Table of Contents

Automation Overview

The word 'Automation' is derived from Greek words "Auto" (self) and "Matos" (moving). Automation therefore is the mechanism for systems that "move by itself".

However, apart from this original sense of the word, automated systems also achieve significantly superior performance than what is possible with manual systems, in terms of power, precision and speed of operation.

Definition: Automation is a set of technologies that results in operation of machines and systems without significant human intervention and achieves performance superior to manual operation

A Definition from Encyclopedia Britannica: The application of machines to tasks once performed by human beings or, increasingly, to tasks that would otherwise be impossible.

Although the term mechanization is often used to refer to the simple replacement of human labor by machines, automation generally implies the integration of machines into a self-governing system.

Types of automation

Automation systems can be categorized based on the flexibility and level of integration in manufacturing process operations. Various automation systems can be classified as follows

Fixed Automation: It is used in high volume production with dedicated equipment, which has a fixed set of operation and designed to be efficient for this set. Continuous flow and Discrete Mass Production systems use this automation. E.g. Distillation Process, Conveyors, Paint Shops, Transfer lines etc. A process using mechanized machinery to perform fixed and repetitive operations in order to produce a high volume of similar parts.

Programmable Automation: It is used for a changeable sequence of operation and configuration of the machines using electronic controls. However, non-trivial programming effort may be needed to reprogram the machine or sequence of operations. Investment on programmable equipment is less, as production process is not changed frequently. It is typically used in Batch process where job variety is low and product volume is medium to high, and sometimes in mass production also. E.g. in Steel Rolling Mills, Paper Mills etc.

Flexible Automation: It is used in Flexible Manufacturing Systems (FMS) which is invariably computer controlled. Human operators give high-level commands in the form of codes entered into computer identifying product and its location in the sequence and the lower level changes are done automatically. Each production machine receives settings/instructions from computer. These automatically loads/unloads required tools and carries out their processing instructions. After processing, products are automatically transferred to next machine. It is typically used in job shops and batch processes where product varieties are high and job volumes are medium to low. Such systems typically use Multipurpose CNC machines, Automated Guided Vehicles (AGV) etc.

Integrated Automation: It denotes complete automation of a manufacturing plant, with all processes functioning under computer control and under coordination through digital information processing. It includes technologies such as computer-aided design and

manufacturing, computer-aided process planning, computer numerical control machine tools, flexible machining systems, automated storage and retrieval systems, automated material handling systems such as robots and automated cranes and conveyors, computerized scheduling and production control. It may also integrate a business system through a common database. In other words, it symbolizes full integration of process and management operations using information and communication technologies. Typical examples of such technologies are seen in Advanced Process Automation Systems and Computer Integrated Manufacturing (CIM).

Advantages & disadvantages of automation

The main advantages of automation are:

- Increased throughput or productivity.
- Improved quality or increased predictability of quality.
- Improved robustness (consistency), of processes or product.
- Increased consistency of output.
- Reduced direct human labour costs and expenses.

The following methods are often employed to improve productivity, quality, or robustness.

- Install automation in operations to reduce cycle time.
- Install automation where a high degree of accuracy is required.
- Replacing human operators in tasks that involve hard physical or monotonous work.
- Replacing humans in tasks done in dangerous environments (i.e. fire, space, volcanoes, nuclear facilities, underwater, etc.)
- Performing tasks that are beyond human capabilities of size, weight, speed, endurance, etc.
- Economic improvement: Automation may improve in economy of enterprises, society or most of humanity. For example, when an enterprise invests in automation, technology recovers its investment; or when a state or country increases its income due to automation like Germany or Japan in the 20th Century.
- Reduces operation time and work handling time significantly.
- Frees up workers to take on other roles.
- Provides higher level jobs in the development, deployment, maintenance and running of the automated processes.

The main disadvantages of automation are:

- Security Threats/Vulnerability: An automated system may have a limited level of intelligence, and is therefore more susceptible to committing errors outside of its immediate scope of knowledge (e.g., it is typically unable to apply the rules of simple logic to general propositions).
- Unpredictable/excessive development costs: The research and development cost of automating a process may exceed the cost saved by the automation itself.
- High initial cost: The automation of a new product or plant typically requires a very large initial investment in comparison with the unit cost of the product, although the cost of automation may be spread among many products and over time.

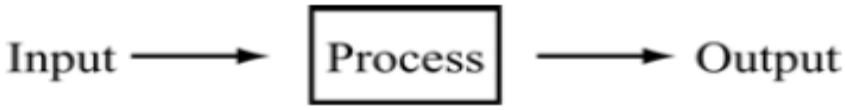
- Additional training and technical help is needed for the installation as well as maintenance.

Control strategies

There are basically two types of control system: the open loop system and the closed loop system. They can both be represented by block diagrams. A block diagram uses blocks to represent processes, while arrows are used to connect different input, process and output parts. Technological Studies Control Systems

1. Open loop control system

    - A simple open loop control system.
    - Its operation is very simple, when an input signal directs the control element to respond, an output will be produced. Examples of the open loop control systems include washing machines, light switches, gas ovens, etc.

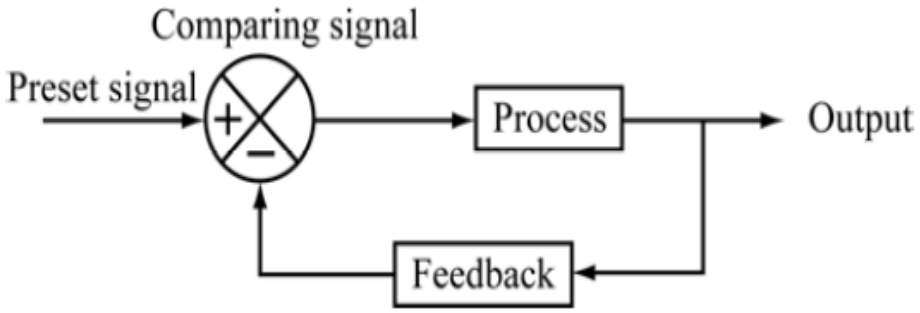


    - More sophisticated example of an open loop control system is the burglar alarm system. The function of the sensor is to collect data regarding the concerned house. When the electronic sensor is triggered off, it will send a signal to the receiver. The receiver will then activate the alarm, which will in turn generate an alarm signal. The alarm signal will not cease until the alarm is stopped manually.
    - The drawback of an open loop control system is that it is incapable of making automatic adjustments. Even when the magnitude of the output is too big or too small, the system will not make the appropriate adjustments. For this reason, an open loop control system is not suitable for use as a complex control system. Sometimes it may even require monitoring and response from the user.

2. Closed loop control system

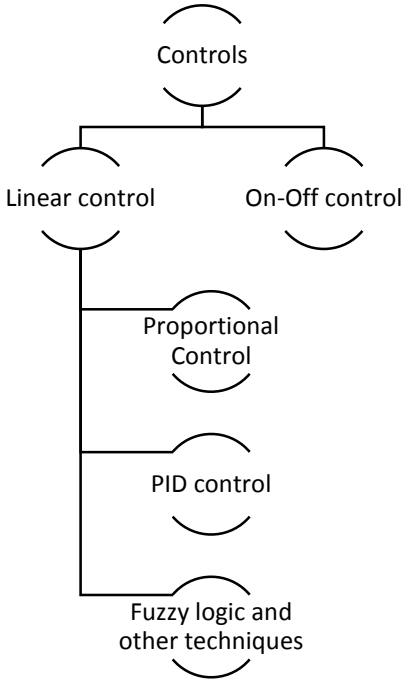
    - Sometimes, we may use the output of the control system to adjust the input signal. This is called feedback. Feedback is a special feature of a closed loop control system. A closed loop control system compares the output with the expected result or command status, then it takes appropriate control actions to adjust the input signal. Therefore, a closed loop system is always equipped with

a sensor, which is used to monitor the output and compare it with the expected result.



- A simple closed loop system.
- The output signal is fed back to the input to produce a new output. A well-designed feedback system can often increase the accuracy of the output. Figure above can be divided into positive feedback and negative feedback. Positive feedback causes the new output to deviate from the present command status. For example, an amplifier is put next to a microphone, so the input volume will keep increasing, resulting in a very high output volume. Negative feedback directs the new output towards the present command status, so as to allow more sophisticated control. For example, a driver has to steer continuously to keep his car on the right track. Most modern appliances and machinery are equipped with closed loop control systems. Examples include air conditioners, refrigerators, automatic rice cookers, automatic ticketing machines, etc.
- One advantage of using the closed loop control system is that it is able to adjust its output automatically by feeding the output signal back to the input. When the load changes, the error signals generated by the system will adjust the output. However, closed loop control systems are generally more complicated and thus more expensive to make.

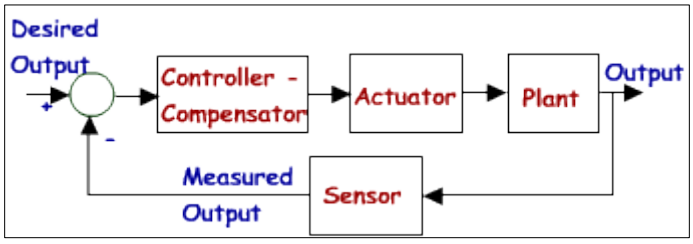
Types of control



On-Off control: According to Feedback, Logic will be developed and then with the help of controllers, 2 state controlling is being performed.

Linear Control: According to Feedback, process is being controlled linearly. It can be proportionally/ integrally, derivatively controlled. There are some other methods for controlling too.

Components of automation



Sensor: Produce the input value for the controller as feedback. For example, RTD, strain gauge, float sensor.

Controller: Controls the process by sending the particular value according to the set value and feedback coming from the sensor. Difference between them called error. According to the behavior of error, control strategies is being decided. Controllers can be Digital controllers, dedicated controllers, PLCs etc.

Actuators: Component which is acting on the response of the controller. And as a result process value will be effected.

Plant: The whole process and machines which is producing the product is called plant whose parameters are measures by sensors.

PLC introduction

A programmable logic controller (PLC) is an industrially hardened computer-based unit that performs discrete or continuous control functions in a variety of processing plant and factory environments. Originally intended as relay replacement equipment for the automotive industry, the PLC is now used in virtually every industry imaginable. Though they were commonly referred to as PCs before 1980, PLC became the accepted abbreviation for programmable logic controllers, as the term "PC" became synonymous with personal computers in recent decades.

Difference between PC & PLC

The PC is designed to be flexible and handle thousands of different types of applications, but PLCs are especially designed for control.[35] PCs started showing up on the factory floor in the mid-80s in programming and HMI applications. However, now the PC is migrating toward actual control. In response, PLC manufacturers have adapted PC technology, such as Ethernet. Some systems actually provide a "PC on a card" to plug into the PLC back plane.[11] PLCs are often thought of as computers. To a certain extent this is true; however, there are important differences between PLCs and computers.

Real-Time Operation/Orientation *the* PLC is designed to operate in a real-time control environment. The first priority of the CPU is to scan the I/O for status, make sequential control decisions (as defined by the program), implement those decisions, and repeat this procedure all within the allotted scan time. Most PLCs have internal clocks and "watchdog timers" built into their operations to ensure that a software error like "divide by zero" or an endless loop does not send the central processor into an undefined state. When the watchdog time is exceeded, the processor shuts down in a predetermined manner and usually turns off all outputs. In real time systems, reliability is a big concern. PLC manufacturers' experience shows mean time between failures (MTBF) ranging from 20,000 to 400,000 hours. "This is far in excess of almost any other type of electronic or control equipment."

Environmental Considerations PLCs are designed to operate near the equipment they control. This means they function in hot, humid, dirty, noisy, and dusty industrial environments. Typical PLCs can operate in temperatures as high as 140°F (60°C) and as low as 32º F (0°C), with tolerable relative humidity ranging from 0 to 95% noncondensing. In addition, they have electrical noise immunities comparable with those required in military specifications.

Programming Languages and Techniques PLC languages are designed to emulate the popular relay ladder diagram format. This format is read and understood worldwide by maintenance technicians as well as engineers. Unlike computer programming, PLC programming does not require extensive special training. Applications know-how is much more important. Although certain special techniques are important for programming efficiency, they are easily learned. The major goal is the control program performance. Another difference

between computers and PLCs is the sequential operation of the PLC. Program operations are performed by the PLC in the order they were programmed (Figure 5.4o). This is an extremely useful feature that allows easy programming of shift registers, ring counters, drum timers, and other useful indexing techniques for real-time control applications.

Maintenance and Troubleshooting As a plant floor controller, the plant electrician or the instrument technician must maintain the PLC. It would be highly impractical to require computer-type maintenance service. To this end, PLC manufacturers build in self-diagnostics to allow for easy troubleshooting and repair of problems. Most PLC components are modular and simple to isolate; remove-and-replace (system modules) diagnostic techniques are usually implemented.

Advantages & Disadvantages of PLC

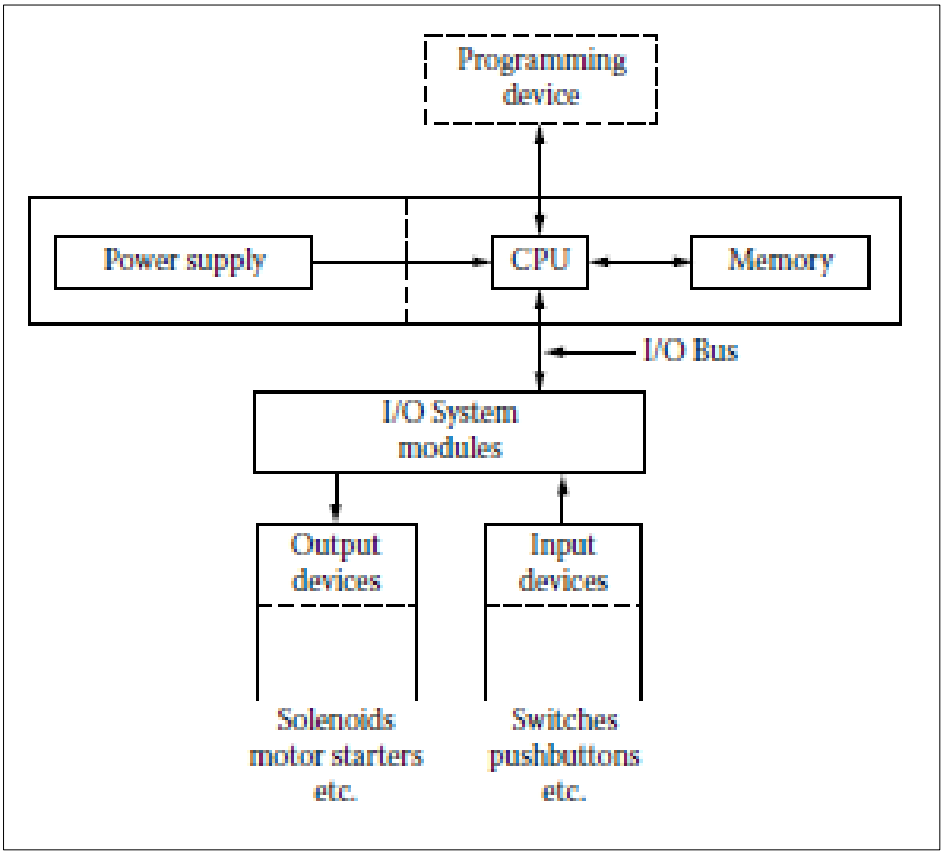
Advantages:

- Very High Accuracy
- Low Power Consumption (Energy Saving)
- High Level human Safety
- Less Plan Running cost
- Small in Size (Required Lased space)
- Rugged Construction
- Easily programmable
- Easy Maintenance
- Economical & high Flexible
- Shorter project program
- Easy Documentation

Disadvantages:

- High Skilled Engineering Required for that high skilled person required
- Difficulty with changes or replacements
- Some high Initial Cost.

Block diagram



This is the detailed block diagram of PLC.

Scan cycle of PLC

A PLC program is generally executed repeatedly as long as the controlled system is running. The status of physical input points is copied to an area of memory accessible to the processor, sometimes called the "I/O Image Table". The program is then run from its first instruction rung down to the last rung. It takes some time for the processor of the PLC to evaluate all the rungs and update the I/O image table with the status of outputs. This scan time may be a few milliseconds for a small program or on a fast processor, but older PLCs running very large programs could take much longer (say, up to 100 ms) to execute the program. If the scan time were too long, the response of the PLC to process conditions would be too slow to be useful.

Special-purpose I/O modules may be used where the scan time of the PLC is too long to allow predictable performance. Precision timing modules, or counter modules for use with shaft encoders, are used where the scan time would be too long to reliably count pulses or detect the sense of rotation of an encoder. The relatively slow PLC can still interpret the counted values to control a machine, but the accumulation of pulses is done by a dedicated module that is unaffected by the speed of the program execution.

System overview

SIMATIC S7 overview

SIMATIC is a core part of Totally Integrated Automation and its range includes numerous standardized products and systems - such as the SIMATIC Controllers presented in this brochure. Whether you prefer a conventional PLC, an embedded or a PC based automation solution: The complete range of SIMATIC Controllers covers solutions for all application areas – and offers the performance capability and flexibility you need.

Positioning of Modular S7 Controllers

The Modular Controllers have been optimized for control tasks and specially designed for ruggedness and long-term availability. They can be flexibly expanded at any time using plug-in I/O modules, function modules, and communication modules. Depending on the size of the application, the right controller can be selected from a wide range according to performance, quantity frameworks, and communication interfaces. The modular controllers can also be used as fault-tolerant or fail-safe systems.

Your benefits

1. Flexible in use
2. Openness in hardware and software configuration
3. Use of existing PC resources
4. Participation in the continuous PC innovation process
5. Multifunctional
6. Customized PC variants
7. Embedded bundles:
    a. Ready to use
    b. Rugged
    c. Maintenance-free

Fields of application

1. Control, operator control and monitoring
2. Technological tasks
3. Data acquisition and archiving
4. Link to PC hardware and software
5. Integration of C/C++/C# programs
6. Data exchange via OPC
7. Fail-safe control

SIMATIC S7-1200: The Modular Mini-PLC

- Compact controllers for the low to mid-performance ranges
- Large-scale integration, space-saving, powerful
- With exceptional real-time performance and powerful communication options:
    - Controller with integrated PROFINET IO controller interface for communication between SIMATIC controllers, HMI, programming device or other automation components
- All CPUs can be used in stand-alone mode, in networks and within distributed structures
- Extremely simple installation, programming and operation
- Integrated web server with standard and user-specific web pages

- Data logging functionality for archiving of data at runtime from the user program
- Powerful, integrated technology functions such as counting, measuring, closed-loop control, and motion control
- Integrated digital and analog inputs/outputs
- Flexible expansion facilities
  - Signal boards for direct use in a controller
  - Signal modules for expansion of controllers by input/output channels
  - Accessories, e.g. power supply, switch module or SIMATIC Memory Card

① Power connector
② Memory card slot under top door
③ Removable user wiring connectors (behind the doors)
④ Status LEDs for the on-board I/O
⑤ PROFINET connector (on the bottom of the CPU)

SIMATIC S7-1500: Modular Controller for mid to upper range Performance

- Modular, scalable, and universally usable system in IP20 level of protection
- The system solution for a variety of automation applications in discrete automation
- Highest performance with excellent usability
- Configurable exclusively in the Totally Integrated Automation Portal with STEP 7 Professional V12 or higher

Performance

- Increase in performance through
  - Faster command execution
  - Language extensions
  - New data types
  - Faster backplane bus
  - Optimized code generation
- Powerful communication:
  - PROFINET IO (2-port switch) as standard interface;

From CPU 1515-2 PN, one or more additional integrated PROFINET interfaces, e.g. for network separation

- Expandable with communication modules for bus systems and point-to-point connection

Integrated technology

- Motion Control integrated without additional modules:
  - Standardized blocks (PLC open) for connection of analog and PROFI drive-capable drives
  - The Motion Control functionality supports speed-controlled and positioning axes as well as external encoders
  - Position wise precise gearing between axes
- Comprehensive trace functions for all CPU tags for real-time diagnosis and sporadic error detection;

  For effective commissioning and quick optimization of drives and controls
- Comprehensive control functionalities:

  E.g. easily configurable blocks for automatic optimization of the control parameters for optimum control quality
- Additional functions through available technology modules:

  E.g. high-speed counting, position detection, or measurement functions for signals up to 1 MHz

Safety Integrated

Protection of personnel and machinery – within the framework of an integrated complete system

- Failsafe SIMATIC S7-1500F controllers for processing standard and safety programs on the same controller.

  Generation of the failsafe and standard user program is carried out in the TIA Portal with the same editors; this enables failsafe data to be evaluated like standard data in the standard user program, for example. Due to this integration the system benefits and the comprehensive functionality of SIMATIC are also available for failsafe applications.

Security Integrated

- Password-based know-how protection against unauthorized reading and modification of program blocks
- Copy protection for greater protection against unauthorized copying of program blocks:

  With copy protection, individual blocks on the SIMATIC memory card can be tied to its serial number so that the block can only be run if the configured memory card is inserted into the CPU.

- Rights concept with four different authorization levels:

  Different access rights can be assigned to various user groups. The new protection level 4 makes it possible to also restrict communication to HMI devices.

- Improved manipulation protection:

  Changed or unauthorized transfers of engineering data are detected by the controller.

- For use of an Ethernet CP (CP 1543-1):
  - Additional access protection by means of a firewall
  - Setup of secure VPN connections (V12 SP1 or higher)

Design and handling

- CPUs with display for plain text information:
  - Information about article numbers, firmware version, and the serial number of all connected modules can be displayed
  - Setting the IP address of the CPU and additional network settings directly on site, without programming device
  - Display of occurring error messages directly as plain text message, meaning reduction in downtime
- Uniform front connectors for all modules and integrated potential bridges for flexible potential group formation simplify stock keeping and reduce wiring costs
- Integrated DIN rail in the S7-1500 mounting rail:

  Quick and easy installation of additional components such as miniature circuit breakers, relays, etc.

- Central expansion with signal modules:

  For flexible adaptation to any application

- System cabling for digital signal modules:

  For fast and clearly arranged connecting to sensors and actuators in the field and simple wiring inside the control cabinet

- Power supply:
  - Load power supply modules (PMs) for supplying the module with 24 V
  - Power supply modules to supply power to the internal module electronics via the backplane bus
- Distributed expansion:
  - Use of up to 30 signal modules, communication modules, and technology modules via the PROFINET interface module IM 155-5 for the ET 200MP I/O system

- No difference in terms of handling and system functions in central and distributed operation

Integrated system diagnostics

- Integrated system diagnostics for CPUs, activated by default:
  - Consistent plain text display of system diagnostic information in the display, TIA Portal, HMI, and web server, even for drive messages. Messages are updated even if the CPU is in STOP state.
  - System diagnostics integrated in the CPU firmware. Configuration by user not required. The diagnostics is automatically updated on configuration changes.

Data log (archives) and recipes

- SIMATIC memory card:
- Plug-in load memory
- Permits firmware updates
- Storage option for STEP 7 projects (including comments and symbols), additional documentation, or csv files (for recipes and archives)
- Easy access to plant-relevant operating data and configuration data with Office tools via the SD Card reader (two-way data exchange from and to the controller)
- Integrated web server:
- Easy access to plant-relevant operating data and configuration data via a Web browser

Approvals

The SIMATIC S7-1500 complies with the following national and international standards:

- cULus approval
- cULus HazLoc approval
- FM approval
- ATEX approval (only for 24 V; not for 230 V)
- CE
- C-TICK
- KCC
- IECEx (24 V only; not for 230 V)
- EN 61000-6-4
- EN 60068-2-1/ -2/ -6/ -14/ -27/ -30/ -32
- EN 61131-2

SIMATIC S7-1200/1500: Technology Functions

S7-1200 Functions

The S7-1200 is characterized by:

- Easy getting started:

Special starter packages including simulators and documentation facilitate familiarization.

- Uncomplicated operation:

  Powerful standard commands which are simple to use, together with the user-friendly programming software, reduce the programming overhead to a minimum.

- Real-time properties:

  Special interrupt functions, fast counters, and pulse outputs permit use even with time-critical processes.

The SIMATIC S7-1200 meets national and international standards:

- UL 508
- CSA C22.2 No. 142
- FM Class I, Div. 2, Groups A, B, C, D; T4A Class I, Zone 2, IIC, T4
- VDE 0160
- EN 61131-2
- Requirements of the EMC directive in accordance with EN 50081-1, 50081-2 and 50082-2

S7-1500 functions

A host of features support users in programming, commissioning, and servicing the S7-1500.

- Performance
- Faster command processing, depending on the CPU type, language extensions and new data types
- Significantly shorter response times through optimized code generation
- Integrated technology
- Simple, fast programming of motion sequences via standard PLC open Motion blocks
- Position wise precise gearing between axes
- Convenient diagnostic and commissioning tools provide support in commissioning drives
- Automatic alarm messages to engineering system and HMI: Simplified troubleshooting saves time and effort in commissioning.
- Isochronous mode
- Synchronous coupling of distributed signal acquisition, signal transmission and program execution to the cycle of the PROFIBUS and PROFINET with constant bus cycle time:
- The input signals are acquired and processed and output signals are output at fixed intervals (constant bus cycle time). A consistent process image partition is created at the same time.
- Precisely reproducible and defined process response times due to synchronous signal processing with constant bus cycle times by the distributed I/O

- In distributed automation solutions, the SIMATIC S7-1500 thus also permits high-speed processing operations and enables the achievement of maximum precision and reproducibility. This means increased production with optimal and constant quality.
- Comprehensive range of components for complex tasks such as motion control, measured value acquisition, high-speed control, etc.
- Security Integrated
- Password-based know-how protection against unauthorized readout and modification of program blocks (in combination with STEP 7)
- Copy protection:

  Protection against unauthorized copying of program blocks. With copy protection, individual blocks on the SIMATIC memory card can be tied to its serial number so that the block can only be run if the configured memory card is inserted into the CPU.

- 4-stage authorization concept

  Different access rights can be assigned to user groups. The new protection level 4 makes it possible to also restrict communication to HMI devices. Improved manipulation protection allows changed or unauthorized transfers of engineering data to be detected by the controller.

- Design and handling
- CPUs with integrated display

  For convenient evaluation of module states for central and distributed modules or to set or change IP address (es) without programming device. System diagnostics and user alarms are shown in plain text on the display and help to respond to occurring error messages quickly and efficiently. The menu and message texts are available in multiple languages on the display.

- Integrated system diagnostics
- System diagnostics information is displayed consistently and in plain text on the display, TIA Portal, HMI device and web server, including for messages from the drives, and are even possible in the CPU's STOP state. This functionality is integrated is the CPU firmware as a system feature and does not have to be configured separately by the user. If new hardware components are configured, the diagnostic information is updated automatically.
- Simple and rapid diagnostics directly in the user program through the quality information:
- By activating the quality information (QI) of a module, the validity of the supplied process value can be queried and evaluated directly in the user program. Here, access is via the process image using simple binary or load commands. The prerequisite is that the module can be diagnosed and the quality information can be configured in the TIA Portal.
- Configuration by means of the SIMATIC STEP 7 Professional engineering software, V12 and higher

- The SIMATIC S7-1500 controller family is programmed in the Totally Integrated Automation Portal using STEP 7 Professional V12 or higher. SIMATIC STEP 7 Professional, V12 and higher, is the intuitively operated engineering system for the SIMATIC S7-1500.
- Compatibility
- Migration:

  A migration tool integrated in SIMATIC STEP 7 Professional V12 or higher provides support in switching from the S7-300/S7-400 to the S7-1500 controller and converts the program code automatically. Program code that cannot be converted automatically is logged and can be adapted manually. The migration tool is also made available as a standalone tool in the download area of Customer Support. STEP 7 V11 projects can continue to be used with STEP 7 V12 in compatibility mode. S7-1200 programs can also be transferred to the S7-1500 by means of copy paste.

- SIMATIC memory card (required for operation of the CPU)
- The SIMATIC memory card is used as plug-in load memory or for updating the firmware. STEP 7 projects including comments and symbols, additional documentation or csv files (for recipes and archives) can also be stored on the SIMATIC memory card. Data blocks can be created and data stored or read via SFCs on the SIMATIC memory card with the user program.
- Safety Integrated (option for S7-1500F controller)
- "STEP 7 Safety Advanced" option package; required for programming the safety-related program sections of the S7-1500F controller.
- The package contains all the functions and blocks required to create an F program. STEP 7 Safety Advanced V12 can run under SIMATIC STEP 7 Professional V12 SP1.

SIMATIC S7-1200/1500: Memory Cards

In a free analogy, comparing a S7-1500 and a PC, the Work memory represents the PC's RAM and the Load memory represents the PS's hard disc drive / pen drives.

Code Work memory (inside CPU – volatile - not expansible):

Where stay the programs blocks when the CPU is running

Data Work memory (inside CPU – volatile - not expansible):

Where stays the DBs when the CPU is running

Load Memory (inside the SIMATIC SD card – nonvolatile - different sizes could be used):

Where stays the program blocks, DB, HW configuration, etc. when the CPU stay turned off.

Highlights:

S7-1500/1200 (like S7-300 modern CPU) need necessarily a memory card (each this respective type, SD / MMC).

- Just SIMATIC Memory Card should be used (no "Standard" SD-Cards could be used).

- S7-1500 can use the same Memory Card types as S7-1200. But can't be used of S7-300 or S7-400.

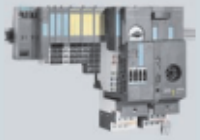
- If you "format" the SIMATIC Memory Card by Window, the card will be damage, and could not be "restored" by the user.

- Commentaries are stored in Load memory (each character takes one byte).

Distributed I/O systems

SIMATIC ET 200S –
the all-rounder with a comprehensive range of functions

- Bit-modular design with multi-conductor connection
- Multifunctional thanks to a wide range of modules: motor starters, frequency converters, safety technology, distributed intelligence, IO-link modules.
- Use in hazardous areas (Zone 2)
- Also available as expandable block version with integral DI/DO: SIMATIC ET 200S COMPACT



SIMATIC ET 200M –
the multi-channel S7-300

- Modular design using standard SIMATIC S7-300 modules; redundant design also possible
- Fail-safe I/O modules
- For use in hazardous areas up to Zone 2, sensors and actuators up to Zone 1.
- High plant availability thanks to redundancy, hot swapping, and configuration changes during operation

SIMATIC ET 200L –
digital block I/O

- Low-cost digital block I/O
- Digital electronic blocks of up to 32 channels

SIMATIC ET 200iSP –
the intrinsically-safe version for hazardous areas

- Modular design, also available with redundancy
- Rugged, intrinsically-safe design
- Use in hazardous areas up to Zone 1/21, sensors and actuators may even be located in Zone 0/20
- High plant availability thanks to redundancy, hot swapping, and configuration changes during operation

SIMATIC S7-300: Modular Automation System

S7-300

- The modular mini PLC system for the low and mid-performance ranges

- With comprehensive range of modules for optimum adaptation to the automation task
- Flexible use through simple implementation of distributed structures and versatile networking
- User-friendly handling and uncomplicated design without a fan
- Can be expanded without problems when the tasks increase
- Powerful thanks to a range of integrated functions

S7-300F

- Failsafe automation system for plants with increased safety requirements for production technology
- Based on S7-300
- Additional ET 200S and ET 200M distributed I/O stations complete with safety-related modules can be connected
- Safety-related communication via PROFIBUS DP with PROFI safe profile
- Standard modules can be used in addition for non-safety-relevant applications

SIMATIC S7-300: Modules

The S7-300 automation system is modular in design. It has a comprehensive range of modules that can be combined individually.

A system includes the following:

- A CPU:

  Different CPUs are available for different performance ranges, including CPUs with integral inputs/outputs and the corresponding functions, as well as CPUs with integral PROFIBUS DP, PROFINET and point-to-point interfaces.
- Signal modules (SMs) for digital and analog inputs/outputs.
- Communications processors (CPs) for bus connection and point-to-point connections.
- Function modules (FMs) for high-speed counting, positioning (open-loop/closed-loop) and PID control.

The following can also be used depending on requirements:

- Load power supply (PS) for connecting the SIMATIC S7-300 to a supply voltage of 120/230 V AC.
- Interface modules (IMs) for connecting the central controller (CC) and expansion units (EUs) in multi-tier configurations. The SIMATIC S7-300 can be operated with up to 32 modules distributed across the CC and 3 EUs. All modules can be operated in enclosures and without fans.
- SIPLUS modules for extended environmental conditions:

Suitable for temperature range -25 to +60 °C, and higher humidity, condensation and frost loads. Can be used direct on vehicles or outside building in an IP20 cabinet protected against direct sunlight and rainwater/spray water. Air-conditioned cabinet and IP65 housing not required.

Digital Fundamentals

Bit-Byte Word Concepts

What is a bit?

Bit is short for 'binary digit.' It's a single digit in a binary number, and it can be either 1 or 0.

Inside a computer, you can think of a bit as being a mechanical switch, which can be either switched on or off (the earliest computers actually stored information in memory using mechanical switches, with electromagnets to turn each one on or off).

Now if you only have one of these switches, you can only store two different states, on or off. This is useful in itself, you can record that something is either true or false.

But if you have, say, eight of them, you can store 256 different combinations of on and off states between the eight switches.

What is a byte?

A byte is 8 bits. That's the definition. With 8 bits you can store any number between 0 and 255, since there are 256 different combinations of 1 and 0 to choose from.

Why eight bits? The original intention was that, when storing text, 8 bits would be enough to assign a unique number every possible language character you might want to use in your document. The idea was that each character in a file would take up one byte of memory (in most cases, this is still true).

Let's see: there are 26 uppercase letters (A-Z), 26 lowercase (a-z), 10 numerical digits (0-9), 32 punctuation characters and other symbols on a US keyboard, the space character that's already 94 different characters. Then there's a few characters for creating newlines, a tab character for indentations, there's even a 'bell' character which programs would output in order to make the user's terminal beep. You can see how it all adds up

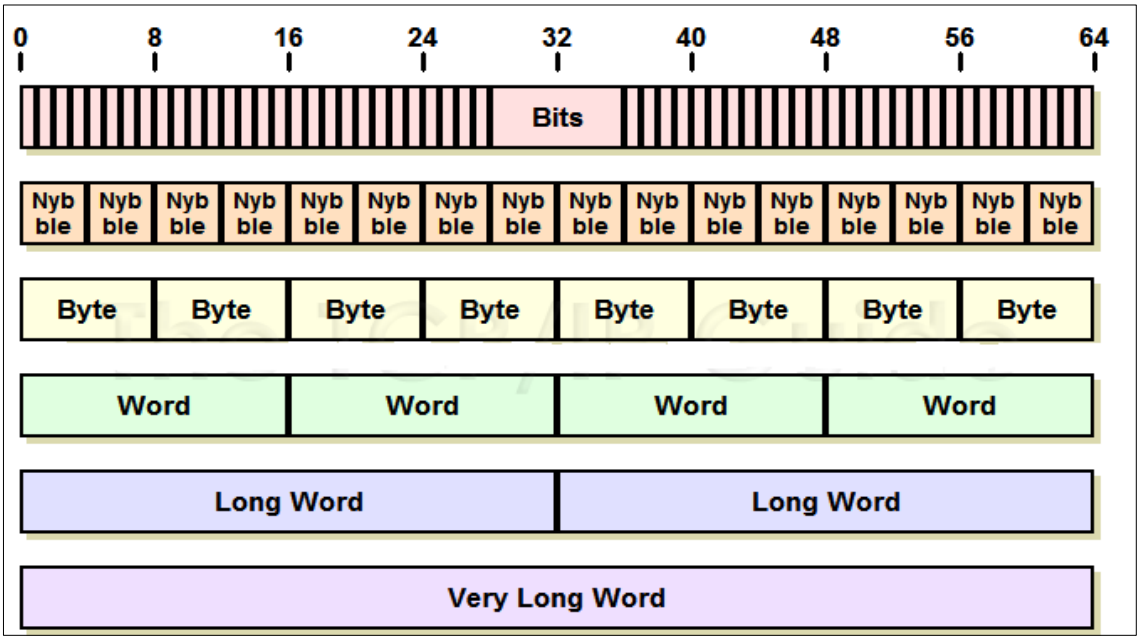
In practice, only characters up to 127 were ever standardized (the standard is called ASCII, which stands for American Standard Code for Information Interchange, because in the early days, one of the eight bits was set aside for error testing purposes (back when computers were far less reliable), and 7 bits only gives you 128 different combinations.

What is a word?

You often hear about 32-bit or 64-bit computer architectures. A word is basically the number of bits a particular computer's CPU can deal with in one go. It varies depending on the computer architecture you're using.

Imagine looking at an imaginary computer's circuitry very closely. On a 32-bit machine, you would see 32 wires running parallel to each other between the computer's memory controller and the CPU, for the purpose of giving the CPU access to one particular word of memory.

Actually, there would be an additional 32 wires (perhaps less) for the CPU to select a particular memory address to access. If a CPU can access 32 bits of memory in one go, then it turns out that it makes a lot of sense to address the computer's memory using ≤32 bits. (This happens to be why the 32-bit version of Windows can't deal with more than 2GB of RAM, but the 64-bit version can.)



Different Logic Gates Circuit Diagrams

In electronics, a **logic gate** is an idealized or physical device implementing a Boolean function; that is, it performs a logical operation on one or more logical inputs, and produces a single logical output. Depending on the context, the term may refer to an **ideal logic gate**, one that has for instance zero rise time and unlimited fan-out, or it may refer to a non-ideal physical device[1] (see Ideal and real op-amps for comparison).

Logic gates are primarily implemented using diodes or transistors acting as electronic switches, but can also be constructed using vacuum tubes, electromagnetic relays (relay logic), fluidic logic, pneumatic logic, optics, molecules, or even mechanical elements. With amplification, logic gates can be cascaded in the same way that Boolean functions can be composed, allowing the construction of a physical model of all of Boolean logic, and therefore, all of the algorithms and mathematics that can be described with Boolean logic.

Logic circuits include such devices as multiplexers, registers, arithmetic logic units (ALUs), and computer memory, all the way up through complete microprocessors, which may contain more than 100 million gates. In modern practice, most gates are made from field-effect transistors (FETs), particularly MOSFETs (metal–oxide–semiconductor field-effect transistors).

Compound logic gates AND-OR-Invert (AOI) and OR-AND-Invert (OAI) are often employed in circuit design because their construction using MOSFETs is simpler and more efficient than the sum of the individual gates.

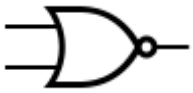
In reversible logic, Toffoli gates are used.

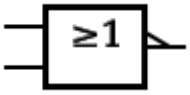
UNIVERSAL GATES

NAND

		OUTPUTS
A	B	Y=A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR





INPUTS		OUTPUTS
A	B	Y=A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

ELEMENTORY LOGIC GATES

AND

INPUTS		OUTPUTS
A	B	Y=A NOR B
0	0	0
0	1	0
1	0	0
1	1	1

OR

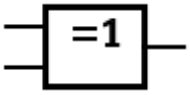
INPUTS		OUTPUTS
A	B	Y=A NOR B
0	0	0
0	1	1
1	0	1
1	1	1

NOT

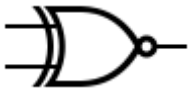
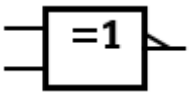
INPUTS	OUTPUTS
A	Y=A NOT
0	1
1	0

EXCLUSIVE GATES

EX-OR



INPUTS		OUTPUTS
A	B	Y=A EX-OR B
0	0	0
0	1	1
1	0	1
1	1	0

EX-NOR			

INPUTS		OUTPUTS
A	B	Y=A EX-NOR B
0	0	1
0	1	0
1	0	0
1	1	1

Truth-Tables, Boolean Equations

A truth table shows how a logic circuit's output responds to various combinations of the inputs, using logic 1 for true and logic 0 for false. All permutations of the inputs are listed on the left, and the output of the circuit is listed on the right. The desired output can be achieved by a combination of logic gates. A truth table for two inputs is shown, but it can be extended to any number of inputs. The input columns are usually constructed in the order of binary counting with a number of bits equal to the number of inputs.

Truth tables are an important tool for evaluating statements and arguments. We can create our own truth tables using following steps:

1. Translate statements of ordinary language.
2. Break all complex statements into smaller parts.
3. Determine how many columns are required.
4. Determine how many rows are required.
5. Determine the truth values of statement letters.
6. Determine the truth values of complex statements.

A law of Boolean algebra is an identity such as $x \vee (y \vee z) = (x \vee y) \vee z$ between two Boolean terms, where a Boolean term is defined as an expression built up from variables and the constants 0 and 1 using the operations $\wedge$, $\vee$, and $\neg$. The concept can be extended to terms involving other Boolean operations such as $\oplus$, $\rightarrow$, and $\equiv$, but such extensions are unnecessary for the purposes to which the laws are put. Such purposes include the definition of a Boolean algebra as any model of the Boolean laws, and as a means for deriving new laws from old as in the derivation of $x \vee (y \wedge z) = x \vee (z \wedge y)$ from $y \wedge z = z \wedge y$ as treated in the section on axiomatization.

Monotone laws

Boolean algebra satisfies many of the same laws as ordinary algebra when one matches up $\vee$ with addition and $\wedge$ with multiplication. In particular the following laws are common to both kinds of algebra.

Associativity of $\vee$	$x \vee (y \vee z) = (x \vee y) \vee z$
Associativity of $\wedge$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Commutativity of $\vee$	$x \vee y = y \vee x$
Commutativity of $\wedge$	$x \wedge y = y \wedge x$
Distributivity of $\wedge$ over $\vee$	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
Distributivity of $\vee$ over $\wedge$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Identity for $\vee$	$x \vee 0 = x$
Identity for $\wedge$	$x \wedge 1 = x$
Annihilator for $\wedge$	$x \wedge 0 = 0$
Annihilator for $\vee$	$x \vee 1 = 1$
Idempotence of $\vee$	$x \vee x = x$
Idempotence of $\wedge$	$x \wedge x = x$
Absorption 1	$x \wedge (x \vee y) = x$
Absorption 2	$x \vee (x \wedge y) = x$

A consequence of the first of these laws is 1∨1 = 1, which is false in ordinary algebra, where 1+1 = 2. Taking x = 2 in the second law shows that it is not an ordinary algebra law either, since 2×2 = 4. The remaining four laws can be falsified in ordinary algebra by taking all variables to be 1, for example in Absorption Law 1 the left hand side is 1(1+1) = 2 while the right hand side is 1, and so on.

All of the laws treated so far have been for conjunction and disjunction. These operations have the property that changing either argument either leaves the output unchanged or the output changes in the same way as the input. Equivalently, changing any variable from 0 to 1 never results in the output changing from 1 to 0. Operations with this property are said to be **monotone**. Thus the axioms so far have all been for monotonic Boolean logic. No monotonicity enters via complement ¬ as follows.

Non-monotone laws

The complement operation is defined by the following two laws.

Complementation 1 $x \wedge \neg x = 0$

Complementation 2 $x \vee \neg x = 1$

All properties of negation including the laws below follow from the above two laws alone.

In both ordinary and Boolean algebra, negation works by exchanging pairs of elements, whence in both algebras it satisfies the double negation law (also called involution law)

Double negation $\neg(\neg x) = x$

But whereas *ordinary algebra* satisfies the two laws

$$(-x)(-y) = xy$$
$$(-x) + (-y) = -(x + y)$$

Boolean algebra satisfies De Morgan's laws:

$$\text{De Morgan 1} \quad \neg x \wedge \neg y = \neg(x \vee y)$$
$$\text{De Morgan 2} \quad \neg x \vee \neg y = \neg(x \wedge y)$$

Completeness

The laws listed above define Boolean algebra, in the sense that they entail the rest of the subject. The laws *Complementation* 1 and 2, together with the monotone laws, suffice for this purpose and can therefore be taken as one possible *complete* set of laws or axiomatization of Boolean algebra. Every law of Boolean algebra follows logically from these axioms. Furthermore, Boolean algebras can then be defined as the models of these axioms as treated in the section thereon.

To clarify, writing down further laws of Boolean algebra cannot give rise to any new consequences of these axioms, nor can it rule out any model of them. In contrast, in a list of some but not all of the same laws, there could have been Boolean laws that did not follow from those on the list, and moreover there would have been models of the listed laws that were not Boolean algebras.

This axiomatization is by no means the only one, or even necessarily the most natural given that we did not pay attention to whether some of the axioms followed from others but simply chose to stop when we noticed we had enough laws, treated further in the section on axiomatizations. Or the intermediate notion of axiom can be sidestepped altogether by defining a Boolean law directly as any **tautology**, understood as an equation that holds for all values of its variables over 0 and 1. All these definitions of Boolean algebra can be shown to be equivalent.

Boolean algebra has the interesting property that $x = y$ can be proved from any non-tautology. This is because the substitution instance of any non-tautology obtained by instantiating its variables with constants 0 or 1 so as to witness its non-tautologyhood reduces by equational reasoning to 0 = 1. For example, the non-tautologyhood of $x \wedge y = x$ is witnessed by $x = 1$ and $y = 0$ and so taking this as an axiom would allow us to infer $1 \wedge 0 = 1$ as a substitution instance of the axiom and hence 0 = 1. We can then show $x = y$ by the reasoning $x = x \wedge 1 = x \wedge 0 = 0 = 1 = y \vee 1 = y \vee 0 = y$.

Duality principle

Principle: If {X,R} is a poset, then {X,R(inverse)} is also a poset.

There is nothing magical about the choice of symbols for the values of Boolean algebra. We could rename 0 and 1 to say α and β, and as long as we did so consistently throughout it would still be Boolean algebra, albeit with some obvious cosmetic differences.

But suppose we rename 0 and 1 to 1 and 0 respectively. Then it would still be Boolean algebra, and moreover operating on the same values. However it would not be identical to our original Boolean algebra because now we find ∨ behaving the way ∧ used to do and vice versa. So there

are still some cosmetic differences to show that we've been fiddling with the notation, despite the fact that we're still using 0s and 1s.

But if in addition to interchanging the names of the values we also interchange the names of the two binary operations, *now* there is no trace of what we have done. The end product is completely indistinguishable from what we started with. We might notice that the columns for $x \wedge y$ and $x \vee y$ in the truth tables had changed places, but that switch is immaterial.

When values and operations can be paired up in a way that leaves everything important unchanged when all pairs are switched simultaneously, we call the members of each pair **dual** to each other. Thus 0 and 1 are dual, and $\wedge$ and $\vee$ are dual. The **Duality Principle**, also called De Morgan duality, asserts that Boolean algebra is unchanged when all dual pairs are interchanged.

One change we did not need to make as part of this interchange was to complement. We say that complement is a **self-dual** operation. The identity or do-nothing operation x (copy the input to the output) is also self-dual. A more complicated example of a self-dual operation is $(x \wedge y) \vee (y \wedge z) \vee (z \wedge x)$. There is no self-dual binary operation that depends on both its arguments. A composition of self-dual operations is a self-dual operation. For example, if $f(x,y,z) = (x \wedge y) \vee (y \wedge z) \vee (z \wedge x)$, then $f(f(x,y,z),x,t)$ is a self-dual operation of four arguments x,y,z,t.

The principle of duality can be explained from a group theory perspective by fact that there are exactly four functions that are one-to-one mappings (automorphisms) of the set of Boolean polynomials back to itself: the identity function, the complement function, the dual function and the contradual function (complemented dual). These four functions form a group under function composition, isomorphic to the Klein four-group, acting on the set of Boolean polynomials. Walter Gottschalk remarked that consequently a more appropriate name for the phenomenon would be the *principle (or square) of quaternality*.

