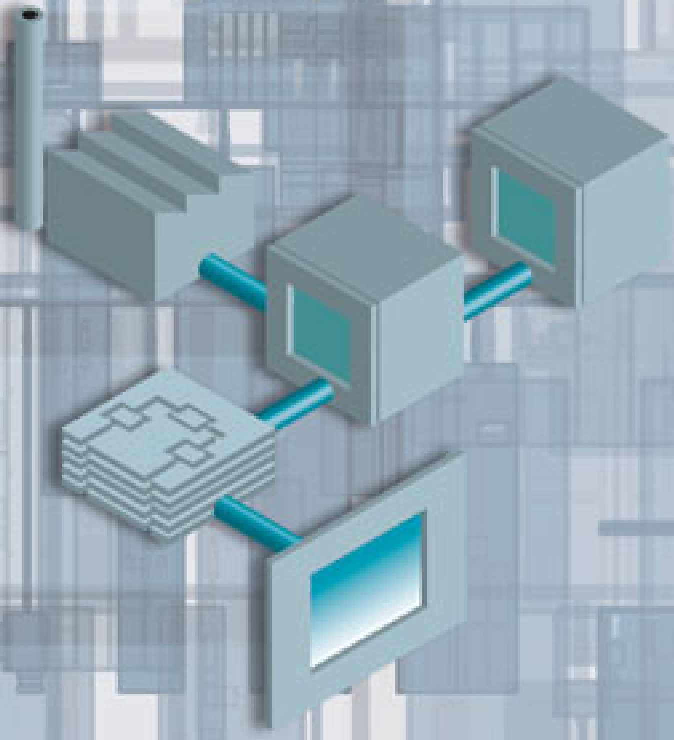
Hans Berger

Automating with SIMATIC

Controllers, Software, Programming, Data Communication, Operator Control and Process Monitoring

SIEMENS



Fifth Edition

Berger Automating with SIMATIC

Automating with SIMATIC

Controllers, Software, Programming,
Data Communication, Operator Control
and Process Monitoring

by Hans Berger

5th revised and enlarged edition, 2013

Publicis Publishing

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

The author, translator and publisher have taken great care with all texts and illustrations in this book. Nevertheless, errors can never be completely avoided. The publisher, author and translator accept no liability, regardless of legal basis. Designations used in this book may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

www.publicis-books.de

Print ISBN: 978-3-89578-387-6
ePDF ISBN: 978-3-89578-676-1

5th edition, 2013

Editor: Siemens Aktiengesellschaft, Berlin and Munich
Translation: Siemens I IA Control Components and Systems Engineering, Erlangen
Publisher: Publicis Publishing, Erlangen

Printed in Germany

Foreword

The automation of industrial plants results in a growing demand for components which are increasingly different and more complex. Therefore a new challenge nowadays is not the further development of highly specialized devices but the optimization of their interaction.

The *Totally Integrated Automation* concept permits uniform handling of all automation components using a single system platform and tools with uniform operator interfaces. These requirement is fulfilled by the new SIMATIC, which provides uniformity for configuration, programming, data management, and communication.

The STEP 7 engineering software is used for the complete configuration and programming of all components. Optional packages for expanding functionalities can also be introduced seamlessly in STEP 7 if they have the same operating philosophy. The SIMATIC Manager of STEP 7 V5.5 and the TIA Portal of STEP 7 V11 coordinate all tools and centrally manage any automation data. All tools have access to this central data management so that duplicate entries are avoided and coordination problems are prevented right from the start.

Integrated communication between all automation components is a prerequisite for “distributed automation”. Communication mechanisms that are tuned to one another permit the smooth interaction of controllers, visualization systems, and distributed I/O without additional overhead. This puts the seminal concept of “distributed intelligence” within reach. Communication with SIMATIC is not only uniform in itself, it is also open to the outside. This means that SIMATIC applies widely-used standards such as PROFIBUS for field devices and Industrial Ethernet and TCP/IP protocol for the best possible connections to the office world and thus to the management level.

The 5th edition of this book provides an overview of the structure and principle of operation of a modern automation system with its state-of-the-art controllers and HMI devices, and describes the expanded facilities of distribution with PROFIBUS and PROFINET. Using the SIMATIC S7 programmable controllers as example, this book provides an insight into the hardware and software configuration of the controller, presents the programming level with its various languages, explains the exchange of data over networks, and describes the numerous possibilities for operator control and monitoring of the process.

Erlangen, July 2012 Hans Berger

Contents

1 Introduction

1.1 Components of the SIMATIC Automation System

The SIMATIC automation system consists of many components that are matched to each other through the concept of “Totally Integrated Automation” (TIA). Totally Integrated Automation means automation with integrated configuration, programming, data storage, and data transfer.

As programmable logic controllers (PLCs), the **SIMATIC S7 controllers** form the basis of the automation system. SIMATIC S7-200 and S7-1200 are micro systems for the low-end performance range – as a stand-alone solution or in a bus network. The SIMATIC S7-300 with standard CPUs and compact CPUs is the system solution with a focus on the manufacturing industry. And as the top-level device with the highest performance power of the SIMATIC controllers, the SIMATIC S7-400 enables system solutions for the manufacturing and process industries.

SIMATIC WinAC (Windows Automation Center) combines the functions of open-loop control, technology, data processing, visualization, and communication on one personal computer (PC) and is the first choice if you have to handle PC applications in addition to classic PLC tasks.

SIMATIC WinCC (Windows Control Center) is the engineering and runtime software for PC-based devices. HMI devices are configured with WinCC as engineering software, while WinCC as runtime software turns personal computers into HMI devices for industrial plants and processes.

SIMATIC HMI stands for Human Machine Interface and is the interface between operators and the machine. From the simplest text display to the operator station with powerful graphics, the human-machine interface provides all the facilities you need for operating and monitoring a machine or plant. Powerful software indicates the state of the plant with event and fault messages, manages recipes and measured value archives, and supports plant operators with troubleshooting, servicing, and maintenance.

SIMATIC NET links all SIMATIC stations and ensures trouble-free data communication. Various bus systems with graded performance also allow third-party devices to be connected, whether field devices in the plant or process PCs connected at the control level. Data traffic can go beyond the limits of various subnets, such as the transfer of automation data such as measured values and alarms or the commissioning and troubleshooting of a central location in the network group.

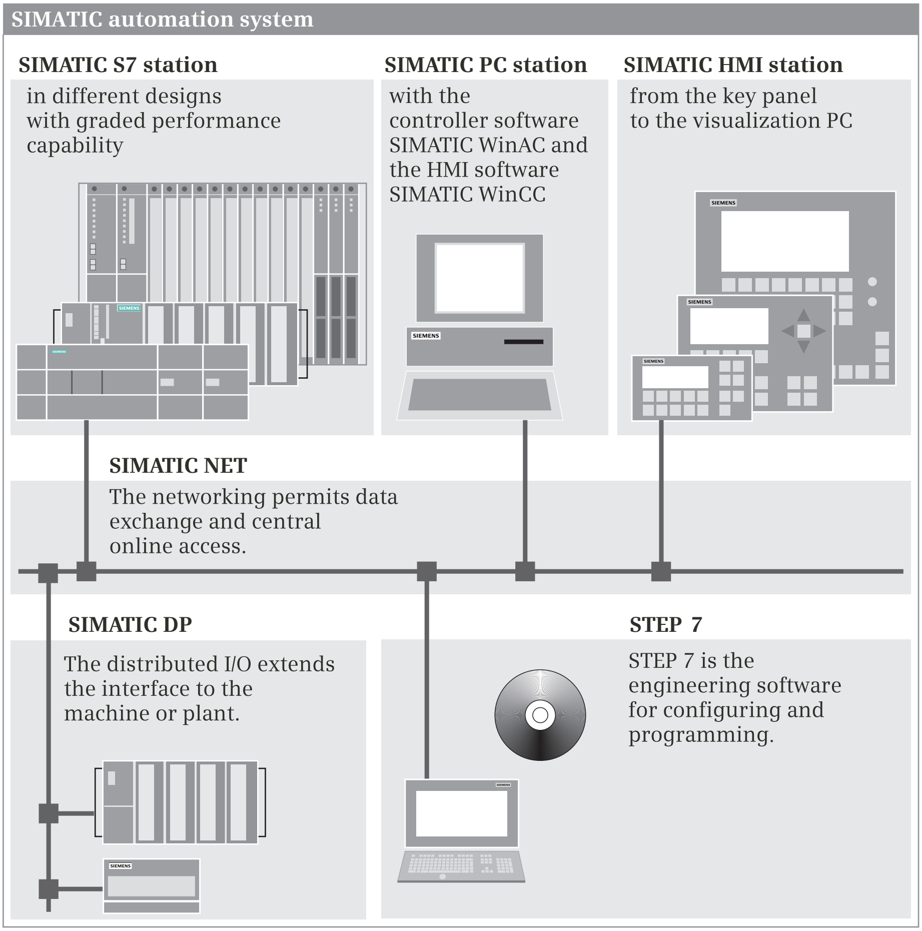

SIMATIC automation system
SIMATIC S7 station
in different designs with graded performance capability
SIMATIC PC station
with the controller software SIMATIC WinAC and the HMI software SIMATIC WinCC
SIMATIC HMI station
from the key panel to the visualization PC
SIMATIC NET
The networking permits data exchange and central online access.
SIMATIC DP
The distributed I/O extends the interface to the machine or plant.
STEP 7
STEP 7 is the engineering software for configuring and programming.
SIEMENS

Fig. 1.1 Components of the SIMATIC automation system

SIMATIC DP stands for distributed I/O. It expands the interface between the central controller and machine or plant with I/O modules directly on site. This distributed I/O, which is spatially separate from the controller, is connected to the central control via the PROFIBUS DP and PROFINET IO bus systems – thus reducing wiring overhead. SIMATIC controllers, ET 200 I/O system modules, and third-party devices can be used as distributed I/O.

The **STEP 7** engineering software is used to configure, parameterize, and program SIMATIC components. The SIMATIC Manager in the “classic” STEP 7 version and the TIA Portal in the updated version are the central tools for managing automation data and related software editors in the form of a hierarchically organized project.

The main activities performed with STEP 7 are:

- ▷ Configuring the hardware
  (arranging modules in racks and parameterizing the module properties)
- ▷ Configuring the communication connections
  (defining communication partners and connection properties)
- ▷ Programming the user program
  (creating the control software and testing the program)

The user program can be created in the programming languages ladder logic (LAD), function block diagram (FBD), statement list (STL), and structured control language (SCL) as “logic control”, or in the programming language GRAPH as “sequence control”

1.2 From the Automation Task to the Finished Program

When you start to solve an automation problem, you have to ask yourself what type of PLC you are going to use. If the machine to be controlled is a small one, will an S7-200 be big enough or do you need an S7-300? Is it better to control the plant with an S7-400 or with a pair of S7-300s? Compact central I/Os in the control cabinet or distributed I/Os in the plant?

The following is a general outline of the steps that lead from the automation task to the finished program. In individual cases, specific requirements must be met.

Choosing the hardware

There are many criteria for selecting the type of controller. For “small” controls the main criteria are the number of inputs and outputs and the size of the user program. For larger plants you need to ask yourself whether the response time is short enough, and whether the user memory is big enough for the volume of data to be managed (recipes, archives). To be able to estimate the resources you need from the requirements alone, you need a lot of experience of previous automation solutions; there is no rule of thumb.

A production machine will probably be controlled by a single station. In this case, the number of inputs/outputs, the size of the user memory and, possibly, the speed (response time) will enable you to decide between the S7-200, S7-1200, S7-300, or the S7-400. How is the machine controlled? What HMI devices will be used?

Spatially distributed systems raise the question of what is overall the better value: the use of centralized or distributed I/O. In many cases distributed I/O not only reduces the wiring overhead needed, but also the response time and the engineering costs. This is possible due to the use of “intelligent” I/O devices with their own user program for preprocessing of signals “on site”.

Distributed automation solutions have advantages: The user programs for the different plant units are smaller with faster response times, and can often be commissioned independently of the rest of the plant. The necessary exchange of data with a "central controller" is particularly easy within the SIMATIC system using standardized bus systems.

Which programming language?

The choice of programming language depends on the task. If it mainly consists of binary signal processing, the graphical programming languages LAD (Ladder Logic) and FBD (Function Block Diagram) are ideal. For more difficult tasks requiring complex variable handling and indirect addressing, you can use the STL (Statement List) programming language, which has an assembly language format. SCL (Structured Control Language) is the best choice for people who are familiar with a high-level programming language and who mainly want to write programs for processing large quantities of data.

If an automation task consists of sequential processes, GRAPH can be used. GRAPH creates sequencers with steps and step enabling conditions that are processed sequentially. All programming languages – including GRAPH – can be used together in a user program. Every program section and "block" can be created with the suitable programming language, depending on requirements.

Creating a project

All the data for your automation solution is collected together in a "project". You create a project using STEP 7. A project is a (software) folder in which all the data is stored in a hierarchic structure. The next level down from the project are the "stations", which in turn contain one or more CPUs with a user program. All these objects are folders which can contain other folders or objects that represent the automation data on the screen. You use menu commands to insert new objects, open these objects, and automatically start the tool required to work with them.

An example: The user program consists of blocks, which are individual program sections with limited functions. All programmed blocks are listed in the block folder. Depending on the programming language used, double-clicking on a block starts the suitable program editor, with which you can alter or expand the program in the block, guided by menus and supported by online help.

Configuring hardware

A project must contain at least one station, either a programmable logic control (PLC) station or a human-machine interface (HMI) station. A PLC station is required to control a machine or plant. After the station is opened, a rack is shown onscreen, to which you can add the desired modules. To do this, drag the required modules from the hardware catalog to the relevant slot. If needed, change the default module properties to meet your requirements.

A project can contain additional stations that you configure in the same manner as the first station. The data transfer between the stations takes place via a subnet.

Using network configuration, you connect the bus interfaces of the “communication modules” to the subnet and thus create the network group.

Writing, debugging and saving the user program

The user program is the totality of all instructions and declarations programmed by the user for signal processing by means of which the machine or plant to be controlled is influenced in accordance with the control task. Large, complex tasks are easier to solve if they are divided into small, manageable units, which can be programmed in “blocks” (subroutines). The division can be process-oriented or function-oriented. In the first case, each program unit corresponds to a part of the machine or plant (mixer, conveyor belt, drilling assembly). In the second case, the program is divided up according to control functions, for example signaling, communication, operating modes. In practice, mixed forms of the two structuring concepts are generally used.

In the user program, signal states and variable values are used that you should preferably address with a name (symbolic addressing). A name is assigned to a memory location in the symbol table or in the PLC variable table. You can then use the name in the program. After you have entered the user program you “compile” it so that it can process the relevant control processor. The user program is created “offline”, without a connection to a controller, and is saved to the hard disk of the programming device.

You can test smaller programs, as well as individual parts of larger programs, offline with the PLCSIM simulation software and thus find and correct any possible errors before the user program is used in the machine or plant.

For commissioning, connect the programming device with the CPU, transfer the program to the CPU user memory, and test it using the STEP 7 testing functions. You can monitor and change the variable values and monitor the processing of the program by the control processor. Comprehensive diagnostic functions allow quick identification of error location and cause.

After commissioning is concluded, you document the project, e. g. in circuit manual format, by means of DOCPRO. With the “classic” version of STEP 7, you can archive an entire project, including documentation, in compressed form.

1.3 How Does a Programmable Logic Controller Work?

In conventional control engineering, a control task is solved by wiring up contactors and relays individually, i.e. depending on the task. They are therefore referred to as contactor and relay controllers, and electronic controllers assembled from individual components are referred to as *hard-wired programmed* controllers. The “program” is in the wiring. *Programmable logic* controllers, on the other hand, are made up of standard components that implement the desired control function by means of a userprogram.

SIMATIC S7 is an automation system that is based on programmable logic controllers. The solution of the control task is stored in the user memory on the CPU in the form of program instructions. The control processor reads the individual instructions sequentially, interprets their content, and executes the programmed function.

The CPU module contains an additional program: the operating system. It ensures the execution of the device-internal operating functions, such as communication with the programming device and backing up data in the event of power failure. The operating system also initiates the processing of the user program, either recurring cyclically or dependent on a trigger event such as an alarm.

Cyclic program processing

The prevalent processing type of the user program for programmable logic controllers is cyclic program processing: After the user program has been completely processed once, it is then processed again immediately from the beginning. The user program is also executed if no actions are requested "from outside", such as if the controlled machine is not running. This provides advantages when programming: For example, you program the ladder logic as if you were drawing a circuit diagram, or program the function block diagram as if you were connecting electronic components. Roughly speaking, a programmable logic controller has characteristics like those of a contactor or relay control: The many programmed operations are effective quasi simultaneously "in parallel".

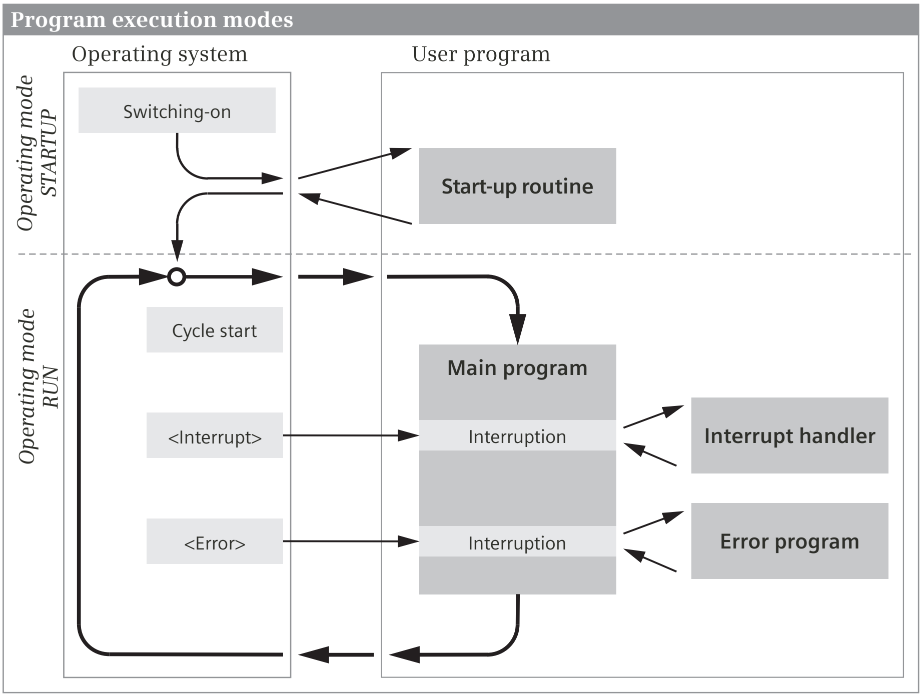


Fig. 1.2 Execution of the user program in a SIMATIC controller

After the power is switched on and the operating function test runs, the operating system starts an (optional) start-up routine once (Fig. 1.2). The main program is next in the sequence. If it has been processed to the end, processing begins again immediately at the start of the program. The main program can be interrupted by alarm or error events. The operating system then starts an interrupt handler or error program. If the interruption-controlled program has been completely processed, the program processing continues from the point of interruption in the main program. A priority scheduler controls the program execution order if several interrupt events occur simultaneously.

The user program is made up of blocks. There are several block types. Organization blocks represent the interface to the operating system. After a start event occurs (power up, cycle start, alarm, error), the operating system calls the associated organization block. It contains the appropriate user program for the event. An organization block only has to be programmed if the automation solution requires it. The program in an organization block can, if needed, be structured by function blocks (blocks with static local data) and functions (blocks without static local data). Data blocks in the user memory or the bit memory address area in the system memory are available to store user data.

1.4 The path of a binary signal from the sensor to the program

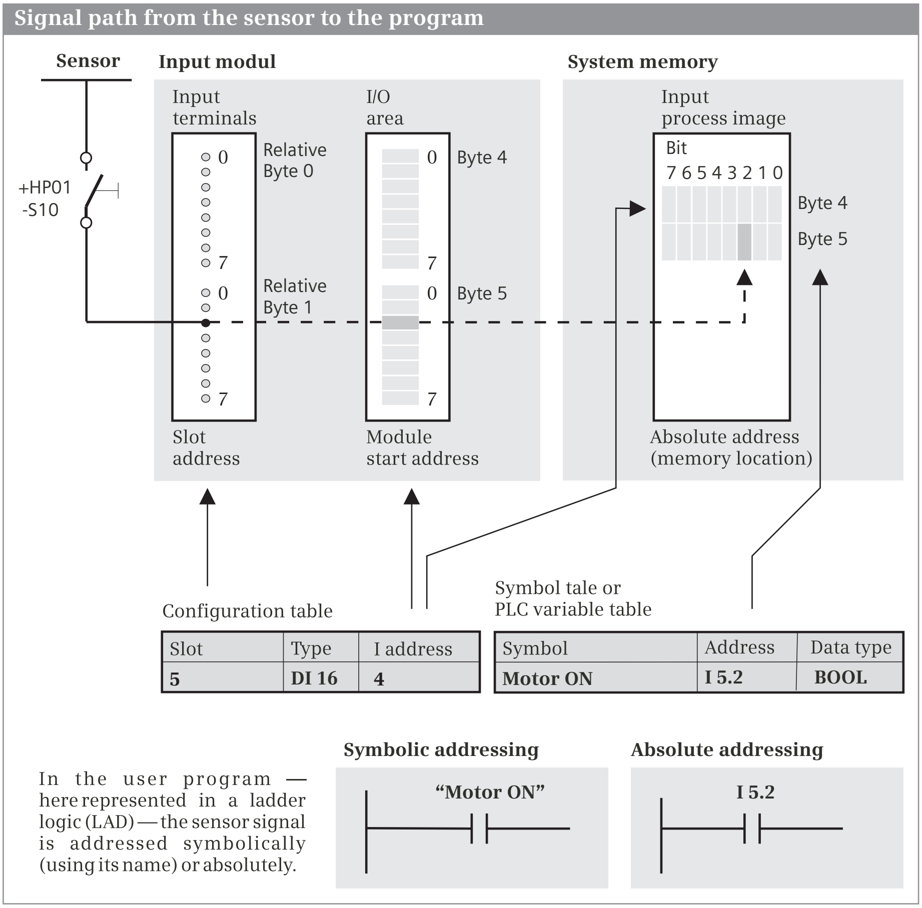
In order to do its job, the control processor in the controller needs to be connected to the machine or plant it is controlling. I/O modules that are wired to the sensors and actuators create this connection.

Connection to the programmable logic controller, module address

When wiring the machine or plant, you define which signals are connected to the programmable logic controller, and where. An input signal, e.g. the signal from pushbutton +HP01-S10 with the significance “Switch on motor”, is connected to a specific terminal on an input module (Fig. 1.3).

Each module is located in a particular slot on the rack, the number of which is the slot address. In addition, each I/O module has a so-called “logical” address. The user program uses this address to address a signal of the module. In the logical address, the binary signals are aggregated into bytes (bundles of eight bits). Bytes are numbered starting from zero – even with gaps. The bit address is counted from 0 to 7 for each byte.

You determine the slot address by plugging the module into a certain place on the rack. STEP 7 assigns the logical address consecutively, which you can change in the configuration table. The first byte of the module receives the module start address, which is the lowest address of the module. If a module has more bytes, the byte addresses are automatically incremented. In the example, the module has the module start address four – either set by STEP 7 as default or set by you – and the



Slot	Type	I address
5	**DI 16**	**4**

Symbol	Address	Data type
Motor ON	**I 5.2**	**BOOL**

In the user program — here represented in a ladder logic (LAD) — the sensor signal is addressed symbolically (using its name) or absolutely.

Fig. 1.3 Path of a signal from the sensor to its use in the program

next byte thus automatically has the number five. The "switch on motor" signal is connected to terminal two of the second byte (relative byte 1). By specifying the module address as 4, you are given the address of the signal: "Input in byte 5 to bit 2" or in short: I 5.2.

Symbolic address

The address "I 5.2" denotes the memory location and is the absolute address. It is much easier if you can address this signal in the program with a name that matches the meaning of the signal, for example "switch on motor". This is the symbolic address. You can find the assignment of absolute addresses to symbolic addresses in the symbol table or the PLC variable table. In this table, the "global" symbols are defined; these are the symbols that are valid in the entire user program. You specify the symbols that are valid for only one block ("local" symbols) when programming the block.

Process images

When you use the signal “switch on motor” or I 5.2 in the program, you do not address the signal memory in the module but a storage area within the CPU. This storage area is referred to as the “process image”. It is also available for the outputs, which in principle are treated in the same way as the inputs.

The CPU operating system automatically transfers the signal states between the modules and the process image in each program cycle. It is also possible to address the signals directly on the modules from the user program. However, the use of a process image has advantages compared to direct access, including much faster access to the signal states and the steady signal state of an input signal during a program cycle (data consistency). The disadvantage is the increased response time, which is also dependent on the program execution time.

1.5 Data management in the SIMATIC automation system

The automation data is present in various memory locations in the automation system. First of all, there is the programming device. All automation data of a STEP 7 project is saved on its hard disk. Configuration and programming of the project data with STEP 7 are carried out in the main memory of the programming device (Fig. 1.4).

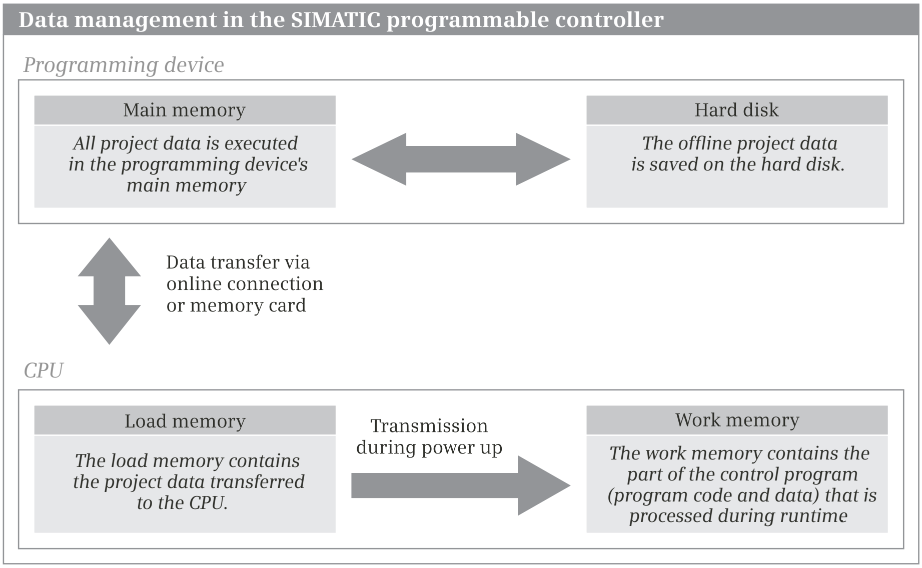

Data management in the SIMATIC programmable controller
Programming device
Main memory
All project data is executed in the programming device's main memory
Hard disk
The offline project data is saved on the hard disk.
Data transfer via online connection or memory card
CPU
Load memory
The load memory contains the project data transferred to the CPU.
Transmission during power up
Work memory
The work memory contains the part of the control program (program code and data) that is processed during runtime

Fig. 1.4 Data management in the SIMATIC automation system

The automation data on the hard disk is also referred to as the *offline project data*. Once STEP 7 has appropriately compiled the automation data, this can be downloaded to a connected programmable logic controller. The data downloaded into the user memory of the CPU is known as the *online project data*.

The user memory on the CPU is divided into two components: The *load memory* contains the complete user program, including the configuration data, and the *work memory* contains the executable user program with the current control data. The load memory of the CPU 1200 can be expanded with a plug-in memory card. For the CPU 300, the load memory is on the memory card, which therefore must always be inserted in the CPU for use. The memory card for the CPU 1200 and CPU 300 is an SD Card, for failsafe storage of the automation data. On the CPU 400, the memory card expands the load memory; here, the memory card is a RAM card (so that the user program can be changed during testing), or a FEPROM card (for failsafe storage of the user program).

2 SIMATIC Controllers – the Hardware Platform

SIMATIC controllers – the core of the automation systems – control production machines, manufacturing plants, or industrial processes. The following description mainly refers to programmable logic controllers (PLC). Siemens also offers PC-based SIMATIC controllers.

SIMATIC S7 are programmable logic controllers (PLCs), which are available in four designs:

▷ SIMATIC S7-200, the compact micro PLC

▷ SIMATIC S7-1200, the modular „micro PLC“,

▷ SIMATIC S7-300, the modular PLC of the mid performance range

▷ SIMATIC S7-400, the modular PLC of the upper performance range

The S7-200 station consists of a basic unit and can be expanded with additional modules. In S7-300/400 stations, the power supply unit, the CPU module and the I/O modules are installed in the same mounting rack. This centralized configuration can be extended with expansion racks for the installation of additional I/O modules. The expansion rack may be a remote installation, that is it can be placed at a distance from the central rack. You use the STEP 7 Micro programming language to program the SIMATIC S7-200 controller. STEP 7 with its different programming languages is provided for the controllers of the SIMATIC S7-300/400 series.

PC-based automation is the umbrella term for PLCs based on a personal computer (PC):

▷ The industrial PC is available as a Rack PC or Box PC.

▷ The SIMATIC Panel PC is a combination of HMI device and controller.

▷ **SIMATIC WinAC** is the generic term for the program packages of SIMATIC PC-based Automation. WinAC runs on a standard PC under a Windows operating system. Distributed I/O modules form the link to the process.

With SIMATIC PC-based Control, the controller can take the form of a purely software solution (Software PLC) or a plug-in card (Slot PLC).

SIMATIC DP are modules installed on site at the machine or in the plant and are connected to the master station via PROFIBUS DP and/or PROFINET IO. Many SIMATIC CPUs feature an integrated PROFIBUS or PROFINET interface, which greatly facilitates the connection of distributed I/O. Since operation on the

PROFIBUS and PROFINET is standardized independent of the vendor, it is also possible to connect third-party devices to a SIMATIC controller.

2.1 Components of a SIMATIC Station

A complete programmable controller including all I/O modules is referred to as a “station”. The core is the CPU, which is expanded with I/O modules if needed.

The following list shows the components a SIMATIC station can consist of:

▷ Racks
These accommodate the modules and form the basis for central and expansion units. The S7-1200 and S7-300 use a simple mounting rail; its length is determined by the number and width of the modules. The S7-400 uses an aluminum rack that has a defined number of slots with backplane bus and bus connectors.

▷ Power supply (PS)
It provides the internal supply voltage; the input voltage is either 120/230 V AC voltage or 24 V DC voltage.

▷ Central processor unit (CPU)
This stores and processes the user program; communicates with the programming device and any other stations over the MPI bus; controls the central and distributed I/O modules; and can also be a DP slave on PROFIBUS DP or IO Device on PROFINET IO.

▷ Interface modules (IM)
These connect the racks with each other.

▷ Signal modules (SM)
These adapt the signals from the controlled plant to the internal signal level or control contactors, actuators, lights, etc. Signal modules are available as input and output modules for digital and analog signals and can also be used to connect sensors and actuators located in hazardous areas of zones 1 and 2.

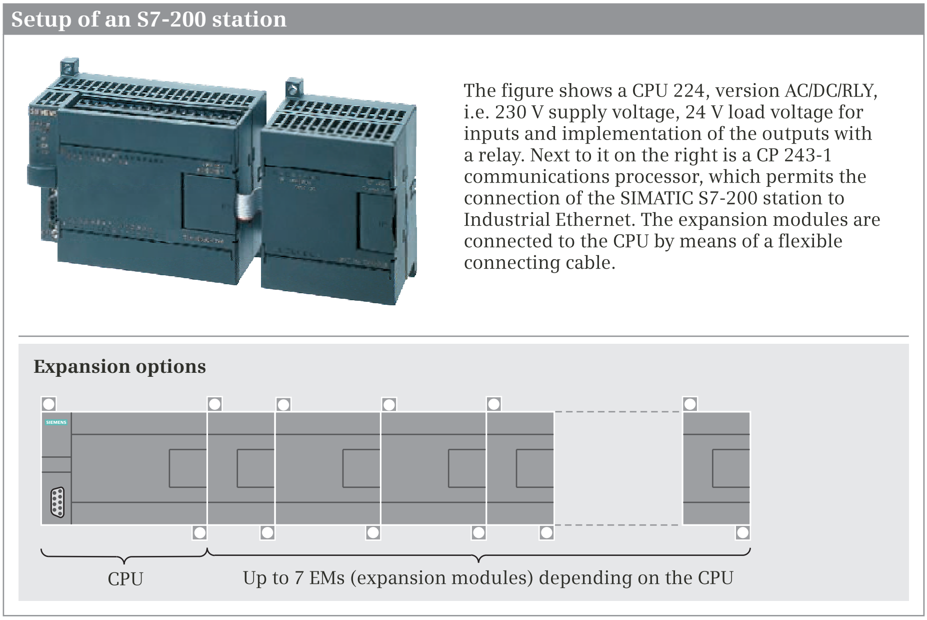
▷ Function modules (FM)
These handle complex or time-critical processes independently of the CPU, e.g. counting, position control, and closed-loop control.

▷ Communications processors (CP)
These connect the SIMATIC station with the subnets such as Industrial Ethernet, PROFIBUS FMS, AS-Interface, or a serial point-to-point connection.

The distributed I/O modules connected to a station are also part of this station. If the distributed I/O system is connected over PROFIBUS DP, a DP master controls “its” DP slaves and thus the field units; if the connection is over PROFINET IO, an IO-Controller controls the IO-Devices. The DP slaves or IO-Devices are integrated in the address area of the centralized I/O system and are principally treated just like the I/O modules installed locally in the central and expansion racks.

2.2 The Micro PLC SIMATIC S7-200

The SIMATIC S7-200 is a compact micro PLC that replaces relay and contactor control arrangements and increasingly also specific electronic circuits in mechanical and system engineering applications. It can be used as a stand-alone system or in a network with other control systems. Various expansion modules for the connection to the machine or plant are available. STEP 7 Micro/WIN is used for programming a SIMATIC S7-200.


Setup of an S7-200 station

The figure shows a CPU 224, version AC/DC/RLY, i.e. 230 V supply voltage, 24 V load voltage for inputs and implementation of the outputs with a relay. Next to it on the right is a CP 243-1 communications processor, which permits the connection of the SIMATIC S7-200 station to Industrial Ethernet. The expansion modules are connected to the CPU by means of a flexible connecting cable.

Expansion options

Fig. 2.1 Setup and expansion options for an S7-200 station

Compact-type basic modules and expansion modules

There are various S7-200 basic modules graded by configuration and performance capability. The basic module contains the CPU and – in different types and numbers for each basic module – integrated inputs and outputs at 24 V DC, 100 V AC/120 V to 230 V AC, as well as relay outputs. Fast alarm and counter inputs enhance the real-time performance. A digital value can be set without a programming device using a screwdriver via a potentiometer on the basic module.

You can increase the number of inputs/outputs by connecting an expansion module. Such modules are available for both digital and analog inputs/outputs. The expansion modules are snapped onto the rail (e.g., standard DIN rail) next to the basic module and electrically connected by means of a bus connector.

Operating modes of a CPU 200

A CPU 200 has two operating modes, STOP and RUN, which are indicated by LEDs on the front of the CPU. The user program is not executed in STOP mode. The CPU must be in STOP mode to load the user program or the hardware configuration. In RUN mode, the CPU executes the control functions in the user program. To switch between modes, use the mode switch on the CPU or the programming device in online operation.

User memory in a CPU 200

The user memory of a CPU 200 consists of the program memory and the data memory. When loading the user program to the CPU with a programming device, the program code and the data are stored in an EEPROM memory, where they are protected against power outages. The current data is then copied to the RAM area where it is processed during runtime.

The user program can also be transferred via a memory module that can be plugged into the CPU. Additional data such as recipes, data log configurations, and documentation files in any format can be stored on this memory module. This additional data can also be loaded from a programming device to the CPU if the memory module is plugged in.

On a CPU 200, you can use a program to store a variable value in the failsafe EEPROM memory area. Note that – due to the physical limitations of the medium – only a limited number of write operations is allowed. Excessive writing, for example with every program cycle, reduces the lifespan of the EEPROM memory. The data on any plugged-in memory module is not changed.

Table 2.1 Quantity structure of the S7-200 CPUs

CPU	**Integrated I/O** Input/output channels			**Expandable with** Expansion modules (EM) of the S7-22x series	**Memory configuration** Data memory / program memory / with active RunTime Edit	**Counter** Number / frequency
	DI	DO	AI *)			
CPU 221	6	4	1	cannot be expanded	2 KB/ 4 KB	4/30 kHz
CPU 222	8	6	1	Max. 2 EMs	2 KB/ 4 KB	4/30 kHz
CPU 224	14	10	2	Max. 7 EMs	8 KB/ 12 KB/ 8 KB	6/30 kHz
CPU 224 XP	14	10	2	Max. 7 EMs	10 KB/ 16 KB/ 12 KB	2/200 kHz 4/30 kHz
CPU 226	24	16	2	Max. 7 EMs	10 KB/ 24 KB/ 16 KB	6/30 kHz

*) Analog potentiometer with 8-bit resolution

Retentive behavior

Part of the data memory is reserved for retentive data defined by the user. Retentive data is even retained if the supply voltage is interrupted. A high-performance capacitor or an optional battery module bridges the loss of voltage.

If, when the power supply is switched on, the high-performance capacitor and – if present – the optional battery module do not exhibit errors, the values of the retentively configured variables (bit memories, timers, counters) are not changed and the non-retentive memory areas are cleared.

If the contents of the RAM cannot be buffered, the CPU 200 deletes all user data, retrieves the user program from the EEPROM memory area, and restores the first 14 bytes of the bit memories from the non-volatile memory, as long as these bytes have previously been configured as retentive.

Communication capability just like a “large” station

Depending on its design, a CPU 200 hasone or two RS-485 interfaces, which as point-to-point interfaces (PPI) permit the connection of programming and HMI devices and allow linking to another CPU 200. As multi-point interfaces (MPI), they permit the operation of a CPU 200 as MPI slave on an MPI master, e.g. a CPU 300/400. This interface can also be used as a freely programmable interface with interrupt facility for serial data exchange with third-party devices such as barcode readers using the ASCII protocol.

Various supplementary modules expand the communication possibilities, such as the EM 241 modem for remote maintenance and diagnostics, the EM 277 DP module for operation of a CPU 200 as PROFIBUS DP slave, and the CP 243-2 (AS-Interface master), CP 243-1 (connection to Industrial Ethernet), and CP 243-1 IT (Industrial Ethernet with IT communication) communications processors.

Operator control and monitoring with S7-200

For the S7-200 controllers, the Micro Panels are the ideal operator control and monitoring devices. They are connected to the interface of the CPU by means of a supplied cable. For a description of these HMI devices, refer to chapter 7.5 "Micro Panels" on page 259.

Configuring and programming with STEP 7-Micro/Win

STEP 7-Micro/Win is the engineering software for the controllers of the S7-200 series and runs under Windows 2000 and Windows XP. A CPU 200 can be programmed with the statement list (STL), ladder logic (LAD), and function block diagram (FBD) programming languages. These programming languages for S7-200 differ in function and handling from the programming languages for the other SIMATIC S7 automation systems.

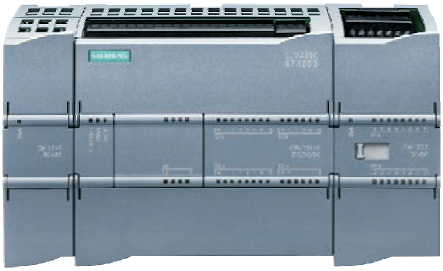
2.3 The SIMATIC S7-1200 Modular Micro Controller

The youngest member of the controller family is the SIMATIC S7-1200. An S7-1200 automation system consists of a central processing unit which – depending on the CPU version – can be expanded with digital and analog input and output modules. Using the PROFINET interface, the central processing unit can be connected to Industrial Ethernet. S7-1200 is configured and programmed inside TIA Portal using STEP 7 Basic/Professional.

Compact design for S7-1200

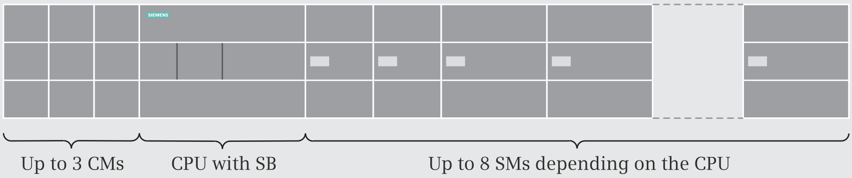
Three central processing units with different performance capability in each of the variants DC/DC/DC, DC/DC/relay, and AC/DC/relay are offered. The first specification refers to the supply voltage (24 V DC, 85 to 264 V AC), the second to the signal voltage of the digital inputs (24 V DC), and the third to the type of digital outputs (24 V DC electronic or relay outputs 5 to 30 V DC, 5 to 250 V AC). Table 2.2 shows the expandability and the memory configuration. Rapid counters with counting frequencies of up to 100/200 kHz are integrated with the central processing unit, which in connection with a pulse generator and the "Axis" technology object can control a stepper motor or a servomotor with pulse interface.

Setup of an S7-1200 station



The figure shows a CPU 1214C in the center, version DC/DC/DC, i.e. 24 V supply voltage and 24 V load voltage for the inputs and outputs. The communication modules (a CM 1341 for RS485 connection in this figure) are attached to the left of the CPU; the signal modules (an SM 1223 with 8 x 24 V digital inputs and 8 x 24 V/0.5 A digital outputs in this figure) are plugged in to the right of the CPU. A signal board can be operated on the CPU (an SB 1232 with one analog output in this figure).

Expansion options



CM	Communication module
SM	Signal module
SB	Signal board

Fig. 2.2 Setup and expansion options for an S7-1200 station

Table 2.2 Quantity structure of the S7-1200 CPUs

CPU	Onboard input/output channels		Expandable with
			SB = signal board
			SM = signal module
			CM = comm. module
				Memory configuration
				Load memory/
				work memory /
				retentive memory

	digital	analog		
CPU 1211C	6 DI/4 DO	2 AI/-	1 SB, 3 CM	1 MB/25 KB/2 KB
CPU 1212C	8 DI/6 DO	2 AI/-	1 SB, 2 SM, 3 CM	1 MB/25 KB/2 KB
CPU 1214C	14 DI/10 DO	2 AI/-	1 SB, 8 SM, 3 CM	2 MB/50 KB/2 KB

A two-tier design is possible using a 2-meter long extension cable. But the number of modules that can be used is not changed as a result.

Operating modes of the CPU

The CPU 1200 has the operating modes STOP, STARTUP, and RUN. In STOP, the user program is not processed, but the CPU is capable of communication and can, for example, be loaded with the user program. If the supply voltage is switched on, the CPU is first in STOP mode, then switches to STARTUP mode, in which it parameterizes the modules and passes through a user start-up routine, and after an error-free start reaches RUN mode. Now the main user program is processed. In the case of a “serious” error, the CPU switches from STARTUP or RUN mode back to STOP mode.

The modes are switched with the programming device in online operation. A mode switch is not provided.

The user memory consists of a load memory and a work memory

The user program is located on the CPU in two areas: in the load memory and work memory. The load memory contains the entire user program including configuration data; it is integrated into the CPU or available on a plug-in memory card. The work memory in the CPU is integrated fast RAM that contains the execution-relevant program code and user data.

The programming device transfers the entire user program, including configuration data, to the load memory. The operating system interprets the configuration data, and parameterizes the CPU and – during startup – the I/O modules. The execution-relevant program code and user data are copied into the work memory.

Retentivity without backup battery

Retentivity means that the contents of a memory area remain after the supply voltage is switched off and on again. With a CPU 1200, this behavior makes possible a retentive memory for bit memories and data variables and non-volatile load memory for the user program, so it does not require a backup battery. During runtime, data areas such as recipes can be read from the load memory with a user program, and other data areas such as archives can be written to the load memory.

A memory card expands the load memory

The memory card for S7-1200 is an SD Card that has been pre-formatted by Siemens. The memory card can be set as a program card or a transfer card. As a program card, the memory card replaces the integrated load memory and must be inserted during operation of the CPU. As a transfer card, the memory card allows the user program to be transferred without a programming device. It is also possible to update the CPU firmware with a transfer card.

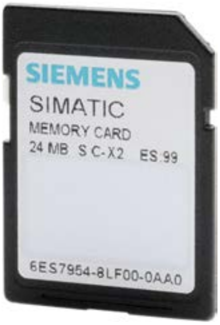


Fig. 2.3 SIMATIC Memory Card

The signal board extends the onboard I/O

The signal board (SB) expands the onboard I/O without changing the dimensions of the CPU. The associated slot is located on the front of the CPU.

Signal boards are available with 24 V and 5 V digital inputs and outputs, which can be operated at a frequency of up to 200 kHz. The frequency of the high-speed counters (HSC) and pulse generators integrated into the CPU can thus be increased.

Voltage transmitters (± 10 V), current transmitters (0 to 20 mA), thermocouples (type J or K), or resistance thermometers (PT 100 or PT 1000) can be connected to a signal board with analog input module. The signal board with analog output module is available for ± 10 V output voltage or 0 to 20 mA output current.

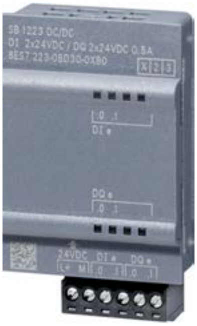


Fig. 2.4 Signal Board 1223

High-speed counter

A high-speed counter (HSC) is a high-speed hardware counter in the CPU. The CPU 1211 contains three counters, the CPU 1212 has four counters, and the CPU 1214 has six counters. A high-speed counter as up/down counter has a counting range of $\pm 2^{31}$. There are special counter inputs on the CPU to capture the pulse train output; these allow a maximum frequency of 100 kHz. If a signal board with fast inputs is used, the maximum counting frequency increases to 200 kHz.

Pulse generators

A pulse generator generates pulses at a special output channel. If the output channel to the onboard I/O belongs to the CPU, the maximum pulse frequency is 100 kHz. If the output channel is on the signal board, the maximum achievable frequency increases to 200 kHz. Each CPU 1200 has two pulse generators. The pulse generators have two modes of operation: PTO (pulse train output) and PWM (pulse width modulation).

“Axis” technology object

The axis technology object controls a stepper motor or a servomotor with pulse interface. It forms the interface between the motion control instructions in the user program and the drive. Motion profiles of the drive can be created using the job table technology object.

A maximum of two axis technology objects can be set up in each CPU 1200. Each axis technology object requires a pulse generator in PTO mode, an HSC, and a rapid pulse output channel.

“PID controller” technology object

The PID controller technology object is available in two versions: as a universal controller (PID_Compact) for technical processes with continuous I/O signals and as a controller for motor-operated devices such as valves (PID_3STEP) that use digital signals to open and close.

A PID controller requires an analog input channel for the actual value and an analog output channel for the (analog) manipulated variable. Digital output channels are required if the manipulated variable is issued as a pulse width modulated signal (PID_Compact) or as a close/open signal (PID_3STEP). The PID controller technology object calculates the PID shares independently during the self-adjustment at initial start. Further optimization is possible by means of fine adjustment during operation.

Peripheral expansion with digital and analog modules

The onboard peripherals of a CPU 1212 or CPU 1214 can be expanded with two or eight signal modules (SM). Digital modules are available with 8 or 16 binary channels for 24 V input and output voltage or with relay outputs. Voltage transmitters, current transmitters, thermocouples, or resistance thermometers can be connected to an analog input module with 8 or 16 analog channels. The analog output module is available with 2 or 4 analog channels for ± 10 V output voltage or 0 to 20 mA output current. Which properties are important in the selection of I/O modules can be seen in chapters 2.10 "Process Connection with Digital Modules" on page 47 and 2.11 "Process connection with analog modules" on page 48.

Communication for S7-1200

The PROFINET interface connects a CPU 1200 with other devices via Industrial Ethernet. This can be a programming device, a Basic Panel, or another PLC. Open User Communication performs the data exchange between the programmable controllers. If only one device is connected, this can be done directly with a standard or crossover cable. The connection of multiple devices requires an interface multiplier, for example, the CSM 1277 compact switch module, to which up to three additional stations can be connected.

A CPU 1200 may be the IO Controller in a PROFINET IO system. Additional information on PROFINET IO is available in chapter 6.12 "Distributed I/O with PROFINET IO" on page 225.

The CM 1241 communication module permits a point-to-point connection based on RS232 or RS485. With a CB 1241 communication board – plugged into the front of the CPU – a point-to-point connection based on RS485 can be set up without changing the dimensions of the CPU. The following standard protocols are available: ASCII protocol, MODBUS protocol with RTU format, and USS drive protocol.

The CM 1242-5 (DP slave) and CM 1243-5 (DP master) communication modules permit the connection of a CPU 1200 to a PROFIBUS DP master system. Further information on PROFIBUS DP can be found in chapter 6.15 "Distributed I/O with PROFIBUS DP" on page 242.

Operator control and monitoring with S7-1200

The Basic Panels are the ideal operator control and monitoring devices for the S7-1200 controllers. They are connected via the PROFINET interface and configured with WinCC Basic, which is supplied with STEP 7 Basic/Professional V1x inside TIA Portal. The HMI devices for S7-1200 are described in chapter 7.2 "Basic Panels" on page 255.

Configuring and programming with STEP 7 inside TIA Portal

A CPU 1200 is configured and programmed with the STEP 7 Basic/Professional engineering software inside TIA Portal. Programming in ladder logic (LAD), function block diagram (FBD), and structured control language (SCL) is possible with version V11 of STEP 7 Basic and V2.x of the CPU firmware.

STEP 7 inside TIA Portal contains all functions for hardware configuration, networking with PROFIBUS and PROFINET, and programming and testing the user program. The engineering software is described in chapter 3.4 "Editing projects with STEP 7 inside TIA Portal" on page 70.

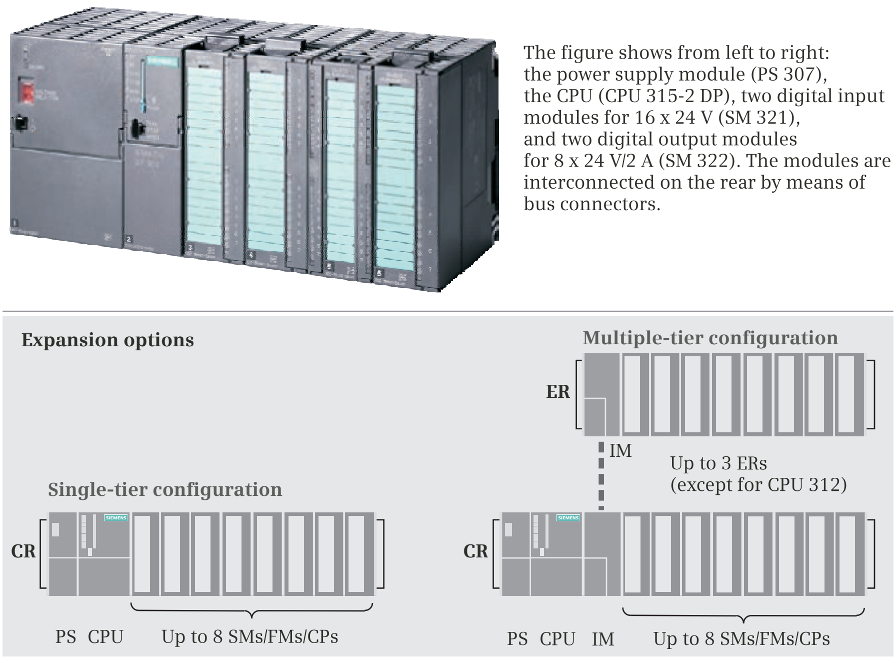
2.4 The SIMATIC S7-300 modular mini controller

SIMATIC S7-300 is the modular mini controller system for the lower and medium performance ranges. Possible applications include the control of packaging, textile and special-purpose machinery. An S7-300 station consists of a central controller and – as required – up to four expansion devices.

Central unit

The central controller contains the CPU and up to 8 I/O modules. The CPU requires a supply voltage of 24 V DC, which, for example, can be drawn from one of the power supplies on the mounting rail to the left of the CPU. A serial backplane bus that transfers both the I/O signals and the parameterization data connects the mod-

Setup of an S7-300 station



The figure shows from left to right: the power supply module (PS 307), the CPU (CPU 315-2 DP), two digital input modules for 16 x 24 V (SM 321), and two digital output modules for 8 x 24 V/2 A (SM 322). The modules are interconnected on the rear by means of bus connectors.

PS	Power supply	SM	Signal module	CR	Central rack
CPU	Central processor unit	FM	Function module	ER	Expansion rack
IM	Interface module	CP	Communications processor		

Fig. 2.5 Setup and expansion options for an S7-300 station

ules to one another. The bus is routed from module to module via bus connectors. It is only available where modules are inserted.

The slots in the rack are numbered: Slot 1 is reserved for the power supply, even if there is no power supply, slot 2 is assigned to the CPU, and slot 3 is assigned to the interface module (even if there is no interface module). Slots 4 to max. 11 are intended for the I/O modules, which are plugged in without gaps. The slot number is independent of the module width.

Expansion unit

If a single-tier configuration is not enough, with the CPU 314 and above you can choose either a two-tier configuration (IM 365) or up to a four-tier configuration (IM 360/IM 361) with up to 32 additional I/O modules.

The IM module is inserted between the CPU and the first I/O module. Like the central rack, the expansion rack consists of a mounting rail with snapped-on modules. The receiver IM, which establishes the connection to the central rack, occupies slot 2; a send IM leading to another rack occupies slot 3, and additional I/O modules are inserted without gaps into the slots 4 to max. 11.

Broad field of application

A broad range of **standard CPU 300s** covers the lower and middle performance range in the manufacturing industry. With a comprehensive range of modules and flexible networking capability, the requirement is met for optimum adaptation to the machine or plant to be controlled.

Table 2.3 Quantity structure of standard S7-300 CPUs

CPU	**Address ranges**					**Memory configuration**
	Peripherals (bytes)	Process image (bytes) *)	Bit memory (bytes)	SIMATIC timers	SIMATIC counters	Load memory / work memory / retentive memory
312	1024	1024 / 128	256	256	256	8 MB/32 KB/32 KB
314	1024	1024 / 128	256	256	256	8 MB/128 KB/64 KB
315-2 DP	2048	2048 / 128	2048	256	256	8 MB/256 KB/128 KB
315-2 PN/DP	2048	2048 / 128	2048	256	256	8 MB/384 KB/128 KB
317-2 DP	8192	2048 / 256	4096	512	512	8 MB/512 KB/256 KB
317-2 PN/DP	8192	8192 / 256	4096	512	512	8 MB/1024 KB/256 KB
319-3 PN/DP	8192	8192 / 256	8192	2048	2048	8 MB/2048 KB/700 KB

*) maximum/preset (both for inputs and outputs)

In addition to the equipment for standard CPUs, the **3xxC compact CPUs** also contain technological functions (counting, measuring, closed-loop control, positioning) with integral inputs/outputs, thus making compact design of mini controllers possible.

The **3xxF failsafe CPUs** permit the construction of a failsafe automation system for plants with increased safety requirements. Failsafe and standard I/O modules can be operated both in centralized and distributed configurations.

The **3xxT technology CPUs** combine control functions with simple motion control functions. The control component is designed like a standard CPU; it is configured, parameterized, and programmed using STEP 7 V5.x. The technology objects and the motion control component require the S7 Technology option package, which is integrated in the SIMATIC Manager following installation.

Operating modes of a CPU 300

The CPU 300 has the operating modes STOP, STARTUP, HOLD, and RUN. After switching on the power supply, the CPU is initially in STOP mode. The user program is not processed, but the CPU is still capable of communication, i.e. the user program can be loaded or the diagnostic buffer can be read, for example.

The CPU is switched into RUN mode with the mode switch on the CPU or with a programming device in online mode. Here, the STARTUP mode is executed in which the modules are parameterized and a user start-up routine is executed. The main user program is executed in RUN mode.

In the operating modes STARTUP and RUN, test functions can be performed that lead to the HOLD mode. The execution of the user program is stopped at predetermined points in the program. The outputs to the machine to be controlled are switched off or set to a pre-defined value for safety reasons.

The user memory consists of a load memory and a work memory

The user program is located on the CPU in two areas: in the load memory and work memory. The load memory contains the entire user program including configuration data; it is located on a plug-in Micro Memory Card. The work memory in the CPU is integrated fast RAM that contains the execution-relevant program code and user data.

The programming device transfers the entire user program, including configuration data, to the load memory. The operating system interprets the configuration data, and parameterizes the CPU and – during startup – the I/O modules. The execution-relevant program code and user data are copied into the work memory.

Retentivity without backup battery

Retentivity means that the contents of a memory area remain after the supply voltage is switched off and on again. With a CPU 300 this behavior enables retentive memory for bit memories, data variables, and timer and counter functions for SIMATIC. The user program is also stored in non-volatile form in the load memory on the Micro Memory Card so that a backup battery is not needed. During runtime, data areas such as recipes can be read from the load memory with a user program, and other data areas such as archives can be written to the load memory.

The load memory is a Micro Memory Card

With a CPU 300, the load memory is located on the Micro Memory Card, with the result that a Micro Memory Card must always be plugged in to operate a CPU 300. The Micro Memory Card can also be used to transfer the user program without a programming device to the CPU or to perform a firmware update of the CPU.

Fig. 2.6 SIMATIC Micro Memory Card

Signal modules are the interface to the process

There is a wide range of signal modules available for the S7-300. Different digital input modules with 8, 16, 32, or 64 binary channels record signals with voltages of 24 V DC with sourcing or sinking output, 24 to 48 V DC or AC, 48 to 125 V DC, and 120 V or 230 V AC. Depending on the design, the digital output modules are available with 8, 16, 32, or 64 binary channels for 24 V output voltage with sourcing or sinking output, for 48 to 125 V DC, for 120 V or 230 V AC, and for relay outputs.

Voltage transmitters, current transmitters, thermocouples, or resistance thermometers can be connected to an analog input module with 2, 6, or 8 analog channels. The analog output module is available with 2, 4, or 8 analog channels for ±10 V output voltage or 0 to 20 mA output current.

Which properties are important in the selection of I/O modules can be seen in chapters 2.10 "Process Connection with Digital Modules" on page 47 and 2.11 "Process connection with analog modules" on page 48.

Function modules relieve the CPU

A function module (FM) is a signal-preprocessing, “intelligent” module that prepares and processes signals coming from the process independently from the CPU, and either returns them to the process or makes them available at the CPU's internal interface. Function modules handle functions that the CPU cannot usually execute quickly enough, such as counting pulses and positioning or controlling drives. The function modules available for S7-300 are listed in chapter 2.12 "FM modules relieve the CPU" on page 49.

Communication for S7-300

Each CPU 300 has an MPI to connect a programming device and for data exchange with another CPU with MPI. With the CPU 316-2 PN/DP and above, this interface is implemented as a combined MPI/DP interface. A CPU with a DP interface can be either DP master or DP slave on PROFIBUS DP. Additional information on PROFIBUS DP is available in chapter 6.15 "Distributed I/O with PROFIBUS DP" on page 242.

A CPU with PN interface can be both an IO controller and an IO device on PROFINET IO. Further information on PROFINET IO can be found in chapter 6.12 "Distributed I/O with PROFINET IO" on page 225.

Communication processors to connect to PROFIBUS, Industrial Ethernet, AS-Interface, and for point-to-point coupling are also available, as described in chapter 2.13 "Bus connection with communication modules" on page 50.

Operator control and monitoring with S7-300

The entire range of HMI devices and operator panels, as described in chapter 7 "Operator control and monitoring" on page 252, is available for the S7-300 controllers.

Configuration and programming with STEP 7

A CPU 300 can be configured and programmed with full functional scope (and even for older modules) with STEP 7 V5.x. The programming languages for the user program – ladder logic (LAD), function block diagram (FBD), and statement list (STL) – are integrated in STEP 7. The Continuous Function Chart (CFC) graphical program editor, the GRAPH sequence control, and the HiGraph state control are available as option packages for the structured control language (SCL) programming language.

Configuration and programming with STEP 7 Professional inside TIA Portal accounts for the modules which are currently being delivered. The programming languages for the user program – ladder logic (LAD), function block diagram (FBD), and statement list (STL), structured control language (SCL), as well as the GRAPH sequence control – are integrated in STEP 7 Professional V11.

2.5 Technological functions of a CPU 300C

A compact CPU 300C contains technological functions in the operating system that are used via integral system function blocks (SFBs) in the user program. The technological functions are permanently assigned to the digital and analog inputs/outputs on the CPU. Some of the assignments overlap, and therefore sometimes not all technological functions can be used together.

In the case of **open-loop positioning with analog output**, the actual value is acquired with asymmetric 24 V incremental encoders. The drive (power section) is controlled using an analog output in the range ±10 V (voltage signal) or ±20 mA (current signal). SFB 44 ANALOG is called in the user program. This SFB allows jog mode, reference point approach, relative and absolute incremental mode, set reference point, delete distance to go, and linear measurement.

In the case of **open-loop positioning with digital output**, the actual value is acquired with asymmetric 24 V incremental encoders. The drive (power section) is controlled using four digital outputs that switch the direction of travel and the velocity levels (rapid traverse or creep speed). SFB 46 DIGITAL is called in the user program. This SFB allows jog mode, reference point approach, relative and absolute incremental mode, set reference point, delete distance to go, and linear measurement.

Depending on the CPU used, **counting** is performed at a frequency of up to 60 kHz using a permanently assigned 24 V digital input. SFB 47 COUNT is called in the user program. This SFB allows continuous counting, single-shot counting, and periodic counting. The gate function can start, stop, or interrupt counting. A comparator can force a digital output or trigger a hardware interrupt when a preset count value has been reached.

In the case of a **frequency measurement**, the CPU counts the pulses within an adjustable time window. The time window can be set within the range 10 ms to 10,000 ms in steps of 1 ms. Depending on the CPU used, a frequency of up to 60 kHz can be measured. In the user program, you call SFB 48 FREQUENC. The gate function starts or stops the measurement. If the frequency reaches a lower or upper limit, a digital output can be forced or a process interrupt can be triggered.

Pulse width modulation converts a numeric value into a pulse train with the relevant pulse/pause ratio and outputs this at a digital output. The output fre-

Compact CPU 314C

The compact CPU 314C-2 PN/DP can be operated in standalone mode as station if the number of integrated I/O channels suffices. The CPU features onboard 24 x 24 V digital inputs, 16 x 24V/0.5 A digital outputs, 4 analog input channels, and 2 analog output channels.

The combined MPI/DP interface permits the connection to an MPI or the operation as DP master or DP slave on PROFIBUS DP. With the PN interface, the CPU can be both an IO Controller or IO Device on PROFINET IO.

Technological functions

The following technological functions are integrated in the CPU:

- Positioning via 1 channel with analog or digital output
- Counting with 4 counters (2 if positioning is used simultaneously) for frequency measuring with a maximum counting frequency of 60 kHz or pulse-width modulation with up to 2.5 kHz
- Controlling using a continuous controller (PID), step controller (PI), and pulse generator (PID with pulse output)

If the 16 available outputs, 4 can be used as igh-speed utputs for the technological functions.

Fig. 2.7 Performance features of a compact CPU 314C-2 PN/DP

quency is up to 2.5 kHz and the minimum pulse duration is 200 µs. Pulse width modulation is integrated into the user program with SFB 49 PULSE. The gate function starts or stops output of the pulse train.

Depending on the CPU used, the ASCII, 3964(R) and RK512 protocols are available for **point-to-point connection**. Up to 1024 bytes can be transferred at a data rate of 19.2 kbit/s (full duplex) or 38.4 kbit/s (half duplex). SFBs 60 and 65 form the interface to the user program.

Depending on the CPU used, the ASCII, 3964(R) and RK512 protocols are available for **point-to-point connection**. Up to 1024 bytes can be transferred at a data rate of 19.2 kbit/s (full duplex) or 38.4 kbit/s (half duplex). SFBs 60 and 65 form the interface to the user program.

The **closed-loop control functions** are software controllers implemented with SFB 41 CONT_C (PID controller with continuous manipulated variable output, suitable for two-step or three-step controls), SFB 42 CONT_S (PI controller with binary manipulated variable output without position feedback), and SFB 43 PULSEGEN (PID two-step or three-step controller with pulse width modulation).

2.6 SIMATIC S7-400 for demanding tasks

SIMATIC S7-400 is the highest-performance SIMATIC S7 controller. As an automation platform for system solutions in production and process engineering, the S7-400 is characterized primarily by its modularity and performance reserves. An S7-400 station consists of a central controller and – as required – of up to 21 expansion devices.

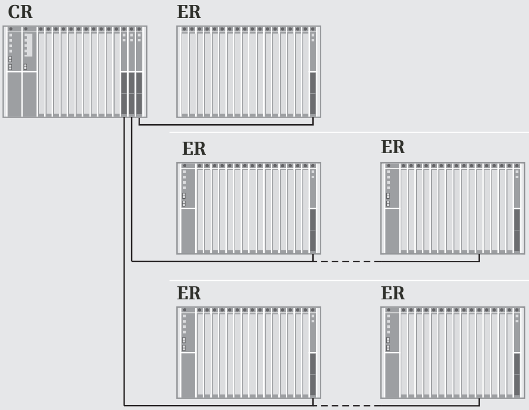
Setup of an S7-400 station

The figure shows a UR2 rack with 9 slots. The power supply module PS 407 occupies the first two slots, then a CPU 416-3 PN/DP and two signal modules (SM 422 with 32 x 24 V/0.5 A outputs and SM 421 with 32 x 24 V inputs).

A CPU 414-2 is also inserted in the same rack. It implements multi-processor operation together with the first CPU. The last two slots are occupied by an analog output module (8 x 13 bits) and a CP 443-1 Advanced communications processor for connecting the S7-400 station to Industrial Ethernet.

In multi-processor mode, the I/O modules can be freely assigned to a CPU.

Expansion options



Connection in the close-up range up to 1.5 m:
2 × ERs per connection in the CR, for "simple" signal modules
in the CR: IM 460-1
in the ER: IM 461-1

Connection in the close-up range up to 5 m:
2 × ERs per connection in the CR, for all I/O modules
in the CR: IM 460-0
in the ER: IM 461-0

Connection in the far range up to 102 m:
2 × ERs per connection in the CR, for all I/O modules
in the CR: IM 460-3
in the ER: IM 461-3

Max. 21 expansion racks can be connected to a central rack.

CR Central rack
ER Expansion rack
IM Interface module

Fig. 2.8 Setup and expansion options for an S7-400 station

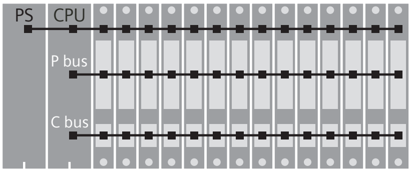
Central rack

The S7-400 mounting rack, an aluminum DIN rail of a fixed length with backplane bus and bus connectors, can be used as a central rack (CR), an expansion rack (ER), or as a combination of both (UR, universal rack).

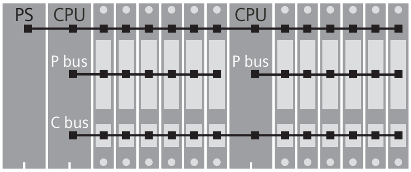
For the S7-400 there are central racks available with 18, 9, or 4 slots (UR1, UR2, or CR3) of fixed mounting widths. The power supply unit and the CPU also occupy module slots, possibly even 2 slots or more per module. Typically, the module arrangement begins on the left side with the power supply, followed by the CPU and the I/O modules, where the slots can be freely chosen, even with gaps. On the right-hand side of the rack are the interface modules to any expansion racks. The backplane bus, consisting of the parallel P bus for the I/O signals and the serial C-bus for the transmission of parameter and diagnostic data sets, connects the slots to each other.

Backplane bus in an S7-400 central rack

Universal rack UR1
On the UR1, all slots are connected to the C bus and I/O bus. The power supply of the rack is handled by a standard power supply unit or a redundant power supply.



Central rack CR2
The 18 slots on the CR2 are divided into segments of 10 and 8 slots. The I/O bus is divided and is used for one segment in each case.
One single power supply unit supplies the entire rack.



Universal rack UR2-H
The rack is divided into two segments with 9 slots each. The complete backplane bus is split so that a second power supply is required.

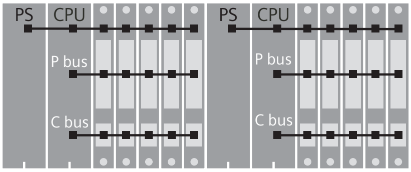


Fig. 2.9 Central rack for S7-400

The CR2 segmental rack allows you to use two CPUs on a common power supply. The CPUs exchange data over the communication bus, but each uses its own I/O bus to communicate with its I/O modules. The left segment provides 10 module slots, the right segment 8 module slots.

The *UR2-H segmental rack* consists of 2 segments with 9 module slots each. You can use it as a central rack or as an expansion rack in standard S7-400 stations or high-availability S7-400H stations. Each segment requires its own power supply; I/O bus and communication bus are separate.

Expansion racks

If there is insufficient space for the I/O modules in the central rack, or if you want to install modules at a distance, you supplement the station with one or more expansion units. Universal racks and expansion racks can be used as expansion units, with up to 21 on one central rack. The send and receive interfaces that are used in pairs transmit the signals over various distances:

Interface pair in central controller	IM 460-0	IM 460-1	IM 460-3
in expansion unit	IM 461-0	IM 461-1	IM 461-3
Max. distance	5 m	1.5 m	102 m
Max. number of connectable expansion units	8	2	8
Transmitting buses (P = peripheral bus, C = communications bus)	P + C	P	P + C
Supply voltage is transferred	No	Yes	No

The ER1 and ER2 expansion racks with 18 or 9 slots are intended for “simple” signal modules which do not trigger process alarms, do not require a 24 V DC supply over the P bus nor a backup supply, and are not equipped with a communication bus interface. The UR1 and UR2 racks are equipped with a communication bus if they are used either as central racks or expansion racks with the ID numbers 1 to 6.

Connection of SIMATIC S5 modules

The IM 463-2 interface module is used for connecting SIMATIC S5 expansion units (EG 183U, EG 185U, EG 186U, ER 701-2 and ER 701-3) to an S7-400 station. These expansion units can again be expanded in a centralized arrangement. An IM 314 interface module in the S5 expansion unit establishes the connection. In such an arrangement you can use any of the digital and analog modules listed for the specified expansion units. In a decentralized setup, you can connect up to 16 S5 expansion units to one S7- 400 station.

If you want to use a single S5 module in an S7-400 station, you have to make use of an adapter module.

Versatile application

A broad range of **standard CPU 400s** covers the middle and upper performance range in the manufacturing and process engineering. With a comprehensive range of modules and flexible networking capability, the requirement is met for optimum adaptation of the plant or process to be controlled.

The **4xxF failsafe CPUs** permit the construction of a failsafe automation system for plants with increased safety requirements. The failsafe I/O modules offer distributed connection to the failsafe PROFIsafe bus protocol via PROFIBUS DP or PROFINET IO.

Table 2.4 Quantity structure of standard S7-400 CPUs

CPU	Address ranges					Memory configuration
	Peripherals (bytes)	Process image (bytes) *)	Bit memory (bytes)	SIMATIC timers	SIMATIC counters	Load memory FEPROM / Load memory RAM integrated **) / work memory ***)
412-1 412-2 412-2 PN	4096	4096 / 128	4096	2048	2048	64 MB/512 KB/144 KB 64 MB/512 KB/256 KB 64 MB/512 KB/512 KB
414-2 414-3 414-3 PN/DP	8192	8192 / 256	8192	2048	2048	64 MB/512 KB/0.5 MB 64 MB/512 KB/1.4 MB 64 MB/512 KB/2 MB
416-2 416-3 416-3 PN/DP	16 384	16 384 / 512	16 384	2048	2048	64 MB/1 MB/2.8 MB 64 MB/1 MB/5.6 MB 64 MB/1 MB/8 MB
417-4	16 364	16 384 / 1024	16 384	2048	2048	64 MB/1 MB/15 MB

*) maximum / preset (both for inputs and outputs)

**) A RAM load memory on the memory card has the same size as a FEPROM load memory

***) Each for both program and data

The **fault-tolerant 4xxH CPUs** are used in control systems in which two CPUs – one “master” and the other “hot standby” – control a system in parallel. If the master system fails, the reserve CPU can smoothly take control of the system.

Enhanced performance with multiprocessing

You can convert the S7-400 station into a multiprocessing controller by inserting several CPUs (4 max.). Make sure you select CPUs that are specified for this type of operation which is also called “multicomputing.” As soon as you install more than one CPU, the programmable controller will automatically assume multiprocessing operation. All CPUs have the same operating mode. This means they start together and also all of them go to STOP if one of the CPUs fails. Each CPU executes its own user program independently of the other CPUs.

Each I/O module is assigned to one specific processor. This includes its address and alarms. All I/O modules are located in the same I/O bus segment. This means that you must give them different addresses even though they refer to different CPUs. The same rule applies to distributed I/O modules. Although each CPU can operate one or several DP master systems independently from the other processors, each centralized and distributed module must have its unique address (I/O bus addresses) in the overall system.

Operating modes of a CPU 400

The CPU 400 has the operating modes STOP, STARTUP, HOLD, and RUN. After switching on the power supply, the CPU is initially in STOP mode. The user program is not processed, but the CPU is still capable of communication, i.e. the user program can be loaded or the diagnostic buffer can be read, for example.

The CPU is switched into RUN mode with the mode switch or with a programming device in online mode. Here, the STARTUP mode is executed in which the modules are parameterized and a user start-up routine is executed. The main user program is executed in RUN mode.

In the operating modes STARTUP and RUN, test functions can be performed that lead to the HOLD mode. The execution of the user program is stopped at predetermined points in the program. The outputs to the machine to be controlled are switched off or set to a pre-defined value for safety reasons.

The user memory consists of a load memory and a work memory

The user program is located on the CPU in two areas: in the load memory and work memory. The load memory contains the entire user program including configuration data; it is integrated with the CPU and can be expanded with a memory card. The work memory is fast RAM that contains the execution-relevant program code and user data.

The programming device transfers the entire user program, including configuration data, to the load memory. The operating system interprets the configuration data, and parameterizes the CPU and – during startup – the I/O modules. The execution-relevant program code and user data are copied into the work memory.

A memory card expands the load memory

The size of the RAM load memory integrated on the CPU is sufficient for smaller user programs. For large user programs, the load memory card be expanded with a RAM memory card. If the user program has to be stored failsafe, a FEPROM memory card is used.

The FEPROM memory card can also be used to perform a firmware update of the CPU.

Fig. 2.10 SIMATIC RAM memory card

Voltage buffering of an S7-400

The RAM load and work memory is buffered with a backup battery in the power supply module. It is also possible to connect the CPU to an external backup voltage. Bit memories, SIMATIC timers and counters, and data blocks can be included in the buffered memory area.

Signal modules are the interface to the process

For the S7-400, various digital input modules are available with 16 or 32 binary channels. They can record the signals with voltages of 24 V DC, 24 to 60 V or 120 V DC or AC, and 120 V or 230 V AC. Depending on the model, the digital output modules are available with 16 or 32 binary channels for 24 V output voltage and for 120 V or 230 V AC.

Voltage transmitters, current transmitters, thermocouples, or resistance thermometers can be connected to an analog input module with 8 or 16 analog channels. The analog output module is available with 8 analog channels for ±10 V output voltage or 0 to 20 mA output current.

Which properties are important in the selection of I/O modules can be seen in chapters 2.10 "Process Connection with Digital Modules" on page 47 and 2.11 "Process connection with analog modules" on page 48.

Function modules relieve the CPU

A function module (FM) is a signal-preprocessing, “intelligent” module that prepares and processes signals coming from the process independently from the CPU, and either returns them to the process or makes them available at the CPU's internal interface. Function modules handle functions that the CPU cannot usually execute quickly enough, such as counting pulses and positioning or controlling drives. The function modules available for S7-400 are listed in chapter 2.12 "FM modules relieve the CPU" on page 49.

Communication for S7-400

Each CPU 400 has an MPI to connect a programming device and for data exchange with another CPU with MPI. A CPU with a DP interface can be either DP master or DP slave on PROFIBUS DP. On the CPU 414-3, 416-3, and 417-4, an interface module (or two on the CPU 414-4) can be plugged in to allow an additional DP interface with DP master or DP slave functionality. Additional information on PROFIBUS DP is available in chapter 6.15 "Distributed I/O with PROFIBUS DP" on page 242.

A CPU with PN interface can be both an IO controller and an IO device on PROFINET IO. Further information on PROFINET IO can be found in chapter 6.12 "Distributed I/O with PROFINET IO" on page 225.

In addition, communication processors to connect to PROFIBUS, Industrial Ethernet, and point-to-point coupling are also available, as listed in chapter 2.13 "Bus connection with communication modules" on page 50.

Operator control and monitoring with S7-400

The entire range of HMI devices and operator panels, as described in chapter 7 "Operator control and monitoring" on page 252, is available for the S7-400 controllers.

Configuration and programming with STEP 7

A CPU 400 can be configured and programmed with full functional scope (and even for older modules) with STEP 7 V5.x. The programming languages for the user program – ladder logic (LAD), function block diagram (FBD), and statement list (STL) – are integrated in STEP 7. The Continuous Function Chart (CFC) graphical program editor, the GRAPH sequence control, and the HiGraph state control are available as option packages for the structured control language (SCL) programming language.

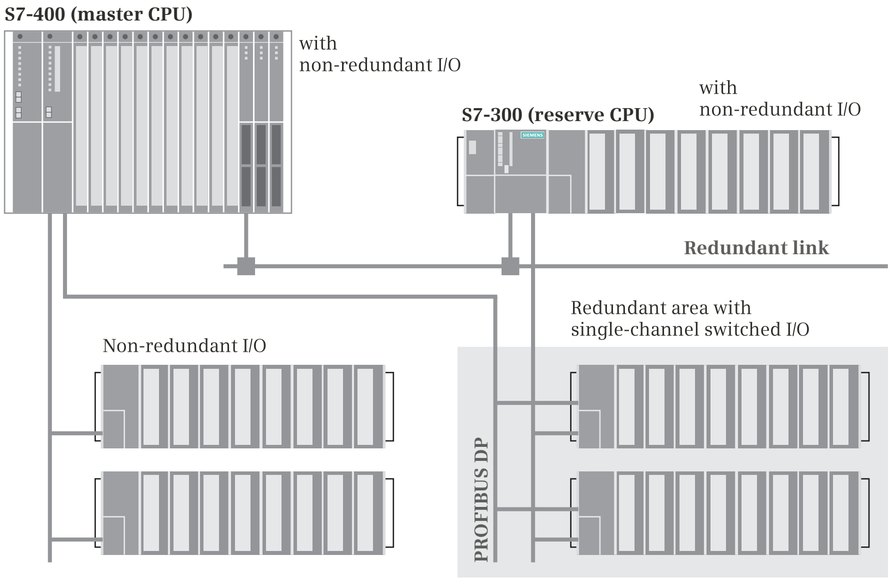
Configuration and programming with STEP 7 Professional inside TIA Portal accounts for the modules which are currently being delivered. The programming languages for the user program – ladder logic (LAD), function block diagram (FBD), and statement list (STL), structured control language (SCL), as well as the GRAPH sequence control – are integrated in STEP 7 Professional V11.

2.7 High Availability with SIMATIC

For slow processes: software redundancy with standard components

The standard SIMATIC S7-300/400 components allow you to build a redundant system based on software. In such a redundant setup, a standby station assumes control of the process if the master station breaks down. High availability through software redundancy is suitable for slow processes as the switchover to the standby station could take up to several seconds depending on the configuration of the

Software redundancy



The software redundancy operates with standard components. A CPU 300 or CPU 400 can be used as master and reserve CPU. The redundant link uses MPI, PROFIBUS, or Industrial Ethernet for data transmission. In addition to the redundantly designed I/O, non-redundant centralized or distributed I/O can also be available.

Fig. 2.11 Example of a setup with software redundancy

PLCs. During this period, all process signals are “frozen”. The standby station then continues processing with the most recent valid data from the master station.

Both participating CPUs exchange the current data over a (random) subnet. Existing communication links can also be used. Single-channel redundancy of the input/output modules is implemented with distributed I/O (ET 200M with IM 153-2 interface module for redundant PROFIBUS DP). Each CPU can also independently control non-redundant I/O, both in central and distributed configurations.

The “Software Redundancy” option package is available for configuration.

High availability using hot standby: SIMATIC S7-400H

SIMATIC S7-400H is a highly available programmable logic controller using a redundant configuration with two CPUs 4xxH or 4xxF/H, each equipped with a synchronization module for data synchronization via fiber-optic cable. Both devices work in “hot standby” mode with automatic, bumpless switchover to the standby unit, which takes over full control of the user program if the primary unit fails.

The high-availability system can comprise either two separate central racks UR1 or UR2 or one split rack UR2-H. Alongside the always redundant power supplies and CPUs, you can connect peripherals with normal or high availability (Fig. 2.12).

A high-availability S7 connection over an Industrial Ethernet or PROFIBUS subnet can consist of up to four subordinate connections depending on the configuration. Of these, two are always established (active) in order to retain the communication in the event of a fault.

The required software “S7 H Systems” is included in STEP 7 V5.3 or higher. The library “Redundant IO (V1)” supports you when scanning and controlling the redundant I/O.

2.8 Safety Integrated with SIMATIC S7

For increased safety requirements: safety-related SIMATIC

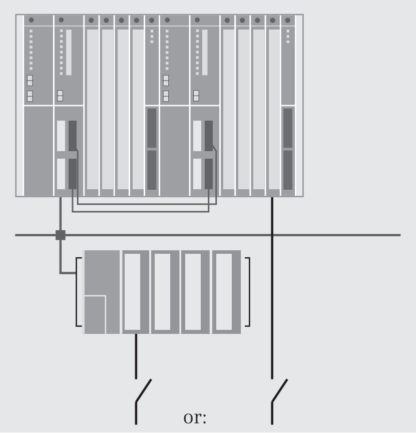
Failsafe automation systems control processes and machines with the aim of reducing the hazards to personnel and the environment as far as possible without restricting production more than is absolutely necessary. The safety-related SIMATIC systems are used in cases where the safe state can be achieved by immediate shutdown. They comply with the following safety requirements: Safety Integrity Level SIL1 to SIL3 in accordance with IEC 61058 and Category 1 to Category 4 in accordance with EN 954-1.

The enhanced safety functions are essentially achieved by means of a safety-related user program in an appropriately designed CPU (F-CPU), by means of failsafe I/O modules (F modules), and by the higher-level PROFIsafe safety profile with the PROFIBUS DP or PROFINET IO “standard” bus systems.

SIMATIC S7-400H

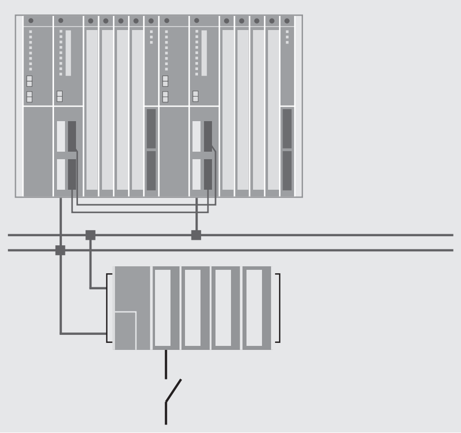
Single-channel single-sided configuration

The I/O modules are present once and connected to one of the two subsystems, either centrally or distributed. The signal states are available in both subsystems via the redundant link. The user program is the same in both subsystems. If a subsystem fails, the I/O connected to it is also no longer available.



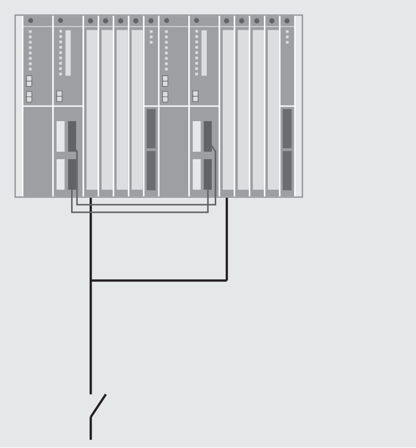
Single-channel switched configuration

The I/O modules are present once as ET 200M distributed I/O with active backplane bus. The I/O must be set up symmetrically; each subsystem has access to the I/O via the IM 153-3 interface module with redundant DP interface. The H-system currently uses only one of the interfaces; the signal states are available in both subsystems via the redundant link. The user program is the same in both subsystems.



Redundant single-sided configuration

The I/O modules are plugged in pairs into both subsystems, centrally in a rack. The redundancy function is implemented in the user program.



Redundant switched configuration

The I/O modules are plugged in pairs into two ET 200M with active backplane bus, which with the redundant IM 153 interface module is connected to both subsystems. The redundancy function is implemented in the user program.

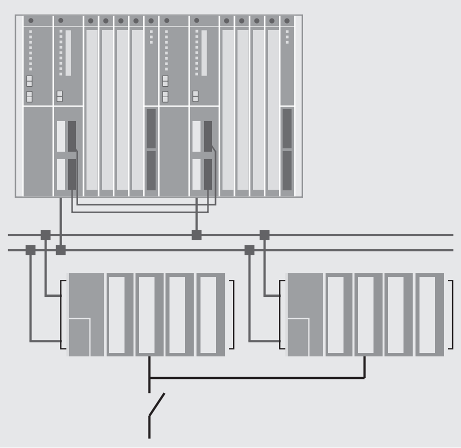


Fig. 2.12 Process I/O designs in a SIMATIC S7-400H system

Failsafe I/O

The F modules required for safety operation are available in different versions. In the S7-300 design, F modules are provided for centralized use in a failsafe S7-300 station or for distributed use in an ET 200M station. Failsafe power modules and electronic modules are used centrally and in distributed configurations in an ET 200S station. Using S7 Distributed Safety, you can also connect failsafe PROFIBUS DP standard slaves.

F modules can also be used in standard operation with increased diagnostics requirements. You can have standard operation and safety-related operation combined in the same automation system, and on the S7-400FH even with fault tolerance.

Focusing on machine control application: S7 Distributed Safety

Fig. 2.13 CPU 315F-2 PN/DP

S7 Distributed Safety comprises a failsafe CPU and failsafe modules connected to the F CPU via PROFIBUS DP or PROFINET IO. Here, the PROFIsafe bus profile guarantees failsafe transmission. Currently available as F CPUs are the CPU 3xxF-2 for the S7-300, the CPU 416F-2 for the S7-400, and the IM151-7 F CPU basic module for the ET 200S.

Together with the F modules, the following configurations are available: Centralized configuration with failsafe modules on the S7-300 with CPU 3xxF-2 and on the ET 200S with IM 151-7 F-CPU, distributed configuration with ET 200M, ET 200S, ET 200eco or ET 200pro stations with failsafe modules as well as failsafe DP standard slaves and failsafe IO standard devices connected to a centralized S7-300/400 station with F-CPU.

The safety-related program section is programmed with the languages F LAD and F FBD with a limited operation set and fewer data types than the basic languages. Both languages are included in the “S7 Distributed Safety” option package that you require for configuring and programming the safety mode. The option package also contains a block library for the safety program with F blocks and templates.

Focusing on the process industry: S7 F/FH Systems

S7 F/FH Systems is based on the S7-400 automation system in the version with normal availability and in the fault-tolerant version. The F modules required for safety operation are connected to the S7-400 station via PROFIBUS DP with the PROFIsafe bus profile. The following are used: ET 200M stations with F modules of

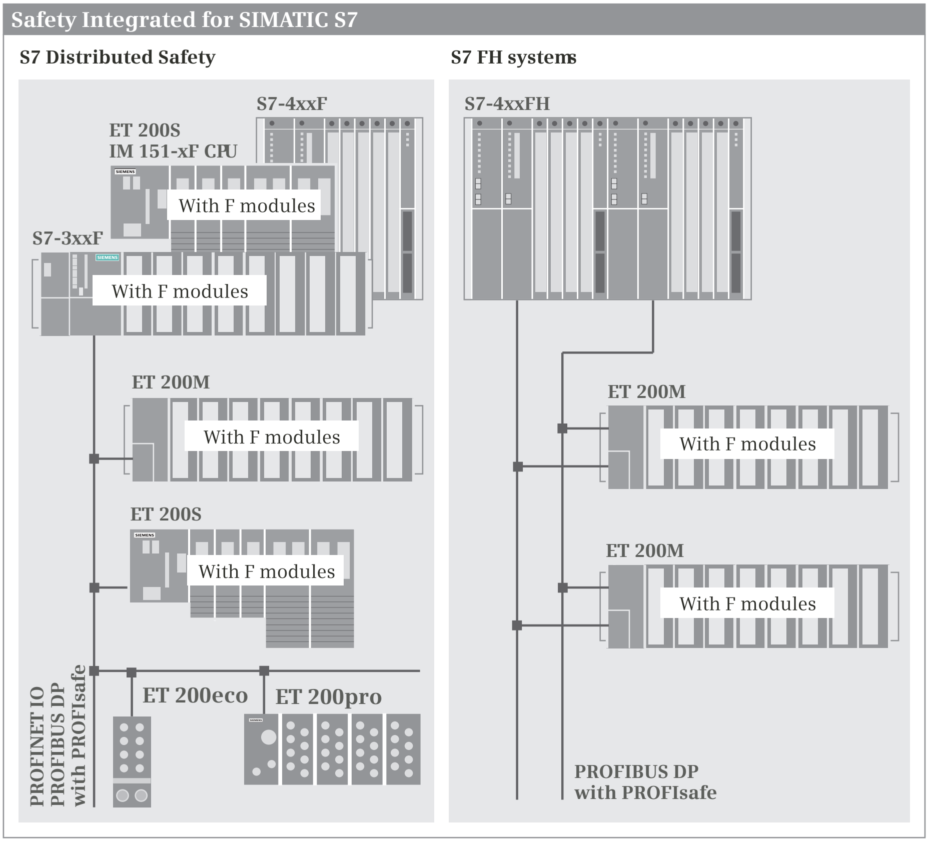


Fig. 2.14 Design versions for S7 Distributed Safety and S7 FH Systems

S7-300 design, ET 200S stations with safety-related power and electronics modules, and ET 200eco F modules.

In the fault tolerant version S7-400FH, ET 200M stations are controlled via a redundant PROFIBUS. If a detected fault results in STOP of the master CPU on the S7-400FH, the system performs a reaction-free switchover to the hot standby CPU.

An F-Runtime license must be loaded onto each CPU to operate the S7-400F/FH. STEP 7 V5.1 or higher is required for configuration. The option package “S7 F Systems” is required for configuring the F modules and programming the failsafe program section. Also required are the option package “CFC” from V5.0 SP3, the option package “S7-SCL” from V5.0, and for the fault-tolerant functions the option package “S7 H Systems” from V5.1.

With CFC (Continuous Function Chart), you interconnect special function blocks from the supplied F library. As well as functions for programming safety functions, the blocks also contain functions for detecting and responding to faults. In the event of failures and faults, the F system can thus be maintained in a safe state or changed to a safe state. If a fault is detected in the safety program, the F section of the plant is switched off, whereas the remaining part can continue to operate.

2.9 Use Under Difficult Conditions: SIPLUS

“SIPLUS extreme” is the product range with hardened components for use in harsh environments. Standard devices that have been specially converted and refined for the respective application are used as the basis for the SIPLUS components.

Harsh environmental conditions can include:

- Ambient temperature range from −40/−25 °C to +60/+70 °C
- Condensation, increased humidity, increased degree of protection (dust, water)
- Extreme loading by media (conformal coating), e.g. toxic atmospheres
- Increased mechanical load, increased electrical immunity
- Voltage ranges deviating from the standard
- Ambient conditions on rail vehicles

SIPLUS modules are manufactured on request for the environmental conditions specified. Please therefore observe the technical specifications of the respective module.

Fig. 2.15
SIPLUS CPU 1214C DC/DC/DC,
6AG1 214-1AE30-0XB0
based on 6ES7 214-1AE30-0XB0

SIPLUS modules are available for SIMATIC programmable controllers in all designs: S7-200, S7-1200, S7-300, S7-400, ET 200, and for other Siemens device families. SIPLUS modules include power supplies; CPUs, signal, function and communication modules, and numerous special modules.

Configuring of SIPLUS modules

The functionality of a SIPLUS module is the same as that of the corresponding standard module; the Order No. (MLFB, machine-readable product code) begins with 6AG1. SIPLUS modules are not included with their order numbers in the hardware catalog of STEP 7.

Since the SIPLUS modules have the same functions as existing modules, you can use the corresponding equivalent type (the standard module) when configuring.

This equivalent type (“based on 6ES7 ... ”) can be found on the device's nameplate, in the SIPLUS data sheets, and on the Internet in the Siemens Industry Mall.

2.10 Process Connection with Digital Modules

Digital modules are signal converters for binary process signals. The CPU of the SIMATIC station receives information about the operating modes of the process through digital input modules and intervenes in the process through digital output modules. The digital signals between backplane bus and process are isolated by means of optocouplers.

Digital modules are available with one, two, four, or eight bytes corresponding to 8, 16, 32 or 64 signals. Digital modules are preferably addressed in the process image so that the signal states can be processed bit by bit. User-friendly digital modules supply diagnostic information about the state of the module.

Fig. 2.16
I/O modules in different designs

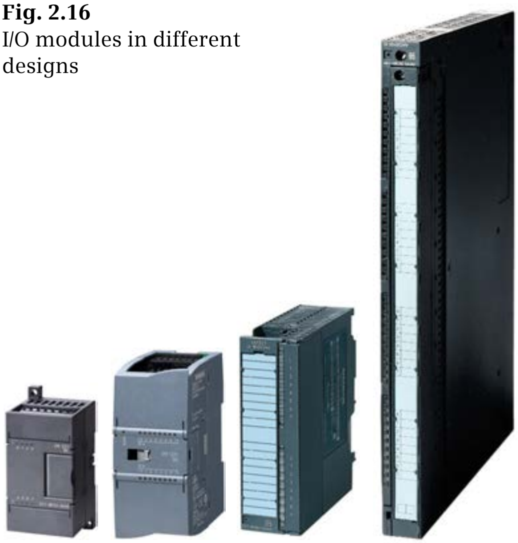


Fig. 2.16 shows digital input modules with the various designs of the SIMATIC automation families in approximately the same scale. From left to right, these are the modules

- ▷ EM 221
  (S7-200: DI 8 × 24 V DC)
- ▷ SM 1221
  (S7-1200: DI 16 × 24 V DC)
- ▷ SM 321
  (S7-300: DI 16 × 24 V DC)
- ▷ SM 421
  (S7-400: DI 32 × 24 V DC).

