

# SCE Training Curriculum

Siemens Automation Cooperates with Education | 05/2017

## TIA Portal Module 032-200 Basics of FB Programming with SIMATIC S7-1500

Cooperates  
with Education  
  
Automation

**SIEMENS**

## Matching SCE trainer packages for these training curriculums

### SIMATIC Controllers

- **SIMATIC ET 200SP Open Controller CPU 1515SP PC F and HMI RT SW**  
Order no.: 6ES7677-2FA41-4AB1
- **SIMATIC ET 200SP Distributed Controller CPU 1512SP F-1 PN Safety**  
Order no.: 6ES7512-1SK00-4AB2
- **SIMATIC CPU 1516F PN/DP Safety**  
Order no.: 6ES7516-3FN00-4AB2
- **SIMATIC S7 CPU 1516-3 PN/DP**  
Order no.: 6ES7516-3AN00-4AB3
- **SIMATIC CPU 1512C PN with Software and PM 1507**  
Order no.: 6ES7512-1CK00-4AB1
- **SIMATIC CPU 1512C PN with Software, PM 1507 and CP 1542-5 (PROFIBUS)**  
Order no.: 6ES7512-1CK00-4AB2
- **SIMATIC CPU 1512C PN with Software**  
Order no.: 6ES7512-1CK00-4AB6
- **SIMATIC CPU 1512C PN with Software and CP 1542-5 (PROFIBUS)**  
Order no.: 6ES7512-1CK00-4AB7

### SIMATIC STEP 7 Software for Training

- **SIMATIC STEP 7 Professional V14 SP1 - Single license**  
Order no.: 6ES7822-1AA04-4YA5
- **SIMATIC STEP 7 Professional V14 SP1- Classroom license (up to 6 users)**  
Order no.: 6ES7822-1BA04-4YA5
- **SIMATIC STEP 7 Professional V14 SP1 - Upgrade license (up to 6 users)**  
Order no.: 6ES7822-1AA04-4YE5
- **SIMATIC STEP 7 Professional V14 SP1 - Student license (up to 20 users)**  
Order no.: 6ES7822-1AC04-4YA5

Note that these trainer packages are replaced with successor packages when necessary.  
An overview of the currently available SCE packages is provided at: [siemens.com/sce/tp](http://siemens.com/sce/tp)

## Continued training

For regional Siemens SCE continued training, get in touch with your regional SCE contact  
[siemens.com/sce/contact](http://siemens.com/sce/contact)

## Additional information regarding SCE

[siemens.com/sce](http://siemens.com/sce)

## Information regarding use

The SCE training curriculum for the integrated automation solution Totally Integrated Automation (TIA) was prepared for the program "Siemens Automation Cooperates with Education (SCE)" specifically for training purposes for public educational and R&D institutions. Siemens AG does not guarantee the contents.

This document is to be used only for initial training on Siemens products/systems. This means it can be copied in whole or part and given to those being trained for use within the scope of their training. Circulation or copying this training curriculum and sharing its content is permitted within public training and advanced training facilities for training purposes.

Exceptions require written consent from the Siemens AG contact: Roland Scheuerer [roland.scheuerer@siemens.com](mailto:roland.scheuerer@siemens.com).

Offenders will be held liable. All rights including translation are reserved, particularly if a patent is granted or a utility model or design is registered.

Use for industrial customer courses is expressly prohibited. We do not consent to commercial use of the training curriculums.

We wish to thank the TU Dresden, particularly Prof. Dr.-Ing. Leon Urbas, the Michael Dziallas Engineering Corporation and all other involved persons for their support during the preparation of this training curriculum.

# Table of contents

1	Goal .....	5
2	Prerequisite.....	5
3	Required hardware and software .....	6
4	Theory.....	7
4.1	Operating system and application program .....	7
4.2	Organization blocks .....	8
4.3	Process image and cyclic program processing.....	9
4.4	Functions .....	11
4.5	Function blocks and instance data blocks.....	12
4.6	Global data blocks .....	13
4.7	Library-compatible code blocks.....	14
4.8	Programming languages .....	15
5	Task .....	16
6	Planning .....	16
6.1	EMERGENCY STOP .....	16
6.2	Automatic mode - Conveyor motor.....	16
7	Structured step-by-step instructions .....	17
7.1	Retrieve an existing project .....	17
7.2	Create a new tag table .....	18
7.3	Create new tags within a tag table .....	20
7.4	Import "Tag_table_sorting_station" .....	21
7.5	Create function block FB1 "MOTOR_AUTO" for the conveyor motor in automatic mode .....	25
7.6	Define the interface of FB1 "MOTOR_AUTO" .....	27
7.7	Program FB1: MOTOR_AUTO.....	30
7.8	Program organization block OB1 – Control of the forward belt tracking in automatic mode .....	38
7.9	The result in the LAD (Ladder Logic) programming language has the following appearance....	43
7.10	Save and compile the program .....	44
7.11	Download the program .....	45
7.12	Monitor program blocks.....	46
7.13	Archive the project.....	49
8	Checklist .....	50
9	Exercise .....	51
9.1	Task – Exercise .....	51
9.2	Planning.....	51
9.3	Checklist – Exercise .....	52
10	Additional information .....	53

# BASICS OF FB PROGRAMMING

## 1 Goal

In this chapter, you will get to know the basic elements of a control program – the **organization blocks (OBs)**, **functions (FCs)**, **function blocks (FBs)** and **data blocks (DBs)**. In addition, you will be introduced to **library-compatible** function und function block programming. You will get to know the **Function Block Diagram (FBD)** programming language and use it to program a function block (FB1) and an organization block (OB1).

The SIMATIC S7 controllers listed in Chapter 3 can be used.

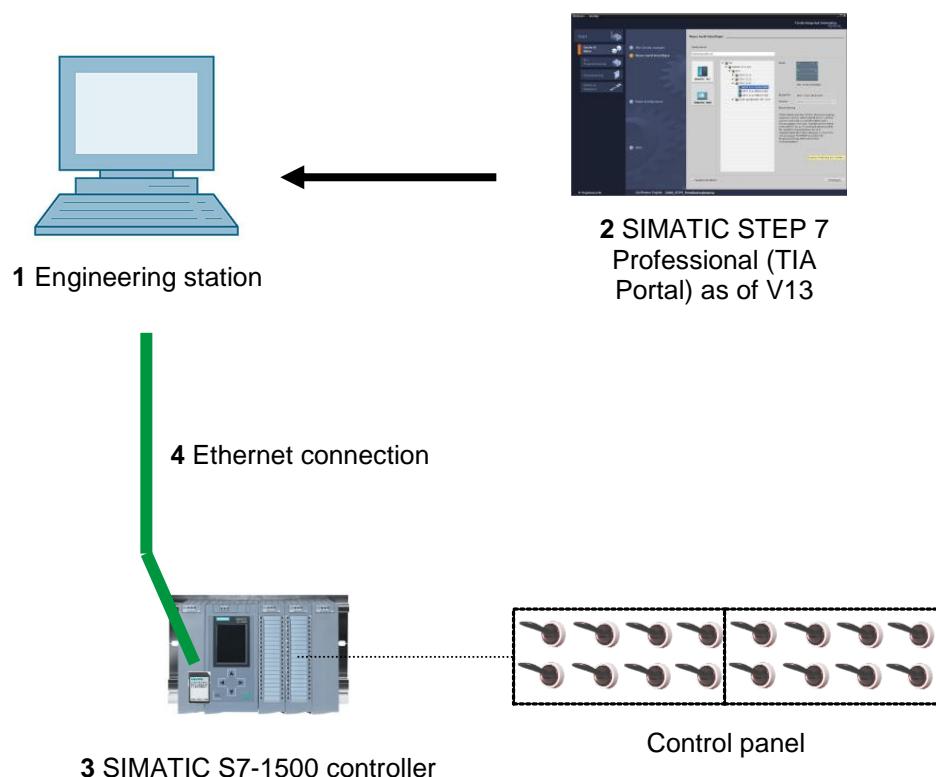
## 2 Prerequisite

This chapter builds on the hardware configuration of SIMATIC S7 CPU1516F-3 PN/DP. However, other hardware configurations that have digital input and output cards can be used. You can use the following project for this chapter, for example:

[SCE\\_EN\\_012\\_101\\_Hardware\\_Configuration\\_CPU1516F.zap13](#)

### 3 Required hardware and software

- 1 Engineering station: requirements include hardware and operating system  
(for additional information, see Readme on the TIA Portal Installation DVDs)
- 2 SIMATIC STEP 7 Professional software in TIA Portal – as of V13
- 3 SIMATIC S7-1500/S7-1200/S7-300 controller, e.g. CPU 1516F-3 PN/DP –  
Firmware as of V1.6 with memory card and 16DI/16DO and 2AI/1AO  
Note: The digital inputs should be fed out to a control panel.
- 4 Ethernet connection between engineering station and controller



## 4 Theory

### 4.1 Operating system and application program

Every controller (CPU) contains an **operating system**, which organizes all functions and processes of the CPU that are not associated with a specific control task. The tasks of the operating system include the following:

- Performing a warm restart
- Updating the process image of the inputs and outputs
- Cyclically calling the user program
- Detecting interrupts and calling interrupt OBs
- Detecting and handling errors
- Managing memory areas

The operating system is an integral component of the CPU and comes pre-installed.

The **user program** contains all functions that are necessary for executing your specific automation task. The tasks of the user program include the following:

- Checking the basic requirements for a warm restart using startup OBs
- Processing of process data, i.e. activation of output signals as a function of the input signal states
- Reaction to interrupts and interrupt inputs
- Error handling during normal program execution

## 4.2 Organization blocks

Organization blocks (OBs) form the interface between the operating system of the controller (CPU) and the application program. They are called from the operating system and control the following operations:

- Cyclic program processing (e.g. OB1)
- Startup characteristics of the controller
- Interrupt-driven program processing
- Error handling

A project must have ***an organization block for cyclic program processing*** at a minimum. An OB is called by a ***start event*** as shown in Figure 1. In addition, the individual OBs have defined priorities so that, for example, an OB82 for error handling can interrupt the cyclic OB1.

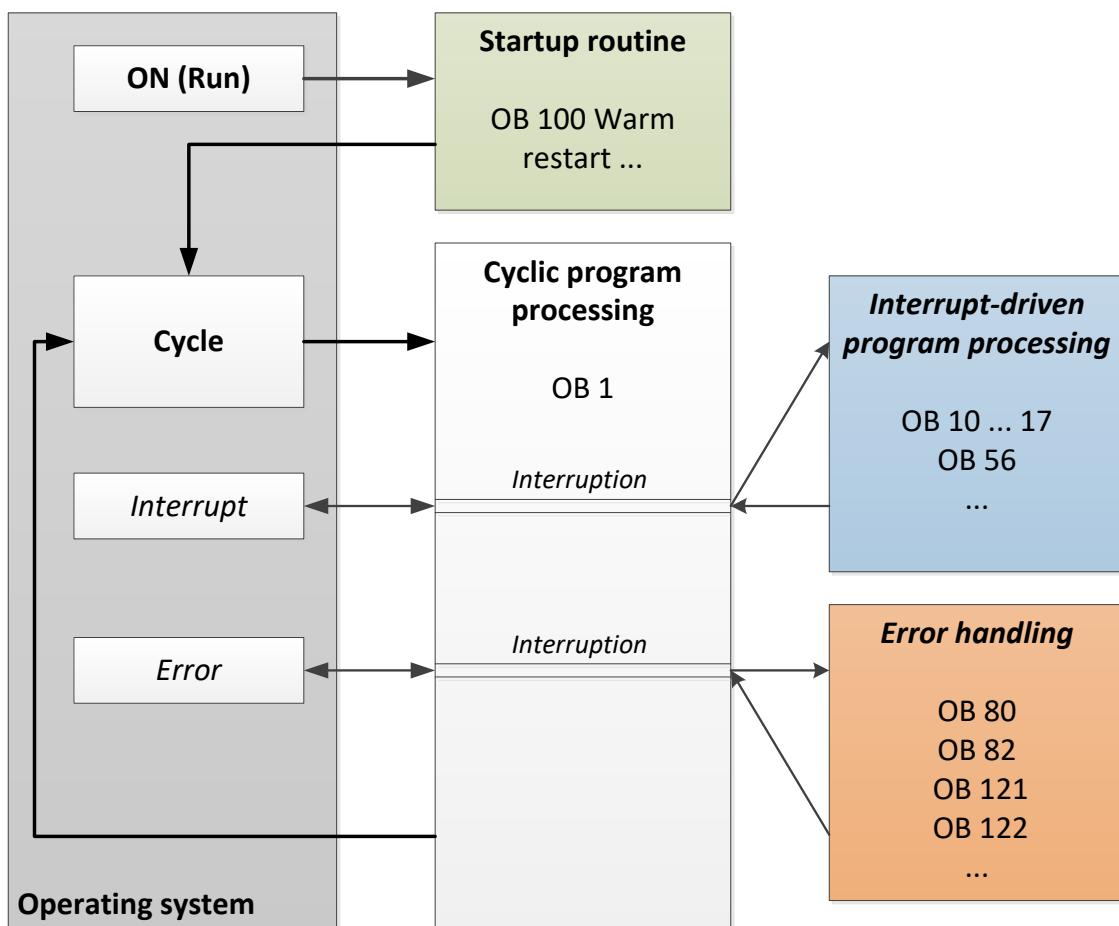


Figure 1: Start events in the operating system and OB calls

When a start event occurs, the following reactions are possible:

- If an OB has been assigned to the event, this event triggers the execution of the assigned OB. If the priority of the assigned OB is greater than the priority of the OB that is currently being executed, it is executed immediately (interrupt). If not, the assigned OB waits until the higher-priority OB has been completely executed.
- If an OB is not assigned to the event, the default system reaction is performed.

Table 1 gives a couple of examples of start events for a SIMATIC S7-1500, their possible OB number(s) and the default system reaction in the event the organization block is not present in the controller.

<b>Start event</b>	<b>Possible OB numbers</b>	<b>Default system reaction</b>
Startup	100, $\geq 123$	Ignore
<b>Cyclic program</b>	1, $\geq 123$	Ignore
Time-of-day interrupt	10 to 17, $\geq 123$	-
Update interrupt	56	Ignore
Scan cycle monitoring time exceeded once	80	STOP
Diagnostic interrupt	82	Ignore
Programming error	121	STOP
IO access error	122	Ignore

Table 1: OB numbers for various start events

### 4.3 Process image and cyclic program processing

When the cyclic user program addresses the inputs (I) and outputs (O), it does not query the signal states directly from the input/output modules. Instead, it accesses a memory area of the CPU. This memory area contains an image of the signal states and is called the **process image**.

The cyclic program processing sequence is as follows:

1. At the start of the cyclic program, a query is sent to determine whether or not the individual inputs are energized. This status of the inputs is stored in the **process image of the inputs (PII)**. In doing so, the information 1 or "High" is stored for energized inputs and the information 0 or "Low" for de-energized inputs.
2. The CPU now executes the program stored in the cyclic organization block. For the required input information, the CPU accesses the previously read **process image of the inputs (PII)** and the results of logic operation (RLOs) are written to a so-called **process image of the outputs (PIQ)**.
3. At the end of the cycle, the **process image of the outputs (PIQ)** is transferred as the signal state to the output modules and these are energized or de-energized. The sequence then continues again with Item 1.

**1. Save status of inputs in PII.**

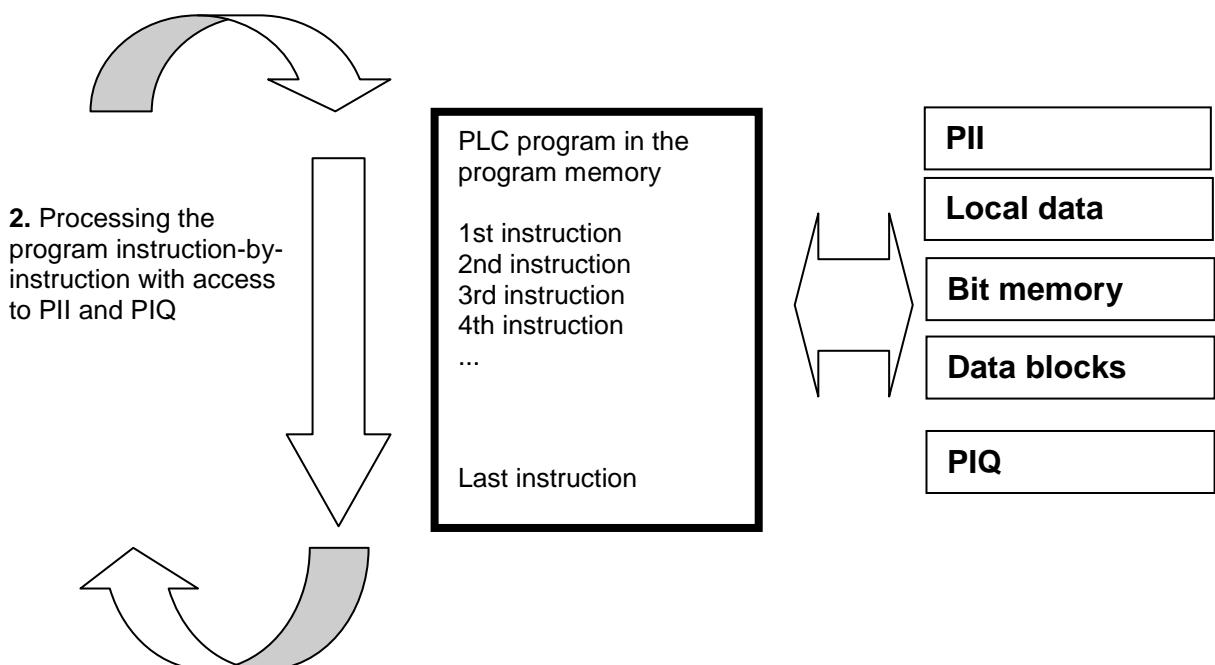


Figure 2: Cyclic program processing

**Note:** The time the CPU needs for this sequence is called *cycle time*. This depends, in turn, on the number and type of instructions and the processor performance of the controller.

## 4.4 Functions

Functions (FCs) are logic blocks without memory. They **have no data memory** in which values of block parameters can be stored. Therefore, all interface parameters must be connected when a function is called. To store data permanently, global data blocks must be created beforehand.

A function contains a program that is executed whenever the function is called from another code block.

Functions can be used, for example, for the following purposes:

- Math functions – that return a result dependent on input values.
- Technological functions – such as individual controls with binary logic operations.

A function can also be called several times at different points within a program.