Combination Logic Circuits

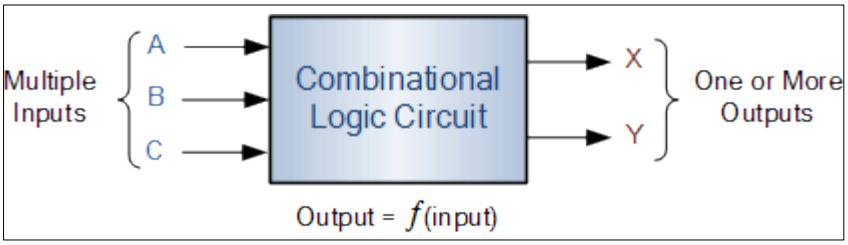
Unlike Sequential Logic Circuits whose outputs are dependent on both their present inputs and their previous output state giving them some form of Memory, the outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependent at all times on the combination of its inputs. So if one of its inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have "no memory", "timing" or "feedback loops" within their design.

Combinational Logic

Combinational Logic Circuits are made up from basic logic **NAND**, **NOR** or **NOT** gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of Combinational Logic Circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into

a number of different output lines, one at a time producing an equivalent decimal code at its output.

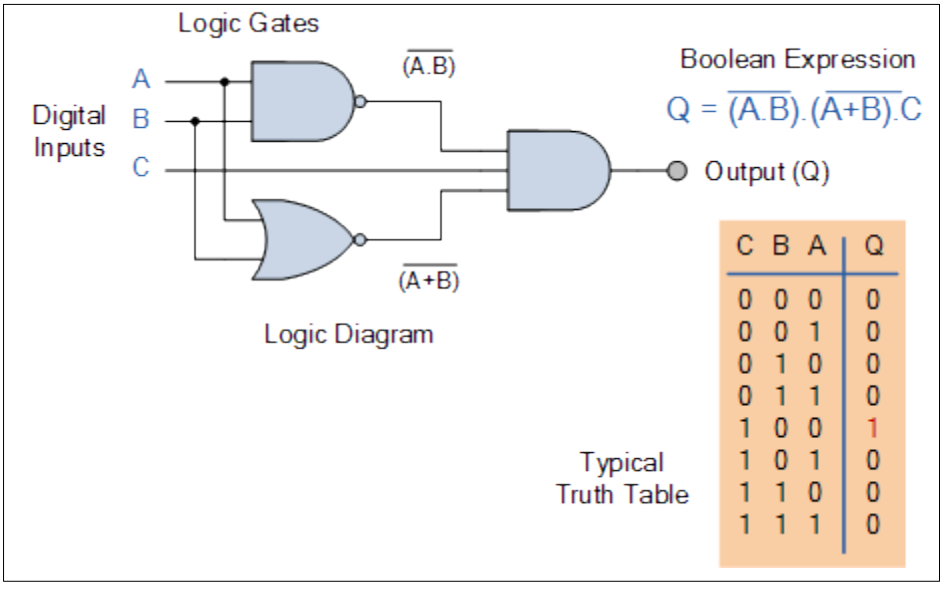


Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.

The three main ways of specifying the function of a combinational logic circuit are:

7. **Boolean algebra** – This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic "1" output.
8. **Truth Table** – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
9. **Logic Diagram** – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol that implements the logic circuit.

And all three of these logic circuit representations are shown below.



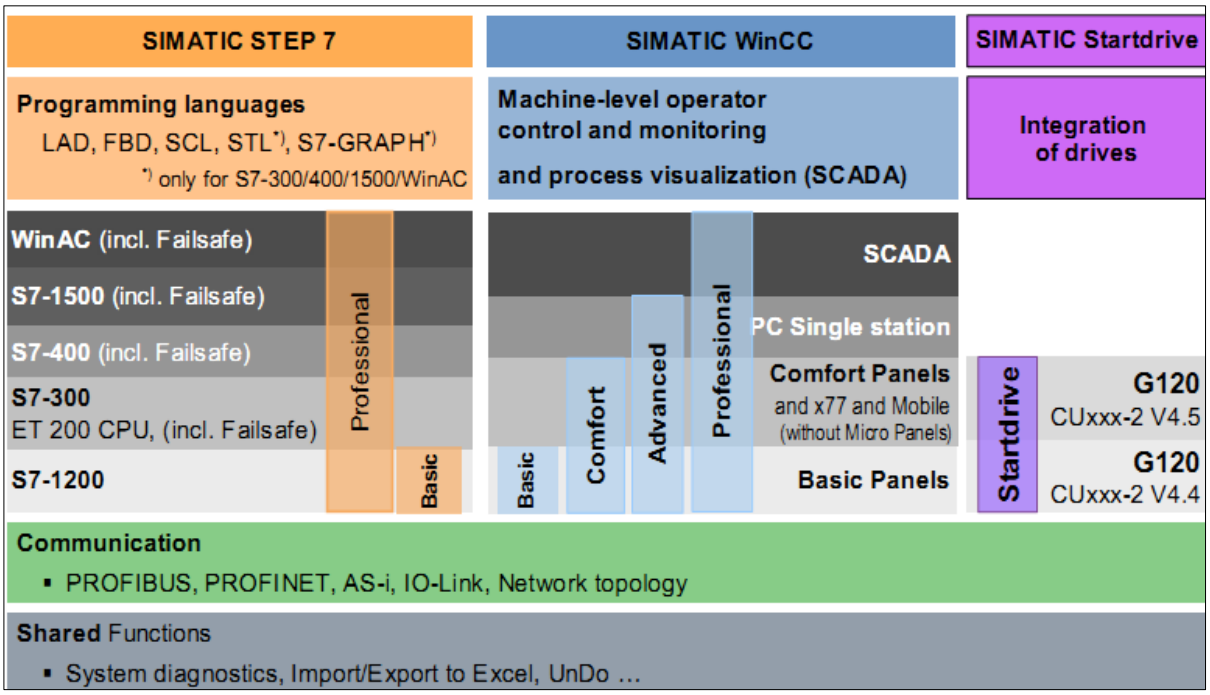
As combinational logic circuits are made up from individual logic gates only, they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application include **Multiplexers**, **De-multiplexers**, **Encoders**,**Decoders**, **Full** and **Half Adders** etc.

Engineering Software TIA Portal

TIA PORTAL-Central Engineering Framework

With the Totally Integrated Automation Portal (TIA Portal), Siemens follows a vision of providing an engineering framework for implementing automation solutions in all industries around the globe. From designing, commissioning, operating and maintaining to upgrading automation systems, TIA Portal saves engineering time, cost, and effort.

Scope of the Products



STEP S7 Range of Products

IMATIC STEP 7 is the world's best known and most widely used engineering software in industrial automation. And: STEP 7 is standard-compliant.

SIMATIC STEP 7 in the TIA Portal – the engineering system in the Totally Integrated Automation Portal – continues the success story of SIMATIC STEP 7. With SIMATIC STEP 7 in the TIA Portal, users can configure, program, test, and diagnose all modular and PC-based SIMATIC controllers.

Using the SIMATIC STEP 7 Safety Advanced option, you can exploit all the advantages of the TIA Portal for your fail-safe automation as well. All configuration and programming tools required for generating a safety-oriented program are integrated into the STEP 7 user interface and use a common project structure.

Benefits

- Powerful programming editors for efficient engineering
- Scalability across all series of controllers
- Optimum interaction between the controller, HMI and drive in a working environment
- Shared data management and uniform symbols
- System diagnostics as an integral component
- Variables trace for effective commissioning
- Scalable and flexible motion control functionality
- Comprehensive library concept
- Security Integrated
- Migration support for existing hardware and software products

Application

SIMATIC STEP 7 Professional V13 SP1 is the easy-to-use, integrated engineering system for the current SIMATIC controllers S7-1200, S7-1500, S7-300, S7-400, WinAC, software controllers, and ET 200 CPU. PLCSIM for simulation of S7-1200/1500 CPU and WinCC Basic for configuration of Basic Panels are included in the scope of delivery.

SIMATIC STEP 7 Basic V13 SP1, the easy-to-use engineering system for the modular SIMATIC S7-1200 micro PLC, as well as the associated I/O. It contains PLCSIM for simulation purposes and SIMATIC WinCC Basic for configuring the SIMATIC Basic Panels.

STEP 7 V13 SP1 thus provides support in all phases of the automation project:

- Configuring and parameterizing the hardware
- Specifying the communication
- Programming in IEC programming editors
- Configuration of the visualization
- Test, commissioning and service

WinCC Range of Products

Family of configuration systems with WinCC Basic, Comfort, Advanced and Professional for SIMATIC operator panels, as well as for the PC-based visualization systems WinCC Runtime Advanced and WinCC Runtime Professional

10. SIMATIC WinCC Runtime Advanced visualization software

- PC-based HMI solution for single-user systems directly at the machine

- Basic package for visualization, reporting and logging, user administration, can be expanded flexibly with VB scripts
- Basic package expandable by means of option packages
- Integration of customer-specific ActiveX Controls created with WinCC Control Development
- Can be integrated into automation solutions based on TCP/IP networks
- Expanded service concepts with remote operation, diagnostics and administration over the Intranet and Internet in combination with email communication

11. SIMATIC WinCC Runtime Professional visualization software

- PC-based operator control and monitoring system for visualization and operator control of processes, production flows, machines and plants in all sectors – from the simple single-user station through to distributed multi-user systems and cross-location solutions with web clients. WinCC Runtime Professional is the information hub for corporation-wide vertical integration.
- Industry-standard functions for signaling and acknowledging events, archiving of messages and measured values, logging of all process and configuration data, user administration, can be expanded flexibly with VB and C scripts
- Basic package expandable by means of option packages
- Also included are APIs for the Runtime to utilize the open programming interfaces
- Integration of customer-specific ActiveX Controls created with WinCC Control Development

OVERVIEW

- Integrated family of engineering tools for configuring SIMATIC HMI operator panels, as well as for the PC-based visualization systems WinCC Runtime Advanced and WinCC Runtime Professional.
- WinCC (TIA Portal) is based on the new central engineering framework Totally Integrated Automation Portal (TIA Portal), which offers the user a uniform, efficient and intuitive solution to all automation tasks.
- WinCC (TIA Portal) also offers uniform engineering from the Basic Panel through to SCADA applications.
- Together with the STEP 7 (TIA Portal) products, WinCC (TIA Portal) forms the optimum solution for integrated, efficient engineering.

Current version:

- SIMATIC WinCC Basic V13 SP1
- SIMATIC WinCC Comfort V13 SP1
- SIMATIC WinCC Advanced V13 SP1
- SIMATIC WinCC Professional V13 SP1

Benefits

- The integrated configuration software reduces training, maintenance and service overhead and protects the customer's investments.

- Minimized engineering overhead and reduction of lifecycle costs thanks to Totally Integrated Automation (TIA)
- Minimized configuration overhead due to reuse of scalable and dynamizable objects
- Intelligent tools for efficient and simple configuration:
    - Wizard for defining the basic structure of the HMI project
    - Table-based editors simplify the generation and processing of similar types of object, e.g. for tags, texts, or alarms.
    - Complex configuration tasks such as the definition of paths of motion or the creation of the fundamental operator prompting are simplified by means of graphical configuration.
- Comprehensive support of multi-language configurations for worldwide use
    - Selectable views for entering configuration data in several languages
    - System and user-specific text lexicons
    - Export/import of language-dependent texts
- Investment protection due to
    - Import of the configuration from WinCC flexible 2008 SP2 and 2008 SP3
    - Transfer of the configuration from WinCC V7.0 SP3

Application

SIMATIC WinCC in the editions Basic, Comfort, Advanced and Professional are innovative engineering tools for configuring SIMATIC HMI operator panels, as well as for the PC-based visualization systems WinCC Runtime Advanced and WinCC Runtime Professional.

Depending on the selected product, various target systems can be configured:

WinCC Basic

- Basic Panels (1st Generation): KP300 Basic, KTP400 Basic, KTP600 Basic, KTP1000 Basic, TP1500 Basic
- Basic Panels (2nd Generation): KTP400 Basic, KTP700 Basic, KTP900 Basic, KTP1200 Basic

WinCC Comfort

As WinCC Basic, plus:

- Comfort Panels
- Mobile Panels: Mobile Panel 177, Mobile Panel 277
- Panels of the 70 series: OP 73, OP 77A, OP 77B
- Panels of the 170 series: TP 177A, TP 177B, OP 177B
- Panels of the 270 series: TP 277, OP 277
- Multi Panels: MP 177, MP 277, MP 377

WinCC Advanced

As WinCC Comfort, plus:

- SIMATIC PCs with WinCC Runtime Advanced:

  - SIMATIC Rack PC: Rack PC 547B, IPC547C, IPC547D, Rack PC IPC647C, IPC647D, Rack PC IPC847C, IPC847D
  - SIMATIC Box PC: IPC227D, Box PC 427B, IPC427C, IPC427D, Box PC 627B, IPC627C, Box PC 827B, IPC827C
  - SIMATIC Panel PC: IPC277D, Panel PC 477B, IPC477C, IPC477D, Panel PC 577B, IPC577C, Panel PC 677B, IPC677C, IPC677D
  - SIMATIC modular Embedded Controller: EC31
  - Industrial Flat Panel (Multi-Touch)
- Standard PC with WinCC Runtime Advanced
- SINUMERIK PC: PCU 50.3, PCU 50.5

WinCC Professional

As WinCC Advanced, plus:

- SIMATIC PCs with WinCC Runtime Professional:
  - SIMATIC Rack PC: Rack PC 547B, IPC547C, IPC547D, IPC547E, Rack PC 647B, IPC647C, Rack PC 847B, IPC847C, IPC647D, IPC847D
  - SIMATIC Box PC: IPC427C, IPC427D, Box PC 627B, IPC627C, IPC827C, IPC627D
  - SIMATIC Panel PC: IPC477C, IPC477D, Panel PC 577B, IPC577C, Panel PC 677B, IPC677C, IPC677D
  - Industrial Flat Panel (Multi-Touch)
- Standard PC with WinCC Runtime Professional

Design

The functionalities of the engineering tools of the SIMATIC WinCC family are based on each other. The available editors are largely determined by the respective configurable target systems and their function. A more comprehensive engineering tool such as WinCC Advanced can always be used to configure lower-level target devices as well (e.g. Basic Panels)

A Power pack can be used to upgrade from a smaller edition to a larger one. This does not apply to WinCC Basic.

The functionality of WinCC engineering tools already contains the configuration support of the available Runtime options for SIMATIC Panels, WinCC Runtime Advanced or WinCC Runtime Professional, irrespective of the purchased RT licenses. A separate license is required for the target system when using the configured Runtime options.

Start drive Range of Products

SINAMICS Startdrive is a tool for configuring, commissioning, and diagnosing the SINAMICS family of drives and is integrated into the TIA Portal.

SINAMICS Startdrive can be used to implement drive applications involving the following inverters:

- SINAMICS G120
- SINAMICS G120C

- SINAMICS G120D
- SINAMICS G120P
- SINAMICS G110M

The SINAMICS Startdrive commissioning tool has been optimized with regard to user friendliness and consistent use of the TIA Portal benefits of a common working environment for PLC, HMI and drives.

All of the available Control Units from SINAMICS Firmware V4.4 are supported for these devices (including PROFINET, PROFIBUS, Safety Integrated). All combinable Power Modules up to 400 kW can be configured.

Benefits

Efficient commissioning with easy configuration and powerful tools:

- High degree of usability thanks to task-based navigation through the engineering workflow
    - Hardware configuration
    - Parameterization
    - Commissioning
    - Diagnostics
- Time-saving and guided step-by-step commissioning
- User-friendly graphic function view for all drive functions
- List of drive parameters structured according to functions
- Easy integration of SIMOTICS motors
- Integrated control panel for direct operation of the inverter from the TIA Portal
- Powerful real-time trace for commissioning and drive diagnostics
- Intuitive and efficient inverter diagnostics through automatic display of messages
- Context-sensitive online help, e.g. for drive messages
- Integrated detailed inverter diagnostic functions
    - Control/status words
    - Parameter status
    - Operating conditions
    - Communication states
- Simple configuration for drive-end Safety Integrated and the drive-internal basic positioning function (EPos)
- Graphic configuration of drive-internal free function blocks (FFB)
- Online work on the inverter
    - Without previous creation of an offline project
    - With new SINAMICS firmware (e.g. V4.7), without having to perform a tool update
    - Available online functions without project: Commissioning with wizard and control panel, full parameter access with graphic function view and structured parameter list with complete inverter diagnostics

SIMATIC licenses at a glance

When using SIMATIC software you have a variety of licensing options tailored to your individual needs and preferred use.

License type	Trial version	Rental version	Full version	Full version
Scope of functions	full	full	full	full
Term	e.g. 21 days	365 days	unlimited	unlimited
Permitted installation units	1 PC	1 PC	1 PC	multiple PCs
License storage	local	local	local	local / server

Trial – a license to try us out

With a Trial license you can install the corresponding SIMATIC software product on one computer. After the first time you launch the program you can use it without restrictions for a limited time (e.g. 21 days), for testing and evaluation purposes – however it is not meant for production. We are not subject to liability of any kind for this type of license. Once the Trial license expires, the software cannot be re-launched till you download and apply a corresponding Floating or Single License key for the same version. There is no need to reinstall the software.

When is a Trial version the right one for you?

- You want to try out the software before you decide to buy.
- You want to use the software to evaluate internal processes.
- You don’t want any compromises on functionality or performance during the testing and evaluation phase.
- Download SIMATIC software Trial versions

Automation License Manager

The Automation License Manager (ALM) makes managing all your licenses especially efficient, and as easy as pie. It reduces management effort and expense and provides transparency about the software you’re using, right down to the machine level.

Manage licenses locally or with server support

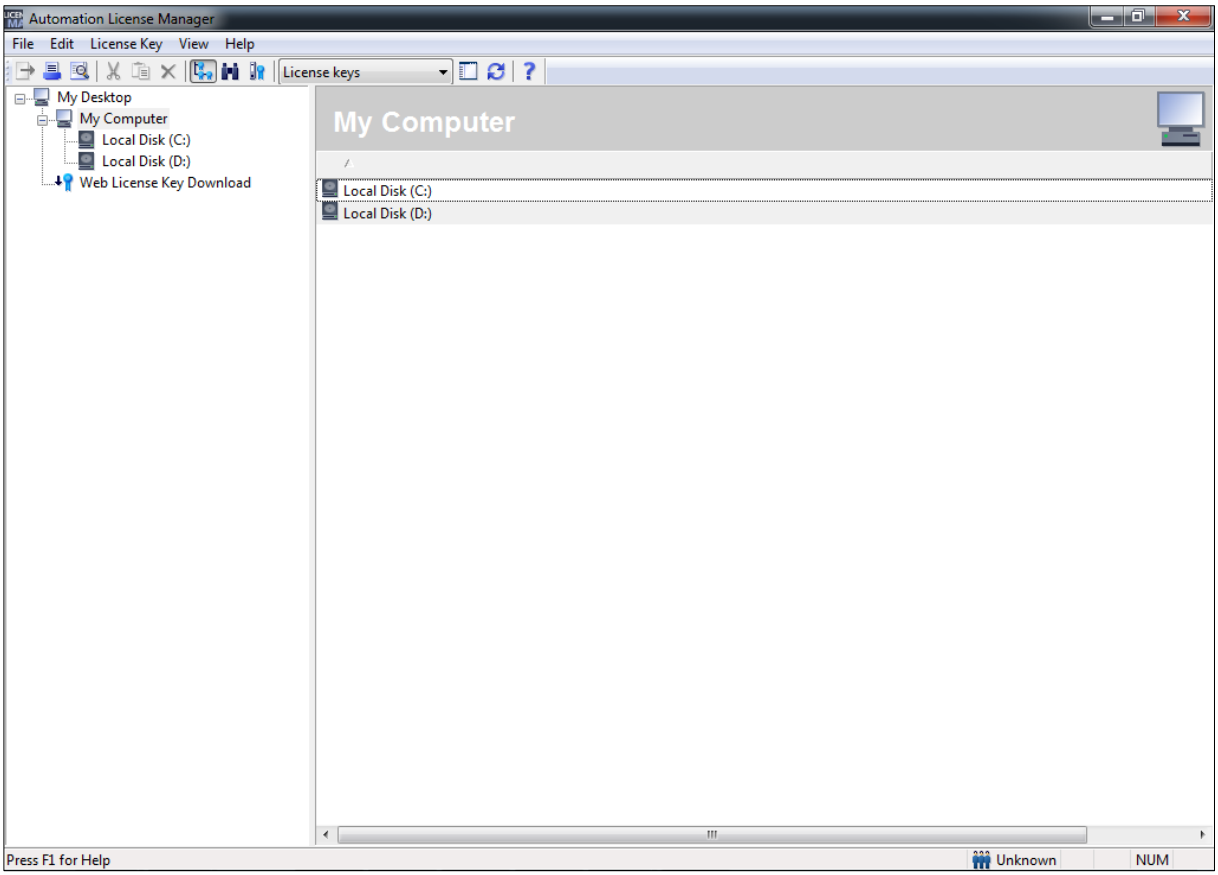
- You can use your licenses within the company network or locally

- License overview

- ALM gives you a very convenient overview of all the licenses available

- Get licenses online

- You can download licenses easily via drag & drop from the Siemens Online Software Delivery (OSD) platform

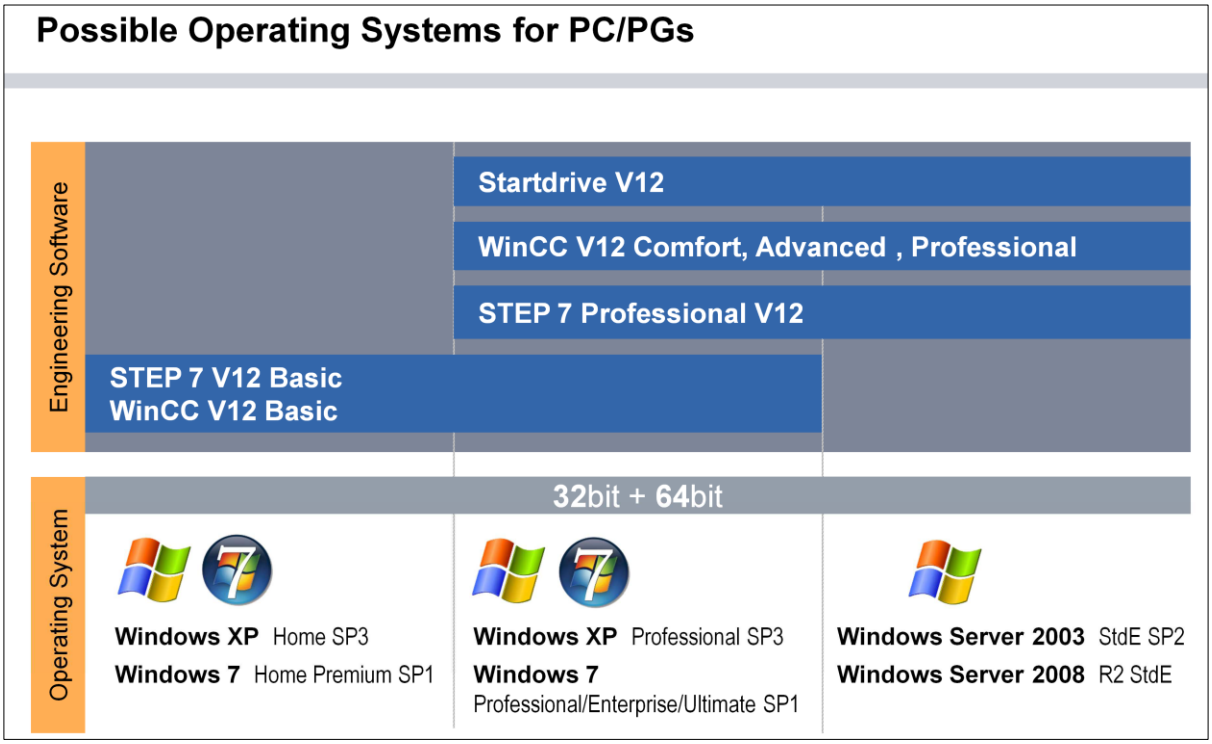


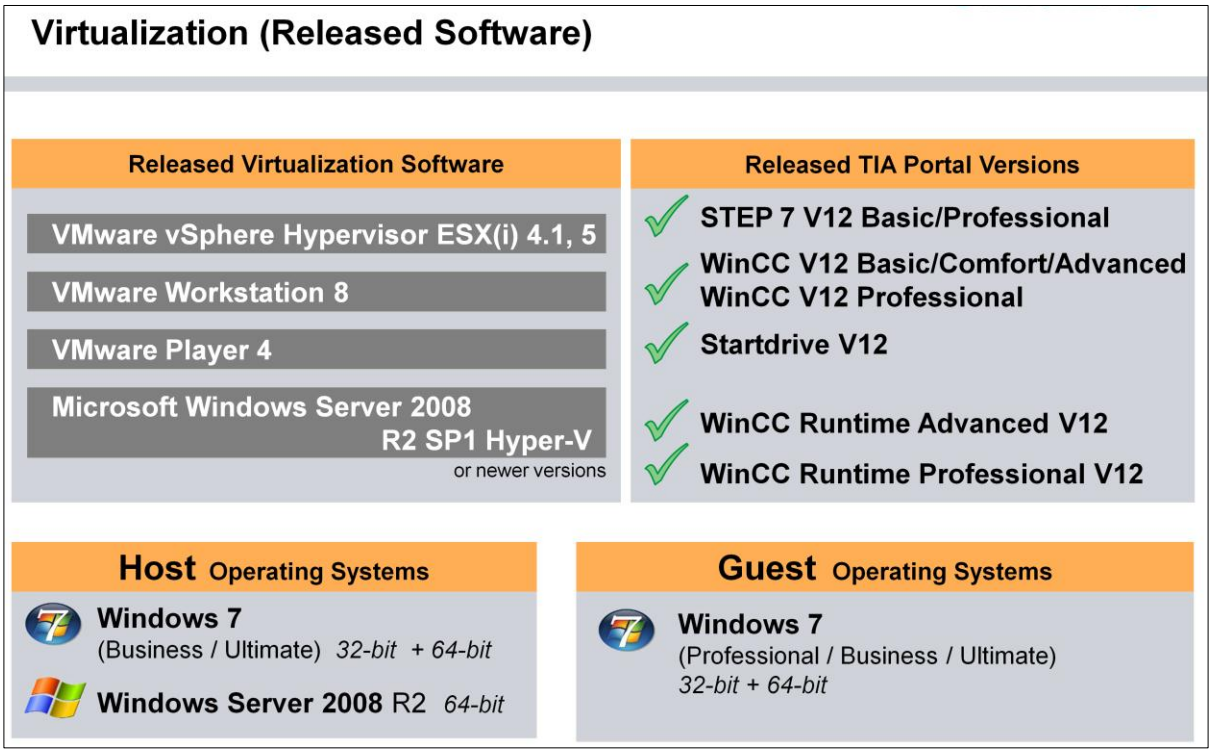
The Automation License Manager is an integral part of every SIMATIC software product, and provides a complete overview of all your available licenses. It makes it faster and easier than ever before to manage existing licenses and get new ones.

You can download SIMATIC software directly through the Automation License Manager. Transfer licenses directly by drag & drop from the Online Software Delivery (OSD) platform to your hard disk.

Download the Automation License Manager from the Service & Support Portal

Operating Systems for PG/PCs





Parallel Installation “Side-by-Side”

Compatibility of STEP 7 with other SIMATIC products

STEP 7 Professional / Basic V13 SP1 (incl. WinCC Basic V13 SP1) can be installed on a PC in parallel with other versions of STEP 7 V12, V5.4 or V5.5, STEP 7 Micro/WIN, WinCC flexible (from 2008), S7-PCT (from V3.3) and WinCC (from V7.0 SP2).

Compatibility of WinCC with other SIMATIC products

Side-by-side installation with all other SIMATIC products

- WinCC Basic/Comfort/Advanced V13 can be installed on one computer in parallel with STEP 7 V5.4 or V5.5, STEP 7 Micro/WIN, STEP 7 V10.5/V11/V12, WinCC V11/V12, WinCC flexible (2008 and higher) and WinCC (V7.0 SP3 and higher)..
- WinCC Professional V13 can be installed on one computer in parallel with STEP 7 V5.4 or V5.5, STEP 7 V10.5/V11/V12, WinCC V11/V12 (except WinCC Professional), WinCC flexible (2008 and higher). The parallel installation with WinCC V6/V7 is not permitted.
- The following virus scanners have been tested with WinCC V13:
    - Symantec Endpoint Protection 12.1
    - Trend Micro Office Scan Corporate Edition 10.6
    - McAfee Virus Scan Enterprise 8.8
    - Kaspersky Anti-Virus 2014
    - Windows Defender (Windows 8.1 and higher)

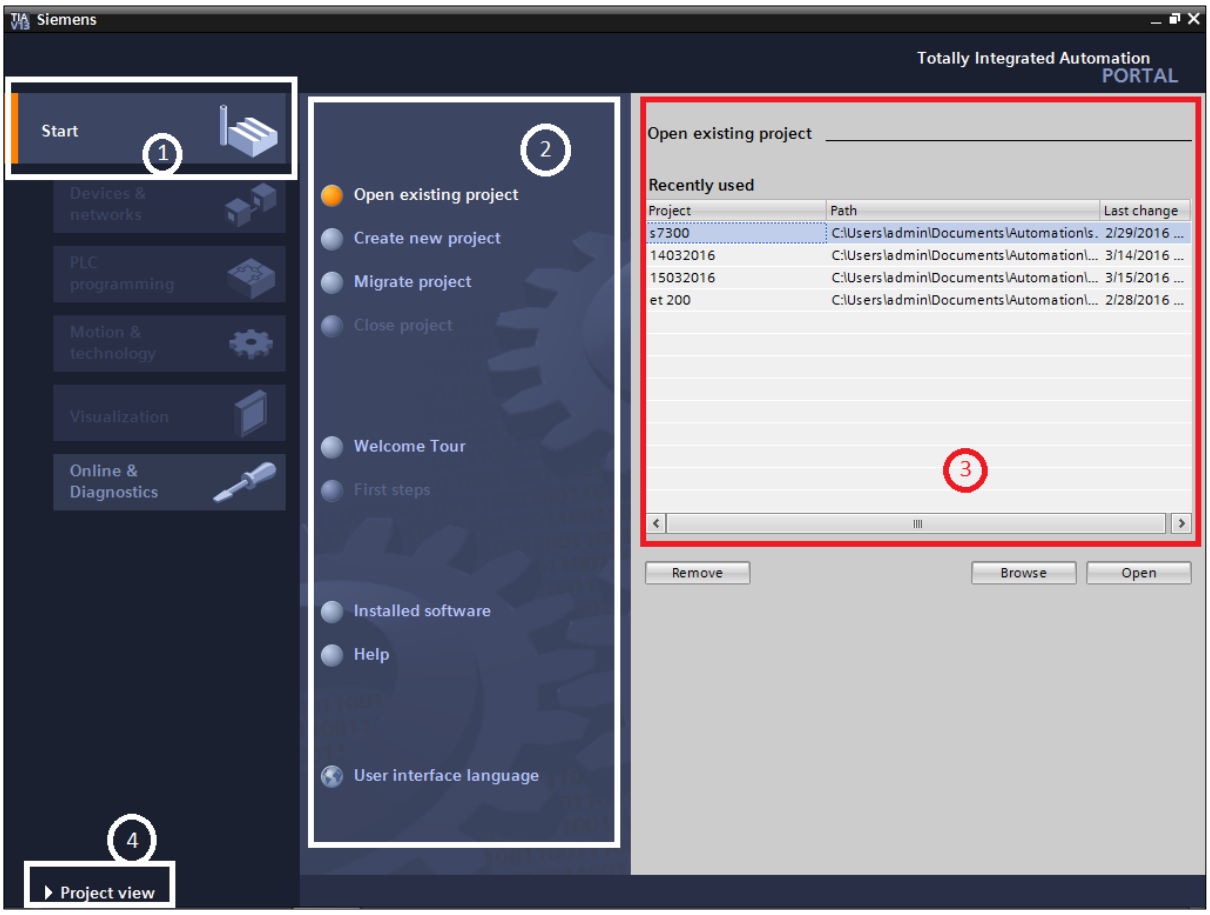
Compatibility of StartDrive with other products

- SINAMICS Startdrive can be installed alongside STARTER
- SINAMICS Startdrive V13 operates with STEP 7 Basic/Professional V13 and WinCC V13 in a framework
- SINAMICS Startdrive V13 can be installed on a computer alongside other versions of Startdrive V12, STEP 7 V12, V5.4 or V5.5, STEP 7 Micro/WIN, WinCC flexible (2008 and above) and WinCC (V7.0 SP2 and above)
- Supported virtualization platforms:
    - VMware Workstation 10
    - VMware Player 6.0
    - Microsoft Windows Server 2012 R2 Hyper-V
- SINAMICS Startdrive has been tested with the following virus scanners:
    - Symantec Endpoint Protection 12.1
    - Trend Micro Office Scan Corporate Edition 10.6
    - Kaspersky Anti-Virus 2014
    - Windows Defender (Windows version 8.1 and above)

TIA PORTAL: PORTAL VIEW & PROJECT VIEW

STEP 7 Basic provides a user-friendly environment to develop controller logic, configure HMI visualization, and setup network communication. To help increase your productivity, STEP 7 Basic provides two different views of the project: a task-oriented set of portals that are organized on the functionality of the tools (Portal view), or a project-oriented view of the elements within the project (Project view). Choose which view helps you work most efficiently. With a single click, you can toggle between the Portal view and the Project view.

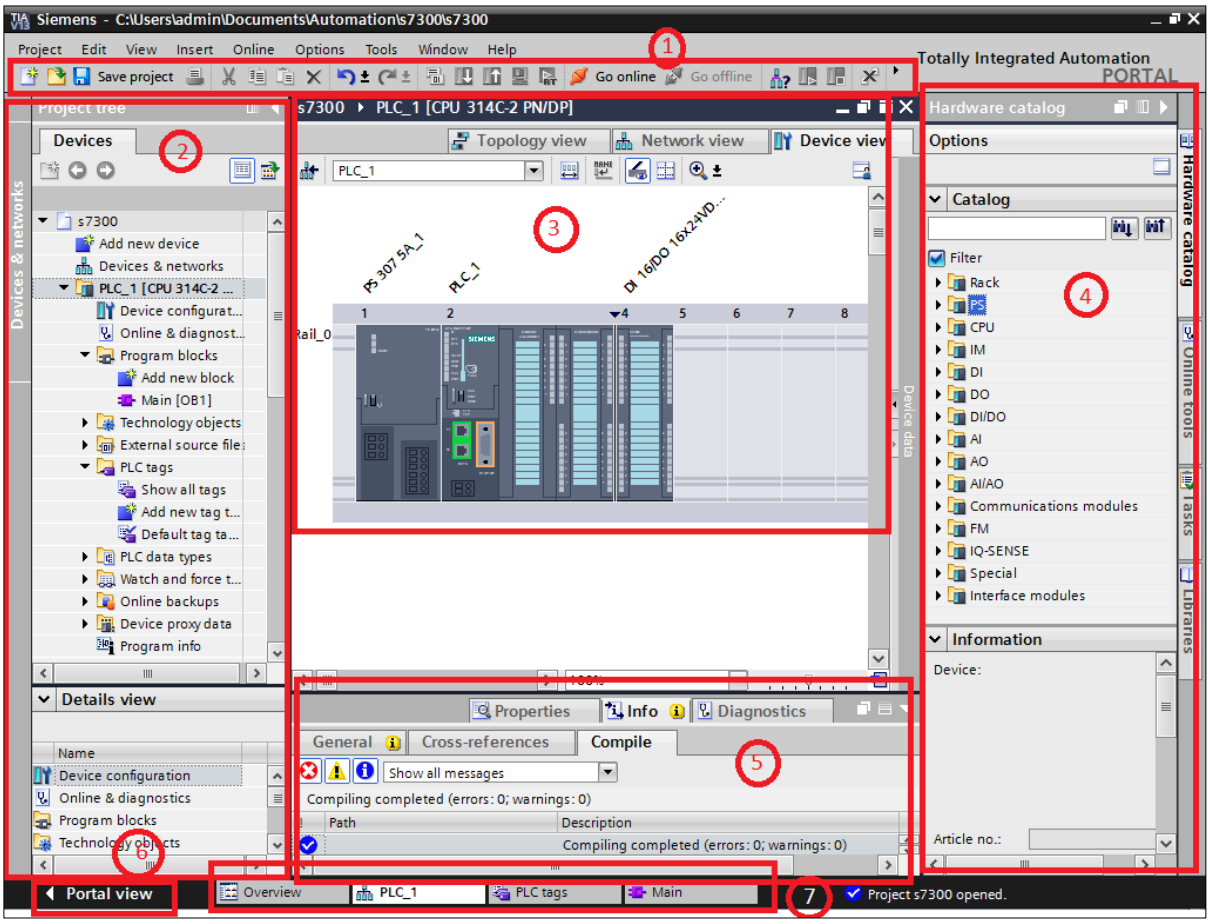
The Portal view



The Portal view provides a functional view of the project tasks and organizes the tools according to the tasks to be accomplished. You can easily determine how to proceed and which task to choose.

1. Portals for the different tasks
2. Tasks for the selected portal
3. Selection panel for the selected action
4. Changes to the Project view

The Project view



The Project view provides access to all of the components within a project.

1. Menus and toolbar
2. Project navigator
3. Work area
4. Task cards
5. Inspector window
6. Changes to the Portal view
7. Editor bar

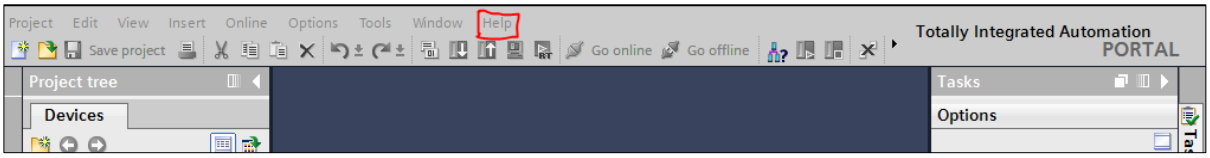
With all of these components in one place, you have easy access to every aspect of your project. For example, the inspector window shows the properties and information for the object that you have selected in the work area. As you select different objects, the inspector window displays the properties that you can configure. The inspector window includes tabs that allow you to see diagnostic information and other messages.

By showing all of the editors that are open, the editor bar helps you work more quickly and efficiently. To toggle between the open editors, simply click the different editor. You can also arrange two editors to appear together, arranged either vertically or horizontally. This feature allows you to drag and drop between editors.

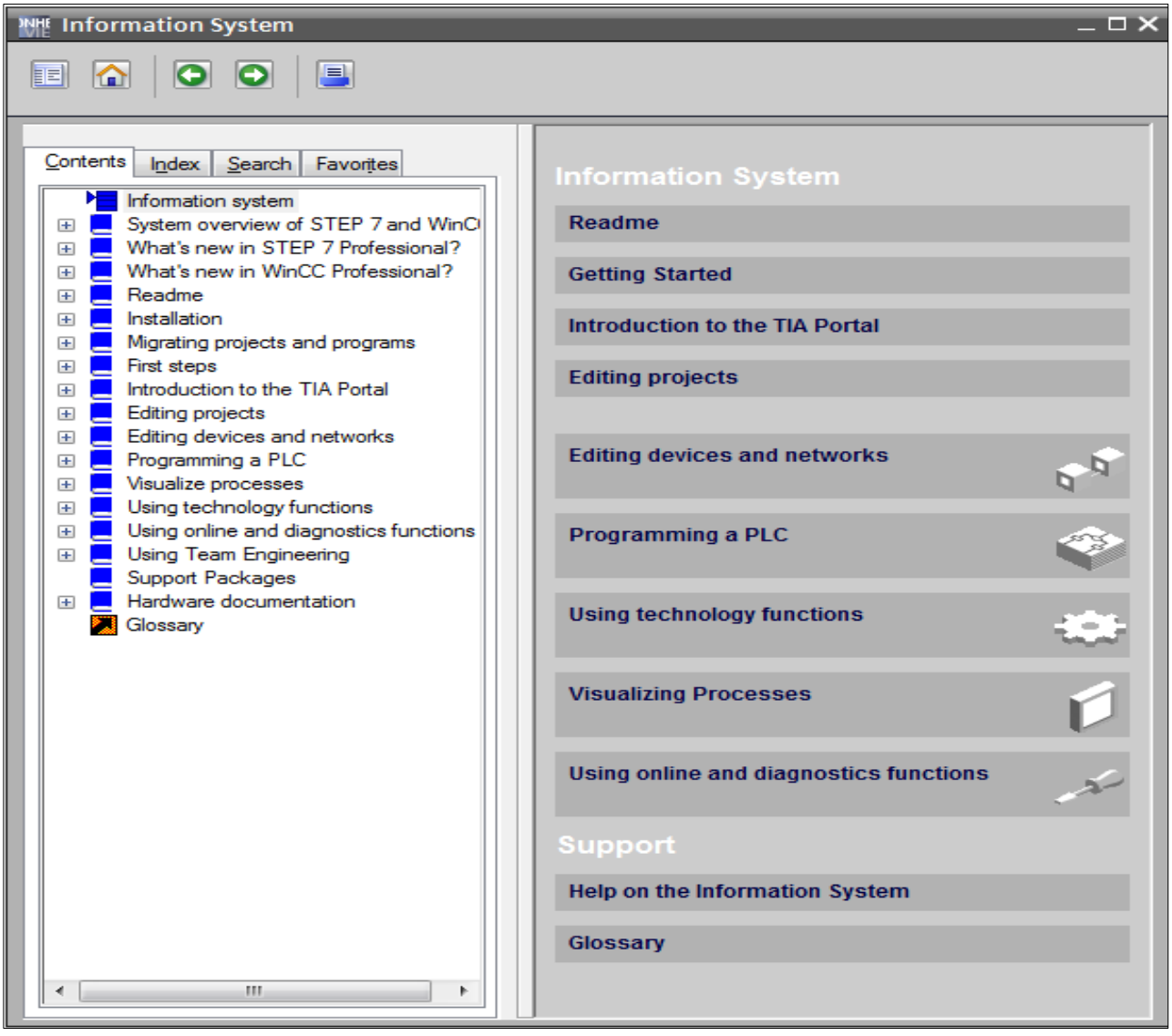
HELP functions

To help you to find more information or to resolve issues quickly and efficiently, STEP 7 Basic provides intelligent point-of-need assistance. Hovering over an element of the software interface displays the tool tip. Some of the tool tips in the interface "cascade" to provide additional information and even include a link to a specific topic in the online information system. A black triangle alongside the tool tip signifies that more information is available.

Help is always just a click away! From the Portal view, select the Start portal and click the "Help" command. From the Project view, select the "Show help" command in the "Help" menu.



The information system opens in a window that does not obscure the work areas. To undock the help window and display the contents, click the "Show/hide contents" button. You can then resize the help window.

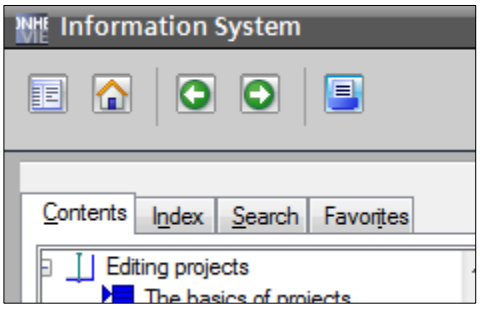


If STEP 7 Basic is maximized, clicking the "Show/hide contents" button does not undock the help window.

Click the "Restore down" button to undock the help window.

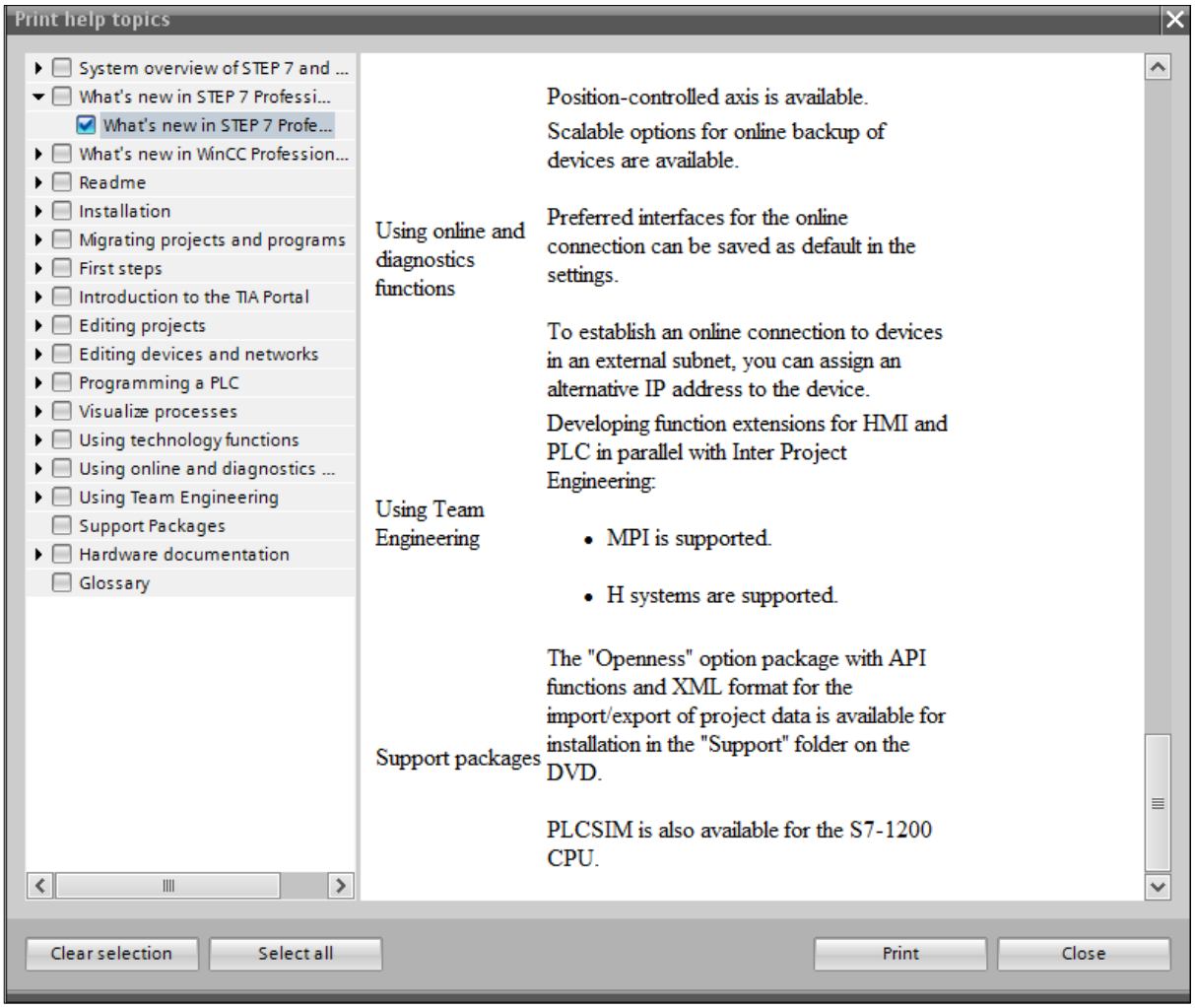
You can then move and resize the help window.

Printing from the online help



To print from the information system, click the "Print" button on the help window.

The "Print" dialog allows you to select the topics to print. Make certain that the panel displays a topic. You can then select any other topic to print.



Click the "Print" button to send the selected topics to your printer.

ICONs of the help topics

Icon	Information	Explanation
	Handling instructions	Describes the procedure for performing a particular task.
	Example	Contains a concrete application example to explain a particular task.
	Factual information	Contains important background information that is necessary for performing a task.
	Reference	This contains reference information.

