SIEMENS

Learn-/Training Document

Siemens Automation Cooperates with Education (SCE) | Version V15 and higher

TIA Portal module 034-100
Basics of FC Programming
with SIMATIC IOT2000EDU

siemens.com/sce

SIEMENS
Global Industry Partner of WorldSkills International

worldskills

Matching SCE Trainer Packages for these Learn/Training Document

- **SIMATIC IOT2020 with Intel Quark x1000, 512 MB RAM**
  Order No.: 124-4037 Can be ordered from RS Components rs-components.com
- **SIMATIC IOT2040 with Intel Quark x1020 (+Secure Boot), 1 GB RAM**
  Order No.: 6ES7647-0AA00-1YA2
- **SIMATIC IOT2000EDU Software Controller executable on IOT2020 and IOT2040**
  Order No.: 6ES7671-0LE00-0YB0
- **SIMATIC IO-Shield: SIMATIC IOT2000 Input/Output Module with 5 DI, 2 DO, 2 AI, ARDUINO Shield for IOT2020/2040**
  Order No.: 6ES7647-0KA01-0AA2
- **3rd Party IO-Shield: IKHDS Power Shield for IOT2020/2040 with 6 DI, 5 DO (RA), 1 DO (PWM), 2 AI, 1 AO**
  Order No.: 100301 Can be ordered from KAFTAN media UG kaftan-media.com/iot2000

SIMATIC STEP 7 Software for Training

- **SIMATIC STEP 7 Professional V15 - Single license**
  Order no.: 6ES7822-1AA05-4YA5
- **SIMATIC STEP 7 Professional V15 - Classroom license (up to 6 users)**
  Order no.: 6ES7822-1BA05-4YA5
- **SIMATIC STEP 7 Professional V15 - Upgrade license (up to 6 users)**
  Order no.: 6ES7822-1AA05-4YE5
- **SIMATIC STEP 7 Professional V15 - Student license (up to 20 users)**
  Order no.: 6ES7822-1AC05-4YA5

Continued education

For regional Siemens SCE continued education, get in touch with your regional SCE contact:
siemens.com/sce/contact

Additional information regarding SCE

siemens.com/sce

Information regarding use

The SCE Learn-/Training Document 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 facilities 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, which 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 Learn-/Training Document 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 person: Roland Scheuerer roland.scheuerer@siemens.com.

We would like to thank Michael Dziallas Engineering and all those involved for their support in creating this SCE Learn-/Training Document.

Table of contents

Basics of FC 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, we introduce ***library-compatible*** function and function block programming. You will get to know the ***Function Block Diagram (FBD)*** programming language and use it to program a function (FC1) and an organization block (OB1).

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

2 Requirement

This chapter builds on the SIMATIC IOT2000 hardware configuration. The task can be realized with any shields that have a corresponding number of digital inputs and outputs. You can use the following project for this chapter, for example:

SCE_EN_014-101_Hardware_Configuration_IOT2000.zap14

3 Required hardware and software

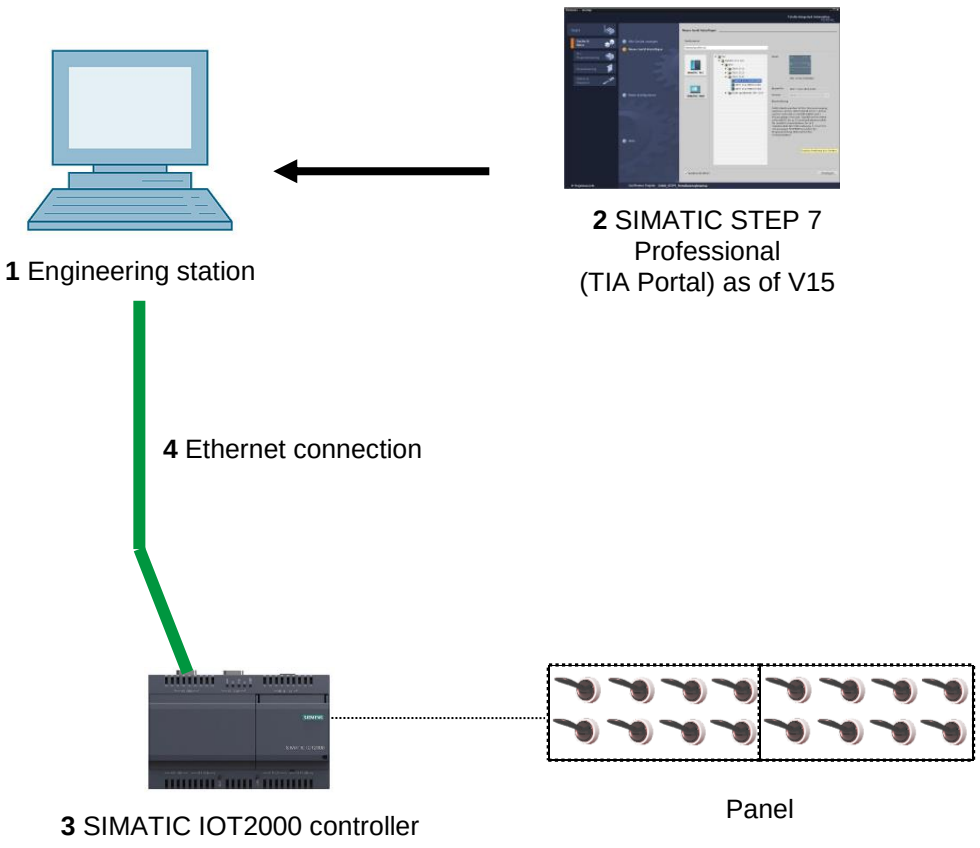
1 Engineering station: requirements include hardware and operating system (for additional information, see Readme on the TIA Portal Installation DVD)

2 SIMATIC STEP 7 Professional software in TIA Portal – V15 or higher

3 SIMATIC IOT2000 controller, e.g. IOT2040 with MicroSD card and IO shield

Note: The digital inputs should be fed out to a 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:

– Processing a warm restart

– Updating the process image input and the process image output

– 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 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.

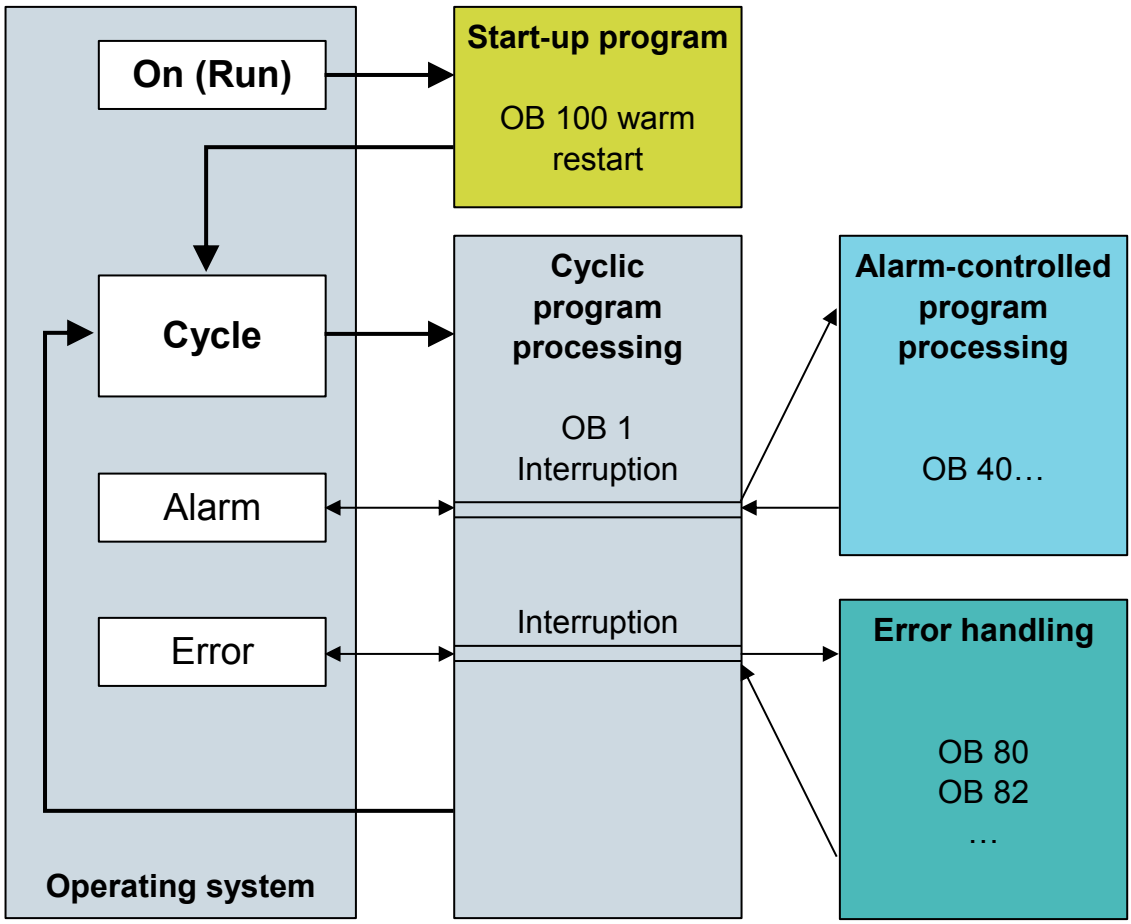


Figure 1: Start events in the operating system and organization block call

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 you have not assigned an OB to the event, the default system reaction is performed.

Table 1 shows examples for different start events for SIMATIC IOT2000. Possible OB number(s) and the preset system reactions that occur if the respective organization block (OB) is not present in the controller are also illustrated.

Start event	Possible OB number	Default system reaction
Startup	100	Ignore
Cyclic program	1	Ignore
Time-of-day interrupt	10	-
Maximum cycle time exceeded	80	STOP

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 (Q), 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 beginning of the cyclic program, the query is made as to whether or not the individual inputs carry voltage. This status of the inputs is stored in the **process image input (PII)**. The information 1 or "High" is hereby stored for energized inputs and the information 0 or "Low" for de-energized inputs.
2. The CPU then executes the program stored in the cyclic organization block. Then, for the required input information, the CPU accesses the previously read **process image input (PII)** and the results of logic operation (RLOs) are written to a so-called **process image output (PIQ)**.
1. At the end of the cycle, the **process image output** (**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.

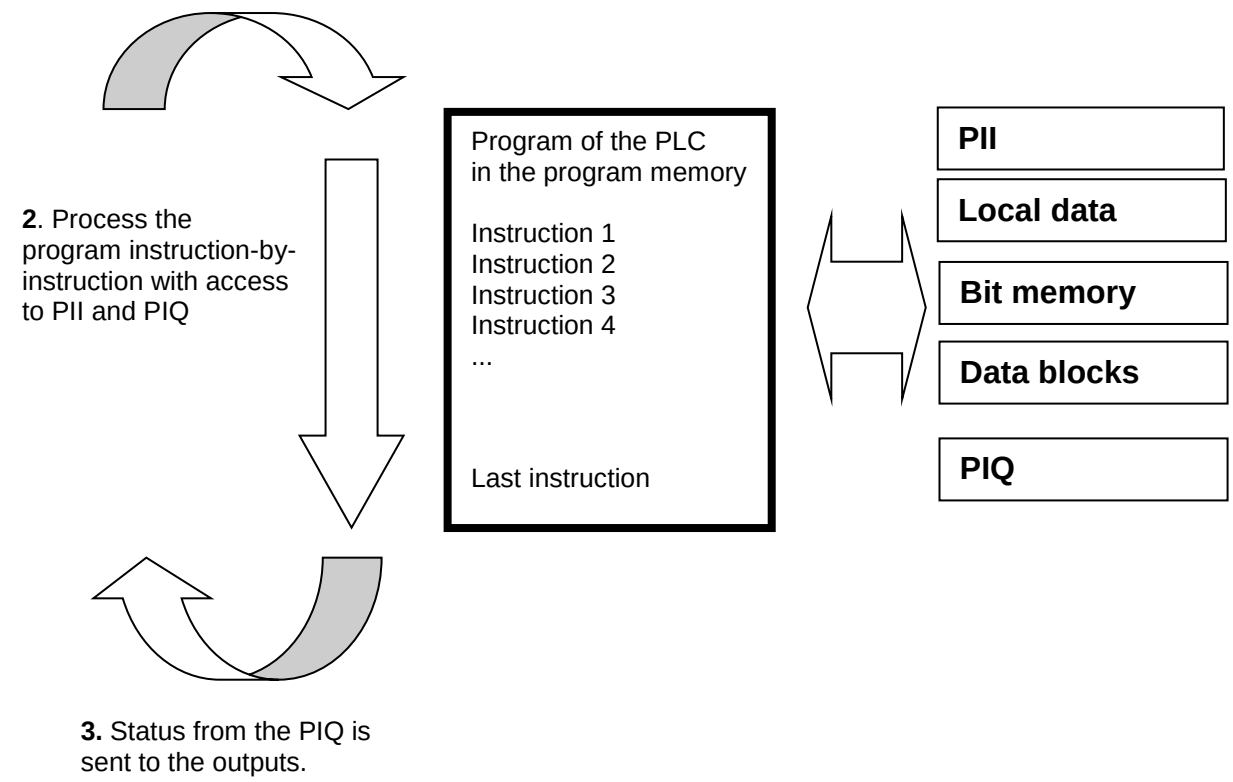


Figure 2: Cyclic program processing

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

– Mathematical functions – that report 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.

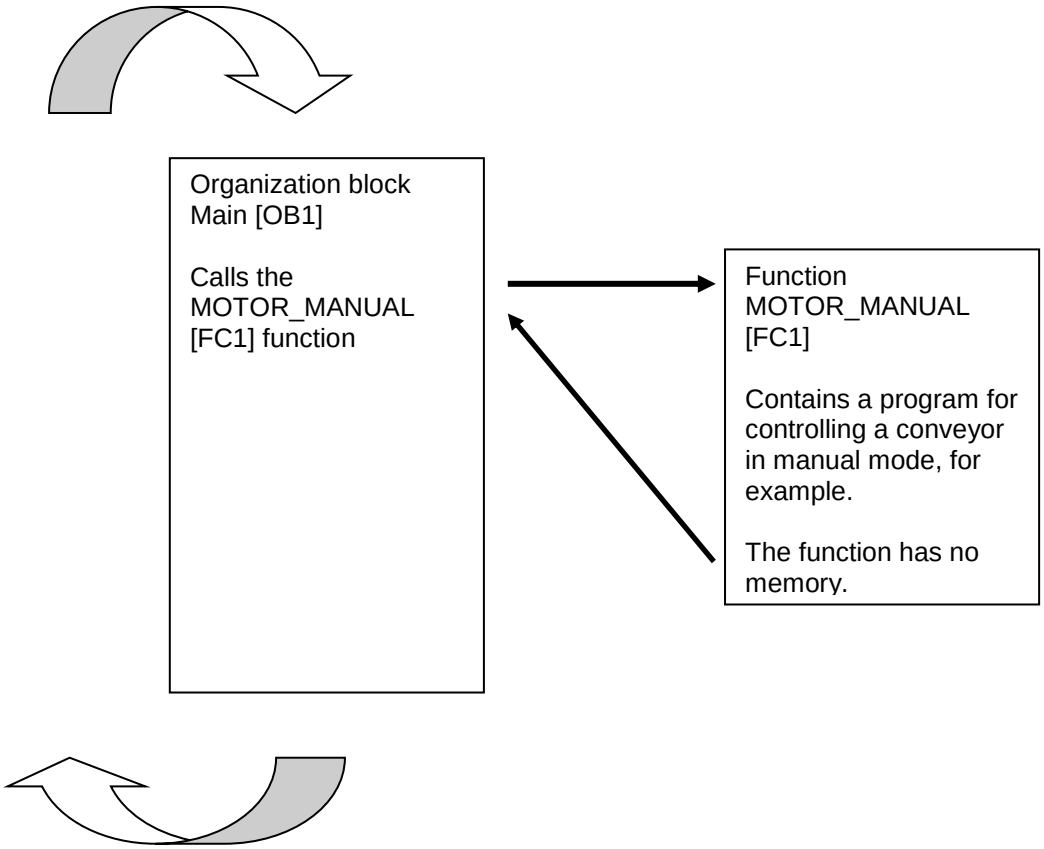


Figure 3: Function with call from organization block Main [OB1]

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 also 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. They 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 or

– Whenever information must be saved in the program, such as pre-selection of the operating mode with a button.

Function blocks are always executed if 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 and 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.

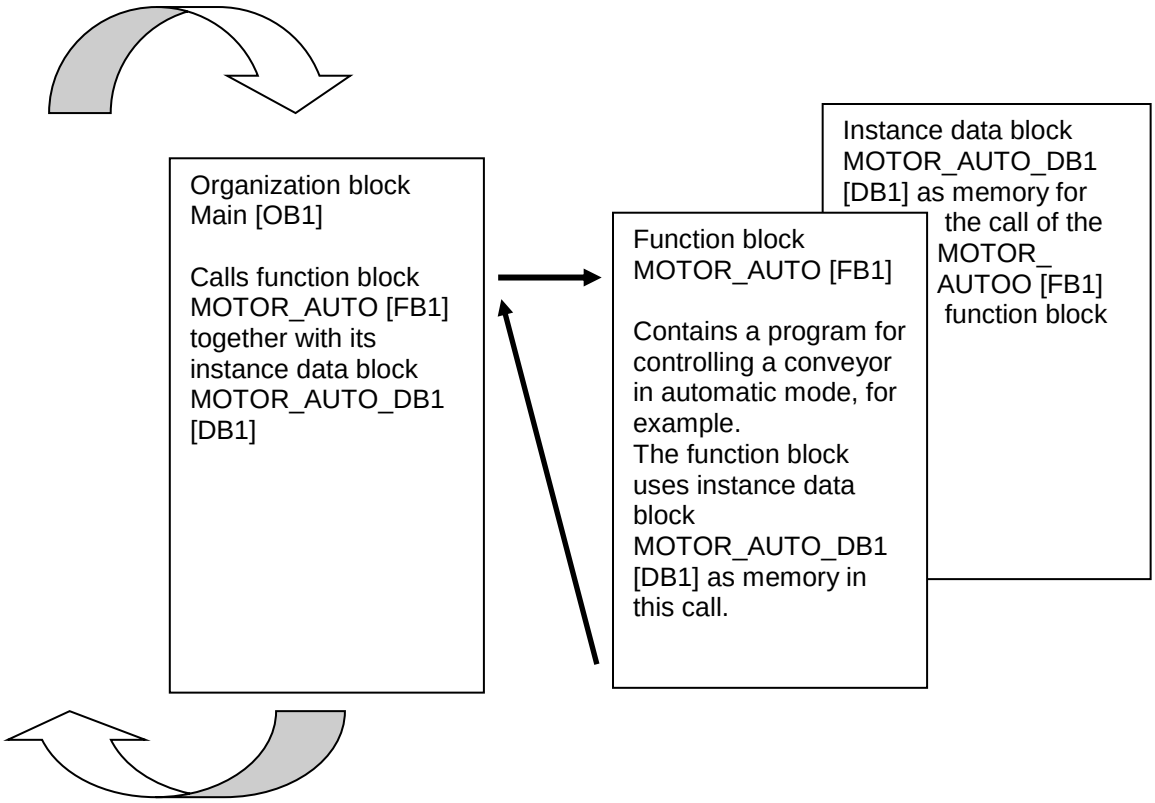


Figure 4: Function block and instance with call from organization block Main[OB1]

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.