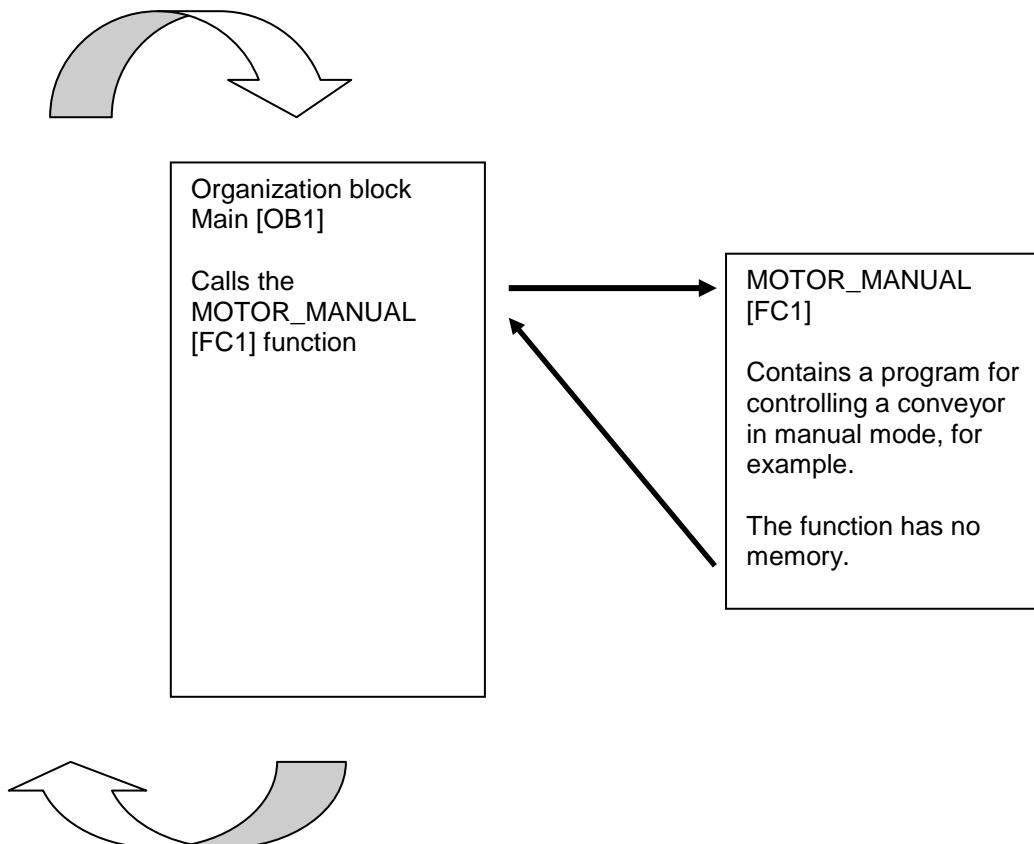


Figure 3: Function with call from organization block Main [OB1]

## 4.5 Function blocks and instance data blocks

Function blocks are code blocks that store their input, output and in-out tags as well as static tags permanently in instance data blocks, so that they **are available after the block has been executed**. For this reason, they are also referred to as blocks with "memory".

Function blocks can also operate with temporary tags. Temporary tags are not stored in the instance DB, however. Instead, they are only available for one cycle.

Function blocks are used for tasks that cannot be implemented with functions:

- Whenever timers and counters are required in the blocks.
- Whenever information must be saved in the program, such as pre-selection of the operating mode with a button.

Function blocks are always executed when called from another code block. A function block can also be called several times at different points within a program. This facilitates the programming of frequently recurring complex functions.

A call of a function block is referred to as an instance. Each instance of a function block is assigned a memory area that contains the data that the function block uses. This memory is made available by data blocks created automatically by the software.

It is also possible to provide memory for multiple instances in one data block in the form of a **multi-instance**. The maximum size of instance data blocks varies depending on the CPU. The tags declared in the function block determine the structure of the instance data block.

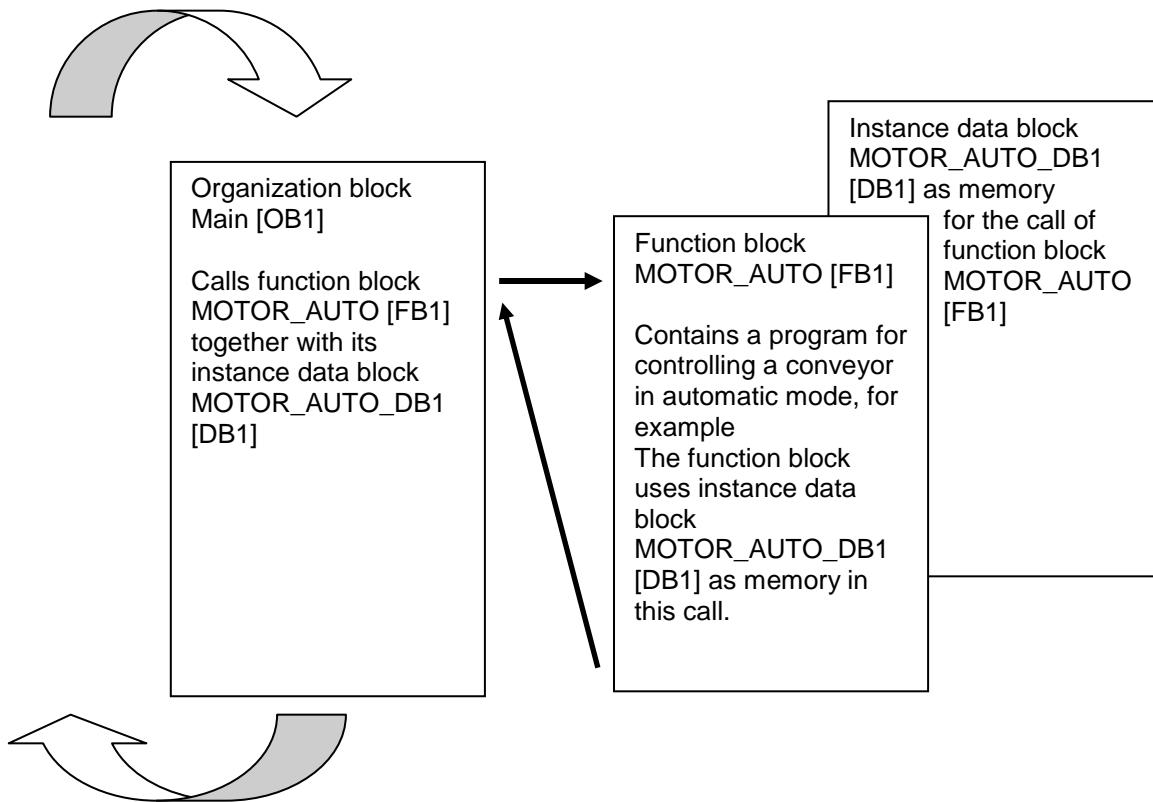


Figure 4: Function block and instance with call from organization block Main [OB1]

## 4.6 Global data blocks

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

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

Global data blocks store data that can be used **by all other blocks** (see Figure 5). Only the associated function block should access instance data blocks. The maximum size of data blocks varies depending on the CPU.

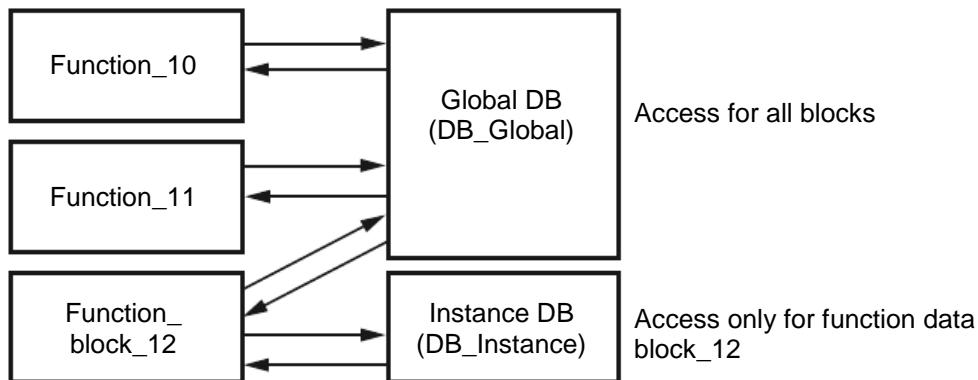


Figure 5: Difference between global DB and instance DB.

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

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

## 4.7 Library-compatible code blocks

A user program can be created with linear or structured programming. **Linear programming** writes the entire user program in the cycle OB, but is only suitable for very simple programs for which other less expensive control systems, such as LOGO!, can now be used.

**Structured programming** is always recommended for more complex programs. Here, the overall automation task can be broken down into small sub-tasks in order to implement a solution for them in functions and function blocks.

In this case, library-compatible logic blocks should be created preferentially. This means that the input and output parameters of a function or function block are defined generally and only supplied with the current global tags (inputs/outputs) when the block is used.

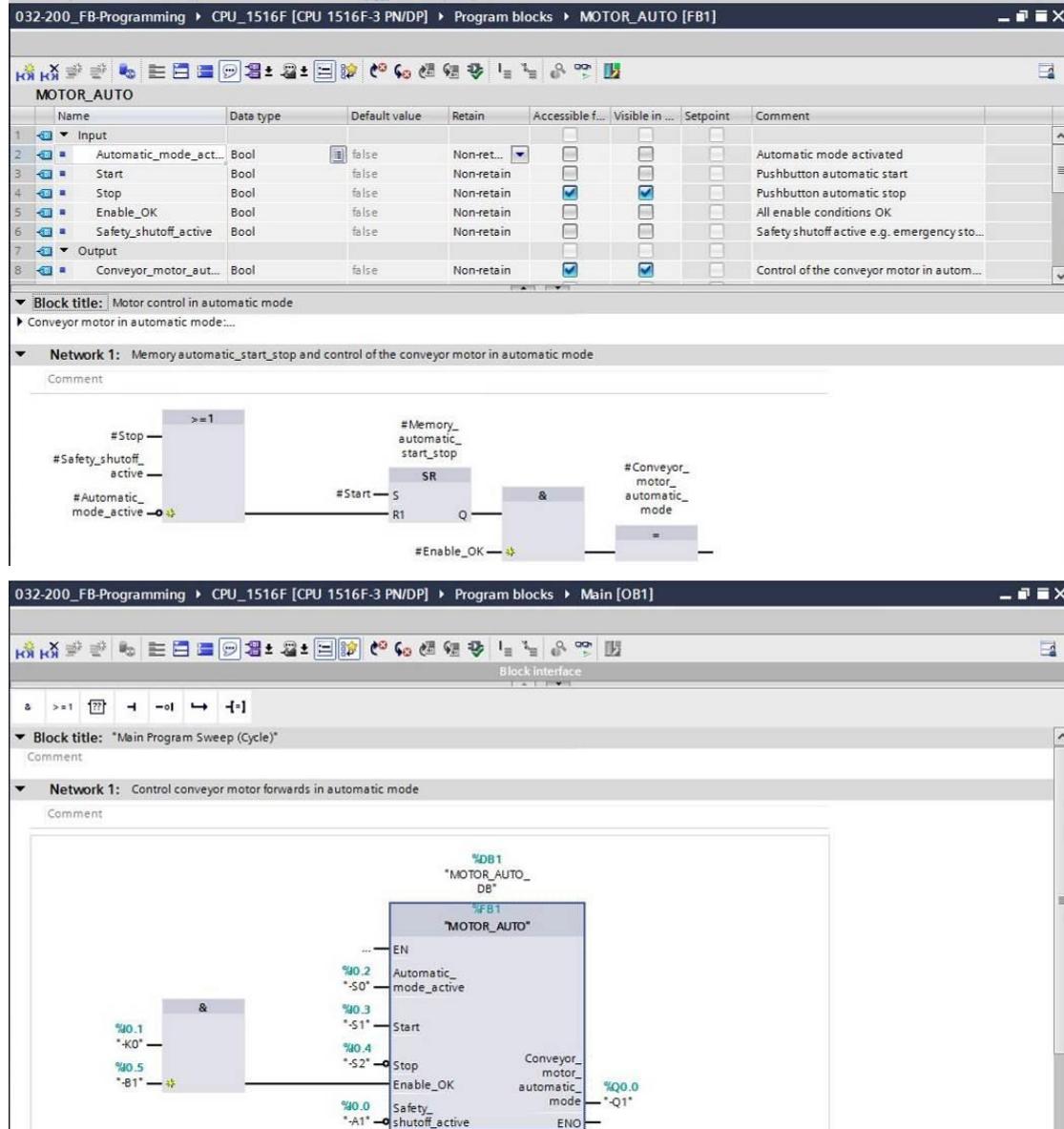


Figure 6: Library-compatible function with call in OB1

## 4.8 Programming languages

The available programming languages for programming functions are Function Block Diagram (FBD), Ladder Logic (LAD), Statement List (STL) and Structured Control Language (SCL). For function blocks, the GRAPH programming language is additionally available for programming graphical step sequences.

The **Function Block Diagram (FBD)** programming language will be presented in the following.

FBD is a graphical programming language. The representation is based on electronic switching systems. The program is mapped in networks. A network contains one or more logic operation paths. Binary and analog signals are linked by boxes. The graphical logic symbols known from Boolean algebra are used to represent the binary logic.

You can use binary functions to query binary operands and to logically combine their signal states. The following instructions are examples of binary functions: "AND operation", "OR operation" and "EXCLUSIVE OR operation". These are shown in Figure 7.

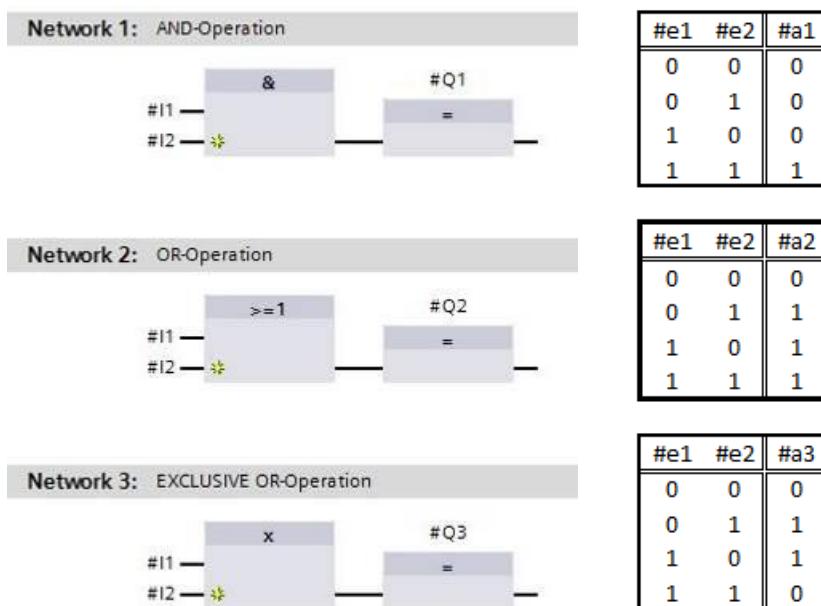


Figure 7: Binary functions in FBD and associated logic table

You can thus use simple instructions, for example, to control binary outputs, evaluate edges and execute jump functions in the program.

Program elements such as IEC timers and IEC counters provide complex instructions.

The empty box serves as a placeholder in which you can select the required instruction.

Enable input EN (enable)/ Enable output ENO (enable output) mechanism:

- An instruction without EN/ENO mechanism is executed independent of the signal state at the box inputs.
- Instructions with EN/ENO mechanism are only executed if enable input "EN" input has signal state "1". When the box is processed correctly, enable output "ENO" has signal state "1". As soon as an error occurs during the processing, the "ENO" enable output is reset. If enable input EN is not connected, the box is always executed.

## 5 Task

The following functions of the sorting station process description will be planned, programmed and tested in this chapter:

- Automatic mode - Conveyor motor

## 6 Planning

The programming of all functions in OB1 is not recommended for reasons of clarity and reusability. The majority of the program code will therefore be moved into functions (FCs) and function blocks (FBs). The decision on which functions is to be moved to the FB and which is to run in OB 1 is planned below.

### 6.1 EMERGENCY STOP

The EMERGENCY STOP does not require a separate function. Just like the operating mode, the current state of the EMERGENCY STOP relay can be used directly at the blocks.

### 6.2 Automatic mode - Conveyor motor

Automatic mode of the conveyor motor is to be encapsulated in a function block (FB) "MOTOR\_AUTO". On the one hand, this preserves the clarity of OB1. On the other hand, it enables reuse if another conveyor belt is added to the station. Table 2 lists the planned parameters.

<b>Input</b>	Data	Comment
Automatic_mode_active	BOOL	Automatic mode activated
Start	BOOL	Pushbutton automatic start
Stop	BOOL	Pushbutton automatic stop
Enable_OK	BOOL	All enable conditions OK
Safety_shutoff_active	BOOL	Safety shutoff active, e.g. emergency stop pressed
<b>Output</b>		
Conveyor_motor_automatic_mode	BOOL	Control of the conveyor motor in automatic mode
<b>Static</b>		
Memory_automatic_start_stop	BOOL	Memory used for start/stop automatic mode

Table 2: Parameters for FB "MOTOR\_AUTO"

The Memory\_automatic\_start\_stop is latched with Start but only if the reset conditions are not present.

The Memory\_automatic\_start\_stop is reset if Stop is present or safety shutoff is active or automatic mode is not activated (manual mode).

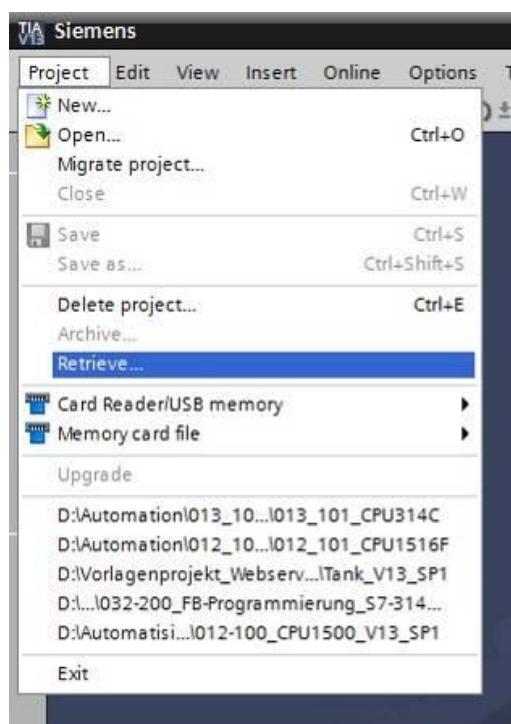
The Conveyor\_motor\_automatic\_mode output is controlled when Memory\_automatic\_start\_stop is set and the enable conditions are met.

