

SCE Training Curriculum for Integrated Automation Solutions Totally Integrated Automation (TIA)

Siemens Automation Cooperates with Education

TIA Portal Module 010-020 Block Types for the SIMATIC S7-1200

Cooperates
with Education

Automation

SIEMENS

Matching SCE training packages for these training curriculums

- **SIMATIC S7-1200 AC/DC/RELAY 6er "TIA Portal"**
Order number: 6ES7214-1BE30-4AB3
- **SIMATIC S7-1200 DC/DC/DC 6er "TIA Portal"**
Order number 6ES7214-1AE30-4AB3
- **SIMATIC S7-SW for Training STEP 7 BASIC V11 Upgrade (for S7-1200) 6er "TIA Portal"**
Order number 6ES7822-0AA01-4YE0

Please note that these training packages are replaced with successor packages when necessary.
An overview of the currently available SCE packages is provided under: [siemens.com/sce/tp](https://www.siemens.com/sce/tp)

Continued Training

For regional Siemens SCE continued training, please contact your regional SCE contact person
[siemens.com/sce/contact](https://www.siemens.com/sce/contact)

Additional information regarding SCE

[siemens.com/sce](https://www.siemens.com/sce)

Information regarding Usage

This 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 education facilities and R&D facilities. Siemens AG does not guarantee the contents.

This document is to be used only for initial training on Siemens products/systems; i.e., it can be copied entirely or partially and given to those being trained for usage within the scope of their training. Passing on as well as copying this training curriculum and sharing its content is permitted within public training and advanced training facilities for training purposes.

Exceptions require written permission by the Siemens AG contact person: Roland Scheuerer
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.

Usage for industrial customer courses is explicitly not permitted. We do not consent to the training curriculums being used commercially.

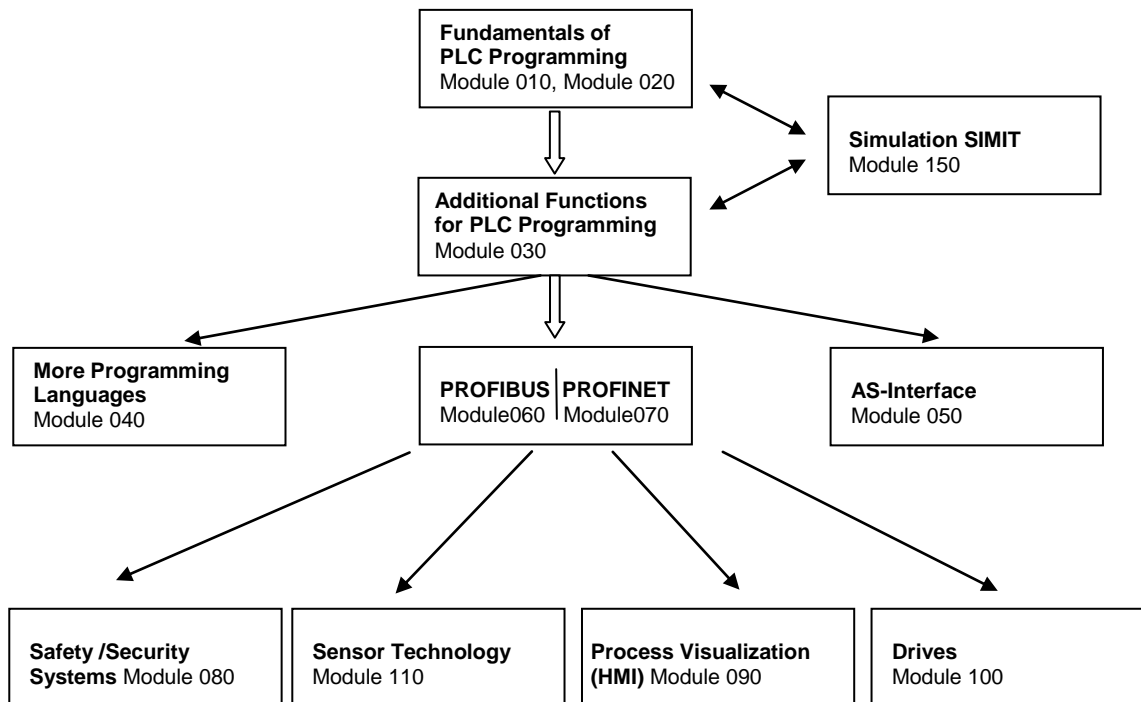
We wish to thank the Michael Dziallas Engineering Corporation and all other involved persons for their support during the preparation of this training curriculum.

Contents

1.	Preface.....	4
2.	Notes on Programming the SIMATIC S7-1200	6
2.1	Automation System SIMATIC S7-1200	6
2.2	Programming Software STEP 7 Professional V11 (TIA Portal V11)	6
3.	Block Types for the SIMATIC S7-1200.....	7
3.1	Linear Programming	7
3.2	Structured Programming.....	8
3.3	User Blocks for the SIMATIC S7-1200	9
3.3.1	Organization Blocks	10
3.3.2	Functions.....	11
3.3.3	Function Blocks	11
3.3.4	Data Blocks	12
4.	Sample Task Function Block for Conveyor Control	13
5.	Programming the Conveyor Control for the SIMATIC S7-1200	14

1. Preface

Regarding its content, module SCE_EN_010-020 is part of the training unit '**Basics of PLC Programming**' and represents a **fast entry point** for programming the SIMATIC S7-1200 with the TIA Portal.



Training Objective:

In module SCE_EN_010-020, the reader will become acquainted with the different blocks used for programming the SIMATIC S7-1200 with the programming tool TIA Portal. The module explains the various block types and shows in the steps listed below how to generate a program in a function block.

- Generating the function block
- Defining internal variables
- Programming with internal variables in the function block
- Calling and parameterizing the function block in OB1

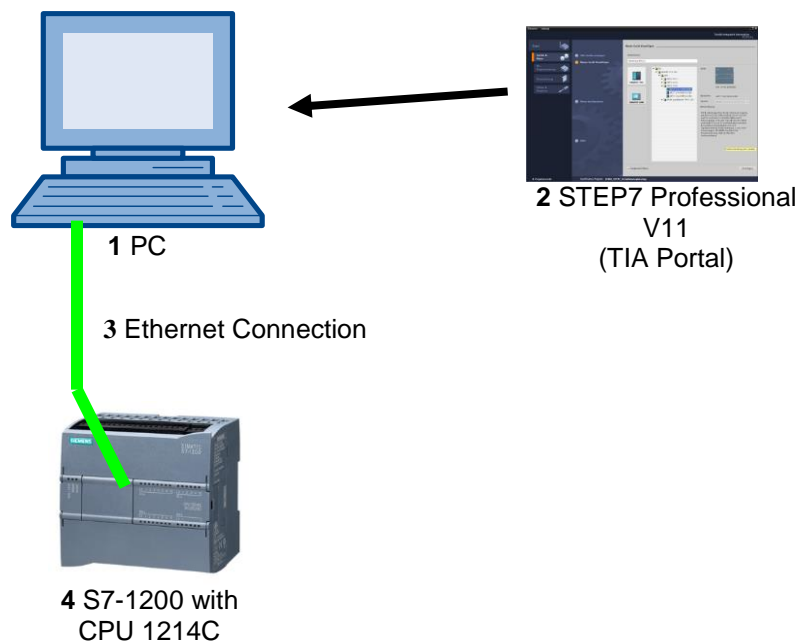
Prerequisites:

To successfully process this module, the following knowledge is assumed:

- How to operate Windows
- Basics of PLC programming with the TIA Portal (for example, Module 010-010 'Startup' Programming of the SIMATIC S7-1200 with TIA Portal V11)

Hardware and software required

- 1 PC Pentium 4, 1.7 GHz 1 (XP) – 2 (Vista) GB RAM, free disk storage approx. 2 GB operating system Windows XP Professional SP3/Windows 7, Professional/Windows 7 Enterprise/Windows 7 Ultimate/Windows 2003 Server R2/Windows Server 2008 Premium SP1, Business SP1, Ultimate SP
- 2 Software STEP7 Professional V11 SP1 (Totally Integrated Automation (TIA) Portal V11)
- 3 Ethernet connection between PC and CPU 315F-2 PN/DP
- 4 PLC SIMATIC S7-1200; for example, CPU 1214C.
The inputs have to be brought out to a panel.



2. Notes on Programming the SIMATIC S7-1200

2.1 Automation System SIMATIC S7-1200

The SIMATIC S7-1200 automation system is a modular mini-control system for the lower and medium performance range.

An extensive module spectrum is available for optimum adaptation to the automation task.

The S7 controller consists of a power supply, a CPU, and input and output modules for digital and analog signals.

If necessary, communication processors and function modules are used for special tasks, such as step motor control.

With the S7 program, the programmable logic controller (PLC) monitors and controls a machine or a process; the IO modules are polled in the S7 program by means of the input addresses (%I), and addressed by means of output addresses (%Q).

The system is programmed with the software STEP 7.

2.2 Programming Software STEP 7 Professional V11 (TIA Portal V11)

The software STEP 7 Professional V11 (TIA Portal V11) is the programming tool for the automation systems

- SIMATIC S7-1200
- SIMATIC S7-300
- SIMATIC S7-400
- SIMATIC WinAC

With STEP 7 Professional V11, the following functions can be utilized to automate a plant:

- Configuring and parameterizing the hardware
- Defining communication
- Programming
- Testing, commissioning and service with the operating/diagnostic functions
- Documentation
- Generating visual displays for the SIMATIC basic panels with integrated WinCC Basic
- With additional WinCC packages, visualization solutions for PCs and other panels can be generated

All functions are supported with detailed online help.

3. Block Types for the SIMATIC S7-1200

For the SIMATIC S7-1200, the program is written in so-called blocks.

As a matter of standard, the organization block Main[OB1] is already provided.

It represents the interface to the CPU's operating system, is called by it automatically, and processed cyclically.

If the control task is extensive, we subdivide the program into small program blocks that are arranged according to functions, and easy to follow.

These blocks in turn are called from the organization blocks. At the end of the block, we jump back to the organization block that performed the call, exactly to the line behind the call.

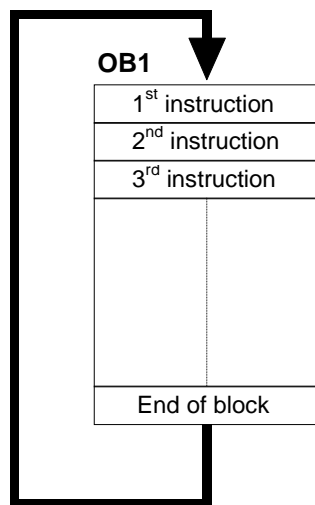
3.1 Linear Programming

For linear programming, the instructions are stored in a block and processed in the sequence in which they are stored in the program memory. When the end of the program (end of block) is reached, program processing begins again from the start.

This is called cyclical processing.

The time a device needs to process all instructions once is called cycle time.