Devices & Networks:

Online Tools, Configuring and Parameterizing the Hardware

"Devices & Networks" Editor

Almost all devices or components of an automation solution, such as, PLCs or touch panels can be parameterized. The parameterization of devices and network settings necessary for commissioning is done with the "Devices and Networks" Editor.

For example, IP addresses for the communication are assigned to all components of an Ethernet network. But even within the automation device, the address areas for I/O modules have to be defined and the cycle monitoring time of the CPU has to set, for example.

Online Connection via Industrial Ethernet: IP Address & Subnet Mask

Internet Protocol

The Internet Protocol (IP) forms the basis for all TCP/IP networks. It creates the so-called datagrams – specially tailored data packets for the Internet protocol and takes care of their transport in the local sub-network or their "routing" to other subnets.

IP Addresses

IP addresses are not assigned to a specific computer, rather to the network interfaces that a computer has. A computer with several network connections (such as, routers) must therefore have an IP address assigned for each connection.

IP addresses consist of 4 bytes. With the dot notation, each byte of the IP address is expressed as a decimal number between 0 and 255. The four decimal numbers are separated from one another through periods (see slide).

MAC Address

Each Ethernet interface has been assigned a fixed and worldwide unique address by its manufacturer. This address is called hardware or MAC address (Media Access Control). It is stored on the network card and is used as the unique identification in a local network. The cooperation of the manufacturers guarantees that the address is unique worldwide.

Subnet Mask

By means of the subnet mask, the subdivision of the IP address into network and computer address is carried out. Furthermore, with the subnet mask a network can be divided into subnets.

For this, a part of the computer address is used as the sub-network address. As a result, networks can be flexibly adapted to organizational and physical factors.

Online Access: Accessible Devices in the Portal View

Accessible devices in the Portal view:

This method offers quick access (e.g. for service purposes) even if there is no offline project data on the programming device for the target systems.

All accessible, programmable modules (CPUs, FMs and CPs) are listed in the portal view, even if they are located in other subnets.

- "Assign IP address":
- As soon as the button "Show in project tree" is clicked to access a module that is located in a different subnet to the PG, a prompt in the dialog box asks whether an additional IP address should be assigned to the PG. Following confirmation, an additional IP address is assigned which lies in the same subnet as the address of the CPU. All online functions can then be used.

CPU Memory Reset (MRES) using Mode Selector Switch

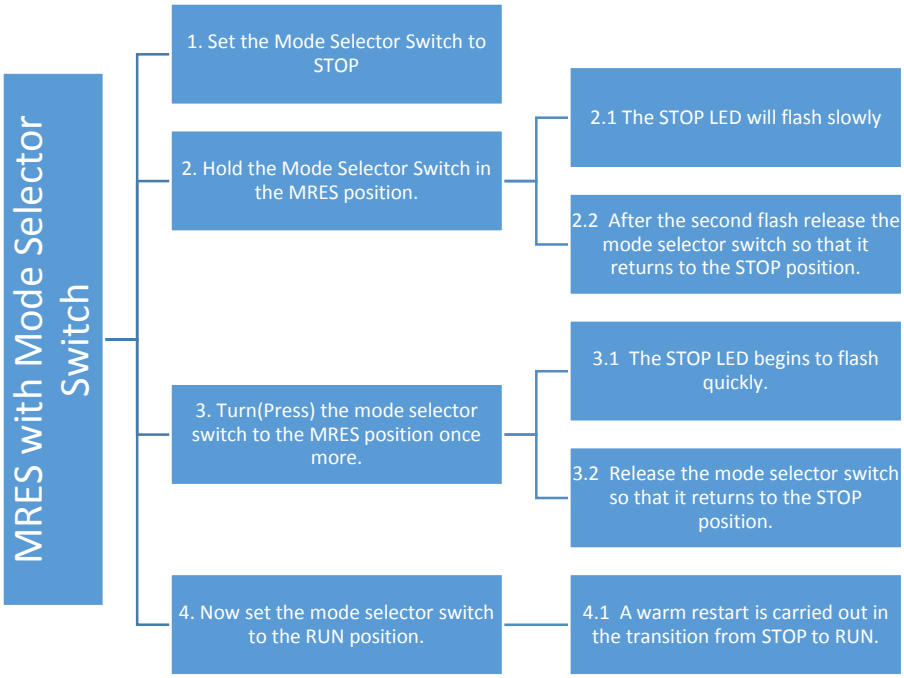
With a Memory Reset, the CPU resets the memory, that is, its work memory, the retentive areas and the diagnostic buffer is deleted.

In Siemens, some series are there where you can perform the Memory Reset with the help of hardware only.

A memory reset clears all work memory, clears retentive and non-retentive memory areas, and copies load memory to work memory. A memory reset does not clear the diagnostics buffer or the permanently saved values of the IP address.

In S7-1200, with the help of STEP7 software only one can operate memory reset. In s7-300, we have option through Mode Selector Switch as well as Step7.As previously mention, with the help of “ONLINE TOOLS” one can click on MRES button.

The procedure for the Mode Selector Switch is described here in the flow chart.



SIMATIC CARD READER

SIMATIC Card

The SIMATIC Memory Card of an S7-1200 is a memory card pre-formatted by Siemens. It can be read and written with the Windows Explorer but under no circumstances can it be formatted with it!

Attention:

The system files "__LOG__" and "crdinfo.bin" stored and hidden on the card are not to be deleted since the CPU then no longer can interpret the card contents. If a completely deleted card is inserted in the CPU, the two files named are recreated on the card by the CPU.

Card type of the SIMATIC Card:

The SIMATIC card is used as a Program card or a Transfer card or for Firmware Updates. Before the data is stored on the SD card, the card type must be selected as shown in the picture.

1. SIMATIC card as Program card:

- The card contains all configuration and parameterization data for the station as well as the complete user program with documentation. During operation, the card must remain inserted in the CPU because it is used as a replacement for the internal CPU load memory which remains unused.

2. SIMATIC card as Transfer card:

- The card contains the same data as a Program card but it doesn't have to remain inserted during operation. After inserting the card and subsequent Power ON, all

data is copied into the internal load memory of the CPU. Then the card has to be removed and a restart has to take place.

3. SIMATIC card to Update firmware:

- The SIMATIC card contains the files required for a firmware update. After execution (instructions are included as a Text file) the SIMATIC card must be removed.

Components of "Devices & Network" Editor

Device editor

You open the device and network editor from the project tree. The hardware and network editor consists of the following components:

- Device view or network view
- Inspector window
- Hardware catalog

Device / Network / Topology view

The hardware and network editor consists of a Device view, a Network view and a Topology view. The Device view is used to configure devices, the Network view is used to network devices and the Topology view is used to determine the physical layout of networks.

Inspector window

The inspector window has the following tabs:

- Properties
- Info
- Diagnostics

The "Properties" tab is used for parameterization. Here, all properties or parameters of modules are displayed and can also be changed. In the left part of the Properties tab is the area navigation in which the parameters are arranged in groups.

Hardware catalog

The "Catalog" pane contains all devices and hardware components arranged in a tree structure. You can drag the devices or modules you want from the catalog into the graphic work area of the device view or network view.

Search and filter function

The "Catalog" pane with the search and filter functions allows you to easily search for particular hardware components. There is a filter function in the hardware catalog. If the filter function is deactivated, all the objects available in the catalog are displayed for you in the

hardware catalog. To only display the objects that you can use in the current context, activate the "Filter" check box. If you have activated the filter, only the following objects are displayed:

- In the network view, only those objects that can be networked are displayed.
- All modules that are part of the context of the current device are displayed in the device view. If you switch between network and device view, the view of the filter objects is adapted to the current context.

Information

The "Information" pane contains detailed information on the object selected from the catalog:

- Schematic diagram
- Name
- Version number
- Order No.
- Short description

Set-point Configuration: Creating Hardware Station

When you configure a system, a set configuration is created. It contains a hardware station with the planned modules and the associated parameters. The PLC system is assembled according to the set configuration. During commissioning, the set configuration is downloaded to the CPU.

Set configuration: Creating a new hardware station

With the set configuration, you define the arrangement of modules on the rack. When you create (add) a new device, a suitable rack is automatically added. The selected device is plugged into the first permitted slot on the rack. Regardless of the chosen way, the added device is visible in the Device view and in the Network view of the Devices & networks editor.

Downloading Actual Configuration into project: Inserting an Unspecified CPU

Upload actual configuration to PG

From a real existing station, the existing configuration (without module parameters!) can be read out.

This becomes necessary, for example, when there is no matching offline project on the programming device. After reading out the actual configuration, you can check, change, save and reload the parameterizations of the modules into the CPU. For this, the first step is to add an "Unspecified CPU" in the offline project.

Detect the online station

As soon as the "Unspecified CPU" is added, all accessible devices can be detected via "detect" in the dialog shown.

Click the word "detect" with the mouse to open a new window in which all accessible devices are displayed. Search for the device that you want to insert in your configuration and click the "Load" button. The configuration is detected and inserted in your project.

Actual configuration

The actual configuration for the selected device is read out and placed in the offline project.

Note:

In reading out, only the hardware configuration is uploaded, no hardware parameterization and also no S7 program blocks.

Compiling the Hardware configuration and downloading it into the CPU

Compile and download hardware configuration

The following components of a hardware station can be compiled or downloaded:

- All
- The complete hardware configuration and hardware parameterization as well as the complete user program is compiled.
- Hardware configuration
- Only the complete hardware configuration and hardware parameterization is compiled.
- Software / Software (rebuild all)
- With "Software (rebuild all)", all blocks of the user program are compiled; with "Software", only the modified blocks

CPU Properties: Ethernet address

CPU PROFINET interface

Regardless of whether the Devices & networks Editor is in the Device or the Network view, the settings for the CPU-PROFINET interface can be made in the "Properties" tab in the Inspector window when the CPU is highlighted.

If an online connection between the programming device and the CPU is to be established, the same subnet mask must be assigned to the two devices. The IP addresses have to be in the same subnet.

PLC TAGS

Meaning of Variables and Data Types

Variables:

A variable is a placeholder for a data value that can be changed in the program. The format of the data value is defined. The use of variables makes your program more flexible. For example, you can assign different values to variables that you have declared in the block interface for each block call. As a result, you can reuse a block you have already programmed for various purposes.

A variable consists of the following elements:

- Name
- Data type
- Absolute address
    - PLC tags and DB tags in blocks with standard access have an absolute address.
    - DB variables in blocks with optimized access have no absolute address.
- Value (optional)

Declaring Variables:

You can define variables with different scopes for your program:

- PLC tags that apply in all areas of the CPU
- DB variables in global data block that can be used by all blocks throughout the CPU.
- DB tags in instance data blocks that are predominantly used within the block in which they are declared.

The following table shows the difference between the variable types:

	PLC tags	Variables in instance DBs	Variables in global DBs
Range of application	Are valid throughout the entire CPU. Can be used by all blocks on the CPU. The name is unique within the CPU.	Are predominantly used in the block in which they are defined. The name is unique within the instance DB.	Can be used by all blocks on the CPU. The name is unique within the global DB.
Permissible characters	Letters, numbers, special characters Quotation marks are not permitted. Reserved keywords are not permitted.	Letters, numbers, special characters Reserved keywords are not permitted.	Letters, numbers, special characters Reserved keywords are not permitted.

Use	I/O signals (I, IB, IW, ID, Q, QB, QW, QD) Bit memory (M, MB, MW, MD)	Block parameters (input, output and in-out parameters), Static data of a block	Static data
Location of definition	PLC tag table	Block interface	Declaration table of the global DB

Data types:

PLC data types are data structures that you define and that can be used multiple times within the program. The structure of a PLC is made up of several components, each of which can contain different data types. You define the type of components during the declaration of the PLC data type.

You can create up to 65534 PLC data types for a CPU of the S7-1200 or S7-1500 series. Each of these PLC data types can include up to 252 components.

PLC data types can be used for the following applications:

- PLC data types can be used as data types for variables in the variable declaration of logic blocks or in data blocks.
- PLC data types can be used as templates for the creation of global data blocks with identical data structures.
- PLC data types can be used in S7-1200 and S7-1500 as a template for the creation of structured PLC tags.

PLC Tags

Introduction

PLC tag tables contain the definitions of the PLC tags and symbolic constants that are valid throughout the CPU. A PLC tag table is created automatically for each CPU used in the project. You can create additional tag tables and use these to sort and group tags and constants.

In the project tree there is a "PLC tags" folder for each CPU of the project. The following tables are included:

- "All tags" table
- Standard tag table
- Optional: Other user-defined tag tables

All tags

The "All tags" table gives an overview of all PLC tags, user constants and system constants of the CPU. This table cannot be deleted or moved.

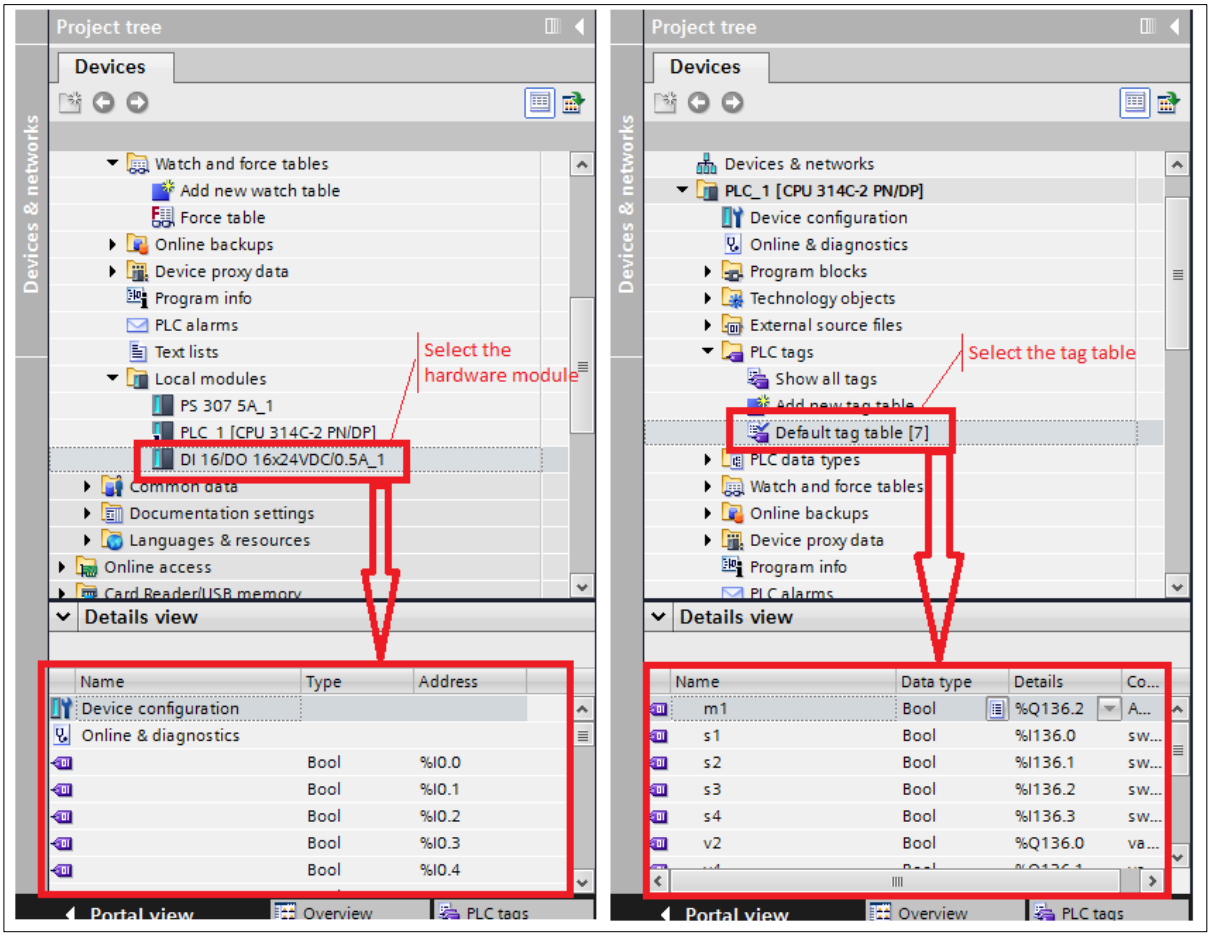
Standard tag table

There is one standard tag table for each CPU of the project. It cannot be deleted, renamed or moved. The default tag table contains PLC tags, user constants and system constants. You can declare all PLC tags in the default tag table, or create additional user-defined tag tables as you want.

User-defined tag tables

You can create multiple user-defined tag tables for each CPU to group tags according to your requirements. You can rename, gather into groups, or delete user-defined tag tables. User-defined tag tables can contain PLC tags and user constants.

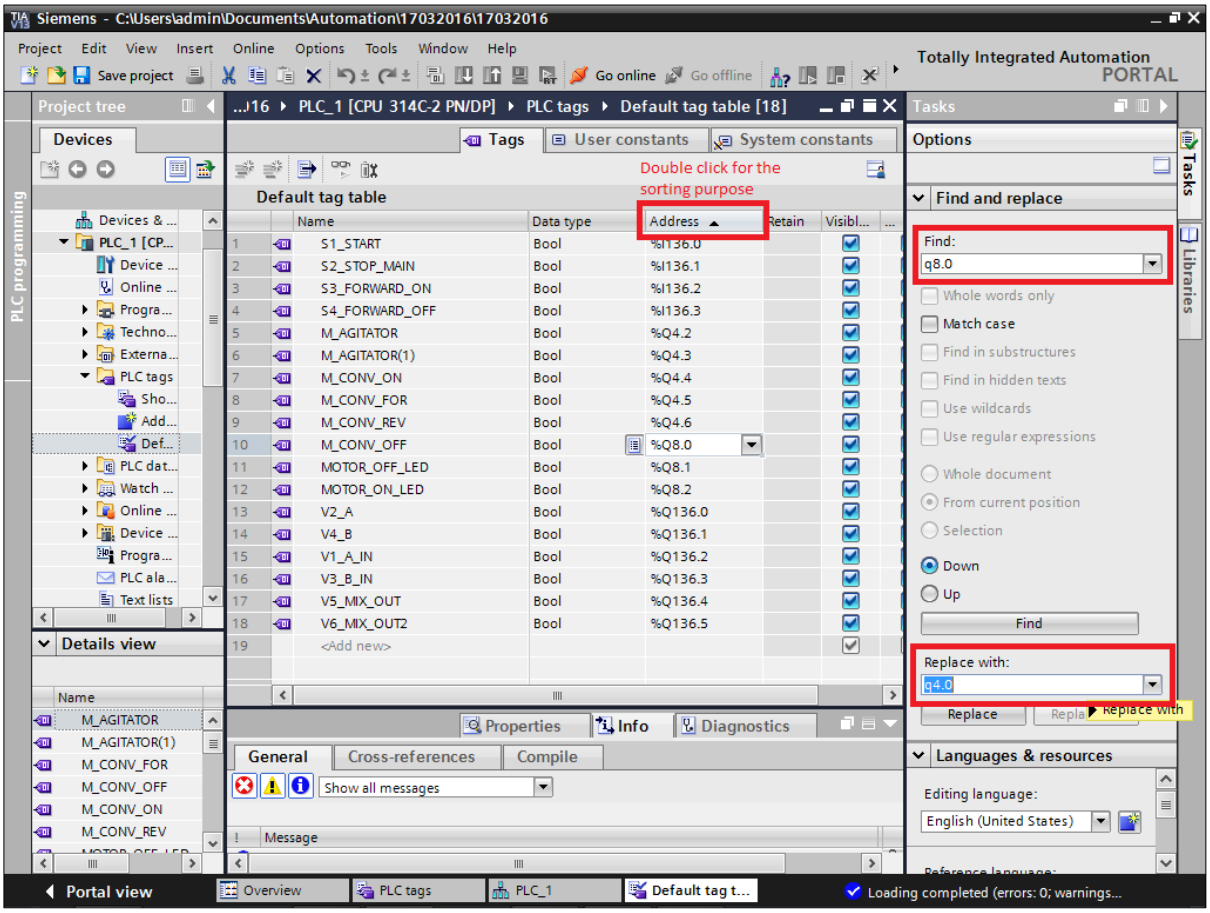
Details View of PLC Tags



The details view shows:

- Tags of the selected tag table
- The channels of the selected local modules and their tags (if defined)

Finding/Replacing/Sorting PLC Tags



Find / Replace with

In the PLC tag table, tags can be found and replaced via the "Tasks" task card. Dummies can also be used (? for one character, * for several characters).

Example of "Find and Replace with":

Assign bit address q4.0 with bit address q8.0:

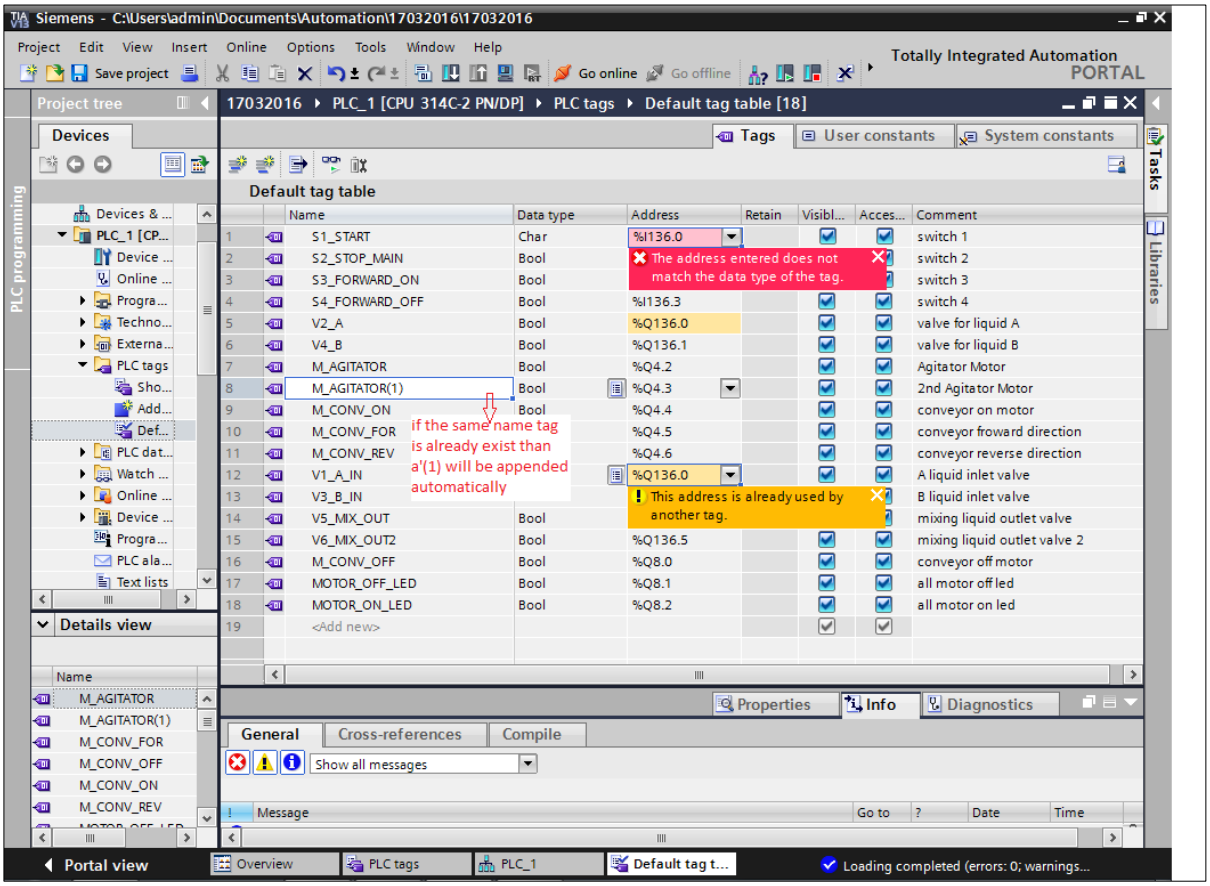
Find: Q 8.0

Replace with: Q 4.0 (Already shown in the image before)

Sorting

By clicking on one of the column names "Name", "Data type" or "Address", the tags are sorted alphabetically or according to address (ascending or descending) depending on the column.

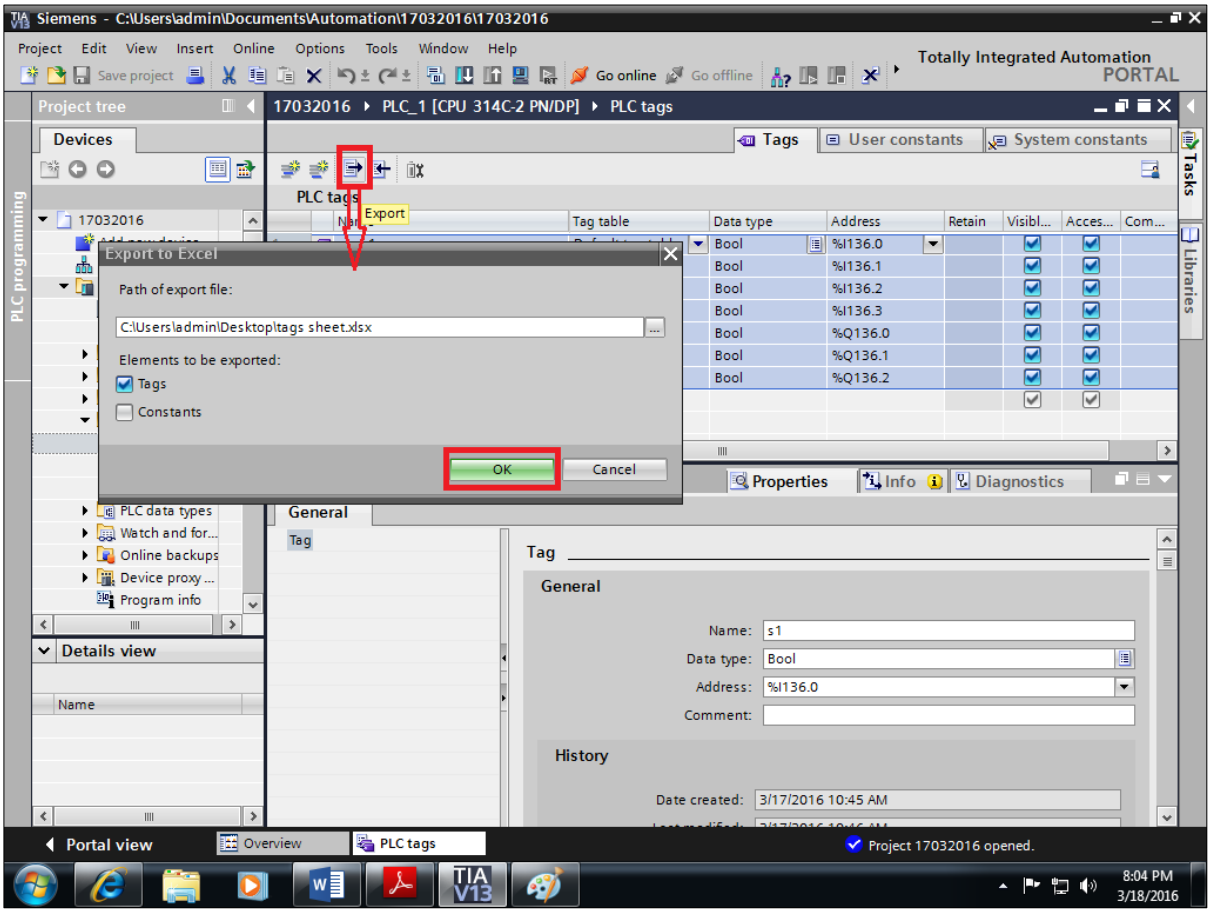
Error Indication in PLC Tag Table



Syntax check

With every entry, there is a syntax check in which existing errors are displayed in RED or YELLOW. A still faulty PLC tag table can be saved but as long as it is still faulty, the program cannot be compiled and downloaded into the CPU.

Copy and Paste PLC Tags to Excel



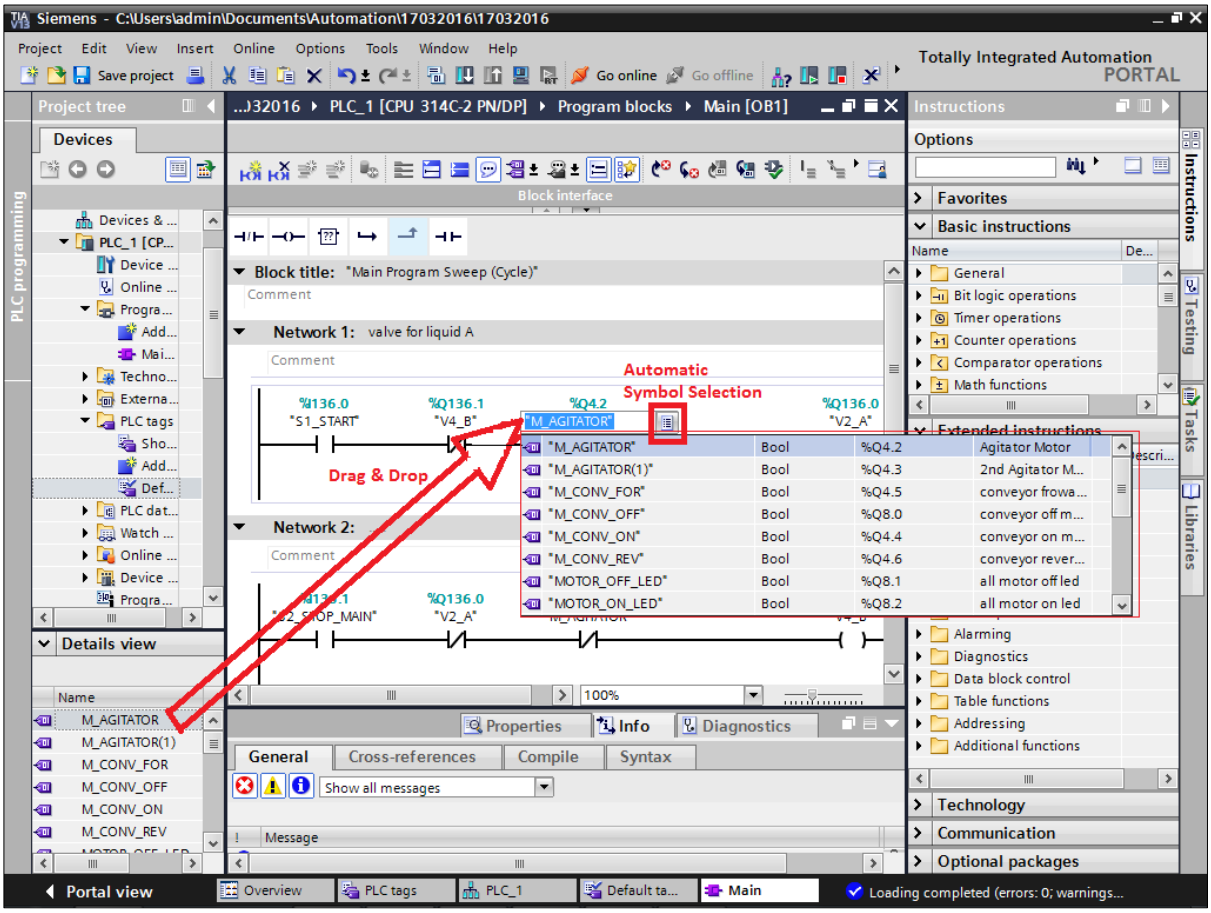
Copy & Paste from and to Excel

The Windows Copy & Paste function can be used to easily copy individual or several tags from a PLC tag table to Excel to further process it/them there and then to copy it/them back from Excel to the PLC tag table.

	A	B	C	D	E	F	G
1	Name	Path	Data Type	Logical Address	Comment	Hmi Visible	Hmi Accessible
2	s1	Default tag table	Bool	%I136.0	switch 1	True	True
3	s2	Default tag table	Bool	%I136.1	switch 2	True	True
4	s3	Default tag table	Bool	%I136.2	switch 3	True	True
5	s4	Default tag table	Bool	%I136.3	switch 4	True	True
6	v2	Default tag table	Bool	%Q136.0	valve for liquid A	True	True
7	v4	Default tag table	Bool	%Q136.1	valve for liquid B	True	True
8	m1	Default tag table	Bool	%Q136.2	Agitator Motor	True	True
9							
10							
11							
12							
13							
14							
15							
16							

PLC Tags

Using a PLC Tag as an Operand



Use tag as operand

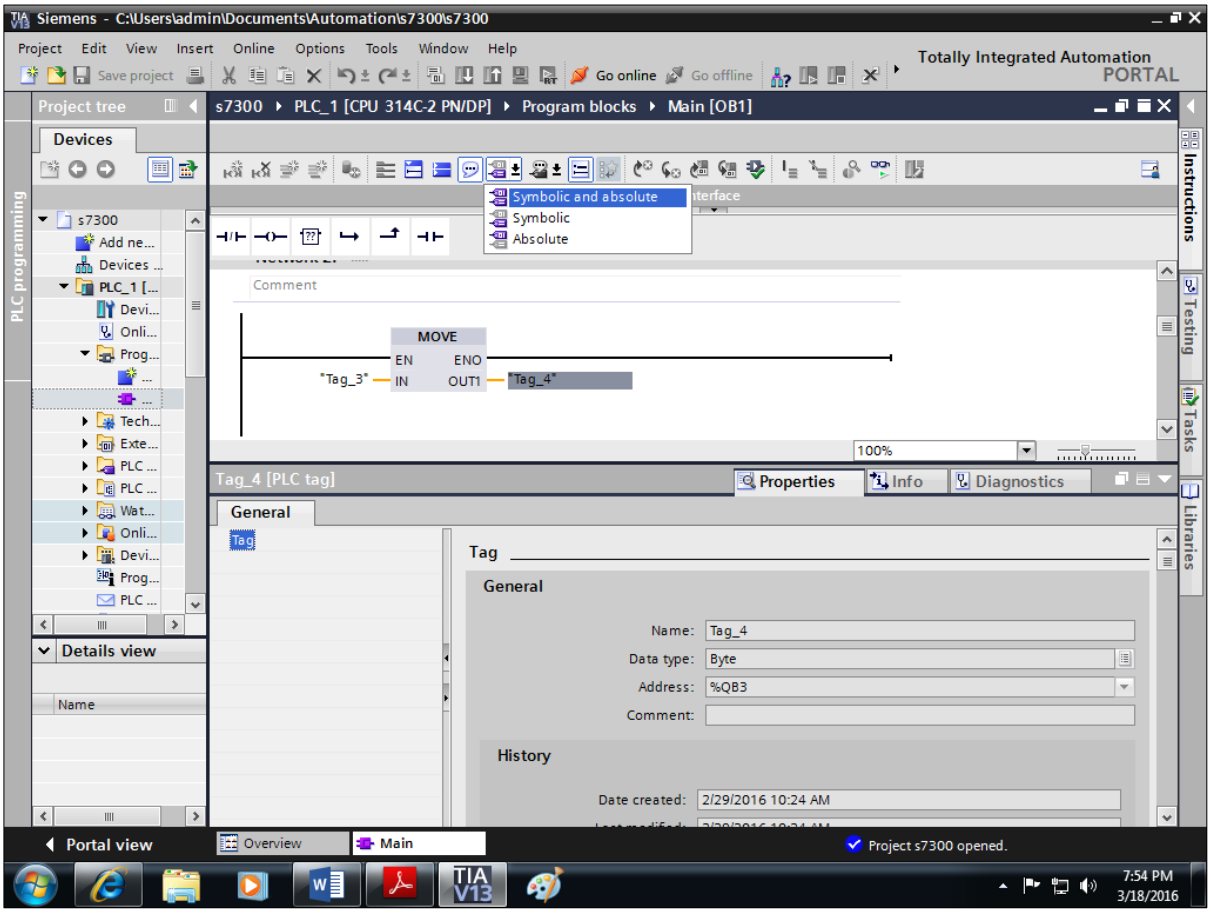
During programming, the name of the tag can be entered either from the automatic symbol selection or selected from the Details view.

Symbol selection

When operands are selected, after the first letter of an operand has been entered, a selection of all the operands of the corresponding data type that start with the entered letter is displayed. All the operands that are valid for this block are displayed. These comprise all global tags (also those that are declared in data blocks), local tags (temporary and static) and the parameters of the affected block.

In the first column of the symbol selection, either the symbol or the absolute operand can be displayed.

Absolute & Symbolic Addressing



Symbolic and absolute addressing

All global tags (such as, inputs, outputs, memory bits) have both an absolute and a symbolic address. You can define which is to be displayed or with which is to be programmed (see slide).

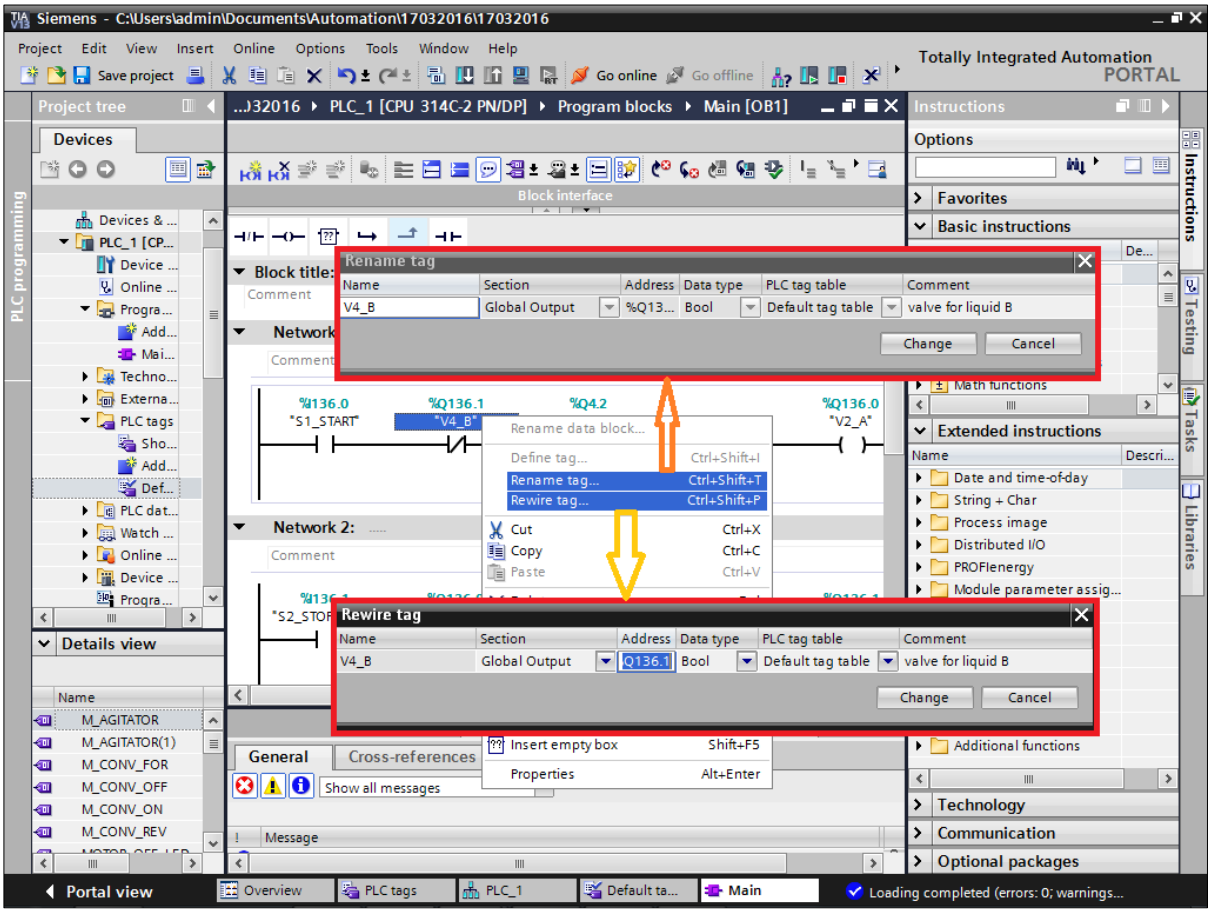
When you use a symbolic address (for example, "M_Jog_RIGHT") to which an absolute address has not yet been assigned, you can save the block but you cannot compile and download it into the controller.

When you use an absolute address (for example, M16.2), it is automatically assigned a symbolic standard address (for example, "Tag_1") which you can change.

Properties

If a block or the PLC tag table is open in the working area and a tag is selected (highlighted) there, then all details are displayed in the "Properties" tab in the Inspector window where they can also be edited.

Renaming/rewiring PLC Tags



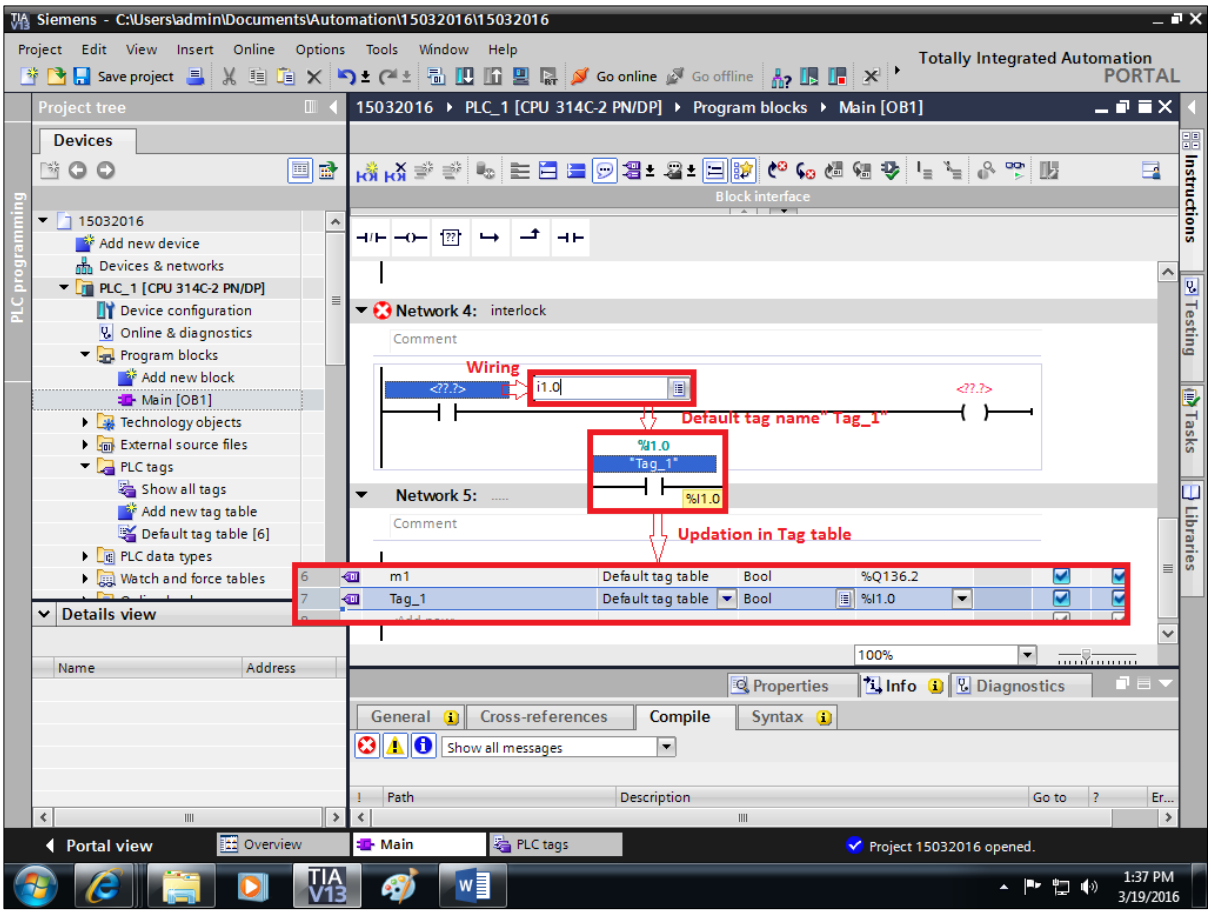
Rename and rewire tags

Tags can be renamed or rewired directly in the PLC tag table or as shown in the picture using the Blocks Editor. The changes are immediately adopted in the PLC tag table and affect the entire program.

- Rename:
- Change the tag name, while the absolute address remains unchanged.
- Rewire:
- Change the associated absolute address, while the name remains unchanged.

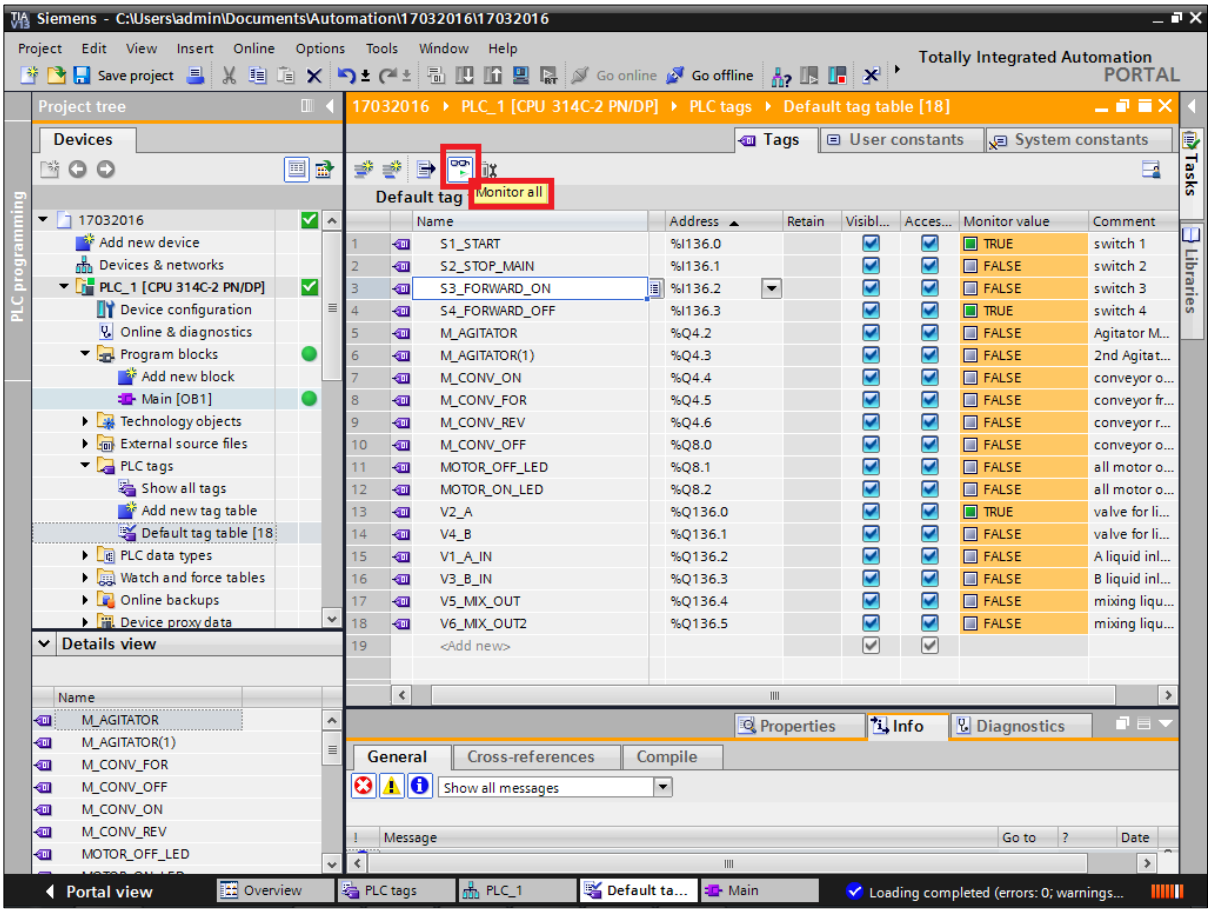
Defining Tags while Programming

While during programming also, programmer can define the tag which will be automatically updated in “Default Tag Table”. While programming, wire the contact with the physical address (e.g. I1.0). Name of the tag will be “Tag_1” by default.