## 7 Structured step-by-step instructions

You can find instructions on how to carry out planning below. If you already have a good understanding of everything, it will be sufficient to focus on the numbered steps. Otherwise, simply follow the detailed steps in the instructions.

### 7.1 Retrieve an existing project

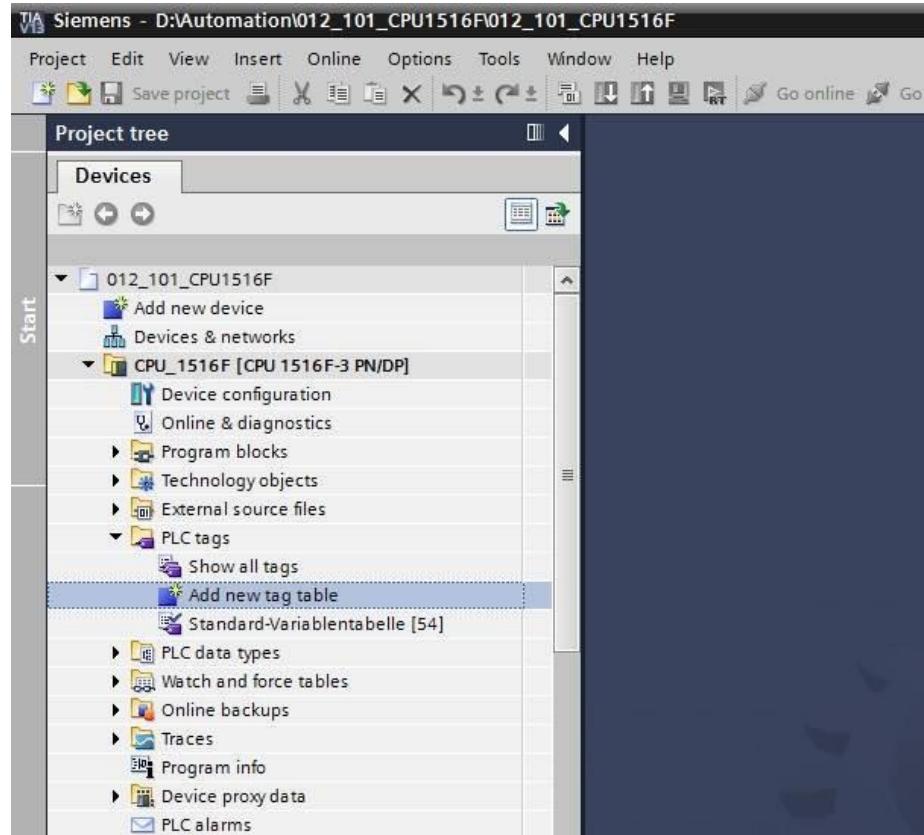
- Before we can start programming the function block (FB) "MOTOR\_AUTO", we need a project with a hardware configuration (e.g. SCE\_EN\_012\_101\_Hardware\_Configuration\_S7-1516F\_R1502.zap). To retrieve an existing project that has been archived, you must select the relevant archive with → Project → Retrieve in the project view. Confirm your selection with Open. (→ Project → Retrieve → Select a .zap archive → Open)



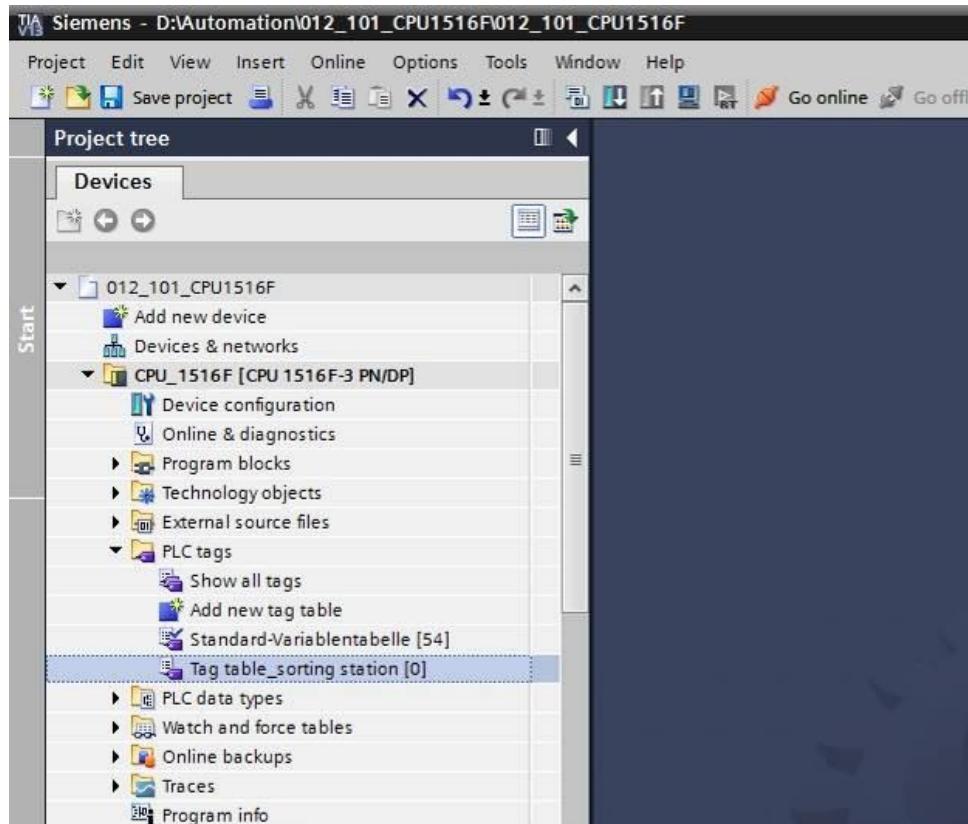
- The next step is to select the target directory where the retrieved project will be stored. Confirm your selection with "OK". (→ Target directory → OK)

## 7.2 Create a new tag table

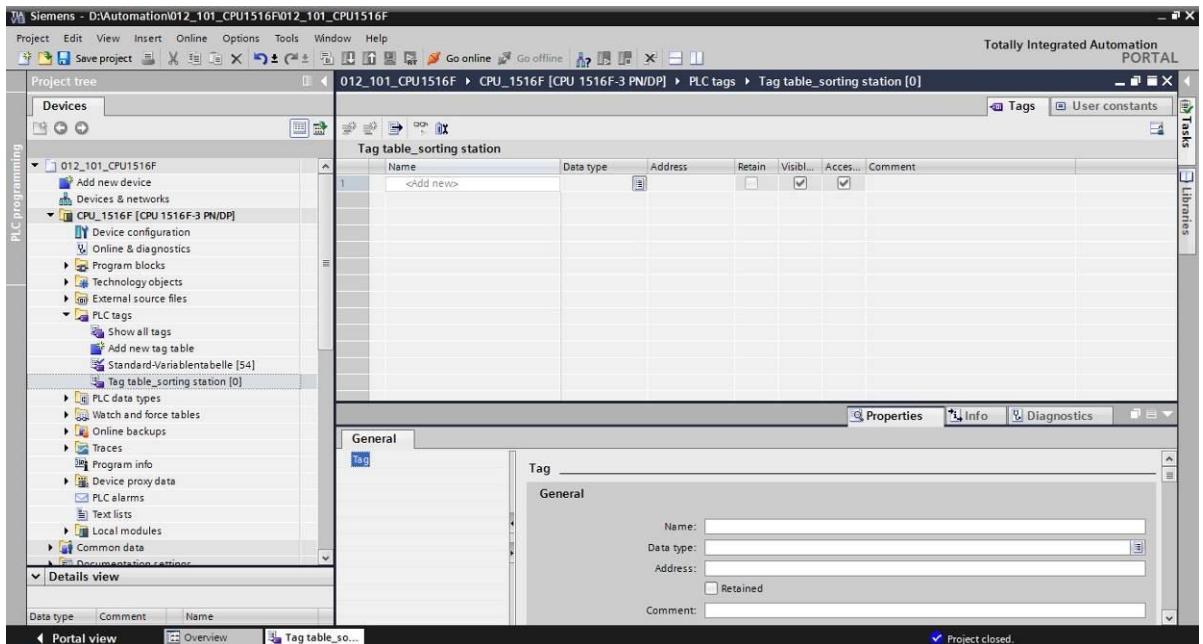
- In the project view, navigate to the → PLC tags of your controller and create a new tag table by double-clicking → Add new tag table.



- Rename the tag table you just created as "Tag\_table\_sorting\_station" (→ right-click "Tag\_table\_1" → "Rename" → Tag\_table\_sorting\_station).

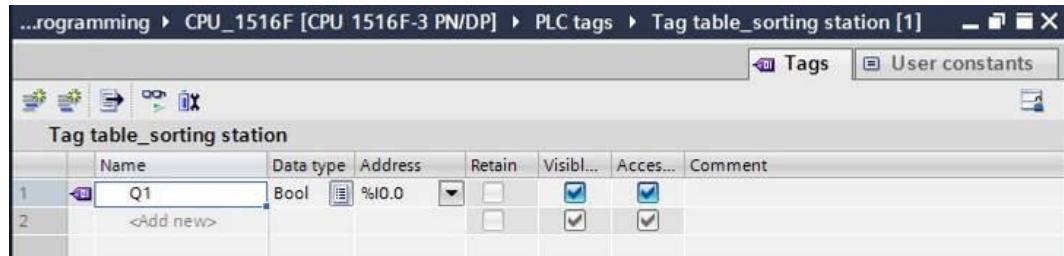


- Open this tag table with a double-click. (→ Tag\_table\_sorting\_station)

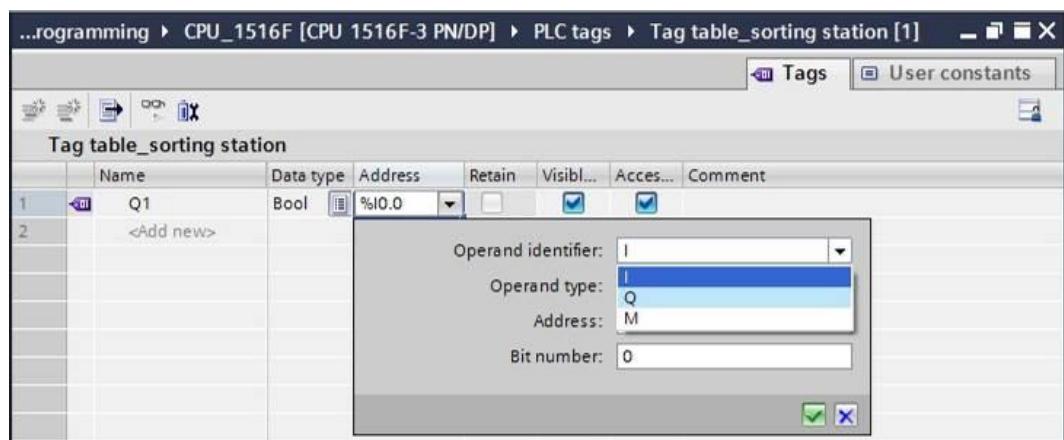


### 7.3 Create new tags within a tag table

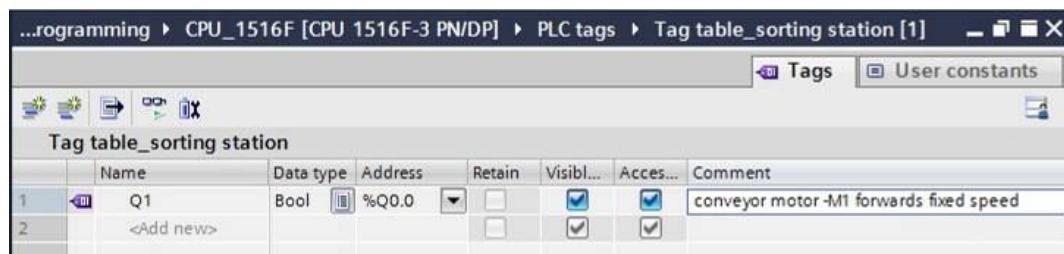
- Add the name Q1 and confirm the entry with the Enter key. If you have not yet created additional tags, TIA Portal now automatically assigns data type "Bool" and address %I0.0 (I 0.0) (→ <Add> → Q1 → Enter).



- Change the address to %Q0.0 (Q 0.0) by entering this directly or by clicking the drop-down arrow to open the Addressing menu, changing the operand identifier to Q and confirming with Enter or by clicking the check mark. (→ %I0.0 → Operand identifier → Q → )



- Enter the "Conveyor motor M1 forwards fixed speed" comment for the tag.



- Add a new Q2 tag in line 2. TIA Portal has automatically assigned the same data type as in line 1 and has incremented the address by 1 to %Q0.1 (Q0.1). Enter the comment "Conveyor motor M1 backwards fixed speed".

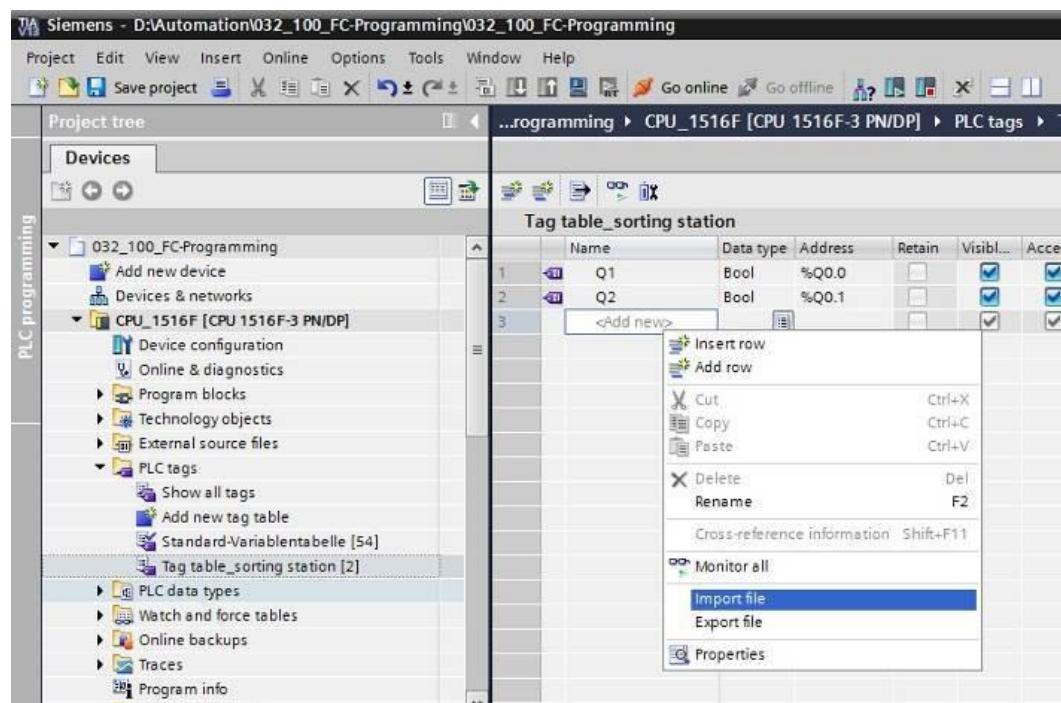
(→ <Add> → Q2 → Enter → Comment → Conveyor motor M1 backwards fixed speed)

	Name	Data type	Address	Retain	Visible...	Access...	Comment
1	Q1	Bool	%Q0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor motor -M1 forwards fixed speed
2	Q2	Bool	%Q0.1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor motor -M1 backwards fixed speed
3	<Add new>			<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

## 7.4 Import "Tag\_table\_sorting\_station"

- To insert an existing symbol table, right-click on an empty field of the created "Tag\_table\_sorting\_station". Select "Import file" in the shortcut menu.

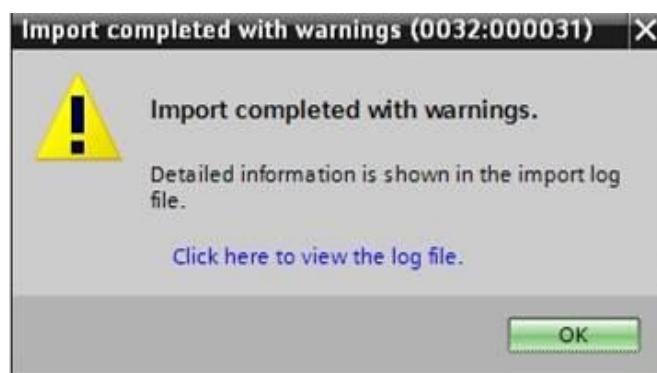
(→ Right-click in an empty field of the tag table → Import file)



- Select the desired symbol table (e.g. in .xlsx format) and confirm the selection with "Open".

(→ SCE\_EN\_020-100\_Tag\_table\_sorting\_station... → Open)

- When the import is finished, you will see a confirmation window and have an opportunity to view the log file for the import. Click → OK.