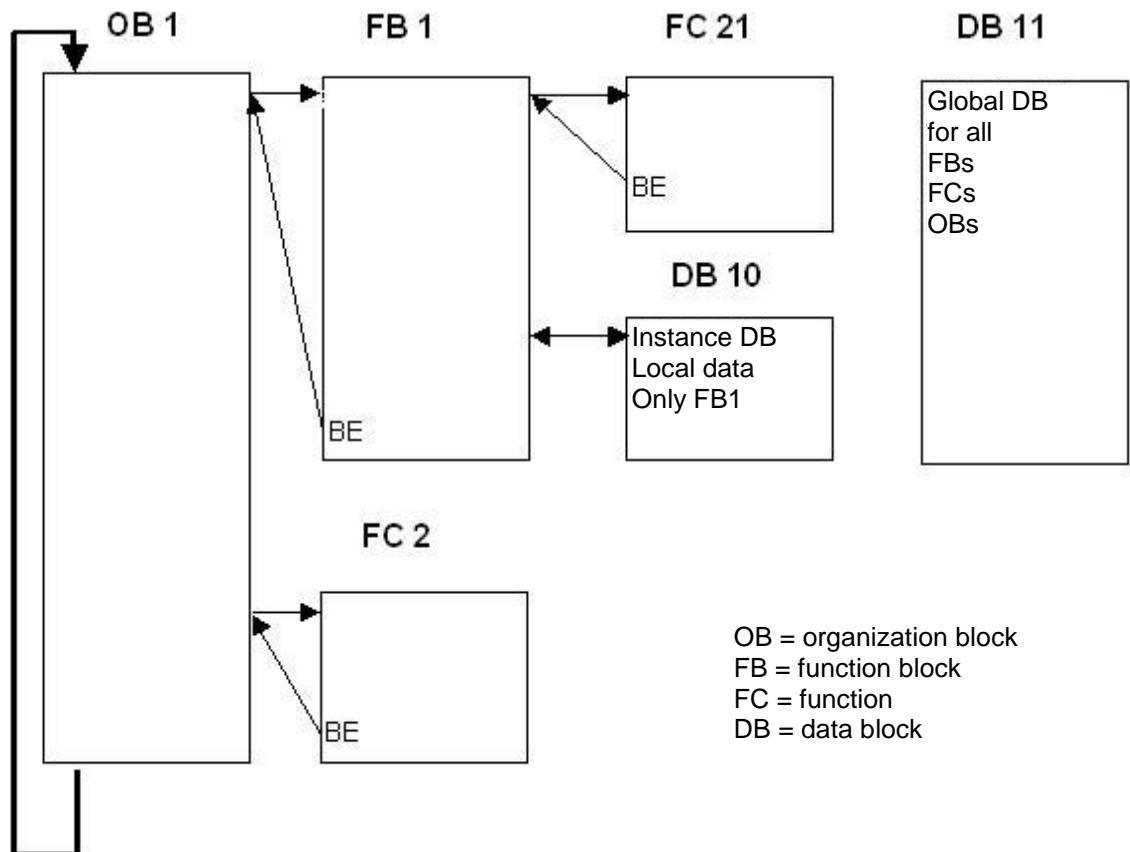
Linear program processing is usually used for simple control tasks that are not too extensive; it can be implemented in one OB.



3.2 Structured Programming

If the control task is extensive, we subdivide it into small program blocks arranged according to functions and that are easy to follow. The advantage: program parts can be tested individually and if they work, they can be merged into a total function.

The primary block has to call the program blocks. If the end of the block (BE) is recognized, the program continues to be processed in the calling block behind the call.



3.3 User Blocks for the SIMATIC S7-1200

The following user blocks are provided for structured programming:

- OB (organization block):

OBs are called by the operating system cyclically and are the interface between the user program and the operating system. In this OB, the PLC's control unit is informed by means of block call commands as to which program blocks it has to process.

- FB (function block):

For each call (instance), the FB needs an assigned memory area. When an FB is called, a data block (DB) for example can be assigned to it as instance DB.

The data in this instance DB is then accessed by means of the variables of the FB.

Different memory areas have to be assigned to an FB if it called several times.

Additional FBs and FCs can be called in a function block in turn.

- FC (function):

A FC does not have an assigned memory area. The local data of a function is lost after the function is processed.

Additional FBs and FCs can be called in a function in turn.

- DB (data block):

DBs are used to provide memory for the data variables. There are two types of data blocks: global DBs where all OBs, FBs and FCs can read the stored data or themselves can write data into the DB, and instance DBs that are assigned to a certain FB.

Note:

If during FC and FB programming only internal variables were used, they can be used multiple times in the form of standard blocks. They can then be called any number of times. However, the FBs have to be assigned a memory area, a so-called instance (for example, a DB), for each call.

3.3.1 Organization Blocks

Organization blocks (OBs) are the interface between the operating system and the user program. They are called by the operating system, and control the following processes:

Startup behavior of the automation system

- Cyclical program processing
- Alarm-controlled program processing
- Error handling

You can program the organization blocks as desired, and thus determine the CPU's behavior.

You have various options for using organization blocks in your program:

- **Startup OB, Cycle OB, Timing Error OB and Diagnosis OB:**

You can simply insert and program these organization blocks in your project. You don't have to assign parameters to them, nor do you have to call them.

- **Process Alarm OB and Time Interrupt OB:**

These organization blocks have to be parameterized after you inserted them in your program. In addition, process alarm OBs can be assigned to an event at execution time using the instruction ATTACH, or separated again with DETACH.

- **Time Delay Interrupt OB:**

The time delay interrupt OB can be inserted in your project and programmed. In addition, you have to call it in the user program with the instruction SRT_DINT. Parameterization is not necessary.

Start Information

When some organization blocks are started, the operating system reads out information that can be evaluated in the user program.

This can be very helpful for diagnosing errors.

Whether and which information is read out is provided in the descriptions of the organization blocks.

3.3.2 Functions

A function contains a program that is executed when another code block calls the function.

Functions (FCs) are code blocks without memory. The data of the temporary variables is lost after the function is processed. Global data blocks can be used to store FC data.

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

- Returning function values to the calling block; for example, in the case of mathematical functions
- Executing technological functions; for example, individual controls with binary operations

Also, a function can be called several times at different locations within a program. This facilitates programming complicated repetitious functions.

3.3.3 Function Blocks

Function blocks contain subprograms that are always executed when a function block is called by another code block.

Function blocks are code blocks that store their values in instance data blocks so that these values are available also after the block is processed.

Store your input, output and in/out parameters permanently in instance data blocks. They will still be available after the block is processed. For that reason, they are also called blocks with 'memory'.

Function blocks are used for tasks that can't be implemented with functions:

- Always when timers and counters are needed in the blocks (refer to module M3)
- Always when information has to be stored in the program; for example, when preselecting an operating mode with a button.

A function block can also be called multiple times at different locations within a program. This facilitates programming repetitious, complicated functions.

Instances of function blocks

The call of a function block is called an instance.

To each instance of a function block, a memory area is assigned that contains the data the function block uses for processing. This memory is provided by data blocks that the software generates automatically. It is also possible to provide memory for several instances in one data block as **multi-instance**.

3.3.4 Data Blocks

In contrast to code blocks, data blocks do not contain instructions but are used to store user data. That means, the data blocks contain variable data that the user program uses for processing.

Global data blocks store data that can be used by all other blocks.

The maximum size of data blocks varies, depending on the CPU. The structure of global data blocks can be specified as required.

Application examples are:

- Storing the information of a warehouse system. "Which product is located where"
- Storing recipes for certain products

Every function block, every function or every organization block can read data from a global data block, or write data into a global data block. This data is retained in the data block even when the data block is exited.

The call of a function block is referred to as an instance. To each call of a function block with parameter transfer, an **instance data block** is assigned that serves as data storage. In it, the actual parameters and the static data of the function blocks are stored.

The maximum size of the instance data blocks varies, depending on the CPU. The variables declared in the function block determine the structure of the instance data block.

A global data block and an instance data block can be open at the same time.

4. Sample Task Function Block for Conveyor Control

When blocks are to be generated that are working in any program like a "Black Box" as it were, they have to be programmed by using variables. In this case, the following rule applies: that in these blocks, no absolute-addressed inputs/outputs, flags etc. must be used. Within the block, only variables and constants are used.

In the example below, a function block is to be generated with a variable declaration containing a conveyor control that is dependent on the operating mode.

With button 'S1', the operating mode 'Manual' and with button 'S2' the operating mode 'Automatic' can be selected.

In the operating mode 'Manual', the motor is switched on as long as button 'S3' is operated, whereby button 'S4' must not be operated.

In the operating mode 'Automatic', the conveyor motor is switched on with button 'S3' and switched off with button 'S4' (break contact).

Assignment list:

Address	Symbol	Comment
%I 0.0	S1	Button operating mode Manual S1 NO
%I 0.1	S2	Button operating mode Automatic S2 NO
%I 0.2	S3	On button S3 NO
%I 0.3	S4	Off button S4 NC
%Q 0.2	M1	Conveyor motor M1

Note: The Off button S4 is a break contact here in order to ensure wire break safety. That means: if there is a wire break at this button, the system stops automatically. Otherwise, it could not stop if there were a wire break. For that reason, in control engineering all Stop buttons, Off buttons/switches have to be designed as break contacts.

5. Programming the Conveyor Control for the SIMATIC S7-1200

The project is managed and the components are programmed with the software '**Totally Integrated Automation Portal**'.

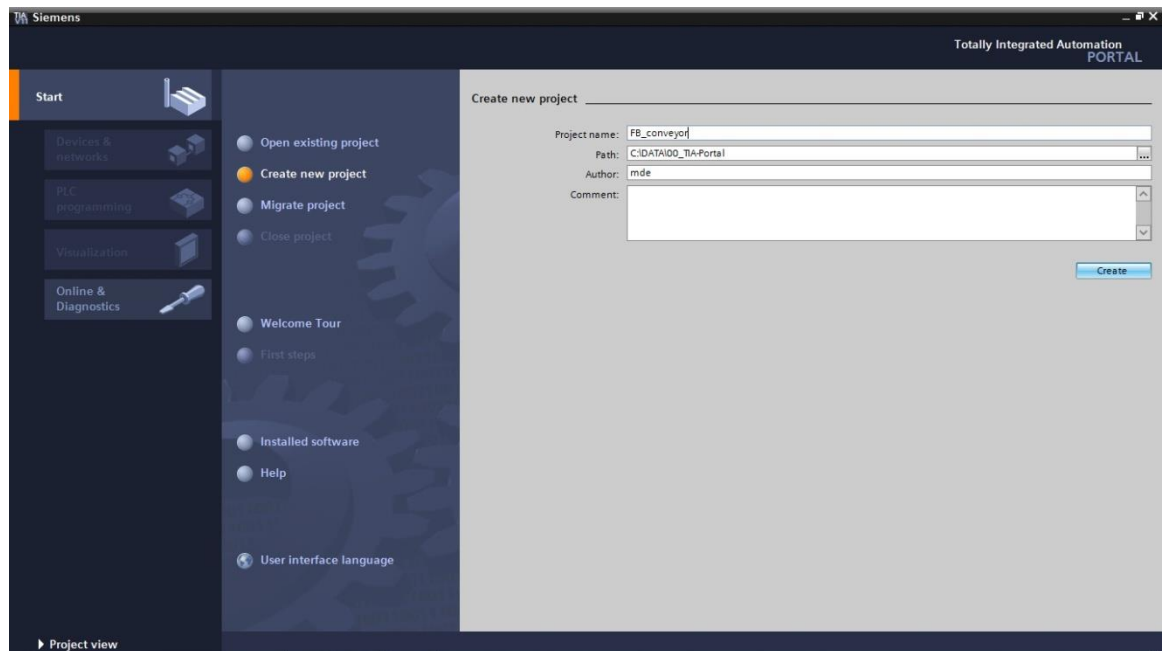
Here, under a uniform interface, the components such as controller, visual display and networking of the automation solution are set up, parameterized and programmed. Online tools are provided for error diagnosis

In the steps below, a project can be set up for the SIMATIC S7-1200 and the solution for a task can be programmed:

