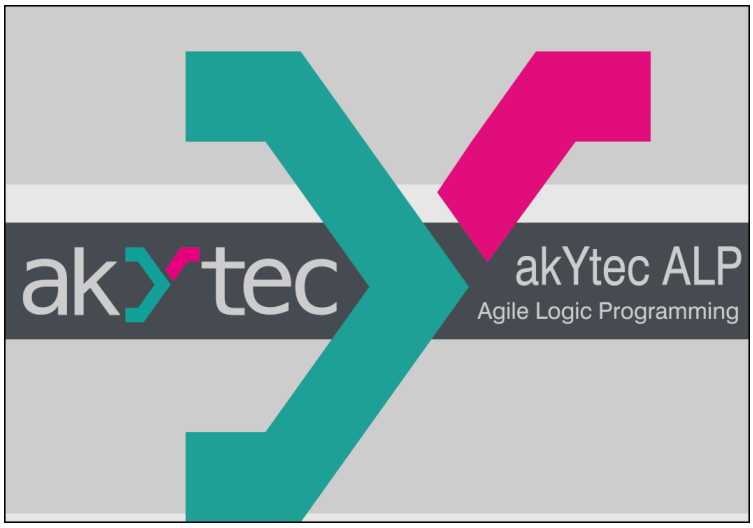


akYtec ALP v2.7

Programming software

User manual

akYtec ALP v2.7_3-EN-130795-1.10

Contents

1 Introduction

ALP is the programming software for programmable devices of akYtec GmbH. The programming language is the graphical language FBD (Function block diagram) and ST (Structure Language) according to IEC 61131-3. The software enables simulation and debugging of the created program and its upload to the non-volatile memory of the device.
The created project contains minimum one circuit program and device configuration.
The first workspace contains the main circuit program. Macros can be created as circuit programs in separate workspaces.
If the target device has a display, it can be programmed using display forms in separate workspaces.
Only one project can be opened in ALP at a time.
The akYtec ALP functionality differs for the following groups of devices:

– devices on the initial hardware platform (PR100, PR102, PR200 and SMI200)
– devices on the new hardware platform (PR103 and newer)

Basic akYtec ALP functionality is available for all devices, functionality and interfaces for devices on the new platform are not available for older devices.

1.1 System requirements

Operation systems:

– Windows 7 (SP1+)
– Windows 8.1
– Windows 10
– Windows 11

System libraries:

– Microsoft .NET Framework 4.8
– Microsoft .NET Desktop Runtime 6.0.8
– Microsoft Visual C++ 2015-2022

Recommended hardware requirements:

– 3.2 GHz processor
– 4 GB RAM
– 700 MB available hard disk space
– Free USB port
– Keyboard and mouse
– Screen resolution 1280×800

Internet connection is required for:

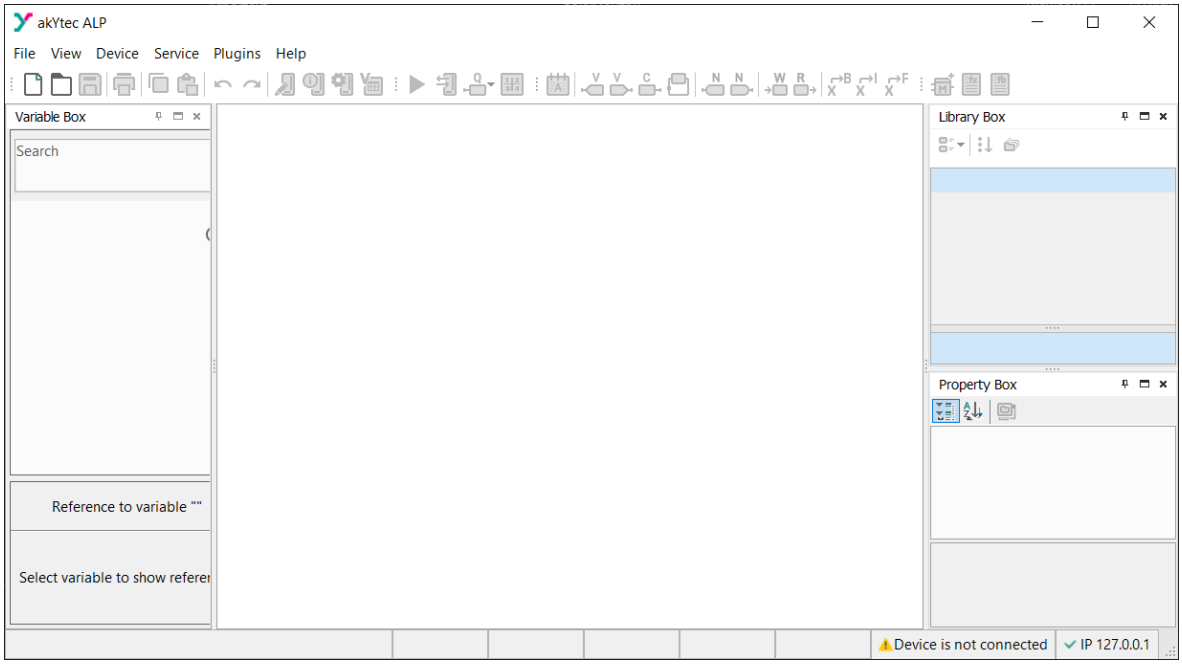
– Software update
– Device firmware update
– Macros download in Component manager

1.2 Terms and abbreviations

– **ALP** — programming software for programmable devices of akYtec GmbH.
– **EEPROM** — electrically erasable programmable read-only memory of the device.
– **FBD (Function Block Diagram)** — graphical programming language supported by IEC 61131-3.
– **Function** — Structural program unit with one return value. The function does not store information about its internal state, i.e. if the function is called with the same input values, it returns the same output value.
– **Function block** — Structural program unit with internal memory and one or more output values. It is used in program as an instance, i. e. a copy with its own memory.
– **Macro** — user function block.
– **Program cycle** — execution time of the circuit program, which depends of its complexity.
– **Project** — user application created for a specific device with ALP software, contained the circuit program.
– **ST (Structured Text)** — a text programming language supported by IEC 611131-3.

– **Target device** — device type for which the project is created.
– **Workspace** — graphic area in the user interface for the program creation, modification and debugging.

2 User interface



1. ***Title bar*** — shows the name of the software and the path to the open project file.
2. ***Menu bar 2.1*** — consists of the following groups: File, View, Device, Service, Plugins and Help.
3. ***Tool bars*** *2.2* – Standard, Service and Insert: quick access to the essential functions of ALP.
4. ***Library box*** *2.3* – the panel shows all the functions, function blocks, and macros that can be used in the project (drag-and-drop using).
5. ***Property Box*** *2.4* — the panel where the properties of the selected project element can be viewed and modified.
6. ***Workspace*** *2.6* — the graphic area in which circuit programs, ST programs, display elements or display forms can be created and modified.
7. ***Status bar*** *2.7* — shows status and error messages, target device memory usage, status of the connected device and the programming interface.
8. ***Display Manager*** *2.8* — a tool to program the display information (available only for the devices with display).
9. ***Variable Box*** *2.5* — the panel shows all project variables with their parameters and references (drag-and-drop using).
10. ***Component manager*** *3.10* — special tool in a separate window to access Online Database and to add the elements from the Online Database to the offline library or to the project library. Internet connection is needed.

2.1 Main menu

File

New project	Open a new project. The current project will be closed
Change target device	Change the target device in the project
Open project	Open a previously saved project
Save active workspace	Save the active workspace
Save project	Save the current project
Save project as…	Make a copy of the project in a different folder or under a different name

Create key file…	Create a file with a key to protect the project from unauthorized access (in development)
Project information	View and modify the *information about the project 3.9*
New macro	Open a new macro in the separate *workspace 3.11*
Import	Import a macro, an ST function or an ST function block from a file into the project library
Export	Save the current macro, ST function or ST function block as file
Component manager	Open the *Component manager 3.10* interface
Print	Open the dialog to set the print options and print the active workspace
Recent projects	List of recently opened projects
Exit	Close ALP

View

Undo	Undo the last action
Redo	Redo the last undone action
Status indictors	Add / remove the in indicators to / from the *status bar 2.7*
Library Box	Show / hide *Library Box 2.3*
Property Box	Show / hide *Property Box 2.4*
Variable Box	Show / hide *Variable Box 2.5*
Display manager	Show / hide *Display Manager 3.4*
Reset layout	Restore the default panel layout

Device

Transfer application to device	Upload the current project to the device memory
Firmware update	Update the firmware of the connected device
Device information	*Information 7.1* about the software, the target device and the connected device
Variable table	The *editable table 5* of the project variables with their parameters
Calibration	Start analog I/O *calibration 7.4*
Device configuration	Open *Device configuration 4* interface
Port configuration	Settings of the *programming interface 3.6*

Service

Arrange elements	Function blocks of the same type are automatically renumbered in the workspace from top to bottom and from right to left
Simulation mode	Start / stop *simulation 3.5* mode

Language	Select the interface language
OFFLINE mode	Activate OFFLINE mode

Help

Automatic update check	If activated, the update check is performed on program startup
Check for updates...	Check for software updates
Help	Open Help window
Version history	Running list of the changes have been made in all software versions
About Software	Information about the current software version

2.2 Toolbars

Standard

	New project	Open a new project. The currently opened project will be closed
	Open project	Open a previously saved project
	Save project	Save the current project
	Print	Set the print options for the active workspace
	Copy	Copy the selected element
	Paste	Paste the copied element
	Undo	Undo the last action
	Redo	Redo the last undone action
	Transfer application to device	Upload the current project to the device memory
	Device information	Information about the software, the target device and the connected device
	Device configuration	Device configuration
	Variable table	Table of project variables

Service

	Simulation	Start / stop simulation
	Online debugging	Start / stop online debugging
	Execution order	Change the execution order for the outputs or delay lines on a circuit program or in a macro
	Arrange elements	Function blocks of the same type are automatically renumbered in the workspace from top to bottom and from right to left

Insert

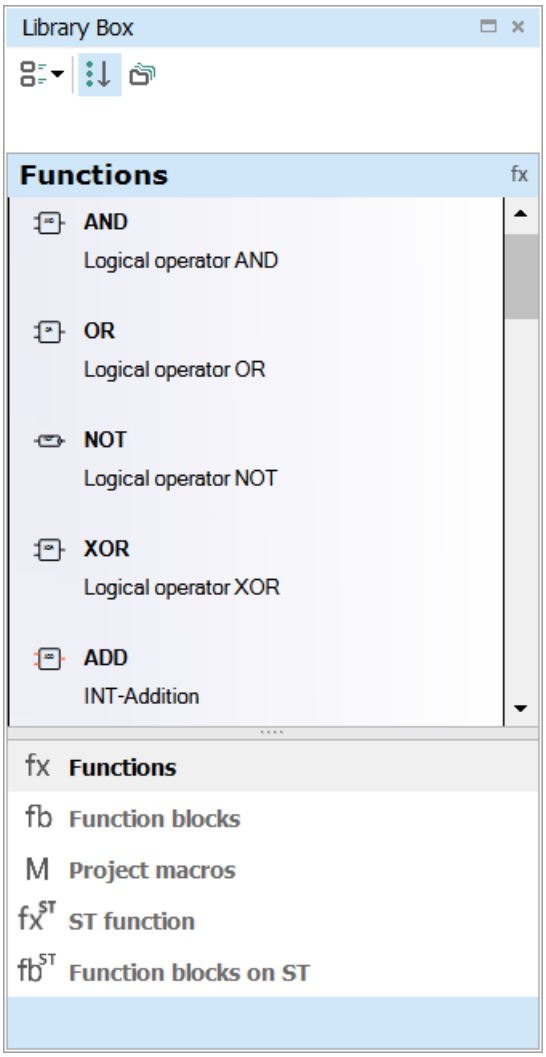
	Text field	Text field to comment the program
	Variable output block	Variable, which value can be written in the program
	Variable input block	Variable, which value can be read in the program
	Constant block	Constant value
	Delay line	Feedback with one-cycle delay
	Network variable output block	Variable, which value can be written via network
	Network variable input block	Variable. which value can be read via network
	WriteToFB block	Connects the output value of the block to the selected parameter of the selected function block and used to change the parameter
	ReadFromFB block	Connects the output value of the block to the selected parameter of the selected function block and used to read the parameter
	Conversion to BOOL	Conversion of any values to a BOOL value
	Conversion to INT	Conversion of any values to an INT value
	Conversion to REAL	Conversion of any values to a REAL value
	New macro	New user macro
	New ST function	New user function in ST language
	New ST function block	New user function block in ST language

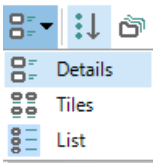
2.3 Library Box

The ***Library Box*** panel is a summary of program blocks available in the project. The panel consists of libraries:

– *Functions 6.1*
– *Function blocks 6.2*
– *Project macros 6.3*
– *ST functions 3.12*
– *ST function bocks 3.13*

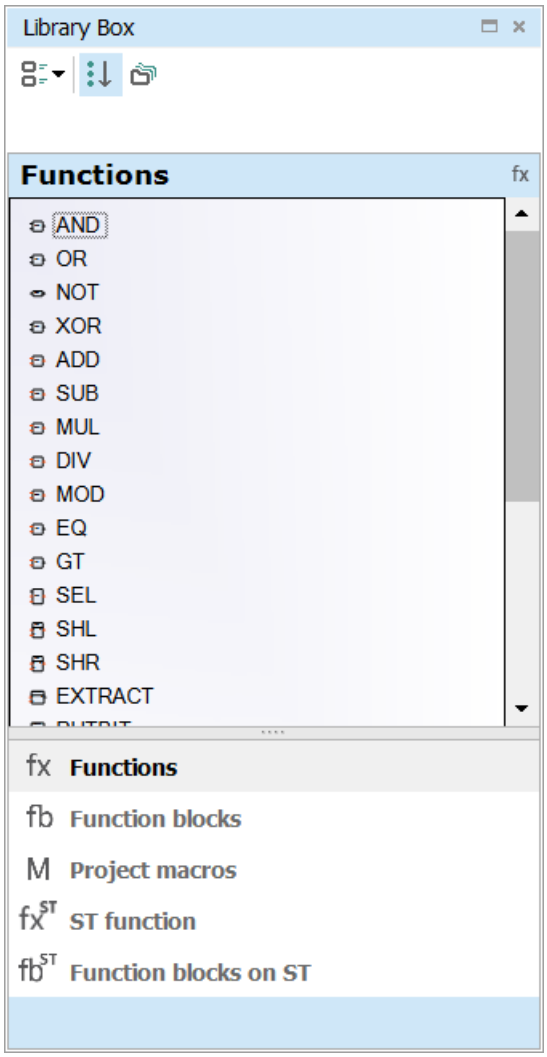
Select an item on the lower toolbar of the panel to show the respective content.
The standard position of the panel is the upper right window corner (can be changed).



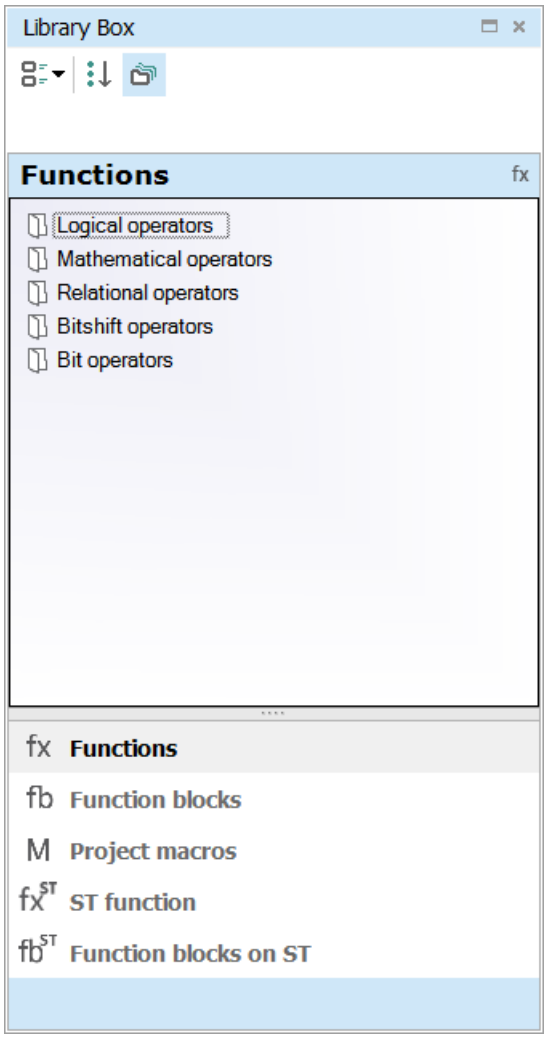


The panel view can be changed using the icons on the upper toolbar.

– Click the icon **Show all** to show all the blocks of the selected library.



– Click the icon **Show grouped** to show the blocks of the selected library group. Double-click the folder to open it.



For description of the library groups and individual blocks see section *Library 6*.

2.4 Property box

The panel Property Box is used to view and modify the parameters of the program elements. Select the element to view its properties. If no item is selected, the panel displays the properties of the workspace.
The standard position of the panel is the lower right window corner (can be changed).
The parameters are shown grouped by categories by default.

To show them in alphabetical order, click the icon .

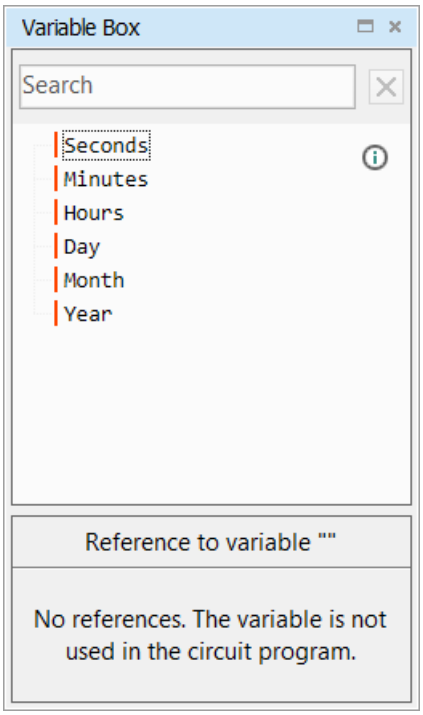
To show them grouped, click the icon .
Select the parameter input field to edit its value.

2.5 Variable Box

The panel ***Variable Box*** shows the project variables from the variable table.
The standard position of the panel in the upper left window corner and can be changed.
You can view the information about the variable in a tooltip text.

Variable block in the workspace

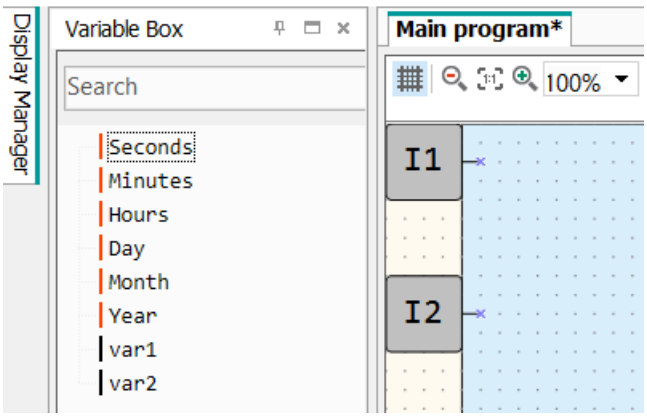
Drag-and-drop a variable to place it in the circuit program as an input block.



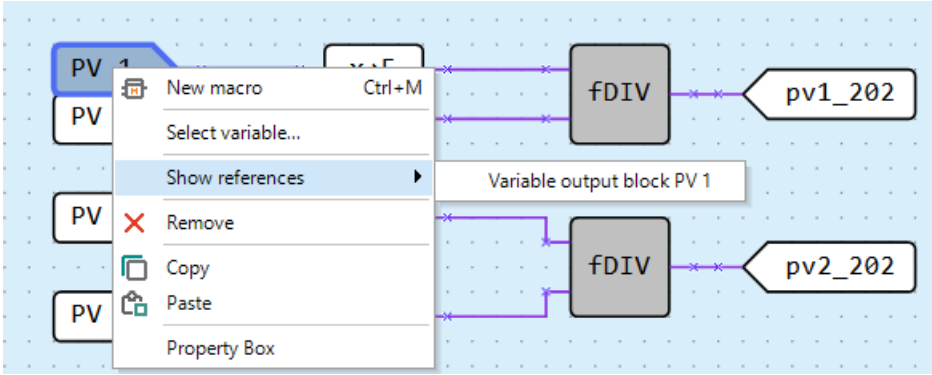
To use a variable as an output block, hold the Shift key pressed as you drag-and-drop the variable. If a variable is drag-and-dropped onto a connection pin of a block, the created variable block is connected to this connection pin.

References

The variable references are shown as links in the lower panel part. If you click on the link, the block to which the variable is referred is highlighted in the workspace.

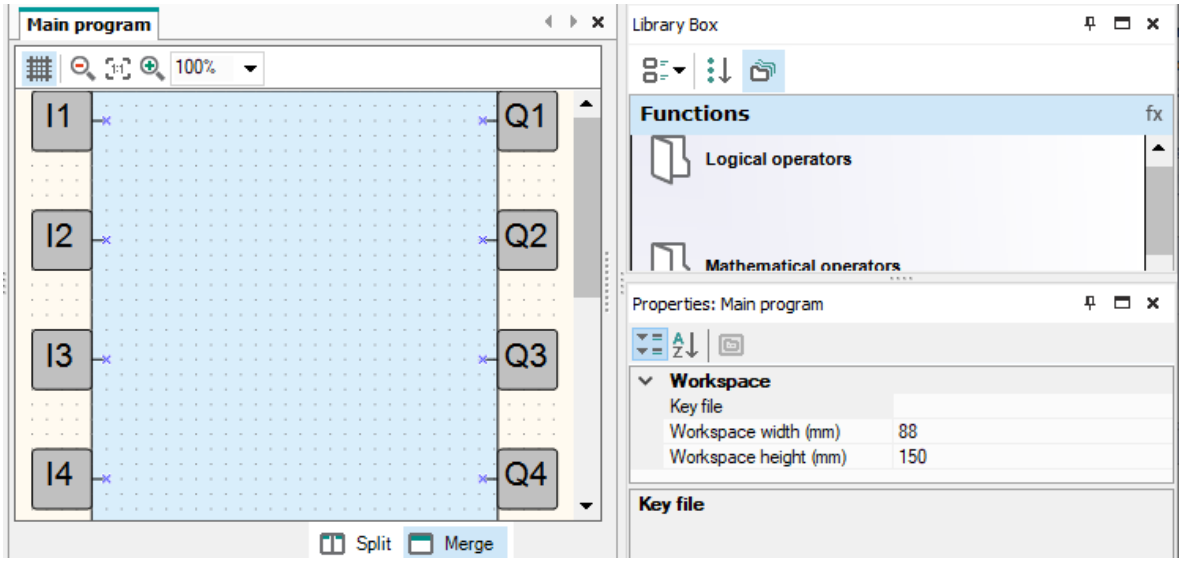


If a variable is used at more than one place in the project, all the references can be viewed with the item ***Show references*** in the variable block context menu. Click on the link to view the reference.



2.6 Workspace

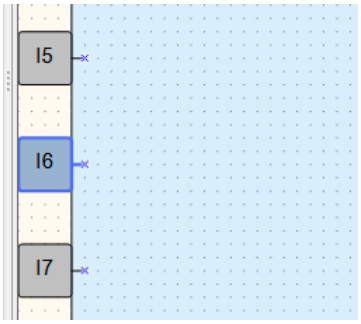
When a project is open, the workspace with the tab ***Main program*** is shown in the middle of the window.
If your device supports ST functions, the **Function Editor** tab appears.



Circuit program is built by placing program blocks and connecting lines between them in the workspace. The size of the workspace can be changed in Property Box. The inputs (left) and outputs (right) are signed as follows:

– **Ix** – digital inputs
– **AIx** – analog inputs
– **Qx** – relay outputs
– **AOx** – analog outputs
– **Fx** – LED indicators

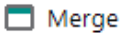
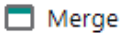
The numbers (x) correspond to the ordinal numbers of physical I/O points of the target device. I/O points can be moved up and down along the workspace by drag-and-drop.



Workspace toolbar

	**Show / Hide grid**	Show / hide vertical and horizontal rulers and a grid in the workspace. If the grid is visible, the blocks and connecting lines are snapped to the grid
	**Zoom –**	Decrease the workspace by 10 %
	**Original size**	Return to the original size (100 %)
	**Zoom +**	Increase the workspace by 10 %

You can set the required scale using the drop-down menu to the right of the buttons described above.

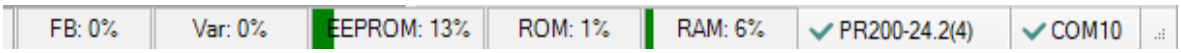
The icons  and  are located on a toolbar below the workspace. Use the icon  to show the same circuit program in two workspaces. It can be useful if the program is too large and you want to view two different parts of the program at the same time. Use the icon  to one workspace.

2.7 Status bar

Status and error messages are displayed in the left field of the status bar.

View

In the right field the status bar contains different status indicators that show information about the memory usage of the target device (resource indicators), status of the connected device and the programming interface. Which indicators are available in the status bar, depends on the type of the target device.



Resource indicators show the used resource in percent of the total available amount. Move the mouse cursor over the indicator to see the absolute amount of the resource.

Indicators

If the device is connected, the status bar contains the following information:

- **FB** — the number of the available and used function blocks.
- **Var** — the number of the available and used variables.
- **Stack** — the number of the available and used stack levels. The stack is used for intermediate calculations in the program.
- **Sys EEPROM** — the available and used system non-volatile memory. The indicator is filled in if the program uses network variables, variables are linked to visualization or device parameters, as well as expansion modules are added to the project.
- **EEPROM** — the available and used non-volatile memory. The indicator is filled in if the standard non-volatile variables are used in the program.
- **ROM** — the available and used ROM memory.
- **Sys RAM** — the available and used system RAM memory. The indicator appears when the RAM is over 80 % full.

  NOTE
  ALP software automatically calculated the available resources of the device and shows a warning if the critical value is reached.

- **RAM** — the available and user RAM memory.
- **Device** — the ripe of the connected device

NOTE
*Click the indicator to switch to **OFFLINE** mode. In this mode the connection to device is interrupted. The next click disables the **OFFLINE** mode.*

– **COMx** — the selected COM port number (programing interface)

NOTE
*Click on the indicator to open the window **Port settings**.*

NOTICE
If any of the indicators overflow, writing a program to the device is disabled.

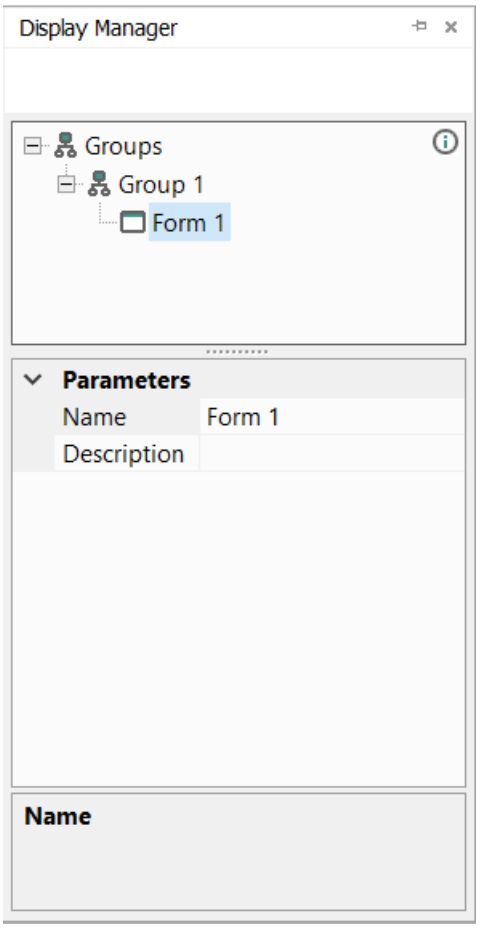
The indicators can be added to the status bar or removed from it in the ***View*** menu.

OFFLINE mode

In the ***OFFLINE*** mode, the connection between ALP and the device is interrupted.
OFFLINE mode can be activated / deactivated using the menu item ***Service > OFFLINE mode*** or by clicking the status indicator ***Device*** . With the next click is OFFLINE mode is deactivated.
For more details see section *Upload project to device* .

2.8 Display manager

If the target device has a display, the displayed information can be programmed using one or more display forms. For further details about display programming see section *Display programming 3.4*
The programming is carried out using the programming tool ***Display Manager.*** The tab Display Manager is located in the upper left corner of the window. Click the tab to open the panel. The panel contains a toolbar, a hierarchical structure (tree) of display forms and the parameters of the selected object.

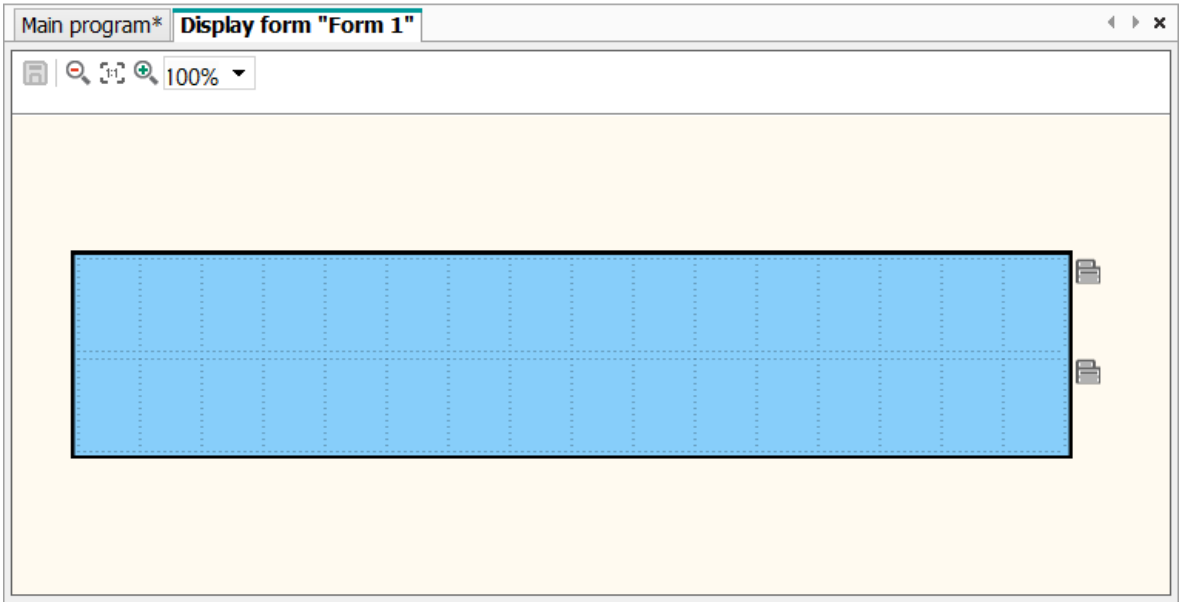


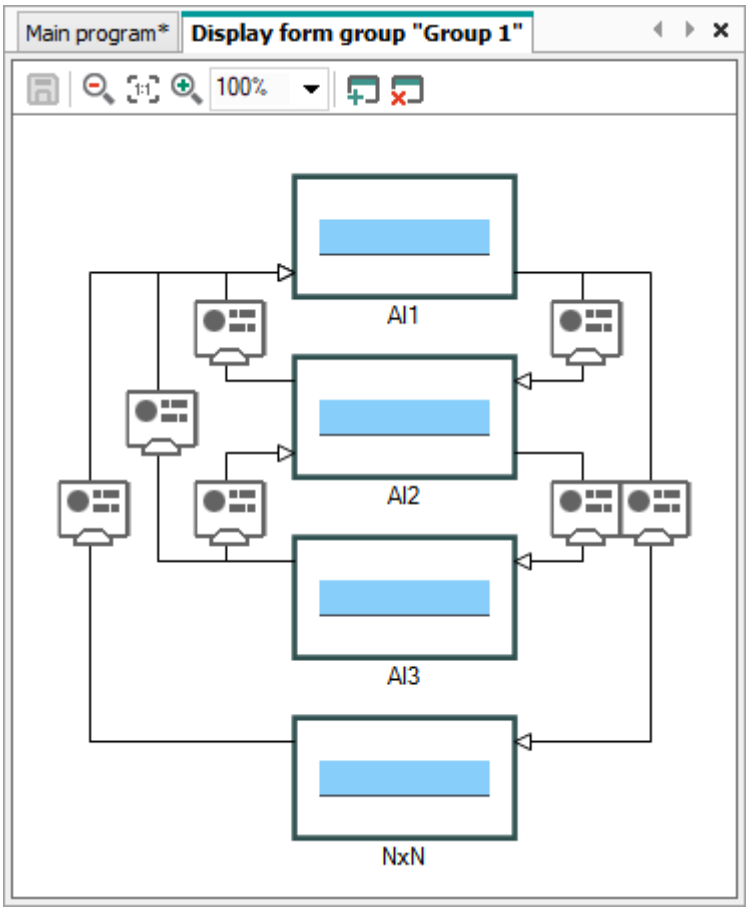
The parameters of the selected display form are shown in the lower part of the panel.

To program the selected form, open it in a separate workspace ***Display Editor***, using the context menu or double-click the form in the group.

	Add display form
	Delete display form
	Edit display form

The workspace shows the selected display form with the icons  to the right of it, which are used to change the number of displayed rows. The rows displayed first are bold outlined.
If more than one display forms are created, you should specify “jumps” between them so that the operator can switch between forms to see the desired information. It can be done in a separate workspace ***Structure Editor***, which represents the graphical structure of display forms with “jumps” and their conditions. To open it, use the command ***Edit group*** in the group context menu.





At the top of the display editor and screen group there are buttons:

	**Save workspace**
	**Zoom out by 10 %**
	**Original size**
	**Zoom in by 10 %**

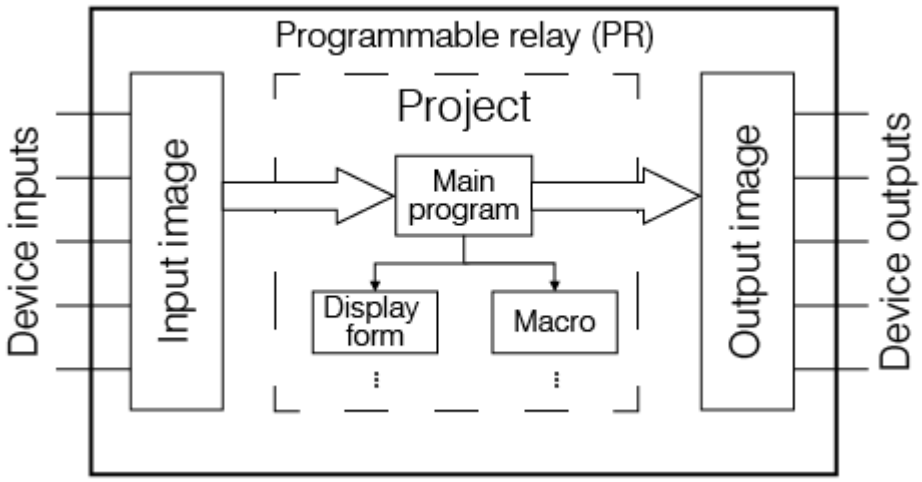
The scale can be changed using the drop-down menu to the right of the buttons.

3 General information

This section describes the basic concepts of the device and the principle of creating a program for loading into the device:

3.1 Program execution

The selected target device determines the number of available inputs and outputs and the availability of a real-time clock.
The general structure of the programmable relay:



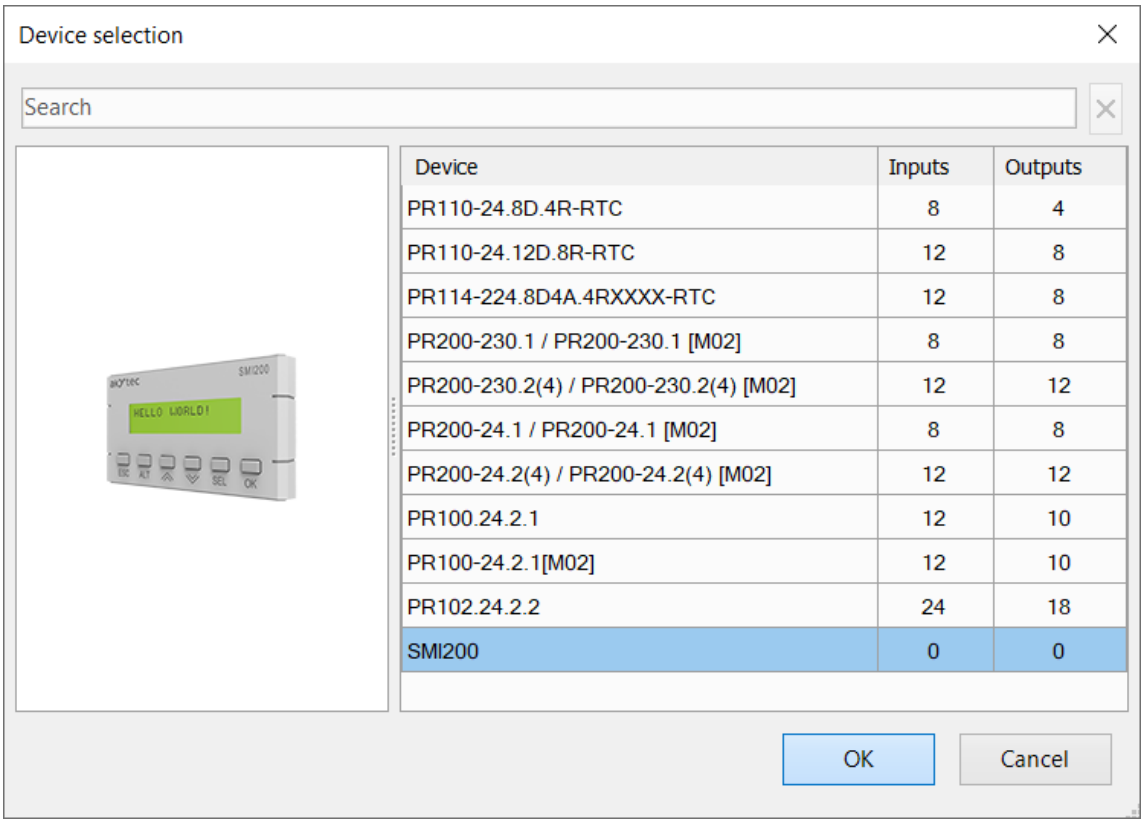
The programmable relay is a kind of PLC with a cyclically executed program:
Step 1 – The status of physical inputs is saved to the input memory cells (Input Image Table).
Step 2 – The input memory cells are read out and the program is executed from its first instruction to the last one.
Step 3 – The results are saved to the output memory cells (Output Image Table) and applied to the outputs.
When the last step is completed, the program runs again from the first step.

3.2 Project creation

Project creation

To create a new project:

1. Click icon in the taskbar or select ***File →New project…*** in the main menu.
2. Select the target device in the dialog window ***Device selection*** and confirm it with ***OK***.



The new project appearance:
– Workspace – empty circuit program
– *Status bar 2.7* – information about available resources
– *Library Box 2.3* – available program blocks
– *Property Box 2.4* – workspace properties

NOTE
If a device is connected to the PC, ALP will suggest the model of the connected device in the selection window.

If the selected device has a display, the *Device Manager 2.8* tab appears to the left from the workspace. With this tool you can *program 3.4* the displayed information.
You can save the current project or open a saved project using the corresponding buttons on the toolbar or in the main ***File*** menu.

Circuit program development

Now you can create the *main circuit program 3.3* in the workspace using the common program blocks from the toolbar ***Insert*** and the specific program blocks from *Library Box 6*. Draw connecting lines between inputs, outputs and blocks to establish logical connections in the program.

Simulation

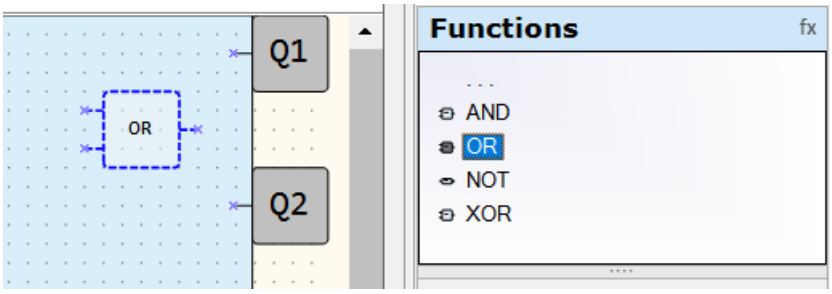
Program can be simulated offline. Start the *simulation mode 3.5* using the menu item ***Service*** →***Simulation*** or the toolbar icon ▶, change the state of the inputs and notice the state of the outputs to check the correctness of the program.

Online debugging

If the device is connected and the program in the device and in ALP is the same, you can use *online debugging 3.8* to check the correctness of the program in the device.

3.3 Program development

It is recommended to start creating a circuit program with planning. The plan should describe all possible states of the device during operation in form of a mode diagram, a table of I/O states, an electrical or functional diagram, etc.
After all the operation tasks are described, the program can be developed using the standard blocks from the toolbar ***Insert*** and the specific blocks from the project library. The project library presented in *Library Box 6* contains the functions and the function blocks available for the target device, as well as the macros added to the project.
To place a library block in the circuit program, select the desired block in Library Box and move it onto the workspace by drag-and-drop.



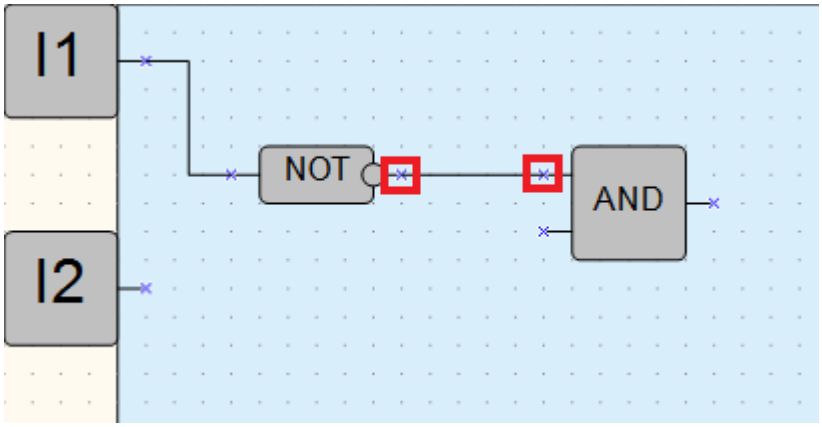
To draw connecting lines between inputs and outputs of the device and program blocks, use the left mouse button

– Click the input pin of the device. The line is attached to it and follows the mouse cursor.
– To change the line direction, click a point in the workspace.
– Pull the line up to the input pin of a block and click on it to finish the line.

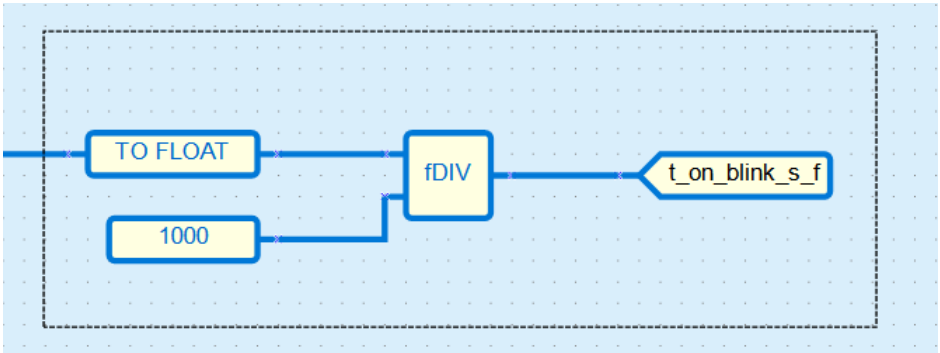
The connecting line can be drawn only between connection pins assigned to the same data type.

NOTE
If the input and output types are not identical, the line will not be created.

Click the block to select it. To select a group, pull the rectangle around several blocks.

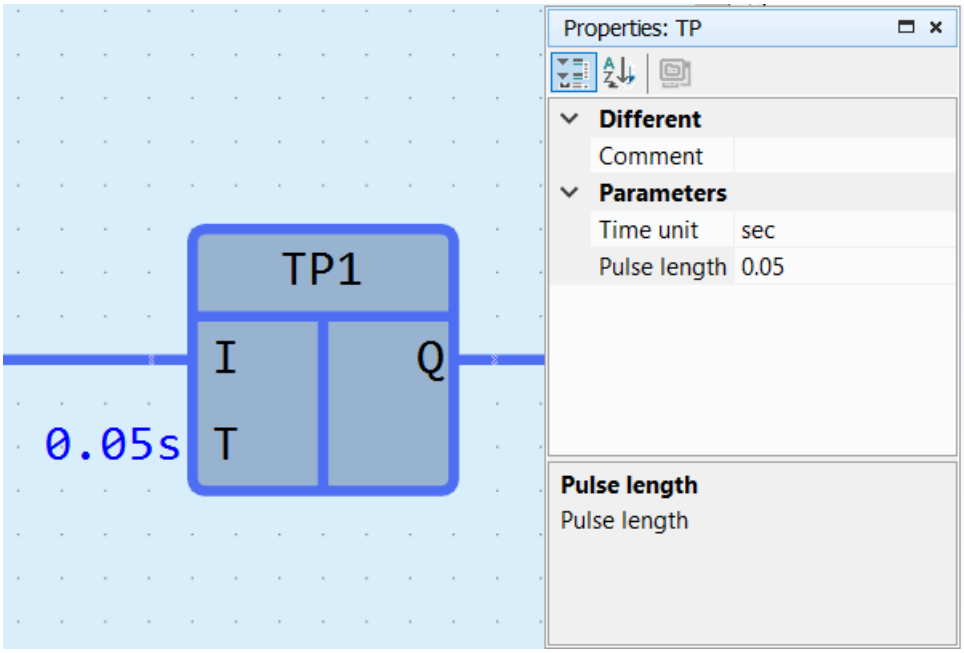


To connect the connection pins assigned to different data types, use conversion blocks.

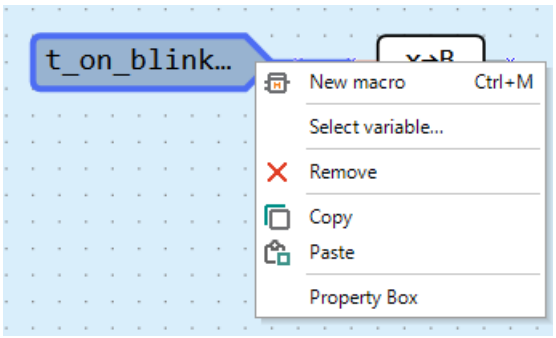


Component settings

The parameters of each element can be set in *Property Box 2.4*.



Use block context menu for all manipulations available for the element.



To develop the program, the following blocks and functions are used, called in the insertion panel:

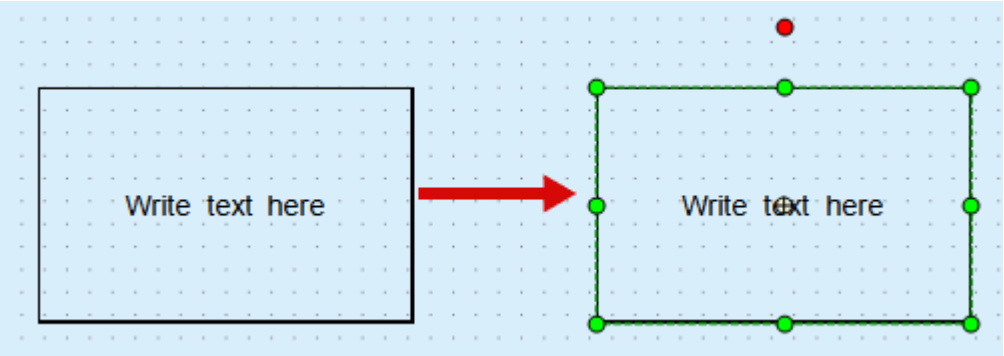
Text field 3.3.1	Placing a text comment on the circuit program
Variable block 3.3.2	Placing a variable block for writing or reading program values
Constant block 3.3.3	Placing a block with a fixed numeric value
Delay lines 3.3.4	Creating a delay for one cycle of transferring a value from the component's output to its input

Network variable block 3.3.5	Placing blocks for data exchange between devices connected to a common network
Read / write to FB 3.3.6	Writing/reading the values of individual parameters from the FB to a variable and vice versa
Conversion blocks 3.3.7	Converting values of different types for transmission
Arrange elements 3.3.8	Reassignment of sequence numbers of FB schemes
Execution sequence 3.3.9	Changing the order in which program output values are calculated

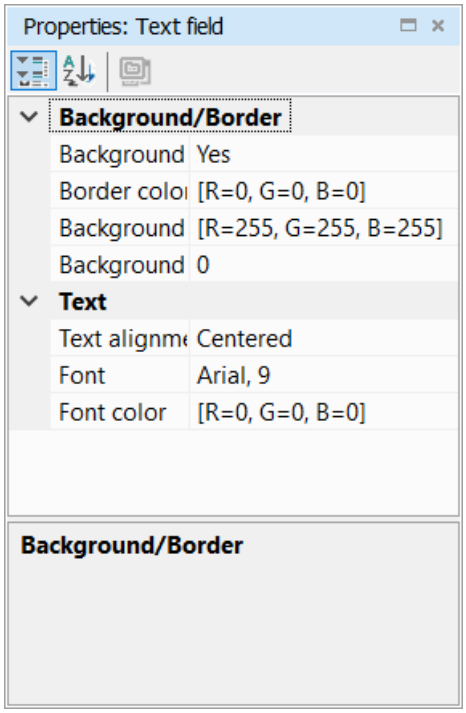
3.3.1 Text field

The text fields are used to explain the program.

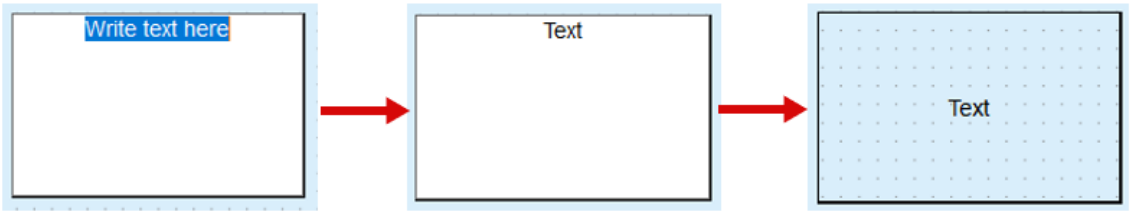
To add a text field to the program, click the item  in the toolbar ***Insert***, then click the point in the workspace to place the upper/left corner of the text field and draw a rectangle to set its size.



The parameters of the text field can be changed in ***Property Box***.



To make the background color of the text block visible, it is recommended to set the ***Background transparency*** greater than 20 %.
Double-click the text field to write the text.



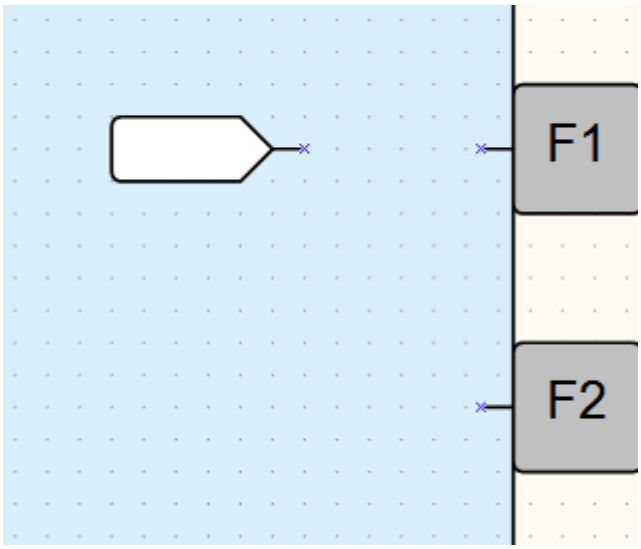
3.3.2 Standard variable block

The variable block enables the use of a variable in a circuit program.
To add a variable block to the program, click on its icon in the insert panel:

– **input variable block** — to pass a value to the program

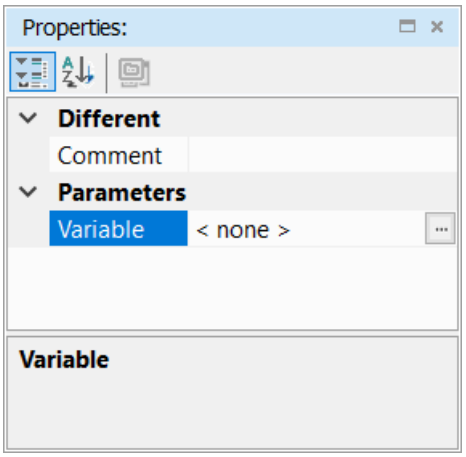
– **output variable block** — to write values from the program into it

Click the point in the workspace to place the variable block.

