SIEMENS

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

Siemens Automation Cooperates with Education

TIA Portal Module 010-030

IEC Timers and IEC Counters at 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

Continued Training

For regional Siemens SCE continued training, please contact your regional SCE contact person
siemens.com/sce/contact

Additional information regarding SCE

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.

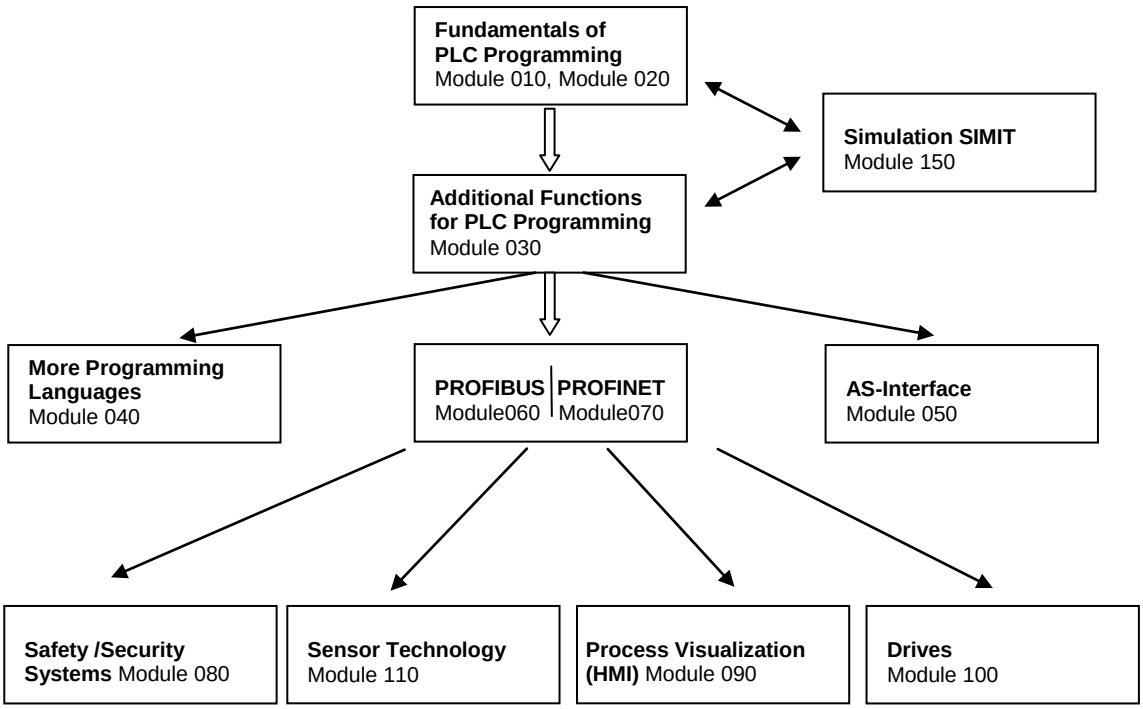
We wish to thank the Michael Dziallas Engineering Corporation and all other involved persons for their support during the preparation of this training curriculum.

PAGE

Contents

1. Preface

Regarding its content, module SCE_EN_010-030 is part of the training unit **'Basics of PLC Programming'** and represents a fast **entry point** for programming the SIMATIC S7 1200 with TIA Portal.



Training Objective:

In this module 010-030, the reader learns how to program the programmable logic controller (PLC) SIMATIC S7-1200 using the programming tool TIA Portal. Module 010-030 provides the fundamentals and shows in the steps below how it is done based on a detailed example. .

- Installing the software and setting the program interface
- Explanation of what a PLC is and how it processes
- Configuration and operation of the PLC SIMATIC S7-1200
- Creating, loading and testing a sample program

Prerequisites:

To successfully work through module 010-030, the following knowledge is assumed:

- How to operate Windows
- Basics of PLC programming with the TIA Portal (for example, Module 010-010 'Startup' Programming the SIMATIC S7-1200 with TIA Portal V11)
- Blocks for the SIMATIC S7-1200 (for example, Module 010-020 Block Types at the SIMATIC S7-1200)

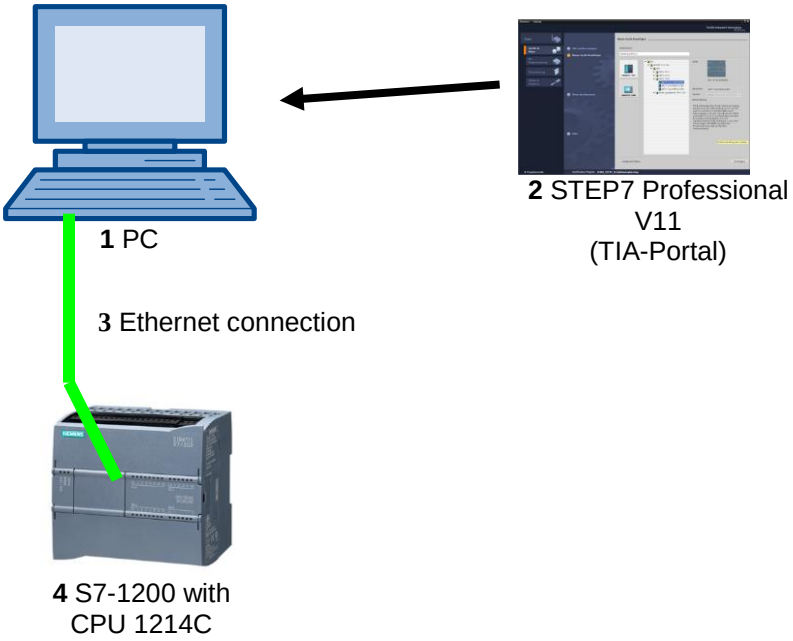
Hardware and software required

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 SP1

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. Instances and Multi-Instances when Programming the SIMATIC S7-1200

Calling a function block is referred to as **instance**. To each call of a function block, an **instance data block** is assigned that is used for data storage. The actual parameters and the static data are stored here.

The variables declared in the function block determine the structure of the instance data block.

Applying single and multi-instances

Instance data blocks can be assigned as follows:

- Call as **single instance**:

  - A separate instance data block for each instance of a function block

- Call as **multi-instance**:

  - One instance data block for several instances of one or several function blocks

3.1 Instance Data Blocks/Single Instances

The call of a function block to which its own instance data block is assigned is referred to as **single instance**.

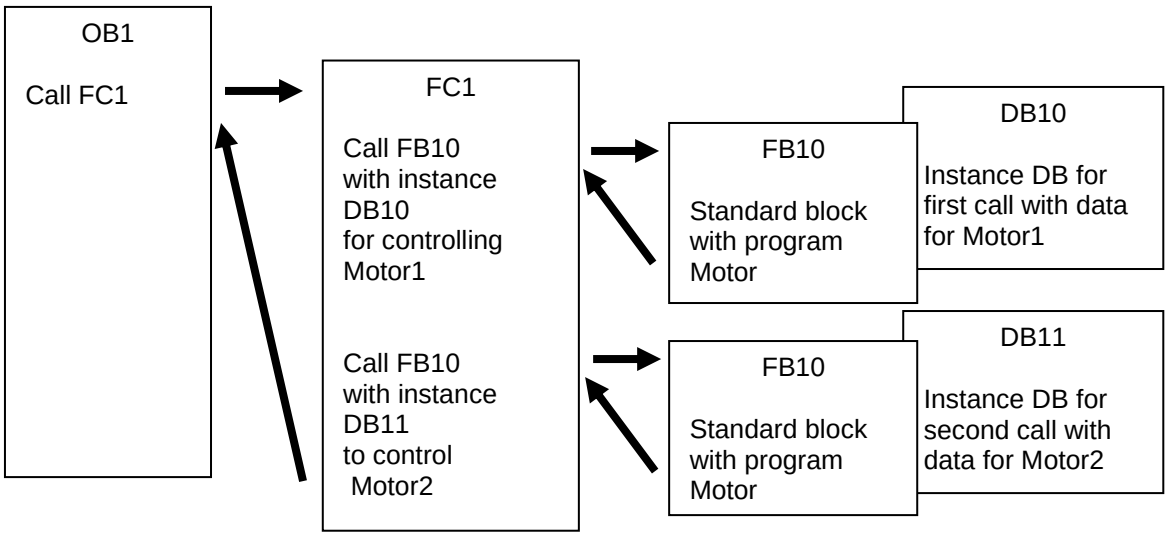
If the function block was generated according to the rules for standard blocks (refer to Module 010-020), it can be called multiple times.

However, for each call as single instance, you have to assign a different instance data block.

Example of single instances:

The figure below shows two motors being controlled with a function block FB10 and two different data blocks:

The different data for the individual motors -for example, speed, power-up time, total operating time- is stored in the different instance data blocks DB10 and DB11.

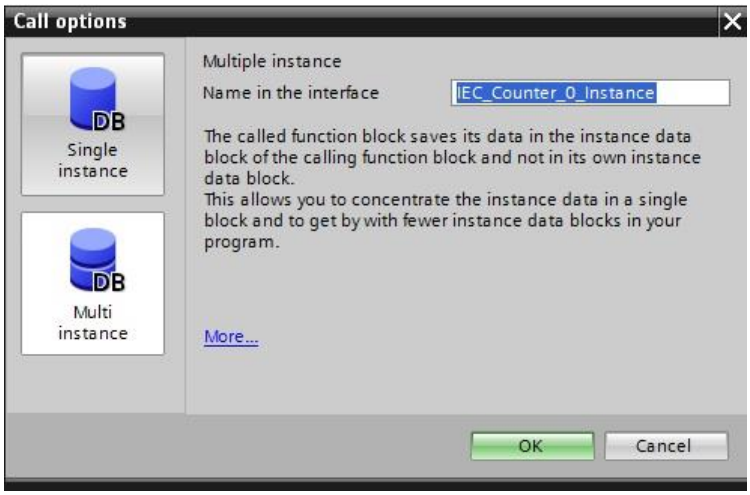


Note
Some instructions such as timers and counters behave like function blocks. If they are called, they also represent instances and need an assigned memory area; in the form of an instance data block, for example.

3.2 Multi-Instances

Because of the memory capacity of the CPUs used, it is possible that you want to or you can allocate only a limited number of data blocks for instance data.

If in your user program, additional already existing function blocks, timers, counters, etc. are called in a function block, it is possible to call these additional function blocks without their own (that is, additional) instance DBs.
Simply select the call options **'Multi-Instance'**:



Notes:

For a function block that was called, multi-instances make it possible to place its data in the instance data block of the function block that is calling.

The block that is calling always has to be a function block in this case.

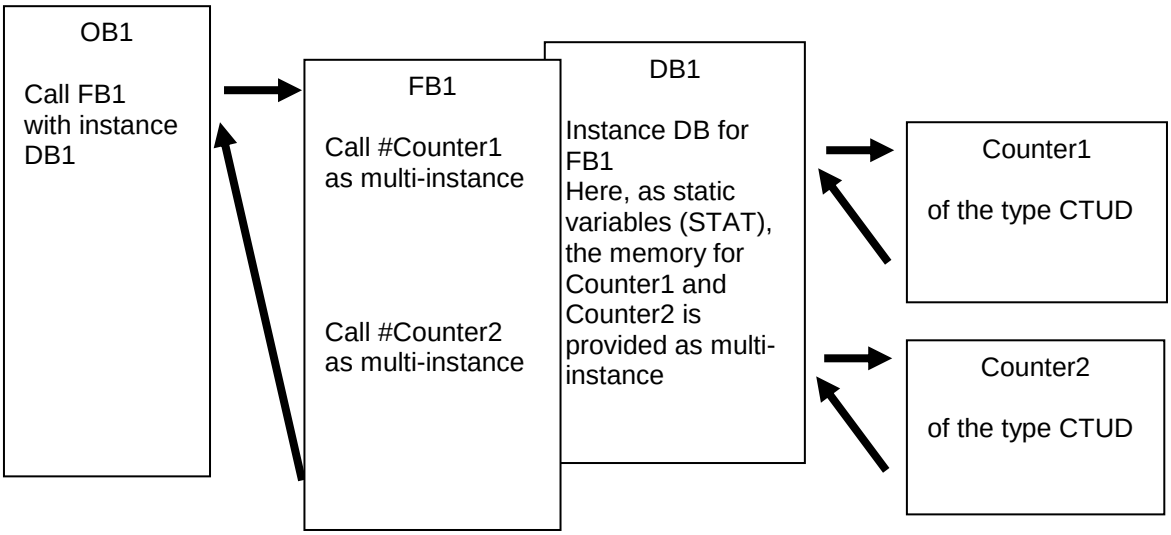
In this way, you concentrate the instance data in one instance data block; i.e., you can utilize the available number of DBs more efficiently.

This, by the way, always has to be done if the block that is calling is to be reusable as a standard block.

Example for Multi-Instances:

The figure below shows a counter of the type CTUD (up and down counter) being called twice..

The different data for the two counters is stored as different **multi-instances** in instance data block DB1 of the calling function block FB1.



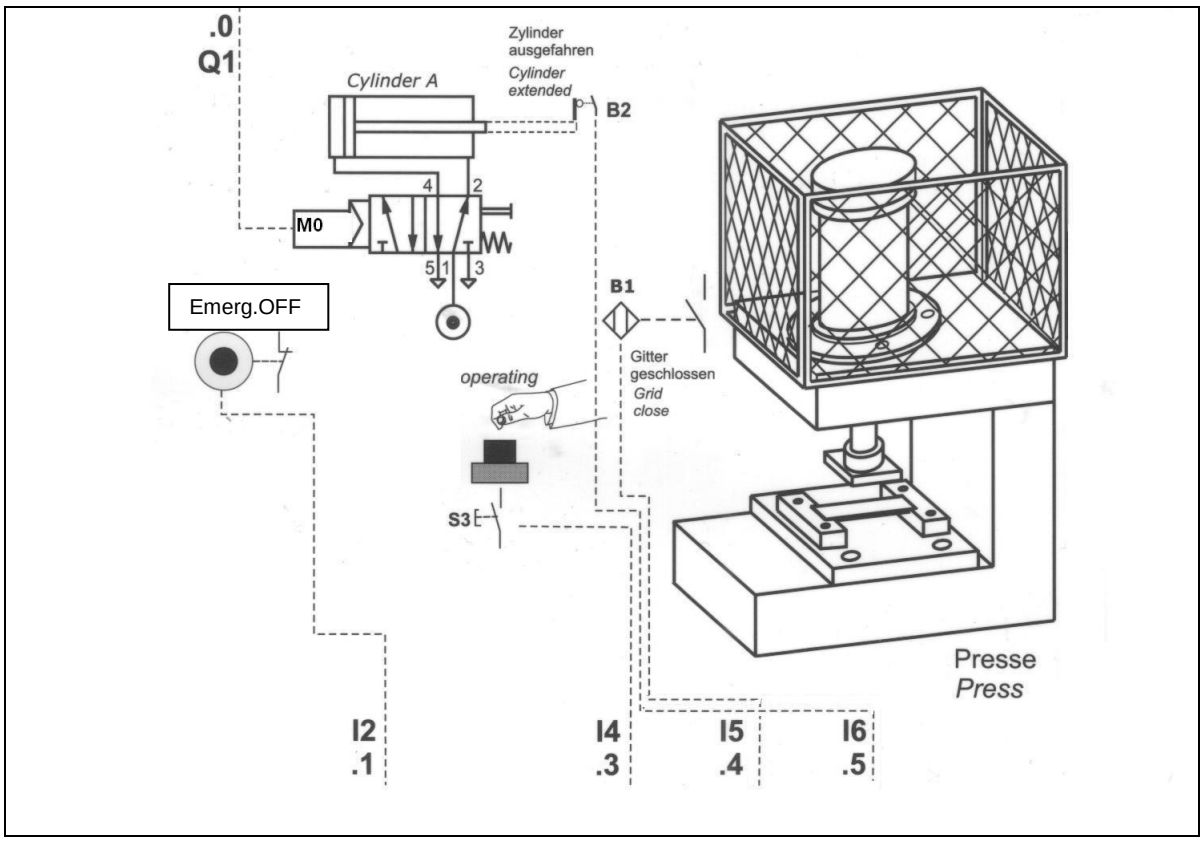
4. Sample Task: Press Control with Timer and Instance DB

For our program, a timer will be added to the press control in Module 010-010.

The task to be performed is as follows:
A press with a safety fence is to be started with a START button S3 only if the safety fence is closed. This state is monitored with a sensor Safety fence closed B1.
If this is the case, a 5/2 way valve M0 for the press cylinder is activated so that a plastic shape can be pressed.
The press is to retract again when the EMERGENCY OFF button (NC) is operated, or the sensor Safety Fence B1 no longer responds.
If the sensor Cylinder extended B2 responds, the press is to retract again after a press time of 5 seconds.
An instance DB is used as the memory for the timer.

Assignment list:

Address	Symbol	Comment
%I 0.1	EMERGENCY OFF	EMERGENCY OFF button NC
%I 0.3	S3	Start button S3 NO
%I 0.4	B1	Sensor Safety fence closed NO
%I 0.5	B2	Sensor Cylinder extended NO
%Q 0.0	M0	Extend Cylinder A



5. Programming the Press with a Time Delay using the SIMATIC S7-1200

The software **'Totally Integrated Automation Portal'** manages the project and does the programming.