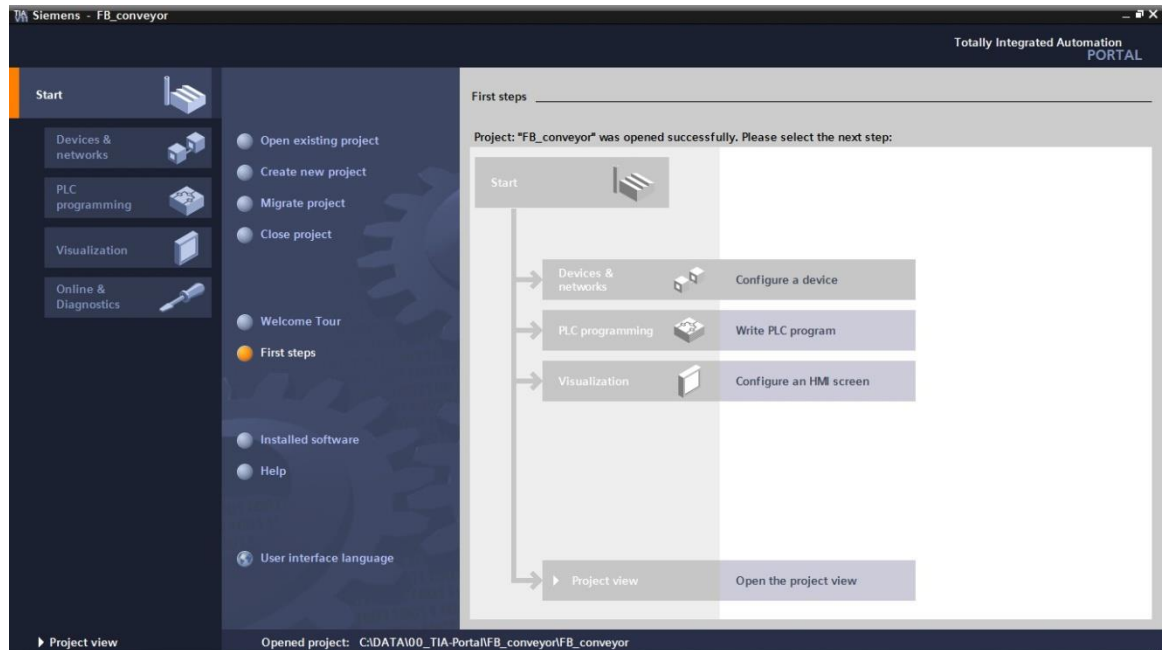
1. The central tool is the '**Totally Integrated Automation Portal**' that we call here with a double click (→ Totally Integrated Automation Portal V11)



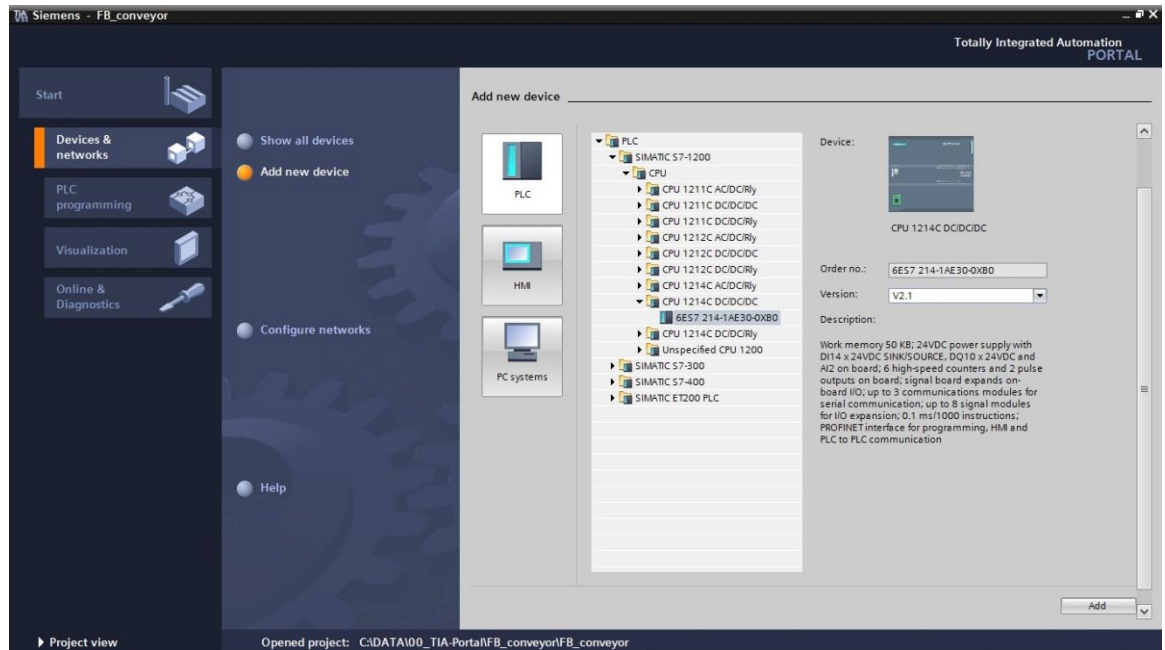
2. Programs for the SIMATIC S7-1200 are managed in projects. Such a project is now set up in the portal view → Create new project → FB_conveyor → Create)



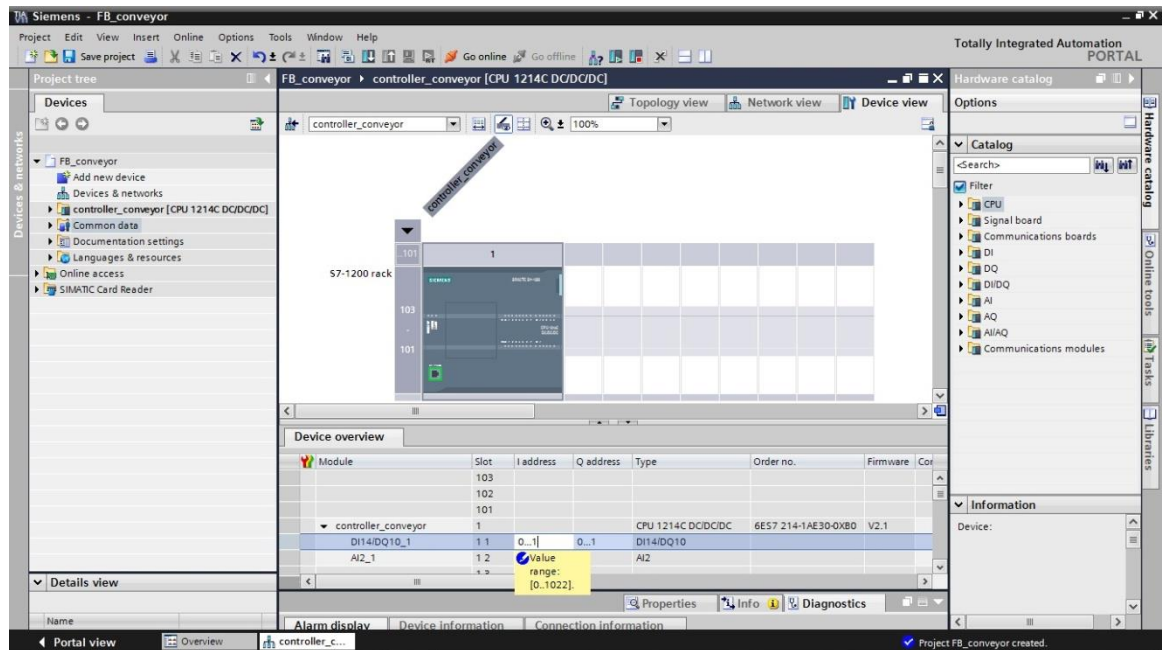
3. **'First Steps'** are suggested regarding the configuration. First, we want to **'Configure a device'**. (→ First Steps → Configure a device)



4. Next, we 'Add new device' with the 'Device name conveyor control'. To this end, we select from the catalog the 'CPU1214C' with the matching order number. (→ Add new device → conveyor control → CPU1214C → 6ES7 → Add)

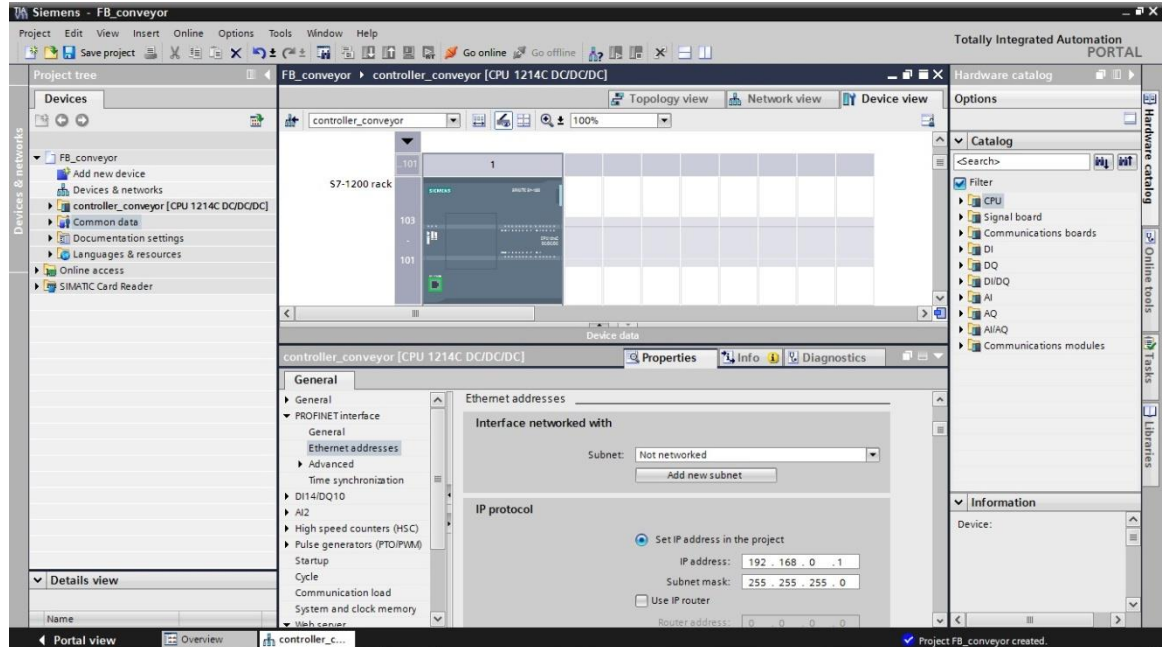


5. Now, the software automatically switches to the project view with the opened hardware configuration. Here, additional modules from the hardware catalog (to the right) can be added and in the '**Device overview**', the addresses of the inputs and outputs can be set. The integrated inputs of the CPU have the addresses %I0.0 to %I1.5 and the integrated outputs have the addresses %Q0.0 to %Q1.1 (→ Device overview → DI14/DO10 → 0...1)



6. So that the software later accesses the correct CPU, its IP address and the subnet mask have to be set.

(→ Properties → General → PROFINET interface → IP address: 192.168.0.1 → Subnet mask: 255.255.255.0)

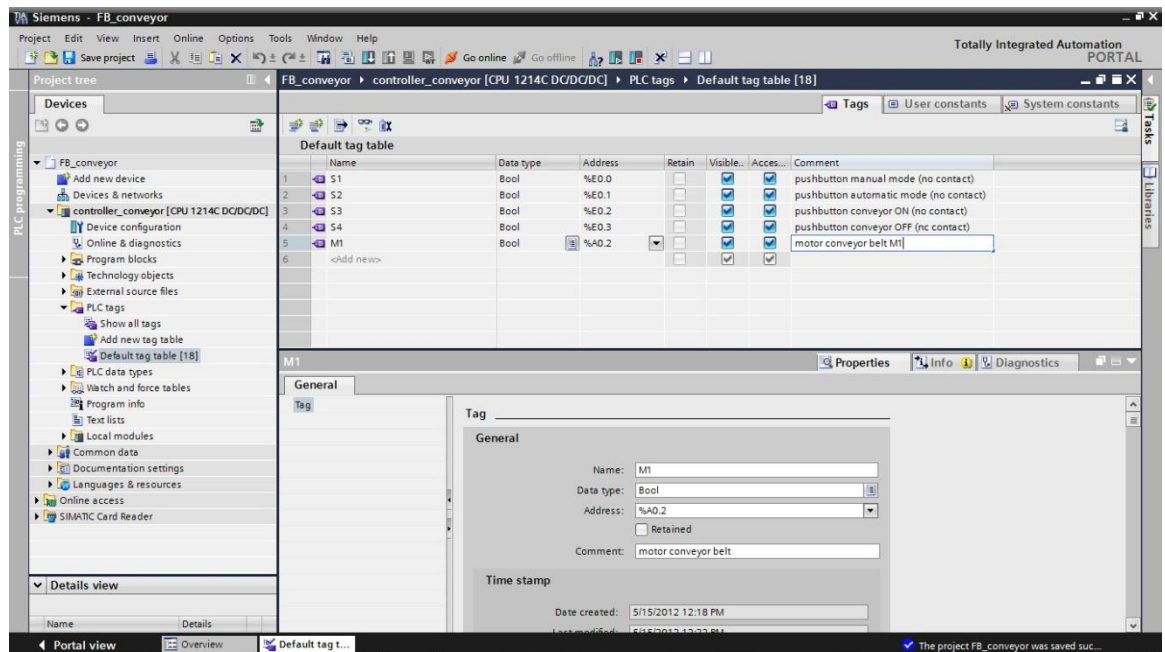


7. Since modern programming is not carried out with absolute addresses but with variables, the **global PLC tags** have to be specified here.