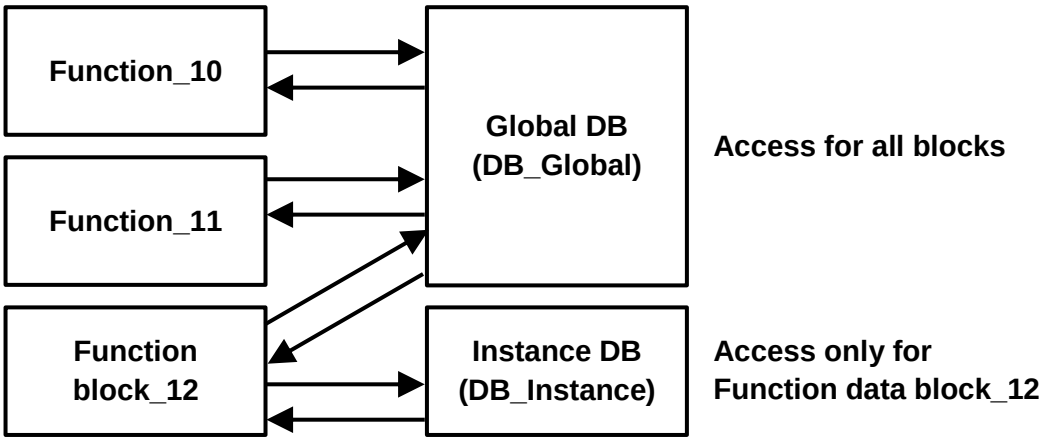


Figure 5: Difference between global DB and instance DB.

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

– Saving 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 to the cycle OB, but is only suitable for very simple programs.

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 preferably be created. 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.

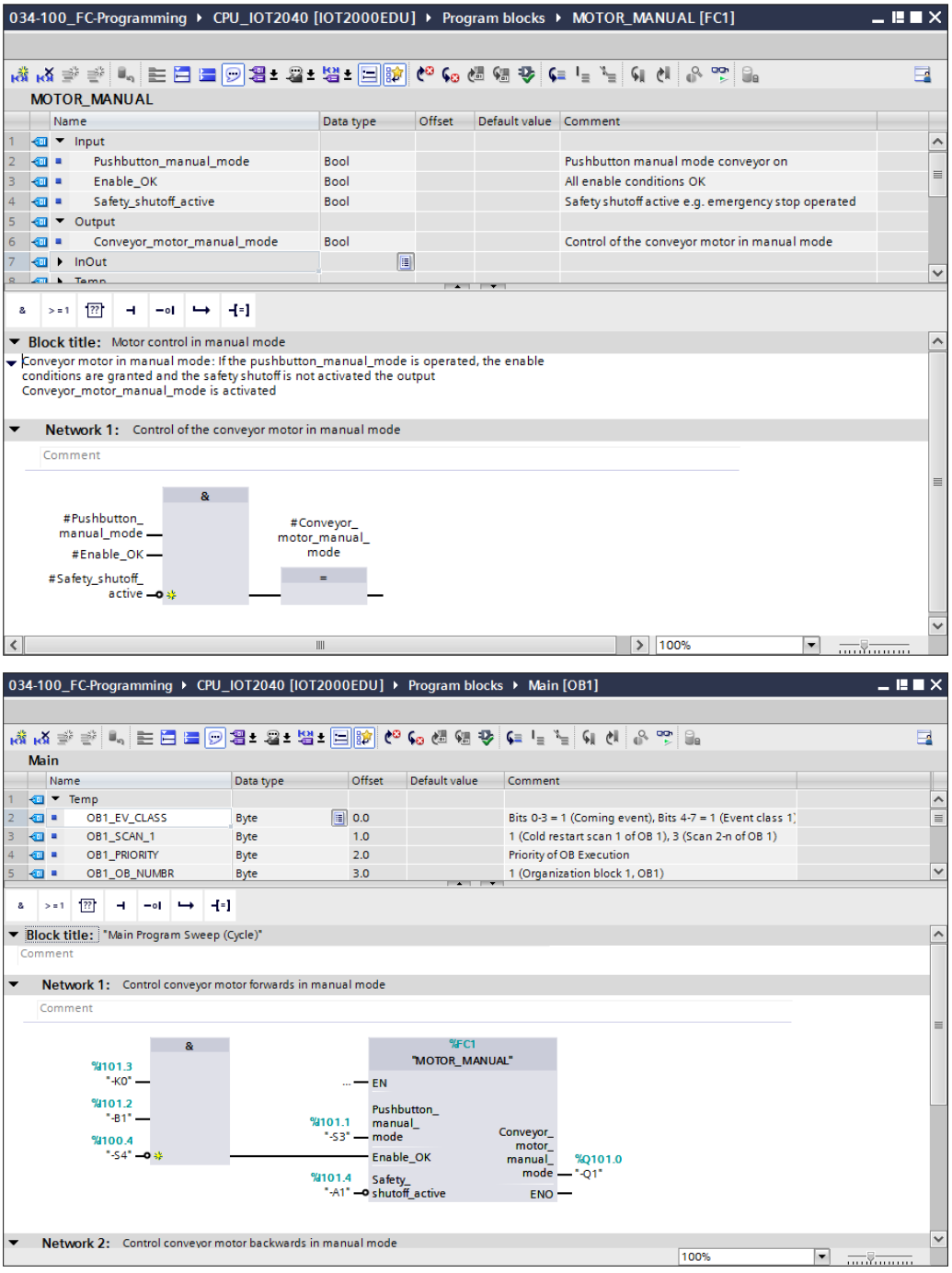


Figure 6: Library-compatible function with call in OB1

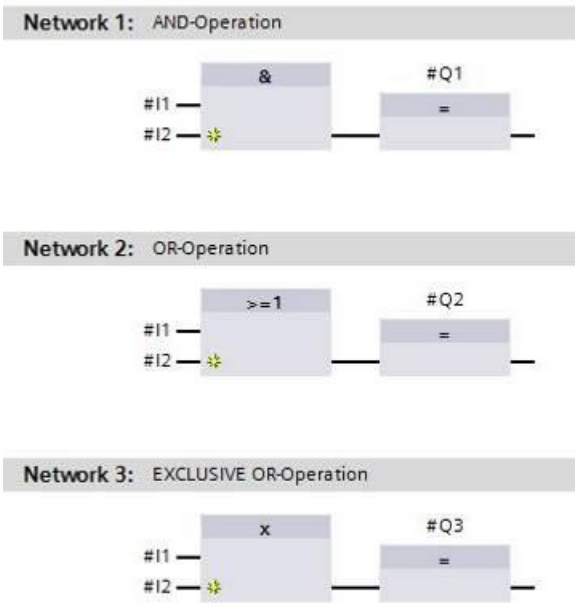
4.8 Programming languages

The available programming languages for programming functions and function blocks for SIMATIC S7-1200 are Function Block Diagram (FBD), Ladder Diagram (LAD) and Structured Control Language (SCL).

The ***Function Block Diagram (FBD)*** programming language is presented below.

FBD is a graphical programming language. The representation is based on electronic circuit 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.



#e1	#e2	#a1
0	0	0
0	1	0
1	0	0
1	1	1

#e1	#e2	#a2
0	0	0
0	1	1
1	0	1
1	1	1

#e1	#e2	#a3
0	0	0
0	1	1
1	0	1
1	1	0

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 that enables you to 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" has signal state "1". When the box is processed correctly, enable output "ENO" has signal state "1". If an error occurs during the processing, the "ENO" enable output is reset. If the 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:

– Manual mode: Control conveyor motor forwards in manual mode

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 are to be moved to FCs and which are to run in OB 1 is planned below.

6.1 EMERGENCY STOP

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 Manual mode – Conveyor motor in manual mode

Manual mode of the conveyor motor is to be encapsulated in a function (FC) "MOTOR_MANUAL". On the one hand, this ensures 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.

Input	Data	Comment
Pushbutton_manual_mode	BOOL	Pushbutton manual mode conveyor on
Enable_OK	BOOL	All enable conditions OK
Safety_shutoff_active	BOOL	Safety shutoff active, e.g. EMERGENCY STOP pressed
Output		
Conveyor_motor_manual_mode	BOOL	Control of the conveyor motor in manual mode

Table 2: Parameters for FC "MOTOR_MANUAL"

Output Conveyor_motor_manual_mode is ON as long as Pushbutton_manual_mode is pressed, the enable is set and the safety shutoff is not active.

6.3 Technology diagram

Here, you see the technology diagram for the task.

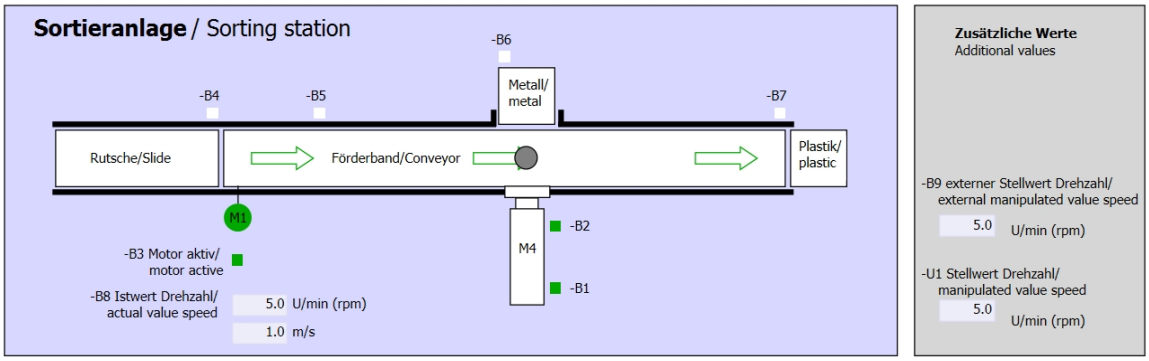


Figure 8: Technology diagram

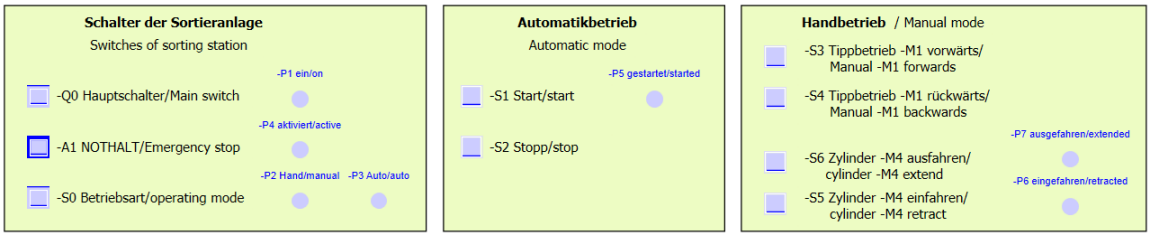


Figure 9: Control panel

6.4 Reference list

The following signals are required as operands for this task.

DI	Type	Identifier	Function	NC/NO
I 101.4	BOOL	-A1	Return signal emergency stop OK	NC
I 101.3	BOOL	-K0	Station "ON"	NO
I 101.2	BOOL	-B1	Sensor cylinder -M4 retract	NO
I 101.1	BOOL	-S3	Pushbutton manual mode conveyor -M1 forwards	NO
I 100.4	BOOL	-S4	Pushbutton manual mode conveyor -M1 backwards	NO

DQ	Type	Identifier	Function	
Q 101.0	BOOL	-Q1	Conveyor motor -M1 forwards fixed speed	

Legend for reference list

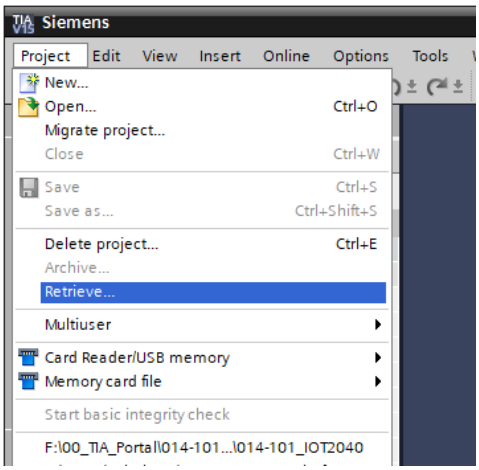
DI	Digital input	DQ	Digital output
AI	Analog input	AQ	Analog output
I	Input	Q	Output
NC	Normally Closed		
NO	Normally Open		

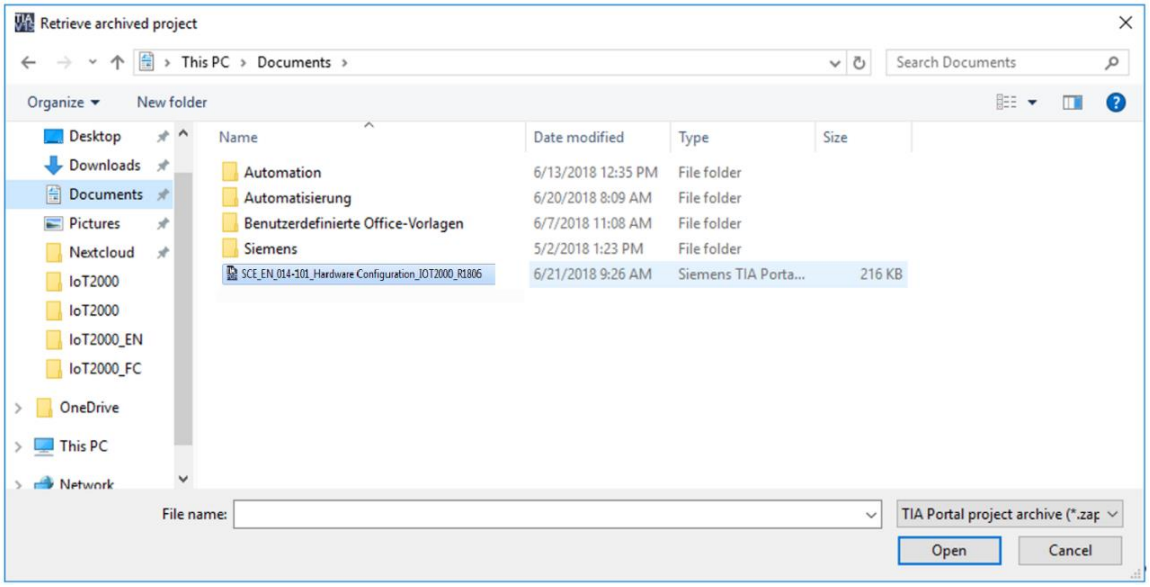
7 Structured step-by-step instructions

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.

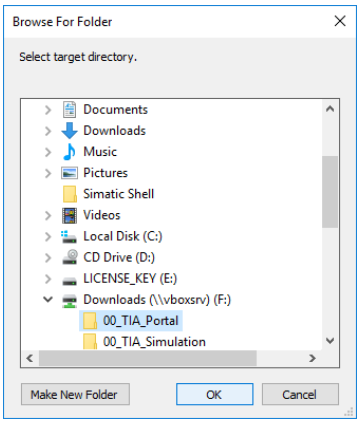
7.1 Retrieving an existing project

→ Before we can start programming the function (FC) "MOTOR_MANUAL", we need a project with a hardware configuration.
(e.g. SCE_DE_014_101_Hardware_Configuration_IOT2000.zap14). 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)



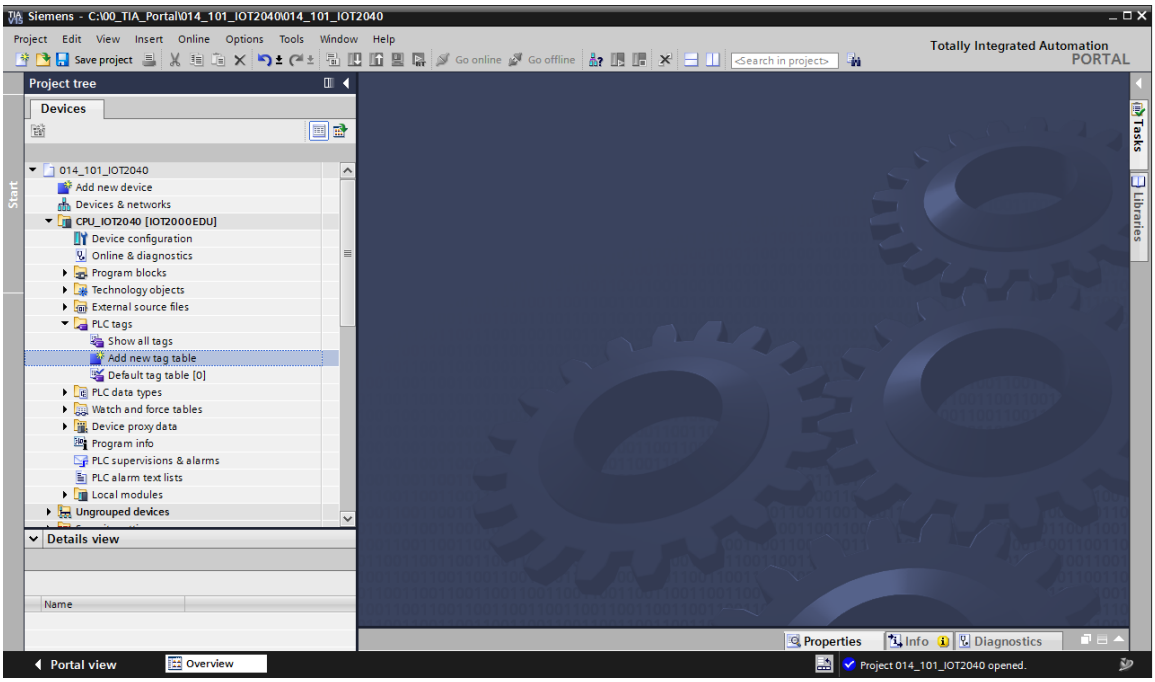


→ With the next step, you select the target directory where the retrieved project will be stored. Confirm your selection with "OK". (→ Target directory → OK)

