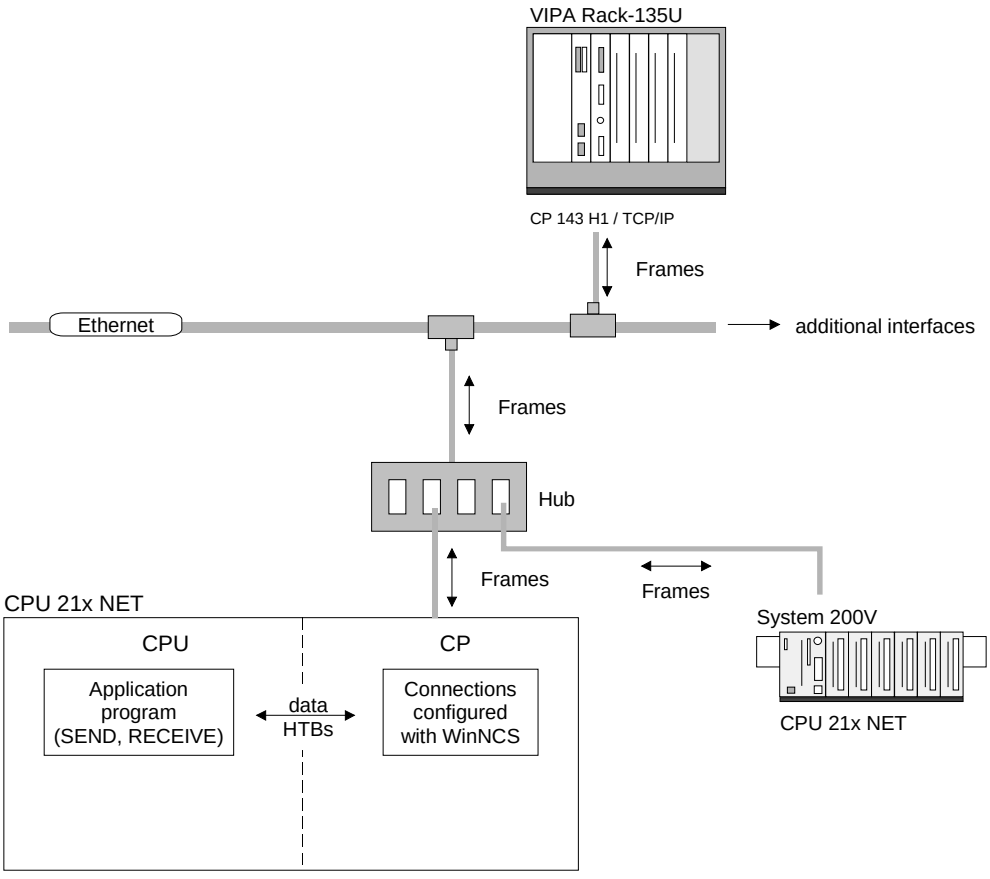
Communication

The internal CP of the CPU 21x-2BT02 is directly connected to the CPU 21xNET by means of a Dual-Port-RAM. The Dual-Port-RAM is divided into 4 equal segments called page frames.

These 4 page frames are available at the CPU as standard CP interface. Data is exchanged by means of standard handler blocks (SEND and RECEIVE).

H1 and TCP/IP communication is controlled by means of connections that are defined with the VIPA configuration tool WinNCS and are directly transferred into the CPU via the twisted-pair connection.

For details on the configuration refer to the WinNCS manual (HB91).



H1

H1 is a protocol that is based upon the Ethernet standard.

H1 information is exchanged between stations by means of H1-frames that are transferred via transport connections.

A transport connection is a logical link between two access points to the transport services on different stations. A transport connection is based upon addressing information that provides an exact description of the path between the two access points.

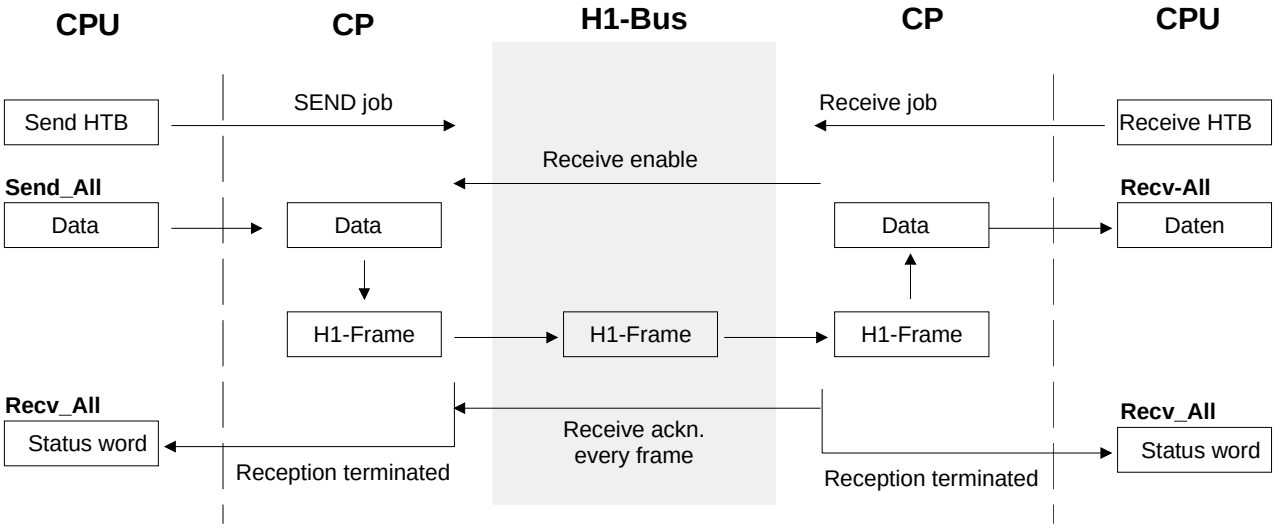
The following parameters describe a transport connection individually:

- *MAC address,* also referred to as Ethernet address, gives an unique definition of the access facility of a station.
- *TSAP Transport Service Access Points* identify access channels for the services of the transport protocol.

The CP prepares a data buffer and transfers the data into the data buffer by means of the background communication function SEND_ALL. Then the CP creates an H1-frame and transfers this to the partner station as soon as this has enabled reception. When the partner station has received the H1-frames, the CP receives an acknowledgment receipt. Next it uses the background communication function RECEIVE_ALL to transfer the status of the SEND-job into the respective indicator word.

This ensures error-free communications.

The following illustration shows the principle once again:



Note!

Due to the large quantity of acknowledgment receipts that are transferred via an H1 transport connection, the load on the network is appreciably higher under H1 than under TCP/IP, however, under TCP/IP the security of the data is reduced!

TCP/IP

TCP/IP protocols are available on all major systems. At the bottom end this applies to simple PCs, through to the typical mini computer up to mainframes (TCP/IP implementations also exist for IBM systems) and special processors like vector processors and parallel computers. For this reason TCP/IP is often used to assemble heterogeneous system pools.

TCP/IP can be employed to establish extensive open network solutions between the different business units of an enterprise.

For example, TCP/IP may be used for the following applications:

- centralized control and supervision of production plants,
- transfer of the state of production machines,
- management information,
- production statistics,
- the transfer of large quantities of data.

TCP and IP only provide support for two of the protocols required for a complete architecture. Programs like "FTP" and "Telnet" are available for the application layer of the PC.

The application layer of the CPU 21x2BT02 CP is defined by the application program using the standard handler blocks.

These application programs exchange data by means of the TCP or UDP protocols of the transportation layer. These themselves communicate with the IP protocol of the Internet layer.

IP

The main purpose of IP is to provide the addressing of data packets. This means that IP has the same function as an envelope has for a letter. The address is used by the network to determine the destination and to route the data packets accordingly.

The protocol divides the data into small portions since different networks use differently sized data packets.

A number is assigned to each packet. This is used to acknowledge reception and to reassemble the original data. To transfer these sequence numbers via the network TCP and IP is provided with a unique envelope where these numbers are recorded.

TCP

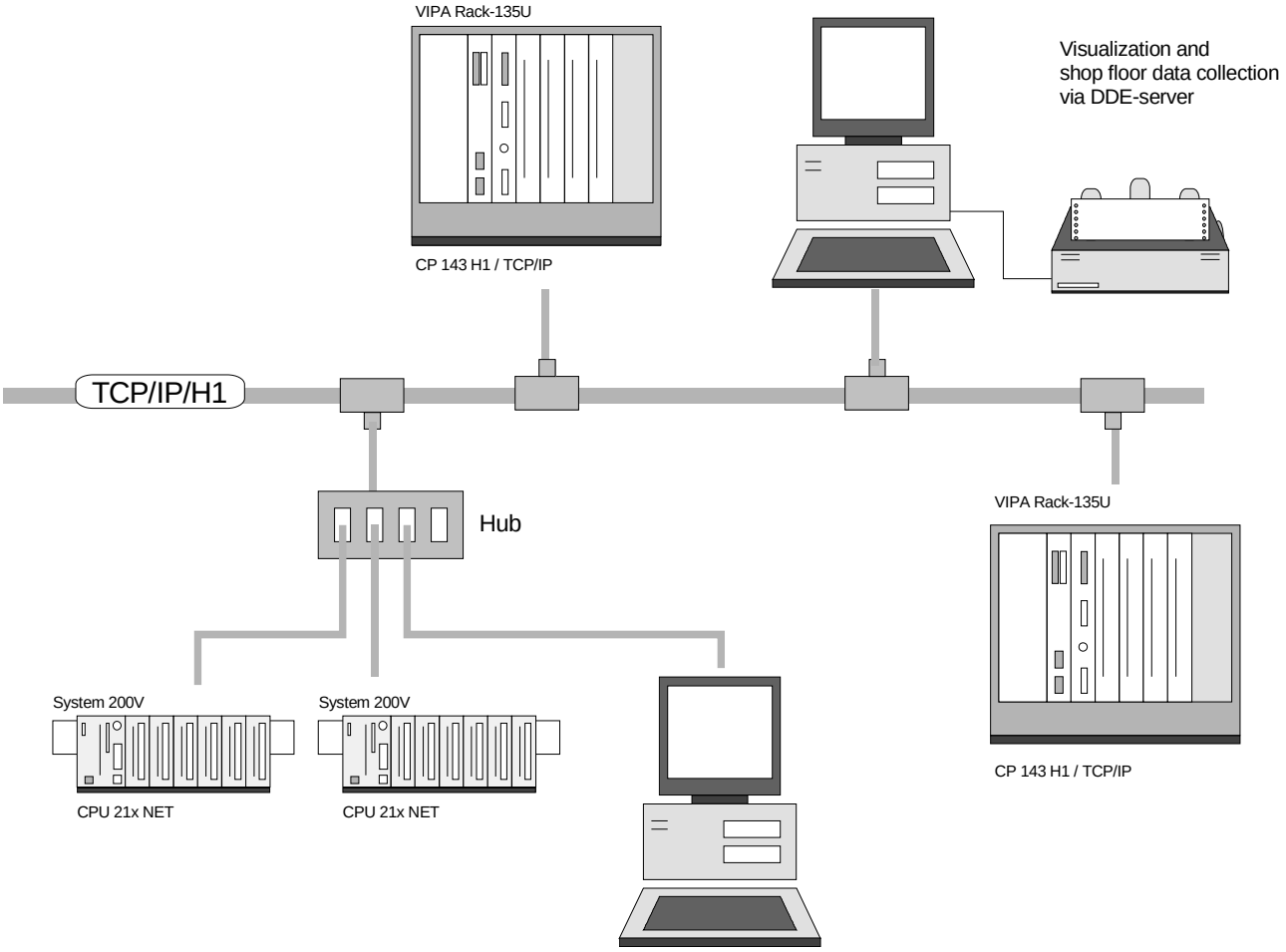
A packet of data is inserted into a TCP envelope. This is then inserted into an IP-envelope and transferred to the network. TCP provides for the secure transfer of data via network. TCP detects and corrects communication errors.

In this way TCP connections are relatively safe.

UDP provides a much faster communication link. However, it does not cater for missing data packets, nor does it check the sequence of the packets. UDP is an unsecured protocol.

Example for H1 or TCP/IP application

Deployment under TCP/IP or H1



Network planning

Standards and guidelines

The applicable rules and regulations have to be satisfied in order to establish reliable communications between the different stations. These agreements define the form of the data protocol, the method of bus access and other principles that are important for reliable communications.

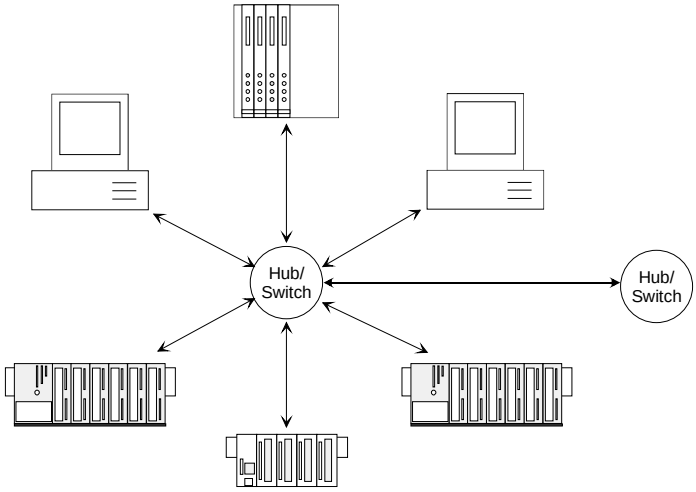
The VIPA CPU 21xNET was developed in accordance with the standards defined by ISO.

International and national committees have defined the following standards and guidelines for networking technologies:

ANSI	American National Standards Institute The ANSI X3T9.5 standard currently defines the provisions for high-speed LANs (100 MB/s) based on fiber optic technology. (FDDI) Fiber Distributed Data Interface.
CCITT	Committee Consultative Internationale de Telephone et Telegraph. Amongst others, this advisory committee has produced the provisions for the connection of industrial networks (MAP) to office networks (TOP) on Wide Area Networks (WAN).
ECMA	European Computer Manufacturers Association. Has produced various MAP and TOP standards.
EIA	Electrical Industries Association (USA) This committee has issued standard definitions like RS-232 (V.24) and RS-511.
IEC	International Electrotechnical Commission. Defines certain special standards, e.g. for the Field Bus.
ISO	International Organization for Standardization. This association of national standards organizations developed the OSI-model (ISO/TC97/SC16). It provides the framework for the standardization of data communications. ISO standards are included in different national standards like for example UL and DIN.
IEEE	Institute of Electrical and Electronic Engineers (USA). The project-group 802 determines LAN-standards for transfer rates of 1 to 1000MB/s. IEEE standards often form the basis for ISO-standards, e.g. IEEE 802.3 = ISO 8802.3.

Overview of components

The CPU 21x-2BT02 is exclusively used for employment in a Twisted-Pair network. Within a Twisted-Pair network all participating stations are connected in star topology via a Twisted-Pair cable to a hub/switch which is also able to communicate with another hub/switch. Two connected stations are building a segment where the length of the Twisted-Pair cable between two stations must be max. 100m.



Twisted pair cable

A twisted pair cable has 8 conductors twisted together in pairs. The different conductors have a diameter of 0.4 to 0.6mm.

Analyzing the requirements

- What is the size of the area that must be served by the network?
- How many network segments provide the best solution for the physical (space, interference related) conditions encountered on site?
- How many network stations (SPS, IPC, PC, transceiver, bridges if required) must be connected to the cable?
- What is the distance between the different stations on the network?
- What is the expected “growth rate” and the expected number of connections that must be catered for by the system?
- What data amount has to be handled (band width, accesses/sec.)?

Drawing a network diagram

Draw a diagram of the network. Identify every hardware item (i.e. station cable, hub, switch). Observe the applicable rules and restrictions.

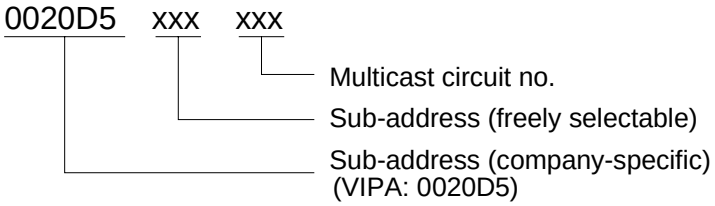
Measure the distance between all components to ensure that the maximum length is not exceeded.

Ethernet and IP addresses

Structure of an Ethernet address

A station is addressed by means of the Ethernet address, which is also known as the MAC address. The Ethernet addresses of stations in a network must be unique.

Ethernet addresses consist of the following elements:



The Ethernet address has a length of 6Byte. The first 3Byte are defined by the manufacturer. These 3Byte are assigned by the IEEE-Committee. The last 3Byte may be defined as required.

The network administrator determines the Ethernet address.

The broadcast address (to transmit messages to all stations on the network) always is:

FFFFFFFFFFFFh

IP address structure

The IP address is a 32Bit address that must be unique within the network. The IP address consists of 4 numbers that are separated by a dot.

The structure of an IP address is as follows: XXX.XXX.XXX.XXX

Range: 000.000.000.000 to 255.255.255.255

The network administrator also defines IP addresses.

The broadcast address (transmit a message to all stations) is always:

255.255.255.255

Attention!

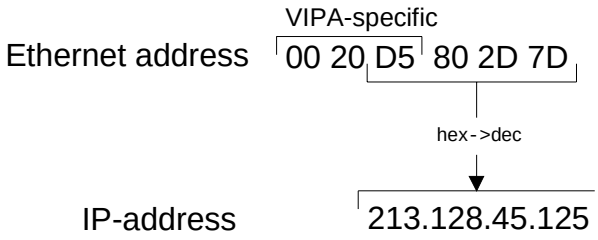
Certain IP addresses are restricted! These addresses are reserved for special services!

Initial address

When the CPU 21x-2BT02 is first turned on the module has a predefined initial Ethernet address.

This address is available from the label that has been attached to the side of the module.

This Ethernet address is only used when the module is first turned on, to calculate a unique IP address according to the following formula:



Note!

A relationship between the Ethernet address and the IP address only exists when the module is turned on for the first time.

You may always use the function CP-Init in WinNCS to assign a different address.

Attention!

The original Ethernet address may not be restored since it is not possible to perform an overall reset of the CP section.

Project Engineering of the CPU 21x-2BT02

Outline

The project engineering procedure for TCP/IP consists of 3 parts:

- **CP-configuration** by means of VIPA WinNCS (Ethernet connection).
- **Hardware configuration** (including the CP into the CPU)
- **PLC programming** by means of an application program (PLC connect).

Fast introduction

Precondition

CP is connected to Ethernet, powered an running.

CP project engineering under WinNCS

- Start WinNCS.
- By choosing  you set the functionality "Ethernet".
- Create a project with the function group "Ethernet" via **File** > *Project set-up/open*.
- Click in "Parameter"-window on [Search stations] → The available VIPA CPs are listed by their IP address.
- If your target CP is inside your IP circle, the CP can online be projected. Otherwise you have to change the IP address by using [Change IP]. After that click to [Search stations]. Please regard! If you change the IP Address of the CP by using [Change IP], the configuration inside the CP is cleared.
- Via double click on the wanted station, the according project is imported and monitored in the "network" window for further parameterization.
- Type the *station name, IP address* and *subnet mask* and click on [Apply].
  If needed, you get the IP address and the subnet mask from your system operator. The rest of the entries remain in default.
- For the project engineering of the READ/WRITE connection click at  . Type the according parameters for *connection name, order type, order model, order no., priority, IP addresses and ports* and click on [Apply].

... continue fast introduction

Transfer CP project

In the "network" window mark the station to be transfered.

- Activate the online functions via 

.
- If there is still an online connection to the CP, set the CP into software STOP via 

an start transfer with 

.

  Otherwise set "IP protocol" under 

and type the IP address 172.16.192.11 (delivery address).
- Establish a connection via 

.
- Set the CP into software STOP via 

and start the transfer into the CP using 

.

  If a request for a NCS file appears, you forgot to choose the CP in the network window! Return to the last step, choose the correct setting and start the transfer again.
- Since the data are stored unsecured in the RAM, you have to store these durable in the Flash ROM using 

. With each further flash command the date will be replaced.
- As soon as the transfer has finished, reboot the CP via 

.

Now the CP is online available with the assigned IP address.

... continue fast introduction

Hardware configuration

Precondition: SIMATIC manager from Siemens starting V. 5.1 and SIMATIC NET

- Start the SIMATIC manager from Siemens with a new project.
- Add a new System300 station via **Insert** > *Station* > *SIMATIC 300-Station*.
- Activate the station "SIMATIC 300" and open the hardware configurator by clicking on "Hardware".
- Configure a rack (Simatic300 > Rack-300 > Profile rail).
- For all CPUs 21x from VIPA are configured as CPU 315-2DP, you select the CPU 315-2DP with the order no. 6ES7 315-2AF03-0AB0 in the hardware catalog.
  This is to find under Simatic300 > CPU-300 > CPU 315-2 DP.
- If needed parameterize the CPU res. the modules. The parameter window is opened at double click on the depending module.
- Attach the System "VIPA_CPU21x" to the subnet. The respective entries are located in the hardware catalog under *PROFIBUS DP > Additional Field Devices > IO > VIPA_System_200V.* Assign Profibus address 1 to this slave.
- Place the VIPA CPU 21x-2BT02 that you want to deploy at the 1st slot of the configurator.
- Include your System 200V modules in the location sequence starting from plug-in location 1.
- Save your project.

The transfer of the hardware configuration happens together with the user application.

User application

- Create a communication channel between CPU and CP by means of the SYNCHRON block.
- Program the according SEND and RECEIVE blocks for initializing send and receive orders.
- Program the blocks SEND_ALL res. RECEIVE_ALL for data transfer.

... continue fast introduction

Transfer of user application and hardware configuration

To transfer your user application and the hardware configuration you have the following possibilities:

a) Transfer via MPI

b) Transfer via MMC

to a) Transfer via MPI

- Connect your PU res. PC via MPI with your CPU. For a serial point-to-point connection you may use the VIPA Green Cable (please regard the hints in the "Principles").
- Configure the MPI interface of your PC in the SIMATIC manager from Siemens under **Options** > *Set PG/PC Interface*. When using the Green Cable, set a transfer rate of 38400Baud.
- Via **PLC** > *Download* you transfer your project into the CPU.

to b) Transfer via MMC

Read access at the MMC always happens after an OVERALL_RESET.

To write onto the MMC you either use **PLC** > *Copy RAM to ROM* or a MMC reading device from VIPA (Order no: VIPA 950-0AD00).

Please take care that your recent project engineering is stored in the root directory and has the file name: **S7PROG.WLD.**

During the write process the yellow "MMC"-LED of the CPU blinks.

At the same time, a write access to the internal flash of the CPU takes place.

At an successful write process you'll see 0xE200 and 0xE300 in the diagnostic buffer.

When there is no valid user application on the MMC or if the transfer fails, an OVERALL-RESET of the CPU is executed and the STOP-LED blinks three times.

Here the fast introduction ends. The following pages contain a detailed description of the project engineering.

CP configuration with WinNCS

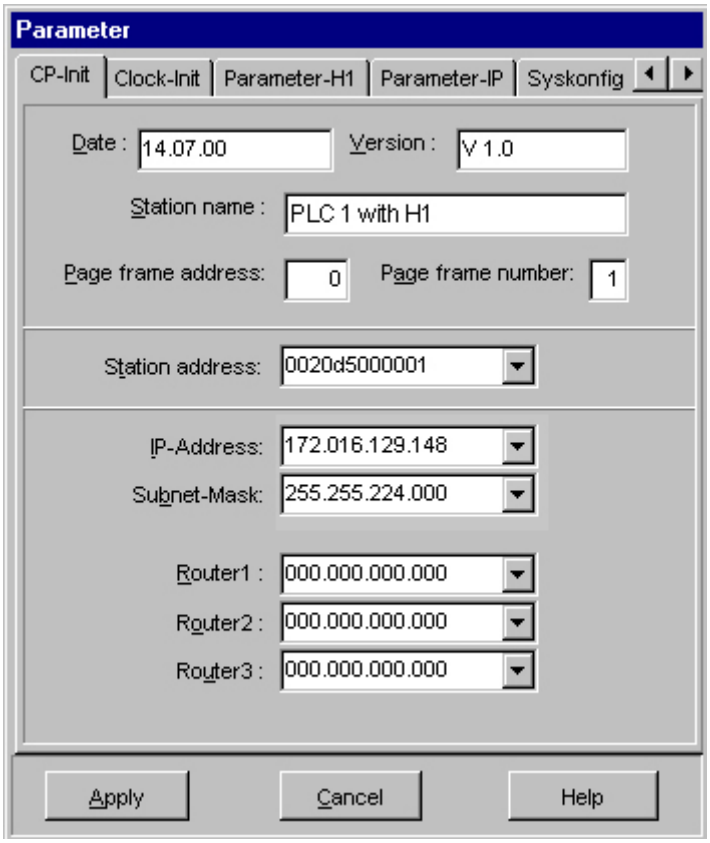
The CP section of the CPU 21x-2BT02 may only be configured with WinNCS from VIPA and consists of the following 3 parts:

- The initial CP configuration,
- configuration of connection modules,
- transfer configuration data into the CP.

Initial CP configuration

This is where the address and other identification parameters of a station are defined.

Under "Ethernet" you insert a new station into the network window and enter the configuration data for your station into the parameter window.



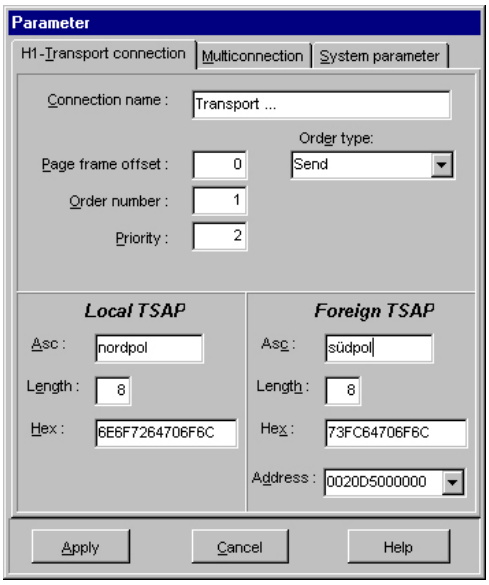
The basic CP configuration determines the behavior of your station on the network.

Configuration of a connection block

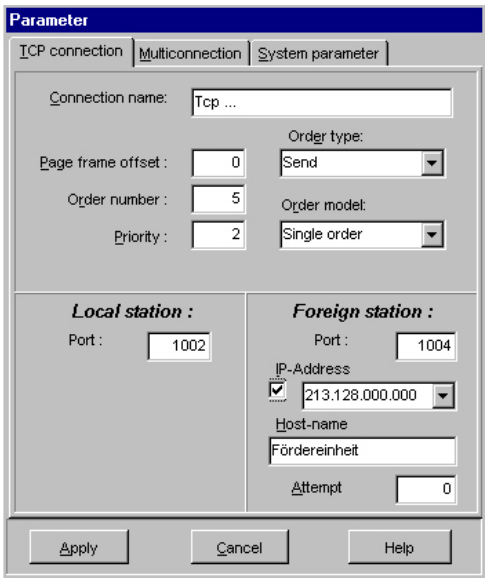
A connection block contains the remote parameters, i.e. parameters that are oriented towards the partner on the network, and local parameters, i.e. parameters that apply to the PLC program of a connection.

Depending on the selected protocol you may configure H1 or TCP/IP connections by selecting the symbol of the station and inserting/configuring the respective connection.

H1-Connection



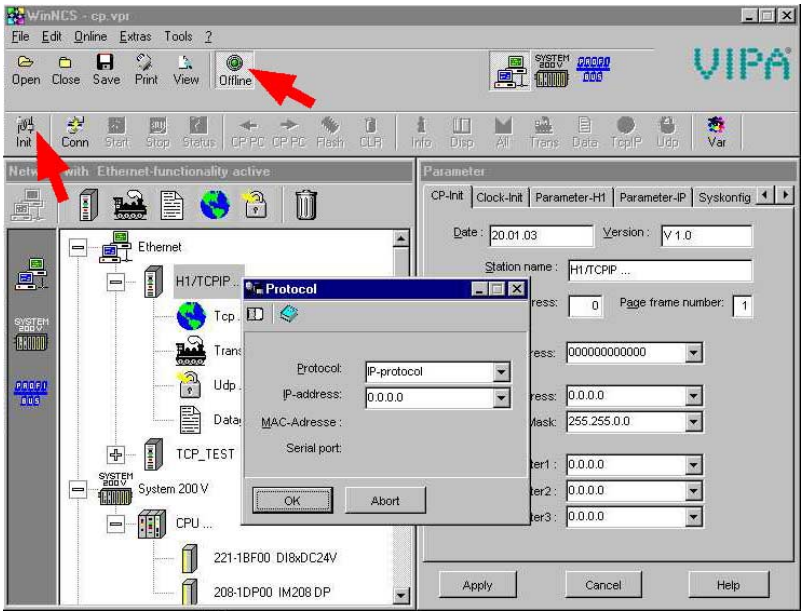
TCP/IP- Connection



Transferring the configuration to the CP

When all the required connections have been configured, you have to transfer the parameters to the CP. This operation is available from the "Module transfer functions" of WinNCS.

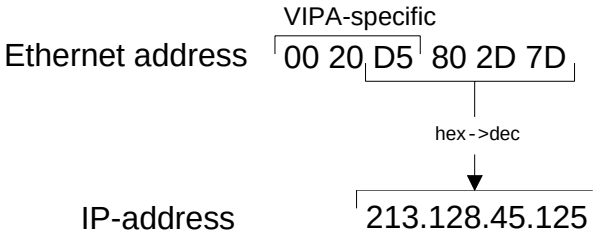
For transferring the configuration data, please activate the „online functions“ and click on the button INIT:



Like described before the CPU 21x-2BT02 is delivered with a predefined Ethernet address.

You will find this predefined address on the label at the side of the module.

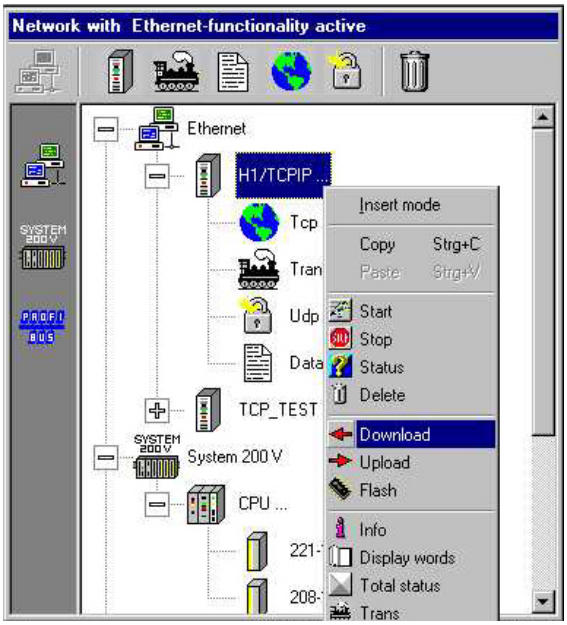
When using the module for the first time this default address is changed into an individual IP address following this rules:



Choose „IP protocol“ in the *Protocol*-window and type the determined IP address. Confirm your entry with [OK].

Now switch to the window *network* and click the according station. Use the right mouse button and choose „Download“.

Your project will now be directly transferred into the RAM of the CP.



Since the data are stored unsecured in the RAM, you have to store these durable in the Flash ROM using 

. With each further flash command the date will be replaced.

PLC application programming

To enable the PLC to process connection requests, it requires an active PLC application program on the CPU. This uses the handling blocks (SEND, RECEIVE, ...) that are included in the CPU 21x-2BT02 amongst others.

The PLC program also requires that a communication channel between the CPU and the CP ("synchronization") is specified first. This function is performed by the SYNCHRON block.

Transmission and reception is initiated by means of SEND and RECEIVE. A data transfer is initiated by means of SEND_ALL or RECEIVE_ALL.

Error messages will appear in the indicator word.

Synchronization

The used interface of the CP has to be synchronized in the start-up OB OB 100 by means of the handling block SYNCHRON.

After power is turned on, the CPU21x-2BT02 requires app. 15s for the boot procedure. If the PLC should issue a request for synchronization during this time, an error is returned in the configuration error byte PAFE. This message is removed when the CP module has completed the boot process.

The timer in this block is initially set to 20s. Processing will be stopped if the synchronization is not completed properly within this period.

Block sizes

The following table shows the available block sizes.

Block size	CP Block size in Byte
0	Default
1	16
2	32
3	64
4	128
5	256
6	512
255	512

Cycle

The sending and receiving blocks SEND and RECEIVE which initiate the send and receive operations must be configured in the cycle program OB1. The blocks SEND_ALL and RECEIVE_ALL perform the actual data-transfer.

Purely passive connections only require the components SEND_ALL or RECEIVE_ALL.

To protect the data transfer you should integrate various checkpoints that evaluate the indicator word.

Handling blocks The following table lists the required handling blocks. More detailed information is to find in the chapter "Integrated OBs, SFBs and SFCs".

SFC	Label	Description
SFC 228	RW_frame	Read/Write page frame
SFC 230	Send	Send to CP via page frame
SFC 231	Receive	Receive from CP via page frame
SFC 232	Fetch	Fetch starts the data request via page frame. FETCH is only permissible with RW identifier and delivers the order initialization for read.
SFC 233	Control	The CONTROL block is used for the status request of an order that means that the ANZW of a defined order is updated.
SFC 234	Reset	The RESET block initializes the clearing of am order of the defined connection.
SFC 235	Synchron	The SYNCHRON block serves the synchronization of the CPU and the CP during start-up. At the same time, the page frame is cleared and the block size between CPU and CP is calculated. Active data communication may only be executed between synchronized page frames.
SFC 236	Send_All	Initialization of the data transfer from the CPU to the CP.
SFC 237	Recv_All	Initialization of the data reception from the CP to the CPU.
SFC 238	Control1	Control for page frame communication with type ANZW: Pointer and parameter IND.

Transfer of user application and hardware configuration

For the transfer of your user application and the hardware configuration you have the following possibilities:

a) Transfer via MPI

b) Transfer via MMC

to a)
Transfer via MPI

- Connect your PU res. PC via MPI with your CPU. If your programming device has no MPI interface, you may establish a serial point-to-point connection with the VIPA Green Cable.

  The "Green Cable" has the order no. VIPA 950-0KB00 and may only be deployed with the VIPA CPUs with MP^2I interface. Please regard the hints in the "Principles".
- Configure the MPI interface of your PC.
- Via **PLC** > *Download* you transfer your project into the CPU.
- For the additional storage of your project on MMC, you plug-in a MMC and transfer your user application to the MMC via **PLC** > *Copy RAM to ROM*.
  During the write process, the "MC"-LED at the CPU is blinking. Due to the system, the successful execution of the write operation is announced too soon. The write process is finished completely, when the LED is off.

Hints for the configuration of an MMC interface are to find in the documentation of your programming software.

... continue to a)
Transfer via MPI

Here only the usage of the VIPA "Green Cable" together with the programming tool from Siemens shall be outlined.

The "Green Cable" establishes a serial connection between the COM interface of the PC and the MP^{2}I interface of the CPU.

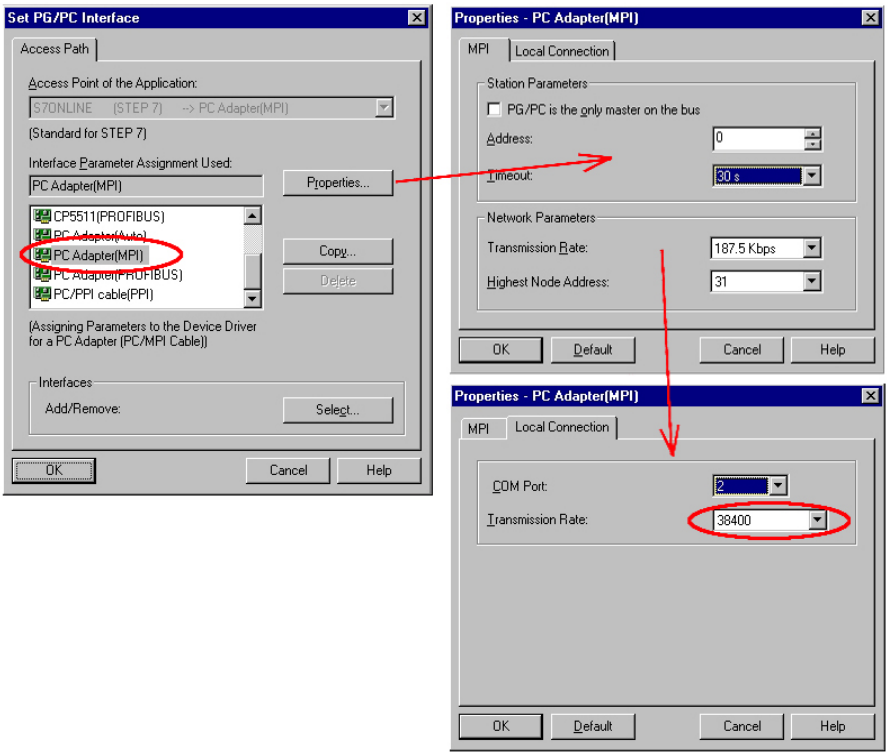
Attention!

Please regard that you may deploy the "Green Cable" exclusively with the MP^{2}I interfaces of VIPA CPUs!

Approach

- Start the SIMATIC manager from Siemens.
- Choose **Options** > *Set PG/PC Interface.*

  → The following dialog window appears where you may configure the MPI interface you want to use:



- Choose the "PC Adapter (MPI)" from the list; possibly you may have to add it first.
- Click on [Properties].

→ The following two sub dialogs you may configure your PC adapter as shown in the picture.

Note!

Please set the transfer rate to 38400 Baud when using the Green Cable.

to b)
Transfer via MMC

As external storage medium a MultiMediaCard (MMC) is used (order no. VIPA 953-0KX10).

Read access at the MMC always happens after an OVERALL_RESET.

To write onto the MMC you either use a write command of the hardware configurator from Siemens or with a MMC reading device from VIPA (order no. VIPA 950-0AD00). Thus allows to create programs at the PC and transfer them via the MMC into the VIPA CPU (copy onto MMC and plug it into CPU). The MMC modules from VIPA are preformatted with the FAT16 file system.

Required files

You may store several projects and subdirectories on a MMC.

Please make sure that the project you want to transfer is stored in the root directory and has the file name: **S7PROG.WLD.**

Transfer CPU → MMC

Plug-in the MMC and use a write command to transfer the content of the battery buffered RAM to the MMC.

Start the write command in the hardware configurator from Siemens via **PLC** > *Copy RAM to ROM*.

During the write process, the yellow "MC"-LED of the CPU is blinking.

Simultaneously, a write process into the internal Flash of the CPU is executed.

Control of the transfer process

After a write process on MMC, an according ID event is written into the diagnostic buffer of the CPU. To monitor the diagnostic entries you choose **PLC** > *Module Information* in the SIMATIC manager. Via the register "Diagnostic buffer" you reach the diagnosis window. More detailed information is to find in the chapter "Command listing".

The following events may occur when writing on MMC:

Event ID	Description
0xE100	MMC access error
0xE101	MMC error file system
0xE102	MMC error FAT
0xE200	MMC writing finished
0xE300	Internal Flash writing finished

Note!

At a successful write process you find 0xE200 and 0xE300 in the diagnostic buffer.

... continue to b)
Transfer via MMC

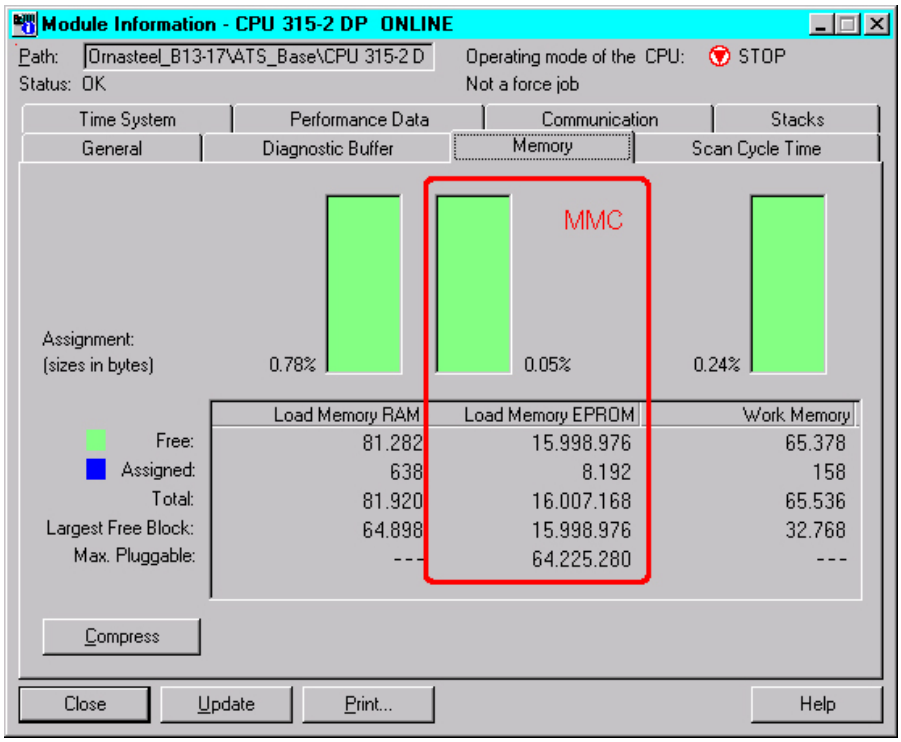
Transfer MMC → CPU

The transfer of the user application from MMC into the CPU always happens after an OVERALL_RESET. During the write process, the yellow "MMC"-LED of the CPU is blinking.

If there is no valid user application on the plugged MMC or if the transfer fails, the CPU executes an OVERALL_RESET and the STOP-LED blinks three times.

Monitor memory size of the MMC

To monitor the memory space of the MMC, you select **PLC** > *Module Information* in the SIMATIC manager from Siemens. Via the register "Memory" you reach the window that shows the current memory expansion of the CPU.



Access to the internal flash

Like shown further above, the content of the battery buffered RAM is transferred to the MMC and into the internal flash via a write command.

The write command is started in the hardware configurator from Siemens via **PLC** > *Copy RAM to ROM*.

A read access to the internal flash only takes place at empty buffer battery when no MMC is plugged-in.

Note!

If the user application is larger than the user memory in the CPU, the content of the MMC is not transferred to the CPU.

Please execute a compression of the file before you transfer it for this doesn't happen automatically.

Control project engineering

- Activate the online functions in WinNCS via 

.
- Select "IP protocol" under 

and type the new IP address.
- Establish a connection via 

. Now you communicate via the IP address given under CP-Init.
- The CP has to be in RUN. Ensure this via 

. If the CP is in *Idle*-Mode, the synchronization with the CPU has failed. Please control the SYNCHRON block in OB 100.
- For control you may monitor the summary status of the TCP connections via 

.

Now the project engineering of the CPU and the CP is finished.

Configuration example CPU 21x-2BT02

Overview

This chapter provides an introduction to use the bus system TCP/IP for the System 200V. The object of this chapter is to create a small communication system between two VIPA CPU 21x-2BT02 that provides a simple approach to the control of the communication processes.

Outline and requirements

Knowledge of the CP handler blocks is required. CP handler blocks are standard function blocks. These provide the options required to use the communication functions in the applications of the programmable logic controllers.

The minimum technical equipment required for the examples is as follows:

Hardware

- 2 CPU 21x-2BT02 from VIPA
- 1 PC or PG with twisted-pair Ethernet connection

Communication line

- 3 bus cables
- 1 Mini-Switch CM 240

Software package

- Configuration software WinNCS from VIPA
- Programming package WinPLC7 from VIPA or Siemens SIMATIC Manager for CPU 21xNET

The implementation of the example is to program the two CPUs as well as configuring the communication processors by means of WinNCS.

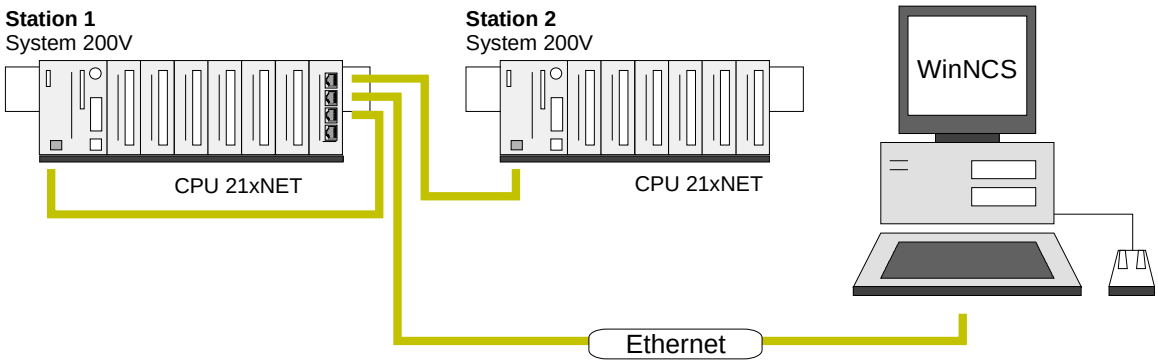
Note!

The complete example is stored at our ftp-server at:

ftp.vipa.de/support/software/demo_files.

You may transfer the PLC application directly to both CPUs.

Structure



Purpose of the PLCs

The introductory example is the application of a communication task, described below. Both of the CPUs run with the same PLC program, only the configuration of the CPs have to be adjusted.

Both stations are sending and receiving 16 data words per second.

- Data block DB11 transfers the data words DW0 to DW15 at an interval of 1s. Data word DW0 in DB11 is used as message counter. It is only incremented, if the preceding send command was processed correctly (completed without error). The remaining data words (DW1 to DW15) may be used for the transfer of user data.
- The receiving station stores the data in DB12 (DW 0 to DW 15).
- SEND is configured with job number A-No. = 1 and with a page frame offset SSNR = 0.
- RECEIVE is configured with job number A-No. = 11 and a page frame offset SSNR = 0.
- The source and destination parameters have to be configured directly.

At this point the purpose and the required settings have been outlined. The programs provide additional details of the configuration of the handler blocks. A detailed description of a suitable configuration of the CPs under control of H1 or TCP/IP is also included.

Configuration under WinNCS

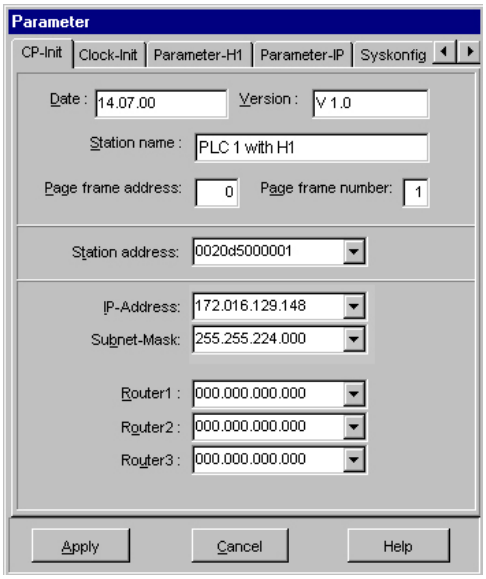
The two CPs are configured exclusively by means of WinNCS. Start WinNCS and create a project containing the function group "Ethernet_H1". The procedure is the same for both stations. It differs only in the parameters that have to be defined and is divided into the following 3 parts:

- Basic CP configuration
- Configuration of connection blocks
- Transfer of configuration data into the CP

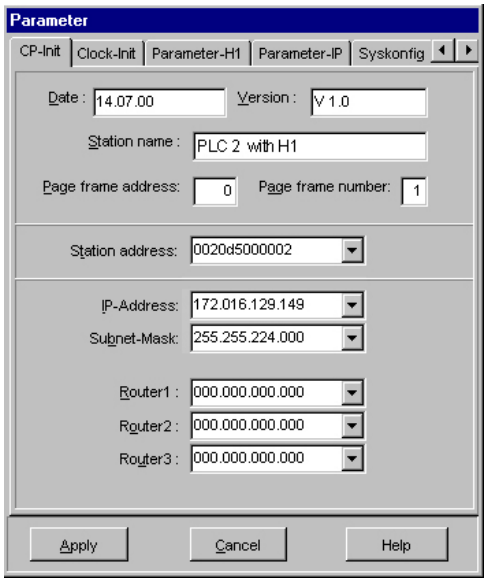
Basic CP configuration

Insert two stations and select the following settings:

Station 1



Station 2



Request the required station addresses from your system administrator.

If necessary, you may enter additional settings into the configuration windows. Details are obtainable from your system administrator.

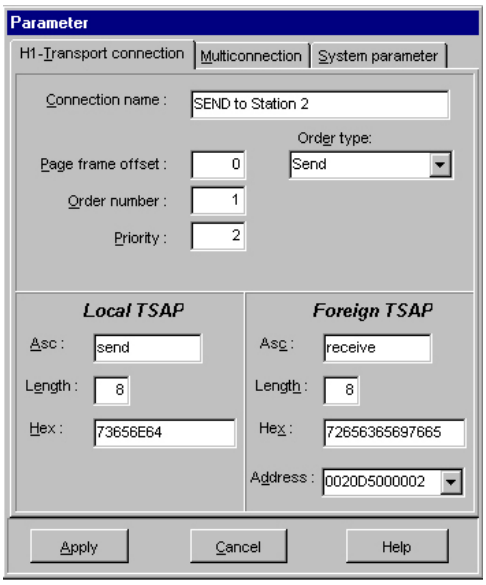
Connection configuration

H1 connections

You configure your H1 connection by inserting an H1 transport connection below the stations by means of  and entering the following parameters for the stations:

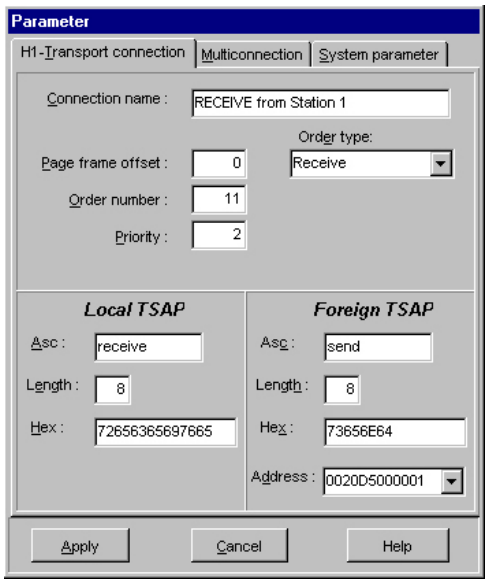
Station 1

Send to Adr.: 0020D5000002



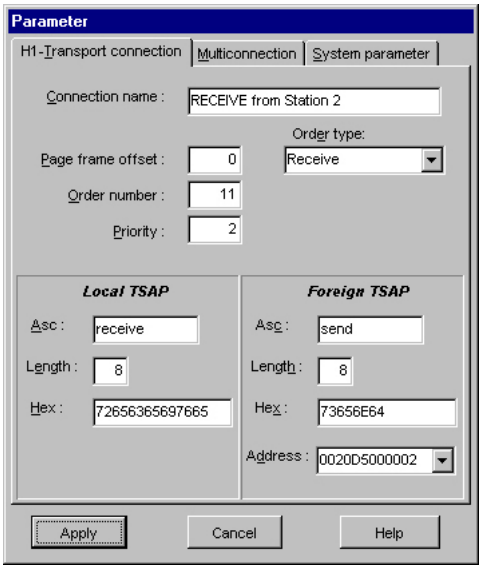
Station 2

Receive from Station 1



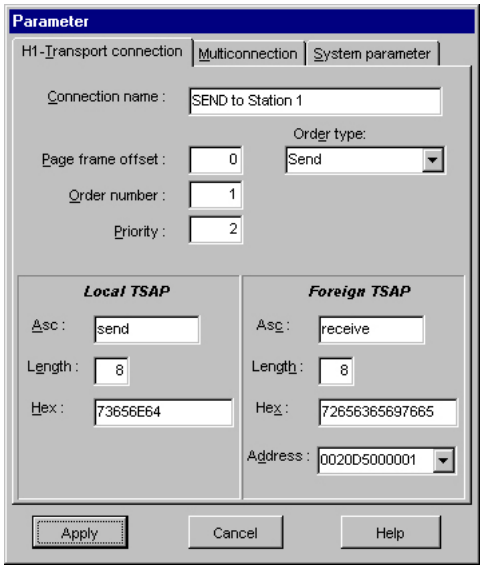
Station 1

Receive from Station 2



Station 2

Send to Adr.: 0020D5000001

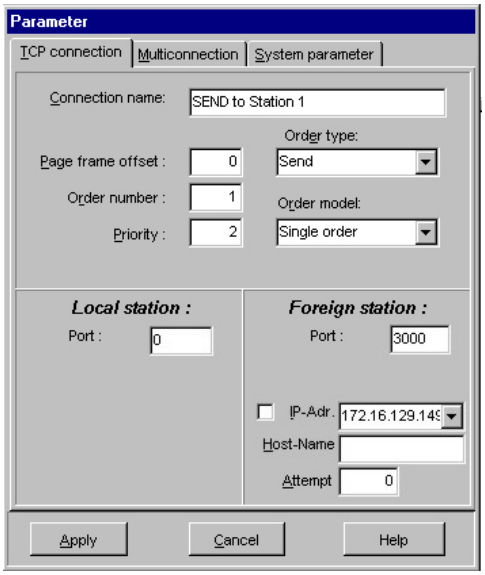


TCP/IP connections

You configure your TCP/IP connection by inserting a TCP connection below the stations by means of  and entering the following parameters for the stations:

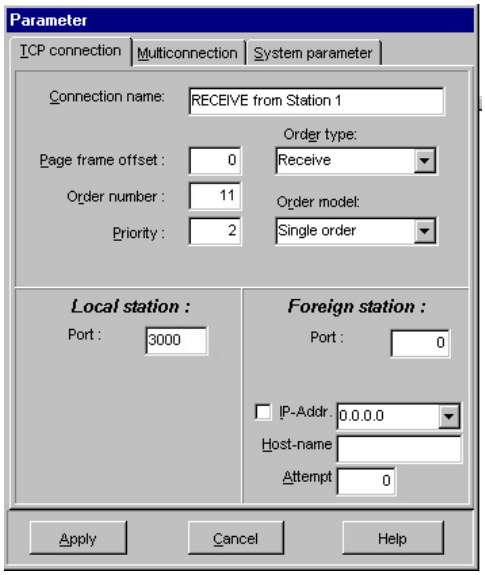
Station 1

Send to IP: 172.16.129.149

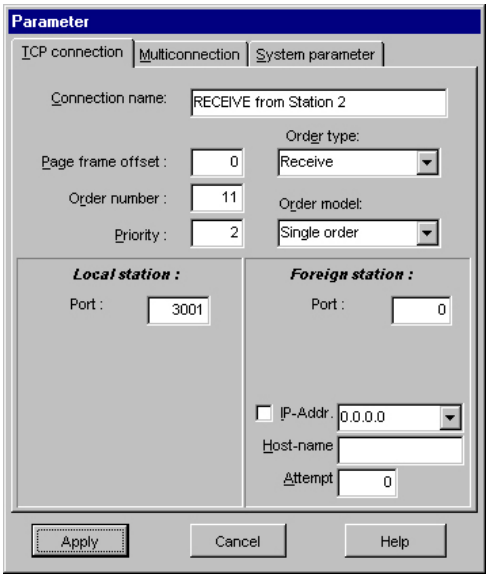


Station 2

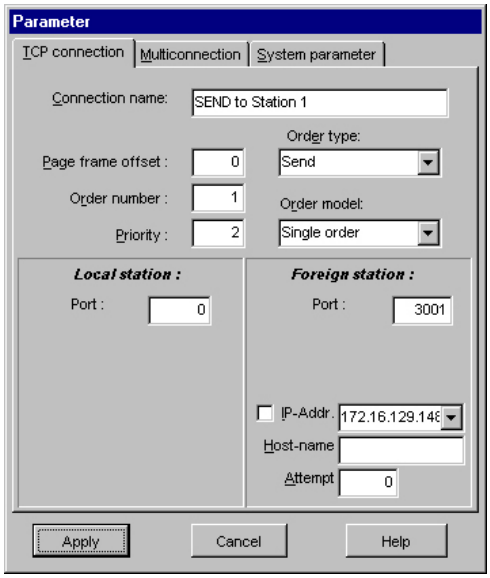
Receive from Station 1



Receive from Station 1



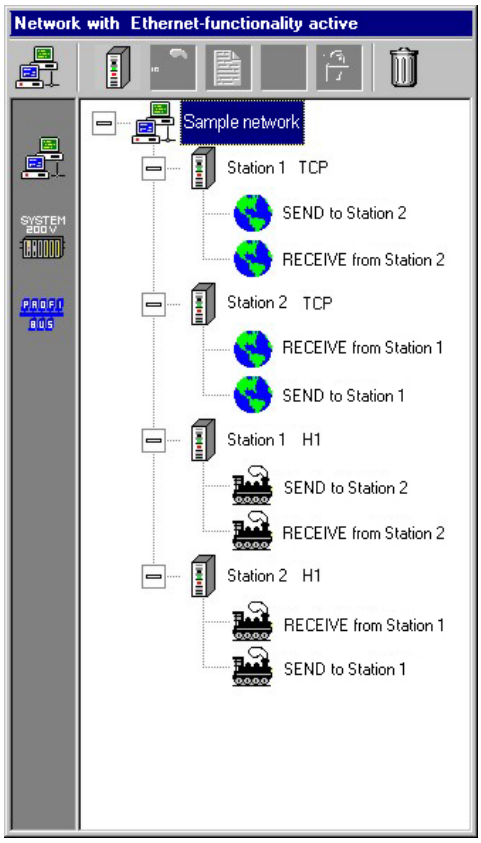
Send to IP: 172.16.129.148



Save your project!