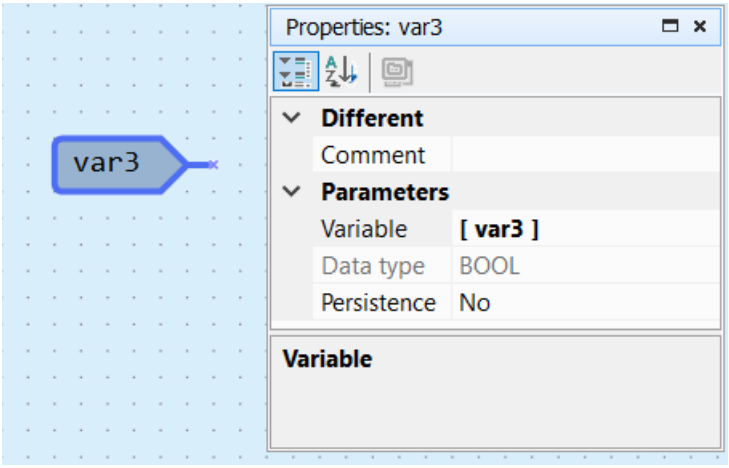


A variable block can also be added to the program from *Variable table 2.5*.
To assign a variable to a variable block:

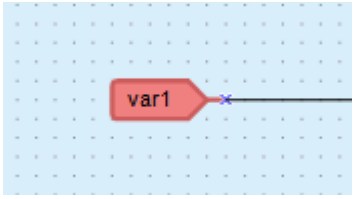
1. Select a variable block.
2. Double click the block or click the icon «**...**» in the row ***Variable*** in Property Box to select a variable for block or create a new one in the opened *variable table 2.5*.



3. Confirm the selection with ***OK***. The variable is assigned to the block.



Only the appropriate tabs in the table are available for selection. The availability is limited by the block type.
Connect the variable block to the desired element in the workspace.
If the variable block is highlighted in red, it means that the creation is incorrect or not completed. The information about the error is displayed in the status bar.

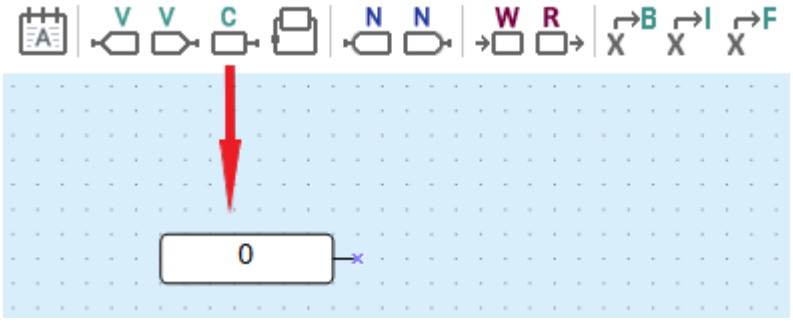


It is recommended to start programming with the creation of variables in the variable table.
If the variable is used more than once in a project, all references can be followed with the item ***Show references*** in the variable block context menu. The function is also available in simulation and online debugging modes.

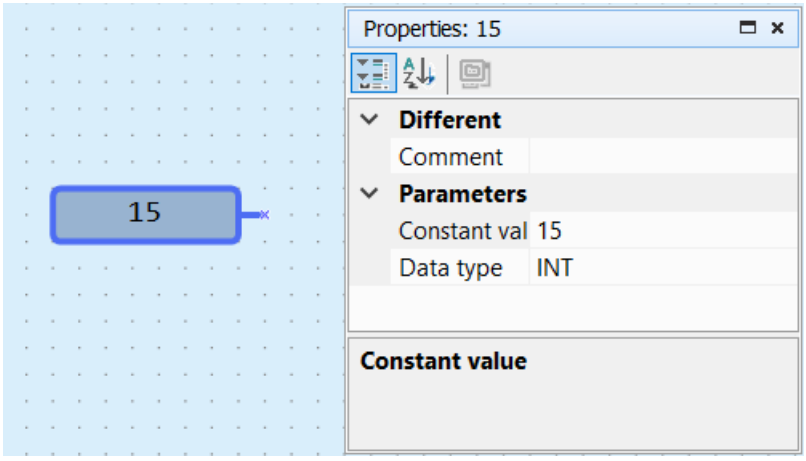
3.3.3 Constant block

To add a constant value to the program, click the icon 

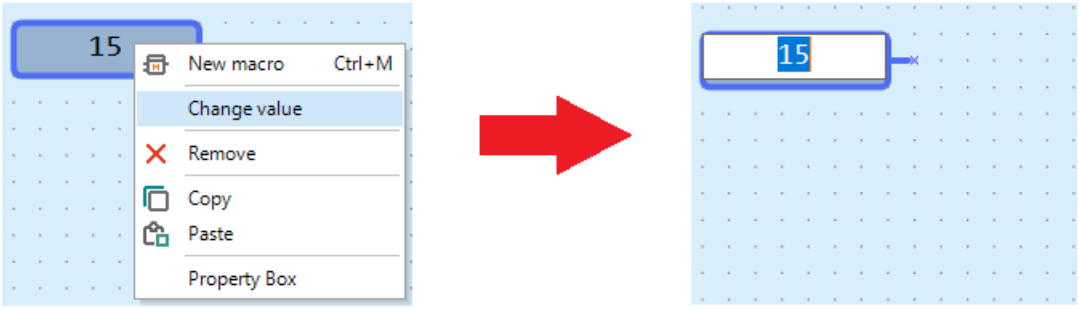
in the toolbar ***Insert***, then click the point in the workspace to place the constant block.



Select the data type using the icon «**…**» in the row ***Data type*** and enter the value on the row ***Constant value*** in ***Property Box***.



The value of the constant is not subject to change throughout the program execution.
It can be changed by double-clicking on the constant block, in ***Property Box*** or by selecting ***Change value*** in the block context menu.



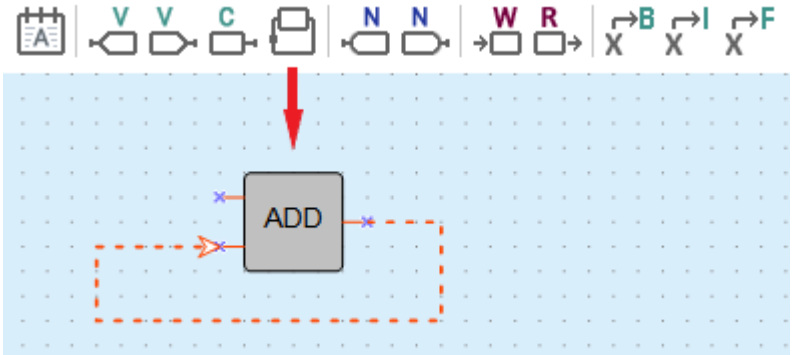
Data types valid values:

BOOL	0 / 1
INT	0 … 4,294,967,295
REAL	–3.402823e+38 … 3.402823e+38

3.3.4 Delay line

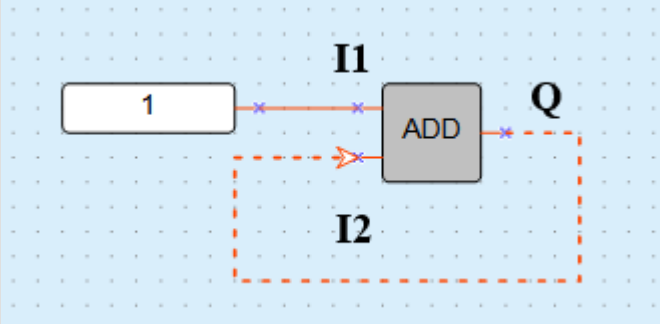
The delay line is used to transfer the value from the block output to the block input, delayed for one cycle. The output and input may belong to different blocks.

Click the icon  in the toolbar ***Insert*** and draw a line from the output to the input of a function block. The delay line is displayed as a red dashed line with an arrow.



Example:

A constant value 1 is transferred to the input I1 of the addition block ADD (Integer). A value from the block output (Q) calculated in the previous cycle is transferred to the input I1 over delay line.



Cycle signal values:

Cycle	1	2	3	4	5	6	7	8	9	10
I2	0	0	1	1	2	2	3	3	4	4
Q	1	1	2	2	3	3	4	4	5	5

3.3.5 Network variable

The network input and output variable blocks are special type of variable blocks for data exchange between devices connected to a common network.

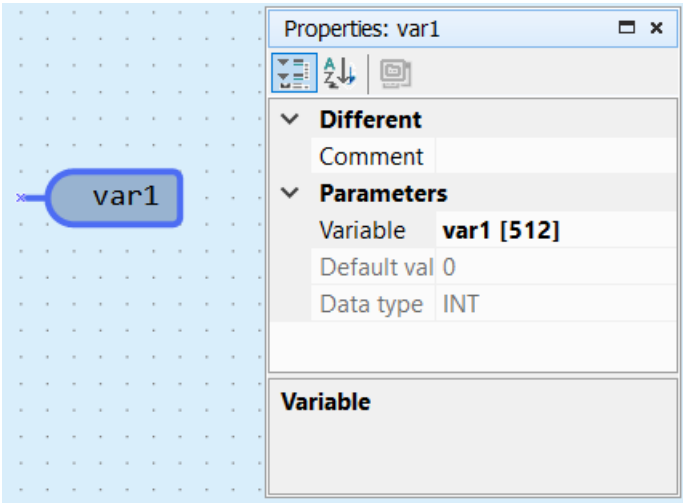
–  – network output variables are the variables that can be read via the network.

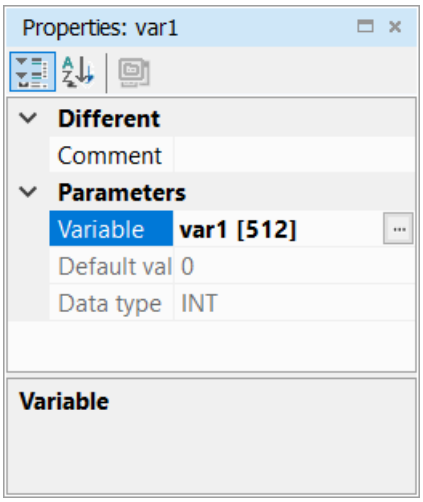
–  – network input variables are the variables that can be written via the network.

Note: A variable cannot be assigned to the block if there are no communication interfaces in the device configuration.

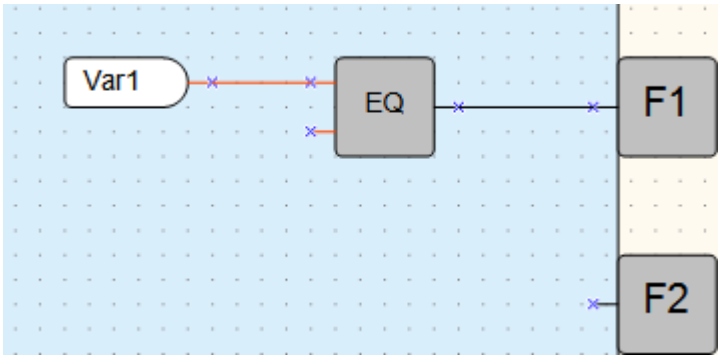
To add a network variable to the program:

1. Click the icon  or  in the toolbar ***Insert***.
2. Click the point in the workspace to place the variable block.
3. Double-click the block or click the icon «**…**» in the row ***Variable*** in Property Box to select a variable for the block or create a new one in the opened *variable table 5*.
4. Confirm the selection with ***OK***. The variable is assigned to the block.





Only the appropriate tabs in the table are available for selection. The availability is limited by the block type.
Connect the network variable block to the desired element in the workspace.



If the variable block is highlighted in red, it means that the creation is incorrect or not completed.
The information about the error is displayed in the status bar.
It is recommended to start programming with the creation of variables in the variable table.
If the variable is used more than once in a project, all references can be followed with the item ***Show references*** in the variable block context menu.

3.3.6 Read from/Write to FB

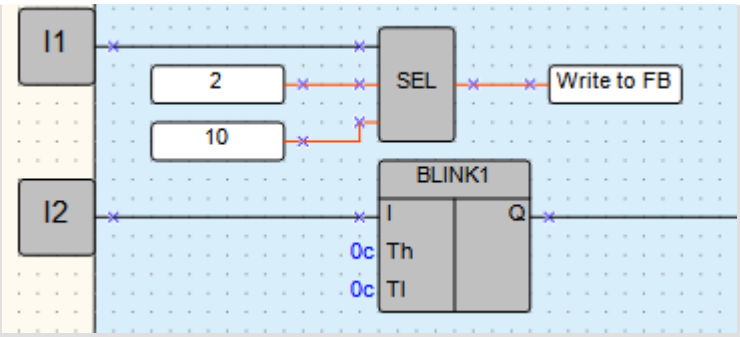
The ReadFromFB/WriteToFB blocks are used to read or write a function block's parameter's value during the program execution.
The following blocks can be added to the circuit:

– W → — to write the value to a FB;

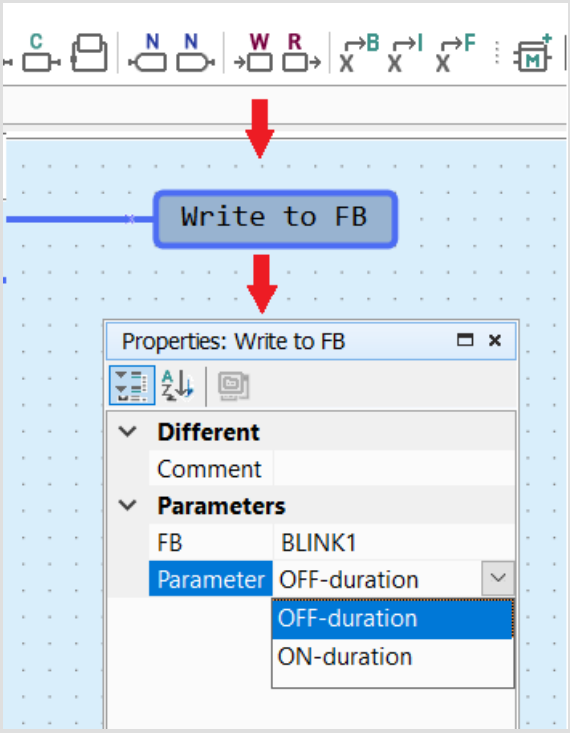
– R → — to read the value from a FB.

The block ***WriteToFB*** (W →) is used to change an FB parameter during the process.

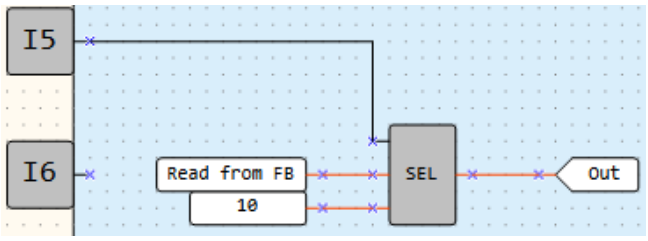
Example:
The value of the parameter ***ON-duration*** of the FB ***BLINK1*** should be 2 or 10 depending on the value at the input ***I1***.



Go to ***Property Box***, select the FB ***BLINK1*** in the row ***Function block*** and the parameter of the FB in the row ***Parameter in FB***.

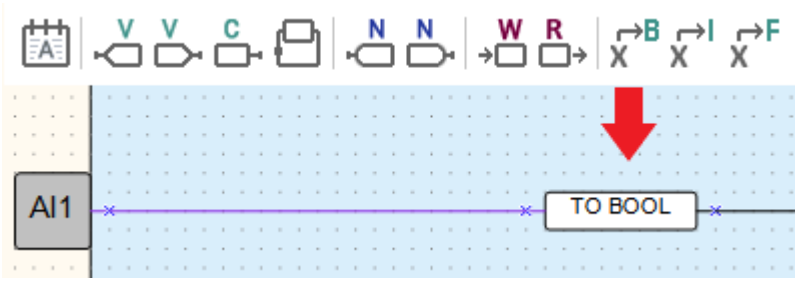


The block ***ReadFromFB*** (R) is used to read the current value of an FB parameter and use it in the program. The use is the same as in the case of the **WriteToFB** block.



3.3.7 Conversion block

A connecting line between program components can only be created for input and output of the same type: BOOL, INT, or REAL. To create a connection line between input and output of different types, conversion blocks should be used.
To add the conversion block to the program, click the corresponding icon in the toolbar ***Insert***, then click the desired place in the workspace.

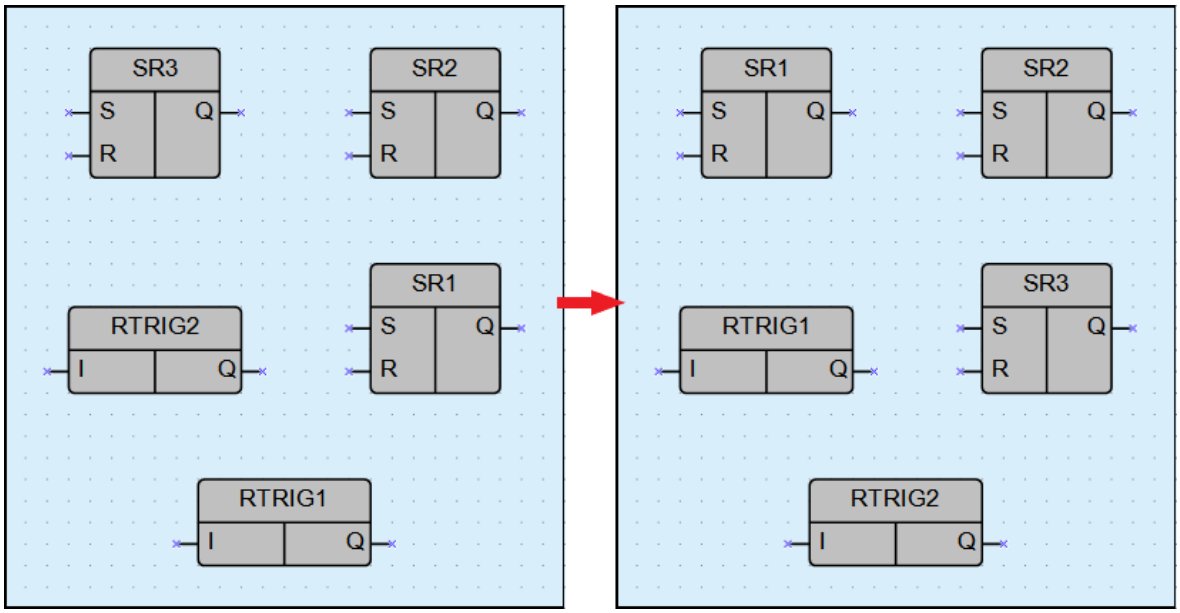


Conversion blocks:

	Conversion to BOOL	Conversion of INT or REAL to BOOL If the input value > 0, the output = 1 (True)
	Conversion to INT	Conversion of BOOL or REAL to INT REAL is rounded down to INT, negative value is converted to 0
	Conversion to REAL	Conversion of BOOL or INT to REAL

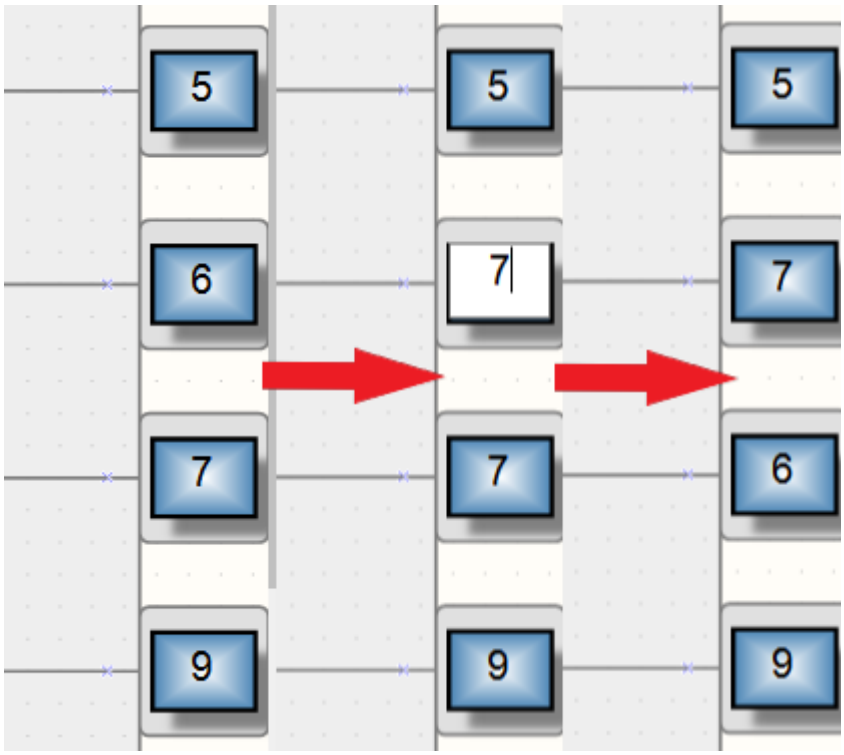
3.3.8 Arrange elements

The sequence numbers of the function blocks can be automatically reassigned by clicking the button ***Arrange elements*** in the toolbar ***Service***. The blocks of the same type are numbered sequentially from top to bottom and from left to right.



3.3.9 Execution sequence

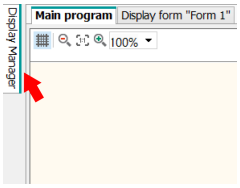
Calculation of the values for outputs and delay lines is performed in a certain order. To see this order, click the arrow near the icon Q in the toolbar ***Service*** and select ***Delay lines*** or ***Outputs***.
ALP will switch to the execution order setting mode – the sequence numbers of the execution order will be displayed near the outputs and feedbacks.
To change the order, double-click an output or a delay line and enter the desired number.



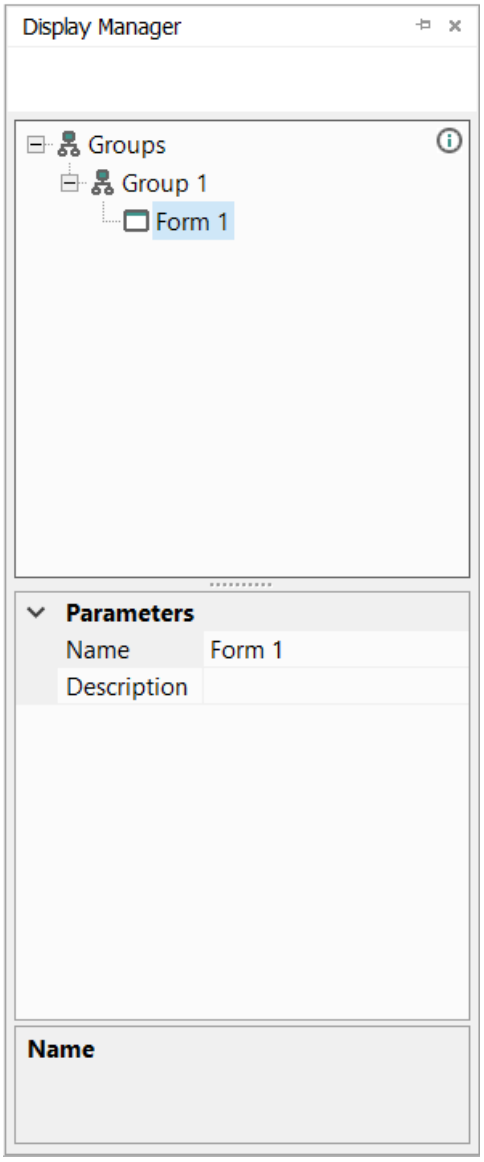
Click the icon  once more to deactivate the edit mode.

3.4 Display programming

To determine the displayed information, use the tab *Display manager 2.8* in the upper left corner of the window. *Display manager 2.8* is only for target devices with display available.

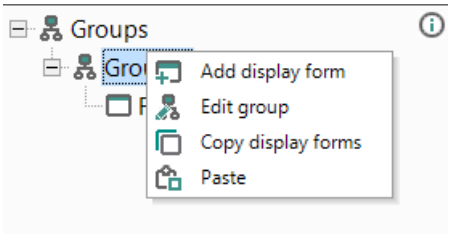


By default, the *Display manager 2.8* displays one screen.

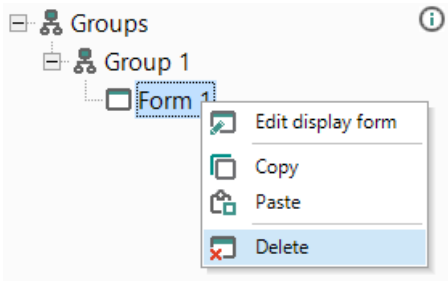


Adding display froms

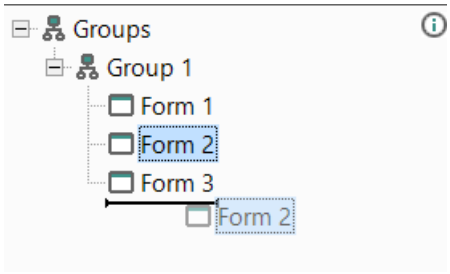
The display can be programmed using one or more display forms with “jumps” between them so that the displayed information can be changed by program events (change of variable) or by the operator (button event).
To add a display form, right-click on the **Group 1** element and select **Add display form** in the context menu.



To delete a display form, right-click on the desired form and select **Delete** in the context menu.



To change the position of the display form, drag it while holding the **Shift** key to a new location.

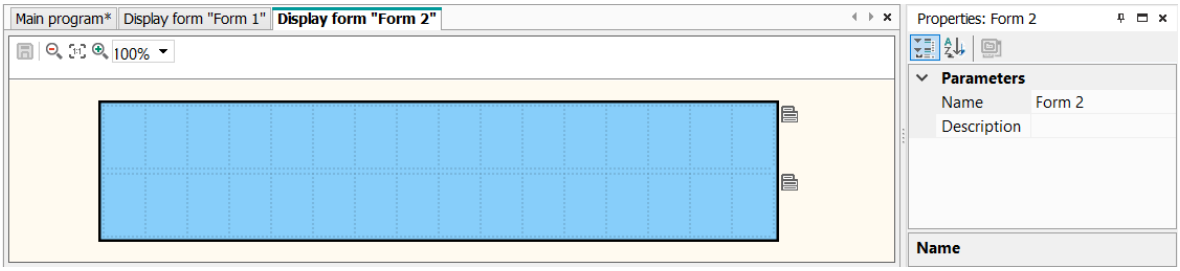


When you drag, the new position will appear as a horizontal marker.

Display form properties

To open the selected form in ***Display Editor***, use the command ***Edit display form*** in the form context menu or double-click the form in the tree.
Display form properies:

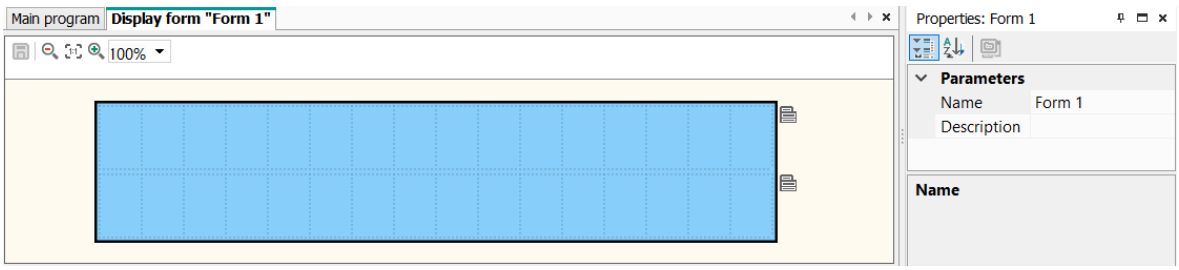
– **Name** – to be displayed in the display form manager and in the display editor header
– **Description** – text description of the display form



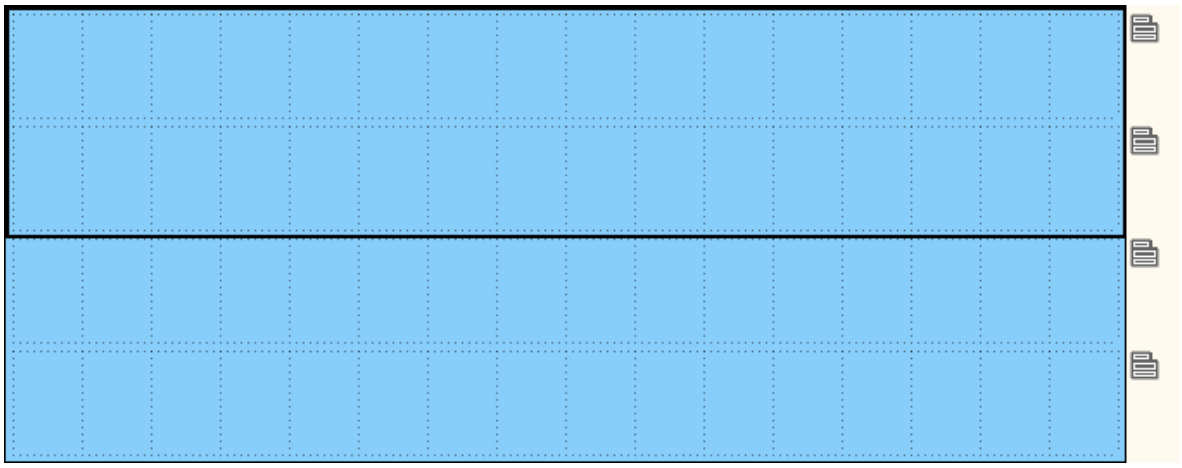
3.4.1 Monochrome text LCD

Display editor

To open the selected form in ***Display Editor***, use the command  ***Edit display form*** in the form context menu or double-click the form in the tree.



An icon  on the right edge of each row represents the row context menu, which is used to change the number and the order of the displayed rows.

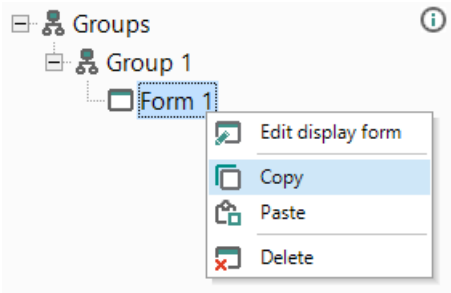


Put the *display elements 6.6* from ***Library Box*** by drag-and-drop onto the form.

NOTE
The character set is implemented within the Windows-1251 encoding.

Copy-paste display form

In the Display Manager, you can copy forms for pasting into the current or another project. To copy selected forms, select the **Copy** command in the context menu of a form or group of forms, or press the **Ctrl + C** key combination. Multiple screens can be selected by holding down the **Ctrl** or **Shift** key.



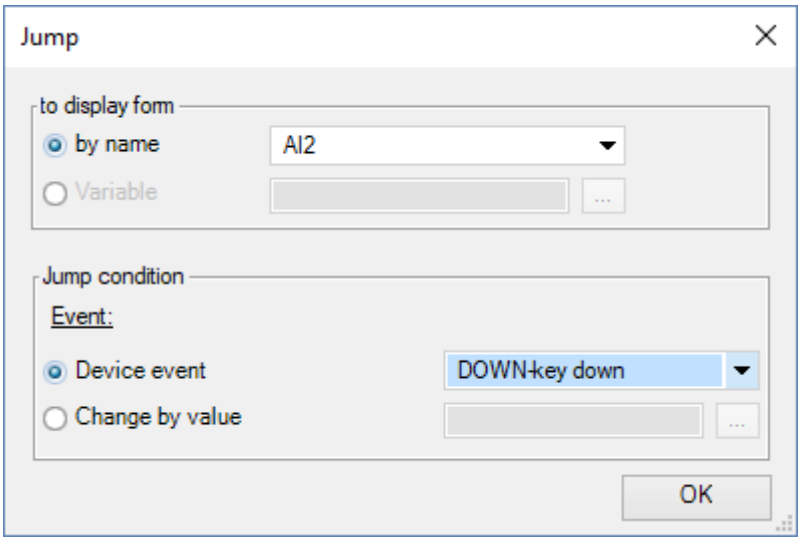
To paste copied forms, select the **Paste** command in the context menu of a form or group of forms, or press the key combination **Ctrl + V**.
All controls and screen properties placed on the form are copied along with the form. The variables associated with the form are also copied, according to the rules described in the section *Copy-paste variables 5.4*.
Jumps between the selected forms are copied as well. If only one of the forms connected by the jump is selected, the jump will be deleted during the insertion.

Jumps

If the display structure consists of more than one form, “jumps” should be defined to enable the navigation between forms.
To create a jump:

1. Right-click on the **Group 1** element in the display manager tree and select **Edit group** in the context menu. The screen group editor tab will open.
2. Select the start form in the form group editor.
3. Click the «**…**» icon in the row ***to display form*** in ***Property box***. Jump dialog window will open.

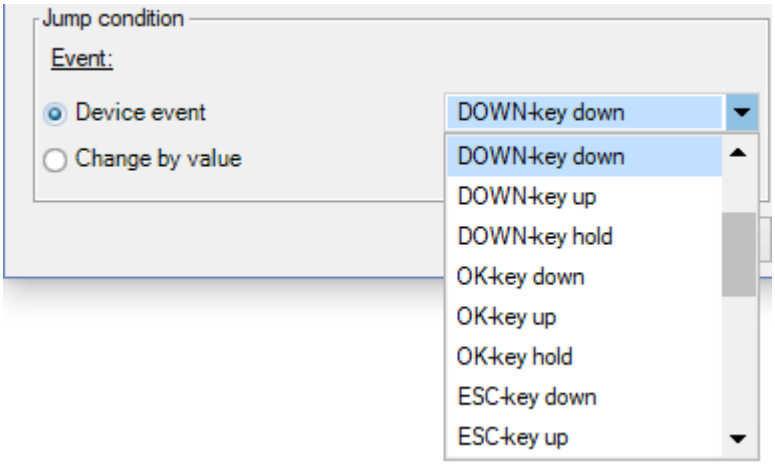
   In the drop-down menu, select another form to go to.

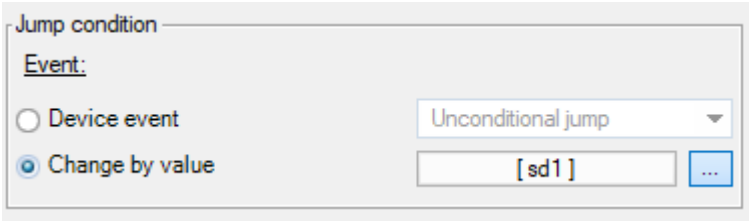


4. Select the event in the section ***Jump condition***, as device event or change of a variable by value.

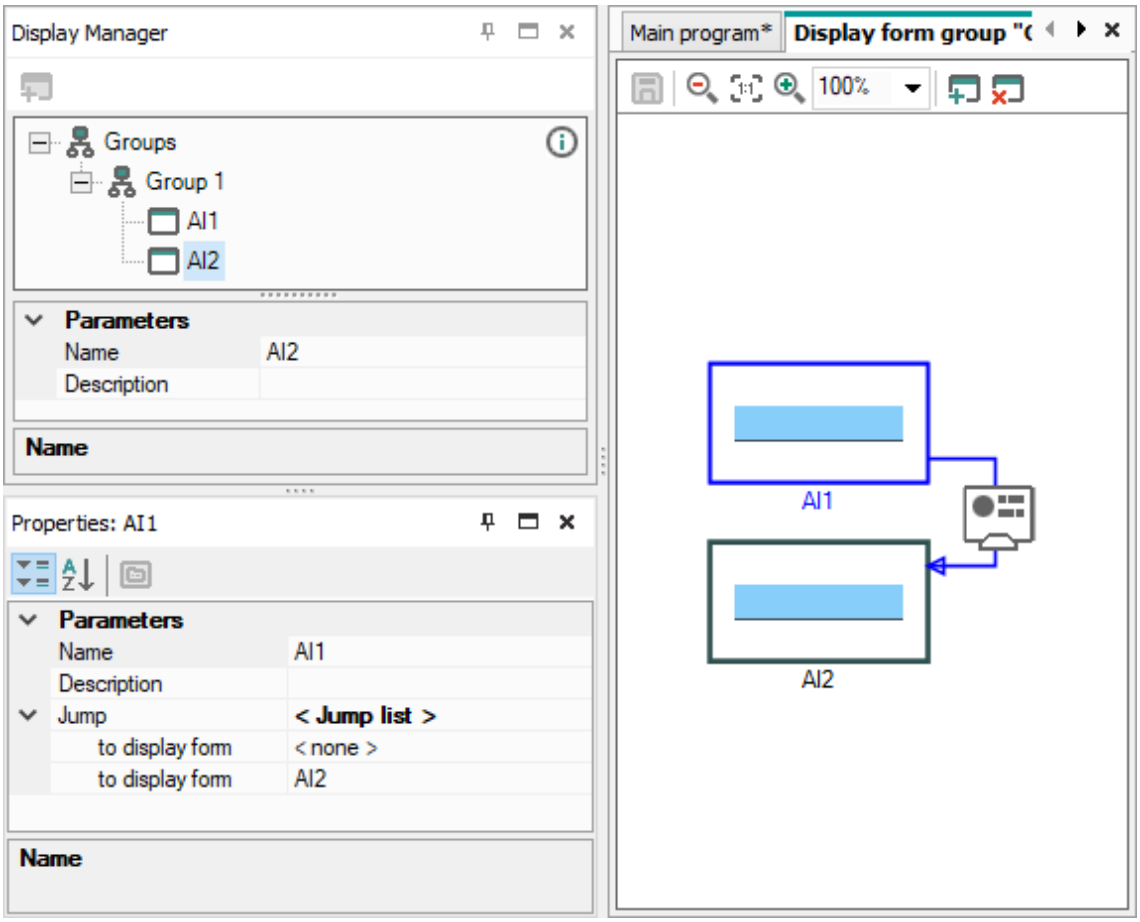
   A button event can be selected as ***Device event***.

   A BOOL variable can be selected for ***Change by value*** event.





5. Confirm with ***OK***. The created jump is shown in the structure.



The jump between two forms can occur by several events and the graphical structure can reach a very high complexity.

3.5 Simulation

Use the simulation to proove the correctness of the created program. Only the offline simulation is currently possible. The simulation enables to analyze the values of all signals within the circuit program. Change the values at the digital and analog inputs as well as of variables and constants and check the values at the outputs.

To start / stop the simulation mode, click the icon ▶ in the toolbar ***Service***→ ***Simulation***. A new toolbar ***Simulation*** is displayed.

Simulation toolbar

▶	**Start**	Start the permanent simulation
↪	**Single cycle**	Step-by-step simulation. Click the icon to execute one program cycle
⏸	**Pause**	Interrupt the simulation. Clock the icon once more time to continue the simulation
■	**Stop**	Stop simulation
	Refresh time	Input field for setting the information refresh period on the scheme in milliseconds

	Cycle time	Input fields for setting the cycle time of program execution in simulation mode. Cycle time units: milliseconds, seconds, minutes, hours
	**Watch Window**	Open/close the window to watch the variables vaues at each program step

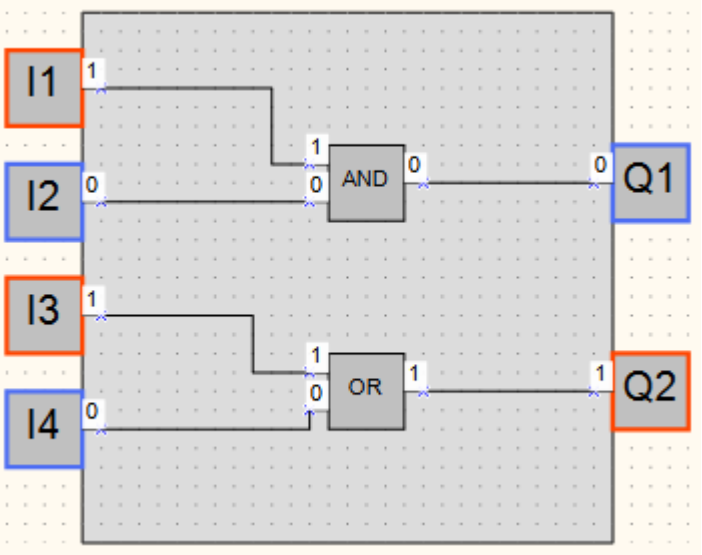
CAUTION
*The parameters **Cycle time** in ALP simulation mode and **Cycle time** 7.2 in the device are different in spite of the same name.*

Calender toolbar

An additional toolbar ***Calendar*** is displayed in simulation mode if there are FBs of type *CLOCK 6.2.2.4* or *CLOCK WEEK 6.2.2.5* in the program (available only for devices with real-time clock). It is used for simulation of such blocks.

Simulation procedure

1. Run simulation in one of the modes: real time ( ) or step-by-step ( ).
2. Set the input values on program blocks.



3. Select values of parameters ***Refresh time***, ***Cycle time*** and **Cycle unit time** for convenient simulation.
4. Exit simulation mode to correct the program.

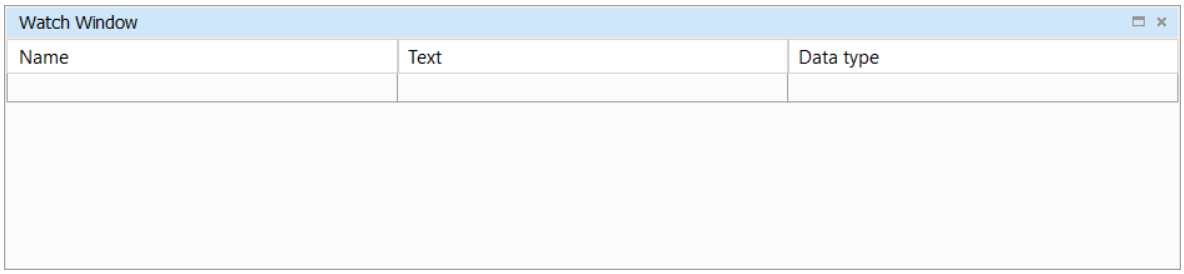
In simulation mode you can change the values of the device inputs by clicking on them. In this case a discrete input will change its Boolean value and the color, for analog inputs the value is set in the dialog window with the input field.

NOTE
Macros are excluded from simulation. Simulation for macros should be performed separately in the macro workspace.
Simulation cannot be performed for:
- *blocks without connection with device outputs or network variable output blocks*
- *incorrectly associated variables*
- *retain variables*

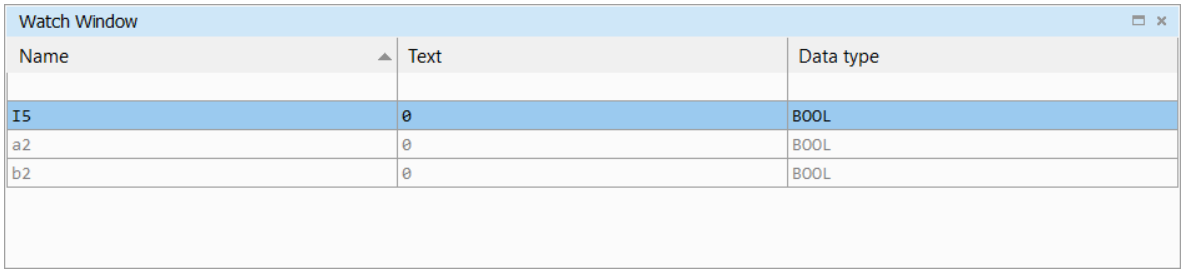
You can also specify the variable value directly on the scheme. Double-click on a variable to open a dialog with an input field for a new value. The value of the network variables can be set as well.

Watch window

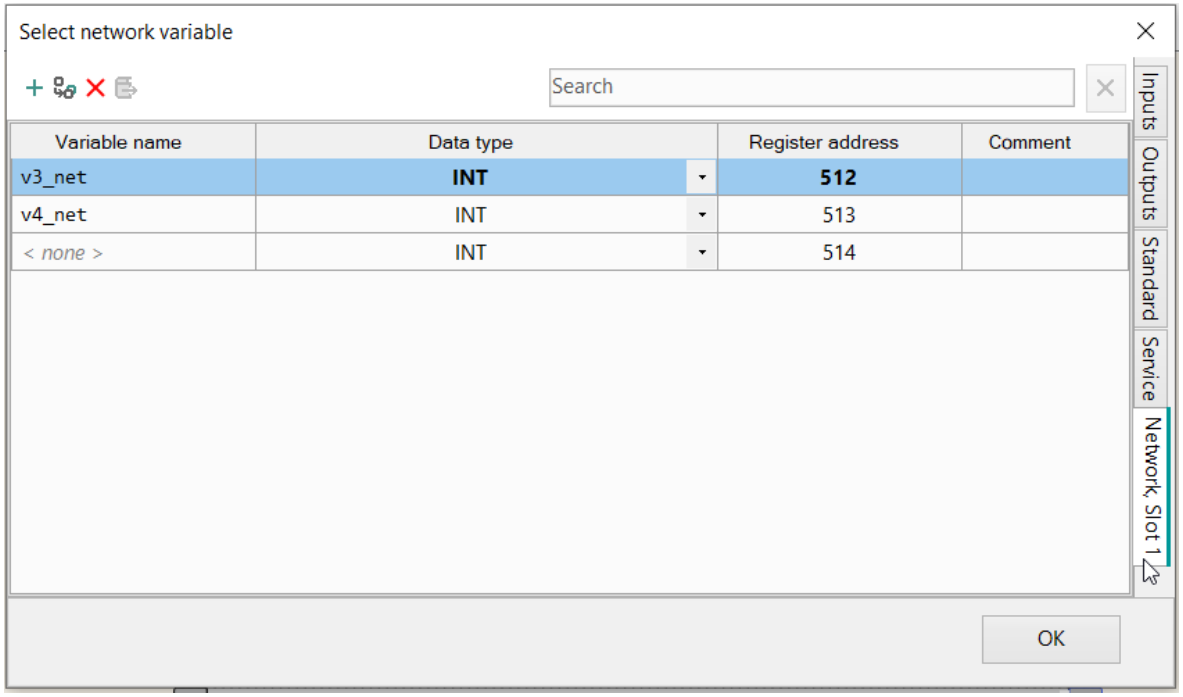
Click icon on the simulation toolbar to watch the input, output or variable values at every program step.



To add an input, an output or a variable to the ***Watch list*** , click the empty field in the ***Name*** column and then click the «**...**» icon appeared to the left.

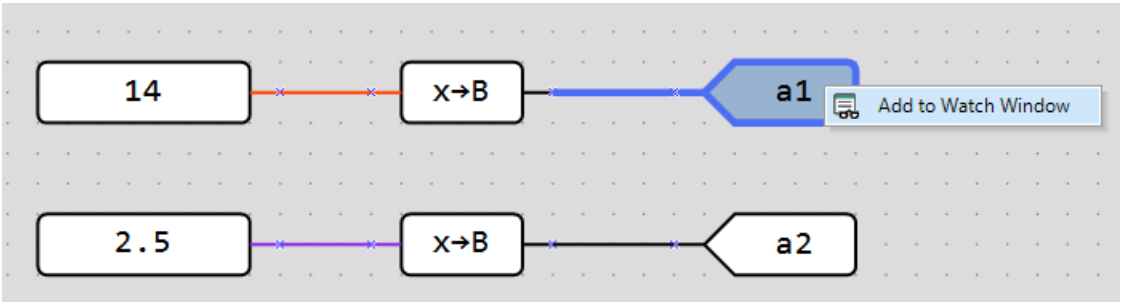


The ***Variable table*** will open. Here project variables, inputs and outputs can be selected.



The selected variables are added to the preview window.

The block context menu can also be used.



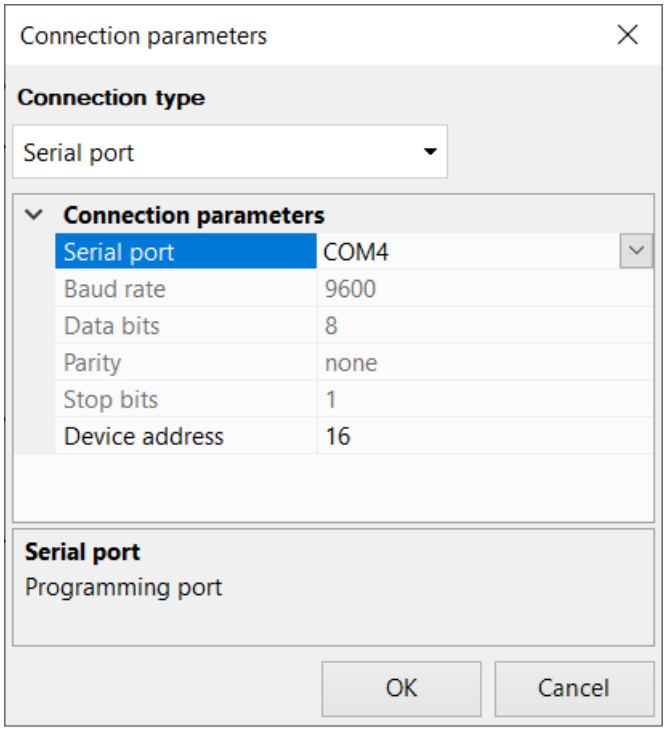
The values of the variables, inputs and outputs can be set in the ***Value*** column during simulation.

3.6 Connection to device

CAUTION
The device must be powered off before connecting to PC.

All devices can be connected to PC over USB. If the device has an Ethernet interface, it can be connected over Ethernet. To temporarily interrupt the connection, use ***OFFLINE mode***.

Connection over USB

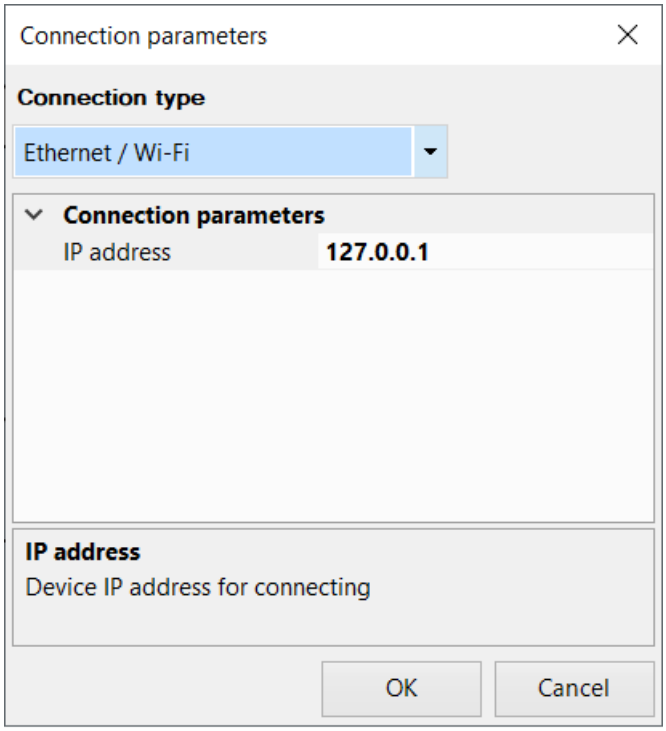


Devices can be connected to PC over USB. The required connection cable for PR200. However, connection cable is not included for devices that use Micro-USB port.

1. Connect the device to a USB port of the PC and switch the device on.
2. Start ALP and select the menu item ***Device > Port settings***.
3. Select ***Serial port*** for ***Connection type***.
4. Select the serial port in the opened dialog. The number of the emulated COM port can be found in the Windows Device Manager under “Connections (COM and LPT)”.
5. Enter the ***Device address*** (16 by default) and confirm with ***OK***. All other parameters are displayed only for your information.

If the connection is established, the information about the connected device and the serial port is shown in the status indicators.

Connection over Ethernet



To connect the device to a PC via Ethernet interface or Wi-Fi, consider the following steps:

1. Connect the device to the same local network as the PC.
2. Find out the IP address of the connected device. The default IP address is specified in the device's User Manual. The current IP address of the device can be read using the software.
3. Select ***Ethernet / Wi-Fi*** for ***Connection type***.
4. Enter the ***IP address*** of the connected device and confirm with ***OK***.