The same designs (appearance, proportions) apply for the entire module range of signal modules (SM), function modules (FM), and communication modules (CP).

Digital input modules

Digital input modules convert the external signal voltage, usually 24 V DC or 120 V/230 V AC, into the internal signal level. The connected sensor must be within the permissible voltage range and provide the required input current with the signal status “1” so that the module can switch reliably. In addition, the input signals are filtered, i.e. interference on the lines is suppressed and glitches eliminated. The filtering results in the input signals being delayed. For digital input modules with hardware interrupt triggering, you can reduce the input delay. A compromise must be found here between interference resistance (long delay) and fast signal recording (short delay).

Digital output modules

To be able to intervene in the process, the CPU requires signal converters which convert the internal signal states to the voltage and current levels used in the plant or process. The digital output modules are equipped with a data memory which stores the received data and then passes this information on to an amplifier. The amplifier provides the required switching capacity. For d.c. voltage amplifiers, short-circuit protection is implemented electronically, whereas a.c. voltage amplifiers are protected by means of fine-wire fuses.

When you select a digital output module, you consider its switching capacity, its total permissible load and the residual current. The residual current at signal state “0” must not fall below the permitted limit, otherwise the activated device will not respond to a stop signal.

In the operating modes STOP and HALT, and also during the start-up period leading to program execution, an output disable signal (OD signal) disables all digital output modules. In this state, they either do not supply any voltage, supply a defined substitute value, or maintain the value set last.

Explosion-proof digital modules

SIMATIC explosion-proof digital modules are “associated electrical equipment” with “intrinsically safe” protection ([EEx ib] IIC acc. to DIN EN 50020). They are approved for the connection of intrinsically safe electrical devices located in zone 1 or 2. The modules are installed outside hazardous areas. The load voltage of 24V DC is routed through the LK393 cable duct. Explosion-proof modules are used in S7-300 stations or as distributed I/O in ET 200M stations. You use STEP 7 to configure these modules.

2.11 Process connection with analog modules

Analog modules are signal converters for analog process signals. Analog input modules convert the analog signals coming from the process or plant to digital signals which can be processed by the CPU of the SIMATIC station. Analog output modules convert the digital signals from the SIMATIC station to analog signals for the process, such as setpoint values for actuators. Each analog quantity, such as a measuring or setpoint value, occupies a “channel” on the module. Analog modules are available with 4, 8 or 16 channels which correspond to 8, 16 or 32 bytes. A digitized analog value is internally represented as a 16-bit integer (data type INT). Advanced analog modules supply diagnostic information about the state of the module or limit value information.

Analog modules are preferably addressed outside of the process image table, especially if they are read from or written to directly. This is the case, for example, in closed-loop control circuits whose processing cycle is independent of the main program.

Analog input modules

Analog input modules use an integration method to convert the analog quantities received from the process (voltage, current, resistance) to digital values. Depending on the interference voltage suppression (for 400/60/50/10 Hz), the conversion takes 2.5/20/20/100 milliseconds. The resolution is correspondingly high (9/12/12/15 bits + sign). You set the basic current/voltage ranges mechanically, by means of coding switches. Use the STEP 7 tool HW Config to set the range to more precise values.

Analog output modules

Analog output modules convert internal digital values to analog voltages and currents required by the process. Different modules for different voltage and current ranges are available. Internal and external signals are electrically isolated. The digital values received from the CPU are stored in a data memory on the module. From here they are passed on to the digital-to-analog converter which converts the signals to analog quantities within 0.8/1.5 milliseconds. They are then transmitted to the process.

Explosion-proof analog modules

SIMATIC explosion-proof analog modules are “associated electrical equipment” with “intrinsically safe” protection ([EEx ib] IIC acc. to DIN EN 50020). They are approved for the connection of intrinsically safe electrical devices located in zone 1 or 2. The modules are installed outside hazardous areas. The load voltage of 24V DC is supplied by means of the LK393 cable duct. Explosion-proof modules are used in S7-300 stations or as distributed I/O in ET 200M stations. You use STEP 7 to configure these modules.

SIMATIC S7-HART analog modules are also of the “intrinsically safe” type and can be used in the ET 200M distributed I/O station. The modules take over the task of a HART master allowing the connection of two HART field devices acting as slaves. Data communication is based on the HART protocol, version 5.4.

2.12 FM modules relieve the CPU

Function modules (FM) are signal-preprocessing, “intelligent” modules that prepare and process signals coming from the process independent of the CPU, and either return them to the process or make them available at the CPU's internal interface. They handle functions that the CPU cannot usually execute quickly enough, such as counting pulses, positioning, or controlling drives (Table 2.5).

Table 2.5 Selection of FM modules with functions and properties (S7-300/400)

FM 350, FM 450 Fast count up/ down	FM 350-1: 1 channel FM 450-1: 2 channels Count up to 500 kHz with ± 31 bits. FM 350-2: 8 channels Count up to 20 kHz with ± 31 bits, also suitable for NAMUR encoder.	▷ Connection of 5 V/24 V incremental-encoders ▷ Gate control, also directly with digital-input on the FM ▷ At zero point or when a comparison value is reached: direct control of digital output or generation of hardware interrupt
FM 351, FM 451 Positioning for rapid and creep speed drives	Adjustment of 2 or 3 independent axes with 4 digital outputs per axis for controlling contactors or frequency converters. Position detection can be carried out either incrementally or synchronous-serially.	▷ Set up (traversing in inching mode) ▷ Incremental feed mode, absolute and relative ▷ Reference point approach ▷ Zero offset ▷ Set reference point ▷ Delete distance-to-go
FM 352, FM 452 Electronic cam control	32 cam tracks with 32/64/128 cams and parameterizable cam characteristics (e.g. position-based or time-based cams) control 13 or 16 digital outputs. Position detection can be carried out either incrementally or synchronous-serially.	▷ Length measurement ▷ Set reference point ▷ On-the-fly actual value setting ▷ Zero offset ▷ Change cam edges ▷ Simulation operation
FM 353, FM 453 Positioning for stepper motors **FM 354, FM 453** Positioning for servomotors	Positioning of one (FM 353/354) or up to three (FM 453) drives, can be used for linear and rotary axes. The modules handle both simple point-to-point positioning as well as complex traversing profiles.	▷ Set up (traversing in inching mode) ▷ Incremental mode ▷ Automatic following block/single block ▷ Length measurement ▷ Start/stop via direct digital inputs ▷ Jerk limit ▷ On-the-fly actual value setting
FM 455 Control unit for pressure, temperature and flow rate control	PID control algorithm or self-optimizing temperature control algorithm with 16 channels FM 455C: continuous controller FM 455S: step or pulse controller On failure or STOP of the CPU, the FM 455 enters backup mode.	Pre-configured controller structures for e.g.: ▷ Fixed setpoint control ▷ Cascade control ▷ Ratio control ▷ 3-component control Sampling time at a resolution of ▷ 12 bits: 20 ms at 50 Hz ▷ 14 bits: 100 ms at 50 Hz

The FM modules are generally parameterized with HW Config. Just like the loadable standard blocks for the user program, the required configuration filters are part of the scope of supply and are included automatically when STEP 7 is installed. Diagnostic screens for testing and commissioning are available.

2.13 Bus connection with communication modules

Communication processors (CPs), for S7-1200: communication modules (CM), or communication boards (CB) relieve the CPU of communication tasks. They establish the physical connection to the network, take over establishment of the connection and data transport via the network, and provide the required communication services for the CPU and the user program (Table 2.6).

Table 2.6 Selection of communications processors with functions and properties

CM 1241, CB 1241, CP 340, CP 341, CP 440, CP 441 Point-to-point connection	Serial point-to-point connection with one interface (CP 441-2 with two interfaces). The interfaces of CP 341 and CP 441-2 are designed for non-resident customer-specific drivers.	Physical transmission properties: ▷ RS 232C (V.24) (not with CP 440, CB 1241) ▷ 20 mA (TTY) (not with CP 440, CM/CB 1241) ▷ RS 422/RS 485 (X.27) Implemented protocols: ▷ ASCII driver ▷ 3964(R) (not with CM/CB 1241) ▷ RK 512 (with CP 341 and CP 441-2) ▷ Printer driver (CP 341 and CP 441-2)
CM 1243-2, CP 343-2P AS-i master	AS-Interface master for connection of up to 62 AS-Interface slaves.	With configuration support of the AS-Interface network
CM 1242-5, CM 1243-5, CP 342-5, CP 443-5 Ext. PROFIBUS DP	PROFIBUS DP master (CM 1243-5, CP 342-5, CP 443-5 Ext.), PROFIBUS DP slave (CM 1242-5, CP 342-5).	Communication services: ▷ PROFIBUS DP With CP 342-5 and CP 443-5 Ext. also: ▷ S7 communication ▷ S5-compatible communication
CP 343-5, CP 443-5 Basic PROFIBUS FMS	Connection to PROFIBUS FMS.	Communication services: ▷ PROFIBUS FMS ▷ S7 communication ▷ S5-compatible communication
CP 343-1 Lean Industrial Ethernet	Connection to Industrial Ethernet with integrated 2-port switch. Can be used as PROFINET IO device.	Communication services/connections: ▷ S7 communication ▷ S5-compatible communication ▷ IE communication (TCP/IP and UDP)
CP 343-1, CP 443-1 Industrial Ethernet	Connection to Industrial Ethernet with integrated 2-port switch. Can be used as PROFINET IO controller, CP 343-1 also as PROFINET IO device.	Communication services/connections: ▷ S7 communication ▷ S5-compatible communication ▷ IE communication (TCP/IP and UDP) ▷ PROFINET CBA
CP 343-1 Adv. CP 443-1 Adv. Industrial Ethernet	Connection to Industrial Ethernet. Can be used as PROFINET IO controller with real-time capabilities, CP 343-1 Adv. also as PROFINET IO device. With one Gigabit interface.	Communication services/connections: ▷ S7 communication ▷ IE communication (TCP/IP and UDP) ▷ PROFINET CBA ▷ IT communication (HTTP communication, e-mail client, FTP communication, access to data blocks via FTP server)

The communication modules are parameterized with HW Config. Just like the loadable standard blocks for the user program, the required configuration filters are part of the scope of supply and are included automatically when STEP 7 is installed. Diagnostic screens for testing and commissioning are available.

The communication modules for connecting to PROFIBUS and PROFINET or Industrial Ethernet supplement the corresponding interfaces on the CPU – in part with additional functions. With these communication modules, additional PROFIBUS DP master systems or PROFINET IO systems can be created in the PLC station.

2.14 SIMATIC PC-based Automation

The PC-based automation gives you the functionality of a PLC on a personal computer (PC). You can then connect this with PC applications, e.g. for data processing, communication, and visualization. Several industry-standard PCs support you in the use of PC-based Automation.

WinAC – the software PLC for the PC

SIMATIC WinAC (Windows Automation Center) combines the functions of open-loop control, technology, data processing, visualization, and communication on one personal computer (PC) and is the first choice if you have to handle PC applications in addition to classic PLC tasks. WinAC controllers are configured and programmed exactly like S7 controllers using the standard STEP 7 software.

SIMATIC WinAC RTX is the SIMATIC software controller for PC-based automation solutions and permits real-time capable, deterministic control functions to be carried out on the PC. In the failsafe WinAC RTX F version, the software controller is sufficient in addition to the safety requirements up to SIL 3 (IEC 61508). WinAC RTX 2010 is executable under the operating systems Windows XP SP2 and SP3 as well as Windows 7 (32-bit).

A real-time response means that the system processes external events within a defined period. If it reacts in a predictable manner, it is referred to as deterministic. With WinAC RTX 2008, the control program is executed in a fixed cycle, and can also interrupt the Windows applications being carried out in parallel. It is possible to define the priority of the control program compared to the Windows applications.

The process I/O is connected via PROFIBUS or PROFINET. Suitable PCI adapters are available for this purpose. Here, WinAC RTX is the DP master or the IO controller. Isochronous mode is supported in both cases, that is the synchronous reading in, processing, and output of I/O signals in a fixed time pattern (constant bus cycle time).

SIMATIC WinAC ODK (open development kit) permits the control program to flexibly use all resources of the PC. WinAC ODK V4.2 provides three interfaces for this purpose: Custom Code Extension Interface (CCX) for calling own programs in C/C++ from the control program of WinAC, Shared Memory Extension Interface (SMX) for fast data exchange between WinAC and Windows applications, and Controller Management Interface (CMI) for integrating the WinAC panel functionality into a Windows application.

Industrial PC – the hardware platform for PC-based automation

The industrial PCs are available as Rack PCs in a 19” enclosure, Box PCs in a compact, space-saving enclosure, and Panel PCs with a display for use in control cabinets, consoles, and switchboards directly on-site (see chapter 7.10 "SIMATIC Panel PC" on page 264).

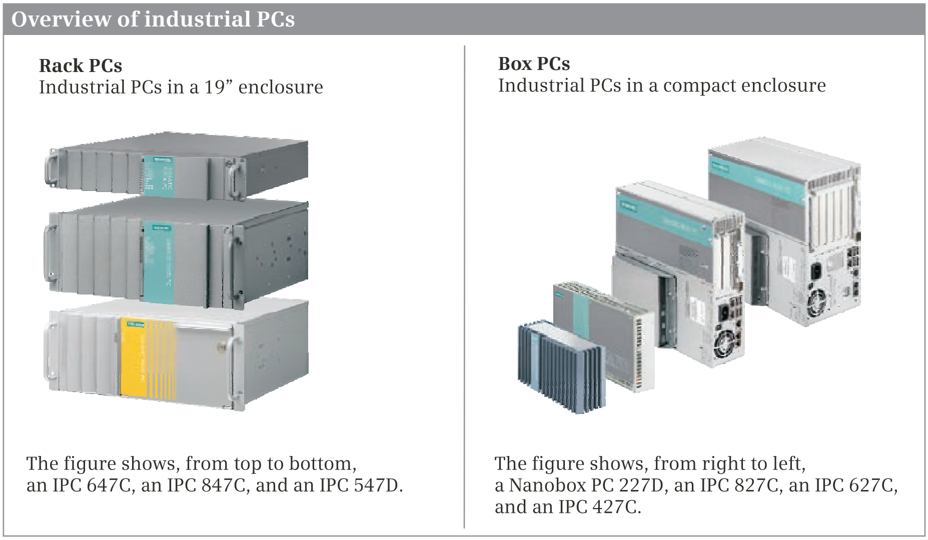
Rack PCs are flexible, fault-tolerant industrial PC systems for high-performance applications in a space-optimized 19” enclosure with depth of only 500 mm for installation in control cabinets. The front side or front door can be locked. Rack PCs are designed for 24-hour continuous operation. Dust protection thanks to the overpressure ventilation concept with fan on the front and dust filter.

Three device classes are available for various requirements:

- SIMATIC IPC 547 – performance at an attractive price
- SIMATIC IPC 647 – high compactness with high industrial functionality
- SIMATIC IPC 847 – high expandability with high industrial functionality

The processors are available in several versions (see table 2.7), as is the mass storage: Serial ATA 3.5” hard drive with NCQ technology or Serial ATA 2.5” solid-state drive (SSD) with SLC technology, each for internal installation or in a swap frame on the front (“hot swap”: replacement of the hard drive possible during operation). For increased data security through mirror disks, the Rack PCs can be equipped with a RAID1 system, and the IPC 547D can also be equipped with a RAID5 system (block striping with distributed parity). The optical drive is either a DVD-ROM or a DVD ±/-R/RW.

Box PCs are particularly robust industrial PC systems for high-performance applications in a space-optimized enclosure. Box PCs are designed for 24-hour continuous operation. Mass storage with CompactFlash or Solid-State Drive (SSD) ensure robust data storage.



Overview of industrial PCs

Rack PCs
Industrial PCs in a 19” enclosure

The figure shows, from top to bottom, an IPC 647C, an IPC 847C, and an IPC 547D.

Box PCs
Industrial PCs in a compact enclosure

The figure shows, from right to left, a Nanobox PC 227D, an IPC 827C, an IPC 627C, and an IPC 427C.

Fig. 2.17 Overview of Rack PCs and Box PCs

Table 2.7 Selected specifications for Rack PCs (basic configuration)

Type	IPC 547D	IPC 647C	IPC 847C
Processors (versions)	Intel Core i7-2600 (4C/8T, 3.40 GHz, 8 MB Last Level Cache) Intel Core i5-2400 (4C/4T, 3.10 GHz, 6 MB Last Level Cache) Intel Pentium Dual Core G850 (2C/2T, 2.90 GHz, 3 MB Last Level Cache)	Intel Core i7-610E (2C/4T, 2.53 GHz, 4 MB cache) Intel Core i5-520E (2C/4T, 2.4 GHz, 3 MB cache) Intel Core i3-330E (2C/4T, 2.13 GHz, 3 MB cache)	Intel Core i7-610E (2C/4T, 2.53 GHz, 4 MB cache) Intel Core i5-520E (2C/4T, 2.4 GHz, 3 MB cache) Intel Core i3-330E (2C/4T, 2.13 GHz, 3 MB cache)
Main memory configuration	1 GB to 32 GB, DDR3 1333 SDRAM	From 1 GB to 8 GB, DDR3 1066 SDRAM	From 1 GB to 8 GB, DDR3 1066 SDRAM
Graphic	Onboard, Intel HD 2000 integrated in processor, up to 2560 x 1600 pixels, 60 Hz, 32-bit colors	Onboard, Intel GMA HD integrated in processor, up to 2048 x 1536 pixels, 60 Hz, 16-bit colors	Onboard, Intel GMA HD integrated in processor, up to 2048 x 1536 pixels, 60 Hz, 16-bit colors
Hard drives (variants) Internal installation	1 x 500 GB 1 x 1 TB RAID1, 2 x 1 TB 1 x 50 GB SSD (SLC)	1 x 32 GB SSD in single level cell (SLC) architecture	1 x 250 GB HDD 1 x 32 GB SSD in single level cell (SLC) architecture
Installation at the front in the swap frame	1 x 500 GB 2 x 500 GB RAID1, 2 x 1 TB RAID5, 3 x 1 TB 1 x 50 GB SSD (SLC) RAID1, 2 x 1 TB + 1 x 50 GB SSD (SLC)	1 x 250 GB 1 x or 2 x 500 GB RAID1, 2 x 500 GB	1 x 250 GB 1 x or 2 x 500 GB RAID1, 2 x 500 GB RAID5, 3 x 500 GB 1x 32 GB SSD (SLC)

Slots for drives
On the front
	1 x 3.5";
	3 x 5.25" or 1 x 3.5";
	1 x 5.25";
	3 x HDD swap frames (low profile)
		2 x HDD swap frames (low profile);
		1 x optical drive (slimline) or 1 x CF drive
			1 x 3.5";
			3 x 5.25" or 1 x 3.5";
			1 x 5.25";
			3 x HDD swap frames (low profile)

internal	2 x 3.5"	2 x 3.5" *)	2 x 3.5" *)
Ports	2 x Gbit LAN (RJ45) 11 x USB 2.0 (8 x rear, 4 x front, 1 x internal) 2 x PS/2 1 x DisplayPort, COM1, DVI-I each	2 x Gbit LAN (RJ45) 9 x USB 2.0 (4 x rear, 2 x front, 1 x internal) 2 x PS/2 1 x COM1, COM2, LPT1, DVI-I each	2 x Gbit LAN (RJ45) 9 x USB 2.0 (4 x rear, 2 x front, 1 x internal) 2 x PS/2 1 x COM1, COM2, LPT1, DVI-I each
Free slots (all long)	4 x PCI 1 x PCI Express x16 1 x PCI Express x16 (4 lanes) 1 x PCI Express x8 (1 lane)	1 x PCI 1 x PCI Express x16 1 x PCI Express x8 (4 lanes)	7 x PCI 1 x PCI Express x16 3 x PCI Express x4

*) in optional, shock and vibration-damped disk-drive support as an alternative to swap frames

Table 2.8 Selected specifications for Box PCs (basic configuration)

Type	IPC 227D (Nanobox)	IPC 427C (Microbox)	IPC 627C (Box PC) IPC 827C (Box PC)
Processors (optional versions)	Intel Atom E660 1.3 GHz, 2 GB RAM	Intel Core2 Duo 1.2 GHz, 800 MHz FSB, 3 MB SLC	Intel Core i7-610E processor (2C/4T, 2.53 GHz, 4 MB cache)
	Intel Atom E640 1.0 GHz, 1 GB RAM	Intel Core2 Solo 1.2 GHz, 800 MHz FSB, 3 MB SLC	Intel Core i3-330E processor (2C/4T, 2.13 GHz, 3 MB cache)
	Intel Atom E620 600 MHz, 512 MB RAM	Intel Celeron M 1.2 GHz, 800 MHz FSB, SLC 1 MB	Intel Celeron P4505 processor (2C/2T, 1.86 GHz, 2 MB Cache)
Main memory configuration	(see above under Processors)	1 GB to 4 GB, DDR3 1066 SDRAM	1 GB to 8 GB, DDR3 1066, ECC memory 2/4/8 GB, DDR3 1066
Graphic	Onboard: DVI (digital, resolution up to 1920 x 1200 pixels)	Onboard, on AGP bus: VGA (analog, resolution up to 1920 x 1200 pixels/true color/60 to 120 Hz) and DVI (digital, resolution up to 1920 x 1200 pixels/true color)	Onboard (resolution 1600 x 1200 Pixel, 85 Hz, 32-bit colors)
Mass storage (optional versions)	Serial ATA hard disk, 2.5”, 250 MB	Serial ATA hard disk, 2.5”, >250 GB	SATA hard disks: 250/500 GB, 3.5”; RAID1 2 x 250 GB, 2.5”
	Solid-state drive (SSD) 50 GB in SLC technology	Solid-state drive (SSD) 32 GB in SLC technology	Solid-state drive (SSD) 32 GB in SLC technology
	CompactFlash drive (replaceable, accessible): 2/4/8 GB	CompactFlash drive (replaceable, accessible): 256 MB, 2/4/8 GB	Additionally for IP 627C: optional second CompactFlash drive (internal)
		CompactFlash drive (internal, not accessible): 256 MB, 2/4/8 GB	
Ports	2 x Gbit LAN (RJ45) 4 x USB V2.0 1 x COM1 (RS232)	2 x Gbit LAN (RJ45) 4 x USB V2.0 1 x COM1 (RS232)	2 x Gbit LAN (RJ45) DVI-I graphics interface 4 x USB 2.0 1 x serial (COM1)
Free slots	–	In expansion rack: 3 x PCI104-Plus cards	IPC 627C: 2 x PCI (175/265 mm) or 1 x PCI-Express x16 (175 mm) and 1 x PCI (265 mm) IPC 827C: 2 x PCI (290 mm), 1 x PCI (240 mm), 1 x PCIe x16 (240 mm) and 1 x PCIe x4 (185 mm)

Four device classes are available for different requirements:

- SIMATIC IPC 227D – the new Nanobox PC with maximum flexibility – absolutely maintenance-free
- SIMATIC IPC 427C – ultra-compact and maintenance-free: the flexible Microbox PC
- SIMATIC IPC 627C (Box PC) – high performance in the most restricted space
- SIMATIC IPC 827C (Box PC) – high performance with high expandability

The processors are available in several versions (see table 2.8), as is the mass storage: Hard disks Serial ATA 3.5”, Solid-State Drive (SSD) Serial ATA 2.5” with SLC technology or CompactFlash drive. For increased data security through mirror drives, the IPC 627C and IPC 827C can be equipped with a RAID1 system.

I/O modules are connected via PROFIBUS or PROFINET (not with IPC 227D). IPC 427C can be equipped with up to four expansion frames that contain binary or analog inputs and outputs, depending on version.

Optical drives can be connected via USB for the IPC 227D and IPC 427C. IPC 627C and IPC 827C include an optical DVD drive.

The devices can be supplied without operating system, with a standard operating system, or with an embedded operating system. The IPC 227D and IPC 427C are available as preinstalled turnkey bundles with the WinAC RTX PLC software and/or WinCC flexible/RT HMI software.

Embedded controllers in S7-300 design

SIMATIC S7-mEC (modular embedded controller) is an embedded controller in S7-300 design and can be expanded by up to 32 I/O modules (SM) in S7-300 design in several tiers. The most important field of use is in the construction of special and series machines. Programming, diagnostics, and commissioning are carried out with STEP 7.

The processor is an Intel Core Duo 1.2 GHz, the memory comprises 1 GB RAM and 4 GB Flash Disk. The controller has interfaces for 10/100 Mbit/s Ethernet RJ45 (PROFINET) and USB. The operating system is Microsoft Windows XP embedded. The software packages WinAC RTX 2010 and SIMATIC Softnet-S7 Lean, including SIMATIC NET OPC Server, are pre-installed in the RTX version.

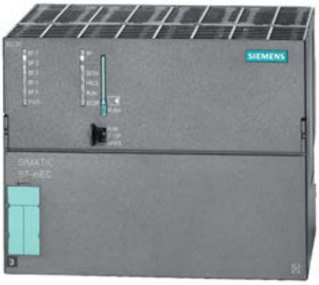


Fig. 2.18 SIMATIC S7-mEC

There are extension modules available for additional PC interfaces (DVI-I, USB, Gigabit Ethernet and serial interfaces, memory card and PCI-104 slots).

2.15 ET 200 distributed I/O system

ET 200 is the device family for the distributed I/Os on PROFIBUS DP and PROFINET IO. Depending on their use (locally on the machine or in the process), the mechanical properties can be highly different, especially the degree of protection: IP 20 for installation in a control cabinet and IP 65 for mounting directly on the machine.

The range of ET 200 stations extends from a simple compact station practically corresponding to an I/O module, to a station with modular design and several modules, up to the “intelligent” station, which can execute a user program with its own CPU.

ET 200L

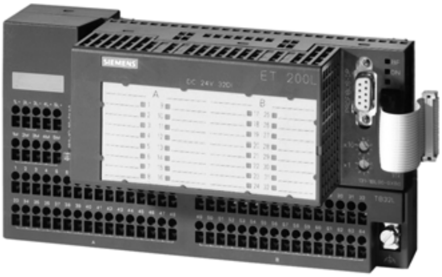


Fig. 2.19 ET 200L for PROFIBUS DP

ET 200L is a very small and compact I/O device with IP 20 degree of protection and is preferably used for the lower performance range and where only limited space is available.

The maximum data transfer rate on the PROFIBUS is 1.5 Mbit/s. ET 200L consists of a terminal block to which the wiring is connected and an electronics block containing the digital inputs and outputs. ET 200L is available with 16 or 32 channels and cannot be expanded. The ET 200L with 16 digital inputs is also available in a hardened SIPLUS version.

ET 200M

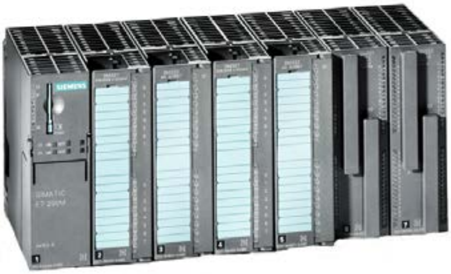


Fig. 2.20 ET 200M with IM 153-4 PN

ET 200M is a modular I/O system with IP 20 degree of protection and is particularly suitable for individual and complex automation tasks. Depending on the interface module, up to 8 or 12 modules from the S7-300 range can be used (the High-Feature version also allows the use of function and communication modules).

The internal bus signals are passed on from module to module via a bus connector. If active bus submodules are used onto which the modules are snapped, the modules can be replaced during running operation (hot swapping).

The maximum data transfer rate on the PROFIBUS DP is 12 Mbit/s and 10 or 100 Mbit/s on the PROFINET IO. With the integral 2-port switch, a linear topology can be implemented with the ET 200M as IO device without external devices.

The ET 200M is also available in a hardened SIPLUS version, and can be used together with S7-300 modules possessing the same characteristics in environments with increased demands.

ET 200M can also be used in fault-tolerant systems for redundant operation. The failsafe S7-300 modules can be used in the ET 200M – also mixed with standard modules. Together with Ex digital and analog modules, intrinsically-safe sensors and actuators can be connected from zones 1 and 2 of hazardous plants.

ET 200S

ET 200S is a versatile I/O system with degree of protection IP 20 whose bit-modular design allows exact adaptation to the automation task. Digital input/output modules, analog input/output modules, technology modules, motor starters, and frequency converters are available. Up to 63 I/O modules can be connected to the ET 200S interface module. The I/O modules can be replaced during ongoing operation; they are snapped onto terminal modules which contain the wiring. ET 200S is available with a PROFIBUS DP interface (maximum data transfer rate 12 Mbit/s) or a PROFINET IO interface (maximum data transfer rate 100 Mbit/s).

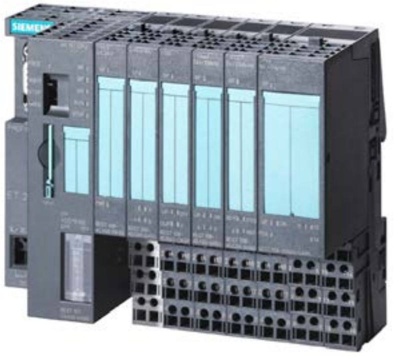


Fig. 2.21 ET 200S with IM 151 CPU

Together with the IM 151-7 CPU interface module, ET 200S can be used as a mini PLC. In association with the DP master module, the IM 151-7 CPU also has DP master functionality. The PLC functionality corresponds to that of a CPU S7-314. ET 200S with the IM 151-8 PN/DP CPU interface module can additionally be operated as an IO controller on PROFINET IO.

ET 200S is available with integral safety technology, where standard modules and failsafe modules can be used together. A failsafe mini PLC can be implemented with the IM 151-7 F-CPU interface module and the S7 Distributed Safety option package.

The ET 200S is also available in a hardened SIPLUS version as a PROFIBUS-DP slave with digital inputs and outputs.

ET 200S COMPACT is a range of interface modules with onboard I/O, either with 32 digital inputs or with 16 digital inputs and outputs. Up to 12 ET 200S I/O modules (except F modules) can be connected to these interface modules so that a station can have up to 128 channels (mixed digital and analog).

ET 200S can also be used in fault-tolerant systems following a Y-Link (bus coupler for transition from redundant to single-channel PROFIBUS DP).

ET 200iSP

ET 200iSP is an intrinsically-safe I/O system with degree of protection IP 30 for use in hazardous gas and dust areas, i.e. in zones 1 and 2 as well as 21 and 22, with connection of intrinsically-safe signals from zones 0, 1 or 2 and 20, 21, or 22.

The ET 200iSP consists of a power supply module, an interface module and up to 32 electronics modules for the digital and analog inputs and outputs. The modules are snapped onto terminal modules, and hot swapping is possible.

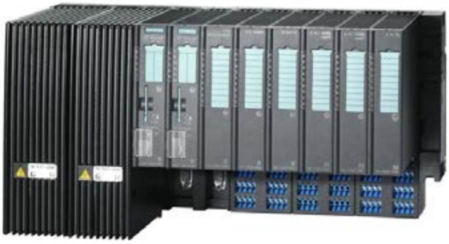


Fig. 2.22 ET 200iSP with redundant IM 152-1

To achieve intrinsically-safe operation of an ET200iSP, the bus segment must also have an intrinsically-safe design. This is achieved using an RS 485-IS coupler (6ES7 972-0AC80-0XA0) as isolating transformer. The maximum data transfer rate is 1.5 Mbit/s. The ET 200iSP can also be used for redundant operation in fault-tolerant systems.

ET 200R

ET 200R is available as a handling or welding module with IP 65 degree of protection in a metal enclosure for environments with high electromagnetic interference. The module has 16 channels, of which 8 channels are digital outputs and 8 channels can be individually parameterized as digital inputs or outputs. The maximum data transfer rate on the PROFIBUS is 12 Mbit/s.

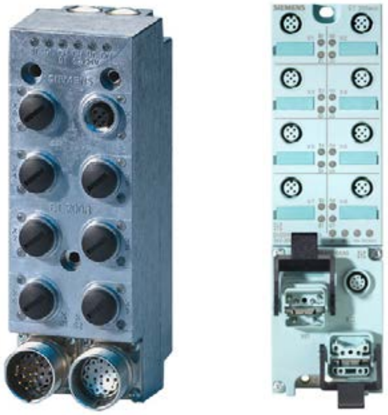


Fig. 2.23 ET 200R (left) and ET 200eco with ECOFAST connection

ET 200eco

ET 200eco with IP 65/67 degree of protection is the low-cost solution for processing digital signals at machine level.ET 200eco comprises a basic module and a connection block of different designs. Modules are available with 8 or 16 digital inputs, 8 or 16 digital outputs, 8 digital inputs and outputs each, and in fail-safe versions with 4 or 8 digital inputs.

The maximum data transfer rate on PROFIBUS is 12 Mbit/s. During commissioning and servicing, the modules can be disconnected interruption-free from the PROFIBUS and reconnected.

ET 200eco PN is the compact block I/O for processing digital, analog and IO-Link signals for connection to the PROFINET IO bus system. The design of the digital input and output modules is as with the PROFIBUS version of ET 200eco. Additionally available are an analog input module with 8 channels (4 × U/I, 4 × TC/RTD), an analog output module with 4 channels (U/I), and an IO-Link master with 4 IO-Link signals, 8 digital inputs, and 4 digital outputs.

ET 200eco PN is equipped with a 2-port switch so that a linear topology can be set up without additional devices. The maximum data transfer rate on the PROFINET is 100 Mbit/s.

ET 200pro

ET 200pro is a modular I/O system with IP 65/67 degree of protection for use without a control cabinet. It consists of a module support and connection modules which accommodate the interface module for the bus connection and the electronic modules. Power modules for the load power supply combine the electronic modules into potential groups.

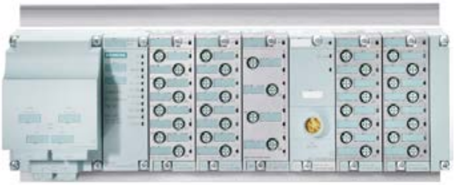


Fig. 2.24 ET 200pro with digital modules

The electronic modules are digital inputs/outputs and analog inputs/outputs. They can be replaced during ongoing operation. A frequency converter and motor starter (direct-on-line and reversing starter) as well as a pneumatic interface module with 16 outputs for the FESTO CPV 10 valve terminal are also available in this design.

Interface modules are available for ET 200pro with a PROFIBUS DP interface (maximum data transfer rate 12 Mbit/s) or a PROFINET IO interface (maximum data transfer rate 100 Mbit/s) with the facility for wireless connection to a PROFINET IO controller. The PROFINET interface module has a 2-port switch for easy configuration of a linear topology.

Together with the IM 154-8 PN/DP CPU interface module, ET 200pro can be used as a mini PLC on site. Operation as a DP master or DP slave is possible on the PROFIBUS DP and as an IO controller on the PROFINET IO. The PLC functionality of the interface module corresponds to that of a CPU 315-2 PN/DP.

2.16 The SIMATIC programming device

The **Field PG M3** is a mobile, industry-standard notebook for the commissioning, service, and maintenance of automation plants. It features high-performance Intel Core i processor technology, a 15.6” widescreen display, long battery life, fast work memory, and all standard interfaces for industrial applications.

Either Windows XP Professional English Multilanguage (32-bit) or Windows 7 Ultimate (32-bit, available soon: 64-bit) is pre-installed as operating system; English, German, French, Spanish, or Italian can be selected as language.

The engineering software is also pre-installed (status in 2011): STEP 7 (V5.5 with option packages for S7-GRAPH, S7-SCL and S7-PLCSIM), STEP 7 Professional (TIA Portal, V11.0), WinCC flexible Advanced, WinCC Advanced (TIA Portal) and – under Windows XP also – STEP 7-Micro/Win and STEP 5.

Fig. 2.25 Field PG M3

The pre-installed software must be activated. Every Field PG M3 is delivered as standard with a license that is limited to 14 days. The following licenses can also be acquired:

- STEP 7 Professional license (includes licenses for STEP 7 Basic/Professional TIA Portal, WinCC flexible Advanced, STEP 7-Micro/Win and – optionally – STEP 5)
- STEP 7 Professional Power Pack license for upgrading from STEP 7 to STEP 7 Professional (TIA Portal)
- Upgrade license for STEP 7 for users who already own older versions of the respective software

Available as an accessory, the Image & Partition Creator V3.1 data backup software allows the creation of a backup image of a drive and the restoration of a drive from a backup image.

The programming device can be ordered as a standard or premium version. The Premium/S5 version also has an integrated S5 online interface and an S5 EPROM adapter. All SIMATIC Memory Cards can be programmed (interfaces for memory card and Micro Memory Card).

The integral wireless LAN (in accordance with IEEE 802.11 a/b/g/n) is approved for operation in Europe (CE), the USA (FCC), Canada (IC), and China (CCC).

3 STEP 7: Engineering Tool for SIMATIC

STEP 7 is the engineering tool for SIMATIC and runs on a standard PC under the Microsoft Windows operating systems. STEP 7 is available in different versions.

A “generation change” is currently underway. In addition to the standard V5.x versions of STEP 7 (“STEP 7 with SIMATIC Manager”), there are now also the new STEP 7 V1x versions (“STEP 7 inside TIA Portal”). The most important new features are an improved user interface and integration into the TIA Portal with cross-application data management in combination with other automation tools such as WinCC, the engineering tool for HMI devices.

3.1 Overview of STEP 7 variants

STEP 7 V5.5

STEP 7 is the basic software for the SIMATIC S7-300/400, SIMATIC ET 200 CPU, and SIMATIC WinAC programmable logic controllers. As the central tool, *SIMATIC Manager* manages any generated automation data and all tools required for processing this data. With STEP 7, you configure the hardware of the SIMATIC controllers, set the module addresses and parameters, and configure the network connections. Last but not least, STEP 7 provides the programming languages ladder logic (LAD), function block diagram (FBD), and statement list (STL), with which you can write the user program for the controller.

STEP 7 is supplied with support for five languages (English, German, French, Italian, and Spanish). Versions in Japanese and Chinese are also available. In version V5.5 with SP1, STEP 7 requires the operating system MS Windows XP Professional with SP2 or SP3, MS Windows Server 2003 SP2 standard edition as a workstation, MS Windows 7 32/64-bit Ultimate, Professional, and Enterprise (standard installation, not with XP mode), or MS Windows Server 2008 R2 64-bit. STEP 7 requires up to 2 GB on the hard disk, depending on the scope of installation and on the operating system used. A swap file is also necessary and its size must be at least twice that of the main memory.

You require a license (user authorization) in order to use STEP 7. It is supplied on a USB flash drive. You will be requested to provide authorization after installing STEP 7 if a license key is not yet present on the hard disk. You can also provide the authorization at a later time. If you lose the authorization, e.g. due to a defective hard disk, you can revert to the trial license delivered with STEP 7, which is valid for a limited duration until you acquire a new authorization.

STEP 7 Professional

STEP 7 Professional comprises STEP 7 and the option packages S7-GRAPH, S7-SCL, and S7-PLCSIM. In addition to the PLC languages known from STEP 7, ladder logic (LAD), function block diagram (FBD), and statement list (STL), STEP 7 Professional thus also supports the remaining IEC languages GRAPH and structured control language (SCL). S7-PLCSIM furthermore permits an offline simulation of the user program.

The requirements to install and operate STEP 7 Professional are the same as with STEP 7.

STEP 7 Lite

STEP 7 Lite is a low-cost alternative for simple stand-alone applications with SIMATIC S7-300, ET 200S, and ET 200X. With STEP 7 Lite you configure the hardware for SIMATIC S7-300 including the distributed I/O. The programming languages ladder logic (LAD), function block diagram (FBD), and statement list (STL) are fully supported. User programs created with STEP 7 Lite can be further processed under STEP 7. The option packages S7-PLCSIM and TeleService can be used in combination with STEP 7 Lite. In version V3.0 with SP4, STEP 7 Lite runs under the operating systems MS Windows XP Home with SP2 and SP3 and MS Windows XP Professional with SP2 and SP3, with MS Internet Explorer 6.0 or above.

STEP 7 Micro/Win

STEP 7 Micro/WIN is the engineering software for SIMATIC S7-200. It permits you to create the user program, optimized for processing in an S7-200 CPU, in the form of a statement list or graphically as an LAD or FBD representation. The user program consists of one singular block that can contain subprograms. The functional scope of the S7-200 contains, among others, binary operations, timer and counter functions, fixed point and floating point arithmetic, comparison functions, PID control, and data transfer. STEP 7 Micro/Win V4.0 with SP4 runs under the operating systems MS Windows 2000 with SP3 and MS Windows XP Home/Professional.

STEP 7 V11 (TIA Portal)

STEP 7 Basic/Professional inside TIA Portal is supplied in five languages: English, French, German, Italian, and Spanish. You require a license (user authorization) in order to use STEP 7. It is supplied on a USB flash drive. You will be requested to provide authorization after installing STEP 7 if a license key is not yet present on the hard disk. You can also provide the authorization at a later time. If you lose the authorization, e.g. due to a defective hard disk, you can revert to the trial license delivered with STEP 7, which is valid for a limited duration until you acquire a new authorization.

A dual-core processor with 2.2 GHz or similar is recommended. The main memory should be 2 GB of DDR2 RAM. STEP 7 Basic requires approx. 2 GB on the hard disk.

STEP 7 Basic V11 (TIA Portal)

STEP 7 Basic (TIA Portal) is the tool for programming the SIMATIC S7-1200 programmable controller. With STEP 7 Basic, you configure the hardware of the S7-1200 controller, parameterize the modules, and configure the data communication between S7 stations and to HMI stations.

The control program is created in the programming languages Ladder Logic (LAD), Function Block Diagram (FBD), and Structured Control Language (SCL). STEP 7 Basic also supports you in testing, commissioning, service, and in configuring the integrated motion and technology functionality in a CPU 1200. Furthermore, STEP 7 Basic contains the engineering tool WinCC Basic for configuring the PROFINET-based SIMATIC HMI Basic Panels.

In version V11 with SP 2, STEP 7 Basic requires the operating system MS Windows XP (Home with SP3 or Professional with SP3) or MS Windows 7 (Home Premium, Professional, Enterprise, or Ultimate) 32 and 64-bit.

STEP 7 Professional V11 (TIA Portal)

STEP 7 Professional (TIA Portal) is the tool for programming the automation systems SIMATIC S7-300/400, SIMATIC S7-1200, SIMATIC ET 200 CPU and SIMATIC PC systems (SIMATIC WinAC). With STEP 7 Professional, you configure the hardware of the SIMATIC controllers, parameterize the modules, and configure the data communication between S7 stations and to HMI stations.

The control program is created in the programming languages Ladder Logic (LAD), Function Block Diagram (FBD), and Structured Control Language (SCL). In addition, statement list (STL) and GRAPH sequence control are available (not for S7-1200). The simulation software PLCSIM is integrated in STEP 7 Professional for testing the user program in offline mode. STEP 7 Professional also supports you in testing, commissioning, service, and in configuring the motion and technology functionality integrated in a CPU. Furthermore, STEP 7 Professional contains the engineering tool WinCC Basic for configuring the PROFINET-based SIMATIC HMI Basic Panels.

STEP 7 Professional V11 with SP 2 requires as operating system MS Windows XP (Professional with SP3), MS Windows 2003 Server R2 (SP2 standard edition), MS Windows 7 (Ultimate, Professional or Enterprise) 32-bit and 64-bit, or MS Windows 2008 Server (SP2 standard edition).

3.2 Automating with STEP 7

STEP 7 is the central automation tool for SIMATIC controllers. It contains all required editors for creating and managing automation data. All the data arising in automation is collected in a so-called “project”, where it is available in a hierarchical structure and can be processed. The following basic procedure applies to the processing of automation data:

- ▷ Creating a new project
  The first step is to create a new project. You give the project a name and determine the storage location in the file system on the hard disk of the programming device.

- ▷ Adding a station
  The next step is to add a station (a device). For example, this can be a SIMATIC station or an HMI device.

- ▷ Configuring the hardware
  After the SIMATIC station is added, you specify its hardware setup. Depending on the station type, you specify the central rack and “equip” it with modules. If needed, you set the parameters of the modules, such as the runtime properties of the CPU or the user data addresses of the signal modules. If necessary, you can add expansion racks, connect them with the central rack, and equip them with modules.

- ▷ Networking the stations
  A project may contain several stations. With the network configuration, you create links between the stations to exchange data between them.

- ▷ Giving names to the I/O connections
  Before creating the control program, assign the I/O signals symbolic names.

- ▷ Creating the control program
  The control program consists of individual sections called “blocks”. You program the blocks with one of the available programming languages, and by so doing you specify the processing order of the blocks.

- ▷ Placing the control program in operation
  You connect the programming device with the SIMATIC station that controls the machine or process, load the control program into the CPU, and test the correct sequence with the help of STEP 7.

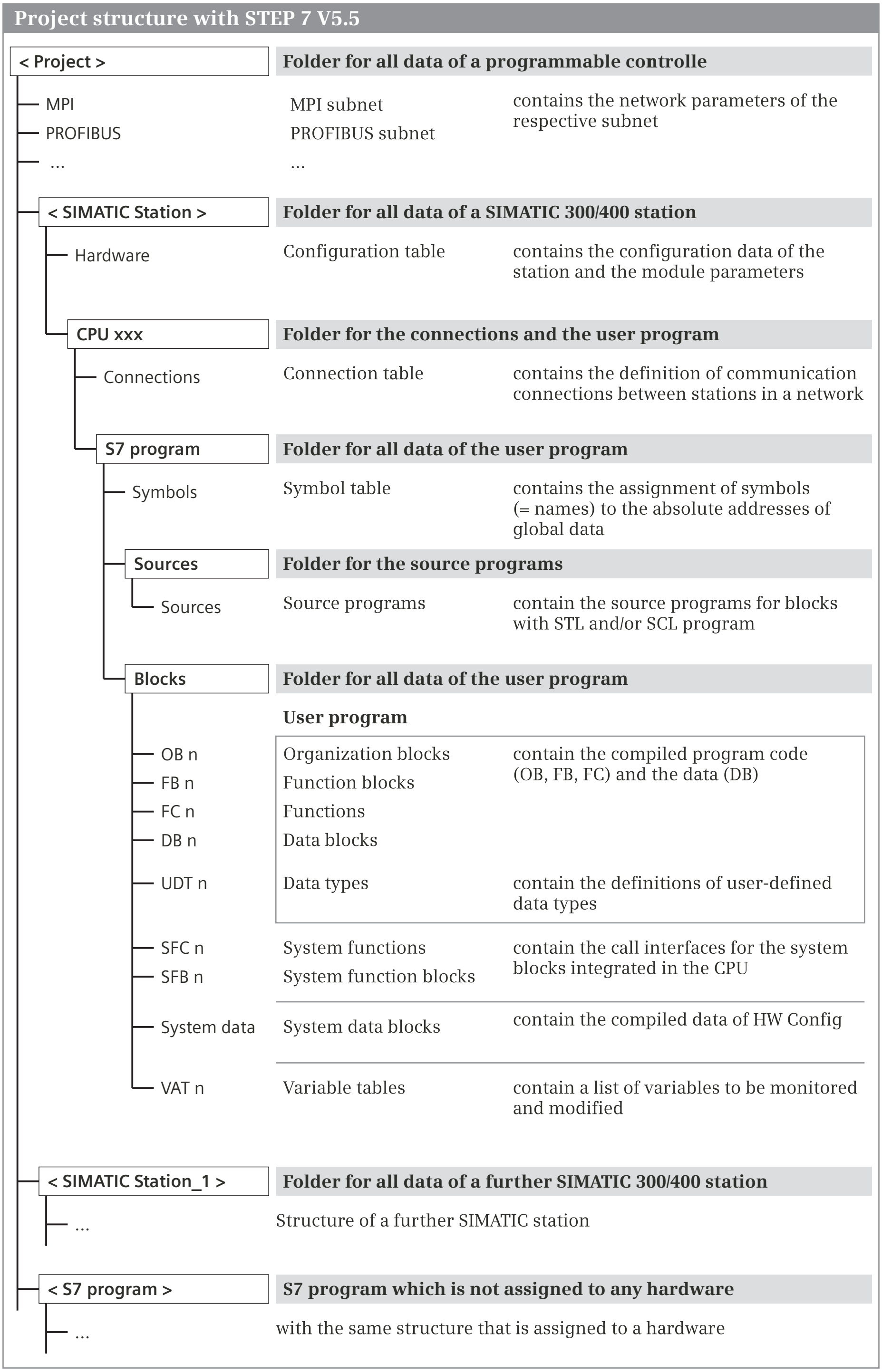
After commissioning, you create the final documentation and archive the project.

3.3 Editing projects with STEP 7 V5.5

SIMATIC Manager

SIMATIC Manager is the central tool in STEP 7 V5.5; after installation, you will find its icon on the Windows desktop. A double-click on the icon starts SIMATIC Manager and thus STEP 7. Alternatively, you can also start it via the taskbar:
Start > SIMATIC > SIMATIC Manager

With SIMATIC Manager, you work with objects in the STEP 7 world. These “logical” objects represent the “real” objects in your plant. A *project* contains the entire plant, a *station* corresponds to a programmable controller. In a station there is a



Project structure with STEP 7 V5.5

< Project >	**Folder for all data of a programmable controlle**	
MPI	MPI subnet	contains the network parameters of the respective subnet
PROFIBUS	PROFIBUS subnet	
…	…	
< SIMATIC Station >	**Folder for all data of a SIMATIC 300/400 station**	
Hardware	Configuration table	contains the configuration data of the station and the module parameters
CPU xxx	**Folder for the connections and the user program**	
Connections	Connection table	contains the definition of communication connections between stations in a network
S7 program	**Folder for all data of the user program**	
Symbols	Symbol table	contains the assignment of symbols (= names) to the absolute addresses of global data
Sources	**Folder for the source programs**	
Sources	Source programs	contain the source programs for blocks with STL and/or SCL program
Blocks	**Folder for all data of the user program**	
	User program	
OB n	Organization blocks	contain the compiled program code (OB, FB, FC) and the data (DB)
FB n	Function blocks	
FC n	Functions	
DB n	Data blocks	
UDT n	Data types	contain the definitions of user-defined data types
SFC n	System functions	contain the call interfaces for the system blocks integrated in the CPU
SFB n	System function blocks	
System data	System data blocks	contain the compiled data of HW Config
VAT n	Variable tables	contain a list of variables to be monitored and modified
< SIMATIC Station_1 >	**Folder for all data of a further SIMATIC 300/400 station**	
…	Structure of a further SIMATIC station	
< S7 program >	**S7 program which is not assigned to any hardware**	
…	with the same structure that is assigned to a hardware	

Fig. 3.1 Object hierarchy in a project created with STEP 7 V5.5

CPU containing a *program*, in our case an S7 program. This program in turn contains “folders” for further objects such as the *Blocks* object, which contains compiled blocks among other things. Fig. 3.1 shows the main parts of the program structure.

To start programming, you open an existing project or create a new one. The included sample projects are intended as an introduction. If you open such a project with *File* > *Open*, you will see the partitioned project window: On the left is the project tree (the object hierarchy); on the right is the content of the selected object (Fig. 3.2). Click on the box with the plus sign in the left pane to open further structure levels.

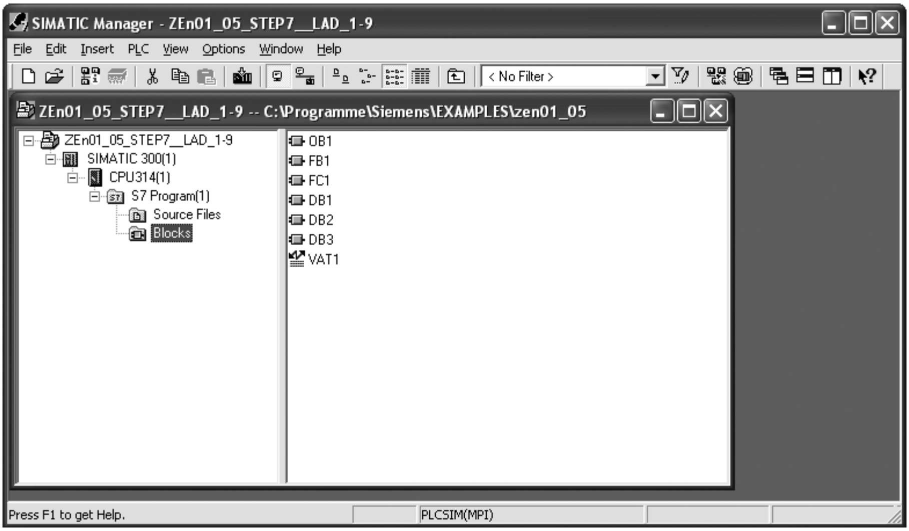

SIMATIC Manager - ZEn01_05_STEP7__LAD_1-9
File Edit Insert PLC View Options Window Help
< No Filter >
ZEn01_05_STEP7__LAD_1-9 -- C:\Programme\Siemens\EXAMPLES\zen01_05
ZEn01_05_STEP7__LAD_1-9
SIMATIC 300(1)
CPU314(1)
S7 Program(1)
Source Files
Blocks
OB1
FB1
FC1
DB1
DB2
DB3
VAT1
Press F1 to get Help.
PLCSIM(MPI)

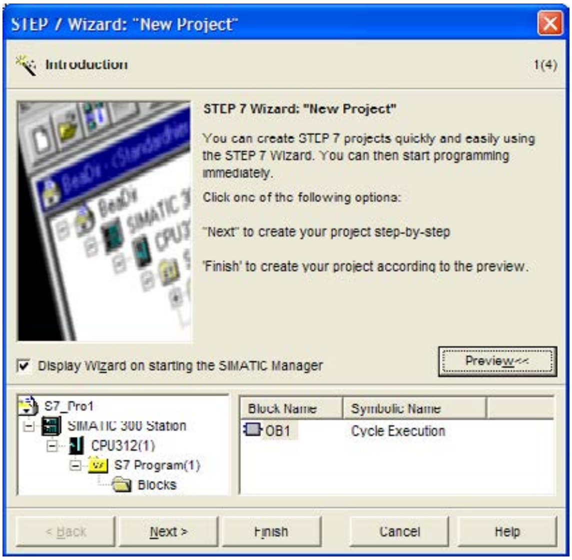
Fig. 3.2 Example of a project window in SIMATIC Manager

Opening objects on the lowest hierarchical level, e.g. double-clicking the OB1 icon in the right half of the window, opens the tool belonging to the object, in this example the program editor.

Creating a project

The STEP 7 wizard helps you to create a new project. In SIMATIC Manager, select the command *File* > *Wizard 'New Project'*. In the dialog window that appears (Fig. 3.3), you specify in steps the CPU, the planned organization blocks, and the programming language for the blocks.

The wizard then creates a project with an S7 station and the selected CPU, as well as an S7 Program folder, a Source Files folder and a Blocks folder with the selected organization blocks. Clicking on *Preview* shows the project structure created up to the present step.



STEP 7 Wizard: "New Project"

Introduction 1(4)

STEP 7 Wizard: "New Project"

You can create STEP 7 projects quickly and easily using the STEP 7 Wizard. You can then start programming immediately.

Click one of the following options:

"Next" to create your project step-by-step

'Finish' to create your project according to the preview.

☑ Display Wizard on starting the SIMATIC Manager

S7_Pro1
- SIMATIC 300 Station
  - CPU312(1)
    - S7 Program(1)
      - Blocks

Block Name	Symbolic Name
OB1	Cycle Execution

Fig. 3.3
STEP 7 wizard for creating a new project

If you would like to create a project manually or expand a project, start SIMATIC Manager and open an existing project with *File* > *Open* or create a new project with *File* > *New*.

Adding a SIMATIC station

To add a SIMATIC station, select the project in the project tree (in the left part of the project window) and select *Insert New Object* > ... from the shortcut menu, or in the SIMATIC Manager select *Insert* > *Station* > ... . The SIMATIC Manager adds the selected station to the project tree under the project. To configure the station, select the station and choose *Open Object* from the shortcut menu or *Edit* > *Open Object* from the main menu. The hardware configurator editor opens. The further operation is described in chapter 3.5 "Configuring a SIMATIC station" on page 74.

Libraries

Libraries are used to save reusable program components. Libraries are also structured hierarchically: They can contain S7 programs, which in turn may contain a user program (a folder for compiled blocks), a folder for source programs, and a symbol table. With the exception of online connections (no testing ability), when creating a program or program part the same functionality is available to you in a library as in a project. The STEP 7 scope of delivery contains the *Standard Library*, which includes the programs *Organization Blocks* (templates for the startup information in the temporary local data) and *System Function Blocks* (call interfaces for the SFC system functions and SFB system function blocks).

Managing, reorganizing and archiving projects

SIMATIC Manager manages projects and libraries in project lists or library lists. With *File* > *Manage*, SIMATIC Manager displays all known projects with name and storage path. You can now delete projects from the list that you do not want to display any longer (“hide”) or accept new projects into the project list (“display”). You manage the libraries in the same manner.

If you select *File* > *Rearrange*, SIMATIC Manager removes the gaps created due to the deletion of objects and optimizes the data storage on the hard disk similar to a defragmenting program. Rearranging can take a long time, depending on the data movement.

You can also archive a project or library by choosing *File* > *Archive*. The SIMATIC Manager stores the selected object in compressed form in an archive file (the project directory or library directory with all underlying directories and files). During archiving, stop editing the project or the library and make sure that all windows are closed.

Projects and libraries cannot be processed in the archived (compressed) state. Choose *File* > *Retrieve* to unpack an archived object and edit it again. The retrieved objects are automatically accepted in the project or library management.

Multi-projects make working with large projects easier

Multi-projects combine projects and libraries into a single unit. In multi-projects, communication links between projects such as S7 connections can be processed across projects. The individual projects can then be scaled down, made more manageable, and processed in parallel if necessary. Multi-projects are treated like projects, e.g. for administration, archiving, and retrieving.

Help functions

The SIMATIC Manager's online help supplies you with information during the programming session without the need to consult manuals. You need Microsoft Internet Explorer to use the online help. You select the help topics using the *Help* menu item. Under *Help* > *First Steps*, for example, the online help provides a brief overview of working with the SIMATIC Manager.

Help > *Contents* starts the central help function for STEP 7 from any application. This help function provides fundamental information. *Help* > *Context-sensitive help F1* gives you help that is dependent on the context, i.e. it displays information about an object that is selected with the mouse or about a current error message if you press the F1 key.

In the toolbar, there is a button with an arrow and a question mark. If you click on this button, the mouse pointer is also given a question mark. With this “Help” mouse pointer, you can now click on an object on the screen, e.g. an icon or a menu command, and receive the associated online help.

3.4 Editing projects with STEP 7 inside TIA Portal

TIA Portal

The TIA Portal is the framework software in which the engineering tool STEP 7 V11 is integrated. After installation, the icon can be seen on the Windows desktop. A double-click on the icon starts the TIA Portal and thus STEP 7. Alternatively, you can also start it via the task bar:
Start > *Siemens Automation*> *TIA Portal V11*

STEP 7 inside TIA Portal provides two views: the Portal view and the Project view.

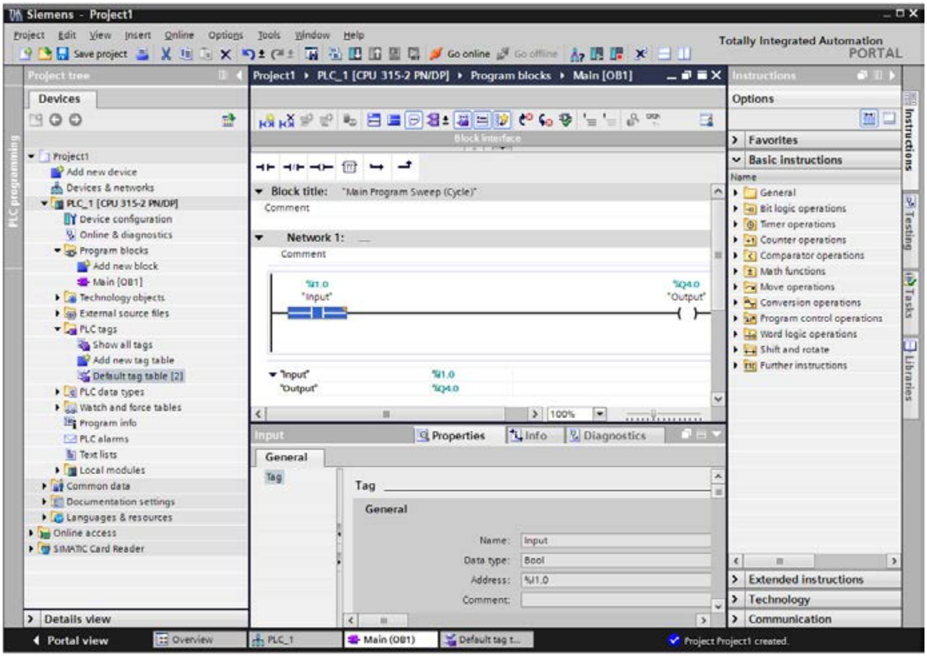
The **Portal view** (Fig. 3.5) is task-oriented. In the *Start portal* you create a project or open an existing project. A project includes all data of the automation project, both the controller and HMI device data. *First steps* shows you the procedure for solving an automation task and *Help* opens an information system in which all SIMATIC automation issues are explained in detail.

In the *Devices & networks* portal, you start the hardware and network configuration. The *PLC programming* portal offers an overview of the program blocks that have already been programmed; from here, you can also open the editors to create the control program, the cross-reference list, and the program structure. The *Visualization* portal provides the most important tools for configuration of PROFINET-based Basic Panels. The *Online & Diagnose* portal allows you to select the target system for the online connection and to start online operation.

Project view supports object-oriented work (Fig. 3.4). The processed object is in the *working window* in the center, with further windows that supply supporting information arranged around it. The properties of the object selected in the working window are displayed and edited in the *inspector window* at the lower edge of the screen. To the left of the working window is the *project tree*, which contains all the objects of a project, and all editors required to process it, in a hierarchical structure.

Fig. 3.6 shows the main parts of the project structure in the graphic. On the right hand side of the project view, a *task window* makes the available automation objects available: for HW Config, for example, the hardware catalog; for the PLC programming, the instructions catalog and the block libraries; for the visualization configuration, the catalog with the process image objects; and in online mode, the online tools.

As in Microsoft Windows, the size of all windows can be changed. The windows at the edge of the screen can also be minimized to create room for the working window. In addition, the working window can be separated from the window arrangement and maximized as a separate window.



Block title: "Main Program Sweep (Cycle)"

"Input"	%I1.0
"Output"	%Q4.0

Tag

General

Name: Input

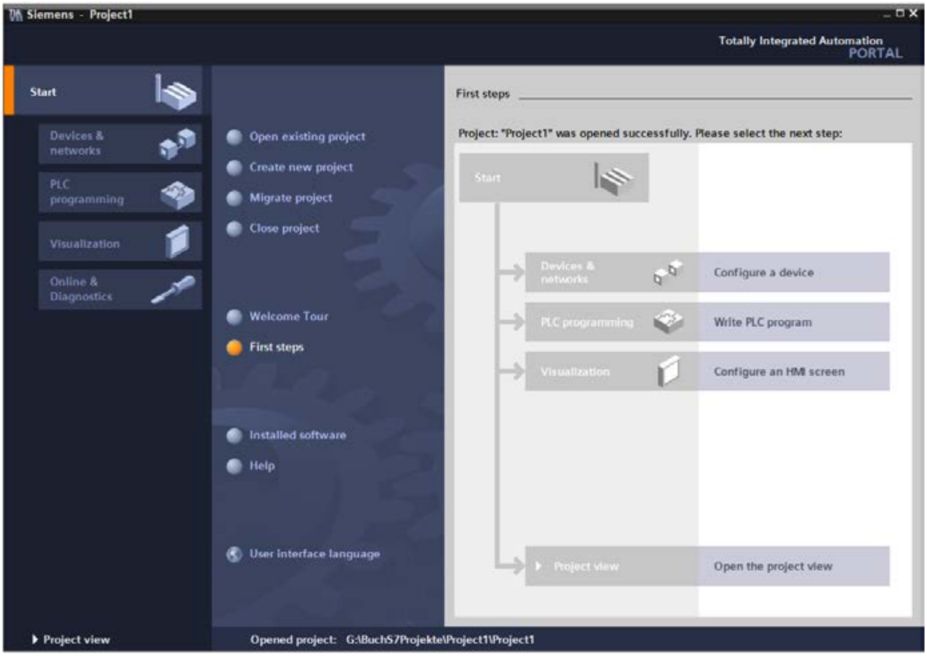
Data type: Bool

Address: %I1.0

Comment:

Project Project1 created.

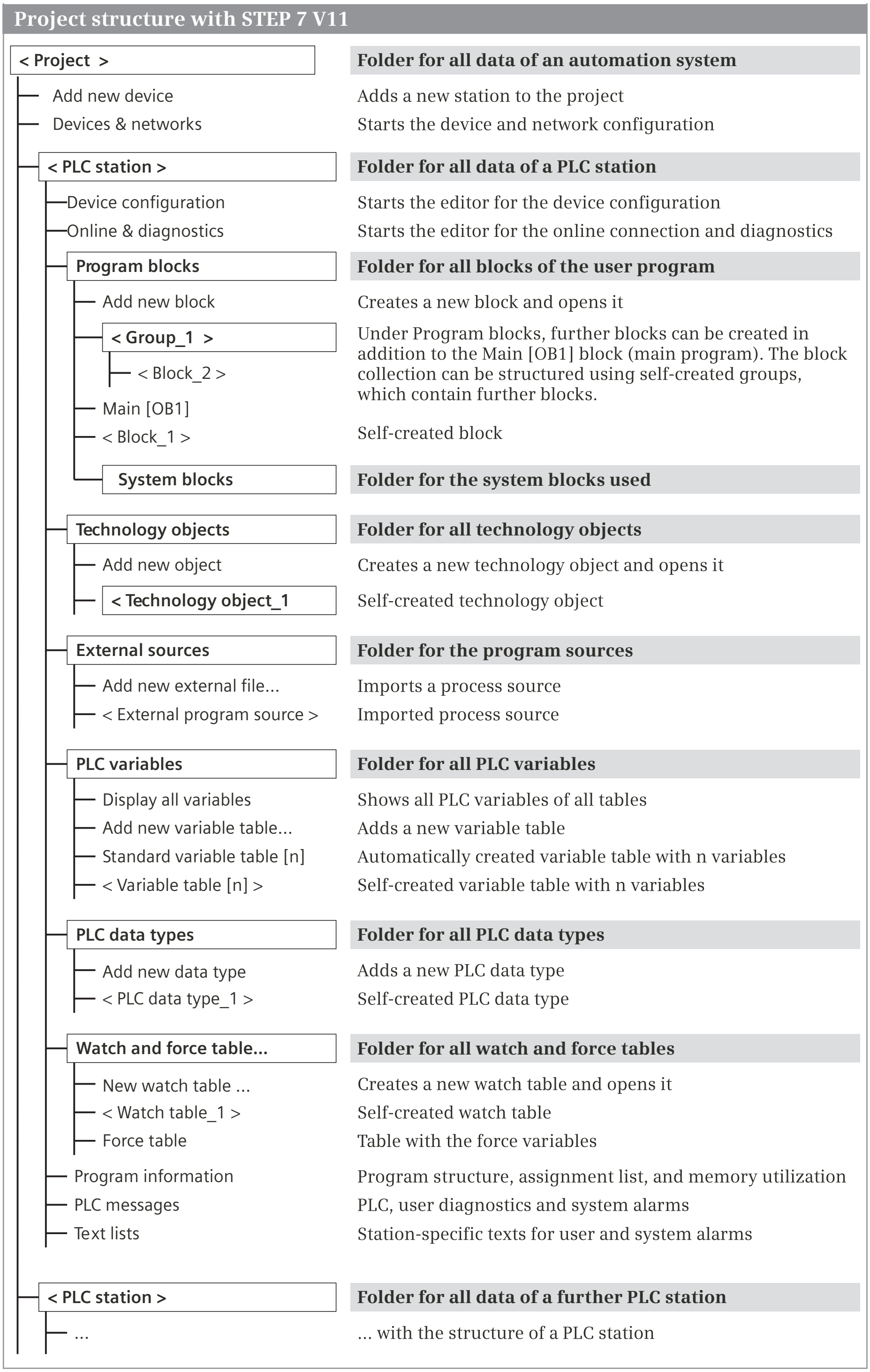
Fig. 3.4 Project view for STEP 7 inside TIA Portal



Project: "Project1" was opened successfully. Please select the next step:

Opened project: G:\BuchS7Projekte\Project1\Project1

Fig. 3.5 Portal view for STEP 7 inside TIA Portal



Project structure with STEP 7 V11

< Project >	**Folder for all data of an automation system**
Add new device	Adds a new station to the project
Devices & networks	Starts the device and network configuration
< PLC station >	**Folder for all data of a PLC station**
Device configuration	Starts the editor for the device configuration
Online & diagnostics	Starts the editor for the online connection and diagnostics
Program blocks	**Folder for all blocks of the user program**
Add new block	Creates a new block and opens it
< Group_1 >	Under Program blocks, further blocks can be created in addition to the Main [OB1] block (main program). The block collection can be structured using self-created groups, which contain further blocks.
< Block_2 >	
Main [OB1]	
< Block_1 >	Self-created block
System blocks	**Folder for the system blocks used**
Technology objects	**Folder for all technology objects**
Add new object	Creates a new technology object and opens it
< Technology object_1	Self-created technology object
External sources	**Folder for the program sources**
Add new external file...	Imports a process source
< External program source >	Imported process source
PLC variables	**Folder for all PLC variables**
Display all variables	Shows all PLC variables of all tables
Add new variable table...	Adds a new variable table
Standard variable table [n]	Automatically created variable table with n variables
< Variable table [n] >	Self-created variable table with n variables
PLC data types	**Folder for all PLC data types**
Add new data type	Adds a new PLC data type
< PLC data type_1 >	Self-created PLC data type
Watch and force table...	**Folder for all watch and force tables**
New watch table ...	Creates a new watch table and opens it
< Watch table_1 >	Self-created watch table
Force table	Table with the force variables
Program information	Program structure, assignment list, and memory utilization
PLC messages	PLC, user diagnostics and system alarms
Text lists	Station-specific texts for user and system alarms
< PLC station >	**Folder for all data of a further PLC station**
...	... with the structure of a PLC station

Fig. 3.6 Object hierarchy in a project created with STEP 7 (TIA Portal)

Creating a project inside TIA Portal

To create a project, open the TIA Portal and select *Create new project* in the Start portal. Assign a name to the project and set a path in which the project is to be saved. After clicking the *Create* button, the project is created and the next steps are displayed in the Start portal for selection (see also Fig. 3.5 on Seite 71):

- Configuring a device
  STEP 7 changes to the *Devices & networks* portal in which you can insert a new PLC station (SIMATIC station) into the project and open it for editing.
- Creating a PLC program
  STEP 7 changes to the *PLC programming* portal in which you can edit the *Main* block (organization block OB 1) or insert a new block and open it for editing. Select a PLC station if applicable. This can also be an “unspecified CPU” which you can later replace with a CPU from the hardware catalog.
- Configuring an HMI screen (if WinCC is installed inside TIA Portal)
  STEP 7 changes to the *Visualization* portal in which you can create a new HMI station (HMI device) or configure an existing one. From this portal you start the configuration of the process images, editing of HMI variables and messages, and the HMI simulator.
- Opening the project view
  STEP 7 changes to the project view, in which you can carry out the next steps (insert and configure PLC station, insert and program block, or insert and configure HMI station).

In the project view, select *Project > New* to create a new project or open an existing one with *Project > Open*.

If during project editing you want to save a block, for example, you must always save the entire project with *Project > Save*. *Project > Save As...* saves the project under another name or path.

Libraries

Libraries are used to save reusable program components. These can include stations, blocks, PLC variable tables, process images, or picture elements, for example. The libraries are displayed in a task card in the task window. A project library and global libraries are available.

A project library which you can fill with objects is automatically created when you create a project. You can structure the contents of the library using folders. A project library is always opened, saved, and closed together with the project.

Objects which can be used in multiple projects are saved in global libraries. There are global system libraries which are supplied with STEP 7, and global user libraries which you create yourself. A global library is opened, saved, and closed independent of the project. If you wish to use a global library simultaneously with other users, the library must be opened in read-only mode.

Migrating projects

Automation projects that have been created using STEP 7 V5.4 SP5 or later can be migrated into the TIA Portal. The target project resulting from the original project can then be edited further inside TIA Portal using STEP 7. The migration tool is delivered together with STEP 7 or can be downloaded from the Internet.

The prerequisite for migration is the installation of all applications with which the original project was created. This also includes the option packages and the Hardware Support Packages (HSP). If these applications are installed together with the TIA Portal on the programming device, you can migrate the original project directly. Otherwise you install the migration tool on the programming device which contains the original project with the required applications, create an intermediate project with the file extension .am11 from the original project, transfer this intermediate project to a programming device with the TIA Portal, and then migrate the project.

Help information system

To call the help function, click on *Help* in the portal view or select the *Help > Display help* command in the main menu in the project view. A window appears with the help information system. The online help is roughly divided according to the project processing steps: Configuration, parameterization, and networking of devices, structuring and programming of the user program, visualization of processes, and utilization of the online and diagnostics functions. *Readme* provides general information on STEP 7 and further information which could no longer be included in the online help. A comprehensive description of all available instructions, including extended instructions, can be found under *PLC programming* and *References*.

3.5 Configuring a SIMATIC station

You plan the hardware configuration of your programmable logic controller with HW Config. Configuration is carried out offline without a connection to the CPU. The prerequisite for work with HW Config is an open project with a PLC station. As a result of the hardware configuration you are given an image of your plant's “real” automation system.

General procedure

In a rack depicted in tables or graphically, “plug” the desired module into a permissible slot. Then hold down the mouse button to “drag” the module from the hardware catalog, which contains all available modules, to the slot in the rack and “drop” the component into place. When plugging in the modules, follow the slot rules of the automation system you are using. A “no stopping” icon shows that you cannot place the module you have just selected in the desired slot.

The hardware catalog shows all currently available modules in a hierarchical structure. Using filter settings, you can reduce the number of displayed objects.

Click down through the folder structure until you reach the desired object. If you select an object in the hardware catalog, the most important object properties are shown.

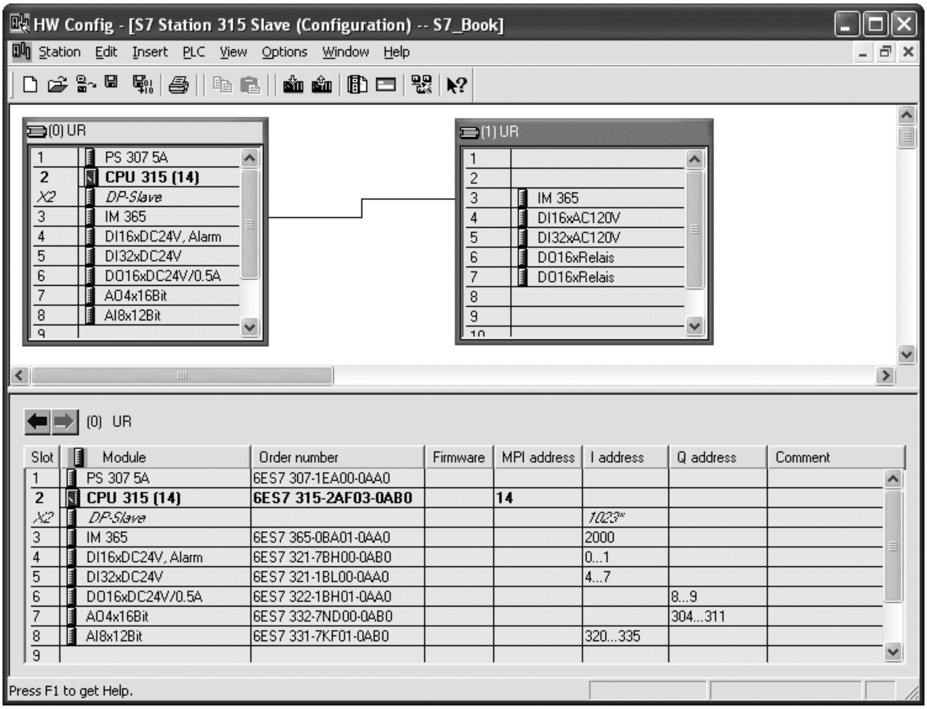
A module is placed in the slot with the default properties. STEP 7 also automatically sets the address of the modules. If necessary, you can later change the module properties and the addresses to meet your requirements.

If you add expansion units to an S7-300/400 station, the procedure is the same as with the central controller. Drag the rack from the hardware catalog to the working window, add the modules to the expansion rack, and connect the interface modules in the central controller and expansion unit to each other.

You can save the configuration data any time. If the hardware configuration is complete, compile it in a form that can be read by the relevant controller. You can only load a hardware configuration to a CPU if the compilation is without errors.

Arranging the modules using STEP 7 V5.5

Start HW Config for the selected station with *Edit* > *Open Object* or by double-clicking on *Hardware* in the open *SIMATIC 300/400 Station* folder. After opening, HW Config displays the working window and the hardware catalog. In the upper part, the working window shows the racks with the slot assignment, and the lower part it shows the selected rack as a configuration table (Fig. 3.7).F



1	PS 307 5A
2	**CPU 315 (14)**
X2	*DP-Slave*
3	IM 365
4	DI16xDC24V, Alarm
5	DI32xDC24V
6	DO16xDC24V/0.5A
7	AO4x16Bit
8	AI8x12Bit
9	

1	
2	
3	IM 365
4	DI16xAC120V
5	DI32xAC120V
6	DO16xRelais
7	DO16xRelais
8	
9	
10	

Slot	Module	Order number	Firmware	MPI address	I address	Q address	Comment
1	PS 307 5A	6ES7 307-1EA00-0AA0					
2	**CPU 315 (14)**	**6ES7 315-2AF03-0AB0**		**14**			
X2	*DP-Slave*				*1023**		
3	IM 365	6ES7 365-0BA01-0AA0			2000		
4	DI16xDC24V, Alarm	6ES7 321-7BH00-0AB0			0...1		
5	DI32xDC24V	6ES7 321-1BL00-0AA0			4...7		
6	DO16xDC24V/0.5A	6ES7 322-1BH01-0AA0				8...9	
7	AO4x16Bit	6ES7 332-7ND00-0AB0				304...311	
8	AI8x12Bit	6ES7 331-7KF01-0AB0			320...335		
9							

Fig. 3.7 Example of a HW Config working window with STEP 7 V5.5

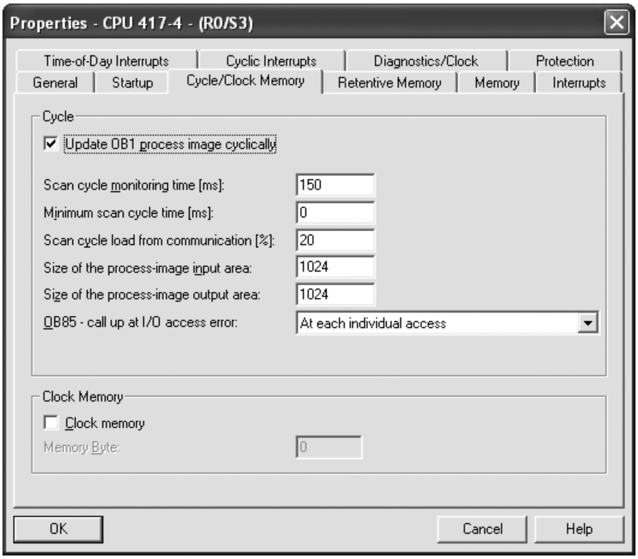
You can display and hide the hardware catalog via *View* > *Catalog*. It contains all available racks, modules, and interface modules that STEP 7 recognizes. Using *Options* > *Edit Catalog Profiles*, you can make up your own hardware catalog, which – in the structure of your choice – only displays the modules with which you want to work.

Now you can select the desired modules from the Hardware Catalog and, keeping the left mouse button pressed, drag them to the slot in the rack. Drag another rack from the hardware catalog to an empty space in the working window.

After configuration, *Station* > *Check Consistency* shows you whether all entries are error-free. *Station* > *Save* stores the configuration tables with all parameterization data in your project to the hard drive. *Station* > *Save and Compile* simultaneously compiles and saves the configuration tables and stores the compiled data in the *System Data* object in the offline *Blocks* folder.

Parameterizing and addressing the modules using STEP 7 V5.5

You define the properties of a module by assigning parameters to it. Assigning parameters is only necessary if you want to change the default parameters. A prerequisite for assigning parameters is that you arrange the module in a rack. Double-click the module in the rack or select the module and choose *Edit* > *Object Properties*. A dialog appears with module-specific tabs that show the adjustable parameters. If you assign parameters to a CPU in this manner, you set the execution properties of the user program (Fig. 3.8).



Cycle

☑ Update OB1 process image cyclically

Scan cycle monitoring time [ms]: 150

Minimum scan cycle time [ms]: 0

Scan cycle load from communication [%]: 20

Size of the process-image input area: 1024

Size of the process-image output area: 1024

OB85 - call up at I/O access error: At each individual access

Clock Memory

□ Clock memory

Memory Byte: 0

Fig. 3.8 Parameterizing modules with STEP 7 V5.5, example for a CPU

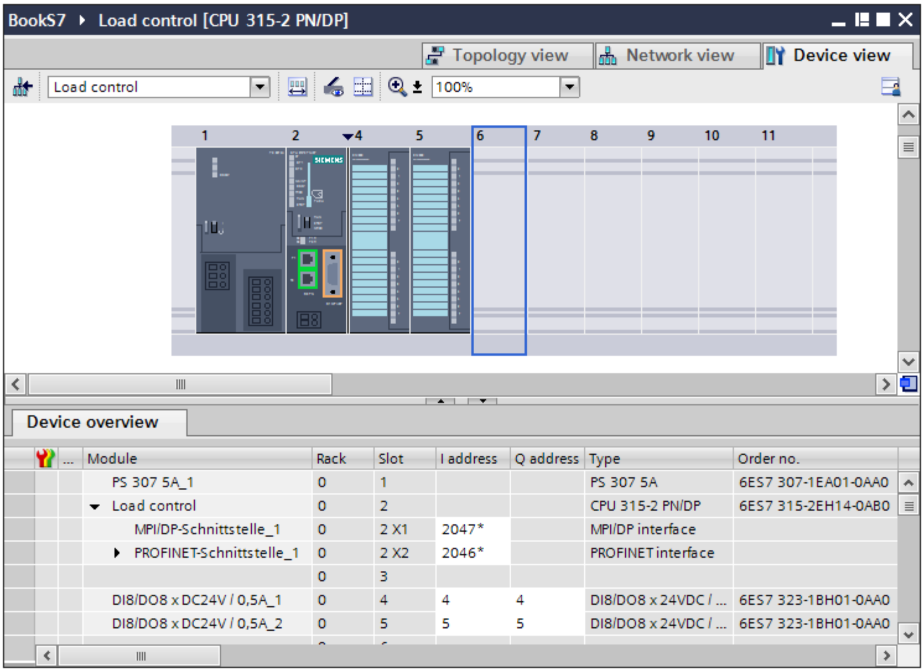
When arranging the modules, STEP 7 automatically assigns a module start address. You can see this address in the configuration table in the bottom part of the station window or in the object properties of the respective module. That is the address of the peripheral inputs/outputs. Further details on address areas can be found in chapter 4.8 "Global address areas" on page 143. Addressing is described in chapter 4.9 "Absolute and symbolic addressing" on page 149.

When assigning the module start address, you can also make the assignment to a process image partition depending on the CPU used. If more than one CPU is inserted in the central rack of an S7-400, multiprocessor mode is automatically set and you must assign the module to a CPU.

Choose *View* > *Address Overview* to open a window that shows all module addresses used for the selected CPU.

Arranging the modules using STEP 7 inside TIA Portal

You start HW Config by double-clicking on *Device configuration* in the project tree underneath the PLC station. After opening, HW Config displays the working window and the hardware catalog. In the *Device View* tab in the upper part of the rack, the working window shows the racks with the slot assignment, and the lower part shows the selected rack as a configuration table (Fig. 3.9).



Module	Rack	Slot	I address	Q address	Type	Order no.
PS 307 5A_1	0	1			PS 307 5A	6ES7 307-1EA01-0AA0
Load control	0	2			CPU 315-2 PN/DP	6ES7 315-2EH14-0AB0
MPI/DP-Schnittstelle_1	0	2 X1	2047*		MPI/DP interface	
PROFINET-Schnittstelle_1	0	2 X2	2046*		PROFINET interface	
	0	3				
DI8/DO8 x DC24V / 0,5A_1	0	4	4	4	DI8/DO8 x 24VDC / ...	6ES7 323-1BH01-0AA0
DI8/DO8 x DC24V / 0,5A_2	0	5	5	5	DI8/DO8 x 24VDC / ...	6ES7 323-1BH01-0AA0

Fig. 3.9 Example of a HW Config working window inside TIA Portal

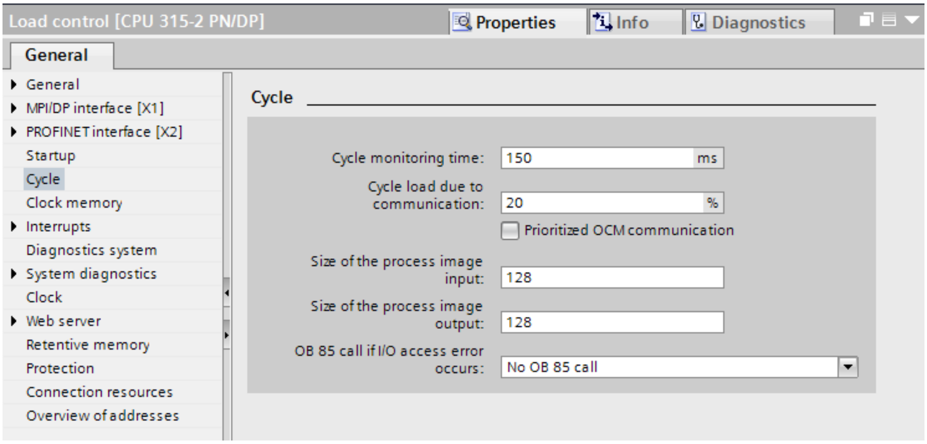
The hardware catalog contains all available racks, modules, and interface modules that STEP 7 recognizes. If you check the *Filter* checkbox, only the objects are shown that can be placed in the current context.

Now you can select modules from the hardware catalog and, keeping the left mouse button pressed, drag them to the slot in the rack. Drag additional racks from the hardware catalog to an empty space in the working window.

You save the configured PLC station by saving the entire project: Choose *Project* > *Save* from the main menu or click on the *Save Project* icon in the toolbar.

Parameterizing and addressing the modules using STEP 7 inside TIA Portal

You define the properties of a module by assigning parameters to it. Assigning parameters is only necessary if you want to change the default parameters. A prerequisite for assigning parameters is that you arrange the module in the rack. Double-click the module in the rack or select the module and choose *Edit* > *Properties*. In the inspector window, the parameterizable module properties are shown in the *Properties* tab. If you assign parameters to a CPU in this manner, you set the execution properties of the user program (Fig. 3.10).



Cycle

Cycle monitoring time: 150 ms

Cycle load due to communication: 20 %

□ Prioritized OCM communication

Size of the process image input: 128

Size of the process image output: 128

OB 85 call if I/O access error occurs: No OB 85 call

Fig. 3.10 Parameterizing modules inside TIA Portal; example for a CPU

When arranging the modules, STEP 7 automatically assigns a module start address. You can see this address in the configuration table in the bottom part of the working window or in the properties of the respective module. That is the address of the peripheral inputs/outputs. Further details on address areas can be found in chapter 4.8 "Global address areas" on page 143. Addressing is described in chapter 4.9 "Absolute and symbolic addressing" on page 149.

When assigning the module start address, you can also make the assignment to a process image partition depending on the CPU used. Under *Overview of addresses* in the rack properties, all module addresses in use for the selected CPU are shown.

3.6 Tools for programming

The user program consists of individual sections known as “blocks”. You can read about what blocks are and what kinds of blocks there are in chapter 5.16 "Overview of user blocks" on page 193 and the following. In these blocks, you program the control functions such as AND, the related addresses such as inputs whose signals must be linked, and the output containing the result of this link.

Symbol table and PLC variable table

The addresses can be addressed via their location in memory, for example I 1.0 (bit 0 in byte 1 in the *Inputs* memory area). It is much easier to understand if you give the memory location a name and then address it with this name, such as *Motor ON*. The assignment between memory location (absolute address) and name (symbolic address) is made by STEP 7 V5.5 in the symbol table and by STEP 7 in the variable table in TIA Portal.

Program editor

With the program editor you program the user program block-by-block. You have a choice of several programming languages, as described in chapter 4 "The programming languages" on page 119. You can select the programming language to best implement the control function for each block. You install SCL and GRAPH in STEP 7 V5.5 as an option package for the SIMATIC Manager.

For *direct program input* (incremental program input), enter the program in steps directly into the block. The STEP 7 V5.5 program editor expects an error-free, syntactically correct program before it will save the block and simultaneously compile it in a form that can be read by the CPU. The STEP 7 program editor inside TIA Portal also allows a block to be saved if the program contains errors. The compilation process is started separately.

You can use *source-oriented program input* for blocks with the programming languages STL and SCL. Here, the user program is written – likewise in blocks – with any text editor and then imported into STEP 7 and compiled.

Help in program creation

Additional tools simplify the creation of the user program. The *cross-reference list* shows the points in the user program where addresss and blocks are used. The *assignment list* shows the inputs, outputs, and bit memories in use. The *call structure* describes the call hierarchy of the blocks in the user program, the *dependency structure* shows each block's dependency on other blocks.

3.7 Giving the addresses a name

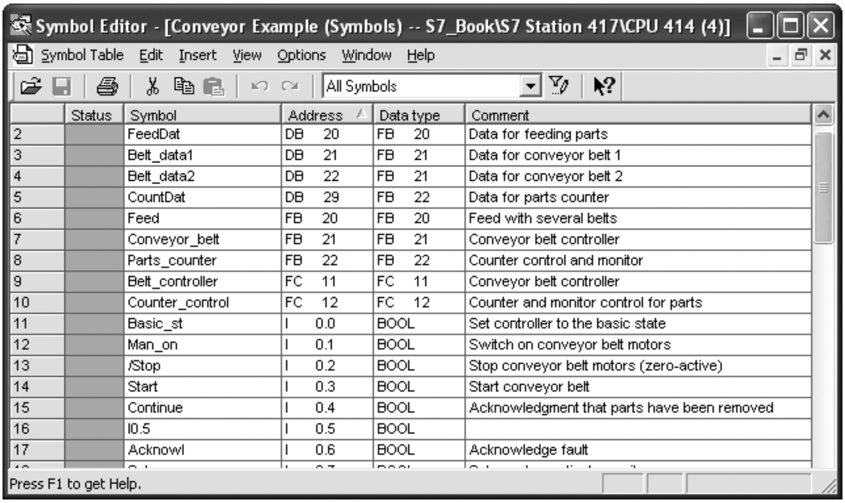
The control program links signal states or calculates with values that can be addressed via their memory location (absolute address) or via a name (symbolic address). A channel from an analog input module, for example, occupies word 12 in the memory area of the inputs. The absolute address is thus “IW 12”. You can assign this input word a name such as “temperature”, and then address the analog channel with this symbolic address. Further information about addresses and their addressing can be found in chapters 4.8 "Global address areas" on page 143 and 4.9 "Absolute and symbolic addressing" on page 149.

The definition of a symbol also includes the data type. It defines certain properties of the data associated with the symbol, basically the representation of the data content. For example, the BOOL data type describes a binary variable and the INT data type describes a digital variable, the contents of which are represented by a 16-bit integer. The possible data types for STEP 7 are described in chapter 4.11 "Elementary data types" on page 153 and the following chapters.

In symbolic addressing, we distinguish between *local* symbols and *global* symbols. A local symbol is only known in the block in which it was defined. You can use the same local symbols in various blocks for different purposes. A global symbol is known throughout the entire user program and has the same meaning in all blocks. You define global symbols with STEP 7 V5.5 in the symbol table and with STEP 7 inside TIA Portal in the variable table.

Working with the symbol table in STEP 7 V5.5

When adding a CPU to a SIMATIC station, the SIMATIC Manager also creates the symbol table. To edit the symbol table, in the SIMATIC Manager navigation, open the *CPU* folder and then the *S7 Program* folder. In the right-hand part of the win-


Symbol Editor - [Conveyor Example (Symbols) -- S7_Book\S7 Station 417\CPU 414 (4)]
Symbol Table
Edit
Insert
View
Options
Window
Help
All Symbols
Press F1 to get Help.

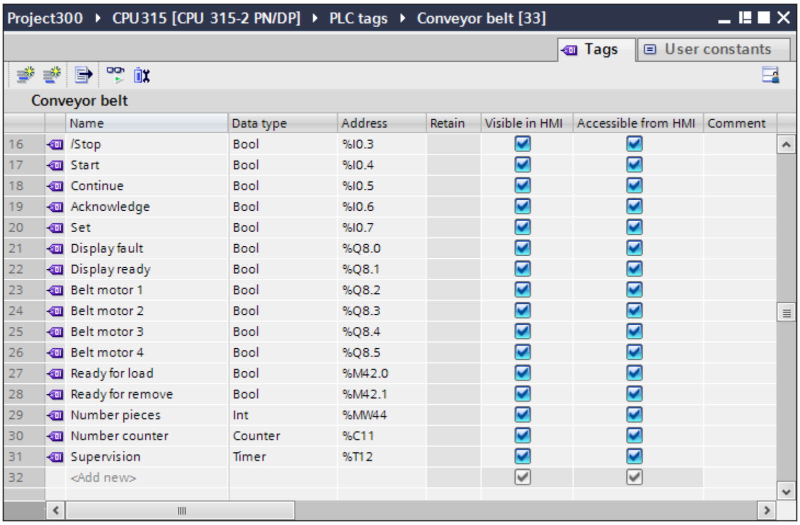
	Status	Symbol	Address	Data type	Comment
2		FeedDat	DB 20	FB 20	Data for feeding parts
3		Belt_data1	DB 21	FB 21	Data for conveyor belt 1
4		Belt_data2	DB 22	FB 21	Data for conveyor belt 2
5		CountDat	DB 29	FB 22	Data for parts counter
6		Feed	FB 20	FB 20	Feed with several belts
7		Conveyor_belt	FB 21	FB 21	Conveyor belt controller
8		Parts_counter	FB 22	FB 22	Counter control and monitor
9		Belt_controller	FC 11	FC 11	Conveyor belt controller
10		Counter_control	FC 12	FC 12	Counter and monitor control for parts
11		Basic_st	I 0.0	BOOL	Set controller to the basic state
12		Man_on	I 0.1	BOOL	Switch on conveyor belt motors
13		/Stop	I 0.2	BOOL	Stop conveyor belt motors (zero-active)
14		Start	I 0.3	BOOL	Start conveyor belt
15		Continue	I 0.4	BOOL	Acknowledgment that parts have been removed
16		I0.5	I 0.5	BOOL	
17		Acknowl	I 0.6	BOOL	Acknowledge fault

Fig. 3.11 Example of a symbol table for STEP 7 V5.5

dow, double-click on the *Symbols* object and open the symbol table. Now you can assign the desired names to the absolute addresses (Fig. 3.11). There can always only be one symbol table under a CPU.

