12-1 Cursus

HMI programming in Siemens TIA Portal V16

Introduction

Due to production processes are becoming more and more complex and requirements for machine and plant functionality are increasing, operators need a powerful tool for controlling and monitoring production plants. An HMI system (human-machine interface) represents the interface between man (operator) and process (machine/plant). It is the controller that actually controls the process. Hence, there is an interface between the operator and WinCC (at the HMI device) and an interface between WinCC and the controller.

Device description

The SIMATIC HMI Basic Panels product line features key and touch panels (operator input via keyboard and touch screen)
SIMATIC HMI Basic Panels cover all requirements described in the previous section.

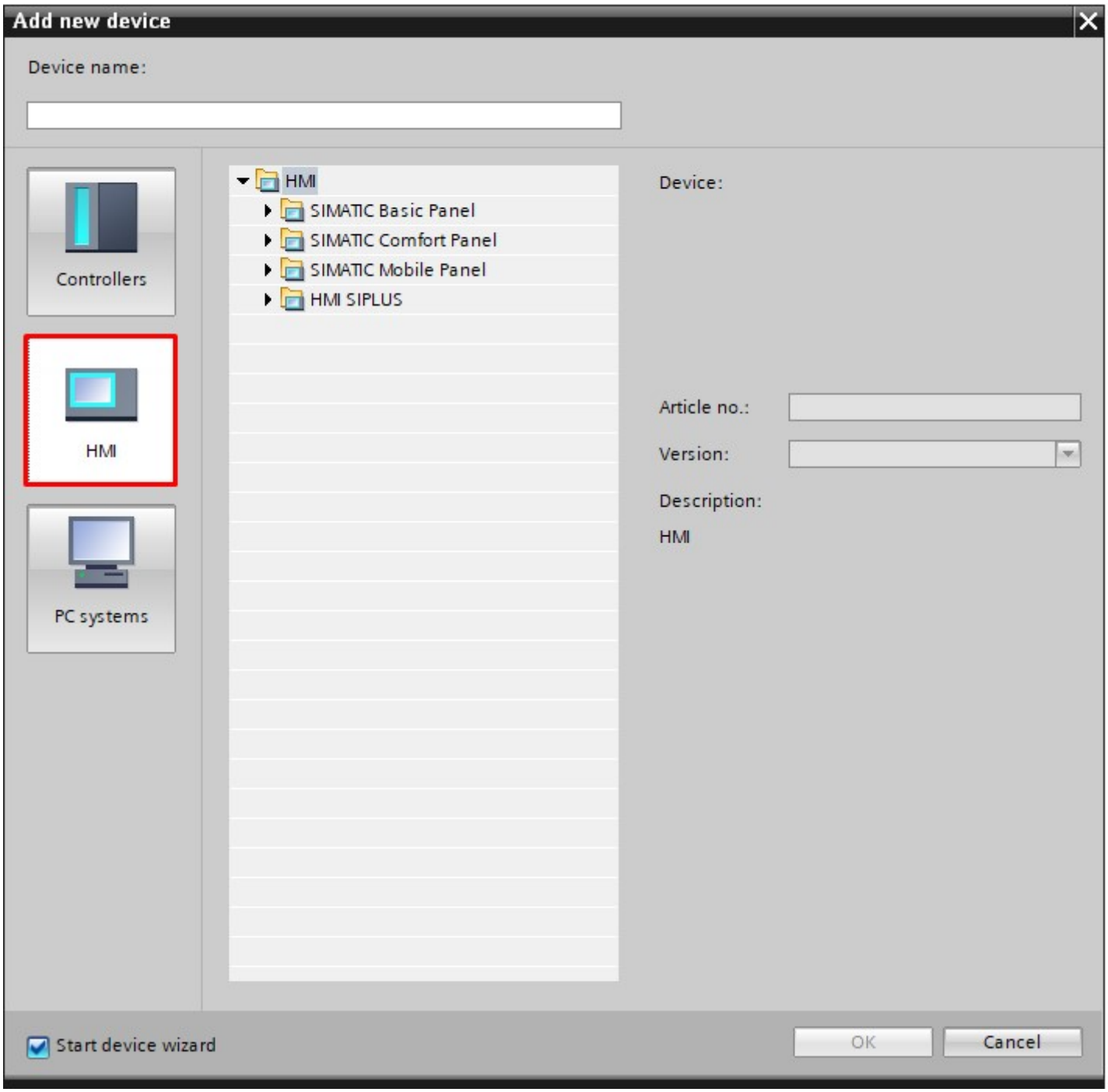
Hardware configuration

For a HMI to be correctly used in TIA Portal V16, one will need to make the right hardware configuration. For this to be correct you will need a correct CPU configuration mentioned in M2C3-ADD03.

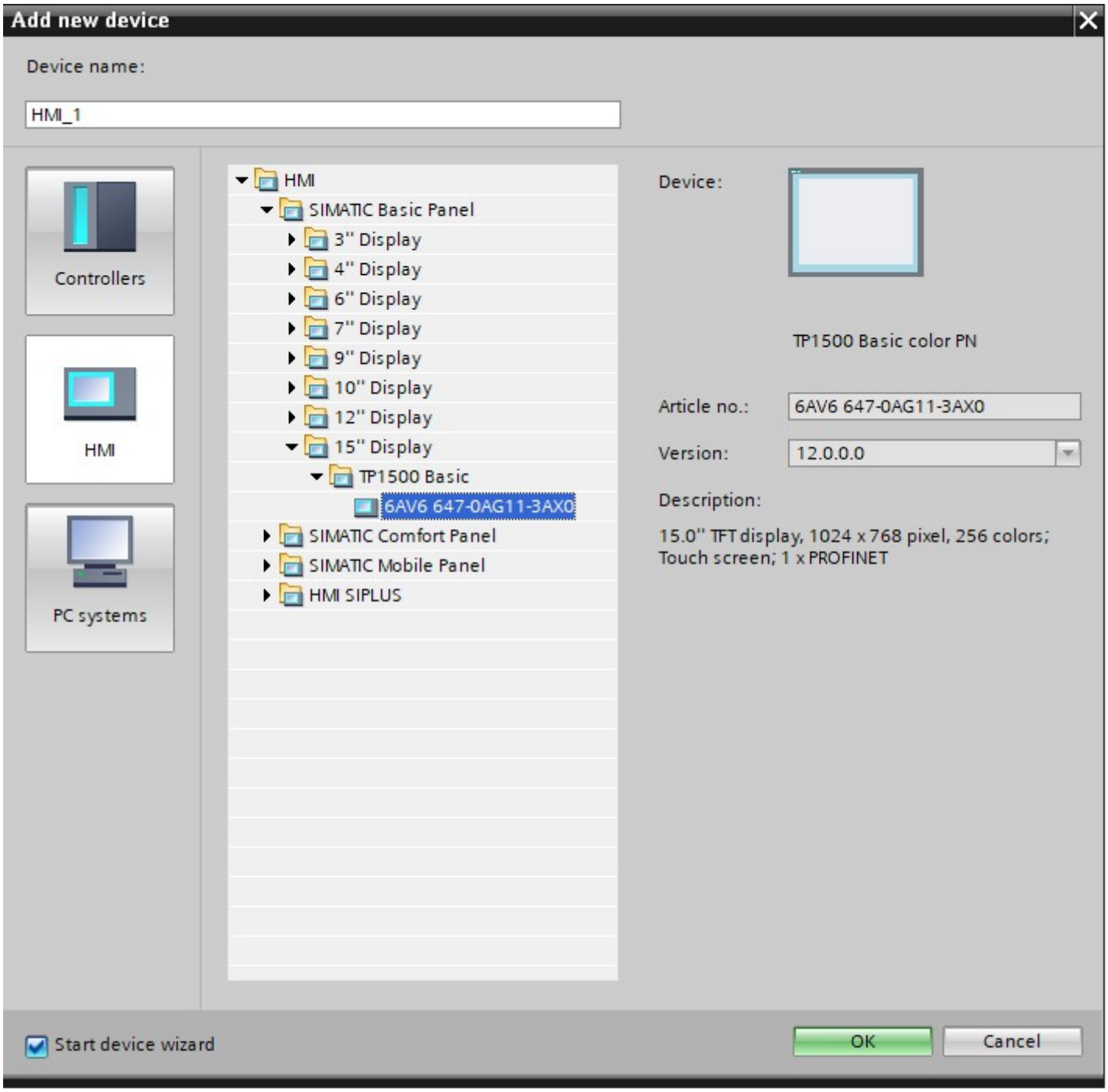
To add a HMI to a current project you will have to add a new device. This can be done through the 2 views within TIA portal.

Project View	Portal View
"New Device"	"Devices & network > Add new device"

The menu that pops up has 3 main options to select between Controllers / HMI / PC systems. For our example we'll be needing the HMI tab:



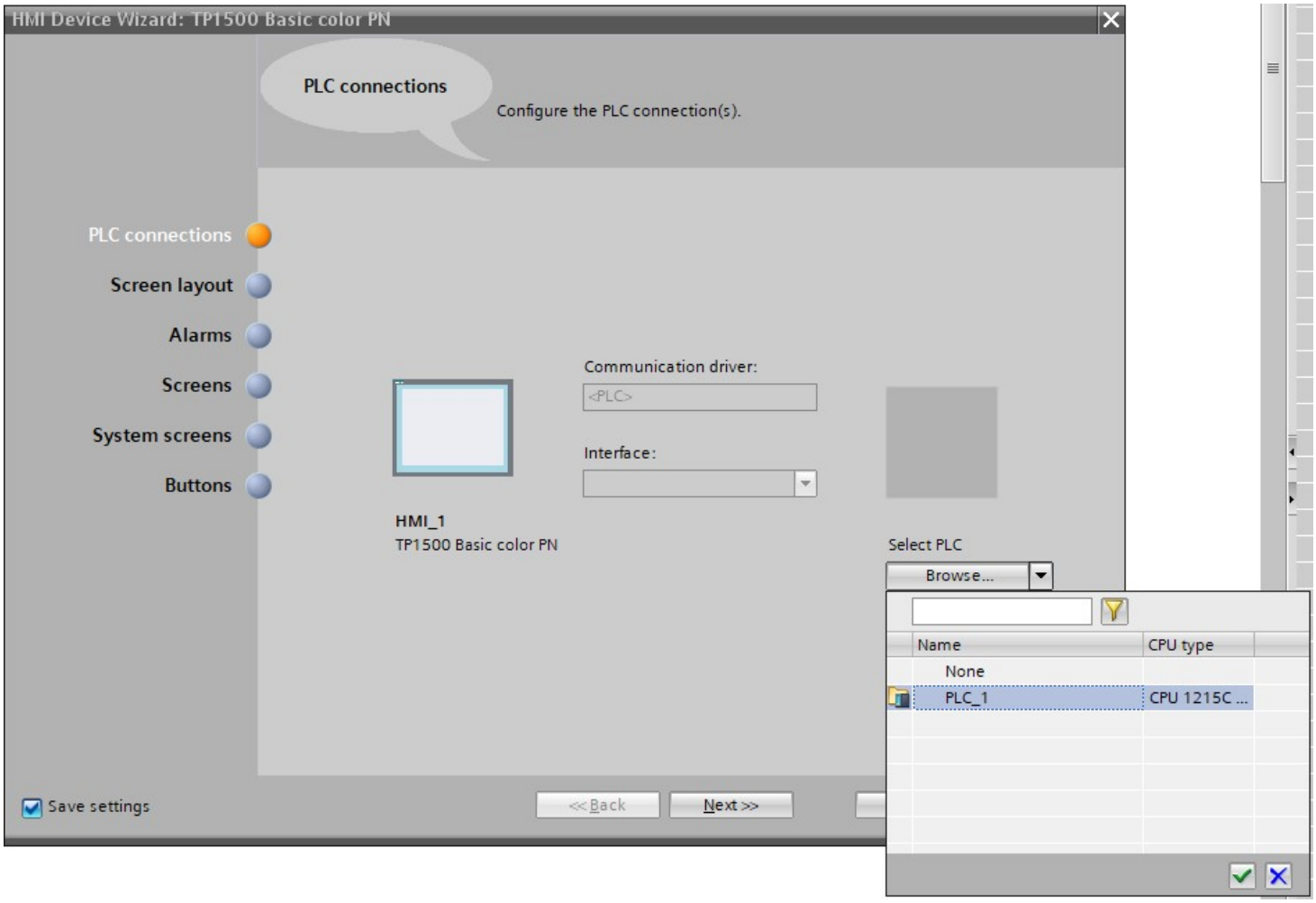
After which you select the desired HMI. HMI > SIMATIC Basic Panel (we'll be using the simulator of TIA Portal, for this example i took the KTP1500 Basic):



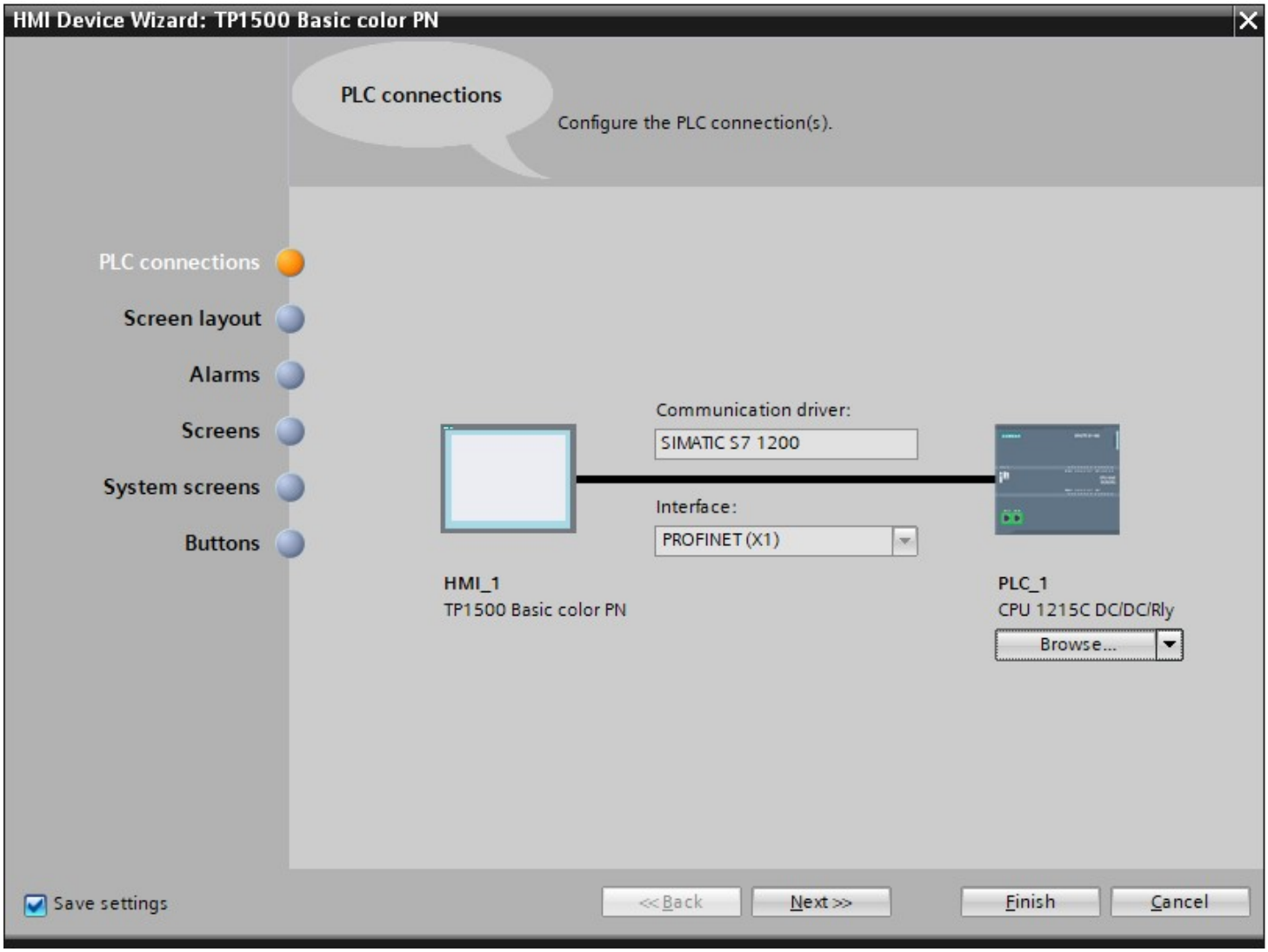
HMI wizard

When you selected the correct/desired HMI you get the following screen presented:

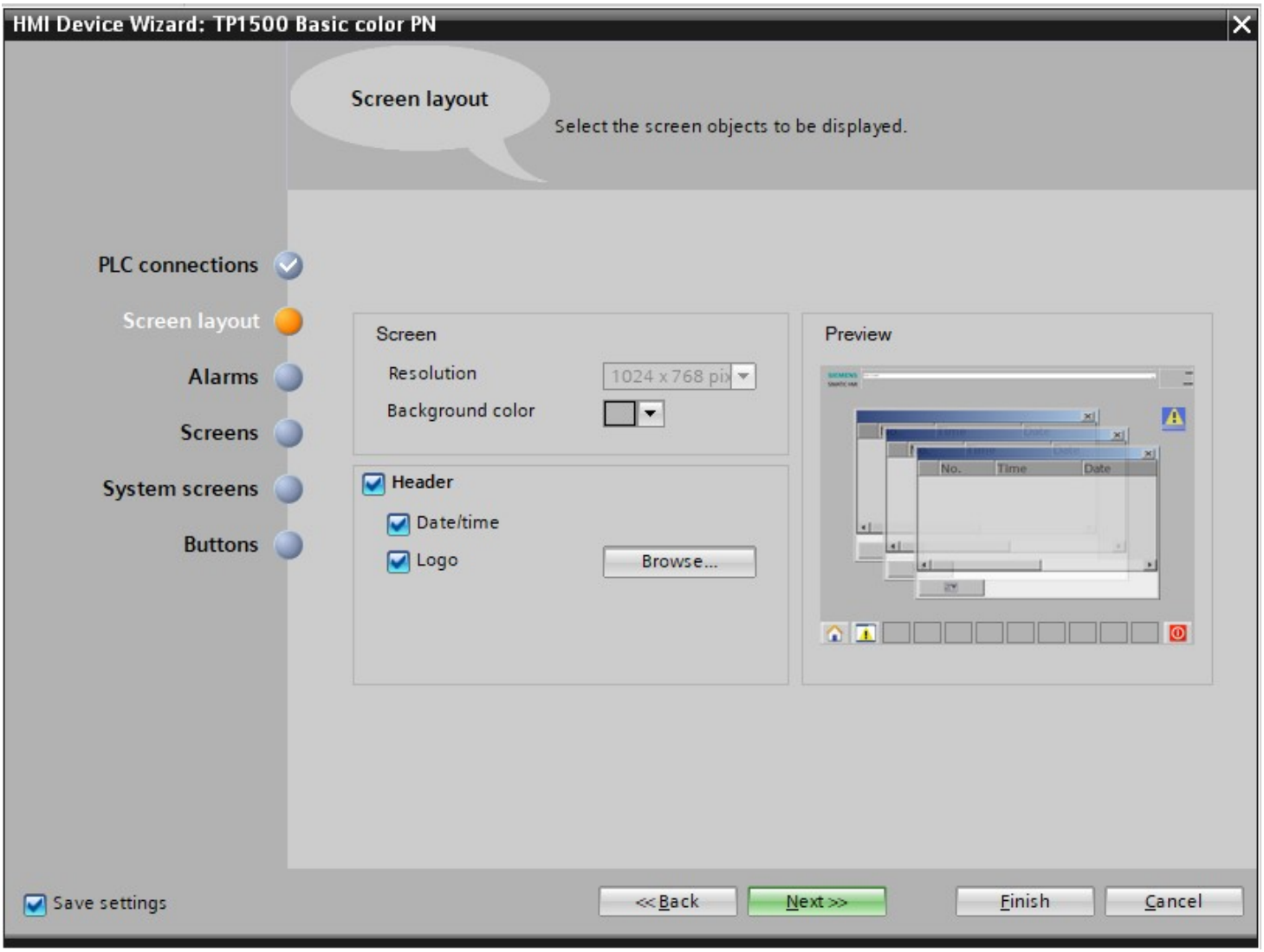
Here you need to link the HMI to the correct PLC, this is done by selecting the right PLC under Select PLC -> PLC_1



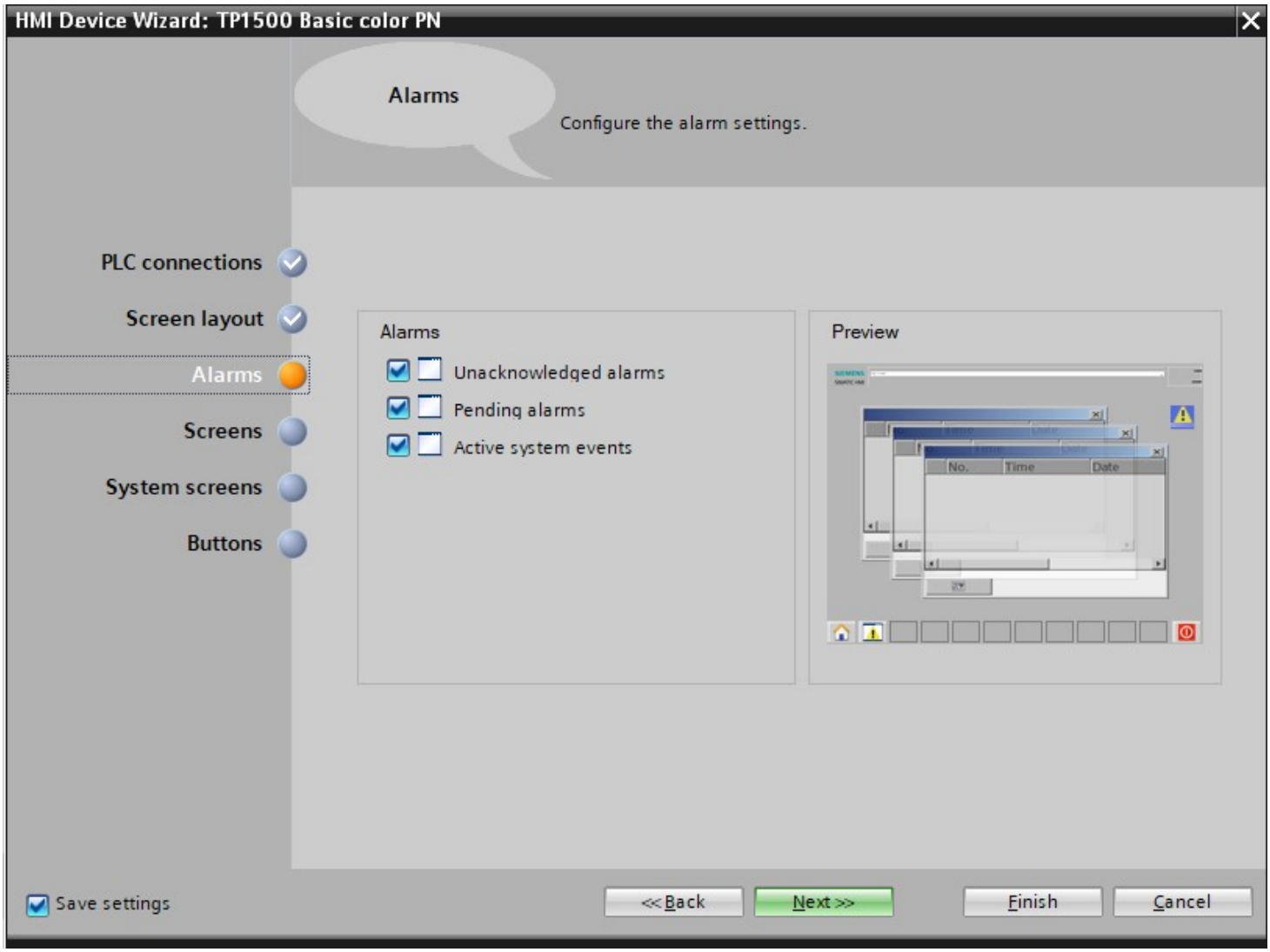
If the correct PLC is selected it should automatically link the HMI to the PLC through ProfiNET. Now proceed to the next screen by pressing "Next".



You can change the default background color of your panel under "Screen layout".
Select the "Header", "Date/time" and "Logo" check boxes. Confirm your selection by clicking on "Next ".



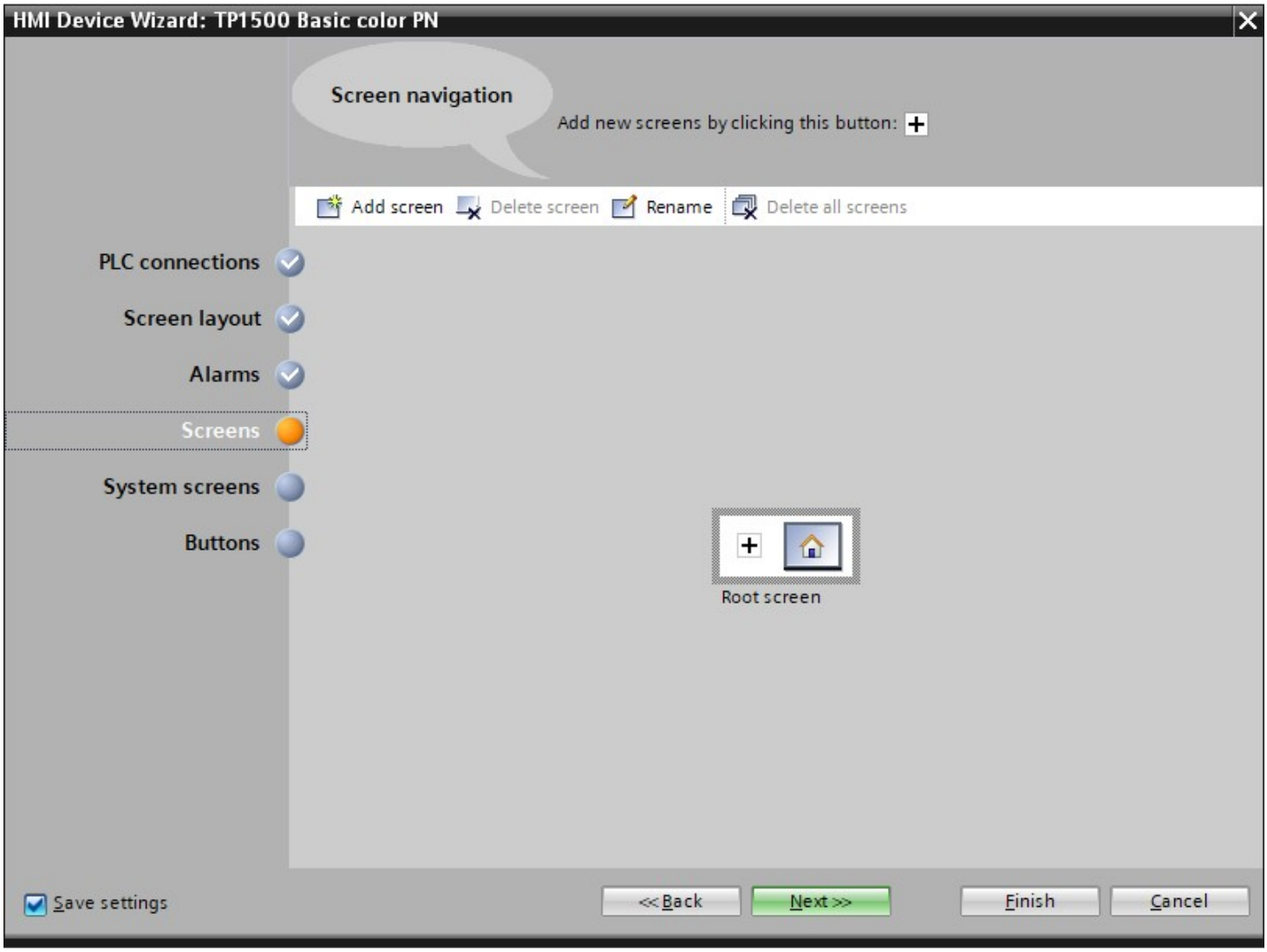
In the "Alarms" section, you can specify which of the alarms are to be displayed in a window. Select all three alarm types. Confirm your selection by clicking on
"Next".



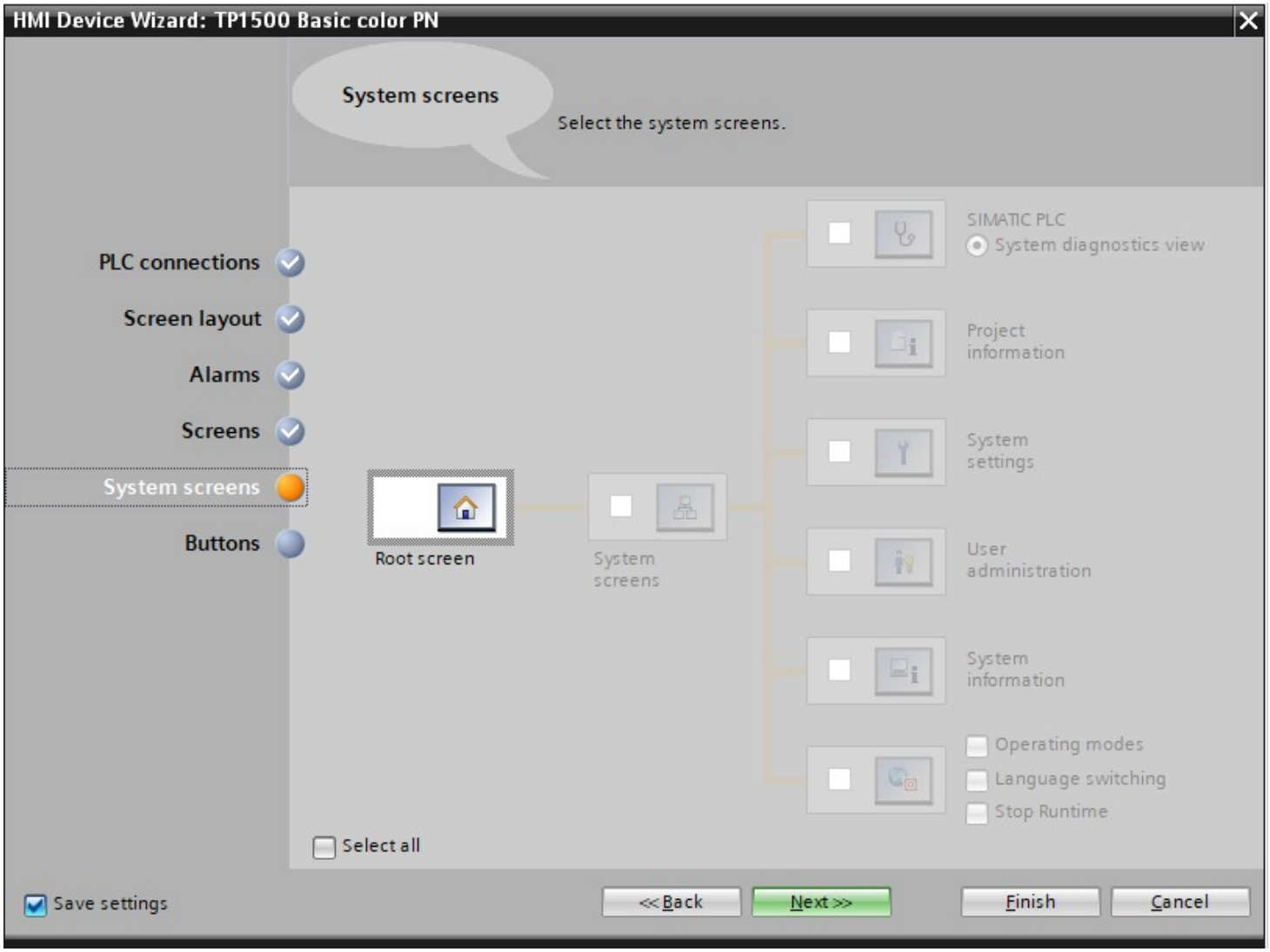
In the "Screen navigation" section, the screen structure is displayed with the screen name of the last created project, starting with the root screen on the far left. A new name can be assigned simply by clicking on a screen name.

If you click on + you can insert new screens in the hierarchy ® and delete selected screens by clicking on "Delete screen".

Confirm your selection by clicking on "Next".

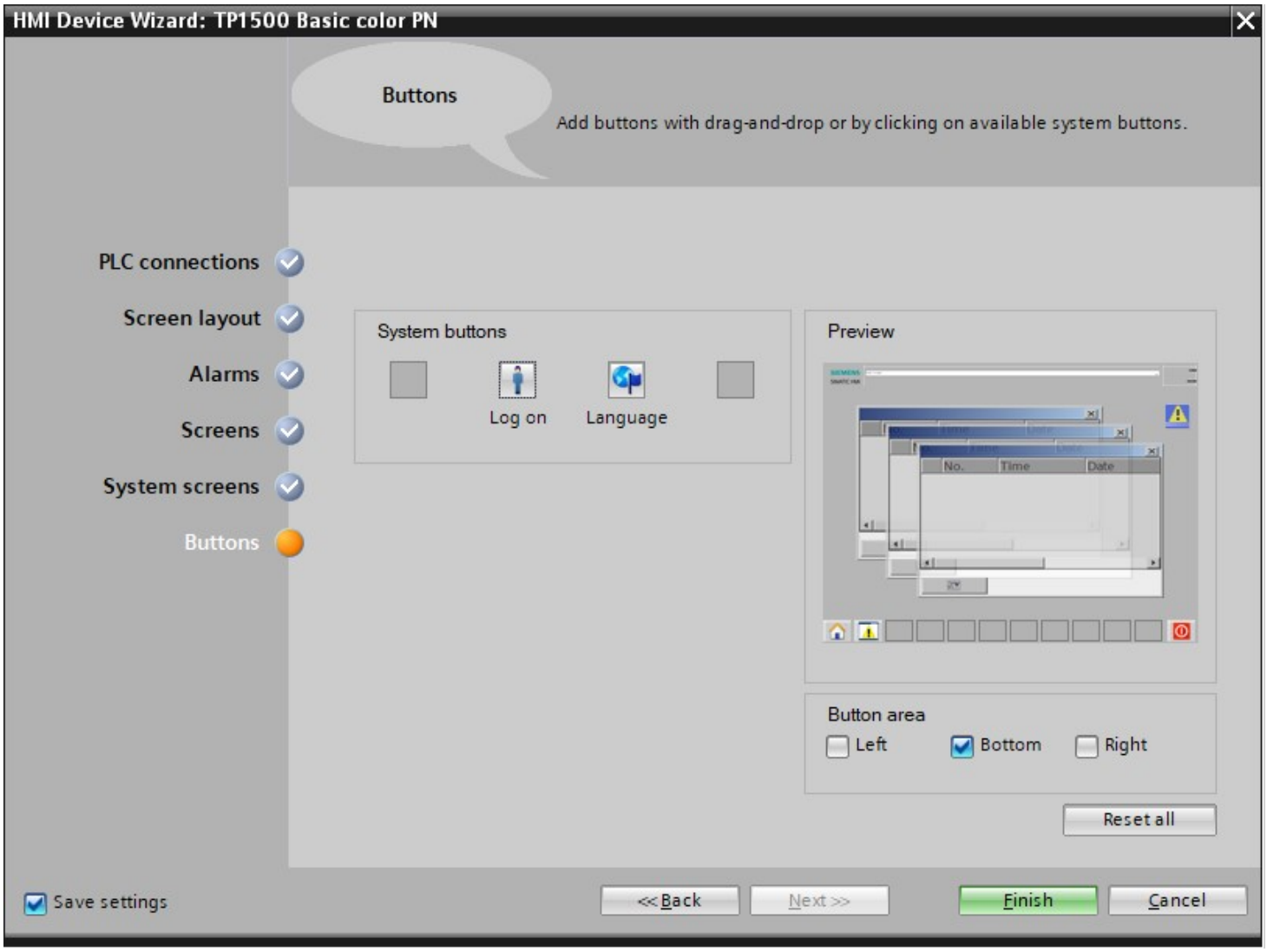


In the System screens section, you can select previously preset views for system functions and have them automatically added. Select all system screens by clicking "Select all". Confirm your selection by clicking on "Next".



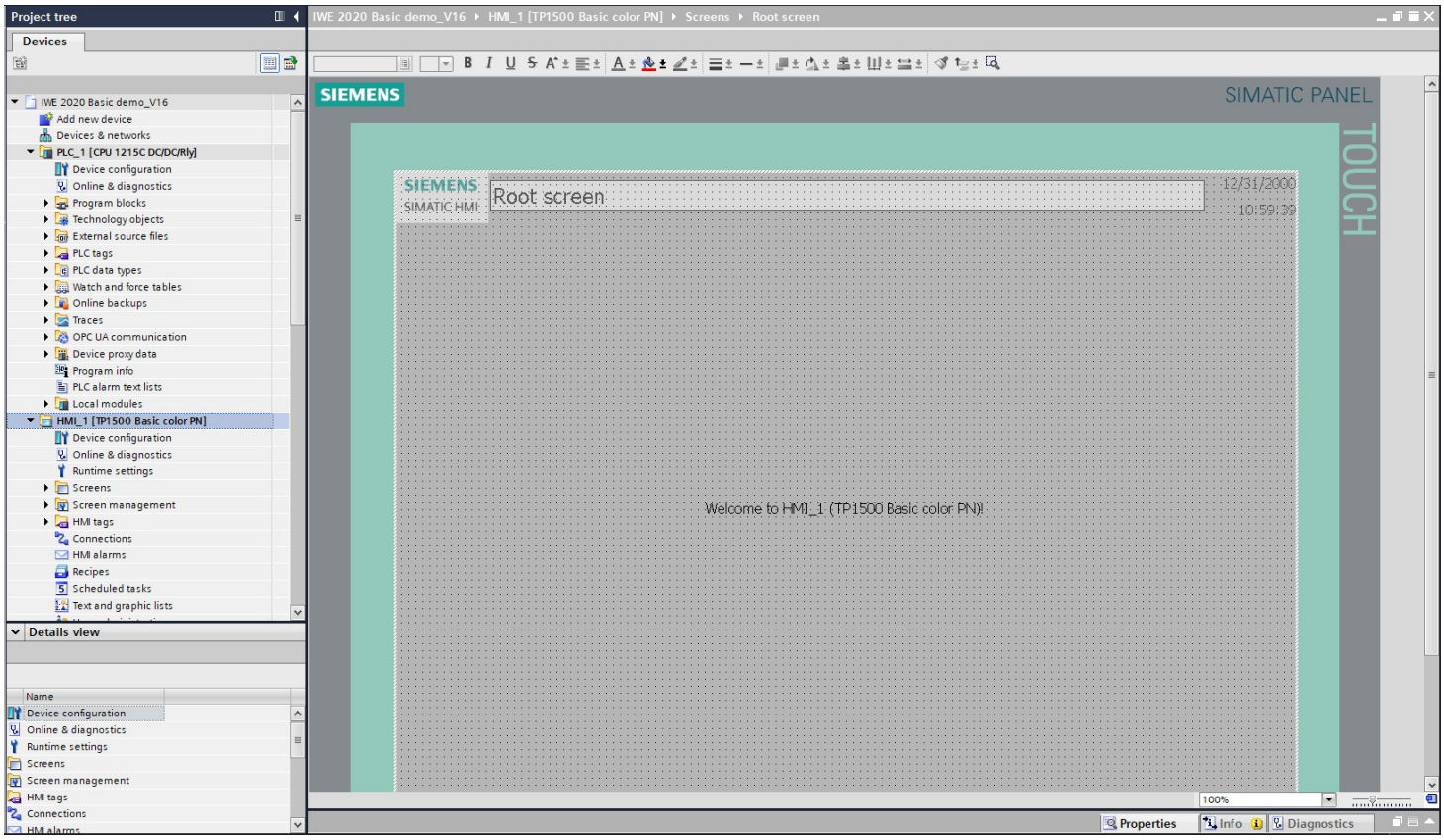
In the System buttons section, you will find four user-selectable buttons for Exit(Runtime), Log on, Language and Root screen. You can place these buttons on the provided button areas "Left", "Bottom" or "Right" as desired. An "Open alarm window" button is already created.

Select only the "Button area", "Bottom".Insert the button for the "Root screen" on the left and the button for "Exit" Runtime on the right. Confirm your selection by clicking on "Finish".

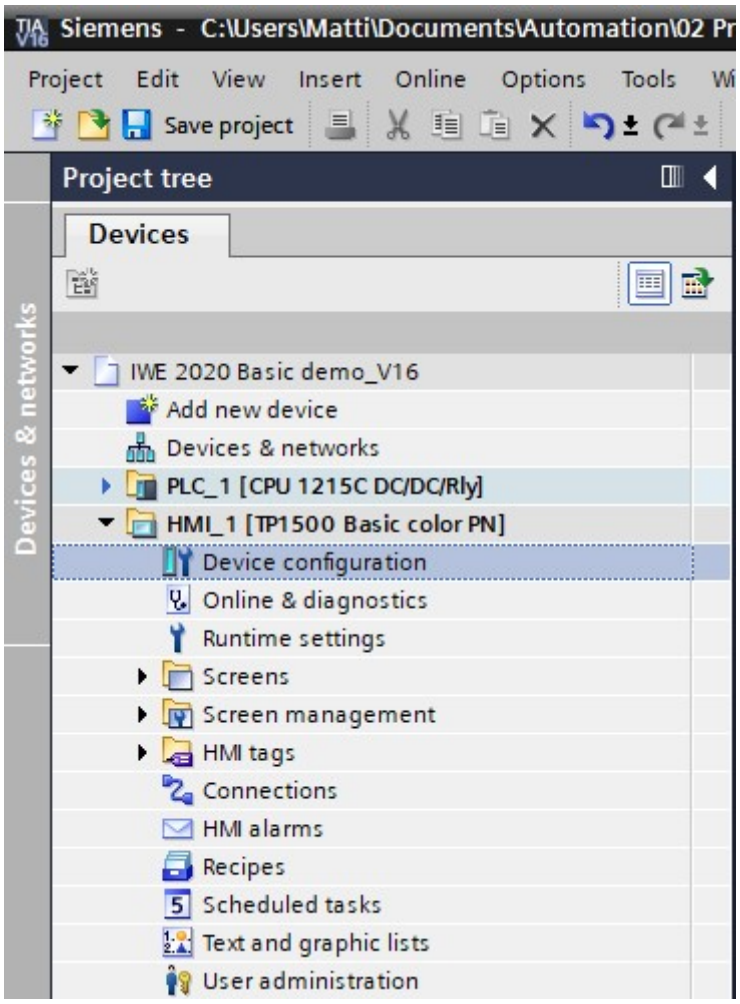


Device configuration of HMI Panel

The TIA Portal now automatically changes to the Project view and displays the root screen of the visualization.

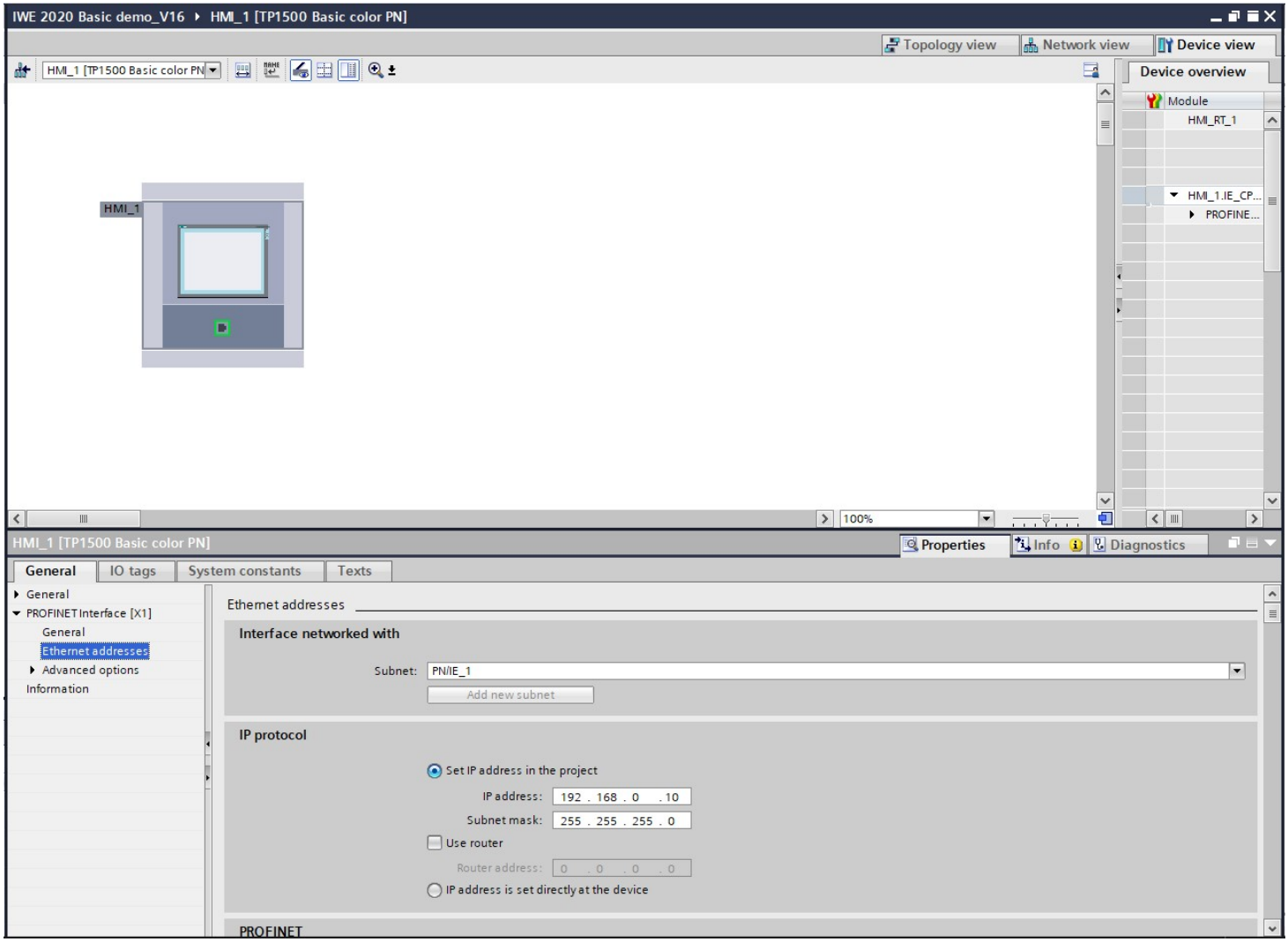


To configure the panel, select "Panel KTP700 Basic" in the project tree and open its "Device configuration" with a double-click.



Setting the IP adress

Select the Ethernet interface of the panel in the Device view with a double-click.
Under "General" in "Properties", open menu item "PROFINET interface [X1]" and select in the "Ethernet addresses" entry.
Set the IP address "192.168.0.10" under IP protocol.