If the connection is established, the information about the connected device and the serial port is shown in the status indicators.

3.7 Upload project to device

Upload project to device

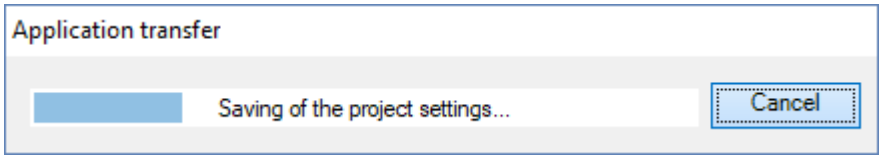
CAUTION
When a new project is uploaded to the device, the program already stored in the device memory (ROM) will be replaced by the new one.

Proceed as follows:

1. *Connect* the device to PC.
2. Power on the device.

3. Adjust the port settings if necessary.
4. Upload project to the device.

   The project can be uploaded to the device using the menu item ***File → Transfer application to device*** or clicking the icon in the toolbar. When the upload is completed, the device can be powered off and disconnected from the PC.



If the target device does not match the connected device, a warning message will be displayed.

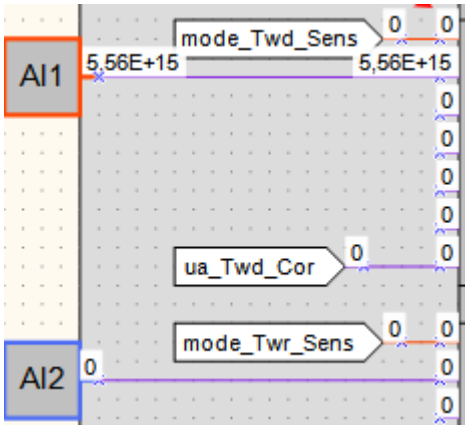
NOTE
When the program transfer is completed, the device goes to the operating mode and the program starts automatically.

OFFLINE mode

In the ***OFFLINE*** mode, the connection between ALP and the device is interrupted.
The mode is helpful when you work with two ALP instances running on PC and trying to communicate with the same device. Both applications will alternately occupy the port and the connection to the device will be constantly interrupted.
The ALP instance that is not to interact with the device has to be set to OFFLINE mode.
OFFLINE mode can be activated / deactivated using the menu item ***Service→ OFFLINE mode*** or by clicking the *status indicator 2.7* ***Device***. With the next click OFFLINE mode deactivated.

3.8 Online debugging

To start the online debugging, click the toolbar icon .
In this mode the current values of all program variables including functions, function blocks, macros, inputs and outputs are read out from the connected device and displayed in the workspace. This way you can check the logic of the device program.



The online debugging is possible only if:
– the device is connected to the PC
– the program in the device and the program opened in ALP is the same
– the version of the device firmware is compatible with the current version of ALP

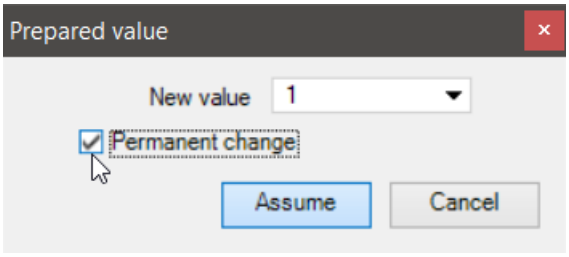
The online debugging is only available for the main program workspace, not within macros.

It is not possible to make changes in the project during online debugging. If you want to modify the project, exit online debugging by clicking the  icon once more.

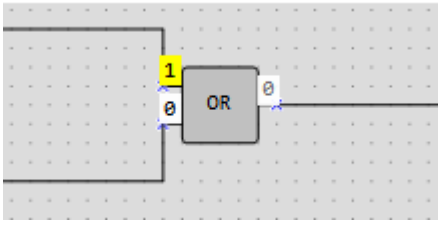
NOTE
If communication with the device is lost, online debugging is terminated after 10 seconds and the device is switched to operating mode. If the connection is restored within 10 seconds, online debugging continues, but the entered values are reset.

Manual value entry

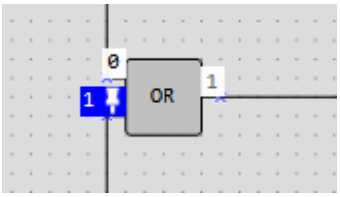
In the online debugging mode, it is possible to set the input values of functions, function blocks and macros manually by clicking on the displayed value. The new value should be entered in the field ***New value*** in the opened dialog ***Prepared value***. There are two options to change the value: one-off or permanent change.



The one-off change is active when the option ***Permanent change*** is unchecked. This enables to change the block input value for one program cycle. In the subsequent cycle, the signal from one of the device inputs or the output of another program block connected to this input is applied. The option is useful for single pulse simulation.
The new value in the workspace is highlighted in yellow during its validity.



When the option ***Permanent change*** is checked, the entered value is applied to the input until it is changed or the online debugging is stopped. The permanent value in the workspace is highlighted in blue with a white pin.



Troubleshooting

If the connection with the device is lost, the online debugging mode will be reset after 10 seconds, and the device will go into operating mode. If you manage to restore the connection, online debugging will continue, but the recorded values will be reset.

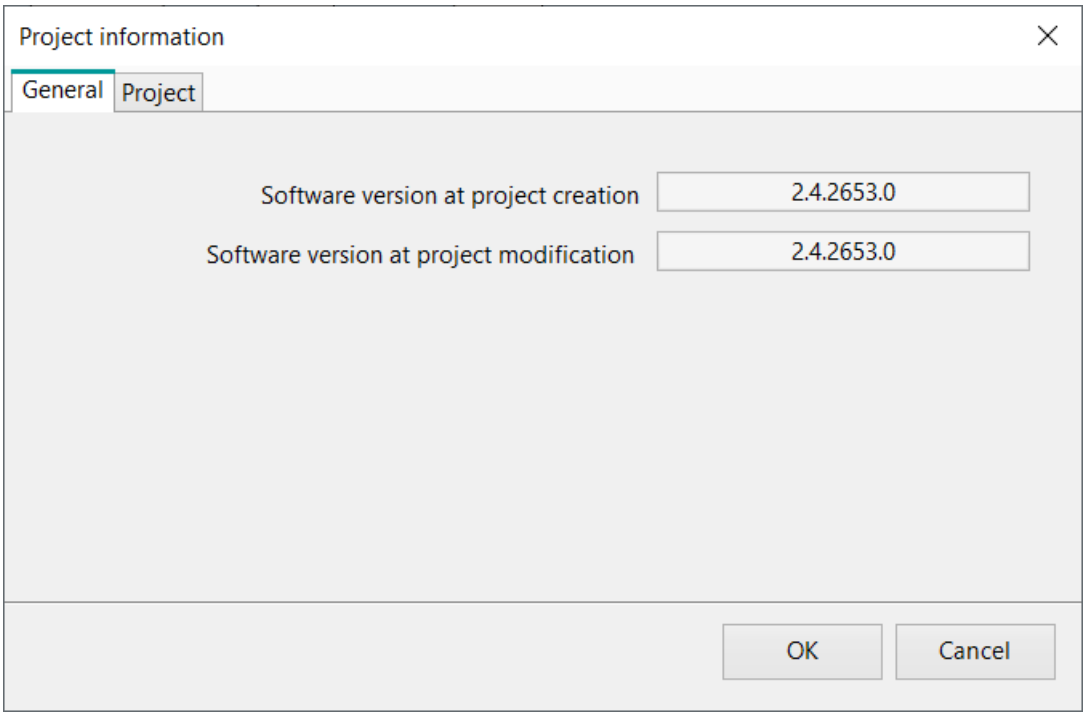
NOTE
For each modification of the device there is a limit on the transmitted values in online debugging mode. If the diagram displays empty cells of values, then a limitation is triggered, and you should increase the scale of the diagram so that fewer values fall into the "visible window". Fixed values remain frozen if they do not fall into the "visible window", but reduce the limit of transferred values because they occupy memory areas.

3.9 Project information

Use the menu item ***File*** → ***Project information*** to view and modify the information about the program.

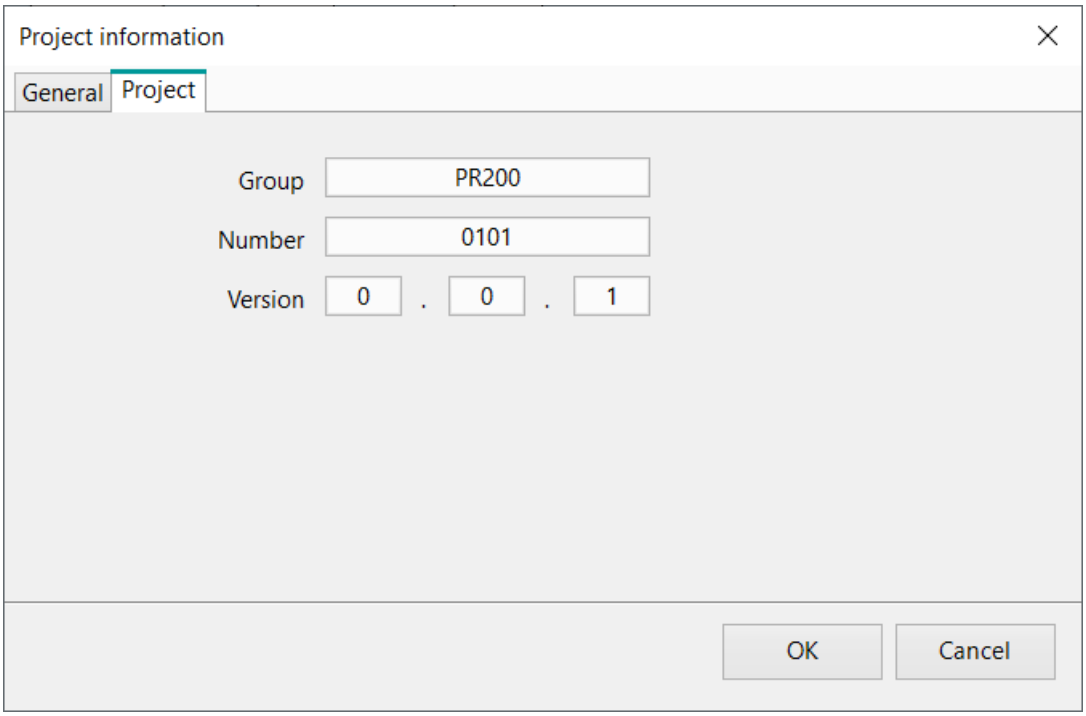
General

The tab ***General*** contains the information about the software version.
Software version at project creation – the version of the software in which the project has been created.
Software version at project modification – the version of the software in which the project has been modified.



Project

The tab ***Project*** is not available for each device. In the tab you can specify information about the group, number and version of the program to be displayed in the ***Device information*** window of the connected device after the project is saved to it.



– ***Group*** –project group name
– ***Number*** – project number within the group
– ***Version*** – project version

Click ***OK*** to save the information in the project, or click ***Cancel*** to discard it.

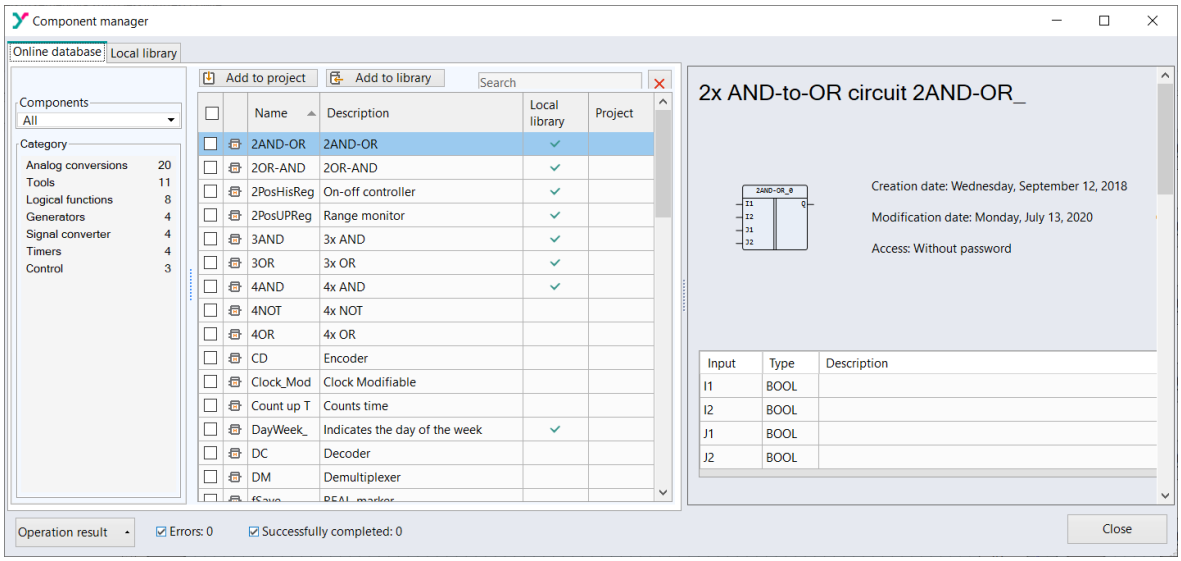
3.10 Component manager

New macros and device templates can be downloaded from akYtec Online Database. Component Manager is the tool for all interactions with this database. The internet access is necessary for this interaction.
Select the menu item ***File*** → ***Component manager*** to open it in a separate window.

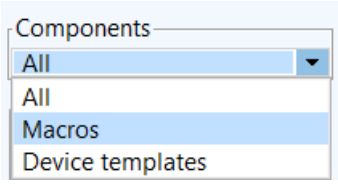
Online Database

– ***Add to project*** button – the selected blocks (macros or device templates) from Online Database are added to the project library. The blocks are then stored in the project file and can be viewed in *Library Box 2.3* in the ***Project Macros*** area.
– ***Add to library*** button – the selected blocks from Online Database are downloaded to the local library and can then be used offline.

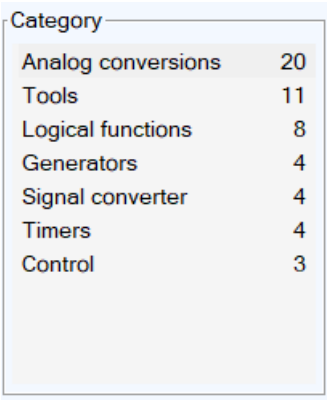
A check mark in the column ***Project*** or ***Library*** indicates that the block has been successfully downloaded (added).



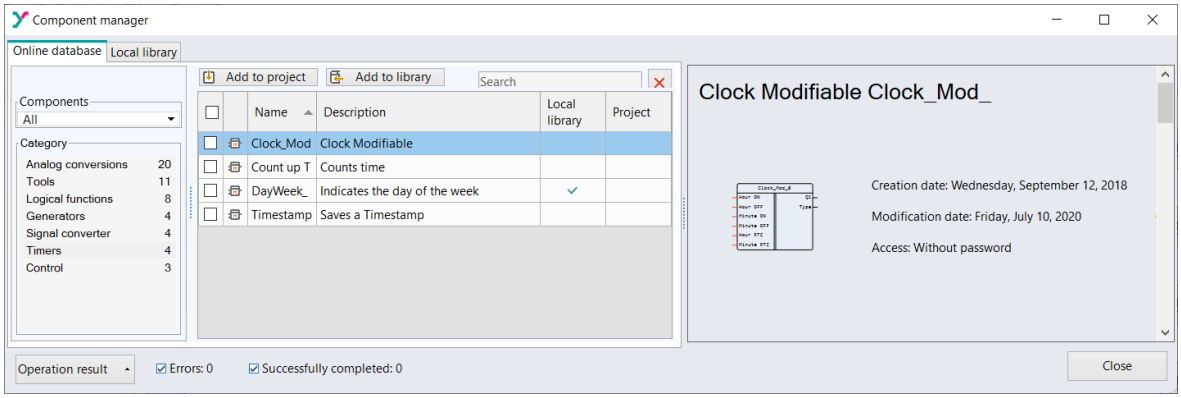
The ***Components*** drop-down menu allows you to filter the list by type:



Macros are further divided into categories depending on their purpose:



The brief description of the selected block is displayed in the upper right field, and the full description in the lower right field. The full description is a PDF document. Scroll the document to the end to see the PDF toolbar. Using it, you can download the document or print it.



Click the button ***Operation result*** at the bottom of the window to view the program messages about the performed operations.

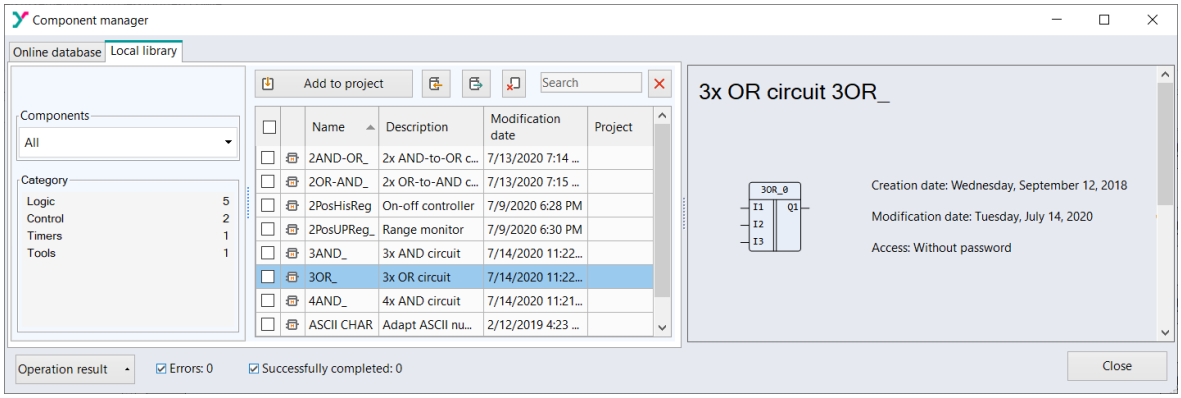
Local library

– ***Add to project*** button – the selected blocks (macros or device templates) from Online Database are added to the project library. The blocks are saved in the project file and can be viewed in the Library Box (sect. 2.4) in the ***Project Macros*** (sect. 6.3) area.

– – the selected blocks are added from a file in the project library

– – the selected block from the project library is saved as a file under the specified path for further use

– – the selected blocks are removed from the local library



NOTE
Library files are stored at the local address:
***C: \ Users \ [username] \ Documents \ akYtec ALP \ Library ***

3.11 Macro development

Macro is a user function block opened in a separate workspace. A macro can be created in the project in two ways:
Basic functions with macros:

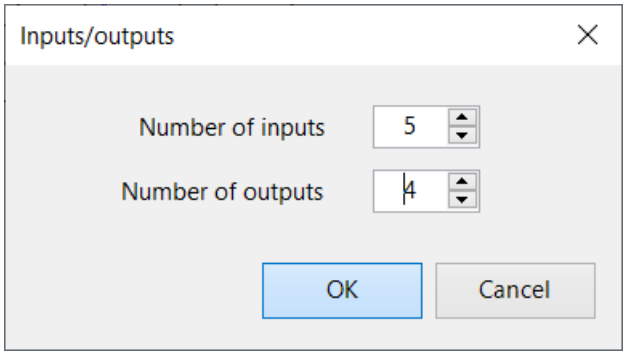
– using the main menu item ***File → New macro***

– drawing a rectangle around several blocks I the main workspace to select them and clicking the item ***New macro*** from the workspace context menu.

New macro using main menu

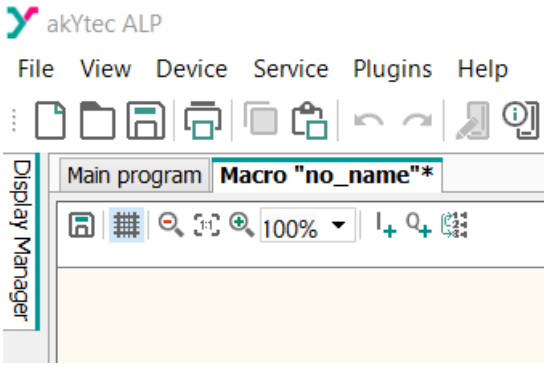
To create a new macro:

1. Select the item ***File→ New macro*** in the main menu. Then specify the number of inputs and outputs in the opened dialog window:

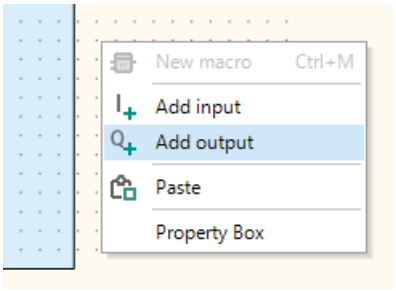


NOTE
The number of inputs and outputs can be changed after creating a macro.

2. Develop the macro algorithm in the **Macro Editor** tab, similar to developing the program in the diagram.

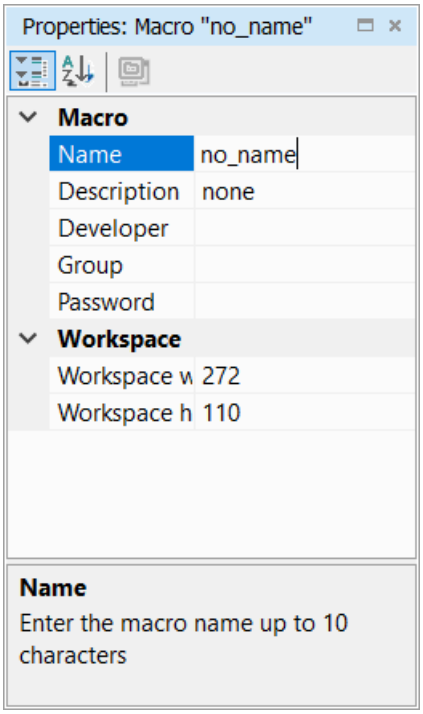


The number of inputs and outputs can be always changed. To add a new input or output, use the items  or  in the toolbar or in the workspace context menu.



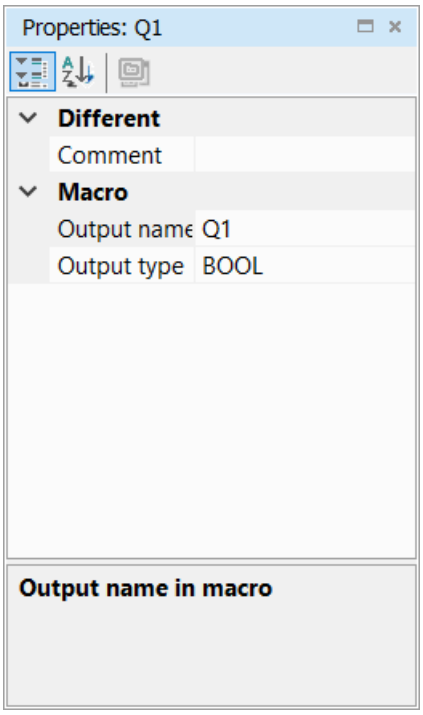
To remove an input or an output, use ***Remove*** in its context menu.

3. In Property Box, give a name, a description and a group to the macro:

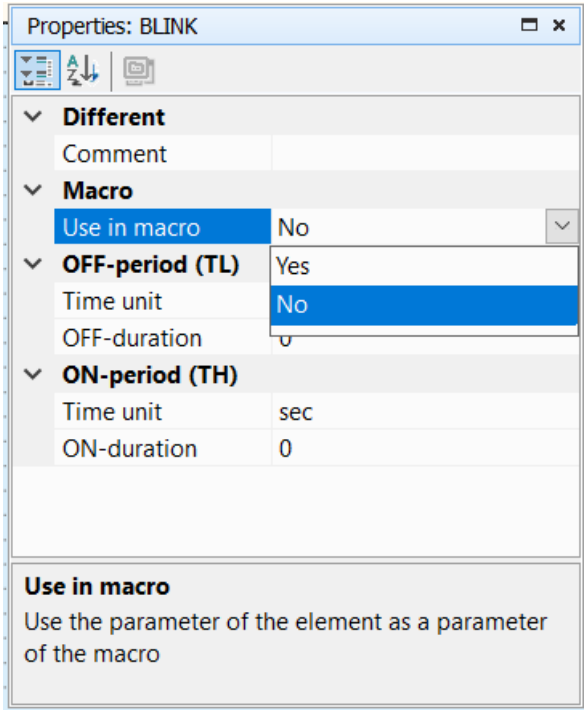


The name is displayed in the workspace tab header and in Library Box.

4. In Property Box, the name and the data type can be changed for each input and output.

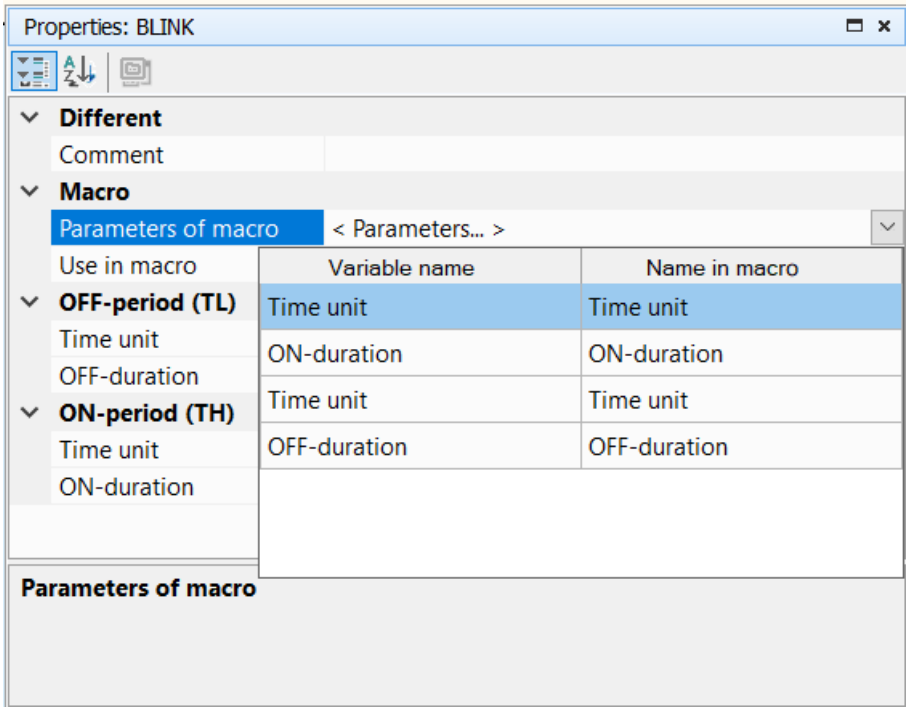


5. Next, you can set the visibility of the FB parameters used in the macro in the main program.

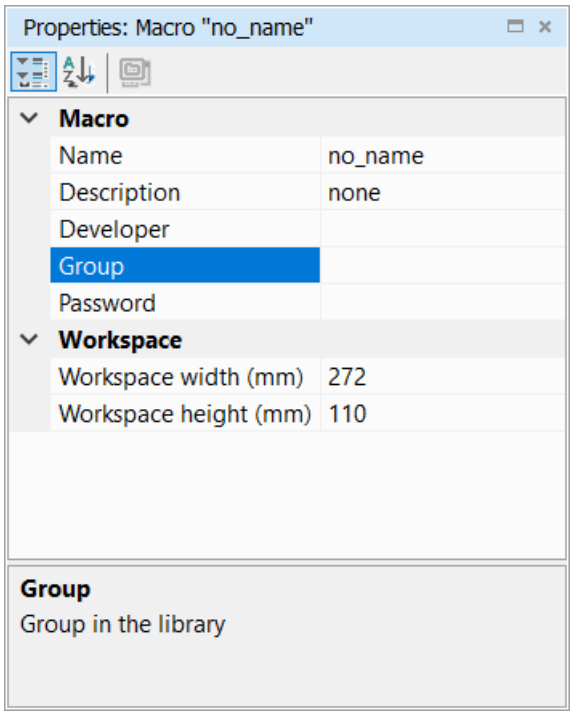


If the parameter ***Use in macro*** is set to ***Yes***, the FB parameters became parameters of the macro and a new option ***Parameters of macro*** is added to the macro in Property Box.

It is a list of names of the FB parameters, where the user can specify the name for each FB parameter in the macro for use in the main program. If you want the parameter names in the macro to be different from those in FB, click on the ***<Parameters...>*** line to edit the parameter names.



6. The macro can be *simulated 3.5* in the same way as the main program.
7. Before saving the macro, you can fill in the following fields: ***Name***, ***Description***, ***Developer***, ***Group*** and ***Password***.



It is recommended to select a short and clear name. The text in the parameter ***Description*** is displayed in Library Box under the macro name and in a tooltip, when the mouse cursor is over the macro in the main workspace.

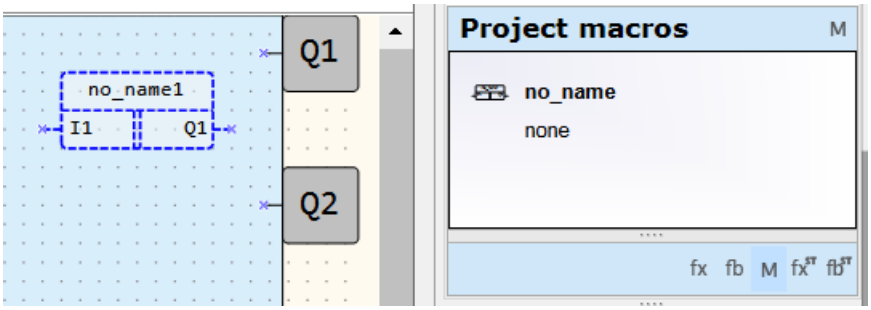
If you set the password for the macro, it will be asked every time the menu item Edit macro is selected. Otherwise, editing of the macro is available to everyone.

The name in the parameter ***Group*** is used in the project library. If the group name is empty, the macro is assigned to the group ***Other*** in the library.

The macro can be saved by selecting ***File→ Save macro as…*** or by clicking the icon in the Macro Editor toolbar.

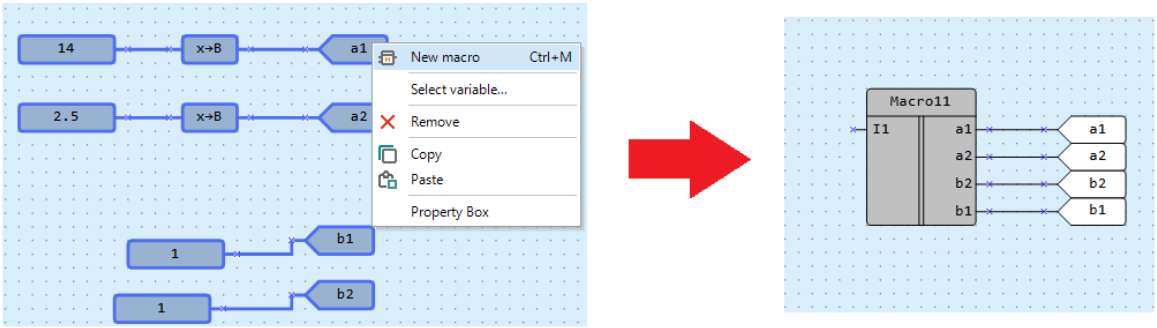
Saved macro is available only for an open project, to use the macro in other project it must be exported in a file and then imported in an other project.

8. Select section ***Project macros*** in the ***Library Box*** and drag it to the workspace.



New macro using context menu

You can create a macro by drawing a selection rectangle in the workspace and using the item ***New macro*** in the workspace context menu. All selected blocks will be moved into the new macro block that will replace the selected blocks in the main workspace. All external connecting lines will be retained.



There are some specific aspects of creating macros using context menu:

1. The number of inputs and outputs of the macro is equal to the number of connected input and output connections in the selected area. In case that blocks without connections are selected, the macro with one input and one output will be created.
2. If a standard variable block is selected, the variable will be copied under the same name into the macro.

Note: The variables in the macro and in the main program are different in spite of the same name, there is no conflict between them.

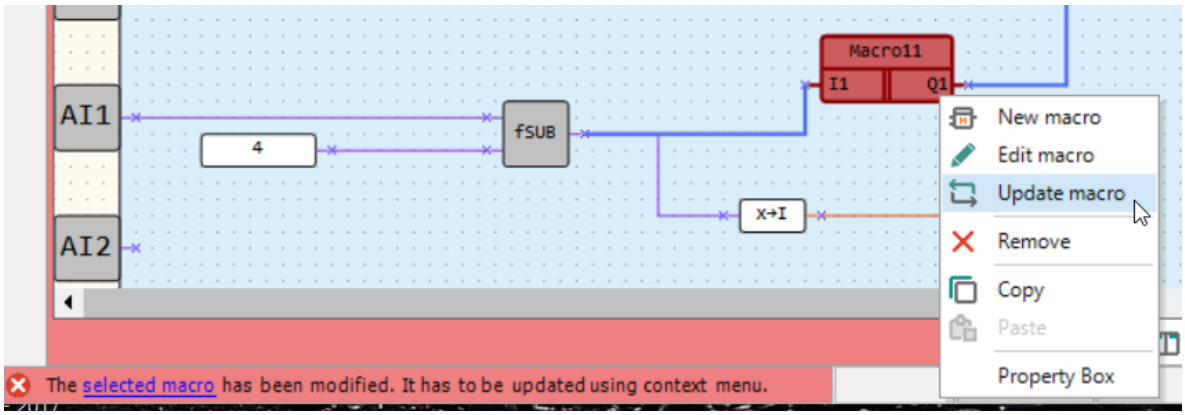
1. If all blocks of a variable are selected and it has no other references in the program, the variable will be moved into the macro.
2. If the selected variable is used (has blocks or other references) outside the selected area, it will be copied under the same name into the macro and the original will remain in the workspace.
3. If only one block of the input or output variable is selected, the variable will be copied under the same name into the macro and the original will remain in the workspace.
4. If the macro is created using the context menu, the following blocks will not be included in it:
   - device inputs and outputs
   - service variables
   - network variables
   - PID controller

In case the above-mentioned blocks are selected, they will remain in the main workspace and will be connected to the corresponding I/O points of the macro.

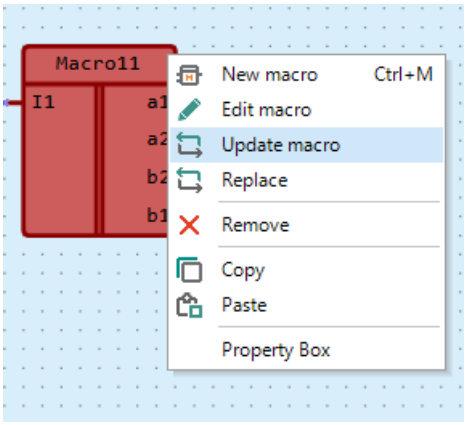
5. If any ***WriteToFB*** / ***ReadFromFB*** blocks (sect. 7.6) are assigned to the selected FB, they will be included to the macro, even if they are not selected. If the read/write blocks are selected but not the assigned FB, they will not be included in the macro.

Update macro

If the macro used in the main program has been modified, (name, type, number of I/O points, elements or the parameter ***Use in macro*** of any FB), it will be highlighted in red in the main program and the user will be prompted to update the macro. The macro is considered to be modified when the changes made in Macro Editor are saved.



To update the macro, use its context menu.



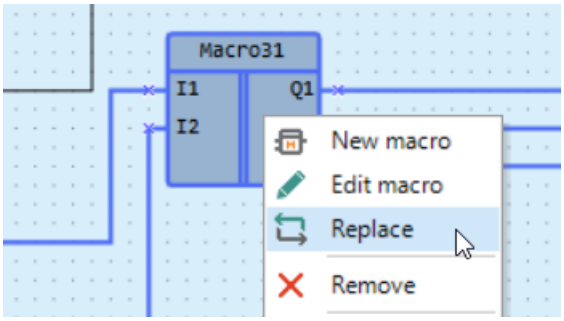
Once the macro has been updated in the main program, the next modified macro will be prompted to update.
Update rules:

– If the type or name of the macro I/O point with the attached connection is changed, the connection will be disconnected after the update.
– If I/O points are added to the macro, the existing I/O points will not be disconnected after the update.
– Macro I/O points are identified by name and type. If you change the name or type of an I/O point with an external connection and create a new I/O point with the same name and type, the connection will be automatically linked to the new I/O point after the macro update.

Replace macro

If a macro should be replaced with another one, it can be done manually: delete the macro, add a new macro and restore the connecting lines.

It is more efficient to replace the macro using the context menu command ***Replace***. The connecting lines to macro I/O points will be retained if the names and the data types of the old and the new I/O points are the same.
If the name or data type of an I/O point does not match, the connecting line will be cut and should be repaired manually.



FB in macro

If an FB is used in the macro, the user can define whether the FB parameters are available (visible) in the main program as the parameters of the macro.
If the parameter ***Use in macro*** is set to ***No***, the FB parameters are visible and can be used only within the macro.
If the parameter ***Use in macro*** is set to ***Yes***, the FB parameters became parameters of the macro and a new option ***Parameters of macro*** is added to the macro in Property Box.
It is a list of names of the FB parameters, where the user can specify the name for each FB parameter in the macro for use in the main program. If you want the parameter names in the macro to be different from those in FB, click on the ***<Parameters...>*** line to edit the parameter names.

Changing I/O points order

The I/O points of the macro are placed on the sides of the macro in the order in which they were added, from top to bottom. This order can be changed.
This can be useful if you want to place logically related I/O points nearby, or if you want to insert an empty macro into the program and determine the position of its I/O points later, after developing its algorithm.
Proceed as follows:

1. Open the macro in the editor, drag and drop the I/O points into the desired order.
2. Click on the toolbar icon ***Synchronize I/O order*** to synchronize the positions of the I/O points, and then save the macro.

   Note: The synchronization does not work if the macro I/O points are not connected to other program blocks.
3. Go to the main program. The changed macro is highlighted red and it is offered to update it using its context menu (sect. 6.3.5).
4. After the update, the order of the macro I/O points in the main program will be the same as in the macro editor. The connecting lines of the macro will be retained.
5. If the synchronization switch is not activated, the macro I/O points will be displayed in the main program in the default order. This can be useful if you need to add an I/O point to the macro, but don't want to entangle the existing lines.

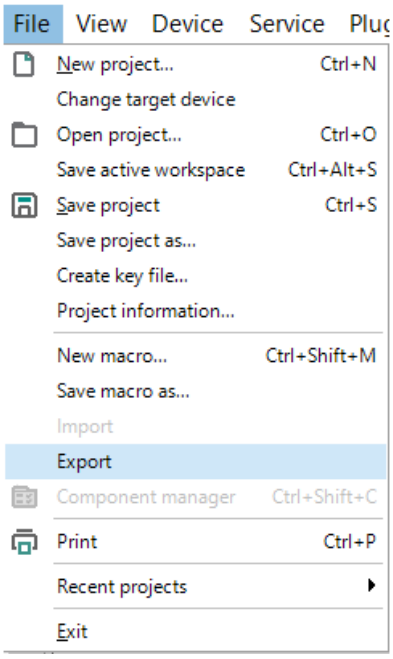
Export macro

Exporting a macro to a file is only possible when the macro editor window is open. To export a macro, select **File** → **Export** in the main menu.
To export a macro:

1. Open the macro in the editor.

If you need to edit the macro before saving, you should drag it onto the project canvas and select Edit and make changes in the macro context menu.

2. Select the main menu item **File** → **Export**.

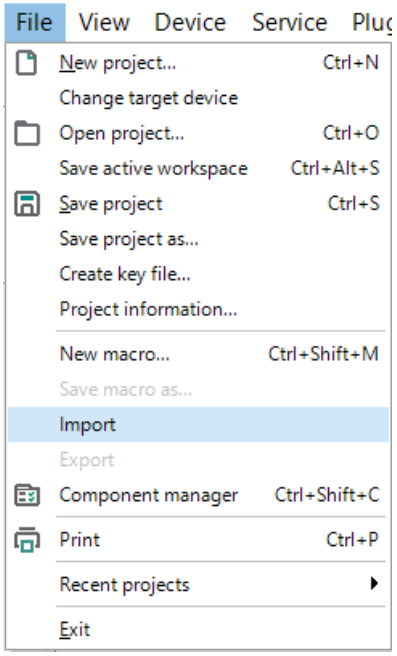


3. In the window that opens, select a location and save the macro file with the extension *.*tpl*. After saving, a message indicating that the macro was exported successfully will be displayed.

Import macro

If you need to use a macro created in another project to create a program, you can import the required macro into the project.
To import a macro use the main menu item **File** → **Import**.

NOTE
The **Import** item is active only when the focus is on the workspace.



In the window that opens, select the desired file and click the **OK** button. The macro will be added to the ***Library Box*** in the ***Project macros*** section, and can now be used in the project.

Copy macro

Macros can be copied from project to project for reuse and reduced development time. To copy a macro, select the macro block in the source project and click the  on the toolbar or select the ***Copy*** command in the block context menu. The macro is inserted into another project by clicking the  button on the toolbar or by selecting the ***Insert*** in the canvas context menu. You can also use keyboard shortcuts to copy and paste, see *Keyboard shortcuts 9*.
Once inserted, the macro will be available in the ***Project macros*** section of the ***Library Box***.

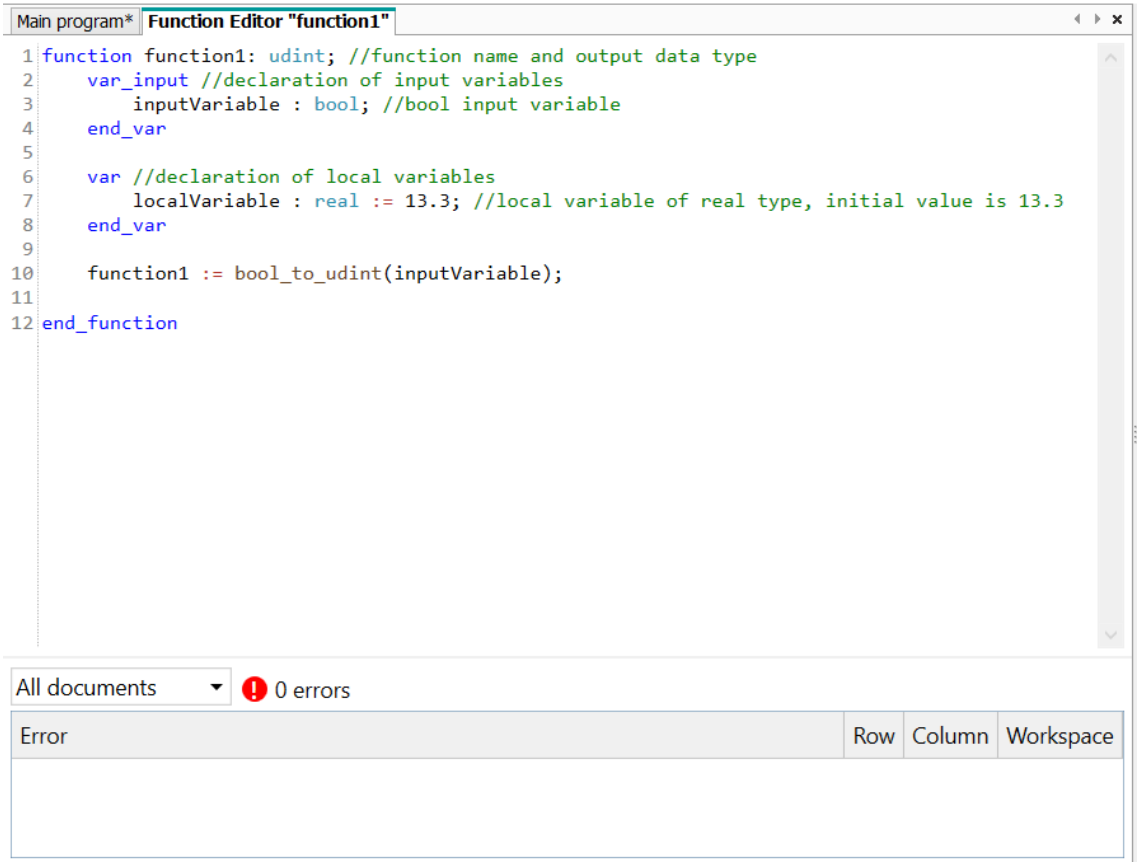
3.12 Using ST function

Creation of user functions in *ST language 11* is available for devices on the new hardware platform. If the project is created for such a device, the toolbar icon  ***New ST function*** is active.

NOTICE
ST funtions reserve space in ROM memory after they are added to the project library, regardless of whether they are used in the project or not.

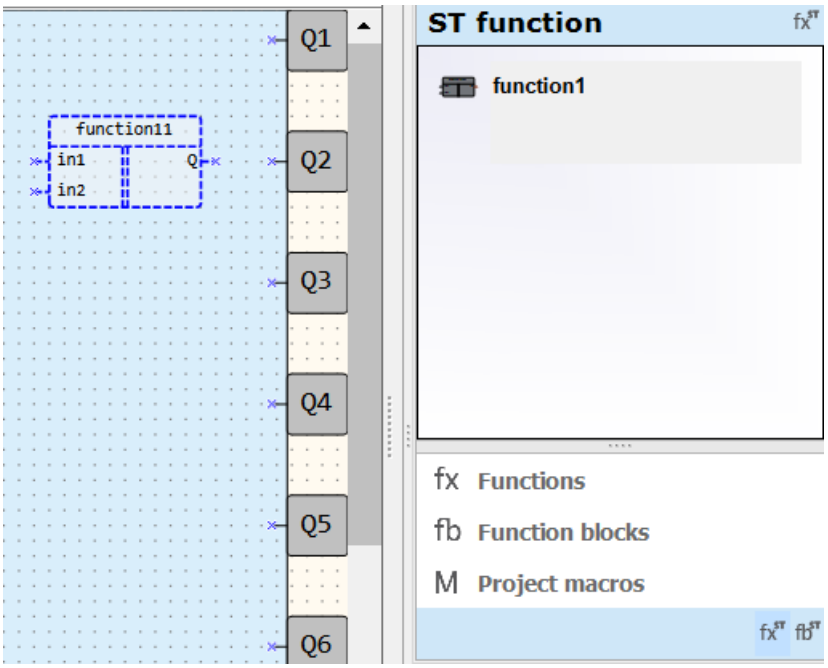
Creation of ST function

1. Click the toolbar icon  ***New ST function***. Function editor with an ST function template opens in a new workspace.

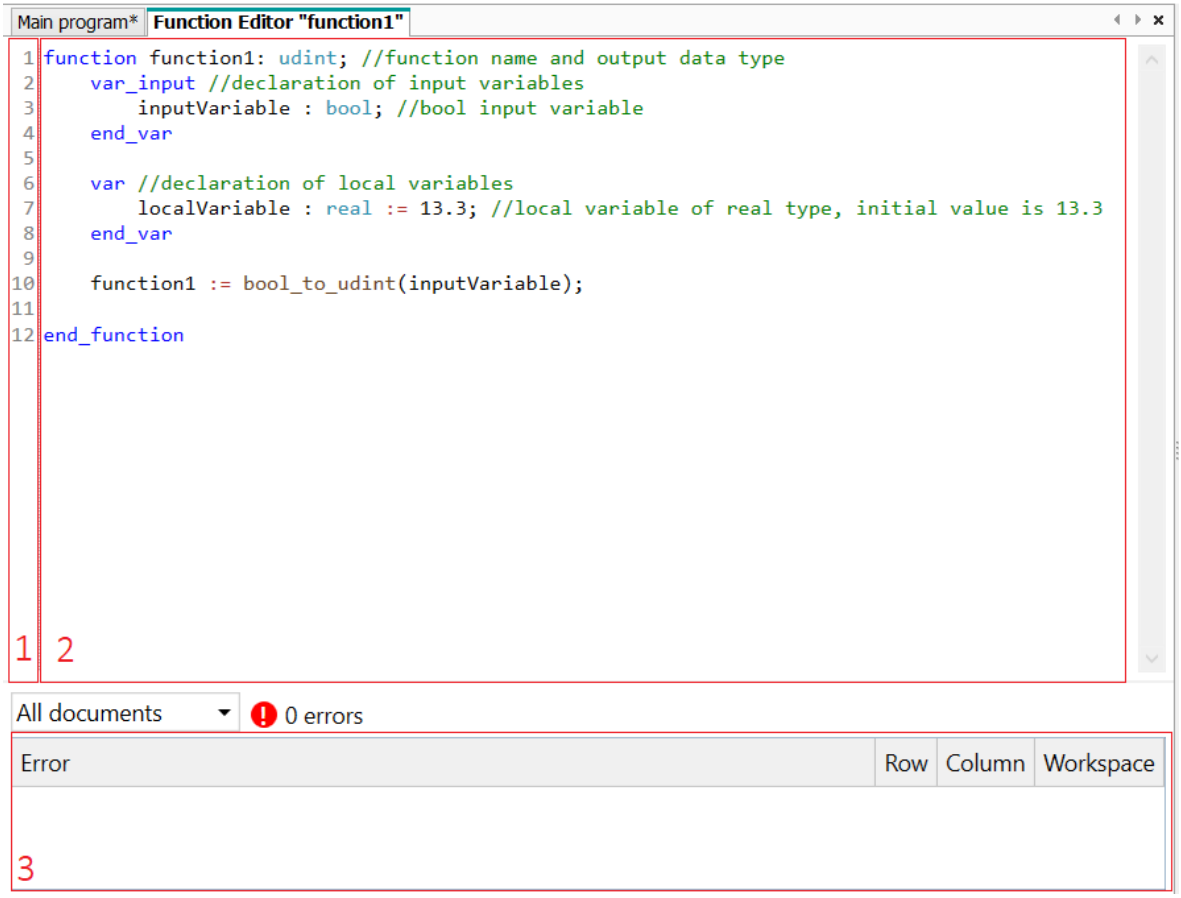


2. Specify the function name and the output data type in the first line.
3. Specify all required inputs variables in the input variable declaration block ***var_input***.
4. Specify all required local variables in the local variable declaration block ***var***.

5. Develop a function algorithm in accordance with the ST syntax rules.
6. Switch to the ***Main program*** tab or close the ***Function editor*** tab. The function will be saved automatically.
7. Select the section ***ST functions*** in ***Library Box*** and drag the saved function onto the project workspace.



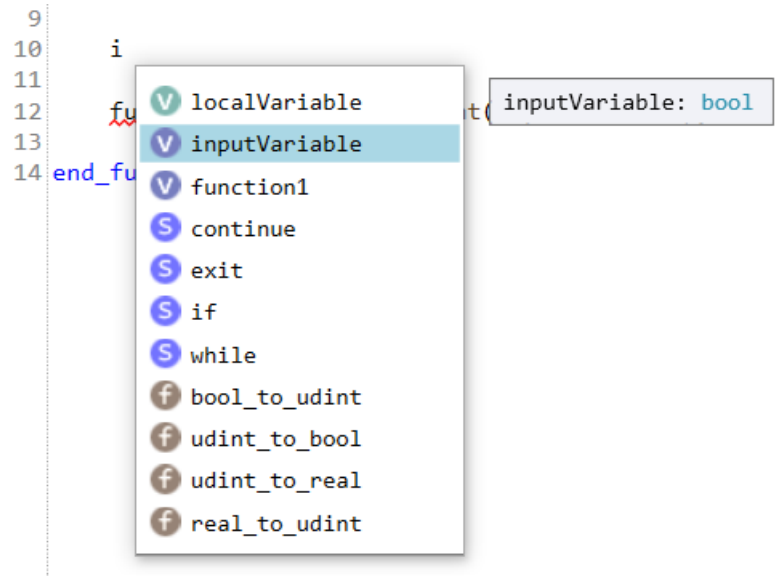
Function editor interface



1. **Line numbers** — sequential line numbers in the program code.
2. **Code editor** — code editing area with automatic syntax highlighting.
3. **Error panel** — error display area.

Snippet management

Snippet management is a text editor feature that allows easy insertion of content from a catalogue of repeatedly used text. If you enter the first character in the editor, a context menu opens with focus on the first line. Use the cursor keys to select a snippet. To insert the selected snippet into the code, press ***Enter*** or ***Tab*** or double-click on the list item.

