

Contents

15. Troubleshooting	15-3
15.1. Categories of Errors.....	15-4
15.2. STEP 7 - Test Functions, Overview.....	15-5
15.3. Overview Window	15-6
15.3.1. Global Search / Find and Replace in the Editor.....	15-7
15.4. Detailed Information in the Overview of Addresses.....	15-8
15.5. System Diagnostics - Overview	15-9
15.6. Status LEDs	15-10
15.6.1. Status LEDs of the S7-1500 CPU.....	15-10
15.6.2. Status LEDs of the Central DI/DO Modules of the S7-1500 CPU	15-11
15.7. Hardware Diagnostics.....	15-12
15.8. Online & Diagnostics: General.....	15-13
15.8.1. Online & Diagnostics: CPU Diagnostics Buffer.....	15-14
15.8.2. CPU Diagnostics Buffer: Interpreting Error Messages	15-15
15.8.3. CPU Diagnostics Buffer: Opening a Faulty Block.....	15-16
15.9. Call Hierarchy (Block Stack)	15-17
15.9.1. Exercise 1: Creating a Program Backup Copy in the Project Library	15-18
15.9.2. Exercise 2: Copying the Faulty Program	15-19
15.9.3. Exercise 3: STOP Troubleshooting Worksheet	15-20
15.10. Monitor Block (Block Status).....	15-21
15.10.1. Monitor Block: Modify Tags.....	15-22
15.10.2. Monitoring Structures.....	15-23
15.10.3. Monitor Block: Trigger Conditions / Call Environment	15-24
15.11. Monitor / Modify Variables (Tags): Watch Tables.....	15-25
15.11.1. Monitor / Modify Variables (Tags): Trigger Points	15-26
15.11.2. Enable Peripheral Outputs (in planning for S7-1500)	15-27
15.11.3. Force Variables (Tags)	15-28
15.12. Reference Data: Cross-references of Tags	15-29
15.12.1. Reference Data: Cross-references / Show Overlapping Accesses	15-30
15.12.2. Reference Data: Cross-references of a Variable (Tag) in the Block Editor.....	15-31
15.12.3. Go To.....	15-32
15.12.4. Reference data: Assignment I/Q/M/T/C.....	15-33
15.12.5. Reference Data: Call Structure.....	15-34
15.12.6. Reference Data: Dependency Structure.....	15-35
15.13. Resources.....	15-36
15.14. Reference Projects	15-37
15.15. Compare (1) - Offline / Online.....	15-38
15.15.1. Compare (2) – Block Detailed Comparison	15-39
15.15.2. Compare (3) - Software Offline / Offline	15-40

15.15.3.	Compare (4) - Hardware Offline / Offline	15-41
15.16.	Exercise 4: Testing the Motor Jog	15-42
15.17.	Exercise 5: Entering the Setpoint Quantity	15-43
15.18.	Exercise 6: Testing the Evaluation of Fault 3	15-44
15.19.	TRACE Analyzer Function	15-45
15.19.1.	Configuring a TRACE - Signals and Sampling	15-46
15.19.2.	Configuring a TRACE – Trigger and Saving Measurement on Device.....	15-47
15.19.3.	Downloading a TRACE into the CPU and Activating It.....	15-48
15.19.4.	Evaluating, Saving, Exporting a TRACE in STEP 7	15-49
15.19.5.	Trace Task Card	15-50
15.20.	Task Description: Creating, Looking at and Saving a TRACE	15-51
15.20.1.	Exercise 7: Creating a Cyclic Interrupt OB for the Recording Level.....	15-52
15.20.2.	Exercise 8: Creating, Looking at and Saving a TRACE	15-53
15.21.	Additional Information	15-55
15.21.1.	Diagnostic Information about the SIMATIC Memory Card	15-56
15.21.2.	Reading-out the Checksum	15-57

15. Troubleshooting

At the end of the chapter the participant will...

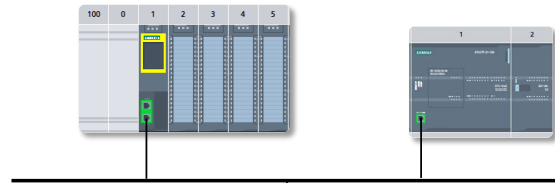


- ... be familiar with the search functions
- ... be familiar with the possibilities of troubleshooting
- ... be familiar with the possibilities of project analysis
- ... be able to apply troubleshooting functions for STOP troubleshooting and to be able to use them for error correction
- ... be able to apply troubleshooting functions for searching for logical errors and to be able to use them for error correction
- ... be able to use the Trace analysis function

15.1. Categories of Errors

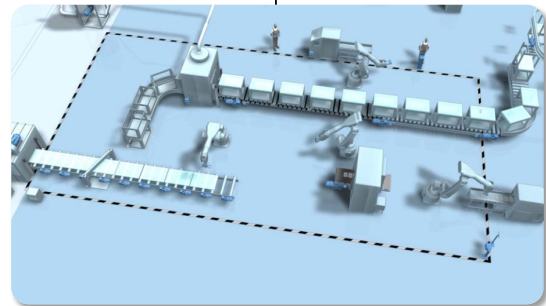
Errors Detected by the System

- Acquiring, evaluating and indicating errors within a PLC
 - Module failure
 - Short-circuit in signal cables
 - Scan time overrun
 - Programming error (accessing a non-existent block)



Functional Errors

- Desired function is either not executed at all or is not correctly executed
 - Process fault (sensor/actuator, cable defective)
 - Logical programming error (not detected during creation and commissioning)



Monitoring Functions

Diagnosis is important in the operating phase of a system or machine. Diagnosis usually occurs when a problem (disturbance) leads to standstill or to the incorrect functioning of the system or machine. Due to the costs associated with downtimes or faulty functions, the associated cause of the disturbance has to be found quickly and then eliminated.

Categories of Errors

Errors that occur can be divided into two categories, depending on whether or not they are detected by the PLC:

- Errors that are detected by the PLC's operating system and that normally lead to the Stop state of the CPU.
- Functional errors, that is, the CPU executes the program as usual, but the desired function is either not executed at all or it is executed incorrectly. The search for these types of errors is much more difficult, as a rule, since the cause of the error is initially hard to determine.

Possible causes could be:

- A logical programming error (software error) that was not detected during creation and commissioning of the user program and probably occurs only on extremely rare occasions.
- A process fault that was triggered by the faulty functioning of components directly associated with the process control, such as cables to sensors/actuators or by a defect in the sensor/actuator itself.

15.2. STEP 7 - Test Functions, Overview

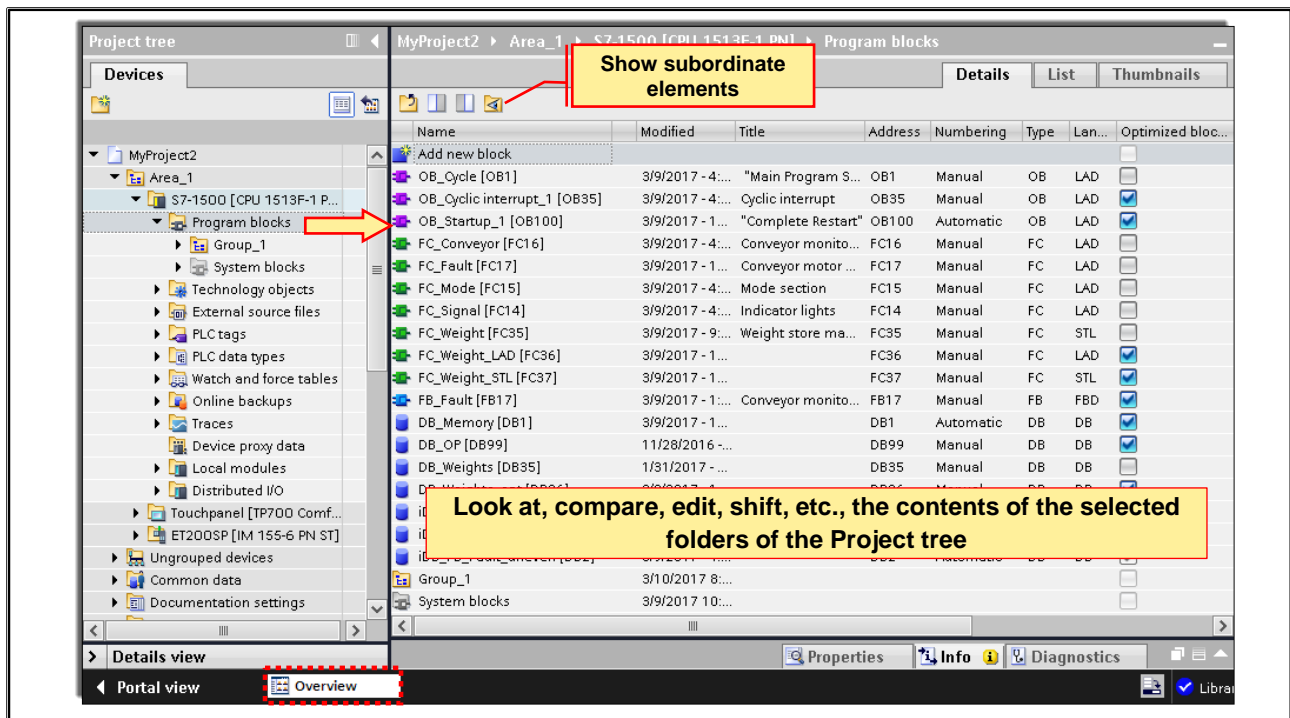
Errors Detected by the System General rule: CPU in Run	Functional Errors General rule: CPU in RUN
<ul style="list-style-type: none"> • Programming error (w/o OB121): STOP • Access error (w/o OB122): RUN • Asynchronous error (w/o OB82, 83, 86): RUN • Online & Diagnostics <ul style="list-style-type: none"> - Diagnostics buffer, • Task Card "Testing" <ul style="list-style-type: none"> - Call hierarchy/Block stack - Local data stack (in planning stage) • Diagnose Modules <ul style="list-style-type: none"> - Diagnostic status (all modules) 	<ul style="list-style-type: none"> • Process fault (e.g. wire break) • Logical programming error (e.g. double assignment) • Monitor and Modify Variables <ul style="list-style-type: none"> → Watch and Force tables • Monitor Blocks (Block Status) <ul style="list-style-type: none"> → Monitor in the Blocks editor <ul style="list-style-type: none"> - with call environment • Tools <ul style="list-style-type: none"> - Cross references - Assignment list (I/Q/M/T/C)
<ul style="list-style-type: none"> • "Trace" analyzer function • Program/Block comparison • Set breakpoints 	

Test Functions

There are various STEP 7 test functions for troubleshooting, depending on the type of error caused:

- when CPU in STOP
For errors that are detected by the system, the test functions Diagnostics buffer, Call hierarchy/Block stack, Local data stack and Hardware diagnostics give detailed information on the cause of the error and the location of the interruption. By programming Error OBs, information on the error that occurred can be evaluated by program and the transition of the CPU into the STOP state can be prevented. If the CPU has stopped, the use of the test functions Monitor / Modify Variable and Monitor Blocks makes little sense since the CPU neither reads nor outputs process images while in the STOP state, and also no longer executes the program.
- when CPU in RUN
Vice versa, it makes little sense, as a rule, to use test functions such as Local data stack for troubleshooting when the CPU is in RUN, since program execution has not been interrupted and the system does not provide any information on the error that occurred. The Module Information test function merely provides general information on the CPU's operating status or on errors that occurred in the past. Functional errors can be diagnosed as follows:
 - Process Fault (such as a wiring error)
Wiring test of the inputs: Monitor Variable
Wiring test of the outputs: Enable Peripheral Outputs (only for CPU-STOP)
 - Logical Programming Errors (such as a double assignment)
All test functions listed, with the exception of Enable Peripheral Outputs, can be used for searching for logical program errors.
 - Force: Forced control of operands regardless of the program logic
 - Breakpoints: Program execution in single steps

15.3. Overview Window



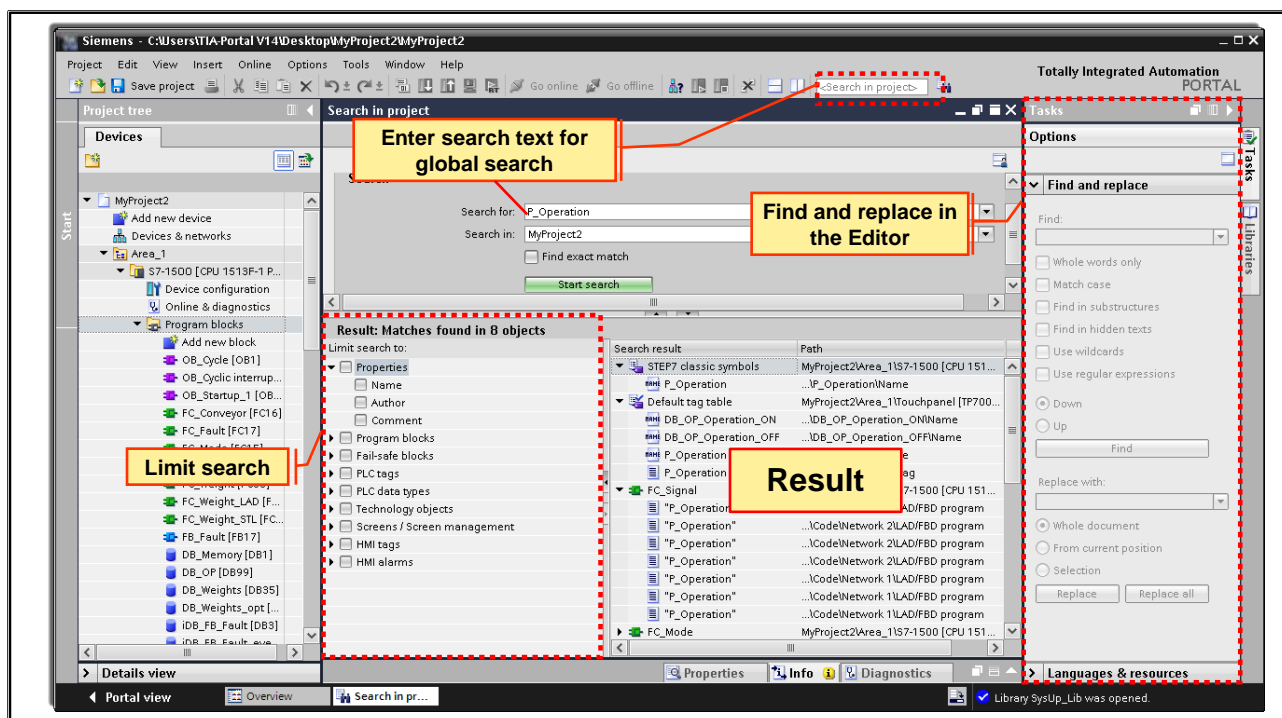
The Overview window is used to supplement the Project tree. The Overview window displays the contents of the folder currently selected in the Project tree.

As well, you can carry out the following actions in the Overview window:

- Open objects
- Display and edit Properties of objects in the Inspector window
- Rename objects
- Call object-specific actions via the Context menu
- Compare objects
- Carry out different object operations, such as, insert objects from the library using drag & drop, shifting, copying, inserting and deleting objects

It is possible to compare the contents of two folders or objects. To do so, the View can be split. Beyond that, it is possible to shift objects between the two split windows using drag & drop.

15.3.1. Global Search / Find and Replace in the Editor



Within the TIA Portal, you can use the following search possibilities:

- Search entire project (toolbar or CTRL+F when the focus is not in the editor (working) area)
- Find and replace within an editor ('Tasks' task card > Find and replace)
- Search the Hardware catalog

Search Entire Project

You can search the entire project for a specific text. For this, there is a Search editor available in which you can, for example, narrow down the search. The objects which contain the searched-for text are presented clearly in a table. You can open each object from the Search editor in order to look at the relevant location.

The Global Search (Search in project) can be started by means of CTRL+F (when the focus/selection is not in the editor (working) area), or, via the Context menu of the project name, or, via the menu bar, or, via the menu Edit > Search in project.

If the focus is in an editor, then the function "Find and replace" is started with the key combination CTRL+F.

Search and Replace within an Editor (CTRL+F when the focus is in the editor (working) area)

It is possible to search for texts within an editor. The search function finds all texts within the currently opened editor which contain the search term. The results are selected one after the other in the opened editor.

Furthermore, you have the following possibilities:

- Refine the search through additional options
- Replace found text

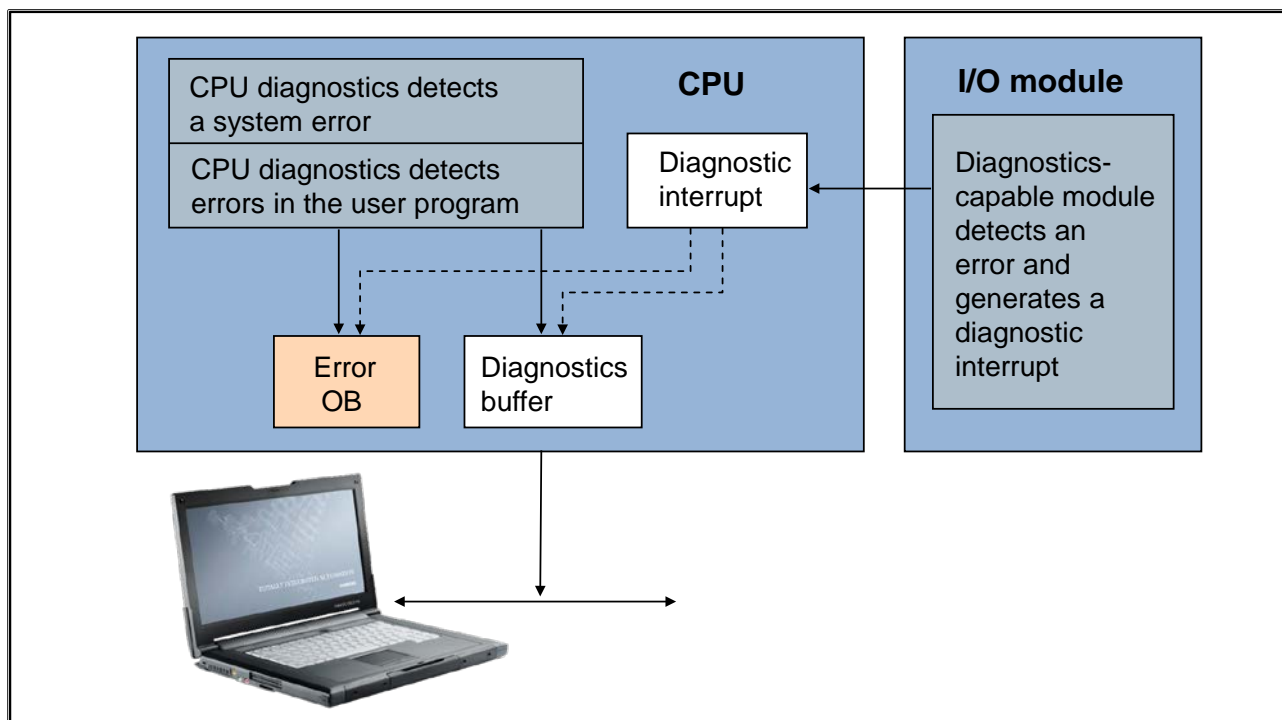
15.4. Detailed Information in the Overview of Addresses

The screenshot shows the SIMATIC TIA Portal interface. The main window displays a rack diagram for an S7-1500 CPU 1513F-1 PN. The rack has slots 0-7 and 15-31. Slot 0 contains a power supply, slot 1 contains the CPU, and slots 2-7 contain I/O modules. The Properties window is open to the 'Overview of addresses' tab, showing a table of I/O modules and their address ranges.

Type	Addr. fr...	Addr. to	Size	Module	Rack	Slot	Device name	Devic...	Master / IO system
I	10	25	16 Bytes	AI 8x12/12V/TC ST_1	0	4	S7-1500 [CPU 1513F-1 PN]	-	-
I	0	3	4 Bytes	DI 32x24VDC HF_1	0	2	S7-1500 [CPU 1513F-1 PN]	-	-
O	0	3	4 Bytes	DQ 32x24VDC/0.5A ST_1	0	3	S7-1500 [CPU 1513F-1 PN]	-	-
I	30	37	8 Bytes	AI4 x UI 2-Draht ST_1	0	4	ET200SP [IM 155-6 PN ST]	1	PROFINET IO-System...
O	30	37	8 Bytes	AQ4 x UI ST_1	0	5	ET200SP [IM 155-6 PN ST]	1	PROFINET IO-System...
I	5	5	1 Bytes	DIB x DC24V ST_2	0	2	ET200SP [IM 155-6 PN ST]	1	PROFINET IO-System...
I	4	4	1 Bytes	DIB x DC24V ST_1	0	1	ET200SP [IM 155-6 PN ST]	1	PROFINET IO-System...

In menu 'Overview of addresses' in the Properties of the CPU, an entire overview of addresses with central and distributed I/O can be displayed.

15.5. System Diagnostics - Overview



System Diagnostics

All those monitoring functions that deal with the correct functioning of the components of an automation system are grouped together under System Diagnostics. All S7-CPU have an intelligent diagnostics system. The acquisition of diagnostic data by the system diagnostics does not have to be programmed. It is integrated in the operating system of the CPU and in other diagnostics-capable modules and runs automatically. The CPU (temporarily) stores errors that occur in the diagnostics buffer and thus enables a fast and targeted error diagnosis by service personnel, even for sporadically occurring errors.

