Automatisierungstechnik für Mechatroniker (8610718)

2023, Prof. Jörg Wollert (FH Aachen)

Jul 14, 2023

LAB SESSIONS 12:

CHAPTER

ONE

PNEUMATICS & LOGIC USING FLUID SIM

These practical sessions deal with the creation of pneumatic circuits (lab session 1) and the usage of logic modules (lab session 2) to control cylinders and motors.

Note: Please prepare all the `preparation` sections before the actual lab session. This is obligatory for attending and will save you time during the session. Preparation may include drawing diagrams, answering questions, or reading through the basics.

1.1 Learning Outcome

1.1.1 Introduction

This practical session deals with the planning and implementation of pneumatic and electric circuits using the software FluidSim.

1.1.2 Requirements

- *Introduction to Pneumatics*
- *Festo FluidSIM*

1.1.3 What you need

Hardware

- MecLab programming stations

Software

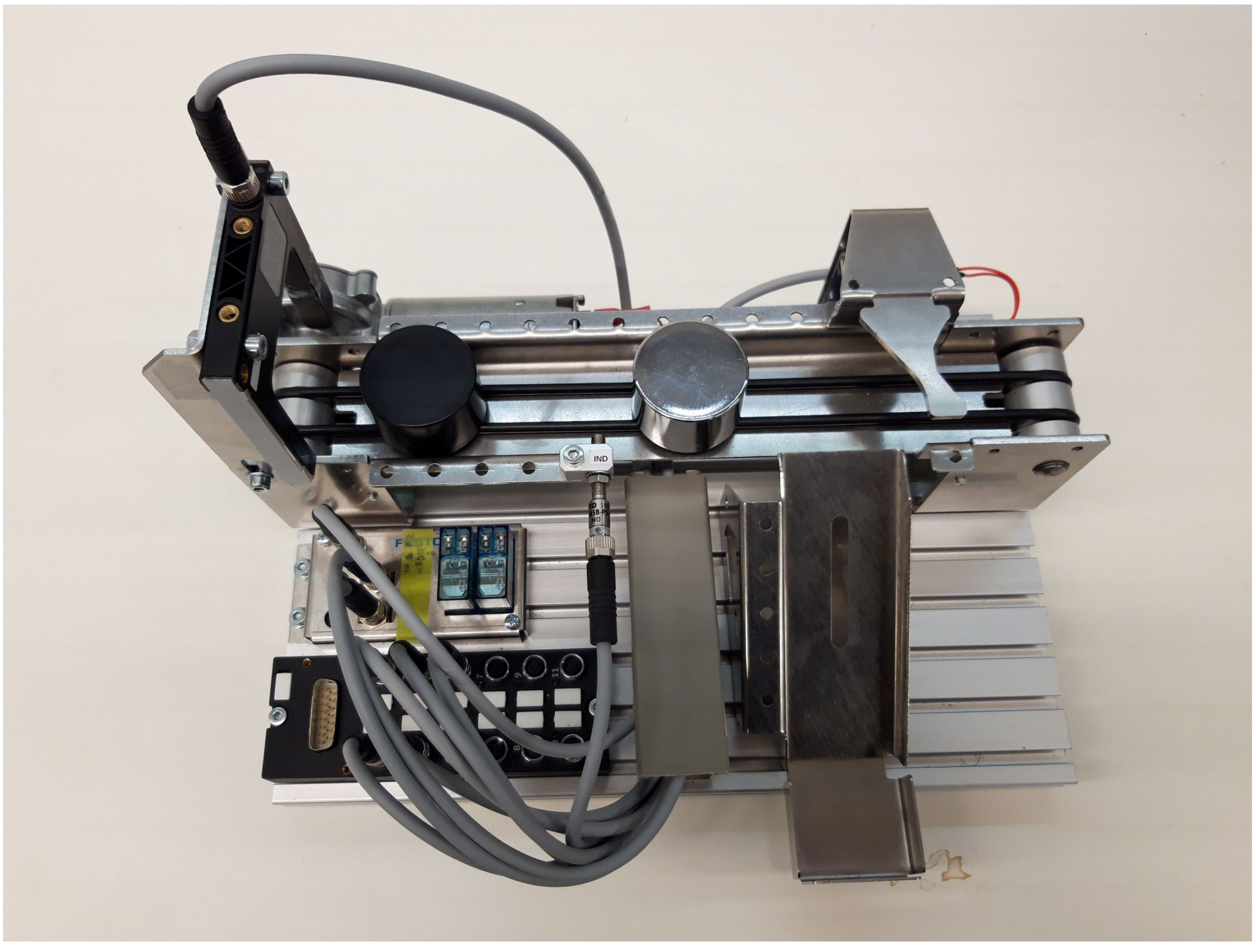
- Festo FluidSim (pre-installed on lab PC)

1.2 Preparation: Festo Programming Stations

The following sections are required to understand the station functionalities.

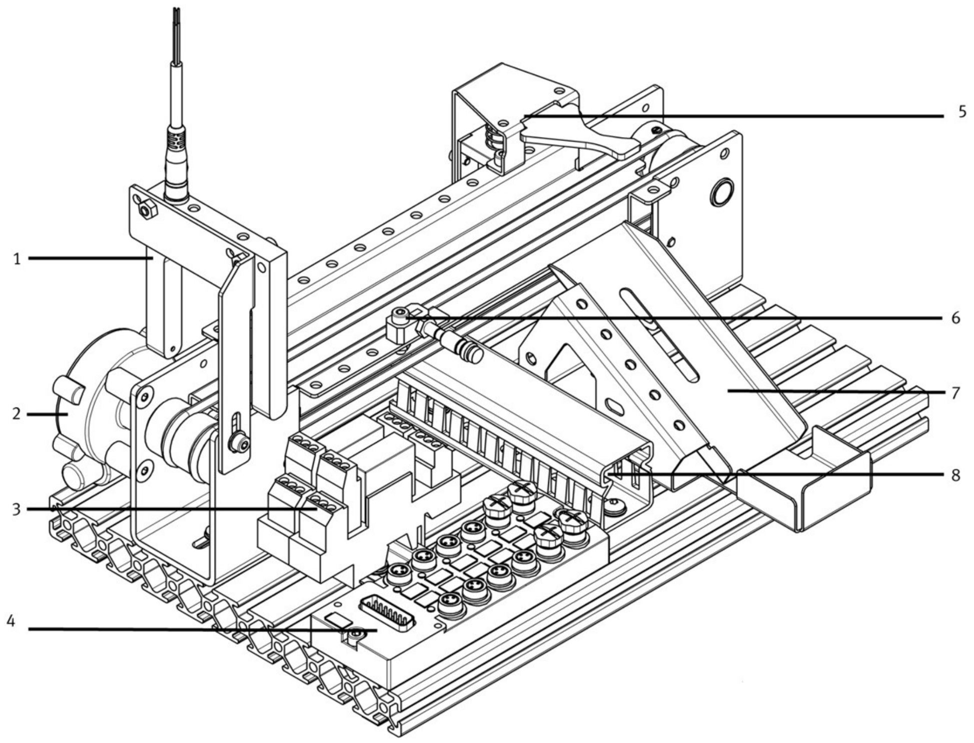
An overview of the stations is provided in this video: https://www.youtube.com/watch?v=Jav9SERe0sE

1.2.1 Conveyor belt



Todo: Assign the correct designation to the components and describe their task in the station.

Todo: Create a schematic sketch of the conveyor belt.



1.2.2 Stack magazine

Todo: Assign the correct designation to the components and describe their task in the station.

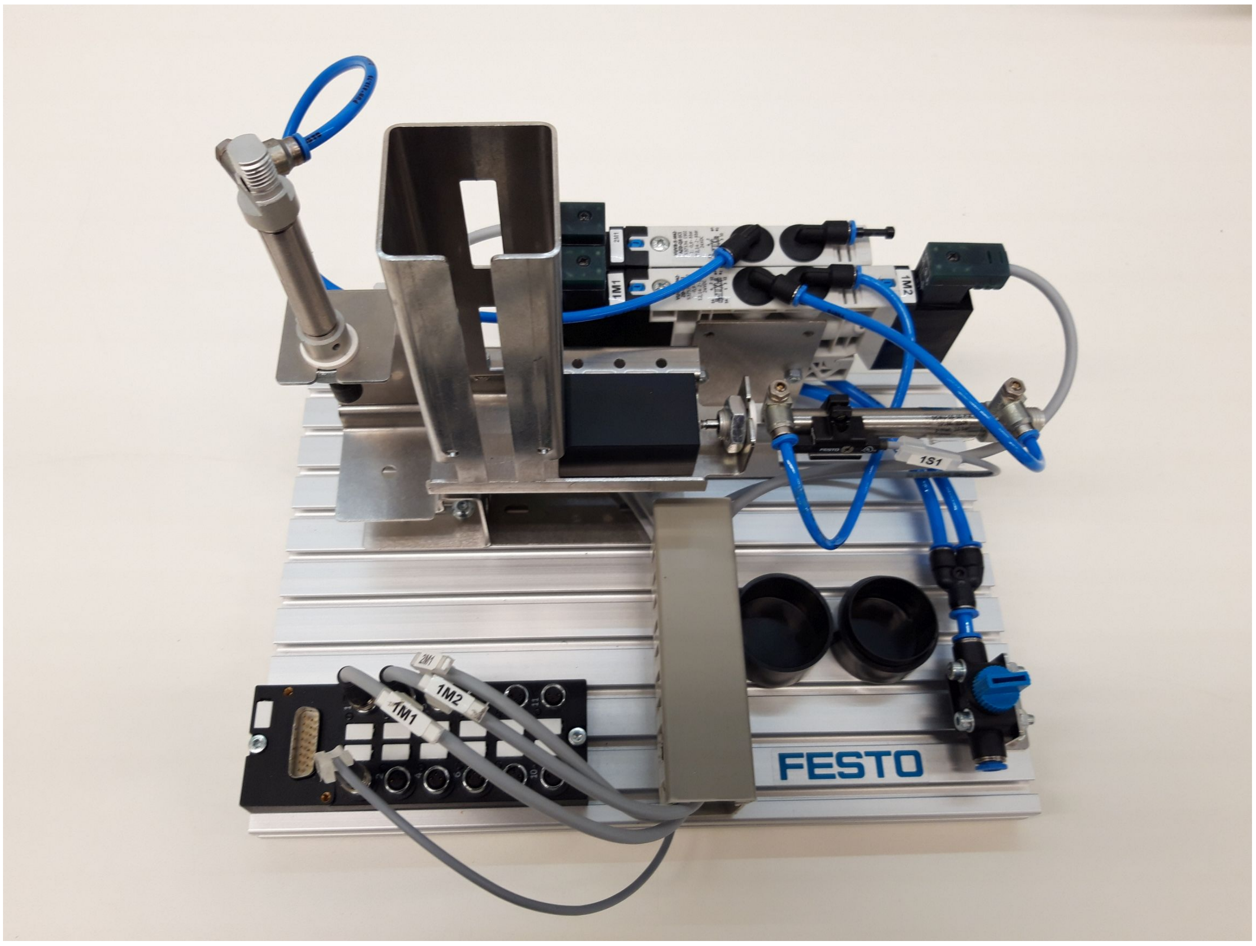
Todo: Create a schematic sketch of the stack magazine.

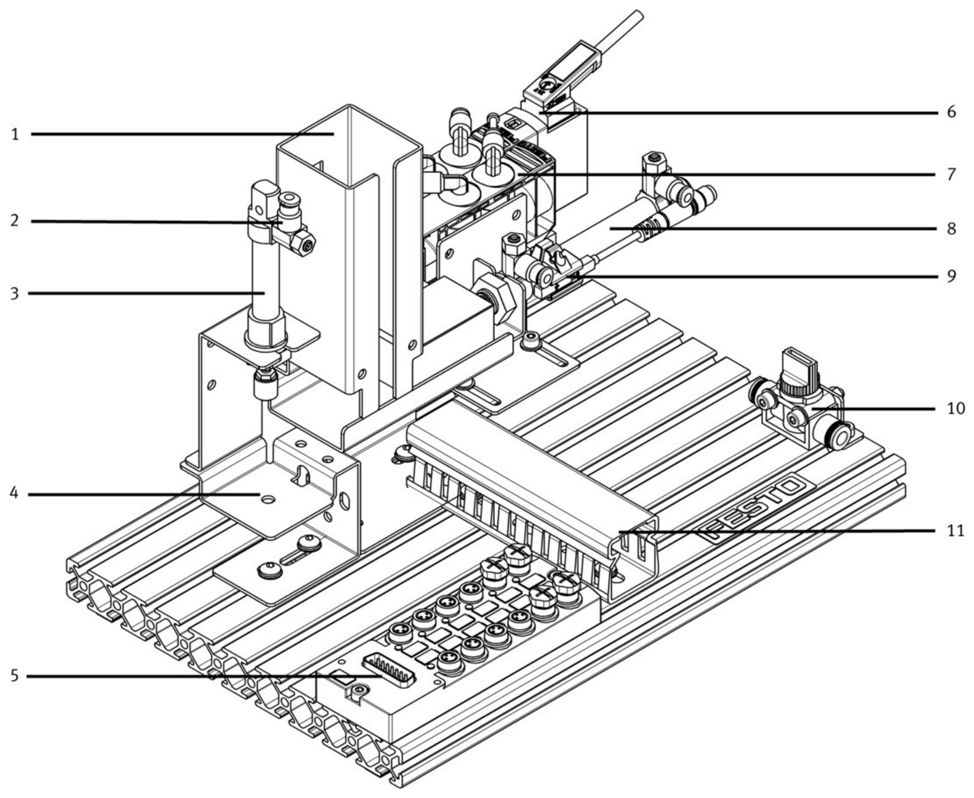
An important function of the stacking magazine station is the pressing of can and lid. A control system is to be designed for this purpose. A vertically arranged pneumatic cylinder is to be used for pressing in, which is supplied with air by a solenoid valve and controlled by the PC. The cylinder is to extend at the push of a button and remain extended as long as the button remains pressed. An important boundary condition is that, for safety reasons, the cylinder also returns to the upper end position in the event of a power failure.

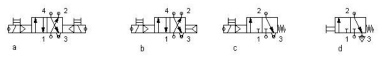
Todo: Which type of cylinder would you use for the described pressing process (single or double acting cylinder) and why?

Todo: Which one of the following valves would you use?

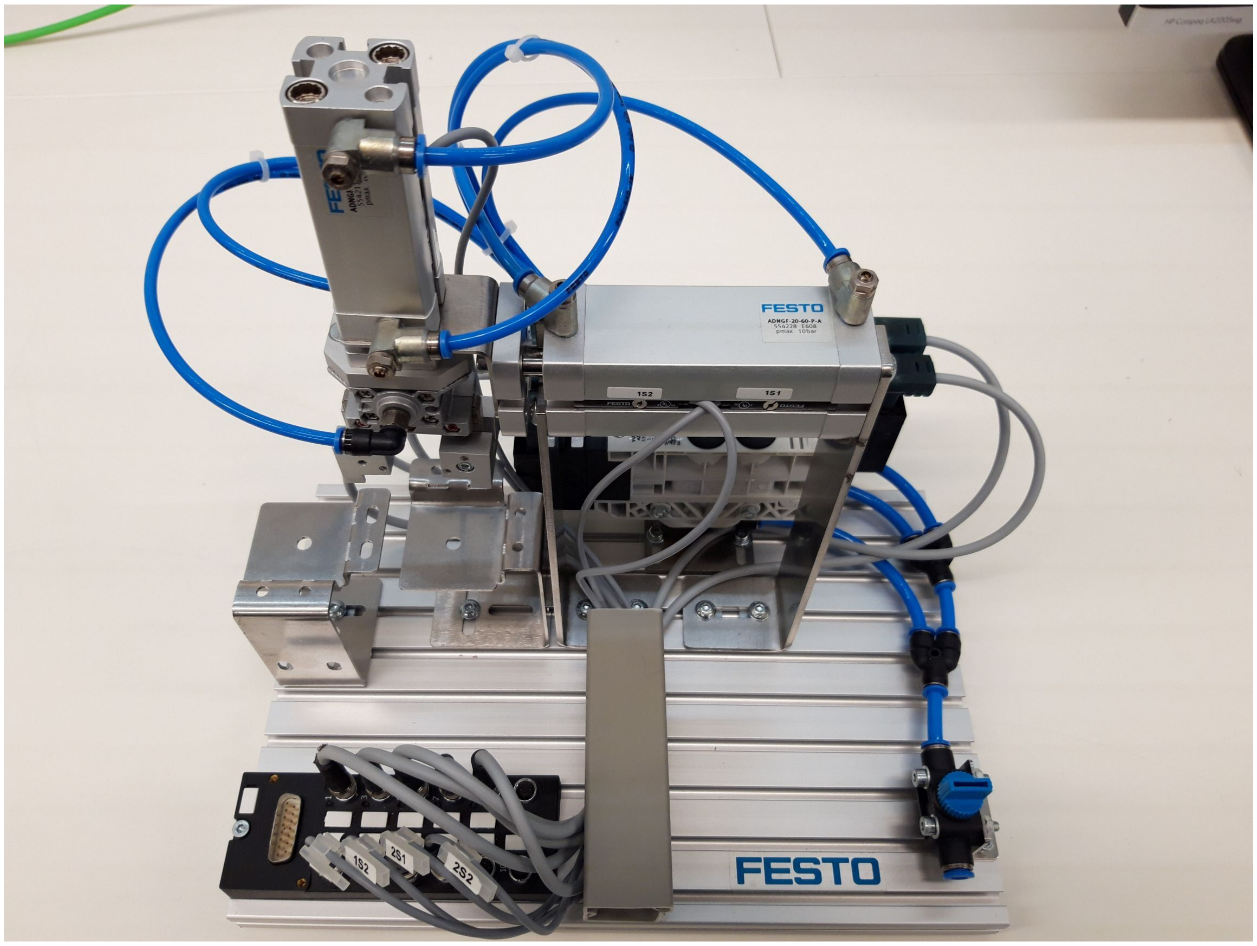
Todo: Design a pneumatic circuit diagram from the selected components and test the function in the simulation. Use FluidSIM® for this purpose. Do not forget the compressed air source and a manual push button to start the cylinder. Test the circuit in simulation mode by clicking on the manual override of the valve with the mouse.


1M1
1M2
1S1
2M1
FESTO


1
2
3
4
5
6
7
8
9
10
11
FESTO


a
b
c
d

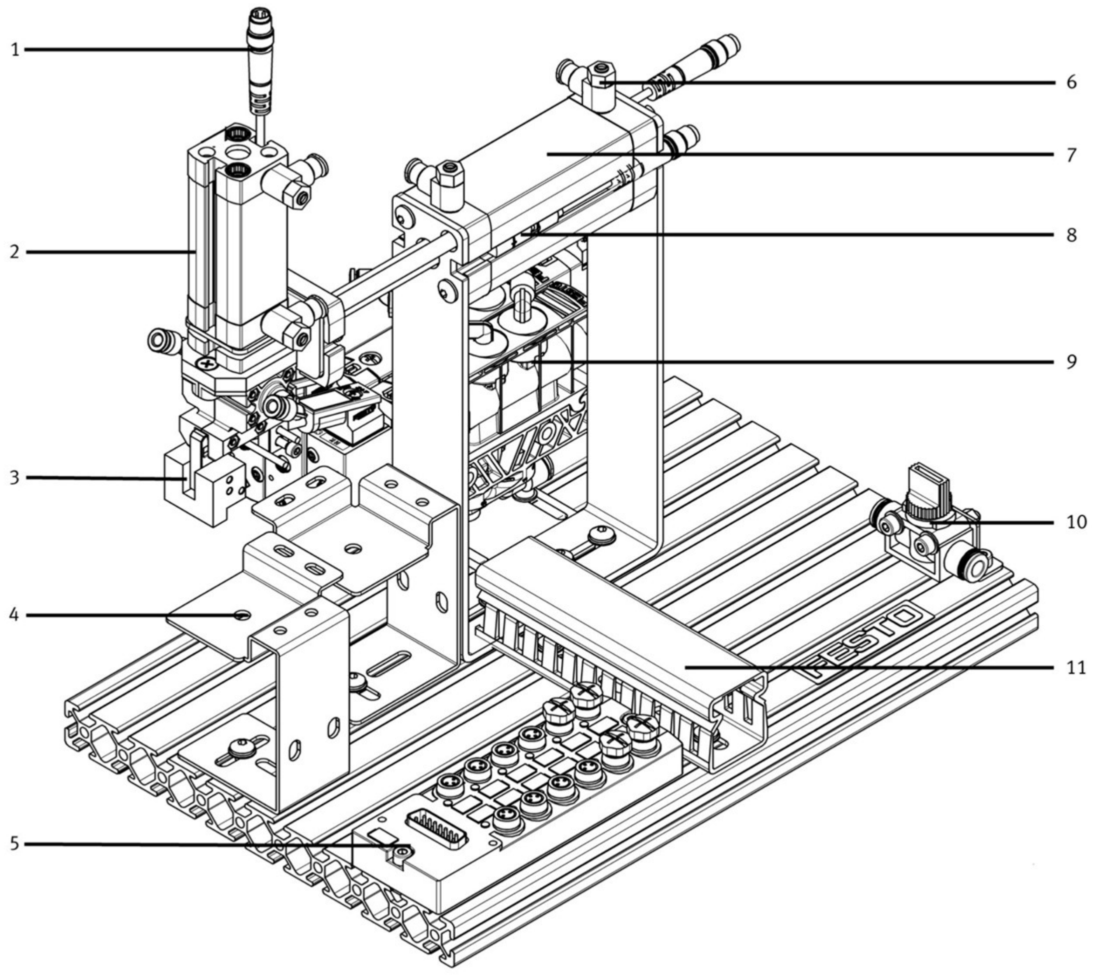
1.2.3 Handling station



Todo: Assign the correct designation to the components and describe their task in the station.

Todo: Create a schematic sketch of the handling station.

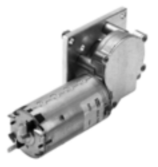
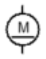
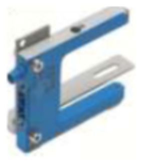
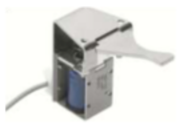
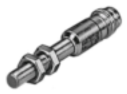
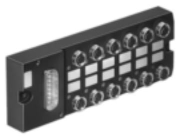
Todo: Design a pneumatic circuit diagram from the selected components and test the function in the simulation. Use FluidSIM® for this purpose. Do not forget the compressed air source and a manual push button to start the cylinder. Test the circuit in simulation mode by clicking on the manual override of the valve with the mouse.


1
2
3
4
5
6
7
8
9
10
11

1.3 Preparation: Basic Electronic Parts

The following section contain an introduction to the basic electronic parts used during the practical sessions. Please carefully execute the todos as this knowledge is required for the practical session.

Todo: Assign the correct symbols and descriptions to the following components.

Component	Symbol	Description
1 		Motor
2 		
3 		
4 		
5 		

The DC motor is one of the most important drives of all. It is used in many consumer electronics devices, household appliances, toys and industrial machines. In this task, a control for this type of motor is to be developed.

Todo: Find out how the DC motor and the solenoid work and answer the following questions:

a) What must be done to change the direction of the motor?

b) Can the direction of the solenoid be changed?

Program logic can be created by using relays in different designs.

Todo: Sketch the symbols for pushbuttons (Taster), switches (Schalter), make contacts (Schließer), break contacts (Öffner), and changeover contacts (Wechsler). What are these components used for?

Todo: Create a circuit in FluidSIM® with which the DC motor can be switched on and off manually and the running direction can be changed.

Todo: Extend the previously created circuit so that the DC motor is indirectly switched on and off and reversed via relays.

Solenoids provide an electric alternative to pneumatic cyclinders. They can operate fast and are controlled easily.

Todo: Create a control of a solenoid coil with a pushbutton and a relay in FluidSIM®. To do this, complete the circuit diagram shown below. (The solenoid is represented by the pneumatic circuit.)

For safety reasons, a two-hand control is often used. In this case, a machine may only start when two buttons are pressed. This is to prevent the machine operator from reaching into the machine with one hand while it is operating.

Todo: Create a circuit for two-hand control of a single-acting cylinder. Test the circuit in the simulation mode of FluidSIM®. Could this function also be implemented with switches?

Todo: Check your options in FluidSim to create time-based actions (e.g., wait for 10s till returning the cylinder again).

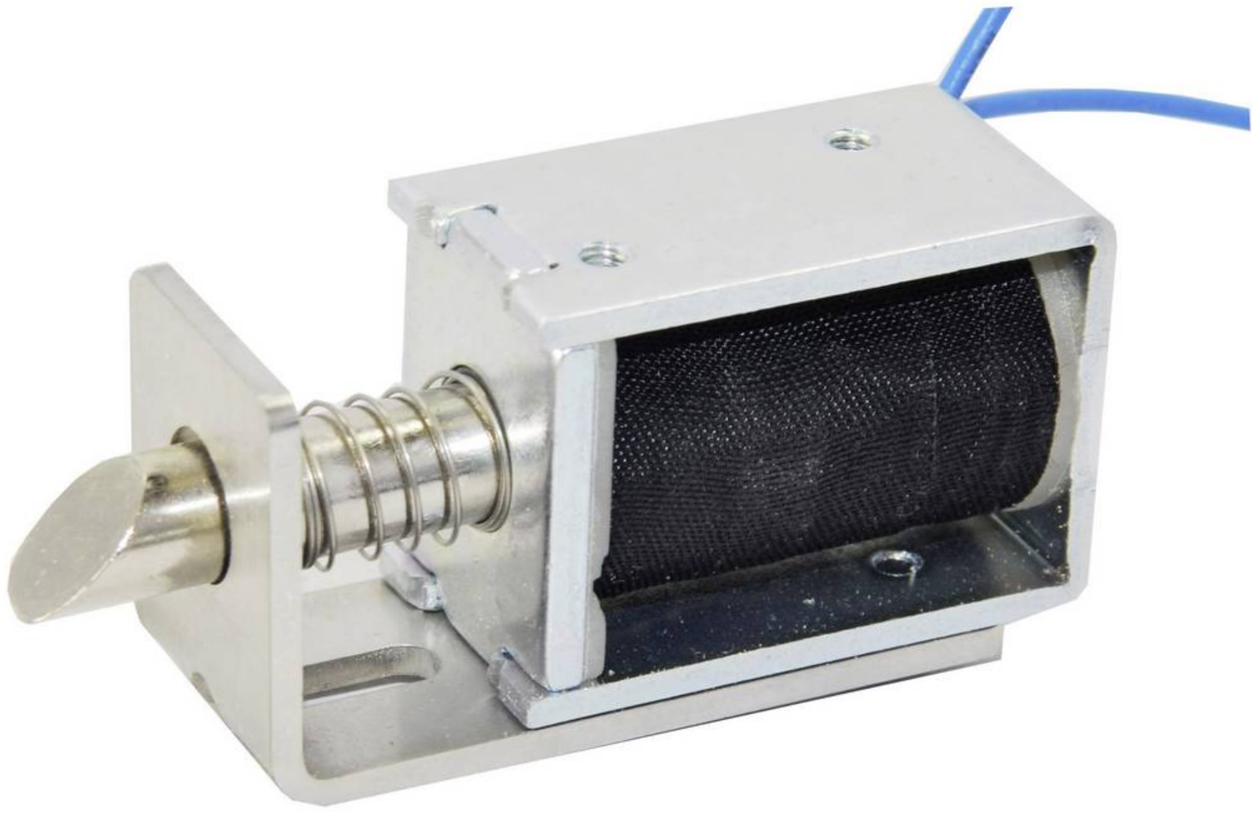
1.4 Preparation: Basic Pneumatic Elements

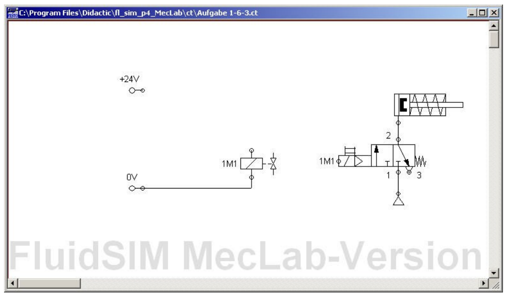
The following sections contain an introduction to pneumatic elements and their usages. Please carefully execute the todos as this knowledge is required for the practical session.

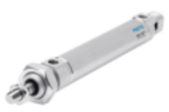
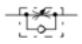
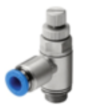
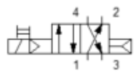
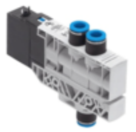
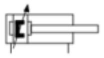
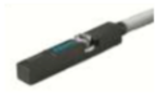
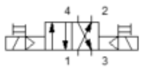
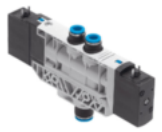
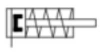
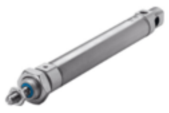
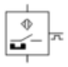
Todo: Match the images, sketches, and descriptions accordingly.

Todo: Create a basic sketches with an air source, a single acting cylinder, and one valve. The valve should control the cylinder.

Todo: What do you have to change if you want to use a double-acting cylinder?




C:\Program Files\Didactic\fl_sim_p4_MecLab\ct\Aufgabe 1-6-3.ct
+24V
0V
1M1
1M1
2
1
3
FluidSIM MecLab-Version

Component	Symbol	Description
1 	[ ] 	[ ] One-way flow control valve Drossel-Rückschlagventil
2 	[ ] 	[ ] Double-acting cylinder Doppeltwirkender Zylinder
3 	[ ] 	[ ] 4/2-way solenoid valve, monostable 4/2-Wege-Magnetventil, monostabil
4 	[ ] 	[ ] Single-acting cylinder Einfachwirkender Zylinder
5 	[ ] 	[ ] Magnetic proximity switch Magnetischer Näherungsschalter
6 	[ ] 	[ ] 4/2-way solenoid valve, bistable 4/2-Wege-Magnetvenil, bistabil

1.5 Task: Relais-based control

Use FluidSim to solve the following tasks. Please solve **one task** during the practical session.

1.5.1 Conveyor belt

Workpieces are transported in every automated assembly. In the MecLab, a conveyor belt is provided for this purpose. The conveyor belt should not run continuously in order to save energy. Therefore, the conveyor belt should always switch on when a workpiece is placed at the start of the belt and stop when the transport task has been completed. The workpieces can be of any color. Use the sketch from the preparation as a guide.

Todo: Create a circuit in FluidSIM® in which the optical and inductive sensors activate a lamp when they switch. Simulate different switching states. Test different objects (e.g., plastics, metal, coins, hand, paper) and write down the sensors' behaviours.

Todo: How can it be achieved that the belt runs only when a workpiece is in contact?

Todo: Implement the belt functionality in FluidSim using a simulated motor.

Conveying and sorting tasks are important functions in any production. The task is to design a conveyor belt and an associated control program that has the following characteristics: Workpieces (black cans) are to be transported from the beginning of the belt to the end of the belt. The transport should start when a workpiece is placed at the beginning of the belt and stop after the workpiece has left the belt at the other end. Silver workpieces are to be sorted out onto the chute.

Todo: Expand the simulation of the conveyor belt which fulfills the described functionalities. Add an ON/OFF switch for the whole system.

Now, the system works in simulation mode. Thus, the next step is to test everything in real life.

Todo: Check the pin assignment at the distributor ("Multipolverteiler").

Todo: Control the real conveyor belt via FluidSim.

1.5.2 Stack magazine

An important function of the stacking magazine station is the pressing of can and lid. A control system is to be designed for this purpose.

A vertically arranged pneumatic cylinder is used for pressing in.

In the first step, the cylinder should extend when a button is pressed and should remain extended as long as the button remains pressed. An important boundary condition is that, for safety reasons, the cylinder also returns to the upper end position in the event of a power failure.

Todo: Create a simulation of the single-acting cylinder which operates the pressing activity when a virtual button is pressed.

The function of the stacking magazine is to store workpieces and eject them individually. The parts are pushed out by a double-acting pneumatic cylinder. A control system should now be developed for this.

Todo: Create a simulation of the double-acting cylinder which ejects the parts. Use one valve and two flow control valves in addition to the cylinder. The flow control valves should be used to adjust the operation speed. After pressing a pushbutton (in the software), the double-acting cylinder extends and moves the workpiece to the stacking position. After pressing another button, the cylinder retracts again.

Sensors are important components of any automated system. In the stacking magazine station, there is a magnetic limit switch which detects the position of the cylinder piston.

A control system for the stacking magazine is to be developed with the following features:

- The operator places a can in the assembly fixture and presses the start button.
- The double-acting cylinder pushes a lid out of the magazine tower (onto the can) and then moves back to the starting position.
- The single-acting cylinder presses the lid into the can for exactly 10 seconds.
- The operator removes the finished workpiece (lid plus can).
- The can and lid may be of any color.

Todo: How can you ensure that the single-acting cylinder does not extend until the double-acting cylinder is fully extended? Which component is required?

Todo: Create a new circuit to simulate the whole functionality of the stack magazine. Add an ON/OFF switch for the whole system.

Todo: Check the pin assignment at the distributor (“Multipolverteiler”).

Todo: Control the real stack magazine belt via FluidSim.

1.6 Preparation: Logic Elements

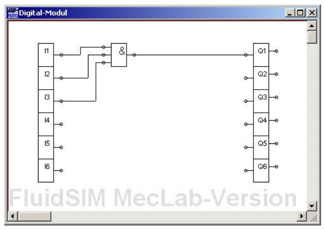
The following section contain an introduction to the logic elements to create logic circuits. Please carefully execute the todos as this knowledge is required for the practical session.

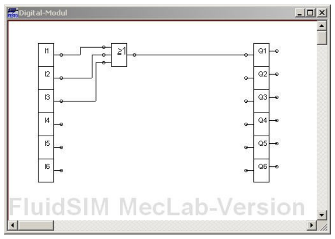
1.6.1 Logic circuits

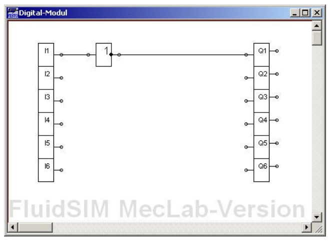
In FluidSIM®, logic modules can be used to control interactions, e.g. by relais. The digital module can be found in FluidSIM® in the components under the Digital Technology category. Place it in a new circuit; double-click on the placed digital module to open a window in which you can edit the functions of the digital module.

Todo: Transfer the following logic circuits into FluidSIM® and examine the behavior of the circuits by starting the simulation and setting the input channels I1 to I3 to the HIGH state by clicking on them.

Additionally, fill in the truth table for the inputs and the output Q1.



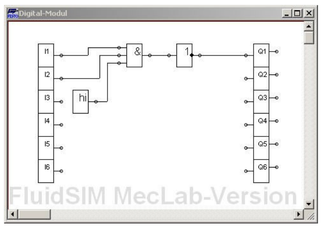




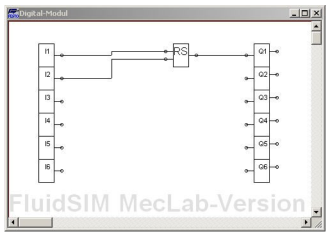
1.6.2 Flip-Flops

Holding elements - or flip-flops - are storing elements. The RS flip-flop is the simplest flip-flop. It has a set input and a reset input.

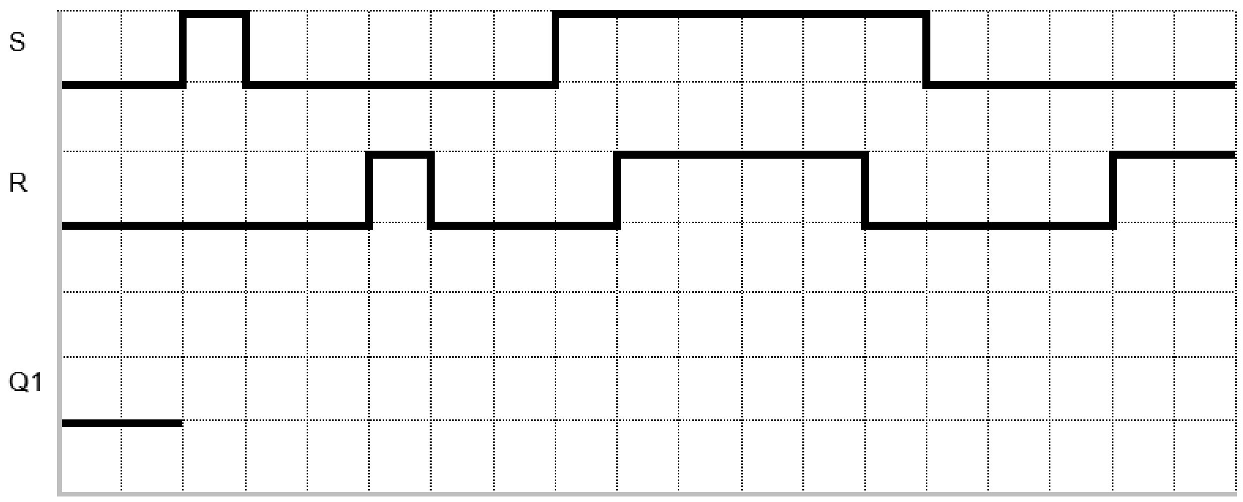
Flags: Often the results of logic operations depend on previous results. These results must be stored temporarily so that they can be used in the next cycle. Flags are used for this purpose.



Todo: Create the logic circuit shown below in FluidSIM®, test the behavior, and describe it. For which control task can the holding element (flip-flop) be used?



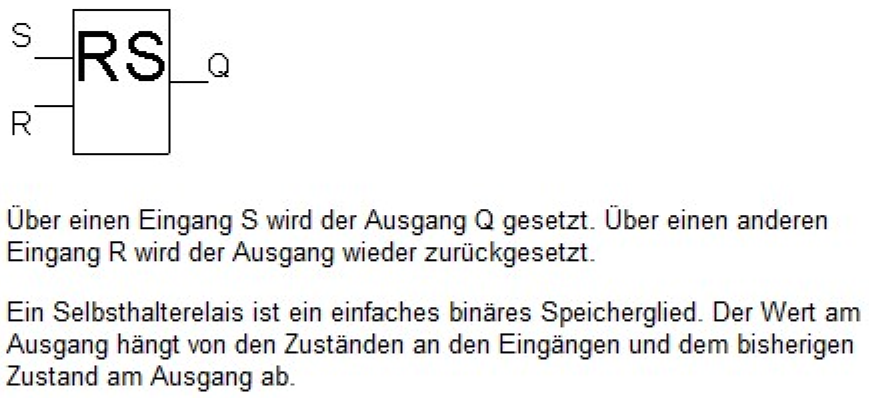
Todo: Plot the Q1 signal waveform on the timing diagram below.



Warning: Attention with FluidSIM: If both S and R are `High`, no dominance is set. **If both inputs are active, the state of the function block is undefined!**

Warning: Depending on literature and PLC manufacturer, the RS and SR function blocks are partly defined differently. Attention: Only from the designation RS or SR one cannot find out whether the setting or the resetting is dominant!

Therefore we use the following notation for the lecture: The dominance is indicated by a “1” at the end of the input name.

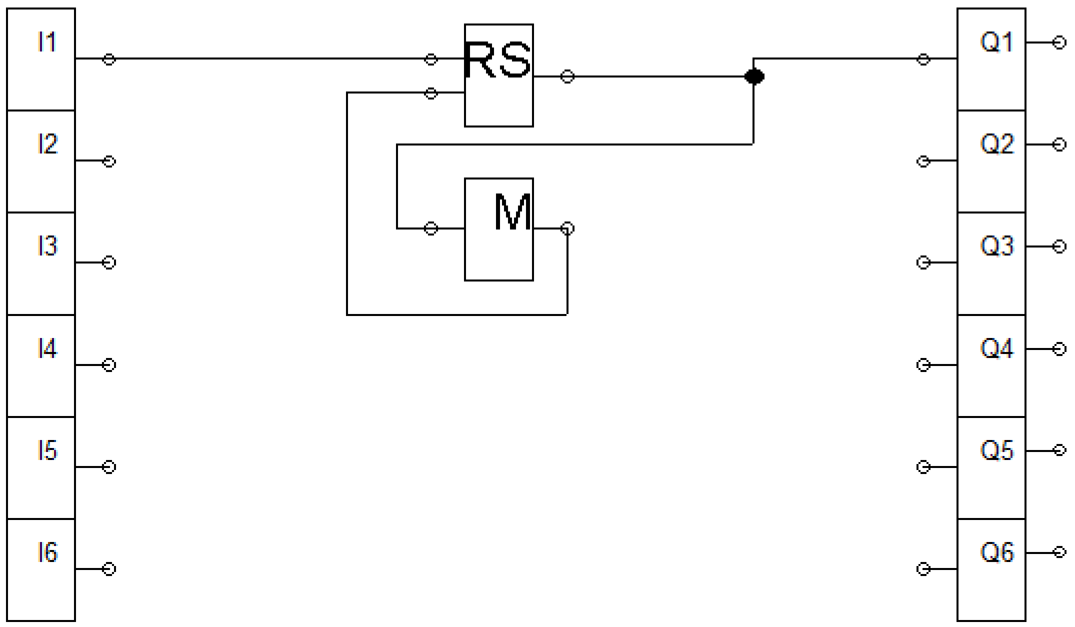


Über einen Eingang S wird der Ausgang Q gesetzt. Über einen anderen Eingang R wird der Ausgang wieder zurückgesetzt.

Ein Selbsthalterelais ist ein einfaches binäres Speicherglied. Der Wert am Ausgang hängt von den Zuständen an den Eingängen und dem bisherigen Zustand am Ausgang ab.

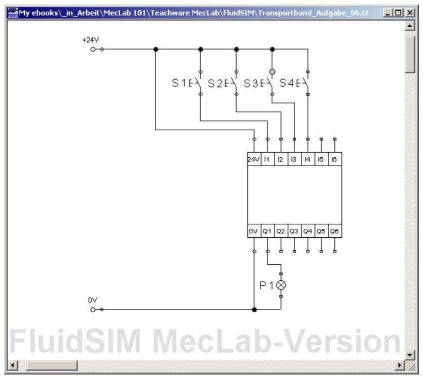
To couple an output signal back to an input, a memory element - a flag - is required. A flag stores the current value and passes it to the output in the next cycle.

Todo: Test the following circuit and create the timing diagram.



1.6.3 Bringing all together

Todo: Create the circuit shown below in FluidSIM®:


My ebooks_in_Arbeit\MecLab 101\Teachware MecLab\FluidSIM\Transportband_Aufgabe_06.ct
+24V
S1
S2
S3
S4
24V
I1
I2
I3
I4
I5
I6
0V
Q1
Q2
Q3
Q4
Q5
Q6
P1
0V
FluidSIM MecLab-Version

Todo: Open the logic module and create a program with the following properties:

The lamp P1 should light up when the buttons S1 and S2 have been pressed simultaneously; thus, light up after push-buttons S1 and S2 have been released again.

The lamp should go out when pushbuttons S3 or S4 have been pressed.

Hint: A button can be pressed permanently by holding down the shift key while clicking.