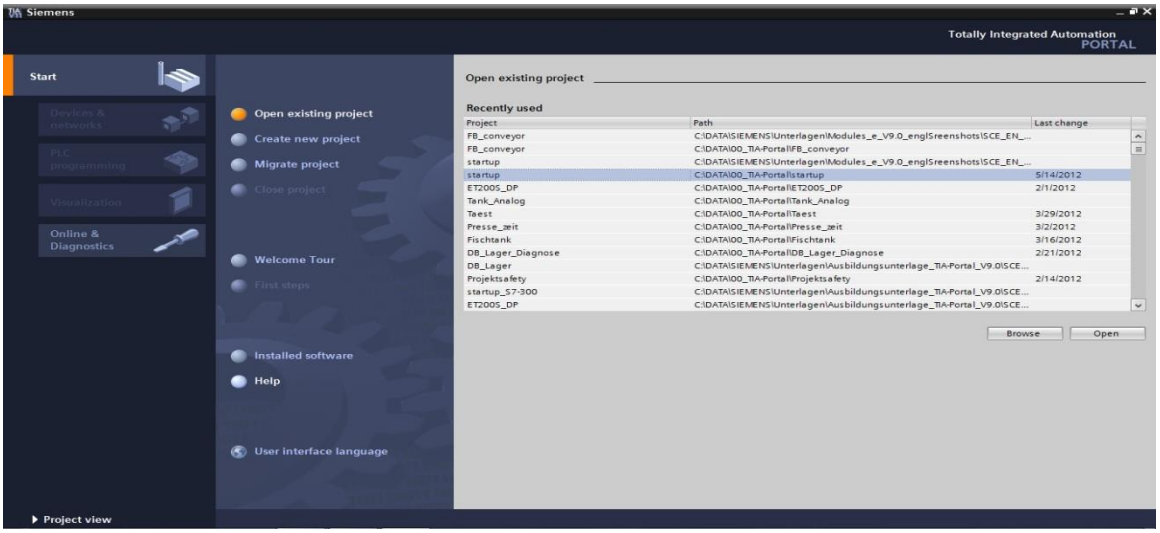
Here, under a uniform interface, the components such as the controller, visualization and networking the automation solution are set up, parameterized and programmed.
Online tools are provided for error diagnosis.

.

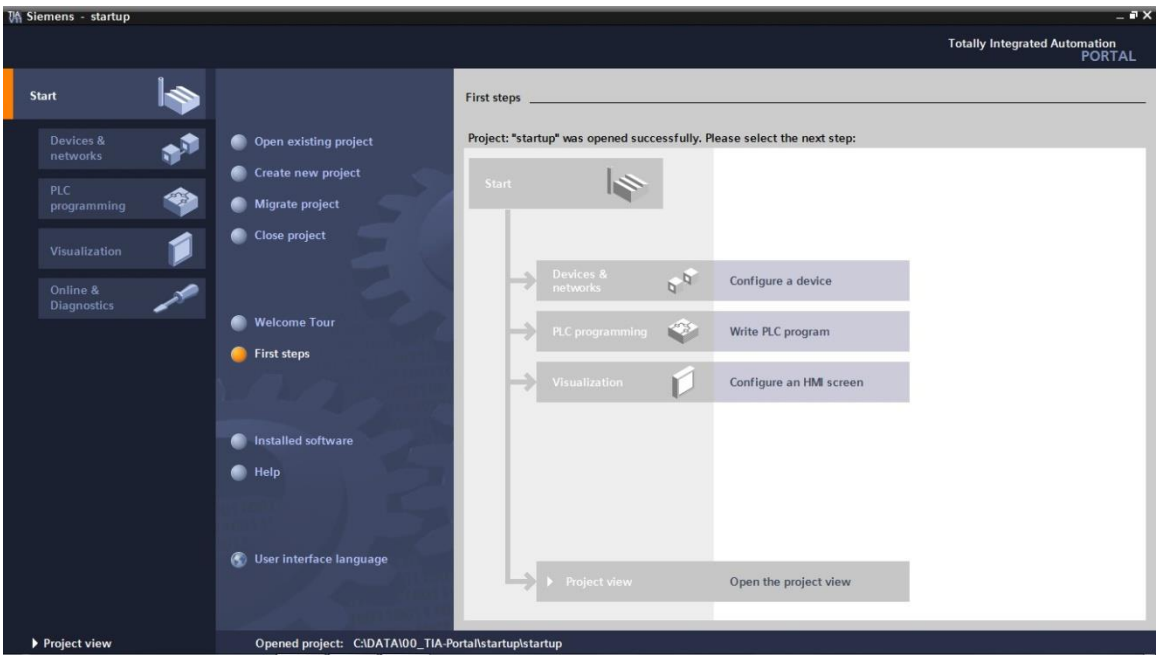
In the steps that follow, a project can be opened for the SIMATIC S7-1200, it can be stored under a different name and adapted to the new requirement.

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

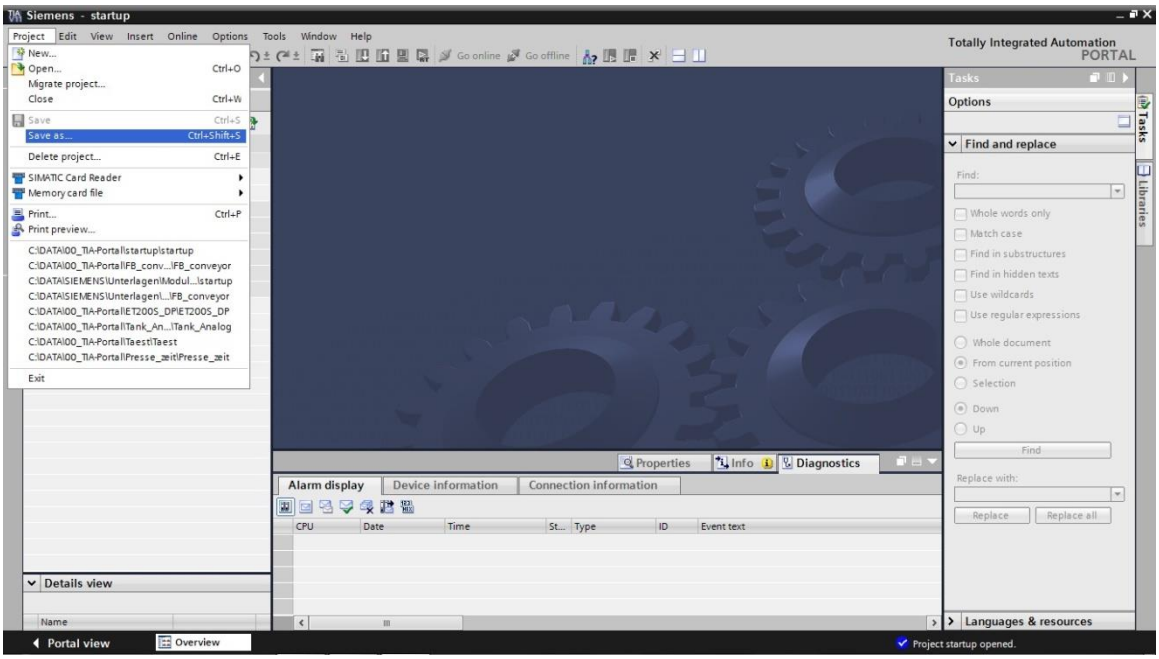
2. The project **"startup“** from Module 010-010 is now opened in the portal view as the basis for this program. (→ Open existing project → startup → open)



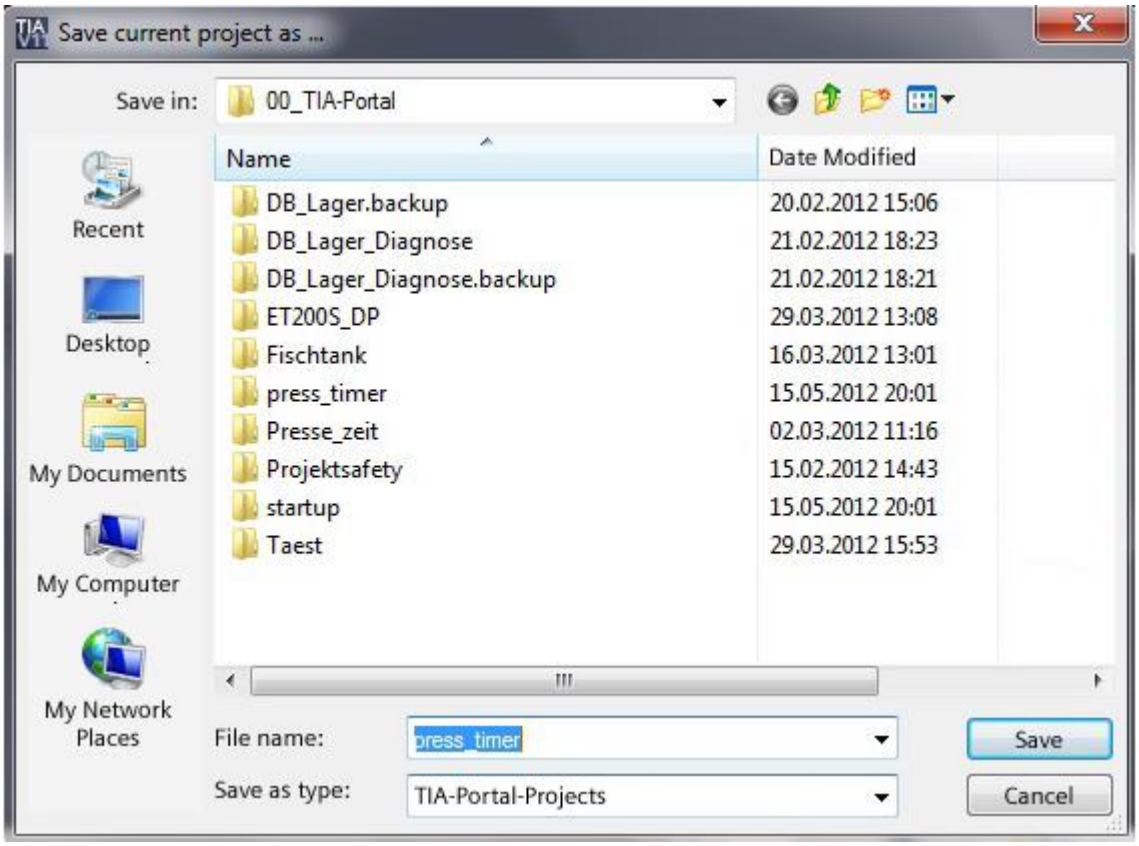
3. Next, **'First Steps'** for the configuration are suggested. We want to **'Open project view'.** (→ Open project view)



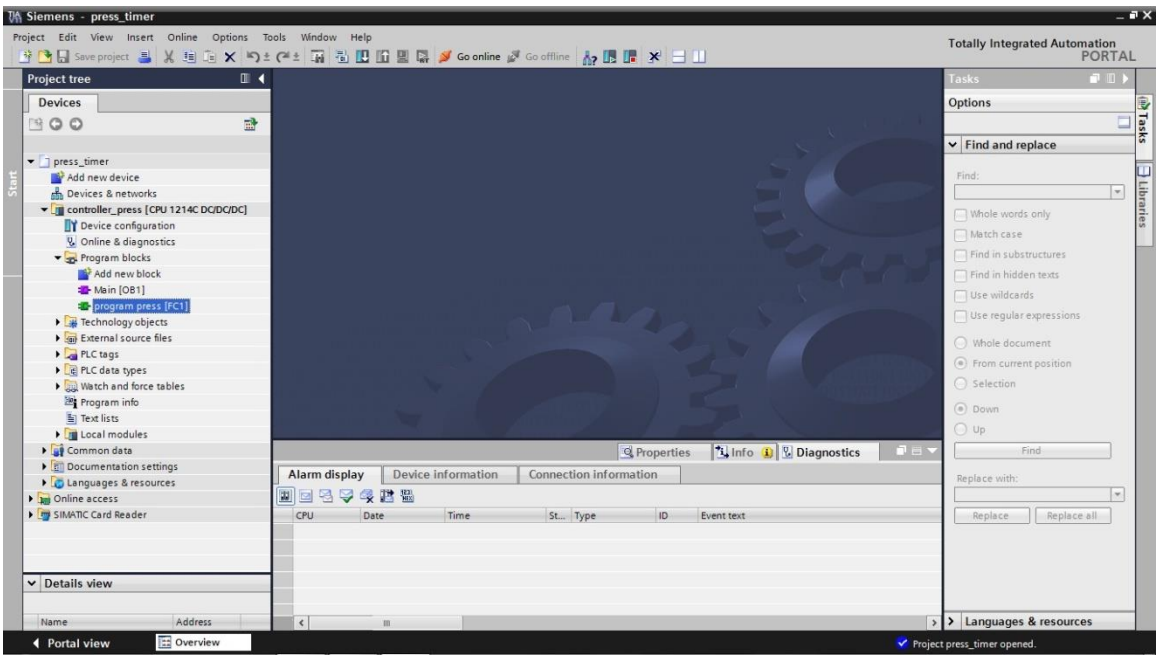
4. First, we want to save the project under another name. (→ Project → Save as)



5. Now, **'Save'** the project under the new name **'press_timer'.** (→ press_timer → Save)

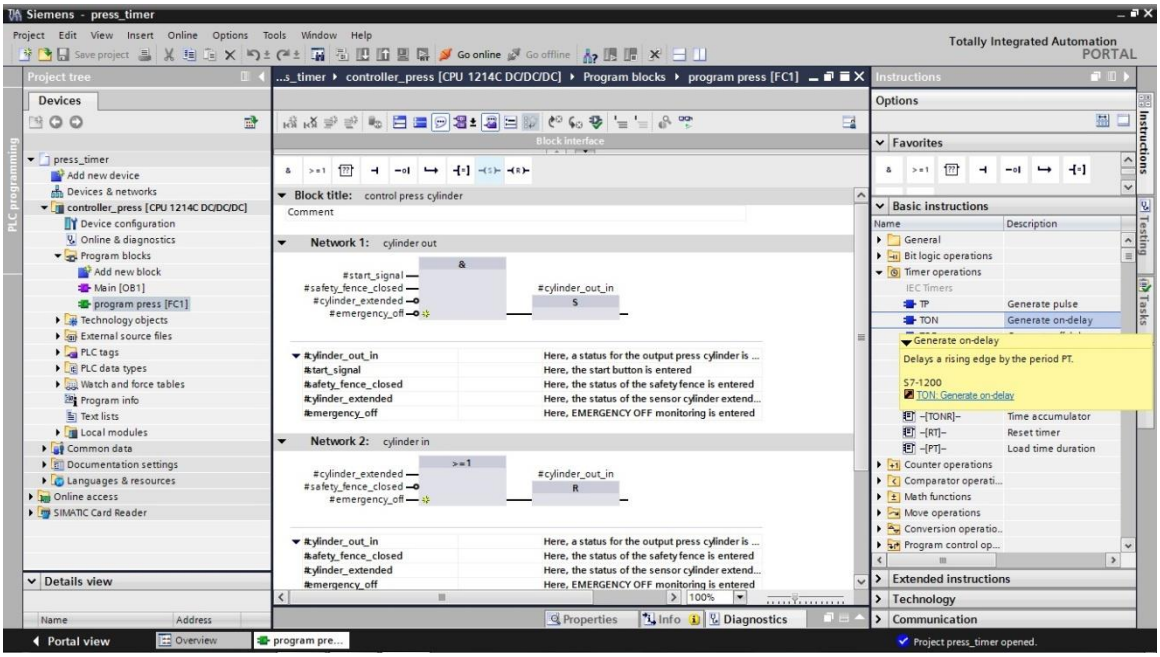


6. To make the changes, open the block **'program press[FC1]'** with a double click. (→ program press[FC1])

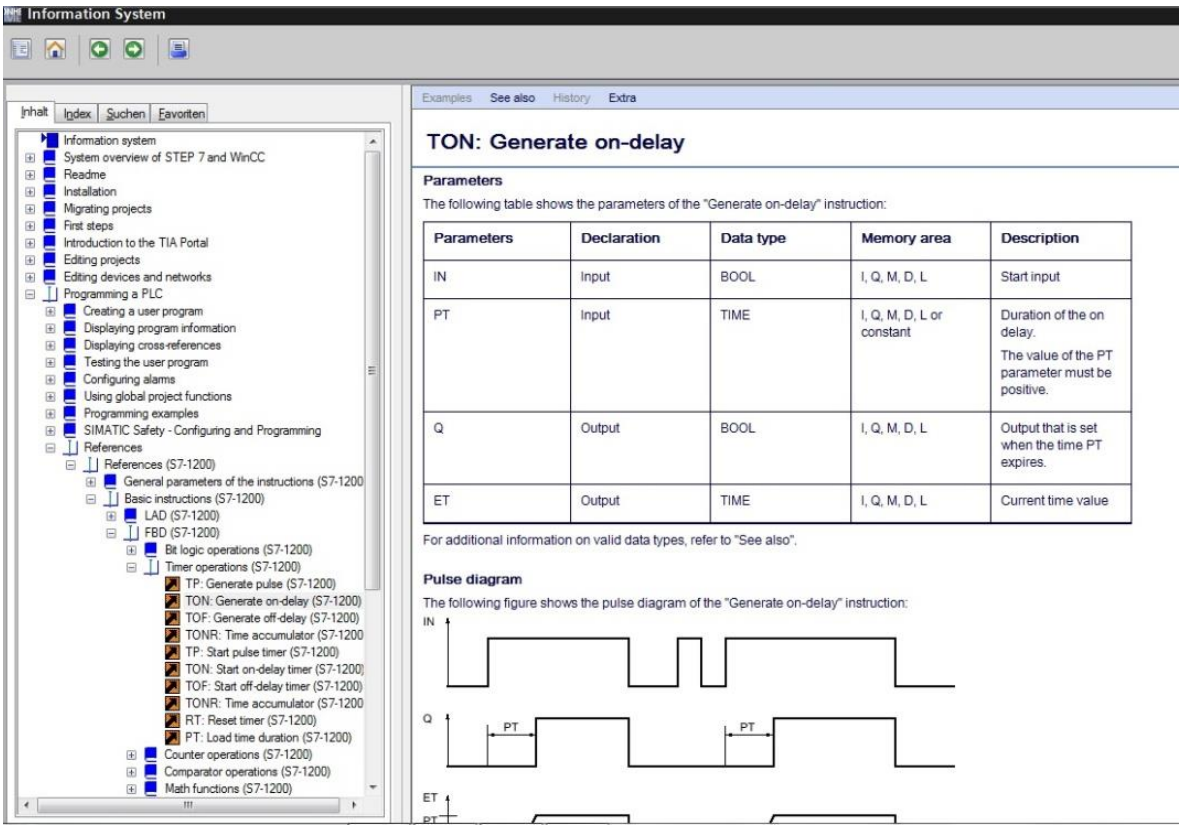


7. Now we can start changing the program.

When generating our solution with the delay, we need an ON delay **'TON'**. It is located under **'Instructions'** in the folder **'Timer operations'**. If you point with the mouse to an object such as the time TON, details about this object will be provided. (→ Instructions → Timer operations → TON)