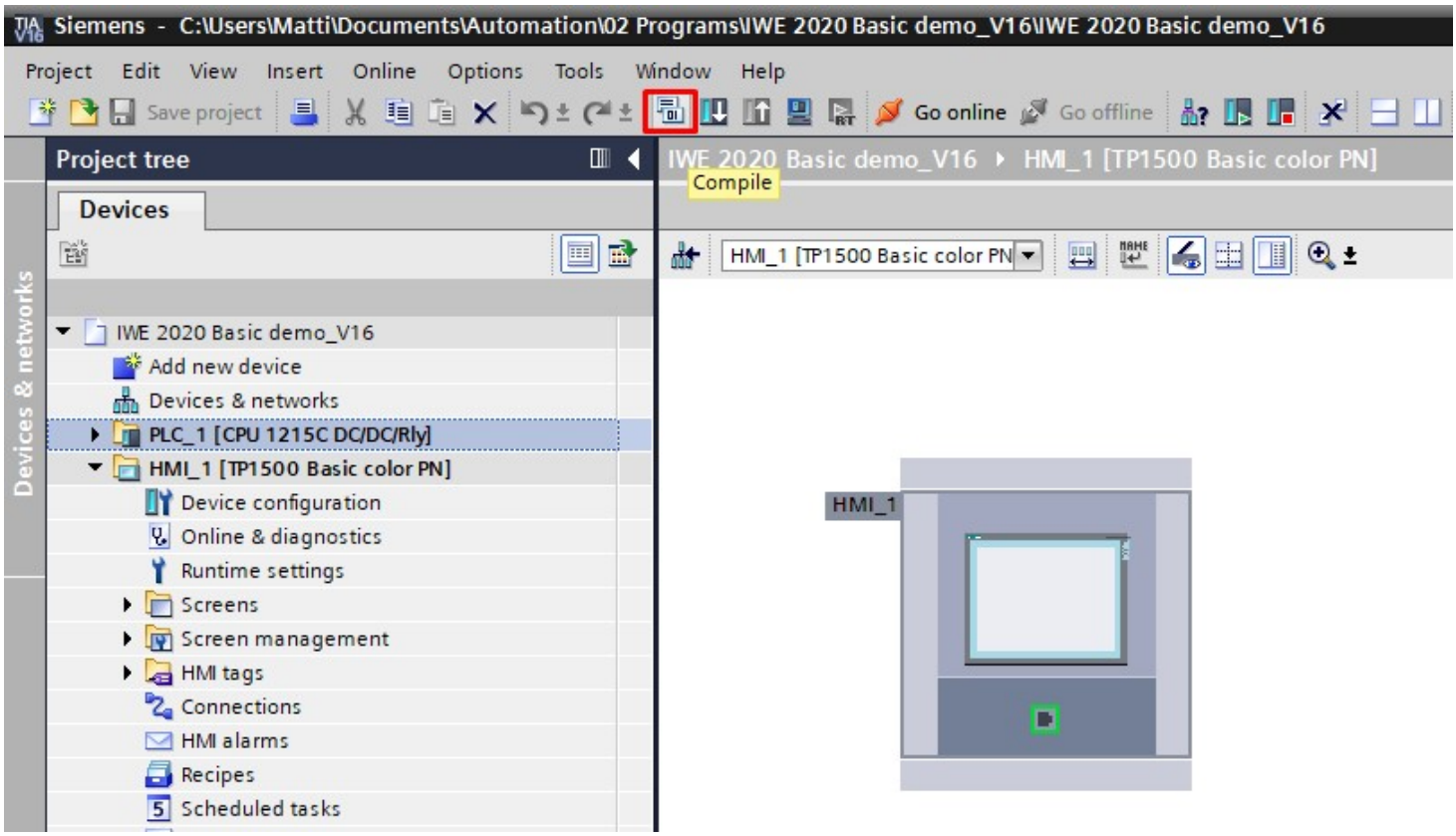
Remarks

– The subnet mask was already set in the settings of the CPU 1215C and is automatically applied by the panel.

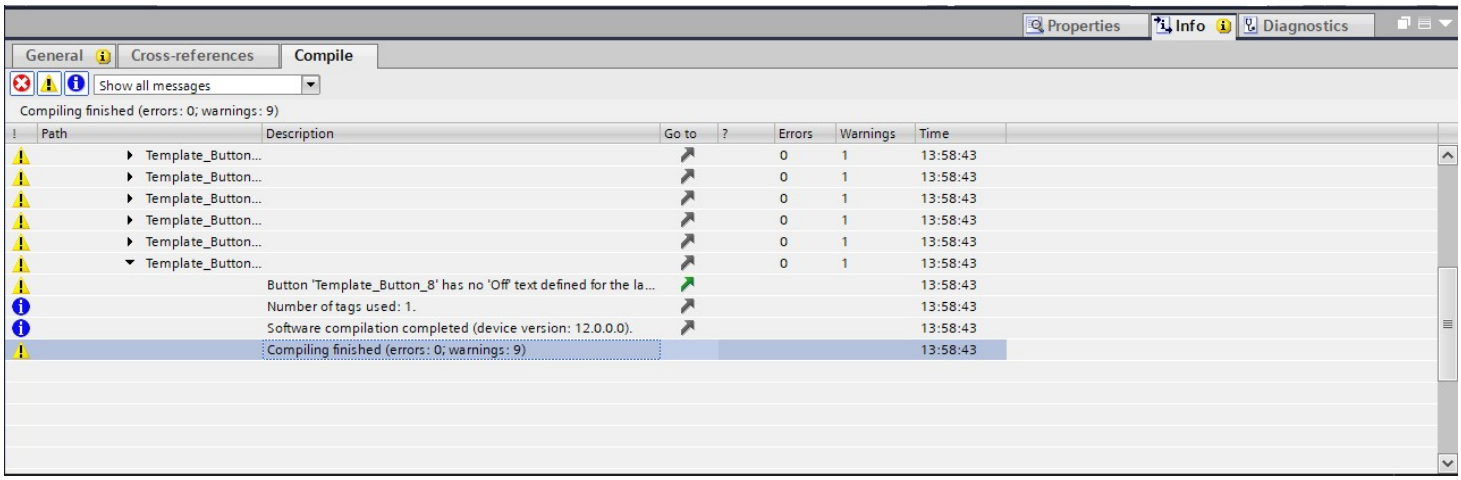
Compiling the CPU and panel and saving the project

To compile the CPU, click on the "CPU_1215C" folder, and select the "Compile" button for compiling in the menu. To compile the panel, click on the "Panel KTP1500 Basic" folder, and select the "Compile" button for compiling in the menu. You can save your project by clicking on the "Save Project" button in the menu.

(CPU_1215C > "Compile" > Panel KTP700 Basic > "Compile" > Save project).



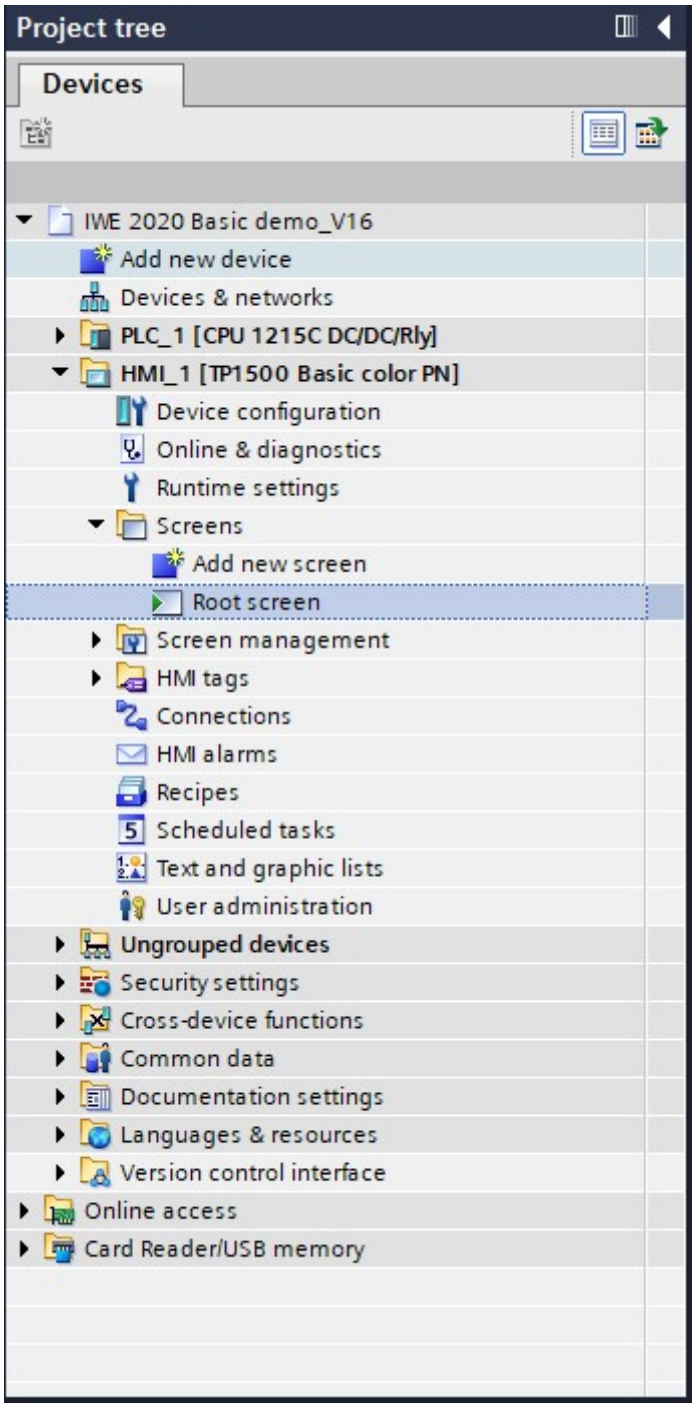
In the "Info" area under "Compile", it is then shown whether the compilation was successful or whether warnings or errors have occurred.



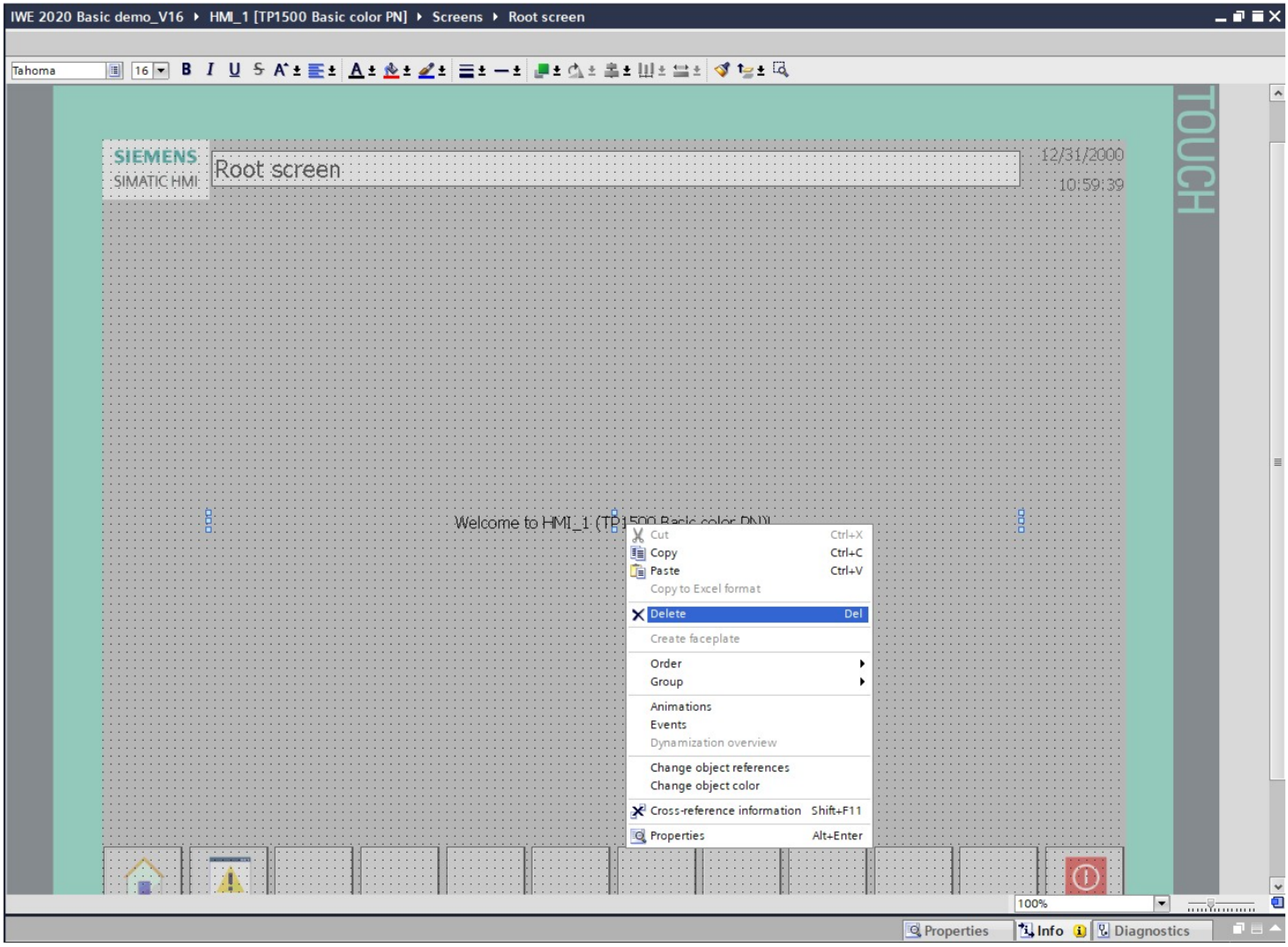
Changin screens and objects

Screens

After successful compilation, you want to design the first screen for the visualization. To do this, open the > "Root" screen with a double-click:

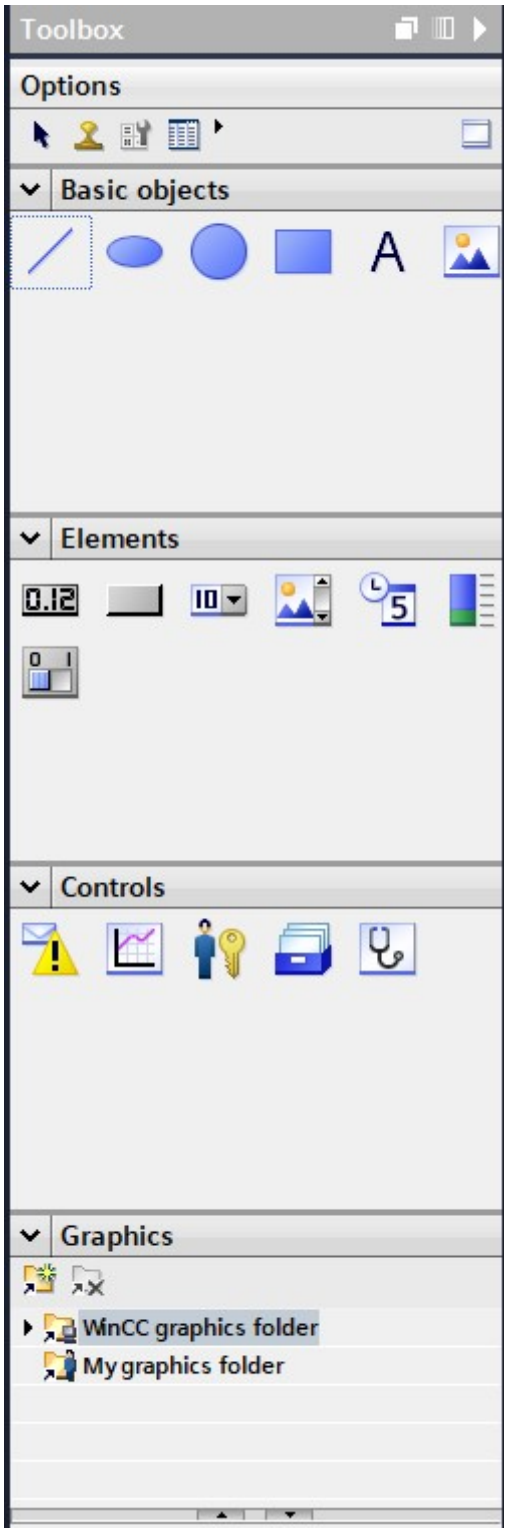


You will see a text box in the center of the screen, we will remove this by right clicking the text box and selecting "Delete". This also can be done by pressing you keyboard button "Delete".



Toolbox

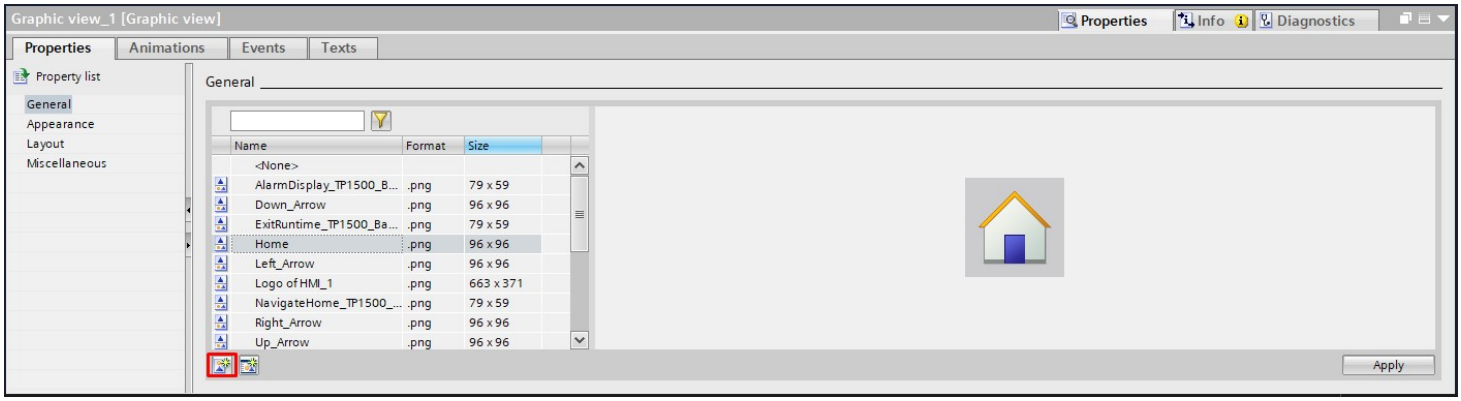
These contain the most commonly used objects, elements and controls used in a HMI display.



Basic objects

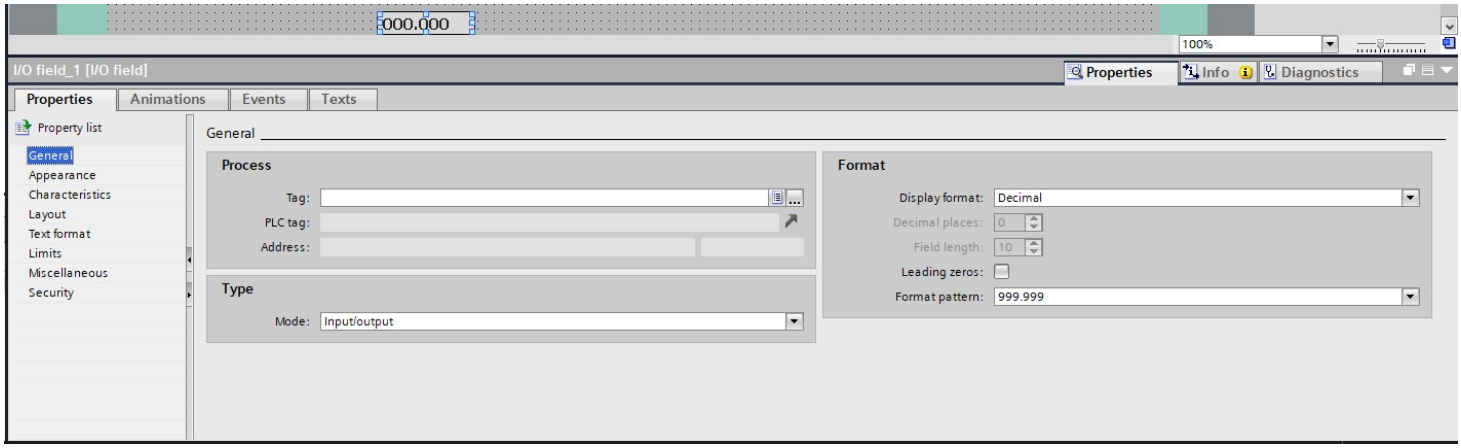
The basic objects contain a text box, rectangle, circle, line, ellipse and a graphic view.
These are added by simply dragging them onto on the screen or selecting one in the toolbox section and clicking on the desired screen. To customise them further you'll have to click on the properties of the object. For example a rectangle, in the properties you can change the appearance, layout and miscelaneous.

These properties are the same for all basic objects except for the text box and graphic view. Graphic view has the ability to show custom images. This is done by selecting the grpahic view and placing it on a screen. Then in options you'll have to go to General > bottom button to add custom files. There is already a pre existing list of images you can choose from. But to add a custom image:

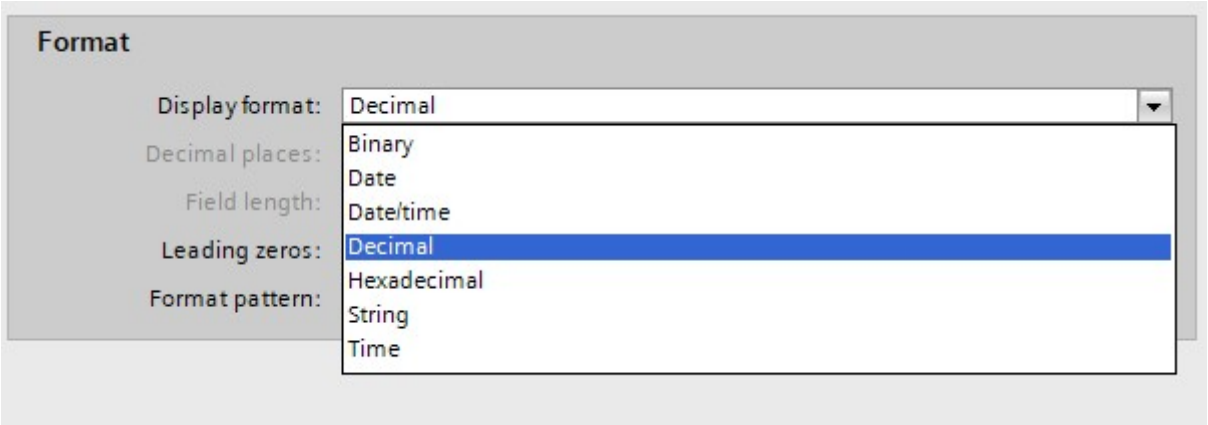


Elements

The elements are objects that can be linked to actual PLC data / are ment to be used as a way of interacting with PLC data. These stock ones contain an I/O-Field, button, symbolic I/O-Field, graphical I/O Field, date/time field, bar and a switch. Once dragged into the screen you can change the appearance however you want and link them with the correct PLC data type. For example an I/O-Field.



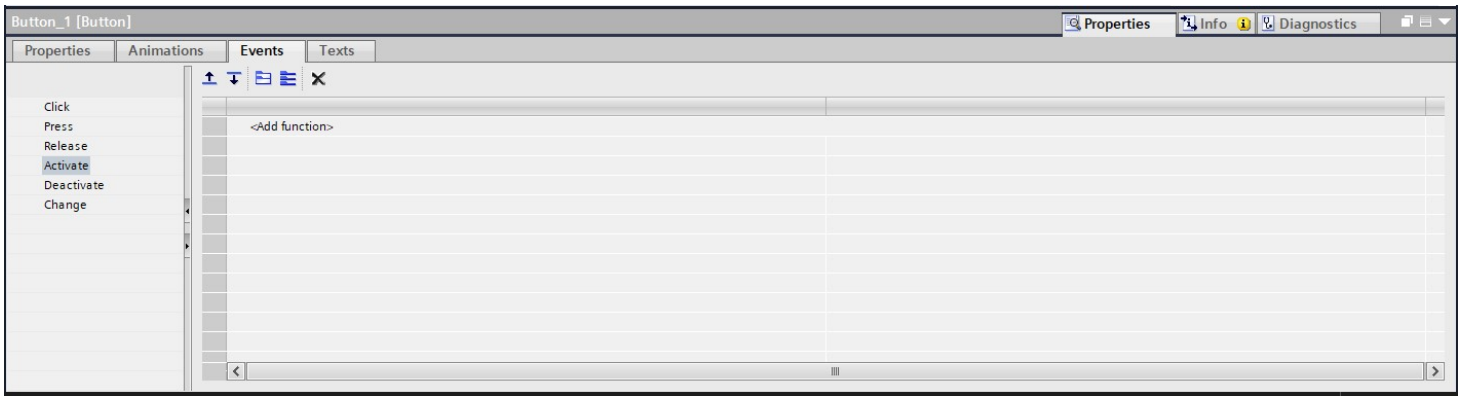
In the properties of this I/O field you can assign a Tag from the HMI or PLC to it. This will read out the value that tag has. An I/O field is mostly used for reading or giving in a desired value. In "format" you can customise how the value gets displayed. For example:



You can also change the type of I/O field by changing it to input or output only. Input only will only show input values. Output only will only allow a measured value to be displayed and NOT changed.

Events

Each object or element can have events. These can change values, screens on the HMI, calculations scripts etc.. For an example we'll use a button to make it behave like a Start button.



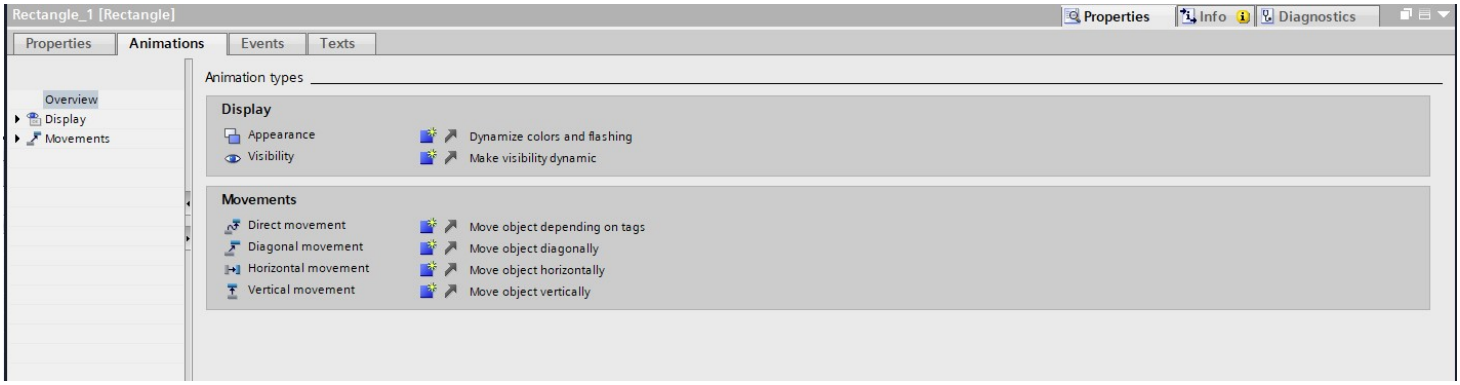
To assign a function to the button you click on "Properties" > "Events" > "Click" > "< Add function >" you will get the following selection.

Select "Edit bits", in here there are several options of changing a bit. For a start button the most common function is "SetBitWhileKeyPressed", this will set the asigned bit to 1 while you press the button. If you release the button the bit value will be set to 0 again.

Animations

Display

Within the properties screen you can find the tab "Animations". After which you get the option to add either a **Display** or **Movements**. In this chapter i'll explain the **Display**.

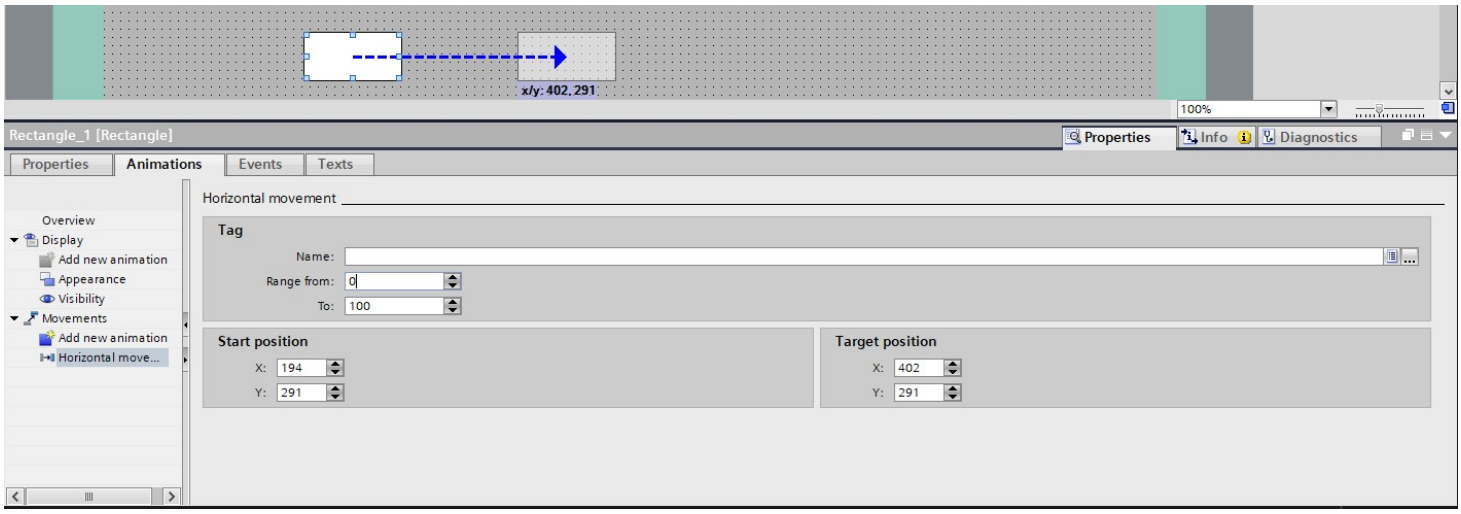


Within a display you can have a "Appearance" or "Visibility" animation. The appearance will give you the ability to change the colours of a object depending on a tag value. Visibility will give you the option to hide the object depending on a tag value.

Movements

Within **movements** there are 4 different movements, direct movement, diagonal movement, horizontal movement and vertical movement. Direct movement will allow you to move a object from point A to B in a direct way. Diagonal movement will move from point A to B diagonally. Horizontal movement will move from point A to B horizontaly. Vertical movement will move from point A to B vertically.

Horizontal and vertical movement can be assigned to a tag value so that depending on that value the object will move towards one point. For example:



The range will determine how much of your tag value will be used (0-50 of the tag value will move your object between the two points in that range 0 being point A and 50 being point B).

GRAFCET

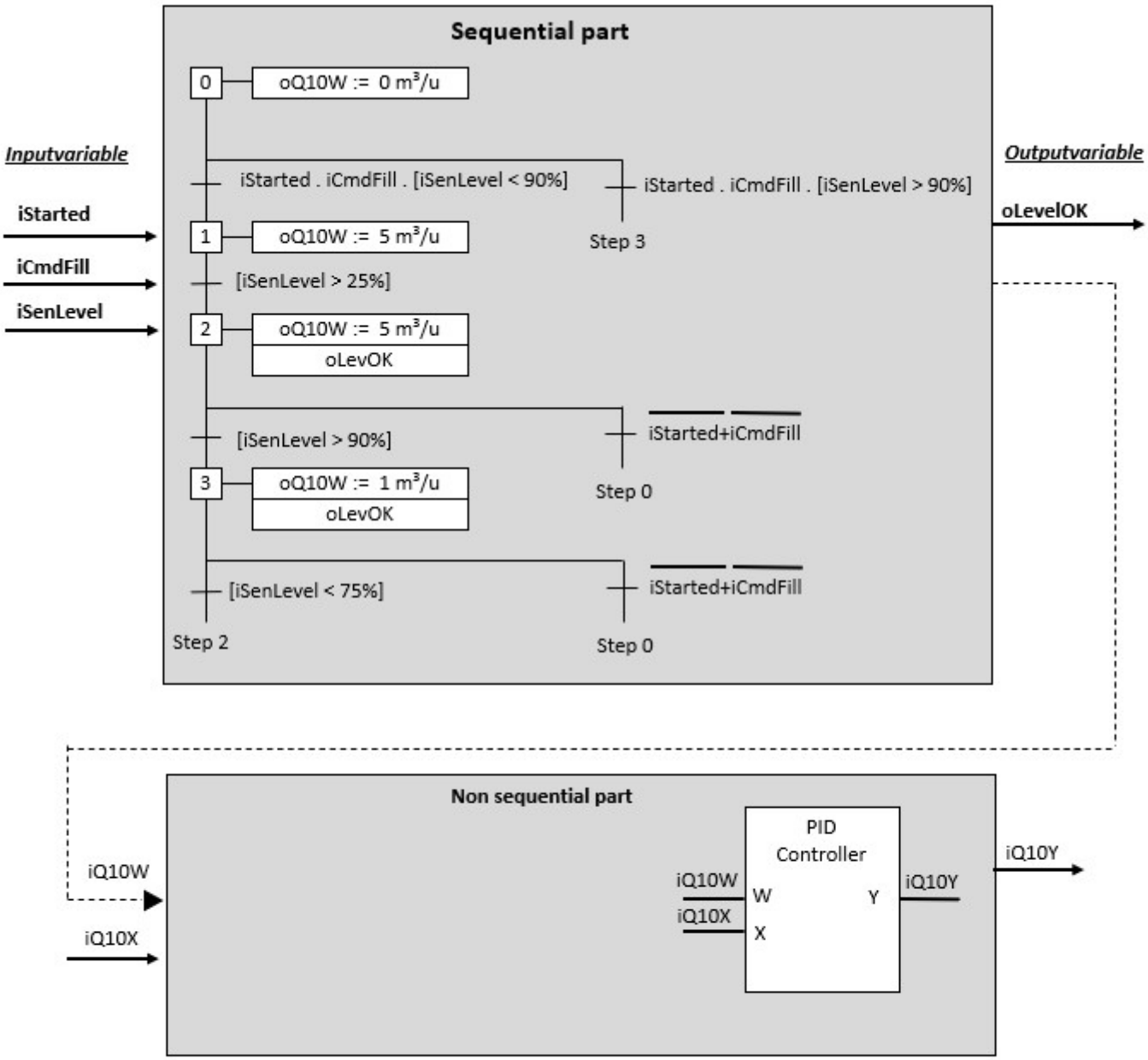
General

The implementation of an automated system requires, in particular, a description relating cause and effect. To do this, the logical aspect of the desired behaviour of the system will be described.

The **sequential part** of the system is the logical aspect of this physical system. The behavior indicates the way which the output variables depend on the input variables. The object of the GRAFCET chart is to specify the behavior of the sequential part of the system.

The **GRAFCET design language** is characterized by different graphical elements and by text that gives information about the variables. By connecting these various elements and text, the behavior of the automatic machine/installation is described.

This behavior is known as steps and a GRAFCET contains multiple steps. The evolution of one step to another is translated by one or several transitions.



iStarted - Automatic process is started
iCmdFill - Command to fill the tank
iSenLevel - Tank level sensor in %

oLevelOK - Level OK for the next tank
oQ10W - Desired flow rate
oQ10X - Measured flow rate
oQ10Y - Controloutput

A GRAFCET will be executed as follows:

- A GRAFCET will run from top to bottom
- A GRAFCET starts with an initial step
- A transition is displayed as a mathematic boolean expression
- The result of a transition is TRUE or FALSE
- While the GRAFCET is executed there is at least one active step
- Only steps connected to the active step can be executed
- Other steps can be activated on the condition that they are connected with the active step if the result of the connected transition is TRUE

There are 5 programming languages included in the standard IEC 61131 including SFC[1] which is inspired on the GRAFCET design language. However, there are some differences:

- SFC is a programming language
- GRAFCET is a design language
- The SFC program language uses other program languages, such as FBD and LAD, and different abbreviations to program transitions and actions
- The execution of an OR-convergence, if all conditions are TRUE, is different

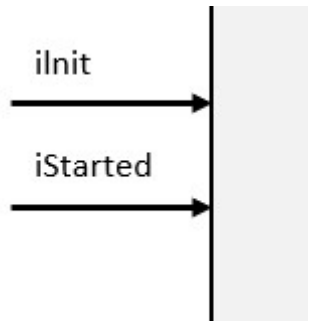
Conclusions

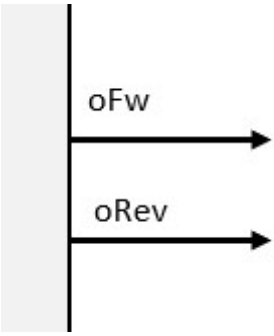
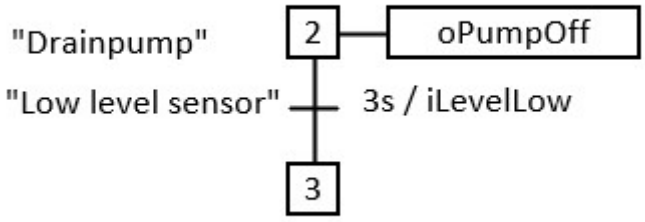
- It is possible to program a GRAFCET in each programming language described in the standard IEC 61131
- The SFC program language looks like GRAFCET design language but is not 100% the same

Designing of a GRAFCET in IEC 60848

8.2 Addendum 4 GRAFCET

GRAFCET diagram

Symbol	Description
	A GRAFCET diagram is a collection of steps, actions, transitions, connections, etc. which form a complete diagram. The collection of all elements is surrounded by a rectangle.
	Input variables are on the left with an incoming arrow **Example**: "Initial step activation" and "installation started" inputs 

Symbol	Description
	Output variables are located on the right with an outgoing arrow. **Example**: "Forward" and "Backward" output signals. 
"*"	A **comment** clarifies the working of certain parts and is written between double quotation marks, whereby the asterisk symbol gets replaced by the description. **Example**: Stop a drain pump if the level is too low. 

Step

A **step** displays a defined condition of the sequential process. A step is either **Active** or **Not Active**.

On a certain moment during the sequential proces:

- Is a step active or not active
- The set of active steps determines the state of the process
- The GRAFCET determines which step or steps can become active

Symbol	Descpription

Symbol	Descpription
	A **step** is shown as a square with an unique label. For practical reasons the most commonly used label are numbers which replaces the asterisk symbol. **Example**: Step 2 
	The **initial step** characterizes the initial situation and is displayed as a double square. In case of the initial step being active, all other steps in the GRAFCET won't be active. **Example**: Step 0 
	An **enclosed step** means that this step contains other steps. If the conditions after the enclosed step are TRUE, we will proceed to the next step and all internal steps will be inactive. It is allowed that an enclosed step contains multiple GRAFCET diagrams, but the enclosed internal steps can only be assigned to one enclosed step.
	An **enclosed initial step** means that this step has multiple internal steps that participate in the initial condition. The enclosed initial step contains minimum one internal initial step and can contain multiple GRAFCET diagrams.

Symbol	Descpription
	A **macro step** means that this step has multiple internal steps. We can describe it as an independent piece of software. A macro is not designed as standalone software, it’s meant to support a different piece of software. The internal steps always start with a source step and always end with an end step. Only in the case of the end step being active can the macro exit. Unlike an enclosed step, a macro contains a maximum of one GRAFCET diagram and the asterisk symbol gets replaced by one unique label that can deviate from step labels, in numbered order.
	In case that an **active step** needs to be displayed, this will be done by placing a point under the label.

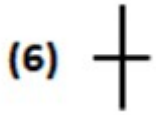
Connections

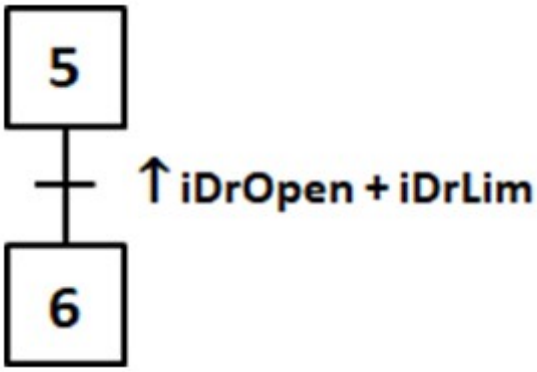
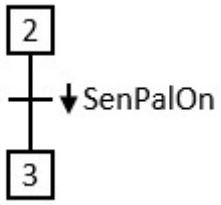
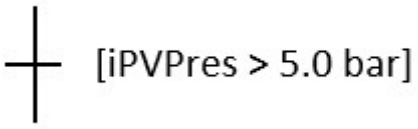
Symbol	Description
	Connections are lines in the network that connect steps.
	Both horizontal and vertical lines are allowed. Diagonal lines are to be avoided. They are allowed, but only to clarify.
	The flow of a connection is always from top to bottom. The use of arrows is allowed in case of clarification.

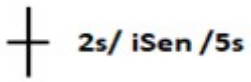
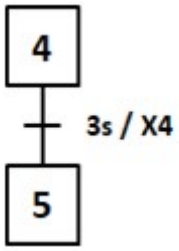
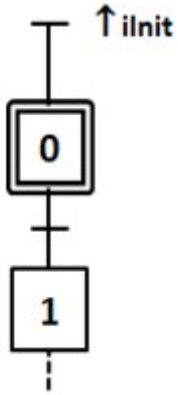
Symbol	Description
	If a directed link has to be broken (for example for complex charts or when a chart covers several pages) the number of the destination steps and the number of the page on which it appears, shall be indicated. **Example**: Reference to step 12 on page 2 

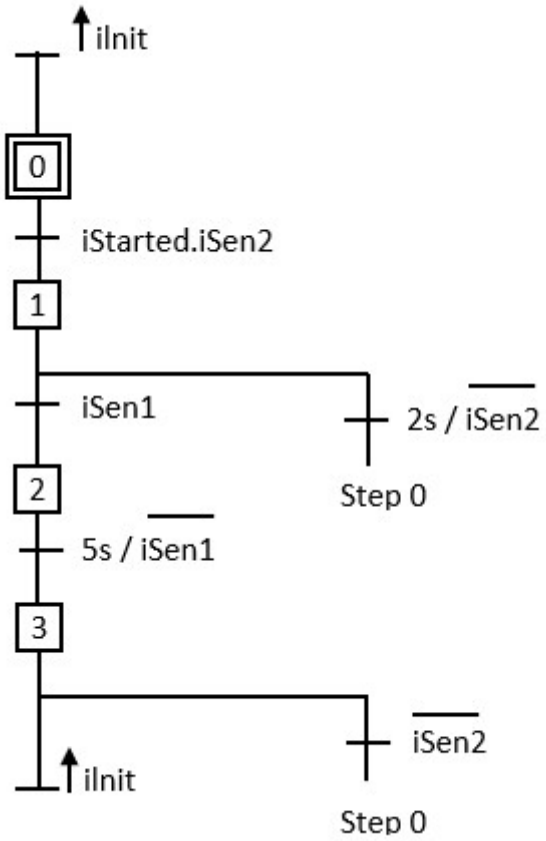
Transition

Symbol	Description
	A transition between two steps is indicated by a horizontal line right through the connection line. The transition-condition is active if the previous step is active. Between 2 steps one condition is allowed.
	It’s allowed to use vertical transitions for graphical reasons.

Symbol	Description
	Each transition contains a condition. This is a mathematical boolean expression composed by variables, they replace the asterisk symbol. The result of a transition-condition is TRUE or FALSE. The transition-condition is always at the right of the transition. **Example**: Start button AND stop button 
	The transition may have an unique designation ( ) at the left of the transition. **Example**: Label 6 
	A transition condition that is always TRUE is displayed with the underscored expression 1.
	The status of a step (active or not active) can be added in a transition-condition with the capital letter "X". The asterisk symbol will be replaced by a label of the step. **Example**: Step variable of step 7

Symbol	Description
	An upward arrow in the transition-condition means that it is only TRUE the moment the variable changes from FALSE to TRUE. **Example**: The transition-condition is TRUE on a rising edge of the "Door is Open" sensor OR if the "Door Limit switch" is activated. 
	A downward arrow in the transition-condition means that it is only TRUE the moment the variable changes from TRUE to FALSE. **Example**: The transition condition is TRUE on a decreasing flank of the pallet fotocel. 
	A **comparison** is noted between [ ]. The asterisk symbol gets replaced with a comparison. The result of a comparison instruction is TRUE or FALSE. **Example**: The transition-condition is TRUE in case the actual pressure is higher than 5,0 bar. 

Symbol	Description
	A variable that is time dependent is displayed with the **/** symbols (TON / variable / TOF). Hereby, the transition-condition is TRUE after an on-delay and stays TRUE with an off-delay. It is allowed to simplify the notation by removing the off-delay in case this isn't used. **Example**: The transition condition is TRUE 2 s after the iSen is TRUE and stays 5s TRUE after iSen becomes FALSE.  **Example**: 3 s after step 4 is activated the transition-condition becomes TRUE and step 5 will be activated. 
	A **source transition-condition** is a transition-condition without previous steps. Each time the transition condition is TRUE, the next step will be activated. It is recommended to provide a transition condition with a rising or dropping flank to avoid the activation of the next step. **Example**:The initialization step 0 will be activated on a rising flank from the initialization input signal. 

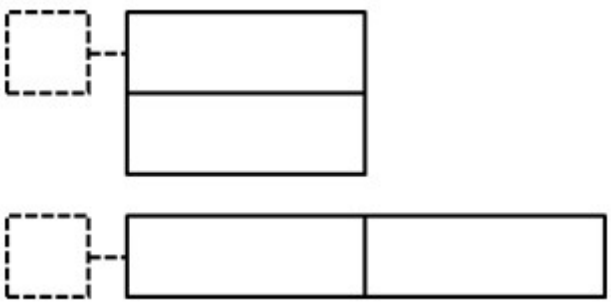
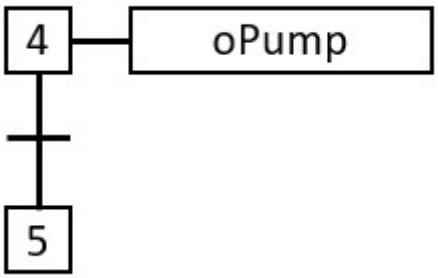
Symbol	Description
	An **end transition-condition** is a transition-condition where no steps follows. Each time the transition condition is TRUE, the upwards steps will be disabled. **Example**: Sequence with initialization of a sourcestep where all steps get activated 

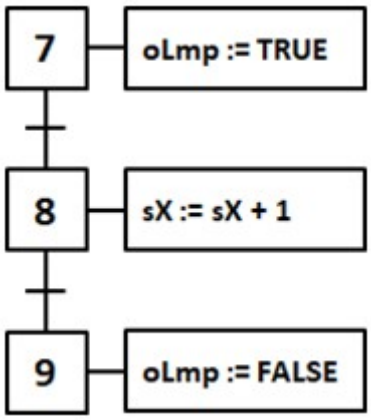
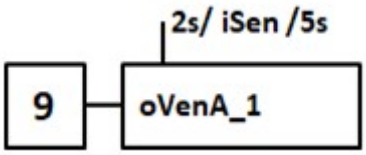
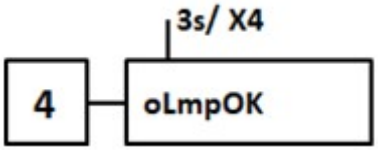
Explanation symbolic image

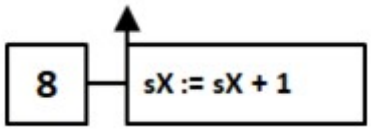
- iInit = digital input – GRAFCET initialise
- iStarted = digital input – Result of a start-stop circuit
- iSen1 = digital input – Sensor 1
- iSen2 = digital input – Sensor 2

Action

Symbol	Description

Symbol	Description
	An **action** is assigned to a step and gets illustrated by a rectangle which is connected to that step with a horizontal line. It is allowed to use multiple actions in the same step, if each of them have their own rectangle. **Allowed multiple actions:** 
	Each action has an action label which clarifies the executed task. The label is written in the rectangle where the asterisk symbol is replaced by a variable. A **continue action** will have the status of the variable TRUE the moment the corresponding step is active. All other moments the action is FALSE. **Example**: The pump action is TRUE on step 4 and FALSE on step 5 

Symbol	Description
	A **memory action** has a specific value assigned to a variable which gets stored. The asterisk symbol gets replaced by a variable and the # symbol gets replaced by a (mathematical) value, formula, ... **Example**: The lamp gets activated in step 7, is still activated in step 8 and turns off in step 9. The internal variable "sX" is increased by 1 in step 8. 
	A time dependent **conditional action** is displayed with the / symbols. The action is TRUE after an on-delay and stays TRUE with an off-delay. It is allowed to simplify the condition by removing the off-delay in case this isn't used. **Example**: 2s after "iSen" becomes TRUE valve A+ will be activated. 5s after "iSen" become FALSE valve A+ will be deactivated if step 9 is activated.  **Example**: 3s after step 4 is activated the OK lamp lights up. 

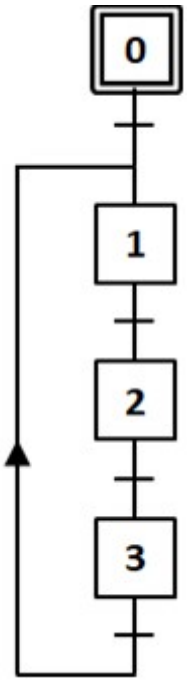
Symbol	Description
	It is possible to run a memory action with the activation of a step. This is indicated with an upwards arrow. **Example**: With the activation of step 8 the formula will be ran. 