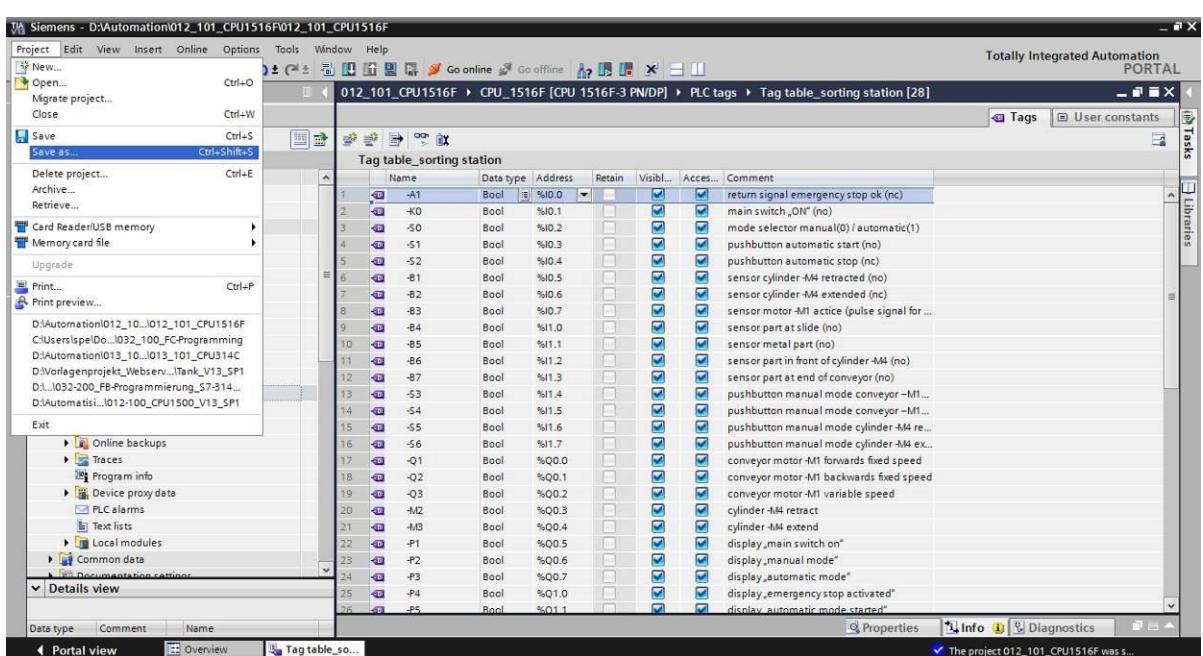


- You can see that some addresses have been highlighted in orange. These are duplicate addresses and the names of the associated tags have been numbered automatically to avoid confusion.
  - Delete the duplicate tags by selecting the lines and pressing the Del key on your keyboard or by selecting "Delete" in the shortcut menu.
- (→ Right-click on selected tags → Delete)

Name	Data type	Address	Retain	Visibl...	Acces...	Comment
1 Q1	Bool	%Q0.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor motor -M1 forwards fixed speed
2 Q2	Bool	%Q0.1		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor motor -M1 backwards fixed speed
3				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	return signal emergency stop ok (nc)
4				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	main switch „ON“ (no)
5				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	mode selector manual(0) / automatic(1)
6				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	pushbutton automatic start (no)
7				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	pushbutton automatic stop (nc)
8				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	sensor cylinder -M4 retracted (no)
9				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	sensor cylinder -M4 extended (nc)
10				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	sensor motor -M1 active (pulse signal for positioning) (no)
11				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	sensor part at slide (no)
12				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	sensor metal part (no)
13				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	sensor part in front of cylinder -M4 (no)
14				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	sensor part at end of conveyor (no)
15				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	pushbutton manual mode conveyor -M1 forwards (no)
16				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	pushbutton manual mode conveyor -M1 backwards (no)
17				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	pushbutton manual mode cylinder -M4 retract (no)
18				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	pushbutton manual mode cylinder -M4 extend (no)
19				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor motor -M1 forwards fixed speed
20				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor motor -M1 backwards fixed speed
21				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor motor -M1 variable speed
22				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	cylinder -M4 retract
23				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	cylinder -M4 extend
24				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	display „main switch on“
25				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	display „manual mode“
26				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	display „automatic mode“

- You now have a complete symbol table of the digital inputs and outputs in front of you.  
 Save your project under the name 032-100\_FCProgramming.

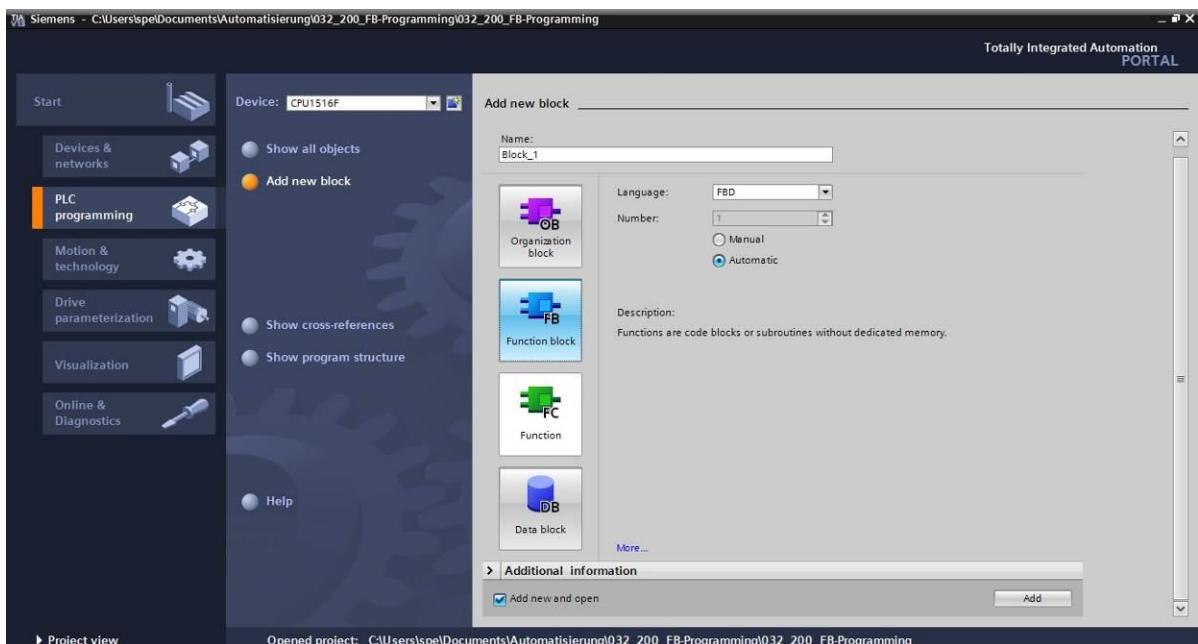
(→ Project → Save as ... → 032-200\_FBProgramming → Save)



## 7.5 Create function block FB1 "MOTOR\_AUTO" for the conveyor motor in automatic mode

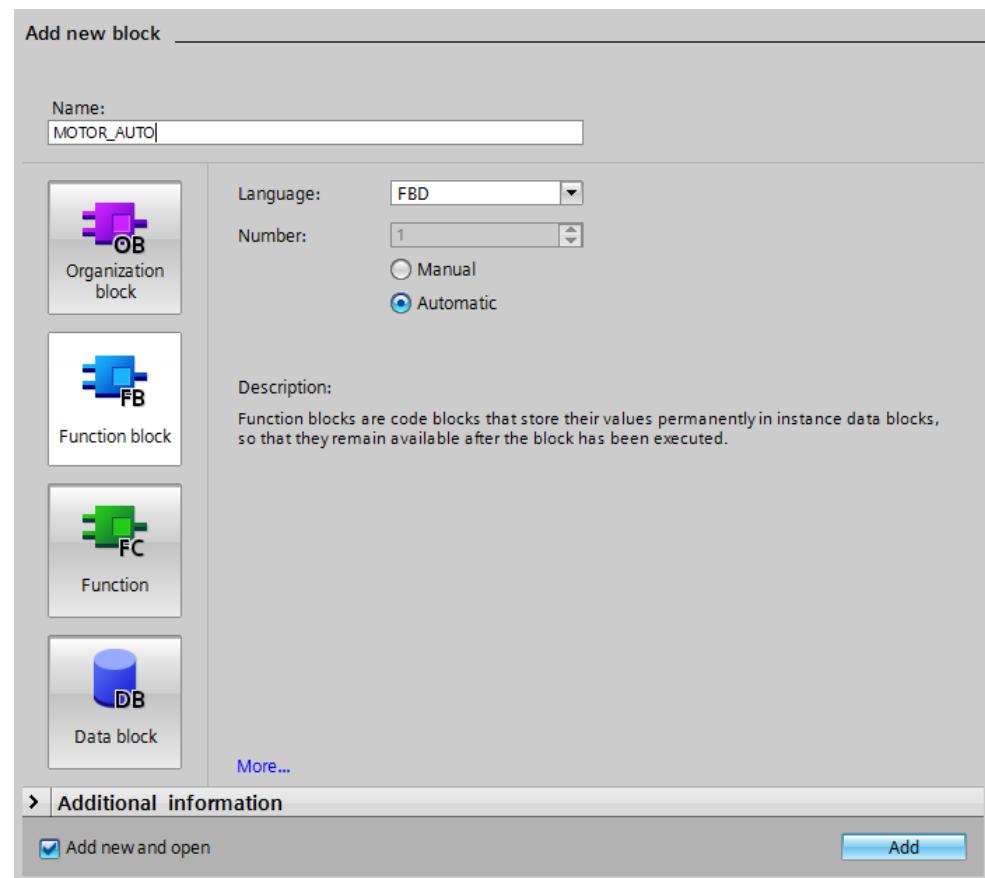
- In the PLC programming section of the portal view, click "Add new block" to create a new function block.

(→ PLC programming → Add new block →  )



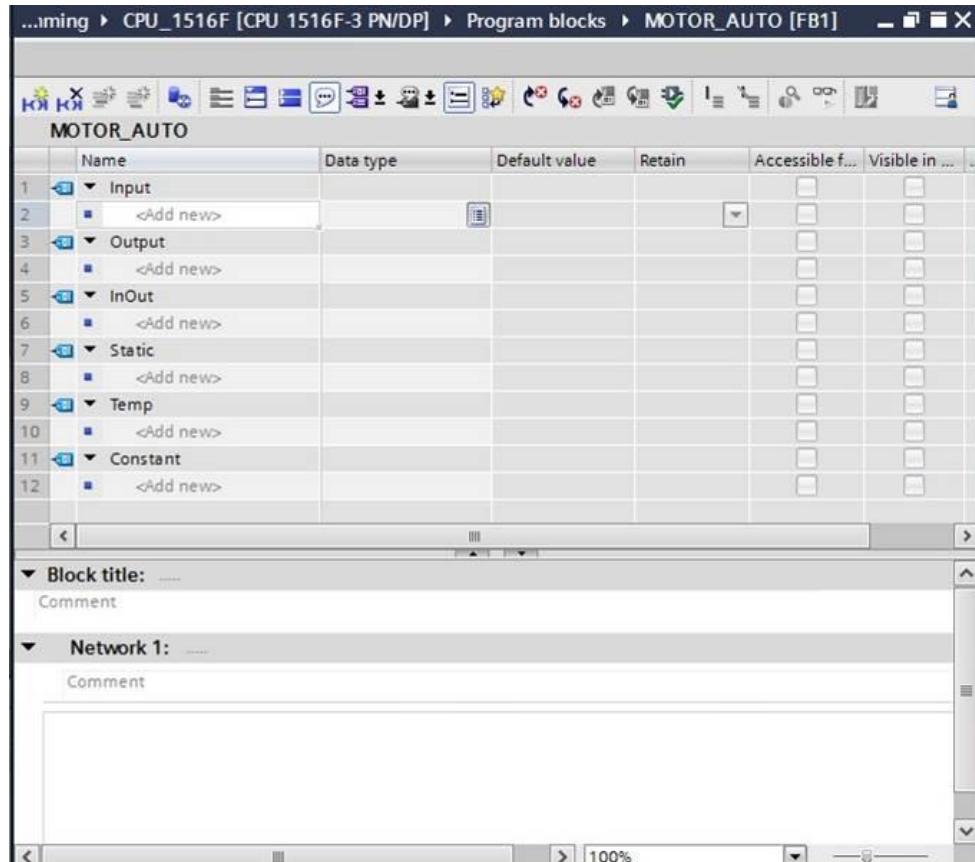
→ Rename your new block to: "MOTOR\_AUTO", set the language to FBD and keep automatic assignment of the number. Select the "Add new and open" check box. You are then taken automatically to your created function block in the project view. Click "Add".

(→ Name: MOTOR\_AUTO → Language: FBD → Number: Automatic →  Add new and open → Add)



## 7.6 Define the interface of FB1 "MOTOR\_AUTO"

- If you selected "Add new and open", the project view opens with a window for creating the block you just added.
- You can find the interface description of your function block in the upper section of your programming view.



- A binary output signal is needed for controlling the conveyor motor. For this reason, we first create local output tag #Conveyor\_motor\_automatic\_mode of the "Bool" type. Enter the comment "Control of the conveyor motor in automatic mode" for the parameter.

(→ Output: Conveyor\_motor\_automatic\_mode → Bool → Control of the conveyor motor in automatic mode)

Name	Data t...	Defaul...	Retain	Acces...	Visibl...	Setp...	Comment
1 Input							
2 <Add new>							
3 Output							
4 Conveyor_motor_automatic_mode	Bool	false	Non-r...	✓	✓		Control of the conveyor motor in automatic mode
5 <Add new>							
6 InOut							
7 <Add new>							
8 Static							
9 <Add new>							
10 Temp							
11 <Add new>							
12 Constant							
13 <Add new>							

- Add parameter #Automatic\_mode\_active as the input interface under Input and confirm the entry with the Enter key or by exiting the entry field. Data type "Bool" is assigned automatically. This will be retained. Next, enter the associated comment "Automatic mode activated".

(→ Automatic\_mode\_active → Bool → Automatic mode activated)

- Add parameters #Start, #Stop, #Enable\_OK and #Safety\_shutoff\_active as additional binary input parameters under Input and check their data types. Add descriptive comments.

Name	Data t...	Defaul...	Retain	Acces...	Visibl...	Setp...	Comment
1 Input							
2 Automatic_mode_active	Bool	false	Non-r...				Automatic mode activated
3 Start	Bool	false	Non-r...				Pushbutton automatic start
4 Stop	Bool	false	Non-r...	✓	✓		Pushbutton automatic stop
5 Enable_OK	Bool	false	Non-r...				All enable conditions OK
6 Safety_shutoff_active	Bool	false	Non-r...				Safety shutoff active e.g. emergency stop operated
7 <Add new>							
8 Output							
9 Conveyor_motor_automatic_mode	Bool	false	Non-r...	✓	✓		Control of the conveyor motor in automatic mode
10 <Add new>							
11 InOut							
12 <Add new>							
13 Static							
14 <Add new>							
15 Temp							
16 <Add new>							

The conveyor is started and stopped with pushbuttons. We therefore need a "Static" tag as a memory. Under Static, add tag #Memory\_automatic\_start\_stop and confirm the entry with the Enter key or by exiting the entry field. Data type "Bool" is assigned automatically. This will be retained. Enter the associated comment "Memory used for start/stop automatic mode". (→ Memory\_automatic\_start\_stop → Bool → Memory used for start/stop automatic mode)

MOTOR_AUTO								
	Name	Data t...	Defaul...	Retain	Acce...	Visibl...	Setp...	Comment
1	Input							
2	Automatic_mode_active	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Automatic mode activated
3	Start	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Pushbutton automatic start
4	Stop	Bool	false	Non-r...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Pushbutton automatic stop
5	Enable_OK	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	All enable conditions OK
6	Safety_shutoff_active	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Safety shutoff active e.g. emergency stop operated
7	<Add new>							
8	Output							
9	Conveyor_motor_automatic_mode	Bool	false	Non-r...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Control of the conveyor motor in automatic mode
10	<Add new>							
11	InOut							
12	<Add new>							
13	Static							
14	Memory_automatic_start_stop	Bool	false	Non-r...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Memory used for start/ stop automatic mode

→ For purposes of program documentation, assign the block title, a block comment and a helpful network title for Network 1.

(→ Block title: Motor control in automatic mode → Network 1:  
Memory\_automatic\_start\_stop and control of the conveyor motor in automatic mode)

MOTOR_AUTO								
	Name	Data t...	Defaul...	Retain	Acce...	Visibl...	Setp...	Comment
1	Input							
2	Automatic_mode_active	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Automatic mode activated
3	Start	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Pushbutton automatic start
4	Stop	Bool	false	Non-r...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Pushbutton automatic stop
5	Enable_OK	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	All enable conditions OK
6	Safety_shutoff_active	Bool	false	Non-r...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Safety shutoff active e.g. emergenc...
7	<Add new>							
8	Output							
9	Conveyor_motor_automatic_mode	Bool	false	Non-r...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Control of the conveyor motor in a...
10	<Add new>							
11	InOut							
12	<Add new>							
13	Static							
14	Memory_automatic_start_stop	Bool	false	Non-r...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Memory used for start/ stop autom...

▼ Block title: Motor control in automatic mode
▼ Conveyor motor in automatic mode:
The bit Memory_automatic_start_stop is set with the input Start, but only if the reset conditions are not fulfilled.
The bit Memory_automatic_start_stop is reset with the input Stop or if the safety shutoff is activated or if the automatic mode is not activated ( manual mode).
If Memory_automatic_start_stop is set, the enable conditions are granted and Memory_conveyor_start_stop is set the output Conveyor_motor_automatic_mode is activated.
For reasons of energy efficiency the conveyor motor should only run if a part is present.
Therefore Memory_conveyor_start_stop is set if there is a part detected in front of Sensor_slide and reset with a negative edge at Sensor_end_of_conveyor or if the safety shutoff is activated or if the automatic mode is not activated ( manual mode).
▼ Network 1: Memory automatic_start_stop and control of the conveyor motor in automatic mode

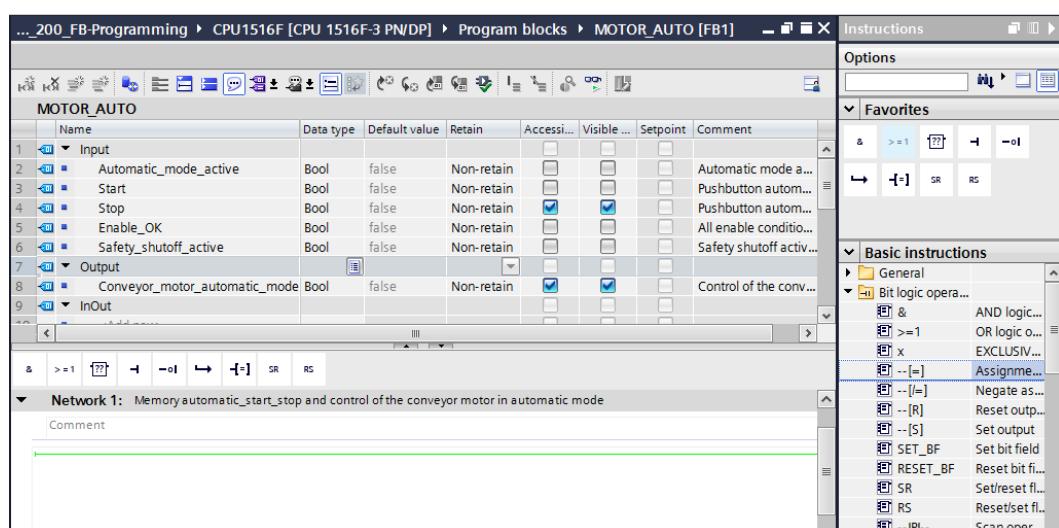
## 7.7 Program FB1: MOTOR\_AUTO

→ Below the interface description, you see a toolbar in the programming window with various logic functions and below that an area with networks. We have already specified the block title and the title for the first network there. Programming is performed within the networks using individual logic blocks. Distribution among multiple networks helps to preserve the clarity of the program. In the following, you will get to know the various ways you can insert logic blocks.