Working with the variable table for STEP 7 inside TIA Portal

When creating a PLC station, STEP 7 also creates a *PLC variables* folder with the *standard variable table*. You open the variable table by double-clicking. In the *Tags* tab, you can now assign the desired names to the absolute addresses (Fig. 3.12). There can only ever be one (complete) variable table under a PLC station. However, you can structure this into several (partial) tables and thus improve the overview in very extensive assignments. Double-clicking on *Show All Tags* shows all implemented assignments.


Project300 ▸ CPU315 [CPU 315-2 PN/DP] ▸ PLC tags ▸ Conveyor belt [33]
Tags
User constants

Conveyor belt

	Name	Data type	Address	Retain	Visible in HMI	Accessible from HMI	Comment
16	/Stop	Bool	%I0.3		☑	☑	
17	Start	Bool	%I0.4		☑	☑	
18	Continue	Bool	%I0.5		☑	☑	
19	Acknowledge	Bool	%I0.6		☑	☑	
20	Set	Bool	%I0.7		☑	☑	
21	Display fault	Bool	%Q8.0		☑	☑	
22	Display ready	Bool	%Q8.1		☑	☑	
23	Belt motor 1	Bool	%Q8.2		☑	☑	
24	Belt motor 2	Bool	%Q8.3		☑	☑	
25	Belt motor 3	Bool	%Q8.4		☑	☑	
26	Belt motor 4	Bool	%Q8.5		☑	☑	
27	Ready for load	Bool	%M42.0		☑	☑	
28	Ready for remove	Bool	%M42.1		☑	☑	
29	Number pieces	Int	%MW44		☑	☑	
30	Number counter	Counter	%C11		☑	☑	
31	Supervision	Timer	%T12		☑	☑	
32	<Add new>				☑	☑	

Fig. 3.12 Example of a PLC variable table for STEP 7 inside TIA Portal

3.8 Programming a logic block

Organization blocks, function blocks, and functions contain the user program code. An organization block is the "start block" for a program type (for a "priority level"): For a start event – startup, main program, interrupt handler, or error program – the CPU's operating system calls the relevant organization block. The program in this organization block can be structured by function blocks and/or functions. Each logic block consists of a block interface and an instruction part. The block-specific variables (temporary and static local variables and block parameters) are declared in the block interface; the block program is in the instruction part. Further details can be found in chapter 5.16 "Overview of user blocks" on page 193 and the following.

General procedure when programming a logic block

To program a block, add a module to the user program, give it a name and possibly a different number, specify the programming language, and open it.

To the extent that the local block variables have already been defined, enter them into the block interface. The block interface can be amended or modified at any time when the block program is created.

When the block program is created, first place the program elements in the graphical languages by dragging them with the mouse from the Program Elements catalog to the working area. The program elements, such as contacts or function boxes, are then supplied with the necessary variables. In the text languages, you write the statements line by line. Complex functions are also retrieved from the Program Elements catalog by dragging them with the mouse.

If the program input is completed, save the block.

Programming a logic block using STEP 7 V5.5

You begin programming a block by opening it, either with a double-click on the block in the project window of SIMATIC Manager or with *File* > *Open* in the program editor. If the block does not exist, generate it

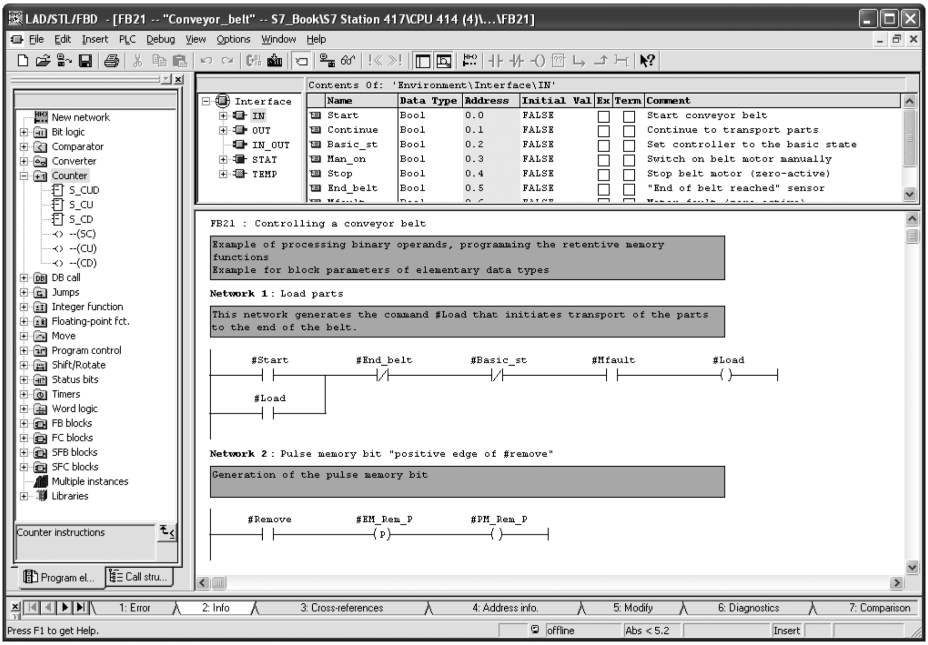
- either in SIMATIC Manager: In the left pane of the project window, select the *Blocks* folder and generate a new block (more precisely: a new block object that you then open for editing) with *Insert > S7 Block >* ...
- or in the program editor: Choose *File > New* to open a dialog in which you specify the desired block under *Object Name*.

The program editor starts when the block is opened. It shows the block interface with the block parameters and local data in the upper part of the working window and the block program in the lower part of the working window (Fig. 3.13).

You define the block-specific variables in the block interface. These are variables that you work with in the block: the temporary and the static local data as well as the block parameters, divided according to input, output and in-out parameters. Not every type of variable can be programmed in every logic block (see table 5.12 "Variable types in the block interface" on page 199). If you do not use a type of variable, the respective declaration subsection remains empty.

In the working window, you can see – depending on the default setting of the program editor – the fields for the block title and block comment, as well as for the first network the fields for the network title, network comment, and the field for the program input. In the program section of a logic block, you control the display of comments and symbols with the menu options *View* > *Display*.... Choose *View* > *Zoom In*, *View* > *Zoom Out* and *View* > *Zoom Factor* to change the size of the display. You define the size of the graphic in the Editor Properties.

An LAD/FBD program is divided into networks that each represent a current path or a link. The division into networks is optional in an STL program; an SCL pro-



Contents Of: 'Environment\Interface\IN'

Name	Data Type	Address	Initial Val	Ex	Term	Comment
Start	Bool	0.0	FALSE	□	□	Start conveyor belt
Continue	Bool	0.1	FALSE	□	□	Continue to transport parts
Basic_st	Bool	0.2	FALSE	□	□	Set controller to the basic state
Man_on	Bool	0.3	FALSE	□	□	Switch on belt motor manually
Stop	Bool	0.4	FALSE	□	□	Stop belt motor (zero-active)
End_belt	Bool	0.5	FALSE	□	□	"End of belt reached" sensor

FB21 : Controlling a conveyor belt

Example of processing binary operands, programming the retentive memory functions
Example for block parameters of elementary data types

Network 1 : Load parts

This network generates the command #Load that initiates transport of the parts to the end of the belt.

Network 2 : Pulse memory bit "positive edge of #remove"

Generation of the pulse memory bit

Fig. 3.13 Example of the program editor's working window in STEP 7 V5.5

gram does not use networks. The program editor automatically numbers the networks starting from 1. You can assign a title and a comment to each network. During editing you can directly choose each network with *Edit* > *Go To* >.... Choose *Insert* > *Network* to program a new network. The program editor then inserts an empty network after the currently selected network.

Input of the program depends on the programming language. In the graphic languages LAD and FBD, you select program elements such as contacts or boxes from the Program Elements catalog and place them in the working window, either in a current path (LAD) or at the input or output of a box (FBD). For STL and SCL, you enter the program line by line, statement by statement.

If the Program Elements catalog is not visible, you can display it on the screen by choosing *View* > *Overviews*. The Program Elements catalog is found in its own window, which you can “attach” to the edge of the Editor window and also release again (each time by double-clicking on the title bar of the catalog window).

Enter the required addresses and variables either with the absolute address or with the symbolic address. You can use a global variable with its symbolic address if it is assigned to an address in the symbol table. A local variable must be declared in the block interface. Variables that are not available can be defined during program entry and used immediately.

The program editor of STEP 7 V5.5 checks the input immediately for correct format (syntax). Only an error-free block can be stored. When saving, the block is immediately compiled into a code that the CPU can read.

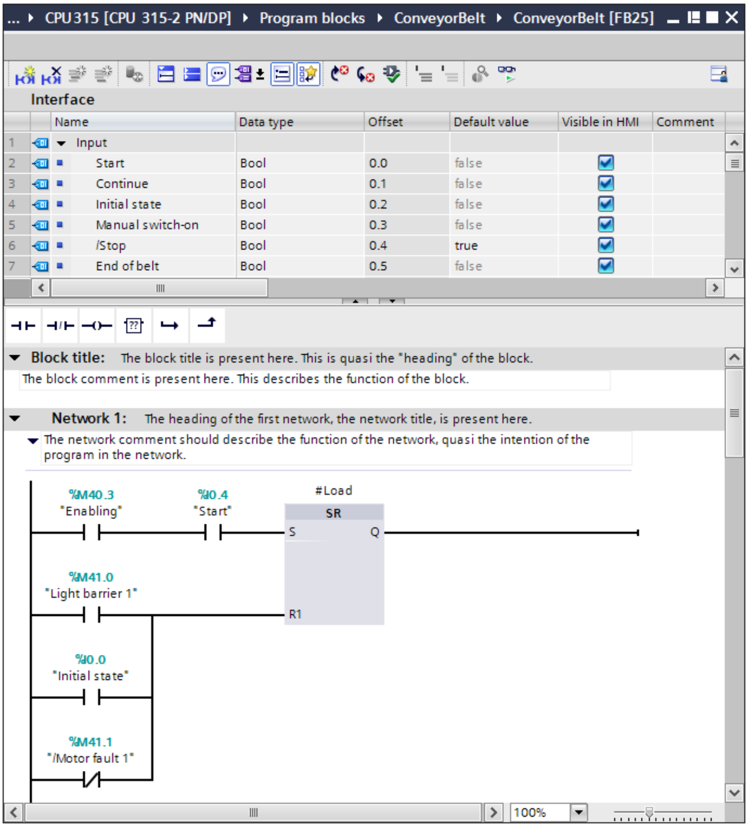
You can assign each logic block in the symbol table a name: Open the symbol table and assign a symbol name to the logic block (OB n, FB n, FC n). The absolute address (OB n, n FB, FC, n) of a logic block is in the *Data type* column.

Programming a logic block using STEP 7 inside TIA Portal

The blocks of the user program are in the project tree in the *Program Blocks* folder under the PLC station. To program a block, open it by double-clicking. If the block does not exist, generate it by double-clicking on *Add new block*.

In the dialog that opens, select the type of the block and give it a name. You can also select the programming language and assign the block number individually. After the dialog is closed, the program editor starts. The program editor shows the block interface with the block parameters and local data in the upper part of the working window and the block program in the lower part of the working window (Fig. 3.14).

You define the block-specific variables in the block interface. These are variables that you work with in the block: the temporary and the static local data as well as



... ▸ CPU315 [CPU 315-2 PN/DP] ▸ Program blocks ▸ ConveyorBelt ▸ ConveyorBelt [FB25]

Interface

	Name	Data type	Offset	Default value	Visible in HMI	Comment
1	▾ Input					
2	Start	Bool	0.0	false	☑	
3	Continue	Bool	0.1	false	☑	
4	Initial state	Bool	0.2	false	☑	
5	Manual switch-on	Bool	0.3	false	☑	
6	/Stop	Bool	0.4	true	☑	
7	End of belt	Bool	0.5	false	☑	

Block title: The block title is present here. This is quasi the "heading" of the block.

The block comment is present here. This describes the function of the block.

Network 1: The heading of the first network, the network title, is present here.

The network comment should describe the function of the network, quasi the intention of the program in the network.

Fig. 3.14 Example of the program editor's working window in STEP 7 inside TIA Portal

the block parameters, divided according to input, output and in-out parameters. Not every type of variable can be programmed in every logic block (see Table 5.12 "Variable types in the block interface" on page 199). If you do not use a type of variable, the respective table section remains empty.

In the working window, you can see – depending on the default setting of the program editor – the fields for the block title and block comment, as well as for the first network the fields for the network title, network comment, and the field for the program input. Click the icons in the function bar of the working window to control the display of comments and addresses as well as the size of the display.

An LAD/FBD program is divided into networks that each represent a current path or a link. The division into networks is optional in an STL program; an SCL program does not use networks. The program editor automatically numbers the networks starting from 1. You can assign a title and a comment to each network. During editing you can directly choose each network with *Edit > Go To >...*. Choose *Insert > Network* to program a new network. The program editor then inserts an empty network after the currently selected network.

Input of the program depends on the programming language. In the graphic languages LAD and FBD, you select program elements such as contacts or boxes from the Program Elements catalog and place them in the working window, either in a current path (LAD) or at the input or output of a box (FBD). For STL and SCL, you enter the program line by line, statement by statement.

If the Program Elements catalog is not visible, you can display it on the screen by opening a block and choosing*View > Task Card*. The Program Elements catalog is found in the task window, which you can “attach” to the right edge of the working window and also release again.

Enter the required addresses and variables either with the absolute address or with the symbolic address. You can use a global variable with its symbolic address if it is assigned to an address in the PLC variable table. A local variable must be declared in the block interface. Variables that are not available can be defined during program entry and used immediately.

The program editor of STEP 7 inside TIA Portal allows you to save a block at any time, even if it is still incomplete or contains syntax errors. A block must only be error-free when it is compiled in the CPU-readable code. You save a block by saving the entire project.

3.9 Programming a data block

Data blocks contain the data of the user program. You can create a data block in several ways: As a global data block that contains the data variables in a freely selectable structure; as a type data block whose data structure is based on a data type; and as an instance data block that contains the data of a function block or system function block. Further details can be found in chapter 5.16 "Overview of user blocks" on page 193 and the following.

General procedure when programming a data block

To program a data block, add it to the user program, give it a name and possibly a different number, and specify the type of data block.

With a global data block, now enter the data variable with a name and data type in the order you desire. If the data block is based on a data type or a function block, the data type or the function module must already be available. Any number of data blocks can be based on a specific data type and for each call of a function block (for each application of the function block) you can create a separate instance data block.

The data variables in a data block are initialized with a *default value* as standard. With a global data block, this is the default value of the data type of the variable; with a type or instance data block, the value entered in the data type or function block is accepted as the default value.

The data variables have a second value, the *initial value*. This is the value that is transmitted online to the CPU and with which the user program begins to work. The default value is the initial value as standard. However, you can also specify an individual initial value and thus use different initial values to differentiate multiple data blocks that are based on a data type or a function block. The value that a data variable in the user memory of the CPU has and can be changed by the user program is called the *actual value*.

Programming a data block using STEP 7 V5.5

You begin programming a block by opening it, either with a double-click on the block in the project window of SIMATIC Manager or with *File* > *Open* in the program editor. If the block does not exist, generate it

- either in SIMATIC Manager: In the left pane of the project window, select the *Blocks* folder and generate a new block (more accurately: a block object that you then open for editing) with *Insert* > *S7 Block* > *Data Block*
- or in the program editor: Choose *File* > *New* to open a dialog in which you enter the desired block under *Object name*.

When inserting a new data block, you specify the type of the data block. Click one of the three following options to select it:

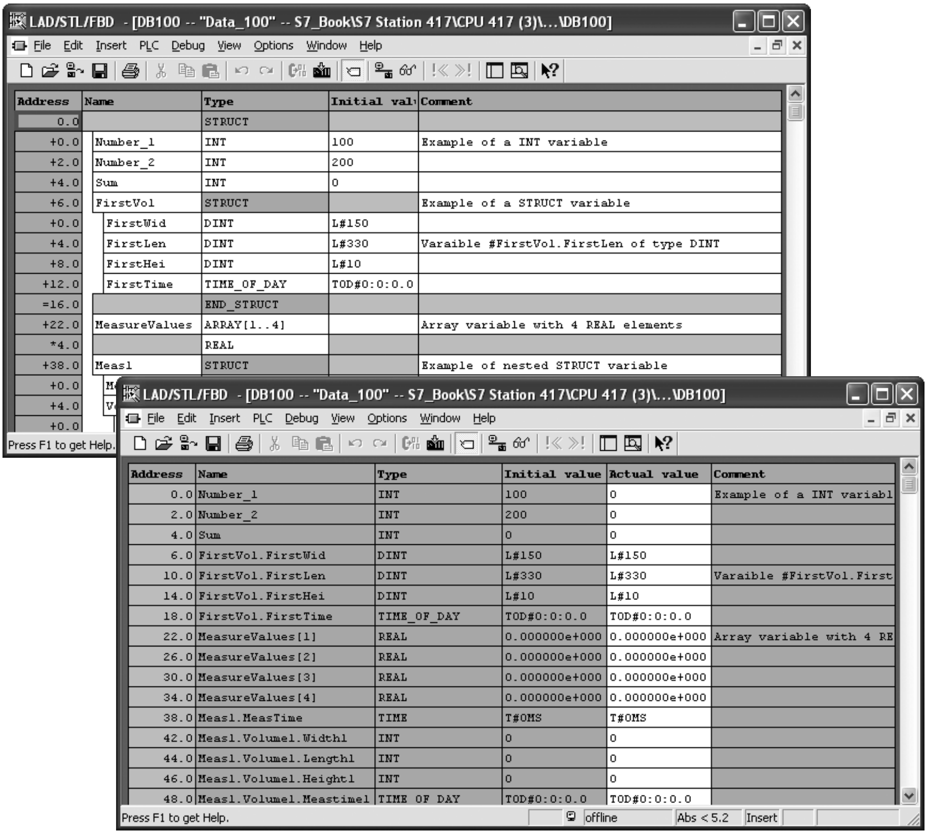
- *Data block* or *Global DB*
  Create as global data block; with this you declare the data addresses when programming the data block
- *Data block with assigned user-defined data type* or *DB of the type*
  Create as data block with user-defined data type; with this you declare the data structure as user-defined data type UDT
- *Data block with assigned function block* or *instance DB*
  Create as instance data block; with this, the data structure is used that you declared when you programmed the associated function block.

With a global data block, now enter the data variable with a name and data type in the order you desire. The default value of the variables is in the *Initial Value* column. Data blocks based on a data type or a function block already have a specified variable structure that can no longer be changed in the data block. However, an individual initial value can be specified for each data variable.

The Program Editor displays the contents of a data block in two views:

- ▷ In the *Declaration View* (*View* > *Declaration View*) you define the data addresses and you see the variables just as you have defined them, e.g. an array or a user-defined data type as one single variable.
- ▷ In the *Data View* (*View* > *Data View*), the program editor displays every variable and every component of an array or a structure individually. Now you can see an additional *Actual Value* column. Here you can enter the initial value with which the user program begins. In the online view, in this column you can observe the actual value which the variable adopts during runtime.

Both views are compared in Fig. 3.15.


LAD/STL/FBD - [DB100 -- "Data_100" -- S7_Book\S7 Station 417\CPU 417 (3)\...\DB100]
File Edit Insert PLC Debug View Options Window Help
Press F1 to get Help.

Address	Name	Type	Initial val	Comment
0.0		STRUCT		
+0.0	Number_1	INT	100	Example of a INT variable
+2.0	Number_2	INT	200	
+4.0	Sum	INT	0	
+6.0	FirstVol	STRUCT		Example of a STRUCT variable
+0.0	FirstWid	DINT	L#150	
+4.0	FirstLen	DINT	L#330	Varaible #FirstVol.FirstLen of type DINT
+8.0	FirstHei	DINT	L#10	
+12.0	FirstTime	TIME_OF_DAY	TOD#0:0:0.0	
=16.0		END_STRUCT		
+22.0	MeasureValues	ARRAY[1..4]		Array variable with 4 REAL elements
*4.0		REAL		
+38.0	Meas1	STRUCT		Example of nested STRUCT variable
+0.0	M			
+4.0	V			
+0.0				

LAD/STL/FBD - [DB100 -- "Data_100" -- S7_Book\S7 Station 417\CPU 417 (3)\...\DB100]
File Edit Insert PLC Debug View Options Window Help
Press F1 to get Help.
offline
Abs < 5.2
Insert

Address	Name	Type	Initial value	Actual value	Comment
0.0	Number_1	INT	100	0	Example of a INT variabl
2.0	Number_2	INT	200	0	
4.0	Sum	INT	0	0	
6.0	FirstVol.FirstWid	DINT	L#150	L#150	
10.0	FirstVol.FirstLen	DINT	L#330	L#330	Varaible #FirstVol.First
14.0	FirstVol.FirstHei	DINT	L#10	L#10	
18.0	FirstVol.FirstTime	TIME_OF_DAY	TOD#0:0:0.0	TOD#0:0:0.0	
22.0	MeasureValues[1]	REAL	0.000000e+000	0.000000e+000	Array variable with 4 RE
26.0	MeasureValues[2]	REAL	0.000000e+000	0.000000e+000	
30.0	MeasureValues[3]	REAL	0.000000e+000	0.000000e+000	
34.0	MeasureValues[4]	REAL	0.000000e+000	0.000000e+000	
38.0	Meas1.MeasTime	TIME	T#0MS	T#0MS	
42.0	Meas1.Volume1.Width1	INT	0	0	
44.0	Meas1.Volume1.Length1	INT	0	0	
46.0	Meas1.Volume1.Height1	INT	0	0	
48.0	Meas1.Volume1.Meastime1	TIME_OF_DAY	TOD#0:0:0.0	TOD#0:0:0.0	

Fig. 3.15 Example of the declaration view and the data view in STEP 7 V5.5

You can assign each data block in the symbol table a name: Open the symbol table and assign a symbol name to the data block DB n. In the *Data Type* column, the absolute address DB n stands for a global data block; the underlying function block or system block stands for an instance data block; and the underlying data type stands for a type data block.

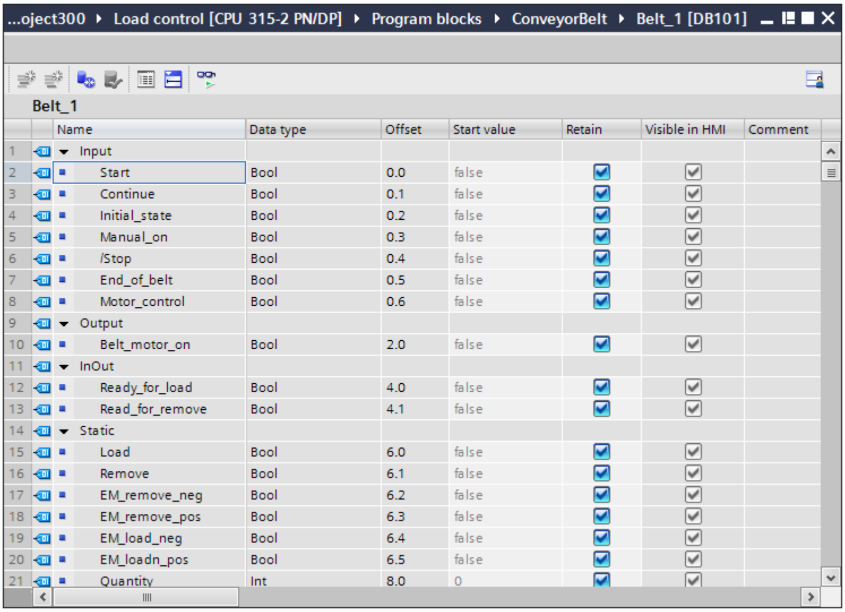
Programming a data block using STEP 7 inside TIA Portal

The blocks of the user program are in the project tree in the *Program Blocks* folder under the PLC station. To program a block, open it by double-clicking. If the block does not exist, generate it by double-clicking on *Add new block*.

In the dialog windows that appears, click the icon for the data block and give the block a name and possibly a different number. Select the type of the data block from a drop-down list containing the entry *Global DB* and the already programmed data types and function blocks (including system function blocks).

After the dialog window closes, the program editor starts and shows the contents of the data block in the working window. With a global data block, now enter the data variables with a name, data type, and possibly initial value and comment, in the order you desire. For a type and instance data block, the data structure is shown but cannot be changed. Only an individual preassignment with initial values is possible.

With the *Expanded Mode* icon in the function bar of the working window, you can open variables with complex data types (fields and structures) and thus supply the individual components, for example, with an individual initial value.


...oject300 ▸ Load control [CPU 315-2 PN/DP] ▸ Program blocks ▸ ConveyorBelt ▸ Belt_1 [DB101]

Belt_1

	Name	Data type	Offset	Start value	Retain	Visible in HMI	Comment
1	▼ Input						
2	Start	Bool	0.0	false	☑	☑	
3	Continue	Bool	0.1	false	☑	☑	
4	Initial_state	Bool	0.2	false	☑	☑	
5	Manual_on	Bool	0.3	false	☑	☑	
6	/Stop	Bool	0.4	false	☑	☑	
7	End_of_belt	Bool	0.5	false	☑	☑	
8	Motor_control	Bool	0.6	false	☑	☑	
9	▼ Output						
10	Belt_motor_on	Bool	2.0	false	☑	☑	
11	▼ InOut						
12	Ready_for_load	Bool	4.0	false	☑	☑	
13	Read_for_remove	Bool	4.1	false	☑	☑	
14	▼ Static						
15	Load	Bool	6.0	false	☑	☑	
16	Remove	Bool	6.1	false	☑	☑	
17	EM_remove_neg	Bool	6.2	false	☑	☑	
18	EM_remove_pos	Bool	6.3	false	☑	☑	
19	EM_load_neg	Bool	6.4	false	☑	☑	
20	EM_loadn_pos	Bool	6.5	false	☑	☑	
21	Quantity	Int	8.0	0	☑	☑	

Fig. 3.16 Example of an opened data block in STEP 7 inside TIA Portal

You save a data block by saving the entire project. If you have compiled the data block in a code that can be read by the CPU, the *Offset* column of the program editor shows the data variables' start address which they have in the data block (Fig. 3.16).

3.10 Programming a user-defined data type

A user-defined data type (UDT) is a compilation of any data types into a data structure, similar to the STRUCT data type. You can use a user-defined data type if you want to give a data structure a name, if a data structure is commonly found in the user program, or if you want to set up one or more data blocks with a specific data structure. The available data types in SIMATIC are described in chapter 4.11 "Elementary data types" on page 153 and the following.

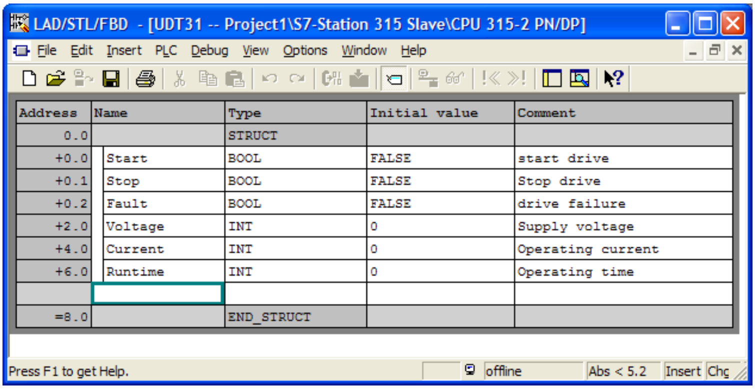
A user-defined data type is globally valid, i.e. once declared, it can be used in all blocks.

General procedure when programming a user-defined data type

You add a user-defined data type to a user program, open it and program it as with a data block. However, the user-defined data type only contains a collection of data types, and no variables (no memory location and thus no address). When programming, enter the names and data types in your preferred sequence and supplement them with an individually specified default value and a comment.

Programming a user-defined data type using STEP 7 V5.5

To program a user-defined data type, open the *Blocks* folder in SIMATIC Manager and use *Insert* > *S7 Block* > *Data Type* to add a user-defined data type (UDT with a number). Double-clicking on the UDT object in the working window opens a declaration table in which you enter the individual (future) variables with name, data type, default value (in the *Inital Value* column), and comments (Fig. 3.17).



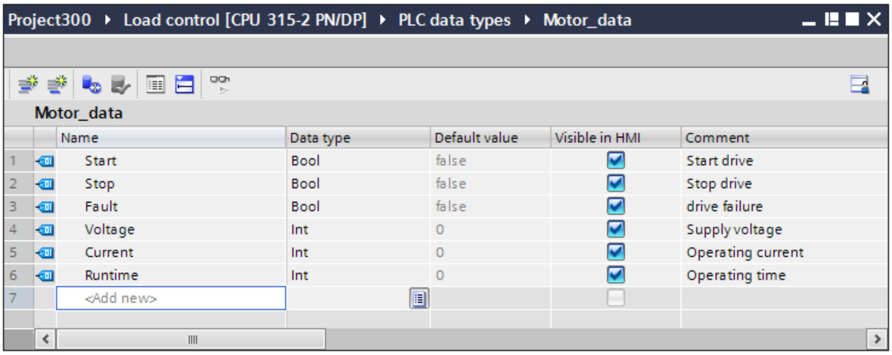
Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	Start	BOOL	FALSE	start drive
+0.1	Stop	BOOL	FALSE	Stop drive
+0.2	Fault	BOOL	FALSE	drive failure
+2.0	Voltage	INT	0	Supply voltage
+4.0	Current	INT	0	Operating current
+6.0	Runtime	INT	0	Operating time
=8.0		END_STRUCT		

Fig. 3.17 Example of a user-defined data type for STEP 7 V5.5

You can also address a user-defined data type symbolically: Open the symbol table and assign a symbol name to the user-defined data type with the absolute address UDT n. The absolute address UDT n is entered in the *Data Type* column.

Programming a user-defined data type using STEP 7 inside TIA Portal

The user-defined data types are in the PLC Data Types folder in the project tree under the PLC station. Double-click on *Add New Data Type* to start the program editor. In the working window it shows the declaration table in which you enter the individual components with name, data type, default value, and comments in the desired order (Fig. 3.18).



Project300 ▸ Load control [CPU 315-2 PN/DP] ▸ PLC data types ▸ Motor_data

Motor_data

	Name	Data type	Default value	Visible in HMI	Comment
1	Start	Bool	false	☑	Start drive
2	Stop	Bool	false	☑	Stop drive
3	Fault	Bool	false	☑	drive failure
4	Voltage	Int	0	☑	Supply voltage
5	Current	Int	0	☑	Operating current
6	Runtime	Int	0	☑	Operating time
7	<Add new>			□	

Fig. 3.18 Example of a user-defined data type for STEP 7 inside TIA Portal

The program editor gives a user-defined data type the name *Userdatatype_n* as standard, with n as a consecutive number. You can change the name in the properties of the user-defined data type: You select the user-defined data type in the *PLC data types* folder, select *Properties* in the shortcut menu, and enter another name under *General*.

3.11 Working with program source files

Blocks with the programming languages STL or SCL can be programmed as a text file with any text editor. These text files are referred to as “source files” or “program source files”. A program source file can contain one or several blocks; these can be logic or data blocks as well as user-defined data types. The assignments of absolute addresses to symbolic addresses (symbol table or PLC variable table) are created separately from the program source file as an Excel table and imported.

In the program source file, specific keywords define the properties of blocks and the program. These keywords should be used in a specific sequence. A logic block begins with the block header in which the block type and the block properties are defined. This is followed by the block interface and the actual program. A keyword for the block end concludes the programming of a block (Table 3.1).

Table 3.1 Keywords for logic blocks

Section	Keyword	Description
Block type	ORGANIZATION_BLOCK "*OB_name*" FUNCTION_BLOCK "*FB_name*" FUNCTION "*FC_name*" : *Data type*	Start of an organization block Start of a function block Start of a function
Header	TITLE = *block title* //Block comment	Block title in the block properties Block comment in the block properties
	CODE_VERSION1 KNOW_HOW_PROTECT	Only with FB ("not with multi-instance capability"), only with STL Know-how protection (cannot be canceled)
	NAME : *Block name* FAMILY : *Block family* AUTHOR : *Created by* VERSION : *Version*	Block property: Block name Block property: Block family Block property: Created by Block property: Block version
Declaration	VAR_INPUT name : Data type := Default setting; *) END_VAR	Input parameter (not with OB)
	VAR_OUTPUT name : Data type := Default setting; *) END_VAR	Output parameter (not with OB)
	VAR_IN_OUT name : Data type := Default setting; *) END_VAR	In/out parameter (not with OB)
	VAR name : Data type := Default setting; *) END_VAR	Static local data (only with FB)
	VAR_TEMP name : Data type := Default setting; *) END_VAR	Temporary local data
Program	BEGIN	Start of block program, can be omitted with SCL
	NETWORK	Network start, only with STL
	TITLE = *Network title*	Network title, only with STL
	//Network comment	Network comment; line comment with SCL
	Program statement;	Termination of each statement with semicolon
	//Line comment	Line comment up to end of line, also programmable following statements
	(* Block comment *)	Block comment, can extend over several lines, only with SCL
	NETWORK	Start of next network, only with STL
	…	… etc.

Block end	END_ORGANIZATION_ BLOCK
	END_FUNCTION_BLOCK
	END_FUNCTION
		End of an organization block
		End of a function block
		End of a function

*) Superimposing of data types with the keyword AT is additionally possible with SCL

A data block begins with the block header in which the block type and the block properties are defined. The data addresses and, if the data block is based on a data type or a function block, the entry of the data type or the function block then follow. A keyword for the block end concludes the programming of a data block (Table 3.2).

Table 3.2 Keywords for data blocks

Section	**Keyword**	**Description**
Block type	DATA_BLOCK "*DB_name*"	Start of a data block
Header	TITLE = *block title* //Block comment	Block title Block comment
	KNOW_HOW_PROTECT UNLINKED READ_ONLY NON_RETAIN	Know-how protection (cannot be canceled) Block attribute: not executable Block attribute: read-only Block attribute: non-retentive
	NAME : *Block name* FAMILY : *Block family* AUTHOR : *Created by* VERSION : *Version*	Block property: Block name Block property: Block family Block property: Created by Block property: Block version

Declaration	STRUCT
	name : Data type := Default setting;
	END_STRUCT
		for a global data block

	Data type_name	alternatively for a type data block
	FB_name	alternatively for an instance data block

Initialization	BEGIN
	name := Default setting;
		
		Assignment with individual initial values

Block end	END_DATA_BLOCK	End of a data block

A user-defined data type begins with a keyword for the beginning of the definition. The components of the data type then follow in the form of a STRUCT data structure. A keyword for the data type end concludes the programming of a user-defined data type (Table 3.3).

Table 3.3 Keywords for user-defined data types

Section	**Keyword**	**Description**
Block type	TYPE "*Type_name*"	Start of a user-defined data type
Header	TITLE = *Data type title* //Data type comment	Data type title Data type comment

Declaration	STRUCT
	name : Data type := Default setting;
	END_STRUCT
		Declaration of data type components

Block end	END_TYPE	End of the user-defined data type

If you use objects in the program source file that themselves are only defined in the program source file – such as call logic blocks or address data variables – you must arrange these objects before their point of use in the program source file. When calling a logic block or a complex function with input and output parameters, list the parameters in parenthesis after the call function, in the specified sequence and each separated by a comma.

Editing program source files with STEP 7 V5.5

To create programs with the SCL programming language, STEP 7 V5.5 requires the S7-SCL option package.

The program source files are saved in the *Sources* folder. The *Sources* folder is created under the *S7 Program* folder when a CPU is added. To add a new program source file, select the *Sources* folder in SIMATIC Manager and select *Insert* > *S7 Software* > *STL Source* or ... > *SCL Source*.

Double-click to open and edit them with the program editor. Choose *Insert* > *Block Template* > ... to facilitate the creation of new blocks. With *Insert* > *Object* > *Block*, the program editor inserts after the cursor an already compiled block as an ASCII source into the source file.

You also have the option of generating a new program source file from one or several compiled blocks using the program editor and the *File* > *Generate Source File* command.

If you have created an STL source file with another text editor, you can fetch it with *Insert* > *External Source File* under SIMATIC Manager into the folder *Source Files*. With *Edit* > *Export* Source you can copy the selected source file to the hard disk in a folder of your choice.

You can save the program source file any time during processing even if the program is not yet complete. Only when the source file is compiled does the program editor generate executable blocks, which it stores in the *Blocks* folder. If you used global symbols in the program source file, then the completed symbol table must also be available during compiling. Blocks that have been called must already be available as compiled blocks in the *Blocks* folder or in the program source file before being called.

Choose *Options* > *Customize* on the *Source Files* tab to set the properties of the compiler, e.g. whether existing blocks are to be overwritten or if blocks are only to be created if the entire program source file is error-free. In the *Block* tab, you can set the automatic updating of the reference data during the compiling of a block.

You can check the syntax of the program source file with *File* > *Consistency Check*, without compiling the blocks.

You start compilation with the program source file open, using the menu option *File* > *Compile*. All error-free blocks that are found in the program source file are compiled. Any erroneous blocks are not compiled. If warnings occur, the block is still compiled; however, execution in the CPU might be faulty.

Editing program source files with STEP 7 inside TIA Portal

The program source files are in the *External Sources* folder in the project tree under the PLC station.

You create the program source file outside the TIA Portal with any text editor. Logic blocks programmed with STL are saved in a text file with the extension .awl. For logic blocks programmed with SCL, the ending is .scl. In both cases, the text file can contain data blocks and user-defined data types.

To import a program source file, open the *External sources* folder in the project tree and double-click on *Add new external source*. In the dialog window, select the type of source (*STL Sources* or *SCL Sources*), navigate to the storage location, select the source file, and import it by clicking on the *Open* button.

You can also edit the source files directly in the *External Sources* folder if you link the file extensions .awl and .scl with a text editor in the Windows operating system.

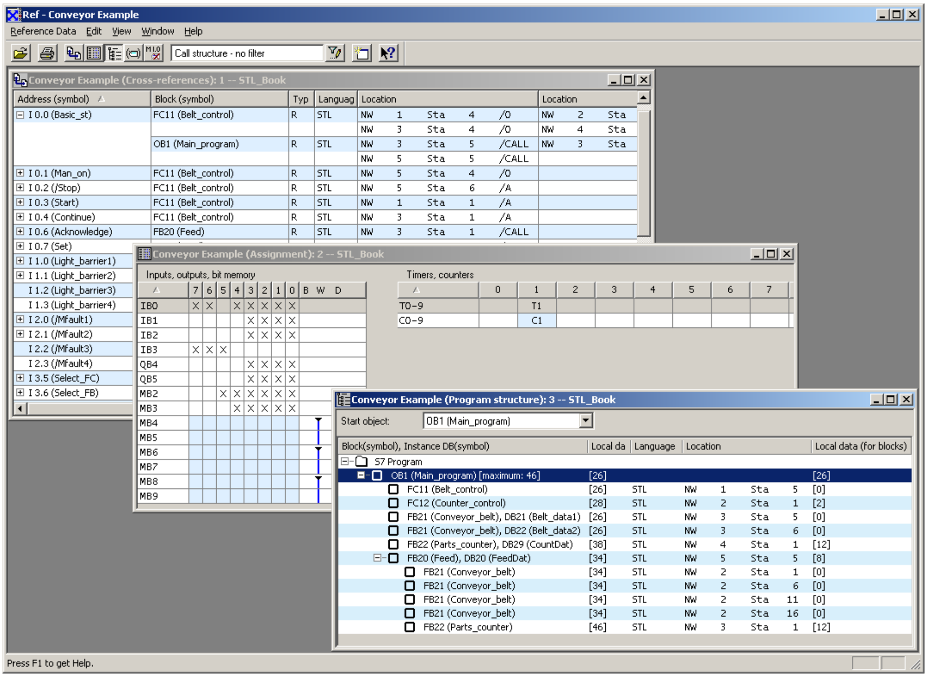
To transfer the blocks from the source file to the *Program Blocks* folder, select the source file in the *External Sources* folder and then the *Generate blocks* command from the shortcut menu. It is recommendable to compile the blocks imported from a source file prior to further processing inside TIA Portal.

3.12 Help on Program Creation

STEP 7 supports program creation with display of reference data that you can use as a basis for corrections or program tests. The tools of the reference data are

- the cross-reference list, which indicates the use of addresses and blocks in the user program
- the assignment list, which displays the assignment of the address areas inputs, outputs, bit memories, SIMATIC timers, and SIMATIC counters by the user program
- the call structure, which shows the structure of the user program based on the programmed organization blocks and the blocks called in them
- the dependency structure, which shows which block is called by which other block.

The reference data are are only shown by the offline data management, even if the function is called in a block opened online. Fig. 3.19 shows the reference data as it is presented by STEP 7 V5.5. Fig. 3.20 presents the call structure of STEP 7 inside TIA Portal.



Ref - Conveyor Example

Reference Data Edit View Window Help

Call structure - no filter

Conveyor Example (Cross-references): 1 -- STL_Book

Address (symbol)	Block (symbol)	Typ	Languag	Location	Location
I 0.0 (Basic_st)	FC11 (Belt_control)	R	STL	NW 1 Sta 4 /O	NW 2 Sta
				NW 3 Sta 4 /O	NW 4 Sta
	OB1 (Main_program)	R	STL	NW 3 Sta 5 /CALL	NW 3 Sta
				NW 5 Sta 5 /CALL	
I 0.1 (Man_on)	FC11 (Belt_control)	R	STL	NW 5 Sta 4 /O	
I 0.2 (/Stop)	FC11 (Belt_control)	R	STL	NW 5 Sta 6 /A	
I 0.3 (Start)	FC11 (Belt_control)	R	STL	NW 1 Sta 1 /A	
I 0.4 (Continue)	FC11 (Belt_control)	R	STL	NW 3 Sta 1 /A	
I 0.6 (Acknowledge)	FB20 (Feed)	R	STL	NW 3 Sta 1 /CALL	
I 0.7 (Set)					
I 1.0 (Light_barrier1)					
I 1.1 (Light_barrier2)					
I 1.2 (Light_barrier3)					
I 1.3 (Light_barrier4)					
I 2.0 (/Mfault1)					
I 2.1 (/Mfault2)					
I 2.2 (/Mfault3)					
I 2.3 (/Mfault4)					
I 3.5 (Select_FC)					
I 3.6 (Select_FB)					

Conveyor Example (Assignment): 2 -- STL_Book

Inputs, outputs, bit memory

	7	6	5	4	3	2	1	0	B W D
IB0	X	X		X	X	X	X	X	
IB1					X	X	X	X	
IB2					X	X	X	X	
IB3	X	X	X						
QB4					X	X	X	X	
QB5					X	X	X	X	
MB2			X	X	X	X	X	X	
MB3				X	X	X	X	X	
MB4									
MB5									
MB6									
MB7									
MB8									
MB9									

Timers, counters

	0	1	2	3	4	5	6	7
T0-9		T1						
C0-9		C1						

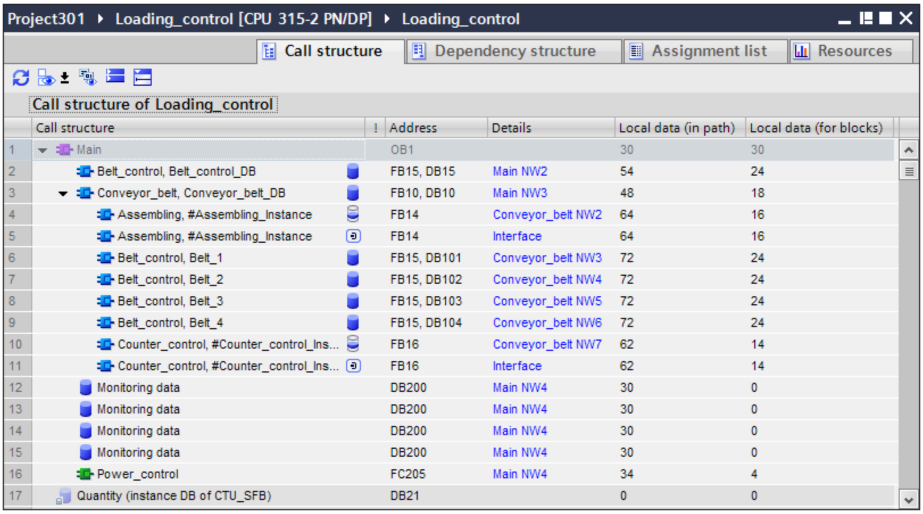
Conveyor Example (Program structure): 3 -- STL_Book

Start object: OB1 (Main_program)

Block(symbol), Instance DB(symbol)	Local da	Language	Location	Local data (for blocks)
S7 Program				
OB1 (Main_program) [maximum: 46]	[26]			[26]
FC11 (Belt_control)	[26]	STL	NW 1 Sta 5	[0]
FC12 (Counter_control)	[28]	STL	NW 2 Sta 1	[2]
FB21 (Conveyor_belt), DB21 (Belt_data1)	[26]	STL	NW 3 Sta 5	[0]
FB21 (Conveyor_belt), DB22 (Belt_data2)	[26]	STL	NW 3 Sta 6	[0]
FB22 (Parts_counter), DB29 (CountDat)	[38]	STL	NW 4 Sta 1	[12]
FB20 (Feed), DB20 (FeedDat)	[34]	STL	NW 5 Sta 5	[8]
FB21 (Conveyor_belt)	[34]	STL	NW 2 Sta 1	[0]
FB21 (Conveyor_belt)	[34]	STL	NW 2 Sta 6	[0]
FB21 (Conveyor_belt)	[34]	STL	NW 2 Sta 11	[0]
FB21 (Conveyor_belt)	[34]	STL	NW 2 Sta 16	[0]
FB22 (Parts_counter)	[46]	STL	NW 3 Sta 1	[12]

Press F1 to get Help.

Fig. 3.19 Example of reference data type for STEP 7 V5.5



Project301 ▸ Loading_control [CPU 315-2 PN/DP] ▸ Loading_control

Call structure of Loading_control

	Call structure	!	Address	Details	Local data (in path)	Local data (for blocks)
1	Main		OB1		30	30
2	Belt_control, Belt_control_DB		FB15, DB15	Main NW2	54	24
3	Conveyor_belt, Conveyor_belt_DB		FB10, DB10	Main NW3	48	18
4	Assembling, #Assembling_Instance		FB14	Conveyor_belt NW2	64	16
5	Assembling, #Assembling_Instance		FB14	Interface	64	16
6	Belt_control, Belt_1		FB15, DB101	Conveyor_belt NW3	72	24
7	Belt_control, Belt_2		FB15, DB102	Conveyor_belt NW4	72	24
8	Belt_control, Belt_3		FB15, DB103	Conveyor_belt NW5	72	24
9	Belt_control, Belt_4		FB15, DB104	Conveyor_belt NW6	72	24
10	Counter_control, #Counter_control_Ins...		FB16	Conveyor_belt NW7	62	14
11	Counter_control, #Counter_control_Ins...		FB16	Interface	62	14
12	Monitoring data		DB200	Main NW4	30	0
13	Monitoring data		DB200	Main NW4	30	0
14	Monitoring data		DB200	Main NW4	30	0
15	Monitoring data		DB200	Main NW4	30	0
16	Power_control		FC205	Main NW4	34	4
17	Quantity (instance DB of CTU_SFB)		DB21		0	0

Fig. 3.20 Example of a call structure for STEP 7 inside TIA Portal

3.13 Downloading the user program to the CPU

You create the hardware configuration and the user program “offline” on the programming device without a connection to a CPU. The entered data is changed into a code that the target CPU can process through “compiling”. The compiled project data must now be transferred to the CPU. This can occur with the help of a memory card or by connecting a programming device. If you connect a programming device and switch to online mode, you can not only exchange project data between the offline data management on the hard drive and the online data in the CPU. With the programming device, you can also control the CPU, test the user program, and find errors in the hardware configuration of the PLC station using diagnostic functions.

Connecting the programming device to the CPU

You can connect the programming device to any bus interface of the CPU: MPI, DP or PN interface. The prerequisite is that the appropriate interface module is plugged into the programming device. Current units are all supplied with a LAN adapter that allows connection to Industrial Ethernet and thus to the PN interface of the CPU.

The connection of a programming device to a CPU is not configured. Only the bus addresses – depending on the connection type, the MPI, PROFIBUS, or IP address – must be set appropriately. You set the bus address with the Hardware Config when parameterizing the CPU.

If the programming device is connected directly to a CPU in a single PLC station, the assignment to the target device is automatic. If there are several PLC stations on a bus system, the programming device can be connected to the bus system at any point and still reach all the PLC stations – even across subnets (“routing”). First, the nodes on the bus system must receive a unique bus address, such as by transferring the configuration data to the CPU. If, when connecting to Industrial Ethernet, a CPU has no IP address, which is the case for brand-new CPUs, they can be identified by their MAC address and their IP address can also be set in the bus network.

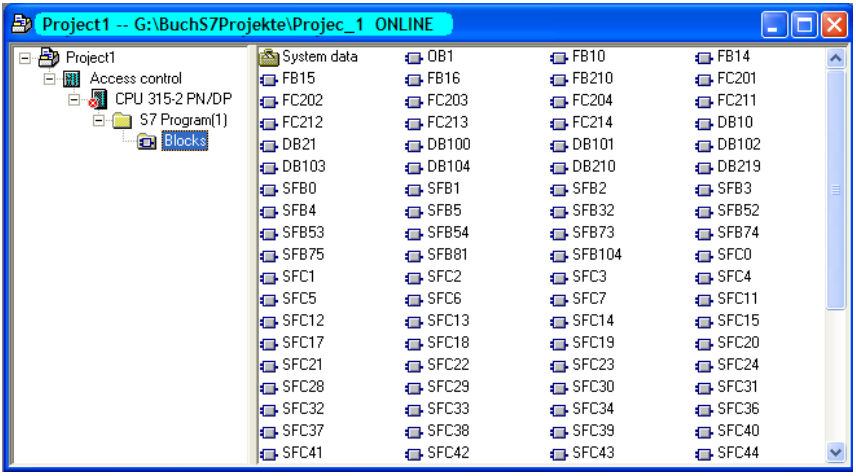
Plugging in the connecting cable is not sufficient to communicate with the CPU. The mechanical connection (“networking”) must be expanded by an agreement on the allowable sequence and meaning of the signals (“connection protocol”). Only when this “connection” is established can the data exchange begin. The programming device is then in online mode.

Establishing a connection between the programming device and the CPU is done largely automatically. The manual activities depend on the current situation. If the assignment to the target CPU is unambiguous, you simply activate online mode. If the PLC station is located on a bus system, you can let the programming device show all available nodes, select the desired node (desired PLC stations), and then use it to commence online operation.

Activating online mode with STEP 7 V5.5

With the command *Target system* > *Display accessible devices* in SIMATIC Manager, all PLC stations can be displayed that the programming device finds on the selected bus connection. If you open a project that is stored on the hard disk, it will appear in the offline window. If the project data has already been loaded to the CPU, you can view it with the command *View* > *Online* command in the online window. In the online window, the heading in the title bar is highlighted and also reads "ONLINE". The *View* > *Offline* command returns to the offline window.

The figure shows an example of an online window. All system blocks available in the CPU are displayed in addition to the user blocks in the online Blocks folder.



Project1 -- G:\BuchS7Projekte\Projec_1 ONLINE

- Project1
  - Access control
    - CPU 315-2 PN/DP
      - S7 Program(1)
        - Blocks

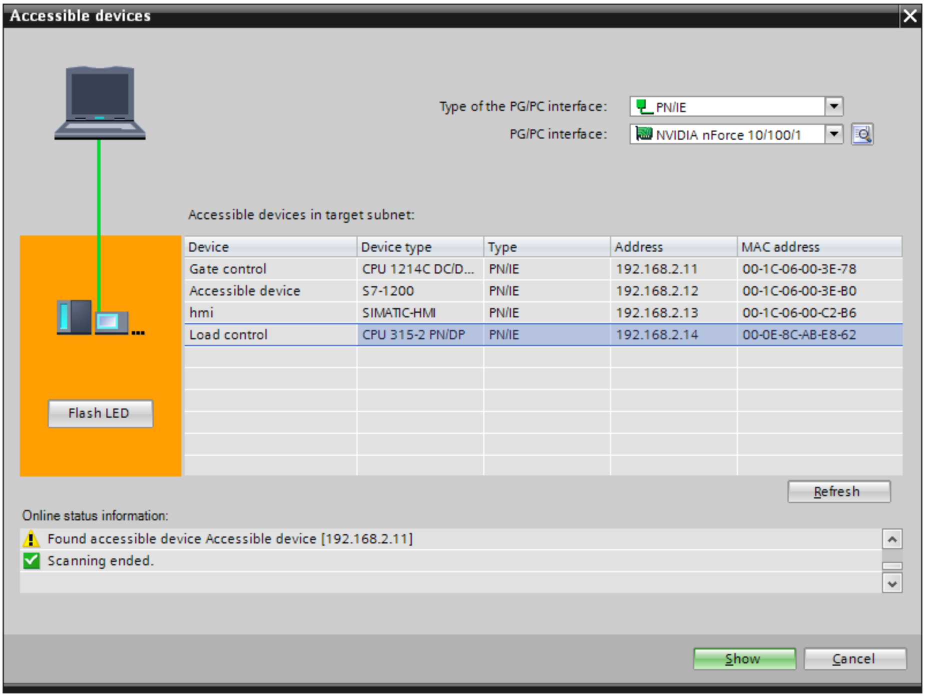
System data	OB1	FB10	FB14
FB15	FB16	FB210	FC201
FC202	FC203	FC204	FC211
FC212	FC213	FC214	DB10
DB21	DB100	DB101	DB102
DB103	DB104	DB210	DB219
SFB0	SFB1	SFB2	SFB3
SFB4	SFB5	SFB32	SFB52
SFB53	SFB54	SFB73	SFB74
SFB75	SFB81	SFB104	SFC0
SFC1	SFC2	SFC3	SFC4
SFC5	SFC6	SFC7	SFC11
SFC12	SFC13	SFC14	SFC15
SFC17	SFC18	SFC19	SFC20
SFC21	SFC22	SFC23	SFC24
SFC28	SFC29	SFC30	SFC31
SFC32	SFC33	SFC34	SFC36
SFC37	SFC38	SFC39	SFC40
SFC41	SFC42	SFC43	SFC44

Fig. 3.21 Example of an online window for STEP 7 V5.5

Activating online mode with STEP 7 inside TIA Portal

With the command *Online* > *Accessible Nodes...* in the main menu bar, all PLC stations are displayed that the programming device finds on the selected bus connection (Fig. 3.22). For further processing in the project view, select the desired PLC station and click the *View* button. Turn on the online mode in the project tree under *Online access* by double-clicking on the desired PLC station at *Online and Diagnosis*.

If the project data in a PLC station has already been loaded to the CPU, you can turn on online mode for the selected PLC station with the *Connect Online* icon. All the windows displaying online data – that is, data from a connected PLC station – have an orange title bar. You can use the *Cancel Online Connection* icon to switch the PLC station to offline mode again.



Type of the PG/PC interface: PN/IE

PG/PC interface: NVIDIA nForce 10/100/1

Accessible devices in target subnet:

Device	Device type	Type	Address	MAC address
Gate control	CPU 1214C DC/D...	PN/IE	192.168.2.11	00-1C-06-00-3E-78
Accessible device	S7-1200	PN/IE	192.168.2.12	00-1C-06-00-3E-B0
hmi	SIMATIC-HMI	PN/IE	192.168.2.13	00-1C-06-00-C2-B6
Load control	CPU 315-2 PN/DP	PN/IE	192.168.2.14	00-0E-8C-AB-E8-62

Online status information:

```
Found accessible device Accessible device [192.168.2.11]
Scanning ended.
```

Fig. 3.22 Example of the “Accessible Nodes” display in STEP 7 inside TIA Portal

Loading project data

For transferring project data to the CPU, you have two options: transfer with a memory card or transfer with a connected programming device.

To transfer using a memory card, which retains data even without power supply, write the compiled project data to the memory card – provided you have the appropriate card programming module on the programming device. Then you insert the memory card into the CPU and turn on the CPU. When starting up, the CPU imports the project data from the memory card into its own data storage.

When transferring using a programming device, connect the programming device to the CPU, activate online mode, and transfer the project data to the CPU.

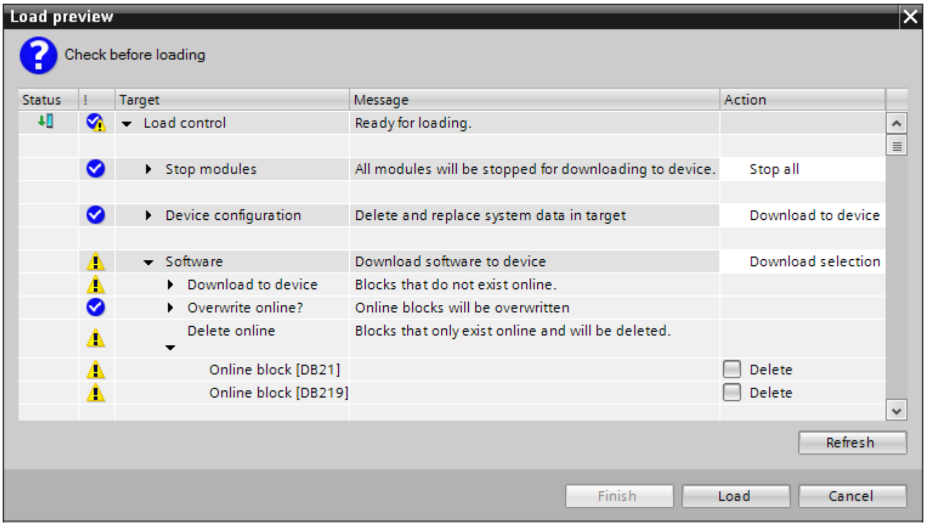
When loading into the CPU unit – whether from a memory card or a programming device – the complete user program is written to the load memory. The “execution-relevant” data – the logic and data blocks – is then transferred to the work memory and executed when you activate RUN mode.

Loading the user program with STEP 7 V5.5

To load the project data, select the SIMATIC station in the SIMATIC Manager and select the command *Target System* > *Load*. If the SIMATIC station cannot be assigned to a node, a table of reachable nodes will be shown, from which you can select the desired node. Only error-free compiled project data is loaded. When the configuration data is loaded, the CPU must be in STOP mode. When loading is complete, the SIMATIC station can be switched to RUN mode.

Loading the user program with STEP 7 inside TIA Portal

To load the project data, select the PLC station in the project tree and then select the *Download to the device* > *All* command from the shortcut menu. If the PLC station cannot be assigned to a node, a table of reachable nodes will be shown, from which you can select the desired node. Before loading, the project data is compiled. Only project data that has been compiled without errors can be loaded. STEP 7 shows the further loading procedure in a preview window (Fig. 3.23). Select the desired action in the *Action* column and click on the *Load* button. When the configuration data is loaded, the CPU must be in STOP mode. When loading is complete, the PLC station can be switched to RUN mode.



Load preview

Check before loading

Status	!	Target	Message	Action
⬇	⚠	▾ Load control	Ready for loading.	
	✓	▸ Stop modules	All modules will be stopped for downloading to device.	Stop all
	✓	▸ Device configuration	Delete and replace system data in target	Download to device
	⚠	▾ Software	Download software to device	Download selection
	⚠	▸ Download to device	Blocks that do not exist online.	
	✓	▸ Overwrite online?	Online blocks will be overwritten	
	⚠	Delete online ▾	Blocks that only exist online and will be deleted.	
	⚠	Online block [DB21]		□ Delete
	⚠	Online block [DB219]		□ Delete

Fig. 3.23 Example of the preview when loading the project data in STEP 7 inside TIA Portal

3.14 Processing the user program online

With a programming device is connected to the CPU, you can edit both the user program offline on the hard drive and online in the CPU, for example adding, changing, deleting, or comparing blocks.

Editing individual blocks offline/online with STEP 7 V5.5

The STEP 7 V5.5 program editor provides two working windows for block processing: the offline window and the online window. The offline window shows the block that is stored on the hard disk, and the online window shows the block that is in the user memory of the CPU. To differentiate, the heading in the title bar in the online window is highlighted and also reads “ONLINE”. Use the *Offline/Online Partners* icon in the program editor toolbar to switch between two windows.

Transfer an open block to the hard disk with the *File* > *Save* command and to the CPU with *Target System* > *Load*. However, you should ensure that the offline and online data storage do not diverge. A change in an online block that you have made for testing should either be repeated in the offline data storage, or rolled back by reloading the offline block.

You transfer one or more blocks to the CPU by going to the offline window of the SIMATIC Manager and opening the *Blocks* folder, selecting the desired block(s), and selecting the command *Target System* > *Load*. If the online window is opened in parallel, you can also drag the block(s) with the mouse from the offline window to the online window.

To add a new block to the online user program by going to the online window of the SIMATIC Manager and selecting *Insert* > *S7 block* > *[block type]* or going to the program editor and selecting *File* > *New...* and activating the option *Online* in the dialog window.

To delete a block, select the block in the offline or the online window and select the command *Edit* > *Delete*. The selected block is deleted from the offline and online data storage.

If a data block is transferred to the CPU, it is saved with the *initial values* of the data variables in the load memory and then in the work memory. In the work memory, the *actual values* are then derived from the initial values, because the user program can change the values of data variables at runtime so that the contents of a data block in load memory and in work memory will differ. You can see the actual values of a data block in the work memory: Open the data block and select the command *View* > *Data View*. The actual values are shown in the *Actual Values* column.

To “upload” individual blocks from the CPU to the programming device, select the block in the online window and choose the command *Target System* > *Load in PG* or “drag” it with the mouse from the online window to the offline window. For data blocks, the actual value is retrieved from the work memory and used as the initial value in the offline data storage.

Just as with individual blocks, you can also individually transfer only the configuration data to the CPU, for example after a change in the hardware configuration. In the offline window, select the *System Data* object in the *Blocks* folder and choose *Target System* > *Load*. Loading configuration data is only possible in STOP mode. It is also possible to “upload” the online configuration to the offline data management.

You can compare two blocks with each other. To compare, select a block in the *Blocks* folder and select the command *Options* > *Compare Blocks*.... In the dialog window, you can compare the offline block with the online block or a block from another project. If the *Perform Code Comparison* option is activated, the program code of the blocks is compared; if the *Only Compare Timestamps* option is enabled, only the timestamps of the block interfaces are compared.

Editing individual blocks offline/online with STEP 7 inside TIA Portal

A block is available in two versions for the program editor of STEP 7 inside TIA Portal: the offline version in the programming device and the online version in the CPU. Only the offline version of the block can be edited.

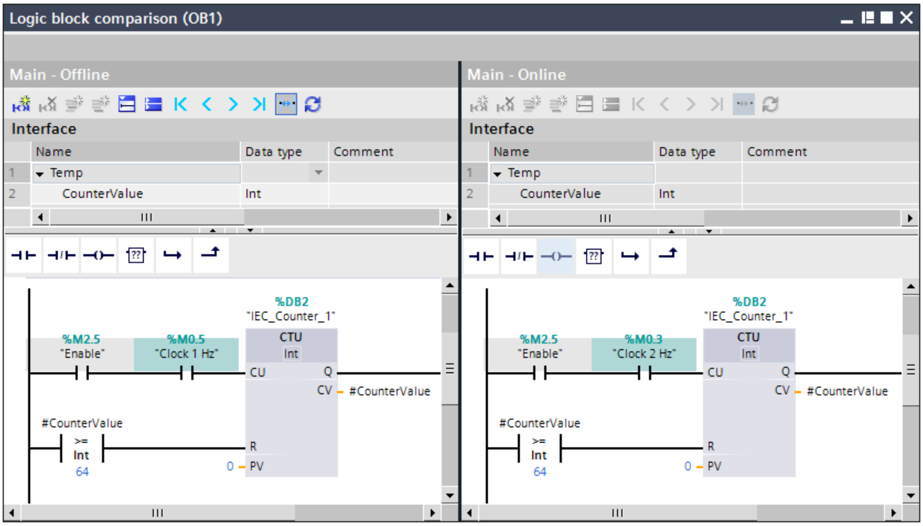
In the working window, the program editor shows either the offline version, or – if online mode is activated – the online version of the block. To differentiate, the window with the online version has an orange title bar. If you want to change the program of the online block, the program editor switches to the offline version. Here you can change the block and then transfer the changed block to the CPU. In this way, the offline and the online version of a block remain identical.

Transfer one or more blocks to the CPU by selecting the blocks in the *Program Blocks* folder in the project tree and select the command *Download to the device* > *Software* from the shortcut menu or *Online* > *Download to the device* from the main menu.

You add a new block to the online user program by creating a new block in the offline data management, programming it, and transferring it to the CPU.

To delete a block, select the block in the project tree and choose *Delete* from the shortcut menu. If the block is available in both the offline and the online version, you can choose in a dialog whether you want to delete the offline or the online version or both.

If a data block is transferred to the CPU, it is saved with the *initial values* of the data variables in the load memory and then in the work memory. In the work memory, the *actual values* are then derived from the initial values, because the user program can change the values of data variables at runtime so that the contents of a data block in load memory and in work memory will differ. You can see the actual values of a data block in the work memory: Open the data block in online mode and then turn on the monitoring mode. The actual values are shown in the *Monitor value* column.



Logic block comparison (OB1)

Main - Offline

Interface

	Name	Data type	Comment
1	Temp		
2	CounterValue	Int	

Main - Online

Interface

	Name	Data type	Comment
1	Temp		
2	CounterValue	Int	

Fig. 3.24 Example of a block comparison for STEP 7 inside TIA Portal

"Uploading" from the CPU to the programming device is not possible for individual blocks. You can "upload" the complete online user program: Create a new project with a PLC station and the CPU in use; with an open project, activate online mode; select the *Program blocks* folder in the project tree and select *Online* > *Load from device* in the main menu. The user program in the CPU is then loaded to the project, but without the symbolic addresses and the names of local variables and blocks. For data blocks, the actual value is retrieved from the work memory and used as the initial value in the offline data storage (not for S7-1200).

You can compare two blocks with each other. Here, the points in time of a program change (timestamp) are compared. Two data blocks are considered identical if the data structure is the same; the contents may vary. To compare the offline with the online version of a block, you select it in the project tree and select *Compare* > *Offline/online* in the shortcut menu. The *Compare* > *Offline/offline* command allows you to compare the block with a block from another project (Fig. 3.24).

3.15 Controlling the user program with online tools

In online mode, STEP 7 makes functions ("online tools") available that allow you to control the PLC station from the programming device and obtain information about the running user program.

Online tools in STEP 7 V5.5

In online mode, additional information from the CPU can be called. To do this, select the CPU in the SIMATIC Manager and choose the commands listed below.

- ▷ *Target System > Diagnostics/Setting > Operating Mode*
  The current operating mode – such as RUN or STOP – is shown. With buttons that mimic the mode switch, the operating mode can be changed from the programming device.

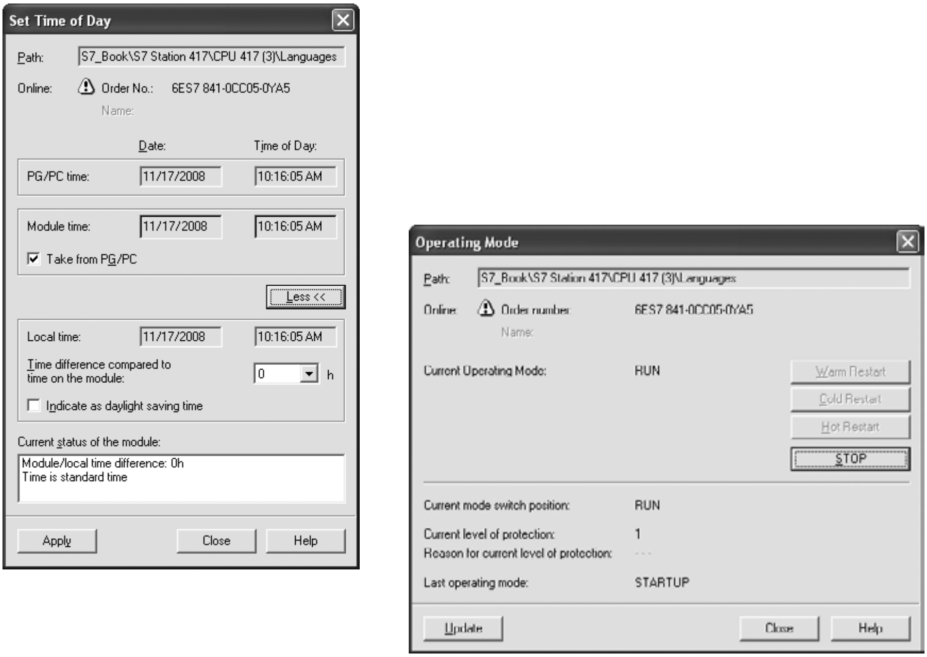
- ▷ *Target System > Diagnostics/Setting > Reset*
  This command performs a CPU reset (see section "Resetting the CPU" on page 104).

- ▷ *Target System > Diagnostics/Setting > Module Status*
  The module status contains the CPU properties such as the current utilization of work and load memory, the processing time of the longest, shortest, and last program cycle, and the performance data of the CPU. This command also reads out the diagnostic buffer (see section "Diagnostic buffer" on page 107).

- ▷ *Target System > Diagnostics/Setting > Set Time*
  This command sets the CPU-internal clock.

- ▷ *Target System > CPU Messages*
  This command displays asynchronous error messages and user-defined messages in the CPU that are generated in the program with system blocks.

Fig. 3.25 shows the dialog window for setting the CPU clock and the operating mode to control the CPU with STEP 7 V5.5.



Set Time of Day

Path: S7_Book\S7 Station 417\CPU 417 (3)\Languages

Online: Order No.: 6ES7 841-0CC05-0YA5

Name:

	Date:	Time of Day:
PG/PC time:	11/17/2008	10:16:05 AM
Module time:	11/17/2008	10:16:05 AM
Local time:	11/17/2008	10:16:05 AM

☑ Take from PG/PC

Time difference compared to time on the module: 0 h

☐ Indicate as daylight saving time

Current status of the module:
Module/local time difference: 0h
Time is standard time

Operating Mode

Path: S7_Book\S7 Station 417\CPU 417 (3)\Languages

Online: Order number: 6ES7 841-0CC05-0YA5

Name:

Current Operating Mode: RUN

Current mode switch position: RUN

Current level of protection: 1

Reason for current level of protection: - - -

Last operating mode: STARTUP

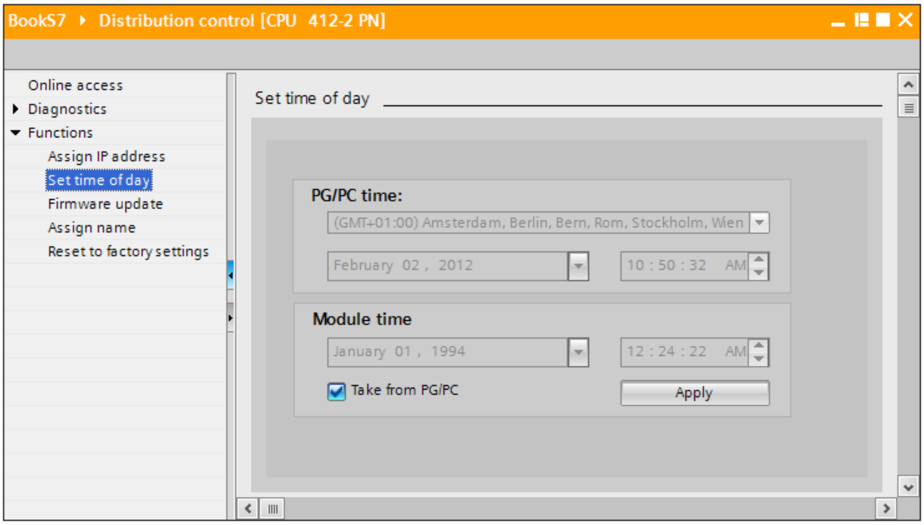
Fig. 3.25 CPU information “Set Time of Day” and “Operating Mode” in STEP 7 V5.5

Online tools for STEP 7 inside TIA Portal

To activate the online tools, double-click in the project tree under the CPU on *Online & diagnostics*. This opens the diagnostics window that shows the connection status for the CPU, and the task card with the online tools. If there is not yet any connection, click the *Connect Online* icon or the *Connect Online* button.

The *CPU Control Panel* pallet displays the status of the LED on the front side of the CPU and provides the RUN, STOP, and MRES (reset) buttons to control the CPU. The *Cycle Time* palette displays the shortest, current, and longest cycle time since the last power-up. The *Memory* palette contains the assignment display of the load memory, work memory, and retentive memory.

In the diagnostics window, under the *Functions* group, you can set the time of the CPU, set the device name and IP address for a PROFINET connection, and reset the CPU to the factory settings (Fig. 3.26).



Set time of day

PG/PC time:

(GMT+01:00) Amsterdam, Berlin, Bern, Rom, Stockholm, Wien

February 02 , 2012 10 : 50 : 32 AM

Module time

January 01 , 1994 12 : 24 : 22 AM

☑ Take from PG/PC

Fig. 3.26 Setting the time of the CPU with STEP 7 inside TIA Portal

Resetting the CPU

A memory reset returns the CPU to its “initial state”. The CPU deletes the entire user program in the work memory and in the RAM load memory. The system memory with the address areas is also deleted, regardless of the setting for retentive behavior. The CPU resets the parameters of all modules – including its own – to the standard settings. The MPI parameters form an exception. These are not changed so that a reset CPU remains addressable on the MPI bus. The diagnostic buffer, the real-time clock, and the runtime meters are not reset either. If there is a user program in the (F)EPROM load memory, the execution-relevant part is copied into the work memory and the configuration data is transferred to the CPU.

With a CPU 300 and CPU 400, you can trigger a reset using the mode switch on the front of the CPU: Hold the switch in the MRES position for at least 3 s, release, and then at the latest within 3 s hold in the MRES position again for at least 3 s.

With a programming device you initiate a reset in the online tools if the CPU is in STOP mode.

Protection of the user program in the CPU

Access to the user program in the CPU can be protected by a password. Anyone with knowledge of the password has unlimited access to the user program. You can define three protection levels for all those who do not know the password:

- Protection level 1 (no protection) is the default setting
- Protection level 2 allows only the user program to be read
- Protection level 3 permits neither read nor write access to the user program

Test functions, such as reading the diagnostic buffer or monitoring variable values, are possible in every protection level without password. You set the protection levels in Hardware Config when parameterizing the CPU properties.

Web server

CPUs with an Ethernet interface have a web server that provides information from the CPU. To read out the information, you require a web browser that displays HTML pages.

The web server is activated with Hardware Config when configuring the CPU properties. You can set the update interval of the web pages, the project language, and the authorized users, among others.

If no users are configured, anyone can read all web pages without being logged on. The (default) user “Everybody” can access all web pages enabled for the user “Everybody” without being logged on and without a password. Access privileges to the web pages can be assigned individually to a configured user with password.

To start the web server, enter the IP address of the CPU in the web browser in the form http://aaa.bbb.ccc.ddd or – if a secure connection is configured – https://aaa.bbbb.ccc.ddd.

The first page displayed by the web server is the Welcome page. By clicking ENTER, you come to the start page, which contains general information about the CPU.

The web server can show, for example, the contents of the diagnostic buffer, the diagnostic status of the CPU, and the configured messages. The state or the value of variables can be monitored using a variable table. It is also possible to create your own personalized web pages, to load these to the CPU, and then to display them using a web browser.

3.16 Finding hardware faults using diagnostic functions

Hardware diagnostics can detect and report faults in modules, such as a failure of the load voltage or a wire break on signal modules. An error in the PLC station is displayed on the front of the CPU with a red error LED. Modules with self-diagnosis ability can also signal a detected error by means of a frontal LED.

In online mode, STEP 7 makes extensive functions available to locate a faulty module and identify the cause of the fault. The operating system of the CPU provides additional diagnostic features that can be used with the user program at runtime (see chapter 5.12 "Synchronous errors with a CPU 300/400" on page 184 and the chapters following it).

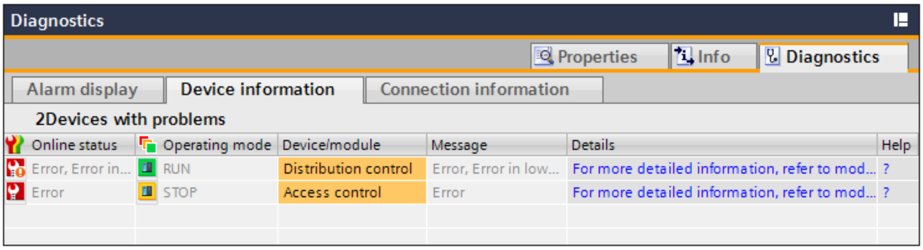
Locating and reading out faulty modules with STEP 7 V5.5

You are given an overview of the faulty modules when you open the project in the SIMATIC Manager, select the SIMATIC station, and select *Target System* > *Diagnostics/Setting* > *Diagnose Hardware* in the main menu. In the default setting, which you can change via *Options* > *Settings* in the *View* tab, STEP 7 shows a module overview in the “quick view”.

The *Module Status* button in the quick view gives you detailed diagnostic information, depending on the diagnostic capability of the module. You receive the same information if you open the module in the diagnostics view in Hardware Config, or in SIMATIC Manager by selecting theSIMATIC station and choosing *Target System* > *Diagnostics/Setting* > *Module Status*. The *Module Status* window shows the data loaded by the CPU data in multiple tabs, such as the memory usage of the user program in the CPU, the current processing cycle time, or the messages from the diagnostic buffer.

Locating and reading out faulty modules with STEP 7 inside TIA Portal

The inspector window in the *Diagnostics* > *Device Information* tab displays the status of the devices reported as faulty. A device is considered to be faulty if it is inaccessible when establishing the online connection, if it signals a fault, or if it is not in RUN mode. Via the link in the *Details* column you can access the *Connect online* dialog or the online and diagnostics view of the faulty device (Fig. 3.27). In online



2Devices with problems

Online status	Operating mode	Device/module	Message	Details	Help
Error, Error in...	RUN	Distribution control	Error, Error in low...	For more detailed information, refer to mod...	?
Error	STOP	Access control	Error	For more detailed information, refer to mod...	?

Fig. 3.27 Diagnostics in the inspector window for STEP 7 inside TIA Portal

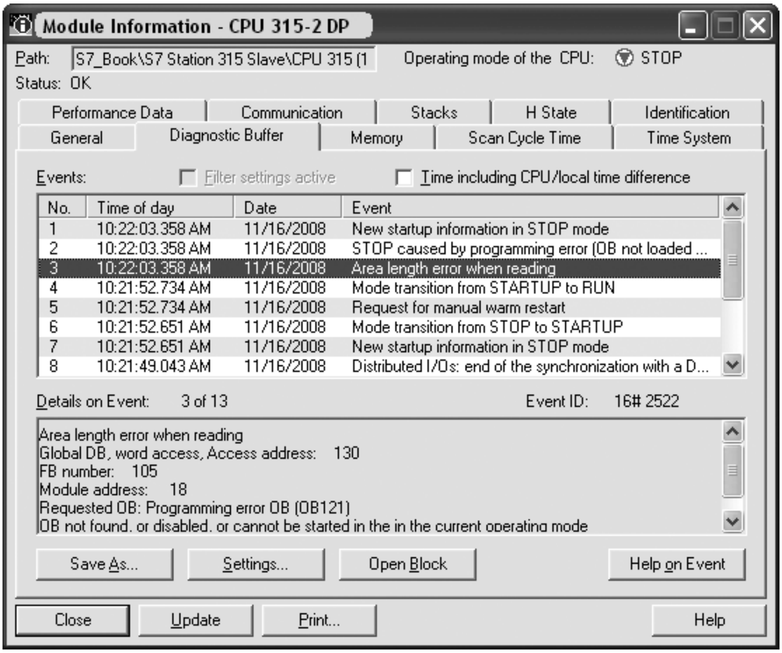
mode, the Hardware Config in the device or network view shows the operating and diagnostic status using icons at every connected PLC station. The project tree also uses colored icons to indicate the diagnostic status.

Double-clicking on *Online & diagnostics* under the PLC station in the project tree opens the diagnostic window. Under *Online access* the diagnostic window shows the connection status and the interface properties. A click on the *Connect Online* button establishes a connection to the CPU.

The information read by the CPU is displayed in the *Diagnostics* group; under *Diagnostics* > *General* e.g. the order number, serial number, and hardware and firmware version are shown; and *Diagnostics* > *Diagnostic Status* shows the diagnostic status of the module. You can read out the current cycle processing time and current memory usage of the user memory. The diagnostic buffer can be read out with *Diagnostics* > *Diagnostic Buffer.*

Diagnostic buffer

All diagnosis events signaled to the CPU's operating system are entered into a *diagnostic buffer* in the sequence of their occurrence with date and time. The diagnostic buffer is a buffered memory area in the CPU, which retains its contents even after a reset. The diagnostic buffer is a ring buffer; its size is CPU-specific. If the diagnostic buffer is full, the oldest entry is overwritten by the current diagnostic event.



Path: S7_Book\S7 Station 315 Slave\CPU 315 (1   Operating mode of the CPU: STOP
Status: OK

Events: ☐ Filter settings active   ☐ Time including CPU/local time difference

No.	Time of day	Date	Event
1	10:22:03.358 AM	11/16/2008	New startup information in STOP mode
2	10:22:03.358 AM	11/16/2008	STOP caused by programming error (OB not loaded ...
3	10:22:03.358 AM	11/16/2008	Area length error when reading
4	10:21:52.734 AM	11/16/2008	Mode transition from STARTUP to RUN
5	10:21:52.734 AM	11/16/2008	Request for manual warm restart
6	10:21:52.651 AM	11/16/2008	Mode transition from STOP to STARTUP
7	10:21:52.651 AM	11/16/2008	New startup information in STOP mode
8	10:21:49.043 AM	11/16/2008	Distributed I/Os: end of the synchronization with a D...

Details on Event: 3 of 13   Event ID: 16# 2522

Area length error when reading
Global DB, word access, Access address: 130
FB number: 105
Module address: 18
Requested OB: Programming error OB (OB121)
OB not found, or disabled, or cannot be started in the in the current operating mode

Fig. 3.28 Example of entries in the diagnostic buffer of the CPU in STEP 7 V5.5

You can read out the diagnostic buffer with a programming device at any time. When configuring the hardware, you can select in the CPU properties if you want extended diagnostic buffer entries (in addition to all of the calls of the organization blocks). You can also set whether the last diagnostics entry – before the CPU goes to STOP mode – is sent to a node logged in for this purpose.

For the diagnostic buffer in STEP 7 V5.5, select the CPU in the SIMATIC Manager, choose the command *Target System > Diagnostics/Setting > Module Status*, and open the *Diagnostic Buffer* tab.

With STEP 7 inside TIA Portal, double-click in the project tree on *Online & diagnostics* under the PLC station, activate the online connection, and click *Diagnostic Buffer* in the *Diagnostics* group.

In the diagnostic buffer, the most recent message numbered 1 allows you to see the cause of the stop, e.g. “STOP caused by programming error OB”. The error that led to this can be found in the previous messages, e.g. “Area length error when reading”. By clicking on the message number, an extended commentary on the message is displayed in the underlying display field. If the message concerns a programming error in a block, you can open and edit the block by clicking *Block Open* (STEP 7 V5.5) or *Open in Editor* (STEP 7 inside TIA Portal).

3.17 Testing with watch tables

An excellent way to test the user program is to monitor and control variables using watch tables (with STEP 7 V5.5, these are variables tables or VAT). This allows the signal states or values of variables to be displayed with elementary data types. If you have access to the user program, you can also control variables, i.e. change the signal state or assign new values.

General procedure for monitoring and controlling variables

You create a new watch table and list all variables in the table whose values you wish to monitor and/or control. Here you define the format with which the value of the variable should be displayed and controlled; this display format does not have to match the data type of the variable in the user program.

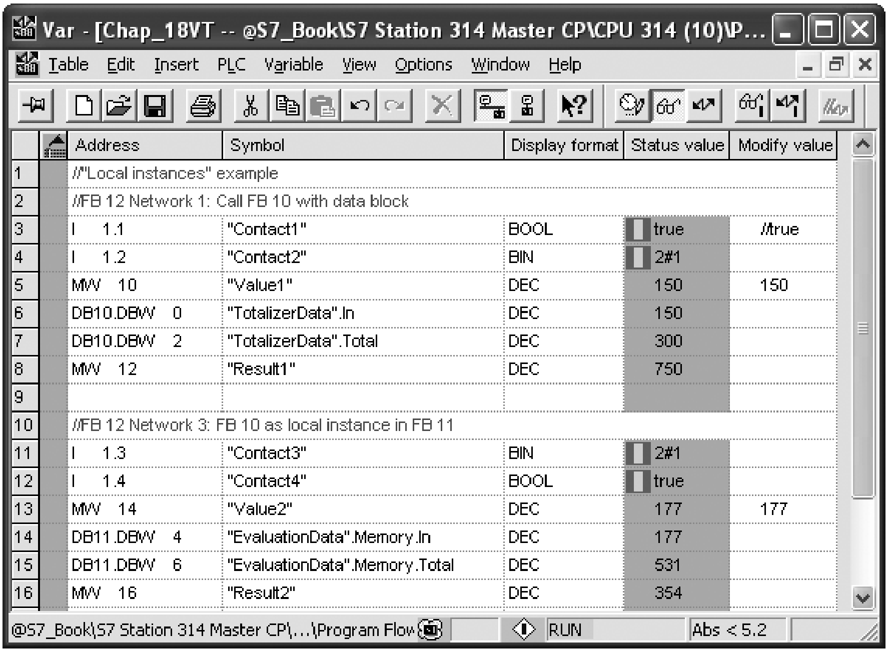
You then specify the point in time – separately for monitoring and control – at which the CPU reads the values from the system memory or writes to system memory. For example, if you specify Start of Cycle as the trigger point for modifying, the variables being modified will be given the specified value before cyclic processing of the program begins. If you specify End of Cycle as the trigger point for monitoring, the values displayed for the variables will be displayed at the end of cyclic processing. Monitoring and modifying can take place once only or permanently.

Once you have placed the watch table in online mode, activate the monitoring of the variables, observe the status value and watch how it changes. If you want to specify a specific value for a variable, enter the control value and activate controlling.

Testing with watch tables with STEP 7 V5.5

To add a new watch table (here: variable table) open the project in the SIMATIC Manager, select the *Blocks* folder, and select *Insert > S7 Block > Variable Table*. Double-clicking the VAT icon opens the variable table in a separate window. Now enter the variables (addresses). The command *Variable > Trigger* allows you to set the time – separately for monitoring and control – at which the CPU reads the values from the system memory or writes to the system memory and whether this should take place once or permanently. The *Target System > Connect To >* ... command allows you to connect the variable table online to the configured CPU, to the directly connected CPU, or to another accessible CPU.

Variable > Monitor updates the variable values depending on the set trigger conditions (Fig. 3.29). With *Variable > Update Status Value* the update takes place once. The command *Variable > Control* transfers the predetermined control values to the CPU depending on the trigger conditions. You first enter the control values in the *Control Value* column (only for the variables whose values should be changed). Entered values can be commented out with a double slash, and they are then ignored in controlling. The transfer occurs once with *Variable > Activate Control Value*.


Var - [Chap_18VT -- @S7_Book\S7 Station 314 Master CP\CPU 314 (10)\P...
Table
Edit
Insert
PLC
Variable
View
Options
Window
Help

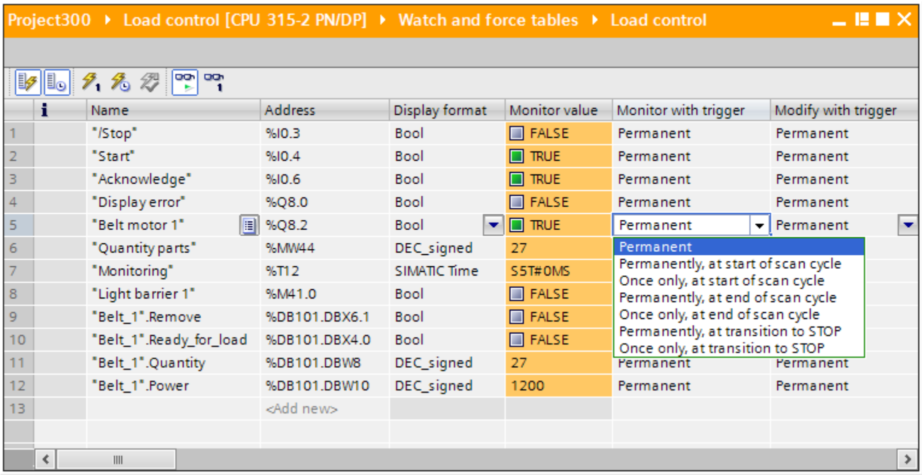
	Address	Symbol	Display format	Status value	Modify value
1	//"Local instances" example				
2	//FB 12 Network 1: Call FB 10 with data block				
3	I 1.1	"Contact1"	BOOL	true	//true
4	I 1.2	"Contact2"	BIN	2#1	
5	MW 10	"Value1"	DEC	150	150
6	DB10.DBW 0	"TotalizerData".In	DEC	150	
7	DB10.DBW 2	"TotalizerData".Total	DEC	300	
8	MW 12	"Result1"	DEC	750	
9					
10	//FB 12 Network 3: FB 10 as local instance in FB 11				
11	I 1.3	"Contact3"	BIN	2#1	
12	I 1.4	"Contact4"	BOOL	true	
13	MW 14	"Value2"	DEC	177	177
14	DB11.DBW 4	"EvaluationData".Memory.In	DEC	177	
15	DB11.DBW 6	"EvaluationData".Memory.Total	DEC	531	
16	MW 16	"Result2"	DEC	354	

@S7_Book\S7 Station 314 Master CP\...\Program Flow | RUN | Abs < 5.2

Fig. 3.29 Example of a variable table for STEP 7 V5.5

Testing with watch tables with STEP 7 inside TIA Portal

To add a new watch table, open the *Watch and force tables* folder under the PLC station in the project tree and double-click on *Add New Watch Table*. Open the empty watch table and enter the variables that should be monitored and/or controlled. To set the trigger time, click on the icon for the expanded mode and enter the trigger condition in the *Monitor with trigger* column (Fig. 3.30). To modify variables in the watch table, display the control columns in the table by clicking on the relevant icon; you can now enter the control value. In expanded mode, you can also specify the trigger conditions in the *Modify with trigger* column.



Project300 ▸ Load control [CPU 315-2 PN/DP] ▸ Watch and force tables ▸ Load control

	i	Name	Address	Display format	Monitor value	Monitor with trigger	Modify with trigger
1		"/Stop"	%I0.3	Bool	□ FALSE	Permanent	Permanent
2		"Start"	%I0.4	Bool	🟩 TRUE	Permanent	Permanent
3		"Acknowledge"	%I0.6	Bool	🟩 TRUE	Permanent	Permanent
4		"Display error"	%Q8.0	Bool	□ FALSE	Permanent	Permanent
5		"Belt motor 1"	%Q8.2	Bool	🟩 TRUE	Permanent	Permanent
6		"Quantity parts"	%MW44	DEC_signed	27		
7		"Monitoring"	%T12	SIMATIC Time	S5T#0MS		
8		"Light barrier 1"	%M41.0	Bool	□ FALSE		
9		"Belt_1".Remove	%DB101.DBX6.1	Bool	□ FALSE		
10		"Belt_1".Ready_for_load	%DB101.DBX4.0	Bool	□ FALSE		
11		"Belt_1".Quantity	%DB101.DBW8	DEC_signed	27	Permanent	Permanent
12		"Belt_1".Power	%DB101.DBW10	DEC_signed	1200	Permanent	Permanent
13			<Add new>				

- Permanent
- Permanently, at start of scan cycle
- Once only, at start of scan cycle
- Permanently, at end of scan cycle
- Once only, at end of scan cycle
- Permanently, at transition to STOP
- Once only, at transition to STOP

Fig. 3.30 Example of the monitoring of variables in expanded mode inside TIA Portal

The *Online* > *Connect online* command in the main menu connects the watch table with the CPU.

You start the monitoring and modifying with the appropriate icons in the toolbar of the working window. Monitoring and modifying may occur in dependency on the set trigger conditions or "once and immediately".

Forcing variables

Forcing is the preallocation of variables with fixed values. The variables are entered into a force table with the force value entered and transferred to the CPU. Forcing will remain valid until it is specifically disabled. Interruption of the online connection or switching the CPU off and on again do not stop the forcing.

In STEP 7 V5.5, you open a variable table (watch table) and select the command *Variable* > *Show Force Values*. This will open a new table into which you enter the variables to be forced with the force value. The *Variable* > *Force* command transfers the force values to the CPU, where they are effective immediately. The *Variable > Delete Force* command ends forcing.

For STEP 7 inside TIA Portal, open the force table in the *Watch and force tables* folder. Enter the variables and the force values. You can start and end forcing by means of the icons in the function bar of the working window.

Enabling peripheral outputs

In STOP mode, the output modules are usually locked, so they issue the value “zero” or substitute value. The *Enable PQ* function allows you to remove the disable signal and also control the output modules in CPU STOP. An application for this would be checking the wiring of the outputs in STOP mode and without a user program.

In STEP 7 V5.5, create a variable table (watch table) and enter the peripheral outputs to be controlled and the control values. Switch the variable table online and stop the CPU if applicable. *Variable > Enable PQ* deactivates the output disable. With *Variable > Activate Control Value* you control the peripheral outputs. You can change the control value and resume controlling. Use the ESC key to switch the function off. The output disable is active again and the module outputs are reset to “0” or the substitute value.