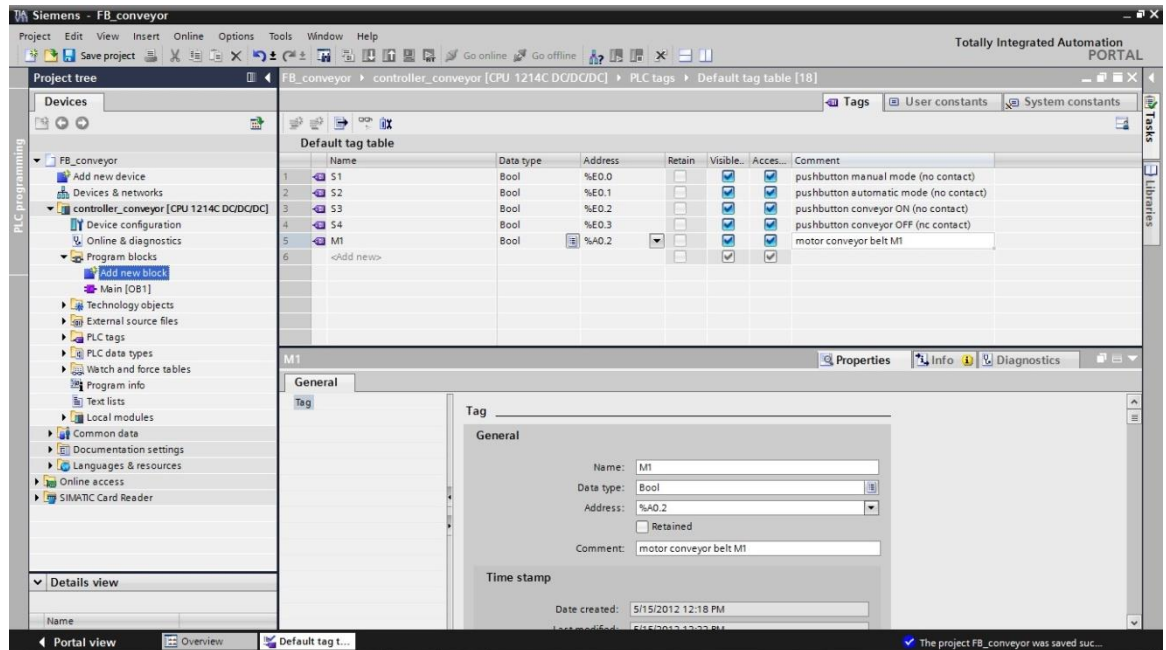
These global PLC variables are descriptive names with comments for those inputs and outputs that are used in the program. Later, during programming, this name is used to access the global PLC tags.

These global tags can be used in the entire program, in all blocks.

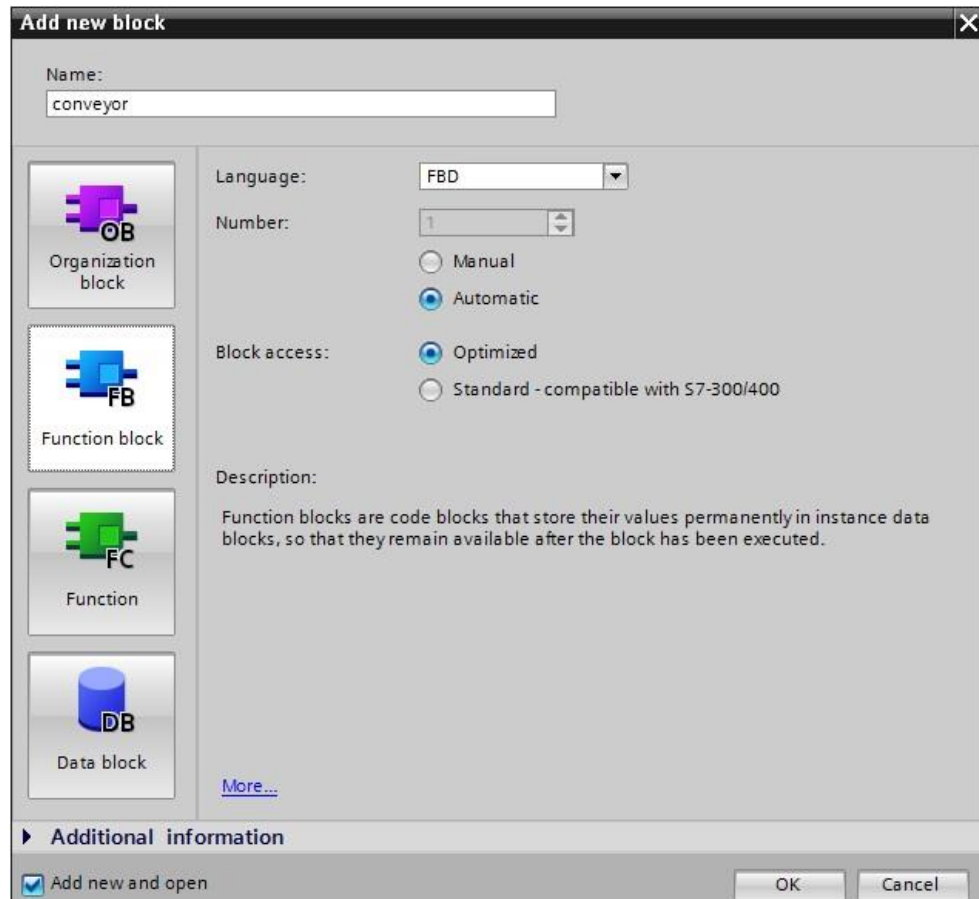
To this end, select in Project Navigation the '**controller_conveyorCPU1214C DC/DC/DC**' and then '**PLC tags**'. Open the table '**PLC tags**' with a double click and enter the names for the inputs and outputs as shown below (→ controller_conveyor[CPU1214C DC/DC/DC] → PLC tags→ PLC tags)



8. To generate the function block FB1, first select '**controller_conveyor[CPU1214C DC/DC/DC]**' in project navigation, and then '**Program blocks**'. Now, double click on '**Add new block**' (→ controller_conveyor[CPU1214C DC/DC/DC] → Program blocks → Add new block)



9. In the selection, select '**Function block (FB)**' and assign the name '**conveyor**'. As programming language, we specify function block diagram '**FBD**'. Enumeration is automatic. Since this FB1 is called later with the symbolic name anyhow, this number is no longer that important. Accept the input with '**OK**'. (→ Function block (FB1) → conveyor → FBD → OK)



10. The block '**conveyor[FB1]**' will be opened automatically. But before we can write the program, we have to declare the block's interface.

When the interface is declared, the local variables -known only in this block- are specified.

The variables are divided into two groups:

- Block parameters that generate the interface of the block for the call in the program.

Type	Name	Function	Available in
Input parameters	Input	Parameters whose values the block reads	Functions, function blocks and some types of organization blocks
Output parameters	Output	Parameters whose values the block writes	Functions and function blocks
In/out parameters	InOut	Parameters whose value the block reads when called, and after processing writes to the same parameter	Functions and function blocks

- Local data that is used for storing intermediate results

Type	Name	Function	Available in
Temporary local data	Temp	Variables that are used for storing temporary intermediate results. Temporary data is retained for one cycle only.	Functions, function blocks and organization blocks
Static local data	Static	Variables that are used for storing static intermediate results in instance data blocks. Static data is retained -also over several cycles-.until it is written anew.	Function blocks

11. To declare local variables, the following variables are needed for our example.

Input:

manual	Here, the signal for selecting the operating mode Manual is entered
automatic	Here, the signal for selecting the operating mode Automatic is entered
on	Here, the start signal is entered
off	Here, the stop signal is entered

Output:

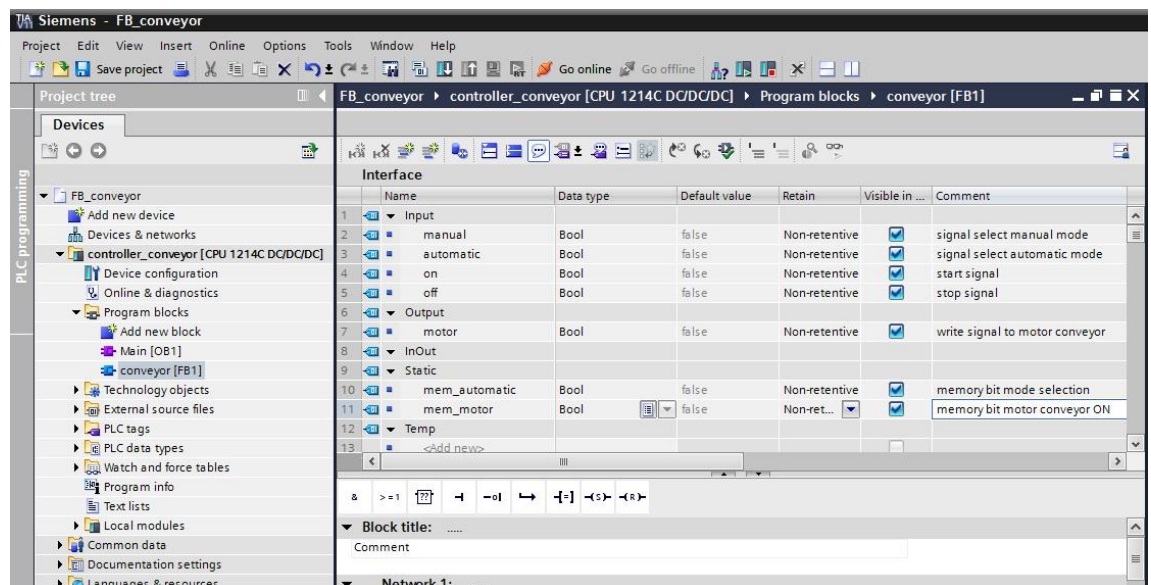
motor	Here, the output signal for the output conveyor motor is written
-------	--

Static (exists only in the function blocks FB):

memory_automatic	Here, the preselected operating mode is stored
memory_motor	Here, we store when the motor was started in the Automatic mode