Another option for declaration of tags are updating the tag table. Updation if tag table is also possible.

Monitoring PLC Tags



Monitor

You can monitor the current data values of the tags on the CPU directly in the PLC tag table.

Procedure

To monitor the data values, follow these steps:

- Start monitoring by clicking the "Monitor all" button.
- The additional "block title" column is displayed in the table. It shows the current data values.
- End monitoring by clicking the "Monitor all" button again.

Monitor value

Current data value in the CPU. This column is visible if an online connection is available and the "Monitor" button has been clicked.

Retentiveness of PLC Tags7

Retentive PLC tags

To prevent data loss in the event of a power failure, you can mark specific data as retentive. These are stored in a retentive memory area. A retentive memory area is an area whose content remains available on restart (warm restart), i.e. after switching off the supply voltage and switching it on again, on transition from STOP to RUN.

In the case of a cold restart, the values of the data defined as retentive will be erased.

Available retentive memory in [Bytes]:

Here you have to specify how many bytes of the retentive memory area of the CPU are still available. Please pay attention to the fact that tags in data blocks which are declared as retentive also reduce the number of bytes in the "available retentive memory".

Settings

You can define the following data as retentive:

- Memory bit: You can define the width of the retentive memory area for bit memories precisely in the PLC tag table.
- Tags of a function block (FB): In the interface of an FB, you can define the individual tags as retentive when the symbolic addressing of tags is active for this block. If symbolic addressing is not active for an FB, you can only define the tags as retentive in the associated instance data block.
- Tags of a global data block: In a global data block, depending on the setting for symbolic addressing, you can either define individual tags or all the tags of a block as retentive: The attribute "Symbolic access only" of the DB is activated: Retention can be set for each individual tag. The attribute "Symbolic access only" of the DB is deactivated: The retention setting applies to all tags of the DB; either all tags are retentive or no tags are retentive.

HMI Access to PLC Tags

HMI tag access

The S7-1200 has protective mechanisms with which unwanted accesses to PLC tags from HMI devices can be prevented:

"Visible in HMI":

During HMI configuration, only PLC tags with the attribute "Visible in HMI" can be selected. This filter function can, however, be disabled in the selection dialog shown by activating "Show all".

"Accessible in HMI":

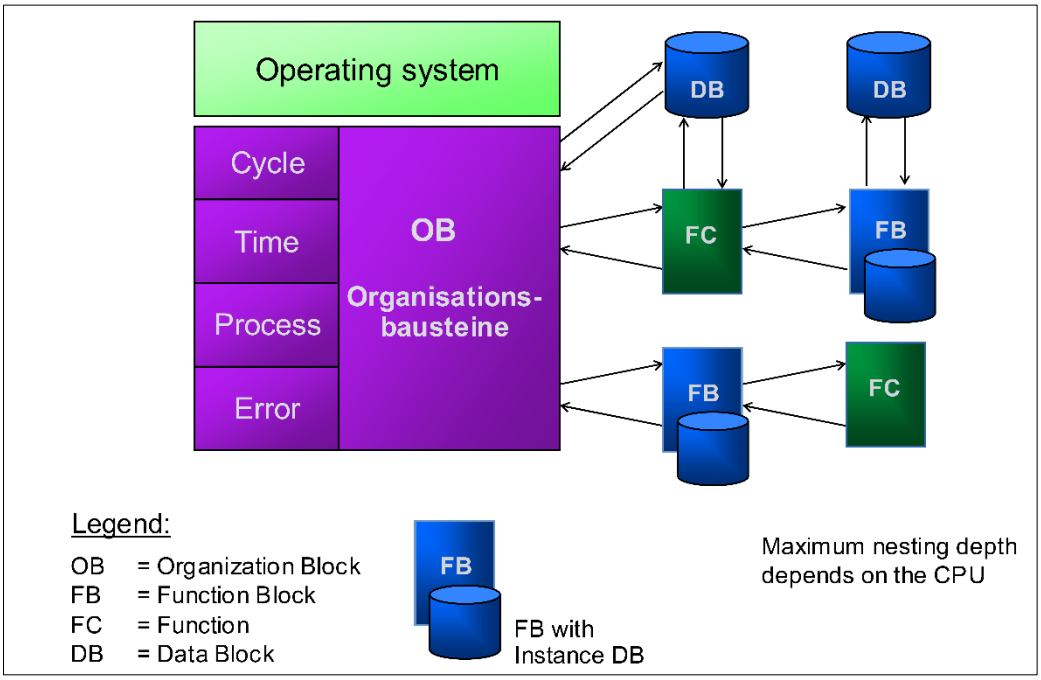
The HMI device can only access the PLC tags online which have the attribute "Accessible in HMI". This protective function ensures that the HMI device does not overwrite certain tags. Tags which are not "Accessible in HMI", are correspondingly also not "Visible in HMI".

Program Blocks & Program Editor

Types of Program Blocks

Blocks

The programmable logic controller provides various types of blocks in which the user program and the related data can be stored. Depending on the requirements of the process, the program can be structured in different blocks. You can use the entire operation set in all blocks (FB, FC and OB).



Organization blocks (OBs)

Organization blocks (OBs) form the interface between the operating system and the user program. The entire program can be stored in OB1 that is cyclically called by the operating system (linear program) or the program can be divided and stored in several blocks (structured program).

Function (FCs)

A function (FC) contains a partial functionality of the program. It is possible to program functions so that they can be assigned parameters. As a result, functions are also suited for programming recurring, complex partial functionalities such as calculations.

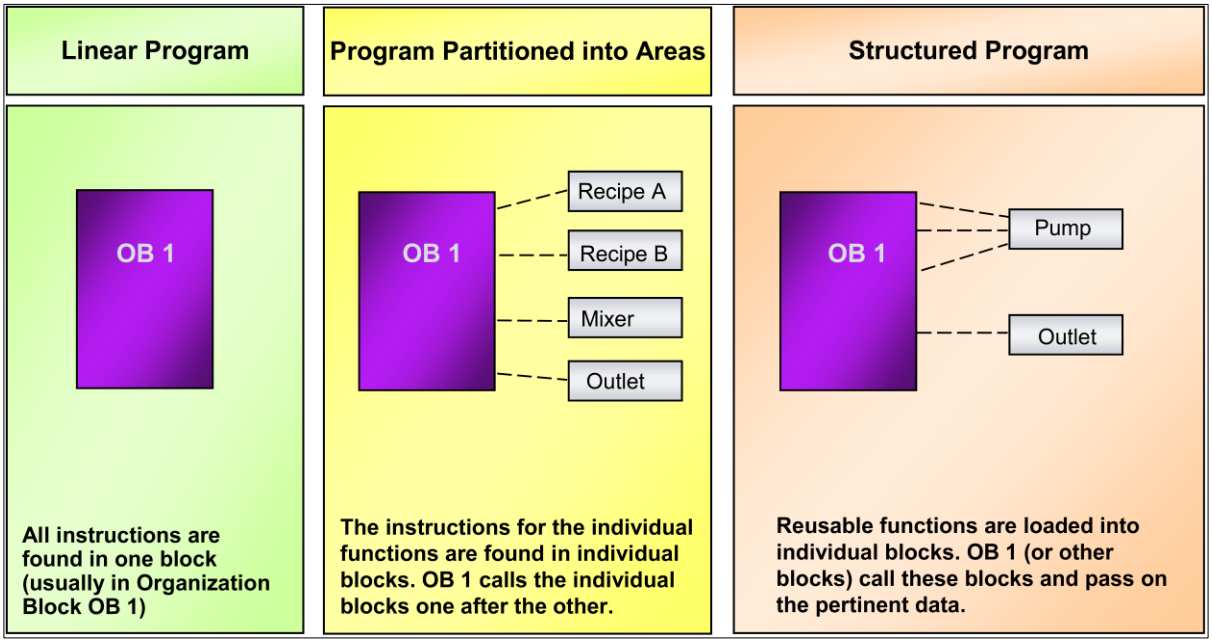
Function Block (FBs)

Basically, function blocks offer the same possibilities as functions. In addition, function blocks have their own memory area in the form of instance data blocks. As a result, function blocks are suited for programming frequently recurring, complex functionalities such as closed-loop control tasks.

Structured Programming

Linear Program

The entire program is found in one continuous program block. This model resembles a hard-wired relay control that was replaced by a programmable logic controller. The CPU processes the individual instructions one after the other.



Partitioned Program

The program is divided into blocks, whereby every block only contains the program for solving a partial task. Further partitioning through networks is possible within a block. You can generate network templates for networks of the same type. The OB 1 organization block contains instructions that call the other blocks in a defined sequence.

Structured Program

A structured program is divided into blocks. The code in OB1 is kept to a minimum with calls to other blocks containing code. The blocks are parameter assignable. These blocks can be written to pass parameters so they can be used universally. When a parameter assignable block is called, the programming editor lists the local variable names of the blocks. Parameter values are assigned in the calling block and passed to the function or function block.

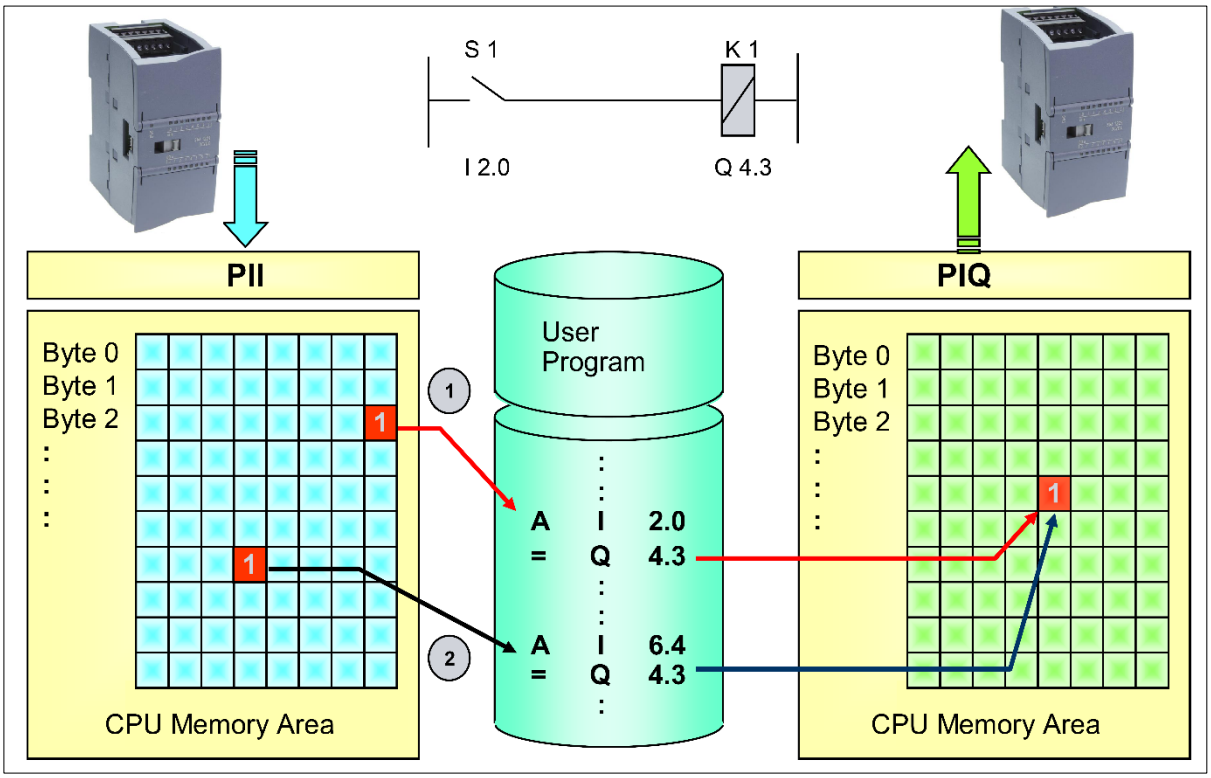
Example:

- A "pump block" contains instructions for the control of a pump.
- The program blocks, which are responsible for the control of special pumps, call the "pump block" and give it information about which pump is to be controlled with which parameters.

Process Images

Process Images

For the storage of all digital input and output states, the CPU has reserved memory areas: the process-image input table (PII) and the process-image output table (PIQ). During program execution, the CPU accesses these memory areas exclusively. It does not access the digital input and output modules directly.



PII

The Process-Image Input table is the memory area in which the states of all digital inputs are stored. The image is read in from the digital input modules at the beginning of the cycle. If inputs are queried in the user program (for example, A I 2.0), then the state of this input that is stored in the PII is queried from the PII. This state cannot change within a cycle since the PII is only updated or read in at the beginning of a cycle. This guarantees that when there are multiple queries of the input in one cycle, the same result is always delivered.

PIQ

The Process-Image Output table is the memory area in which the states of all digital outputs are stored. The image is output to the digital output modules at the end of the cycle. Outputs can be assigned as well as queried in the program. If an output is assigned a state in several locations in the program, then only the state that was assigned last is transferred to the particular output module (see slide). As a rule, these types of double assignments are programming errors.

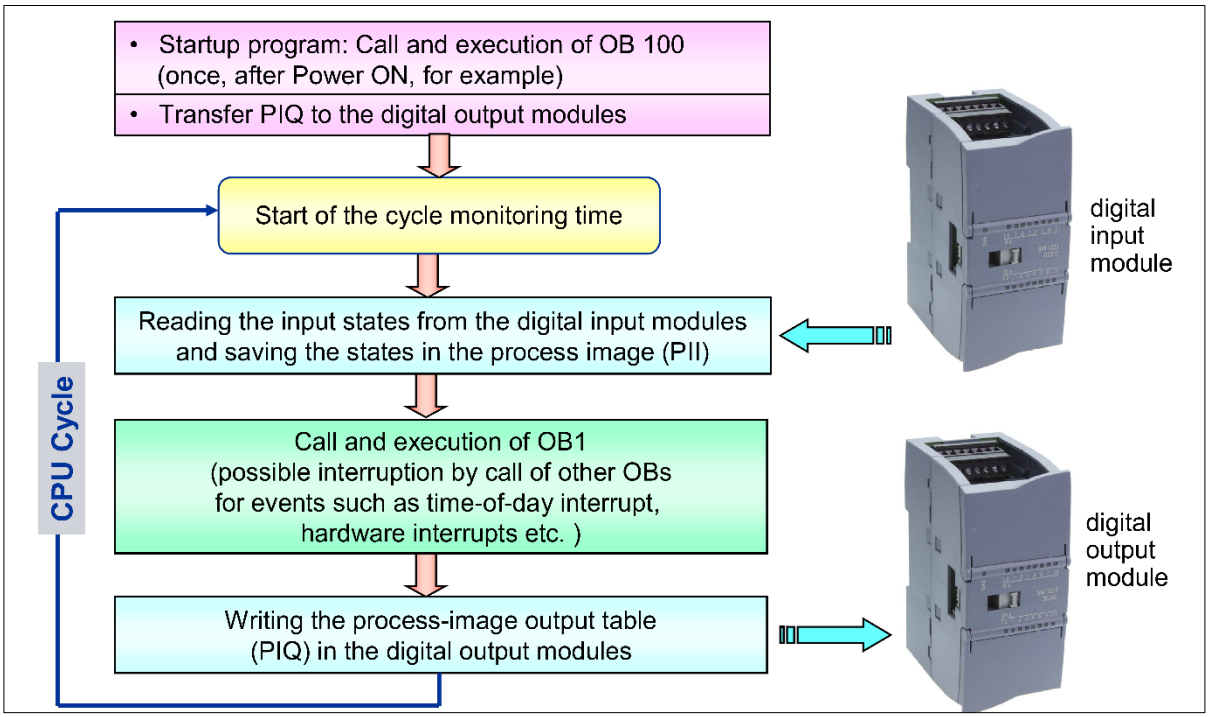
Cyclic Program Execution

Restart

When you switch on or switch from STOP --> RUN, the CPU carries out a complete restart (with OB100). During restart, the operating system deletes the non-retentive memory bits, timers and counters, deletes the Interrupt stack and the Block stack, resets all stored process interrupts and diagnostic interrupts and starts the cycle monitoring time.

Cyclic program execution

Cyclic program execution occurs in an endless loop. After the execution of a program cycle is completed, the execution of the next cycle occurs automatically. In every program cycle, the CPU carries out the following steps.



The CPU scans the states of the input signals and updates the process image inputs.

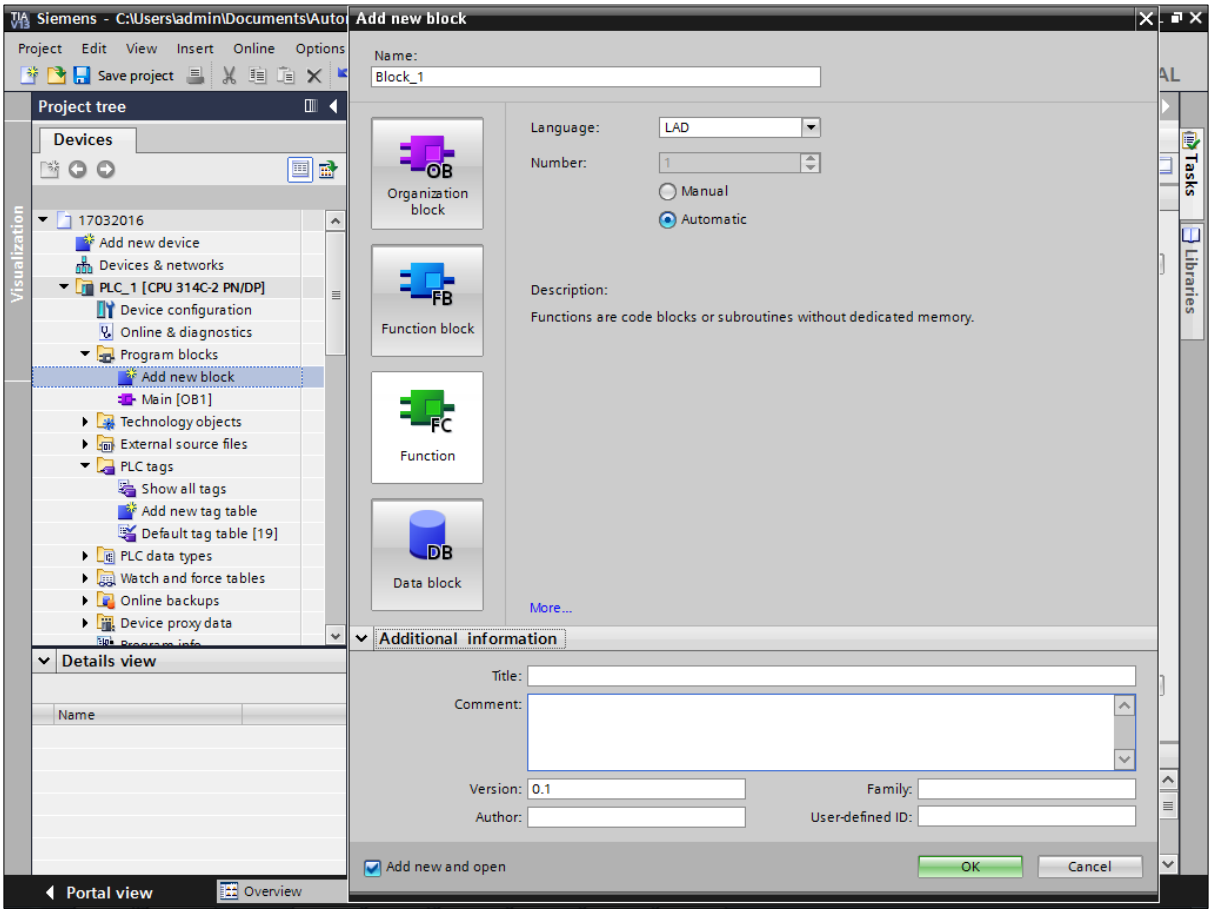
The CPU sequentially processes the instructions of the user program and so works directly with the process images, not with the inputs and outputs of the digital input / output modules.

The CPU transfers the output states from the process image outputs to the digital output modules.

Cycle and cycle monitoring time

The time that the CPU requires for the execution of a complete program cycle, is the cycle time which is monitored for time by the CPU operating system. If the cycle time exceeds the cycle monitoring time defined in the CPU properties by more than double, the CPU goes into the STOP state.

Adding a New Block

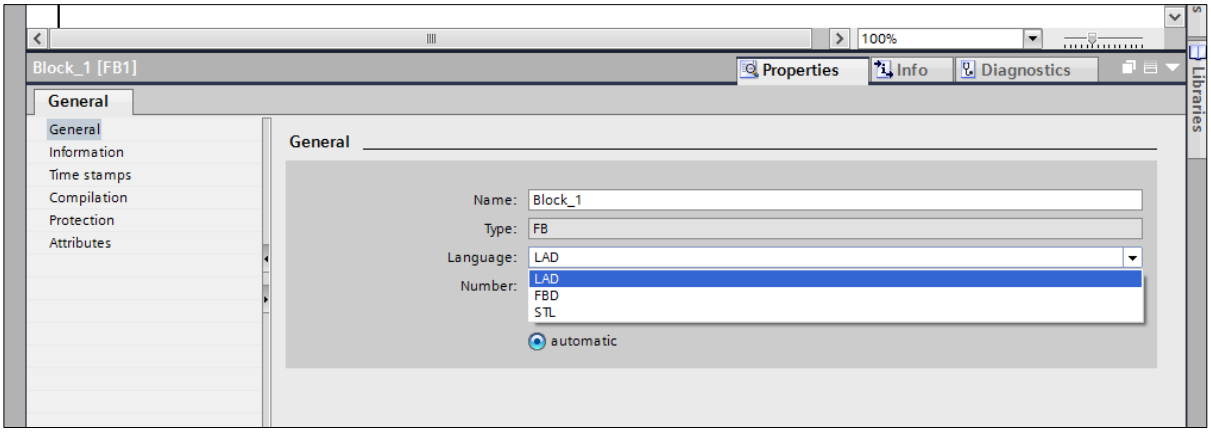


Inserting a block

A new block is created as shown in the picture. When you create a block, the type of block (OB, FB, FC or DB), the programming language, the symbolic name and number, among other things, must be defined. The block numbers can also be assigned automatically or manually. Under "Further information", the block can be documented in more detail, among other things, with a Version number and an Author.

Block Properties: Programming Language

Each block has certain properties that you can display and edit. These properties are used to:



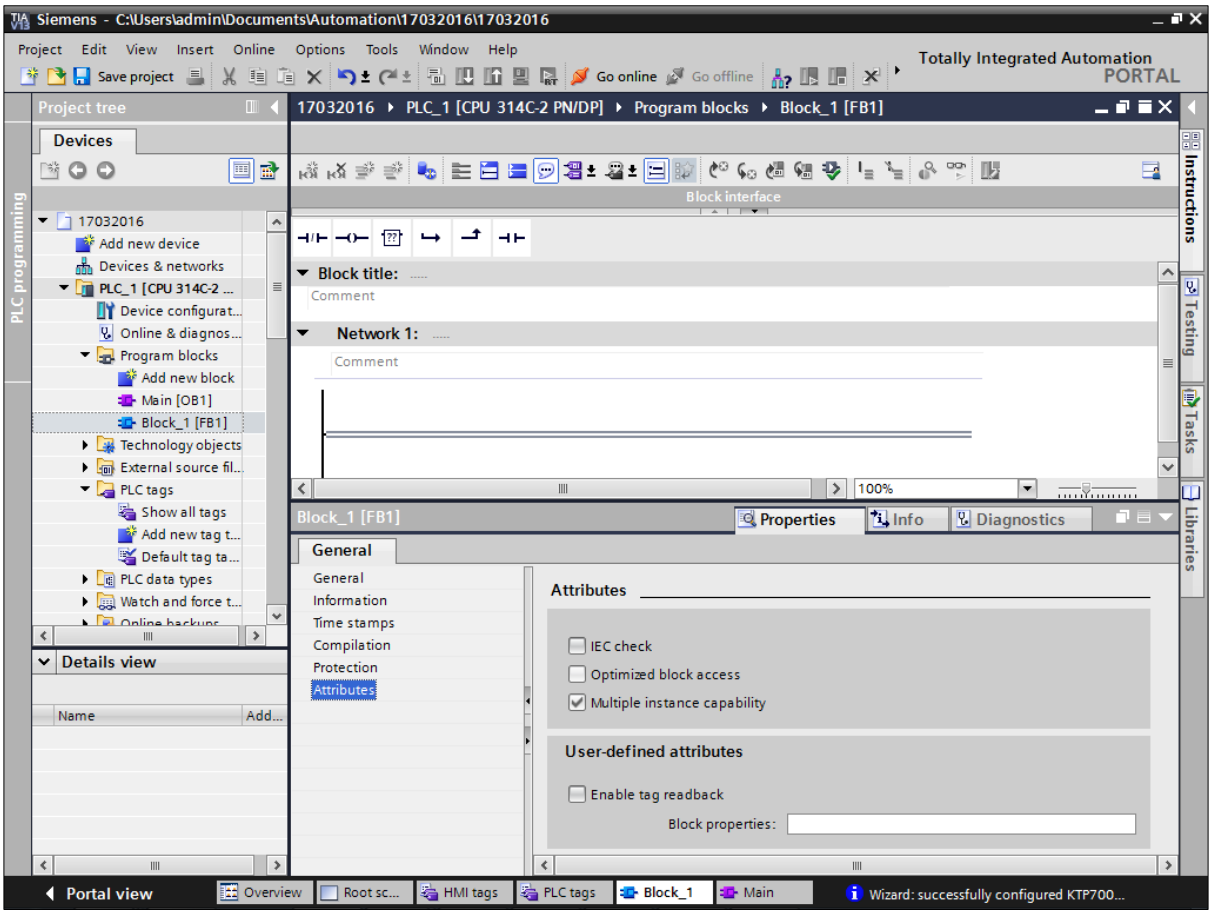
- Identify the block
- Display the memory requirements and the compilation status of the block
- Display the time stamp
- Display the reference information
- Specify the access protection

Block parameters

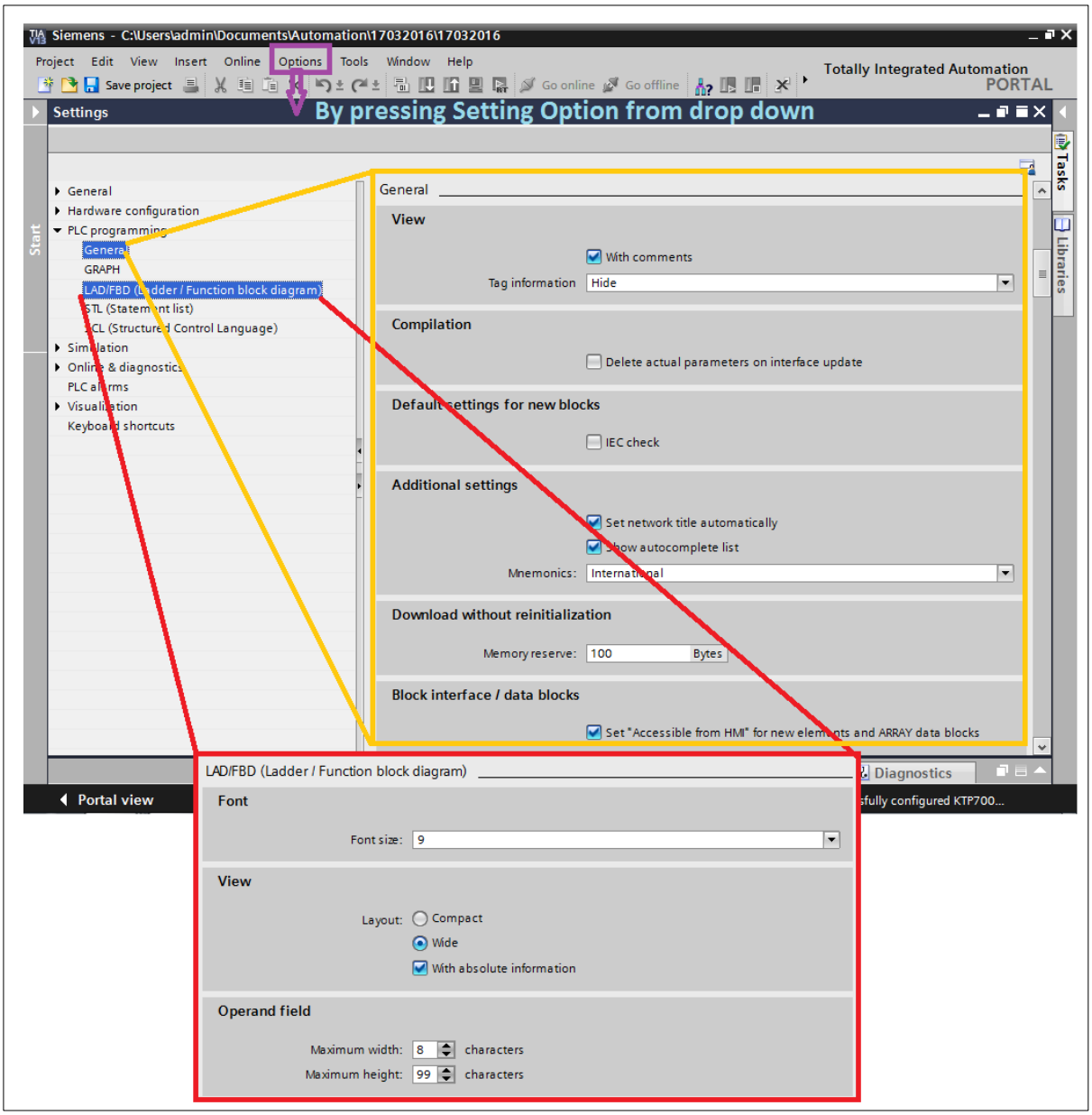
Organization blocks have block parameters that you can use to parameterize specific responses, e.g. assignment of an event to an organization block.

Other Block Attributes, Editor Settings, Networks

Attributes



Block editor settings



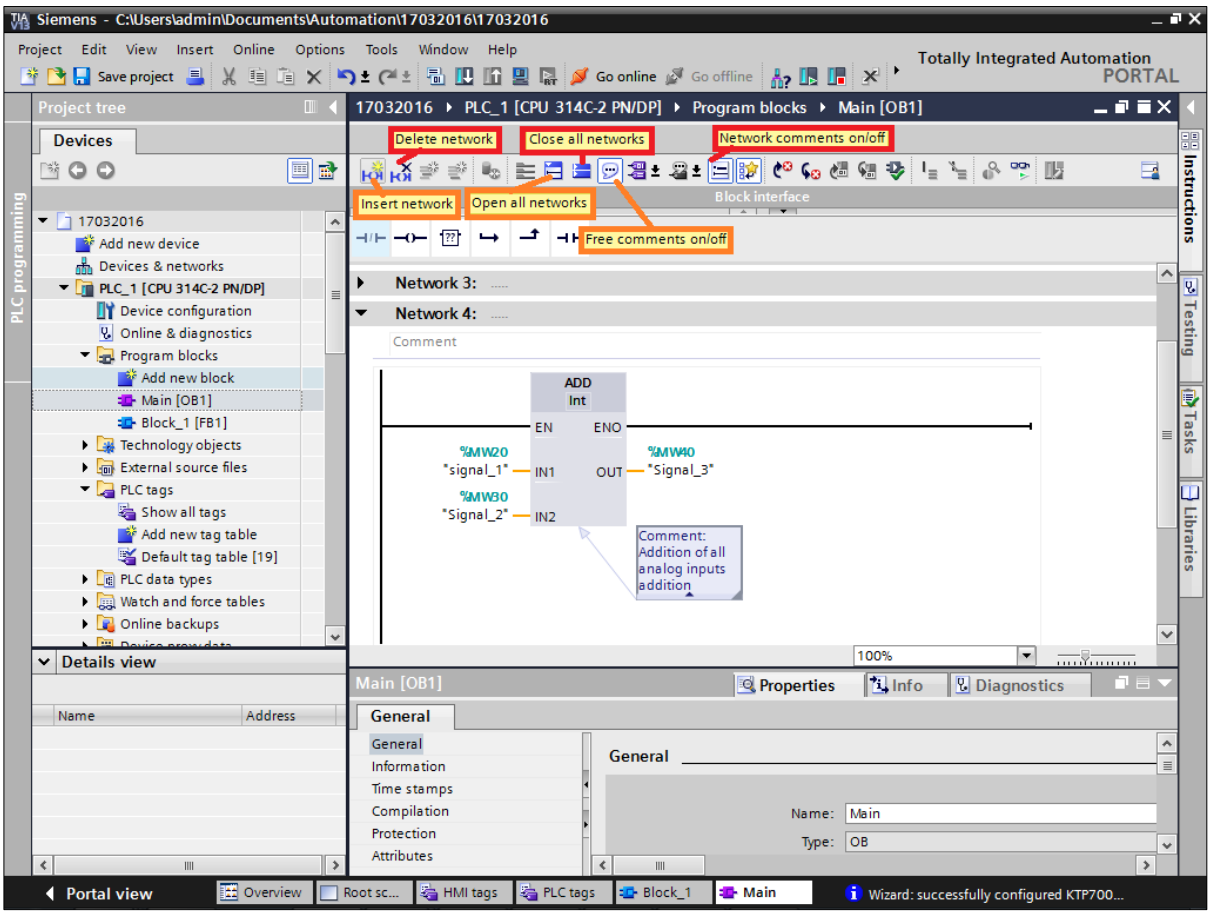
With the settings, you merely define how a block is to be represented when it is opened. In the editor, you can make changes (such as showing and hiding comments) at any time.

- Compilation
- When "Delete actual parameters on interface update" is activated, the calls of parameterized blocks are automatically adjusted if, within the block, parameters are deleted after the fact.
- IEC check
- Only variables of an absolutely correct data type can be used. If an operation requires a variable of the data type INT, no variable of the data type WORD can be used even if the dimension (16 bits) is the same.
- Optimized block access

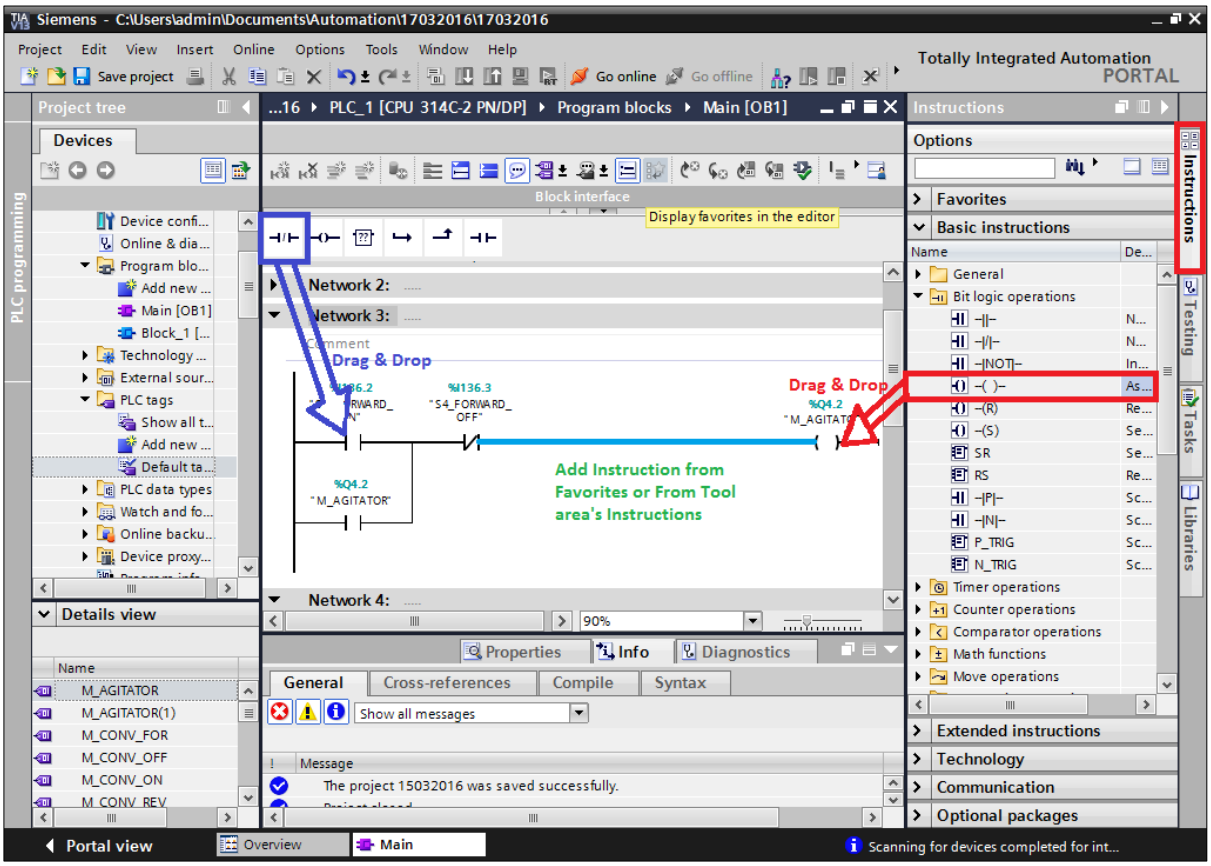
- Data block variables and local variables within blocks can only be addressed symbolically and not absolutely. Benefit: optimum memory allocation and shorter access times
- Mnemonics
- Setting the syntax for the programming language: German (e.g. E for Eingang (Input)) or International (e.g. I for Input)
- Layout
- When "with absolute information" is activated, the absolute addresses of global operands are also displayed.
- Operand field
- Setting the maximum width and height of function block diagram and ladder diagram symbols.

Networks

Just as the entire user program is subdivided into individual blocks, the individual blocks in turn are made up of individual networks. The subdivision of a block into networks is defined by the user. Every network can be given a network label and a comment. Within the networks, the individual instructions can be given instruction comments.



Block Programming



The instructions within a block can be programmed as follows:

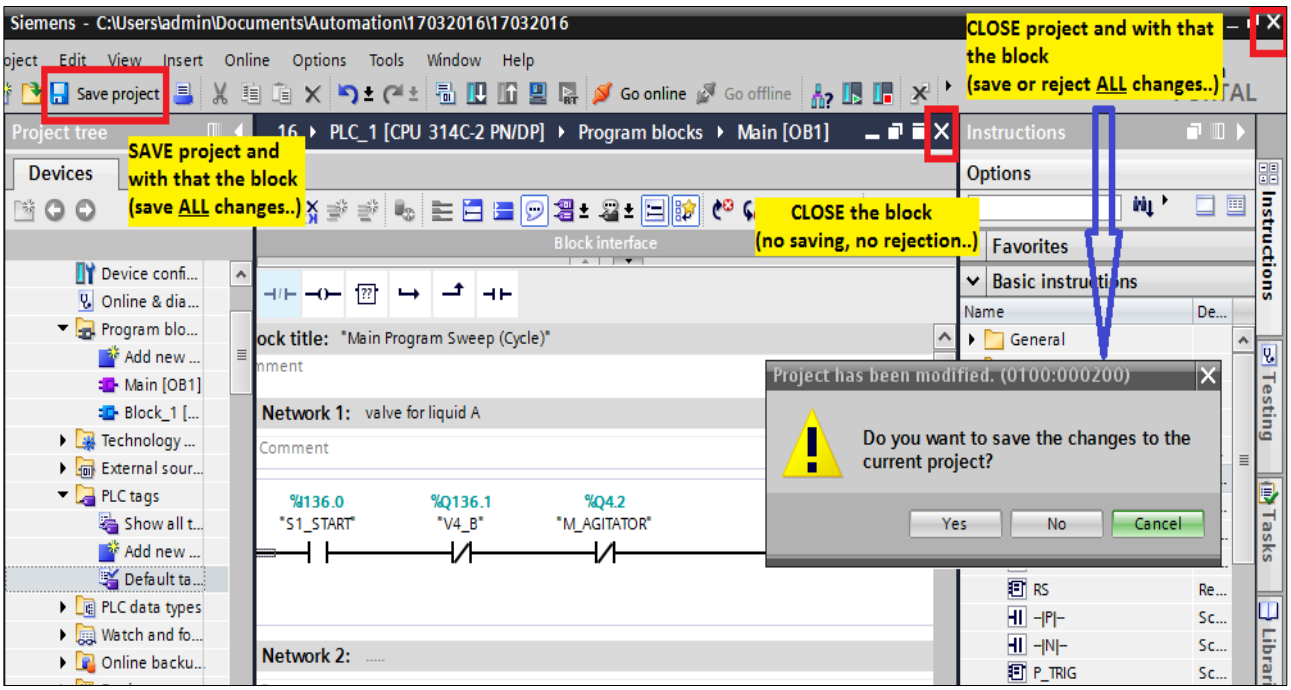
- using drag & drop from the Favorites or the Instructions catalog to anywhere in the program
- by first selecting the location in the program and then double-clicking on the desired instruction in the Favorites or the Instructions catalog

Operands can be entered with an absolute or a symbolic address. If the tag table is highlighted (not opened!) in the Project tree, tags (variables) can also be pulled from the Details view using drag & drop to the appropriate location in the program.

Favorites

Frequently used FBD elements are available in the symbol bar which can be expanded individually using drag & drop from the Instructions catalog.

Closing/Saving/Rejecting a Block



Closing a block

By clicking on the symbol in the title bar, the block is merely closed. Changes are neither rejected nor are they saved on the hard drive!

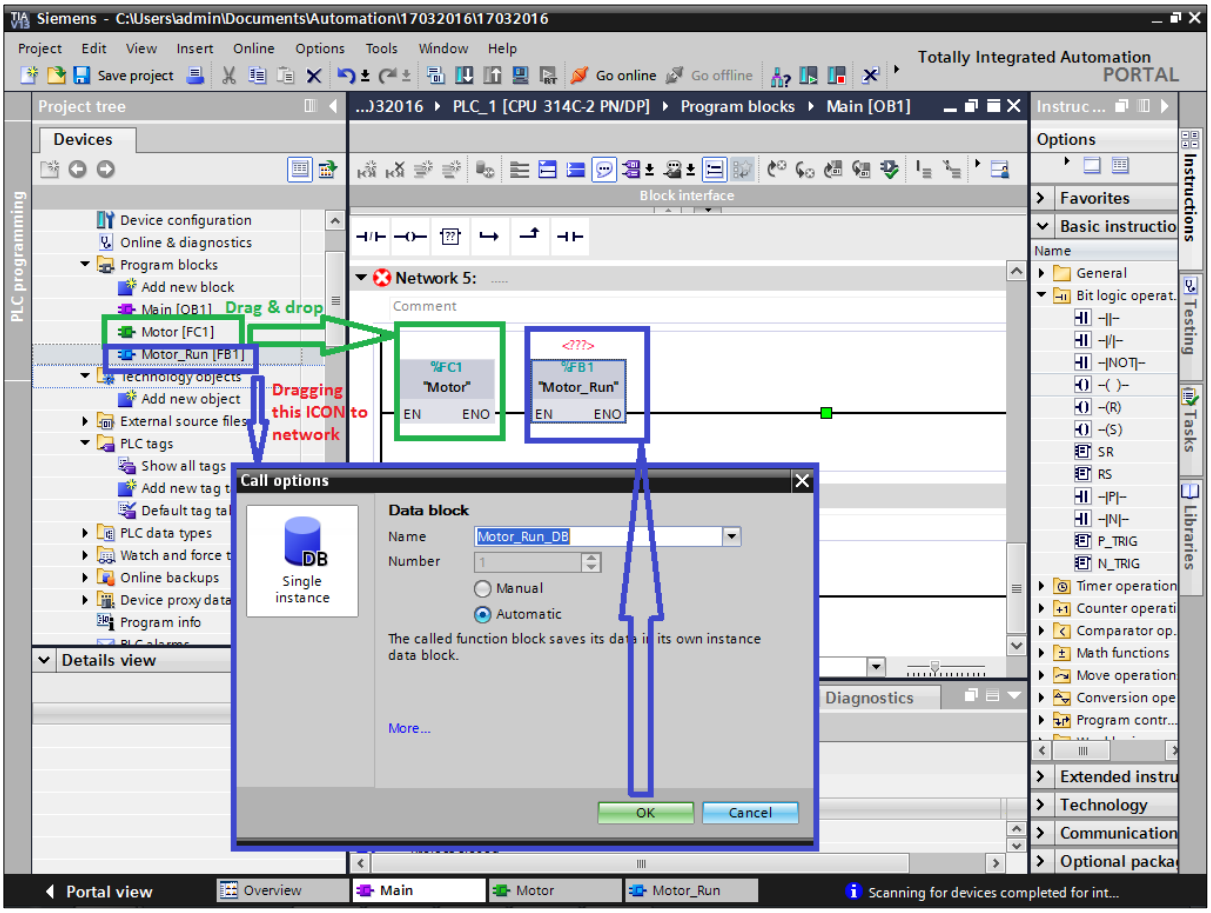
Save a block

By using "Save project" the entire project, and with that also the block, is saved on the hard drive. All changes made to the project are saved.

Rejecting a block

It is only possible to reject block changes by closing the entire project without saving. All changes made in the project are rejected.

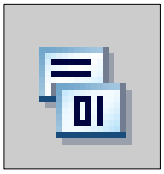
Block Calls



If a block calls another block, the instructions of the called block are executed. Only when the execution of the called block is completed, is the execution of the calling block taken up again and execution continues with the instruction that follows the block call.

The block call can be programmed using drag & drop or copy & paste.

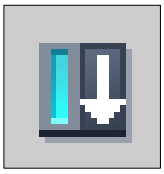
Compiling a Block



Whatever is selected (highlighted) in the Project tree is compiled (in the example shown only "FC_ConvMotor" FC16 is compiled). Individual blocks, the complete program ("Program blocks" selected) or the complete station with software and hardware ("Station" selected) can be compiled.

In the Inspectors window "Info -> Compile", the status of the compilation is displayed. If errors occurred during compilation, you can jump directly from the error entry to the error location by double-clicking.

Downloading Blocks into the CPU

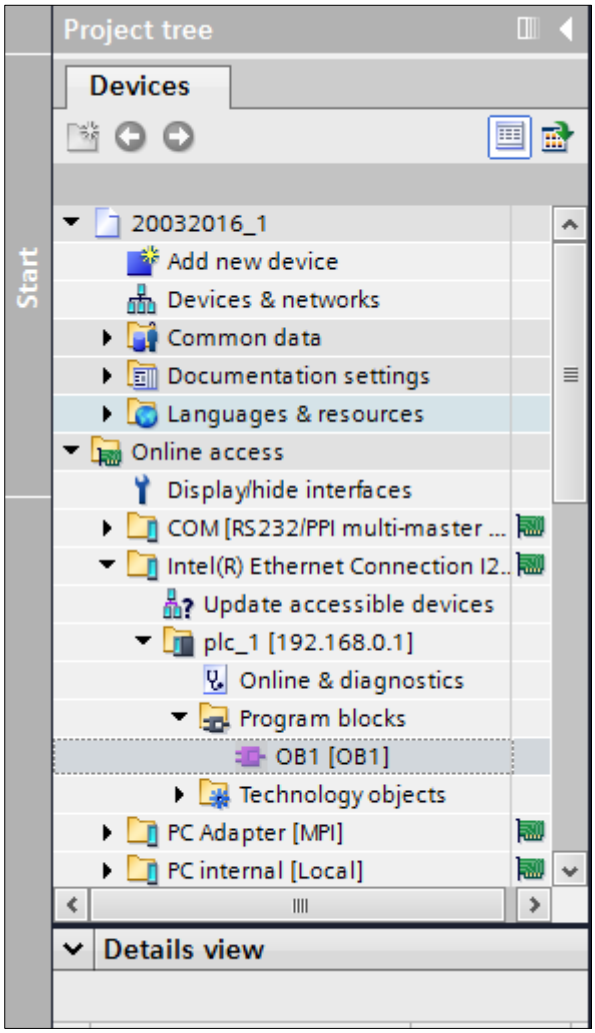


The project data that is downloaded into the devices is divided into hardware and software project data:

- Hardware project data results from configuring the hardware, networks, and connections. The first time you download, the entire hardware project data is loaded. In subsequent downloads, only changes to the configuration will be loaded.
- Software project data involves the blocks of the user program. The first time you download, the entire software project data is loaded. For subsequent downloads, you can determine whether the complete software or only the software changes should be downloaded.