In STEP 7 inside TIA Portal, create a watch table and enter the peripheral outputs to be controlled and the control values. Switch the variable table online and stop the CPU if applicable. Click the icon for the command output disable to deactivate the output disable and the “Control once and immediately” icon to control the peripheral outputs. You can change the control value and resume controlling. *Enable PQ* is ended if a CPU operating mode changes or the online connection is deactivated. The output disable is active again and the module outputs are reset to “0” or the substitute value.

3.18 Testing the program with the program status

The program status shows the current signal states and variable values on the open block. The block whose program you want to test is located in the user memory of the CPU, where it is called and processed. Open this block, go to the point in the program that you want to watch, and turn on the program status.

In STEP 7 V5.5, select the Blocks folder in the SIMATIC Manager and use the *View > Online* command to open the online window. Double-click the block to be tested to open it. Go to the point in the program that you want to test, and use *Debug > Monitor* to switch on the program status. Now you can monitor the signal flow in the program. Click again on *Debug > Monitor* to end the program status.

In STEP 7 inside TIA Portal, open the *Program Blocks* folder in the project tree and double-click the block to be tested. In the open block, go to the point in the program that you want to test and click on the *Monitoring On/Off* icon in the toolbar of the working window. The online mode is turned on and you can monitor the signal flow in the program. Click again on *Monitoring On/Off* to end the program status.

In the LAD program status, continuous lines are used to identify contacts, coils, and the connections between the program elements which have signal state “1”. Program elements with signal state “0” are identified by dashed lines. The value of a digital variable is located right at the variable (Fig. 3.31).

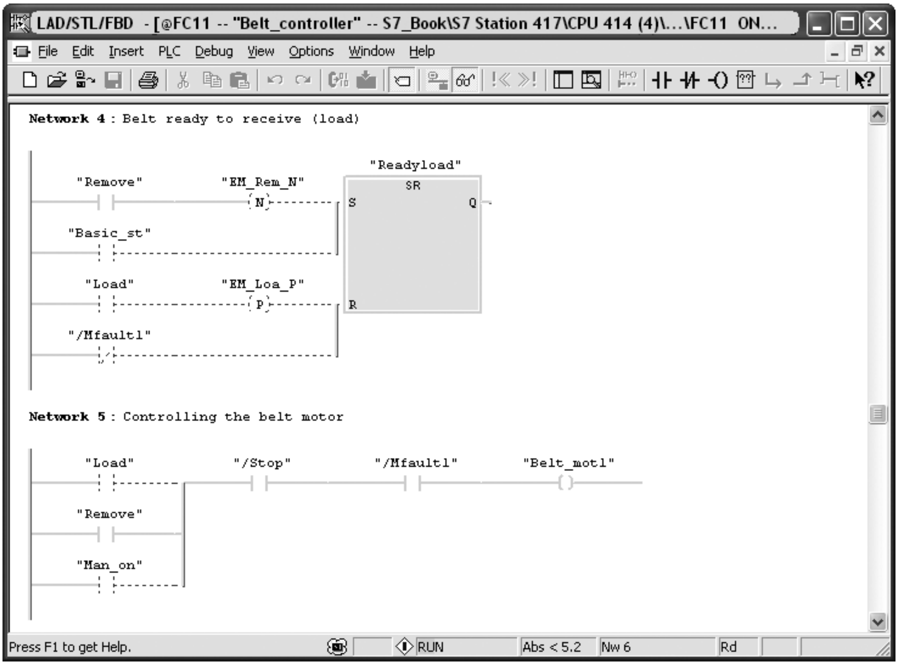

LAD/STL/FBD - [@FC11 -- "Belt_controller" -- S7_Book\S7 Station 417\CPU 414 (4)\...\FC11 ON...
File Edit Insert PLC Debug View Options Window Help
Network 4 : Belt ready to receive (load)
"Readyload"
"Remove"
"EM_Rem_N"
N
SR
S
Q
"Basic_st"
"Load"
"EM_Loa_P"
P
R
"/Mfault1"
Network 5 : Controlling the belt motor
"Load"
"/Stop"
"/Mfault1"
"Belt_mot1"
"Remove"
"Man_on"
Press F1 to get Help.
RUN
Abs < 5.2
Nw 6
Rd

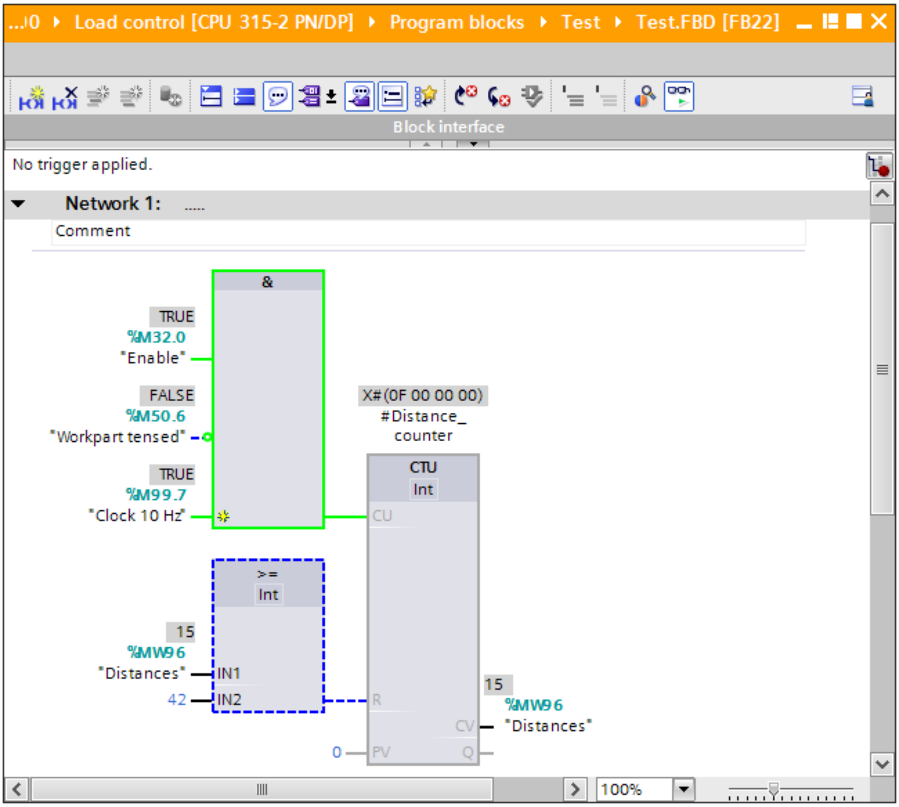
Fig. 3.31 Example of LAD program status for STEP 7 V5.5

In the FBD program status, the boxes of the binary program elements and connections with signal state “1” are represented by solid lines. Program elements with signal state “0” are identified by dashed lines. The value of a digital variable is located right at the variable (Fig. 3.32).

The STL program status is shown in tabular form to the right of the statements so that the variable value can be read for each statement line (Fig. 3.33).

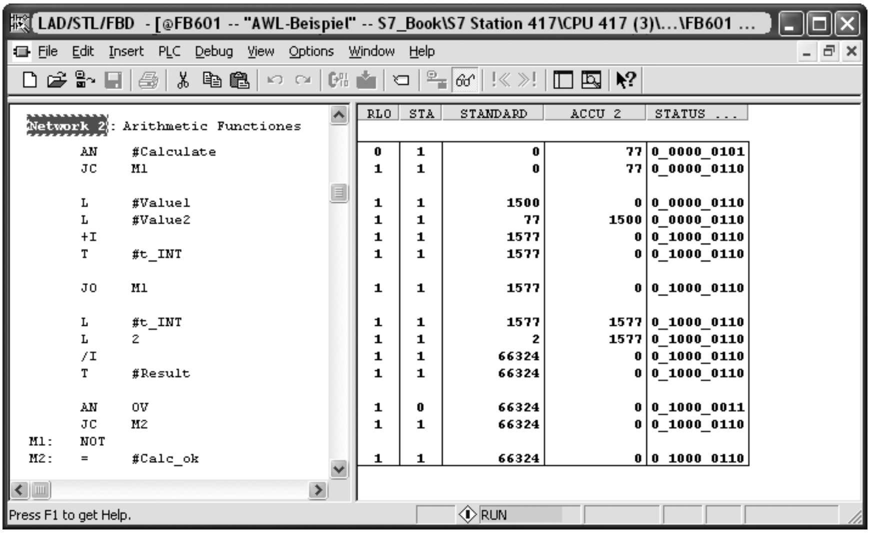
In the SCL program status, the variables used in the row are displayed with their values to the right of the statements so that the value for each variable can be read (Fig. 3.34).

Monitoring the program status extends the cyclic processing time of the program. You have a choice of two modes of operation for the test. In *Test Operation* you can



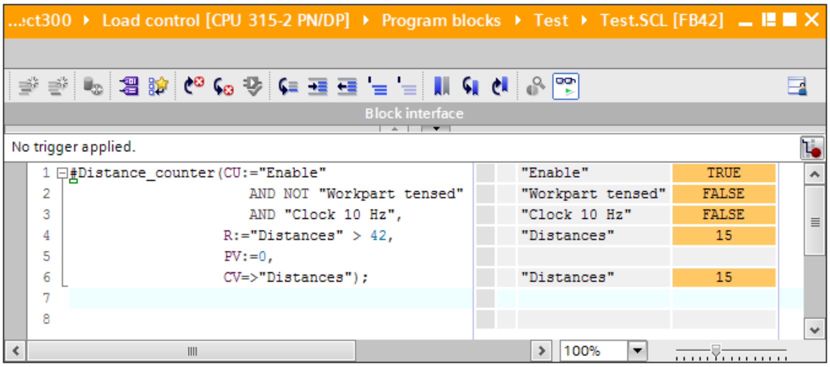
No trigger applied.

Fig. 3.32 Example of an FBD program status for STEP 7 inside TIA Portal


LAD/STL/FBD - [@FB601 -- "AWL-Beispiel" -- S7_Book\S7 Station 417\CPU 417 (3)\...\FB601 ...
File Edit Insert PLC Debug View Options Window Help
Press F1 to get Help.
RUN

```
Network 2: Arithmetic Functiones          RLO  STA  STANDARD    ACCU 2  STATUS ...

      AN    #Calculate                     0    1          0        77  0_0000_0101
      JC    M1                             1    1          0        77  0_0000_0110

      L     #Value1                        1    1       1500         0  0_0000_0110
      L     #Value2                        1    1         77      1500  0_0000_0110
      +I                                   1    1       1577         0  0_1000_0110
      T     #t_INT                         1    1       1577         0  0_1000_0110

      JO    M1                             1    1       1577         0  0_1000_0110

      L     #t_INT                         1    1       1577      1577  0_1000_0110
      L     2                              1    1          2      1577  0_1000_0110
      /I                                   1    1      66324         0  0_1000_0110
      T     #Result                        1    1      66324         0  0_1000_0110

      AN    OV                             1    0      66324         0  0_1000_0011
      JC    M2                             1    1      66324         0  0_1000_0110
M1:   NOT
M2:   =     #Calc_ok                       1    1      66324         0  0 1000 0110
```

Fig. 3.33 Example of STL program status for STEP 7 V5.5


...ct300 ▸ Load control [CPU 315-2 PN/DP] ▸ Program blocks ▸ Test ▸ Test.SCL [FB42]
Block interface
No trigger applied.
100%

```
#Distance_counter(CU:="Enable"
                  AND NOT "Workpart tensed"
                  AND "Clock 10 Hz",
               R:="Distances" > 42,
               PV:=0,
               CV=>"Distances");
```

"Enable"	TRUE
"Workpart tensed"	FALSE
"Clock 10 Hz"	FALSE
"Distances"	15
"Distances"	15

Fig. 3.34 Example of an SCL program status for STEP 7 inside TIA Portal

use all the debugging functions without restrictions, and in *Process Operation* the extension of the scan cycle time is kept to a minimum, which may result in restrictions.

Testing in single step mode

In the programming languages STL and SCL, you can test the program statement by statement in single step mode. The CPU is in the HOLD state in this case; the peripheral outputs should be switched off as a precaution. You can use breakpoints to stop the program at any desired position and test step-by-step.

To test in single step mode, set breakpoints in the offline block. If program processing is to stop at a breakpoint, you must activate it. You can set any number of breakpoints, but only activate one quantity that depends on the CPU. When activated, the breakpoint is transferred to the CPU. Program execution is carried out in RUN mode up to the first active breakpoint and then stops in HOLD mode. You can now advance the program execution statement by statement, and monitor the course of the program with the signal states, variable values, and register contents.

In STEP 7 V5.5 you must open the block to be tested, place the cursor in the appropriate statement line, and select *Debug* > *Set Breakpoint*. With *Debug* > *Active Breakpoint*, the breakpoint is transferred to the CPU, and the program is executed up to the breakpoint. With *Debug* > *Perform next statement* you can now have the program process each instruction individually. You exit single step mode by deleting all breakpoints and the command *Debug* > *Resume*.

In STEP 7 inside TIA Portal, open the block to be tested, right-click in the gray box in front of the statement line, and select the command for setting a breakpoint from the shortcut menu. To activate the breakpoint, select it and choose *Activate Breakpoint* from the shortcut menu. The program then executes up to the breakpoint. The *Skip* command executes the current instruction and stops at the next instruction. To exit single step mode, delete all breakpoints, then click the gray column before the instruction and the *Execute* command.

3.19 Testing user programs offline using S7-PLCSIM

The S7-PLCSIM optional software permits you to test the user program offline without additional hardware. S7-PLCSIM simulates a PLC station on the programming device. The CPU is not specified, which means that you can test any user program with S7-PLCSIM. It makes no difference which programming language the blocks are written in. Only one PLC can be simulated at any one time.

For STEP 7 V5.5, S7-PLCSIM is an option package that is incorporated into SIMATIC Manager after installation. You start the simulation in SIMATIC Manager with *Options > Simulate Modules*. For STEP 7 inside TIA Portal, S7-PLCSIM is already integrated, and is started with a selected PLC station using *Online > Simulation > Start*.

The S7-PLCSIM user interface

The S7-PLCSIM user interface appears in a separate window. Following selection of a program or *Simulation > New PLC*, the CPU subwindow is displayed. It shows the controls and LEDs of a *CPU*. You just click with the mouse to switch the simulated CPU to the RUN and RUN-P modes and back to STOP or perform a memory reset on the CPU. If the simulated CPU goes into STOP mode as a result of a programming error, for example, this is indicated by the group error LED and the STOP LED.

To simulate the plant to be controlled, there are subwindows for inputs and outputs. Each subwindow can represent a bit, byte, word or doubleword address in different data formats. You can use these subwindows, for example, to simulate the signal states of inputs (by clicking them with the mouse) and observe how the outputs are set and reset.

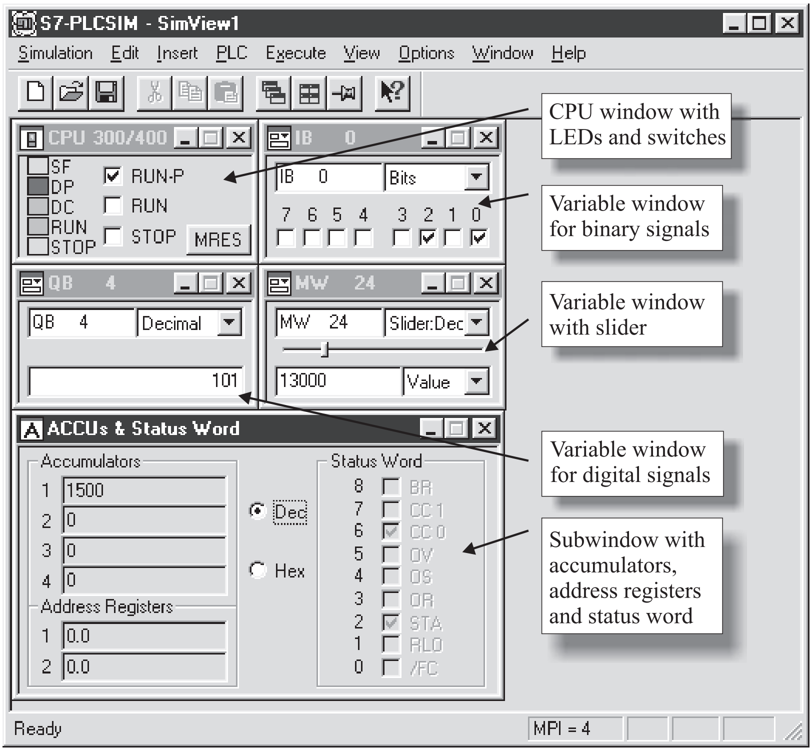
You specify digital values in the correct format for the data type. A slider enables you to simulate values that are constantly changing, such as analog values. You can change the position of the sliders using either the mouse or the arrow keys.

You can also monitor and modify internal CPU addresses in separate subwindows. There are subwindows for bit memories, timers, counters and general variables. In a variable subwindow (inputs, outputs, bit memories or general variables) you can enter any shared addresses, including specific data addresses (for example DB62.DBB15).

You can select *Tools > Options > Attach Symbols...* to assign a symbol table to S7-PLCSIM so that you can display symbolic names for the variables (by selecting the menu commands *Tools > Options > Show Symbols*). You can also edit the assigned symbol table from S7-PLCSIM.

Downloading and running a program

When you start S7-PLCSIM, the creation system is online to an imaginary CPU. You can now download the user program to the user memory of the CPU by clicking the menu commands PLC → DOWNLOAD in the SIMATIC Manager – as you would with a real CPU. If you switch to online view, you will see the blocks that have been downloaded to the CPU in the online window. Then you click RUN or RUN-P in the CPU pane, S7-PLCSIM processes the downloaded user program like a real CPU.



☑ RUN-P
□ RUN
□ STOP

IB 0 Bits

7	6	5	4	3	2	1	0
□	□	□	□	□	☑	□	☑

QB 4 Decimal

101

MW 24 Slider:Dec

13000 Value

Accumulators

1 1500
2 0
3 0
4 0

Address Registers

1 0.0
2 0.0

● Dec
○ Hex

Status Word

8 □ BR
7 □ CC 1
6 ☑ CC 0
5 □ OV
4 □ OS
3 □ OR
2 ☑ STA
1 □ RLO
0 □ /FC

MPI = 4

CPU window with LEDs and switches

Variable window for binary signals

Variable window with slider

Variable window for digital signals

Subwindow with accumulators, address registers and status word

Fig. 3.35 Testing programs with S7-PLCSIM

If a program error occurs, e.g. access to an address that does not exist, S7-PLCSIM responds by calling the synchronous error organization block OB 122. If this is not present in the program, the CPU goes into STOP mode. The further procedure is as when testing with a real CPU, for example reading out the diagnostic buffer and determining the cause of the stop.

You test the user program using the variable table (watch table) and the program status; for STL and SCL also single-step mode with breakpoints. The CPU simulated with S7-PLCSIM behaves like a real CPU here, too.

3.20 Documentation in wiring manual format with DOCPRO

You use the optional software DOCPRO for STEP 7 V5.5 to create and manage your system documentation. DOCPRO enables you to:

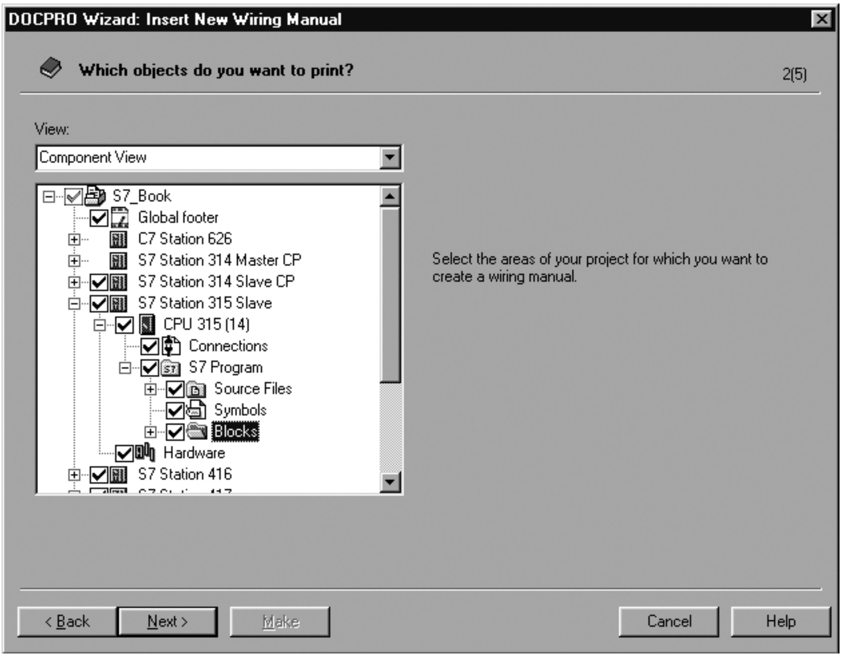
- Put the data to be printed together in any order to produce wiring manuals of uniform appearance with uniform footers
- Design the layout to DIN 6771 (standard for technical documentation) using the templates provided or to suit your own individual requirements

- ▷ Number drawings automatically or manually
- ▷ Create document indexes automatically

Examples of data you can document with DOCPRO are program code in the relevant programming language, symbol tables, reference data, configuration tables, global data table and connection table.

Generating system documentation

After installing DOCPRO, insert an object called *Documentation* in the project folder with *Insert > Project Documentation*. Double-click on *Documentation* to start DOCPRO. In the left-hand side of the documentation window you will see the structure of the system documentation in the form of wiring manuals and job lists and on the right-hand side you will see their contents. The wiring manual wizard helps you create new system documentation (Fig. 3.36).



DOCPRO Wizard: Insert New Wiring Manual

Which objects do you want to print? 2(5)

View: Component View

- ☑ S7_Book
  - ☑ Global footer
  - C7 Station 626
  - S7 Station 314 Master CP
  - ☑ S7 Station 314 Slave CP
  - ☑ S7 Station 315 Slave
    - ☑ CPU 315 (14)
      - ☑ Connections
      - ☑ S7 Program
        - ☑ Source Files
        - ☑ Symbols
        - ☑ Blocks
    - ☑ Hardware
  - ☑ S7 Station 416
  - S7 Station 417

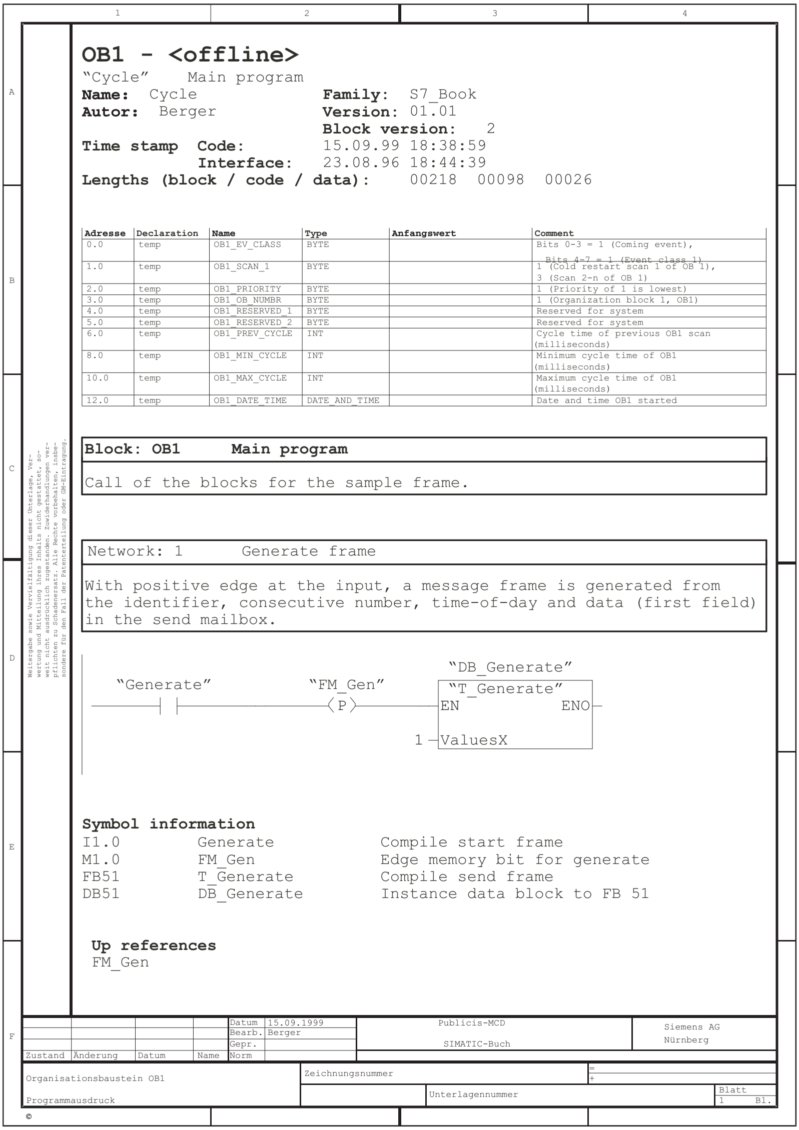
Select the areas of your project for which you want to create a wiring manual.

Fig. 3.36 Example of the DOCPRO circuit manual wizard

You can change the layout and sequence of the documentation at any time. Set the desired layout by choosing *Options > Settings for Print Object Types...*. Choose *Insert > Wiring Manual*, *Insert > Job List*, *Insert > Print Objects...* and *Insert > Coversheet* to add items to the documentation. The documentation can consist of several wiring manuals, which can each contain several job lists, in which the print objects (e.g. symbol table, compiled blocks, cross-reference list) are listed in the order in which they are to be printed.

When the printing process is completed, a document index is printed. This is a table showing the objects that have been printed. You can specify a layout for the document index in the same way as for any other print object.

Fig. 3.37 shows an example of a wiring manual printout for OB 1 with DIN A4 layout in portrait format with footer. You can also choose DIN A3 in portrait or landscape format or design your own layout.


1
2
3
4
A
B
C
D
E
F

Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte vorbehalten, insbesondere für den Fall der Patenterteilung oder GM-Eintragung.

```
OB1 - <offline>
"Cycle"    Main program
Name:  Cycle                 Family:  S7_Book
Autor:  Berger               Version: 01.01
                             Block version:   2
Time stamp  Code:            15.09.99 18:38:59
            Interface:       23.08.96 18:44:39
Lengths (block / code / data):     00218  00098  00026
```

Adresse	Declaration	Name	Type	Anfangswert	Comment
0.0	temp	OB1_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
1.0	temp	OB1_SCAN_1	BYTE		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
2.0	temp	OB1_PRIORITY	BYTE		1 (Priority of 1 is lowest)
3.0	temp	OB1_OB_NUMBR	BYTE		1 (Organization block 1, OB1)
4.0	temp	OB1_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB1_RESERVED_2	BYTE		Reserved for system
6.0	temp	OB1_PREV_CYCLE	INT		Cycle time of previous OB1 scan (milliseconds)
8.0	temp	OB1_MIN_CYCLE	INT		Minimum cycle time of OB1 (milliseconds)
10.0	temp	OB1_MAX_CYCLE	INT		Maximum cycle time of OB1 (milliseconds)
12.0	temp	OB1_DATE_TIME	DATE_AND_TIME		Date and time OB1 started

```
Block: OB1      Main program

Call of the blocks for the sample frame.
```

```
Network: 1       Generate frame

With positive edge at the input, a message frame is generated from
the identifier, consecutive number, time-of-day and data (first field)
in the send mailbox.
```

```
Symbol information
I1.0        Generate           Compile start frame
M1.0        FM_Gen             Edge memory bit for generate
FB51        T_Generate         Compile send frame
DB51        DB_Generate        Instance data block to FB 51

 Up references
 FM_Gen
```

				Datum	15.09.1999	Publicis-MCD	Siemens AG
				Bearb.	Berger		Nürnberg
				Gepr.		SIMATIC-Buch	
Zustand	Änderung	Datum	Name	Norm			

Organisationsbaustein OB1	Zeichnungsnummer		= +
Programmausdruck		Unterlagennummer	Blatt 1 Bl.

©

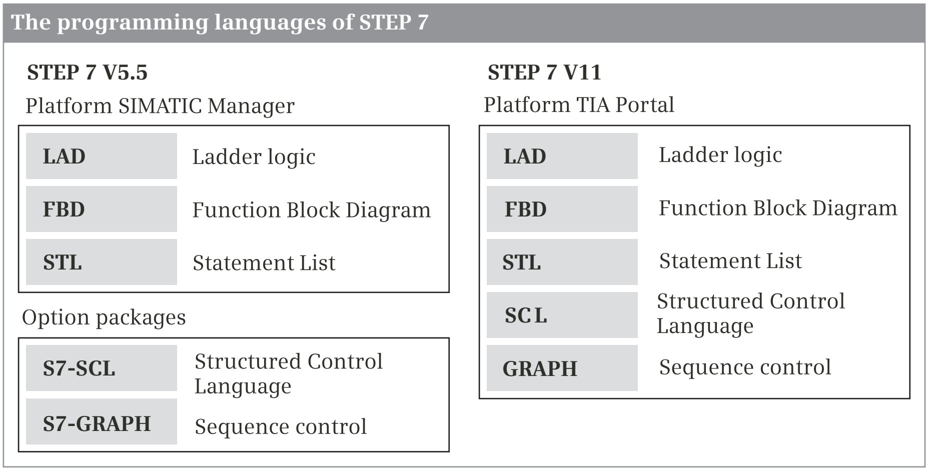
Fig. 3.37 Example of a wiring manual printout using DOCPRO

4 The programming languages

You use the programming languages to write the user program. You have a choice of several programming languages and programming methods to suit your particular needs or preference. STEP 7 V5.5 is supplied with the programming languages LAD, FBD, and STL. SCL and GRAPH are available as option packages. STEP 7 Professional V11 inside TIA Portal is supplied with LAD, FBD, STL, SCL, and GRAPH (Fig. 4.1).

Every programming language has a certain *functional scope* that also depends on the CPU type for which the user program is intended. For example, there are functions for the logic operation of binary signal states, for arithmetic calculations, and for controlling program execution.

The values manipulated in the user program are located in specific areas of the memory called *address areas*. A distinction is made between global addresses, which are available throughout the user program, and block-specific variables, which are only "valid" in the specific block. For an address to be addressed, it needs an *address*. Absolute addressing uses the storage location as identification, while symbolic addressing uses a character string (a name). The *data types* specify the value range and the internal structure (data storage) of the variable. There are basic data types that can be edited using the "simple" statements of a programming language, and complex data types that are made up of individual components and represent a unit.



The programming languages of STEP 7

STEP 7 V5.5
Platform SIMATIC Manager

LAD	Ladder logic
FBD	Function Block Diagram
STL	Statement List

Option packages

S7-SCL	Structured Control Language
S7-GRAPH	Sequence control

STEP 7 V11
Platform TIA Portal

LAD	Ladder logic
FBD	Function Block Diagram
STL	Statement List
SC L	Structured Control Language
GRAPH	Sequence control

Fig. 4.1 The programming languages of STEP 7

Overview of programming languages

LAD (ladder logic) emulates a circuit diagram. The series and parallel connection of contacts allows binary signals to be linked. Timer, counter, and digital functions are inserted as boxes in the circuit diagram. The signal states of binary signals can be represented very clearly here as “current flows”.

FBD (function block diagram) represents all the links as boxes, similar to the depiction of electronic circuitry. The inputs of the boxes are assigned to the variables whose signal states and values are to be linked. The result of the link at the box output is fed to a variable or an input of another box.

STL (statement list) consists of single statements listed line by line. Logic operations are implemented that process the signal states and values of addresses and variables. In addition to the linking of binary and digital variables, complex variables can be edited with STL and addresses and variables can be addressed indirectly.

SCL (Structured Control Language) is a textual programming language for programming complex algorithms and handling large quantities of structured data. The instruction set comprises various types of expressions, with which variable values are transmitted, compared, and calculated. Control statements control the processing sequence in an SCL program.

GRAPH is a programming method for sequence control processes, i.e. for controlling sequencers. The program is executed step by step. A certain number of control statements are performed in each step. You can choose to display the step enabling conditions for the next step in LAD or FBD. Alternative or parallel branching extends the scope of the linear execution of consecutive steps.

The user program consists of individual sections known as “blocks”. How the user program is structured is described in chapter 5 "The user program" on page 161. You can program each block with the programming language of your choice. LAD or FBD is well suited if primarily binary signals are linked in the block program, STL is suitable for processing complex variables with indirect addressing, and SCL allows you to easily program blocks with program branches and program loops.

A sequence control programmed with GRAPH with all process steps and step enabling conditions is implemented in a single function block. A user program can contain multiple sequence controls, each in a different function block. The rest of the user program is then in additional blocks programmed with other languages?

The user program in a CPU 1200 can be created with the LAD, FBD, and SCL programming languages. The user program in a CPU 300 or CPU 400 can be created using any programming language.

4.1 Ladder Logic LAD

Programming with LAD

In the ladder logic (LAD) programming language, you formulate the control function by arranging graphical program elements. These are essentially contacts, coils, and boxes that you connect with each other in the form of a circuit diagram.

With ladder logic you program the program in a block. You set the programming language with which the block is programmed in the block properties. A block that is programmed in LAD is divided into sections referred to as “networks”. Each network contains at least one current path that may also have an extremely complex structure. Each network is terminated by at least one coil or box.

To program, open the block and drag the program element from the Program Elements catalog to the opened network. Most program elements must be provided with variables. It is best if you initially place all program elements in a current path and subsequently label them.

Fig. 4.2 shows the structure of a block with LAD program as represented by STEP 7 inside TIA Portal. The block header (block title) and the block comment are located at the beginning of the program. Heading and comment are optional. These are followed by the first network with its number, heading, and comment. Heading and comment are also optional for the networks.

The first network shows a current path as an example with series and parallel connection of contacts, a memory function within the current path, and two coils as termination of the current path. The second network shows the processing of boxes, which can be arranged in series or parallel. A block is not terminated by a special network or function, you simply finish the program input.

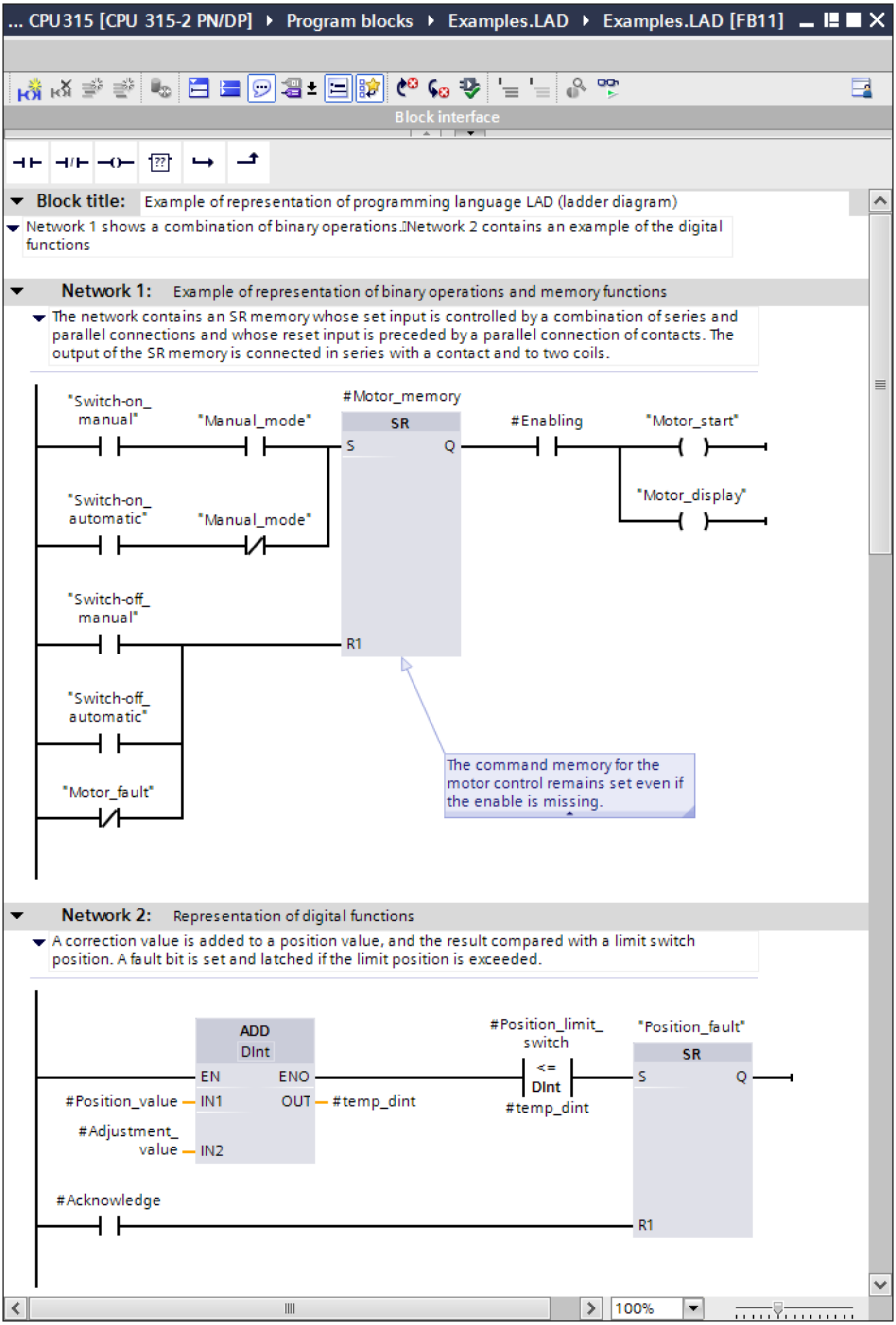
The LAD editor establishes a network in accordance with the principle of the “main current path”: This is the highest branch, which commences directly on the left-hand power rail and must be terminated by a coil or box. All LAD elements can be positioned within it.

An LAD element must not be “short-circuited” by an “empty” parallel branch, and “current” must not flow from right to left through a program element. A parallel branch that does not end “open” must be closed for the branch on which it was opened.

“Open” parallel branches can lead out from the main current path and need not lead back to the main current path; these are known as “T branches”. There are certain limitations in the selection of the permissible program elements in the case of these parallel branches which do not commence on the left-hand power rail.

Program elements of ladder logic

Fig. 4.3 shows which types of LAD elements exist. The signal state of a binary variable is queried with a contact. An NO contact takes over the signal state without



... CPU315 [CPU 315-2 PN/DP] ▸ Program blocks ▸ Examples.LAD ▸ Examples.LAD [FB11]
Block interface
100%

Block title: Example of representation of programming language LAD (ladder diagram)

Network 1 shows a combination of binary operations.Network 2 contains an example of the digital functions

Network 1: Example of representation of binary operations and memory functions

The network contains an SR memory whose set input is controlled by a combination of series and parallel connections and whose reset input is preceded by a parallel connection of contacts. The output of the SR memory is connected in series with a contact and to two coils.

The command memory for the motor control remains set even if the enable is missing.

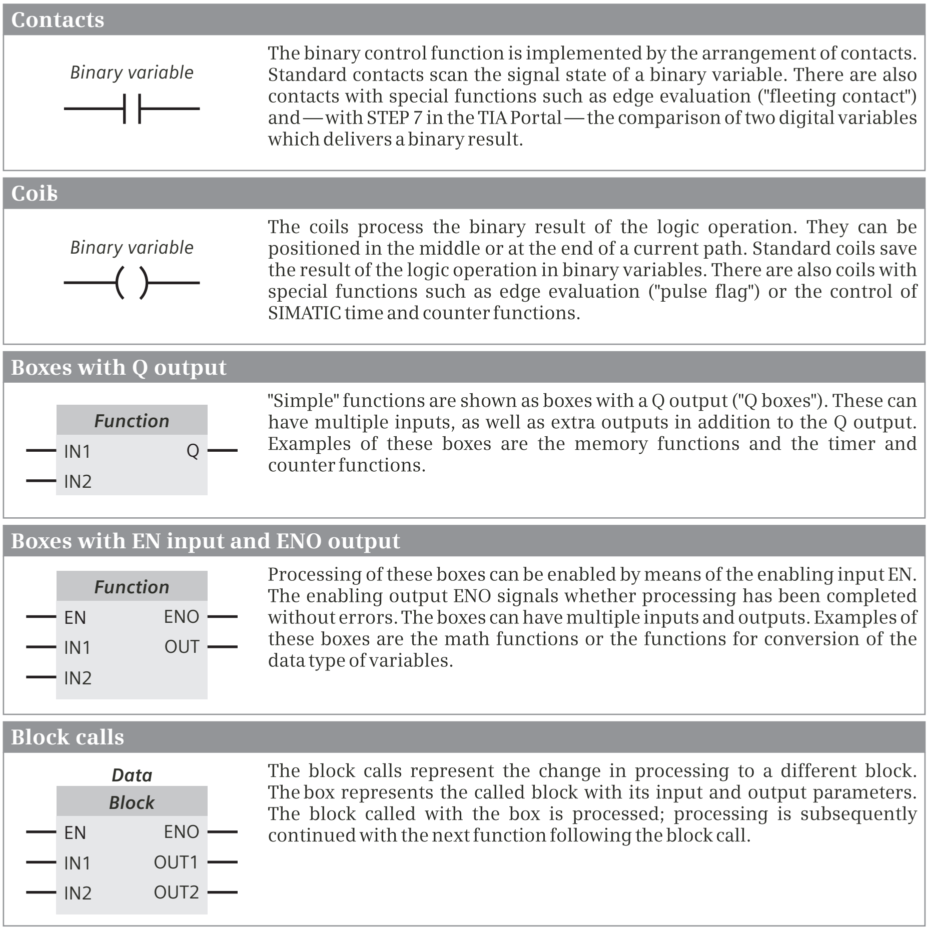
Network 2: Representation of digital functions

A correction value is added to a position value, and the result compared with a limit switch position. A fault bit is set and latched if the limit position is exceeded.

Fig. 4.2 Structure of a block with LAD program for STEP 7 inside TIA Portal

change; an NC contact negates the signal state. The arrangement of the contacts in series and parallel connection corresponds to binary logic operations according to AND and OR. The result of the logic operation is saved in a coil or processed further at the input of a box.

For the “simple” Q boxes, the signal state of the Q output must be linked further. EN/ENO boxes can be connected in series if the ENO output leads to the EN input of the following box. These are processed only if the previous box has been processed without errors. With a contact at the beginning of the “main current path” with boxes connected in series, all boxes can be simultaneously turned on and off.



Contacts

The binary control function is implemented by the arrangement of contacts. Standard contacts scan the signal state of a binary variable. There are also contacts with special functions such as edge evaluation ("fleeting contact") and—with STEP 7 in the TIA Portal—the comparison of two digital variables which delivers a binary result.

Coils

The coils process the binary result of the logic operation. They can be positioned in the middle or at the end of a current path. Standard coils save the result of the logic operation in binary variables. There are also coils with special functions such as edge evaluation ("pulse flag") or the control of SIMATIC time and counter functions.

Boxes with Q output

"Simple" functions are shown as boxes with a Q output ("Q boxes"). These can have multiple inputs, as well as extra outputs in addition to the Q output. Examples of these boxes are the memory functions and the timer and counter functions.

Boxes with EN input and ENO output

Processing of these boxes can be enabled by means of the enabling input EN. The enabling output ENO signals whether processing has been completed without errors. The boxes can have multiple inputs and outputs. Examples of these boxes are the math functions or the functions for conversion of the data type of variables.

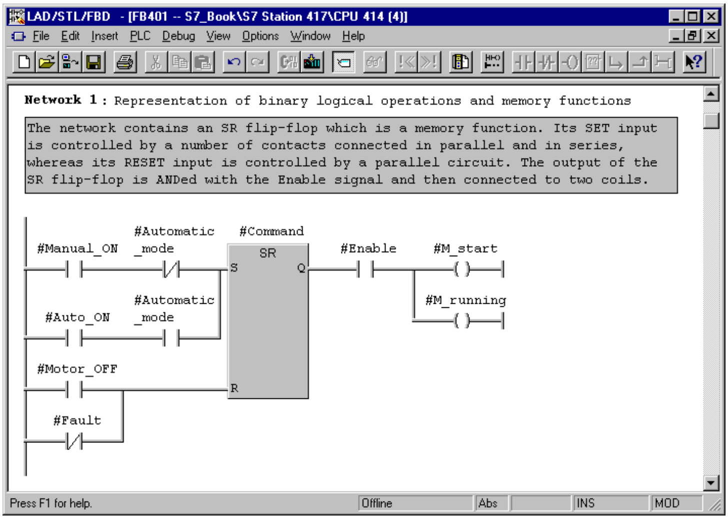
Block calls

The block calls represent the change in processing to a different block. The box represents the called block with its input and output parameters. The block called with the box is processed; processing is subsequently continued with the next function following the block call.

Fig. 4.3 Overview of ladder logic program elements

Examples of LAD representation

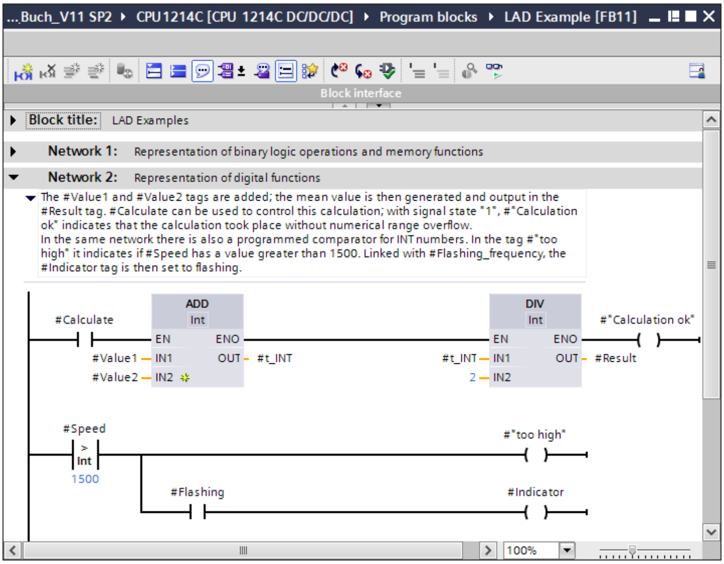
Fig. 4.4 shows how STEP 7 V5.5 represents the series and parallel connection of contacts in combination with a Q box. The example with digital functions in Fig. 4.5 shows how STEP 7 inside TIA Portal displays the ladder logic. Here it is possible to arrange additional current paths in a network.



Network 1 : Representation of binary logical operations and memory functions

The network contains an SR flip-flop which is a memory function. Its SET input is controlled by a number of contacts connected in parallel and in series, whereas its RESET input is controlled by a parallel circuit. The output of the SR flip-flop is ANDed with the Enable signal and then connected to two coils.

Fig. 4.4 Examples of series and parallel connection of contacts with STEP 7 V5.5



Block title: LAD Examples

Network 1: Representation of binary logic operations and memory functions

Network 2: Representation of digital functions

The #Value1 and #Value2 tags are added; the mean value is then generated and output in the #Result tag. #Calculate can be used to control this calculation; with signal state "1", #"Calculation ok" indicates that the calculation took place without numerical range overflow.
In the same network there is also a programmed comparator for INT numbers. In the tag #"too high" it indicates if #Speed has a value greater than 1500. Linked with #Flashing_frequency, the #Indicator tag is then set to flashing.

Fig. 4.5 Example of programming digital functions with STEP 7 inside TIA Portal

4.2 Function Block Diagram FBD

Programming with FBD

In the function block diagram (FBD) programming language, you formulate the control function by connecting boxes. FBD makes available function boxes for linking signal states, simple boxes to process the results of logic operation, and complex boxes for non-binary functions.

With FBD you program the program in a block. You set the programming language with which the block is programmed in the block properties. A block that is programmed in FBD is divided into sections referred to as “networks”. Each network contains at least one logic operation, which can also have an extremely complex structure. Each network is terminated by at least one box.

To program, open the block and drag the program element from the Program Elements catalog to the opened network. Most program elements must be provided with variables. It is best if you initially positionall program elements in a logic operation and subsequently label them.

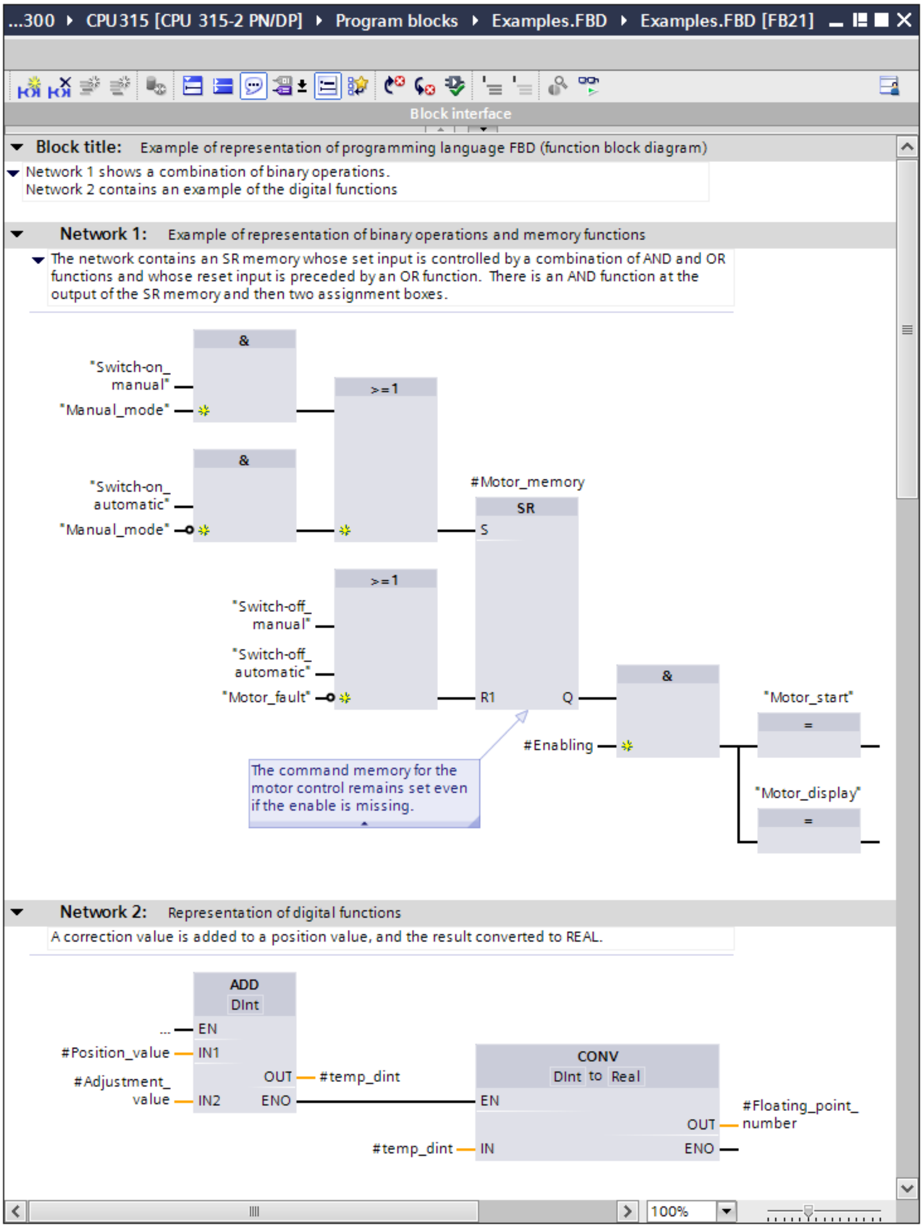
Fig. 4.6 shows the structure of a block with LAD program as represented by STEP 7 inside TIA Portal. The block header (block title) and the block comment are located at the beginning of the program. Heading and comment are optional. These are followed by the first network with its number, heading, and comment. Heading and comment are also optional for the networks.

The first network shows a logic operation as example with AND and OR boxes, a memory function within the logic operation, and two assignments as termination of the logic operation. The second network shows the processing of EN/ENO boxes, of which two are arranged in series. A block is not terminated by a special network or function, you simply finish the program input.

The program editor constructs an FBD network from left to right: Position the first program element underneath the network comment and insert further program elements at the inputs and outputs. The boxes with binary logic operations can be extended by additional inputs. Box outputs cannot be directly connected to each other.

A logic operation must always be terminated, for example by an assignment. The assignment controls a binary variable using the result of the logic operation.

“Open” parallel branches can lead out from the top logic operation and not be “wired back” to the top logic operation; these are known as “T branches”. In these T branches, there are certain limitations with regard to which permissible program elements can be selected.



...300 ▸ CPU315 [CPU 315-2 PN/DP] ▸ Program blocks ▸ Examples.FBD ▸ Examples.FBD [FB21]

Block interface

Block title: Example of representation of programming language FBD (function block diagram)

Network 1 shows a combination of binary operations.
Network 2 contains an example of the digital functions

Network 1: Example of representation of binary operations and memory functions

The network contains an SR memory whose set input is controlled by a combination of AND and OR functions and whose reset input is preceded by an OR function. There is an AND function at the output of the SR memory and then two assignment boxes.

The command memory for the motor control remains set even if the enable is missing.

Network 2: Representation of digital functions

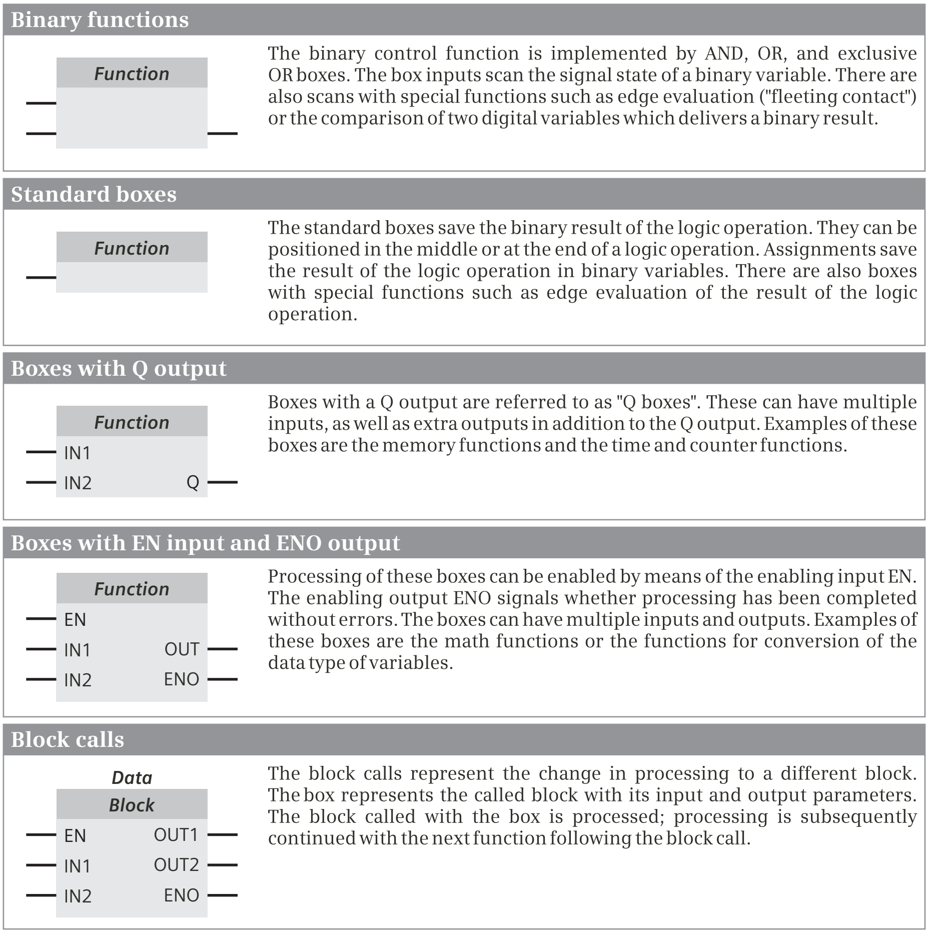
A correction value is added to a position value, and the result converted to REAL.

Fig. 4.6 Structure of a block with FBD program for STEP 7 inside TIA Portal

Program elements of the function block diagram

Fig. 4.7 shows which types of FBD elements exist. The signal state of a binary variable is queried with an input on a box. An input fed directly to the box imports the signal state without change; a negation symbol (a small circle) at the box input negates the signal state. The AND, OR and exclusive OR boxes can be interconnected as required. The result of the logic operation is further processed at the input of the next box, for example stored with an assign box in a variable.

For the “simple” Q boxes, the signal state of the Q output must be linked further. EN/ENO boxes can be connected in series if the ENO output leads to the EN input of the following box. These are processed only if the previous box has been processed without errors.



Binary functions

The binary control function is implemented by AND, OR, and exclusive OR boxes. The box inputs scan the signal state of a binary variable. There are also scans with special functions such as edge evaluation ("fleeting contact") or the comparison of two digital variables which delivers a binary result.

Standard boxes

The standard boxes save the binary result of the logic operation. They can be positioned in the middle or at the end of a logic operation. Assignments save the result of the logic operation in binary variables. There are also boxes with special functions such as edge evaluation of the result of the logic operation.

Boxes with Q output

Boxes with a Q output are referred to as "Q boxes". These can have multiple inputs, as well as extra outputs in addition to the Q output. Examples of these boxes are the memory functions and the time and counter functions.

Boxes with EN input and ENO output

Processing of these boxes can be enabled by means of the enabling input EN. The enabling output ENO signals whether processing has been completed without errors. The boxes can have multiple inputs and outputs. Examples of these boxes are the math functions or the functions for conversion of the data type of variables.

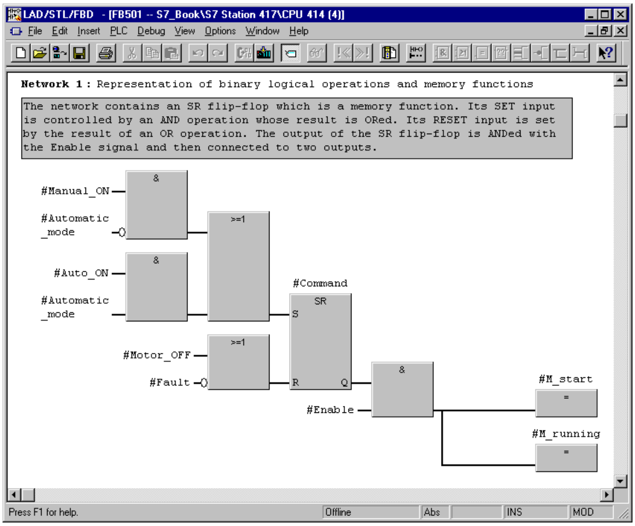
Block calls

The block calls represent the change in processing to a different block. The box represents the called block with its input and output parameters. The block called with the box is processed; processing is subsequently continued with the next function following the block call.

Fig. 4.7 Overview of program elements of the function block diagram

Examples of FBD representation

Fig. 4.8 shows how STEP 7 V5.5 represents a Q box with binary logic operations at the inputs and at the output in a function block diagram. How STEP 7 inside TIA Portal displays the function block diagram is shown in Fig. 4.9. Here, further (independent) logic operations can be arranged in a network.

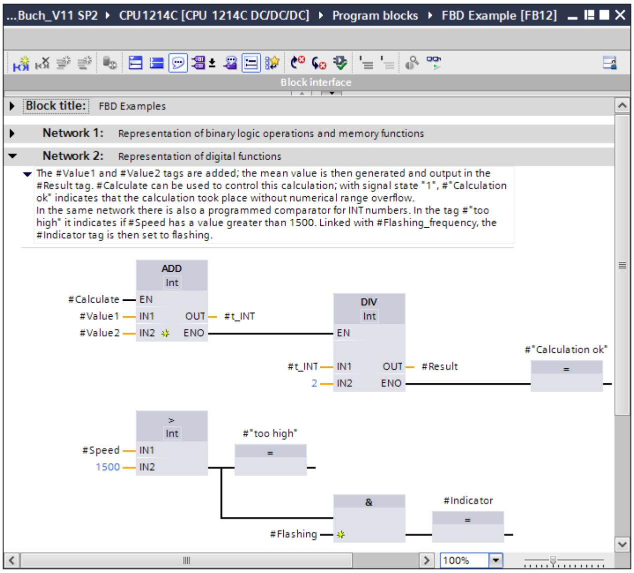


LAD/STL/FBD - [FB501 -- S7_Book\S7 Station 417\CPU 414 (4)]

Network 1 : Representation of binary logical operations and memory functions

The network contains an SR flip-flop which is a memory function. Its SET input is controlled by an AND operation whose result is ORed. Its RESET input is set by the result of an OR operation. The output of the SR flip-flop is ANDed with the Enable signal and then connected to two outputs.

Fig. 4.8 Example of binary functions in FBD for STEP 7 V5.5



...Buch_V11 SP2 ▸ CPU 1214C [CPU 1214C DC/DC/DC] ▸ Program blocks ▸ FBD Example [FB12]

Block title: FBD Examples

Network 1: Representation of binary logic operations and memory functions

Network 2: Representation of digital functions

The #Value1 and #Value2 tags are added; the mean value is then generated and output in the #Result tag. #Calculate can be used to control this calculation; with signal state "1", #"Calculation ok" indicates that the calculation took place without numerical range overflow.
In the same network there is also a programmed comparator for INT numbers. In the tag #"too high" it indicates if #Speed has a value greater than 1500. Linked with #Flashing_frequency, the #Indicator tag is then set to flashing.

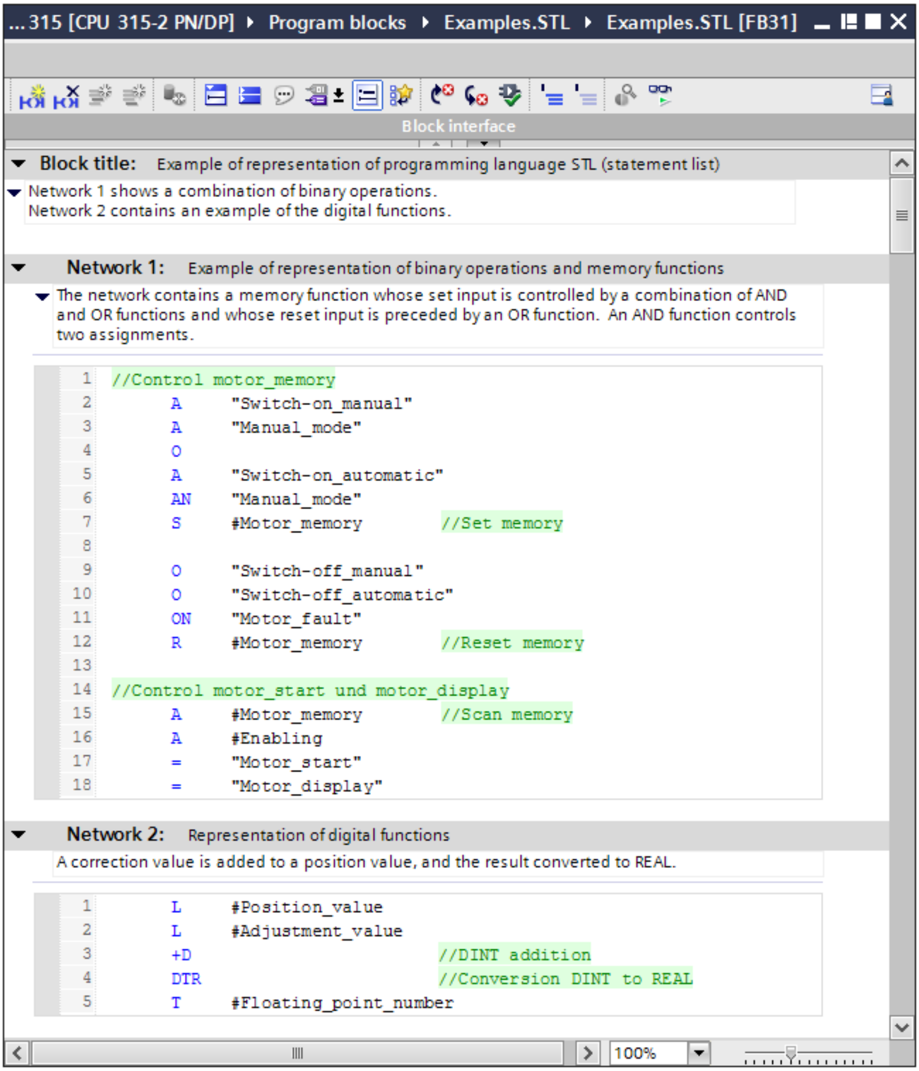
Fig. 4.9 Example of digital functions in FBD for STEP 7 inside TIA Portal

4.3 Statement List STL

Programming with STL

In the statement list (STL) programming language you formulate the control function by entering STL statements line by line. Each line holds one statement, such as querying and connecting to an input signal or adding two floating point numbers. Block calls require additional lines for the parameter list.

With STL you program the program in a block. You set the programming language with which the block is programmed in the block properties. A block that is pro-


...315 [CPU 315-2 PN/DP] ▸ Program blocks ▸ Examples.STL ▸ Examples.STL [FB31]
Block interface
100%

Block title: Example of representation of programming language STL (statement list)

Network 1 shows a combination of binary operations.
Network 2 contains an example of the digital functions.

Network 1: Example of representation of binary operations and memory functions

The network contains a memory function whose set input is controlled by a combination of AND and OR functions and whose reset input is preceded by an OR function. An AND function controls two assignments.

```
//Control motor_memory
      A     "Switch-on_manual"
      A     "Manual_mode"
      O
      A     "Switch-on_automatic"
      AN    "Manual_mode"
      S     #Motor_memory          //Set memory

      O     "Switch-off_manual"
      O     "Switch-off_automatic"
      ON    "Motor_fault"
      R     #Motor_memory          //Reset memory

//Control motor_start und motor_display
      A     #Motor_memory          //Scan memory
      A     #Enabling
      =     "Motor_start"
      =     "Motor_display"
```

Network 2: Representation of digital functions

A correction value is added to a position value, and the result converted to REAL.

```
      L     #Position_value
      L     #Adjustment_value
      +D                           //DINT addition
      DTR                          //Conversion DINT to REAL
      T     #Floating_point_number
```

Fig. 4.10 Structure of a block with STL program for STEP 7 inside TIA Portal

grammed in STL can be divided into sections referred to as “networks”. Each network can contain any program. Networks are not required for the function of an STL program, but they do increase clarity: When processing the block program, you can go directly to any network to open it.

For programming, open the block and use the keyboard to enter the individual STL statements line by line. You can drag complex functions with multiple inputs and outputs from the Program Elements catalog to the desired line. Even though working with the Program Elements catalog can be more complicated than entering via the keyboard, it gives you an overview of the available functions.

Comments commence with two slashes, either as a line comment or as a statement comment. You can insert empty lines to structure the sequence of statements. These and the comments have no effect on the control function.

Fig. 4.10 shows the structure of a block with the STL program. The block header (block title) and the block comment are located at the beginning of the program. These are followed by the first network with its number, heading, and comment. Further networks are optional.

The first network shows a logic operation as example with AND and OR statements, a memory function, as well as an AND function with two assignments as termination. The second network shows the processing of digital values. Two digital values with data type DINT are added and the result converted to REAL before being transferred to a variable. A block need not be terminated by a special function, you simply terminate the program input.

Structure of an STL statement

The STL program consists of a sequence of individual STL statements. A statement is the smallest independent unit of the user program. It represents a procedural specification for the control processor. Fig. 4.11 shows the structure of an STL statement.

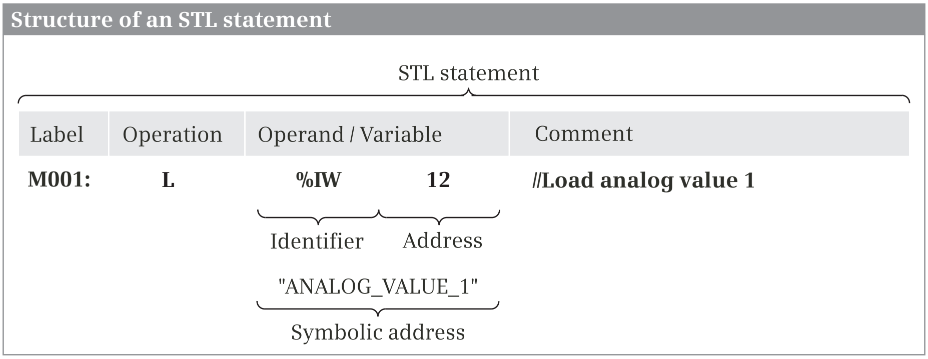


Fig. 4.11 Structure of an STL statement

An STL statement consists of

- ▷ A jump label (optional), which must end with a colon.
- ▷ An operation that describes what the control processor has to do (e.g. load, query, logic operation according to AND, compare, etc.).
- ▷ An address that contains the information necessary for executing the operation (e.g. an absolutely addressed address %IW12, a symbolically addressed variable ANALOGVALUE_1, a constant W#16#F001, a jump label, etc.). The address can also be omitted depending on the operation.
- ▷ A comment (optional), commenced by two slashes and up to the end of the line (only printable characters, no tabulators).

With a block call, the call operation is followed by the parameter list in round brackets.

Processing of a binary logic operation, operation step

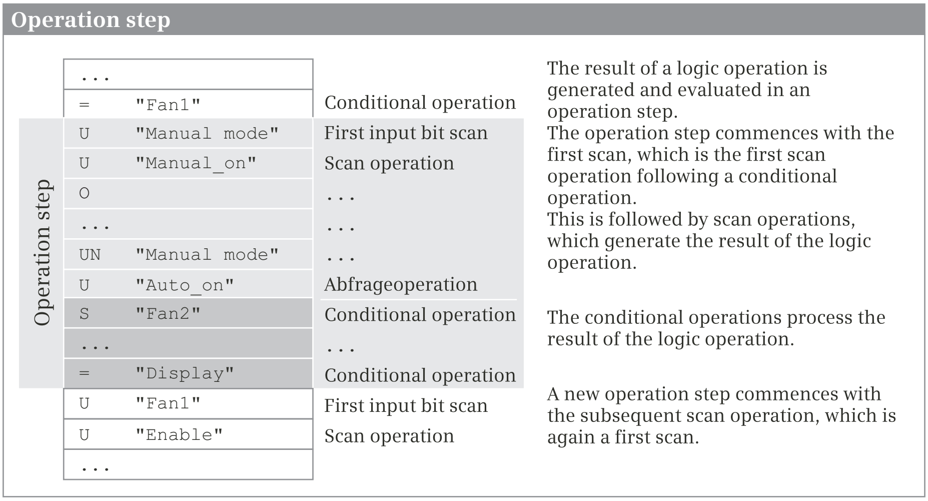
A binary logic operation consists of scan operations and conditional operations. The sequence of scan operations and subsequent conditional operations is referred to as an operation step (Fig. 4.12).

The first scan operation processed following a conditional operation is the *first input bit scan*. This is of special significance because the control processor directly imports the scan result of this statement as the result of a logic operation. The “old” result of a logic operation is thus lost. The first input bit scan always represents the beginning of a logic operation. The logic operation (AND, OR, exclusive OR) specified in the first input bit scan does not play any role here.

The result of the logic operation is generated by the *scan operations*. You query the signal state of a binary address for “1” or “0” and link it according to AND, OR, or exclusive OR. The result of this logic operation is saved by the control processor as the new result of logic operation.

Conditional operations are operations whose execution depends on the result of logic operation. These are operations for assigning, setting and resetting binary addresses, for starting timers and counters, etc. The conditional operations (apart from a few exceptions) are executed if the result of logic operation (RLO) is “1” and not executed if RLO is “0”. They do not change the RLO (apart from a few exceptions), and therefore the RLO is the same for several successive conditional operations.

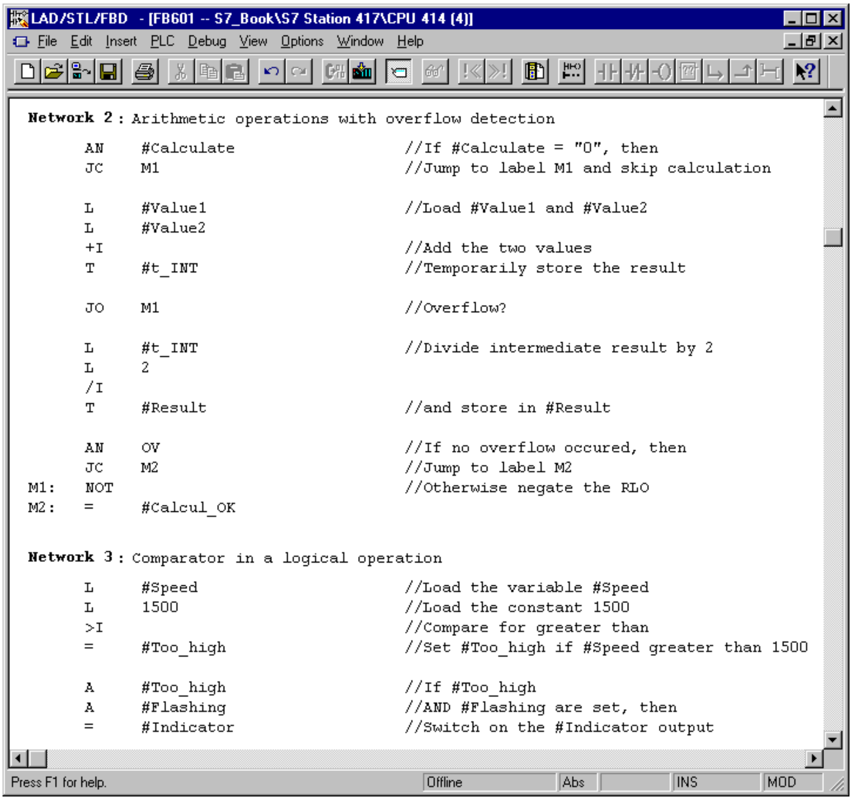
The example with digital functions and jump functions in Fig. 4.13 shows how STEP 7 V5.5 displays the statement list.



Operation step

...		
= "Fan1"	Conditional operation	The result of a logic operation is generated and evaluated in an operation step.
U "Manual mode"	First input bit scan	The operation step commences with the first scan, which is the first scan operation following a conditional operation.
U "Manual_on"	Scan operation	This is followed by scan operations, which generate the result of the logic operation.
O	...	
...	...	
UN "Manual mode"	...	
U "Auto_on"	Abfrageoperation	
S "Fan2"	Conditional operation	The conditional operations process the result of the logic operation.
...	...	
= "Display"	Conditional operation	
U "Fan1"	First input bit scan	A new operation step commences with the subsequent scan operation, which is again a first scan.
U "Enable"	Scan operation	
...		

Fig. 4.12 Binary logic operation with STL, definition of operation step



LAD/STL/FBD - [FB601 -- S7_Book\S7 Station 417\CPU 414 (4)]
File Edit Insert PLC Debug View Options Window Help
Press F1 for help.
Offline
Abs
INS
MOD

```
Network 2 : Arithmetic operations with overflow detection

       AN    #Calculate                //If #Calculate = "0", then
       JC    M1                        //Jump to label M1 and skip calculation

       L     #Value1                   //Load #Value1 and #Value2
       L     #Value2
       +I                              //Add the two values
       T     #t_INT                    //Temporarily store the result

       JO    M1                        //Overflow?

       L     #t_INT                    //Divide intermediate result by 2
       L     2
       /I
       T     #Result                   //and store in #Result

       AN    OV                        //If no overflow occured, then
       JC    M2                        //Jump to label M2
M1:    NOT                             //Otherwise negate the RLO
M2:    =     #Calcul_OK

Network 3 : Comparator in a logical operation

       L     #Speed                    //Load the variable #Speed
       L     1500                      //Load the constant 1500
       >I                              //Compare for greater than
       =     #Too_high                 //Set #Too_high if #Speed greater than 1500

       A     #Too_high                 //If #Too_high
       A     #Flashing                 //AND #Flashing are set, then
       =     #Indicator                //Switch on the #Indicator output
```

Fig. 4.13 STL representation with STEP 7 V5.5

4.4 Structured Control Language SCL

In the SCL programming language you formulate the control function by entering SCL statements line by line. The SCL statements are essentially expressions for transmitting, linking, and converting the variable values and control statements to control program execution.

With SCL you program the program in a block. You set the programming language with which the block is programmed in the block properties. A further division of the block program into networks as with LAD, FBD or STL is not possible in SCL.

Programming with SCL in STEP 7 V5.5

In STEP 7 V5.5, the SCL programming language is an option package that is integrated into SIMATIC Manager during installation. With SCL, your programming is source-oriented: You create a program source file and then compile it into an executable program code.

You create a new SCL program source file for programming. In the SIMATIC Manager, select the *Sources* folder under the *S7 Program* folder and select *Insert* > *S7 Software* > *SCL Source*. Double-click on the program source file to open it. In a program source file, you use specified keywords in a specific order for the properties of blocks and the program. To see how a program source file is structured and which keywords there are, refer to chapter 3.11 "Working with program source files" on page 90.

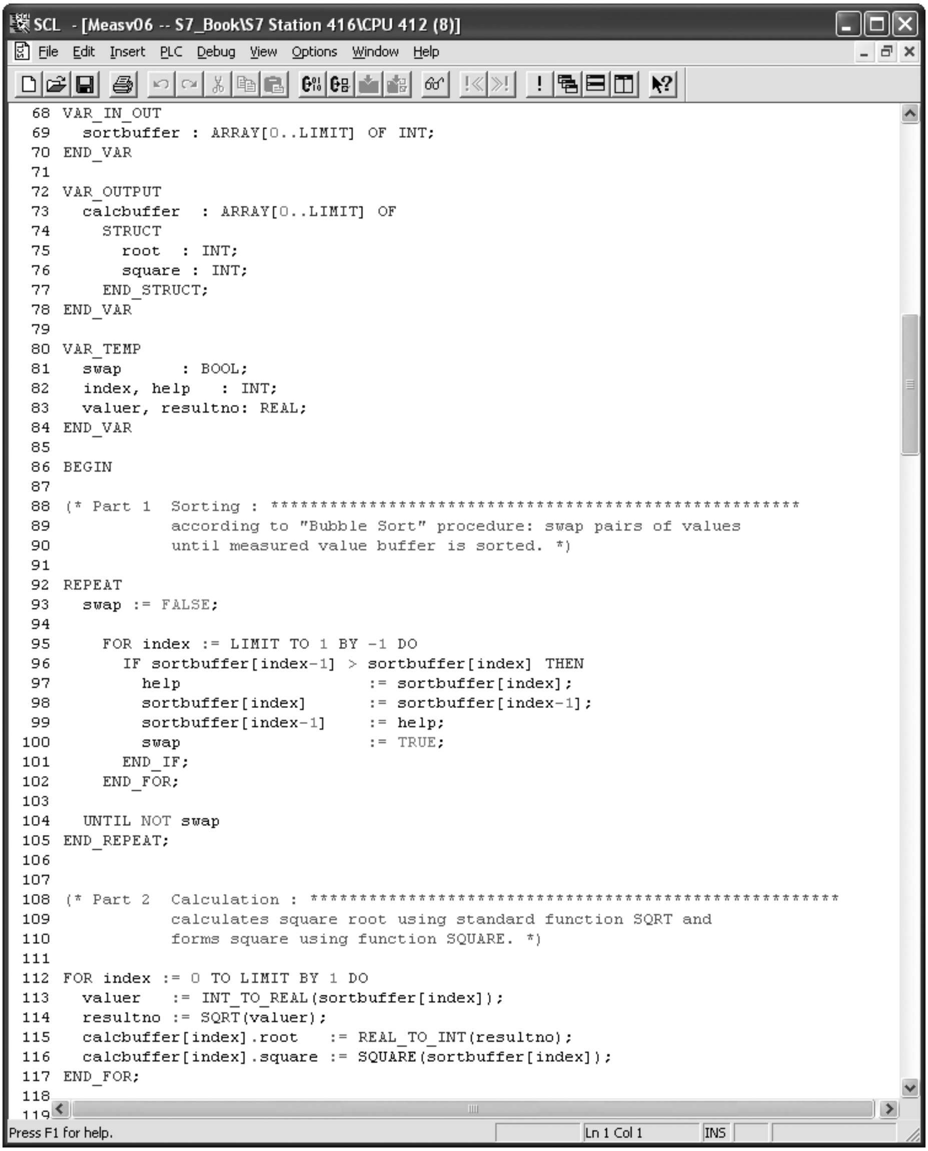
The program editor helps you to enter keywords. *Insert* > *Block Call* allows you to insert the call of a pre-programmed user block or a system block at the cursor in the source text, *Insert* > *Block Template* > ... facilitates the creation of new blocks, and *Insert* > *Control Structure* > ... allows you to create control statements in the source text.

Compiling the program source file generates executable blocks. You create the settings for all compilation processes in the SIMATIC Manager under *Options* > *Settings* in the *Compiler* tab or with special keywords directly in the program source file. You start compilation with the program source file open, using the menu option *File* > *Compile*. With a compilation control file, you can compile multiple program source files in the chosen order on a single event.

Fig. 4.14 shows an example of an SCL program source file that is created with the program editor of STEP 7 V5.5.

Programming with SCL in STEP 7 inside TIA Portal

For programming, open the block and use the keyboard to enter the individual SCL statements line by line. Each SCL statement is concluded by a semicolon. You can write several statements in one line, or one statement can occupy several lines. You can drag complex functions with multiple inputs and outputs from the Program Elements catalog to the desired line. Even though working with the Pro-

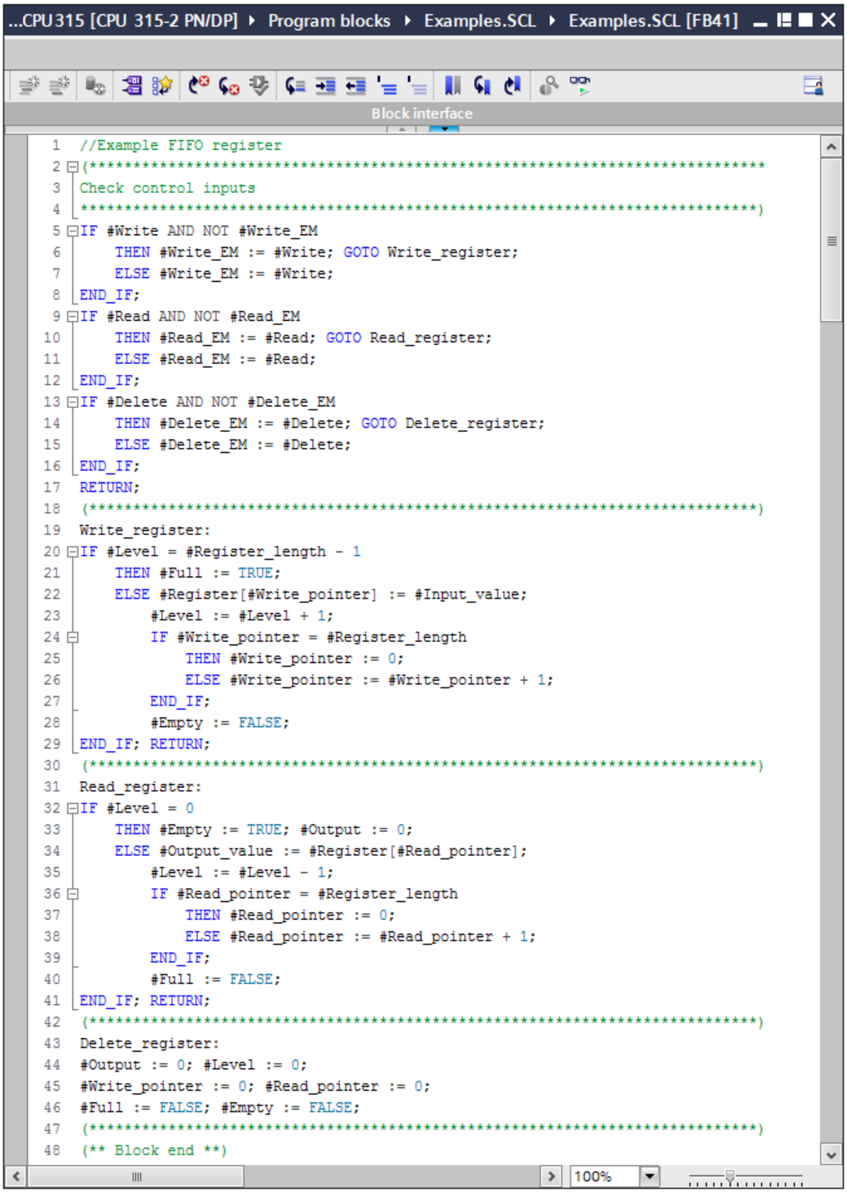

SCL - [Measv06 -- S7_Book\S7 Station 416\CPU 412 (8)]
File Edit Insert PLC Debug View Options Window Help
Press F1 for help.
Ln 1 Col 1
INS

```
VAR_IN_OUT
  sortbuffer : ARRAY[0..LIMIT] OF INT;
END_VAR

VAR_OUTPUT
  calcbuffer  : ARRAY[0..LIMIT] OF
    STRUCT
      root   : INT;
      square : INT;
    END_STRUCT;
END_VAR

VAR_TEMP
  swap       : BOOL;
  index, help    : INT;
  valuer, resultno: REAL;
END_VAR

BEGIN

(* Part 1  Sorting : ******************************************************
           according to "Bubble Sort" procedure: swap pairs of values
           until measured value buffer is sorted. *)

REPEAT
  swap := FALSE;

    FOR index := LIMIT TO 1 BY -1 DO
      IF sortbuffer[index-1] > sortbuffer[index] THEN
        help                    := sortbuffer[index];
        sortbuffer[index]       := sortbuffer[index-1];
        sortbuffer[index-1]     := help;
        swap                    := TRUE;
      END_IF;
    END_FOR;

  UNTIL NOT swap
END_REPEAT;


(* Part 2  Calculation : ******************************************************
           calculates square root using standard function SQRT and
           forms square using function SQUARE. *)

FOR index := 0 TO LIMIT BY 1 DO
  valuer   := INT_TO_REAL(sortbuffer[index]);
  resultno := SQRT(valuer);
  calcbuffer[index].root    := REAL_TO_INT(resultno);
  calcbuffer[index].square := SQUARE(sortbuffer[index]);
END_FOR;
```

Fig. 4.14 Example of an SCL program with STEP 7 V5.5

gram Elements catalog can be more complicated than entering via the keyboard, it gives you an overview of the available functions.

You can make the SCL program clearer and easier to read by using comments and empty lines. Comments and empty lines have no influence on the function of the SCL program. Line comments commence with two slashes and terminate at the end of the line. Block comments commence with left parenthesis and asterisk, can extend over several lines, and terminate with asterisk and right parenthesis.


...CPU315 [CPU 315-2 PN/DP] ▸ Program blocks ▸ Examples.SCL ▸ Examples.SCL [FB41]
Block interface
100%

```
//Example FIFO register
(***************************************************************************
Check control inputs
**************************************************************************)
IF #Write AND NOT #Write_EM
    THEN #Write_EM := #Write; GOTO Write_register;
    ELSE #Write_EM := #Write;
END_IF;
IF #Read AND NOT #Read_EM
    THEN #Read_EM := #Read; GOTO Read_register;
    ELSE #Read_EM := #Read;
END_IF;
IF #Delete AND NOT #Delete_EM
    THEN #Delete_EM := #Delete; GOTO Delete_register;
    ELSE #Delete_EM := #Delete;
END_IF;
RETURN;
(*************************************************************************)
Write_register:
IF #Level = #Register_length - 1
    THEN #Full := TRUE;
    ELSE #Register[#Write_pointer] := #Input_value;
        #Level := #Level + 1;
        IF #Write_pointer = #Register_length
            THEN #Write_pointer := 0;
            ELSE #Write_pointer := #Write_pointer + 1;
        END_IF;
        #Empty := FALSE;
END_IF; RETURN;
(*************************************************************************)
Read_register:
IF #Level = 0
    THEN #Empty := TRUE; #Output := 0;
    ELSE #Output_value := #Register[#Read_pointer];
        #Level := #Level - 1;
        IF #Read_pointer = #Register_length
            THEN #Read_pointer := 0;
            ELSE #Read_pointer := #Read_pointer + 1;
        END_IF;
        #Full := FALSE;
END_IF; RETURN;
(*************************************************************************)
Delete_register:
#Output := 0; #Level := 0;
#Write_pointer := 0; #Read_pointer := 0;
#Full := FALSE; #Empty := FALSE;
(*************************************************************************)
(** Block end **)
```

Fig. 4.15 Example of a block with SCL program for STEP 7 inside TIA Portal

Fig. 4.15 shows the SCL program for a FIFO register. With a rising edge at *#Write*, this block writes the value present at the *#Input* parameter into a FIFO register. With a rising edge at *#Read*, the value at *#Output* is output again. The values are read out in the order in which they were written into the register (FIFO, first in first out). The register can be emptied using *#Delete*. The two displays *#Full* and

#Empty show the status of the register (*#Full* and *#Empty* are each set following writing or reading). The block works with a write pointer and a read pointer.

4.5 S7-GRAPH sequence control

In the case of sequence controls, static assignment of the input signals to the outputs does not predominate (as with logic controls), but rather their time sequence. The control procedures executed in succession are divided into sequence steps, or steps for short. A step contains one or more actions such as switch motor on or off. Only the actions of an active (processed) step are carried out. Progression to the next step is carried out by means of transitions (step enabling conditions). The transition can be process-dependent, e.g. as a result of signals from the controlled machine or plant, or time-dependent, e.g. following expiry of a delay time.

A sequential control, or “sequencer”, starts with an initial step. There can be several in a sequencer. In a linear sequencer this is followed by alternate transitions and steps. In addition to the linear sequence – one step followed by another single step – there are also branches. In an alternative branch (OR branch) only one of the choice of sub-sequencers is executed, in a parallel branch (AND branch) all the sub-sequencers are executed.

An *interlock* condition is specific to a step. If the interlock condition is satisfied, the instructions depending on the interlock are carried out for the active step. A *supervision* is a monitoring condition that is specific to a step. If the supervision condition is satisfied, a fault is present. *Permanent instructions* are program components that are processed in every cycle, independent of the status of the sequence control. Permanent pre-instructions are processed prior to the sequence control, post-instructions after the sequence control.

Fig. 4.17 shows an example of the working window of the GRAPH Editor in STEP 7 inside TIA Portal. The sequencer is shown in the GRAPH navigation on the left side and the working area shows step S2 with the transition T2 as selected in the navigation.

A sequence control consists of a function block and a data block. The function block controls the sequence of the steps and transitions; the data block is the instance data block of the sequencer function block and contains the structure of the sequencer and the associated data. A sequence control can contain several independent sequencers.

Programming a sequence control with STEP 7 V5.5

In STEP 7 V5.5, the GRAPH programming language is an option package that is integrated into SIMATIC Manager during installation. To program a sequence control, select the *Blocks* folder and use *Insert* > *S7 Program* > *Function Block* to create a function block. In the displayed Properties window, set *GRAPH* as the programming language. Then double-click the block to start the GRAPH editor.

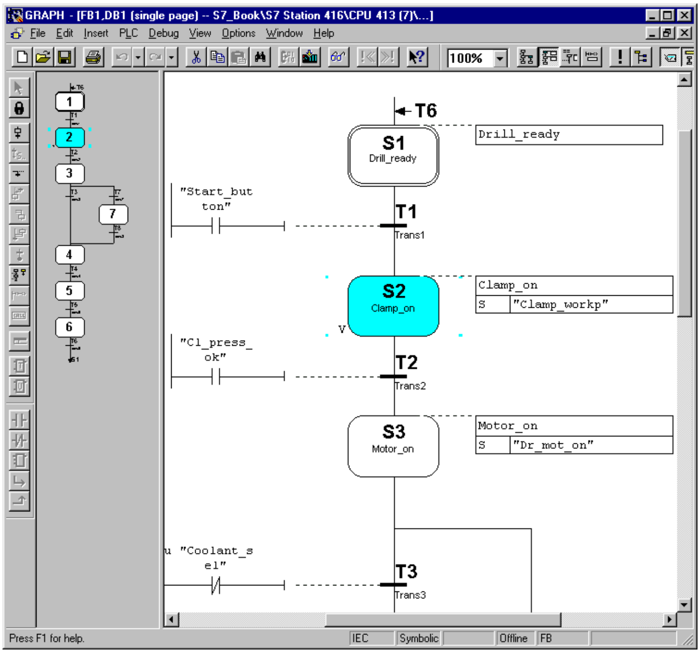


Fig. 4.16 Example for displaying a sequencer in STEP 7 V5.5

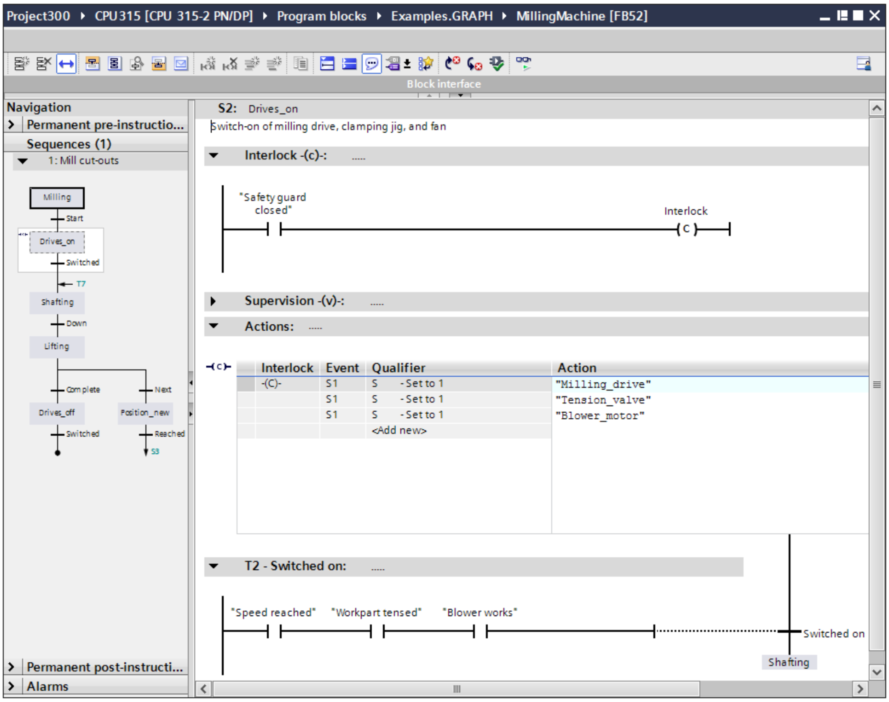
The GRAPH editor inserts the first step and the first transition into an empty block. First create the structure of the sequencer. You have two options: In the *Insert* > *Direct* editing mode, place a sequencer icon, e.g. a step/transition pair, in the selected location; in the *Insert* > *Preselection* editing mode, drag the icon to the desired location.