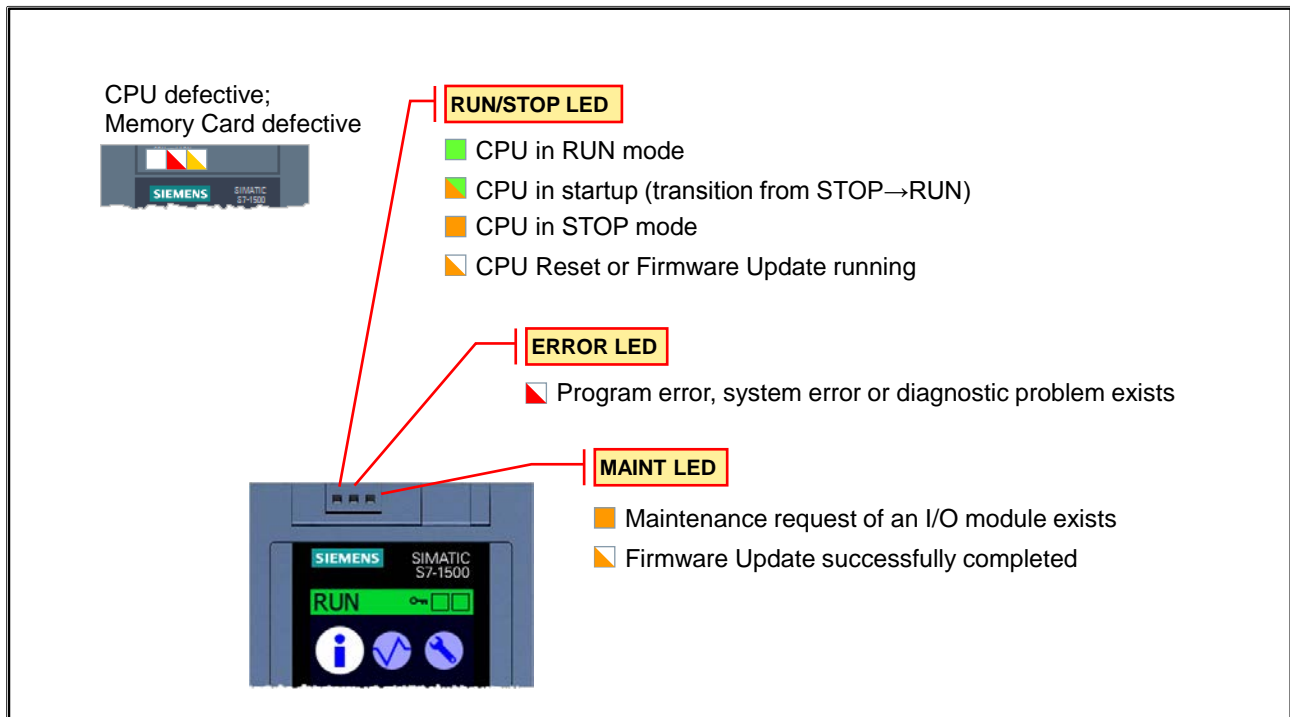
System Reaction

The operating system takes the following actions when it detects an error or a STOP event, such as an operating mode change (RUN → STOP):

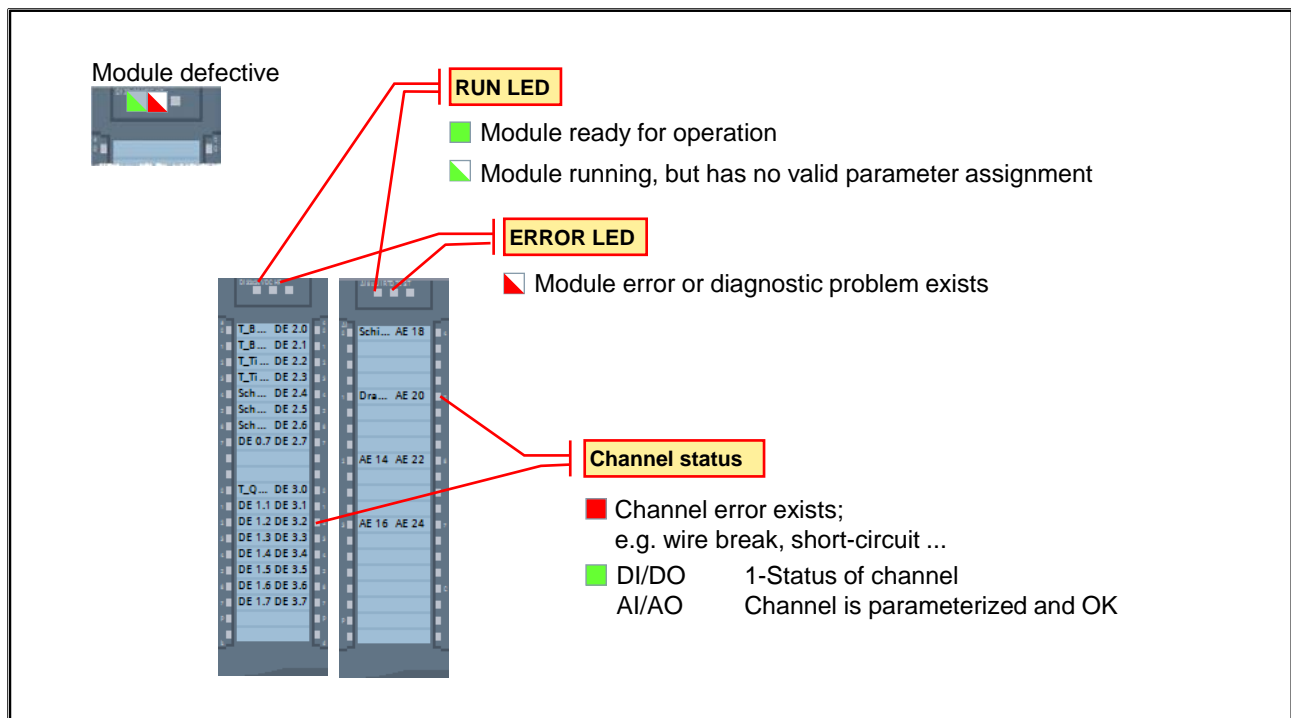
- A message on the cause and the effect of the occurring error is entered in the diagnostics buffer, complete with the date and time. The diagnostics buffer is a FIFO (circular) buffer on the CPU module for storing error events. The size of the diagnostics buffer depends on the CPU. In the FIFO buffer structure, the most recently entered message overwrites the oldest diagnostics buffer entry. A CPU Memory Reset cannot delete the diagnostics buffer.
- The Error OB associated with this error is called. This gives the user the opportunity of carrying out an own error handling.

15.6. Status LEDs

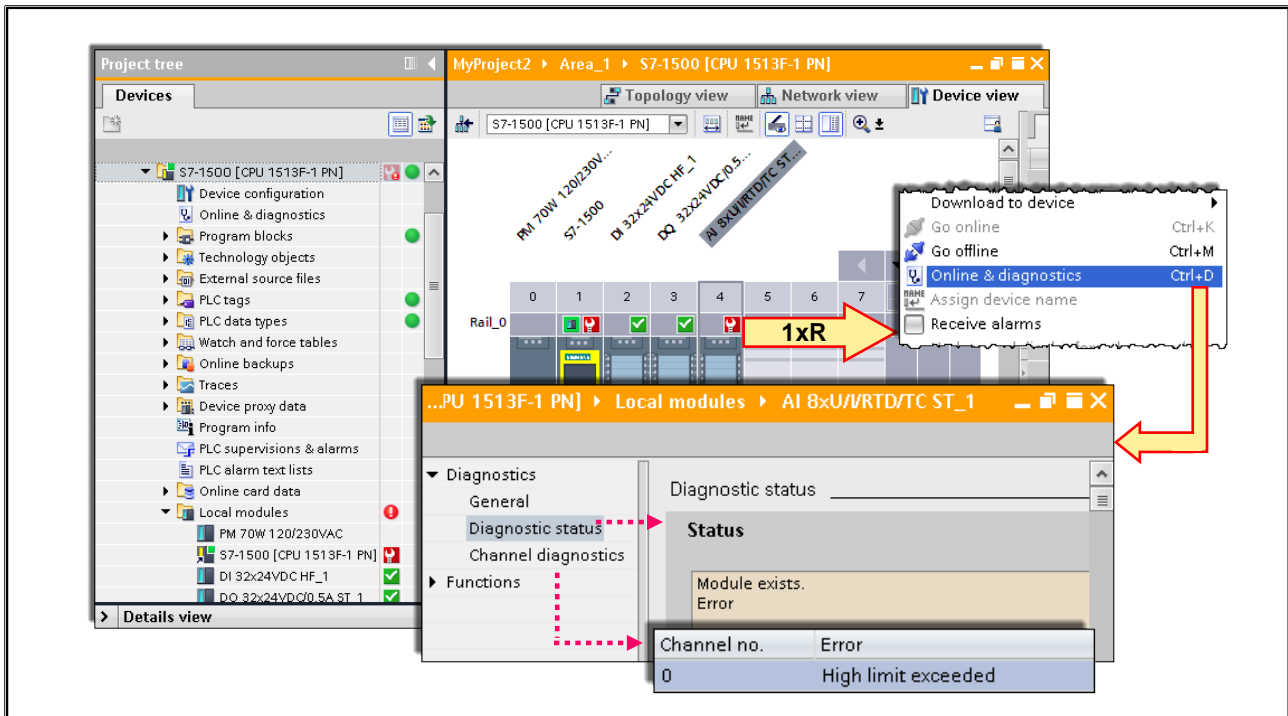
15.6.1. Status LEDs of the S7-1500 CPU



15.6.2. Status LEDs of the Central DI/DO Modules of the S7-1500 CPU



15.7. Hardware Diagnostics



Diagnosing Hardware

To use this function you must open the "Device configuration" and establish an online connection. The online view of the hardware gives information about the status or operating status of the modules. You can see that there is diagnostic information for a module when you see the diagnostic symbols that indicate the status of the associated module or the operating status of the CPU.

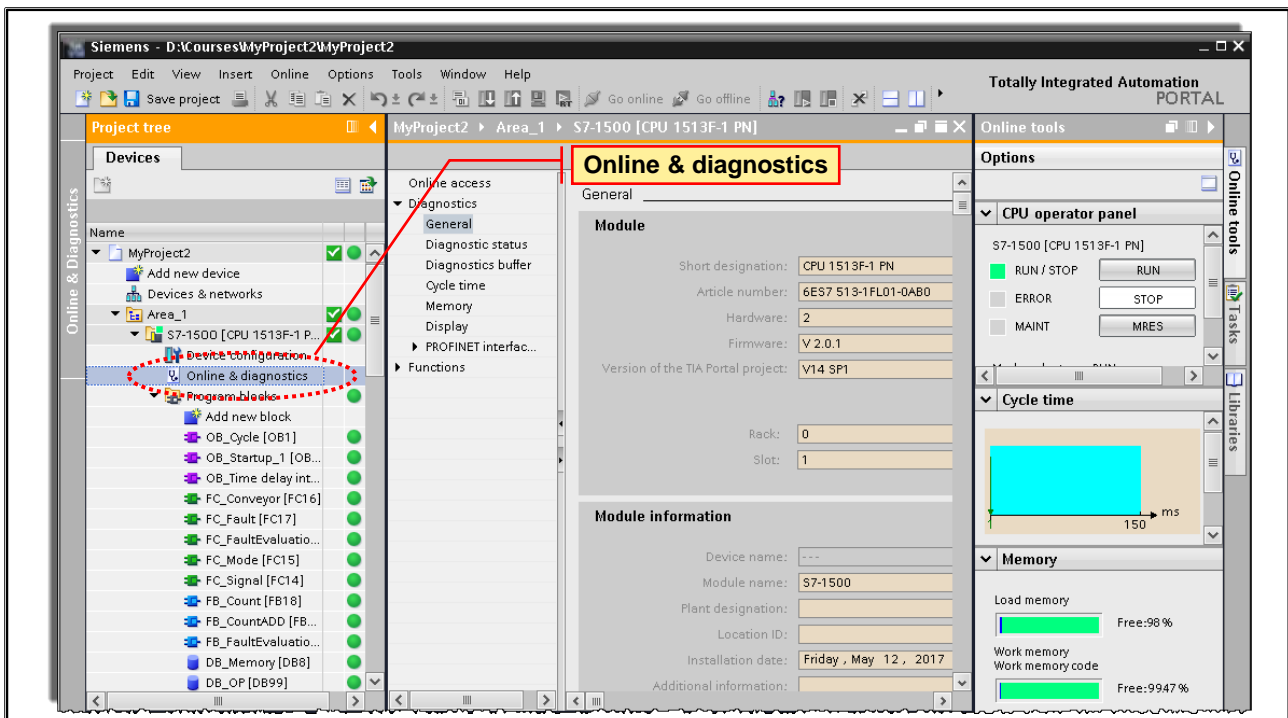
In the example shown, the analog input module (slot 4) has triggered a diagnostic interrupt. As a result, the CPU has gone into the STOP mode.

Both modules have been given symbols accordingly. Via the Context menu of the CPU and subsequent starting of "Online & diagnostics", the CPU's diagnostic buffer is output; for the analog module, the associated diagnostic data (see picture).

Symbol Meanings

Icon	Meaning	Icon	Meaning
	Hardware error in lower-level component: The online and offline versions differ (only in the project tree) in at least one lower-level hardware component.		Hardware error in lower-level component: The online and offline versions differ (only in the project tree) in at least one lower-level hardware component.
	No fault		Software error in lower-level component: The online and offline versions differ (only in the project tree) in at least one lower-level software component.
	Maintenance required		Online and offline versions of the object are different
	Maintenance demanded		Object only exists online
	Error		Object only exists offline
			Online and offline versions of the object are the same

15.8. Online & Diagnostics: General

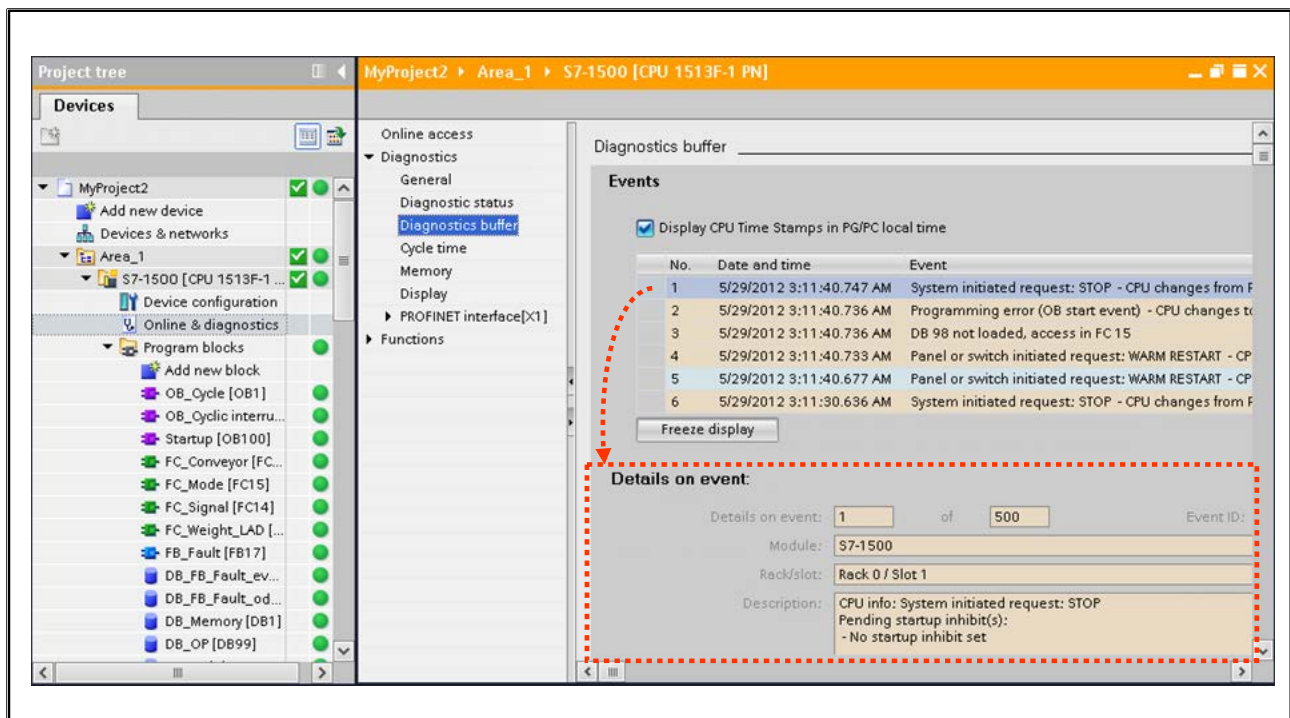


Online and Diagnostics

The Module Information function reads the most important data from the directly connected module. You will find additional information in the individual tabs:

- **General:** Among other things, the module designation, hardware and firmware versions
- **Diagnostics status:** Current status of the module
- **Diagnostics buffer:** It contains all diagnostic events in the order they occurred. In the display, all events are listed in plain language and in the order they occurred.
- **Cycle time:** Displays the selected minimum time and monitoring time as well as displaying the shortest, the longest and the current cycle time.
- **Memory:** Information about the entire size, how many bytes are used (occupied) and how many are free in the Load memory, Code work-memory, Data work-memory, and Retain memory.
- **Display:** General information about the Display used (Article number, Firmware etc.)

15.8.1. Online & Diagnostics: CPU Diagnostics Buffer



Diagnostics Buffer

The diagnostics buffer is a buffered memory area on the CPU organized as a circular buffer. It contains all diagnostics events (error messages, diagnostic interrupts, startup information etc.) of the CPU in the order in which they occurred. The highest entry is the last event to occur.

All events can be displayed on the programming device in plain language and in the order in which they occurred.



The size of the diagnostics buffer depends on the CPU. As well, not all of the diagnostics buffer is buffered with PowerOFF (only a part is retentive).

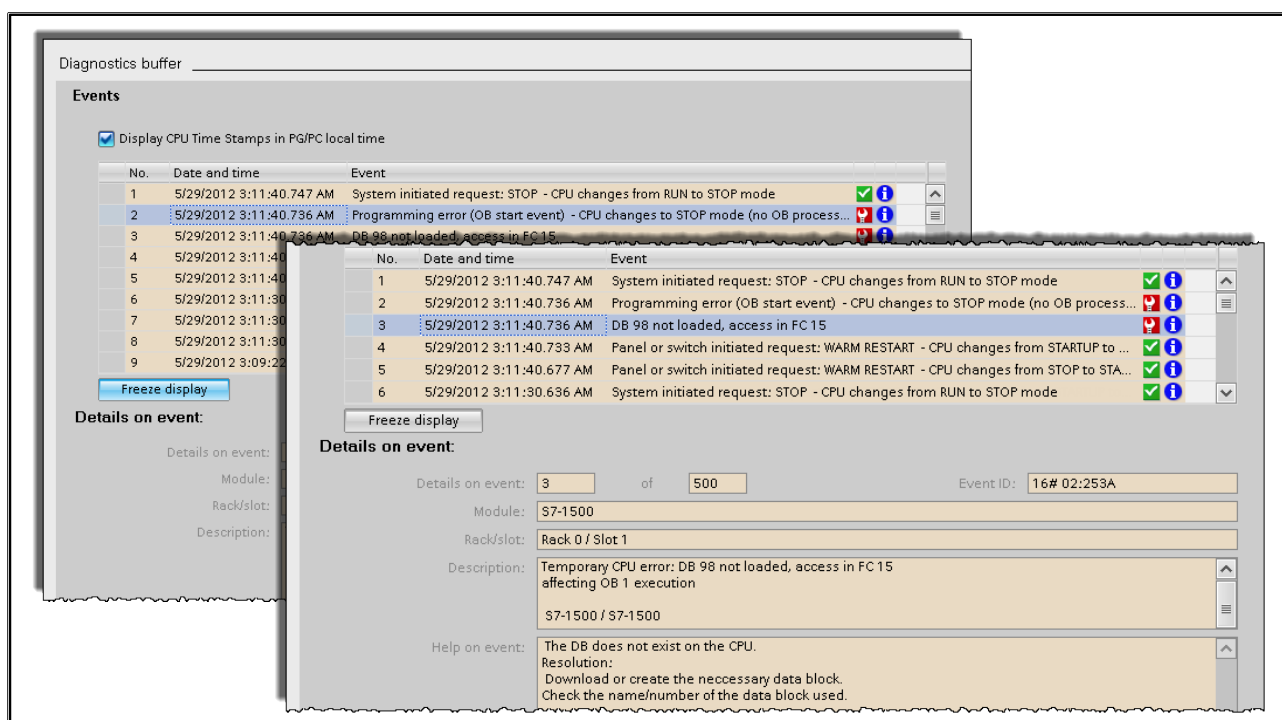
- Number of entries, 1000 to 3200
- Of that, 500 to 1000 retentive

Details on Event

Some additional information is also provided for the selected event in the "Details on event" box:

- Event name and number,
- Additional information depending on the event, such as, the address of the instruction that caused the event etc.

15.8.2. CPU Diagnostics Buffer: Interpreting Error Messages



Interpreting the Diagnostics Buffer

To interpret the diagnostics buffer, you have to look at the events that belong together in the sequence in which they occurred, in other words, from bottom to top:
For orientation:

- In our example, a WARM RESTART was performed before the most recent error occurred (events no.4 and 5).