Todo: Extend the circuit so that an electric motor is switched on and off in addition to the lamp. (Parallel connection!)

1.7 Task: Logic-based control

Use FluidSim to solve the following tasks. Please solve **one task** during the practical session. **Select the station which you have not worked on before.**

1.7.1 Conveyor belt

Workpieces are transported in every automated assembly. In the MecLab, a conveyor belt is provided for this purpose. The conveyor belt should not run continuously in order to save energy. Therefore, the conveyor belt should always switch on when a workpiece is placed at the start of the belt and stop when the transport task has been completed. The workpieces can be of any color.

Todo: How can it be achieved that the belt only runs when a workpiece is in contact?

Todo: Which logic component can be used to ensure that the motor runs as long as the workpiece is transported to the end of the conveyor?

Todo: Implement the belt functionality in FluidSim using a simulated motor and logic elements in the digital module.

Conveying and sorting tasks are important functions in any production. The task is to design a conveyor belt and an associated control program that has the following characteristics: Workpieces (black cans) are to be transported from the beginning of the belt to the end of the belt. The transport should start when a workpiece is placed at the beginning of the belt and stop after the workpiece has left the belt at the other end. Silver workpieces are to be sorted out onto the chute.

Todo: Expand the simulation of the conveyor belt which fulfills the described functionalities. Add an ON/OFF switch or the whole system. **The functionality should be implemented using logic elements in the digital module.**

Now, the system works in simulation mode. Thus, the next step is to test everything in real life.

Todo: Check the pin assignment at the distributor ("Multipolverteiler").

Todo: Control the real conveyor belt via FluidSim.

Todo: What happens if you first place a silver can and shortly afterwards another silver can (distance approx. 3 cm) on the conveyor belt? Is it sorted out correctly?

If not, extend or change the logic diagram in the digital module to achieve more precise sorting. Also test other combinations.

1.7.2 Stack magazine

An important function of the stacking magazine station is the pressing of can and lid. A control system is to be designed for this purpose.

A vertically arranged pneumatic cylinder is to be used for pressing in, which is supplied with air by a solenoid valve and controlled by the PC. The cylinder is to extend at the push of a button and remain extended as long as the button remains pressed. An important boundary condition is that, for safety reasons, the cylinder also returns to the upper end position in the event of a power failure.

Todo: Create a simulation of both cylinders in which the magnetic valves are controlled by buttons. The single-acting cylinder should be operated by one button, the double-acting cyclinder by two buttons. Use flow control valves to adjust the operation speed.

Sensors are important components of any automated system. In the stacking magazine station, there is a magnetic limit switch which detects the position of the cylinder piston.

A control system for the stacking magazine is to be developed with the following features:

- The operator places a can in the assembly fixture and presses the start button.

- The double-acting cylinder pushes a lid out of the magazine tower (onto the can) and then moves back to the starting position.
- The single-acting cylinder presses the lid into the can for exactly 10 seconds.
- The operator removes the finished workpiece (lid plus can).
- The can and lid may be of any color.

Todo: Modify the circuit from the previous task so that the solenoid valves are controlled by the digital module instead of by pushbuttons. Add a start button and the magnetic limit switch of the double-acting cylinder and connect them to the inputs of the digital module.

Todo: Create the circuit for the plant in the **digital module using logic elements**. Test the control in the simulation. Add an ON/OFF switch for the whole system.

Todo: Check the pin assignment at the distributor (“Multipolverteiler”).

Todo: Control the real stack magazine belt via FluidSim.

Todo: What happens,

a) if you press the start button again during the pressing process?

b) if you press the start button permanently?

Could this cause problems with the operation of the station? If so, extend your logic diagram to avoid this. Test again with the help of the simulation.

1.8 Problem Solving

Will be expanded if required.

CHAPTER

TWO

PLC CONTROL OF THE FESTO STATIONS

In these practical sessions you will write PLC programs to control one of the Festo stations. Both the conveyor belt session and the handling station session are mandatory. The order in which you carry out the sessions is not important.

Note: Please prepare all the `preparation` sections before the actual lab session. This is obligatory for attending and will save you time during the session. Preparation may include drawing diagrams, answering questions, or reading through the basics.

2.1 Learning Outcome

2.1.1 Introduction

The inputs and outputs of the Festo stations can be processed and set using a PLC (Programmable Logic Controller). In this module you will:

- create a state machine diagram for the station.
- apply your knowledge of the IEC 61131-3 programming language ST to implement the created state machine

2.1.2 What you need

Software

- e!Cockpit version 1.10.0.1 (pre-installed on lab PC)

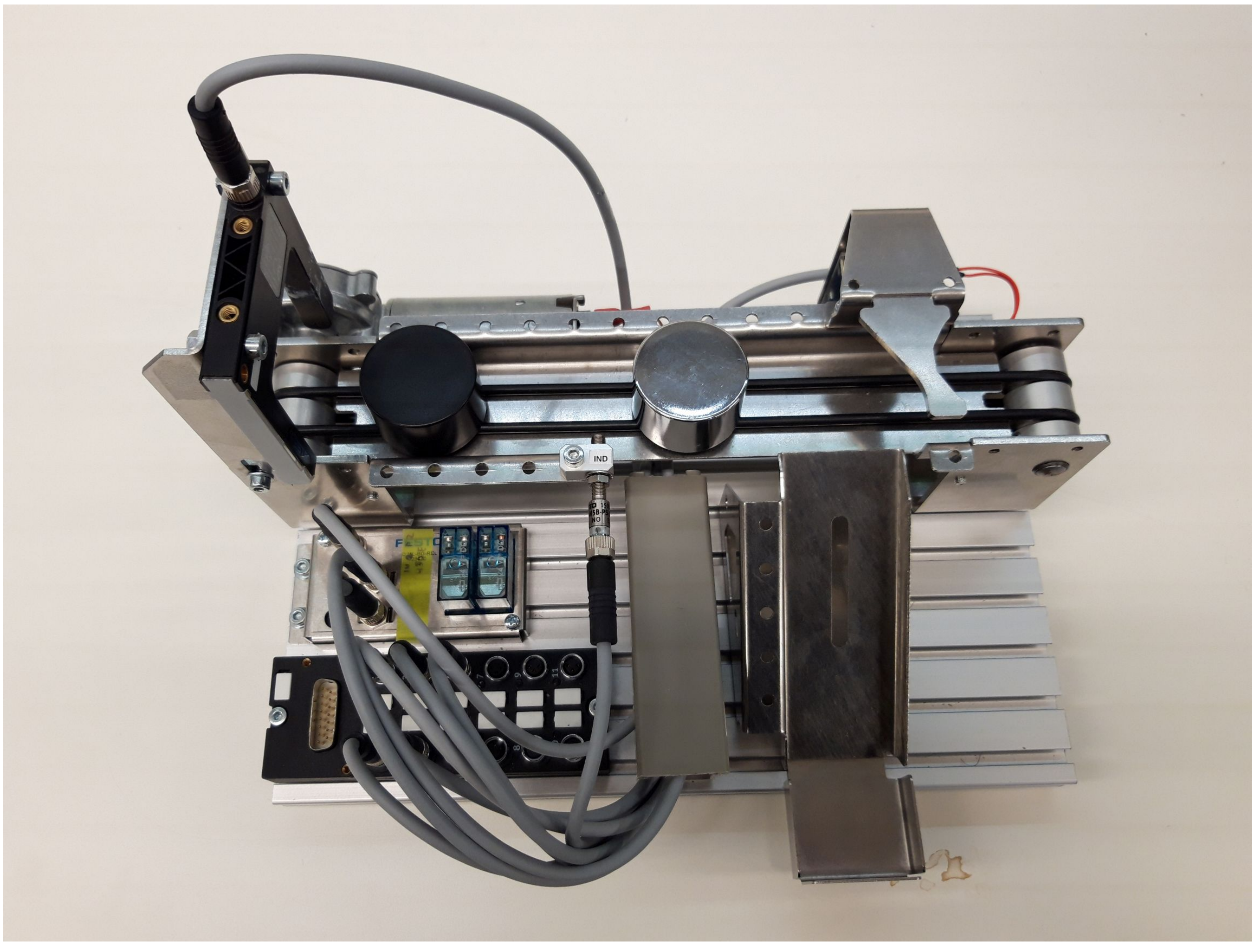
Hardware

- Either the conveyor belt or the handling station.

2.2 Preparation: Conveyor Belt

2.2.1 Goal

- To get to know and describe the structure and functionality of the conveyor belt



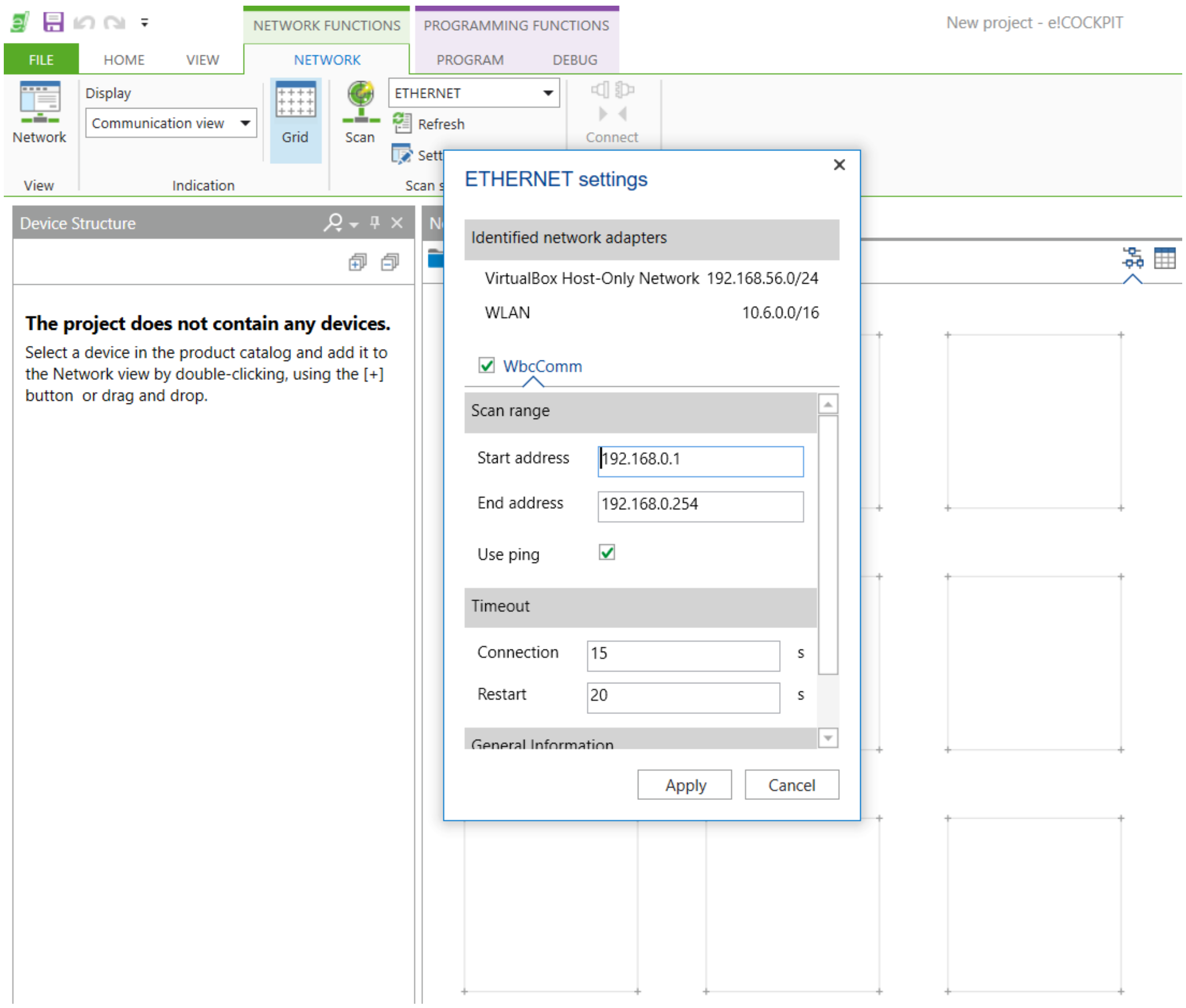
The conveyor should be turned on once a piece is detected by the ligh beam sensor (left side). If the piece is metal, it should be send back to the beginning of the conveyor and the station should wait until the piece is manually taken away. If the piece is not metal, the stopper should be actuated and the piece should be redirected onto the slide.

Todo: Create a state machine diagram of the conveyor belt. Specify what should be done in the `entry`, `do`, and `exit` phases of each state. Define the transition conditions between each state and the following state.

2.3 Assignment: Conveyor Belt

Todo: Scan for and connect to the PLC from inside e!Cockpit. This is how to do it:

Take a look at the PLC rack and read the PLC's IP address. Then navigate to Network - Settings and put in the right IP range to scan. For example, scanning in the range from 192.168.1.1 till 192.168.1.254 will detect all PLC's in this network that have IP addresses within that range. Please note that this scanning process can take multiple trials to detect the PLC. If a PLC is not detected immediately, scan again.



The project does not contain any devices.
Select a device in the product catalog and add it to the Network view by double-clicking, using the [+] button or drag and drop.

ETHERNET settings

Identified network adapters

VirtualBox Host-Only Network	192.168.56.0/24
WLAN	10.6.0.0/16

☑ WbcComm

Scan range

Start address	192.168.0.1	
End address	192.168.0.254	
Use ping	✓	

Timeout

Connection	15	s
Restart	20	s

General Information

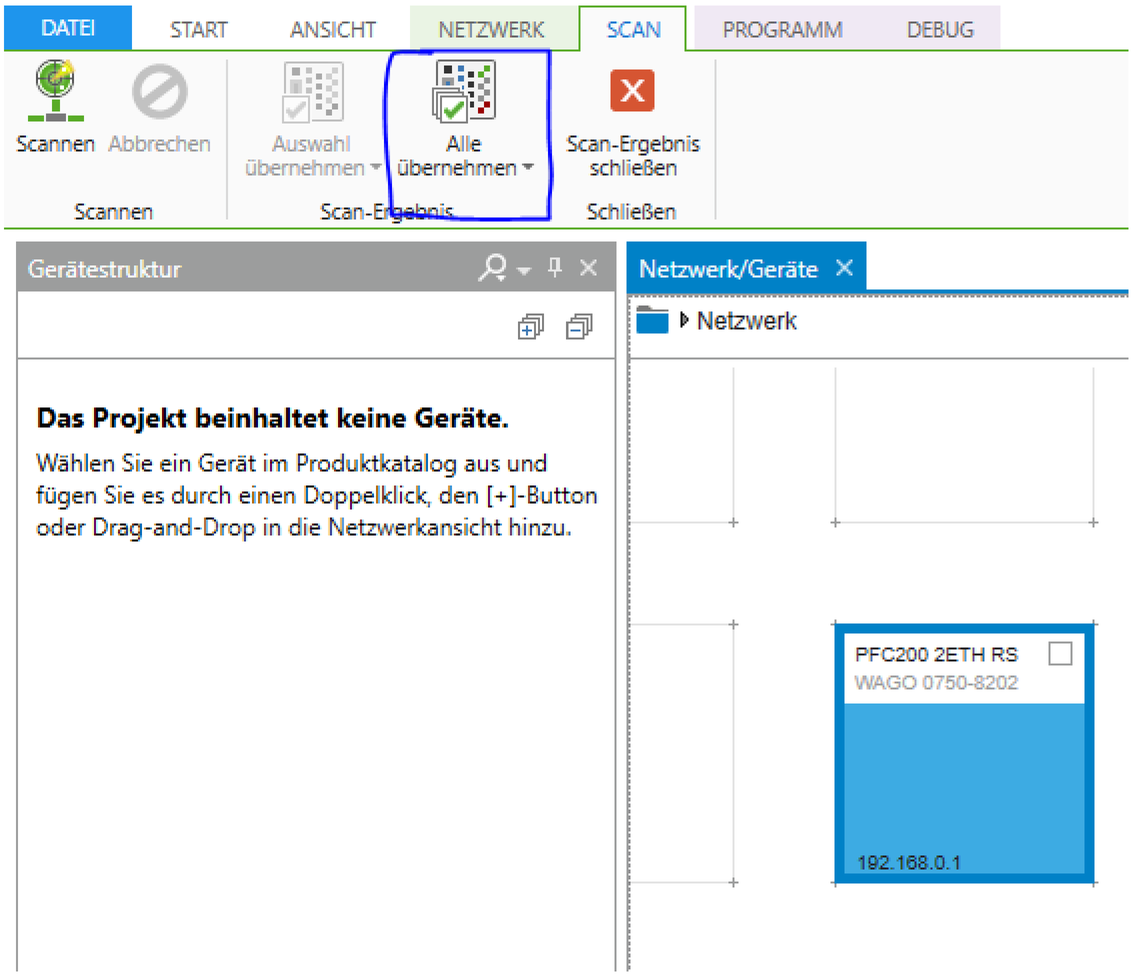
Apply Cancel

After detecting the PLC, click Apply All to add the PLC to your project

Hint: The IP address of the PLC used in this session is 192.168.0.1

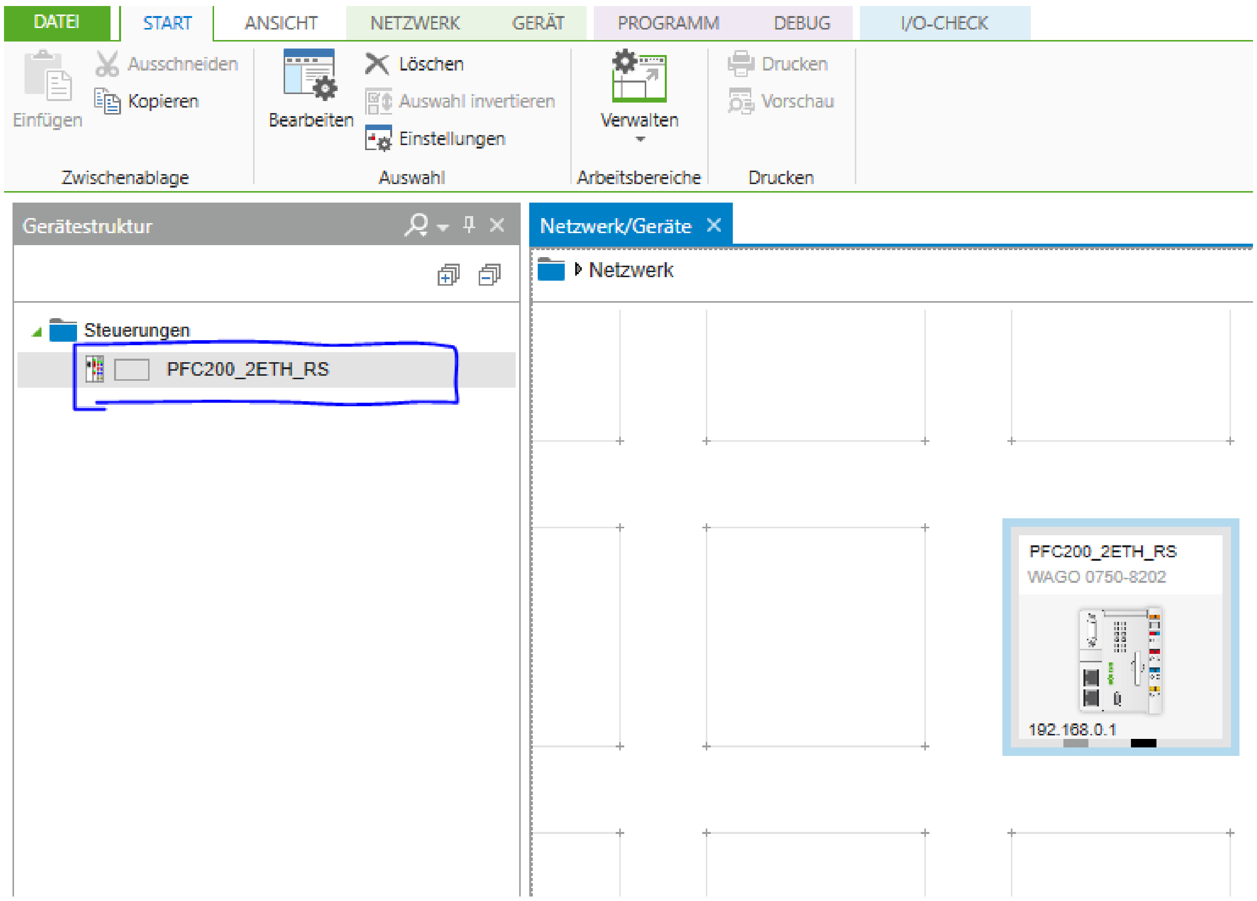
Note that now the PLC has been added to your project.

This has only added the PLC module itself. But the PLC has other modules attached to it that were not yet scanned for. Every module adds new functionality to the PLC. For example, module 750-430 is an 8-channel digital input card. Meaning that the wire-plugs on the 750-430 accept digital sensors like an inductive sensor, light beam sensor, etc.



Das Projekt beinhaltet keine Geräte.

Wählen Sie ein Gerät im Produktkatalog aus und fügen Sie es durch einen Doppelklick, den [+]-Button oder Drag-and-Drop in die Netzwerkansicht hinzu.


DATEI
START
ANSICHT
NETZWERK
GERÄT
PROGRAMM
DEBUG
I/O-CHECK
Einfügen
Ausschneiden
Kopieren
Zwischenablage
Bearbeiten
Löschen
Auswahl invertieren
Einstellungen
Auswahl
Verwalten
Arbeitsbereiche
Drucken
Vorschau
Drucken
Gerätestruktur
Steuerungen
PFC200_2ETH_RS
Netzwerk/Geräte
Netzwerk
PFC200_2ETH_RS
WAGO 0750-8202
192.168.0.1

If you are wondering what a certian module does, search its module number online.

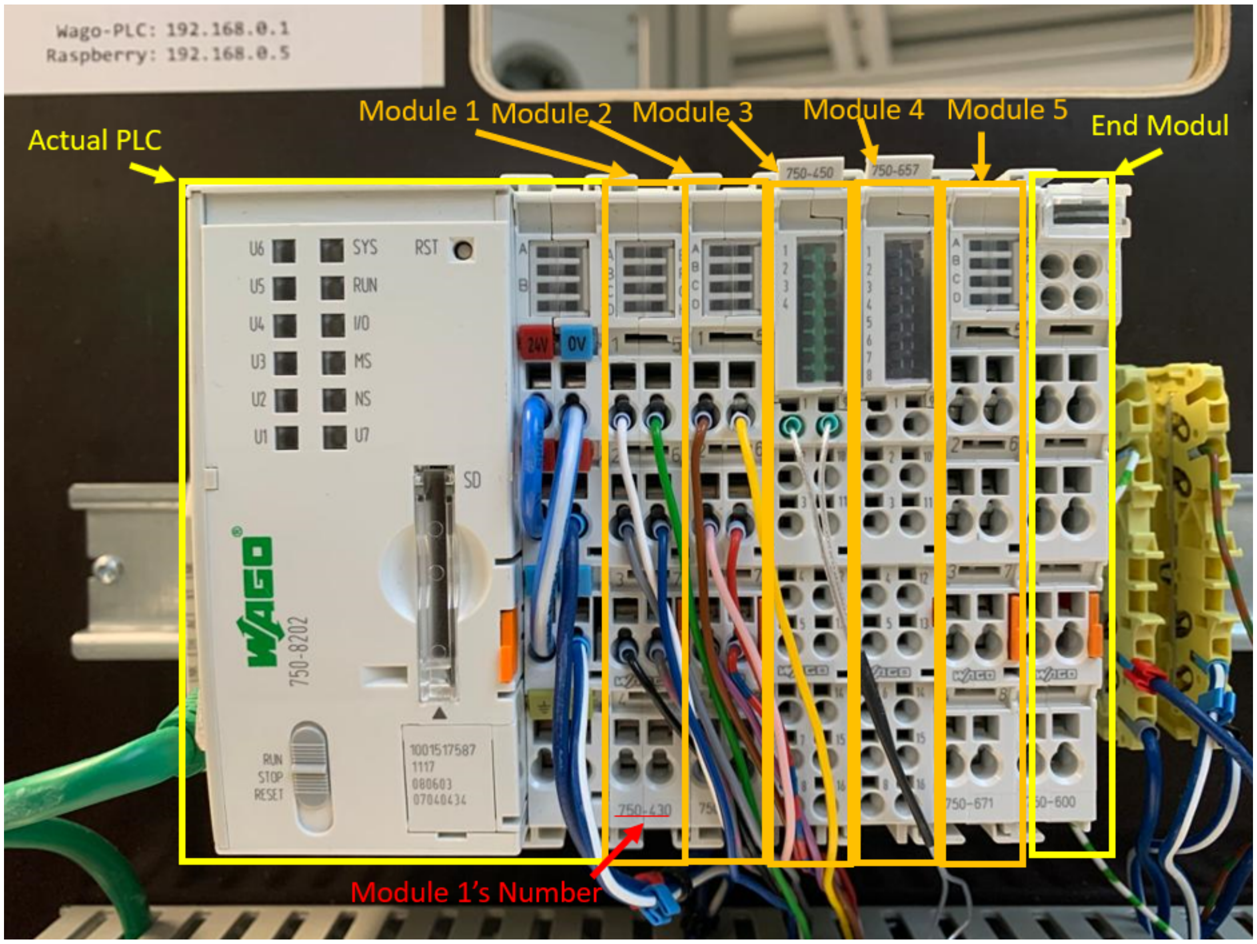


Fig. 2.1: The PLC in this session has 5 modules inserted between the PLC and the End Module. Note that the End Module has to always be there at the end.

Todo: Right click on the newly added PLC and choose scan to scan for the modules inserted next to the PLC.

The scan results should identify 5 modules. Click add all to add all modules to the project

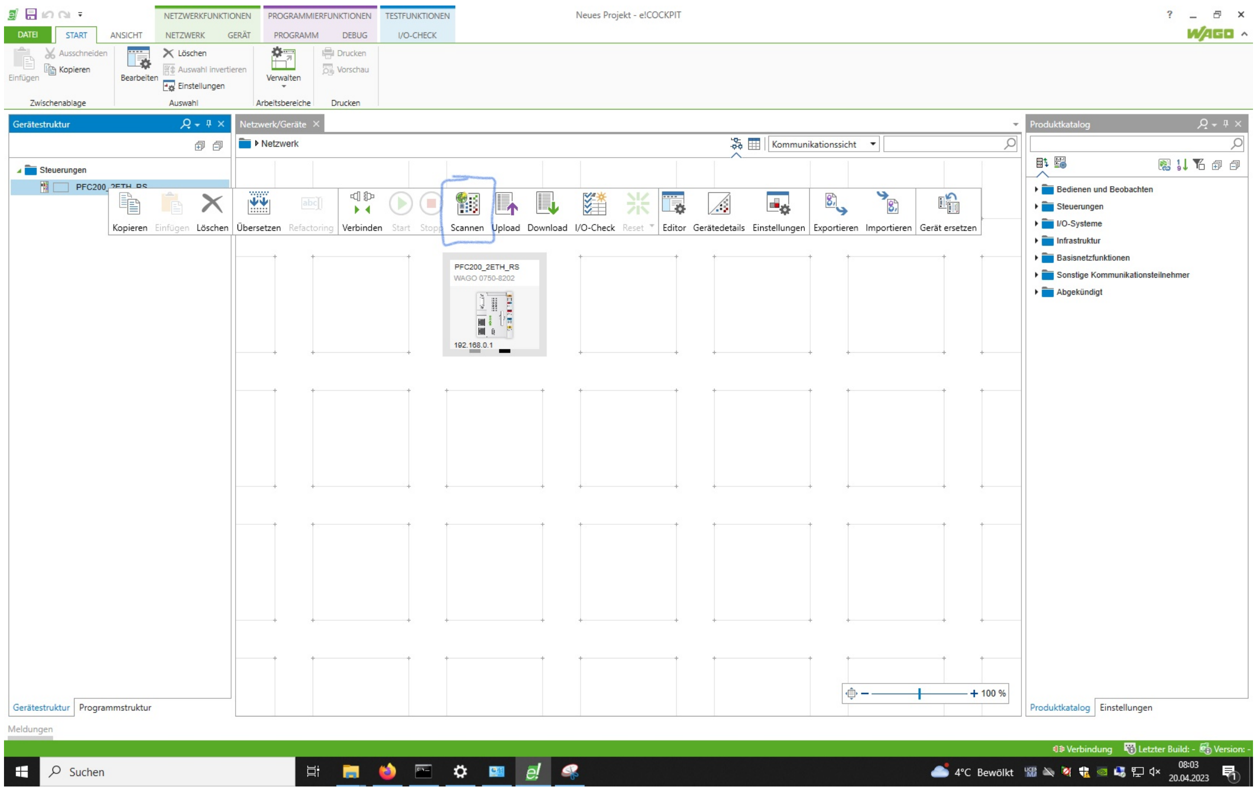
After adding all scanned modules, your device structure should look like this:

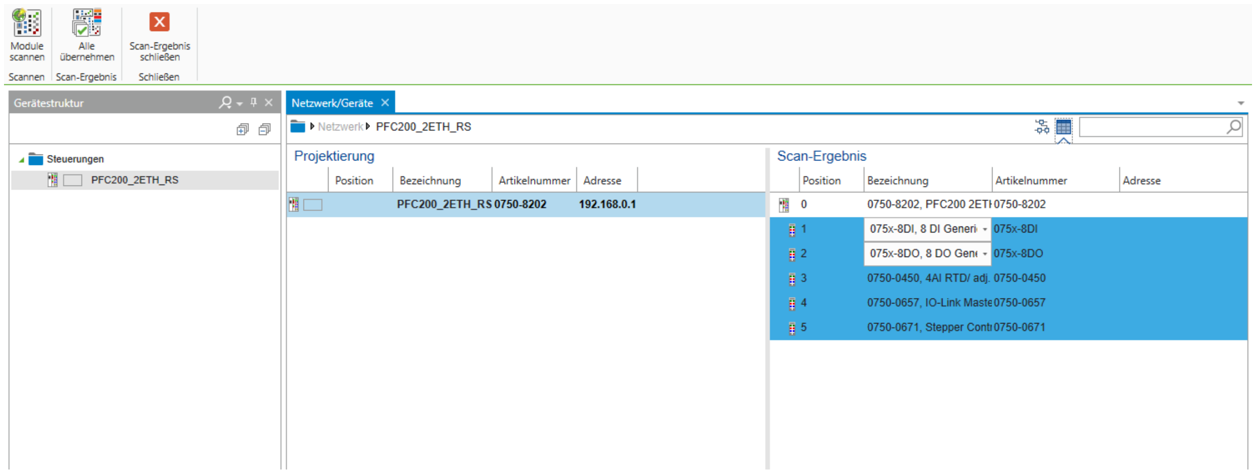
2.3.1 Signal Mapping

The PLC is wired to the conveyor belt's components as follows:

Table 2.1: Connections to the PLC

Component	Input address	Output address
Light beam sensor	%IX1.0	
Inductive sensor	%IX1.1	
Motor on		%QX0.0
Stopper		%QX0.1
Motor direction change		%QX0.2



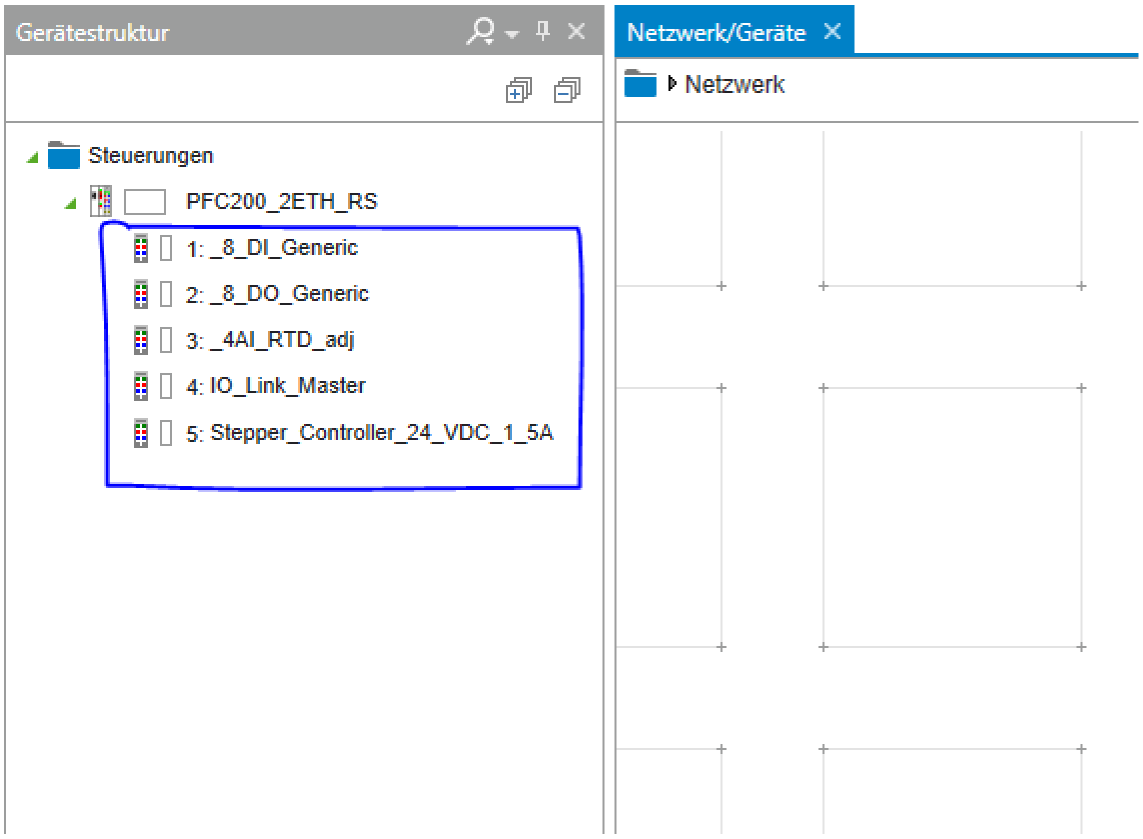


Projektierung

Position	Bezeichnung	Artikelnummer	Adresse
	PFC200_2ETH_RS	0750-8202	192.168.0.1

Scan-Ergebnis

Position	Bezeichnung	Artikelnummer	Adresse
0	0750-8202, PFC200 2ETI	0750-8202	
1	075x-8DI, 8 DI Generi	075x-8DI	
2	075x-8DO, 8 DO Gen	075x-8DO	
3	0750-0450, 4AI RTD/ adj.	0750-0450	
4	0750-0657, IO-Link Maste	0750-0657	
5	0750-0671, Stepper Conti	0750-0671	



Todo: Navigate to `Device Structure` and give these inputs/outputs suitable names to be used in the program.

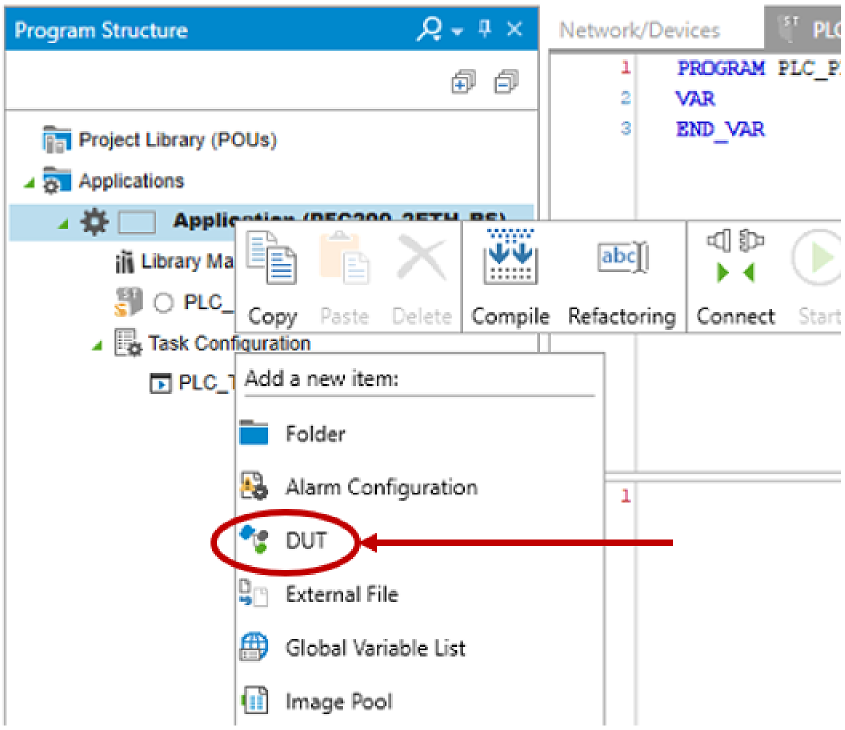
2.3.2 State Machine as Enumeration

State machines can be implemented as enumerations in ST. An enumeration is a user defined type that holds a specific number of elements. In the case of state machines, each enumeration element represents a state.

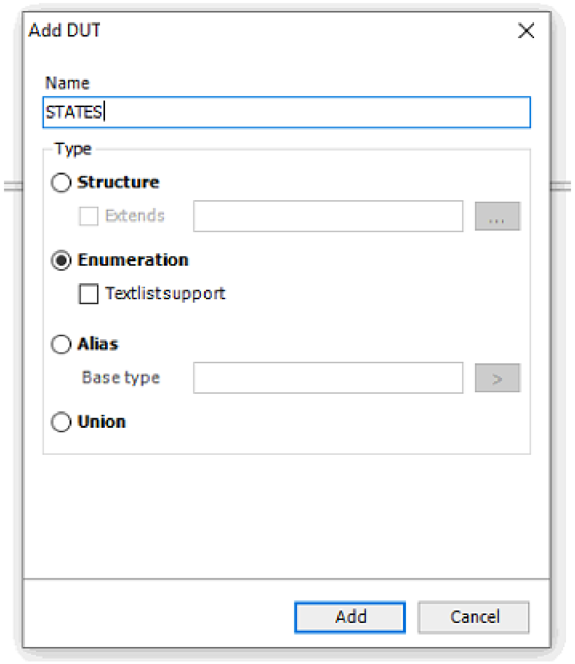
To create an enumeration in e!Cockpit, navigate to `Program Structure` in the bottom left corner, right click on `Applicaition` and select `DUT`.

Todo: Create an enumeration and give it the name `STATES`.

Todo: Add one state element for each state you have in your state machine. Separate each state by a comma and name each state according to your state machine diagram

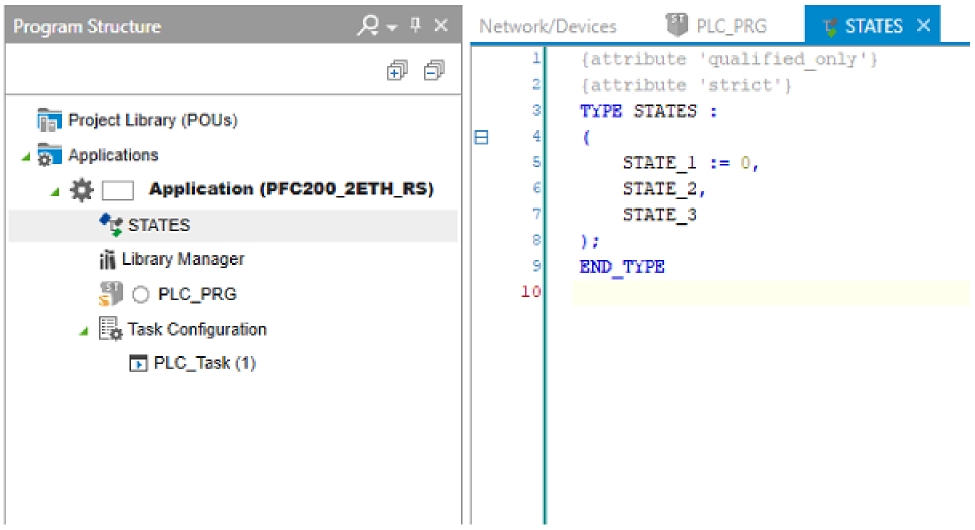


```
PROGRAM PLC_P
VAR
END_VAR
```



Name: STATES

Type:
- ( ) Structure
  - [ ] Extends
- (●) Enumeration
  - [ ] Textlistsupport
- ( ) Alias
  - Base type
- ( ) Union



```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE STATES :
(
    STATE_1 := 0,
    STATE_2,
    STATE_3
);
END_TYPE
```

Fig. 2.2: Example of an enumeration with unspecific state names. Note: you have to give states meaningful names (e.g., MOTOR_FORWARD, MOTOR_BACKWARD, etc.) not call the states STATE_1, STATE_2, etc.

2.3.3 PLC_PRG and the Switch Case Statement

Todo: Create an instance of the STATES enumeration in you program (call it `state`) and an `Entry` boolean variable. Initialize your state variable with the `INIT` state.