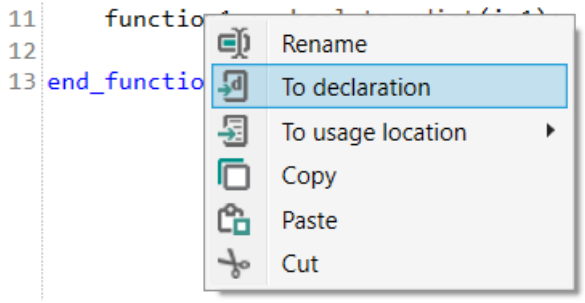


Snippet groups:
– V — local variables
– S — statements (***while***, ***for*** etc.)
– K — keywords (***true***, ***false***)
– F — built-in functions
– T — other functions
Within groups, snippets are arrange alphabetically.

Jump to declaration or usage location

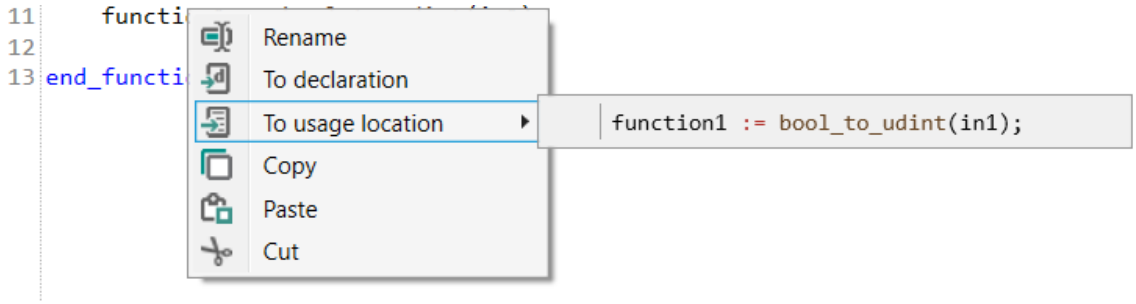
For convenient work with the code, a search is implemented to find the places in the program code where a function or variable is declared or used.
To jump to declaration:

1. Place the cursor on the name of a function or variable in the program code.
2. Right-click on the name.
3. Select ***To declaration*** in the context menu.



To jump to usage location:

1. Place the cursor on the name of a function or variable in the program code.
2. Right-click on the name.
3. Select ***To usage location*** in the context menu. A list of places in the code opens the selected function or variable is used.

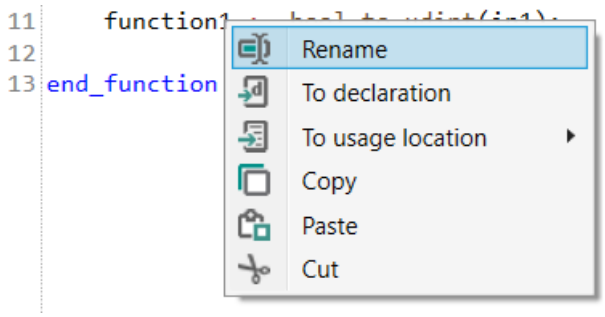


4. Left-click on the selected usage location. The cursor will move to the line where the function or variable is used.

Rename variable or function

Centralized change of the name of a variable or function throughout the code is available.
Proceed as follows:

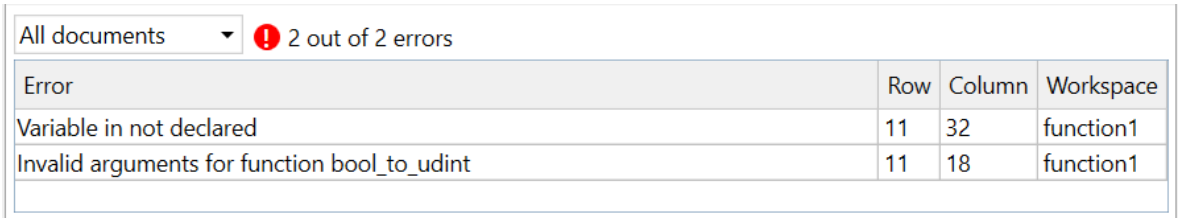
1. Place the cursor on the name of a function or variable in the program code.
2. Right-click on the name.
3. Select ***Rename*** in the context menu. The name will be marked green in all places where it is used.



4. Enter a new name in one of the green marked locations and click on another place in the code. Now the name is changed in the whole program.

Error panel

All errors occurred during the code writing, are listed In ***Error panel***. Left-click on a row in the list to jump to the error in the code.

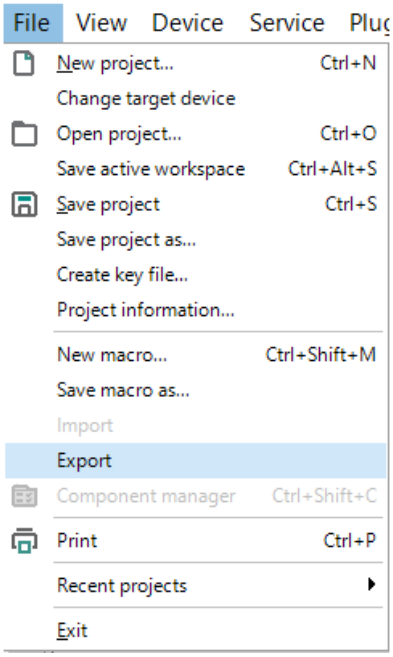


Error	Row	Column	Workspace
Variable in not declared	11	32	function1
Invalid arguments for function bool_to_udint	11	18	function1

Export ST function

Exporting a function to a file is only possible when the function editor tab is open. To export a function, select **File** → **Export** in the main menu.
To export function proceed as follows:

1. Open the function in the editor.
2. Select **File** → **Export** from the main menu.

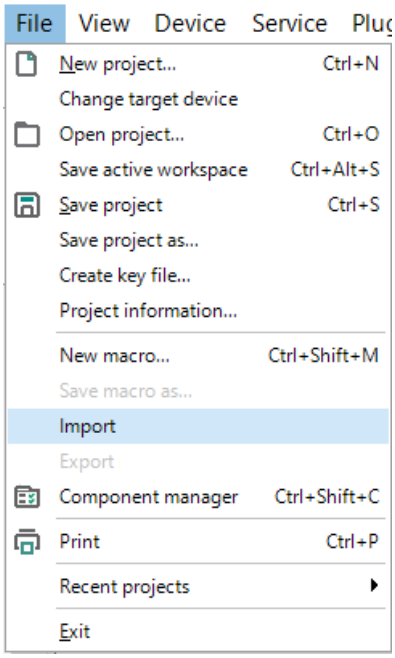


3. In the window that opens, select a location and save the function file with the extension **.fst*.
   Once saved, a message indicating that the function was exported successfully will be displayed.

Import ST function

If you need to use a function created in another project to create a program, you can import it into the project.
To import a function block, select **File** → **Import** in the main menu.

NOTE
The **Import** item is active only when the focus is on the project workspace.



In the window that opens, select the desired file and click the ***OK*** button. The function will be added to the ***Library Box*** in the ***ST functions*** section, now it can be used in the project.

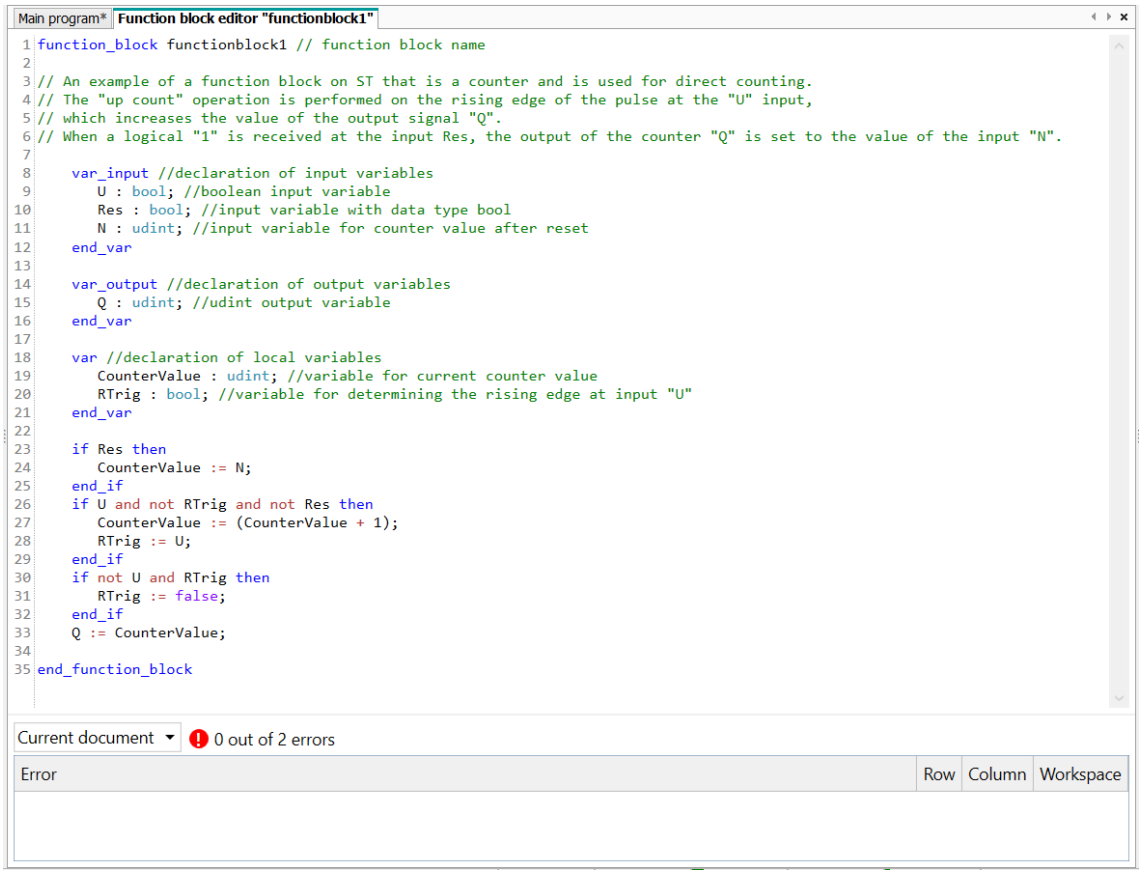
3.13 ST function blocks

Creation of user ST function blocks is available for devices on the new hardware platform. If the project is created for such a device, the toolbar icon  ***New ST function block*** is active.

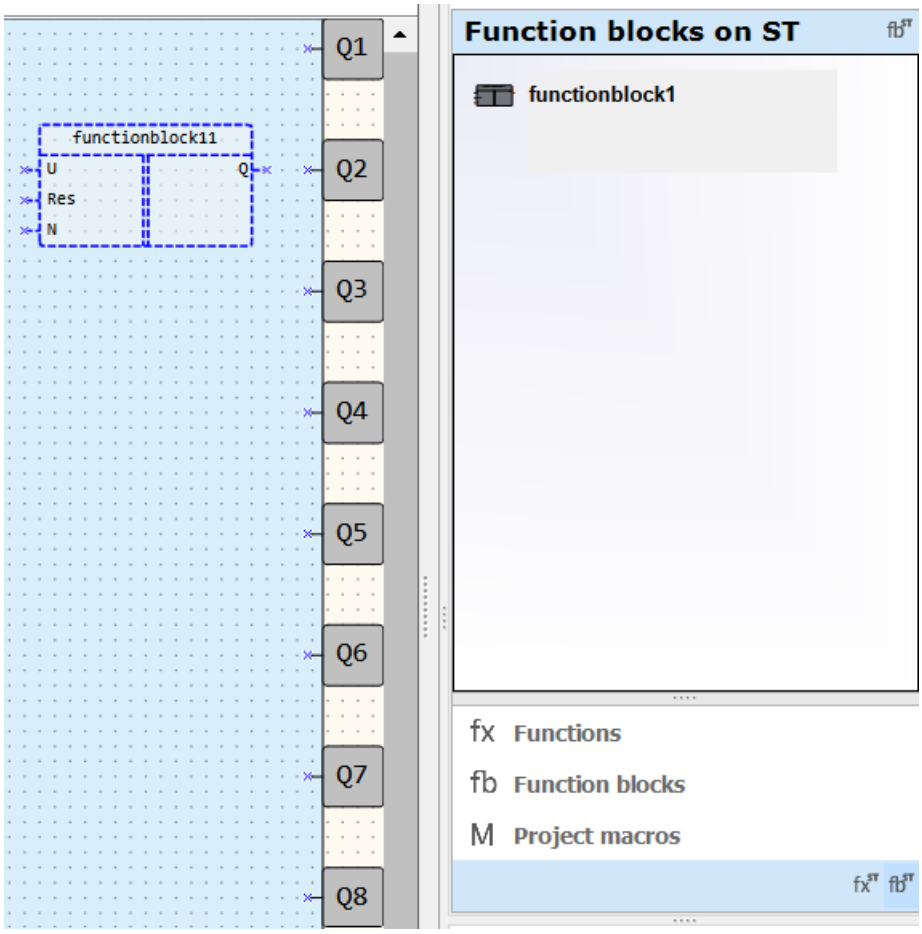
Creating an ST function block

To create an ST function block:

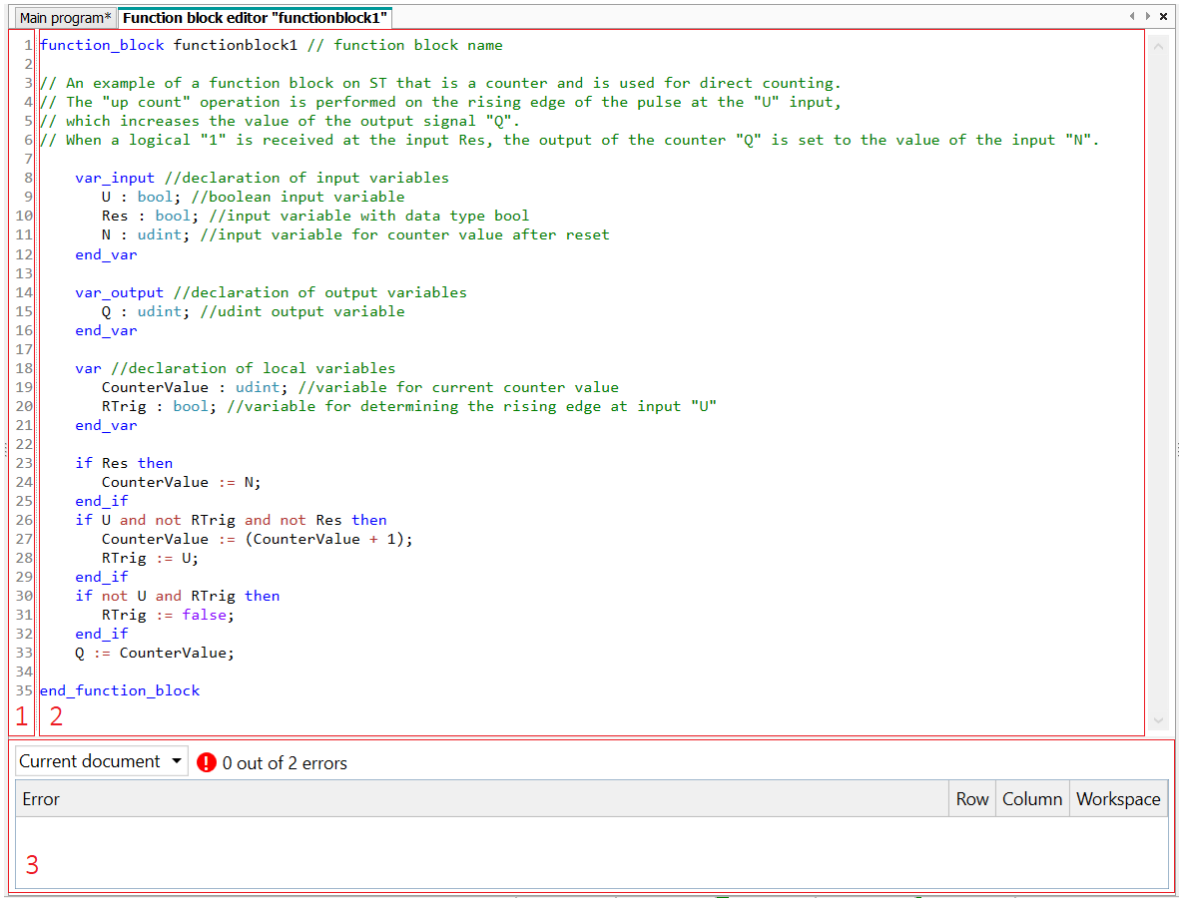
1. Click the toolbar icon  ***New ST function block***  . Function block editor with an ST function block template opens in a new workspace.



2. Specify the function block name and the output data type in the first line.
3. Specify all required inputs variables in the input variable declaration block ***var_input***.
4. Specify all required local variables in the local variable declaration block ***var***.
5. Develop a function block algorithm in accordance with the ST syntax rules.
6. Switch to the ***Main program*** tab or close the ***Function block editor*** tab. The function block will be saved automatically.
7. Select the section ***ST function blocks*** in ***Library Box*** and drag the saved block onto the project workspace.



Function block editor interface



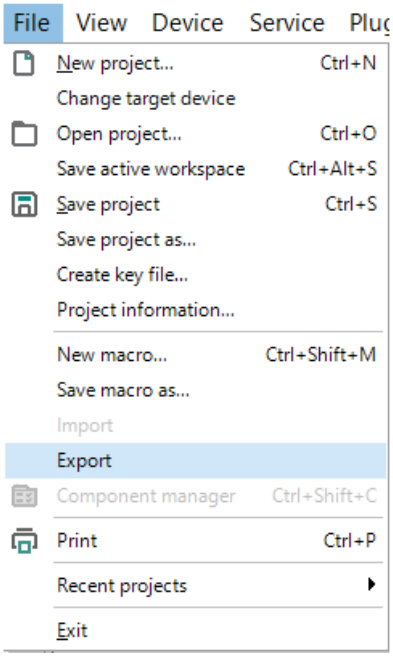
1. **Line numbers** — sequential line numbers in the program code.
2. **Code editor** — code editing area with automatic syntax highlighting.
3. **Error panel** — error display area.

NOTE
Similar to the function editor, the function block editor supports the functions *snippets* , *tracking the location of declaration and use* , *renaming* and *error tracking* .

Export ST function block

Exporting a function block to a file is only possible when the function block editor tab is open. To export a function, select **File** → **Export** in the main menu.
To export function block proceed as follows:

1. Open the function block in the editor.
2. Select **File** → **Export** from the main menu.

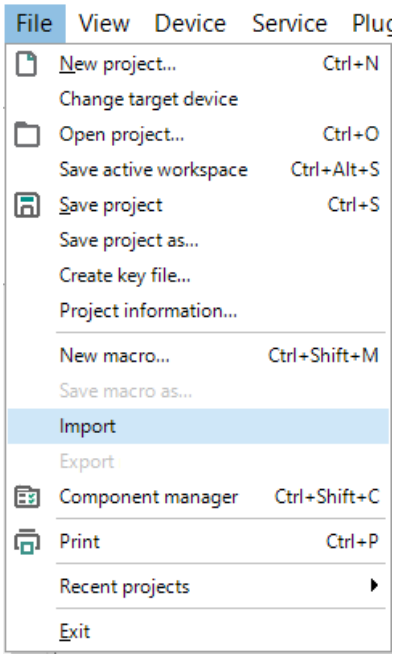


3. In the window that opens, select a location and save the function file with the extension **.fbst*. Once saved, a message indicating that the function was exported successfully will be displayed.

Import ST function block

If you need to use a function block created in another project to create a program, you can import it into the project.
To import a function blovk , select **File** → **Import** in the main menu.

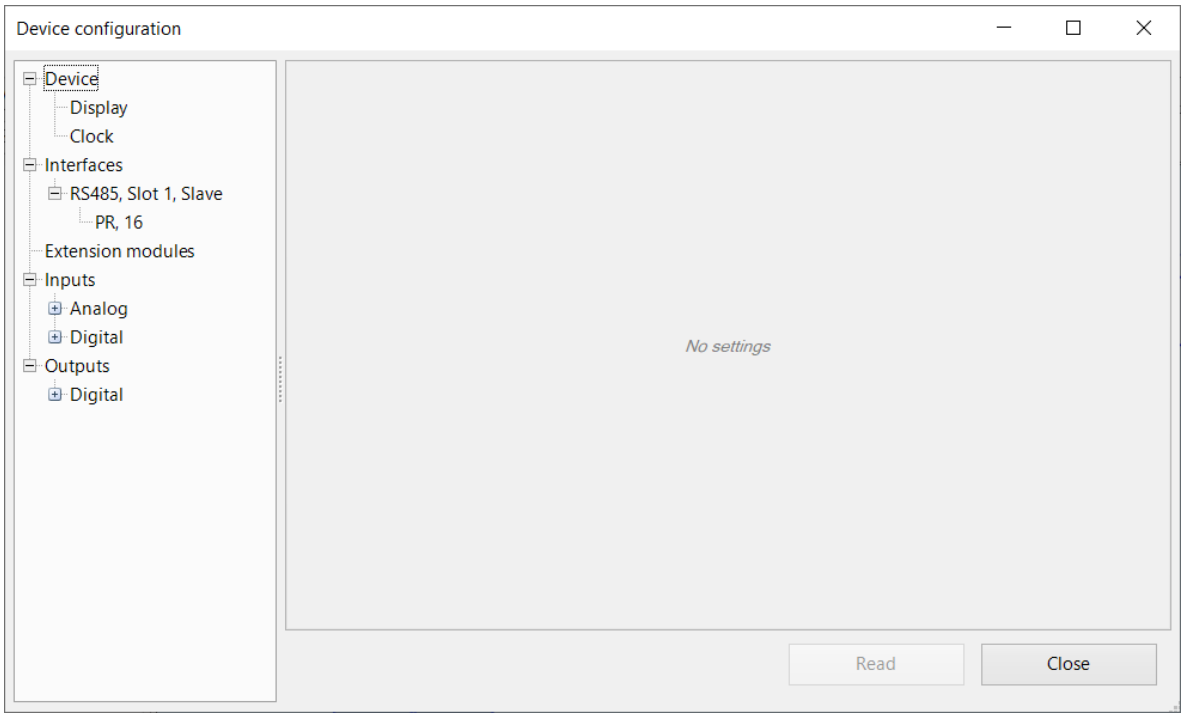
NOTE
The **Import** item is active only when the focus is on the project workspace.



In the window that opens, select the desired file and click the ***OK*** button. The function block will be added to the ***Library Box*** in the ***ST function blocks*** section, now it can be used in the project.

4 Device configuration

The configuration of the device is a part of a project and can be set using the menu item ***Device→ Configuration***. The dialog window ***Device configuration*** consists of two parts. The configurable parameters of the device are presented in the parameter tree in the left part of the window. The content of a group is presented in the right part.



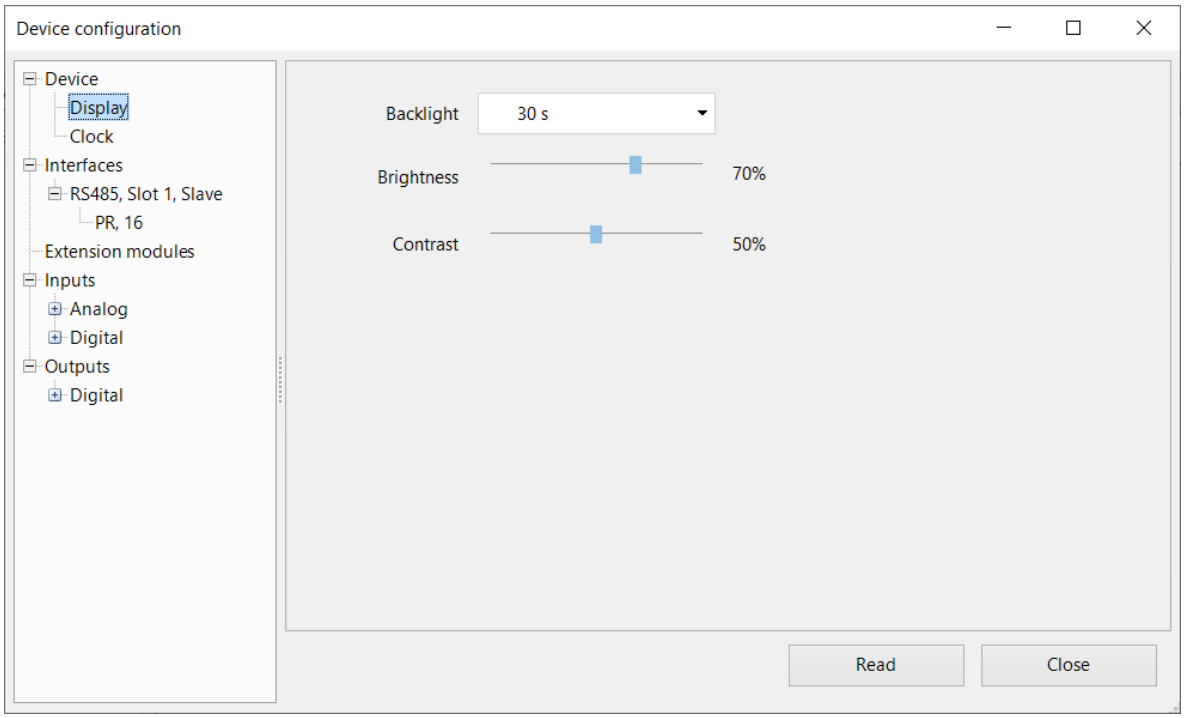
The content of the parameter tree depends on the target device and may include the following groups:

– *Display 4.1*;
– *Clock 4.2*;
– *Interfaces 4.3.1*;
– *Extension modules 4.4*;
– *Inputs and outputs 4.5*.

All the settings are saved in the project, except the clock settings. The configuration is also possible without connecting the device.

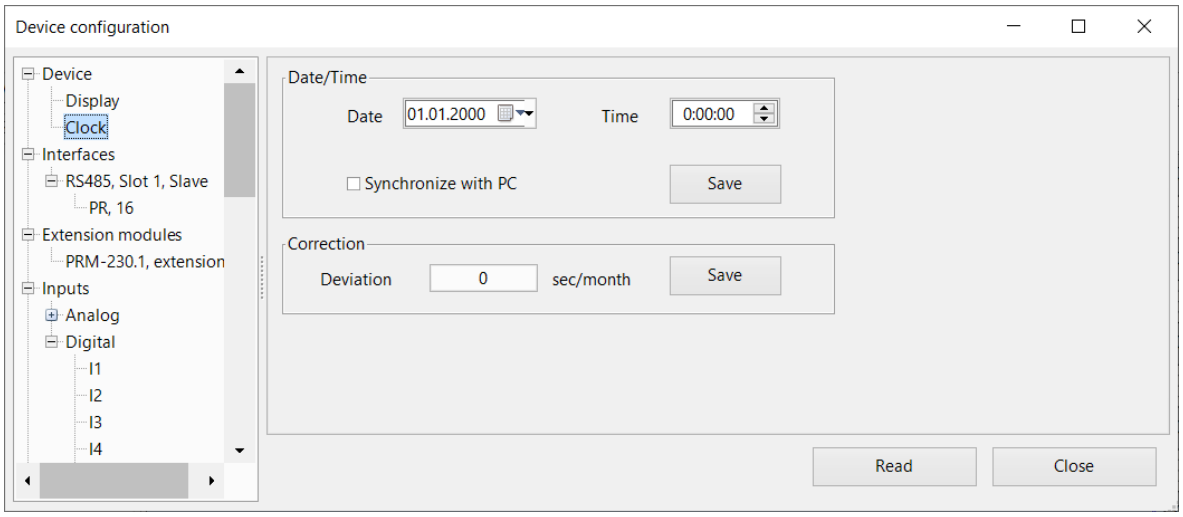
4.1 Display

If the target device has a display, the following parameters can be set:
Backlight – the duration of the backlight since the last user activity
Brightness – 0…100%
Contrast – 0…100%
The button ***Read*** can be used to read out the current display settings from the connected device.



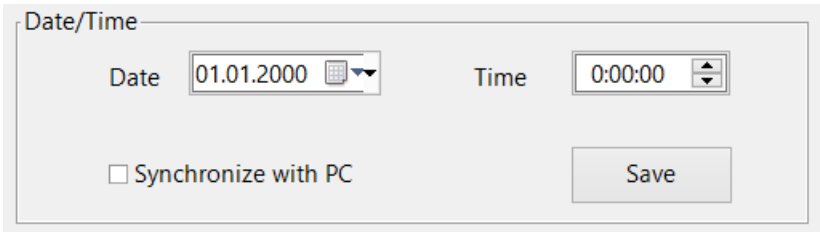
4.2 Clock

If the target device has a built-in real-time clock, the date and time can be set in the ***Clock*** group.



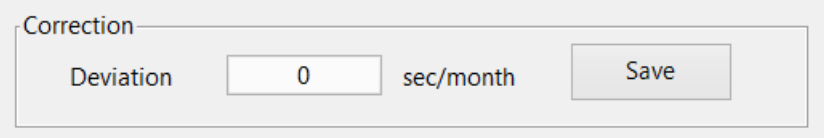
Date and time

To synchronize the device clock with the PC clock, check the checkbox ***Synchronize with PC***. In this case the fields ***Date*** and ***Time*** become inactive. To set the device clock to the new values click the button ***Save*** in the section ***Date/Time***.



Correction

Specify the clock error in seconds per month in the field ***Deviation*** to set the clock correction. Enter a negative value if the device clock is too fast.



To save the clock correction in the device, click the button ***Save*** in the section ***Correction***.
The button ***Read*** can be used to read the current time settings from the connected device.

Clock configuration for the new hardware devices

The clock settings window for devices on the new hardware platform has a different interface and does not have time correction (their hardware provides greater accuracy). Setting the time zone is required to display local time correctly, since the device stores the time value as Greenwich Mean Time (GMT). Enabling the **Set computer time zone** option synchronizes the real world clock device time with PC clock.

4.3 Data exchange

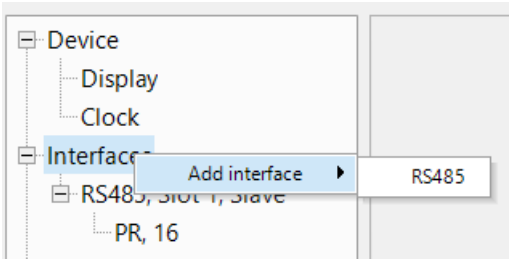
– *Interfaces 4.3.1*
– *Modbus 4.3.2*

4.3.1 Interfaces

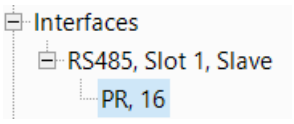
If the target device has a serial network interface RS485, its parameters can be set in the group ***Interfaces***.
By default, there is one interface configured as a slave and assigned to the hardware slot 1 with the following settings: master device with the name PR and the network address 16.
If the number of interfaces on the target device can be changed, interfaces can be added or deleted in the configuration, but their number cannot exceed the number of the existing slots.
If an interface is configured as a master, slaves can be added to the configuration or removed, but their number may not exceed 16.

Add interface

If the device has a slot, for which no interface is configured, an appropriate interface can be added using the item ***Add Interface*** in the context menu.

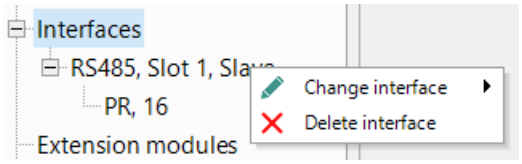


An interface of the selected type with default settings is added.

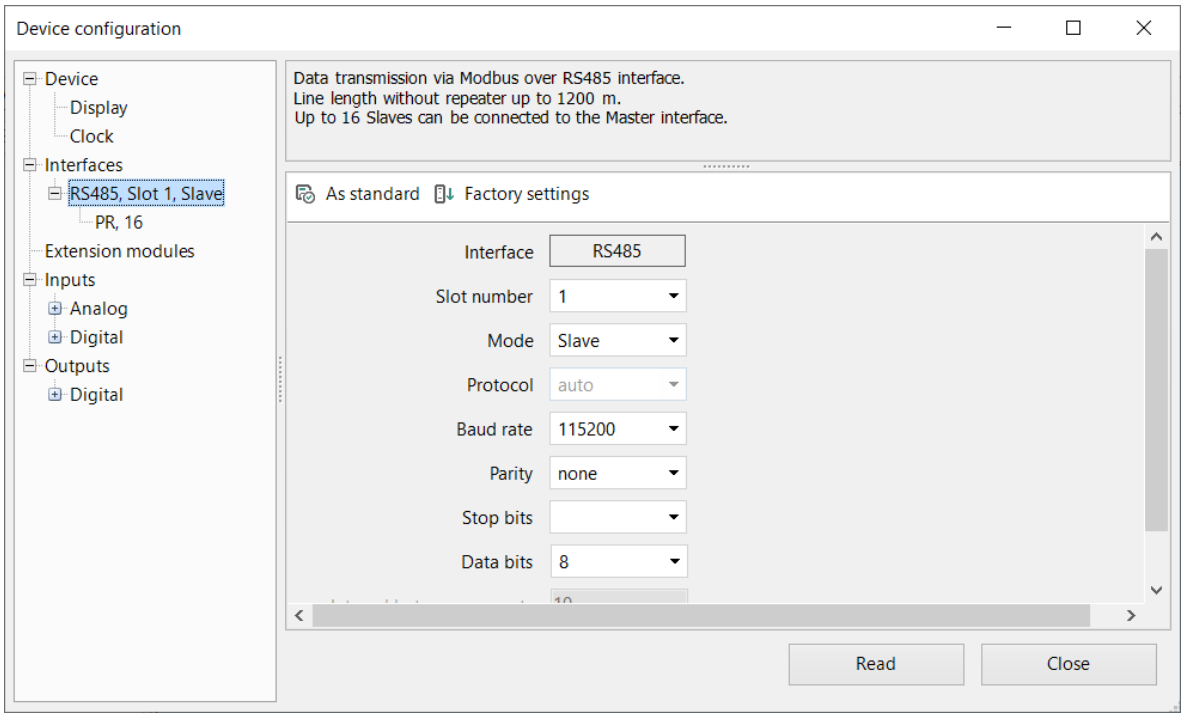


Replace/remove interface

Depending on device, the interface can be replaced by another type of interface or removed using the context menu.



4.3.1.1 RS485 interface configuration



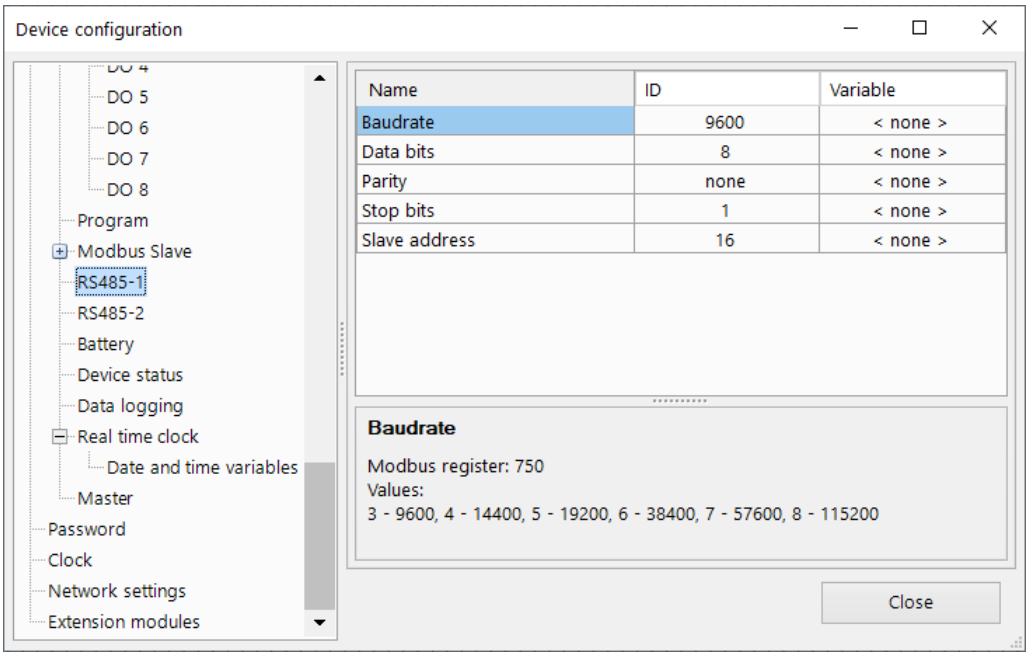
The type of the interface (RS485), the number of the assigned slot and the mode (master / slave) are displayed in the tree.
To establish the connection over the interface, it has to be configured. The parameters of the interface are displayed in the right part of the window. The default value depends on the target device. The parameters ***Protocol*** and ***Interval between requests*** are only available in the master mode. In the slave mode they are inactive and grayed out.

The icon  ***As standard*** is used to save the settings as default values for future projects.

The icon  ***Factory settings*** is used for to apply the unchangeable factory settings.
The button ***Read*** is used to read out the current settings from the connected device.
Use the button ***Close*** to save the settings in the project and close the dialog.

Interface parameters for devices on the new hardware platform

For devices on the new platform, the interface parameters are located in the **RS-485 port settings** section in the settings tree. The right side of the device settings window displays interface parameters. The settings window looks like the figure below.



4.3.1.2 Ethernet interface configuration

Ethernet settings are available only for PR103 in the **Ethernet Settings** menu in the **Network settings** section of the settings tree.
The settings window displays the current network parameters of the device, and also sets new ones. After saving the settings with the new IP address, the device should be rebooted.

NOTE
After setting a new IP address, the device will lose connection with the PC. For the new connection you need to specify a new IP address (see *Connection to device 3.6*).

4.3.2 Modbus

– *Modbus working 4.3.2.1*;
– *Master mode 4.3.2.2*;
– *Slave mode 4.3.2.3*.

4.3.2.1 Modbus working

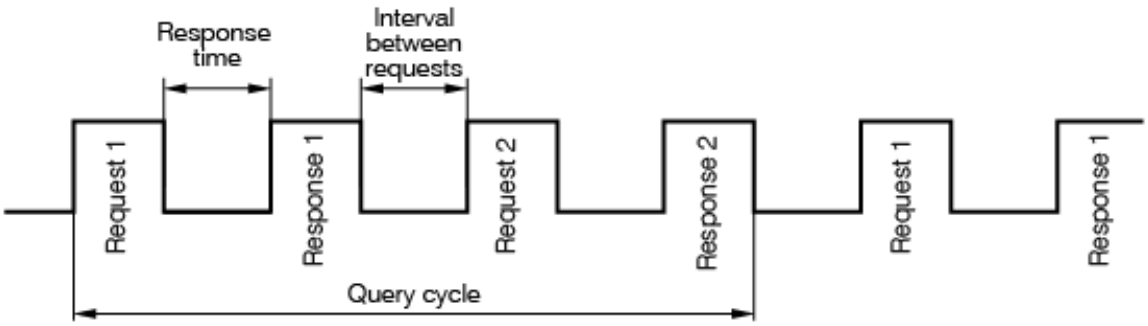
ALP can be used to program devices that support Modbus-RTU or Modbus-ASCII (master / slave) protocols.
In order to organize data exchange in the network over the RS485 interface, a master device is required. There can be only one master in the network.

Cycle time

The program execution time (cycle time) is automatically adjusted (auto-tuned) depending on the program complexity. The auto-tuning affects data exchange over Modbus, since the program execution has a higher priority than request processing. If the program is large, it can take up all the CPU time and Modbus data exchange will not be performed correctly.
To avoid this problem, the lower limit for the volume of the Modbus data exchange is reserved: 50 requests per second. Thus, at least 50 requests per second can be executed even if the user program is large, and even more if the program is small and the processor capacity is sufficient. If there is not enough time to poll all devices, the number of requests should be optimized in the user program.
The ***Query cycle*** setting depends on the number of polled variables and the polling frequency in the program. It is recommended to set ***Query cycle*** to 1 s. In this case, the device will be able to request up to 50 variables.

Query time

The query time is the actual time it takes the device to run all requests in a queue. If the queue is short, the device will perform all the request-response cycles and wait for the specified ***Query cycle*** to expire (Fig. 4.5). If the queue is long and the query takes longer than the specified ***Query cycle***, the device will poll all slaves in the shortest possible time.

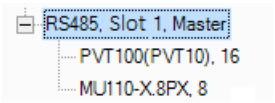


To minimize the request time, the following is recommended:

– If one or several slaves are not connected or temporarily unavailable, consider to block the polling in a program or to minimize the ***Timeout*** parameter for these devices.
– Consider the number of slaves and the total number of requests when setting the ***Query cycle*** parameter. If the processing time of all requests (query time) takes longer than ***Query cycle***, the parameter will be ignored.

Polling of multiple devices in the network

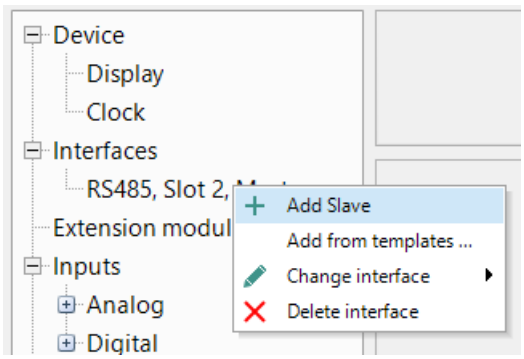
Slaves are polled according to the generated queue from the smallest to the largest address. In the following example, the slave with the address 8 is polled first, while the one with address 32 is polled last.



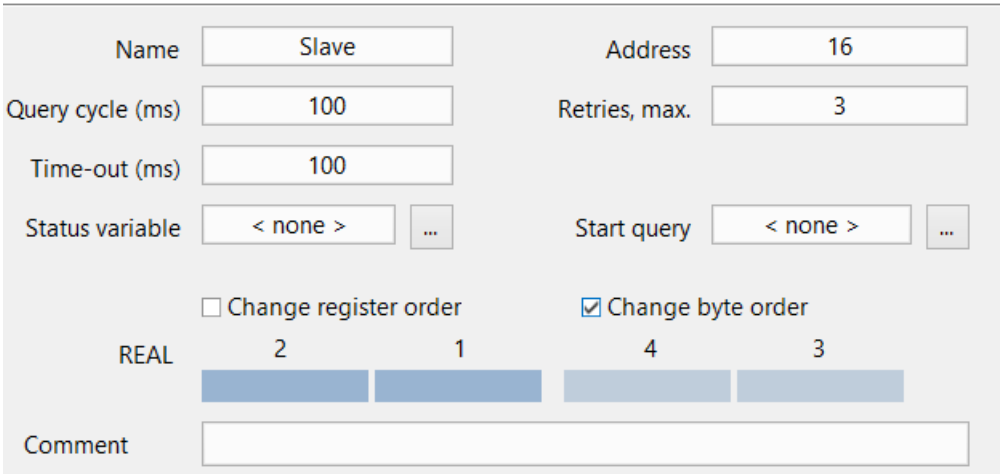
Query cycle can be set for each slave individually.

4.3.2.2 Master mode

Each interface can control up to 16 slaves. Each slave supports up to 256 variables. The addresses and names of the variables need only be unique if they belong to the same slave.
In the master mode, all slaves connected to the interface are sequentially requested. Select the mode ***Master*** in the parameter list, set other connection parameters and add the required number of slaves using the item ***Add slave*** in the interface context menu.



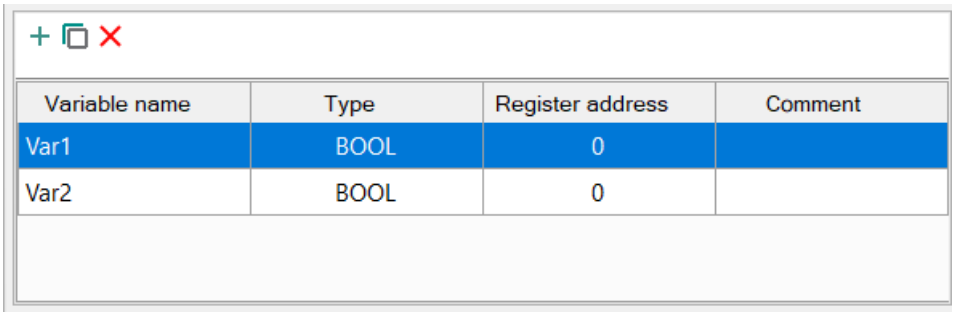
The added slave device is displayed with its name and address in the tree below the interface. Select a slave to configure it in the right part of the window.
To delete the slave, use the context menu or the icon ***Remove Slave***.



- ***Name*** – the name of the slave displayed in the tree
- ***Address*** – the network address of the slave
- ***Query cycle (ms)*** – the time interval between queries. A query comprises the number of requests according to the number of variables listed for the slave. The valid range is 0…65535 ms.
- ***Timeout (ms)*** – the time that request can take before the attempt is considered as failed. The valid range is 0…65535 ms.
- ***Retries, max.*** – the number of the failed request attempts before query is stopped and the status of the device changes. The valid range is 0…255.
- ***Burst request*** – group request of consecutive registers to increase the data throughput
- ***Status variable*** – select a BOOL variable using the icon «**…**» to record the device status:
  - 1 – the device functions properly
  - 0 – the device is not connected.
- ***Start query*** – select a BOOL variable using the icon «**…**» to control the query:
  - 0 – query disabled
  - 1 – query enabled.
- ***Change register order*** – determines the register order in two-register variables
- ***Change byte order*** – determines the byte order in the register
- ***Comment*** – description text

The list of the variables to be requested from this slave is in the lower part of the window. Each variable created in this list can be found in the variable table under the tab ***Network, Slot X*** with a separate list of variables for each slave device.

Add a variable by clicking the icon  , and set its properties.

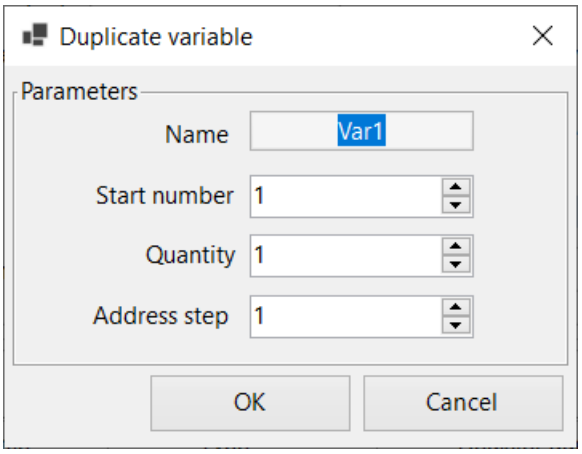


Variable name	Type	Register address	Comment
Var1	BOOL	0	
Var2	BOOL	0	

- ***Name*** – the name of the variable
- ***Type*** – the data type of the variable: BOOL, INT or REAL
- ***Register*** – the register address
- ***Bit*** – the number of the bit of the register (0…15) (only for BOOL variables)
- ***Read function / Write function*** – selection of the read / write function or disable reading / writing.

– ***Number of registers*** – the number of registers occupied by the variable (only for INT variables)
– ***Start reading*** – assign the BOOL variable for forced reading of the requested variable
– ***Start writing*** – assign the BOOL variable for forced writing of the requested variable
– ***Status variable*** – assign the INT variable to record the error code
– ***Comment*** – description text

To create several variables with the same settings, select a variable and click the icon ***Duplicate***.



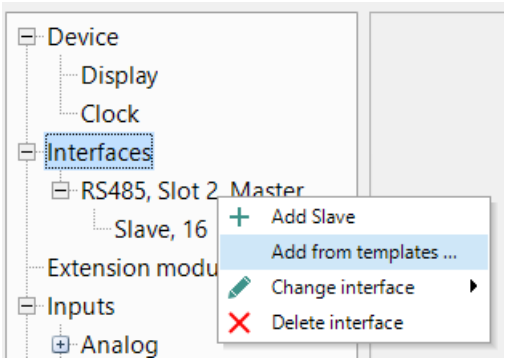
– ***Name*** – the name of the duplicated variable
– ***Start number*** – the initial number to add to the name of the duplicated variable
– ***Quantity*** – the quantity of the duplicated variables
– ***Address step*** – the address increment

Click ***OK*** to add the duplicated variables to the list of variables. The variables will be stored in adjacent register cells with consecutive addresses.

To remove the variable from the list, use the icon ***Delete***.

Templates

A slave device in the configuration mask can be saved as a template, with its parameters and variables, to be used in further projects. Use the context menu item or the icon ***Save Slave as a template***. The template is saved as a file with the extension ***.dvtp**.
A slave can be added to a master as a template using the context menu item ***Add from templates…***



Master mode for the devices on the new hardware platform

To configure parameters for polling connected devices, select the **Modbus Master** node in the device parameter tree.
Modbus Master parameters:

– ***Interval between requests, ms*** - the time period after which the survey is repeated. The valid range is from 1 to 10,000 ms.

NOTE
The maximum number of devices on one interface is 32.

To change the parameters of the device being polled, click on its name in the settings tree. The right part of the window will display the available parameters: in the upper part – device parameters, in the lower part – network variables of the device.
Parameters of the device being polled:

– ***Name*** — device name to be displayed in the settings tree
– ***Interface*** — interface through which the device being polled is connected. The list of available parameters depends on the selected interface.
– ***Address*** — device network address
– ***Number of re-request*** — number of unsuccessful polling attempts. Valid range is from 0 to 3
– ***Response timeout, ms*** — the time after which a polling attempt is considered unsuccessful. Valid range is from 10 to 10,000 ms
– ***Byte order*** — determines the order of bytes in the packet
– ***Comment*** — text description of the device

Specific parameters of the polled device connected via the Ethernet interface:

– ***IP address*** — unique network address of the device, valid range from 0.0.0.0 to 255.255.255.255
– ***Port*** — port number, valid range from 0 to 65535.

Properties of network variables of the polled device:

– ***Name*** — name to display in *variable table 5*
– ***Type*** — *type 5.1* of the variable: boolean, integer or floating point
– **Register** — the value of the register accessed by the device is displayed in the table
– ***Bit (Boolean variables only)*** — bit number to read
– ***Number of registers (integer variables only)*** — number of registers occupied by a variable: 1 or 2
– ***Comment*** — text description of the variable to be displayed in *variable table 5*
– ***Function*** — disable or select the write/read function.

NOTE
Creating variables with the same names is not allowed.

The list of parameters to configure depends on the choice of the write/read function.
Reading function parameters:

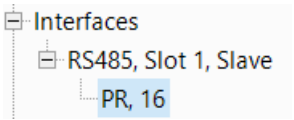
– ***Reading period*** is a time interval between requests;
– ***Read command*** is a Boolean type variable, changing which causes the parameter to be read.

Writing function parameters:

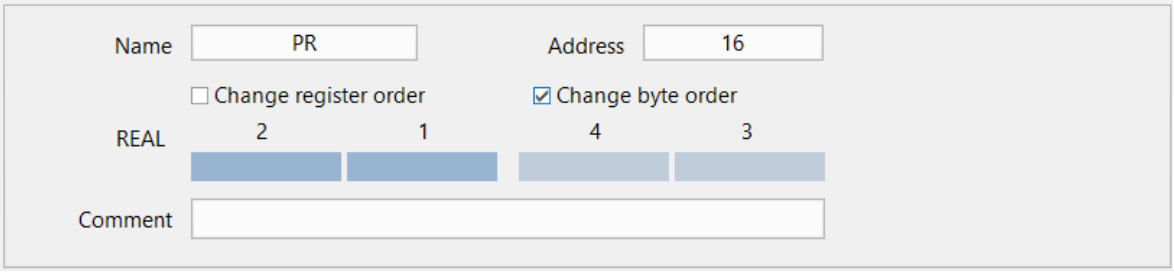
– ***Writing period*** is a time interval between rewrite operations;
– ***Write command*** is a Boolean type variable, the change of which causes the parameter to be written.
– ***Write on change*** — if this function is activated, then if the value of a variable changes, the master initiates writing the value of the variable to the Slave device.

4.3.2.3 Slave mode

An RS485 interface added to the tree item ***Interfaces*** has the default mode Slave and the default master with the name PR and the address 16 added below. Select the interface to set the connection parameters.
To configure the data transfer parameters, click on the device name (PR, 16 by default) in the tree.



Select the master in the tree to set the parameters for data exchange.

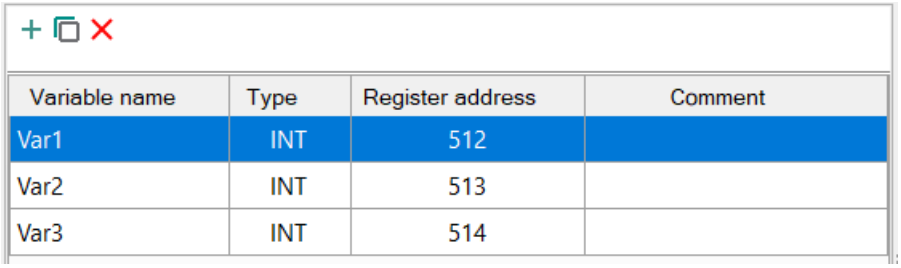


The common parameters for data exchange can be set in the upper window part.