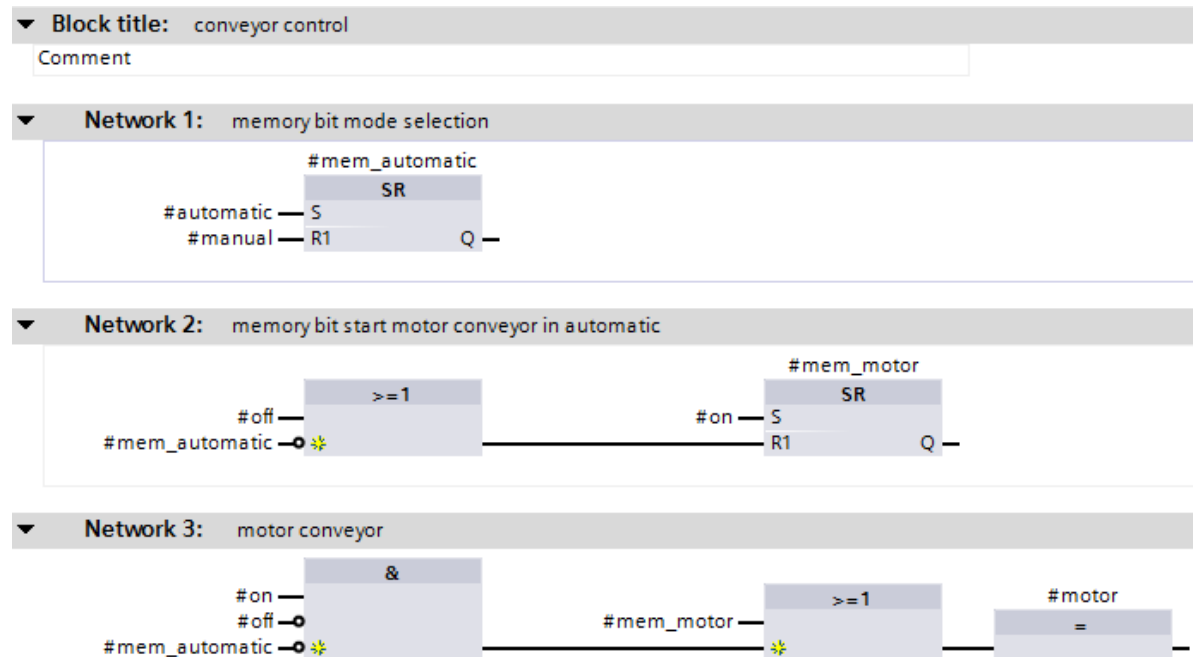
All variables are of the type 'Bool'; that means binary variables that only can have the status '0' (false) or '1' (true).

In this example, it is important to note that the status of the two variables 'memory_automatic' and 'memory_motor' has to be stored over a longer period of time. For that reason, the variable type '**Static**' has to be used here. This variable type in turn exists only in a function block FB. For the sake of clarity, all local variables should also be provided with a sufficient comment.

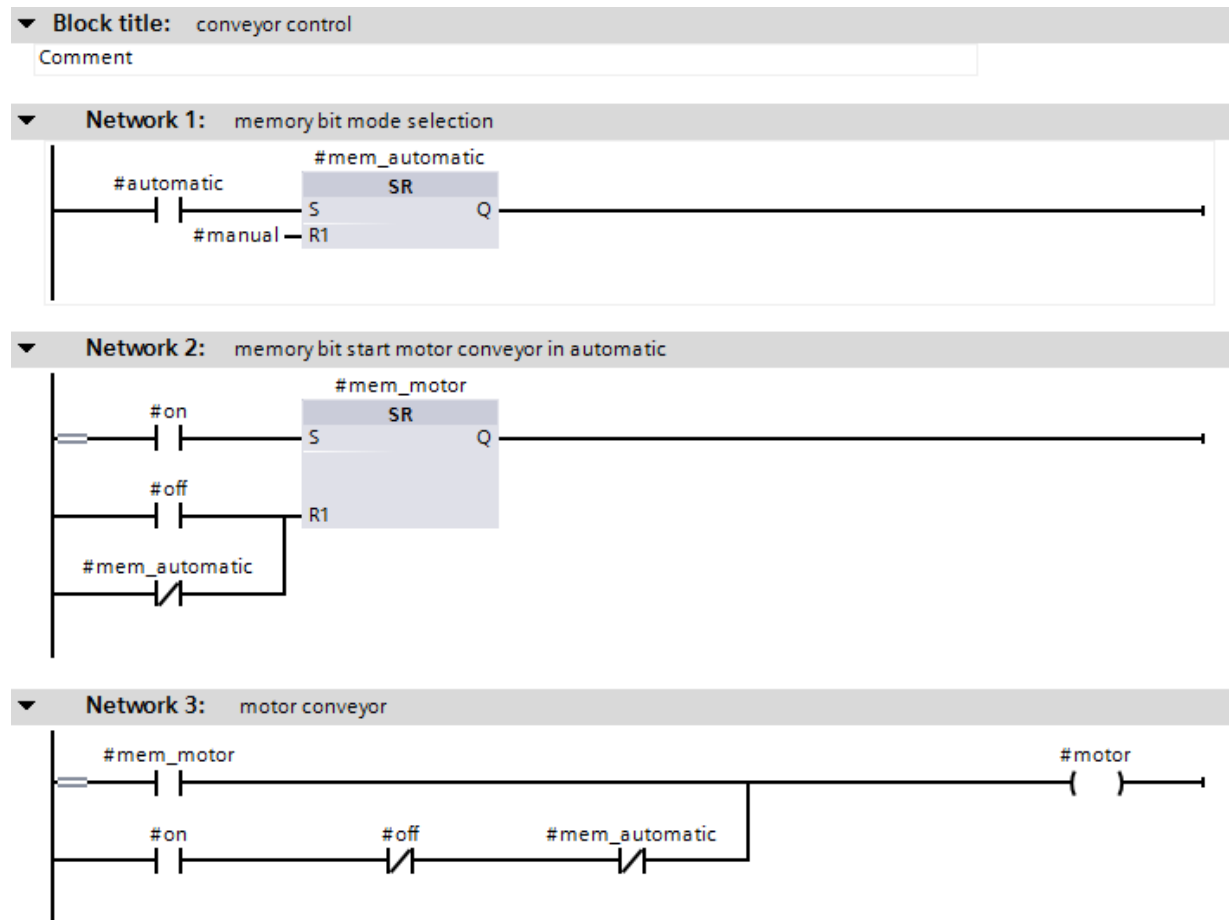


12. After the local variables have been declared, the program can now be entered by using the variable names (variables are identified with the symbol '#'). For the example in FBD, it could look like this:

Program in function block diagram (FBD):

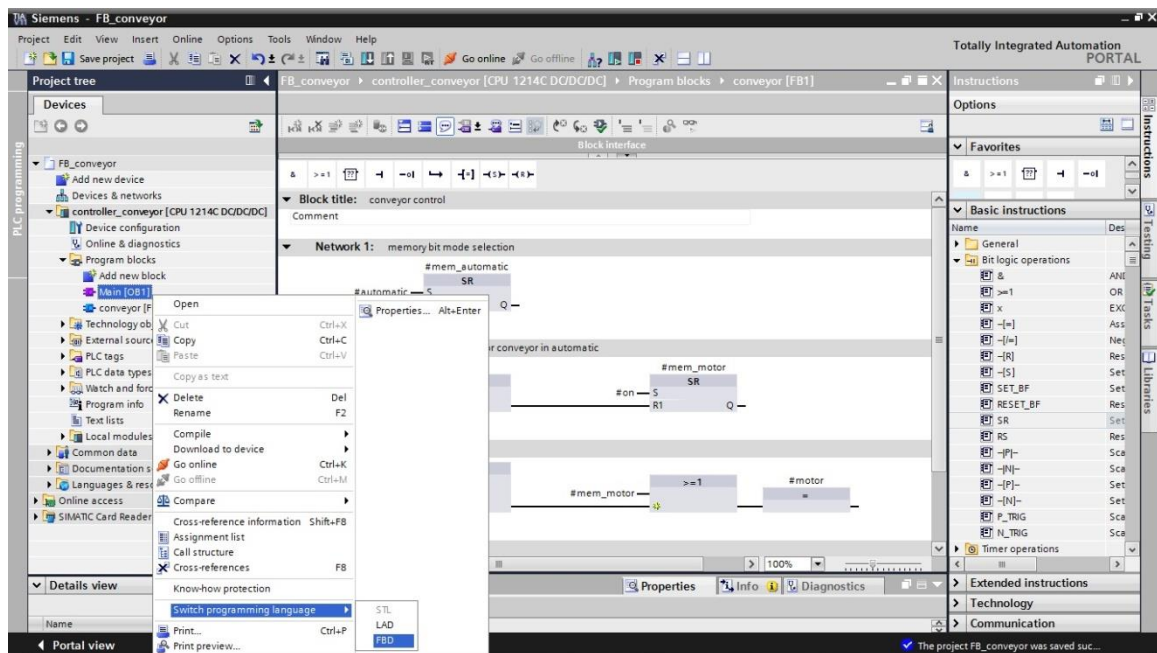


Program in ladder diagram (LAD):



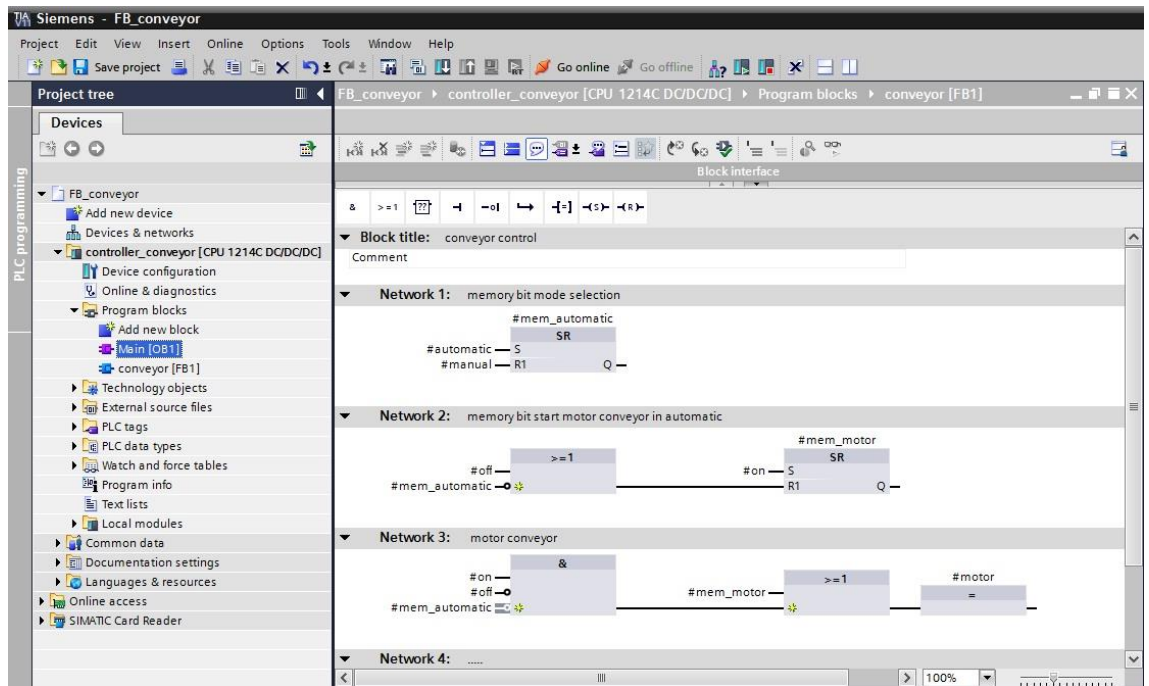
13. Next, right-click on the block 'Main[OB1]'.

Then, under '**Switch programming language**', select the function block diagram 'FBD'.