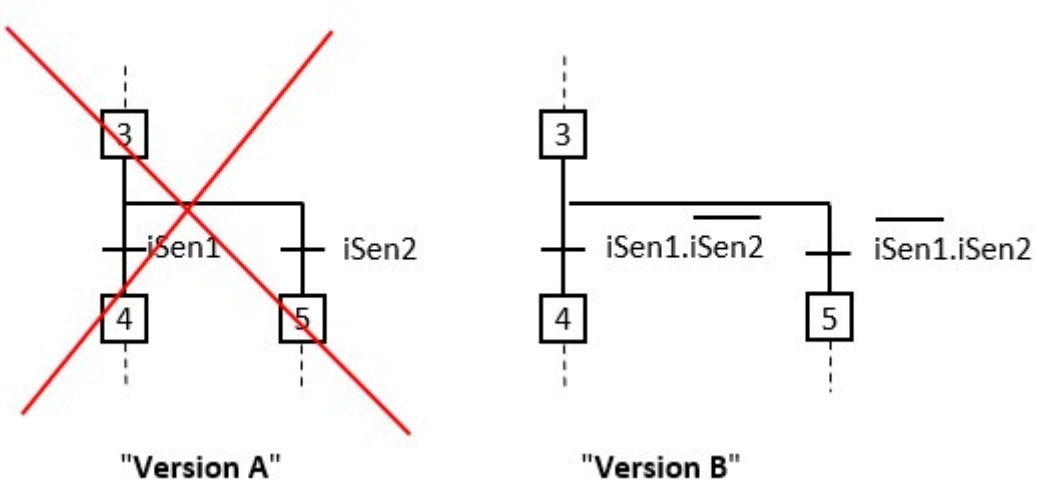
Explanation of the used symbols:

- oPump = digital output – Activation of a pump
- oLmp = digital output – Activation of a lamp
- oLmpOK = digital output – Activation of an OK lamp
- oVlvA_1 = digital output – Activation of Valve A+
- sX = static variable X
- sNumNOK = static variable – Amount of NOK parts

Structures

Symbol	Description
	A **sequence** is a series of steps where each step contains max. one transition-condition. The sequence is active if at least one step of the sequence is active. The sequence is inactive when all steps are inactive.
	A simple **loop sequence** is a sequence of steps whereby each step contains max. one transition-condition and where the last step is connected to the first step.

Symbol	Description
	A **sequence with source step** has a step without previous transition- condition. **Example**: Sequence with a initializing sourcestep. 
	A **sequence with end step** has a step where there are no transition- conditions after it. An end step (and a source step) are necessary with macros.
	It is possible to jump to a step with a **forward sequence skip**. Notice that between 2 steps only one transition is allowed.
	It is possible to loop back with a **backwards sequence skip**. This makes it possible to repeat a sequence. Notice that between 2 steps only one transition is possible.

Symbol	Description
	Using a **OR-convergence** makes it possible to choose between different sequences where between 2 steps only one transition is allowed. The designer needs to make sure that both sequences can't be activated at the same time. **Example**: In the GRAFCET version A it is possible to activate both step 4 and 5. This can happen when "iSen1" and "iSen2" have the status TRUE and in the moment step 3 is active. In the GRAFCET version B it isn't possible due to the extended transition-condition. 
	A **AND-convergence** allows to activate parallel sequences at the same time. They will be started after a starting transition. An startind AND-convergence is showed by means of a double line after the starting transition. Once the parallel sequence is activated both sequences will run seperatly from each other. An AND-convergence gets back ends if all the parallel end steps are active and the ending transition-condition is TRUE. An ending AND-convergence is showed by a double line before the ending transition.

Function rules

The **function of a GRAFCET** is in general step by step.
If a step is active and the transition condition(s) are met than the next step will be activated. If the next

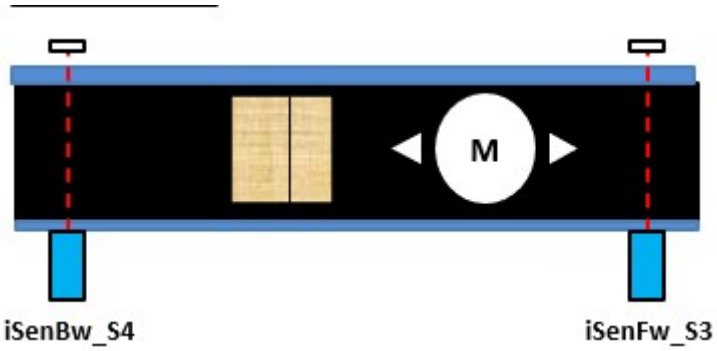
step gets activated the previous step will be deactivated immediately.

It is possible that the status of the different transtion-conditions the fucntion of a GRAFCET seems not to run step by step. It is the task of the designer to avoid that functions which can cause an unstable function of actions.

Function	Description
	A **non transient action** will run step by step. Situation: Step 4 active, iSen1 = iSen2 = Isen3 = FALSE Function: iSen1 (1) is TRUE which activates step 5 and deactivates step 4.
	With a **transient action** the steps won't run step by step. Situation Step 4 active, iSen1 = iSen3 = FALSE iSen2 = TRUE Function: iSen (1) is TRUE which causes step 5 to be activated and step 4 gets deactivated. Because iSen3 (2) is true, step 6 will inmediatly be activated and step 5 will be deactivated. Disadvantage: In case we use an action instead of a memory action, it is possible that the assigned actions of a transient step are not or transient executed (= unstable function).
	An **AND-convergence** parallel sequences will be started in case the previous transition condition is TRUE. Situation 1: If step 1 is active and transition conditoin iGestart is TRUE then step 2 and 4 will be activated. Situation 2: Once the parallel sequences are activated they will run separatly. Situation 3: Step 9 gets activated in case step 3 and step 6 are active and in case the transition-condition "iSen1.iSen2" is TRUE.

Example

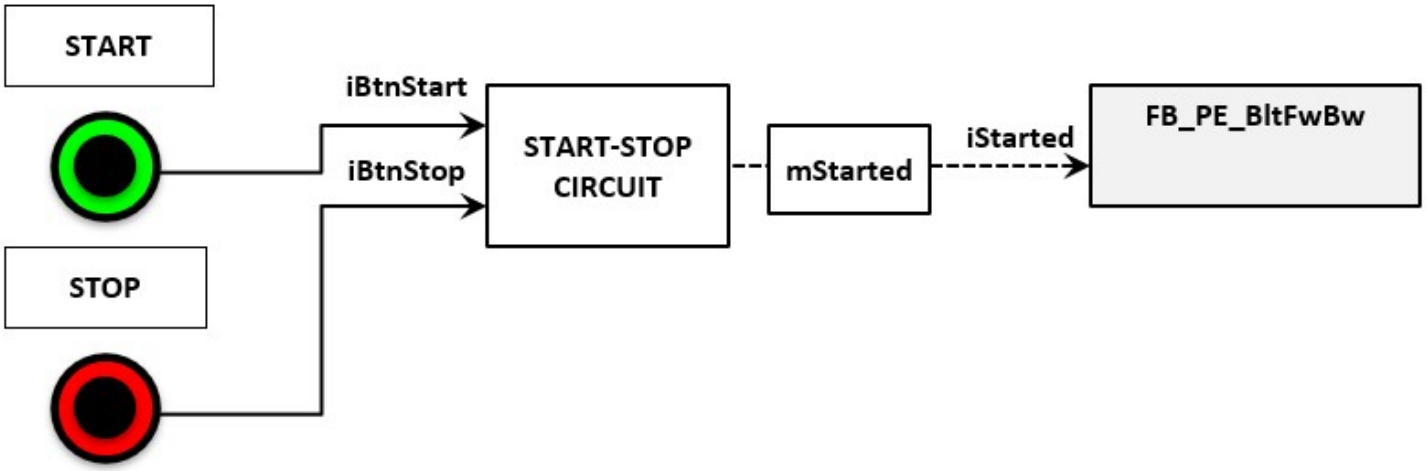
The next example shows a GRAFCET for the functionality of a conveyor belt. A box is displaced 5x times from start to end before it stops. After this operation it is necessary to restart the installation.



The GRAFCET has the name FB_PE_BeltFwBw:

- FB = GRAFCET will be programmed in a function block (FB)
- PE = This part is a procedure element according ANSI/ISA S88 standard
- BeltFwBw = Conveyor belt forwards & backwards

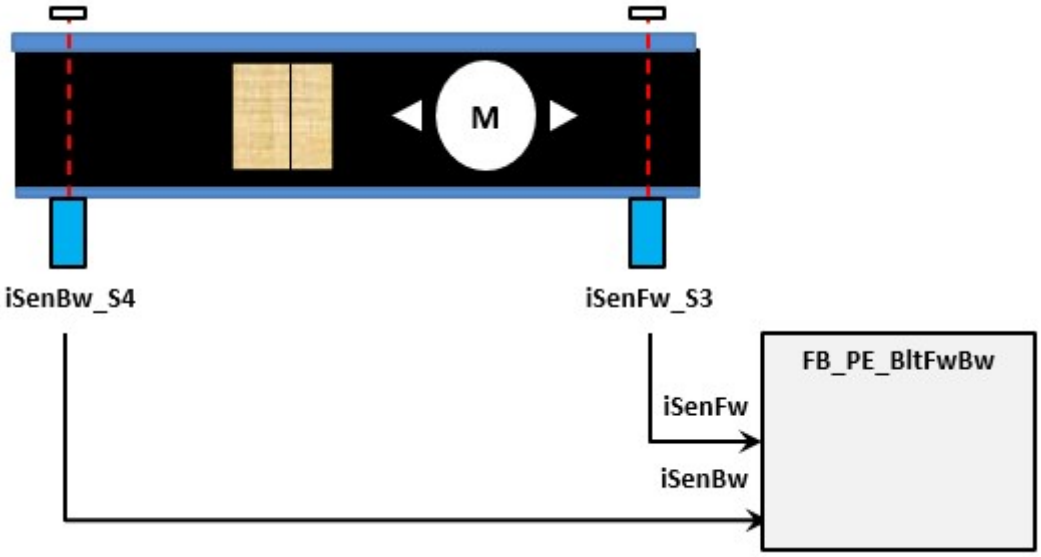
The conveyor belt is **started and stopped** by means of a start button and a stop button. The functionality of these buttons is not included in the GRAFCET but gets executed by an external start-stop basic circuit. The result of this start-stop basic circuit will be linked with the GRAFCET input variable "iStarted".



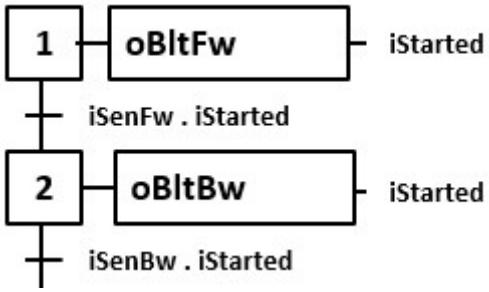
Each time the stop button is pressed the conveyor belt will immediatly stop. When the start button is pressed again, the conveyor and GRAFCET continues where they ended.

Conveyorbelt GRAFCET

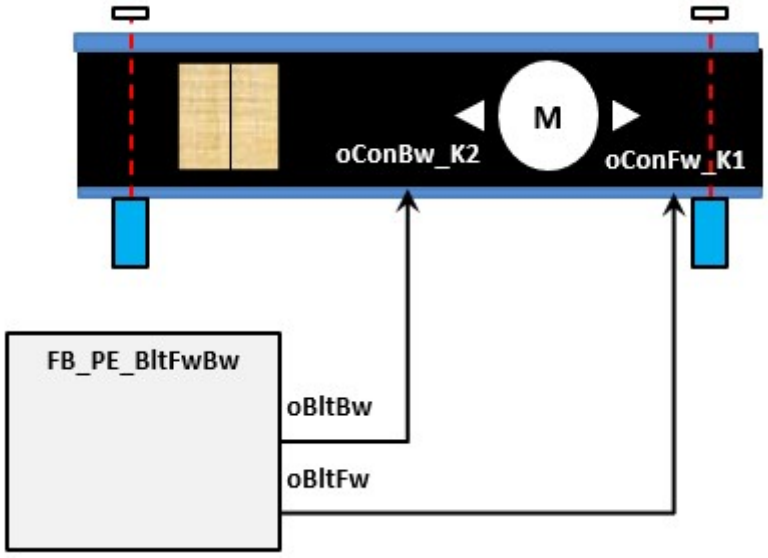
The **photocell** sensors on the conveyor belt detects the presence of the box when the infrared beam between photocell and reflector is interrupted. The status of the photocells (%I) is linked with the GRAFCET input variables "iSenFw" and "iSenBw".



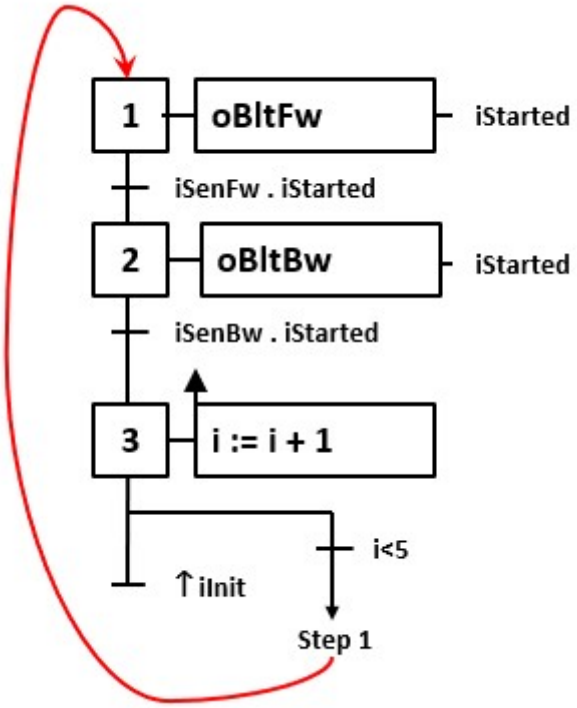
Controlling the conveyor belt forwards and backwards will be determined by step 1 and step 2 on condition that the installation is started.



The effective **control of the conveyor belt** happens by the GRAFCET output variables "oBeltFw" and "oBeltBw" which are linked to the contactors (%Q) and the conveyor belt motor (asynchronous motor).

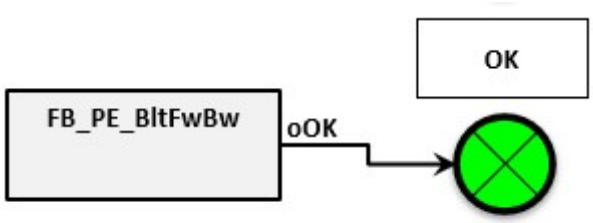


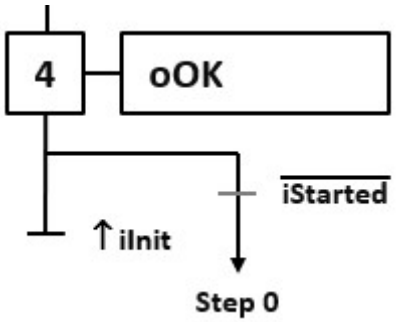
Counting of the **number of backward and forward movements** is controlled by the internal INT variable "i". This variable is an internal function block parameter of the type STATIC. This makes it possible to remember the condition of variable "i" also without voltage (=retentive).



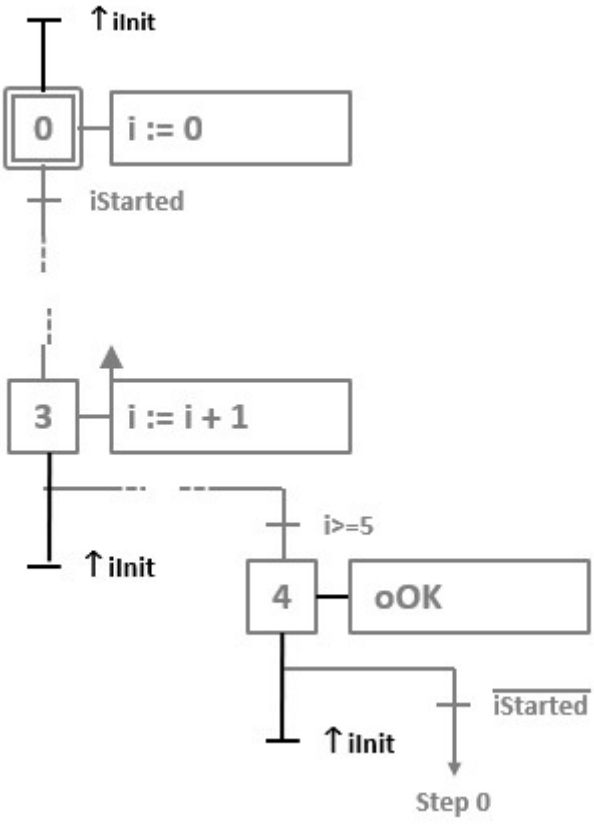
Increasing the variable "i" is executed in step 3, after which step 1 gets activated because there is a loop sequence between step 3 and step 1 but only if the value of the variable "i" is less than the decimal value 5. Noticed that the increasing of the variable "i" is only executed on the moment that step 3 is activated (rising edge). This is to prevent wrongly increasing the value of the variable in case step 3 is active longer the one PLC cycle.

In case the box went 5x backwards and forward this will be displayed with a green OK lamp. This lamp (%Q) is connected with the GRAFCET output variable "oOk". Now the installation needs to be stopped with the stop button before the installation can restart.

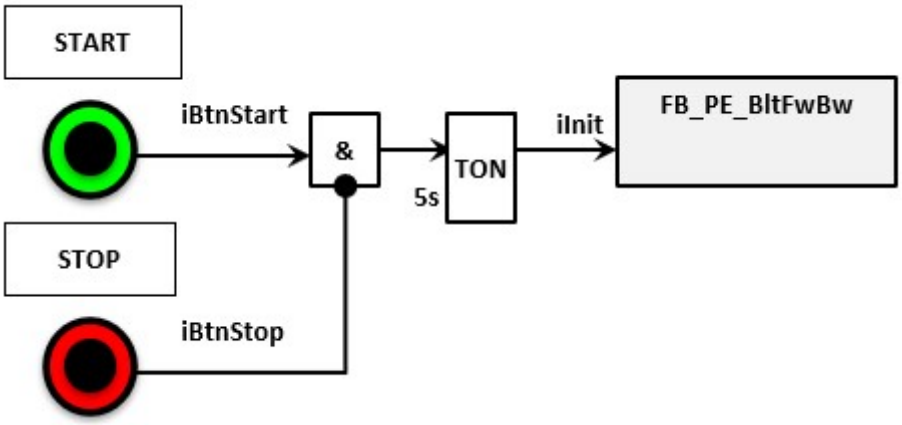




It is possible to **initialize** the GRAFCET. This is the activation of the initial step (step 0) by using GRAFCET input variable "iInit". All the other active steps get deactivated. The initialising is only activated on the rising edge of "iInit".



You could choose to initialize the installation in case you press the start and stop button simultaneously for 5 seconds or more.



GRAFCET programming in LAD-FBD using BOOL

Converting a **GRAFCET design to software code** is demonstrated with the GRAFCET described in subchapter 2.

The GRAFCET is programmed in the LAD or FBD programming language in the function block (%FB) with the use of STATIC parameters. STATIC parameters can remember their status without the PLC being powered on if they are configured to retain.

FB_PE_BltFwBw

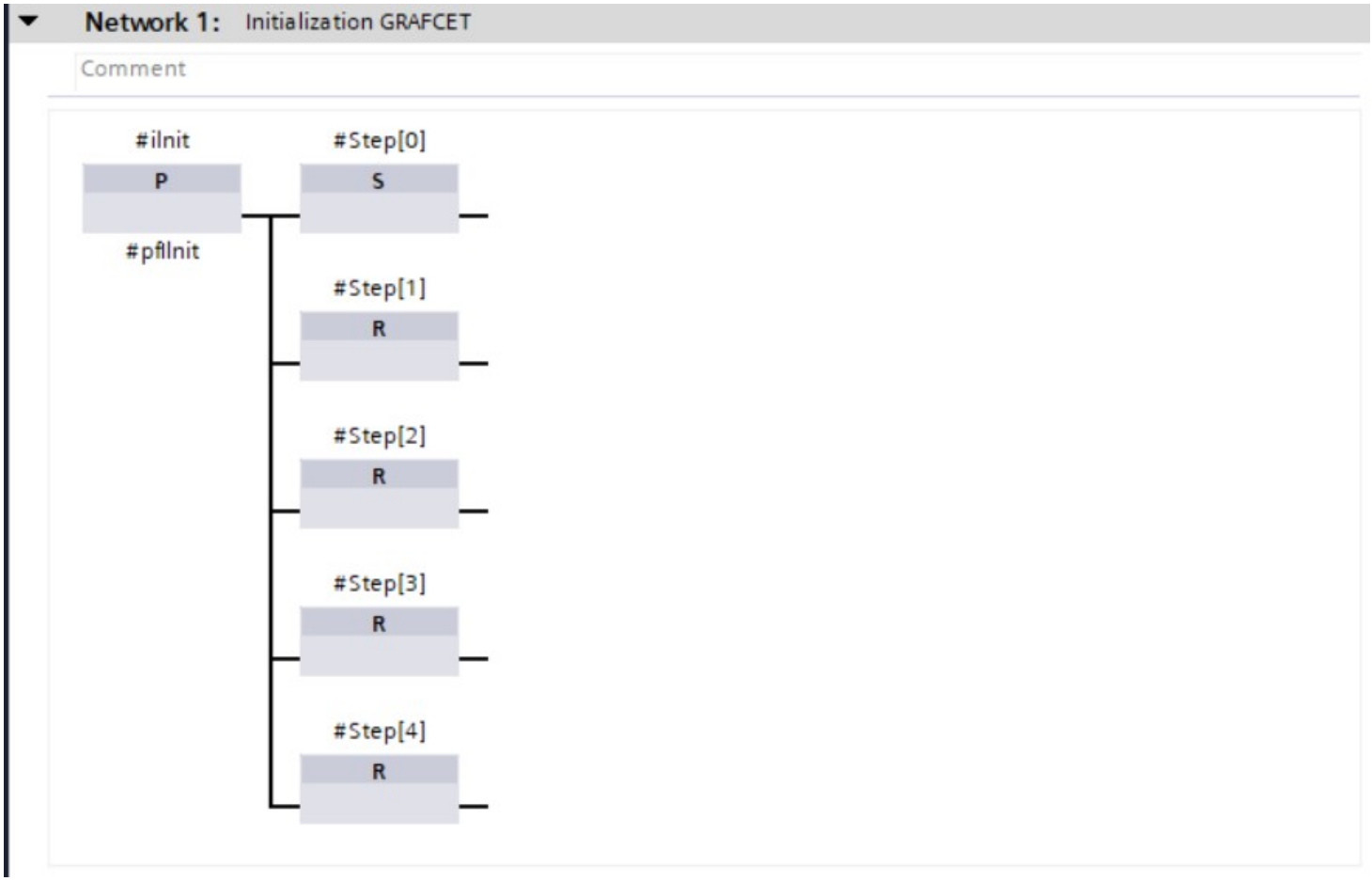
	Name	Data type	Default value	Retain	Setpoint	Comment
2	iInit	Bool	false	Non-retain		Initialisation
3	iStarted	Bool	false	Non-retain		Installation started
4	iSenFw	Bool	false	Non-retain		Front sensor
5	iSenBw	Bool	false	Non-retain		Back sensor
6	Output					
7	oBltFw	Bool	false	Non-retain		Belt forward
8	oBltBw	Bool	false	Non-retain		Belt backwards
9	oOK	Bool	false	Non-retain		Movement finished
10	InOut					
11	Static					
12	pfiInit	Bool	false	Retain		Help variable rising edge iInit
13	pfx3	Bool	false	Retain		Help variable rising edge step 3
14	Step	Array[0..4] of Bool		Retain		Active GRAFCET step
15	i	Int	0	Retain		Counter

The programming is split into **3 parts** which are chronologically programmed in different networks:

- Initialization (network 1)
- Transition-conditions (network 3 ... x)
- Actions (network x+1 ... last network)

The **GRAFCET programming in LAD/FBD with BOOL** follows the next rules

- Each step is repesented by an unique BOOL variable
- This variable is an ARRAY of BOOL starting with 0 and ending with max. step number
- In case the corresponding variable is TRUE, the step will be active
- Input "iInit" is always present which causes the activation of the initial step on a rising edge of this input
- Input "iStarted" is always present which processes the result of an external start-stop basic circuit

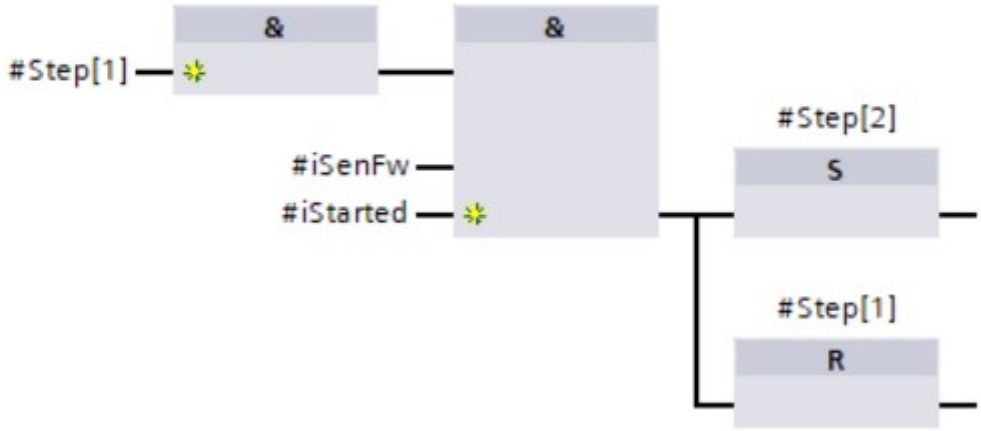


Network 2: Transition-condition step 0 to step 1

Comment

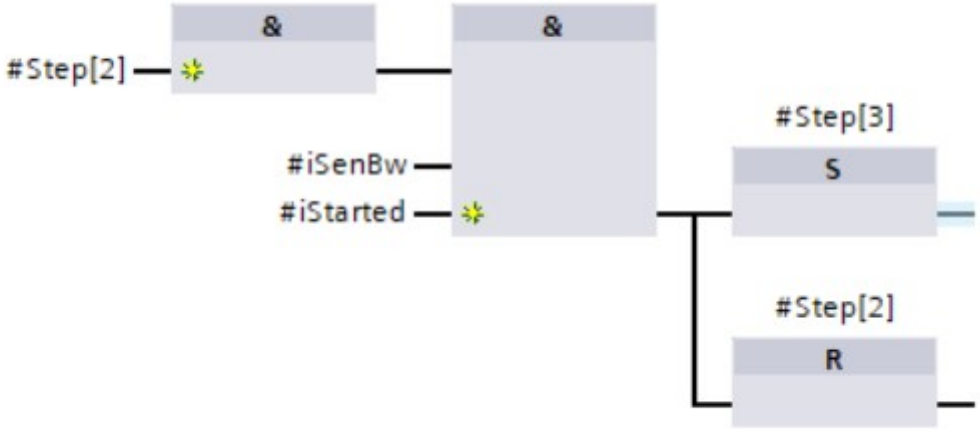
Network 3: Transition-condition step 1 to step 2

Comment



Network 4: Transition condition step 2 to step 3

Comment



Network 5: Transition-condition step 3 to step 1 or step 3

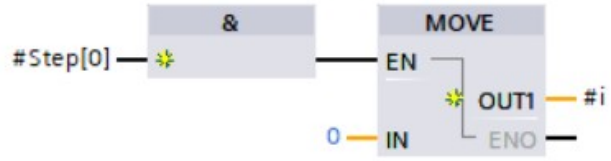
Comment

Network 6: Transition-condition step 4 to step 0

Comment

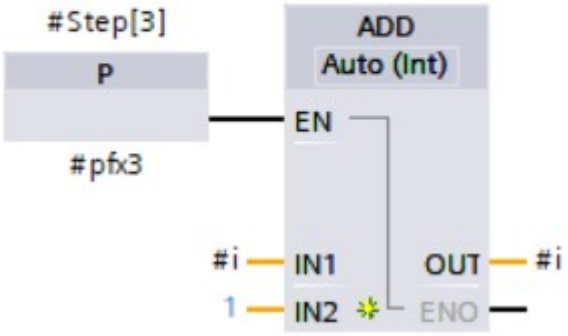
Network 7: Action - set i to 0

Comment



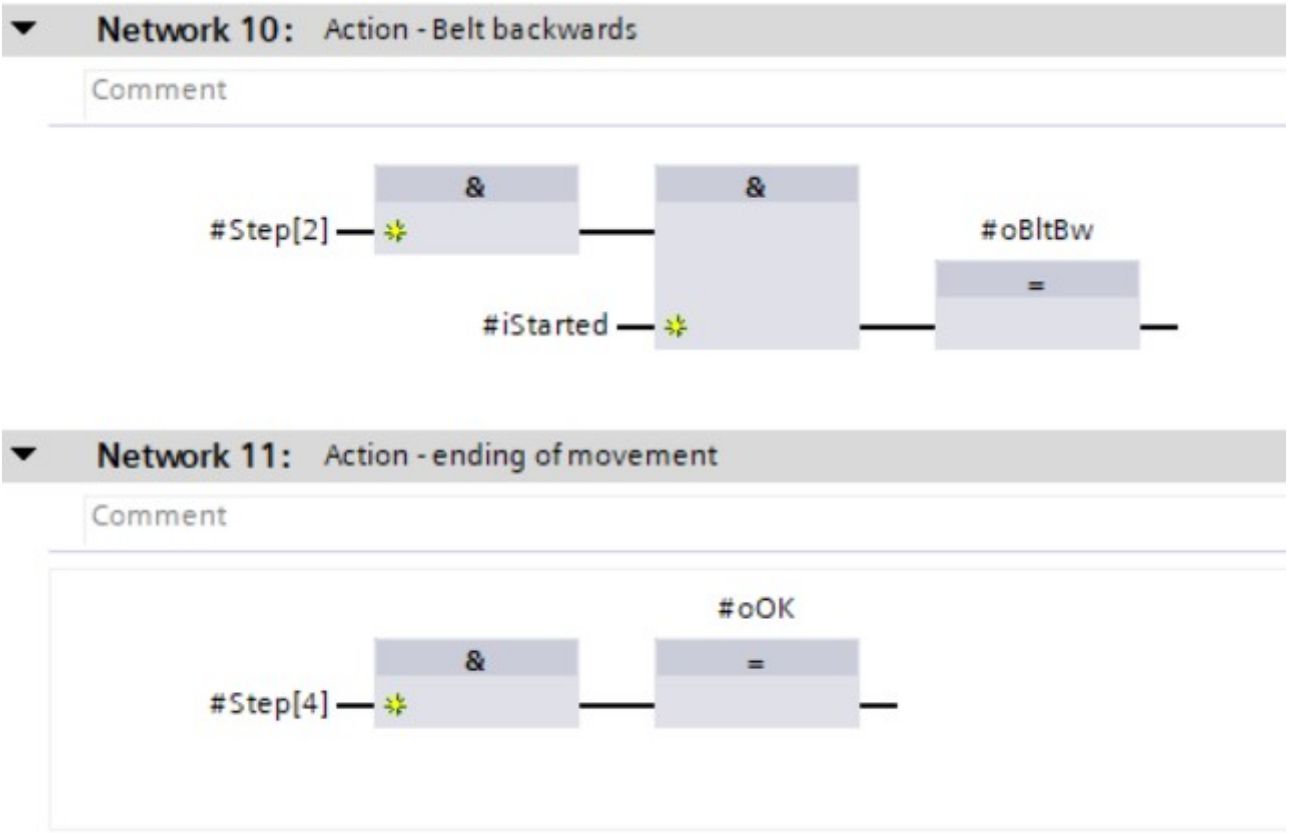
Network 8: Action i= i +1

Comment



Network 9: Action - Belt forward

Comment



Advantages	Disadvantages
Simplicity (1 step = 1 variable)	Initial step is not activated during the first download of the program
	Monitoring of active steps is complicated

GRAFCET programming in LAD-FBD using INT

Converting a **GRAFCET design to software code** is demonstrated with the GRAFCET described in subchapter 2.

The GRAFCET is programmed in the LAD or FBD programming language in the function block (%FB) with the use of STATIC parameters. STATIC parameters can remember their status without the PLC being powered on if they are configured to retain.

FB_PE_BltFwBw

		Name	Data type	Default value	Retain	Setpoint	Comment
1		Input					
2		iInit	Bool	false	Non-retain		Initialisation
3		iStarted	Bool	false	Non-retain		Installation started
4		iSenFw	Bool	false	Non-retain		Front sensor
5		iSenBw	Bool	false	Non-retain		Back sensor
6		Output					
7		oBltFw	Bool	false	Non-retain		Belt forward
8		oBltBw	Bool	false	Non-retain		Belt backwards
9		oOK	Bool	false	Non-retain		Movement finished
10		InOut					
11		Static					
12		pfiInit	Bool	false	Retain		Help variable rising edge iInit
13		pfx3	Bool	false	Retain		Help variable rising edge step 3
14		Step	Int	0	Retain		Active GRAFCET step
15		i	Int	0	Retain		Counter

The programming is split into **3 parts** which are chronologically programmed in different networks:

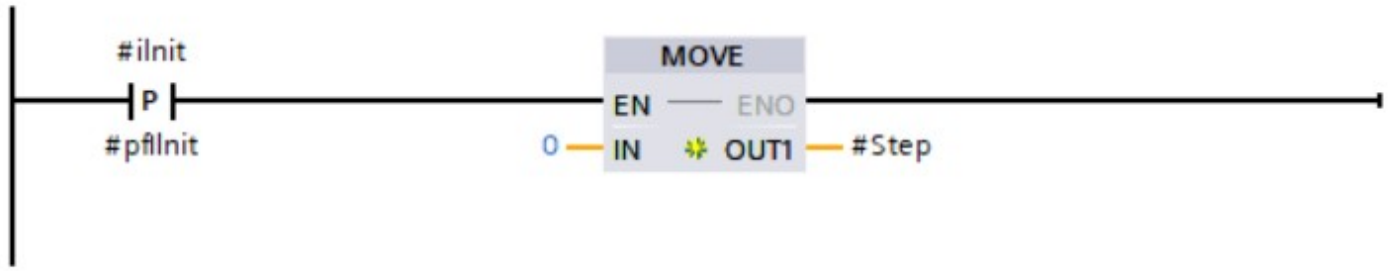
- Initialization (network 1)
- Transition-conditions (network 3 ... x)
- Actions (network x+1 ... last network)

The **GRAFCET programming in LAD/FBD with INT** follows the next rules

- Only the actual step needs to be known
- The actual step is represented by an STATIC INT variable (step)
- The initial value of this variable is the decimal value 0
- The actual value of this variable corresponds to the active GRAFCET step
- The initial step is automatically activated the first time the software is downloaded to the PLC; this is because the INT number initial value is equal to the decimal value 0
- Input "iInit" is always present which causes the activation of the initial step on a rising edge of this input
- Input "iStarted" is always present which processes the result of an external start-stop basic circuit

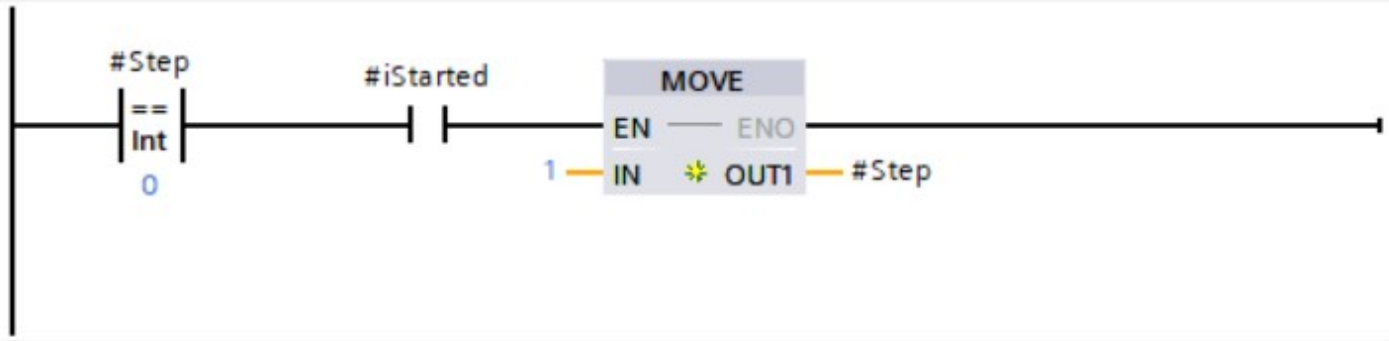
Network 1: Initialization GRAFCET

Comment



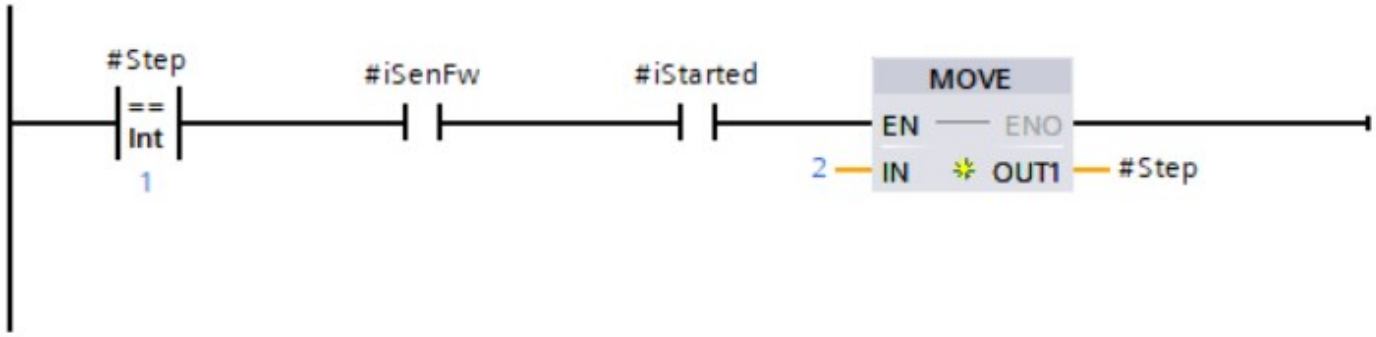
Network 2: Transition-condition step 0 to step 1

Comment



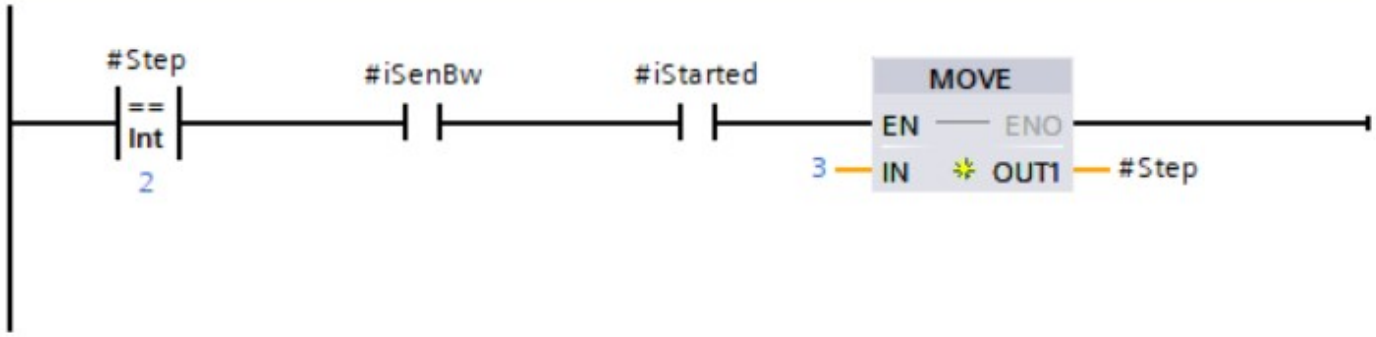
Network 3: Transition-condition step 1 to step 2

Comment



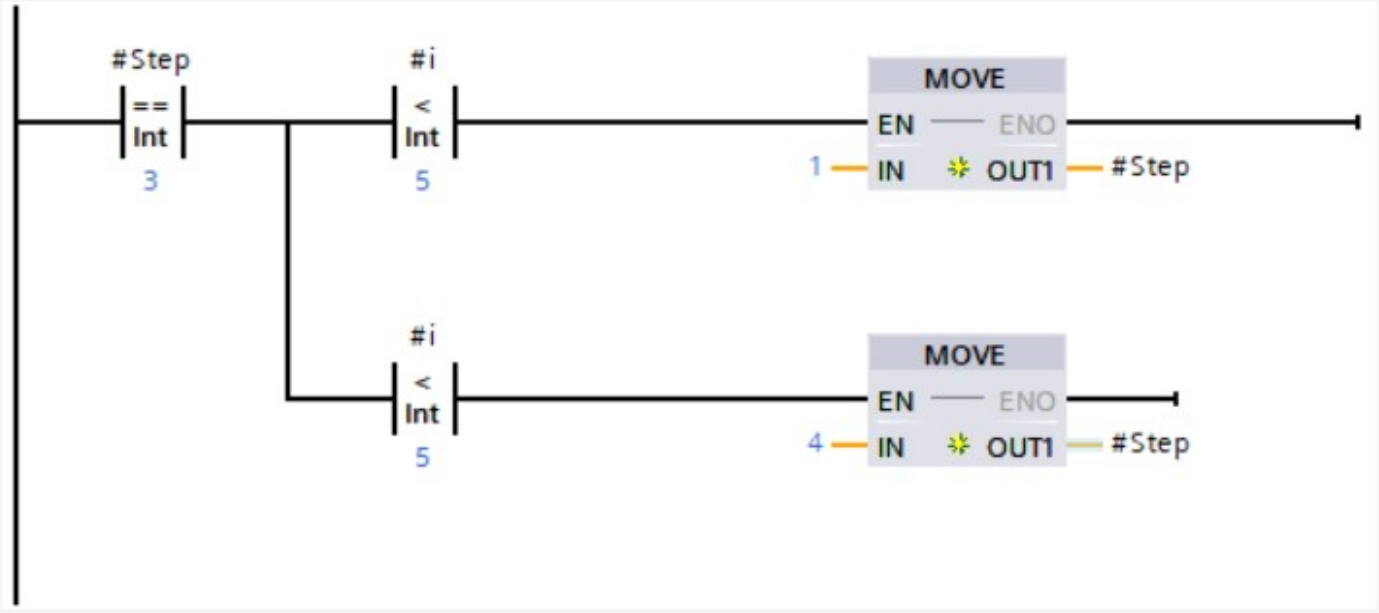
Network 4: Transition condition step 2 to step 3

Comment



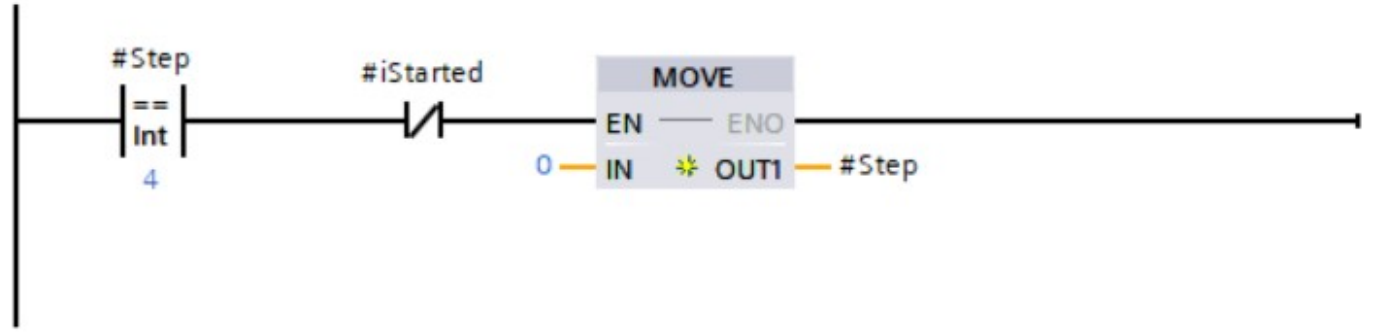
Network 5: Transition-condition step 3 to step 1 or step 3

Comment



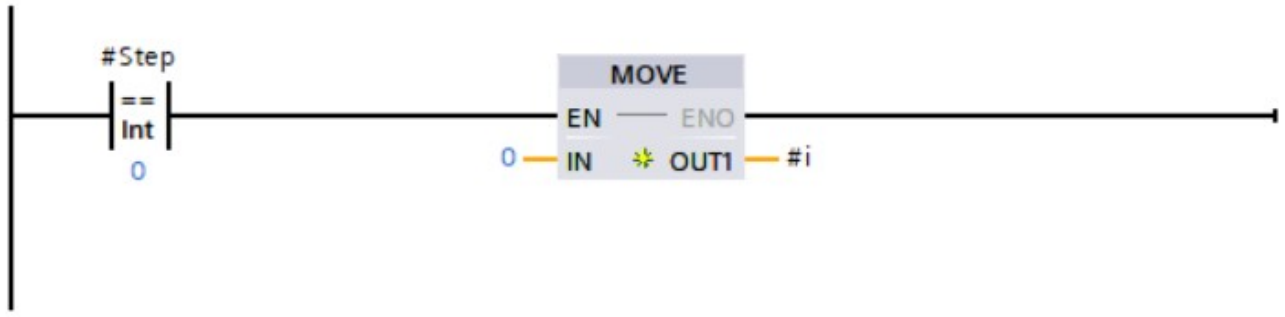
Network 6: Transition-condition step 4 to step 0

Comment



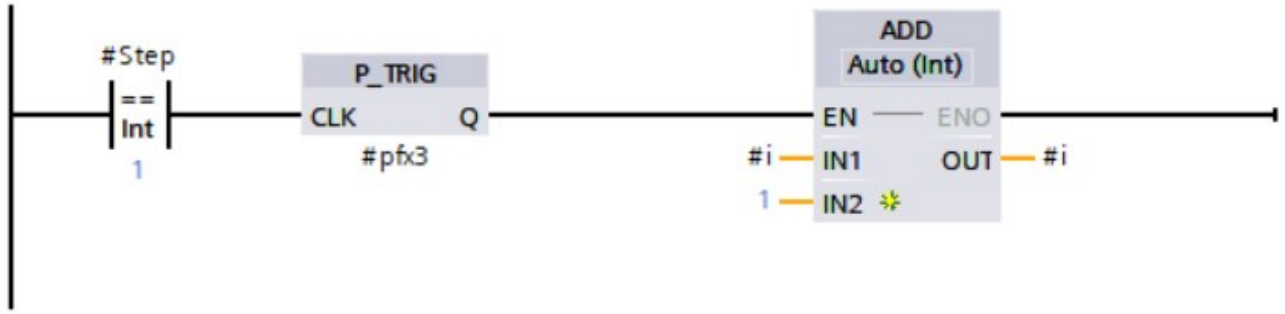
Network 7: Action - set i to 0

Comment



Network 8: Action i= i +1

Comment

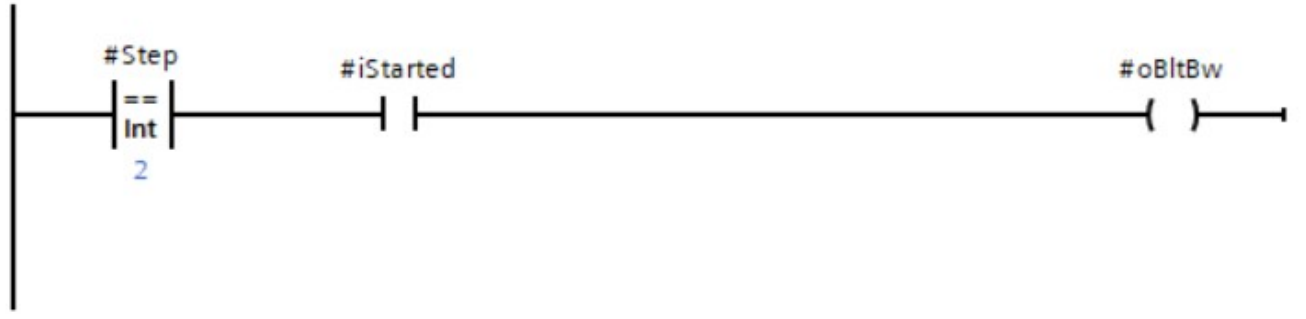


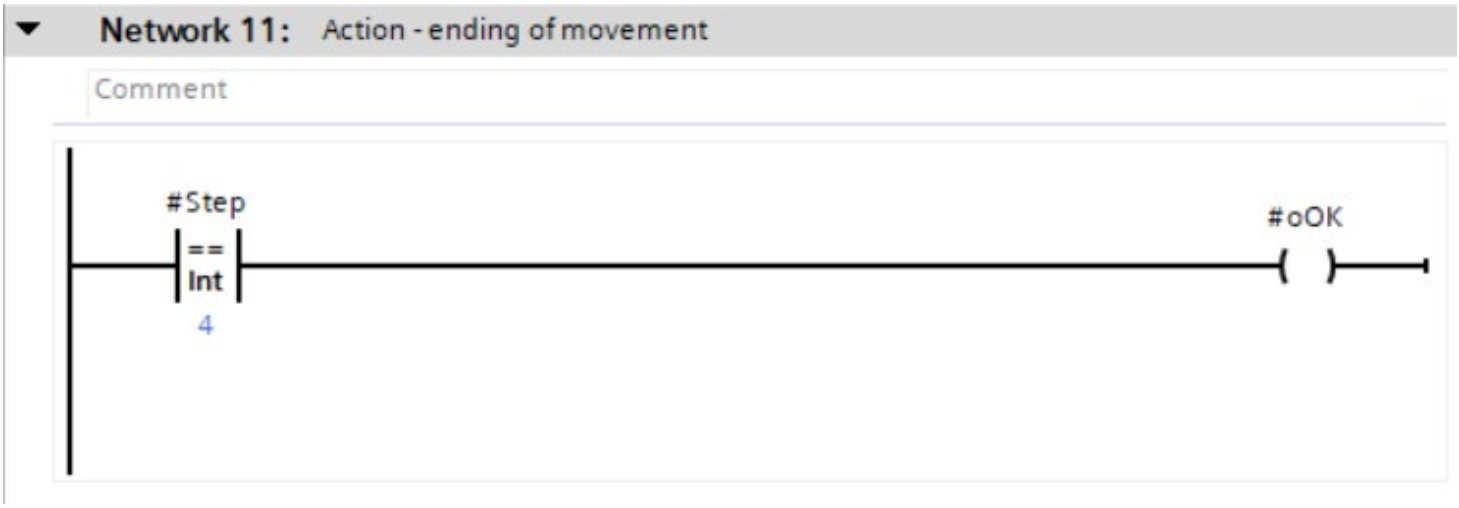
Network 9: Action - Belt forward

Comment

Network 10: Action - Belt backwards

Comment





Advantages	Disadvantages
Initial step is activated during the first download of the program	More complex, advanced programming then with LAD/FBD BOOL method
Monitoring of active steps is easier	Programming of AND-convergence is more complex then with LAD/FBD BOOL variant

GRAFCET programming in ST

Converting a **GRAFCET design to softwarecode** is demonstrated with the GRAFCET described in subchapter 2.