Network window

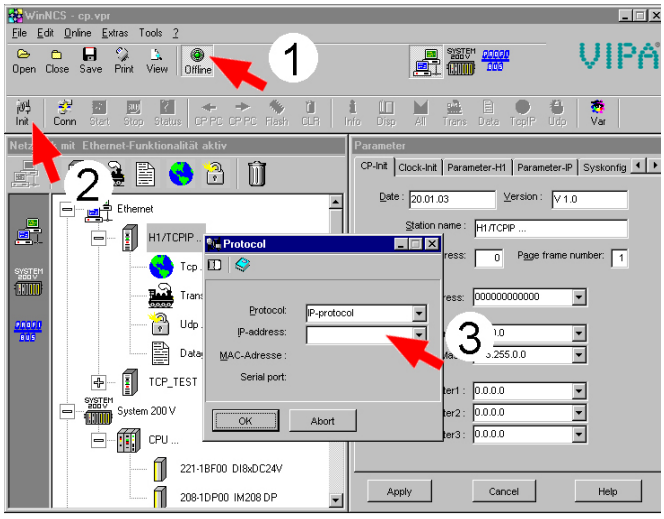
Your network window should have the following contents:



Transferring the configuration data into the CPUs

You can transfer your configuration online via the network into the respective CPUs. Create the system structure as shown above and start both CPUs.

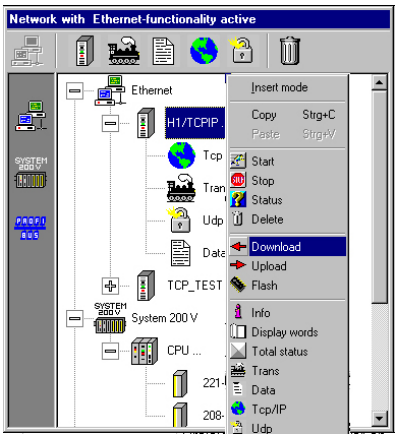
For the data transfer please activate the online functions and click on the button INIT:



Now choose „IP-protocol“ in the *Protocol* window and type the according IP address. Confirm with [OK].

Select the station to which you wish to transfer the configuration data in the *network* window of WinNCS and activate "Download".

Your project will now be transferred to the RAM of the CPU.

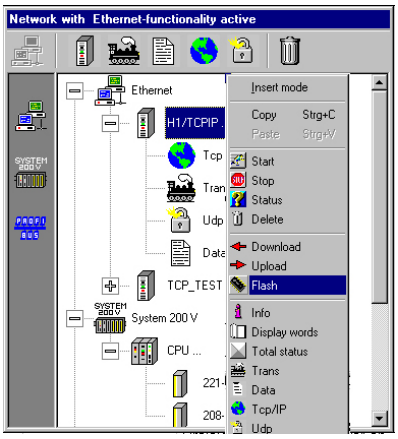


Attention!

After transferring the project into the CPU-RAM you should save the project additionally in the Flash-ROM. Otherwise the data will get lost at power off.

Therefore you are searching the according station in the *network* window. Click on the right mouse button and choose „Flash“.

The RAM data are now transferred to the Flash-ROM.



Repeat this procedure above for station 2 and don’t forget to save the project to Flash-ROM.

This concludes the configuration of the CP side. The following pages contain information on the programming for the PLC section.

PLC programs for the CPUs

The PLC programming in this example does not depend on the protocol and can therefore be used for H1 and TCP/IP.

This PLC program is used in both CPUs.

OB100 Interface Synchronization

Synchronization of the interfaces

In the start-up OB OB100 of the CPU the interface used on the CP has to be synchronized by means of the handler block SYNCHRON.

OB100 verifies that the synchronization procedure was completed without errors. If an error is detected, the error number is entered into MB200.

Operation block OB100:

```
OB100 :  "Complete Restart"
Commentary:

Network 1: Title:
Commentary:

      L     S5T#20S                  // Timer 20sec
      CLR                            // VKE 0
      SV    T      1
      SET                            // VKE 1
      SV    T      1                 // Timer Start

loop: CALL  "Synchron"               // CP synchron
       SSNR:=0                       // Interface number
       BLGR:=6                       // Block size 6 : 512 byte
       PAFE:=MB200                   // Configuration error byte

      UN    M    200.0               // no error - ready
      BEB
      U     T      1                 // Timer running
      SPB   loop                     // recall synchron
```

OB1 - Cycle
FC 1 - SEND
FC 2 - RECEIVE

The OB1 cycle controls the sending and receiving of the data. The initiation of transmission in station 1 is issued by a SEND handler block called in FC1. The partner station answers with RECEIVE (FC2). By means of SEND_ALL the data will be send and received by the partner with the command RECEIVE_ALL.

Cycle operation block OB1:

```
OB1 : Cycle
Send-All, Rec-All, Applikation

Network 1: All Function
Send- und Rec-All

      CALL  "Receive_All"              // takes data at CP
       SSNR:=0                         // is also used write passiv
       PAFE:=MB196                     // Parametrization error byte MB196
       ANZW:=MD180                     // Indication word:
//                                        showes the working order number

      CALL  "Send_All"                 // send data at CP
       SSNR:=0                         // is also used from Fetch passiv
       PAFE:=MB197                     // Parametrization error byte
       ANZW:=MD184                     // Indication word:
//                                        showes the working order number

Network 2: Title
SEND

      CALL  FC  1                      // Function call 1: sending


Network 3: Title
RECEIVE

      CALL  FC  2                      // Function call 2: receive
```

FC1 - SEND

```
FC1 : Send
Send function

Network 1 : Title
Time control sending

      CALL  "Control"                  //control send
       SSNR:=0                         //Interface number 0
       ANR :=1                         //Order number 1
       PAFE:=MB195                     //Parametrization error MB195
       ANZW:=MD174                     //Indication word MD174

      0     M    175.1                 //Job runs
      0     T      1                   //Timer runs
      BEB                              //end

      L     S5T#1S                     // Timer 1 sec
      CLR                              // vke 0
      SV    T      1
      SET                              // vke 1
      SV    T      1                   // Timer start

      CALL  "Send"                     // call send
       SSNR:=0                         // Interface number 0
       ANR :=1                         // Order number 1
       IND :=0                         // Indirect configuration
       QANF:=P#DB11.DBX 0.0 BYTE 16    // Data : db11, dw0, 16 byte
       PAFE:=MB196                     // Parametrization error MB 196
       ANZW:=MD174                     // Indicator word MD 174

      L     DB11.DBW    0              // Message counter
      +     1                          // DB11, DW0
      T     DB11.DBW    0              // increment message counter
```

FC2 - RECEIVE

```
FC2 : RECEIVE
RECEIVE funktion

Network 1 : Title
Cycle

      CALL  "Control"                  // SFC233
       SSNR:=0                         // Interface number 0
       ANR :=11                        // Order number 11
       PAFE:=MB195                     // Parametrization error MB 195
       ANZW:=MD184                     // Indicator word MD 184

      UN    M    185.0                 // if no handshake
      BEB                              // end


      CALL  "Receive"                  // SFC231
       SSNR:=0                         // Interface number 0
       ANR :=11                        // Order number 11
       IND :=0                         // Indirect configuration
       ZANF:=P#DB12.DBX0.0 BYTE 16
       PAFE:=MB195                     // Parametrization error MB 195
       ANZW:=MD184                     // Indicator word MD 184

      L     MB   188                   // Increment message counter
      +     1
      T     MB   188
```

Data blocks DB11, DB12

The frequency with which a SEND job is issued, depends on the time that was configured for the FC1 call. This timer is programmed for 1000ms in this example. For this the sample program initiates the SEND job at a rate of once every 1000ms.

Data word DW0 in data block DB11 is incremented ahead of every SEND call that actually transmits a message. This occurs in function block FC1. Altogether 16Byte of data are transferred. The partner receives the information and stores it in DB12.

Together with DB0 there is still 15Byte space for user data.

DB11 and DB12 are identical in design:

Address	Name	Type	Startvalue	Comment
0.0		STRUCT		
+0.0	STAT0	BYTE	B#16#0	
+1.0	STAT1	BYTE	B#16#0	
+2.0	STAT2	BYTE	B#16#0	
+3.0	STAT3	BYTE	B#16#0	
+4.0	STAT4	BYTE	B#16#0	
+5.0	STAT5	BYTE	B#16#0	
+6.0	STAT6	BYTE	B#16#0	
+7.0	STAT7	BYTE	B#16#0	
+8.0	STAT8	BYTE	B#16#0	
+9.0	STAT9	BYTE	B#16#0	
+10.0	STAT10	BYTE	B#16#0	
+11.0	STAT11	BYTE	B#16#0	
+12.0	STAT12	BYTE	B#16#0	
+13.0	STAT13	BYTE	B#16#0	
+14.0	STAT14	BYTE	B#16#0	
+15.0	STAT15	BYTE	B#16#0	
+16.0	STAT16	BYTE	B#16#0	
=18.0		END_STRUC		

Transfer project

The data transfer is realized via MPI. If your programming device has no MP-interface, you may also use the "Green Cable" (VIPA 950-0KB00) from VIPA.

The "Green Cable" may only be used at the VIPA CPUs of the Systems 100V, 200V, 300V and 500V!

Please regard the notes about the "Green Cable" in chapter 1!

- Connect your PG with the CPU
- With **PLC** > L*oad to module* in your configuration tool you transfer your project into the CPU.
- Insert a Multi Media Card (MMC) into the according CPU-slot and transfer your application via **PLC** > *Copy RAM to ROM*.
- During the writing process the MC-LED on the CPU is blinking. Due to the system the dialog message of the successful writing process appears a little bit too soon. This process is not ready before the LED on the CPU is extinguished.
- Switch both CPUs to RUN.

Monitoring the data transfer in Siemens SIMATIC manager

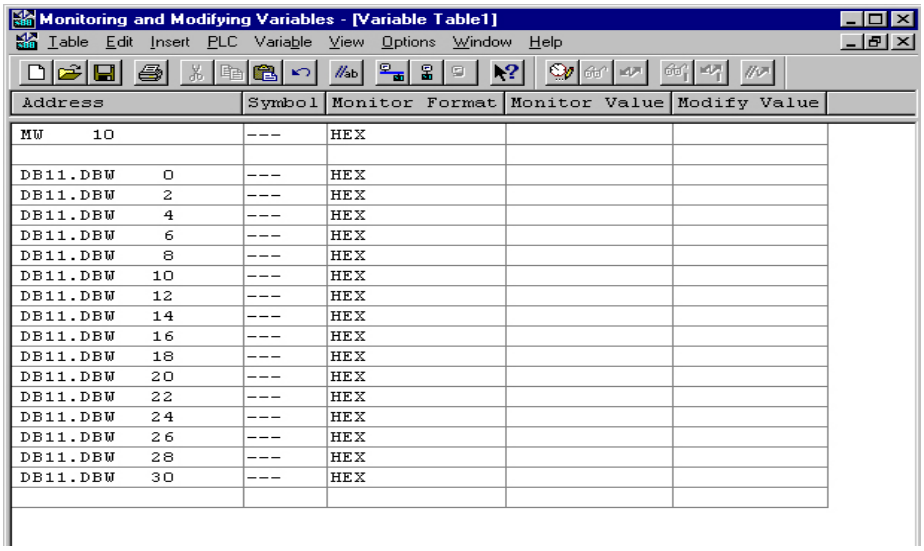
It is assumed that the CPs were programmed and that an overall reset was issued to the CPUs where the RUN/STOP switch has to be in STOP position.

You now load the above PLC programs into your CPUs. Start the programs by placing the respective RUN/STOP switch into the position RUN.

At this point, communication between the modules is established. This is indicated by the COMM-LEDs.

Start the Siemens SIMATIC manager and execute the following steps to monitor the transmitting job:

- **PLC** > *Monitor/Modify Value*
- In the "Operand" column you enter the respective data block number and the data word (DB11.DW0-15).
- Establish a connection and click "Monitor"  .



Entering user data

You may enter user data starting with DW1. Place the cursor on *Modify value* and enter the value you wish to transfer, e.g. W#16#1111.

The  button transfers the new value in every cycle and the  button initiates a single transfer.

Start-up behavior

Overview

When the power supply is turned on, the CPU and the CP execute the respective BIOS routine (hardware and driver initialization and memory test).

While the CPU detects the installed modules on the backplane bus and loads the application program, the CP starts the page frame administration routine.

After app. 15s the CP waits for the synchronization request from the CPU. In this condition data communication with the CPU is inhibited and it is only enabled after synchronization has taken place.

The boot time of the CPU 21x-2BT02 including the CP amounts to app. 18s.

Status after CP boot-up

With every status change from STOP to RUN as well as from RUN to STOP back to RUN, the CPU 21x-2BT02 performs a cold-/warm reset. All connections are cleared and reestablished after the CP has rebooted.

Three different reasons can cause these status change requests:

- Resynchronization of a CP by the SYNCHRON-HTBs of the CPU (warm start) after it has already been synchronized,
- STOP/START-function of the configuration tool WinNCS (warm start),
- RESET_ALL (warm start).

Note!

When a defective project engineering is transferred to the CP so that is not able to start, the project engineering is deleted and a default setting is used. Here the CP gets its original IP address again.

This allows you to clear the CP independently from the CPU.

System properties of the CPU 21x-2BT02

Note

System properties of a CP should not be regarded as restrictions or equated with malfunctions. Certain functions can not be provided or are not desired when the overall system is taken into account.

General

The boot time of the CP section of the CPU 21x-2BT02 is app. 18 seconds. The integrated SYNCHRON block (delay time 30s) caters for this boot time.

Note only for H1

- Jobs with a priority 0/1 may transmit and/or receive a maximum quantity of data as defined by the SYNCHRON-HTB. Jobs of this priority are not blocked. This results in a maximum data transfer rate of 512Byte per job for a block size of 255 (refer also to block size).
- RECEIVE jobs that are mapped to the communication type "broadcast" cannot receive all data messages from a fast cyclic transmitter. Messages that have not been received are discarded.

Note only for TCP/IP

- The default length (-1, 0xFFFF) is not permitted for the ORG format length, i.e. the user has to define the exact length of the data that will be received.
- Jobs with a priority of 1 may send and/or receive a maximum quantity of data as defined by the SYNCHRON-HTB. Jobs of this priority are not blocked. This results in a maximum data transfer rate of 512Byte per job for a block size of 255 (see also block size).
- RECEIVE-jobs that mapped to communication type UDP cannot receive all data messages from a fast cyclic station. Messages that have not been received are discarded.

The TCP/IP protocol stack has a global buffer pool where the receive and transmit buffers are located. This is where system collisions can occur if:

- Data for a receive job is not collected. After a period of time a lack of resources will occur and the other connections will terminate themselves.
  It is only possible to re-establish proper communications when the receive buffers of this connection have been released (connection terminated) or when the data has been retrieved by means of RECEIVE.
- one or more cyclic stations place a load on a CP. When resource bottlenecks are encountered, the CP can also initiate the termination of connections.
- A station transmits two or more messages and the receiver did not have a chance to accept them. Then the reception of the unknown data type would cause collisions in the receiver. However, the CP prevents this. The PLC application requires a defined size for the reception of data and the wildcard length is not permitted. The size of the receive block of Prio1 - RECEIVE jobs is defined implicitly by the pre-defined block size (16, 32, 64, 128, 256, 512Byte).
- VIPA recommends the use of acknowledgment messages on the user level to ensure that data transfers are one hundred percent safe.
- Please regard, that the **Port 7777** is used by WinNCS for communication. This may not be occupied by other applications!

Communication to other systems

ORG format

The organization format is the abbreviated description of a data source or a data destination in a PLC environment. The following table lists the available ORG formats.

In the case of READ and WRITE the ORG block is optional.

The ERW identifier is used for the addressing of data blocks. In this case the data block number is entered into this identifier. The start address and quantity provide the address for the memory area and they are stored in HIGH/LOW format (Motorola-formatted addresses)

Description	Type	Range
ORG identifier	BYTE	1..x
ERW identifier	BYTE	1..255
Start address	HILOWORD	0..y
Length	HILOWORD	1..z

The following table contains a list of available ORG formats:

ORG identifier 01h-04h

CPU area	DB	MB	EB	AB
ORG identifier	01h	02h	03h	04h
Description	Source/destination data from/into data block in main memory.	source/destination data from/into flag area.	Source/destination data from/into process image of the inputs (PAE).	source/destination data from/into process image of the outputs (PAA).
ERW identifier (DBNR) valid range:	DB from where the source data is retrieved or to where the destination data is transferred. 1...255	irrelevant	irrelevant	irrelevant
Start address Significance valid range:	DB-No. from where the data is retrieved or where data is saved. 0...2047	MB- No. from where the data is retrieved or where data is saved. 0...255	EB-No. from where the data is retrieved or where data is saved. 0...127	AB-No. from where the data is retrieved or where data is saved. 0...127
Length Significance valid range:	Length of the source/destination data - block in words. 1...2048	Length of the source/destination data block in words. 1...256	Length of the source/destination data block in words. 1...128	Length of the source/destination data block in words. 1...128

Structure of PLC header

For every READ and WRITE the CP generates PLC headers for request messages and for acknowledgment messages. Normally the length of these headers is 16Byte and they have the following structure:

for WRITE

Request message

System identifier	="S"
	="5"
Header length	=16d
Ident.OP code	=01
OP code length	=03
OP Code	**=03**
ORG block	=03
ORG block length	=08
ORG identifier	
ERW identifier	
Start address	H
	L
Length	H
	L
Dummy block	=FFh
Dummy block length	=02
64K data only if error no.=0	

Acknowledgment message

System identifier	="S"
	="5"
Header length	=16d
Ident.OP code	=01
OP Code length	=03
OP Code	**=04**
Ack. Block	=0Fh
Ack. Block length	=03
Error No.	=No.
Dummy block	=FFh
Dummy block length	=07
not used	

for READ

Request message

System identifier	="S"
	="5"
Header length	=16d
Ident.OP code	=01
OP Code length	=03
OP Code	**=05**
ORG block	=03
ORG block length	=08
ORG identifier	
ERW identifier	
Start address	H
	L
Length	H
	L
Dummy block	=FFh
Dummy block length	=02

Acknowledgment message

System identifier	="S"
	="5"
Header length	=16d
Ident.OP code	=01
OP Code length	=03
OP Code	**=06**
Ack. block	=0Fh
Ack. Block length	=03
Error No.	=No.
Dummy block	=FFh
Dummy block length	=07
not used	
64K data only if error no.=0	

SEND / RECEIVE of the type TRADA

TRADA stands for **Tra**nsparent **Da**ta exchange. A transparent data exchange may transfer application data with varying block lengths. A 16Byte header that defines the length of the application data precedes the application data.

With TRADA you may enter a wildcard length into the PLC application.

If you enter -1 as length into the RECEIVE-FB (parameter: ZLAE) you are defining a variable length (wildcard length) for the application data. With wildcard lengths the actual length of the data is retrieved from the respective TRADA header.

With the TRADA functionality the following header will precede a SEND job and it is analyzed by the RECEIVE function.

SEND of type TRADA
OP-code = 07

System identifier	="S"		
	="5"		
Header length	=16d		
Ident.OP code	=01		
OP code length	=03		
OP code	=**07**		
ORG block	=03		
ORG block length	=08		
ORG identifier			
ERW identifier (**irrelevant**)			
Start address	H		
	L		
Length	**H**	→	Length of the user data
	L		
Dummy block	=FFh		
Dummy block length	=02		
64K data if error no.=0			

Length

The length file contains the number of bytes in a data block.

If you are synchronizing with a block size of 6 (512Byte) the length is entered in words.

Test program for TCP/IP connections

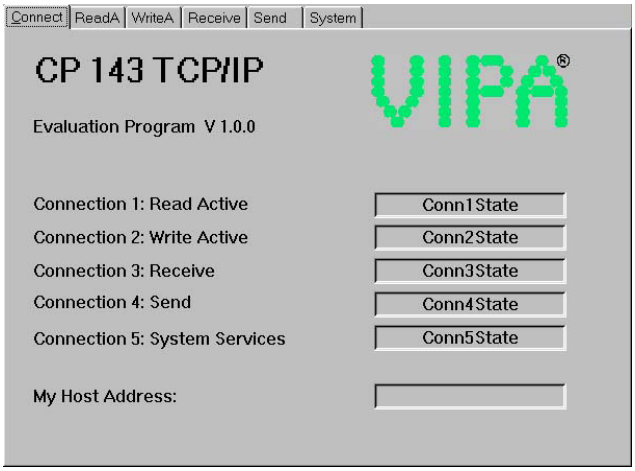
TCPTest.exe is to be found at VIPA-ftp-Server using the link ftp.vipa.de/support/software. You can use this test program to create simple TCP/IP connections and analyze them.

TCPTest requires no further installation, is executable at usual operating systems and communicates via Ethernet.

The following section provides a short introduction to the test-program.

For this purpose please start TCPTEST.EXE. The test program is executed and displays the following window:

Initial display



Tab sheets

The menu has the appearance of tab sheets. The respective dialog window can be displayed by left clicking with the mouse.

Tab sheets	
Connect	Window containing the status of the connections and the local IP-address.
ReadA	Configuration window for READ AKTIV connections (FETCH).
WriteA	Configuration window for WRITE AKTIV connections.
Receive	Configuration window for RECEIVE connections.
Send	Configuration window for SEND connections.
System	Control windows for status requests and toggling between RUN/STOP of the CP.

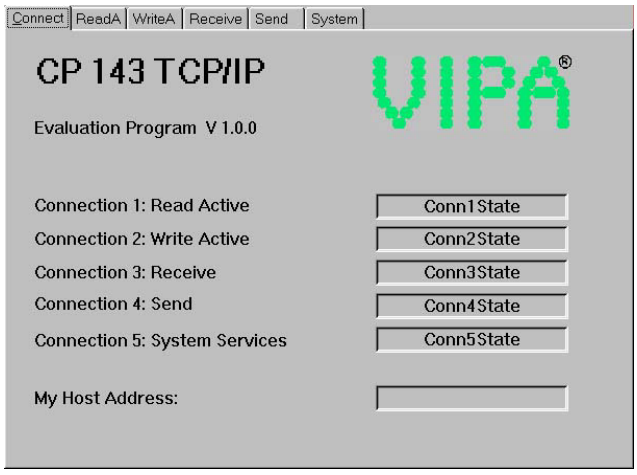
Context menu (right mouse key)

You can activate a context menu in each tab sheet. This is activated by means of the right mouse key or button.

You can always access the context menu by clicking the right mouse key. This menu offers the following selection:

Save All	Save all parameters.
Save Conn1 to *Save Conn5*	Saves the respective connection.
Save Win Pos	Saves the current window position.
Show Hints	When you place the cursor on an input field or on a button, a hint is displayed if you have selected "Show Hints".

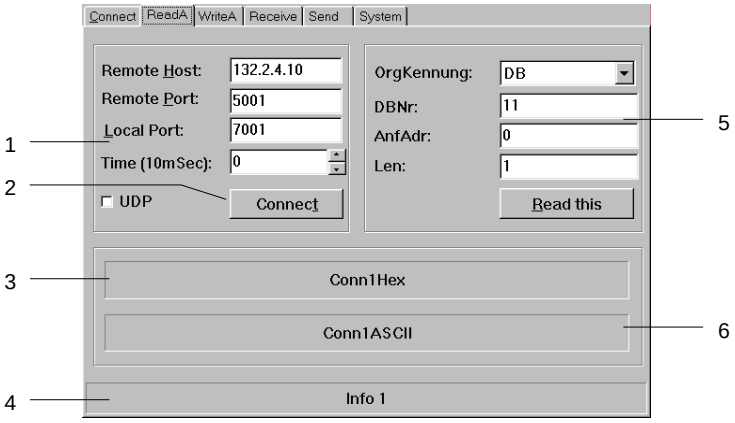
Connect tab (Status)



This window displays the status of all the connections that can be configured in this program. Here you can recognize in one screen, which connections are stable and which are unstable. When a status changes in a register the change is displayed in this window.

For reference, your own IP address is also displayed in the window.

ReadA tab



[1] port data
[2] establish a connection
[3] hexadecimal number
[4] information window for the status of the connection
[5] source data
[6] ASCII formatted display of the data received

Here you can configure an active read connection.

In addition to the data required to establish the connection you must also specify the source from where the data should be read.

Input fields

Remote Host	IP-address of the station from which you wish to read data.
Remote Port	Port address of the remote station.
Local Port	Port address of your own (local) station. To simplify matters you can specify the same port address for remote and local.
Time (10mSec)	Definable interval for cyclic read operations.
OrgKennung	Type of the source block.
DBNr	Number of the source block.
AnfAdr	Start address of the source block.
Len	Word length of the source block.

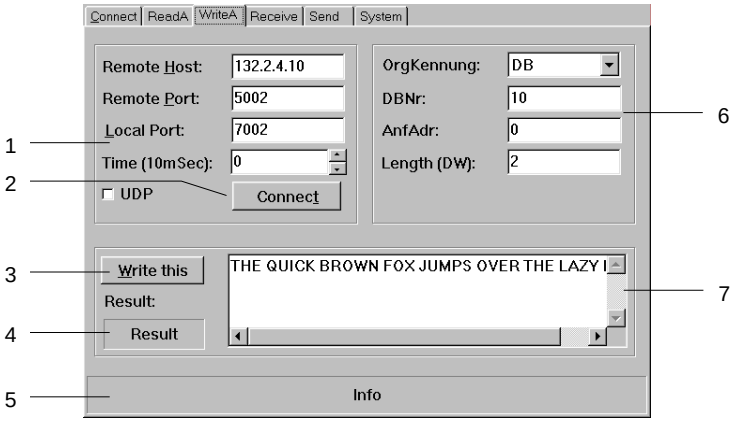
Tick-box

UDP	This tick mark selects unsecured communications. No virtual connections are used by unsecured communication links. In this manner you can only display UDP messages.

Buttons

Connect	The connection is established and prepared for the read operation.
Read this	The data requested is read via this connection.

WriteA tab



[1] port data
[2] establish a connection
[3] transfer the data via the connection
[4] result-code of the write job

[5] information window for the status of the connection
[6] source data
[7] ASCII-text that must be transferred to the CP

This is where you activate an active write connection.

In the same way as for the READ active command you declare the destination block where the data must be transferred in addition to the data required for establishing the connection.

Input fields

Remote Host	IP-address of the station where to which you wish to write the data.
Remote Port	Port address of the remote station.
Local Port	Port address of your own (local) station. To simplify matters you can specify the same port address for remote and local.
Time (10mSec)	Definable interval for cyclic write operations. The minimum timer value for cyclic writes is 5.
OrgKennung	Type of destination block.
DBNr	Number of the destination block.
AnfAdr	Start address of destination block.
Len	Word length of destination block.

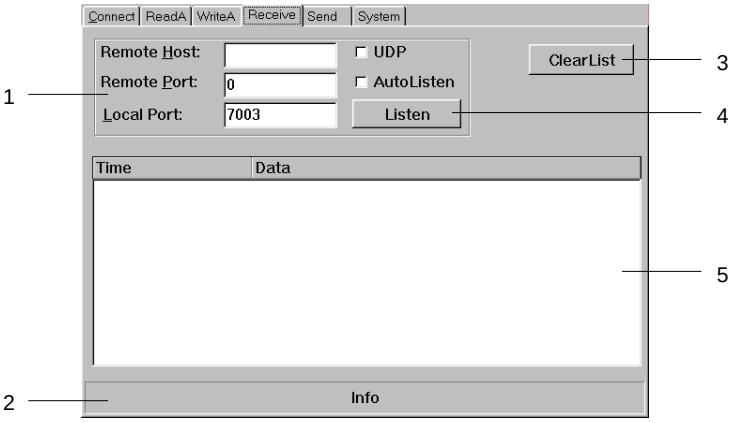
Tick-box

UDP	This tick mark selects unsecured communications. No virtual connections are used by unsecured communication links. In this manner you can only display UDP messages.

Buttons

Connect	The connection is established and prepared for the write operation.
Write this	Data entered into the ASCII field is transferred to the CP via the connection that was established by means of *Connect*.

Receive tab



[1] port data
[2] connection status information bar
[3] clear received list
[4] list the messages
[5] list of received messages

In this dialog window you can configure the reception of messages from a specific host.

Input fields

Remote Host IP-address of the station where the data must be saved.

Remote Port Port address of the remote station.

Local Port Port address of own (local) station. To simplify matters you can specify the same port address for remote and local.

Tick-box

UDP This tick mark selects unsecured communications. No virtual connections are used by unsecured communication links. In this manner you can only display UDP messages.

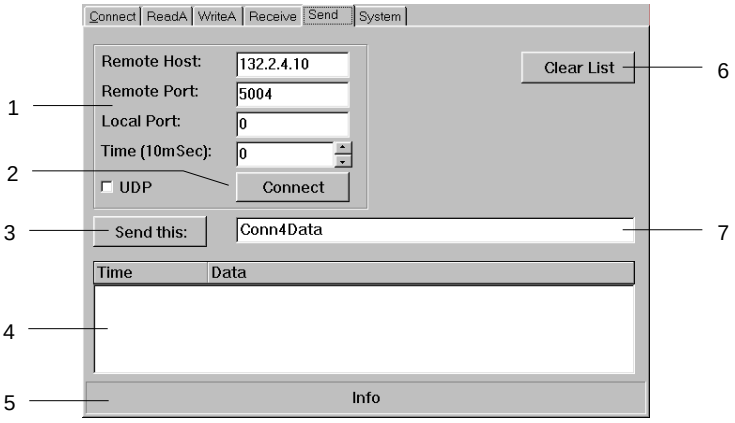
AutoListen If you select "AutoListen" the program goes to receive mode. Every message received from the remote CP is displayed in the list. Interruptions of the connection are detected and displayed, however, the program remains ready to receive data. As soon as the connection is reestablished messages will again be listed.

Buttons

Listen Any received messages are entered into the list. The listing is stopped when you click the "STOP" button or the connection is interrupted. You can also stop the listing by entering a new set of connection parameters.

ClearList Clears the received list, new entries will appear at the top of the list.

Send tab



[1] port data
[2] establish a connection
[3] transfer data via the connection
[4] list of transmitted messages
[5] information bar for the connection status
[6] clear the list of messages
[7] ASCII-text that must be transferred to the CP

You can use this dialog window to send a message to a specific host.

Input fields

Remote Host	IP-address of the station where the data must be saved.
Remote Port	Port address of the remote station.
Local Port	Port address of own (local) station. To simplify matters you can specify the same port address for remote and local.
Time (10mSec)	Definable interval for cyclic write operations. The minimum timer value for cyclic writes is 5.

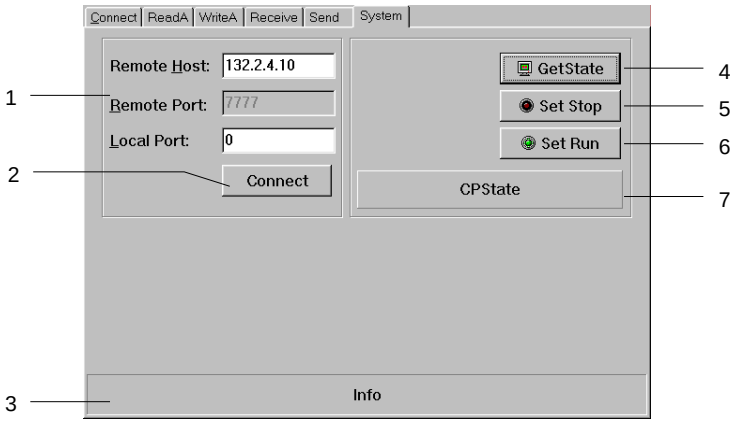
Tick-box

UDP	This tick mark selects unsecured communications. No virtual connections are used by unsecured communication links. In this manner you can only display UDP messages.

Buttons

Connect	The connection is established and prepared for the write operation.
Send this	Data entered into the ASCII field is transferred to the CP via the connection that was established by means of *Connect*.

System tab



[1] port data
[2] establish a connection
[3] information bar for the connection status
[4] CP status request
[5] CP in STOP
[6] CP in RUN
[7] status monitor, requested with GetState

This dialog window gives you information about the specified host-CP.

Input fields

Remote Host IP address of the station, where the data is stored to.

Remote Port Connection address of the foreign station.

Local Port Connection address of the own station. To simplify the process, you may use the same address for remote and local connections.

Buttons

Connect The connection is established and prepared for communication.

GetState Via the connection, established by means of *Connect*, the status of the CP is transferred and monitored in the status window. Monitored may be:

- Hardware-Stop (Run/Stop-switch at the CP is in stop position)
  The CP must not be remoted via the test program.
- Hardware-Run (Run/Stop-switch at the CP is in run position)
  The CP may be remoted via the test program.
- Software-Stop (Run/Stop-switch at the CP has to be in run position)
  The CP has been set to stop by means of *SetStop*.
- Software-Run (Run/Stop-switch at the CP has to be in run position)
  The CP has been set to start by means of *SetRun*.

SetStop The CP is set to stop. This function is only available, if the CP-switch is in run position.

SetRun The CP is set to run. This function is only available, if the CP-switch is in run position.

Chapter 6 Deployment of the CPU 21xDPM

Outline

Content of this chapter is the deployment of the CPU 21xDPM under Profibus. After a short introduction into the Profibus system, the project engineering and the usage under MPI is shown.

This chapter ends with information to the operating modes of the DP master and to the commissioning.

The following text describes:

- Principles of Profibus-DP
- Proyect engineering of a CPU 21xDPM
- Deployment of the MPI-interface and MMC
- Operating modes of the DP master
- Commissioning

Content

Principles

General

Profibus is an open fieldbus standard for building, manufacturing and process automation. Profibus defines the technical and functional properties of a serial fieldbus system that is used to create a network of distributed digital field automation equipment on the lower (sensor-/actuator level) to middle performance level (process level).

Profibus comprises various compatible versions. The specifications contained in this description refer to Profibus-DP.

Profibus-DP

Profibus-DP is particularly suitable for applications in production automation. DP is very fast, offers plug&play and is a cost-effective alternative to parallel cabling between PLC and the decentral/distributed periphery. Profibus-DP is conceived for high-speed data exchange on the sensor-actuator level.

The data exchange is processed cyclically. During a one bus cycle the master reads the input values from the various slaves and writes new output information into the slaves.

Master and slaves

Profibus distinguishes between active stations (masters) and passive stations (slaves).

Master equipment

Master equipment controls the data traffic on the bus. There may be also several masters at one Profibus. This is referred to as multi master operation. The bus protocol establishes a logical token ring between the intelligent devices connected to the bus.

A master like in the CPU 21xDPM may send unsolicited messages if it has the bus access permission (Token). In the Profibus protocol masters are also referred to as active stations.

Slave equipment

Typical slave equipment holds data of peripheral equipment, sensors, actuators or transducers. The VIPA Profibus are modular slave equipment, transferring data between the System 200V periphery and the leading master.

These devices do not have bus access permission in accordance with the Profibus standard. They may only acknowledge messages or transfer messages to a master if requested by this. Slaves occupy a very limited part of the bus protocol. Slaves are also referred to as passive stations.

Communication

The bus communication protocol provides two procedures for accessing the bus:

Master to Master

Communications with the master is also referred to as token passing procedure. Token passing guarantees that the station receives access permission to the bus. This access right to the bus is passed between the stations in form of a "token". A token is a specific message that is transferred via the bus.

When a master possesses the token, it has the access right to the bus and is allowed to communicate with all other active and passive stations. The token retention time is defined when the system is being configured. When the token retention time has expired, the token is passed along to the next master that acquires the bus access rights with the token so that this may now communicate with all other stations.

Master slave procedure

Data is exchanged in a fixed repetitive sequence between the master and the slaves assigned to this respective master. When you configure the system, you define which slaves are assigned to a certain master. You may also specify which DP slave is included in the cyclic exchange of application data and which ones are excluded.

The master slave data transfer is divided into parameterization, configuration and data transfer phases. Before a DP slave is included into the data transfer phase, the master verifies during the parameterization and configuration phase whether the specified configuration agrees with the effective configuration. This verification process checks the device type, format and length as well as the number of inputs and outputs. This provides you with effective protection against configuration errors.

The master handles application data transfers independently. In addition you may also send new configuration data to a bus coupler.

If in the status DE „Data Exchange“, the master is sending new basic data to the slave and the receipt of the slave transfers the recent input data to the master.

Data consistency

Data is referred to as being consistent, if it has the same logical contents. Data that belongs together is: the high- and low-byte of an analog value (word consistency) and the control and the status byte with the respective parameter word, required to access the registers.

The data consistency during the interaction between the peripherals and the controller is only guaranteed for 1Byte. That is, the bits of one byte are acquired together and they are transmitted together. Byte-wise consistency is sufficient for the processing of digital signals.

Transfer medium

As transfer medium Profibus uses an isolated drilled 2 core line based upon the RS485 interface or a duplex optical waveguide (OWG). The transfer rate is for both methods max. 12Mbaud.

More information about this topic is available at the installation guideline.

Electrical system over RS485

The RS485 interface is working with voltage differences. Though it is less irritable from failures than a voltage or a current interface. You are able to configure the network as well linear as in a tree structure. Your VIPA Profibus coupler includes a 9pin slot where you link up the Profibus coupler into the Profibus network as a slave.

The bus structure under RS485 allows an easy connection resp. disconnection of stations as well as starting the system step by step. Later expansions don't have any influence on stations that are already integrated. The system realizes automatically if one partner had a fail down or if it is new in the network.

Optical system via optical waveguide (OWG)

The optical waveguide system uses monochromatical light impulses. The optical waveguide is totally independent from disturbing voltage from other machines. An optical waveguide system is built up linear. Every module has to be connected with two links: one input link and one back. You don't need to terminate the last module.

For the structure is a linear one, connecting and disconnecting stations is not free of consequences.

Addressing

Every partner of the Profibus network has to identify itself with a certain address. This address may be existing only one time in the bus system and has a value between 0 and 125.

At the CPU 21xDPM you choose the address via your software tool.

Electronic Data Sheet (GSD-file)

To configure the slave connections in your own configuration tool, you've got all the information about your VIPA modules in form of an electronic data sheet file (German: ***G****erätes****t****amm****d****atei* = *GSD-file*).

Structure and content of this file are dictated by the Profibus User Organization (PNO) and may be seen there.

Install the GSD-file in your configuration tool. Look for more information in the online help of the according tool.

Project engineering CPU with integrated Profibus-DP master

Outline

For the project engineering of the Profibus-DP master you have to use the hardware manager from Siemens. Your Profibus projects are transferred via MPI into the CPU 21xDPM by means of the **PLC** functions. The CPU passes the data on to the Profibus-DP master.

Compatibility to SIMATIC manager from Siemens

The address allocation and the parameterization of the directly plugged-in modules takes place via the SIMATIC manager from Siemens in form of a virtual Profibus system.

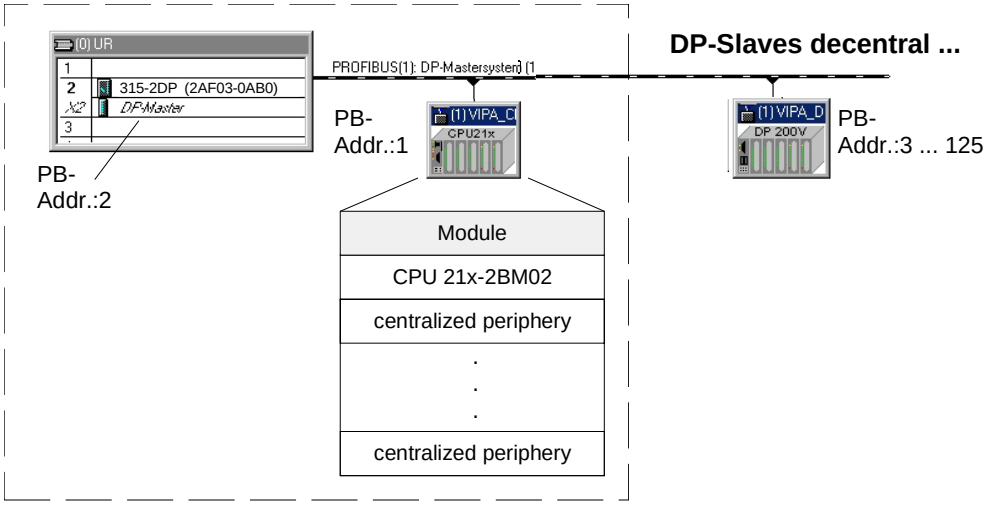
For the Profibus interface is standardized in software, you'll have the complete functionality of our modules at your disposal by including the GSD-file into the SIMATIC manager.

Steps of engineering

To be compatible with the SIMATIC manager from Siemens, you have to execute the following steps for the System 200V:

- Project the CPU 315-2DP with the DP master system (address 2).
- Add the Profibus slave **VIPA_CPU21x** with address 1.
- Insert the CPU type **21xDPM** at the 1st slot of the slave system.
- Include directly plugged-in peripheral modules also via this slave at the sequencing slots.

CPU 21xDPM central



Note!

For the project engineering of the CPU and the Profibus-DP masters a thorough knowledge of the SIMATIC manager and the hardware configurator from Siemens is required!

Preconditions

For the project engineering of the Profibus-DP master in the CPU 21xDPM the following preconditions have to be fulfilled:

- SIMATIC manager from Siemens is installed.
- When using Profibus-DP slaves of the Systems 100V, 200V and 300V from VIPA: GSD-files are included into the hardware configurator
- There is a transfer possibility between configuration tool and CPU 21xDPM.

Install Siemens hardware configurator

The hardware configurator is a part of the SIMATIC manager from Siemens. It serves the project engineering. The modules that may be configured here are listed in the hardware catalog.

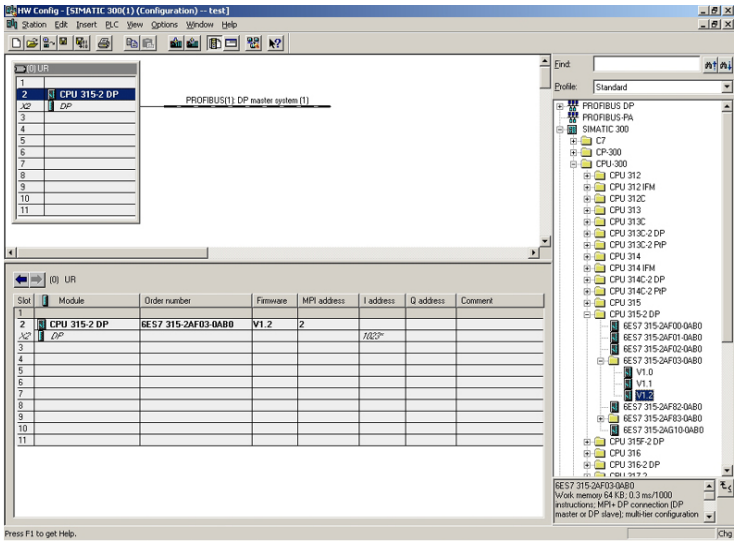
For the deployment of the Profibus-DP slaves of the Systems 100V, 200V and 300V from VIPA you have to include the modules into the hardware catalog by means of the GSD-file from VIPA.

Project engineering CPU 21xDPM

The following section describes the single steps of the project engineering.

Create a virtual Profibus system

- Create a new project System 300V.
- Add a profile rail from the hardware catalog.
- In the hardware catalog the CPU with Profibus master is listed as: Simatic300 > *CPU-300* > *CPU315-2DP* > **6ES7 315-2AF03-0AB0**
- Insert the CPU 315-2DP (**6ES7 315-2AF03-0AB0 V1.2**).
- Type the Profibus address >1 of your master.
- Click on "DP", choose the operating mode "DP master" under *object properties* and confirm your entry with OK.
- Click on "DP" with the right mouse button and choose "add master system". Create a new Profibus subnet.



Configure CPU section

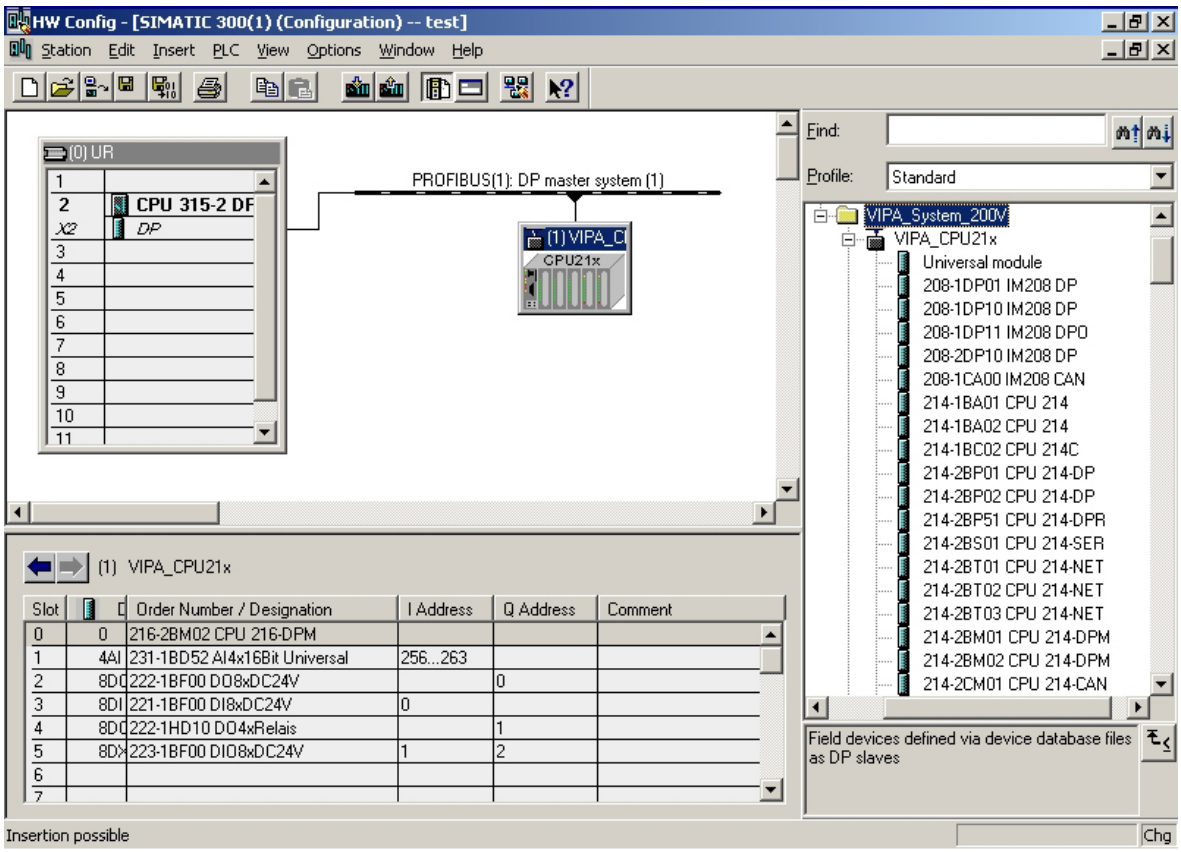
To be, like mentioned above, compatible to the SIMATIC manager from Siemens, you have to include the CPU section explicitly.

- Add the System "*VIPA_CPU21x*" to the Profibus subnet. You find this in the hardware catalog under *PROFIBUS DP > Additional field devices > IO > VIPA_System_200V*.
- Assign the Profibus address 1 to this slave.
- Place the CPU 21x-2BM02 from VIPA at the 1st slot in your configurator. Choose it in the hardware catalog under *VIPA-CPU21x.*

Now the project engineering of your Profibus-DP master and your CPU is finished. The following shows how to include the directly plugged-in System 200V modules.

Configure central periphery

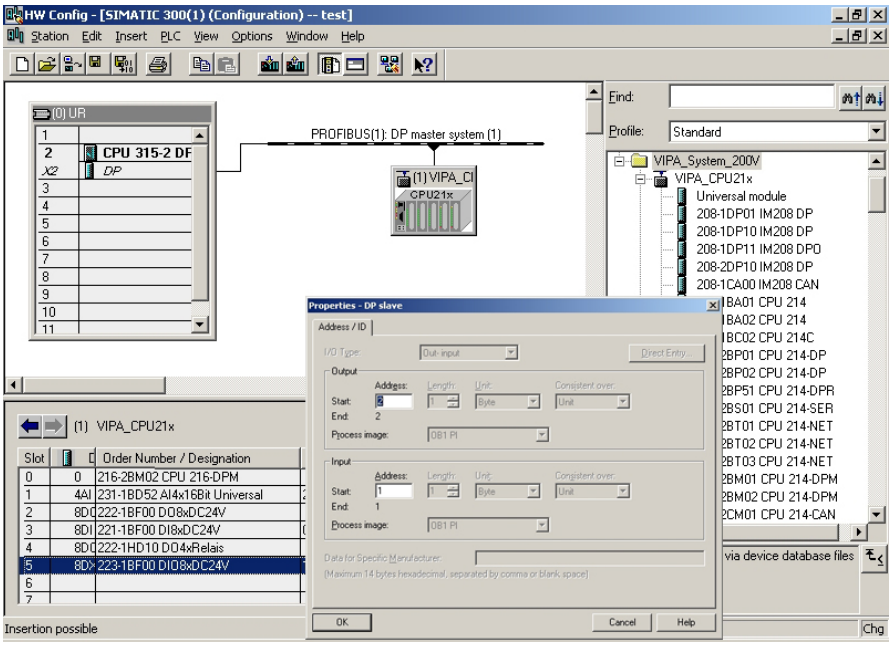
To include the modules plugged-in at the VIPA bus, you drag the according System 200V modules from the hardware catalog at *VIPA_CPU21x* and drop it on the slots below the CPU.



Parameterize modules

System 200V modules may get up to 16Byte parameter data from the CPU. Using the SIMATIC manager from Siemens, you may assign parameters for parameterizable System 200V modules at any time.

For this purpose double-click in the plug-in location overview on the concerning module.



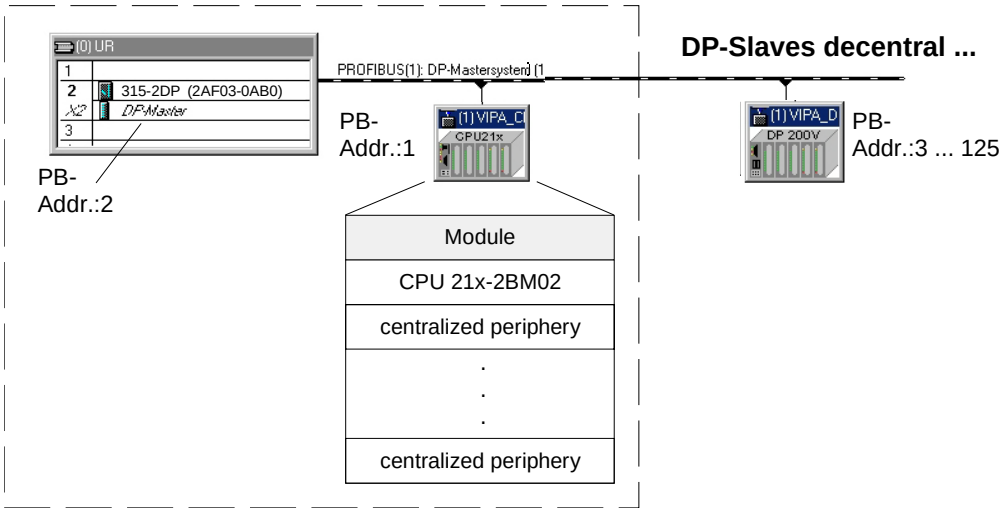
Configure DP-Slaves

For the project engineering of Profibus-DP slaves coupled at the DP master of the CPU 21xDPM, you approach analog to the DP200V system.

Search the concerning Profibus-DP slave in the hardware catalog and drag&drop it in the subnet of your master.

Assign a valid Profibus address to the DP slave (> 3).

CPU 21xDPM central



Project transfer

Outline

There are 2 possibilities for the transfer of your project into the CPU:

- Transfer via MPI
- Transfer via MMC at deployment of a MMC reading device

Transfer via MPI

The structure of a MPI net is in principal same to the one of a 1.5MBaud Profibus net. This means, the same rules are valid and you use the same components for assembly.

Per default, the MPI net is running with 187kBaud.

Every bus participant identifies itself at the bus with an unique MPI address.

You connect the single participant by means of bus interface plugs and the Profibus bus cable.

Terminating resistor

A conductor has to be terminated with its ripple resistor. Herefore you activate the terminating resistor at the first and the last participant of a net or a segment.

Please regard, that the participants with the activated terminating resistors are always supplied with voltage during start-up and operation.

Approach

- Connect your PU or your PC via MPI with the CPU.
  If your programming device has no MPI-interface, you may use the VIPA "Green Cable" to establish a serial point-to-point-connection from your PC to MPI.

  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 the notes about the "Green Cable" in chapter 1!
- Configure the MPI-interface of your PC.
- With **PLC** > *Load to module* in your projecting tool, you transfer your project into the CPU.
- For an additional backup of your project on the MMC, you plug in a MMC and transfer the user application to the MMC by means of **PLC** > *Copy RAM to ROM*.
  During the writing process the "MC"-LED at the CPU is blinking. Due to the system, a successful writing operation is signaled to soon. The writing operation is only finished, when the LED extinguishes.

Note!

Further information to MPI is to find under "Commissioning".

Configure MPI

Hints for the configuration of a MPI-interface is to find in the documentation of your programming software.

Here, only the usage of the VIPA "Green Cable" together with the programming tool from Siemens shall be shown.

The "Green Cable" establishes a serial point-to-point connection between the COM-interface of the PC and the MP^{2}I-interface of the CPU.

Attention!

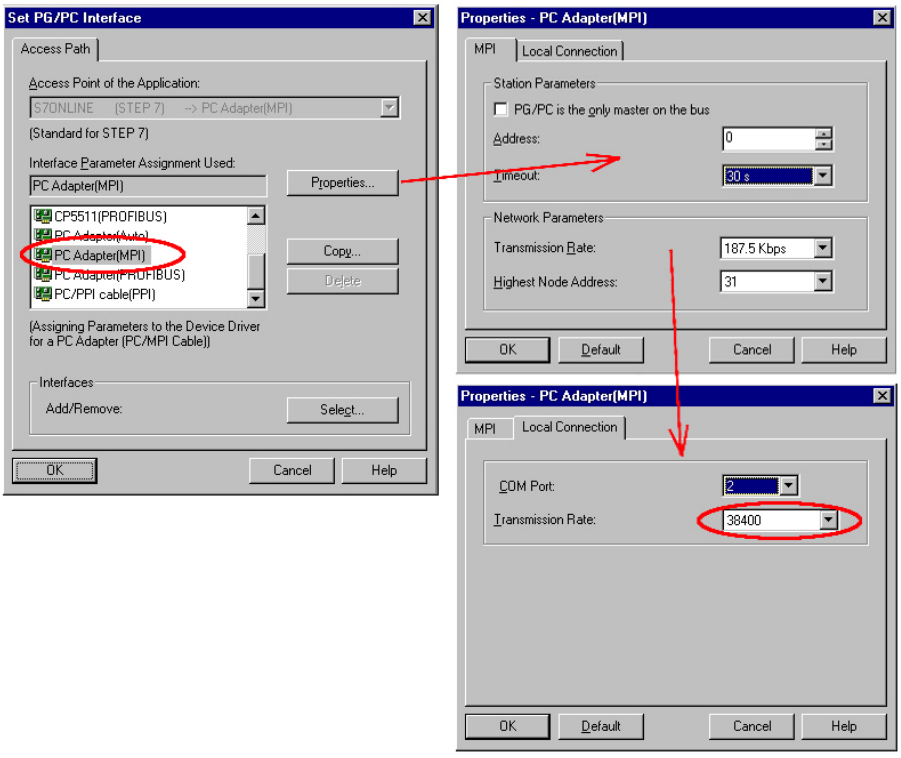
The Green Cable is a green connection cable, manufactured exclusively for the deployment at VIPA CPUs with MP^{2}I interface.

Please regard the notes about the "Green Cable" in chapter 1!

Approach

- Start the SIMATIC manager from Siemens.
- Choose **Options** > *Set PG/PC interface*

  → The following dialog window appears, where you may configure the wanted MPI-interface:



- Choose "PC Adapter (MPI)" from the list; if necessary you have to add it first.
- Click on [Properties].

→ In the following 2 sub dialogs you may configure your PC adapter, like shown in the picture.