You now provide the individual sequence steps with actions, which you insert to the right of the step under the step description. You specify the address to be controlled and the operation. Examples: S for set, R for reset, N for non holding (the address is set only as long as the step is active), and D for delay (the address is set only after a defined time elapses and is reset when the step is deactivated). You can program the transitions either in ladder logic (LAD) or in a function block diagram (FBD).

Programming a sequence control with STEP 7 inside TIA Portal

Specify the properties of the sequence control prior to programming. Select the *Options* > *Settings* command in the main menu and click on *GRAPH* in the *PLC programming* group. Now you can set the sequencer properties such as time monitoring for the sequence steps.

To program a sequence control, add a new function block for which you set the language to *GRAPH*. The open block then shows the GRAPH navigation with an overview presentation of the sequencer in the left part of the working window, and



S2: Drives_on

Switch-on of milling drive, clamping jig, and fan

Interlock -(c)-: .....

Supervision -(v)-: .....

Actions: .....

Interlock	Event	Qualifier	Action
-(C)-	S1	S - Set to 1	"Milling_drive"
	S1	S - Set to 1	"Tension_valve"
	S1	S - Set to 1	"Blower_motor"
		<Add new>	

T2 - Switched on: .....

Fig. 4.17 Example of a step in a sequencer for STEP 7 inside TIA Portal

the selected step with the associated step enabling condition in the right-hand part.

If you click on the *Sequence View* icon in the working window, you can edit the sequence structure. Drag the desired program element – a step with transition, a jump, or a branch – from the Program Elements catalog to the working area and create in this way the structure of the sequencer.

To program a step, click the *Single Step View* icon, select the step, and enter the actions in a table, for example, setting a binary variable or starting a time function. The step enabling conditions can be programmed in ladder logic (LAD) or a function block diagram (FBD) – depending on the settings in the sequencer properties.

4.6 The function library of LAD, FBD, and STL

The function library of LAD, FBD, and STL is roughly divided into two areas:

- The basic instructions directly manipulate the variable values. Examples of these are instructions for transferring a variable value from one storage location to another, for the logical operation of two signal states, for the comparison of two numbers, or for the calculation of a sum.

▷ The extended instructions are generally system and standard blocks that, depending on the function, are treated in the programming interface as a simple instruction or a block call. Among these are functions for technological applications such as PID controllers and communications functions for data transmission to other programmable controllers.

Fig. 4.18 shows an overview of the “simple” statements for LAD, FBD, and STL.

Basic functions

The basis functions enable you to program the programmable controller in a certain “basic functionality” in the functional scope of contactor or relay controllers or wired logic circuits.

The logic operations AND, OR, and XOR can be simulated in the ladder logic with the series and parallel connection of contacts. Each control function can be programmed together with the negation of the signal state or the result of the logic operation (the “current flow” in the ladder logic). A change in the result of logic operation or signal state can be detected with an edge evaluation and be linked further.

Timer and counter functions are implemented by means of system blocks. With a CPU 300/400, the address areas of the SIMATIC time functions and SIMATIC counter functions can be used.

Digital functions

The digital functions process variable values (bit pattern, numerical values). The numerical values are available as fixed-point or floating point numbers, which are compared to each other, linked, and converted as needed into other data types. The shift functions and the word logic operations allow digital values (bit patterns) to be manipulated bit by bit.

The “extended” statements provide additional functions for handling time values and character strings (STRING variables). For example, a period of time can be added to a time of day or part of a character string can be replaced with another character string.

Functions for program control

The jump functions and block functions make it possible to exit the linear program execution, depending on conditions. The step functions are used in the program within a block. Block functions are used to continue or end program execution in the called block.

Additional functions for statement list

STL is a “machine-level” language, with which the contents of the processor registers can be manipulated. This applies to the contents of the accumulators (arithmetic registers) and the data block registers (responsible for opening the

Overview of the basic instructions for LAD, FBD, and STL

Basic functions	Work mainly with binary signal states
Bit logic	Links the signal states of binary variables according to an AND, OR, and exclusive OR function (with LAD: series and parallel connection) and generate the result of logic operation
Memory functions	Transfer the result of the logic operation to a binary variable and save it
Edge evaluations	Detect a positive or negative change of a result of logic operation or signal state change
Time functions	Control time flows such as wait and monitoring times, measure time periods, or generate pulses
Counter functions	Count pulses both up with increasing count value and down with decreasing count value
Digital functions	Work mainly with digital values
Transfer functions	Transfer values (bit patterns, numbers, characters) between variables or between memory areas
Comparison functions	Compare two digital values and generate a binary comparison result
Arithmetic functions	Apply the basic arithmetical operations to two numerical values (add, subtract, multiply, divide)
Mathematical functions	Calculate a numerical value according to a math function (trigonometric functions, logarithm, etc.)
Conversion functions	Convert a numerical value into a different data type, generation of absolute value and two's complement
Shift functions	Shift the content of a digital variable bit-by-bit to the left or right
Word logic operations	Links two digital values bit-by-bit according to AND, OR, and exclusive OR
Program control	Controls the sequence of program processing
Jump functions	Control program processing by branching independent of or dependent on conditions
Block functions	Call logic blocks and terminate logic blocks, open data blocks
Master control relay	Controls the output of binary signal states and digital values depending on the conditions in a program section
STL functions	Further functions for STL
Accumulator functions	Manipulate the contents of the accumulators (register in the control processor)
Indirect addressing	Calculate the address of a variable during runtime (dynamic addressing), permit the manipulation of complex variables

Fig. 4.18 Function library of the “simple” statements for LAD, FBD, and STL

data blocks). In addition, STL has statements for determining the memory address of complex variables and local variables to be able to change the contents of variables.

4.7 The function library of SCL

The SCL program consists of a sequence of individual statements, which are shown in Fig. 4.19. A *value assignment* transfers the result of an expression to a variable. In an *expression*, the variable values are linked together through *operators*. *Control statements* guide program execution, for example with program loops. *Block calls* are used to continue program execution in the called block.

Operators

An expression represents a value. It can comprise a single address (a single variable) or several addresses (variables) which are linked by operators.

Example: “a + b” is an expression; “a” and “b” are addresses, “+” is the operator.

The sequence of logic operations is defined by the priority of the operators and can be controlled by parentheses. Mixing of expressions is permissible providing the data types generated during calculation of the expression permit this.

SCL provides the operators specified in Table 4.1. Operators of equal priority are processed from left to right.

Expressions

An expression is a formula for calculating a value and consists of addresses (variables) and operators. In the simplest case, an expression is an address, a variable, or a constant. A sign or a negation can also be included.

An expression can consist of addresses that are linked together by operators. Expressions can also be linked by operators. Expression can therefore have a very complex structure. Parentheses can be used to control the processing sequence in an expression. The result of an expression can be assigned to a variable or a block parameter or used as a condition in a control statement. Expressions are distinguished according to the type of logic operation into arithmetic expressions, comparison expressions, and logic expressions.

SCL functions

For processing digital values, SCL provides transfer functions, comparison functions, arithmetic functions, mathematic functions, conversion functions, shift functions, word logic operations, and functions for processing time values and character strings.

In SCL, you can also use each function (each FC block) with a function value as a “real” SCL function in an expression. Example: The function with the name *adder* –

SCL statements

SCL statement

An SCL statement consists of a jump label with subsequent colon and the actual statement, which is terminated by a semicolon. The statement can extend over several lines. The statement can be followed by a (line) comment, which is commenced by two slashes and extends up to the end of the line. The jump label and the line comment can be omitted.

General SCL statement

Label	:	SCL statement	;	//	Comment

Value assignment

A value assignment transfers the value of an expression to a variable. An expression can be a single variable or a formula for calculating a value. A formula links the variables by means of operators. Depending on the type of logic operation, a distinction is made between arithmetic expressions, comparison expressions, and logical expressions.

Value assignment with assignment operator

Label	:	Variable	:=	Expression	;	//	Comment
		#Result	:=	#Variable **AND** #Variable	;	Logical expression	
		#Result	:=	#Variable **>=** #Variable	;	Comparison expression	
		#Result	:=	#Variable **+** #Variable	;	Arithmetic expression	

Control statement

A control statement controls the processing sequence in the program by means of branching and program loops which are be processed repeatedly. A control statement commences with a keyword (xxx) and is terminated by END_xxx.

Control statement

Label	:	**xxx**	Statement list	**END_xxx**	;	//	Comment
		IF	Statement list	**END_IF**	;	IF branch	
		CASE	Statement list	**END_CASE**	;	CASE branch	
		FOR	Statement list	**END_FOR**	;	FOR loop	
		WHILE	Statement list	**END_WHILE**	;	WHILE loop	
		REPEAT	Statement list	**END_REPEAT**	;	REPEAT loop	

Block call

The call of a block without return value consists of the block name and the following parameter list in parentheses. If the block has a return value, the block call following an assignment operator is present in a value assignment or an expression.
Most extended statements in the program elements catalog are calls of system blocks with return value.

Block call

Label	:	Block name (parameter list)	;	//	Comment
		Variable := block name (parameter list)	;		

Fig. 4.19 Types of SCL statements

Table 4.1 Operators with SCL

Logic operation	Designation	Operator	Priority
Parentheses	Left parenthesis, right parenthesis	(,)	1
Arithmetic	Power	**	2
	Unary plus, unary minus (sign)	+, –	3
	Multiplication, division	*, /, DIV, MOD	4
	Addition, subtraction	+, –	5
Comparison	Less than, less than-equal to, greater than, greater than-equal to	<, <=, >, >=	6
	Equal to, not equal to	=, <>	7
Binary logic operation	Negation (unary)	NOT	3
	AND logic operation	AND, &	8
	Exclusive OR	XOR	9
	OR logic operation	OR	10
Assignment	Assignment	:=	11

"Unary" means that this operator has a fixed assignment to an address

implemented in the self-written block FC 401 – adds three variable values specified at the inputs and outputs the sum at the function value. You can use this function in an SCL expression as follows:

```
#too_big:=  Adder (In1:= #Value1, In2:= #Value2,
            In3:= #Value3) > 10_000;
```

The *adder* function value is compared with the value 10 000; if it is larger, the variable *#too_big* is set to TRUE, otherwise to FALSE.

4.8 Global address areas

Fig. 4.20 shows the address areas that are available in a SIMATIC CPU. The address areas of the SIMATIC time and counter functions are not present in a CPU 1200; these functions are implemented by calling system blocks.

You can address the variables in the global address areas from anywhere in the user program, but you can address the variables in the (block-)local address areas only from the program of each block (see also section "Temporary local data" on page 200 and section "Static local data" on page 201).

Peripheral inputs

You use the peripheral inputs address area if you read values from the user data area of the input modules. Part of the address area leads to the process image. This part always starts at I/O address 0, and the length of the area is CPU-specific.

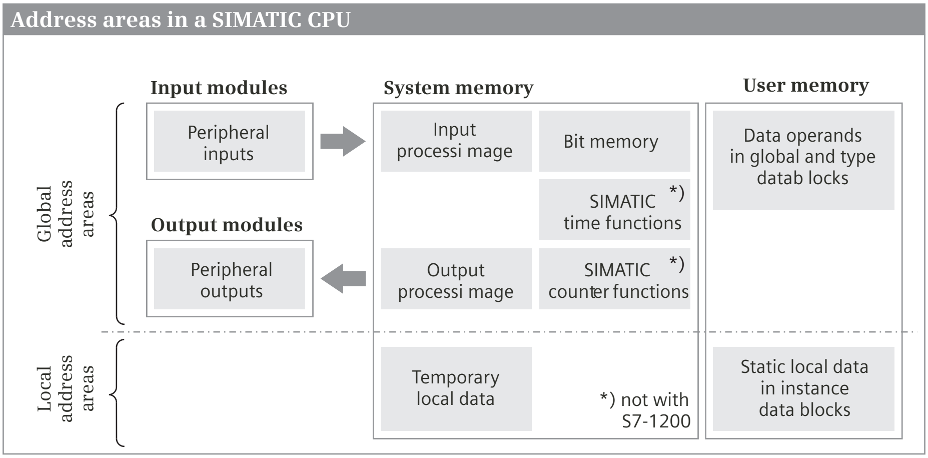


Fig. 4.20 Address areas in a SIMATIC CPU

Directly reading the I/O allows you to address modules whose interface does not lead to the process image input, such as analog input modules. The signal states of modules that lead to the process image input can also be read directly. The current signal state of the input bits on the module terminals is then queried.

In STEP 7 V5.5, the peripheral inputs address area has its own address ID: “PI”. Example: Peripheral input word 10 has the absolute address PIW10. In STEP 7 inside TIA Portal, a “:P” is appended to the input address if the peripheral inputs should be addressed. Example: The peripheral input word 10 has the absolute address %IW10:P.

Peripheral outputs

You use the peripheral outputs address area if you write values to the user data area of the output modules. Part of the address area leads to the process image. This part always starts at I/O address 0, and the length of the area is CPU-specific. Directly writing to the I/O allows you to address modules whose interface does not lead to the process image output, such as analog output modules. The signal states of modules that are controlled by the process image output can also be influenced directly. The signal state of the module bits then changes immediately as does – in parallel – the signal state of the corresponding outputs.

In STEP 7 V5.5, the peripheral outputs address area has its own address ID: “PQ”. Example: Peripheral output word 10 has the absolute address PQW10. In STEP 7 inside TIA Portal, a “:P” is appended to the output address if the peripheral outputs should be addressed. Example: Peripheral output word 10 has the absolute address %QW10:P.

Inputs

The inputs address area (short code: I) is identical to the process image of the inputs. Each time before program processing begins, the CPU transfers the signal states of the input modules to the process image and thus to the inputs. During program execution, the signal state of an input remains unchanged (data consistency for a program run).

Outputs

The outputs address area (short code: Q) is identical to the process image of the outputs. After each program execution ends, the CPU transfers the output signal states to the output modules. A change in the output signal state during program execution remains without effect for the corresponding module output; only the signal state at the end of the program cycle is transferred.

Bit memory

Bit memories (short code: M) are, so to speak, the “contactor relays” of the controller. They mainly serve to save binary signal states. They can be treated like outputs, but are not connected “to the outside”. Bit memories are used if intermediate results are to be valid beyond block limits and are to be processed in several blocks. Some of the bit memories can be set “retentive” when parameterizing the CPU, i.e. this part retains its signal state even when deenergized.

A special feature is the “clock memory byte” and also for a CPU 1200 the “system memory byte”. A memory byte whose bits are controlled by the CPU can be specified in the CPU properties for each case (Fig. 4.21).

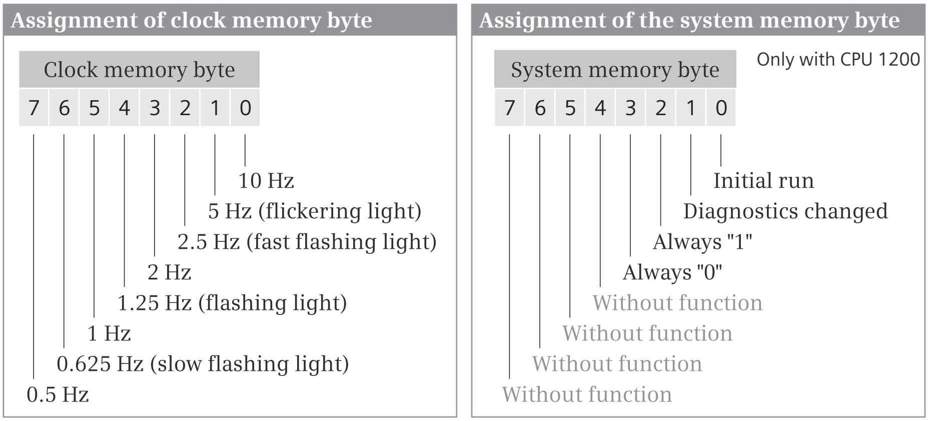


Fig. 4.21 Clock and system memory byte

SIMATIC time functions

You can use the time functions to implement timing processes in the program such as waiting and monitoring times, measurement of a time interval, or the generation of pulses. The time functions are in the system memory of the CPU, the number of time functions is CPU-specific.

The following responses of a SIMATIC time function are available:

- Pulse timer
- Extended pulse timer
- On-delay timer
- Retentive on-delay timer
- Off-delay timer

You can program a SIMATIC time function with individual elements, and also as a box in the graphical languages LAD and FBD. The box of a time function contains a related representation of all time operations in the form of function inputs and outputs and has the further advantage that you do not have to pay attention to the correct order of the individual elements to obtain a proper function. The Fig. 4.22 shows the graphic representation (in LAD) and the response patterns.

A time function starts (the time starts running) when the signal state at the start input changes. A change in signal state is always required to start a time function. In the case of an off-delay timer, the signal state must change from “1” to “0” (negative edge), in all other cases the time starts when changing from “0” to “1” (positive edge). On startup, the time function takes over the specified time period and counts down – depending on the length of the time value – in a time grid of 10 ms, 100 ms, 1 s, or 10 s. If the time period = 0, the time has expired.

A time function is reset if the reset input has signal state “1”. As long as the time function is reset, the binary time status has the signal state “0”. Resetting of the time function sets the time value and the time scale to zero.

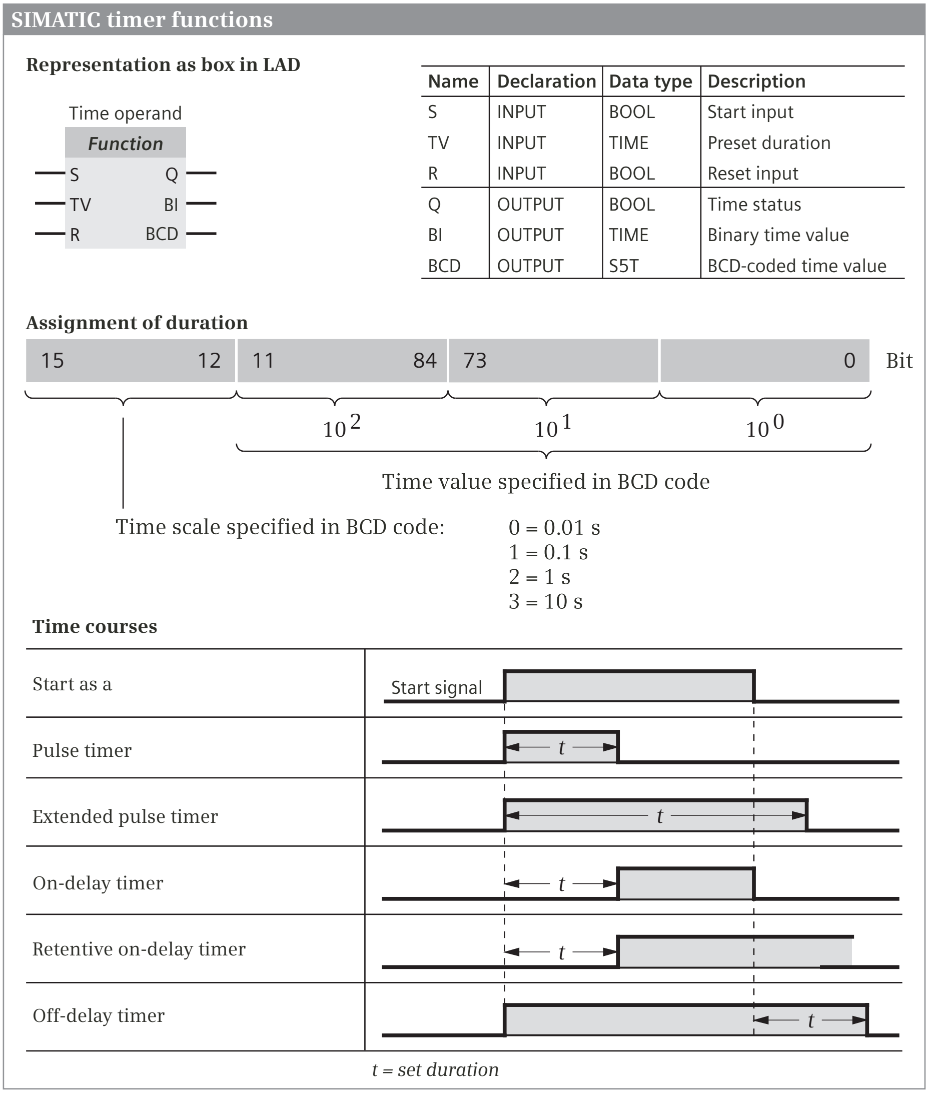
The time status with the BOOL data type shows the response of the time function. The time sequence differs depending on the time response (Fig. 4.22).

The outputs BI and BCD represent the time value in the time function in binary or BCD code. It is the current value at the time of scanning: With a time function running, the time value is counted down from the set value to zero.

SIMATIC counter functions

You can use the counter functions to execute counting tasks directly using the CPU. The counter functions can count up and down; the numerical range extends over three decades (000 to 999). The counter functions are in the system memory of the CPU; the number of counter functions is CPU-specific.

The counting frequency of the counter functions depends on the execution time of the user program. In order to count, the CPU must recognize a change in the sig-



Name	Declaration	Data type	Description
S	INPUT	BOOL	Start input
TV	INPUT	TIME	Preset duration
R	INPUT	BOOL	Reset input
Q	OUTPUT	BOOL	Time status
BI	OUTPUT	TIME	Binary time value
BCD	OUTPUT	S5T	BCD-coded time value

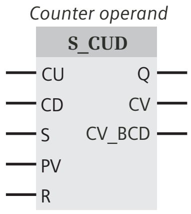
Fig. 4.22 SIMATIC time function

nal state of the input pulse, i.e. an input pulse (or a pause) must be present for at least one program cycle. The longer the program execution time, the lower the counting frequency.

You can program a SIMATIC counter function with individual elements, and also as a box in the graphical languages LAD and FBD. The box of a counter function contains a related representation of all counter operations in the form of function inputs and outputs and has the further advantage that you do not have to pay attention to the correct order of the individual elements to obtain a proper function. Fig. 4.23 shows the graphic representation (in LAD).

SIMATIC counter functions

Representation as box in LAD



Name	Declaration	Data type	Description
CU	INPUT	BOOL	Count up input
CD	INPUT	BOOL	Count down input
SI	NPUT	BOOL	Set input
PV	INPUT	WORD	Preset count value
RI	NPUT	BOOL	Reset input
Q	OUTPUT	BOOL	Counter status
CV	OUTPUT	WORD	Current count value
CV_BCD	OUTPUT	WORD	BCD-coded count value

Assignment of count value

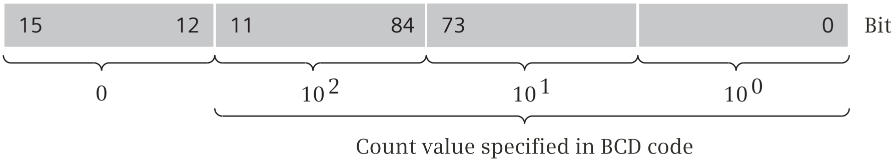


Fig. 4.23 SIMATIC counter function, using the up/down counter as an example

A counter is set if the signal state changes from “0” to “1” at the set input. To set a counter, a positive edge is always required. “Set counter” means that the counting function is set to the initial value specified by the count value. The range of values is from 0 to 999.

A counter function is reset if the reset input has signal state “1”. As long as signal state “1” is present, the counter function has the counter status “0”. Resetting the counter function sets the count value to “zero”. The reset input of the counter box does not need to be connected.

A counter function is *counted up* if the signal state changes from “0” to “1” at the count up input. For counting up, a positive edge is always required. Each positive edge when counting up increments the count value by one unit until the upper limit of 999 is reached. Any further positive edges for counting up then have no effect. Carrying forward does not take place.

A counter function is *counted down* if the signal state changes from “0” to “1” at the count down input. Each positive edge when counting down decrements the count value by one unit until the lower limit of 0 is reached. Any further positive edges for counting down then have no effect. Counting does not occur with a negative count value.

LAD and FBD: The three counter boxes S_CUD, S_CU and S_CD differ only in the type and number of counter inputs. While S_CUD has the inputs for both count directions, S_CU only has the count up input and S_CD only has the count down input.

The counter status with data type BOOL shows with signal state “1” that that the current count value is not zero, and with signal state “0” that the count value is zero.

The outputs CV and CV_BCD make the count value in the counter function available in binary or BCD code. It is the current count value at the time of scanning.

Global data addresses

The address area *Data* is organized in data blocks that are present in the user memory of the CPU (Figure 4.20 on page 144). To access a global data address, the data block in which the data address is stored must first be selected before the data address can be addressed. Depending on the addressing method, the program editor opens the data block (complete addressing) or the user must program the opening himself (partial addressing). Further details are described in chapter 4.9 "Absolute and symbolic addressing" on page 149.

4.9 Absolute and symbolic addressing

So that a variable value can be read or written to, it needs an address. Absolute addressing uses the memory space for identification (more specifically, the relative address for the beginning of the relevant address area). You can also assign a name to an absolute address and then work with this symbolic addressing.

This chapter describes the addressing of global address areas. Sections "Temporary local data" on page 200 and "Static local data" on page 201 provide information about the addressing of local data.

Absolute addressing of inputs, outputs, and memory bits

The absolute address consists of the short code of the address area, the indication of the address width, and the relative address in the address area. Example: The address “IW10” is in the process image of the inputs (I), is of word width (W), and begins with the tenth byte (10). In STEP 7 inside TIA Portal, the absolute address is indicated with a leading percent sign (%IW10).

A bit-wide address does not indicate the address width and the byte address is expanded with the bit address. Example: The address M14.2 indicates the second bit in bit memory byte 14. An address addressed absolutely can be bit-wide, byte-wide (8 bits, short code: B), word-wide (2 bytes, short code: W), or doubleword-wide (4 bytes, short code: D). How the memory space is occupied is shown in Fig. 4.24.

Absolute addressing of peripheral inputs and outputs

In STEP 7 V5.5 the address ID for the I/O inputs is PI and for the I/O outputs it is PQ. The addresses can be byte-wide (PIB, PQB), word-wide (PIW, PQW), or doubleword-wide (PID PQD).

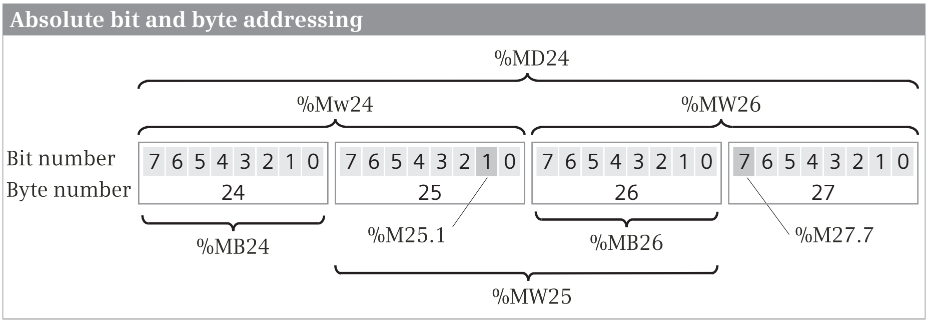


Fig. 4.24 Example of bit and byte assignments

For STEP 7 inside TIA Portal, an address from the I/O range is indicated with the suffix “:P”. Example: The address specification %IW10:P indicates the peripheral input word from byte 10. The addresses can be byte-wide (%IBy:P, %QBy:P), word-wide (%IWy:P, %QWy:P), or doubleword-wide (%IDy:P, %QDy:P); with a CPU 1200, they can also be bit-wide (%Iy.x:P, %Qy.x:P).

Absolute addressing of SIMATIC timer and counter functions

The absolute address of a SIMATIC time and counter function consists of the address ID (T, C) and a number. Example: The specification T13 is time function 13, the specification C25 is counter function 25.

In STEP 7 inside TIA Portal, the absolute address begins with a percent sign (example: %T13, %C25).

Absolute addressing of data addresses

With a CPU 1200, absolute addressing of data addresses is only possible if the *optimized block access* attribute is not selected in the data block.

The data of the user program is saved in data blocks. To address a specific address in a data block, the data block containing the data address must also be specified, because the addressing of the data addresses begins at byte 0 in each data block. The addressing can be done in two ways: complete addressing and partial addressing.

With *complete addressing*, the data block is part of the data address. First, the data block is indicated and then, separated by a point, the data address. Examples:

DB 10.DBX 2.0	Data bit 2.0 in data block DB 10
DB 11.DBB 14	Data byte 14 in data block DB 11
DB 20.DBW 20	Data word 20 in data block DB 20
DB 22.DBD 10	Data doubleword 10 in data block DB 22.

In STEP 7 inside TIA Portal, the absolute address begins with a percent sign. Examples: %DB10.DB2.0, %DB11.DBB14, %DB20.DBW20, %DB22.DBD10.

For *partial addressing*, you must first select (“open”) the desired data block. Then you can then access individual data addresses. The SCL programming language in general as well as the CPU 1200 do not understand partial addressing of data addresses.

Partial addressing is only considered for special applications since, in order to use partial addressing without errors, it is necessary to know how the program editor compiles the user program into machine code. There are statements in which the program editor opens another data block “in the background”, without it being visible in the programming interface. Complete addressing is recommended, because it ensures that the data address is always addressed in the “correct” data block.

A data block can be opened via two data block registers: the DB register and the DI register. Correspondingly, in partial addressing, there are also two address IDs: DB for a data address whose data block was opened via the DB register, and DI for a data address whose data block was opened via the DI register. A data address can be bit-wide (DBX, DIX), byte-wide (DBB, DIB), word-wide (DBW, DIW), or doubleword-wide (DBD DID).

An example of partial addressing of data addresses, programmed in a statement list with STEP 7 V5.5:

```
OPN  DB 12        //Open DB 12 via the DB register
OPN  DI 18        //Open DB 18 via the DI register
L    DBW 20       //Load data word DW 12 in DB 12
T    DIW 16       //Transfer to data word DW 16 in DB 18
```

With complete addressing, the statement sequence is:

```
L    DB 12.DBW 20 //Load data word DW 20 from DB 12
T    DB 18.DBW 16 //Transfer to data word DW 16 in DB 18
```

Symbolic addressing

The symbolic addressing uses a name or a symbol that you define in place of the absolute address. The name or symbol must be assigned to an absolute address – an address. A distinction is made between symbols for global addresses and symbols for block-specific addresses (local data).

You declare *global symbols* with STEP 7 V5.5 in the symbol table and with STEP 7 inside TIA Portal in the PLC variable table (see chapter 3.7 "Giving the addresses a name" on page 80). Global symbols can be used in the entire program; they must be unique throughout the program. In the compiled block, the program editor always shows the global symbols in quotation marks.

Block-specific symbols, which are the names for the local data, are specified in the declaration part of the corresponding block. The program editor displays local symbols preceded by a hash/pound sign (#). In STEP 7 inside TIA Portal, special characters for the block-specific symbols are also allowed, which are additionally shown in quotation marks.

In symbolic addressing of data addresses, you assign names for the data addresses within a data block. You can use the same names for different data addresses in different data blocks (the data addresses are themselves block-specific variables). In the symbol table or in the block properties, you then assign the data block a name that distinguishes it from the other data blocks. Examples of symbolic addressing are:

"Motor1".actual value	Variable *actual value* in data block "Motor1"
"Motor2".actual value	Variable *actual value* in data block "Motor2"

4.10 Indirect addressing

Indirect addressing allows you to address addresses whose address is only defined during runtime. You can also use indirect addressing to repeatedly execute program sections, e.g. in a loop, and use different addresses in each cycle.

The statements required for indirect addressing are available in the Statement List (STL) and Structured Control Language (SCL) programming languages, which use different methods.

Indirect addressing with STL

STL distinguishes between memory-indirect and register-indirect addressing:

▷ Memory-indirect addressing,
example: *IW [MD200]*, the number of the input word is present in the bit memory doubleword MD200.

▷ Register-indirect, area-internal addressing,
example: *IW [AR1, P#2.0]*, the number of the input word is present in the address register AR1; it is incremented by the offset P#2.0 when the operation is executed.

▷ Register-indirect, area-crossing addressing,
example: *W [AR1, P#0.0]*, the address area and the number of the address are present in the address register AR 1; the number is not incremented when the operation is executed.

Memory-indirect addressing uses doublewords from the address areas "data" (DBD and DID), "bit memories" (MD), and "temporary local data" (LD) as "address registers". These addresses can be addressed absolutely or symbolically. With symbolic addressing, the data types must have the required width of 16 bits or 32 bits. Register-indirect addressing uses the two address registers AR1 and AR2.

Indirect addressing with SCL

SCL allows indirect addressing of addresses and arrays:

▷ With indirect addressing of addresses, SCL considers an address area like an array whose elements can be addressed individually. Example: *MW(#index)*, the number of the bit memory word is present in the *#index* variable.

- ▷ In the case of a variable with the data type ARRAY, SCL permits a variable as index. Example: *#Array[#index]*, the number of the array element is present in the *#index* variable.

The index variables can be global or local variables addressed absolutely or symbolically. With symbolic addressing, the index variables must be of data type INT.

4.11 Elementary data types

Data types define the properties of data, essentially the representation of the contents of a variable and the permissible ranges. STEP 7 provides predefined data types that you can also compile as self-defined data types. The data types are globally available and can be used in any block.

Bit string data types

The bit string data types consist of a bit or a sequence of bits whose position is unevaluated (Table 4.2). Variables of the BOOL data type are used in conjunction with binary logic operations, memory functions, and edge evaluations. Variables with the other bit string data types are mostly handled with comparison functions, shift functions, and word logic operations.

Table 4.2 Overview of bit string data types

Data type				Present with	
Name	Width	Designation	Assignment, area	CPU 300/400	CPU 1200
BOOL	1 bit	1-bit binary value	0, 1, FALSE, TRUE	×	×
BYTE	8 bits	8-bit binary value	16#00...16#FF	×	×
WORD	16 bits	16-bit binary value	16#0000... 16#FFFF	×	×
DWORD	32 bits	32-bit binary value	16#0000 0000... 16#FFFF FFFF	×	×

Fixed-point numbers

Integers are represented with fixed-point data types. The numerical range for fixed-point numbers with sign includes positive and negative numbers; the numerical range for fixed-point numbers without sign includes only positive numbers (Table 4.3). The individual bits are evaluated, and the place value corresponds to a power of two: Bit 0 has the value 2^0, bit 1 has the values 2^1, etc. You get the value of a (positive) fixed-point number when you add the place value of all bits set to “1”.

Table 4.3 Overview of fixed-point numbers

Data type				Present with	
Name	Width	Designation	Assignment, area	CPU 300/400	CPU 1200
SINT	8 bits	8-bit fixed-point number with sign	−128...+127	–	×
INT	16 bits	16-bit fixed-point number with sign	−32 768...+32 767	×	×
DINT	32 bits	32-bit fixed-point number with sign	−2 147 483 648... +2 147 483 647	×	×
USINT	8 bits	8-bit fixed-point number without sign	0...255	–	×
UINT	16 bits	16-bit fixed-point number without sign	0...65 535	–	×
UDINT	32 bits	32-bit fixed-point number without sign	0...4 294 967 296	–	×

Unsigned fixed-point numbers also do not have a bit for the sign. For the fixed-point numbers with a sign, this is the leftmost bit – depending on data type SINT, INT, or DINT this is bit 7, 15, or 31. If the sign bit has the value “1”, then the number is negative and the significance of the bits is represented as a two's complement. You obtain the value of a negative fixed-point number if the significance of all bits with the value “1” is added and the sum is added to the negative place value of the most significant bit (the sign).

Fixed-point numbers are used in conjunction with comparison functions, arithmetic functions, and conversion functions.

Floating-point numbers

Fractional numbers (numbers with decimal places) are shown with floating-point data types. A variable with data type REAL or LREAL consists internally of three components: a sign (1 bit), an exponent (8 or 11 bits), and a mantissa (23 or 52 bits). STEP 7 performs the conversion to the internal components. The representation in programming is either a decimal fraction (e.g. 123.45) or the exponential representation to base 10 (e.g. 12 34e12 corresponding to 12.34×10^{12}).

The LREAL data type available with the CP 1200 can only be used if the *Optimized Block Access* attribute is activated (Table 4.4).

Table 4.4 Overview of floating-point numbers

Data type				Present with	
Name	Width	Designation	Assignment, area	CPU 300/400	CPU 1200
REAL	32 bits	32-bit floating-point number	$\pm1.18\times10^{-38}$ ... $\pm3.40\times10^{38}$	×	×
LREAL	64 bits	64-bit floating-point number	$\pm2.23\times10^{-308}$ ... $\pm1.80\times10^{308}$	–	×

Fixed-point numbers are used in conjunction with comparison functions, arithmetic functions, mathematical functions, and conversion functions.

Points in time and durations

A variable with data type DATE is saved in a word as an unsigned fixed-point number. The content of the variable corresponds to the number of days since 01.01.1990. The date is given with year, month, and day. STEP 7 performs the conversion to the internal representation (Table 4.5).

Table 4.5 Overview of points in time and durations

Data type				Present with	
Name	Width	Designation	Assignment, area	CPU 300/400	CPU 1200
DATE	16 bits	Date	D#1990-01-01 ... D#2168-12-31	×	×
S5TIME	16 bits	Duration in SIMATIC format	S5T#0ms ... S5T#2h46m30s	×	–
TIME	32 bits	Time value in IEC format	T#–24d20h31m23s647ms ... T#24d20h31m23s647ms	×	×
TIME_OF_DAY	32 bits	Time of day	TOD#00:00:00 ... TOD#23:59:59.999	×	×

A variable with data type TIME (duration) occupies a doubleword. The content of the variable corresponds to the number of milliseconds, and is saved as a 23-bit fixed-point number with sign. The time period is specified in days, hours, minutes, seconds, and milliseconds, where unused units can be omitted. STEP 7 performs the conversion to the internal representation.

A variable with data type TIME_OF_DAY occupies a doubleword. It contains the number of milliseconds since the beginning of the day (0:00) as an unsigned fixed-point number. The time-of-day is specified in hours, minutes, seconds, and milliseconds, where the milliseconds specification can be omitted. STEP 7 performs the conversion to the internal representation.

For processing variables with the data types DATE, TIME, and TIME_OF_DAY, there are system and standard functions such as the addition of a time period to the time of day or the extraction of the date from the complex data type DATE_AND_TIME. STEP 7 V5.5 supplies this function in the *Standard Library* in the *IEC Function Blocks* program. STEP 7 inside TIA Portal provides these functions in the Program Elements catalog under *Advanced statements* > *Date and time*.

A variable with data type S5TIME is required to supply the SIMATIC time functions with a duration. The data type assigns a decade to a word for the time grid and three decades for the time value (see Figure 4.22 on page 147). The time period is specified in hours, minutes, seconds, and milliseconds, where unused units can be omitted. STEP 7 performs the conversion to the internal representation.

Further elementary data types

A variable with data type CHAR (character) occupies one byte. The data type CHAR represents a single character which is saved in ASCII format. Any printable characters can be given in single quotation marks (Table 4.6).

Table 4.6 Overview of further elementary data types

Data type				Present with	
Name	Width	Designation	Assignment, area	CPU 300/400	CPU 1200
CHAR	8 bits	One character in ASCII code	'a', 'A', '1', ...	×	×
BCD16 [1]	16 bits	3 decades with sign	−999 ... +999	×	×
BCD32 [1]	32 bits	7 decades with sign	−9 999 999 ... +9 999 999	×	×

[1] Not a data type in a narrower sense; only relevant to data type conversion

A variable in binary coded decimal (BCD) representation is represented with a sign decade and three or seven numerical decades. The sign decade contains zeros (corresponding to a positive sign) or ones (corresponding to a negative sign). Each numerical decade is stored in hexadecimal code in the range from 0 to 9. The BCD representation is not really a data type. The representation is used when converting to and from fixed-point numbers. For example, the count value of a SIMATIC counter function is represented as a BCD16 number, which can then be converted into a fixed-point number.

4.12 Complex data types

The complex data types cannot be edited with the “simple” statements. For the processing of variables with these data types, there are system and standard functions such as extracting the date (DATE) from the data type DTL (DATE_AND_TIME) or combining two STRING variables. STEP 7 V5.5 supplies this function in the *Standard Library* in the *IEC Function Blocks* program. STEP 7 inside TIA Portal provides these functions in the Program Elements catalog under *Advanced statements > Date and time* or *Advanced statements > String + Char*.

Table 4.7 gives an overview of the complex data types.

DT (DATE_AND_TIME)

The data type DT (date and time) represents a point in time in the range of DT#1990-01-01-00:00:00.000 to DT#2168-12-31-23:59:59.999. The data type consists of 16 BCD-coded numbers and is 8 bytes long. The last byte indicates the day of the week (1 = Sunday through 7 = Saturday). The general representation is: DT#year-month-day-hours:minutes:seconds.milliseconds.

Table 4.7 Overview of complex data types

Data type			Present with	
Name	Width	Designation	CPU 300/400	CPU 1200
DT	8 bytes	Date and time	×	–
DTL	12 bytes	Current date and time (long format)	–	×
STRING	2+n bytes	ASCII-coded character string	×	×
ARRAY	n bytes	Field with components of the same type	×	×
STRUCT	n bytes	Structure with components of different types	×	×

DTL (DATE_AND_TIME)

The data type DT (date and time, long format) represents a point in time in the range from DTL#1990-01-01-00:00:00:00 to DT#2168-12-31-23:59:59:999.999.999. The data type consists of 24 BCD-coded numbers and is 12 bytes long. The general representation is:
DTL#year-month-day-hours:minutes:seconds:nanoseconds.

STRING

The data type STRING represents a character string consisting of up to 254 characters. In the declaration, enter the maximum number of allowed characters after the keyword STRING in square brackets. This information may be omitted, in which case the editor sets the maximum number of 254 bytes. The (actual) length of the string is defined between STRING[0] and STRING[254].

A variable of the STRING data type reserves two bytes more than the declared maximum length in memory. The default value is encoded with ASCII characters in single quotes. If the default value is shorter than the declared maximum length, the remaining character positions are not occupied. When a variable with the data type STRING is further processed, only the actually occupied character positions are considered.

ARRAY

The data type ARRAY is an array with a fixed number of components of the same data type. After the data type ARRAY, enter the range of array indices in square brackets. The initial value on the left must be less than or equal to the end value on the right. Both indices are INT numbers in the range from -32 768 to +32 767. The index can also be a variable in SCL. An array can have up to 6 dimensions, whose limits are each separated by a comma. Example of a two-dimensional array with 32 × 16 REAL elements:

```
ARRAY [1..32, 1..16] OF REAL
```

Any data type except ARRAY is allowed for the individual array components. A complete array variable is defined in a global data block or block-specific variable.

It can be also be applied to a block parameter of the same type. The array components are individually treated as variables of the same data type.

STRUCT

The STRUCT data type represents a data structure with a fixed number of components, which can each have a different data type. All data types can be used, as well as other structures. Nesting of up to 6 structures is allowed.

A complete structure variable is defined in a global data block or as block-specific variable. It can be also applied to a block parameter of the same type if the structure, the names, and data type of all components match. The structure components are individually treated as variables of the same data type.

Example of declaration in a program source file:

```
MotorData1      : STRUCT
  Switch on     : BOOL       := FALSE;
  Switch off    : BOOL       := TRUE;
  SetpointSpeed: INT         := 5000;
  ActualSpeed   : INT;
  Description   : STRING[10] := '=MD01-M003';
END_STRUCT;
```

The variable *MotorData1* consists of two BOOL components, two INT components, and one STRING component. An individual component is addressed with *Variablename.Componentname*, e.g. *MotorData1.SwitchOn*.

4.13 Data types for block parameters

The data types for block parameters (“parameter types”) are used to pass on the SIMATIC time and counter functions, blocks, and pointers to global addresses to the called block. Table 4.8 gives an overview of the parameter types.

TIMER

The TIMER data type allows a SIMATIC time function to be passed on to the called block. The TIMER data type is also used in the symbol table (with STEP 7 V5.5) or in the PLC variable table (with STEP 7 inside TIA Portal) to assign a name to a SIMATIC time function.

COUNTER

The COUNTER data type allows a SIMATIC counter function to be passed on to the called block. The COUNTER data type is also used in the symbol table (with STEP 7 V5.5) or in the PLC variable table (with STEP 7 inside TIA Portal) to assign a name to a SIMATIC counter function.

Table 4.8 Overview of parameter types

Data type		Present with	
Name	Designation	CPU 300/400	CPU 1200
TIMER COUNTER	SIMATIC time function SIMATIC counter function	×	–
BLOCK_FC BLOCK_FB BLOCK_DB	Function FC Function block FB Data block DB	×	–
POINTER	Pointer to a variable with elementary data type	×	–
ANY	Pointer to a variable or a data area	×	–
VARIANT	Pointer to any variable	–	×
VOID	Deactivating the function value of a function FC	×	×

BLOCK_FB, BLOCK_FC

For block parameters declared with the data types BLOCK_FC or BLOCK_FB, you can create a function block (FB) or a function (FC) as actual parameter. These blocks may not have any block parameters themselves.

BLOCK_DB

For a block parameter with data type BLOCK_DB you can create a data block as an actual parameter. In the called block, you can then call this data block, and thus access the data addresses of this data block with absolute addressing.

POINTER

For a block parameter with data type POINTER, you can create an address such as %M200.0 or its symbol, or a pointer in the form

P#[Datablock.]Address Byteaddress[.Bitaddress]

. Only addresses or variables with elementary data types are permitted.

ANY

For a block parameter with data type ANY, you can create an address such as %M200.0 or its symbol, or a pointer in the form

P#[Datablock.]Address Byteaddress[.Bitaddress] Datatype Number.

All addresses and variables are allowed, including, for example, array and structure variables. With the data type ANY, a variable in the temporary local data can also be declared and then accommodate an ANY pointer. This pointer can be manipulated at runtime, for example, as a variable address of a data source during copying.

VARIANT

A block parameter with data type VARIANT contains a pointer to a variable or a data area. Variables of all data types are allowed in a block parameter with data type VARIANT. The variables which can be connected to the block parameter or which are meaningful are defined by the programming within the called block. For addressing a data range, the ANY pointer can be used in the form

P#[Datablock.]Address Byteaddress[.Bitaddress] Datatype Number

VOID

The data type VOID (= without type) is used in the function value of a function FC if the function value should not be displayed (if the function FC should not have a function value).

4.14 Further data types

User-defined data type UDT

A user-defined data type (UDT) is a compilation of data types that a user programs (see also chapter 3.10 "Programming a user-defined data type" on page 89). A user data type is structured like the complex data type STRUCT. A user data type can be the data type of data addresses or local data or be used as a template for a type data block.

System-defined data type SDT

A system-defined data type (SDT) is a predefined, non-modifiable data type that is structured like the data type STRUCT. A system-defined data type is used together with certain functions or statements. Example: On a CPU 1200, the time function has the data type IEC_TIMER (SDT 31).

Hardware data types

With the hardware data types, STEP 7 inside TIA Portal addresses hardware and software objects. The data type and the contents are specified, and the name can be changed in the *Constants* tab of the PLC variable table.

Table 4.9 Overview of further data types

Data type	Present with	
	CPU 300/400	CPU 1200
User-defined data type (UDT)	×	×
System-defined data type (SDT)	–	×
Hardware data type	–	×

5 The user program

On delivery, the SIMATIC controller with the input/output modules is not yet able to control the machine or the plant. The CPU requires a program which it processes step by step, executing the instructions stored in the program, and thereby solving the control task. This program is called the “user program” since it is created by you. The programming languages of the STEP 7 programming software are used to write the program.

The user program is executed in various ways. After power-on, the CPU processes a startup program followed by cyclic (i.e., repeated continuously) execution of the main program which can be interrupted by alarm or error events with assigned programs. Priority classes control the mutual interruptibility.

The user program is usually divided into individual sections each representing a self-contained technological or functional unit. These program sections are called *blocks*. Before it can be processed, a block must be called. You can then call other blocks as so-called subprograms within one block, thereby structuring the user program. When skillfully organized, the call sequence in the main program (in the organization block OB 1) represents the technological or functional structure of the machine or plant to be controlled.

5.1 Program execution with SIMATIC

Program execution modes of a SIMATIC user program

The complete program of a CPU comprises the operating system and the user program (control program).

The operating system is the totality of all instructions and declarations of internal operating functions (e.g. saving of data in event of power failure, activation of priority classes, etc.). The operating system is a fixed part of the CPU which you cannot modify. However, you can reload the operating system, e.g. for a program update.

The user program is the totality of all instructions and declarations programmed by you for signal processing by means of which the plant (the process) to be controlled is influenced in accordance with the control task.

The organization blocks are the interfaces between operating system and user program. The *organization blocks* are part of the user program and are called and processed by the operating system when certain events occur. The organization blocks are divided into *priority classes* which determine the sequence of program processing (mutual interruptibility) when several events occur.

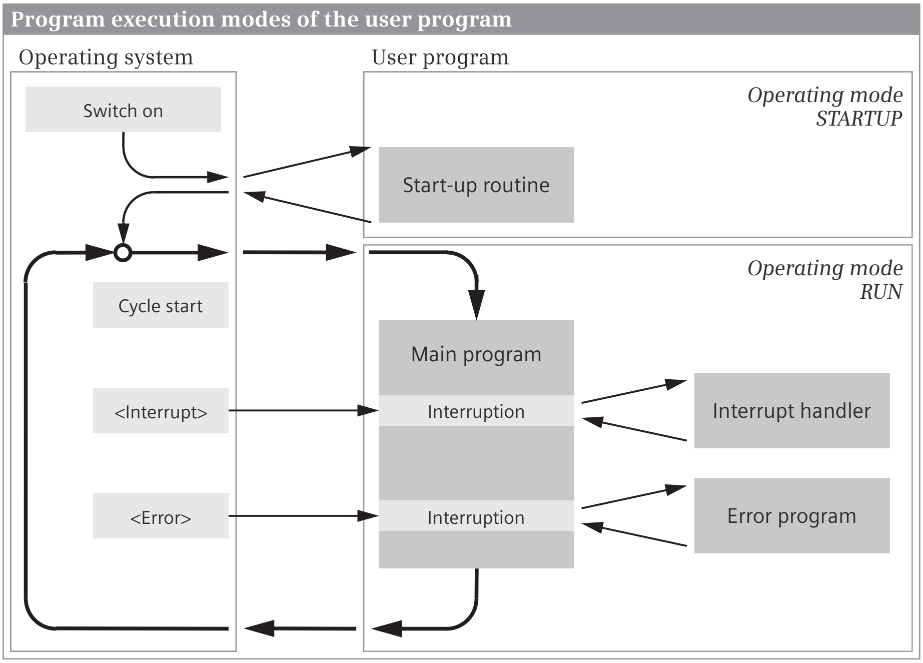


Fig. 5.1 Program execution modes of a SIMATIC user program

An example: An I/O module detects a hardware interrupt and sends the result to the CPU. The CPU then interrupts the processing of the main program and calls the organization block belonging to the event. This contains the interrupt handler assigned to the hardware interrupt. In each organization block, the user program can be structured through additional blocks.

Fig. 5.1 shows a rough breakdown of the program types of a SIMATIC CPU: The start-up routine, the main program, and the interrupt handler and error programs. Program execution commences in the CPU with the start-up routine. The start-up routine is optional. Following execution of the start-up routine, the CPU commences with execution of the main program. If the CPU has processed the main program, it starts processing again from the beginning. This cyclic processing is typical for programmable logic controllers.

The main program has the lowest processing priority. Any events that occur can interrupt the main program following each instruction; the CPU then executes the associated interrupt handler or error program and subsequently returns to execution of the main program.

Overview of the organization blocks

Table 5.1 shows an overview of the possible organization block of a CPU 300/400 (maximum configuration). Which of the organization blocks are actually available depends on the CPU type. The organization block of a CPU 1200 are shown in Tabelle 5.5 auf Seite 168.

Table 5.1 Organization blocks with a CPU 300/400 (maximum configuration)

Organization block		Call
Main program	OB 1	Cyclically (repeated continuously) by the operating system
Time-of-day interrupt	OB 10 to OB 17	At a certain time or at periodic intervals (e.g. monthly)
Time-delay interrupt	OB 20 to OB 23	After a settable time, controlled by the user program
Cyclic interrupt	OB 30 to OB 38	Periodically at settable time intervals (e.g. every 100 ms)
Hardware interrupt	OB 40 to OB 47	In the event of an interrupt signal from an I/O module
DPV1 interrupts	OB 55 to OB 57	If the event of status, update and manufacturer interrupts from PROFIBUS DPV1 slaves
Multiprocessor interrupt	OB 60	Event-controlled by the user program in multiprocessor operation
Isochronous mode interrupt	OB 61 to OB 64	Synchronous with data transmission in a PROFIBUS DP master system or in a PROFINET IO system
Redundancy error interrupts	OB 70, OB 72, OB 73	In the event of redundancy loss due to I/O, or CPU and communication redundancy errors (H system)
Asynchronous error	OB 80 to OB 88	For errors not related to program execution (e.g. station failure)
Background processing	OB 90	If the minimum cycle has not yet been completed
Startup	OB 100, OB 101, OB 102	During startup of the programmable controller
Synchronous error	OB 121, OB 122	For errors related to program processing (e.g. I/O access errors)

5.2 The start-up routine

The CPU performs a startup and is in STARTUP operating mode

▷ After the power is turned on

▷ After the mode selector switch is turned from STOP to RUN

▷ following a request by a communication function (triggered by programming device or by communication function blocks of another SIMATIC station).

A startup can be triggered *manually* with the mode switch or a communication function, or *automatically* by turning on the power supply.

There are no restrictions on the length of the start-up routine and no time restrictions on the execution of the start-up routine. No interrupts are processed while the start-up routine is being executed. Errors are handled as in RUN mode. During startup, the CPU updates the timer functions, the runtime meters, and the real-time clock. The output modules are disabled during startup, i.e. no output signals can be output during startup.

What startup types there are and what the main differences are can be seen in Table 5.2. A CPU 300 performs – just like a CPU 1200 – a warm restart on startup. For a CPU 1200, additional organization blocks with a number larger than or equal to 123 can contain a start-up routine. A CPU 400 can perform cold restarts, warm restarts, and hot restarts. Fig. 5.2 shows the activities of a CPU 300 in the STARTUP and RUN modes.

Table 5.2 Startup methods and assigned organization blocks

Startup method		CPU 1200	CPU 300	CPU 400
Cold restart	All data is deleted	–	–	OB 102
Warm restart	All non-retentive data is deleted	OB 100, ≥ OB 123	OB 100	OB 100
Hot restart	All data is retained	–	–	OB 101

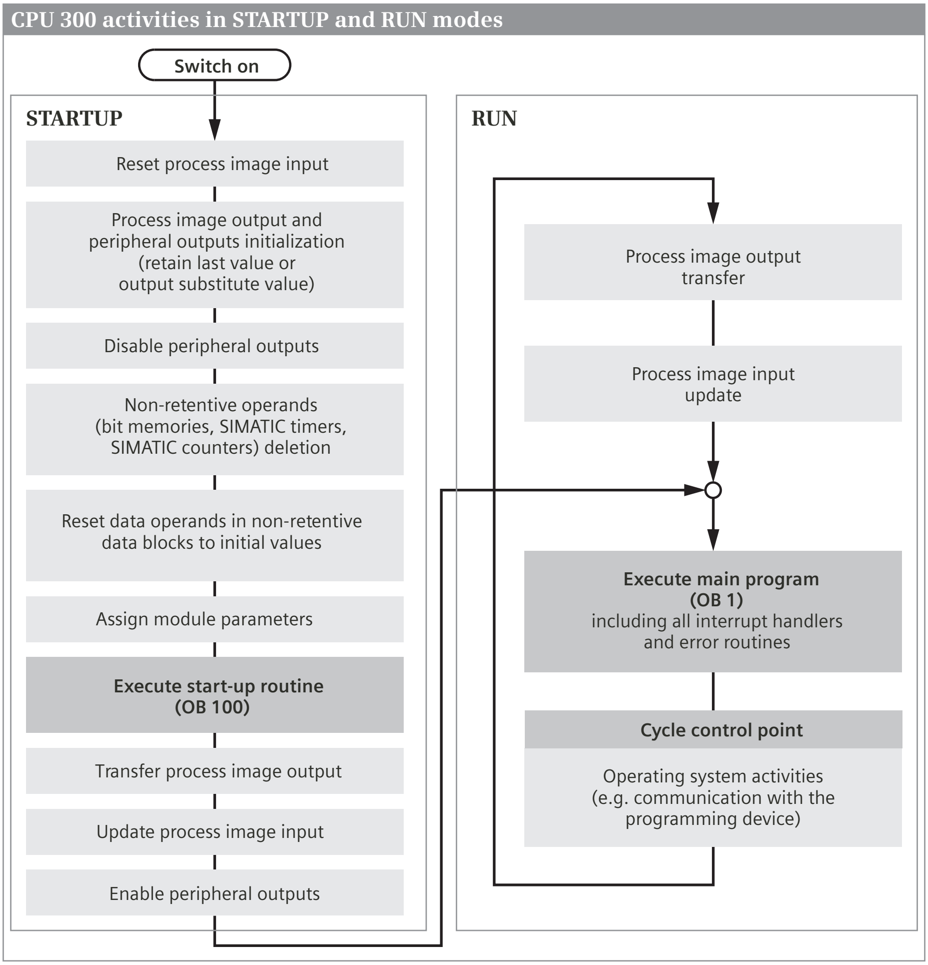

CPU 300 activities in STARTUP and RUN modes
Switch on
STARTUP
Reset process image input
Process image output and peripheral outputs initialization (retain last value or output substitute value)
Disable peripheral outputs
Non-retentive operands (bit memories, SIMATIC timers, SIMATIC counters) deletion
Reset data operands in non-retentive data blocks to initial values
Assign module parameters
Execute start-up routine (OB 100)
Transfer process image output
Update process image input
Enable peripheral outputs
RUN
Process image output transfer
Process image input update
Execute main program (OB 1) including all interrupt handlers and error routines
Cycle control point
Operating system activities (e.g. communication with the programming device)

Fig. 5.2 CPU 300 activities in the STARTUP and RUN modes

Cold restart

During a cold restart, the CPU puts itself and the modules into the configured basic state, deletes all data from the system memory (also the retentive data), and calls the organization block OB 102. The current program and the current data in the work memory are deleted and the program from the load memory is reloaded (in contrast to a memory reset, an RAM load memory is not deleted). After a cold restart, the CPU processes the main program in OB 1 from the beginning.

Warm restart

During a warm restart, the CPU puts itself and the modules into the configured basic state, deletes any non-retentive data from the system memory, and calls OB 100. The current program and the current data in the work memory are retained. After a warm restart, the CPU processes the main program in OB 1 from the beginning.

With a CPU 1200, alongside the OB 100 organization block you can program additional startup organization blocks that have a number greater than or equal to 123 and the *Startup* event class. The additional organization blocks are then processed after OB 100 in the order of their numbers.

Hot restart

In the event of a STOP or power failure, the CPU stores all interruption events and the internal CPU registers which apply to processing the user program. During a hot restart, it can then continue processing at the point in the program at which it was interrupted. This can be the main program or also an interrupt handler or error program. All "old" interruption events have been stored and are processed.

On hot restart, the OB 101 organization block is processed and the so-called "remaining cycle" is then processed in organization block OB 1. The "remaining cycle" from the point in the program at which the CPU continues after a hot restart up to the end of the main program is regarded as startup. No new interrupts are processed. The output modules are disabled and are in the basic state.

By setting the CPU parameters, you can specify the interruption duration after which the CPU may still perform a hot restart (100 ms to 1 hour). If the interruption takes longer, only a cold restart or a warm restart is permitted.

5.3 The main program

The main program is the cyclically processed user program. Cyclic program processing is the "normal" program processing for programmable logic controllers. The majority of controllers use only this type of program processing. When event-controlled program processing is used, this is usually only a supplement to the main program.

Table 5.3 Organization blocks for the main and background program

Program type	CPU 1200	CPU 300	CPU 400
Main program	OB 1, ≥ OB 123	OB 1	OB 1
Background program	–	–	OB 90

The CPU only processes the main program when it is in RUN mode. The main program is present in organization block OB 1, which has the lowest processing priority. It can be interrupted by all alarm and error events. With a CPU 1200, alongside the OB 1 organization block you can program additional main program organization blocks that have a number greater than or equal to 123 and the *Program cycle* event class. The additional organization blocks are then processed after OB 1 the order of their numbers (Table 5.3).

Background processing

You can set the processing time for the main program so that there is still time for background processing. The CPU calls organization block OB 90, which depending on the time available is processed “piece by piece” in alternation with a complete cycle of the main program. Just like OB 1, OB 90 can also be interrupted by all alarm events and error events.

Program organization

The user program is usually divided into individual program sections (i.e., blocks). When a block is to be processed, it must be called. Only the organization blocks are not called by the user program. They are started by the operating system of the CPU instead.

The structure of the program specifies the event for which the CPU is to process the blocks and the sequence in which these blocks are to be processed. For example, a rough program structure is created when you program the calls of the blocks for the main program in organization block OB 1. In these “higher-level” blocks, you can then call other blocks and structure the user program in more detail, and so on.

In principle, there are two ways to divide the overall automation task into smaller subtasks:

A *technological program structure* is primarily based on the setup of the plant to be controlled and is divided into plant, plant section, and component. The individual parts of the plant or the process to be controlled correspond to the individual program sections. Example: The plant unit “conveyor belt” may consist of different conveying elements, shunting vehicles, and lifting station; these elements also consist of individual components such as motors, valves, and indicator elements.

A *functional program structure* is based on the control function to be performed. The lower-level blocks contain the program of the subfunctions. Example: The “message acquisition” function may consist of message conditioning, message storage and message output.

Nesting depth of the block calls

During run time, the CPU makes an entry in the block stack (B stack) for every block which is called. With this information, the CPU is able to continue processing in the calling block, after processing of the called block has been concluded. The next block which is called overwrites the data of the previous call in the B stack. If a “nested” block call is made within the called block, the CPU sets up a new element in the B stack. The number of B stack elements is limited to a CPU-specific maximum value. If too many “nested” block calls are made, the CPU assumes the STOP state with the error message “block stack overflow”.

Start information for a CPU 300 and CPU 400

When an organization block is called, the CPU's operating system transfers start information in the temporary local data. The start information can only be directly scanned in the program of the organization block. The system block RD_SINFO also permits access to the start information from the blocks called in the organization block.

The program editor automatically configures the start information when adding an organization block to the user program. Names and comments in English can be adapted according to your requirements.

This start information is 20 bytes long for every organizational block and practically identical. The “standard structure” of the start information shown in Table 5.4 can be found as a basic framework in all organization blocks.

Table 5.4 Structure of the start information

Byte	Data type	Structure element	Meaning, remark
0	BYTE	EV_CLASS	Bits 0 to 3: Event detection Bits 4 to 7: Event class
1	BYTE	EV_NUM	Event number
2	BYTE	PRIORITY	Priority class, number of execution level
3	BYTE	NUM	OB number
4	BYTE	TYP2_3	Data ID 2_3: identifies the information entered in ZI2_3
5	BYTE	TYPE_1	Data ID 1: identifies the information entered in ZI1
6 ... 7	WORD	ZI1	Additional information 1
8 ... 11	DWORD	ZI2_3	Additional information 2_3
12 ... 19	DATE_AND_TIME	Event time	Beginning of event

Start information for a CPU 1200

A CPU 1200 makes the start information available to an organization block in its block interface in the *Input* declaration section (input parameters). The structure and content of the start information depends on the organization block. Not all organization blocks have start information (Table 5.5). The organization blocks with a variable number are assigned by the event class of a program type.

Table 5.5 Organization blocks and start information for a CPU 1200

	OB number			
Program type	**Fixed**	**Variable**	**Event class**	**Start information**
Main program	OB 1	≥ 123	Program cycle	–
Start-up routine	OB 100	≥ 123	Startup	Retentivity and time-of-day error
Time-delay interrupt	–	20 ... 23, ≥ 123	Time delay interrupt	–
Cyclic interrupt	–	30 ... 38, ≥ 123	Cyclic interrupt	–
Hardware interrupt	–	40 ... 47, ≥ 123	Hardware interrupt	–
Time error	OB 80	–	Time error interrupt	Error ID, number and priority of OB signaling the error
Diagnostic interrupt	OB 82	–	Diagnostic error interrupt	Diagnostic status and identification of the module reporting the error, channel number

5.4 The process images

Module addresses in the process image

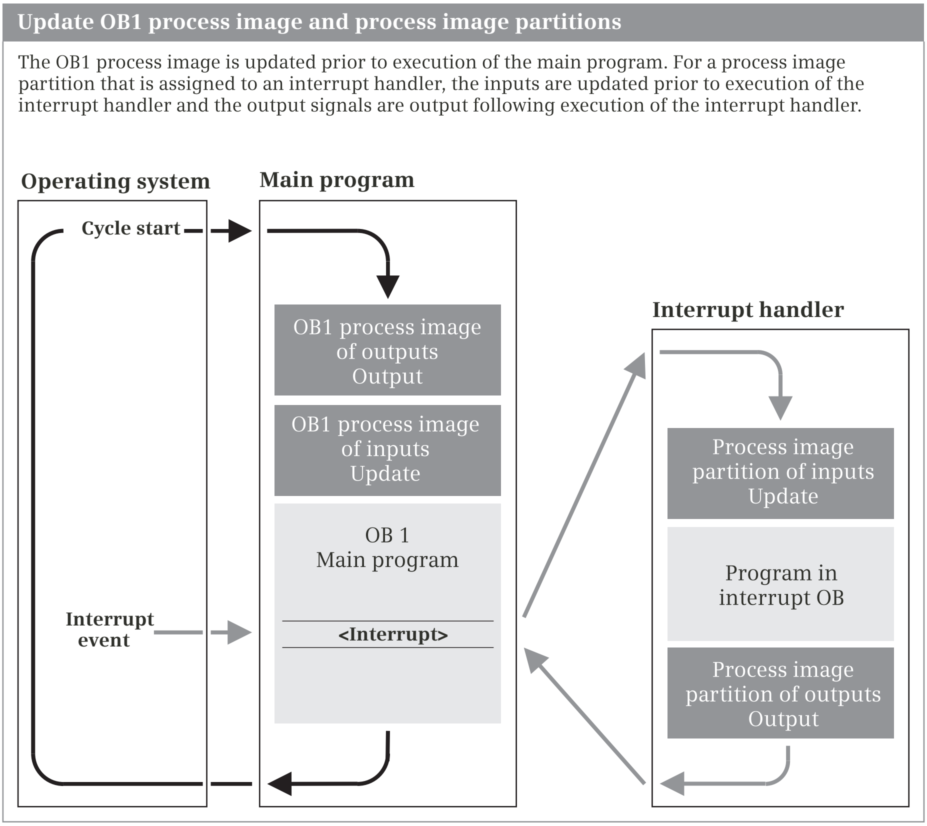
The process images contain the signal states of some of the input and output modules and correspondingly it is organized into a process image input and process image output. You address the process image input via the address area I inputs and the process image output via the address area Q outputs. In general, the machine or process is controlled via the inputs and the outputs (see also Chapter 1.4 "The path of a binary signal from the sensor to the program" on page 15).

The process image is part of internal CPU system memory. It begins at I/O address 0 and ends at an upper limit specified by the particular CPU. This limit can be set with appropriately designed CPUs.

When you define the parameters of the module addresses, you also specify whether the signal states of a module are to be included in the process image. You can use the addresses of the process image which are not assigned to modules similar to the bit memory area. Modules whose addresses are not included in the process image are addressed with the address areas peripheral inputs (PI) and peripheral outputs (PQ).

Updating the process image

Following a CPU startup and prior to initial processing of the organization block OB 1, the operating system transfers the signal states of the process image output to the output modules and accepts the signal states of the input modules into the process image input. This is followed by execution of the main program in OB 1, where usually the signal states of the inputs are linked to each other and the outputs are controlled. Following termination of OB 1, a new cycle begins with updating of the process image (Fig. 5.3).



Update OB1 process image and process image partitions

The OB1 process image is updated prior to execution of the main program. For a process image partition that is assigned to an interrupt handler, the inputs are updated prior to execution of the interrupt handler and the output signals are output following execution of the interrupt handler.

Operating system
Main program
Interrupt handler
Cycle start
Interrupt event
OB1 process image of outputs Output
OB1 process image of inputs Update
OB 1 Main program
<Interrupt>
Process image partition of inputs Update
Program in interrupt OB
Process image partition of outputs Output

Fig. 5.3 Process image update for main program and interrupt routines

If an error occurs during automatic updating of the process image, e.g. because a module can no longer be addressed, the organization block OB 85 “Program execution error” is called. If OB 85 is not present, the CPU switches to STOP mode.

Part process images

Some CPUs allow you to divide the process image into part process images. You do this when you define the parameters of the signal modules with the hardware configuration tool HW Config. When you allocate the addresses with HW Config, you specify the part process image with which the module is to be addressed. You can divide both process images – inputs and outputs.

All modules with an address in the process image area which you do not assign to a process image partition are located in the “OB1 process image”. This corresponds to the “total process image” of those CPUs which cannot handle a division into process image partitions. This OB1 process image is automatically updated by the operating system of the CPU during cyclic processing. You can also disable this automatic update for the CPU 400.

Allocation to interrupt organization blocks

Some CPUs allow you to assign part process images to the organization blocks for interrupt and asynchronous errors. You do this when you define the parameters for the CPU. We recommend placing in these part process images the addresses of those modules that you address in the interrupt program. When an alarm event occurs, the assigned part process images are updated.

System functions for part process images

The system functions UPDAT_PI and UPDAT_PO or SYNC_PI and SYNC_PO for isochronous mode interrupts update the parameterized process image partition of the inputs or outputs when they are called. The system functions can be called at any point in the program to update a process image partition. If you specify process image partition 0, you can also update the process image of the main program, e.g. if you disabled automatic updating. If an error occurs during the update, the error is reported with the return value of the system function.

5.5 Cycle Time, Reaction Time

Cycle monitoring time

Program processing in the organization block OB 1 is time-monitored. This is handled by the so-called “cycle time monitoring” function. The default monitoring time is set to 150 ms. You can change this value within the range from 1 ms to 6 s in the CPU properties. If processing of the main program takes longer than the set cycle monitoring time, the CPU calls the organization block OB 80 *Time error*. If this is not present, the CPU switches to the stop status.

The cycle monitoring time covers the total processing time of OB 1. This also includes the processing times for higher priority classes which interrupt the main program (in the current cycle). Communication processes by the operating system (e.g. programming device accesses to the CPU) also add to the runtime of the main program.

Minimum cycle time, background processing – OB 90

If the CPUs are designed accordingly, you can specify a minimum cycle time. If main program processing, including interruptions, does not take as long as specified, the CPU waits until the set minimum cycle time is reached. Only then the CPU starts the next cycle by calling OB 1 again. The minimum cycle time is disabled by default. You can set a value between 1 ms and 6 s in the CPU properties.

The CPU processes organization block OB 90 *background processing* in the time between the actual end of the cycle and the expiration of the minimum cycle time. OB 90 is processed “piece by piece”: Processing of OB 90 is interrupted when OB 1 is called by the operating system, and continued again at the point of interruption when OB 1 processing is concluded. An interruption by OB 1 can occur after every instruction (Fig. 5.4).

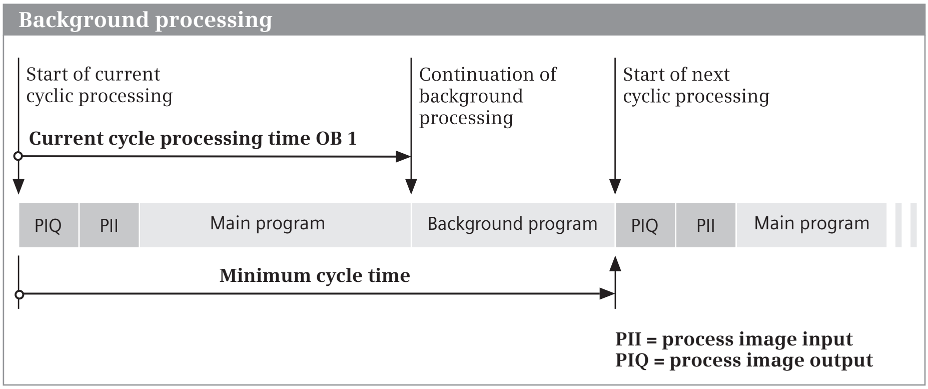

Background processing
Start of current cyclic processing
Continuation of background processing
Start of next cyclic processing
Current cycle processing time OB 1
PIQ
PII
Main program
Background program
PIQ
PII
Main program
Minimum cycle time
PII = process image input
PIQ = process image output

Fig. 5.4 Minimum cycle time and background processing

The closer the processing time of OB 1 is to the minimum cycle time, the less time remains for processing OB 90. The program processing time is not monitored in OB 90. Processing of OB 90 only takes place in RUN mode. Just like OB 1, it can also be interrupted by alarm and error events.

Reaction time

If the user program in OB 1 works with the signal states of the process images, this results in a response time which is dependent on the program execution time (the cycle time). The response time lies between one and two cycle times, as shown in Fig. 5.5.

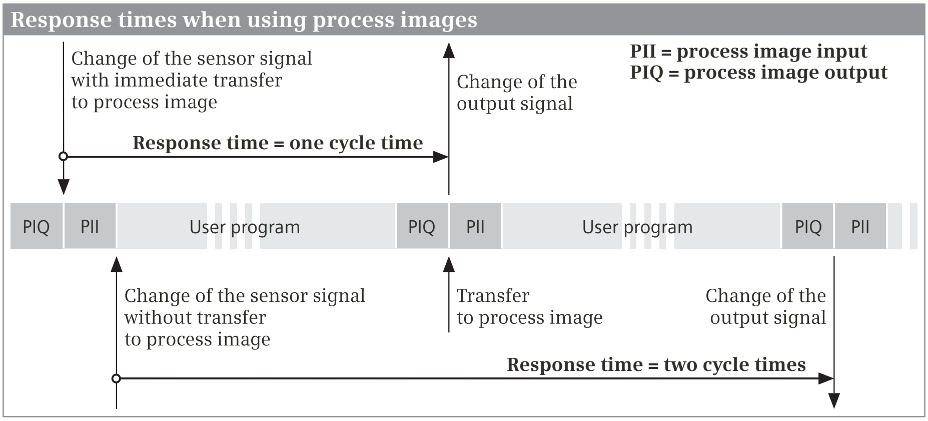

Response times when using process images
Change of the sensor signal with immediate transfer to process image
Response time = one cycle time
Change of the output signal
PII = process image input
PIQ = process image output
PIQ
PII
User program
PIQ
PII
User program
PIQ
PII
Change of the sensor signal without transfer to process image
Transfer to process image
Change of the output signal
Response time = two cycle times

Fig. 5.5 Reaction times in a programmable logic controller

If a limit switch is activated, for example, it changes its signal state from “0” to “1”. The controller detects this change during subsequent updating of the process image, and sets the input allocated to the limit switch to “1”. The program evaluates this change by resetting an output, for example, in order to switch off the corresponding motor. The new signal state of the output that was reset is transferred at the end of program execution; only then is the corresponding bit reset on the digital output module.

In a best-case situation, the process image is updated immediately following the change in the limit switch's signal. Then it only takes one cycle for the corresponding output to respond. In a worst-case situation, updating of the process image has just been completed when the limit switch's signal changes. It is then necessary to wait approximately one cycle for the controller to detect this change and set the input. Then the module output responds after one further cycle.

The reaction time to a change of the input signal may be between one and two cycle times. The delay times for the input modules and the switching times of contactors and so on also add to the reaction time.

The processing time of the user program thus contains all procedures in a program cycle (e.g. also processing of interrupts, processing in the operating system such as updating the timers, controlling the bus interface and updating the process image).

In individual cases, you can reduce reaction times by addressing the I/O directly or using event-controlled calls of specific program sections.

5.6 Program functions

The operating system of the CPU makes functions available that you can use from the user program. The program for these system functions is in the operating system, the function calls are in the user program. A selection of these system functions is listed in Table 5.6.

Table 5.6 CPU functions

Function, description		Present with		
		CPU 1200	CPU 300	CPU 400
WR_SYS_T RD_SYS_T SNC_RTCB SET_CLKS	Set clock (SET_CLK) Read clock (READ_CLK) Synchronizing the time-of-day Set time-of-day with time-of-day status	× × – –	× × – –	× × × ×
RTM	Control runtime meter	×	×	×
TIME_TCK	Reading the system time	–	×	×
RE_TRIGR	Retrigger cycle time	×	×	×
OB_RT	Measure the runtime of an organization block	–	–	×
STP	Stop program execution	×	×	×
WAIT	Delay program execution	–	×	×
COMPRESS	Compress the program	–	–	×
PROTECT	Protect the program	–	×	×
CIR	System modifications during operation	–	–	×

Real-time clock

The real-time clock provides the date and time. The time-of-day interrupts and impulses for the runtime meter are derived from the real-time clock. If the backup battery is working, the real-time clock continues to run even if the power supply is off. Without backup, on power on the clock starts with the time when the power was last switched off. A memory reset of the CPU has no effect on the real-time clock.

If several CPUs in a subnet are connected to each other, you can synchronize the clocks of the other CPUs automatically using a master clock. By calling the system function SNC_RTCB, you synchronize all clocks in the subnet regardless of the automatic synchronization. If you use SET_CLK or SET_CLKS to set a master clock, all other clocks in the subnet will be automatically synchronized to this value.

The time used in the CPU is the module time. Appropriately configured CPUs also save a time-of-day status. This includes a correction value which, added to the module time, produces the local time (display time). The local time can be used to visualize time zones.

Runtime meter

A runtime meter in a CPU counts the hours while running. You can use the runtime meter, for example, to record the operating hours of connected devices. If the CPU is in STOP or HOLD, the runtime meter stops too; if the CPU starts up again, counting continues at the last value. If the maximum duration has been reached, the runtime meter remains stationary and signals an overflow. A runtime meter can only be set to a new value or zero by means of a call from RTM. A memory reset of the CPU has no effect on the runtime meter.

The system function RTM is used to set a runtime meter to a new value, start or stop a runtime meter, or read the current value of the runtime meter. After a warm restart or cold restart you must also restart a runtime meter by means of RTM. The RTM system function processes runtime meters with runtimes of up to 2^{31}-1 hours.

Reading the system time

The system time of a CPU starts when the CPU is switched on. The system time runs for as long as the CPU is in the STARTUP or RUN mode. The current value of the system time is “frozen” when at STOP or HOLD. If there is a restart, the system time starts running again at the stored value. A cold restart or warm restart resets the system time. The system time is available in the data format TIME, where only positive values are possible. In the event of an overflow, the system time restarts at zero.

The TIME_TCK system function is used to read the current system time. You can use the system time, for example, to calculate the time between two TIME_TCK calls by calculating the difference.

Restart cycle monitoring time

Calling the system function RE_TRIGR in the main program restarts the cycle monitoring time. This then starts with the value set during CPU parameterization.

Measuring the program runtime of an organization block

The operating system of a CPU 400 saves the runtimes of the organization blocks using an internal timer in a microsecond grid. In the transition from STOP to RUN, the timer starts, runs to the upper limit of $2^{31}-1$ and begins again at zero. The data saved in the operating system can be read using the system function OB_RT for the last completed call and for the current call of the organization block. This enables you to determine the time load (utilization) of the user program.

Stopping the execution of the user program

The system function STP ends the processing of the user program. The CPU then updates the process image output and enters the STOP state. In the module properties of correspondingly designed modules, you can set the signal states of the digital and analog outputs which the CPU is to output in the STOP state: Retain last value or Connect substitute value. As standard, the signal state “0” is output at the digital outputs and a value of zero at the analog outputs at STOP. In the STOP operating mode, the CPU continues communication with the programming device and the diagnostics activities.

Delay execution of the user program

The system function WAIT holds program execution for a defined duration. The system function WAIT has the input parameter WT with data type INT, in which you can specify the hold time in microseconds (µs). The maximum hold time is 32 767 µs, the smallest possible hold time corresponds to the CPU-dependent execution time of the system function. WAIT can be interrupted by events of higher priority.

Compressing the user program

After repeated deleting and reloading of blocks, for example when modifying blocks online, gaps can form in the work memory in the CPU and the RAM load memory that reduce the usable storage area. The “compress” function can be used to trigger a program in the CPU operating system that fills these gaps by pushing the blocks together. You can initiate a “compress” operation with a connected programming device or by calling the system function COMPRESS.

The compression process is distributed over several program cycles. COMPRESS cannot compress if you are currently running an external compression process, if the “erase block” function is active, or if programming device functions are currently accessing the block to be moved (such as program status). Note that blocks above a certain CPU-specific maximum length cannot be moved with COMPRESS. Gaps thus remain in the CPU memory. Only the compression in STOP mode triggered by the programming device closes all the gaps.

Protecting the user program

The user program in a CPU can be protected against access in three protection levels: no protection, write protection, and read/write protection. You can set the protection level when parameterizing the CPU.

Program-driven toggling between protection levels “No protection” and “Write protection” is possible with the system function PROTECT. Calling the system function PROTECT is only effective if you have set the protection level “No protection” with the hardware configuration. It has no effect if write protection or read/write protection is set. The protection level set with PROTECT remains unchanged if

- ▷ the CPU goes to STOP due to a (program) error, an STP call, or operator intervention, or
- ▷ after switching the power supply off and on again.

In all other cases, the protection level “No protection” is set in the case of an operating mode transition. Even if you switch the mode switch to STOP, protection level “No protection” is (re)set.

Configuration in RUN

Configuration in RUN (CiR) means system modification in running operation. This functionality permits you to change the configuration of the distributed I/O of an S7-400 station without the CPU entering STOP or having to be set to STOP (Fig. 5.6).

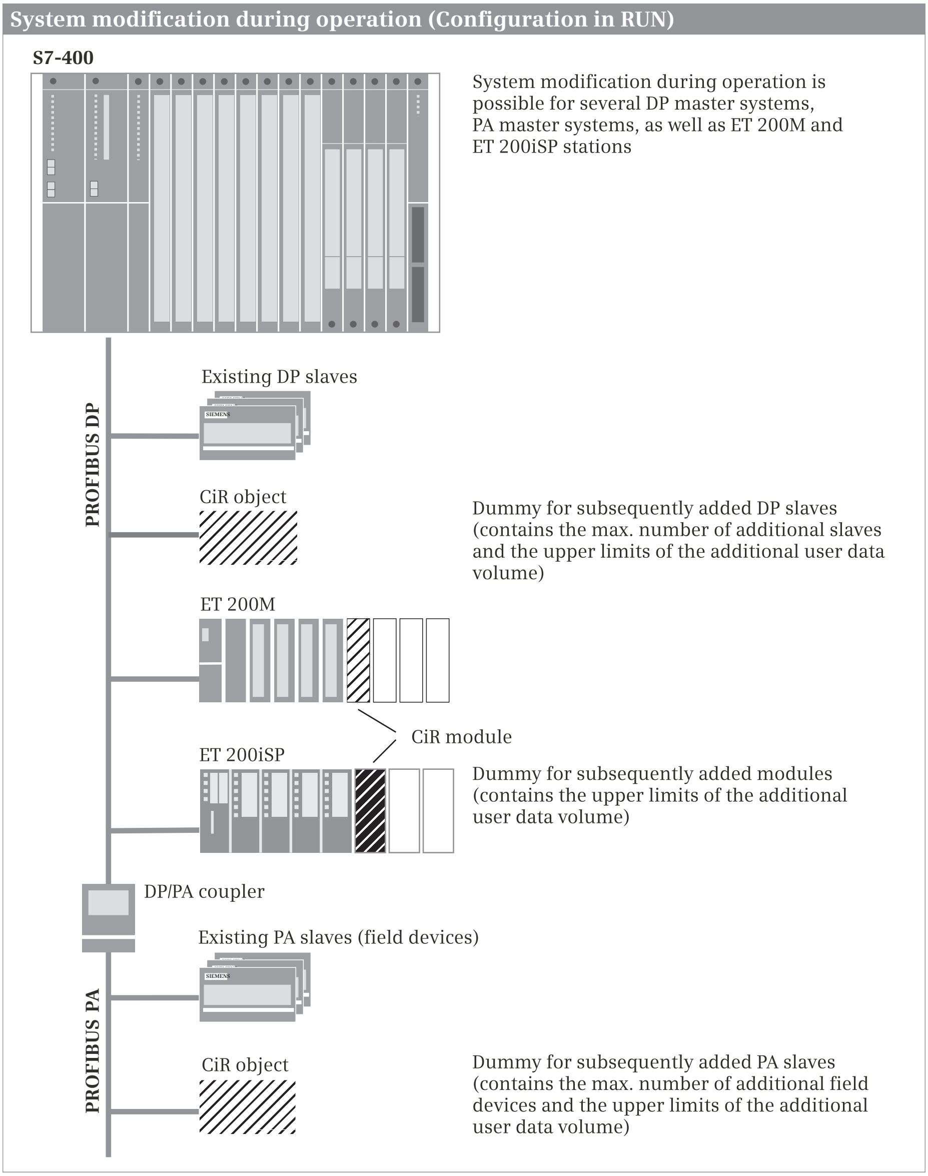

System modification during operation (Configuration in RUN)
S7-400
System modification during operation is possible for several DP master systems, PA master systems, as well as ET 200M and ET 200iSP stations
PROFIBUS DP
Existing DP slaves
SIEMENS
CiR object
Dummy for subsequently added DP slaves (contains the max. number of additional slaves and the upper limits of the additional user data volume)
ET 200M
CiR module
ET 200iSP
Dummy for subsequently added modules (contains the upper limits of the additional user data volume)
DP/PA coupler
Existing PA slaves (field devices)
SIEMENS
PROFIBUS PA
CiR object
Dummy for subsequently added PA slaves (contains the max. number of additional field devices and the upper limits of the additional user data volume)

Fig. 5.6 CiR elements in the hardware configuration

The changes comprise adding compact DP slaves, ET 200M stations, and PA master systems to an existing DP master system, adding modules to ET 200M stations, and adding PA slaves (field devices) to existing PA master systems. All objects added during running operation can also be removed during running operation. Components with and without CiR functionality can be used together; but modifications can only be made to components with CiR capability.

When reconfiguring, process execution is interrupted for a short period (typically 1 s, can be parameterized). The time can be kept short if only few modifications are always carried out.

5.7 Time-of-day interrupts

You use a time-of-day interrupt if you wish to execute a program once at a particular time or periodically, for example daily. For a CPU 300, the organization block OB 10 is provided for processing a time-of-day interrupt. For a CPU 400, the organization blocks OB 10 to OB 17 are provided. Which of these eight organization blocks are actually available depends on the CPU you are using. You can configure a time-of-day interrupt in HW Config in the CPU properties or control it from the user program during runtime using system functions. A time-of-day interrupt can only be triggered properly if the real-time clock is set correctly (Table 5.7).

Table 5.7 Organization blocks for time-of-day interrupts

Organization blocks, functions	CPU 1200	CPU 300	CPU 400
Permissible organization blocks	–	OB 10	OB 10 ... OB 17 *)
SET_TINT Set time-of-day interrupt	–	×	×
ACT_TINT Activate time-of-day interrupt	–	×	×
CAN_TINT Cancel time-of-day interrupt	–	×	×
QRY_TINT Query time-of-day interrupt	–	×	×

*) depending on the CPU type

Using a time-of-day interrupt

To start a time-of-day interrupt, you must first set the start time and then activate the time-of-day interrupt. You can carry out both activities separately using HW Config or also with system functions. Note that activation with HW Config means that the time-of-day interrupt is automatically started following parameterization of the CPU.

With SET_TINT you set the starting time and period of the time-of-day interrupt; with ACT_TINT you start the time-of-day interrupt. You can start a time-of-day interrupt once or periodically. The time-of-day interrupt is canceled following a single call of the time-of-day interrupt OB. You can also cancel an active time-of-day interrupt using CAN_TINT. If you wish to reuse a canceled time-of-day interrupt, you must set the start time again and activate the time-of-day interrupt. You can query the status of a time-of-day interrupt with QRY_TINT (Fig. 5.7).

Time-of-day interrupts are only executed if the CPU is in RUN mode. A time-of-day interrupt that is activated in a start-up routine is not started until the CPU goes into the RUN mode.

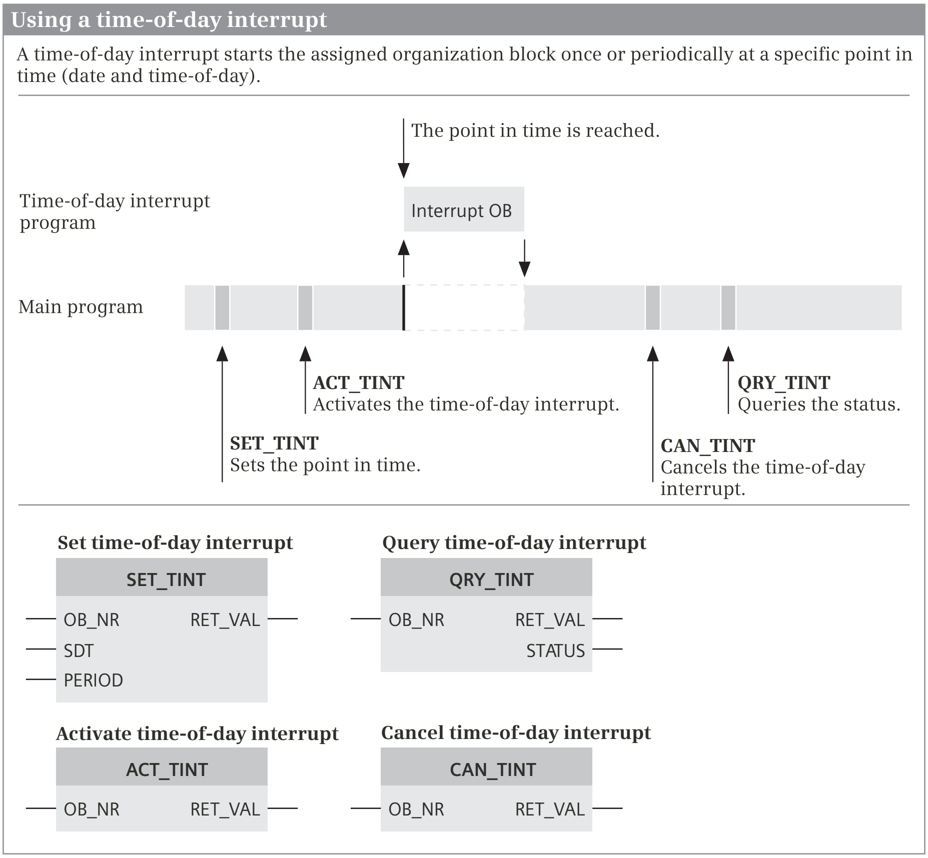

Using a time-of-day interrupt
A time-of-day interrupt starts the assigned organization block once or periodically at a specific point in time (date and time-of-day).
The point in time is reached.
Time-of-day interrupt program
Interrupt OB
Main program
ACT_TINT
Activates the time-of-day interrupt.
QRY_TINT
Queries the status.
SET_TINT
Sets the point in time.
CAN_TINT
Cancels the time-of-day interrupt.
Set time-of-day interrupt
SET_TINT
OB_NR
RET_VAL
SDT
PERIOD
Query time-of-day interrupt
QRY_TINT
OB_NR
RET_VAL
STATUS
Activate time-of-day interrupt
ACT_TINT
OB_NR
RET_VAL
Cancel time-of-day interrupt
CAN_TINT
OB_NR
RET_VAL

Fig. 5.7 Principle of time-of-day interrupt processing and associated system functions

5.8 Time-Delay Interrupts

A time-delay interrupt allows you to implement a time delay independent of the timer functions. For a CPU 300, the organization blocks OB 20 and OB 21 are provided for processing a time-delay interrupt. For a CPU 400, the organization blocks OB 20 to OB 23 are provided. Which of these four organization blocks are actually available depends on the CPU you are using (Table 5.8). With a CPU 1200, the organization blocks for time-delay interrupts can have a number from 20 to 23 and 123 or above. It is possible to have a maximum total of four organization blocks for time-delay interrupts and cyclic interrupts.

Table 5.8 Organization blocks for time-delay interrupts

Organization blocks, functions	CPU 1200	CPU 300	CPU 400
Permissible organization blocks	OB 20 ... OB 23 ≥ OB 123 **)	OB 20, OB 21	OB 20 ... OB 23 *)
SRT_DINT Start time-delay interrupt	×	×	×
CAN_DINT Cancel time-delay interrupt	×	×	×

*) depending on the CPU type

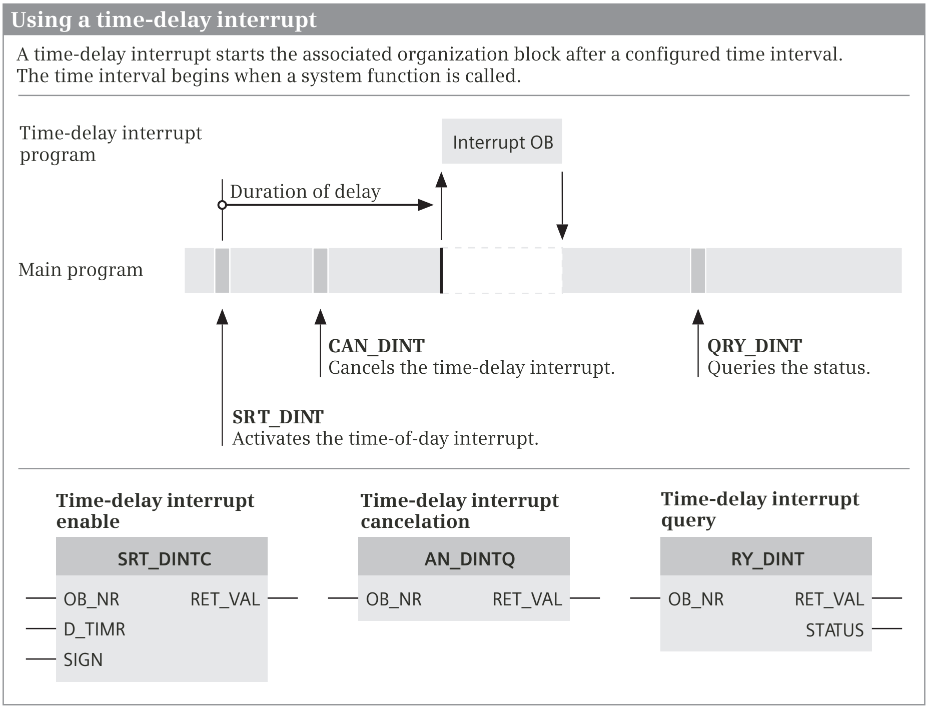
**) Together with cyclic interrupts, maximum four organization blocks are possible

Using time-delay interrupts

You activate a time-delay interrupt in the user program by calling the system function SRT_DINT. The call also transfers the delay duration and the number of the selected organization block to the operating system. The function call is simultaneously the start time for the parameterized period. After the delay time has expired, the selected organization block is started. Note that the processing of a time-delay interrupt organization block may be delayed if a higher-priority organization block is being processed at the time the interrupt OB is called (Fig. 5.8).