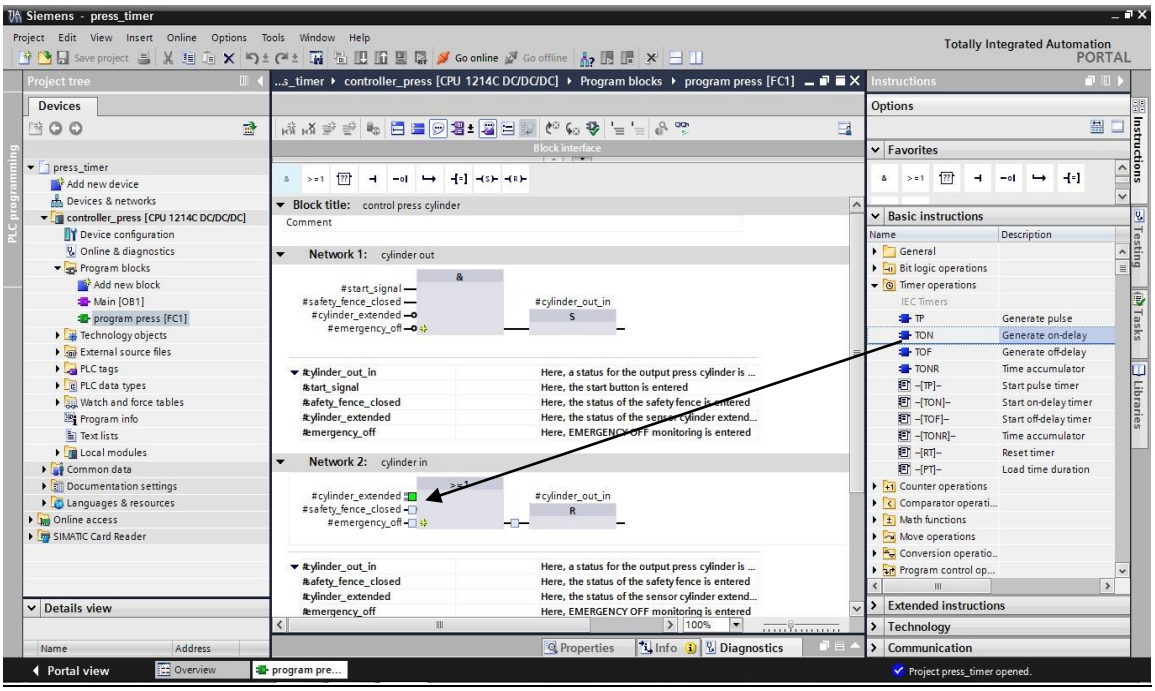
8. If you highlight an object and then press the button **'F1'** on your PC, online help regarding this object is displayed in a window to the right. (→ F1)



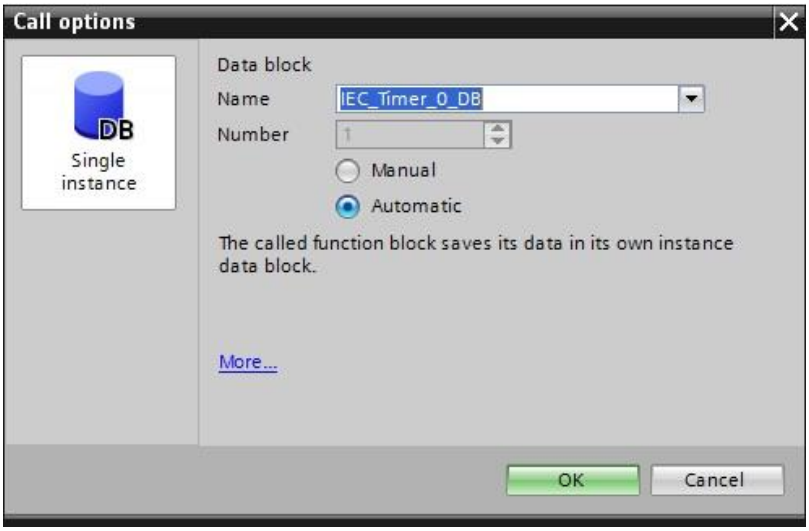
Note

Here, go to online help where details are provided about all timer functions.

9. Next, drag the timer **'TON'** with the mouse to the third contact of the OR function behind the variable **'#cylinder_extended'** (→ TON → #cylinder_extended)



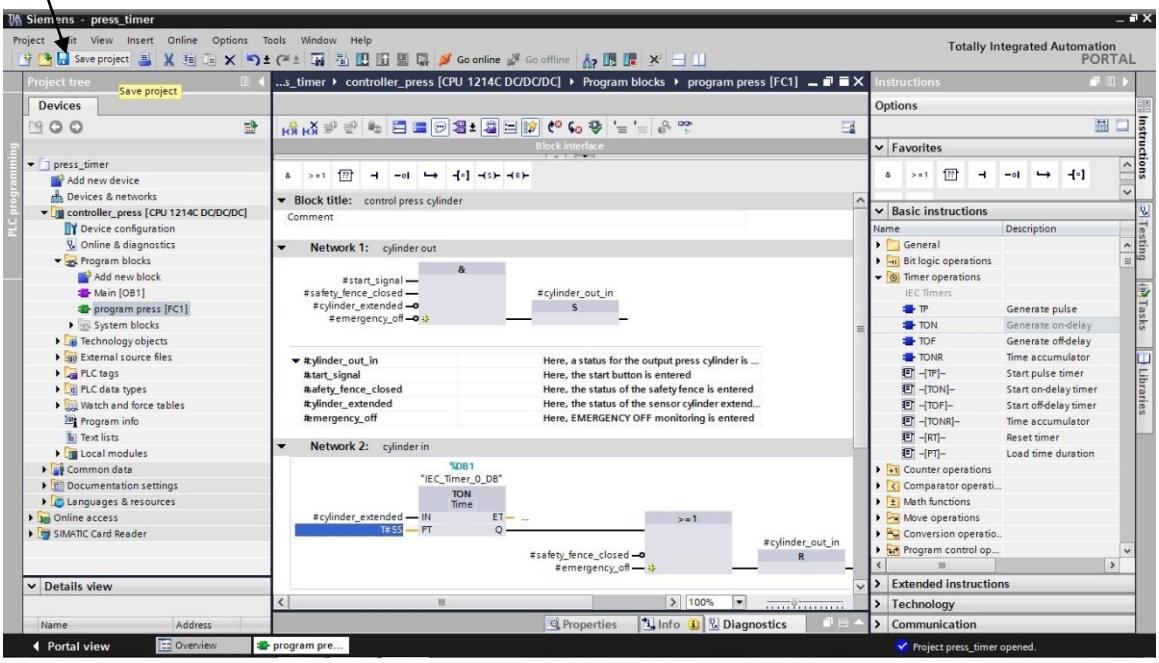
10. For the timing function we need memory. Here, it can be made available only by generating a new instance data block as a **'Single instance'.** (→ OK)



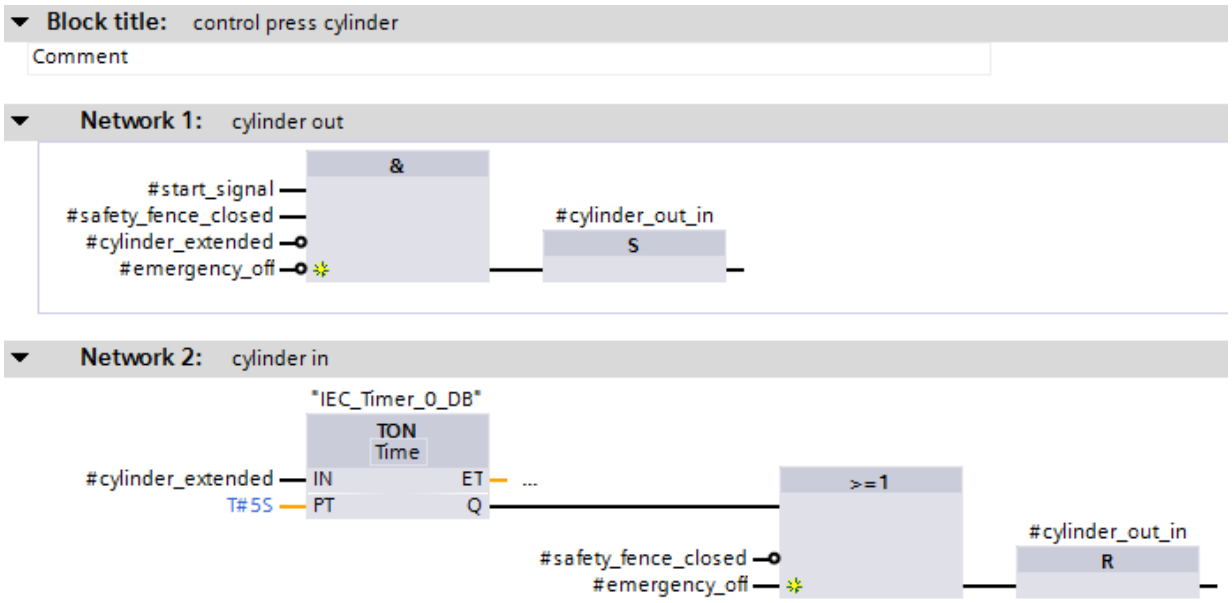
Note

Multi-instances can be used only when programming within a function block. This will be shown below in the example for the IEC counter.

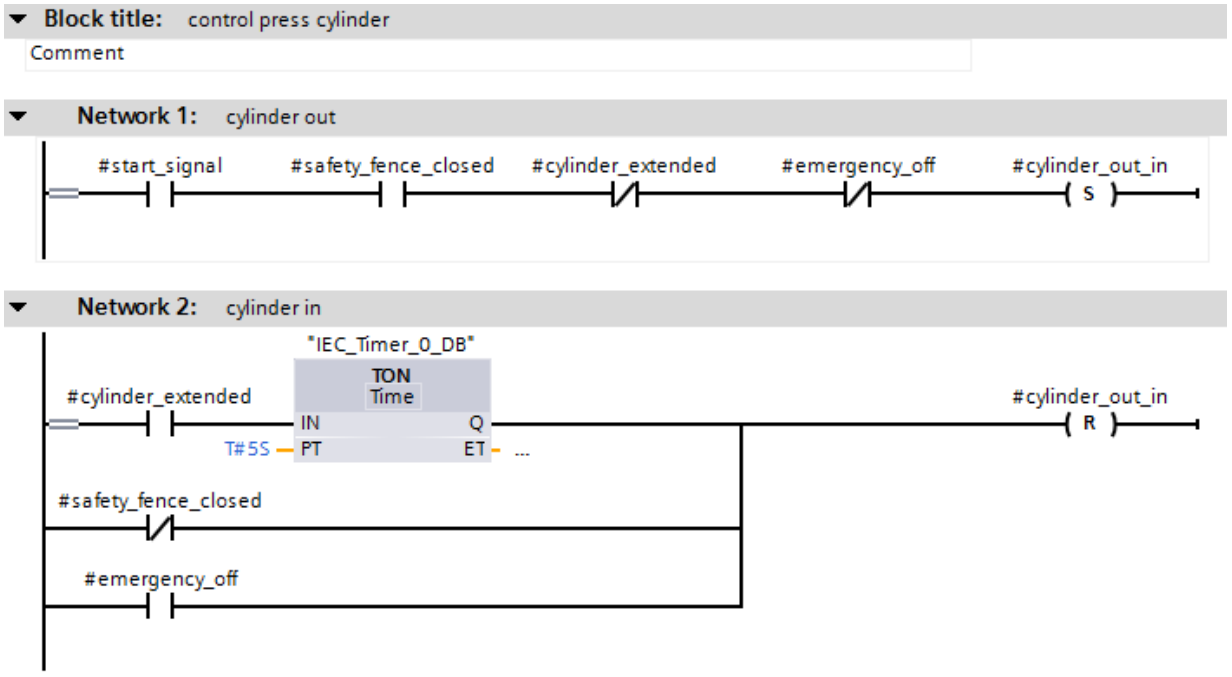
11. Now, connect the time delay **'TON'** with the time base **'t#5s'** for 5 seconds. By clicking on Save project, the project is saved. (→ t#5s → Save project)



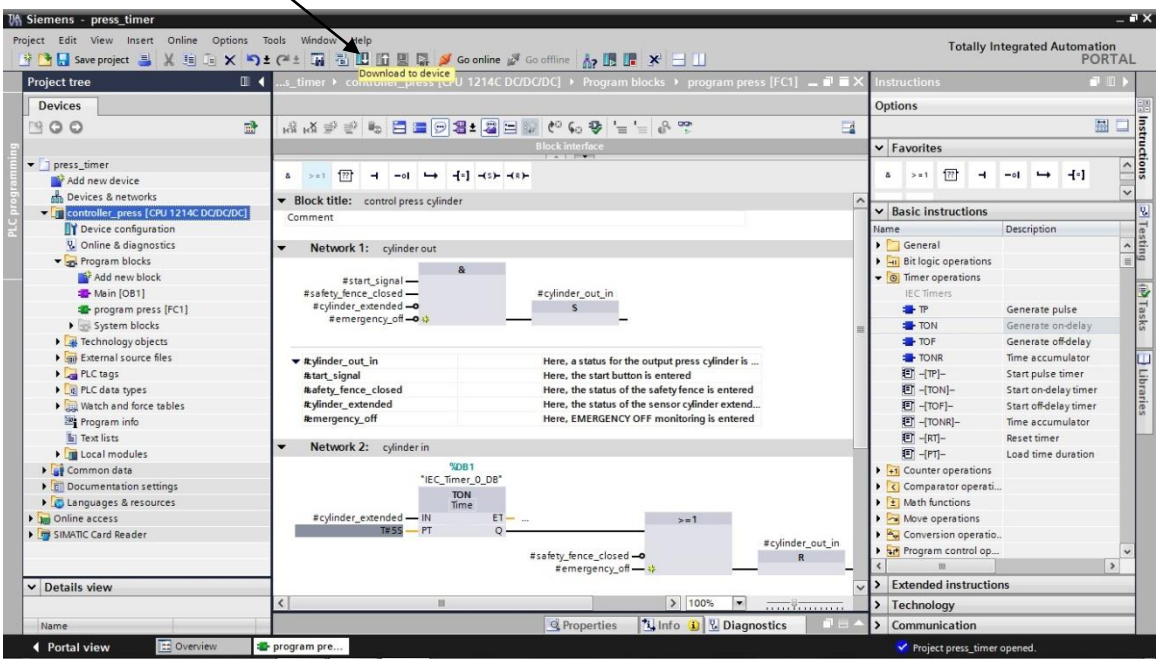
Program in function block diagram (FBD)



Program in ladder diagram (LAD)

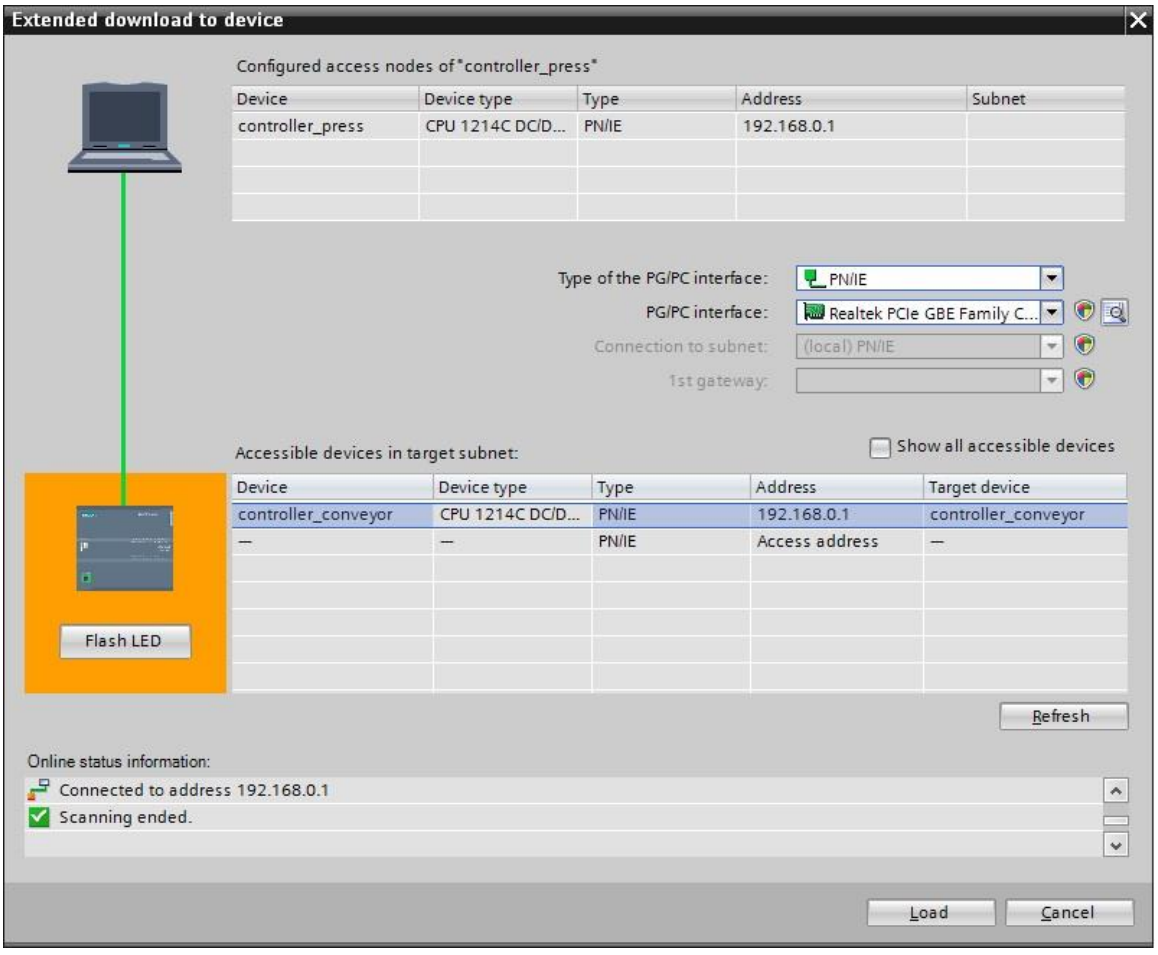


12. To load your entire program into the CPU, highlight the folder **'controller_press'** and then click on the symbol Load to device. (→ controller_press →)