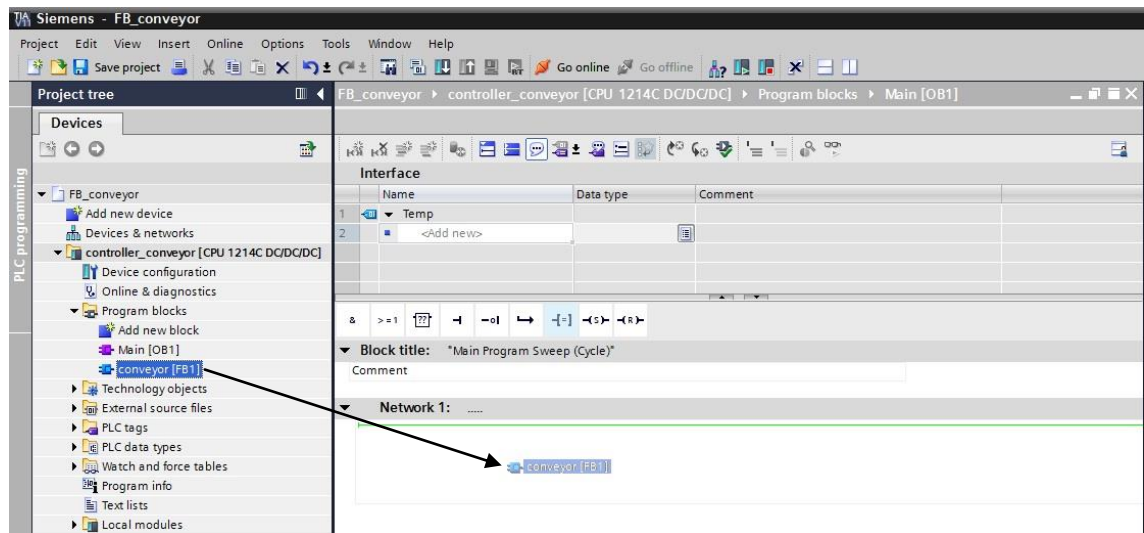


14. Now, the block “conveyor” has to be called from the program block Main[OB1]. Otherwise, the block would not be processed.

A double click on ‘Main[OB1]’ opens this block. (→ Main[OB1])

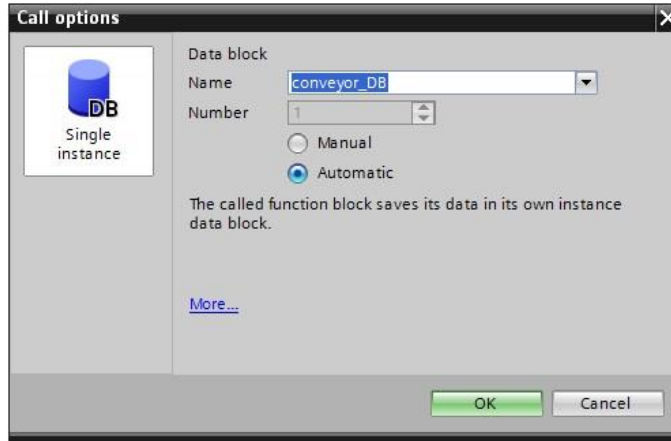


15. Now, you can drag the block **"conveyor[FB1]"** with Drag&Drop to Network 1 of the block Main[OB1]. (→ conveyor[FB1])

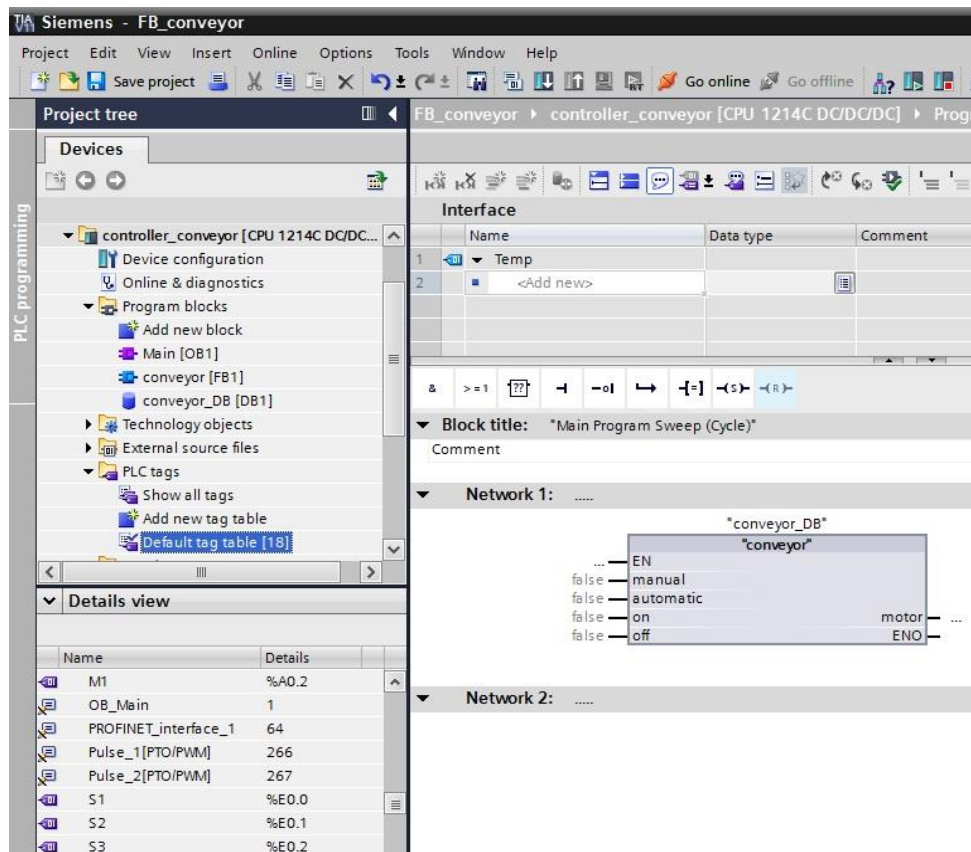


16. Since we are dealing with a function block, it has to be provided with memory. In SIMATIC S7-1200, data blocks are provided as memory. Such an assigned data block is called **Instance Data block**.

Here, it is to be specified and generated '**automatically**'. (→ Automatic→ OK)





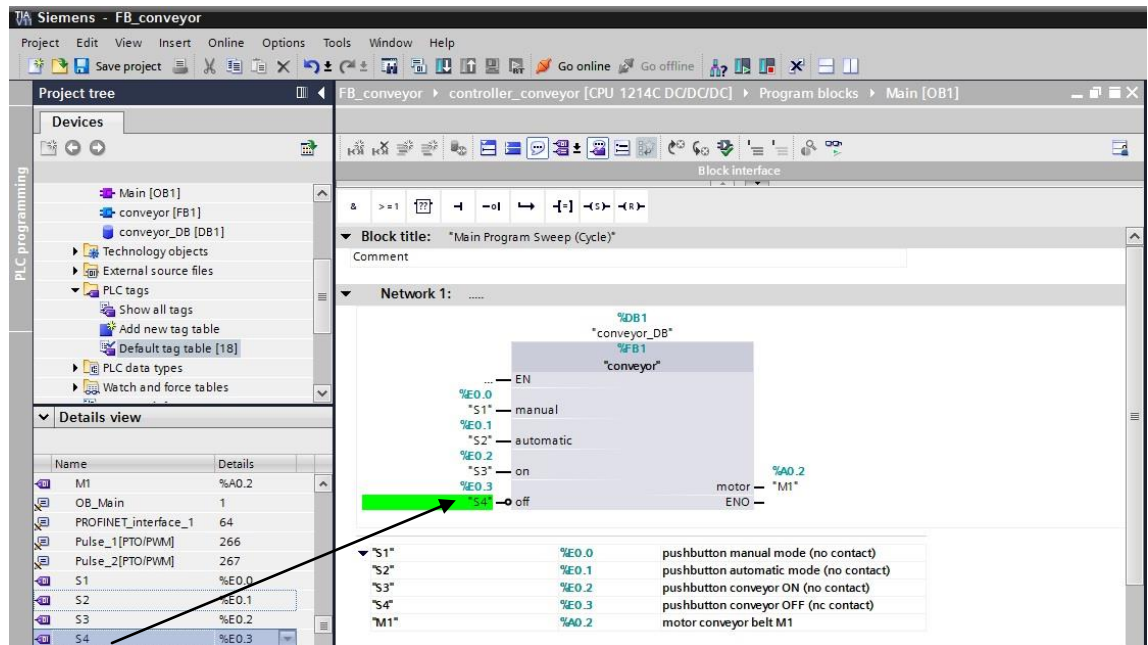
Highlight the default tag table.



17. In OB1, we now connect the input variables and the output variable with the PLC tags shown here.



To this end, just drag the PLC tags to the block variables.

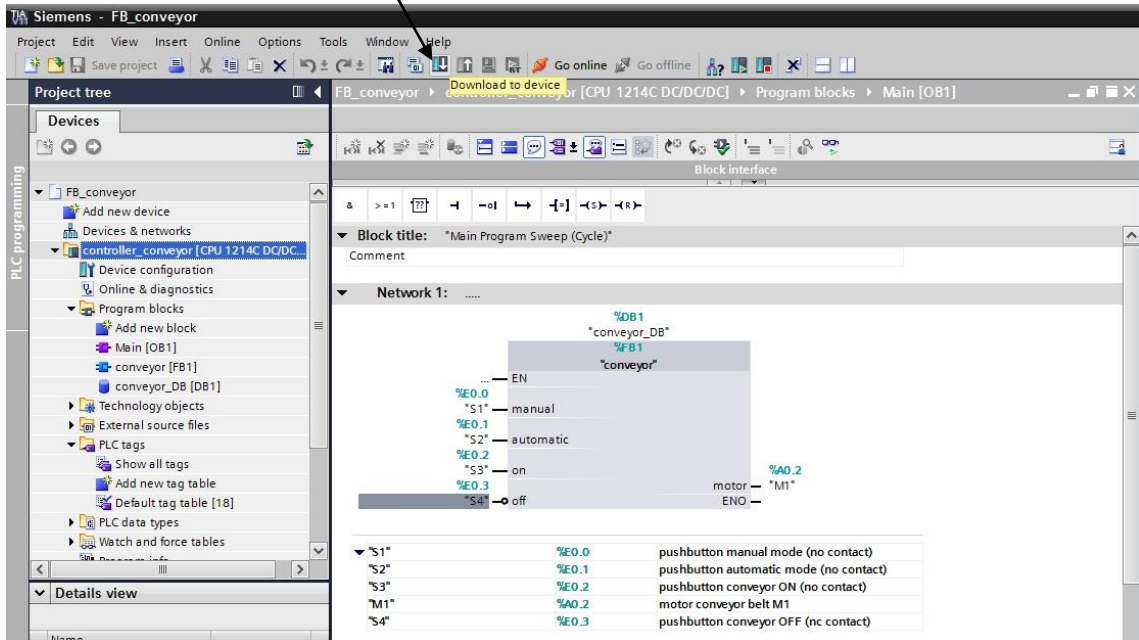
By clicking on , the project is saved. (→ "S1" → "S2" → "S3" → "S4" → "M1" → )



Important!

The Off button S4 is a break contact (NC) and has to be negated at the block during wiring; i.e., the Off function in the block is pending when the Off button S4 is operated and thus no signal is pending at terminal %I0.3.