Note!

Please make sure to adjust the transmission rate to 38400Baud when using the "Green Cable".

Usage of the MMC

As external storage medium the Multi Media Card (MMC) is used. The MMC is available from VIPA under the order number VIPA 953-0KX00.

The reading of the MMC takes always place after an OVERALL_RESET.

The writing on the MMC starts via a WRITE command from the hardware configurator from Siemens or via a MMC reading device from VIPA (Order No.: VIPA 950-0AD00). Thus it is possible to create applications at the PC, copy them to the MMC and transfer them to the VIPA CPU by plugging-in the MMC.

The MMC modules are delivered from VIPA preformatted with the file system FAT16.

Required Files

There may be several projects and subfolders on one MMC storage module. Therefore you have to make sure, that your recent project is located in the root directory and has the following file name: **S7PROG.WLD.**

Transfer CPU → MMC

If there is a plugged MMC in the CPU, the content of the battery buffered RAM is transferred to the MMC by means of a WRITE command.

The write command is started from the hardware configurator from Siemens via **PLC** > *Copy RAM to ROM.*

During the writing process the yellow "MC"-LED of the CPU is blinking.

Transfer MMC → CPU

The transfer of the user application from the MMC into the CPU always takes place after an OVERALL_RESET. The blinking of the yellow LED "MC" on the CPU marks the transfer process.

If there is no valid user application on the plugged MMC or the transfer fails, an OVERALL_RESET of the CPU takes place and the STOP-LED blinks for three times.

Now the master is linked up to the network with the following default parameters:

Default-Bus-Parameter: Address: 1, Transfer rate: 1.5MBaud

Note!

If the user application is larger than the user memory of the CPU, the content of the MMC is not transferred into the CPU.

If you initialize the writing process without plugged MMC, this leads to an error message about inadequate memory.

Before transferring the user application to the MMC it is convenient to initialize a compression.

DP master operating modes

STOP → RUN (automatically)

After POWER_ON and with valid configuration data in the CPU, the master switches automatically into RUN. There is no operating mode lever for the master.

Now the communication to the DP slaves is established. During this time only the RUN-LED is on. At successful communication and valid bus parameters, the DP master switches to Data Exchange (DE). The LEDs RUN and DE are blinking.

At invalid parameters the DP master switches to RUN and monitors a parameterization error via the IF-LED.

Now the DP master is linked up at the bus with the following default bus parameters:

Default-Bus-Parameter: Address:1, Transfer rate:1.5MBaud.

RUN

In the RUN mode, the RUN- and the DE-LEDs are blinking. Now data may be exchanged. If an error occurs, like e.g. DP slave failure, this is shown at the DP master via the ERR-LED and an alarm is initialized to the CPU.

Note!

If the CPU goes into STOP during operation, the DP master stays in RUN.

By means of the BASP signal all outputs of the peripheral modules, linked up via the DP slaves, are set to zero.

After a slave failure, the process image of the inputs is in the same state than before the failure.

Commissioning and Start-up behavior

Check list for commissioning

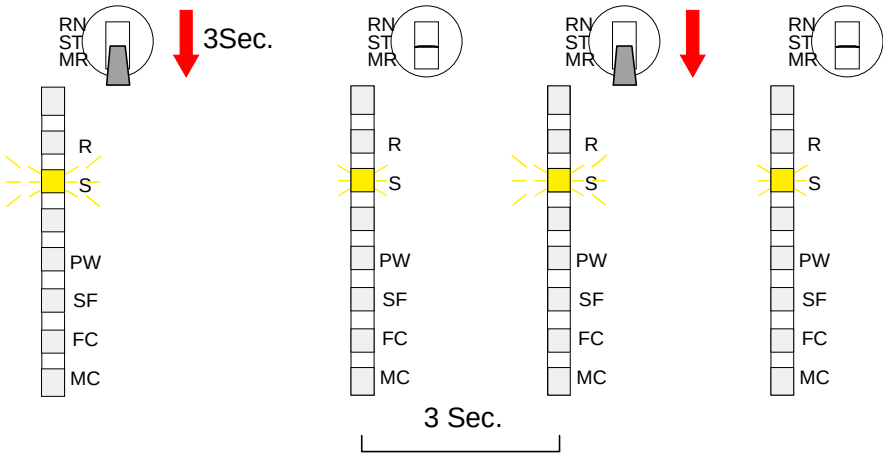
- Turn off your power supply
- Build up your system
- Cable your system
- Plug in your MMC with CPU program and Profibus project
- Turn on your power supply
- For transferring your project from the MMC into the CPU, request an OVERALL_RESET

Turn on power supply

Turn on the power supply. The course of events, described under "start-up behavior", is following.

OVERALL_RESET

The following picture shows the approach once more:



Default boot procedure, as delivered

When the CPU is delivered it has been reset.

After a STOP → RUN transition the CPU switches to RUN without program.

After NETZ_EIN (power on), the DP master tries to get parameters from the CPU.

For the master doesn't get valid parameters from the CPU, it starts with default parameters (Addr.:1, 1.5MBit) from its ROM and shows this via the "IF"-LED.

Boot procedure with valid data in the CPU

The CPU switches to RUN with the program stored in the battery buffered RAM.

The DP master receives valid parameters and starts with them.

Boot procedure with valid Memory Card

The reading of a MMC only takes place after an OVERALL_RESET.

After the OVERALL_RESET, the DP master proofs the validity of the parameters in the CPU.

If these are valid, they are taken over.

If they are invalid, the CPU switches to STOP and the DP master starts with the default values.

Start-up at empty battery

The battery is loaded directly via the integrated power supply by means of a special load electronic and provides a buffer of max. 30 days. Is this time exceeded, there may be a total discharge of the battery and the battery buffered RAM is erased.

Now the CPU proceeds an overall_reset. If a MMC is plugged-in, the application on it is transferred into the RAM and the DP master is supported with parameters.

If these are valid, the DP master links-up to the bus with those parameters.

If they are not valid, the master starts with default values of its ROM (Add.:1; 1.5MBit) and monitors this via the IF-LED.

Depending on the operating mode selected at the module, the CPU switches to RUN resp. stays in STOP.

This procedure is fixed in the diagnostic buffer by means of the following entry: "Automatic start OVERALL_RESET (unbuffered NETZ_EIN/ Power_on)".

Chapter 7 Deployment of the CPU 21xDP

Outline

This chapter describes applications of the CPU 21xDP under Profibus. You'll get all information for the deployment of an intelligent Profibus-DP slave. An example for the CP 21xDP and the CPU 21xDPM concludes the chapter.

This chapter contains a description of:

- the principles of Profibus-DP
- configuration and parameterization of a CPU 21xDP
- diagnostic and status messages
- assembly and commissioning
- communication example

Contents

Principles

General

Profibus is an international open fieldbus standard for building, manufacturing and process automation. Profibus defines the technical and functional properties of a serial fieldbus system that can be used to create a network of distributed digital field automation equipment on the lower (sensor-/actuator level) to middle performance level (process level).

Profibus comprises various compatible versions. The specifications contained in this description refer to Profibus-DP.

Profibus-DP

Profibus-DP is particularly suitable for applications in production automation. DP is very fast, offers plug&play and is a cost-effective alternative to parallel cabling between PLC and the distributed/decentralized periphery. Profibus-DP is conceived for high-speed data exchange on the sensor-actuator level.

The data exchange happens cyclically. During one bus cycle the master reads the input values from the various slaves and writes new output information into the slaves.

Master and slaves

Profibus distinguishes between active stations (masters) and passive stations (slaves).

Master equipment

Master equipment controls the data traffic on the bus. There may be also several masters at one Profibus. This is referred to as multi master operation. The bus protocol establishes a logical token ring between the intelligent devices connected to the bus.

A master is allowed to send unsolicited messages if it has the bus access permission (Token). In the Profibus protocol these masters are also referred to as active stations.

Slave equipment

Typical slave equipment holds data of peripheral equipment, sensors, actuators, transducers. The VIPA Profibus couplers are modular slave devices that transfer data between the System 200V periphery and the leading master.

These devices do not have bus access permission in accordance with the Profibus standard. They may only acknowledge messages or transfer messages to a master if requested by the respective master. Slaves are also referred to as passive stations.

Communication

The bus communication protocol provides two procedures for accessing the bus:

Master to Master

Communication with the master is also referred to as token passing procedure. Token passing guarantees that the station receives access permission to the bus. This access right to the bus is passed between the stations in form of a "token". A token is a specific message that is transferred via the bus.

When a master possesses the token it also has the access right to the bus and is allowed to communicate with all other active and passive stations. The token retention time is defined at system configuration. When the token retention time has expired, the token is passed along to the next master that acquires the bus access rights with the token so that it may communicate with all other stations.

Master slave procedure

Data is exchanged in a fixed repetitive sequence between the master and the slaves assigned to the respective master. When you configure the system you define which slaves are assigned to a certain master. You may also specify which DP slave is included in the cyclic exchange of application data and which ones are excluded.

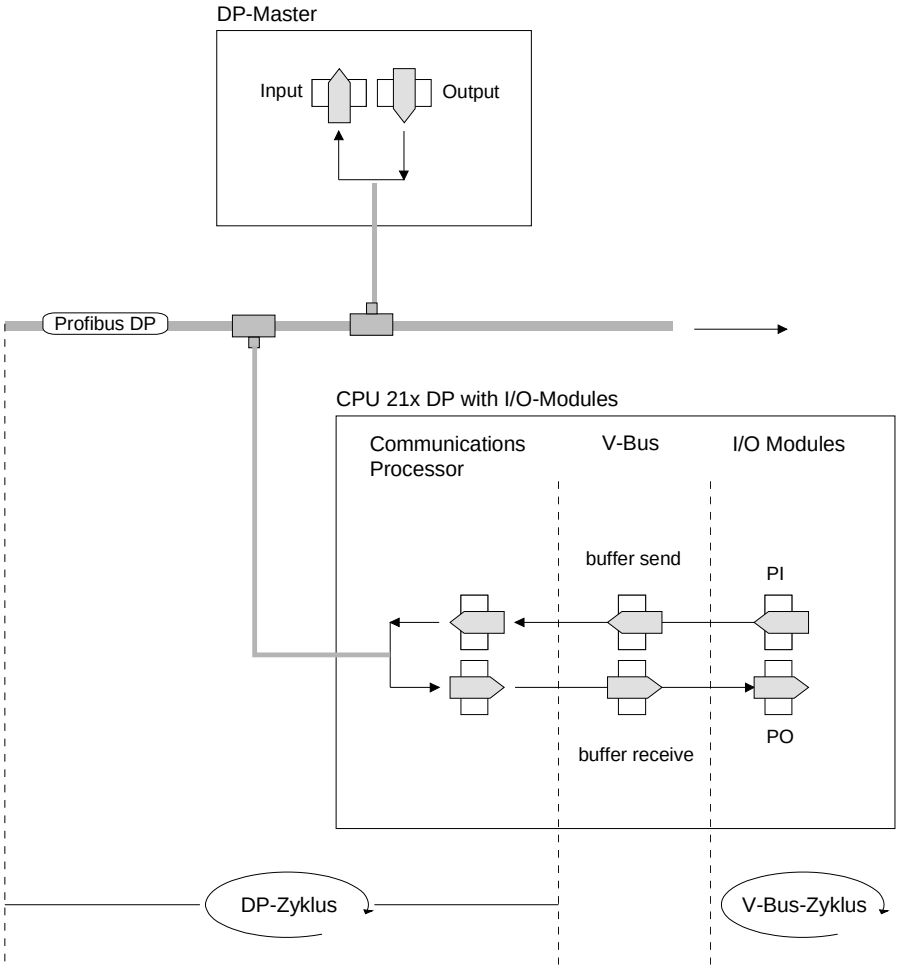
The master slave data transfer is divided into parameterization, configuration and data transfer phases. Before a DP slave is included in the data transfer phase the master verifies during the parameterization and configuration phase, whether the specified configuration agrees with the effective configuration. This verification process checks the device type, format and length as well as the number of inputs and outputs. This provides you with effective protection against configuration errors.

The master handles application data transfers independently. In addition you may also send new configuration data to a bus coupler.

If in the status DE „Data Exchange“, the master is sending new basic data to the slave and the responding telegram of the slave transfers the recent input data to the master.

The principle of data transfer operations

The data exchange between the DP master and the DP slave is performed in a cycle using send and receive buffers.



PI: Process image of the inputs

PO: Process image of the outputs

V-bus cycle

In one V-bus cycle (V-bus = VIPA-backplane bus) all input data of the single modules are collected in the PE and all output data from the PO are transferred to the output modules. After the data exchange is completed, the PI is transferred to the sending buffer (buffer send) and the content of the input buffer (buffer receive) is transferred to PO.

DP cycle

In one Profibus cycle the master contacts all its slaves with a data exchange. There the memory areas assigned to the Profibus are written resp. read.

Afterwards the DP master transmits data of the input area to the receive buffer of the communication processor and the data of the send buffer is transferred into the Profibus output area.

The DP master to DP slave data exchange on the bus is repeated cyclically and does not depend on the V-bus cycle.

V-bus cycle vs. DP cycle

To guarantee a simultaneous data transfer the V-bus cycle time should always be same or lower than the DP cycle time.

In the delivered GSD you'll find the parameter **min_slave_interval = 3ms**.

Thus guarantees that the Profibus data on the V-bus is updated latest every 3 ms. Though you are allowed to execute one data exchange with the slave every 3 ms.

Data consistency

Data is referred to as being consistent, if it has the same logical contents. Data that belongs together is: the high- and low-byte of an analog value (word consistency) and the control and the status byte with the respective parameter word required to access the registers.

The data consistency during the interaction between the peripherals and the controller is only guaranteed for 1 byte. That is, the bits of one byte are acquired together and they are transmitted together. Byte-wise consistency is sufficient for the processing of digital signals.

Where the length of the data exceeds a single byte, e.g. analog values, the data consistency must be expanded. Profibus guarantees consistency for the required length of data. Please ensure that you use the correct method to read consistent data from the Profibus master into your PLC.

For additional information please refer to the manual on your Profibus master as well as the one for the interface module.

Restrictions

If a high-level master fails, this is not recognized automatically by the CPU. You should always pass along a control byte to indicate the presence of the master thereby identifying valid master data.

The example at the end of this chapter also explains the use of the control byte.

Diagnosis

There is a wide range of diagnosis functions under Profibus-DP to allow a fast error localization. The diagnosis data are broadcasted by the bus system and summarized at the master.

Transfer medium

As transfer medium Profibus uses an isolated twisted-pair cable based upon the RS485 interface or a duplex photo cable. The transfer rate is for both methods max. 12Mbaud.

More information about this topic is available at „installation guideline“.

Electrical system over RS485

The RS485-interface is working with voltage differences. Though it is less irritable from failures than a voltage or a current interface. You are able to configure the network as well linear as in a tree structure. Your VIPA Profibus coupler includes a 9pin slot where you connect the Profibus coupler into the Profibus network as a slave.

The bus structure under RS485 allows an easy connection resp. disconnection of stations as well as starting the system step by step. Later expansions don’t have any influence on stations that are already integrated. The system realizes automatically if one partner had a fail down or is new in the network.

Optical system via optical waveguide (OWG)

The optical waveguide system uses monochromatically light impulses. The optical waveguide is totally independent from disturbing voltage from other machines. An optical waveguide system is built up linear. Every module has to be connected with two links: one input link and one back. You don’t need to terminate the last module.

For the structure is a linear one, connecting and disconnecting stations is not free of consequences.

Addressing

Every partner of the Profibus network has to identify itself with a certain address. This address may be existing only one time in the bus system and has a value between 0 and 125.

At the CPU 21xDP you choose the address via your software tool.

GSD-file

To configure the slave connections in your own configuration tool, you’ve got all the information about your VIPA modules in form of an electronic data sheet file.

Structure and content of this file are dictated by the Profibus User Organization (PNO) and can be seen there.

Install this file in your configuration tool. Look for more information at the next pages "CPU 21xDP configuration" or in the manual of the according tool.

CPU 21xDP configuration

Outline

In contrast to the VIPA Profibus slave IM 253DP, the Profibus coupler in the CPU 21xDP is an "intelligent coupler".

The "intelligent coupler" processes data that is available from an input or an output area of the CPU. This area and an area for status and diagnostic data are fixed via the CPU 21xDP properties. Separate memory areas are used for input and for output data. Those areas are to handle with your PLC program.

Due to the system, the address areas occupied by the coupler are not displayed in the hardware configurator from Siemens. For the directly plugged-in System 200V modules are also included in the periphery address range, there may occur address overlappings during the automatic address allocation.

Note!

For configuring the CPU and the Profibus-DP master a thorough knowledge of the SIMATIC manager and the hardware configurator from Siemens is required!

Possibility of configuration in the SIMATIC manager from Siemens

The address allocation and the parameterization of the directly plugged-in System 200V modules takes place via the SIMATIC manager from Siemens in form of a virtual Profibus system.

For the Profibus interface is software standardized, VIPA is able to guarantee the availability of the complete functions of the System 200V modules by including the GSD-file vipa_21x.gsd in the manager.

Steps of the CPU 21xDP configuration

To be compatible with the SIMATIC manager from Siemens, you have to execute the following steps:

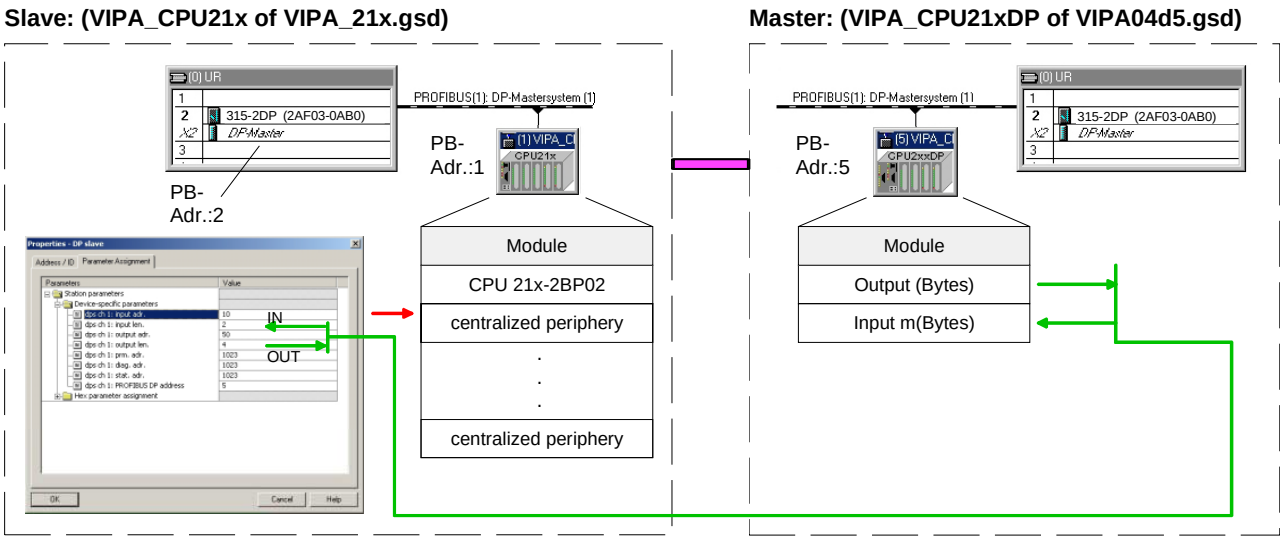
- Project the CPU 315-2DP with DP master system (address 2).
- Add the Profibus slave "VIPA_CPU21x" from the VIPA_21x.gsd with address 1.
- Include the CPU type 21xDP at the 1st slot of the slave system.
- Adjust the Profibus parameters of the CPU 21xDP.
- Include directly plugged-in periphery modules at the following slots.
- Transfer project via MPI into the CPU 21xDP

Steps of the master configuration

- Project CPU with DP master system (address 2).
- Add the Profibus slave "VIPA_CPU2xxDP" from the VIPA04D5.gsd.
- Set the Profibus in- and output ranges starting with the 1st slot. Please regard that the length settings at the slave must be congruent to the Byte settings in the master!

Reference between master and slave

The following picture illustrates the project engineering at the master and the slave:



Project engineering of the CPU 21xDP

The CPU 21xDP is configured in the SIMATIC manager from Siemens.

Preconditions

Following requirements for System 200V resp. System 300 have to be fulfilled before configuration:

- SIMATIC manager from Siemens is installed.
- GSD-file has been integrated in the hardware configurator of Siemens.
- There is a possibility for data transfer between PG and CPU.

Install hardware configurator from Siemens

The hardware configurator is part of the SIMATIC manager from Siemens for project engineering. The modules parameterizable via this tool are in the hardware catalog of this tool.

For the deployment of the Profibus-DP slaves from VIPA you have to include the modules via an GSD-file from VIPA in the hardware catalog.

Include GSD-file

The installation of a GSD-file requires the unzipped version as .gsd.

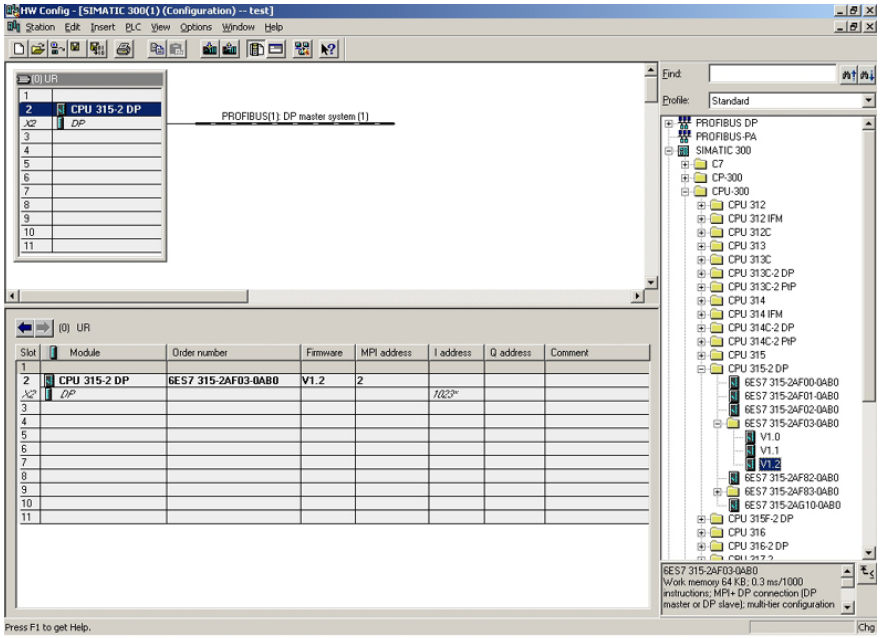
- Start the hardware configurator from Siemens. For including a new GSD-file, no project may be opened.
- Open the GSD-file window under **Options** > *Install new GSD-file*.
- Insert the delivered data device and select the wanted GSD-file. Via [Open] you install the GSD-file.

Now you'll find the VIPA modules included in the hardware catalog under *Profibus-DP > Additional Field devices > I/O > VIPA*.

Create a virtual Profibus system

- Create a new System 300 project and add a profile rail from the hardware catalog.
- Add the CPU 315-2DP (**6ES7 315-3AF03-0AB0**). You find the CPU with Profibus master in the hardware catalog under: *Simatic* > *CPU-300* > *CPU 315-2DP* > **6ES7 315-3AF03-0AB0 V1.2**.
- Assign the Profibus address 2 to your master.
- Click at DP and select the operating mode "DP master" under *Object properties*. Confirm your entry with OK.
- By clicking on "DP" with the right mouse button, the context menu opens. Choose "Add master system". Create a new Profibus subnet via NEW.

  The following picture shows the created master system:



Project engineering CPU 21xDP and modules

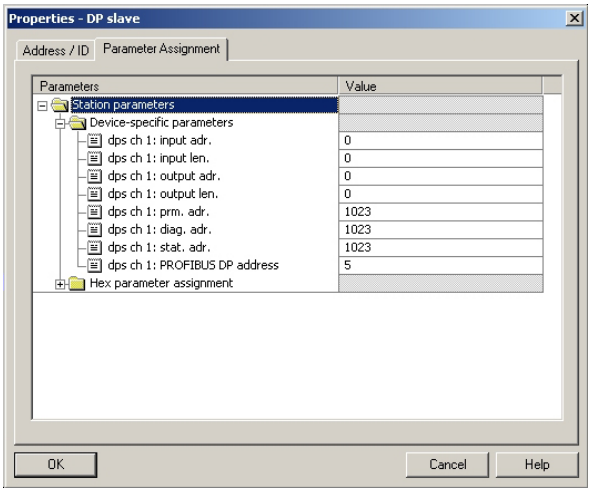
To be compatible with the SIMATIC manager from Siemens you have to include the CPU 21xDP explicitly like mentioned above.

- Add the System "VIPA_CPU21x" to your subnet. The module is to find under *PROFIBUS-DP* > *Additional Field devices* > *I/O* > *VIPA_System_200V*. Assign the Profibus address 1 to the DP slave.
- Place the CPU 21x-2BP02 from VIPA at the 1st slot of the hardware configurator.
- You may fix the data areas of the Profibus section in the CPU parameter window. More details are under "Include Profibus section".
- Include your System 200V modules in the assembled order.
- If there is a DP master at the backplane bus, you have to include this at the concerning plug-in location, too. Here the configuration takes place via WinNCS from VIPA.
- Save your project and transfer it via MPI to the CPU 21xDP.

Please regard the hints in chapter "Principles"!

Including the Profibus section

The Profibus section creates an image of its data area in the addressing space of the CPU 21xDP. The assignment of the areas happens via the properties of the CPU 21xDP. Via a double-click at the CPU 21xDP you reach the dialog window for parameterization of the data areas for the Profibus slave. Details are to find in the chapter "DP slave parameters".

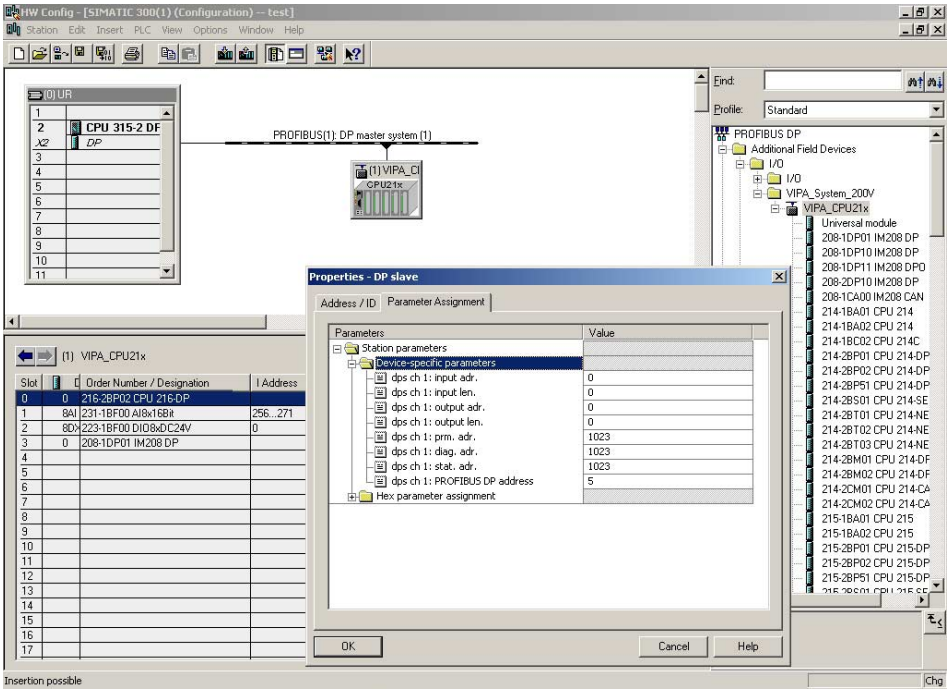


Attention!

Please regard, that the lengths of the data areas are identical at the master and the slave project engineering.

Due to the restrictions imposed by the system, the memory areas of the CPU that are used for the Profibus section may only be displayed in the CPU configuration window.

In the following all relevant dialog windows of the slave project engineering are shown. Here you may also see, how to include your System 200V:

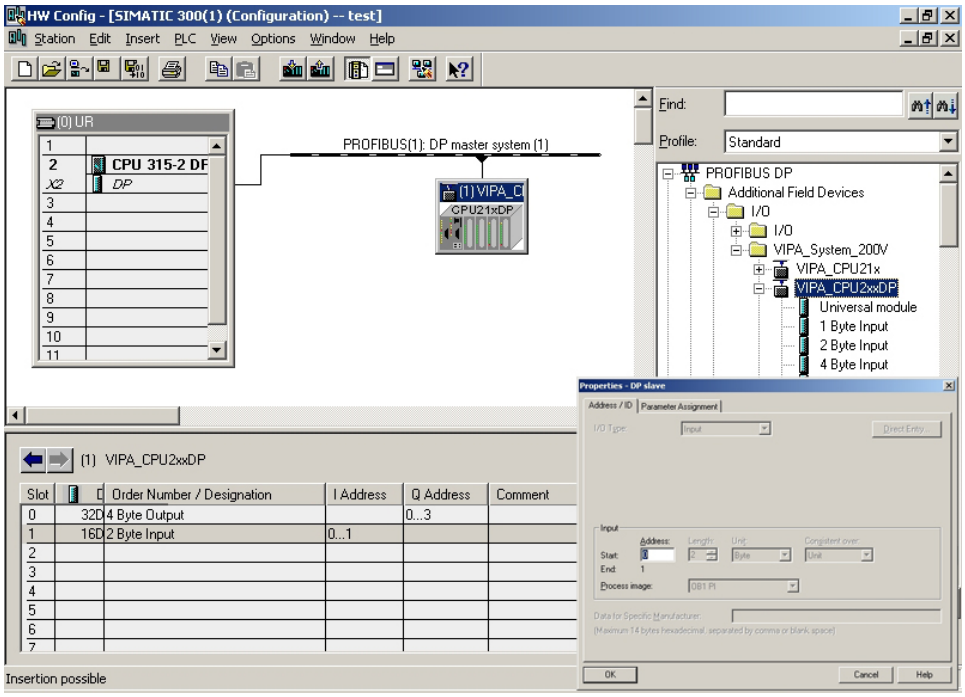


Configuration in the leading DP master

To include the CPU 21xDP into a leading master system, the GSD-file vipa04d5.gsd included in the package is required.

- Start your configuration program and configure the Profibus-DP master that will be the leader of your CPU 21xDP.
- Add a DP slave of the station type "CPU2xxDP". This is to find in the hardware catalog under *Profibus-DP > Additional field devices > I/O > VIPA > VIPA_CPU2xxDP*.
- Assign the Profibus address to the DP slave that you've parameterized at the slave.
- For the Profibus communication, create the same I/O range that you've parameterized at the slave in form of "modules". Please regard that a slave output area relates to a master input area and vice versa.
- Save your project and transfer it into the CPU of your master system.

In the following the relevant dialog windows for the master configuration are shown:



Note!

If your DP master system is a System 200V module from VIPA, you may parameterize the directly plugged-in modules by including a "DP200V" slave system.

To enable the VIPA CPU to recognize the project as central system, you have to assign the Profibus address 1 to your slave system!

Please take care at deployment of the IM 208 Profibus-DP master that this has a firmware version v3.0 or higher; otherwise this may not be used at a CPU 21x with a firmware version v3.0 or higher. The firmware versions are on a label at the backside of the modules.

DP slave parameters

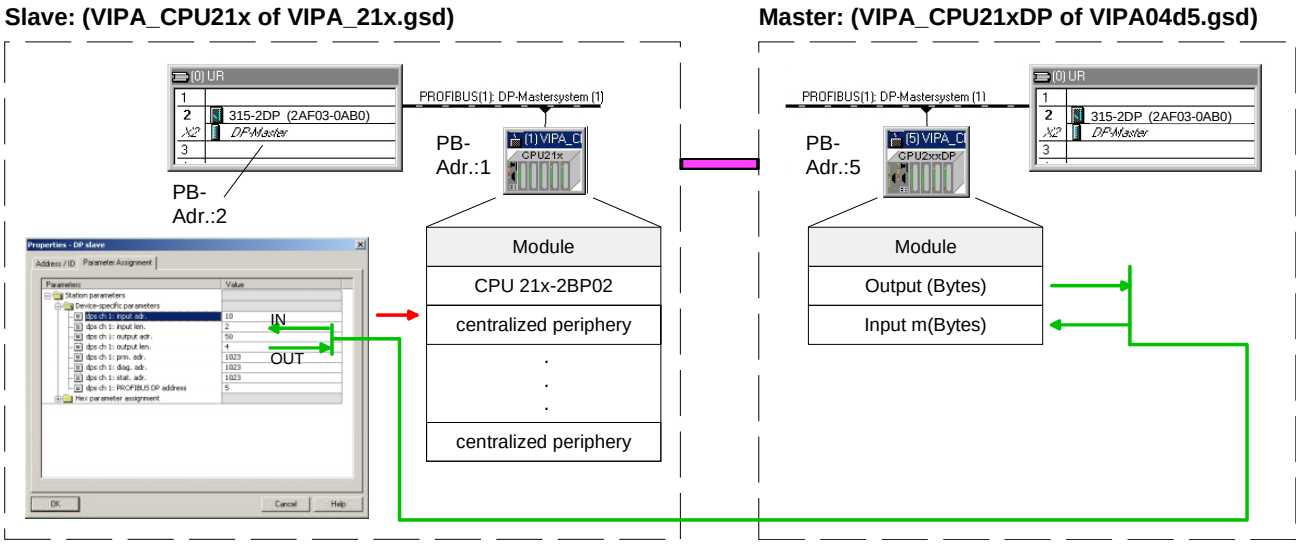
Outline

With an "intelligent slave", the Profibus section writes its data areas into the memory area of the CPU 21xDP. The assignment of the areas happens via the "properties" of the CPU 21xDP.

The input or output areas have to be supported with an according PLC program.

Attention!

The length values for in- and output area have to be identical to the byte values of the master configuration. Otherwise there is no Profibus communication possible and the master notes a slave failure!



Release CPU areas

As soon as you set a length value to 0, the concerning data does not occupy any memory space in the CPU.

By assigning 255 (memory limit) at the parameters PRN, DIAG and STAT you may also release memory space in the CPU.

Note!

Until CPU firmware version v2.2.0 the CPU 21x and the Profibus-DP system support an address range from 0 to 255.

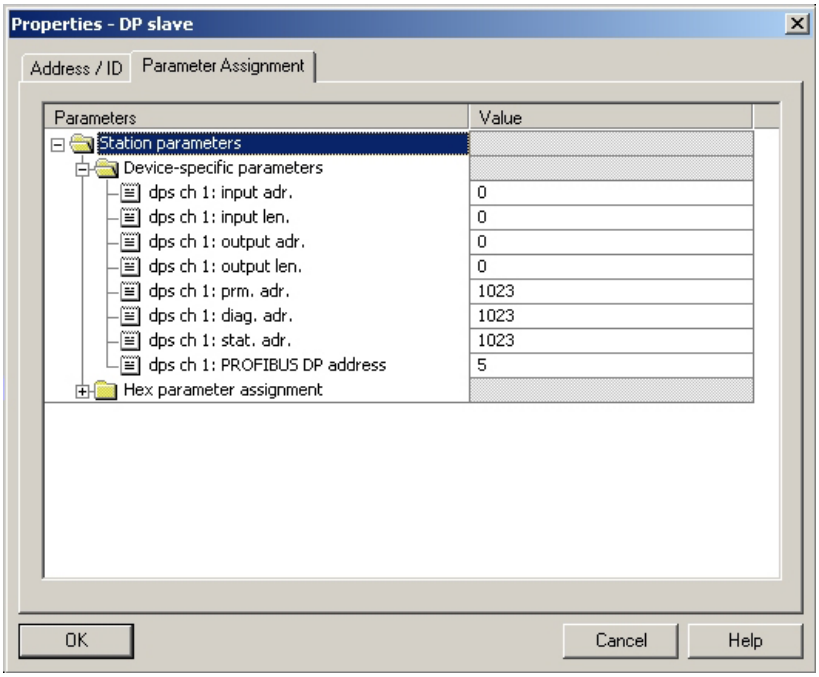
From firmware version v3.0 on, the CPU 21x and the Profibus-DP system from VIPA support an address range from 0 to 1023.

The firmware versions are on a label at the backside of the modules.

Here 1023 is the value for deactivating *PRN, DIAG* and *STAT*.

Description of the parameter data

Via a double-click at the CPU 21xDP in the hardware configurator from Siemens, the following dialog window for configuring the Profibus slave data areas appears:



input adr, len

Address, from where on the data coming from Profibus shall be stored in the CPU, together with the according length.

The length value of 0 occupies no CPU memory space for the input area.

output adr, len

Address, from where on the data are stored that shall be send via Profibus. here also you predefine the data length via *len*.

The length value of 0 occupies no CPU memory space for the input area.

prm. adr.

The parameter data are an extract of the parameter telegram. The parameter telegram is created during the master configuration and is send to the slave when:

- the CPU 21xDP is in start-up
- the connection between CPU 21xDP and master has been interrupted, like e.g. short-time release of the bus connector.

A parameter telegram exists of Profibus-specific data (bus parameters) and user specific data, where at the CPU 21xDP the in- and output bytes are defined.

The user specific data (Byte 7...31) is mapped into the memory area of the CPU with a fixed length of 24Byte, starting from the address fixed under *prm*.

Thus you may proof the parameters your slave is getting from the master.

diag. adr.

The various diagnostic functions of Profibus-DP allow a fast error detection and localization. The diagnostic messages are transferred via the bus and collected at the master.

The CPU 21xDP is sending diagnostic data at request from the master or in case of an error. The diagnostic data consists of:

- Norm diagnostic data (Byte0...5),
- Device-related diagnostic data (Byte6...10)
- **User-specific diagnostic data (Byte11...15)**

Via *diag* you define the start address, from where on the 6Byte user-specific diagnostic data shall be stored in the CPU.

You may initiate and influence diagnostics by controlled access to this area.

Note!

More detailed information about the structure and the control possibilities on diagnostic messages are to find under "Diagnostic functions".

stat. adr.

The recent state of the Profibus communication is readable from a 2Byte status area in the periphery address area of the CPU, starting from the status address.

Note!

More detailed information about the structure of a status message is to find under "Status message internal to CPU".

Profibus DP address

Via this parameter you assign a Profibus address to your Profibus system.

Release CPU areas

As soon as you assign the length 0, the according data don't occupy any memory space in the CPU.

You may also release CPU memory space by assigning the address range limit (255 resp. 1023 for CPU firmware versions > 2.2.0) to the parameters *prn*, *diag* and *stat*.

Diagnostic functions

Outline

Profibus-DP is provided with an extensive set of diagnostic functions that may be used to locate problems quickly and effectively. Diagnostic messages are transferred via the bus and collected by the master.

The CPU 21xDP transmits diagnostic data when requested by the master or when an error occurs. Since a portion of the diagnostic data (Byte11...15) is located in the peripheral address area of the CPU, you may start the diagnostics and modify the diagnostic data. Diagnostic data consists of:

- standard diagnostic data (Byte 0.. 5),
- equipment related diagnostic data (Byte 6...15).

Structure

The structure of the diagnostic data is as follows:

Standard diagnostic data

Byte 0	Station status 1
Byte 1	Station status 2
Byte 2	Station status 3
Byte 3	Master address
Byte 4	Ident number (low)
Byte 5	Ident number (high)

Device related diagnostic data

Byte 6	Length and code of device related diagnostics
Byte 7	Equipment related diagnostic messages
Byte 8 ... Byte 10	reserved
Byte 11 ... Byte 15	User-specific diagnostic data is mapped into the peripheral addressing range of the CPU and may be modified and send to the master.

Standard diagnostic data

Details on the structure of the standard diagnostic data are available from the literature on the Profibus standards. This documentation is available from the Profibus User Organization.

The slave standard diagnostic data have the following structure:

Byte	Bit 7 ... Bit 0
0	Bit 0: fixed at 0 Bit 1: slave not ready for data transfer Bit 2: configuration data is not identical Bit 3: slave got extern diagnostic data Bit 4: slave does not provide this function Bit 5: fixed at 0 Bit 6: wrong parameterization Bit 7: fixed at 0
1	Bit 0: slave needs new parameterization Bit 1: statistical diagnosis Bit 2: fixed at 1 Bit 3: response control active Bit 4: freeze command received Bit 5: sync command received Bit 6: reserved Bit 7: fixed at 0
2	Bit 0...Bit 6: reserved Bit 7: Diagnostic data overflow
3	Master address after parameterization FFh: Slave is without parameterization
4	Ident number High Byte
5	Ident number Low Byte

Device related diagnostic data

The device related diagnostic data provide detailed information on the slave and the peripheral modules. The length of the device related diagnostic data is fixed at 10Byte.

Byte	Bit 7...Bit 0
6	Bit 0...5: length equipment related diagnostic data 001010: length 10Byte (fixed) Bit 6...7: Code for equipment related diagnostics 00: Code 00 (fixed)
7	Bit 0...Bit 7: equipment related diagnostic message 12h: Error: data length parameters 13h: Error: data length configuration data 14h: Error: configuration entry 15h: Error: VPC3 buffer calculation 16h: Error: missing configuration data 17h: Error: Difference DP parameterization and configuration 40h: User defined diagnostic is valid
8...10	reserved
11...15	**User specific diagnostic data that are stored behind the diagnostic status byte in the process picture of the CPU. This data may be overwritten and forwarded to the master.**

Starting diagnostics

In case of a diagnostic action the contents of Byte 11...15 of the equipment related diagnostic data will be transferred to the process image of the CPU and get a status byte in front.

Where this diagnostic block with a length of 6Byte is located in the process image is definable via the CPU parameters.

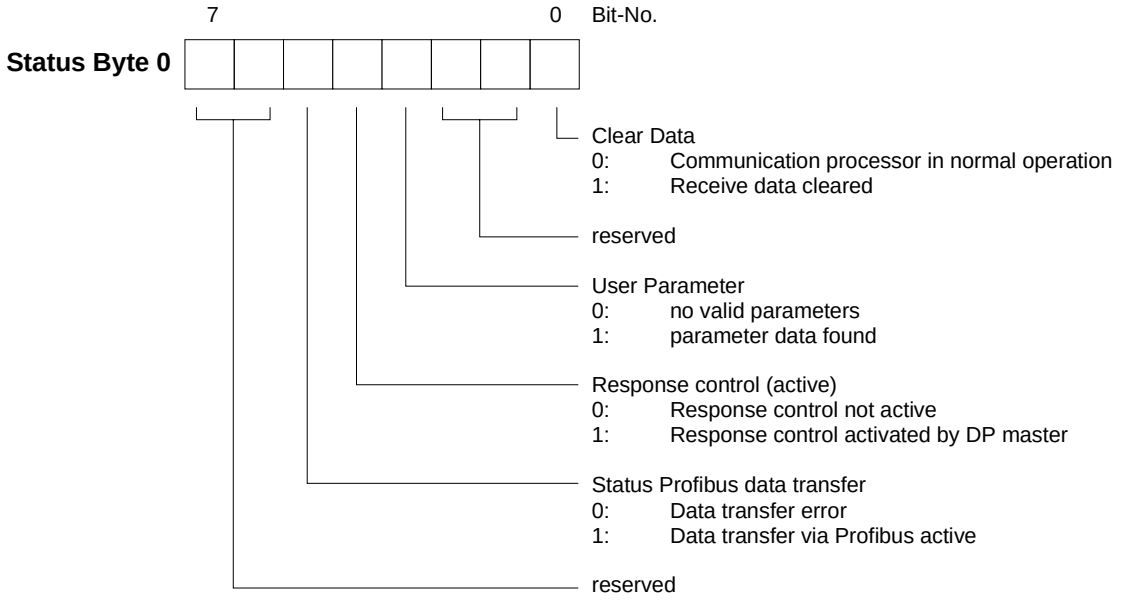
You start diagnostics by means of a status change from 0 $\rightarrow$ 1 in the diagnostic status byte. This transmits the respective diagnostic message to the master. **A status of 0000 0011 is ignored!**

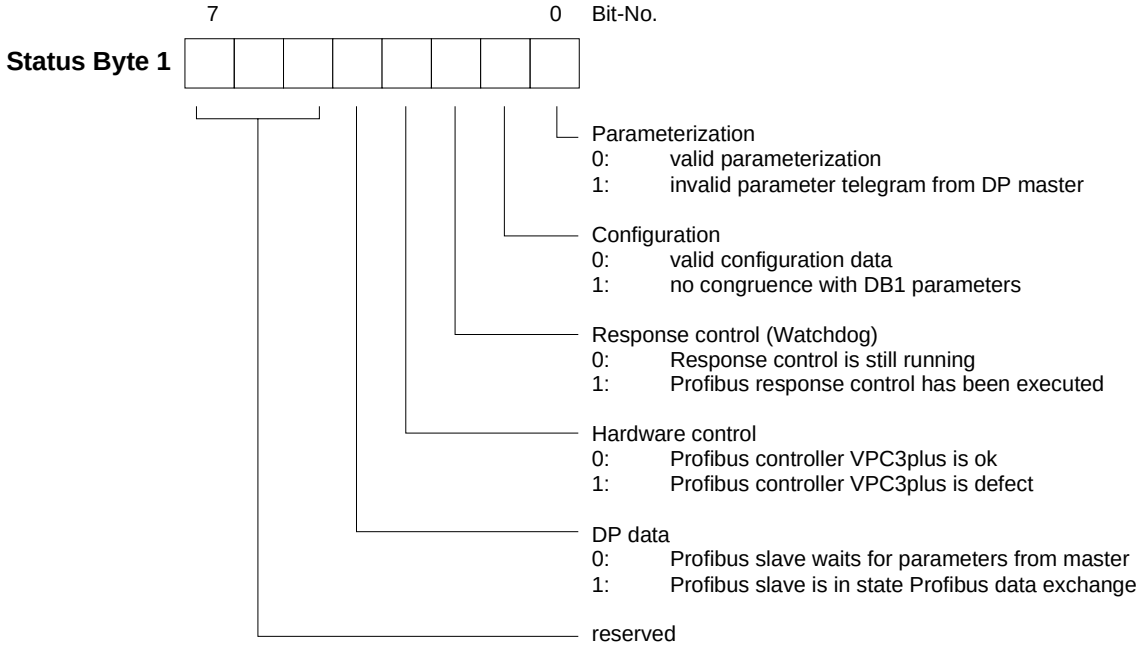
The diagnostic block of the CPU has the following structure:

Byte	Bit 7...Bit 0
0	diagnostic status byte: Bit 0: user specific diagnostic data 0: invalid diagnostic data 1: valid diagnostic data (starting a diagnosis) Bit 1: delete diagnostic 0: diagnostic deletion invalid 1: diagnostic deletion valid Bit 2...Bit 7: reserved
1...5	Bit 0...Bit 7: user specific diagnostic data equal to Byte 11...15 of equipment related diagnostic

Internal status messages to CPU

The current status of the Profibus communication procedure is obtainable from the status messages that are mapped into the peripheral addressing range of the CPU. Status messages consist of 2Byte with the following structure:





Parameter

Clear Data	The transmit and receive buffers are cleared when an error occurs.
reserved	These bits are reserved for future use.
User Parameter	Indicates validity of the parameter data. Parameter data is entered into the master configuration tool.
Response monitoring (active)	Indicates the status of the activation of the response monitor in the leading Profibus master. The slave will terminate communications when the response monitoring time is exceeded.
Profibus data exchange status	Status indicator for master communications. A bad configuration or bad parameters will terminate communications and the respective error is indicated by means of this bit.
Parameter configuration	Displays the status of the configuration data. The length of the configuration data and the number of parameter bytes is analyzed. The configuration is only accepted as being correct if these are equal and if no more than 31Byte of parameter data is transferred.
Configuration	Status indicator of the configuration data that are received from the Profibus master. You define the configuration by means of the master configuration tool.
Response monitoring (Watchdog)	Indicates the status of the response monitor in the Profibus master. This location contains an error when the response monitor has been activated and the required response time has been exceeded in the slave.
Hardware monitoring	This bit is set if the Profibus controller in the CPU 21xDP is defective. In this case you should contact the VIPA Hotline.
DP data	Any transfer error on the Profibus will set this error.

Profibus Installation guidelines

Profibus in general

- The VIPA Profibus-DP network must have a linear structure.
- Profibus-DP consists of at least one segment with a minimum of one master and one slave.
- A master must always be used in conjunction with a CPU.
- Profibus supports a max. of 125 stations.
- A max. of 32 stations are permitted per segment.
- The maximum length of a segment depends on the data transfer rate:

9.6...187.5kBaud	→	1000m
500kBaud	→	400m
1.5MBaud	→	200m
3...12MBaud	→	100m

- A maximum of 10 segments may be established. Segments are connected by means of repeaters. Every repeater is regarded as a station.
- All the stations communicate at the same baudrate. Slaves adapt automatically to the baudrate.
- The bus has to be terminated at both ends.
- Masters and slaves may be installed in any sequence.

Note!

When using optical participants, you should place a cover over the socket for the next station at the end of the bus, to avoid eye damage and to eliminate the chance of disturbance by external radiation. Use the rubber inserts for this purpose by inserting them into the two remaining openings of the FO-connector.

Assembly and installation into Profibus

- Assemble your Profibus system complete with the respective modules.
- Set the address of your bus coupler to an unused address.
- Transfer the GSD-file supplied with the modules into your configuration system and configure the system.
- Transfer the configuration into the master.
- Connect the Profibus cable to the coupler and turn the power supply on.

Note!

The Profibus cable has to be terminated with a terminating resistor of the characteristic impedance of the cable. Please ensure to install a terminating resistor at the last station on the bus.

Transfer medium

Profibus employs a screened twisted two-core cable as communication medium, which is based on the RS485 interface.

The RS485 interface employs differential voltages. For this reason it is less sensitive to electrical interference than a voltage or a current based interface. You may configure networks with linear as well as with tree geometry.

Your VIPA CPU 21xDP carries a 9pin socket. You connect the Profibus coupler directly to your Profibus network as a slave by means of this connector.

Every segment supports a maximum of 32 stations. Different segments are connected by means of repeaters. That maximum length of a segment depends on the data communication rate.

The rate of data transfer of a Profibus-DP link is set to a value between 9.6kBaud and 12MBaud. Slaves are configured automatically. All the stations on the network communicate at the same baudrate.

The structure of the bus is such that stations may be inserted or removed without repercussions or to commission the system in different stages. Extensions to the network do not influence those stations that have already been commissioned. New stations or stations that have failed are detected automatically.

Profibus using RS485

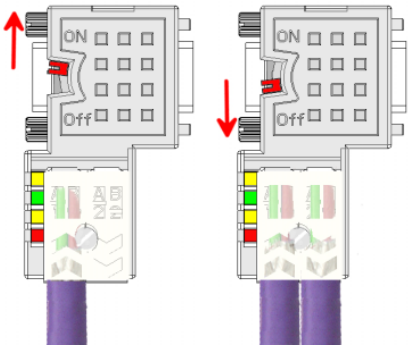
Profibus employs a screened twisted pair cable based on RS485 interface specifications as the data communication medium.

Note!

The Profibus line must be terminated with ripple resistor. Please ensure that the last participant the line is terminated by means of a terminating resistor.

Termination

The "EasyConn" bus connector is provided with a switch that is used to activate a terminating resistor.



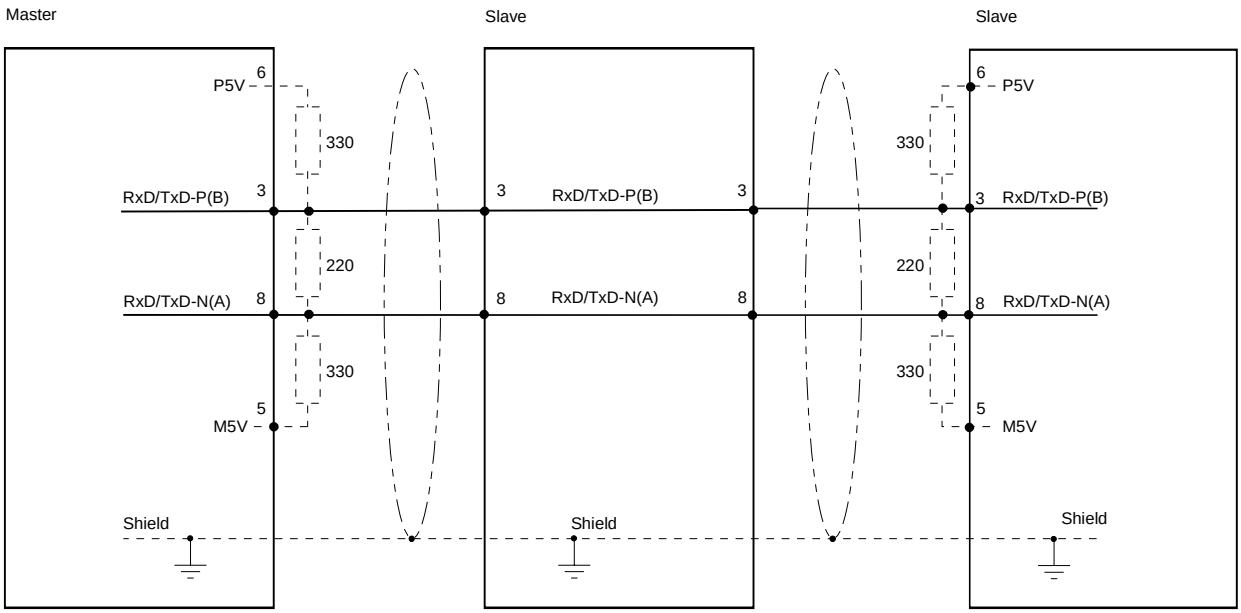
Attention!

The terminating resistor is only effective, if the connector is installed at a slave and the slave is connected to a power supply.

Note!

A complete description of installation and deployment of the terminating resistors is delivered with the connector.

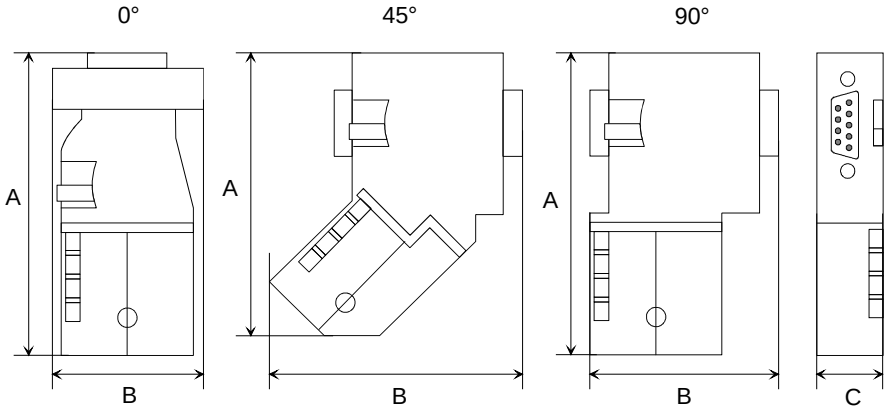
The following picture illustrates the terminating resistors of the respective start and end station.



Bus connector

In systems with more than two stations all partners are wired in parallel. For that purpose, the bus cable must be feed-through uninterrupted.

Via the order number VIPA 972-0DP10 you may order the bus connector "EasyConn". This is a bus connector with switchable terminating resistor and integrated bus diagnostic.



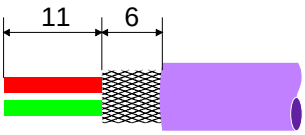
	0°	45°	90°
A	64	61	66
B	34	53	40
C	15,8	15,8	15,8

all in mm

Note!

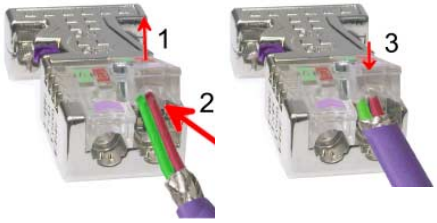
To connect this plug, please use the standard Profibus cable type A with solid wire core according to EN50170.