The GRAFCET is programmed in the LAD or FBD programming language in the function block (%FB) with the use of STATIC parameters. STATIC parameters can remeber their status also without voltage if they are configured as retain.

FB_PE_BltFwBw

	Name	Data type	Default value	Retain	Setpoint	Comment
1	Input					
2	iInit	Bool	false	Non-retain		Initialization
3	iStarted	Bool	false	Non-retain		Installation started
4	iSenFw	Bool	false	Non-retain		Front sensor
5	iSenBw	Bool	false	Non-retain		Back sensor
6	Output					
7	oBltFw	Bool	false	Non-retain		Belt forward
8	oBltBw	Bool	false	Non-retain		Belt backwards
9	oOK	Bool	false	Non-retain		Movement finished
10	InOut					
11	Static					
12	pfiInit	Bool	false	Non-retain		Help variable rising edge iInit
13	pfx3	Bool	false	Non-retain		Help variable rising edge step 3
14	Step	Int	0	Non-retain		Active GRAFCET step
15	i	Int	0	Non-retain		Counter
16	ID_pfiInit	R_TRIG				Rising edge iInit
17	ID_pfx3	R_TRIG				Rising edge pfx3

The programming is split into **3 parts** which are chronologically programmed in different networks:

- Initialisation (network 1)
- Transition-conditions (network 3 ... x)
- Actions (network x+1 ... last network)

The **GRAFCET prgramming in ST** is submitted to the next rules

- The use of CASE .. OF .. ELSE control structure which handles the processing of transition-conditions
- Only the actual step needs to be known
- The actual step is represented by a STATIC ANY_INT variable (step)
- The initial value of this variable is the decimal value 0
- The actual value of this variable corresponds to the active GRAFCET step
- The initial step is automatically activated the first time the software is downloaded to the PLC; this because the INT number initial value is equal to the decimal value 0
- Input "iInit" is always present which causes the activation of the initial step on a rising edge of this input
- Input "iStarted" is always present which processes the result of an external start-stop basic circuit

```
//=====================================================
// PART 1 : INITIALIZATION
// =====================================================
// Rising edge
#ID_pfilnit(CLK := #pfilnit,
            Q => #pfilnit);

// Deleting current step and setting the value to 0
// End transition-condition & source transition-condition
IF #pfilnit = TRUE THEN
    #Step := 0;
END_IF;

//=====================================================
// PART 2 : TRANSITION-CONDITIONS
// =====================================================

CASE #Step OF
    0://Transition-condition from step 0 to step 1
        IF #iStarted = TRUE THEN
            #Step := 1;
        END_IF;
    1://Transition-condition from step 1 to 2
        IF #iSenFw = TRUE AND #iStarted = TRUE THEN
            #Step := 2;
        END_IF;
    2://Transition-condition from step 2 to 3
        IF #iSenBw = TRUE AND #iStarted = TRUE THEN
            #Step := 3;
        END_IF;
    3://Transition-condition from step 3 to 1 or step 4
        IF #i < 5 THEN
            #Step := 1;
        ELSE
            #Step := 4;
        END_IF;
    4://Transition-condition from step 4 to step 0
        IF #iStarted = FALSE THEN
            #Step := 0;
        END_IF;
END_CASE;
```

```
43  //=====================================================
44  // PART 3 : ACTIONS
45  // =====================================================
46  // Counter i
47  // STEP 0 => i:0
48 IF #Step = 0 THEN
49      #i := 0;
50  END_IF;
51
52  // STEP 3 => i:=i+1
53  #ID_pfx3(CLK:= #Step=3,Q=>#pfx3);
54 IF #pfx3 = TRUE THEN
55      #i := #i + 1;
56  END_IF;
57
58  // Output conveyor belt forward
59 IF #Step = 1 AND #iStarted = TRUE THEN
60      #oBltFw := TRUE;
61      #oBltBw := FALSE;
62  END_IF;
63
64  // Output conveyor belt backwards
65 IF #Step = 2 AND #iStarted = TRUE THEN
66      #oBltBw := TRUE;
67      #oBltFw := FALSE;
68  END_IF;


70  // Output conveyor belt OK
71 IF #Step = 3 THEN
72      #oOK := TRUE;
73  ELSE
74      #oOK := FALSE;
75  END_IF;
```

Advantages	Disadvantages
Initial step is not activated while the the first download of the program	More complex programming than LAD/FBD variant
Smaller programming then LAD/FBD variant	Programming of AND-convergence is more complex than the LAD/FBD BOOL method
Monitoring of active steps are easier	Debugging [2] in ST is harder than in FBD/LAD

Characteristics and defenitions

Controllers are used mainly to control continuing processes. Also non continuing can be controlled. This way a GRAFCET can run t he controller as action in a determined step. We use analog sensors to control analog or digital actuators.

Controllers are defined by several characteristics which are explained in the table below.

Definition	Abbreviation	Description
Process value	X *PV*	The value measured by the analog sensor *German name = Istwert*
Setpoint	W *SP*	The value that we want to achieve. *German name = Sollwert*
Loop manipulated value	Y *LMN*	The result that the actuator that affects the process adjusts *German name = Regelabweichung*
Error	E *ER*	Difference between measured values and the setpoint *German naming = Stellgrösse*
Hysteresis	H	The area wherein an actuator gets turned on and off.
Dead band	XDo *Db*	The area in which multiple digital acutators get turned off. With analog actuators the status of the output that isn't in the dead band changes.H *German name = Bandbreite*
Dead time	T0/Tt Td	The time delay between the change of the loop manipulated value and the change of the measured value. *German name = Totzeit*
Gain	Kr *GAIN*	The proportional action or P-action amplifies the output in proportion to input. The magnitude is determined by the gain factor, which can be positive or negative.
Reset time	TI	The integrating action ensures a constant sum of the error and keeps outputting more signals depending on how long an error exists between the measured and desired value. The integrator or I action is characterized by the time response of the integrator

Definition	Abbreviation	Description
Derivative time	TD	The D action responds to the rate of change of the error. So only when creating a step will the D action give its contribution to the controller. The differentiating action or D action is characterized by the time response of the differentiator

The measure difference is the difference between the loop manipulated value and the measured value.

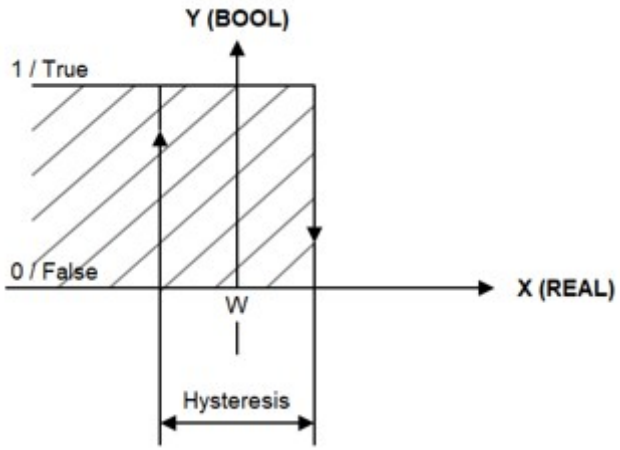
E=W-X

Notice that the control difference can have a positive or a negative value.

On-off circuit

The **on-off circuit** gets used to switch a control output Y [BOOL] on or off in function of a measured value X [REAL] and a set value W [REAL]. The on-off switch ensures that the actuator does not switch on and off too often by using 2 threshold values, namely:

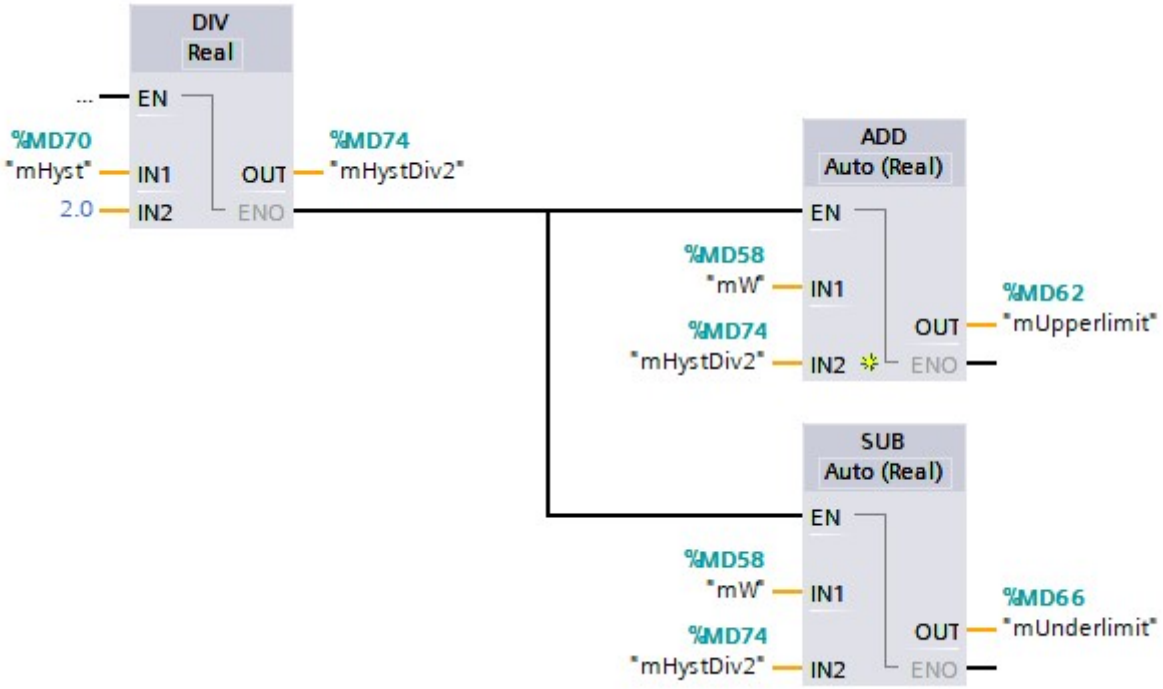
- The switch-on threshold value (lower limit)
- The switch-off threshold value (upper limit)



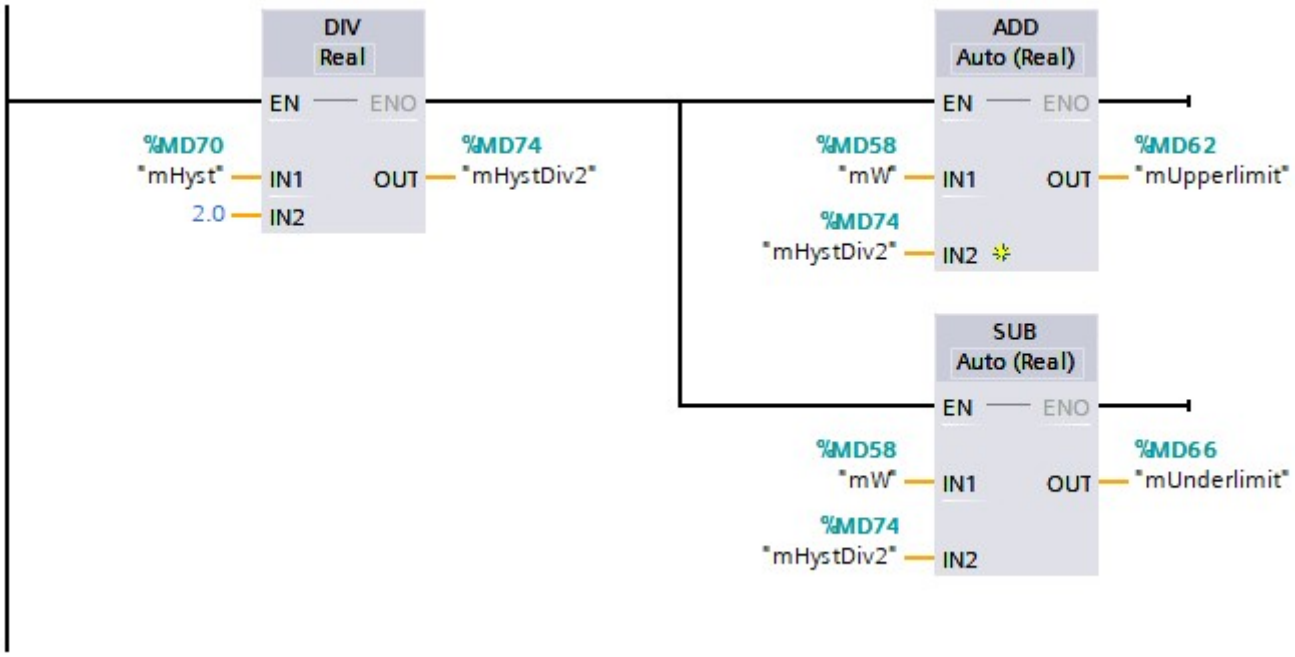
The difference between the switch-on and switch-off threshold values becomes the so called hysteresis. The following mathematical formulas apply:

Switch-on threshold or lower limit = X - Hysteresis/2.0
Switch-off threshold or upper limit = X + Hysteresis/2.0

FBD



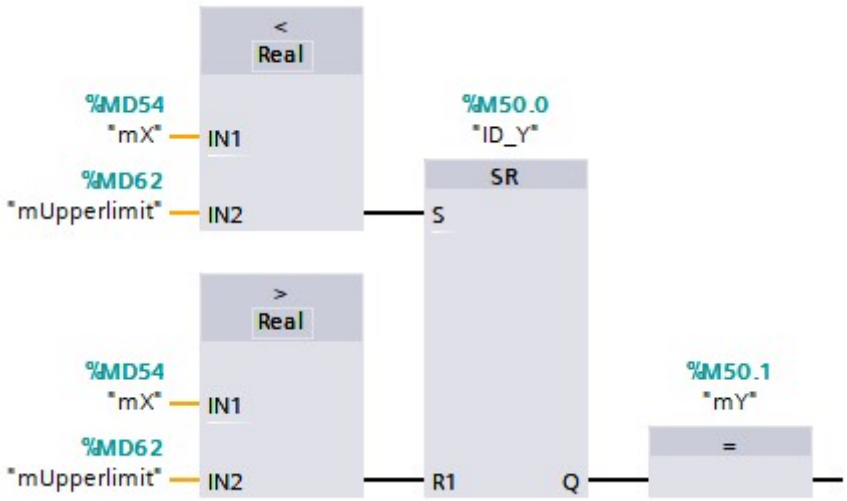
LD



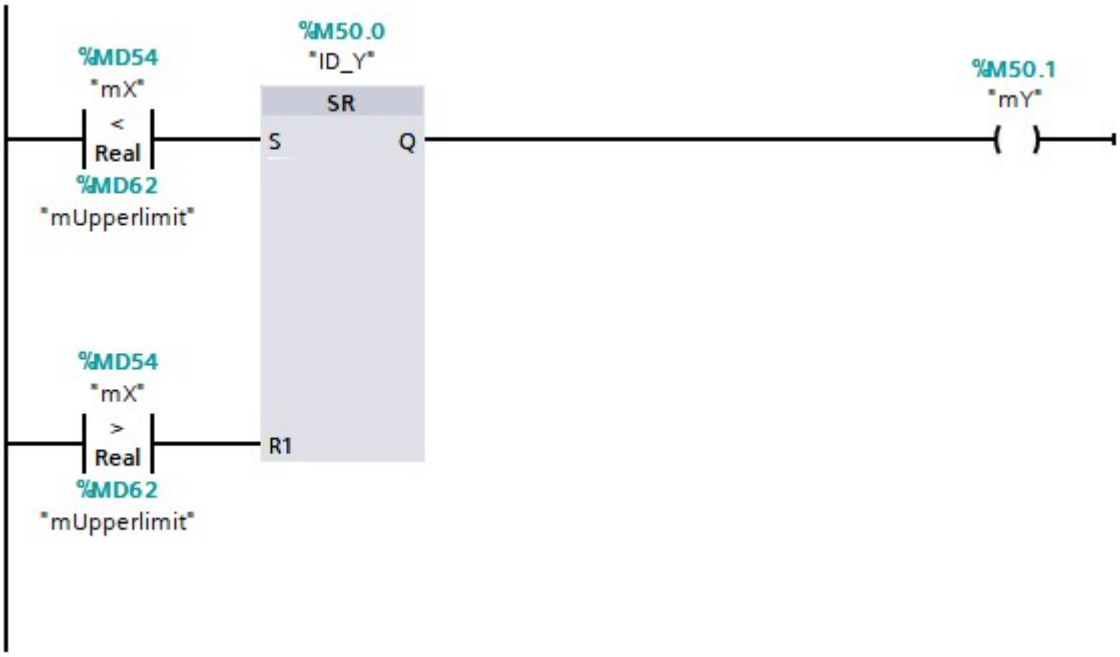
In case the process value is lower then the switch-on threshold, the loop manipulated value will be turn off.

As soon as the process value reaches the switch-off threshold, the loop manipulated value will be turn off.

FBD



LD



Example on-off switch - Heating in a home

Homes are often equipped with a thermostat to measure and control the temperature in a room.

- The thermostat measures the room temperature = measured value X
- The ideal temperature is entered on the thermostat = desired value W
- If it is too cold, the thermostat ensures that the boiler is switched on = closed contact = control output Y on
- If it is too hot, the thermostat ensures that the boiler is switched off = open contact = control output Y off
- Depending on the type of thermostat, the hysteresis is a fixed value or adjustable (order of magnitude 0.5 to 1.0 ° C)

PID controller

8.2 Addendum 5 Controllers

Functioning

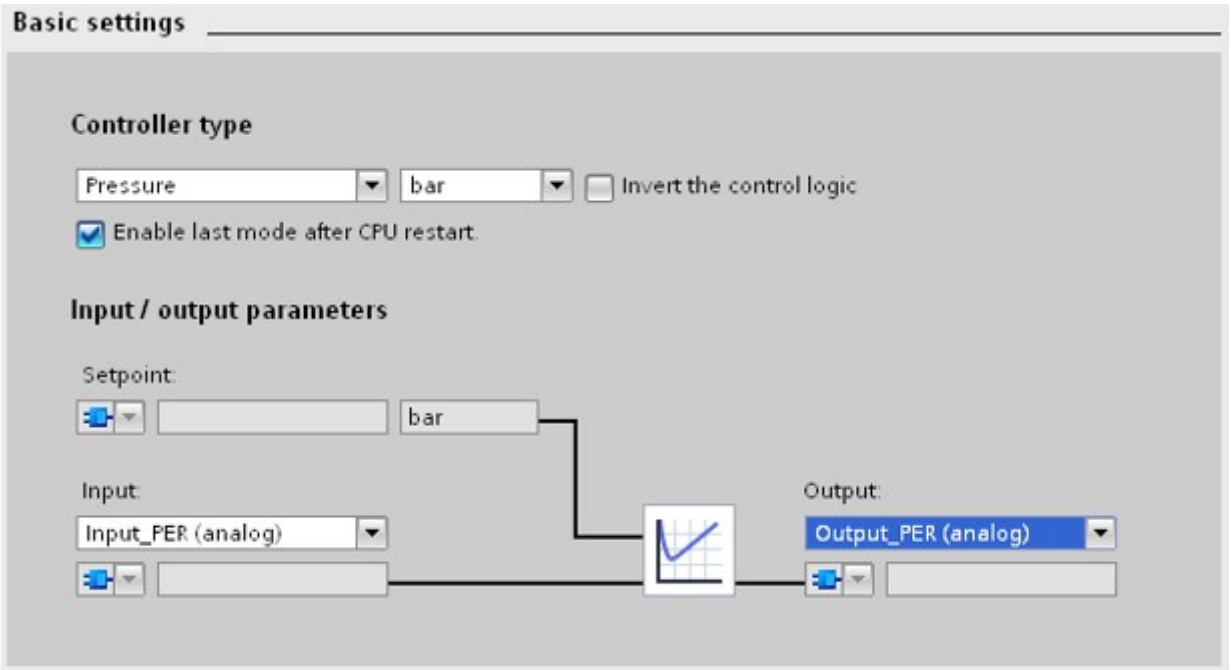
A PID controller is used to control processes via an analog actuator
[Y = INT or WORD]. A PID controller consists of several sub-functions. So, one distinguishes:

- P or Proportional action
- I or Integrative action
- D or Differentiating action

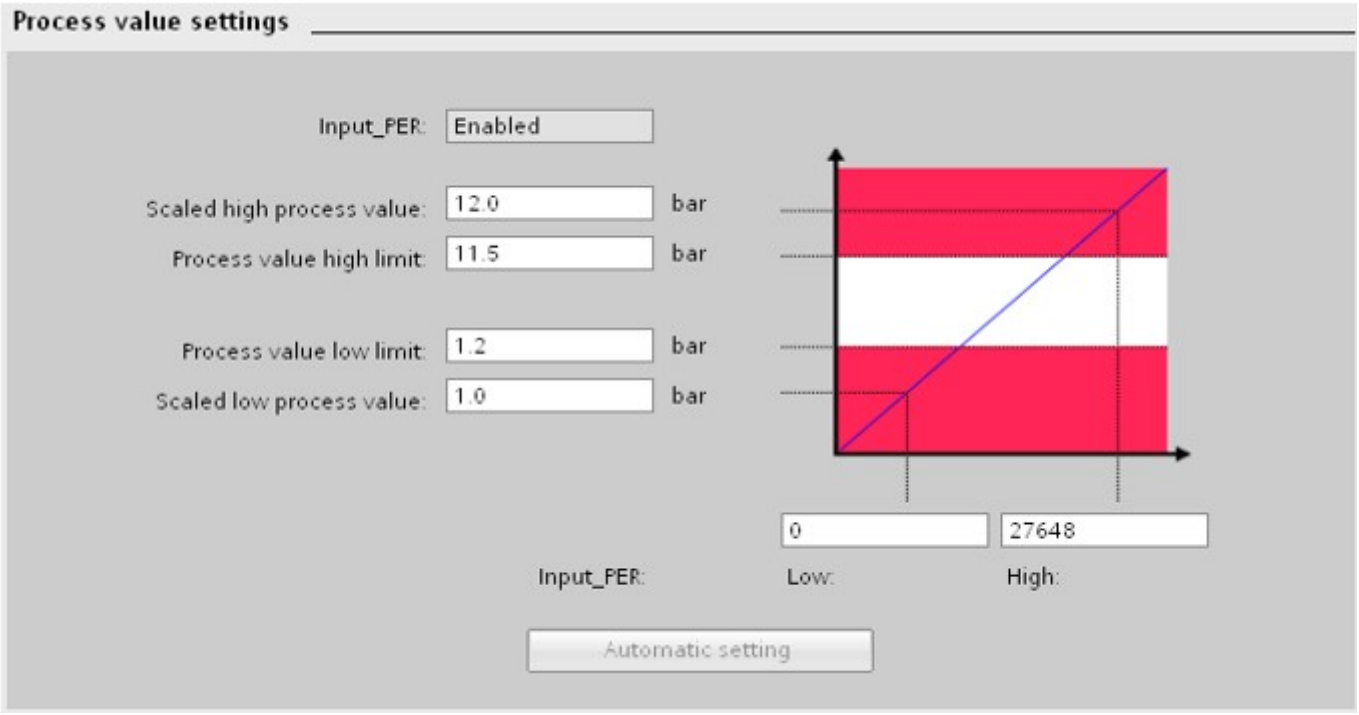
Compact PID controller

The compact Siemens PID controller is limited compared to the continuous PID controller in its possibilities. Most of the parameters can be set via associated pop-up menus and can therefore only be set using a programming device (PC with TIA Portal).

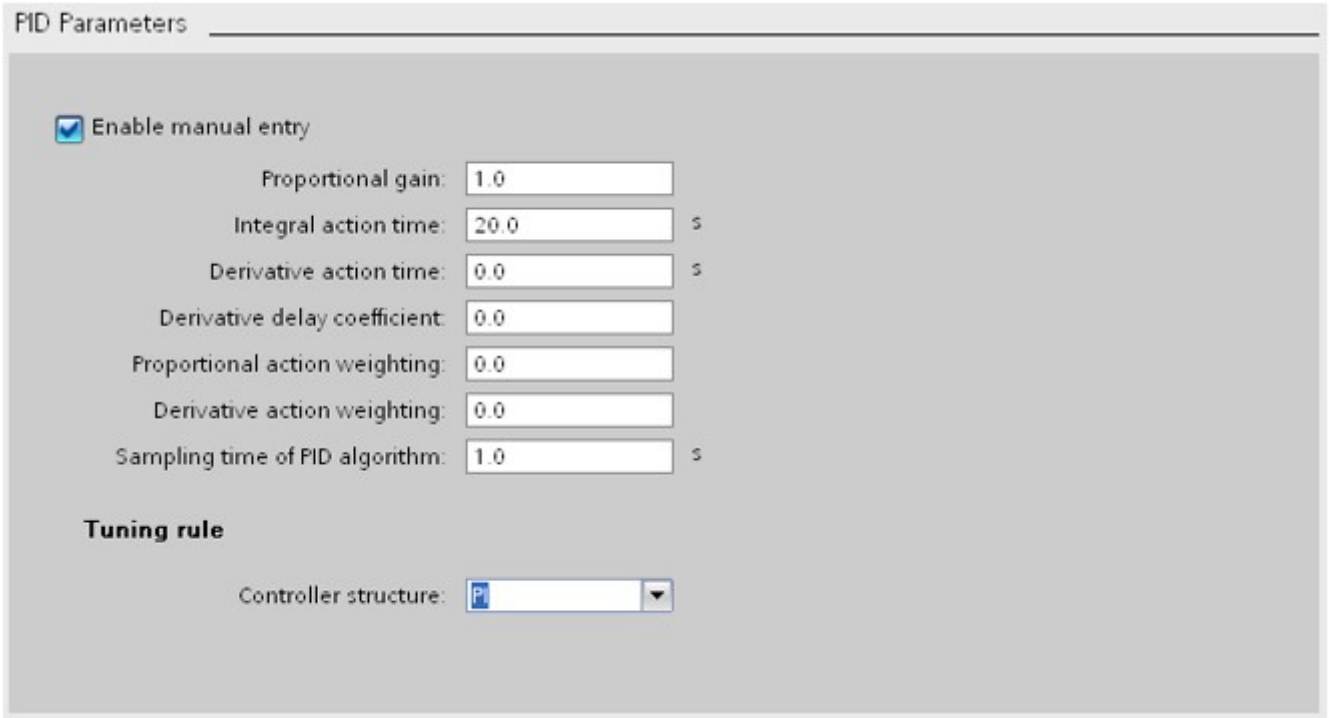
As an advantage it can be quoted that configuring this controller is done by using these pop-up screens which is much easier. This way one can make a choice between the different quantities and different SI units and one can opt to operate via a normalized input (output) or a periphery (entrance exit).



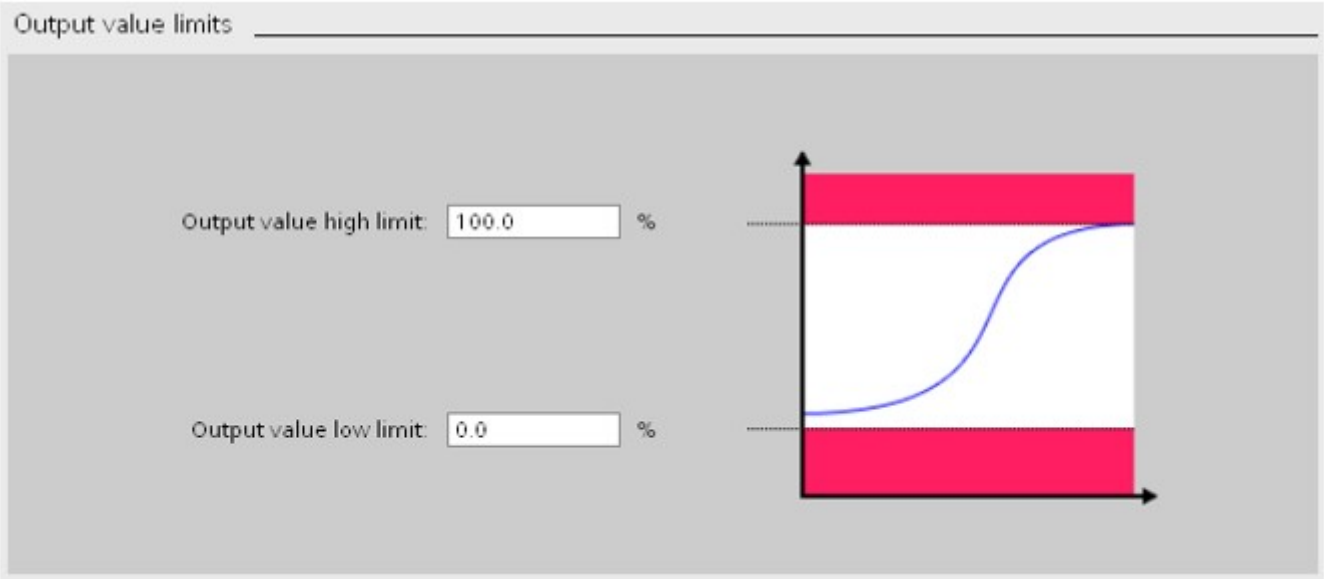
If you opt for a peripheral input, you can convert it to the correct measuring range using the screen *“Process value settings”*.



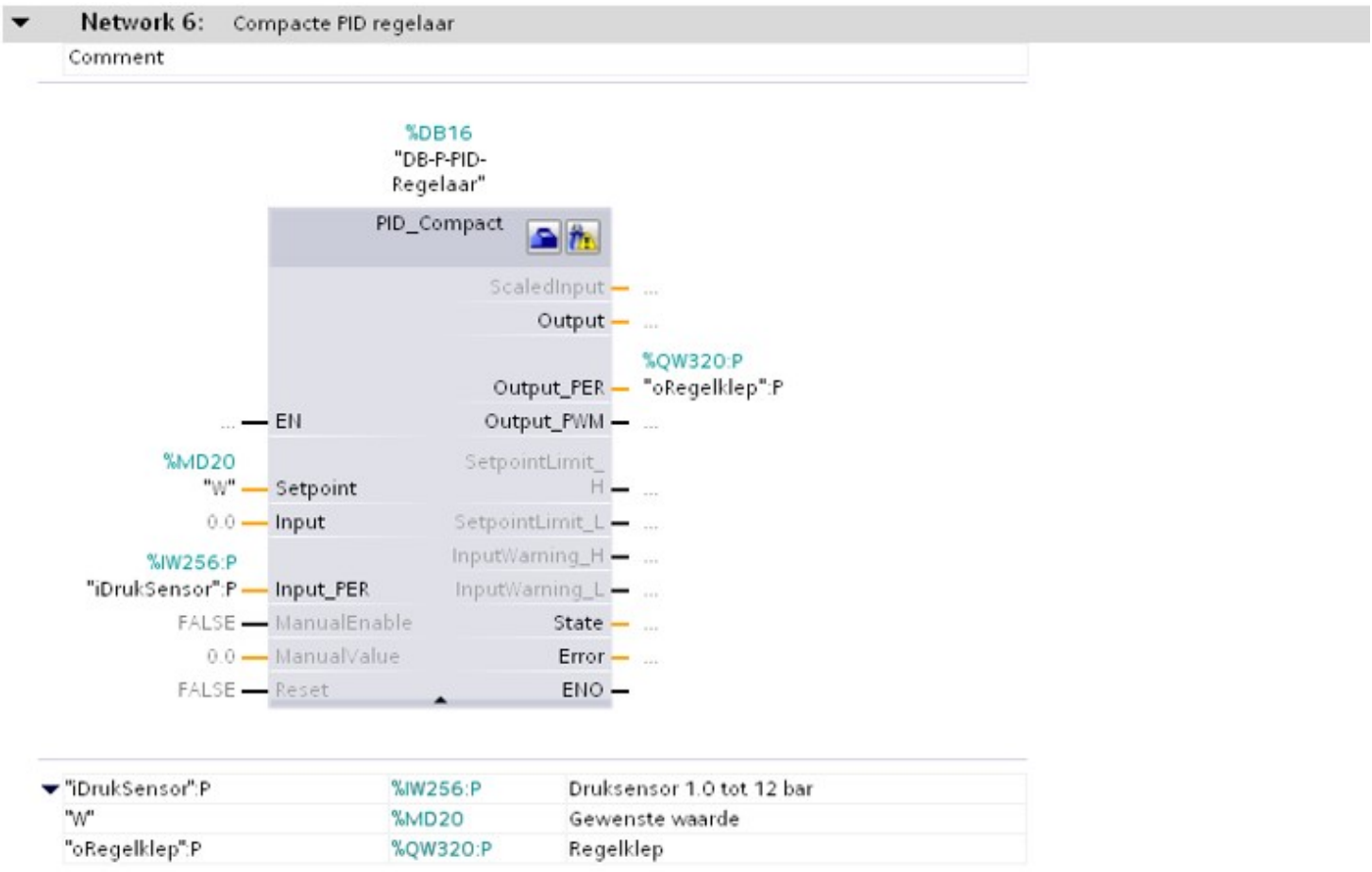
The P, I and D action can be set via the screen *“PID Parameters”*. Notice that the different actions cannot be selected individually. One can only make a choice between a PID and a PI controller. Note that every action is separately correctable. For example, one can calculate the weight of the P action and the D-action can be individually adjusted to even turn off and one can assign a wait coefficient for the D action.



The output can be limited just like a continue controller. This can be done in the *"Output value limits"* screen.



Because of the different parameters being adjustable by using pop-up screens, the PID building block will be compacter than the continuing Siemens PID controller.



Programming example – Continue PID controller

In the programming example below will you find the programming of a continue PID controller.

The analog pressure sensor %IW256 gets formed internally so that the following results are obtained:

- PV_NORM = (output CRP_IN) . PV_FAC + PV_OFF
- PV_NORM = (output CRP_IN) . 0,11 + 1.0
- 1.0 bar = (0%) . 0,11 + 1.0
- 12.0 bar = (100%) . 0,11 + 1.0

The loop manipulated value is variable adjustable suing %MD20. This way we can adjust the loop manipulated value with changing the user program.

The dead band is adjustable at 1.2 bar. In case (W - XDo/2) < X < (W + XDo /2)
The output of the controller doesn't change.

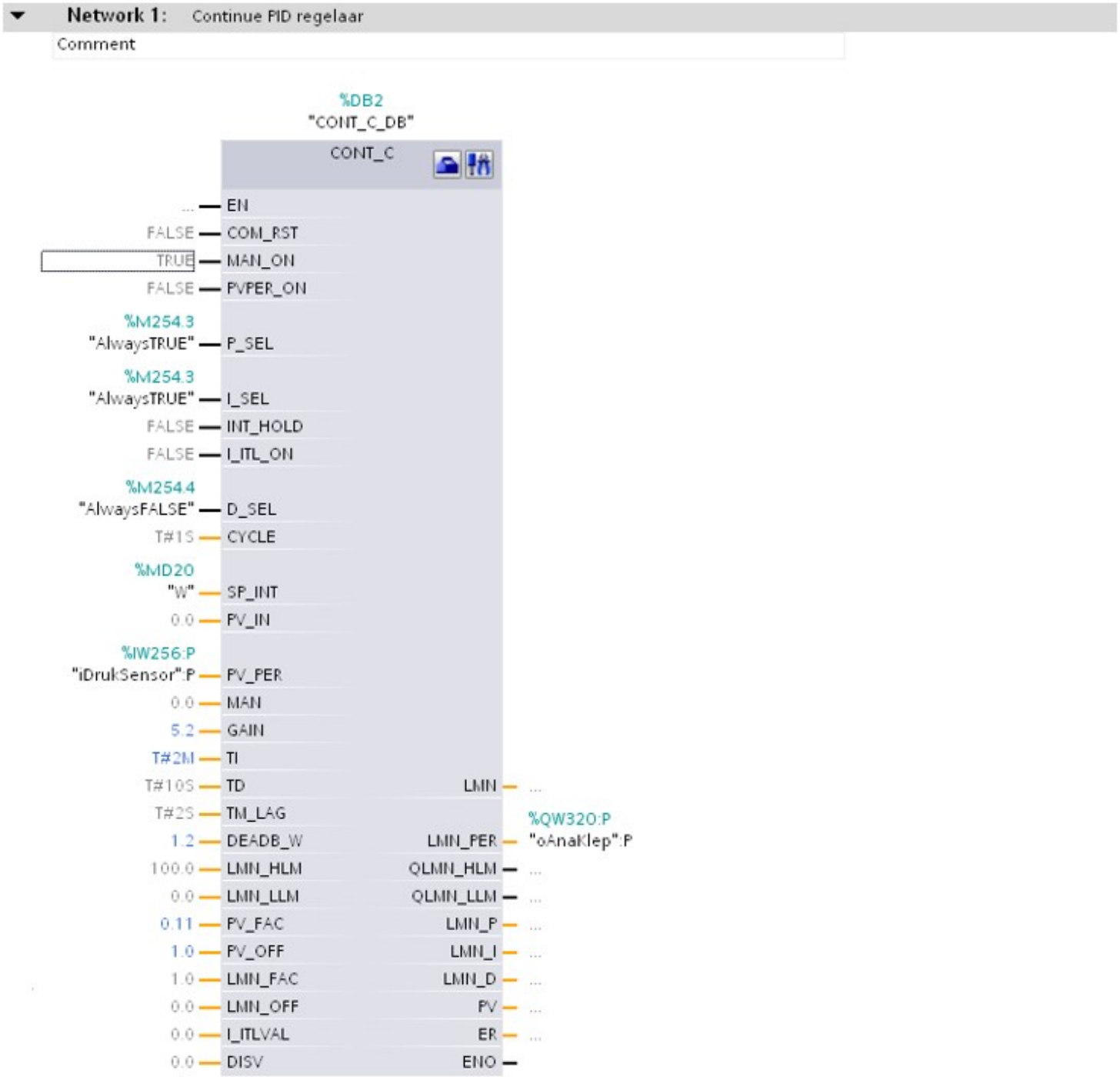
The controller is adjustable so that:

- The P-action is activated and the gain is set to 5.2
- The I-action is activated and the reset time is set to 2 min.
- The D-action is deactivated

The output is limited between 0 and 100+ (standard settings) and transformed to the peripheral output %PQW320.

Notice that the values are shown in light grey colour which are the standard values.

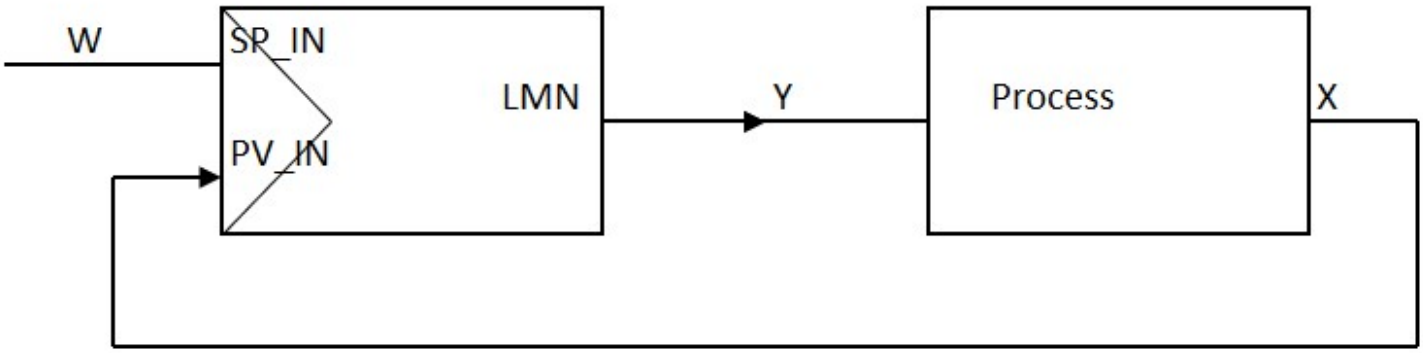
The PID buildblock gets saved by TIA PORTAL in the folder“Program Blocks\System Blocks\Program Resources”



Controller cicruit structuring

Singular control circuit

This is the simplest control circuit. The controller keeps the controlled unit [X] stable on the set value [W]

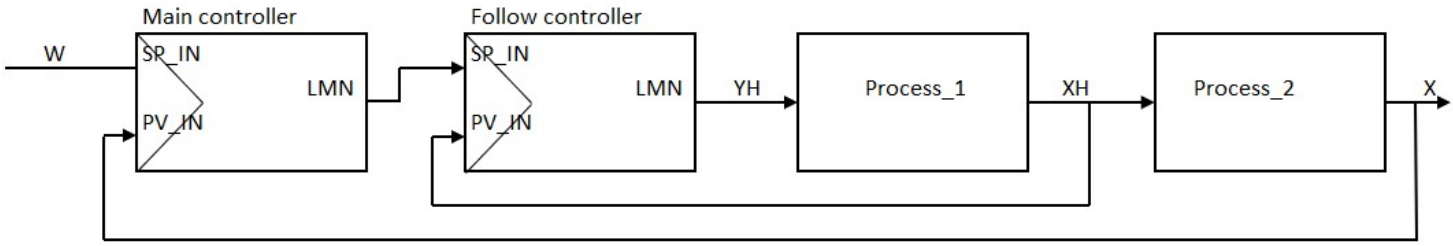


Singular feedback control circuits are commonly used where the influence of the controldynamic and the control result are minor.

Cascade controller or master/slave controller

The imperfections of a single control loop are mainly improved by a cascade control. In cascade control, the control loop becomes divided into a main control loop and an auxiliary control loop. For this there is at least one main control controller (master) and one auxiliary or follower control (slave) required.

- The main or control controller regulates the main control variable to the desired one value [W]
- The main or control controller returns an analog SI unit [Yf] which is processed by the auxiliary or follow-up controller as the desired value [Yf = Wh]
- The result of the auxiliary or tracking controller [Yh] influences the process of the analog measurement of the main or control controller



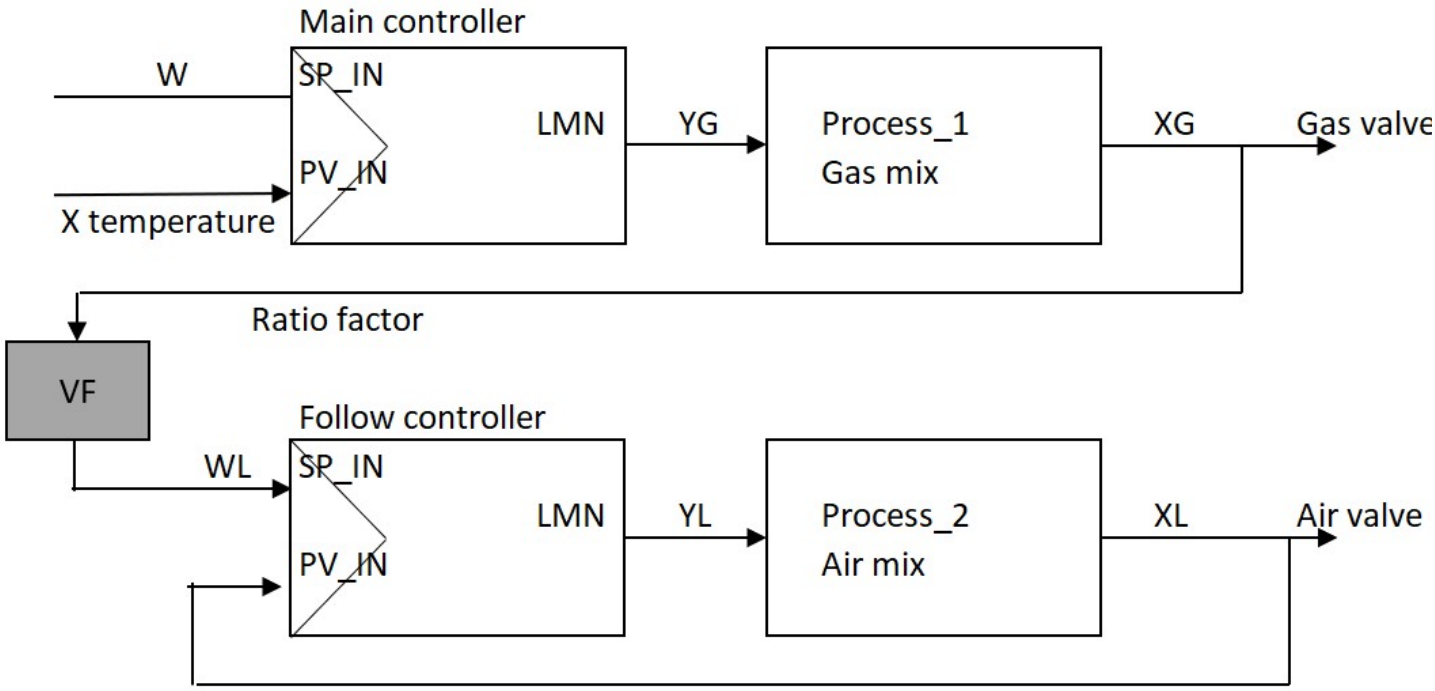
Depending on the needs a cascade controller can be equipped with multiple help controllers. Consequently you could place multiple help/follow controllers behind a head controller.

Ratio controller

The ratio controller has just like a cascade controller a head controller and a help- or follow controller. The intention is to have multiple processunits in a constant ratio. The ratio controller gets used mostly

for controlling 2 flow streams, between these 2 flow streams a determined ratio needs to be present.

The simplest example of a ratio controller is for example the gas and air supply in a gas incinerator. The head controller controls the amount of gas, depending on the desired oven temperature. The help or follow controller gets controlled by the actual value of the head controller which then controls the amount of air.