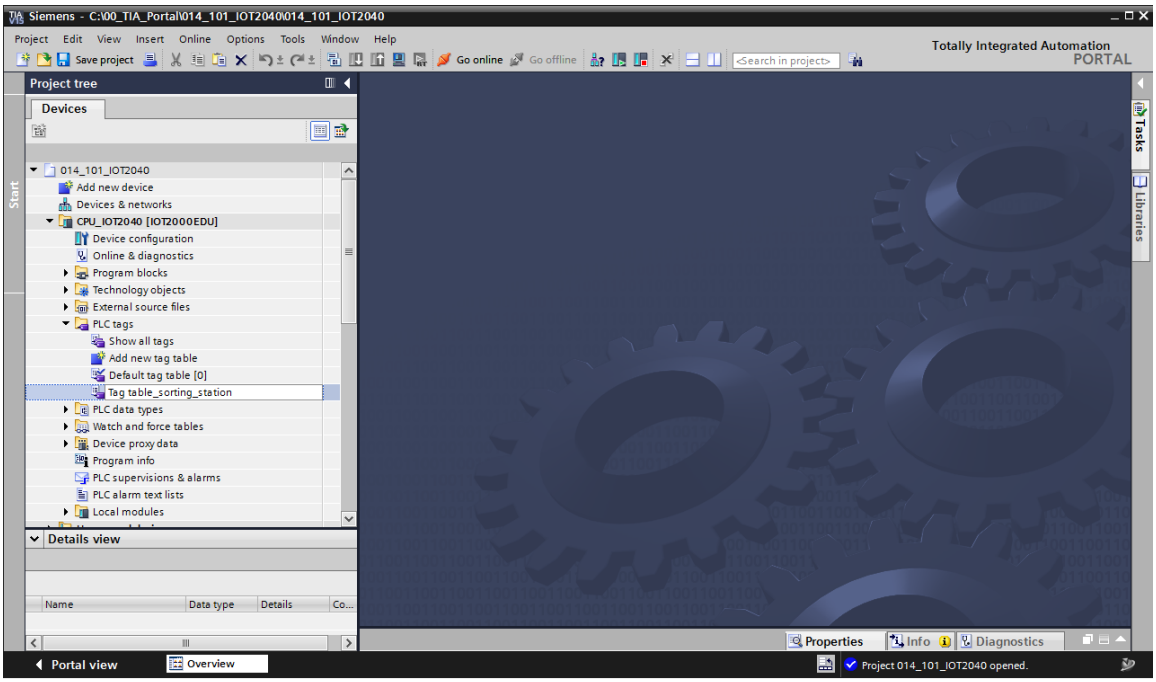


7.2 Creating a new tag table

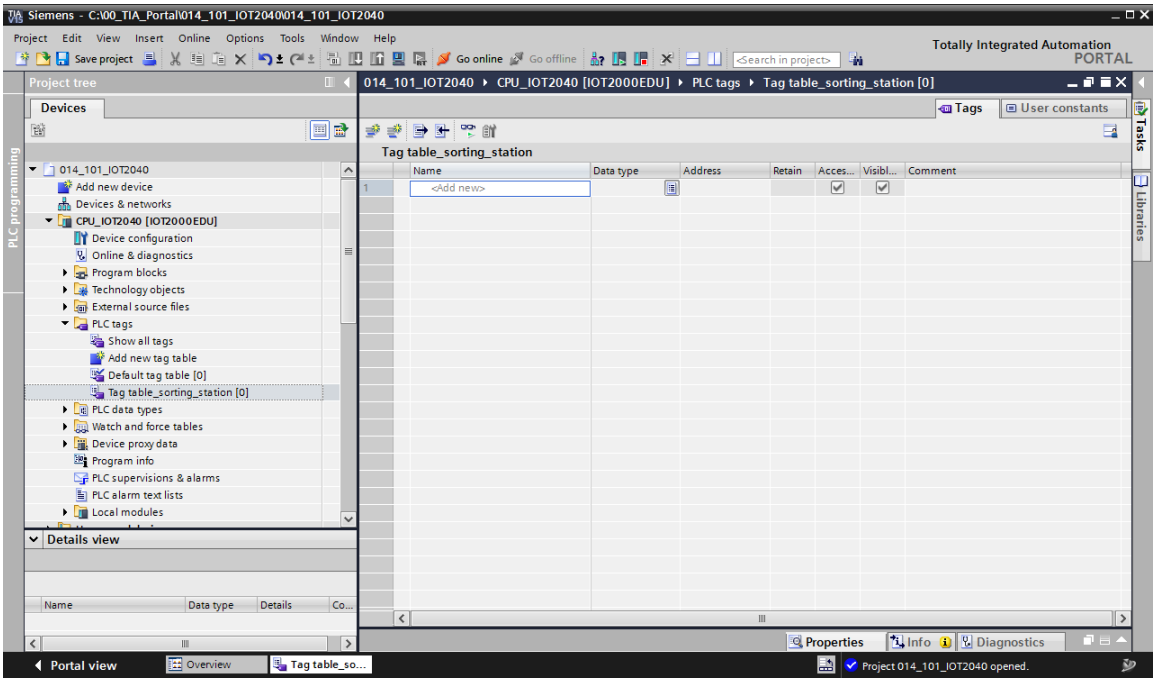
→ In the project view, navigate to the → PLC tags of your controller and create a new tag table with a double-click on → Add new tag table.



→ Rename the tag table you just created as "Tag table_sorting_station" (→ right-click "Tag table_1" → "Rename" → Tag_table_sorting_station).

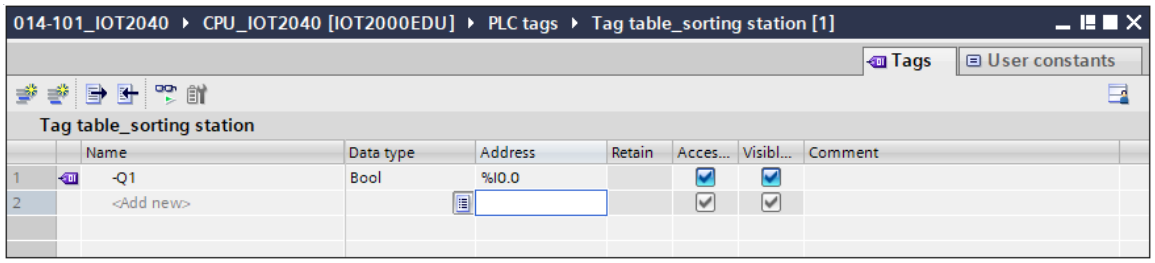


→ Open it with a double-click. (→ Tag table_sorting_station)



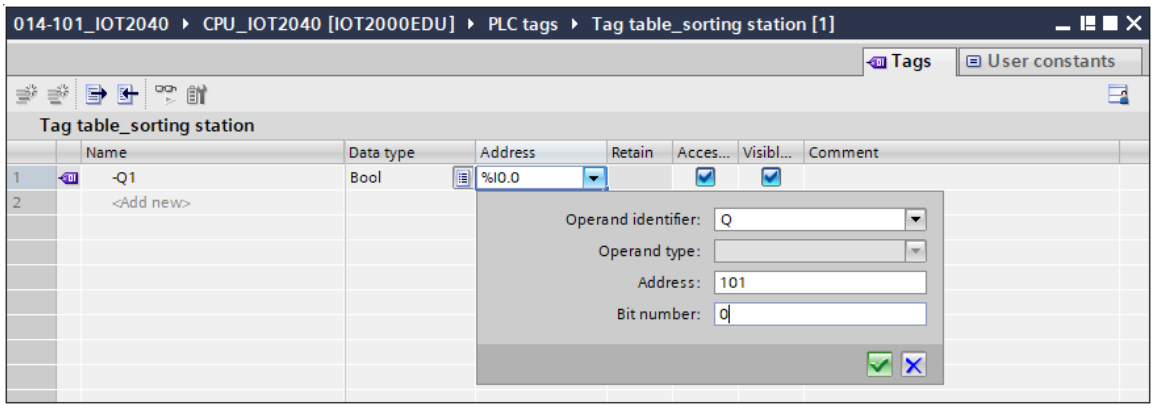
7.3 Creating new tags within a tag table

→ 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 (→ <Add> → -Q1 → Enter).

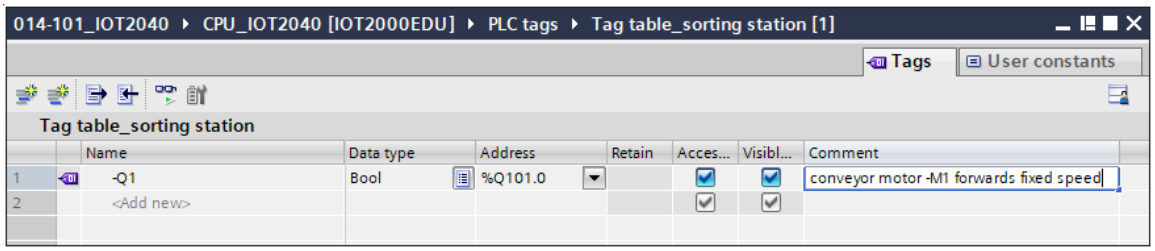


→ Change the address to %Q101.0 by entering this directly or by clicking the drop-down arrow to open the Address menu. Change the operand identifier to A and the address to 101.

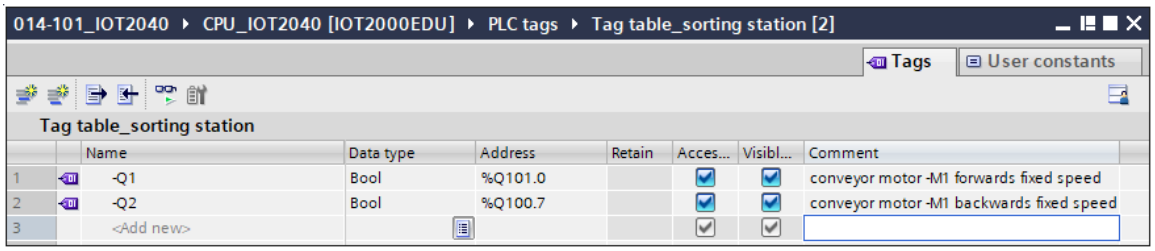
Confirm with Enter or by clicking on the check mark. (→ %I0.0 → Operand identifier → A → Address→ 101 → ✓)



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

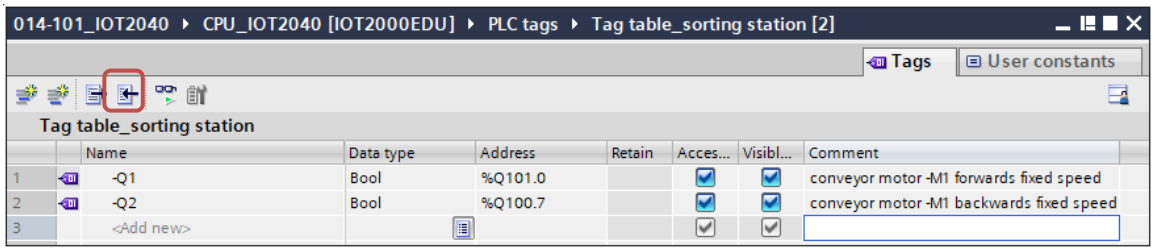


→ Add a new -Q2 tag in line 2. TIA Portal has automatically assigned the same data type as in line 1 and has incremented the address by 1 to…. Enter the comment "conveyor motor M1 backwards fixed speed" and change the address to %Q100.7. (→ <Add> → -Q2 → Enter → Comment → Conveyor motor M1 backwards fixed speed)

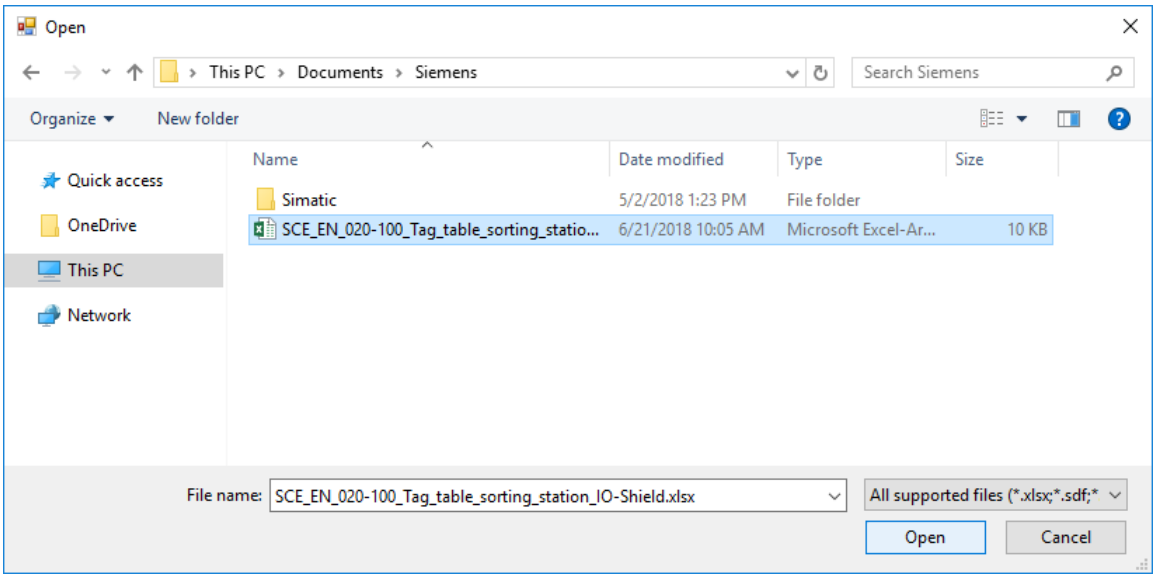


7.4 Importing "Tag_table_sorting_station"

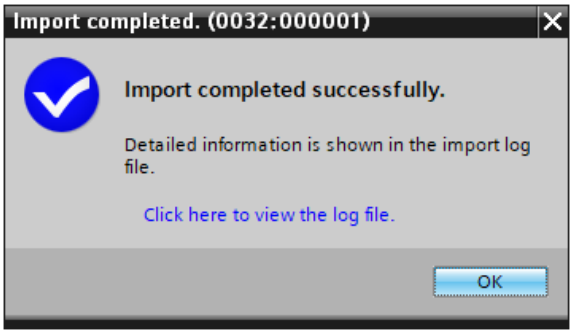
→ To insert an existing symbol table, click on the "Import" icon in the tag table toolbar. (→ Import)



→ Select the desired symbol table (e.g. in .xlsx format) and confirm the selection with "Open". (→ SCE_DE_020-100_Tag_table_sorting_station_IO-Shield … → Open)



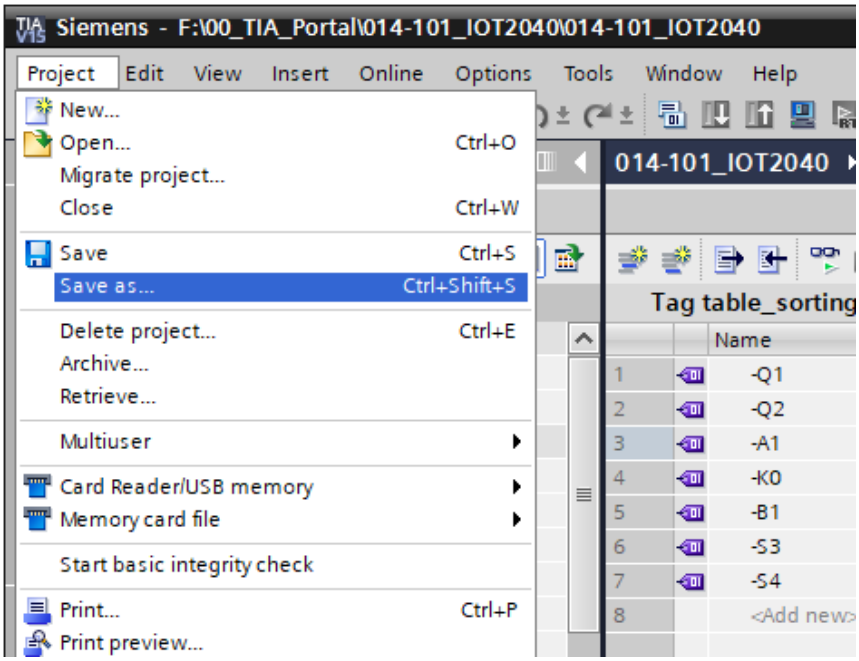
→ When the import is finished, you will see a confirmation window and have an opportunity to view the log file for the import. Click → OK.

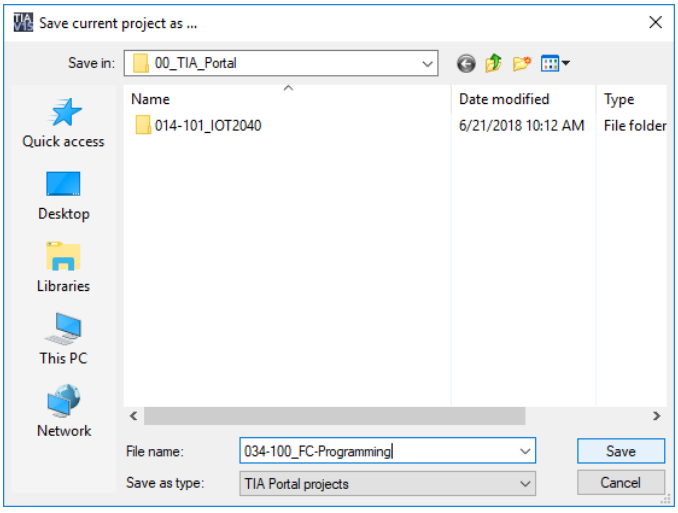


→ Tags that already exist in the system are updated and missing tags are added. Tags that exist in the project but not in the import file are retained.

→ You now have a complete symbol table of the digital inputs and outputs in front of you. Now save your project under the name 034-100_FC-Programming.