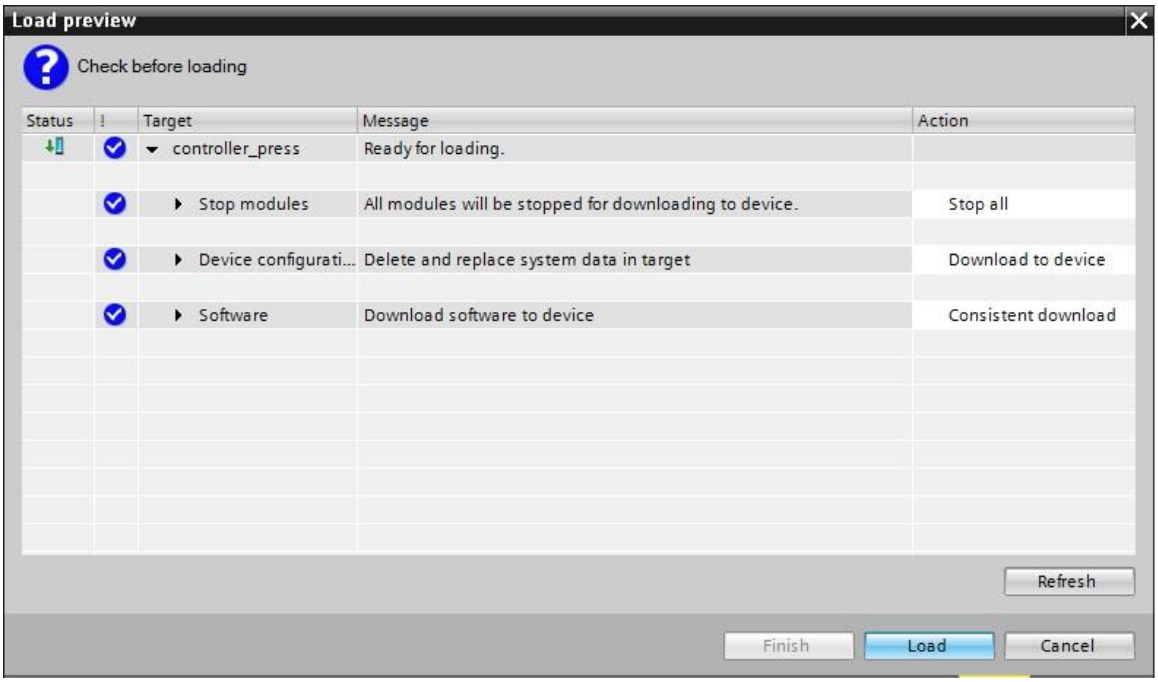


13. If the CPU is in the **'RUN' mode,** you will be asked whether you want to take it to the **'STOP'** mode. Confirm with **'OK'.** (→ OK)

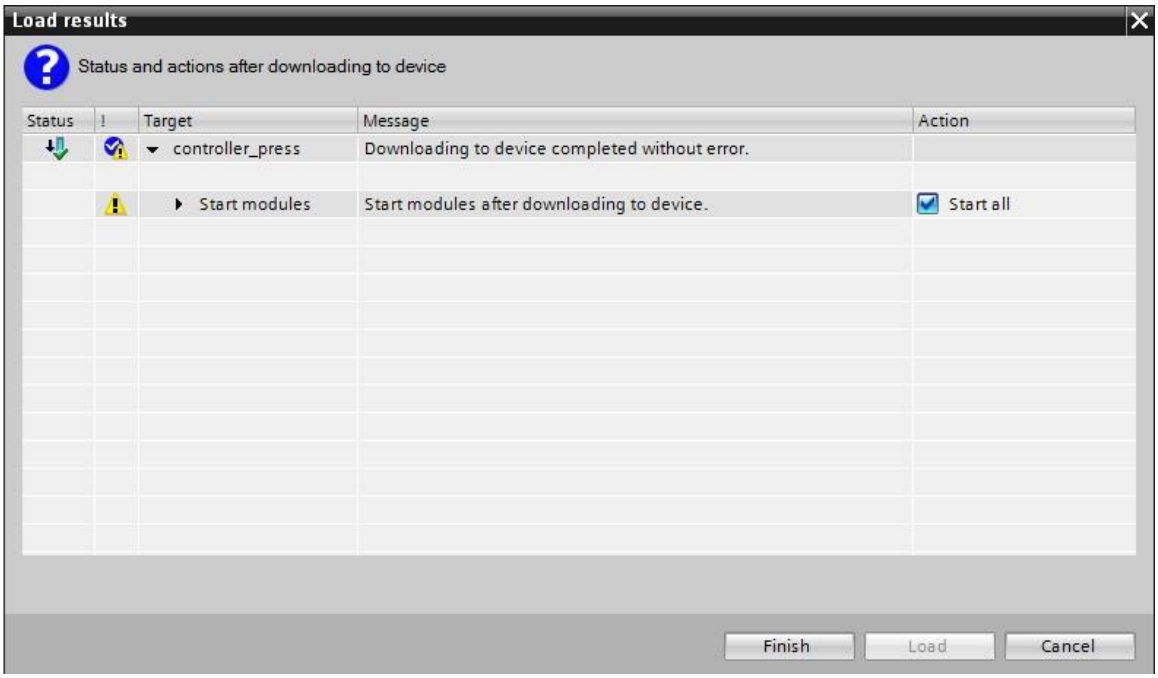
14. If you omitted to specify the PG/PC interface beforehand, a window is displayed where you can do this now. (→ PG/PC interface for loading → Load)



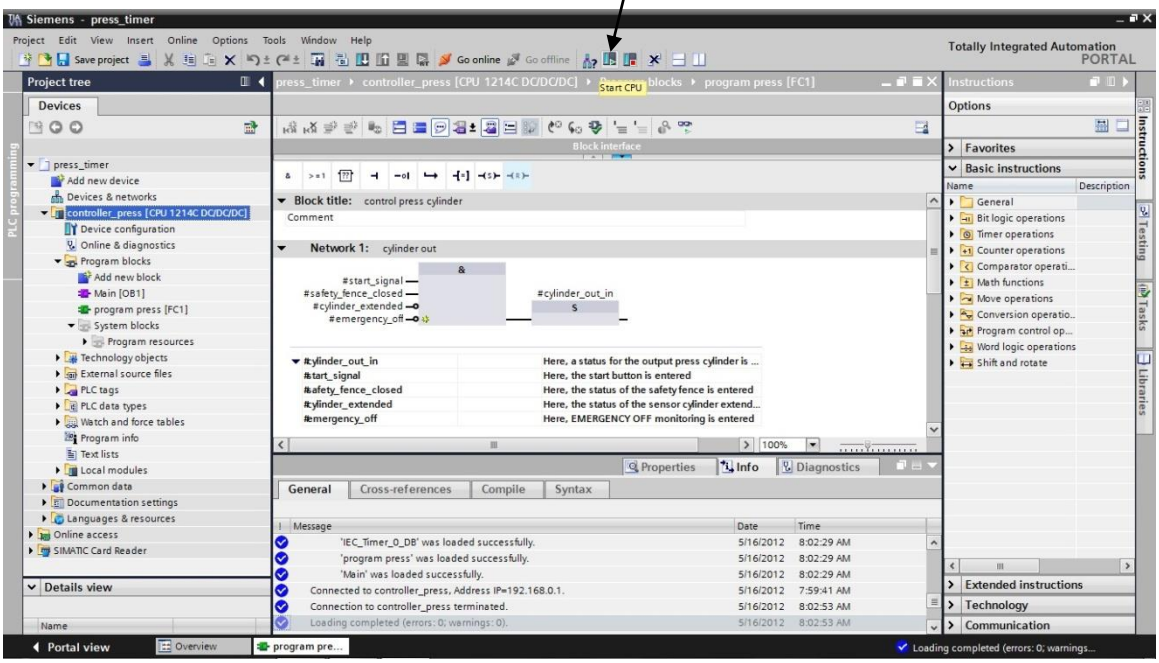
15. Confirm **'Load'** once more. During loading, the status is shown in a window. (→ Load)



16. If loading was successful, it is displayed in a window. Now click on **'Finish'.** (→ Finish)

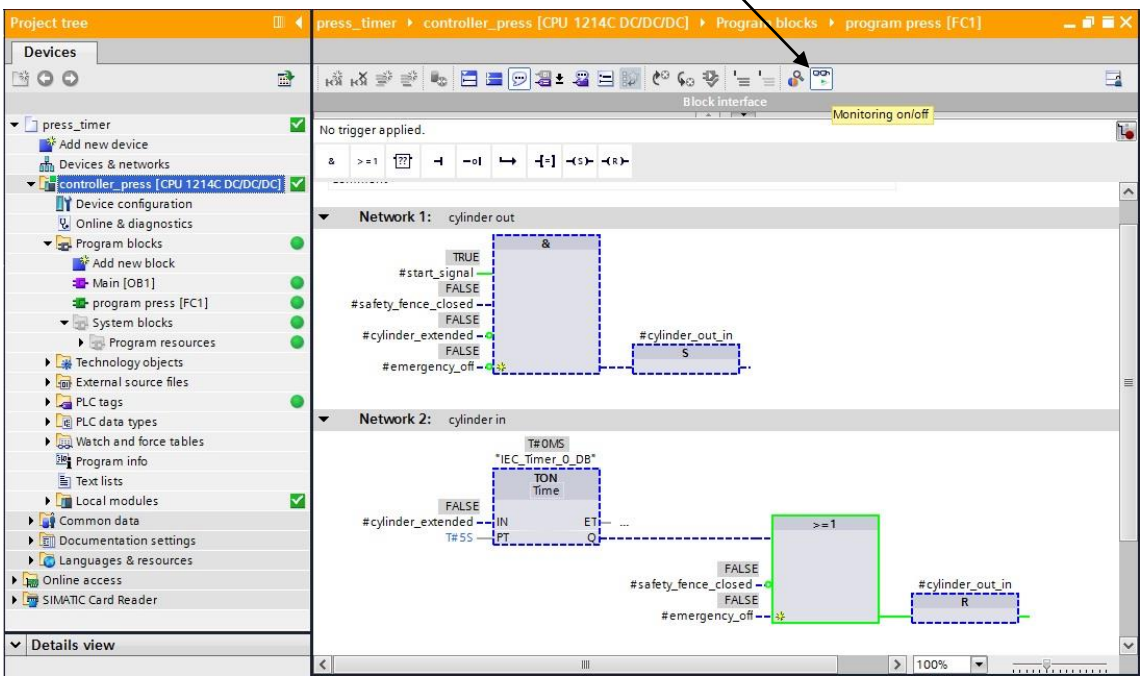


17. Next, start the CPU by clicking on the symbol  . (→ )



18. Confirm the question whether you actually want to start the CPU with **'OK'.** (→ OK)

19. By clicking on the symbol  Monitoring on/off you can, while the program is tested, observe the status of the timer as well as the time that expired. (→ )



6. Sample Task for Conveyor Control with Counter and Multi-Instance

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.

If secondary function blocks -or timers/counters- are called from a block that can be used multiple times, they must not be assigned their own data block.
The required memory is provided as **multi-instance** within the instance DB that is assigned to the function block doing the calling.

In the example below, we add a bottle counter to the function block that already contains a conveyor control dependent on the operating mode.
With this conveyor, 20 bottles are to be transported to a case. When the case is full, the conveyor is stopped and the case has to be exchanged.

With the button 'S1', we want to select the operating mode 'Manual’, and with the button 'S2' the operating mode 'Automatic’.
In the operating mode 'Manual’, the motor is switched on as long as the button 'S3’ is operated; 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).
In addition, there is a sensor 'B0’ that counts the bottles into the case. After counting 20 bottles, the conveyor is stopped.
When a new case is put in place, this has to be confirmed with the button 'S5’.

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
%I 0.6	S5	Button S5 NO Reset counter/new case
%I 0.7	B0	Sensor B0 NO bottle counter
%Q 0.2	M1	Conveyor motor M1

7. Programming the Conveyor with the SIMATIC S7-1200

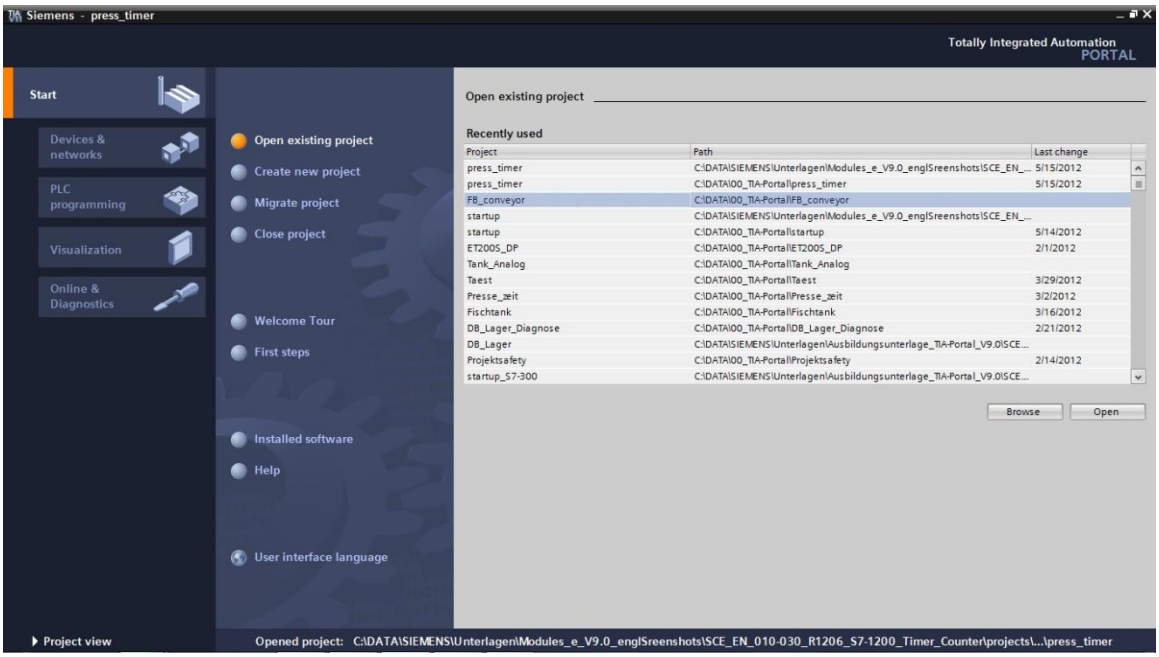
The **'Totally Integrated Automation Portal'** software manages the project and does the programming.

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.

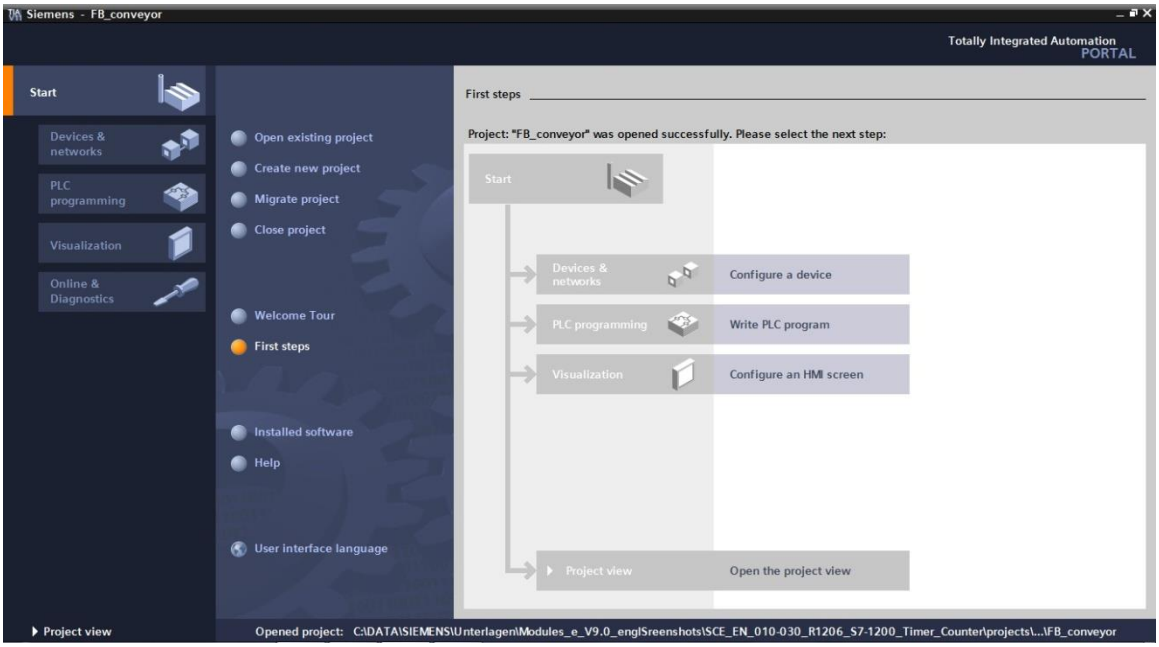
In the steps below, for the SIMATIC S7-1200 a project can be opened, stored under a different name and adapted to the new requirement.