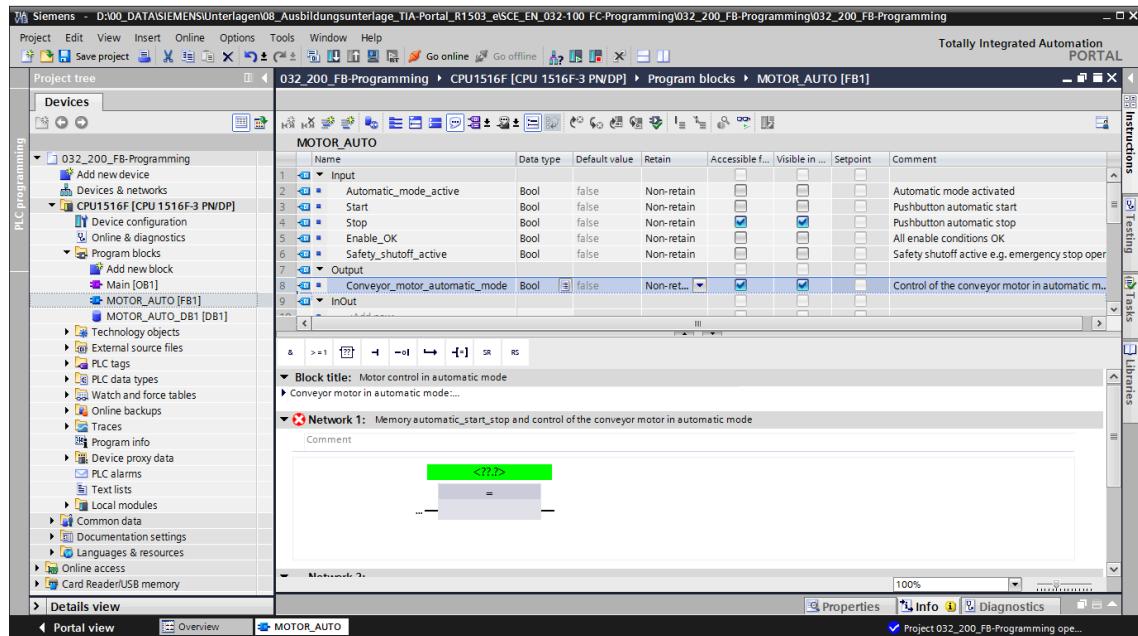
→ You can see a list of instructions you can use in the program on the right side of your programming window. Under → Basic instructions → Bit logic operations, find function (Assignment) and use a drag-and-drop operation to move it to Network 1 (green line appears, mouse pointer with + symbol).

(→ Instructions → Basic instructions → Bit logic operations → )



- Now use drag-and-drop to move your output parameter  
`#Conveyor_motor_automatic_mode` onto `<??>` above the block you just inserted. The best way to select a parameter in the interface description is by "grabbing" it at the blue symbol .

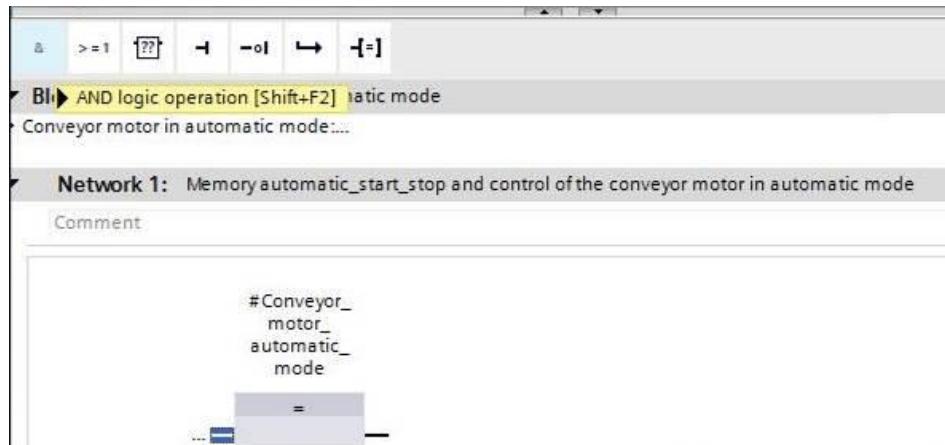
(→  `Conveyor_motor_automatic_mode`)



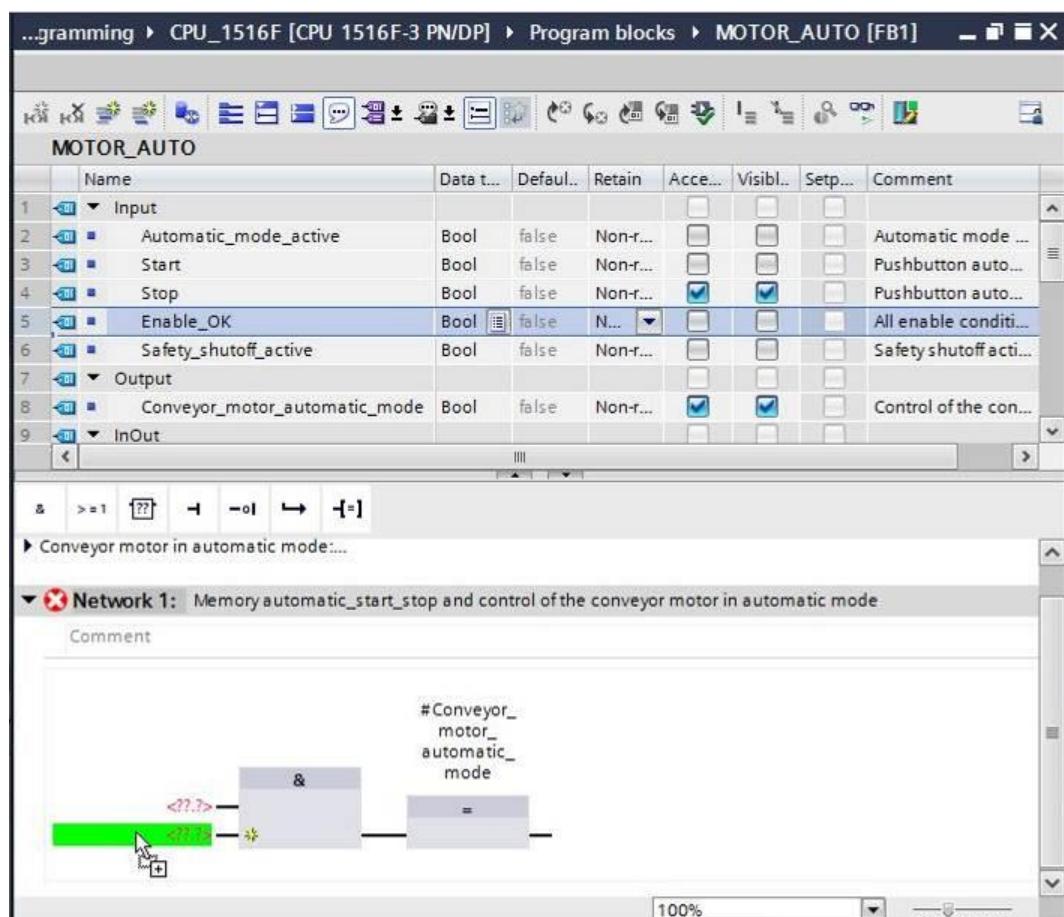
- This determines that the `#Conveyor_motor_automatic_mode` parameter is written by this block. Still missing, however, are the input conditions so that this actually happens. An SR flip-flop and `#Enable_OK` parameter are logically combined with an AND logic operation at the input of the assignment block. To do this, first click the input of the block so that the input line has a blue background.



- Click the  icon in your logic toolbar to insert an AND logic operation before your assignment block.

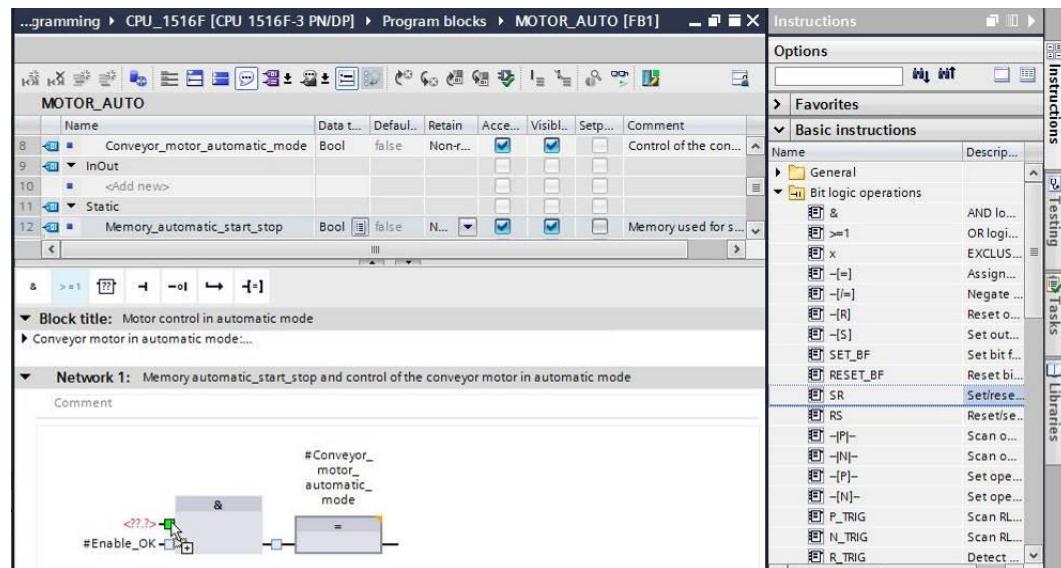


- Use drag-and-drop to move input parameter #Enable\_OK onto the second input of the & logic operation <???.>. (→  Enable\_OK)

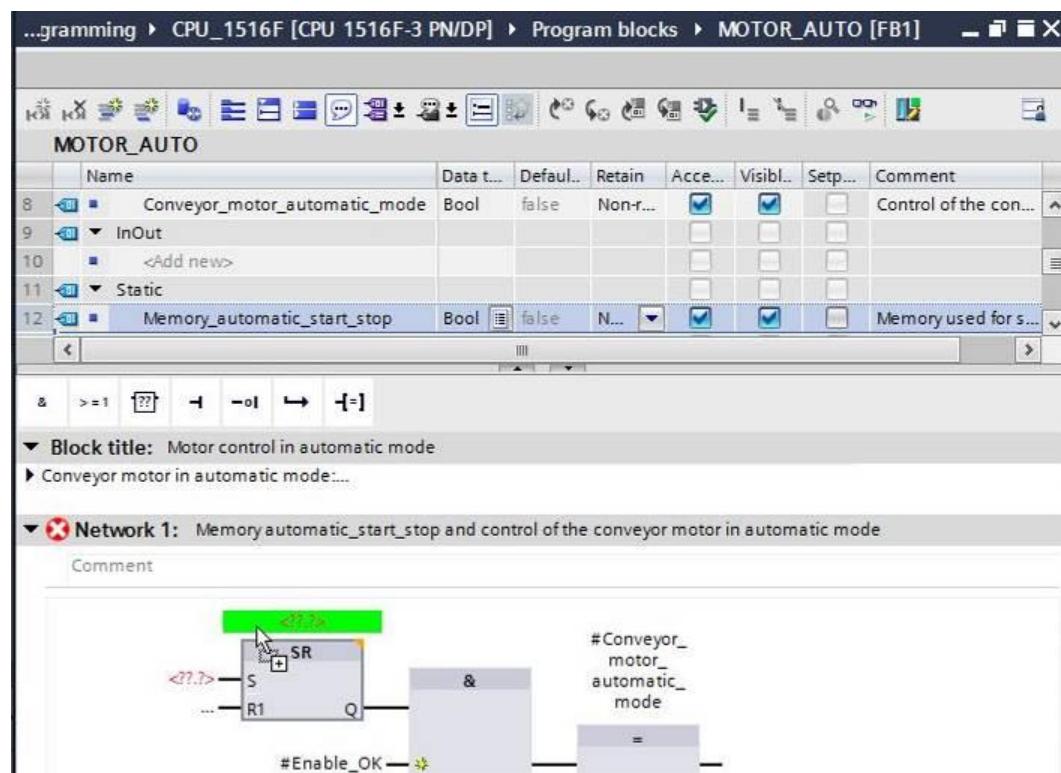


- Use drag-and-drop to move the Set/reset flip-flop function from the list of instructions under → Basic instructions → Bit logic operations onto the first input of the & operation .

(→ Instructions → Basic instructions → Bit logic operations → →

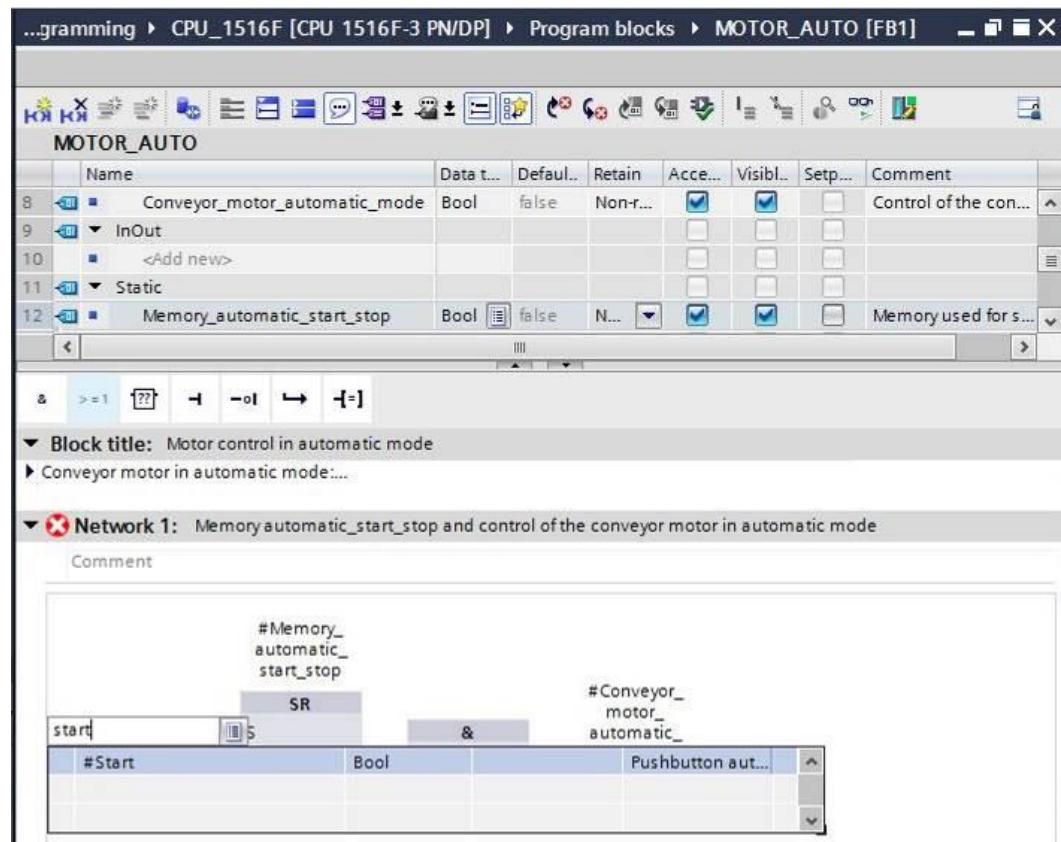


- The SR flip-flop requires a memory tag. For this, use drag-and-drop to move static parameter #Memory\_automatic\_start\_stop onto the above the SR flip-flop. (→ Memory\_automatic\_start\_stop)



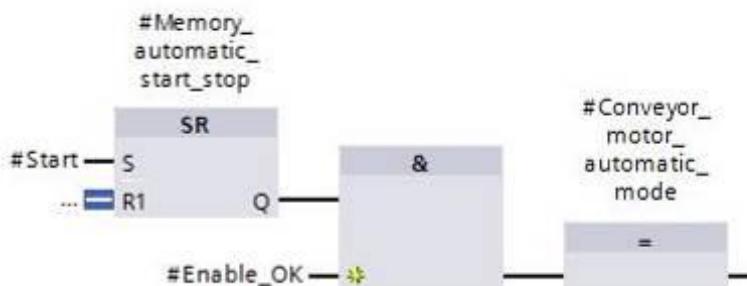
- The #Memory\_automatic\_start\_stop will be set with input tag #Start. Click twice on the S input of the SR flip-flop <???.?> and enter "Start" in the field that appears in order to see a list of available tags starting with "Start". Click the #Start tag and apply with → Enter.

(→ SR flip-flop → <???.?> → Start → #Start → Enter)

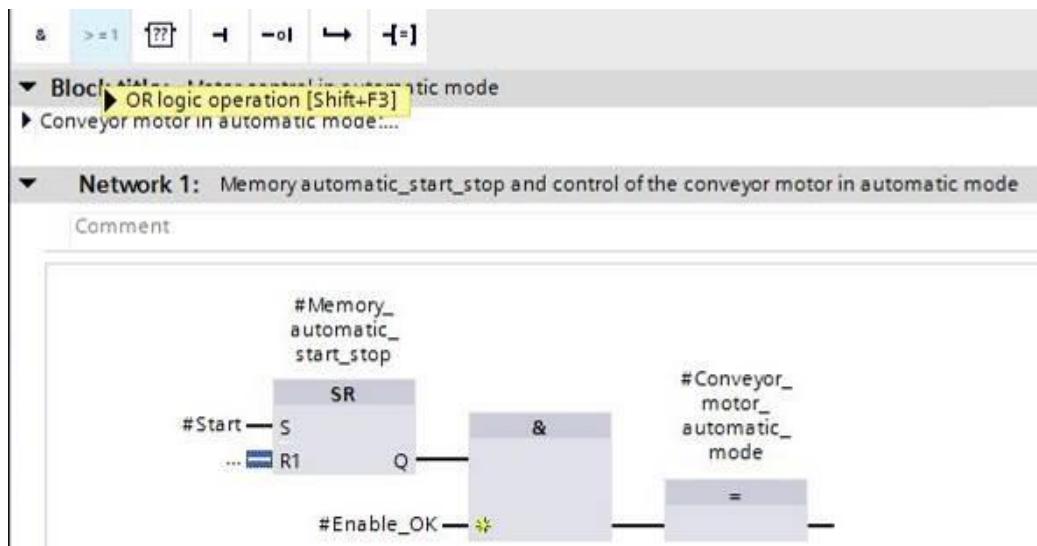


**Note:** When assigning tags in this way, there is a risk of a mix-up with the global tags from the tag table. The previously presented procedure using drag and drop from the interface description should therefore be used preferentially.

- Multiple conditions are to be able to stop the conveyor. An OR block is therefore needed at the R1 input of the SR flip-flop. First, click the R1 input of the SR flip-flop so that the input line has a blue background.