You can overwrite a running delay time by a new value by calling SRT_DINT again. The new time delay then commences when the function is called. You cancel an activated time-delay interrupt by calling the system function CAN_DINT. The associated organization block is then no longer called. You can query the status of a time-delay interrupt with QRY_DINT.

Time-delay interrupts are only executed if the CPU is in RUN mode. You can start a time-delay interrupt in the start-up routine by calling SRT_DINT. The CPU must be in RUN mode when the delay has expired. If not, the CPU waits to call the organization block until the end of the start-up routine and calls the time-delay interrupt OB during transition to RUN mode, even before starting the main program.



Using a time-delay interrupt

A time-delay interrupt starts the associated organization block after a configured time interval. The time interval begins when a system function is called.

Time-delay interrupt program
Interrupt OB
Duration of delay
Main program
CAN_DINT
Cancels the time-delay interrupt.
QRY_DINT
Queries the status.
SR T_DINT
Activates the time-of-day interrupt.
Time-delay interrupt enable
SRT_DINTC
OB_NR
RET_VAL
D_TIMR
SIGN
Time-delay interrupt cancelation
AN_DINTQ
OB_NR
RET_VAL
Time-delay interrupt query
RY_DINT
OB_NR
RET_VAL
STATUS

Fig. 5.8 Principle of time-delay interrupt processing and associated system functions

5.9 Cyclic Interrupts

A cyclic interrupt is an interrupt triggered at periodic intervals and initiates execution of a cyclic interrupt organization block. You can use a cyclic interrupt to have a certain program processed at a time interval which is not dependent on the processing time of the cyclic program. For a CPU 300, the organization blocks OB 32 to OB 35 are provided for processing a cyclic interrupt. For a CPU 400, the organization blocks OB 30 to OB 38 are provided. Which of these nine organization blocks are actually available depends on the CPU you are using (Table 5.9). With a CPU 1200, the organization blocks for cyclic interrupts can have a number from 30 to 38 and 123 or above. It is possible to have a maximum total of four organization blocks for time-delay interrupts and cyclic interrupts.

Table 5.9 Organization blocks for cyclic interrupts

Organization blocks	CPU 1200	CPU 300	CPU 400
Permissible organization blocks	OB 30 ... OB 38 ≥ OB 123 **)	OB 32 to OB 35	OB 30 ... OB 38 *)

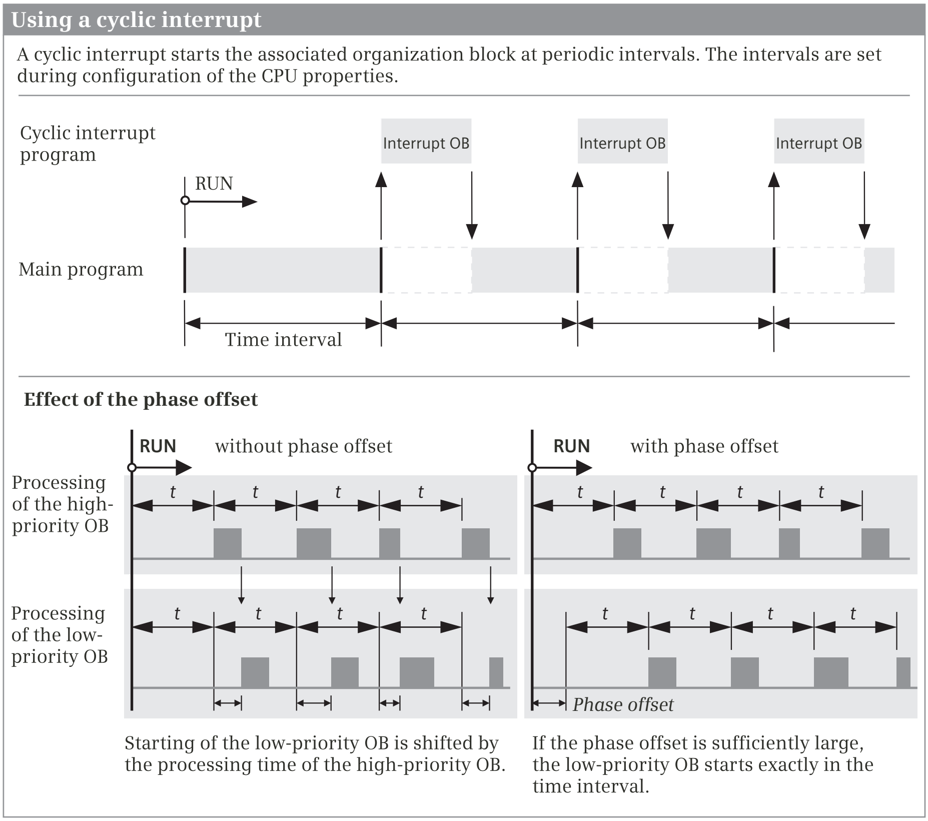
*) depending on the CPU type

**) Together with time-delay interrupts, maximum four organization blocks are possible

Using cyclic interrupts

A cyclic interrupt has three parameters: time interval (execution), phase offset, and priority. You can use the phase offset to process cyclic interrupt programs with a time delay even though they have a common multiple in the time interval. This results in higher accuracy of the processing period because the lower-priority organization block does not have to wait.

The time intervals and phase offset commence upon transition to the RUN operating mode. The call instant for a cyclic interrupt OB is thus the time interval plus the phase offset. An example is shown in Fig. 5.9. No phase offset is set in the left section, and consequently start of processing of the lower priority organization block is delayed by the current processing time of the higher priority organization block in each case.



Using a cyclic interrupt

A cyclic interrupt starts the associated organization block at periodic intervals. The intervals are set during configuration of the CPU properties.

Starting of the low-priority OB is shifted by the processing time of the high-priority OB.

If the phase offset is sufficiently large, the low-priority OB starts exactly in the time interval.

Fig. 5.9 Principle of cyclic interrupt processing

5.10 Hardware Interrupts

You use hardware interrupts to immediately detect events in the controlled plant or machine in the user program and respond with an appropriate routine. For a CPU 300, the organization block OB 40 is provided for processing a hardware interrupt. For a CPU 400, the organization blocks OB 40 to OB 47 are provided. Which of these eight organization blocks are actually available depends on the CPU you are using (Table 5.10). With a CPU 1200, the organization blocks for hardware interrupts can have a number from 40 to 47 and 123 or above. A maximum total of 50 organization blocks are possible for hardware interrupts.

Table 5.10 Organization blocks for hardware interrupts

Organization blocks, functions	CPU 1200	CPU 300	CPU 400
Permissible organization blocks	OB 40 ... OB 47 ≥ OB 123 **)	OB 40	OB 40 ... OB 47 *)
ATTACH Assign hardware interrupt during runtime DETACH Cancel hardware interrupt assignment	× ×	– –	– –

*) depending on the CPU type

**) Maximum 50 organization blocks possible

Triggering a hardware interrupt

A hardware interrupt is triggered on a module designed for this. This can be, for example, a digital input module which records a signal coming from the process, or a function module which triggers a hardware interrupt due to an event on the module. With a CPU 1200, the integrated high-speed counters (HSC) can also trigger hardware interrupts.

Triggering of a hardware interrupt is initially disabled by default. When you define the parameters for the module using the HW Config tool, you enable the processing of a hardware interrupt. You can select whether the hardware interrupt is to be triggered by an incoming event, by an outgoing event, or by both. When defining the parameters for the module, you also assign an organization block to the hardware interrupt. The assignment of an organization block to an interrupt event can also occur during runtime for a CPU 1200 (Fig. 5.10).

Hardware interrupts are only triggered when the CPU is in RUN mode. Hardware interrupts that occur during startup are rejected.

Scanning for interrupt information with a CPU 300/400

In the start information of the interrupt organization block, you can query which module triggered the interrupt. Bytes 5, 6 and 7 of the start information contain the start address of the module. Bytes 8 to 11 contain the signal status of the inputs for digital input modules, and the interrupt status for other modules.

Using a hardware interrupt

A hardware interrupt is triggered on a correspondingly configured module and starts the assigned organization block. An organization block is assigned to a hardware interrupt during the configuration of the module properties. For a CPU 1200, this assignment and cancelation of the assignment can also take place during runtime.

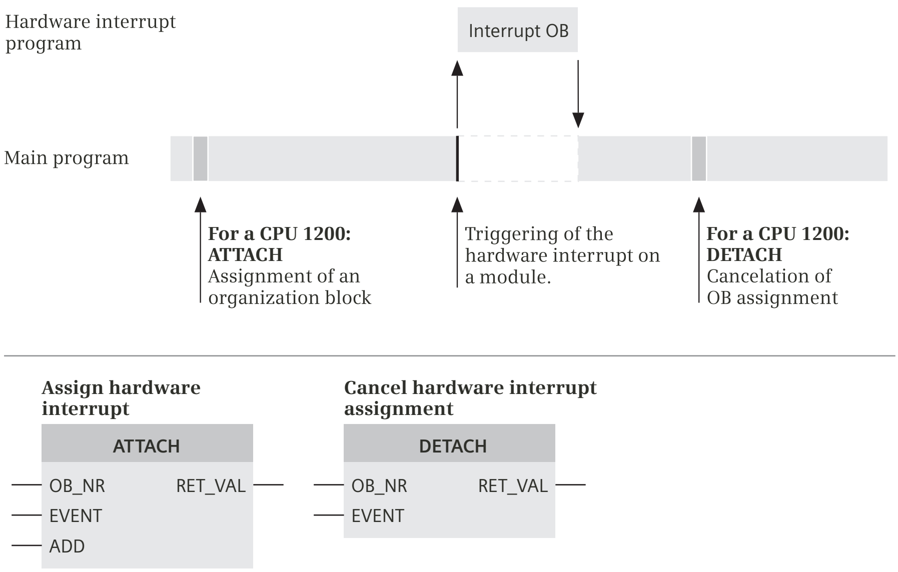

Hardware interrupt program
Interrupt OB
Main program
For a CPU 1200: ATTACH
Assignment of an organization block
Triggering of the hardware interrupt on a module.
For a CPU 1200: DETACH
Cancelation of OB assignment
Assign hardware interrupt
ATTACH
OB_NR
EVENT
ADD
RET_VAL
Cancel hardware interrupt assignment
DETACH
OB_NR
EVENT
RET_VAL

Fig. 5.10 Principle of hardware interrupt processing and associated system functions

5.11 Multiprocessor Interrupt

In multiprocessor mode, the multiprocessor interrupt permits you to respond to an event synchronously in all participating CPUs. The system function MP_ALM triggers the multiprocessor interrupt. The organization block OB 60 is available for processing the multiprocessor interrupt. Multiprocessor operation is only possible in an S7-400 station. It is automatically activated if there is more than one CPU in the rack.

Triggering the multiprocessor interrupt

Calling the system function MP_ALM generates a multiprocessor interrupt. You can assign an identifier to the call of the system function so that, for example, you can determine in the program in which CPU the interrupt was triggered.

The multiprocessor interrupt OB is started on all CPUs simultaneously. This means that the CPU in which the system function MP_ALM was called also waits to call OB 60 until all other CPUs have reported that they are ready. The organization block for the multiprocessor interrupt is only started in RUN mode. A call of the system function MP_ALM in the start-up routine is rejected with the return of an error message (Fig. 5.11).

Using a multiprocessor interrupt

A multiprocessor interrupt is triggered on a CPU. Once all other CPUs have signaled that they are ready, the assigned organization block starts in all CPUs simultaneously.

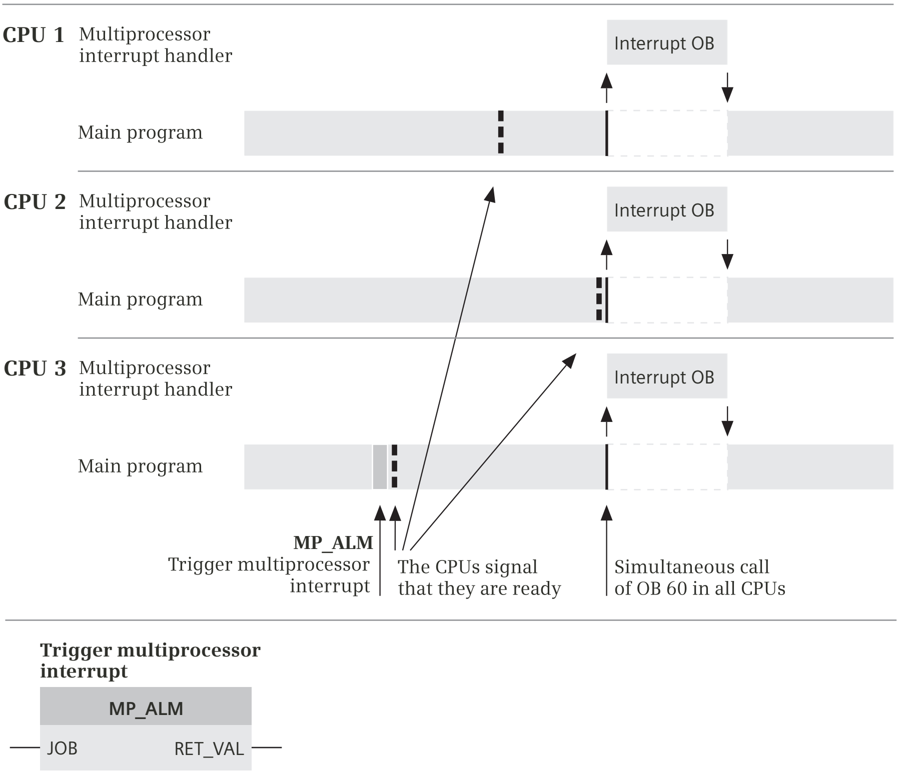

CPU 1
Multiprocessor interrupt handler
Interrupt OB
Main program
CPU 2
Multiprocessor interrupt handler
Interrupt OB
Main program
CPU 3
Multiprocessor interrupt handler
Interrupt OB
Main program
MP_ALM
Trigger multiprocessor interrupt
The CPUs signal that they are ready
Simultaneous call of OB 60 in all CPUs
Trigger multiprocessor interrupt
MP_ALM
JOB
RET_VAL

Fig. 5.11 Principle of processing a multiprocessor interrupt

5.12 Synchronous errors with a CPU 300/400

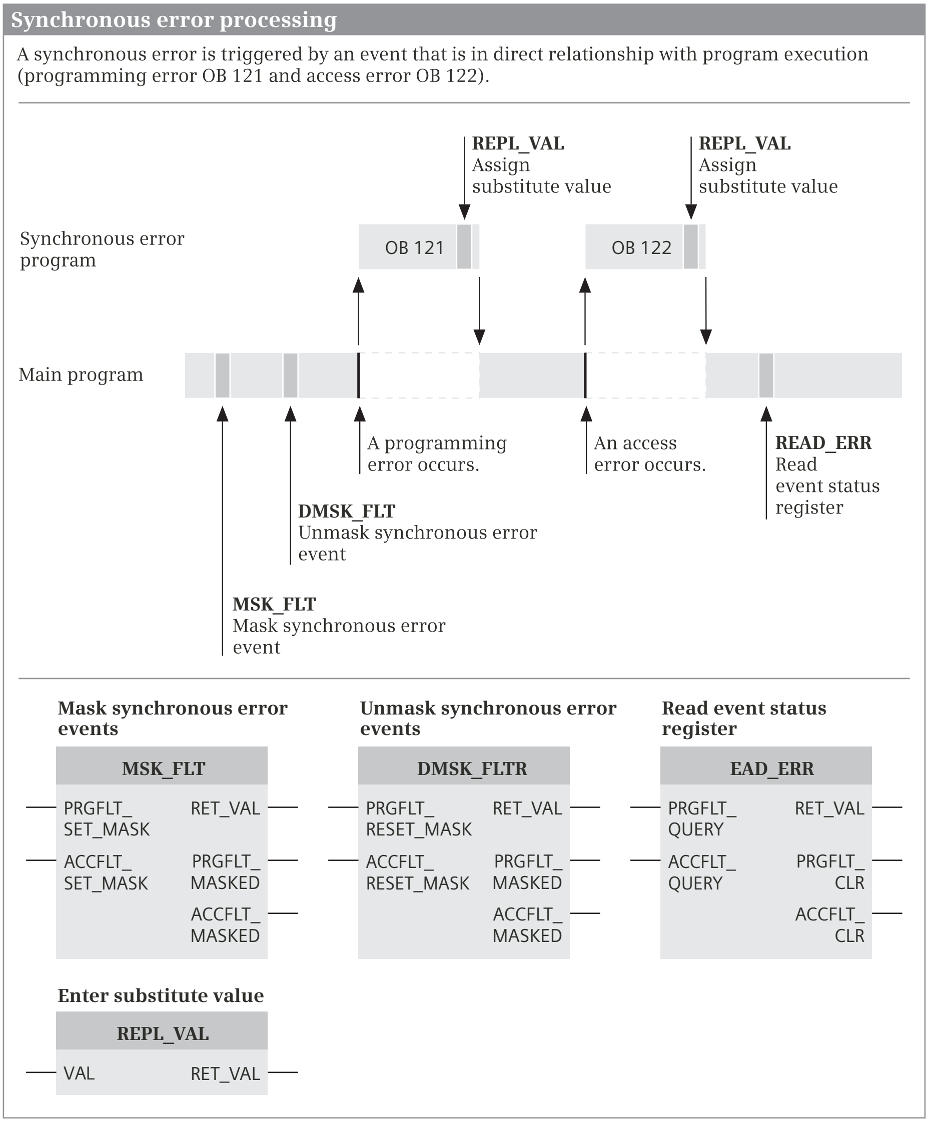
The operating system of the CPU generates a synchronous error event when an error occurs which is directly related to program processing. If you have not programmed a synchronous error OB, the CPU goes into the STOP mode when a synchronous error event occurs. There are two types of errors.

- Programming errors; OB 121 is called
- Access errors; OB 122 is called

A synchronous error OB has the same priority as the block that caused the error. Therefore, the registers of the interrupted block can be accessed by the synchronous error OB. Another effect is that the program in the synchronous error OB can return the registers with modified contents to the interrupted block.

Please note that, when a synchronous error OB is called, its 20 bytes of start information are stored in the L stack of the error-causing priority class, in addition to the remaining temporary local data of the synchronous error OB and the temporary local data of all blocks called in this OB.

Fig. 5.12 shows the program execution in a synchronous error event and the system functions that can be used in this context.



A synchronous error is triggered by an event that is in direct relationship with program execution (programming error OB 121 and access error OB 122).

Synchronous error processing
REPL_VAL Assign substitute value
REPL_VAL Assign substitute value
Synchronous error program
OB 121
OB 122
Main program
A programming error occurs.
An access error occurs.
READ_ERR Read event status register
DMSK_FLT Unmask synchronous error event
MSK_FLT Mask synchronous error event
Mask synchronous error events
MSK_FLT
PRGFLT_SET_MASK
ACCFLT_SET_MASK
RET_VAL
PRGFLT_MASKED
ACCFLT_MASKED
Unmask synchronous error events
DMSK_FLTR
PRGFLT_RESET_MASK
ACCFLT_RESET_MASK
RET_VAL
PRGFLT_MASKED
ACCFLT_MASKED
Read event status register
EAD_ERR
PRGFLT_QUERY
ACCFLT_QUERY
RET_VAL
PRGFLT_CLR
ACCFLT_CLR
Enter substitute value
REPL_VAL
VAL
RET_VAL

Fig. 5.12 Principle of synchronous error processing and associated system functions

Entering a substitute value

Using the REPL_VAL system function, you can write a substitute value into accumulator 1 from a synchronous error OB. Example: Values can no longer be read from an input module; OB 122 *access error* is then called and with REPL_VAL a substitute value can be entered and processing can continue with this value.

Masking and unmasking synchronous errors

If a synchronous error occurs, the operating system calls the respective organization block or – if the OB is not available – changes to the STOP operating mode. Certain synchronous error events can be "masked out" so that they no longer cause a switchover to STOP or a call of the error organization block. If a masked synchronous error event occurs, the CPU stores this in an event status register.

The system function MSK_FLT is used to mask the synchronous error events. A bit in an error mask is assigned to each synchronous error. This masking only takes effect in the current priority class (program processing level) in which MSK_FLT is called. You read the event status register by means of the system function READ_ERR to detect a synchronous error which occurred during masking. Call system function DMSK_FLT to unmask the synchronous error and re-enable processing.

5.13 Asynchronous errors with a CPU 300/400

Asynchronous errors are errors which can occur regardless of program processing. When an asynchronous error occurs, the operating system calls one of the following organization blocks:

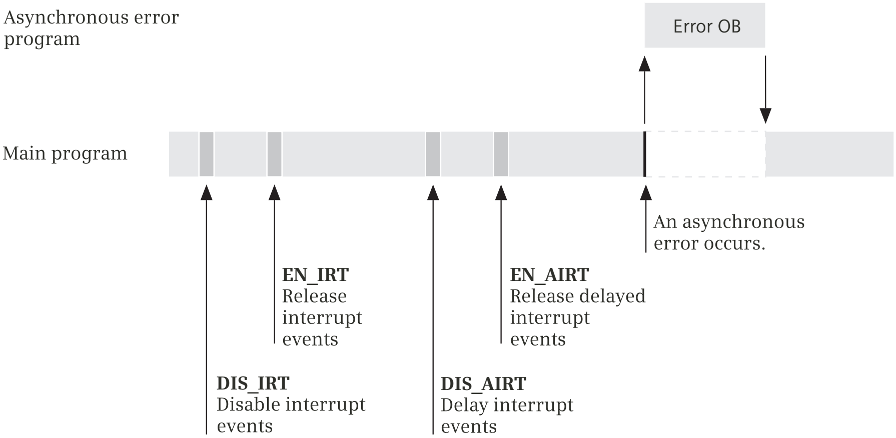
- Time error – OB 80
  A time error means, for example, that the cycle monitoring time has been exceeded, an OB has been required while it was still being processed, or a time-of-day interrupt has expired because the real-time clock was set forward.
- Power supply error – OB 81
  This OB is called when a battery on the CPU rack or an expansion rack is empty, battery backup has completely failed, or the 24 V supply has failed.
- Diagnostic interrupt OB 82
  A module with corresponding capability uses a diagnostic interrupt to report the diagnostic event to the CPU.
- Insert/remove module interrupt – OB 83
  The operating system monitors the module configuration every few seconds. Each time a module is removed or inserted during RUN, STOP and STARTUP, an entry is made in the diagnostic buffer and the system state list, and OB 83 is called. If a suitable module is installed in the configured slot, the module is automatically loaded with the parameter set and data records by the CPU. Only then is OB 83 called to report that the installed module is ready again.

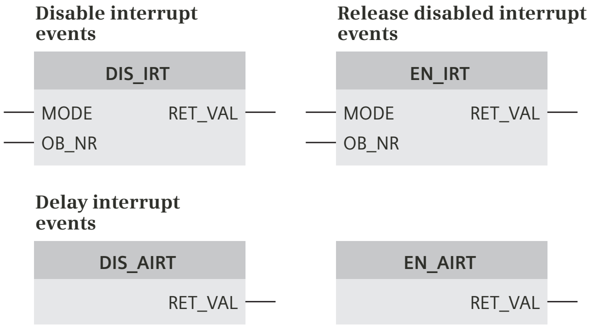
Asynchronous error processing

An asynchronous error is triggered by an event that is not in direct relationship with program execution.

Asynchronous error	**Error OB**	**Present with**	
		CPU 300	**CPU 400**
Time error	OB 80	x	x
Power supply error	OB 81	—	x
Diagnostics interrupt	OB 82	x	x
Insert/remove module interrupt	OB 83	*)	x
CPU hardware fault	OB 84	—	x
Program execution error	OB 85	x	x
Rack failure	OB 86	*)	x
Communications error	OB 87	x	x
Processing interrupt	OB 88	—	x

*) not for all CPU types


Asynchronous error program
Error OB
Main program
An asynchronous error occurs.
EN_IRT Release interrupt events
EN_AIRT Release delayed interrupt events
DIS_IRT Disable interrupt events
DIS_AIRT Delay interrupt events



These functions permit processing of interrupt and asynchronous error events.

Fig. 5.13 Principle of asynchronous error processing and associated system functions

▷ CPU hardware error – OB 84
The operating system calls organization block OB 84 when an interface error (MPI network, PROFIBUS DP) occurs or disappears.

▷ Program execution error OB 85
The operating system calls organization block OB 85 if a non-existent organiza-

tion block is to be called, if there is a block access error in the operating system (e.g. missing instance data block when an SFB system function block is called), and if an I/O access error occurs while the process image is being updated (automatically) on the system side.

- ▷ Module rack failure – OB 86
  The operating system calls organization block OB 86 when it detects a module rack failure (power failure, interrupted line, defective interface), or a failure of a subnet or a station of the distributed I/O.
- ▷ Communication error – OB 87
  The operating system calls organization block OB 87 when a communication error occurs (e.g., wrong telegram identifier or telegram length error during global data communication, error during time synchronization).
- ▷ Execution abort – OB 88
  The operating system calls organization block OB 88 when the execution of a block in the user program is aborted because, for example, the permissible block nesting depth has been exceeded.

The CPU operating system allows existing interrupt events to be ignored or to be processed at a later time. The processing of the interrupt event is controlled with system functions (Fig. 5.13).

Disabling, delaying and enabling asynchronous errors and interrupts

Asynchronous errors and interrupts during cyclic program processing are usually unpredictable. They can interrupt the main program at any point to call the interrupt or error program. A lower-priority interrupt program can also be interrupted in this way. This interruption can have a negative effect when the interrupted program section must be processed within a certain time, for example when a short reaction time is essential, or when an instruction sequence may not be interrupted, as would be the case during reading of consecutive values from an I/O module.

You can use system function DIS_IRT to disable interrupts and asynchronous errors. Events that occur after the disable are rejected. The related organization block is not called, nor does the CPU go to STOP due to a non-existent interrupt OB. You have several choices: disabling all interrupts and asynchronous errors, disabling asynchronous errors or an interrupt class (e.g., only the hardware interrupts), or disabling an individual interrupt or asynchronous error organization block. The disable action applies to all priority classes, that is, to all levels of program processing. If you want to cancel the disable action, call system function EN_IRT. This re-enables processing of all interrupts and asynchronous errors, even after a warm restart and a cold restart.

Delaying the interrupts and asynchronous errors with system function SFC 41 DIS_AIRT suppresses their processing. Processing resumes when the delay is canceled with system function SFC 42 EN_AIRT. The delay of interrupt and asynchronous error processing can also be nested. In this case, make sure that the delay is enabled as many times as it was activated.

5.14 Error handling with a CPU 1200

Program execution error

You can specify the response of a CPU 1200 to a faulty block call or an I/O access error yourself: Use system responses or run the “local error handling”. By default, the system responses – depending on the error – are “ignore error” or “go into STOP mode”. If you use the local error handling for a block, it has precedence over the system response.

You activate the (block-)local error handling by calling one of the functions GetErrorID *Read program error number* or GetError *Read program error information* (Fig. 5.14). Local error handling is effective only in the block in which GetErrorID or GetError is called.

The default responses are as follows with the local error handling active:

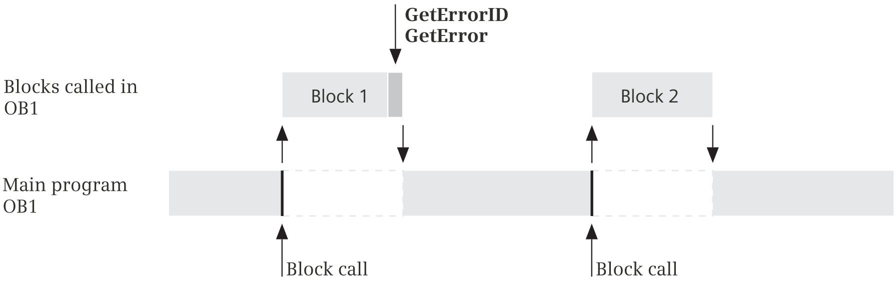
▷ With a write error: the error is ignored, and program execution is continued.

▷ With a read error: the substitute value FALSE or zero is read, and program execution is continued.

▷ With an execution error: execution of the faulty instruction (faulty function) is aborted, and program execution is continued with the next instruction.

Local error handling with a CPU 1200

In a logic block, local error handling replaces the system response to a program execution error or an I/O area access error.



In Block 1, local error handling is activated by calling either the function GetErrorID or GetError and errors are signaled using an error number (with *GetErrorID*) or by means of detailed error information (with *GetError*). If there is an error in Block 2, the operating system triggers the standard system response.

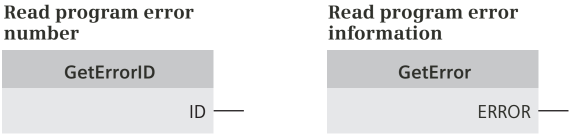


Fig. 5.14 Principle of local error handling with a CPU 1200

GetErrorID makes an error number available, and GetError makes the complete error information available in the *ErrorStruct* data type. You can assign the *ErrorStruct* data type to a variable in a data block or in a block interface (of a local variable). You can evaluate the error number or error information in the user program and respond appropriately.

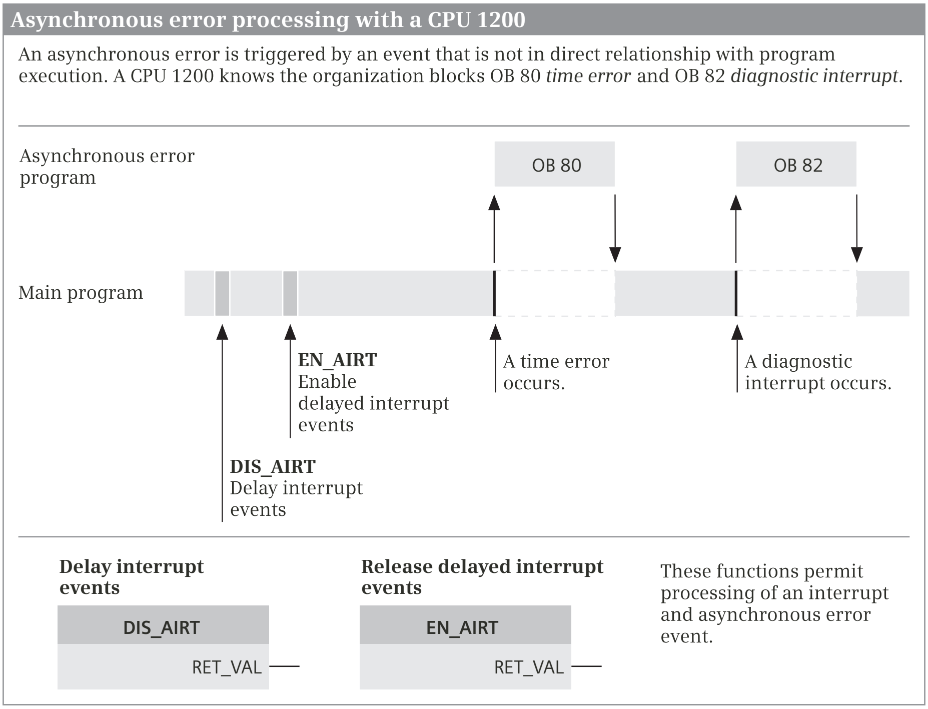
GetError and GetErrorID return the first detected error. If several errors occur at the same time, the error with the highest priority will be displayed. GetError and GetErrorID do not save the error pattern.

Time error OB 80

The operating system calls the organization block OB 80 *time error* if one of the following events occurs:

- ▷ Cycle monitoring time exceeded
- ▷ OB request error: the requested organization block is still being processed (possible with time-delay and cyclic interrupts) or an organization block is requested too frequently within a given priority class (queue overflow)
- ▷ An interrupt is lost due to interrupt overload.

The time error organization block is assigned to event class *Time error interrupt*. Only one time error organization block can be programmed. The error is ignored if OB 80 is not present when a time error occurs (Fig. 5.15).



Asynchronous error processing with a CPU 1200

An asynchronous error is triggered by an event that is not in direct relationship with program execution. A CPU 1200 knows the organization blocks OB 80 *time error* and OB 82 *diagnostic interrupt*.

These functions permit processing of an interrupt and asynchronous error event.

Fig. 5.15 Principle of asynchronous error processing with a CPU 1200

Diagnostic interrupt OB 82

Appropriately designed modules can detect diagnosis events, for example “No load voltage present” (I/O modules), overshoot and undershoot, wire break and short-circuit (analog input modules). If the detection of a diagnosis event is activated in the device configuration, the organization block OB 82 *diagnostic interrupt* is called if the event occurs.

The diagnostic interrupt organization block is assigned to the event class *diagnostic interrupt*. You can only use one diagnostic interrupt OB in your program. The OB 82 is processed if no other interrupt organization block is active, otherwise the diagnosis event is entered into the queue. If a diagnosis event occurs, it is written into the diagnostic buffer. If an OB 82 is not present when a diagnosis event occurs, the CPU ignores the event.

Calling of the diagnostic interrupt OB can be delayed or enabled using the DIS_AIRT and EN_AIRT functions.

5.15 Diagnostic functions with a CPU 300/400

Several system functions support diagnostics during runtime. How you detect causes of error with programming device support is described in chapter Chapter 3.16 "Finding hardware faults using diagnostic functions" on page 106.

Reading start information

The system function RD_SINFO provides the start information of the current organization block – this is the organization block at the top of the call tree – and that of the last executed startup organization block even at a lower call level (Fig. 5.16).

The TOP_SI output parameter contains the first 12 bytes of the start information of the current OB, the START_UP_SI output parameter contains the first 12 bytes of the start information of the last executed startup OB. The time stamp is not included in either case.

Calling of RD_SINFO is not only permissible at any position within the main program but also in each priority class, including the program of an error organization block or in the startup. For example, if RD_SINFO is called in an interrupt organization block, TOP_SI contains the start information of the interrupt OB. TOP_SI and START_UP_SI have identical contents when calling in the startup.

Read system state list

The system state list (SSL) describes the current status of an automation system. The contents of the SSL can only be read using information functions, contents cannot be modified. Since the complete system state list is extremely comprehensive, reading is carried out in partial lists and partial list extracts. The partial lists are only compiled by the CPU's operating system and read using the RDSYSST function upon request (Fig. 5.16).

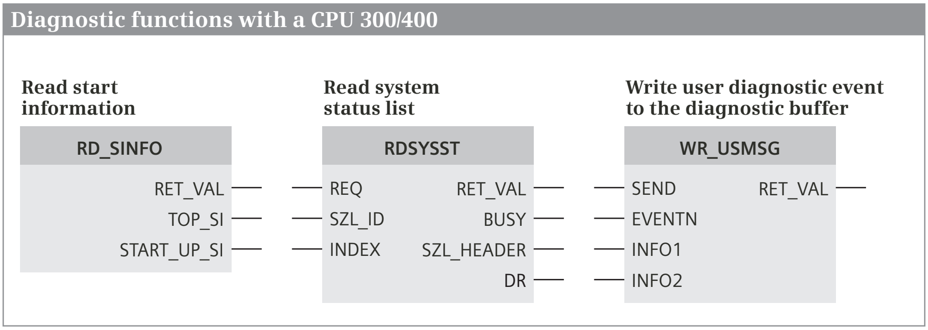


Fig. 5.16 Diagnostic functions with a CPU 300/400

The SSL ID is available for identification of a partial list. This contains the module type class to which the list applies, the number of the partial list extract, and the actual system state partial list number. Together with the index which specifies an object of a partial list, you are provided with the desired information. The CPU always provides information on the automation system as standard, but FM and CP modules can also use this service to provide information (see documentation on module). You can find the possible system state lists of a CPU in the description of the operation.

You trigger reading with REQ = “1”, and BUSY = “0” indicates completion of the read operation. You supply parameter SSL_ID with the partial list number and the parameter INDEX with an object from the partial list. The parameter SSL_HEADER contains the length of a data record and the number of transmitted data records. In the DR parameter you specify the data area in which RDSYSST is to enter the read data records.

Write user diagnostic event to the diagnostic buffer

You use the system function WR_USMSG to write an entry into the diagnostic buffer and you can send it to all the stations that are logged in (Fig. 5.16). The entry in the diagnostic buffer has the same structure as a system event, e.g. the start information of an organization block.

Permitted as event ID (parameter EVENTN) for a user entry are: The event classes 8 (diagnostics entries for signal modules), 9 (standard user events), A and B (freely available user events). Additional information 1 (parameter INFO1) corresponds to the bytes 7 and 8 of the buffer entry (one word) and additional information 2 (parameter INFO2) to the bytes 9 to 12 (one doubleword). The contents of both variables are freely selectable.

If the SEND parameter has signal state “1”, the diagnostic event is also sent to the nodes logged on for this purpose.

5.16 Overview of user blocks

STEP 7 allows you to break down your user program into self-contained program sections which have a specific function, structure or relate to a technological task. These program sections are called blocks. With large and complex projects, structuring of the program into individual blocks is recommended and sometimes a necessity. The blocks are called for processing either in succession or in nested sequence.

Organization blocks

Organization blocks (OBs) represent the interface between operating system and user program. The CPU's operating system calls an organization blocks if a specific event occur, e.g. in the event of a hardware or time-of-day interrupt. The main program is in organization block OB 1. The other organization blocks correspondingly have predefined numbers corresponding to the call events or can be assigned a number on a CPU 1200. In the program of an organization block, function blocks and functions are then called as needed. Mutual interruptibility of program processing is handled by priority control.

Function blocks

Function blocks (FBs) are parameterizable parts of the user program which can statically store some of their block-specific data in a permanently assigned data block. A different data block can be assigned to each function block call. Such a permanently assigned data block is called an instance data block, and the combination of function block call and instance data block is referred to as a call instance, or “instance” for short. Function blocks can also store their variables as “local instances” in the instance data block of the calling function block, the “multi-instance”.

Functions

Functions (FCs) are preferably used to program frequently recurring automation functions. They can be parameterized and supply a return value (the function value) to the calling block. The function value is optional. In addition to the function value, functions can also have output parameters. Functions do not store information and have no assigned data block.

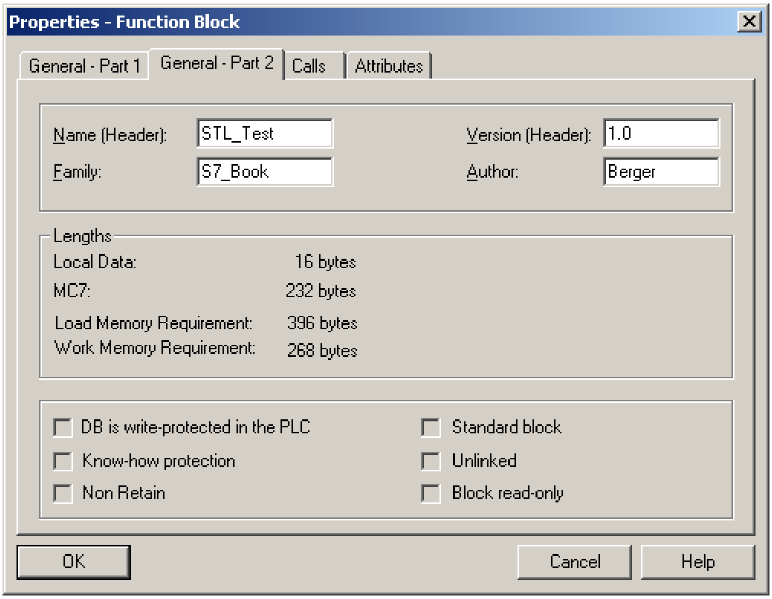
Data blocks

Data blocks (DB) contain the data of the user program. A data block can be generated as a global data block, as an instance data block, or as a type data block. With a global data block, you program the data variables directly in the data block. With an instance data block, the programming of the assigned function block determines the data variables present in the data block. A type data block has the structure of a user-defined data type UDT (PLC data type).

5.17 Block properties

Block properties with STEP 7 V5.5

You can view and modify the block properties with the menu command *Edit* > *Object Properties* in the SIMATIC Manager when the block is selected, or with *File* > *Properties* in the program editor (Fig. 5.17).



Name (Header):	STL_Test	Version (Header):	1.0
Family:	S7_Book	Author:	Berger

Lengths

Local Data:	16 bytes
MC7:	232 bytes
Load Memory Requirement:	396 bytes
Work Memory Requirement:	268 bytes

□ DB is write-protected in the PLC □ Standard block
□ Know-how protection □ Unlinked
□ Non Retain □ Block read-only

Fig. 5.17 Block properties with STEP 7 V5.5

The block properties are found in two tabs. The *General - Part 1* tab contains the absolute address, the symbolic address, the symbol comment, the creation language, and the project path. Furthermore, time stamps and block comments are displayed. The tab *General - Part 2* informs about the following properties:

The block header contains the *Name (Header)*. It is used for identifying the block and is not identical to the symbolic address. Different blocks can have the same name. Under *Family* you can assign a common feature to a group of blocks. Under *Author* you can enter the creator of the block and under *Version (Header)* the block version.

The section *Lengths* provides information on the memory requirements of the block in the load memory and in the work memory. *MC7* indicates the length of the program code. Under *Local Data* you can see the memory requirements in the local data stack (L stack); this also includes the temporary local data used by the editor, which is not visible in the program.

The property *Know-how protection* stands for block protection (see Chapter 5.18 "Know-how protection, copy protection" on page 197).

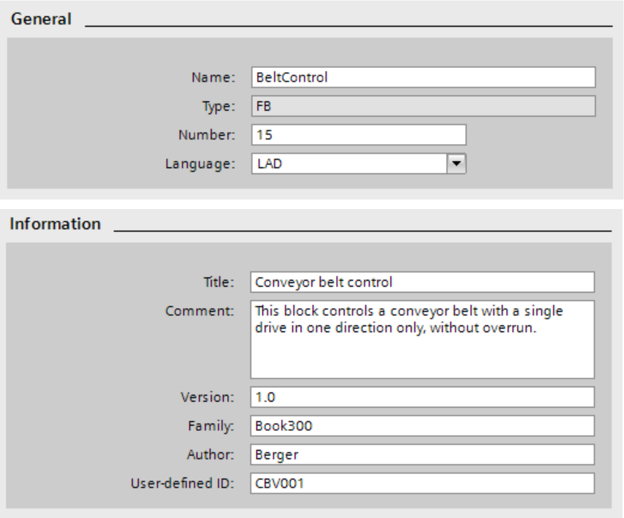
DB is write-protected in the PLC means that you can only read from this data block using a program. A data block with the property *Unlinked* is only located in the load memory. It is not relevant to processing. Data blocks in the load memory can be read and, in connection with a MicroMemory Card, also written to. The property *Non Retain* defines the retentivity of data blocks.

Block read-only indicates that a block has been saved such that it can be monitored but not modified.

The designation *Standard block* is located in the block header of standard blocks supplied by Siemens.

Block properties with STEP 7 V11 inside TIA Portal

To display and change the block properties, select the block in the project tree and then the *Properties* command in the shortcut menu. Fig. 5.18 shows as example the *General* and *Information* sections of the block properties of a function block.



General

Name: BeltControl

Type: FB

Number: 15

Language: LAD

Information

Title: Conveyor belt control

Comment: This block controls a conveyor belt with a single drive in one direction only, without overrun.

Version: 1.0

Family: Book300

Author: Berger

User-defined ID: CBV001

Fig. 5.18 Block properties *General* and *Information* for STEP 7 inside TIA Portal

The *Time stamp* section indicates the date of creation of the block and the date of the last modification to the block, interface, and program. The *Compile* section provides information on the processing status of the block, and the *Protection* section indicates the block protection (see Chapter 5.18 "Know-how protection, copy protection" on page 197).

Table 5.11 lists the *block attributes* for data blocks and logic blocks with the LAD, FBD or STL programs. Blocks with SCL and GRAPH program have additional attributes.

Table 5.11 Block attributes

Attribute	for the block	Meaning with attribute activated
IEC check	OB, FB, FC	A stricter test of the data types takes place.
Multiple instance capability *)	FB	The function block can be called in another function block as a local instance.
Data block write-protected in the device	Global DB, type DB	The data of the data block cannot be overwritten by means of a program during runtime.
Only store in load memory	Global DB, type DB	The data block is not transferred to the work memory; it is only present in the load memory.
Optimized block access **)	OB, FB, FC, DB	The block-specific variables are stored in memory-optimized form and can only be addressed symbolically. The retentivity can be set for individual variables and the LREAL data type can be used.
Handle errors within block **)	OB, FB, FC	The default system response to a program execution and access error is suspended in favor of a self-programmed error routine.

*) Only shown for blocks in a CPU 300/400
**) Only available for S7-1200

The *IEC check* attribute indicates how strict the data type test is to be in the logic block. With the attribute not activated, it is usually sufficient if the variables used have the data width required for execution of the function or instruction; with the attribute activated, the data types of the variables must correspond to the required data types.

The *Multi-instance capability* attribute is only present with function blocks. If the *Multi-instance capability* attribute is activated (this is the standard setting), you can call the block as a local instance in another function block.

Data block write-protected in device is an attribute for a global and type data block. It means that you can only read from this data block using a program. Overwriting of the data is prevented and an error message is generated. The write protection applies to the data relevant to execution (actual values) in the work memory; the data in the load memory (initial values) can be overwritten even if the data block is provided with write protection. Write protection must not be confused with block protection: A data block with block protection can be read and written by the program; however, its data can no longer be viewed using a programming or monitoring device.

Global and type data blocks can be assigned the *Only store in load memory* attribute. Such types of data block are only present in the load memory, they are “not relevant to execution”. Since their data is not in the work memory, direct access is not possible. Data in the load memory can be read and with a CPU 300 and CPU 1200 also written using system functions. Data blocks with the *Only store in load* *memory* attribute are suitable for data which is only accessed rarely, e.g. recipes or archives.

The *Optimized block access* attribute causes variables to be stored in memory-optimized form. For instance data blocks, the attribute is “inherited” from the assigned function block. The *Optimized block access* attribute also has effects on the retentivity setting of the data variables: with the attribute activated, individual variables can be set as retentive (in the associated function block for instance data blocks), but only the complete block can be set when the attribute is not activated. With an activated attribute, the local data can only be addressed symbolically. Finally, the data type LREAL can only be used in blocks with the *Optimized block access* attribute activated.

The *Local error handling in the block* attribute is activated as soon as one of the functions *GetError* or *GetErrorID* is inserted when programming the block. The system reaction to a program execution or access error is then suppressed in favor of a self-programmed error routine.

5.18 Know-how protection, copy protection

With the know-how protection for a block you can prevent a program or its data from being read out, printed, or modified. Using copy protection, you link the processing of a block to a specific CPU or specific memory card.

Know-how protection with STEP 7 V5.5

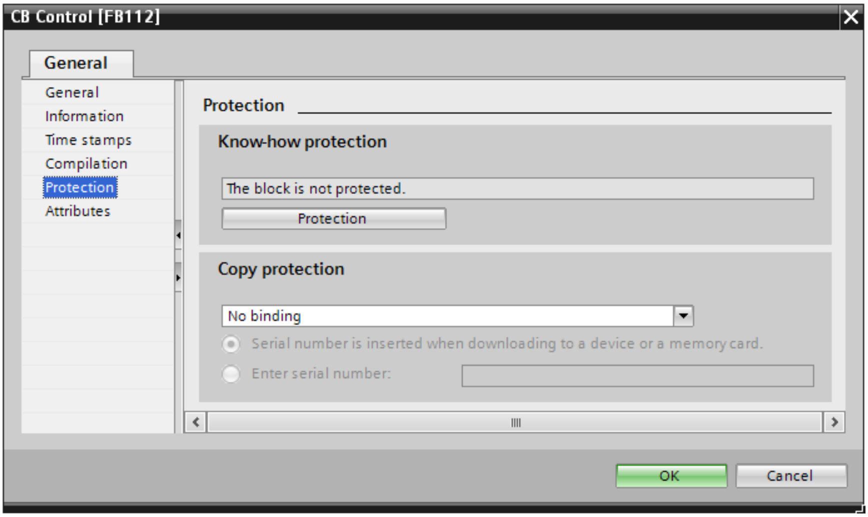
In STEP 7 V5.5, the know-how protection of a block is set in the source-oriented programming. The keyword is KNOW_HOW_PROTECT. It is specified in the block header of the block to be protected. If the block is compiled, it is protected. This protection can no longer be removed; the program source file should thus be kept available.

Know-how protection with STEP 7 V11 inside TIA Portal

To activate the know-how protection in STEP 7 V11, select the block in the project tree, and choose *Know-how protection* from the shortcut menu. Click on *Define* in the dialog window, enter a password, and confirm it. If a function block is protected, the protection is “inherited” by the instance data block when calling as a single instance.

You change the password for the know-how protection by clicking on *Protection* in the *Protection* block properties and then on *Change* in the following dialog window (Fig. 5.19). You remove the know-how protection by clicking on *Protection* in the *Protection* section of the block properties, deactivate the *Don't Show Code (Know-How Protection)* checkbox in the dialog window, enter the password, and click *OK*.

Note: If the password is lost, no further access to the block is possible. You can only cancel the know-how protection of a block in its offline version. If you download a



CB Control [FB112]

General

Protection

Know-how protection

The block is not protected.

Copy protection

No binding

○ Serial number is inserted when downloading to a device or a memory card.

○ Enter serial number:

Fig. 5.19 Know-how protection and copy protection with STEP 7 inside TIA Portal

compiled block to the CPU, the recovery information is lost. A protected block which you have uploaded from the CPU cannot be opened, not even with the correct password.

Copy protection with STEP 7 V11 inside TIA Portal

Copy protection can be set up for blocks in a CPU 1200. The processing of the block then depends on a particular CPU or a particular memory card. So that the copy protection cannot be removed, the block must then be provided with the know-how protection.

When the copy protection is being set up, the know-how protection for the block must be switched off. To set up the copy protection, select the block in the project tree, select *Properties* from the shortcut menu and then *Protection* (Fig. 5.19). In the *Copy protection* area, you can choose:

▷ No binding
No copy protection is set or a set copy protection is canceled.

▷ Link to serial number of memory card
The block can only be executed if the memory card has the specified serial number.

▷ Link to serial number of CPU
The block can only be executed if the CPU has the specified serial number.

To enter the serial number, the options *Serial number is inserted when downloading to a device or a memory card* and *Enter serial number* are available with an input field for the serial number.

5.19 Block interface

The block interface of a logic blocks contains the declaration of the block-specific variables. These are the variables that you only use in this block. Functions FC and function blocks FB can exchange data with the calling block via *Block parameters*; there are input, output, and in/out parameters. Each logic block has *temporary local data*, in which intermediate results are stored. Function blocks ultimately save the *static local data* in the instance data block.

Block parameters

Block parameters represent the transfer interface between the calling block and the called block. By means of block parameters you enable parameterization of the processing specification (the block function) present in a block. In the block interface, you specify – depending on the block type – the name and data type of the variables used for each variable type (Table 5.12).

Table 5.12 Variable types in the block interface

Variable type	Declaration section	Possible in block mode		
Input parameter	Input, IN	-	FC	FB
Output parameter	Output, OUT	-	FC	FB
In/out parameters	InOut, IN_OUT	-	FC	FB
Static local data	Static, STAT	-	-	FB
Temporary local data	Temp, TEMP	OB	FC	FB
Function value	Return, RETURN	-	FC	-

An *input parameter* transfers a value to the program in the block and may only be read. Input parameters are shown in the block call in the sequence of their declaration, with LAD and FBD on the left side of the call box and with STL and SCL at the start of the parameter list.

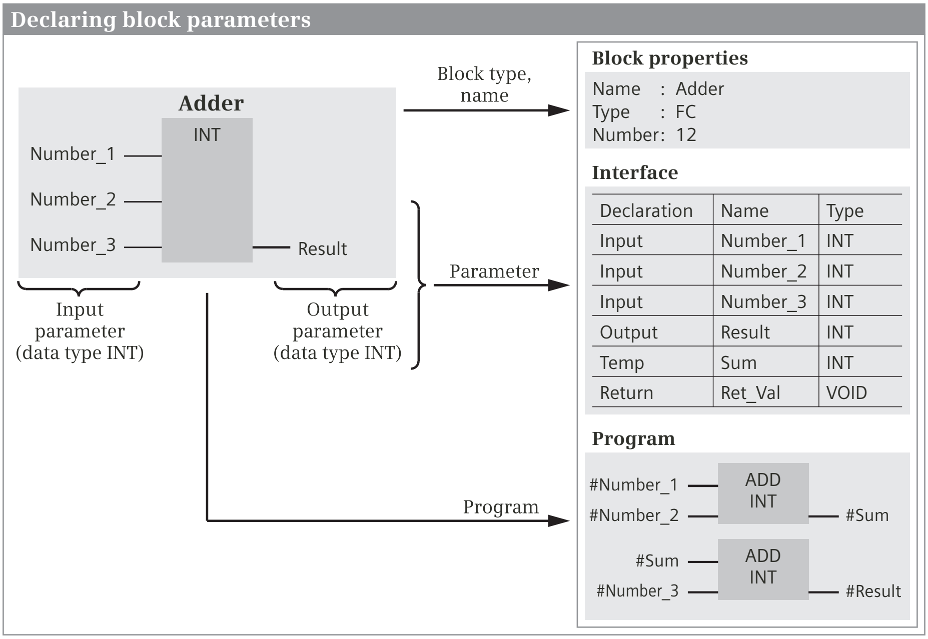
An *output parameter* transfers a value to the calling block and may only be written. Output parameters are shown in the block call in the sequence of their declaration, with LAD and FBD on the right side of the call box and with STL and SCL following the input parameters in the parameter list.

An *in/out parameter* transfers a value to the program in the block and can return it to the calling block, usually with a changed content. An in/out parameter can be read and written. In/out parameters are shown in the block call in the sequence of their declaration, with LAD and FBD on the left side of the call box under the input parameters and with STL and SCL at the end of the parameter list.

The *function value* for functions is an output parameter which is handled in a special manner. As standard it has the name *Ret_Val* (RET_VAL) with the declaration *Return* (RETURN) and the data type VOID (= without type). In this case it is not displayed. If the function value is declared with a different data type, it is the first output parameter. In the SCL programming language, the function can then be integrated into an expression (see section "Calling a function FC" on page 202).

Example of use of block parameters

A block must add three variable values of data type INT. The block should be used in the program multiple times with different values. Since the block does not have to permanently (statically) cache variable values, a function FC is suitable for this application. The variable values are transferred via block parameters, in this case three input parameters and one output parameter (Fig. 5.20).



Declaration	Name	Type
Input	Number_1	INT
Input	Number_2	INT
Input	Number_3	INT
Output	Result	INT
Temp	Sum	INT
Return	Ret_Val	VOID

Fig. 5.20 Example of declaration of block parameters

In the example, the variables *#Sum* and *#Ret_Val* are also declared. *#Sum* stores the intermediate result in the temporary local data. *#Ret_Val* has data type VOID and is not shown.

The names of the block parameters are present in the program of the adder as dummies for the subsequent variable values; here, they are called *formal parameters*. You use the formal parameters as symbolically addressed variables.

Temporary local data

Temporary local data are local block-specific variables. They store intermediate results which are generated during program processing of a block. The signal states or values of the temporary local data are only available while the block is being processed. They are lost after the block is concluded.

You declare the temporary local data in the block interface of the logic block. Usually, the temporary local data is addressed symbolically, but it is also possible to use absolute addressing with a CPU 300/400. You address a bit with L *y.x*, a byte with LB *y*, a word with LW *y*, and a doubleword with LD *y* (*y* = byte address, *x* = bit address).

Start information for organization blocks

When an organization block is called, the operating system of a CPU 300/400 transfers start information in the temporary local data. This start information is 20 bytes long for each organization block and contains notes on the start event; for example, for a hardware interrupt, the address of the module that generated the interrupt.

The operating system of a CPU 1200 provides start information only for the organization blocks OB 100 (start-up routine), OB 80 (time error), and OB 82 (diagnostic interrupt). This start information is in the block interface in the *Input* declaration section.

Static local data

Static local data are block-specific variables in function blocks. These are in the instance data block to which each call of a function block is allocated. For a local instance, this can also be the instance data block of the calling function block ("multi-instance").

The static local data retains its value until it is changed by the program, as do the data addresses in global data blocks. You declare the static local data in the block interface of the function block. The static local data is addressed symbolically within the function block. Since the static local data is in a data block, it can, like data variables, be addressed with *"data block".localvariable* (for a single instance), with *"data block".instancename.localvariable* (for a multi-instance), or with the absolute address. For function blocks, the same applies to the block parameters.

5.20 Calling blocks

If the program of a block should be processed, the block must be called. When calling, the block can be parameterized, i.e. the block is given values it needs to work with.

In the graphical programming languages LAD and FBD there is a box for a block call. Above the box is the address of the calling block in absolute or symbolic form. From the left the (block) inputs lead to the box; on the right are the (block) outputs. Via the inputs the called block receives the values with which it is to work. The block returns the results via the outputs. In the textual languages STL and SCL, the block parameters are a list after the actual call statement.

The values transferred in the block call are called *actual parameters*. These can be constant values, addresses, variables, data areas, or blocks. This depends on the type of block parameter and on the processing within the called block. An actual parameter must have the same data type as the block parameter that it supplies with a value.

Calling a function FC

When calling a function FC you must supply all block parameters.

The function value of a function FC is not shown on a call if the function value is of the data type VOID. If the function value has a different data type, it is shown in LAD, FBD, and STL as the first output parameter. In SCL the function FC in this case returns a value that has the data type of the function value. The function can thus be used in an expression like a variable with the data type of the function value. An example is shown in Fig. 5.21.

Declaration of the function value in the "Adder2" function

Block interface

Input	Number_1	INT
	Number_2	INT
	Number_3	INT
Return	**Result**	**INT**

The block interface contains the three input parameters and the function value as the result of the addition
#Result := #Number_1 + # Number_2 + # Number_3;
The declaration as function value has no effects for the program in the "Adder2" block. The program could also have been created using one of the programming languages LAD, FBD, or STL.

Call of the "Adder2" block in the SCL program

SCL

```
//Call within an exprssion
#Variable_1 := "Adder2" (Number_1 := %MW30, Number_2 := %MW32, Number_3 := %MW34) + #Variable_2;
```

The "Adder2" function can now be used in an expression in the programming language SCL.
The function has the data type which was assigned to the function value.

Fig. 5.21 Use of the function value with SCL

Calling a function block FB

When calling a function block FB, supplying of block parameters is optional. Block parameters that you do not supply retain their old value, because the values in the instance data block are stored.

When calling a function block, the instance data block belonging to the call must also be specified. The call can be executed as a “single instance”, in which case each call has its own instance data block. If the function block in the program of a function block is called, it is alternatively possible to store the instance data of a function block called as a “local instance” in the instance data block of the calling function block, the “multi-instance” (Fig. 5.22).

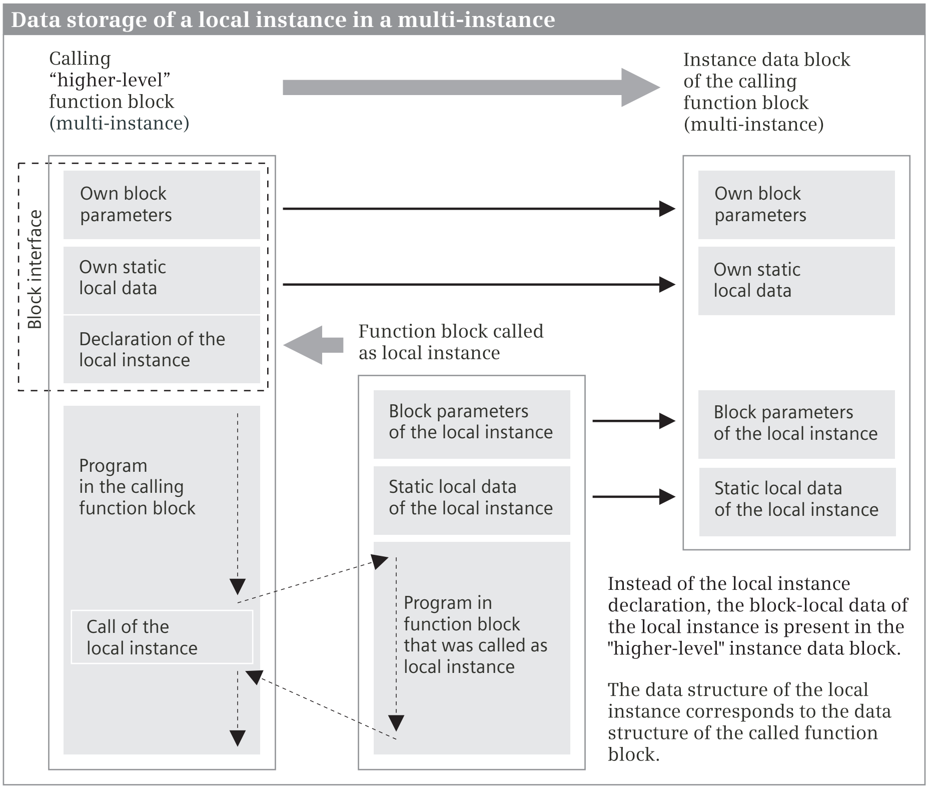

Data storage of a local instance in a multi-instance
Calling "higher-level" function block (multi-instance)
Instance data block of the calling function block (multi-instance)
Block interface
Own block parameters
Own static local data
Declaration of the local instance
Function block called as local instance
Program in the calling function block
Call of the local instance
Block parameters of the local instance
Static local data of the local instance
Program in function block that was called as local instance
Own block parameters
Own static local data
Block parameters of the local instance
Static local data of the local instance
Instead of the local instance declaration, the block-local data of the local instance is present in the "higher-level" instance data block.
The data structure of the local instance corresponds to the data structure of the called function block.

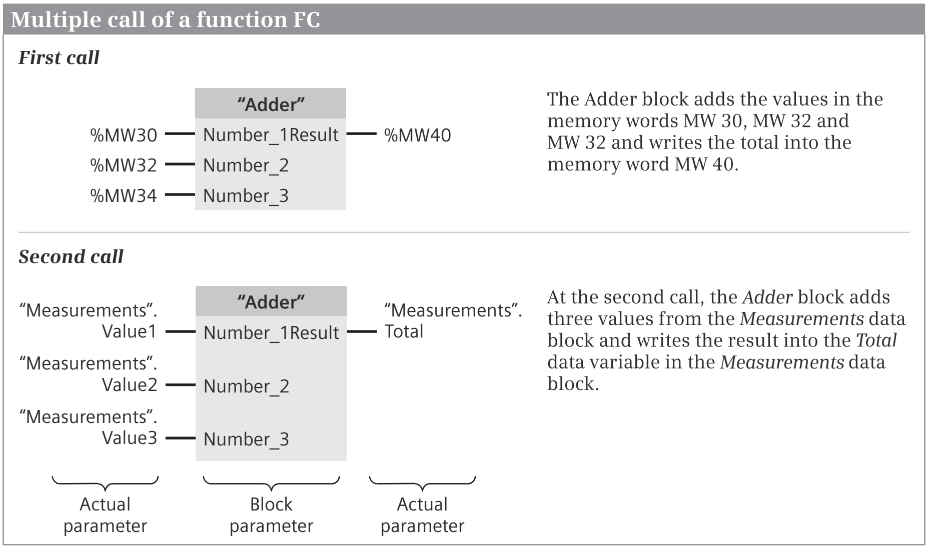
Fig. 5.22 Data storage for a local instance

You can declare any number of function blocks to be local instances, including those that themselves contain local instances. The nesting depth for local instances has the value 8. You can also enter a function block multiply as a local instance – each with a different name. The limitation is the length of the data block: It must be able to accept all the data of subordinate (including nested) local instances.

Multiple calling of a block

You can call a block several times in the program and provide the call with other addresses every time. With each call, the block function is then executed with different values.

Example of a function FC: The *Adder* function described in the chapter should be called twice, the first time with absolutely addressed memory words and the second time with symbolically addressed data addresses. Fig. 5.23 displays a graphical representation of the block calls based on the representation of STEP 7 inside TIA Portal.



The Adder block adds the values in the memory words MW 30, MW 32 and MW 32 and writes the total into the memory word MW 40.

At the second call, the *Adder* block adds three values from the *Measurements* data block and writes the result into the *Total* data variable in the *Measurements* data block.

Fig. 5.23 Multiple call of a function FC

6 Communication

Each SIMATIC S7 station can exchange data with another station. Simple data exchange is generally performed by the CPU itself. Data exchange with connection protocols that the CPU is not capable of is performed by communication modules.

A network is a group of stations for the purpose of communication. It consists of one or more subnets of the same or different kinds linked together. A subnet includes all communication nodes that are connected to each other over a hardware link with uniform physical properties and transfer parameters such as transfer rate and that exchange data via a shared transfer method.

Depending on the requirements, various subnets can be selected:

- MPI (Multi Point Interface) is designed for easy data exchange between S7-300/400 stations.
- Industrial Ethernet permits the rapid exchange of large amounts of data between S7 stations and to third-party devices. PROFINET IO is the open Industrial Ethernet standard for the connection of distributed I/O stations.
- PROFIBUS is the transmission standard for data, process and fieldbus communication. PROFIBUS DP allows the rapid exchange of data with distributed I/O stations.
- AS-Interface (AS-i) is used to query sensors and control actuators on the fieldbus level.
- A point-to-point connection (PtP) uses the physical standard RS232 or RS485 to link a SIMATIC station to an input/output device via a serial connection.

The data exchange is configured with STEP 7. *Networking* connects the S7 stations involved in data exchange to a subnet. The protocol for data traffic is then defined with a *connection*.

In the user program, communication functions ensure the handling of data traffic. The communication functions are either integrated as system modules in the operating system of the CPU or they are provided as standard blocks in the Program Elements catalog or in libraries.

6.1 Configuring the network

The network configuration permits the graphic display and documentation of configured networks and their stations (nodes). It is a part of the device configuration. If a PLC station is operated on its own – without an HMI station and without data communication to other PLC stations – the network configuration is not required. Connection of a programming device to transfer the user program and for program testing does not require configuration either.

General procedure

You use HW Config in the project to connect the stations to be networked to the necessary “communication-capable” modules – the CPU and CP modules. Here you can also add the relevant subnets to the project, connect the stations with the subnets, and set the node addresses.

In the network configuration, you then configure the communication links. A connection is needed for S7 communication (event-driven communication service to exchange large amounts of data) or if the communication partner is not a SIMATIC PLC station. The number of possible connections is CPU-specific. STEP 7 specifies a connection ID for each connection and partner. You need this information when you use the communication functions in your program.

In a *connection configured at one end* (one-sided data traffic) with only one partner, the communication functions in the user program control the data traffic. The operating system takes over the data traffic in the remote partner. In a *two-sided connection* (two-sided data traffic), the communication functions are present in both partners and can initiate the exchange of data from the user program.

To exchange data between two stations belonging to different projects, select “unspecified” (in the local station in both projects) as the connection partner in the connection table. Make sure that the connection data is consistent in both projects.

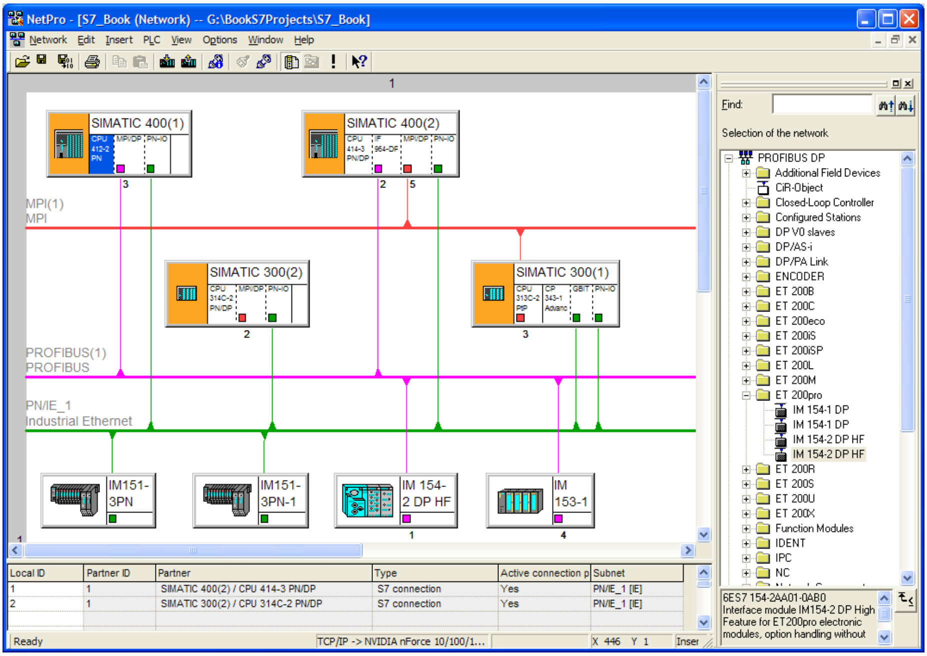
In the network configuration, you can also add and network further stations and subnets. If the network configuration is complete, start the compilation process, which generates the configuration data for the networked stations.

Network configuration with STEP 7 V5.5

To start the network configuration, open the project using the SIMATIC Manager and select *Options* > *Configure network* from the main menu. Alternatively, in SIMATIC Manager you can also double-click on the icon of an existing subnet in the project folder or on the icon of the connection table in the CPU folder.

The network configuration shows the existing stations and subnets and their networking in a separate window (Fig. 6.1). To add a station, drag it from the network object catalog to the working area. If the network objects catalog is not visible, you can display it in the network window by choosing *View* > *Catalog*. You add a subnet in the same way.

To equip the newly added station with a communication-capable module, double-click on the station to start HW Config, choose a rack, and attach at least the CPU or CP module. Save the configured station and return to the network configuration.



Local ID	Partner ID	Partner	Type	Active connection p	Subnet
1	1	SIMATIC 400(2) / CPU 414-3 PN/DP	S7 connection	Yes	PN/IE_1 [IE]
2	1	SIMATIC 300(2) / CPU 314C-2 PN/DP	S7 connection	Yes	PN/IE_1 [IE]

6ES7 154-2AA01-0AB0
Interface module IM154-2 DP High
Feature for ET200pro electronic
modules, option handling without

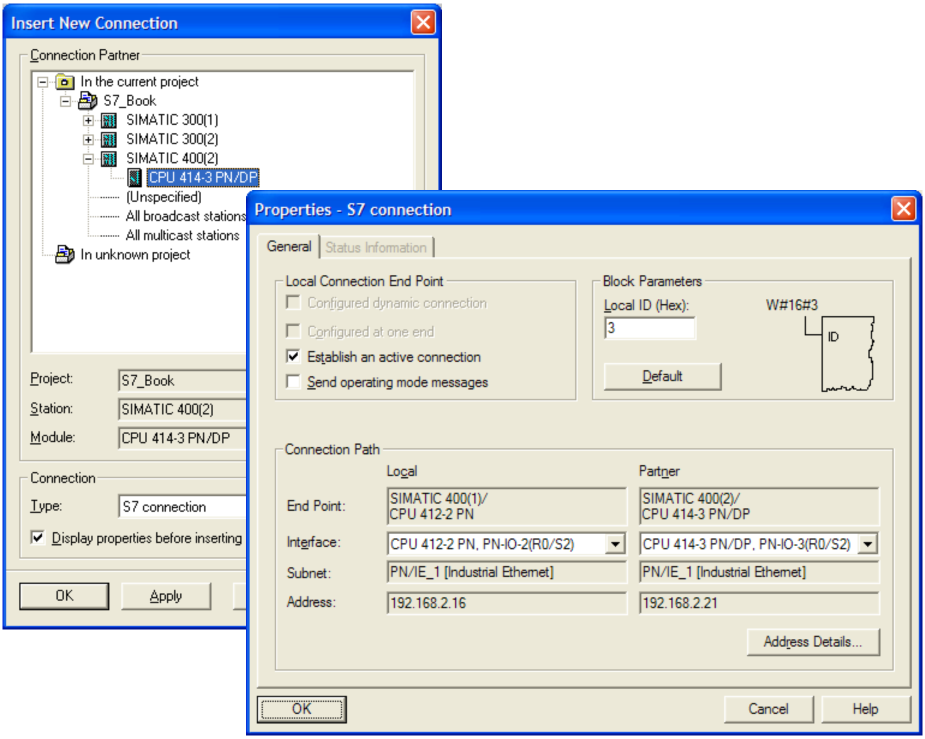
Fig. 6.1 Example of the network window for STEP 7 V5.5

For networking, drag a bus interface to the appropriate subnet. Alternatively, select the bus interface and choose *Object Properties* from the shortcut menu. In the dialog box, you can now connect the bus interface to a subnet, or break off the connection again. In addition, you can change the node address here.

If you select a CPU, the connection table is displayed in the lower part of the network window. If you select a subnet, you are given an overview of the networked stations and their node addresses.

The established connections are listed in the connection table. You add a new connection by selecting a row and choosing *Insert New Connection* in the shortcut menu. In the dialog box, select the connection partner. If a connection exists, change the connection partner with *Connection Partner...* from the shortcut menu or set the connection properties with *Object Properties...* (Fig. 6.2).

The *Network* > *Check Consistency* command causes STEP 7 to check whether the current configuration and parameter assignment produce consistent connection data. Save the network configuration with *Network* > *Save*. To compile, select *Network* > *Save and Compile*. The compiled configuration data is stored in the *System Data* object in the *Blocks* folder.



Connection Partner

- In the current project
  - S7_Book
    - SIMATIC 300(1)
    - SIMATIC 300(2)
    - SIMATIC 400(2)
      - CPU 414-3 PN/DP
    - (Unspecified)
    - All broadcast stations
    - All multicast stations
- In unknown project

Project: S7_Book
Station: SIMATIC 400(2)
Module: CPU 414-3 PN/DP

Connection
Type: S7 connection
☑ Display properties before inserting

Local Connection End Point
☐ Configured dynamic connection
☐ Configured at one end
☑ Establish an active connection
☐ Send operating mode messages

Block Parameters
Local ID (Hex): 3
W#16#3 ID

Connection Path

	Local	Partner
End Point:	SIMATIC 400(1)/ CPU 412-2 PN	SIMATIC 400(2)/ CPU 414-3 PN/DP
Interface:	CPU 412-2 PN, PN-IO-2(R0/S2)	CPU 414-3 PN/DP, PN-IO-3(R0/S2)
Subnet:	PN/IE_1 [Industrial Ethernet]	PN/IE_1 [Industrial Ethernet]
Address:	192.168.2.16	192.168.2.21

Fig. 6.2 Example of the configuration of a communication connection

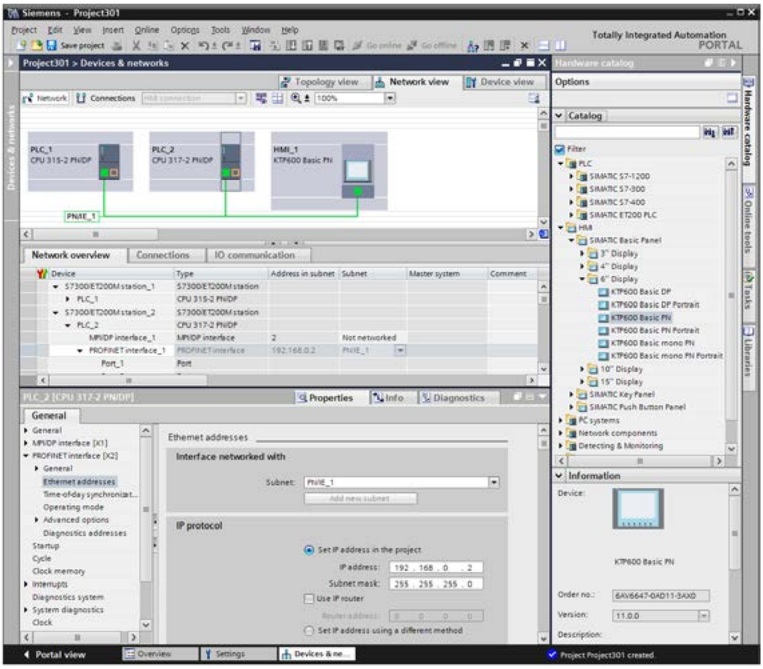
Network configuration with STEP 7 V11 inside TIA Portal

You can access the network configuration with the project opened in the Portal view via *Devices & networks* and *Configure networks* or in the Project view with the *Devices & networks* editor, which is positioned in the project navigation underneath the project. In the working window of the device configuration, change to the *Network view* tab (Fig. 6.3).

In the top part of the working window, the network view graphically displays all PLC, PC, and HMI stations available in the project. If the *Network* button is activated in the toolbar of the work window, the network is shown if it has already been set up during device configuration. You can drag further stations from the hardware catalog to the working area to add them to the project. You select a station via the CPU, to which you add a CP module from the hardware catalog if needed.

To add a subnet, select the bus interface in the station graphic and select *Add Subnet* from the shortcut menu. Alternatively, drag a bus interface to another of the same kind. This creates an appropriate subnet automatically.

When networking a module, the editor automatically claims the next unused node address for the bus interface. To display the node addresses, click in the toolbar of the working window on the *Display Addresses* icon. You can change the automati-



Device	Type	Address in subnet	Subnet	Master system	Comment
S7300/ET200M station_1	S7300/ET200M station				
PLC_1	CPU 315-2 PN/DP				
S7300/ET200M station_2	S7300/ET200M station				
PLC_2	CPU 317-2 PN/DP				
MPI/DP interface_1	MPI/DP interface	2	Not networked		
PROFINET interface_1	PROFINET interface	192.168.0.2	PN/IE_1		
Port_1	Port				

PLC_2 [CPU 317-2 PN/DP]

General

- General
- MPI/DP interface [X1]
- PROFINET interface [X2]
  - General
  - Ethernet addresses
  - Time-of-day synchronizat...
  - Operating mode
  - Advanced options
  - Diagnostics addresses
- Startup
- Cycle
- Clock memory
- Interrupts
- Diagnostics system
- System diagnostics
- Clock

Ethernet addresses

Interface networked with

Subnet: PN/IE_1

Add new subnet

IP protocol

● Set IP address in the project

IP address: 192 . 168 . 0 . 2

Subnet mask: 255 . 255 . 255 . 0

□ Use IP router

Router address: 0 . 0 . 0 . 0

○ Set IP address using a different method

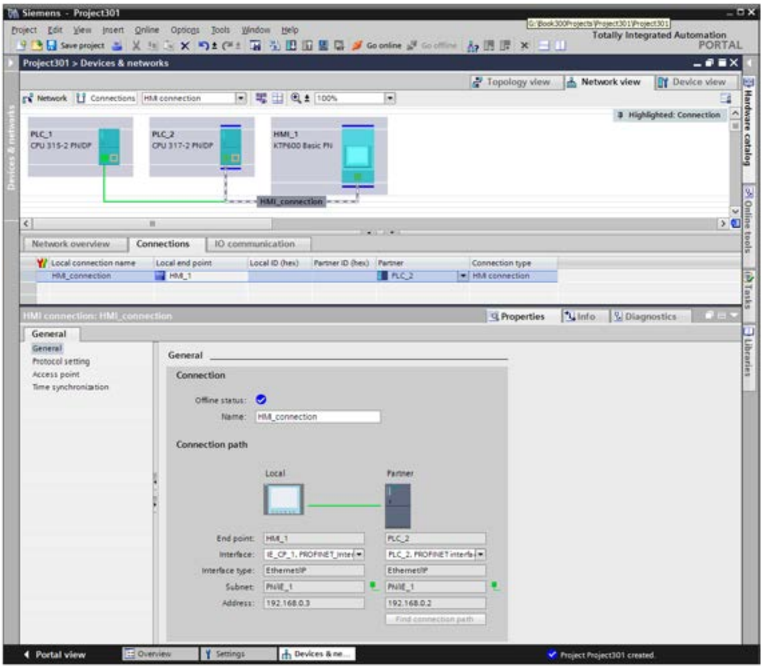
Device: KTP600 Basic PN

Order no.: 6AV6647-0AD11-3AX0

Version: 11.0.0

Description:

Fig. 6.3 Example of the network configuration working area with STEP 7 inside TIA Portal



Local connection name	Local end point	Local ID (hex)	Partner ID (hex)	Partner	Connection type
HMI_connection	HMI_1			PLC_2	HMI connection

HMI connection: HMI_connection

General

- General
- Protocol setting
- Access point
- Time synchronization

General

Connection

Offline status: ✓

Name: HMI_connection

Connection path

	Local	Partner
End point:	HMI_1	PLC_2
Interface:	IE_CP_1, PROFINET_inter	PLC_2, PROFINET interfa
Interface type:	Ethernet/IP	Ethernet/IP
Subnet:	PN/IE_1	PN/IE_1
Address:	192.168.0.3	192.168.0.2

Find connection path

Fig. 6.4 Display of connection properties for STEP 7 inside TIA Portal

cally assigned node address in the module properties in the inspector window with the bus interface selected.

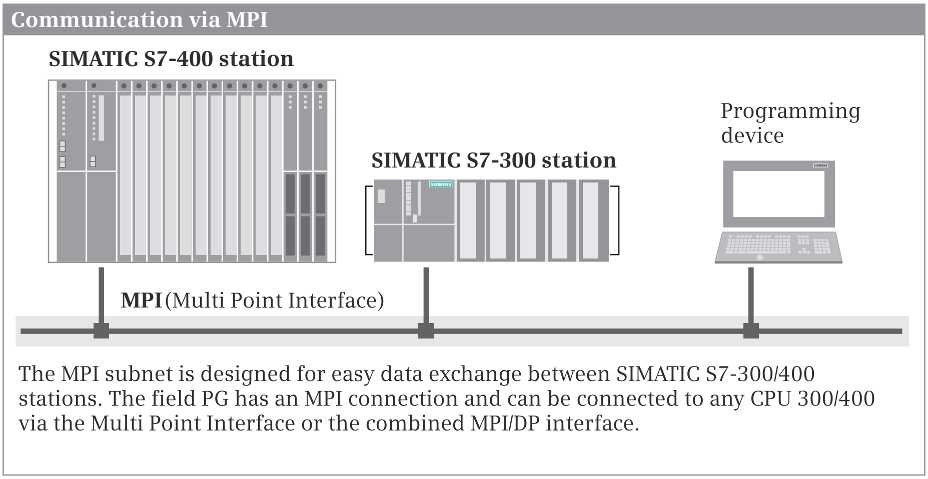
In order to configure a connection, click on the *Connections* button and select the connection type in the adjacent list. The devices suitable for this connection type are then displayed highlighted in the Network view. When you drag a highlighted station onto another, a corresponding connection is established. This is shown with a blue-white-patterned line and listed in the *Connections* tab in the lower part of the working window. If you select a connection, the connection properties are shown in the *Properties* tab in the inspector window and can be changed (Fig. 6.4).

Save the network configuration with *Project* > *Save* from the main menu. To compile, choose e.g. the PLC station in the project navigation and select *Compile* > *Hardware*from the shortcut menu.

6.2 The MPI subnet

Every CPU 300/400 is equipped with a “Multi Point Interface” (MPI). Data is exchanged by means of a proprietary Siemens protocol (Fig. 6.5).

MPI uses either a shielded two-wire cable or a fiber-optic conductor of glass or plastic as the transmission medium. The maximum cable length in a bus segment is 50 m for non-isolated interfaces and 1000 m for isolated interfaces. These lengths can be increased by using intermediary RS485 repeaters (up to 1100 m) or optical link modules (up to > 100 km). The transmission rate is generally 187.5 Kbit/s.



The MPI subnet is designed for easy data exchange between SIMATIC S7-300/400 stations. The field PG has an MPI connection and can be connected to any CPU 300/400 via the Multi Point Interface or the combined MPI/DP interface.

Fig. 6.5 Networking with MPI

The maximum number of stations is 32. Each station receives a certain amount of time to access the bus and send data telegrams. When this time expires, the station passes the token (access rights) to the next station. This procedure is called token passing.

Via an MPI subnet, you can use global data communications, external station S7 basic communication, or S7 communication to exchange data between CPUs (Table 6.1). No additional modules are required.

Table 6.1 Communication services via MPI

Communication service	Modules	Configuration, interface
Global data communications	every CPU 300 and CPU 400	GD table
External station S7 basic communication	every CPU 300 and CPU 400	SFC calls
S7 communication	every CPU 300 and CPU 400	Connection table, FB/SFB calls

To configure the networking, proceed as described in chapter 6.1 "Configuring the network" on page 205. If the CPU has a combined MPI/DP interface, the interface must be set to MPI in the CPU properties before connecting to the MPI subnet.

6.3 External station S7 basic communication

External station S7 basic communication (“MPI communication”) handles the data traffic between SIMATIC-300/400 stations. The stations must be interconnected over an MPI subnet, either using an MPI cable or – with a S7-400 station – via the backplane bus between the CPUs in the case of multiprocessor operation. The communication functions required for this are system functions in the CPU operating system, the calls for which are contained in STEP 7 V5.5 in the *Standard Library* under *Communication Blocks* and in STEP 7 inside TIA Portal in the Program Elements catalog under *Communication > Communication with iSlave/iDevice* (Table 6.2).

The communication functions establish communication connections as needed at runtime themselves; they are not configured in the connection table (“communi-

Table 6.2 System functions for external station S7 basic communication

Function		Comment
X_SEND	Send data	The communication functions must be called in pairs in both connection partners for each connection (two-sided data traffic).
X_RCV	Receive data	
X_GET	Read data	One of the communication functions is called in one connection partner, while the CPU operating system takes over control of the data traffic in the other partner (one-sided data traffic).
X_PUT	Write data	
X_ABORT	Terminate connection	The connection is set up dynamically as needed and can be configured to disconnect after the transfer ends. X_ABORT terminates an existing connection.

cation via non-configured connections"). If the data transmission has ended, it can be determined by parameter assignment if the connection is closed again and thus the connection resource should be released or whether the connection will remain available for further transmissions. The connection can be forcibly terminated with X_ABORT.

There is always only one connection to a communication partner at any one time. The communication partner in the S7 basic communication is not defined statically. Once a job is completed, re-parameterization can allow a different communication partner to be chosen. This means the number of communication partners is not limited by the available number of connection resources.

A maximum of 76 bytes are transferred as user data in S7 basic communication. The CPU's operating system combines the user data into blocks independent of the transfer direction, and these blocks are data-consistent. The length of the data transmitted consistently is a CPU-specific variable. If two CPUs exchange data via the connection configured at one end, the block size of the "passive" CPU is decisive for the consistency of the transferred data.

With signal state "1" at the parameter CONT, a connection to the partner CPU is established as a prerequisite for data transmission. If the CONT parameter is still "1" after the job is completed, the connection remains active, otherwise it is terminated.

The system function X_SEND sends data over a connection configured at both ends. The data is received in the partner device with the system function X_REC. The partner device is defined with the MPI address at parameter DEST_ID. The job is initiated if the signal state is "1" at the parameter REQ. The data addressed at the parameter SD is transferred. Parameter REQ_ID makes it possible to give the transmit data an identifier that can be evaluated at parameter REQ_ID by X_REC. As long as BUSY has signal state "1", the send request is still being processed.

The system function X_RCV receives data sent by X_SEND. EN_DT = "0" determines whether data has been received; NDA then is "1". If EN_DT = "1", received data is written to the target area defined by the parameter RD. The RET_VAL parameter indicates the number of received bytes.

If REQ = "1", the system function X_GET reads data from the partner station whose MPI address is specified at parameter DEST_ID. The data to be read is specified in parameter VAR_ADDR and written to the target area, which is defined at parameter RD. The RET_VAL parameter indicates the number of received bytes. As long as BUSY has signal state "1", the job is still being processed.

If REQ = "1", the system function X_PUT writes data to the partner station whose MPI address is specified at parameter DEST_ID. The send area is defined by the parameter SD. The parameter VAR_ADDR defines the target area in the partner device. As long as BUSY has signal state "1", the job is still being processed.

The system function X_ABORT terminates an existing connection with REQ = "1". Only a connection established in its own station with X_SEND, X_GET, or X_PUT can be canceled.

6.4 Global data communications

Global data communications (GD communications) is a communication service that is integrated in the operating system of a CPU 300/400. It exchanges small amounts of non-time-critical data via the MPI bus. The CPUs are interconnected using an MPI cable or – with a S7-400 station – via the backplane bus between the CPUs in the case of multiprocessor operation.

Cyclic global data communications does not require a user program. The CPU operating system contains the required communication functions. In a CPU 400, global data communications with system functions can also be event-driven in the user program (Table 6.3).

Table 6.3 System functions for global data communications with S7-400

Function		Comment
GD_SEND	Send GD packet	The system functions send or receive a GD packet. They can be called in addition to cyclic global data communications.
GD_RCV	Receive GD packet	

Sending and receiving takes place asynchronously between sender and receiver at the cycle control point, i.e. after cyclic program processing, before a new program cycle begins. Data is exchanged in the form of data packets. The CPUs exchanging a shared data packet form a GD circle. In a GD circle, a maximum of 15 CPUs can exchange data with each other. A CPU can also belong to several GD circles (Fig. 6.6).

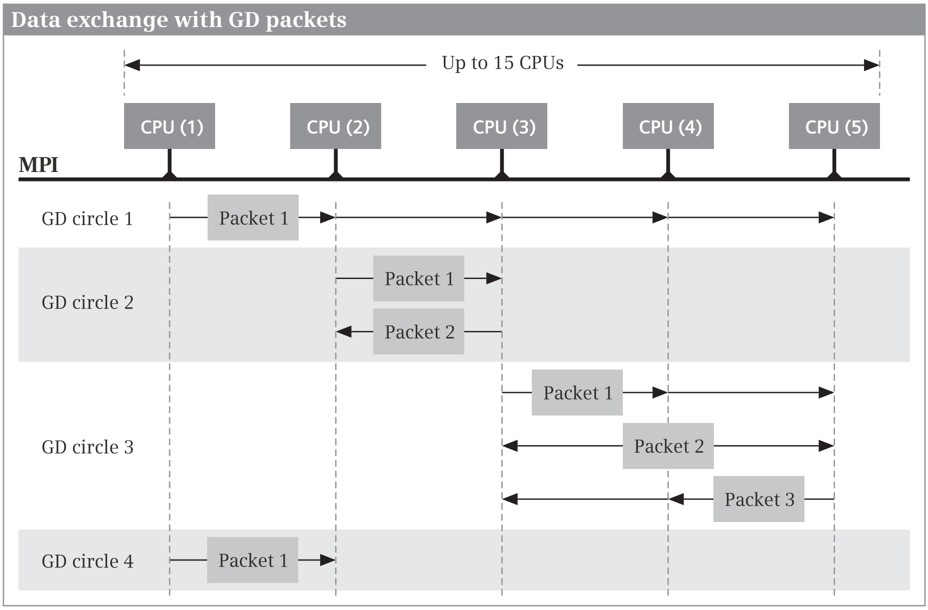


Fig. 6.6 Example of GD circles in global data communications

Global data communications is configured with STEP 7 V5.5. Select the icon for the MPI subnet in SIMATIC Manager or in the network configuration, choose *Options* > *Define Global Data*; an empty global data table will appear. In the column headings, enter the participating CPUs and define the data packets to be transferred line by line. A line can contain several receivers, but only one sender. The transferrable global data comprises inputs and outputs (process images), bit memories and data in data blocks, as well as time and counter values as data to be sent. A GD packet can consist of several GD elements that can be individual addresses or address areas.

After completing this, select *GD Table* > *Compile*. After the first compilation, you can specify the reduction ratios and, if required, the addresses for the transfer status. With the reduction ratios you specify the number of (user program) cycles after which the CPU sends or receives the data. A reduction ratio of 0 deactivates the cyclic data exchange in a a CPU 400 if you just want event-driven sending and receiving with system functions. Finally, you compile the GD table again.

With a CPU 400, you can also control global data communications from the user program. Additionally or alternatively to the cyclic transmission of global data, you can send a GD packet with the system function GD_SND and receive a GD packet with the system function GD_RCV. A prerequisite for these system functions is a configured global data table. The system functions do not have to be called in pairs, a “mixed” mode is also possible. For example, you can send event-driven GD packets with GD_SND, but then receive these cyclically.

6.5 The Industrial Ethernet subnet

Industrial Ethernet is the subnet for connecting computers and programmable controllers, with a focus on the industrial area, defined by the international standard IEEE 802.3. Any CPU with a PN interface and any PC with an Ethernet adapter can be connected to the subnet (Fig. 6.7).

Industrial Ethernet can be designed physically as an electrical, optical, or wireless network. FastConnect Twisted Pair (FC TP) cables with RJ45 connections or Industrial Twisted Pairs (ITP) cables with sub-D connections are available for implementing the electrical cabling. Fiber-optic (FO) cabling can consist of glass fiber, PCF, or POF. It offers galvanic isolation, is impervious to electromagnetic influences, and is suitable for long distances. Wireless transmission uses the frequencies 2.4 GHz and 5 GHz with data transfer rates up to 54 Mbit/s (depending on the national approvals).

Up to 1024 nodes can be networked per segment. Before a network access, each node checks if another node is presently sending data. If so, it waits a random time before attempting a new network access (CSMA/CD access method). All nodes have equal rights.

Using Industrial Ethernet, you can exchange data and use the S7 functions with the S7 communication. With the corresponding CP modules, you can also set up

ISO transport, ISO-on-TCP, TCP, UDP, and email connections (Table 6.4). To configure the networking and the connections, proceed as described in chapter 6.1 "Configuring the network" on page 205.