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

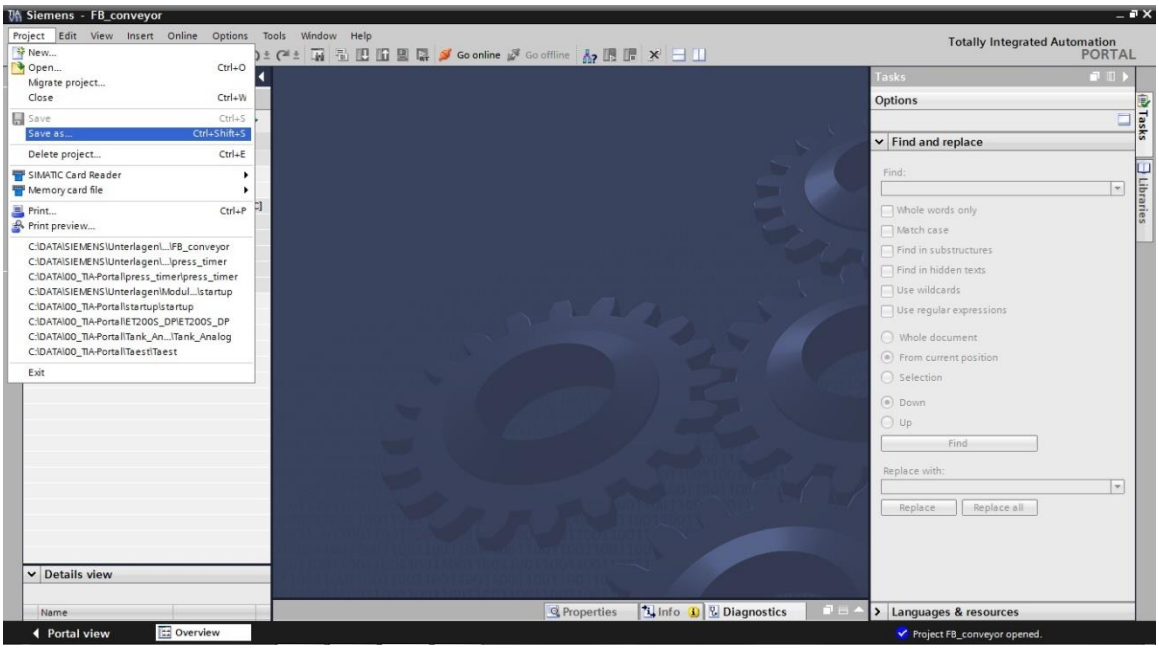
2. We now open the project **"FB_conveyor“** from Module 010-020 in the portal view as the basis for this program. (→ Open existing project → FB_conveyor → Open)



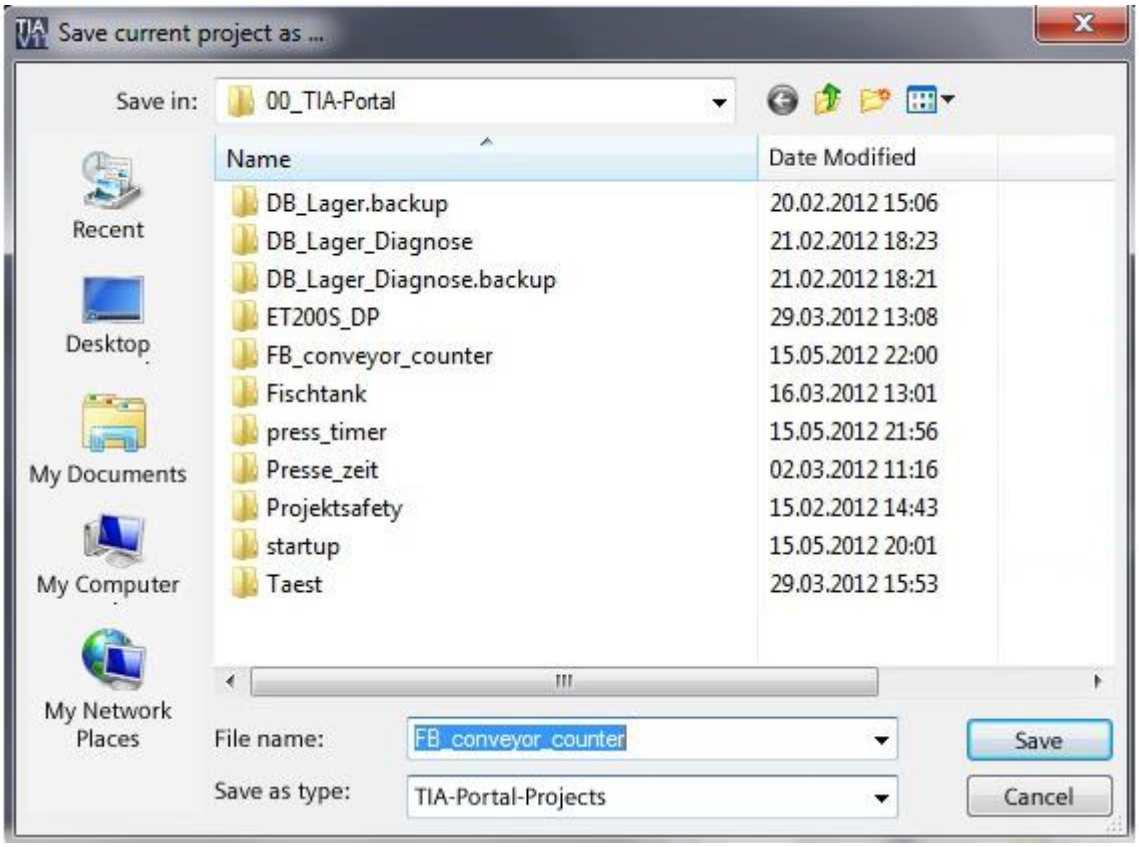
3. Next, **'First Steps'** are suggested for the configuration. We want to **'Open project view'.** (→ Open project view)



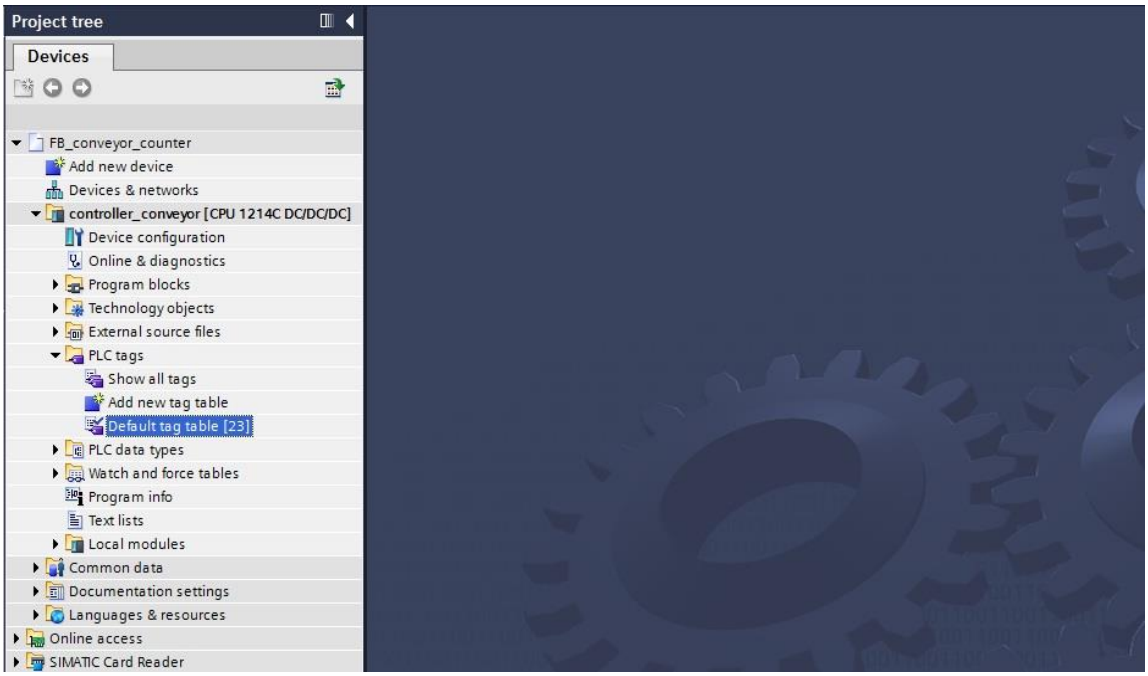
4. Now, we first save the project under a different name. (→ Project → Save as)



5. Next, **'Save'** the project under the new name **'FB_conveyor_counter'.** (→ FB_conveyor_counter → Save)



6. To set up new global variables, open with a double click on **'PLC tags'** under the **'controller_conveyor' 'Default tag table.** (→ controller_conveyor → PLC tags → Default tag table)

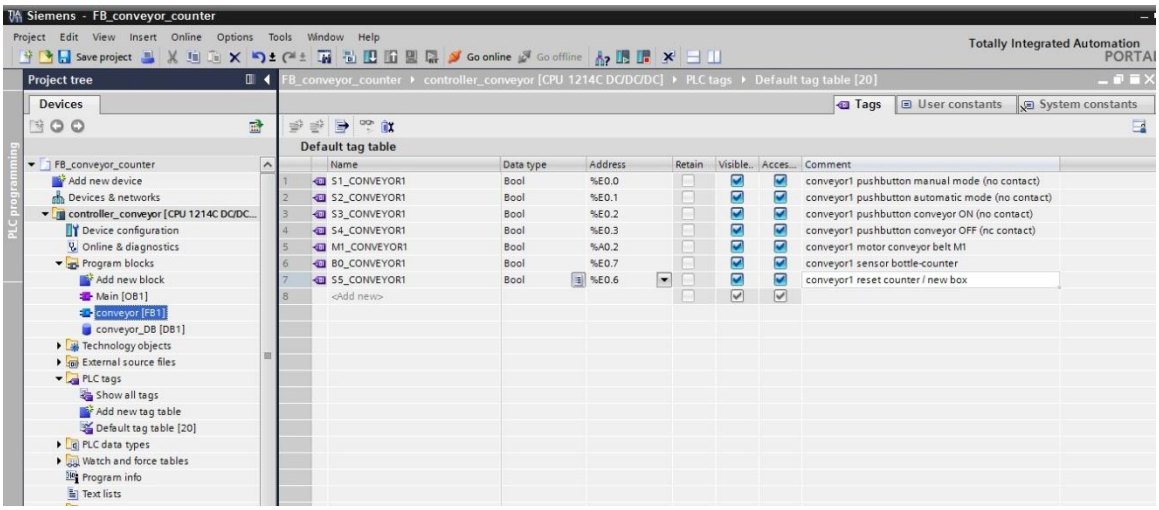


7. Change the tag table according to the default setting.

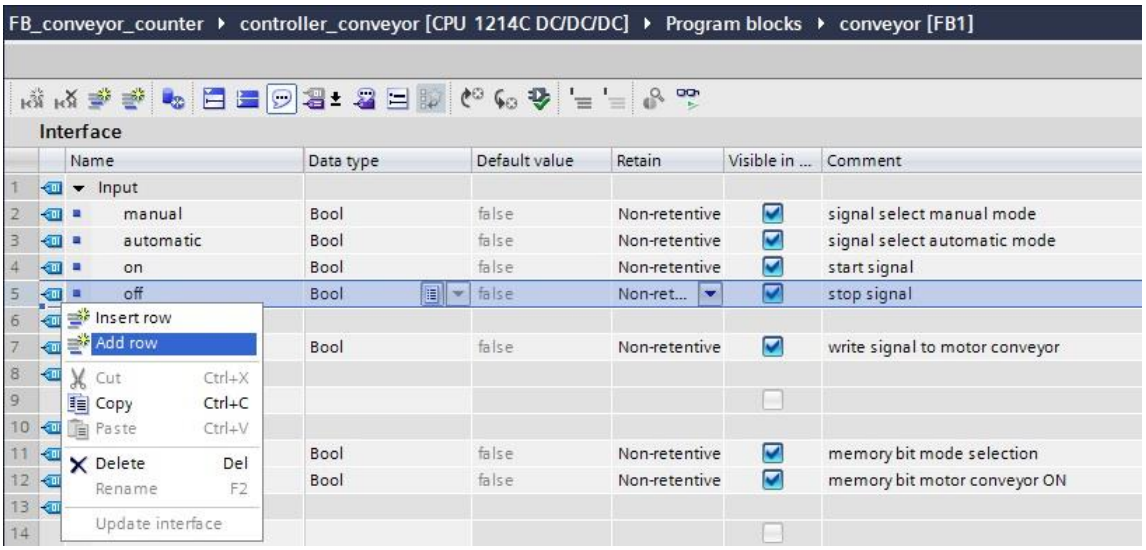
Next, set up the two global variables **'B0'** and **'S5'.** (→ B0/Bool/%I0.7/sensor bottle counter → S5/Bool/%I0.6/reset counter/new box)

	Name	Data type	Address	Retain	Visible..	Acces...	Comment
1	S1_CONVEYOR1	Bool	%E0.0	☐	☑	☑	conveyor1 pushbutton manual mode (no contact)
2	S2_CONVEYOR1	Bool	%E0.1	☐	☑	☑	conveyor1 pushbutton automatic mode (no contact)
3	S3_CONVEYOR1	Bool	%E0.2	☐	☑	☑	conveyor1 pushbutton conveyor ON (no contact)
4	S4_CONVEYOR1	Bool	%E0.3	☐	☑	☑	conveyor1 pushbutton conveyor OFF (nc contact)
5	M1_CONVEYOR1	Bool	%A0.2	☐	☑	☑	conveyor1 motor conveyor belt M1
6	B0_CONVEYOR1	Bool	%E0.7	☐	☑	☑	conveyor1 sensor bottle-counter
7	S5_CONVEYOR1	Bool	%E0.6	☐	☑	☑	conveyor1 reset counter / new box

8. To make the changes in the program, open the block **'conveyor[FB1]'** with a double click. (→ conveyor[FB1])



9. First, add two rows under Interface for the input variables. (→ Interface → Input → Add row)



10. When declaring the local variables, we are adding the following variables.

Input:

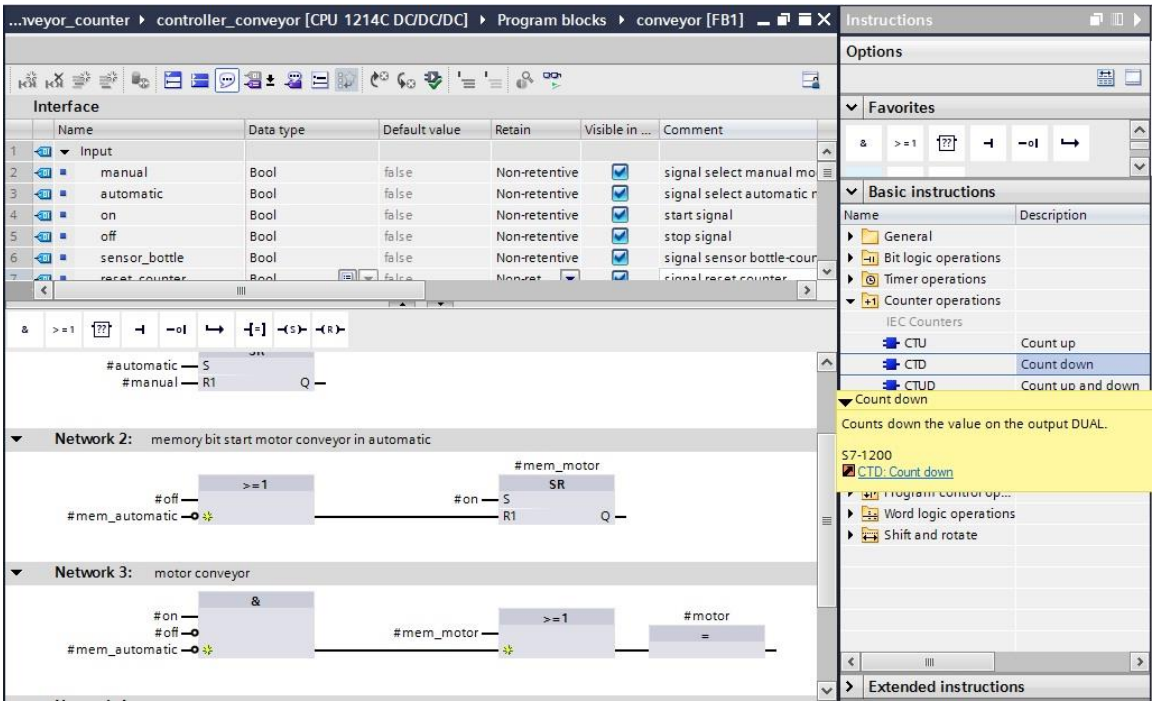
sensor_bottle	Here, the sensor of the bottle counter is polled
reset_counter	Here, the signal for resetting the counter is entered