– ***Name*** – the name of the master displayed in the tree
– **Address** – the network address of the master
– ***Change register order*** – the register order in two-register variables
– ***Change byte order*** – the byte order in the register
– ***Comment*** – description text

The list of the variables to be requested by the master is in the lower part of the window. Each variable created in this list can be found in the variable table under the tab ***Network, Slot X***.

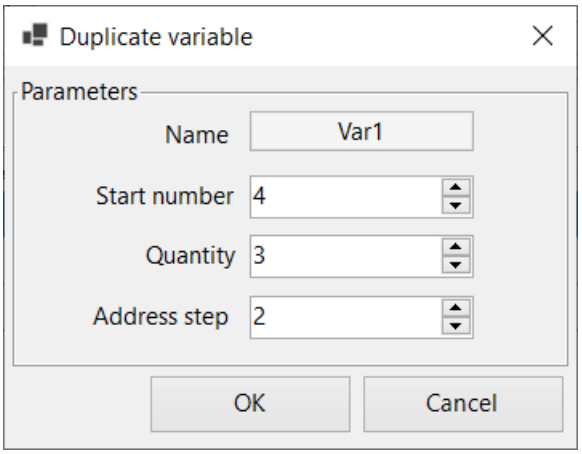
Add a variable by clicking the icon  ***New variable*** and set its properties.



Variable name	Type	Register address	Comment
Var1	INT	512	
Var2	INT	513	
Var3	INT	514	

– ***Name*** – the name of the variable
– ***Type*** – the data type of the variable: BOOL, INT or REAL
– ***Register*** – the register address. The range of the available addresses is specified in the device user guide.
– ***Comment*** – description text

To create several variables with the same settings, select a variable and click the icon  ***Duplicate***.



– ***Name*** – the name of the duplicated variable
– ***Start number*** – the initial number added to the name of the duplicated variable

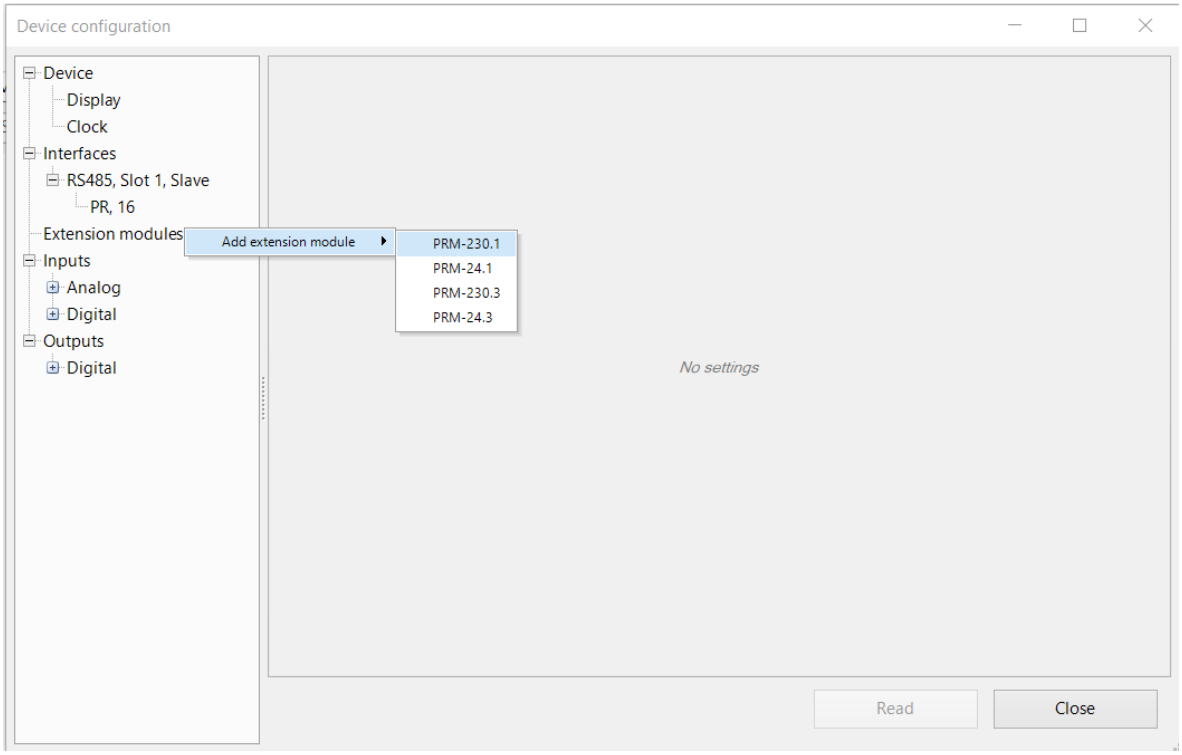
– ***Quantity*** – the quantity of the duplicated variables
– ***Address step*** – the address increment

Click ***OK*** to add the duplicated variables to the list of variables. The variables will be stored in adjacent register cells with consecutive addresses.

To remove the variable from the list, use the icon ***Delete***.

4.4 Extension modules

Up to two I/O extension modules of type PRM can be connected to base device. For further information about extension modules refer to the PRM user guide.
To use module I/O points in the circuit program, add the module to the group ***Extension modules*** using its context menu.

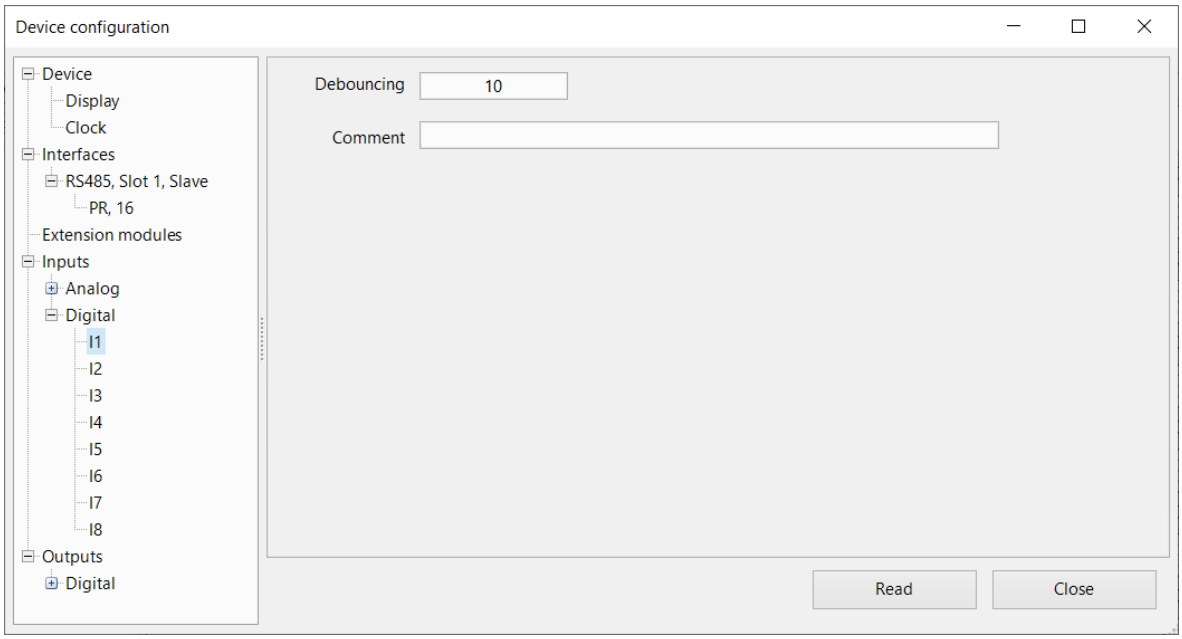


The additional I/O points of the added modules can be configured in branches ***Inputs*** and ***Outputs*** respectivel. They are displayed in the tree as ***Ix(y)*** and ***Qx(y)*** respectively, where ***x*** is the ordinal number of the I/O point on the module and ***y*** is the ordinal number of the module counting from the base device.
Before uploading the project to the base device, all modules must be connected via the internal bus to base device and powered on. The module firmware is synchronized with the current version of ALP when uploading a project.

4.5 Inputs and outputs

Inputs

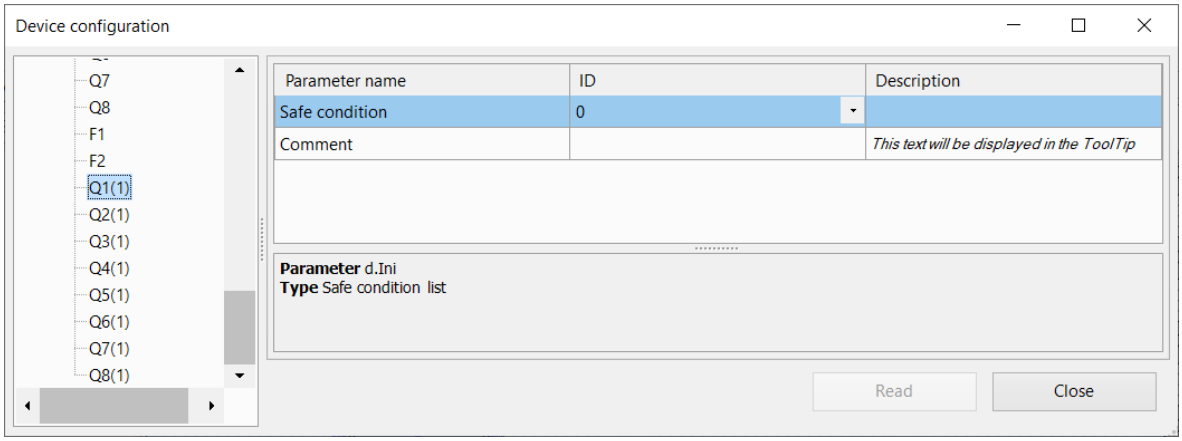
The content of the branch ***Inputs*** depends on the resources of the target device. It can be analog and/or digital inputs.
The parameter ***Comment*** is common for all types of inputs. The text in this field is displayed in a tooltip, when the mouse cursor is over the input in the workspace. The text can be entered in Property Box too.
For further details about the configuration of the inputs, refer to the device user guide.



Other input parameters depend on the types of the input and the device.

Outputs

The content of the branch ***Outputs*** depends on the resources of the target device. It can be analog and/or digital outputs.
The parameter ***Comment*** is common for all types of outputs. The text in this field is displayed in a tooltip, when the mouse cursor is over the output in the workspace. It can be entered in Property Box too.
For further details about the configuration of the outputs, refer to the device user guide.
The digital outputs of the extension module have an additional parameter ***Safe condition***. The parameter specifies the output state in case the connection between the module and the base device is lost.



Settings for devices on the new hardware platform

The window for setting up inputs and outputs for devices on the new platform has a different interface, and the parameters on the right side of the window are presented in a table.
For devices on the new platform, the menu for setting the safe state of the outputs is located in the branch of the added expansion module.

4.6 Password

For devices on the new platform, you can set a password to protect the device.

The password is set in the **Password** section in the settings tree only for the device connected to the PC.

Create password

If a password is not set in the device, then password creation will be active in the settings window. To install, you must enter and confirm your password.

Changing and resetting password

If the device has already set a password, you can change or reset it.
To change the password, enter the current password and the new password in the **Change password** columns.
To reset your password, enter your current password in the **Reset password** column.
If you have lost your password, see the device user guide in order to reset it.
For a password-protected device, a password is required when recording a program, see *Upload project to device 3.7*.

5 Variables

To see all project variables, click the icon in the toolbar or use the menu item ***Device*** → ***Variable table***.
The variables are divided into three groups, each of which has a separate tab in the table:

– **Standard**
– *Service 5.2*
– *Network 5.3*

To create a new variable, you can use:

– toolbar icon
– key combination ***Ctrl+N***

or simply write a new variable name in the last row.
If you create a new variable after an unsuccessful search, the name entered in the search field will be proposed as the name of the new variable.
To duplicate an existent variable you can use:

– toolbar icon
– key combination ***Ctrl+D***
– context menu item ***Duplicate***

To delete a variable, you can use:

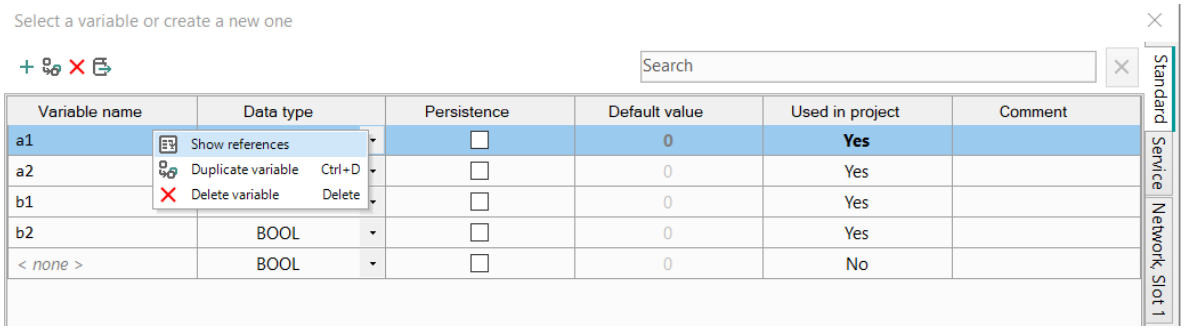
– toolbar icon
– key ***DEL***
– context menu item ***Delete***

Service variables can neither be created nor deleted.
The rows in the Variable Table can be sorted by each column.

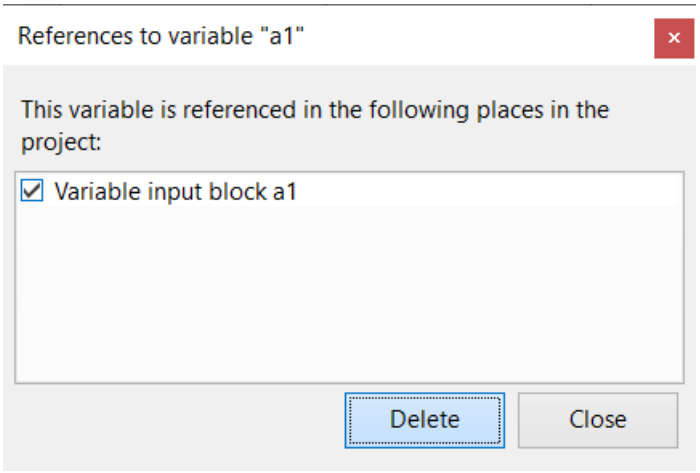
Variable properties

– ***Name*** – the name of the variable.
– ***Data type*** – *BOOL, INT or REAL 5.1*.
– ***Persistence*** – only for standard variables available. The variable is stored in the non-volatile memory of the device and becomes a retained variable. For detailed information about storage time and memory size, refer to the device user guide.
– ***Default value*** – available only for retained and network variables. It is the value at the first start of the program, new values are assigned to them.
– ***Used in project*** – the variable has a reference to a block in the program.
– ***Comment*** – the text displayed in a tooltip in the workspace, when the mouse cursor is over the variable.

Use the item ***Show references*** in the variable context menu to see where the variable is used in the project.



In the dialog window ***References to the variable*** select the reference you want to delete and click ***Delete***.
To remove the variable from the table, use the item ***Delete variable*** in its context menu.



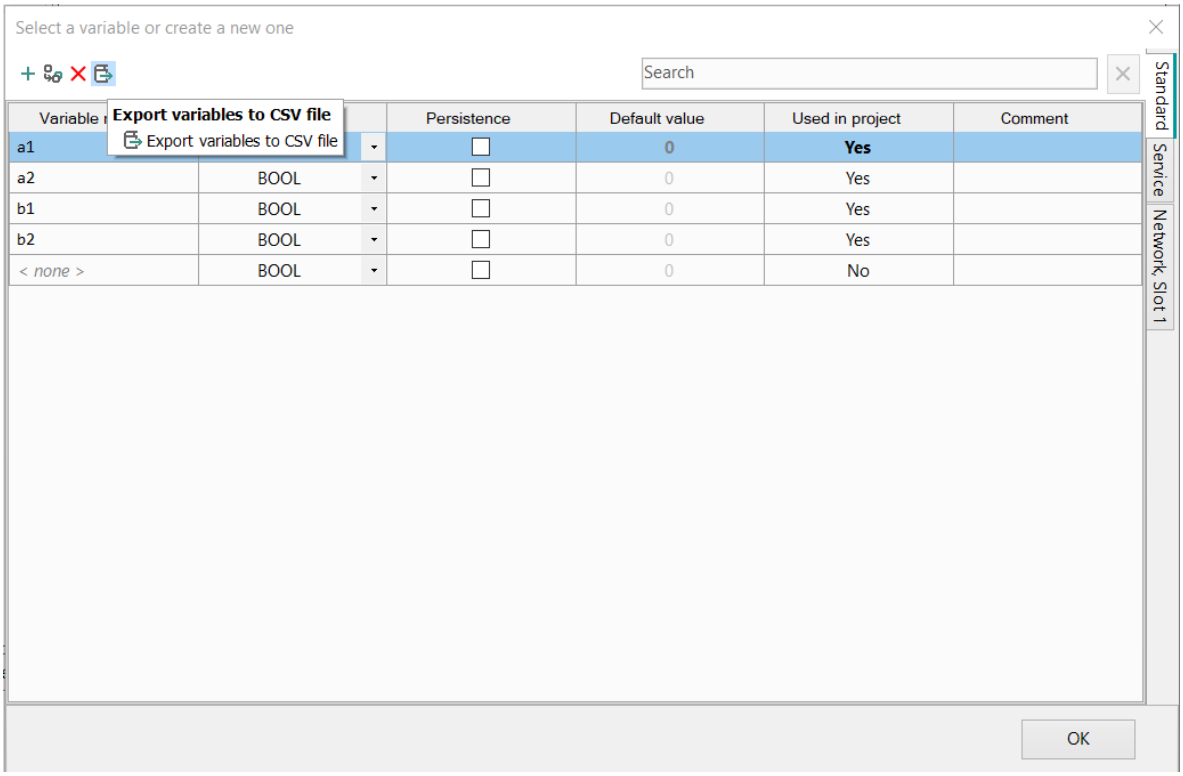
Exporting variables to a file

The variable tab can be exported as a table in ***.csv*** format. To do it proceed as follows:

1. Click ***Export variables tab to CSV file*** in the upper left part of the table.
2. In the window that opens, specify the location for uploading the file.
3. Press the ***Save*** button.

NOTE
The file name is formed depending on the exported tab according to the scheme ***ProjectName_Tab_Variables***.

NOTE
For instruments on the new platform, the Slave tab of network variables is exported along with the variables panel.



5.1 Data types

The variables of the following types can be used in a program:

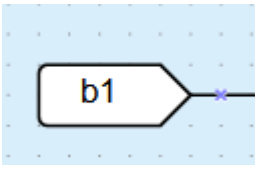
– Boolean (***BOOL***)

– Integer (***INT***)
– Real (***REAL***).

NOTE
Different devices can have restrictions related to support of certain types of variables.

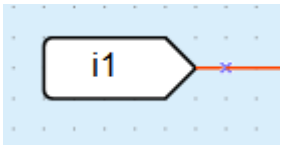
BOOL

A variable of this type has only two possible values: 1 (**True**) or 0 (**False**).
The connecting lines between the BOOL variables in the circuit program are displayed in gray.



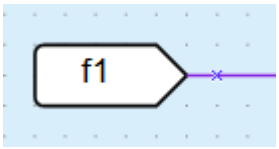
INT

A variable of this type is an unsigned integer in the range of 0…4,294,967,295 (4 Byte).
The connecting lines between the INT variables in the circuit program are displayed in red.



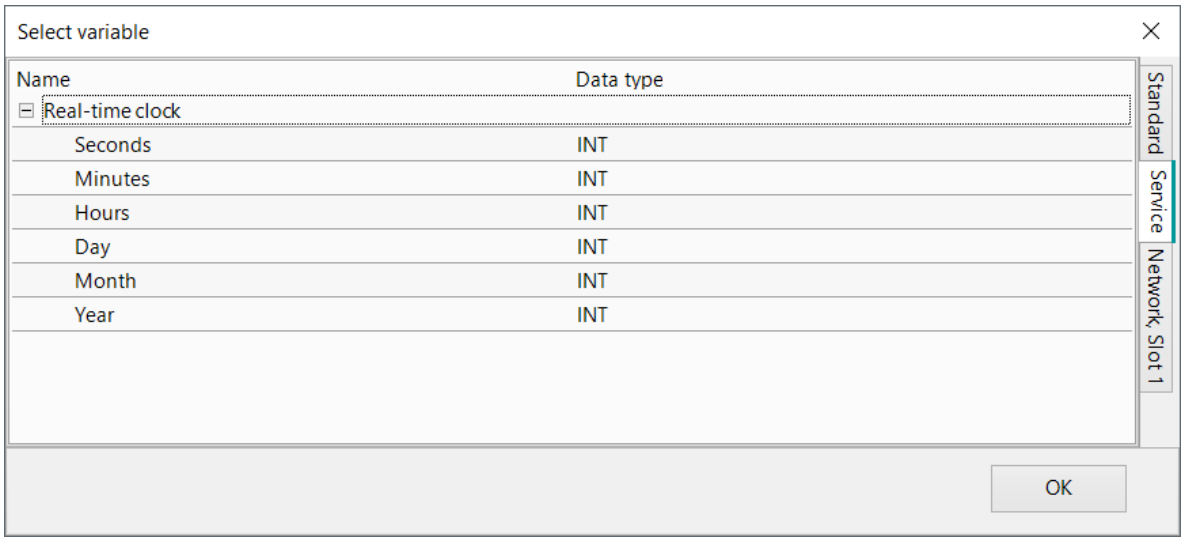
REAL

A variable of this type has a value in the range of -3.402823e+38…3.402823e+38. It is represented by a floating-point number of single-precision (4 Byte).
The connecting lines between the REAL variables in the circuit program are displayed in violet.

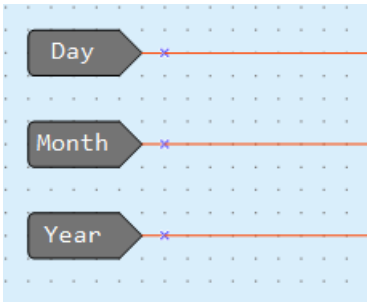


5.2 Service variables

Service variables are associated with the device settings and can differ, depending on the device. Service variables are related to hardware features, such as the real-time clock, interface card in the slot, etc., and cannot be deleted. Access rights to service variables may be limited.
The service variables are listed in the variable table under the tab ***Service***.

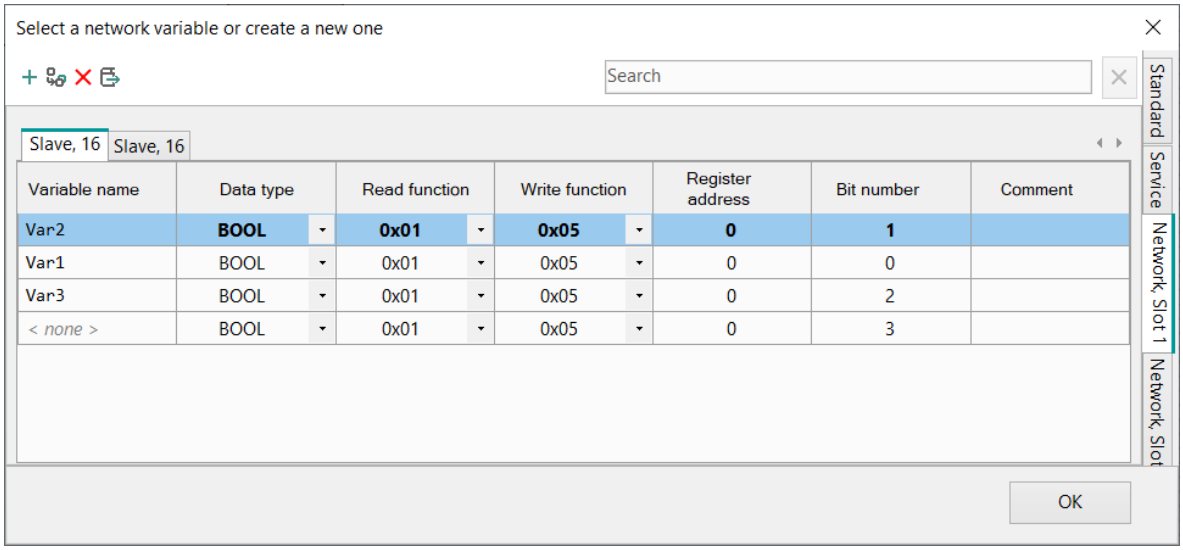


The blocks of service variables are shown in the circuit program with a gray background.

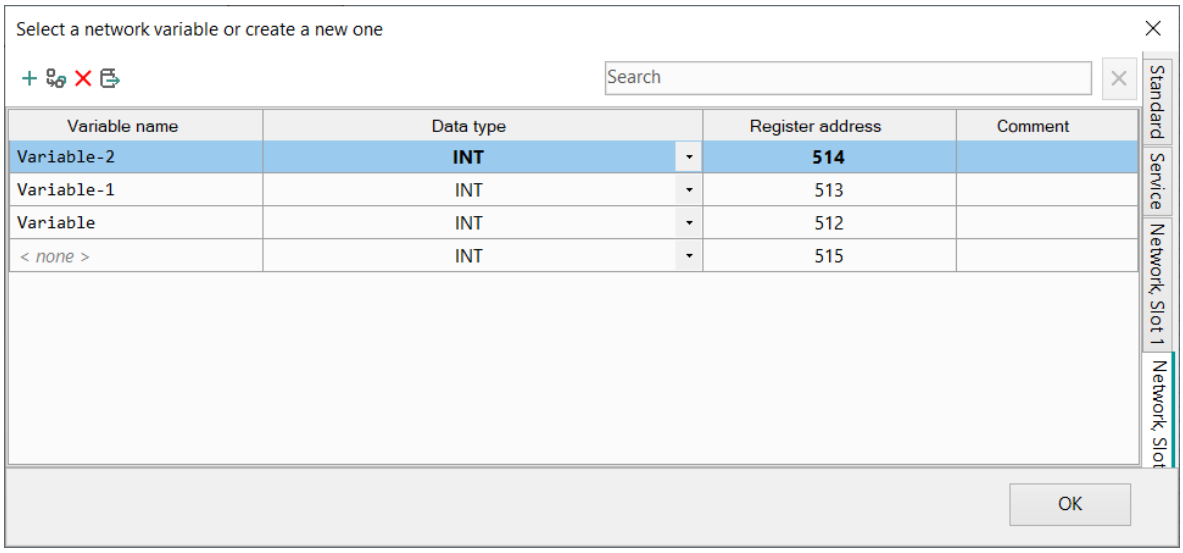


5.3 Network variables

Each interface slot has a separate tab in the table.
If the interface is configured as a master, there are separate tabs for each slave device within the slot tab. The Slave tab contains the variables to be requested for this slave device.



Network variables and their references are deleted in the same way as standard variables.
For more details about network variables for master interface see *Master mode 4.3.2.2* section.
If the interface is configured as a Slave, all network variables to be requested by the master are shown in one list.



5.4 Copy-paste variable block

The variable blocks can be copied and pasted into another project.

To copy a variable block, select it in the workspace and use the toolbar icon or the item ***Copy*** in the block context menu.
To paste a variable block into another project, open it in the second ALP instance and use the toolbar icon or the item ***Paste*** in the workspace context menu.
Rules for copying all variables associated with the block:

– If the variable associated with the block is unique in the second project, it will be added with all properties to the variable table.
– If there is an identical variable in the second project, it will be associated with the pasted block. No new variables will be added to the variable table.
– If the second project has a variable with the same name but different parameters, a new variable will be created. To resolve the naming conflict, the name of one of the variables should be changed manually.
– It is not possible to insert variables of REAL type into a project for a target device that does not support REAL data type.
– Retained (persistent) variables cannot be copied into a project for a target device that does not support them.

Rules for copying service variables:

– Service variables cannot be copied to a project written to a target device without a real-time clock.

Copy rules for network variables:

– Only the variables of slave interfaces can be copied into another project and the interfaces in both projects must have the same slot numbers. The variables of the master interface should be created manually.
– Any register conflict must be resolved manually.

6 Library

If a project is open, the panel ***Library Box*** contains the following libraries:

– *Functions 6.1*
– *Function blocks 6.2*
– *Project macros 6.3*

Select an icon in the lower part of the panel to show the respective content.
Project macros library comprises the macros that have been created, imported or included to the project from Online Database.
View options can be changed using the icons located in the toolbar of the panel.

6.1 Functions

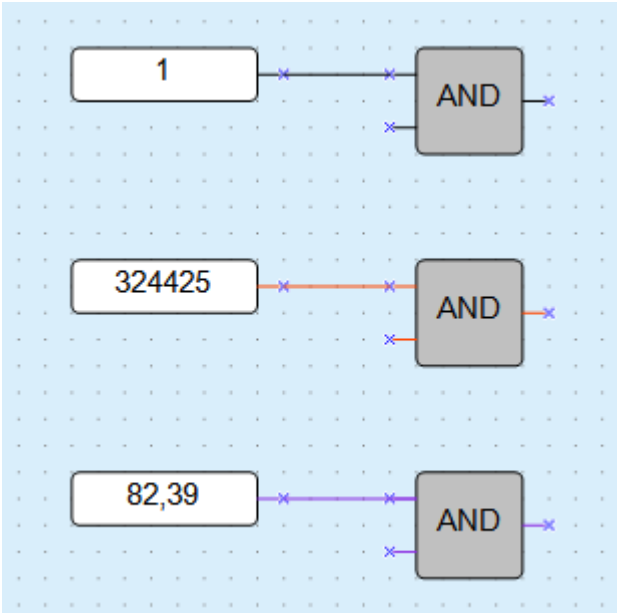
The library contains the following function groups:

– *Logical operators 6.1.1*;
– *Mathematical operators 6.1.2*;
– *Relational operators 6.1.3*;
– *Bitshift operators 6.1.4*;
– *Bit operators 6.1.4*.

6.1.1 Logical operators

– *Conjunction (AND) 6.1.1.1*
– *Disjunction (OR) 6.1.1.2*
– *Negation (NOT) 6.1.1.3*
– *Exclusive OR (XOR) 6.1.1.4*

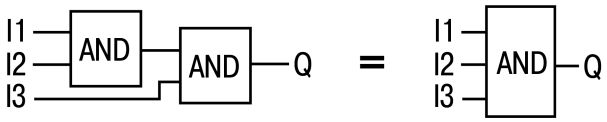
The logical operators can operate with BOOL or INT variables.
If the input values are INT, the operation is performed bitwise and the output is also an INT.



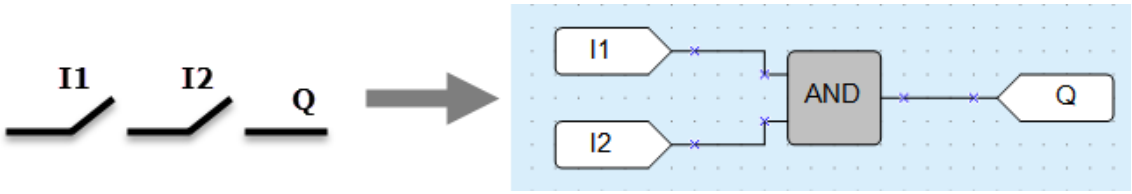
For **AND** and **OR** operators, it should be taken into account that unconnected block inputs will have the following states:

– for **AND** – TRUE
– for **OR** – FALSE

In this case, the blocks act as a signal repeater. To increase the number of inputs for logical operators, their cascade connection is used:



6.1.1.1 Conjuction (AND)



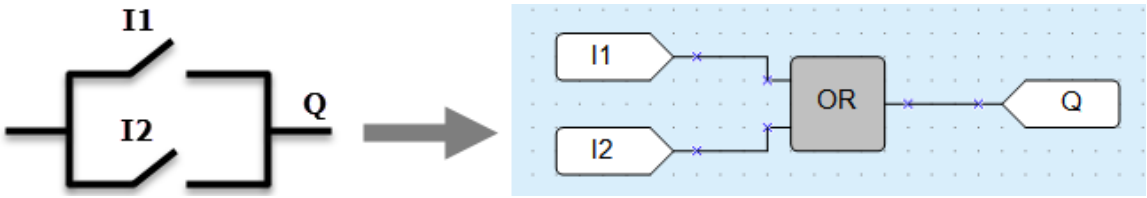
The output **Q** is **True** if both inputs are **True**. The function **AND** represents a serial connection in an electrical circuit.
Truth table:

I1	I2	Q
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation example with integer inputs:

AND	0011 (decimal 3)
	0101 (decimal 5)
	0001 (decimal 1)

6.1.1.2 Disjunction (OR)



The output **Q** is **True** if at least one of the inputs is **True**. The function **OR** represents a parallel connection in an electrical circuit.
Truth table:

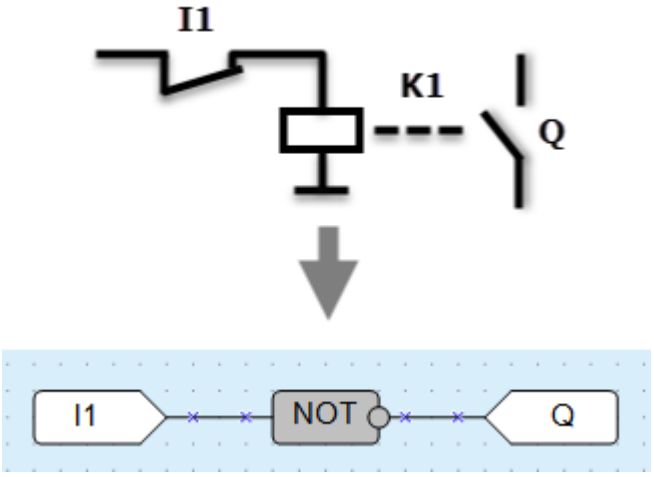
I1	I2	Q
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation example with integer inputs:

OR	0011
	0101
	0111

6.1.1.3 Negation (NOT)

The function **NOT** inverts the signal. The output **Q** is **True** if the input is **False** and vice versa.



Truth table:

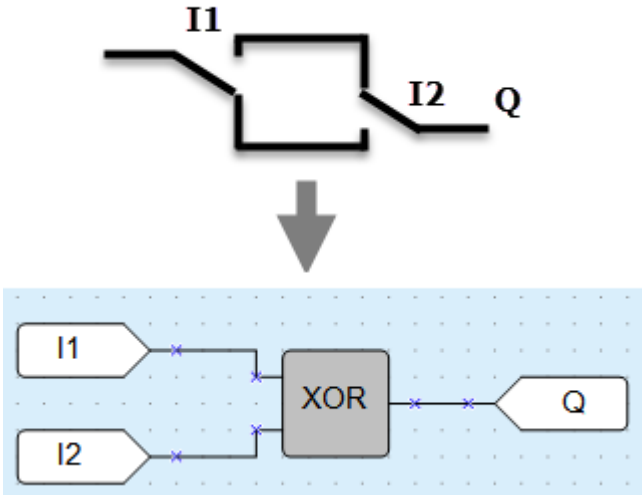
I1	Q
0	1
1	0

Bitwise operation example with integer inputs:

NOT	01
	10

The bitwise NOT, or complement, is a unary operation that performs logical negation on each bit, forming the ones' complement of the given binary value.

6.1.1.4 Exclusive OR (XOR)



The output **Q** is **True** if only one of the inputs is **True**.
Truth table:

I1	I2	Q
0	0	0
0	1	1

I1	I2	Q
1	0	1
1	1	0

Bitwise operation example with integer inputs:

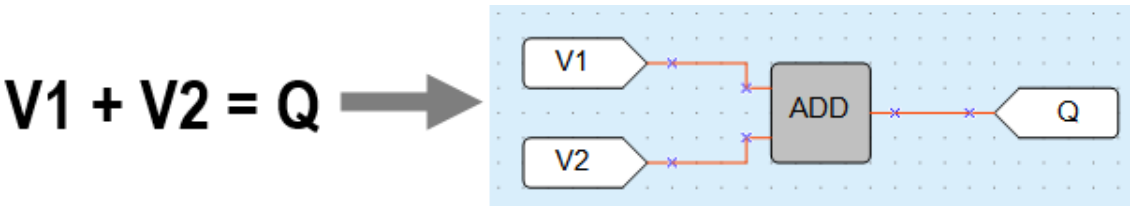
XOR	0011
	0101
	0110

6.1.2 Mathematical operators

There are different operators for different data types:

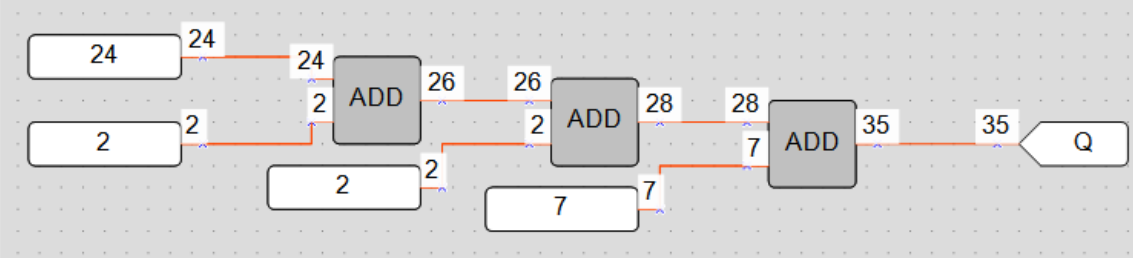
Operator	INT	REAL
Addition	*ADD 6.1.2.1*	*fADD 6.1.2.1*
Subtraction	*SUB 6.1.2.2*	*fSUB 6.1.2.2*
Multiplication	*MUL 6.1.2.3*	*fMUL 6.1.2.3*
Division	*DIV 6.1.2.4*	*fDIV 6.1.2.4*
Modulo operation	*MOD 6.1.2.5*	–
Power function	–	*fPOW 6.1.2.6*
Absolute value	–	*fABS 6.1.2.7*

6.1.2.1 Addition (ADD, fADD)



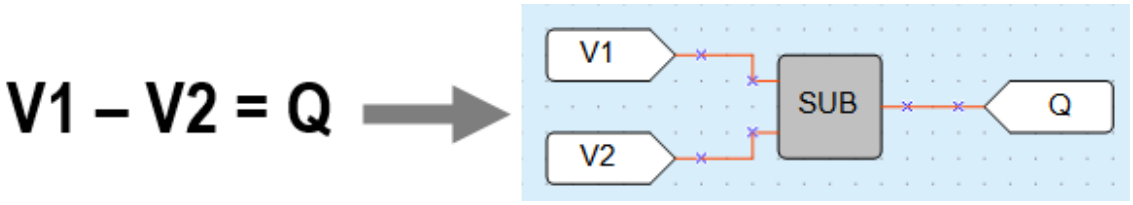
The function ADD operates with INT variables, while the function fADD operates with REAL variables. The output value Q is the sum of the input values.

Example:



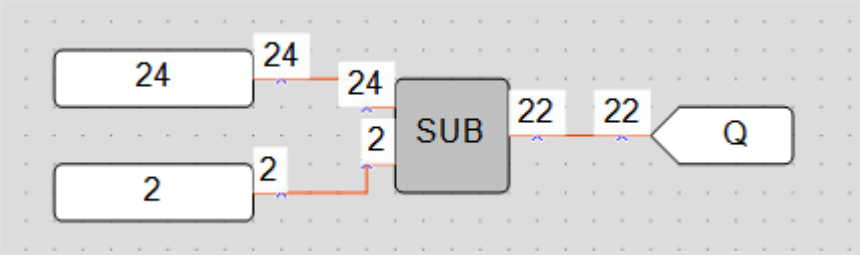
The output value may not exceed 4294967295 (32 bits). Otherwise the extra bits will be truncated.

6.1.2.2 Subtraction (SUB, fSUB)

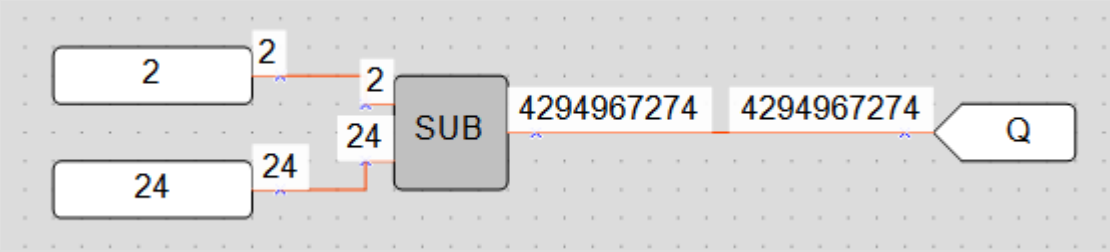


The function **SUB** operates with INT variables, while the function **fSUB** operates with REAL variables.
The output value **Q** is the result of subtraction of the value **I2** from the value **I1**.

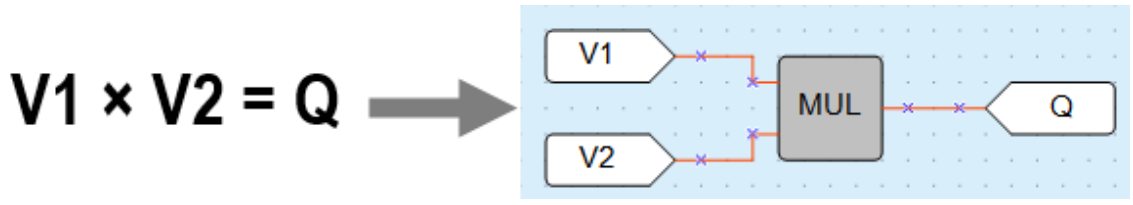
Example:



If the value **I1** is less than the value **I2**, the output is calculated as follows:
Q = I1 + 0x100000000 – I2
0x100000000 = 4294967296

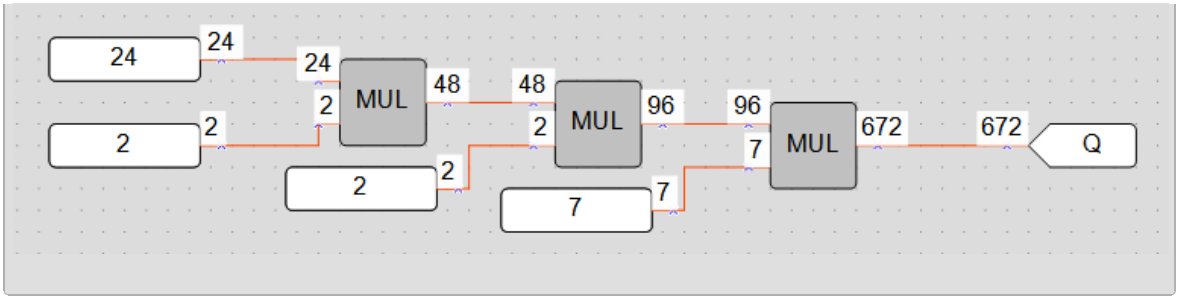


6.1.2.3 Multiplication (MUL, fMUL)



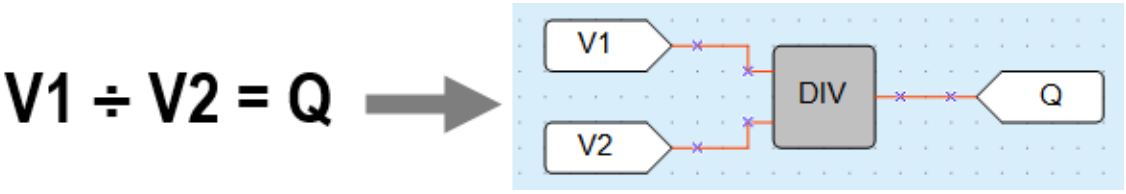
The function **MUL** operates with INT variables, while the function **fMUL** operates with REAL variables.
The output value **Q** is the product of the input values.

Example:



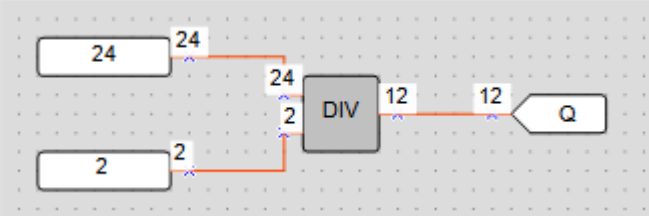
The output value may not exceed 4294967295 (32 bits). If it does happen, the extra bits will be truncated.

6.1.2.4 Division (DIV, fDIV)

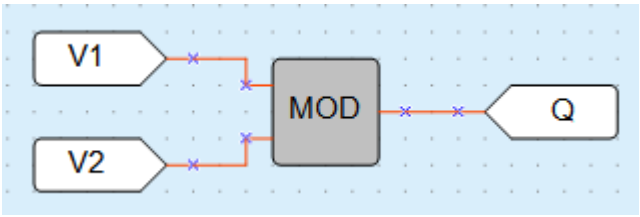


The function **DIV** operates with INT variables, the function **fDIV** operates with REAL variables.
The output value **Q** is the quotient of the input values, where the value **I1** is the dividend and the value **I2** is the divisor.
If the quotient is not an INT, it is rounded down to an INT.
In case of division by 0 the output value is 0xFFFFFFFF.

Example:

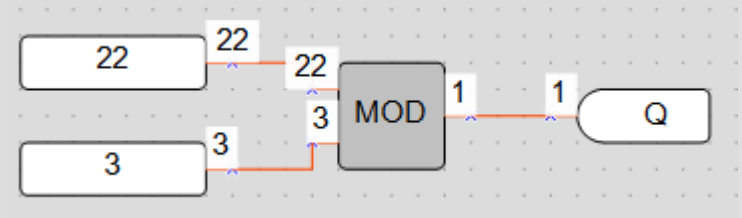


6.1.2.5 Modulo operator (MOD)

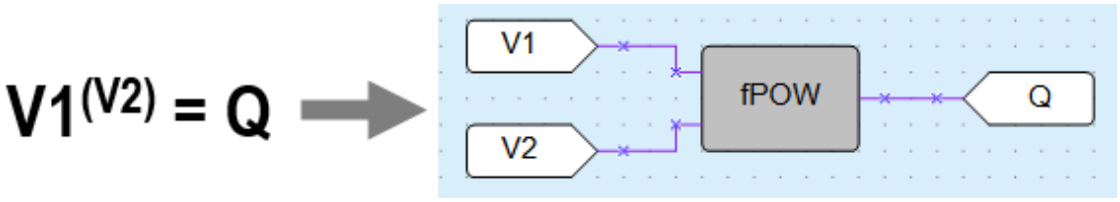


The function **MOD** operates with INT variables. The output **Q** is a remainder of the division of input values.

Example:

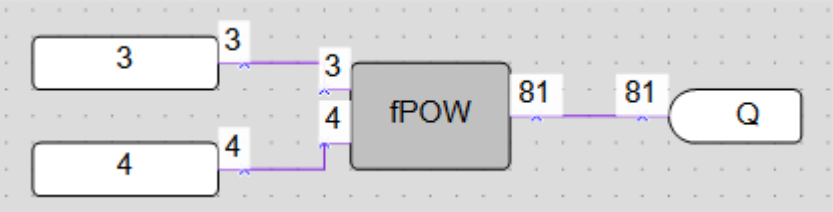


6.1.2.6 REAL-Power function (fPOW)

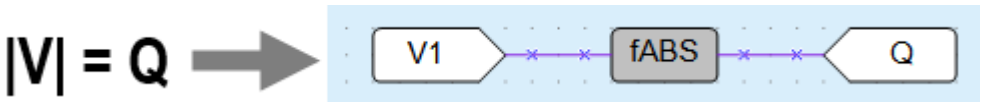


The function **fPOW** operates with REAL variables.
The output value **Q** is the value **I1** raised to the power of the value **I2**.

Example:

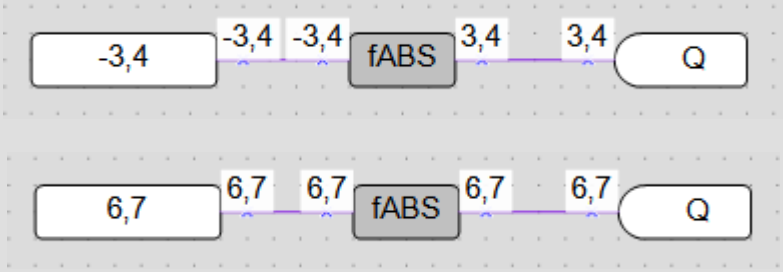


6.1.2.7 REAL-Absolute function (fABS)



The function **fABS** operates with REAL variables.
The output value **Q** is an absolute value of the input value.

Example:

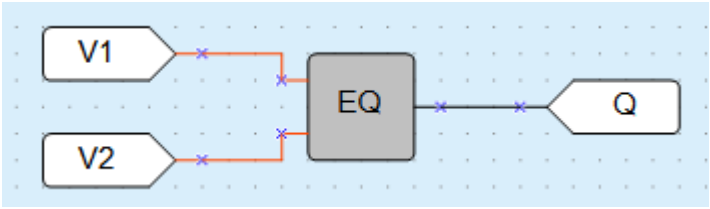


6.1.3 Relational operators

The relational operators are functions that test or define some kind of relation between two or more values.

– *Equal (EQ) 6.1.3.1*;
– *Greater than (GT, fGT) 6.1.3.2*;
– *Binary selection (SEL) 6.1.3.3*.

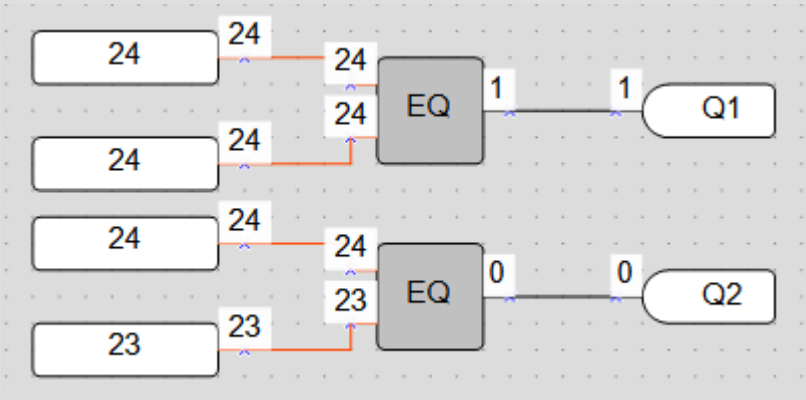
6.1.3.1 Equal (EQ)



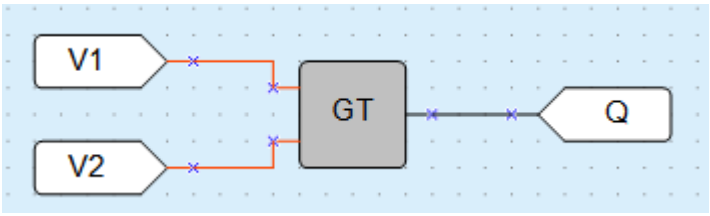
The function **EQ** operates with INT variables.
The output value **Q** is **True** if values **I1** and **I2** are equal.

– V1 = V2 → Q = 1;
– V1 > V2 → Q = 0;
– V1 < V2 → Q = 0.

Example:



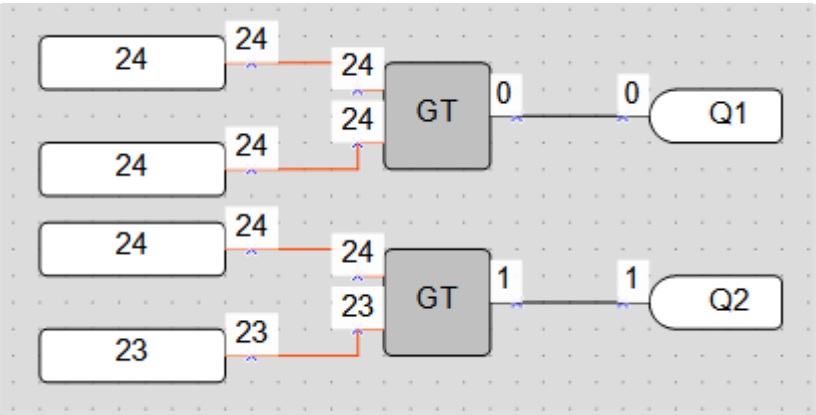
6.1.3.2 Greater than (GT, fGT)



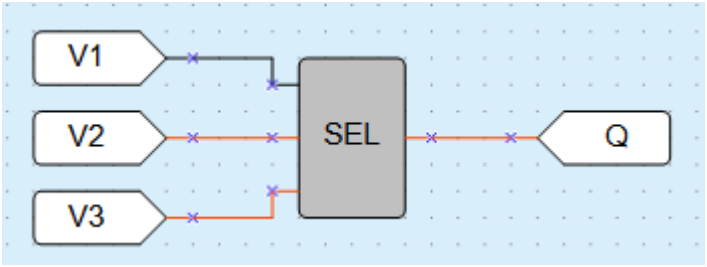
The function **GT** operates with INT variables, while the function **fGT** operates with REAL variables.
The output value **Q** is **True** if the value **I1** is greater than the value **I2**.