Under the order no. 905-6AA00 VIPA offers the "EasyStrip" de-isolating tool, that makes the connection of the EasyConn much easier.



all in mm

Connecting the profibus cable



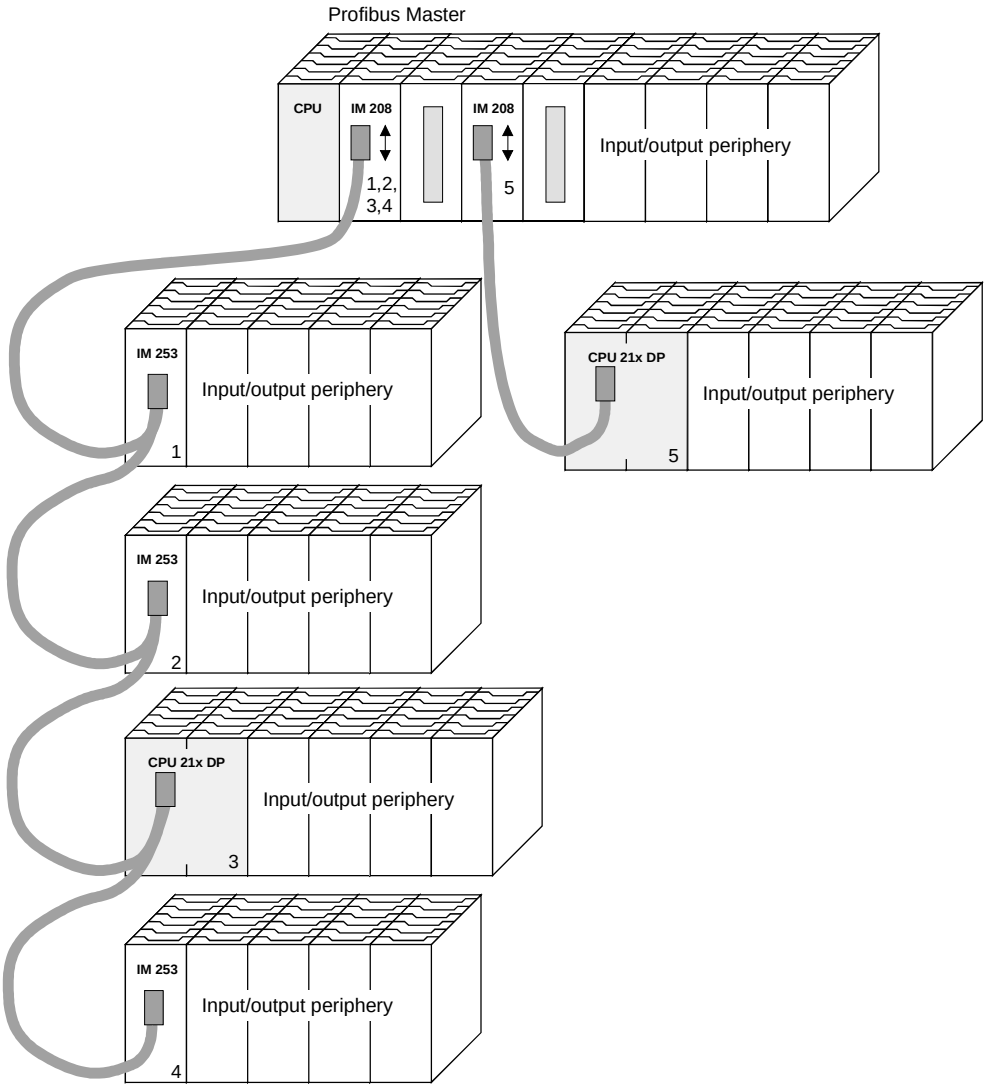
- Loosen the screw
- Lift contact-cover
- Insert both wires into the ducts provided (watch for the correct line color as below!)
- Please take care not to cause a short circuit between screen and data lines!
- Close the contact cover
- Tighten screw
  (max. tightening torque 4NM)

Please note: **the green line must be connected to A, the red line to B!**

Examples for Profibus networks

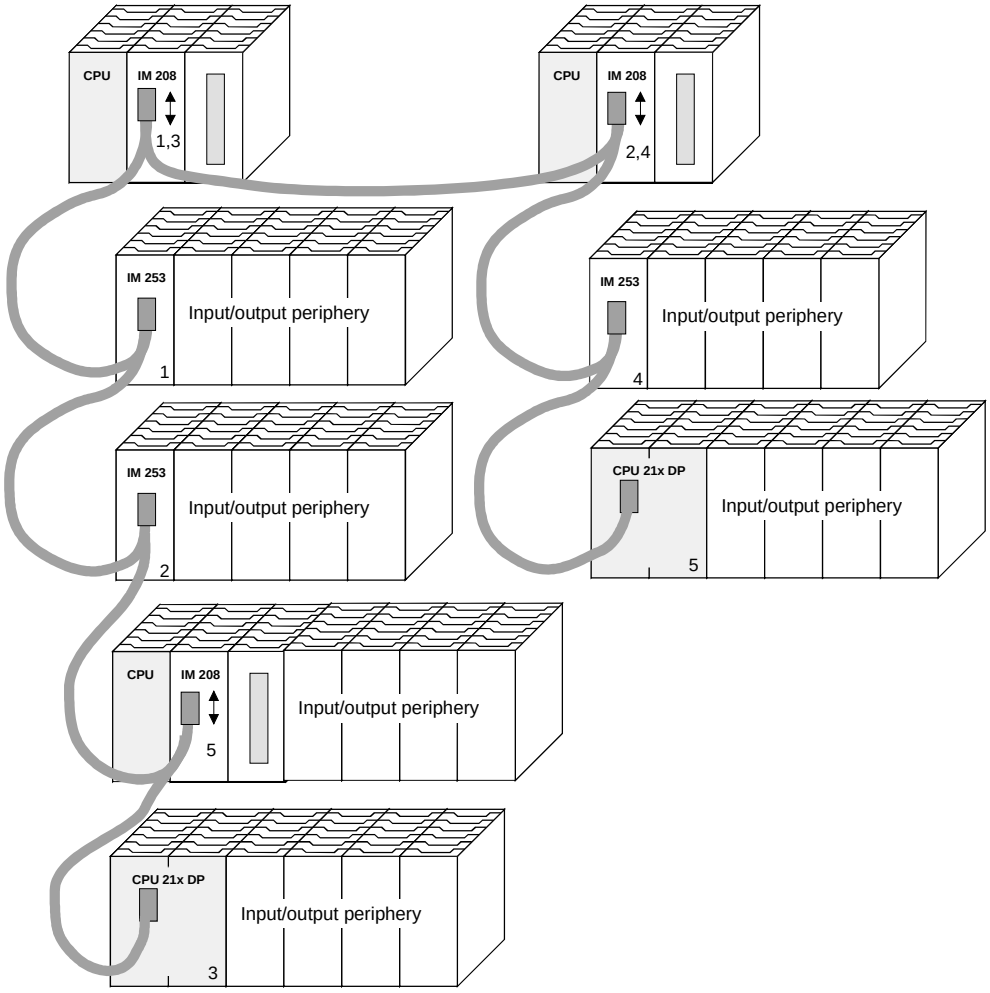
One CPU and several master interfaces

The CPU should have a short cycle time to ensure that the data of slave No.5 (at the right) is always up to date. This scheme is only viable if the slower line (at the left) is connected to slaves that do not require up to date data. This portion of the line should also not be connected to modules that issue alarms.



Multi master system

More than one master and multiple slaves connected to one bus:



Commissioning

Outline

- Install the CPU 21xDP.
- Configure the CPU 21xDP in your master system.
- Configure the I/O periphery that is connected to the backplane bus.
- Connect the CPU 21xDP to your Profibus.
- Turn on the power supply.
- Transfer your project to the CPUs.

Installation

Assemble the CPU 21xDP with the required peripheral modules. Do not exceed the maximum current capacity of your power supply.

Note!

The bus cable has always to be terminated with the ripple resistor to avoid reflections and therefore communication problems!

Configuration in the master system

Configure your CPU 21xDP in your master system. You may use the VIPA WinNCS package for this purpose. To configure the System 200V Profibus slave modules from VIPA, you need to include the according GSD-file.

Configuration CPU 21xDP and I/O periphery

The System 200V peripheral modules that are connected with the CPU 21xDP directly via the backplane bus are automatically mapped into the CPU address area. You may alter the address allocation in the hardware configurator from Siemens at any time.

Voltage supply

The CPU 21xDP has an integrated power supply that has to be provided with DC 24V.

Via the voltage supply not only the CPU is provided with voltage but also the bus coupler and, via the backplane bus, the peripheral modules. Please regard, that the integrated power supply may provide the backplane bus with max. 3A.

Profibus and backplane bus are galvanically separated from each other.

Transfer project

The transfer of the hardware configuration into the CPU takes place via MPI.

- Connect your PG res. the PC via MPI with the CPU.
  If your programming device has no MPI slot, you may use the VIPA Green Cable to establish a serial point-to-point connection.

  The Green Cable has the order no. VIPA 950-0KB00 and only be used with the VIPA CPUs with MP^{2}I interface.
- Configure the MP interface of your PC.
- With **PLC** > *Load to module* in your projecting tool, you transfer your project into the CPU.
- For the additional security copy of your project on MMC, you plug-in a MMC and transfer the user application to the MMC via **PLC** > *Copy RAM to ROM*.
  During write operation the MC-LED on the CPU blinks. Due to the system, the successful writing is signalized too soon. The write command has only been completed, when the LED extinguishes.

Attention!

Please regard the hints for deploying the Green Cable and the MP^{2}I jack in chapter 1.

Initialization phase

The Profibus coupler executes a self-test routine after it is powered on. On this occasion it checks the internal operations, the backplane bus communication and the Profibus communication.

When the test is completed successfully, the parameters of the CPU are read and the Profibus slave parameters are checked.

After the successful test, the bus coupler status changes to "READY".

If the bus coupler detects communication errors on the backplane bus, its status first changes to STOP and after a delay of 2 seconds it is initialized again. As soon as the test is completed successfully, the RD-LED blinks.

The DE-LED is turned on when communication is started.

Example

Objective

This example is intended to show the communication between a master CPU 214DPM and a slave CPU 214DP.

The counters are to communicate via the Profibus and to be displayed at the output module of the respective partner.

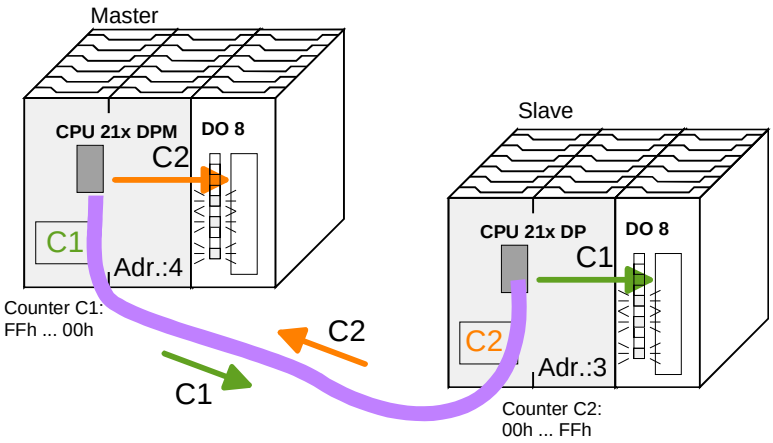
Detailed description of the objective

The CPU 214DPM shall count from FFh...00h and transfer the count cyclically into the output area of the Profibus master. The master has then to transfer this value to the slave of the CPU 214DP.

The value received shall be saved in the peripheral input area of the CPU and monitored at the output module (at address 0) via the backplane bus.

On the other hand, the CPU 214DP should count from 00h to FFh. This count shall also be saved in the output area of the CPU slave and be transferred to the master via Profibus.

This value shall be monitored at the output module (address 0) of the CPU 214DPM.



Configuration data

CPU 21xDPM

Counter: MB 0 (FFh...00h)

Profibus address:	4	
Input area:	Address 10	length: 2 Byte
Output area:	Address 20	length: 2 Byte

CPU 21xDP

Counter: MB 0 (00h...FFh)

Profibus address:	3	
Input area:	Address 30	length: 2 Byte
Output area:	Address 40	length: 2 Byte
Parameter data:	Address 50	length: 24 Byte (fixed)
Diagnostic data:	Address 60	length: 6 Byte (fixed)
Status data:	Address 100	length: 2 Byte (fixed)

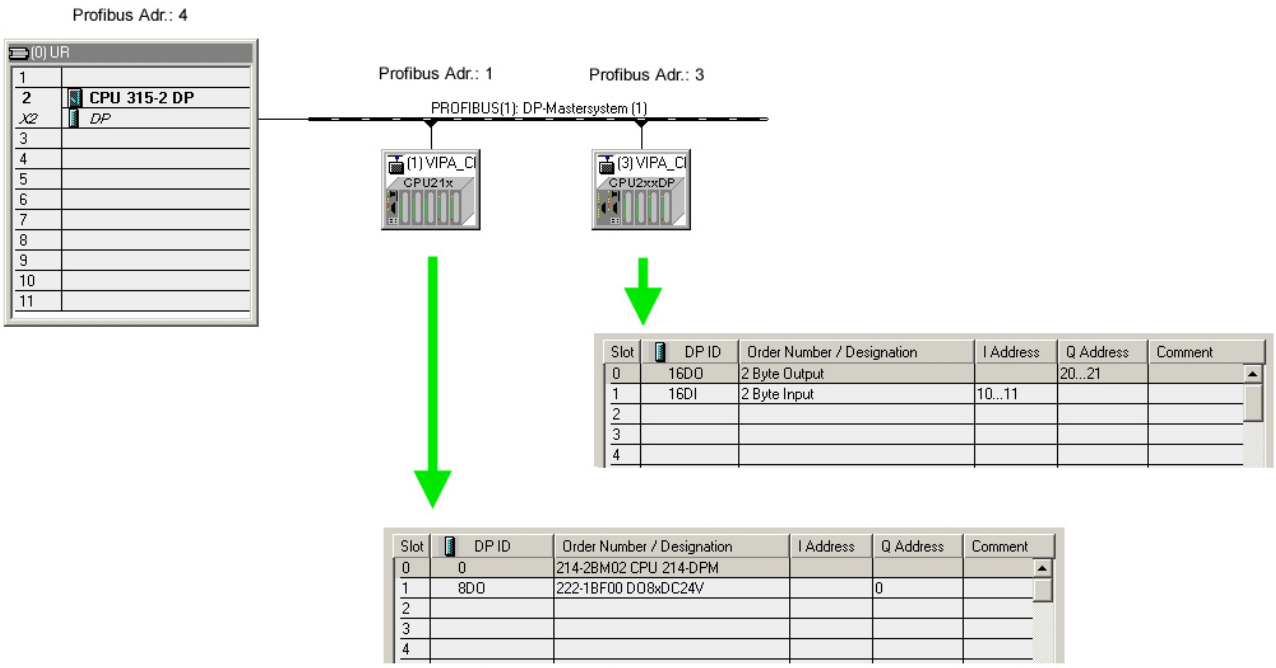
Configuration of CPU 21xDPM

To be compatible to the SIMATIC manager from Siemens, you have to execute the following steps for the System 200V:

- Start the hardware configurator from Siemens.
- Install the GSD-Datei VIPA_21x.gsd.
- Configure a CPU 315-2DP with DP master system (address 4).
- Add a Profibus slave **"VIPA_CPU21x"** at address 1.
- Include the CPU **214-2BM02** at the 1st slot of the slave system.
- Include the output module 222-1BF00.

For linking-up the CPU 21xDP you have to execute the following steps after including the GSD-file (VIPA04d5.gsd):

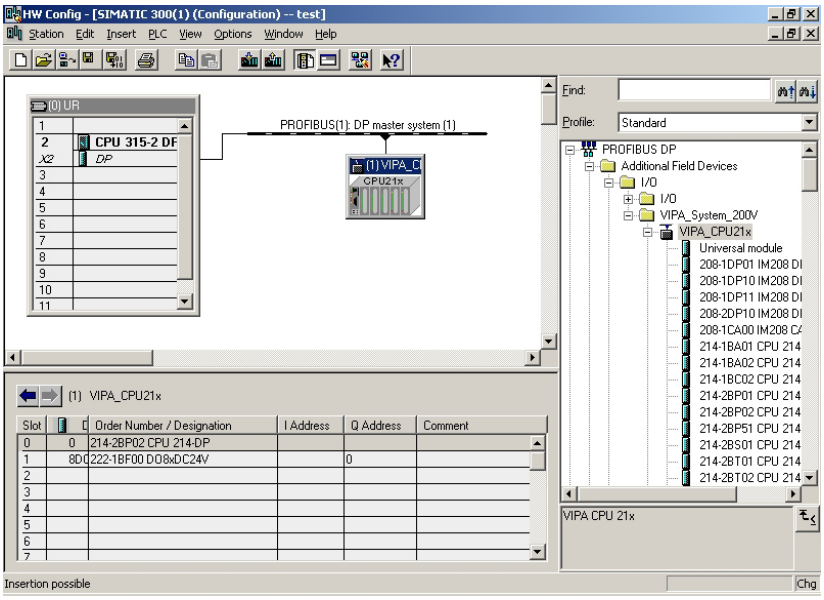
- Add the Profibus slave **"VIPA_CPU2xxDP"** (address 3). The DP slave is in the hardware catalog under:
  Profibus-DP > Additional field devices > I/O > VIPA_System_200V > VIPA_CPU2xxDP.
- Assign memory areas of the CPU to the in- and output of the Profibus-DP master section in form of Byte blocks. For this, you have to include the "2Byte output" element on the 1st slot and select the output address 20. Include the "2Byte input" element on the next slot and select the input address 10.
- Save your project.



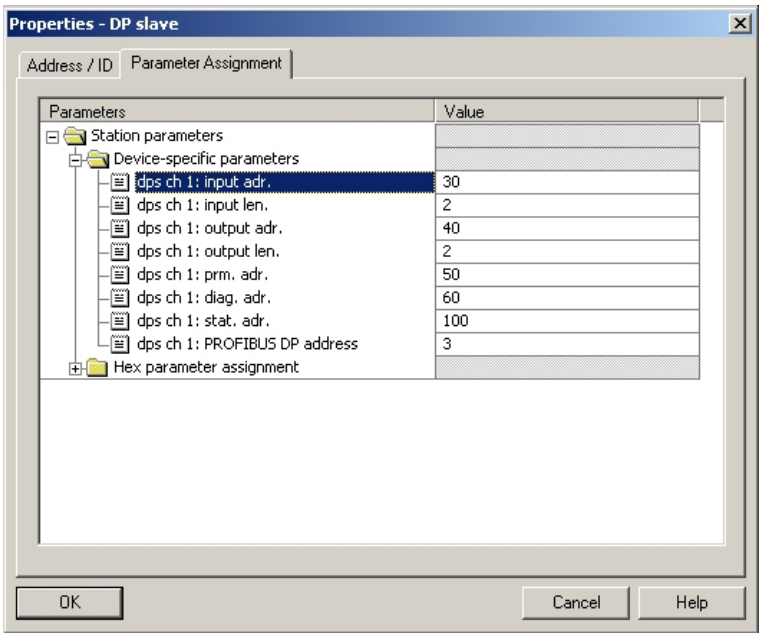
Configuration CPU 21xDP

To be compatible to the SIMATIC manager from Siemens, you have to execute the following steps for the System 200V:

- Start the hardware configurator from Siemens.
- Configure a CPU 315-2DP with DP master system (address 2).
- Add a Profibus slave *"VIPA_CPU21x"* at address 1.
- Include the CPU **214-2BP02** at the 1st slot of the slave system.
- Include at the next slot the output module 222-1BF00.



- Choose the following parameters in the parameter window of the CPU 214-2BP02:



- Save your project.

Application program in the CPU 214DPM

The application program in the CPU 214DPM has two tasks that are handled by two OBs:

- to test communications by means of the control byte.
  Load the input byte from Profibus and transfer the value to the output module.

  OB 1 (cyclic call)

```
L    B#16#FF
T    AB  20             Control byte for Slave-CPU

L    B#16#FE            Load control value 0xFE
L    EB  10             Was the control byte transferred
<>I                     correctly from the slave CPU?
BEB                     No -> End

                        ------------------------------
                        Data exchange via Profibus

L    EB  11             Load input byte 11 (output data
                        of the CPU214DP) and
T    AB   0             transfer to output byte 0

BE
```

- Read counter value from MB 0, decrement, save in MB0 and put it out to CPU 214DP via Profibus.

  OB 35 (timer-OB)

```
L    MB   0             Counter from 0xFF to 0x00
L    1
-I
T    MB   0

T    AB  21             Transfer into output byte 21
                        (input data of CPU214DP)
BE
```

At this point the programming of the CPU 214DPM is completed. The Profibus communication has also been defined for both sides.

Transfer your project into the CPU 214DPM via MPI by means of the **PLC**-functions.

Application program in the CPU 214DP

As shown above, the application program in the CPU has two tasks that are handled by two OBs:

- Load the input byte from the Profibus slave and transfer the value to the output module.

OB 1 (cyclic call)

```
L     PEW 100           Load status data and save in
T     MW 100            flag word

UN    M  100.5          Commissioning by the DP master
BEB                     occurred? No -> End

U     M  101.4          Valid receive data?
BEB                     No -> End
L     B#16#FF           Load control value and compare
L     PEB  30           to control byte
<>I                     (1st input byte)
BEB                     Received data does not contain
                        valid values

L     B#16#FE           Control byte for master-CPU
T     PAB  40
                        -------------------------------
                        Data exchange via Profibus

L     PEB  31           Load peripheral byte 31 (input
                        data from Profibus slave) and
T     AB   0            transfer into output byte 0

BE
```

- Read counter value from MB 0, increment, save into MB0 and put it out to CPU 21x via Profibus.

OB 35 (timer-OB)

```
L     MB   0            Counter from 0x00 to 0xFF
L     1
+I
T     MB   0

T     PAB  41           Transfer counter value into
                        peripheral byte 41 (output data
                        of the Profibus slave)
BE
```

Chapter 8 Deployment CPU 21xCAN

Overview

Content of this chapter is the Deployment of the CPU 21xCAN under CANopen. Here you'll find all information required for the usage of the integrated CAN master.

Below follows a description of:

- CAN-Bus principles
- Project engineering CAN master and CPU
- Course diagram after POWER ON
- Process image for RPDOs and TPDOs
- Telegram structure and overview over the supported objects
- Device model with description of PDOs and SDOs
- Objects directory

Contents

Principles CAN-Bus

General

The CAN-Bus (**C**ontrol **A**rea **N**etwork) is an international standard for open fieldbus systems intended for building, manufacturing and process automation applications that was originally designed for automotive applications.

Due to its extensive error detection facilities, the CAN-Bus system is regarded as the most secure bus system. It has a residual error probability of less than $4.7x10^{-11}$. Bad messages are flagged and retransmitted automatically.

In contrast to Profibus and INTERBUS-S, CAN defines under the CAL-level-7-protocol (CAL=**C**AN **a**pplication **l**ayer) defines various level-7 user profiles for the CAN-Bus. One standard user profile defined by the CIA (**C**AN **i**n **A**utomation) e.V. is CANopen.

CANopen

CANopen is a user profile for industrial real-time systems, which is currently supported by a large number of manufacturers. CANopen was published under the heading of DS-301 by the CAN in Automation association (CIA). The communication specifications DS-301 define standards for CAN devices. These specifications mean that the equipment supplied by different manufacturers is interchangeable. The compatibility of the equipment is further enhanced by the equipment specification DS-401 that defines standards for the technical data and process data of the equipment. DS-401 contains the standards for digital and analog input/output modules.

CANopen comprises a communication profile that defines the objects that must be used for the transfer of certain data as well as the device profiles that specify the type of data that must be transferred by means of other objects.

The CANopen communication profile is based upon an object directory that is similar to the profile used by Profibus. The communication profile DS-301 defines two standard objects as well as a number of special objects:

- Process data objects (PDO)
  PDOs are used for real-time data transfers
- Service data objects (SDO)
  SDOs provide access to the object directory for read and write operations

Communication medium

CAN is based on a linear bus topology. You can use router nodes to construct a network. The number of devices per network is only limited by the performance of the bus driver modules.

The maximum distance covered by the network is determined by the runtimes of the signals. This means that a data rate of 1Mbaud limits the network to 40m and 80kBaud limits the network to 1000m.

The CAN-Bus communication medium employs a screened three-core cable (optionally a five-core).

The CAN-Bus operates by means of differential voltages. For this reason it is less sensitive to external interference than a pure voltage or current based interface. The network must be configured as a serial bus, which is terminated by a 120Ω terminating resistor.

Your VIPA CAN-Bus coupler contains a 9pin socket. You must use this socket to connect the CAN-Bus coupler as a slave directly to your CAN-Bus network.

All devices on the network use the same baud rate.

Due to the bus structure of the network it is possible to connect or disconnect any station without interruption to the system. It is therefore also possible to commission a system in various stages. Extensions to the system do not affect the operational stations. Defective stations or new stations are recognized automatically.

Bus access method

Bus access methods are commonly divided into controlled (deterministic) and uncontrolled (random) bus access systems.

CAN employs a Carrier-Sense Multiple Access (CSMA) method, i.e. all stations have the same right to access the bus as long as the bus is not in use (random bus access).

Data communications is message related and not station related. Every message contains a unique identifier, which also defines the priority of the message. At any instance only one station can occupy the bus for a message.

CAN-Bus access control is performed by means of a collision-free, bit-based arbitration algorithm. Collision-free means that the final winner of the arbitration process does not have to repeat his message. The station with the highest priority is selected automatically when more than one station accesses the bus simultaneously. Any station that is has information to send will delay the transmission if it detects that the bus is occupied.

Project engineering of the CPU 21xCAN

Overview

The project engineering of the CANopen master happens in WinCoCT (**Win**dows **C**AN**o**pen **C**onfiguration **T**ool) from VIPA. You export your project from WinCoCT as wld-file. This wld-file can then be imported into the hardware configurator from Siemens.

Create a virtual Profibus system "VIPA_CPU21x" and include the CPU21xCAN (VIPA 21x-2CM02) at plug-in location 0.

Fast introduction

For the deployment of System 200V modules and the CAN master, you have to include the System 200V modules into the hardware catalog via the GSD-file from VIPA. For the project engineering in the hardware configurator you have to execute the following steps:

- Start WinCoCT and project the CANopen network.
- Create a master group with  and insert a CANopen master via  .
- Activate the master function via "Device Access" and "Device is NMT Master".
- Activate in the register "CANopen Manager" Device is NMT Master and confirm your entry.
- Set parameters like diagnosis behavior and CPU address ranges with "Set PLC Parameters".
- Create a "slave" group with  and add your CANopen slaves via  .
- Add modules to your slaves via "Modules" and parameterize them if needed.
- Set your process data connections in the matrix via "Connections" and proof your entries if needed in the process image of the master.
- Save the project and export it as wld-file.
- Include vipa_21x.gsd in the hardware configurator from Siemens.
- Switch to the SIMATIC manager from Siemens and copy the data block from the CAN-wld-file into the block directory.
- Project the Profibus-DP master system in the hardware configurator with the following Siemens-CPU: CPU 315-2DP (6ES7 315-2AF03-0AB0 V1.2)
- The DP master receives an address >1.
- Add the System 200V DP slave system "VIPA_CPU21x" from the hardware catalog to the master system.
- The slave system always requires the address 1.
- Place the System 200V modules in plugged sequence starting with the CPU 21xCAN at the 1st slot.
- Save all and transfer the PLC project together with the wld-file via MPI into the CPU.

In the following, these steps are explained more detailed.

Precondition for the project engineering

The hardware configurator is a part of the SIMATIC manager from Siemens. It serves the project engineering. The modules that can be parameterized with are monitored in the hardware catalog.

For the deployment of the System 200V modules, the inclusion of the System 200V modules into the hardware catalog is necessary. This happens via a GSD-file vipa_21x.gsd from VIPA.

Note!

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

Include GSD-file

- Copy the delivered VIPA GSD-file VIPA_21x.gsd into your GSD-directory... \siemens\step7\s7data\gsd
- Start the hardware configurator from Siemens
- Close all projects
- Choose **Options** > *Install new GSD-file*
- Select **VIPA_21x.GSD**

Now the modules of the System 200V from VIPA are integrated in the hardware catalog and can be projected.

Note

To be compatible to the SIMATIC manager from Siemens, the System 200V CPUs from VIPA have to be projected as

CPU 315-2DP (6ES7 315-2AF03-0AB0 V1.2)!

To be able to directly address the modules, you have to include them in the hardware configurator from Siemens in form of a virtual Profibus system. By including the GSD-file from VIPA, you are able to access the complete function range of the modules.

Engineer the CAN master in your virtual Profibus system by placing a CPU 21xCAN on the 1st slot.

The concrete project engineering happens in the CANopen configuration tool WinCoCT. You may export your project as wld-file and transfer it as DB into your PLC program.

WinCoCT

WinCoCT (**Win**dows **C**AN**o**pen **C**onfiguration **T**ool is a configuration tool developed from VIPA to allow the comfortable project engineering of CANopen networks.

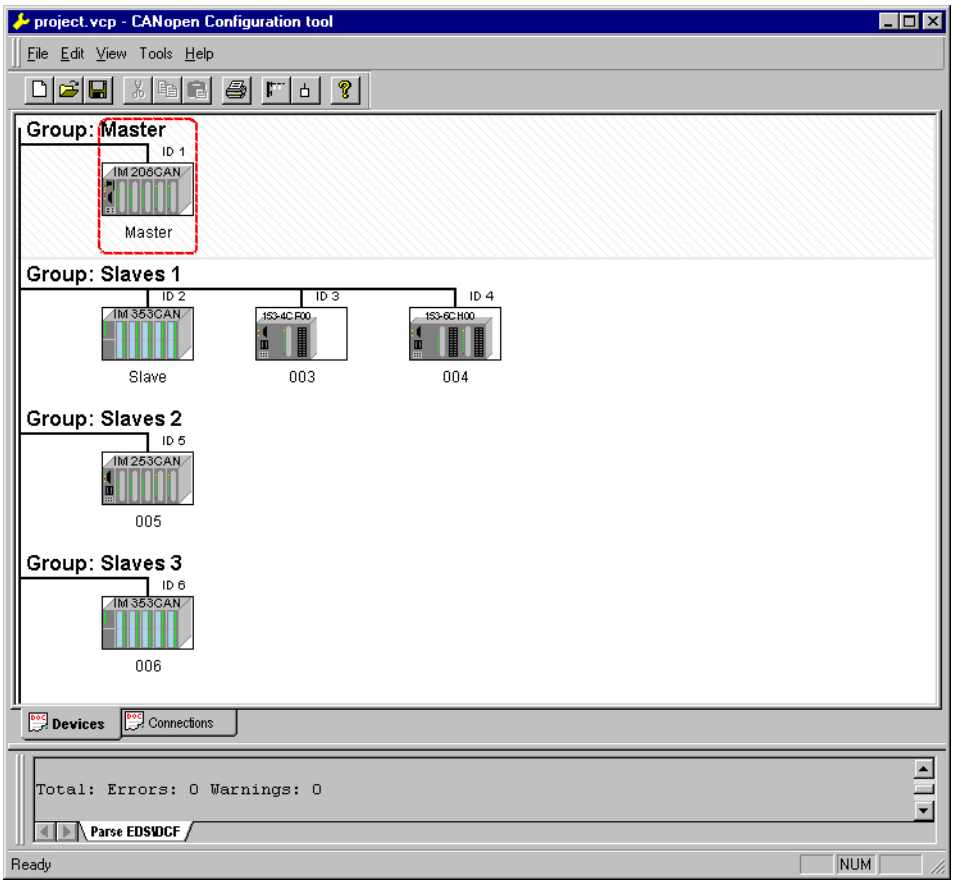
WinCoCT monitors the CANopen network topology in a graphical user interface. Here you may place, parameterize and group field devices and controls and engineer connections.

The selection of the devices happens via a list that can be extended for your needs with an EDS-file (**E**lectronic **D**ata **S**heet) at any time.

A right click onto a device opens a context menu consisting partly of static and partly of dynamic components.

For the configuration of the process data exchange, all process data are monitored in a matrix with the device inputs as rows and the device outputs as columns. Mark a cross point to create the wanted connection.

The telegram collection and optimization is executed by WinCoCT.



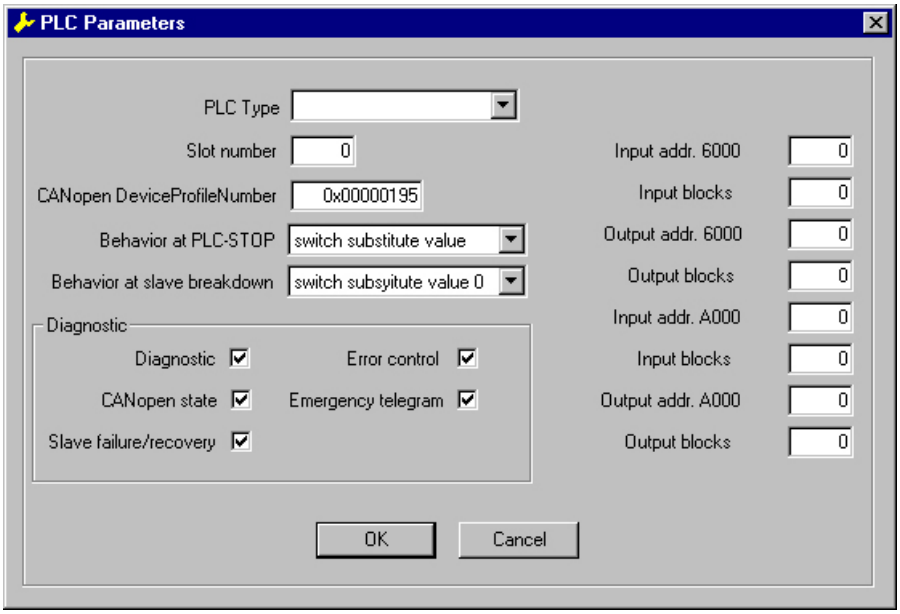
Set project parameters

Via **Tools** > *Project options* you may preset CAN specific parameters like baud rate, selection of the master etc.

More detailed information is to find in the WinCoCT manual.

Parameter CAN master

WinCoCT allows you to preset VIPA specific parameters for the CAN master by doing a right click onto the master and call the following dialog window with Set PLC-Parameters:



PLC Type

Reserved for later extensions

Slot number

Plug-in location no. at the bus

0: For the addressing of the CAN master integrated in the CPU

1 ... 32: For the addressing of CAN master at the standard bus

CANopen DeviceProfileNumber

Fix at 0x195

Behavior at PLC-STOP

Here you can define the reaction of the output channels if the CPU switches to STOP. The following values are available:

Switch substitute value 0: Sets all outputs to 0

Keep last value: Keeps the recent state of the outputs.

Behavior at Slave breakdown

Here you set the reaction for the slave input data in case of a slave failure.

Switch substitute value 0: The data is set to 0.

Keep the last value: The recent date remain unchanged.

Diagnostic

This area allows you to define the diagnostic reaction of the CAN master.

Diagnostic: Activates the diagnostic function

CANopen state: When activated, the CAN master sends its state "preoperational" or "operational" to the CPU. You may request the state via SFC 13.

Slave failure/recovery: When activated, the OB 86 is called in the CPU in case of slave failure and reboot.

Error control: If this option is selected, the NMT master sends all Guarding errors as diagnosis to the CPU, that calls the OB 82.

Emergency Telegram: At activation, the NMT master sends all Emergency telegrams as diagnosis to the CPU, that calls the OB 82.

Address range in the CPU

The following fields allow you to preset the address ranges in the CPU for the CANopen master in- and output ranges. Each block consists of 4Byte.

Input addr. 6000, Input blocks

PI basic address in the CPU that are occupied from 0x6000 CAN input data. For input blocks max. 16 (64Byte) can be entered.

Output addr. 6000, Output blocks

PO basic address in the CPU that are occupied from 0x6000 CAN output data. For output blocks max. 16 (64Byte) can be entered.

Input addr. A000, Input blocks

PI basic address in the CPU that are occupied from 0xA000 CAN input network variables. For input blocks max. 80 (320Byte) can be entered.

Output addr. A000, Output blocks

PO basic address in the CPU that are occupied from 0xA000 CAN output network variables. For output blocks max. 80 (320Byte) can be entered.

Activate CANopen slave in the CANopen Manager

To enable the master to access a CANopen slave, you have to register it at the according master via WinCoCT. Right click onto your CAN master, choose "Device access" and switch to the register "CANopen Manager".

Via [Change] you can register every single slave res. via [Global] all slaves at your master and preset the error behavior.

Please don’t forget to apply the settings into your project engineering by clicking on [Apply to slaves].

Steps of the project engineering

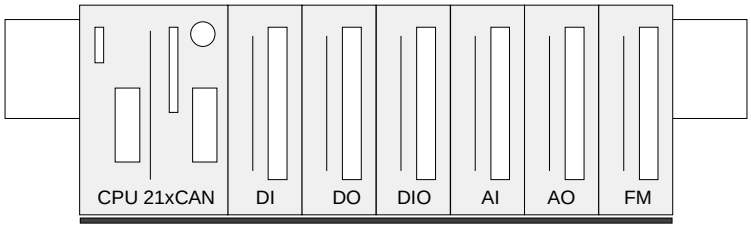
The following text describes the approach of the project engineering with an abstract sample:

The project engineering is divided into three parts:

- CAN master project engineering in WinCoCT and export as wld-file
- Import CAN master project engineering
- Project engineering of the CPU 21xCAN an the System 200V modules

Hardware structure

System 200V

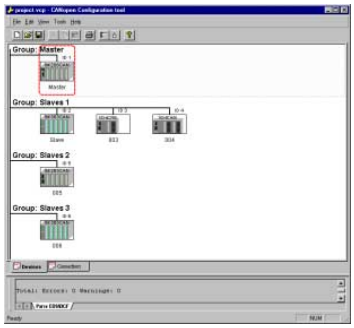


Preconditions

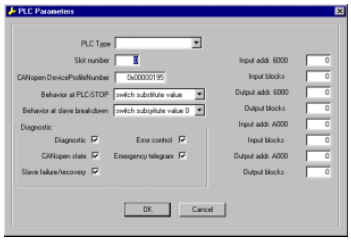
For the project engineering of a CANopen system, the most recent EDS-file has to be transferred into the EDS-directory of WinCoCT.

For the deployment of the System 200V modules, you have to include the System 200V modules with the GSD-file VIPA_21x.gsd from VIPA into the hardware catalog.

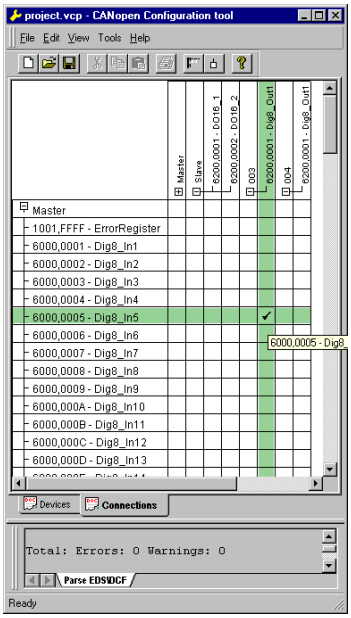
CAN master project engineering in WinCoCT



- Copy the required EDS-files into the EDS-directory and start WinCoCT.
- Create a "master" group via  and insert a CANopen master via  (VIPA_21x_2CM02.eds).
- Create a "slave" group with  and add your CANopen slaves via  .
- Right click on the according slave and add the needed modules via „Modules“.
- Parameterize the modules with [Parameter] res. via the according object directory.
- Right click on the master and open the dialog "Device Access".
- Activate Device is NMT Master in the register "CANopen Manager" and register the according slaves at the master. Don’t forget to apply your settings into your project engineering with [Apply to slaves]!



- Right click onto the master and open the VIPA specific dialog "Set PLC Parameters". Here you may adjust the diagnosis behavior and the address ranges that the master occupies in the CPU.
  Under "Slot number" type the slot no. 0 for your CPU 21xCAN. At export, WinCoCT creates the DB 2000.



- Change to the register "Connections" in the main window. Here the process data are shown in a matrix as inputs (1st column) and as outputs (1st row).
  To monitor the process data of a device with a "+" click on the according device.
- For helping you, you may only define a connection when the appearing cross has green color. Select the according cell with the mouse pointer in row and column in the matrix and click on it. → The cell is marked with a "✓". You can control the connection by changing into "Devices", click on the master and monitor the process image of the master via "Device Access".
- Save your project.
- Via **File** > *Export* your CANopen project is exported into a wld-file. The name is the combination of project name + node address + ID **Ma**ster/**Sl**a**v**e.

Now your CANopen project engineering under WinCoCT is ready.

Import into PLC program

- Start the SIMATIC manager from Siemens with your PLC project for the CPU 21xCAN.
- Open the wld-file via **File** > *Memory Card File* > *open*
- Copy the DB 2000 into your block directory.

As soon as you transfer this block to the CPU, it is recognized by the CPU and the according parameters are transferred to the CAN master.

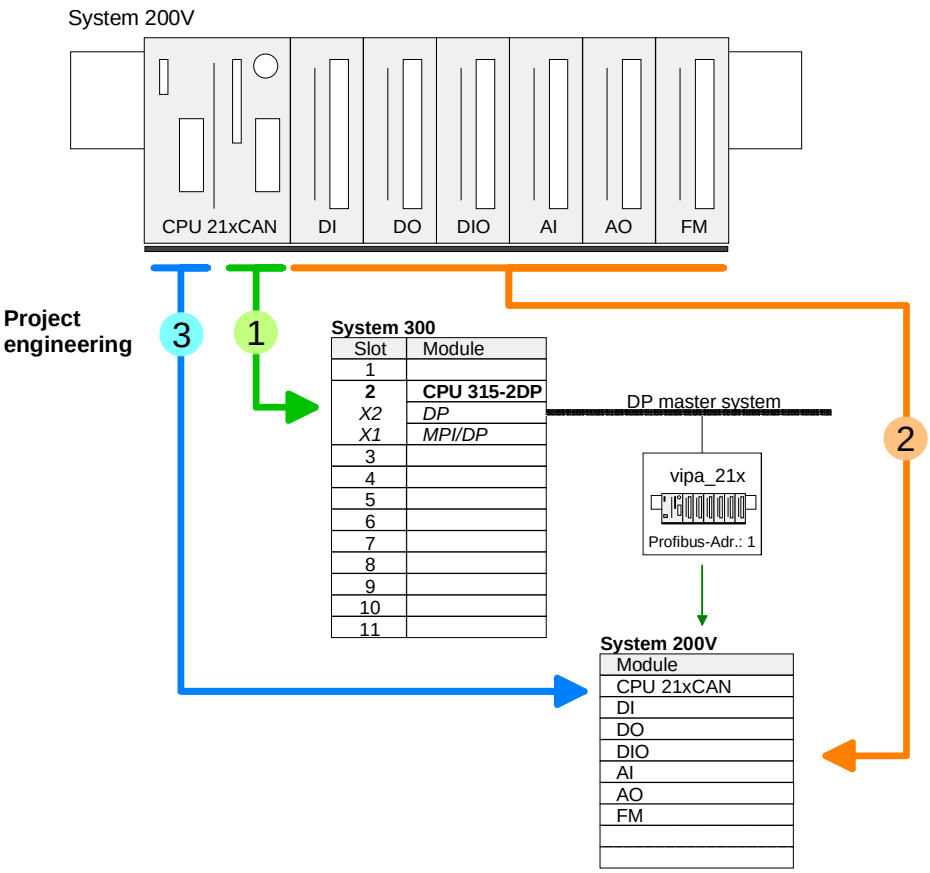
This is only possible if your CAN master CPU is included in the hardware configuration as virtual Profibus system. The approach is to find at the following pages.

Hardware configuration CPU 21xCAN and System 200V modules

The hardware configuration of the System 200V has the following approach:

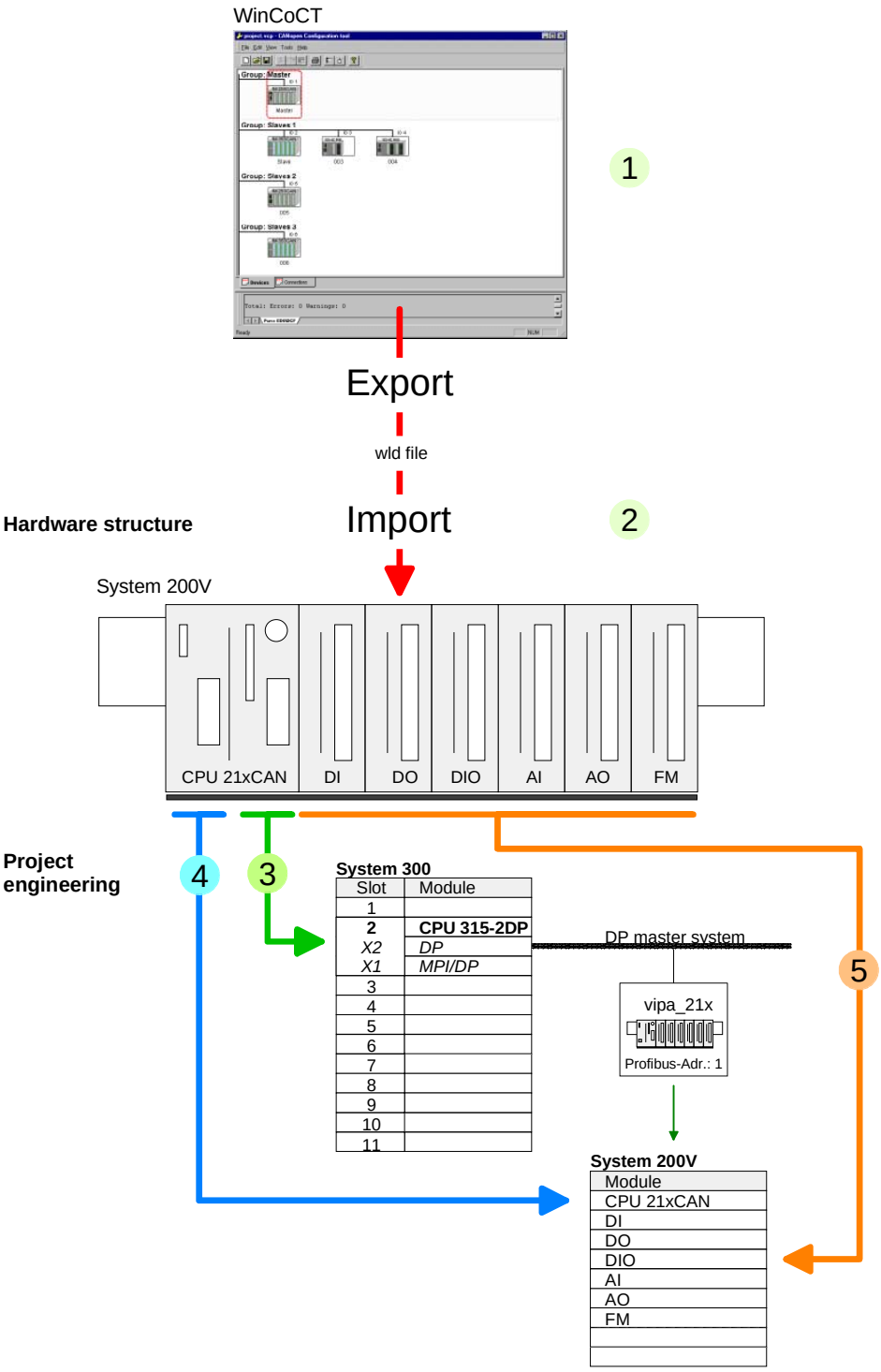
- Start the hardware configurator from Siemens with a new project and add a profile rail from the hardware catalog.
- Add the CPU 315-2DP (6ES7 315-2AF03-0AB0 V1.2). Create a new Profibus subnet for that.
- Add the System "VIPA_CPU21x" to the subnet. This is to find in the hardware catalog under *PROFIBUS DP > Additional field devices > IO >* VIPA_System_200V. Assign the Profibus address 1 to this module.
- Place the CPU 21xCAN at the 1st slot from the hardware catalog in your configurator.
- Include your System 200V modules in the plugged sequence.
- If needed, parameterize the CPU res. the modules. The parameter window opens with a double click on the according module.
- Save your project.

Hardware strukture

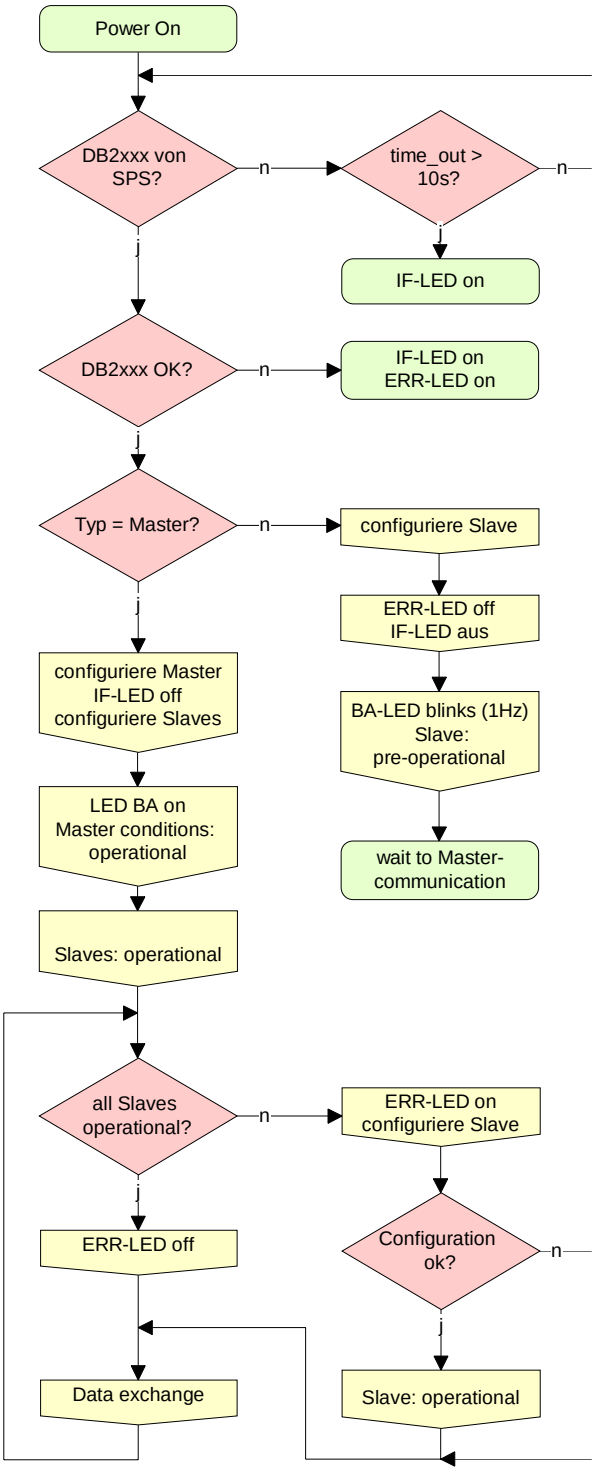


Conclusion

The following picture shows the conclusion of the engineering steps:



Modes



STOP → RUN (automatically)

After POWER ON and at valid project data in the CPU, the master switches automatically into RUN. The master has no operating mode lever.

After POWER ON, the project data is automatically send from the CPU to the CAN master. This establishes a communication to the CAN slaves.

At active communication and valid bus parameters, the CAN master switches into the state "operational". The LEDs RUN and BA are on.

At invalid parameters, the CAN master remains in STOP and shows the parameterization error via the IF-LED.

RUN

In RUN, the RUN- and BA-LEDs are on. Now data can be exchanged.

In case of an error, like e.g. slave failure, the ERR-LED at the CAN master is on and an alarm is send to the CPU.

Process image of the CPU 21xCAN

The process image is build of the following parts:

- Process image for input data (PI) for RPDOs
- Process image for output data (PO) for TPDOs

Every part consists of 64Byte "Digital-Data"- and 320Byte "Network Variables".

Input data

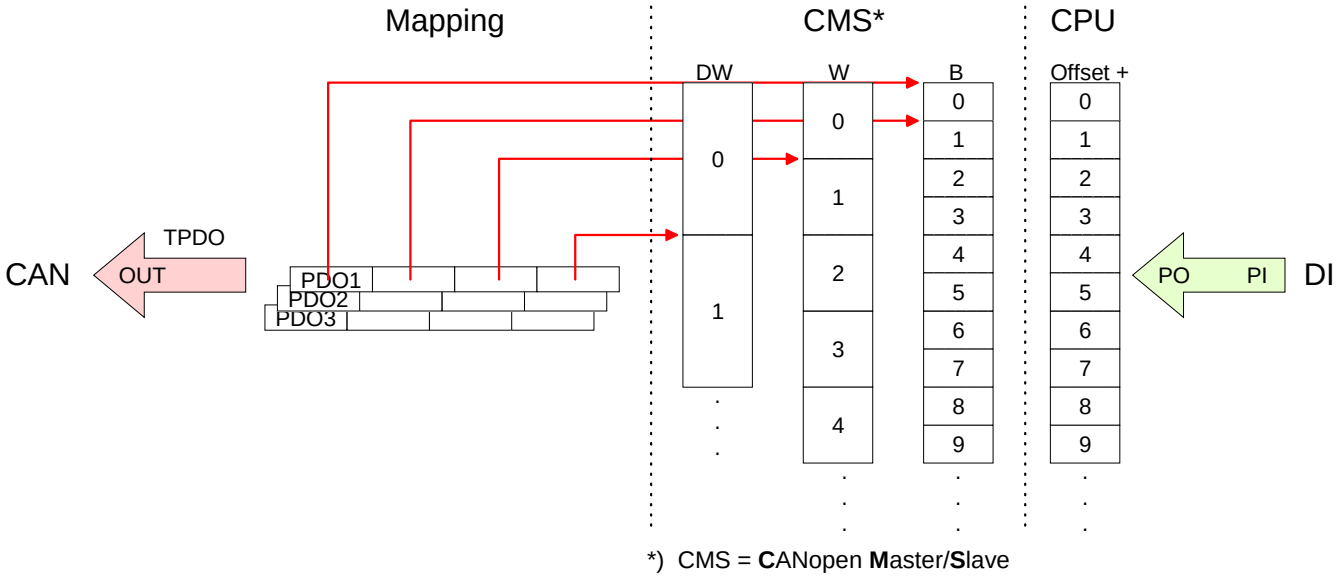
For input data, the following objects are available:

- 8 Bit digital input (Object 0x6000)
- 16 Bit digital input (Object 0x6100)
- 32 Bit digital input (Object 0x6120)
- 8 Bit input network variables (Object 0xA040)
- 16 Bit input network variables (Object 0xA100)
- 32 Bit input network variables (Object 0xA200)
- 64 Bit input network variables (Object 0xA440)

Like to see in the following illustration, the objects of the digital input data use the same memory area of the CPU.

For example, an access to Index 0x6000 with Subindex 2 corresponds an access to Index 0x6100 with Subindex 1. Both objects occupy the same memory cell in the CPU.

Please regard that the input network variables also use the same memory area.



*) CMS = **C**ANopen **M**aster/**S**lave

Output data

For the digital output data, the assignment is similar.

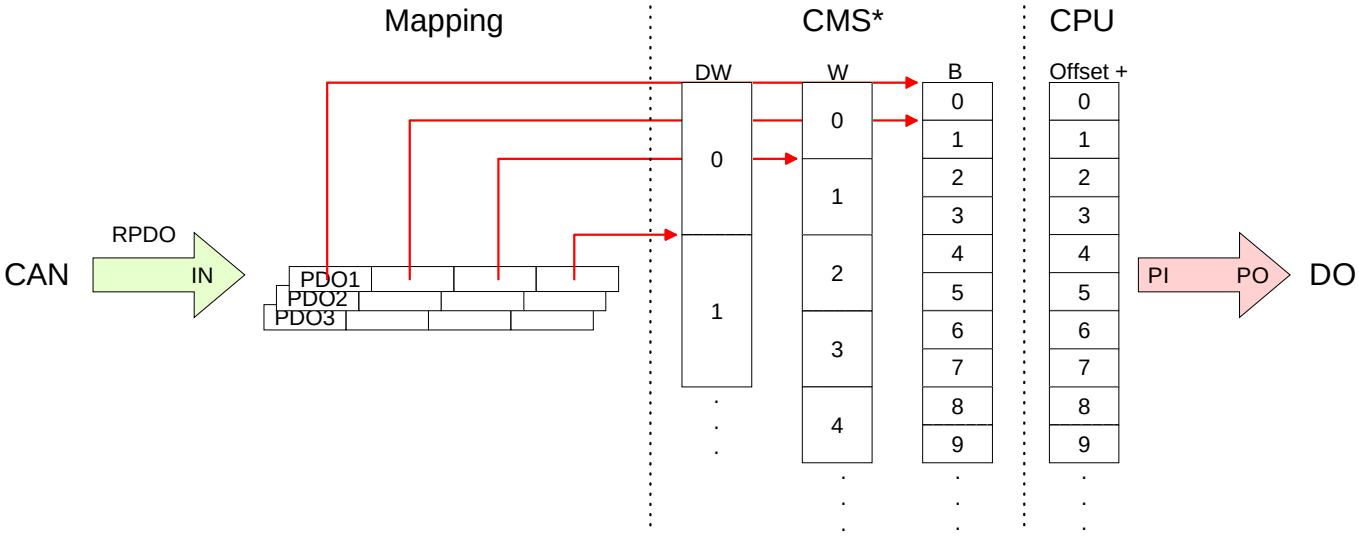
For output data, the following objects are available:

- 8 Bit digital output (Object 0x6200)
- 16 Bit digital output (Object 0x6300)
- 32 Bit digital output (Object 0x6320)
- 8 Bit output network variables (Object 0xA400)
- 16 Bit output network variables (Object 0xA580)
- 32 Bit output network variables (Object 0xA680)
- 64 Bit output network variables (Object 0xA8C0)

Like to see in the following illustration, the objects of the digital output data use the same memory area of the CPU.

For example, an access to Index 0x6200 with Subindex 2 corresponds an access to Index 0x6300 with Subindex 1. Both objects occupy the same memory cell in the CPU.