(→ Project → Save as … → 034-100_FC-Programming → Save)

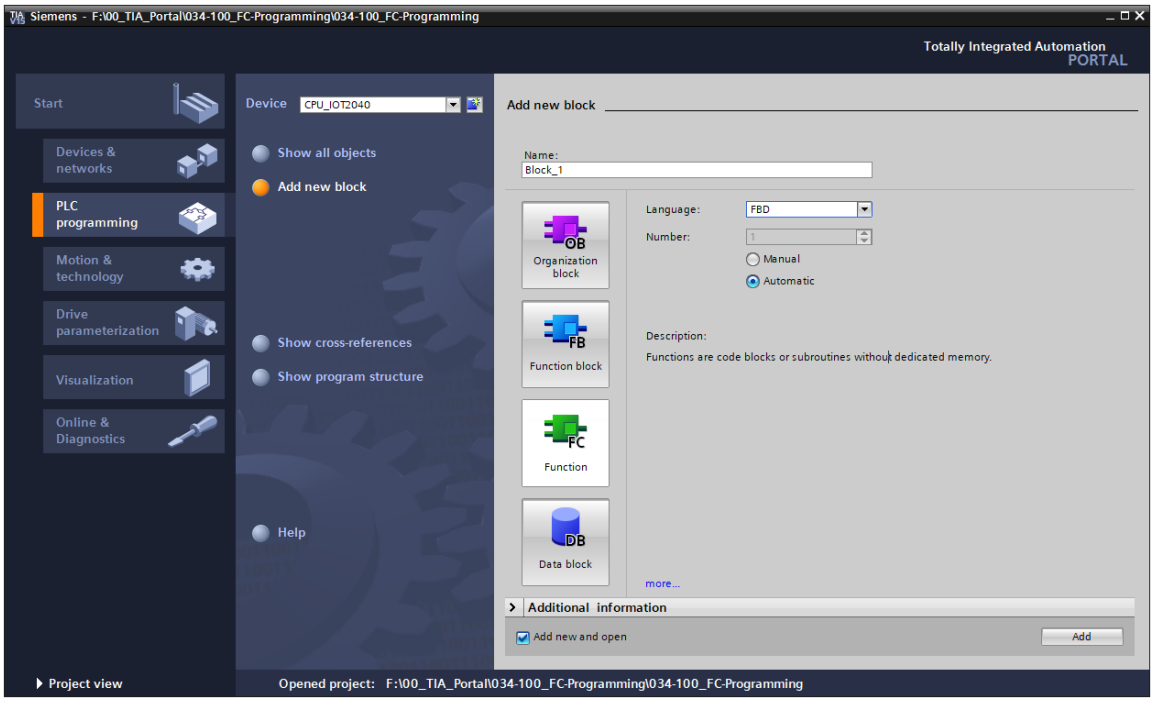




7.5 Creating function FC1 "MOTOR_MANUAL" for the conveyor motor in manual mode

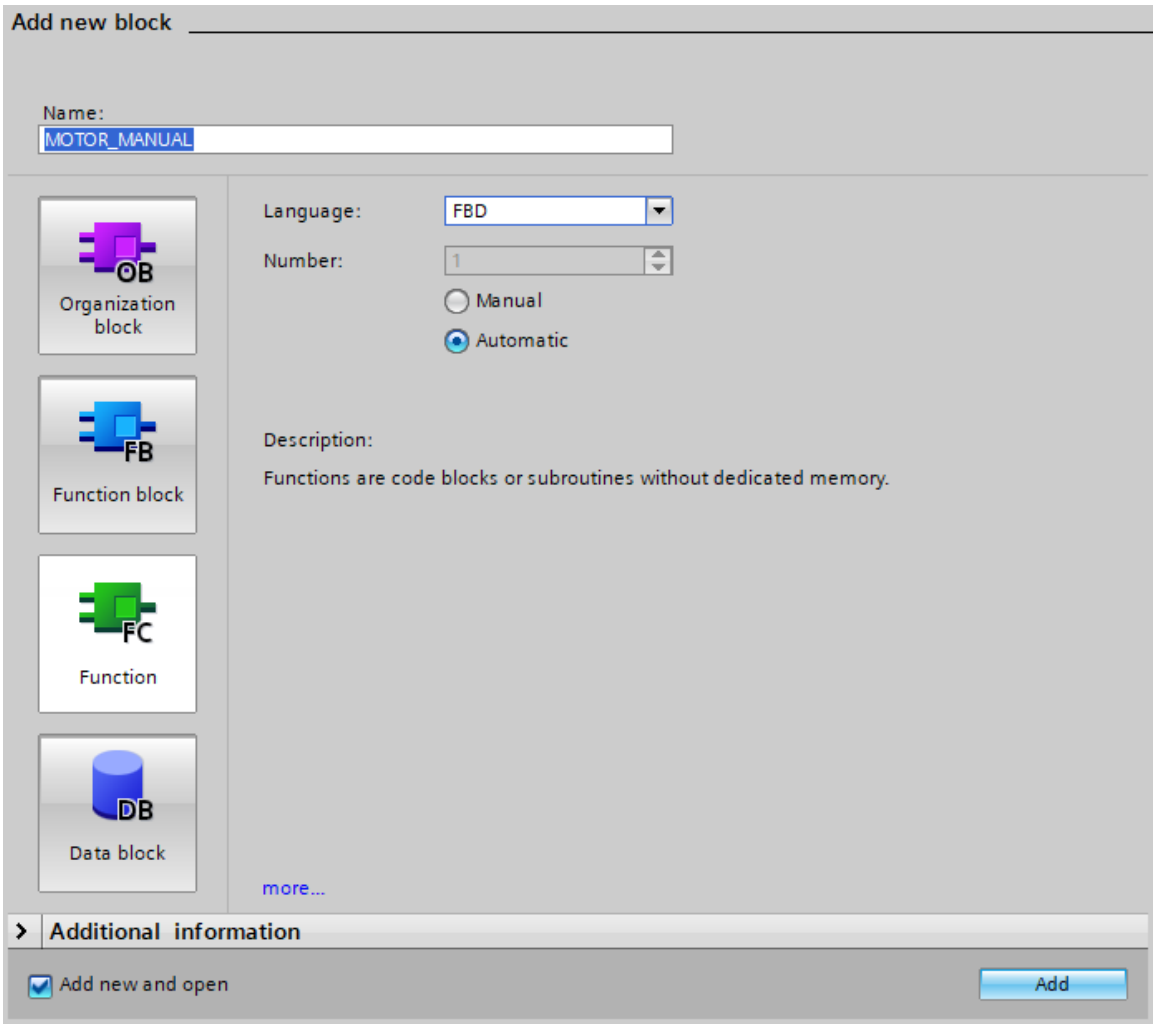
→ In the PLC programming section of the portal view, click "Add new block" to create a new function here.

(→ PLC programming → Add new block → )



→ Rename your new block to: "MOTOR_MANUAL" Then set the language to FBD and allow 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".

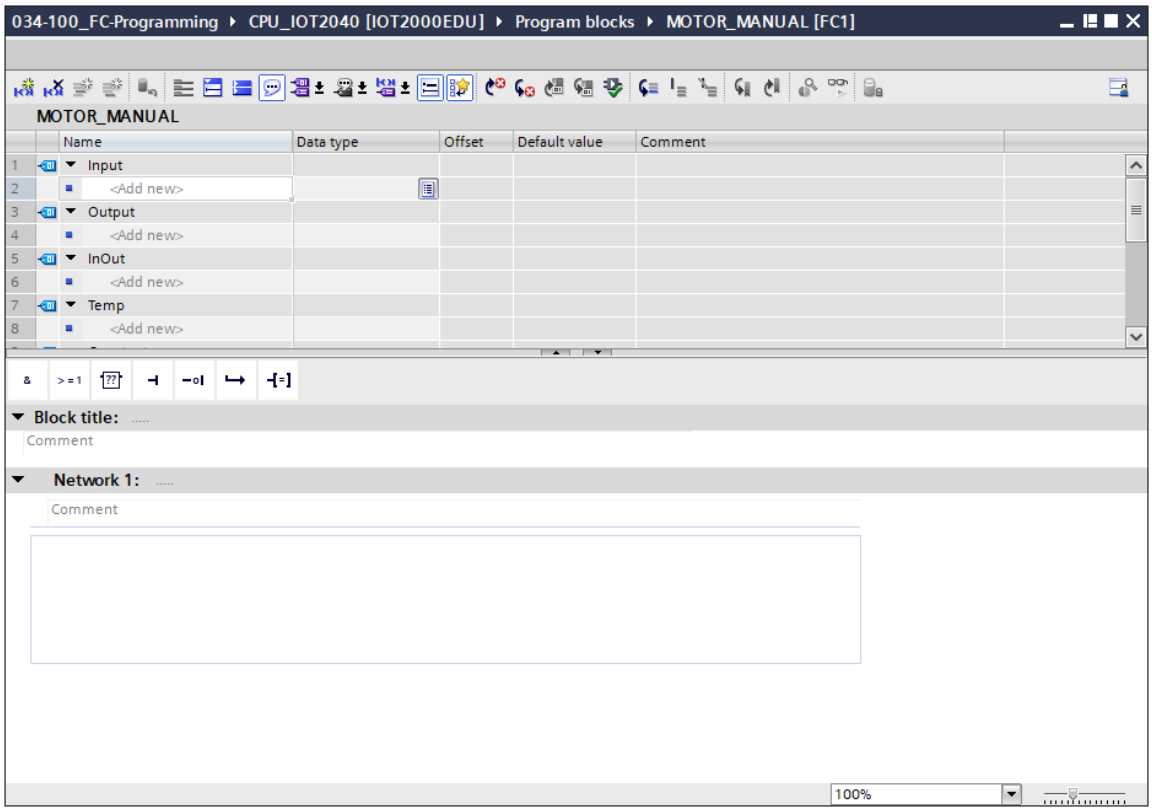
(→ Name: MOTOR_MANUAL→ Language: FBD → Number: Automatic → ☑ Add new and open → Add)



7.6 Defining the interface of function FC1 "MOTOR_MANUAL"

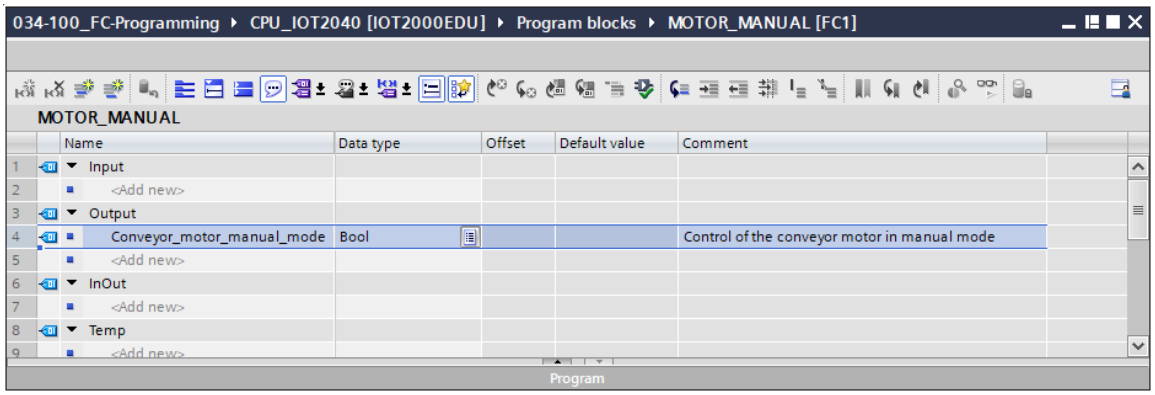
If you selected "Add new and open", the project view opens with a window for creating the block you just added.

→ You can find the interface description of your function in the upper section of your programming view.



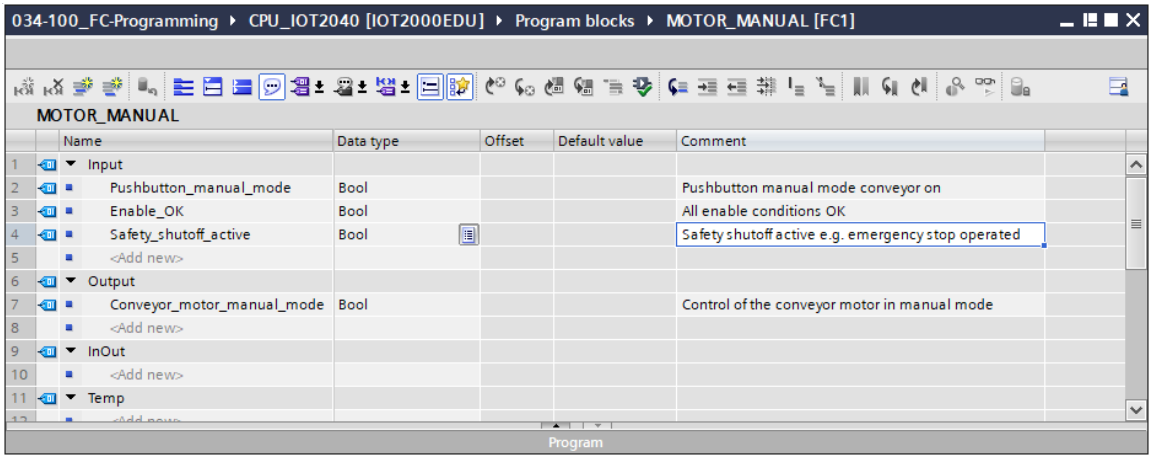
→ A binary output signal is needed for controlling the conveyor motor. For this reason, we first create local output tag #Conveyor_motor_manual_mode of the "Bool" type. Enter comment "Control of the conveyor motor in manual mode" for the parameter.

(→ Output: Conveyor_motor_manual_mode → Bool → Control of the conveyor motor in manual mode)



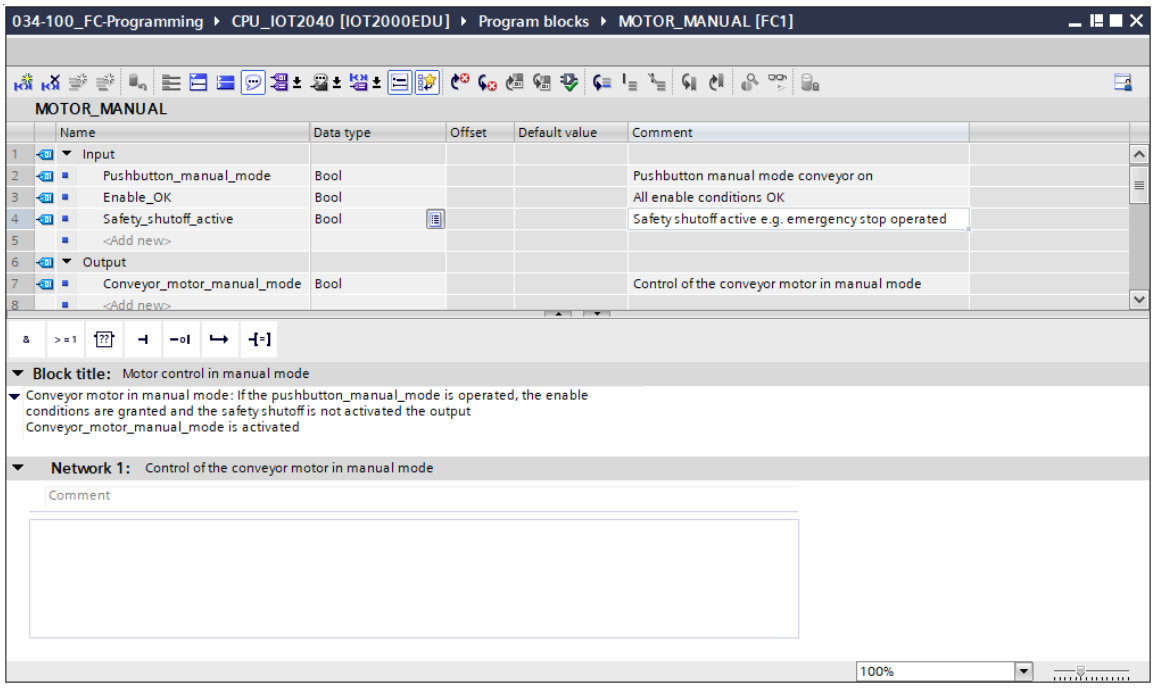
→ Add parameter #Pushbutton_manual_mode as the input interface under Input and confirm the entry with the Enter key or by exiting the text box. Data type "Bool" is assigned automatically. This will be retained. Then enter the corresponding comment "Pushbutton manual mode conveyor on". (→ Pushbutton_manual_mode → Enter → Bool → Pushbutton manual mode conveyor on)

→ Now add under Input the parameters #Pushbutton_manual_mode, #Enable_OK and #Safety_shutoff _active as additional binary input parameters and check their data types. Then add meaningful comments.