– V1 = V2 → Q = 0;
– V1 > V2 → Q = 1;
– V1 < V2 → Q = 0.

Example:



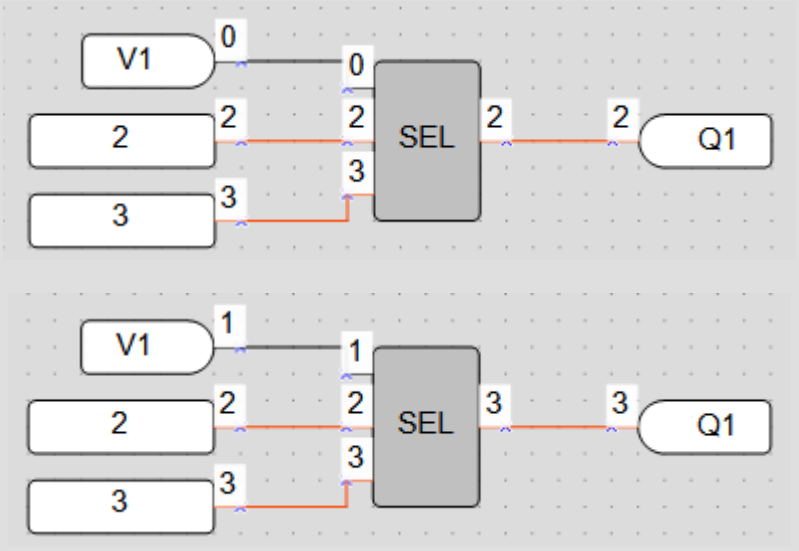
6.1.3.3 Binary selection (SEL, fSEL)



The function **SEL** operates with INT variables, the function **fSEL** operates with REAL variables.
If the value **I1** is **False**, the output value **Q** is set to the value **I2**, else to the value **I3**.

– V1 = 0 → Q = V2;
– V1 = 1 → Q = V3.

Example:



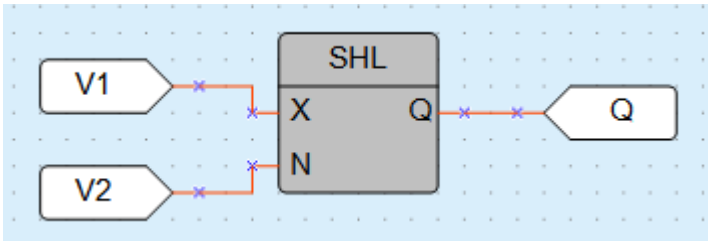
6.1.4 Bitshift operators

The bitshift operators treat a variable as a series of bits that can be moved (shifted) to the left or right.

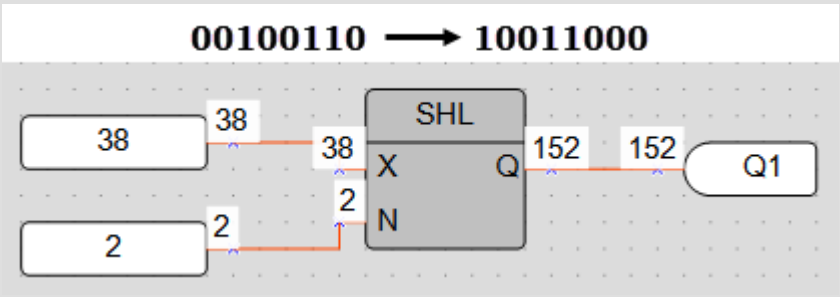
– *Shift register left (SHL) 6.1.4.1*;
– *Shift register right (SHR) 6.1.4.2*.

6.1.4.1 Shift register left (SHL)

The function **SHL** operates with INT variables. It is used to shift all bits of the operand **X** to the left by the **N** number of bits; vacated bits are zero-filled. The result is set to the output **Q**.

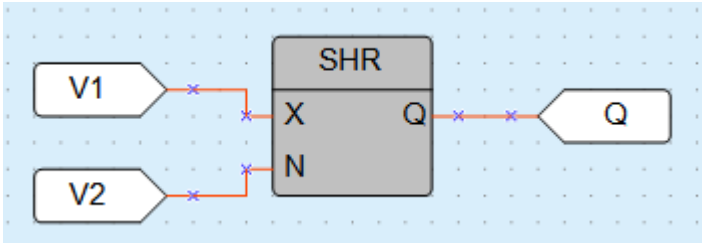


Example:
Left shift of the number 38 (decimal) = 00100110 (binary) by 2 bits

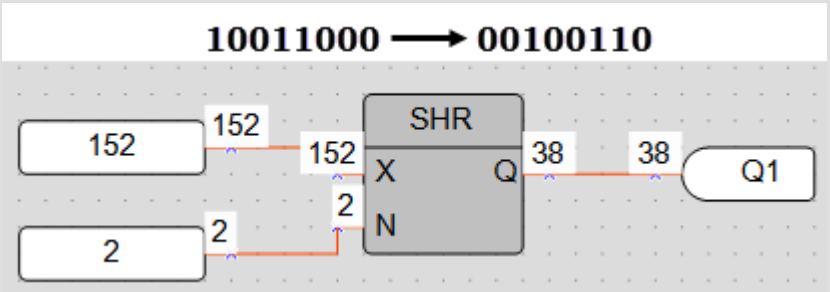


6.1.4.2 Shift register right (SHR)

The function **SHR** operates with INT variables. It is used to shift all bits of the operand **X** to the right by the **N** number of bits; vacated bits are zero-filled. The result is set to the output **Q**.



Example:
Right shift of the number 152 (decimal) = 10011000 (binary) by 2 bits

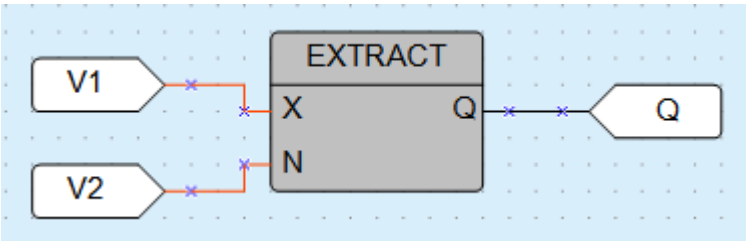


6.1.5 Bit operators

The bit operator treats a value as a series of bits to perform operations on one or more individual bits of an operand.

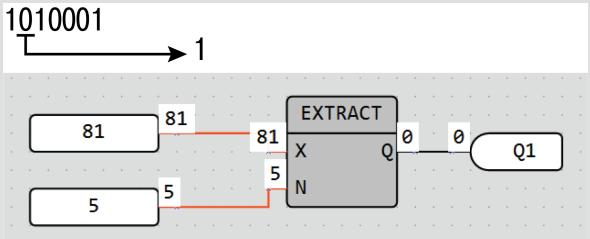
– *Read single bit (EXTRACT) 6.1.5.1*;
– *Set single bit (PUTBIT) 6.1.5.2*;
– *Decoder (DC32) 6.1.5.3*;
– *Encoder (CD32) 6.1.5.4*.

6.1.5.1 Read single bit(EXTRACT)

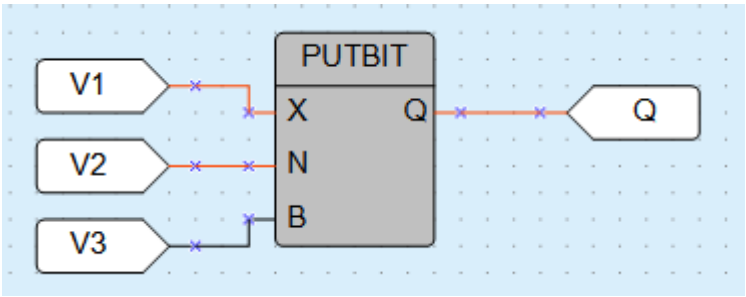


The output value **Q** (BOOL) of the function **EXTRACT** is the value of bit **N** (INT) in the operand **X** (INT). The bit numbering is zero-based.

Example:
Reading of the 5th bit from the number 81 (decimal) = 1010001 (binary):

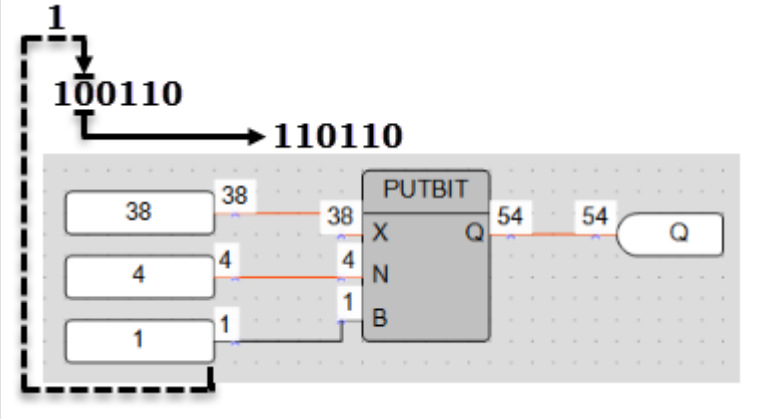


6.1.5.2 Set single bit (PUTBIT)

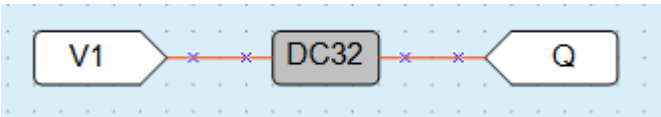


This output value **Q** (INT) is the value of the operand **X** (INT) where the bit **N** (INT) is set to the value at the input **B** (BOOL). The bit numbering is zero-based.

Example:
Setting of the 4th bit to 1 in the number 38 (decimal) = 100110 (binary):



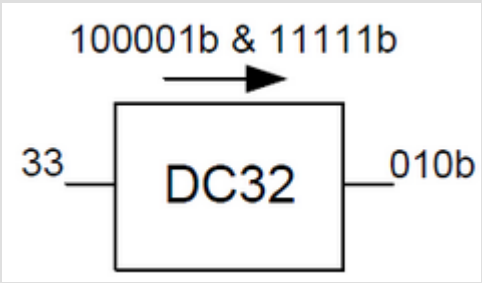
6.1.5.3 Decoder (DC32)



The decoder converts a binary code at the input to a position code at the output.
Decoding is carried out bitwise by the logical operation **AND** with the operand 0x1F (11111b).
Truth table:

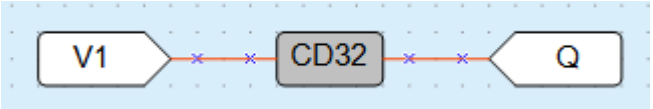
Binary code					Position code											
5	4	3	2	1	32	31	…	6	5	4	3	2	1
0	0	0	0	0	0	0		0	0	0	0	0	1
0	0	0	0	1	0	0		0	0	0	0	1	0
0	0	0	1	0	0	0		0	0	0	1	0	0
0	0	0	1	1	0	0		0	0	1	0	0	0
0	0	1	0	0	0	0		0	1	0	0	0	0
…					…			…					
1	1	1	0	1	0	0		0	0	0	0	0	0
1	1	1	1	0	0	1		0	0	0	0	0	0
1	1	1	1	1	1	0		0	0	0	0	0	0

Example:



6.1.5.4 Encoder

Encoder (CD32) is used to perform the operation of converting the positional code at the input into binary code at the output.



The encoder converts a position code at the input to a binary code at the output.
If there is more than one “1” bits in the position code, the encoder operates only with the most significant “1” bit.
Truth table:

Binary code					Position code											
5	4	3	2	1	32	31	…	6	5	4	3	2	1
0	0	0	0	0	0	0		0	0	0	0	0	1
0	0	0	0	1	0	0		0	0	0	0	1	0
0	0	0	1	0	0	0		0	0	0	1	0	0
0	0	0	1	1	0	0		0	0	1	0	0	0
0	0	1	0	0	0	0		0	1	0	0	0	0
…					…			…					
1	1	1	0	1	0	0		0	0	0	0	0	0
1	1	1	1	0	0	1		0	0	0	0	0	0
1	1	1	1	1	1	0		0	0	0	0	0	0

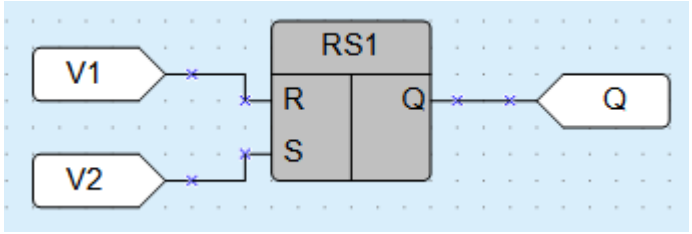
6.2 Function blocks

- *Triggers 6.2.1*
- *Timers 6.2.2*
- *Generators 6.2.3*
- *Counters 6.2.4*
- *Controllers 6.2.5*

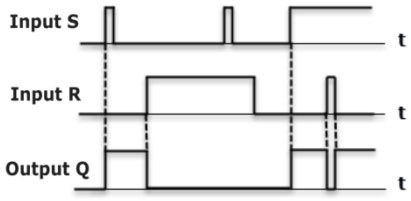
6.2.1 Triggers

- *RS trigger reset dominant (RS) 6.2.1.1*;
- *SR trigger set dominant (SR) 6.2.1.2*;
- *Rising edge (RTRIG) 6.2.1.3*;
- *Falling edge (FTRIG) 6.2.1.4*;
- *D-trigger (DTRIG) 6.2.1.5*.

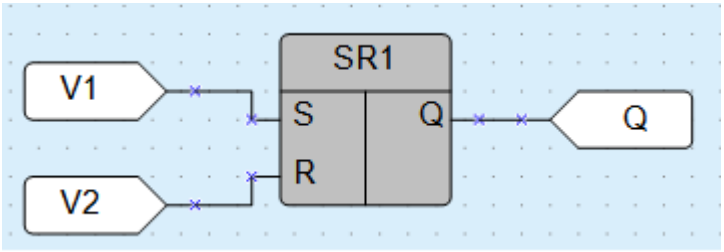
6.2.1.1 RS trigger reset dominant



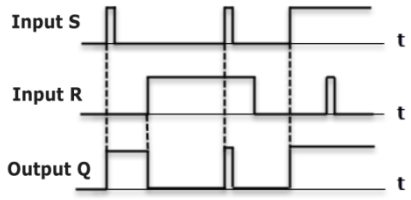
The output **Q** is **True** with a rising edge at the input **S** (Set) and **False** with a rising edge at the input **R** (Reset). The input **R** has higher priority.



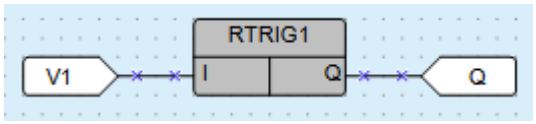
6.2.1.2 SR trigger set dominant



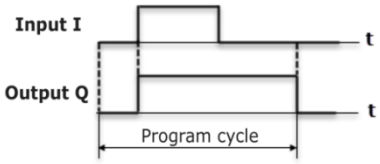
The output **Q** is **True** with a rising edge at the input **S** (Set) and **False** with a rising edge at the input **R** (Reset). The input **S** has higher priority.



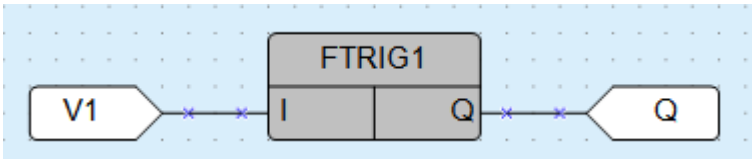
6.2.1.3 Rising edge (RTRIG)



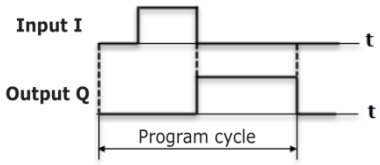
Detector for a rising edge appears.
The output **Q** remains **False** until a rising edge at the input **I**. As soon as the input **I** becomes **True**, the output becomes **True** and remains **True** for one program cycle.



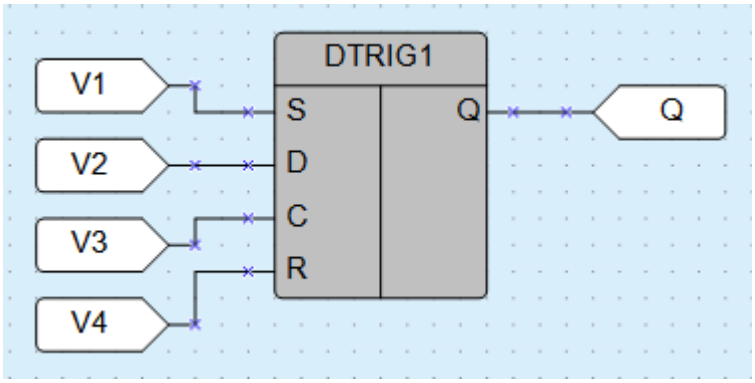
6.2.1.4 Falling edge (FTRIG)



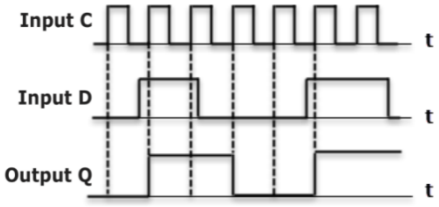
Detector for a falling edge.
The output **Q** remains **False** until a falling edge at the input **I**. As soon as the input **I** becomes **False**, the output becomes **True** and remains **True** for one program cycle.



6.2.1.5 D-trigger (DTRIG)



D-trigger generates a pulse at the output **Q** with the pulse duration specified at the input **D** and synchronized with the clock pulse at the input **C**.
If the input **D** is **True**, the output **Q** becomes **True** with a rising edge of the clock pulse at the input **C**.
If the input **D** is **False**, the output **Q** becomes **False** with a rising edge of the clock pulse at the input **C**.

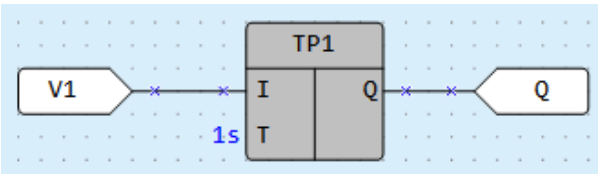


The output **Q** can be forced to set to **True** with a rising edge at the input **S** (Set) and forced to reset to **False** with a rising edge at the input **R** (Reset), regardless of the states of the inputs **C** and **D**. The input **R** has higher priority.

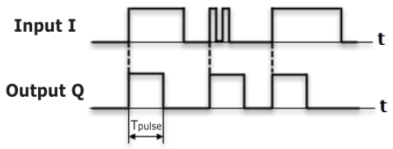
6.2.2 Timers

- *Pulse (TP) 6.2.2.1*;
- *ON-delay timer (TON) 6.2.2.2*;
- *OFF-delay timer (TOF) 6.2.2.3*;
- *Timer (CLOCK) 6.2.2.4*;
- *Weekly timer (CLOCKWEEK) 6.2.2.5*.

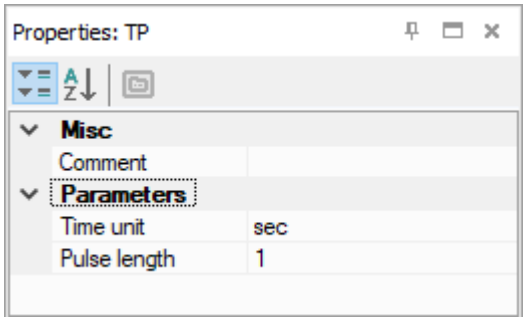
6.2.2.1 Pulse (TP)



The block **TP** is used to generate one output pulse with the specified pulse duration.

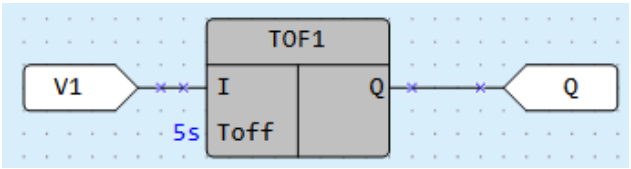


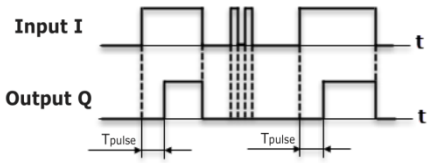
The output **Q** becomes **True** with a rising edge at the input **I** for the time specified at the input **T**. During this time, the output **Q** remains **True** regardless of the signal change at the input **I**. The output **Q** is reset to **False** with the end of pulse.
The pulse duration and the time unit can be set in Property Box.



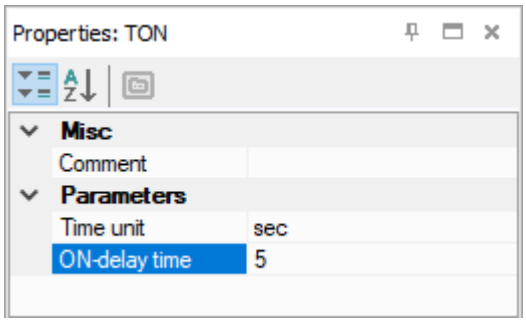
Time range: 0…4147200000 ms or 48 days.

6.2.2.2 ON-delay timer (TON)



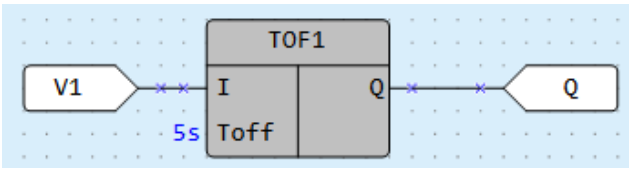


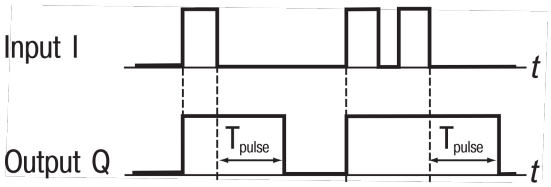
The output **Q** = **False** if the input **I** = **False**. The delay time specified at the input **TON** starts with a rising edge at the input **I**. When the time **TON** is elapsed, the output **Q** becomes **True** and remains **False** until a falling edge appears at the input **I**. Input changes shorter than **TON** are ignored.
The delay time and the time unit can be set in Property Box.



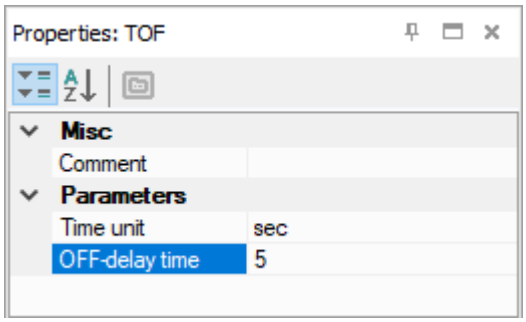
Time range: 0…4147200000 ms or 48 days.

6.2.2.3 OFF-delay timer (TOF)



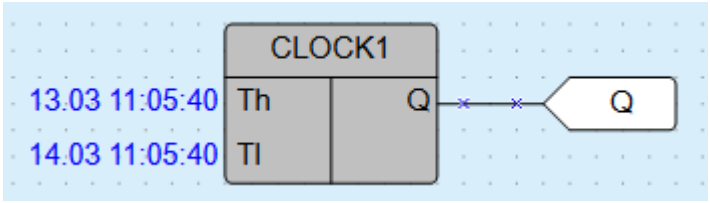


The output **Q** = **False** if the input **I** = **False**. The delay time specified at the input **TOFF** starts with a falling edge at the input **I**. When the time **TOFF** is elapsed, the output **Q** becomes **False** and remains **False** until a rising edge appears at the input **I**. Input changes shorter than **TOFF** are ignored.
The delay time and the time unit can be set in Property Box.

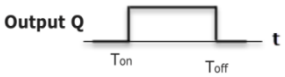


Time range: 0…4147200000 ms or 48 days.

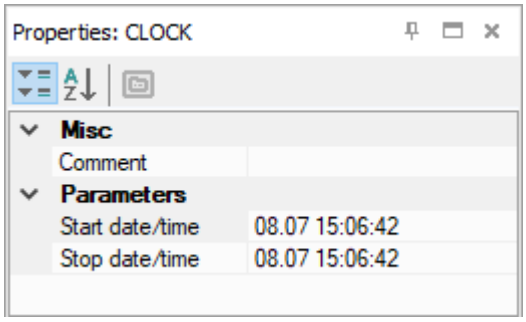
6.2.2.4 Timer (CLOCK)



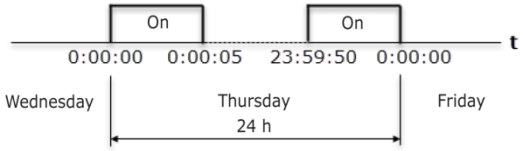
The block **CLOCK** is an interval timer controlled by a real-time clock.



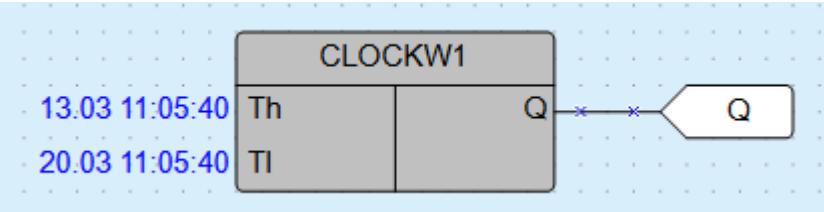
The times **TH** and **TL** can be set in Property Box.



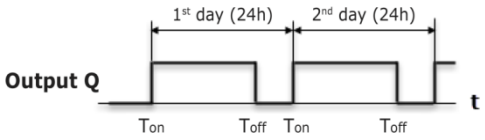
Time range: from 0.00 seconds to 24 hours.
If **TH** < **TL**, the state of the output **Q** is as follows:



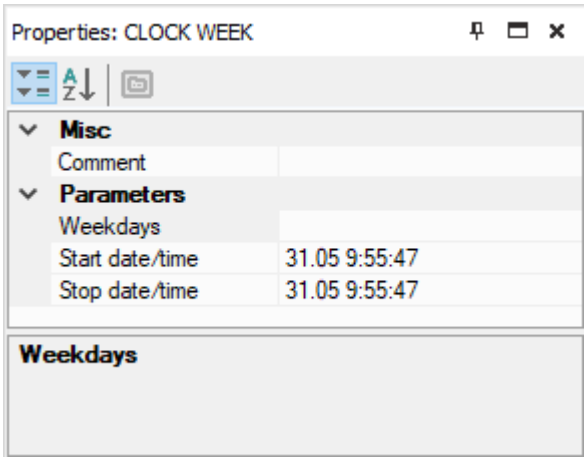
6.2.2.5 Weekly timer (CLOCKWEEK)



The block **CLOCKWEEK** is an interval timer with the parameter ***Weekdays*** controlled by a real-time clock.



The times **TH** and **TL** can be set in Property Box.

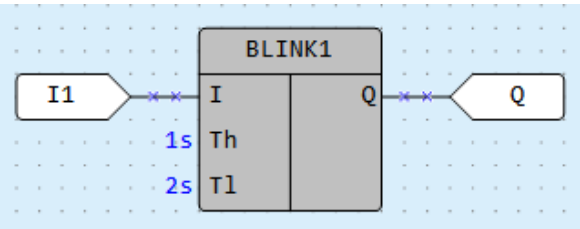


Time range: from 0.00 seconds to 24 hours.

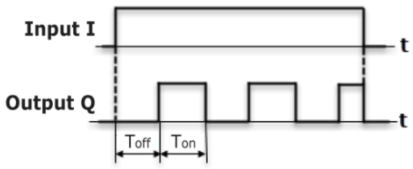
6.2.3 Generators

– *Pulse generator (BLINK) 6.2.3.1*.

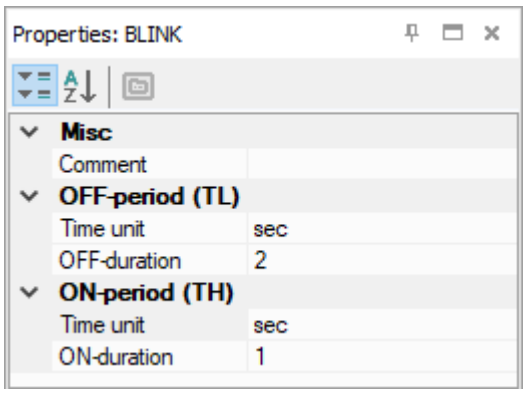
6.2.3.1 Pulse generator (BLINK)



If the input **I** becomes **True**, the block **BLINK** generates a square wave on the output **Q** with a period of **TH** + **TL**, starting with an interval of the duration of **TL**, followed by a pulse of the duration of **TH**. It continues that way until the input **I** is **False**.



The times **TH** and **TL** and the time units can be set in Property Box.

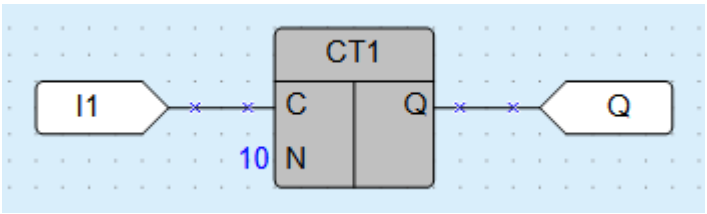


Time range: 0…4233600000 milliseconds or 49 days.

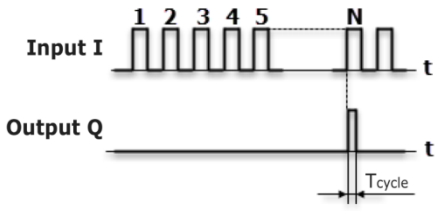
6.2.4 Counters

– *Threshold counter with self-reset (CT) 6.2.4.1*;
– *Universal counter (CTN) 6.2.4.2*;
– *Threshold counter (CTU) 6.2.4.3*.

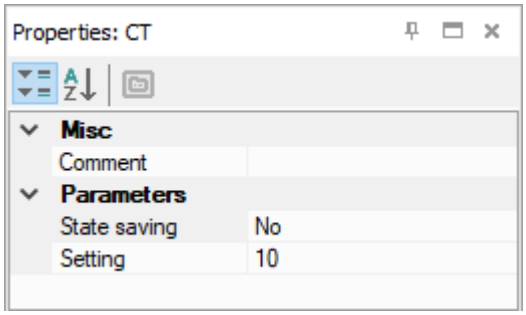
6.2.4.1 Threshold counter with self-reset (CT)



The output **Q** is of type BOOL. If the number of pulses counted on the input **C** exceeds the threshold (***Setting***) specified at the input **N**, the output **Q** becomes **True** and remains for one program cycle. The operation of the counter is explained in the diagram below.

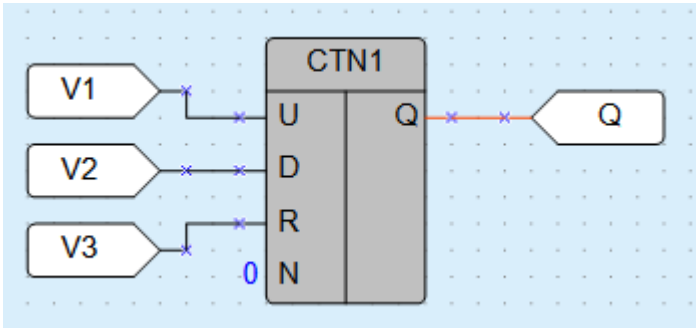


The parameters ***Setting*** and ***State saving*** can be set in Property Box.

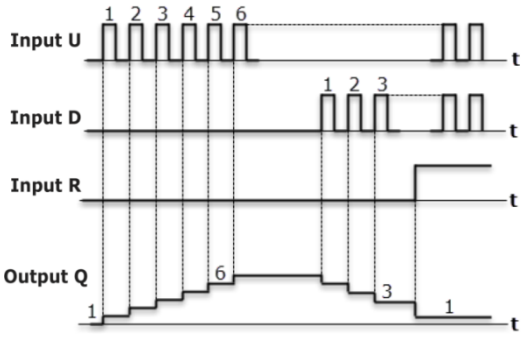


Threshold range: 0…65535.

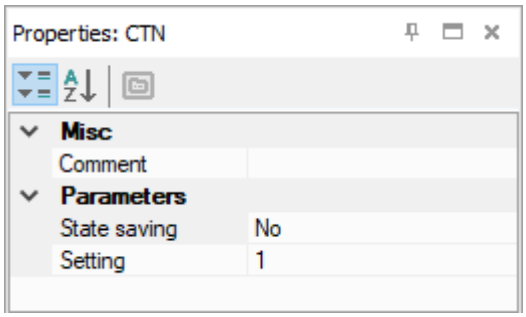
6.2.4.2 Universal counter (CTN)



The output **Q** is of type INT. A rising edge at the input **U** increases the value at the output **Q** by 1. A rising edge at the input **D** decreases the value at the output **Q** by 1.
If the input **R** = **True**, the output **Q** becomes the value ***Setting*** at the input **N**.

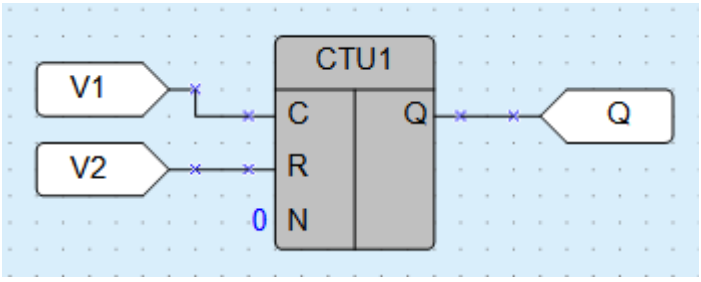


The input **U** has higher priority than the input **D**.
The parameters ***Setting*** and ***State saving*** can be set in Property Box.

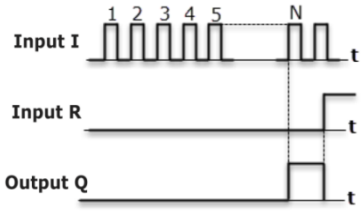


Setting range: 0…65535.
If ***State saving*** = ***Yes***, the state of the counter is permanently stored in the non-volatile memory.

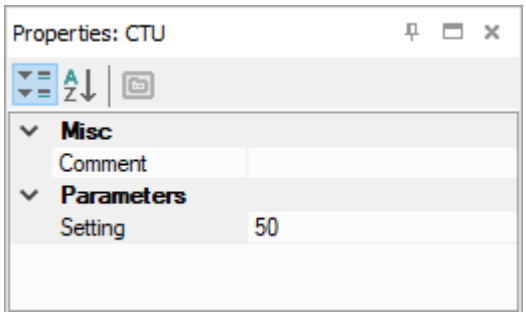
6.2.4.3 Threshold counter (CTU)



The output **Q** is of type Boolean. If the number of pulses counted on the input C exceeds the threshold (***Setting***) specified at the input **N**, the output **Q** becomes **True** and remains **True** until a rising edge at the input **R**. The input R has higher priority than the input **C**.
The operation of the counter is explained in the diagram below.



The parameter ***Setting*** can be set in Property Box.

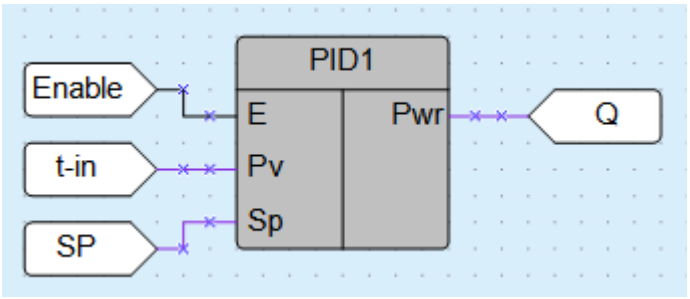


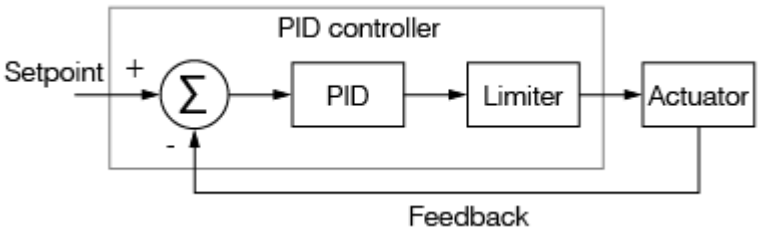
Threshold range: 0…65535.

6.2.5 Analog

– *PID-controller (PID) 6.2.5.1*

6.2.5.1 PID-controller (PID)





The function block **PID** is used for implementation of the proportional-integral-derivative control.

Table 6.1 PID block inputs/outputs

Name	Type	I/O	Description	Values
E	BOOL	I	Enable control (0 = Off, 1 = On). If disabled, the parameter ***Pwr*** takes the value of the parameter ***Output safe state***.	0 – Off 1 – On
Pv	REAL	I	Process value	
Sp	REAL	I	Setpoint	
Pwr	REAL	O	Output power, %	0…100

Table 6.2 PID block parameters

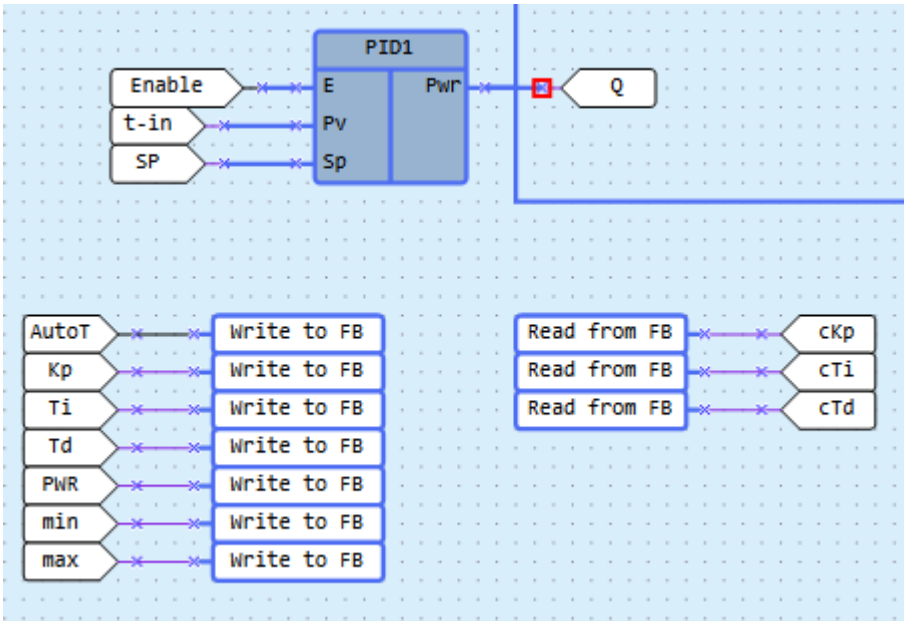
Name	Type	Description	Values	Access		
				Property Box	Write ToFB	Read From-FB
Control mode	BOOL	0 – Heating 1 – Cooling	0/1	X	X	
Output safe state	REAL	Output value when control is disabled, %	0…100	X	X	
Kp	REAL	Proportional gain, multiplication factor for proportional control	0…100	X	X	
Ti (s)	REAL	Integral time, time constant for integral control in seconds	–3,402823E +38… 3,402823E+38	X	X	
Td (s)	REAL	Derivative time, time constant for derivative control in seconds	–3,402823E +38… 3,402823E+38	X	X	
Output max.	REAL	Output upper limit, % (default 80 %)	0…100	X	X	

Name	Type	Description	Values	Access		
				Property Box	Write ToFB	Read From-FB
Output min.	REAL	Output lower limit, % (default 20 %)	0…100	X	X	
Start AT	BOOL	0 – stop auto-tuning 1 – start auto-tuning	0/1		X	
AT completed	BOOL	Flag: 0 – auto-tuning stopped 1 – auto-tuning started	0/1			X
Kp calculated	REAL	Calculated proportional gain	–3,402823E +38… 3,402823E+38			X
Ti calculated	REAL	Calculated integral time	–3,402823E +38… 3,402823E+38			X
Td calculated	REAL	Calculated derivative time	–3,402823E +38… 3,402823E+38			X

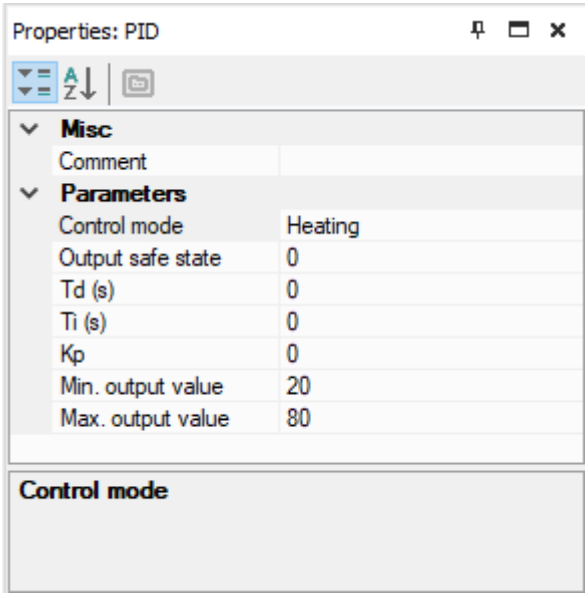
Tuning of a control loop is the adjustment of its control parameters (***Kp***, ***Ti***, ***Td***) to the optimal values for the desired control response.

Auto-tuning

Programmable loop tuning can be performed using the blocks *WriteToFB* W and *ReadFromFB* R *3.3.6*.
To write the parameters, use the block WriteToFB or Property Box.
To read the parameters, use the block ReadFromFB.



To use auto-tuning, add the block WriteToFB to the circuit program and set the reference to the parameter ***Start AT*** of the PID block.
To start the auto-tuning, enable control (***E*** = **1**) and set the parameter ***Start AT*** = **1**.

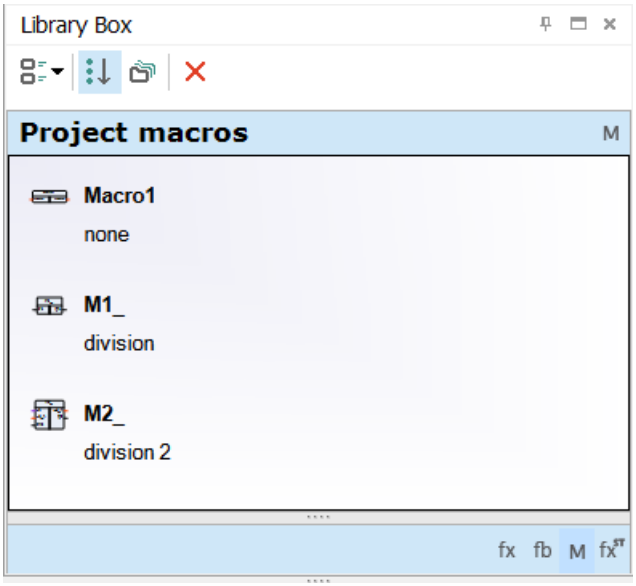


Upon completion of the auto-tuning, the new values of the parameters ***Kp***, ***Ti*** and ***Td*** are calculated and the flag ***AT completed*** becomes **1**.
If ***Start AT*** = **0**, the flag ***AT completed*** = **0** as well.
If you set ***Start AT*** = **0** before the completion of auto-tuning, the auto-tuning is stopped, the flag ***AT completed*** becomes **0** and no new coefficients are calculated.
During the auto-tuning, a test signal limited by parameters ***Output max.*** and ***Output min.*** is applied to the output **Pwr**.

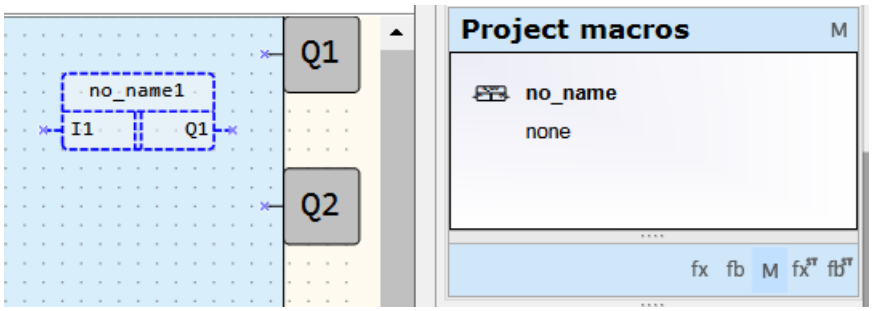
NOTE
*If the maximum gain is not sufficient to reach the setpoint, the auto-tuning cannot be completed and will continue until it is stopped with **Start AT** = 0.*

6.3 Project macros

Project Macros section contains macros *created by the user 3.11* or downloaded from Online Database using *Component Manager 3.10*.



To add a macro to a project, drag-and-drop the macro from the *Library Box 2.3* to the workspace.



To open the project macro in the separate workspace for editing, select it in the workspace or in the library and use the item Edit macro in the macro context menu.

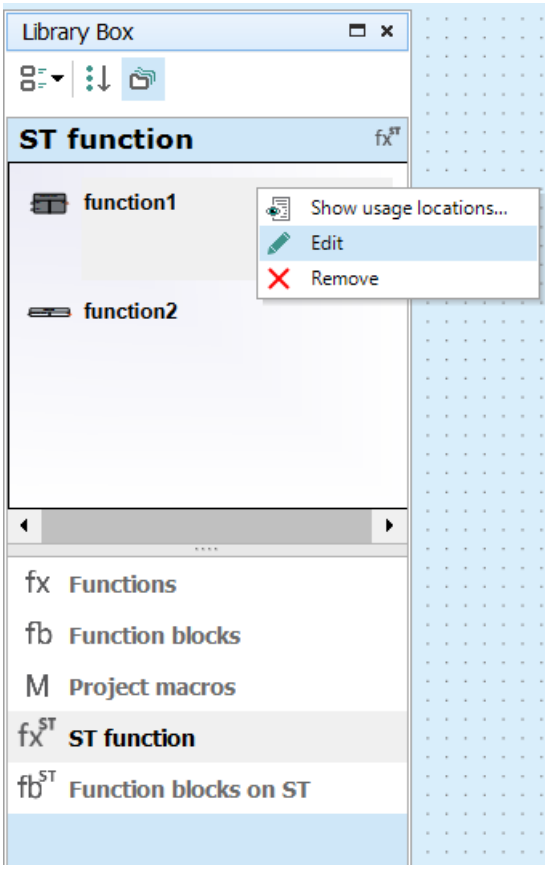
To remove the macro from the Library Box, select the macro and click the ✕ icon in the panel toolbar.

For details about macros creation, development and handling see section *Macro development 3.11*.

6.4 ST functions

NOTE
For devices of the PR100(M02), PR102, PR200, PR103 and SMI120, creation of user functions in the ST language is available.

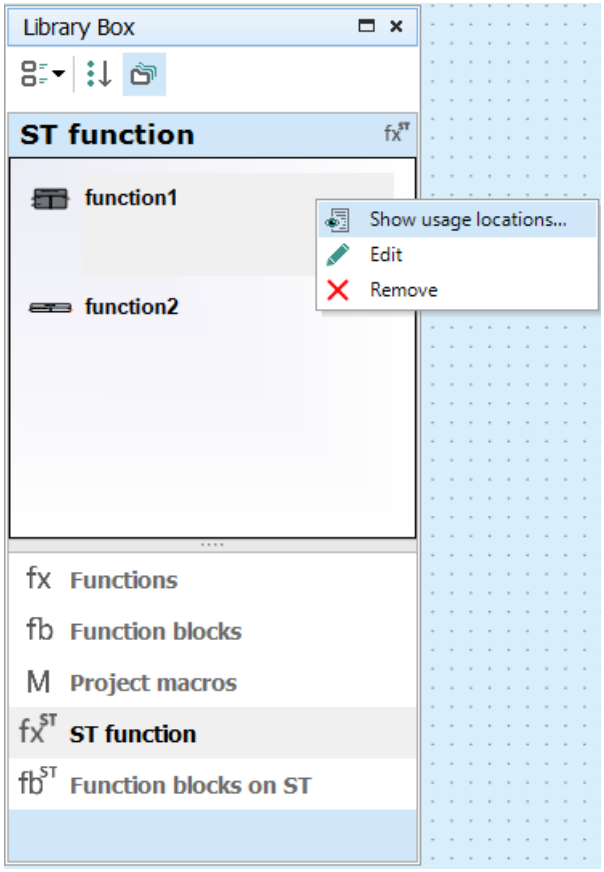
If you have *created ST functions 3.12* in your project, they will be available in the Library Box.



Usage locations

To view all places where the function is used:

1. Right-click on the function name
2. Select item ***Show usage location…***.



The **Function usage locations** panel will open at the bottom of the window, displaying where the function is used in the diagram and in the function editor.

3. Right-click on the line that indicates where the function is used.
4. Select item **To usage location**.

The focus will shift to where the function is used in the diagram or in the function editor.

NOTE
Double-click leads to the same result.

If the places where functions are used have changed while working with the program, you should update the **Function usage locations** panel:

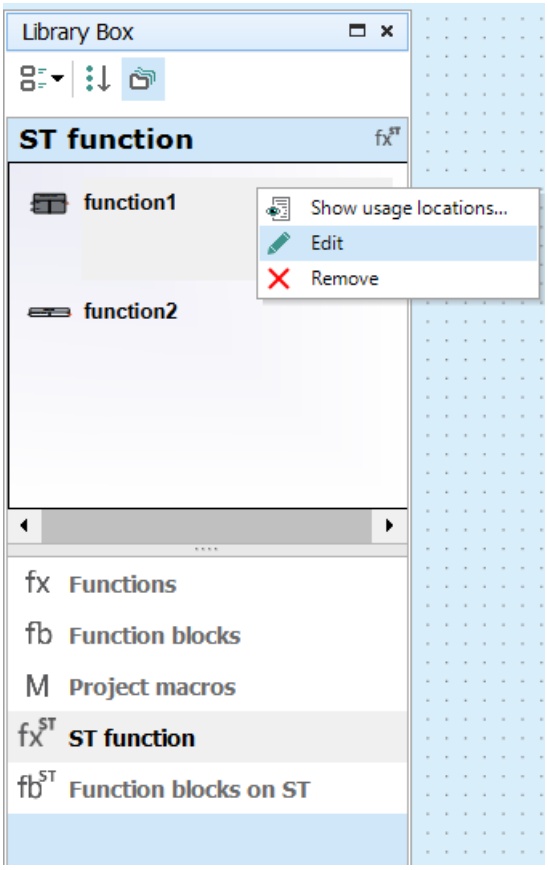
1. Right-click on any line of the panel.

2. Select the item ***Refresh usage locations***.

Go to the function editor

To go to the *function editor 3.12*:

1. Right-click on the function name.
2. Select the item ***Edit***.

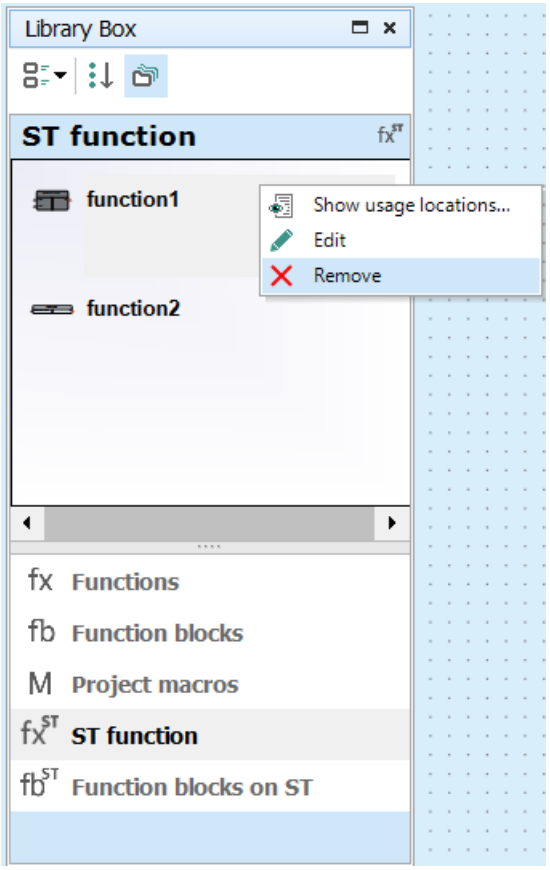


The *function editor 3.12* will open.