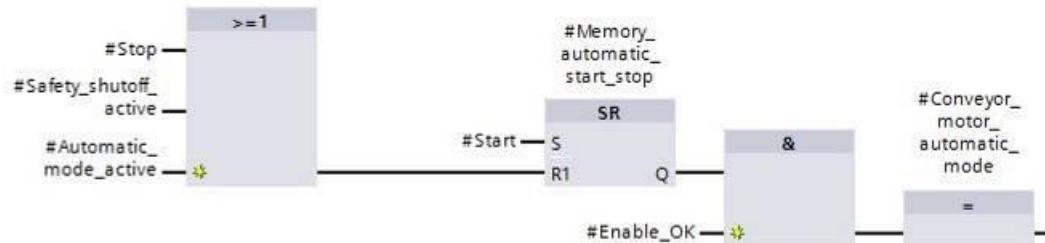
- Click the  $>=1$  icon in your logic toolbar to insert an OR logic operation.



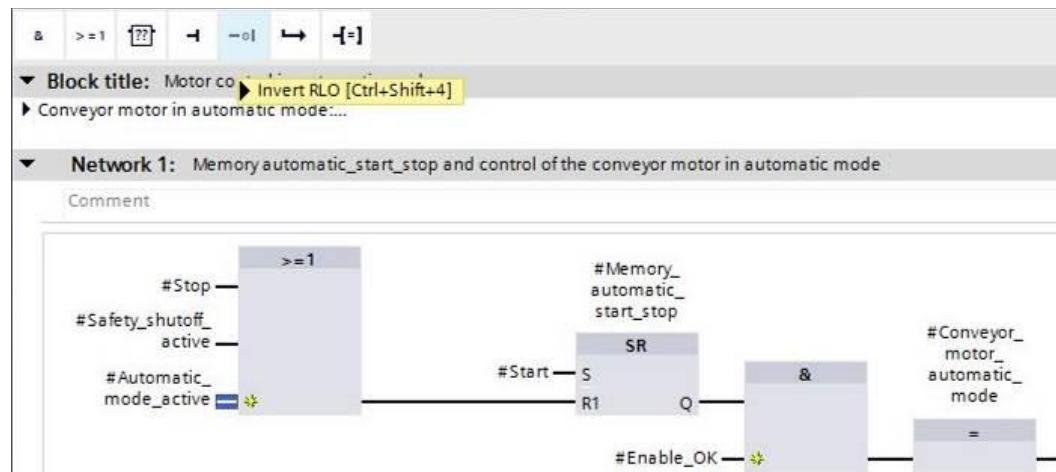
- The OR block has 2 inputs initially. In order to logically combine an additional input tag, click the yellow star of the OR block.



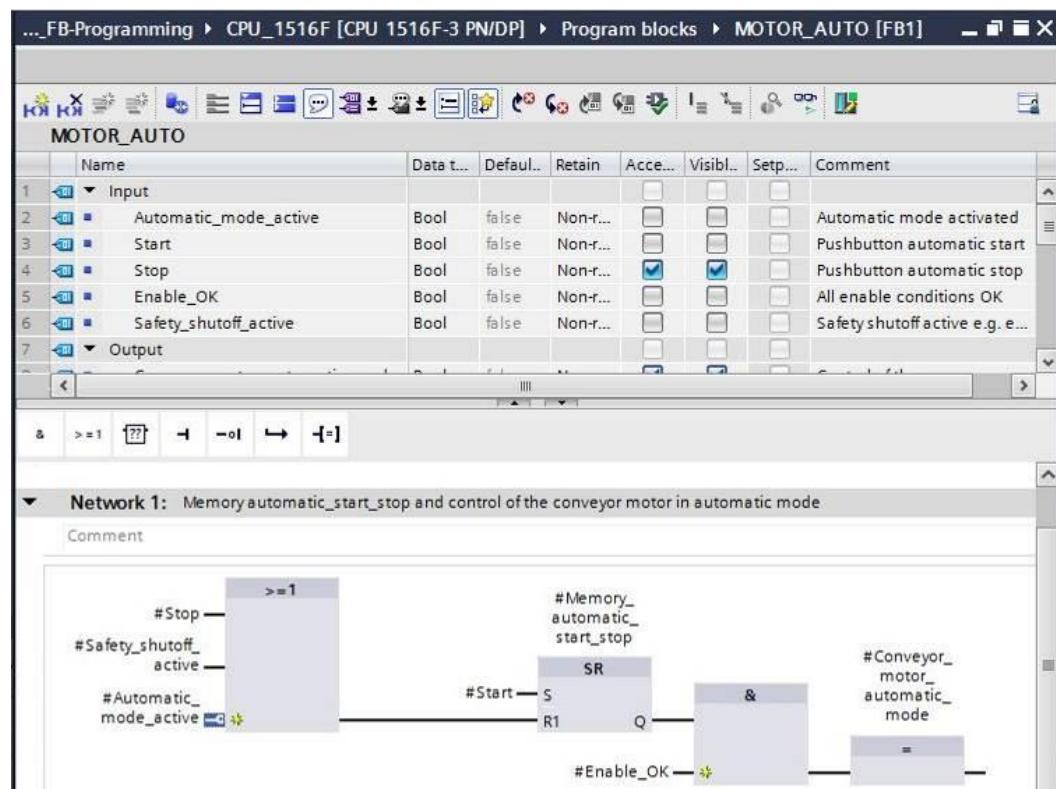
- Add input tags #Stop, #Safety\_shutoff\_active and #Automatic\_mode\_active to the 3 inputs of the OR block.



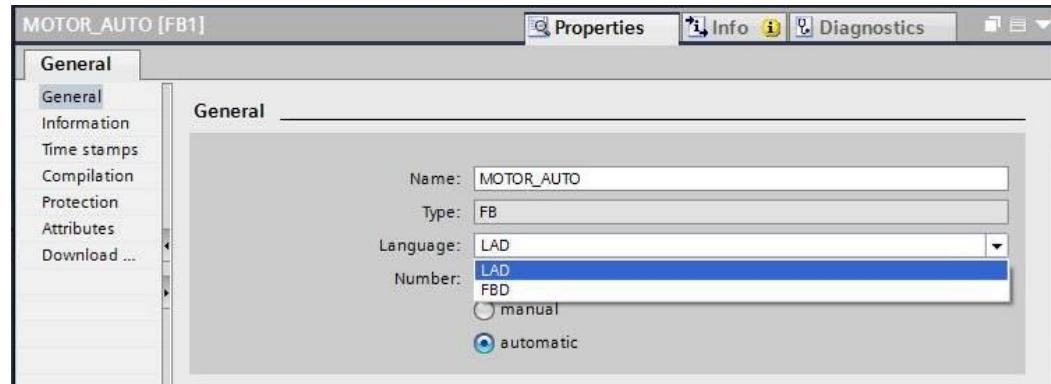
- Negate the input connected to parameter #Automatic\_mode\_active by selecting it and clicking .



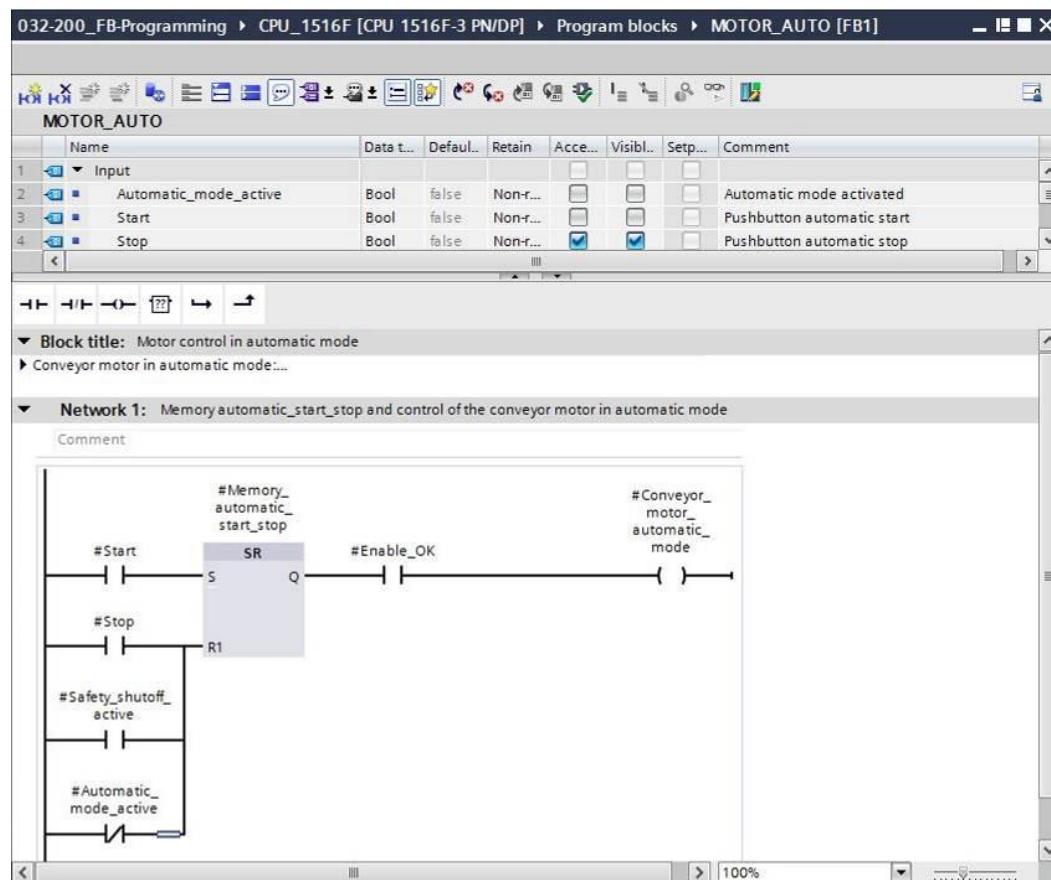
- Do not forget to click  Save project. The finished function block "MOTOR\_AUTO" [FB1] in FBD is shown below.



- Under "General" in the properties of the block, you can change the "Language" to LAD (Ladder Logic) (→Properties → General → Language: LAD)

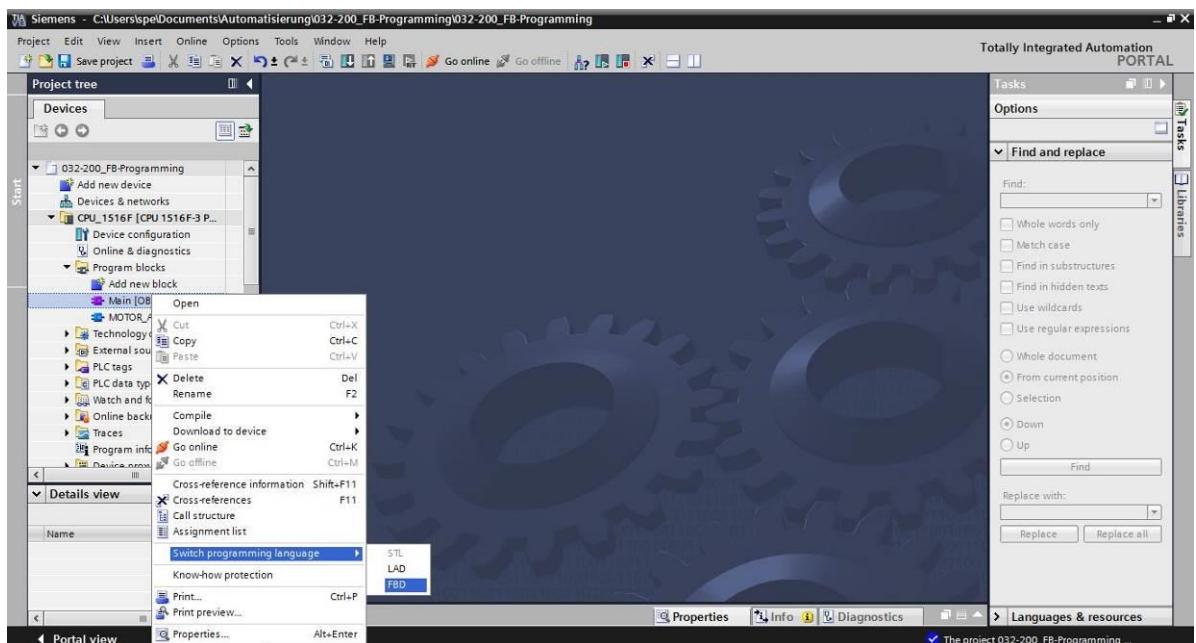


- The program has the following appearance in LAD.

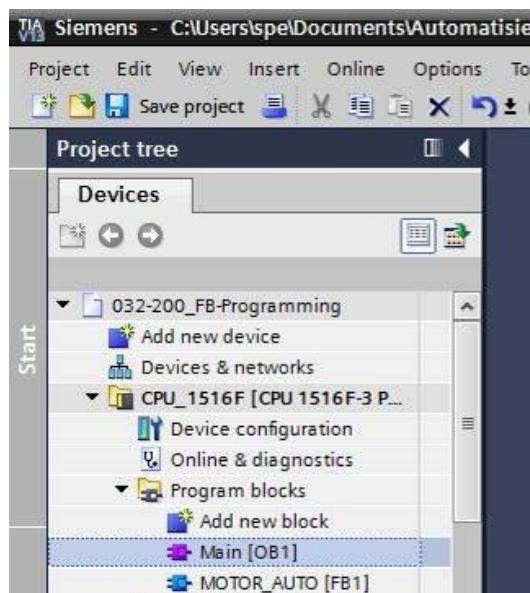


## 7.8 Program organization block OB1 – Control of the forward belt tracking in automatic mode

- Before programming organization block "Main [OB1]", we switch the programming language to FBD (Function Block Diagram). To do so, first click on "Main [OB1]" in the "Program blocks" folder.
- (→ CPU\_1516F[CPU 1516F-3 PN/DP → Program blocks → Main [OB1] → Switch programming language → FBD)

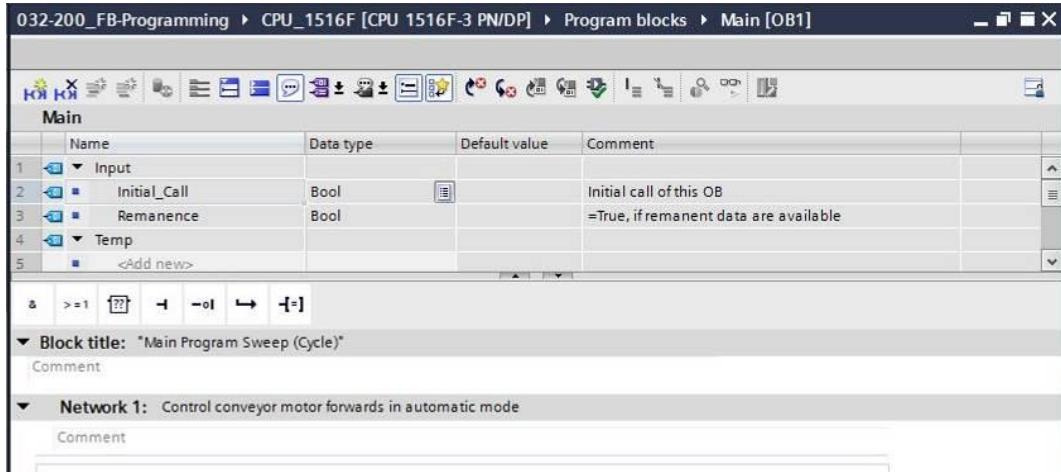


- Open the "Main [OB1]" organization block with a double-click.

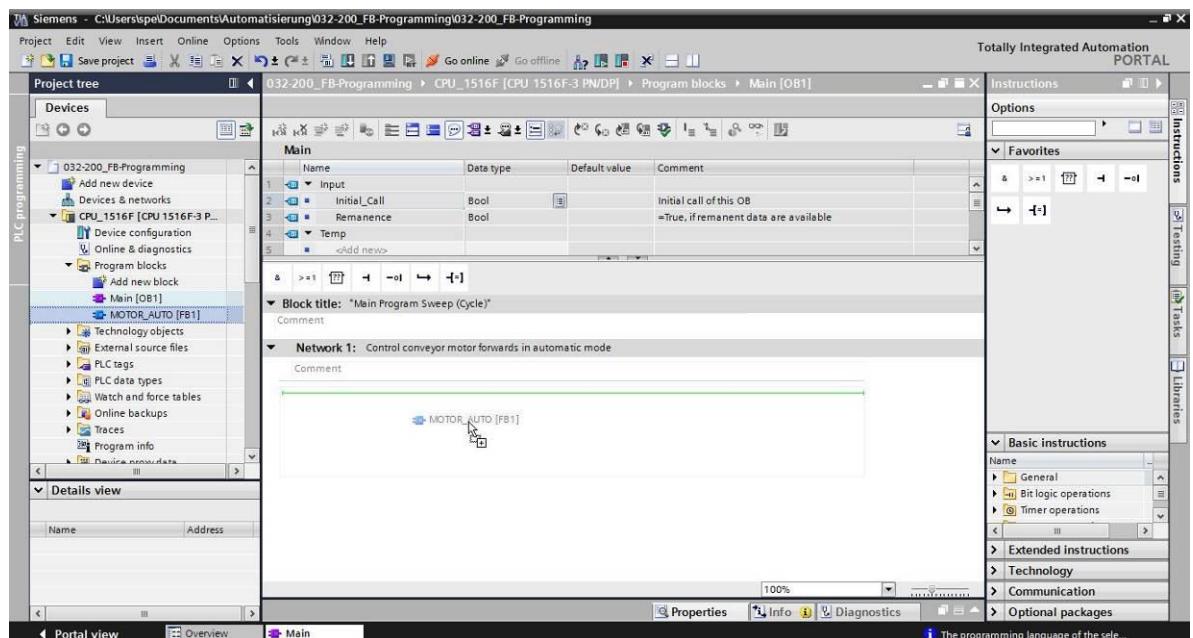


→ Assign Network 1 the name "Control conveyor tracking forward in automatic mode"

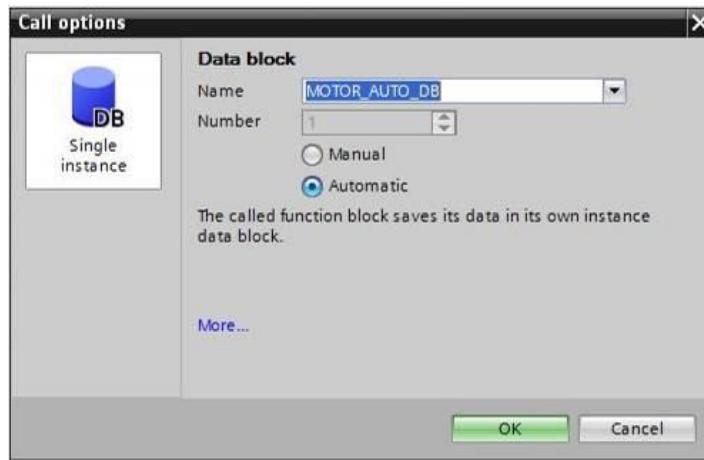
(→ Network 1:... → Control conveyor motor forwards in automatic mode)



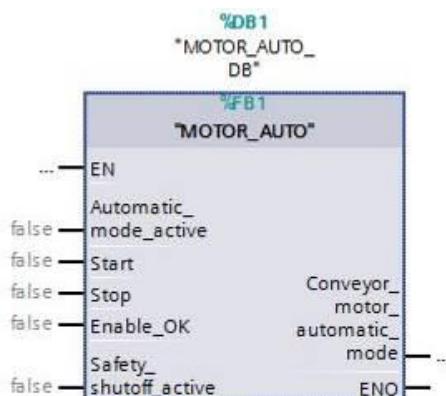
→ Use drag-and-drop to move your "MOTOR\_AUTO [FB1]" function block onto the green line in Network 1.