Note: Every state machine should include the `INIT` state. This is the first state the program should run when powered. If you didn't include an `INIT` state in your preparation, include one now.

```
PROGRAM PLC_PRG
VAR
        state : STATES := STATES.INIT;
        Entry : BOOL := TRUE;
END_VAR
```

Todo: In the program code window, write a switch-case statement to switch between your states. Use the `Entry` variable to implement the entry code.

```
CASE state OF
        STATES.INIT:
                IF Entry THEN
                        Entry := FALSE;
                        // insert "entry" code here. This code will be carried out only␣
↪once upon entry to the state.

                END_IF
                // insert "do" code here. This code will be carried out every PLC cycle␣
↪while the program is in this state.
```

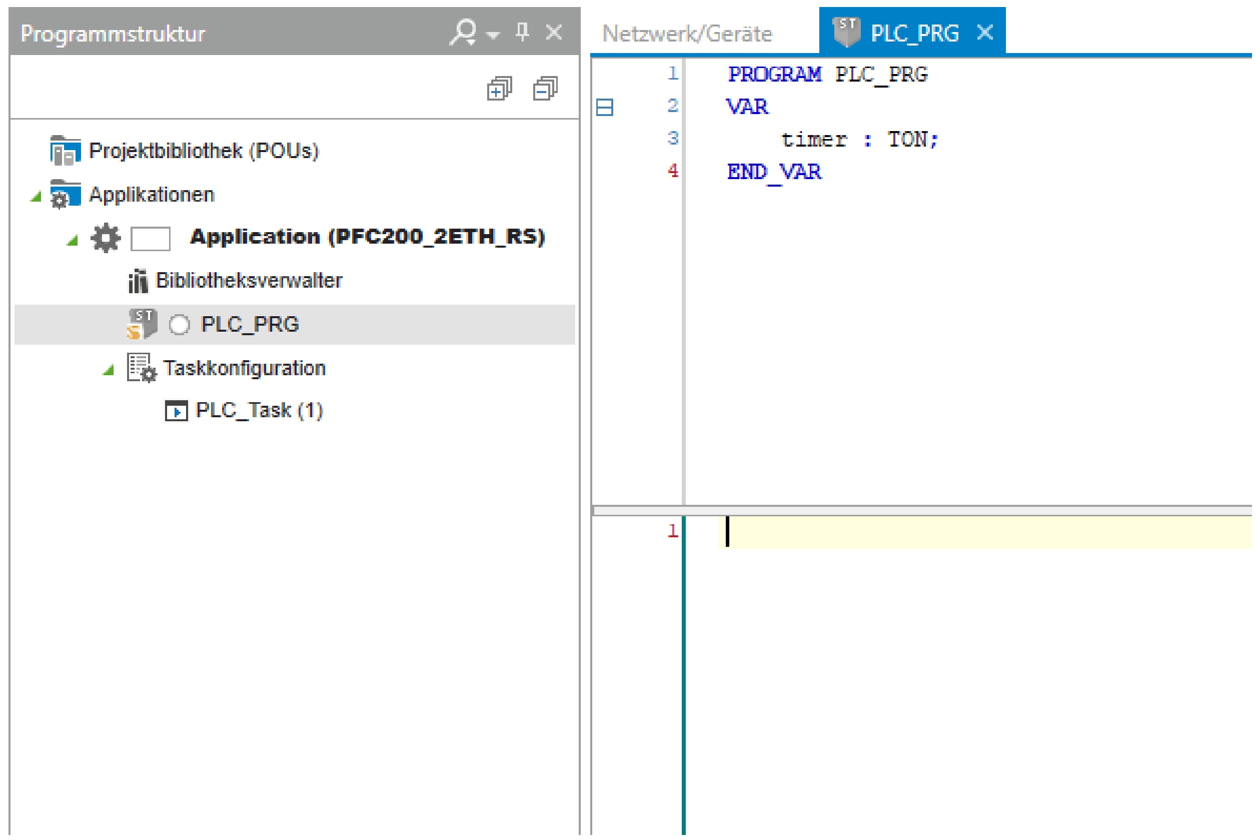
(continues on next page)

(continued from previous page)

```
                //exit
                IF [exit_condition_here] THEN
                        Entry := TRUE;
                        state := STATES.[next_state_here];
                        // insert "exit" code here. This code will run only once upon␣
↪exiting this state.

                END_IF;
        STATES.NEXT_STATE: // repeat the pattern above for every state.
        .
        .
        .
        .
        .
END_CASE
```

Hint: To use a timer, create a timer variable in PLC_PRG and call it anything, for example, timer. This variable is of data type TON. Give it no default value.

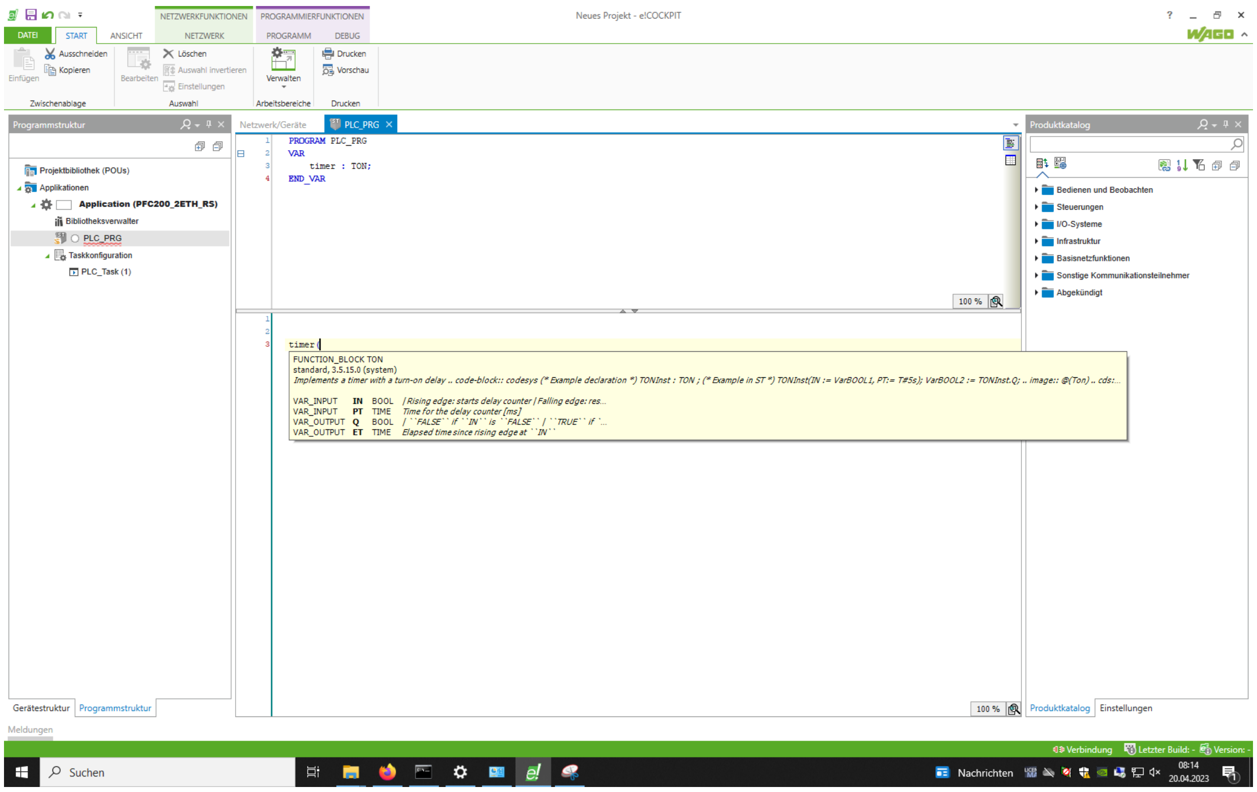
Your timer declaration may look like this:



```
PROGRAM PLC_PRG
VAR
    timer : TON;
END_VAR
```

When attempting to use the timer in the main program, you may see the following TON documentation.

This timer function block (of type TON) accepts 2 inputs, namely IN and PT, and gives us 2 optional outputs, namely



```
PROGRAM PLC_PRG
VAR
    timer : TON;
END_VAR
```

```
timer(
```

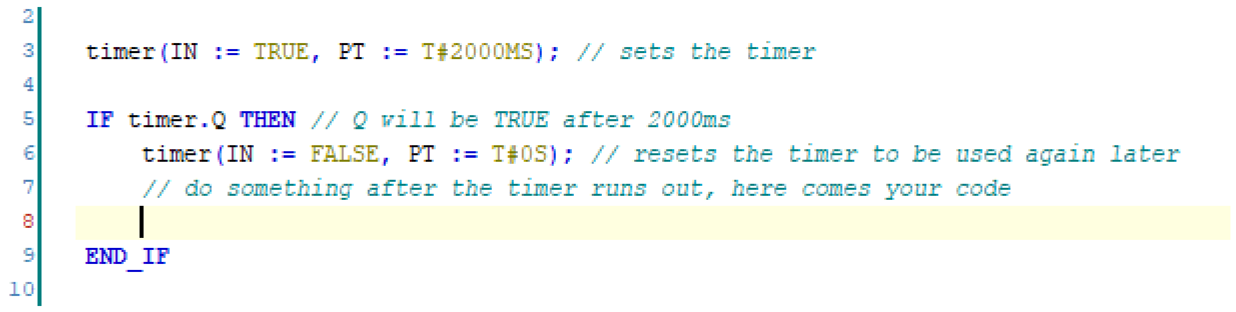
```
FUNCTION_BLOCK TON
standard, 3.5.15.0 (system)
Implements a timer with a turn-on delay .. code-block:: codesys (* Example declaration *) TONInst : TON ; (* Example in ST *) TONInst(IN := VarBOOL1, PT:= T#5s); VarBOOL2 := TONInst.Q; .. image:: @(Ton) .. cds:...

VAR_INPUT   IN  BOOL  | Rising edge: starts delay counter | Falling edge: res...
VAR_INPUT   PT  TIME  Time for the delay counter [ms]
VAR_OUTPUT  Q   BOOL  | ``FALSE`` if ``IN`` is ``FALSE`` | ``TRUE`` if `...
VAR_OUTPUT  ET  TIME  Elapsed time since rising edge at ``IN``
```

Q and ET.

Hint:

- IN sets the timer
- PT gives the time value that the timer should count
- Q is set to TRUE when the timer finishes counting
- ET gives the elapsed time since IN was set to TRUE

To check if the timer has counted untill the end, check if timer.Q is TRUE. Do not forget to reset IN after the counter has finished counting, otherwise the counter wont be usable again.



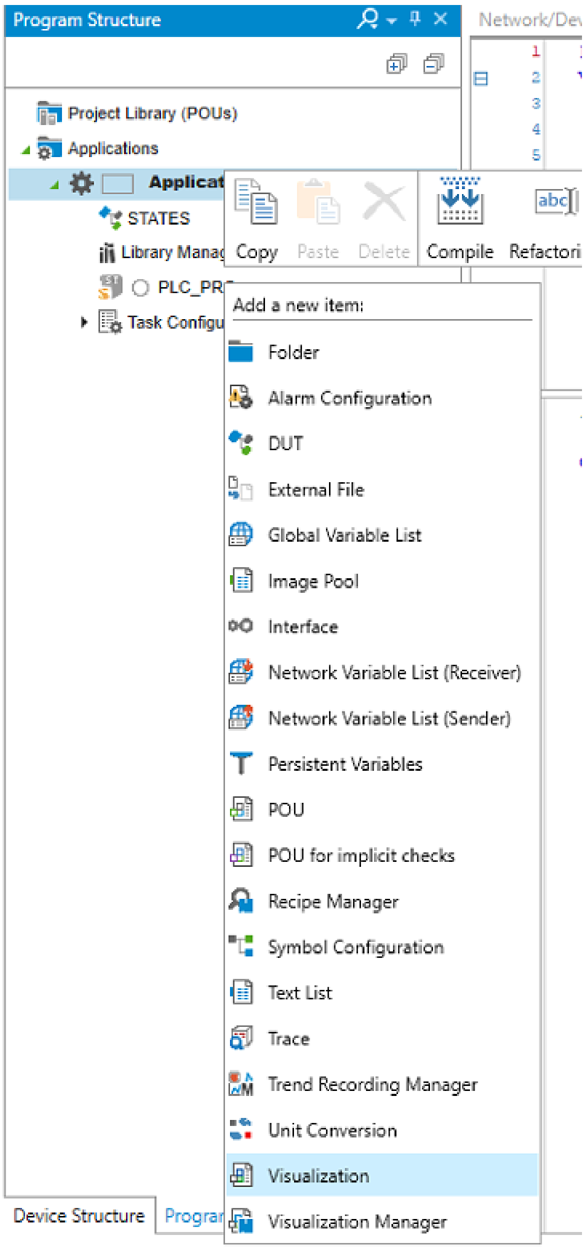
```

timer(IN := TRUE, PT := T#2000MS); // sets the timer

IF timer.Q THEN // Q will be TRUE after 2000ms
    timer(IN := FALSE, PT := T#0S); // resets the timer to be used again later
    // do something after the timer runs out, here comes your code

END_IF

```

Todo: After writing the program, connect the PLC to the station using the 15-pin plug attached to the PLC rack, upload the program to the PLC and test if it works as intented.

2.3.4 Visualization in e!Cockpit

In this section, a visualization of the program will be created. Visualization help show what is going on in the real machine at a glance.

Todo: Right click on `Applicaition` and choose `Visualization` to add a visualization to your program.



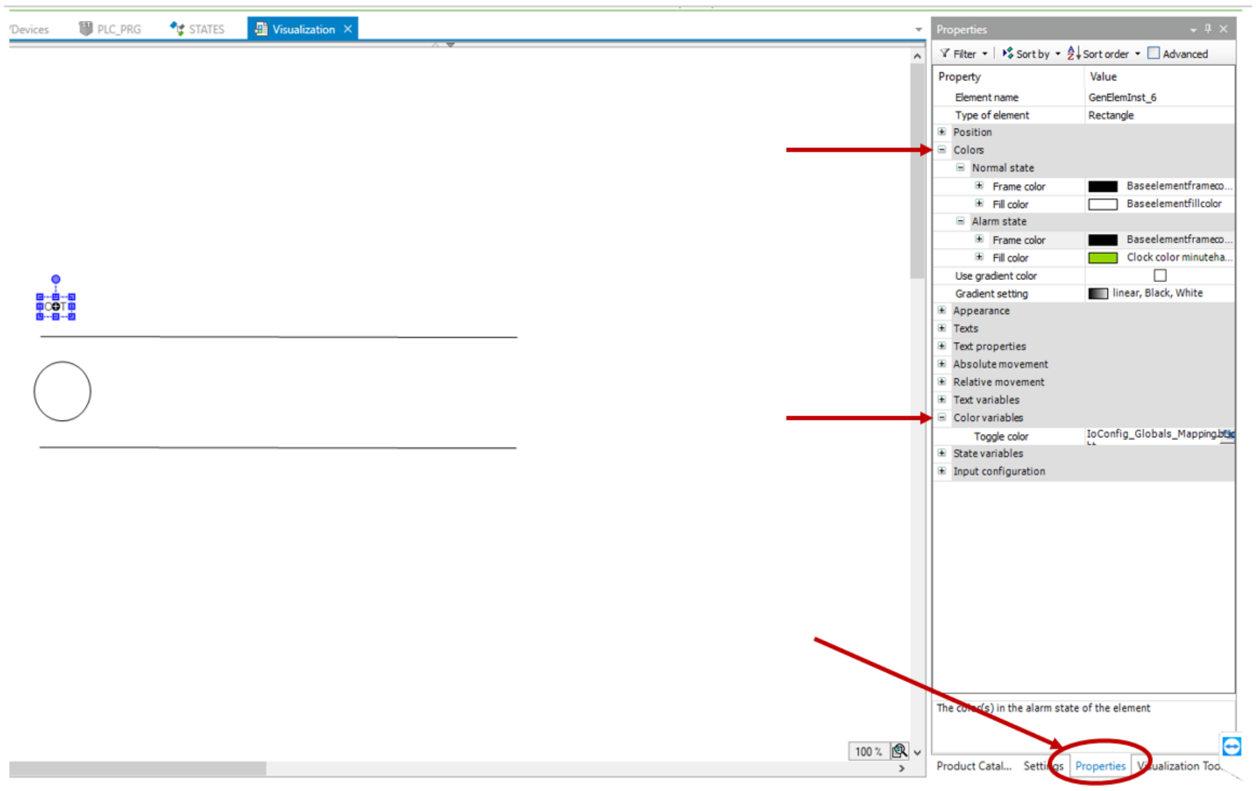
Todo: Check `Active` on both available symbol libraries and click `Add`.

Changing colors

Todo: In the visualization window, drag and drop some basic shapes from the `Visualization Toolbox` on the left. Try to draw a simple conveyor belt shape using the basic shapes. E.g., use lines for the belt and a circle for the workpiece, etc.

Variables can be assigned to shapes from within the `Properties` tab on the bottom left. E.g., variables can toggle a color change of shapes or move the shapes along an axis in the visualization.

Todo: Drag and drop a simple rectangle. Assign the OPT sensor's variable as the color variable. In `Colors`, make the rectangle turn green when in alarm state. This makes the rectangle in the visualization turn green when the sensor is triggered. Upload your new program and check if it works.



Property	Value
Element name	GenElemInst_6
Type of element	Rectangle
⊞ Position	
⊟ Colors	
⊟ Normal state	
⊞ Frame color	■ Baseelementframeco...
⊞ Fill color	□ Baseelementfillcolor
⊟ Alarm state	
⊞ Frame color	■ Baseelementframeco...
⊞ Fill color	🟩 Clock color minuteha...
Use gradient color	☐
Gradient setting	linear, Black, White
⊞ Appearance	
⊞ Texts	
⊞ Text properties	
⊞ Absolute movement	
⊞ Relative movement	
⊞ Text variables	
⊟ Color variables	
Toggle color	IoConfig_Globals_Mapping.b
⊞ State variables	
⊞ Input configuration	

The color(s) in the alarm state of the element

Showing current state

Todo: From the `Common Controls` tab, drag and drop a `Text Field`.

Text fields can show dynamic variables that change during runtime. This is done by using placeholders.

Todo: In your main program, create a new STRING type variable to store the current state's name in. In every state's do code, assign the current state's name to that variable. In the visualization, show that variable by typing `Current`

`State :  %s` into a text field. In that text field's properties tab, link your newly created STRING variable as the `Text variable`. Run the program and see if the current state is showed in the visualization.

Note: The placeholder `%s` is used to hold a string type variable. A list of placeholders can be found here.

Moving an object

An object's position in the visualization can be coupled on an INT type variable from the program. This is done in `Properties` under `Absolute movement`, `Movement`, and either `X` or `Y` for the corresponding axis.

Todo: Create a moving circle (represents the workpiece) that moves when the real band moves. Use a new INT type variable as your `Absolute movement` in X direction variable and a new TON type variable to increment it. Adjust the incrementation size and delay for a better visualization.

Final task

Todo: Put everything together to build a visualization of the conveyor belt. Show the current state as well as whether sensors are triggered or not (over color change). Moving parts should also move in the visualization. A color change is sufficient to show if the stopper solenoid is actuated or not.

Note: You can use parameters to show/hide objects in the visualization.

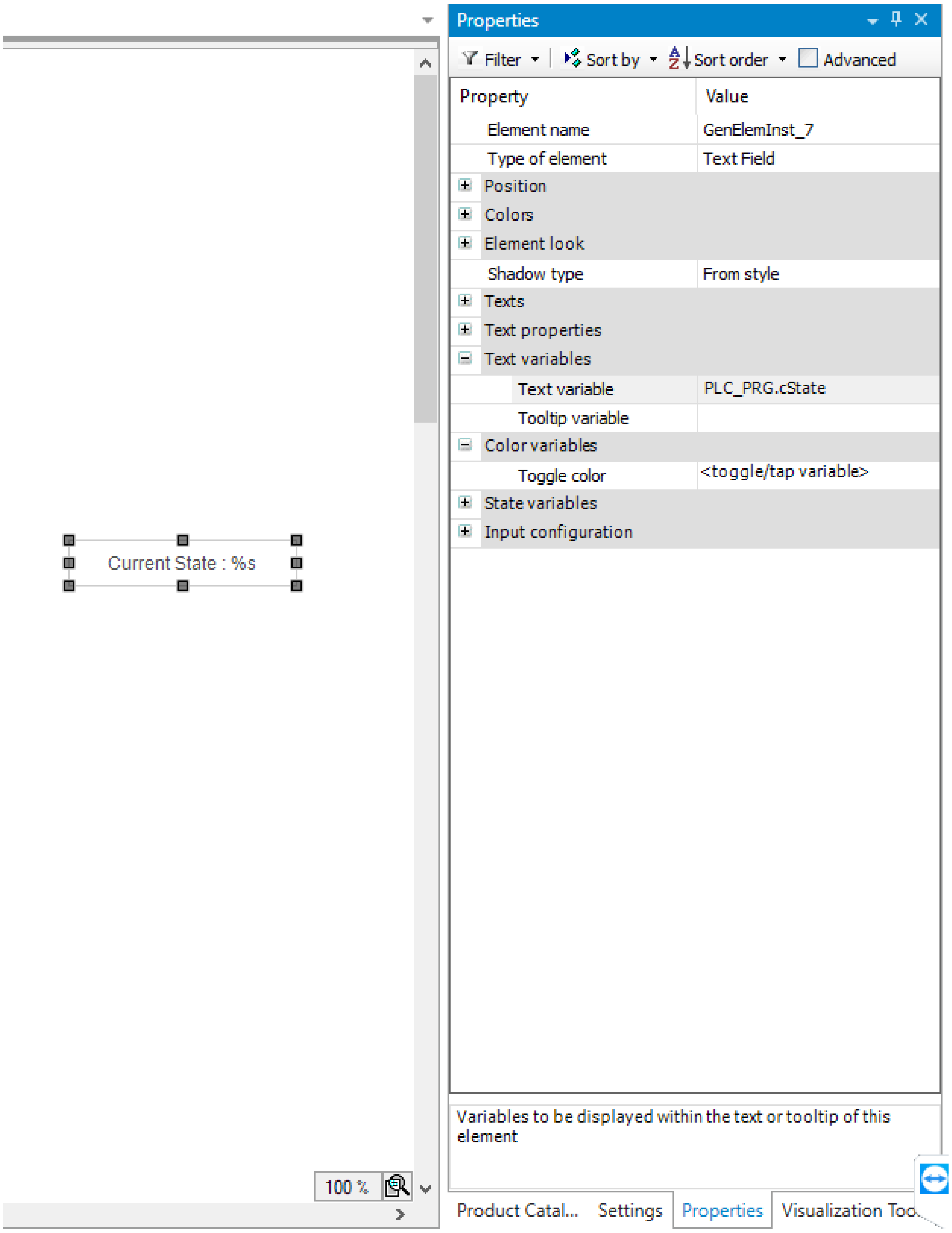
2.4 Preparation: Handling Station

2.4.1 Goal

- To get to know and describe the structure and functionality of the handling station

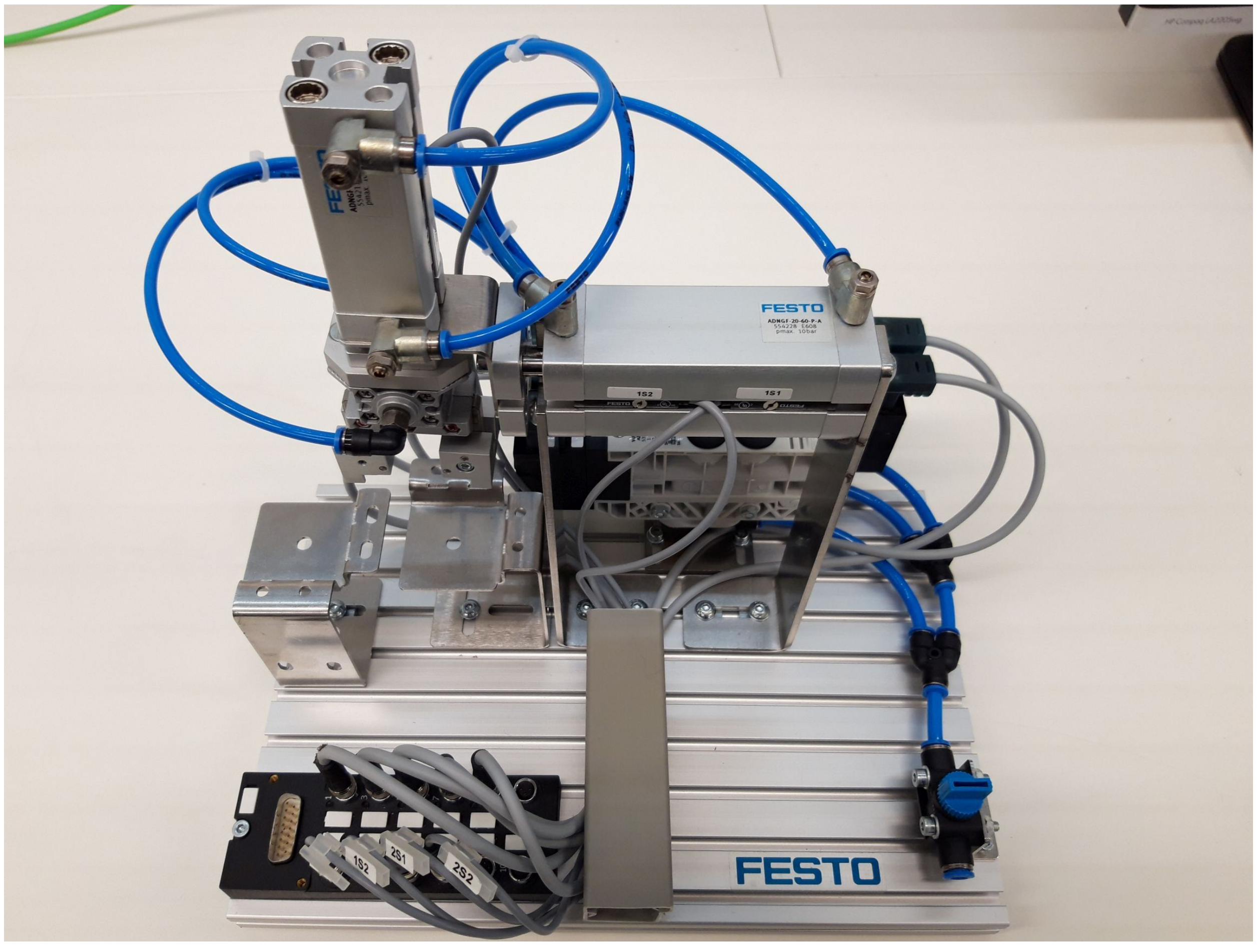
The station should mount a top part (which is on the right side) on top of a bottom part (which is on the left side). The station uses 2 different double acting cylinders and a pneumatic gripper. The functionality of the station can be seen in this video: https://www.youtube.com/watch?v=Jav9SERe0sE

Todo: Create a state machine diagram of the handling station. Specify what should be done in the `entry`, `do`, and `exit` phases of each state. Define the transition conditions between each state and the next state.



Property	Value
Element name	GenElemInst_7
Type of element	Text Field
⊞ Position	
⊞ Colors	
⊞ Element look	
Shadow type	From style
⊞ Texts	
⊞ Text properties	
⊟ Text variables	
Text variable	PLC_PRG.cState
Tooltip variable	
⊟ Color variables	
Toggle color	<toggle/tap variable>
⊞ State variables	
⊞ Input configuration	

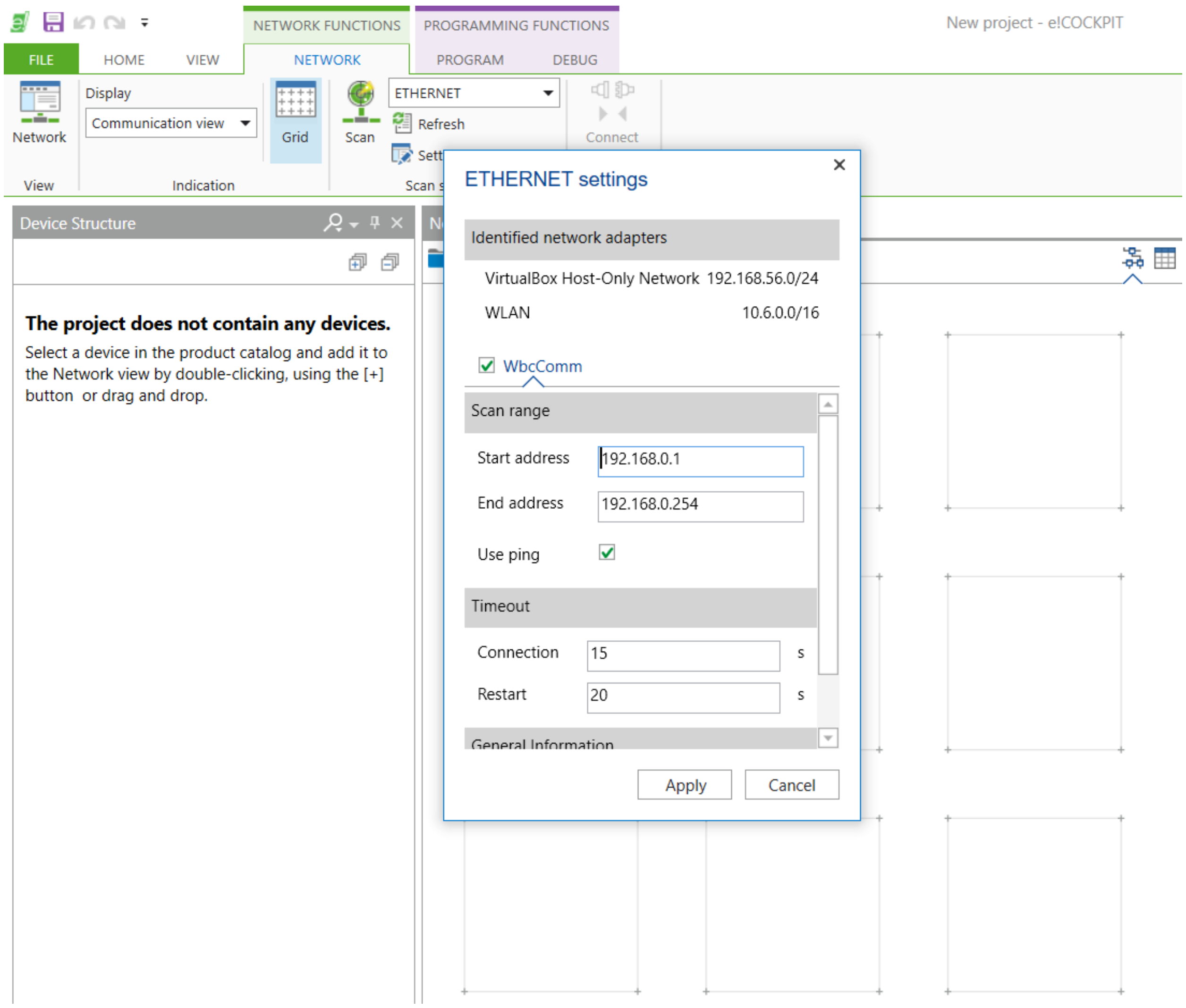
Variables to be displayed within the text or tooltip of this element


FESTO
1S2
1S1
2S1
2S2
FESTO

2.5 Assignment: Handling Station

Todo: Scan for and connect to the PLC from inside e!Cockpit. This is how to do it:

Take a look at the PLC rack and read the PLC's IP address. Then navigate to Network - Settings and put in the right IP range to scan. For example, scanning in the range from 192.168.1.1 till 192.168.1.254 will detect all PLC's in this network that have IP addresses within that range. Please note that this scanning process can take multiple trials to detect the PLC. If a PLC is not detected immediately, scan again.



The project does not contain any devices.

Select a device in the product catalog and add it to the Network view by double-clicking, using the [+] button or drag and drop.

ETHERNET settings

Identified network adapters

VirtualBox Host-Only Network	192.168.56.0/24
WLAN	10.6.0.0/16

✓ WbcComm

Scan range

Start address	192.168.0.1	
End address	192.168.0.254	
Use ping	✓	

Timeout

Connection	15	s
Restart	20	s

General Information

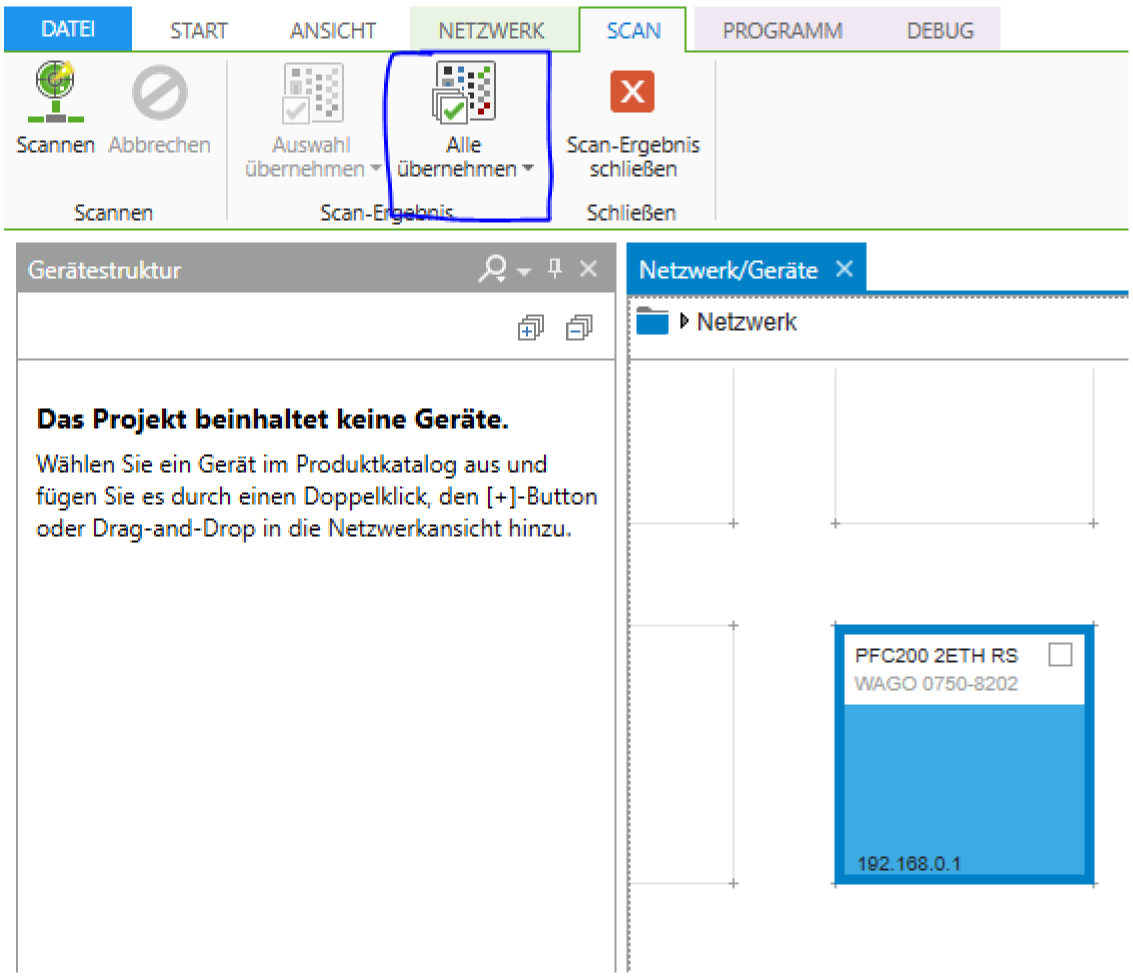
Apply Cancel

After detecting the PLC, click Apply All to add the PLC to your project

Hint: The IP address of the PLC used in this session is 192.168.0.1

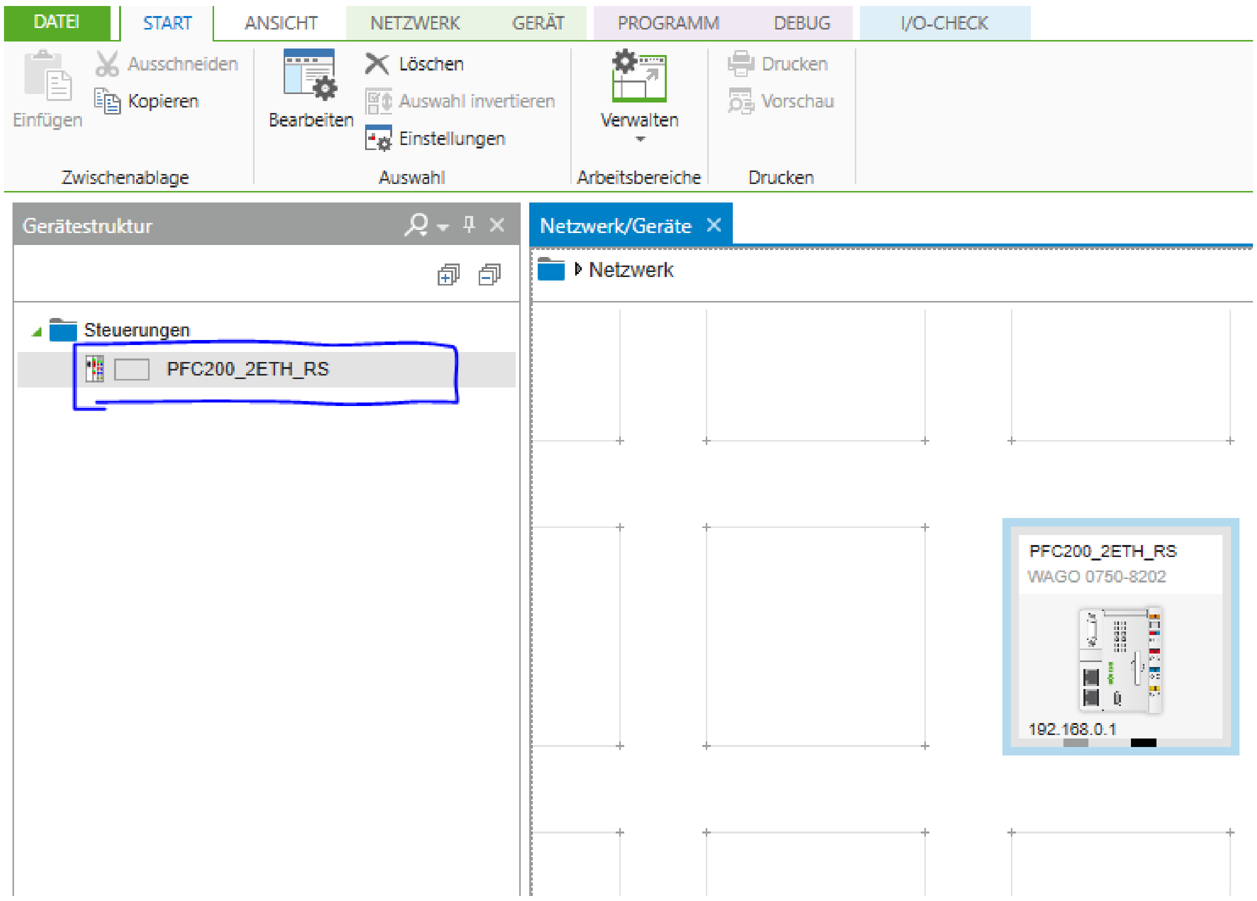
Note that now the PLC has been added to your project.

This has only added the PLC module itself. But the PLC has other modules attached to it that were not yet scanned for. Every module adds new functionality to the PLC. For example, module 750-430 is an 8-channel digital input card. Meaning that the wire-plugs on the 750-430 accept digital sensors like an inductive sensor, light beam sensor, etc.



Das Projekt beinhaltet keine Geräte.

Wählen Sie ein Gerät im Produktkatalog aus und fügen Sie es durch einen Doppelklick, den [+]-Button oder Drag-and-Drop in die Netzwerkansicht hinzu.


DATEI
START
ANSICHT
NETZWERK
GERÄT
PROGRAMM
DEBUG
I/O-CHECK
Einfügen
Ausschneiden
Kopieren
Zwischenablage
Bearbeiten
Löschen
Auswahl invertieren
Einstellungen
Auswahl
Verwalten
Arbeitsbereiche
Drucken
Vorschau
Drucken
Gerätestruktur
Steuerungen
PFC200_2ETH_RS
Netzwerk/Geräte
Netzwerk
PFC200_2ETH_RS
WAGO 0750-8202
192.168.0.1

If you are wondering what a certian module does, search its module number online.

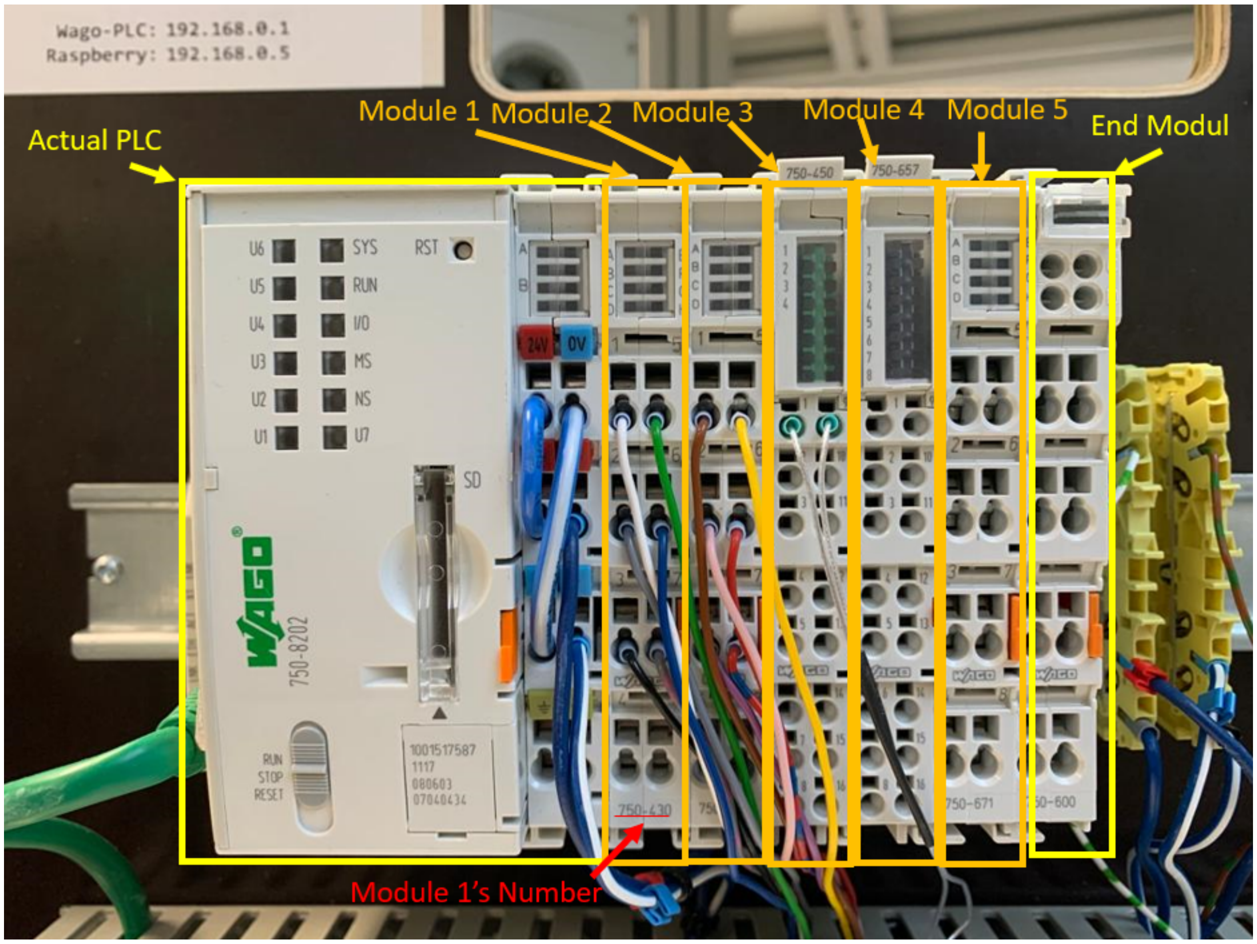
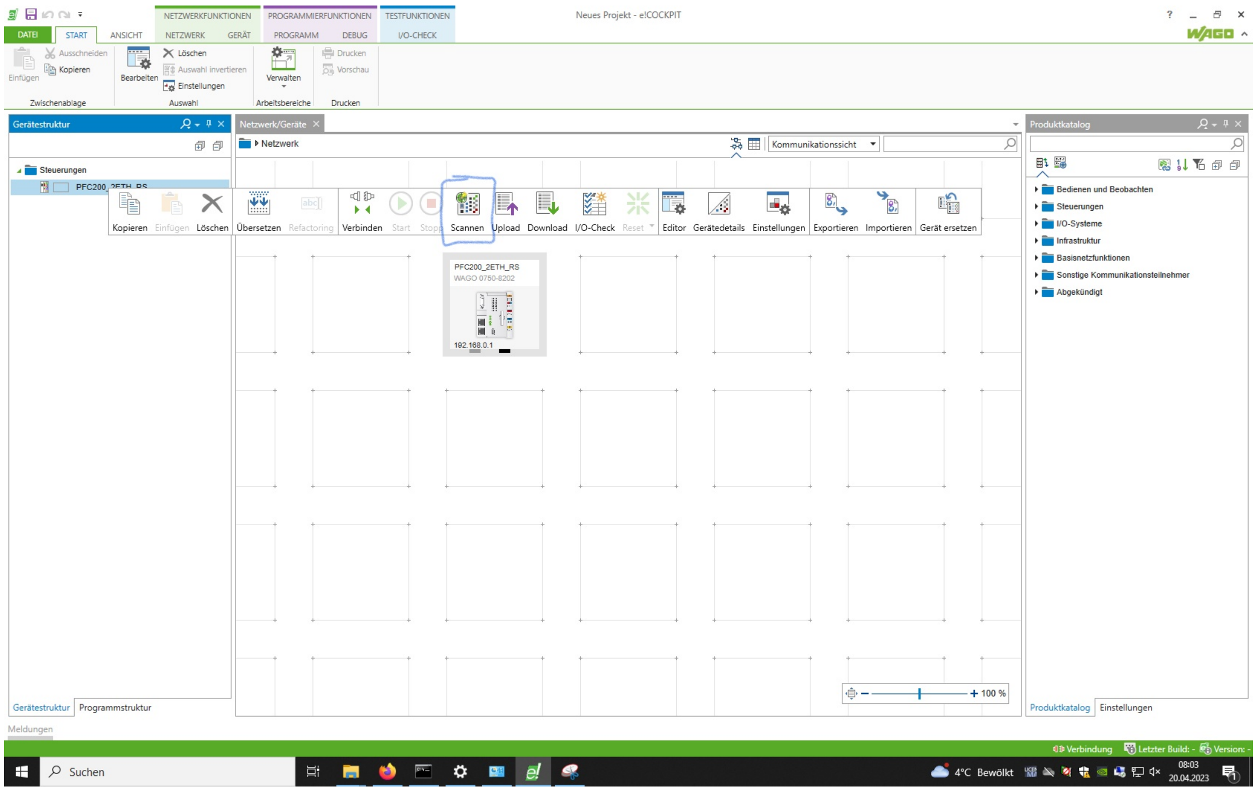


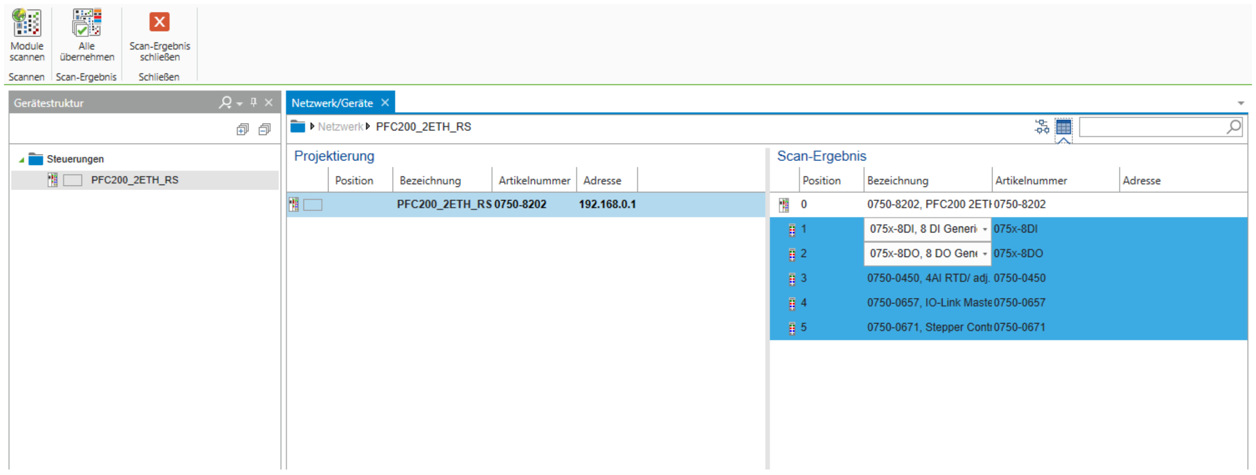
Fig. 2.3: The PLC in this session has 5 modules inserted between the PLC and the End Module. Note that the End Module has to always be there at the end.