Delete function

To remove a function from a project:

1. Right-click on the function name.
2. Select the item **Remove**.

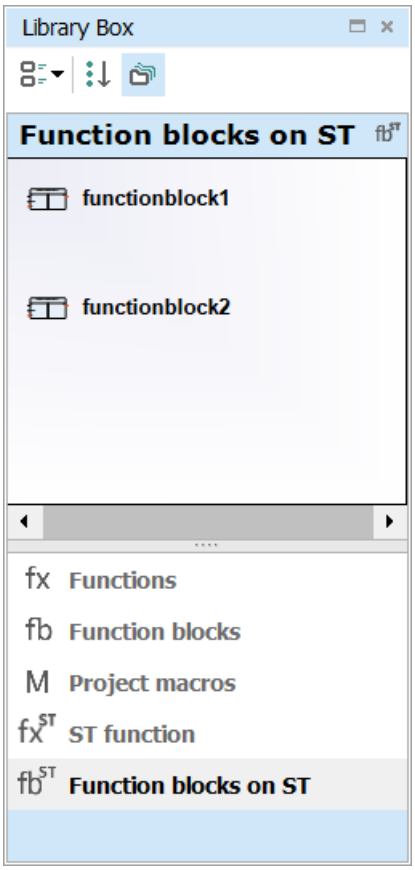


NOTICE
If the function is used in the diagram and/or in other functions, deletion may result in compilation errors.

6.5 ST function block

NOTE
For devices of the PR100(M02), PR102, PR200, PR103 and SMI120, creation of user function blocks in the ST language is available.

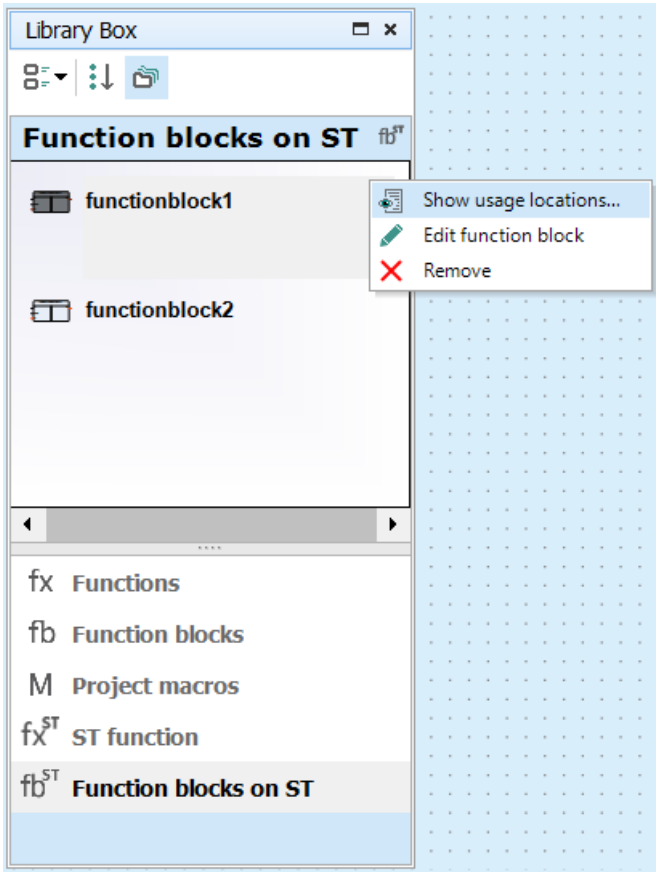
If ST function blocks are created in the project, they will be available in the component library.



Usage location

To view all places where the function is used:

1. Right-click on the name of the function block.
2. Select the item **Show usage locations...**.



The **Function block usage locations** panel will open at the bottom of the window, displaying where the function is used in the main program and in the editor.

3. Right-click on the line with the location where the function block is used.
4. Select the item **To usage location**.

The focus will shift to where the function block is used on the diagram or in the editor.

NOTE
Double-click leads to the same result.

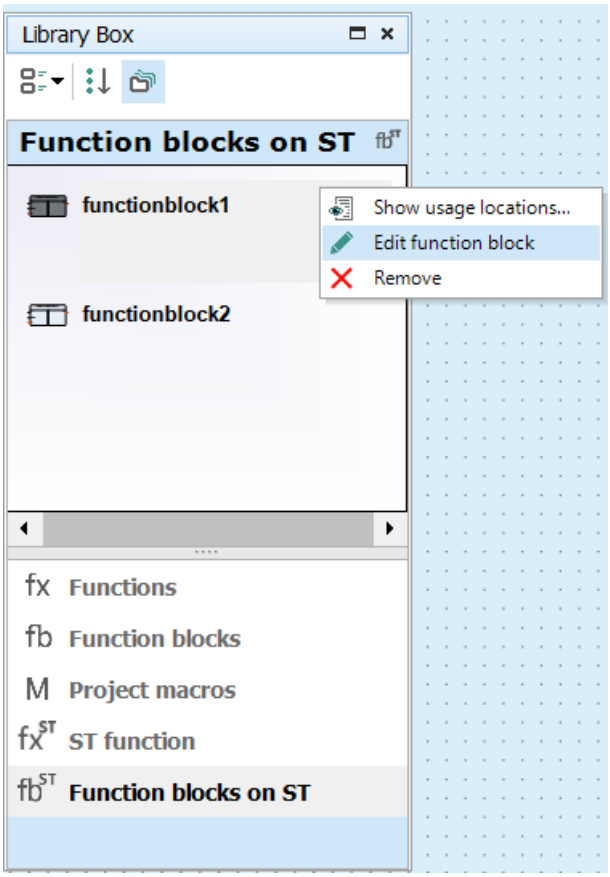
If the places where function blocks are used have changed while working with the program, you should update the **Function block usage location** panel:

1. Right-click on any line of the panel.
2. Select the item **Refresh usage locations**.

Go to the function block editor

To go to the *function block editor 3.13*:

1. Right-click on the function block name.
2. Select the item **Edit**.

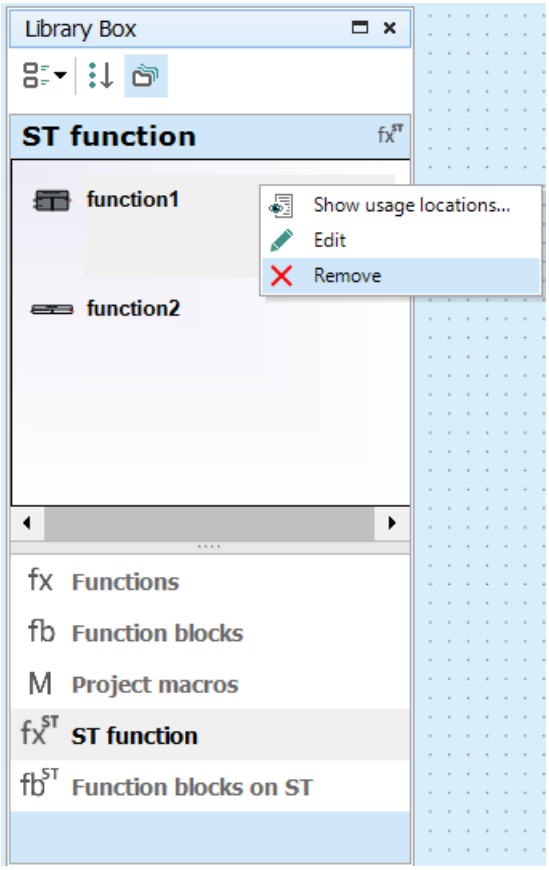


The *function block editor 3.13* will open.

Delete function block

To delete a function block:

1. Right-click the function block name.
2. Select the item **Remove**.



NOTICE
If the function block is used in the main program and/or in other functions, deletion may result in compilation errors.

6.6 Display elements

If the workspace with a display form is active, only display elements are available in Library Box. With these blocks, the information displayed on the device display can be controlled. The display elements can be placed within the display form by drag-and-drop. The following elements are available:

– *Text box 6.6.1*;
– *I/O box (INT/REAL) 6.6.2*;
– *I/O box (BOOL) 6.6.3*;
– *Dynamic box 6.6.4*;
– *Combobox 6.6.5*.

Use Property Box to customize an element.
Common parameters for all elements:

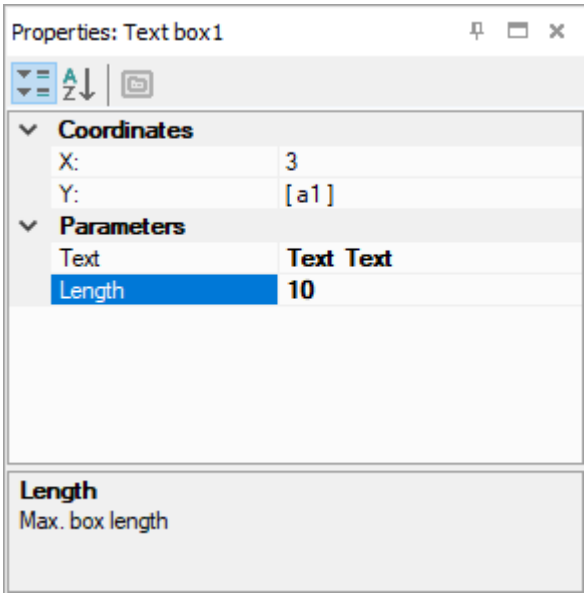
– ***Coordinate X*** – the position of the first (left) character placeholder of the element from the left edge of the form (from 0 to 15).
– ***Coordinate Y*** – the position of the first (left) character placeholder of the element from the upper edge of the form, depending on number of the rows in the form.
– There are two ways to determine coordinates: constant (default) or variable. To use a coordinate dependent on a variable, select the coordinate and open the list on the right of the input field.
  – ***Constant*** – specify the coordinates in Property Box or place the element within the form by drag-and-drop.
  – ***Variable*** – click ***Select*** to select an INT variable from the list and confirm with ***OK***. The display element will move according to the coordinate value controlled by the variable.
– ***Length*** – the number of reserved characters. The display element occupies one display row in height, its length can be from 1 to 16 characters.

6.6.1 Text box

Text box is used to display plain text.

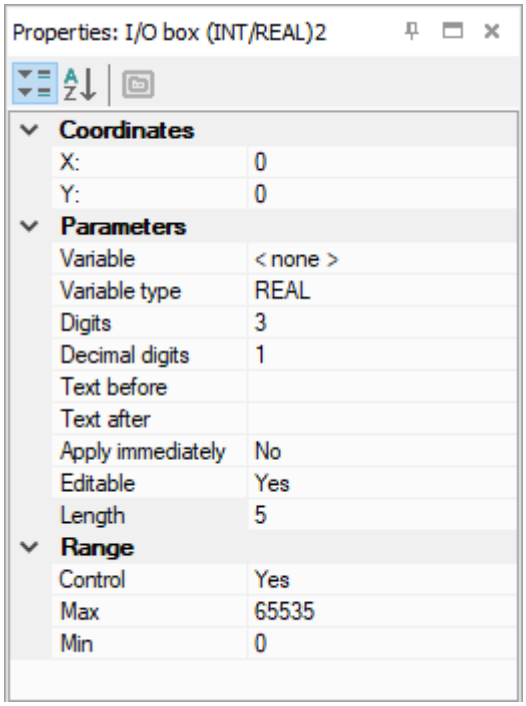
Parameters

Text – text to display. The parameter ***Length*** specifies the number of the reserved characters.



6.6.2 I/O box (INT/REAL)

I/O box (INT/REAL) is used to display a variable of type INT or REAL. The value of the variable can be changed with the device function buttons.



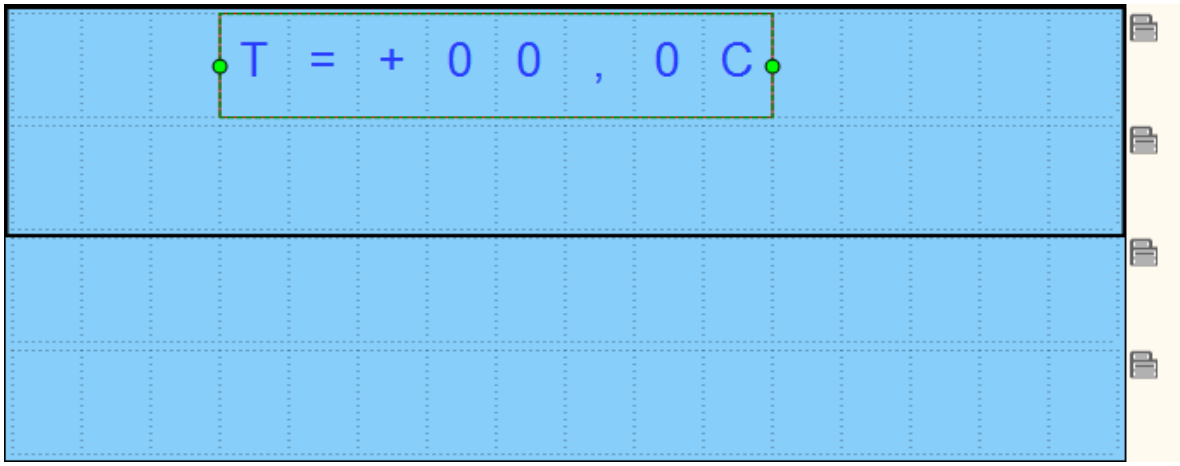
Parameters

– ***Variable*** — the reference to a variable. Use the icon «**…**» in the input field to select the variable.

- ***Data type*** — INT or REAL. If the variable has been already selected, its data type will be accepted.
- ***Digits*** — the total number of displayed digits.
- ***Decimal digits*** — the number of the characters after the decimal point: 0…6 characters or ***Auto*** for Auto-precision*.
- ***Text before*** — the text to the left of the displayed variable.
- ***Text after*** — the text to the right of the displayed variable.
- ***Editable*** — if Yes, the displayed value can be changed using the device function buttons. *An output variable should be selected. The option has no effect with an input variable.*
- ***Length*** — the total number of reserved characters including both the text before and after.

Range:
The group of parameters is used to limit the input value. If ***Editable*** = ***No***, the parameters of this group have no effect.

- ***Limit*** — if ***Yes***, the value entered using the device function buttons is limited by the user parameters ***Max*** and ***Min***, else it is limited only by the available memory area.
- ***Max*** — the maximum input value.
- ***Min*** — the minimum input value.

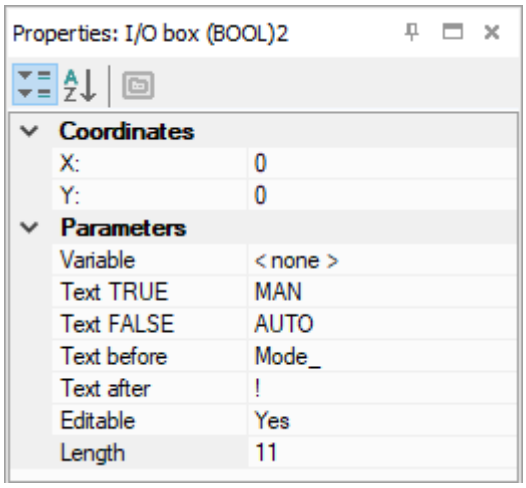


*** Auto-precision**
The option enables to display a REAL variable most precisely for the set number of reserved characters (parameter ***Digits***). To use the option, select in the workspace an I/O-Box display element with associated variable of REAL type and select ***Auto*** for the parameter ***Decimal digits*** in the Property Box.

Example:
To display the variable VAR1, 4 digits with Auto-precision are reserved. The value 1.546745 will be displayed rounded as 1.547. If the value will be changed to 110.478696, it will be displayed as 110.5.

6.6.3 I/O box (BOOL)

I/O box (BOOL) is used to display a variable of BOOL type. The value of the variable can be changed with the device function buttons.

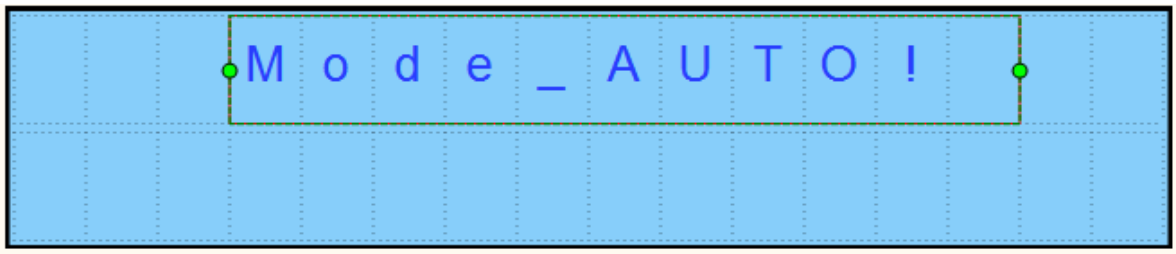


Parameters

- ***Variable*** — the reference to a variable. Use the icon «**…**» in the input field to select the variable.
- ***Text TRUE*** — the text displayed if the variable is ***True***.
- ***Text FALSE*** — the text displayed if the variable is ***False***.
- ***Text before*** — the text to the left of the displayed variable.
- ***Text after*** — the text to the right of the displayed variable.
- ***Editable*** — if ***Yes***, the displayed value can be changed using the device function buttons.

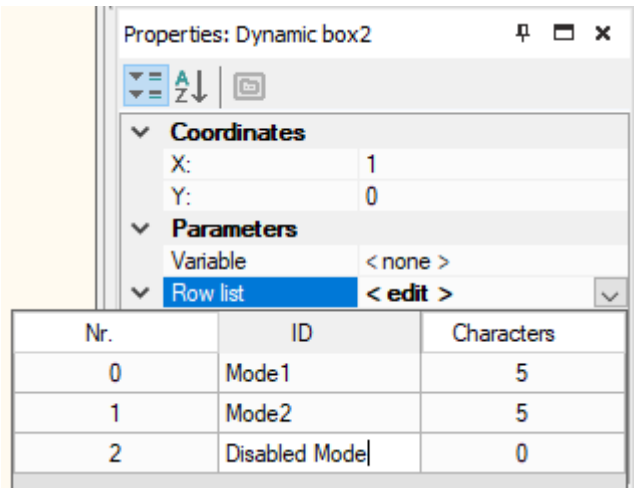
  NOTE
  An output variable should be selected. The option has no effect with an input variable.

- ***Length*** — the total number of reserved characters including both the text before and after.



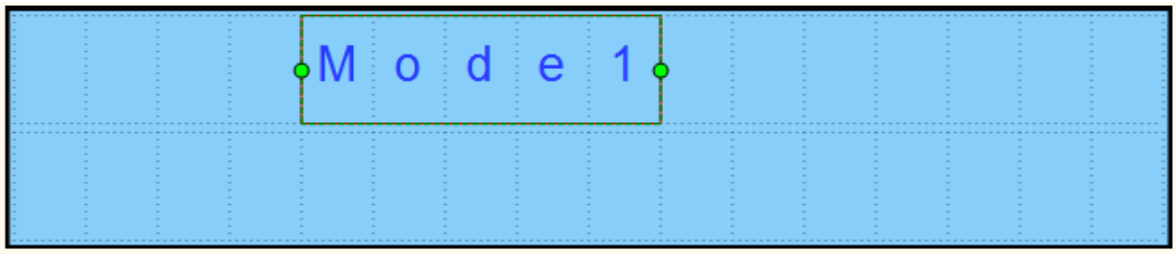
6.6.4 Dynamic box

Dynamic box is an output field. It is used to display one of the text rows from a list depending on a row ***ID***. The row ***ID*** is saved in a referenced variable of INT type.



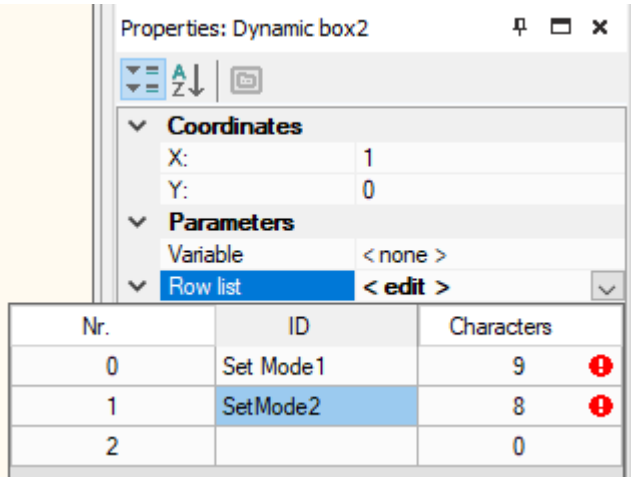
Parameters

- ***Variable*** — the reference to an integer project variable. To select the variable, click the «...» button and select from the *variable table 5*;
- ***Row list*** — the list with text rows. The ***Text*** from the row is displayed if the value of the referenced variable iequals to the row ***ID***. The column ***Characters*** shows the number of characters in the text. An exclamation mark is displayed near the number if the value of the parameter ***Length*** is exceeded.
- ***Length*** — the number of reserved characters.



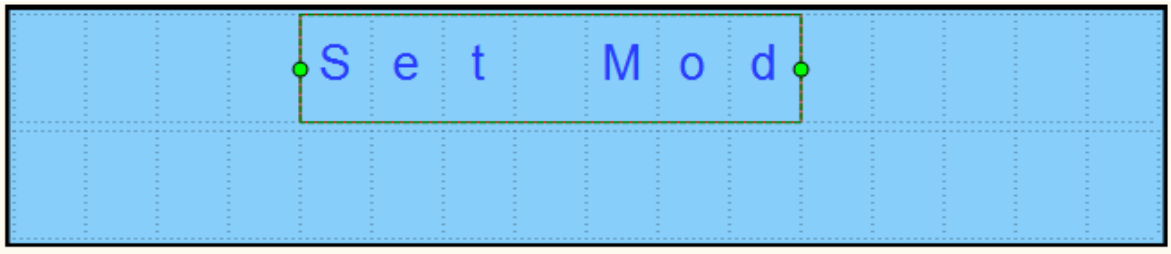
6.6.5 ComboBox

ComboBox is an input / output field. It is used to display one of the text rows from a list depending on a row ***ID***. The row ***ID*** is saved in a referenced variable of INT type. The ***ID*** can also be selected using the device function buttons.



Parameters

- ***Variable*** — the reference to a program variable. Use the icon «**...**» in the input field to select the variable.
- ***Row list*** — the table with text rows. The ***Text*** of the selected row is displayed and the row ***ID*** is saved in the referenced output variable. The column ***Characters*** shows the number of characters in the text. An exclamation mark is displayed near the number if the value of the parameter ***Length*** is exceeded.
- ***Length*** — the number of the reserved characters.

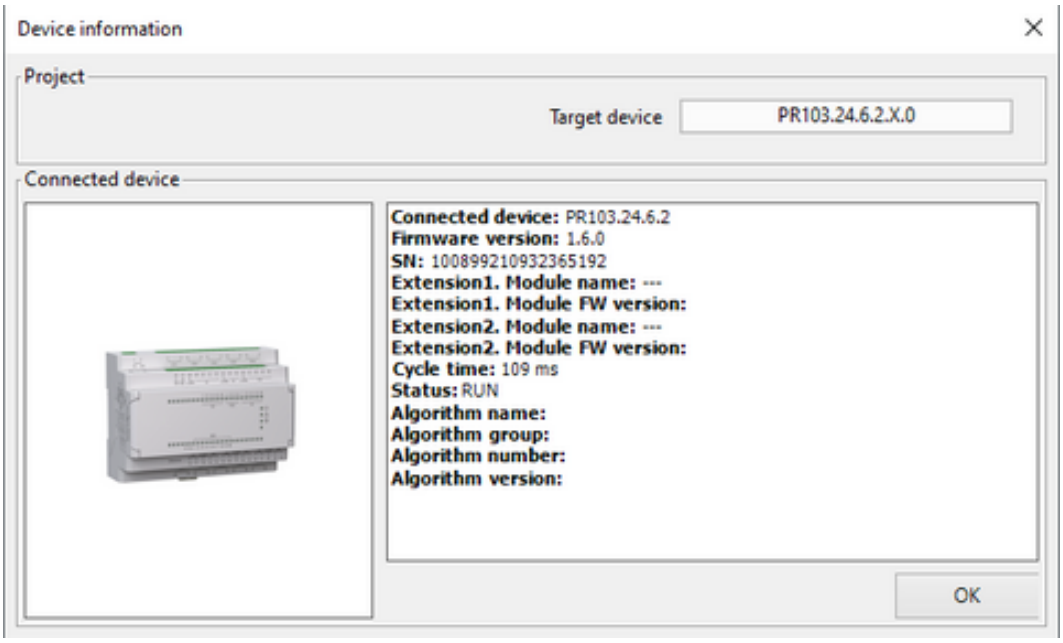


7 Device

This section describes the operating functions and configuration of the device:
– *Device information 7.1*
– *Cycle time 7.2*
– *Firmware update 7.3*;
– *Calibration 7.4*.

7.1 Device information

To view information about the software, the target device and the connected device use the menu item ***Device*** → ***Information...*** or the icon in the toolbar.
A window containing information about the connected device appears:



The window ***Device Information*** contains the following information:
Target device – the device for which the project was created
Connected device – the information about the device connected to the PC
Alternatively, the type of each output can be manually changed in ***Property Box*** in accordance with the hardware.

Information about the device on the new hardware platform

For devices on the new platform, the information displayed in the window differs.
Project information:
– ***Selected device model*** - the model and modification of the device selected when creating the project.

Information about the connected device:
– ***Device name*** – model and modification of the connected device
– ***Firmware version*** – firmware version of the connected device
– ***S/N*** – unique device identifier
– **PRM Slot. Module name** – model of the *extension module 4.4*, connected to the device
– **PRM Slot. Module Firmware Version** – firmware version of the *extension module 4.4*, connected to the device.

7.2 Cycle time

Cycle time is the time it takes to complete the operating cycle of the device, namely:
– polling the state of the physical inputs of the device and copying their values into memory cells
– program processing
– read/write program network variables
– writing the results of the program to the physical outputs of the device

The default cycle time is **1 ms**. The device adjusts the cycle time depending on the complexity of the program.
Conditions for increasing cycle time:
– the complexity of the algorithm increases (a large number of FBs and macros are involved)
– the program uses a large number of network variables
– the project uses a large number of data controls via the device display

The user cannot set the cycle time. If the device is equipped with a display, the current cycle time can be viewed in the system menu of the device. If the device is connected to a PC, the cycle time can be viewed in the *Device Information 7.1* window.

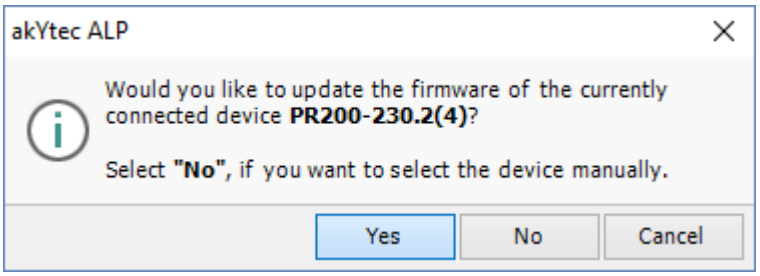
7.3 Firmware update / repair

If a new ALP version includes a new version of the firmware for the connected device or extension module, you will be prompted to update the firmware before uploading a user program to the device. No internet connection is needed. Click ***Yes*** to start the update.

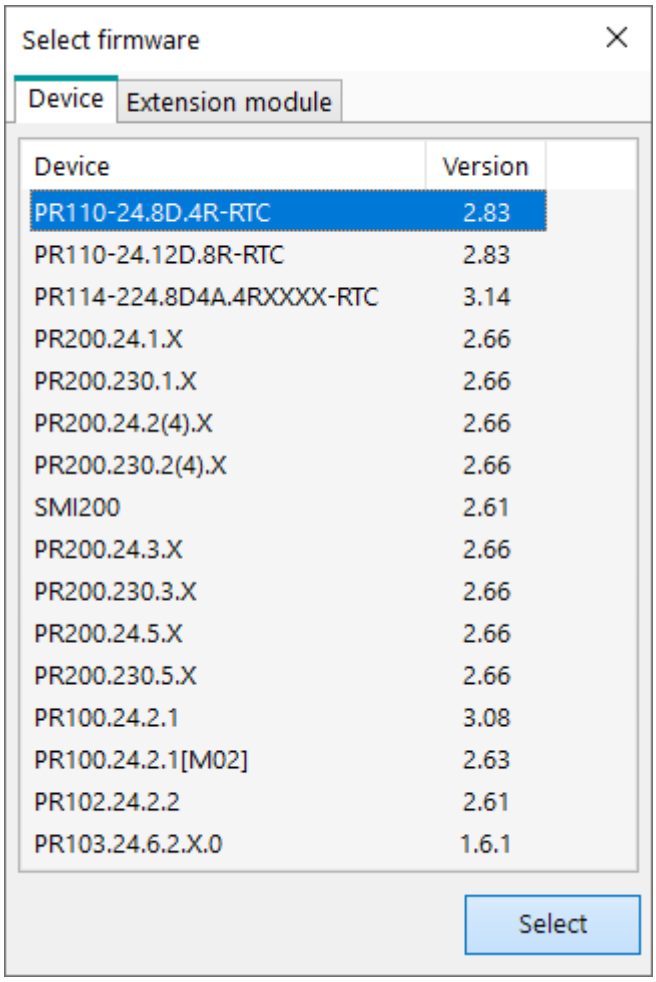
NOTE
Ensure the power supply of the device and extension modules (if any) and the safe connection between the PC, the device and the extension modules (if any) during the update process.

You can also update the firmware manually using the menu item ***Device > Firmware update***. This way the firmware can be repaired when the firmware damage is detected (see respective user guide, table “Error indication”).

NOTE
The user program will not be affected by firmware update.



If you select ***Yes***, the firmware of the currently connected and recognized device will be updated (repaired).
If you select ***No***, lists of devices and extension modules will be offered to select from. The opened window has two tabs: ***Device*** and ***Extension Module***. This way a forced firmware update can be made.
Click Select to confirm the selection and start the update (repair) process. The message about the update result is shown upon the update completion.



Forced firmware update / repair

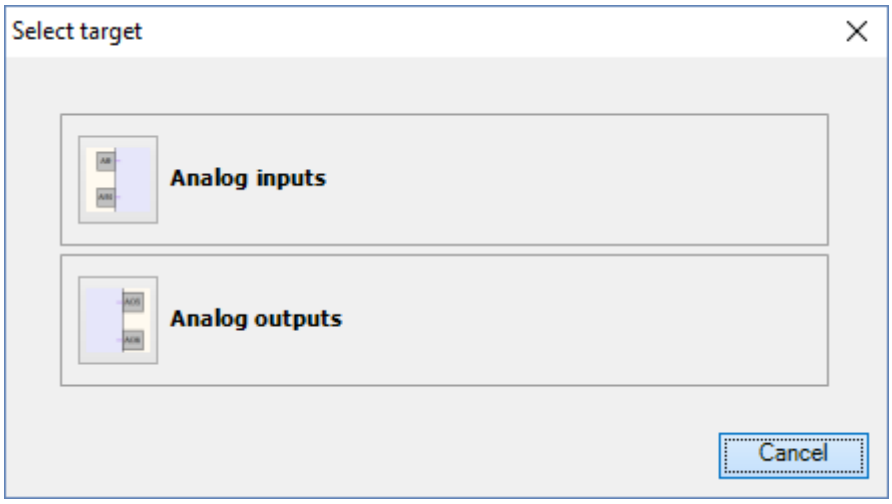
If the firmware is damaged (see respective user guide, table “Error indication”) and device automatic recognition is not possible, a forced firmware update should be used. Proceed as follows:

1. set the device in the forced download mode (see the device user guide)
2. select the menu item ***Device > Firmware update***, lists of devices and extension modules will be offered to select from
3. select the device (extension module)
4. click ***Select*** to confirm the selection and start the update (repair) process.

The message about the update result is shown upon the update completion.
If the device and the extension module have incompatible firmware versions and the user program is uploaded to the device without the extension module connected, this may lead to an expansion module error being displayed. To fix the error, use forced firmware update for the expansion module as described, skipping step 1.

7.4 Calibration

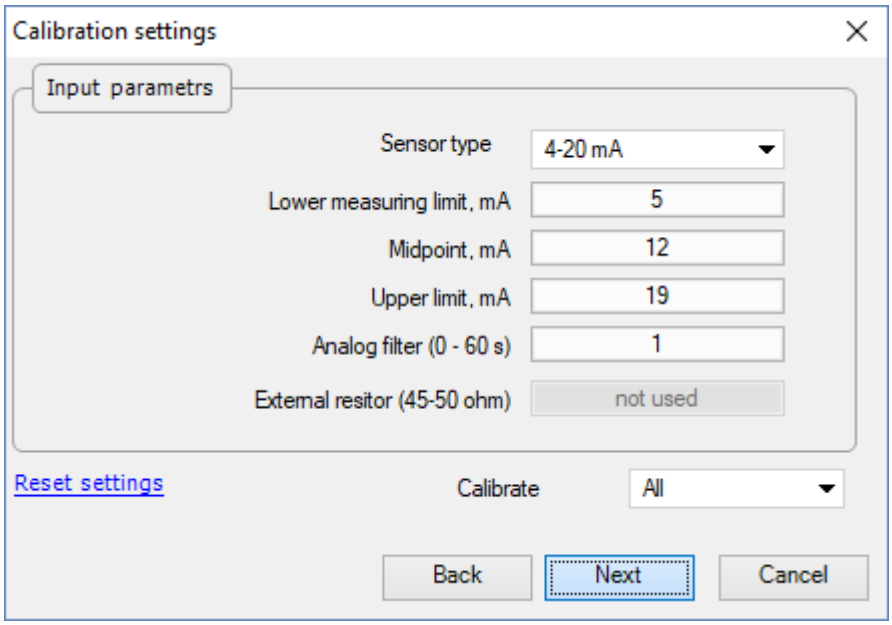
Only general information about calibration of analog inputs or outputs is given in this section. For detailed information about calibration refer to the user guide of the device.
If calibration of analog inputs or outputs is necessary, use the menu item ***Device*** → ***Calibration…***.
The item is active only if a device is connected. Select the calibration target (inputs or outputs) in the opened dialog.

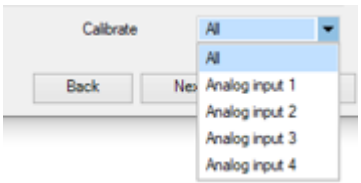


After the calibration target selection, the execution of the program in the device is stopped. The program starts again upon the successful completion of the calibration.

7.4.1 Input calibration

To calibrate inputs, connect a reference signal source to them. Start calibration, select the type of signal connected to the input and set the calibration parameters in the opened dialog.

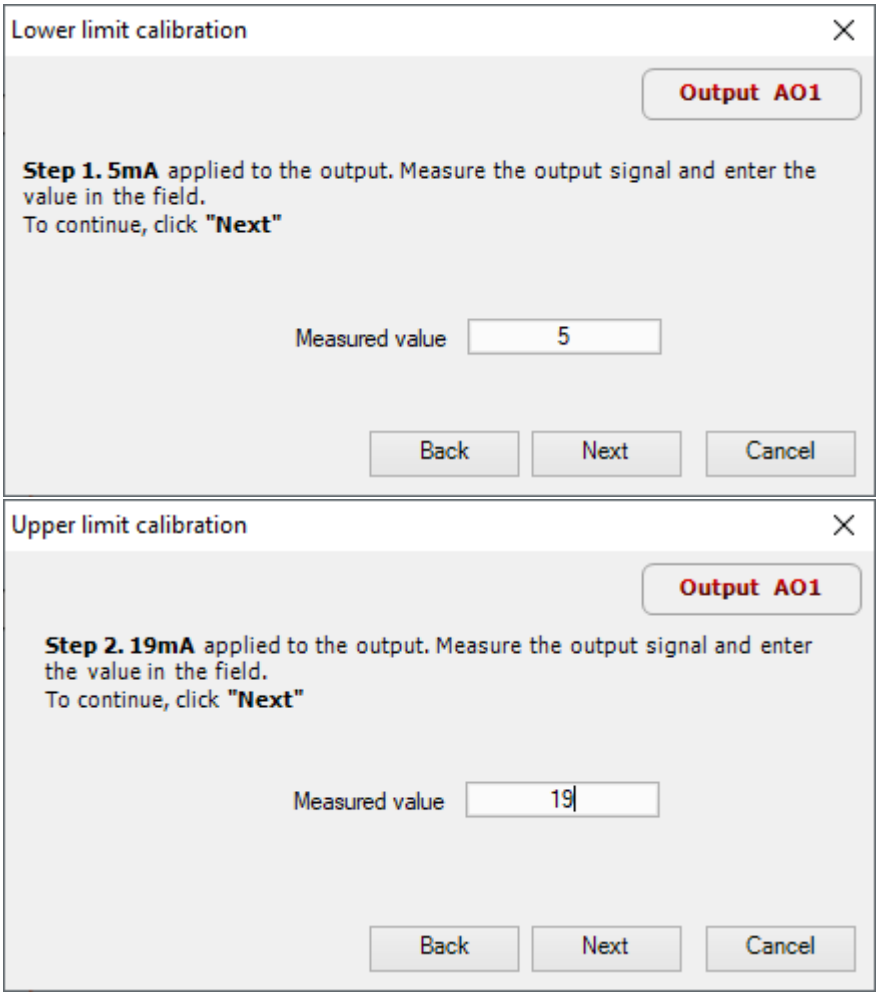




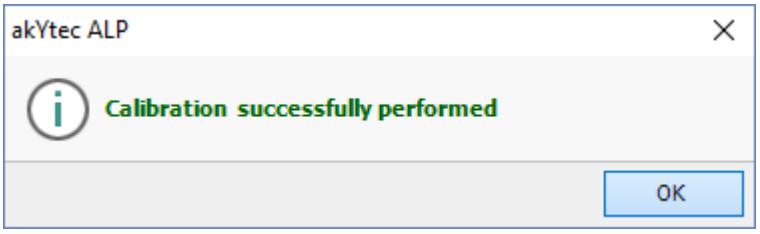
Use the item ***Reset settings*** to apply the default settings for calibration.
Use the list ***Select input*** to select the input to calibrate, click the button ***Next*** and follow the instructions.

7.4.2 Output calibration

Before calibrating an analog output, prepare the appropriate measuring device, than start calibration and follow the instructions. Measure the signal at the output indicated at the top right of the window and enter the value in the input field.
Proceed the same way with the other outputs if needed. The message about the calibration results will appear after the completion of the calibration.



Proceed the same way with the other outputs if needed. The message about the calibration results will appear after the completion of the calibration.



8 Change target device

The target device of a project can be changed using the menu item ***File > Change target device***. A list of devices to which you can transfer the project appears. Select the device and confirm with ***OK***. Check and repair broken references in the project, if any. The program can be checked using simulation. Save the modified project.
Consider the replacement rules:

1. The workspace size will be automatically adjusted to the changed number of I/O points.
2. User-configured layout of I/O points remains. New I/O points will be placed after existing I/O points of the original project.
3. The connections of I/O points whose data type has been changed will be removed.
4. If the number of I/O points becomes less than the one in the original project, the connections of the removed I/O points will also be removed.
5. If there were extension modules in the original project, they will be transferred to the new configuration with their connections.
6. Settings of analog I/O points will be transferred if there are analog I/O points on the new device.
7. Network interfaces will be transferred unchanged.
8. Display settings will be transferred unchanged.
9. Variables will be transferred unchanged.

9 Keyboard shortcuts

Keyboard shortcut	Action
Menu/File	
Ctrl + N	Create a new project
Ctrl + O	Open an existing project
Ctrl + Alt + S	Save an open project under a different name
Ctrl + S	Save an open project
Ctrl + P	Print
Ctrl + Shift + C	Open Component Manager
Menu/View	
Ctrl + Z	Undo last change
Ctrl + Y	Return (restore) a canceled action
Menu/Device	
Ctrl + F7	Transfer the application to the device
Ctrl + Shift + V	Open Variable Table
Ctrl + Shift + S	Open device configuration
Menu/Service	
Shift + F5	Go to simulation mode
F6	Start simulation
F7	Stop simulation
F8	Pause simulation
F10	Single cycle
Ctrl + F5	Go to debug mode
Menu/Plugins	
Ctrl + Shift + P	Open plugins manager
Menu/Help	
F1	Open Help
Insert panel	
Ctrl + Shift + F	Create an ST function
Ctrl + Shift + M	Create a macro
Ctrl + Shift + B	Create an ST function block
Ctrl + M	Create a macro from a selection
Usage location panel	
F5	Refresh usage locations
Keys for working with elements	
Ctrl + C	Copy an element
Ctrl + V	Paste from clipboard
Delete	Deleting a selected item

Keyboard shortcut	Action
Element resizing keys	
Ctrl + →	Increasing the width of a selected element
Ctrl + ←	Decreasing the width of a selected element
Ctrl + ↓	Increasing the height of a selected element
Ctrl + ↑	Decreasing the height of a selected element
Scaling the workspace	
Ctrl + Mouse wheel	When you rotate the mouse wheel away from you, the scale of the workspace increases. When you rotate the mouse wheel toward you, the scale of the workspace decreases
Ctrl + «+»	Increase scale
Ctrl + «–»	Decrease scale
Switch between tabs	
Tab + →	Switch between tabs
Tab + ←	

10 Program examples

Two examples with simple tasks explain the creation of a circuit program in the ALP programming software.

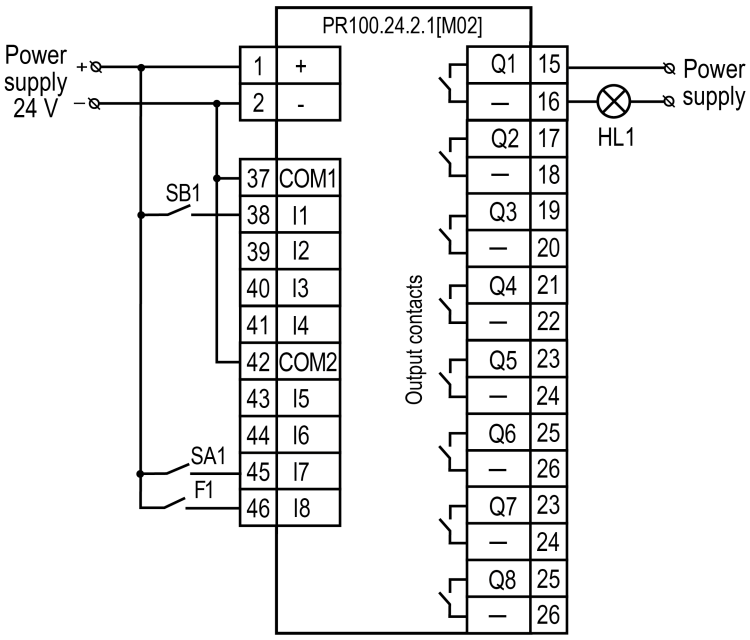
– *Light switch with automatic switch-off 10.1*
– *Mixer control 10.2*

10.1 Task 1: Light switch with automatic switch-off

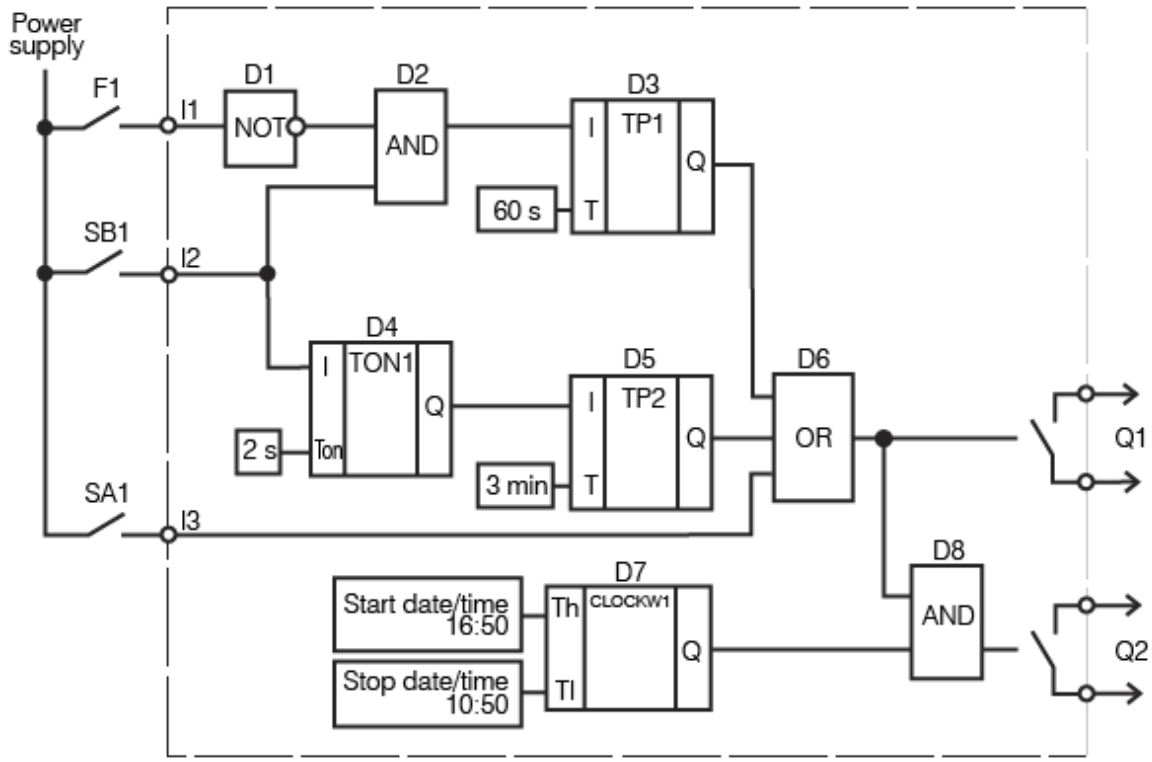
The task is to switch the light on for a certain time, e.g. for a house entry.
Task definition:

1. The light sensor F1 and the light button SB1 “TIME” are installed in front of the entrance door.
2. If the button SB1 is shortly pressed and the ambient light is insufficient, the light should be switched on for 1 minute – this time should be enough to find a key hole and to open the door.
3. If the button SB1 is pressed for 2 seconds, the light should be switched on for 3 minutes regardless of the ambient light – this mode can be useful for entrance cleaning.
4. Provide the possibility to control the light by commands from external devices or with the switch SA1 “CONST” regardless of the ambient light. This mode can be useful during the reception of guests or for further automation of the apartment as part of the "smart house" program.
5. Provide the possibility to switch on the light only at a certain time.

Device selection:
The control device must have minimum two digital inputs, one digital output and an integrated real-time clock to implement this task. These features can be provided by devices of PR100 series.
The task implementation with the device PR100.24.2.1(M02):



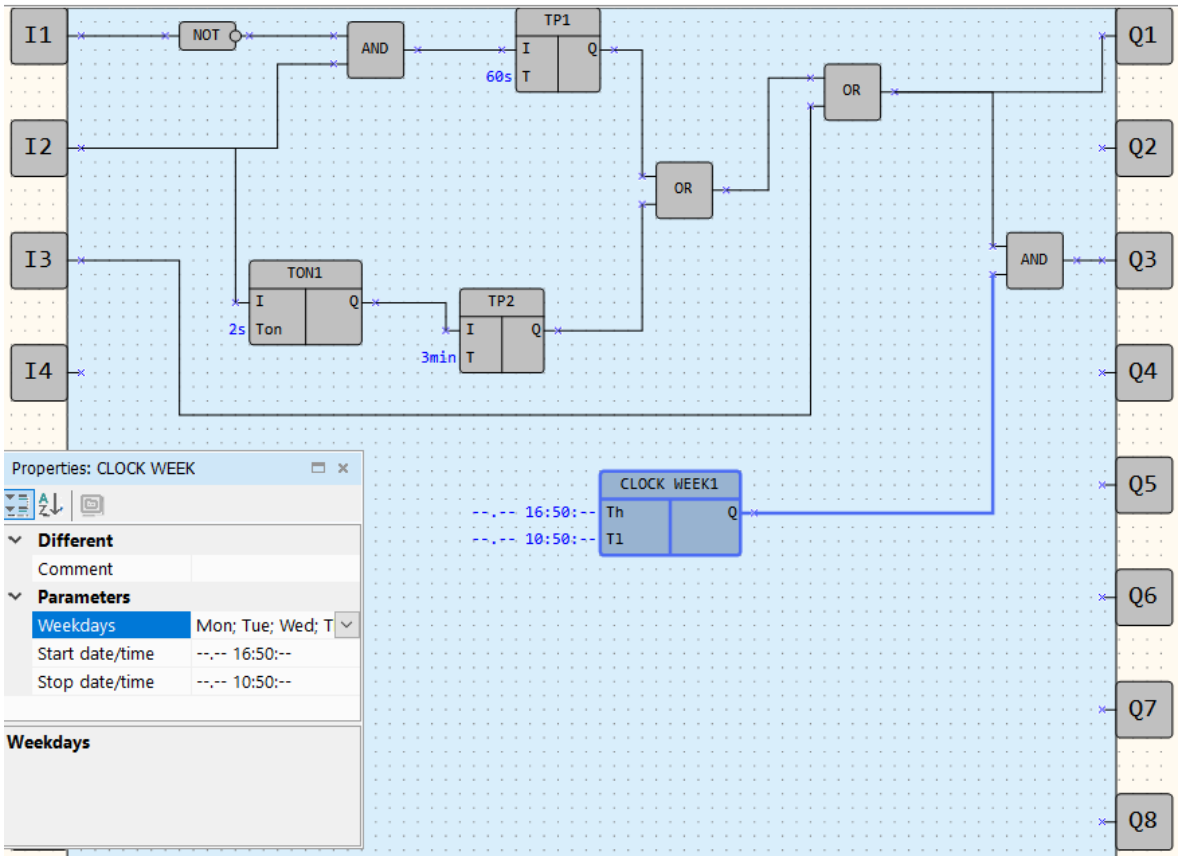
Circuit program
The circuit program can be implemented in the way shown in figure below.



Input I1 – connected to the light sensor F1
Input I2 – connected to the button SB1
Input I3 – connected to the switch SA1
Output Q1 – output to implement the task points 1-4
Output Q2 – output to implement the task point 5
Program description:

1. If the button SB1 is shortly pressed (< 2 s), the logical AND (D2) is enabled. If the ambient light is insufficient, the first input of D2 is also **True** and the timer TP “Pulse” (D3) forms a pulse with 1 minute duration. This pulse activates the output Q1 over the logical OR (D6) and the light is switched on for 1 minute.
2. If the button SB1 is pressed for > 2 s, the on-delay timer TON (D4) activates the timer TP “Pulse” (D5), a pulse with the duration of 3 minutes activates the output Q1 over logical OR (D6) and the light is switched on for 3 minutes.
3. If the ambient light is sufficient, the contact of the sensor F1 is closed, the logical AND (D2) is disabled and the timer TP “Pulse” (D3) is blocked.
4. If the switch SA1 “CONST” is closed, the output Q1 is activated over the logical OR (D6) and the light is switched on constantly.
5. If you want to use the light only on certain weekdays at certain times, you can use the output Q2. With the weekly timer CLOCKW (D7) you can set the start and the stop time and the weekdays for lighting.

The circuit program created in ALP is shown in figure below.