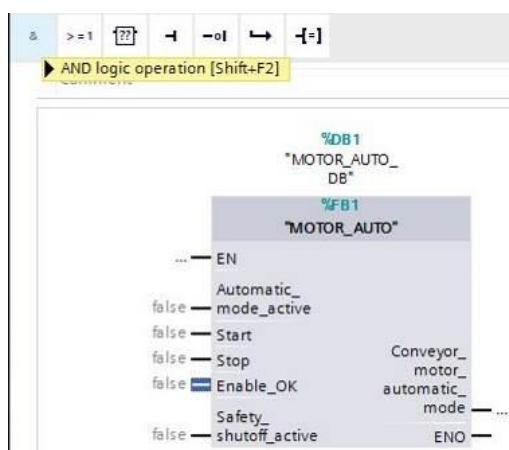
- The instance data block for this call of FB1 is created automatically. Assign a name and apply it with OK. (→ MOTOR\_AUTO\_DB1 → OK)



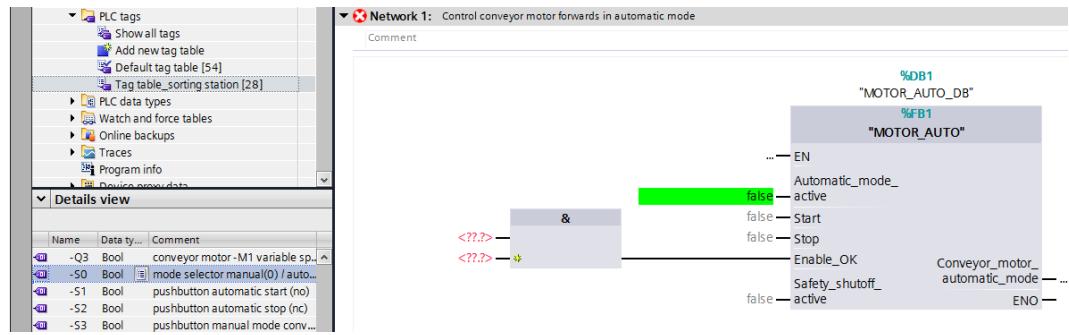
- A block with the interface you defined, the instance data block and connections EN and ENO are inserted in Network 1.



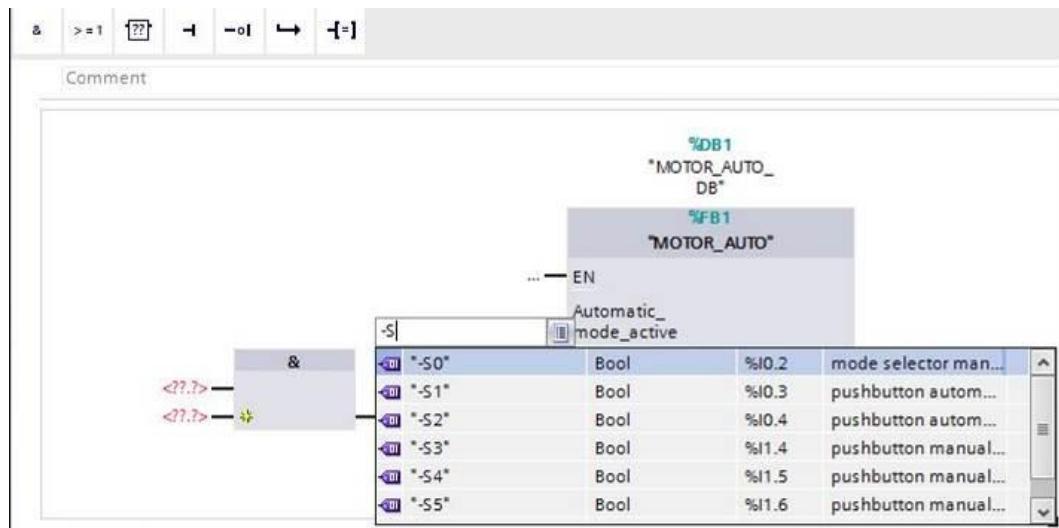
- To insert an AND before input parameter "Enable\_OK", select this input and insert the AND by clicking the icon in your logic toolbar (→ ).



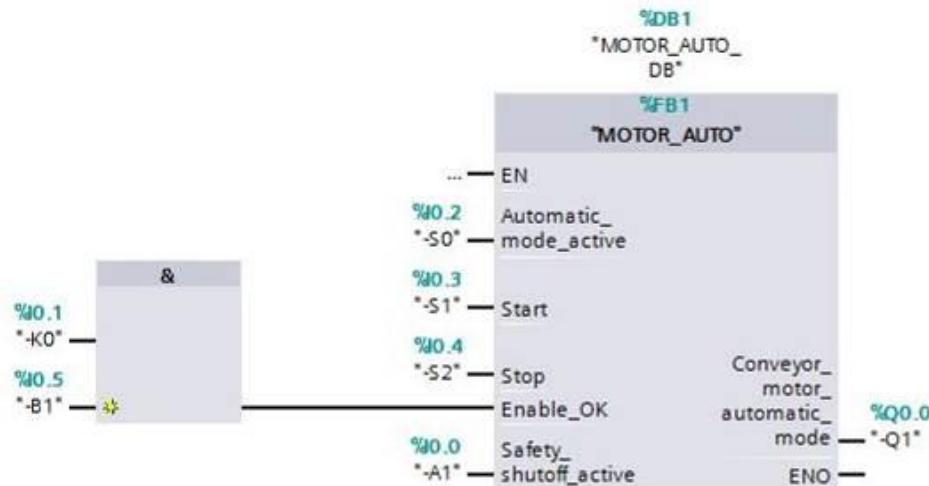
- To connect the block to the global tags from "Tag\_table\_sorting\_station", we have two options:
- Either select the "Tag\_table\_sorting\_station" in the project tree and use drag-and-drop to move the desired global tag from the Details view to the interface of FC1 ( → Tag\_table\_sorting\_station → Details view. → S0 → Automatic\_mode\_active)



- Or, enter the starting letters (e.g. "S") “-S” of the desired global tag for <???.?> and select the global input tag "-S0" (%I0.2) from the displayed list. (→ Automatic\_mode\_active → S → -S0)

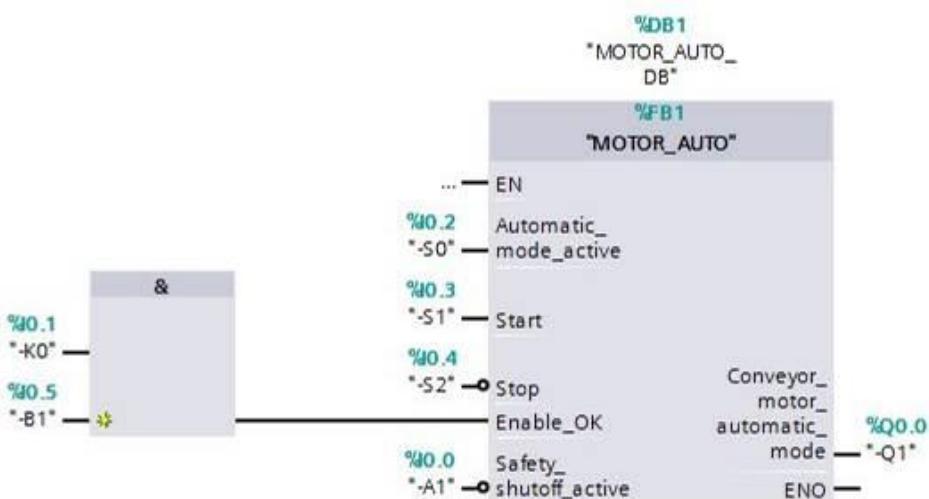


- Insert the other input tags "-S1", "-S2", "-K0", "-B1" and "-A1" and insert output tag "-Q1" (%Q0.0) at output "Conveyor\_motor\_automatic\_mode".

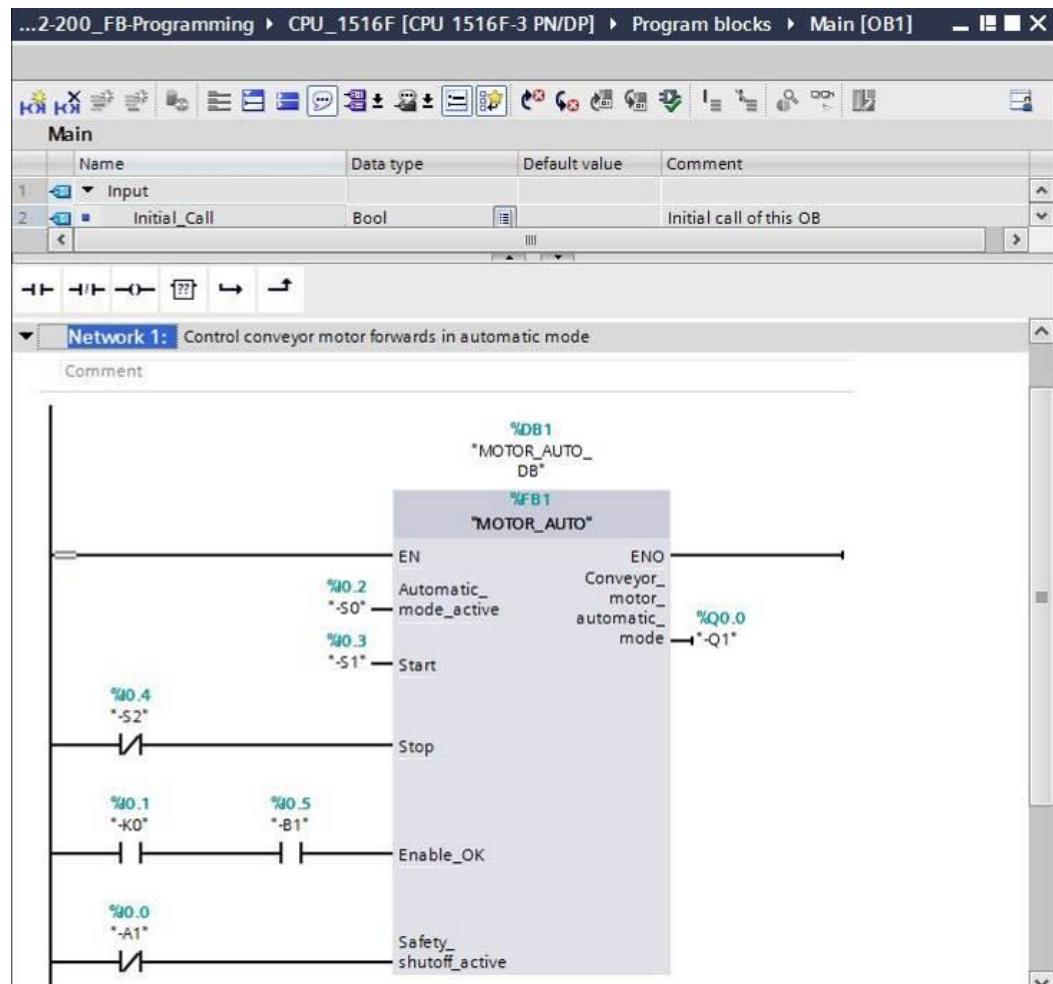


- Negate the querying of input tags "-S2" and "-A1" by selecting them and clicking

( $\rightarrow -S2 \rightarrow \boxed{-\text{OI}} \rightarrow -A1 \rightarrow \boxed{-\text{OI}})$

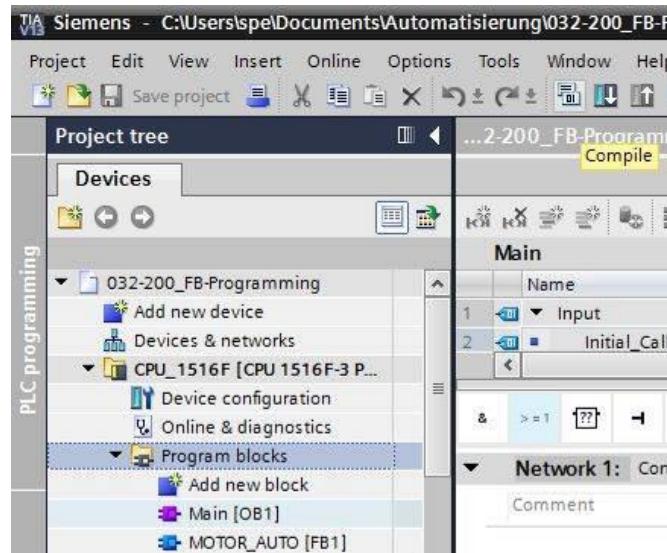


## 7.9 The result in the LAD (Ladder Logic) programming language has the following appearance.

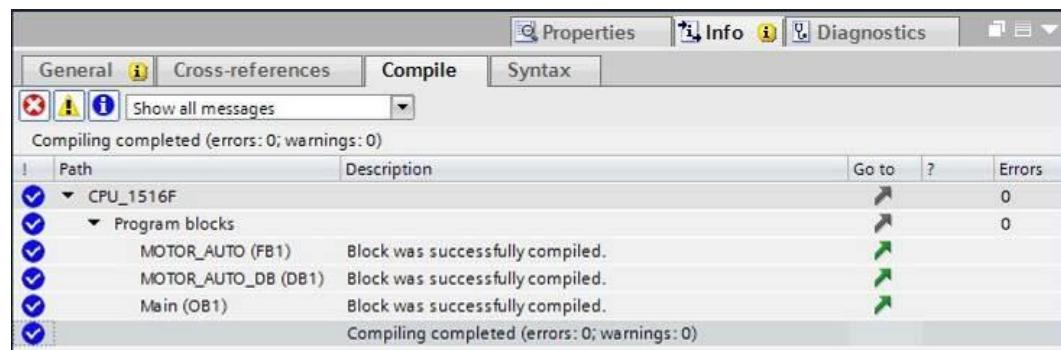


## 7.10 Save and compile the program

- To save your project, select the **Save project** button in the menu. To compile all blocks, click the "Program blocks" folder and select the icon for compiling in the menu  
 (→ **Save project** → **Program blocks** → ).

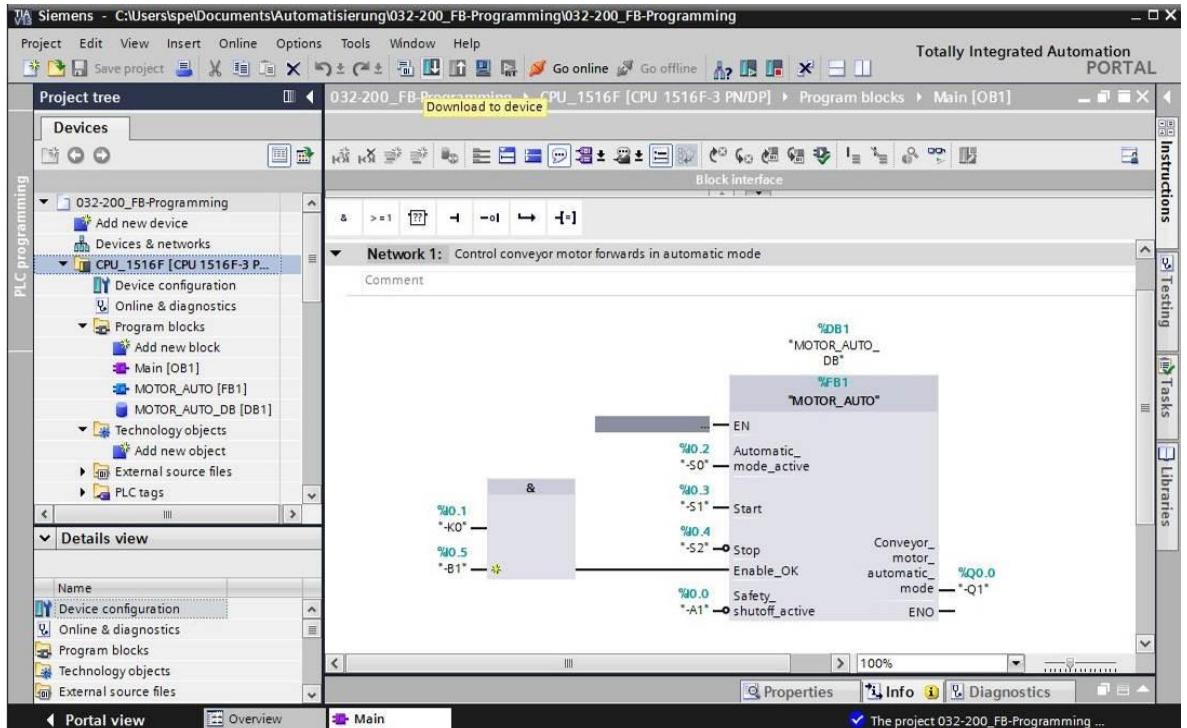


- The "Info", "Compile" area shows which blocks were successfully compiled.