Please regard that the output network variables also use the same memory area.



*) CMS = **C**ANopen **M**aster/**S**lave

CANopen - Messages

Identifier

All CANopen messages have the following structure according to CIA DS-301:

Identifier

Byte	Bit 7 ... Bit 0
1	Bit 3 ... Bit 0: most significant 4 bits of the module-ID Bit 7 ... Bit 4: CANopen function code
2	Bit 3 ... Bit 0: data length code (DLC) Bit 4: RTR-Bit: 0: no data (request code) 1: data available Bit 7 ... Bit 5: Least significant 3 bits of the module-ID

Data

Data

Byte	Bit 7 ... Bit 0
3 ... 10	Data

An additional division of the 2Byte identifier into function portion and a module-ID gives the difference between this and a level 2 message. The function determines the type of message (object) and the module-ID addresses the receiver.

CANopen devices exchange data in the form of objects. The CANopen communication profile defines two different object types as well as a number of special objects.

The VIPA CAN master supports the following objects:

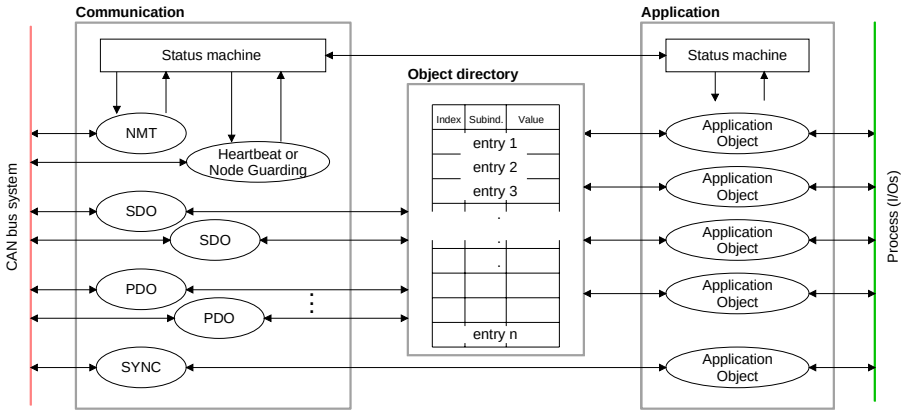
- 40 Transmit PDOs (PDO Linking, PDO Mapping)
- 40 Receive PDOs (PDO Linking, PDO Mapping)
- 2 Standard SDOs (1 Server, 127 Clients)
- 1 Emergency Object
- 1 Network management Object NMT
- Node Guarding
- Heartbeat

Note!

The exact structure and data content of all objects is described in the CiA-Profiles DS-301, DS-302, DS-401 and DS-405.

Structure of the device model

A CANopen device can be structured as follows:



Communication

Serves the communication data objects and the concerning functionality for data transfer via the CANopen network.

Application

The application data objects contain e.g. in- and output data. In case of an error, an application status machine switches the outputs in a secure state.

The object directory is organized as 2 dimension table. The data is addressed via index and sub-index.

Object directory

This object directory contains all data objects (application data + parameters) that are accessible and that influence the behavior of communication, application and status machines.

PDO

In many fieldbus systems the whole process image is transferred - mostly more or less cyclically. CANopen is not limited to this communication principle, for CAN supports more possibilities through multi master bus access coordination.

CANopen divides the process data into segments of max. 8Byte. These segments are called **p**rocess **d**ata **o**bjects (PDOs). Every PDO represents one CAN telegram and is identified and prioritized via its specific CAN identifier.

For the exchange of process data, the VIPA CAN-Master supports 80 PDOs. Every PDO consists of a maximum of 8 data bytes. The transfer of PDOs is not verified by means of acknowledgments since the CAN protocol guarantees the transfer.

There are 40 Tx transmit PDOs for input data and 40 Rx receive PDOs for output data. The PDOs are named seen from the CAN-Master:

Receive PDOs (RxPDOs) are received by the CAN-Master and contain input data.

Transmit PDOs (TxPDOs) are send by the CAN-Master and contain output data.

The assignment of the PDOs to input or output data occurs via WinCoCT automatically.

SDO

For access to the object directory, the **S**ervice-**D**ata-**O**bject (SDO) is used. The SDO allows you a read or write access to the object directory. In the CAL-Layer-7-Protocol you find the specification of the Multiplexed-Domain-Transfer-Protocol that is used by the SDOs. This protocol allows you to transfer data with any length. At need, the messages are divided into several CAN messages with identical identifier (segmentation). A SDO is transferred acknowledged, i.e. every reception of a message is acknowledged.

Note!

A more detailed description of the SDO telegrams is to find in the CiA norm DS-301.

In the following only the error messages are described that may occur at a wrong parameter communication.

SFC 219 CAN_TLGR SDO request to CAN master

Every CPU has the SFC 219 integrated. This allows you to start a SDO read or write access from your PLC program to the CAN master.

You address your master via the plug-in location and the destination slave via its CAN address. The process data is defined by index and subindex. Via SDO every access transfers max. one data word process data. The SFC 219 contains the following parameters:

Adresse	Deklaration	Name	Typ	Anfangswe	Kommentar
0.0	in	Request	BOOL		
1.0	in	Slot_Master	BYTE		
2.0	in	NodeID	BYTE		
3.0	in	Transfertyp	BYTE		
4.0	in	Index	DWORD		
8.0	in	Subindex	DWORD		
12.0	out	CanOpenError	DWORD		
16.0	out	RetVal	WORD		
18.0	out	Busy	BOOL		
20.0	in_out	DataBuffer	ANY		

Request — Control parameter: 1: Start the order

Slot_Master — depending on plug-in location no.
0: for addressing the integrated CAN master
1 ... 32: for addressing stand-alone System 200V CAN master

NodelD — Address of the CANopen node (1...127)

Transfer type — 40h, 60h: Read SDO
61h: Write SDO (undefined length)
23h: Write SDO (1 DWORD)
2Bh: Write SDO (1 WORD)
2Fh: Write SDO (1 BYTE)

Index — CANopen Index

Subindex — CANopen Subindex

CanOpenError

If no error occurs CANopenError returns value 0.

In case of error the CANopenError contains one of the following error messages which are generated in the CAN master:

Code	Description
0x05030000	Toggle bit not alternated
0x05040000	SDO protocol timed out
0x05040001	Client/server command specifier not valid or unknown
0x05040002	Invalid block size (block mode only)
0x05040003	Invalid sequence number (block mode only)
0x05040004	CRC error (block mode only)
0x05040005	Out of memory
0x06010000	Unsupported access to an object
0x06010001	Attempt to read a write only object
0x06010002	Attempt to write a read only object
0x06020000	Object does not exist in the object dictionary
0x06040041	Object cannot be mapped to the PDO
0x06040042	The number and length of the objects to be mapped would exceed PDO length
0x06040043	General parameter incompatibility reason
0x06040047	General internal incompatibility in the device
0x06060000	Access failed due to an hardware error
0x06070010	Data type does not match, length of service parameter does not match
0x06070012	Data type does not match, length of service parameter too high
0x06070013	Data type does not match, length of service parameter too low
0x06090011	Sub-index does not exist
0x06090030	Value range of parameter exceeded (only for write access)
0x06090031	Value of parameter written too high
0x06090032	Value of parameter written too low
0x06090036	Maximum value is less than minimum value
0x08000000	general error
0x08000020	Data cannot be transferred or stored to the application
0x08000021	Data cannot be transferred or stored to the application because of local control
0x08000022	Data cannot be transferred or stored to the application because of the present device state
0x08000023	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error)

RetVal

When the function has been executed successfully, the return value contains the valid length of the respond data: 1: BYTE, 2: WORD, 4: DWORD.

If an error occurs during function processing, the return value contains an error code.

Value	Description
F021h	Invalid slave address (Call parameter equal 0 or above 127)
F022h	Invalid Transfer type (Value unequal 60h, 61h)
F023h	Invalid data length (data buffer to small, at SDO read access it should be at least 4Byte, at SDO write access 1Byte, 2Byte or 4Byte).
F024h	The SFC is not supported
F025h	Write buffer in the CANopen master full, service can not be processed at this time.
F026h	Read buffer in the CANopen master full, service can not be processed at this time.
F027h	The SDO read or write access returned wrong answer, see CANopen Error Codes.
F028h	SDO-Timeout (no CANopen participant with this Node-Id has been found).

Busy

Busy = 1: The read/write job is not yet completed.

DataBuffer

SFC data communication area. Set here an ANY pointer of the type Byte.

Read SDO: Destination area for the SDO data that were read.

Write SDO: Source area for the SDO data that were write.

Note

Unless a SDO demand was processed error free, RetVal contains the length of the valid response data in 1, 2 or 4 byte and the CanOpenError the value 0.

Object directory

Structure

The CANopen object directory contains all relevant CANopen objects for the bus coupler. Every entry in the object directory is marked by a 16Bit index.

If an object exists of several components (e.g. object type Array or Record), the components are marked via an 8Bit sub-index.

The object name describes its function. The data type attribute specifies the data type of the entry.

The access attribute defines, if the entry may only be read, only be written or read and written.

The object directory is divided into the following 3 parts:

Communication specific profile area (0x1000 – 0x1FFF)

This area contains the description of all relevant parameters for the communication.

0x1000 – 0x1011	General communication specific parameters (e.g. device name)
0x1400 – 0x1427	Communication parameters (e.g. identifier) of the receive PDOs
0x1600 – 0x1627	Mapping parameters of the receive PDOs The mapping parameters contain the cross-references to the application objects that are mapped into the PDOs and the data width of the depending object.
0x1800 – 0x1827 0x1A00 – 0x1A27	Communication and mapping parameters of the transmit PDOs

Manufacturer specific profile area (0x2000 – 0x5FFF)

Here you find the manufacturer specific entries. The CAN master from VIPA has no manufacturer specific entries.

Standardized device profile area (0x6000 – 0x9FFF)

This area contains the objects for the device profile acc. DS-401.

Note!

For the CiA norms are exclusively available in English, we adapted the object tables. Some entries are described below the according tables.

A more detailed description of the table entries is to find below the according table.

Object directory overview

Index	Content of Object
1000h	Device type
1001h	Error register
1005h	COB-ID SYNC
1006h	Communication Cycle Period
1007h	Synchronous Window Length
1008h	Manufacturer Hardware Version
1009h	Hardware Version
100Ah	Software Version
100Ch	Guard Time
100Dh	Life Time Factor
1016h	Consumer Heartbeat Time
1017h	Producer Heartbeat Time
1018h	Identity Object
1400h to 1427h	Receive PDO Communication Parameter
1600h to 1627h	Receive PDO Mapping Parameter
1800h to 1827h	Transmit PDO Communication Parameter
1A00h to 1A27h	Transmit PDO Mapping Parameter
1F22h	Concise DCF
1F25h	Post Configuration
1F80h	NMT StartUp
1F81h	Slave Assignment
1F82h	Request NMT
1F83h	Request Guarding
6000h	Digital-Input-8-Bit Array (see DS 401)
6100h	Digital-Input-16-Bit Array (see DS 401)
6120h	Digital-Input-32Bit Array (see DS 401)
6200h	Digital-Output-8-Bit Array (see DS 401)
6300h	Digital-Output-16-Bit Array (see DS 401)
6320h	Digital-Output-32-Bit Array (see DS 401)
A040h	Dynamic Unsigned8 Input
A100h	Dynamic Unsigned16 Input
A200h	Dynamic Unsigned32 Input
A4400h	Dynamic Unsigned64 Input
A4C0h	Dynamic Unsigned8 Output
A580h	Dynamic Unsigned16 Output
A680h	Dynamic Unsigned32 Output
A8C0h	Dynamic Unsigned64 Output

Device Type

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1000	0	Device Type	Unsigned32	ro	N	0x00050191	Statement of device type

The 32Bit value is divided into two 16Bit fields:

MSB	LSB
Additional information Device	**profile number**
0000 0000 0000 wxyz (bit)	405dec=0x0195

The "additional information" contains data related to the signal types of the I/O device:

z=1 digital inputs

y=1 digital outputs

x=1 analog inputs

w=1 analog outputs

Error register

Index	Sub-Index	Name	Type	Attr.	Map.	Default value	Meaning
0x1001	0	Error Register	Unsigned8	ro	Y	0x00	Error register

Bit 7							Bit 0
ManSpec	reserved	reserved	Comm.	reserved	reserved	reserved	Generic

ManSpec.: Manufacturer specific error, specified in object 0x1003.

Comm.: Communication error (overrun CAN)

Generic: A not more precisely specified error occurred (flag is set at every error message)

SYNC identifier

Index	Sub-Index	Name	Type	Attr.	Map.	Default value	Meaning
0x1005	0	COB-Id sync message	Unsigned32	ro	N	0x80000080	Identifier of the SYNC message

The lower 11Bit of the 32Bit value contain the identifier (0x80=128dez), while the MSBit indicates whether the device receives the SYNC telegram (1) or not (0).

Attention: In contrast to the PDO identifiers, the MSB being set indicates that this identifier is relevant for the node.

SYNC interval

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1006	0	Communi-cation cycle period	Unsigned32	rw	N	0x00000000	Maximum length of the SYNC interval in µs.

If a value other than zero is entered here, the master goes into error state if no SYNC telegram is received within the set time during synchronous PDO operation.

Synchronous Window Length

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1007	0	Synchronous window length	Unsigned32	rw	N	0x00000000	Contains the length of time window for synchronous PDOs in µs.

Device name

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1008	0	Manufacturer device name	Visible string	ro	N		Device name of the bus coupler

VIPA 21x-2CM02

Since the returned value is longer than 4Byte, the segmented SDO protocol is used for transmission.

Hardware version

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1009	0	Manufacturer Hardware version	Visible string	ro	N		Hardware version number of bus coupler

1.00

Since the returned value is longer than 4Byte, the segmented SDO protocol is used for transmission.

Software version

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x100A	0	Manufacturer Software version	Visible string	ro	N		Software version number CANopen software

1.xx

Since the returned value is longer than 4Byte, the segmented SDO protocol is used for transmission.

Guard time

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x100C	0	Guard time [ms]	Unsigned16	rw	N	0x0000	Interval between two guard telegrams. Is set by the NMT master or configuration tool.

Life time factor

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x100D	0	Life time factor	Unsigned8	rw	N	0x00	Life time factor x guard time = life time (watchdog for life guarding)

If a guarding telegram is not received within the life time, the node enters the error state. If the life time factor and/or guard time =0, the node does not carry out any life guarding, but can itself be monitored by the master (node guarding).

Consumer Heartbeat Time

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1016	0 1...127	Consumer heartbeat time	Unsigned8 Unsigned32	ro rw	N N	0x05 0x00000000	Number of entries Consumer heartbeat time

Structure of the "Consumer Heartbeat Time" entry::

Bits	31-24	23-16	15-0
Value	Reserved	Node-ID	Heartbeat time
Encoded as	Unsigned8	Unsigned8	Unsigned16

As soon as you try to configure a consumer heartbeat time unequal zero for the same node-ID, the node interrupts the SDO download and throws the error code 0604 0043hex.

Producer Heartbeat Time

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1017	0	Producer heartbeat time	Unsigned16	rw	N	0x0000	Defines the cycle time of heartbeat in ms

Identity Object

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1018	0	Identity Object	Unsigned8	ro	N	0x04	Contains general Information about the device (number of entries)
	1	Vendor ID	Unsigned32	ro	N	0xAFFEAFFE	Vendor ID
	2	Product Code	Unsigned32	ro	N	0x2142CA02	Product Code
	3	Revision Number	Unsigned32	ro	N		Revision Number
	4	Serial Number	Unsigned32	ro	N		Serial Number

Communication parameter RxPDO

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1400 ... 0x1427	0	Number of Elements	Unsigned8	ro	N	0x02	Communication parameter for the first receive PDOs, Subindex 0: number of following parameters
	1	COB-ID	Unsigned32	rw	N	0xC0000200 + NODE_ID	COB-ID RxPDO1
	2	Transmis-sion type	Unsigned8	rw	N	0xFF	Transmission type of the PDO

Sub-index 1 (COB-ID): The lower 11Bit of the 32Bit value (Bits 0-10) contain the CAN identifier, the MSBit (Bit 31) shows if the PDO is active (1) or not(0), Bit 30 shows if a RTR access to this PDO is permitted (0) or not (1).

The sub-index 2 contains the transmission type.

Mapping RxPDO

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1600 ... 0x1627	0	Number of Elements	Unsigned8	rw	N	0x01	Mapping parameter of the first receive PDO; subindex 0: number of mapped objects
	1	1st mapped object	Unsigned32	rw	N	0x62000108	(2 byte index, 1 byte subindex, 1 byte bit-width)
	2	2nd mapped object	Unsigned32	rw	N	0x62000208	(2 byte index, 1 byte subindex, 1 byte bit-width)
	...	...	...	...	...	...	...
	8	8th mapped	Unsigned32	rw	N	0x62000808	(2 byte index, 1 byte subindex, 1 byte bit-width)

The reception PDOs get a default mapping automatically from the master depending on the connected modules.

Communication parameter TxPDO1

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1800 ... 0x1827	0	Number of Elements	Unsigned8	ro	N	0x05	Communication parameter of the first transmit PDO, subindex 0: number of following parameters
	1	COB-ID	Unsigned32	rw	N	0x80000180 + NODE_ID	COB-ID TxPDO1
	2	Transmission type	Unsigned8	rw	N	0xFF	Transmission type of the PDO
	3	Inhibit time	Unsigned16	rw	N	0x0000	Repetition delay [value x 100 µs]
	5	Event time	Unsigned16	rw	N	0x0000	Event timer [value x 1 ms]

Sub-index 1 (COB-ID): The lower 11Bit of the 32Bit value (Bits 0-10) contain the CAN identifier, the MSBit (Bit 31) shows if the PDO is active (1) or not (0), Bit 30 shows if a RTR access to this PDO is permitted (0) or not (1). The sub-index 2 contains the transmission type, sub-index 3 the repetition delay time between two equal PDOs. If an event timer exists with a value unequal 0, the PDO is transmitted when the timer exceeds.

If a "inhibit timer" exists, the event is delayed for this time.

Mapping TxPDO1

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1A00 ... 0x1A27	0	Number of Elements	Unsigned8	rw	N	depending on the components fitted	Mapping parameter of the first transmit PDO; subindex 0: number of mapped objects
	1	1st mapped object	Unsigned32	rw	N	0x60000108	(2 byte index, 1 byte subindex, 1 byte bit-width)
	2	2nd mapped object	Unsigned32	rw	N	0x60000208	(2 byte index, 1 byte subindex, 1 byte bit-width)
	...	...	...	...	...	...	...
	8	8th mapped object	Unsigned32	rw	N	0x60000808	(2 byte index, 1 byte subindex, 1 byte bit-width)

The send PDOs get a default mapping automatically from the coupler depending on the connected modules.

Concise DCF

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1F22	Array	Concise DCF	Domain	rw	N		

This object is required for the Configuration Manager. The Concise-DCF is the short form of the DCF (**D**evice **C**onfiguration **F**ile).

Post Configuration

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1F25	Array	ConfigureSlave	Unsigned32	rw	N	0x00000000	

Via this entry, the Configuration Manager can be forced to transfer a stored configuration into the net.

The configuration can be initiated for a defined node at any time via the index 0x1F25.

Subindex 0 has the value 128.

Subindex x (with x = 1..127): Starts the reconfiguration for nodes with the node ID x.

Subindex 128: reconfiguration of all nodes.

For example: If you want to initiate the configuration for node 2 and there are configuration data for this node available, you have to write the value 0x666E6F63 (ASCII = "conf") to the object 1F25h Subindex 2.

NMT Start-up

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1F80	0x00	NMTStartup	Unsigned32	rw	N	0x00000000	

Define the device as NMT master.

Bit	Meaning
Bit 0	0 : Device is NOT the NMT Master. All other bits have to be ignored. The objects of the Network List have to be ignored. 1 : Device is the NMT Master.
Bit 1	0 : Start only explicitly assigned slaves. 1 : After boot-up perform the service NMT Start Remote Node All Nodes
Bit 2..31	Reserved by CiA, always 0

Slave Assignment

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1F81	0x00	SlaveAssignment	Unsigned32	rw	N	0x00000000	

Enter the nodes that are controlled by the master. For every assigned node you need one entry.

Subindex 0 has the value 127. Every other Subindex corresponds with the Node-ID of the node.

Byte	Bit	Description
Byte 0	Bit 0	0: Node with this ID is not a slave 1: Node with this ID is a slave. After configuration (with Configuration Manager) the Node will be set to state Operational.
	Bit 1	0: On Error Control Event or other detection of a booting slave inform the application. 1: On Error Control Event or other detection of a booting slave inform the application and automatically start Error Control service.
	Bit 2	0: On Error Control Event or other detection of a booting slave do NOT automatically configure and start the slave. 1: On Error Control Event or other detection of a booting slave do start the process Start Boot Slave.
	Bit 3..7	Reserved by CiA, always 0
Byte 1		8 Bit Value for the RetryFactor
Byte 2,3		16 Bit Value for the GuardTime

Request NMT

Index	Sub-Index	Name	Type	Attr.	Map.	Default value	Meaning
0x1F82	0x00	RequestNMT	Unsigned32	rw	N	0x00000000	

If a totally automatic start of the stack is not wanted, the functionalities:

- Status change
- Start of the guarding
- Configuration via CMT

can be also executed at request for every node. The request always happens via objects in the object directory.

The switch of the communication state of all nodes in the network (including the local slaves) happens via the entry 1F82h in the local object directory:

Subindex 0 has the value 128.

Subindex x (with x=1..127): Initiates the NMT service for nodes with Node ID x.

Subindex 128: Initiates NMT service for all nodes.

At write access, the wanted state is given as value.

State	Value
Prepared	4
Operational	5
ResetNode	6
ResetCommunication	7
PreOperational	127

Request Guarding

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x1F83	0x00	RequestGuarding	Unsigned32	rw	N	0x00000000	

Subindex 0 has the value 128.

Subindex x (with x=1..127): Initiates guarding for the slave with Node ID x.

Value	Write Access	Read Access
1	Start Guarding	Slave actually is guarded
0	Stop Guarding	Slave actually is not guarded

Subindex 128: Request Start/Stop Guarding for all nodes.

8bit Digital inputs

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x6000	0x00	8bit digital input block	Unsigned8	ro	N	0x01	Number of available digital 8-bit input blocks
	0x01	1st input block	Unsigned8	ro	Y		1st digital input block
	...	...	...	...	...	...	...
	0x40	64th input block	Unsigned8	ro	Y		64th digital input block

16bit Digital inputs

Index	Sub-Index	Name	Type	Attr.	Map.	Default value	Meaning
0x6100	0x00	16bit digital input block	Unsigned8	ro	N	depending on the fitted components	Number of available digital 16-bit input blocks
	0x01	1st input block	Unsigned16	ro	N		1st digital input block
	...	...	...	...	...	...	...
	0x20	32nd input block	Unsigned16	ro	N		32nd digital input block

32bit Digital inputs

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x6120	0x00	32bit digital input block	Unsigned8	ro	N	depending on the compo-nents fitted	Number of available digital 32-bit input blocks
	0x01	1st input block	Unsigned32	ro	N		1st digital input block
	...	...	...	...	...	...	...
	0x10	16th input block	Unsigned32	ro	N		16 digital input block

8bit Digital outputs

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x6200	0x00	8bit digital output block	Unsigned8	ro	N	0x01	Number of available digital 8-bit output blocks
	0x01	1st output block	Unsigned8	rw	Y		1st digital output block
	...	...	...	...	...	...	...
	0x40	64th output block	Unsigned8	rw	Y		64 nd digital output block

16bit Digital outputs

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x6300	0x00	16bit digital input block	Unsigned8	ro	N	Depending on the compo-nents fitted	Number of available digital 16-bit output blocks
	0x01	1st output block	Unsigned16	rw	N		1st digital output block
	...	...	...	...	...	...	...
	0x20	32nd output block	Unsigned16	rw	N		32nd digital output block

32bit Digital outputs

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0x6320	0x00	32bit digital input block	Unsigned8	ro	N	Depending on the compo-nents fitted	Number of available digital 32-bit output blocks
	0x01	1st output block	Unsigned32	rw	N		1st digital output block
	...	...	...	...	...	...	...
	0x10	16th output block	Unsigned32	rw	N		16 digital output block

8bit Network input variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA040	0x00	8bit digital input block	Unsigned8	ro	N	0x01	Number of available digital 8-bit input blocks
	0x01	1st input block	Unsigned8	ro	Y		1st digital input block
	...	...	...	...	...	...	...
	0x140	320th input block	Unsigned8	ro	Y		320 nd digital input block

16bit Network input variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA100	0x00	16bit digital input block	Unsigned8	ro	N	depending on the fitted components	Number of available digital 16-bit input blocks
	0x01	1st input block	Unsigned16	ro	N		1st digital input block
	...	...	...	...	...	...	...
	0xA0	160th input block	Unsigned16	ro	N		160 nd digital input block

32bit Network input variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA200	0x00	32bit digital input block	Unsigned8	ro	N	depending on the compo-nents fitted	Number of available digital 32-bit input blocks
	0x01	1st input block	Unsigned32	ro	N		1st digital input block
	...	...	...	...	...	...	...
	0x50	80th input block	Unsigned32	ro	N		80 digital input block

64bit Network input variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA440	0x00	64bit digital input block	Unsigned8	ro	N	depending on the compo-nents fitted	Number of available digital 64-bit input blocks
	0x01	1st input block	Unsigned32	ro	N		1st digital input block
	...	...	...	...	...	...	...
	0x28	40th input block	Unsigned32	ro	N		40 digital input block

8bit Network output variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA400	0x00	8bit digital output block	Unsigned8	ro	N	0x01	Number of available digital 8-bit output blocks
	0x01	1st output block	Unsigned8	rw	Y		1st digital output block
	...	...	...	...	...	...	...
	0x140	320th output block	Unsigned8	rw	Y		320 nd digital output block

16bit Network output variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA580	0x00	16bit digital input block	Unsigned8	ro	N	Depending on the compo-nents fitted	Number of available digital 16-bit output blocks
	0x01	1st output block	Unsigned16	rw	N		1st digital output block
	...	...	...	...	...	...	...
	0xA0	160th output block	Unsigned16	rw	N		160 th digital output block

32bit Network output variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA680	0x00	32bit digital input block	Unsigned8	ro	N	Depending on the compo-nents fitted	Number of available digital 32-bit output blocks
	0x01	1st output block	Unsigned32	rw	N		1st digital output block
	...	...	...	...	...	...	...
	0x50	80th output block	Unsigned32	rw	N		80 digital output block

64bit Network output variables

Index	Sub-index	Name	Type	Attr.	Map.	Default value	Meaning
0xA8C0	0x00	64bit digital input block	Unsigned8	ro	N	Depending on the compo-nents fitted	Number of available digital 64-bit output blocks
	0x01	1st output block	Unsigned32	rw	N		1st digital output block
	...	...	...	...	...	...	...
	0x50	40th output block	Unsigned32	rw	N		40 digital output block

Chapter 9 Deployment CPU 21xSER-1

Overview

Content of this chapter is the deployment of the CPU 21xSER-1 with RS232C/RS485 interface.

Here you'll find all information about the deployment of the serial interfaces of the CPU 21xSER-1.

The following description includes:

- Principles of the serial communication
- Usage of the protocols ASCII, STX/ETX, 3964R, USS and Modbus
- Deployment of the serial RS232C/RS485 interface
- Example Modbus communication

Content

Fast introduction

General

The CPU 21xSER-1 provides serial interfacing facilities between the processes of different source and destination systems. For the serial communication the CPU 21x-2BS12 has got a RS232C interface and the CPU 21x-2BS32 a RS485 interface.

Protocols

The CPU supports the ASCII, STX/ETX, 3964R, USS and Modbus protocols and procedures.

Parameterization

The parameterization happens during runtime by means of the SFC 216 (SER_CFG). The parameters for STX/ETX, 3964R, USS and Modbus have to be stored in a DB.

Communication

With the help of SFCs you control the communication. The sending is executed with the SFC 217 (SER_SND) and the reception via SFC 218 (SER_RCV).

Another call of the SFC 217 SER_SND, 3964R, USS and Modbus provides you via RetVal with a return value which contains among others recent information about the acknowledgement of the partner.

The protocols USS and Modbus allows you to read the acknowledgement telegram by calling the SFC 218 SER_RCV after a SER_SND.

The SFCs are included in the consignment of the CPU 21xSER-1.

Overview over the SFCs for the serial communication

The following SFCs are deployed for the serial communication:

SFC		Description
SFC 216	SER_CFG	RS232/RS485 Parameterization
SFC 217	SER_SND	RS232/RS485 Send
SFC 218	SER_RCV	RS232/RS485 Receive

Protocols and procedures

Overview

The CPU 21xSER-1 supports the following protocols and procedures:

- ASCII communication
- STX/ETX
- 3964R
- USS
- Modbus

ASCII

ASCII data communication is one of the simple forms of data exchange.

Incoming characters are transferred 1 to 1.

At ASCII, with every cycle the read-SFC is used to store the data that is in the buffer at request time in a parameterized receive data block. If a telegram is spread over various cycles, the data is overwritten. There is no reception acknowledgement. The communication procedure has to be controlled by the concerning user application. An according Receive_ASCII-FB is to be found at ftp.vipa.de.

STX/ETX

STX/ETX is a simple protocol with start and end ID, where STX stands for **S**tart of **Tex**t and ETX for **E**nd of **Tex**t.

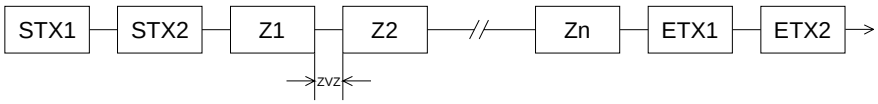
The STX/ETX procedure is suitable for the transfer of ASCII characters. It does not use block checks (BCC). Any data transferred from the periphery must be preceded by an Start followed by the data characters and the end character.

Depending of the byte width the following ASCII characters can be transferred: 5Bit: not allowed: 6Bit: 20...3Fh, 7Bit: 20...7Fh, 8Bit: 20...FFh.

The user data which includes all the characters between Start and End are transferred to the CPU when the End has been received.

When data is send from the CPU to a peripheral device, any user data is handed to the SFC 217 (SER_SND) and is transferred with added Start- and End-ID to the communication partner.

Message structure:



You may define up to 2 start and end characters.

You may work with 1, 2 or no Start- and with 1, 2 or no End-ID. As Start- res. End-ID all Hex values from 01h to 1Fh are permissible. Characters above 1Fh are ignored. In the user data, characters below 20h are not allowed and may cause errors. The number of Start- and End-IDs may be different (1 Start, 2 End res. 2 Start, 1 End or other combinations). If no End-ID is defined, all read characters are transferred to the PLC after a parameterizable character delay time (Timeout).

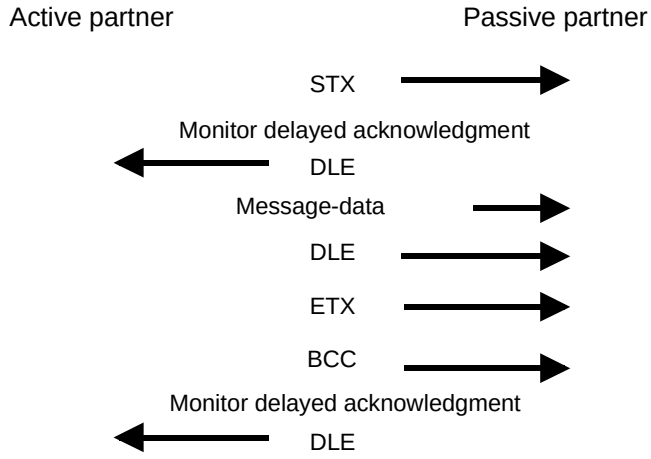
3964R

The 3964R procedure controls the data transfer of a point-to-point link between the CPU 21x SER-1 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 **S**tart of **Te**x**t**
- DLE **D**ata **L**ink **E**scape
- ETX **E**nd of **Te**x**t**
- BCC **B**lock **C**heck **C**haracter
- NAK **N**egative **Ac**k**nowledge**

Procedure



You may transfer a maximum of 255Byte per message.

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 3964R 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.

USS

The USS protocol (**U**niverselle **s**erielle **S**chnittstelle = universal serial interface) is a serial transfer protocol defined by Siemens for the drive and system components. This allows to build-up a serial bus connection between a superordinated master and several slave systems.

The USS protocol enables a time cyclic telegram traffic by presetting a fix telegram length.

The following features characterize the USS protocol:

- Multi point connection
- Master-Slave access procedure
- Single-Master-System
- Max. 32 participants
- Simple and secure telegram frame

You may connect 1 master and max. 31 slaves at the bus where the single slaves are addressed by the master via an address sign in the telegram. The communication happens exclusively in half-duplex operation.

After a send command, the acknowledgement telegram must be read by a call of the SFC 218 SER_RCV.

The telegrams for send and receive have the following structure:

Master-Slave telegram

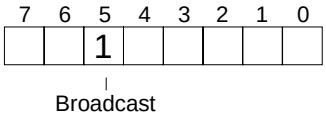
STE	LGE	ADR	PKE		IND		PWE		STW		HSW		BCC
02h			H	L	H	L	H	L	H	L	H	L	

Slave-Master telegram

STE	LGE	ADR	PKE		IND		PWE		ZSW		HIW		BCC
02h			H	L	H	L	H	L	H	L	H	L	

where STX: Start sign
LGE: Telegram length
ADR: Address
PKE: Parameter ID
IND: Index
PWE: Parameter value
STW: Control word
ZSW: State word
HSW: Main set value
HIW: Main effective value
BCC: Block Check Character

Broadcast with set Bit 5 in ADR-Byte



A request can be directed to a certain slave ore be send to all slaves as broadcast message. For the identification of a broadcast message you have to set Bit 5 to 1 in the ADR-Byte. Here the slave addr. (Bit 0 ... 4) is ignored. In opposite to a "normal" send command, the broadcast does not require a telegram evaluation via SFC 218 SER_RCV. Only write commands may be send as broadcast.

Modbus

The Modbus protocol is a communication protocol that fixes a hierarchic structure with one master and several slaves.

Physically, Modbus works with a serial half-duplex connection.

There are no bus conflicts occurring, because the master can only communicate with one slave at a time. After a request from the master, this waits for a preset delay time for an answer of the slave. During the delay time, communication with other slaves is not possible.

After a send command, the acknowledgement telegram must be read by a call of the SFC 218 SER_RCV.

The request telegrams send by the master and the respond telegrams of a slave have the following structure:

Start sign	Slave address	Function Code	Data	Flow control	End sign

Broadcast with slave address = 0

A request can be directed to a special slave or at all slaves as broadcast message. To mark a broadcast message, the slave address 0 is used.

In opposite to a "normal" send command, the broadcast does not require a telegram evaluation via SFC 218 SER_RCV.

Only write commands may be send as broadcast.

ASCII, RTU mode

Modbus offers 2 different transfer modes:

- ASCII mode: Every Byte is transferred in the 2 sign ASCII code. The data are marked with a start and an end sign. This causes a transparent but slow transfer.
- RTU mode: Every Byte is transferred as one character. This enables a higher data pass through as the ASCII mode. Instead of start and end sign, a time control is used.

The mode selection happens during runtime by using the SFC 216 SER_CFG.

Deployment of the serial interface

Outline

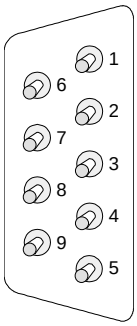
The PLC has got a RS232C- (Order-No.: 21x-2BS12) or RS485-interface (Order-No.:21x-2BS32).

Both interfaces are described in the following.

RS232 interface

- Interface is compatible to the COM interface of a PC
- Logical signals as voltage levels
- Point-to-point links with serial full-duplex transfer up to 15m of distance
- Data transfer rate up to 115.2kBaud
- Protocols supported: ASCII, STX/ETX, 3964R, USS and Modbus
- receive buffer an send buffer each with 2x256Byte
- The maximum telegram length is 255Byte

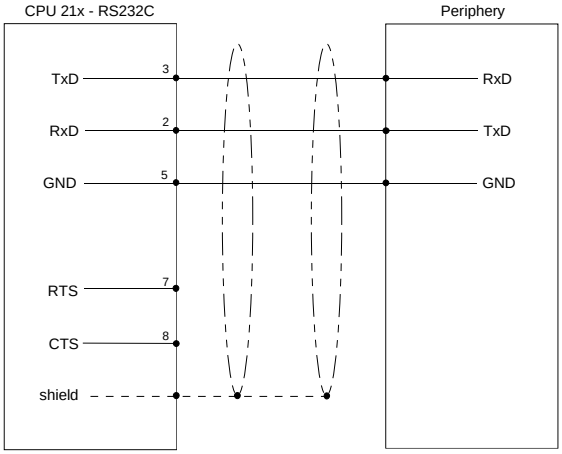
Connection RS232C



9pin plug

Pin	RS232C
1	CD-
2	RxD
3	TxD
4	DTR-
5	GND
6	DSR-
7	RTS-
8	CTS-
9	RI-

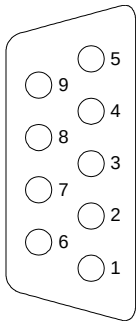
Connection RS232C



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
- Data communications up to a max. distance of 500m
- Data communication rate up to 115.2kBaud

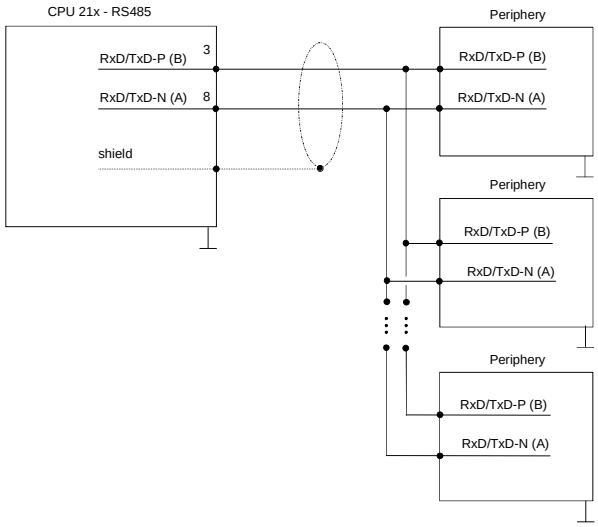
Connection RS485



9pin jack

Pin	RS485
1	n.c.
2	n.c.
3	RxD/TxD-P (Line B)
4	RTS
5	M5V
6	P5V
7	n.c.
8	RxD/TxD-N (Line A)
9	n.c.

Connection RS485



Principles of data transfer

Overview

The data transfer is handled during runtime by using SFCs. The principles of data transfer are the same for all protocols and is shortly illustrated in the following.

Principle

Data that is into the according data channel by the PLC, is stored in a FIFO send buffer (**f**irst **i**n **f**irst **o**ut) with a size of 2x256Byte and then put out via the interface.

When the interface receives data, this is stored in a FIFO receive buffer with a size of 2x256Byte and can there be read by the PLC.

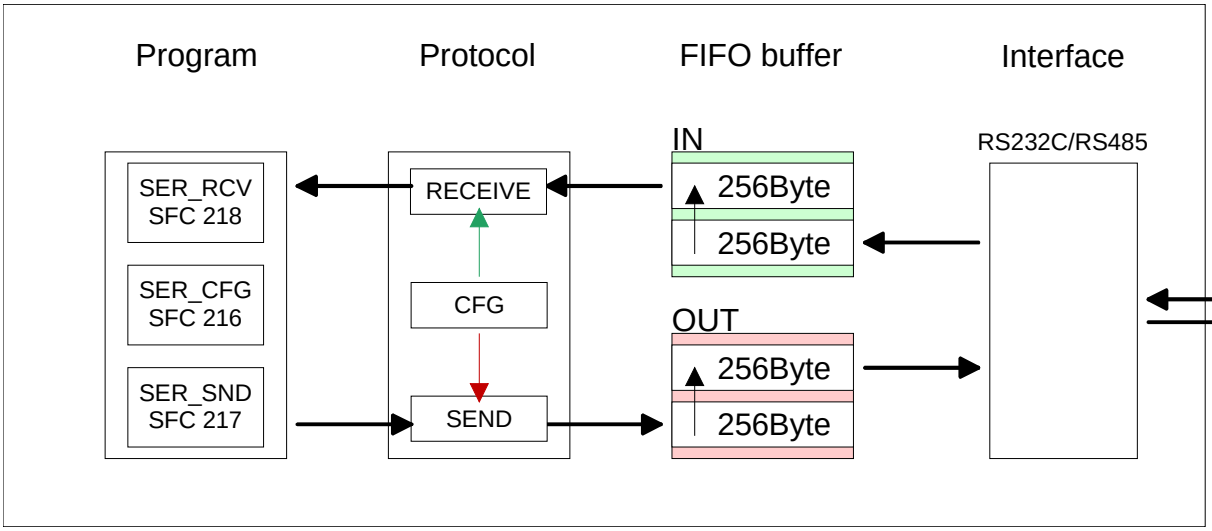
If the data is transferred via a protocol, the adaption of the data to the according protocol happens automatically.

In opposite to ASCII and STX/ETX, the protocols 3964R, USS and Modbus master require the acknowledgement of the partner.

An additional call of the SFC 217 SER_SND causes a return value in RetVal that includes among others recent information about the acknowledgement of the partner.

Further on for USS and Modbus master after a SER_SND the acknowledgement telegram must be evaluated by call of the SFC 218 SER_RCV.

CPU 21xSER-1



Principles for Modbus Slave

Data that the CPU has to provide for the Modbus master are stored in a FIFO send buffer (**f**irst **i**n **f**irst **o**ut) with a size of 2x256Byte. In opposite to the other protocols the data remain in the send buffer until they are requested by the Modbus master via a read command (function code 01h, 03h).

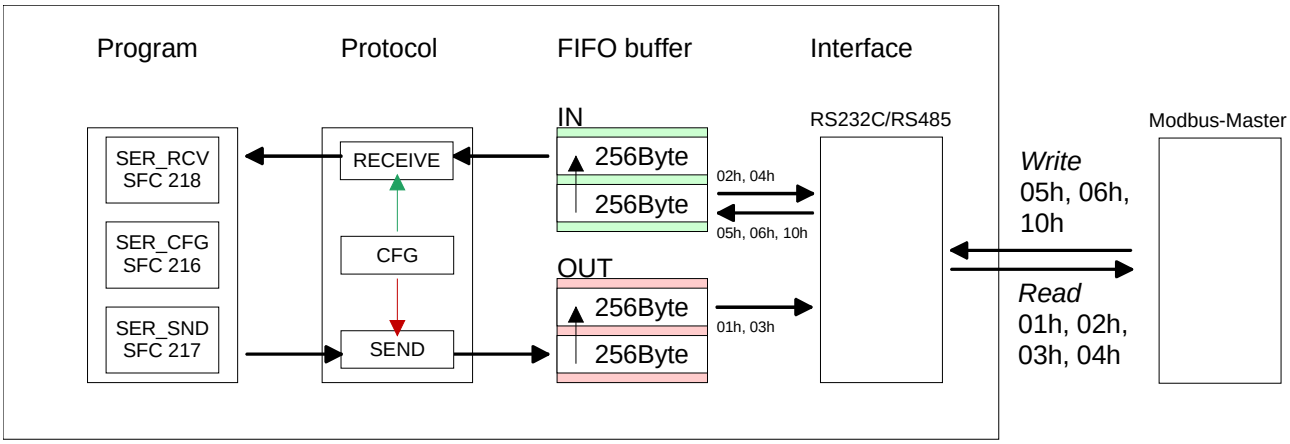
If the interface receives data from the master (function code 05h, 06h, 10h) these are stored in a FIFO receive buffer with a size of 2x256Byte and may there be read by the CPU.

The embedding of the data into the Modbus protocol happens automatically.

Please regard that the Modbus master may access the IN res. OUT buffer by according presetting of the read function code. By means of a read access to the IN buffer (function code 02h, 04h) the master may read data that it has sent to the Modbus slave before. The data remain in the buffer until they are overwritten by the Modbus master.

The following picture shows the communication principle. More information is also to be found in the chapter "Modbus slave function codes" further below.

CPU 21xSER-1



Parameterization

SFC 216 (SER_CFG)

The parameterization happens during runtime deploying the SFC 216 (SER_CFG). You have to store the parameters for STX/ETX, 3964R, USS and Modbus in a DB.

Please regard that not all protocols support the complete value range of the parameters. More detailed information is to be found in the description of the according parameter.

Note!

Please regard that the SFC216 is not called again during a communication because as a result of this all buffers are cleared.

If you don't want to alter the communication parameter any more, you should place the call of the SFC 216 in the start-up OB OB 100.

Name	Declaration	Type	Comment
Protocol	IN	BYTE	No. of protocol
Parameter	IN	ANY	Pointer to protocol-parameters
Baudrate	IN	BYTE	No of Baudrate
CharLen	IN	BYTE	0=5Bit, 1=6Bit, 2=7Bit, 3=8Bit
Parity	IN	BYTE	0=Non, 1=Odd, 2=Even
StopBits	IN	BYTE	1=1Bit, 2=1,5Bit, 3=2Bit
FlowControl	IN	BYTE	1 (fix)
RetVal	OUT	WORD	Error Code (0 = OK)

Protocol

Here you fix the protocol to be used. You may choose between:

1: ASCII

2: STX/ETX

3: 3964R

4: USS Master

5: Modbus RTU Master

6: Modbus ASCII Master

7: Modbus RTU Slave

8: Modbus ASCII Slave

Parameter (as DB) At ASCII protocol, this parameter is ignored.

At STX/ETX, 3964R, USS and Modbus you fix here a DB that contains the communication parameters and has the following structure for the according protocols:

Data block at STX/ETX

DBB0:	STX1	BYTE	(1. Start-ID in hexadecimal)
DBB1:	STX2	BYTE	(2. Start-ID in hexadecimal)
DBB2:	ETX1	BYTE	(1. End-ID in hexadecimal)
DBB3:	ETX2	BYTE	(2. End-ID in hexadecimal)
DBB4:	TIMEOUT	WORD	(max. delay time between 2 tele-grams in a time window of 10ms)

Note!

The start res. end sign should always be a value <20, otherwise the sign is ignored!

Data block at 3964R

DBB0:	Prio	BYTE	(The priority of both partners must be different. Prio 0 and 1 are possible)
DBB1:	ConnAttmptNr	BYTE	(Number of connection trials)
DBB2:	SendAttmptNr	BYTE	(Number of telegram retries)
DBB4:	CharTimeout	WORD	(Char. delay time in 10ms time window)
DBB6:	ConfTimeout	WORD	(Ackn. delay time in 10ms time window)

Data block at USS

DBB0:	Timeout	WORD	(Delay time in 10ms time grid)

Data block at Modbus-Master

DBB0:	Timeout	WORD	(Respond delay time in 10ms time grid)

Data block at Modbus-Slave

DBB0:	Address	BYTE	(Address 1...247 in the Modbus network)
DBB2:	Timeout	WORD	(Respond delay time in 10ms time grid)

Baud rate

Velocity of data transfer in Bit/s (Baud).

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

CharLen

Number of data bits where a character is mapped to.

0: 5Bit 1: 6Bit 2: 7Bit 3: 8Bit

Supported values:

Bit	ASCII	STX/ETX	3964R	USS	Modbus RTU	Modbus ASCII
5	x		x			
6	x	x	x			
7	x	x	x			x
8	x	x	x	x	x	x

Parity

The parity is -depending on the value- even or odd. For parity control, the information bits are extended with the parity bit, that amends via its value ("0" or "1") the value of all bits to a defined status. If no parity is set, the parity bit is set to "1", but not evaluated.

0: NONE 1: ODD 2: EVEN

StopBits

The stop bits are set at the end of each transferred character and mark the end of a character.

1: 1Bit 2: 1.5Bit 3: 2Bit

1.5Bit can only be used with CharLen 5 at this number of data 2Bit is not allowed.

FlowControl

With this bit you affect the behavior from signal **R**equest **t**o **s**end

"0" = RTS off

"1" = RTS is "0" at Receive
RTS is "1" at Send

Note: For RS485 FlowControl must be "1" !

RetVal (Return value)

Value	Description
0000h	no error
809Ah	interface not found
8x24h	Error at SFC-Parameter x, with x: 1: Error at "Protocol" 2: Error at "Parameter" 3: Error at "Baudrate" 4: Error at "CharLength" 5: Error at "Parity" 6: Error at "StopBits" 7: Error at "FlowControl"
809xh	Error in SFC parameter value x, where x: 1: Error at "Protocol" 3: Error at "Baudrate" 4: Error at "CharLength" 5: Error at "Parity" 6: Error at "StopBits" 7: Error at "FlowControl"
8092h	Access error in parameter DB (DB too short)
828xh	Error in parameter x of DB parameter, where x: 1: Error 1st parameter 2: Error 2nd parameter ...

Communication

Overview

The communication happens via the send and receive blocks SFC 217 (SER_SND) and SFC 218 (SER_RCV).

If data is transferred by means of a protocol, the embedding of the data into the according protocol happens automatically. Depending on the protocol you have to regard the following aspects

ASCII
STX/ETX

With ASCII res. STX/ETX the sending of the data happens without acknowledgement of the partner.

3964R

Another call of the SFC 217 SER_SND provides you via RetVal with a return value which contains among others recent information about the acknowledgement of the partner.

Modbus master
USS

Sending happens with acknowledgement of the partner. Another call of the SFC 217 SER_SND provides you via RetVal with a return value which contains among others recent information about the acknowledgement of the partner. After the transfer with SER_Send you receive the acknowledgement telegram of the partner by calling the SFC 218 SER_RCV.

Note!

Please regard that the SFC 216 is not called again during a communication because as a result of this all buffers are cleared.

SFC 217 (SER_SND)

This block allows to send data via the serial interface.

Name	Declaration	Type	Comment
DataPtr	IN	ANY	Pointer to Data Buffer for sending data
DataLen	OUT	WORD	Length of data to send
RetVal	OUT	WORD	Error Code (0 = OK)

DataPtr

Here you define a range of the type Pointer for the send buffer where the data that has to be send is stored. You have to set type, start and length.

Example: Data is stored in DB5 starting at 0.0 with a length of 124Byte.
DataPtr:=P#DB5.DBX0.0 BYTE 124

DataLen

Word where the number of sent bytes is stored.

At **STX/ETX** and **3964R**, the length set in DataPtr or 0 is entered.

At **ASCII**, the value may be different from the sent length when the data is sent that fast that not all data can be stored in the send buffer of 256Byte.

RetVal (Return value)

Value	Description
0000h	Send data - ready
1000h	Nothing sent (data length 0)
20xxh	Protocol executed error free with xx bit pattern for diagnosis
7001h	Data is stored in internal buffer - active (busy)
7002h	Transfer - active
80xxh	Protocol executed with errors with xx bit pattern for diagnosis (no acknowledgement by partner)
90xxh	Protocol not executed with xx bit pattern for diagnosis (no acknowledgement by partner)
8x24h	Error in SFC parameter x, where x: 1: Error in "DataPtr" 2: Error in "DataLen"
8122h	Error in parameter "Data" of SFC 216 (e.g. no DB)
807Fh	Internal error
809Ah	RS232C interface not found
809Bh	RS232C interface not configured

Protocol specific RetVal values

ASCII

Value	Description
9000h	Buffer overflow (no data send)

STX/ETX

Value	Description
9000h	Buffer overflow (no data send)
9001h	Data too long (>256Byte)
9004h	Character not allowed

3964R

Value	Description
2000h	Send ready without error
80FFh	Data transfer without acknowledgement of partner
9000h	Buffer overflow (no data send)
9001h	Data too long (>256Byte)

USS

Value	Description
2000h	Send ready without error
8080h	Receive buffer overflow (no space for receipt)
8090h	Acknowledgement delay time exceeded
80F0h	Wrong checksum in respond
80FEh	Wrong start sign in respond
80FFh	Wrong slave address in respond
9000h	Buffer overflow (no data send)
9001h	Data too long (>256Byte)
9002h	Data too short (<2Byte)

Modbus RTU/ASCII Master

Value	Description
2000h	Send ready without error
2001h	Send ready with error
8080h	Receive buffer overflow (no space for receipt)
8090h	Acknowledgement delay time exceeded
80F0h	Wrong checksum in respond
80FDh	Length of respond too long
80FEh	Wrong function code in respond
80FFh	Wrong slave address in respond
9000h	Buffer overflow (no data send)
9001h	Data too long (>256Byte)
9002h	Data too short (<2Byte)

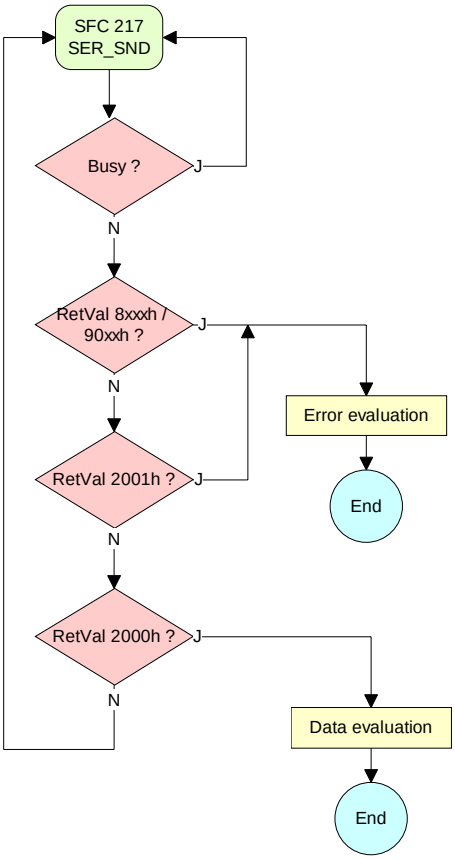
Modbus RTU/ASCII Slave

Value	Description
0000h	Send data - ready
9001h	Data too long (>256Byte)

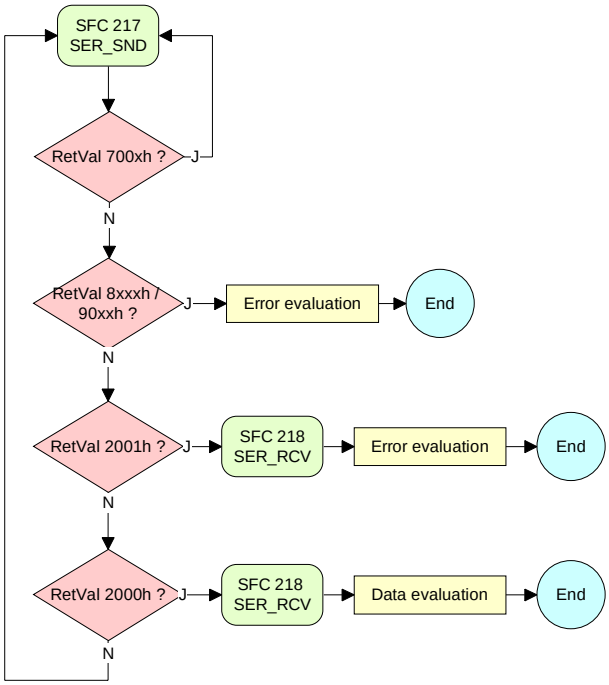
Principles of programming

The following text shortly illustrates the structure of programming a send command for the different protocols.

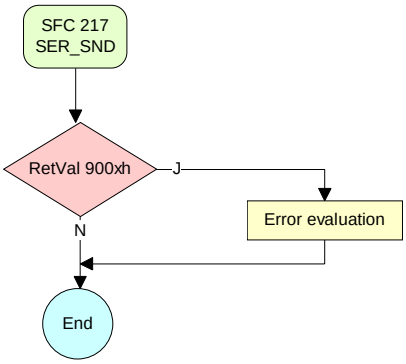
3964R



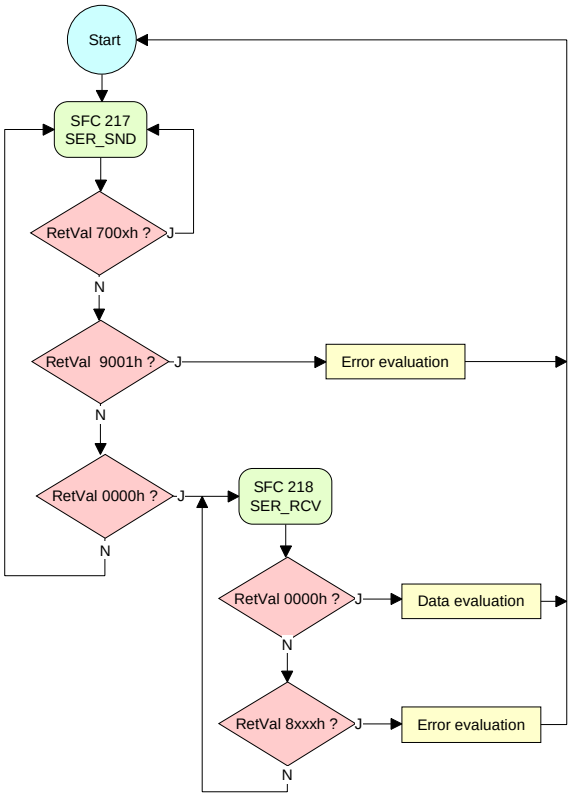
USS / Modbus master



ASCII / STX/ETX



Modbus slave



SFC 218 (SER_RCV)

This block receives data via the serial interface.