• All:	• Hardware and software
• Hardware configuration:	• Hardware only
• Software:	• Changed blocks only
• Software (all blocks):	• All blocks

"Upload" blocks from device



Using the "Online access", program blocks can be uploaded into the project from any PLC. All blocks with complete symbols and comments are uploaded into the project from the CPU.

Binary operations

Binary Logic operations: AND, OR

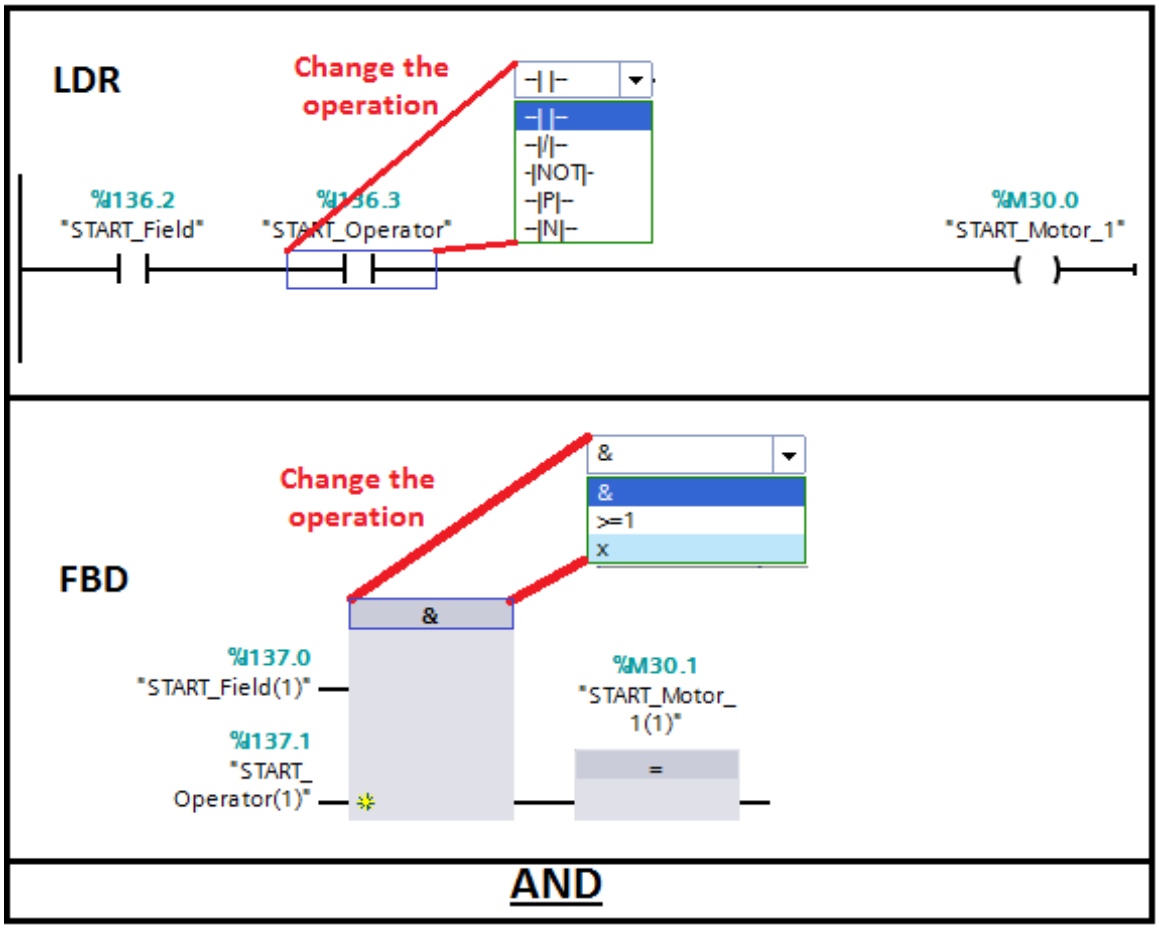
AND and OR logic operations

With the AND and OR logic operations, basically all binary operands can be checked, even outputs. Instead of individual operands, the results of other logic operations can also be further logically linked. Also, the logic operations themselves can be linked.

All inputs of the logic operations can be programmed as check for signal state or Status '0' and '1', regardless of whether a hardware NO contact or NC contact is connected in the process.

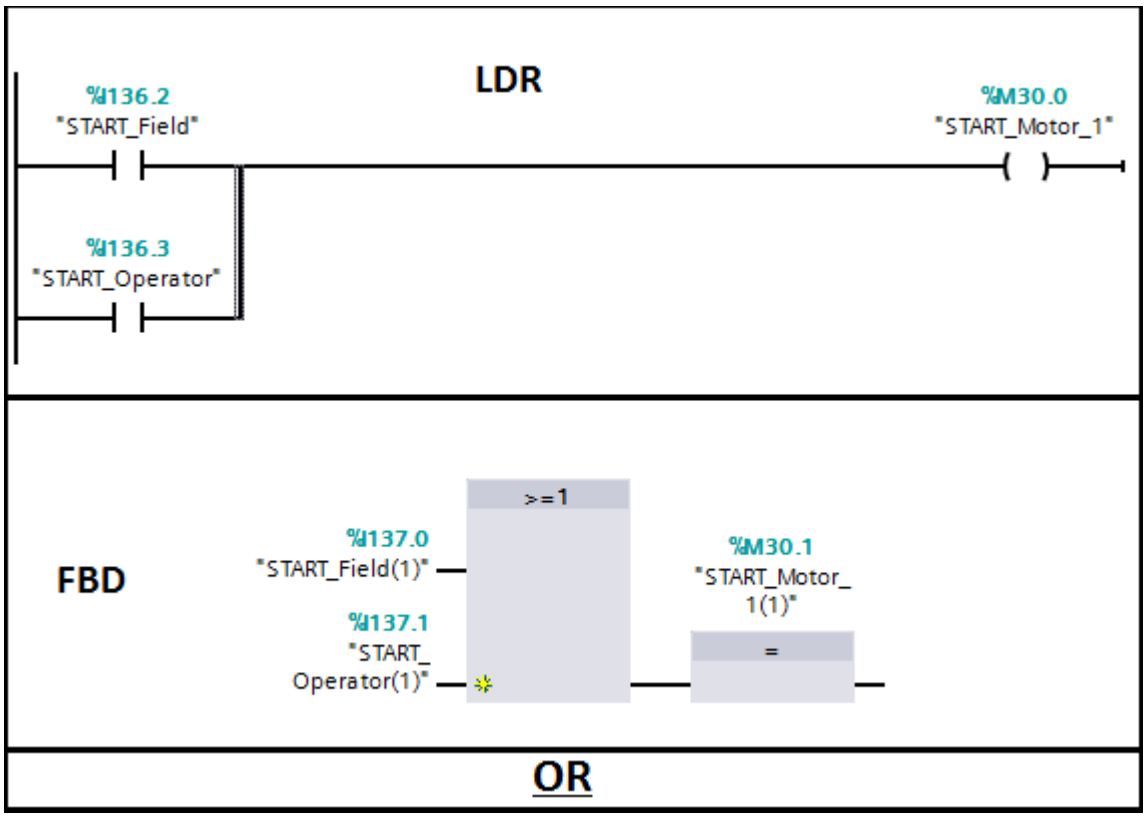
AND logic operation

For an AND logic operation, the result of logic operation (RLO) = '1', when **all** input signals have Status '1'.



OR logic operation

For an OR logic operation, the result of logic operation (RLO) = '1', when **at least one** input signal is Status '1'.



Sensors and Symbols

Process			Interpretation in the PLC Program				
The sensor is a ...	The sensor is ...	Voltage present at input?	Signal state at input	Check for signal state "1"		Check for signal state "0"	
				Symbol / Instruction	Result of check	Symbol / Instruction	Result of check
NO contact	activated	yes	1	*LAD:* "NO contact"	"Yes" 1	*LAD:* "NC contact"	"No" 0
	not activated	no	0		"No" 0		"Yes" 1
NC contact	activated	no	0	*FBD:* &	"No" 0	*FBD:* &	"Yes" 1
	not activated	yes	1	*STL:* A I x.y	"Yes" 1	*STL:* AN I x.y	"No" 0

Process

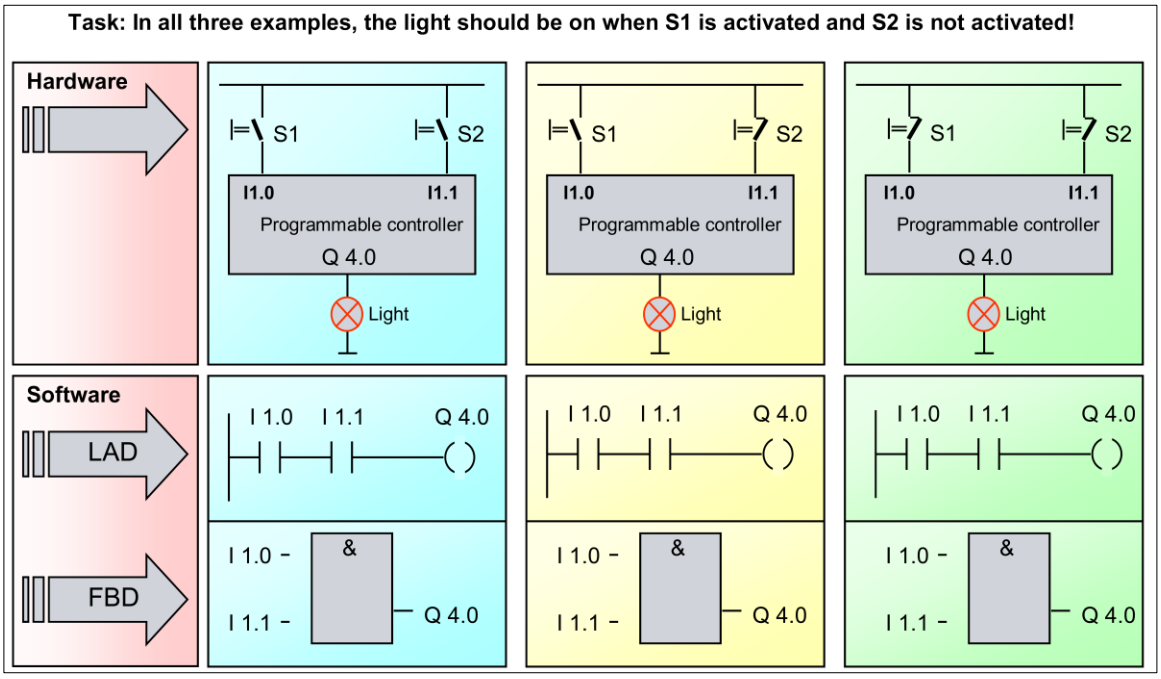
The use of normally open or normally closed contacts for the sensors in a controlled process depends on the safety regulations for that process. Normally closed contacts are always used for limit switches and safety switches, so that dangerous conditions do not arise if a wire break occurs in the sensor circuit. Normally closed contacts are also used for switching off machinery for the same reason.

Symbols

In LAD, a symbol with the name "NO contact" is used for checking for signal state "1" and a symbol with the name "NC contact" to check for signal state "0". It makes no difference whether the process signal "1" is supplied by an activated NO contact or a non-activated NC contact.

Example

If an NC contact in the machine is not activated, the signal in the process image table will be "1". You use the NO contact symbol in LAD to check for a signal state of "1". General: The "NC contact" symbol delivers the result of check "1" when the checked address state or status is "0".



Signal State & Result of Logic Operation

In Siemens PLCs the Status Word is an internal CPU register used to keep track of the state of the instructions as they are being processed. In order to use STL more effectively it is important to understand the Status Word and its functions.

Each bit in the Status Word has a specific function to keep track of bit logic (RLO, STA), math (OV, OS), comparison operations (CC0, CC1) and whether the logic should continue, be nested or start new (/FC, OR, BR). Only the first 9 of the 16 bits are used.

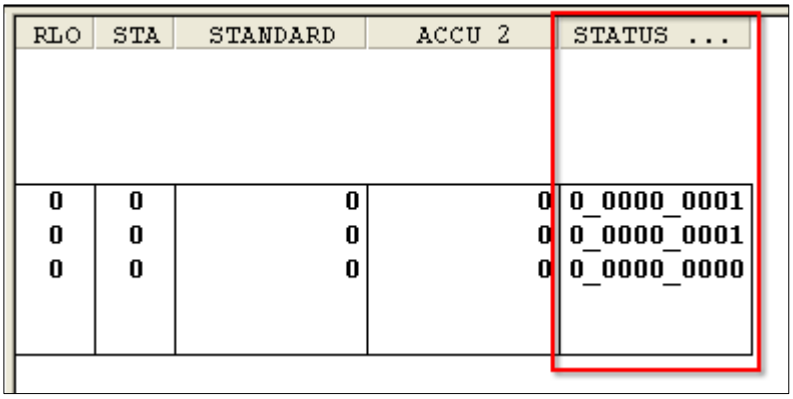
Bit Positions

8	7	6	5	4	3	2	1	0
BR	CC0	CC1	OV	OS	OR	STA	RLO	/FC

Each instruction may do the following to each bit in the status word.

-	No read or write
*	Read
x	May write "1" or "0"
0	Reset to "0"
1	Set to "1"

The status word can be seen by displaying the STATUS column while monitoring in STL view. The RLO (bit 1) and the STA (bit 2) are also displayed in the RLO and STA column.



RLO	STA	STANDARD	ACCU 2	STATUS ...
0	0	0	0	0_0000_0001
0	0	0	0	0_0000_0001
0	0	0	0	0_0000_0000

The Most Important Status Word Bits

/FC “First Check” (bit 0)

If the /FC bit is a 0 then the instruction is considered to be the first instruction being processed. If the /FC is a 1 then the instruction being scanned will use the logic from the previous instruction. Certain instructions like =, S and R will set the /FC bit to 0 thus starting new logic after it. Other instructions like A or O will set the /FC bit to 1 signaling to combine the logic with the next instruction.

RLO “Result of Logic Operation” (bit 1)

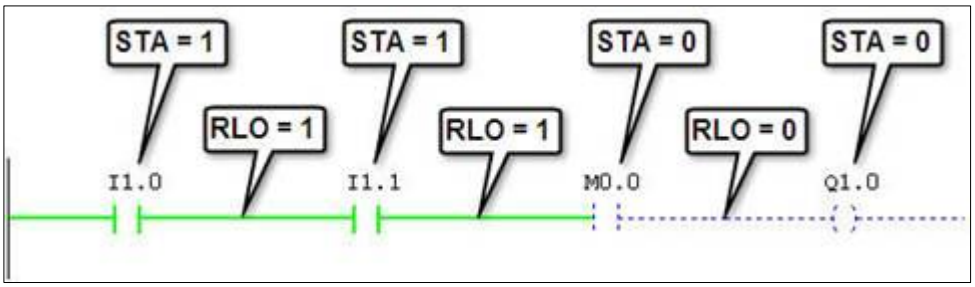
The RLO bit stores the running logic state of the currently processing instruction. Certain bit logic and comparison instruction will turn the RLO to a 1 when the condition is TRUE and write a 0 when the condition is FALSE.Â Other instructions read the RLO (=, S, R) to determine how they are to execute.

STA “Status” (bit 2)

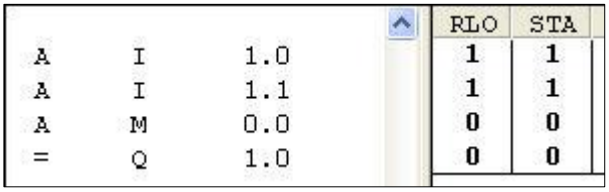
The STA bit reflects the state of the current Boolean address.

Help with RLO, STA and /FC

If you are used to ladder logic and struggling to understand the purpose of the RLO and STA it may help to visualize a rung like below. The STA is used to keep track of the state of the addresses. The RLO is used to keep track of the state of the rung.

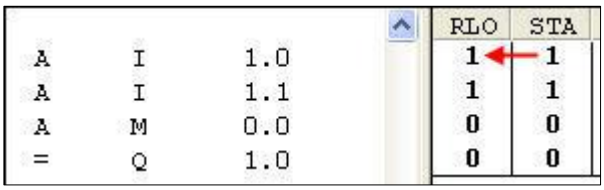


The equivalent STL is shown below.

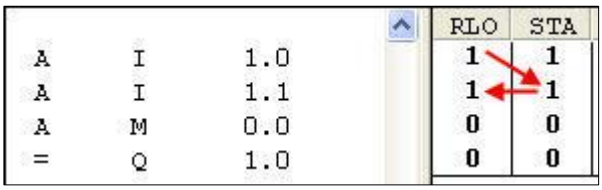


It steps through the logic as follows:

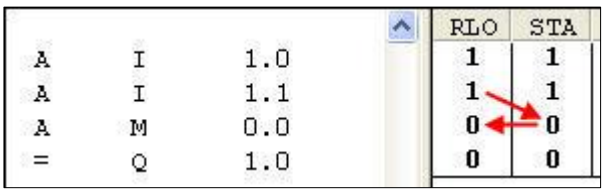
1. At the start the First Check bit (/FC) is zero so an AND instruction will logically mirror the Status bit (STA) over to the Result of Logic Operation (RLO). In this case the address I0.0 is 1 so the STA is one and the result of the logic (RLO) will be 1. The 'A' instruction writes a 1 to /FC.



2. On the second line, the /FC bit is now 1 indicating that this line needs to use the RLO from the previous line. The address I1.1 is on so the STA = 1. The RLO from the last line is 1 and this is with the current STA with a result of 1 in the current RLO.

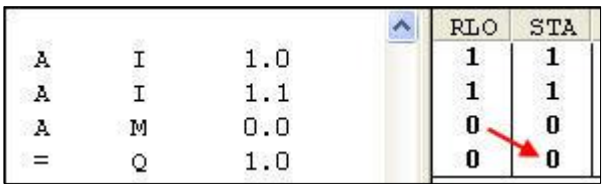


3. The same thing happens on the second line but this time 1 and 0 makes the current RLO = 0.



4.
5. The fourth is the Assign instruction which takes the RLO and writes it out to the corresponding address. In this case the final RLO = 0 so the output will be off. If

M0.0 was 1 then the operation will evaluate to true making the RLO = 1 which will then turn on the output Q1.0.



The Other Status Bits

OR (bit 3)

The OR bit is used for combining AND functions before OR functions.

OS "Overflow Stored" (bit 4)

In the event of an overflow (OV bit 5) the OS bit will store the value even after the OV bit has been reset. The following commands reset the OS bit: JOS (Jump if OS=1), block call instructions, block end instructions.

OV "Overflow" (bit 5)

The OV bit is set by a math instruction with floating point numbers after a fault has occurred (overflow, illegal operation, comparison unordered). The OV bit is reset when the fault is eliminated.

CC0, CC1 "Condition Code" (bits 6 and 7)

The Condition Code bits provide results for comparison and math instructions.

Comparison Instructions

CC 1	CC 0	Meaning
0	0	ACCU 2 = ACCU 1
0	1	ACCU 2 < ACCU 1
1	0	ACCU 2 > ACCU 1
1	1	Unordered (floating point comparison only)

Math Instructions, without Overflow

CC 1	CC 0	Meaning
0	0	Result = 0
0	1	Result < 0
1	0	Result > 0

Integer Math Instructions, with Overflow

CC 1	CC 0	Meaning

0	0	Negative range overflow in ADD_I and ADD_DI
0	1	Negative range overflow in MUL_I and MUL_DI
1	0	Negative range overflow in ADD_I, ADD_DI, SUB_I, and SUB_DI
1	1	Division by 0 in DIV_I, DIV_DI, and MOD_DI

Floating Point Math Instructions, with Overflow

CC 1	CC 0	Meaning
0	0	Gradual underflow
0	1	Negative range overflow
1	0	Positive range overflow
1	1	Not a valid floating-point number

Shift and Rotate Instructions

CC 1	CC 0	Meaning
0	0	Bit shifted out = 0
1	0	Bit shifted out = 1

Word Logic Instructions

CC 1	CC 0	Meaning
0	0	Result = 0
1	0	Result <> 0

BR “Binary Result” (bit 8)

The Binary Result transfers the result of the operations onto the next instruction for reference. When the BR bit is 1 it enables the output of the block (ENO) to be TRUE and thus allow other blocks after it to be processed. The SAVE, JCB and JNB instructions set the BR bit.

Binary Logic Operations-Exclusive-OR (XOR)

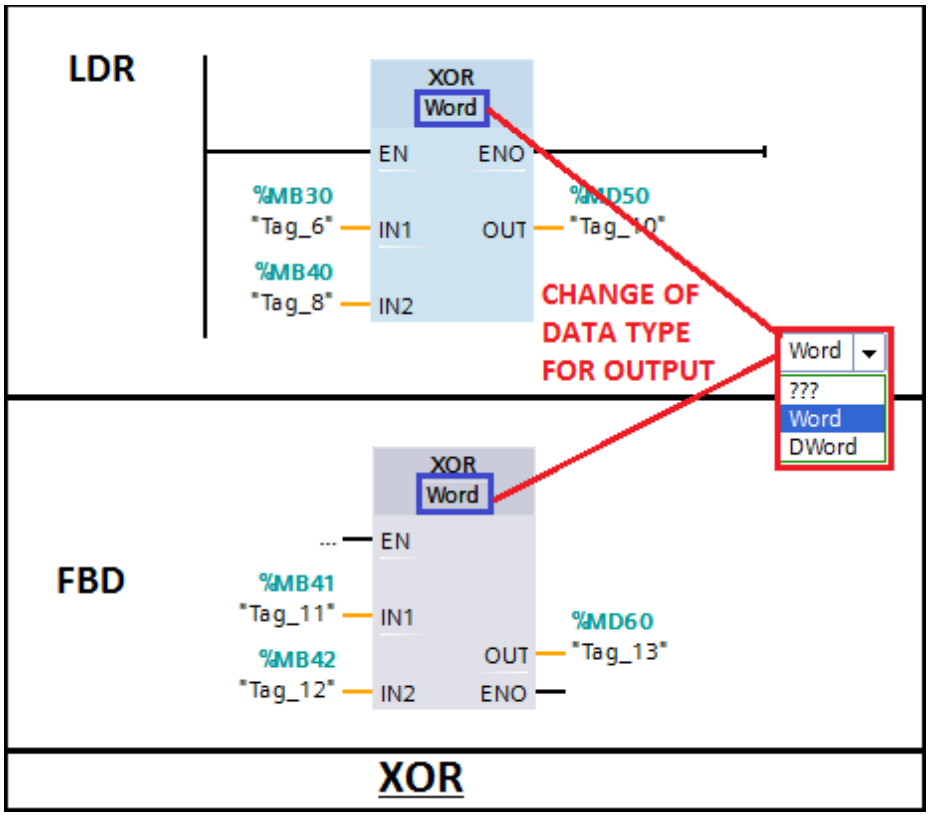
XOR logic operation

With the XOR logic operation, basically all binary operands can be checked, even outputs. Instead of individual operands, the results of other logic operations can also be further logically linked. Also, the logic operations themselves can be linked.

All inputs of the logic operations can be programmed as check for signal state or Status '0' and '1', regardless of whether a hardware NO contact or NC contact is connected in the process.

- For an XOR logic operation with 2 inputs, the result of logic operation (RLO) = '1', when one and only one of the two inputs signals is Status '1'.

- For an XOR logic operation with more than 2 operands, the RLO:
- = '1', when an uneven number of checked operands has Status '1'
- = '0', when an even number of checked operands has Status '1'.



XOR in the programming languages FBD and LAD

In the LAD programming language, there is no explicit XOR logic operation. It must be generated by programming the discrete instructions shown in the picture above.

Assignment, set, Reset, Not

Assignment

With an assignment, the specified operand is always assigned the current RLO as status. The assigned RLO remains available after the assignment and can be assigned to a further operand or it can be further logically linked.

Set

If RLO = "1", the specified operand is assigned Status '1'; if RLO = "0", the status of the operand remains unchanged.

Reset

If RLO = "1", the specified operand is assigned Status ' 0'; if RLO = "0", the status of the operand remains unchanged.

NOT

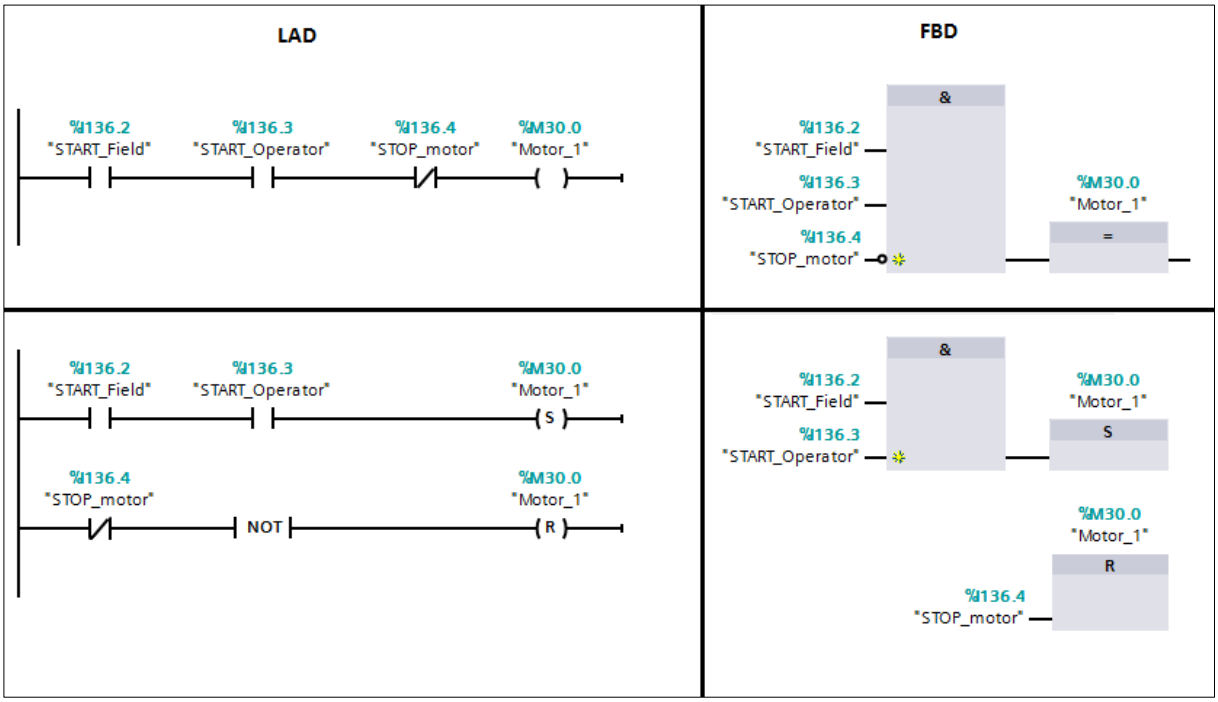
The NOT instruction inverts the result of logic operation (RLO). If, in the example shown, the RLO of the AND logic operation = '1', the NOT instruction inverts it to RLO '0' and the Set instruction is not executed (the status of "Tag_3" (Q20.0) then remains unchanged).

If the RLO of the AND logic operation = '0', the NOT instruction inverts it to RLO '1' and the Set instruction is executed ("Tag_3" (Q20.0) is assigned Status '1').\

EXAMPLE:

One switch is inside the field and another switch is on the operator panel to start motor1. One common stop is already there which is used for the stopping purpose of the motor.

Different ways to explain it is given below.



Flip-Flops

Flip Flop

A flip flop has a Set input and a Reset input. The memory bit is set or reset, depending on which input has an RLO=1.

Priority

If there is an RLO=1 at both inputs at the same time, the priority must be determined. In LAD and FBD there are different symbols for Dominant Set and Dominant Reset memory functions. In STL, the instruction that was programmed last has priority.

Note

With a warm restart of the CPU, all outputs are reset. That is, they are overwritten with the state ‘0’.

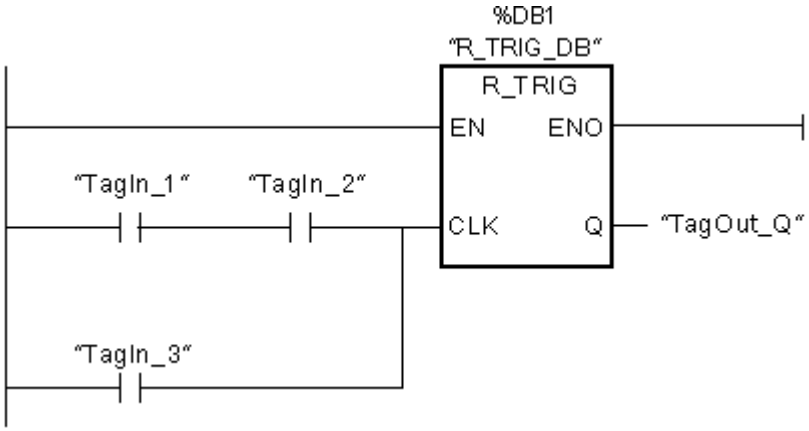
Signal-Edge Detection

Signal edge detection (P, N)

With a signal edge detection it is possible to detect the status change of an individual operand (in the example "T_ON") from '0' to '1' (rising or positive edge) or from '1' to '0' (falling or negative edge). If this is the case, the instruction supplies RLO '1' as the result, which can be further logically linked (in the example as set condition) or assigned to another operand (for example, a memory bit) as status. In the following cycle, the instruction then once again supplies '0' as the result even if "T_ON" still is status '1'.

The instruction compares the current status of the operand "T_ON" with its status in the previous program cycle. This status is stored in a so-called edge memory bit for this (in the example "M_Fl_ON"). It must be ensured that the status of this edge memory bit is not overwritten at another location in the program. For each edge detection, a separate edge memory bit must be used accordingly, even then when the same operand (in the example, "T_ON") is detected once again, for example, in another block!

The following example shows how the instruction works:



The previous state of the tag at the CLK input is stored in the "R_TRIG_DB" tag. If a change in the signal state from "0" to "1" is detected in the "TagIn_1" and "TagIn_2" operands or in the "TagIn_3" operand, the "TagOut_Q" output has signal state "1" for one cycle.

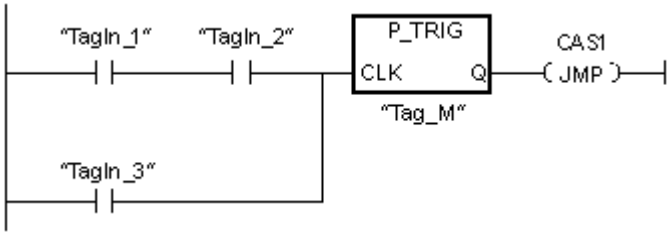
RLO-Edge Detection

RLO edge detection (P=, P_TRIG)

An RLO edge detection detects whether the status of an individual operand or the RLO of a logic operation has changed from '0' to '1' (rising or positive edge) or from '1' to '0' (falling or negative edge). If this is the case, both edge detections supply RLO '1' as a result to their output for the duration of one cycle. In the following cycle, the instructions then once again supply RLO '0' as a result, even if the status or the RLO of the operand or the logic operation has notchanged. The instructions compare the current status of the operand or the RLO of the logic operation with its status in the previous program cycle which is stored in a so-called edge memory bit for this (in the example, "M_Fl_Count_pos" or "M_Fl_Count_neg). It must be

ensured that the status of this edge memory bit is not overwritten at another location in the program. For every edge detection, a separate edge memory bit must be used accordingly, even then when the same operand is detected once again, for example, in another block!

The following example shows how the instruction works:



The RLO of the previous query is saved in the edge memory bit "Tag_M". If a "0" to "1" change is detected in the signal state of the RLO, the program jumps to jump label CAS1.

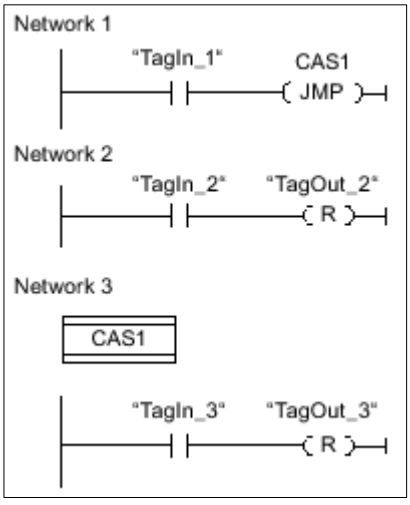
Jump Instructions JMP, JMPN, RET

Jump Instructions JMP and JMPN

With the jump instructions JMP and JMPN, the linear execution of the program can be interrupted within a block and continued in another network. With the jump instruction, a Label is specified which also identifies the target network. The specified label must be located in the same block and be unique. Each label can be jumped to from several locations. The jump can take place in networks with higher (forwards) or lower numbers (backwards).

1. JMP:

   - For RLO ='1', the jump into the target network is executed; for RLO ='0', the jump is not executed and the linear program execution continues.
   - Example
   - The following example shows how the instruction works:
   -



- If operand "TagIn_1" has the signal state "1", the "Jump if RLO = 1" instruction is executed. The linear execution of the program is interrupted and continues in Network 3, which is identified by the jump label CAS1. If the "TagIn_3" input has the signal state "1", the "TagOut_3" output is set.

2. JMPN:

- For RLO ='0', the jump into the target network is executed; for RLO ='1', the jump is not executed and the linear program execution continues.

End block execution RET

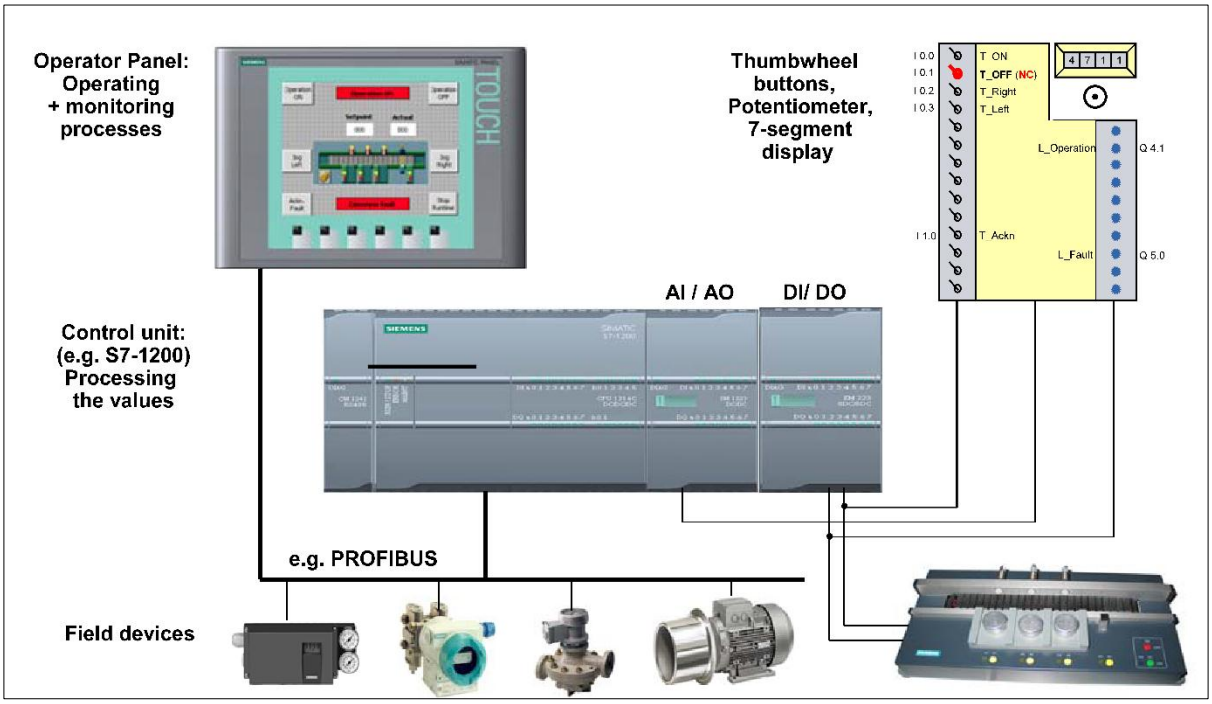
With the instruction RET, the program execution of the entire block is ended. The program execution is then continued in the calling block with the instruction that follows the call of this block.

Digital Operation

Acquiring, Processing and Outputting Data

Binary/Digital Processing

True logic control systems are recognizable in the fact that they exclusively process binary data. The performance of today‘s control computer, as well as tasks in the data processing, quality control areas, among others, has increased the importance of digital data processing using PLCs. Digital process variables can be found in all areas of open-loop control such as in connected devices for process operating and monitoring or in the control of field devices.



Operating and Monitoring

The goal of process monitoring is to provide the operator with up to the-minute information about the working machine or system quickly, concisely and clearly as well as the opportunity to intervene and control and influence the process. While in the past mostly simple, that is, "dumb" input and output devices, such as 7-segment displays and thumbwheel buttons were used to display and enter digital values, today "intelligent" operating and monitoring devices are frequently connected to a PLC. Depending on the type of device connected, different number formats for the coding of data are used to transmit data between device and PLC, as well as for storing and processing data in the PLC.

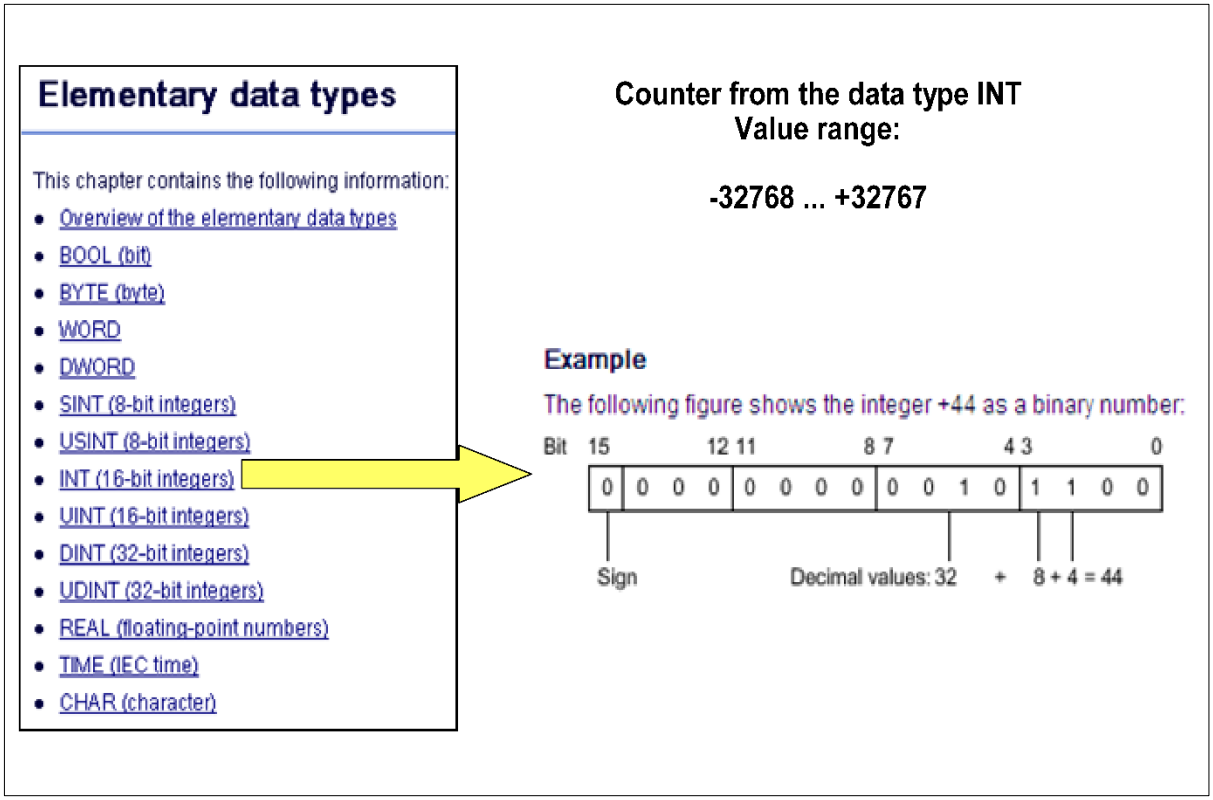
Field Devices

Today as well, field devices that acquire process data or that control the process are supplied directly with digital variables through field bus systems. The connection of field devices, such as drives or weighing systems, using analog input and output modules is becoming more and more a thing of the past.

Integer (INT, 16 bit integer) Data Type

Integer Data Type

An Integer data type value is a whole number value, that is, a value without a decimal point (16- bit Integer). SIMATIC S7 stores Integer data type values with sign in 16 bit code. This results in the value range shown in the picture above. As well, SIMATIC S7 provides arithmetic operations for processing Integer values. When selecting one of the Integer data types, the value range of the counter is defined.



Decimal

STEP7 uses the Decimal (not BCD!) display format to specify the constants of the Integer data type with sign and without explicit format description. The use of constant Integer values in the Binary and Hexadecimal display formats is possible in principle, but because of the poor legibility, they are more or less not suitable. For this reason, the syntax of STEP7 provides the specification of Integer values only in the decimal display format.

Double Integer (DINT, 32 bit integer) Data Type

An operand of data type DINT (Double INT) has a length of 32 bits and consists of two components: a sign and a numerical value in the two's complement. The signal states of bits 0 to 30 represent the number value. The signal state of bit 31 represents the sign. The sign may assume "0" for the positive, or "1" for the negative signal state.

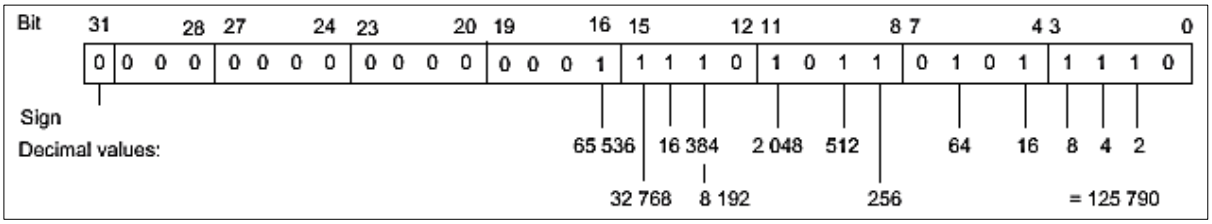
An operand of data type DINT occupies four BYTE in the memory.

The following table shows the properties of data type DINT:

Length (bits)	Format	Range of values	Examples of value input
32	Signed integers	-2147483648 to +2147483647	125790, DINT#125790, L#275

Example

The following figure shows the integer +125790 as a binary number:



Real (Floating Point Number, 32 bit integer) Data Type

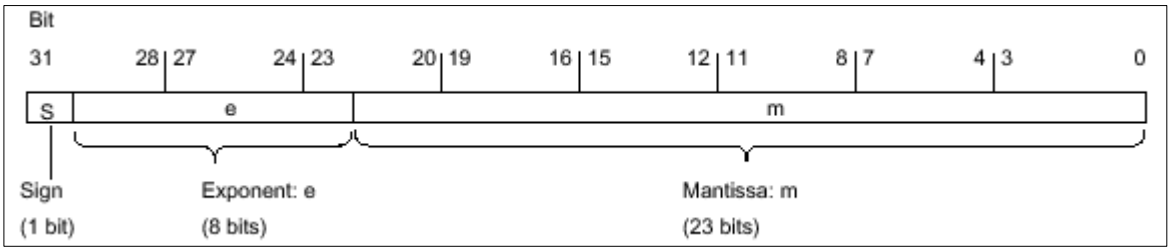
Description

Operands of the data type REAL have a length of 32 bits and are used to display floating-point numbers. An operand of the REAL data type consists of the following three components:

- Sign: The sign is determined by the signal state of bit 31. The bit 31 assume the value "0" (positive) or "1" (negative).
- 8-bit exponents to basis 2: The exponent is increased by a constant (base, +127), so that it has a value range of 0 to 255.
- 23-bit mantissa: Only the fraction part of the mantissa is shown. The integer part of the mantissa is always 1 with normalized floating-point numbers and is not stored.

The REAL data type is processed with a precision of 7 digits after the decimal point.

The following figure shows the structure of the REAL data type:



Note

With floating-point numbers, only the precision defined by the IEEE754 standard is stored. Additionally specified decimals are rounded off according to IEEE754.

The number of decimal places may decrease for frequently nested arithmetic calculations.

If more decimal places are specified than can be stored by the data type, the number is rounded to the corresponding value of the precision allowed by this value range.

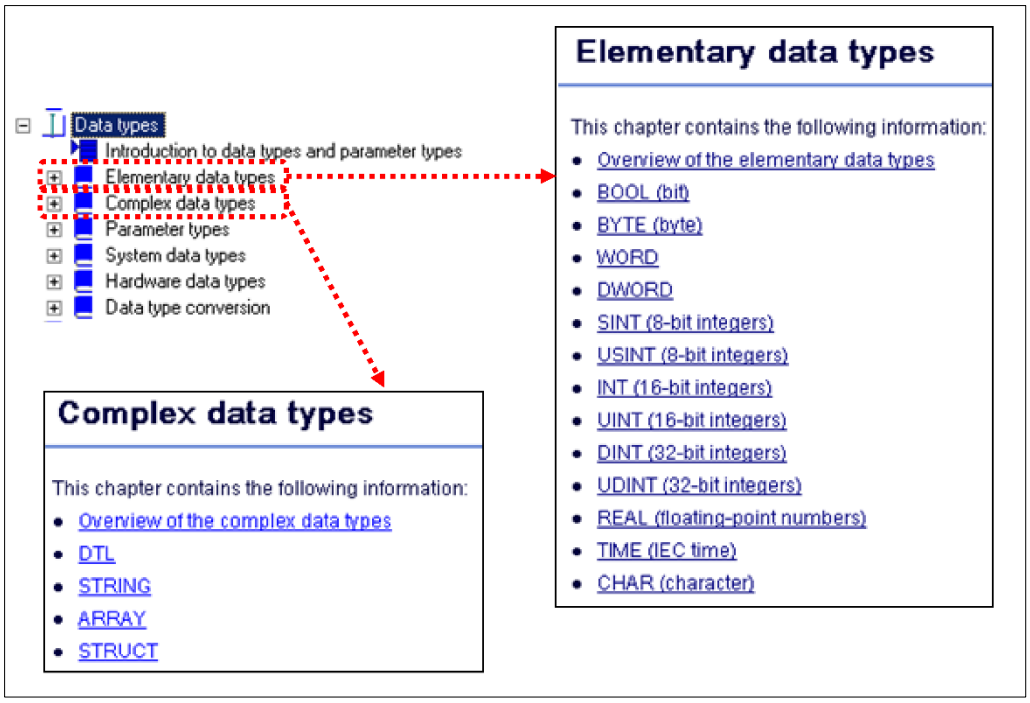
The following table shows the properties of data type REAL:

Length (bits)	Format	Range of values	Examples of value input
32	Floating-point numbers according to IEEE754	-3.402823e+38 to -1.175 495e-38	1.0e-5, REAL#1.0e-5
	Floating-point numbers	±0 +1.175 495e-38 to +3.402823e+38	1.0, REAL#1.0

Data types

Data types

Variables are used to store data. The data type of a variable specifies its memory requirements, its value range and the representation of the variable value in the Editor. As well, the possible operations with which a variable can be processed can be obtained from the data type.