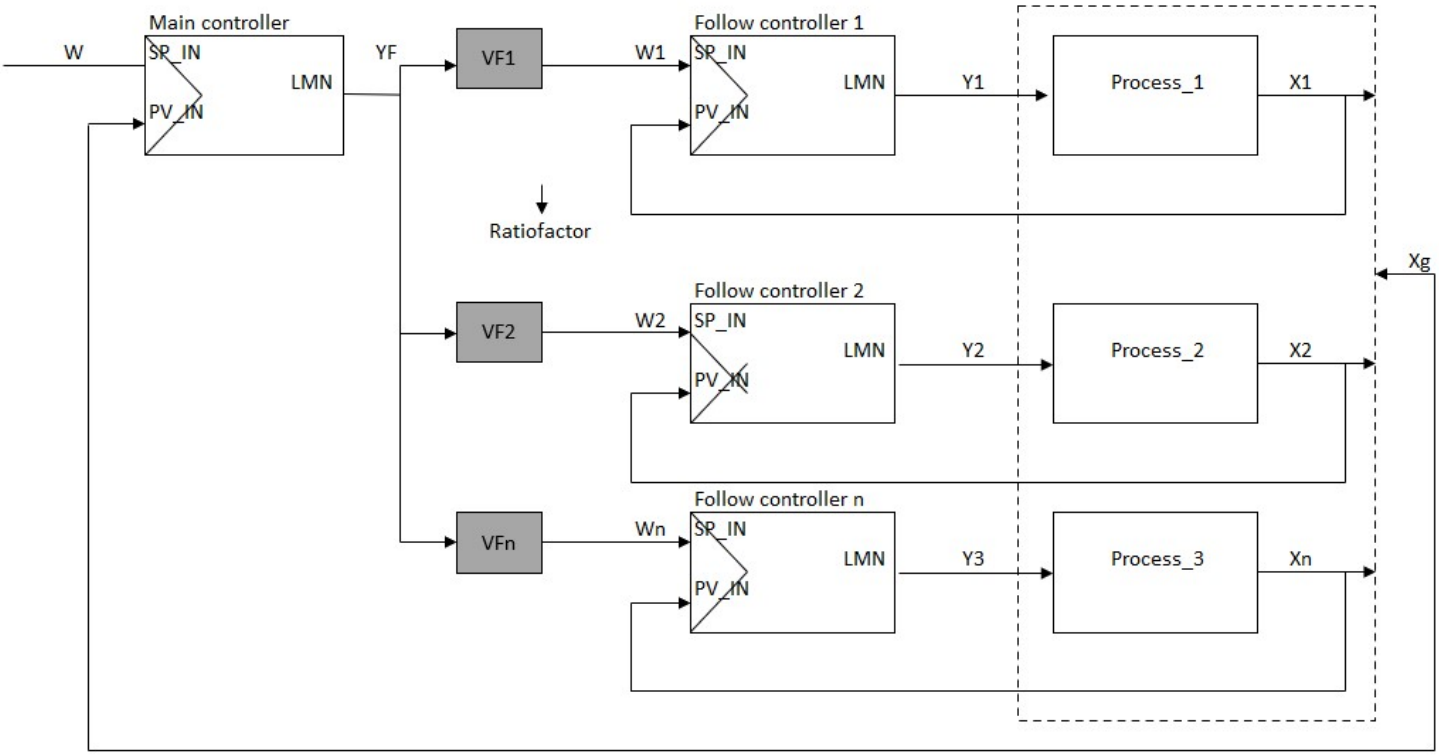
The ratio between both SI units, gas and air gets used with a ratio factor on the setpoint for the control or follow controller.

Mix Ratio Controller

A mix ratio controller is a ratio controller with a main controller and several subordinate auxiliary or follow-up controllers.

With the mixing ratio control it is possible to make a product from several basic components consists of mixing [X1, X2,… .Xn] into a final product with a constant mixing ratio.

The main- or control-controller controls the joint composition [Xg] it controls all subordinate component controllers with its control output [Yf].The percentage share of each component [X1, X2 ..Xn] with respect to the joint mixing ratio [Xg] is entered with the ratio factor "for".



Split range controller

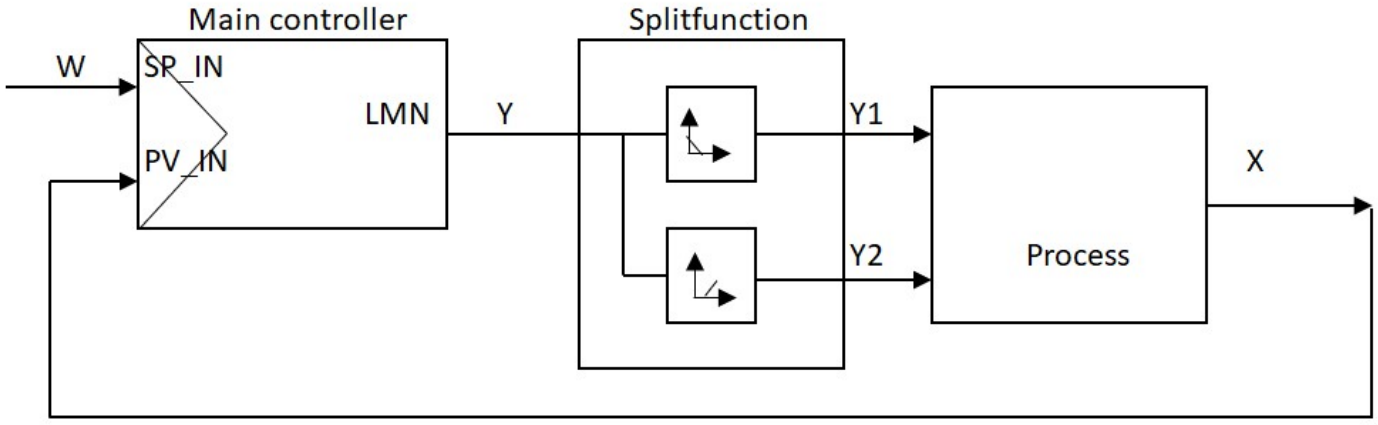
Some applications need multiple ajustment ratios, that can be achieved with only one adjusting device, for example a controlvalve.

A split range controller is a controller with one controlling SI unit and multiple controlled SI units.

The controlling unit divides its actions over for example two adjustment devices.

Split range controllers get used a lot in systems for heating and cooling.

In case the controlled variable varies over a big range, will it be useful for applications.



The controller output gets split in parallel paths, each with an adjusting device.

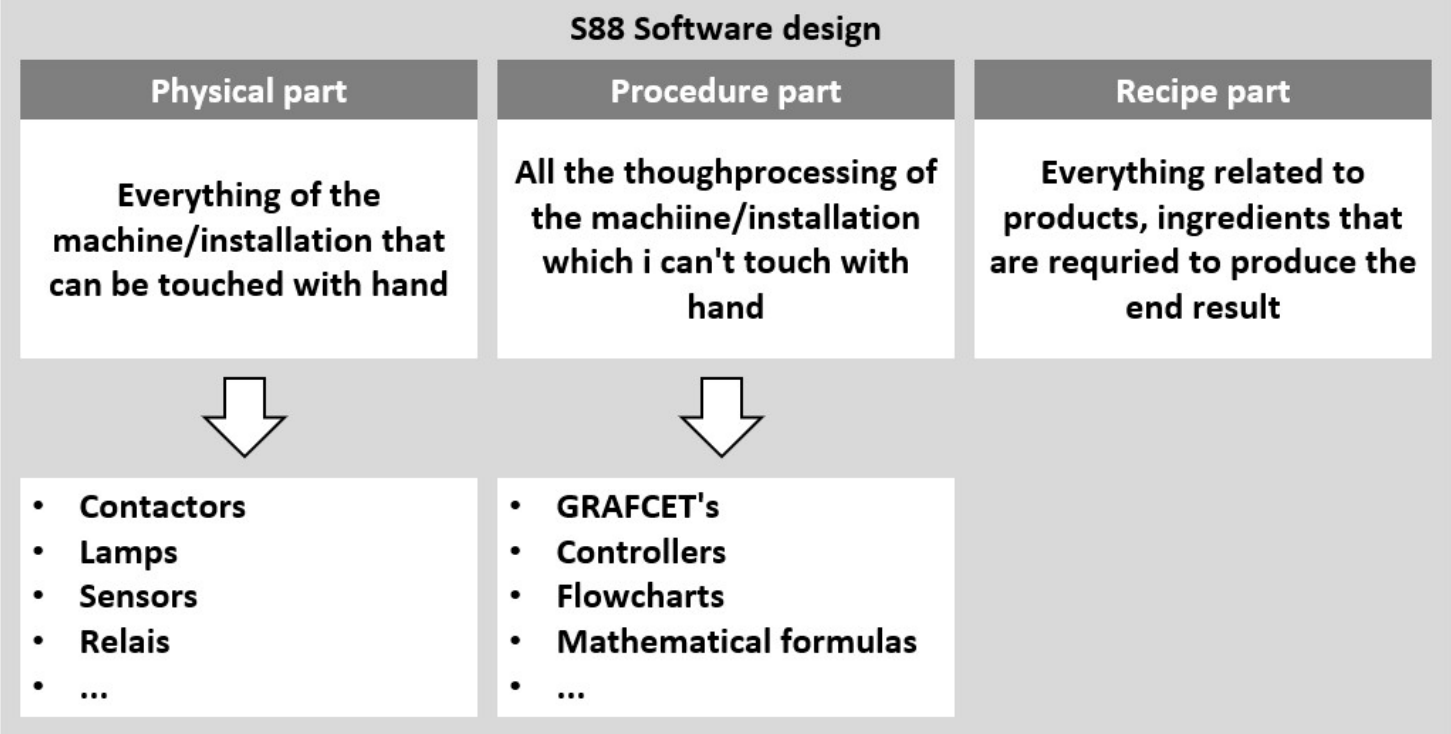
Software model following ANSI-ISA-88

The different parts

The ANSI/ISA-88 norm or the **S88 software model** is a norm that describes how a machine/installation (batch)process can be subdivided in different parts.

The advantage of this is that one big problem[3] will be divided in different smaller partial problems; smaller problems are often easier to solve than bigger problems. A strategy will be developed for each small partial problem that will cause the bigger problems to be solved one by one.

- The physical part
- The procedure part
- The recipe part



Because the S88 software model is very abstract and expanded we will be using a very simple form in this course:

- The back spine is the physical part
- The procedure part gets integrated in the physical part.
- The recipe part won't be applied

This means that the software building blocks only get designed for the processing of the physic part or only the procedure part.
The building blocks will exhange information between each other.

The following chapters describe different building blocks that are included in the software library. The operation of each individual building block is explained with the use of a operation scheme with the following symbols:

Symbol	Description
	AND port
	OR Port
	NOT connection
	Connection
	Risedelay
	Drop-off delay
	Time Puls
	A collection of instructions that together a combination basiccircuit form (in this case the start-stop circuit)
	Positive flank signal
	Negative flank signal

The operation scheme gets drawn up so that every incoming signal is as far to the left as possible and all output signals are to the right. Connections are when needed drawn with dotted lines (with crossing lines) to avoid confusion.

Physical part - Control modules

Control modules are software blocks that
- Process sensor input singals (%I)
- Activate / control output signals (%Q)

This way a control module gets represented by a certain type of actuator or sensor and by preference gets included in the software library.

Control modules are preferably progammed in "Function buildblocks" whereby, the TAG-naming gets expanded with the letters CM.

Sensors

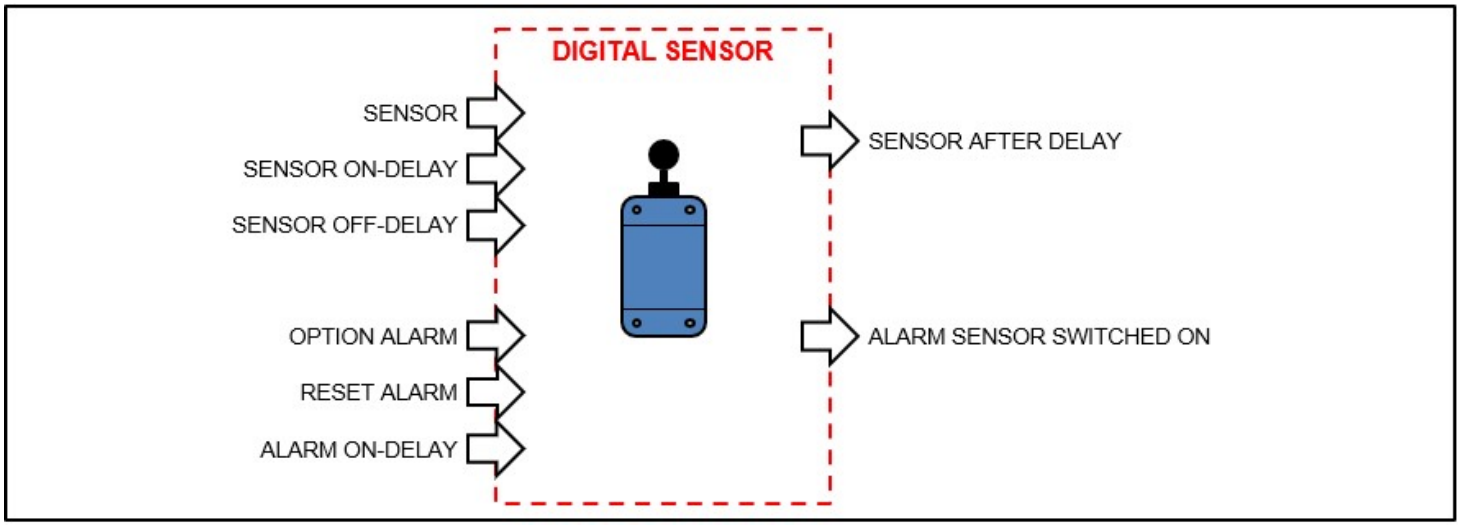
Digital sensor

A **digital sensor** (Ex.:limit switch, inductive sensor, capacitive sensor, tuning fork,...) processes mainly the presence of a product, object, person, etc. It has 2 conditions which indicate whether these products, objects or persons are preset or not.

Functionally seen these sensors monitor:

- The correct automatic process whether or not with the needed delay (example, opening of a door with the help of a photocell after which the door stays open for a bit)
- The protection against defects (example, overflow protection)

One can design a building block wherein all the above functionalities are processed. As a Consequently, this building block is the software representative of a digital sensor.

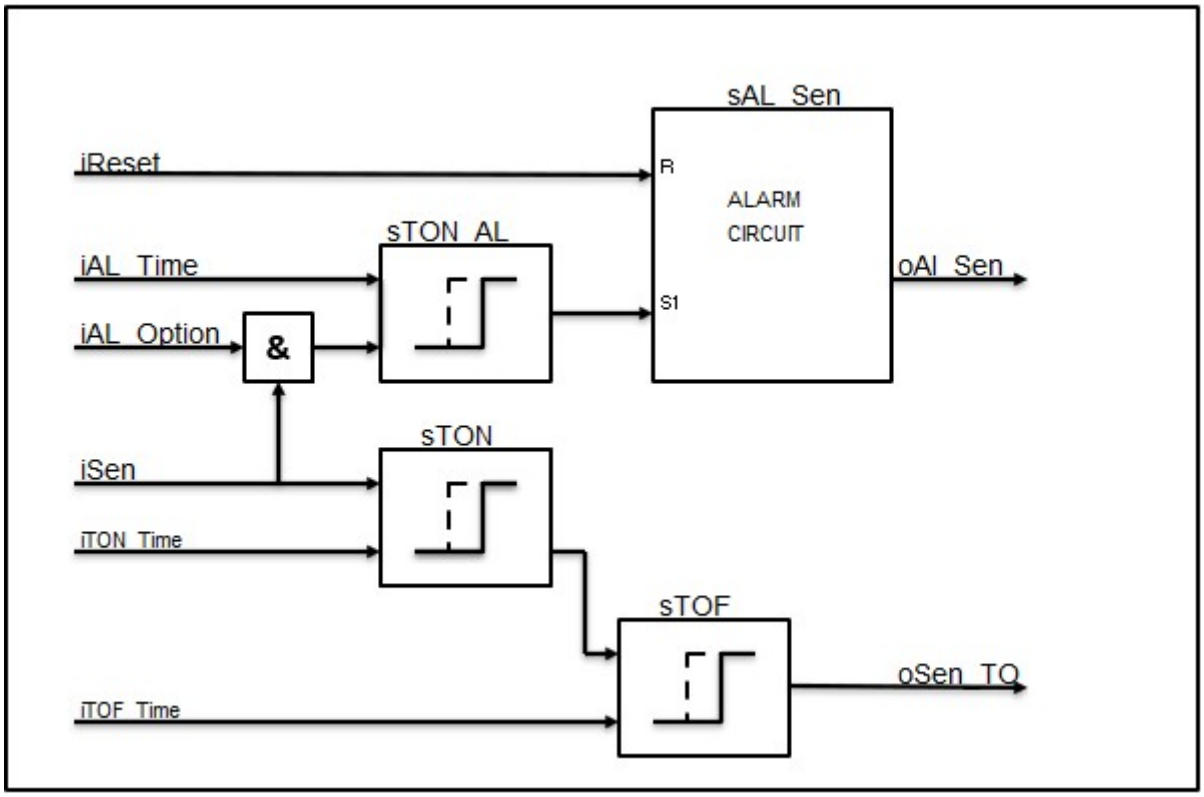


This building block belongs to the group of **control modules** are provided with following functionalities:

- The module will ensure that the sensor signal (iSen) which optionally can be delayed with a on-delay(iTON_Time) and/or a off-delay (iTOF_Time) for a correct operation, which then provides the output signal (oSen_TO)

- In case the option alarm (iAL_Option) is enabled, the module will activate an alarm(ioAL_Sen) as soon as the sensor (iSen) is disabled, taking into account the off-delay(iAL_Time).
- If the sensor(iSen) isn't switched on anymore, the module will turn off the alarm(ioAL_Sen) as soon as it gets reset(iReset)

It is possible with the description to draft an operation scheme for the control module with the name FB_CM_DI_Sensor.



This results into a **"Function buildblock"** which looks like the following images.

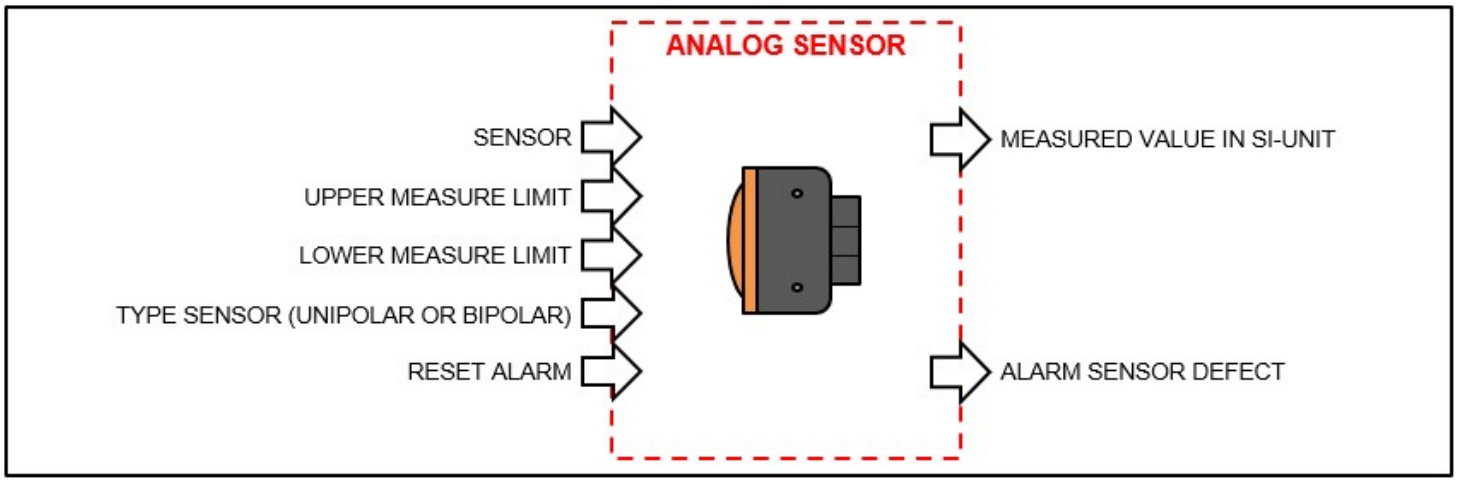
Text	Image
FDB example	
More simple example	

Analog sensors

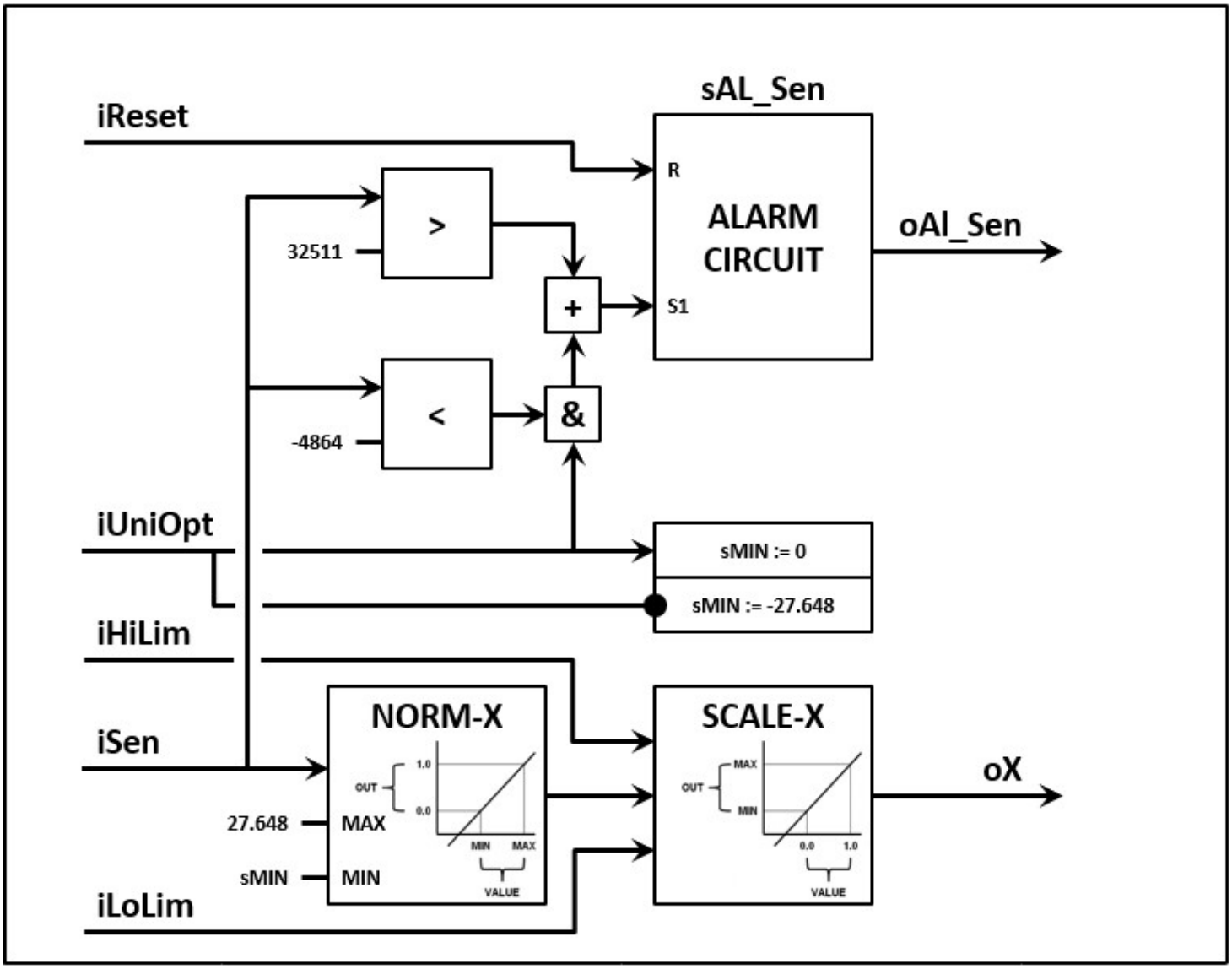
It is possible to process for example: the current level of a liquid with an **analog sensor** in a programmable manager circuit ("sturing"). It's needed to convert the number given to the correct SI-unit, this is because the current value isn't given in the correct SI-unit (ex. : mm). A unipolar sensor is only capable of converting a positive signal (ex. 0 .. 20mA, 4 .. 20mA, 0 .. 10 V, ...). A bipolar sensor processes both positive and negative signals (ex. -10V .. + 10 V).

A **control module** for this type of sensors has the following tasks:

- Converting the provided number (iSen) which is a mirror of the measured value, to a SI-unit (oX) whereby taking into account the unipolar and bipolar signals (iUniOption) and the max. (iHiLim) and min. (iLoLim) measure range of the analog sensor.
- Making an alarm (ioAL_Sen) in case of an abnormal situation (ex. Cable break or overcurrent)
- If the abnormal situation is fixed will the module will turn off the alarm as soon as it gets resetted (iReset)



It is possible with the description to draft an operation scheme for the control module with the name FB_CM_AI_Sensor.



Notice that the operation scheme is drafted for analog sensors that work following the Siemens principle, they have a normal range of 0 .. 27.648 / -27648 .. + 27648.

The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

Examples

Tag	Processing of a digital sensor
FB_CM_DI_Sensor	Processing of a digital sensor
FB_CM_AI_Sensor	Processing of an analog sensor

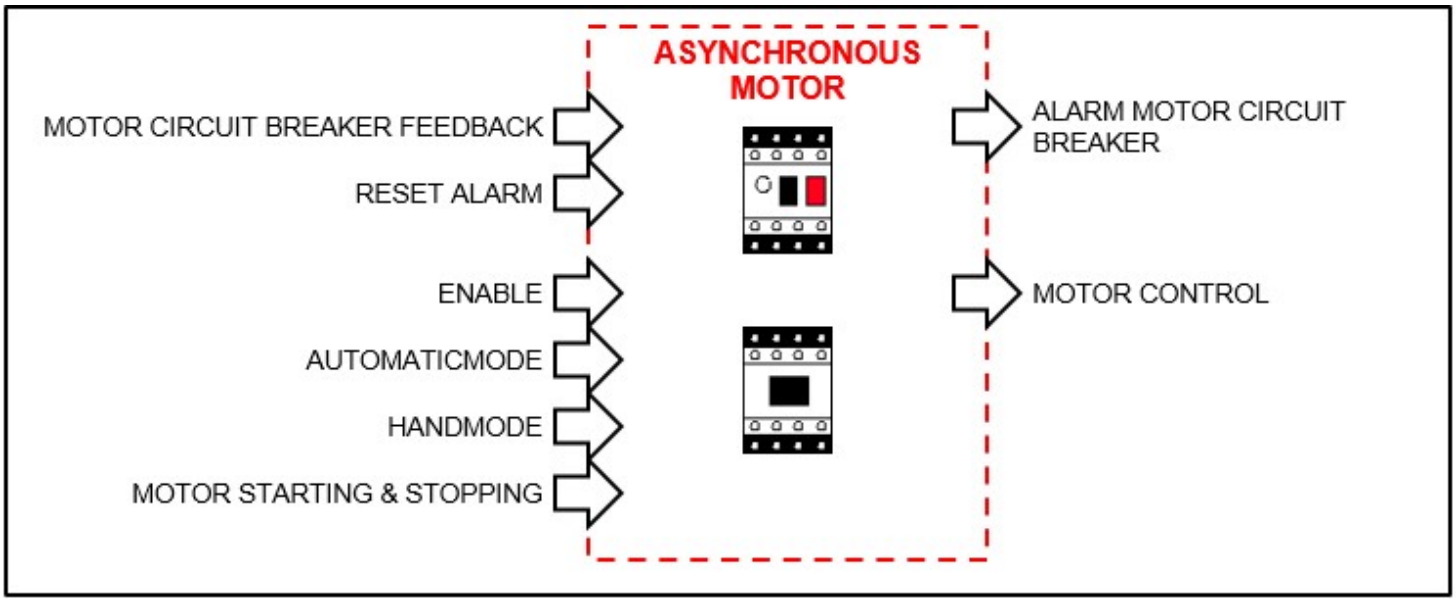
Tag	Processing of a digital sensor
FB_CM_DOL	Controlling an asynchronous motor with 1 speed and 1 rotating direction
FB_CM_DOLRev	Controlling an asynchronous motor with 1 speed and 2 rotating directions
FB_CM_Valve	Controlling of a (pressurised air) valve
FB_CM_Relay	Controlling of a relay
FB_CM_Lamp	Controlling of a (LED) lamp

Asynchronous motors

Asynchronous motor with set speed and one turn direction

With the current technology an asynchronous motor that only runs forward with a set speed can't operate without:

- 1x motor circuit breaker
- 1x relay



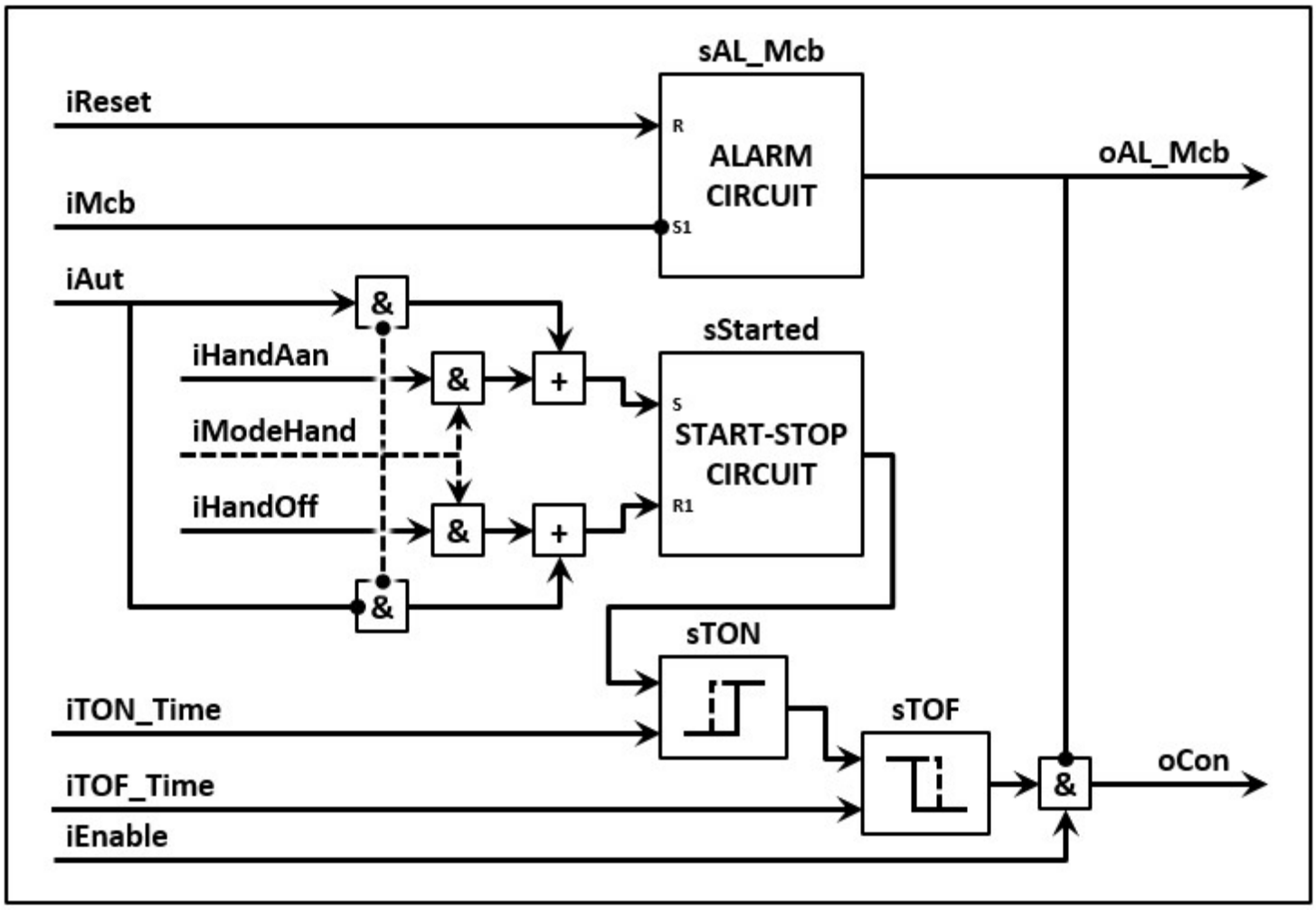
A **control module** for this type actuator is inseperable connected with a motor circuit breaker and a relay. The control module shortly does the following:

- If the think process(iAut) asks that if the motor needs to run, the control module will let the motor run (oCon)
- But if the motor circuit breaker (iMcb) is turned off the module won't let the motor run en it'll activate the alarm (ioAL_Mcb)
- The motor will only start running again if the motor circuit breaker has been activated and the alarm reset has been reset(iReset)

One can expand the functionalities of the control module:

- The motor will only run (oCon) if the module is enable (iEnable)
- If manual mode is activated (iModeHand), the module will ignore the request from the think process (iAut) and runs the motor whenever the manual signals are given (iHandOn & iHandOut)
- If the motor needs to run (oCon) the module can start the motor with an on delay (iTON_Time) and/or an off delay(iTOF_Time)

It is possible with the description to draft an operation scheme for the control module with the name FB_CM_DOL.



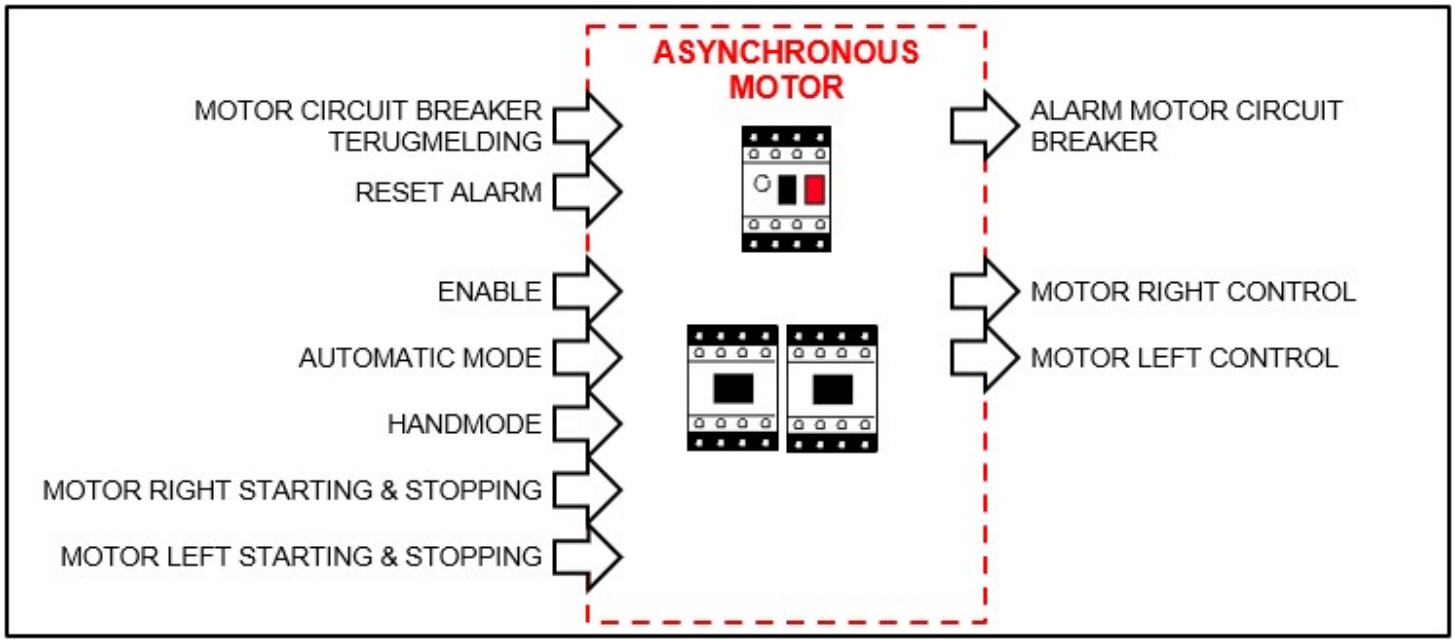
The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

Asynchronous motor with set speed and two turn directions

With the current technology an asynchronous motor that runs forward and backwards with a set speed can't operate without:

- 1x motor circuit breaker
- 2x relays



A **control module** for this type of actuator is inseperably connected with a motor circuit breaker and a relay. The control module, in short does the following:

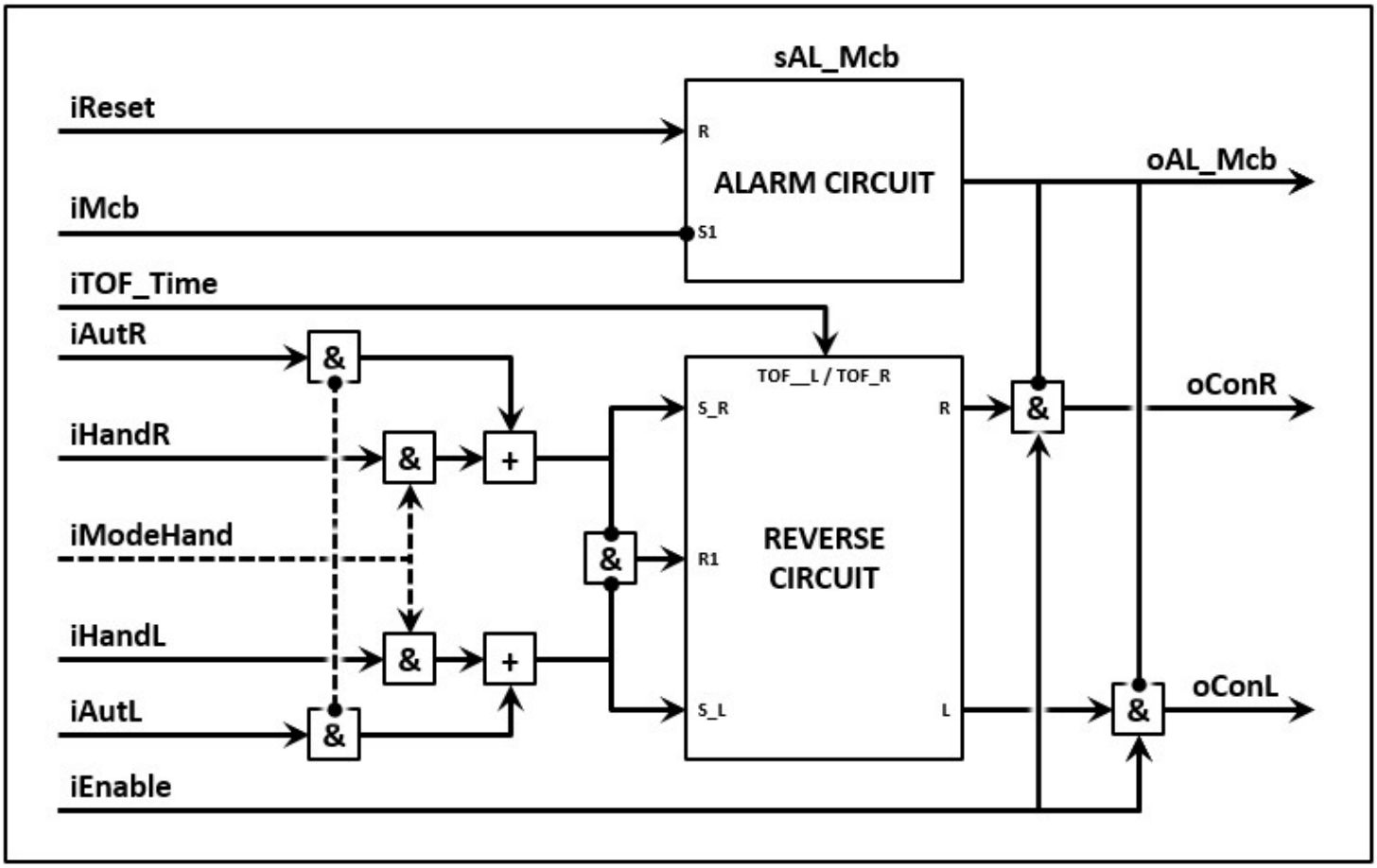
- If the think process(iAutL) asks if the motor needs to run left, the control module will let the motor run left (oConL)
- If the think process(iAutR) asks if the motor needs to run right, the control module will let the motor run right (oConR)
- It is possible to run through a waiting time(iTOF_Time) if it is requested to change direction
- But if both directions are requested (left and right) the direction doesn't change and the motor will keep spinning the way it was before the request.

- But if the motor circuit breaker (iMcb) is turned off the module won't let the motor run en it'll activate the alarm (ioAL_Mcb)
- The motor will only start running again if the motor circuit breaker has been activated and the alarm reset has been reset(iReset)

One can expand the functionalities of the control module:

- The motor will only run (oConR & oConL) if the module is enable (iEnable)
- If manual mode is activated (iModeHand), the module will ignore the request from the think process (iAutR & iAutL) and runs the motor to the right(oConR) whenever the manual signal is given (iHandR)
- If manual mode is activated (iModeHand), the module will ignore the request from the think process (iAutR & iAutL) and runs the motor to the left(oConL) whenever the manual signal is given (iHandL)
- The control module won't change running condition if the mode changes from automatic mode (NOT iModeHand) to hand mode (iModeHand)

It is possible with the description to draft an operation scheme for the control module with the name FB_CM_DOLRev



The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

Valve

A valve is used to power compressed air or hydraulic actuators (ex. compressed air cylinder). The combination of a valve and actuator has a specific functionality:

- Monostable power circuit: In case there is a loss of power the actuator will automatically take its residual state (ex. single-acting compressed air cylinder)
- Bistable power circuit: In case there is a loss of power the actuator will keep the last position (ex. double-acting compressed air cylinder)
- Monostable control circuit: In case there is a loss of power in the control circuit the valve will automatically take its residual state (ex. 3/2 valve with spring return)
- Bistable control circuit: In case there is a loss of power in the control circuit the valve will automatically keep its last state (ex. 5/2 valve with electromagnetic control on both sides)

A **control module** for a valve will be built without taking into account the already existing functions of the combination valve - actuator. The control module will work following the bistable process.

Why would you use a monostable valve and cylinder?

During the design of a machine/installation one needs to ask themselves: What can go wrong during an unexpected event? How do i take care of potential risk to a human? Following the machine guidelines, no moving part of the machine or no single object that is held by the machine can be dropped or ejected. (2006/42/EG,2006)

Unexpected events like for example a fire, an accident, an earthquake, a flood, etc. can cause interruptions in the control- and/or power circuits. The interruption of these circuits can cause dangerous situations.

Example

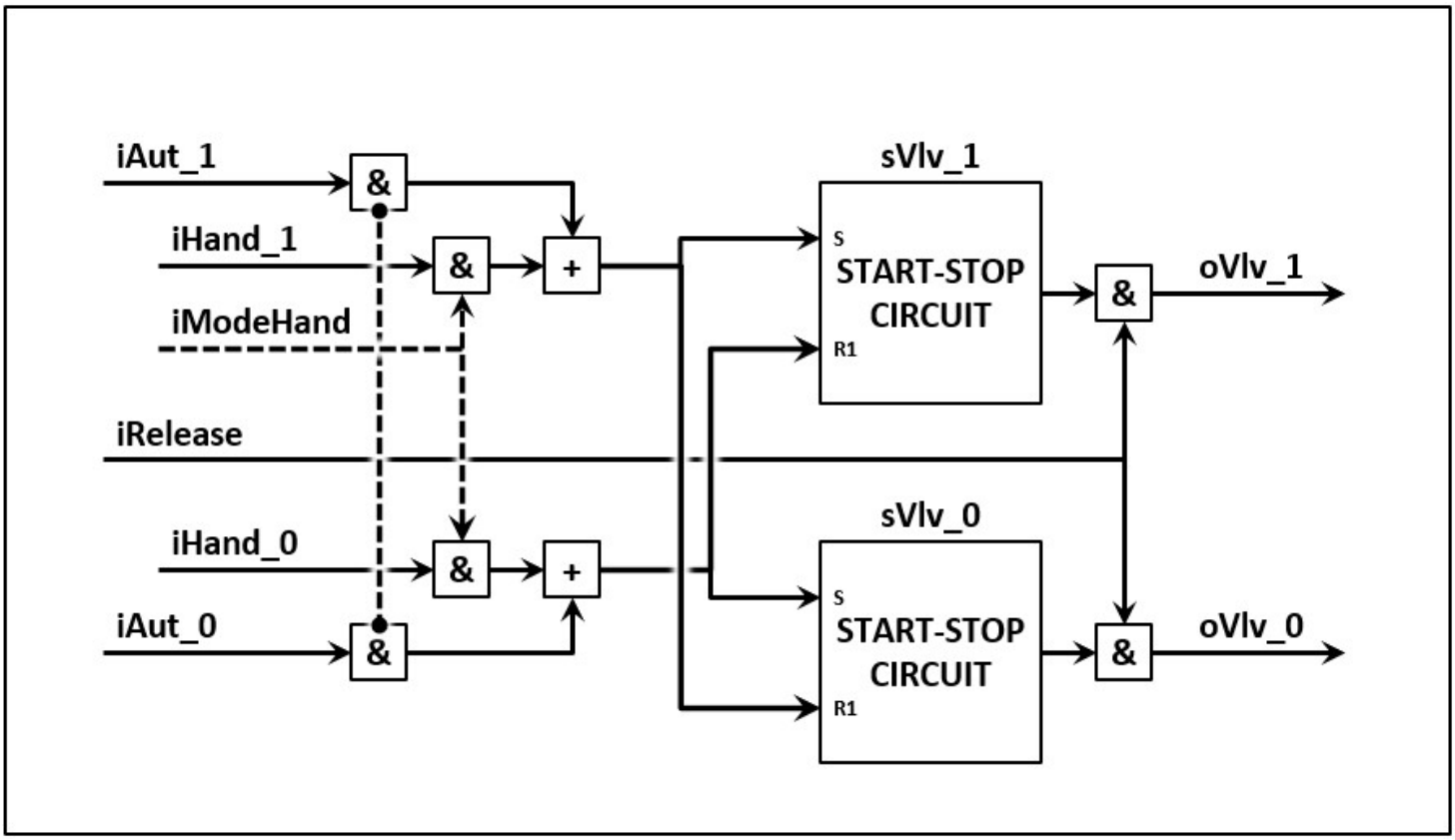
Because of a fire in a technical room, the air compressor stopped working which has the side effect that there is no more compressed air. A robot that moves part of +/- 2 kg on highspeed is equipped with a pneumatic grabber. There is the danger that in case the compressed air falls away on the moment the robot moves at high speed, that the robot lets go of the object and basically throws away the object. This has the potential to hurt humans that work near the robot.

In this situation one chooses a monostable compressed air cylinder with a spring return, this will keep the grabber "grijper" to remain closed in case the compressed air falls away.

The functionality of this type control module:

- The control module will work on the following a bistable principle;
- - In case the think process asks to activate the "+" side of the valve (iAut_1) the control module will do this (oVlv_1) and it will keep the condition in case the think process doesn't ask for it anymore, until the "-" side gets activated
- - In case the thinkprocess asks to activate the "-" side of the valve (iAut_0) the control module will do this (oVlv_0) and it will keep the condition in case the think process doesn't ask for it anymore, until the "+" side gets activated
- The control module will only change the condition of the electromagnetic control in case this is allowed (iEnable)
- If the hand mode is active (iModeHand) the control module ignores the think process signals (iAut_1 & iAut_0) and sends the valve (oVlv_1 & oVlv_0) commands based off the hand signals (iHand_1 & iHand_0)

It is possible with the description to draft a operation scheme for the control module with the name FB_CM_Valve



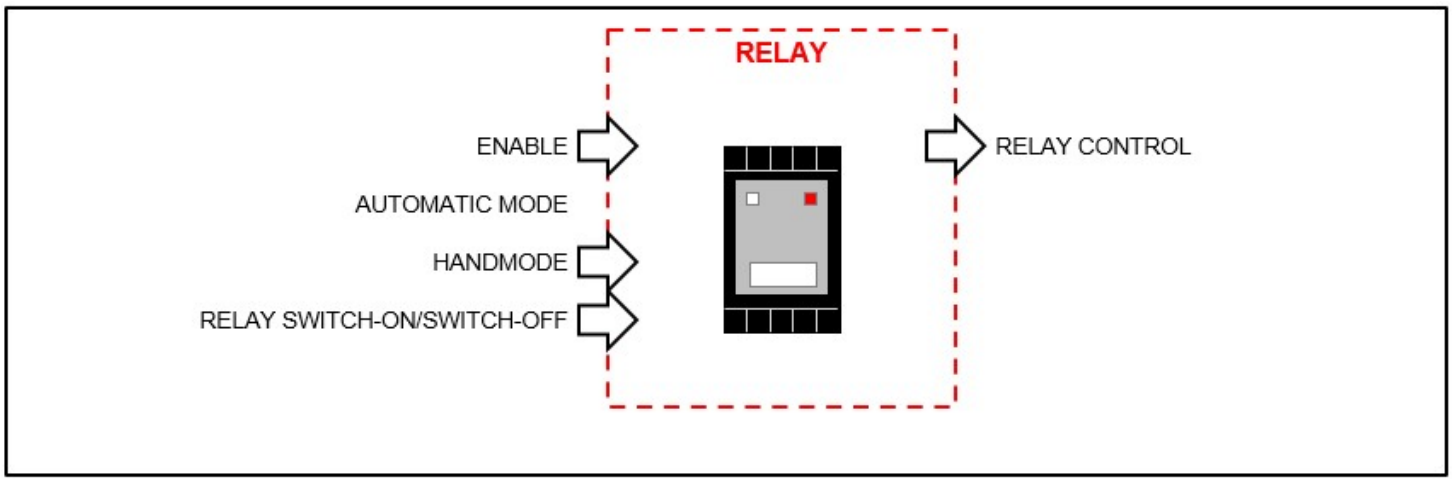
The endresult is a **"Function buildblock"** which looks like the following images.