Entries in the Diagnostics Buffer

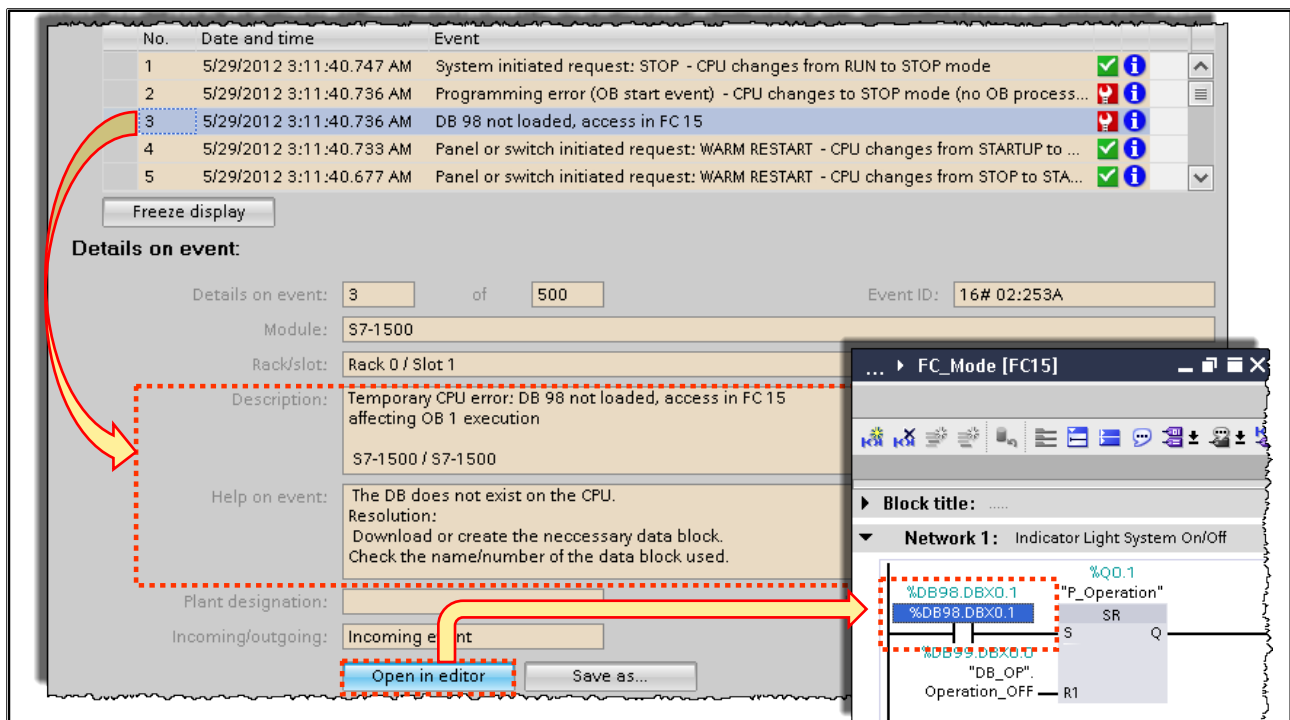
The last error that occurred after this warm restart leads to the following entries in the diagnostics buffer:

- Event No. 3:
DB 98 not loaded, access in FC15

Details on event:
 - DB 98 not loaded, access in FC15
(DB does not exist on the CPU)
 - affects OB1 execution
(FC15 is called in the cyclic program)
- Event No. 2:
Programming error (OB start event)
(if it exists, the operating system calls an OB for a programming error)

Details on event
 - CPU changes to STOP mode
 - no OB processing
(because the programming error OB121 was not programmed)
- Event No. 1:
– CPU changes to STOP mode

15.8.3. CPU Diagnostics Buffer: Opening a Faulty Block

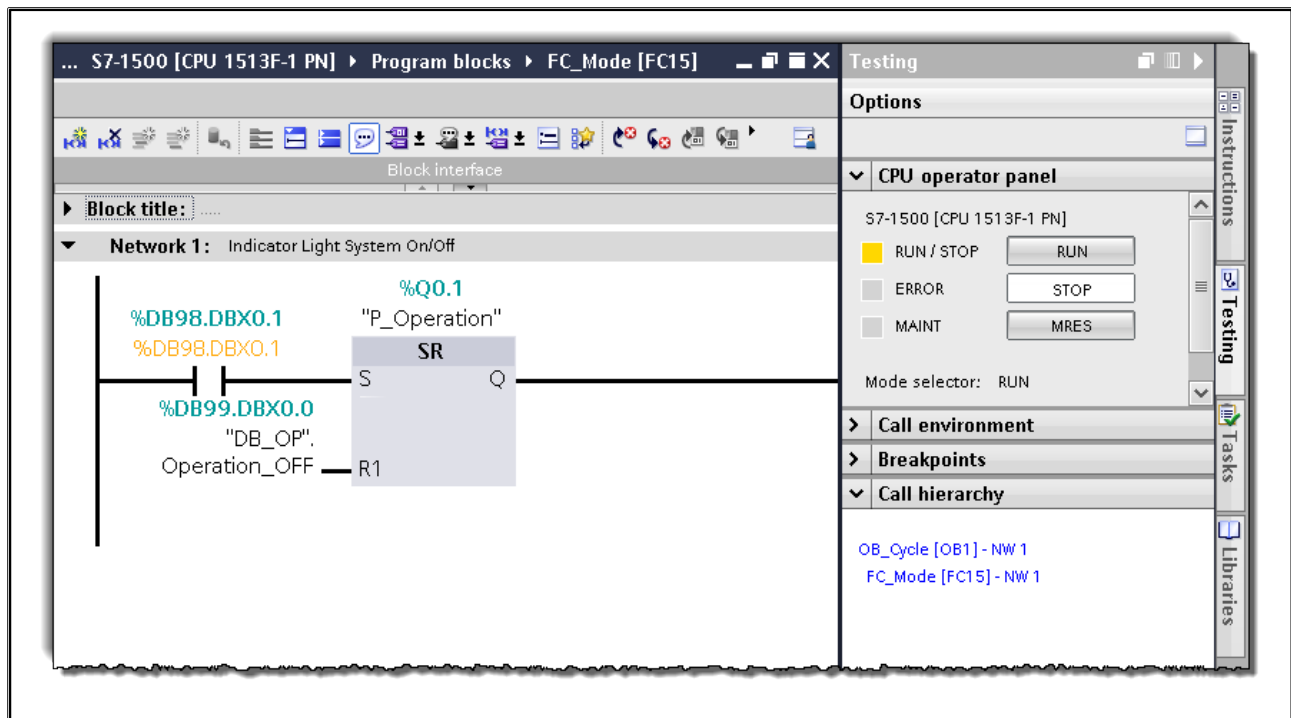


Opening a Block

For synchronous errors, that is, for errors that were triggered by a faulty instruction in the user program, you can open the block in which the interruption occurred by clicking on the "Open in editor" button.

If the SCL or STL language is selected, the cursor is positioned directly in front of the instruction that caused the interruption. In LAD/FBD, the network causing the interruption is highlighted. In the example shown, an attempt is made to access a data block which does not exist.

15.9. Call Hierarchy (Block Stack)



Call Hierarchy

The "Call hierarchy" gives you the information in which call path the block is opened.

If the block was opened from the diagnostic buffer via the button "Open in editor", then by looking at the entry in the Call hierarchy, you can see in which path the error occurred.

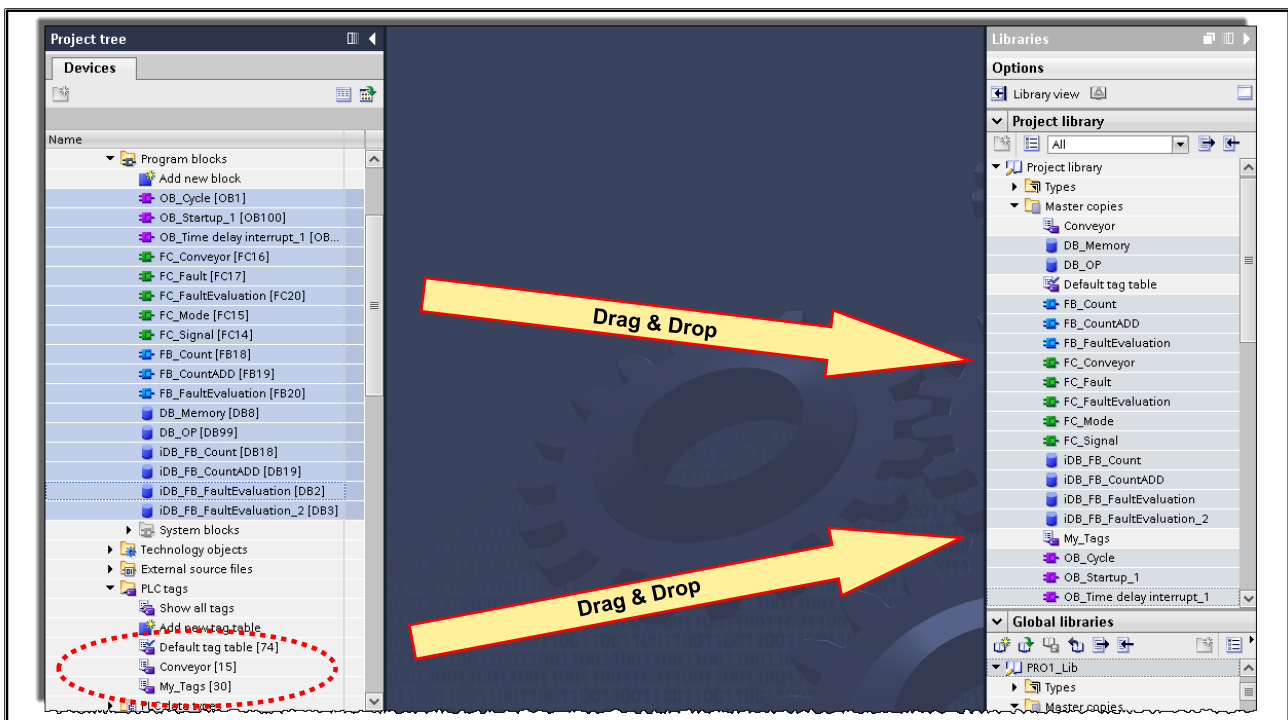
You can open the calling block by clicking on the relevant (appropriate) Link.

Note:

If the CPU is in "STOP" mode, the current block stack at the time of the STOP transition can be read out via the call hierarchy. The block stack lists all blocks whose execution was not completed at the moment when the CPU went into "STOP". The blocks are listed in the order in which the execution was started.

With an existing online connection, open any block for this and switch into the Task Card "Test" > Call hierarchy.

15.9.1. Exercise 1: Creating a Program Backup Copy in the Project Library



Task

You are to make a backup copy of your own program and the associated tags in the Project library of your project "MyProject2" since you will be working with a prepared faulty program afterwards.

What to Do

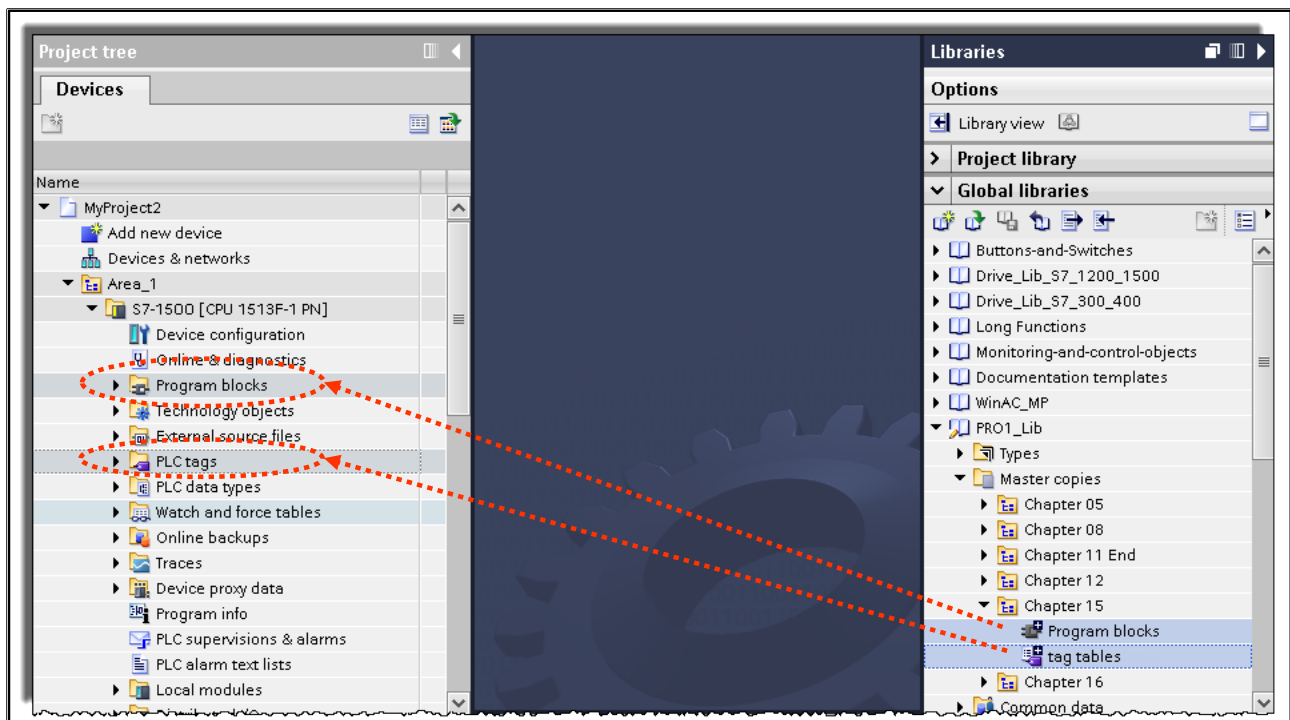
1. In the "Libraries", open the Project library.
2. Select all S7 blocks of the CPU program and copy them into your Project library using drag & drop.
3. Select all tag tables and copy them into your Project library using drag & drop.
4. Save your project and with that also the Project library.

Note:

If you would like to copy several objects combined as one object into the library, then highlight the objects, select "Copy" in the Context menu or press CTRL+C. Then, in the library, in the Context menu of the Master copies folder, you will find the command "Paste as a single master copy".

Combined objects can also only be copied together into a project/device.

15.9.2. Exercise 2: Copying the Faulty Program



Task

You are to copy a prepared program from the "PRO1_Lib" Global library into your own project.

What to Do

1. In your project, delete all S7 blocks and all PLC tag tables.
2. In the task card Libraries, open the "Global library" PRO1_Lib
<Drive>:\02_Archives\TIA_Portal \TIA-PRO1\PRO1_Lib
3. Using drag & drop, copy the blocks "Program blocks" and the PLC tag tables "tag tables" into your project (see picture).
4. Download all blocks of the faulty program into the CPU by selecting the CPU in the Project tree and then clicking on the Download button.
5. Carry out a CPU restart.
6. Save your project.

15.9.3. Exercise 3: STOP Troubleshooting Worksheet

Task

The faulty program contains one STOP error which if now to be corrected by you. If the CPU remains in RUN after error correction and subsequent restart, the exercise has been successfully completed.

In addition to the error (STOP error) detected by the system, the program also contains logical errors (RUN errors) so that the correct functioning of the program is still not established even after the STOP error is eliminated. The logical errors will be eliminated in the next exercises.

What to Do

Please note that after every STOP error correction, a CPU restart must be carried out. If the CPU once again goes into the STOP mode after the restart, a further STOP error exists or/that is, the error is not eliminated.

During error correction, answer the following questions on the error that occurs:

- STOP error determined:



- Interrupted block:
- Error
- Correction: (old instruction -> new instruction)
-

15.10. Monitor Block (Block Status)

Monitor block On / Off

Online values are being updated

		Address	RLO	Value	Extra
1	A	"DB_OP".Operation_ON	%DB99.DBX...	1	
2	FP	"M_aux_Op_on"	%M15.1	0	
3	S	"P_Operation"	%Q0.1	0	
4	A	"DB_OP".Operation_OFF	%DB99.DBX...	0	
5	R	"P_Operation"	%Q0.1	0	
6	NOP	0			

Area of Use

The Monitor Block test function is used to be able to follow the program execution within a block. For this, the states or contents of the operands used in the block at the time of program execution are displayed on the screen. You can activate (switch on) the "Monitor" ("Block Status") test mode for the block which is currently open in the LAD/STL/FBD Editor by clicking the Glasses icon.

At the beginning of the test function, it is insignificant whether the block to be monitored is opened online or offline in the Editor. Should, however, the block opened offline not match the block saved online in the CPU, you first either have to open the block saved online or load the block opened offline into the CPU and then monitor it.

In the test mode, the states of the operands and LAD / FBD elements are displayed in different colors. You define these by selecting the menu option Options → Settings:

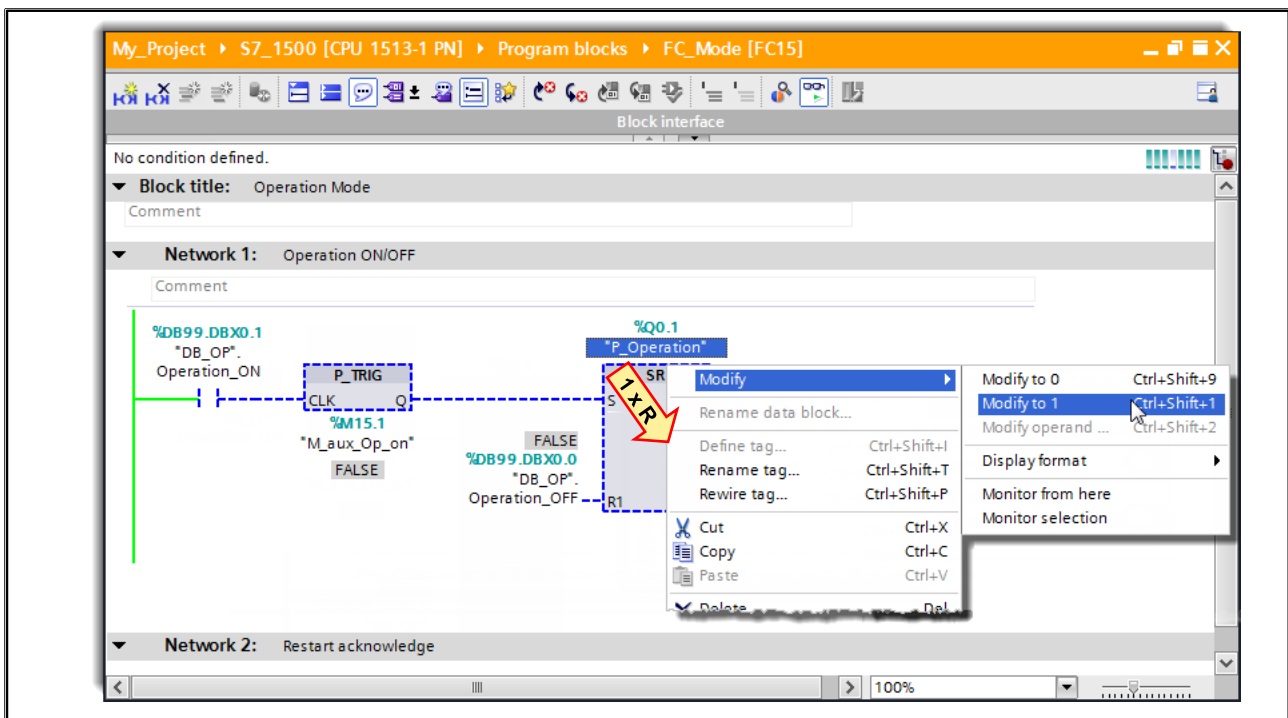
Examples:

- Status fulfilled → "Element is displayed in green"
- Status not fulfilled → "Element is displayed in blue"

Notes

The monitoring values are only active when the CPU is in RUN mode and the instructions to be monitored are being processed!
This is indicated by the progress bar "Online values are being updated" in the upper right corner of the block.

15.10.1. Monitor Block: Modify Tags



Modify Tags

When the "Monitor block" test function is activated, it is possible to modify tags to status '0' or '1'. The assignment of the status occurs once. When you use tags that are not Boolean, you can modify via the menu item "Modify operand...".

If the tag, whose status was changed, is not overwritten by the program, the tag remains at the assigned status. If, for example, an output is modified to status '1' and this tag is not overwritten by the program, the output remains switched on or to status '1'.

15.10.2. Monitoring Structures

Monitor Structures

For INOUT parameters Input value and Output value

Name	Data type	Input value	Output value
▼ "DB_Weights_opt".Wei...	"UDT_WeightStore"		
MaxNo	Int	10	10
ActNo	Int	0	1
▼ PartWeight	Array[0..9] of Int		
PartWeight[0]	Int	238	238
PartWeight[1]	Int	0	160
PartWeight[2]	Int	0	0
PartWeight[3]	Int	0	0

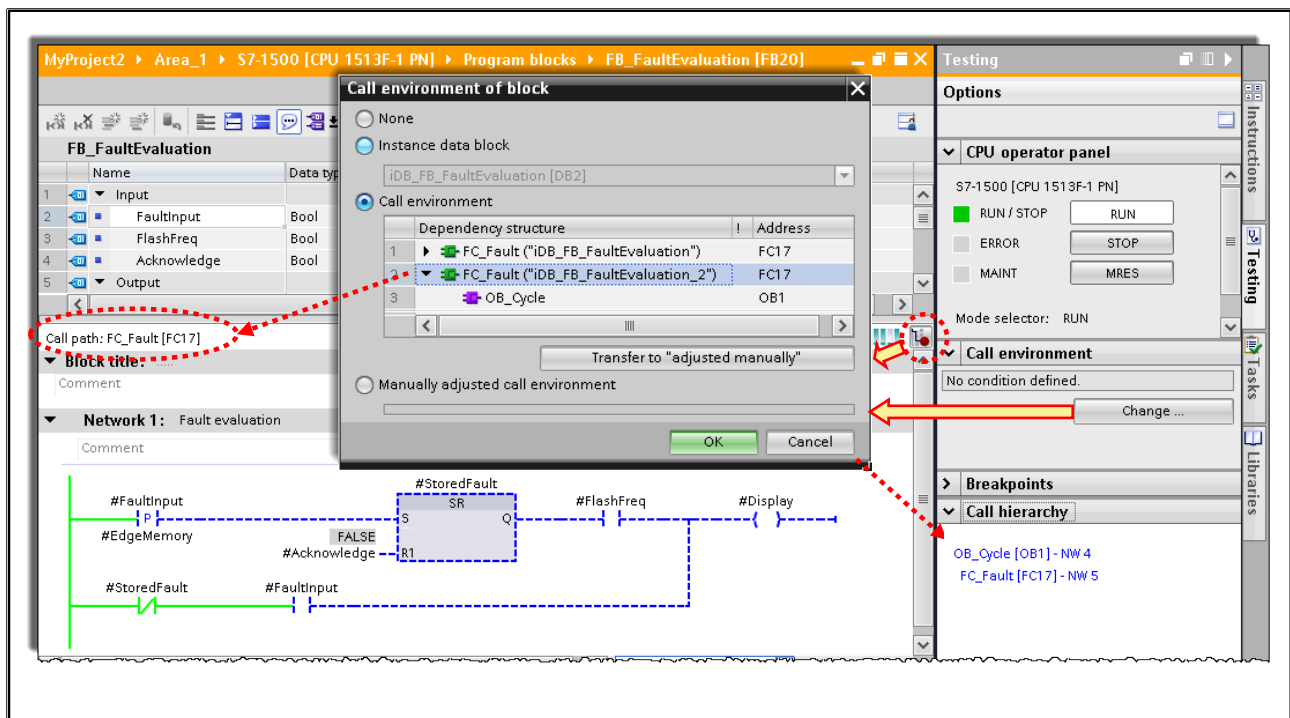
Rules for Monitoring Structures (S7-1200/1500)

When monitoring structures, the values of a structured PLC tag are displayed in the Inspector window → Diagnostics → Monitor value, with the following exception:

- Structures, whose elements have adjustable retentive properties, cannot be monitored.