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

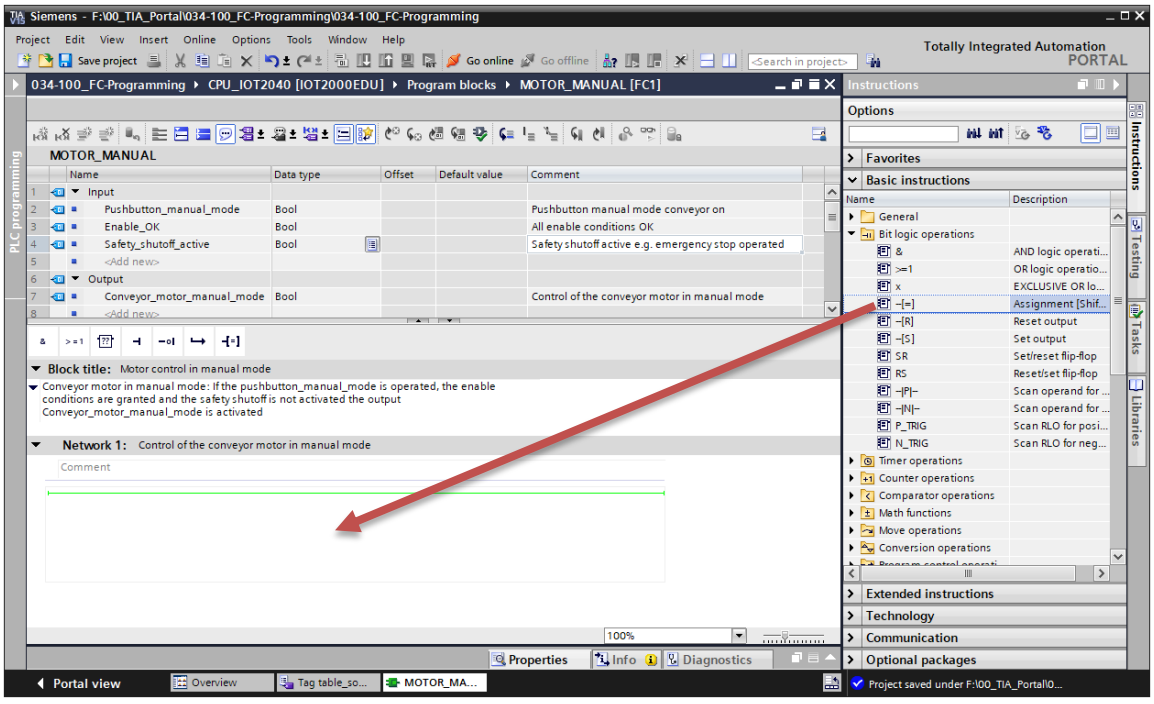
(→ Block title: Motor control in manual mode → Network 1: Control of the conveyor motor in manual mode)



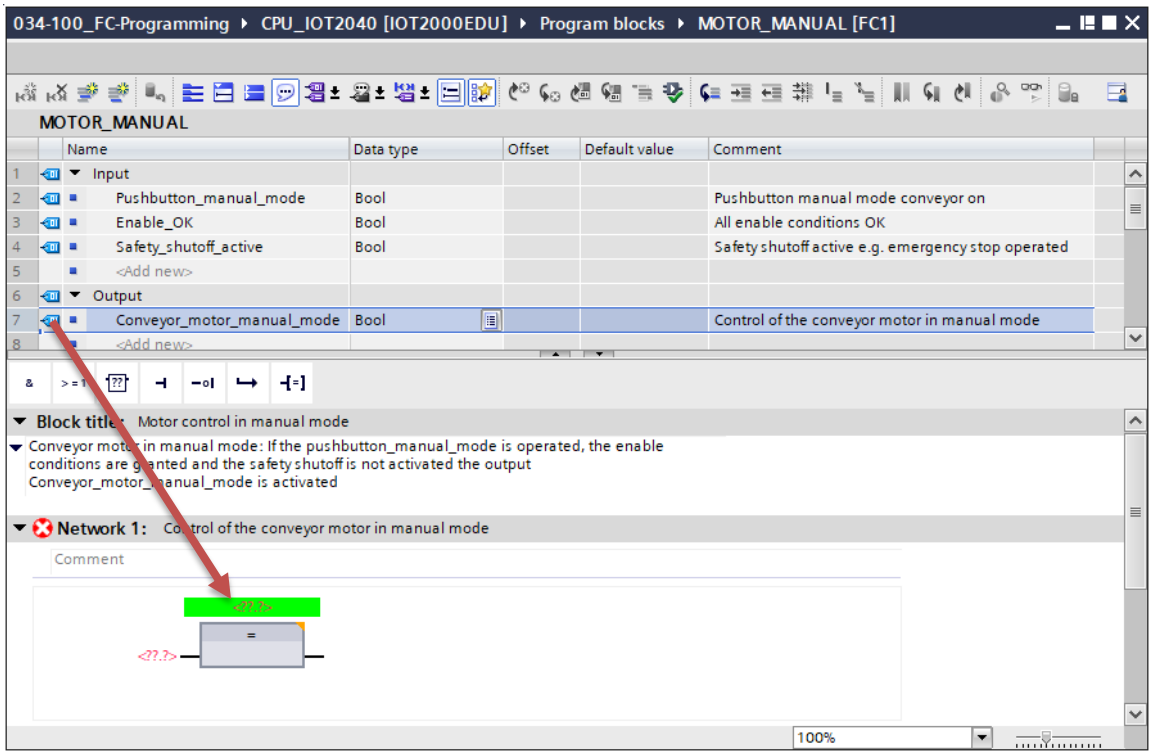
7.7 Programming FC1: MOTOR_MANUAL

→ 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. Below, you will learn about the various options for inserting logic blocks.

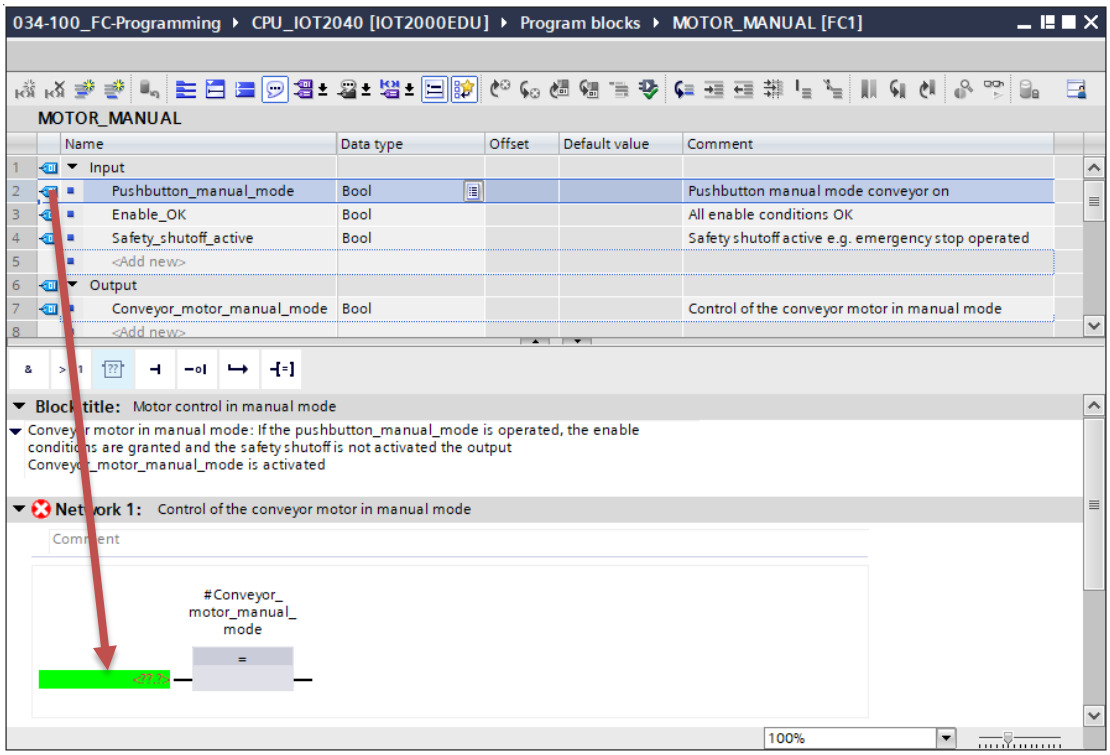
→ You will find a list of instructions that you can use on the right side of your programming window. Under → Basic instructions → Bit logic operations, find the function –[=] (Assignment) and use drag & drop to move it to Network 1 (green line appears, mouse pointer with + symbol). (→ Instructions → Basic instructions → Bit logic operations → –[=])



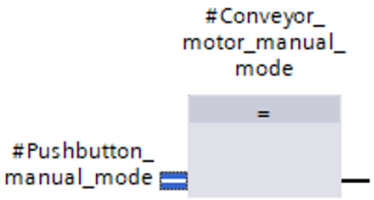
→ Now use drag & drop to move your output parameter #Conveyor_motor_manual_mode onto <??.?> above the block you just inserted. The best way to select a parameter in the interface description is by clicking on it at the blue symbol . (→ Conveyor_motor_manual_mode)



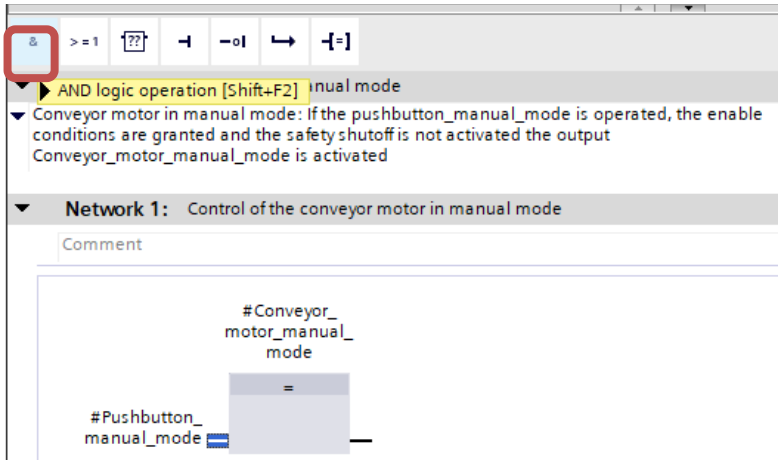
→ This determines that the #Conveyor_motor_manual_mode parameter is written by this block. Still missing, however, are the input conditions so that this actually happens. For this, use drag & drop to move input parameter #Pushbutton_manual_mode to "<??.?>" at the left side of the assignment block. (→ Pushbutton_manual_mode)



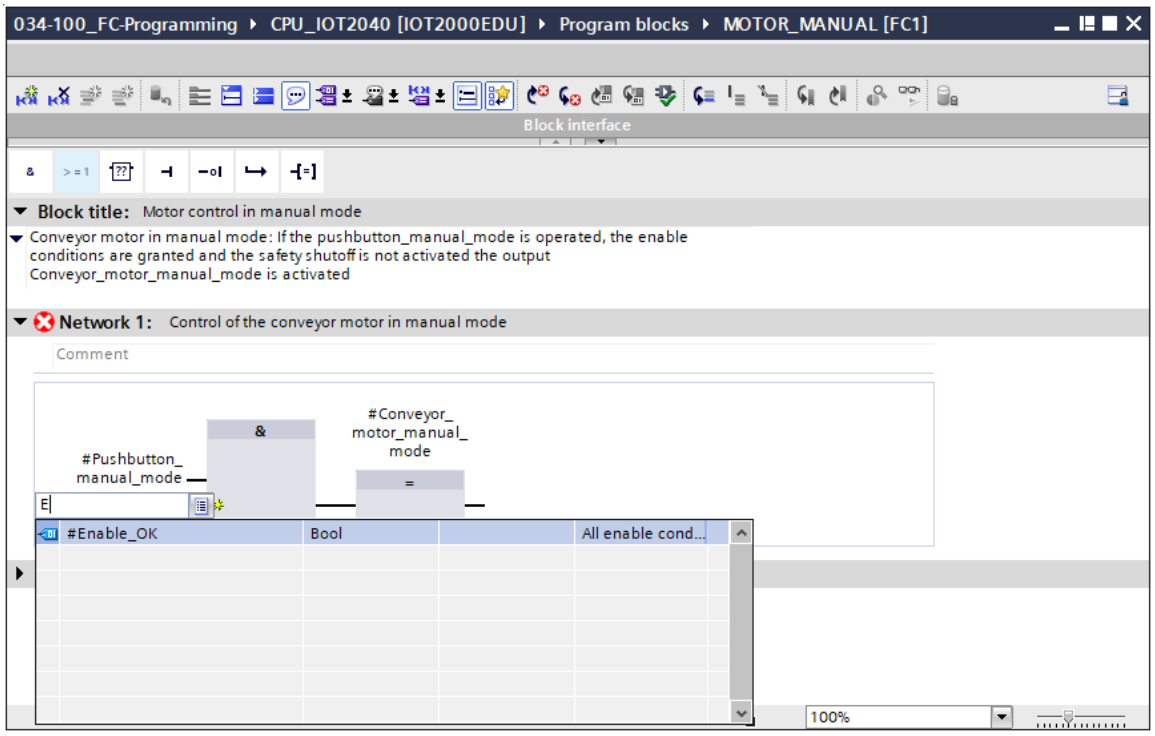
→ The input of the assignment block will also be logically combined with other parameters by an AND logic operation. To do this, first click the input of the block to which #Pushbutton_manual_mode is already connected, so that the input line has a blue background.



→ Click the & on in your logic toolbar to insert an AND logic operation between the #Pushbutton_manual_mode tag and your assignment block.

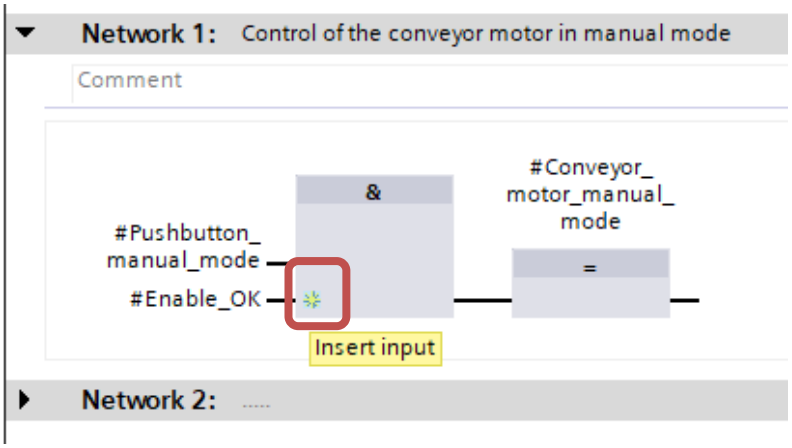


→ Double-click on the second input of the & link <??.?>. Then enter the letter "F" in the field that appears in order to see a list of available tags starting with "F". Now click on the tag #Enable_OK and apply it with → Enter. (→ & block → <??.?> → F → #Enable_OK → Enter)

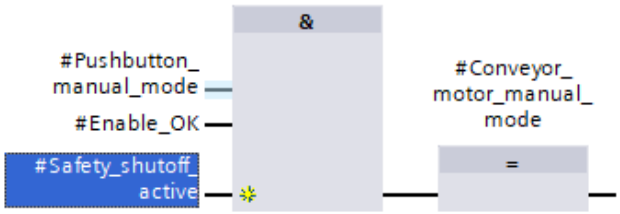


Note: *When assigning tags in this way, there is a risk of a mix-up with the global tags from the tag table. The previously presented procedure using drag & drop from the interface description should therefore preferably be used.*

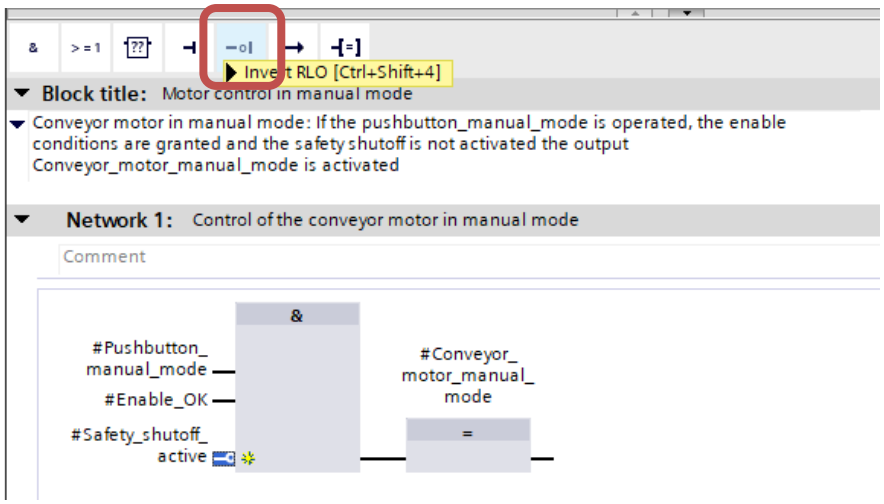
→ To ensure that the output is only activated when the safety shutoff is not active, the input tag #Safety_shutoff_active should be logically combined with the AND logic operation. To do this, click on the yellow star ✳ of the AND block to add another input.



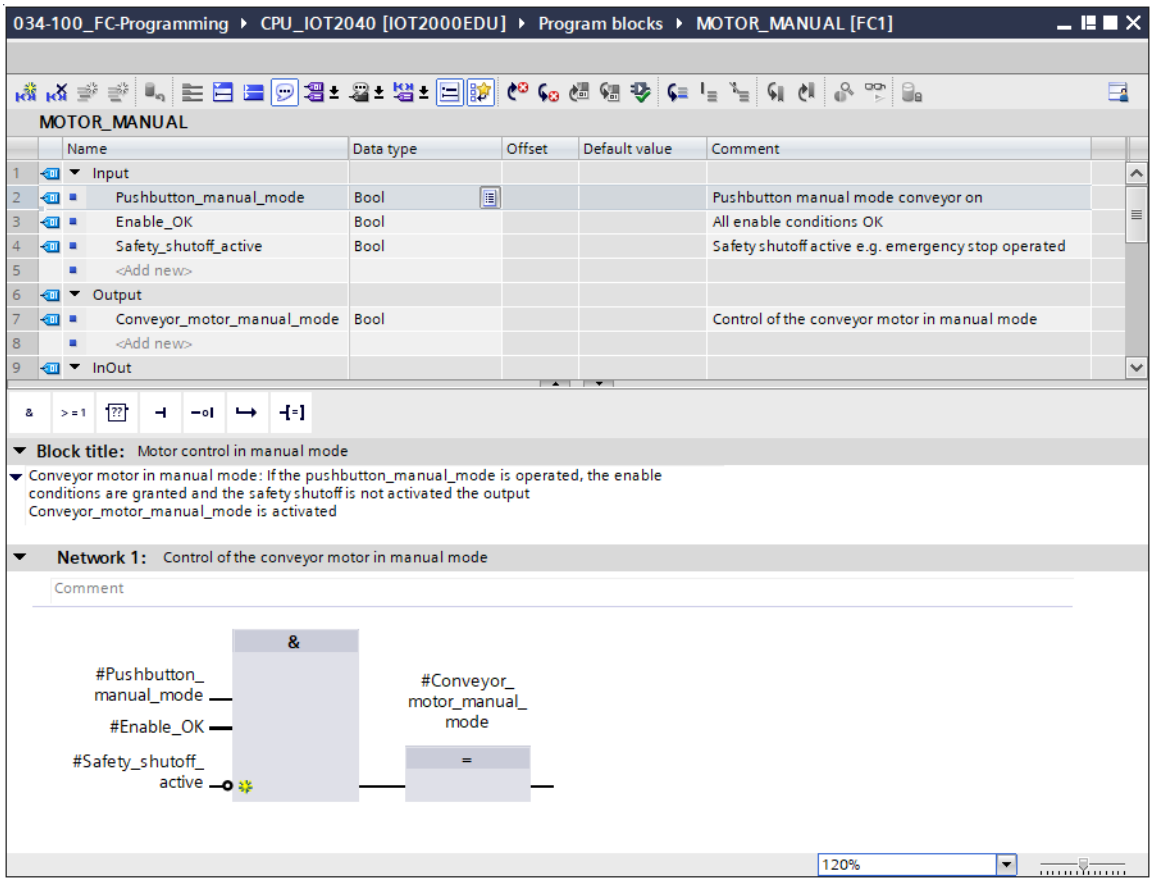
→ Add the input tag #Safety_shutoff_active to your newly created input of the AND element.