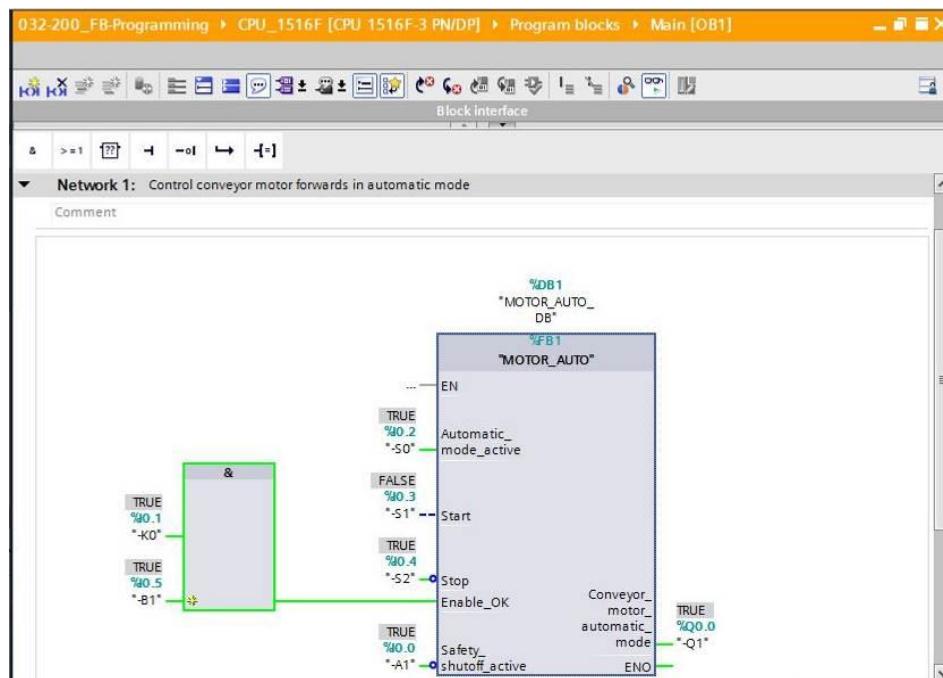
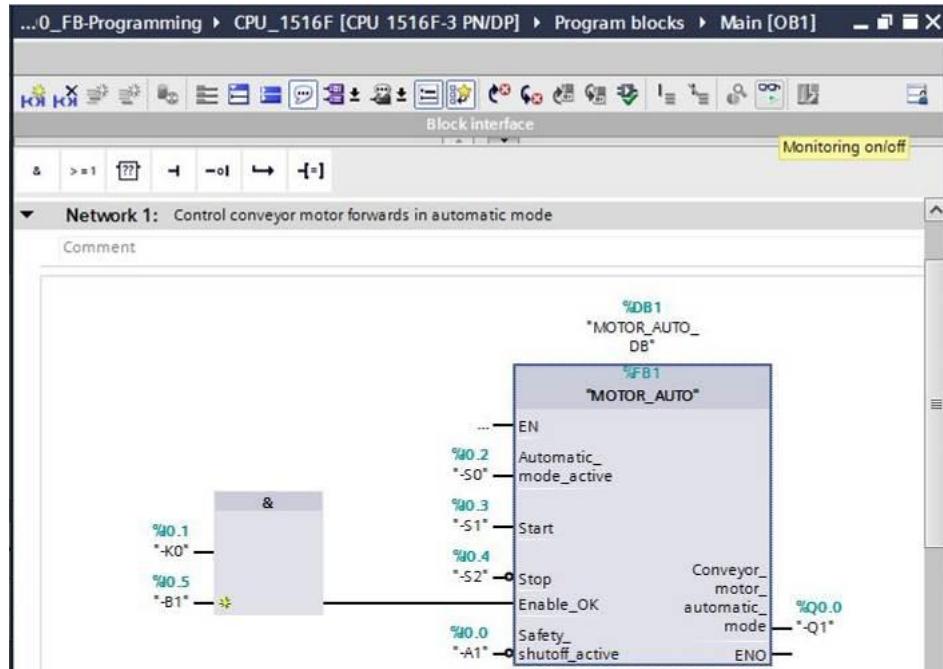
## 7.11 Download the program

- After successful compilation, the complete controller with the created program, as previously described in the modules for hardware configuration, can be downloaded



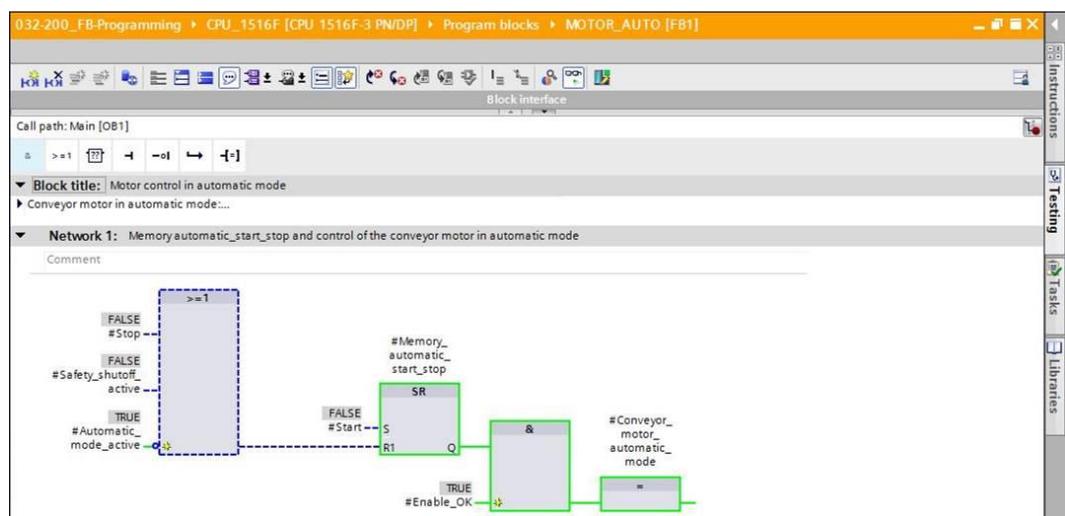
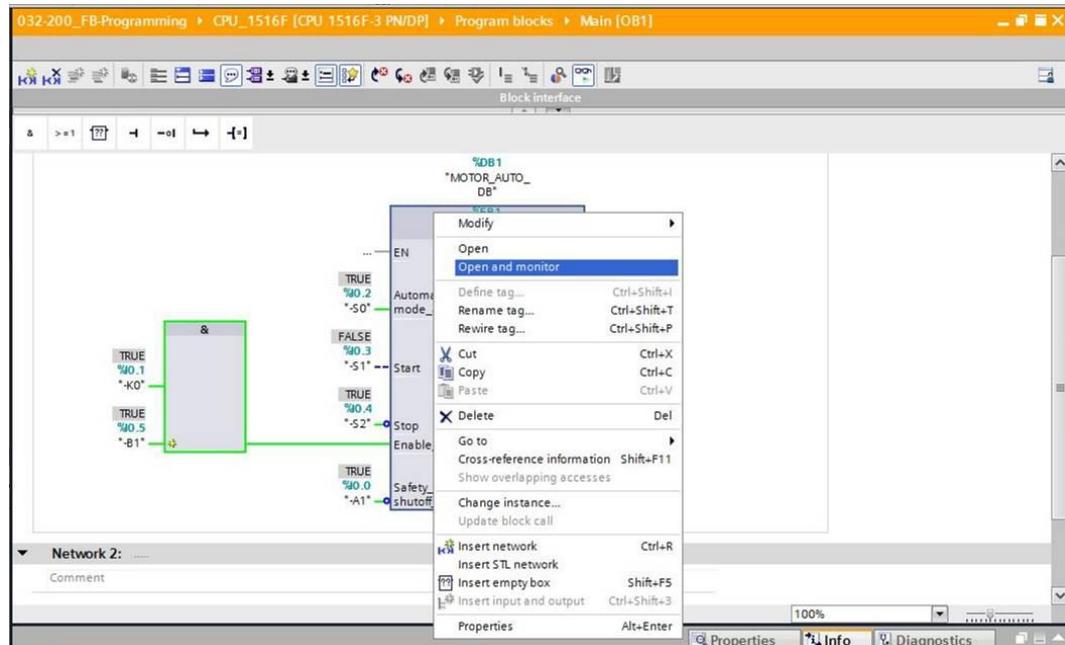
## 7.12 Monitor program blocks

- The desired block must be open for monitoring the downloaded program. The monitoring can be activated/deactivated by clicking the icon. (→ Main [OB1] →



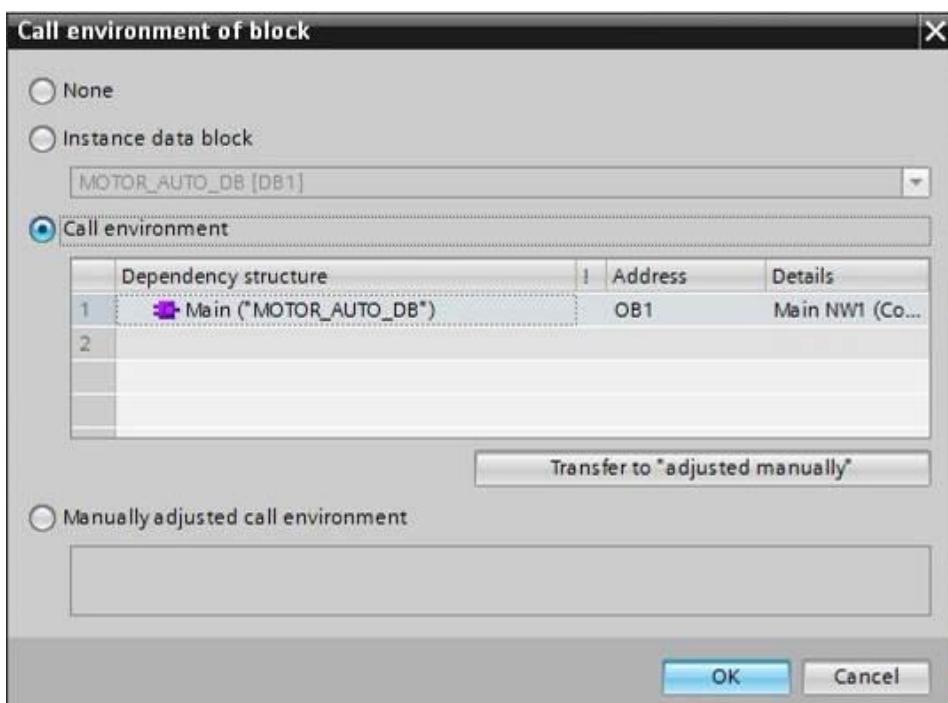
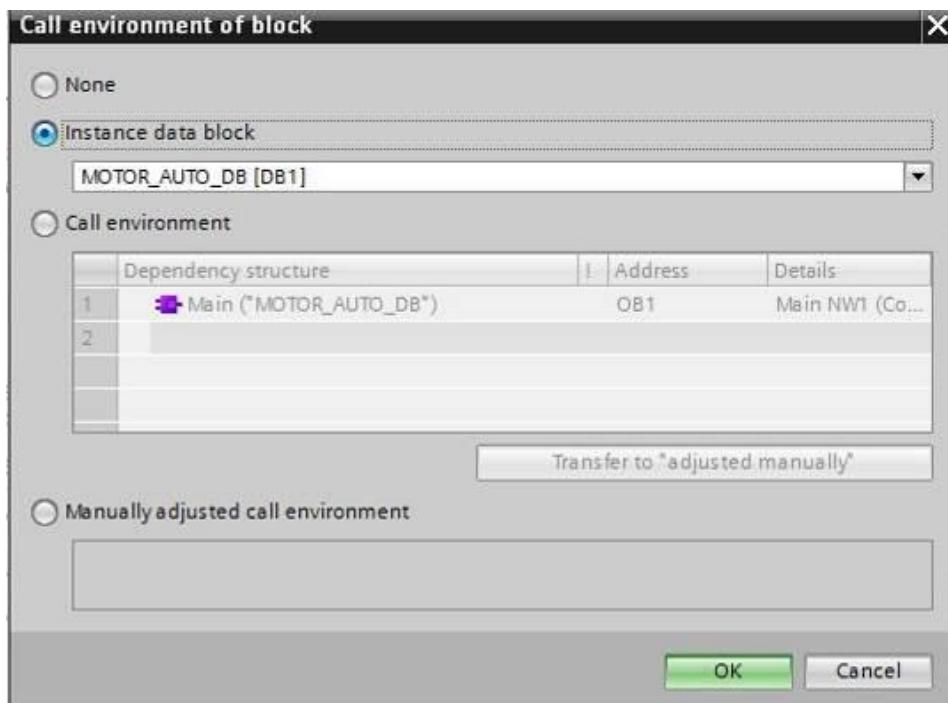
**Note:** The monitoring here is signal-related and controller-dependent. The signal states at the terminals are indicated with TRUE or FALSE.

- The "MOTOR\_AUTO" [FB1] function block called in the "Main [OB1]" organization block can be selected directly for "Open and monitor" after right-clicking (→ "MOTOR\_AUTO" [FB1] → Open and monitor).



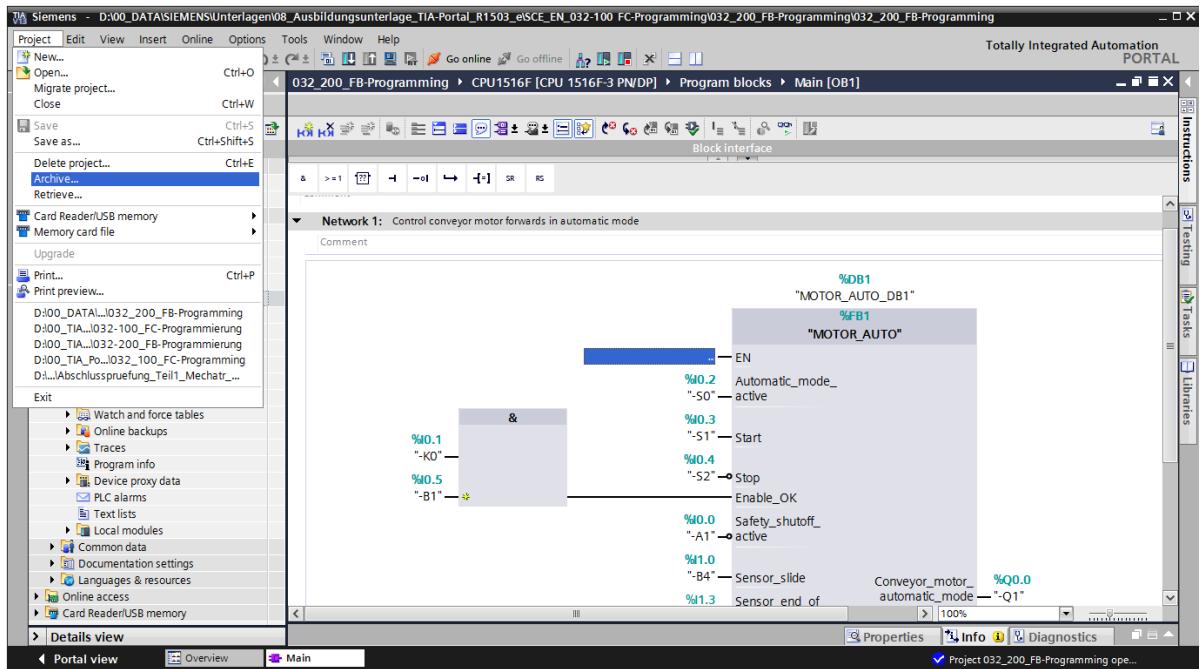
**Note:** The monitoring here is function-related and controller-independent. The actuation of sensors and the station status are shown here with TRUE or FALSE.

→ If a particular point of use of a "MOTOR\_AUTO" [FB1] function block that is called multiple times is to be monitored, this can be performed using the  icon. There are two alternatives available for specifying the call environment: using the call environment or the instance data block. (→  → Instance data block → MOTOR\_AUTO\_DB1 [DB1] → Call environment → Address: OB1 → Details: Main NW1 → OK).



## 7.13 Archive the project

- As the final step, we want to archive the complete project. Select the → "Archive ..." command in the → "Project" menu. Select a folder where you want to archive your project and save it with the file type "TIA Portal project archive". (→ Project → Archive → TIA Portal project archive → 032-200\_FBProgramming.... → Save)



## 8 Checklist

No.	Description	Completed
1	Compiling successful and without error message	
2	Download successful and without error message	
3	Switch on station (-K0 = 1) Cylinder retracted / Feedback activated (-B1 = 1) EMERGENCY OFF (-A1 = 1) not activated AUTOMATIC mode (-S0 = 1) Pushbutton automatic stop not actuated (-S2 = 1) Briefly press the automatic start pushbutton (-S1 = 1) then conveyor motor forwards fixed speed (-Q1 = 1) switches on and stays on.	
4	Briefly press the automatic stop pushbutton (-S2 = 0) → Q1 = 0	
5	Activate EMERGENCY OFF (-A1 = 0) → Q1 = 0	
6	Manual mode (-S0 = 0) → Q1 = 0	
7	Switch off station (-K0 = 0) → Q1 = 0	
8	Cylinder not retracted (-B1 = 0) → Q1 = 0	
9	Project successfully archived	

## 9 Exercise

### 9.1 Task – Exercise

In this exercise, an energy saving function is to be added to the MOTOR\_AUTO [FB1] function block. The expanded function block will be planned, programmed and tested:

To save energy, the conveyor should only run when a part is present.

The Conveyor\_motor\_automatic\_mode output is therefore only activated when Memory\_automatic\_start\_stop is set, the enable conditions are met and Memory\_conveyor\_start\_stop is set.

The Memory\_conveyor\_start\_stop is set when Sensor\_chute\_occupied signals a part and is reset when Sensor\_end\_of\_conveyor produces a negative edge or safety shutoff is active or automatic mode is not activated (manual mode).

### 9.2 Planning

Plan the implementation of the task on your own.

**Note:** Learn about the use of the negative edge in SIMATIC S7-1500 in the online help.

### 9.3 Checklist – Exercise

No.	Description	Completed
1	Compiling successful and without error message	
2	Download successful and without error message	
3	Switch on station (-K0 = 1) Cylinder retracted / Feedback activated (-B1 = 1) EMERGENCY OFF (-A1 = 1) not activated AUTOMATIC mode (-S0 = 1) Pushbutton automatic stop not actuated (-S2 = 1) Briefly press the automatic start pushbutton (-S1 = 1) Sensor at chute activated (-B4 = 1) then conveyor motor forwards fixed speed (-Q1 = 1) switches on and stays on.	
4	Sensor at end of conveyor activated (-B7 = 1) → -Q1 = 0	
5	Briefly press the automatic stop pushbutton (-S2 = 0) → -Q1 = 0	
6	Activate EMERGENCY OFF (-A1 = 0) → -Q1 = 0	
7	Manual mode (-S0 = 0) → -Q1 = 0	
8	Switch off station (-K0 = 0) → -Q1 = 0	
9	Cylinder not retracted (-B1 = 0) → -Q1 = 0	
10	Project successfully archived	

## 10 Additional information

You can find additional information as an orientation aid for initial and advanced training, for example: Getting Started, videos, tutorials, apps, manuals, programming guidelines and trial software/firmware, at the following link:

[www.siemens.com/sce/s7-1500](http://www.siemens.com/sce/s7-1500)