Interface

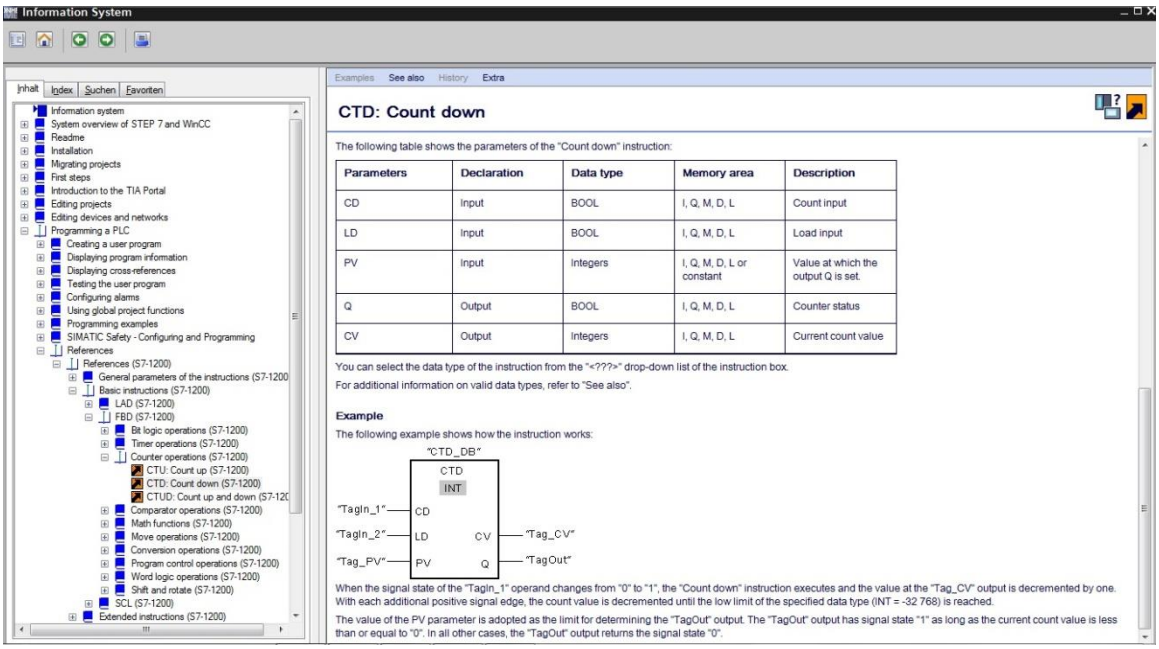
	Name	Data type	Default value	Retain	Visible in ...	Comment
1	Input					
2	manual	Bool	false	Non-retentive	☑	signal select manual mode
3	automatic	Bool	false	Non-retentive	☑	signal select automatic mode
4	on	Bool	false	Non-retentive	☑	start signal
5	off	Bool	false	Non-retentive	☑	stop signal
6	sensor_bottle	Bool	false	Non-retentive	☑	signal sensor bottle-counter
7	reset_counter	Bool	false	Non-ret...	☑	signal reset counter
8	<Add new>				☐	
9	Output					
10	motor	Bool	false	Non-retentive	☑	write signal to motor conveyor
11	InOut					
12	<Add new>				☐	
13	Static					
14	mem_automatic	Bool	false	Non-retentive	☑	memory bit mode selection
15	mem_motor	Bool	false	Non-retentive	☑	memory bit motor conveyor ON
16	Temp					
17	<Add new>				☐	

11. Now we can start changing the program.

As we generate our solution with the counter, we need a down counter **'CTD'**. It is located under **'Instructions'** in the folder **'Counters'**. If you point with the mouse to an object such as the counter CTD, you will be provided with detailed information about this object. (→ Instructions → Counters → CTD)



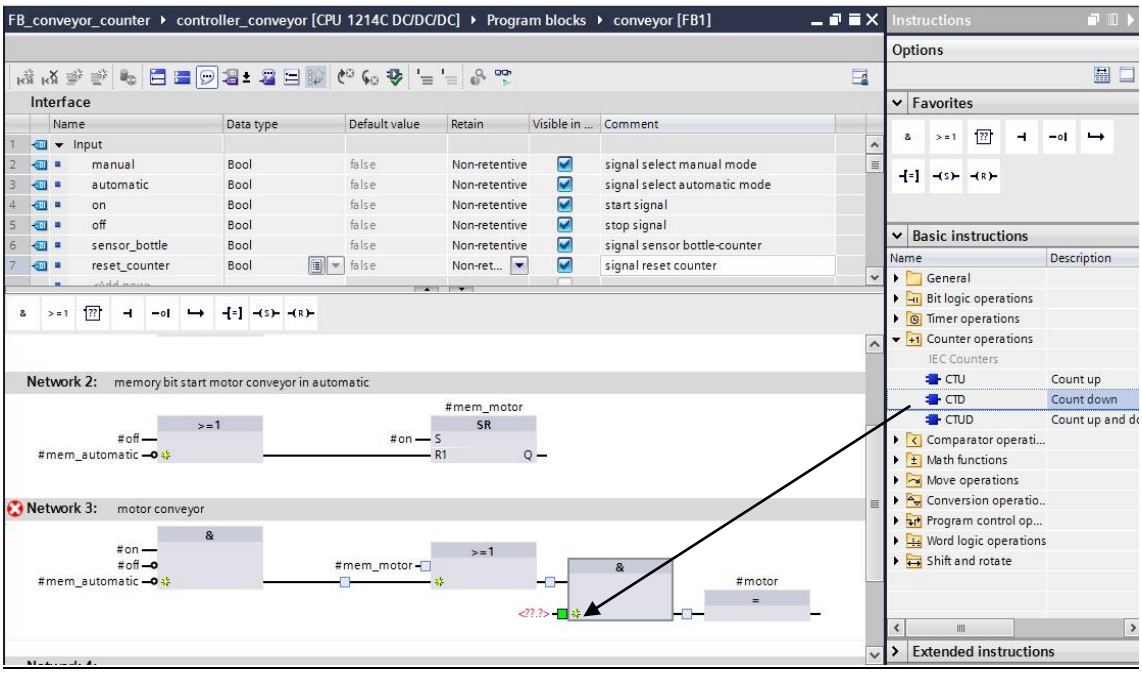
12. If you highlight an object and then press the **'F1'** key on your PC, online help regarding this object is displayed in a window to the right. (→ F1)



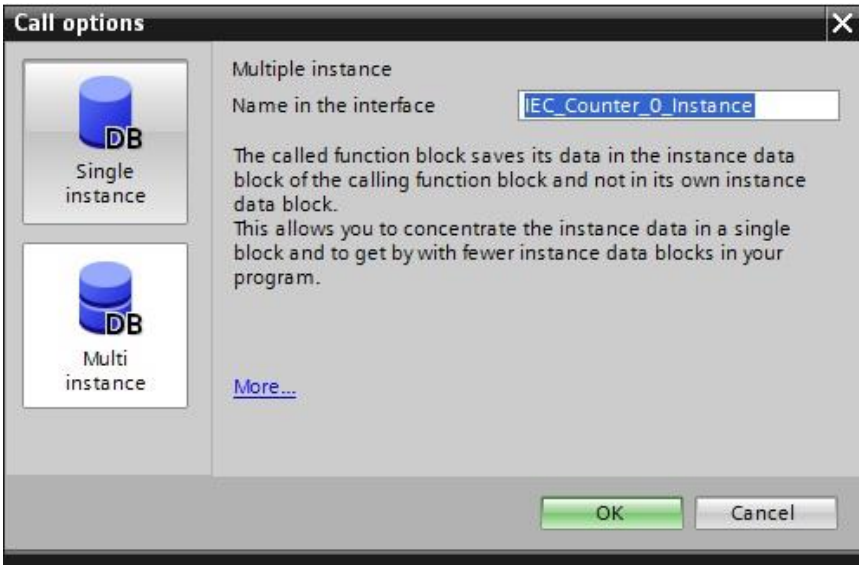
Note

Here, go to online help to obtain the details about all counters.

13. Now, first insert an AND between OR and assignment, and then drag the counter **'CTD'** to the second contact of the AND function. (→ & → CTD)



14. We need memory for the counter function. Here, the function block makes it available within the instance data block as **'Multi-Instance'**, without generating a new instance data block. (→ Multi-Instance → OK)

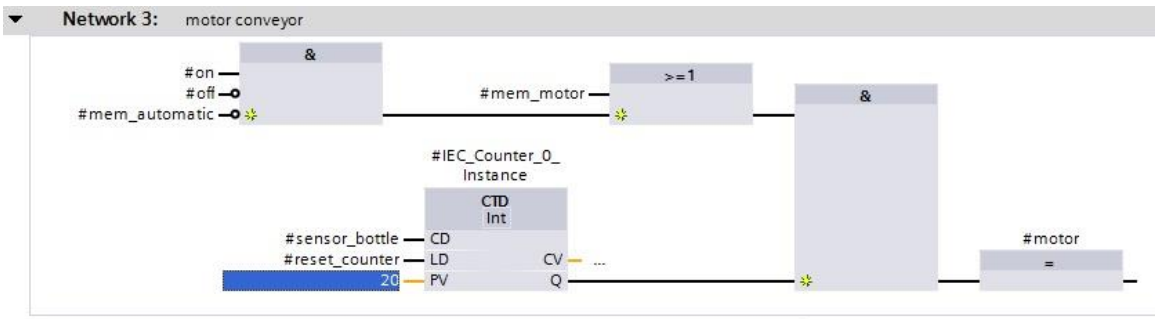


Note

A multi-instance can be used only when programming within a function block.

15. Now, connect the down counter **'CTD'** to the specified value **'PV'** for the 20 bottles and connect the input **'CD'** to **'#sensor_bottle'**, and the input **'LD'** to **'#reset_counter'**. Next, negate the second contact of the AND function.

Click on Save project and the project will be saved. (→ 20 → #sensor_bottle → #reset_counter → → Save project)

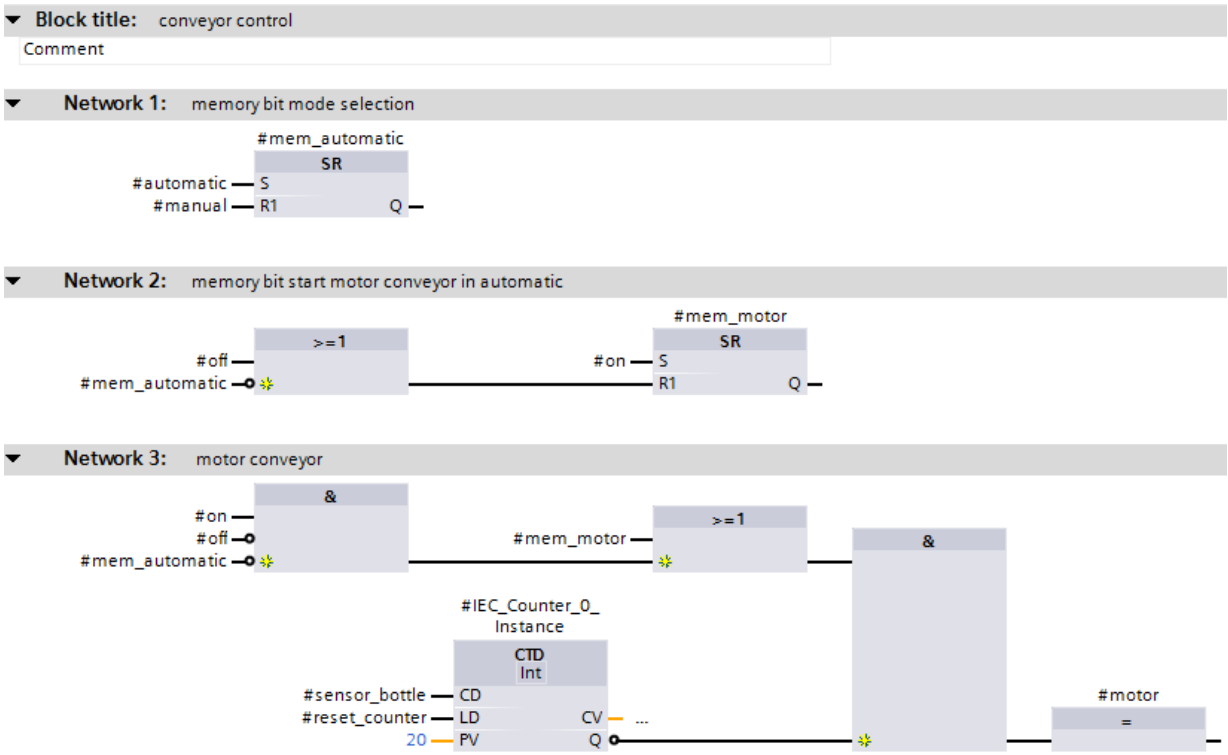


#mem_automatic		memory bit mode selection
#mem_motor		memory bit motor conveyor ON
#on		start signal
#off		stop signal
#motor		write signal to motor conveyor
#IEC_Counter_0_Instance		
#sensor_bottle		signal sensor bottle-counter
#reset_counter		signal reset counter

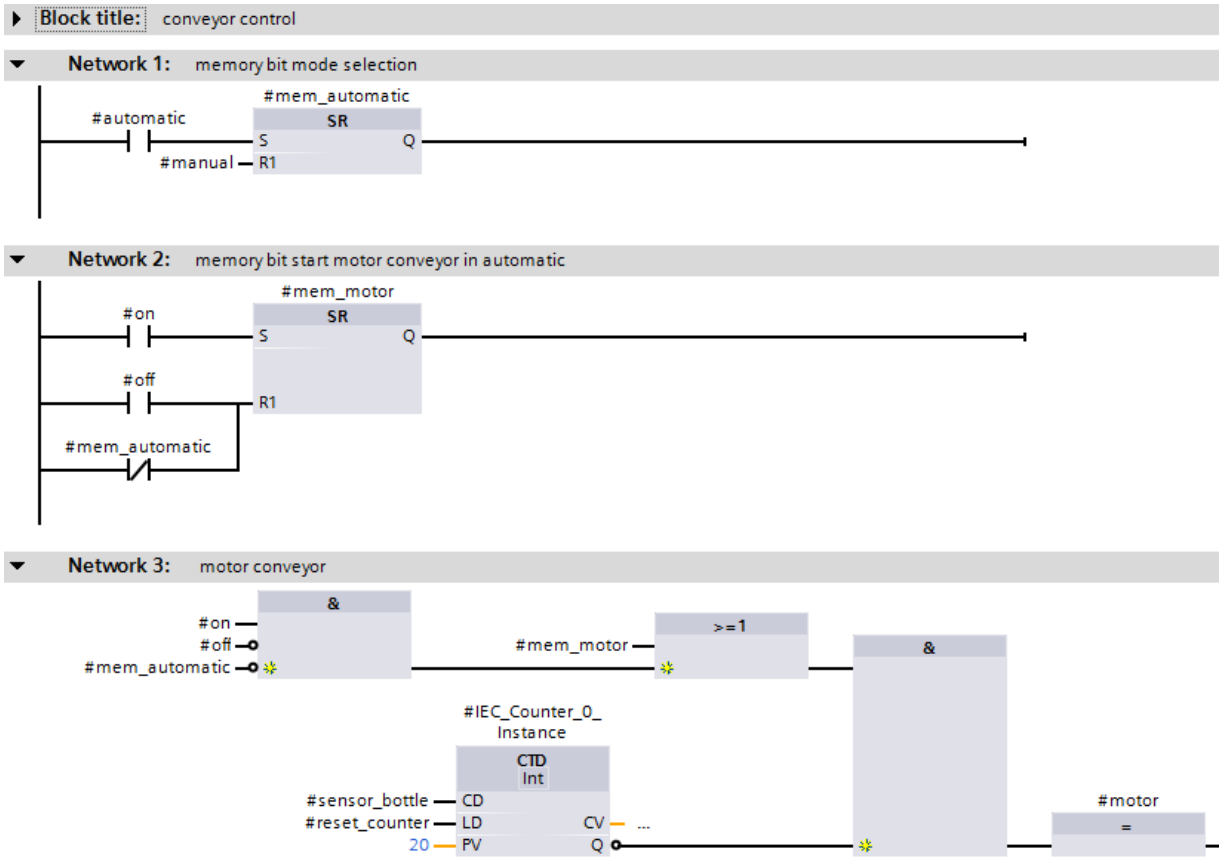
Note

The down counter is most suitable for counting specified quantities, since simply the binary output **'Q'** can be used for further connections. Otherwise, a comparator would have to be programmed.