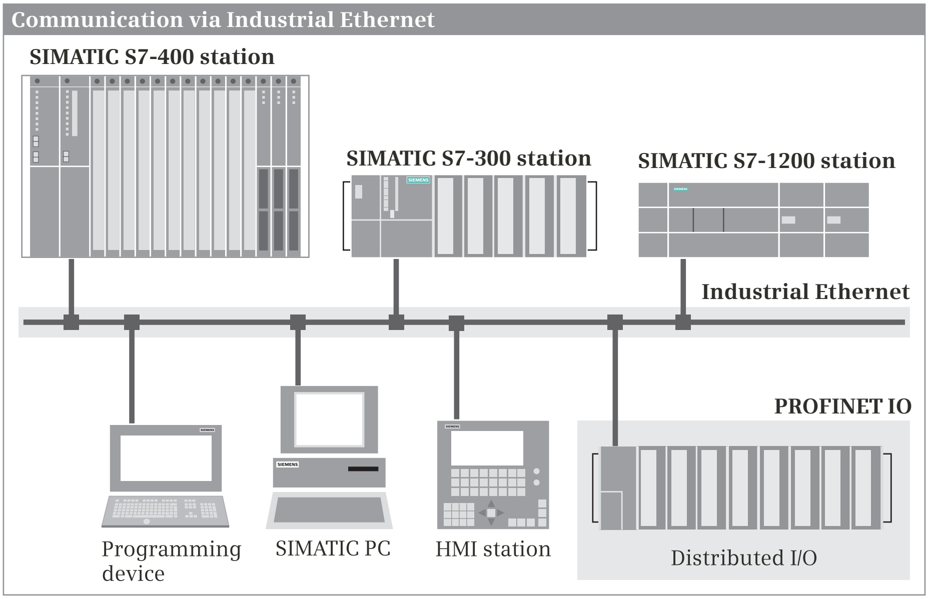


Fig. 6.7 Networking with Industrial Ethernet

Table 6.4 Communication services via Industrial Ethernet

Communication service	Modules	Configuration, interface
S7 communication	CPUs with PN interface CP 343-1, CP 343-1 Lean, CP 343-1 Advanced CP 443-1, CP 443-1 Advanced	Connection table, FB/SFB calls
Open User Communication (IE communication)	CPUs with PN interface CP 343-1, CP 343-1 Lean, CP 343-1 Advanced CP 443-1, CP 443-1 Advanced	FB calls
PROFINET IO (IO controller)	CPUs with PN interface CP 343-1, CP 343-1 Advanced CP 443-1, CP 443-1 Advanced	Hardware configuration, inputs/outputs, SFC/SFB calls
PROFINET IO (IO device)	CPU 300, CPU 400 and ET 200 CPU with PN interface CP 343-1, CP 343-1 Lean, CP 343-1 Advanced	Hardware configuration, inputs/outputs, SFC/SFB calls
S5-compatible communication	CP 343-1, CP 343-1 Lean CP 443-1	Connection table, SEND/RECEIVE calls
IT communication (HTTP, FTP, e-mail)	CP 343-1 Advanced CP 443-1 Advanced	Connection table, SEND/RECEIVE calls

As an open standard of the PROFIBUS user organization, PROFINET IO controls distributed I/O in Industrial Ethernet. The configuration of the distributed I/O with PROFINET IO is described in chapter 6.12 "Distributed I/O with PROFINET IO" on page 225.

6.6 Open User Communication, IE communication

Open User Communication (“open communication via Industrial Ethernet”, in short: IE communication) transfers data between two devices connected to the Ethernet subnet. Communication can be implemented using the TCP native protocol in accordance with RFC 793, the ISO-on-TCP protocol in accordance with RFC 1006, or the UDP protocol in accordance with RFC 768.

The communication functions are loadable standard function blocks (FB), which are contained in STEP 7 V5.5 in the *Standard Library* under *Communication Blocks* and in STEP 7 inside TIA Portal in the Program Elements catalog under *Communication* > *Open User Communication* (Table 6.5).

Table 6.5 Standard blocks for Open User Communication

Function		Comment
TSEND_C	Establish connection and send data with TCP	CPU 1200: These communication functions establish a connection for data transfer, and after the end of transmission they can be programmed to disconnect it. The connection is established in the “active” station.
TRCV_C	Establish connection and receive data with TCP	
TCON	Establish connection	CPU 300, CPU 400, and CPU 1200: To send and receive, a connection must be established with TCON in the “active” station. This can be disconnected with TDISCON. TSEND and TRCV transfer data using the TCP native and ISO-on-TCP protocols. TUSEND and TURCV transfer data using the UDP protocol.
TDISCON	Disconnect	
TSEND	Send data with TCP	
TRCV	Receive data with TCP	
TUSEND	Send data with UDP	
TURCV	Receive data with UDP	

Before data can be transferred by means of IE communication, a connection must be established to the communication partner in the case of the TCP native and ISO-on-TCP protocols (“connection-oriented protocols”) or a connection to the communication layer of the CPU operating system in the case of the UDP protocol (“connectionless protocol”). The partner is then addressed when the relevant function block is called.

The connection is configured via a data area (and not using the network configuration via the connection table). The necessary data structures for a CPU 300/400 are stored in the user data types and for a CPU 1200 in the system data types (Table 6.6). Data can be transferred in parallel in both directions over a connection. Several connections can exist on one physical line.

Table 6.6 Data types for the structure of the connection data and address information

Data structure of the	Data type	CPU
Connection data	TCON_Param	CPU 1200
	TCON_PAR	CPU 300/400
Address information of the remote partner with UDP	TADDR_Param	CPU 1200
	TADD_PAR	CPU 300/400

For a CPU 1200, TSEND_C and TRCV_C transfer data using the TCP native or ISO-on-TCP protocol. The parameters REQ and EN_R control the data transfer. The DONE, BUSY, ERROR and STATUS parameters indicate the job status. The CONT parameter controls the connection establishment and disconnection. With signal state “1”, the connection is established and the data can be transferred. If CONT is still “1” when the job is completed, the connection remains active, otherwise it is terminated. A pointer to the connection data is created at the CONNECT parameter. The DATA parameter defines the send or receive mailbox for the transferred data.

A connection for the communication functions TSEND and TRCV or TUSEND and TURCV is established by means of TCON. TDISCON cancels the connection again and thus releases the resources used. The REQ parameter triggers the establishment or disconnection of a connection. The DONE, BUSY, ERROR and STATUS parameters indicate the job status. A pointer to the connection data is created at the CONNECT parameter.

TSEND and TRCV transfer data using the TCP or ISO-on-TCP protocol. A connection must previously have been established with TCON. The parameters REQ and EN_R control the data transfer. The DONE, BUSY, NDR, ERROR, and STATUS parameters indicate the job status. The DATA parameter defines the send or receive mailbox for the transferred data.

TUSEND and TURCV transfer the data using the UDP protocol. A connection must previously have been established with TCON. The parameters REQ and EN_R control the data transfer. The DONE, BUSY, NDR, ERROR, and STATUS parameters indicate the job status. The DATA parameter defines the send or receive mailbox for the transferred data. A pointer at the ADDR parameter points to the address of the sender or receiver.

6.7 S7 communication

S7 communication transmits large amounts of data between SIMATIC S7 stations. The stations are interconnected via a subnet. This can be an MPI, PROFIBUS or Ethernet subnet. The communication connections are static and are configured in the connection table (“communication via configured connections”).

In a CPU 300, the S7 communication is implemented with loadable standard function blocks (FB), which are contained in STEP 7 V5.5 in the *Standard Library* under

Communication Blocks and in STEP 7 inside TIA Portal in the Program Elements catalog under *Communication* > *S7 Communication*.

In a CPU 400, the S7 communication is implemented with system function blocks (SFB) integrated in the operating system. They are contained in STEP 7 V5.5 in the *Standard Library* under *System Function Blocks* and in STEP 7 inside TIA Portal in the Program Elements catalog under *Communication* > *S7 Communication*.

With a CPU 1200, the S7 communication is implemented with system blocks integrated in the operating system. They are included in the Program Elements catalog under *Communication* > *S7 Communication*.

Configuring S7 communication

A prerequisite for S7 communication is the networking of PLC stations and the establishment of S7 connections with a configured connection table in which the communication connections are defined (see chapter 6.1 "Configuring the network" on page 205).

A communication connection is specified by a connection ID for every communication partner. STEP 7 assigns the connection ID when editing the connection table. You use the “Local ID” for parameterization of the communication functions in the CPU from which the connection is considered and the “Partner ID” for parameterization of the communication functions in the (remote) partner CPU. It is possible to use the same logical connection for different send and receive jobs.

Two-sided data communication

In the case of two-sided data communication, you require a send block and a receive block at each end of a connection. Both blocks have the connection IDs located in the same line in the connection table. You can also use several “pairs of blocks” which are then distinguished by the job ID (Table 6.7).

USEND and URCV transfer a data packet without acknowledgment by the partner CPU. BSEND and BRCV transfer a data block in segments of up to 64 KB. Data traffic is controlled by the parameters REQ and EN_R. The DONE, NDR, ERROR, and STATUS parameters indicate the job status. At the parameter ID is the connection

Table 6.7 Standard blocks for S7 communication (two-sided data communication)

Function		Comment
USEND	Uncoordinated sending of a data packet	CPU 300 and CPU 400: These communication functions are used in pairs. For the CPU 300, there are several types of functions.
URCV	Uncoordinated receipt of a data packet	
BSEND	Send a data block with a length of up to 64 KB	CPU 300 and CPU 400: These communication functions are used in pairs. The data block is transferred in segments. For the CPU 300, there are several types of functions.
BRCV	Receive a data block with a length of up to 64 KB	

ID from the connection table; at parameter R_ID is a job identifier. The data to be transmitted is taken from the send mailbox to which the parameter SD_n points, and saved in the receive mailbox to which the parameter RD_n points. With a CPU 300, only one of the parameters (SD_1, RD_1) can be used; with a CPU 400, all four can be used.

One-sided data traffic

In the case of one-sided data exchange, the call of the communication function is only present in the local CPU. In the partner CPU, the operating system takes care of the necessary communication tasks; a send or receive (user) program is not required here. The partner CPU can perform the required communication services in both RUN and STOP mode. The size of the consistently transmitted data blocks depends on the (server) CPU used (Table 6.8).

Table 6.8 Standard blocks for S7 communication (one-sided data traffic)

Function		Comment
GET	Read data from a partner CPU	CPU 300, CPU 400, and CPU 1200: These communication functions are used in only one CPU; in the partner CPU, the operating system takes over the data traffic. For the CPU 300, there are several types of functions.
PUT	Write data to a partner CPU	

The REQ parameter initiates data traffic. The DONE, NDR, ERROR and STATUS parameters indicate the job status. In the local CPU, the parameter SD_n points to the send mailbox and the parameter RD_n points to the receive mailbox. The parameter ADDR_n indicates the data area where the operating system in the partner CPU should read or write. With a CPU 300, only one of the parameters (SD_1, RD_1, ADDR_1) can be used; with a CPU 400 or CPU 1200, all four can be used.

Table 6.9 Standard blocks for S7 communication (control and monitoring)

Function		Present with
STATUS	Poll status of the partner device. The status is requested from the partner device.	CPU 400
USTATUS	The status of the partner device is received. The status is sent by the partner device on a change without being requested.	CPU 400
CONTROL	Poll status of a communication instance.	CPU 400
C_CNTRL	Poll status of a connection.	CPU 300
C_DIAG	Determine connection status.	CPU 400
START	Perform a restart on the partner device.	CPU 400
STOP	Switch the partner device to STOP status.	CPU 400
RESUME	Perform a hot restart on the partner device.	CPU 400
PRINT	Transfer data to a printer connected via a CP module.	CPU 400

Control and monitoring functions

Control and monitoring functions are also part of one-sided data traffic. No user program is required in the partner CPU for the execution of control and monitoring functions (Table 6.9). The parameter REQ starts the job; the parameters DONE, ERROR, and STATUS indicate the job status.

6.8 The PROFIBUS subnet

PROFIBUS is the acronym for “Process Field Bus”. It is a vendor-independent standard according to EN 50170 Volume 2 for the networking of field devices (Fig. 6.8).

The transmission medium used is either a shielded two-wire line or a fiber-optic cable of glass or plastic. The cable length in a bus segment depends on the data transfer rate; this is 100 m for the highest transfer rate (12 Mbit/s) and 1000 m for the lowest (9.6 Kbit/s). The network span can be increased using repeaters or optical link modules.

The maximum number of nodes is 127; nodes can be active or passive. An active node can access the bus for a specific period and can send data frames. After this period, it passes the access permission on to the next active node (“token passing” access method). If passive nodes (slaves) are allocated to an active node (master), the master will perform the data exchange with the slaves assigned to it while it has access permission. A passive node has no access authorization.

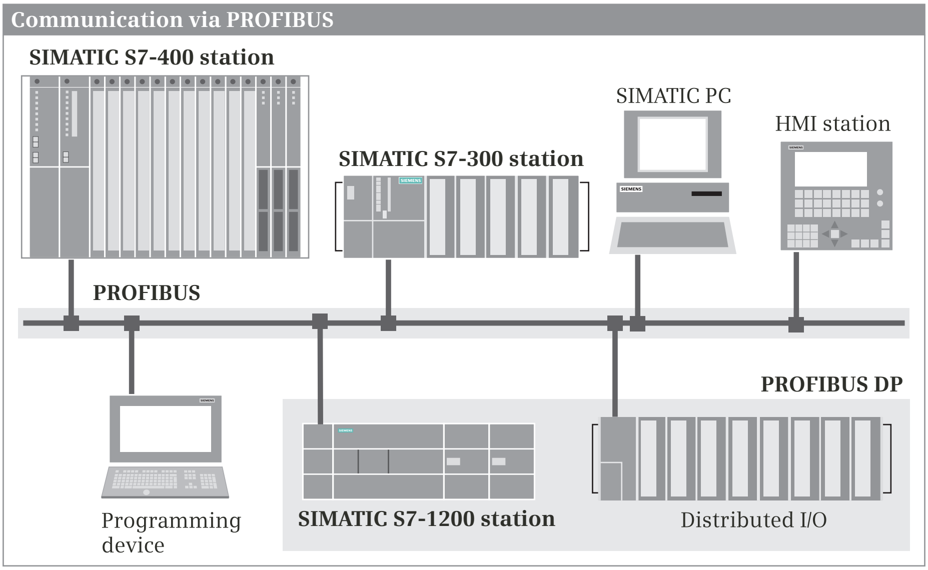


Fig. 6.8 Networking with PROFIBUS

Table 6.10 Communication services via PROFIBUS

Communication service	Modules	Configuration, interface
S7 communication	CPUs with DP interface CP 342-5, CP 343-5 CP 443-5 Basic, CP 443-5 Extended	Connection table, FB/SFB calls
Internal station S7 basic communication	CPUs with DP interface CP 342-5, CP 343-5 CP 443-5 Basic, CP 443-5 Extended IM 467	SFC calls
PROFIBUS DP (DP master)	CPUs with DP interface CM 1243-5 CP 342-5 (DPV0) CP 443-5 Extended IM 467	Hardware configuration, inputs/outputs, SFC/SFB calls
PROFIBUS DP (DP slave)	CPUs with DP interface CM 1242-5 CP 342-5 (DPV0) CP 443-5 Extended	Hardware configuration, inputs/outputs, SFC/SFB calls
S5-compatible communication	CP 342-5, CP 343-5 CP 443-5 Basic, CP 443-5 Extended	Connection table, SEND/RECEIVE calls
PROFIBUS FMS	CP 343-5 CP 443-5 Basic	Connection table, FMS interface

With corresponding CP modules, you can transfer data with PROFIBUS FMS. There are loadable communication functions to serve as an interface to the user program (FMS interface or SEND/RECEIVE interface). An overview of the modules with PROFIBUS connection and the available communication services is shown in Table 6.10.

To configure the networking and the connections, proceed as described in chapter 6.1 "Configuring the network" on page 205. You connect distributed I/O via a PROFIBUS subnet (chapter 6.15 "Distributed I/O with PROFIBUS DP" on page 242).

6.9 Internal station S7 basic communication

The internal station S7 basic communication transfers data between programmable modules within a SIMATIC 300/400 station. Data is exchanged via PROFIBUS. Example: Internal station S7 basic communication can, for example, take place in parallel to the cyclic data exchange via PROFIBUS DP between the master CPU and the slave CPU where event-driven data is transferred. The communication functions required for this are system functions in the CPU operating system, the calls for which are contained in STEP 7 V5.5 in the *Standard Library* under *Communication Blocks* and in STEP 7 inside TIA Portal in the Program Elements catalog under *Communication* > *Communication with iSlave/iDevice* (Table 6.11).

Table 6.11 System functions for internal station S7 basic communication

Function		Comment
I_GET	Read data	The communication function is called in the user program of the local CPU. The CPU operating system takes over control of the data traffic in the connection partner.
I_PUT	Write data	
I_ABORT	Terminate connection	The connection is set up dynamically as needed and can be configured to disconnect after the transfer ends. I_ABORT terminates an existing connection.

The communication functions establish communication connections as needed at runtime themselves; they are not configured in the connection table ("communication via non-configured connections"). If the data transmission has ended, it can be determined by parameter assignment if the connection is closed again and thus the connection resource should be released or whether the connection will remain available for further transmissions. The connection can also be forcibly terminated with I_ABORT.

There is always only one connection to a communication partner at any one time. The communication partner in the S7 basic communication is not defined statically. Once a job is completed, re-parameterization can allow a different communication partner to be chosen. This means the number of communication partners is not limited by the available number of connection resources.

A maximum of 76 bytes are transferred as user data in S7 basic communication. The CPU's operating system combines the user data into blocks independent of the transfer direction, and these blocks are data-consistent. The length of the data transmitted consistently is a CPU-specific variable. If two CPUs exchange data via the connection configured at one end, the block size of the "passive" CPU is decisive for the consistency of the transferred data.

With signal state "1" at the parameter CONT, a connection to the partner CPU is established as a prerequisite for data transmission. If the CONT parameter is still "1" after the job is completed, the connection remains active, otherwise it is terminated.

On REQ = "1", the system function I_GET reads data from the module whose user data address is specified at parameters IOID and LADDR. The data to be read is specified in parameter VAR_ADDR and written to the target area, which is defined at parameter RD. The RET_VAL parameter indicates the number of received bytes. As long as BUSY has signal state "1", the job is still being processed.

On REQ = "1", the system function I_PUT writes data to the partner station whose user data address is specified at parameters IOID and LADDR. The send area is defined by the parameter SD. The parameter VAR_ADDR defines the target area in the partner device. As long as BUSY has signal state "1", the job is still being processed.

The system function I_ABORT terminates an existing connection with REQ = "1". Only a connection established in its own station with I_GET or I_PUT can be canceled.

6.10 The AS-Interface subnet

The AS-Interface (actuator/sensor interface, in short: AS-i) is an open, international standard in accordance with EN 50295 and IEC 62026-2 for the lowest process level in automation plants. An AS-Interface master controls the AS-Interface slaves via a 2-wire AS-Interface cable, which transfers both the control signals and the supply voltage (Fig. 6.9).

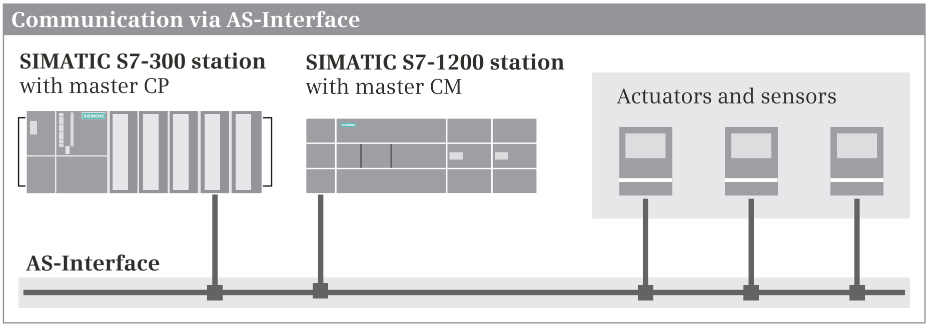


Fig. 6.9 Networking with AS-Interface

The transmission medium is an unshielded two-wire line which supplies the actuators and sensors with both data and power (power supply unit required). With an extension plug and two repeaters in parallel connection, the network span can be up to 600 m.

An AS-i master controls up to 62 AS-i slaves with cyclic polling and thus guarantees a response time of 10 ms when fully configured with A/B addresses (double address assignment), depending on the slave profile. The AS-i master is available, for example, as communication module in a SIMATIC station or as link

Table 6.12 Master modules for AS-Interface

Modules	Comment
CP 343-2	AS-i master in S7-300 design
CP 343-2P	AS-i master in S7-300 design, network configuration with STEP 7 inside TIA Portal
CM 1243-2	AS-i master V3.0 in S7-1200 design, network configuration with STEP 7 inside TIA Portal
IE/AS-i Link PN IO	Gateway; in PROFINET the link is an IO Device, while on the AS-i bus it is a single or double master
DP/AS-i Link 20 E	Gateway; in PROFIBUS the link is a modular DP slave, while on the AS-i bus it is a single master
DP/AS-i Link Advanced	Gateway; in PROFIBUS the link is a modular DP slave, while on the AS-i bus it is a single or double master

with a gateway from PROFINET IO and PROFIBUS DP. In STEP 7 V5.5, the AS-i master system is configured using a table. STEP 7 inside TIA Portal represents the AS-i bus as a subnet for the master modules CP 343-2P and CM 1243-2. (Table 6.12). For the communication modules CP 343-2 and CP 343-2P, there is the standard block AS-3422, which controls the AS-i master via command jobs.

6.11 The point-to-point connection

A point-to-point connection (PTP) allows data transfer over a serial connection (Fig. 6.10).

The transmission medium is an electrical cable with interface-dependent assignment. RS 232C (V.24), 20 mA (TTY) and RS 422/485 (X.27) are available as interfaces. The transmission rate ranges from 300 bit/s up to 19.2 Kbit/s on a 20 mA interface or 76.8 Kbit/s with RS 232C and RS 422/485. The cable length is dependent on the physical interface type and the transmission rate; it is 10 m for RS 232C, 1000 m for a 20 mA interface with 9.6 Kbit/s and 1200 m for RS 422/485 with 19.2 Kbit/s.

For each different interface, there is a different module type or plug-in interface module (Table 6.13). Protocols and procedures include 3964 (R), RK 512, printer drivers, and an ASCII driver that allows you to define your own procedure. For special cases there are loadable special drivers.

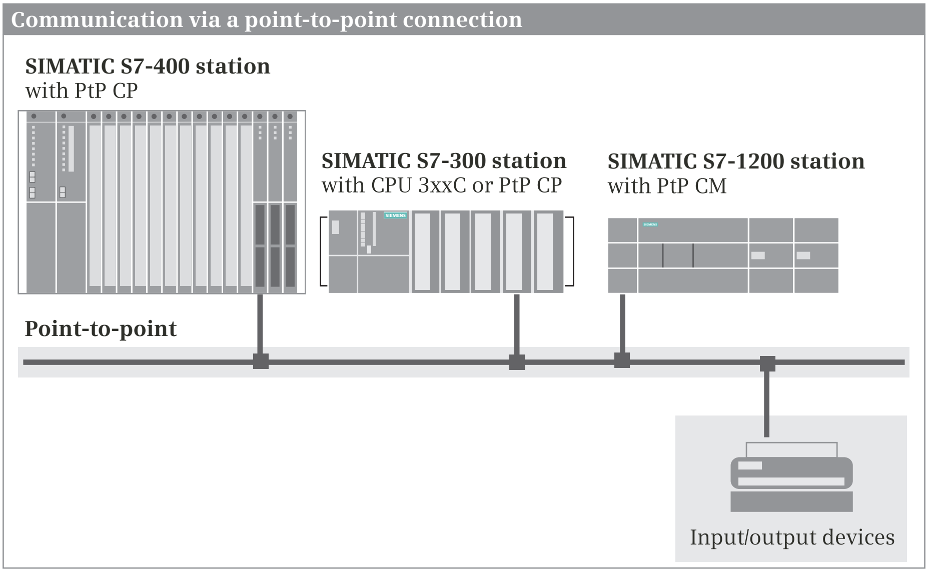


Fig. 6.10 Point-to-point networking

Table 6.13 Modules, interfaces, and drivers for the PtP subnet

Module	Interface				Drivers			
		V.24	TTY	X.27	ASCII	3964(R)	RK512	Other
CPU 313C-2PtP	1	–	–	×	×	×	–	–
CPU 314C-2PtP	1	–	–	×	×	×	×	–
CP 340	1	×	×	×	×	(×)	–	Printer driver
CP 341	1	×	×	×	×	×	×	Reloadable protocols
CP 440	1	–	–	×	×	×	–	–
CP 441-1	1	×	×	×	×	×	–	Printer driver
CP 441-2	2	×	×	×	×	×	×	Printer drivers, reloadable protocols
CM 1241	1	×	–	×	×	–	–	USS drive protocol, Modbus RTU, reloadable protocols
CB 1241	1	–	–	×	×	–	–	USS drive protocol, Modbus RTU, reloadable protocols

The interface properties are parameterized with the hardware configuration. Communication functions are available depending on the interface type and application.

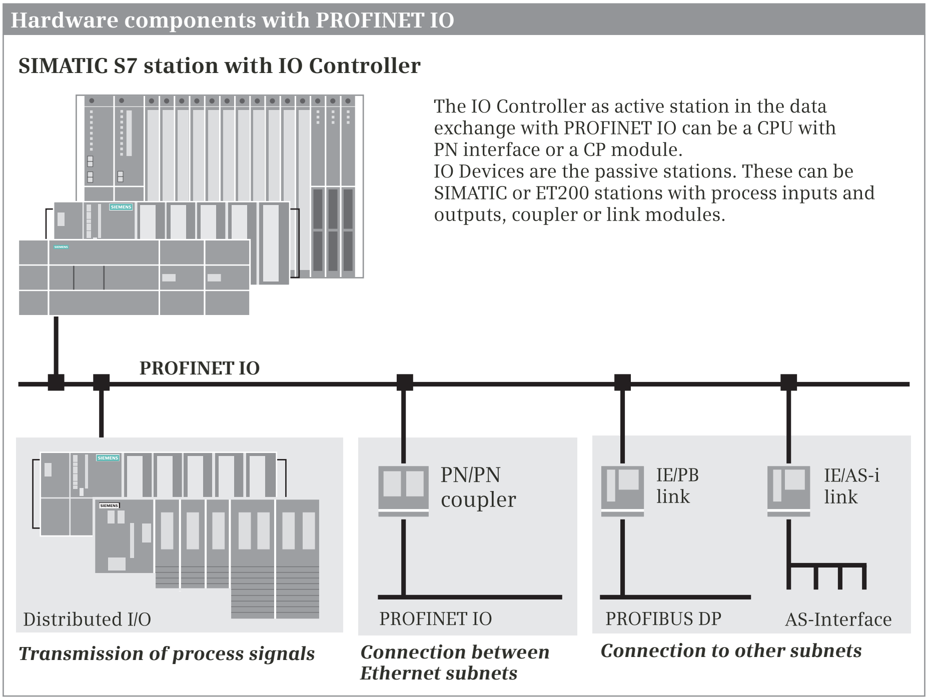
6.12 Distributed I/O with PROFINET IO

PROFINET (process field network) provides a standardized interface for transmission over Industrial Ethernet, standardized according to the manufacturer-independent standard IEC 61158/61784. Industrial Ethernet ("real-time Ethernet") is an area and cell network according to the international standards IEEE 802.3 (Ethernet) and IEEE 802.11 a/b/g/h/n (wireless LAN), designed for the industrial sector down to the field level.

With PROFINET, there are the types

- PROFINET CBA (component based automation), an automation concept for implementing decentralized controls, and
- PROFINET IO for communication on the fieldbus level between an IO Controller and the IO Device assigned to it.

PROFINET IO primarily transfers binary process data between an IO Controller in the (central) programmable controller and the distributed field devices – the IO Devices. (Fig. 6.11). The IO Controller and all IO Devices controlled by it constitute a PROFINET IO system. Several PROFINET IO systems can be operated in a Ethernet subnet.


Hardware components with PROFINET IO
SIMATIC S7 station with IO Controller
PROFINET IO
PN/PN coupler
IE/PB link
IE/AS-i link
Distributed I/O
PROFINET IO
PROFIBUS DP
AS-Interface
Transmission of process signals
Connection between Ethernet subnets
Connection to other subnets

The IO Controller as active station in the data exchange with PROFINET IO can be a CPU with PN interface or a CP module.
IO Devices are the passive stations. These can be SIMATIC or ET200 stations with process inputs and outputs, coupler or link modules.

Fig. 6.11 Components of a PROFINET IO system

With PROFINET IO, up to 256 devices – depending on the device – can be managed by one IO Controller. The number of nodes in the network is more or less unlimited. The data transmission rate is 10 or 100 Mbit/s.

General procedure for configuring PROFINET IO

A prerequisite for the configuration of a PROFINET IO system is a project with a PLC station. The PLC station contains a CPU with PN interface or a CP module with integrated IO Controller.

Connect the PN interface with an Ethernet subnet and assign a PROFINET IO system to the IO Controller. Select the IO Devices from the hardware catalog, drag them to the working area, and connect their PN interface with the PROFINET IO system.

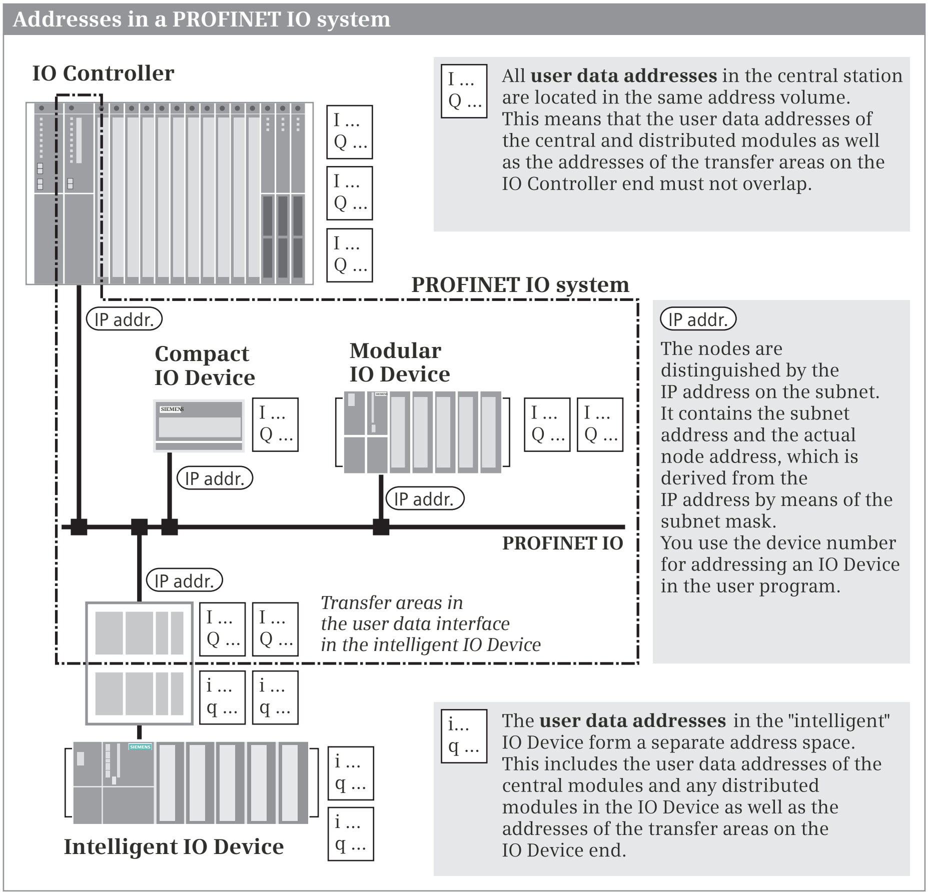
An “intelligent” IO Device (in short: “I-Device”), i.e. an independent station with a CPU, is coupled via a transfer area to the PROFINET IO system. To configure an I-Device, create an S7 or ET200 station with a CPU, set the operating mode of the PN interface to “IO Device”, connect the PN interface with the PROFINET IO system, and configure the transfer area as the user data interface to the IO Controller.

Addresses in the PROFINET IO system

A PROFINET IO system with the IO Controller and all IO Devices is integrated into the address structure of the central CPU, because the IO Devices are addressed by the central CPU as centrally arranged modules. The following addresses are available in a PROFINET IO system:

▷ IP address
Every node on the Industrial Ethernet subnet using the TCP/IP protocol requires a unique IP address that is assigned once for the IO Controller, and from which the IP addresses for the IO Devices can then be derived. The four-byte-long address consists of the subnet address and the node address. Their respective parts are specified by the subnet mask.

▷ Node address:
The node address is part of the IP address and is unique in the PROFINET IO system.



All **user data addresses** in the central station are located in the same address volume. This means that the user data addresses of the central and distributed modules as well as the addresses of the transfer areas on the IO Controller end must not overlap.

The nodes are distinguished by the IP address on the subnet. It contains the subnet address and the actual node address, which is derived from the IP address by means of the subnet mask. You use the device number for addressing an IO Device in the user program.

Transfer areas in the user data interface in the intelligent IO Device

The **user data addresses** in the "intelligent" IO Device form a separate address space. This includes the user data addresses of the central modules and any distributed modules in the IO Device as well as the addresses of the transfer areas on the IO Device end.

Fig. 6.12 Addresses in a PROFINET IO system

- Device name, device number
  Give the IO Controller and every IO Device a device name. In addition to the device name, each IO Device is given a device number (station number) that you can change and with which you address the IO Device from the user program.
- Geographic address:
  The geographic address of an IO Device corresponds to the slot address of a centrally located module. It is composed of the number of the PROFINET IO system, the station number, and the slot number.
- Logical address, module start address:
  You access the user data of an IO Device under the logical address. The smallest logical address corresponds to the module start address of a central module.
- Diagnostic address:
  Modules without user data that can send diagnostic data are addressed via a diagnostic address. The diagnostic address occupies one byte in the address area of the peripheral inputs.

A compact IO Device behaves like a single module, while a modular IO Device behaves like several modules. In an “intelligent” IO Device, the transfer areas of the user data interface simulate modules. The user data addresses of all modules of a station may not overlap (Fig. 6.12).

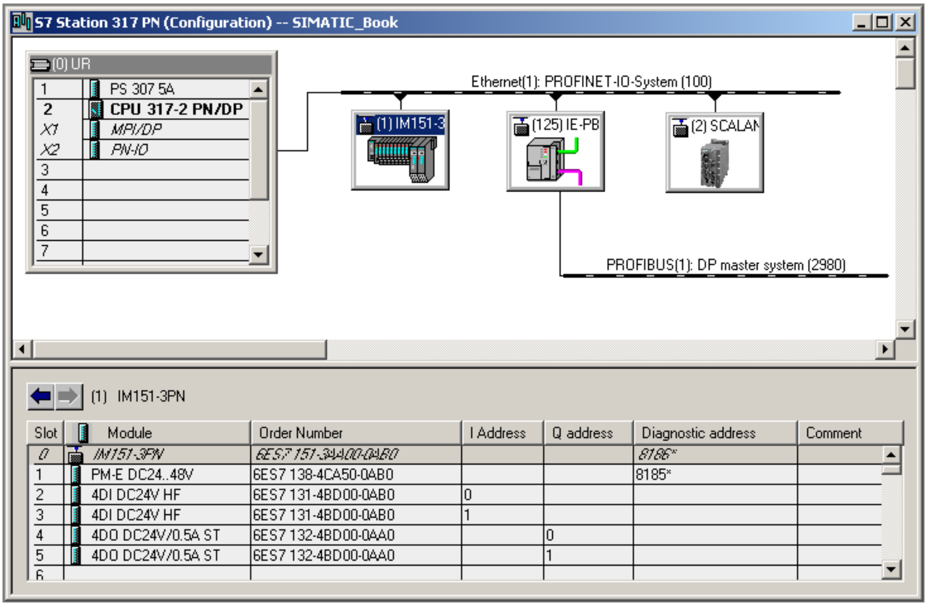
Configuring a PROFINET IO system with STEP 7 V5.5

Prerequisite: You have created a project with the central PLC station and placed a module with a PN interface. The IO Controller is activated in the properties of the PN interface.

You configure the properties of the PN interface by double-clicking in the configuration table with the station open on the line with the PN interface. Click the *Properties* button, and in the dialog box select the subnet to which the PN interface is to be connected. To create a PROFINET IO system, select the command *Insert > PROFINET IO system*.

To add an IO Device, drag the desired IO Device from the hardware catalog to the PROFINET IO system in the working area. The IO Devices can be found in the hardware catalog under PROFINET IO and the corresponding folder. Double-clicking on the IO Device opens the Properties dialog. Here, for a modular IO Device, you can insert additional modules that you drag from the hardware catalog under the interface module used (!) to the working area and set the user data addresses if necessary. An example of a PROFINET IO system is shown in Fig. 6.13.

You first create an “intelligent” IO Device as an autonomous PLC station using the SIMATIC Manager, insert a module with a PN interface, and connect it to the Ethernet subnet. To set the operating mode, select the PN interface in the configuration table and then choose *Edit > Object Properties*. Select the *I-Device* tab in the Properties window, and check the *I-Device mode* checkbox there.



S7 Station 317 PN (Configuration) -- SIMATIC_Book

(0) UR

1	PS 307 5A
2	**CPU 317-2 PN/DP**
X1	*MPI/DP*
X2	*PN-IO*
3	
4	
5	
6	
7	

Ethernet(1): PROFINET-IO-System (100)
(1) IM151-3
(125) IE-PB
(2) SCALAN
PROFIBUS(1): DP master system (2980)

(1) IM151-3PN

Slot	Module	Order Number	I Address	Q address	Diagnostic address	Comment
0	*IM151-3PN*	*6ES7 151-3AA00-0AB0*			*8186**	
1	PM-E DC24..48V	6ES7 138-4CA50-0AB0			8185*	
2	4DI DC24V HF	6ES7 131-4BD00-0AB0	0			
3	4DI DC24V HF	6ES7 131-4BD00-0AB0	1			
4	4DO DC24V/0.5A ST	6ES7 132-4BD00-0AA0		0		
5	4DO DC24V/0.5A ST	6ES7 132-4BD00-0AA0		1		
6						

Fig. 6.13 Example of a PROFINET IO system for STEP 7 V5.5

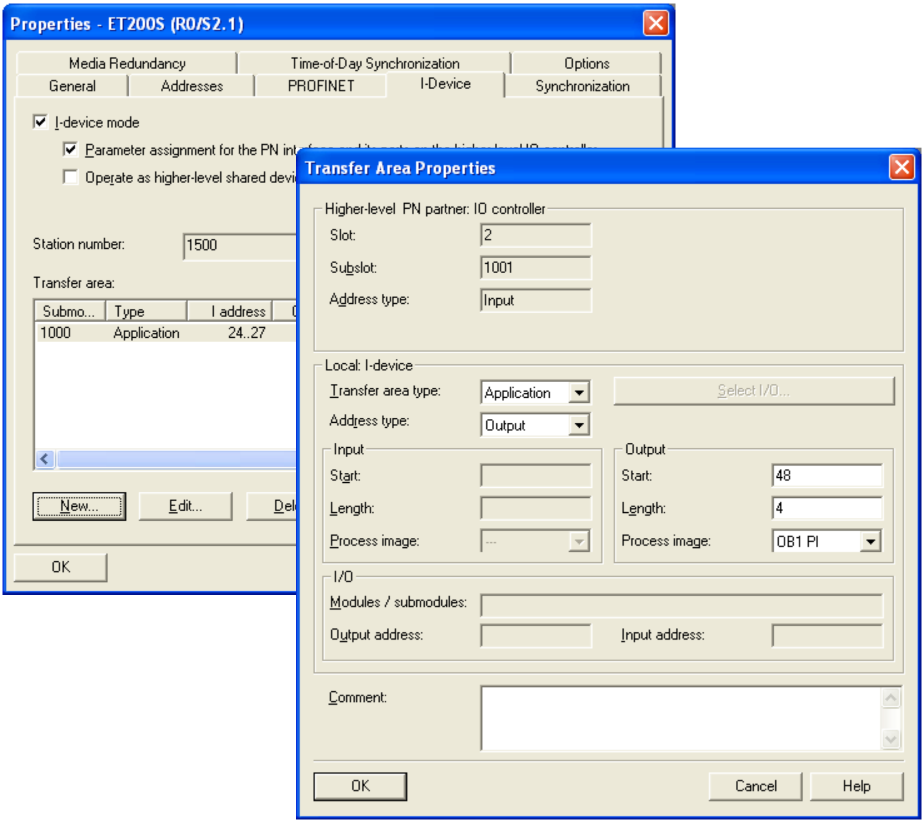
After activating the operating mode, click *New...* to create a new transfer area in the user data interface to the IO Controller. In the properties of the transfer area, specify the direction of transmission and the user data addresses from the viewpoint of the IO Device (Fig. 6.14).

Once you have configured all transfer areas, save and compile the station and generate a GSD file with *Options > Create GSD file for I-Device ...* from the I-Device. In the dialog window, click the *Create* button and for immediate installation the *Install* button, and then *Close*. STEP 7 creates a folder *Preconfigured Stations* in the hardware catalog under *PROFINET IO*, and inserts a folder with the icon for the I-Device that was just created.

To couple the I-Device with the IO Controller, open the PLC station with the IO Controller and drag the I-Device to the PROFINET IO system. With the IO Device selected, the transfer areas are displayed in the configuration table as a subslot of slot 2 with the user data address from the viewpoint of the IO Controller. To change an address, double-click on the address line and enter the new address on the *Addresses* tab in the Properties window.

Configuration of a PROFINET IO system with STEP 7 inside TIA Portal

Prerequisite: You have created a project with the central PLC station and inserted a module with a PN interface. The IO Controller is activated in the properties of the PN interface.



☑ I-device mode

☑ Parameter assignment for the PN int

□ Operate as higher-level shared devi

Station number: 1500

Transfer area:

Submo...	Type	I address	(
1000	Application	24..27	

Higher-level PN partner: IO controller

Slot: 2

Subslot: 1001

Address type: Input

Local: I-device

Transfer area type: Application

Address type: Output

Input

Start:

Length:

Process image: ---

Output

Start: 48

Length: 4

Process image: OB1 PI

I/O

Modules / submodules:

Output address:

Input address:

Comment:

Fig. 6.14 Example of a transfer area of an I-Device (STEP 7 V5.5)

Open the PLC station with HW Config and select the *Network view* tab in the working window. Connect the PN interface of the IO Controller to a subnet by right-clicking on the PN interface and selecting *Add subnet* from the shortcut menu. To configure a PROFINET IO system, choose *Assign IO system* from the shortcut menu.

To add an IO Device to the PROFINET IO system, drag the desired IO Device from the hardware catalog to the PROFINET IO system in the working area. With a selected IO Device, you can set the properties of the IO Device in the device view, for example, by adding desired modules and setting their addresses.

Fig. 6.15 shows a PROFINET IO system in network view with the central IO Controller and a compact IO Device (ET 200eco), a modular IO Device (ET 200 with IM 153-4PN), and an I-Device (ET 200S with IM 151-8PN/DP).

You create an I-Device as a separate PLC station. You establish a connection to the existing subnet by dragging the PN interface of the I-Device to the PN interface of another device on the subnet, for example, the PN interface of the IO Controller. In the interface properties of the IO Device, activate the *IO Device* checkbox in the

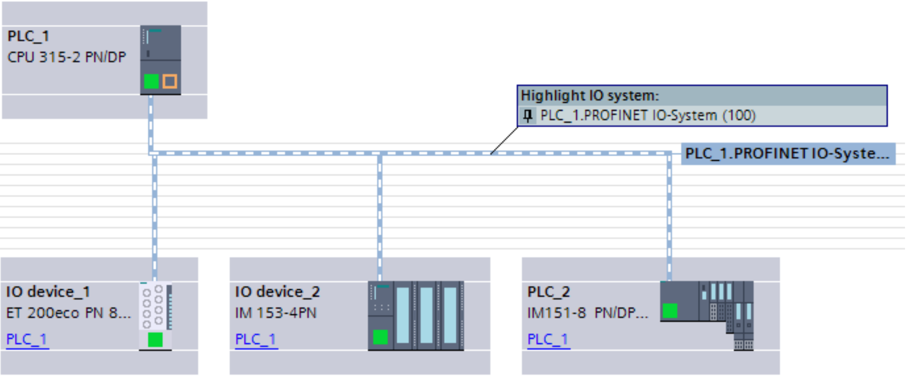
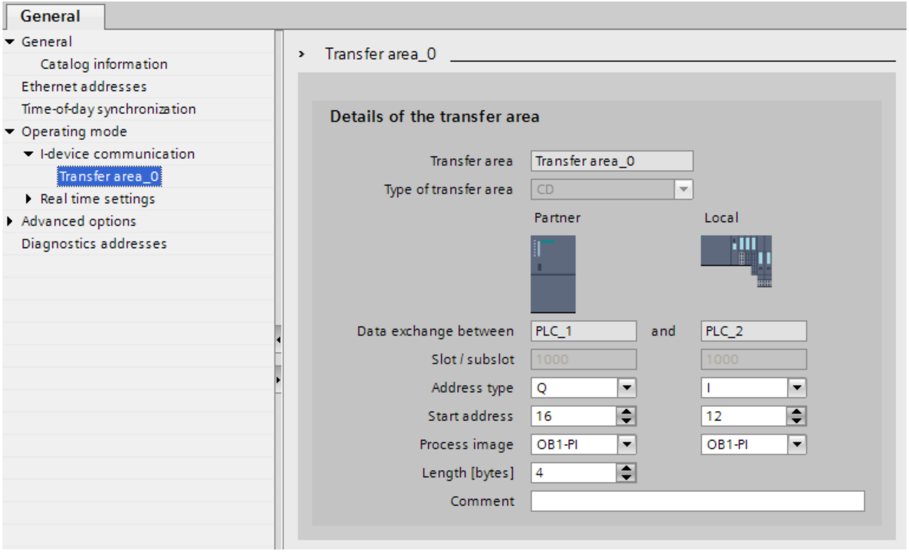


Fig. 6.15 Example of representation of a PROFINET IO system (STEP 7 inside TIA Portal)

Operating mode group and select the assigned IO Controller from the drop-down list. The station is then added as an IO Device to the PROFINET IO system.

You configure the user data interface to the IO Controller in the interface properties of the I-Device. In the *Operating mode* group, choose *I-Device communication* and double-click in the *Transfer areas* table on <*Add new*>. A new transfer area is created. Set the user data addresses here, both on the side of the I-Device as well as on the side of the IO Controller. You can create several transfer areas. You are given detailed information about a transfer area when you select it in the Properties navigation (Fig. 6.16).



Details of the transfer area

Transfer area	Transfer area_0		
Type of transfer area	CD		
	Partner		Local
Data exchange between	PLC_1	and	PLC_2
Slot / subslot	1000		1000
Address type	Q		I
Start address	16		12
Process image	OB1-PI		OB1-PI
Length [bytes]	4		
Comment			

Fig. 6.16 Example of the configuration of a transfer area (STEP 7 inside TIA Portal)

6.13 Special functions for PROFINET IO

Real-time communication with PROFINET IO

PROFINET IO offers several types of data transfer:

- Non-time-critical data such as configuration and diagnostic information is transferred acyclically with the TCP/IP communication standard.
- User data (input/output information) is exchanged cyclically between the IO Controller and the IO Device (real-time RT) within a defined time period – the update time.
- Time-critical user data, e.g. for motion control applications, is transferred isochronously with hardware support (isochronous real-time IRT).

A permanent communication channel is reserved on the Ethernet subnet for IRT communication. RT communication – cyclic data exchange between the IO Controller and IO Devices – and non-real-time TCP/IP communication take place parallel to the update time. In this way, all three communication types can exist in parallel on the same subnet.

Cyclic data exchange is handled in a specific time pattern, the **send clock**. STEP 7 calculates the send clock from the configuration information on the PROFINET IO system. The send clock is the shortest possible update time.

With STEP 7 V5.5 you configure the send clock (without IRT communication) centrally in the properties dialog of the PN interface in the *PROFINET* tab or in the properties tab of the PROFINET IO systems in the *Update time* tab.

With STEP 7 V11 you configure the send clock in the interface properties of the IO Controller. With the PN interface selected, select a value in the properties tab under *Advanced options > Real-time settings > IO communication* from the drop-down list *Smallest possible updating interval*.

The **update time** is the period within which each IO Device in the IO System has exchanged its user data with the IO Controller. The update time corresponds to the send clock or a multiple thereof. You can increase the update time manually, for example to reduce the bus load. Under certain circumstances, you can reduce the update time for individual IO Devices if you in return increase the update time for other devices whose user data can be exchanged non-time-critically.

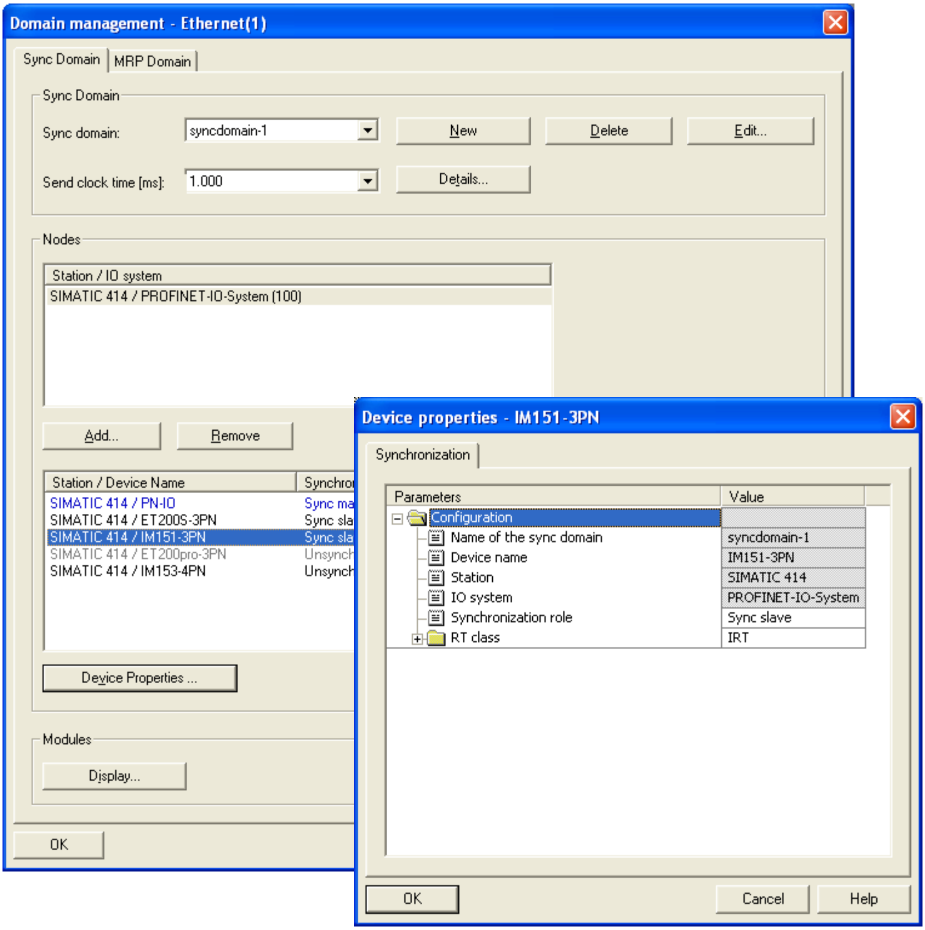
If the IO Device is not supplied by the IO Controller with input or output data within the **watchdog timer**, it switches to a safe state. The watchdog timer is calculated as the product of the update time and “Accepted update cycles without IO data”.

Real-time (RT) means that a system processes external events within a defined timeframe. If it responds predictably, it is called deterministic. In RT communication, transfer takes place at a specific point in time (send clock) within a defined interval (update time). PROFINET IO allows the use of standard network components for RT communication.

If not all data to be exchanged is transferred within the planned time frame, for example due to the addition of new network components, some data is distributed to other send clocks. This can result in an increase in the update time for individual IO Devices.

Isochronous real-time (IRT) is hardware-supported real-time communication designed, for example, for motion control applications. IRT frames are deterministically transmitted via planned communication paths in a specified order. IRT communication therefore requires network components that support this planned data transmission. Isochronous real-time is available in the "High flexibility" option for simple configuration and system expansion and in the "High performance" option for fast update times.

You use STEP 7 V5.5 to configure the IRT communication: You set up a new sync domain and determine a sync master to handle the synchronized distribution of



Sync domain: syncdomain-1

Send clock time [ms]: 1.000

Station / IO system
SIMATIC 414 / PROFINET-IO-System (100)

Station / Device Name	Synchro
SIMATIC 414 / PN-IO	Sync ma
SIMATIC 414 / ET200S-3PN	Sync sla
SIMATIC 414 / IM151-3PN	Sync sla
SIMATIC 414 / ET200pro-3PN	Unsynch
SIMATIC 414 / IM153-4PN	Unsynch

Parameters	Value
Configuration	
Name of the sync domain	syncdomain-1
Device name	IM151-3PN
Station	SIMATIC 414
IO system	PROFINET-IO-System
Synchronization role	Sync slave
RT class	IRT

Fig. 6.17 Example of the configuration of a new sync domain

the IRT frames to the sync slaves. IRT with the “High performance” option requires a topology configuration and thus a defined structure that takes account for the transmission properties of the cables and the switches used.

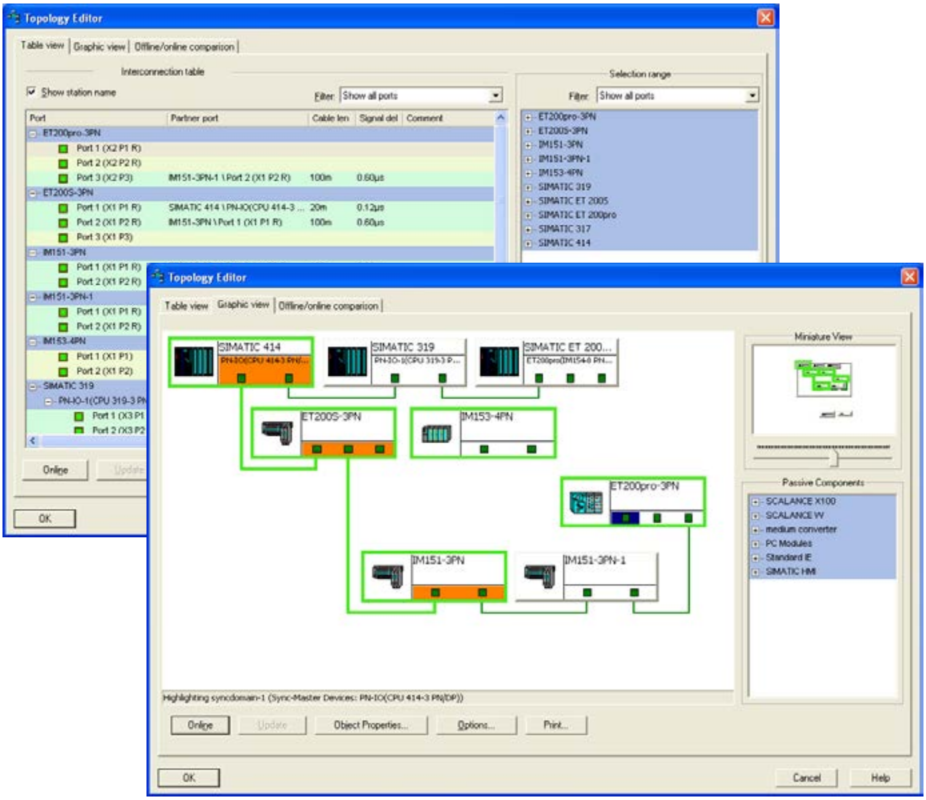
A **sync domain** is a group of PROFINET IO nodes that exchange synchronized data with each other. One node (this can be an IO Controller or an IO Device) assumes the role of the sync master, and the others are the sync slaves. A sync domain can contain several IO Systems, but an IO System is always assigned entirely to one single sync domain. Several sync domains can exist on one Ethernet subnet.

If an IO System is configured, a special sync domain is automatically created: the *syncdomain-defaul*t. All configured IO Systems, IO Controllers and IO Devices are first located in the sync domain syncdomain-default. You create a new sync domain for IRT communication, and transfer the IO System (from the *syncdomain-default*) to the new sync domain. Not all devices of an IO System have to be synchronized, or in other words, exchange data with IRT communication. During configuration, unsynchronized nodes are also added to the sync domain at first; at runtime only synchronized nodes remain in the sync domain. An example of the configuration of a sync domain an be seen in Fig. 6.17.

The **Topology Editor** allows cabling configuring of devices on the Industrial Ethernet subnet. The logical connections between the PROFINET devices are configured with the configuring tools HW Config and Network Config. The Topology Editor is used to configure the physical connections with the Length and Cable Type properties for determining the signal runtimes. Use of the Topology Editor is a prerequisite for using IRT communication (isochronous real time) with the “High performance” option.

The physical connections between devices on the Ethernet subnet are point-to-point connections. The connections on a PN interface are called ports. The Ethernet cable connects a device port with a port on the partner device. To enable several nodes to communicate with each other, they are connected to a switch that has several connections (ports) and that distributes signals. There are also S7 devices featuring a PN interface that has two or more ports connected by an integral switch. With this interface you can wire communication devices together in a linear bus topology without external switches. In tabular view, the Topology Editor shows the interconnection table with the port pairs of all configured active and passive components. The graphic view shows the configured devices, their ports, and the interconnection (Fig. 6.18).

The **Isochronous mode** function permits synchronous input, processing, and output of I/O signals in a fixed time pattern (constant bus cycle time). A prerequisite for isochronous mode is isochronous real-time (IRT) with the “High performance” option. The basis of the time pattern is the cycle time and the data cycle derived from this (the update time). Isochronous mode is available for a PROFINET IO system and for a PROFIBUS DP master system. With STEP 7 V5.5, you can configure isochronous mode for both systems; with STEP 7 V11 you can configure it for the PROFIBUS DP master system (chapter 6.14 "Isochronous mode program" on page 238).



Port	Partner port	Cable len	Signal del	Comment
ET200pro-3PN				
Port 1 (X2 P1 R)				
Port 2 (X2 P2 R)				
Port 3 (X2 P3)	IM151-3PN-1 \ Port 2 (X1 P2 R)	100m	0.60µs	
ET200S-3PN				
Port 1 (X1 P1 R)	SIMATIC 414 \ PN-IO(CPU 414-3 ...	20m	0.12µs	
Port 2 (X1 P2 R)	IM151-3PN \ Port 1 (X1 P1 R)	100m	0.60µs	
Port 3 (X1 P3)				
IM151-3PN				
Port 1 (X1 P1 R)				
Port 2 (X1 P2 R)				
IM151-3PN-1				
Port 1 (X1 P1 R)				
Port 2 (X1 P2 R)				
IM153-4PN				
Port 1 (X1 P1)				
Port 2 (X1 P2)				
SIMATIC 319				
PN-IO-1(CPU 319-3 PN				
Port 1 (X3 P1				
Port 2 (X3 P2				

Highlighting syncdomain-1 (Sync-Master Devices: PN-IO(CPU 414-3 PN/DP))

Fig. 6.18 Example of tabular and graphic view of the Topology Editor

The **Shared device** function allows different IO Controllers to access submodules (I/O modules and transfer areas) in one IO Device. The associated IO Device is used by the IO Controllers together ("shared" device). Each submodule of the shared device is assigned to an IO Controller. The basic conditions for use of a shared device are:

- ▷ The IO Controller and the IO Device must be present in the same Ethernet subnet.
- ▷ When using isochronous real-time communication (IRT), a shared device can only be used with the IRT option "High performance".
- ▷ The shared device function can only be used with "even" send cycle times.
- ▷ A shared device cannot be operated in an isochronous manner with the constant PROFINET IO cycle.

The *shared device* function is configured with STEP 7 V5.5 and is available for a CPU 400 with firmware version 6.0 and higher and for a CPU 300 or CPU ET 200 with firmware version 3.2 and higher. The prerequisite for configuring a shared device is a project with two or more IO Controllers and PROFINET IO systems on the same Ethernet subnet.

To create a (modular) shared device, open a controller station and drag the IO Device from the hardware catalog to the PROFINET IO system. Configure the modules by dragging them from the hardware catalog to the slot in the configuration table. Position all modules for all IO Controllers.

Following configuration, copy the IO Device into the clipboard, for example using the *Copy* command from the shortcut menu. Save the controller station, and open another one.

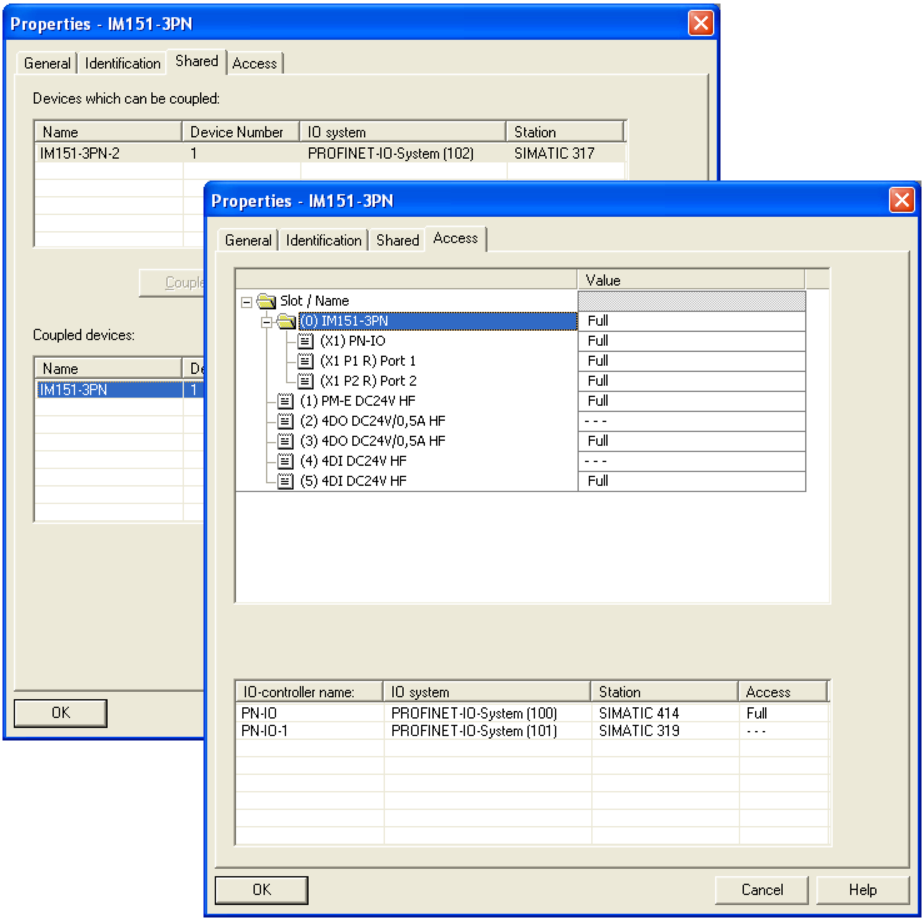
To insert the saved IO Device, right-click on the PROFINET IO system and select the *Shared insert* command from the shortcut menu. Then save the controller station. In both controller stations there is now an IO Device with identical configuration. Repeat inserting for the other IO Controllers if applicable.

To assign the modules to an IO Controller, open the *Access* tab. All modules are listed in a tree structure. A module has the value “Full” if it is assigned to the IO Controller of the currently open PROFINET IO system. Otherwise it has the value “---”. Open the shared devices in succession in each PROFINET IO system, and assign the modules to the associated IO Controller by clicking in the *Value* column (Fig. 6.19).

The **media redundancy** is used to increase the network availability by means of a special topology. The ends of a linear topology are connected into a ring topology in a station at the two connections of the PN interface. This station is the redundancy manager and the connections are the ring ports. If a station in the ring network fails, an alternative communication path can be made available. Up to 50 devices can participate per ring by means of the Media Redundancy Protocol (MRP) used with SIMATIC S7. The media redundancy must be configured in the interface properties of all participating stations in the tab *Media Redundancy* (STEP 7 V5.5) or under *Advanced options > Media redundancy* (STEP 7 V11). IRT communication cannot be used if media redundancy is configured.

Device replacement without removable medium: When replacing an IO Device, a device name must be assigned to the new IO Device in order to make it known (again) to the IO Controller. This can be carried out – depending on the IO Device – using a memory card or the programming device. Under certain conditions, the new IO Device can be identified by means of neighbor relationships between the other IO Devices and the IO Controller and assigned a new device name by the IO Controller. Among the prerequisites are that a port interconnection is configured, and that with STEP 7 V5.5, in the interface properties of the IO Controller in the *General* tab, the *Support device replacement without removable medium* checkbox is checked, or that with STEP 7 V11, when configuring the interface properties under *Advanced options > Interface options* the *Allow device replacement without removable medium* checkbox is checked. Only new IO Devices or IO Devices that have been reset to the factory settings should be used as replacement devices.

With a **prioritized startup**, the startup of IO Devices in a PROFINET IO system with RT and IRT communication is carried out faster. Here, special conditions



Devices which can be coupled:

Name	Device Number	IO system	Station
IM151-3PN-2	1	PROFINET-IO-System (102)	SIMATIC 317

Coupled devices:

Name	De
IM151-3PN	1

Slot / Name	Value
(0) IM151-3PN	Full
(X1) PN-IO	Full
(X1 P1 R) Port 1	Full
(X1 P2 R) Port 2	Full
(1) PM-E DC24V HF	Full
(2) 4DO DC24V/0,5A HF	- - -
(3) 4DO DC24V/0,5A HF	Full
(4) 4DI DC24V HF	- - -
(5) 4DI DC24V HF	Full

IO-controller name:	IO system	Station	Access
PN-IO	PROFINET-IO-System (100)	SIMATIC 414	Full
PN-IO-1	PROFINET-IO-System (101)	SIMATIC 319	- - -

Fig. 6.19 Example of configuration of a shared device

must be observed when wiring and supporting activities must be performed in the user program. The maximum possible number of IO Devices controlled with prioritized startup depends on the IO Controller used. With STEP 7 V5.5 you configure the prioritized startup in the properties of the PROFINET interface of an IO Device in the *General* tab by means of the *Prioritized startup* checkbox. With STEP 7 V11 you configure the prioritized startup in the properties of the PROFINET interface of an IO Device by means of the *Prioritized startup* checkbox. You can find the checkbox under *Advanced options* > *Interface Options* or – with an intelligent IO Device – under *Operating mode* (with IO Device mode switched on and assigned IO Controller).

6.14 Isochronous mode program

Reference is made to isochronous mode if a program is executed synchronous to a PROFIBUS DP cycle or PROFINET IO cycle. Reproducible response times are obtained in connection with constant bus cycle times. The user program executed in isochronous mode is present in organization blocks OB 61 to OB 64. For process image updating in isochronous mode, there are the system functions SYNC_PI and SYNC_PO (Table 6.14).

Table 6.14 Blocks for isochronous mode interrupts

Blocks	CPU 1200	CPU 300	CPU 400
Organization blocks	–	OB 61 *)	OB 61 to OB 64
SYNC_PI Isochronous updating of input process image partition	–	×	×
SYNC_PO Isochronous updating of output process image partition	–	×	×

*) not for all CPU types

Configuration of isochronous mode for PROFIBUS with STEP 7 V5.5

First, you configure the DP master system and the stations and modules participating in isochronous mode. In the CPU properties, assign the priority, the DP master system, and the process image partitions to the organization block in the *Isochronous mode interrupts* tab.

To turn on the constant bus cycle time and isochronous mode, select the DP master and choose *Edit > Object properties*. In the following dialog boxes, set the *DP master* operating mode and one of the bus profiles *DP* or *Custom*. You then select the *Activate constant bus cycle time* checkbox.

You can modify the suggested constant bus cycle time, but not below the displayed minimum time. The *Details* button shows the individual proportions of the equidistance time. Please note that the constant bus cycle time increases the more programming devices are connected directly to the PROFIBUS subnet and the more intelligent DP slaves are in the DP master system.

In addition, you activate the isochronous mode in the participating DP stations and modules or, and electronic modules. For the modules and electronic modules, you can also set the corresponding process image partition for the isochronous update that you have specified in the CPU parameter assignment.

Configuration of isochronous mode for PROFINET with STEP 7 V5.5

In order to configure isochronous mode, you create a PROFINET IO system with the controller station and the IO Devices, import the stations into a sync domain with the IRT option *High performance*, and configure networking between the stations using the Topology Editor. In the device stations, you assign a process image partition to the modules in their properties on the *Addresses* tab.

You assign the isochronous mode organization block to the PROFINET IO system in the CPU properties: Open the controller station and double-click on the CPU to open the CPU properties window; select the *Isochronous mode interrupts* tab and set the PROFINET IO system for the organizational block. Click on the *Details* button. The duration of the application cycle is calculated from the data cycle, multiplied by a factor that you specify on this tab. It is therefore necessary to estimate the processing time of the isochronous mode program and to compare this with the data cycle time.

If applicable, you set the delay time on this tab with which the isochronous mode OB is to start, and assign the process image partition which you have set for the module addresses in the IO Devices. The following methods are available for determining the times Ti and To:

- *Automatic* – STEP 7 determines the times and sets them the same for all IO Devices
- *Fixed* – you enter the times which then apply to all IO Devices
- *In the IO Device* – the times are then set individually in the respective IO Device.

To assign the modules to isochronous mode, select the IO Device, double-click on the PN interface in the configuration table, and select the *IO cycle* tab in the Properties dialog. In the section *Isochronous mode*, assign the isochronous mode OB to the IO Device, and click on the *Isochronous modules/submodules...* button. You can activate or deactivate the individual modules of the IO Device for isochronous mode in the displayed window. Proceed in the same manner for the other IO Devices. You are provided with an overview of the configuration if, with the controller station selected, you then select the *Edit > PROFINET IO > Isochronous mode* command.

Configuring isochronous mode for PROFIBUS with STEP 7 V11 inside TIA Portal

A prerequisite for configuration of isochronous mode is the constant bus cycle time and the corresponding functionality of the participating DP components. Following configuration of the DP master system with appropriate modules (CPU with integral DP interface as well as ET 200S and/or ET 200M DP interface modules with input/output modules with isochronous mode capability), assign the DP master system and the process image partition to the organization block in the CPU properties in the *Isochronous mode interrupts* tab.

To switch on the constant bus cycle time and isochronous mode, activate the *Activate constant bus cycle time* checkbox in the properties of the PROFIBUS subnet under *Constant bus cycle time*. Activate isochronous mode for the participating DP slaves in the *Detail overview* section and, if you “open” a line with a DP slave, the isochronous mode of the individual I/O modules in the DP slave. In the *Ti/To values* column you can select the mode from a drop-down list for calculation of the Ti/To values:

- *From the subnet*: The currently configured DP slave obtains the Ti/To values from the subnet and thus has the same values as the other DP slaves which also obtain their values from the subnet.
- *Automatic minimal*: If you manually change the Ti/To values of another DP slave when in this setting, any adaptations which may be necessary on the currently configured DP slave are carried out automatically.
- *Manual*: With this setting, you manually enter the Ti/To values for the currently configured DP slave.

You can also make these settings in the interface properties of the DP slave under *Isochronous mode*. Each module or submodule involved in isochronous mode must be addressed in a process image partition. You set the process image partition for the module in the Device view in the module properties under I/O addresses.

Isochronous updating of process images

The system functions SYNC_PI *Isochronous update of inputs* and SYNC_PO *Isochronous update of outputs* are available for an isochronous and data-consistent update of process image partitions. Both system functions must only be called in an isochronous mode interrupt OB. Direct access to these process image partitions should be avoided.

Processing of isochronous mode interrupts

Isochronous mode interrupts are only processed in RUN mode. A isochronous mode interrupt in the STARTUP, STOP or HOLD operating modes is rejected. The reaction time in isochronous mode is the sum of the times Ti and To as well as the bus cycle time (see Fig. 6.20). With PROFINET it is possible to wait for several bus cycles before the isochronous OB is called again.

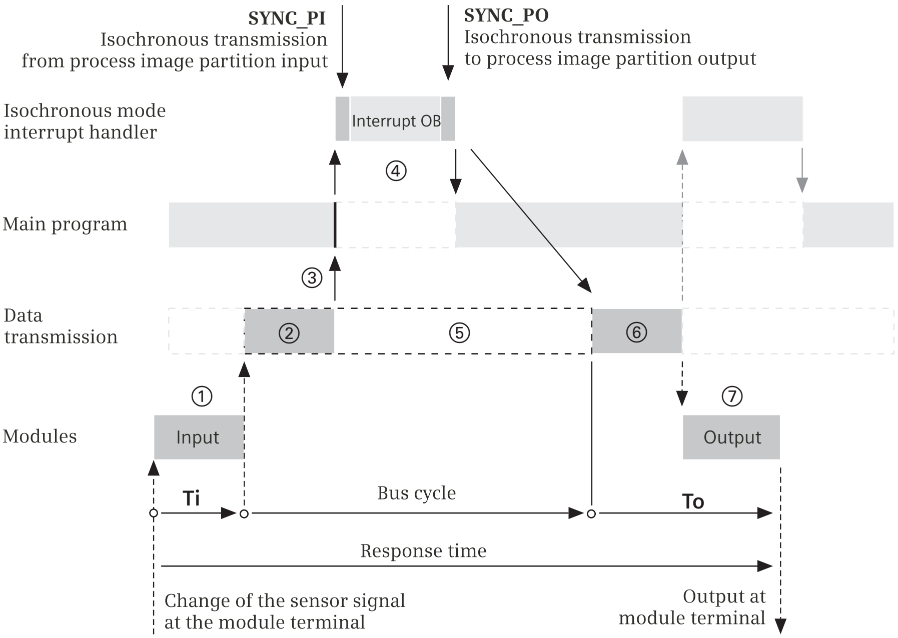
Ti is the time required for reading in the I/O signals. It contains the execution time in the input modules or electronic modules and additionally, in the case of modular DP slaves or IO Devices, the transfer time on the backplane bus.

At the end of Ti, the input information for the transfer is available. The bus cycle then commences. In PROFIBUS, the bus cycle is composed of the cyclic data transmission, the transmission of the acyclic services, and a reserve time. In PROFINET, the IRT communication is performed first, then the RT communication, and finally the transmission of the acyclic services. After data transfer to the DP slaves or after IRT communication, the isochronous mode interrupt OB begins. Between completion of the execution of this OB and the next bus cycle, there must be time for the execution of the main program.

To is the time required for output of the I/O signals. It comprises the transfer time on the subnet as well as the processing time in the output modules or electronic

Principle of isochronous program processing

Isochronous program processing is implemented in parallel with data transmission in a PROFIBUS DP master system or in a PROFINET IO system. With an isochronous mode interrupt, the bus cycle synchronizes the start of the assigned organization block.



1) An input module in a distributed station requires the time Ti to provide an input signal change at the bus.

2) The bus cycle begins with the cyclic transmission of input/output signals from and to the DP slaves (PROFIBUS DP) or with IRT communication (PROFINET IO).

3) After completing transmission, the isochronous mode interrupt is triggered, which starts the isochronous mode OB.

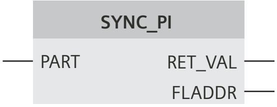
4) In the isochronous mode OB, the system functions SYNC_PI and SYNC_PO update the process image partitions of the input and output signals assigned to the isochronous mode OB.

5) Bus transmission continues in the meantime. Acyclic services such as access to programming devices and, on PROFINET, RT communication can now also be implemented.

6) The output signals that were generated in the isochronous mode program are transmitted to the distributed stations during the following bus cycle.

7) The time To also includes the time required by a distributed station to connect a bus signal to a terminal of the output module.

Isochronous update of process image partition of the inputs



Isochronous update of process image partition of the outputs

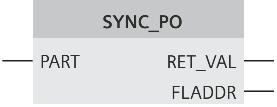


Fig. 6.20 Using an isochronous mode interrupt

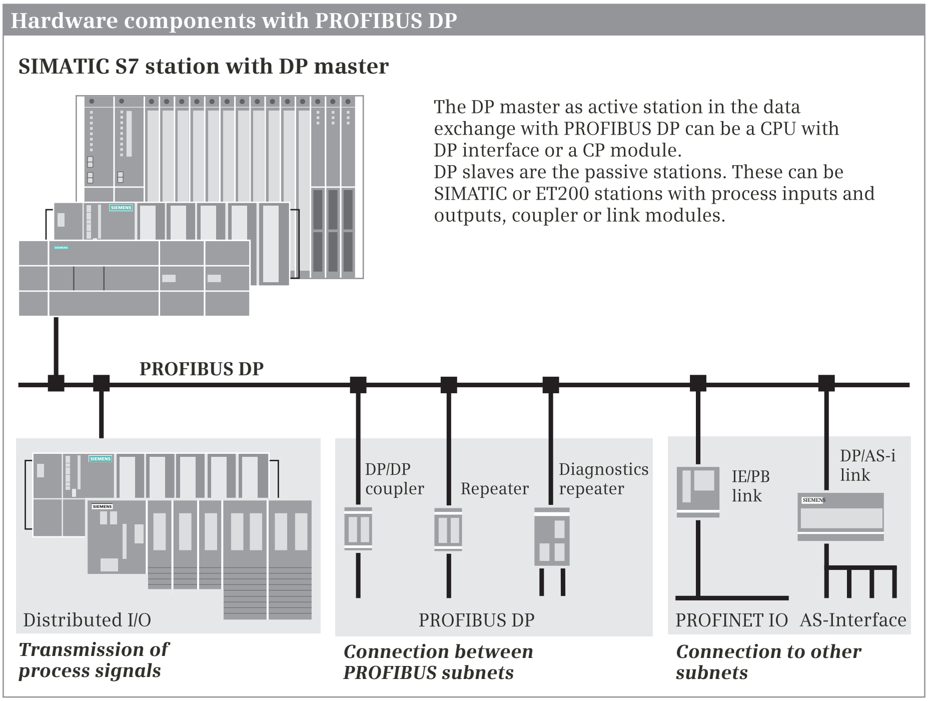
modules. In the case of modular DP slaves or IO Devices, the transfer time on the backplane bus is also added.

6.15 Distributed I/O with PROFIBUS DP

PROFIBUS (Process Field Bus) is a vendor-independent standard according to IEC 61158 and EN 50170. The following PROFIBUS variants exist:

- ▷ PROFIBUS FMS (fieldbus message specification) for universal communication tasks in the field
- ▷ PROFIBUS PA (process automation) for process automation, particularly in intrinsically safe areas with the physical characteristic according to IEC 1158-2, and
- ▷ PROFIBUS DP (distributed peripherals) for process automation with fast cyclic transfer of small amounts of data.

PROFIBUS DP enables the transfer of predominantly binary process data between a “DP master” in the (central) programmable controller and the decentralized field devices, the “DP slaves” (Fig. 6.21). A maximum of 126 nodes (stations) can be operated on a PROFIBUS network, which can be divided into segments



The DP master as active station in the data exchange with PROFIBUS DP can be a CPU with DP interface or a CP module.
DP slaves are the passive stations. These can be SIMATIC or ET200 stations with process inputs and outputs, coupler or link modules.

Hardware components with PROFIBUS DP
SIMATIC S7 station with DP master
PROFIBUS DP
DP/DP coupler
Repeater
Diagnostics repeater
IE/PB link
DP/AS-i link
Distributed I/O
PROFIBUS DP
PROFINET IO
AS-Interface
Transmission of process signals
Connection between PROFIBUS subnets
Connection to other subnets

Fig. 6.21 Components of a PROFIBUS DP master system

with up to 32 stations. The DP master and all DP slaves controlled by it form a PROFIBUS DP master system. Several PROFIBUS DP master systems can be operated in a PROFIBUS subnet.

General procedure for configuring PROFIBUS DP

A prerequisite for the configuration of a PROFIBUS DP master system is a project with a PLC station. The PLC station contains a CPU with DP interface or a CP module with integrated DP master.

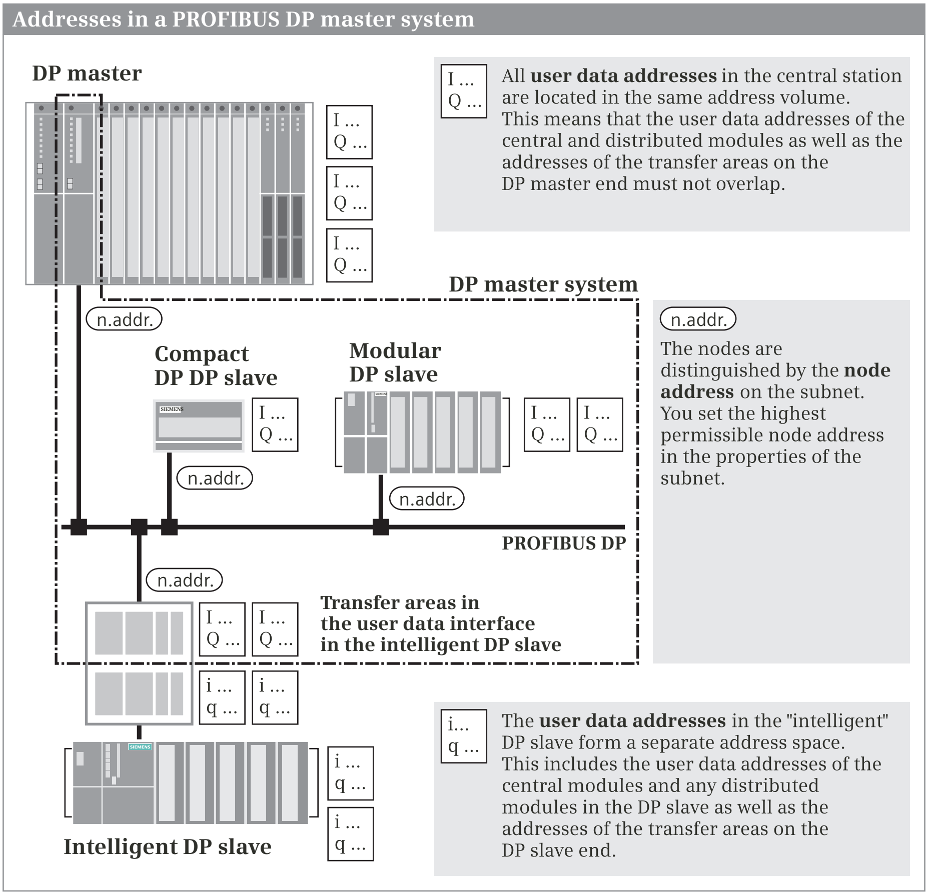
In the CPU properties, activate the DP master under the DP interface. If the interface is a combined MPI/DP interface, you must first switch the interface to *PROFIBUS*. Connect the DP interface to a PROFIBUS subnet and assign a PROFIBUS DP master system to the DP master. Select the DP slaves from the hardware catalog, drag them to the working area, and connect their DP interface with the PROFIBUS DP master system.

An “intelligent” DP slave (in short: “I-Slave”) is an independent station with a CPU that is coupled via a transfer area to the PROFIBUS DP master system. To configure an I-Slave, create an S7 or ET200 station with a CPU, set the operating mode of the DP interface to “DP slave”, connect the DP interface to the PROFIBUS DP master system, and configure the transfer area as the user data interface to the DP master.

Addresses in the PROFIBUS DP master system

A PROFIBUS DP master system with the DP master and all DP slaves is integrated into the address structure of the central CPU, because the DP slaves are addressed from the central CPU as centrally arranged modules. The following addresses are available in a PROFIBUS DP master system:

- ▷ Node address, station number
  Each station on the PROFIBUS subnet has a unique address within the subnet. The station (the DP master or a DP slave) is addressed on PROFIBUS via this node address.
- ▷ Geographic address:
  The geographic address of a DP slave corresponds to the slot address of a centrally located module. It is composed of the number of the PROFIBUS DP master system, the station number, and the slot number.
- ▷ Logical address, module start address:
  You access the user data of a DP slave under the logical address. The smallest logical address corresponds to the module start address of a central module.
- ▷ Diagnostic address:
  Modules without user data that can send diagnostic data are addressed via a diagnostic address. The diagnostic address occupies one byte in the address area of the peripheral inputs.



All **user data addresses** in the central station are located in the same address volume. This means that the user data addresses of the central and distributed modules as well as the addresses of the transfer areas on the DP master end must not overlap.

The nodes are distinguished by the **node address** on the subnet. You set the highest permissible node address in the properties of the subnet.

The **user data addresses** in the "intelligent" DP slave form a separate address space. This includes the user data addresses of the central modules and any distributed modules in the DP slave as well as the addresses of the transfer areas on the DP slave end.

Fig. 6.22 Addresses in a PROFIBUS DP master system

A compact DP slave behaves like a single module, while a modular DP slave behaves like several modules. In an “intelligent” DP slave, the transfer areas of the user data interface simulate modules. The user data addresses of all modules of a station may not overlap (Fig. 6.22).

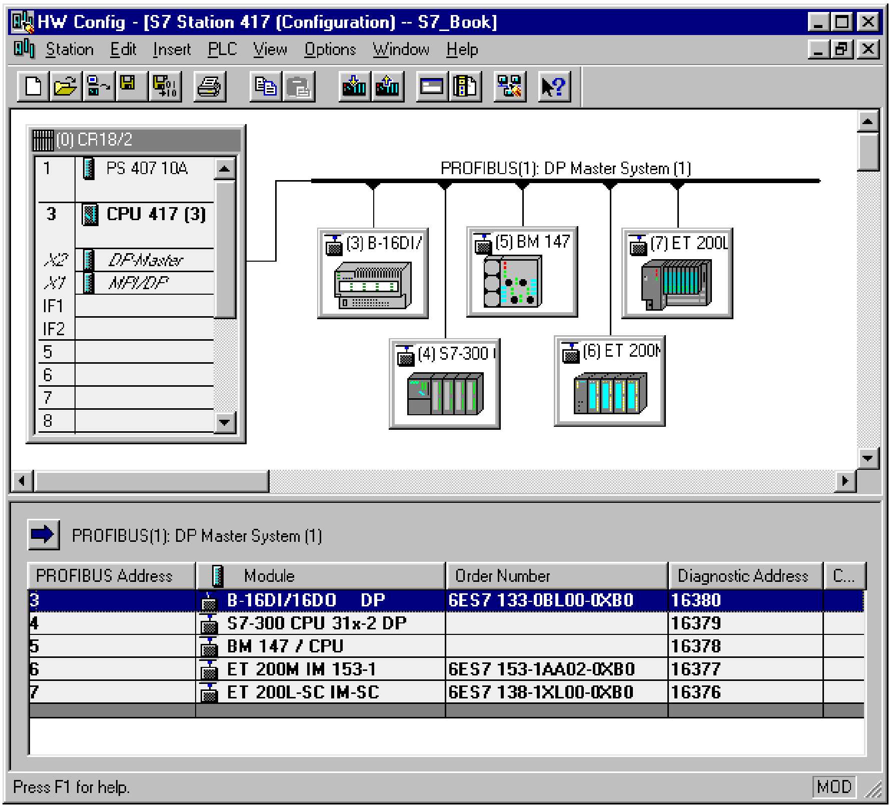
Configuration of a PROFIBUS DP master system with STEP 7 V5.5

Prerequisite: You have created a project with the central PLC station and placed a module with a DP interface.

Double-clicking on the *Hardware* object in the SIMATIC Manager starts HW Config. Double-clicking on the DP interface in the configuration table opens the properties dialog of the interface. In the *General* tab, with a combined MPI/DP interface, set the type of interface. To network with a subnet, click the *Properties* button, and in the dialog box select the subnet to which the DP interface is to be connected. In the *Operating mode* tab, enable operation as a DP master. To create a PROFIBUS DP master system, with the DP interface selected, choose the *Insert > DP master system* command.

To add a DP slave, drag the desired DP slave from the hardware catalog to the PROFIBUS DP master system in the working area. The DP slaves can be found in the hardware catalog under PROFIBUS DP and the corresponding folder. Double-click the DP slave to open the Properties dialog. With a modular DP slave, you can insert additional modules here that you drag from the hardware catalog under the interface module used (!) to the working area and set the user data addresses if necessary. An example of a PROFIBUS DP master system is shown in Fig. 6.23.

You first create an "intelligent" DP slave as an autonomous PLC station using the SIMATIC Manager, insert a module with a DP interface, and connect it to the PROFIBUS subnet. To set the operating mode, select the DP interface in the configuration table and then choose *Edit > Object Properties*. Select the *Operating mode* tab in the Properties window, and check the *DP slave* checkbox there. Now you can configure the transfer areas of the user data interface on the *Configuration* tab from the viewpoint of the DP slave.


HW Config - [S7 Station 417 (Configuration) -- S7_Book]
Station Edit Insert PLC View Options Window Help
PROFIBUS(1): DP Master System (1)
(3) B-16DI/
(5) BM 147
(7) ET 200L
(4) S7-300 (
(6) ET 200M

(0) CR18/2

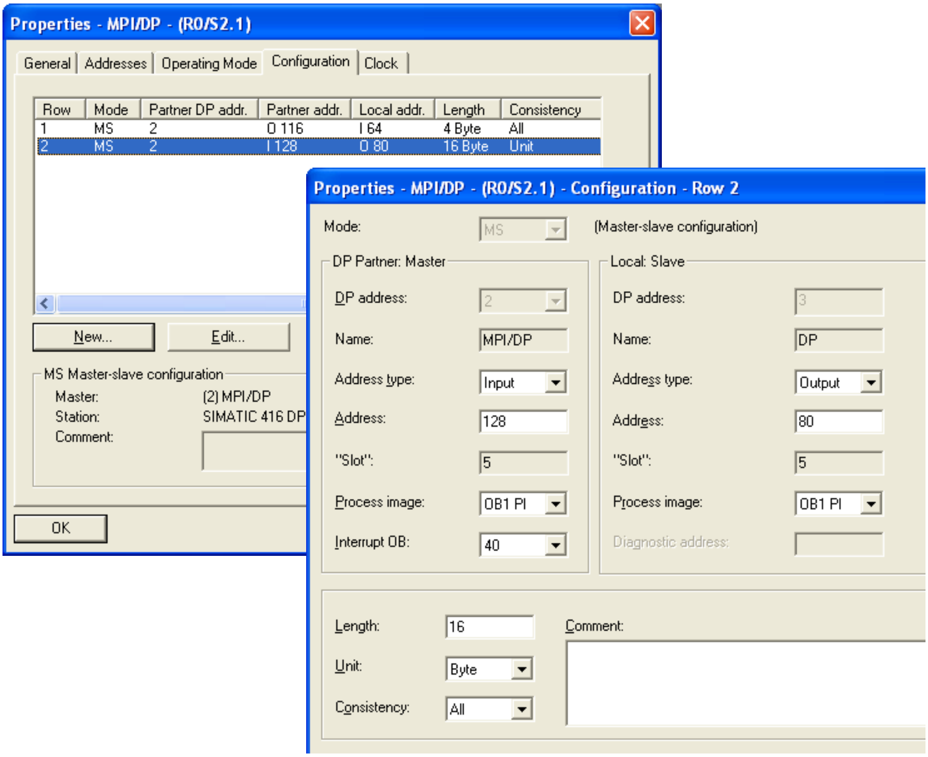
1	PS 407 10A
3	CPU 417 (3)
X2	DP-Master
X1	MPI/DP
IF1	
IF2	
5	
6	
7	
8	

PROFIBUS(1): DP Master System (1)

PROFIBUS Address	Module	Order Number	Diagnostic Address	C...
3	B-16DI/16DO DP	6ES7 133-0BL00-0XB0	16380	
4	S7-300 CPU 31x-2 DP		16379	
5	BM 147 / CPU		16378	
6	ET 200M IM 153-1	6ES7 153-1AA02-0XB0	16377	
7	ET 200L-SC IM-SC	6ES7 138-1XL00-0XB0	16376	

Press F1 for help. MOD

Fig. 6.23 Representation of a PROFIBUS DP master system in STEP 7 V5.5


Properties - MPI/DP - (R0/S2.1)
General
Addresses
Operating Mode
Configuration
Clock
New...
Edit...
OK
Properties - MPI/DP - (R0/S2.1) - Configuration - Row 2

Row	Mode	Partner DP addr.	Partner addr.	Local addr.	Length	Consistency
1	MS	2	O 116	I 64	4 Byte	All
2	MS	2	I 128	O 80	16 Byte	Unit

MS Master-slave configuration
Master: (2) MPI/DP
Station: SIMATIC 416 DP
Comment:

Mode: MS (Master-slave configuration)

DP Partner: Master		Local: Slave	
DP address:	2	DP address:	3
Name:	MPI/DP	Name:	DP
Address type:	Input	Address type:	Output
Address:	128	Address:	80
"Slot":	5	"Slot":	5
Process image:	OB1 PI	Process image:	OB1 PI
Interrupt OB:	40	Diagnostic address:	

Length: 16
Unit: Byte
Consistency: All
Comment:

Fig. 6.24 Example of a transfer area of an I-Slave (STEP 7 V5.5)

After activating the operating mode, click *New...* to create a new transfer area in the user data interface to the IO Controller. In the properties of the transfer area, specify the direction of transmission and the user data addresses from the viewpoint of the IO Device (Fig. 6.24).

The configured I-slaves are stored in the hardware catalog under *PROFIBUS DP* and *Configured stations*. An icon represents each type of station, for example *ET200S/CPU* for an ET200S station with an I-Slave. Select the desired station type and drag it to the PROFIBUS DP master system. In the *Coupling* tab, select the configured -I-Slave and click the *Coupling* button. Now you can configure the user data addresses of the transfer areas on the *Configuration* tab from the viewpoint of the DP master.

Configuration of a PROFIBUS DP master system with STEP 7 inside TIA Portal

Prerequisite: You have created a project with the central PLC station and inserted a module with a DP interface.

Open the PLC station with the hardware configuration and select the *Device view* tab in the working window. With a combined MPI/DP interface, under *MPI/DP inter-*

face > PROFIBUS address in the module properties, select the interface type *PROFIBUS* and network the interface with a PROFIBUS subnet. In the *Operating mode* tab, enable operation as a *DP master*. To insert a PROFIBUS DP master system, switch to the *Network view* tab, select the DP interface of the DP master, and choose *Assign master system* from the shortcut menu.

To add a DP slave to the PROFIBUS DP system, drag the desired DP slave from the hardware catalog to the PROFIBUS DP master system in the working area. With a selected DP slave, you can set the properties of the DP slave in the device view, for example, by adding desired modules and setting their addresses. Fig. 6.25 shows a PROFIBUS DP master system in network view with the central DP master and a compact DP slave (ET 200eco), a modular DP slave (ET 200 with IM 153-2), and an I-Slave (ET 200S with IM 151-7CPU).