In order to display the Monitor values for a structure, you must first of all activate the monitoring via the Context menu.

15.10.3. Monitor Block: Trigger Conditions / Call Environment

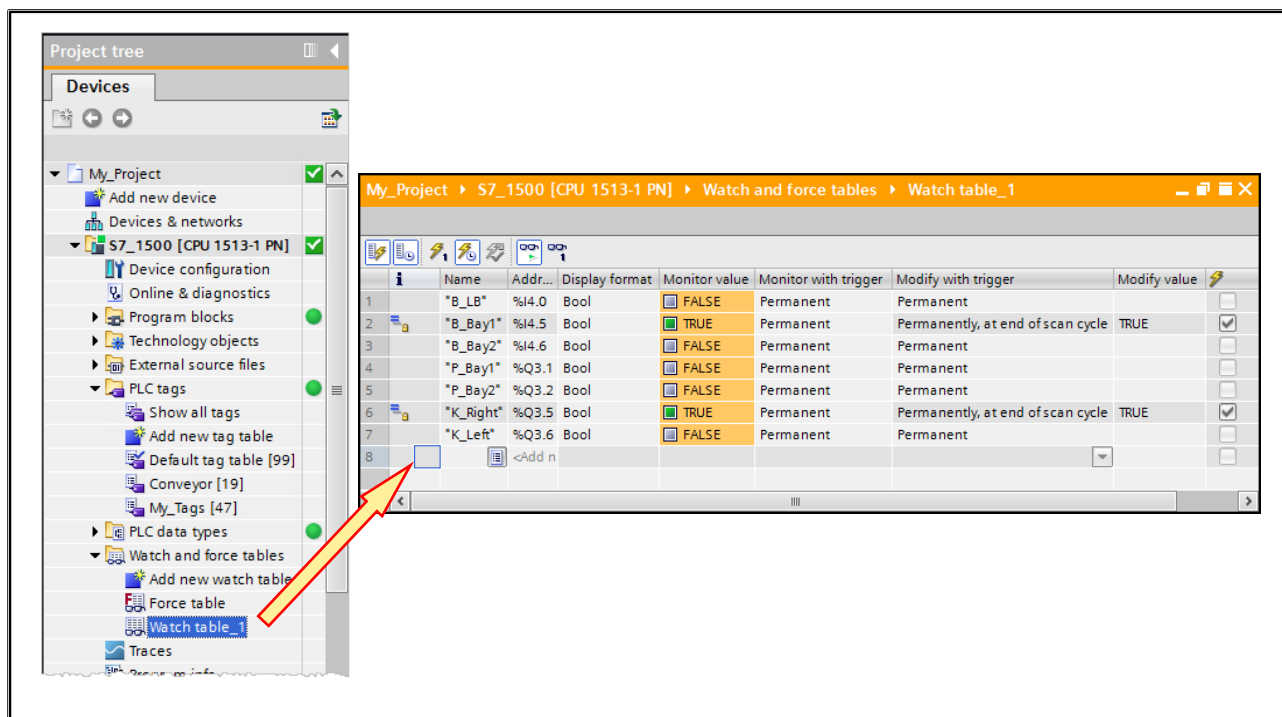


Function

The call conditions for blocks and for breakpoints can be defined. Thereby it is determined under which conditions the program status of a block is displayed or the program execution is interrupted at a breakpoint. The following conditions can be selected:

- Instance data block
The program status of a function block is only displayed when the function block is called with the selected instance data block.
- Call path
The program status of a block is only displayed when the block is called by a specific block or from a specific path.

15.11. Monitor / Modify Variables (Tags): Watch Tables



Area of Use

The "Monitor/Modify Variables (Tags)" test function is used to monitor and / or modify variables (tags) in any format you choose. For this, the desired variables are entered in a watch table. With the exception of block-local, temporary variables, you can monitor and/or modify all variables (tags) or operands.

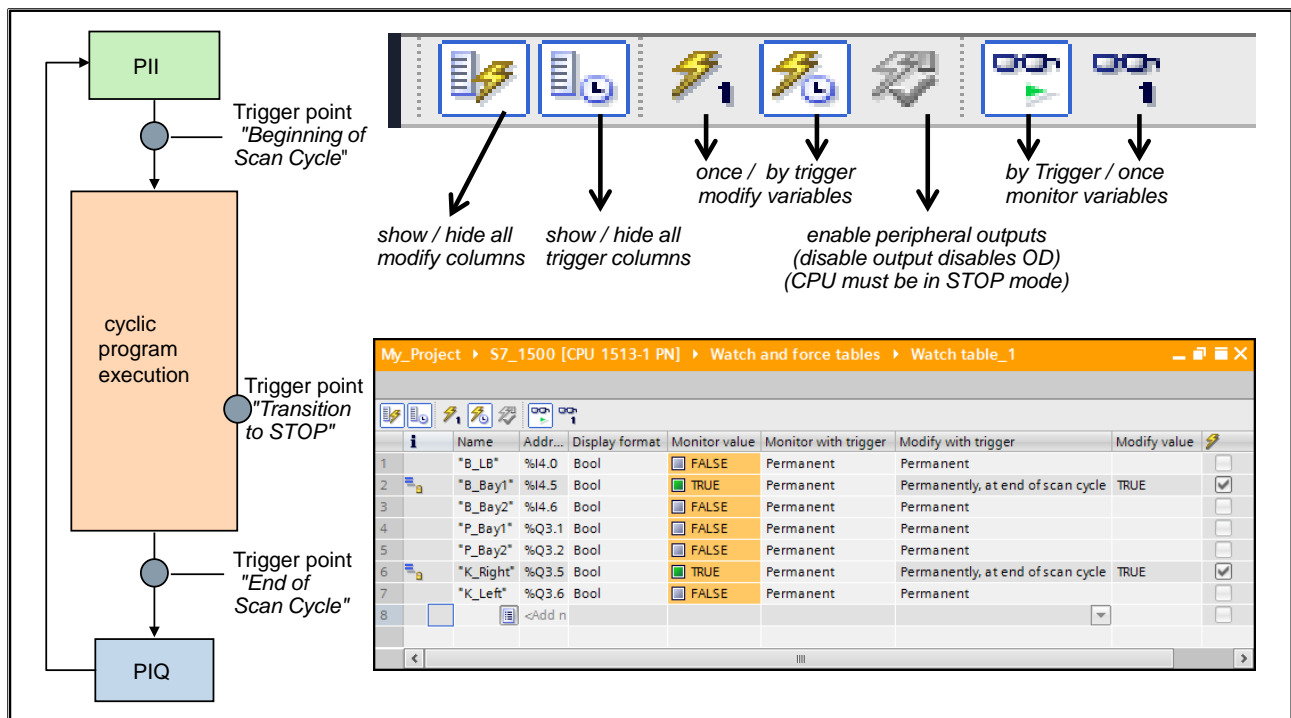
You can choose the columns displayed in the Watch table via the menu 'View'. The columns have the following meanings:

- Name: symbolic name of the variable (tag)
- Address: absolute address of the variable (tag).
- Symbol comment: comment on the variable (tag) displayed
- Display format: a data format you can choose per mouse click (such as binary or decimal), in which the contents of the variable (tag) is displayed
- Monitor value: variable (tag) value in the selected status format
- Modify value: value to be assigned to the variable (tag)

Watch Table

You can choose any name for the Watch table. Saved Watch tables can be reused to monitor and modify so that a renewed input of the variables to be monitored is no longer necessary.

15.11.1. Monitor / Modify Variables (Tags): Trigger Points



Trigger Points

Through the "Monitor with trigger or Modify with trigger" columns, you can define the trigger points for monitoring and modifying. The "Trigger Point for Monitoring" specifies when the values of the variables being monitored are to be updated on the screen. The "Trigger Point for Modifying" specifies when the given modify values are to be assigned to the variables being modified.

Trigger Condition

In the "Monitor with trigger" column you can specify whether the values are to be updated on the screen once only when the trigger point is reached or permanently (when the trigger point is reached).

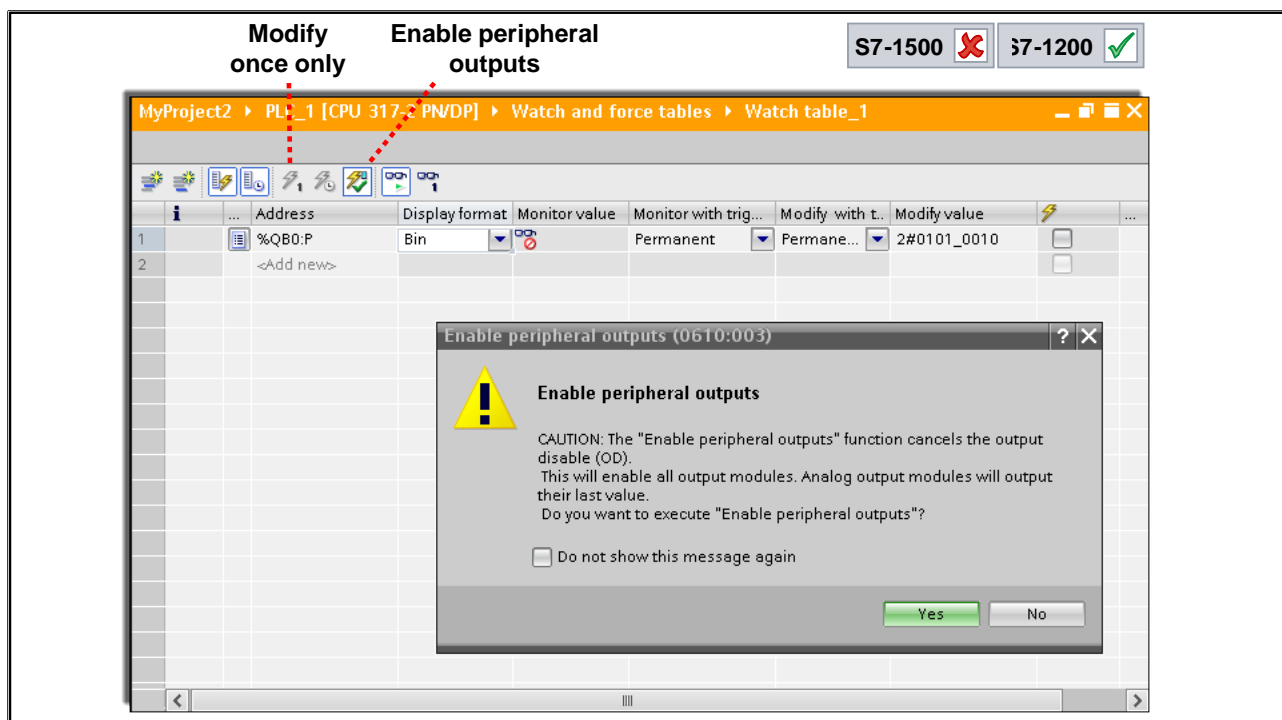
In the "Modify with trigger" column you can specify whether the given modify values are to be assigned to the variables being modified once only or permanently (every time the trigger point is reached).

Area of Use

The following tests, among others, can be implemented with the appropriate selection of trigger points and conditions:

- Wiring test of the inputs:
Monitor variables, Trigger point: Start of scan cycle, Trigger condition: Permanent
- Simulate input states (user specified, independent of process):
Modify variables, Trigger point: Start of scan cycle, Trigger condition: Permanent
- Differentiation between hardware / software errors (an actuator that should be activated in the process is not controlled)
Monitor variables, in order to monitor the relevant output, Trigger point: End of scan cycle, Trigger condition: Permanent
(output state = '1' > program logic OK > process error (hardware))
(output state = '0' > program logic error (such as double assignment))
- Control Outputs (independent of the program logic)
Modify variables, Trigger point: End of scan cycle, Trigger condition: Permanent

15.11.2. Enable Peripheral Outputs (in planning for S7-1500)



The Function "Enable Peripheral Outputs" (S71500 in Planning)

The "Enable Peripheral Outputs" function is used to check the functioning of the output modules, the wiring of the digital output modules or it can be used to continue to control actuators in the process even though the CPU finds itself in the STOP state because of an error that has occurred.

The "Enable Peripheral Outputs" function cancels the output disable of the peripheral outputs (PQ), which enables you to control the outputs in spite of the CPU's STOP state.

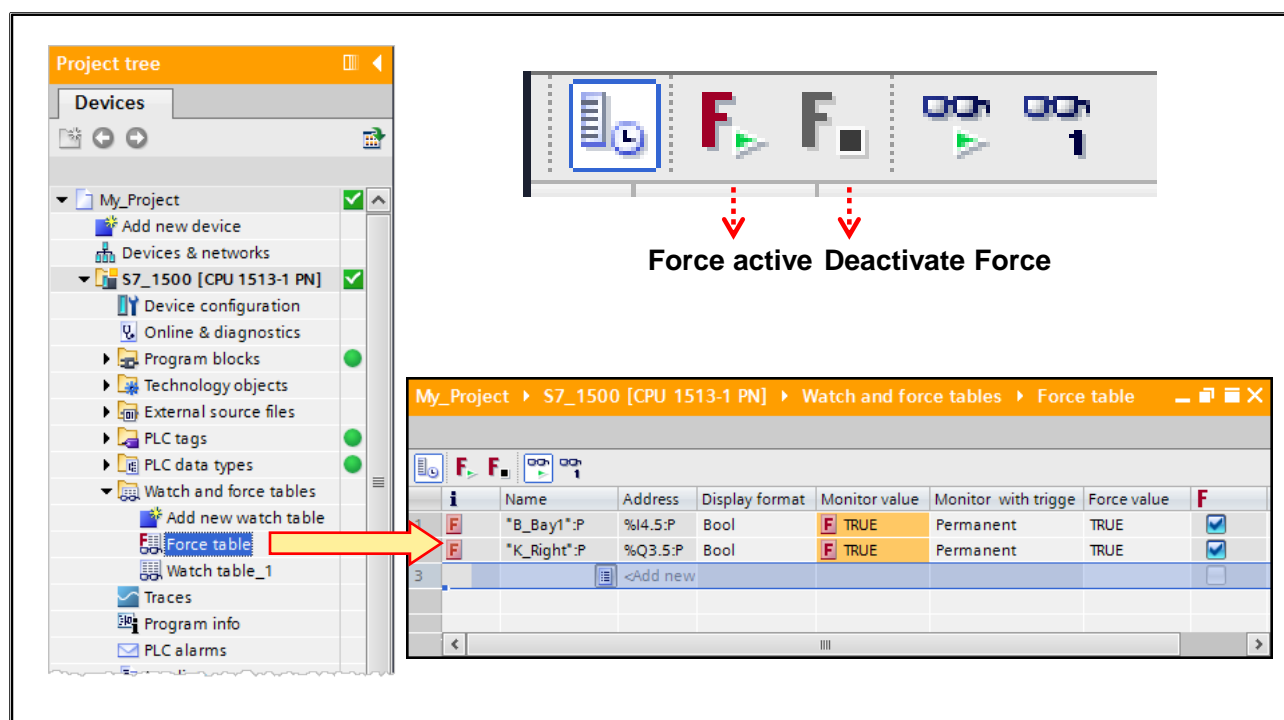
Conditions

- The CPU must be in STOP mode
- A Force task must not be active in the CPU
- The Watch table must be displayed in "extended mode", in other words, displayed with trigger columns
- The peripheral outputs to be enabled are to be specified byte by byte, word by word or double-word by double-word with the suffix :P (for peripheral)
- After the peripheral outputs have been enabled, the modify values can be activated via the "Modify once only" button (not via "Modify with trigger").

Note

When changing the CPU's operating status from STOP to RUN or STARTUP, Enable Peripheral Outputs is deactivated and a message pops up.

15.11.3. Force Variables (Tags)



Function and Area of Use

With Force, you can overwrite variables (tags) with any values you like, independent of the user program. Only one "Force Values" window can be open at a time for a CPU.

With the S7-300, you can only force the inputs and outputs in the process image; with the S7-400 you can also force memory bits and peripherals.

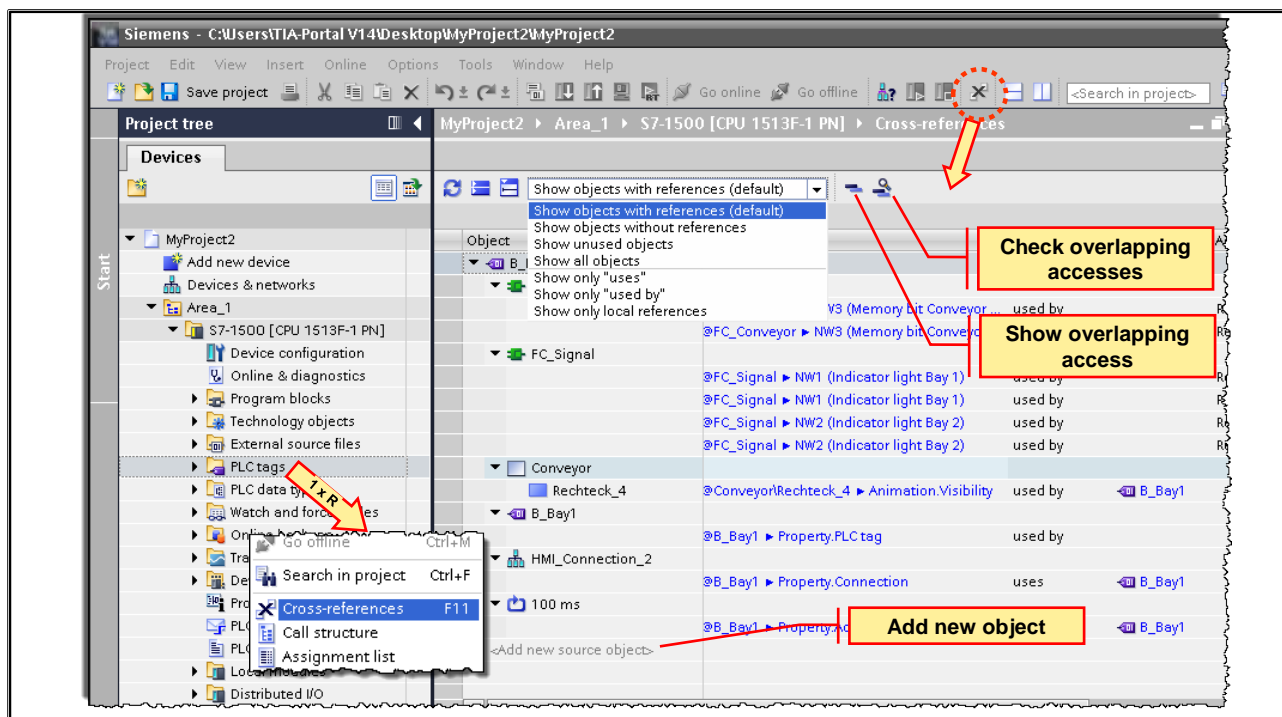
With the S7-1500/1200, the peripherals of the inputs and outputs can be forced.

Notes on Forcing

Before your start the "Force" function, you should make sure that no one else is carrying out this function at the same time on the same CPU.

A force task can only be canceled through an explicit "stop forcing" [deactivating the icon] (not via Edit → Undo!). Closing the Force table or exiting the application does not cancel the force task on the CPU.

15.12. Reference Data: Cross-references of Tags



Introduction

The cross-references list offers an overview of the use of operands and variables (tags) within the user program. From the cross-references list, you can jump directly to the point of use.

The cross-references list contains the following information:

- Which operand is used in which block with which instruction,
- Which tag is used in which HMI screen,
- Which block is called by which other block

As part of the project documentation, the cross-references supply a comprehensive overview of all operands, memory areas, blocks, variables (tags) and screens used.

Views

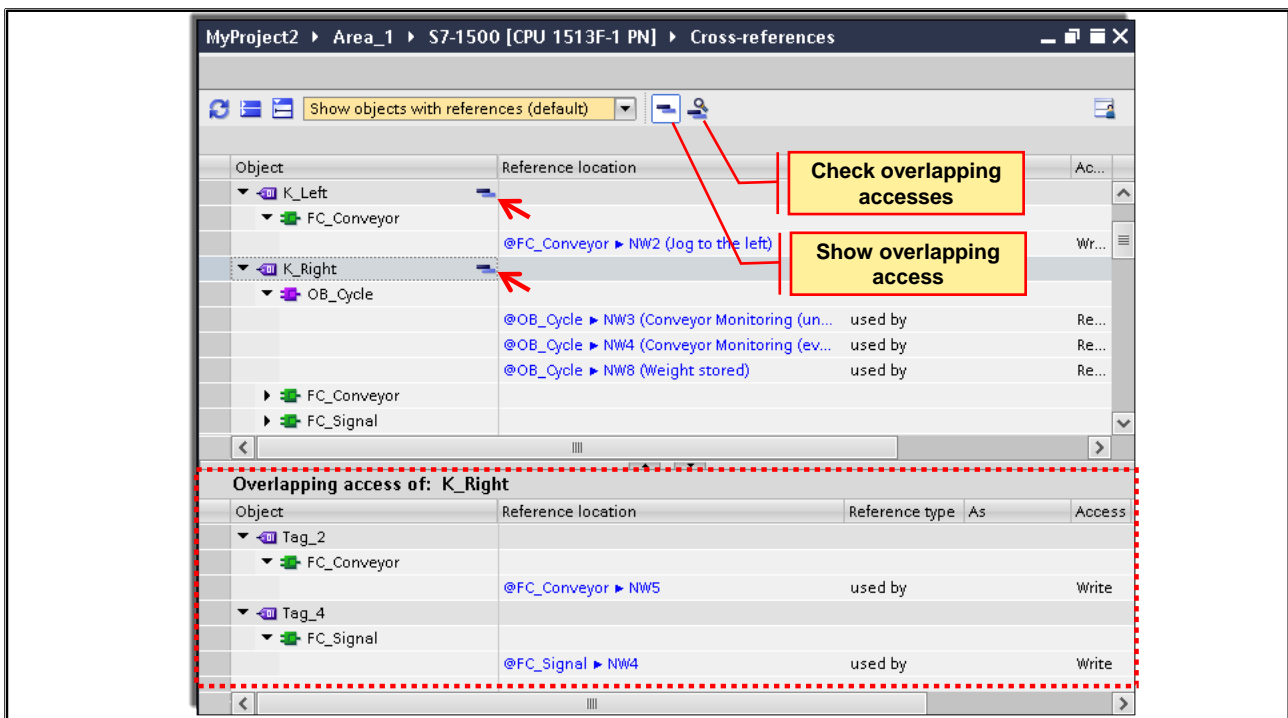
There are two views of the cross-references list which differentiate themselves by which objects are displayed in the first column:

- Used by:
Displays the referenced objects
Here, the Reference location where the object is used are displayed.
- Used:
Displays the referencing objects.
Here, the users of the object are displayed.
The associated tooltips give further information on the respective objects.