Todo: Right click on the newly added PLC and choose scan to scan for the modules inserted next to the PLC.

The scan results should identify 5 modules. Click add all to add all modules to the project

After adding all scanned modules, your device structure should look like this:



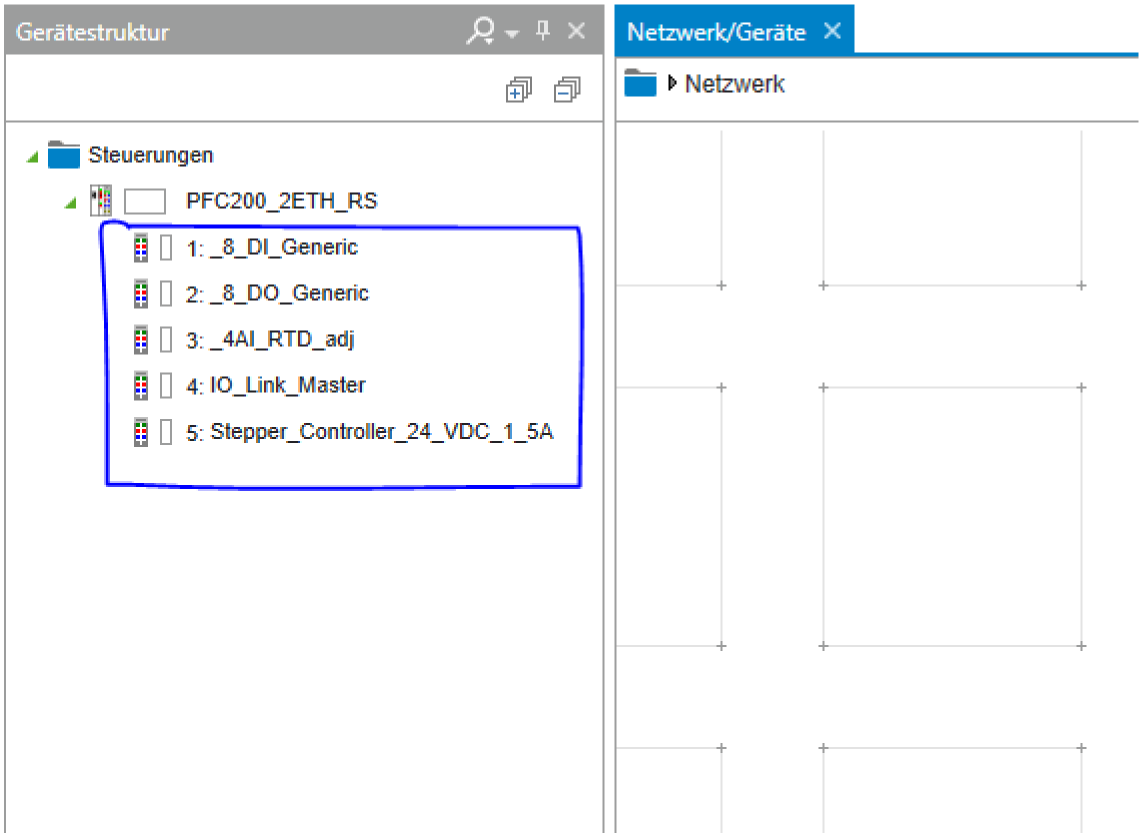


Projektierung

Position	Bezeichnung	Artikelnummer	Adresse
	PFC200_2ETH_RS	0750-8202	192.168.0.1

Scan-Ergebnis

Position	Bezeichnung	Artikelnummer	Adresse
0	0750-8202, PFC200 2ETH	0750-8202	
1	075x-8DI, 8 DI Generi	075x-8DI	
2	075x-8DO, 8 DO Gene	075x-8DO	
3	0750-0450, 4AI RTD/ adj.	0750-0450	
4	0750-0657, IO-Link Maste	0750-0657	
5	0750-0671, Stepper Contı	0750-0671	



2.5.1 Signal Mapping

The PLC is wired to the conveyor belt's components as follows:

Table 2.2: Connections to the PLC

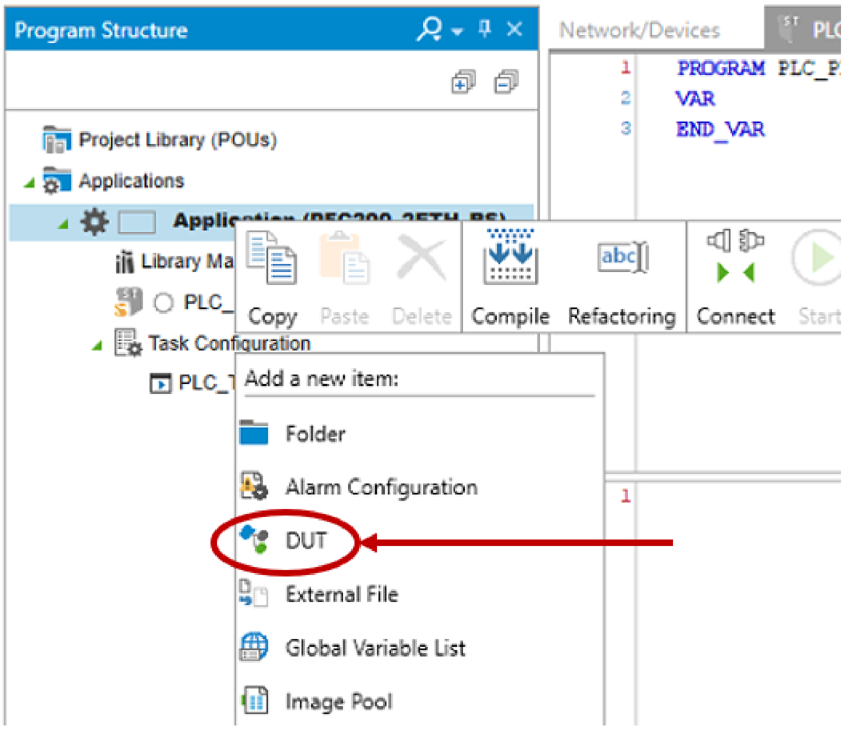
Component	Input address	Output address
1S1	%IX1.0	
1S2	%IX1.1	
2S1	%IX1.2	
2S2	%IX1.3	
1M1		%QX0.0
1M2		%QX0.1
2M1		%QX0.2
2M2		%QX0.3
3M1		%QX0.4

Todo: Navigate to `Device Structure` and give these inputs/outputs suitable names to be used in the program.

2.5.2 State Machine as Enumeration

State machines can be implemented as enumerations in ST. An enumeration is a user defined type that holds a specific number of elements. In the case of state machines, each enumeration element represents a state.

To create an enumeration in e!Cockpit, navigate to `Program Structure` in the bottom left corner, right click on `Applicaition` and select `DUT`.



```
PROGRAM PLC_P
VAR
END_VAR
```

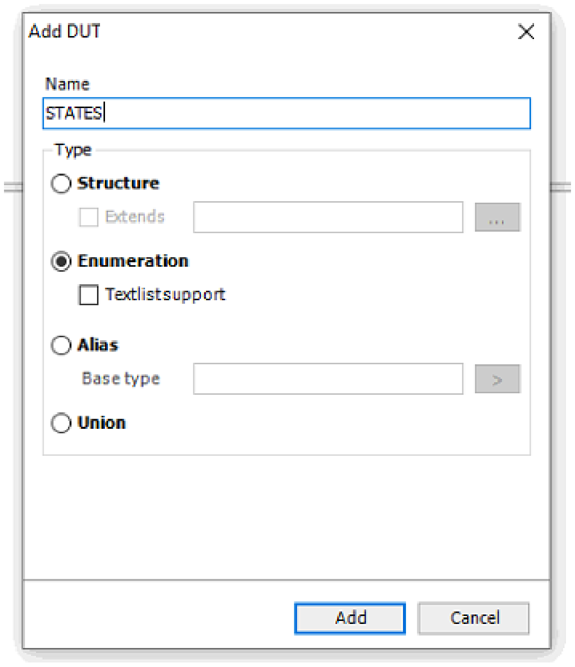
Todo: Create an enumeration and give it the name `STATES`.

Todo: Add one state element for each state you have in your state machine. Separate each state by a comma and name each state according to your state machine diagram

2.5.3 PLC_PRG and the Switch Case Statement

Todo: Create an instance of the STATES enumeration in you program (call it `state`) and an `Entry` boolean variable. Initialize your state variable with the `INIT` state.

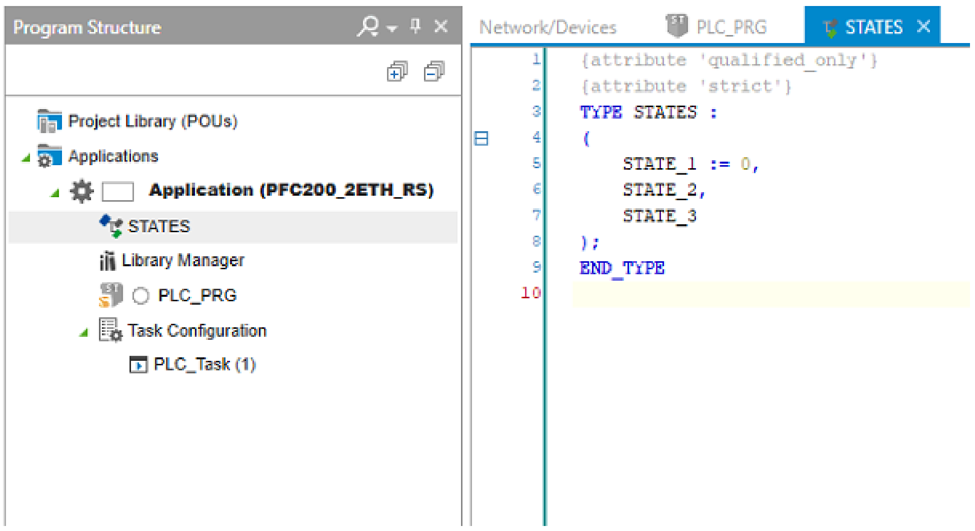
Note: Every state machine should include the `INIT` state. This is the first state the program should run when powered. If you didn't include an `INIT` state in your preparation, include one now.



Name: STATES

Type: Structure [✓ Enumeration] Alias Union

Textlistsupport: unchecked



```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE STATES :
(
    STATE_1 := 0,
    STATE_2,
    STATE_3
);
END_TYPE
```

Fig. 2.4: Example of an enumeration with unspecific state names. Note: you have to give states meaningful names (e.g., MOTOR_FORWARD, MOTOR_BACKWARD, etc.) not call the states STATE_1, STATE_2, etc.

```
PROGRAM PLC_PRG
VAR
        state : STATES := STATES.INIT;
        Entry : BOOL := TRUE;
END_VAR
```

Todo: In the program code window, write a switch-case statement to switch between your states. Use the `Entry` variable to implement the entry code.

```
CASE state OF
        STATES.INIT:
                IF Entry THEN
                        Entry := FALSE;
                        // insert "entry" code here. This code will be carried out only␣
↪once upon entry to the state.

                END_IF
                // insert "do" code here. This code will be carried out every PLC cycle␣
↪while the program is in this state.

                //exit
                IF [exit_condition_here] THEN
                        Entry := TRUE;
                        state := STATES.[next_state_here];
                        // insert "exit" code here. This code will run only once upon␣
↪exiting this state.

                END_IF;
        STATES.NEXT_STATE: // repeat the pattern above for every state.
        .
        .
        .
        .
        .
END_CASE
```

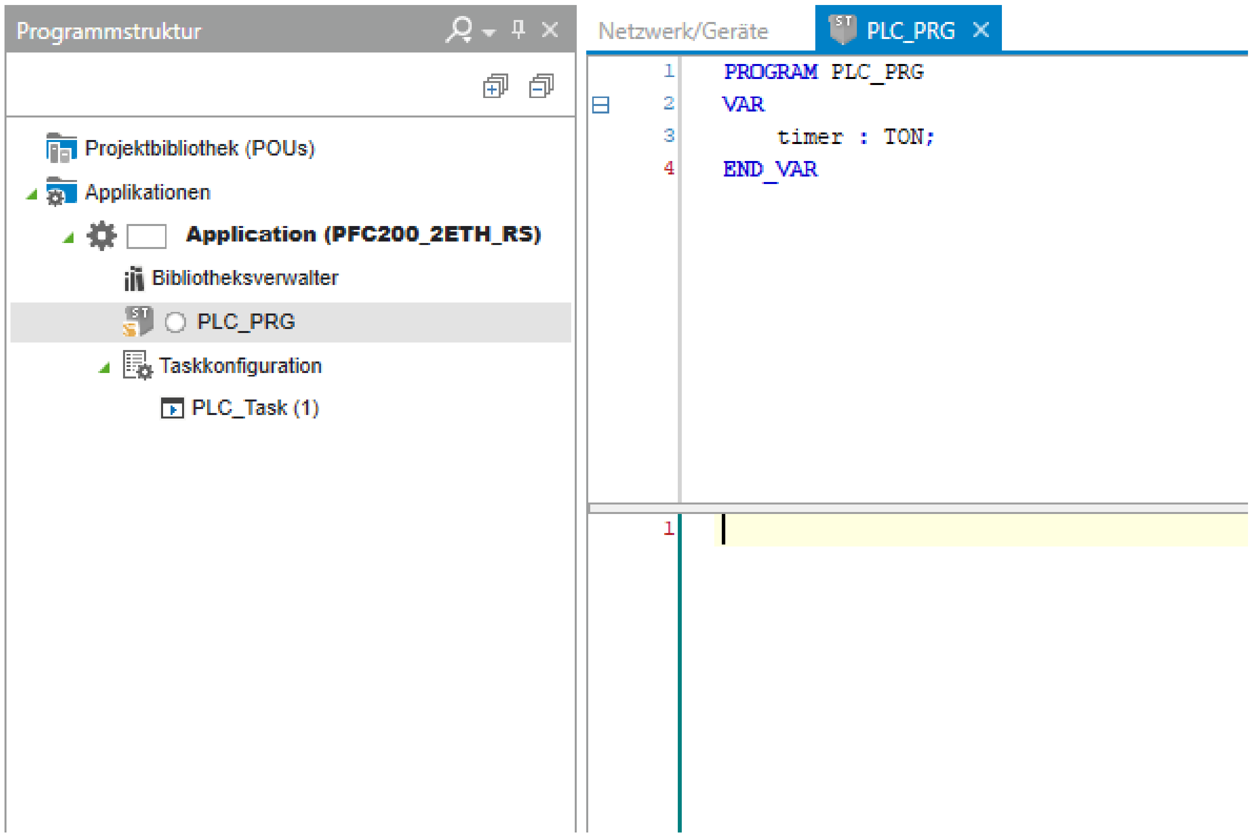
Danger: With the double acting cylinders, do not set both outlets as `TRUE` at the same time.

Whenever you set an outlet as `TRUE`, be sure to follow it with the reverse on the opposite outlet.

```
// retracts cylinder 1
b1M1 := FALSE;
b1M2 := NOT b1M1; // this makes sure that not both outlets are actuated

// extends cylinder 2
b2M1 := TRUE;
b2M2 := NOT b2M1;
```

Hint: To use a timer, create a timer variable in PLC_PRG and call it anything, for example, timer. This variable is of data type TON. Give it no default value.

Your timer declaration may look like this:



```
PROGRAM PLC_PRG
VAR
    timer : TON;
END_VAR
```

When attempting to use the timer in the main program, you may see the following TON documentation.

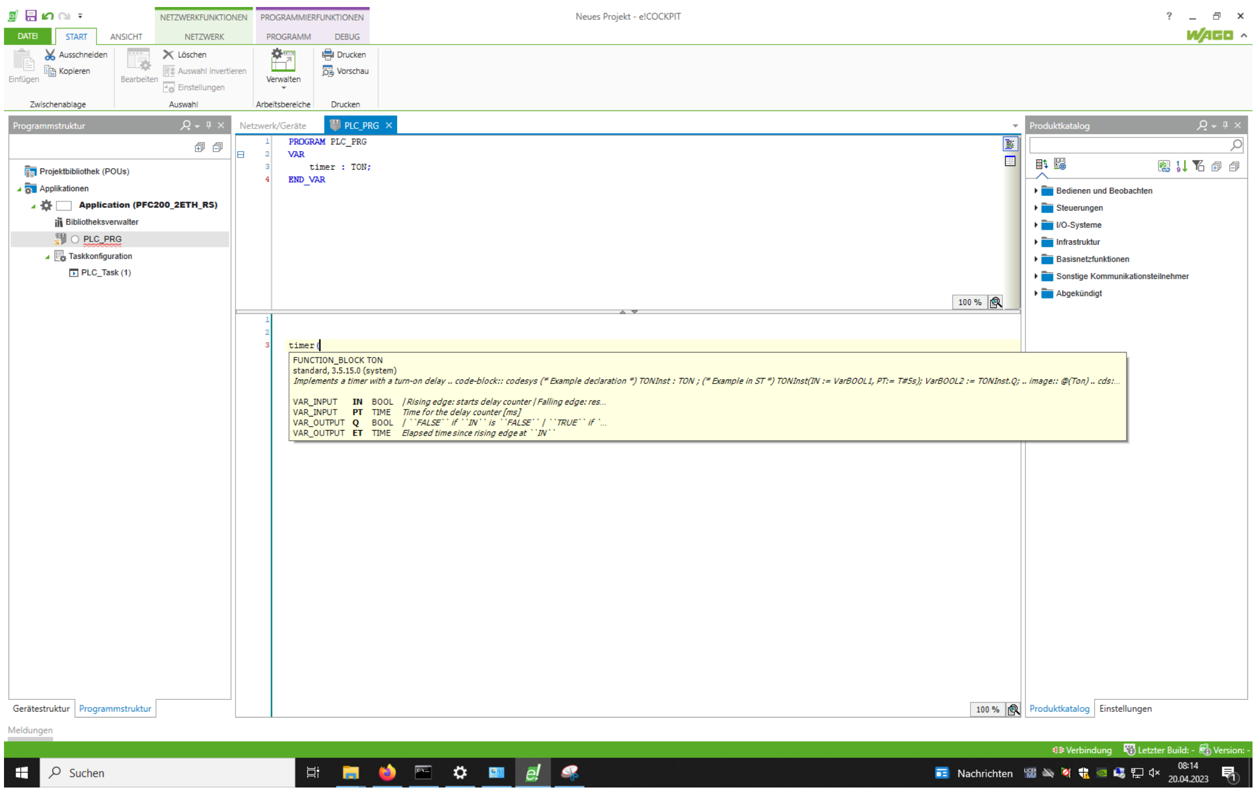
This timer function block (of type TON) accepts 2 inputs, namely IN and PT, and gives us 2 optional outputs, namely Q and ET.

Hint:

- IN sets the timer
- PT gives the time value that the timer should count
- Q is set to TRUE when the timer finishes counting
- ET gives the elapsed time since IN was set to TRUE

To check if the timer has counted untill the end, check if timer.Q is TRUE. Do not forget to reset IN after the counter has finished counting, otherwise the counter wont be usable again.

Todo: After writing the program, connect the PLC to the station using the 15-pin plug attached to the PLC rack, upload the program to the PLC and test if it works as intented.

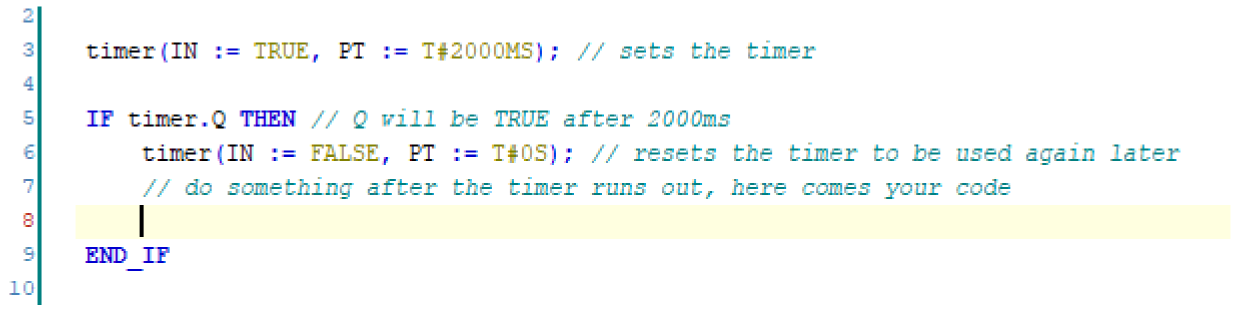


```
PROGRAM PLC_PRG
VAR
    timer : TON;
END_VAR
```

```
timer(
```

FUNCTION_BLOCK TON
standard, 3.5.15.0 (system)
Implements a timer with a turn-on delay .. code-block:: codesys (Example declaration *) TONInst : TON ; (* Example in ST *) TONInst(IN := VarBOOL1, PT:= T#5s); VarBOOL2 := TONInst.Q; .. image:: @(Ton) .. cds:...*

```
VAR_INPUT   IN  BOOL  | Rising edge: starts delay counter | Falling edge: res...
VAR_INPUT   PT  TIME  Time for the delay counter [ms]
VAR_OUTPUT  Q   BOOL  | ``FALSE`` if ``IN`` is ``FALSE`` | ``TRUE`` if `...
VAR_OUTPUT  ET  TIME  Elapsed time since rising edge at ``IN``
```

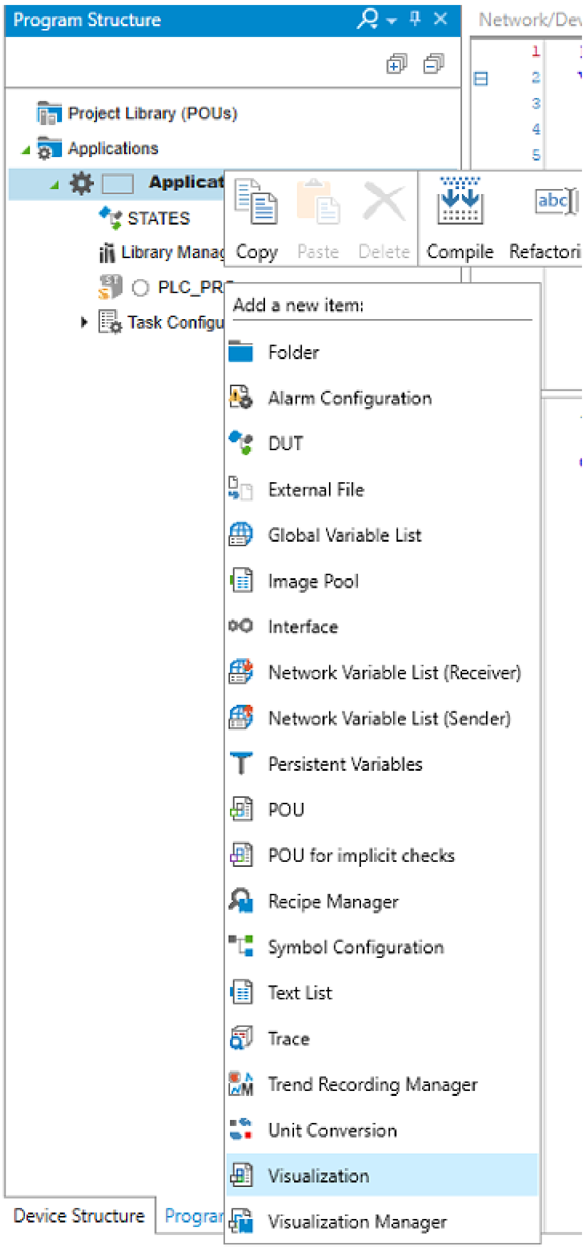


```

timer(IN := TRUE, PT := T#2000MS); // sets the timer

IF timer.Q THEN // Q will be TRUE after 2000ms
    timer(IN := FALSE, PT := T#0S); // resets the timer to be used again later
    // do something after the timer runs out, here comes your code
    
END_IF

```

2.5.4 Visualization in e!Cockpit

In this section, a visualization of the program will be created. Visualization help show what is going on in the real machine at a glance.

Todo: Right click on `Applicaition` and choose `Visualization` to add a visualization to your program.



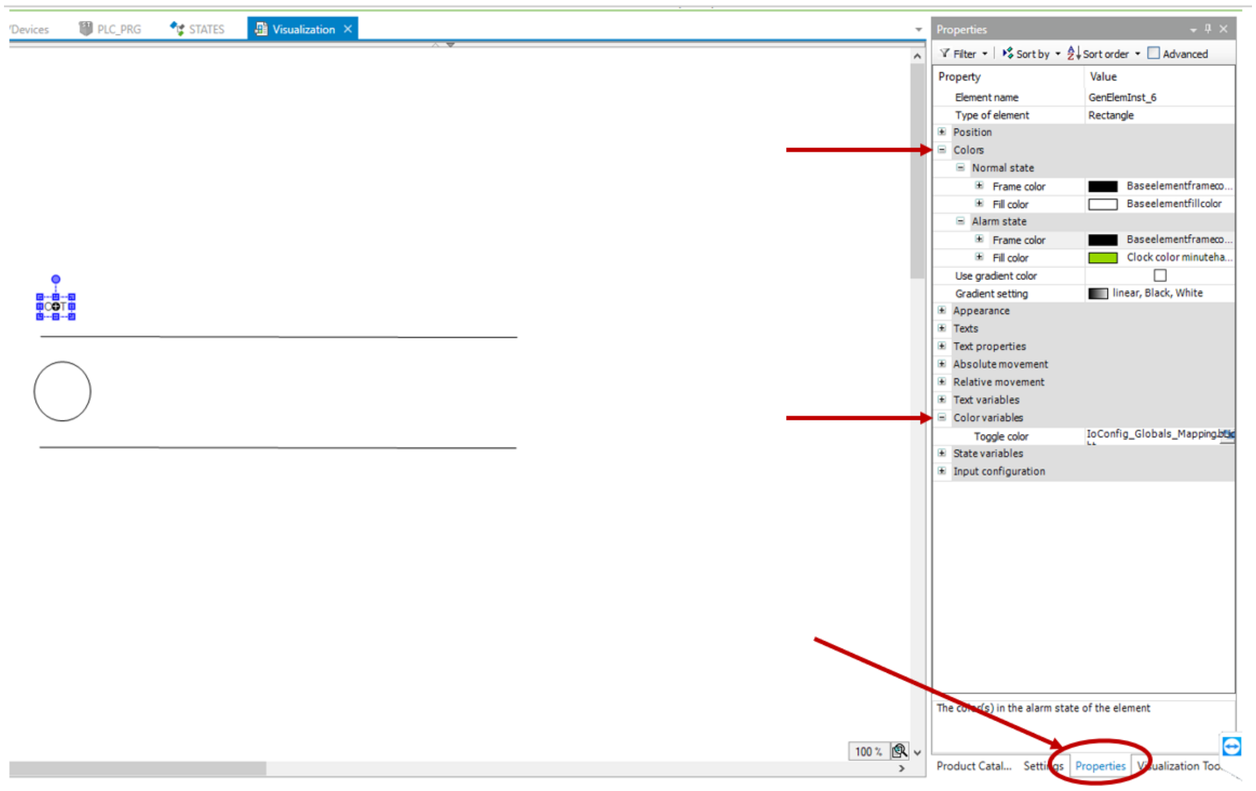
Todo: Check `Active` on both available symbol libraries and click `Add`.

Changing colors

Todo: In the visualization window, drag and drop some basic shapes from the `Visualization Toolbox` on the left. Try to draw a simple conveyor belt shape using the basic shapes. E.g., use lines for the belt and a circle for the workpiece, etc.

Variables can be assigned to shapes from within the `Properties` tab on the bottom left. E.g., variables can toggle a color change of shapes or move the shapes along an axis in the visualization.

Todo: Drag and drop a simple rectangle. Assign the OPT sensor's variable as the color variable. In `Colors`, make the rectangle turn green when in alarm state. This makes the rectangle in the visualization turn green when the sensor is triggered. Upload your new program and check if it works.



Property	Value
Element name	GenElemInst_6
Type of element	Rectangle
⊞ Position	
⊟ Colors	
⊟ Normal state	
⊞ Frame color	⬛ Baseelementframeco...
⊞ Fill color	⬜ Baseelementfillcolor
⊟ Alarm state	
⊞ Frame color	⬛ Baseelementframeco...
⊞ Fill color	🟩 Clock color minuteha...
Use gradient color	☐
Gradient setting	linear, Black, White
⊞ Appearance	
⊞ Texts	
⊞ Text properties	
⊞ Absolute movement	
⊞ Relative movement	
⊞ Text variables	
⊟ Color variables	
Toggle color	IoConfig_Globals_Mapping.b[illegible]
⊞ State variables	
⊞ Input configuration	

The color(s) in the alarm state of the element

Showing current state

Todo: From the `Common Controls` tab, drag and drop a `Text Field`.

Text fields can show dynamic variables that change during runtime. This is done by using placeholders.

Todo: In your main program, create a new STRING type variable to store the current state's name in. In every state's do code, assign the current state's name to that variable. In the visualization, show that variable by typing `Current`

`State :  %s` into a text field. In that text field's properties tab, link your newly created STRING variable as the `Text variable`. Run the program and see if the current state is showed in the visualization.

Note: The placeholder `%s` is used to hold a string type variable. A list of placeholders can be found here.

Moving an object

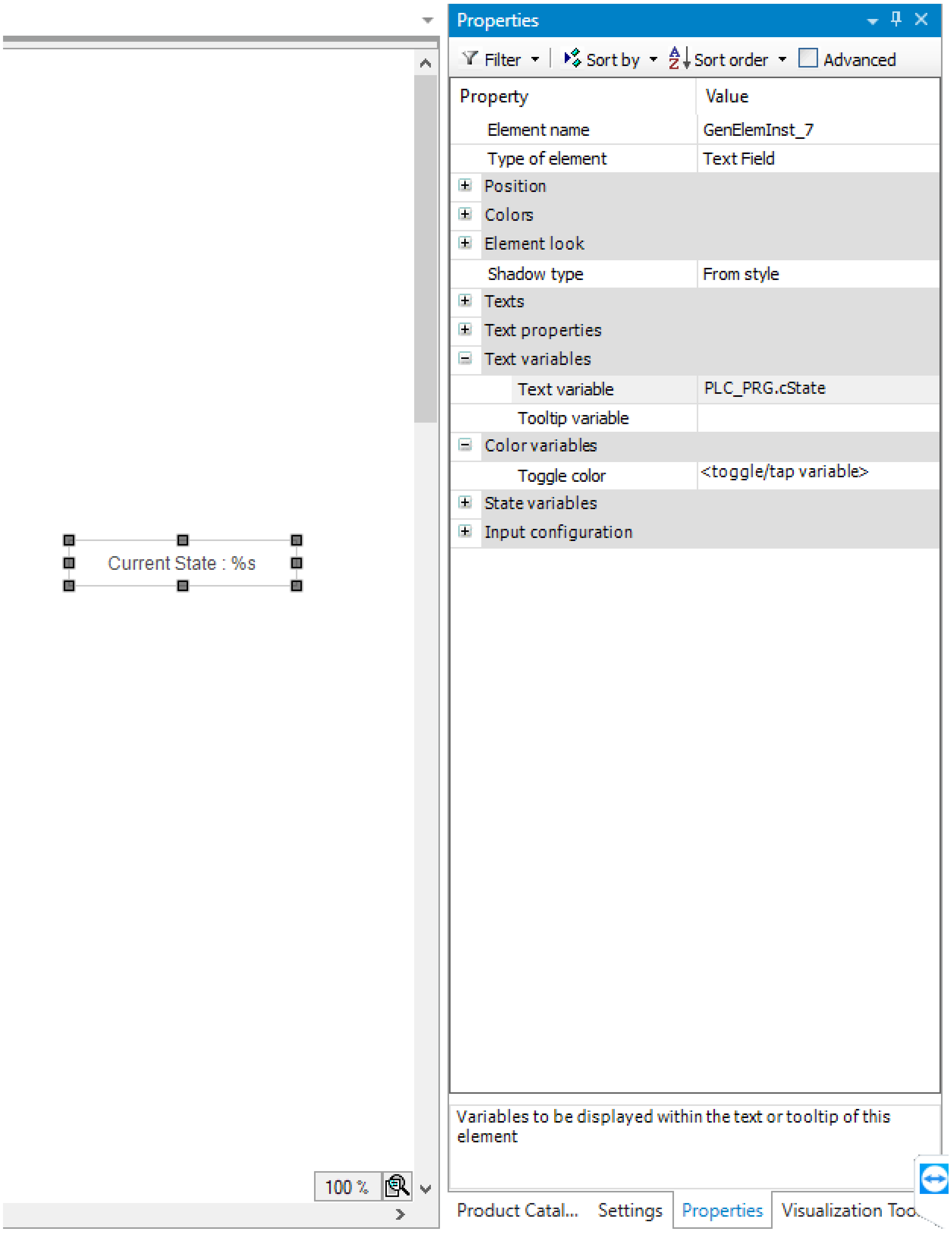
An object's position in the visualization can be coupled on an INT type variable from the program. This is done in `Properties` under `Absolute movement`, `Movement`, and either `X` or `Y` for the corresponding axis.

Todo: Create a moving rectanle (represents the pneumatic cylinder) that moves when the real cylinder moves. Use a new INT type variable as your `Absolute movement` in X direction variable and a new TON type variable to increment it. Adjust the incrementation size and delay for a better visualization.

Final task

Todo: Put everything together to build a visualization of the handling station. Show the current state as well as whether sensors are triggered or not (over color change). Moving parts should also move in the visualization. Use a button in the visualization to start the entire process.

Note: You can use parameters to show/hide objects in the visualization.



Property	Value
Element name	GenElemInst_7
Type of element	Text Field
⊞ Position	
⊞ Colors	
⊞ Element look	
Shadow type	From style
⊞ Texts	
⊞ Text properties	
⊟ Text variables	
Text variable	PLC_PRG.cState
Tooltip variable	
⊟ Color variables	
Toggle color	<toggle/tap variable>
⊞ State variables	
⊞ Input configuration	

Variables to be displayed within the text or tooltip of this element

CHAPTER

THREE

SIEMENS PLC CONTROL OF CONVEYOR BAND

In this session, a Siemens PLC will be used along with a retroreflective sensor to determine the shape of a workpiece.

Note: Please prepare all the `preparation` sections before the actual lab session. This is obligatory for attending and will save you time during the session. Preparation may include drawing diagrams, answering questions, or reading through the basics.

3.1 Learning Outcome

3.1.1 Introduction

In this session you will program a Siemens PLC.

- You will get familiar with Siemens PLCs
- You will get familiar with the TIA Portal environment

3.1.2 Requirements

- *PLC-Basics*
- *TIA Portal Basics*

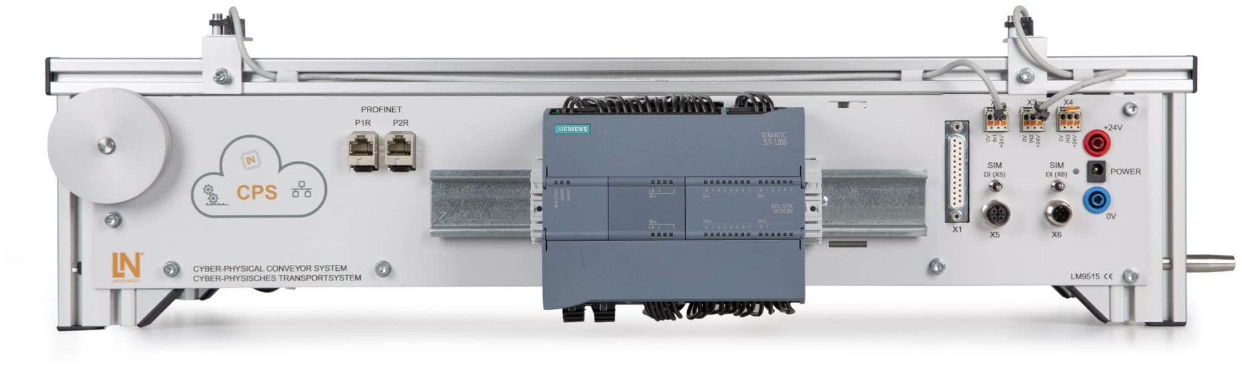
3.1.3 What you need

Software

All software on this list are pre-installed on the lab PC.

- TIA Portal V15 or newer

3.2 Conveyor System



The conveyor system consists of the following:

- Siemens S7 1200 PLC with a signal board installed
- DC motor (labeled `M1`)
- Two magnetic field sensors (labeled `B1` and `B2`)
- A PCB board that (re) wires the I/O of the PLC
- A 20-pin connector to connect the PLC IO to additional sensors and actuators.
- Depending on the revision, the board provides four or five sensor inputs (`X2`, `X3`, `X4`, `X5` and an optional `X6`)
- In addition there are one or two simulation switches (`SIM DI`) for the sensors `X5` and `X6`. The simulation switches simulate a sensor input, even though no sensor is connected to the sensor connector on the conveyor belt.
- The PLC has an analog input module, that is wired in a way to measure the PLC's power consumption and input voltage.
- A `Quick-Chart (in german)`, showing the conveyor's IO is available.

Todo: Make yourself familiar with the conveyor belt and identify each of the inputs

3.3 Preparation

3.3.1 Task Description

The conveyor band shown below is to be controlled by a Siemens PLC. Inductive sensors (1) are mounted on both sides of the conveyor to detect if a workpiece carrier is there. A retroreflective sensor (2) is used to count how many pillars the workpiece has. Different workpieces will be provided with different number of pillars.

The assignment is as follows: first, the conveyor should be waiting in an idle state. If the sensor at the beginning of the conveyor (i.e., on the left side) detects a workpiece, the conveyor's motor should be turned on and the workpiece should be transported to the right side. While the workpiece passes the retroreflective sensor in the middle, the number of pillars on the piece should be counted and saved in a variable of type integer.