Elementary Data Types

Elementary data types are predefined in accordance with IEC 61131-3. They are never more than 32 bits long and can be loaded completely into the accumulators of the S7 processor. They are processed with elementary STEP 7 instructions.

1. Bit: Bit is a binary digit — the basic unit of information storage. A single bit is a one or a zero. While a single bit can define a Boolean value of True (1) or False (0)
2. Byte: The byte data type is an 8-bit storage. It has a minimum value of -128 and a maximum value of 127.
3. Word: The 16-bit data type can hold integer values in the range of −32,768 to 32,767.

4. Double Word: The 32-bit data type can used to store the real number (Floating point value).
5. Integer: An integer is a number that can be written without a fractional component. For example, 21, 4, 0, and −2048 are integers.
6. The set of integers consists of zero (0), the natural numbers (1, 2, 3 …), and their additive inverses (-1, -2, -3 …).
7. Double Integer: The 32-bit data type can hold integer values in the range of −2,14,74,83,648 to 2,14,74,83,647.
8. Floating Point: floating point is the representation of real number. It contains 32 bits

Data Type	Length (In Bits)	Representation
Bool	1	M0.0
Byte	8	MB1
Word	16	MW3
Double Word	32	MD10
Integer	16	MW25
Double Integer	32	MD37
Real	32	MD45

Complex Data Types

Complex data types contain data structures that can be made up of elementary and/or complex data types. Complex data types can be used for the declaration of variables only in global data blocks and within blocks for the declaration of local variables (TEMP, STAT) as well as parameters (IN, OUT and INOUT). Variables of complex data types cannot be completely processed with elementary instructions (such as, A, O, L, T, +I) but only the individual components of the elementary data type.

Counters

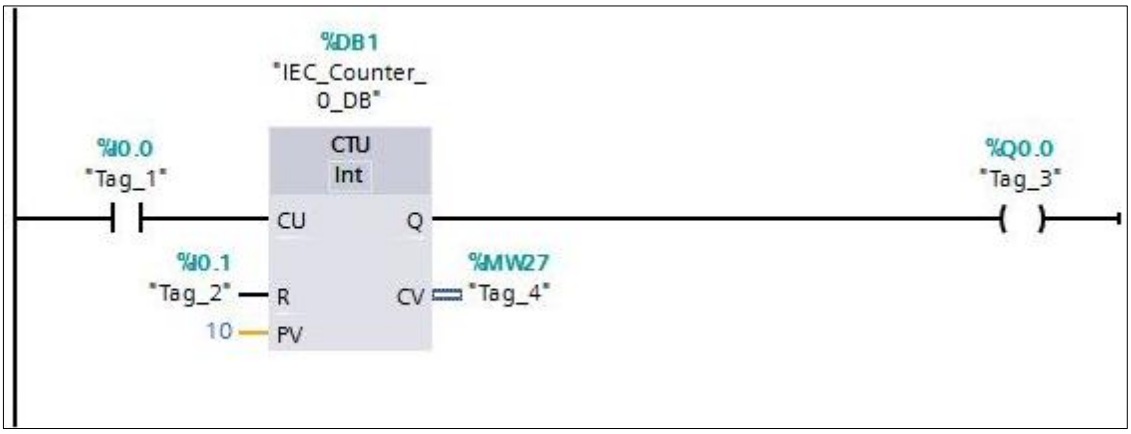
1. *CTU - Counter Up*

You can use the "Count up" instruction to increment the value at output CV. When the signal state at the CU input changes from "0" to "1" (positive signal edge), the instruction executes and the current counter value at the CV output is incremented by one. When the instruction executes for the first time, the current counter value at the CV output is set to zero. The counter value is incremented each time a positive signal edge is detected, until it reaches the high limit for the data type specified at the CV output. When the high limit is reached, the signal state at the CU input no longer has an effect on the instruction.

You can scan the counter status at the Q output. The signal state at the Q output is determined by the parameter PV. If the current counter value is greater than or equal to the value of the PV parameter, the Q output is set to signal state "1". In all other cases, the Q output has signal state "0".

The value at the CV output is reset to zero when the signal state at input R changes to "1". As long as the R input has signal state "1", the signal state at the CU input has no effect on the instruction.

When the signal state of the "I0.0" operand changes from "0" to "1", the "Count up" instruction executes and the current counter value of the operand "CV" is incremented by one. With each additional positive signal edge, the counter value is incremented until the high limit value of the data type (INT = 32767) is reached.



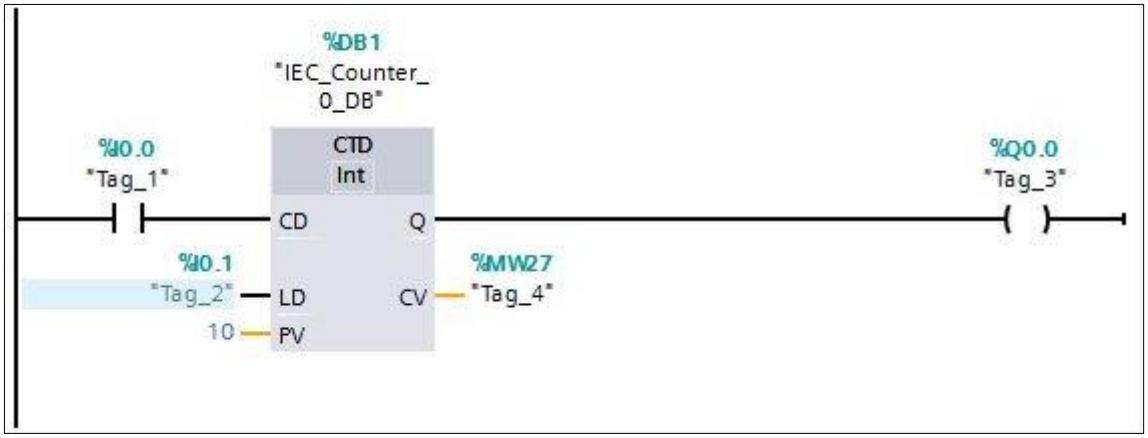
The value of the PV parameter is adopted as the limit for determining the "Q0.0" output. The "Q0.0" output has signal state "1" as long as the current counter value is greater than or equal to the value of the "PV" operand. In all other cases, the "Q0.0" output has signal state "0".

Parameters

Parameter	Declaration	Data type	Description
CU	Input	BOOL	Count up input
R	Input	BOOL	Reset input
PV	Input	Integers	Value at which the output QU is set.
Q	Output	BOOL	Status of the counter up
CV	Output	Integers	Current counter value

2. CTD - Counter Down

You can use the "Count down" instruction to decrement the value at output CV. When the signal state at the CD input changes from "0" to "1" (positive signal edge), the instruction executes and the current counter value at the CV output is decremented by one. When the



instruction executes the first time, the counter value of the CV parameter is set to the value of the PV parameter. Each time a positive signal edge is detected, the counter value is decremented until it reaches the low limit value of the specified data type. When the low limit is reached, the signal state at the CD input no longer has an effect on the instruction.

You can scan the counter status at the Q output. If the current counter value is less than or equal to zero, the Q output is set to signal state "1". In all other cases, the Q output has signal state "0".

The value at the CV output is set to the value of the PV parameter when the signal state at the LD input changes to "1". As long as the LD input has signal state "1", the signal state at the CD input has no effect on the instruction.

When the signal state of the "I0.0" operand changes from "0" to "1", the "Count down" instruction is executed and the value at the "CV" output is decremented by one. With each additional positive signal edge, the counter value is decremented until the low limit of the specified data type (INT = -32768) is reached.

The "Q0.0" output has signal state "1" as long as the current counter value is less than or equal to zero. In all other cases, the "Q0.0" output has signal state "0".

Parameters

Parameter	Declaration	Data type	Description
CD	Input	BOOL	Count down input
LD	Input	BOOL	Load input
PV	Input	Integers	Value at which the output QU is set.
Q	Output	BOOL	Status of the down-counter
CV	Output	Integers	Current counter value

3. CTUD - Counter Up and Down

You can use the "Count up and down" instruction to increment and decrement the counter value at the CV output. If the signal state at the CU input changes from "0" to "1" (positive signal edge), the current counter value is incremented by one and stored at the CV output. If the signal state at the CD input changes from "0" to "1" (positive signal edge), the counter value at the CV output is decremented by one. If there is a positive signal edge at the CU and CD inputs in one program cycle, the current counter value at the CV output remains unchanged.

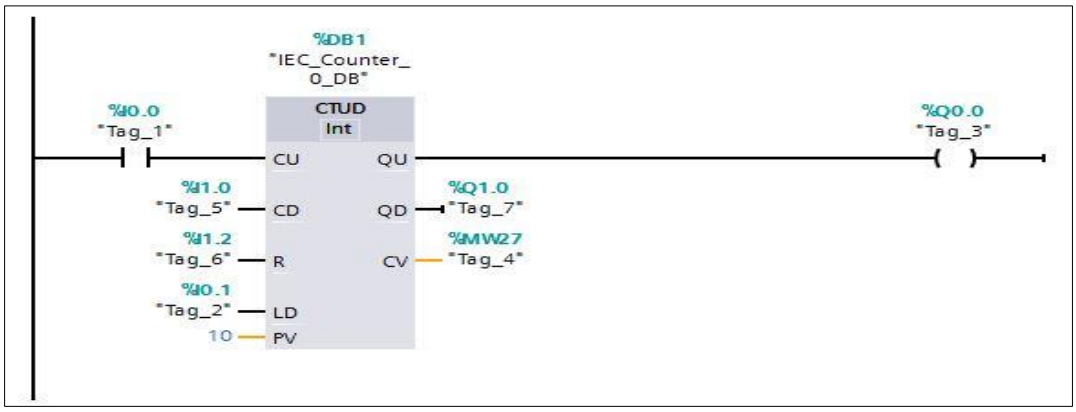
The counter value can be incremented until it reaches the high limit of the data type specified at the CV output. When the high limit value is reached, the counter value is no longer incremented on a positive signal edge. When the low limit of the specified data type is reached, the counter value is not decremented any further.

When the signal state at the LD input changes to "1", the counter value at the CV output is set to the value of the PV parameter. As long as the LD input has the signal state "1", the signal state at the CU and CD inputs has no effect on the instruction.

The counter value is set to zero when the signal state at the R input changes to "1". As long as the R input has signal state "1", a change in the signal state of the CU, CD and LD inputs has no effect on the "Count up and down" instruction.

You can scan the current status of the up counter at the QU output. If the current counter value is greater than or equal to the value of the PV parameter, the QU output is set to signal state "1". In all other cases, the QU output has signal state "0".

You can scan the current status of the down counter at the QD output. If the current counter value is less than or equal to zero, the QD output is set to signal state "1". In all other cases, the



QD output has signal state "0".If the signal state at the "I0.0" or "I1.0" input changes from "0" to "1" (positive signal edge), the "Count up and down" instruction is executed.

When there is a positive signal edge at the "I0.0" input, the current counter value is incremented by one and stored at the "CV" output.

When there is a positive signal edge at the "I1.0" input, the counter value is decremented by one and stored at the "CV" output.

When there is a positive signal edge at the CU input, the counter value is incremented until it reaches the high limit of 32767. If input CD has a positive signal edge, the counter value is decremented until it reaches the low limit value of INT = -32768.

The "Q0.0" output has signal state "1" as long as the current counter value is greater than or equal to the value at the "PV" input. In all other cases, the "Q0.0" output has signal state "0".

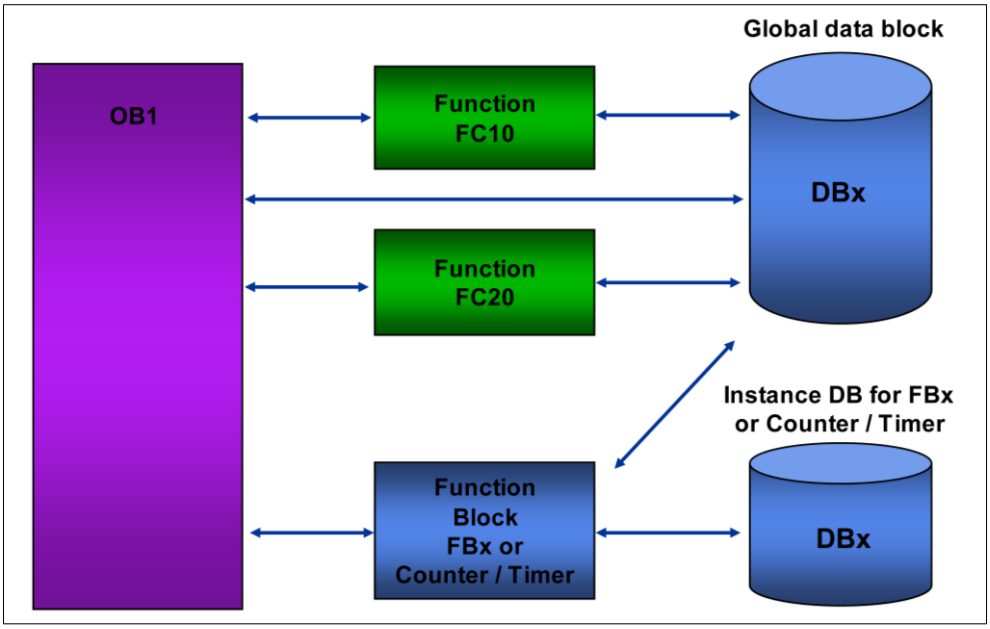
The "Q1.0" output has signal state "1" as long as the current counter value is less than or equal to zero. In all other cases, the "Q1.0" output has signal state "0".

Parameters

Parameter	Declaration	Data type	Description
CU	Input	BOOL	Count up input
CD	Input	BOOL	Count down input
R	Input	BOOL	Reset input
LD	Input	BOOL	Load input
PV	Input	Integers	Value at which the output QU is set.
QU	Output	BOOL	Status of the counter up
QD	Output	BOOL	Status of the down-counter

CV	Output	Integers	Current counter value

Counters/Timers instance data blocks



Overview

Data blocks are used for storing user data. Like logic blocks, data blocks take up space in the user memory. Data blocks contain variable data (such as numeric values) that is used in the user program. The user program can access the data in a data block with bit/byte/word or double word operations

Uses

You can use data blocks in different ways, depending on their contents. You differentiate between:

- Shared data blocks: These contain information that all the logic blocks (that would include OB1) in the user program can access.
- Instance data blocks: These are always assigned to a particular FB or functions such as Counter / Timer. The data of these instance DBs should only be processed by the FB or Counter / Timer function to which it is assigned.

Creating DBs

You can create global DBs with either the Program Editor or with a "user data type" (UDT) that you have already created. Instance data blocks are created when the FB / Counter / Timer is called.

Timer Function

1. *TP: Generate pulse*

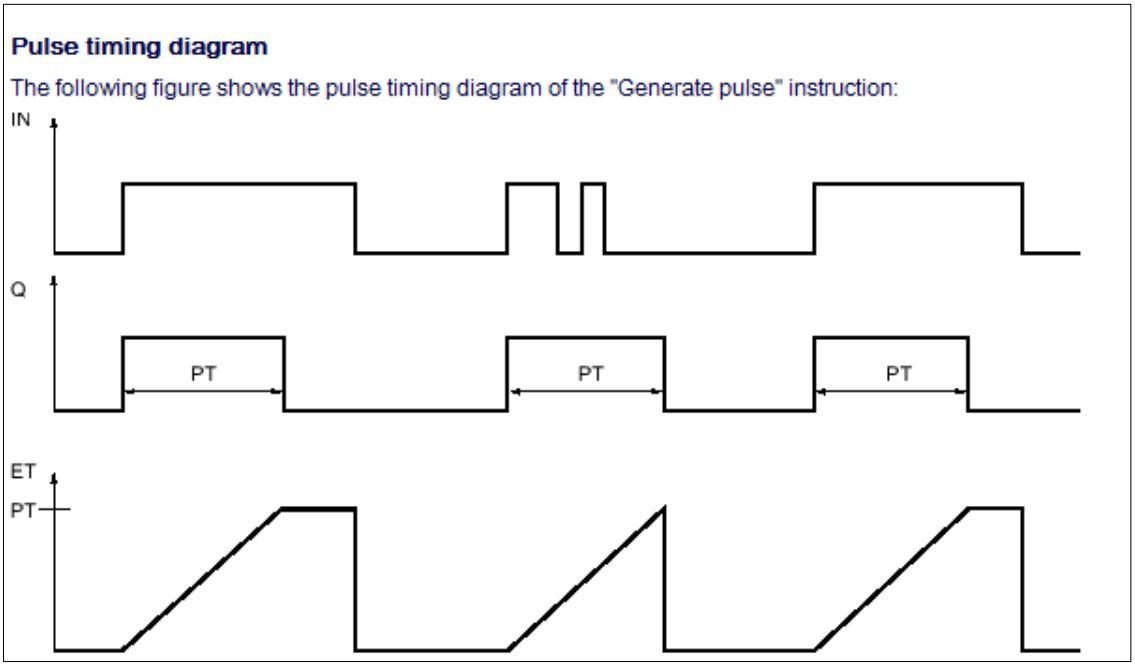
You can use the "Generate pulse" instruction to set the output Q for a programmed duration. The instruction is started when the result of logic operation (RLO) at input IN changes from "0" to "1" (positive signal edge). The programmed time PT begins when the instruction starts.

Output Q is set for the duration PT, regardless of the subsequent course of the input signal. Even if a new positive signal edge is detected, the signal state at the output Q is not affected as long as the PT time duration is running.

You can scan the current time value at the ET output. The time value starts at T#0s and ends when the value of duration PT is reached. When the duration PT is reached and the signal state at input IN is "0", the ET output is reset.

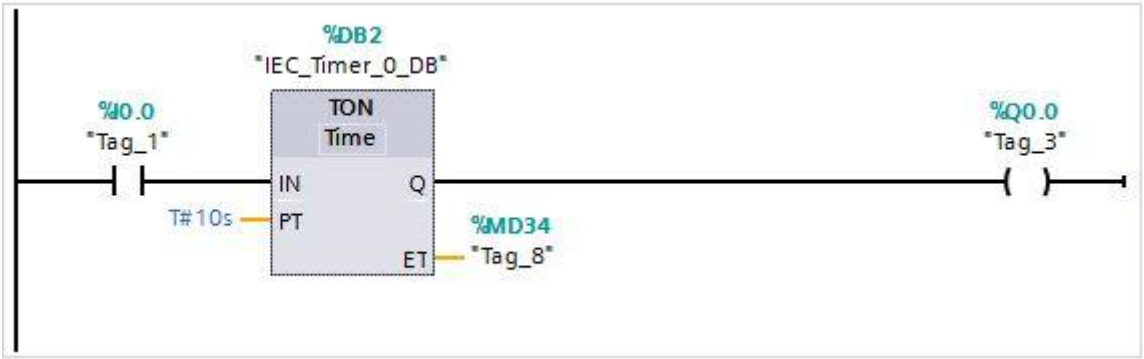
Parameters

Parameter	Declaration	Data type	Description
IN	Input	BOOL	Start input
PT	Input	TIME	Duration of the pulse. The value of the PT parameter must be positive.
Q	Output	BOOL	Pulse output
ET	Output	TIME	Current time value



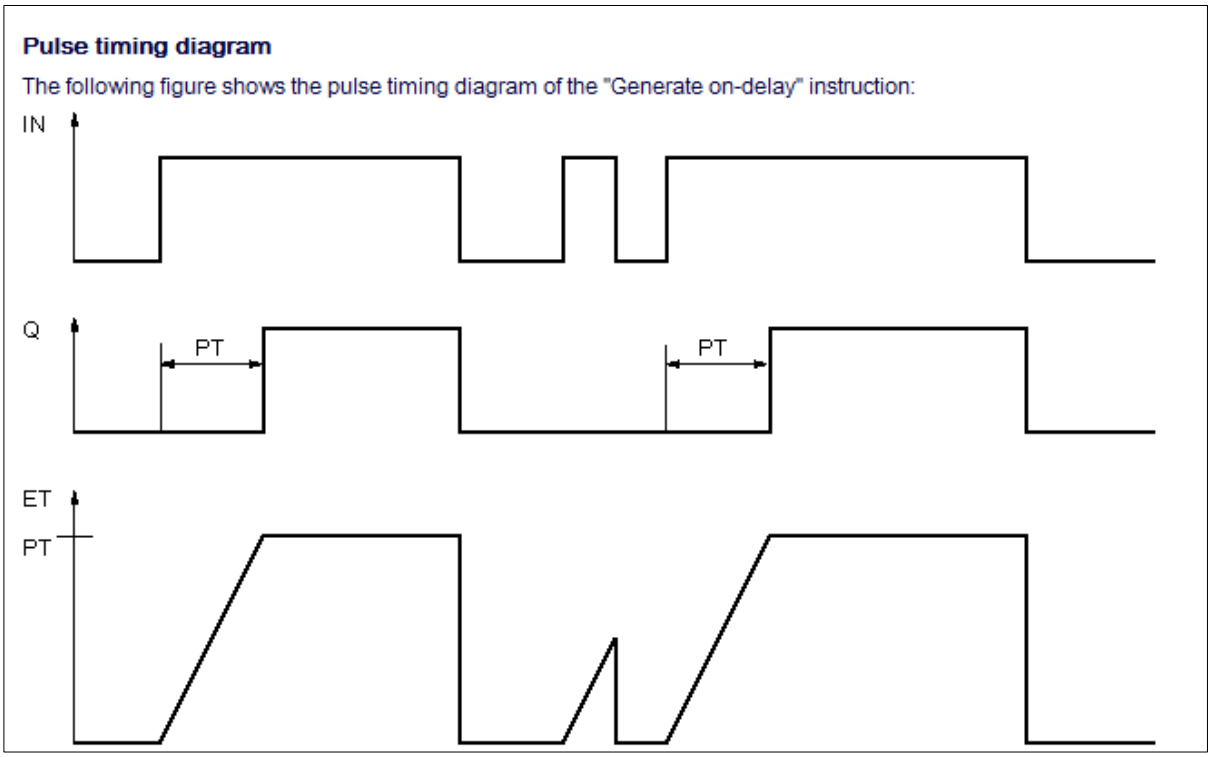
2. TON: Generate on-delay

You can use the "Generate on-delay" instruction to delay setting of the Q output by the programmed duration PT. The instruction is started when the result of logic operation (RLO)



at input IN changes from "0" to "1" (positive signal edge). The programmed time PT begins when the instruction starts. When the duration PT expires, the output Q has the signal state "1". Output Q remains set as long as the start input is still "1". When the signal state at the start input changes from "1" to "0", the Q output is reset. The timer function is started again when a new positive signal edge is detected at the start input.

The current time value can be queried at the ET output. The time value starts at T#0s and ends when the value of duration PT is reached. The ET output is reset as soon as the signal state at the IN input changes to "0".

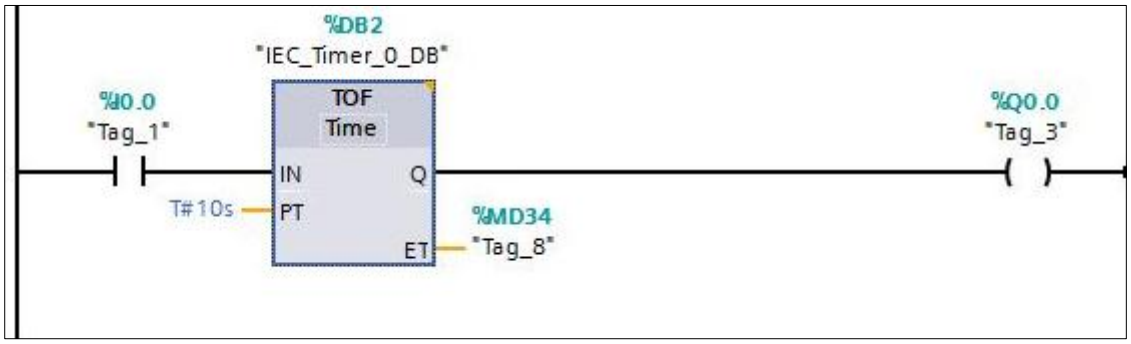


Parameters

Parameter	Declaration	Data type	Description
IN	Input	BOOL	Start input
PT	Input	TIME	Duration of the on delay. The value of the PT parameter must be positive.
Q	Output	BOOL	Output that is set when the time PT expires
ET	Output	TIME	Current time value

3. TOF: Generate off-delay

You can use the "Generate off-delay" instruction to delay resetting of the Q output by the programmed duration PT. The Q output is set when the result of logic operation (RLO) at input



IN changes from "0" to "1" (positive signal edge). When the signal state at input IN changes back to "0", the programmed time PT starts. Output Q remains set as long as the duration PT is running. When duration PT expires, the Q output is reset. If the signal state at input IN changes to "1" before the PT time duration expires, the timer is reset. The signal state at the output Q continues to be "1".

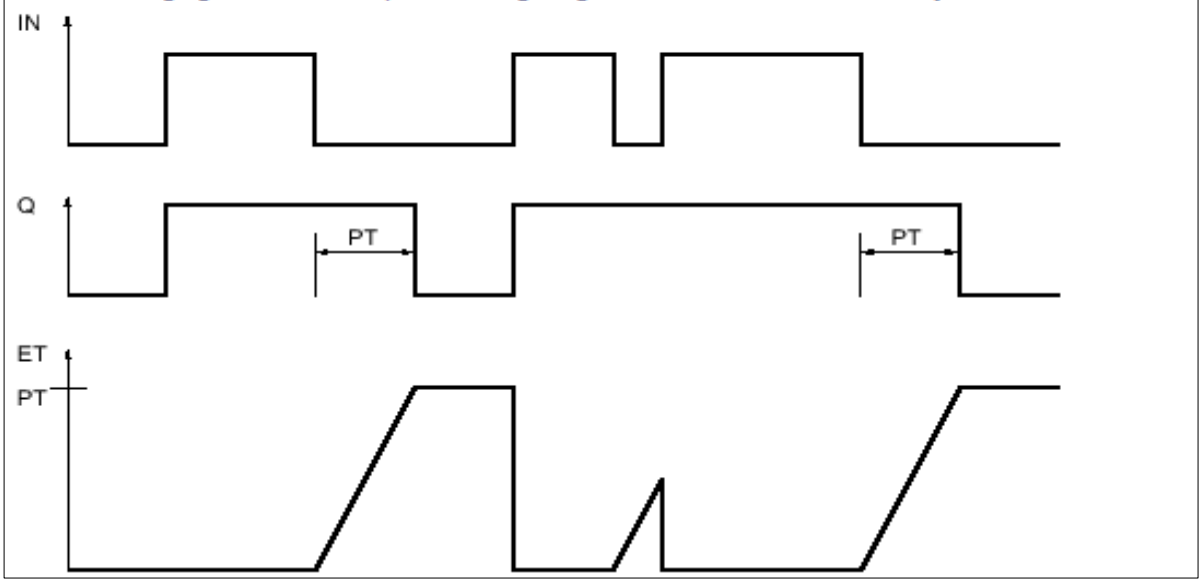
The current time value can be queried at the ET output. The time value starts at T#0s and ends when the value of duration PT is reached. When the time duration PT expires, the ET output remains set to the current value until the IN input changes back to "1". If input IN switches to "1" before the duration PT has expired, the ET output is reset to the value T#0s.

Parameters

Parameter	Declaration	Data type	Description
IN	Input	BOOL	Start input

Pulse timing diagram

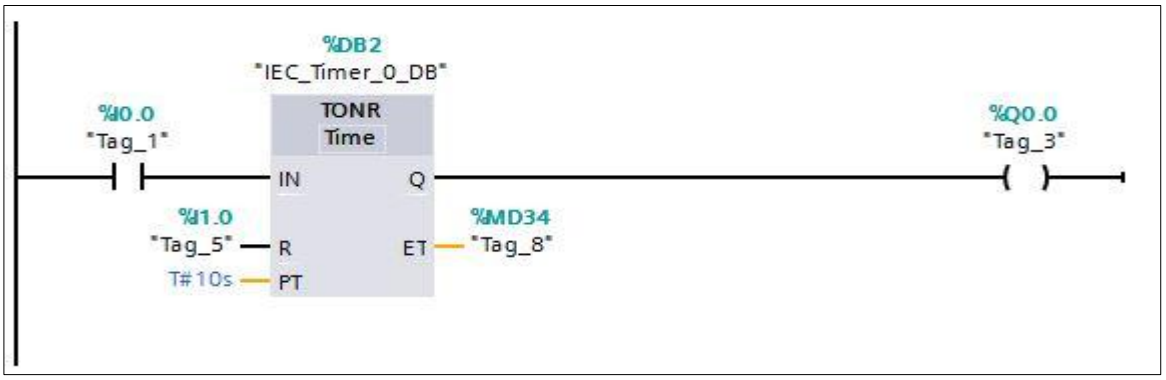
The following figure shows the pulse timing diagram of the "Generate off-delay" instruction:



PT	Input	TIME	Duration of the off delay. The value of the PT parameter must be positive.
Q	Output	BOOL	Output that is reset when the timer PT expires.
ET	Output	TIME	Current time value

4. TONR: Time accumulator

The "Time accumulator" instruction is used to accumulate time values within a period set by the PT parameter. When the signal state at input IN changes from "0" to "1" (positive signal edge), the instruction executes and the duration PT starts. While the duration PT is running, the



time values are accumulated that are recorded when the IN input has signal state "1". The accumulated time is written to output ET and can be queried there. When the duration PT expires, the output Q has the signal state "1". The Q parameter remains set to "1", even when the signal state at the IN parameter changes from "1" to "0" (negative signal edge). The R input resets the outputs ET and Q regardless of the signal state at the start input.

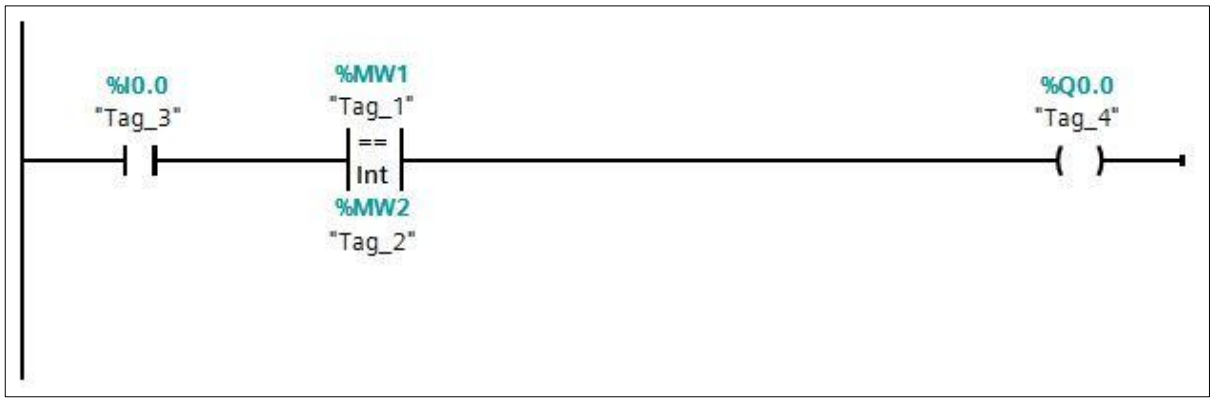
Parameters

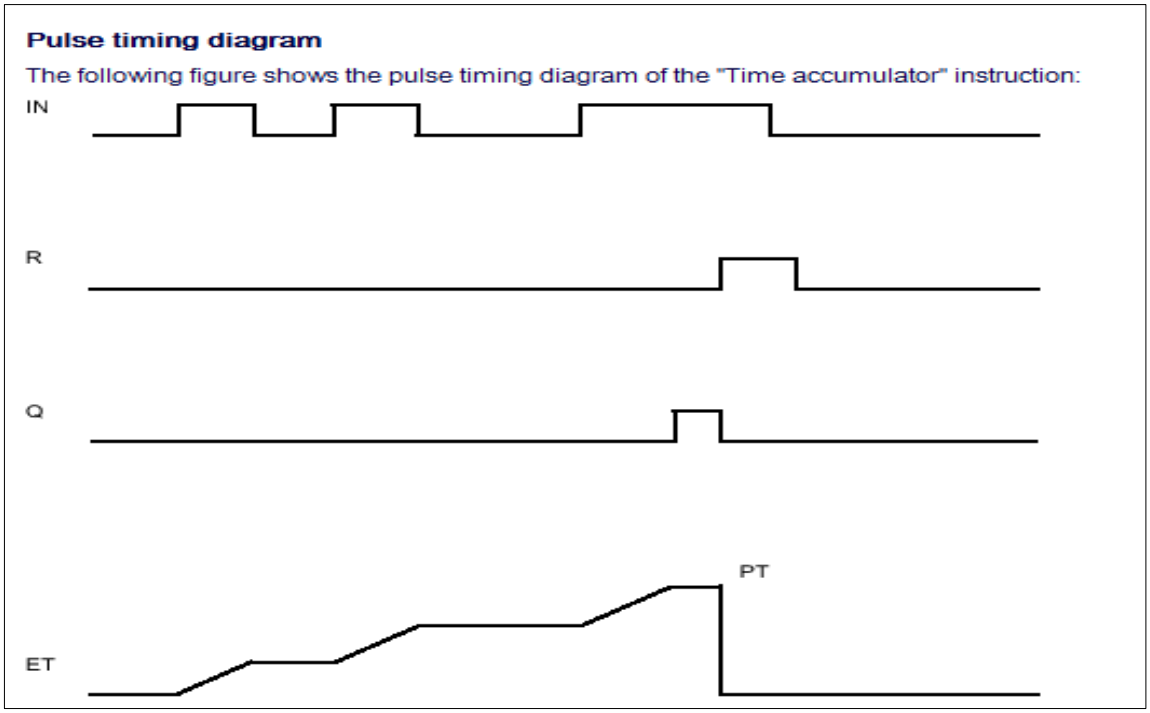
Parameter	Declaration	Data type	Description
IN	Input	BOOL	Start input
R	Input	BOOL	Reset input
PT	Input	TIME	Maximum duration of time recording. The value of the PT parameter must be positive.
Q	Output	BOOL	Output that is set when the time PT expires.
ET	Output	TIME	Accumulated time

Basic Mathematical Functions: Comparison Operations

1. CMP (==): Equal

You can use the "Equal" instruction to determine if a first comparison value (<Operand1>) is equal to a second comparison value (<Operand2>).





If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1". If the comparison condition is not fulfilled, the instruction returns RLO "0".

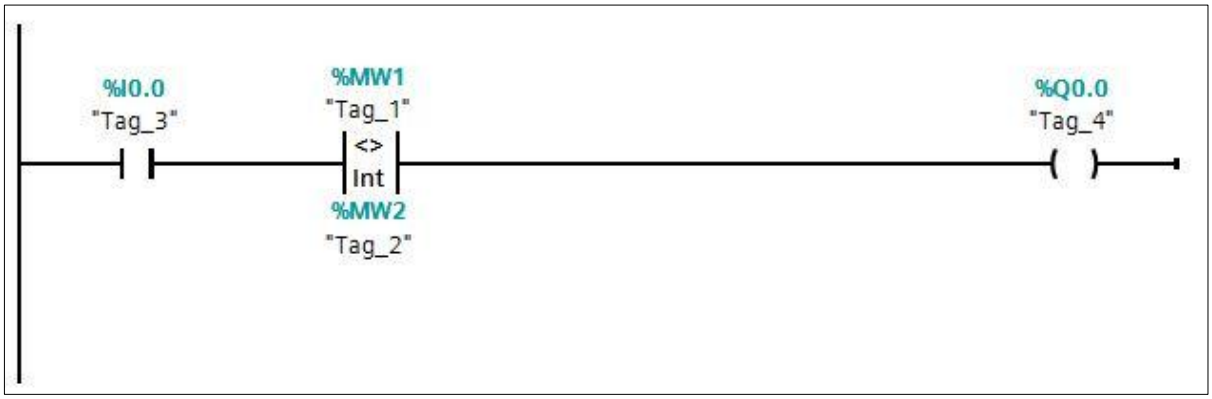
Parameters

Parameter	Declaration	Data type	Description
Operand 1	Input	integers, floating-point numbers, TIME, DATE	First comparison value
Operand 2	Input	integers, floating-point numbers, TIME, DATE	Second value to compare

2. CMP (<>): Not equal

You can use the "Not equal" instruction to determine if a first comparison value (<Operand1>) is not equal to a second comparison value (<Operand2>).

If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1". If the comparison condition is not fulfilled, the instruction returns RLO "0".

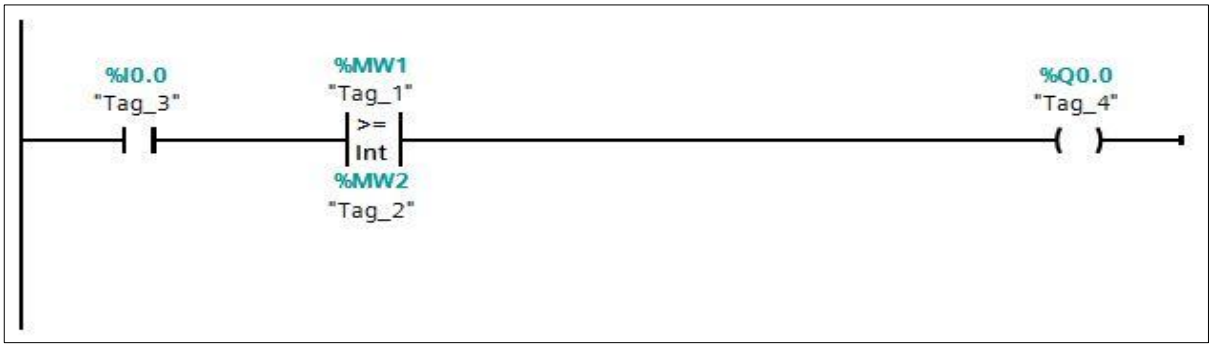


Parameters

Parameter	Declaration	Data type	Description
Operand 1	Input	integers, floating-point numbers, TIME, DATE	First comparison value
Operand 2	Input	integers, floating-point numbers, TIME, DATE	Second value to compare

3. CMP (>=): Greater or equal

You can use the "Greater or equal" instruction to determine if a first comparison value (<Operand1>) is greater than or equal to a second comparison value (<Operand2>). Both values to be compared must be of the same data type. If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1". If the comparison condition is not fulfilled, the instruction returns RLO "0".



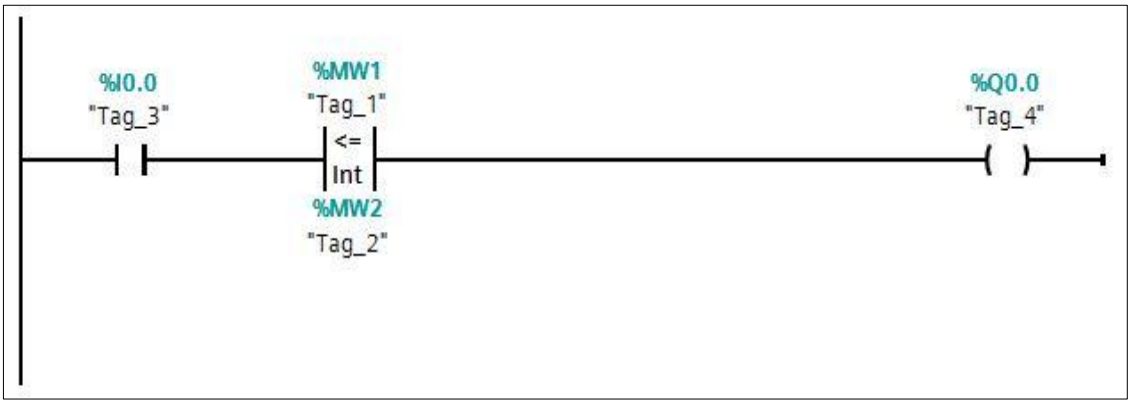
Parameters

Parameter	Declaration	Data type	Description
Operand 1	Input	integers, floating-point numbers, TIME, DATE	First comparison value
Operand 2	Input	integers, floating-point numbers, TIME, DATE	Second value to compare

4. CMP (<=): Less or equal

You can use the "Less or equal" instruction to determine if a first comparison value (<Operand1>) is less than or equal to a second comparison value (<Operand2>). Both values to be compared must be of the same data type.

If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) =1. If the comparison condition is not fulfilled, the instruction returns RLO=0.



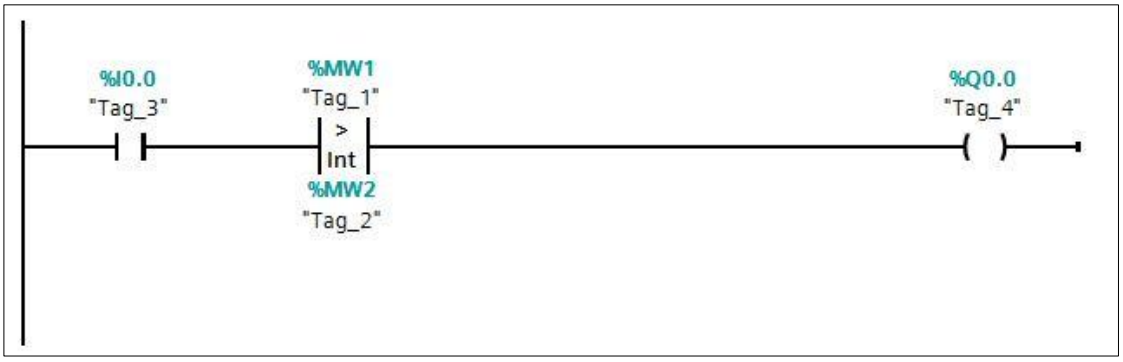
Parameters

Parameter	Declaration	Data type	Description
Operand 1	Input	integers, floating-point numbers, TIME, DATE	First comparison value
Operand 2	Input	integers, floating-point numbers, TIME, DATE	Second value to compare

5. CMP (>): Greater than

You can use the "Greater than" instruction to determine if a first comparison value (<Operand1>) is greater than a second comparison value (<Operand2>). Both values to be compared must be of the same data type.

If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1". If the comparison condition is not fulfilled, the instruction returns RLO "0".



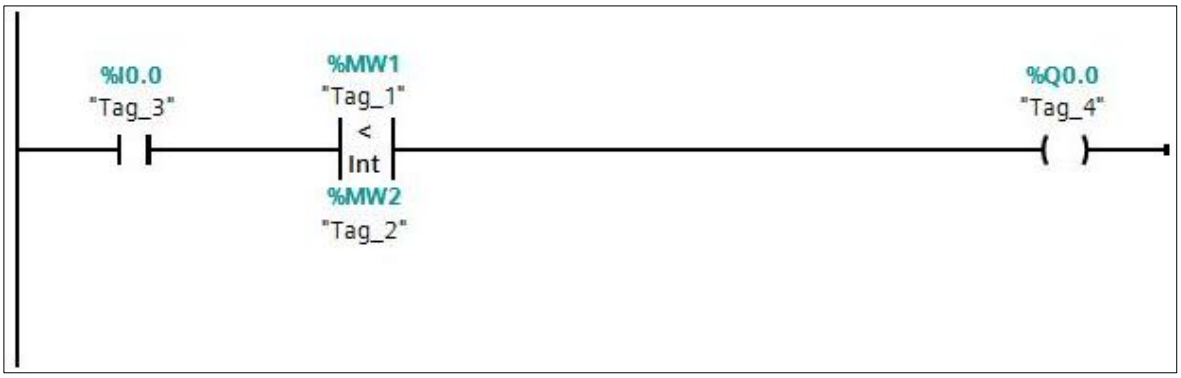
Parameters

Parameter	Declaration	Data type	Description
Operand 1	Input	integers, floating-point numbers, TIME, DATE	First comparison value
Operand 2	Input	integers, floating-point numbers, TIME, DATE	Second value to compare

6. CMP <: Less than

You can use the "Less than" instruction to determine if a first comparison value (<Operand1>) is less than a second comparison value (<Operand2>). Both values to be compared must be of the same data type.

If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1". If the comparison condition is not fulfilled, the instruction returns RLO "0".



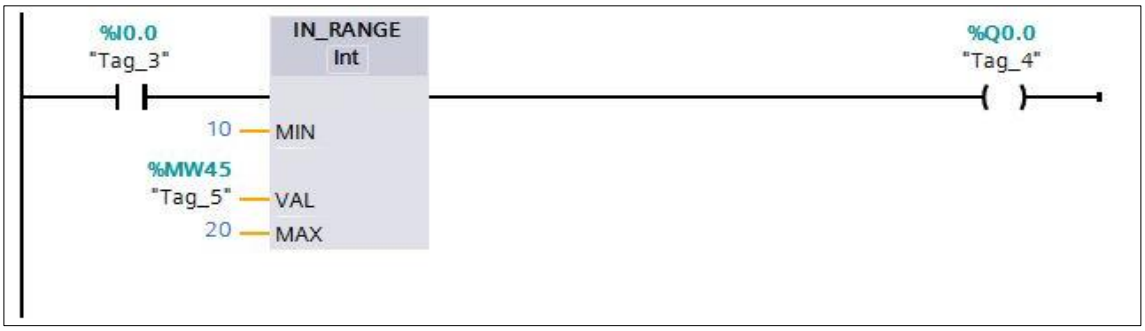
Parameters

Parameter	Declaration	Data type	Description
Operand 1	Input	integers, floating-point numbers, TIME, DATE	First comparison value
Operand 2	Input	integers, floating-point numbers, TIME, DATE	Second value to compare

7. IN_RANGE: Value within range

You can use the "Value within range" instruction to determine if the value at the VAL input is within a specific value range.

You specify the limits of the value range with the MIN and MAX inputs. The "Value within range" instruction compares the value at the VAL input with the values of the MIN and MAX inputs and sends the result to the box output. If the value at the VAL input fulfills the comparison MIN <= VAL or VAL <= MAX, the box output has the signal state "1". If the comparison is not fulfilled, the box output has the signal state "0".



If the box input has the signal state "0", the "Value within range" instruction is not executed. The comparison function can only execute if the values to be compared are of the same data type and the box input is interconnected.

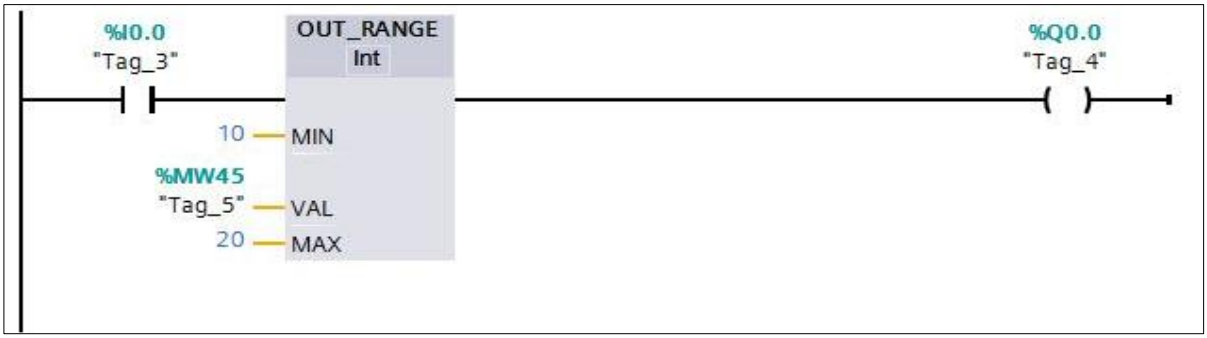
Parameters

Parameter	Declaration	Data type	Description
Input	Input	BOOL	Result of the previous logic operation
MIN	Input	Integers, floating-point numbers	Low limit of the value range
VAL	Input	Integers, floating-point numbers	Comparison value
MAX	Input	Integers, floating-point numbers	High limit of the value range
Output	Output	BOOL	Result of the comparison

8. OUT_RANGE: Value outside range

You can use the "Value outside range" instruction to determine if the value at the VAL input is outside a specific value range.