Name	Declaration	Type	Comment
DataPtr	IN	ANY	Pointer to Data Buffer for received data
DataLen	OUT	WORD	Length of received data
Error	OUT	WORD	Error Number
RetVal	OUT	WORD	Error Code (0 = OK)

DataPtr

Here you set a range of the type Pointer for the receive buffer where the reception data is stored. You have to set type, start and length.

Example: Data is stored in DB5 starting at 0.0 with a length of 124Byte.
DataPtr:=P#DB5.DBX0.0 BYTE 124

DataLen

Word where the number of received bytes is stored.

At **STX/ETX** and **3964R**, the length of the received user data or 0 is entered.

At **ASCII**, the number of read characters is entered. This value may be different from the read telegram length.

Error

At ASCII, this word gets an entry in case of an error. The following error messages are possible:

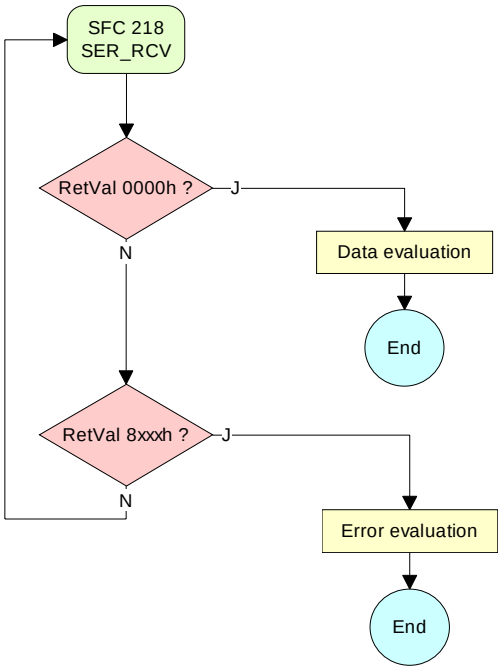
Bit	Error	Description
1	overrun	Overrun when a character could not be read from the interface fast enough
2	parity	Parity error
3	framing error	Error that shows that a defined bit frame is not met, exceeds the allowed length or contains an additional bit sequence (stop bit error)

RetVal (Return value)

Value	Description
0000h	no error
1000h	Receive buffer too small (data loss)
8x24h	Error at SFC-Parameter x, with x:
	1: Error at "DataPtr"
	2: Error at "DataLen"
	3: Error at "Error"
8122h	Error in parameter "Data" of SFC 216 (e.g. no DB)
809Ah	serial interface not found
809Bh	serial interface not configured

Principles of programming

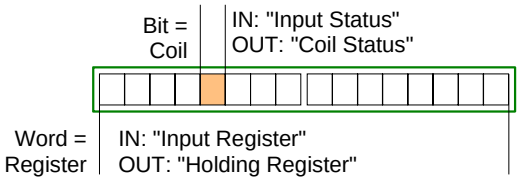
The following picture shows the basic structure for programming a receive command. This structure can be used for all protocols.



Modbus slave 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 under 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.

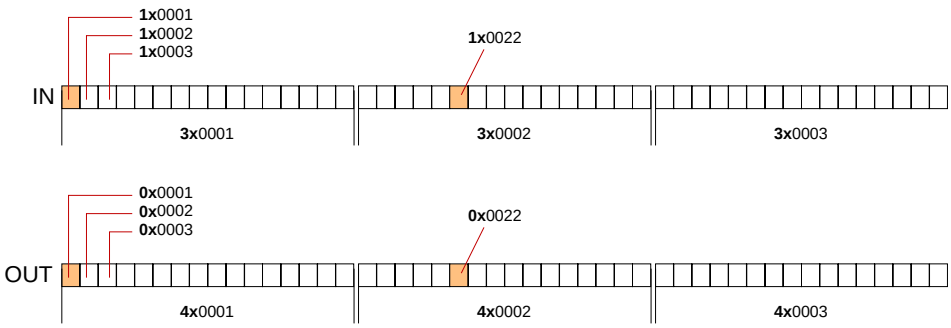
For the CPU 21xSER-1 from VIPA is not differentiating digital and analog data, the following assignment is valid:

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

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

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

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



A description of the function codes follows below.

Overview

The following Modbus function codes are supported by the Modbus slave:

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

Note!

The telegrams are automatically integrated into the according checksum circle of ASCII res. RTU.

Always valid for the Byte sequence in a word is:

1 Word	
High Byte	Low Byte

Respond of the slave

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

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

Read n Bits 01h, 02h

This function enables the reading from a slave bit by bit.

Command telegram

RTU/ASCII frame	Slave address	Function code	Address 1st Bit	Number of Bits	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	1Word

Respond telegram

RTU/ASCII frame	Slave address	Function code	Number of read Bytes	Data 1st Byte	Data 2nd Byte	...	RTU/ASCII frame
	1Byte	1Byte	1Byte	1Byte	1Byte max. 252Byte		1Word

Read n Words 03h, 04h

This function allows to read the slave word by word.

Command telegram

RTU/ASCII-frame	Slave-address	Functions code	Address 1st Word	Number of words	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	1Word

Respond telegram

RTU/ASCII frame	Slave address	Functions code	No. of read Bytes	Data 1st Word	Data 2nd Word	...	RTU/ASCII frame
	1Byte	1Byte	1Byte	1Word	1Word		1Word
				max. 125 Words			

Write 1 Bit 05h

This function allows to alter a Bit in your slave. A status change happens via "Status Bit" with the following values:

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

Command telegram

RTU/ASCII frame	Slave address	Function code	Address Bit	Status Bit	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	1Word

Respond telegram

RTU/ASCII frame	Slave address	Function code	Address Bit	Status Bit	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	1Word

Write 1 word 06h

This function sends a word to the slave. This allows to overwrite a register in the coupler.

Command telegram

RTU/ASCII frame	Slave address	Function code	Address Word	Value Word	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	

Respond telegram

RTU/ASCII frame	Slave address	Function code	Address Word	Value Word	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	

Write n Words 10h

This function allows you to send n words to the slave.

Command telegram

RTU/ASCII frame	Slave address	Functions code	Address 1st Word	Number of words	Number of Bytes	Data 1st Word	Data 2nd Word	...	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	1Byte	1Word	1Word	1Word	1Word
						max. 125Words			

Respond telegram

RTU/ASCII frame	Slave address	Functions code	Address 1st Word	Number of words	RTU/ASCII frame
	1Byte	1Byte	1Word	1Word	1Word

Modbus – Example communication

Outline

The example establishes a communication between a master and a slave via Modbus. The following combination options are shown:

Modbus master (M)	Modbus slave (S)
CPU 21xSER-1	CPU 21xSER-1
CPU 21xSER-1	CP 240

M: CPU 21xSER-1
S: CPU 21xSER-1

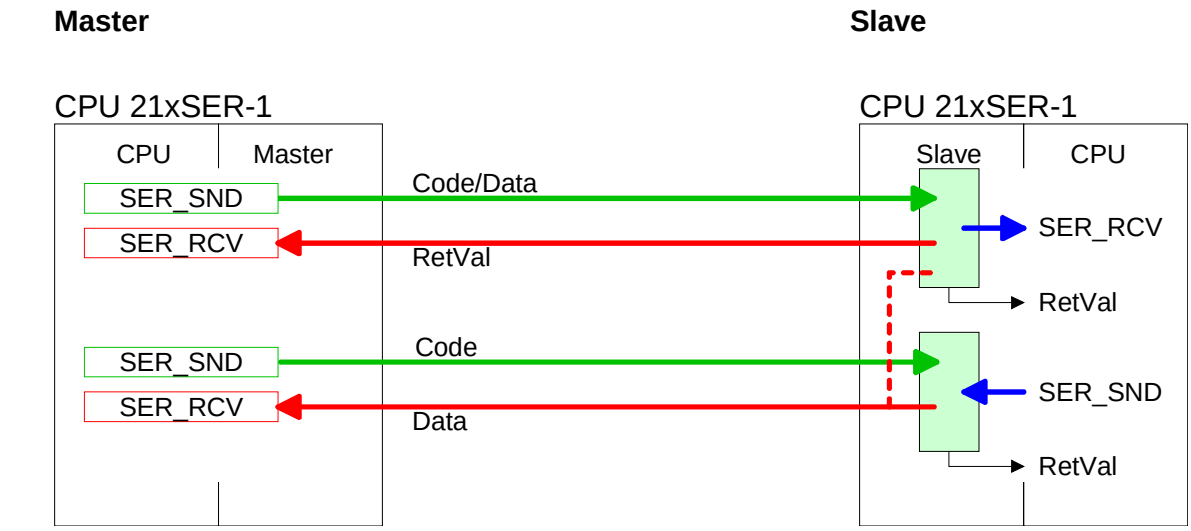
The following components are required for this example:

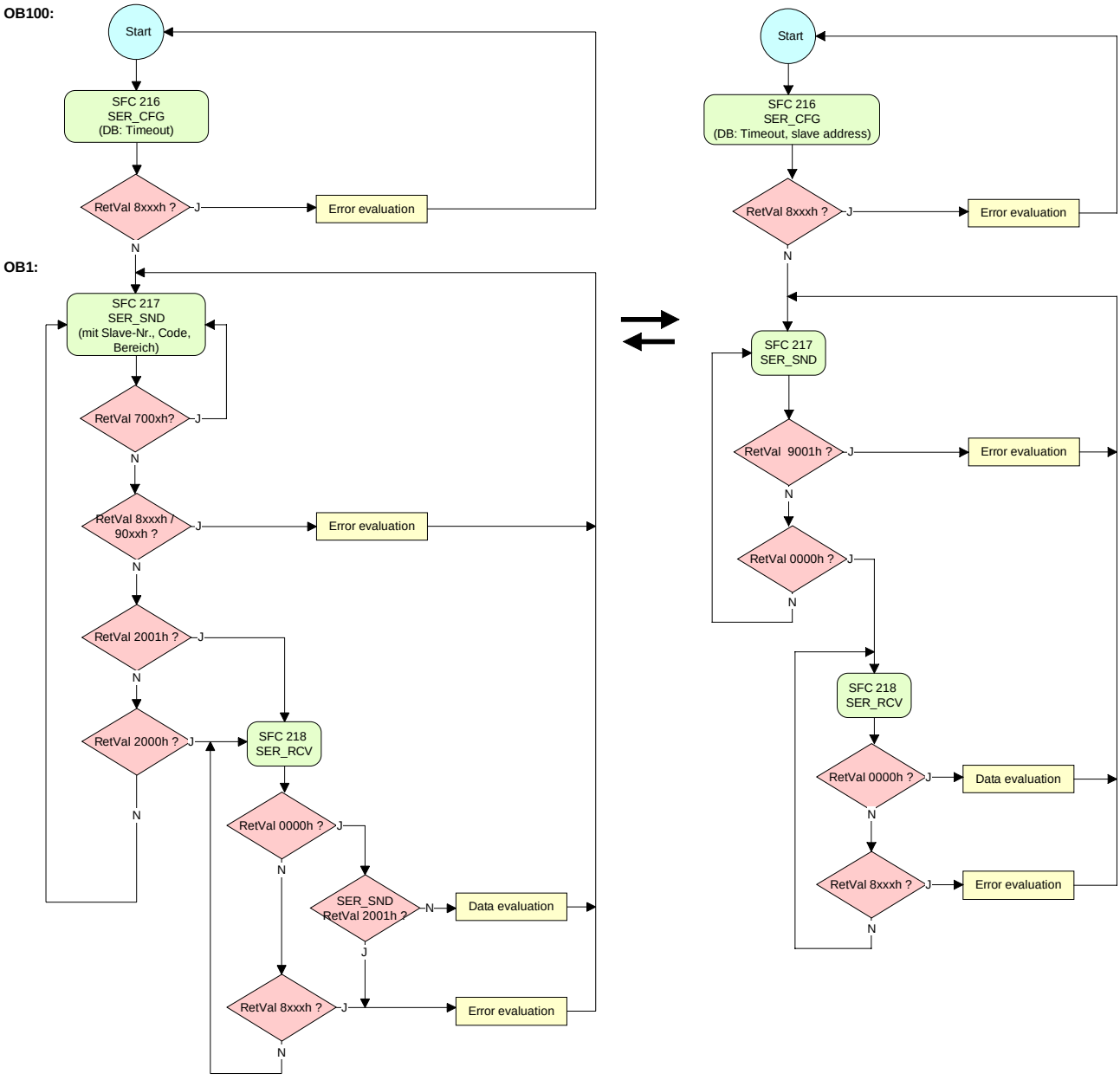
- 2 CPU 21xSER-1 as Modbus RTU master res. Modbus RTU slave
- Siemens SIMATIC Manager and possibilities for the project transfer
- Modbus cable connection

Approach

- Assemble a Modbus system consisting of a CPU 21xSER-1 as Modbus master and a CPU 21xSER-1 as Modbus slave and Modbus cable.
- Execute the project engineering of the master!
  For this you create a PLC user application with the following structure:
  - OB 100: Call SFC 216 (configuration as Modbus RTU master) with timeout setting and error evaluation.
  - OB 1: Call SFC 217 (SER_SND) where the data is send with error evaluation. Here you have to build up the telegram according to the Modbus rules.
    Call SFC 218 (SER_RECV) where the data is received with error evaluation.
- Execute the project engineering of the slave!
  The PLC user application at the slave has the following structure:
  - OB 100: Call SFC 216 (configuration as Modbus RTU slave) with timeout setting and Modbus address in the DB and error evaluation.
  - OB 1: Call SFC 217 (SER_SND) for data transport from the slave CPU to the output buffer.
    Call SFC 218 (SER_RECV) for the data transport from the input buffer to the CPU. Allow an according error evaluation for both directions.

The following page shows the structure for the according PLC programs for master and slave.


Master
Slave
CPU 21xSER-1
CPU
Master
SER_SND
SER_RCV
Code/Data
RetVal
Code
Data
CPU 21xSER-1
Slave
CPU
SER_RCV
RetVal
SER_SND
RetVal


OB100:
Start
SFC 216
SER_CFG
(DB: Timeout)
RetVal 8xxxh ?
J
Error evaluation
N
OB1:
SFC 217
SER_SND
(mit Slave-Nr., Code,
Bereich)
RetVal 700xh?
J
N
RetVal 8xxxh /
90xxh ?
J
Error evaluation
N
RetVal 2001h ?
J
N
RetVal 2000h ?
J
SFC 218
SER_RCV
N
RetVal 0000h ?
J
N
SER_SND
RetVal 2001h ?
N
Data evaluation
J
RetVal 8xxxh ?
J
Error evaluation
N
Start
SFC 216
SER_CFG
(DB: Timeout, slave address)
RetVal 8xxxh ?
J
Error evaluation
N
SFC 217
SER_SND
RetVal 9001h ?
J
Error evaluation
N
RetVal 0000h ?
J
N
SFC 218
SER_RCV
RetVal 0000h ?
J
Data evaluation
N
RetVal 8xxxh ?
J
Error evaluation
N

M: CPU 21xSER-1
S: CP 240

The following components are required for the example:

- 1 CPU 21xSER-1 as Modbus RTU master
- 1 System 200V with CP 240 as Modbus RTU slave
- Siemens SIMATIC Manager and options for project transfer
- Modbus cable connection

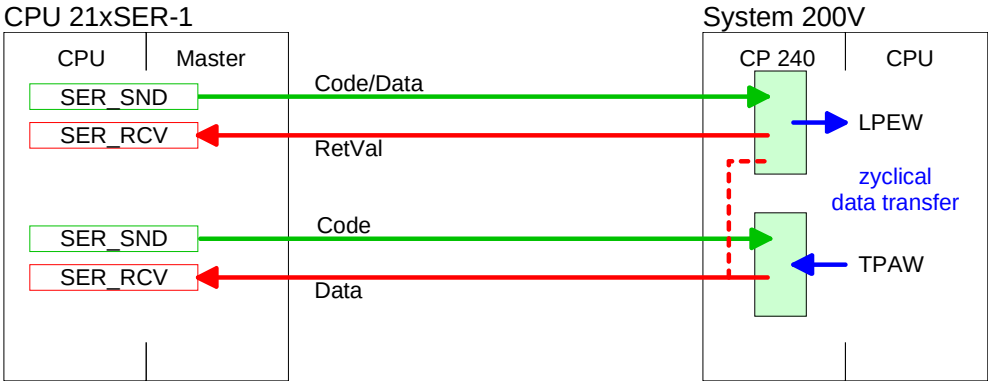
Approach

- Execute the project engineering for the master!
  Configure the master like the CP 240 slave (see structure above)!
- Execute the project engineering of the CP 240 slave!

  The parameterization of the CP 240 happens via the hardware configuration. Here you set for the input and output range the start address from where on the fixed length of 16Byte for in- and output are stored in the peripheral area of the CPU.

  The data transfer via Modbus does not require a PLC application. You only have to make sure that the data received by the master are evaluated in the CPU and that the data that is to be transferred to the master are stored in the output area. This is reached by transferring the according word of the process image cyclically.

  The following picture illustrates this:



The course of a communication

This is the course a communication between master and slave happens:

Master Send block as DB10

Using this Send DB the master sends 16Byte user data to the slave with address 5:

DB10.DBD 0	**DW#16#05100000**	**Command telegram**
	with 05 →	slave address 05h
	10 →	function code 10h (write n words)
	0000 →	Offset 0000h
DB10.DBD 4	**DW#16#000810A0**	**Command telegram + 1 data byte**
	with 0008 →	Word count 0008h
	10 →	Byte count 10h
	A0 →	Start 16Byte 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**	**data byte 14 ... 16 + 1Byte not used**
	with ADAEAF →	data byte 14 ... 16
	00 →	not used by the module

Master Receive block as DB11

If there is no error the following data are transferred to the master by the slave:

DB11.DBD 0	**DW#16#05100000**	**Respond telegram**
	with 05 →	slave address 05h
	10 →	function code 10h (no error)
	0000 →	Offset 0000h
DB11.DBD 4	**DW#16#000810A0**	**Respond telegram + 1 Data byte**
	with 0008 →	Word count 0008h
	10 →	Byte count 10h
	00 →	Start 16Byte data with 00h (irrelevant for 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**	**data byte 14 ... 16 + 1 Byte not used**
	with ADAEAF →	data byte 14 ... 16
	00 →	not used by the module

Receive block with error respond

The communication by Modbus knows 2 kinds of error:

- Slave doesn't respond to a master command

  When the slave is not reacting within the defined timeout period, the master writes the following error message into the receive block:

  ERROR01 NO_DATA. In hex notation the following values are shown:

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 scheduling)

- Slave answers with an error message

  If the slave replies an error, the function code with 80h is send back marked with an OR.

DB11.DBD 0	**DW#16#05900000**	**Respond telegram**
	with 05 →	slave address 05h
	90 →	function code 90h (error message because 10h OR 80h = 90h)
	0000 →	rest data is irrelevant because an error was announced

Chapter 10 Deployment CPU 21xSER-2

Overview

Content of this chapter is the deployment of the CPU 21x-2BS02 with two RS232C interfaces.

Here you'll find all information about the transfer protocols of the CPU 21x-2BS02.

The following description includes:

- Deployment of the RS232C interface
- Usage of the protocols ASCII, STX/ETX, 3964(R) and RK512
- Parameterization

Content

Principles

General

The CPU 21x-2BS02 provide serial interfacing facilities between the processes of different source and destination systems. The CPU has got 2 serial RS232C interfaces.

The communication happens via handling blocks that are stored in the CPU as library.

Protocols

The CPU 21x supports the ASCII, STX/ETX, 3964(R) and RK512 protocols.

Parameterization

For the parameter transfer to the communication processor (CP), you have to execute SEND (SFC 230) with order no. 201 during runtime. For this you store the parameters in a DB where the structure depends on the wanted protocol.

To activate the parameters, you execute a RESET (SFC 234) with order no. 0 after the SEND.

Note!

Please regard that the commands SEND, RECEIVE, FETCH and RESET require a preceding "VKE"=1 otherwise they are not executed.

Communication

The internal CP of the CPU 21x-2BS02 is directly connected to the CPU portion via a Dual-Port-RAM, also called "page frame". This page frame is available at the CPU section as standard CP interface. The data transfer happens via the standard handling blocks (SEND, RECEIVE and FETCH).

The communication via the according protocols is controlled by connection commands that are programmed in the user application.

The following SFCs are deployed:

SFC	Name	Description
SFC 230	Send	Send via page frame (page frame comm.)
SFC 231	Receive	Receive via page frame (page frame comm.)
SFC 232	Fetch	Fetch via page frame (page frame comm.)
SFC 233	Control	Control for page frame communication
SFC 234	Reset	Reset for page frame communication
SFC 235	Synchron	Synchron for page frame communication
SFC 236	Send_All	Send_All via page frame (page frame comm.)
SFC 237	Recv_All	Receive_All via page frame (page frame com.)
SFC 238	Control1	Control for page frame communication with Type ANZW: pointer and parameter IND.

Protocols and Procedures

Übersicht

The CPU 21x-2BS02 supports the following protocols and Procedures:

- ASCII communication
- STX/ETX
- 3964(R) mit RK512

ASCII

ASCII data communication is one of the simple forms of data exchange. Incoming characters are transferred 1 to 1.

To ensure that messages can be divided into logical parts the 'character delay time‘ (ZVZ) of the receiver must be matched by the transmitter. The ZVZ is specified in milliseconds (ms) and it must be larger than or equal to 2ms.

On the transmitter the equivalent of the receiver's character delay time is the 'time from reception of job ‘ (ZNA). These two times can be used to establish a simple serial PLC communication path. Any transmit job is only acknowledged with 'job completed without errors' (AFOF) when the data was transmitted and the ZNA has expired.

If ZNA is set to 0, the send sequence has to be controlled via the user application.

STX/ETX

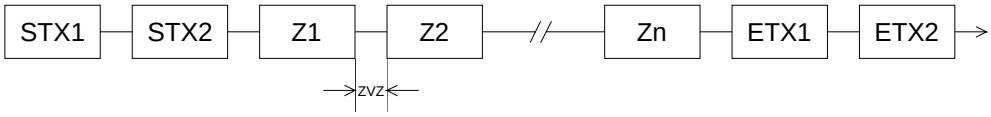
STX/ETX is a simple protocol with start and end ID, where STX stands for **S**tart of **Tex**t and ETX for **E**nd of **Tex**t.

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 Start followed by the data characters and the end character.

The effective data which includes all the characters between Start and End are transferred to the CPU when the End has been received.

When data is send from the CPU to a peripheral device, any user data is handed to the CPU 21x-2BS02 where transferred to the communication partner.

Message structure:



You may define up to 2 start and end characters.

You may work with 1, 2 or no Start- and with 1, 2 or no End-ID. As Start- res. End-ID all Hex values from 01h to 1Fh are permissible. Characters above 1Fh are ignored. In the user data, characters below 20h are not allowed and may cause errors. The number of Start- and End-IDs may be different (1 Start, 2 End res. 2 Start, 1 End or other combinations). If no End-ID is defined, all read characters are transferred to the CPU after a parameterizable character delay time (Timeout).

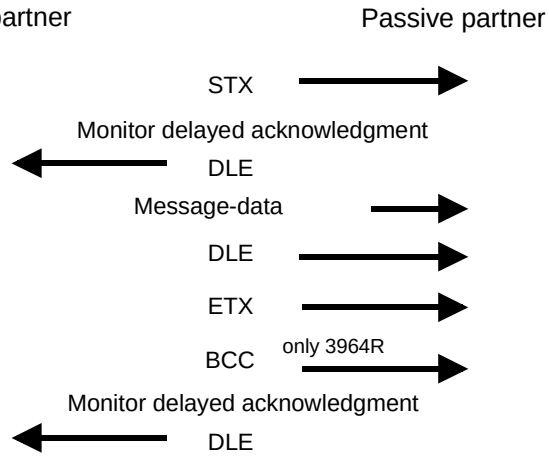
3964(R)

The 3964(R) procedure controls the data transfer of a point-to-point link between the CPU 21x-2BS02 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 **S**tart of **Te**x**t**
- DLE **D**ata **L**ink **E**scape
- ETX **E**nd of **Te**x**t**
- BCC **B**lock **C**heck **C**haracter (only at 3964R)
- NAK **N**egative **Ac**k**nowledge**

Procedure



You may transfer a maximum of 255Byte per message.

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.

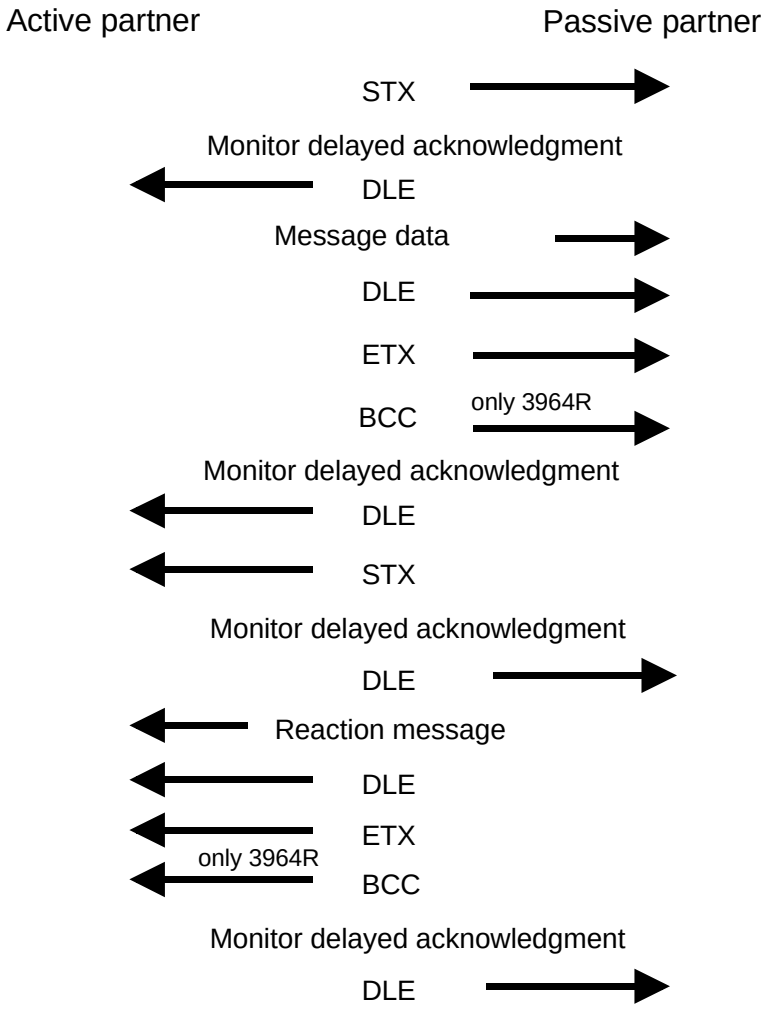
The 3964R procedure requires that a lower priority is assigned to the communication partner. When communication partners issue simultaneous send commands, the station with the lower priority will delay its send command.

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

The following paragraph describes the structure of the procedure and messages:



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.

If you want to transfer telegrams without coordination flag, you have to set FFFFh.

Timeout times

The following time-outs apply :

Ack-overdue-time:	(QVZ)	=	2000 ms
Character-overdue-time:	(ZVZ)	=	220 ms

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 3*) times after which a NAK is transmitted and the attempt is terminated.

*) adjustable via parameter

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 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.

Block-Check-character (BCC-Byte)

The 3964R procedure appends a Block check character 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.

DLE

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

RS232C interface

Properties

- Interface compatible to the COM interface of a PC
- Protocols supported: ASCII, STX/ETX, 3964(R) and RK512
- receive buffer of 256Byte and send buffer with 256Byte.
- The maximum telegram length is 255Byte

Properties of the RS232C interface

- Logical signals as voltage levels
- Point-to-point links with serial full-duplex transfer in 2-wire technology with data transfers over distances of up to 15m (only at RS232C)
- Data transfer rate up to 115,2kBaud

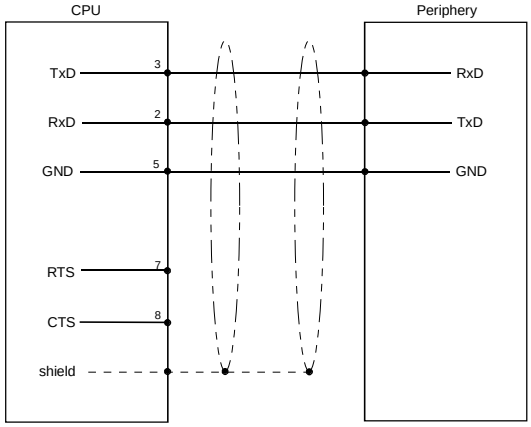
Connection interface

Via 9pin jack, you may establish a serial point-to-point connection.

9pin jack

Pin	Description
1	CD-
2	RxD
3	TxD
4	DTR-
5	GND
6	DSR-
7	RTS-
8	CTS-
9	RI-

Connection RS232C



The CPU 21x-2BS02 currently supports the following RS232C signals:

TxD Transmit Data

The transmit data is transferred via the TxD line. When the transmit line is not used the CPU 21x-2BS02 holds it at a logical "1".

RxD Receive Data

The receive data arrives via the RxD line. When the receive line is not in use, it must be held at a logical "1" by the transmitting station.

Communication

Overview

Data communication is controlled by means of the handler blocks.

The CPU decides the type of data transfer depending on the parameterization. The standard modes use the SEND/RECEIVE blocks for order initialization and the "ALL" blocks for the user data communication.

The following blocks are supplied:

SFC	Label	
SFC 235	SYNCHRON	Synchronization between CPU and CP and presetting of the block size
SFC 230	SEND	Initialize a send order
SFC 236	SEND-ALL	Send user data
SFC 231	RECEIVE	Initialize a receive order
SFC 237	RECEIVE-ALL	Receive user data
SFC 233	CONTROL	Block for communication control
SFC 234	RESET	Deletes all orders and activates new parameter

Programming

The following text shows the approach of programming in easy steps:

Start-up OB100:

- Call SYNCHRON with SFC 235 and enter the wanted block size (Page frame basic address=0, Block size, PAFE)
- Parameterize the interfaces with SEND (SFC 230) with order no. 201 and parameter DB
- To take over the parameters, you call RESET (SFC 234) with order no. 0

Cycle OB1:

- Create SEND and RECEIVE orders for send and receive initialization
- Create SEND ALL and RECEIVE ALL orders for user data transfer

A more detailed description follows.

Initialize interfaces

Overview

The initialization of the interfaces happens in OB 100 and should be executed with the following approach:

- Call SYNCHRON with SFC 235 and enter the wanted block size (page frame basic address=0, block size, PAFE)
- Parameterize the interfaces via SEND (SFC 230) with order no. 201 and parameter DB
- To take over the parameters, call RESET (SFC 234) with order no. 0

SFC 235 SYNCHRON

The SYNCHRON block initializes the synchronization between CPU and CP during the boot process and for this it has to be called in the start-up OB 100. Simultaneously the transition area of the interface is deleted and predefined and the CP and the CPU agree about the block size.

Parameter

Address	Declaration	Name	Type	Initial val	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	BLGR	INT		Blocksize
4.0	out	PAFE	BYTE		Error indicator for configuration errors

SSNR

Number of the logical interface (page frame address) to which the according order refers to. SSNR must be 0!

Block size

To avoid long cycle run-times it is convenient to split large data amounts into smaller blocks for transmitting them between CP and CPU. You declare the size of this blocks by means of „block size“.

A large block size = high data throughput, but also longer run-times and therefore a high cycle time strain.

A small block size = smaller data throughput, but also shorter run-times of the blocks.

Following block sizes are available:

Value	*Block size*	*Value*	*Block size*
0	Default (64Byte)	4	128Byte
1	16Byte	5	256Byte
2	32Byte	6	512Byte
3	64Byte	255	512Byte

Parameter type : Integer

Valid range : 0 ... 255

Example

```
CALL  SFC  235
SSNR:=0
BLGR:=6
PAFE:=MB199
```

SFC 230 - SEND with ANR=201 and Parameter-DB

You may transfer parameters to the CP via SEND (SFC 230), ANR=201 and DB.

Please regard that a send command is only executed when the following conditions are met:

- the SEND has received a VKE "1"
- the Bit "order in process" in the indicator word has been reset

Parameter

Address	Declaration	Name	Type	Init	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Mode of Adressing (direct / indirect)
6.0	in	QANF	ANY		Pointer to data source
16.0	out	PAFE	BYTE		Error indicator for configuration errors
18.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

SFC 234 - RESET

A RESET with order number 0 interrupts all orders and the previous parameters are activated.

Analog to SEND, the block requires a preceding VKE=1.

Parameter

Address	Declaration	Name	Type	Initial	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	out	PAFE	BYTE		Error indicator for configuration errors

Example OB100

```
O     M      0.0
ON    M      0.0              VKE=1
CALL  SFC  230                SEND
SSNR:=0
ANR :=201                     ID for parameterization
IND :=0
QANF:=P#DB9.DBX0.0 BYTE 20    Pointer to parameter for COM1
PAFE:=MB198
ANZW:=MD200

O     M      0.0
ON    M      0.0              VKE=1
CALL  SFC  230                SEND
SSNR:=0
ANR :=201                     ID for parameterization
IND :=0
QANF:=P#DB9.DBX20.0 BYTE 20   Pointer to parameter for COM2
PAFE:=MB198
ANZW:=MD200

O     M      0.0
ON    M      0.0              VKE=1
CALL  SFC  234                RESET
SSNR:=0
ANR :=0                       ID for RESET
PAFE:=MB197
```

Interface parameters

Parameter-DB structure

The parameter transfer to the communication processor happens during runtime by using the SFC 230 with order no. 201. The parameters for the protocols have to be stored in a DB.

To activate the parameters you have to execute a RESET with SFC 234 after transfer.

General parameters for every channel in use:

Data byte	Type	Designator		Values	Default
0	BYTE	Channel	COM 1 COM 2	1 2	
1	BYTE	Mode	MODI_NONE (deactivated) MODI_1 (Parameter following)	0 81h	0
2	BYTE	Baudrate	BAUDRATE_DEF BAUDRATE_150 BAUDRATE_300 BAUDRATE_600 BAUDRATE_1K2 BAUDRATE_1K8 BAUDRATE_2K4 BAUDRATE_4K8 BAUDRATE_7K2 BAUDRATE_9K6 BAUDRATE_14K4 BAUDRATE_19K2 BAUDRATE_38K4 BAUDRATE_57K6	0 1 2 3 4 5 6 7 8 9 10 11 12 13	9
3	BYTE	DataBits	DATABIT_5 DATABIT_6 DATABIT_7 DATABIT_8	0 1 2 3	3
4	BYTE	Parity	PARITY_NONE PARITY_ODD PARITY_EVEN	0 1 3	0
5	BYTE	StopBits	STOPBIT_1 STOPBIT_1_5 STOPBIT_2	1 2 3	1
6	BYTE	FlowControl	FLOW_NONE FLOW_HARDWARE FLOW_XON_XOFF	0 1 2	1
7	BYTE	Protocol	PROTOCOL_ASCII PROTOCOL_STXETX_HTB PROTOCOL_3964 PROTOCOL_3964R PROTOCOL_3964_RK512 PROTOCOL_3964R_RK512	01h 02h 03h 04h 05h 06h	01h

Additional parameters depending on the protocol

Depending on the selected protocol the following parameters must also be specified in the DB:

for PROTOCOL_ASCII:

Data byte	Type	Designator	Values	Default
Transmit channel				
8, 9	WORD	BufAnz	1..n	1
10,11	WORD	BufSize	16..1024	256
12, 13	WORD	ZNA, time delay after job	0..n	500
Receive channel				
14, 15	WORD	BufAnz	1..n	1
16,17	WORD	BufSize	16..1024	256
18, 19	WORD	ZVZ, character delay time	2..n	200

for PROTOCOL_STXETX:

Data byte	Type	Designator	Values	Default
Transmit channel				
8, 9	WORD	BufAnz	1..n	1
10, 11	WORD	BufSize	16..1024	256
12,13	WORD	ZNA, time delay after job	0..n	0
Start code				
14, 15	WORD	Quantity	1, 2	1
16	BYTE	Code 1	0..255	STX
17	BYTE	Code 2	0..255	STX
End code				
18, 19	WORD	Quantity	1, 2	1
20	BYTE	Code 1	0..255	ETX
21	BYTE	Code 2	0..255	ETX
Receive channel				
22, 23	WORD	BufAnz	1..n	1
24, 25	WORD	BufSize	16..1024	256
26, 27	WORD	TMO, Timeout	2..n	200
Start code				
28, 29	WORD	Quantity	1, 2	1
30	BYTE	Code 1	0..255	STX
31	BYTE	Code 2	0..255	STX
End code				
32, 33	WORD	Quantity	1, 2	1
34	BYTE	Code 1	0..255	ETX
35	BYTE	Code 2	0..255	ETX

for *PROTOCOL_3964(R):*

Data byte	Type	Designator	Values	Default
Transmit-/ receive channel				
8, 9	WORD	BufAnz	1..n	1
10, 11	WORD	BufSize	16..1024	128
12, 13	WORD	ZNA, time delay after job	0..n	0
14, 15	WORD	ZVZ char. overdue time	1..n	200
16, 17	WORD	QVZ ack. overdue time	1..n	500
18, 19	WORD	BWZ block delay time (	1..n	10000
20, 21	WORD	STX number of retries connection set-up	1..n	3
22, 23	WORD	DBL number of retries data blocks	1..n	6
24, 25	WORD	Priority 0==Low, >0==High	0, 1	1

wenn PROTOCOL_3964(R)_RK512:

Data byte	Type	Designator	Values	Default
Transmit-/ receive channel				
8, 9	WORD	BufAnz	1..n	1
10, 11	WORD	BufSize	16..1024	128
12, 13	WORD	ZNA, time delay after job	0..n	0
14, 15	WORD	ZVZ char. overdue time	1..n	200
16, 17	WORD	QVZ ack. overdue time	1..n	500
18, 19	WORD	BWZ block delay time	1..n	10000
20, 21	WORD	STX number of retries connection set-up	1..n	3
22, 23	WORD	DBL number of retries data blocks	1..n	6
24, 25	WORD	Priority 0==Low, >0==High	0, 1	1
26, 27	WORD	QVZ for user acknowledgement	1..n	5000

Constant parameters

The following parameters are constants and can not be changed

Parameter	Setting
Page frame base address	0
No. of page frames	1
Job-no.	1: COM 1 SEND 2: COM 1 RECEIVE 3: COM 2 SEND 4: COM 2 RECEIVE 201: for SEND for parameterization
Job priority	2

Interface communication

Overview

The communication happens via the following handling blocks in the OB1:

No.	Designator	
SFC 230	SEND	Initialize a send order
SFC 236	SEND-ALL	Send user data
SFC 231	RECEIVE	Initialize a receive order
SFC 237	RECEIVE-ALL	Receive user data
SFC 233	CONTROL	Block for communication control
SFC 234	RESET	Deletes all orders and activates new parameters

Cycle OB1:

- Create SEND and RECEIVE orders for send and receive initialization
- Create SEND ALL and RECEIVE ALL orders for user data transfer

The following section contains a summary of this blocks. More detailed information is to find in the chapter "Integrated OBs, SFBs, SFCs".

SFC 230 - SEND

The SEND block serves the initialization of a send order to a CP.

Please regard that a send order is only executed when the following conditions are met:

- the SEND has received a VKE "1"
- the Bit "order in process" in the indicator word has been reset

Parameter

Address	Declaration	Name	Type	Init	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Mode of Adressing (direct / indirect)
6.0	in	QANF	ANY		Pointer to data source
16.0	out	PAFE	BYTE		Error indicator for configuration errors
18.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

SEND_ALL for data transmission

If the CP is able to take over the data directly, the SEND block transfers the requested data in one session. If the CP requests only the order parameters or the amount of the depending data is too large, the CP only gets the sending parameters res. the parameter with the first data block. The according data res. the assigned serials blocks for this order are requested from the CP by SEND_ALL to the CPU. For this it is necessary that the block SEND_ALL is called minimum one time per cycle.

SFC 231 - RECEIVE

The RECEIVE block receives data from a CP.

Normally the RECEIVE block is called in the cyclic part of the user application program.

For activating a RECEIVE block is only started, if:

- the RECEIVE has received a VKE "1"
- the CP released the order (Bit "Handshake convenient" = 1)

Parameter

Address	Declaration	Name	Type	Initi	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Mode of Adressing (direct / indirect)
6.0	in	ZANF	ANY		Pointer to data destination
16.0	out	PAFE	BYTE		Error indicator for configuration errors
18.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

SFC 237 - RECEIVE_ALL

Via the RECEIVE_ALL block, the data received from the CP is transmitted from the CP to the CPU by using the declared block size.

Location and size of the data area that is to transmit with RECEIVE_ALL, must be declared before by calling RECEIVE.

In the indicator word that is assigned to the concerned order, the bit "ENABLE/DISABLE" is set, "Data transition starts" and "Data transition/fetch running" is analyzed or altered. The receiving amount is shown in the following word.

Parameter

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	out	PAFE	BYTE		Error indicator for configuration errors
4.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

Note!

In the following cases, the RECEIVE_ALL command has to be called for minimum one time per cycle of the block OB1:

- if the CP should send data to the CPU independently
- if a CP order is initialized via RECEIVE, but the CP still has to request the background communication data of the CPU for this order.
- if the amount of data, that should be transmitted to the CPU by this RECEIVE, is higher than the declared block size.

SFC 232 - FETCH

The FETCH block initializes a FETCH order in the partner station.

The FETCH order defines data source and destination and the data source is transmitted to the partner station.

The CPU from VIPA realizes the definition of source and destination via a pointer parameter.

The partner station provides the *Source* data and transmits them via SEND_ALL back to the requesting station. Via RECEIVE_ALL the data is received and is stored in *Destination*.

The update of the indicator word takes place via FETCH res. CONTROL.

Please regard that a fetch command is only executed when the following conditions are met:

- the FETCH has received a VKE "1"
- the Bit "order in process" in the indicator word has been reset

Parameter

Address	Declaration	Name	Type	Initia	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Mode of Adressing (direct / indirect)
6.0	in	ZANF	ANY		Pointer to data destination
16.0	out	PAFE	BYTE		Error indicator for configuration errors
18.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

SFC 233 - CONTROL

The purpose of the CONTROL block is the following:

- Update of the indicator word
- Query if a certain order of the CP is currently active, e.g. request for a receipt telegram
- Query the CP which order is recently in commission.

The block is independent from the VKE and should be called from the cyclic part of the application.

Parameter

Address	Declaration	Name	Type	Init	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	out	PAFE	BYTE		Error indicator for configuration errors
6.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

Chapter 11 Integrated OBs SFBs SFCs

Outline

Here you find the description of the integrated OBs, SFBs and SFCs of the PLC-CPUs from VIPA for STEP®7 from Siemens.

The information about the listed blocks is valid for the CPUs 11x, 21x, 31x and 51x.

Another part of the chapter are the VIPA specific SFCs that are exclusively used with CPUs from VIPA.

Note!

Please regard that some VIPA specific SFCs are only integrated in certain CPUs. For example, the SFCs for high-speed counter and pulse duration modulation are only integrated in the CPUs of the System 100V.

The assignment of the according SFCs to the CPUs is to find in the sub-chapter "VIPA specific SFCs".

Here you'll also find hints for the description of the VIPA specific SFCs.

The following text describes:

- Overview over the integrated OBs, SFBs, SFCs
- VIPA specific SFCs and their inclusion
- SFCs for access to the MMC
- SFCs for the System 100V
- SFC for access to TD 200
- SFCs for direct access to page frame and page frame communication

Content

Integrated OBs and SFBs

General

The system program of the CPU 21x offers you some additional functions, that you may use by calling FBs, FCs or OBs. Those additional functions are part of the system program and don't use any work memory. Although the additional functions may be requested, they can not be read or altered.

The calling of an additional function via FB, FC or OB is registered as block change and influences the nesting depth for blocks.

Integrated OBs

The following organization blocks (OBs) are available:

OB	Description
OB 1	Free cycle
OB 10	Clock alarm
OB 20	Delay alarm
OB 35	Prompter alarm
OB 40	Process alarm
OB 80	Cycle time exceeded or clock alarm run out
OB 82	Diagnostic alarm
OB 85	OB not available or Periphery error at update process image
OB 86	Slave failure res. restart
OB 100	Reboot
OB 121	Synchronous errors
OB 122	Periphery error at n^{th} access

Integrated SFBs

The following system function blocks (SFBs) are available:

SFB	Label	Description
SFB 0	CTU	Count forward
SFB 1	CTD	Count backwards
SFB 2	CTUD	Count forward and backwards
SFB 3	TP	Create pulse
SFB 4	TON	Create switch-on delay
SFB 5	TOF	Create switch-off delay
SFB 32	DRUM	Realization of a step sequential circuit with a max. of 16 steps

Integrated standard SFCs

Standard SFCs The following standard system functions (SFCs) are available:

SFC	Label	Description
SFC 0	SET_CLK	Set clock
SFC 1	READ_CLK	Read clock
SFC 2	SET_RTM	Set operating time counter
SFC 3	CTRL_RTM	Start/stop operating time counter
SFC 4	READ_RTM	Read operating time counter
SFC 6	RD_SINFO	Read start information of the recent OB
SFC 12	D_ACT_DP	Activation or deactivation of DP slaves
SFC 13	DPNRM_DG	Read slave diagnostic data
SFC 14	DPRD_DAT	Read consistent user data (also from DP slaves → DP master FW ≥ V3.00)
SFC 15	DPWR_DAT	Write consistent user data (also to DP slaves → DP master FW ≥ V3.00)
SFC 20	BLKMOV	Copy variable inside work memory
SFC 21	FILL	Predefine field inside work memory
SFC 22	CREAT_DB	Create data block
SFC 23	DEL_DB	Delete data block
SFC 24	TEST_DB	Test data block
SFC 28	SET_TINT	Set clock alarm
SFC 29	CAN_TINT	Cancel clock alarm
SFC 30	ACT_TINT	Activate clock alarm
SFC 31	QRY_TINT	Request clock alarm
SFC 32	SRT_DINT	Start delay alarm
SFC 33	CAN_DINT	Cancel delay alarm
SFC 34	QRY_DINT	Request delay alarm
SFC 36	MASK_FLT	Mask synchronous error events
SFC 37	DMASK_FLT	Demask synchronous error events
SFC 38	READ_ERR	Read event status register
SFC 41	DIS_AIRT	Delay alarm events
SFC 42	EN_AIRT	Cancel alarm event delay
SFC 43	RE_TRIGR	Re-trigger cycle time surveillance
SFC 44	REPL_VAL	Transfer replacement value into ACCU1
SFC 46	STP	Switch CPU in STOP
SFC 47	WAIT	Delay program processing additionally to delay time
SFC 49	LGC_GADR	Search the plug-in location concerning to a logical address
SFC 50	RD_LGADR	Search all logical addresses of a block

continued ...

... continue standard SFCs

SFC 51	RDSYSST	Read information from system state list
SFC 52	WR_USMSG	Write user entry into diagnostic buffer (sending via MPI in preparation)
SFC 54	RD_DPARM	Read predefined parameters
SFC 55	WR_PARM	Write dynamic parameters (only for analog, digital blocks, CPs / not possible via Profibus)
SFC 56	WR_DPARM	Write predefined parameters (only for analog, digital blocks, CPs / not possible via Profibus)
SFC 57	PARM_MOD	Parameterize block (only for analog, digital blocks, CPs / not possible via Profibus)
SFC 58	WR_REC	Write record set (only for analog, digital blocks, CPs / not possible via Profibus)
SFC 59	RD_REC	Read record set (only for analog, digital blocks, CPs / not possible via Profibus)
SFC 64	TIME_TICK	Read millisecond timer
SFC 65	X_SEND	Send data to external partner
SFC 66	X_RCV	Receive data from external partner
SFC 67	X_GET	Read data from external partner
SFC 68	X_PUT	Write data to external partner
SFC 69	X_ABORT	Interrupt connection to external partner

VIPA specific SFCs

General

The integrated SFCs are programmed in Assembler and for this they have a fast processing time. They do not occupy space in the internal program memory.

The integrated blocks are called in the user application.

The following pages show the VIPA specific blocks that may be called for special functions in the control application.

Assignment table SFC ←→ CPU

For not every SFC is integrated in every CPU family, this table shows the assignment between SFC and CPU.

The first number specifies the CPU family, e.g.11x means: CPU 11x from System 100V.

SFC	Label	Description	11x	21x	31x	51x
SFC 216	SER_CFG	RS232C Parameterization	✓	✓		
SFC 217	SER_SND	RS232C Send	✓	✓		
SFC 218	SER_RCV	RS232C Receive	✓	✓		
SFC 219	CAN_TLGR	CAN telegram	✓	✓		
SFC 220	MMC_CR_F	Create file on MMC	✓	✓	✓	✓
SFC 221	MMC_RD_F	Read file on MMC	✓	✓	✓	✓
SFC 222	MMC_WR_F	Write to file on MMC	✓	✓	✓	✓
SFC 223	PWM	Parameterize pulse duration modulation	✓			
SFC 224	HSC	Parameterize high-speed counter	✓			
SFC 225	HF_PWM	Parameterize HF pulse duration modulation (up to 50kHz)	✓			
SFC 227	TD_PRM	Parameterization for TD200 communication	✓	✓	✓	✓
SFC 228	RW_Kachel	Read/write page frame		✓	✓	✓
SFC 230	Send	Send via page frame (page frame comm.)		✓	✓	✓
SFC 231	Receive	Receive via page frame (page frame comm.)		✓	✓	✓
SFC 232	Fetch	Fetch via page frame (page frame comm.)		✓	✓	✓
SFC 233	Control	Control for page frame communication		✓	✓	✓
SFC 234	Reset	Reset for page frame communication		✓	✓	✓
SFC 235	Synchron	Synchron for page frame communication		✓	✓	✓
SFC 236	Send_All	Send_All via page frame (page frame comm.)		✓	✓	✓
SFC 237	Recv_All	Receive_All via page frame (page frame com.)		✓	✓	✓
SFC 238	Control1	Control for page frame communication with Type ANZW: pointer and parameter IND.		✓	✓	✓

For a better overview, in the following every SFC repeats an extract of this assignment table.

The next pages describe the VIPA specific SFCs.

Include VIPA library

Outline

The VIPA specific SFCs are included in consignment in form of libraries. The libraries are self-extracting exe-files.

When you want to use the VIPA specific SFCs, you have to import them into your project.

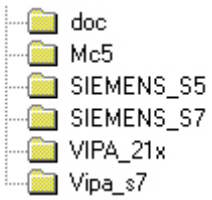
Please follow this steps:

- Execute FX00000z_Vxxx.exe to extract the library
- "De-archivate" the library
- Open the library and transfer SFCs into the project

Extract FX00000z_Vxxx.exe

The self-extraction of FX00000z_Vxxx.exe is started via a double-click on the file. Choose a destination directory and select [Extract].

The following structure is created in the destination directory:



De-archivate library

To de-archivate the SFC library you start the SIMATIC manager from Siemens. Via **File** > *De-archivate*, you open a dialog window to select an archive. The SFC library is to find in the directory structure above under VIPA_S7. The file name is VIPA.ZIP.

Choose VIPA.ZIP and click on [open].

Choose a destination directory where to store the blocks and start the extraction via [OK].

Open library and transfer SFCs into project

After the extraction, open the library.

Open your project and copy the required SFCs from the library into the directory "Blocks" of your project.

Now your user application allows the access to the VIPA specific blocks.

SFC 216 SER_CFG

11x	21x	31x	51x
✓	✓		

Description

These function works for RS232C parameterization of the CPU. The parameterization happens during runtime deploying the SFC 216 (SER_CFG). You have to store the parameters for STX/ETX and 3964R in a DB.

Parameters

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	Protocol	BYTE		1=ASCII, 2=STX/ETX, 3=3964R
2.0	in	Parameter	ANY		Pointer to protocol-parameters
12.0	in	Baudrate	BYTE		Nr for Baudrate
13.0	in	CharLen	BYTE		0=5Bit, 1=6Bit, 2=7Bit, 3=8Bit
14.0	in	Parity	BYTE		0=Non, 1=Odd, 2=Even
15.0	in	StopBits	BYTE		1=1Bit, 2=1,5Bit, 3=2Bit
16.0	in	FlowControl	BYTE		Control of RTS
18.0	out	RetVal	WORD		Error Code (0 = OK)

Protocol

Here you fix the protocol to be used. You may choose between:

1: ASCII

2: STX/ETX

3: 3964(R)

Parameter

At ASCII protocol, this parameter is ignored.

At STX/ETX and 3964R you fix here a DB that contains the communication parameters and has the following structure for the according protocols:

Data block at STX/ETX

DBB0:	STX1	BYTE	(1. Start-ID in hexadecimal)
DBB1:	STX2	BYTE	(2. Start-ID in hexadecimal)
DBB2:	ETX1	BYTE	(1. End-ID in hexadecimal)
DBB3:	ETX2	BYTE	(2. End-ID in hexadecimal)
DBB4:	TIMEOUT	WORD	(max. delay time between 2 telegrams in a time window of 10ms)

Data block at 3964R

DBB0:	Prio	BYTE	(The priority of both partners must be different)
DBB1:	ConnAttmptNr	BYTE	(Number of connection trials)
DBB2:	SendAttmptNr	BYTE	(Number of telegram retries)
DBB4:	CharTimeout	WORD	(Char. delay time in 10ms time window)
DBB6:	ConfTimeout	WORD	(Ackn. delay time in 10ms time window)

Baud rate

Velocity of data transfer in Bit/s (Baud).

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

CharLen

Number of data bits where a character is mapped to.

0: 5Bit 1: 6Bit 2: 7Bit 3: 8Bit

Parity

The parity is -depending on the value- even or odd. For parity control, the information bits are extended with the parity bit, that amends via its value ("0" or "1") the value of all bits to a defined status. If no parity is set, the parity bit is set to "1", but not evaluated.

0: NON 1: ODD 2: EVEN

StopBits

The stop bits are set at the end of each transferred character and mark the end of a character.

1: 1Bit 2: 1.5Bit 3: 2Bit

FlowControl

With this bit you affect the behaviour from signal **R**eady **t**o **s**end

"0" = RTS off "1" = RTS is "0" at Receive
RTS is "1" at Send

Note: For RS485 must FlowControl to be at "1" !

RetVal (Error message)

Error ID:

Error code	Description
0000h	no error
9001h	Parameter "Protocol" is not available (e.g. DB not loaded)
9003h	Parameter "Baud rate" is not available
9004h	Parameter "CharLength" is not available
9005h	Parameter "Parity" is not available
9006h	Parameter "StopBits" is not available
9007h	Parameter "FlowControl" is not available
8001h	Wrong parameter "Protocol"
8002h	Wrong parameter "Baud rate"
8004h	Wrong parameter "CharLength"
8008h	Wrong parameter "Parity"
8010h	Wrong parameter "StopBits"
8020h	Wrong parameter "FlowControl"
8100h	Protocol can not be initialized
F000h	No RS232C interface found

At several error messages, e.g. 8002h (Wrong "Baud rate") and 8010h (Wrong parameter "StopBits"), the error code 8012h is thrown.

SFC 217 SER_SND

11x	21x	31x	51x
✓	✓		

Description This block allows to send data from the CPU via the RS232C interface.

Parameters

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	DataPtr	ANY		Pointer to Data Buffer for sending data
10.0	out	DataLen	WORD		Length of data to send
12.0	out	RetVal	WORD		Error Code (0 = OK)

DataPtr Here you define a range of the type Pointer for the send buffer where the data that has to be send is stored. You have to set type, start and length.

Example: Data is stored in DB5 starting at 0.0 with a length of 124Byte.

DataPtr:=P#DB5.DBX0.0 BYTE 124

DataLen Word where the number of sent Bytes is stored.

At **STX/ETX** and **3964R**, the length set in DataPtr or 0 is entered.

At **ASCII**, the value may be different from the sent length when the data is sent that fast that not all data can be stored in the send buffer of 256Byte.

RetVal (Error message) Error that is thrown in case of an error:

Error code	Description
0000h	no error
9001h	Parameter "Data" is not available (e.g. DB not loaded)
9002h	Parameter "Length" is not available
7000h	Not all data has been sent (only for ASCII protocol)
8000h	Send buffer is not available (Pointer defective)
A000h	Data has not been send (output buffer overrun)
F000h	No RS232C interface found

SFC 218 SER_RCV

11x	21x	31x	51x
✓	✓		

Description This block receives data from the CPU via the RS232C interface.

Parameters

Address	Declarat	Name	Type	Initial value	Comment
0.0	in	DataPtr	ANY		Pointer to Data Buffer for received data
10.0	out	DataLen	WORD		Length of received data
12.0	out	Error	WORD		Error Number
14.0	out	RetVal	WORD		Error Code (0 = OK)

DataPtr Here you set a range of the type Pointer for the receive buffer where the reception data is stored. You have to set type, start and length.