Text	Image
FDB example	
More simple example	

Relay

A **relay** is often used to build up communication with:

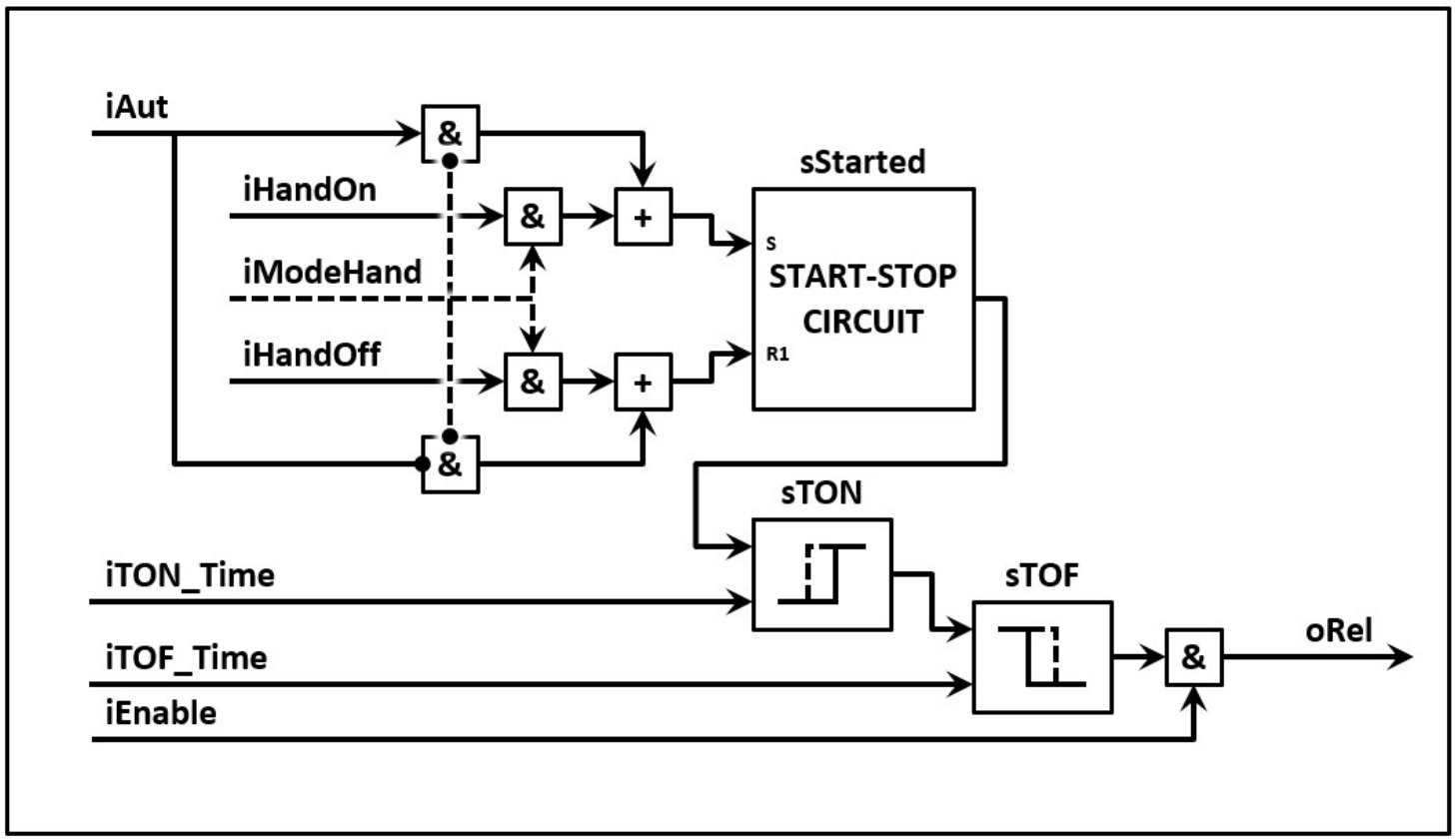
- Other PLC's via a potential-free contacts
- Other devices like frequency controllers



Just like an asynchronous motor one can assign multiple functionalities to the corresponding control module such as:

- If the think process asks to activate the relay (iAut), the relay will be activated (oRel)
- The relay will only be activated if this is enabled (iEnable)
- If hand mode (iModeHand) is activated then the module will ignore the think process's signal (iAut) and activates the relay based on the hand signals
- If it's asked to activate the relay, the module has the possibility to activate a rise delay (iTON_Time) and/or drop-out delay (iTOF_Time)

It is possible with the description to draft an operation scheme for the control module with the name FB_CM_Relay



The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

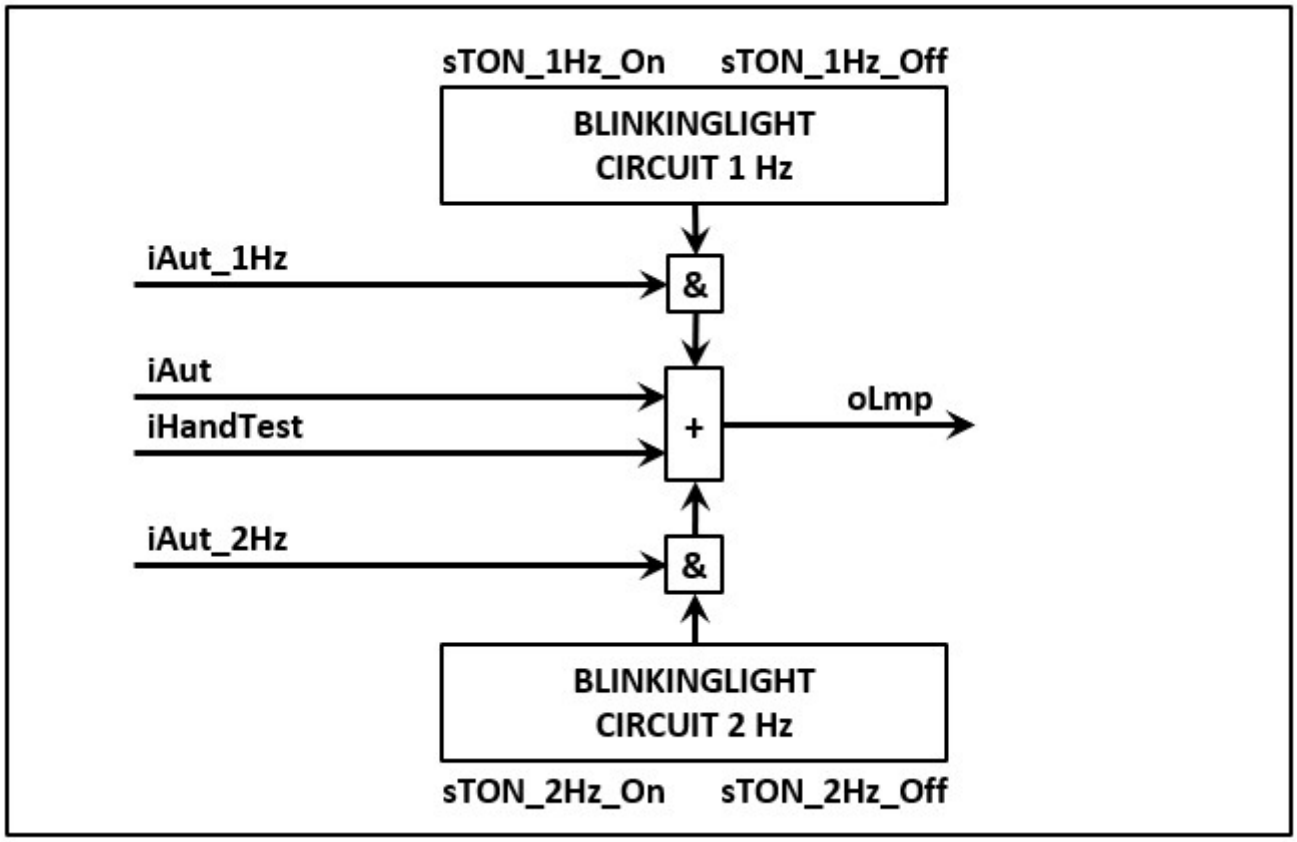
Lamp

A **lamp** is used to inform an operator about the status of a machine/installation or part of it.

A lamp control module handles following functionalities:

- Continued lighting of the lamp (iAut)
- Blinking of the lamp (iAaut_1Hz)
- Fast blinking of the lamp (iAut_2Hz)
- Activating the lamp if a lamp test(= controlling the lamps on defective lamps by maintenance technicians) is being executed (iHandTest)

It is possible with the description to draft an operation scheme for the control module with the name FB_CM_Lamp



The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

Physical part

Equipment modules

An **equipment module** is a collection of multiple control modules and/or other equipment modules. The collection is built upon the physical relationship they have with each other.
In other words, an equipment module is a software building block that has a minimum of 2 building blocks of the type control modules and/or equipment modules/.
Equipment modules are preferably programmed in "Functions", the TAG-naming gets expanded with the letters EM.

Examples	Description

Examples	Description
FC_EM_CC	Includes programming of the control cabinet
FC_EM_Airco	Includes programming of the airco installation
FC_T10_EM_Level	Includes programming of the level controlling of tank T10
FC_EM_X_Axis	Includes programming of the X axis

Procedure part

Procedure

A **procedure** is a strategy, think process to solve a problem. First, a strategy gets designed on paper using a certain method. Next the strategy will be translated to a software building block which we call the procedure element.

The designing of a strategy can be done with the following methods:
- By designing a GRAFCET drawing
- By designing a flowchart drawing
- By determining the needed mathematical formulas
- By drawing of an operation scheme
- By selecting a controller and determining the corresponding parameters in the shape of a table

Procedure element

A **procedure element** is the software translation(programming) of a procedure. A procedure drawing, scheme or table needs to be present for each procedure element. Some of these procedure elements are commonly used in machines/installations. This causes them to preferably be included into a software library. Other procedure elements get delivered by the producer of the processing unit:

- Start-stop procedure (software library) = To start and stop actuators, machines/installations in the correct way
- Reset procedure (software library) = To create a reset signal in the correct way
- Two-point controller (software library) = Controlling a digital sensor using an on-off controller with the help of an analog sensor
- PID-controller (software catalog Siemens) = To control an analog output

Procedure elements get preferably programmed in "Function building blocks", the TAG-naming gets expanded with the letters PE.

Examples	Description
FB_PE_StarStop	Start-stop procedure
FB_PE_Reset	Reset procedure

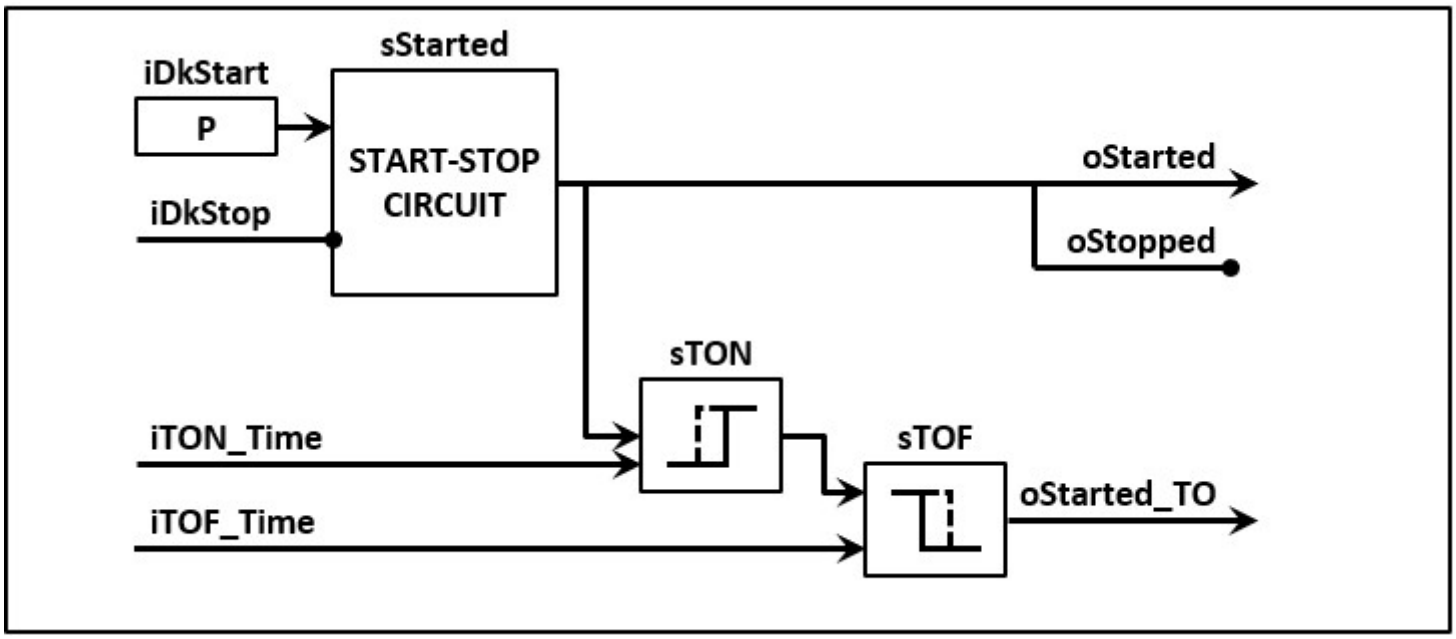
Start-stop procedure elements

A **start-stop procedure** is used to start or stop an actuator and/or the automatic process of a machine/installation (or parts of it). We use a classic start-stop circuit that is expanded with extra functionality.

Characteristics of the start-stop procedure:

- The stop action (iBtnStop) has priority on the start action (iBtnStart) which is mandatory following the machine guidelines
- The start signal is of the type NO-contact[4], the stop signal is of the type NC-contact[5]
- The operator is obligated to press the start button (iBtnStart) (Electrically bridging the start button isn't allowed)
- There is the possibility to start slower (iTON_Time) and/or stopping slower (iTOF_Time)

It is possible with the description to draft an operation scheme for the control module with the name FB_PE_StartStop



The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

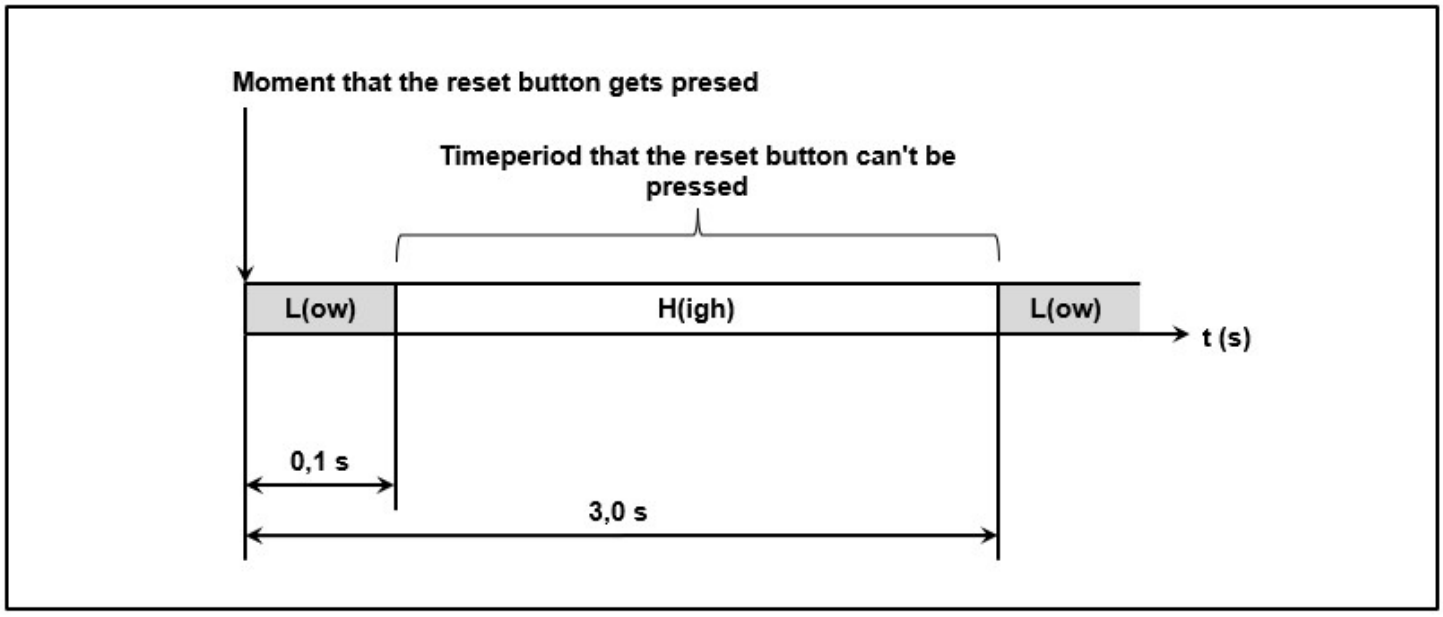
Reset procedure elements

The **reset procedure** gets used to create a checked reset signal. Checked because the signal coming from ex. an electrical reset button(NO-contact) can include issues like:

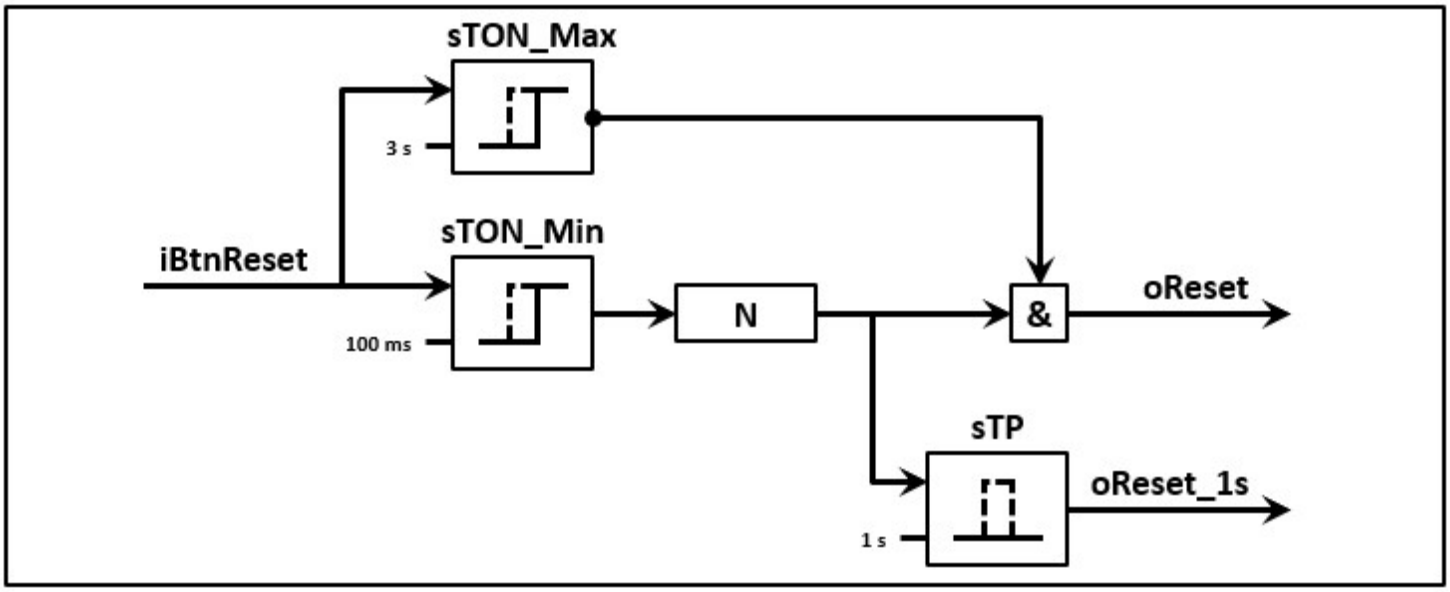
- An electrical circuit of the reset button can have a short circuit (it is like the button is being pressed the entire time)
- The reset button is electrically bridged (it is like the button is being pressed the entire time)
- Bad/fake contact in the electrical circuit of the reset button (it is possible by vibrations in the installation the contact on/off/on/off/on/off/.. switches)

Because these situations can lead to uncontrolled situations, the reset procedure gets designed with the following functionalities:

- LHL[6] functionality after pressing the reset button (iBtnReset) this between a set time (the high range) has to be let go to produce an outputsignal
- A checked output signal that is max. 1s is TRUE (oReset_1s)
- A checked output signal that is max. 1 PLC-cycle TRUE is (=edge signal)(oReset)



It is possible with the description to draft an operation scheme for the control module with the name FB_PE_Reset



The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

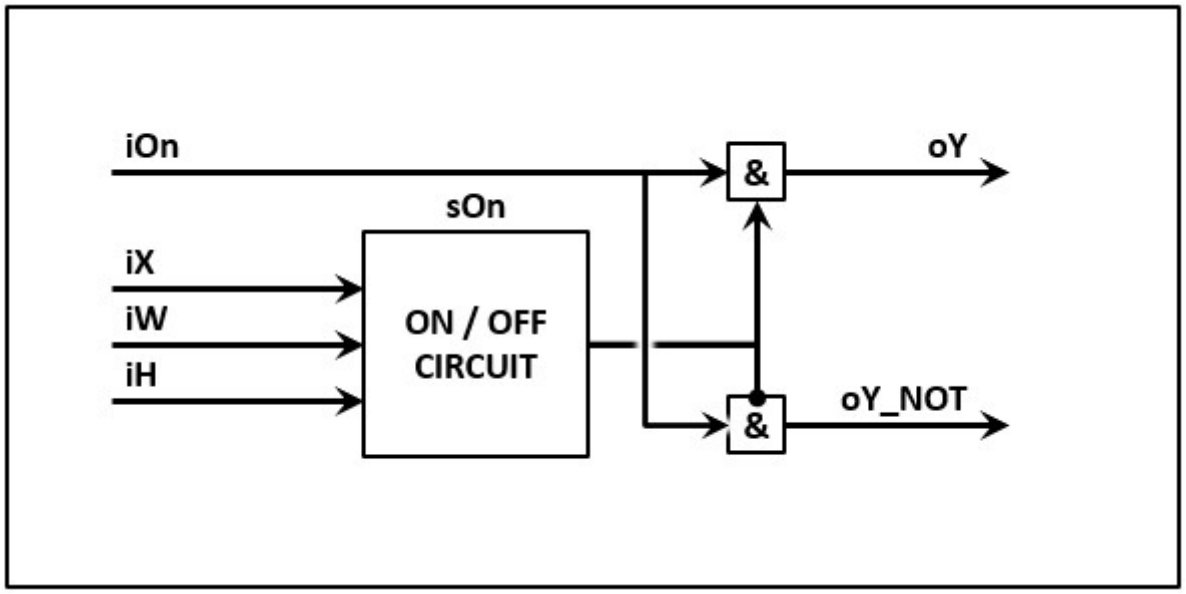
Two-point controller with hysteresis

A **two-point controller with hysteresis** uses an on-off switch to switch an actuator either on or off in function of a measure physical unit (= measured value x) and the desired physical unit (= setpoint W).

A **two-point controller with hysteresis** is consequently an on-off circuit with extra functionality like:

- The result of the controller (oY) is also offered inverted (oY_NOT)
- The possibility to switch the controller on and off. With turned-off controllers all the control outputs have the status FALSE(oY & oY_NOT)
- Setpoint (iW), the measured value (iX) and the hysteresis (iH) are adjustable

It is possible with the description to draft an operation scheme for the control module with the name FB_PE_TWPH



The end result is a **"Function building block"** which looks like the following images.

Text	Image
FDB example	
More simple example	

Specific designed procedure elements

Not all the think process can be collected with standard procedures. It's often necessary to design specific procedures and procedure elements. One of these following analysis methods gets applied to determine the strategy:

- GRAFCET
- Flowchart
- Mathematical formulas
- Operation schemes
- Selection controllers and explanation of control parameters

Exercise 1

Study material

Literature

Equipment

> **1** Engineering station
> **2** SIMATIC S7-1200 controller, e.g. CPU 1215C DC/DC/DC – firmware V4.2 or higher
> **3** SIMATIC STEP 7 software in TIA Portal – V15 SP1 or higher
> **4** Ethernet connection between engineering station and controller

Additional literature

The Crane Project

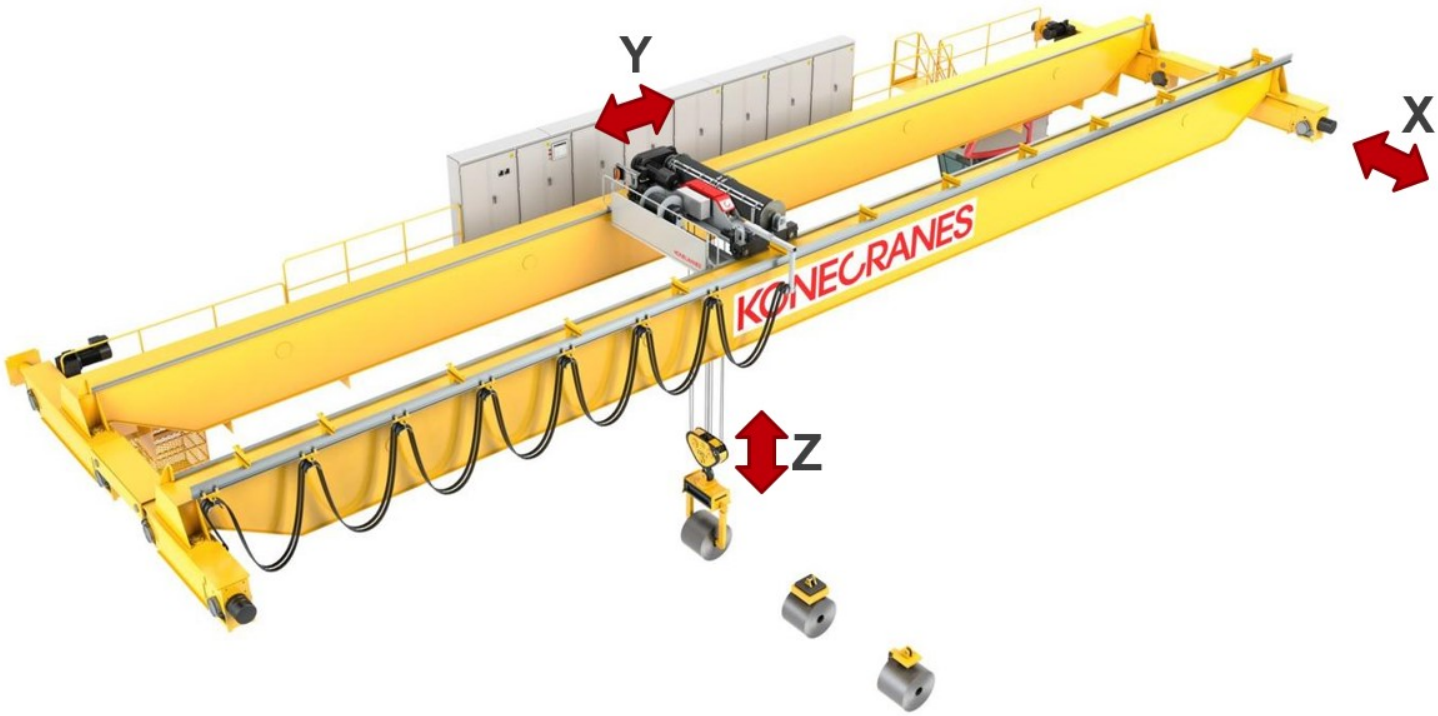
- The first goal is to add ProfiNET based devices
- The second goal is to add Profibus based devices
- The third goal is to add Drives

Back to the project scope

Scope1

Make a networkconfiguration of a crane that has the follwing:

- 5 Digital sensors
- 3 Motorstarters
- 2 Analog sensors
- A digital levelsensor
- 2 Profibus Devices
- 1 Drive



Goal 1 To add ProfiNET based devices

ProfiNET

Step 1 : Create a new TIA Portal project

```
Project name   : Ex1-TheCraneProject
Author         : Your name
Comment        : The Crane Project
```

Step 2 : Add a PLC-device with next CPU settings

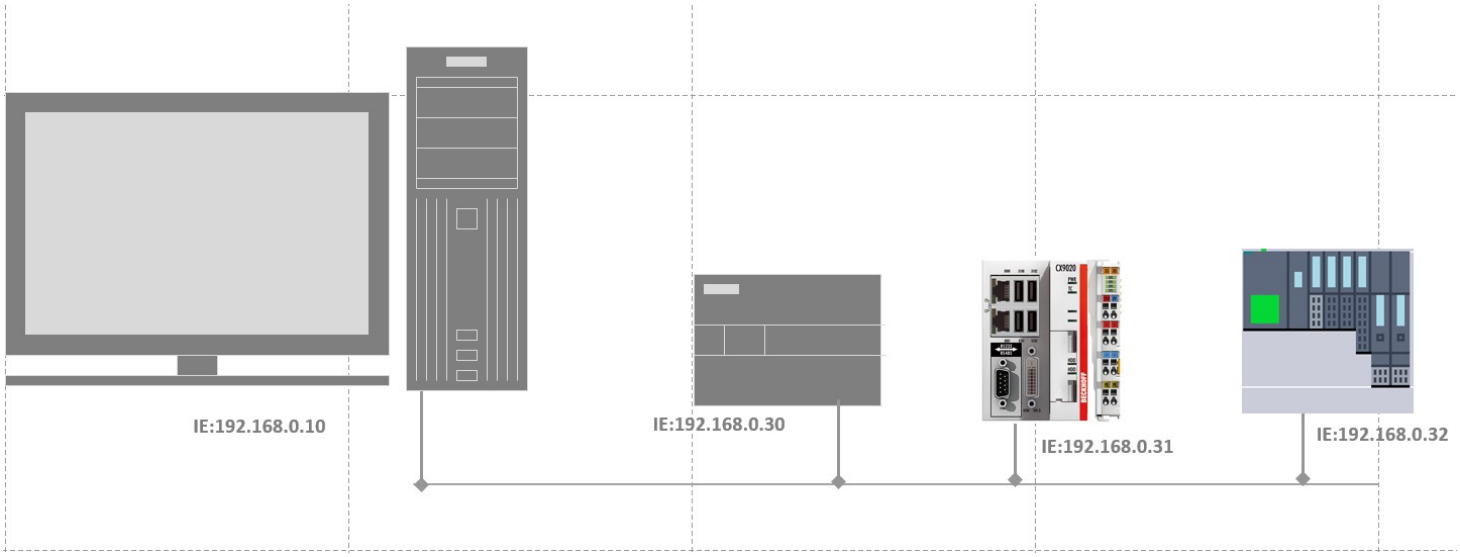
```
Type                          : See available CPU
System byte                   : %MB254
Clock memory byte             : %MB255
Digital input start address   : %IB0
Output start address          : %QB0
Analog input start address    : %IB64
IP-address                    : 192.168.0.30
IP-address subnet mask        : 255.255.255.0
```

Step 3: Search for the correct GSD files for the following devices:

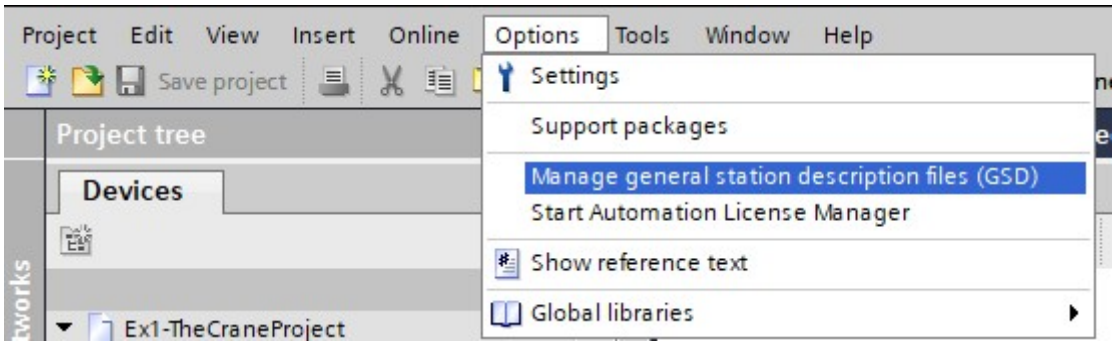
```
Beckhoff CX8093 island (IO on bridge)
-       3x digital sensor (grabber open, rabber closed, grabber on top)
-       2x digital sensor (eindeloop left, eindeloop right)

Siemens ET200S island 1  (pumps)
-       3x motorstarter (supply- & drainpump & heating)
-       2x analog measurement (level and temperature)
-       1x digital levelmeasurement (overflow)
```

If there is no internet available the required GSD files are included in the Documents.

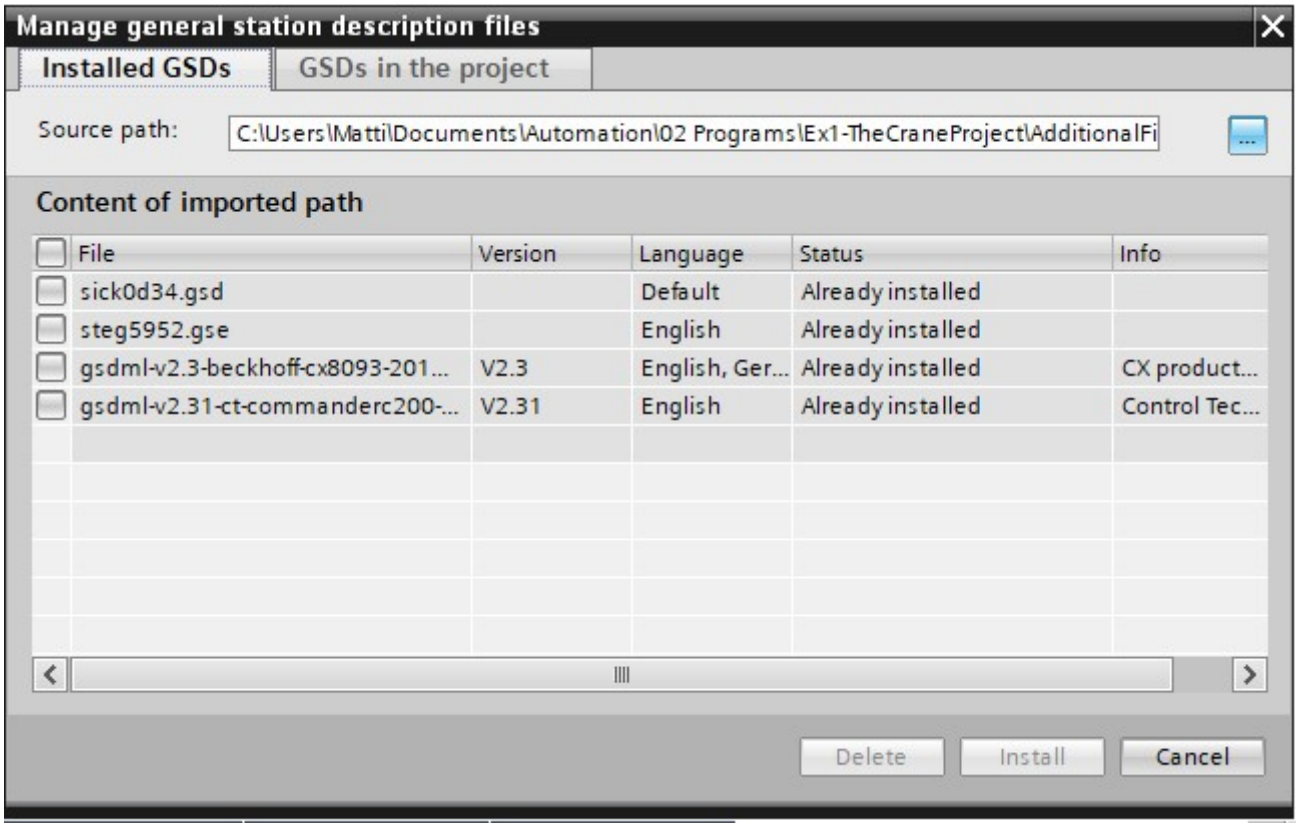
Overview of IP adresses and devices



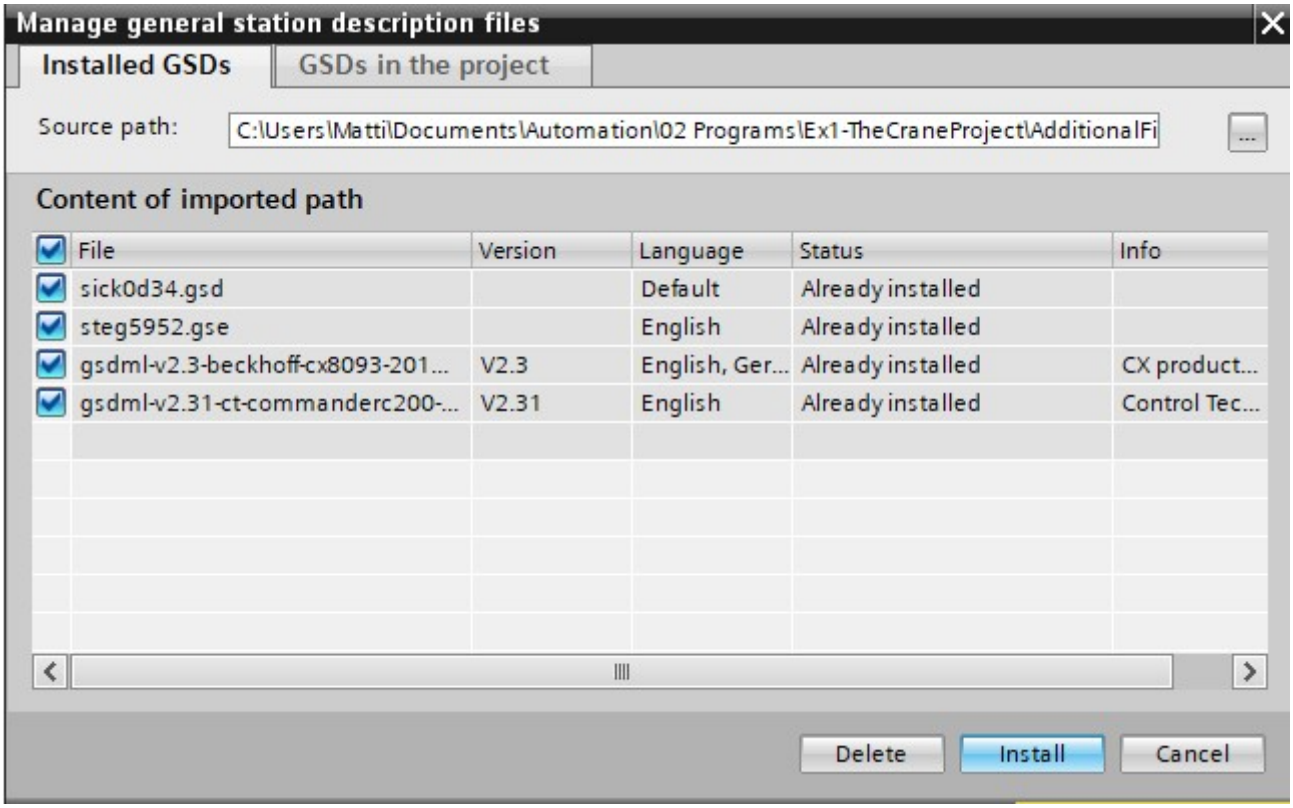
Step 4: Importing these GSD files:



Step 5: Select the right GSD files wherever you stored them:



Step 6: Install the selected GSD files:



Step 7: Import the correct devices from the "Hardware catalog" into "Network view"

Step 8: Connect the devices and give them the right IP adress

Step 9: Configure them for the right configuration given in **Step 3**

The Crane Project

- The first goal is to add ProfiNET based devices
- The second goal is to add Profibus based devices
- The third goal is to add Drives

Back to the project scope

Goal 2 to add Profibus based devices

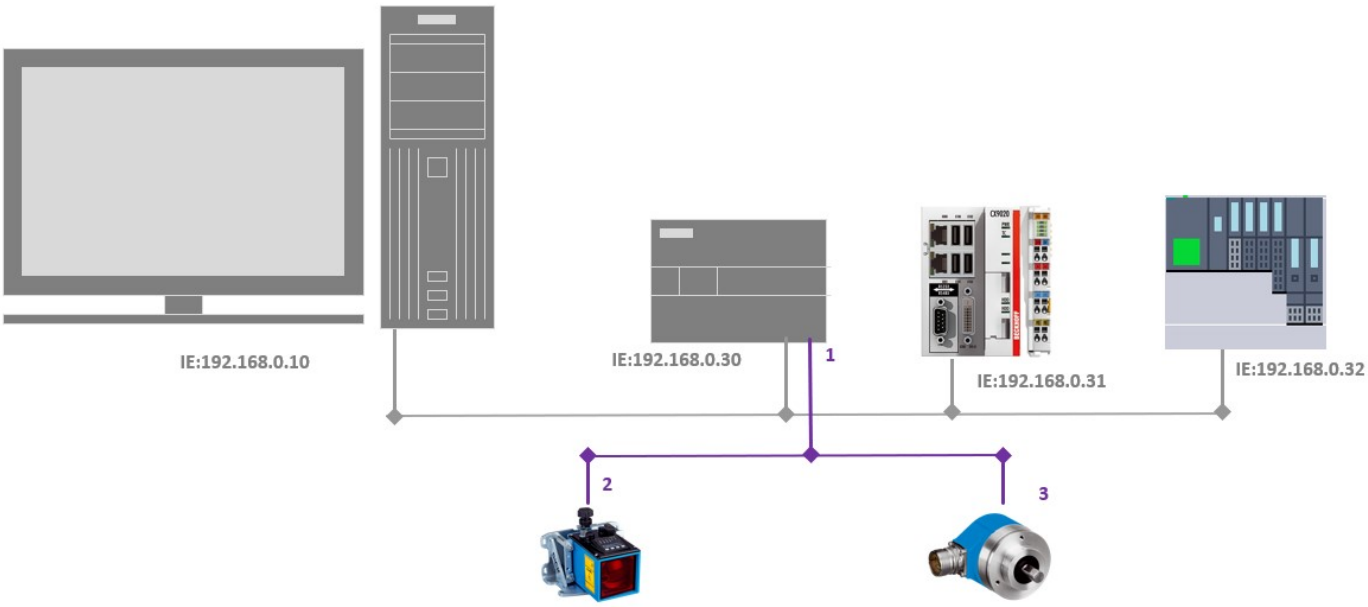
Profibus

Step 1: Search for the correct GSD files for the following devices:

```
- Sick Long-range-sensor DX100
- Sick wire-encoder ATM60
```

Step 2: Link them to the PLC

Overview of the network



Step 3: Configure the imported devices to the right measurement type

The Crane Project

- The first goal is to add ProfiNET based devices
- The second goal is to add Profibus based devices
- The third goal is to add Drives

Back to the project scope

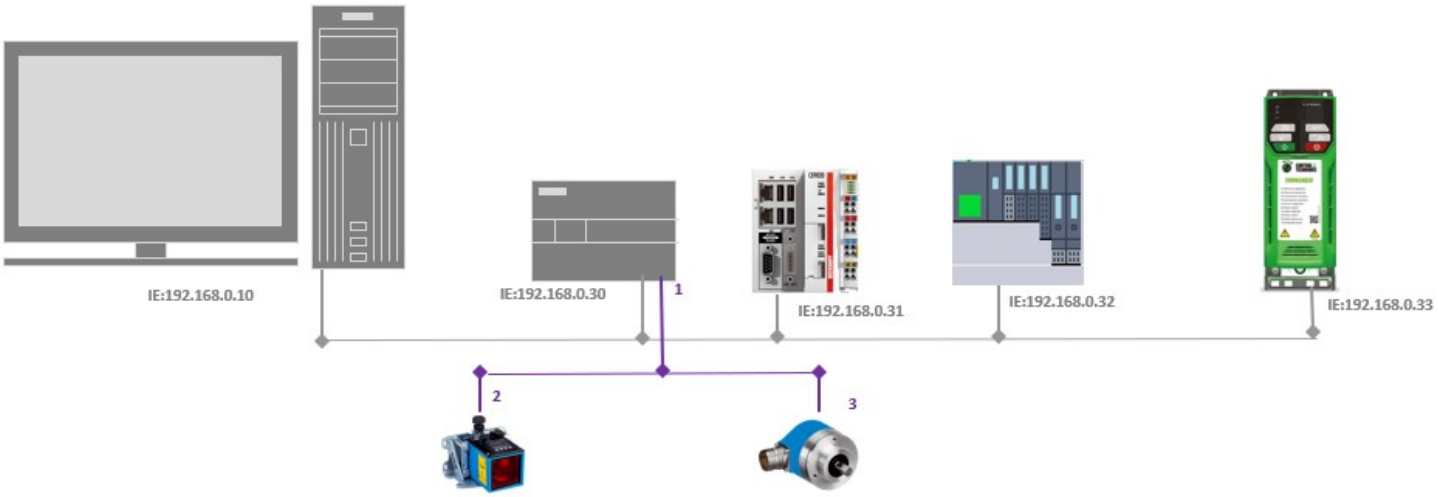
Goal 3 to add Drives

Drives

Step 1: Search for the correct GSD file for the following device:

```
Control Techniques Commander C300
```

Step 2: Link the drive to the PLC



Step 3: Compile the entire project and check for errors

Normal functionallity

- TIA compiles the project without issues

Exercise 2

Study material

Literature

- Addendum 06 : Software model following ANSI/ISA-88

Equipment

1 Engineering station 2 SIMATIC S7-1200 controller, e.g. CPU 1215C DC/DC/DC – firmware V4.2 or higher 3 SIMATIC STEP 7 software in TIA Portal – V15 SP1 or higher 4 Ethernet connection between engineering station and controller 5 Factory IO scene Pick & Place.factoryio

The Pick and Place Project

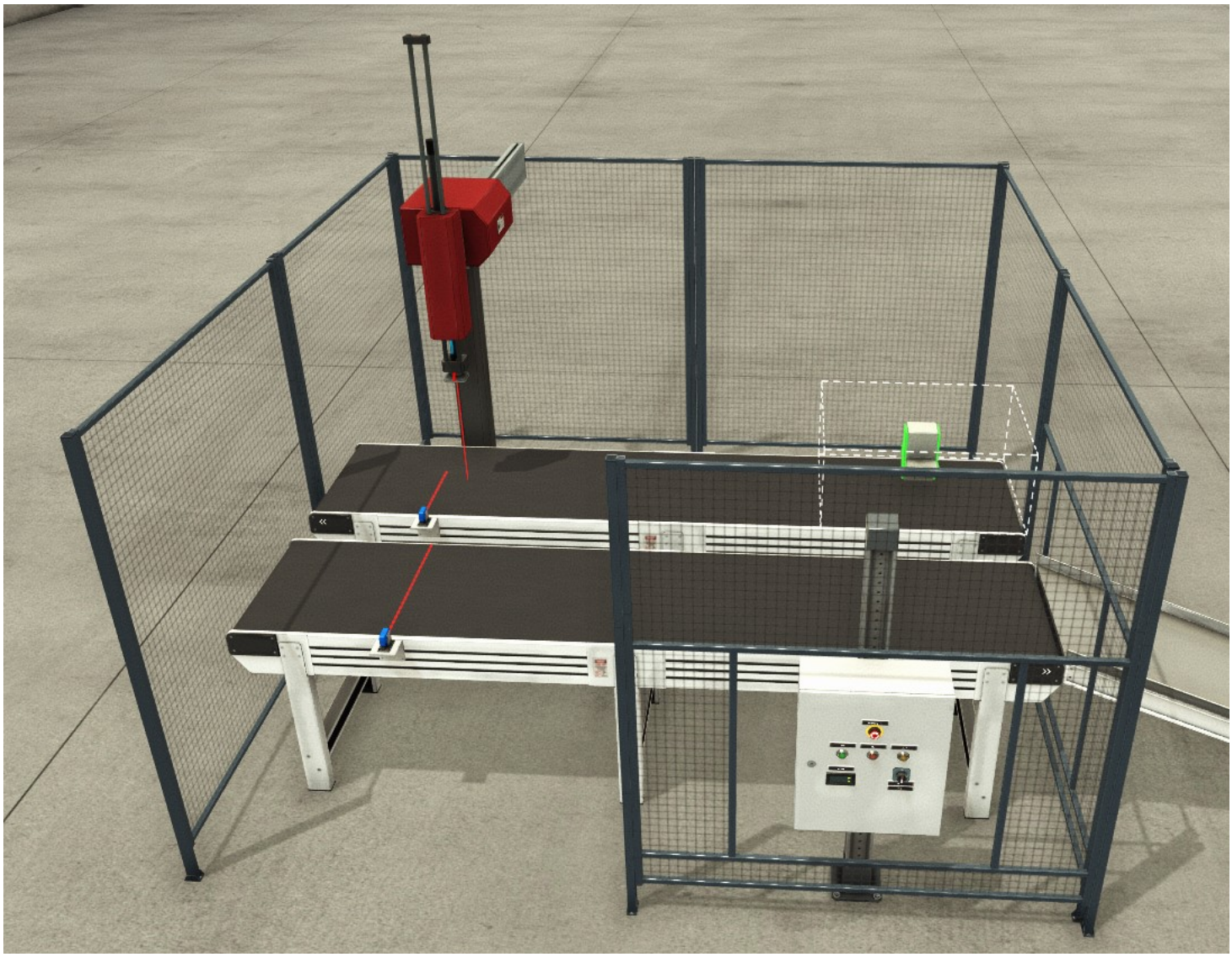
- The first goal is to retrieve an archived program.
- The second goal is to retrieve an archived library
- The third goal is to program the S88 following the S88 design
- The fourth goal is to import a exernal source file
- The last goal is to deliver a working project

Back to the project scope

Scope2

Automate the process of picking up packages and placing them on a different conveyor. This will be equipped with the following:

- 2 Digital photocells
- 2 Digital motor circuit breakers
- 2 Digital preasurised air valves
- 2 Digital contactors to control the conveyorbelt motors
- A digital vacuum grabber



Use the buttons on the PLC to control the motor circuit breakers.
Use the control board in FactoryIO to start and stop the machine.