Show Unused

This is a list of tags which are declared in the PLC tag table but are not used in the S7 user program.

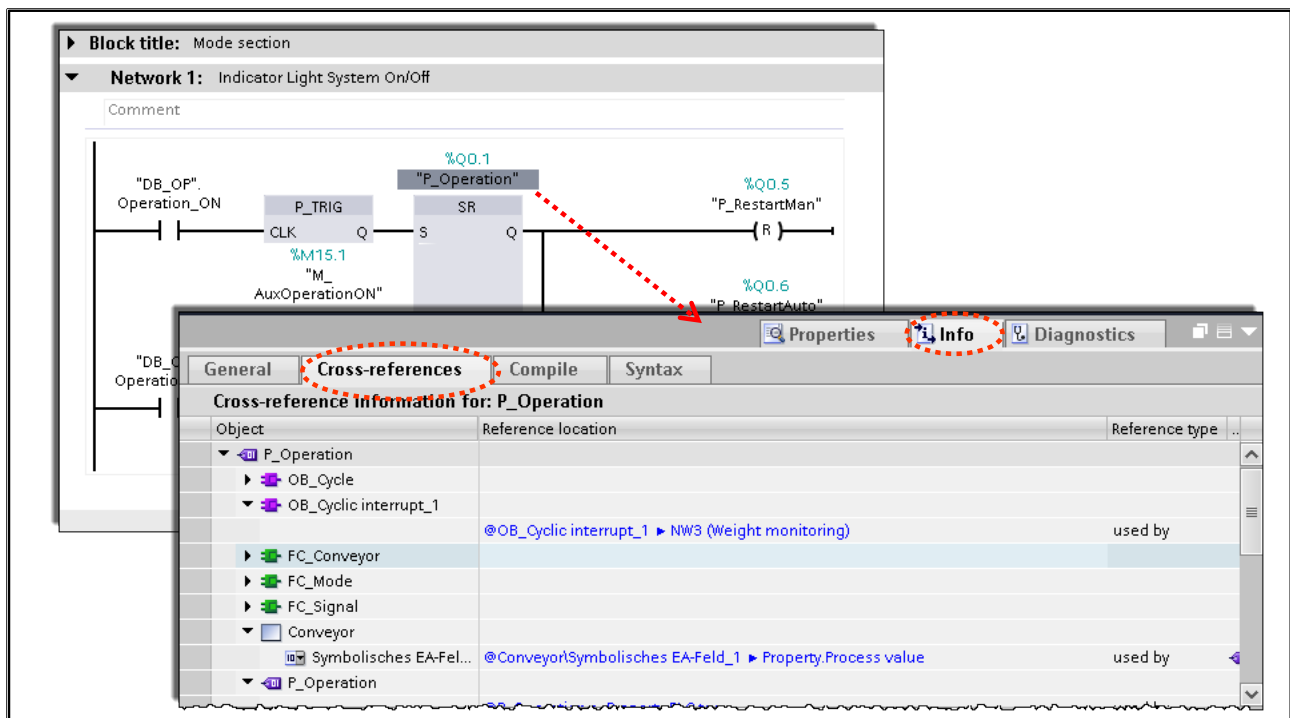
15.12.1. Reference Data: Cross-references / Show Overlapping Accesses



With the help of the "Check overlapping accesses" button, you can check whether overlapping accesses exist for one of the variables (tags).

If this is the case, they can be displayed in a separate table with the help of the "Show overlapping accesses" button.

15.12.2. Reference Data: Cross-references of a Variable (Tag) in the Block Editor



Introduction

In the Inspector window, the cross-reference information for a selected object is displayed in the tabs "Info > Cross-references". In this tab, you will see at which Reference locations and from which other objects every selected object is used.

In the Inspector window, even those blocks that only exist online are displayed in the Cross-references.

Structure

The cross-reference information is displayed in tabular form in the Inspector window. Each column contains specific detailed information on the selected object and its use.

15.12.3. Go To...

The screenshot displays a ladder logic network titled "Network 1: Indicator Light System On/Off". The network contains the following elements:

- Inputs: "DB_OP", Operation_ON (normally open contact) and "DB_OP", Operation_OFF (normally closed contact).
- Logic: A P_TRIG (Pulse Trigger) block with input CLK and output Q. The output Q is connected to the S (Set) input of an SR (Set-Reset) block.
- Outputs: The Q output of the SR block is connected to the R (Reset) input of a coil labeled "P_RestartMan".
- Interlocks: A normally open contact labeled "%M15.1", "M_AuxOperationON" is connected in parallel with the "DB_OP", Operation_ON contact.
- Registers: "%Q0.1" is associated with the SR block, and "%Q0.5" is associated with the "P_RestartMan" coil.

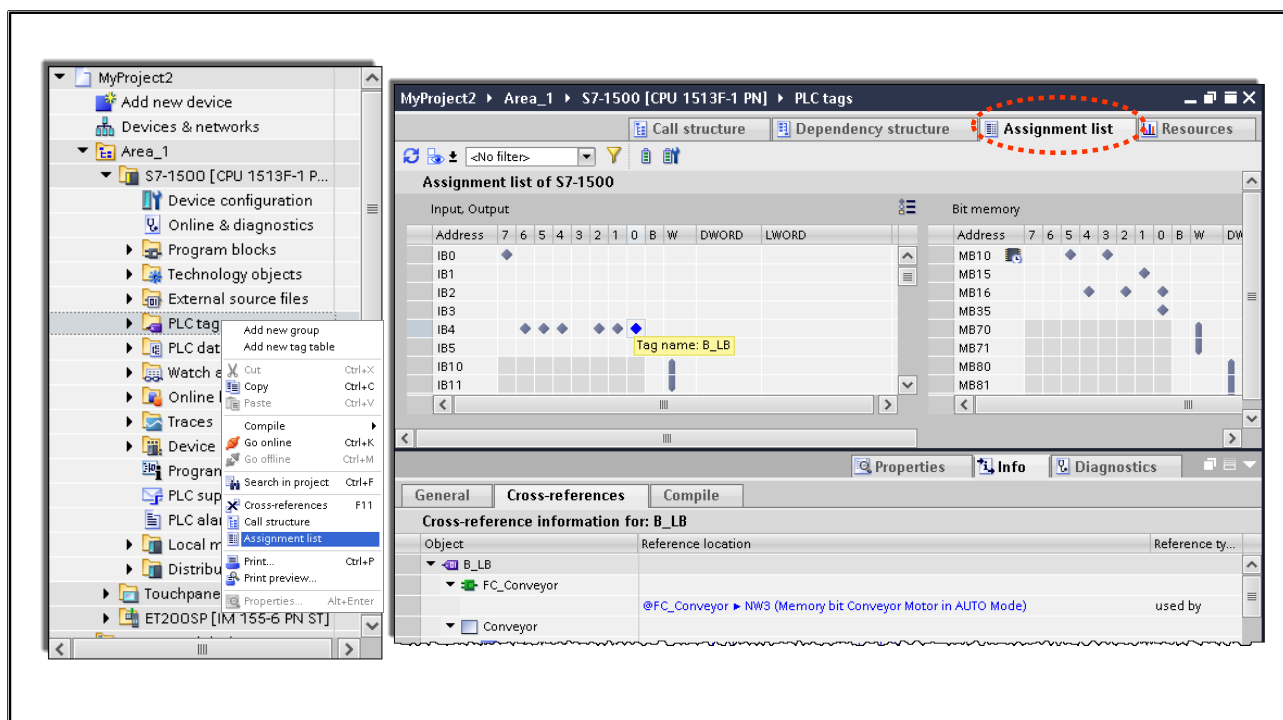
A context menu is open over the "P_RestartMan" coil, showing the "Go to" option selected. The menu includes the following items:

- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Delete (Del)
- Go to** (Shift+F11)
 - Cross-reference information (Shift+F11)
 - Insert network (Ctrl+R)
 - Insert STL network
- Read/write access... (Alt+F8)
- Back to read/write access (Alt+F9)
- Network/line...
 - Next point of use** (Ctrl+Shift+G)
 - Previous point of use (Ctrl+Shift+F)
 - Next error
 - Previous error
 - Definition (Ctrl+Shift+D)
 - Device view

Annotations with red boxes and arrows point to specific features:

- Point of use in the current block**: Points to the "P_RestartMan" coil.
- 1 x R**: A yellow arrow points to the "R" input of the coil.
- Syntax error**: Points to the "Go to" menu item.
- Declaration of operand**: Points to the "%Q0.5" register declaration.
- Input Output in the Device view**: Points to the "Device view" option in the context menu.

15.12.4. Reference data: Assignment I/Q/M/T/C



Assignment I/Q/M/T/C

The assignment list for I/Q/M/T/C is opened via "Right-click on the relevant device -> Assignment list" or via the menu "Tools > Assignment list".

This assignment list gives you an overview of which bit is used from which byte of the memory areas input (I), output (Q) and memory bit (M) and which SIMATIC timers and counters are used. The type of use (reading or writing) is not displayed.

The memory areas inputs (I), outputs (Q) and memory bits (M) are displayed byte-by-byte in lines.

- The bits identified with a small diamond, that is, binary operands (in the picture, for example, I 4.0 or M 16.4) are used explicitly in the program.
- The fields of the individual bits which have a gray background identify byte, word, double-word or long word operands that are used in the user program. The operand dimension (byte, word, double-word or long word) comes from the vertical line in one of the columns "B" (Byte), "W" (Word), "DWORD" (Double word) and "LWORD" (Long word).
- Bits that are marked with both a diamond and a gray background field are used explicitly as a binary operand in the user program and are used via a byte, word, double-word or long word operand.

15.12.5. Reference Data: Call Structure

Call structure of S7-1500

Call structure	Address	Details	Local data (in pa...)
1 OB_Cycle	OB1		0
2 DB_Memory	1	@OB_Cycle ► NW8 (Weight stored)	0
3 DB_Memory	DB1	@OB_Cycle ► NW8 (Weight stored)	0
11 DB_OP	DB99	@OB_Cycle ► NW8 (Weight stored)	0
12 DB_OP	DB99	@OB_Cycle ► NW8 (Weight stored)	0
13 DB_OP	DB99	@OB_Cycle ► NW8 (Weight stored)	0
14 DB_OP	DB99	@OB_Cycle ► NW8 (Weight stored)	0
15 DB_Weights_opt	DB36	@OB_Cycle ► NW8 (Weight stored)	0
16 FB_Fault, iDB_FB_Fault_even	FB17, DB4	@OB_Cycle ► NW4 (Conveyor Monitoring (even))	0
17 FB_Fault, iDB_FB_Fault_uneven	FB17, DB2	@OB_Cycle ► NW3 (Conveyor Monitoring (uneve...)	0
18 FC_Conveyor	FC16	@OB_Cycle ► NW2 (Control the Conveyor Motor)	0
19 FC_Mode	FC15	@OB_Cycle ► NW1 (Operating Modes)	0
20 FC_Signal	FC14	@OB_Cycle ► NW7 (Signal)	0
21 FC_Weight_STL	FC37	@OB_Cycle ► NW8 (Weight stored)	8
22 OB_Cyclic interrupt_1	OB35		0

Call Structure

The call structure is opened via "Right-click on the relevant device -> Call structure" or via the menu "Tools > Call structure" and describes the call hierarchy of the blocks within an S7 program.

It gives an overview of:

- The blocks used
- Jumps to the points of use of the blocks
- Dependencies between the blocks
- Local data requirements of the blocks
- Status of the blocks

15.12.6. Reference Data: Dependency Structure

The screenshot shows the 'Dependency structure' window for S7-1500. The window title is 'MyProject2 > Area_1 > S7-1500 [CPU 1513F-1 PN] > PLC tags'. The 'Dependency structure' tab is selected. The window displays a list of blocks and their dependencies. A red dashed circle highlights the 'Dependency structure' tab. A red arrow points from the 'UDT_WeightStore' block to a yellow box labeled 'Consistency check'. Another red arrow points from the '@OB_Cycle' block to a yellow box labeled 'Interface declaration (to UDT_WeightStore)'.

Dependency structure	Address	Details
1 UDT_WeightStore		
2 DB_Weights, #WeightStore		Block interface
3 DB_Weights, #WeightStore2		Block interface
4 DB_Weights, #WeightStore3		Block interface
5 DB_Weights_opt, #WeightStore1		Block interface
6 OB_Cycle	OB1	@OB_Cycle ▶ NW8 (Weight stored)
7 DB_Weights_opt, #WeightStore2		Block interface
8 OB_Cycle	OB1	@OB_Cycle ▶ NW8 (Weight stored)
9 DB_Weights_opt, #WeightStore3		Block interface
10 FC_Weight, #WeightStore		Block interface
11 FC_Weight_LAD, #WeightStore		Block interface
12 FC_Weight_STL, #WeightStore		Block interface
13 DB_Memory (global DB)	DB1	
14 iDB_FB_Fault_uneven (instance DB of ...)	DB2	
15 iDB_FB_Fault (instance DB of FB_Fault)	DB3	
16 iDB_FB_Fault_even (instance DB of FB_...)	DB4	

Dependency Structure

The display of the dependency structure is opened via the menu "Tools > Dependency structure" and with it you get a list of blocks used in the user program. In the first level (to the very left) is the respective block and indented underneath it are the blocks which call this block or use it.

The dependency structure also shows the status of the individual blocks through the use of symbols. Objects which cause a time stamp conflict and which can lead to an inconsistency in the program are identified with different symbols.

The dependency structure represents an extension of the cross-references list for objects.

15.13. Resources

Objects	Load memory	Code work-memory	Data work-memory	Retain memory	Motion Control
1	2 %	1 %	0 %	0 %	0 %
2					
3	Total: 4 MB	460800 bytes	1572864 bytes	90784 bytes	
4	Used: 97206 bytes	2557 bytes	1178 bytes	190 bytes	
5	Details				
6	OB	23169 bytes	932 bytes		
7	FC	49456 bytes	1490 bytes		
8	FC_Signal [FC14]	7739 bytes	259 bytes		
9	FC_Mode [FC15]	3878 bytes	137 bytes		
10	FC_Conveyor [FC16]	7452 bytes	241 bytes		
11	FC_Fault [FC17]	5490 bytes	128 bytes		
12	FC_Weight [FC35]	9011 bytes	374 bytes		
13	FC_Weight_LAD [FC36]	9923 bytes	253 bytes		
14	FC_Weight_STL [FC37]	5963 bytes	98 bytes		
15	FB	5804 bytes	135 bytes		
16	DB	13144 bytes		1178 bytes	158 bytes
17	Objects for Motion Technology	-		-	0 bytes
18	Data types	1370 bytes			
19	PLC tags	4263 bytes			32 bytes

The Resources is opened via the menu "Tools > Resources" and shows you which (how much) memory area is used by which objects in the CPU.

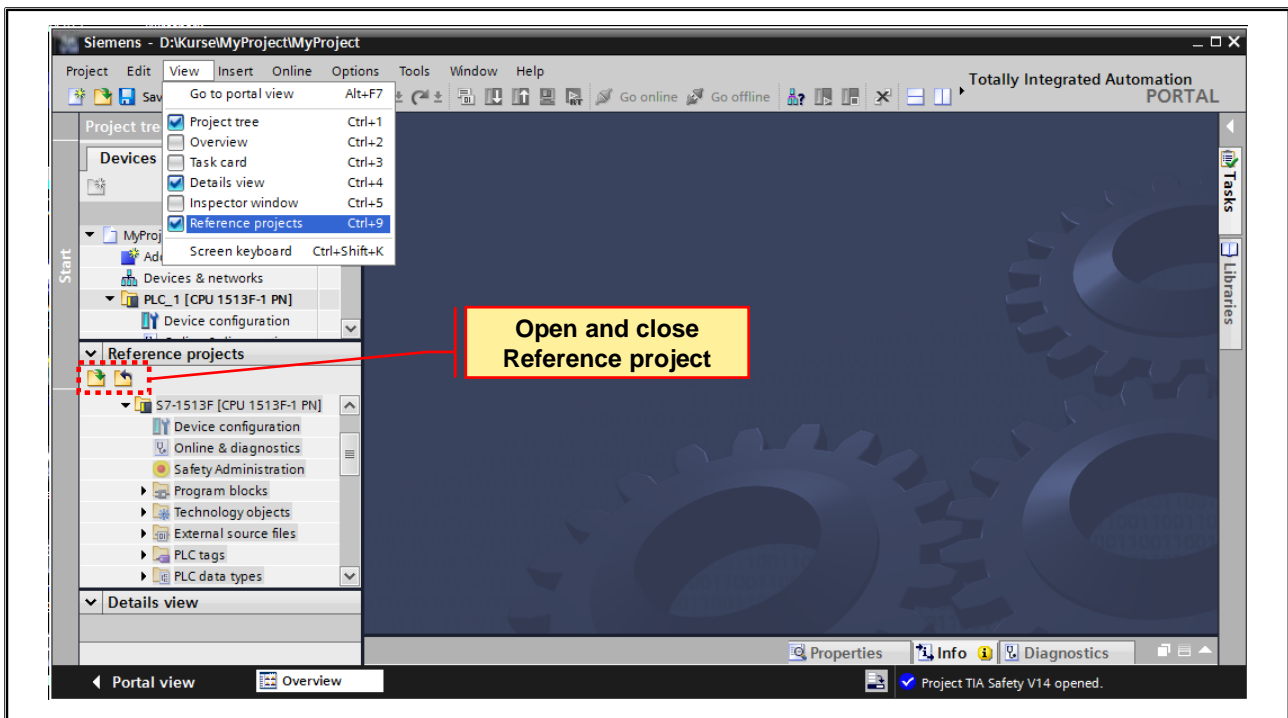
Note

Display of the 'Used' Load memory in the CPU

Please note that the sum of the used load memory cannot be exactly determined if not all blocks have been compiled.

In this case, a ">" placed in front of the sum indicates that the value for the used memory area could be larger than displayed since blocks that are not compiled are not taken into account for the total formation.

15.14. Reference Projects

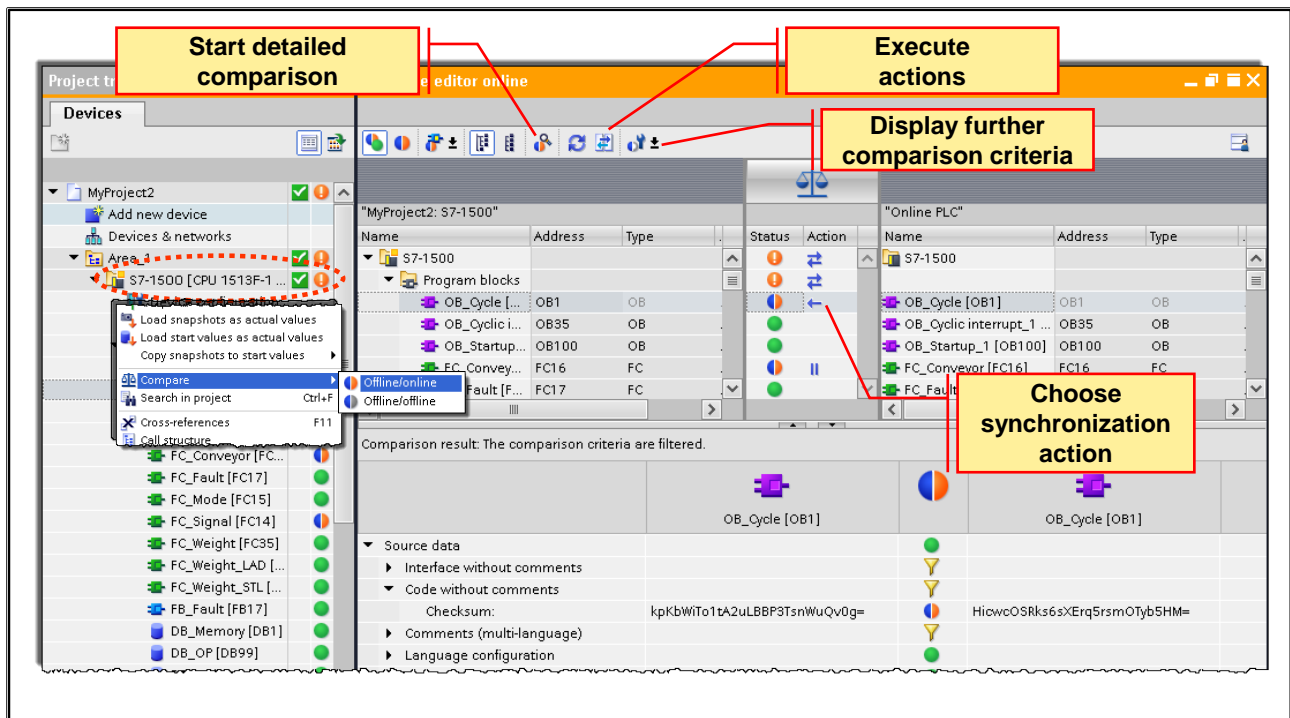


Reference Projects

In addition to the user project, a write-protected Reference Project (see picture, all project elements have a grey background) can be opened via the menu "View". Several elements (for example, Program blocks) can be opened in the editor but they cannot be changed.