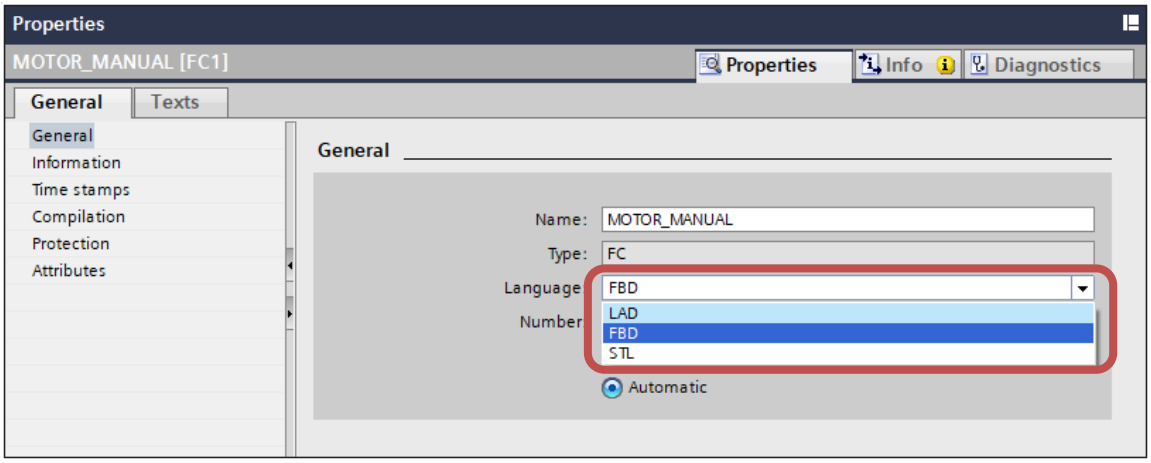
→ Negate the input connected to parameter #Safety_shutoff_active by selecting it and clicking −o|.



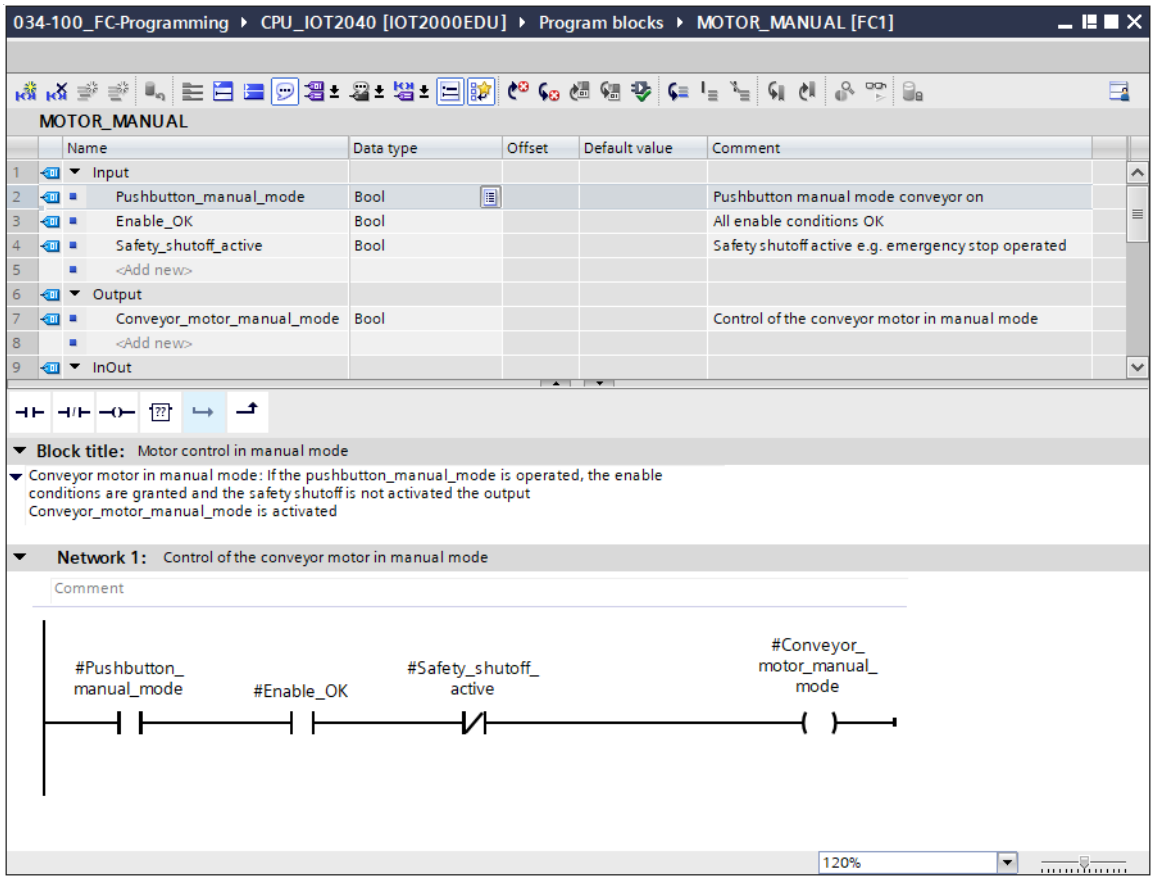
→ Do not forget to regularly click Save project . The finished function "MOTOR_MANUAL" [FC1] in FBD is shown below.



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



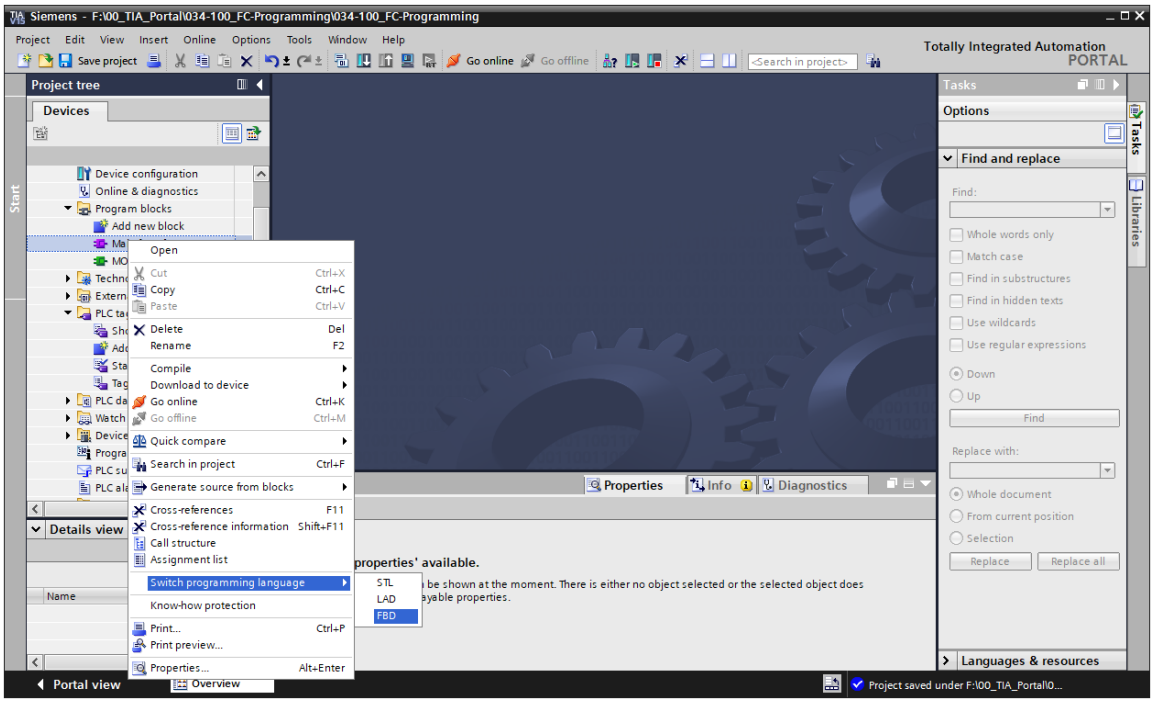
→ The program has the following appearance in LAD.



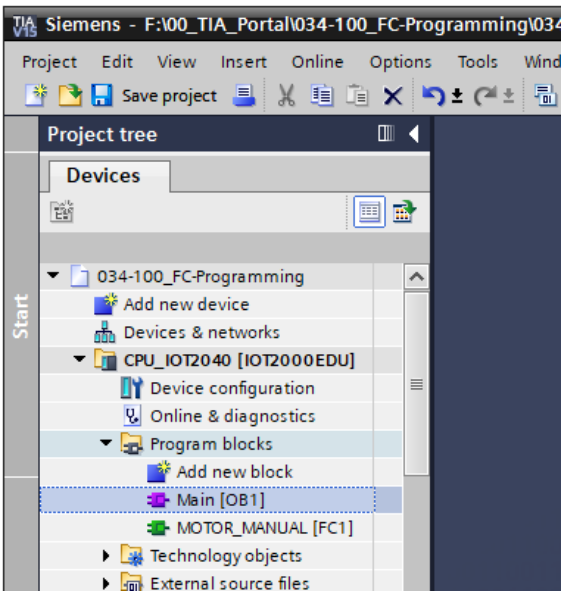
7.8 Programming organization block OB1 – Control conveyor forwards in manual mode

→ Before programming organization block "Main [OB1]", we switch the programming language to FBD (Function Block Diagram). To do this, first click on "Main [OB1]" in the "Program blocks" folder.

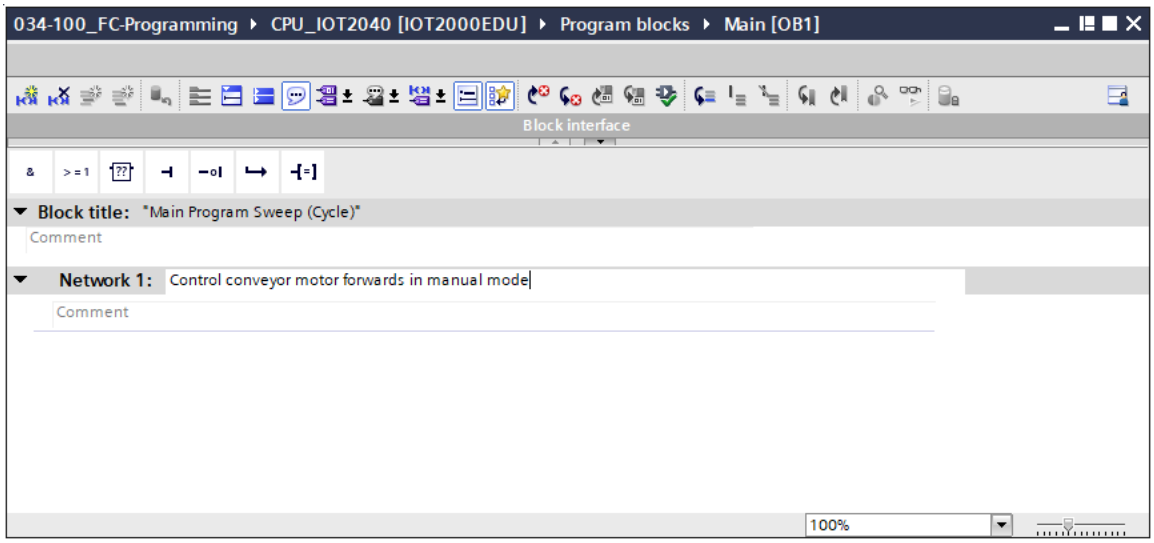
(→ CPU_IOT2040[IOT2000EDU] → Program blocks → Main [OB1] → Switch programming language → FBD)



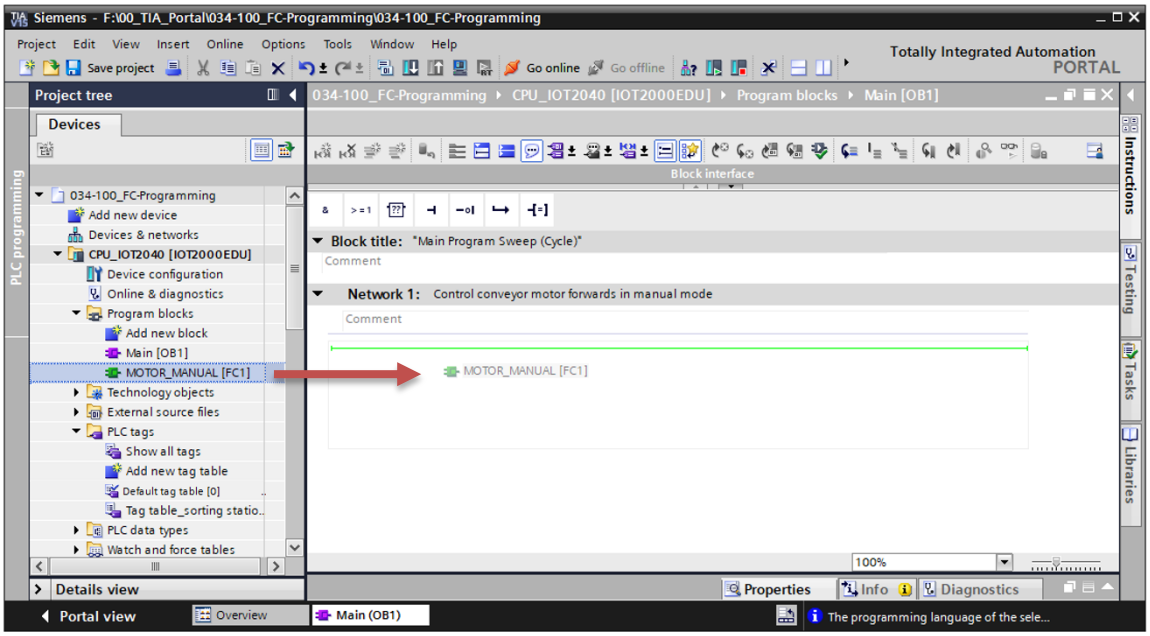
→ Open the "Main [OB1]" organization block with a double-click.



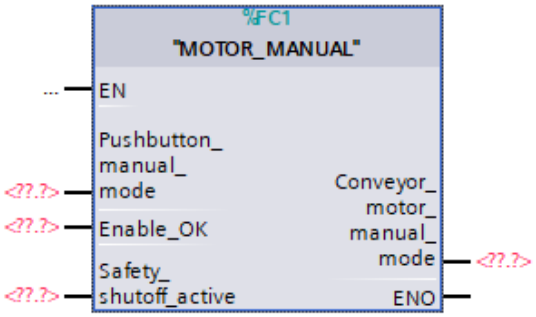
→ Assign Network 1 the name "Control conveyor motor forwards in manual mode" (→ Network 1:… → Control conveyor motor forwards in manual mode)



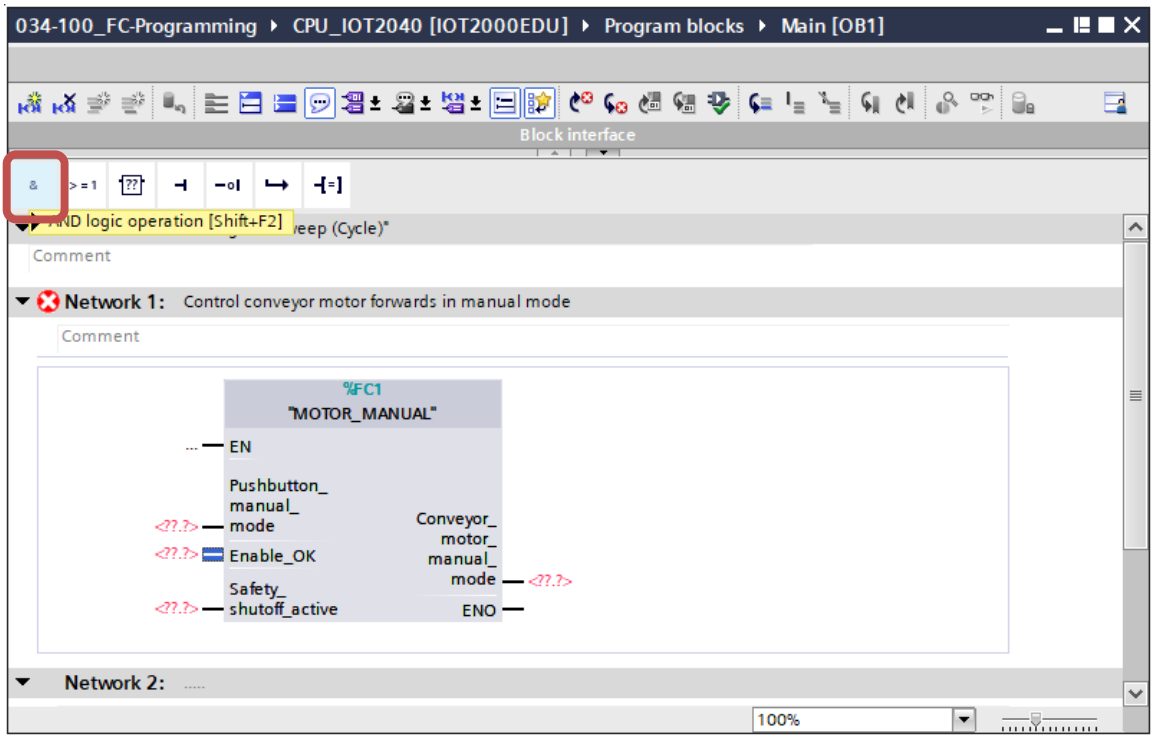
→ Use drag & drop to move your "MOTOR_MANUAL [FC1]" function onto the green line in Network 1.