Example: Data is stored in DB5 starting at 0.0 with a length of 124Byte.

DataPtr:=P#DB5.DBX0.0 BYTE 124

DataLen Word where the number of received Bytes is stored.

At **STX/ETX** and **3964R**, the length of the received user data or 0 is entered.

At **ASCII**, the number of read characters is entered. This value may be different from the read telegram length.

Error At ASCII, this word gets an entry in case of an error. The following error messages are possible:

Bit	Error	Description
1	overrun	Overrun when a character can not be read from the buffer fast enough
2	parity	Parity error
3	framing error	Error that shows that a defined bit frame is not met, exceeds the allowed length or contains an additional bit sequence

RetVal (Error message) Error that is thrown in case of an error:

Error code	Description
0000h	no error
9001h	Parameter "Data" is not available (e.g. DB not loaded)
9002h	Parameter "Length" is not available
9003h	Error
7000h	Receive buffer is too short (Data loss)
8000h	Receive buffer is not available (Pointer defective)
F000h	No RS232C interface found

SFC 219 CAN_TLGR

11x	21x	31x	51x
✓	✓		

SFC 219 CAN_TLGR SDO-demand on CAN-master

This block is used by the PLC to cause the CANopen master to execute a SDO read or write access).

Here you address the master via plug-in place number and the destination slave via his CAN address. The process data will be designated of index and sub index. It is possible to transfer maximum one data word process data per access via SDO.

Parameters

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	Request	BOOL		
1.0	in	Slot_Master	BYTE		
2.0	in	NodeID	BYTE		
3.0	in	Transfertyp	BYTE		
4.0	in	Index	DWORD		
8.0	in	Subindex	DWORD		
12.0	out	CanOpenError	DWORD		
16.0	out	RetVal	WORD		
18.0	out	Busy	BOOL		
20.0	in_out	DataBuffer	ANY		

Request Control parameter: 1: Start order

Slot_Master System 100V: Slot number 0: 21x-2CM02
Slot number 1 … 4: 208-1CA00

System 200V: Slot number 0: 21x-2CM02
Slot number 1 … 32: 208-1CA00

NoteID Address of the CANopen Node (1...127)

Transfertype 40h: read SDO
23h: write SDO (1 DWORD)
2Bh: write SDO (1 WORD)
2Fh: write SDO (1 BYTE)

Index CANopen Index

Subindex CANopen Subindex

CANopenError If no error occures CANopenError returns value 0.

In case of error the CANopenError contains one of the following error messages which are generated in the CAN master:

Code	Description
0x00000000	Successfully service
0x05030000	Toggle bit not alternated
0x05040000	SDO protocol timed out
0x05040001	Client/server command specifier not valid or unknown
0x05040002	Invalid block size (block mode only)
0x05040003	Invalid sequence number (block mode only)
0x05040004	CRC error (block mode only)
0x05040005	Out of memory
0x06010000	Unsupported access to an object
0x06010001	Attempt to read a write only object
0x06010002	Attempt to write a read only object
0x06020000	Object does not exist in the object dictionary
0x06040041	Object cannot be mapped to the PDO
0x06040042	The number and length of the objects to be mapped would exceed PDO length
0x06040043	General parameter incompatibility reason
0x06040047	General internal incompatibility in the device
0x06060000	Access failed due to an hardware error
0x06070010	Data type does not match, length of service parameter does not match
0x06070012	Data type does not match, length of service parameter too high
0x06070013	Data type does not match, length of service parameter too low
0x06090011	Sub-index does not exist
0x06090030	Value range of parameter exceeded (only for write access)
0x06090031	Value of parameter written too high
0x06090032	Value of parameter written too low
0x06090036	Maximum value is less than minimum value
0x08000000	general error
0x08000020	Data cannot be transferred or stored to the application
0x08000021	Data cannot be transferred or stored to the application because of local control
0x08000022	Data cannot be transferred or stored to the application because of the present device state
0x08000023	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error)

RetVal

When the function has been executed successfully, the return value contains the valid length of the respond data: 1: BYTE, 2: WORD, 4: DWORD.

Unless a SDO demand was processed error free, RetVal contains the lenght of the valid response data: 1: BYTE, 2: WORD or 4: BYTE.

If an error occurs during function processing, the return value contains an error code.

Value	Description
0xF021	Invalid slave address (Call parameter equal 0 or above 127)
0xF022	Invalid Transfer type (Value inequal 60h, 61h)
0xF023	Invalid data length (data buffer to small, at SDO read access it should be at least 4Byte, at SDO write access 1Byte, 2Byte or 4Byte).
0xF024	The SFC is not supported
0xF025	Write buffer in the CANopen master full, service can not be processed at this time.
0xF026	Read buffer in the CANopen master full, service can not be processed at this time.
0xF027	The SDO read or write access returned wrong answer, see CANopen Error Codes.
0xF028	SDO-Timeout (no CANopen participant with this Node-Id has been found).

Busy

Busy = 1: The read/write job is not yet completed.

DataBuffer

Datenbereich, über den der SFC kommuniziert

Read SDO: Destination area for the SDO data that were read.

Write SDO:Source area for the SDO data that were write.

Note

Unless a SDO demand was processed error free, RetVal contains the lenght of the valid response data in 1, 2 oder 4 byte and the CanOpenError the value 0.

SFC 220 MMC_CR_F

11x	21x	31x	51x
✓	✓	✓	✓

Description

Deploying this block, you may create a new res. access an existing file when a MMC is plugged-in.

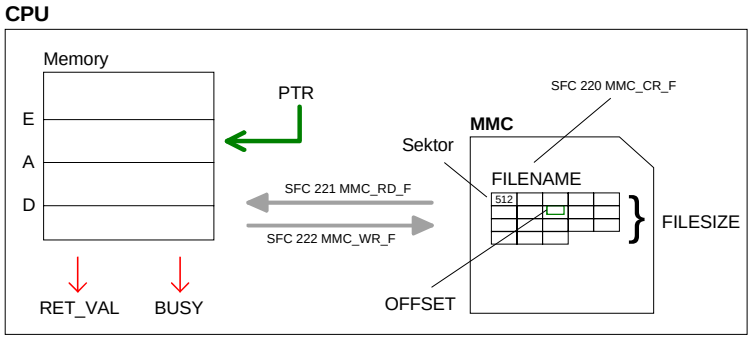
As long as you do not open another file, you may access this file via read/write commands.

Restrictions

For deploying the SFCs 220, 221 and 222, you have to regard the following restrictions:

- A read res. write access to the MMC is only possible after creation res. opening of the file via SFC 220.
- The data on MMC must not be fragmentized, for only complete data blocks may be read res. written.
- When transferring data to the MMC from an external reading device, they may be fragmentized, i.e. the data is divided into blocks. This may be avoided by formatting the MMC before the write access.
- At a write access from the CPU to the MMC, the data is always stored not fragmentized.
- When opening an already existing file, you have to use the same file name and file size that you used at creation of this file.
- A MMC is structured into sectors. Every sector has a size of 512Byte. Sector overlapping writing or reading is not possible. Access to sector overlapping data is only possible by using a write res. read command for every sector. By giving the offset, you define the according sector.

The following picture shows the usage of the single SFCs and their variables:



Note!

For read and write accesses to the MMC, you firstly have to open the file with SFC 220!

Parameters

Calling this SFC, you specify the name and the size of the file that has to be created res. to open.

When calling this SFC 220, you have to transfer the following parameters:

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	FILENAME	STRING[254]		name of the file
256.0	in	FILESIZE	DWORD		size of the file
260.0	out	RET_VAL	WORD		return value

File name

Type the file name used to store the data on the MMC. The file name may not exceed a maximum length of 8+3 signs (FAT).

File size

The file size defines the size of the user data in Byte.

Note!

When accessing an already existing file, it is mandatory to give not only the file name but also the file size. The entry of a "Joker" length is not supported at this time.

RET_VAL

Return Value

Word that returns a diagnostic/error message. 0 means OK.

See the table below for the concerning messages:

Value	Description
Diagnostic messages	
0000h	No errors (appears if new file is generated)
0001h	File already exists, is not fragmentized and the length value is identical or smaller.
8001h	No or unknown type of MMC is plugged-in.
Error messages	
8002h	No FAT on MMC found.
A001h	File name missing. This message appears if file name is inside a not loaded DB.
A002h	File name wrong (not 8.3 or empty)
A003h	File exists but "file size" too bigger than existing file.
A004h	File exists but is fragmentized and cannot be opened.
A005h	Not enough space on MMC.
A006h	No free entry in root directory. Depending on the used MMC there may be min. 16 up to max. 512 entries in the root directory.
B000h	An internal error occurred.

SFC 221 MMC_RD_F

11x	21x	31x	51x
✓	✓	✓	✓

Description

Via the SFC 221 you may read data from a MMC. For read and write accesses to the MMC, you firstly have to open the file with SFC 220 and it has to be not fragmentized.

For more detailed information to this and to the restrictions see the description of the SFC 220.

Parameters

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	PTR	ANY		variable to read
10.0	in	OFFSET	DWORD		offset in the file
14.0	out	BUSY	BOOL		busy flag
16.0	out	RET_VAL	WORD		return value

PTR

This variable of the type pointer points to a data area in the CPU where the content of the MMC has to be written to.

OFFSET

Here you define the start address inside the file on the MMC from where on the data has to be transferred to the CPU.

BUSY

During data transfer this Bit remains set. The Bit is reset as soon as the data transfer is complete.

RET_VAL

Return Value

Word, where a diagnostic/error message is returned to. 0 means OK. The following messages may be set:

Value	Description
0000h	No errors (data was read)
8001h	No or unknown type of MMC is plugged-in
8002h	No FAT found on MMC
9000h	Bit reading has been tried (Boolean variable). Bit reading is not possible.
9001h	Pointer value is wrong (e.g. points outside DB)
9002h	File length exceeded
9003h	Sector limit of 512 has been tried to overrun. Sector overrun reading is not possible.
B000h	An internal error occurred.

SFC 222 MMC_WR_F

11x	21x	31x	51x
✓	✓	✓	✓

Description

Via the SFC 222, you may write to the MMC. For read and write accesses to the MMC, you firstly have to open the file with SFC 220 and it has to be not fragmentized.

For more detailed information to this and to the restrictions see the description of the SFC 220.

Parameters

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	PTR	ANY		variable to write from
10.0	in	OFFSET	DWORD		offset in the file
14.0	out	BUSY	BOOL		busy flag
16.0	out	RET_VAL	WORD		return value

PTR

This variable of the type pointer points to a data area from where on the data starts that will be written to the MMC.

OFFSET

This defines the beginning of the data inside the file on the MMC where the data is written to.

BUSY

During data transfer this Bit remains set. The Bit is reset as soon as the data transfer is complete.

RET_VAL

Return Value

Word, where a diagnostic/error message is returned to. 0 means OK. The following messages may be set:

Value	Description
0000h	No errors
8001h	No or unknown type of MMC is plugged-in
8002h	No FAT found on MMC
9000h	Bit writing has been tried (Boolean variable). Bit writing is not possible.
9001h	Pointer value is wrong (e.g. points outside DB)
9002h	File length exceeded
9003h	Sector limit of 512 has been tried to overrun. Sector overrun reading is not possible.
B000h	An internal error occurred.

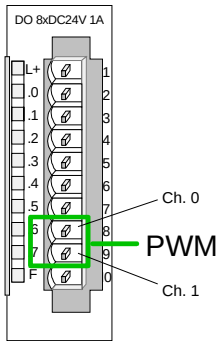
SFC 223 PWM

11x	21x	31x	51x
✓			

Description This block serves the parameterization of the pulse duration modulation for the last two output channels of X5.

Parameters

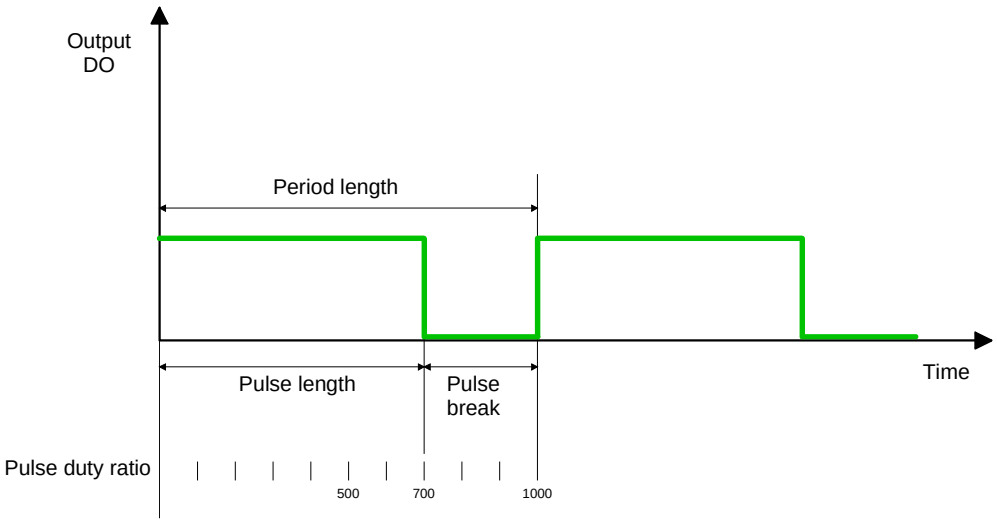
Address	Declaration	Name	Type	Initial value	Comment
0.0	in	Channel	INT		PWM Channel 0..1
2.0	in	Enable	BOOL		True = Start PWM, False = Stop PWM
4.0	in	Timebase	INT		Timebase for period/duty (0 = 0,1ms / 1 = 1,0ms)
6.0	in	Period	DINT		Period of PWM (0..60000)
10.0	in	Duty	DINT		Outputvalue per mille (0..1000)
14.0	in	MinLen	DINT		Minimum pulse (0..60000)
18.0	out	RET_VAL	WORD		Errorcode (0 = no Error)



You define a timebase, a period, the pulse duty ratio and min. pulse length. The CPU determines an pulse series with an according pulse/break relation and issues this via the according output channel.

The SFC returns a certain error code. You can see the concerning error messages in the table at the following page.

The PWM parameters have the following relationship:



Period length = timebase x period

Pulse length = (period length / 1000) x pulse duty ratio

Pulse break = period length - pulse length

The parameters have the following meaning:

Channel Define the output channel that you want to address.
Value range: 0 ... 1

Enable Via this parameter you may activate the PWM function (true) res. deactivate it (false).
Value range: true, false

Timebase Timebase defines the resolution and the value range of the pulse, period and minimum pulse length per channel.
You may choose the values 0 for 0.1ms and 1 for 1ms.
Value range: 0 ... 1

Period Through multiplication of the value defined at period with the timebase you get the period length.
Value range: 0 ... 60000

Duty This parameter shows the pulse duty ratio in promille. Here you define the relationship between pulse length and pulse break, concerned on one period.
1 Promille = 1 Timebase
If the calculated pulse duration is no multiplication of the timebase, it is rounded down to the next smaller time base limit.
Value range: 0 ... 1000

MinLen Via MinLen you define the minimal pulse length. Switches are only made, if the pulse exceeds the here fixed minimum length.
Value range: 0 ... 60000

Ret_Val Via the parameter Ret_Val you get an error number in return. See the table below for the concerning error messages:

Value	Description
0000h	no error
8005h	Parameter “MinLen” outside the permissible range
8006h	Parameter “Duty” outside the permissible range
8007h	Parameter “Period” outside the permissible range
8008h	Parameter “Timebase” outside the permissible range
8009h	Parameter “Channel” outside the permissible range
9001h	Internal error: There was no valid address for a parameter
9002h	Internal hardware error: Please call the VIPA-Service.

SFC 224 HSC

11x	21x	31x	51x
✓			

Description This SFC serves for parameterization of the counter functions (**h**igh **s**peed **c**ounter) for the first 4 inputs.

Parameters

Address	Declaration	Name	Type	Initial	Comment
0.0	in	Channel	INT		HSC Channel 0..3
2.0	in	Enable	BOOL		True = Start HSC, False = Stop HSC
4.0	in	Direction	INT		0=no counting, 1=up, 2=down
6.0	in	PresetValue	DINT		Preset Value for HSC (0..0xFFFFFFFF)
10.0	in	Limit	DINT		Endvalue for Counting up / StartValue for Counting down (0..0xFFFFFFFF)
14.0	out	RET_VAL	WORD		Errorcode (0 = no Error)
16.0	in_out	SetCounter	BOOL		True = Copy PresetValue to HSC (Bit will be reset from SFC)

Channel Type the input channel that you want to activate as counter.
Value range: 0 ... 3

Enable Via this parameter you may activate the counter (true) res. deactivate it (false).
Value range: true, false

Direction Fix the counting direction.
Hereby is: 0: Counter is deactivated, means *Enable*=false
1: count up
2. count down

PresetValue Here you may preset a counter content, that is transferred to the according counter via *SetCounter*=true.
Value range: 0 ... FFFFFFFFh

Limit Via Limit you fix an upper res. lower limit for the counting direction (up res. down). When the limit has been reached, the according counter is set zero and started new. If necessary an alarm occurs.
Value range: 0 ... FFFFFFFFh

Ret_Val

Via the parameter Ret_Val you get an error number in return. See the table below for the concerning error messages:

Value	Description
0000h	No error
8002h	The chosen channel is not configured as counter (Error in the hardware configuration)
8008h	Parameter “Direction” outside the permissible range
8009h	Parameter “Channel” outside the permissible range
9001h	Internal error: There was no valid address for a parameter
9002h	Internal hardware error: Please call the VIPA-Service.

SetCounter

Per SetCounter=true the value given by PresetValue is transferred into the according counter.

The Bit is set back from the SFC.

Value range: true, false

SFC 225 HF_PWM

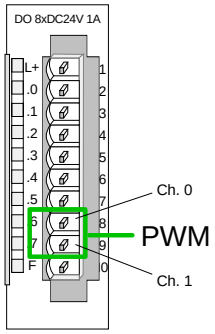
11x	21x	31x	51x
✓			

Description

This block serves the parameterization of the pulse duration modulation for the last two output channels. This block is function identical to SFC 223. Instead of timebase and period, the SFC 225 works with a predefined frequency (up to 50kHz).

Parameters

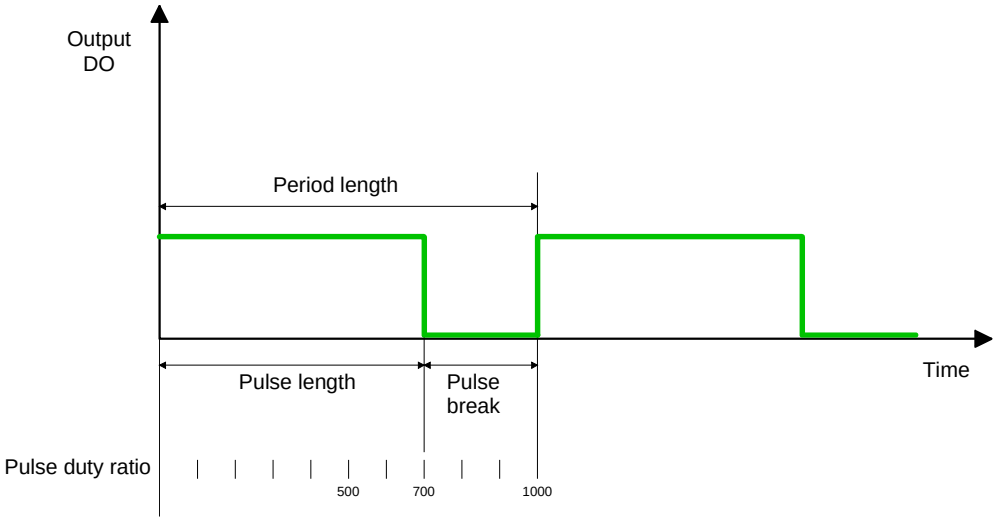
Address	Declaration	Name	Type	Initial value	Comment
0.0	in	Channel	INT		PWM Channel 0..1
2.0	in	Enable	BOOL		True = Start PWM, False = Stop PWM
4.0	in	Frequence	WORD		Frequence of HF PWM in Hz (1250 .. 50000)
6.0	in	Duty	DINT		Outputvalue per mille (0..1000)
10.0	in	MinLen	DINT		Minimum pulse (0..60000)
14.0	out	RET_VAL	WORD		Errorcode (0 = no Error)



You define a frequency, the pulse duty ratio and min. pulse length. The CPU determines an pulse series with an according pulse/break relation and issues this via the according output channel.

The SFC returns a certain error code. You can see the concerning error messages in the table at the following page.

The PWM parameters have the following relationship:



Period length = 1 / frequency

Pulse length = (period length / 1000) x pulse duty ratio

Pulse break = period length - pulse length

Channel Define the output channel that you want to address.
Value range: 0 ... 1

Enable Via this parameter you may activate the PWM function (true) res. deactivate it (false).
Value range: true, false

Frequency Type the frequency in Hz.
Value range: 2500 ... 50000

Duty This parameter shows the pulse duty ratio in Promille. Here you define the relationship between pulse length and pulse break, concerned on one period.
1 Promille = 1 Timebase
If the calculated pulse duration is no multiplication of the timebase, it is rounded down to the next smaller time base limit.
Value range: 0 ... 1000

MinLen Via MinLen you define the minimal pulse length in µs. Switches are only made, if the pulse exceeds the here fixed minimum length.
Value range: 0 ... 60000

Ret_Val Via the parameter Ret_Val you get an error number in return. See the table below for the concerning error messages:

Value	Description
0000h	no error
8005h	Parameter “MinLen” outside the permissible range
8006h	Parameter “Duty” outside the permissible range
8007h	Parameter “Period” outside the permissible range
8008h	Parameter “Timebase” outside the permissible range
8009h	Parameter “Channel” outside the permissible range
9001h	Internal error:
	There was no valid address for a parameter
9002h	Internal hardware error:
	Please call the VIPA-Service.

SFC 227 - TD_PRM

11x	21x	31x	51x
✓	✓	✓	✓

Description

The SFC 227 supports the connection of the TD200 terminal from Siemens to the VIPA CPUs.

Please regard that you have to include a data block with the TD200 configuration data before calling the SFC 227. This data block may be created with the TDWizard from VIPA. This data block contains the general settings like language and display mode and the messages that are comfortably createable with the TDWizard from VIPA. TDWizard is a part of WinPLC7 and is available from VIPA.

Parameters

The call of the SFC 227 specifies the terminal to communicate with. To call the SFC you have to transfer the following parameters:

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	MPI_ADDR	BYTE		MPI address of TD
2.0	in	TD_STRUCT_PTR	ANY		Pointer to begining of TD's data (must be in datablock)
12.0	in	FUNC_KEY_PTR	ANY		Pointer to area for keys
22.0	in	OFFSET_INPUT	BYTE		Forced values offset in input area
23.0	in	OFFSET_OUTPUT	BYTE		Forced values offset in output area
24.0	out	RET_VAL	BYTE		Errorcode (0 = no Error)

MPI_ADR

MPI address

Enter the MPI address of the connected TD200 terminal.

Parameter type: Byte

TD_STRUCT_PTR

Terminal Structure Pointer

Points to the start of the data block containing the parameterization and the text blocks of the terminal.

You may create the data block with TDWizard. This tool is a part of WinPLC7, the programming, test, diagnostic and simulation software from VIPA.

Parameter type: Pointer

Convenient range: DB

FUNC_KEY_PTR

Function Key Pointer

Points to the area where the status byte for hitten keys is stored.

Parameter type: Pointer

Convenient range: DB, M

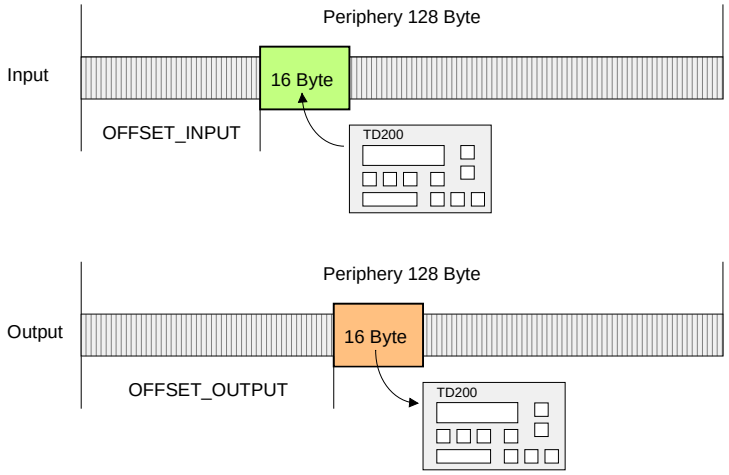
OFFSET_INPUT
OFFSET_OUTPUT

Offset Input, Offset Output

The terminal allows you to set in- and output byte. The TD200 from Siemens supports only a size of 16Byte for in- and outputs.

The CPUs from VIPA enables the access to the complete process image (each 128Byte for in- and outputs).

By setting an offset, you may overlay the periphery range of 128Byte with a window of 16Byte for in- and output. The following picture illustrates this:



RET_VAL

Return Value

Type the bit memory byte (marker byte) where the resulting message should be stored.

For specification of the (error) messages see the table below.

Messages

Value	Description
00h	no error
10h	Error at MPI_ADR
11h	MPI_ADR contains MPI address of the CPU
12h	Value in MPI_ADR exceeds max. MPI address
20h	Error in OFFSET_INPUT
21h	Value in OFFSET_INPUT is too high
30h	Error in OFFSET_OUTPUT
31h	Value in OFFSET_OUTPUT is too high
40h	Error in TD_STRUCT_PTR
41h	TD_STRUCT_PTR points not to a DB
50h	Error in FUNC_KEY_PTR
51h	Error in FUNC_KEY_PTR
52h	FUNC_KEY_PTR points not to an I, Q or M area
60h	An internal error occurred

SFC 228 - RW_KACHEL

11x	21x	31x	51x
	✓	✓	✓

Description

This SFC allows you the direct access to the page frame area of the CPU with a size of 4KByte. The page frame area is divided into four page frames, each with a size of 1KByte.

By fixing the page frame (i.e. "Kachel") no., offset and data width, the SFC 228 enables read and write access to an eligible page frame area.

Note!

This SFC has been developed for test purposes and for building-up proprietary communication systems and is completely at the user's disposal. Please regard that a write access to the page frame area influences a communication directly!

Parameter

Address	Declaration	Name	Type	Initial	Comment
0.0	in	K_Nr	INT		Interface number
2.0	in	Offset	INT		Offset in Byte
4.0	in	R_W	INT		0=Read, 1=Write
6.0	in	Size	INT		1, 2 or 4 Bytes
8.0	out	RET_VAL	BYTE		Errorcode, 0=OK
10.0	in_out	Value	ANY		Value, which is read from interface or should be written to interface

K_NR

Page frame (i.e. "Kachel") no.

Type the page frame no. that you want to access.

Value range: 0 ... 3

OFFSET

Page frame offset

Fix here an offset within the specified page frame.

Value range: 0 ... 1023

R_W

Read/Write

This parameter specifies a read res. write access.

0 = read access, 1 = write access

SIZE

Size

The size defines the width of the data area fixed via K_NR and OFFSET. You may choose between the values 1, 2 and 4Byte.

RET_VAL

Return Value

Byte where an error message is returned to.

IN_OUT

In-/output area

This parameter fixes the in- res. output area for the data transfer.

At a read access, this area up to 4Byte width contains the data read from the page frame area.

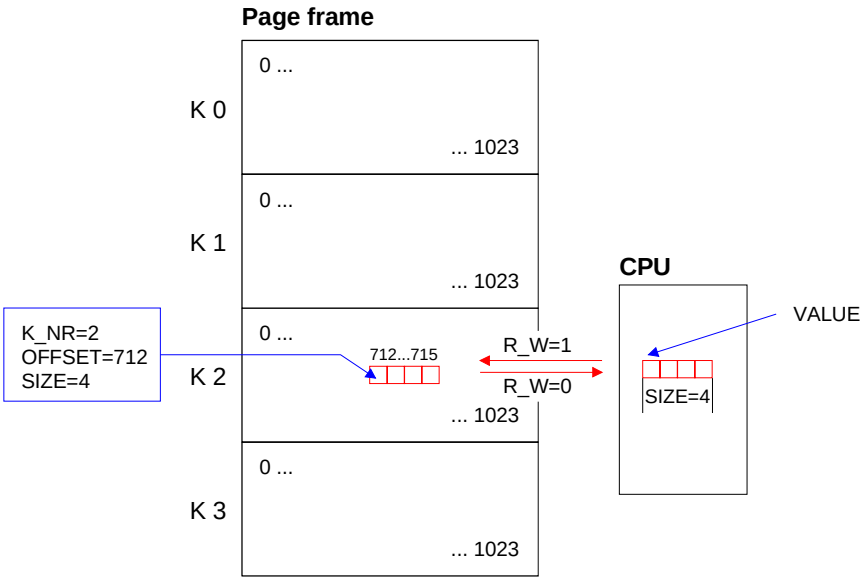
At a write access, the data up to 4Byte width is transferred to the page frame area.

Parameter type: Pointer

Example

The following example shows the read access to 4Byte starting with Byte 712 in page frame 2. The read 4Byte are stored in DB10 starting with Byte 2. For this the following call is required:

```
CALL SFC 228
K_NR      :=2
OFFSET    :=712
R_W       :=0
SIZE      :=4
RET_VAL   :=MB10
VALUE     :=P#DB10.DBX 2.0 Byte 4
```



Error messages

Value	Description
00h	no error
01h ... 05h	Internal error: No valid address found for a parameter
06h	defined page frame does not exist
07h	parameter SIZE ≠ 1, 2 or 4 at read access
08h	parameter SIZE ≠ 1, 2 or 4 at write access
09h	parameter R_W ≠ 0 or 1

Page frame communication - Parameter

11x	21x	31x	51x
	✓	✓	✓

General

The delivered handling blocks allow the deployment of communication processors in the CPUs from VIPA.

This increases the efficiency markable.

The handling blocks control the complete data transfer between CPU and the CPs.

Advantages of the handling blocks:

- you loose only few user application memory space
- short runtimes of the blocks

The handling blocks don't need:

- bit memory area
- time areas
- counter areas

Parameter description

All handling blocks described in the following use an identical interface to the user application with this parameters:

SSNR:	Interface number
ANR:	Order number
ANZW:	Indicator word (double word)
IND:	Indirect fixing of the relative start address of the data source res. destination
QANF/ZANF:	Relative start address within the type
PAFE:	Parameterization error
BLGR:	Block size

A description of these parameters follows on the next pages.

SSNR Interface number

Number of the logical interface (page frame address) to which the according order refers to.

Parameter type : Integer

Convenient range : 0 ... 255

ANR Job number

The called job number for the logical interface.

Parameter type : Integer

Convenient range : 1 ... 223

ANZW Indicator word (double word)

Address of the indicator double word in the user memory where the processing of the order specified under ANR is shown.

Parameter type : Double word

Convenient range : DW or MW; use either DW and DW+1 or MW and MW+2

The value DW refers to the data block opened before the incoming call or to the directly specified DB.

IND Kind of parameterization (direct, indirect)

This parameter defines the kind of data on which the pointer QUANF points.

0: QANF points directly to the initial data of the source res. destination data.

1: the pointer QANF/ZANF points to a memory cell, from where on the source res. destination data are defined (indirect).

2: the pointer QANF/ZANF points to a memory area where the source res. destination information lies (indirect).

5: the pointer QANF/ZANF points to a memory cell, from where on the source res. destination data and parameters of the indicator word are defined (indirect).

6: the pointer QANF/ZANF points to a memory area where the source res. destination data and parameters of the indicator word are laying (indirect).

Parameter type : Integer

Convenient entries : 0, 1, 2, 5, 6

Note!

Please regard, that at IND=5 res. IND=6, the parameter ANZW is ignored!

QANF/ZANF Relative start address of the data source res. destination and at IND=5 res. IND=6 of the indicator word.

This parameter of the type „pointer“ (Any-Pointer) allows you fix the relative starting address and the type of the data source (at SEND) res. the data destination (at RECEIVE).

At IND=5 res. IND=6 the parameters of the indicator word are also in the data source.

Parameter type : Pointer

Convenient range : DB, M, A, E

Example:
```
P#DB10.DBX0.0 BYTE 16
P#M0.0 BYTE 10
P#E 0.0 BYTE 8
P#A 0.0 BYTE 10
```

BLGR Block size

During the boot process the stations agree about the block size (size of the data blocks) by means of SYNCHRON.

A high block size = high data throughput but longer run-times and higher cycle load.

A small block size = lower data throughput but shorter run-times of the blocks.

These block sizes are available:

Value	*Block size*	*Value*	*Block size*
0	Default (64Byte)	4	128Byte
1	16Byte	5	256Byte
2	32Byte	6	512Byte
3	64Byte	255	512Byte

Parameter type : Integer

Convenient range : 0 ... 255

PAFE Error indication at parameterization defects

This "BYTE" (output, marker) is set if the block detects a parameterization error, e.g. interface (plug-in) not detected or a non valid parameterization of QUANF/ZANF.

Parameter type : Byte

Convenient range : AB 0 ... AB127, MB 0...MB 255

Page frame communication - Parameter transfer

11x	21x	31x	51x
	✓	✓	✓

Direct/indirect parameterization

A handling block may be parameterized directly or indirectly. Only the "PAFE" parameter must always been set directly.

When using the direct parameterization, the handling block works off the parameters given immediately with the block call.

When using the indirect parameterization, the handling block gets only pointers per block parameters. These are pointing to other parameter fields (data blocks or data words).

The parameters SSNR, ANR, IND and BLGR are of the type "integer", so you may parameterize them indirectly.

Example for direct and indirect parameter transfer

Direct parameter transfer

```
CALL  SFC  230
       SSNR:=0
       ANR :=3
       IND :=0
       QANF:=P#A 0.0 BYTE 16
       PAFE:=MB79
       ANZW:=MD44
```

Indirect parameter transfer

Please note that you have to load the bit memory words with the corresponding values before.

```
CALL  SFC  230
       SSNR:=MW10
       ANR :=MW12
       IND :=MW14
       QANF:=P#DB10.DBX0.0 BYTE 16
       PAFE:=MB80
       ANZW:=MD48
```

The direct res. indirect transfer of the source and destination parameters are described in the following section.

Page frame communication - Source res. destination definition

11x	21x	31x	51x
	✓	✓	✓

Outline

You have the possibility to set the entries for source, destination and ANZW directly or store it indirectly in a block to which the QANF/ZANF res. ANZW pointer points.

The parameter IND is the switch criterion between direct and indirect parameterization.

Direct parameterization of source and destination details (IND=0)

With IND=0 you fix that the pointer QANF/ZANF shows directly to the source res. destination data.

The following table shows the possible QANF/ZANF parameters at the direct parameterization:

QTYP/ZTYP Description	Data in DB	Data in MB	Data in AB Process image of the outputs	Data in EB Process image of the inputs
Pointer: Example:	`P#DBa.DBX b.0 BYTE C` `P#DB10.DBX 0.0 BYTE 8`	`P#M b.0 BYTE c` `P#M 5.0 BYTE 10`	`P#A b.0 BYTE c` `P#A 0.0 BYTE 2`	`P#E b.0 BYTE c` `P#E 20.0 BYTE 1`
DB, MB, AB, EB Definition	P#DBa "a" means the DB-No., from where the source data is fetched or where to the destination data is transferred.	P#M The data is stored in a MB.	P#A The data is stored in the output byte.	P#E The data is stored in the input byte.
Valid range for "a"	0...32767	irrelevant	irrelevant	irrelevant
Data / Marker Byte, AB, EB Definition	DB-No., where data fetch or write starts.	Bit memory byte no., where data fetch or write starts.	Output byte no., where data fetch or write starts.	Input byte no., where data fetch or write starts.
Valid range for "b"	0.0...2047.0	0...255	0...127	0...127
BYTE c Definition	Length of the Source/ Destination data blocks in Words.	Length of the Source/ Destination data blocks in Bytes.	Length of the Source/ Destination data blocks in Bytes.	Length of the Source/ Destination data blocks in Bytes.
Valid range for "c"	1...2048	1...255	1...128	1...128

Indirect parameterization of source and destination details (IND=1 or IND=2)

Indirect addressing means that QANF/ZANF points to a memory area where the addresses of the source res. destination areas and the indicator word are stored.

In this context you may either define one area for data source, destination and indicator word (IND=1) or each, data source, data destination and the indicator word, get an area of their own (IND=2).

The following table shows the possible QANF/ZANF parameters for indirect parameterization:

QTYP/ZTYP Description	IND=1	IND=2
Definition	Indirect addressing for source **or** destination parameters. The source or destination parameters are stored in a DB. QANF/ZANF DW +0 Data type source +2 DB-Nr. at type "DB", otherwise irrelevant +4 Start address +6 Length in Byte	Indirect addressing for source **and** destination parameters. The source and destination parameters are stored in a DB in a sequential order. QANF/ZANF DW +0 Data type source +2 DB-Nr. at type "DB", otherwise irrelevant (Description data source) +4 Start address +6 Length in Byte +8 Data type destin. +10 DB-Nr. at type "DB", otherwise irrelevant (Description data destination) +12 Start address +14 Length in Byte
valid DB-No.	0...32767	0...32767
Data word Definition	DW-No., where the stored data starts	DW-No., where the stored data starts
Valid range	0.0...2047.0	0.0...2047.0
Length Definition	Length of the DBs in Byte	Length of the DBs in Byte
Valid range	8 fix	16 fix

Indirect parameterization of source and destination details and ANZW (IND=5 or IND=6)

Indirect addressing means that QANF/ZANF points to a memory area where the addresses of the source res. destination areas and the indicator word are stored.

In this context you may either define one area for data source, destination and indicator word (IND=5) or each, data source, data destination and the indicator word, get an area of their own (IND=6).

The following table shows the possible QANF/ZANF parameters for indirect parameterization:

QTYP/ZTYP Description	IND=5	IND=6
Definition	Indirect addressing for source **or** destination parameters and indicator word (ANZW). The source or destination parameters and ANZW are stored in a DB in a sequential order. QANF/ZANF DW +0 Data type source +2 DB-Nr. at type "DB", otherwise irrelevant +4 Start address +6 Length in Byte +8 Data type destin. +10 DB-Nr. at type "DB", otherwise irrelevant +12 Start address Description data source/ destination Description indicator word	Indirect addressing for source **and** destination parameters and indicator word (ANZW). The source and destination parameters and ANZW are stored in a DB in a sequential order. QANF/ZANF DW +0 Data type source +2 DB-Nr. at type "DB", otherwise irrelevant +4 Start address +6 Length in Byte +8 Data type destin. +10 DB-Nr. at type "DB", otherwise irrelevant +12 Start address +14 Length in Byte +16 Data type source +18 DB-Nr. at type "DB", otherwise irrelevant +20 Start address Description data source Description data destination Description indicator word
valid DB-No.	0...32767	0...32767
Data word Definition	DW-No., where the stored data starts	DW-No., where the stored data starts
Valid range	0.0...2047.0	0.0...2047.0
Length Definition	Length of the DBs in Byte	Length of the DBs in Byte
Valid range	14 fix	22 fix

Page frame communication - Indicator word ANZW

11x	21x	31x	51x
	✓	✓	✓

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	Bit 0 ... Bit 3: Error management CPU 0: no error 1...5: CPU-Error 6...15: CP-Error Bit 4 ... Bit 7: reserved
1	*State management CPU* Bit 0: Handshake convenient (data exists) 0: RECEIVE blocked 1: RECEIVE released Bit 1: order commissioning is running 0: SEND/FETCH released 1: SEND/FETCH blocked Bit 2: Order ready without errors Bit 3: Order ready with errors *Data management handling block* Bit 4: Data receive/send is running Bit 5: Data transmission active Bit 6: Data fetch active Bit 7: Disable/Enable data block 0: released 1: blocked
2 ... 3	Length word handling block

In the „length word“ the handling blocks (SEND, RECEIVE) store the data that has already been transferred, i.e. received data in case of a RECEIVE order, send data when there is a SEND order.

The announcement in the length word is always in byte and absolute.

Error management Byte 0, Bit 0 to Bit 3

Those bits announce the error messages of the order. 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:

0 **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.

1 **wrong Q/ZTYP at HTB**

The order has been parameterized with the wrong type label.

2 **AG area not found**

The order impulse had a wrong parameterized DB-No.

3 **AG area too small**

Q/ZANF and Q/ZLAE overwrite the range boundaries. Handling with data blocks the range boundary is defined by the block size. With flags, timers, counters etc. the range size depends on the AG.

4 **QVZ-Error in the AG**

This error message means, that you chose a source res. destination parameter of the AG area, where there is either no block plugged in or the memory has a defect. The QVZ error message can only occur with the type Q/ZTYP AS, PB, QB or memory defects.

5 **Error at indicator word**

The parameterized indicator word can not be handled. This error occurs, if ANZW declared a data word res. double word, that is not (any more) in the specified data block, i.e. DB is too small or doesn’t exist.

6 **no valid ORG-Format**

The data destination res. source isn’t declared, neither at the handling block (Q/TYP="NN") nor at the coupler block.

7 **Reserved**

8 **no available transfer connections**

The capacity for transfer connections is at limit. Delete unnecessary connections.

9 **Remote error**

There was an error at the communication partner during a READ/WRITE-order.

A **Connection error**

The connection is not (yet) established. The message disappears as soon as the connection is stable. If all connections are interrupted, please check the block itself and the bus cable. Another possibility for the occurrence of this error is a wrong parameterization, like e.g. inconsistent addressing.

B **Handshake error**

This could be a system error or the size of the data blocks has been defined out of range.

C **Initial error**

The wrong handling block tried to initialize the order or the size of the given data block was too large.

D **Cancel after RESET**

This is a normal system message. With PRIO 1 and 2 the connection is interrupted but will be established again, as soon as the communication partner is online. PRIO 3 connections are deleted, but can be initialized again.

E **Order with basic load function**

This is a normal system message. This order is a READ/WRITE-PASSIV and can not be started from the AG.

F **Order not found**

The called order is not parameterized on the CP. This error may occur when the SSNR/A-No. combination in the handling block is wrong or no connection block is entered.

The bits 4 to 7 of Byte 2 are reserved for extensions.

Status management Byte 1, Bit 0 to Bit 3

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 0 **Handshake convenient**

Set:	Per plug-in according to the „delete"-announcement in the order status bit: Handshake convenient (=1) is used at the RECEIVE block (telegram exists at PRIO 1 or RECEIVE impulse is possible at PRIO 2/3)
Analyze:	Per RECEIVE block: The RECEIVE initializes the hand-shake with the CP only if this bit is set.
	Per application: for RECEIVE request (request a telegram at PRIO 1).

Bit 1 **Order is 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. Error causes are to find encrypted in the high-part 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 causes are to find in the high-byte of the indicator word.

Data management Byte 1, Bit 4 to Bit 7

Here you may check if the data transfer is still running or if the data fetch res. transmission is already finished. By means of the bit "Enable/Disable" you may block the data transfer for this order (Disable=1; Enable=0).

Bit 4 **Data fetch / Data transmission is active**

Set: Per handling block SEND or RECEIVE, if the fetch/transmission has been started, e.g. when data is transferred with the ALL-function (DMA-replacement), but the impulse came per SEND-DIRECT.

Delete: Per handling blocks SEND or RECEIVE, if the data transfer of an order is finished (last data block has been transferred).

Analyze: Per user: During the data transfer CP << >>AG the user must not change the record set of an order. This is uncritical with PRIO 0/1 orders, because here the data transfer is realizable in one block cycle. Larger data amounts however are transferred in blocks during more AG cycles. To ensure data consistency you should proof that the data block isn't in transfer any more before you change the content!

BIT 5 **Data transmission is active**

Set: Per handling block SEND, when the data transition for an order is ready.

Delete: Per handling block SEND, when the data transfer for a new order has been started (new trigger).

Per user: When analysis is ready (flank creation).

Analyze: Per user: Here you may ascertain, if the record set of an order has already been transferred to the CP res. at which time a new record set concerning a running order (e.g. cyclic transition) may be started.

Bit 6 **Data fetch active**

Set: Per RECEIVE, when data fetch for a new order has been finished.

Delete: Per RECEIVE, when data transfer to AG for a new order (new trigger) has been started. Per user, when analyzing (edge creation).

Analyze: Per user: Here you may ascertain, if the record set of an order has already been transferred to the CP res. at what time a new record set for the current order has been transferred to the AG.

Bit 7 **Disable/Enable data block**

Set: Per user: to avoid overwriting an area by the RECEIVE block res. data transition of an area by the SEND block (only for the first data block).

Delete: Per user: to release the according data area.

Analyze: Per handling blocks SEND and RECEIVE: if Bit 7 is set, there is no data transfer anymore, but the blocks announce an error to the CP.

Length word Byte 2 and Byte 3

In the length word the handling blocks (SEND, RECEIVE) store the already transferred data of the current order, i.e. the received data amount for receiving orders, the sent data amount for sending orders.

Describe: Per SEND, RECEIVE during the data transfer. The length word is calculated from: **current transfer amount + amount of already transferred data**

Delete: Per overwrite res. with every new SEND, RECEIVE, FETCH.

If the bit "order ready without error" res. "Data fetch/data transition ready " is set, the "Length word" contains the current source res. destination length.

If the bit "order ready with error" is set, the length word contains the data amount transferred before the failure occurred.

Status and error reports

Important status and error reports of the CPU

The following section lists important status and error messages that can appear in the "Indicator word". The representation is in HEX patterns. The literal X means "not declared" res. "irrelevant"; No. is the error number.

Possible indicator words

Indicator word: X F X A

The error index "F" shows, that the according order is not defined on the CP. The state index "A" causes a block of this order (for SEND/FETCH and RECEIVE).

Indicator word: X A X A

The error index "A" shows that the connection of the communication order is not (yet) established. Together with the state index "A" SEND, RECEIVE and FETCH are blocked.

Indicator word: X 0 X 8

The connection has been established again (e.g. after a CP reboot), the SEND order is released (SEND-communication order).

Indicator word: X 0 X 9

The connection has been established again, the RECEIVE order is released (RECEIVE-communication order).

Indicator word: X 0 2 4

SEND has been worked off without errors, the data was transferred.

Indicator word: X 0 4 5

RECEIVE was successful, the data arrived at the AG.

Indicator word: X 0 X 2

The SEND-, RECEIVE-, READ- res. WRITE order is still running. At SEND the partner is not yet ready for RECEIVE or vice versa.

Important indicator word states

The following table shows the most important indicator word states:

Messages at SEND

State under H1	Prio 0/1	Prio 2	Prio 3/4
State under TCP/IP	Prio 1	Prio 2	Prio 3
after reboot	0 A 0 A	0 A 0 A	0 0 0 8
after connection start	X 0 X 8	X 0 X 8	.....
after initial impulse	X 0 X 2	X 0 X 2	X 0 X 2
ready without error	X 0 2 4	X 0 2 4	X 0 2 4
ready with error	X No X 8	X No X 8	X No X 8
after RESET	X D X A	X D X A	X D X 8

Messages at RECEIVE

State under H1	Prio 0/1	Prio 2	Prio 3/4
State under TCP/IP	Prio 1	Prio 2	Prio 3
after reboot	0 A 0 A	0 A 0 A	0 0 0 1
after connection start	X 0 X 4	X 0 0 9	.....
after initial impulse	X 0 X 2	X 0 X 2	X 0 X 2
Telegram received	X 0 X 1	.....	.....
ready without error	X 0 4 1	X 0 4 5	X 0 4 5
ready with error	X No X 8	X No X 9	X No X 9
after RESET	X D X A	X D X A	X D X 9

Messages at READ/WRITE-ACTIVE

State under H1	Prio 0/1	Prio 2	Prio 3/4
State under TCP/IP	Prio 1	Prio 2	Prio 3
after reboot		0 A 0 A	
after connection start		X 0 0 8	
after initial impulse		X 0 X 2	
READ ready		X 0 4 4	
WRITE ready		X 0 2 4	
ready with error		X No X 8	
after RESET		X D X A	

Page frame communication - Parameterization error PAFE

11x	21x	31x	51x
	✓	✓	✓

The parameterization error byte PAFE is set (output or bit memory), when the block detects a parameterization error., e.g. there is no interface or there is an invalid parameterization of QANF/ZANF.

PAFE has the following structure:

Byte	Bit 7 ... Bit 0
0	Bit 0: error 0: no error 1: error, error-No. in Bit 4 to Bit 7 Bit 1 ... Bit 3: reserved Bit 4 ... Bit 7: error-No. 0: no error 1: wrong ORG-Format 2: area not found (DB not found) 3: area too small 4: QVZ- error 5: wrong indicator word 6: no Source-/Destinationparameters at SEND/RECEIVE ALL 7: interface not found 8: interface not specified 9: interface overflow A: reserved B: invalid order-No. C: interface of CP doesn't quit or is negative D: Parameter BLGR not allowed E: reserved F: reserved

SFC 230 - SEND

11x	21x	31x	51x
	✓	✓	✓

Description

The SEND block initializes a send order to a CP.

Normally SEND is called in the cyclic part of the user application program. Although the insertion of this block into the interrupt or the time-alarm program part is possible, the indicator word (ANZW), however, may not be updated cyclically. This should be taken over by a CONTROL block.

The connection initialization with the CP for data transmission and for activating a SEND impulse is only started, if:

- the FB VKE received "1"
- the CP released the order. (Bit "order active" in ANZW =0).

During block stand-by, only the indicator word is updated.

Parameter

Address	Declaration	Name	Type	Init	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Mode of Adressing (direct / indirect)
6.0	in	QANF	ANY		Pointer to data source
16.0	out	PAFE	BYTE		Error indicator for configuration errors
18.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

SEND_ALL for data transmission

If the CP is able to take over the data directly, the SEND block transfers the requested data in one session. If the CP requests only the order parameters or the amount of the depending data is too large, the CP only gets the sending parameters res. the parameter with the first data block. The according data res. the assigned serials blocks for this order are requested from the CP by SEND_ALL to the CPU. For this it is necessary that the block SEND_ALL is called minimum one time per cycle.

The user interface is for all initialization types equal, only the transfer time of the data is postponed for minimum one CPU cycle.

SFC 231 - RECEIVE

11x	21x	31x	51x
	✓	✓	✓

Description

The RECEIVE block receives data from a CP.

Normally the RECEIVE block is called in the cyclic part of the user application program. Although the insertion of this block into the interrupt or the waking program part is possible, the indicator word can not be updated cyclically. This should be taken over by a CONTROL block.

The handshake with the CP (order initialization) and for activating a RECEIVE block is only started, if

- the FB VKE received "1"
- the CP released the order (Bit "Handshake convenient" = 1).

Parameter

Address	Declaration	Name	Type	Initi	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Mode of Adressing (direct / indirect)
6.0	in	ZANF	ANY		Pointer to data destination
16.0	out	PAFE	BYTE		Error indicator for configuration errors
18.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

If the block runs in stand-by only the indicator word is updated.

The RECEIVE block reacts different depending from the kind of supply and the CP reaction:

- If the CP transmits a set of parameters although the RECEIVE block itself got destination parameters, the parameter set of the block has the priority above those of the CP.
- Large amounts of data can only be transmitted in blocks. Therefore you have to transmit the assigned serial blocks by means of RECEIVE_ALL to the CPU. It is necessary that the block RECEIVE_ALL is called minimum one time per application cycle and CP interface, if you want to transmit larger data amounts. You also have to integrate the RECEIVE_ALL cyclically, if the CP only uses the RECEIVE for releasing a receipt telegram and the data is transmitted via the background communication of the CPU.

SFC 232 - FETCH

11x	21x	31x	51x
	✓	✓	✓

Description

The FETCH block initializes a FETCH order in the partner station.

The FETCH order defines data source and destination and the data source is transmitted to the partner station.

The CPU from VIPA realizes the definition of source and destination via a pointer parameter.

The partner station provides the *Source* data and transmits them via SEND_ALL back to the requesting station. Via RECEIVE_ALL the data is received and is stored in *Destination*.

The update of the indicator word takes place via FETCH res. CONTROL.

The handshake for initializing FETCH is only started, if

- the FB VKE receives "1"
- the function has been released in the according CP indicator word (order active = 0).

Parameter

Address	Declaration	Name	Type	Initia	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Mode of Adressing (direct / indirect)
6.0	in	ZANF	ANY		Pointer to data destination
16.0	out	PAFE	BYTE		Error indicator for configuration errors
18.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

Note!

More information for indirect parameterization you will find in this chapter in "Page frame communication - Source res. ".

SFC 233 - CONTROL

11x	21x	31x	51x
	✓	✓	✓

Description

The purpose of the CONTROL block is the following:

- Update of the indicator word
- Query if a certain order of the CP is currently active, e.g. request for a receipt telegram
- Query the CP which order is recently in commission.

The CONTROL block is not responsible for the handshake with the CP, it just transfers the announcements in the order status to the parameterized indicator word. The block is independent from the VKE and should be called from the cyclic part of the application.

Parameter

Address	Declaration	Name	Type	Init	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	out	PAFE	BYTE		Error indicator for configuration errors
6.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

ANR

If ANR ≠0, the indicator word is built up and handled equal to all other handling blocks.

If the parameter ANR gets 0, the CONTROL command transmits the content of the order state cell 0 to the LOW part of the indicator words

The order state cell 0 contains the number of the order that is in commission, e.g. the order number of a telegram (set by the CP).

SFC 234 - RESET

11x	21x	31x	51x
	✓	✓	✓

Description

The RESET_ALL function is called via the order number 0. This resets all orders of this logical interface, e.g. deletes all order data and interrupts all active orders.

With a direct function (ANR≠0) only the specified order will be reset on the logical interface.

The block depends on the VKE and may be called from cyclic, time or alarm controlled program parts.

Parameter

Address	Declaration	Name	Type	Initial	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	out	PAFE	BYTE		Error indicator for configuration errors

Operating modes

The block has two different operating modes:

- RESET ALL
- RESET DIRECT

SFC 235 - SYNCHRON

11x	21x	31x	51x
	✓	✓	✓

Description

The SYNCHRON block initializes the synchronization between CPU and CP during the boot process. For this it has to be called from the starting OBs. Simultaneously the transition area of the interface is deleted and predefined and the CP and the CPU agree about the block size.

Parameter

Address	Declaration	Name	Type	Initial val	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	BLGR	INT		Blocksize
4.0	out	PAFE	BYTE		Error indicator for configuration errors

Block size

To avoid long cycle run-times it is convenient to split large data amounts into smaller blocks for transmitting them between CP and CPU. You declare the size of this blocks by means of „block size“.

A large block size = high data throughput, but also longer run-times and therefore a high cycle time strain.

A small block size = smaller data throughput, but also shorter run-times of the blocks.

Following block sizes are available:

Value	*Block size*	*Value*	*Block size*
0	Default (64Byte)	4	128Byte
1	16Byte	5	256Byte
2	32Byte	6	512Byte
3	64Byte	255	512Byte

Parameter type : Integer

Valid range : 0 ... 255

SFC 236 - SEND_ALL

11x	21x	31x	51x
	✓	✓	✓

Description

Via the SEND_ALL block, the data is transmitted from the CPU to the CP by using the declared block size.

Location and size of the data area that is to transmit with SEND_ALL, must be declared before by calling SEND res. FETCH.

In the indicator word that is assigned to the concerned order, the bit "ENABLE/DISABLE" is set, "Data transmission starts" and "Data transmission running" is calculated or altered.

Parameter

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	out	PAFE	BYTE		Error indicator for configuration errors
4.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

ANZW

In the indicator word of the block, the indicator word, that is parameterized in the SEND_ALL block, the current order number is stored (0 means stand-by).

The amount of the transmitted data for one order is shown in the data word of SEND_ALL which follows the indicator word.

Note!

In the following cases, the SEND_ALL command has to be called for minimum one time per cycle of the block OB1:

- if the CP is able to request data from the CPU independently
- if a CP order is initialized via SEND, but the CP still has to request the background communication data of the CPU for this order.
- if the amount of data, that should be transmitted by this SEND to the CP, is higher than the declared block size.

SFC 237 - RECEIVE_ALL

11x	21x	31x	51x
	✓	✓	✓

Description

Via the RECEIVE_ALL block, the data received from the CP is transmitted from the CP to the CPU by using the declared block size.

Location and size of the data area that is to transmit with RECEIVE_ALL, must be declared before by calling RECEIVE.

In the indicator word that is assigned to the concerned order, the bit "ENABLE/DISABLE" is set, "Data transition starts" and "Data transition/fetch running" is analyzed or altered. The receiving amount is shown in the following word.

Parameter

Address	Declaration	Name	Type	Initial value	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	out	PAFE	BYTE		Error indicator for configuration errors
4.0	in_out	ANZW	DWORD		Indicatorword (progress of started jobs is displayed)

ANZW

In the indicator word of the block, the indicator word, , that is parameterized in the RECEIVE_ALL block, the current order number is stored. In the stand-by running mode of RECEIVE_ALL the block indicator word is deleted.

Note!

In the following cases, the RECEIVE_ALL command has to be called for minimum one time per cycle of the block OB1:

- if the CP should send data to the CPU independently
- if a CP order is initialized via RECEIVE, but the CP still has to request the background communication data of the CPU for this order.
- if the amount of data, that should be transmitted to the CPU by this RECEIVE, is higher than the declared block size.