18. To load your entire program to the CPU, first highlight the folder '**controller conveyor**' and then click on the symbol  Load to device. (→ controller_conveyor → )

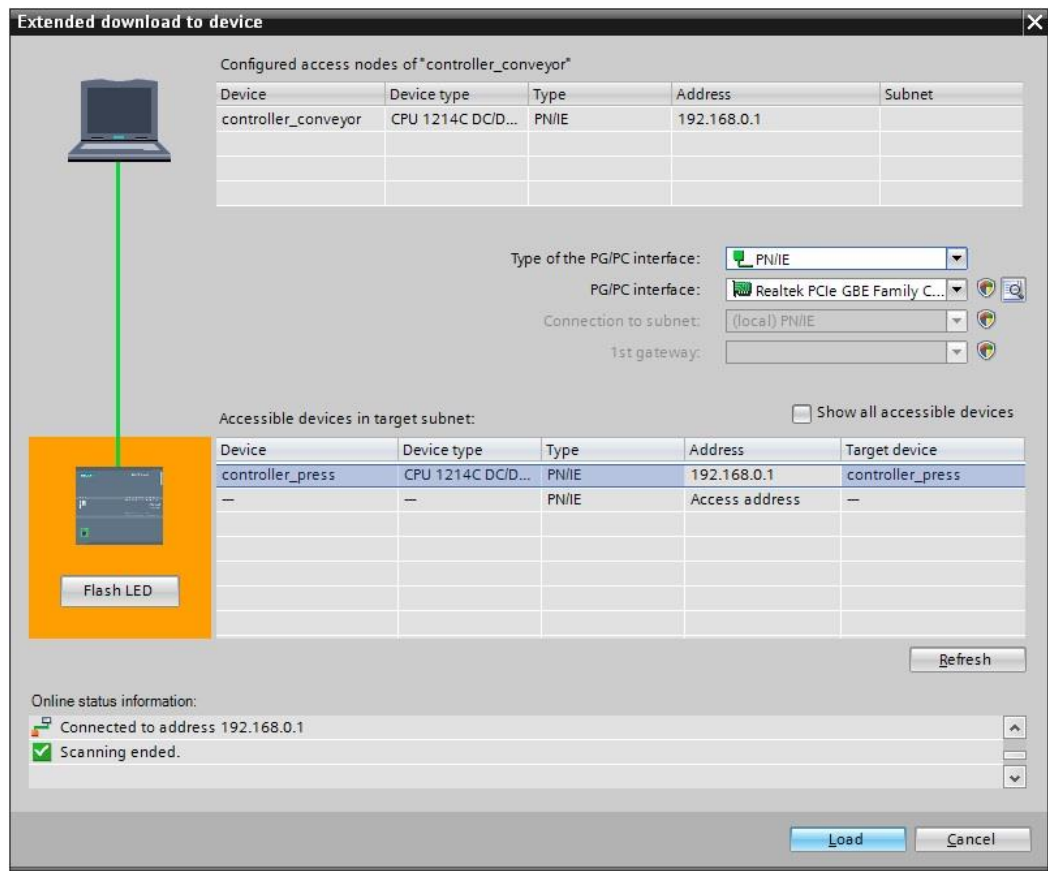


The screenshot shows the Siemens TIA Portal interface. The 'Project tree' on the left has 'controller_conveyor [CPU 1214C DC/DC...]' selected. The main workspace displays the 'Main Program Sweep (Cycle)' network diagram. The diagram shows a network with inputs %E0.0 (manual), %E0.1 (automatic), %E0.2 (on), %E0.3 (off), and %E0.4 (off). The output is %AO.2 (motor conveyor belt M1). A table below the diagram provides a legend for the variables.

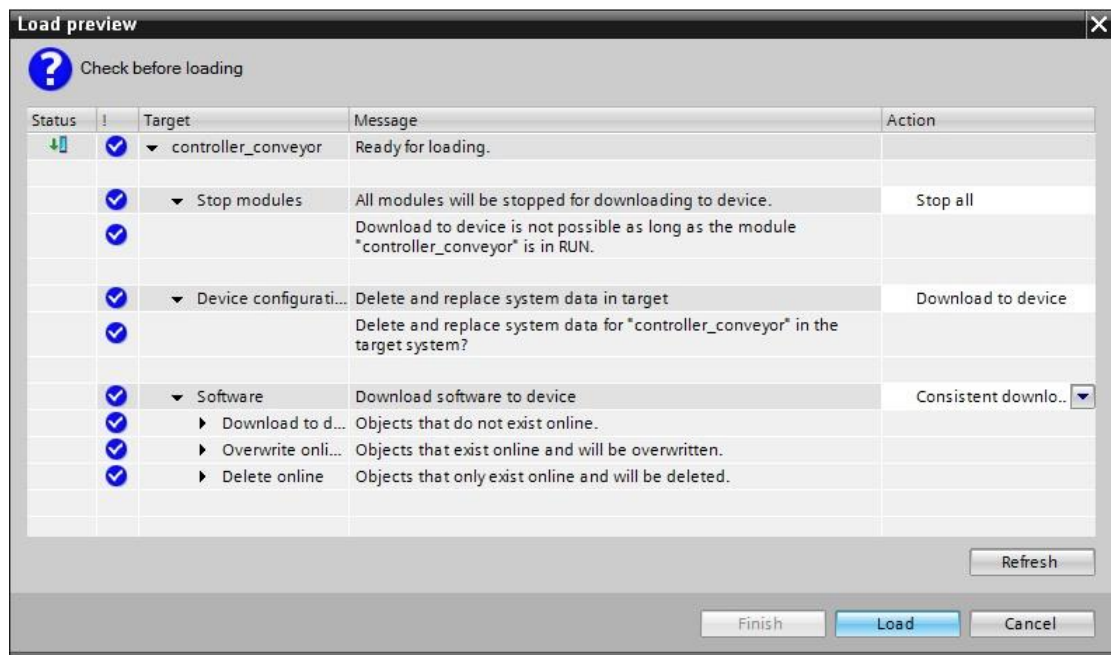
Variable	Address	Description
%S1	%E0.0	pushbutton manual mode (no contact)
%S2	%E0.1	pushbutton automatic mode (no contact)
%S3	%E0.2	pushbutton conveyor ON (no contact)
%M1	%AO.2	motor conveyor belt M1
%S4	%E0.3	pushbutton conveyor OFF (no contact)

19. If you omitted specifying the PG/PC interface beforehand, a window is displayed where you can still do this.

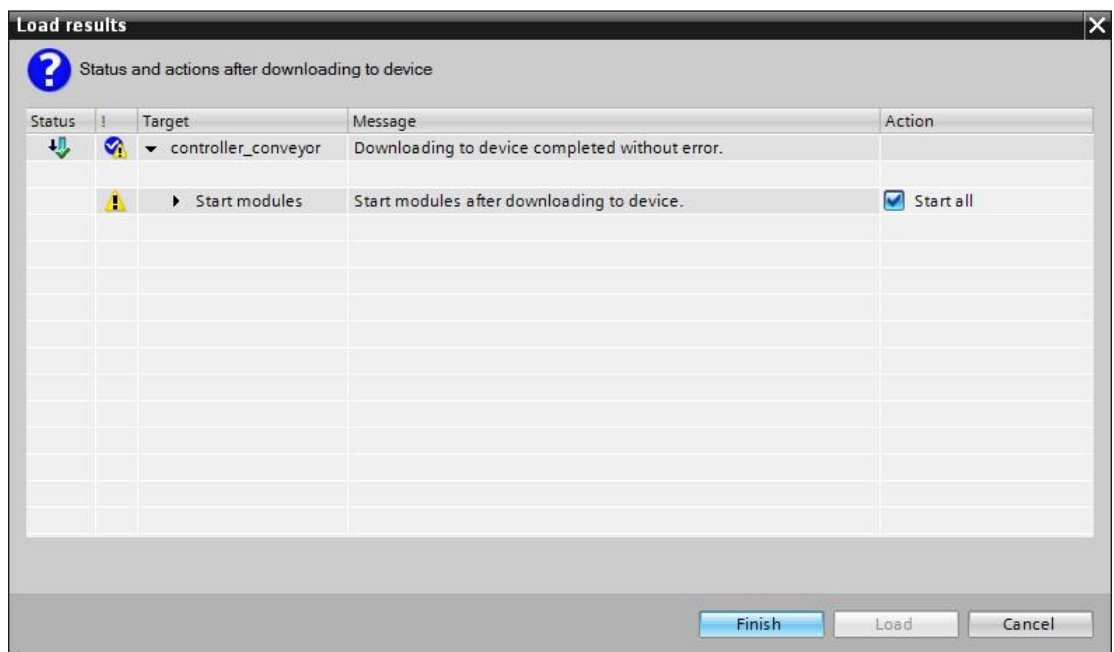
(→ PG/PC interface for loading → Load)




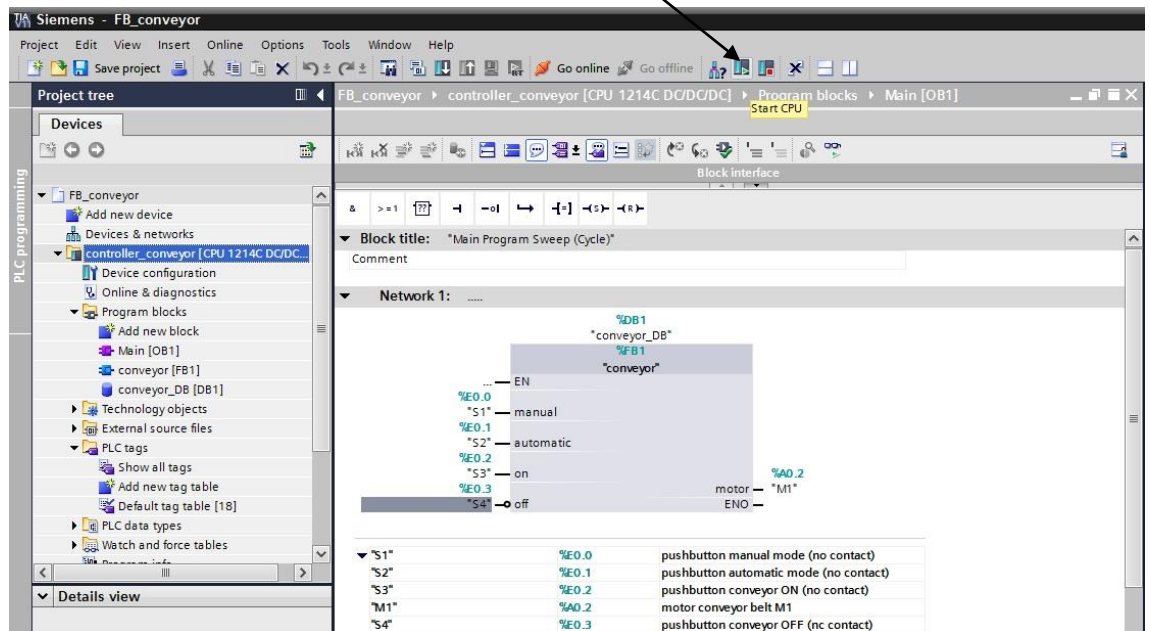
20. Click on 'Load' once more. In a window, the status is indicated during loading. (→ Load)



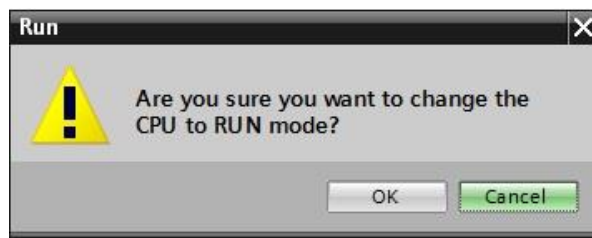
21. If loading was successful, it is shown in a window. Click on '**Finish**'. (→ Finish)