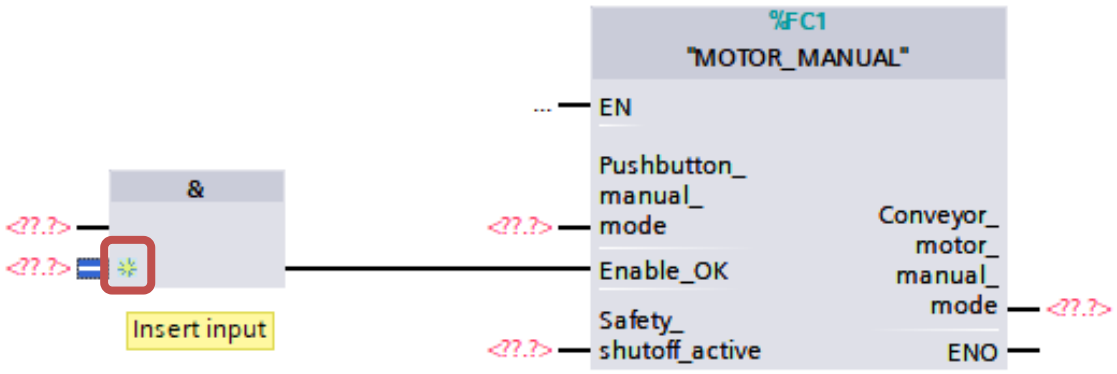
→ A block with the interface you defined and connections EN and ENO is inserted in Network 1.



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

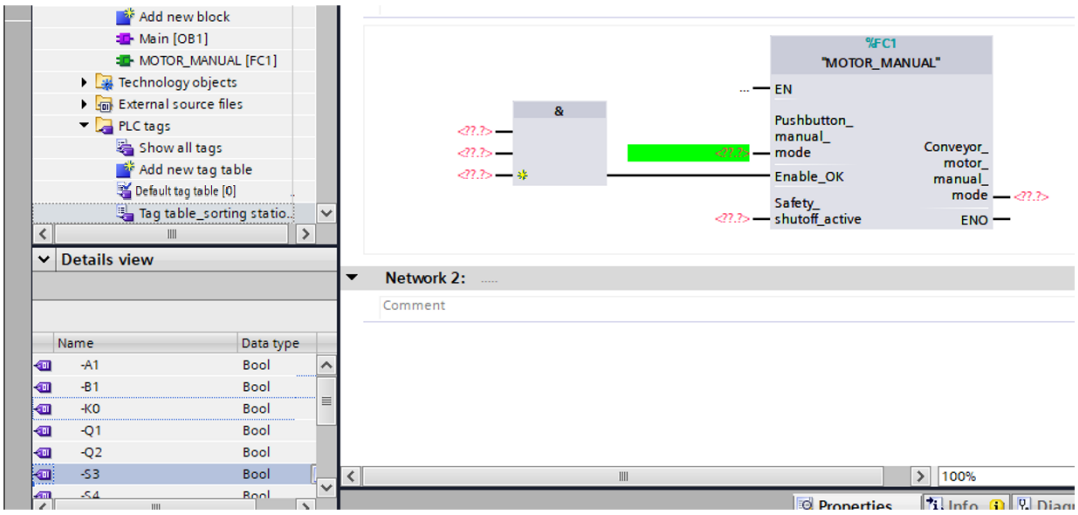


→ Click on the yellow star of the AND element to add another input. (→)

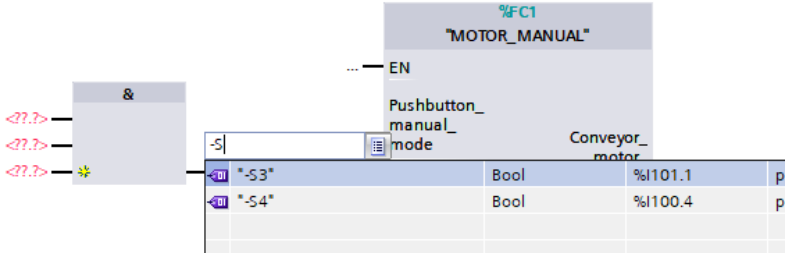


→ To connect the block to the global tags from "Tag_table_sorting_station", we have two options:

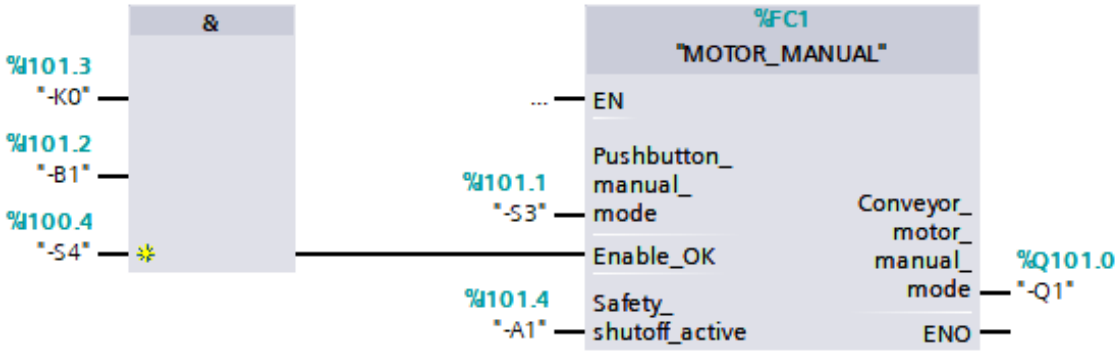
→ Either select the "Tag_table_sorting_station" in the project tree and drag the desired global tag from the Details view to the interface of FC1 (→ Tag_table_sorting_station → Details view. → -S3 → Pushbutton_manual_mode)



Or enter the starting letters (e.g. "S") of the desired global tag for <??.?> and select the global input tag "-S3" (%I101.1) from the displayed list. (→ Pushbutton_manual_mode → -S → -S3)

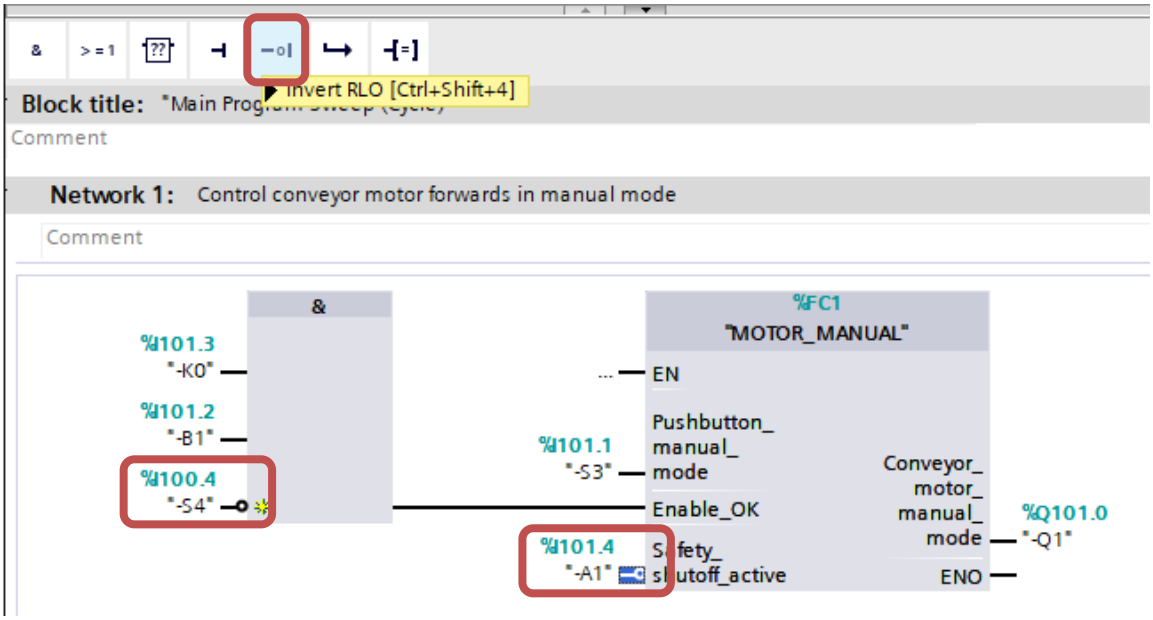


→ Insert the other input tags "-S3", "-K0", "-B1", "-S4" and "-A1" and insert output tag "-Q1" (%Q101.0) at output "Conveyor_motor_manual_mode".



→ Negate the queries of input tags "-S4" and "-A1" by selecting them and clicking 

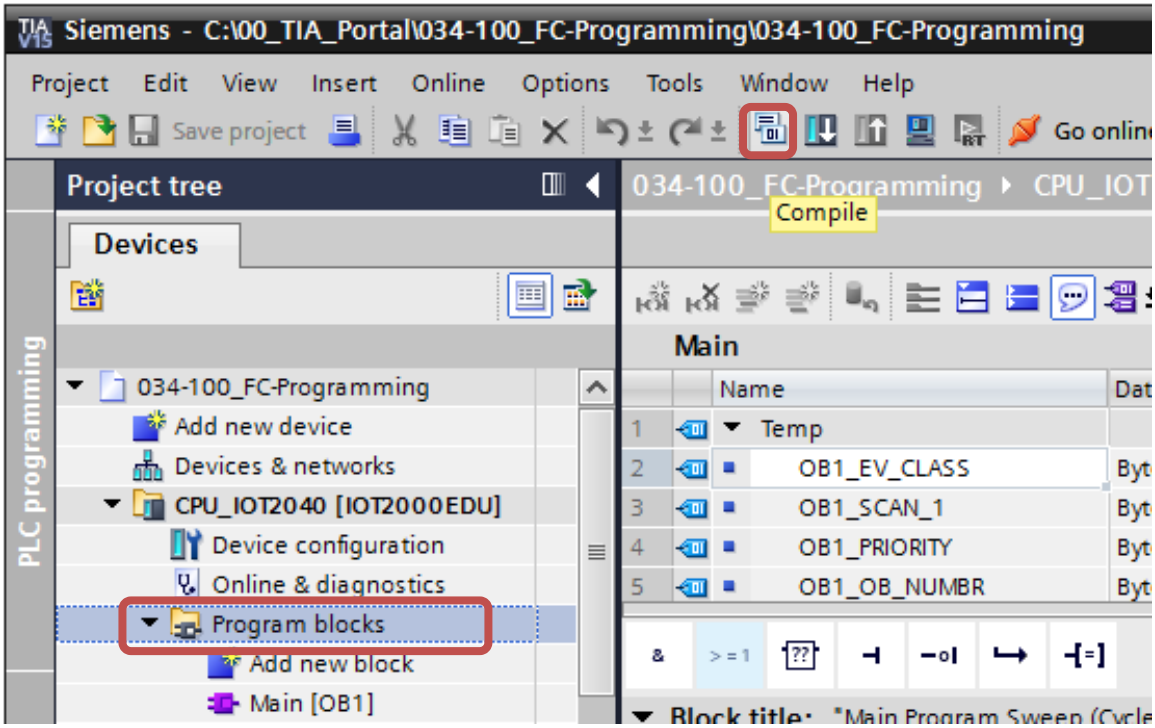
(→ -S4 →  → -A1 → )



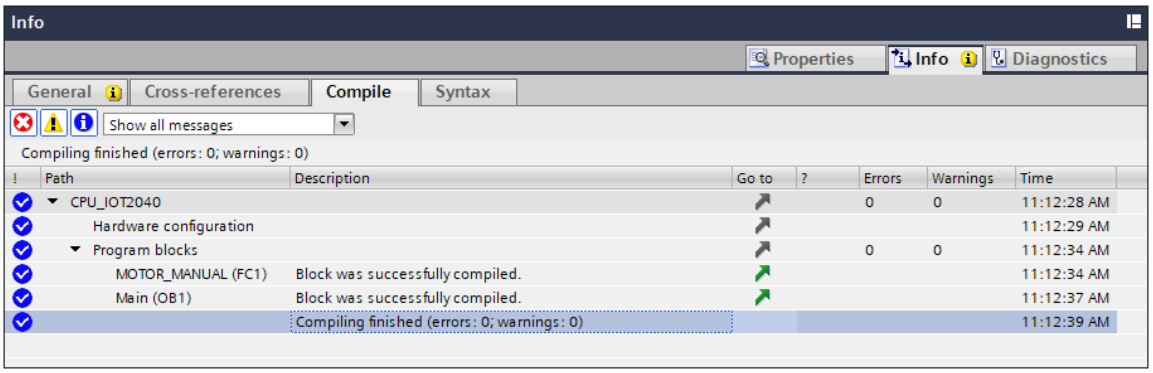
7.9 Saving and compiling the program

→ 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 →)



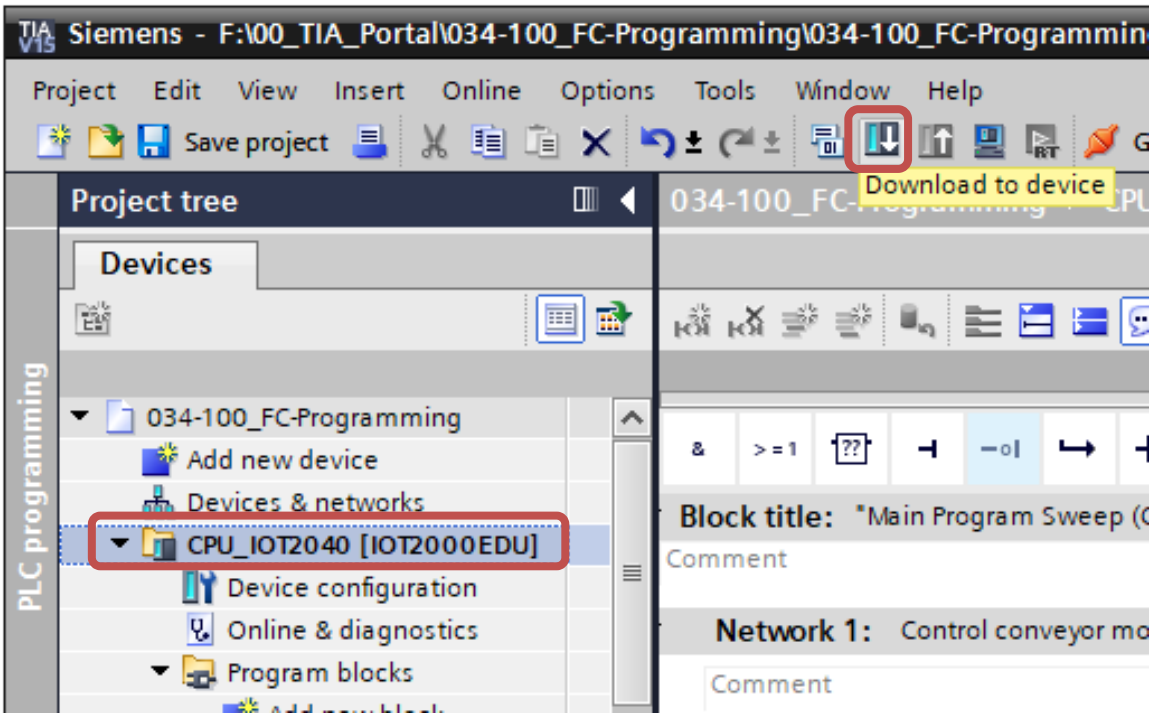
→ The "Info", "Compile" area shows which blocks were successfully compiled.



7.10 Downloading the program

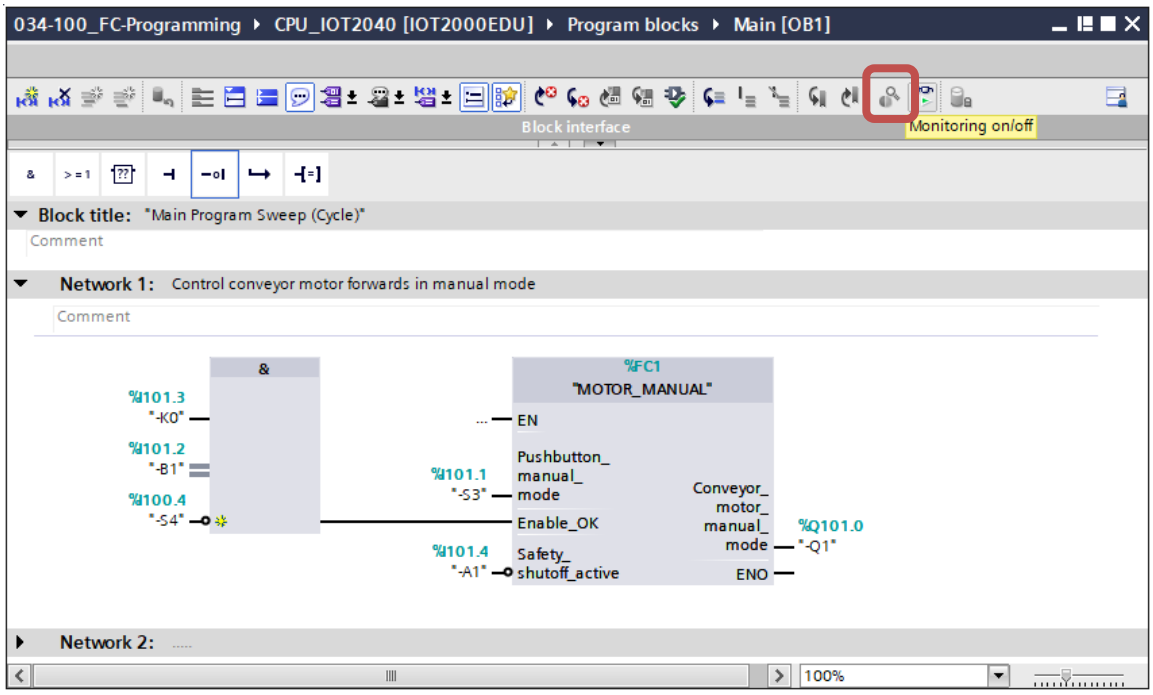
→ After successful compilation, the complete controller with the created program, as previously described in the modules for hardware configuration, can be downloaded.