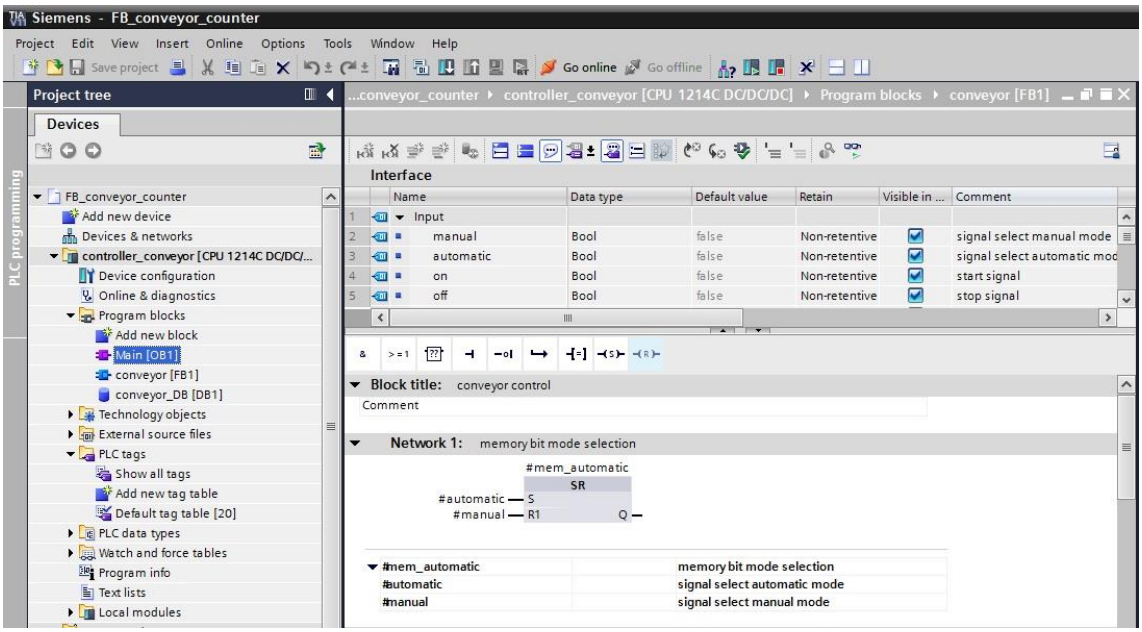
Program in function block diagram (FBD)



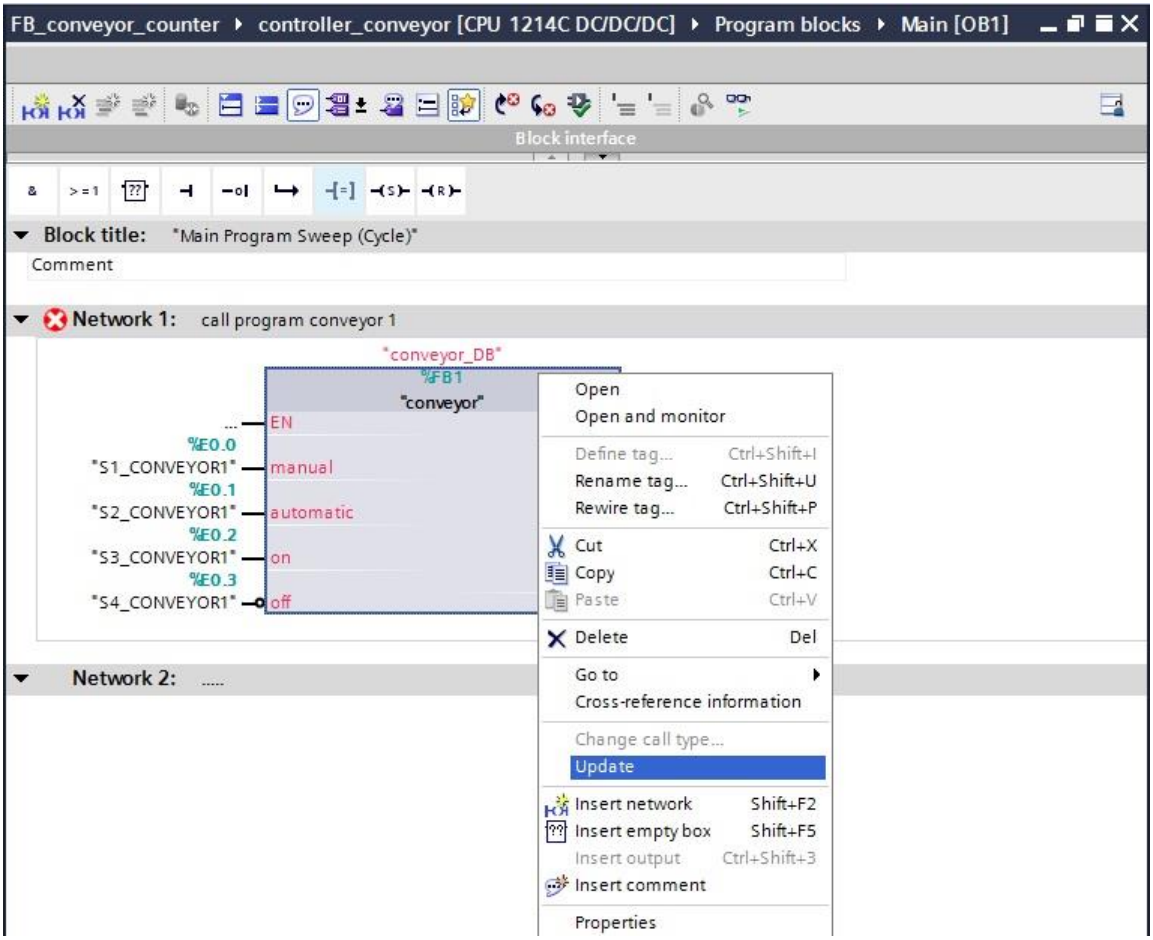
Program in ladder diagram (LAD)



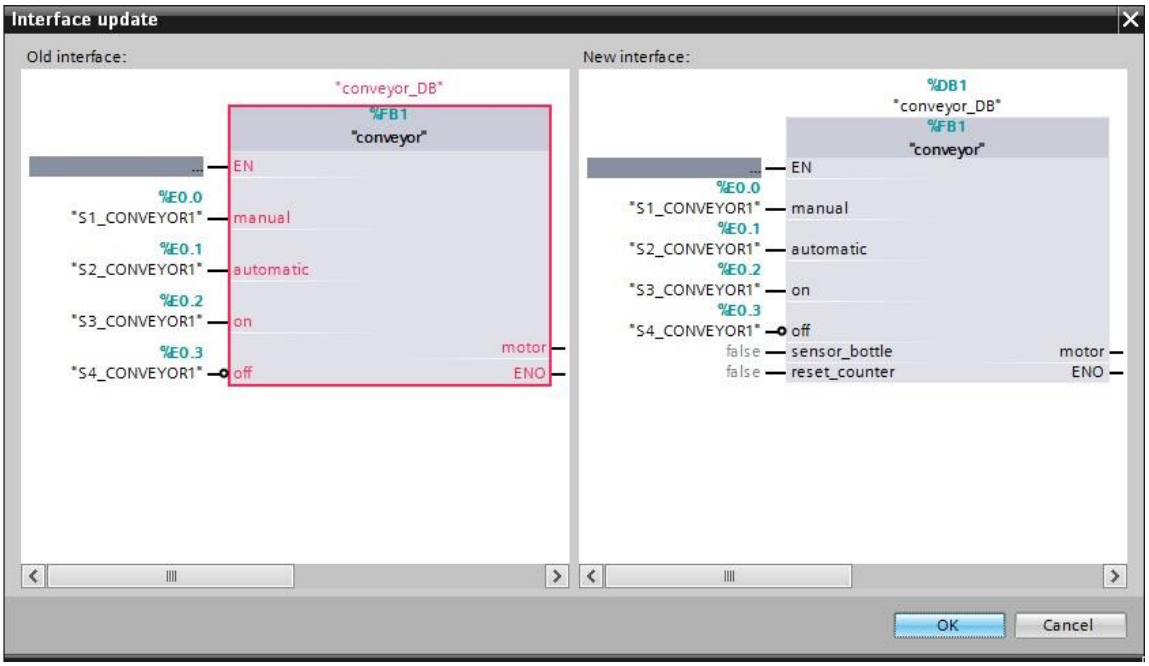
16. Now, open the block **'Main[OB1]'** to update the call of block **'conveyor[FB1]'** (→ Main[OB1])



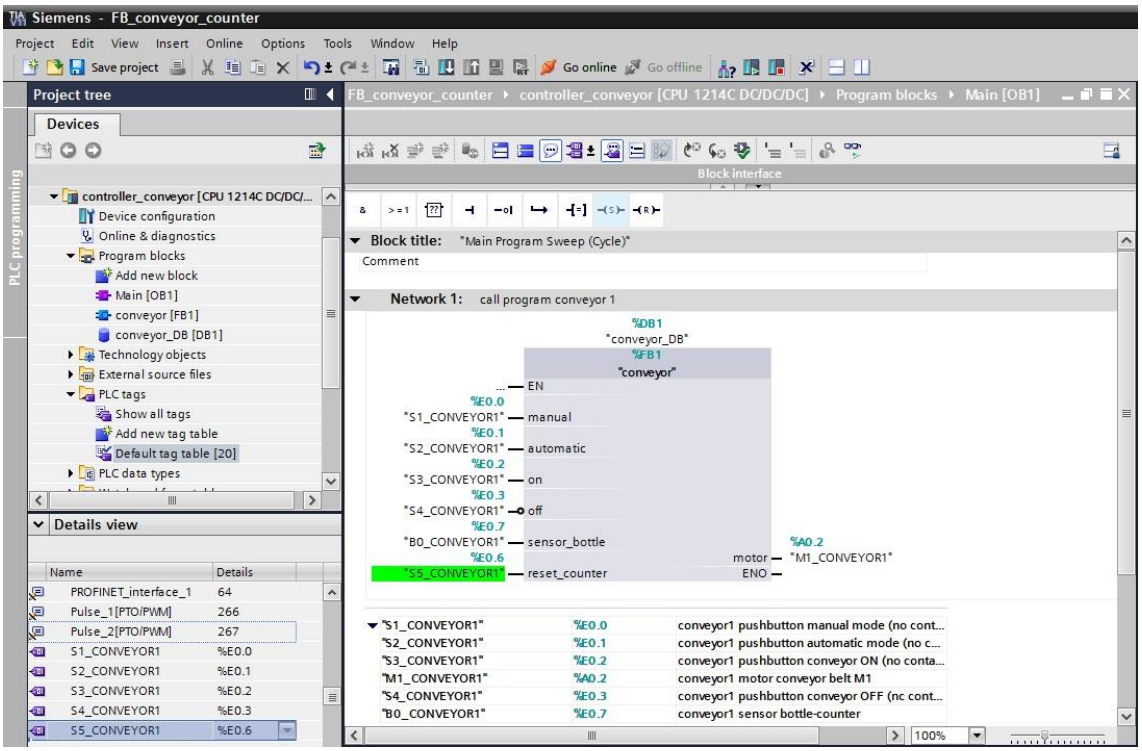
17. In the block **'Main[OB1]'**, click with the right mouse key on **"conveyor"** and then on **'Update'.** (→ Main[OB1] → Update)



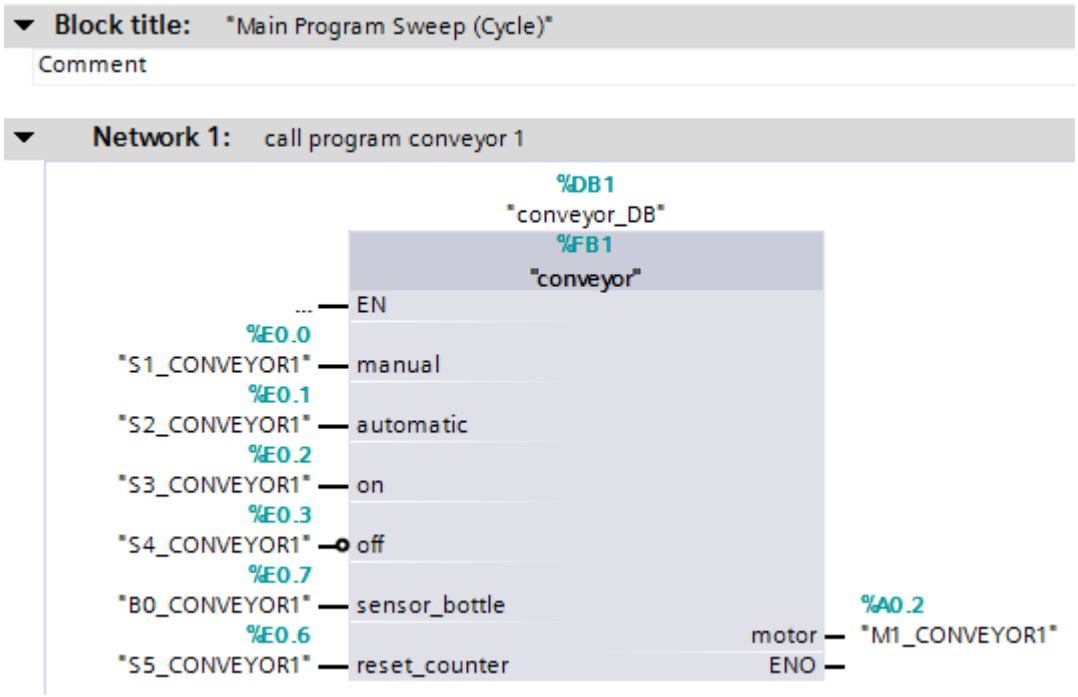
18. Next, select the **'New Interface'** and confirm with **'OK'.** (→ New interface → OK)



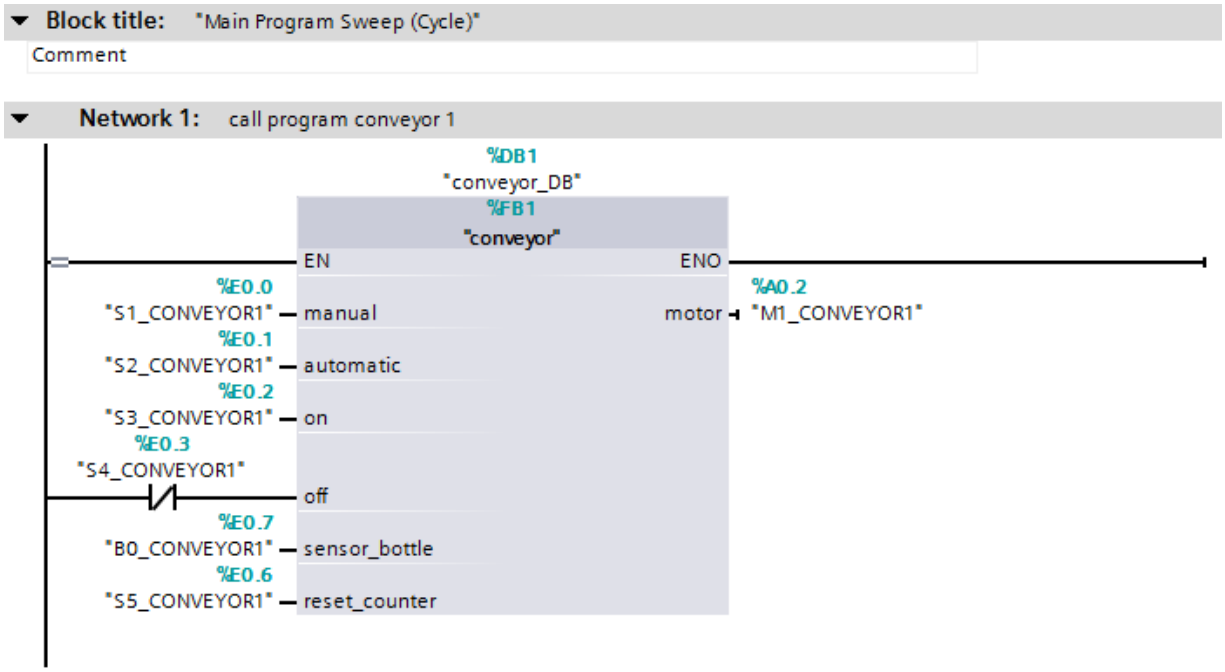
19. Now connect the two new input variables to the PLC tags “B0” and “S5” shown here. Then click on Save project, and the project will be saved. (→ Save project)



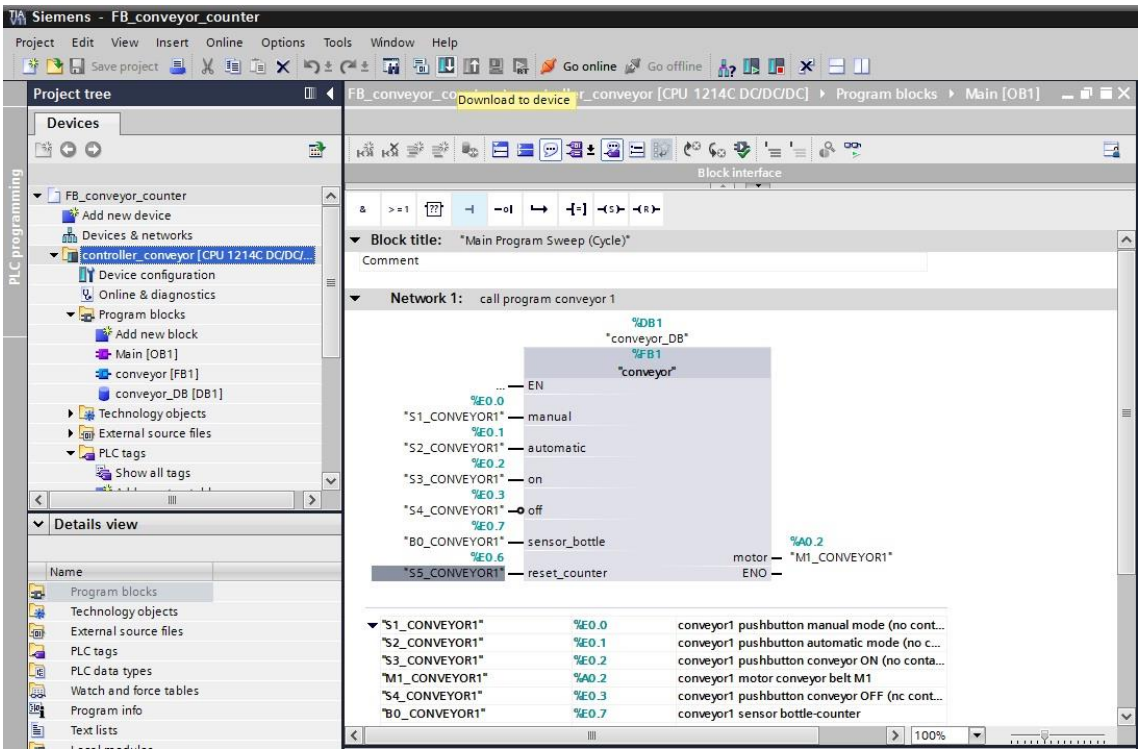
Program in function block diagram (FBD)