You specify the limits of the value range with the MIN and MAX inputs. The "Value outside range" instruction compares the value at the VAL input with the values of the MIN and MAX inputs and sends the result to the box output. If the value at the VAL input fulfills the comparison MIN > VAL or VAL > MAX, the box output has the signal state "1". The box output also has the signal state "1" if a specified operand with the REAL data type shows an invalid value.



The box output returns the signal state "0", if the value at input VAL does not satisfy the MIN > VAL or VAL > MAX condition.

If the box input has the signal state "0", the "Value outside range" instruction is not executed. The comparison function can only execute if the values to be compared are of the same data type and the box input is interconnected.

Parameters

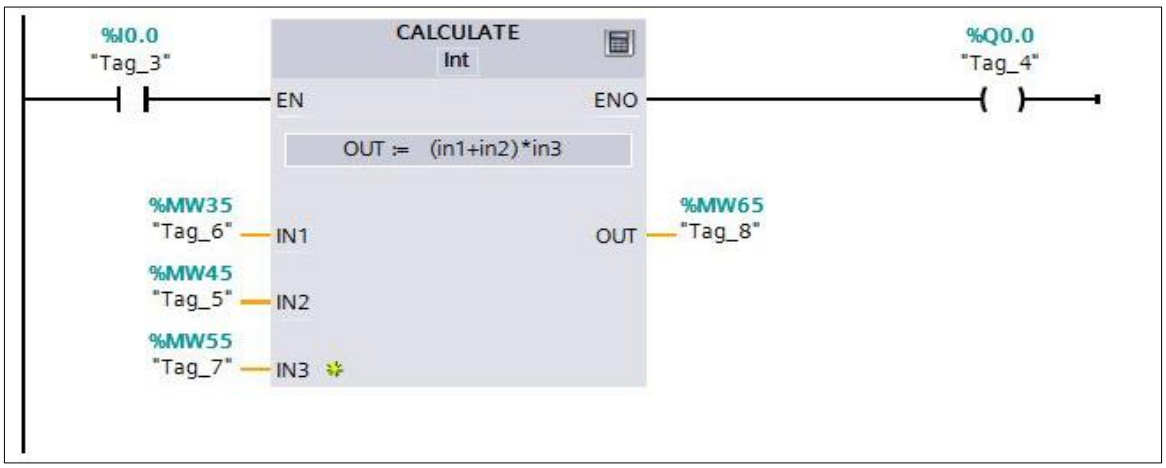
Parameter	Declaration	Data type	Description
Input	Input	BOOL	Result of the previous logic operation
MIN	Input	Integers, floating-point numbers	Low limit of the value range
VAL	Input	Integers, floating-point numbers	Comparison value
MAX	Input	Integers, floating-point numbers	High limit of the value range
Output	Output	BOOL	Result of the comparison

Basic Mathematical Functions:

1. *CALCULATE: Calculate*

The "Calculate" instruction is used to define and execute an expression for the calculation of mathematical operations or complex logic operations depending on the selected data type.

You can select the data type of the instruction from the "<???>" drop-down list of the instruction box. Depending on the data type selected, you can combine the functions of certain instructions to perform a complex calculation. The information for the expression to be calculated is entered in a dialog, which you can open with the icon at the upper right edge of the instruction box. The expression can contain names of input parameters and the syntax of the instructions. Operand names and operand addresses cannot be specified.



Parameters

Parameter	Declaration	Data type	Description
EN	Input	BOOL	Enable input
ENO	Output	BOOL	Enable Output
IN1	Input	integers, floating-point numbers	First available input
IN2	Input	Integers, floating-point numbers	Second available input
OUT	Output	Integers, floating-point numbers	Output to which the end result is to be transferred

2. ADD: Addition

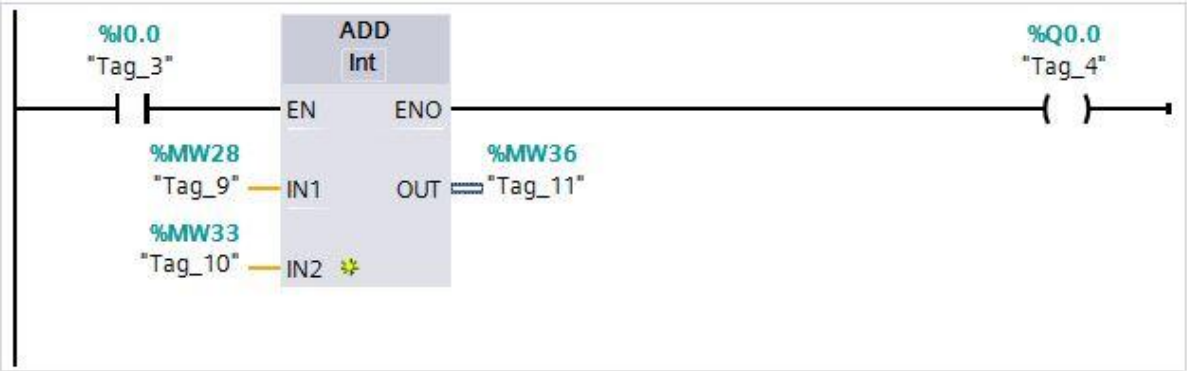
You can use the "Add" instruction to add the value at input IN1 and the value at input IN2 and query the sum at output OUT (OUT := IN1+IN2).

In its initial state, the instruction box contains at least 2 inputs (IN1 and IN2). The number of inputs can be extended. The inserted inputs are numbered in ascending order in the box. When the instruction is executed, the values of all available input parameters are added. The sum is stored at the OUT output.

The ENO enable output has the signal state "0" if one of the following conditions is fulfilled:

The enable input EN has the signal state "0".

The result of the instruction is outside the range permitted for the data type specified at the OUT output.



A floating-point number has an invalid value.

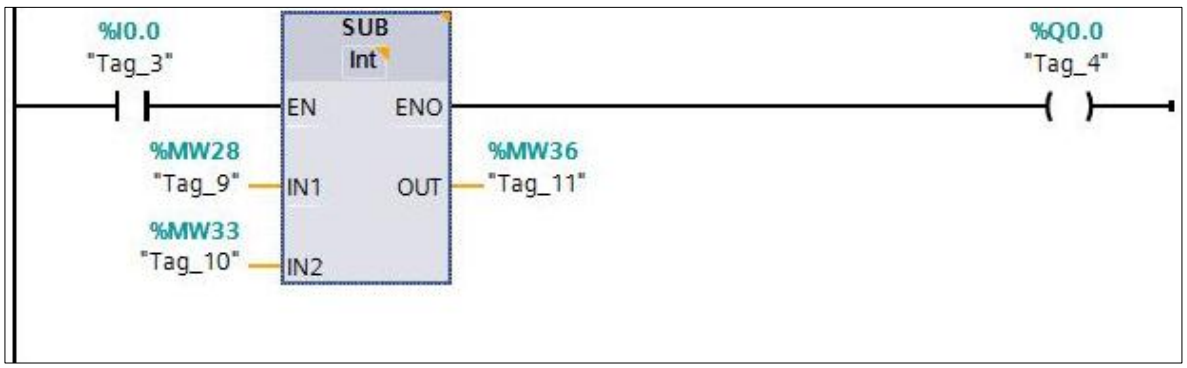
Parameters

Parameter	Declaration	Data type	Description
EN	Input	BOOL	Enable input
ENO	Output	BOOL	Enable Output
IN1	Input	integers, floating numbers	First number to be added
IN2	Input	Integers, floating numbers	Second number to be added
OUT	Output	Integers, floating numbers	Sum

3. SUB: Subtract

You can use the "Subtract" instruction to subtract the value at input IN2 from the value at input IN1 and query the difference at output OUT (OUT := IN1-IN2).

The ENO enable output has the signal state "0" if one of the following conditions is fulfilled:



The EN enable input has the signal state "0".

The result of the instruction is outside the range permitted for the data type specified at the OUT output.

A floating-point number has an invalid value.

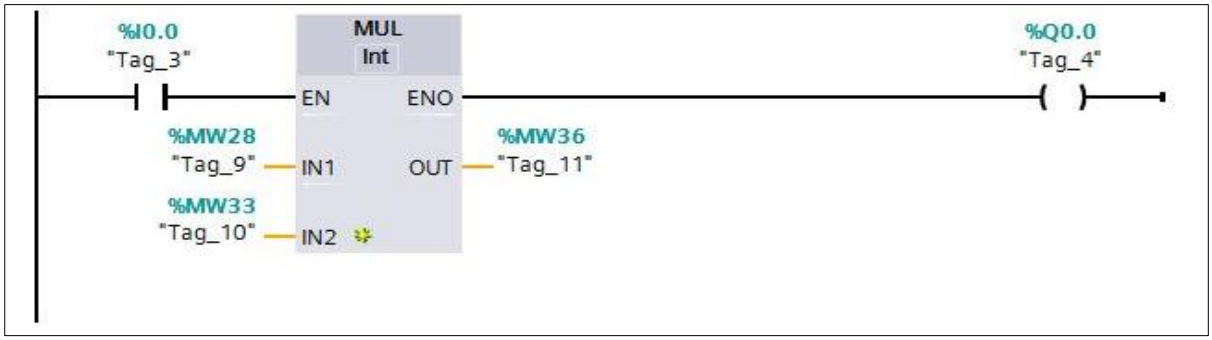
Parameter:

Parameter	Declaration	Data type	Description
EN	Input	BOOL	Enable input
ENO	Output	BOOL	Enable Output
IN1	Input	integers, floating-point numbers	Minuend
IN2	Input	Integers, floating-point numbers	Subtracting
OUT	Output	Integers, floating-point numbers	Difference

4. MUL: Multiply

You can use the "Multiply" instruction to multiply the value at input IN1 with the value at input IN2 and query the product at output OUT (OUT := IN1*IN2).

The number of inputs can be expanded in the instruction box. The added inputs are numbered in ascending order in the box. When the instruction is executed, the values of all available input parameters are multiplied. The product is stored at the OUT output.



The ENO enable output has the signal state "0" if one of the following conditions is fulfilled:

The EN input has the signal state "0". The result is outside the range permitted for the data type specified at output OUT. A floating-point number has an invalid value.

Parameters:

Parameter	Declaration	Data type	Description
EN	Input	BOOL	Enable input
ENO	Output	BOOL	Enable Output
IN1	Input	integers, floating-point numbers	Multiplier
IN2	Input	Integers, floating-point numbers	Number being multiplied
OUT	Output	Integers, floating-point numbers	Product

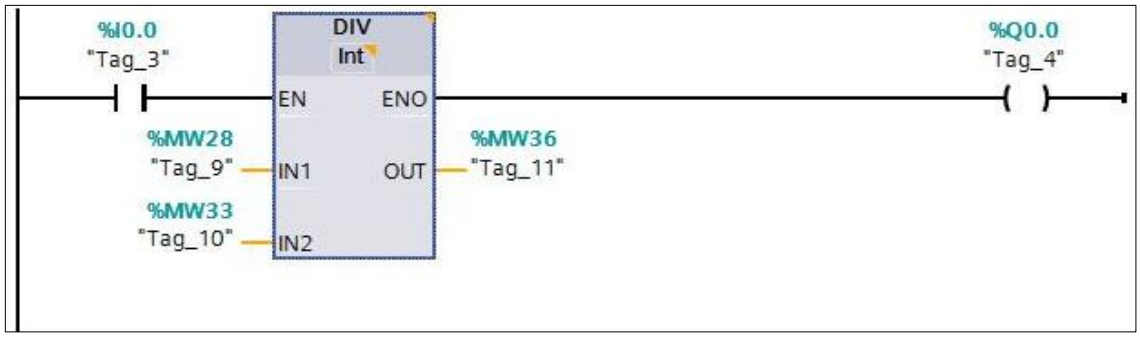
5. DIV: Divide

You can use the "Divide" instruction to divide the value at input IN1 by the value at input IN2 and query the quotient at output OUT (OUT := IN1/IN2).

The ENO enable output has the signal state "0" if one of the following conditions is fulfilled:

The EN enable input has the signal state "0".

The result of the instruction is outside the range permitted for the data type specified at the OUT output.



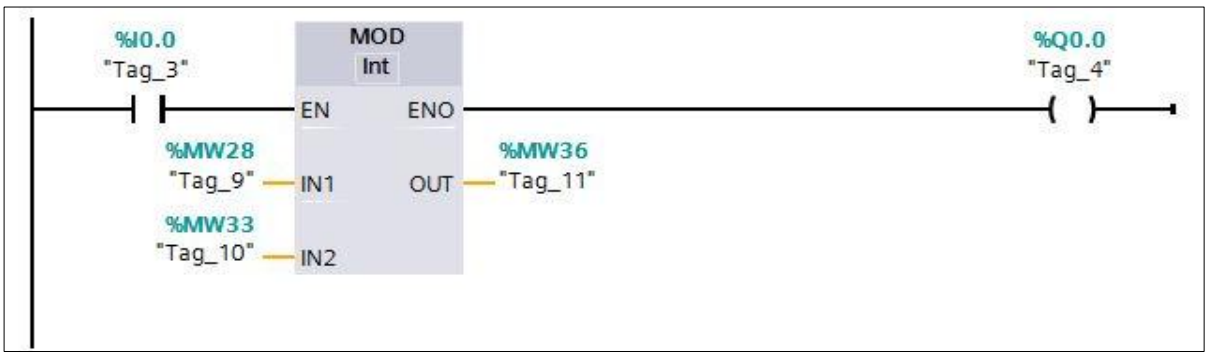
A floating-point number has an invalid value.

Parameters:

Parameter	Declaration	Data type	Description
EN	Input	BOOL	Enable input
ENO	Output	BOOL	Enable Output
IN1	Input	integers, floating-point numbers	Dividend
IN2	Input	Integers, floating-point numbers	Divisor
OUT	Output	Integers, floating-point numbers	Quotient value

6. MOD: Return remainder of division

You can use the "Return remainder of division" instruction to divide the value at input IN1 by the value at input IN2 and query the remainder of division at output OUT.

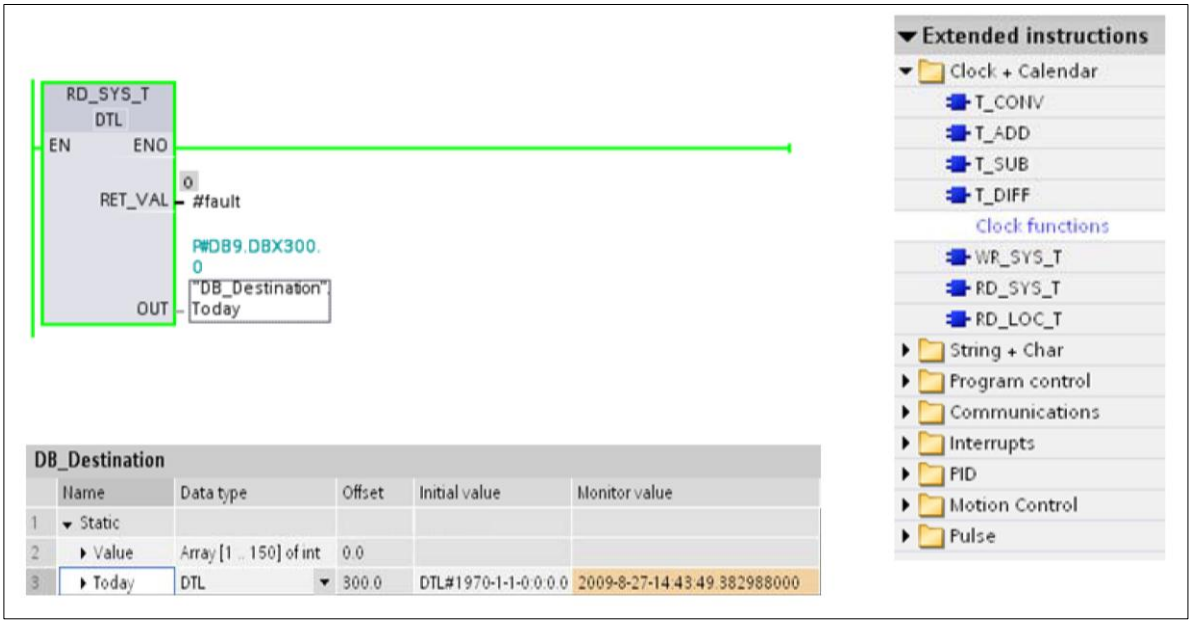


Parameters:

Parameter	Declaration	Data type	Description
EN	Input	BOOL	Enable input
ENO	Output	BOOL	Enable Output
IN1	Input	Integers	Dividend

IN2	Input	Integers	Divisor
OUT	Output	Integers	Remainder of division

Date and Time of day: RD_SYS_T



RD_SYS_T

You can use RD_SYS_T to read the current date and current time of the CPU clock. The data is provided in DTL format at the OUT output of the instruction. The provided value does not include information about the local time zone or daylight saving time. At the RET_VAL output, you can query whether errors have occurred during execution of the instruction.

DTL:

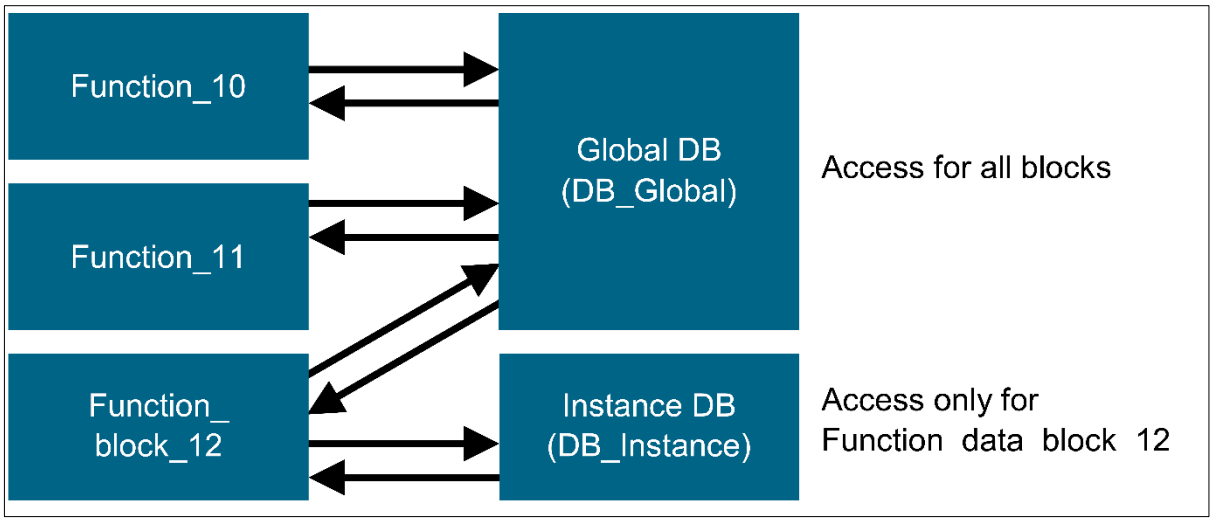
Byte	Component	Data type	Value range
0	Year	UINT	1970 to 2664
1			
2	Month	USINT	0 to 12
3	Day	USINT	1 to 31
4	Day of week	USINT	1(Sunday) to 7 (Saturday) The weekday is not considered in the value entry.
5	Hour	USINT	0 to 23
6	Minute	USINT	0 to 59
7	Second	USINT	0 to 59
8	Nanoseconds	UDINT	0 to 999 999 999
9			
10			
11			

Data Blocks

Data Blocks

In contrast to logic blocks, data blocks contain no instructions. Rather, they serve as memory for user data.

Data blocks thus contain variable data that is used by the user program. You can define the structure of global data blocks as required.



Global data blocks store data that can be used ***by all other blocks*** (see Figure 1). Only the associated function block should access instance data blocks. The maximum size of data blocks varies depending on the utilized CPU.

Application examples for **global data blocks** are:

- Saving of information about a storage system. "Which product is located where?"
- Saving of recipes for particular products.

The data in data blocks is stored retentively in most cases. This data is then retained in the event of a power failure or after a STOP/START of the CPU.

Overview of Data Types in STEP 7

Variables are used to store data. The data type of a variable specifies its memory requirements, its value range and the representation of the variable value in the Editor. As well, the possible operations with which a variable can be processed can be obtained from the data type.

Step 7 has differentiated data types according to the use and area of field as listed below:

1. Elementary data type
2. Complex data type
3. Parameter types
4. System data types
5. Hardware data types

Elementary Data Types in STEP 7

The SIMATIC S7-1200 has many different data types for representing different numerical formats. A list of some of the elementary data types is given below.

Data type	Size (bits)	Range	Example of constant entry
Bool	1	0 to 1	TRUE, FALSE, O, 1
Byte	8	16#00 to 16#FF	16#12, 16#AB
Word	16	16#0000 to 16#FFFF	16#ABCD, 16#0001
DWord	32	16#00000000 to 16#FFFFFFFF	16#02468ACE
Char	8	16#00 to 16#FF	'A', 'r', '@'
Sint	8	-128 to 127	123,-123
Int	16	-32,768 to 32,767	123, -123
Dint	32	-2,147,483,648 to 2,147,483,647	123, -123
USInt	8	0 to 255	123
UInt	16	0 to 65,535	123
UDInt	32	0 to 4,294,967,295	123
Real	32	+/-1.18 x 10 -38 to +/-3.40 x 10 38	123.456, -3.4, 1.2E+12 3.4E-3
LReal	64	+/-2.23 x 10 -308 to +/-1.79 x 10 308	12345.123456789 -1.2E+40
Time	32	T#-24d_20h_31 m_23s_648ms to T#24d_20h_31 m_23s_647ms Saved as: -2,147,483,648 ms to +2,147,483,647 ms	T#5m_30s 5#-2d T#1d_2h_15m_30x_45ms
String	Variable	0 to 254 characters in byte size	'ABC'
Array		With arrays, data of a uniform data type is arranged one after the other and addressed consecutively in the address area. The properties of each array element are identical and are configured in the array tag.	
Struct		The STRUCT data type represents a data structure that consists of a fixed number of components of different data types. Components of STRUCT or ARRAY data type can also be nested in a structure.	
…		For other data types, refer to the online help.	

Data Types for Timers, Date and Time-of-day

Timers

In S7-1200, only one data type for timer is available. Details is given below.

TIME (IEC time)

The contents of an operand of the data type TIME is interpreted as milliseconds. The representation contains information for days (d), hours (h), minutes (m), seconds (s) and milliseconds (ms).

The following table shows the properties of data type TIME:

Length (bits)	Format	Value range	Examples of value input
32	Signed duration	T#-24d20h31m23s648ms to T#+24d20h31m23s647ms	T#10d20h30m20s630ms, TIME#10d20h30m20s630ms

	Hexadecimal numbers	16#00000000 to 16#7FFFFFFF	16#0001EB5E

It is not necessary to specify all time units. T#5h10s is a valid entry, for example. If only one unit is specified, the absolute value of days, hours, and minutes must not exceed the high or low limits. When more than one time unit is specified, the value must not exceed 24 days, 23 hours, 59 minutes, 59 seconds or 999 milliseconds.

Date

The DATE data type saves the date as an unsigned integer. The representation contains the year, the month, and the day.

The contents of an operand of DATE data type correspond in hexadecimal format to the number of days since 01-01-1990 (16#0000).

The following table shows the properties of data type DATE:

Length (bytes)	**Format**	**Range of values**	**Examples of value input**
2	IEC date (Year-Month-Day)	D#1990-01-01 to D#2168-12-31	D#2009-12-31, DATE#2009-12-31
	Hexadecimal numbers	16#0000 to 16#FF62	16#00F2

Time of Day

Data type TOD (TIME_OF_DAY) occupies a double word and stores the number of milliseconds since the beginning of the day (0:00 h) as unsigned integer.

The following table shows the properties of data type TOD:

Length (bytes)	**Format**	**Value range**	**Examples of value input**
4	Time-of-day (hours : minutes : seconds . milliseconds)	TOD#00:00:00.000 to TOD#23:59:59.999	TOD#10:20:30.400, TIME_OF_DAY#10:20:30.400

You always need to specify the hours, minutes and seconds. The specification of milliseconds is optional.

Complex Data Types

Complex data types define data groups that are larger than 32 bits or data groups consisting of other data types. STEP 7 permits the following complex data types:

- DATE_AND_TIME

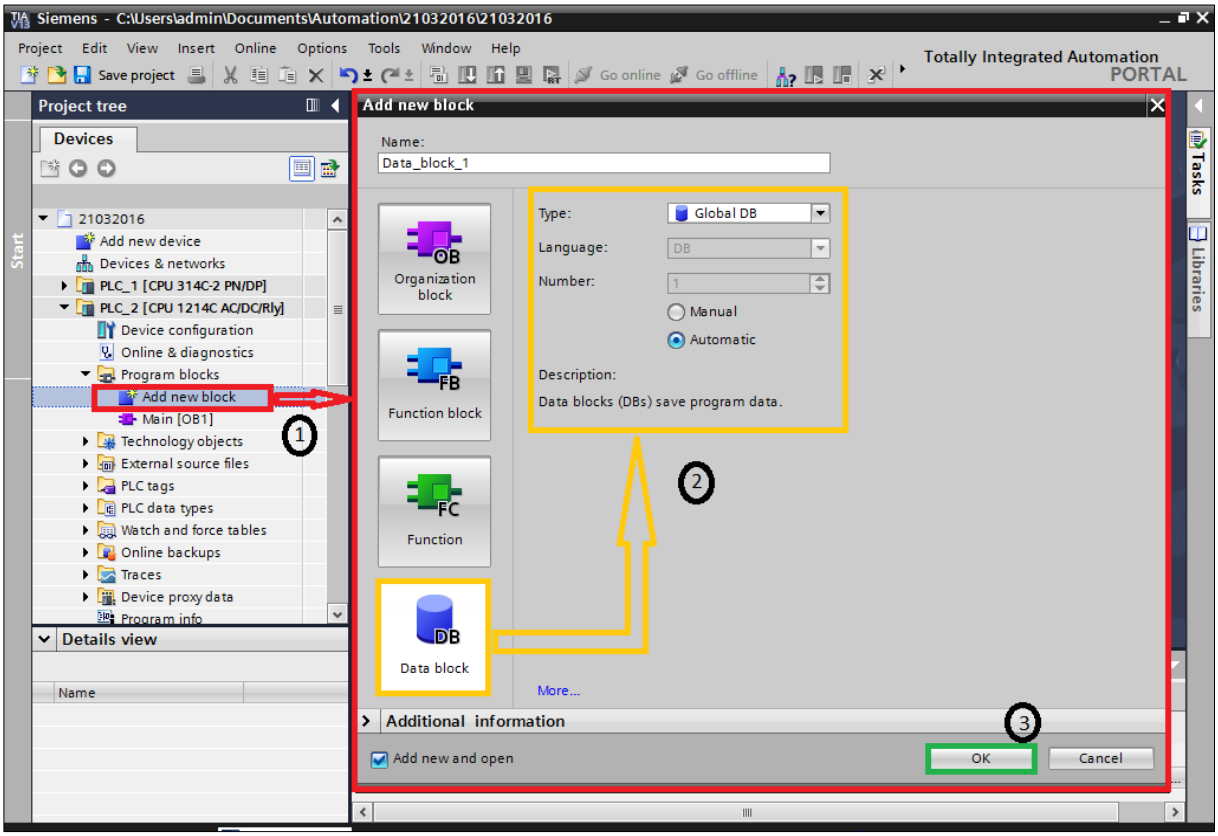
- STRING
- ARRAY
- STRUCT
- UDT (user-defined data types)
- FBs and SFBs

The following table describes the complex data types. You define structures and arrays either in the variable declaration of the logic block or in a data block.

Data Type	**Description**
DATE_AND_TIME DT	Defines an area with 64 bits (8 bytes). This data type saves in binary coded decimal format:
STRING	Defines a group with a maximum of 254 characters (data type CHAR). The standard area reserved for a character string is 256 bytes long. This is the space required to save 254 characters and a header of 2 bytes. You can reduce the memory required for a string by defining the number of characters that will be stored in the character string (for example: string[9] 'Siemens').
ARRAY	Defines a multidimensional grouping of one data type (either elementary or complex). For example: "ARRAY [1..2,1..3] OF INT" defines an array in the format 2 x 3 consisting of integers. You access the data stored in an array using the Index ("[2,2]"). You can define up to a maximum of 6 dimensions in one array. The index can be any integer (-32768 to 32767).
STRUCT	Defines a grouping of any combination of data types. You can, for example, define an array of structures or a structure of structures and arrays.
UDT	Simplifies the structuring of large quantities of data and entering data types when creating data blocks or declaring variables in the variable declaration. In STEP 7, you can combine complex and elementary data types to create your own "user defined" data type. UDTs have their own name and can therefore be used more than once.
FB, SFB	You determine the structure of the assigned instance data block and allow the transfer of instance data for several FB calls in one instance DB.

Structured data types are saved in accordance with word limits (WORD aligned).

Creating a Global Data Block



DB Attributes: "Optimized Block Access" and "Only Store in Load Memory"

Data blocks with optimized access

Data blocks with optimized access have no fixed defined structure. In the declaration, the data elements are assigned only a symbolic name and no fixed address within the block. The elements are saved automatically in the available memory area of the block so that there are no gaps in the memory. This makes for optimal use of the memory capacity.

Tags are identified by their symbolic names in these data blocks. To address the tag, enter its symbolic name. For example, you access the "Fill Level" tag in the "Data" DB as follows:

"Data".Fill Level

Blocks with optimized access offers the following advantages:

- You can create data blocks with any structure without paying attention to the physical arrangement of the individual data elements.
- Quick access to the optimized data is always available because the data storage is optimized and managed by the system.

- Access errors, as with indirect addressing or from the HMI, for example, are not possible.
- You can define specific individual tags as retentive.
- Optimized blocks are equipped with a memory reserve by default which lets you expand the interfaces of function blocks or data blocks during operation. You can download the modified blocks without setting the CPU to STOP and without affecting the values of already loaded tags.

Note

The "Optimized block access" attribute is always enabled for the following blocks and cannot be deselected.

- GRAPH blocks
- ARRAY data blocks

Only Store in Load Memory

Data blocks with the "Only store in load memory" attribute (unlinked DBs) are data blocks that do not occupy any space in the main memory of the controller and only use space in the load memory of the CPU. They are therefore suitable for storing large volumes of data that are not needed often in the controller's program. You need the following to be able to use these DBs for the S7-1200/S7-1500:

- Firmware 2.0 (or higher) and STEP 7 (TIA Portal) V11+SP2 (or higher) for the S7-1200.
- STEP 7 (TIA Portal) V12 (or higher) for the S7-1500.

Example:

Since the main memory has only a limited size, for recipe management, for example, multiple data blocks with different recipe values can only be stored in the load memory. Then, in the main memory there is only a working DB that contains the current recipe. If the DBs mentioned above are configured and loaded into the CPU, then those data blocks are only available in the CPU's load memory and hence do not take up any space in the main memory.

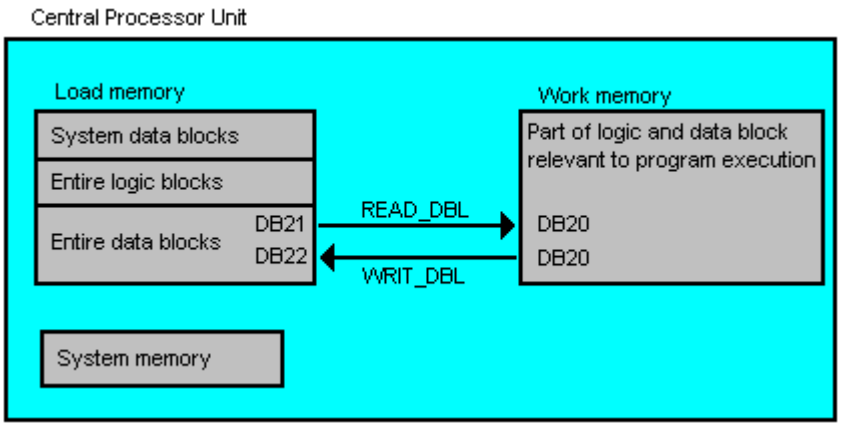


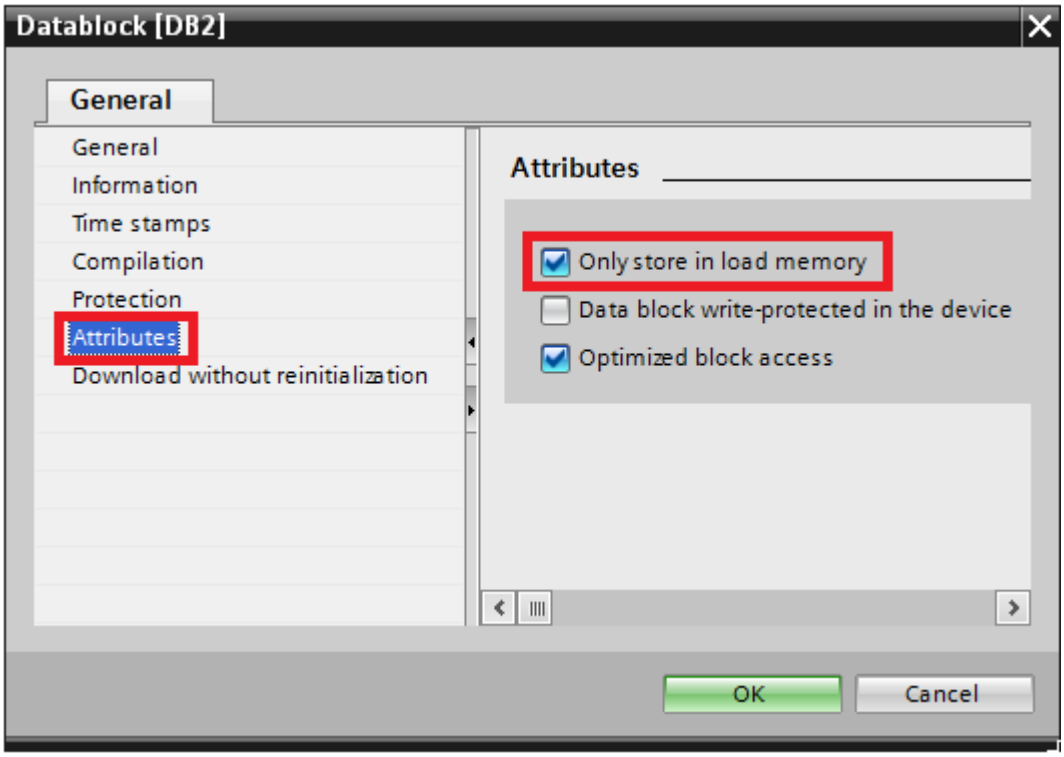
Figure shows an example of data transfer between the unlinked DBs in the load memory and the sequence-relevant DB in the main memory. Use the two instructions below to program data transfer:

- READ_DBL: read from the data block in the load memory
- WRIT_DBL: write to the data block in the load memory

Attached is a sample program for an S7-1500 with the WRITE-READ function. The DB in the load memory was create with the CREATE_DB function.

Below we explain how to proceed to generate an unlinked DB.

1. Open the project navigation and create a global data block.
2. Right-click the newly created global data block and select the "Properties..." item in the pop-up menu.
3. Select the "Attributes" tab and enable the "Only store in load memory" attribute.



4. Confirm with OK.

Note:

For further applications it is useful to store the unlinked DB created in the global library. In this way you can use the element for other projects too. Avoid periodic writing to the load memory, because the number of write cycles and thus the service life of the load memory is limited.

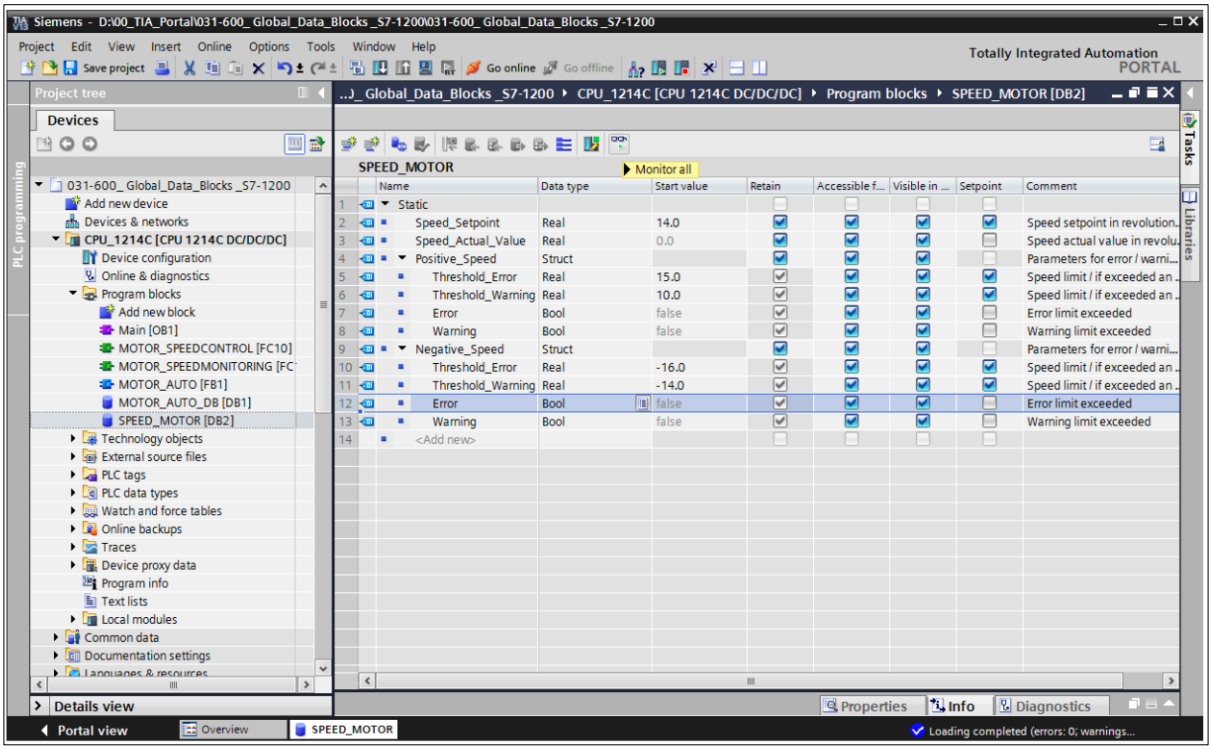
Further information is available in the STEP 7 (TIA Portal) Online Help under

- READ_DBL: Read from the data block in the load memory

- WRIT_DBL: Write to the data block in the load memory

Editing, Saving, Monitoring a Data Block

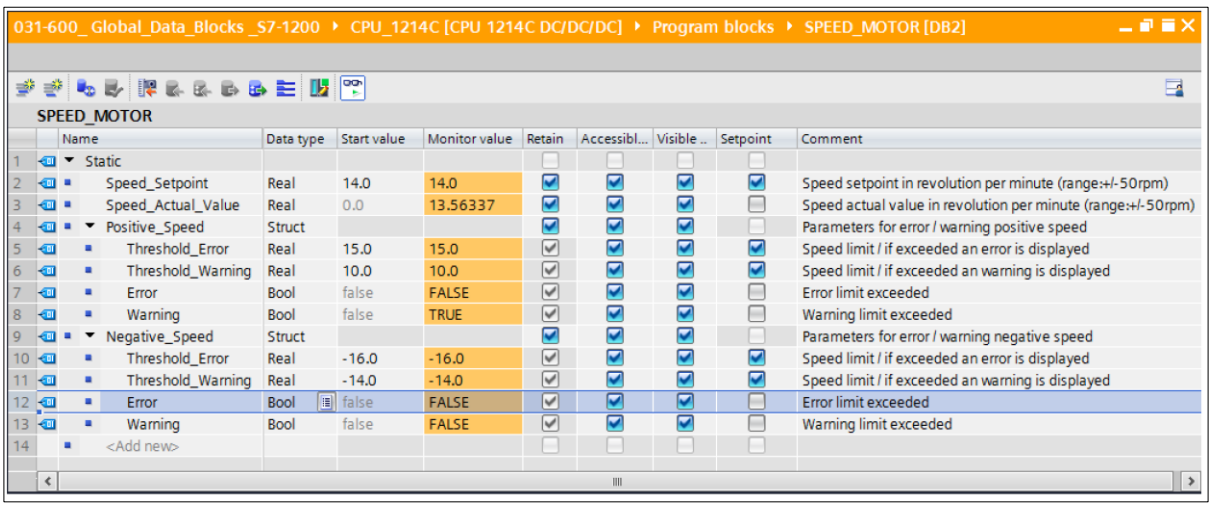
The desired block must be open for monitoring the tags of a downloaded data block. The monitoring can then be activated/deactivated by clicking the icon.



Default, Start and Monitoring Values

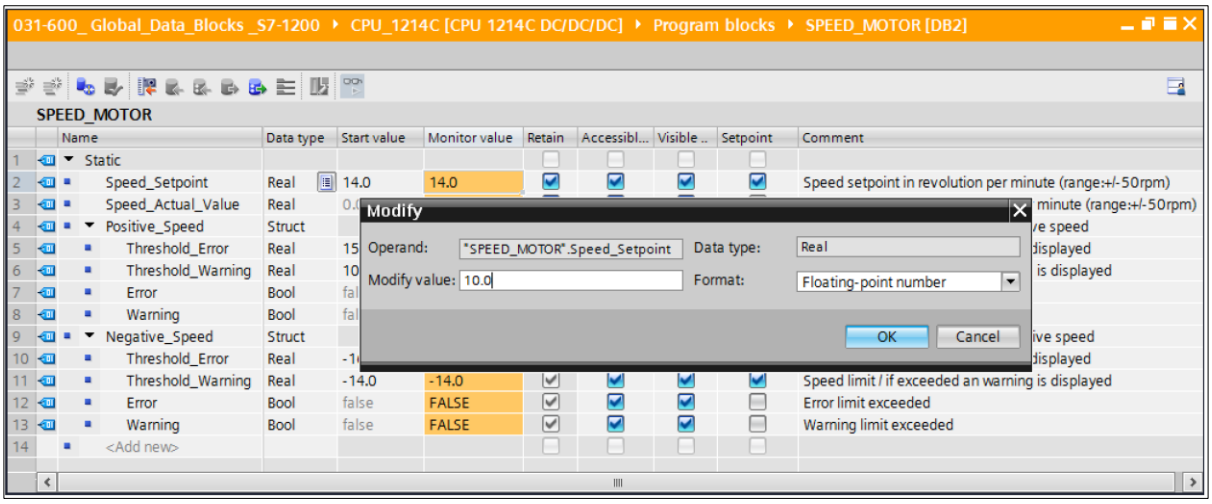
Monitor/modify values in data blocks

The desired block must be open for monitoring the tags of a downloaded data block. The monitoring can then be activated/deactivated by clicking the icon.



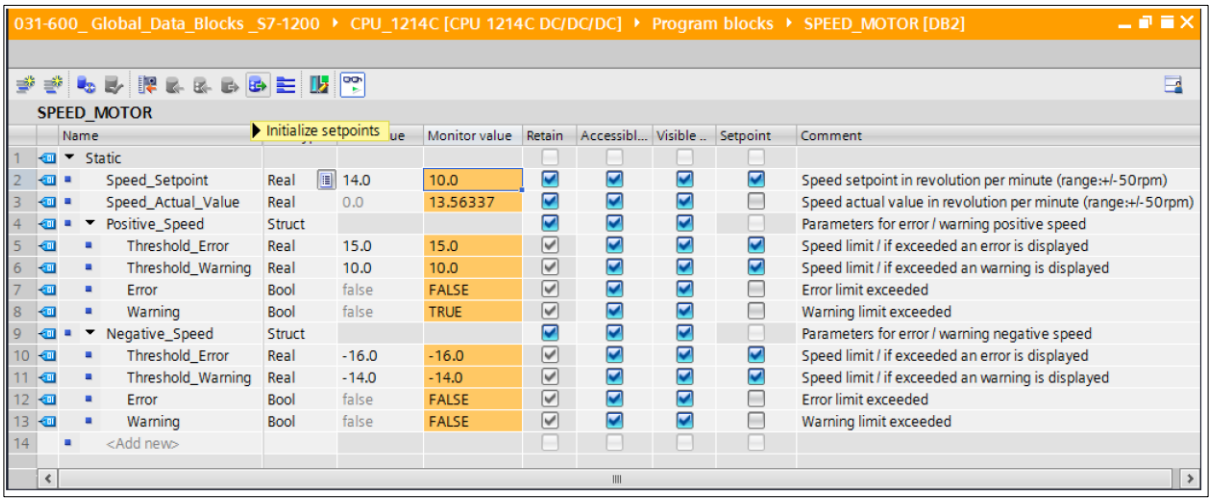
In the 'Monitor value' column, the values currently available in the CPU can be monitored.

If your right-click on one of the values, the 'Modify' dialog for modifying this value opens

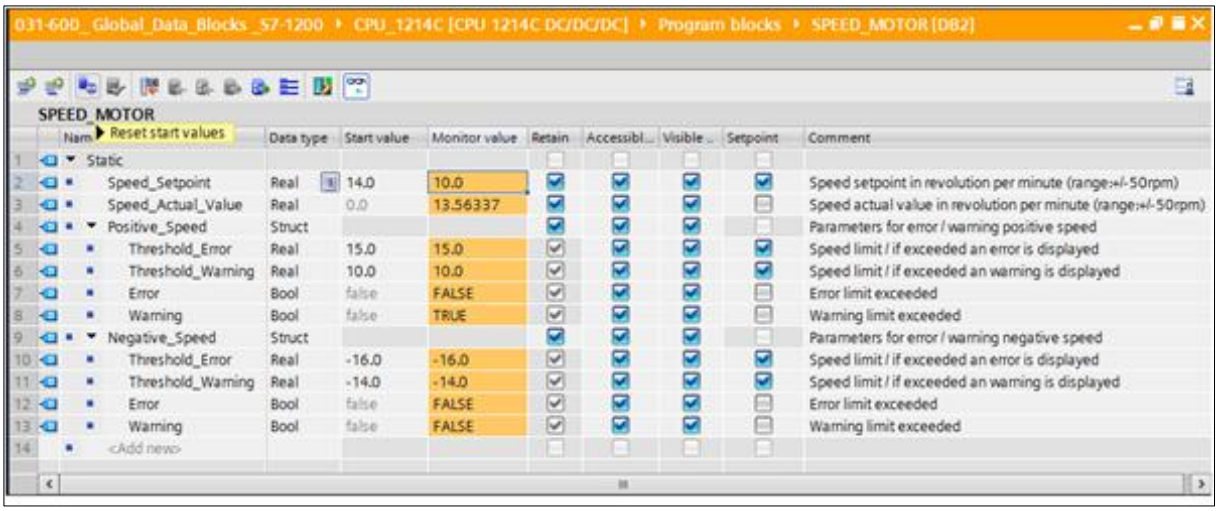


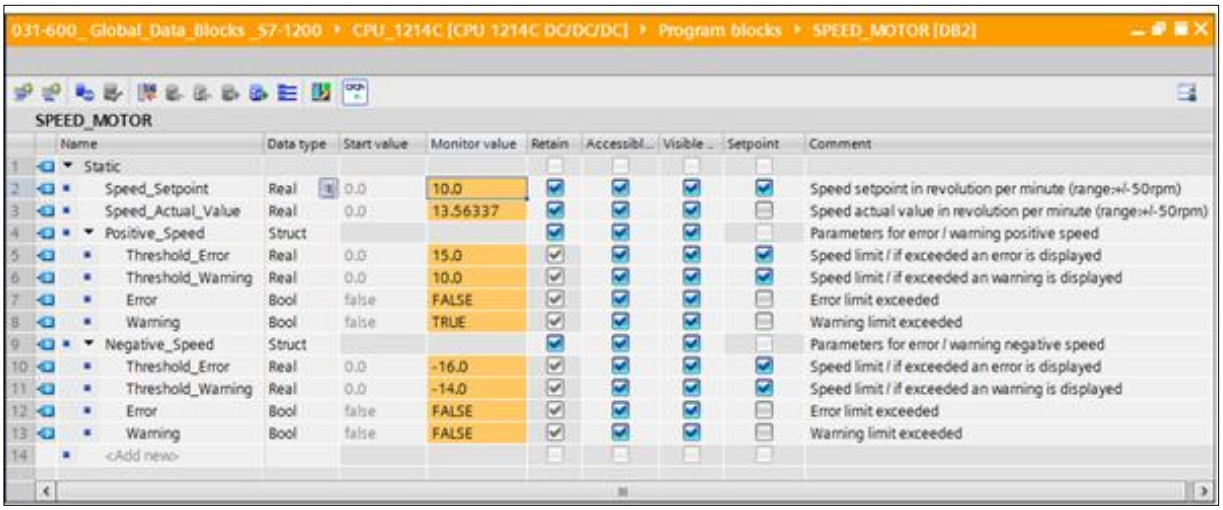
Initialize set points/reset start values

The set points can be initialized by clicking the "  " icon. For the tags whose 'Set-point' check box is selected, the start value will then be applied as the current value.