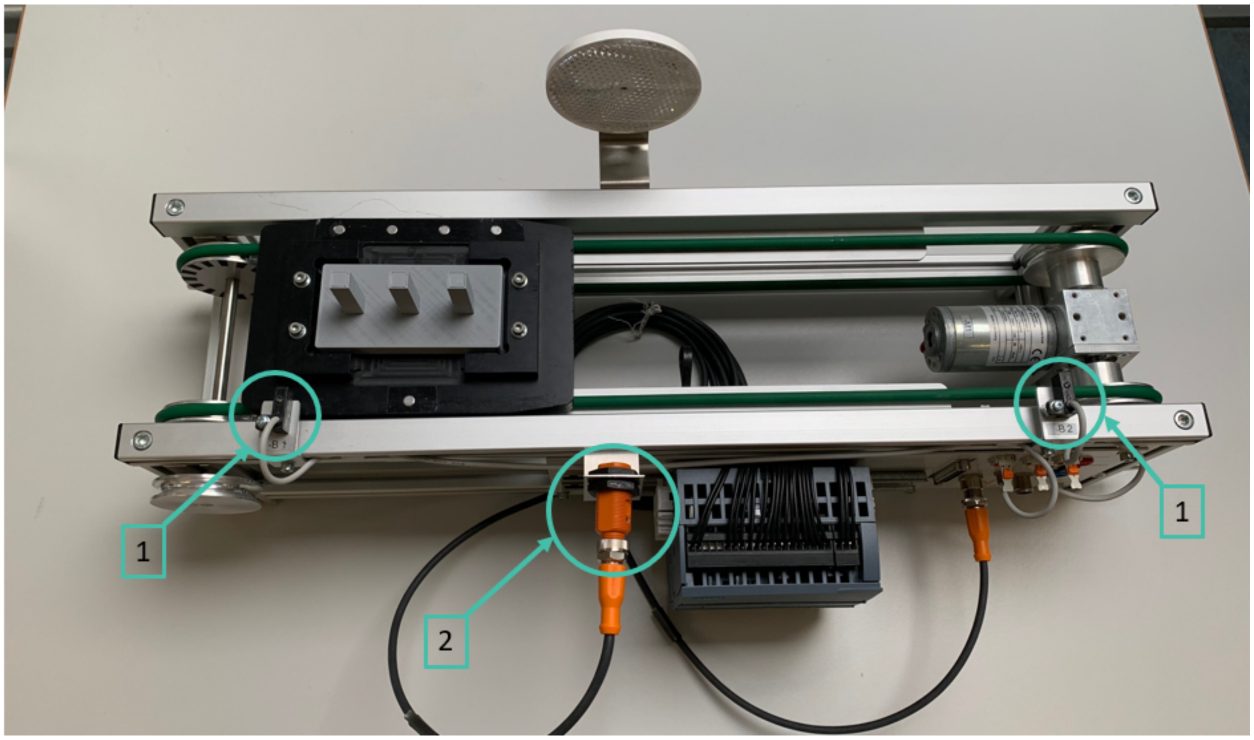


Fig. 3.1: The conveyor band in question

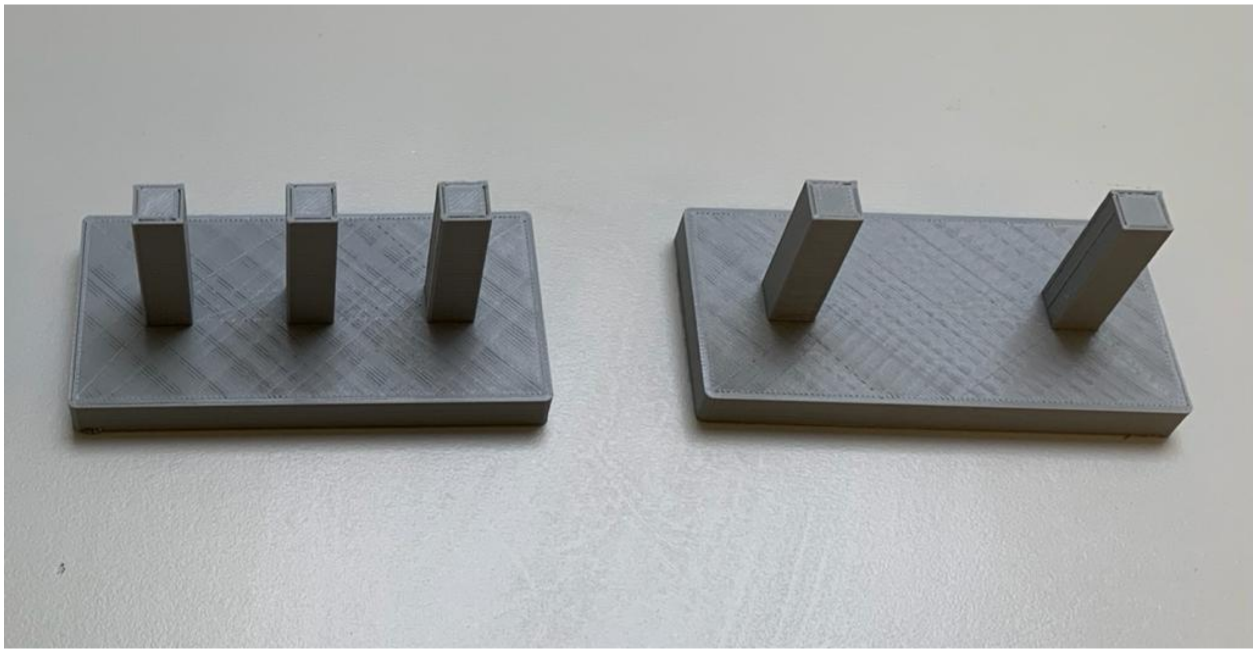


Fig. 3.2: Workpieces with different pillar counts

3.3.2 Preparation before the session

Todo: Draw a state machine diagram that describes the software to fulfill the above assignment.

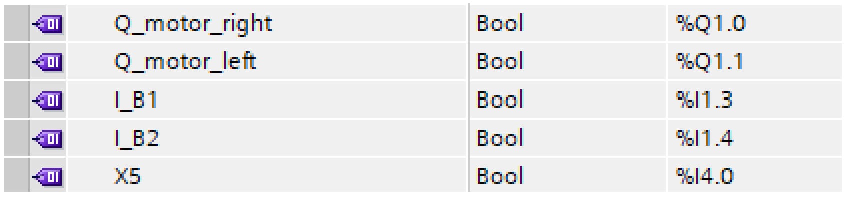
Read through and prepare *PLC-Basics* and *TIA Portal Basics* before the lab session.

3.4 Task

Hint: Refere to *TIA Portal Basics* for help in creating function blocks and other topics.

3.4.1 Tia Portal Project Setup

Follow *TIA Setup* to create and set up a project in TIA Portal. Create the necessary variables for all sensors and map them to their suitable addresses. The sensors and actors are wired to the following addresses.



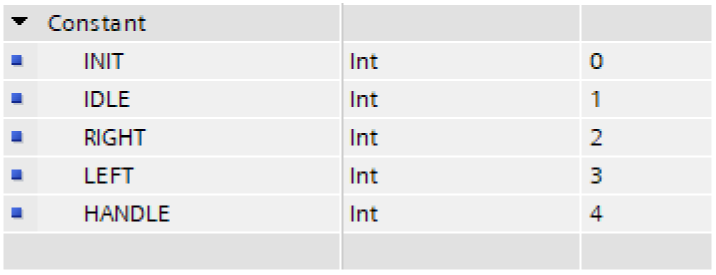
Q_motor_right	Bool	%Q1.0
Q_motor_left	Bool	%Q1.1
I_B1	Bool	%I1.3
I_B2	Bool	%I1.4
X5	Bool	%I4.0

Fig. 3.3: Signal address mapping

3.4.2 Function Block

Create a function block called `FB_ConveyorControl` and implement your state machine diagram from your preparation. Choose SCL as the programming language.

Hint: The Tia Portal environment does not support enumerations. Alternatively, you can use constants to represent your state. An example of this is shown in the following figure.



Constant		
INIT	Int	0
IDLE	Int	1
RIGHT	Int	2
LEFT	Int	3
HANDLE	Int	4

Fig. 3.4: An example state machine. Here constants are defined inside the function block to represent the different states.

3.4.3 Main [OB1]

Todo:

- Add one instance of `FB_ConveyorControl` to your `Main` program (drag & drop).
- Connect the IO to the function-block.
- Test if the program runs correctly.

Hint: Main [OB1] is written in FBD, but the function block should be written in SCL.

CHAPTER

FOUR

SIEMENS NX MCD: SIMPLE ROBOTIC ARM MODELLING

In this lab session you will model a simple robotic arm mechanism using Siemens NX.

Note: Please prepare all the `preparation` sections before the actual lab session. This is obligatory for attending and will save you time during the session. Preparation may include drawing diagrams, answering questions, or reading through the basics.

4.1 What you need

4.1.1 Software

- Siemens NX version 1872 or newer (pre-installed on the lab PC)

4.1.2 Files

CAD files of the examples used in this module can be found here: https://fh-aachen.sciebo.de/s/AFAGrOi2zRFL1y7

For this task, the correct assembly can be found in `simple_robotic_arm`. Download all the parts from the `simple_robotic_arm` file and open `ASSEMBLY_simple_robotic_arm` in Siemens NX.

4.2 Preparation

- *Siemens NX MCD*

4.3 Assignment

In this exercise a running simulation of a simple robotic arm mechanism will be created. The CAD files are available. In Siemens NX MCD, different physical properties will be assigned to parts in order to reflect the function of the simple station.

Note: When you first start NX, you may get an error saying there are no licenses available. Click "OK" and navigate to File->Utilities->Select Bundles. In the window that opens, select and add both available bundles then click `OK`. This solves the error.

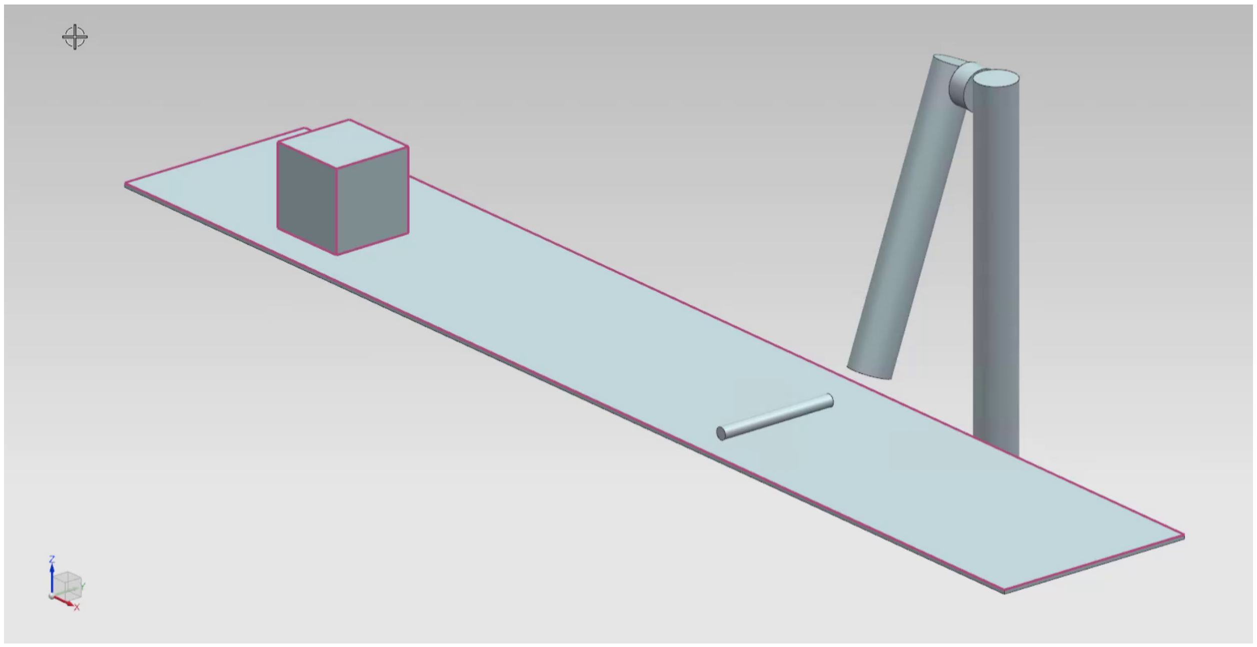


Fig. 4.1: Simple robotic arm functionality

4.3.1 Box and Conveyor Band

In the first steps, rigid and collision bodies will be assigned to parts the will interact with each other.

Todo:

- Assign a rigid body to the box and call it `Box`.
- Assign a collision body to the box and call it `Box`. Hint: to choose the entire box volume, right click on the box and select `Choose from list`. Then select the entire box from the list.
- Assign collision body to the upper surface of the transporter band and call it `Transport_band`.
- Assign a transport surface to the upper surface of the transporter band and call it `Transport_band`. Specify the correct direction vector and set the parallel velocity to `100 mm/s`.
- Run the simulation and check if the box is transported on the transport band.

4.3.2 Simple Robotic Arm

Todo:

- Assign a rigid body to the robotic arm and call it `Robot_arm`.
- Assign a collision body to the robotic arm and call it `Robot_arm`.
- Assign a hinge joint to the robotic arm and call it "Robot_arm_HingeJoint". Specify the direction vector and the anchor point in a way that it rotates around the robotic joint.
- Assign a position control to the hinge joint object. Test different `Destination` and `Speed` settings.
- Finally set the destination to `0` and the speed to `500 °/s`

4.3.3 Collision Sensor

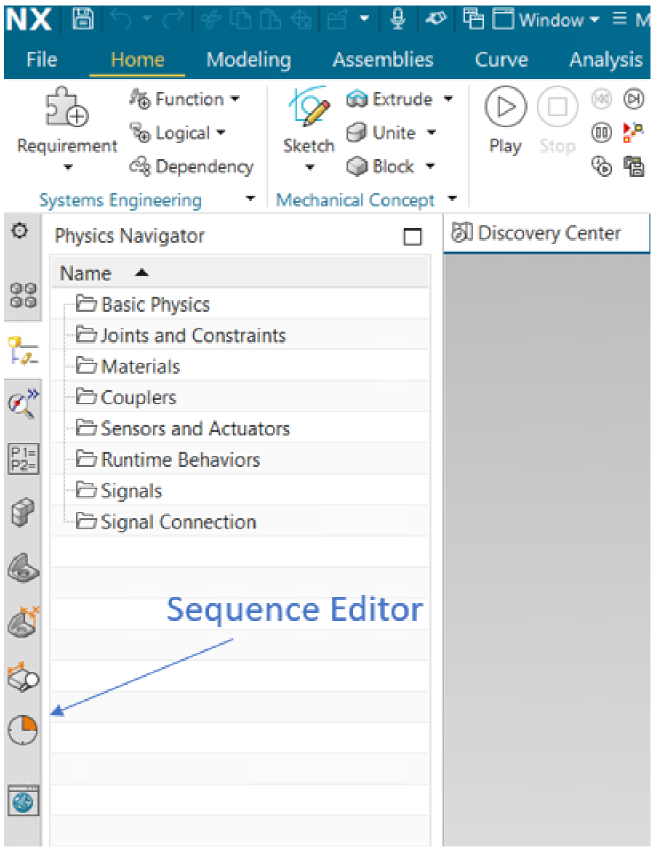
Todo:

- Assign a collision sensor to the small cylinder on the transport band and give it a suitable name.

4.3.4 Conditions - Using Operations as if-Statements

Todo:

- Create an operation in the `Sequence Editor`. Choose the position control object as the `Select Object` choice. Make a check next to `position` and give in a value of `70`. Select the collision sensor object (from the) as the condition object and specify that `triggered == true`. Name the operation `Arm_out`.



- Create a second operation. Choose the position control object as the `Select Object` choice. Make a check next to `position` and give in a value of `0`. Select the collision sensor object (from the `Physics Navigator`) as the condition object and specify that `triggered == false`. Name the operation `Arm_in`.

Run the simulation and check if it achieves the desired behavior.

CHAPTER

FIVE

OPC UA: SIMPLE ROBOTIC ARM MODELLING

In this lab session you will connect a simulation in NX MCD to a simulated PLC.

Note: Please prepare all the `preparation` sections before the actual lab session. This is obligatory for attending and will save you time during the session. Preparation may include drawing diagrams, answering questions, or reading through the basics.

5.1 What you need

5.1.1 Software

- Siemens NX version 1872 or newer (pre-installed on the lab PC)
- PLCSIM Advanced V3

5.1.2 Files

CAD files of the examples used in this module can be found here: https://fh-aachen.sciebo.de/s/AFAGrOi2zRFL1y7

The correct assembly can be found in `OPC_UA`. Download all the parts from the `OPC_UA` file and open `ASSEMBLY_simple_robotic_arm_S_OPC_UA` in Siemens NX.

For this task, you will find the PLC program that controls the arm in `simple_robotic_arm_PLC_program_V15.1`. Download that folder as you will need to open it later with TIA Portal (change the file ending to .ap15_1 instead of .info).

5.2 Preparation

- *Siemens NX MCD*

5.3 Assignment

In this exercise we will connect a running robotic arm simulation to a PLC over OPC-UA. The CAD files as well as the PLC program are available.

Note: When you first start NX, you may get an error saying there are no licenses available. Click "OK" and navigate to File->Utilities->Select Bundles. In the window that opens, select and add both available bundles then click `OK`. This solves the error.

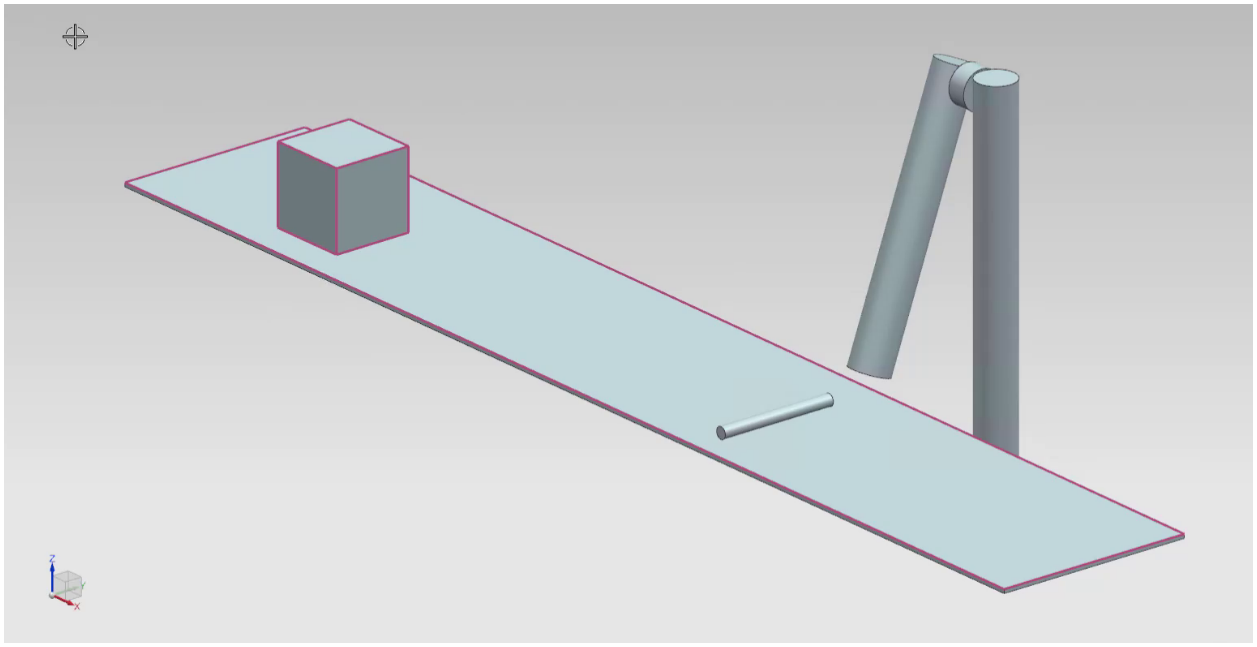


Fig. 5.1: Simple robotic arm functionality

5.3.1 Simple Robotic Arm

Start NX and open the `ASSEMBLY_simple_robotic_arm` file.

Todo:

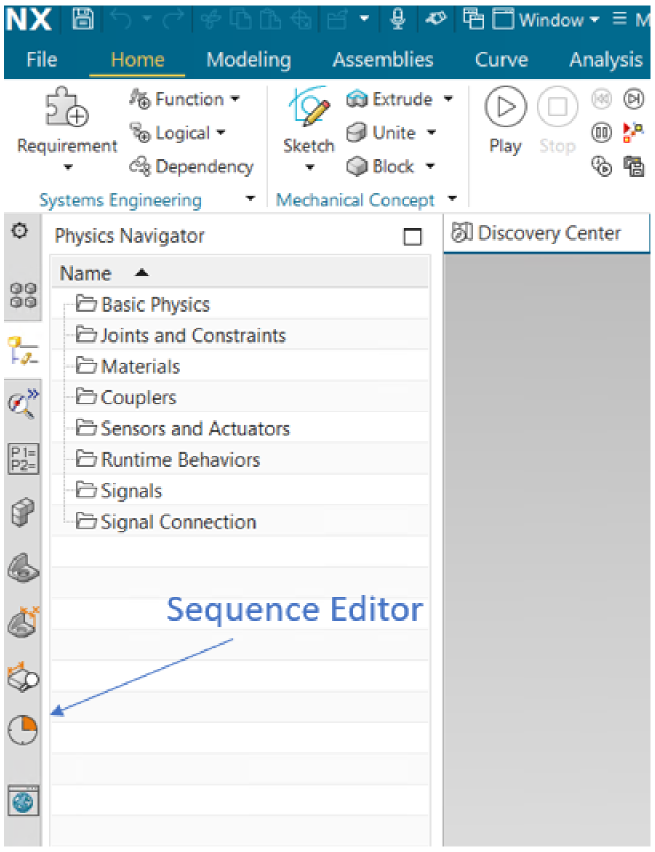
- Create a signal and call it `bSensor`.
- Link the `bSensor` signal to a runtime parameter. Choose the collision sensor object as the runtime object.
- Create a signal and call it `bRobotic_arm_out`.
- Set the `bRobotic_arm_out` signal as Input of data type bool.

5.3.2 Conditions - Using Operations as if-Statements

The `bRobotic_arm_out` is a signal that is controlled by the PLC. Here we link that signal to the action of moving the robotic arm in the simulation.

Todo:

- Create an operation in the `Sequence Editor`. Choose the position control object as the `Select Object` choice. Make a check next to `position` and give in a value of `70`. Select the `bRobotic_arm_out` signal (from the `Physics Navigator`) as the condition object and specify that `triggered == true`. Name the operation `Arm_out`.



- Create a second operation. Choose the position control object as the `Select Object` choice. Make a check next to `position` and give in a value of `0`. Select the `bRobotic_arm_out` signal (from the `Physics Navigator`) as the condition object and specify that `triggered == false`. Name the operation `Arm_in`.

We will check if the desired behaviour is achieved after we set up the communication between the PLC and the arm simulation.

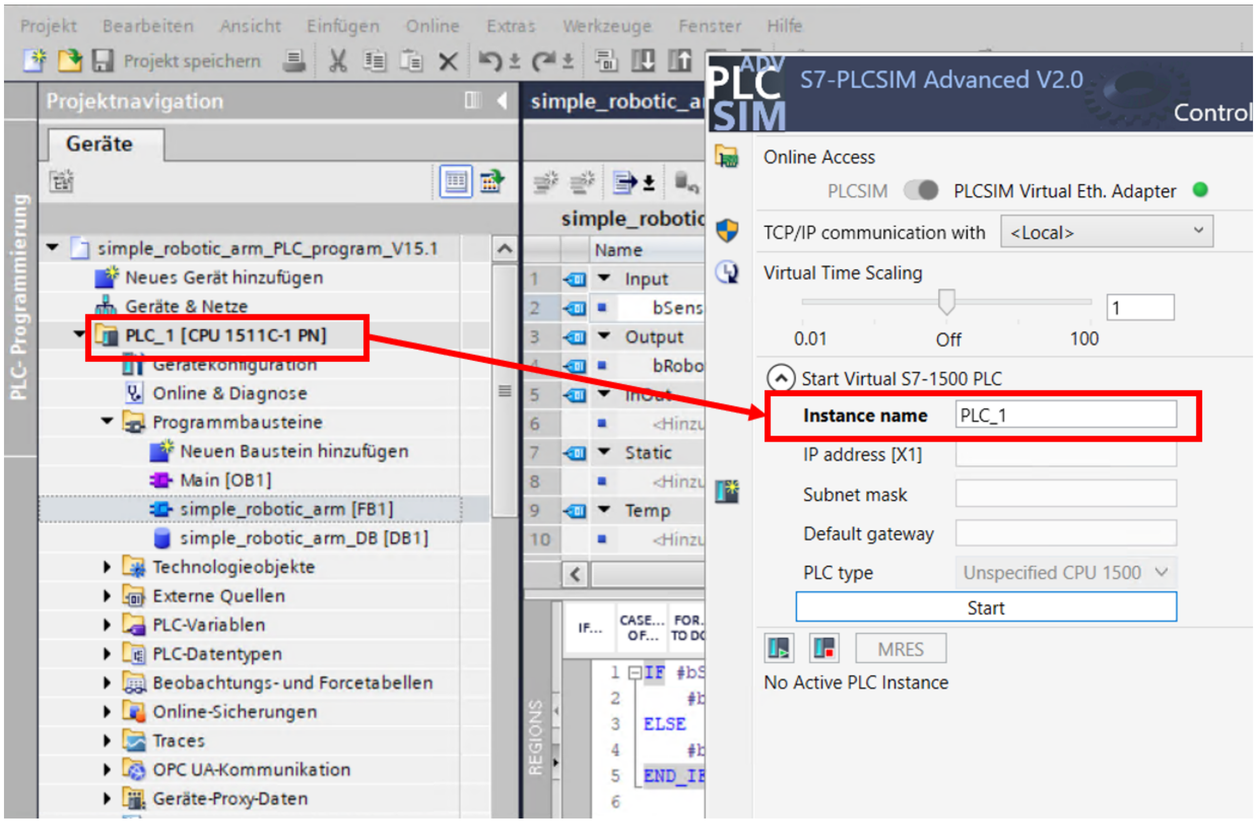
5.3.3 TIA Portal

Open the project downloaded from the Sciebo downloads folder. The project's name is `simple_robotic_arm_PLC_program_V15.1`. You can open it by double clicking on the project's .ap15_1 file. This will start a TIA Portal instance. This PLC program is complete and only needs to be uploaded onto a PLC. In the next step, we will simulate a PLC in order to upload our project onto it.

5.3.4 PLCSIM Advanced

Todo: Start PLCSIM Advanced and start a PLC simulation. Use *PLCSIM Advanced* for a step-by-step guide.

Note: Make sure the PLC simulated has the same name as the PLC in the TIA Portal program. In the downloaded project, the name is PLC_1.



Note: Make sure `PLCSIM Virtual Eth. Adapter` is selected in PLCSIM Advanced.

Todo: In TIA Portal, compile your program and upload it to the simulated PLC.

5.4 Establishing OPC-UA Communication

Todo: Establish an OPC UA connection between MCD and the simulated PLC. Refer to *OPC UA and Signal Mapping* for a guide on how to do that.

Todo: In the `Signal Mapping` window in MCD, click the option `Do Auto Mapping`. This will automatically map identically-named signals to each other.

Todo: Run the simulation and watch the PLC's variables in a watch table. The production station should now be controlled through the simulated PLC.

CHAPTER

SIX

INTRODUCTION TO PNEUMATICS

6.1 Actuators

The single acting cyclinder has only one inlet. If a high pressure (i.e., pressurized air) is applied at the inlet, the cylinder will extend, acting against the spring spring force inside the cylinder and releasing the air directly at the cylinder. If a pressure lower than the atmospheric pressure is applied at the inlet, the cylinder will retract due to the spring force. The single acting cylinder retracts automatically once no high pressure is applied at the inlet anymore.

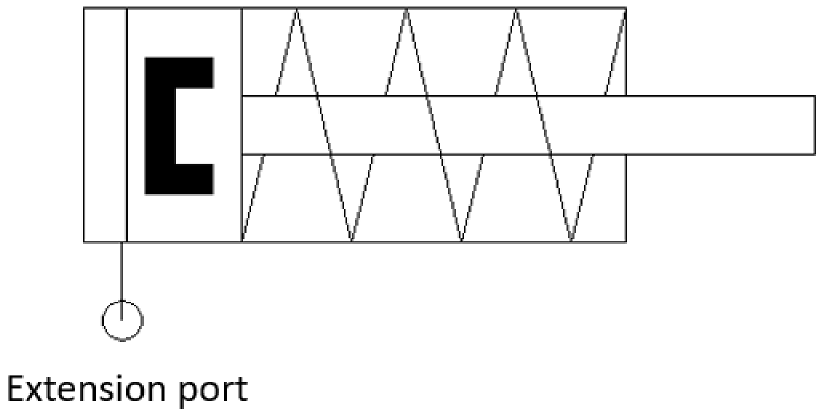


Fig. 6.1: FluidSIM single acting cylinder symbol

The double acting cylinder on the other hand has two inlets. If a high pressure is applied at either one, the cylinder will more (either extend or retract, depending on with inlet recieves the high pressure). Unlike the single acting cylinder, the double acting cylinder will remain in its current position even after no high pressure is recieved through the inlet anymore. This is due to the absense of a spring inside. To reverse the cylinder position, pressurized air must be applied on the opposite inlet.

Note: A 5/2 valve is way more commonly used in the industry than a 4/2 valve

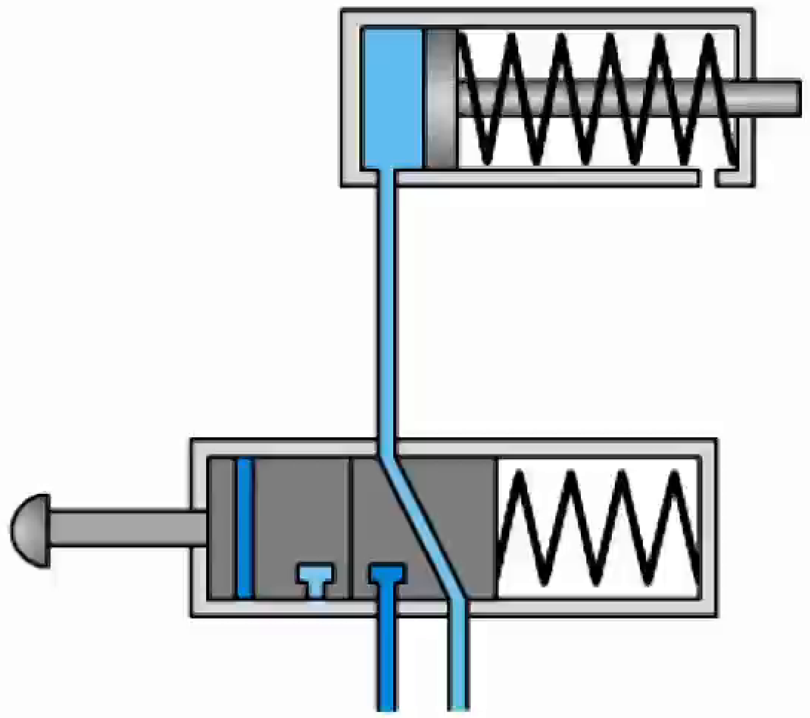


Fig. 6.2: The working principle of a single acting cylinder with a 3/2 way valve

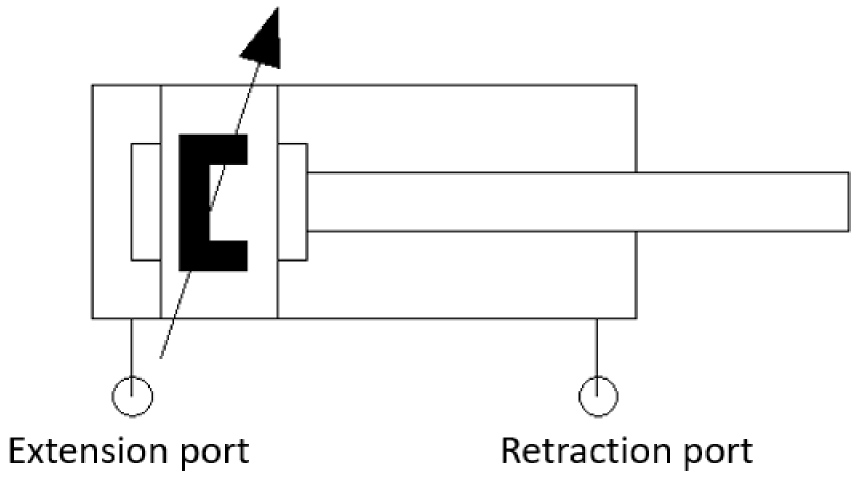


Fig. 6.3: FluidSIM double acting cylinder symbol

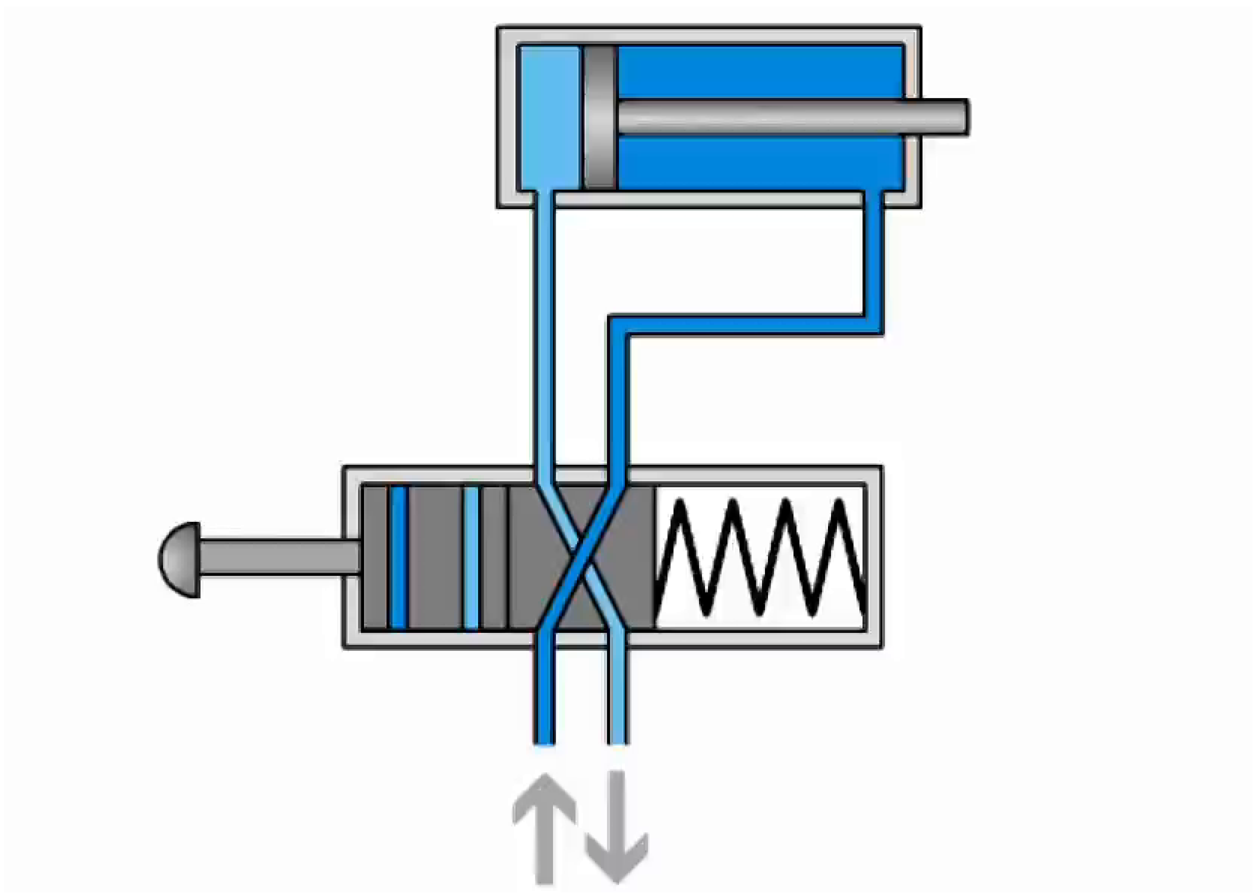


Fig. 6.4: The working principle of a double acting cylinder with a 4/2 way valve.

6.2 Directional Valves

The goal is to control the pressurized air direction to actuate the pneumatic cylinders. The most used valve is a 3/2 way directional valve.

Note: In a 3/2 way directional valve, the '3' stands for the number of connectors and the '2' stands for the number switching positions.

Port 1 has is to be connected with the air inlet (i.e., pressurized air source). Port 2 is considered the working port and is to be connected to a valve (e.g., a single acting cylinder). Finally, port 3 is the exhaust port, which is usually connected to an air outlet object in FluidSim.

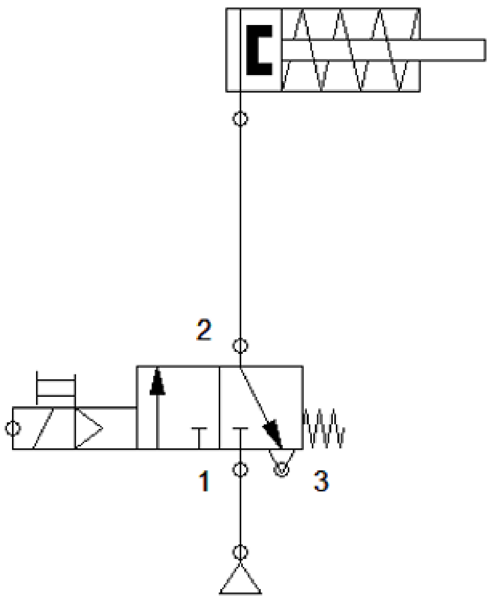


Fig. 6.5: FluidSIM pneumatic circuit with a 3/2 way directional valve connected to a pressurized air source and a single acting cylinder

For a double acting cylinder, a Directional Control Valve DCV (5/2 way valve) is needed. This valve has two outputs and they are connected to the two inlets of the double acting cylinder. The valve has 3 inlets (1 inlet and 2 exhaust outputs).

6.3 Throttle Valves

The speed is limited by the air pressure, the flowrate and the spring force (in case of a single acting cylinder). The next goal is to control the speed of the pressurized air, and thus the actuation speed of the cylinders. To do that, we use valves to limit the speed of the pressurized air.

A throttle valve limits the air flow in both directions.

A check valve completely blocks any air flow in one direction, while allowing air from with no resistance in the opposite direction.

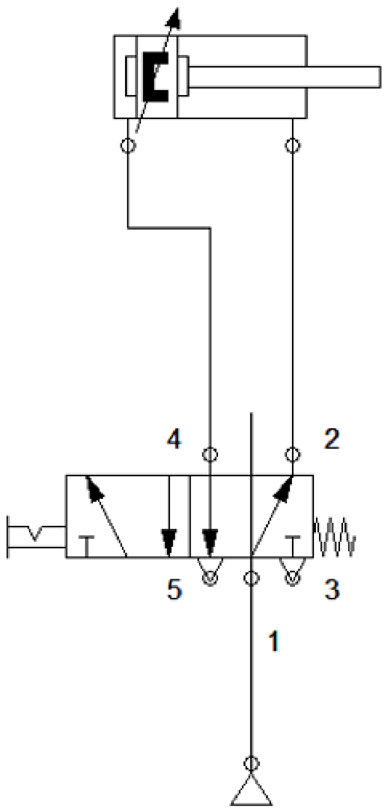


Fig. 6.6: FluidSIM pneumatic circuit with a 5/2 way directional valve connected to a pressurized air source and a double acting cylinder

Note: This is comparable to the diode behavior in electronics, where current is allowed to flow in only one direction and is blocked in the opposite direction.

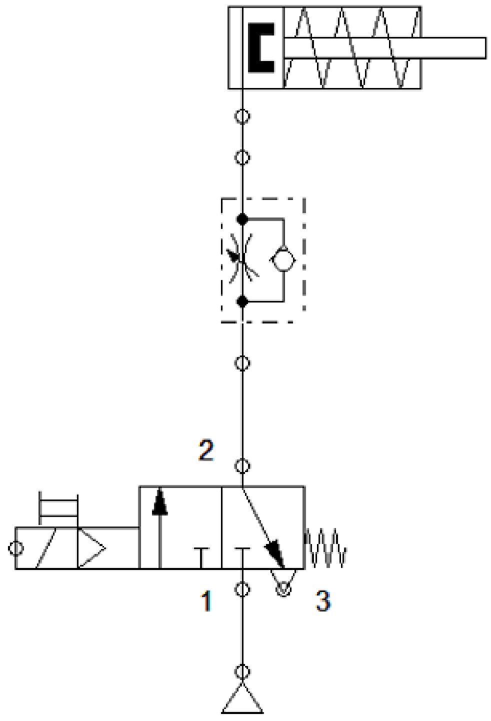


Fig. 6.7: A check valve to control air flow to a single acting cylinder

A throttle check valve limits the air flow in only one direction. In the symbol, the air going from 1 to 2 is stopped by the check valve (represented as a circle in the symbol), which blocks air flow completely along that path and thus air has to go through the throttling path. In the other direction (from 2 to 1), the check valve allows the air to pass through it with no resistance.

For a double acting cylinder, it is best practice to control the air flow on the outlet path. To control the speed in both the extension and the retraction paths, a throttle check vavle is placed in both paths.

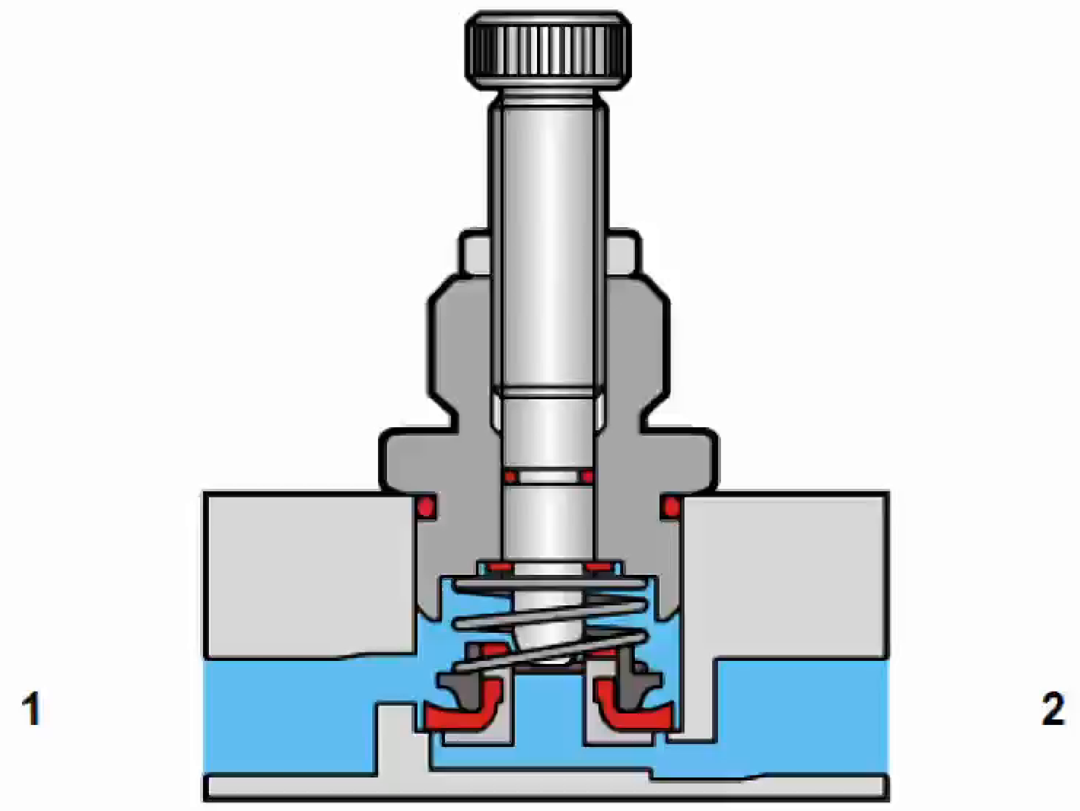


Fig. 6.8: The working principle of a throttle check valve

CHAPTER

SEVEN

FESTO FLUIDSIM

In this module, you'll be using FluidSIM Pneumatik V4.5 Meclab. Further documentation is available online at https://www.festo-didactic.com/ov3/media/customers/1100/698522_fl_sim_p42_de_offset.pdf

7.1 Creating a new project

In order to create a new project, click on `File` `New`. Alternatively, you can click on the `New` icon on the top bar.