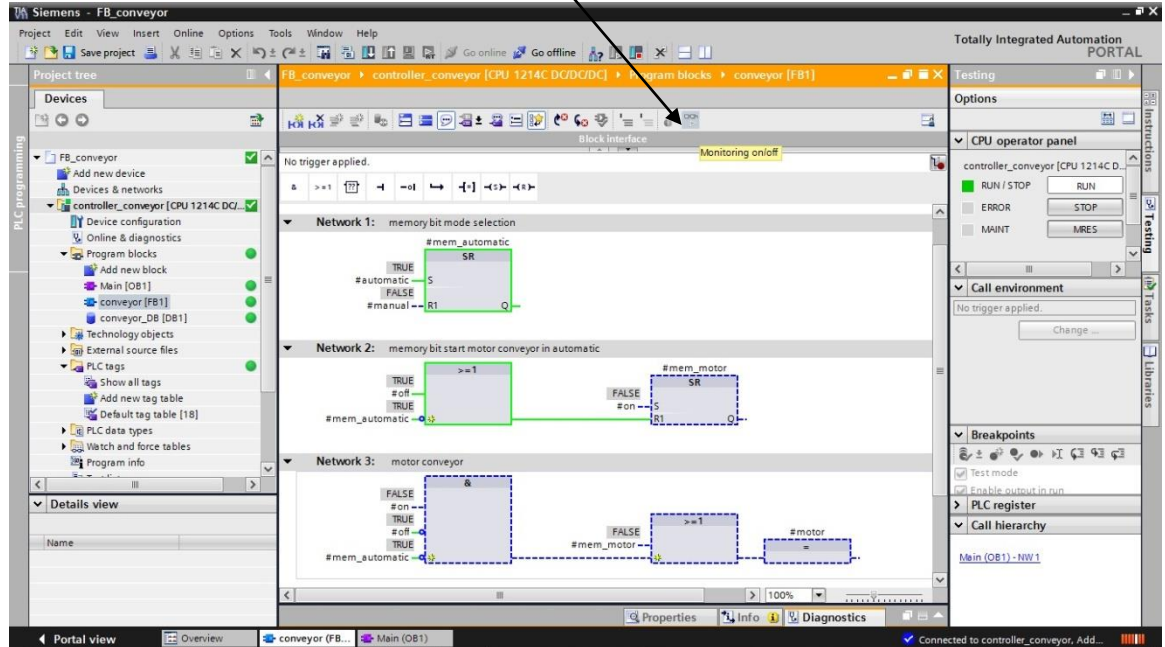
22. Now, start the CPU by clicking on the symbol .



23. With 'OK', confirm the question whether you actually want to start the CPU. (→ OK)



24. By clicking on the symbol  Monitoring On/Off, you can, during the program test, observe the status of the input and output variables at the block "conveyor", but also the program execution in the block "conveyor". (→ conveyor[FB1] → )



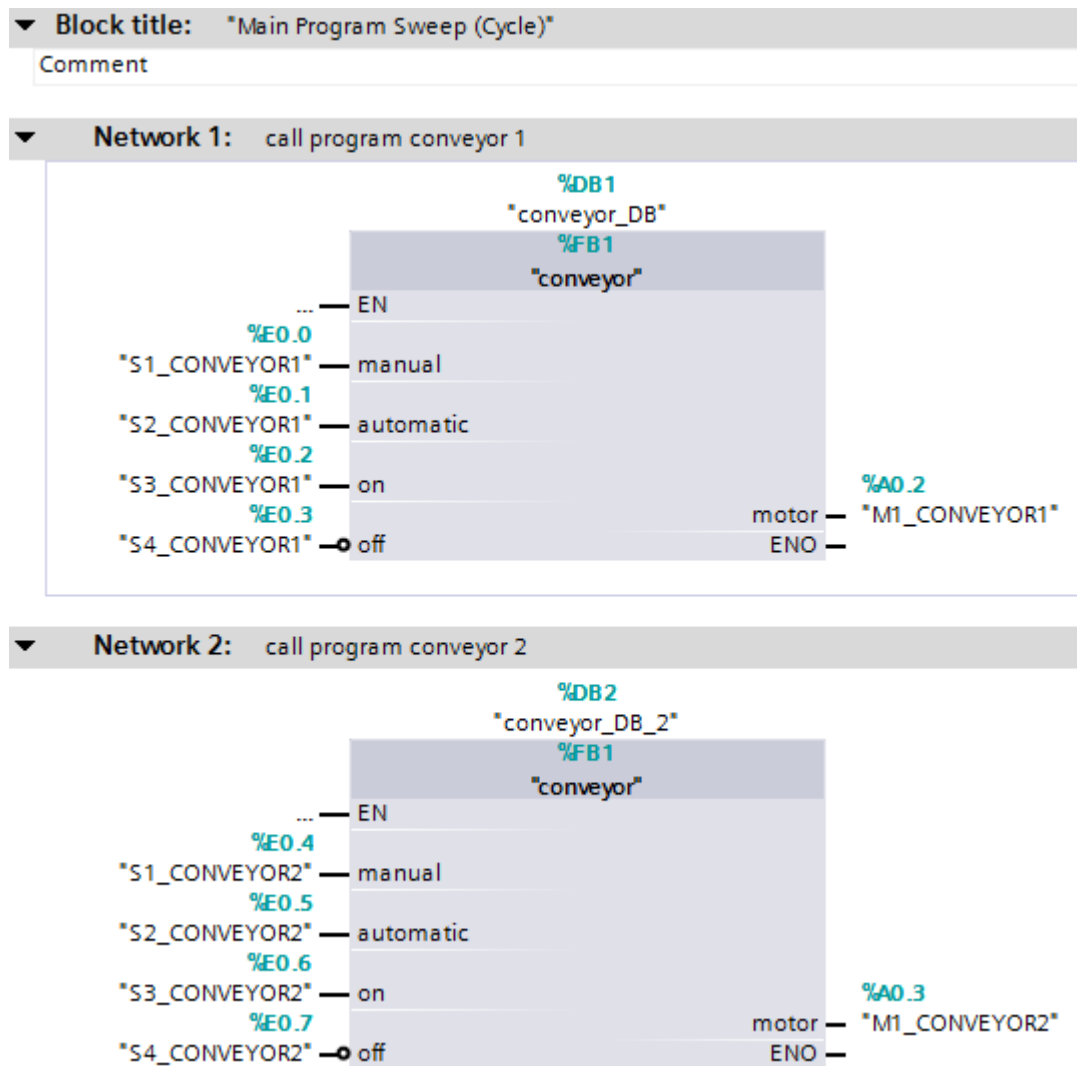
25. Since our block “conveyor” was generated according to the rules for standard blocks (no use of global variables within the block!!!!), it can now be used and called any number of times.

Below, an expanded PLC tag table is shown, with the inputs and outputs for two conveyors.

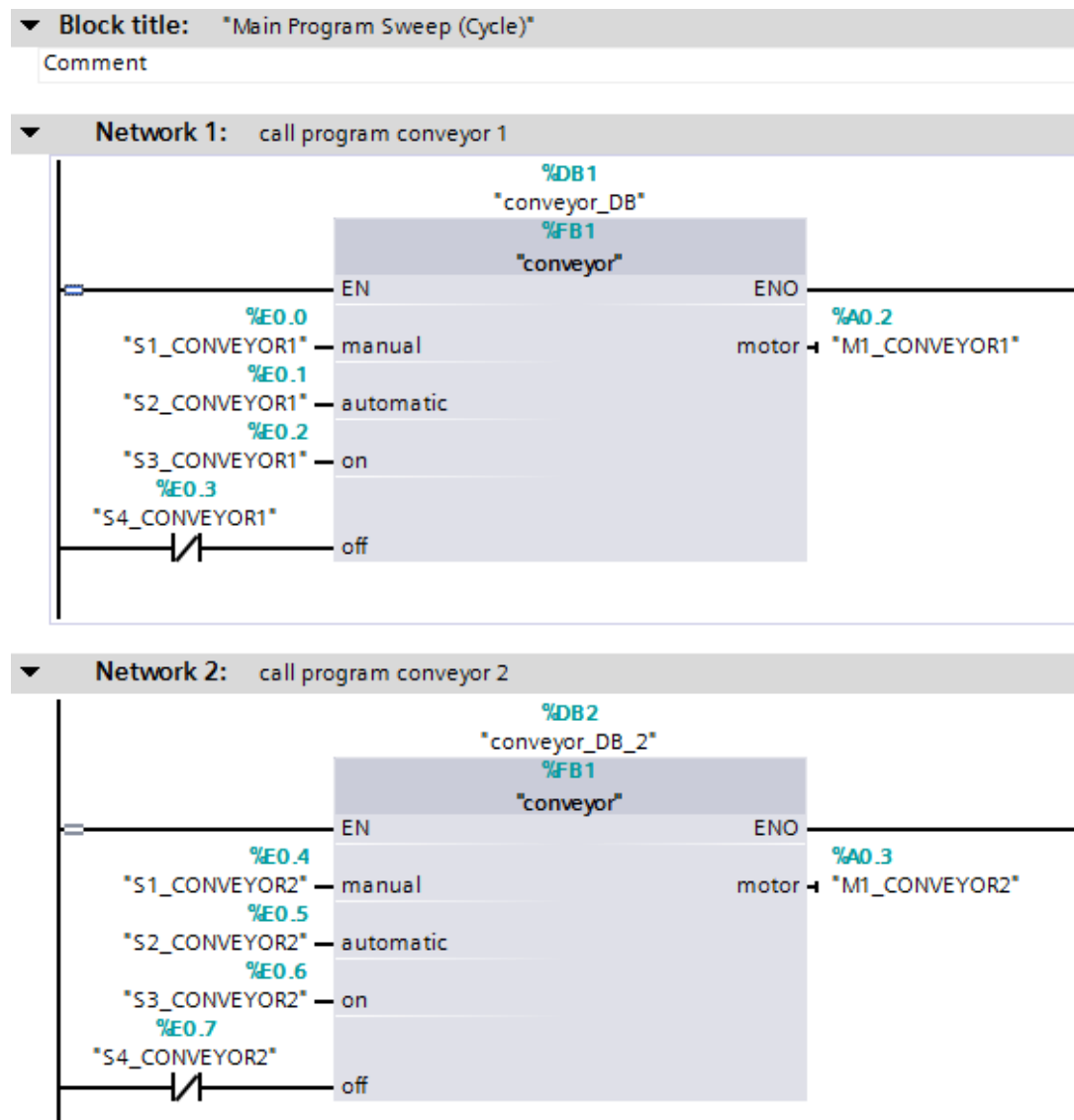
Default tag table							
	Name	Data type	Address	Retain	Visible..	Acces...	Comment
1	S1_CONVEYOR1	Bool	%E0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor1 pushbutton manual mode (no contact)
2	S2_CONVEYOR1	Bool	%E0.1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor1 pushbutton automatic mode (no contact)
3	S3_CONVEYOR1	Bool	%E0.2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor1 pushbutton conveyor ON (no contact)
4	S4_CONVEYOR1	Bool	%E0.3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor1 pushbutton conveyor OFF (nc contact)
5	M1_CONVEYOR1	Bool	%A0.2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor1 motor conveyor belt M1
6	S1_CONVEYOR2	Bool	%E0.4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor2 pushbutton manual mode (no contact)
7	S2_CONVEYOR2	Bool	%E0.5	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor2 pushbutton automatic mode (no contact)
8	S3_CONVEYOR2	Bool	%E0.6	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor2 pushbutton conveyor ON (no contact)
9	S4_CONVEYOR2	Bool	%E0.7	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor2 pushbutton conveyor OFF (nc contact)
10	M1_CONVEYOR2	Bool	%A0.3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	conveyor2 motor conveyor belt M1
11	<Add new>			<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

26. Now, the block **"conveyor"** can also be called twice in OB1, with different wiring respectively. For each call, another instance data block is specified.

Program in function block diagram (FBD):



Program in ladder diagram (LAD):



Two conveyors separated from each other can now be controlled by means of the same conveyor block. Only another instance data block has to be assigned for each call.