Similar to the Libraries, all project elements can be copied from the Reference Project into the user project.

15.15. Compare (1) - Offline / Online



Types of Comparison

In principle, there are two different types of comparison:

Online/Offline comparison:

- The objects in the project are compared with the objects of the relevant device. For this, an online connection to the device is necessary.

Offline/Offline comparison:

- Either the objects of two devices within a project or from different projects are compared.

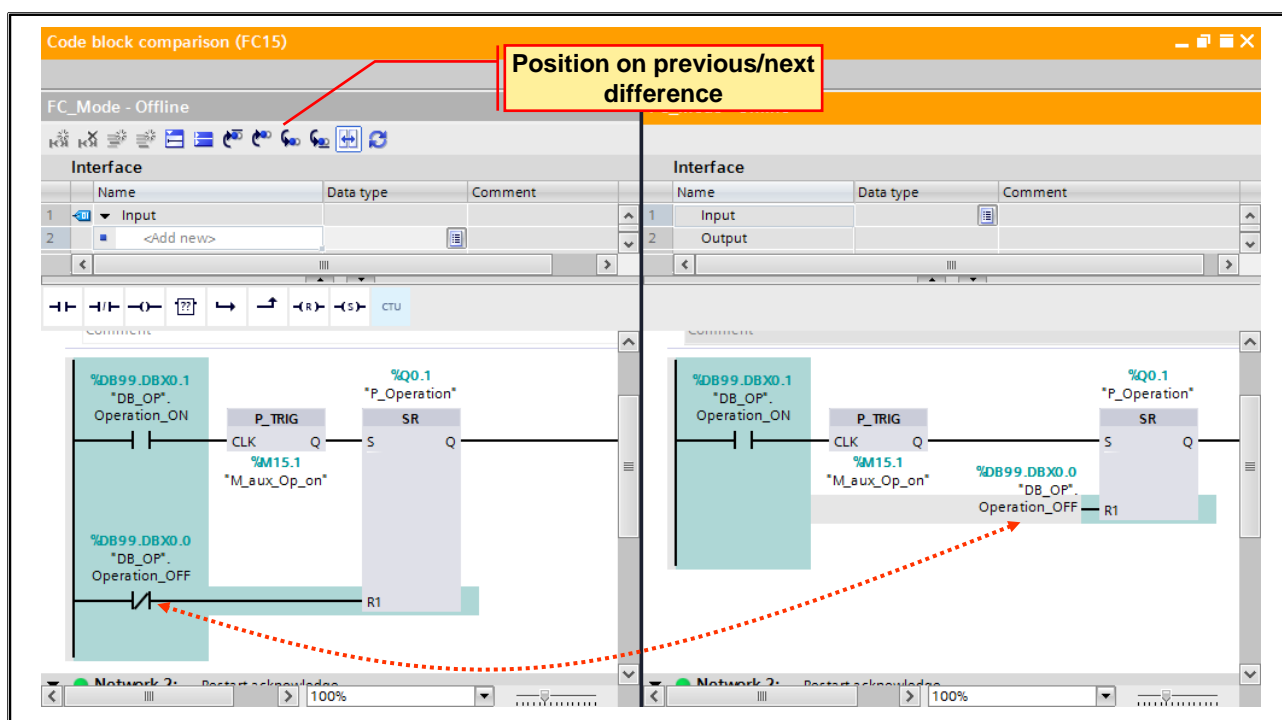
Symbols of the Result Display

The result of the comparison is presented by means of symbols.

The following table shows the symbols for the comparison results of an Online/Offline comparison:

Symbol	Meaning
!	Folder contains objects whose online and offline versions are different
?	Comparison result is unknown
●	Online and offline versions of the object are identical
●	Online and offline versions of the object are different
●	Object only exists offline
●	Object only exists online

15.15.1. Compare (2) – Block Detailed Comparison

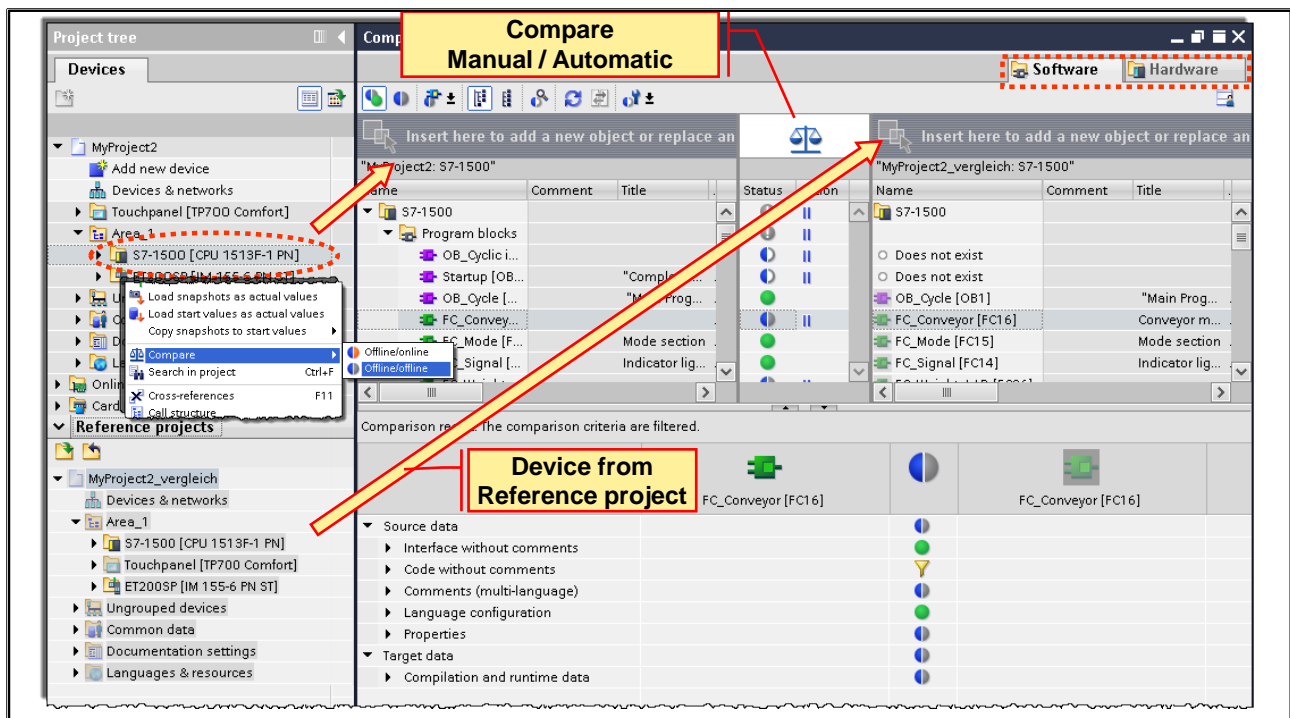


Detailed Comparison

Through the detailed comparison you can identify exactly those locations that are different in the online and offline version of a block. The following identifiers are used:

- Lines in which there are differences are highlighted in grey.
- Different operands and operations are highlighted in green.
- When the number of networks is different, pseudo networks are inserted so that a synchronized representation of identical networks is possible. These pseudo networks are highlighted in grey and contain the text "No corresponding network was found" in the title-bar of the network. Pseudo networks cannot be processed.
- If the sequence of the networks is mixed up, pseudo networks are inserted at the appropriate locations. These pseudo networks are highlighted in grey and contain the text "The networks are not synchronized" in the title-bar of the network. The pseudo network also contains a link "Go to network <No>", through which you can navigate to the associated network.

15.15.2. Compare (3) - Software Offline / Offline



Offline / Offline Software Comparison:

- Objects of two devices within a project,
- Blocks of two devices within a project,
- Blocks in one device,
- Objects from different projects,
- Blocks from different projects,

For this, you can switch between

automatic  and manual  comparison using a mouse click.

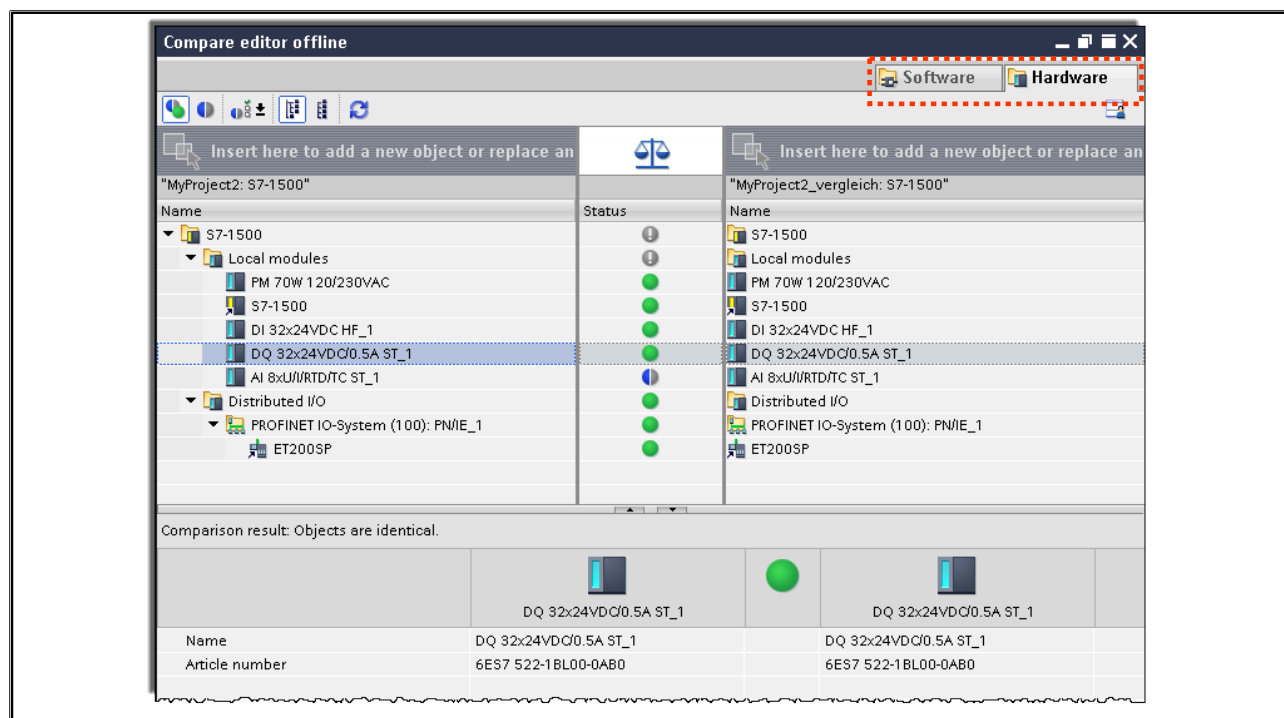
Automatic Comparison:

Blocks and objects of the same type and the same name are compared

Manual Comparison:

You can select which blocks are to be compared. That way, it is possible to compare all blocks.

15.15.3. Compare (4) - Hardware Offline / Offline



Offline / Offline Hardware Comparison:

In addition, it is possible to compare the hardware between two devices or modules in one device.

15.16. Exercise 4: Testing the Motor Jog

Split editor (working area) horizontally

Network 1: Memory bit Jog conveyor RIGHT

Logic elements: "DB_OP" JogRight, "DB_OP" JogLeft, "P_Operation", "TOF-Timer-L", "K_Right"

	Name	Address	Display format	Monitor value	Modify value	Comment
1	"P_Operation"	%Q0.1	Bool	FALSE		
2	"DB_OP" JogRight		Bool	TRUE		
3	"DB_OP" JogLeft		Bool	FALSE		
4	"K_Right"	%Q4.5	Bool	FALSE		
5	"K_Left"	%Q4.6	Bool	FALSE		

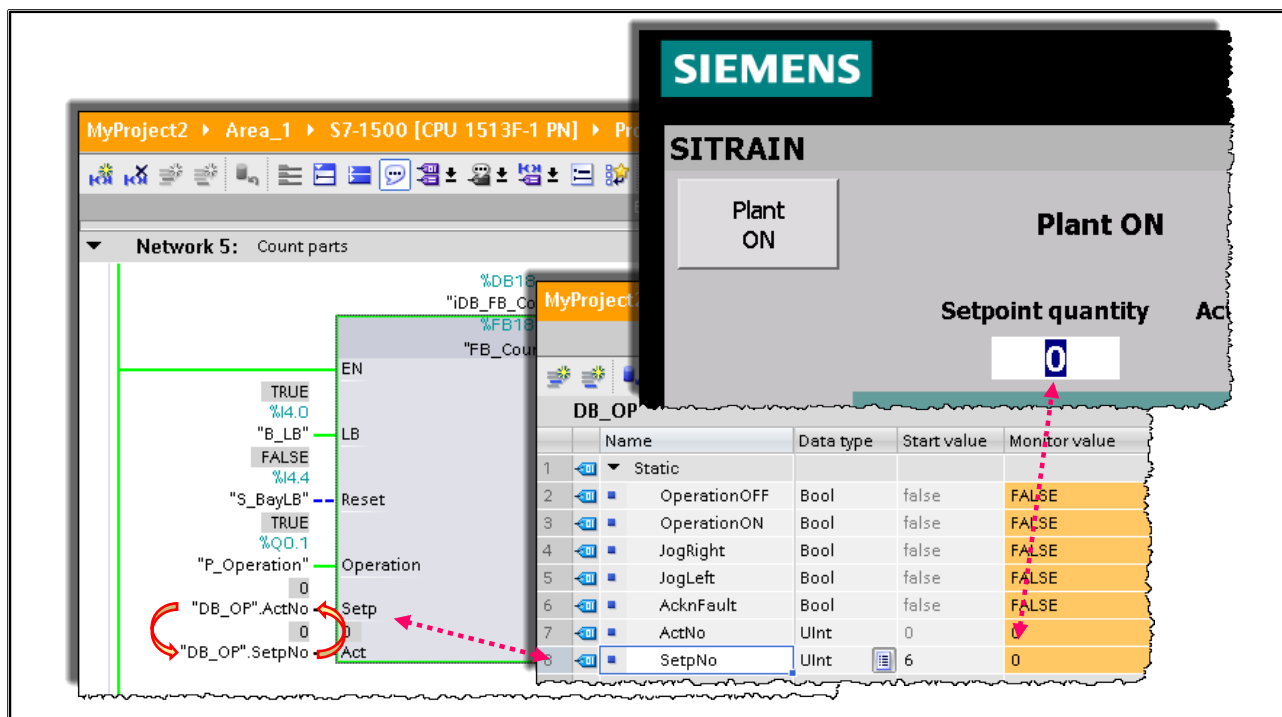
Task

The function "Jog conveyor motor" does not work. The combined use of the PG functions "Monitor block" and "Watch table" (monitor tags [variables]) indicates that there must be a double assignment at output "K_Right" (Q4.5). The task now is to find all instructions in the entire user program that write-access this output.

What to Do:

1. Carry out a CPU restart.
2. On the touchpanel, switch off the "Plant".
3. Open the "FC_Conveyor" block and activate the "Monitor" test function.
4. In the Project tree, under "Watch and force tables" create a new Watch table and in it monitor the output "K_Right" (Q4.5).
5. Display the Blocks Editor with the opened "FC_Conveyor" and the Watch table one below the other by splitting the working area (see picture).
6. Interpret the different status displays of the two test functions.
7. Localize the double assignment at output "K_Right" (Q4.5) with the help of the reference data, correct the error and save the change.
8. Download all modified blocks into the CPU and check the how the program functions.

15.17. Exercise 5: Entering the Setpoint Quantity



Task

In the search for this logical error, you are to use the test function Cross-references.

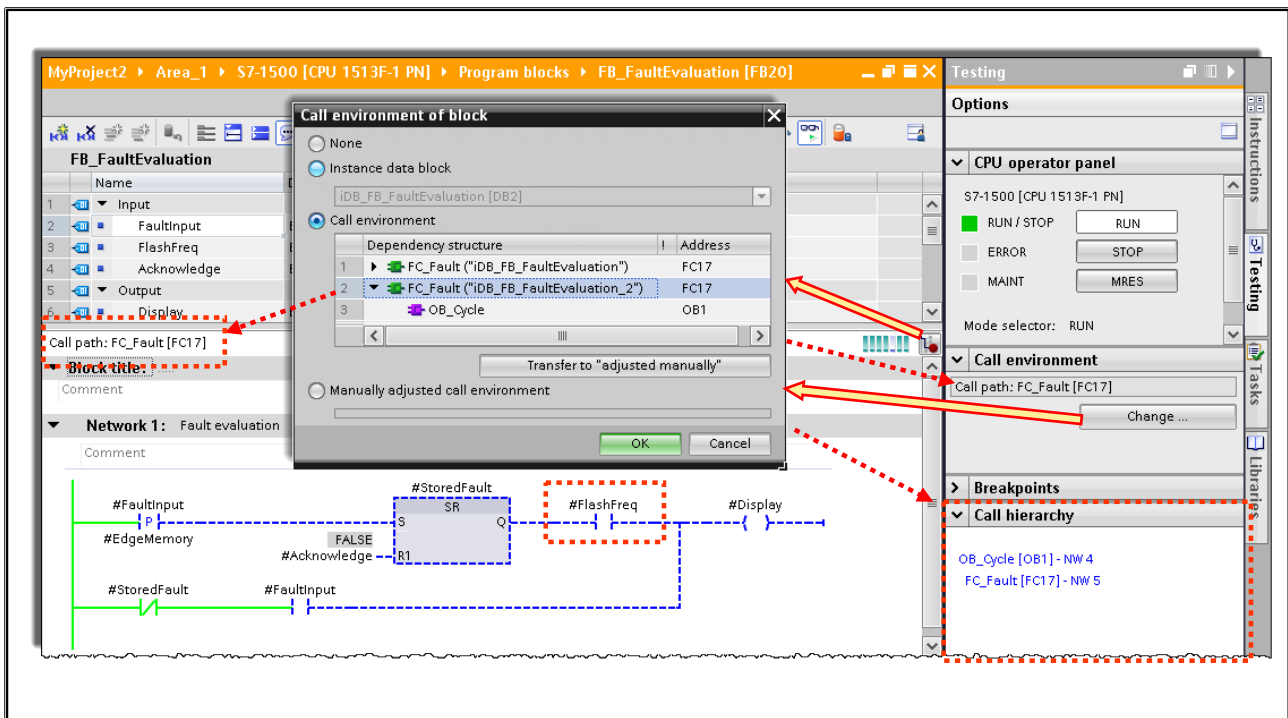
Function Test

Check whether you can enter/change the setpoint quantity of the parts to be transported. Should this not work, correct the error.

What to Do:

1. Try to change the setpoint quantity on the touchpanel.
2. In the project, in the Touchpanel device, open the "Conveyor" screen, highlight the input field "Setpoint quantity" and display the Cross-references in the Inspector window.
Note: via the menu item "Cross-reference information" in the context menu of the input field, you can open the Cross-references in the Inspector window.
3. Switch to the connected PLC tags by clicking the link in the "Address" column.
4. Find a "write" or "read + write" access in the "Access" column.
5. Now you jump to where the tag is used in the PLC by clicking the relevant link in the "Reference location" column.
6. You will see that the tag "DB_OP".SetpNo was assigned as actual parameter to the formal parameter "Act" for the call of "FB_Count" and the tag "DB_OP".ActNo to the formal parameter "Setp".
7. Correct the parameterization of "FB_Count".
8. Save your project, download the modified program and once again test the entry of the setpoint quantity.

15.18. Exercise 6: Testing the Evaluation of Fault 3



Task

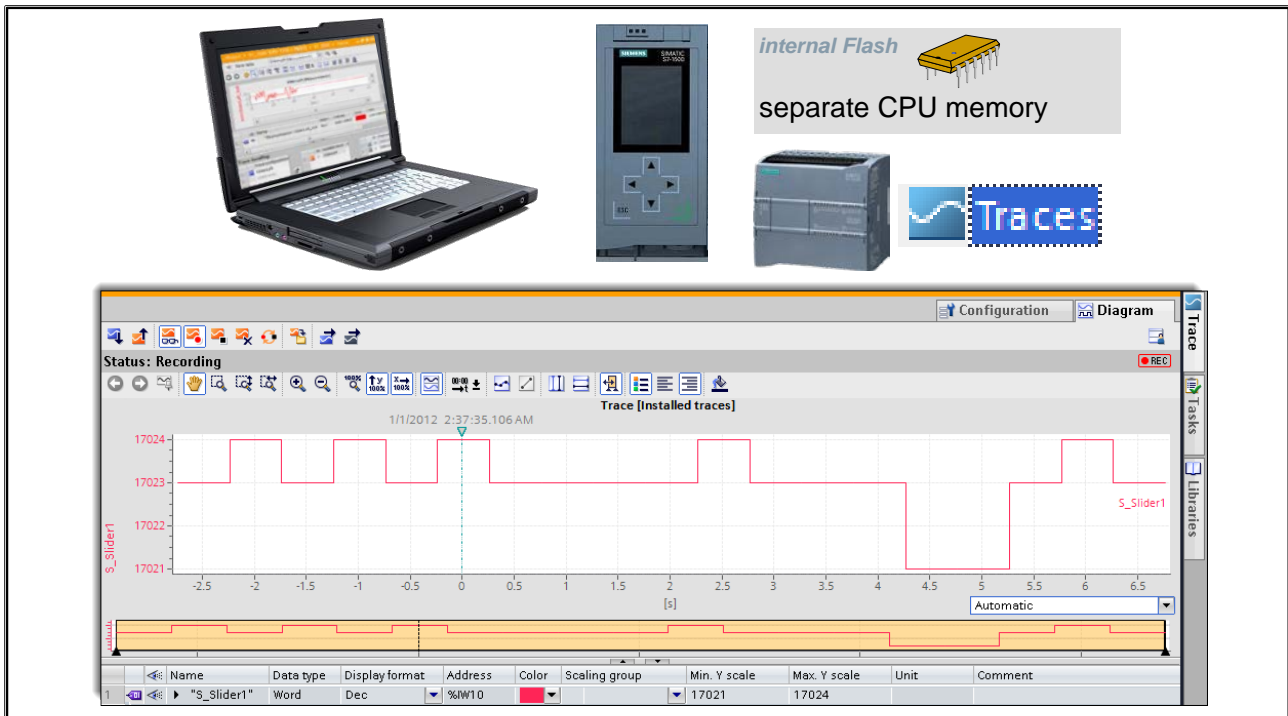
In the search for the last logical error, you are to use the test function 'Monitor block with trigger' on a certain block call.