7.2 Building circuits

Components can be dragged and dropped from the components window on the left side into the workspace on the right side. Pneumatic, electrical and logical circuits can be constructed this way.

7.3 Starting/Stopping the simulation

The start/stop buttons in the top bar can be used to start/stop the simulation. If there are no errors in the circuits, the simulation should run and the start icon should turn green.

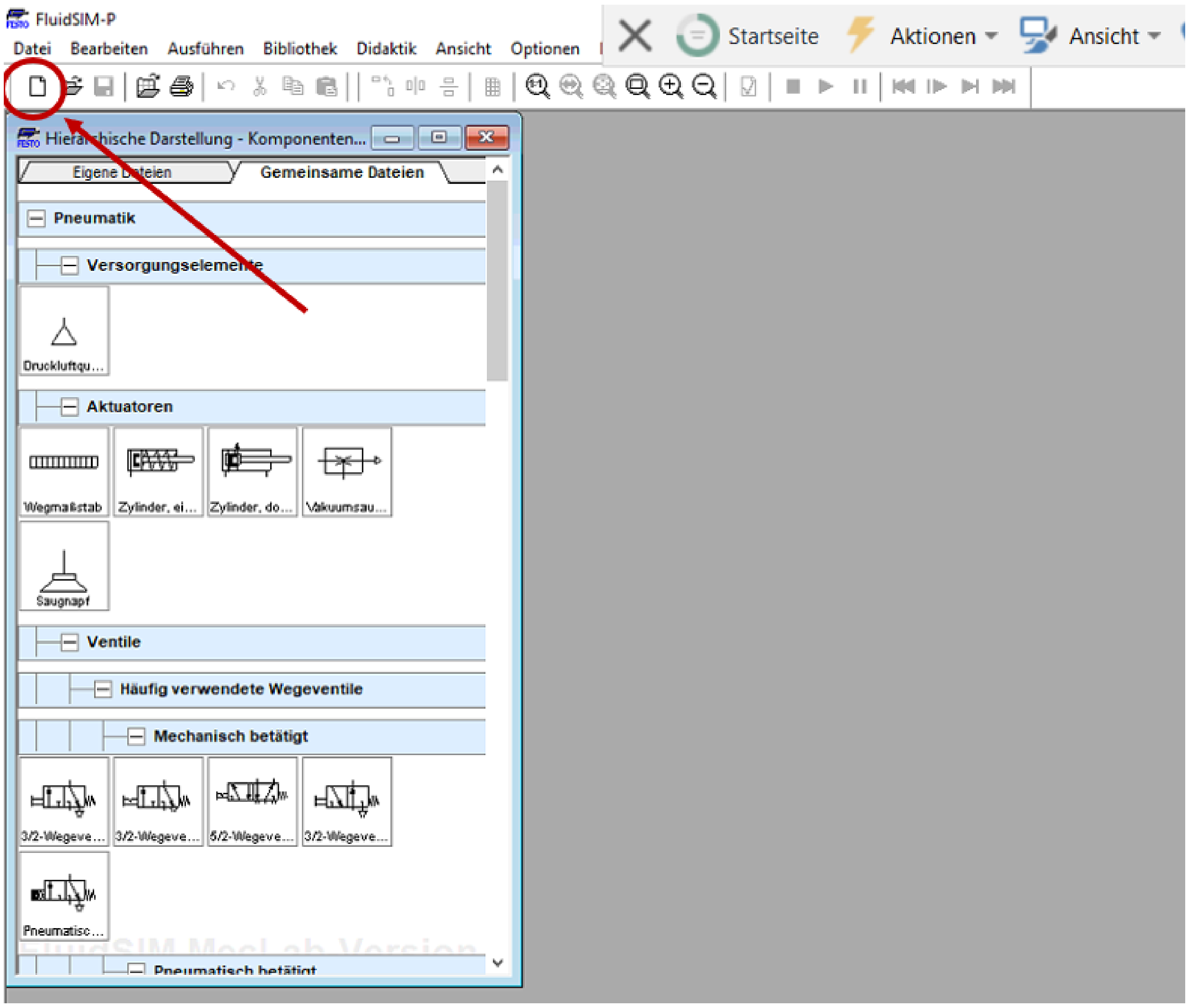


Fig. 7.1: Creating a new project

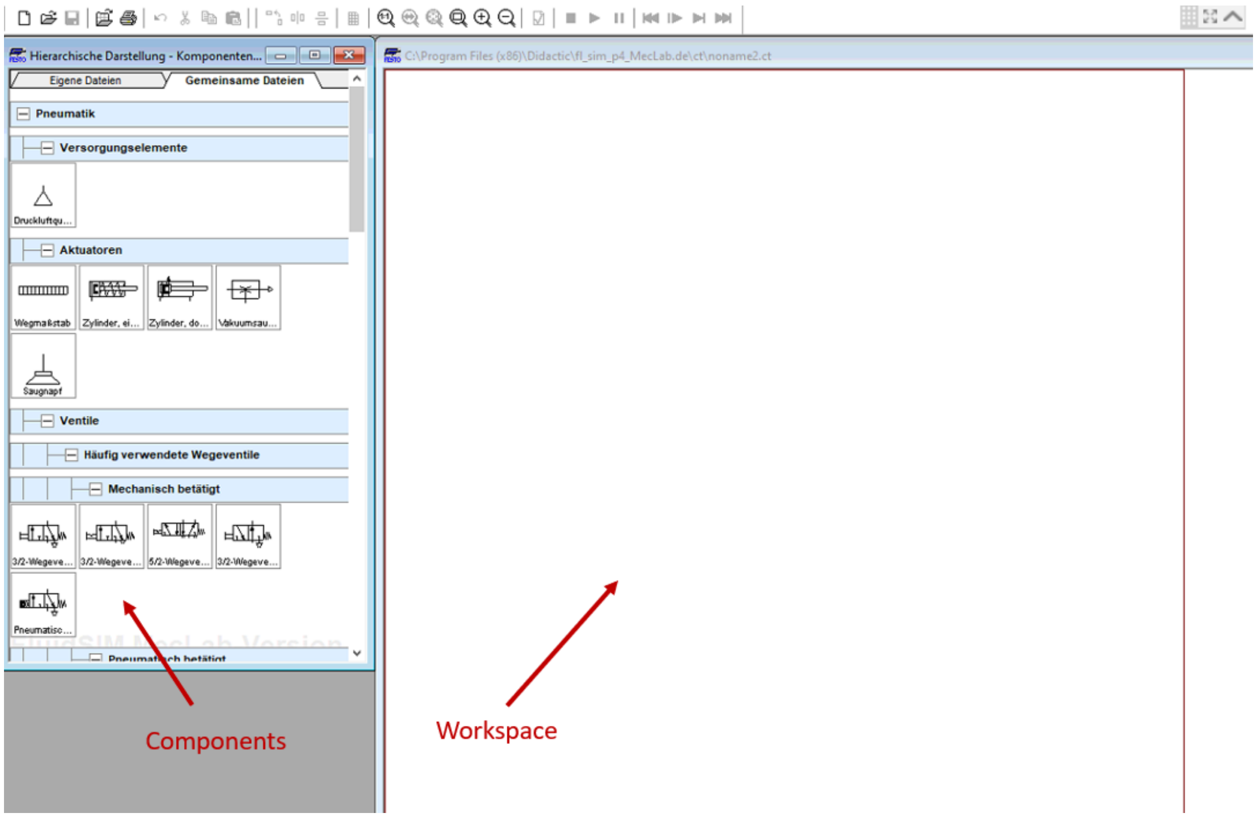


Fig. 7.2: FluidSIM environment

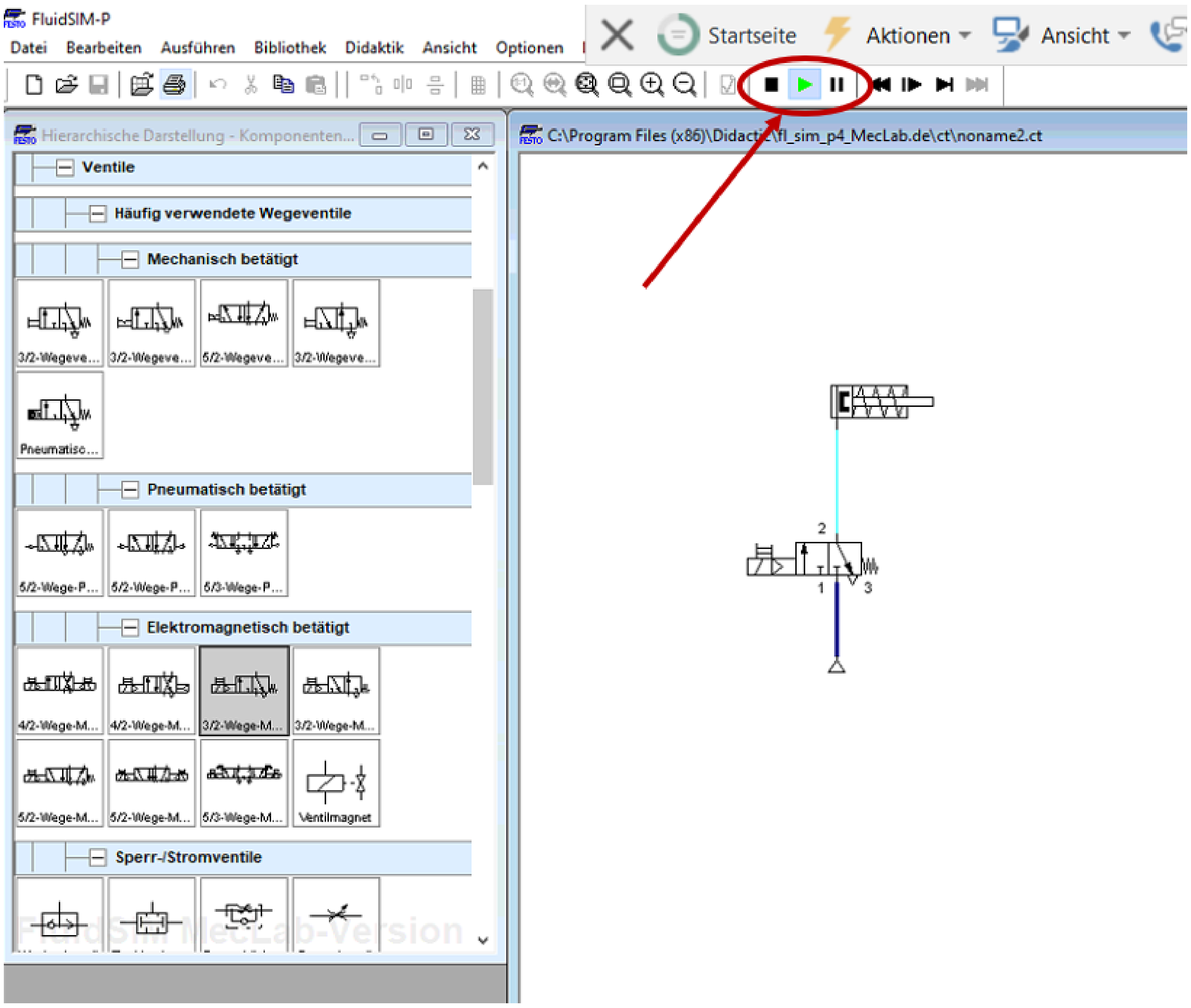


Fig. 7.3: FluidSIM environment

CHAPTER

EIGHT

PLC-BASICS

Programmable Logic Controllers (PLC) are computers which are commonly used in commercial and industrial control applications. Generally speaking, PLCs monitor inputs and other variable values, make decisions based on a stored program and control outputs to automate a process or machine.

A PLC typically consists of input and output modules, a Central Processing Unit (CPU) and a programming device. The primary function of the input unit is to convert the signals at the inputs into logic signals which can be used by the CPU. The CPU analyzes the status of inputs, outputs and other variables and executes the stored program. After that, the CPU changes the output signals.

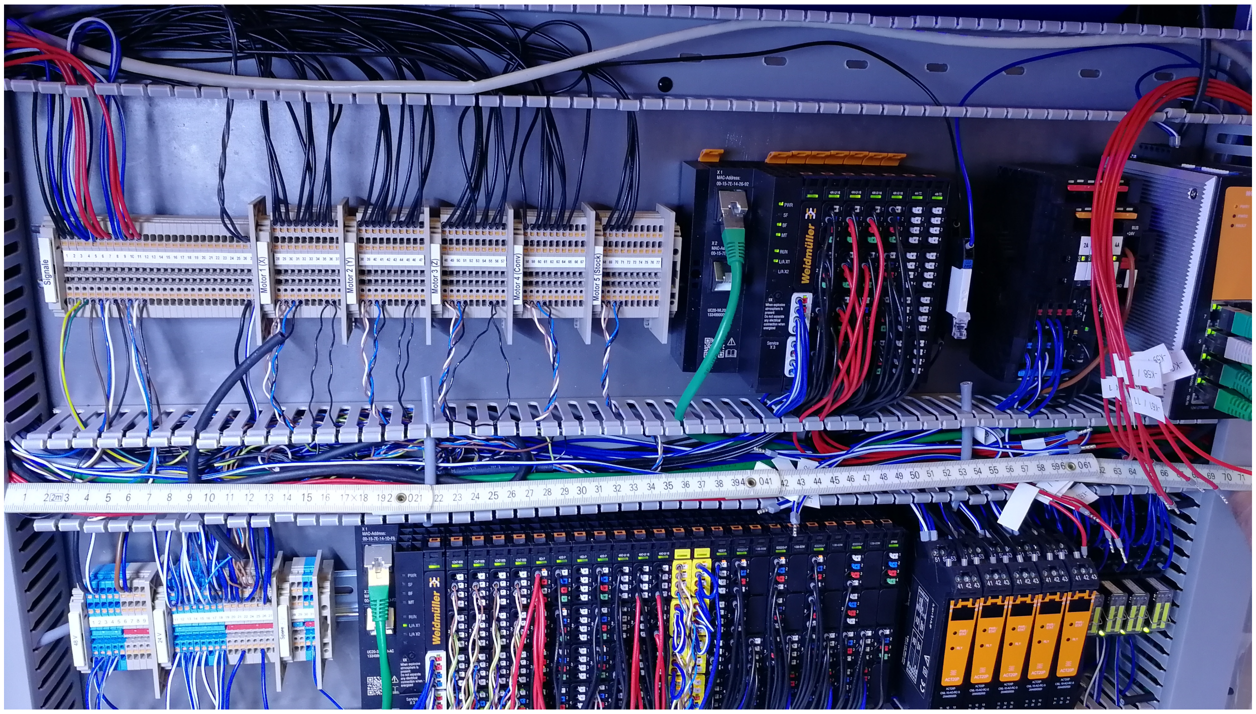


Fig. 8.1: Example for PLC wiring for industrial applications

Advantages in contrast to hard-wired control:

- Easy to modify input and output devices
- High flexibility due to modular design
- Significant reduction of cabling
- Solid-state, no moving parts

- Smaller physical size
- Integrated diagnostics and override functions
- Easy and cost-effective duplication possible
- Communication capabilities

8.1 Information processing

Information at the inputs of the PLC are processed in cycles. First, the CPU queries the input channels and stores the data in the working memory. This storage area is called "input image" as the stored input data does not show the current state of the inputs but the available data at the time of sampling.

Next, the program is executed step by step and the variables are stored in the memory. Finally, the calculated output parameters are stored in the "output image" and transferred to the output channels to control the connected machine.

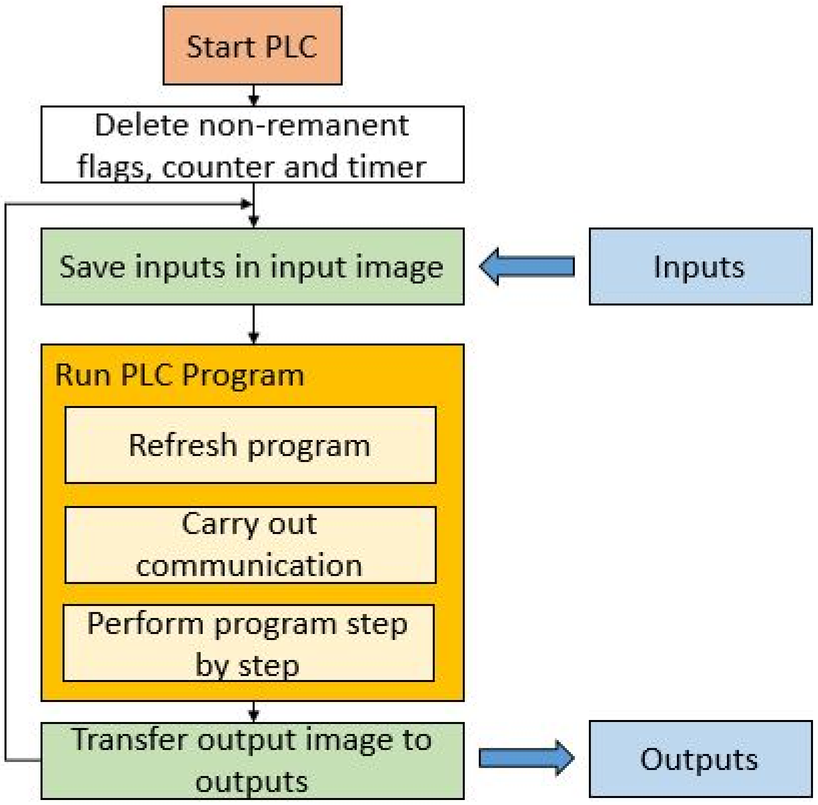

Start PLC
Delete non-remanent flags, counter and timer
Save inputs in input image
Inputs
Run PLC Program
Refresh program
Carry out communication
Perform program step by step
Transfer output image to outputs
Outputs

Fig. 8.2: Cyclic information processing of PLCs

Types of input and output signals:

- Binary inputs: Input module can distinguish between a high and a low level of the input signal (typical value: 0V and 24V).
- Binary outputs: Output modules generate TRUE (5V) or FALSE (0V) values which are typically intensified (24V) using transistors or relays.
- Analog inputs: A physical measurand is transformed into a voltage or current signal using a transducer. The resulting signal is transferred to the analog inputs of the PLC.

- Analog outputs: A voltage or current signal is generated to control actuators.

8.2 PLC Programming (IEC 61131)

PLC programming is standardized in the IEC 61131.

8.2.1 Controller configuration and resources

Creating a controller configuration, the hardware structure is specified in the programming environment. All resources which are controlled by the PLC software (input, output modules, interface cards, etc.) are declared. Nowadays, this is usually done automatically by scanning the connected components or by drag-and- dropping the parts.

Input channels are declared using `%I` and outputs specified writing `%Q`. The following letter defines the data type of the signal (bit `X`, byte `B`, word `W`, doubleword `D`). In this context, the declaration `%IX1.2` calls the second channel of the first binary input card.

Labelling of input and output addressing	1. Letter	2. Letter	Example
AT%	I Input	x Bit	AT%IX1.2
	Q Output	B Byte, 8 Bit	AT%QB0
		W Word, 16 Bit	AT%QW7
		D Doubleword, 32 Bit	AT%QD5

8.2.2 Tasks

Tasks organize the time schedule of programs. It works a bit like a cyclic step counter, which selects the instructions of the program in a clock-controlled way. Based on that, the CPU processes the instruction which is stored in the selected storage area.

Several programs can be assigned to one task running all in the same cycle time. The following process is conducted (IPO principle; input-processing-output):

- Read input data of all programs.
- Processing of all programs.
- Output of the output data of all programs.

The cycle time of all programs assigned to one task is identical as the output data of all programs is transferred simultaneously at the end of the processing cycle.

One CPU can process multiple tasks with different cycle times (multitasking). Therefore, the computing power is distributed between the different tasks using interrupt signals for changing.

8.2.3 Program Organization Units (POUs)

An object of the type `POU` is a Program Organization Unit in a CODESYS project. You write source code for the controller program in `POUs`. There are the following types of POUs:

- Programs
- Function Codes `FC`
- Function Blocks `FB`

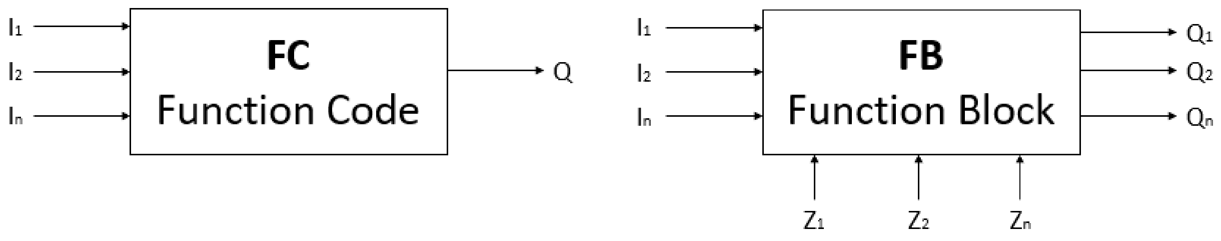


Fig. 8.3: Difference between Function Codes and Function Blocks

Function Codes can have several input variables but only one output variable. This variable is the return value of the Function Code. Function codes have no internal memory, thus they cannot store intermediate values. Function Codes can be separated between standard and user-specific functions. Standard functions like for example AND, ADD or BYTE_TO_WORD can be used directly during programming without declaring it separately.

Function Blocks can also be standardized or user-specific. Examples for pre-defined FBs are triggers, counters and timers. FBs can store intermediate values for the next IPO cycle and can have several outputs `Q`. The output signals are created by connecting the inputs `I` and the states `Z`.

8.2.4 Variables

Variables are used to establish a communication between different program organization units. Different kinds of variables are available:

- `VAR`: Local variables which are only valid in the associated POU.
- `VAR_GLOBAL`: Global variables are valid in all POUs. They are used for communication between different programs.
- `VAR_INPUT`: Input variables which are used to write inputs in FCs or FBs.
- `VAR_OUTPUT`: Output variables of FCs or FBs.
- `VAR_IN_OUT`: Input and output variables can be changed in the FB and then be released.
- `VAR_RETAIN`: Variables retain the value when the PLC is put off and on again.
- `VAR_PERSISTENT`: Variables retain the value if the software is loaded on the PLC.

Standard data types:

	Datatype	Size	Range	Example
Bit sequence	BOOL	1 Bit	FALSE / TRUE	FALSE
	BYTE	8 Bit	16#00 … 16#FF	16#00
	WORD	16 Bit	16#0000 … 16#FFFF	16#0000
	DWORD	32 Bit	16#00000000 … 16#FFFFFFFF	16#00000000
Whole numbers	SINT	8 Bit	-128 … 127	0
	INT	16 Bit	-32768 … 32767	0
	DINT	32 Bit	-2147483648 … 2147483647	0
Whole numbers without sign	USINT	8 Bit	0 … 255	0
	UINT	16 Bit	0 … 65535	0
	UDINT	32 Bit	0 … 4294967295	0
Floating point figure	REAL	32 Bit	-3.4 * 10^38 … 3.4 * 10^38	0
Time	TIME			
Time of day	TIME_OF_DAY			
Date	DATE			
Character sequence	STRING			

A typical variable declaration consists of five parts:

`<variable name> AT<address> :<data type> :=<initial value>; (* comment *)`

example: `buttonStart AT%IX0.0 :BOOL :=FALSE; (*Start Button*)`

Note: It is helpful to use speaking names instead of numbers to make it easy to understand the program (e.g. buttonLight and buttonVentilator instead of button1 and button2).

Derived data types:

Standard data types can be modified to fulfil special needs: `TYPE Name : ... END_TYPE`

Four different kinds can be distinguished:

- Enumeration: `TYPE A_STATUS : (START,RUN,WAIT,STOP); END_TYPE`
- Area: `TYPE A_DATA : UNIT(0..16#3FF); END_TYPE`
- Field: `TYPE A_4IN : ARRAY[1..4] OF A_DATA; END_TYPE`
- Structure: `TYPE str_Motor : STRUCT ... END_STRUCT; END_TYPE`

Enumerations are typically used if different options are possible and if easy readable code should be produced.

8.3 Programming languages (IEC 61131)

Programmable Logic Controller (PLC) programming as other programming tasks has defined set of rules described in the IEC 61131-3. In this Standard, information about programming concepts and industry accepted programming languages is provided. For this module, SCL, LAD and FBD are selected as they are most commonly used.

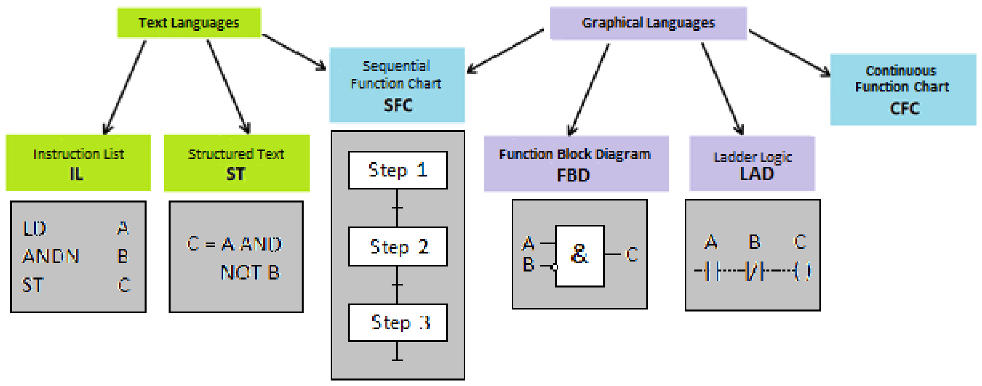
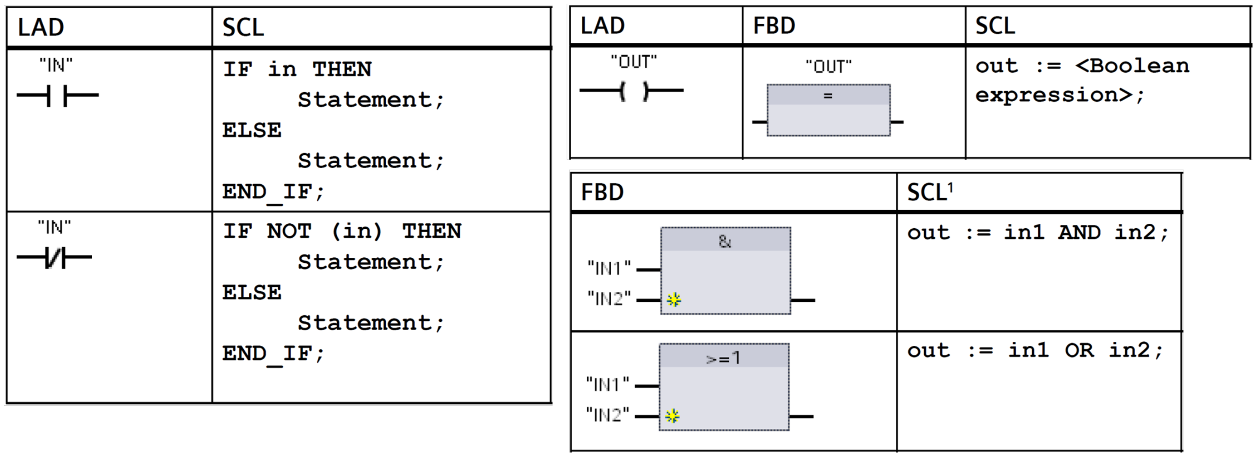


Fig. 8.4: Five programming languages of the IEC 61131-3 standard (source: https://www.motioncontroltips.com/iec-61131-3-plcopen/)



LAD	SCL
"IN"	`IF in THEN Statement; ELSE Statement; END_IF;`
"IN"	`IF NOT (in) THEN Statement; ELSE Statement; END_IF;`

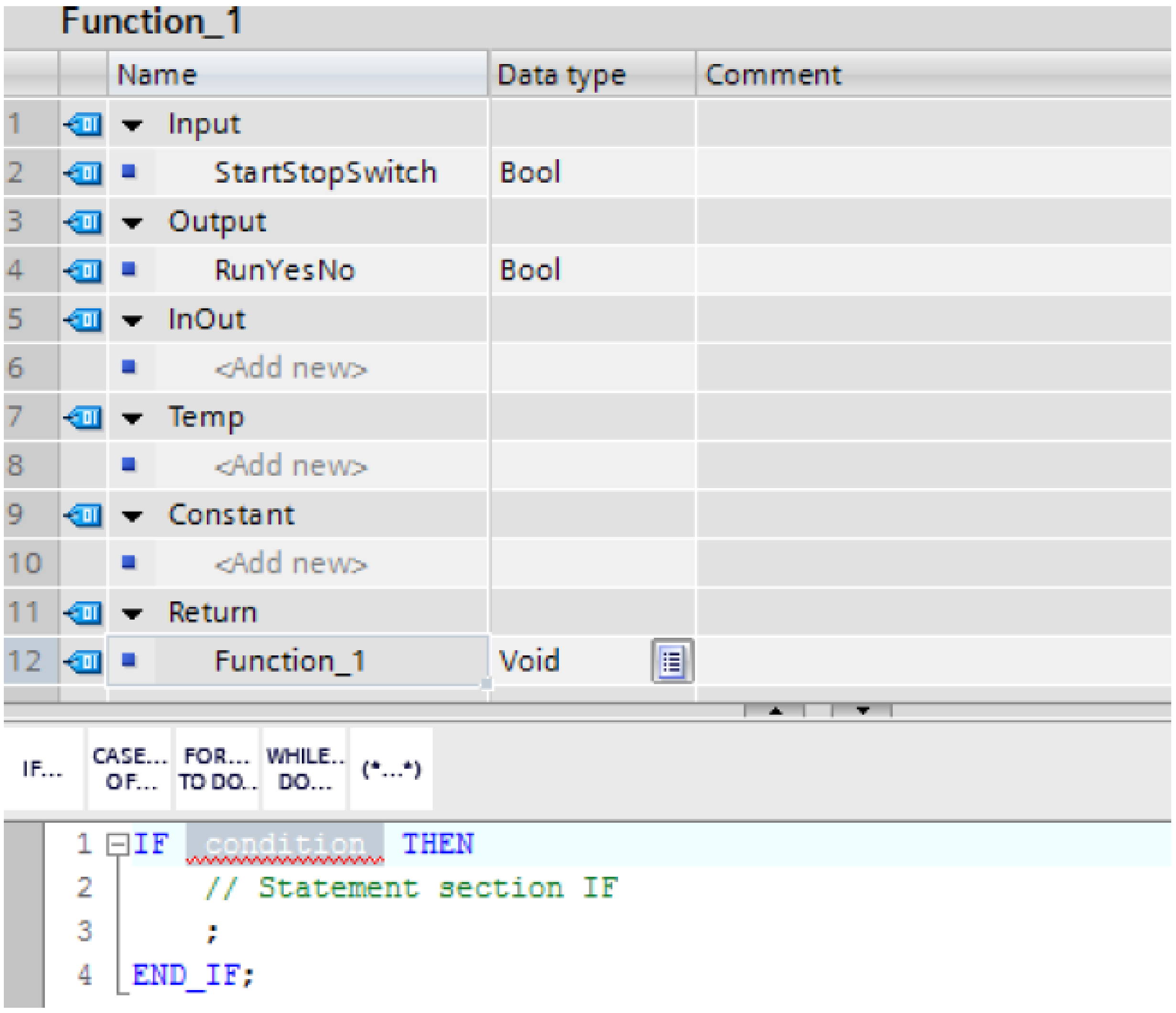
LAD	FBD	SCL
"OUT"	"OUT" =	`out := <Boolean expression>;`

FBD	SCL[1]
& "IN1" "IN2"	`out := in1 AND in2;`
>=1 "IN1" "IN2"	`out := in1 OR in2;`

Fig. 8.5: SCL, LAD & FBD (ref: Siemens S7-1200 Programmable controller System Manual)

8.3.1 Structured Control Language (SCL)

SCL (Structured Control Language) is a high-level text-based programming language. It is based on PASCAL.


Function_1

	Name	Data type	Comment
1	Input		
2	StartStopSwitch	Bool	
3	Output		
4	RunYesNo	Bool	
5	InOut		
6	<Add new>		
7	Temp		
8	<Add new>		
9	Constant		
10	<Add new>		
11	Return		
12	Function_1	Void	

```
IF condition THEN
    // Statement section IF
    ;
END_IF;
```

Fig. 8.6: An example of a SCL code 1.

Various important syntax for SCL are provided below:

Assignments: `A := 10;` (The variable A is assigned the value 10.)

Statements: `blink : BOOL;` (Create a variable `blink` and assign the type `BOOL`.)

Mathematical functions: +, -, *, /

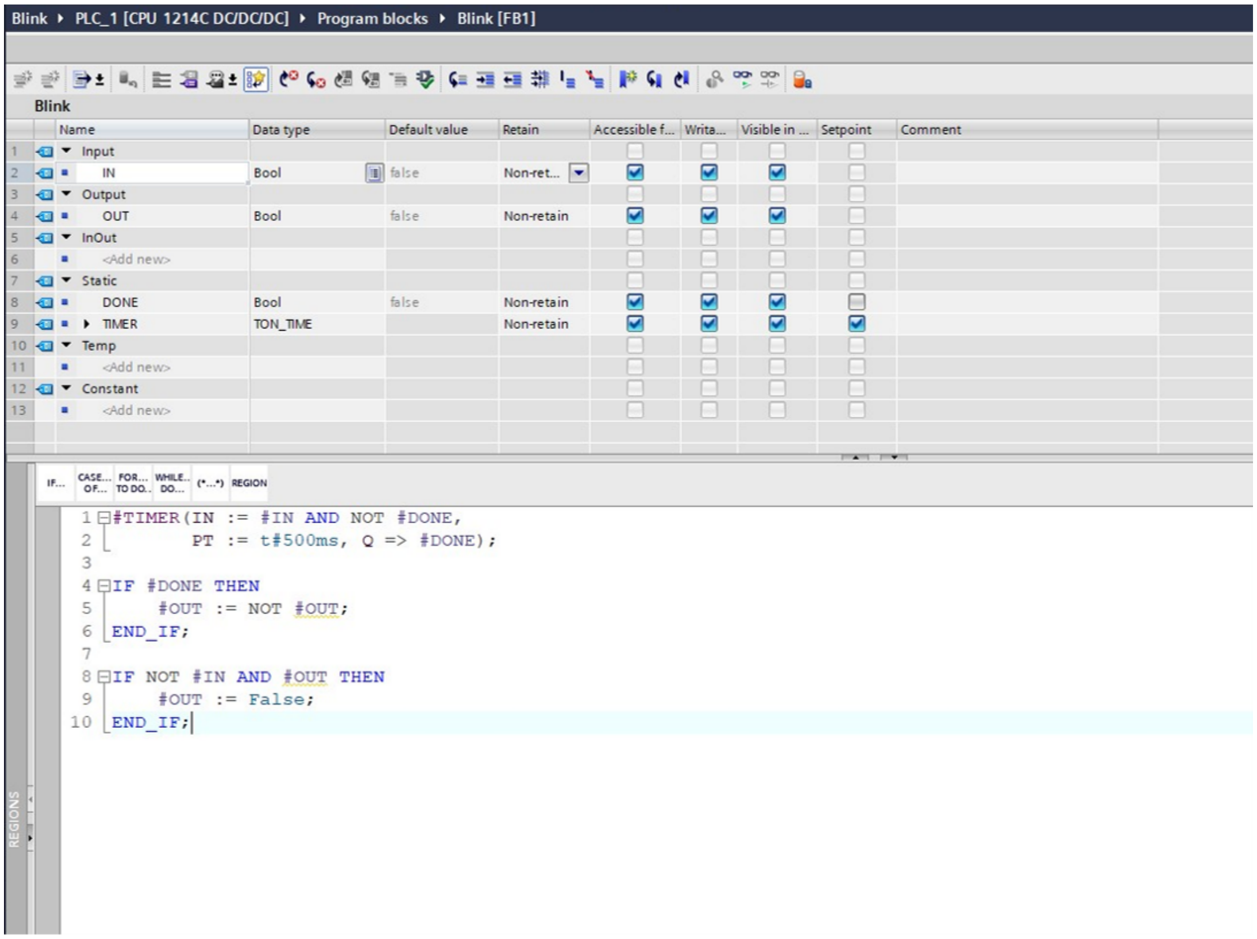
Addressing of global variables (tags): "<tag name>" (Tag name or data block name enclosed in double quotes)

Addressing of local variables: #<variable name> (Variable name preceded by "#" symbol)

Comments

Single line comment: `// comment`

Multi line comment and comments after end of ST line: `<statement>; (* comment *)`



Blink ▸ PLC_1 [CPU 1214C DC/DC/DC] ▸ Program blocks ▸ Blink [FB1]

Blink

	Name	Data type	Default value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	▾ Input				☐	☐	☐	☐	
2	IN	Bool	false	Non-ret...	✓	✓	✓	☐	
3	▾ Output				☐	☐	☐	☐	
4	OUT	Bool	false	Non-retain	✓	✓	✓	☐	
5	▾ InOut				☐	☐	☐	☐	
6	<Add new>				☐	☐	☐	☐	
7	▾ Static				☐	☐	☐	☐	
8	DONE	Bool	false	Non-retain	✓	✓	✓	☐	
9	▸ TIMER	TON_TIME		Non-retain	✓	✓	✓	✓	
10	▾ Temp				☐	☐	☐	☐	
11	<Add new>				☐	☐	☐	☐	
12	▾ Constant				☐	☐	☐	☐	
13	<Add new>				☐	☐	☐	☐	

```
#TIMER(IN := #IN AND NOT #DONE,
       PT := t#500ms, Q => #DONE);

IF #DONE THEN
    #OUT := NOT #OUT;
END_IF;

IF NOT #IN AND #OUT THEN
    #OUT := False;
END_IF;
```

Fig. 8.7: An example of a SCL code 2.

Operators: Standard operators in ST ordered after precedence:

Operation	Symbol	Precedence
Parentheses	`(expr)`	Highest
Function Evaluation	`MAX(A,B)`	
Negation Complement	`NOT`	
Exponentiation	`**`	
Multiply	`*`	
Divide	`/`	
Modulo	`MOD`	
Add	`+`	
Subtract	`-`	
Comparison	`<,>,<=,>=`	
Equality / Inequality	`= / <>`	
Boolean AND	`& AND`	
Boolean Exclusive OR	`XOR`	
Boolean OR	`OR`	Lowest

Note: `A = B` and `A := B` are NOT SAME!

= (equality operator) evaluates if the left and the right side is equal. If value of `A` is equal to value `B`. It returns `TRUE` if yes, else it returns `FALSE`.

`:=` denotes a statement. It is used for assigning value of right side to the left side. In this case, the value of B is assigned to A.

Combining operators:

```
IF (Input1) AND (Input2) OR (Input3) THEN
    Output1 := True;
END_IF
```

IF Statements: If statements are used for boolean queries.

```
IF [boolean expression] THEN
    <statement>;
ELSIF [boolean expression] THEN
    <statement>;
ELSE
    <statement>;
ENDIF;
```

CASE Statements: Case statements are one of the most important structuring methods to generate readable code. Case statements are typically used to program state machines.

```
TYPE
    Steps:(INIT:=0, START, RUN, END);
END _TYPE

VAR
    state: Steps; (*use of enumeration*)
END_VAR
```

(continues on next page)

(continued from previous page)

```
CASE state OF
    INIT: Instruction_A;
    START: Instruction_B;
    RUN: Instruction_C;
    END: Instruction_D;
END_CASE
```

FOR Loops: It is used to repeat code a specific number of times.

```
FOR count := initial_value TO final_value BY increment DO
    <statement>;
END_FOR;
```

WHILE Loops: It is used to repeat the loop as long as some conditions are TRUE. A WHILE loop will repeat as long as a boolean expression evaluates to TRUE.

```
WHILE [boolean_expression] DO
    <statement>;
END_WHILE;
```

REPEAT Loops: It works the opposite way of the WHILE loop. This loop will stop repeating when a boolean expression is TRUE.

```
REPEAT
    <statement>;
UNTIL [boolean_expression]
END_REPEAT;
```

8.3.2 Ladder Diagram (LAD)

Ladder diagrams are specialized schematics commonly used to document industrial control logic systems.

In terms of TIA Portal program for Siemens PLCs, SCL plays an important role in programming of `functions codes` and `function blocks`. The `Main [OB1]` block is preferred to be programmed using the Ladder Diagrams.

A program written in LAD consists of networks. The language is based on relay logic. The flow of the logic 'true' across a network is similar to the flow of electricity across a relay logic. One can think of the left rail as being positive, and the right rail as being negative. The logic 'true' flows from the left rail to the right rail.

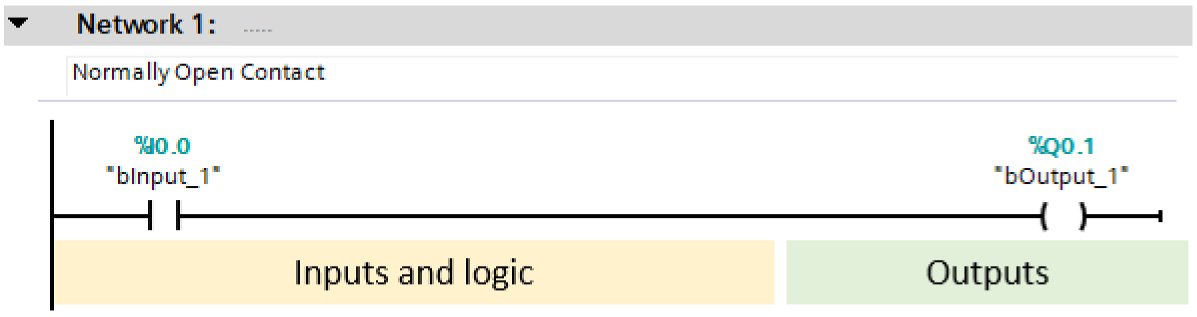


Fig. 8.8: An example of a network in LAD.

Various logical operations can be thus programmed in LAD. The following figure shows AND & OR logic after starting the PLC or simulated PLC.