SFC 238 - CTRL1

11x	21x	31x	51x
	✓	✓	✓

Description

This block is identical to the CONTROL block SFC 233 except that the indicator word is of the type Pointer and that it additionally includes the parameter IND, reserved for further extensions.

The purpose of the CONTROL block is the following:

- Update of the indicator word
- Query if a certain order of the CP is currently active, e.g. request for a receipt telegram
- Query the CP which order is recently in commission.

The CONTROL block is not responsible for the handshake with the CP, it just transfers the announcements in the order status to the parameterized indicator word. The block is independent from the VKE and should be called from the cyclic part of the application.

Parameter

Address	Declaration	Name	Type	Initial	Comment
0.0	in	SSNR	INT		Interface number, number of logical interface
2.0	in	ANR	INT		The job that must be initiated at the interface, i.e. start transmission
4.0	in	IND	INT		Reserved (at the moment)
6.0	out	PAFE	BYTE		Error indicator for configuration errors
8.0	in_out	ANZW	ANY		Indicatorword (progress of started jobs is displayed)

ANR

If ANR ≠0, the indicator word is built up and handled equal to all other handling blocks.

If the parameter ANR gets 0, the CONTROL command transmits the content of the order state cell 0 to the LOW part of the indicator words

The order state cell 0 contains the number of the order that is in commission, e.g. the order number of a telegram (set by the CP).

IND

The parameter IND has no functionality at this time and is reserved for further extensions.

ANZW

The indicator word ANZW is of the type Pointer. This allows you to store the indicator word in a data block.

Chapter 12 Instruction list

Outline

The following chapter lists the available commands of the CPU 11x, 21x, 31x and 51x from VIPA. The instruction list intends to give you an overview over the commands and their syntax. The commands are sorted by topics in alphabetical order.

Via the content the different topics are available.

The alphabetical instruction list gives you direct access to the instructions.

For the parameters are integrated in the instruction list, there is no extra parameter list.

The following chapter describes:

- Instruction and abbreviation list
- Structure of the registers and addressing examples
- Instruction list
- VIPA specific diagnostic entries (Event-IDs)

Content

Alphabetical instruction list

Abbreviations

Abbreviation	Description
/FC	First check bit
2#	Binary constant
a	Byte address
ACCU	Register for processing bytes, words and double words
AR	Address registers, contain the area-internal or area-crossing addresses for the instructions addressed register-indirect
b	Bit address
B	area-crossing, register-indirect addressed Byte
B (b1,b2)	Constant, 2Byte
B (b1,b2,b3,b4)	Constant, 4Byte
B#16#	Byte hexadecimal
BR	Binary result
c	Operand range
C	Counter
C#	Counter constant (BCD-coded)
CC0	Condition code
CC1	Condition code
D	area-crossing, register-indirect addressed double word
D#	IEC date constant
DB	Data block
DBB	Data byte in the data block
DBD	Data double word in the data block
DBW	Data word in the data block
DBX	Data bit in the data block
DI	Instance data block
DIB	Data byte in the instance DB
DID	Data double word in the instance DB
DIW	Data word in the instance DB
DIX	Data bit in the instance DB
DW#16#	Double word hexadecimal
f	Timer/Counter No.
FB	Function block
FC	Functions
g	Operand range
h	Operand range
I	Input (in the PII)
i	Operand range
i8	Integer (8Bit)
i16	Integer (16Bit)
i32	Integer (32 Bit)
IB	Input byte (in the PII)
ID	Input double word (in the PII)
IW	Input word (in the PII)
k8	Constant (8 Bit)
k16	Constant (16 Bit)
k32	Constant (32 Bit)

Abbreviation	Description
L	Local data
L#	Integer constant (32Bit)
LABEL	Symbolic jump address (max. 4 characters)
LB	Local data byte
LD	Local data double word
LW	Local data word
m	Pointer constant P#x.y (pointer)
M	Bit memory bit
MB	Bit memory byte
MD	Bit memory double word
MW	Bit memory word
n	Binary constant
OB	Organization block
OR	Or
OS	Stored overflow
OV	Overflow
p	Hexadecimal constant
P#	Pointer constant
PIQ	Process image of the outputs
PII	Process image of the inputs
PIB	Periphery input byte (direct periphery access)
PID	Periphery input double word (direct periphery access)
PIW	Periphery input word (direct periphery access)
PQB	Periphery output byte (direct periphery access)
PQD	Periphery output double word (direct periphery access)
PQW	Periphery output word (direct periphery access)
Q	Output (in the PIQ)
q	Real number (32bit floating-point number)
QB	Output byte (in the PIQ)
QD	Output double word (in the PIQ)
QW	Output word (in the PIQ)
r	Block no.
RLO	Result of (previous) logic instruction
S5T#	S5 time constant (16Bit), loads the S5-Timer
SFB	System function block
SFC	System function
STA	Status
T	Timer (times)
T#	Time constant (16/32Bit)
TOD#	IEC time constant
W	area-crossing, register-indirect addressed word
W#16#	Word hexadecimal

Registers

ACCU1 and ACCU2 (32Bit)

The ACCUs are registers for the processing of Byte, words or double words. Therefore the operands are loaded in the ACCUs and combined. The result of the instruction is always in ACCU1.

ACCU	Bit
ACCUx (x=1 to 2)	Bit 0 to Bit 31
ACCUx-L	Bit 0 to Bit 15
ACCUx-H	Bit 16 to Bit 31
ACCUx-LL	Bit 0 to Bit 7
ACCUx-LH	Bit 8 to Bit 15
ACCUx-HL	Bit 16 to Bit 23
ACCUx-HH	Bit 24 to Bit 31

Address register AR1 and AR2 (32Bit)

The address registers contain the area-internal or area-crossing addresses for the register-indirect addressed instructions. The address registers are 32Bit wide.

The area-internal or area-crossing addresses have the following structure:

area-internal address:

00000000 00000bbb bbbbbbbb bbbbbxxx

area-crossing address:

10000yyy 00000bbb bbbbbbbb bbbbbxxx

Legend:

b	Byte address
x	Bit number
Y	Range ID (see chapter "Addressing examples")

Status word (16Bit)

The values are analyzed or set by the instructions.
The status word is 16Bit wide.

Bit	Assignment	Description
0	/FC	First check bit*
1	RLO	Result of (previous) logic instruction
2	STA	Status*
3	OR	Or*
4	OS	Stored overflow
5	OV	Overflow
6	CC0	Condition code
7	CC1	Condition code
8	BR	Binary result
9 to 15	not used	-

* Bit may not be analyzed with the instruction L STW in the user application, for the Bit isn't updated during application run-time.

Addressing examples

Addressing example	Description
Immediate addressing	
L +27	Load 16Bit integer constant "27" in ACCU1.
L L#-1	Load 32Bit integer constant "-1" in ACCU1.
L 2#1010101010101010	Load binary constant in ACCU1.
L DW#16#A0F0_BCFD	Load hexadecimal constant in ACCU1.
L 'End'	Load ASCII code in ACCU1.
L T#500ms	Load time value in ACCU1.
L C#100	Load counter value in ACCU1.
L B#(100,12)	Load constant as 2Byte.
L B#(100,12,50,8)	Load constant as 4Byte.
L P#10.0	Load area-internal pointer in ACCU1.
L P#E20.6	Load area-crossing pointer in ACCU1.
L -2.5	Load real number in ACCU1.
L D#1995-01-20	Load date.
L TOD#13:20:33.125	Load time-of-day.
Direct addressing	
A I 0.0	AND operation of input bit 0.0
L IB 1	Load input byte 1 in ACCU1.
L IW 0	Load input word 0 in ACCU1.
L ID 0	Load input double word 0 in ACCU1.

Indirect addressing timer/counter	
SP T [LW 8]	Start timer; timer no. is in local data word 8.
CU C [LW 10]	Start counter; counter no. is in local data word 10.
Memory-indirect, area-internal addressing	
A I [LD 12] e.g.: LP#22.2 T LD 12 A I [LD 12]	AND instruction; input address is in local data double word 12 as pointer.
A I [DBD 1]	AND instruction; input address is in data double word 1 of the DB as pointer.
A Q [DID 12]	AND instruction; output address is in data double word 12 of the instance DB as pointer.
A Q [MD 12]	AND instruction; output address is in bit memory double word 12 as pointer.
Register-indirect, area-internal addressing	
A I [AR1,P#12.2]	AND instruction; input address is calculated "pointer value in address register 1 + pointer P#12.2".
Register-indirect, area-crossing addressing	

For the area-crossing, register indirect addressing the address needs an additional range-ID in the Bits 24-26. The address is in the address register.

Range-ID	**Binary code**	**hex.**	**Area**
P	1000 0**000**	80	Periphery area
I	1000 0**001**	81	Input area
Q	1000 0**010**	82	Output area
M	1000 0**011**	83	Bit memory area
DB	1000 0**100**	84	Data area
DI	1000 0**101**	85	Instance data area
L	1000 0**110**	86	Local data area
VL	1000 0**111**	87	Preceding local data area
			(access to the local data of the calling block)

L B [AR1,P#8.0]	Load byte in ACCU1; the address is calculated "pointer value in address register 1 + pointer P#8.0".
A [AR1,P#32.3]	AND instruction; operand address is calculated "pointer value in address register 1 + pointer P#32.3".
Addressing via parameters	
A parameter	The operand is addressed via the parameter.

Example for pointer calculation

Example when sum of bit addresses ≤ 7:

```
LAR1 P#8.2
A    I [AR1,P#10.2]
```

Result: The input 18.4 is addressed (by adding the byte and bit addresses)

Example when sum of bit addresses > 7:

```
L MD 0          at will calculated pointer, e.g. P#10.5
LAR1
A    I [AR1,P#10.7]
```

Result: Addressed is input 21.4 (by adding the byte and bit addresses with carry)

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Math instructions

Fixed-point arithmetic (16Bit)			*Status word*									Math instructions of two 16Bit numbers. The result is in ACCU1 res. ACCU1-L.	
+I	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Add up two integers (16Bit).	1
			-	-	-	-	-	-	-	-	-	(ACCU1-L)=(ACCU1-L)+(ACCU2-L)	
-I	-		-	Y	Y	Y	Y	-	-	-	-	Subtract two integers (16Bit).	1
												(ACCU1-L)=(ACCU2-L)-(ACCU1-L)	
*I	-											Multiply two integers (16Bit).	1
												(ACCU1-L)=(ACCU2-L)*(ACCU1-L)	
/I	-											Divide two integers (16Bit).	1
												(ACCU1-L)=(ACCU2-L):(ACCU1-L)	
												The remainder is in ACCU1-H.	
Fixed-point arithmetic (32Bit)			*Status word*									Math instructions of two 32Bit numbers. The result is in ACCU1.	
+D	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Add up two integers (32Bit).	1
			-	-	-	-	-	-	-	-	-	(ACCU1)=(ACCU2)+(ACCU1)	
-D	-		-	Y	Y	Y	Y	-	-	-	-	Subtract two integers (32Bit).	1
												(ACCU1)=(ACCU2)-(ACCU1)	
*D	-											Multiply two integers (32Bit).	1
												(ACCU1)=(ACCU2)*(ACCU1)	
/D	-											Divide two integers (32Bit).	1
												(ACCU1)=(ACCU2):(ACCU1)	
MOD	-											Divide two integers (32Bit) and load the rest of the	1
												division in ACCU1.	
												(ACCU1)=remainder of [(ACCU2):(ACCU1)]	
Floating-point arithmetic (32Bit)			*Status word*									The result of the math instructions is in ACCU1. The execution time of the instruction depends on the value to calculate.	
+R	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Add up two real numbers (32Bit)	1
			-	-	-	-	-	-	-	-	-	(ACCU1)=(ACCU2)+(ACCU1)	
-R	-		-	Y	Y	Y	Y	-	-	-	-	Subtract two real numbers (32Bit).	1
												(ACCU1)=(ACCU2)-(ACCU1)	
*R	-											Multiply two real numbers (32Bit).	1
												(ACCU1)=(ACCU2)*(ACCU1)	
/R	-											Divide two real numbers (32Bit).	1
												(ACCU1)=(ACCU2):(ACCU1)	
NEGR	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Negate the real number in ACCU1.	1
			-	-	-	-	-	-	-	-	-		
ABS	-		-	-	-	-	-	-	-	-	-	Form the absolute value of the real number in ACCU1.	1

Command	Operand	Parameter	Status word BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Function	Length in words
												: Instruction depends on : Instruction influences	

			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
Square root an square instructions (32Bit)			*Status word*									The result of the instructions is in ACCU1. The instructions may be interrupted by alarms.	
SQRT	-		-	-	-	-	-	-	-	-	-	Calculate the Square root of a real number in ACCU1.	1
SQR	-		-	Y	Y	Y	Y	-	-	-	-	Form the square of a real number in ACCU1.	1
Logarithmic function (32Bit)			*Status word*									The result of the logarithm function is in ACCU1. The instructions may be interrupted by alarms.	
LN	-		-	-	-	-	-	-	-	-	-	Calculate the natural logarithm of a real number in ACCU1.	1
EXP	-		-	Y	Y	Y	Y	-	-	-	-	Calculate the exponential value of a real number in ACCU1 on basis e (=2.71828).	1
Trigonometrical functions (32Bit)			*Status word*									The result of the trigonometrical function is in ACCU1. The instructions may be interrupted by alarms.	
SIN[1]	-		-	-	-	-	-	-	-	-	-	Calculate the sine of the real number.	1
ASIN[2]	-		-	Y	Y	Y	Y	-	-	-	-	Calculate the arcsine of the real number.	1
COS[1]	-											Calculate the cosine of the real number.	1
ACOS[2]	-											Calculate the arccosine of the real number.	1
TAN[1]	-											Calculate the tangent of the real number.	1
ATAN[2]	-											Calculate the arctangent of the real number.	1
Addition of constants												Addition of integer constants to ACCU1. The condition code bits are not affected.	
+	i8											Add an 8Bit integer constant.	1
+	i16											Add a 16Bit integer constant.	2
+	i32											Add a 32Bit integer constant.	3
Addition via address register												Adding a 16Bit integer to contents of address register. The value is in the instruction or in ACCU1-L. Condition code bits are not affected	
+AR1	-											Add the contents of ACCU1-L to AR1.	1
+AR1	m											Add a pointer constant to the contents of AR1.	2
+AR2	-											Add the contents of ACCU1-L to those of AR2.	1
+AR2	m											Add pointer constant to the contents of AR2.	2

1 Specify the angle in radians; the angle must be given as a floating point value in ACCU 1.

2 The result is an angle in radians.

Command	Operand	Parameter	Status word BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Function	Length in words
												: Instruction depends on	
												: Instruction influences	

Block instructions

Block call instructions			*Status word* BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
CALL	FB r, DB r		-	-	-	-	-	-	-	-	-	Unconditional call of a FB, with parameter transfer	1
CALL	SFB r, DB r		-	-	-	-	0	0	1	-	0	Unconditional call of a SFB, with parameter transfer	2
CALL	FC r											Unconditional call of a function, with parameter transfer	1
CALL	SFC r											Unconditional call of a SFC, with parameter transfer	2
UC	FB r FC r Parameter											Unconditional call of blocks, without parameter transfer FB/FC call via parameters	1
CC	FB r FC r Parameter		BR - -	CC1 - -	CC0 - -	OV - -	OS - 0	OR - 0	STA - 1	RLO Y -	/FC - 0	Conditional call of blocks, without parameter transfer FB/FC call via parameters	1
OPN	DB r DI r Parameter		BR - -	CC1 - -	CC0 - -	OV - -	OS - -	OR - -	STA - -	RLO - -	/FC - -	Open a data block Open a instance data block Open a data block via parameter	1/2 2 2
Block end instructions			*Status word* BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
BE			-	-	-	-	-	-	-	-	-	End block.	1
BEU			-	-	-	-	0	0	1	-	0	End block unconditionally.	1
BEC			BR - -	CC1 - -	CC0 - -	OV - -	OS - Y	OR - 0	STA - 1	RLO Y 1	/FC - 0	End block if RLO="1".	
Exchanging shared data block an instance data block												Exchanging the two current data blocks. The current shared data block becomes the current instance data block and vice versa. The condition code bits are not affected	
CDB												Exchange shared data block and instant data block.	1

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Program display and null instruction instructions

Program display and null operation instructions				The status word is not affected.	
BLD	0 ... 255			Program display instruction: is treated by the CPU like a null operation instruction.	1
NOP	0 1			Null operation instruction	1

Edge-triggered instructions

Edge-triggered instructions			*Status word*									Detection of an edge change. The current signal state of the RLO is compared with the signal state of the instruction or edge bit memory. FP detects a change in the RLO from "0" to "1". FN detects a change in the RLO from "1" to "0".	
FP	I/Q a.b	0.0 to 127.7	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Detecting the positive edge in the RLO. The bit addressed	2
	M a.b	0.0 to 1023.7	-	-	-	-	-	-	-	Y	-	in the instruction is the auxiliary edge bit memory.	2
	L a.b	0.0 to 1043.7	-	-	-	-	-	0	Y	Y	1		2
	DBX a.b	0.0 to 8191.7											2
	DIX a.b	0.0 to 8191.7											2
	c [AR1,m]												2
	c [AR2,m]												2
	[AR1,m]												2
	[AR2,m]												2
	Parameter												2
FN	I/Q a.b	0.0 to 127.7	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Detecting the negative edge in the RLO. The bit addressed	2
	M a.b	0.0 to 1023.7	-	-	-	-	-	-	-	Y	-	in the instruction is the auxiliary edge bit memory.	2
	L a.b	0.0 to 1043.7	-	-	-	-	-	0	Y	Y	1		2
	DBX a.b	0.0 to 8191.7											2
	DIX a.b	0.0 to 8191.7											2
	c [AR1,m]												2
	c [AR2,m]												2
	[AR1,m]												2
	[AR2,m]												2
	Parameter												2

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Load instructions

Load instructions				Loading address identifiers into ACCU1. The contents of ACCU1 and ACCU2 are saved first. The status word is not affected.	
L				Load ...	
	IB a	0 to 127		input byte	1/2
	QB a	0 to 127		output byte	1/2
	PIB a	0 to 1023		periphery input byte	2
	MB a	0 to 1023		bit memory byte	1/2
	LB a	0 to 1043		local data byte	2
	DBB a	0 to 8191		data byte	2
	DIB a	0 to 8191		instance data byte	2
				... in ACCU1	2
	g [AR1,m]			register-indirect, area-internal (AR1)	2
	g [AR2,m]			register-indirect, area-internal (AR2)	2
	B [AR1,m]			area-crossing (AR1)	2
	B [AR2,m]			area-crossing (AR2)	2
	Parameter			via parameters	2
L				Load ...	
	IW a	0 to 126		input word	1/2
	QW a	0 to 126		output word	1/2
	PIW a	0 to 1022		periphery input word	
	MW a	0 to 1022		bit memory word	1/2
	LW a	0 to 1042		local data word	2
	DBW a	0 to 8190		data word	1/2
	DIW a	0 to 8190		instance data word	1/2
				... in ACCU1-L	
	h [AR1,m]			register-indirect, area-internal (AR1)	2
	h [AR2,m]			register-indirect, area-internal (AR2)	2
	W [AR1,m]			area-crossing (AR1)	2
	W [AR2,m]			area-crossing (AR2)	2
	Parameter			via parameters	2

Command	Operand	Parameter	Status word BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Function	Length in words
												: Instruction depends on : Instruction influences	
L												Load ...	
	ID a	0 to 124										input double word.	1/2
	QD a	0 to 124										output double word.	1/2
	PID a	0 to 1020										periphery input double word.	2
	MD a	0 to 1020										bit memory double word.	1/2
	LD a	0 to 1040										local data double word.	2
	DBD a	0 to 8188										data double word.	2
	DID a	0 to 8188										instance data double word.	2
												... in ACCU1-L.	
	i [AR1,m]											register-indirect, area-internal (AR1)	2
	i [AR2,m]											register-indirect, area-internal (AR2)	2
	D [AR1,m]											area-crossing (AR1)	2
	D [AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2
L												Load ...	
	k8											8Bit constant in ACCU1-LL.	1
	k16											16Bit constant in ACCU1-L.	2
	k32											32Bit constant in ACCU1.	3
	Parameter											Load constant in ACCU1 (addressed via parameters)	2
L	2#n											Load 16Bit binary constant in ACCU1-L.	2
												Load 32Bit binary constant in ACCU1.	3
L	B#8#p											Load 8Bit hexadecimal constant in ACCU1-LL.	1
	W#16#p											Load 16Bit hexadecimal constant in ACCU1-L.	2
	DW#16#p											Load 32Bit hexadecimal constant in ACCU1.	3
L	x											Load one character.	
L	xx											Load two characters.	2
L	xxx											Load three characters.	
L	xxxx											Load four characters.	3
L	D# Date											Load IEC-date (BCD-coded).	3
L	S5T# time value											Load time constant (16Bit).	2
L	TOD# time value											Load 32Bit time constant. (IEC-time-of-day)	3
L	T# time											Load 16Bit time constant.	2
	value											Load 32Bit time constant.	3
L	C# counter value											Load 16Bit counter constant.	2
L	P# bit pointer											Load bit pointer.	3
L	L# Integer											Load 32Bit integer constant.	3
L	Real											Load real number.	3

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Load instructions for timer and counter				Load a time or counter value in ACCU1, before the recent content of ACCU1 is saved in ACCU2. The status word is not affected.	
L	T f	0 to 255		Load time value.	1/2
	Timer parameter			Load time value (addressed via parameters).	2
L	C f	0 to 255		Load counter value.	1/2
	Counter parameter			Load counter value (addressed via parameters).	2
LD	T f	0 to 255		Load time value BCD-coded.	1/2
	Timer parameter			Load time value BCD-coded (addressed via parameters).	2
LD	C f	0 to 255		Load counter value BCD-coded.	1/2
	Counter parameter			Load counter value BCD-coded (addressed via parameters).	2

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Shift instructions

Shift instructions			*Status word*									Shifting the contents of ACCU1 and ACCU1-L to the left or right by the specified number of places. If no address identifier is specified, shift the number of places into ACCU2-LL. Any positions that become free are padded with zeros or the sign. The last shifted bit is in condition code bit CC1.	
SLW	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Shift the contents of ACCU1-L to the left.	1
SLW	0 ... 15		-	-	-	-	-	-	-	-	-	Positions that become free are provided with zeros	
SLD	-		-	Y	Y	Y	-	-	-	-	-	Shift the contents of ACCU1 to the left.	1
SLD	0 ... 32											Positions that become free are provided with zeros.	
SRW	-											Shift the contents of ACCU1-L to the right.	1
SRW	0 ... 15											Positions that become free are provided with zeros.	
SRD	-											Shift the contents of ACCU1 to the right.	1
SRD	0 ... 32											Positions that become free are provided with zeros.	
SSI	-											Shift the contents of ACCU1-L to the right with sign.	1
SSI	0 ... 15											Positions that become free are provided with the sign (Bit 15) .	
SSD	-											Shift the contents of ACCU1 to the right with sign	1
SSD	0 ... 32												
Rotation instructions			*Status word*									Rotate the contents of ACCU1 to the left or right by the specified number of places. If no address identifier is specified, rotate the number of places into ACCU2-LL.	
RLD	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Rotate the contents of ACCU1 to the left.	1
RLD	0 ... 32		-	-	-	-	-	-	-	-	-		
RRD	-		-	Y	Y	Y	-	-	-	-	-	Rotate the contents of ACCU1 to the right.	1
RRD	0 ... 32												
RLDA	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Rotate the contents of ACCU1 one bit position to the left,	
			-	-	-	-	-	-	-	-	-	via CC1 bit.	
RRDA	-		-	Y	0	0	-	-	-	-	-	Rotate the contents of ACCU1 one bit position to the right,	
												via CC1 bit.	

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Setting/resetting bit addresses

Set/Reset bit addresses			*Status word*									Assign the value "1" or "0" or the RLO to the addressed instructions.	
S			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Set ...	
	I/Q a.b	0.0 to 127.7	-	-	-	-	-	-	-	Y	-	input/output to "1".	1/2
	M a.b	0.0 to 1023.7	-	-	-	-	-	0	Y	-	0	set bit memory to "1".	1/2
	L a.b	0.0 to 1043.7										local data bit to "1".	2
	DBX a.b	0.0 to 8191.7										data bit to "1".	2
	DIX a.b	0.0 to 8191.7										instance data bit to "1".	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2
R			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Reset ...	
	I/Q a.b	0.0 to 127.7	-	-	-	-	-	-	-	Y	-	input/output to "0".	1/2
	M a.b	0.0 to 1023.7	-	-	-	-	-	0	Y	-	0	set bit memory to "0".	1/2
	L a.b	0.0 to 1043.7										local data bit to "0".	2
	DBX a.b	0.0 to 8191.7										data bit to "0".	2
	DIX a.b	0.0 to 8191.7										instance data bit to "0".	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	W [AR1,m]											area-crossing (AR1)	2
	W [AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2
=			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Assign ...	
	I/Q a.b	0.0 to 127.7	-	-	-	-	-	-	-	Y	-	RLO to input/output	1/2
	M a.b	0.0 to 1023.7	-	-	-	-	-	0	Y	-	0	RLO to bit memory	1/2
	L a.b	0.0 to 1043.7										RLO to local data bit	2
	DBX a.b	0.0 to 8191.7										RLO to data bit	2
	DIX a.b	0.0 to 8191.7										RLO to instance data bit	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Instructions directly affecting the RLO			*Status word*									The following instructions have a directly effect on the RLO.	
CLR			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Set RLO to "0".	1
			-	-	-	-	-	-	-	-	-		
			-	-	-	-	-	0	0	0	0		
SET			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Set RLO to "1".	1
			-	-	-	-	-	-	-	-	-		
			-	-	-	-	-	0	1	1	0		
NOT			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Negate RLO.	1
			-	-	-	-	-	Y	-	Y	-		
			-	-	-	-	-	-	1	Y	-		
SAVE			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Save RLO into BR-Bit.	1
			-	-	-	-	-	-	-	Y	-		
			Y	-	-	-	-	-	-	-	-		

Jump instructions

Jump instructions			*Status word*									Jump, depending on conditions. 8-Bit operands have a jump width of (-128...+127), 16-Bit operands of (-32768...-129) or (+128...+32767)	
JU	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump unconditionally.	1/2
			-	-	-	-	-	-	-	-	-		
			-	-	-	-	-	-	-	-	-		
JC	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump if RLO="1".	1/2
JCN	LABEL		-	-	-	-	-	-	-	Y	-	Jump if RLO="0".	2
			-	-	-	-	-	0	1	1	0		
JCB	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump if RLO="1".	2
			-	-	-	-	-	-	-	Y	-	Save the RLO in the BR-Bit	
JNB	LABEL		Y	-	-	-	-	0	1	1	0	Jump if RLO="0".	2
												Save the RLO in the BR-Bit	
JBI	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump if BR="1".	2
JNBI	LABEL		Y	-	-	-	-	-	-	-	-	Jump if BR="0".	2
			-	-	-	-	-	0	1	-	0		
JO	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump on stored overflow (OV="1").	1/2
			-	-	-	Y	-	-	-	-	-		
			-	-	-	-	-	-	-	-	-		
JOS	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump on stored overflow (OS="1").	2
			-	-	-	-	Y	-	-	-	-		
			-	-	-	-	0	-	-	-	-		

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

JUO	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump if "unordered instruction" (CC1=1 and CC0=1).	2
JZ	LABEL		-	Y	Y	-	-	-	-	-	-	Jump if result=0 (CC1=0 and CC0=0).	1/2
JP	LABEL		-	-	-	-	-	-	-	-	-	Jump if result>0 (CC1=1 and CC0=0).	1/2
JM	LABEL											Jump if result<0 (CC1=0 and CC0=1).	1/2
JN	LABEL											Jump if result≠0. (CC1=1 and CC0=0) or (CC1=0) and (CC0=1)	1/2
JMZ	LABEL											Jump if result≤0. (CC1=0 and CC0=1) or (CC1=0 and CC0=0)	2
JPZ	LABEL											Jump if result≥0. (CC1=1 and CC0=0) or (CC1=0 and CC0=0)	2
JL	LABEL		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Jump distributor	2
			-	-	-	-	-	-	-	-	-	This instruction is followed by a list of jump instructions.	
			-	-	-	-	-	-	-	-	-	The operand is a jump label to subsequent instructions in this list. ACCU1-L contains the number of the jump instruction to be executed.	
LOOP	LABEL											Decrement ACCU1-L and jump if ACCU1-L _ 0 (loop programming)	2

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Transfer instructions

Transfer instructions				Transfer the contents of ACCU1 into the addressed operand. The status word is not affected.	
T				Transfer the contents of ACCU1-LL to...	
	IB a	0 to 127		input byte.	1/2
	QB a	0 to 127		output byte.	1/2
	PQB a	0 to 1023		periphery output byte.	1/2
	MB a	0 to 1023		bit memory byte.	1/2
	LB a	0 to 1043		local data byte.	2
	DBB a	0 to 8191		data byte.	2
	DIB a	0 to 8191		instance data byte.	2
	g [AR1,m]			register-indirect, area-internal (AR1)	2
	g [AR2,m]			register-indirect, area-internal (AR2)	2
	B [AR1,m]			area-crossing (AR1)	2
	B [AR2,m]			area-crossing (AR2)	2
	Parameter			via parameters	2
T				Transfer the contents of ACCU1-L to ...	
	IW	0 to 126		input word.	1/2
	QW	0 to 126		output word.	1/2
	PQW	0 to 1022		periphery output word.	1/2
	MW	0 to 1022		bit memory word.	1/2
	LW	0 to 1042		local data word.	2
	DBW	0 to 8190		data word.	2
	DIW	0 to 8190		instance data word.	2
	h [AR1,m]			register-indirect, area-internal (AR1)	2
	h [AR2,m]			register-indirect, area-internal (AR2)	2
	W [AR1,m]			area-crossing (AR1)	2
	W [AR2,m]			area-crossing (AR2)	2
	Parameter			via parameters	2

Command	Operand	Parameter	Status word BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Function	Length in words
												: Instruction depends on	
												: Instruction influences	

T				Transfer the contents of ACCU1 to...	
	ID	0 to 124		input double word.	1/2
	QD	0 to 124		output double word.	1/2
	PQD	0 to 1020		periphery output double word.	1/2
	MD	0 to 1020		bit memory double word.	1/2
	LD	0 to 1040		local data double word.	2
	DBD	0 to 8188		data double word.	2
	DID	0 to 8188		instance data double word.	2
	i [AR1,m]			register-indirect, area-internal (AR1)	2
	i [AR2,m]			register-indirect, area-internal (AR2)	2
	D [AR1,m]			area-crossing (AR1)	2
	D [AR2,m]			area-crossing (AR2)	2
	Parameter			via parameters	2
Load and transfer instructions for address register				Load a double word from a memory area or a register into AR1 or AR2.	
LAR1				Load the contents from...	
	-			ACCU1.	1
	AR2			address register 2.	1
	DBD a	0 to 8188		data double word.	2
	DID a	0 to 8188		instance data double word.	2
	m			32Bit constant as pointer.	3
	LD a	0 to 1040		local data double word.	2
	MD a	0 to 1020		bit memory double word.	2
				... into AR1.	
LAR2				Load the contents from ...	
	-			ACCU1.	1
	DBD a	0 to 8188		data double word.	2
	DID a	0 to 8188		instance data double word.	2
	m			32Bit constant as pointer.	3
	LD a	0 to 1040		local data double word.	2
	MD a	0 to 1020		bit memory double word.	2
				... into AR2.	
TAR1				Transfer the contents from AR1 to...	
	-			ACCU1.	1
	AR2			address register 2.	1
	DBD a	0 to 8188		data double word.	2
	DID a	0 to 8188		instance data double word.	2
	LD a	0 to 1040		local data double word.	2
	MD a	0 to 1020		bit memory double word.	2

Command	Operand	Parameter	Status word BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Function	Length in words
												: Instruction depends on	
												: Instruction influences	

TAR2												Transfer the contents from AR2 to...	
	-											ACCU1.	1
	DBD a	0 to 8188										data double word.	2
	DID a	0 to 8188										instance data double word.	2
	LD a	0 to 1040										local data double word.	2
	MD a	0 to 1020										bit memory double word.	2
TAR												Exchange the contents of AR1 and AR2	1
Load and transfer instructions for the status word			*Status word*										
L	STW		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Load status word in ACCU1.	
	-		Y	Y	Y	Y	Y	0	0	Y	0		
			-	-	-	-	-	-	-	-	-		
T	STW		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Transfer ACCU1 (Bits 0 to 8) into status word.	
	-		-	-	-	-	-	-	-	-	-		
			Y	Y	Y	Y	Y	-	-	Y	-		
Load instructions for DB number and DB length												Load the number/length of a data block to ACCU1. The old contents of ACCU1 are saved into ACCU2. The condition code bits are not affected	
L	DBNO											Load number of data block.	1
L	DINO											Load number of instance data block.	1
L	DBLG											Load length of data block into byte.	1
L	DILG											Load length of instance data block into byte.	1
ACCU transfer instructions, increment, decrement												The status word is not affected.	
CAW	-											Reverse the order of the bytes in ACCU1-L.	1
												LL, LH becomes LH, LL.	
CAD	-											Reverse the order of the bytes in ACCU1.	1
												LL, LH, HL, HH becomes HH, HL, LH, LL.	
TAK	-											Swap the contents of ACCU1 and ACCU2	1
PUSH	-											The contents of ACCU1 are transferred to ACCU2.	1
POP	-											The contents of ACCU2 are transferred to ACCU1.	1
INC	0 ... 255											Increment ACCU1-LL.	1
DEC	0 ... 255											Decrement ACCU1-LL.	1

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Data type conversion instructions

Data type conversion instructions			*Status word*									The results of the conversion are in ACCU1. When converting real numbers, the execution time depends on the value.	
BTI	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Convert contents of ACCU1 from BCD to integer (16Bit)	1
			-	-	-	-	-	-	-	-	-	(**B**CD **T**o **I**nt.).	
BTD	-		-	-	-	-	-	-	-	-	-	Convert contents of ACCU1 from BCD to integer (32Bit).	1
												(**B**CD **T**o **D**oubleint.).	
DTR	-											Convert cont. of ACCU1 from integer (32Bit) to Real number	1
												(32Bit) (**D**oubleint. **T**o **R**eal).	
ITD	-											Convert contents of ACCU1 from integer (16Bit) to integer	1
												(32Bit) (**I**nt. **T**o **D**oubleint.).	
ITB	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Convert contents of ACCU1 from integer (16Bit) to BCD	1
			-	-	-	-	-	-	-	-	-	0 to +/-999 (**I**nt. **T**o **B**CD).	
DTB	-		-	-	-	Y	Y	-	-	-	-	Convert contents of ACCU1 from integer (32Bit) to BCD	1
												0 to +/-9 999 999 (**D**oubleint. **T**o **B**CD).	
RND	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Convert a real number to 32Bit integer.	1
RND-	-		-	-	-	-	-	-	-	-	-	Convert a real number to 32Bit integer.	1
			-	-	-	Y	Y	-	-	-	-	The number is rounded next hole number..	
RND+	-											Convert real number to 32Bit integer.	1
												It is rounded up to the next integer.	
TRUNC	-											Convert real number to 32Bit integer.	1
												The places after the decimal point are truncated.	
Complement creation			*Status word*										
INVI	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Forms the ones complement of ACCU1-L (integer).	1
INVD	-		-	-	-	-	-	-	-	-	-	Forms the ones complement of ACCU1.	1
			-	-	-	-	-	-	-	-	-		
NEGI	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Forms the twos complement of ACCU1-L.	1
NEGD	-		-	-	-	-	-	-	-	-	-	Forms the twos complement of ACCU1 (double integer).	1
			-	Y	Y	Y	Y	-	-	-	-		

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Comparison instructions

Comparison instructions with integer (16Bit)			*Status word*									Comparing the integer (16Bit) in ACCU1-L and ACCU2-L. RLO=1, if condition is satisfied.	
==I	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	ACCU2-L=ACCU1-L	1
<>I	-		-	-	-	-	-	-	-	-	-	ACCU2-L≠ACCU1-L	1
<I	-		-	Y	Y	0	-	0	Y	Y	1	ACCU2-L<ACCU1-L	1
<=I	-											ACCU2-L<=ACCU1-L	1
>I	-											ACCU2-L>ACCU1-L	1
>=I	-											ACCU2-L>=ACCU1-L	1
Comparison instructions with integer (32Bit)			*Status word*									Comparing the integer (32Bit) in ACCU1 and ACCU2. RLO=1, if condition is satisfied.	
==D	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	ACCU2=ACCU1	1
<>D	-		-	-	-	-	-	-	-	-	-	ACCU2≠ACCU1	1
<D	-		-	Y	Y	0	-	0	Y	Y	1	ACCU2<ACCU1	1
<=D	-											ACCU2<=ACCU1	1
>D	-											ACCU2>ACCU1	1
>=D	-											ACCU2>=ACCU1	1
Comparison instructions with 32Bit real number			*Status word*									Comparing the 32Bit real numbers in ACCU1 and ACCU2. RLO=1, is condition is satisfied. The execution time of the instruction depends on the value to be compared.	
==R	-		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	ACCU2=ACCU1	1
<>R	-		-	-	-	-	-	-	-	-	-	ACCU2≠ACCU1	1
<R	-		-	Y	Y	Y	Y	0	Y	Y	1	ACCU2<ACCU1	1
<=R	-											ACCU2<=ACCU1	1
>R	-											ACCU2>ACCU1	1
>=R	-											ACCU2>=ACCU1	1

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Combination instructions (Bit)

Combination instructions with bit operands			*Status word*									Examining the signal state of the addressed instruction and gating the result with the RLO according to the appropriate logic function.	
A			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND operation at signal state "1"	
	I/Q a.b	0.0 ... 127.7	-	-	-	-	-	Y	-	Y	Y	Input/output	1/2
	M a.b	0.0 ... 1023.7	-	-	-	-	-	Y	Y	Y	1	Bit memory	1/2
	L a.b	0.0 ... 1043.7										Local data bit	2
	DBX a.b	0.0 ... 8191.7										Data bit	2
	DIX a.b	0.0 ... 8191.7										Instance data bit	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2
AN			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND operation of signal state "0"	
	I/Q a.b	0.0 ... 127.7	-	-	-	-	-	Y	-	Y	Y	Input/output	1/2
	M a.b	0.0 ... 1023.7	-	-	-	-	-	Y	Y	Y	1	Bit memory	1/2
	L a.b	0.0 ... 1043.7										Local data bit	2
	DBX a.b	0.0 ... 8191.7										Data bit	2
	DIX a.b	0.0 ... 8191.7										Instance data bit	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2
O			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	OR operation at signal state "1"	
	I/Q a.b	0.0 ... 127.7	-	-	-	-	-	-	-	Y	Y	Input/output	1/2
	M a.b	0.0 ... 1023.7	-	-	-	-	-	0	Y	Y	1	Bit memory	1/2
	L a.b	0.0 ... 1043.7										Local data bit	2
	DBX a.b	0.0 ... 8191.7										Data bit	2
	DIX a.b	0.0 ... 8191.7										Instance data bit	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2

Command	Operand	Parameter	Status word BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Function	Length in words
												: Instruction depends on	
												: Instruction influences	

			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
ON												OR operation at signal state "0"	
	I/Q a.b	0.0 ... 127.7	-	-	-	-	-	-	-	Y	Y	Input/output	1/2
	M a.b	0.0 ... 1023.7	-	-	-	-	-	0	Y	Y	1	Bit memory	1/2
	L a.b	0.0 ... 1043.7										Local data bit	2
	DBX a.b	0.0 ... 8191.7										Data bit	2
	DIX a.b	0.0 ... 8191.7										Instance data bit	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2
X			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	EXCLUSIVE-OR operation at signal state "1"	
	I/Q a.b	0.0 ... 127.7	-	-	-	-	-	-	-	Y	Y	Input/output	2
	M a.b	0.0 ... 1023.7	-	-	-	-	-	0	Y	Y	1	Bit memory	2
	L a.b	0.0 ... 1043.7										Local data bit	2
	DBX a.b	0.0 ... 8191.7										data bit	2
	DIX a.b	0.0 ... 8191.7										Instance data bit	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2
XN			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	EXCLUSIVE-OR operation at signal state "0"	
	I/Q a.b	0.0 ... 127.7	-	-	-	-	-	-	-	Y	Y	Input/output	2
	M a.b	0.0 ... 1023.7	-	-	-	-	-	0	Y	Y	1	Bit memory	2
	L a.b	0.0 ... 1043.7										Local data bit	2
	DBX a.b	0.0 ... 8191.7										Data bit	2
	DIX a.b	0.0 ... 8191.7										Instance data bit	2
	c [AR1,m]											register-indirect, area-internal (AR1)	2
	c [AR2,m]											register-indirect, area-internal (AR2)	2
	[AR1,m]											area-crossing (AR1)	2
	[AR2,m]											area-crossing (AR2)	2
	Parameter											via parameters	2

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Combination instructions with parenthetical expressions			*Status word*									Saving the bits BR, RLO, OR and a function ID (A, AN, ...) at the nesting stack. For each block 7 nesting levels are possible.	
A(			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND left parenthesis	1
AN(			Y	-	-	-	-	Y	-	Y	Y	AND-NOT left parenthesis	1
O(			-	-	-	-	-	0	1	-	0	OR left parenthesis	1
ON(												OR-NOT left parenthesis	1
X(												EXCLUSIVE-OR left parenthesis	1
XN(												EXCLUSIVE-OR-NOT left parenthesis	1
)			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Right parenthesis, popping an entry off the nesting stack,	1
			-	-	-	-	-	-	-	Y	-	gating RLO with the current RLO in the processor.	
			Y	-	-	-	-	Y	1	Y	1		
ORing of AND operations			*Status word*									The ORing of AND operations is implemented according the rule: AND before OR.	
O			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	OR operations of AND functions according the rule:	1
			-	-	-	-	-	Y	-	Y	Y	AND before OR	
			-	-	-	-	-	Y	1	-	Y		

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Combinations instructions with timer and counters			*Status word*									Examining the signal state of the addressed timer/counter an gating the result with the RLO according to the appropriate logic function.	
A			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND operation at signal state	
	T f	0 to 255	-	-	-	-	-	Y	-	Y	Y	Timer	1/2
	C f	0 to 255	-	-	-	-	-	Y	Y	Y	1	Counter	1/2
	Timer para. Counter p.											Timer addressed via parameters Counter addressed via parameters	2
AN			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND operation at signal state	
	T f	0 to 255	-	-	-	-	-	Y	-	Y	Y	Timer	1/2
	C f	0 to 255	-	-	-	-	-	Y	Y	Y	1	Counter	1/2
	Timer para. Counter p.											Timer addressed via parameters Counter addressed via parameters	2
O			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	OR operation at signal state	
	T f	0 to 255	-	-	-	-	-	-	-	Y	Y	Timer	1/2
	C f	0 to 255	-	-	-	-	-	0	Y	Y	1	Counter	1/2
	Timer para. Counter p.											Timer addressed via parameters Counter addressed via parameters	2
ON			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	OR operation at signal state	
	T f	0 to 255	-	-	-	-	-	-	-	Y	Y	Timer	1/2
	C f	0 to 255	-	-	-	-	-	0	Y	Y	1	Counter	1/2
	Timer para. Counter p.											Timer addressed via parameters Counter addressed via parameters	2
X			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	EXCLUSIVE-OR operation at signal state	
	T f	0 to 255	-	-	-	-	-	-	-	Y	Y	Timer	2
	C f	0 to 255	-	-	-	-	-	0	Y	Y	1	Counter	2
	Timer para. Counter p.											Timer addressed via parameters Counter addressed via parameters	2
XN			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	EXCLUSIVE-OR operation at signal state	
	T f	0 to 255	-	-	-	-	-	-	-	Y	Y	Timer	2
	C f	0 to 255	-	-	-	-	-	0	Y	Y	1	Counter	2
	Timer para. Counter p.											Timer addressed via parameters Counter addressed via parameters	2

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Combination instructions using AND, OR and EXCLUSIVE OR			*Status word*									Examining the specified conditions for their signal status, and gating the result with the RLO according to the appropriate function.	
A			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND operation at signal state "1"	
	==0		Y	Y	Y	Y	Y	Y	-	Y	Y	Result=0 (CC1=0) and (CC0=0)	1
	>0		-	-	-	-	-	Y	Y	Y	1	Result>0 (CC1=1) and (CC0=0)	1
	<0											Result<0 (CC1=0) and (CC0=1)	1
	<>0											Result≠0 ((CC1=0) and (CC0=1)) or ((CC1=1) and (CC0=0))	1
	<=0											Result<=0 ((CC1=0) and (CC0=1)) or ((CC1=0) and (CC0=0))	1
	>=0											Result>=0 ((CC1=1) and (CC0=0)) or ((CC1=1) and (CC0=0))	1
	UO											unordered math instruction (CC1=1) and (CC0=1)	1
	OS											OS=1	1
	BR											BR=1	1
	OV											OV=1	1
AN			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND operation at signal state "0"	
	==0		Y	Y	Y	Y	Y	Y	-	Y	Y	Result=0 (CC1=0) and (CC0=0)	1
	>0		-	-	-	-	-	Y	Y	Y	1	Result>0 (CC1=1) and (CC0=0)	1
	<0											Result<0 (CC1=0) and (CC0=1)	1
	<>0											Result≠0 ((CC1=0) and (CC0=1)) or ((CC1=1) and (CC0=0))	1
	<=0											Result<=0 ((CC1=0) and (CC0=1)) or ((CC1=0) and (CC0=0))	1
	>=0											Result>=0 ((CC1=1) and (CC0=0)) or ((CC1=1) and (CC0=0))	1
	UO											unordered math instruction (CC1=1) and (CC0=1)	1
	OS											OS=0	1
	BR											BR=0	1
	OV											OV=0	1
O			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	OR operation at signal state "1"	
	==0		Y	Y	Y	Y	Y	-	-	Y	Y	Result=0 (CC1=0) and (CC0=0)	1
	>0		-	-	-	-	-	0	Y	Y	1	Result>0 (CC1=1) and (CC0=0)	1
	<0											Result<0 (CC1=0) and (CC0=1)	1
	<>0											Result≠0 ((CC1=0) and (CC0=1)) or ((CC1=1) and (CC0=0))	1
	<=0											Result<=0 ((CC1=0) and (CC0=1)) or ((CC1=0) and (CC0=0))	1
	>=0											Result>=0 ((CC1=1) and (CC0=0)) or ((CC1=1) and (CC0=0))	1
	UO											unordered math instruction (CC1=1) and (CC0=1)	1
	OS											OS=1	1
	BR											BR=1	1
	OV											OV=1	1

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

ON			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	OR operation at signal state "0"	
	==0		Y	Y	Y	Y	Y	-	-	Y	Y	Result=0 (CC1=0) and (CC0=0)	1
	>0		-	-	-	-	-	0	Y	Y	1	Result>0 (CC1=1) and (CC0=0)	1
	<0											Result<0 (CC1=0) and (CC0=1)	1
	<>0											Result≠0 ((CC1=0) and (CC0=1)) or ((CC1=1) and (CC0=0))	1
	<=0											Result<=0 ((CC1=0) and (CC0=1)) or ((CC1=0) and (CC0=0))	1
	>=0											Result>=0 ((CC1=1) and (CC0=0)) or ((CC1=1) and (CC0=0))	1
	UO											unordered math instruction (CC1=1) and (CC0=1)	1
	OS											OS=0	1
	BR											BR=0	1
	OV											OV=0	1
X			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	EXCLUSIVE-OR operation at signal state "1"	
	==0		Y	Y	Y	Y	Y	-	-	Y	Y	Result=0 (CC1=0) and (CC0=0)	1
	>0		-	-	-	-	-	0	Y	Y	1	Result>0 (CC1=1) and (CC0=0)	1
	<0											Result<0 (CC1=0) and (CC0=1)	1
	<>0											Result≠0 ((CC1=0) and (CC0=1)) or ((CC1=1) and (CC0=0))	1
	<=0											Result<=0 ((CC1=0) and (CC0=1)) or ((CC1=0) and (CC0=0))	1
	>=0											Result>=0 ((CC1=1) and (CC0=0)) or ((CC1=1) and (CC0=0))	1
	UO											unordered math instruction (CC1=1) and (CC0=1)	1
	OS											OS=1	1
	BR											BR=1	1
	OV											OV=1	1
XN			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	EXCLUSIVE-OR operation at signal state "0"	
	==0		Y	Y	Y	Y	Y	-	-	Y	Y	Result=0 (CC1=0) and (CC0=0)	1
	>0		-	-	-	-	-	0	Y	Y	1	Result>0 (CC1=1) and (CC0=0)	1
	<0											Result<0 (CC1=0) and (CC0=1)	1
	<>0											Result≠0 ((CC1=0) and (CC0=1)) or ((CC1=1) and (CC0=0))	1
	<=0											Result<=0 ((CC1=0) and (CC0=1)) or ((CC1=0) and (CC0=0))	1
	>=0											Result>=0 ((CC1=1) and (CC0=0)) or ((CC1=1) and (CC0=0))	1
	UO											unordered math instruction (CC1=1) and (CC0=1)	1
	OS											OS=0	1
	BR											BR=0	1
	OV											OV=0	1

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Combination instructions (Word)

Combination instructions with the contents of ACCU1			*Status word*									Gating the contents of ACCU1 and/or ACCU1-L with a word or double word according to the appropriate function. The word or double word is either a constant in the instruction or in ACCU2. The result is in ACCU1 and/or ACCU1-L.	
AW			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	AND ACCU2-L	1
AW	k16		-	-	-	-	-	-	-	-	-	AND 16Bit constant	2
OW			-	Y	0	0	-	-	-	-	-	OR ACCU2-L	1
OW	k16											OR 16Bit constant	2
XOW												EXCLUSIVE OR ACCU2-L	1
XOW	k16											EXCLUSIVE OR 16Bit constant	2
AD												AND ACCU2	1
AD	k32											AND 32Bit constant	3
OD												OR ACCU2	1
OD	k32											OR 32Bit constant	3
XOD												EXCLUSIVE OR ACCU2	1
XOD	k32											EXCLUSIVE OR 32Bit constant	3

Timer instructions

Time instructions			*Status word*									Starting or resetting a timer (addressed directly or via parameters). The time value must be in ACCU1-L.	
SP	T f	0 to 255	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Start time as pulse on edge change from "0" to "1".	1/2
	Timer para.		-	-	-	-	-	-	-	Y	-		2
SE	T f	0 to 255	-	-	-	-	-	0	-	-	0	Start timer as extended pulse on edge change from	1/2
	Timer para.											"0" to "1".	2
SD	T f	0 to 255										Start timer as ON delay on edge change	1/2
	Timer para.											from "0" to "1".	2
SS	T f	0 to 255										Start timer as saving start delay on edge change	1/2
	Timer para.											from "0" to "1".	2
SA	T f	0 to 255										Start timer as OFF delay on edge change from	1/2
	Timer para.											"1" to "0".	2
FR	T f	0 to 255										Enable timer for restarting on edge change from "0" to "1"	1/2
	Timer para.											(reset edge bit memory for starting timer).	2
R	T f	0 to 255										Reset timer.	1/2
	Timer para.												2

Command	Operand	Parameter	Status word									Function	Length in words
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC		
												: Instruction depends on	
												: Instruction influences	

Counter instructions

Counter instructions			*Status word*									The counter value is in ACCU1-L res. in the address transferred as parameter.	
S	C f	0 to 255	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	Presetting of counter on edge change from "0" to "1".	1/2
	Counter p.		-	-	-	-	-	-	-	Y	-		2
R	C f	0 to 255	-	-	-	-	-	0	-	-	0	Reset counter to "0".	1/2
	Counter p.												2
CU	C f	0 to 255										Increment counter by 1 on edge change from "0" to "1".	1/2
	Counter p.												2
CD	C f	0 to 255										Decrement counter by 1 on edge change from "0" to "1".	1/2
	Counter p.												2
FR	C f	0 to 255										Enable counter on edge change from "0" to "1".	1/2
	Counter p.											(reset the edge bit memory for up and down counting)	2

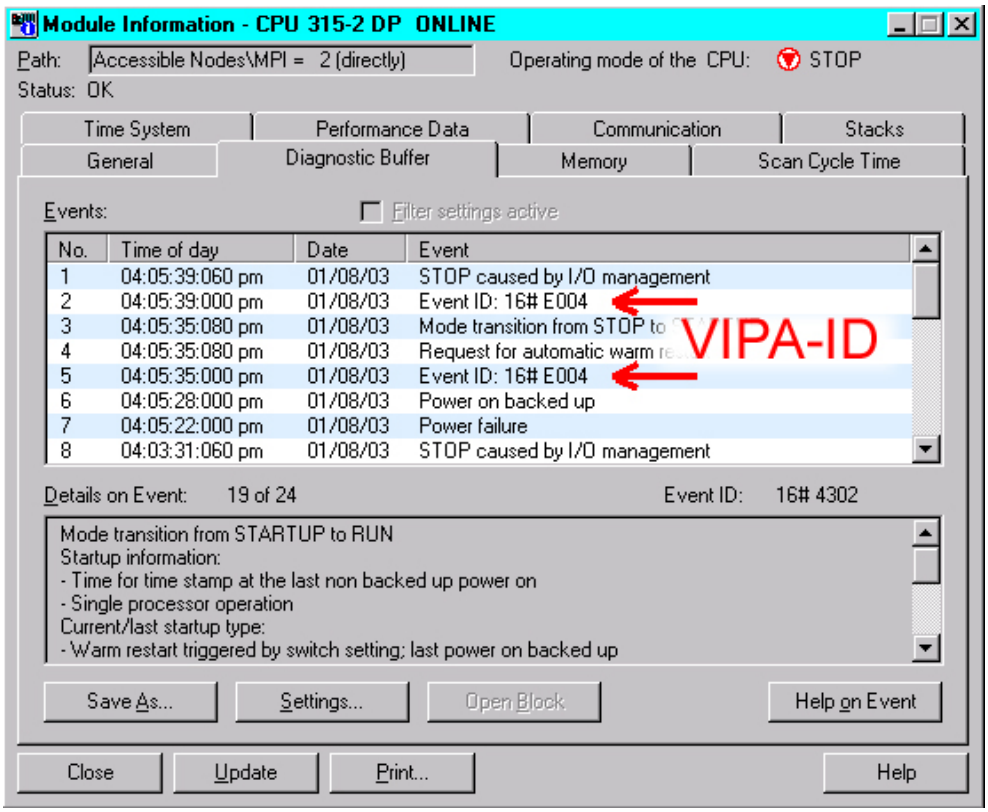
VIPA specific diagnostic entries

Entries in the diagnostic buffer

You may read the diagnostic buffer of the CPU via the SIMATIC manager from Siemens. Besides of the standard entries in the diagnostic buffer, the VIPA CPUs support some additional specific entries in form of event-IDs.

Monitoring the diagnostic entries

To monitor the diagnostic entries you choose the option **PLC** > *Module Information* in the Siemens SIMATIC manager. Via the register "Diagnostic Buffer" you reach the diagnostic window:



The diagnosis is independent from the operating mode of the CPU. You may store a max. of 100 diagnostic entries in the CPU.

The following page shows an overview of the VIPA specific Event-IDs.

Overview of the Event-Ids

Event-ID	Description
0xE003	Error at periphery-access Additional info 1: periphery-address Additional info 2: slot Additional info 3: irrelevant
0xE004	Periphery-address twice used Additional info 1: periphery-address Additional info 2: slot Additional info 3: irrelevant
0xE005	Error at parsing of address information (SDB2000 / SDB1)
0xE006	Error at parsing of DP slave information (SDB2000)
0xE007	Configured in-/output bytes do not fit in the peripheral area (DP slave / High-speed counter)
0xE008	Error at parsing of HighSpeedCounter information (SDB2000)
0xE009	Error at backplane bus access (no Ready / EMV)
0xE010	Not defined module detected at backplane bus Additional info 1: irrelevant Additional info 2: slot Additional info 3: type
0xE011	Masterproject is not possible at Slave CPU or error on slave configuration
0xE012	Error at parametrization / configuration backplane bus
0xE013	Error at access shiftregistermode to digital module
0xE014	Error at Check_Sys
0xE015	Error at access master Additional info 1: irrelevant Additional info 2: slot master (32 = integrated master) Additional info 3: irrelevant
0xE016	Max. blocksize at mastertransfer overrange Additional info 1: periphery-address Additional info 2: slot Additional info 3: irrelevant
0xE017	Error at access to internal DP slave
0xE018	Error at mapping of DP master periphery
0xE019	Error at detection backplane bus
0xE01A	Error at detection of mode (8/9Bit)
0xE0C0	Error not enough memory for storing block (blocksize too big)
0xE0CC	Error at communication MPI / Serial
0xE100	MMC access error
0xE101	MMC file system error
0xE102	MMC FAT error
0xE104	MMC error at saving
0xE200	MMC writing ready (Copy Ram2Rom)
0xE210	MMC reading ready (Loading project after reset)
0xE300	Internal Flash writing ready (Copy Ram2Rom)

Appendix

A Index