All start values can be reset by clicking the  icon.





Retentiveness, Download DB into the CPU / Upload from the CPU

To enable 'Download without re-initialization' for the "SPEED_MOTOR" [DB2] data block, you must go offline " and then open the properties of the data block.

Select the “Optimized block access” check box in the properties under 'General', 'Attributes'. (General -> Attributes -> Optimized block access)

Assign a 'Retentive memory reserve' to the data block for 'Download without re-initialization'. (Download without re-initialization -> Retentive memory reserve -> 10 bytes -> OK)

Downloading Changed Data Blocks into the CPU

After successful compilation, the complete controller with the created program including the hardware configuration can, as described in the previous modules, be downloaded. ( )

Function And Function Blocks

Local and Global Variables

Local Variables	Global Variables
• Valid only in ONE block • It is temporary and static variables • It is overwritten after the associated block is executed • Usable in OBs / FCs / FBs • It is accessible as Absolute and Symbolic • Static variables are retained even after the block is executed	• It is valid in the entire program • PII / PIQ • I/O Peripherals • Bit memories • Variables in DBs • Constants • It is accessible as Absolute and Symbolic

Local data stack

When you create organization blocks, you can declare temporary tags (TEMP) that are only available when the block is executed and are then overwritten again. Each organization block also requires 20 bytes of local data for its start information.

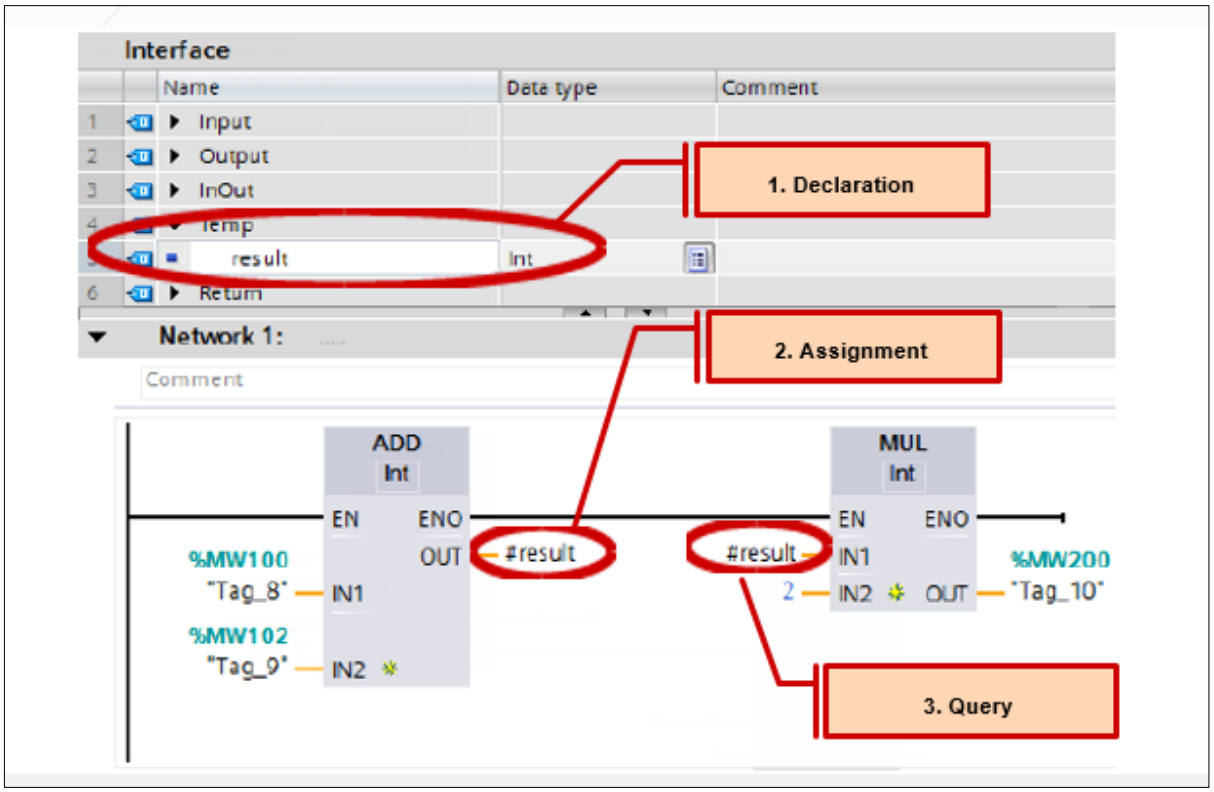
The CPU has a limited amount of memory for the temporary tags of the local data of blocks currently being executed. The size of this local memory, called the local data stack or L stack, depends on the particular CPU. By default, the local data stack is divided equally among the priority classes. This means that each priority class has its own area in the local data stack. This ensures that high-priority classes and their OBs have space available for their local data.

Before the local data stack is accessed for the first time, the local data must be initialized.

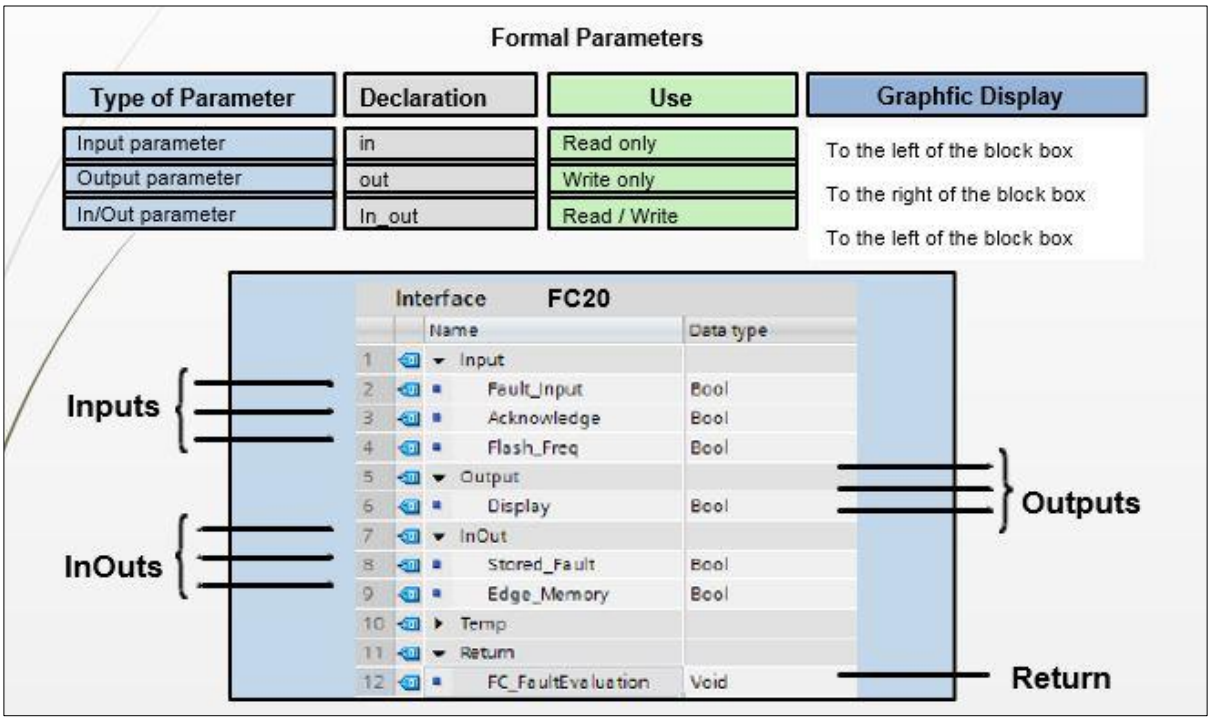
The local data stack stores the following data:

- Temporary tags of the local data of blocks
- Start information of the organization blocks
- Information regarding the transfer of parameters
- Intermediate results of the logic in ladder logic programs

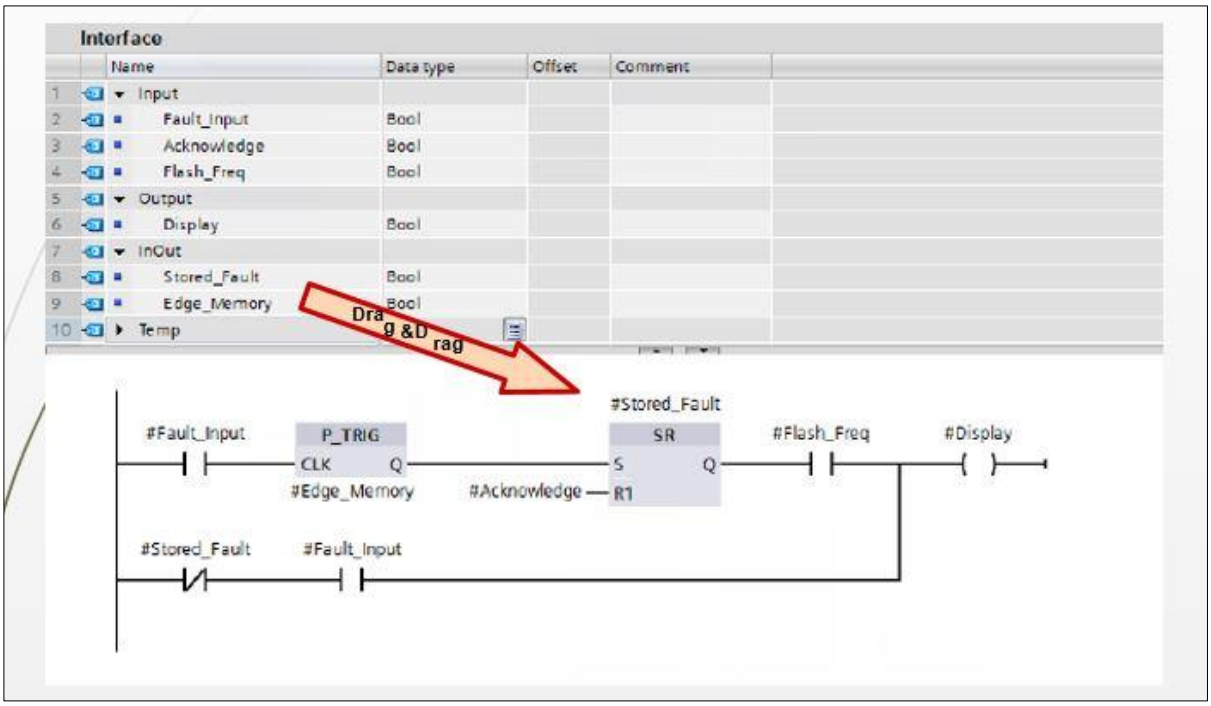
Declaration of Temporary variables



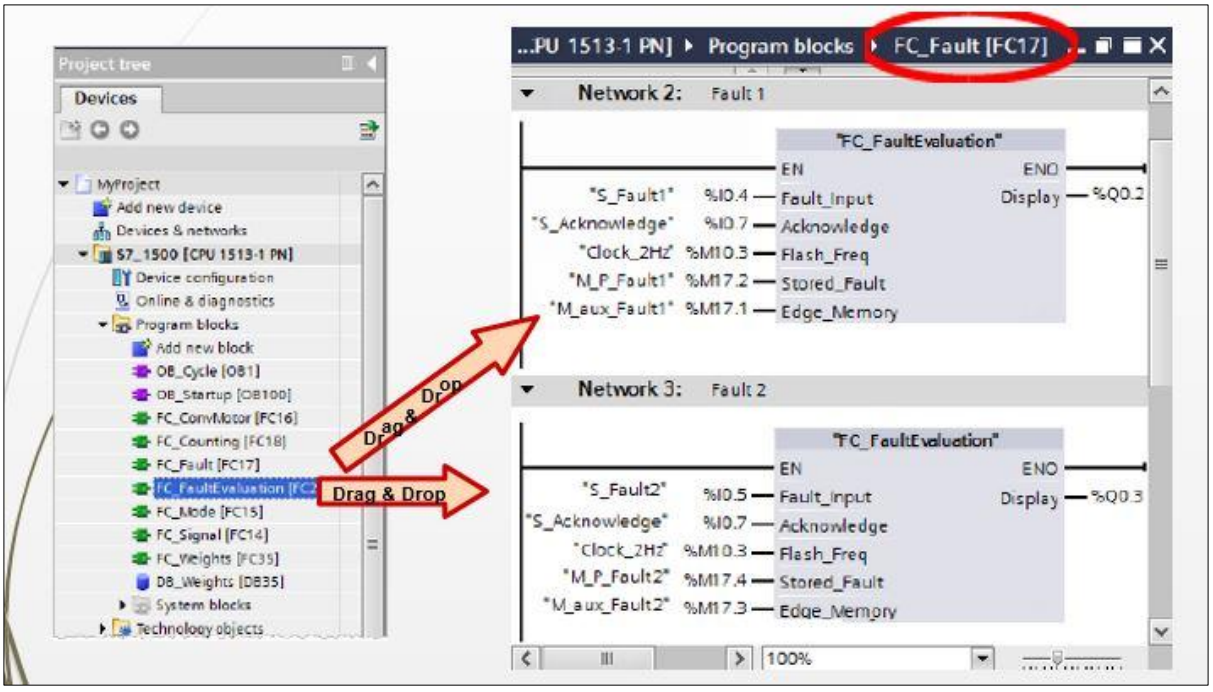
Declaration of Formal Parameters



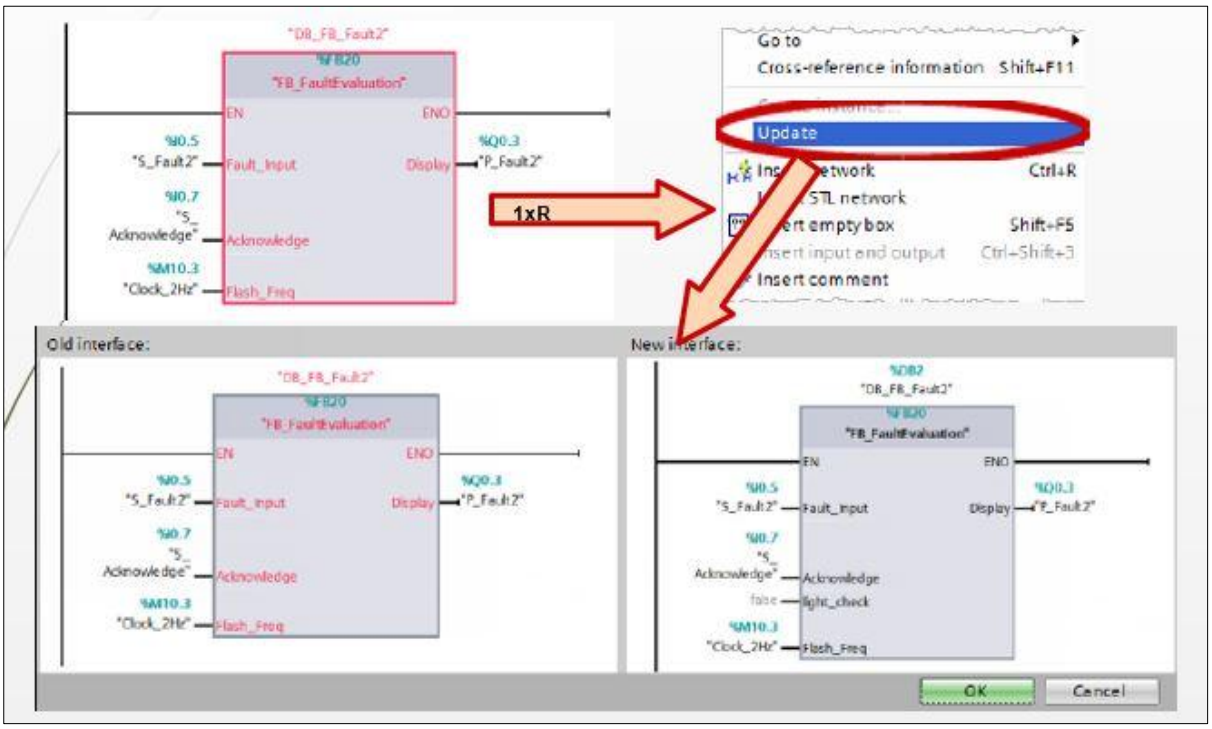
Declaring Parameter-assignable Block



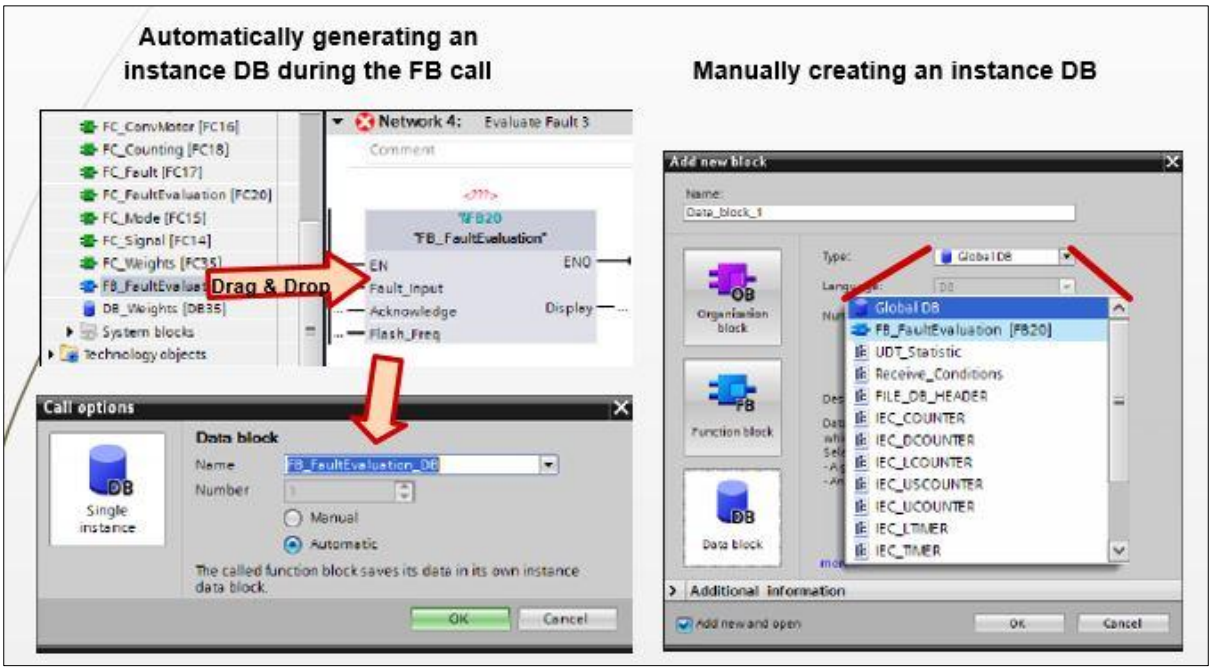
Calling a Parameter-assignable Block



Generating Instance Data Blocks



Updating a Block Call

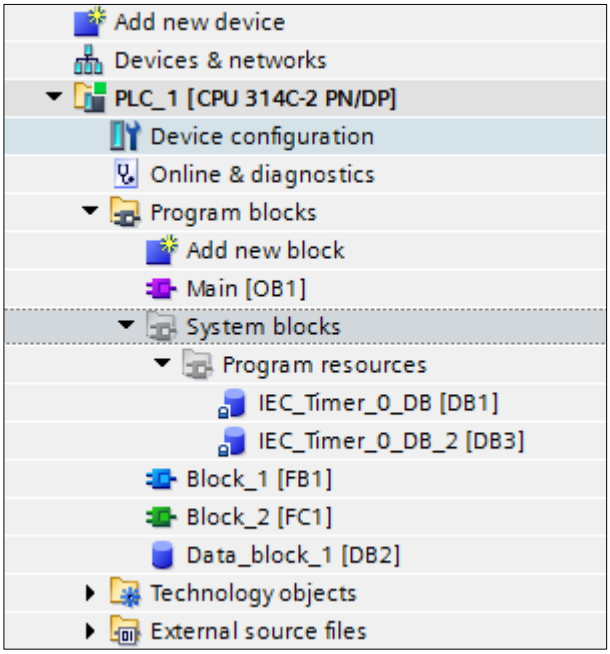


Organization Blocks

Program Blocks

You can find a "Program blocks" folder in the project tree, in which you can create and manage the following blocks (Description is already given in previous chapter):

- Organization blocks (OB)
- Function blocks (FB)
- Functions (FCs)
- Data blocks (DB)



A "System blocks" subfolder containing another subfolder, "Program resources", is also created in the "Program blocks" folder the first time you drag an instruction to your program which is an internal system function block. The instance data block of the internal system function block is also pasted to the "Program resources" folder. You can move or copy such instance data blocks from the "Program resources" folder to any other folder and rename or delete them. You can also move your blocks into the "Program resources" folder. Blocks in the "Program resources" folder that are not required to run the user program are removed during the next compilation. If the "Program resources" folder contains no more blocks then it is also deleted with the "System blocks" folder.

A program cycle OB is automatically generated for each device and inserted in the "Program blocks" folder.

Organization Blocks available in SIEMENS

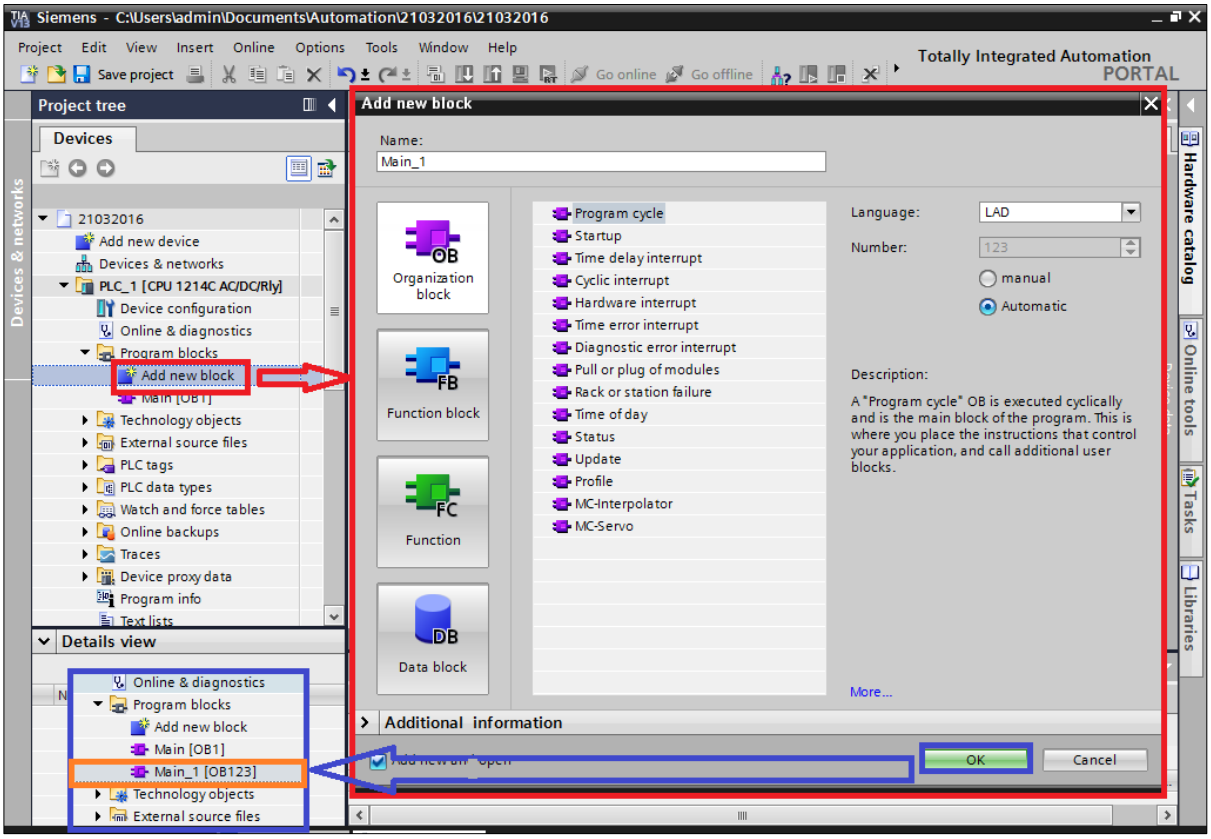
Organizational Blocks and their details which is available in Siemens PLC is described below. Whether it is available in S7-1200 or not that is also mentioned.

S7-1200 , S7-1500			Application	PLC			
Sr No:	OBs No	Function		S7-1200	S7-1500	S7-300	S7-400
1	OB1	For Program Execution	OB1 is the default block for cyclic execution of the user program.	Y	Y	Y	Y
2	OB10 - OB17	For the time-of-day interrupt	The OBs can be started periodically or once at a specific time.	N	Y	Y	Y
3	OB100,101,102	For startup	Startup OBs are processed once when the operating mode of the CPU changes from STOP to RUN.	Y	Y	Y	Y
4	OB121	For the programming error	The operating system of the CPU calls the OB if a programming error occurs while processing an instruction of the user program	N	Y	Y	Y
5	OB122	For the IO access error	The operating system of the CPU calls the OB if an error during direct access to IO data occurs while processing an instruction of the user program.	N	Y	Y	Y
6	OB20	For time delay interrupt.	After expiry of a defined time, the time delay interrupt OBs interrupt the cyclic program processing	Y	Y	N	N
7	OB20-OB23	Time delay interrupt OBs	Event class: Time delay	N	N	Y	Y
8	OB30	For cyclic interrupts.	The cyclic interrupt OBs interrupt the cyclic program processing at defined intervals	Y	Y	N	N
9	OB30-OB38	Cyclic interrupt OBs	Event class: Cyclic	N	N	Y	Y
10	OB40	For HSC hardware interrupts.	The hardware interrupt OBs interrupt the cyclic program processing at the occurrence of a hardware event. The hardware interrupts are not only for "Hardware interrupt HSCs", but also for hardware interrupts of digital channels.	Y	Y	N	N
11	OB40-OB47	Process alarm OBs	Event class: Hardware interrupts	N	N	Y	Y
12	OB55	For the status interrupt	The operating system of the CPU calls the OB if a status interrupt occurs.	N	Y	Y	Y
13	OB56	For the update interrupt or alarm OB	The operating system of the CPU calls the OB if an update interrupt occurs.	N	Y	Y	Y

14	OB57	For the manufacturer- or profile-specific interrupt	The operating system of the CPU calls the OB if a manufacturer- or profile-specific interrupt occurs.	N	Y	Y	Y
15	OB61	For the clocked interrupt	Program parts can be started synchronized with the DP cycle clock or PN send cycle clock.	N	Y	N	N
16	OB61-OB64	Clocked interrupt OB	Event class: Alarm	N	N	Y	Y
17	OB80	For the time error interrupt	If the maximum cycle time is exceeded, the time error interrupt OB interrupts the cyclic program processing	Y	Y	Y	Y
18	OB81	Power supply failure OB	Event class: Fault interrupts	N	N	Y	Y
19	OB82	For diagnostic error interrupt	If the diagnostics-compatible module, for which you have enabled the diagnostic error interrupt, detects an error, the diagnostic error interrupt OB interrupts the cyclic program processing	Y	Y	Y	Y
20	OB83	For pulling and plugging	The operating system of the CPU calls the OB when a configured and non-disabled module or sub-module of the distributed IO is pulled or plugged.	N	Y	Y	Y
21	OB84	CPU hardware error OB	Event class: Fault interrupts	N	N	Y	Y
22	OB85	Program runtime error OB	Event class: Fault interrupts	N	N	Y	Y
23	OB86	For the rack error of a PROFINET IO system	When a DP master system, slave or part of the sub-modules fails, the operating system of the CPU calls the OB	N	Y	Y	Y
24	OB87	Communications error OB	Event class: Fault interrupts	N	N	Y	Y
25	OB88	Processing abortion OB	Event class: Fault interrupts	N	N	Y	Y
26	OB90	Background OB	Event class: Fault interrupts	N	N	Y	Y
27	OB91	For the MC Servo	When creating a technology object (Motion) the OB "MC Servo" is called automatically for processing the technology objects (Motion).	N	Y	N	N
28	OB92	For the MC Interpolator	When creating a technology object (Motion) the OB "MC Interpolator" is called	N	Y	N	N

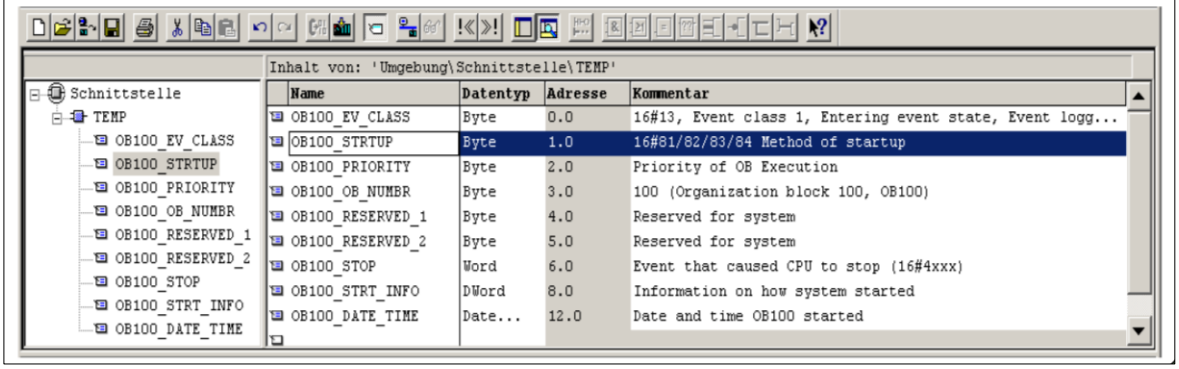
			automatically for processing the technology objects (Motion)				

Creating a New OB



OB Start Information using OB100 as an Example

Address		
0 / 1	Start event	Start-up request
2 / 3	Priority	OB No.
4 / 5	Reserved	
6 / 7	Number of the event that caused the CPU to go into STOP	
8 / 9	Additional information on the current startup???	
10 / 11		
12 / 13	Year	Month
14 / 15	Day	Hours
16 / 17	Minutes	Seconds
18 / 19	1/10 seconds, 1/100 seconds	1 /1000 seconds, weekday



S7-1200 Startup

The startup mode "Warm restart - operating mode before POWER OFF" is set by default in the device configuration. This prevents the CPU from changing unintentionally to the RUN mode during the commissioning phase when the power returns. An unintentional change to the RUN mode will set the outputs and start any machines to which the outputs are connected. However, with this default setting you need a PG with STEP 7 V10.5, STEP 7 V11 or the S7-1200 Tool to be able to change the CPU from STOP mode to RUN mode. The CPU can change to STOP mode for one of the following reasons:

- Inserting an SD card
- Failure of an expansion module
- Other reasons

If STOP mode occurs, you need software for changing to RUN mode.

Remedy:

We recommend setting the startup mode "Warm restart - RUN". Then, the CPU changes automatically back into RUN mode when power returns. In this way, neither you nor your customers need customer support's assistance to set the CPU back into RUN mode.

The table below shows the startup modes of the S7-1200 CPU.

Startup mode	CPU behavior

No startup	CPU remains in STOP mode.
Warm restart – RUN	CPU goes into RUN mode when power returns.
Warm restart - operating mode before POWER OFF	CPU goes into the same mode as before loss of power.

Interrupting the Cyclic Program

The CPU processing is controlled by events. An event triggers an interrupt OB to be executed. You can specify the interrupt OB for an event during the creation of the block, during the device configuration, or with an ATTACH or DETACH instruction. Some events happen on a regular basis like the program cycle or cyclic events. Other events happen only a single time, like the startup event and time delay events. Some events happen when the hardware triggers an event, such as an edge event on an input point or a high speed counter event. Events like the diagnostic error and time error event only happen when an error occurs. The event priorities and queues are used to determine the processing order for the event interrupt OBs.

The CPU processes events in order of priority where 1 is the lowest priority and 26 is the highest priority. Prior to V4.0 of the S7-1200 CPU, each type of OB belonged to a fixed priority class (1 to 26). From V4.0 forward, you can assign a priority class to each OB that you configure. You configure the priority number in the attributes of the OB properties.

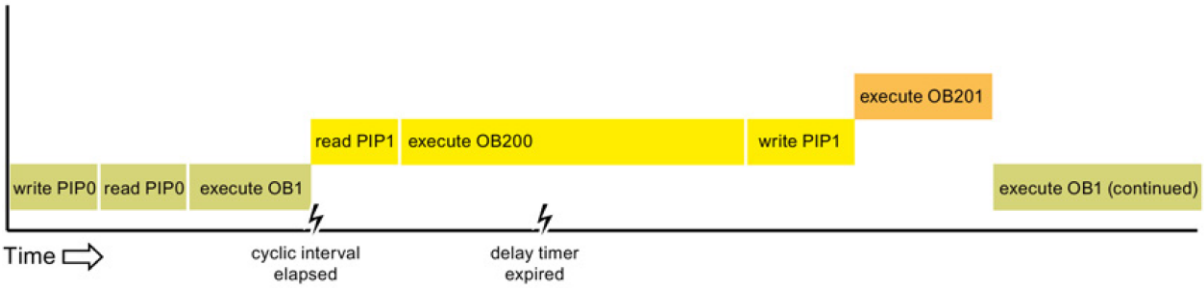
Interruptible and non-interruptible execution modes

OBs (Page 57) execute in priority order of the events that trigger them. From V4.0 forward, you can configure OB execution to be interruptible or non-interruptible. Note that program cycle OBs are always interruptible, but you can configure all other OBs to be either interruptible or non-interruptible.

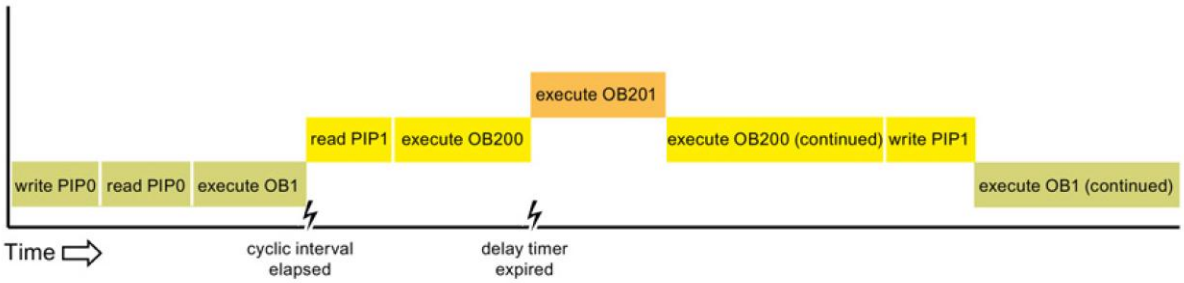
If you set interruptible mode, then if an OB is executing and a higher priority event occurs before the OB completes its execution, the running OB is interrupted to allow the higher priority event OB to run. The higher-priority event runs, and at its completion, the OB that was interrupted continues. When multiple events occur while an interruptible OB is executing, the CPU processes those events in priority order.

If you do not set interruptible mode, then an OB runs to completion when triggered regardless of any other events that trigger during the time that it is running.

Consider the following two cases where interrupt events trigger a cyclic OB and a time delay OB. In both cases, the time delay OB (OB201) has no process image partition assignment and executes at priority 4. The cyclic OB (OB200) has a process image partition assignment of PIP1 and executes at priority 2. The following illustrations show the difference in execution between non-interruptible and interruptible execution modes:



Case 1: Non-interruptible OB execution



Case 2: Interruptible OB execution

Note

If you configure the OB execution mode to be non-interruptible, then a time error OB cannot interrupt OBs other than program cycle OBs. Prior to V4.0 of the S7-1200 CPU, a time error OB could interrupt any executing OB. From V4.0 forward, you must configure OB execution to be interruptible if you want a time error OB (or any other higher priority OB) to be able to interrupt executing OBs that are not program cycle OBs.

Understanding event execution priorities and queuing

The CPU limits the number of pending (queued) events from a single source, using a different queue for each event type. Upon reaching the limit of pending events for a given event type, the next event is lost. You can use a time error interrupt OB to respond to queue overflows.

Each CPU event has an associated priority. In general, the CPU services events in order of priority (highest priority first). The CPU services events of the same priority on a "first-come, first-served" basis.

Event	Quantity allowed	Default OB priority
Program cycle	1 program cycle event Multiple OBs allowed	1[4]
Start up	1 startup event[1] Multiple OBs allowed	1[4]
Time Delay	Up to 4 time events 1 OB per event	3
Cyclic Interrupt	Up to 4 events 1 OB per event	8
Hardware interrupt	Up to 50 hardware interrupt events[2]	18
	1 OB per event, but you can use the same OB for multiple events	18
Time error	1 event (only if configured) [3]	22 or 26[4]

Diagnostic error	1 event (only if configured)	5
Pull or plug of modules	1 event	6
Rack or station failure	1 event	6
Time of day	Up to 2 events	2
Status	1 event	4
Update	1 event	4
Profile	1 event	4

1 The startup event and the program cycle event never occur at the same time because the startup event runs to completion before the program cycle event starts.

2 You can have more than 50 hardware interrupt event OBs if you use the DETACH and ATTACH instructions.

3 You can configure the CPU to stay in RUN if the scan cycle exceeds the maximum scan cycle time or you can use the RE_TRIGR instruction to reset the cycle time. However, the CPU goes to STOP mode the second time that one scan cycle exceeds the maximum scan cycle time.

4 The priority for a new V4.0 or V4.1 CPU is 22. If you exchange a V3.0 CPU for a V4.0 or V4.1 CPU, the priority is 26: the priority that was in effect for V3.0. In either case, the priority field is editable and you can set the priority to any value in the range 22 to 26.

Refer to the topic "Exchanging a V3.0 CPU for a V4.1 CPU (Page 433)" for more details.

In addition, the CPU recognizes other events that do not have associated OBs. The following table describes these events and the corresponding CPU actions:

Event	Description	CPU action
I/O access error	Direct I/O read/write error	The CPU logs the first occurrence in the diagnostic buffer and stays in RUN mode.
Max cycle time error	CPU exceeds the configured cycle time twice	The CPU logs the error in the diagnostic buffer and transitions to STOP mode.
Peripheral access error	I/O error during process image update	The CPU logs the first occurrence in the diagnostic buffer and stays in RUN mode.
Programming error	program execution error	If the block with the error provides error handling, it updates the error structure; if not, the CPU logs the error in the diagnostic buffer and stays in RUN mode.

Interrupt latency

The interrupt event latency (the time from notification of the CPU that an event has occurred until the CPU begins execution of the first instruction in the OB that services the event) is approximately 175 μ sec, provided that a program cycle OB is the only event service routine active at the time of the interrupt event.

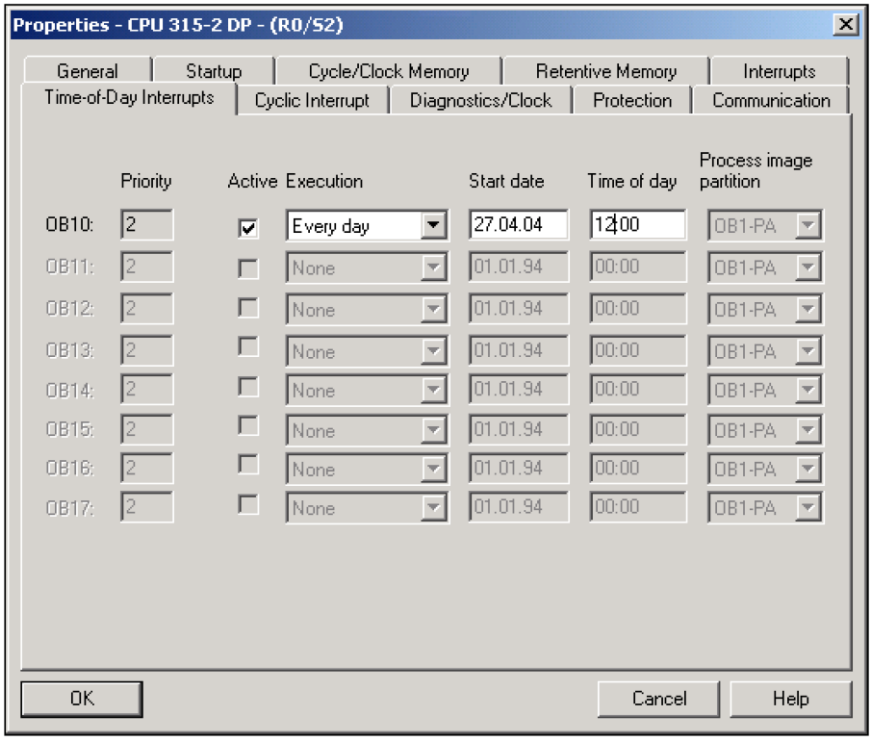
Time-of-Day Interrupt (OB 10)

Description

A time-delay interrupt OB is started after a configurable time delay of the operating system. The delay time starts after the SRT_DINT instruction is called.

You can use up to four time-delay interrupt OBs or cyclic OBs (OB numbers >= 123) in your program. If, for example, you are already using two cyclic interrupt OBs, you can insert a maximum of two further time-delay interrupt OBs in your program.

You can use the CAN_DINT instruction to prevent the execution of a time-delay interrupt that has not yet started.



Function of time-delay interrupt OBs

The operating system starts the corresponding OB after the delay time, which you have transferred with an OB number and an identifier to the SRT_DINT instruction.

To use a time-delay interrupt OB, you must execute the following tasks:

- You must call the instruction SRT_DINT.
- You must download the time-delay interrupt OB to the CPU as part of your program.

The delay time is measured with a precision of 1 ms. A delay time can immediately start again after it expires.

Time delay interrupt OBs are executed only when the CPU is in the "RUN" mode. A warm restart clears all start events of time-delay interrupt OBs.

The operating system calls the time-delay interrupt OB if one of the following events occurs:

- If the operating system attempts to start an OB that is not loaded and you specified its number when calling the SRT_DINT instruction.

- If the next start event for a time-delay interrupt occurs before the time delay OB has completely executed.

You can disable and re-enable time-delay interrupts using the DIS_AIRT and EN_AIRT instructions.

Note
If you disable an interrupt with DIS_AIRT after executing SRT_DINT, this interrupt executes only after it has been enabled with EN_AIRT. The delay time is extended accordingly.

Start information

- None
- Optimized start information:

Name	Data type	Meaning
sign	WORD	User ID: Input parameter SIGN from the call of the "SRT_DINT" instruction

Cyclic Interrupt (OB35)

S7 provides up to nine cyclic interrupt OBs (OB 30 to OB 38). You can use it to start programs after equidistant time phases. The following table shows the default values for the time frame and priority classes for the cyclic interrupt OBs.

Cyclic interrupt OB	Default value for time frame	Default value for the priority class
OB 30	5 s	7
OB 31	2 s	8
OB 32	1 s	9
OB 33	500 ms	10
OB 34	200 ms	11
OB 35	100 ms	12
OB 36	50 ms	13
OB 37	20 ms	14
OB 38	10 ms	15

Function of the cyclic interrupt OBs

The equidistant start times of the cyclic interrupt OBs are determined by the cycle clock and the phase offset.

Note
You must make sure that the run time of each cyclic interrupt OB is significantly shorter than its interval. If a cyclic interrupt OB has not been completely executed before it is due

for execution again because the interval has expired, the time error OB (OB 80) is started. The cyclic interrupt that caused the error is executed later.

You can use the "DIS_IRT" instruction to disable the call of the cyclic interrupt OBs, the "EN_IRT" instruction to re-enable it, and the "DIS_AIRT" and "EN_AIRT" instructions to delay it.

Refer to the specifications of your specific CPU for the range of the parameters cycle clock, priority class, and phase offset. You can change parameter settings through configuration.

Local data for cyclic interrupt OBs

Hardware Interrupt (OB 40)

Description

S7 provides up to eight independent hardware interrupts each with its own OB.

Per configuration you specify which channels will trip a hardware interrupt,

- Under which supplementary condition for each signal module.
- Which hardware interrupt OB is assigned to the individual groups of channels (as default, all hardware interrupts are processed by OB 40).

With CPs and FMs, you must use the appropriate software for the module for this.

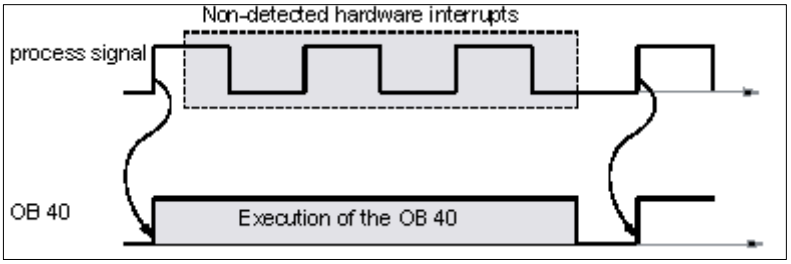
You select the priority classes for the individual hardware interrupt OBs per configuration.

Function of the hardware interrupt OBs

After the module triggers a hardware interrupt the operating system identifies the slot and determines the associated hardware interrupt OB. If this has a higher priority than the priority class active at the moment then it will be started. The channel-specific acknowledgment is sent after this hardware interrupt OB has been executed.

If another event that triggers a hardware interrupt occurs on the same module during the time between identification and acknowledgment of a hardware interrupt, the following applies:

- If the event occurs on the channel that previously triggered the hardware interrupt, then the associated interrupt is lost. The following figure illustrates the connection between a process signal and the execution of the associated hardware interrupt OB based on the example of a channel of a digital input module. The triggering event is the rising edge. The associated hardware interrupt OB is OB 40.



- If the event occurs on a different channel of the same module, then no hardware interrupt can be triggered at that moment. However, it is not lost, it is then triggered after the currently active hardware interrupt has been acknowledged.

If a hardware interrupt is triggered the OB of which is currently active on account of a hardware interrupt of another module, then the new request is registered and the OB is worked off at the specified time.

You can use the "DIS_IRT" instruction to disable the call of the hardware interrupt OBs, the "EN_IRT" instruction to re-enable it, and the "DIS_AIRT" and "EN_AIRT" instructions to delay it.

You can use the "WR_PARM", "WR_DPARM", and "PARM_MOD" instructions to assign the hardware interrupt parameters for a module.