The Pick and Place Project

- The first goal is to retrieve an archived program
- The second goal is to retrieve an archived library
- The third goal is to program the S88 following the S88 design
- The fourth goal is to import a exernal source file
- The last goal is to deliver a working project

Back to the project scope

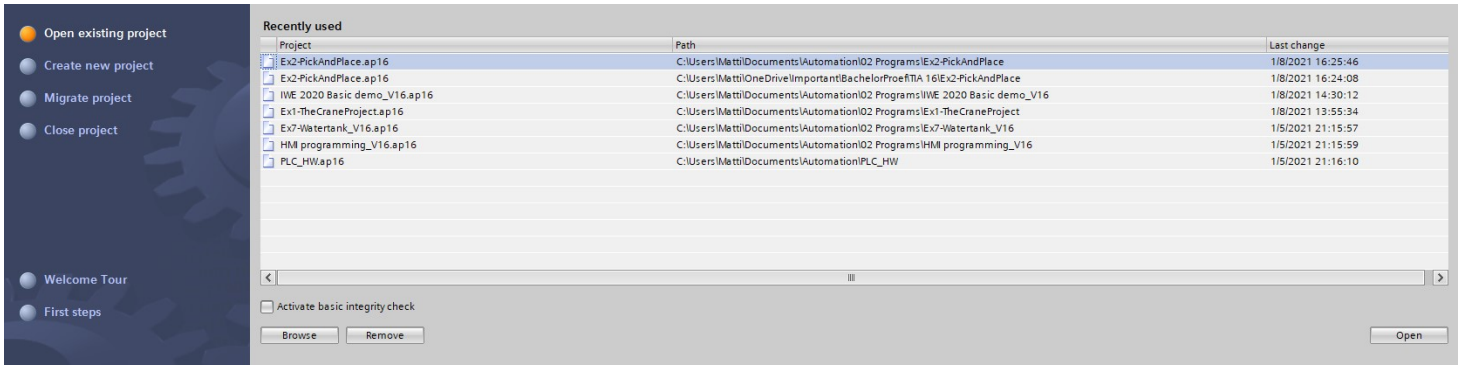
Goal 1 To retrieve an archived program

Step 1:

Copy/download the included Ex2-PickAndPlace.zap16 file. Copy the file into:

```
Filename : Ex2-PickAndPlace.zap16
Destination : \Documents\Automation
```

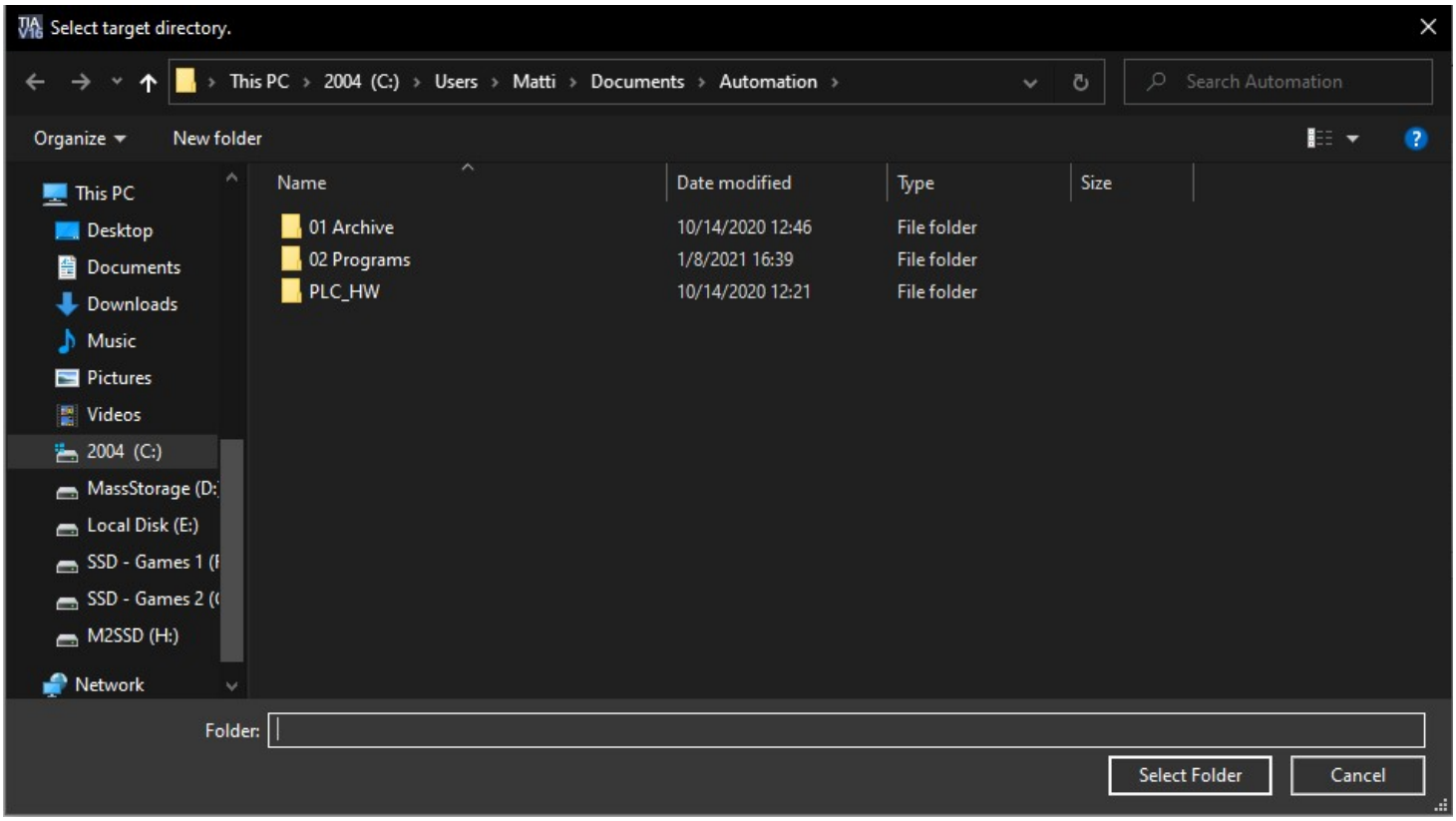
Step 2: Start TIA Portal and click on "Open existing project"



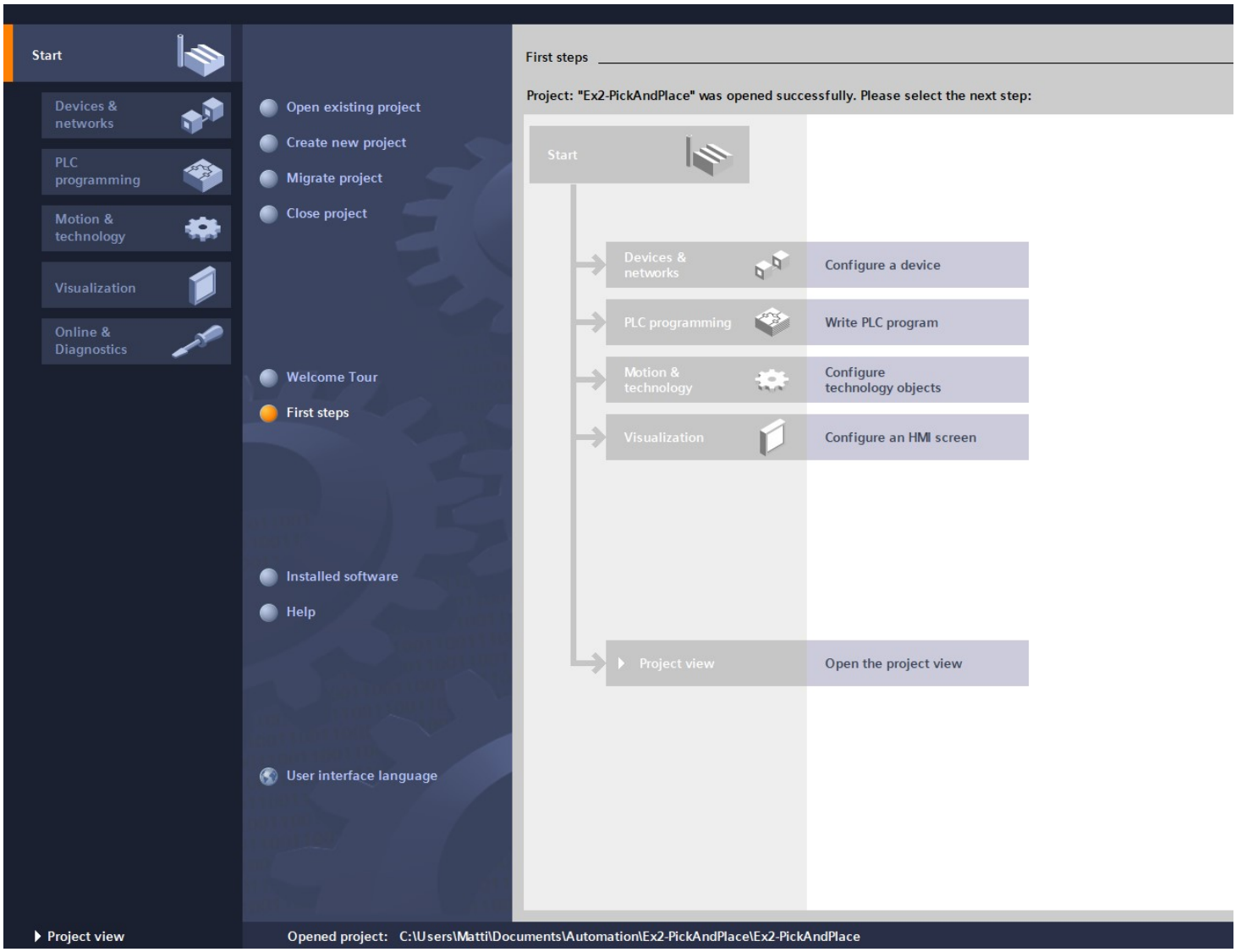
Step 3: Click on browse and search for the right archived file

```
Documents\Automation\Ex2-PickAndPlace.zap16
```

Step 4: Once selected you'll have to select the target destination. Select Documents\Automation in our case.



Step 5: Once this screen pops up you have succesfully opened the archive. Click on "Project view" to continue the exercise



The Pick and Place Project

- The first goal is to retrieve an archived program
- The second goal is to retrieve an archived library
- The third goal is to program the S88 following the S88 design
- The fourth goal is to import a exernal source file
- The last goal is to deliver a working project

Back to the project scope

Goal 2 To retrieve an archived library

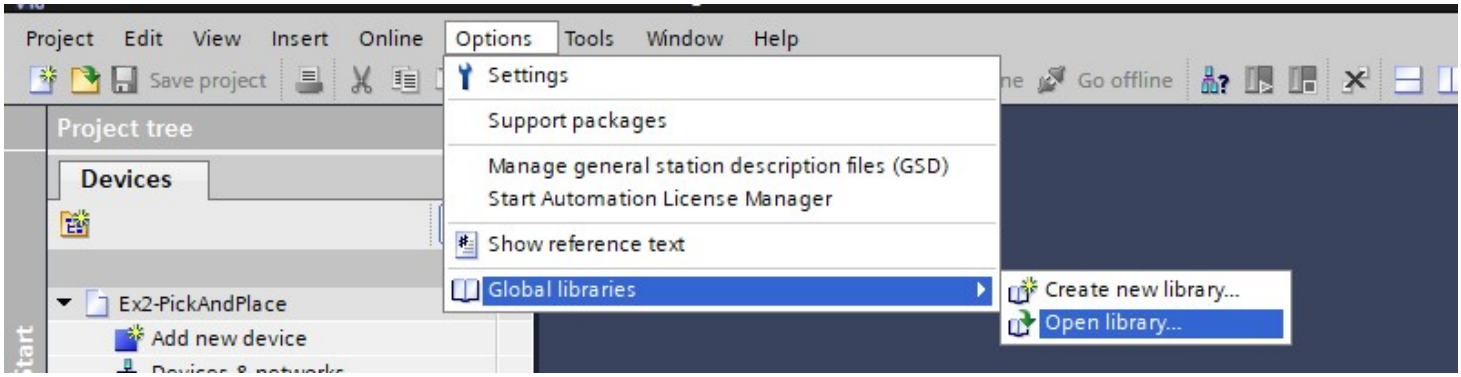
The point of retrieving this archived library is that this has all the control modules and procedure elemts programmed for you. So that later when you build your S88 in TIA you can drag them from the library into your project.

Step 1:

Copy/download the included .zal file named. Make a new subfolder into automation called "Library"(If it doesn't exist already). Copy the file into:

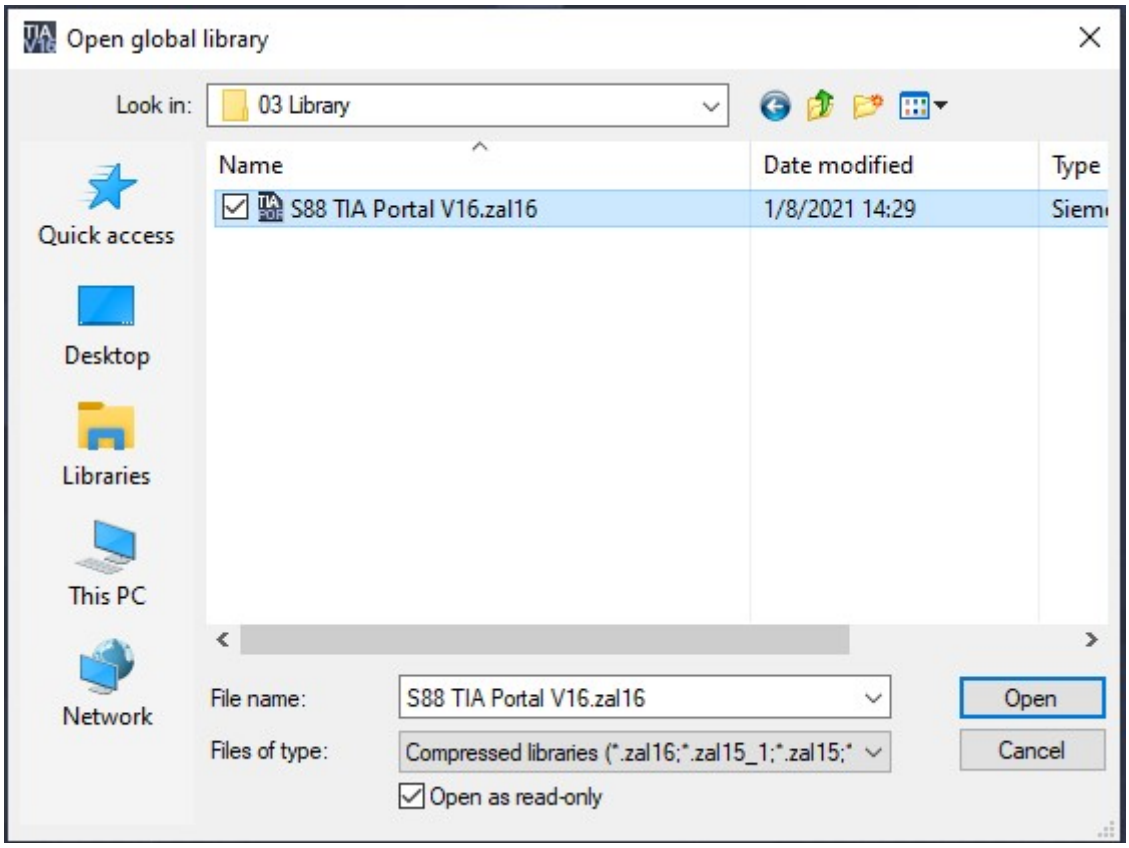
```
Filename : S88 TIA Portal V16.zap16
Destination : \Documents\Automation\Library
```

Step 2: To select the library go to "Options > Global libraries > Open library..."

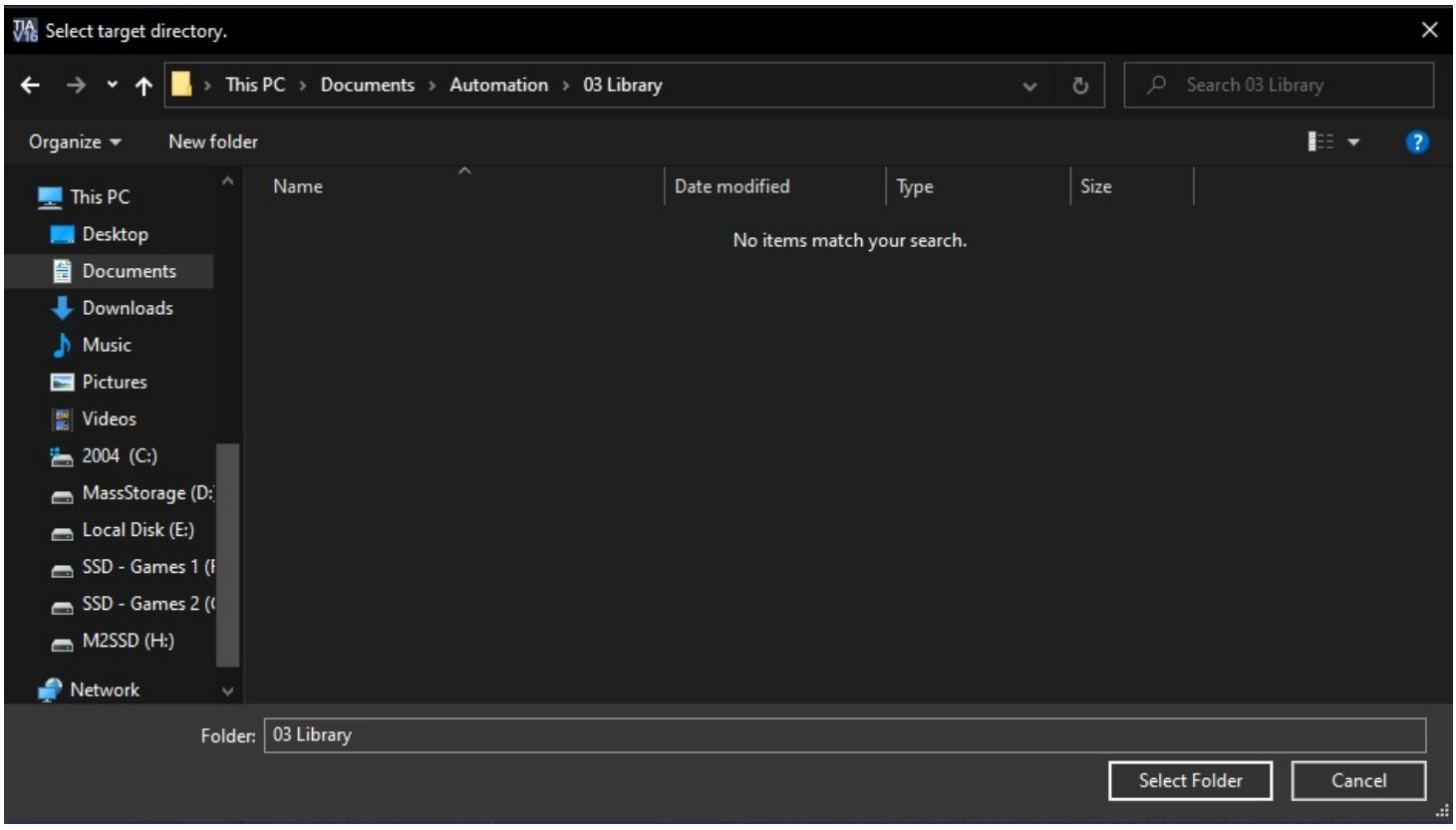


Step 3: Open in the follwing screen the saved library

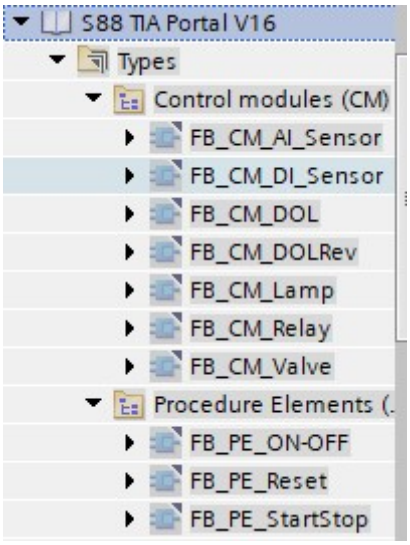
```
Files of type : Compressed Libraries
Destination : \Documents\Automation\Library
```



Step 4: Once selected you'll have to select the target destination. Select "Documents\Automation\Library" in our case.



Step 5: To make sure you have got the library opened check the right bottom of TIA Portal in "Global Libraries". This is done by clicking on libraries on the right side of tia portal.



The Pick and Place Project

- The first goal is to retrieve an archived program.
- The second goal is to retrieve an archived library
- The third goal is to program the S88 following the S88 design
- The fourth goal is to import a exernal source file
- The last goal is to deliver a working project

Back to the project scope

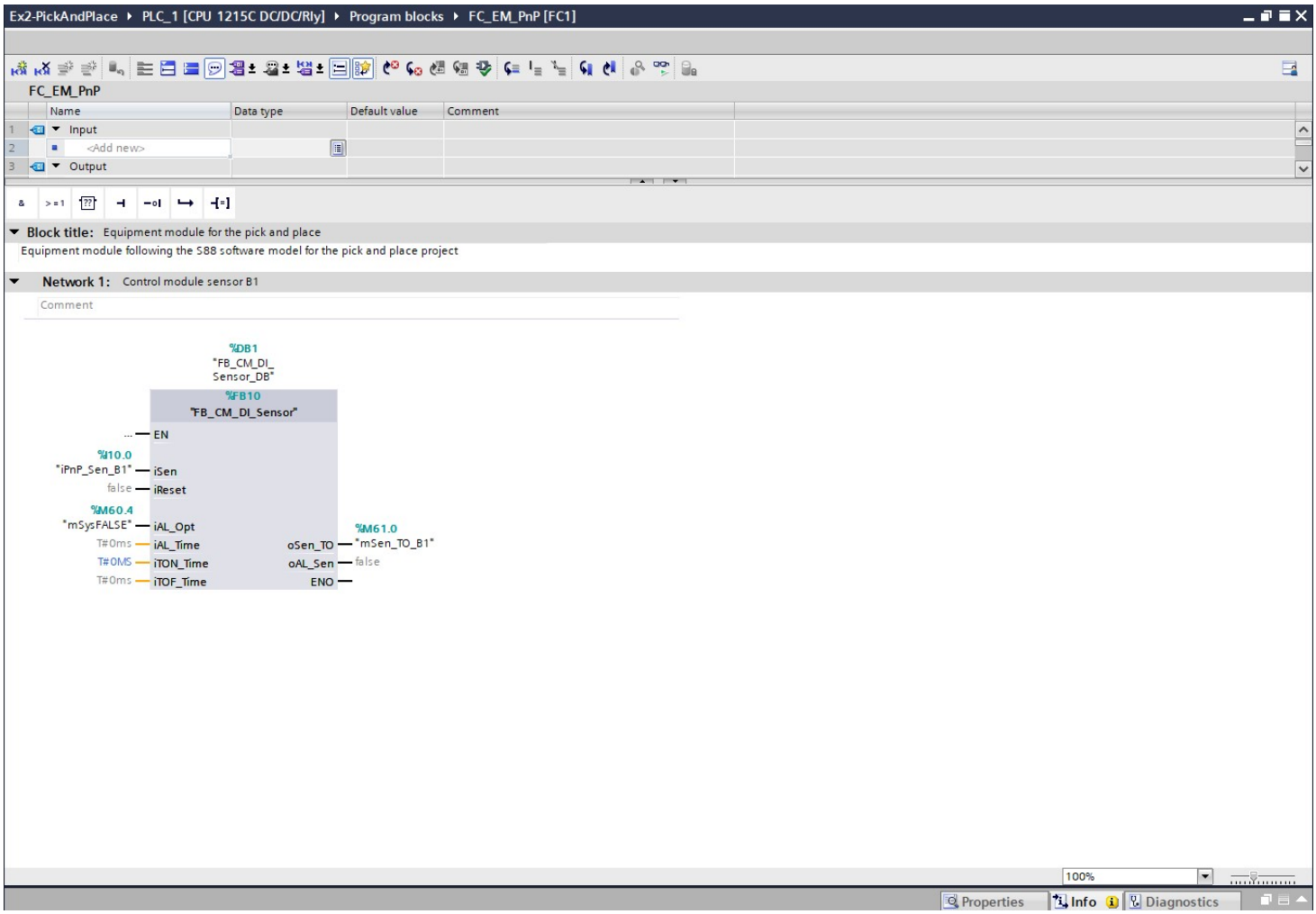
Goal 3 To program the S88

Step 1: Create the necessary PLC Tags:

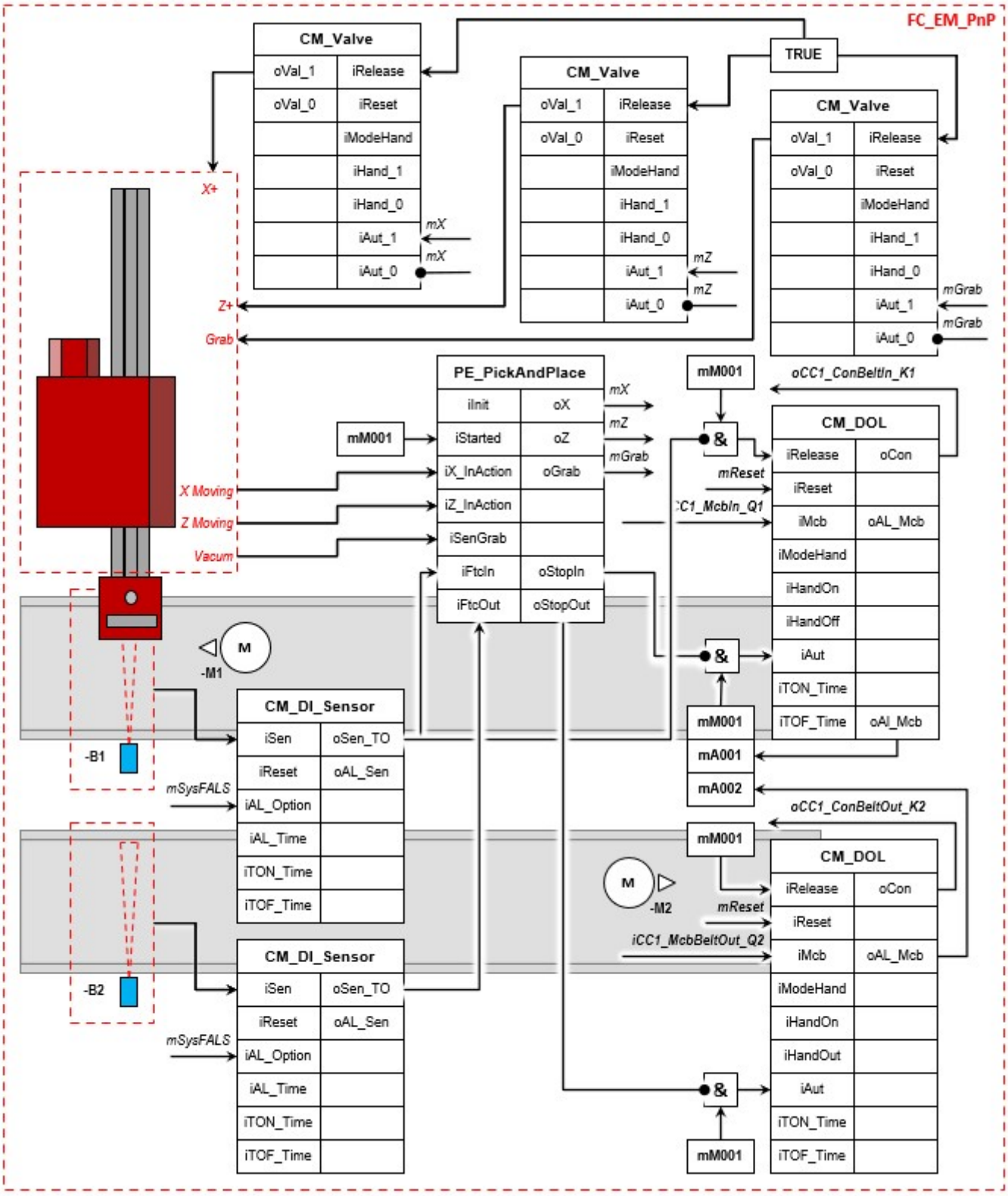
```
//Inputs
iCC1_McbConveyorIn_Q1 - BOOL - %I 0.0 - Motor circuit breaker for conveyor belt entry
iCC1_McbConveyorOut_Q2 - BOOL - %I0.1   - Motor circuit breaker for conveyor belt exit
iPnP_Sen_B1 -   BOOL - %I10.0 - Sensor item at entry
iPnP_Sen_B2 - BOOL - %I10.1 - Sensor item at exit
Moving X - BOOL - %I10.2 - Robot is moving in the X axis
Moving Z - BOOL - %I10.3 - is moving in the Z axis
Vacuum - BOOL - %I10.4 - The vacuum of the robot is active
iCC1_BtnStart_S1 - BOOL - %I10.5 - Start button
iCC1_BtnReset_S3 - BOOL - %I10.6 - Reset button
iCC1_BtnStop_S2 - BOOL - %I10.7 - Stop button
iCC1_BtnEms_S4  - BOOL - %I11.0 - Emergency stop button


//Outputs
iCC1_McbConveryorIn_K1 - BOOL - %Q10.0 - Contactor conveyor belt entry
iCC1_McbConveyorOut_K2 - BOOL - %Q10.1 - Contactor conveyor belt exit
Move X - BOOL - %Q10.2 - Moves the robot in the X axis
Move Z -  BOOL - %Q10.3 - Moves the robot in the Z axis
Grab - BOOL - %Q10.4 - Grabs an item
oCB1_LmpError_H1 - BOOL - %Q10.7 - Error lamp


//Flags
mM001 - BOOL - %M50.0 - System started
mA001 - BOOL - %M50.1 - Motor circuit breaker conveyot belt entry alarm
mA002 - BOOL - %M50.2 - Motor circuit breaker coneyor belt exit alarm

mZ - BOOL - %M60.1 - Flag move Z-axis of the robot
mGrab - BOOL - %M60.2 - Flag grab item
mReset - BOOL - %M60.3 - Flag reset
mSysFALSE - BOOL - %M60.4 - Flag FALSE
mStopIn - BOOL - %M60.6 - Flag stop conveyor belt entry
mStopOut - BOOL - %M60.7 - Flag stop conveyor belt exit
mSen_TO_B1 - BOOL - %M61.0 - Flag sensor B1
mSen_TO_B2 - BOOL - %M61.1 - Flag sensor B2
mSysIlnit       - BOOL - %M60.5 - Flag initilization
```

Step 2 : Open the Function FC_EM_PnP[FC1]

Step 3 : There is one control module already present and linked to the right tags.



Step 4: Program the remaining control modules like the following S88 design. *For each control module or procedure element you need to have a seperate network*



The Pick and Place Project

- The first goal is to retrieve an archived program.

- The second goal is to retrieve an archived library
- The third goal is to program the S88 following the S88 design
- The fourth goal is to import a exernal source file
- The last goal is to deliver a working project

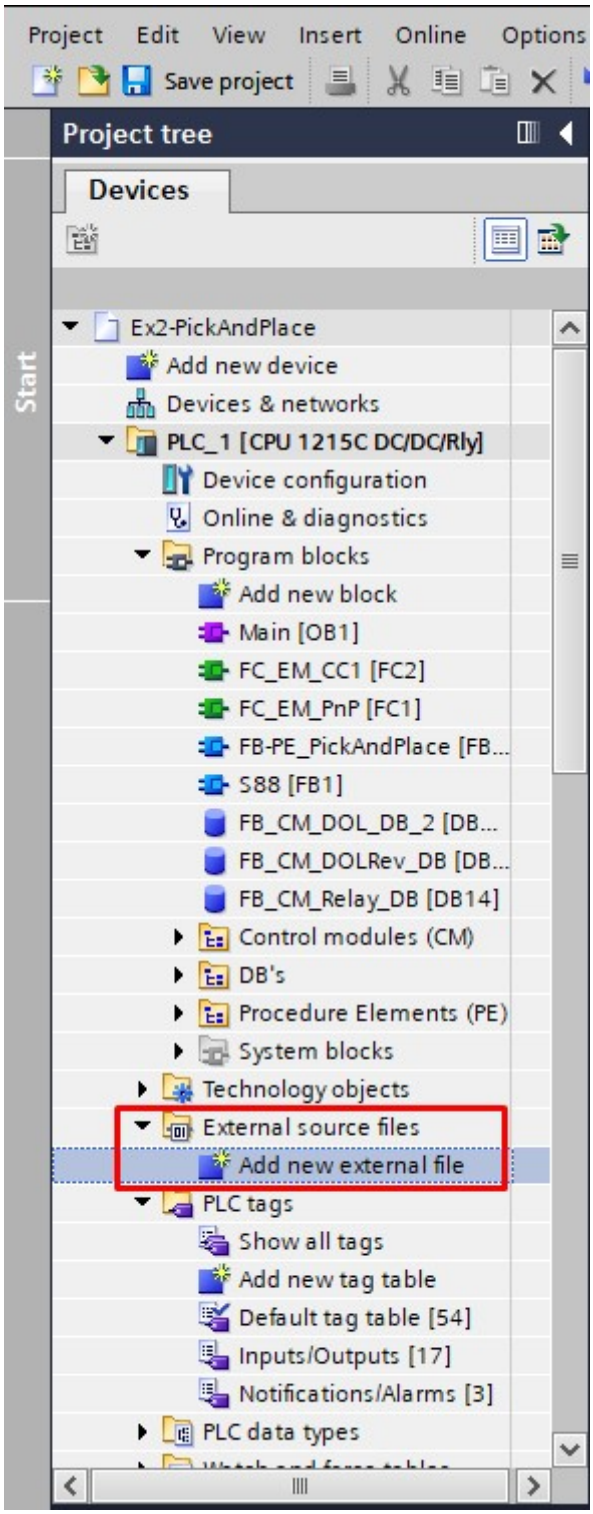
Back to the project scope

Goal 4 To import an external source file

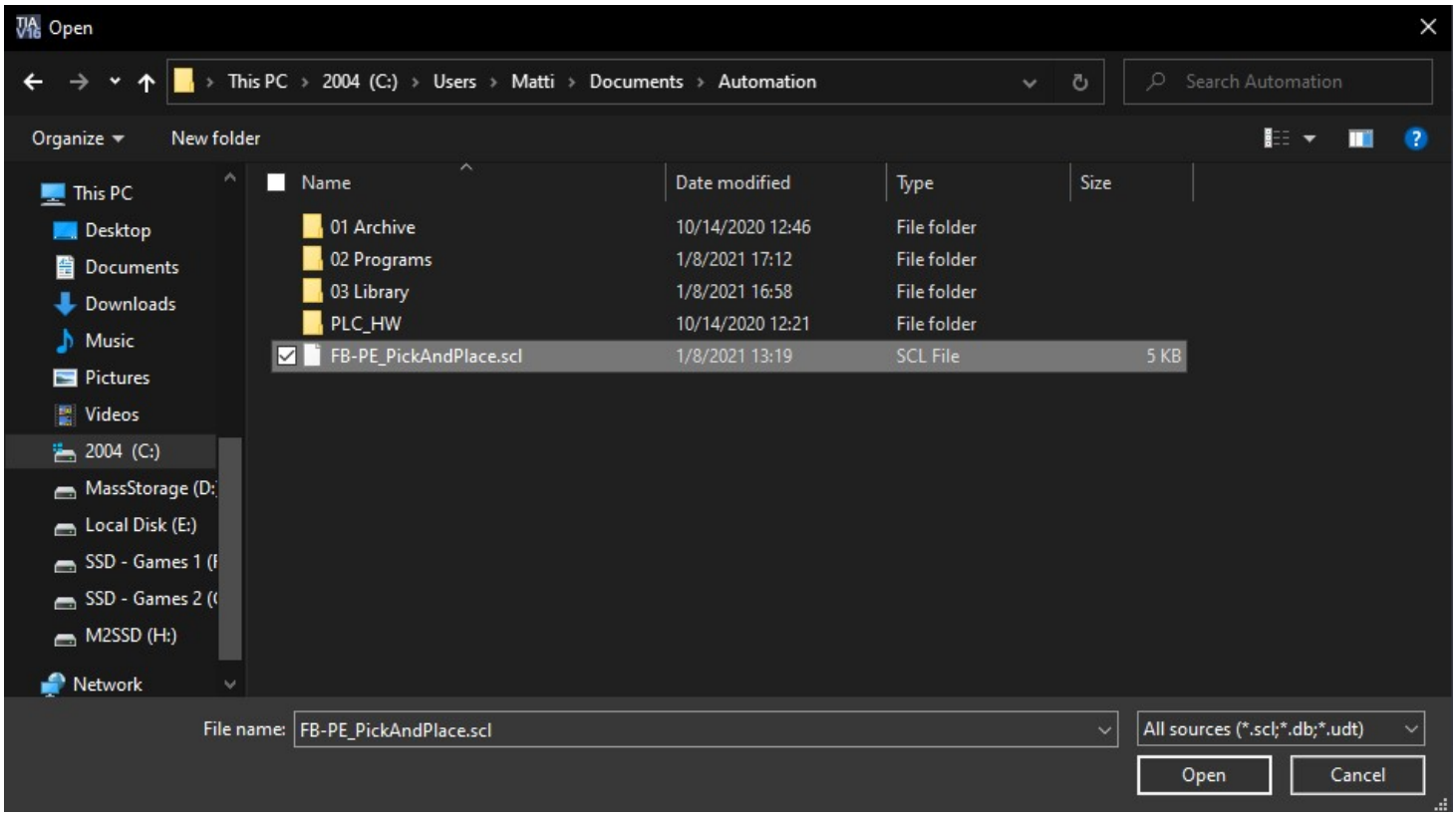
Step 1: Download/copy FB-P_PickAndPlace.scl and place it under

```
Filename : FB-P_PickAndPlace.scl
Destination : \Documents\Automation
```

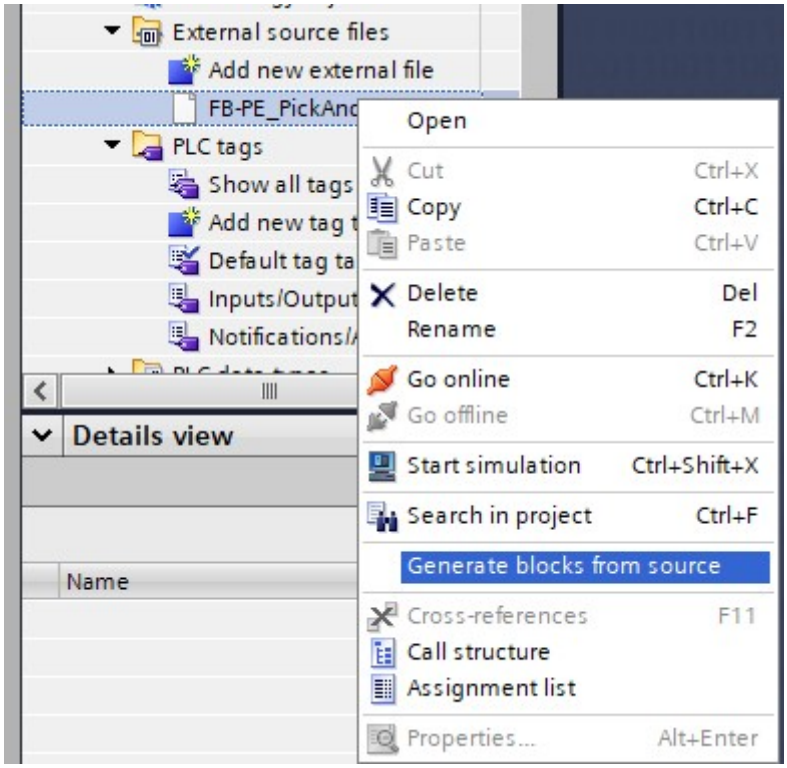
Step 2: Search for "External source files" in the project tree and click on "Add new external file"



Step 3: In the file browser select FB-P_PickAndPlace.scl and open it



Step 4: Under the "External source files" the selected file will pop up. Right click on it and run "Generate source blocks from source"



Step 5: Add both the Functions into *Function block* FC_EM_PnP [FC1]:

```
FB-P_PickAndPlace into a new network
```

Step 5: Link the right tags with the block that has been generated

The Pick and Place Project

- The first goal is to retrieve an archived program.
- The second goal is to retrieve an archived library
- The third goal is to program the S88 following the S88 design
- The fourth goal is to import a exernal source file
- The last goal is to deliver a working project

Back to the project scope

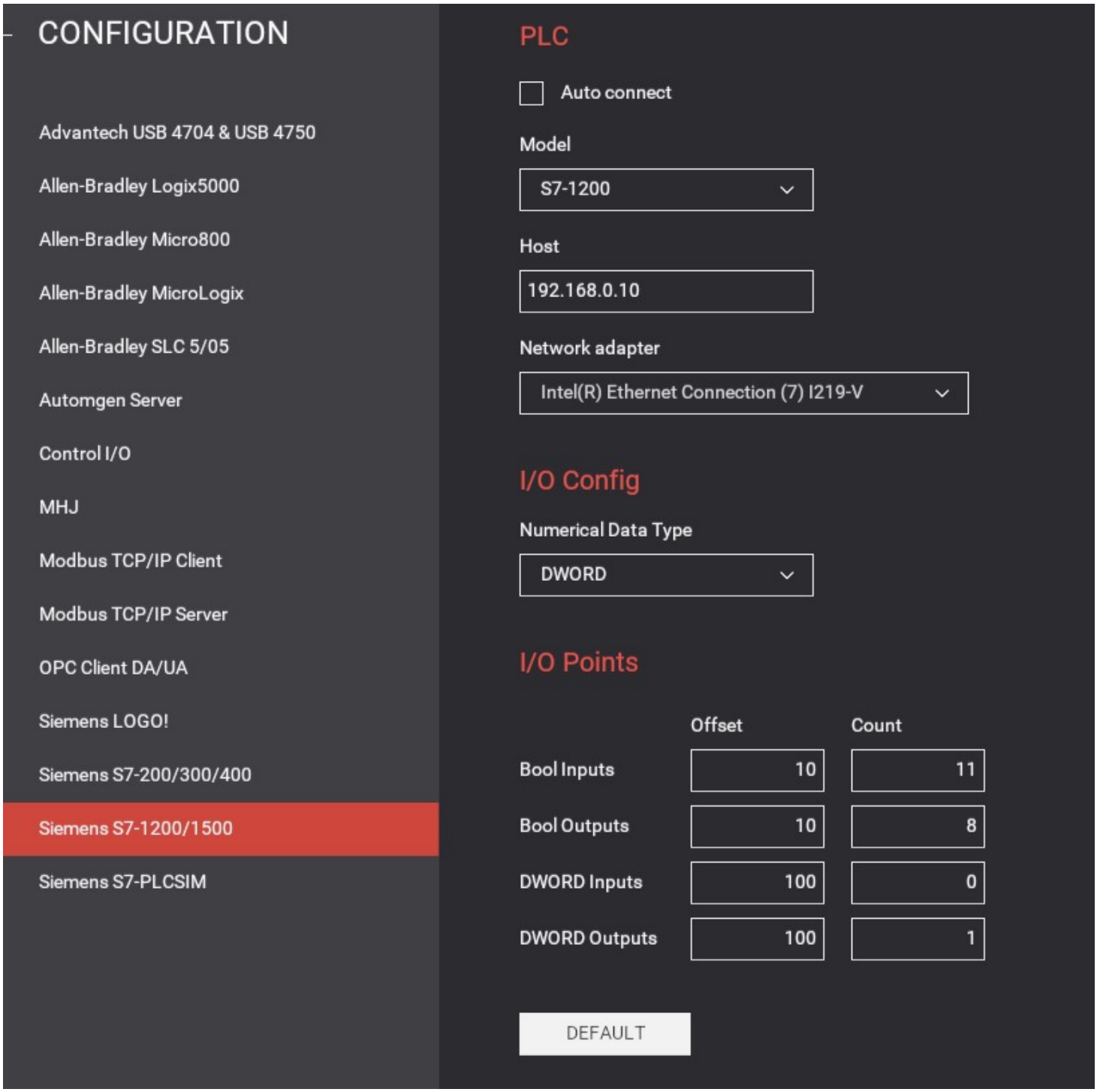
Goal 5 To deliver a working project2

Step 1: Open the FactoryIO scene called "Pick and Place" (it's one of the default scene's)

Step 2: Configure the driver to the right PLC

Step 3: Open up the configuration

```
IP adress: 192.168.0.10
Bool inputs: 10
Bool outputs: 10
```



Step 4: Compile the hardware with a rebuild all command

Step 5: Compile the software with a rebuild all command

Step 6: Download hardware and software to the PLC_1

Step 7: Test the Project

Normal functionallity

- Start the conveyorbelt by pressing the start button in FactoryIO

- The sensor will detect an item on the entry conveyor belt
- Entry conveyor belt will stop and the robot will pick up the item
- Exit conveyor belt will stop and the robot will put down the item

Exercise 3

Study materials

Literature

- Addendum 04 : Grafcet

Equipment

1 Engineering station 2 SIMATIC S7-1200 controller, e.g. CPU 1215C DC/DC/DC – firmware V4.2 or higher 3 SIMATIC STEP 7 software in TIA Portal – V15 SP1 or higher 4 Ethernet connection between engineering station and controller 5 Factory IO scene Pick & Place.factoryio

The Pick and Place Project

- The first goal is to program a GRAFCET
- The second goal is to program a Flowchart
- The third goal is to deliver a working

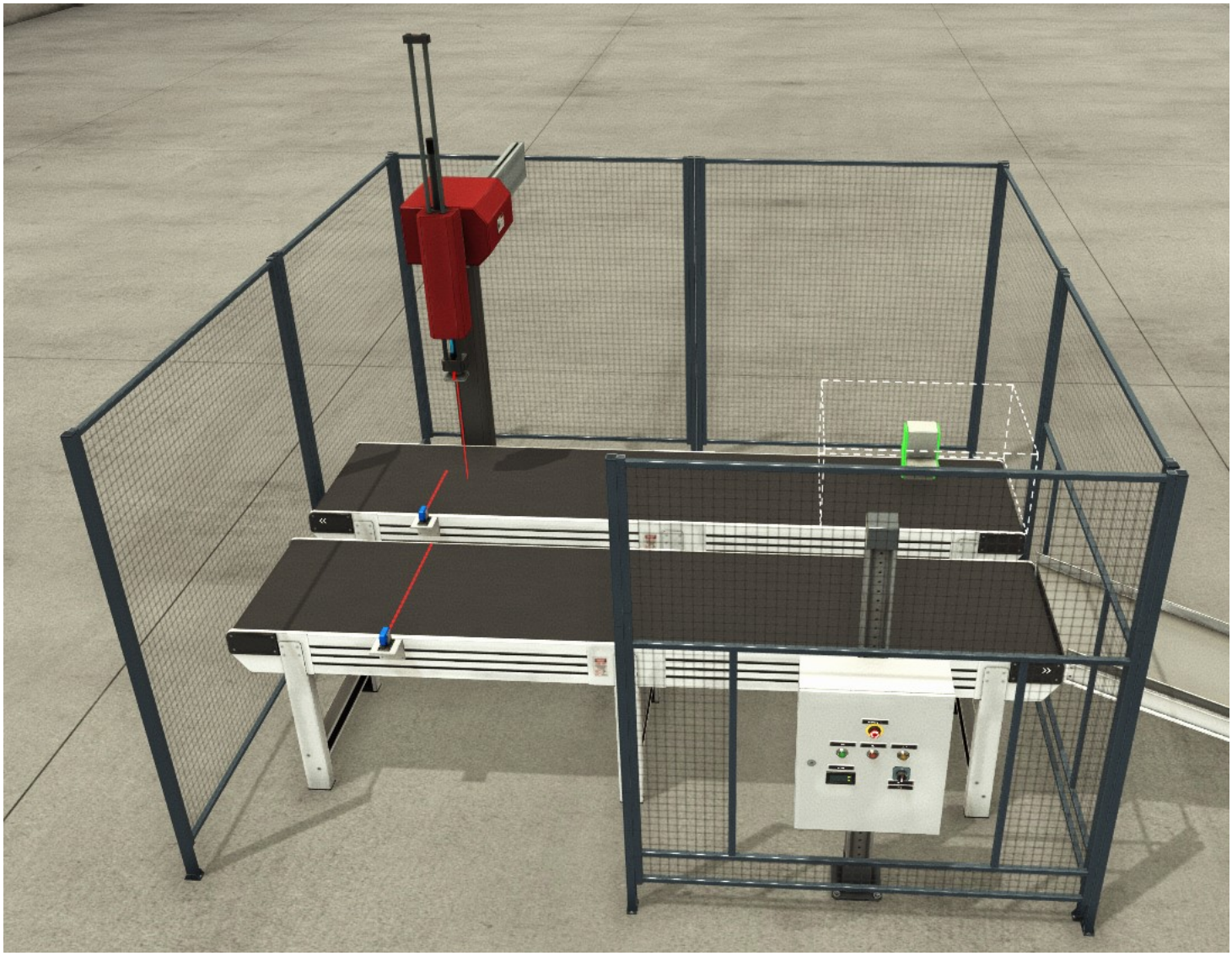
Back to the project scope

Scope3

Automate the process of picking up packages and placing them on a different conveyor. This will be equipped with the following:

- 2 Digital photocells

- 2 Digital motor circuit breakers
- 2 Digital preasurised air valves
- 2 Digital contactors to control the conveyorbelt motors
- A digital vacuum grabber



Use the buttons on the PLC to control the motor circuit breakers.
Use the control board in FactoryIO to start and stop the machine.