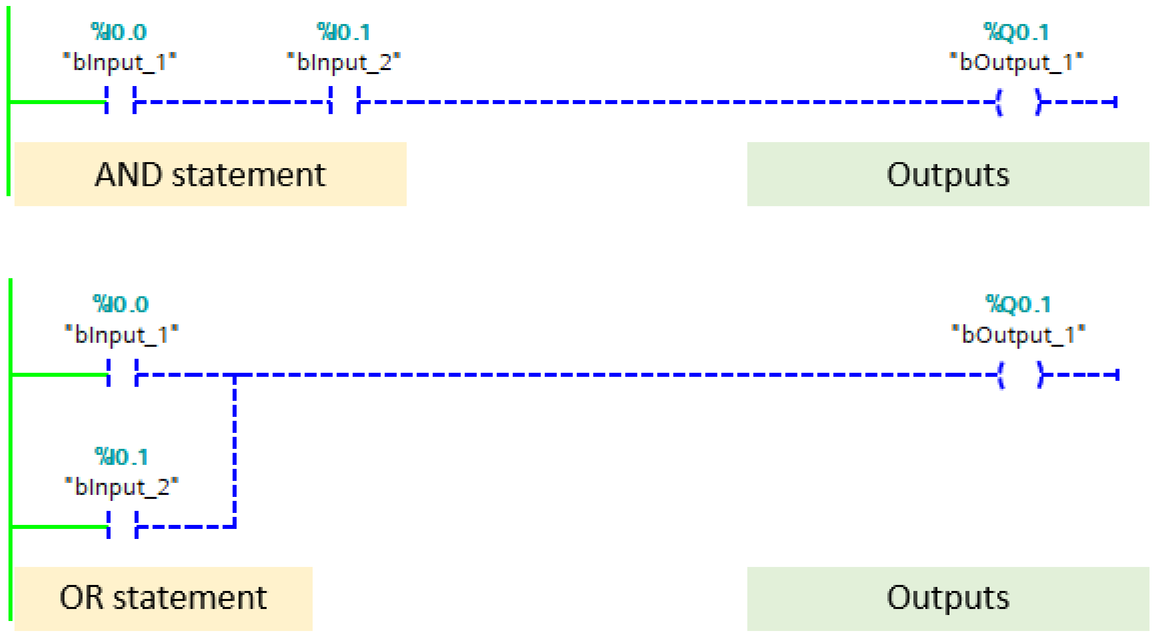


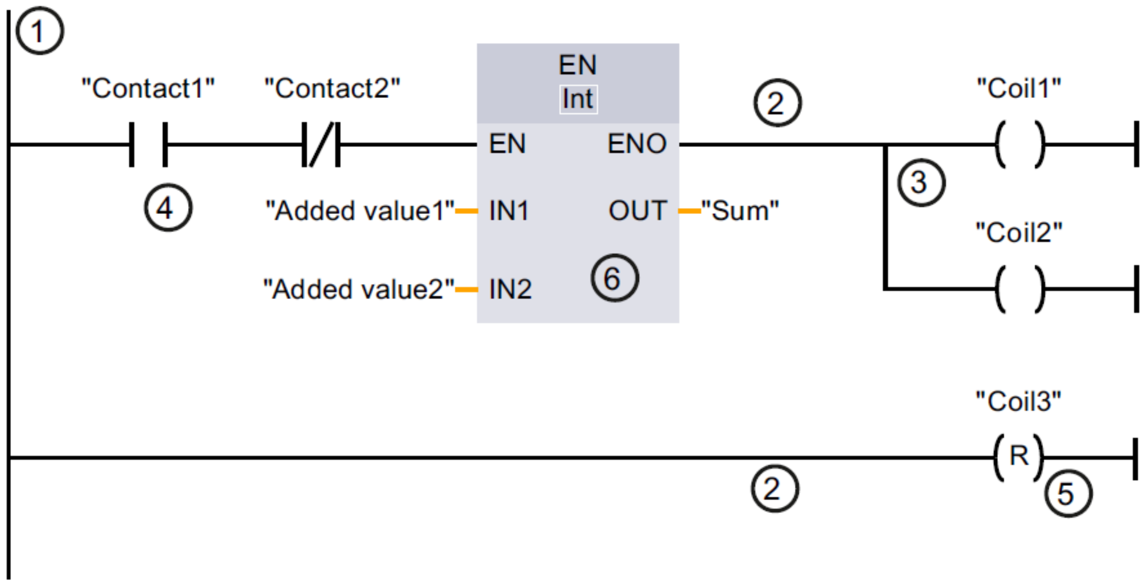
Fig. 8.9: An example of logical operations in LAD.

A general list of Bit Logic instructions used in LAD are mentioned below

Operation	Symbol
Normally Open Contact (Address)	`---| |---`
Normally Closed Contact (Address)	`---| / |---`
Save RLO into BR Memory	`---(SAVE)`
Bit Exclusive OR	`XOR`
Output Coil	`---( )`
Midline Output	`---( # )---`
Invert Power Flow	`---|NOT|---`
Insert branch	
Merge branch	

8.3.3 Function Block Diagram (FBD)

FBD is another graphical programming language. It uses Boolean algebra-based blocks to develop codes. A function block is depicted as a rectangular block. The inputs are on the left and outputs on the right side. It can have standard functions, such as logic gates, mathematical operations, counters, or user defined functions.



①	Power rail
②	Rung
③	Branch
④	Contact
⑤	Coil
⑥	Box

Fig. 8.10: An example of a network in LAD. (ref: SIEMENS TIA Portal STEP 7 Basic V10.5)

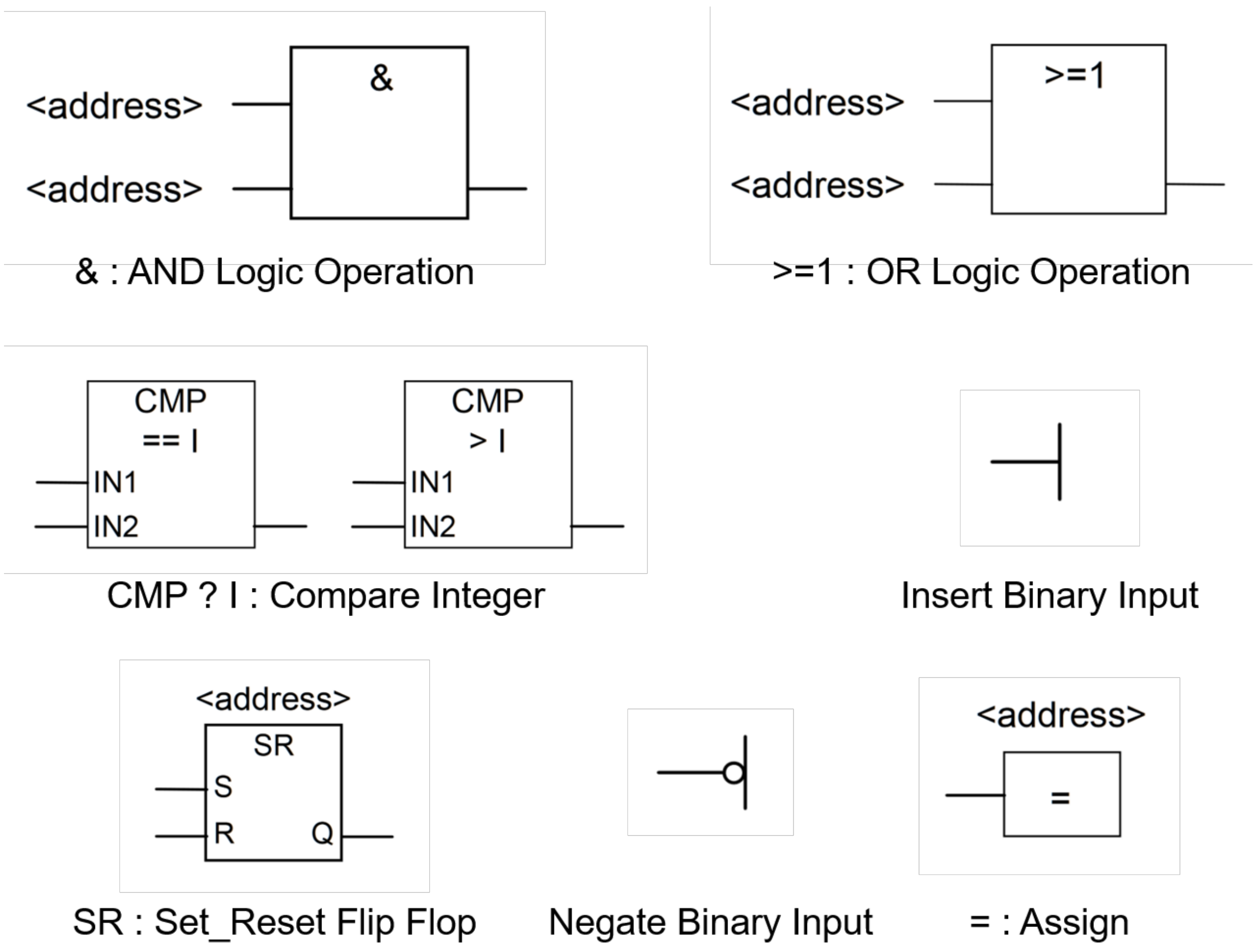


Fig. 8.11: Common FBD blocks (ref: SIEMENS Function Block Diagram (FBD) Reference Manual)

8.3.4 Additional languages

The Instruction List (IL) is very similar to the programming language Assembler and typically used to limit the computing time. In the example, the operand A is loaded in the working memory in the first line. Next, an AND operation with the negated operand B is conducted. Finally, the result is stored in the variable C. However, IL gets very long and confusing if complex tasks are programmed. As computing time usually does not play an important role today, high-level languages like Pascal, C or Structured Text (ST) are used typically.

The Sequential Function Chart (SFC) can only be used to program sequential control tasks in the form of step chains.

8.3.5 Examples

SCL

A complete example (programming of a traffic light) is shown below. First, an enumeration including all process states is defined. Next, all necessary variables are declared (a variable called `state` is defined using the new created data type `stateType`). Finally, the program sequence is established using a `CASE OF` statement.

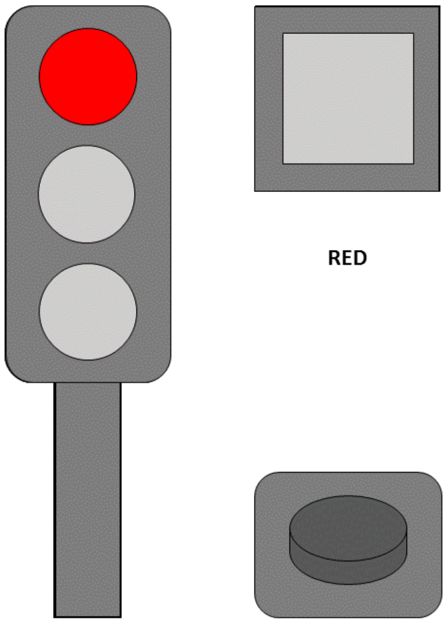
```
TYPE stateType :
    (RED:=1,REDYELLOW,GREEN,YELLOW);
END_TYPE

VAR
    state:stateType;
    button:BOOL;
    lred,lyellow,lgreen:BOOL;
    atime:TON;
END_VAR
```

```
CASE state OF
    stateType.RED:
                IF NOT lred THEN
                    lred:=TRUE;
                END_IF
                IF button THEN
                    state:=stateType.REDYELLOW;
                END_IF
    stateType.REDYELLOW:
                IF NOT lyellow THEN
                    lyellow:=TRUE;
                    atime(IN:=TRUE,PT:=#5s);
                END_IF
                IF atime.Q THEN
                    state:=stateType.GREEN;
                END_IF
    stateType.GREEN:
                IF NOT lgreen THEN
                    lgreen:=TRUE;
                    atime(IN:=TRUE,PT:=#40s);
                END_IF
                IF atime.Q THEN
                    state:=stateType.YELLOW;
```

(continues on next page)

(continued from previous page)

```
                END_IF
    stateType.YELLOW:
                IF NOT lyellow THEN
                    lyellow:=TRUE;
                    atime(IN:=TRUE,PT:=#5s);
                END_IF
                IF atime.Q THEN
                    state:=stateType.GREEN;
                END_IF
END_CASE
```



8.4 Sources

- https://www.plcacademy.com/structured-text-tutorial/
- Book Speicherprogrammierbare Steuerungen für die Fabrik- und Prozessautomation, Matthias Seitz, 2012*

CHAPTER

NINE

OVERVIEW ON HANDS-ON VIDEOS FOR E!COCKPIT

The following videos provide an overview on how to use e!Cockpit programming environment.

For general information about PLCs and PLC programming languages, please refer to the general PLC learning module.

For all videos, a German as well as an English version is provided. Both videos contain the same content.

Warning: This page only contains an extraction of existing learning viedos for e!Cockpit. You can find the full list of videos here:

https://youtube.com/playlist?list=PLfPCU8iUgXjoHI1_VpxFGmjBUgTbPJg_d (German)

and here:

https://youtube.com/playlist?list=PLfPCU8iUgXjoSaptfsNrxYoJtJCrznYvh (English)

9.1 Creating a connection to the PLC

9.1.1 Real PLC available

The following video shows you how to scan a network for existing PLCs.

https://youtu.be/5DtAisRMUJI

https://youtu.be/D6uInVqTLuQ

9.1.2 Usage of a virtual PLC

If you do not have a PLC, you can use the simulation functionalities of e!Cockpit. Therefore, create an empty project and select a PLC from the product catalogue on the right bottom of the e!Cockpit. For instance, you can select a Wago 750-8206.

When using a virtual PLC, you can do the same programming and visualization steps as with a real PLC.

9.2 Adding hardware modules

The following video shows you two options on how to add modules, such as digital inputs, to e!Cockpit. If you are using a real PLC, you can use both options (manual adding as well as automatic recognition). If you are using a simulated PLC, please use the manual adding option.

https://youtu.be/9a0fXNDTeoA

https://youtu.be/Iw5YHhHm9YM

9.3 Program Download and Boot Application

After programming, the program code must be uploaded to the PLC.

https://youtu.be/yV1l1FvjxI0

https://youtu.be/zihFQECdSbU

9.4 Variables

The following video shows how to declare and use variables in e!Cockpit.

9.4.1 Local variables

Local variables which are only valid in the associated program organization unit (POU).

https://youtu.be/fsm3wdhrvwQ

https://youtu.be/gAa9xLlN00Q

9.4.2 Global variables

Global variables are valid in all POUs. They are used for communication between different programs.

https://youtu.be/El1euulrWVw

https://youtu.be/Ac_8nN3LdVk

9.4.3 Retain and persistent variables

The following video shows how to keep the values of variables even if the controller is rebooted or a program is downloaded again.

https://youtu.be/IXIY07RsuTE

https://youtu.be/ZbP8Um7S2c8

9.5 Programs, functions and function blocks

The following video shows the differences between programs, functions and function blocks as well as their usage in e!Cockpit.

https://youtu.be/p0-t5P0_AeI

https://youtu.be/27-JI2aMs7Q

9.6 Own datatype structures

The following video shows how a datatype can be used to combine and structure variables.

https://youtu.be/TOwBdnSekEc

https://youtu.be/9EM4bdiYN5s

9.7 Visualization

e!Cockpit offer built-in visualization options.

https://youtu.be/RDDQy3d8zqc

https://youtu.be/PrsaxV5x6U8

CHAPTER

TEN

SIEMENS NX MCD

In this module, Siemens NX MCD is introduced and the most common functions that are used in constructing kinematic models are presented.

10.1 Learning Outcome

- You will get familiar with virtual commissioning.
- You will get to know the basics of Siemens NX MCD.
- You will understand and be able to create physics-based models.

10.2 Virtual Commissioning

Virtual commissioning is the process of testing and debugging control code on virtual models, possibly before the machine/ components materialize. It allows for early validation of code and decreases real commissioning time.

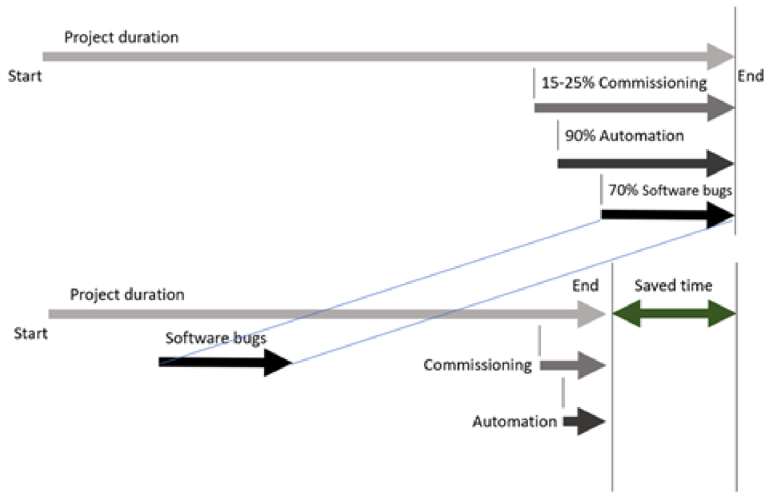


Fig. 10.1: Influence of control software debugging on project time with (figure below) and without (figure above) virtual commissioning

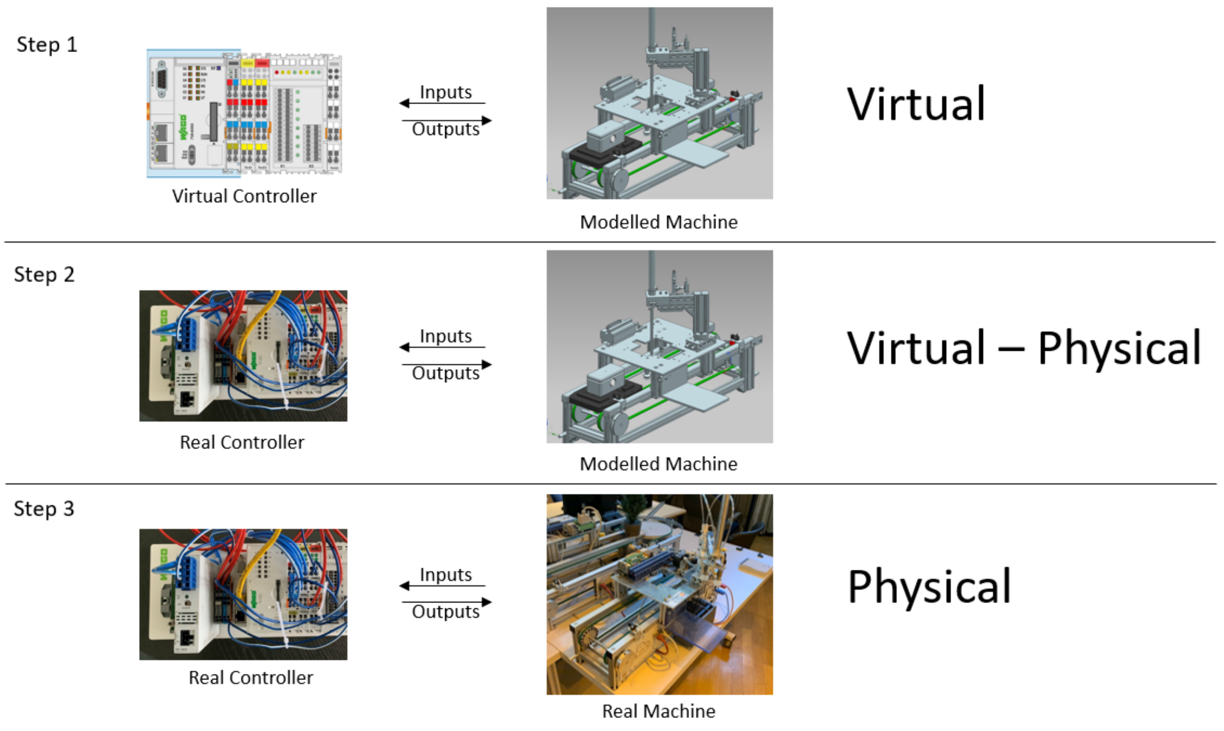


Fig. 10.2: Typical steps of virtual commissioning

Virtual commissioning can start during the development stages of the product/ asset. At stages where the material objects are not yet available, virtual commissioning allows for early detection of bugs in the PLC code and design flaws in the asset.

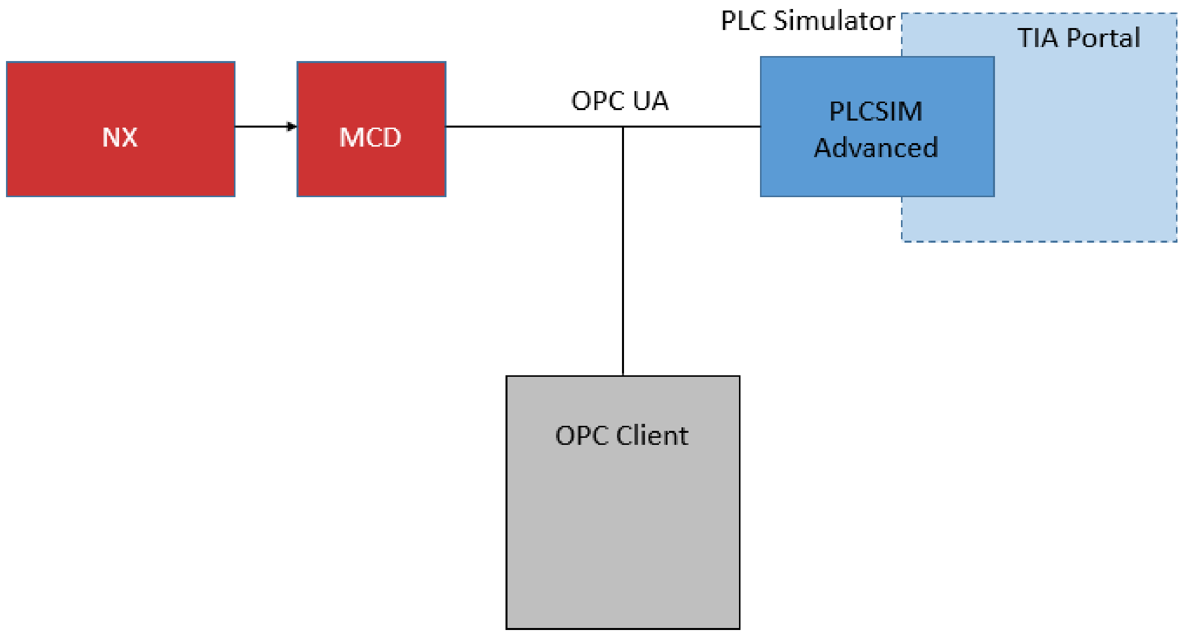
10.3 Mechatronics Concept Designer

10.3.1 Siemens NX General Actions

NX is an advanced CAD/CAM/CAE software. The following are some common actions used when working with a CAD assembly in NX.

Table 10.1: General NX Commands

Action	How to do it
Zoom in/out	Mouse wheel
Move the view left/right/up/down	Click and hold both the mouse wheel + the right mouse button and move the mouse
Rotate the view	Click and hold the mouse wheel and move the mouse
Move/Rotate an object relative to other objects/global coordinate system	Assemblies tab > Move Component > Select Components > Specify Orientation with arrows



10.3.2 Modelling with MCD

Mechatronics Concept Designer is an application inside Siemens NX that allows the creation of physics-based models and simulations based on CAD files. The goal of the following sections is to guide you through the basics of creating a physics-based model and influencing it from outside the simulation using external signals. To navigate to MCD, open a part or an assembly in Siemens NX. In the Application tab, click on more and choose Mechatronics Concept Designer.

Rigid Bodies

Solids (i.e., CAD models with no assigned physical properties in MCD) are only there to visualize the assembly and do not move during the simulation. Once a part gets assigned as a rigid body, it starts participating in the simulation. As a rigid body, the part is changed from just a CAD model to a part with physical characteristics. Rigid bodies have weight, center of mass and inertia. These parameters influence the dynamics of the body in the simulation. It is recommended to assign a material to the solid body, so that a realistic weight can be calculated.

Rigid bodies react to the acceleration due to gravity, which can be defined along any axis in MCD. Usually, acceleration due to gravity is defined in the negative z or y directions. The direction of gravitational acceleration can be changed in `File` -> `Preferences` -> `Mechatronics Concept Designer` -> `General` -> `Acceleration due to Gravity`.

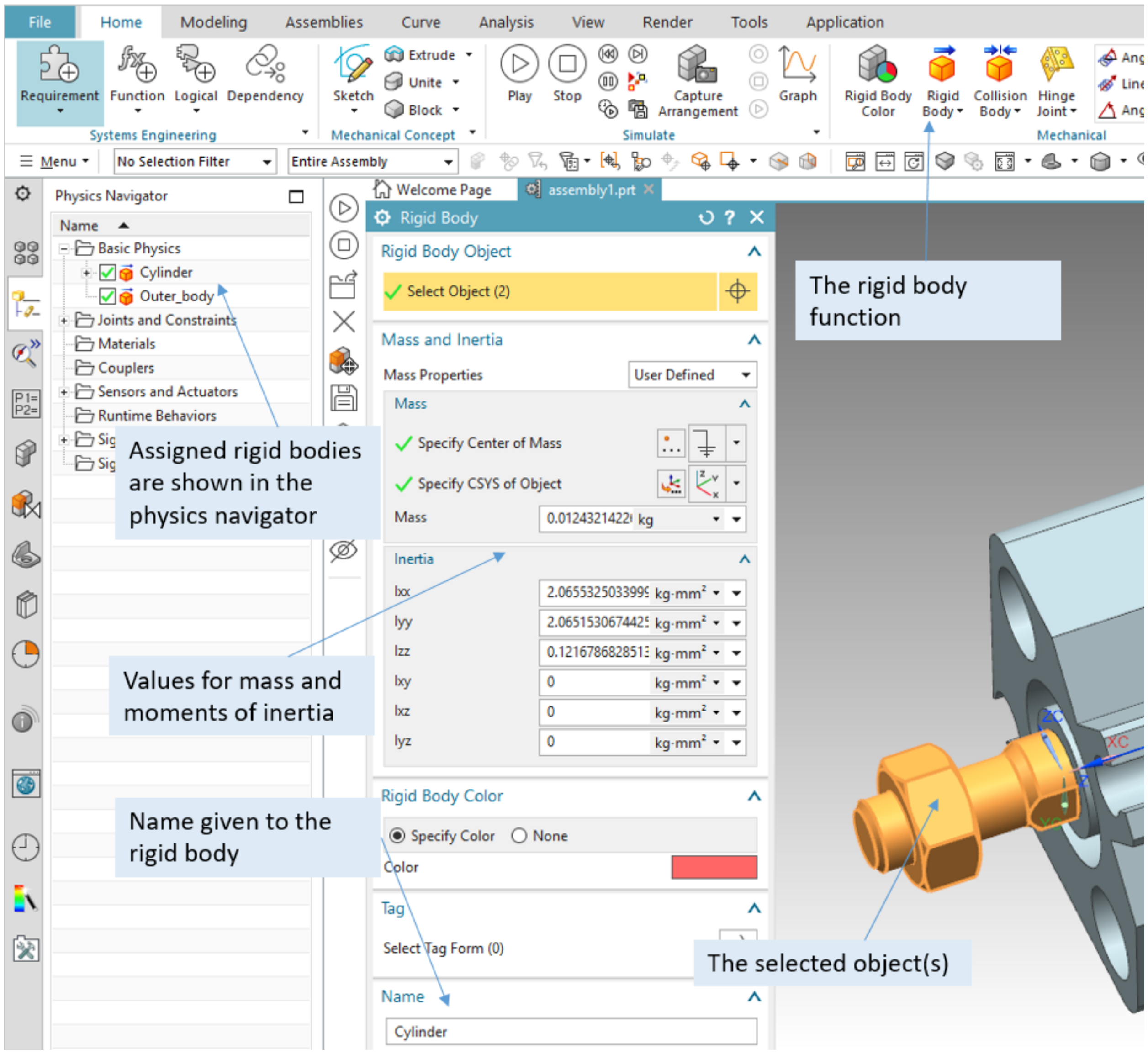
Collision Bodies

Rigid bodies without collision bodies do not collide when they come in contact. Collision is only possible when both objects have collision bodies. Without collision bodies, objects will go right through each other. An object with a collision body, however, will collide with another object that also has a collision body.

Collision is a property which is independent of whether the part is a rigid body or not i.e., a collision object that has no assigned rigid body status will still collide with other objects that have a collision body.

The collision shape of a collision body can be defined in many ways. It is best to choose simple shapes (box, cylinder, etc.) as a collision shape.

Collision mesh



The rigid body function

Assigned rigid bodies are shown in the physics navigator

Values for mass and moments of inertia

Name given to the rigid body

The selected object(s)

Rigid Body

Rigid Body Object
Select Object (2)

Mass and Inertia
Mass Properties: User Defined

Mass
Specify Center of Mass
Specify CSYS of Object
Mass: 0.01243214228 kg

Inertia
Ixx: 2.065532503399 kg-mm²
Iyy: 2.0651530674425 kg-mm²
Izz: 0.121678682851 kg-mm²
Ixy: 0 kg-mm²
Ixz: 0 kg-mm²
Iyz: 0 kg-mm²

Rigid Body Color
[✓ Specify Color] None
Color: 🟥

Tag
Select Tag Form (0)

Name
Cylinder

Fig. 10.3: Assigning a rigid body to a part. One rigid body can consist of multiple parts connected to each other that will always undergo the same motion together.

The option "mesh" offers the highest complexity. It allows for a more realistic simulation of the real object, but requires a high simulation performance. The `Convex Factor` sets the resolution of the mesh. The higher the factor, the more detailed the collision mesh. Collision shapes that consist of more than 300 triangles will automatically set a Siemens NX warning pointing to potential performance losses.

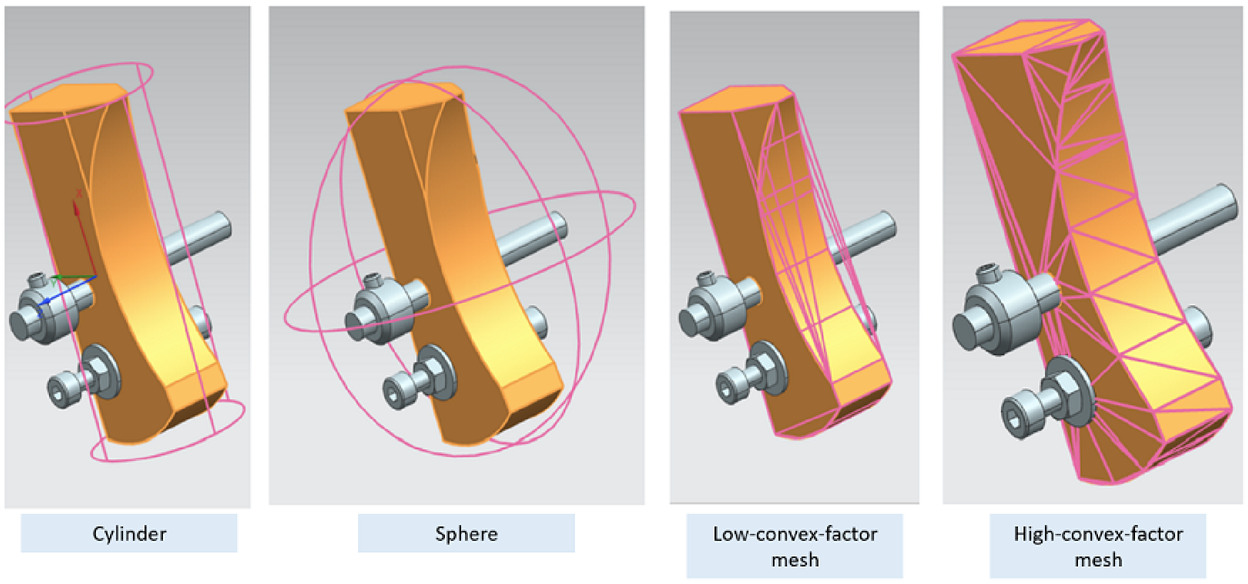


Fig. 10.4: Different ways to set a collision body; the pink area is the active collision area that will collide with other collision objects.

A decision must be made whether an added complexity to the collision shape serves the purpose of the simulation.

Category

The category is a number that can be assigned to a collision body to specify collision possibilities with other collision bodies. Collision bodies only collide within the same categories and with collision bodies in the category 0. Category 0 is an exception; bodies in the category 0 collide with all other collision bodies, regardless of the object's category. Categories can be used to simulate inductive sensors.

Collision material

The choice of collision material influences the physical coefficients dynamic friction, static friction, rolling friction, and restitution. It is recommended to enter accurate friction values for all parts that will take part in collisions to reflect reality.

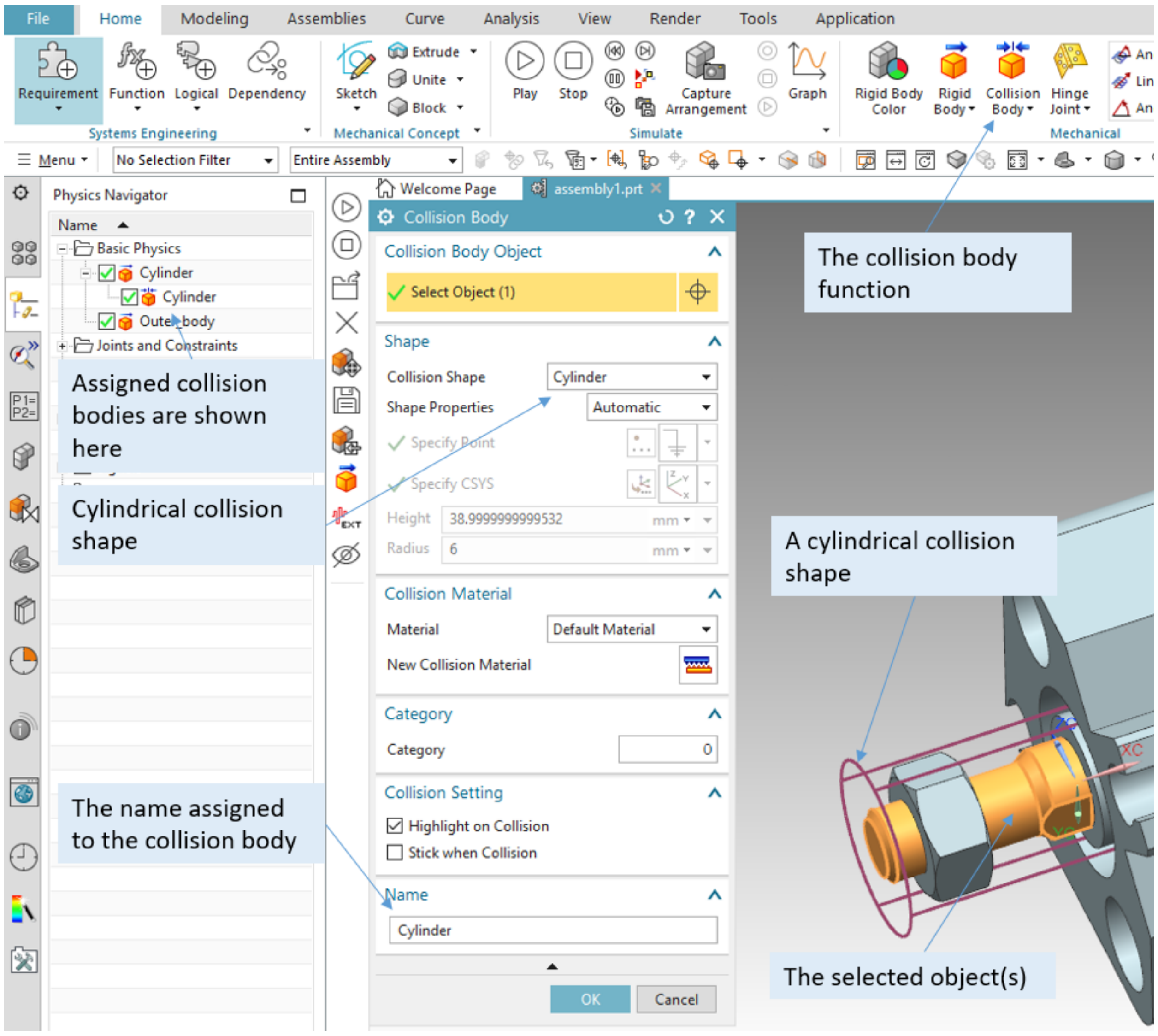
In order to create a custom material in NX MCD: Go to the `Physics Navigator`, right click on `Materials` and select `Create Physics` -> `Collision Material`.

Prevent collision

The prevent collision function allows two collision objects to not collide with one another if they come in contact.

Transport surface

Transport surfaces are used to simulate conveyor belts. Siemens NX MCD provides a function that can define any surface in an assembly as a transport surface.



Collision Body

Collision Body Object

Select Object (1)

Shape

Collision Shape	Cylinder
Shape Properties	Automatic
Specify Point	
Specify CSYS	
Height	38.9999999999532 mm
Radius	6 mm

Collision Material

Material	Default Material
New Collision Material	

Category

Category	0

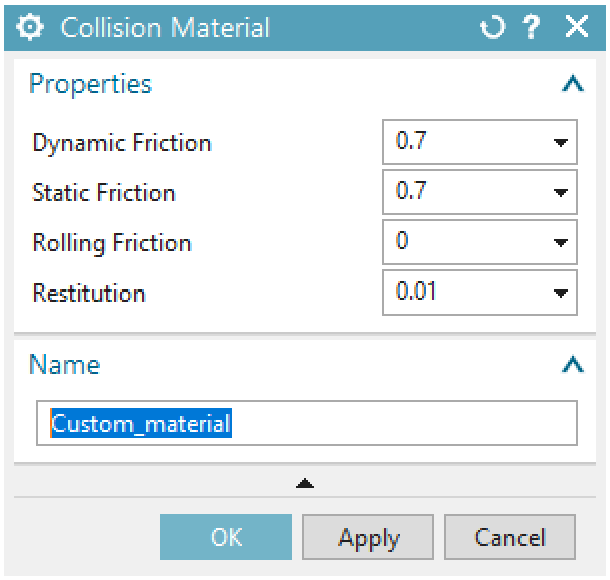
Collision Setting

☑ Highlight on Collision
☐ Stick when Collision

Name

Cylinder

Fig. 10.5: Assigning a collision body



Collision Material

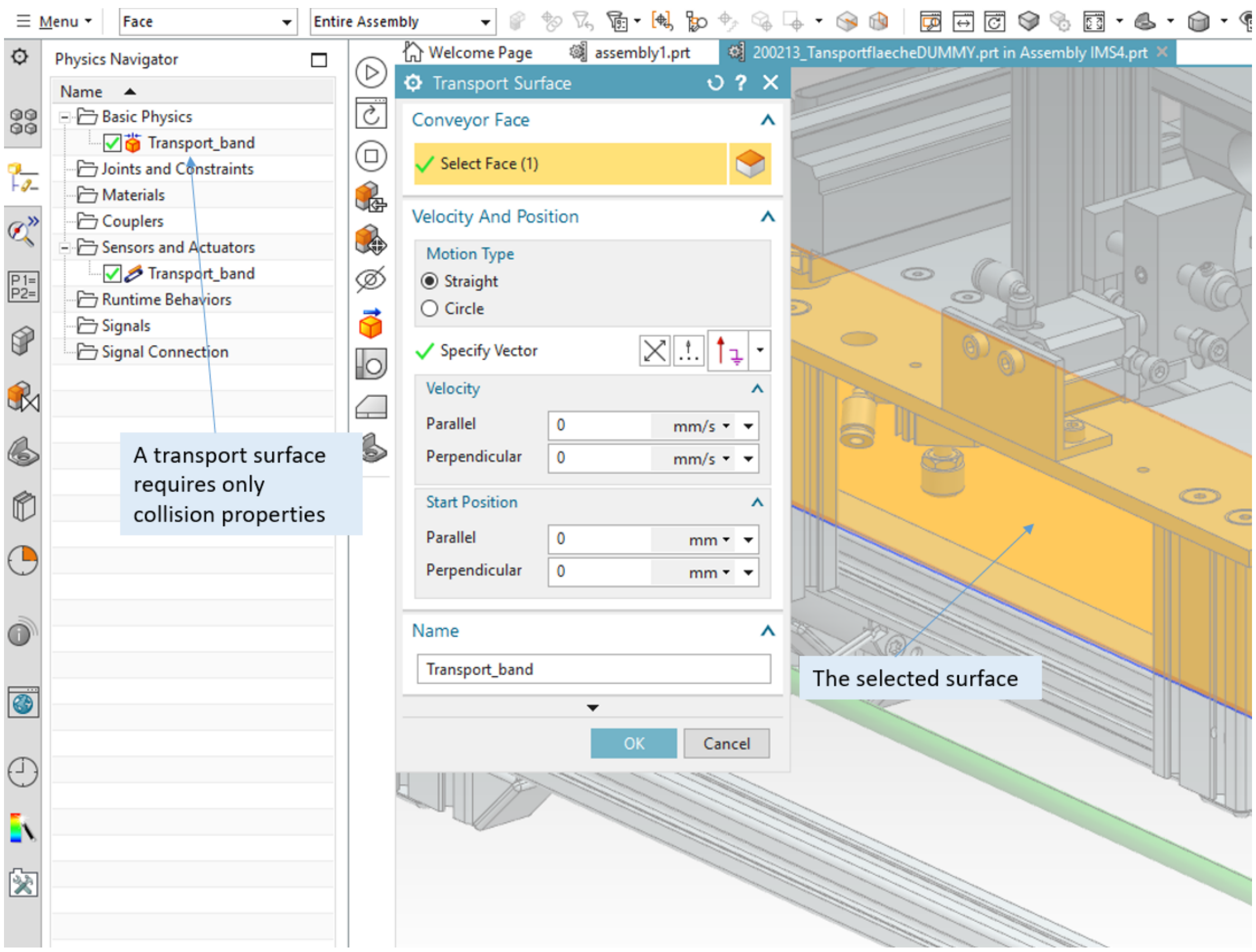
Properties

Dynamic Friction	0.7
Static Friction	0.7
Rolling Friction	0
Restitution	0.01

Name

Custom_material

Fig. 10.6: Creating a collision material



Transport Surface

Conveyor Face

Select Face (1)

Velocity And Position

Motion Type

(● Straight) ○ Circle

Specify Vector

Velocity

Parallel	0	mm/s
Perpendicular	0	mm/s

Start Position

Parallel	0	mm
Perpendicular	0	mm

Name

Transport_band

Fig. 10.7: Assigning transport area properties to a surface

Collision Sensors

One of the sensor functions available in Siemens NX MCD is a function that turns objects into collision sensors. When such an object collides with another object (the second object must have collision properties), a boolean signal is set to true. Collision sensors are used to simulate different types of real life sensors (e.g., optical sensors, inductive sensors, etc.). Collision sensors are usually helping elements and are not displayed (blinded out) during the final simulation.