Check whether the occurrence of *Faults 2 and 3* (switches I 0.5 and I 0.6 on the simulator) is displayed on the simulator LEDs "P_Fault2" (Q0.3) and "P_Fault3" (Q0.4) with a flashing light, and whether, after acknowledging with the simulator pushbutton "S_Acknowledge" (I 0.7), the flashing light of the individual faults switches to a constant light.

What to Do

1. To troubleshoot why no flashing light is displayed after *Fault 3* occurs, first of all monitor "FB_FaultEvaluation" with the test function "Monitor block".
You will see that the static variable #StoredFault is controlled when Fault 2 exists, however, not for Fault 3. The cause of this lies in the fact that you are monitoring the execution of "FB_FaultEvaluation" for the evaluation of Fault 2 or the first FB call in "FC_Fault".
2. Change the call environment as shown in the picture in such a way that you specifically monitor the execution of "FB_FaultEvaluation" for the evaluation of Fault 3.
3. In monitoring the second block call you will see that the status of the input parameter #FlashFreq does not change to 2Hz flashing frequency as expected.
4. Since the parameter in the block is not overwritten, the input parameter must be assigned incorrectly or not assigned at all for the call of the function block.
5. Correct the error and once again monitor the evaluation of Fault 3.
6. Download all modified blocks into the CPU and check the function.
7. Save your project.

15.19. TRACE Analyzer Function



"Trace" Analyzer Function

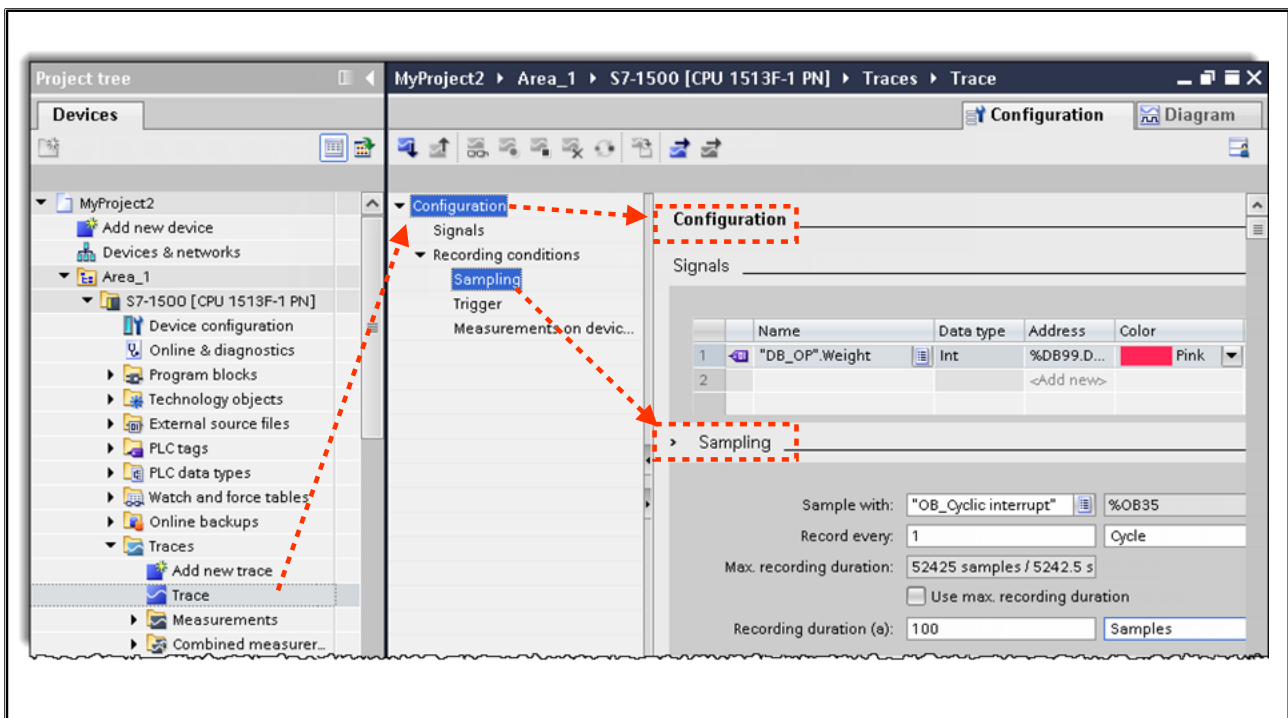
The value-over-time of one or several CPU tags (max. 16) can be stored in a TRACE. In STEP 7, a TRACE recording can be presented graphically.

The number of Traces depends on the CPU.

Depending on the CPU, internal TRACE memories with 512 Kbyte each are available.

- S7-1200 2x TRACE (FW \geq V4.0)
- Up to S7-1517 4x TRACE, S7-1518 8x TRACE
- A maximum of 16 Trace signals or CPU tags (variables) can be recorded per Trace.

15.19.1. Configuring a TRACE - Signals and Sampling



Trace – Signals

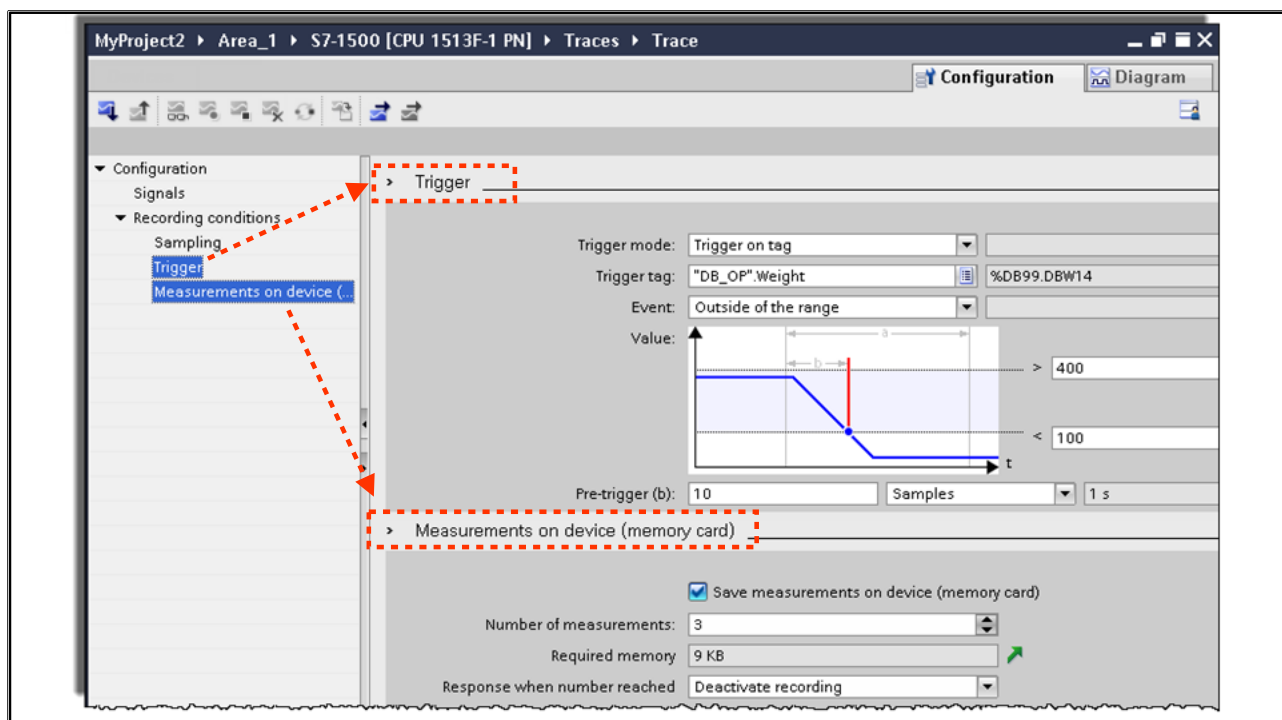
All global PLC tags (variables) of an elementary data type can be recorded.

Trace – Sampling and – Recording Duration

Here, you define how often or in which intervals the Trace signals are to be recorded. From these sampling intervals and the data type or the dimension of the Trace signals you get the maximum duration of a Trace recording since the memory space available for the recording is limited.

- Maximum memory space per Trace: 512 Kbytes – 30 bytes (for internal management) = 524,258 bytes
Each sample is saved with a time stamp (8 bytes). This results in a...
- number of bytes per sample = 8 bytes + number of bytes of a sample
(for Boolean Trace signals, the number of bytes of a measured value is 1 byte)
- Example: Trace with 1x INT variable and a sampling interval of 100ms
 - Trace signal of the data type INT -> 8+2 bytes/sample -> 52,425 possible samples
 - Sampling interval = 100ms -> 10 samples per second
 - -> maximum recording duration: 52425 samples / 10 samples per second = 5242 seconds

15.19.2. Configuring a TRACE – Trigger and Saving Measurement on Device



Triggering the Trace Recording

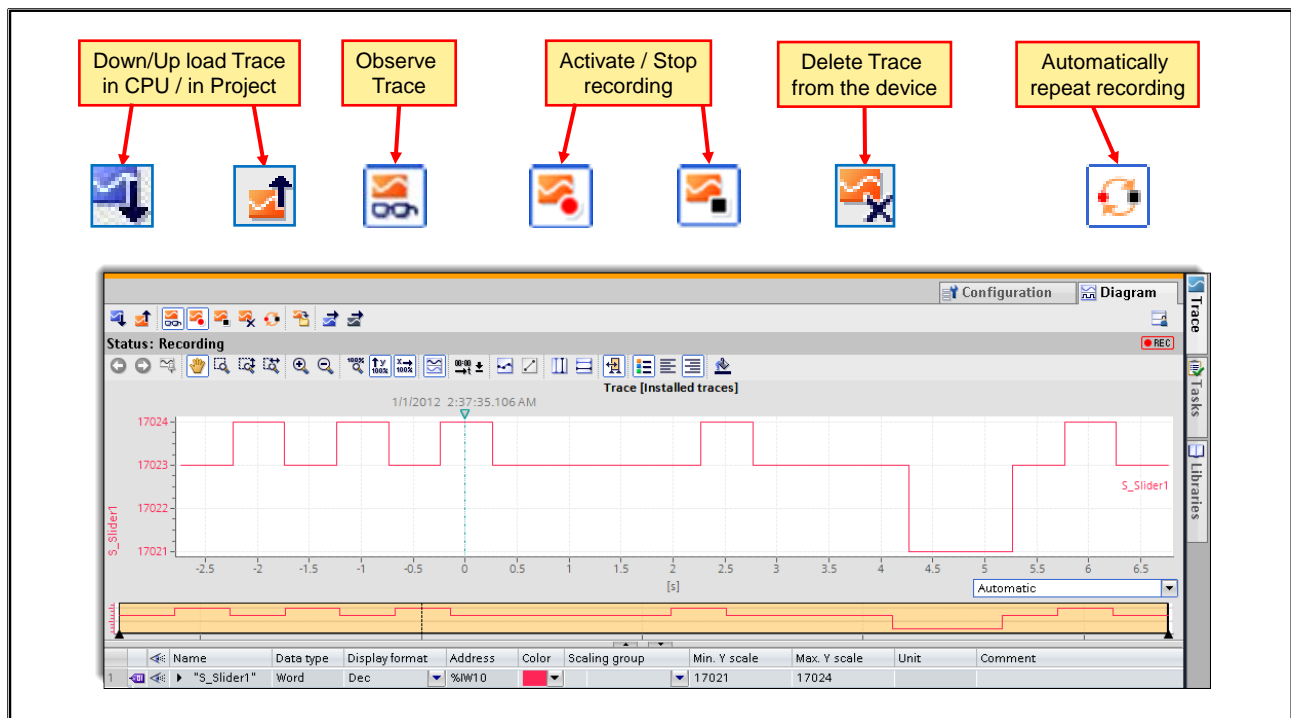
Activating the Trace recording starts the measurement and recording of the Trace signals, however, not the permanent saving of values since these are merely only temporarily saved in a ring buffer which is continuously overwritten with new values. Only when the configured trigger event is fulfilled, are the temporarily saved values permanently saved and no longer overwritten with new values, whereby the trigger event is dependent on the data type of the trigger variable. The Trace recording ends as soon as the maximum recording duration configured in Trace Sampling is reached.



By defining a pre-trigger, you determine how many of the samples recorded before the trigger event occurs are to remain stored.

Measurement on Device (Memory Card)

Completed measurements can be stored on the memory card in order to start a new measurement. In the item "Measurements on device", you can define if several and, if yes, how many measurements are to be made. In addition, you define whether the oldest measurement is to be deleted when the set number of measurements is reached or whether no more measurements are to be made.

15.19.3. Downloading a TRACE into the CPU and Activating It



Transfer Trace Configuration to Device  / **Add Trace Configuration from the Device to Trace Configurations** 

After the Trace has been configured offline, that is, in the project, the configuration must be downloaded into the CPU, since it is not the engineering tool that executed the Trace but the CPU.

Only Trace configurations that exist online in the CPU can be uploaded from the CPU into the project.

Observe Trace 

Observe Trace displays the status of a Trace on the CPU:

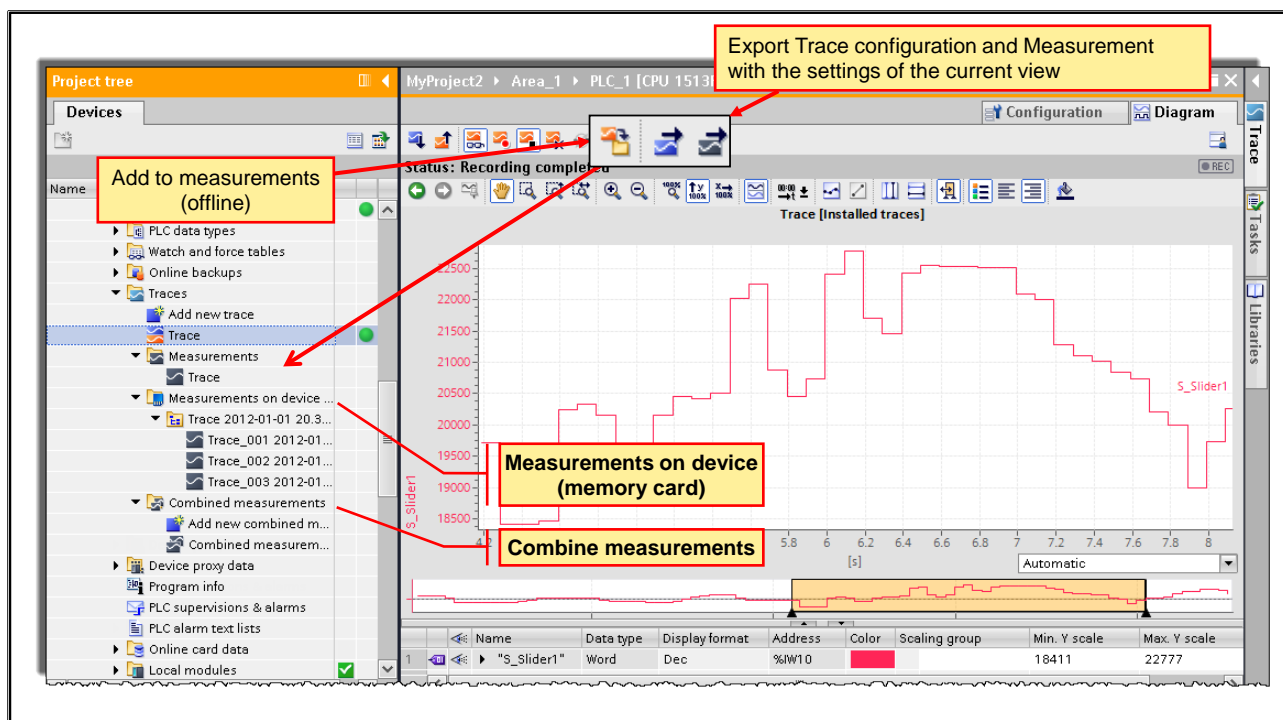
- inactive (Trace already loaded in the CPU, but not yet activated)
- wait for trigger (Trace activated in the CPU, but trigger event not yet fulfilled)
- recording running (Trace activated in the CPU and recording running)
- recording completed (Trace activated in the CPU and recording already completed)

Activate Recording  / **Deactivate Recording** 

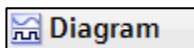
The Trace is activated with "Activate recording", that means that the measuring and recording of the Trace signals is started immediately, even if the possibly configured trigger event is not yet fulfilled. The recorded values are continuously displayed and stored in a ring buffer which is continuously overwritten with new values. Only when the trigger event is fulfilled, are the recorded values no longer overwritten and remain saved. As of this time, the recording is still continued until the maximum recording duration is reached.

Through "Deactivate recording", a Trace with the status "Wait for trigger" is deactivated (stopped) or an already running recording is aborted.

15.19.4. Evaluating, Saving, Exporting a TRACE in STEP 7



View and Evaluate Trace



When an online connection exists, the Trace recording currently saved in the CPU is displayed in the Diagram view of the Trace editor.

You will find the measurements stored on the memory card in the "Measurements on device" folder.

Trace recordings saved offline in the project can be looked at by double-clicking on the Trace recordings saved in the Project tree in the "Measurements" folder.

Furthermore, in the "Combined measurements" folder, measurements can be simultaneously evaluated and compared.

Save Trace in Project



With this function, Traces saved online in the CPU can be uploaded into the offline project (Add to measurements). A Trace can only be saved in the project when it is full or the recording has been stopped.

Trace Configuration



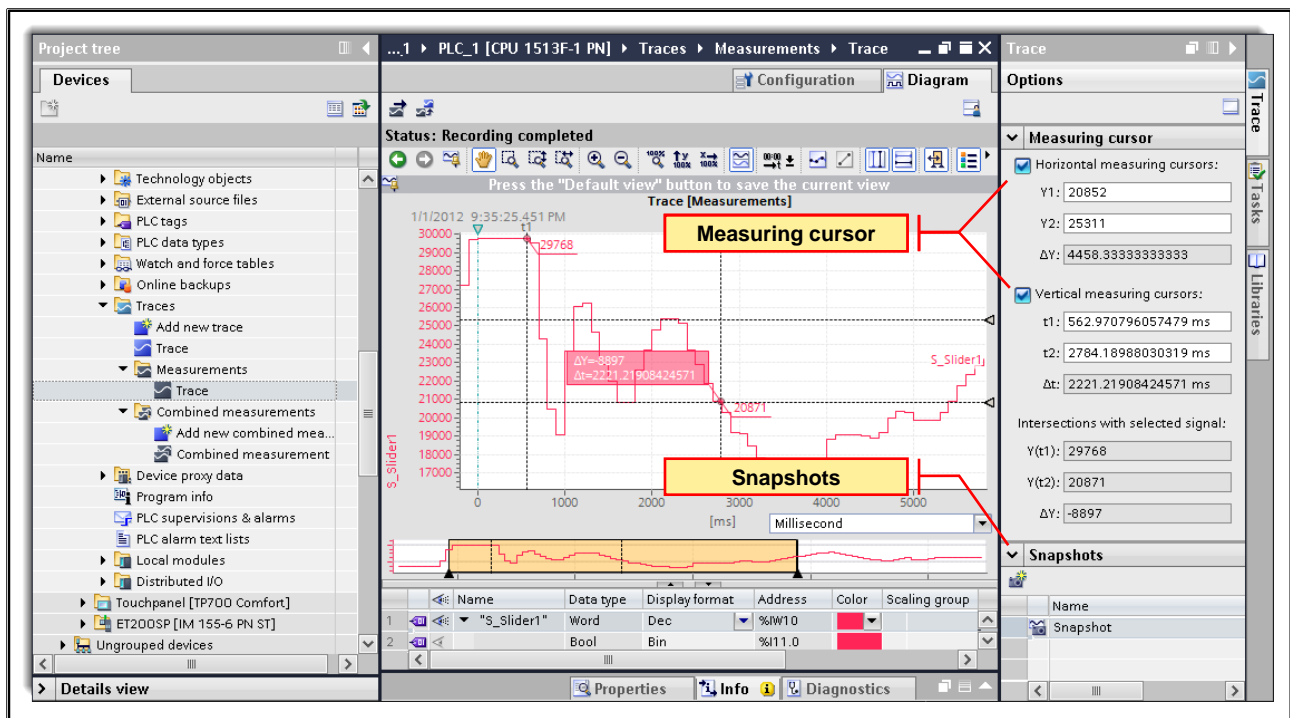
This function exports the configuration of a Trace which can then be imported into other projects. (TTCRX-file)

Trace Measurement:



This function exports a Trace recording in CSV-format which can then be further processed with MS Excel, for example, or, in TTCRX-format which can be imported into a TIA project.