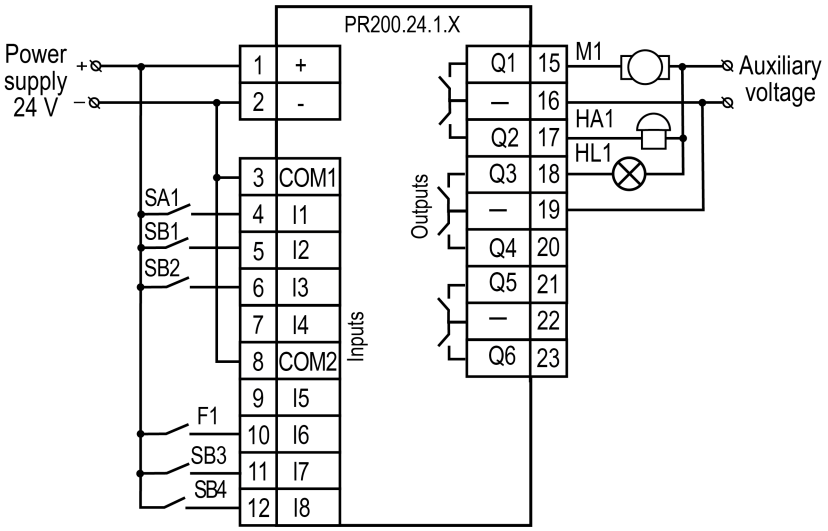


10.2 Task 2: Mixer control

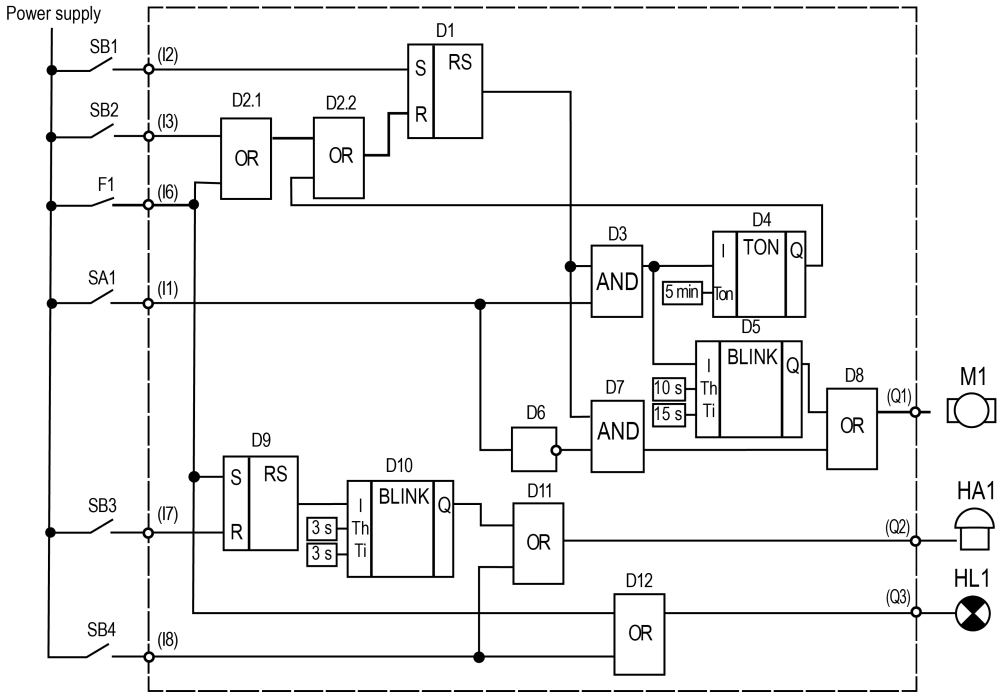
The task is to implement an industrial mixer with simple control functions.
Task definition:

1. Automatic and Manual operation modes are required. The switch SA1 “MODE” is installed to switch between the modes.
2. In Automatic mode the operating cycle can be started with the button SB1 “START” and stopped automatically with the end of the cycle or manually with the button SB2 “STOP”. The cycle duration is 5 minutes. During the cycle the motor of the mixer is on for 15 seconds and off for 10 seconds alternately. All settings can be changed in the program.
3. In Manual mode the motor can be started with the button SB1 “START” and stopped with the button SB2 “STOP”.
4. When the motor is overloaded (overload switch F1), it should be switched off automatically, an intermittent acoustic warning signal (HA1) with the 3-second interval should be produced and an operating error should be indicated by the signal lamp HL1 "Overload".
5. The acoustic signal can be switched off with the button SB3 “RESET”.
6. The button SB4 “CONTROL” is used for the functional test of the lamp HL1 and the acoustic signal HA1.

Device selection:
The control device must have minimum 6 digital inputs and 3 digital outputs to implement this task. These features can be provided by devices of PR200 series.
The task implementation with the device PR200.230.1.X:



Circuit program
The circuit program can be implemented in the way shown in figure below.



Input I1 – connected to the switch SA1 “MODE”
Input I2 – connected to the button SB1 “START”
Input I3 – connected to the button SB2 “STOP”
Input I6 – connected to the overload switch F1
Input I7 – connected to the button SB3 “RESET”
Input I8 – connected to the button SB4 “TEST”
Output Q1 – connected to the motor
Output Q2 – connected to the acoustic signal HA1
Output Q3 – connected to the signal lamp HL1
Program description:

1. Input I2 (SB1 “START”)

   If the button SB1 is pressed, the RS trigger D1 becomes **True** as long as there is no reset signal at the input R. Subsequent signal path depends on the state of the switch SA1 “MODE”:

   – If SA1 is open (Manual mode), the logical AND (D7) and the logical OR (D8) are enabled and the motor M1 (output Q1) is switched on.

- If SA1 is closed (Automatic mode), the logical AND (D7) is disabled and the start signal can only activate the pulse generator BLINK (D5) to start the operating cycle (15 s on / 10 s off) and the on-delay timer TON (D4) to stop it (in 5 min).

2. Input I3 (SB2 “STOP”)

   If the button SB2 is pressed or the switch F1 is activated, the RS trigger D1 is reset over the input R and the output Q1 is disabled.

3. Input I1 (SA1 “MODE”)
   - If the switch SA1 is open (Manual mode), the logical AND D3 is disabled and D7 is enabled, the timer D4 and the pulse generator D5 are disabled and the motor M1 can be only started with SB1 and stopped with SB2.
   - If the switch SA1 is closed (Automatic mode), the logical AND D3 is enabled and D7 is disabled, thus the motor M1 can be only started by the pulse generator D5 (15 s on / 10 s off cycle) and stopped by the timer D4 in 5 minutes.

4. Input I6 (overload switch F1)

   When the motor is overloaded, the F1 contact is closed, the RS trigger D1 is reset and the motor is stopped.

   Concurrently the signal lamp HL1 is switched on over the logical OR (D12) and the acoustic signal HA1 is activated over the RS trigger D9. The pulse generator D10 provides an intermittent acoustic signal with the cycle 3 s on / 3 s off.

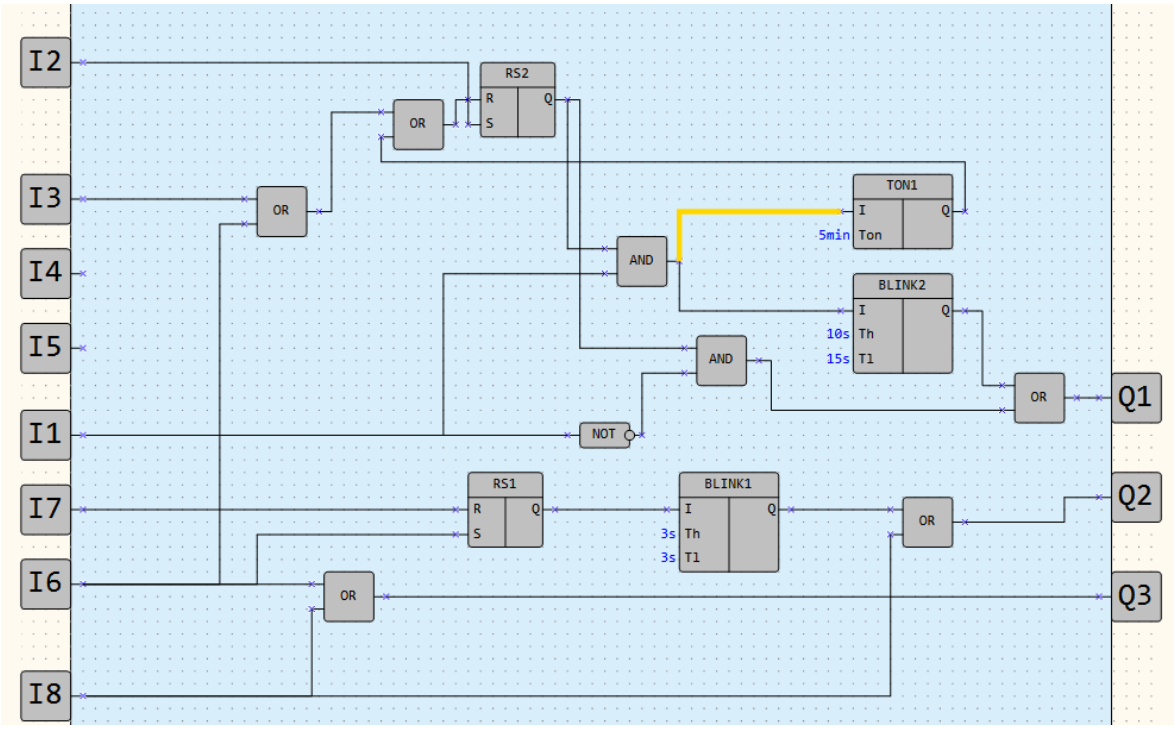
5. Input I7 (SB3 “RESET”)

   The button RESET is used to reset the acoustic signal HA1. If the button SB3 is pressed, the RS trigger D9 is reset and the pulse generator D10 for the acoustic signal HA1 is stopped.

6. Input I8 (SB4 “TEST”)

   The button TEST is used to test the acoustic signal HA1 and the signal lamp HL1. If the button SB4 is pressed, the logical ORs D11 and D12 are enabled, the outputs Q2 and Q3 activated, the acoustic signal and the lamp are switched on.

The circuit program is shown in the figure below.



NOTE

1. *The remaining two unused inputs and one output can be used for implementation of additional functions. For example, to switch between different time settings for automatic motor operation or to switch other operating parameters of the mixer.*
2. *The technological cycle of operation can be completely automated by implementation of an incremental counter (CT) to switch off the RS trigger D1.*

11 ST language

ST (Structured Text) is a high-level text language. It is one of the five languages supported by the IEC 61131-3 standard.
ALP allows you to create functions and function blocks in the ST language.

11.1 Syntax

Keywords can be entered in upper and lower case characters. Spaces and tabs do not affect the syntax and can be used everywhere.
The names of variables, functions and function blocks follow the following rules:

– The name must not contain spaces or special characters (for example, !, @, etc.). The exception is the underscore character (_)
– The name must start with a letter
– The variable name can only contain letters of the Latin alphabet
– The name must not contain multiple underscore (_) characters in a row (i.e. the name **i__Test** is not valid, but the name **i_Te_st** is valid)
– Object names are case insensitive (**ITest** and **iTEST** will be interpreted as the same name)
– There are no restrictions on name length
– The name must not match one of the reserved keywords (eg VAR, INT, etc.)
– It is recommended to use Hungarian notation and lowerCamelCase style for variable names

11.1.1 Using functions in other ST program elements

A function can be called within another function or function block. To do this, you need to use the following format (informal call):
Function name (comma separated list of function inputs)
Example

```
FUNCTION rFun1: REAL;

    VAR_INPUT
        rIn1 : REAL;
        rIn2 : REAL;
    END_VAR

    rFun1 := rIn1 + rIn2;

END_FUNCTION
```

There is a function **rFun1**:
Its call in the **rFun2** function will look like this:

```
FUNCTION rFun2: REAL;

    VAR_INPUT
        rIn1_0 : REAL;
        rInf1_1 : REAL;
        rInf1_2 : REAL;
    END_VAR
```

```
    rFun2 := rIn1_0 * rFun1(rInf1_1, rInf1_2);

END_FUNCTION
```

The **rFun2** function will return a number equal to the product of **rIn1_0** and the sum of **rInf1_1** and **rInf1_2**, which are specified at the input of this function.

11.1.2 Using one function block in another

Instances of one function block can be created in another function block, namely in the local variable declaration area in the format:

(short designation):(function block name)

Once a function block has been instantiated, you can begin working with its data. The inputs of a block instance are externally writable. Outputs are for reading.

You can call an instance in different ways. As an example, consider the call to a counter for forward counting, which is created as a template when creating a function block:

1. Through a formal call:

```
FUNCTION_BLOCK fb2

    VAR_INPUT
        xIn : BOOL;
    END_VAR

    VAR_OUTPUT
        xAlarmMax : BOOL;
        udiQ : UDINT;
    END_VAR

    VAR
        fb1 : functionblock1; //declaration of a function block instance
    END_VAR

    fb1 (U := xIn, Q => udiQ); //calling a function block instance
    IF fb1.Q > 10 THEN
        xAlarmMax := TRUE;
    END_IF

END_FUNCTION_BLOCK
```

NOTE
Pay attention to the specific operator for copying a value from the block's output variables (“=>”).

2. By accessing the inputs/outputs of a function block (variable declarations are similar):

```
    fb1.U := xIn; //write data to FB input
    fb1 (); //call a FB instance
    udiQ := fb1.Q; //read the output of the FB instance
    IF fb1.Q > 10 THEN
        xAlarmMax := TRUE;
    END_IF
```

The **fb2** output will indicate that 10 pulses have been exceeded and the accumulated counter value.

NOTICE
You cannot declare an instance of a function block in the body of a function.

NOTICE
In an ST functional block, the maximum nesting of blocks is no more than 8.

NOTICE
ST function blocks do not support RETAIN type variables.

NOTICE
ST function blocks reserve space in ROM memory after they are added to the project library, regardless of whether they are used in the project or not.

11.1.3 Comments in ST editor

The ST editor in ALP supports the commenting feature. Two types of comments are available:

1. Single-line. Its marker is a double slash – //
2. Multiline. Its marker is:
   – (* – beginning of comment
   – *) – end of comment

11.1.4 Copying ST elements between projects

To copy STcomponents between projects:

1. Select the ST elements that need to be copied on the original project diagram.
2. Copy the selected elements using the keyboard shortcut **Ctrl+ C** or using the context menu.
3. Open the project to paste the copied elements.
4. Paste elements into the second project diagram using the keyboard shortcut **Ctrl + V** or using the context menu

As a result, templates of all copied elements will be added to the corresponding section of the project component library.

NOTE

1. If the component library does not contain the group that was specified for the copied component, it will be created.
2. If the copied component group is not specified, the template will be added to the **Other** folder.

When copying, all connections between all components that were included in the copy operation are preserved.

NOTICE
If any of the copied elements contains an error, the pasting of the elements into the project diagram will be canceled.
An error message will be displayed indicating the name of the incorrect element.
Other components will appear in the component library in the appropriate sections.

11.2 Documentation in the ST editor

ST editor supports documentation function.
The documentation marker is the triple slash “///”.
Documentation is added above objects: function/function block declaration, input/output variable declaration.
Tags for documentation are listed in the table below.

Tag	Description
<Description>…</Description>	Description of the program element (function, function block, input and output (function block only) variable)
<Author>…</Author>	Name of the creator of the function or function block
<GroupName>…</GroupName>	Group name for grouping a function or function block in a content library
<OutputDescription>…</OutputDescriotion>	Function output description

Documenting the function

```
///<Description>Resistance temperature detector (Pt1000)</Description>
///<OuptutDescription>Temperature</OutputDescription>
///<Author>akYtec</Author>
///<GroupName>Temperature sensor</GroupName>

FUNCTION f_PT1000: REAL; // function for PT1000 RTD sensor

    VAR_INPUT

        ///<Description>Resistance</Description>
        R : REAL;

    END_VAR

    VAR_OUTPUT

        ///<OutputDescription>Temperature</OutputDescription>
        Q : REAL;

    END_VAR
```

Documenting the function block

```
///<Description>Counter for direct counting</Description>
///<Author>akYtec</Author>
///<GroupName>Timers and counters</GroupName>

FUNCTION_BLOCK fb_Counter

    VAR_INPUT

        ///<Description>Pulse detector</Description>
        U : BOOL; //bool input variable

        ///<Description>Counter reset flag</Description>
```

```
    Res : BOOL; //bool input variable

    ///<Description>Preset counter value</Description>
    N : UDINT; //bool input value

END_VAR

VAR_OUTPUT

    ///<Description>Counter value</Description>
    Q : UDINT; //udint output value

END_VAR
```

11.2.1 Reserved keywords

NOTE
In the table, words available for use are highlighted in **bold**.

ABS	**END_REPEAT**	READ_ONLY	**THEN**
ACTION	END_RESOURCE	READ_WRITE	**TIME**
AND	END_STEP	**REAL**	TIME_OF_DAY
ARRAY	END_STRUCT	**REAL_TO_BOOL**	**TIME_TO_UDINT**
AT	END_TRANSITION	**REAL_TO_UDINT**	**TO**
BEGIN	END_TYPE	**REPEAT**	TOD
BOOL	**END_VAR**	RESOURCE	TRANSITION
BOOL_TO_REAL	**END_WHILE**	RETAIN	**TRUE**
BOOL_TO_UDINT	ENO	**RETURN**	TYPE
BY	**EXIT**	SEL	**UDINT**
CASE	F_EDGE	**SHL**	**UDINT_TO_BOOL**
CD32	**FALSE**	**SHR**	**UDINT_TO_DT**
CONFIGURATION	**FOR**	SINT	**UDINT_TO_REAL**
CONTINUE	**FROM**	STEP	**UDINT_TO_TIME**
DATE	**FUNCTION**	STRING	UINT
DATE_AND_TIME	**FUNCTION_BLOCK**	STRUCT	ULINT
DC32	**GET_DATE_TIME**	**SYS.BLINK**	UNTIL
DINT	**GET_TIME**	**SYS.CLOCK**	USINT
DO	**IF**	**SYS.CLOCKWEEK**	**VAR**
DT	INITIAL_STEP	**SYS.COMPARE_ DATE_TIME**	VAR_ACCESS
DT_TO_UDINT	INT	**SYS.CT**	VAR_CONFIG
DWORD	LINT	**SYS.CTN**	VAR_EXTERNAL

ELSE	LREAL	**SYS.CTU**	VAR_GLOBAL
ELSIF	LWORD	**SYS.DTRIG**	VAR_IN_OUT
EN	**MOD**	**SYS.FTRIG**	**VAR_INPUT**
END	NON_RETAIN	**SYS.IS_LEAP_YEAR**	**VAR_OUTPUT**
END_ACTION	**NOT**	**SYS.RS**	VAR_TEMP
END_CASE	**OF**	**SYS.RTRIG**	**WHILE**
END_ CONFIGURATION	ON	**SYS.SR**	WITH
END_FOR	**OR**	**SYS.TOF**	WORD
END_FUNCTION	**POW**	**SYS.TON**	WSTRING
END_FUNCTION_ BLOCK	PROGRAM	**SYS.TP**	**XOR**
END_IF	R_EDGE	TASK	

11.3 Data types

Data types, supported in akYtec ALP:

Data type	Description	Valid range	Size
BOOL	Boolean	FALSE, TRUE	4 bytes
UDINT	Unsigned double integer	0…4294967295	4 bytes
REAL	Floating-point	$-1{,}2\times10^{-38}$…$3{,}4\times10^{38}$	4 bytes
TIME	Time interval	T#0…4294967295ms T#0..4294967s T#0..71582m T#0..1193h T#0..49d T#0..49d17h02m47s295ms	4 bytes
DT	Time of day and date	DT#2000-01-01-00:00:00..2136-02-07-6:28:15	4 bytes

The data type of a variable determines the type of information, the range of representations, and the set of allowed operations.
The variable can be used only after its declaration. To assign the value of one variable to another variable is possible only if they are of the same type. Otherwise, type converter should be used.

NOTE
Converting a larger type to a smaller one can result in loss of information.

11.4 Language stuctures

ST language structures include:
- *arithmetic operations 11.4.1.1*
- *bit operations 11.4.1.2*
- *data type conversion operations 11.4.1.3*
- *logical operations 11.4.1.4*
- *relational operations 11.4.1.5*
- *assignment operation 11.4.2*
- ***IF – ELSIF – ELSE** statement 11.4.3*
- ***CASE** statement 11.4.4*

– ***RETURN** statement*
– ***FOR** statement 11.4.6*;
– ***WHILE** statement 11.4.7*;
– ***REPEAT – UNTIL** statement 11.4.8*.

NOTE
When writing expressions, it is permissible to use variables (input, output and local) and constants.

11.4.1 Operations

11.4.1.1 Arithmetic operations

Operation	Operator	Data types	Example
addition	+	IN, OUT: UDINT/ REAL	OUT := IN1 + IN2 + …
multiplication	*		OUT := IN1 * IN2 * …
subtraction	-		OUT := IN1 - IN2
division	/		OUT := IN1 / IN2
modulo	MOD		OUT := IN1 MOD IN2
absolute value	ABS (IN)	IN, OUT: REAL	OUT := ABS (IN1)
exponentiation	POW (IN, N) IN – base N – exponent	IN, N, OUT: REAL	OUT := POW (IN1, N)

The result of the arithmetic operation is the mathematical result of the expression.
The priority of an operation determines the order of its execution in the expression. Parentheses are allowed to define the calculation order in arithmetic expressions.

11.4.1.2 Bit operations

Operation	Operator	Data types	Example
Bitwise shift left	SHL (IN, N)	IN, OUT: UDINT N: 1..32	OUT := SHL (IN1, N)
Bitwise shift right	SHR (IN, N)		OUT := SHR (IN1, N)
Decoder. Converts binary code to positional code	DC32 (IN)	IN, OUT: UDINT	OUT := DC32 (IN1)
Encoder Converts positional code to binary code	CD32 (IN)	IN, OUT: UDINT	OUT := CD32 (IN1)

11.4.1.3 Data type conversion operations

Operation	Operator	Data types	Example
UDINT в REAL	UDINT_TO_REAL (IN)	IN: UDINT OUT: REAL	OUT := UDINT_TO_ REAL (IN)
UDINT в BOOL	UDINT_TO_BOOL (IN)	IN: UDINT OUT: BOOL	OUT := UDINT_TO_ BOOL (IN)
UDINT в TIME	UDINT_TO_TIME (IN)	IN: UDINT OUT: TIME	OUT := UDINT_TO_ TIME (IN)

Operation	Operator	Data types	Example
UDIMT в DT	UDINT_TO_DT (IN)	IN: UDINT OUT: DT	OUT := UDINT_TO_DT (IN)
REAL в UDINT	REAL_TO_UDINT (IN)	IN: REAL OUT: UDINT	OUT := REAL_TO_UDINT (IN)
REAL в BOOL	REAL_TO_BOOL (IN)	IN: REAL OUT: BOOL	OUT := REAL_TO_BOOL (IN)
BOOL в REAL	BOOL_TO_REAL (IN)	IN: BOOL OUT: REAL	OUT := BOOL_TO_REAL (IN)
BOOL в UDINT	BOOL_TO_UDINT (IN)	IN: BOOL OUT: UDINT	OUT := BOOL_TO_UDINT (IN)
TIME в UDINT	TIME_TO_UDINT (IN)	IN: TIME OUT: UDINT	OUT := TIME_TO_UDINT (IN)
DT в UDINT	DT_TO_UDINT (IN)	IN: DT OUT: UDINT	OUT := DT_TO_UDINT (IN)

11.4.1.4 Logical operations

Operation	Operator	Data type	Example
Logical negation	NOT	IN, OUT: BOOL	OUT := NOT IN1
Boolean multiplication	AND &		OUT := IN1 AND IN2 OUT := IN1 & IN2
Boolean addition	OR		OUT := IN1 OR IN2
logical (bitwise) "exclusive OR"	XOR		OUT := IN1 XOR IN2

11.4.1.5 Relational operations

Operation	Operator	Data types	Example
greater than	>	IN: UDINT/REAL OUT: BOOL	OUT := IN1 > IN2
greater than or equal to	>=		OUT := IN1 >= IN2
equal to	=		OUT := IN1 = IN2
less than or equal to	<=		OUT := IN1 <= IN2
less than	<		OUT := IN1 < IN2
not equal to	<>		OUT := IN1 <> IN2

11.4.1.6 Operation priorities

NOTE
The operations in the table are ordered **from highest to lowest priority**. The higher the priority of an operation, the sooner it is executed.

Operation	Operator
Brackets	(expression)
Calling a function and function block	Example: fb1(); function1 := … ;
Bit operations	

Operation	Operator
Unary minus	–
Logical negation	NOT
Exponentiation	POW
Multiplication	*
Division	/
Modulo	MOD
Addition	+
Subtraction	–
Relational operations	>.< <=, >=
Equal to	=
Not equal to	<>
Conjunction Logical multiplication "AND"	& AND
Exclusive OR	XOR
Disjunction Logical addition "OR"	OR

11.4.2 Assignment operation

The paired symbol "**:=**" is used to indicate assignment. The right and left sides of the expression must contain operands of the same type (automatic type casting is not provided). On the left side of the expression (receiving side) only a variable can be used. The right side can contain an expression or a constant.

11.4.3 IF statement

The **IF** operator allows you to test one or more conditions, and, if at least one of the conditions is true, execute the specified expression conditions. After executing the expressions, the operator exits the statement – that is, the remaining conditions are no longer checked.
Let's consider the operator's work using the example of signaling that the temperature value exceeds the permissible limits:

```
FUNCTION_BLOCK fb //function block name

    VAR_INPUT //declaration of inpur variables
        rTemp : REAL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        xHigh : BOOL;
        xLow : BOOL;
    END_VAR

    VAR //declaration of local variables
        rHighTemp : REAL := 20;
        rLowTemp : REAL := 10;
```

```
    END_VAR

    //coding area

    IF rTemp > rHighTemp THEN
        xHigh := TRUE;
    ELSIF rTemp < rLowTemp THEN
        xLow := TRUE;
    ELSE
        xHigh := FALSE;
        xLow := FALSE;
    END_IF

END_FUNCTION_BLOCK
```

If the condition in the **IF** statement is true (the value of the variable **rTemp** is greater than **rHighTemp**), then the variable **xHigh** will be assigned the value **TRUE** and the statement will exit the statement (the next condition will not be checked). If the condition is not met, then the next condition placed in the nested **ELSIF** statement will be checked. If the condition in **ELSIF** is satisfied (the value of the variable **rTemp** is less than **rLowTemp**), then the variable **xLow** will be assigned the value **TRUE** and the statement will exit the statement (the next condition will not be checked). If none of the conditions in **IF** and **ELSIF** are met (that is, the temperature value is within acceptable limits), then the expressions placed in the nested **ELSE statement will be executed – the assignment of value FALSE to the xHigh and xLow variables.**
The use of nested **ELSIF** and **ELSE** statements is optional. An arbitrary number of **ELSIF** statements can be placed inside an **IF** statement.
The construction allows nesting, that is, inside one **IF** there can be another one, etc. Also inside the **IF** operator, loops and the ***CASE 11.4.4*** operator can be used.

11.4.4 CASE statement

The **CASE** operator allows you to compare the value of a given integer variable (selector) with a set of constants or integer values (labels), and if there is a match, execute the expressions specified for this label. After executing the expressions, the operator exits the statement.
Example:

```
FUNCTION_BLOCK fb1 //function block name

    VAR_INPUT //declaratio of input variables
        udiSel : UDINT;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        xOut1 : BOOL;
        xOut2 : BOOL;
        xOut3 : BOOL;
        xOut4 : BOOL
    END_VAR

    //coding area

    xOut1 := FALSE;
```

```
    xOut2 := FALSE;
    xOut3 := FALSE;
    xOut4 := FALSE;

    CASE udiSel OF
        0:
            xOut1 := TRUE;
        1..3:
            xOut2 := TRUE;
        4, 6:
            xOut3 := TRUE;
    ELSE
        xOut4 := TRUE;
    END_CASE


END_FUNCTION_BLOCK
```

If the **udiSel** value is:

– Equal to 0, then **xOut1** will take the value **TRUE**;
– If it falls into the range 1..3, then **xOut2** will take the value **TRUE**;
– Equal to 4 or 6, then **xOut3** will take the value **TRUE**;
– Does not fall into any of the specified values, then **xOut4** will take the value **TRUE**;

As can be seen from the example, a label can include several values, listed, separated by commas “4, 6”, or the range “1..3”. In this case, the values of one of the labels should not coincide with the values of the others. Also, when specifying a range of values, the beginning of the range must be less than its end.
The nested **ELSE** statement is optional; the expressions placed in it are executed if the selector value does not match any of the labels.
The actions provided to handle each of the **CASE** statement cases can use loops, **IF** statements, and **CASE** statements.

11.4.5 RETURN statement

The **RETURN** statement allows you to exit a program object.
Usage example:

```
IF xDone THEN
    RETURN;
END_IF;


udiCounter := udiCounter + 1;
```

If the variable **xDone** takes the value **TRUE**, then the expression “**udiCounter** := **udiCounter** + 1” will not be executed will be (like everything that will be located below in the body of the program).

11.4.6 FOR statement

The **FOR** operator is used to organize a loop with a predetermined number of iterations. It is usually used for operations on arrays of data.
The **IF** and **CASE** operators, as well as other loop operators, can be used inside a loop.
As an example, consider the implementation of bubble sort from smallest to largest:

```
FUNCTION_BLOCK MaxI_MinI
//Maximum and minimum numbers using bubble sort from smallest to largest
```

```
    VAR_INPUT //declaration of input variables
        udiX1, udiX2, udiX3, udiX4, udiX5 : UDINT;
//Adding the required number of input variables and defining the data type
    END_VAR

    VAR_OUTPUT //declaration of output variables
        udiMaxI, udiMinI : UDINT;
    END_VAR

    VAR //declaration of local variables
        udiI, udiJ, udiN, udiK : UDINT;
        audiX : ARRAY [1..5] OF UDINT;
//Specifies the range of the array and the data type of the array
    END_VAR

    //coding area
    //array declaration
    audiX[1] := udiX1;
    audiX[2] := udiX2;
    audiX[3] := udiX3;
    audiX[4] := udiX4;
    audiX[5] := udiX5;
    udiN := 5; // the number of numbers to sort is specified (array size)

    FOR udiI := 1 TO udiN-1 DO
        FOR udiJ := 1 TO udiN-udiI DO
            IF audiX[udiJ] > audiX[udiJ+1] THEN
                udiK := audiX[udiJ];
                audiX[udiJ] := audiX[udiJ+1];
                audiX[udiJ+1] := udiK;
            END_IF;
        END_FOR;
    END_FOR;

    udiMaxI := audiX[udiN]; //the last (maximum) number of the array is displayed
    udiMinI := audiX[1]; //the first (minimum) number of the array is displayed


END_FUNCTION_BLOCK
```

The **udiI** variable is called the loop iterator (counter). This variable must be of a signed integer type (**UDINT**). After each execution of the loop body, the iterator value changes - by default to **+1**. The user can set the iterator “step” using the nested **BY** operator. After this, the transition immediately occurs to the next iteration - that is, the entire cycle is executed “from beginning to end”. The structure will look like this:

```
// udiI will take the values 1, 4, 7, 10, etc.
FOR udiI := 1 TO udiN-1 BY 3 DO
```

```
    .. // loop code
END_FOR
```

The loop iterator (counter), its start and end values, and its increment are integer values. They are calculated before entering the loop, and changing the values of the variables included in any of these expressions will not change the number of iterations.
You can exit a loop early using the **EXIT** operator. Example structure:

```
FOR udiI := 1 TO udiN-1 DO
    IF audiX[2] > 100 THEN
        EXIT;
    ELSE
        .. // loop code
    END_IF
END_FOR
```

You can skip a loop step using the **CONTINUE** operator.

```
FOR udiI := 1 TO udiN-1 DO
    IF udiI = 3 THEN
        CONTINUE;
    ELSE
        .. // loop code
        END_IF
END_FOR
```

11.4.7 WHILE statement

The **WHILE** operator is used to create a loop with an unknown number of iterations. The loop will terminate if the condition being tested returns **FALSE**. In this case, the condition is checked BEFORE the expression is executed (a loop with a precondition). Thus, if the condition immediately returns **FALSE**, then the loop will not be executed even once.
The **IF** and **CASE** operators, as well as other loop operators, can be used inside a loop.
Example

```
WHILE rVar < 100 DO
    rVar := rVar + 1;
END_WHILE
```

The result of executing this loop (with the initial value of the variable **rVar** := 10) will be the number 100.
You can exit a loop early using the EXIT statement.
You can skip a loop step using the **CONTINUE** operator.

11.4.8 REPEAT UNTIL statement

The **REPEAT** statement is used to create a loop with an unknown number of iterations. The loop will terminate if the condition being tested returns **TRUE**. In this case, the condition is checked AFTER the expression is executed (a loop with a postcondition). Thus, if the condition immediately returns **TRUE**, then the loop will be executed once.
The **IF** and **CASE** operators, as well as other loop operators, can be used inside a loop.
Example:

```
REPEAT
    IF rVar > 100 THEN
        EXIT;
    END_IF;
    rVar := rVar + 1;
```

```
UNTIL rVar > 180
END_REPEAT;
```

The result of executing this loop (with the initial value of the variable **rVar** := 10) will be the number 101.
To exit a loop early, you can use the **EXIT** operator.
You can skip a loop step using the **CONTINUE** operator.

11.5 System functions

GET_TIME function

The **GET_TIME** function returns a **TIME** type value (4 bytes) containing the time elapsed since the device was last turned on, in milliseconds.
Sample

```
VAR
    Time_1 : TIME := T#0ms;
    Time_2 : TIME := T#0ms;
    Q : BOOL := FALSE;
END_VAR

IF Time_1 = T#0ms THEN
    Time_1 := GET_TIME();
END_IF

Time_2 := GET_TIME();

IF (Time_2 - Time_1) >= T#1000ms THEN
    Q := NOT Q;
    Time_1 := T#0ms;
    Time_2 := T#0ms;
END_IF
```

GET_DATE_TIME function

The **GET_DATE_TIME** function returns a **DT** type value (4 bytes) containing real time clock data, in seconds since 00:00:00 01/01/2000, taking into account the time zone set in the device.
Sample

```
VAR
    Ton_UDINT : UDINT;
    Ton_DT : DT;
END_VAR

Ton_DT := GET_DATE_TIME();

Ton_UDINT := DT_TO_UDINT(Ton_DT);
```

SYS.COMPARE_DATE_TIME function

The **SYS.COMPARE_DATE_TIME** function compares two UDINT values given as input using a given date/time mask and returns a UDINT value that evaluates to:

– 1 – value 1 is greater than value 2;
– 2 – value 1 is less than value 2;
– 0 – value 1 is equal to value 2.

The comparison is made by the number of seconds starting from 00:00:00 01.01.2000.

Sample

```
FUNCTION udiCompare: UDINT;

    VAR_INPUT
        udiValue1: UDINT; //value 1
    END_VAR

    VAR
        udiValue2 : UDINT; //value 2
        udiMask : UDINT := 63; //date/time mask
    END_VAR

    udiValue2 := DT_TO_UDINT (GET_DATE_TIME ());
    udiCompare := SYS.COMPARE_DATE_TIME (udiValue1, udiValue2, udiMask);

END_FUNCTION
```

Mask = 63 (0b111111) – all bits are used:
0 bit - if 1, then seconds are used;
1 bit - if 1, then minutes are used;
2 bits - if 1, then the clock is used;
3 bits - if 1, then days are used;
4 bits - if 1, then months are used;
5 bits - if 1, then years are used.

SYS.IS_LEAP_YEAR function

The **SYS.IS_LEAP_YEAR** function returns a BOOL value containing data on whether the UDINT number supplied to the function's input corresponds to a leap year (1 - leap year, 0 - not).

Sample

```
FUNCTION xLeapYear: BOOL; //function name and output data type

    VAR_INPUT //declaration of input variables
        udiYear : UDINT; //year being checked
    END_VAR

    xLeapYear := SYS.IS_LEAP_YEAR (udiYear);

END_FUNCTION
```

11.6 System Function Blocks

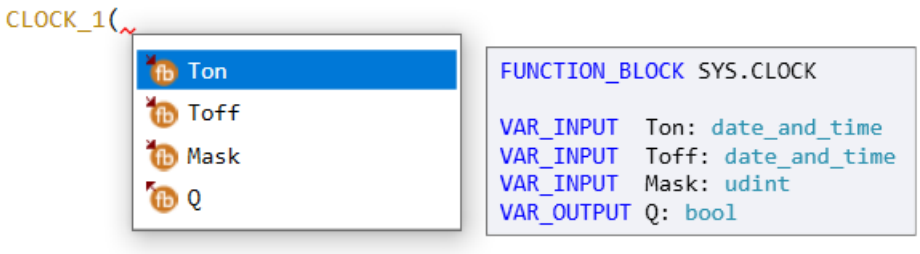
The ST editor supports the following system function blocks:

– *Triggers 11.6.1*;
– *Timers 11.6.2*;
– *Generetors 11.6.3*;

– *Counters 11.6.4*.

The use of system function blocks in other program elements is similar *using custom function blocks 11.1.2*.

For each system function block, the editor provides a hint on how to use it.



11.6.1 Triggers

– *RS trigger reset dominant (SYS.RS) 11.6.1.1*;
– *SR trigger set dominant (SYS.SR) 11.6.1.2*;
– *Rising edge (SYS.RTRIG) 11.6.1.3*;
– *Falling edge (SYS.FTRIG) 11.6.1.4*;
– *D-trigger (SYS.DTRIG) 11.6.1.5*.

11.6.1.1 RS trigger reset dominant (SYS.RS)

The RS trigger reset dominant (SYS.RS) is used to switch with state fixation during the receipt of short pulses at the corresponding input. The output Q will appear HIGH level signal at the edge of the signal at the input S.

Designation	Data type	Description
Inputs		
S	BOOL	SET input
R	BOOL	RESET input
Outputs		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK RS_trigger //Function block name

    VAR_INPUT //declaration of input variables
        R_in : BOOL;
        S_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        RS_1: SYS.RS;
    END_VAR

    //code area

    RS_1(R := R_in, S := S_in, Q => Q_out);

END_FUNCTION_BLOCK
```

In case of simultaneous receipt of signals at both inputs, the signal of input R takes priority.

11.6.1.2 SR trigger set dominant (SYS.SR)

The SR trigger set dominant (SYS.SR) is used to switch with state fixation during the receipt of short pulses at the corresponding input. The output Q will appear HIGH level at the edge of the signal at the input S.

Designation	Data type	Description
Inputs		
S	BOOL	SET input
R	BOOL	RESET input
Outputs		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK SR_trigger //Function block name

    VAR_INPUT //declaration of input variables
        S_in : BOOL;
        R_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        SR_1: SYS.SR;
    END_VAR

    //code area

    SR_1(S := S_in, R := R_in, Q => Q_out);

END_FUNCTION_BLOCK
```

In case of simultaneous receipt of signals at both inputs, the signal of input S takes priority.

11.6.1.3 Rising edge (SYS.RTRIG)

The rising edge (SYS.RTRIG) is used when it is necessary to have a reaction to a change in the state of a digital input signal. A single pulse is generated at the output Q on the rising edge of the input I.

Designation	Data type	Description
Input		
I	BOOL	Trigger input
Output		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK R_trigger //Function block name

    VAR_INPUT //declaration of input variables
        RT_in : BOOL;
    END_VAR
```

```
    VAR_OUTPUT //declaration of output variables
        RT_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        RTrig_1: SYS.RTRIG;
    END_VAR

    //code area

    RTrig_1(I := RT_in, Q => RT_out);

END_FUNCTION_BLOCK
```

11.6.1.4 Falling edge (SYS.FTRIG)

The falling edge (SYS.FTRIG) is used when it is necessary to have a reaction to a change in the state of a digital input signal. A single pulse is generated at the output Q on the leading edge of the input I.

Designation	Data type	Description
Input		
I	BOOL	Trigger input
Output		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK F_trigger //Function block name

    VAR_INPUT //declaration of input variables

        FT_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        FT_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        FTrig_1: SYS.FTRIG;
    END_VAR

    //code area

    FTrig_1(I := FT_in, Q => FT_out);
```

```
END_FUNCTION_BLOCK
```

11.6.1.5 D-trigger (SYS.DTRIG)

D-trigger (SYS.DTRIG) is used to generate a pulse to turn on the output for the time interval of the pulse at the D input, the output interval will be synchronized with the clock frequency at the C input. At the Q trigger output, a HIGH level signal will appear on the front of the clock pulses at the C input if there is a HIGH level signal at the D input. The return of the Q output to the LOW level signal will occur on the front of the clock pulses at the C input if there is a HIGH level signal at the D input.
Input S forces output Q to a HIGH level state.
Input R is the priority input and sets output Q to LOW level.

Designation	Data type	Description
Inputs		
S	BOOL	SET input
D	BOOL	Trigger input
C	BOOl	Clock frequency
R	BOOL	RESET input
Oupputs		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK D_trigger //Function block name

    VAR_INPUT //declaration of input variables
        S_in : BOOL;
        D_in : BOOL;
        C_in : BOOL;
        R_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        DTrig_1: SYS.DTRIG;
    END_VAR

    //code area

    DTrig_1(S := S_in, D := D_in, C := C_in, R := R_in, Q => Q_out);

END_FUNCTION_BLOCK
```

11.6.2 Timers

– *Pulse (SYS.TP) 11.6.2.1*;
– *ON-delay timer (SYS.TON) 11.6.2.2*;

– *OFF-delay timer (SYS:TOF) 11.6.2.3*;
– *Timer (SYS.CLOCK) 11.6.2.4*;
– *Weekly timer (SYS:CLOCKWEEK) 11.6.2.5*.

11.6.2.1 Pulse (SYS.TP)

The pulse (SYS.TP) is used to generate a pulse to turn on the output for a specified time interval. A HIGH level signal appears at the output Q of the block at the edge of the input signal I. After starting, the output Q does not respond to a change in the value of the input signal during the interval T. After the interval T has expired, the output signal is reset to LOW level.

Designation	Data type	Description
Inputs		
I	BOOL	Turning on the timer
T	TIME	Pulse duration
Outputs		
Q	BOOL	Timer output

```
FUNCTION_BLOCK TP_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        TP_1: SYS.TP;
    END_VAR

    //code area
    TP_1(I := I_in, T := T#1000ms);
    //where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days
    Q_out := TP_1.Q;
END_FUNCTION_BLOCK
FUNCTION_BLOCK TP_timer //milliseconds

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
        T_in : UDINT := 5000;//milliseconds
    END_VAR

    VAR_OUTPUT //declaration of output variables

        Q_out : BOOL;
    END_VAR
```

```
    VAR
        TP_1: SYS.TP;
        T_time: TIME;
    END_VAR

    //code area

    T_time := UDINT_TO_TIME(T_in);
    TP_1(I := I_in, T := T_time, Q => Q_out);

END_FUNCTION_BLOCK
```

11.6.2.2 ON-delay timer (SYS.TON)

The ON-delay timer (SYS.TON) is used for the signal transmission delay operation. The timer output Q will produce a HIGH level signal with a delay relative to the input signal front I of at least the duration T and will turn off at the input signal fall.

Designation	Data type	Description
Inputs		
I	BOOL	Timer start (on rising edge)
T	TIME	Delay on power-on
Outputs		
Q	BOOL	Timer output

```
FUNCTION_BLOCK TON_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        TON_1: SYS.TON;
    END_VAR

    //code area

    TON_1(I := I_in, T := T#1000ms);
    //where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days
    Q_out := TON_1.Q;

END_FUNCTION_BLOCK
```

```
FUNCTION_BLOCK TON_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
        Ton_in : UDINT := 5000;//milliseconds
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        TON_1: SYS.TON;
        Ton_time: TIME;
    END_VAR

    //code area

    Ton_time := UDINT_TO_TIME(Ton_in);
    TON_1(I := I_in, T := Ton_time, Q => Q_out);

END_FUNCTION_BLOCK
```

11.6.2.3 OFF-delay timer (SYS.TOF)

The OFF-delay timer (SYS.TOF) is used to delay the output off. The timer output Q will show a HIGH level signal on the rising edge of the signal at the input I, the countdown of the off-delay time T will start on each falling edge of the input signal. After the input signal is off, the output will show a LOW level signal with a delay of T.

Designation	Data type	Description
Inputs		
I	BOOL	Timer start (on falling edge)
T	TIME	Delay on power-on
Outputs		
Q	BOOL	Timer output

```
FUNCTION_BLOCK TOF_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR
```

```
    VAR
        TOF_1: SYS.TOF;
    END_VAR

    //code area

    TOF_1(I := I_in, T := T#1000ms);
    //where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days
    Q_out := TOF_1.Q;

END_FUNCTION_BLOCK
FUNCTION_BLOCK TOF_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
        Tof_in : UDINT := 5000;//milliseconds
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        TOF_1: SYS.TOF;
        Tof_time: TIME;
    END_VAR

    //code area

    Tof_time := UDINT_TO_TIME(Tof_in);
    TOF_1(I := I_in, T := Tof_time, Q => Q_out);

END_FUNCTION_BLOCK
```

11.6.2.4 Timer (SYS.CLOCK)

The timer (SYS.CLOCK) is used to generate a pulse to turn on the Q output according to the real-time clock. The output turn-on time Ton and turn-off time Toff are set as timer parameters.

Designation	Data type	Description
Inputs		
Ton	DT	Turn-on time
Toff	DT	Shutdown time
Mask	UDINT	Selection of quantities to be used

Designation	Data type	Description
Output		
Q	BOOL	Timer output

NOTE
Specifying the Mask variable is optional.
If the value of the Mask variable is not specified, the block defaults to a Mask = 63 (0b111111),
Where:
Mask = 63 (0b111111)
0 bit - if 1, then seconds are used
1 bit - if 1, then minutes are used
2 bits - if 1, then the hours is used
3 bits - if 1, then days are used
4 bits - if 1, then months are used
5 bits - if 1, then years are used

```
FUNCTION_BLOCK CLOCK_timer //Function block name

    VAR_INPUT //declaration of input variables
       I_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        CLOCK_1: SYS.CLOCK;
    END_VAR

    //code area


    CLOCK_1(Ton := DT#2023-09-28-7:20:55, Toff := DT#2023-09-28-12:30:59);

    Q_out := CLOCK_1.Q;

END_FUNCTION_BLOCK
```

11.6.2.5 Weekly timer (SYS.CLOCKWEEK)

The weekly timer (SYS.CLOCKWEEK) is used to generate a pulse to turn on the output Q according to the real-time clock, taking into account the days of the week. The time of turning on Ton and turning off Toff of the output Q and the days of the week of operation are set as timer parameters.

Designation	Data type	Description
Inputs		
Ton	DT	Turn-on time

Designation	Data type	Description
Toff	DT	Shutdown time
DayOfWeekMask	UDINT	Selecting the days to use
DateTimeMask	UDINT	Selection of quantities to be used
Outputs		
Q	BOOL	Timer output

NOTE
Specifying the DayOfWeekMask and DateTimeMask variables is optional.
If the value of the DayOfWeekMask variable is not specified, then the block defaults to a DayOfWeekMask = 127 (0b1111111),
If the value of the DateTimeMask variable is not specified, then the block defaults to a DateTimeMask = 63 (0b111111),
Where:
DayOfWeekMask = 127 (0b1111111)
0 bit - if 1, then Mondays are taken into account
1 bit - if 1, then Tuesdays are taken into account
2 bits - if 1, then Wednesdayы are taken into account
3 bits - if 1, then Thursdays are taken into account
4 bits - if 1, then Fridays are taken into account
5 bits - if 1, then Saturdays are taken into account
6 bits - if 1, then Sundays are taken into account

DateTimeMask = 63 (0b111111)
0 bit - if 1, then seconds are used
1 bit - if 1, then minutes are used
2 bits - if 1, then the clock is used
3 bits - if 1, then days are used
4 bits - if 1, then months are used
5 bits - if 1, then years are used

```
FUNCTION_BLOCK CLOCKWEEK_timer //Function block name

    VAR_INPUT //declaration of input variables
       I_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        CLOCKWEEK_1: SYS.CLOCKWEEK;
    END_VAR

    //code area

    CLOCKWEEK_1(Ton := DT#2023-09-28-7:20:55, Toff := DT#2023-09-28-12:30:59);
    Q_out := CLOCKWEEK_1.Q;
END_FUNCTION_BLOCK
```

11.6.3 Generators

– *Pulse generator (SYS.BLINK) 11.6.3.1*.

11.6.3.1 Pulse generator (SYS.BLINK)

The pulse generator (SYS.BLINK) is used to form rectangular pulses. At the output Q of the generator, pulses are formed with specified parameters of the duration of the on (Th – HIGH level signal) and off (Tl – LOW level signal) state for the duration of the control signal at the input I (HIGH level signal).

Designation	Data type	Description
Inputs		
I	BOOL	Work permission
Th	TIME	Duration of a logical unit
Tl	TIME	Logical zero duration
Outputs		
Q	BOOL	Выход генератора

```
FUNCTION_BLOCK BLINK_generator //Function block name

    VAR_INPUT //declaration of input variables
       I_in : BOOL := FALSE;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        BLINK_1: SYS.BLINK;
    END_VAR

    //code area

    BLINK_1(I := I_in, Th := T#1000ms, Tl := T#1000ms);
    //where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days

    Q_out := BLINK_1.Q;

END_FUNCTION_BLOCK
FUNCTION_BLOCK BLINK_generator //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
        Th_in : UDINT := 5000;//milliseconds
        Tl_in : UDINT := 5000;//milliseconds
```

```
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        BLINK_1: SYS.BLINK;
        Th_time: TIME;
        Tl_time: TIME;
    END_VAR

    //code area

    Th_time := UDINT_TO_TIME(Th_in);
    Tl_time := UDINT_TO_TIME(Tl_in);
    BLINK_1(I := I_in, Th := Th_time, Tl := Tl_time, Q => Q_out);

END_FUNCTION_BLOCK
```

11.6.4 Counters

- *Threshold counter with self-reset (SYS.CT) 11.6.4.1*;
- *Universal counter (SYS.CTN) 11.6.4.2*;
- *Threshold counter (SYS.CTU) 11.6.4.3*.

11.6.4.1 Threshold counter with self-reset (SYS.CT)

The threshold counter with self-reset (SYS.CT) is used to count a specified number of pulses N (input N is the pulse number setting). At the output Q of the counter, a pulse of the HIGH level signal with the duration of the device working cycle (cycle time) will appear if the number of pulses arriving at the input C reaches the set value N.

Designation	Data type	Description
Inputs		
C	BOOL	Counter input
N	UDINT	Counter setting
Outputs		
Q	BOOL	Signaling of the setpoint reached (duration one cycle)

```
FUNCTION_BLOCK CT_counter //Function block name

    VAR_INPUT //declaration of input variables
        C_in : BOOL;
        N_in : UDINT := 10;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        CT_1: SYS.CT;
    END_VAR

    //code area

    CT_1(C := C_in, N := N_in, Q => Q_out);

END_FUNCTION_BLOCK
```

11.6.4.2 Universal counter (SYS.CTN)

The universal counter (SYS.CTN) is used for direct and indirect counting. The "direct counting" operation is performed by the rising edge of the pulse at the direct counting input U, which increases the value of the output signal Q. Pulses arriving at the input D ("decrease counting") decrease the value of the output Q. If a logical "1" signal arrives at the input R, the output of the counter Q is set to the value of the input N.

Designation	Data type	Description
Inputs		
U	BOOL	Direct Counting
D	BOOL	Coutdown
R	BOOL	Reset the output state to value N
N	UDINT	Setpoint
Outputs		
Q	UDINT	Cumulative value of pulses

```
FUNCTION_BLOCK CTN_counter //Function block name

    VAR_INPUT //declaration of input variables
        U_in : BOOL;
        D_in : BOOL;
        R_in : BOOL;
        N_in : UDINT := 10;
    END_VAR
```

```
    VAR_OUTPUT //declaration of output variables
        Q_out : UDINT;
    END_VAR

    VAR //declaration of local variables

        CTN_1: SYS.CTN;
    END_VAR

    //code area

    CTN_1(U := U_in, D := D_in, R := R_in, N := N_in, Q => Q_out);

END_FUNCTION_BLOCK
```

11.6.4.3 Threshold counter (SYS.CTU)

The threshold counter (SYS.CTU) is used to count the number of pulses arriving at the C input. A pulse of the HIGH level signal will appear at the Q counter output if the number of pulses arriving at the input reaches the set value at the N input (N is the setpoint).

Designation	Data type	Description
Inputs		
U	BOOL	Direct Counting
R	BOOL	Reset the counter state to 0
N	UDINT	Setpoint
Outputs		
Q	BOOL	Signaling of setpoint reached

```
FUNCTION_BLOCK CTU_counter //Function block name

    VAR_INPUT //declaration of input variables
        C_in : BOOL;
        R_in : BOOL;
        N_in : UDINT := 10;
    END_VAR

    VAR_OUTPUT //declaration of output variables

        Q_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        CTU_1: SYS.CTU;
    END_VAR
```

```
    //code area

    CTU_1(C := C_in, R := R_in, N := N_in, Q => Q_out);

END_FUNCTION_BLOCK
```