Goal 1 To program a GRAFCET

Step 1: Open the previous exercise Ex2-PickAndPlace.

```
Filename : Ex2-PickAndPlace.ap16
```

Step 2: Remove the function block that we previously ported into TIA.

```
Functionblock : FB-P_PickAndPlace
```

Step 3: Copy/download the included .zal file named. Make a new subfolder into automation called "Library"(If it doesn't exist already). Copy the file into:

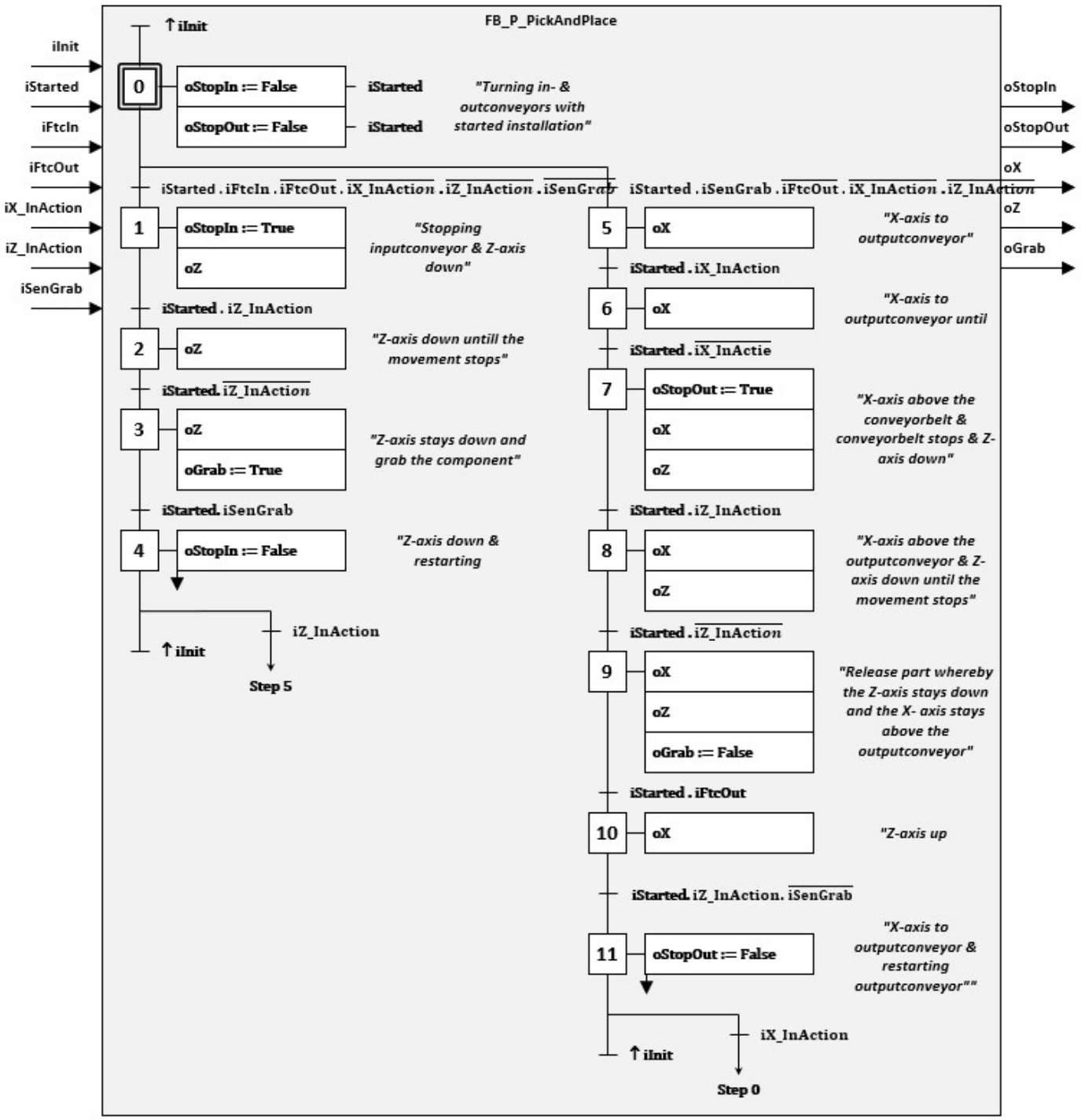
```
Filename : S88 GRAFCET.zap16
Destination : \Documents\Automation\Library
```

Step 4: Open the archive as we did in Ex02. Here you'll find a template of a GRAFCET under "Master Copies" called "FB-P_GRAFCET". Drag this into your TIA portal project.

Step 5: Rename the block to FB-P_PickAndPlace.

Remark: The previous block "FB-P_PickAndPlace" in "FC_EM_PnP" will be red, to fix this right click on the block and press "Update block call".

Step 6: Program the following GRAFCET into FB-P_PickAndPlace.



The Pick and Place Project

- The first goal is to program a GRAFCET
- The second goal is to program a Flowchart
- The third goal is to deliver a working

Back to the project scope

Goal 2 To program a Flowchart

Step 1: Create the necessary PLC Tags:

```
//Outputs
Counter - DWORD - %QD100 - Amount of processed packages

//Flags
mA003 - BOOL - %M50.3 - Motor circuit breaker coneyor belt exit alarm
mCounter - DWORD - %QD100 - Flag amount of processed packages
```

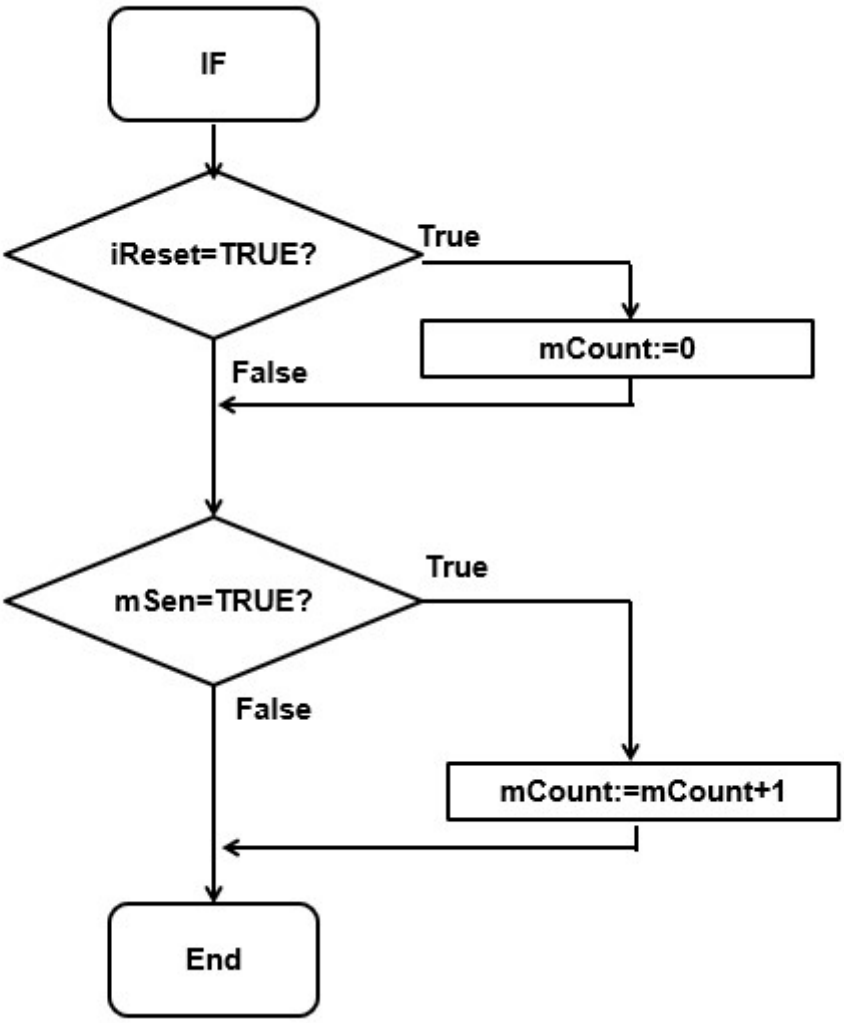
Step 2: Create the Function Block FB_Counter[FB1] in the language SCL

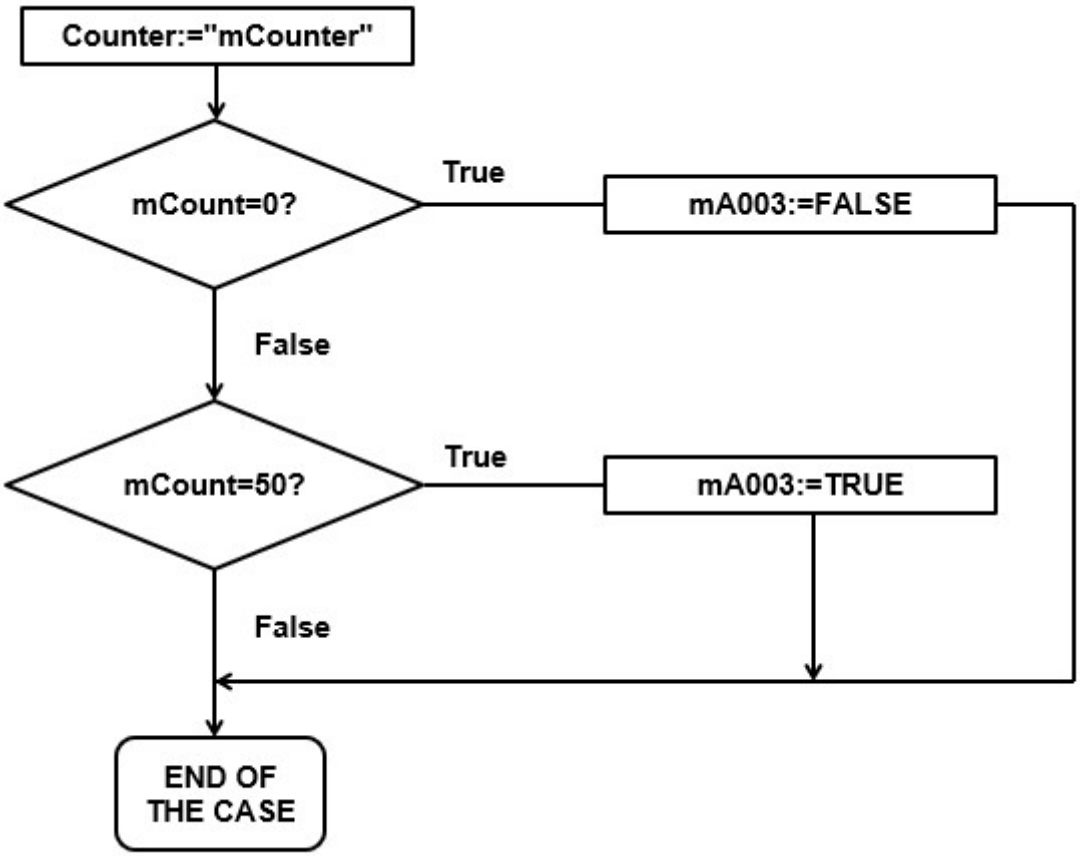
Step 3: Create the necessary block tags:

```
//Static
mSen - BOOL - Rising edge detection for sensor on the output conveyor
```

Step 4: Add a rise edge direction for "mSen_TO_B2"
Remark "mSen_TO_B2" will be the input and "mSen" will be the output

Step 5: Create the following Flowcharts


IF
iReset=TRUE?
True
mCount:=0
False
mSen=TRUE?
True
False
mCount:=mCount+1
End



The Pick and Place Project

- The first goal is to program a GRAFCET
- The second goal is to program a Flowchart
- The third goal is to deliver a working project

Back to the project scope

Goal 3 To deliver a working project3

Step 1: Compile the hardware with a rebuild all command

Step 2: Compile the software with a rebuild all command

Step 3: Download hardware and software to the PLC_1

Step 4: Test the Project

Normal functionallity

- Start the conveyorbelt by pressing the start button in FactoryIO
- The sensor will detect an item on the entry conveyor belt
- Entry conveyor belt will stop and the robot will pick up the item
- Exit conveyor belt will stop and the robot will put down the item
- The counter counts up each time a object gets detected by the fotocell on the outputconveyor
- Stop light lights up when you reach 50 moved packages

Exercise 4

Study material

Literature

- Addendum 05 Controllers

Equipment

1 Engineering station 2 SIMATIC S7-1200 controller, e.g. CPU 1215C DC/DC/DC – firmware V4.2 or higher 3 SIMATIC STEP 7 software in TIA Portal – V15 SP1 or higher 4 Ethernet connection between engineering station and controller 5 Factory IO scene Level Control.factoryio

The Watertank Project

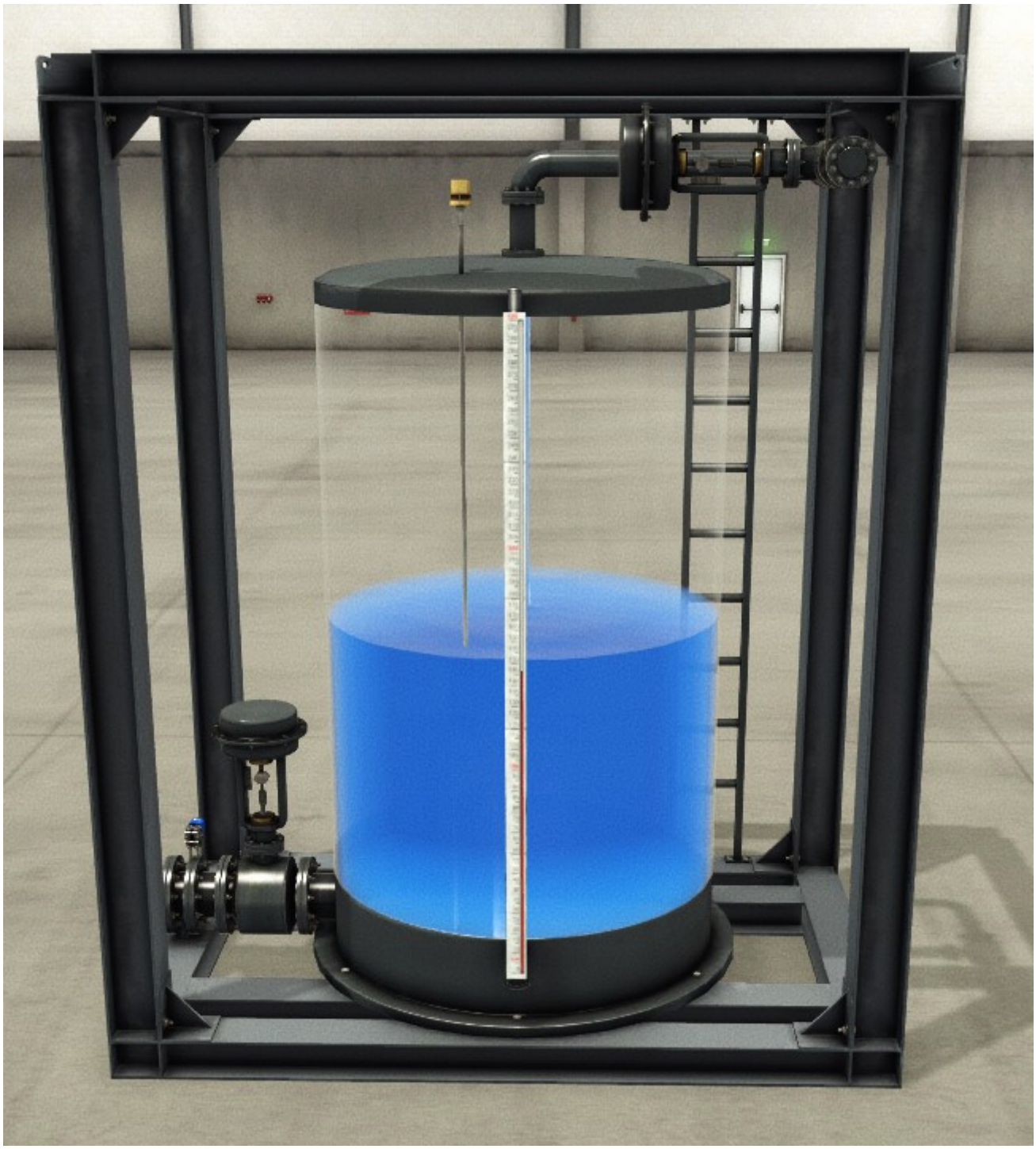
- The first goal is to program an ON/OFF controller
- The second goal is to program a PID controller
- The third goal is to deliver a working program

Back to the project scope

Scope4

Automate controlling the level in **watertank T1**. that is equipped with

- An analog level sensor
- An analog flow sensor on the outlet
- An analog inlet valve
- An analog outlet valve



Use the buttons, lamps, potentiometer and analog indicator on the ASTI PLC board to control the watertank.

Characteristics

- Height: 3 m
- Diameter: 2 m
- Discharge pipe radius: 0.125 m
- Input flow: 0.25 m^3/s
- Output flow: 0.3543 m^3/s

The Watertank Project

- The first goal is to program an ON/OFF controller
- The second goal is to program a PID controller
- The third goal is to deliver a working program

Back to the project scope

Goal 1 To program an ON OFF controller

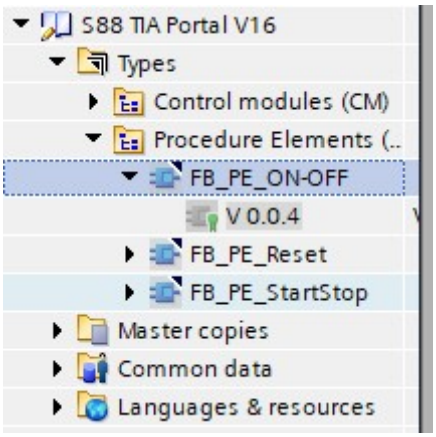
Step 1: Open project Ex7-Watertank

Step 2: Open the *Function* FC_T1[FC2]

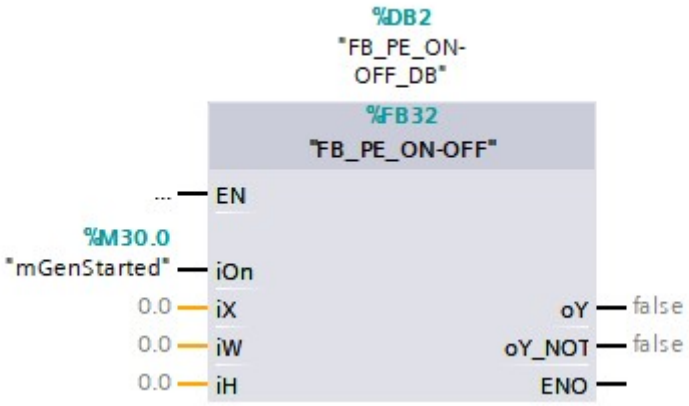
Step 3: Delete the content within network 4 : Level control

Step 4: Open the library "S88 TIA Portal V16"

Step 5: Copy "FB_PE_ON-OFF" into the *Function* FC_T1[FC2] network 4 :



Step 6: Link the right in- & outputs of the ON-OFF controller.



Step 8 : Open the FactoryIO scene called:

```
Filename : Level_Control.factoryio
Filelocation : \Documents\Factory IO\My Scenes
```

Step 9: Compile the hardware with a rebuild all command

Step 10: Compile the software with a rebuild all command

Step 11: Download hardware and software to the PLC_1

Step 12: Test the Project

Normal functionallity

- If you change the iH value the hysteresis of the controller will change
- If you change the iW value the setpoint will change
- The on/off controller will control the level set by you in iW
- Play around with these values and see what they do

The Watertank Project

- The first goal is to program an ON/OFF controller
- The second goal is to program a PID controller
- The third goal is to deliver a working program

Back to the project scope

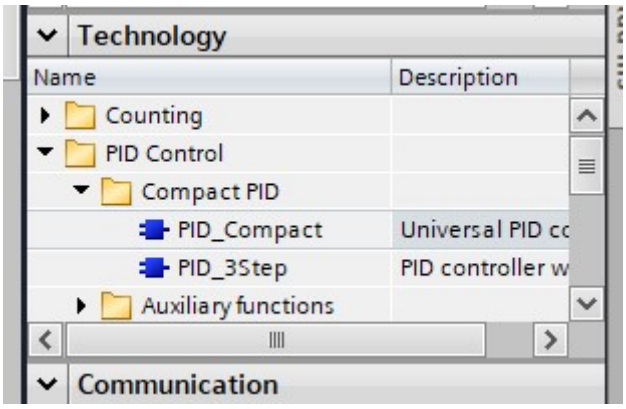
Goal 2 To program a PID controller

Step 1: Open project Ex7-Watertank

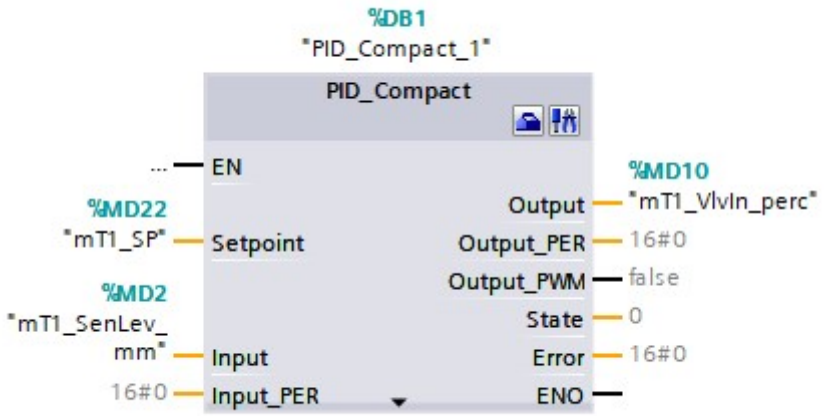
Step 2: Delete network 4 : Level control

Step 3: Create a cyclic interrupt

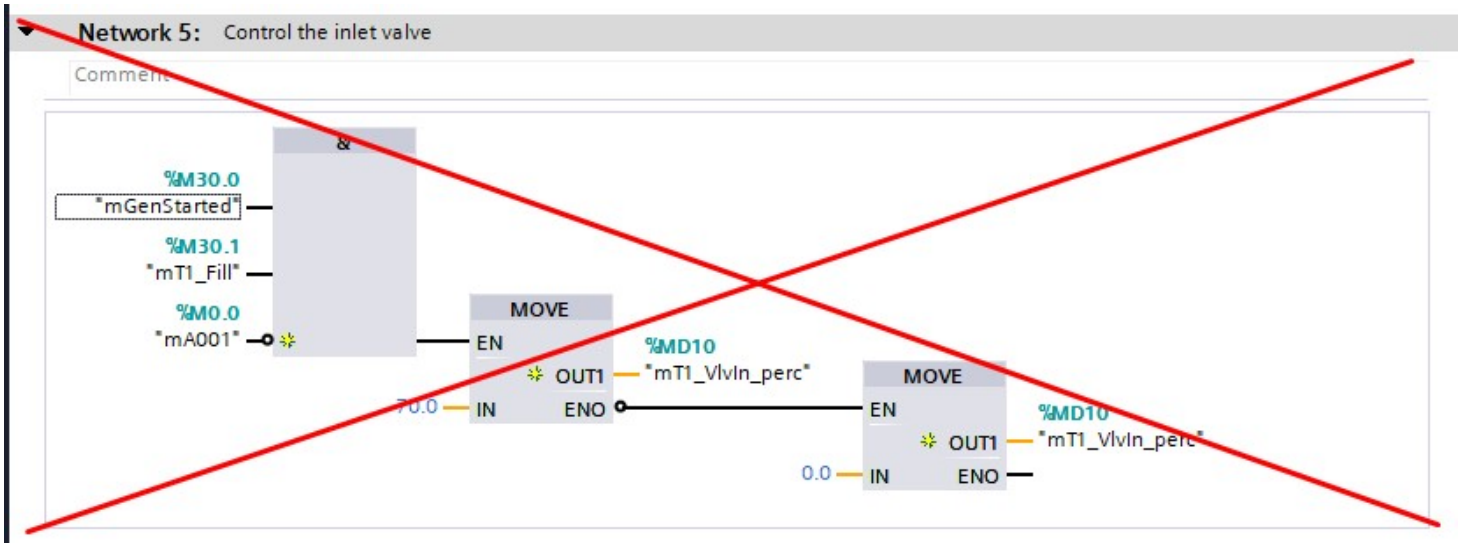
Step 4: Add PID_Compact into *cyclic interrupt*



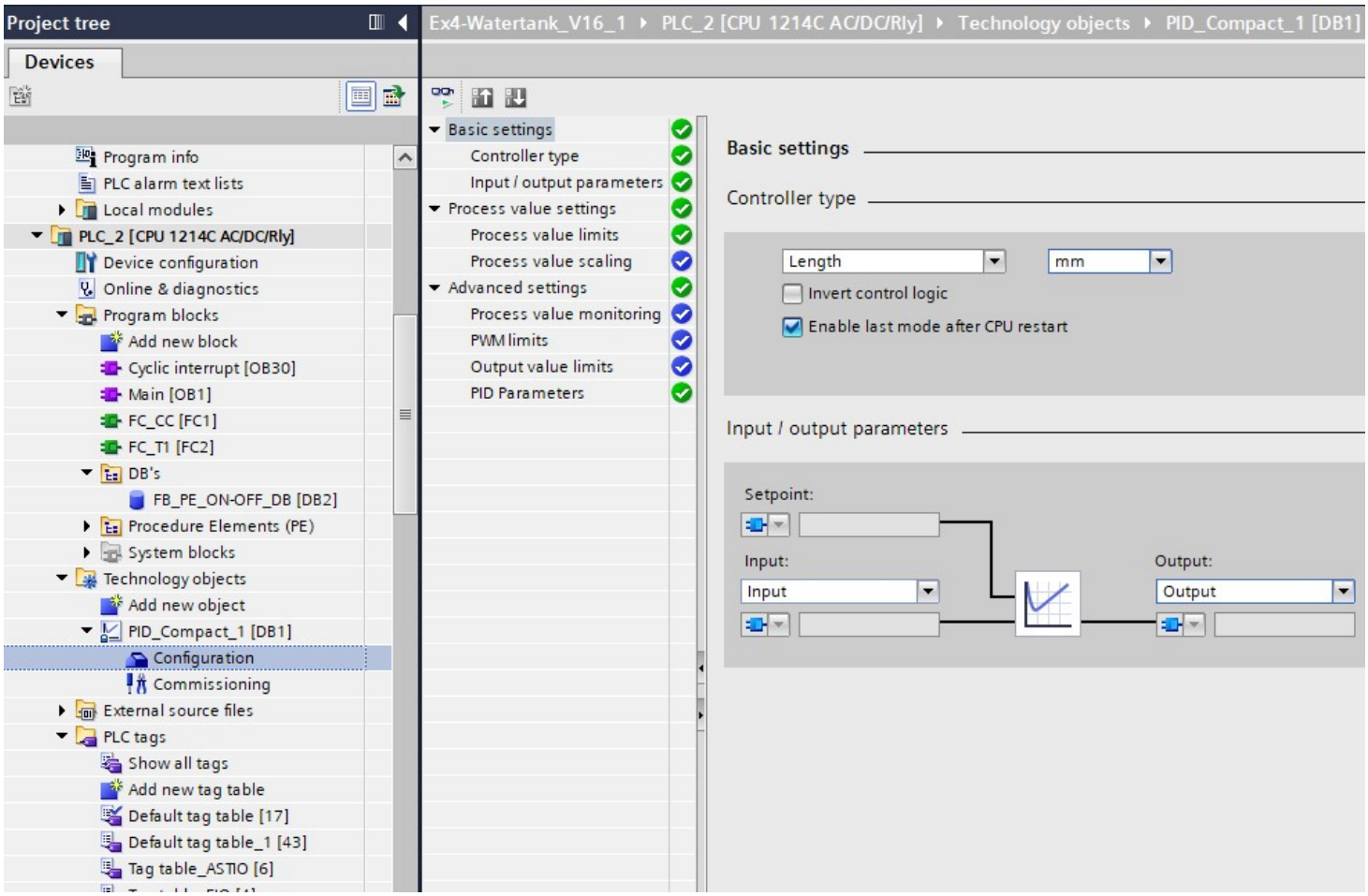
Step 5: Connect the right in- & outputs



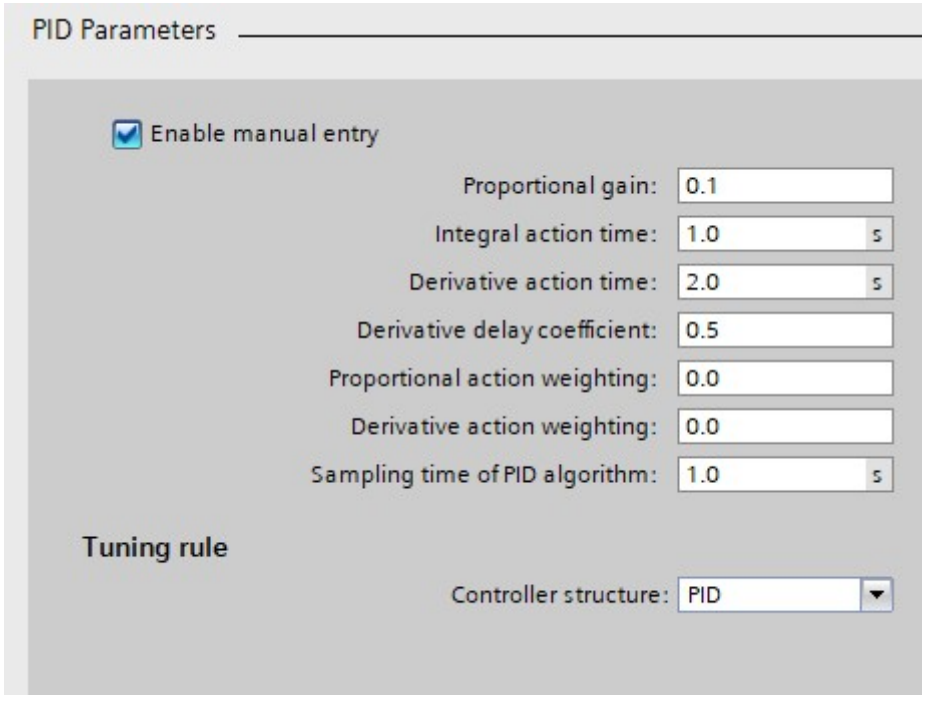
Step 6: Delete network 5 : Control the inlet valve



Step 7: Configure the PID_Compact



Step 8: Play with the following paramaters



Step 9 : Open the FactoryIO scene called:

```
Filename : Level_Control.factoryio
Filelocation : \Documents\Factory IO\My Scenes
```

The Watertank Project

- The first goal is to program an ON/OFF controller
- The second goal is to program a PID controller
- The third goal is to deliver a working program

Back to the project scope

Goal 3 To deliver a working program4

Step 1: Compile the hardware with a rebuild all command

Step 2: Compile the software with a rebuild all command

Step 3: Download hardware and software to the PLC_1

Step 4: Go to Exercise 5 to test the program on a HMI screen

Exercise 5

Study material

Literature

- Addendum 03 HMI
- Addendum 05 Controllers

Equipment

1 Engineering station
2 SIMATIC S7-1200 controller, e.g. CPU 1215C DC/DC/DC – firmware V4.2 or higher
3 SIMATIC STEP 7 software in TIA Portal – V15 SP1 or higher
4 Ethernet connection between engineering station and controller
5 Factory IO scene Pick & Place.factoryio

Additional literature

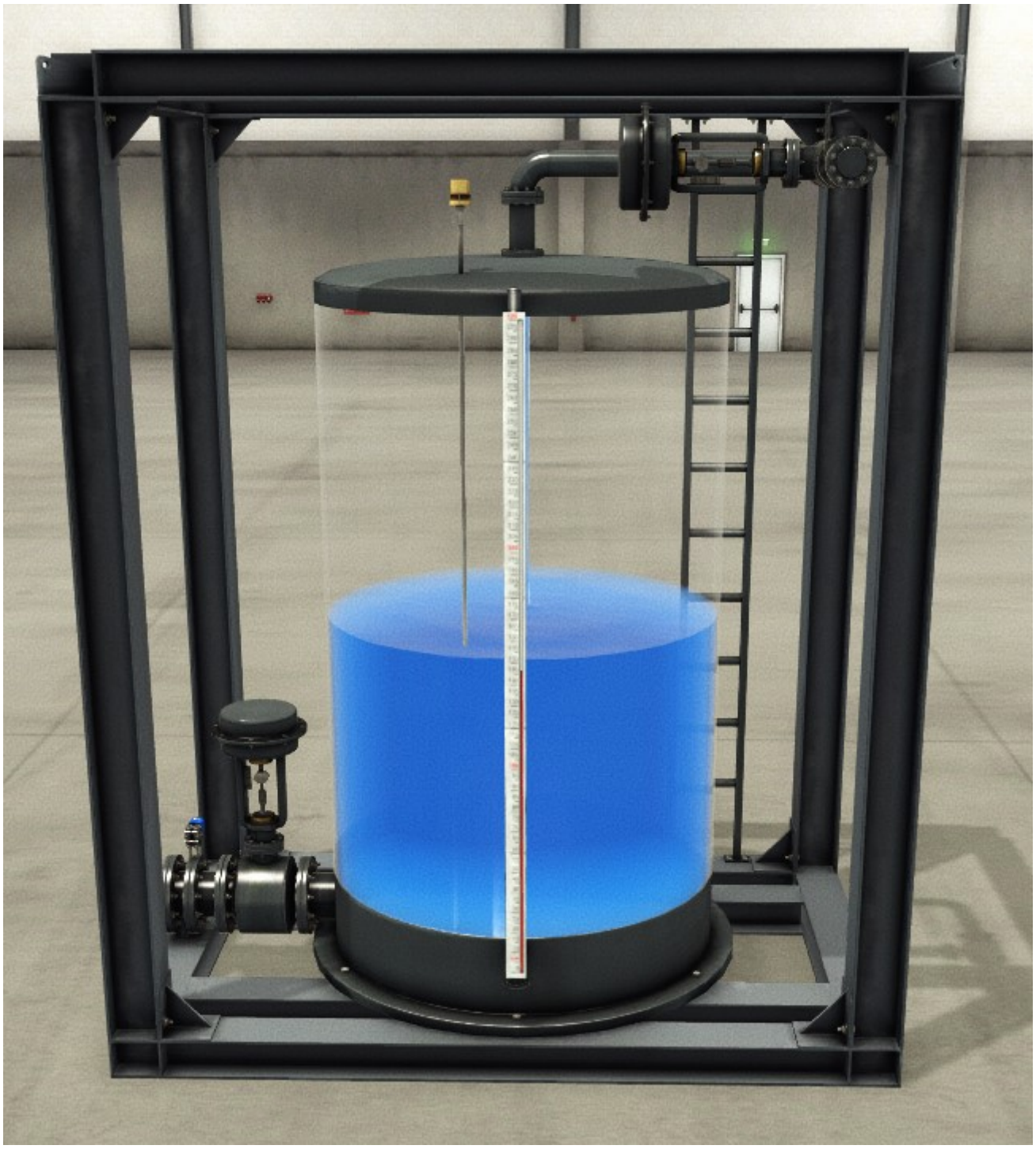
- Real Games Factory IO manual - https://docs.factoryio.com/

The Watertank Project

Scope

Automate controlling the level in **watertank T1**. that is equipped with

- An analog level sensor
- An analog flow sensor on the outlet
- An analog inlet valve
- An analog outlet valve



Use the HMI simulation to control the tank leven and control the PID parameters

Step 1: Open project Ex4-Watertank

Step 2 : Open the FactoryIO scene called:

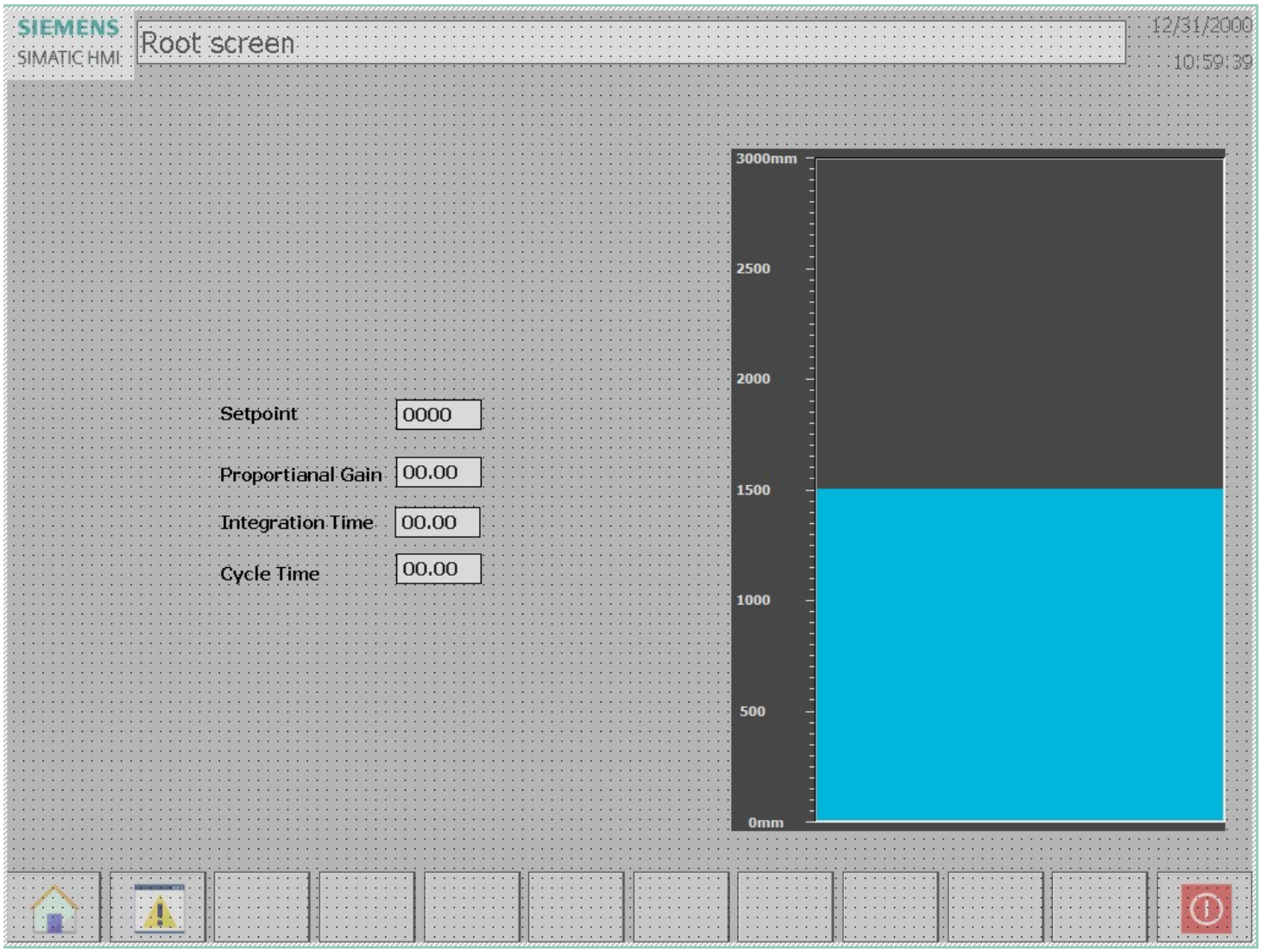
```
Filename : Level_Control.factoryio
Filelocation : \Documents\Factory IO\My Scenes
```

Step 3: Add a TP1500 Basic color PN HMI panel to the project

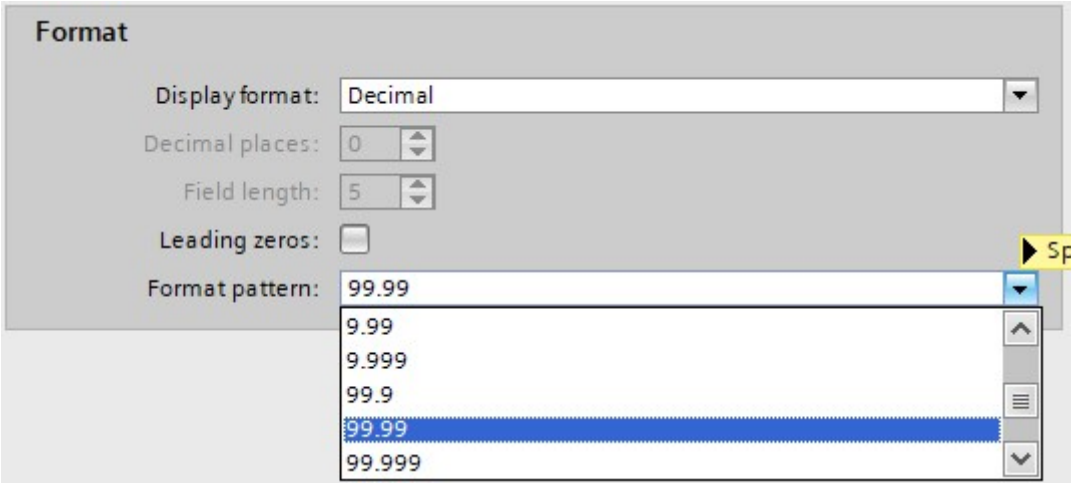
Step 4: Link the HMI to the PLC configured in the project.

```
IP-address                    : 192.168.0.31
IP-address subnet mask        : 255.255.255.0
```

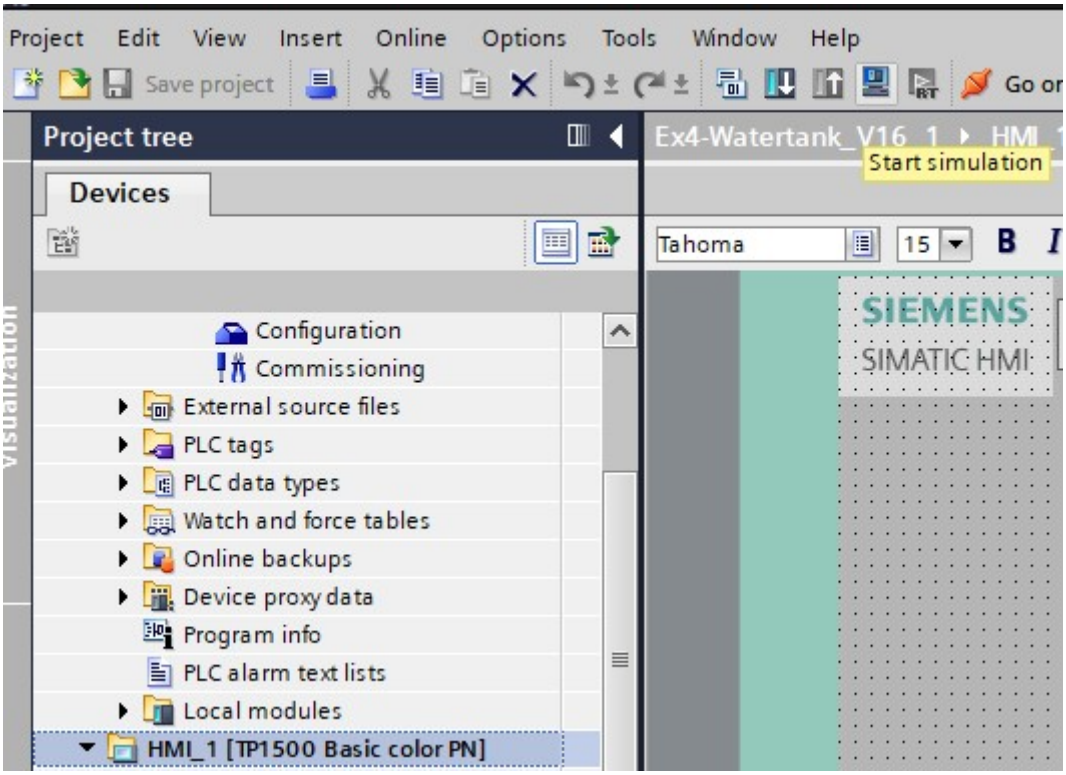
Step 5: Create the following HMI screen, (adjust the PID parameters through HMI)



Remark: to show and use the parameters in 00.00 values in a IO field;



Step 6: Start the HMI simulation



Remark: if the HMI simulation can't find your PLC connection go to the following

```
"C:\Program Files\Common Files\Siemens\CommunicationSettings"
```

Acces point > S7ONLINE > "Your networkcard"

Normal functionallity

- The PID will slowly increase(or decrease) when setpoint changes and ultimately stop at the setpoint
- Play with the PID parameters until it works

1. SFC = Sequential Function Chart, Siemens uses the name GRAPH ↩
2. Debugging = Searching for (programming) faults ↩
3. Within (process)automation this is commonly the automation of an entire machine/installation. The S88 software model divides a machine/installation proces in 3 big parts: ↩
4. NO = Normal open ↩
5. NC = Normal closed ↩
6. LHL = Low / High / Low ↩