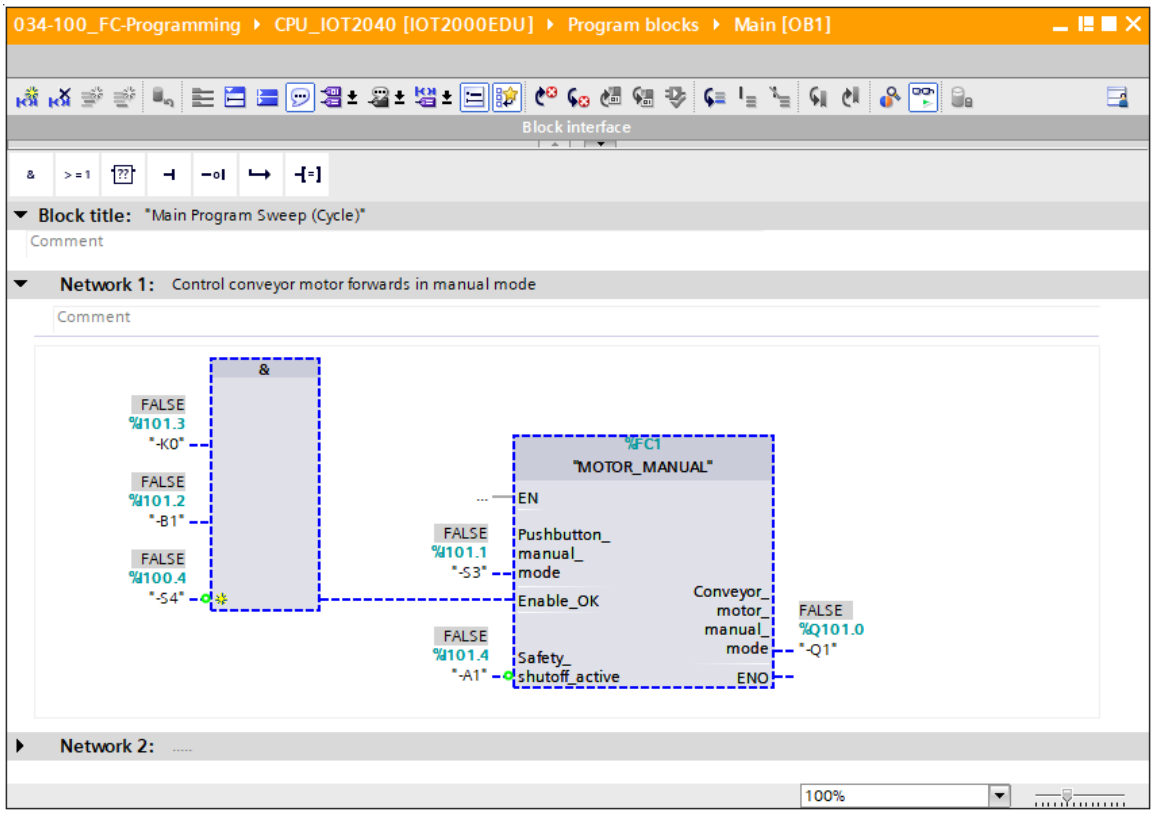
(→ )



7.11 Monitoring program blocks

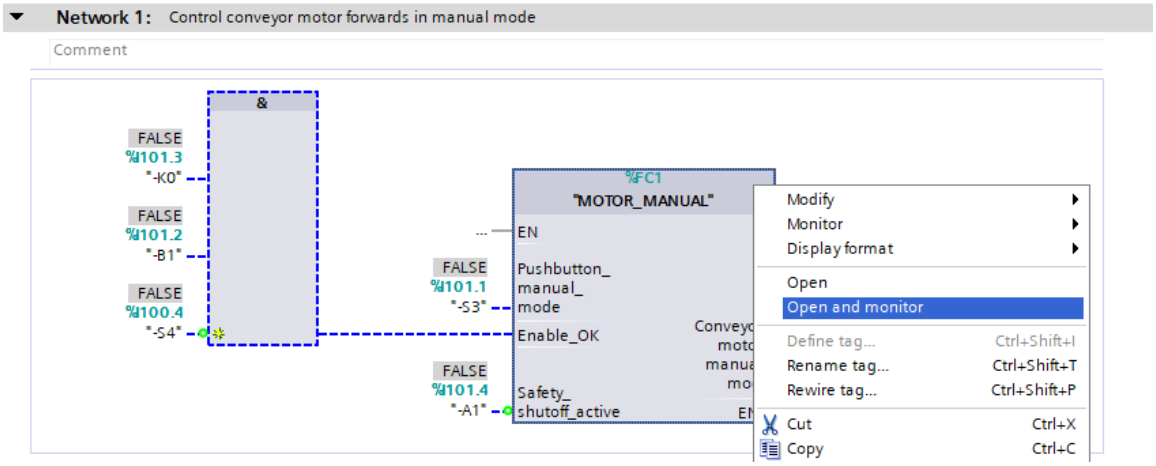
→ The desired block must be open for monitoring the downloaded program. The monitoring can then be activated/deactivated by clicking the  icon. (→ Main [OB1] → )

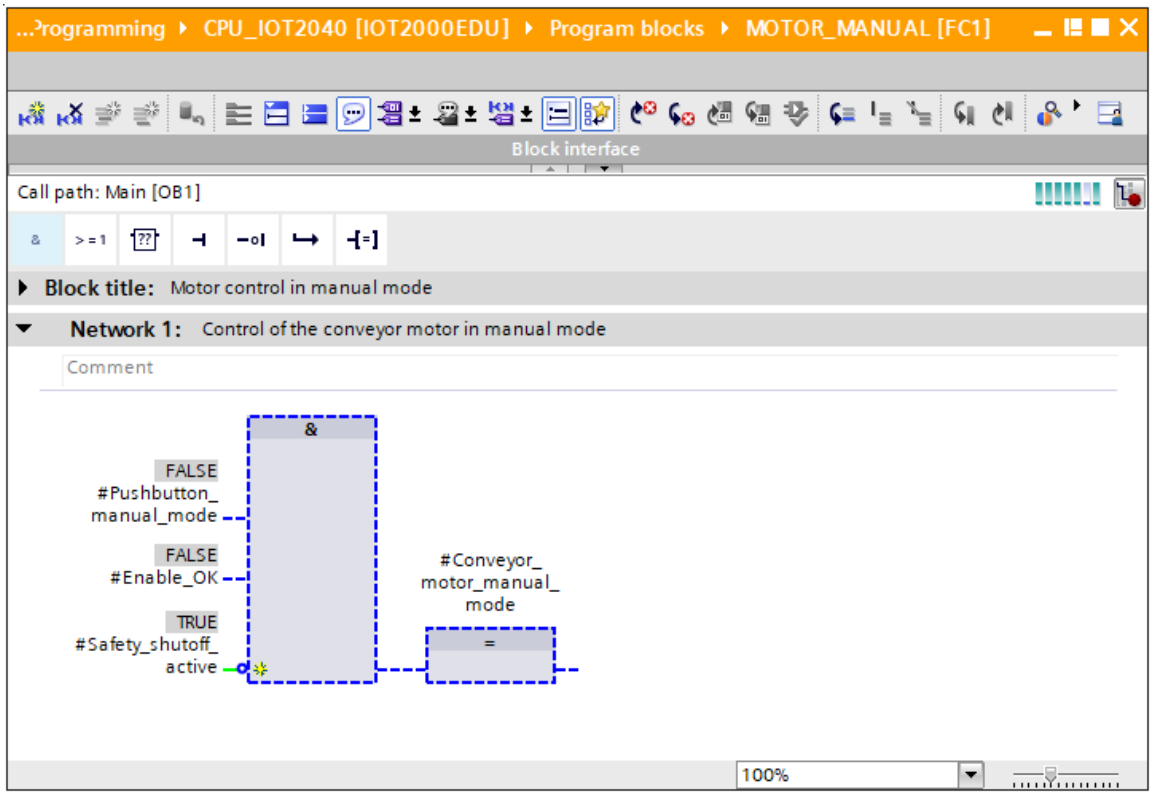




Note: *Monitoring here is signal-related and controller-dependent.*
The signal states at the terminals are indicated with TRUE or FALSE.

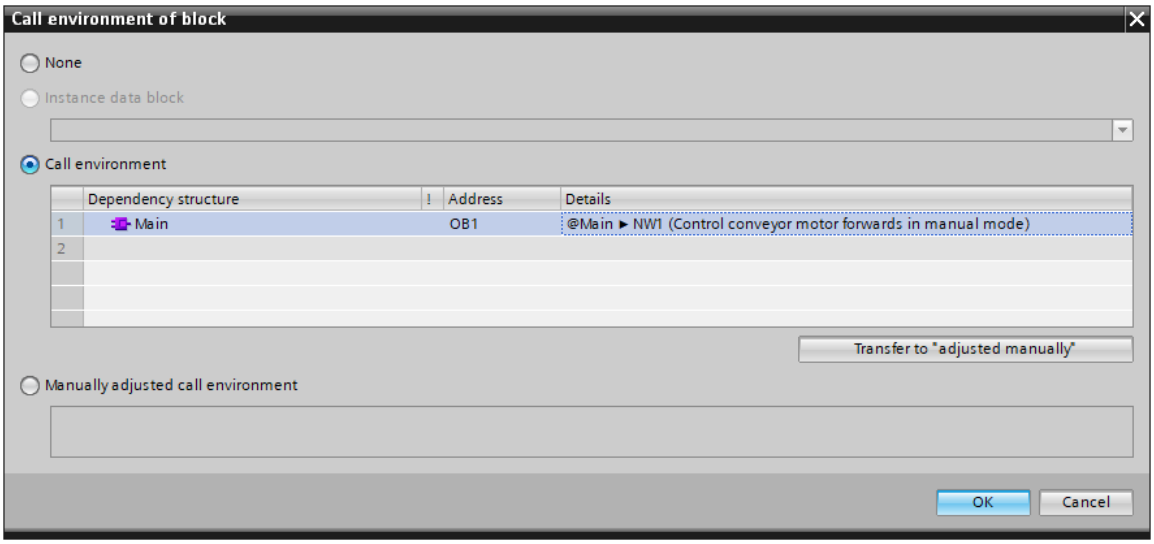
→ The "MOTOR_MANUAL" [FC1] function called in the "Main [OB1]" organization block can be selected directly for "Open and monitor" after right-clicking (→ "MOTOR_MANUAL" [FC1] → Open and monitor)





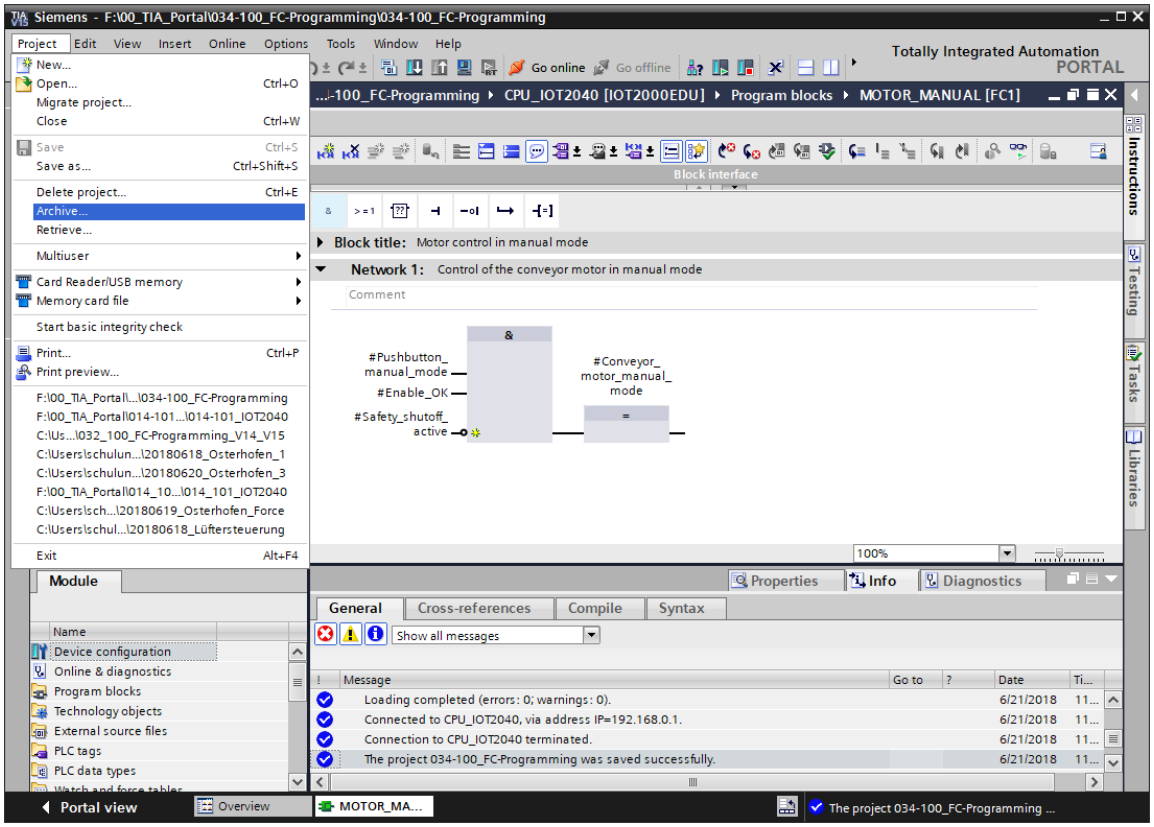
Note: *Monitoring here is function-related and controller-independent. The actuation of sensors and the station status are shown here with TRUE or FALSE.*

→ If a particular point of use of the "MOTOR_MANUAL" [FC1] function is to be monitored, the call environment can be selected using the icon. (→ → Call environment → OK)



7.12 Archiving the project

→ As the final step, we want to archive the complete project. Click on the command → "Archive …" 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 → 034-100_FC Programming. → Save)



7.13 Checklist

No.	Description	Checked
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 Activate conveyor manual mode forwards (-S3 = 1) Conveyor motor forwards fixed speed (-Q1 = 1)	
4	Same as 3 but activate EMERGENCY STOP (-A1 = 0) → -Q1 = 0	
5	Same as 3 but switch off station (-K0 = 0) → -Q1 = 0	
6	Same as 3 but cylinder not retracted (-B1 = 0) → -Q1 = 0	
7	Same as 3 but also activate conveyor manual mode backwards (-S4 = 1) → -Q1 = 0	
8	Project successfully archived	

8 Exercise

8.1 Task – Exercise

In this exercise, the following function of the process description sorting station is also to be planned, programmed and tested:

– Manual mode – Control conveyor motor forwards in manual mode

8.2 Technology diagram

Here, you see the technology diagram for the task.

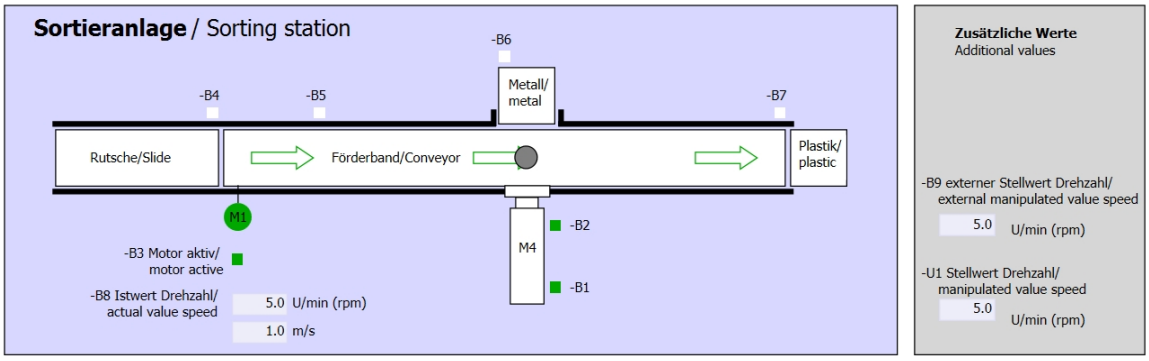


Figure 10: Technology diagram

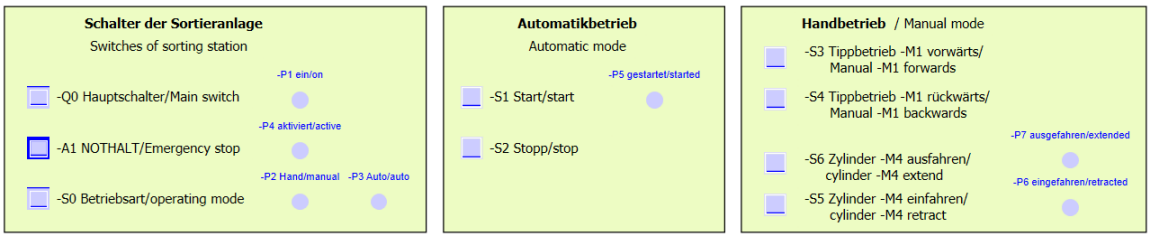


Figure 11: Control panel

8.3 Reference list

The following signals are required as operands for this task.

DI	Type	Identifier	Function	NC/NO
I 101.4	BOOL	-A1	Return signal emergency stop OK	NC
I 101.3	BOOL	-K0	Station "ON"	NO
I 101.2	BOOL	-B1	Sensor cylinder -M4 retract	NO
I 101.1	BOOL	-S3	Pushbutton manual mode conveyor -M1 forwards	NO
I 100.4	BOOL	-S4	Pushbutton manual mode conveyor -M1 backwards	NO

DQ	Type	Identifier	Function	
Q 101.0	BOOL	-Q1	Conveyor motor -M1 forwards fixed speed	
Q 100.7	BOOL	-Q2	Conveyor motor -M1 backwards fixed speed	

Legend for reference list

DI	Digital input	DQ	Digital output
AI	Analog input	AQ	Analog output
I	Input	Q	Output
NC	Normally Closed		
NO	Normally Open		

8.4 Planning

Plan the implementation of the task on your own.

8.5 Checklist – Exercise

No.	Description	Checked
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 Activate conveyor manual mode backwards (-S4 = 1) Conveyor motor backwards fixed speed (-Q2 = 1)	
4	Same as 3, but activate EMERGENCY OFF (-A1 = 0) → -Q2 = 0	
5	Same as 3, but switch off station (-K0 = 0) → -Q2 = 0	
6	Same as 3, but cylinder not retracted (-B1 = 0) → -Q2 = 0	
7	Same as 3, but also activate conveyor manual mode forwards (-S3 = 1) → -Q1 = 0 and -Q2 = 0	
8	Project successfully archived	

9 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/iot2000

Further information

Siemens Automation Cooperates with Education
siemens.com/sce

SCE Training Curriculums
siemens.com/sce/module

SIMATIC IOT2000
siemens.com/sce/IOT2000

SCE Trainer Packages
siemens.com/sce/tp

IOT2000 Forum
siemens.com/iot2000-forum

SCE Contact Partners
siemens.com/sce/contact

Digital Enterprise
siemens.com/digital-enterprise

Industry 4.0
siemens.com/future-of-manufacturing

Totally Integrated Automation (TIA)
siemens.com/tia

TIA Portal
siemens.com/tia-portal

SIMATIC Controller
siemens.com/controller

SIMATIC Technical Documentation
siemens.com/simatic-docu

Industry Online Support
support.industry.siemens.com

Product catalog and online ordering system Industry Mall
mall.industry.siemens.com

Siemens AG
Digital Factory
P.O. Box 4848
90026 Nuremberg
Germany

siemens.com/sce