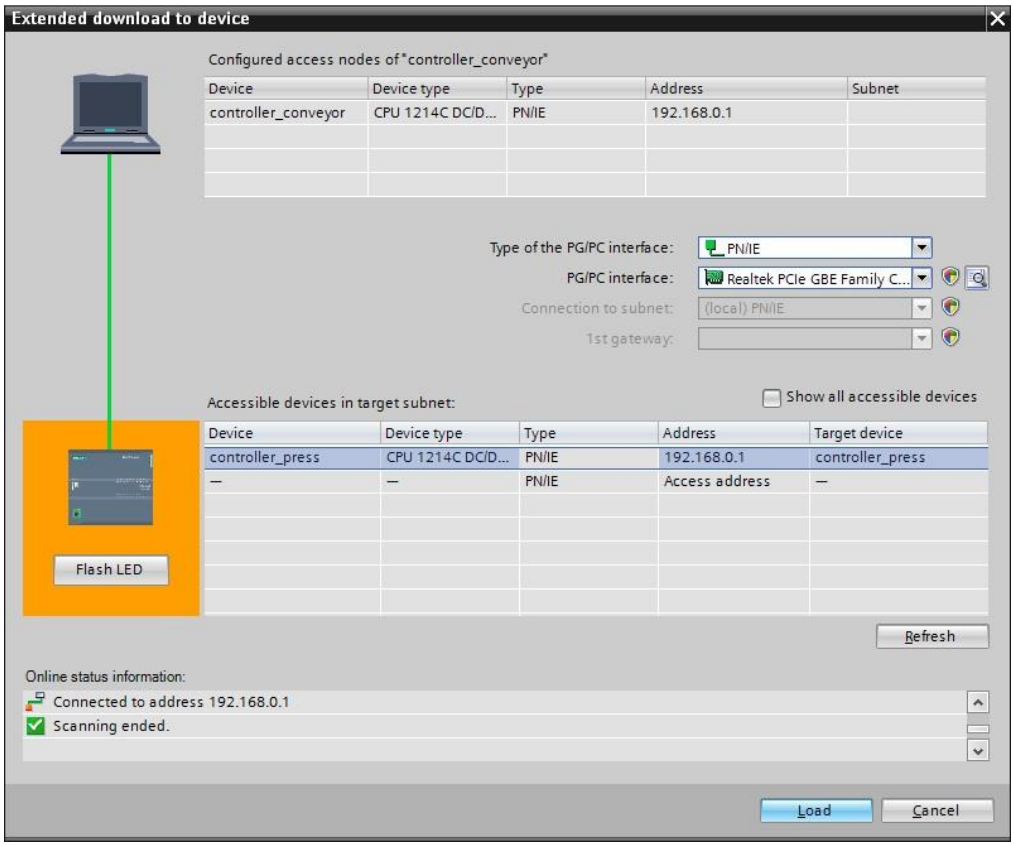
Program in ladder diagram (LAD)



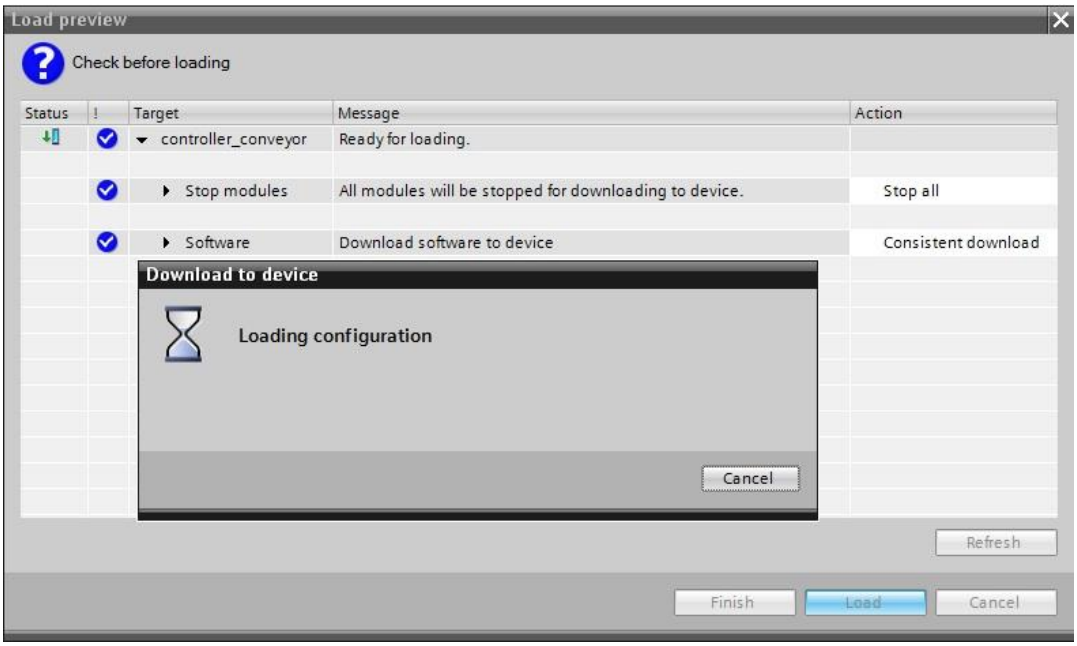
20. To load your entire program into the CPU, highlight the folder **'controller_conceyor'**, and then click on the symbol Load to device. (→ controller_conveyor →)



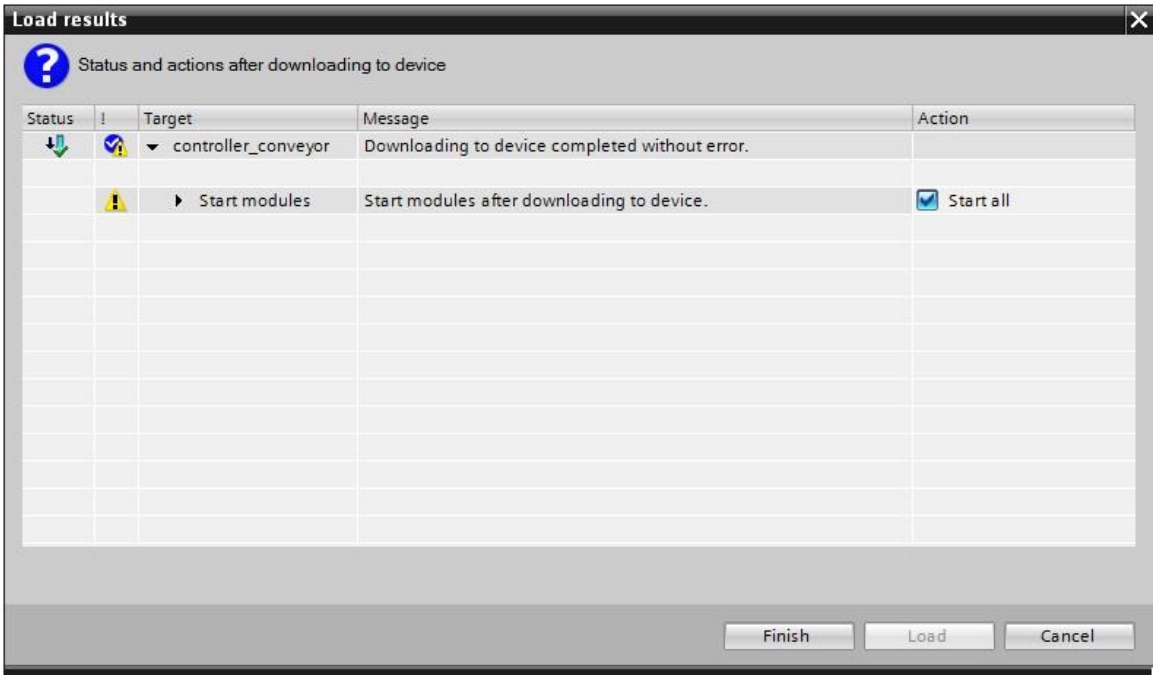
21. Setting the Interface



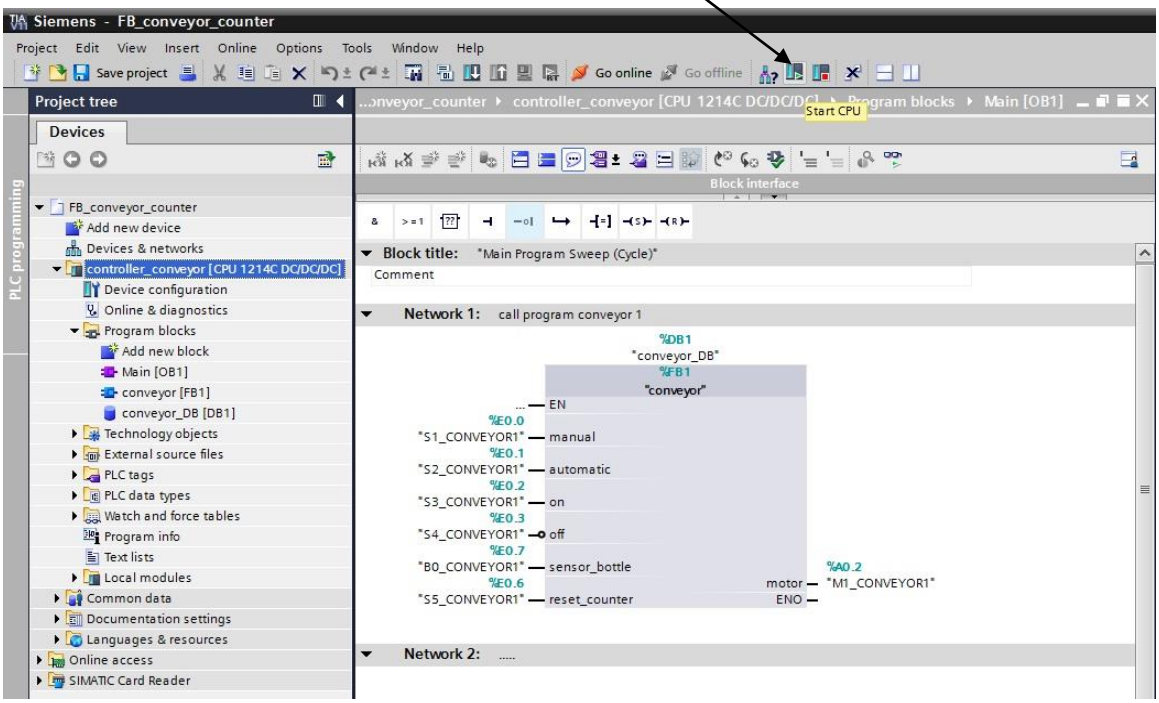
22. Confirm **'Load'** once more. During loading, the status is displayed in a window. (→ Load)



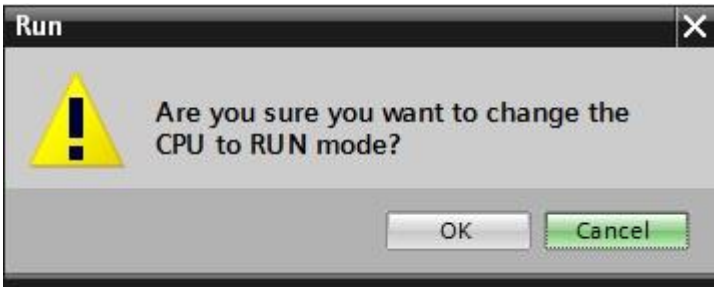
23. If loading was successful, it is displayed in a window. Click on **'Finish'.** (→ Finish)



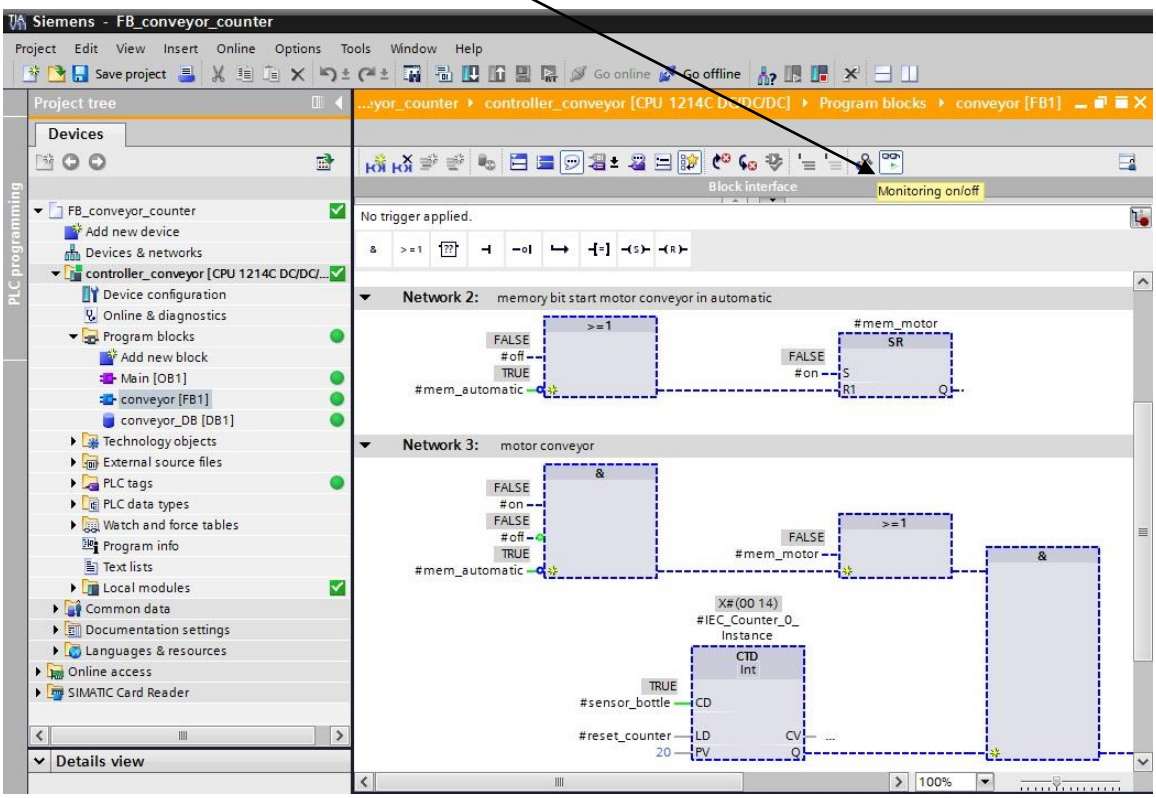
24. Next, start the CPU by clicking on the symbol  .



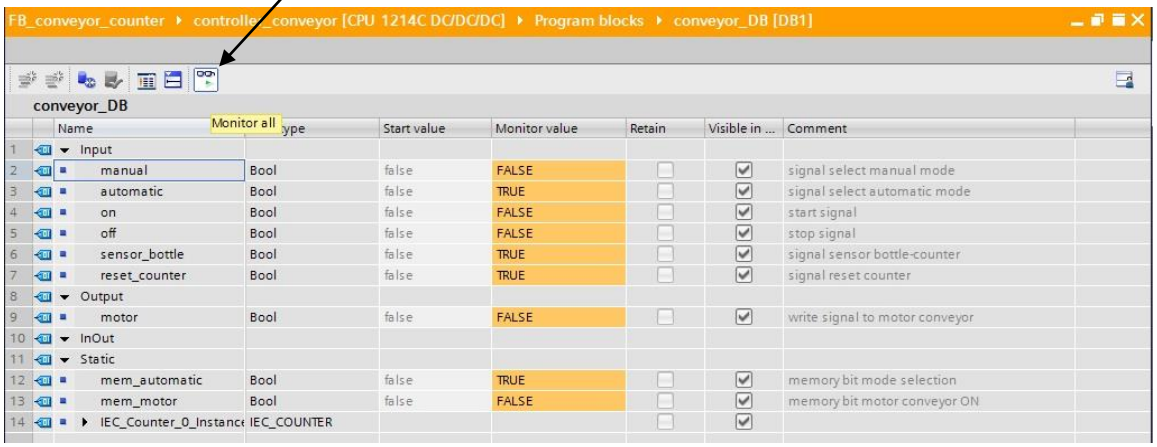
25. Confirm the question whether you actually want to start the CPU with **'OK'.** (→ OK)



26. By clicking on the symbol  Monitoring On/Off, you can observe the counter status while the program is tested.



27. By clicking on the symbol  Monitoring On/Off, you can observe the open data block while the program is tested.



	Name	Type	Start value	Monitor value	Retain	Visible in ...	Comment
1	Input						
2	manual	Bool	false	FALSE		☑	signal select manual mode
3	automatic	Bool	false	TRUE		☑	signal select automatic mode
4	on	Bool	false	FALSE		☑	start signal
5	off	Bool	false	FALSE		☑	stop signal
6	sensor_bottle	Bool	false	TRUE		☑	signal sensor bottle-counter
7	reset_counter	Bool	false	TRUE		☑	signal reset counter
8	Output						
9	motor	Bool	false	FALSE		☑	write signal to motor conveyor
10	InOut						
11	Static						
12	mem_automatic	Bool	false	TRUE		☑	memory bit mode selection
13	mem_motor	Bool	false	FALSE		☑	memory bit motor conveyor ON
14	IEC_Counter_0_Instance	IEC_COUNTER				☑	