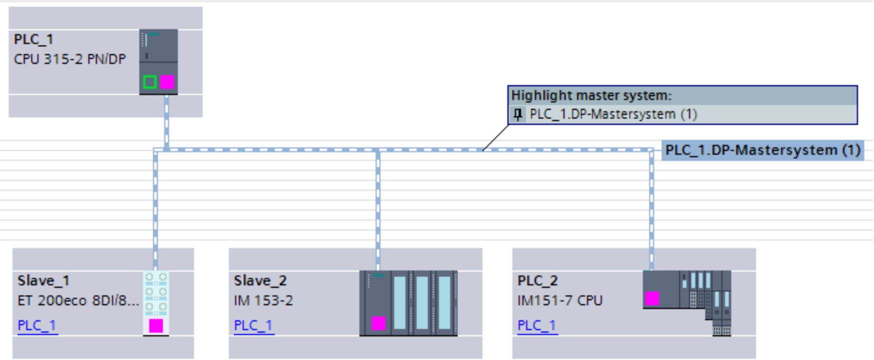
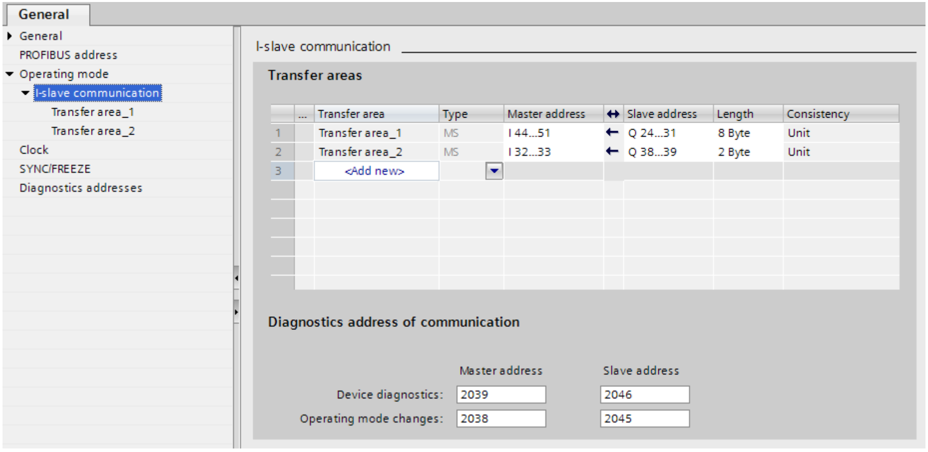


Fig. 6.25 Example of a PROFIBUS DP master system (STEP 7 inside TIA Portal)



I-slave communication

Transfer areas

	...	Transfer area	Type	Master address	↔	Slave address	Length	Consistency
1		Transfer area_1	MS	I 44...51	←	Q 24...31	8 Byte	Unit
2		Transfer area_2	MS	I 32...33	←	Q 38...39	2 Byte	Unit
3		<Add new>						

Diagnostics address of communication

	Master address	Slave address
Device diagnostics:	2039	2046
Operating mode changes:	2038	2045

Fig. 6.26 Example of configuration of a transfer area

You create an I-Slave as a separate PLC station. You establish a connection to the existing subnet by dragging the DP interface of the I-Slave to the DP interface of another device on the subnet, for example to the DP interface of the DP master. In the interface properties of the I-Slave, activate the *DP slave* checkbox in the *Operating mode* group and select the assigned DP master from the drop-down list. The station is then added as DP slave to the PROFIBUS DP master system.

You configure the user data interface to the DP master in the interface properties of the I-Slave. In the *Operating mode* group, choose *I-Slave communication* and double-click in the *Transfer areas* table on *<Add new>*. A new transfer area is created. Set the user data addresses here, both on the side of the I-Slave as well as on the side of the DP master. You can create several transfer areas. You are given detailed information about a transfer area when you select it in the Properties navigation (Fig. 6.26).

6.16 Special functions for PROFIBUS DP

Output intervals of equal length through constant bus cycle time

In the normal case, of the DP master controls the DP slaves assigned to it cyclically and without pauses. The time intervals may vary as a result of S7 communication, for example if the programming device carries out control functions over the PROFIBUS subnet. If outputs are to be controlled via the distributed I/O at intervals that are always equal, constant bus cycle times can be set if there is a correspondingly configured DP master. Here, the DP master must be the only class 1 master on the PROFIBUS. Constant bus cycle times are possible for the bus profiles *DP* and *Custom*, and they are a prerequisite for isochronous mode (see chapter 6.14 "Isochronous mode program" on page 238).

Synchronizing DP slaves with SYNC and FREEZE

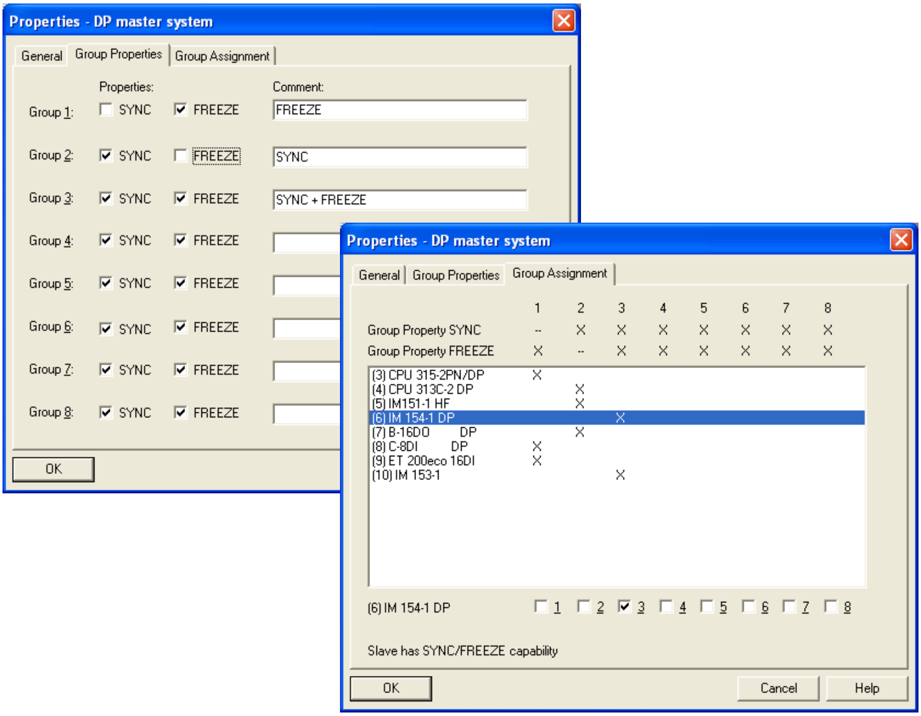
The DP master reads the input data from the DP slaves in chronological succession and receives signal states that are current in this sequence. The command FREEZE allows you to read in related input data simultaneously that is distributed over multiple DP slaves. The FREEZE command requests the DP slaves combined into a group to simultaneously (synchronously) freeze the current input signal states to allow them to then be cyclically fetched by the master. These input signals retain their value until a new FREEZE command causes the DP slaves to read in and freeze the now current input signals or until the UNFREEZE command is issued, which cancels the effects of FREEZE.

A similar procedure is used for the output signals. The DP master writes the output data to the DP slaves consecutively, and the signals are output in this order. The SYNC command outputs related data, which are distributed over several DP slaves, simultaneously to the process. SYNC causes the DP slaves combined in a group to output their output signal states simultaneously (synchronously) and retain these states unchanged. The DP master can now write the new signal states in succession to the DP slaves. After transmission is complete, use the SYNC command again to ensure the synchronous output of the new output signals. The DP slaves retain the states of the output signals until you output the current values with a SYNC command or cancel SYNC with an UNSYNC command.

By calling the system function DPSYC_FR, you cause a SYNC or FREEZE command to be issued in the user program. The DP master then sends the corresponding command simultaneously to all DP slaves. Prerequisite for the use of SYNC and FREEZE is that the DP master and the participating DP slaves provide the corresponding functionality and that you have configured the SYNC/FREEZE groups with HW Config. You can generate up to eight SYNC/FREEZE groups per DP master system that are to execute either the SYNC command, the FREEZE command, or both. You configure the SYNC/FREEZE groups following the configuration of the DP master system when all DP slaves are present in the DP master system.

In STEP 7 V5.5, select the interface of the DP master in the network configuration and choose *Master system...* from the shortcut menu. You see a dialog window in which you can assign a group the properties SYNC, FREEZE, or both in the *Group Properties* tab. In the *Group Assignment* tab, you assign the DP slaves to the groups (Fig. 6.27).

With STEP 7 inside TIA Portal, you assign the DP slave to a group in the interface properties of the DP slave in the SYNC/FREEZE properties group.



	Properties:		Comment:
Group 1:	□ SYNC	☑ FREEZE	FREEZE
Group 2:	☑ SYNC	□ FREEZE	SYNC
Group 3:	☑ SYNC	☑ FREEZE	SYNC + FREEZE
Group 4:	☑ SYNC	☑ FREEZE	
Group 5:	☑ SYNC	☑ FREEZE	
Group 6:	☑ SYNC	☑ FREEZE	
Group 7:	☑ SYNC	☑ FREEZE	
Group 8:	☑ SYNC	☑ FREEZE	

	1	2	3	4	5	6	7	8
Group Property SYNC	--	X	X	X	X	X	X	X
Group Property FREEZE	X	--	X	X	X	X	X	X
(3) CPU 315-2PN/DP	X							
(4) CPU 313C-2 DP		X						
(5) IM151-1 HF		X						
(6) IM 154-1 DP			X					
(7) B-16DO DP		X						
(8) C-8DI DP	X							
(9) ET 200eco 16DI	X							
(10) IM 153-1			X					

(6) IM 154-1 DP □ 1 □ 2 ☑ 3 □ 4 □ 5 □ 6 □ 7 □ 8

Slave has SYNC/FREEZE capability

Fig. 6.27 Example of configuration of SYNC and FREEZE groups (STEP 7 V5.5)

“Listening in” on PROFIBUS with direct data exchange (direct communication)

In a DP master system, a DP master only controls the DP slaves that have been assigned to it. On correspondingly configured stations, only one other node – called master or slave, *receiver* or *subscriber* – can “listen in” on the PROFIBUS subnet as to which input data a DP slave – called *sender* or *publisher* – sends to its DP master. This direct data exchange is also referred to as direct communication. You can also use direct data exchange between two DP master systems on the same PROFIBUS subnet. For example, the DP master in DP master system 1 can “listen in” in this manner to the data of a DP slave in DP master system 2.

6.17 DPV1 interrupts

You use DPV1 interrupts in connection with PROFIBUS DPV1 slaves or PROFINET IO devices. A correspondingly equipped station can call one of the organization blocks OB 55 to OB 57 with a DPV1 interrupt in the CPU (Table 6.15).

Table 6.15 Organization blocks for DPV1 interrupts

Type of program, functions	CPU 1200	CPU 300	CPU 400
DPV1 interrupt routine	–	OB 55 to OB 57 *)	OB 55 to OB 57

*) not for all CPU types

Triggering a DPV1 interrupt

Appropriately equipped PROFIBUS DPV1 slaves and PROFINET IO devices can trigger the following interrupts:

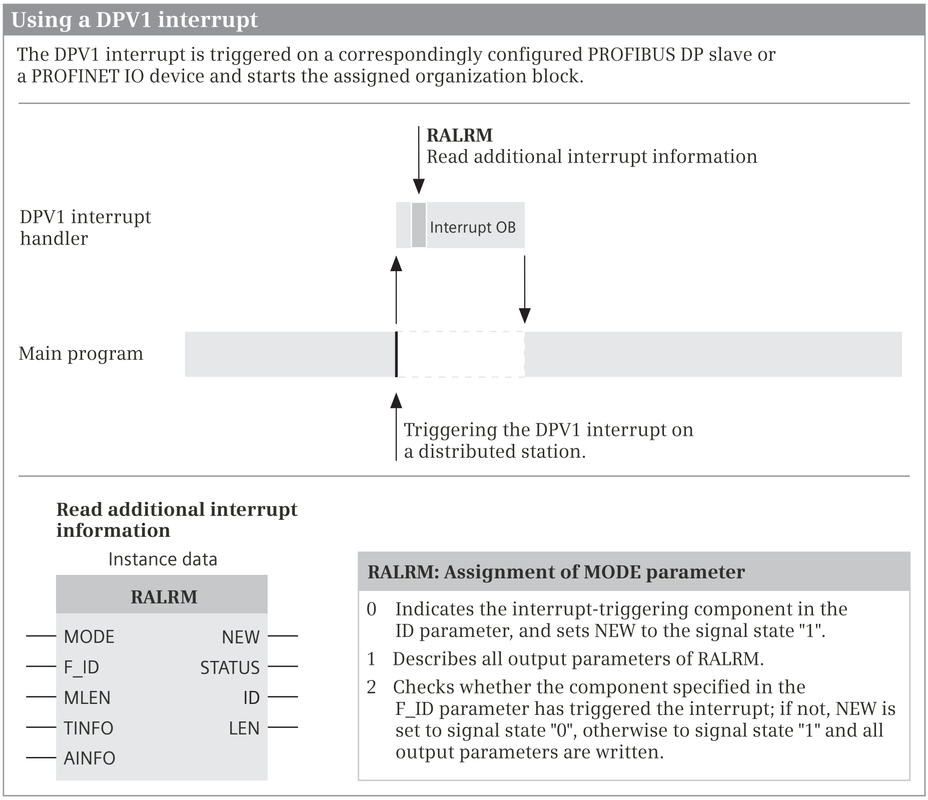
- ▷ Status interrupt if, for example, the DPV1 slave changes its operating mode; the interrupt organization block OB 55 is called.
- ▷ Update interrupt if, for example, the DPV1 slave is reparameterized via PROFIBUS or directly; the interrupt organization block OB 56 is called.
- ▷ Manufacturer interrupt if an event envisaged for this by the manufacturer occurs in the DPV1 slave; the interrupt organization block OB 57 is called. The events triggering the interrupt are defined by the manufacturer of the DPV1 slave.

If a DPV1 interrupt occurs, the assigned organization block must also be available. If this is not the case, the CPU enters a message in the diagnostic buffer and calls the asynchronous error block OB 85 *program execution error* or goes into STOP mode. DPV1 interrupts are only processed if the CPU is in RUN mode. DPV1 interrupts occurring during startup are entered in the diagnostic buffer and the module state data.

Querying interrupt information

In the interrupt organization block, you can query which DP slave triggered the interrupt. Bytes 5, 6, and 7 of the start information contain the start address of this module. Bytes 8 to 11 contain additional information such as the interrupt type and the identification of whether this is an incoming or outgoing event.

The additional interrupt information can be read using the system function block RALRM. The assignment of the MODE parameter determines the mode of the system block RALRM (Fig. 6.28). In bytes 0 to 19, the destination area TINFO (task information) contains the complete start information of the organization block in which RALRM was called, independent of the nesting depth in which it was called. Management information is present in bytes 20 to 27, e.g. which component has triggered the interrupt. In bytes 0 to 3 (bytes 0 to 25 with PROFINET), the destination area AINFO (alarm information) contains the header information, e.g.the number of received bytes of the additional interrupt information or interrupt type. Bytes 4 to 223 (bytes 26 to 1431 with PROFINET) contain the component-specific additional interrupt information itself.



Using a DPV1 interrupt

The DPV1 interrupt is triggered on a correspondingly configured PROFIBUS DP slave or a PROFINET IO device and starts the assigned organization block.

RALRM
Read additional interrupt information
DPV1 interrupt handler
Interrupt OB
Main program
Triggering the DPV1 interrupt on a distributed station.
Read additional interrupt information
Instance data
RALRM
MODE
F_ID
MLEN
TINFO
AINFO
NEW
STATUS
ID
LEN

RALRM: Assignment of MODE parameter

0 Indicates the interrupt-triggering component in the ID parameter, and sets NEW to the signal state "1".

1 Describes all output parameters of RALRM.

2 Checks whether the component specified in the F_ID parameter has triggered the interrupt; if not, NEW is set to signal state "0", otherwise to signal state "1" and all output parameters are written.

Fig. 6.28 Using a DPV1 interrupt

7 Operator control and monitoring

Overview of devices and configuration tools

To control a machine or plant means to monitor the production process and intervene in the process whenever necessary. SIMATIC HMI (Human Machine Interface) provides the necessary devices and tools.

The SIMATIC HMI device families meet all requirements placed on HMI devices, from simple Key Panels and HMI devices with touch-sensitive screens in various sizes for machine-level operator control and monitoring all the way to SCADA systems (Supervisory Control And Data Acquisition) for process control and process monitoring in distributed multi-user systems with redundant servers and web client solutions across locations (Fig. 7.1).

Depending on the design, the HMI devices are connected with the programmable controller via the MPI, PROFIBUS, or Industrial Ethernet bus systems.

WinCC inside TIA Portal, WinCC flexible, and the SCADA system WinCC are available as HMI engineering software for configuring the HMI devices. PC-based HMI devices such as the Panel PC run visualization software that enables operator control and monitoring of the process. Numerous option packages expand the basic functionality of the engineering and visualization software.

This chapter describes a selection of available HMI devices.

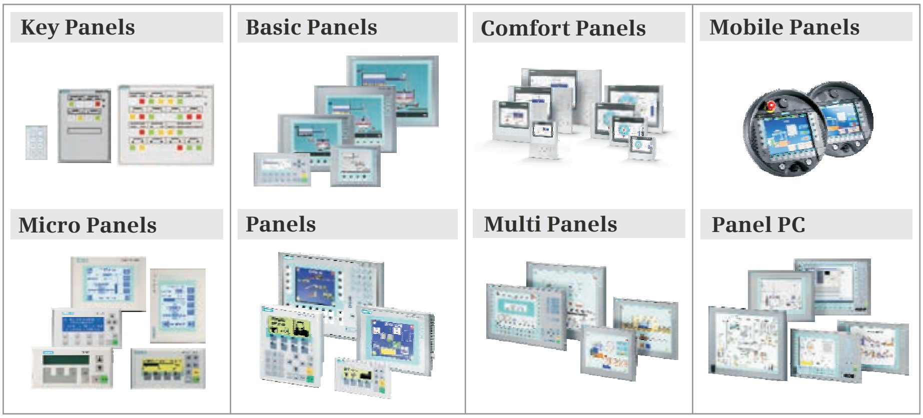


Fig. 7.1 Overview of SIMATIC HMI device families

7.1 Key Panels KP8, PP7 and PP17

Key Panels (KP) and Push Button Panels (PP) are the innovative alternative to conventionally wired key panels. Supplied pre-assembled and ready for installation, the bus-compatible operator panels are the key to drastically reducing wiring times when compared with conventional methods. Fig. 7.2 provides a graphic depiction of the Key Panel KP8 and the two Push Button Panels PP7 and PP17.

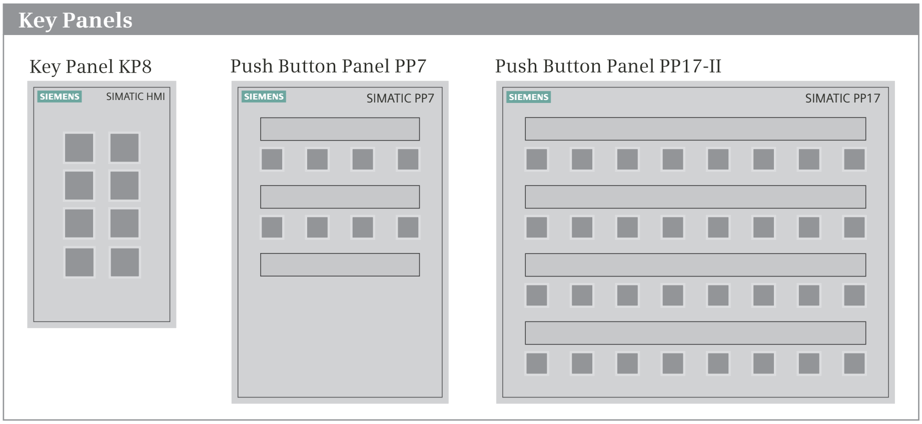


Fig. 7.2 Key Panel KP8/KP8F, Push Button Panels PP7 and PP17-II

Key Panels

Key Panels are pre-assembled key panels for simple machine operation. They feature large illuminated pushbuttons with good tactile feedback, which can even be operated with gloves and are therefore suitable for harsh industrial environments. The buttons have LED backlighting that can be adjusted in terms of brightness and color (red, yellow, green, blue, white). All keys can be individually labeled using slide-in labels.

The connection to the control is implemented via PROFINET. The connection consists of two RJ45 sockets that are interconnected by an integrated switch and which allow the construction of a linear structure without additional module.

Only a rectangular cutout is required for installation. The degree of protection is IP 65 for the front when installed and IP 20 for the rear. Digital inputs/outputs are available on the rear for expansion. The F-variant provides the ability to connect a rear emergency stop button according to SIL 3 or SIL 2.

Key Panel KP8 PN: 8 large mechanical illuminated pushbuttons, 8 freely configurable digital inputs/outputs for the connection of further operator controls, e.g. keyswitches.

Key Panel KP8F PN: 8 large mechanical illuminated pushbuttons, 8 freely configurable digital inputs/outputs for the connection of further operator controls, e.g. keyswitches, 2 additional fail-safe digital inputs for the connection of, for example, emergency stop buttons.

Key Panel KP32F PN: 32 large mechanical illuminated pushbuttons, 16 freely configurable digital inputs/outputs and 16 additional digital inputs for the connection of further operator controls, e.g. keyswitches, 4 additional fail-safe digital inputs for the connection of, for example, emergency stop buttons.

Push Button Panels

Push Button Panels are pre-assembled operator panels for simple machine operation. They feature short-stroke keys in different numbers depending on the design. These keys can be labeled and have built-in, two-color surface LEDs. The connection to the controller is implemented through a serial interface, either via an MPI for a direct connection to the CPU's programming device interface, or as DP standard slave for a PROFIBUS DP connection to any DP masters.

Only a rectangular cutout is required for installation. The degree of protection is IP65 on the front and IP20 on the rear. Digital inputs/outputs are available, for example to allow additional 22.5 mm standard elements such as pushbuttons and lamps to be inserted in the perforated cutouts of the standard version. Push Button Panels can be used immediately without parameter assignment; they are factory-set to the MPI address 10, and memory byte 100 and higher is set for the pushbuttons and LEDs. The interface parameters can be changed without tools on a rear display.

Although already furnished with various functions, such as an integrated lamp and pushbutton test, or 0.5 Hz and 2 Hz flashing frequencies, the Pushbutton Panels PP7 and PP17 can also be tailored to meet your specific requirements with regard to number and layout of the display and operating elements and labeling.

If Push Button Panels are used as DP slaves, parameter assignment is performed in HW Config in STEP 7 as with an ET200 station. All parameters are stored on a memory module, which can be easily exchanged. For diagnostic purposes, all operating modes are shown on the rear display.

Push Button Panel PP7: 8 short-stroke keys, 8 LEDs, 4 additional digital inputs, 3 perforated cutouts for additional 22.5 mm standard elements such as lamps, pushbuttons, etc.

Push Button Panel PP17-I: 16 short-stroke keys, 16 LEDs, 16 additional digital inputs and 16 additional digital outputs, 12 perforated cutouts for additional 22.5 mm standard elements such as lamps or pushbuttons, also available as SIPLUS version.

Push Button Panel PP17-II: 32 short-stroke keys, 32 LEDs, 16 additional digital inputs and 16 additional digital outputs, also available as SIPLUS version.

7.2 Basic Panels

Basic Panels offer basic HMI functionality for small machines and applications. They are available in size 3” as a pure Key Panel (KP), from 4” to 12” with touch screen and additional keys (KTP), and as a pure touch device in the size 15”. Variants can be selected for connection to PROFINET/Ethernet or PROFIBUS DP/MPI. Fig. 7.3 shows the KTP 400 and the KTP 1000.

Basic Panels are HMI devices for the lower performance range. The PROFINET versions of the Basic Panels are equipped with an RJ45 terminal for Industrial Ethernet, and the PROFIBUS versions with an RS422/485 interface for 12 Mbit/s. Depending on the device version, Basic Panels have a touch screen and function keys. The number of function keys depends on the device version (Table 7.1).

The degree of protection achieved when installed is IP65 for the front and IP20 for the rear. The supply voltage for the Basic Panels is 24 V DC. All devices have an unbuffered real-time clock.

Despite the functionality tailored to basic applications, the Basic Panels offer numerous HMI functions as standard: The signaling system can consist of max. 200 discrete alarms and 15 analog alarms. The message text can be up to 80 characters long and can contain up to 8 variables. The message buffer can hold 256 messages. 250 variables (KP 300, KTP 400) and 500 variables (KTP 600, KTP 1000, TP 1500) are available.

Up to 50 process pictures can be configured, where each picture can contain 30 arrays, 30 variables, and 30 complex objects such as bars. Up to 5 recipes can be used. Of the 32 configurable languages, 5 can be switched online. For protection against unauthorized operation, 32 authorizations can be issued to up to 50 users in 50 user groups.

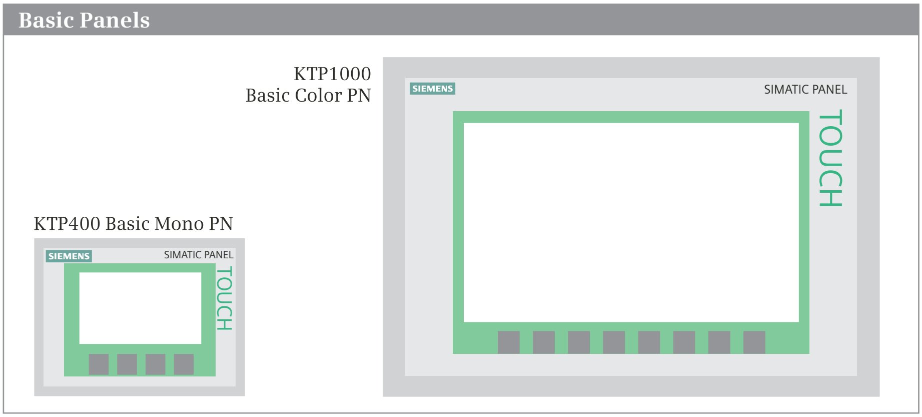


Fig. 7.3 Basic Panels KTP 400 Basic mono PN and KTP 1000 Basic color PN

Table 7.1 Selected technical data of Basic Panels

	KP 300 Basic mono PN	KTP 400 Basic mono PN	KTP 600 Basic mono PN	KTP 600 Basic color PN or DP	KTP 1000 Basic color PN or DP	TP 1500 Basic color PN
Display size	3.6″	3.8″	5.7″	5.7″	10.4″	15″
Resolution, pixels	240 × 80	320 × 240	320 × 240	320 × 240	640 × 480	1024 × 768
Colors	black/white	4 gray levels	4 gray levels	256 colors	256 colors	256 colors
Touch screen	No	Yes	Yes	Yes	Yes	Yes
Function keys	10	4	6	6	8	No
User memory	512 KB	512 KB	512 KB	512 KB	1024 KB	1024 KB

WinCC Basic inside TIA Portal can be used as engineering software for Basic Panels. WinCC Basic is supplied together with STEP 7 Basic/Professional inside TIA Portal.

7.3 Comfort Panels

Comfort Panels offer high-end functionality for implementing automation solutions. The devices are available in versions with 4”, 7”, 9”, and 12” with touch screen (TP) or with keys (KP), as well as 4” version with additional keys (KTP). All function keys are equipped with LED and tactile feedback for additional reliability of operation. Fig. 7.4 shows the KP 400 Comfort Panel and KPT 400 Comfort Panel.

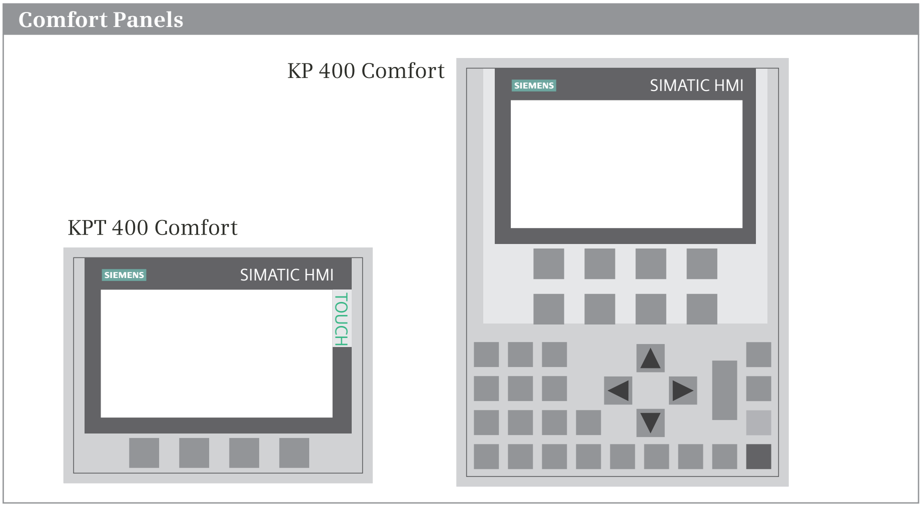


Fig. 7.4 Comfort Panels KPT 400 Comfort and KP 400 Comfort

The Comfort Panels replace the corresponding predecessor units (see Table 7.2). The high-resolution widescreen display with 16 million colors offers up to 40% more visualization area. The LED backlighting is dimmable from 0 to 100% via PROFIenergy, via the HMI project, or via the PLC. All touch devices can also be installed upright. The degree of protection is IP65 on the front when installed. The degree of protection is IP20 on the rear side.

Table 7.2 Selected technical data of the Comfort Panels

	KP 400 Comfort	KTP 400 Comfort	TP 700 Comfort	KP 700 Comfort
Display size	4.3″	4.3″	7.0″	7.0″
Resolution, pixels	480 × 272	480 × 272	800 × 480	800 × 480
Function keys *)	8	4	–	24
User memory	4 MB	4 MB	12 MB	12 MB
Successor to	OP 77B	TP 177B 4″	TP 177B, TP 277, MP 177 6″	OP 177B OP 277 6″
	TP 900 Comfort	**KP 900 Comfort**	**TP 1200 Comfort**	**KP 1200 Comfort**
Display size	9.0″	9.0″	12.1″	12.1″
Resolution, pixels	800 × 480	800 × 480	1280 × 800	1280 × 800
Function keys *)	–	26	–	34
User memory	12 MB	12 MB	12 MB	12 MB
Successor to	MP 277 8″ Touch	MP 277 8″ Key	MP 277 10″ Touch	MP 277 10″ Key

*) programmable, with LED

The Comfort Panels are equipped with PROFINET/Ethernet as well as PROFIBUS DP/MPI interfaces. The PROFINET version with 7” display has two ports with integrated network switch.

The Comfort Panels offer numerous HMI functions as standard (information structured according to device versions 4”/7”, 9”, 12”): The signaling system can hold up to 2000/4000 messages. A retentive message buffer can hold 256/1024 messages. Up to 500/500 process pictures and up to 100/300 recipes can be configured. 1024/2048 variables are available.

The configuration software for Comfort Panels is WinCC (Comfort, Advanced or Professional) inside TIA Portal.

7.4 Mobile Panels

A Mobile Panel is a mobile HMI device for direct operation of plants and machines from any connection box. By reconnection during operation or by radio transmission (Mobile Panel 277(F) IWLAN), they can be taken to any location where direct visual contact with the workpiece or process is required. Depending on the connection point selected, it is possible to enable or block operator actions or privileges.

The buttons and the other display elements and operator controls such as LEDs or function keys can be configured as additional I/Os. For example, activation of an operator control can result in direct setting of an input bit in the central controller via the bus system, or an LED on the Panel can be triggered as an output. Further operator controls – depending on the version – are two 3-stage enabling buttons, optionally a positively latching STOP pushbutton, as well as a handwheel and keyswitch.

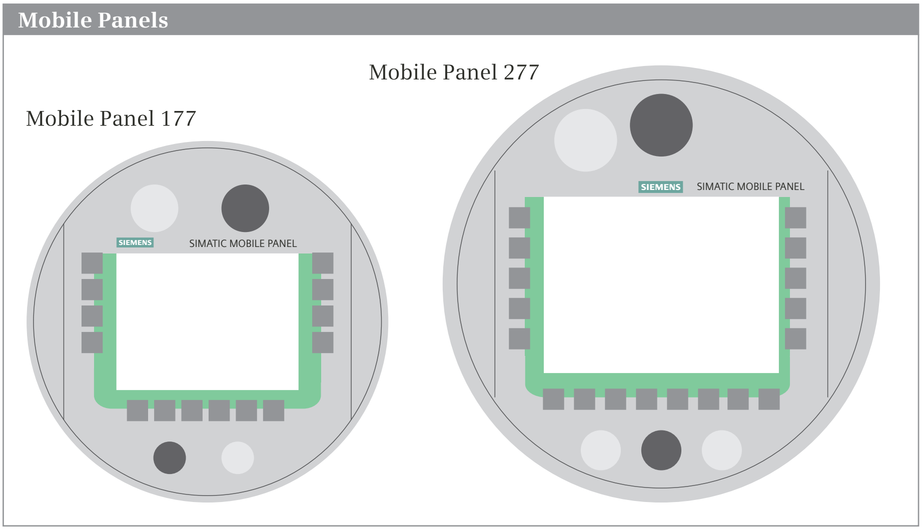


Fig. 7.5 Mobile Panel 177 and Mobile Panel 277

STOP and acknowledgement buttons are designed with dual circuits according to the safety regulations and comply with the requirements of Safety Category 3 in accordance with EN 954-1. The operator controls of the Mobile Panel 277F IWLAN have a failsafe design with use of PROFIsafe. The STOP pushbutton can be incorporated into the emergency stop circuit of the machine or plant by means of the connection boxes. It supplements, but does not replace, the emergency stop equipment according to EN 418, which is permanently installed in the plant. When disconnecting the Mobile Panel, the connection boxes “Plus” prevent an emergency stop by automatically closing the emergency stop circuit.

The Mobile Panels are available in connection versions for MPI/PROFIBUS DP with a transmission rate of max. 12 Mbit/s and for PROFINET IO with 10/100 Mbit/s. Configuration is carried out using WinCC flexible (Compact, Standard, or Advanced) or with WinCC V11 inside TIA Portal (Comfort, Advanced, or Professional).

With the **Mobile Panel 177**, operation is carried out using a pixel graphics 5.7” STN display with touch screen. The resolution is 320 × 240 pixels with 256 colors. Additional control options are provided by the 14 programmable, function key with user-definable labels, of which 8 have green LEDs.

With the **Mobile Panel 277**, operation is carried out using a pixel graphics 7.5” TFT display with touch screen. The resolution is 640 × 480 pixels with 65 536 colors. 18 programmable function keys with user-definable label with LEDs are additionally available. The version with the 10” TFT display has a resolution of 800 × 600 pixels, and the absent function keys are replaced with programmable keys.

7.5 Micro Panels

Designed specifically for applications with the SIMATIC S7-200 Micro PLC, there are control and display units that allow you, for example, to display message texts, set outputs, and control small machines or plants. On the front, the degree of protection is IP65 when installed; on the rear side it is IP20. The TD 200 and TD 400C Text Displays and the OP 73micro and TP 177micro devices with graphic display are available. Fig. 7.6 shows the TD 200 Text Display and the OP 73micro Operator Panel.

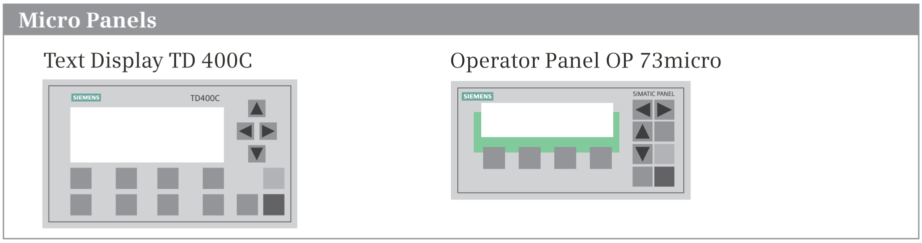


Fig. 7.6 TD 400C Text Display and OP 73micro Operator Panel

The **TD 200** Text Display has two lines with 20 characters each (ASCII, Cyrillic) or 10 characters each (Chinese). It can display up to 80 message texts with up to 6 variables. One memory bit, which can be polled in the user program, is assigned to each of the 8 programmable function keys. The configuration data of the TD 200 is saved in the CPU 200. The TD 200 is configured using STEP 7 Micro/WIN.

The **TD 400C** Text Display features a 3.7” STN display (black/white) with a resolution of 192 × 64 pixels, on which up to 4 lines of text can be configured. It can display up to 80 message texts with up to 6 variables. The 15 keys can be assigned numerous functions. One memory bit, which can be polled in the user program, is assigned to each of the keys. The configuration data of the TD 400C is saved in the CPU 200. The TD 400C is configured using STEP 7 Micro/WIN.

The **OP 73micro** Operator Panel has a monochrome 3” LCD with a resolution of 160 × 48 pixels as well as eight system keys and four freely programmable function keys. The OP 73micro is configured with WinCC flexible.

The **TP 177micro** Touch Panel has a 5.7” STN touch screen with four blue levels and a resolution of 320 × 240 pixels. The TP 177micro is configured with WinCC flexible.

7.6 SIMATIC Panels – Series 70

The SIMATIC Panels of the Series 70 comprise the Operator Panels OP 73, OP 77A, and OP 77B, which replace the text-based Operator Panels OP 3 and OP 7.

The Operator Panels have IP65 degree of protection at the front, which makes them suitable for use at the machine level. Current faults, operating modes, and process values can be monitored and controlled using the LED-backlit LC display and a membrane keyboard which is resistant to various oils, greases and standard detergents. The mounting dimensions correspond to OP 3 (OP 73) or OP 7 (OP 77A/B).

The panels of the Series 70 are configured using WinCC flexible. 32 languages including Asian and Cyrillic fonts are available. Up to 5 languages can be selected online; language-dependent texts and graphics can be implemented. The message system with bit messages can manage up to 500 (OP 73) or 1000 messages (OP 77). You can configure up to 500 process displays with text and graphic objects, fields and variables. OP 77B has a recipe management function for up to 100 recipes. Access protection is provided with passwords and password levels to prevent unauthorized operations.

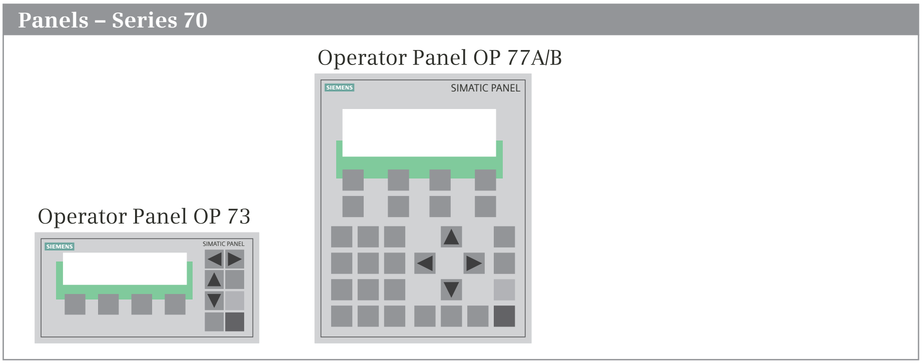


Fig. 7.7 Operator Panels OP 73 and OP 77A/B

You can connect the panels of the Series 70 to S7-200 (PPI interface) or S7-300/400 (MPI interface). Operation is also possible on PROFIBUS DP at 1.5 Mbit/s or 12 Mbit/s (OP 77B).

The **Operator Panel OP 73** has 8 system keys and 4 freely configurable function keys. The pixel graphics, monochrome 3” STN LC display has a resolution of 160 × 48 pixels. A 256 KB flash memory is integrated for the configuration data.

The **Operator Panel OP 77A** has 23 system keys, 8 freely configurable function keys with user-definable label, 4 of them with LED. The pixel graphics, monochrome 4.5” STN LC display has a resolution of 160 × 64 pixels. A 256 KB flash memory is integrated in the device for the configuration data.

The **Operator Panel OP 77B** has the same design as the OP 77A, but it has a larger user memory of 1000 KB for configuration data. A USB port and an SD/multimedia card slot are available as additional interfaces.

7.7 SIMATIC Panels – Series 170

The Panels of the Series 170 with IP65 degree of protection on the front allow use at the machine level. Current faults, operating modes, and process values can be monitored and controlled using the pixel graphics STN Liquid Crystal Display (LCD) and a membrane keyboard which is resistant to various oils, greases and standard detergents. The TP 177B PN/DP INOX is also available with a stainless steel front (DIN EN 1672-2). The degree of protection on the installation side is IP20. The 5.7” display has a resolution of 320 × 240 pixels, and is available with 256 colors or 4 blue levels depending on the panel. TP 177B is also available with a 4.3” TFT display with 256 colors and a resolution of 480 × 272 pixels.

The operating system is Windows CE. The Panels of the Series 170 are configured using WinCC flexible. 32 languages can be configured offline – including Asian

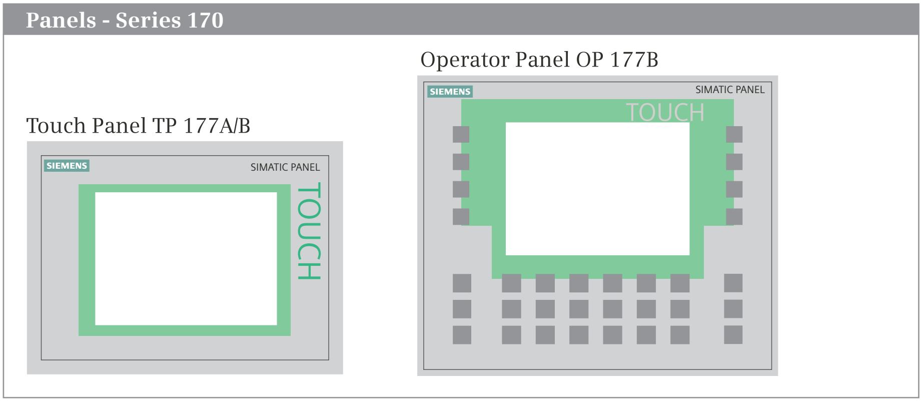


Fig. 7.8 Touch Panel TP 177A/B and Operator Panel OP 177B

and Cyrillic languages – thus facilitating global application of the devices. Up to 16 languages can be switched online during operation.

A user administration function allows authentication depending on process requirements through application of user names and passwords with assignment of user-specific privileges.

The **Touch Panels TP 177A and TP 177B** permit operation directly via the screen. A touch-sensitive screen replaces the keyboard; wherever you have configured a button in the display, the operator can trigger actions by touching it. A mouse and keyboard can be connected via a USB port on the TP 177B.

The display of the **Operator Panel OP 177B** is designed as touch screen with configurable system keys. Additionally, there are freely configurable and inscribable 32 function keys, of which 26 with LED. An external mouse and keyboard can be connected via a USB port.

7.8 SIMATIC Panels – Series 270

The Panels of the Series 270 with IP 65 protection at the front allow use at machine level. Monitoring of faults, operating modes and process values, and appropriate operator actions, are carried out on a TFT LCD and a membrane keyboard which is resistant to various oils, greases and the usual cleaning agents. The degree of protection on the installation side is IP 20. The 5.7” display has 256 colors with a resolution of 320 × 240 pixels.

The operating system is Windows CE. The Panels of the Series 270 are configured using WinCC flexible. The user RAM is 4 MB. Up to 2000 or 4000 messages, 300 recipes and 500 process graphics can be configured. A maximum of 2048 variables are available.

32 languages can be configured offline – including Asian and Cyrillic languages – thus facilitating global application of the devices. Up to 16 languages can be switched online during operation. A user administration function allows authentication depending on process requirements through application of user names and passwords with assignment of user-specific privileges.

RS422, RS485 (MPI, PROFIBUS DP with up to 12 Mbit/s), USB (for connecting an external mouse or keyboard), and Ethernet (RJ45) are available as interfaces. Together with the ProAgent process fault diagnostics function, faults occurring in the plant can be identified faster and downtimes can thus be minimized. If a process fault occurs, ProAgent provides information on the location and cause of the fault – also with the support of predefined standard displays.

The **Touch Panel TP 277** is operated directly via the screen. A touch-sensitive screen replaces the keyboard; wherever you have configured a button in the display, the operator can trigger actions by touching it.

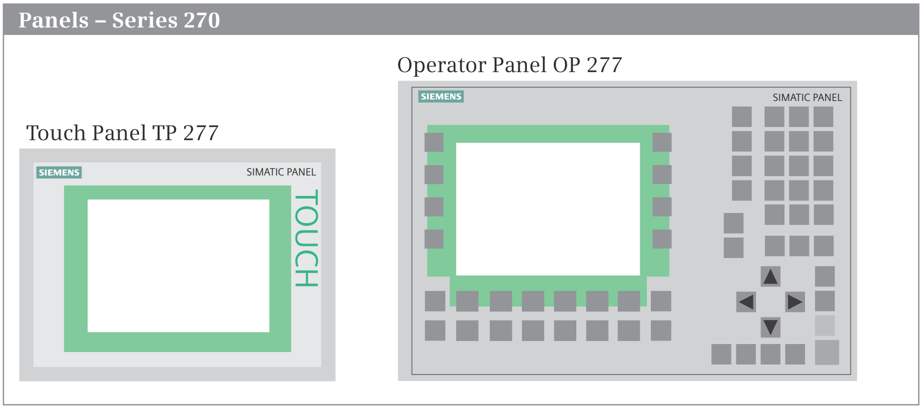


Fig. 7.9 Touch Panel TP 277 and Operator Panel OP 277

The **Operator Panel OP 277** is operated using a membrane keyboard. There are 36 system keys and 24 freely configurable function keys with user-definable label, 18 of which have LEDs.

7.9 Multi Panels

The multifunctional platforms combine the ruggedness of a machine-level HMI device with the flexibility of a personal computer. Multi Panels have degree of protection IP65 on the front, they have no hard disks or fans so that they can be used in harsh industrial environments. Short ramp-up times mean they are quickly ready for use after power-on. Each Multi Panel is available in two versions: with touch screen and with membrane keyboard.

The following on-board interfaces are available: RS422, RS485 (MPI, PROFIBUS DP up to 12 Mbit/s), USB, and Ethernet (RJ45), and also TTY for the MP 370. A slot for an SD/MultiMedia card is also present.

The operating system of the Multi Panels is Windows CE. The panels are configured with WinCC flexible. Optimized versions of the WinAC MP 2007 Software PLC are available for MP 277 and MP 377.

The **Multi Panel MP 177 Touch** has a pixel graphics 5.7” TFT display with a color depth of 64k colors and a resolution of 320 × 240 pixels.

The **Multi Panel MP 277 Key** has a pixel graphics 7.5” or 10.4” TFT display with a color depth of 64k colors and a resolution of 640 × 480 pixels. Additionally, there are 38 system keys, 26 or 36 configurable function keys with user-definable label, 18 or 28 of which with LEDs.

The **Multi Panel MP 277 Touch** has a pixel graphics 7.5” or 10.4” TFT display with a color depth of 64k colors and a resolution of 640 × 480 pixels.

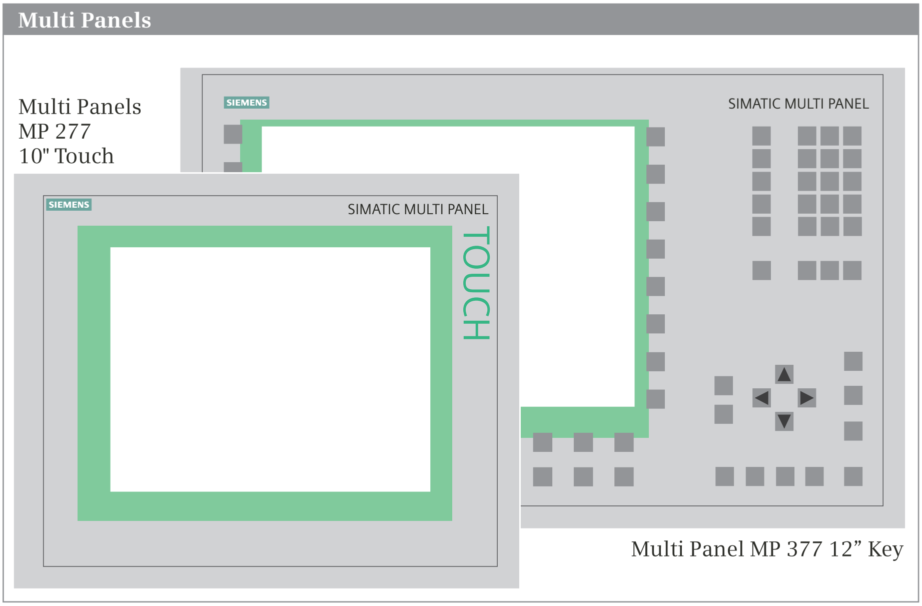


Fig. 7.10 Multi Panels MP 277 10” Touch and MP 377 12” Key

The **Multi Panel MP 377 Key** has a pixel graphics 12.1” TFT display with a color depth of 64k colors and a resolution of 800 × 600 pixels. Additionally, there are 38 system keys, 36 configurable function keys with user-definable label with LEDs.

The **Multi Panel MP 377 Touch** is available in the versions 12” (12.1” TFT display with a resolution of 800 × 600 pixels), 15” (15.1” TFT display with a resolution of 1024 × 768 pixels), and 19” (19” TFT display with a resolution of 1280 × 1024 pixels) with a color depth of 64k colors. The MP 377 PRO 15” Touch is equipped with a robust and very compact aluminum enclosure and is fully enclosed in accordance with IP65.

7.10 SIMATIC Panel PC

The SIMATIC Panel PCs are used in both the manufacturing industry and process industry. They are designed for installation in control cabinets or control panels. The degree of protection when installed is IP65 on the front. The Panel PCs have a high level of electromagnetic compatibility (EMC) and vibration and shock immunity, and they are designed for continuous 24-hour operation. On some device types, the computing unit and the operating unit can also be operated when physically separated. An optional direct key module enables direct process operation via PROFIBUS DP independently of the operating system.

A SIMATIC Panel PC meets the requirements of Totally Integrated Automation and forms the ideal platform for PC-based Automation, whether as a SIMATIC WinAC controller, as a process visualization system at machine level with WinCC flexible, or for complex HMI solutions with WinCC. A Panel PC consists of an operating unit and a computing unit. The Panel PCs are either available without operating system or with a pre-installed operating system Windows XP embedded Standard 2009, Windows XP Professional Multi-Language or Windows embedded Standard 7 or Windows 7.

The **Panel PC IPC 277D** is a particularly compact and energy-efficient Nanopanel PC with the front installation versions 7” Touch (800 × 480 pixels), 9” Touch (800 × 480 pixels), and 12” Touch (1280 × 800 pixels). The versions 15” Touch (1280 × 800 pixels) and 19” Touch (1366 × 768 pixels) will be available soon. The TFT display has a color depth of 16 million colors and is dimmable from 0 to 100%.

The processor is an Intel Atom E660 1.3 GHz and 2 GB of RAM, or alternatively, an Intel Atom E640 1.0 GHz and 1 GB of RAM. The following interfaces are available: 2 × LAN 10/100/1000 Mbit/s Ethernet interface, 3 × high-speed USB V2.0 and 1 × COM1 (RS232).

The **Panel PC IPC 477C** is designed for use directly on the machine, in which the combination of ruggedness and reliability is of major priority and the openness of a PC is also required. The operating unit is available in the following versions: 12” Touch/Key (800 × 600 pixels/36 function keys), 15” Touch/Key (1024 × 768 pixels/

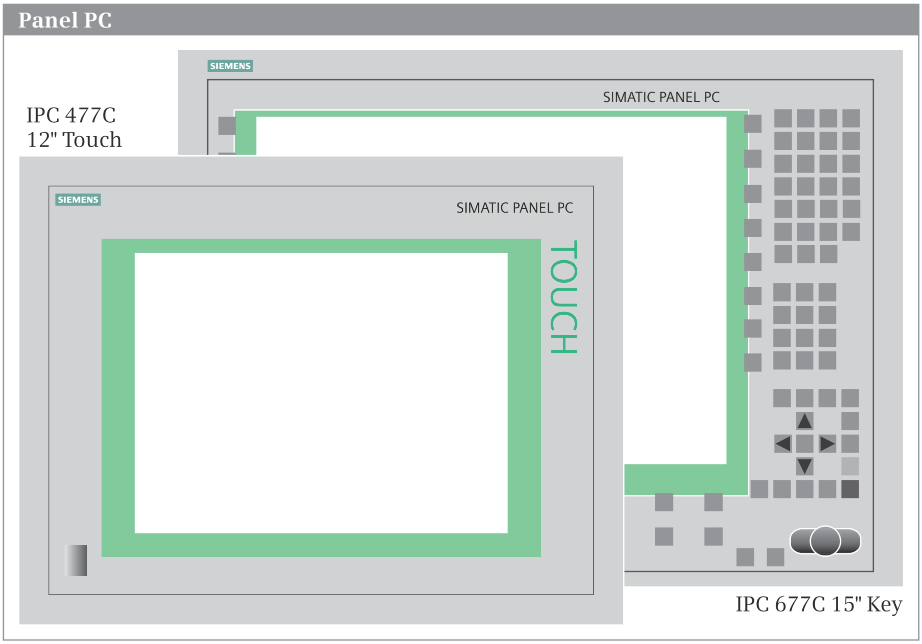


Fig. 7.11 Panel PC IPC 477C 12” Touch and IPC 677C 15” Key

36 function keys), and 19” Touch (1280 × 1024 pixels). For mounting on a support arm of stand, the PRO 15”/19” Touch version (1024 × 768 pixels/ 1280 × 1024 pixels) is available as fully enclosed device with degree of protection IP65. The TFT display has a color depth of 16 million colors.

The processor is an Intel Celeron M 1.2 GHz, an Intel Core2 Solo 1.2 GHz, or an Intel Core2 Duo 1.2 GHz. The main memory can be 1, 2, or 4 GB DDR3 RAM. The following interfaces are available: 2 × PROFINET onboard (optionally 2 × PROFINET onboard and 1 × PROFIBUS onboard or 1 × PROFINET onboard and 1 × PROFINET with 3 ports), 5 × high-speed USB V2.0 (of which 1 × on the front), 1 × COM1 (RS232) and 1 × DVI-I for connecting a second display unit.

The **Panel PC IPC 577C** is designed for installation in control cabinets and consoles; due to its minimal mounting depth it can also be used in configured spaces. The operating unit is available in the following versions: 12” Touch/Key (800 × 600 pixels/36 function keys), 15” Touch/Key (1024 × 768 pixels/36 function keys), and 19” Touch (1280 × 1024 pixels). The TFT display has a color depth of 16 million colors.

The processor is an Intel Celeron M 1.2 GHz, an Intel Core2 Solo 1.2 GHz, or an Intel Core2 Duo 1.86 GHz. The main memory can be 1, 2, or 4 GB DDR3 RAM. The following interfaces are available: 2 × PROFINET onboard (optionally 2 × PROFINET onboard and 1 × PROFIBUS onboard or 1 × PROFINET onboard and 1 × PROFINET with 3 ports), 5 × high-speed USB V2.0 (of which 1 × on the front) and 1 × COM1 (RS232).

The **Panel PC IPC 677C** is designed for installation directly at the machine; due to its minimal mounting depth it can also be used in configured spaces. The operating unit is available in the following versions: 12” Touch/Key (800 × 600 pixels/36 function keys), 15” Touch/Key (1024 × 768 pixels/36 function keys), and 19” Touch (1280 × 1024 pixels). The TFT display has a color depth of 16 million colors.

The processor is an Intel Celeron P4505 1.86 GHz, an Intel Core i3-330E with 2.13 GHz, or an Intel Core i7-610E 2.53 GHz. The main memory is 1 GB, expandable up to 8 GB. The following interfaces are available: 2 × PROFINET onboard, 1 × PROFINET onboard with 3 ports and 1 × PROFIBUS onboard, 4 × high-speed USB V2.0 (of which 1 × on the front) and 1 × COM1 (RS232).

7.11 Configuring SIMATIC HMI

SIMATIC WinCC (Windows Control Center) is the engineering and runtime software for HMI devices and HMI applications. WinCC flexible is used for configuring HMI devices for the machine-level area and PC-based single-user systems, with WinCC also PC-based multi-user systems with SCADA functionality (supervisory control and data acquisition) for monitoring and process control in industrial processes. The WinCC version for the Totally Integrated Automation Portal (TIA Portal) engineering framework covers both areas.

General functions of the engineering software

All configuration data is saved in one project. With WinCC in standalone mode, you create an HMI project, add an HMI station (an operator panel), and configure the operator control and monitoring functions. If WinCC flexible is integrated in STEP 7 or if you use WinCC inside TIA Portal, create a project with the SIMATIC Manager or with the TIA Portal and then add the HMI station. A wizard will guide you through all the necessary steps.

All functionalities are available to the user interface that are supported by the selected HMI station. In WinCC there is a special editor available for each configuration task, e.g. to create a image or a message. With a graphical editor, you open an object in the working area and edit it, e.g. to change the color or the motion sequence of a graphic. A tabular editor displays all similar objects in a table, e.g. all variables, whose properties you then edit in the table.

With WinCC, you create the variables that you use when configuring the HMI station (internal variables), or the variables for the interface to the PLC station (external variables).

You can configure displays with which the process is controlled and monitored. A display can consist of static and dynamic objects, for example, text, numerical values, and trend and bar charts that change depending on process variables. Function keys or configured buttons make it possible to switch to another display. You can use image layers to set the nesting depth of the display. WinCC provides libraries with pre-configured image objects.

The signaling system has system-defined messages that do not require any configuration and user-defined messages that can be configured as analog alarms (upward or downward violation of limits) or discrete alarms (display of statuses or status changes). Messages are assigned to specific alarm classes – such as errors and warnings – whose representation and acknowledgment concept can be configured.

Recipes contain related data, such as for a specific batch in production. A recipe consists of recipe data records, which in turn consist of recipe elements. The recipe data may already be entered during configuration. It is stored in the HMI station and can be changed, amended, or deleted during runtime. It is possible to exchange recipe data between HMI and PLC station.

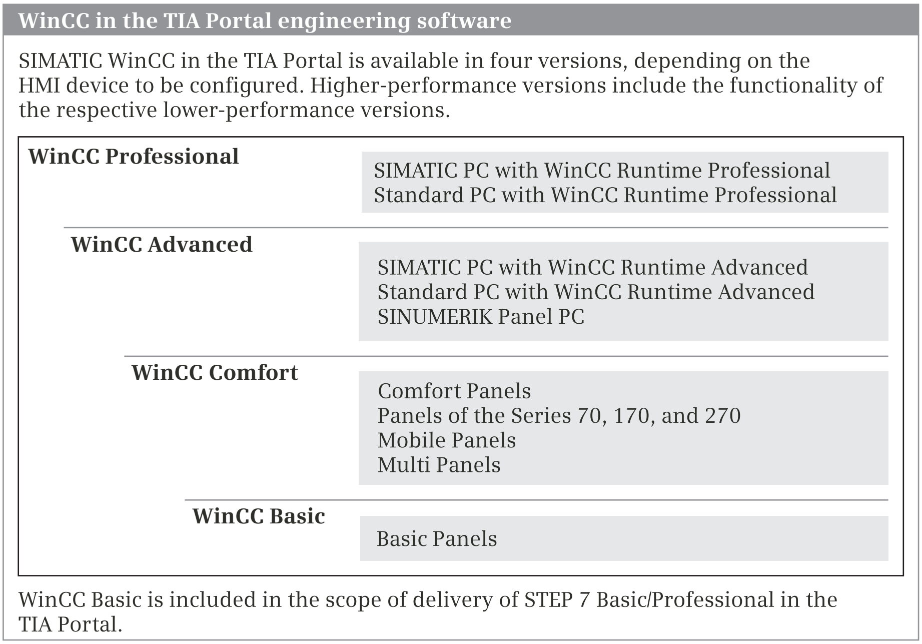
You configure user groups, users, and authorizations with the user administration. Authorizations restrict security-related operations to specific user groups. To do this, you set up access protection for an operating element. Only users who have been assigned to a user group with the appropriate access rights may use this object for operation. For operation at runtime, you determine how many invalid login attempts are allowed and the interval after which the password must be changed. Users and user groups can also be managed at runtime on the HMI device.

WinCC enables you to configure a project in multiple languages. First, you configure the texts in the reference language, then translate them into the desired project languages with the texts of the reference language as a template – for example, through exporting the reference texts and importing the translated texts. You can then switch to the desired language (editing language) during configuration. Runtime languages are those project languages that you transfer to the HMI devices. The maximum number of languages depends on the HMI device. At runtime, the language can be changed on the HMI device.

WinCC inside TIA Portal engineering software

WinCC inside TIA Portal is available in the versions Basic, Comfort, Advanced, and Professional (Fig. 7.12). The engineering tools are largely determined by the configurable target devices, where the more powerful engineering software incorporates the abilities of the underlying software. The upgrade to a more powerful version is available with a PowerPack, except for WinCC Basic.

Integration inside TIA Portal provides the same working environment for the configuration of HMI stations and programming of PLC stations. This means common symbols, common data management, and system diagnostics as an integral component. The engineering interface has the same appearance for HMI and PLC stations and thus increases the ease of use. There is no longer a distinction between HMI and PLC projects. HMI and PLC stations (and even PC-based stations) are in one project and can be networked by means of communication connections that are configured once.



WinCC in the TIA Portal engineering software

SIMATIC WinCC in the TIA Portal is available in four versions, depending on the HMI device to be configured. Higher-performance versions include the functionality of the respective lower-performance versions.

WinCC Professional	SIMATIC PC with WinCC Runtime Professional Standard PC with WinCC Runtime Professional
WinCC Advanced	SIMATIC PC with WinCC Runtime Advanced Standard PC with WinCC Runtime Advanced SINUMERIK Panel PC
WinCC Comfort	Comfort Panels Panels of the Series 70, 170, and 270 Mobile Panels Multi Panels
WinCC Basic	Basic Panels

WinCC Basic is included in the scope of delivery of STEP 7 Basic/Professional in the TIA Portal.

Fig. 7.12 Available versions of WinCC V11 inside TIA Portal

The simulator enables the HMI device to be simulated directly on the configuration device. Either you observe the behavior of the HMI device after you have specified variable values, or the configuration device communicates directly with a PLC station. This can be a “real” PLC station or one simulated with PLCSIM on the configuration device.

SIMATIC WinCC (TIA Portal) V11 SP2 is executable under Windows XP Professional SP3 (32-bit) and Windows 7 Professional/Ultimate/Enterprise (32-bit). WinCC Basic is also executable under Windows XP Home SP3 and Windows 7 Home. WinCC Advanced and WinCC Professional can also be operated under Windows Server 2003 R2 Standard Edition SP2 and Windows Server 2008 Standard Edition SP2. To work with WinCC inside TIA Portal, a user authorization (license) is required. The configuration data from the project created with WinCC flexible can be imported by means of a simple migration.

Options for WinCC inside TIA Portal

WinCC options (TIA Portal) allow you to expand the basic functionality of WinCC Runtime. A special license is required for each option. The WinCC engineering system already contains the functionality of the Runtime options. You do not need a license on your configuration PC to configure the functionality of a runtime option.

The following options are available for WinCC Runtime Advanced:

- ▷ *WinCC Audit* permits the recording of operator actions in an Audit Trail. The operator actions to be recorded can be selected according to whether an electronic signature or comment is required during runtime.
- ▷ *WinCC Sm@rtServer* is used for remote maintenance and servicing over the Internet. An integral Web server provides standard HTML pages with which, for example, recipe data records with password protection can be accessed or configuration data downloaded to the controller.

The following options are available for WinCC Runtime Advanced and WinCC Runtime Professional:

- ▷ *WinCC Recipes* generates and manages machine parameters and production data on the basis of data records and controls exchange with the PLC. Data records can also be imported and exported as CSV files, e.g. for Microsoft Excel.
- ▷ *WinCC Logging* is used to archive messages and process values. The evaluation of message and process value archives can, for example, be performed with Microsoft Excel.
- ▷ *WinCC ControlDevelopment* extends the basic functionality with own controls, which are developed using VB.net or C#.
- ▷ *SIMATIC Logon* creates user administration on a central computer for one or more WinCC stations that are connected via an Ethernet network, and checks each user logon and logoff at one of the connected stations.

The following options are available for WinCC Runtime Professional:

- *WinCC Server* and *WinCC Client* permit the setup of a powerful client/server system. In a common network of PLC and HMI stations, a server provides the connected clients with process and archive data, messages, images, and reports.
- *WinCC WebNavigator* allows the operation and monitoring of plants via the Internet or corporate intranet using an Internet browser with ActiveX support.
- *WinCC DataMonitor* allows the display and evaluation of current process states and saved data on an office PC with standard tools.

SIMATIC WinCC Runtime inside TIA Portal visualization software

WinCC Runtime is a PC-based operator control and monitoring solution and is supplied in the versions *Advance* for single-user systems at the machine level and *Professional* for distributed multi-user systems and multi-site solutions with web clients. The basic packages are expandable by means of option packages. It is also possible to include customer-specific ActiveX controls created using WinCC ControlDevelopment.

SIMATIC WinCC flexible ES engineering software

WinCC flexible ES is the engineering software for HMI applications at the machine level in plant and mechanical engineering. WinCC flexible is available in the versions Micro, Compact, Standard, and Advanced (Fig. 7.13). The engineering software is largely determined by the configurable target devices, where the more powerful engineering software incorporates the abilities of the underlying software. Except for WinCC flexible Micro, the tools can be upgraded to the next level with a PowerPack.

With WinCC flexible you can create and edit HMI projects, either as stand-alone version or in the SIMATIC Manager of STEP 7 V5.5. The latter has the advantage of a common data management, simplified connection configuration, and simpler commissioning.

The simulator enables the HMI device to be simulated directly on the configuration device. Either you observe the behavior of the HMI device after you have specified variable values, or the configuration device communicates directly with a PLC station. If WinCC flexible is integrated in STEP 7, you can also simulate a PLC station with the PLCSIM option package.

SIMATIC WinCC flexible 2008 SP3 is executable under Windows XP Professional SP3 (32-bit) and Windows 7 Professional, Ultimate and Enterprise (32-bit and 64-bit). A user authorization (license) is required to operate WinCC flexible. WinCC flexible has replaced the ProTool configuration software. It is possible to import ProTool configuration data (V6) through a simple conversion.

WinCC flexible engineering software

SIMATIC WinCC flexible is available in four versions, depending on the HMI device to be configured. Higher-performance versions include the functionality of the respective lower-performance versions.

WinCC flexible Advanced	SIMATIC Panel PC SIMOTION Panel PC SINUMERIK Panel PC Standard PC Each with the visualization software WinCC flexible RT (Runtime).
WinCC flexible Standard	Mobile Panels – Series 270 Panels – Series 270 Multi Panels – Series 270 and 370
WinCC flexible Compact	Basic Panels Mobile Panels – Series 170 Panels – Series 70 and 170 Multi Panels – Series 170
WinCC flexible Micro	Micro Panels

For configuring panels released after the start of delivery of WinCC flexible 2008, an HSP (Hardware Support Package) is required that can be downloaded free of charge.

Fig. 7.13 Available versions of WinCC flexible

SIMATIC WinCC flexible RT visualization software

WinCC flexible RT requires a user authorization (runtime license) for operation, which is available with 128, 512, 2048, or 4096 PowerTags (variables with process link to the controller). To configure the visualization software, the engineering software WinCC flexible Advanced is required.

WinCC flexible RT includes the central HMI components for all visualization tasks, such as operator functions, graphics and plots, signaling system, and logging. The visualization takes place via a Windows-compliant user interface and can be extended as needed through option packages.

WinCC flexible options

WinCC flexible options allow you to expand the standard functionality of WinCC flexible Runtime. A special license is required for each option. The WinCC flexible engineering system already contains the functionality of the Runtime options. You do not need a license on your configuration PC to configure the functionality of a runtime option. The following options are available:

- *WinCC flexible /Archives* is used to archive process values and messages either manually, process-controlled or time-controlled. Archived process values can be read back, and used as a basis for a configurable trend display.
- *WinCC flexible /Recipes* generates and manages machine parameters and production data on the basis of data records, and controls exchange with the PLC. Data records can also be imported and exported as CSV files, e.g. for Microsoft Excel.
- *WinCC flexible /Audit* permits the recording of operator actions in an Audit Trail. The operator actions to be recorded can be selected according to whether an electronic signature or comment is required during runtime.
- *WinCC flexible /Sm@rtServices* is used for remote maintenance and servicing over the Internet. An integral Web server provides standard HTML pages with which, *inter alia*, recipe data records with password protection can be accessed or configuration data downloaded to the controller. Events in the controller can trigger the dispatch of an e-mail to maintenance personnel.
- *WinCC flexible /Sm@rtAccess* contains functions for communication between different SIMATIC HMI systems. This facilitates control and monitoring of widely distributed machines with several operator stations by a central operator.
- *WinCC flexible /OPC-Server* contains functions for communication with applications from different vendors on the basis of OPC (OLE for process control). With WinCC flexible RT, OPC is used on the basis of DCOM; for the Multi Panels MP 277 and MP 377, OPC is available on the basis of XML.
- *WinCC flexible /ProAgent* offers a standardized diagnostics concept for various SIMATIC components without necessitating additional configuration for the diagnostics functionality. Thus the controller is also off-loaded with regard to memory requirements and program runtime.

Visualization and operator control using the SCADA system SIMATIC WinCC

SIMATIC WinCC is a PC-based software package for visualization and operation of machine control systems and plant processes in all industries – from simple single-user systems through to distributed multi-user systems with redundant servers and cross-location solutions with web clients.

SIMATIC WinCC V7.0 SP2 is executable under Windows 7 Professional/Enterprise/Ultimate (32-bit), Windows XP Professional SP3, Windows 2003 Server (R2) SP2, and Windows 2008 Server SP2. WinCC contains the Microsoft SQL Server 2005 SP2.

SIMATIC WinCC is the basic software for visualization and operation, and can be expanded by WinCC options and WinCC add-ons. In the basic configuration, WinCC consists of a configuration system (CS) with various WinCC editors and a runtime system (RT).

You can use the configuration system to create process displays for operation and monitoring of a machine or plant, to handle data occurring anywhere in the plant, and to document these in reports. The runtime system permits machine or plant control using the configured GUI, archiving of generated data and events with time tagging in an SQL database, and communication with the configured automation systems.

WinCC Editors

WinCC Explorer is the highest level within the WinCC system. From here you start the editors with which WinCC's different specialist tasks are handled.

User Administrator assigns user rights for the runtime modules of the individual editors. Every time a user tries to log in, the system checks for the correct user rights and then enables the project areas for which the user has the necessary authorization.

You use the *Graphics Designer* to create process pictures. It offers user-friendly and simple user interfaces with tool and graphics palettes and supports systematic configuration through its integrated object and symbol library. Open interfaces allow graphic import and incorporation of self-developed graphic objects; the OLE 2.0 interface is supported. A wizard helps you to configure the dynamic response of the screen objects.

Global Scripts is the generic term for C functions and actions. Project functions and actions are confined to the project in which they were created. Standard and internal functions can be used across projects. Example: dynamization of process value archives, user archives, and condensing archives.

Tag Logging contains functions that read the process information into the HMI system and prepare this data for visual representation and archiving. Essential economic and technical criteria of the operating mode of a plant can be gleaned from this data.

Alarm Logging contains functions for importing events, messages, and alarms from processes and for the preparation, display, acknowledgement, and archiving of these. Alarm Logging is intended to enable you to obtain comprehensive information about fault and operating modes, to detect critical situations in time, to reduce or avoid downtime, and to improve product quality.

The *Report Designer* is the central logging system in WinCC for user reports or project documentation. It provides time-driven and event-driven generation of reports on messages, operator inputs, archive contents, and current or archived data in any freely selectable layouts. During output, the configured placeholders are replaced dynamically by the appropriate data.

Summary of configuration with WinCC

With *WinCC Explorer* you create a single-user or multi-user project, configure the connection to the automation system, and create the required variables. With the *WinCC Graphics Designer* you configure displays with static texts and graphics, insert dynamic picture elements, and combine them using variables. The operator-accessible picture elements are subsequently inserted, and linked to actions.

Depending on the application and the scope of functions, you create e.g. archives for process data using the *Tag Logging* function, and configure the display for the process data trends. With the *Alarm Logging* function you create the signaling system with message blocks, message classes, message texts and limit monitoring functions, define archiving of the messages, and configure the message window.

If operation with access privileges is to be provided, create users and user groups by means of the *User Administrator* and assign corresponding privileges to them. In the case of multilingual operations, you translate the message, display and archive texts, and activate the language selection function. Finally, you define the runtime properties and the start screen, and activate the project.

WinCC Options and WinCC Add-Ons

WinCC options are developed, marketed, and supported by the WinCC manufacturer. WinCC options are, for example WinCC /Server (allows client/server operation with up to 12 WinCC servers and 32 clients), WinCC /Redundancy (increases system availability), WinCC /ProAgent (specific process fault diagnostics), WinCC /Dat@Monitor (display and evaluation of current process statuses and archived data with Internet-based standard tools), WinCC /User Archives (permits storage of freely configurable data records such as the operation parameters of a machine in the WinCC database, representation in the form of a table or form, data import/export), or WinCC /Audit (permits tracing and recording of modifications: What was changed in the project? What operations were made by whom and when?).

WinCC add-ons are created on the initiative of the respective vendor and usually also marketed by them. They can solve diverse tasks such as maintenance and energy management, or communication with non-Siemens controllers. WinCC add-ons are available as communication channel DLLs to WinCC, ActiveX Control, graphic object, or autonomous software package. They are available in two versions: *WinCC Premium add-ons* are checked in the Siemens Test Center for compatibility with the WinCC basic system, and are primarily provided with support from the Siemens Hotline; *WinCC 3rd Party add-ons* are not subject to qualification tests in the Siemens Test Center, and are exclusively provided with support from the respective add-on vendor.

7.12 Process Diagnostics in the User Program Using S7-PDIAG

The S7-PDIAG option package enables the configuration of the process diagnosis in the user program of an S7-300/400 station (CPU 314 and above) with STEP 7 V5.5 in the LAD, FBD and STL programming languages. The process diagnosis monitors the operation of a controlled machine or plant, detects and reports error conditions, provides tips for troubleshooting, and thus helps to reduce downtime. In conjunction with the HMI configuration software and the option package

SIMATIC ProAgent, which is specially tailored to process diagnosis, a powerful system can be created for viewing, diagnosing, and correcting process errors on the 270 and 370 Series Panels as well as on PCs with WinCC.

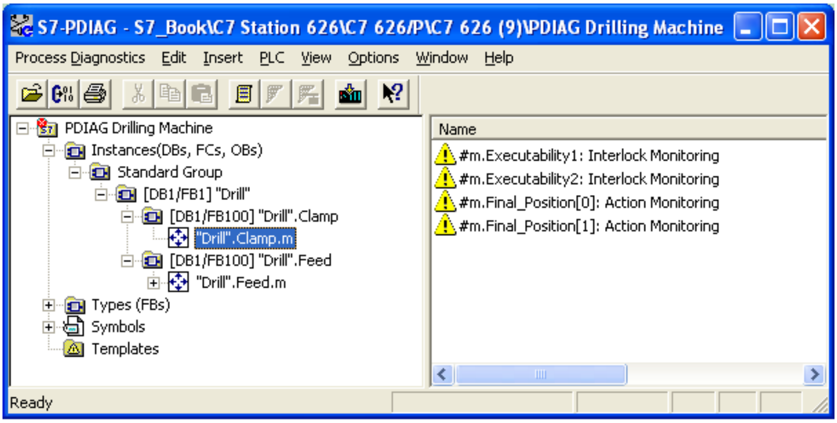
PDIAG monitoring methods

To detect errors, PDIAG monitors the process in various ways. *General monitoring* for monitoring the logic operations on addresses. *Address monitoring* records a signal level or a signal edge of individual addresses, which can be combined with a delay time. *Motion monitoring* checks whether physical motions in the process are performed correctly and quickly enough. The following motion monitoring functions are predefined:

- ▷ Action monitoring (after initiation, is the selected end position reached in a specified time?)
- ▷ Startup Monitoring (after initiation, has the device left the selected end position within the specified delay?)
- ▷ Reaction Monitoring (has the device, after having passed a position flag, reached the end position within the specified delay? Or, has the device, after it has reached the end position, left this end position within the specified delay?)
- ▷ Interlock Monitoring (after initiation, has the interlocking condition been fulfilled within the specified delay?).

Structuring according to units

There is one *unit* for each diagnostics-capable block in the user program. A unit consists of components that belong together technologically and allows rapid localization of the process error. Individual error definitions, motions, or sub-units can be combined into one unit (Fig. 7.14).



#m.Executability1: Interlock Monitoring
#m.Executability2: Interlock Monitoring
#m.Final_Position[0]: Action Monitoring
#m.Final_Position[1]: Action Monitoring

Fig. 7.14 Unit overview of the example supplied with the S7-PDIAG option package

Motions are monitored process operations with two directions and two or more stable end positions. When the motion is initiated, it moves into one of the directions assigned to it. You use the *Error Definition* function to specify the error that generates an error message. This could be, for example, the signal level of an address after a specified delay, or self-defined monitoring logic using PDIAG language elements.

PDIAG shows the units already configured in the unit overview. In the *Blocks* folder are the diagnostics-capable blocks with the subunits and error definitions.

Project Configuration with PDIAG

You create a monitoring function either with the STEP 7 V5.5 program editor or directly with PDIAG.

In the program editor, position the cursor on the address for which you want to create a monitoring function in connection with an assignment statement or a coil or a box, and choose *Edit >Special Object Properties>Monitoring*. In the *Process Monitoring* window, select the desired monitoring mode.

The address selected at the beginning is the “initial diagnostic address”. A criteria analysis can take place based on it. Here, all the signal states are displayed that led to the alarm. The display device performs the criteria analysis without an additional user program. You can also configure monitoring directly in PDIAG (in SIMATIC Manager, start PDIAG with *Options* > *Configure Process Diagnostics*), but you will then receive no criteria analysis. This procedure is suitable, for example, for inputs for which there is no assignment in the user program.

Motion monitoring requires a special program. An LAD program which is adapted to the ProAgent standard diagnostic display is part of the programming example (function block FB 100) delivered with the PDIAG tool. You can use this program as a model for your own monitoring programs.

After you have configured all monitoring functions, select the appropriate blocks in the SIMATIC Manager and choose *Options* > *Configure Process Diagnostics*. In the displayed PDIAG window with the unit overview, you can now specify the blocks for fault detection and initial value/status acquisition using *Options >Customize* on the *Default settings* tab. You then start the compilation with *Process Diagnostics* > *Compile*. To activate the monitoring functions, you must call the blocks with the monitoring functions in a cyclically processed block, e.g. in organization block OB1.

7.13 Process Diagnostics Using SIMATIC ProAgent

SIMATIC ProAgent, in conjunction with the engineering tools STEP 7 V5.5 as well as WinCC and WinCC flexible, makes standard diagnostics screens available for process fault diagnosis in machinery and plants. In the event of failure, information about the location and cause of the error is displayed, which helps to troubleshoot effectively. The standardized user interface offers consistent operation on all supported Operator Panels, Touch Panels, Multi Panels, and PC-based HMI devices on machinery and plants. ProAgent is matched to the requirements of STEP 7 engineering tools in conjunction with SIMATIC S7-300/400 and WinAC.

There is thus a single standardized diagnostics concept for the operation and monitoring of machines and plants. ProAgent reduces the load on the SIMATIC station by the HMI device with respect to memory capacity and program execution time (e.g. by means of criteria analysis in the HMI device, saving structural information, address comments, etc.). At runtime, the standard diagnostics screens are supplemented by process-specific data related to the context, i.e. matching the displayed message or the selected technological unit.

Starting the diagnostics: message diagram displaying faults that have occurred in the process

The message view shows all pending process messages. It may be possible to directly determine the cause of the fault from the alarm message. For more accurate fault localization, you can select a message in the message view and from there access other diagnostics screen.

Get the overall picture: unit overview

The unit overview shows all technological plant or machine units and sub-units and their current operating modes. The operator can thus isolate a faulty unit immediately and take the required actions as specified by the configuration.

Analysis of faulty step sequences and control in the step diagram

In conjunction with S7-GRAPH, the step sequence view supports the analysis and operation of a faulty sequential control, e.g. you can initialize or deactivate the selected step sequence or activate or deactivate a particular step.

Plant control and troubleshooting using the motion diagram

The motion diagram allows you to use the function keys or the direct control keys to initiate specific motions in individual technological units, for example drive a faulty unit to the safe end position.

Condition analysis in the diagnostic detail view

Condition analysis means that an error is traced to its position in the program. The program code is displayed as statement list (STL) or ladder logic (LAD), depending on the programming language used. In the program you can see which signals caused the fault message.

7.14 Telephone network connections with TeleService

You use the TeleService option package to connect a programming device or PC to a PLC or HMI station via the telephone network. This enables you to manage, control, and monitor remote machines or plants from a central point. TeleService is standalone software and does not require the installation of STEP 7. TeleService is included in the scope of delivery of STEP 7 inside TIA Portal. The connection to the central PG/PC is established via a modem. A TS adapter creates the connection on the station side:

- ▷ TS-Adapter II in S7-300 design
  The integrated modem can work with either analog connections or ISDN. The adapter has a USB interface for parameter assignment and an RS232 interface for connecting an external modem, for example, a wireless modem. The adapter can be connected to PPI, MPI, or PROFIBUS DP. Prerequisite for operation is TeleService V6.0.

- ▷ TS-Adapter IE in S7-300 design
  The integrated modem can work with either analog connections or ISDN. The adapter has an RS232 interface for connecting an external modem, for example, a wireless modem. The adapter is equipped with an RJ45 Ethernet interface. Parameters can be assigned using TeleService V6.1 or an Internet browser.

- ▷ TS-Adapter IE Basic in S7-1200 design
  The TS-Adapter IE Basic consists of the basic unit and a TS module with the modem or an interface for connecting to an external modem. The basic unit has an Ethernet interface for connecting to a PG/PC, or programmable controller. The TS-Adapter IE Basic is parameterized with TeleService inside TIA Portal. It is optimized for use with the S7-1200, but can also be used together with the S7-300/400.

Fig. 7.15 shows the basic structure of a remote connection using a TS adapter.

The TS adapter parameters are assigned via the direct connection. The direct connection to the TS-Adapter II can also be used to connect a PC with the COM or USB interface to an MPI or PROFIBUS network.

For remote maintenance, you need the TeleService software or the TIA Portal, a modem on the PG/PC side, and a TS adapter on the system side. In addition, the following functions are possible:

- ▷ PG-PLC remote connection
  PRODAVE V5.0 or higher is installed on the PG/PC – this is a toolbox for process data traffic between the PG/PC and the programmable controller – and the TS-Adapter II on the system side. The programmable controller can also initiate a connection: The function block PG_DIAL in the user program transmits a telephone number and a freely assignable identifier (e.g. a message number) to the TS adapter.

- ▷ PLC-PLC remote connection
  The PLC-PLC remote link allows two S7-300/400 programmable controllers to exchange process data. Each programmable controller can establish a connection via the telephone network using the function block AS_DIAL and the TS-Adapter II. The data is transferred via the external station S7 basic communication with the system functions X_SEND, X_RCV, X_GET, and X_PUT.

- ▷ Sending text messages
  On the system side, a TS-Adapter II and a GSM wireless modem are connected to

the S7-300/400 station. To send a text message, the function block SMS_SEND is called in the user program.

- ▷ Sending emails
  On the system side, a TS-Adapter IE is connected to the S7-300/400 station. To send an email, the function block AS_MAIL is called in the user program. This transfers an email to a mail server by means of SMTP (simple mail transfer protocol).

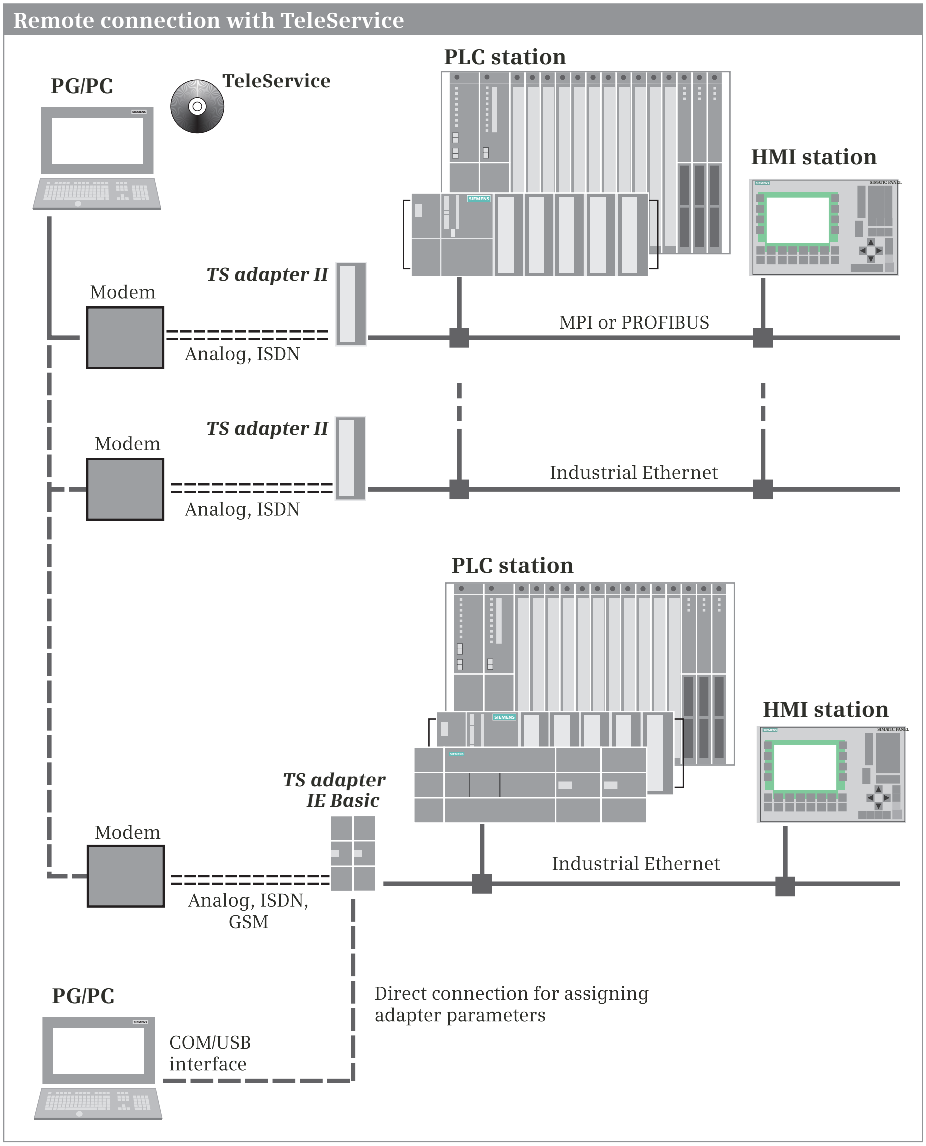


Fig. 7.15 Controlling SIMATIC stations over a remote connection using TeleService

- ▷ Remote control of HMI stations
  For remote control via an Ethernet connection, an Internet browser and the WinCC flexible Sm@rtViewer optional software are required on the PG/PC. On the system side, a TS-Adapter IE is connected to the HMI station, on which the WinCC flexible /Sm@rtService or WinCC flexible /Sm@rtAccess runtime software must be installed.

Index

Abbreviations

AC	Alternating Current
PLC	Programmable Logic Controller
AI	Analog Input
AQ	Analog output
AS	Automation system
ASI	Actuator-sensor interface
BIE	Binary Result
CB	Communication Board
CFC	Continuous Function Chart
CM	Communication Module
CP	Communication Processor
CPU	Central Processing Unit
DB	Data Block
DC	Direct Current
DI	Digital Input
DO	Digital Output
DP	Distributed I/O
DS	Data Set
EPROM	Erasable Programmable Read Only Memory
FB	Function Block
FBD	Function Block Diagram
FC	Function Call
FEPROM	Flash Erasable Programmable Read Only Memory (electrically erasable fixed value memory)
FM	Function Module
HMI	Human Machine Interface
IM	Interface Module
LAD	Ladder Diagram (STEP 7)
LED	Light-emitting Diode
MC	Memory Card
MCR	Master Control Relay
MMC	Micro Memory Card
MP	Multi Panel
MPI	MultiPoint Interface, standard interface of the SIMATIC S7 devices
OB	Organization Block
OP	Operator Panel
PC	Personal Computer
PG	Programming device
PLC	Programmable Logic Controller
PP	Pushbutton Panel
PPI	Point-to-Point Interface
PS	Power Supply
PTP	Point to Point
RAM	Random Access Memory
RLO	Result of Logic Operation
SB	Signal Board
SCL	Structured Control Language
SDB	System Data Block
SDT	System Defined Data Type
SFB	System Function Block
SFC	System Function Call
SM	Signal Module
SSL	System Status List
STL	Statement List
TD	Text Display
TP	Touch Panel
UC	Universal Current
UDT	User Defined Data Type
VAT	Variable Table (STEP 7 V5.5)

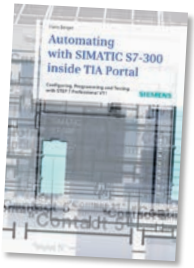
Hans Berger

Automating with SIMATIC S7-1200

Hardware Components, Programming with STEP 7 Basic in LAD and FBD, Visualization with HMI Basic Panels

2011, 413 pages, 290 illustrations, hardcover
ISBN 978-3-89578-356-2, € 49.90

S7-1200 is the first controller of the new SIMATIC generation. The book presents the hardware components of the automation system S7-1200 as well as its configuration and parameterization. A profound introduction into STEP 7 Basic (TIA Portal) shows the basics of programming and trouble shooting.



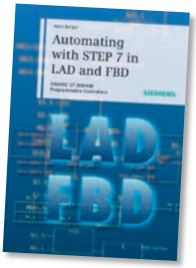
Hans Berger

Automating with SIMATIC S7-300 inside TIA Portal

Configuring, Programming and Testing with STEP 7 Professional V11

2012, 709 pages, 429 illustrations,
85 tables, hardcover
ISBN 978-3-89578-382-1, € 69.90

The user interface of the engineering framework TIA Portal is tuned to intuitive operation and encompasses all the requirements of automation within its range of functions: from configuring the controller, through programming in the different languages, all the way to the program test. The book describes the configuration and network configuring of the SIMATIC S7-300 components with the STEP 7 V11 engineering software in the programming languages LAD, FBD, STL and SCL. The distributed I/O is configured with PROFIBUS DP and PROFINET IO, and data exchange is configured via Industrial Ethernet.



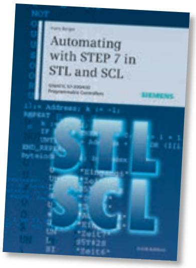
Hans Berger

Automating with STEP 7 in LAD and FBD

SIMATIC S7-300/400
Programmable Controllers

5th revised and enlarged edition, 2012,
451 pages, 163 illustrations, 109 tables, hardcover
ISBN 978-3-89578-410-1, € 69.90

This book was written for all users of SIMATIC S7 controllers. It describes elements and applications of the graphic-oriented programming languages LAD (ladder diagram) and FBD (function block diagram) for use with both SIMATIC S7-300 and SIMATIC S7-400. It provides an introduction to latest version of the engineering software STEP 7 with new functions for PROFINET IO. First-time users are introduced to the field of programmable controllers, while advanced users learn about specific applications of the SIMATIC S7 automation system.



Hans Berger

Automating with STEP 7 in STL and SCL

SIMATIC S7-300/400
Programmable Controllers

6th revised and enlarged edition, 2012,
553 pages, 168 illustrations, 151 tables, hardcover
ISBN 978-3-89578-412-5, € 69.90

The readers learn all about elements and applications of the text-oriented programming languages statement list (STL) and structured control language (SCL) for use with both SIMATIC S7-300 and SIMATIC S7-400. It provides an introduction to the latest version of the programming software STEP 7. First-time users are introduced to the field of programmable controllers, while advanced users learn about specific applications of the SIMATIC S7 automation system.

Raimond Pigan, Mark Metter

Automating with PROFINET

Industrial Communication based on Industrial Ethernet

2nd edition, 2008, 462 pages,
271 illustrations, 237 tables, hardcover
ISBN 978-3-89578-294-7, € 59.90

This book serves as an introduction to PROFINET technology. Engineers, technicians and students are given an overview of the concept and the fundamentals for solving automation tasks. Technical relationships and practical applications are described using SIMATIC products as example.

Norbert Bartneck, Volker Klaas,
Holger Schönherr (Eds.)

Optimizing Processes with RFID and Auto ID

Fundamentals, Problems and Solutions, Example Applications

2009, 255 pages, 86 illustrations, hardcover
ISBN 978-3-89578-330-2, € 34.90

As well as introducing Auto ID technology basics, this book presents tried and tested applications from different areas. It shows the approach, the process and the selection of RFID and Auto ID systems for various problems. A perspective on trends and innovative security solutions shows possible future application options for this technology.

Industry Automation Translation Services (Eds.)

Wörterbuch Elektrotechnik, Energie- und Automatisierungstechnik Dictionary of Electrical Engineering, Power Engineering and Automation

Teil 1: Deutsch-Englisch / Part 1: German-English

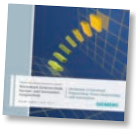
6th extensively revised and substantially edition, 2011,
1042 pages, hardcover, ISBN 978-3-89578-313-5, € 89.90

Dictionary of Electrical Engineering, Power Engineering and Automation Wörterbuch Elektrotechnik, Energie- und Automatisierungstechnik

Part 2 English-German; Teil 2 Englisch-Deutsch

6th extensively revised and substantially edition, 2009,
994 pages, hardcover, ISBN 978-3-89578-314-2, € 89.90

The worldwide-respected standard work for translators, engineers, and technical writers, altogether containing about 240,000 entries and 320,000 translations in both language directions.



CD-ROM, Edition 2011

German-English; English-German
Deutsch-Englisch; Englisch-Deutsch

Windows 7/Vista/XP
ISBN 978-3-89578-315-9, € 189.00

Also available on the App Store

Nicolai Andler

Tools for Project Management, Workshops and Consulting

A Must-Have Compendium of Essential Tools and Techniques

2nd revised and enlarged edition, 2011,
382 pages, 136 illustrations, 55 tables, hardcover
ISBN 978-3-89578-370-8, € 39.90

The unique reference work and guide for all those practising consulting, project management and problem solving. It presents cookbook-style access to more than 120 most important tools, including a rating of each tool in terms of applicability, ease of use and effectiveness.

www.publicis-books.de