15.19.5. Trace Task Card



"Measuring Cursor" Pane

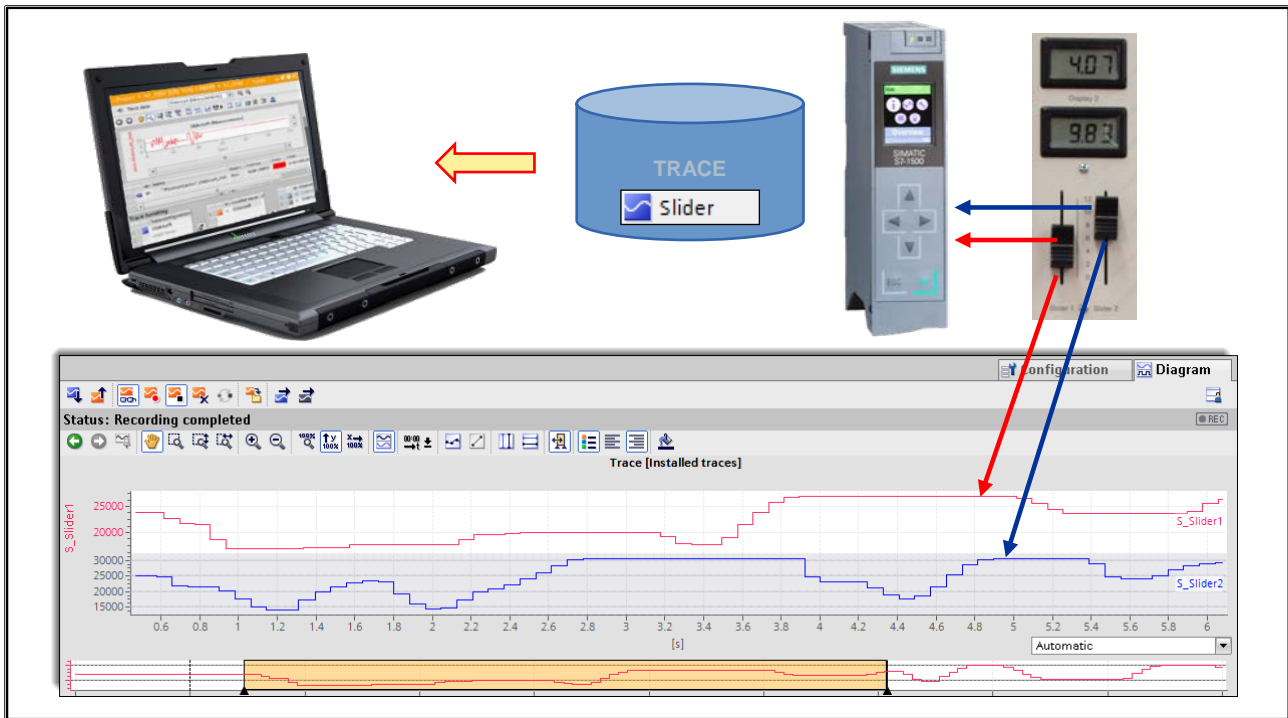
The "Measuring cursor" pane shows the position of the measuring cursor in the graph and the values at the intersections.

"Snapshots" Pane

The "Snapshots" pane enables the saving and restoring of different views of a measurement.

A snapshot is created from the current view in the "Diagram" tab. The snapshots are saved in the measurement with the project.

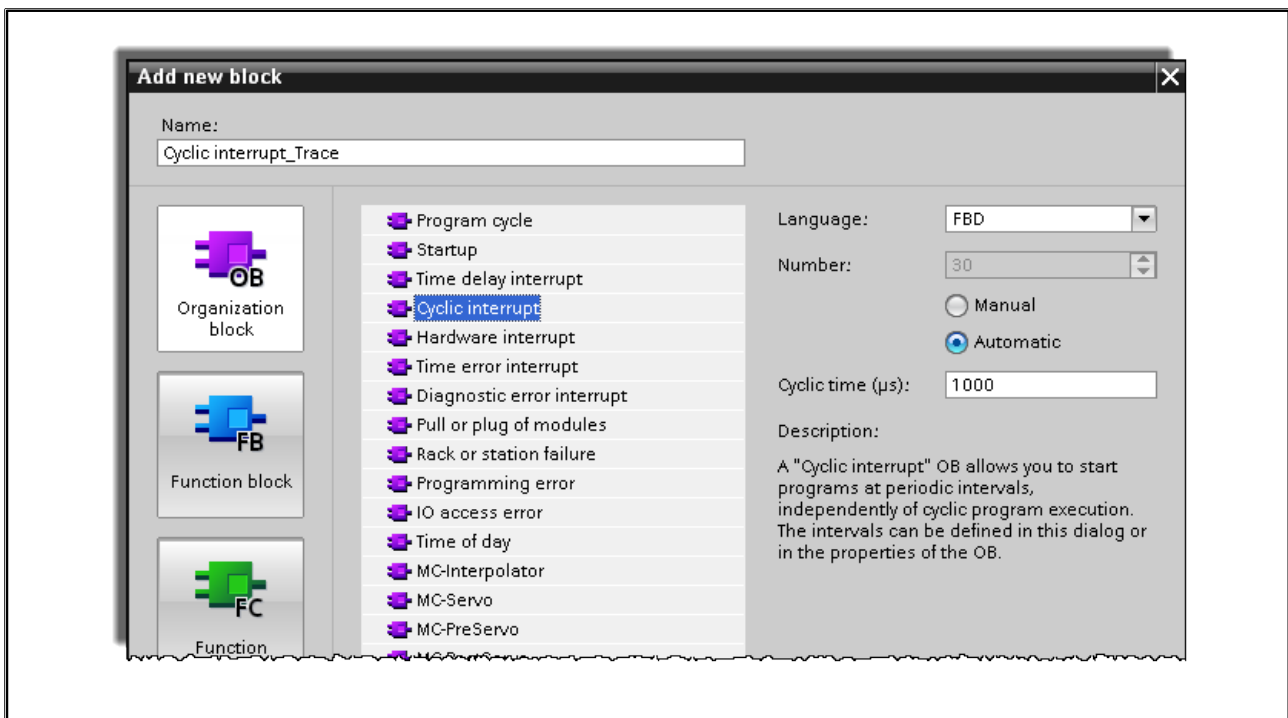
15.20. Task Description: Creating, Looking at and Saving a TRACE



Task Description

The values of Slider 1 and 2 are to be recorded for 10 seconds. The recordings each begin when "P_Operation" (Q0.1) is switched on and are automatically repeated three times. Then, the measurements are to be saved in the project.

15.20.1. Exercise 7: Creating a Cyclic Interrupt OB for the Recording Level



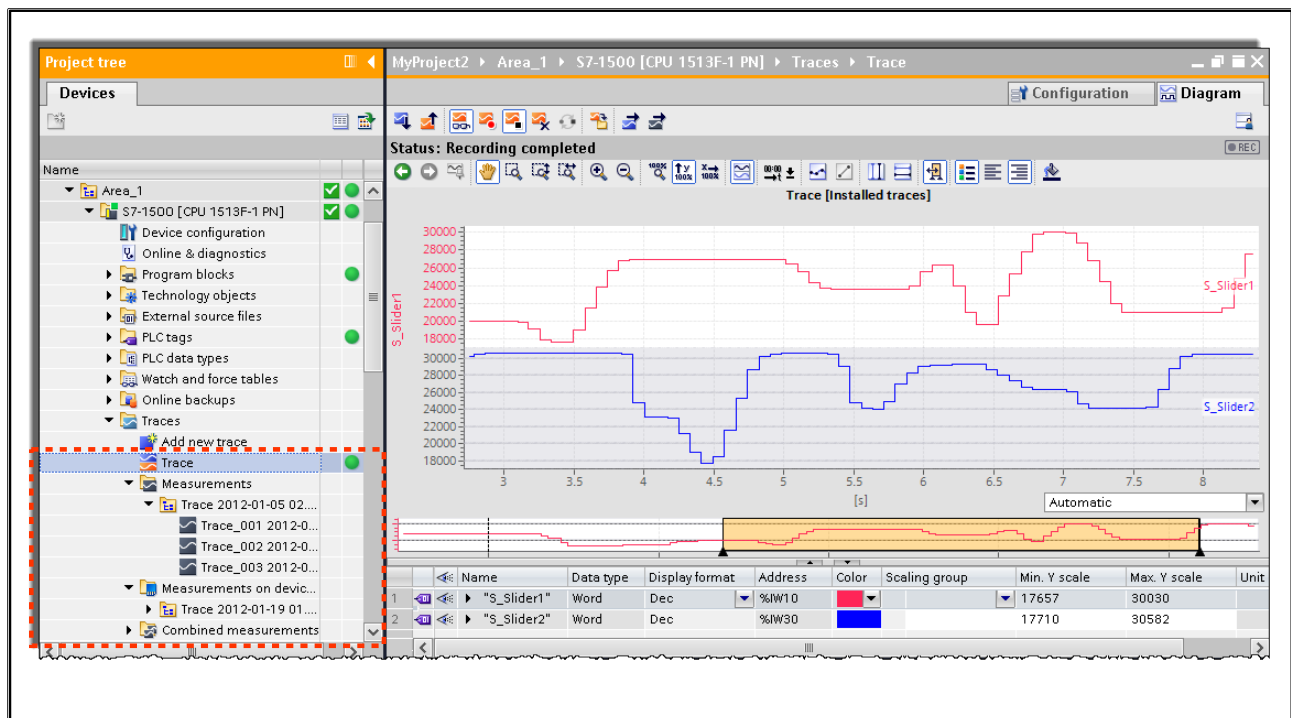
Task

You are to add a cyclic interrupt OB to the program for a regular sampling.

What to Do

1. Start the function "Add new block" in the Program blocks folder.
2. Select the block type "OB" and the start event "Cycle interrupt".
3. Change the Cycle time to 1000µs.
4. Confirm the selection and save your project.

15.20.2. Exercise 8: Creating, Looking at and Saving a TRACE



Task

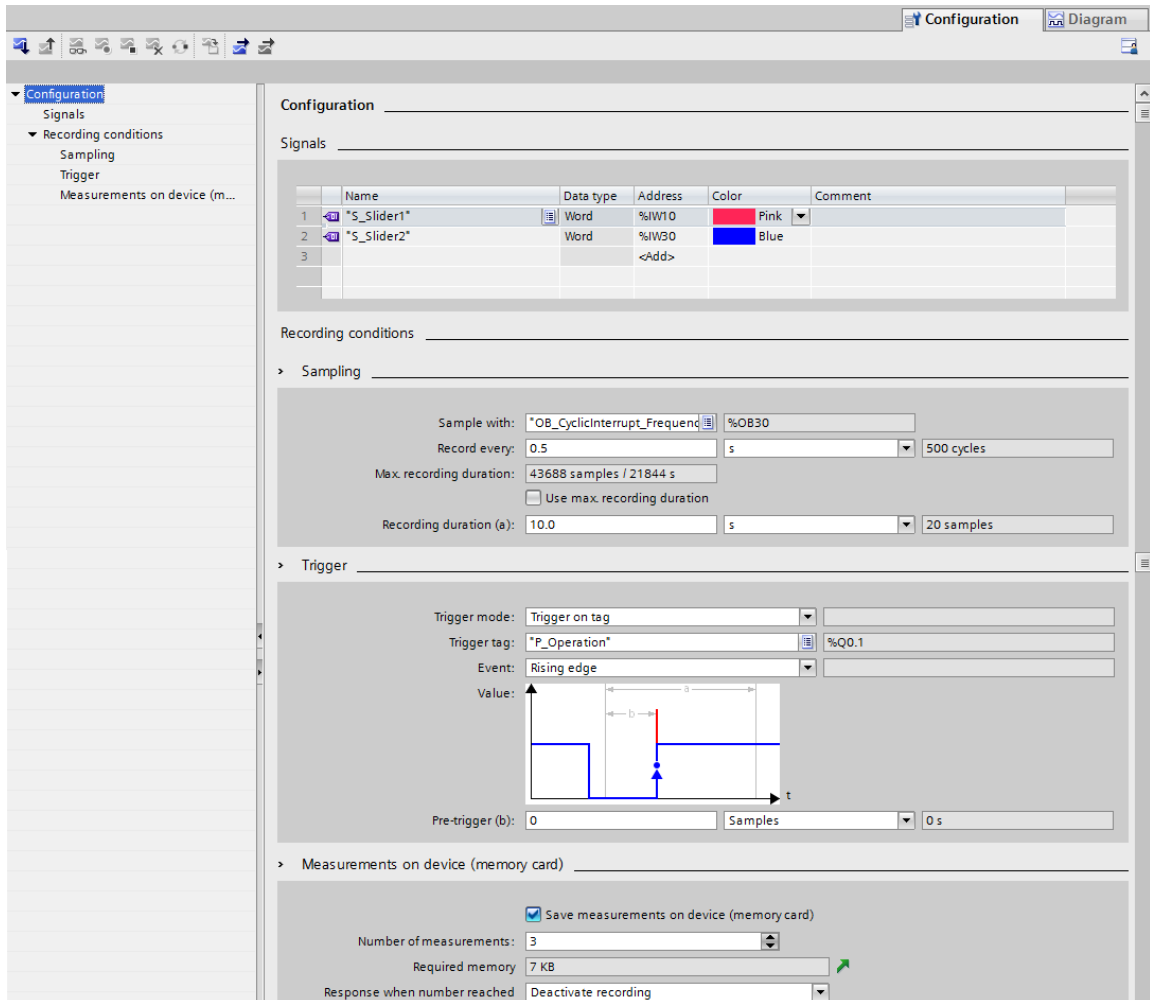
You are to create a Trace configuration with the following properties:

- The Trace signals are to be the variables "S_Slider1" (IW10) and "S_Slider2" (IW30).
- The variables are to be sampled every 0.5 seconds.
- The Trace is to have a recording duration of 10 seconds.
- The trigger condition is a rising edge at the operand "P_Operation".
- The recordings are to begin 1 second before the trigger event.
- After the recording is activated, three measurements are to be recorded and then the recordings are to be stopped.

Continued on the next page

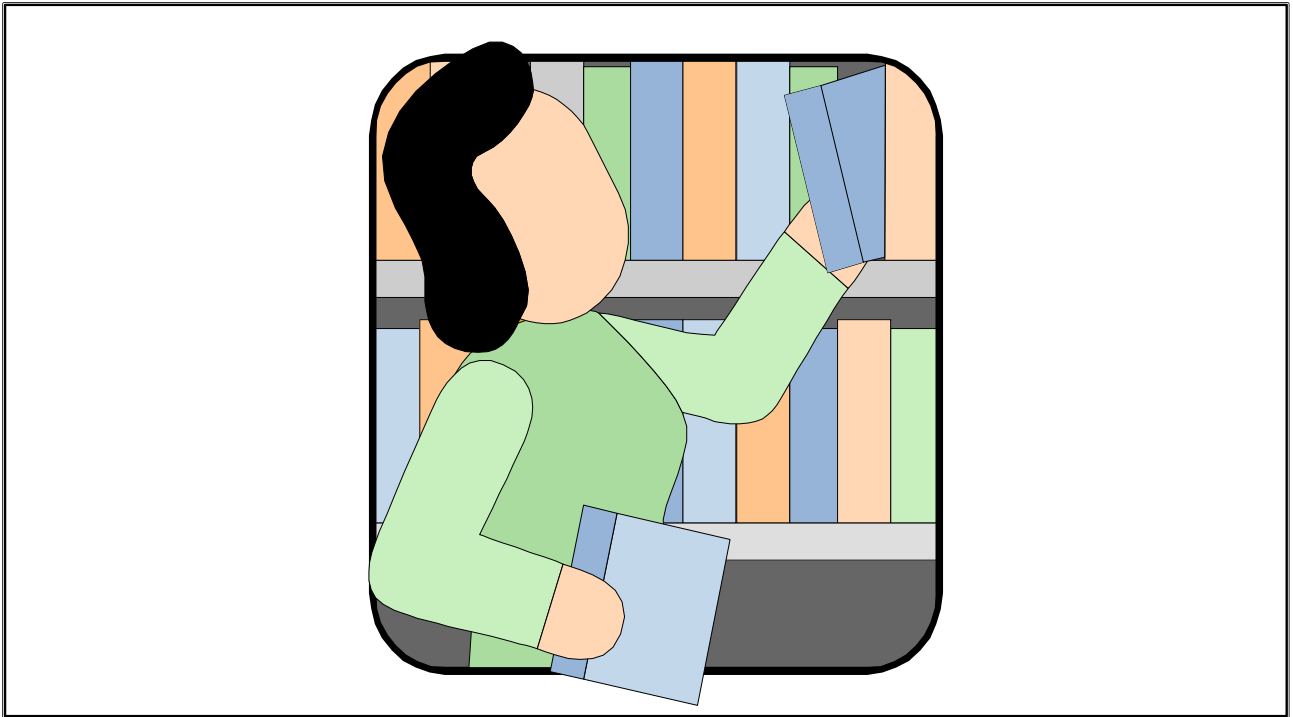
What to Do

1. Create the new trace "Slider" and configure it as shown below.



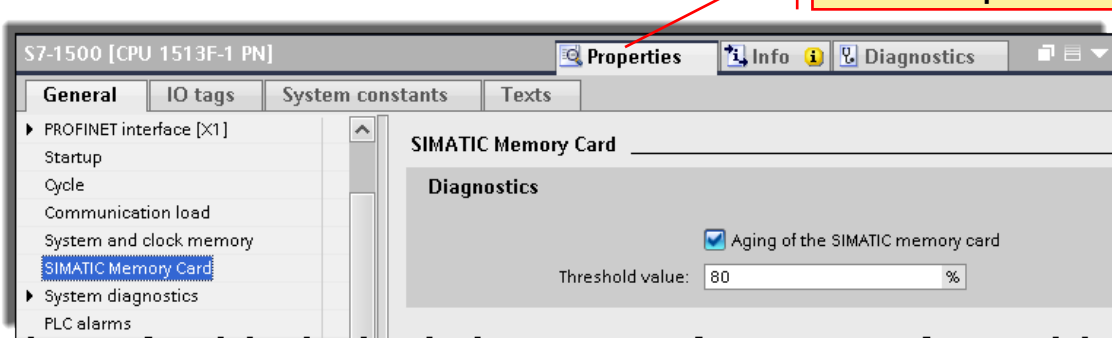
2. Download the trace into the CPU.
3. Activate the recording of the trace and monitor it.
4. Save the measurements in the "Measurements" folder.
5. Save your project.

15.21. Additional Information



15.21.1. Diagnostic Information about the SIMATIC Memory Card

Checking the “Aging” (service life) of the SIMATIC Memory Card



max. delete and write cycles, see Entry ID: 109482591

Diagnostics Setting CPU

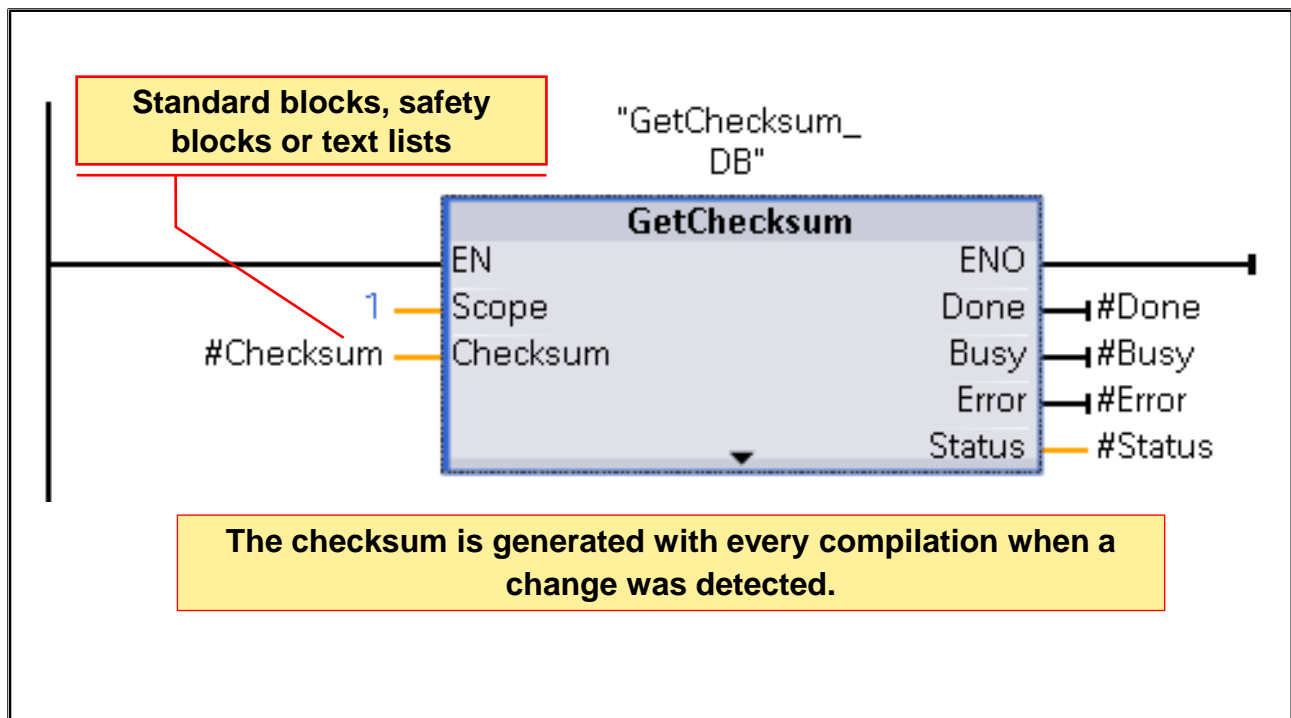
Generation of a diagnostic entry when a service life 'aging' defined by you is exceeded (in percent of the guaranteed write/read cycles)

Optic display on the CPU via the Maintenance-LED

Note:

See **FAQ: 109482591** – "How long is the service life of a 32 GB memory card with the S7-1500 when the minimum number of delete/write procedures is 50,000?"

15.21.2. Reading-out the Checksum



Generating the Checksum

During compilation, PLC programs automatically receive a unique checksum. If, during the next compilation, it is determined that the PLC program was changed, the program receives a new checksum. If the PLC program has not changed and is nevertheless recompiled, the checksum remains the same.

Even when changes are carried out and then are undone, the checksum remains unchanged.

Downloading the Checksum

The checksum is downloaded into the CPU along with the PLC program and is available in the online program. Blocks which are generated or modified during runtime (for example, "WRIT_DBL", "CREAT_DB" and "DELETE_DB") do not change the checksum. When it is uploaded from the CPU, the checksum is not adopted in the offline project since it is automatically regenerated with the next compilation.

Evaluating the Checksum

The checksum is displayed in the CPU Properties (Properties > General > Checksums). From there, you can manually adopt them in your documents. In order to read out the checksum in the program at runtime, the extended instruction "GetChecksum" is available.