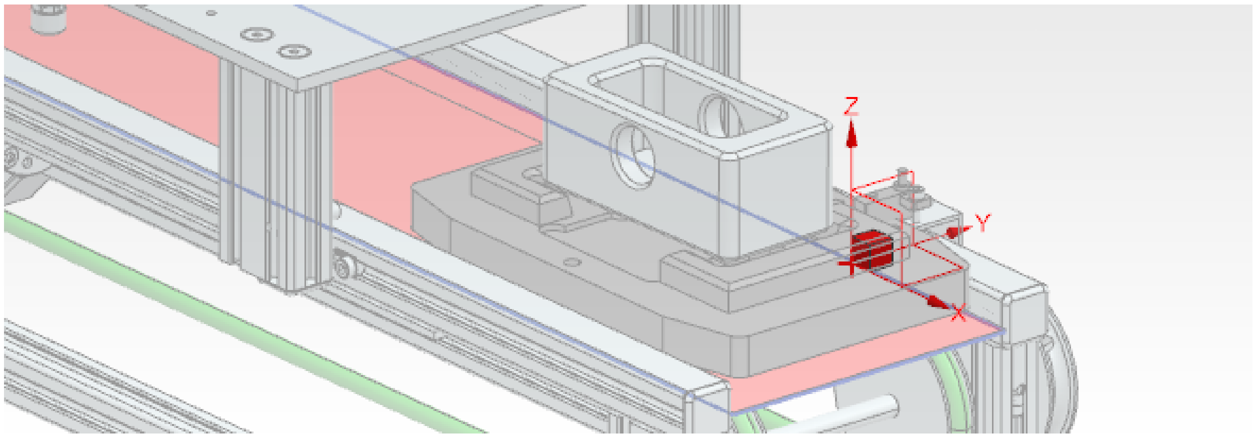


Fig. 10.8: A collision sensor used to simulate the function of an inductive sensor in the assembly. The sensor detects if a part is at the beginning of the conveyor belt.

Joints

Rigid bodies can be fixed, positioned in space or connected to each other by joints of different degrees of freedom. These connections are independent of the assembly constraints from the design application. The joints are available in different degrees of freedom: Hinge (swivel joint f=1), sliding joint (f=1), and cylindrical joint (combined sliding joint and hinge (f=2)).

The following figure shows a sliding joint that is assigned to a rigid body. The rigid body has only one degree of freedom and can be moved only along the axis specified for the sliding joint. Moreover, upper and lower limits are defined along the specified axis which the body cannot exceed during motion.

The following figure shows a hinge joint assigned to a part within an assembly. For a hinge joint (and a sliding joint), an attachment object and a base object should be provided. A base object can be ignored if the base of rotation (or sliding) is not movable in the simulation. In case the base is movable, it must be specified.

Position Controls

The positions of the joints created earlier can be controlled with objects called `Position Controls`, which can be found in the `Electrical` tab. The following figure shows the creation of a position control object to control a sliding joint object.

Position controls can also be created to control the angle of a hinge joint. Position control objects have two important variables: position and speed. Both variables can be manipulated during the simulation using operations and external signals.

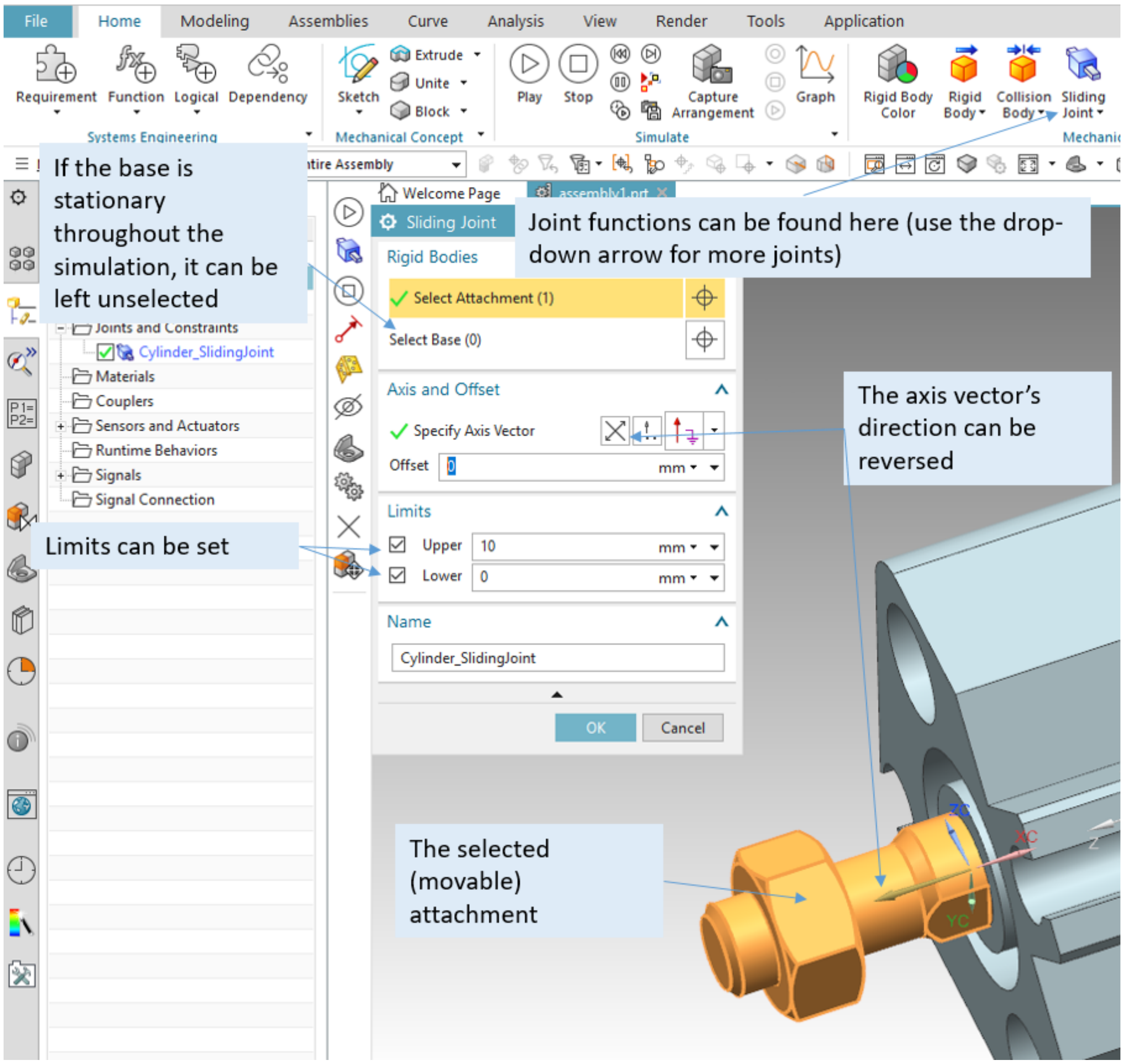
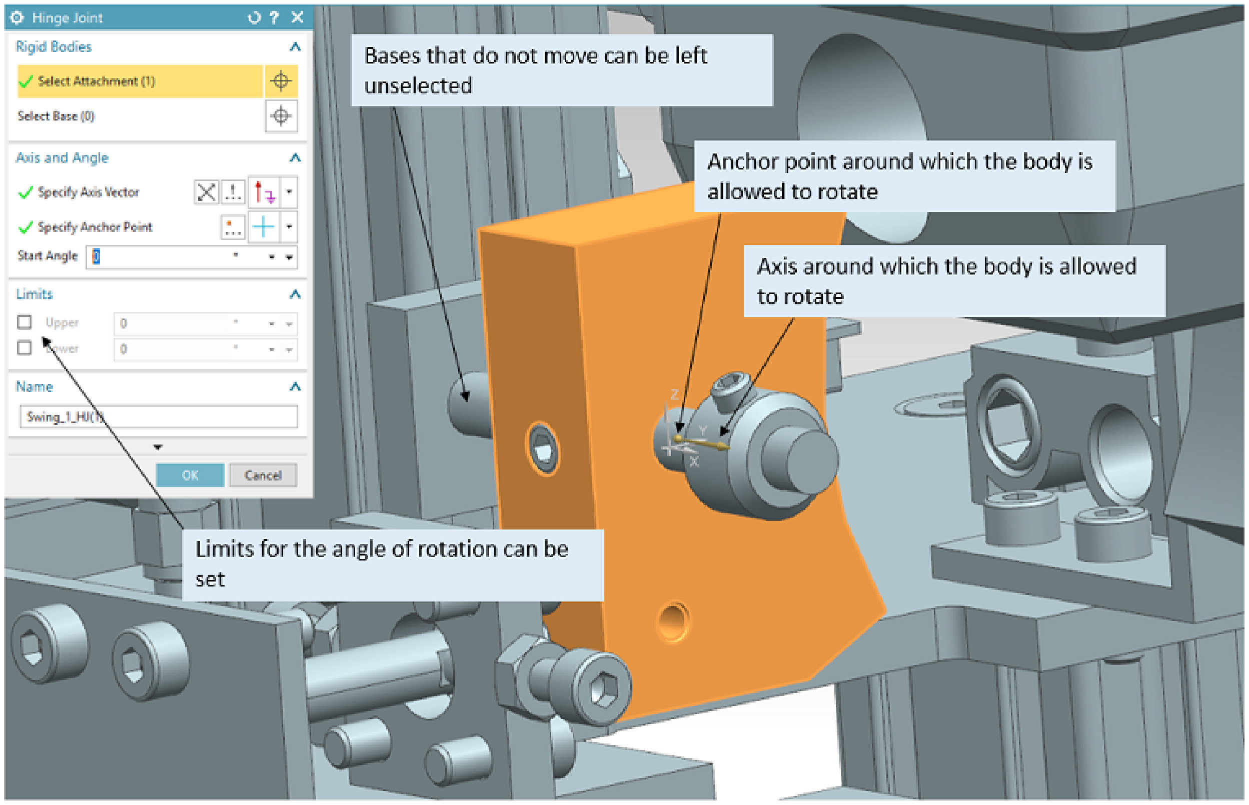


Fig. 10.9: Assigning a sliding joint to an object



Start Angle 0

Upper 0

Lower 0

Name Swing_1_HJ(1)

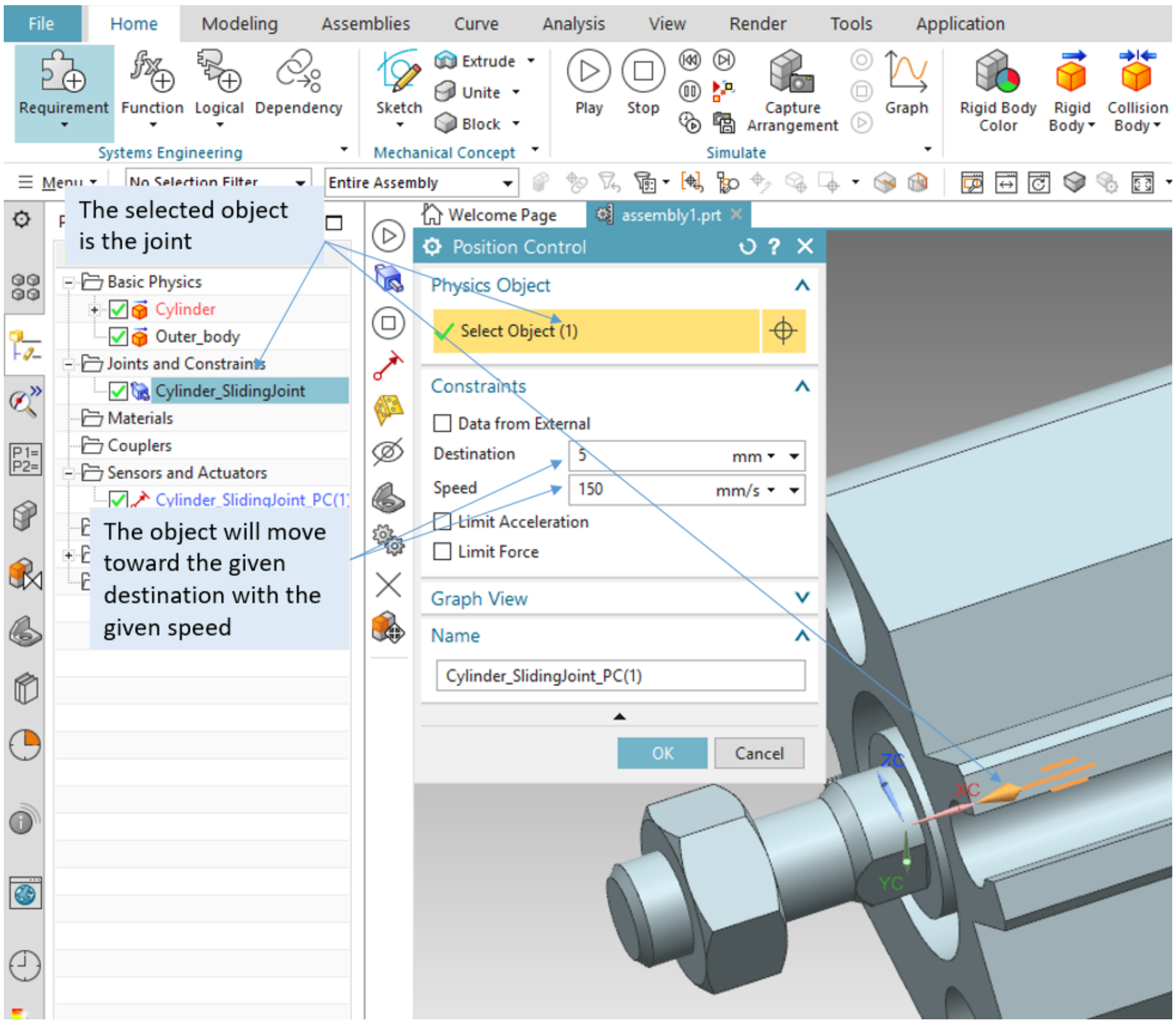
Bases that do not move can be left unselected

Anchor point around which the body is allowed to rotate

Axis around which the body is allowed to rotate

Limits for the angle of rotation can be set

Fig. 10.10: Assigning a hinge joint to an object within an assembly



The selected object is the joint

The object will move toward the given destination with the given speed

Position Control

Physics Object

Select Object (1)

Constraints

Data from External

Destination 5 mm

Speed 150 mm/s

Limit Acceleration

Limit Force

Graph View

Name

Cylinder_SlidingJoint_PC(1)

Fig. 10.11: Creating a position control to control the position of an object along its sliding joint

Signals

Signals are objects that allow the communication to programs outside of the simulation. A signal in MCD can have the type bool, int, or double. A signal can be an input signal or an output signal, allowing a two-way communication from and out of MCD.

Note: When communicating with a PLC, an output signal from the PLC's perspective (for example, an motor_on signal) is an input signal from MCD's perspective and should be defined in MCD as an input signal.

Signals can be linked to runtime parameters, in case of a collision sensor for example, where the collision sensor's value (true/false) is the runtime parameter. Signals can also trigger action in the simulation, e.g. setting a motor to turn on. When interacting with a PLC, an MCD signal should be created for each PLC input and output variable. In order to allow auto mapping procedures and prevent confusion, the names of the signals both in the PLC program and in MCD should be identical.

The following figure shows an input signal (input into MCD, output from PLC) that is meant to control the cylinder's position.

Symbol Tables are tables that show related signals together. They are a way to not lose track of signals.

10.3.3 OPC UA and Signal Mapping

To establish a communication over OPC UA, click on `External Signal Configuration` from the `Automation` tab. The following figure shows the configuration of an OPC-UA communication. Two boolean variables are selected which are relevant to the given simulation.

Note: To be able to communicate to a simulated PLC over OPC UA, make sure that OPC UA is configured in the PLC. (See section on TIA Portal basics)

Now that the external signals are imported, it is time to map them to the signals that were created inside MCD. The following figure shows auto mapping of 3 inputs and 2 outputs (from MCD's viewpoint).

10.3.4 Operations

Operations are one way to allow external or internal signals to manipulate the simulation in real time.

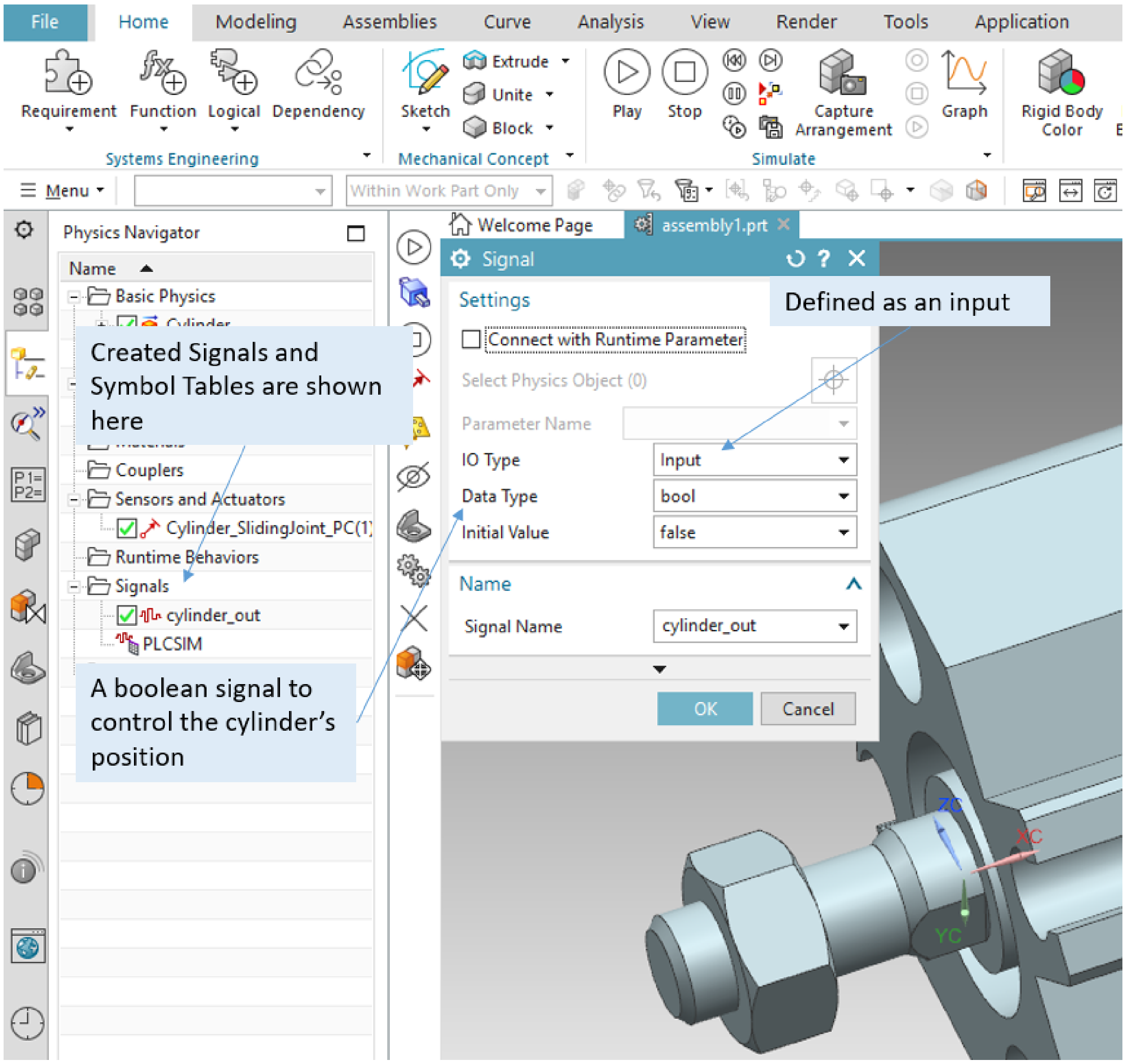
The following figure shows an operation that influences the value of the position variable of a position control object. The shown operation extends a cylinder. The condition object is an MCD signal that is mapped to an external signal. Thus, this cylinder can be controlled from outside the simulation while the simulation is running.

10.3.5 A Note on Assembly Hierarchy

The following figure shows a robotic arm in an assembly. The robotic arm itself (`UR3_step`) is an assembly that is made of single parts. The robotic arm is part of a bigger assembly (called `assembly1`). In this example, `assembly1` is the main assembly (the biggest assembly) and the robotic arm is a sub-assembly inside that main assembly.

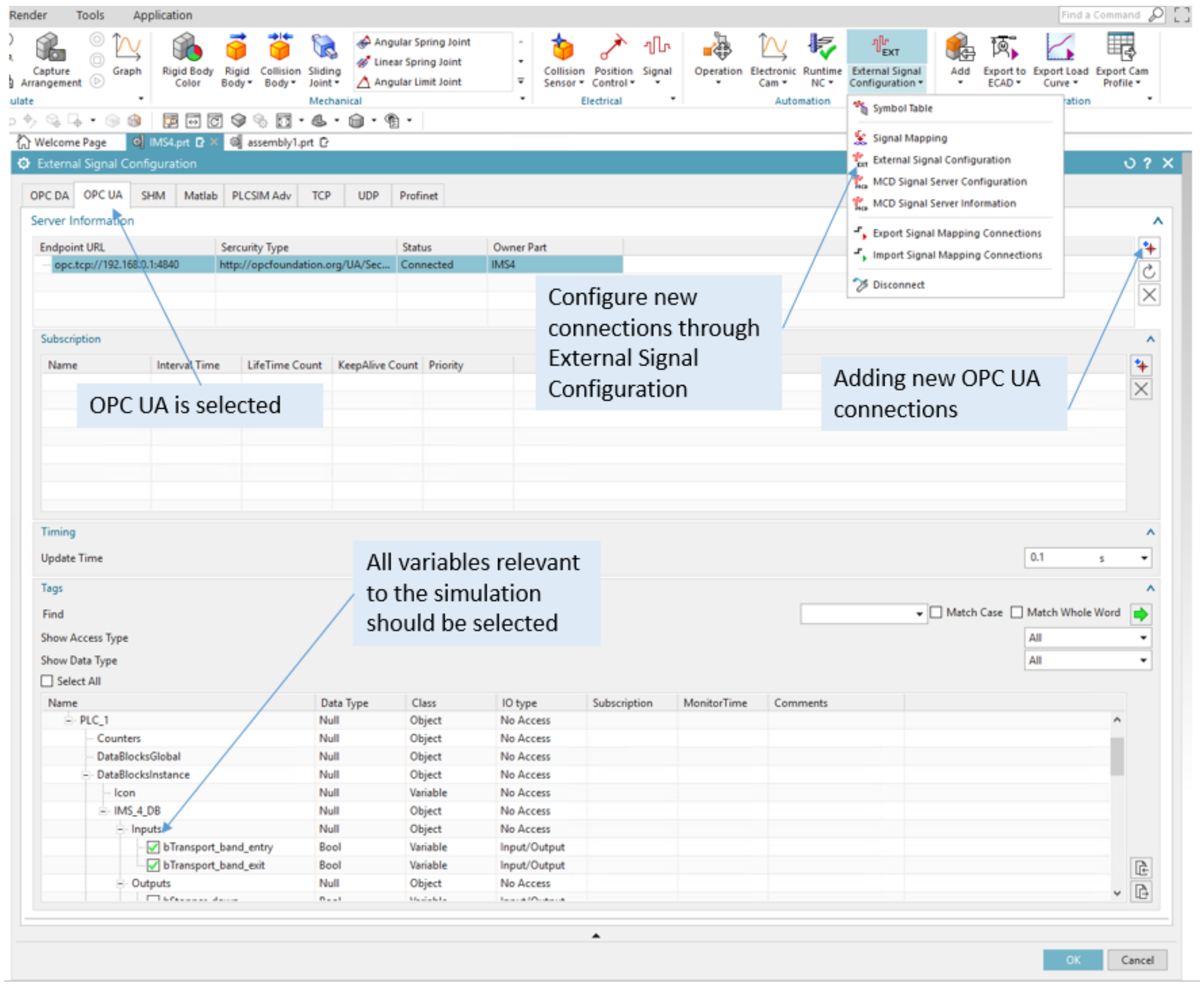
Assemblies can thus include single parts or other sub-assemblies.

By double clicking on a part/assembly, MCD focuses on that selected part/assembly. Simulations only run with the selected part/assembly. To run a simulation including all parts, the main assembly has to be selected. To run a simulation for only one part/assembly, that part/assembly has to be selected.



IO Type: Input
Data Type: bool
Initial Value: false
Signal Name: cylinder_out

Fig. 10.12: A boolean input signal



Configure new connections through External Signal Configuration

OPC UA is selected

Adding new OPC UA connections

All variables relevant to the simulation should be selected

Endpoint URL	Sercurity Type	Status	Owner Part
opc.tcp://192.168.0.1:4840	http://opcfoundation.org/UA/Sec...	Connected	IMS4

Name	Interval Time	LifeTime Count	KeepAlive Count	Priority

Update Time: 0.1 s

Find: ☐ Match Case ☐ Match Whole Word

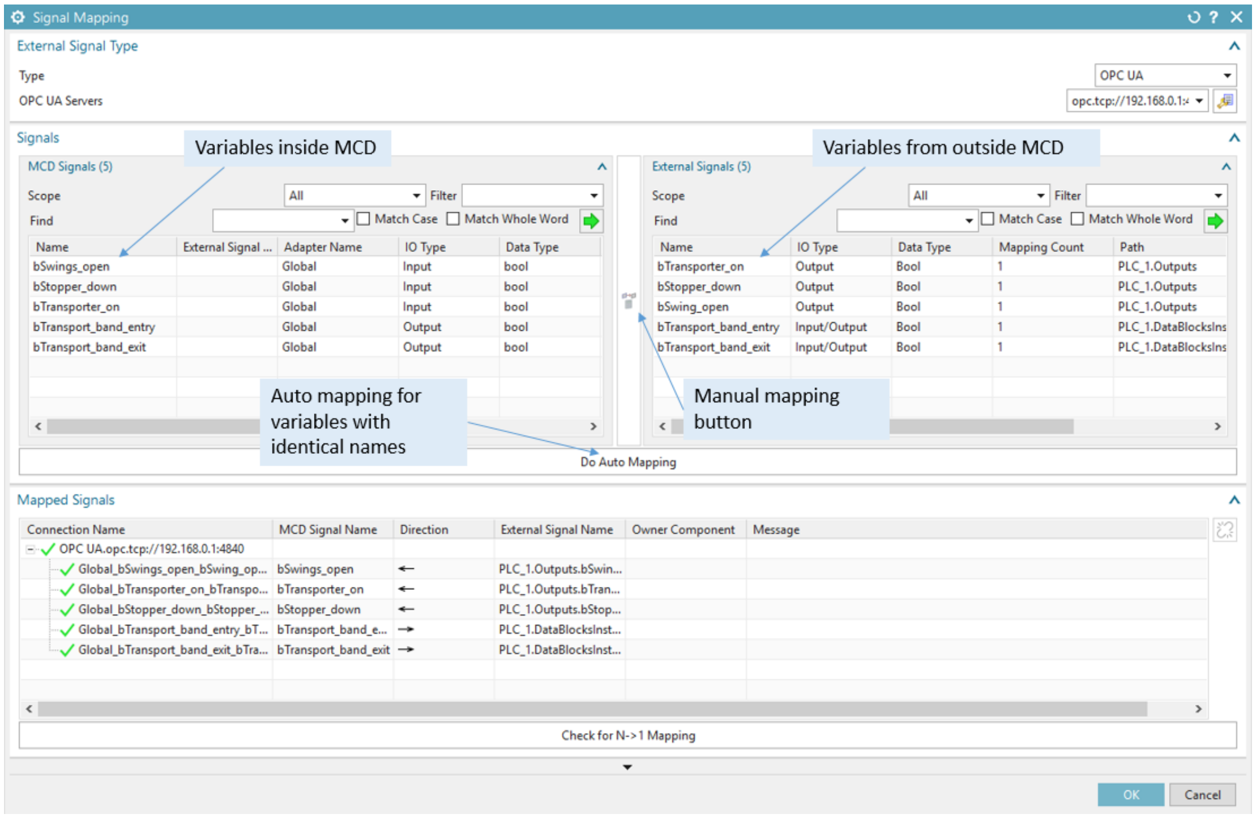
Show Access Type: All

Show Data Type: All

☐ Select All

Name	Data Type	Class	IO type	Subscription	MonitorTime	Comments
PLC_1	Null	Object	No Access			
Counters	Null	Object	No Access			
DataBlocksGlobal	Null	Object	No Access			
DataBlocksInstance	Null	Object	No Access			
Icon	Null	Variable	No Access			
IMS_4_DB	Null	Object	No Access			
Inputs	Null	Object	No Access			
✓ bTransport_band_entry	Bool	Variable	Input/Output			
✓ bTransport_band_exit	Bool	Variable	Input/Output			
Outputs	Null	Object	No Access			

Fig. 10.13: External signal configuration


Variables inside MCD
Variables from outside MCD
Auto mapping for variables with identical names
Manual mapping button

Signal Mapping

External Signal Type

Type: OPC UA

OPC UA Servers: opc.tcp://192.168.0.1:

Signals

MCD Signals (5)

Scope: All · Filter · Find · Match Case · Match Whole Word

Name	External Signal ...	Adapter Name	IO Type	Data Type
bSwings_open		Global	Input	bool
bStopper_down		Global	Input	bool
bTransporter_on		Global	Input	bool
bTransport_band_entry		Global	Output	bool
bTransport_band_exit		Global	Output	bool

External Signals (5)

Scope: All · Filter · Find · Match Case · Match Whole Word

Name	IO Type	Data Type	Mapping Count	Path
bTransporter_on	Output	Bool	1	PLC_1.Outputs
bStopper_down	Output	Bool	1	PLC_1.Outputs
bSwing_open	Output	Bool	1	PLC_1.Outputs
bTransport_band_entry	Input/Output	Bool	1	PLC_1.DataBlocksIns
bTransport_band_exit	Input/Output	Bool	1	PLC_1.DataBlocksIns

Do Auto Mapping

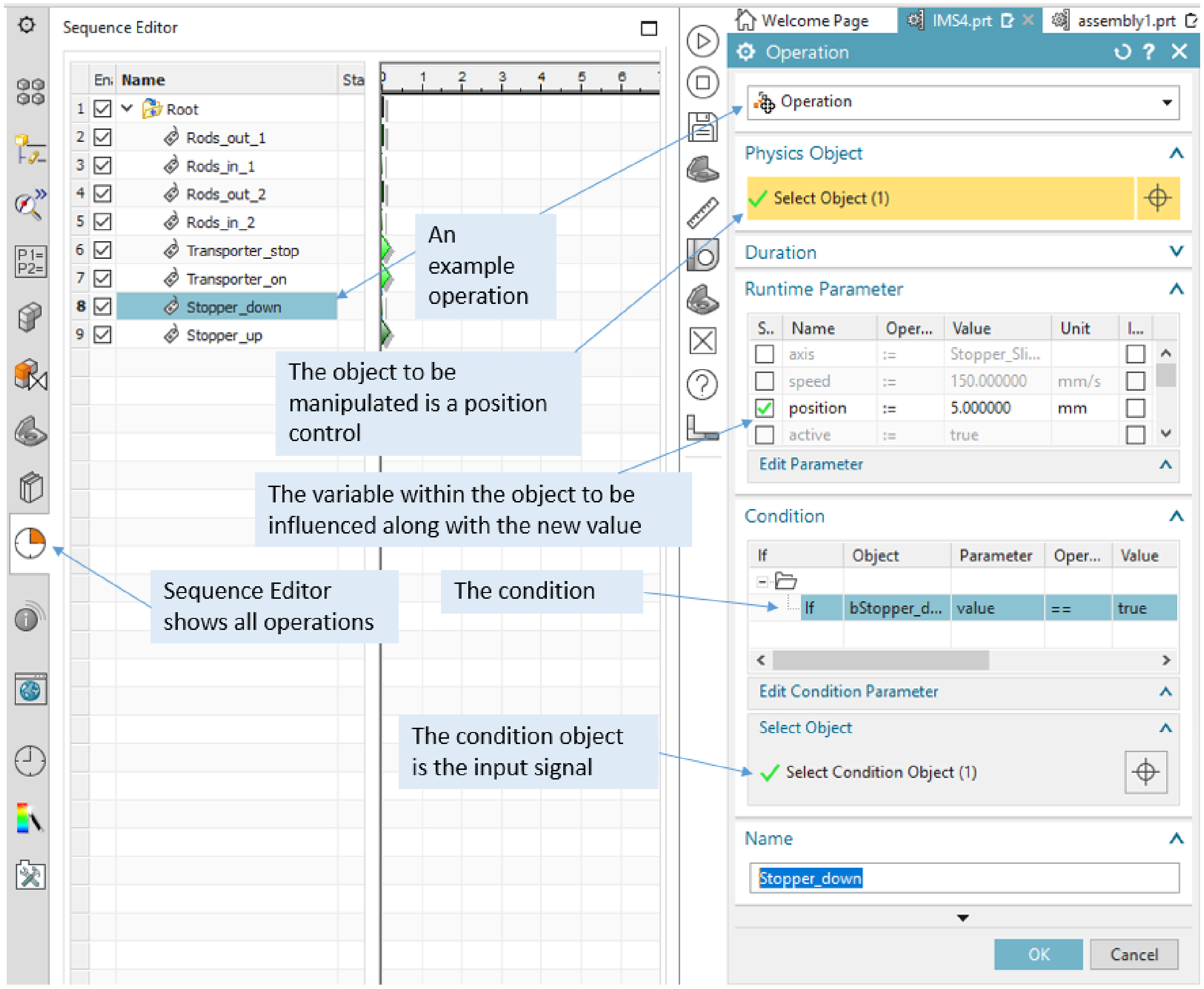
Mapped Signals

Connection Name	MCD Signal Name	Direction	External Signal Name	Owner Component	Message
OPC UA.opc.tcp://192.168.0.1:4840					
✓ Global_bSwings_open_bSwing_op...	bSwings_open	←	PLC_1.Outputs.bSwin...		
✓ Global_bTransporter_on_bTranspo...	bTransporter_on	←	PLC_1.Outputs.bTran...		
✓ Global_bStopper_down_bStopper_...	bStopper_down	←	PLC_1.Outputs.bStop...		
✓ Global_bTransport_band_entry_bT...	bTransport_band_e...	→	PLC_1.DataBlocksInst...		
✓ Global_bTransport_band_exit_bTra...	bTransport_band_exit	→	PLC_1.DataBlocksInst...		

Check for N->1 Mapping

OK · Cancel

Fig. 10.14: Signal mapping


Sequence Editor
An example operation
The object to be manipulated is a position control
The variable within the object to be influenced along with the new value
Sequence Editor shows all operations
The condition
The condition object is the input signal
Welcome Page
IMS4.prt
assembly1.prt
OK
Cancel

	En	Name
1	☑	Root
2	☑	Rods_out_1
3	☑	Rods_in_1
4	☑	Rods_out_2
5	☑	Rods_in_2
6	☑	Transporter_stop
7	☑	Transporter_on
8	☑	Stopper_down
9	☑	Stopper_up

Operation

Physics Object: Select Object (1)

Duration

Runtime Parameter

S..	Name	Oper...	Value	Unit	I...
☐	axis	:=	Stopper_Sli...		☐
☐	speed	:=	150.000000	mm/s	☐
☑	position	:=	5.000000	mm	☐
☐	active	:=	true		☐

Edit Parameter

Condition

If	Object	Parameter	Oper...	Value
If	bStopper_d...	value	==	true

Edit Condition Parameter

Select Object: Select Condition Object (1)

Name: Stopper_down

Fig. 10.15: An example of an operation to influence a position control during simulation in real time

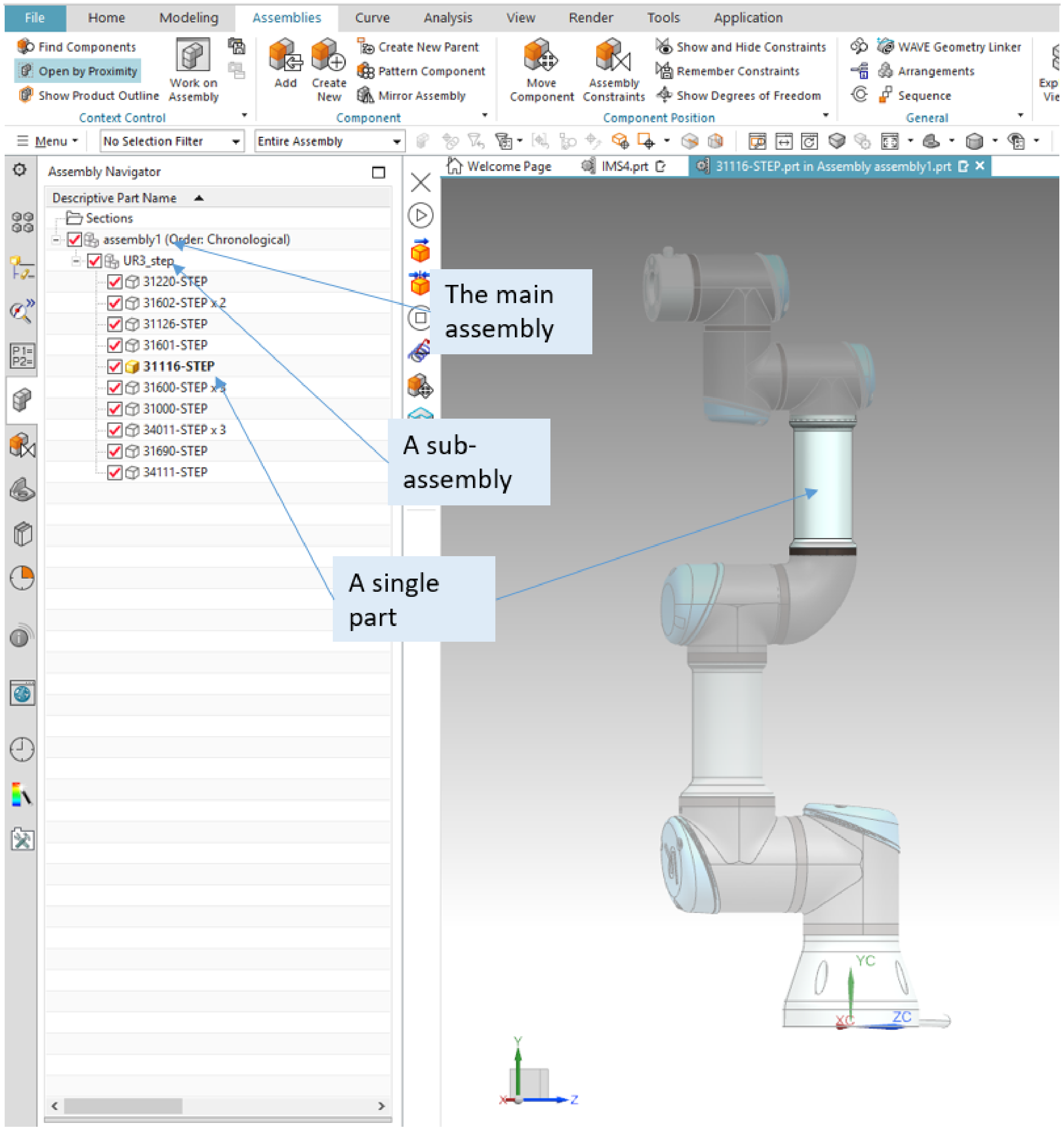
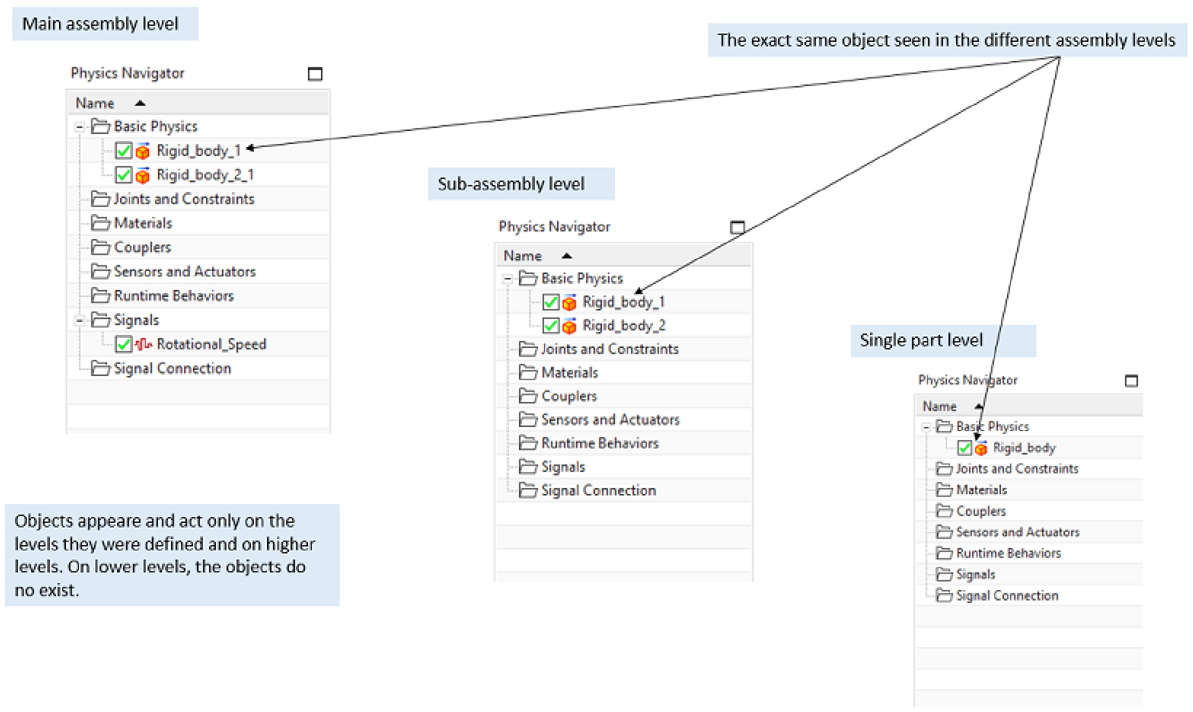


Fig. 10.16: Assemble levels

The location of a single part in the hierarchy can be changed through drag and drop between the assemblies.

The creation of physical properties (rigid bodies, collision bodies, joints, signals etc.) in Mechatronics Concept Designer can be done on any assembly level. Copying parts between files will retain the physical properties created on the copied level and on all lower levels copied along, but not the properties that were created on higher levels.

When a model is copied, the physical properties of all lower levels are copied along.



- Assigning properties on the individual part's level is handy if the part is going to be reused in many other assemblies. This way, the part can retain it's physical properties. This saves the work of reassigning physical properties to the part each time it is copied into a new assembly.
- Assigning the properties on a higher level in the assembly is handy if the part will be used in other assemblies but its properties are not needed anymore. This saves the work of manually deleting all the assigned properties when copying the part into a new assembly.

Because of a bug that causes a communication problem, it is recommended to assign physical properties to parts in a sub-assembly level or on the main assembly level. It is advised to avoid running the simulation on a single part level. Refer to the section on Troubleshooting for more details on communication bugs.

10.4 Summary

Siemens NX MCD offers a tool to create kinematic models using existing CAD models. These kinematic models can be controlled by PLCs (simulated/real), which allows for early verification of control code and a cut in real commissioning time.

CHAPTER

ELEVEN

TIA PORTAL BASICS

In this module, the TIA Portal environment for PLC-programming is introduced.

11.1 Learning Outcome

- You will get familiar with the TIA Portal environment.
- You will be able to program Siemens PLCs using FBD, LAD, and SCL programming languages.

11.1.1 Introduction

TIA Portal is a Siemens software that provides a programming environment for PLCs. Different programming languages, like FBD, LAD, and SCL (a variant of ST), are supported in TIA Portal.

This module will go through the basics of starting a new project in TIA Portal.

11.1.2 Requirements

- *PLC-Basics*

11.1.3 What you need

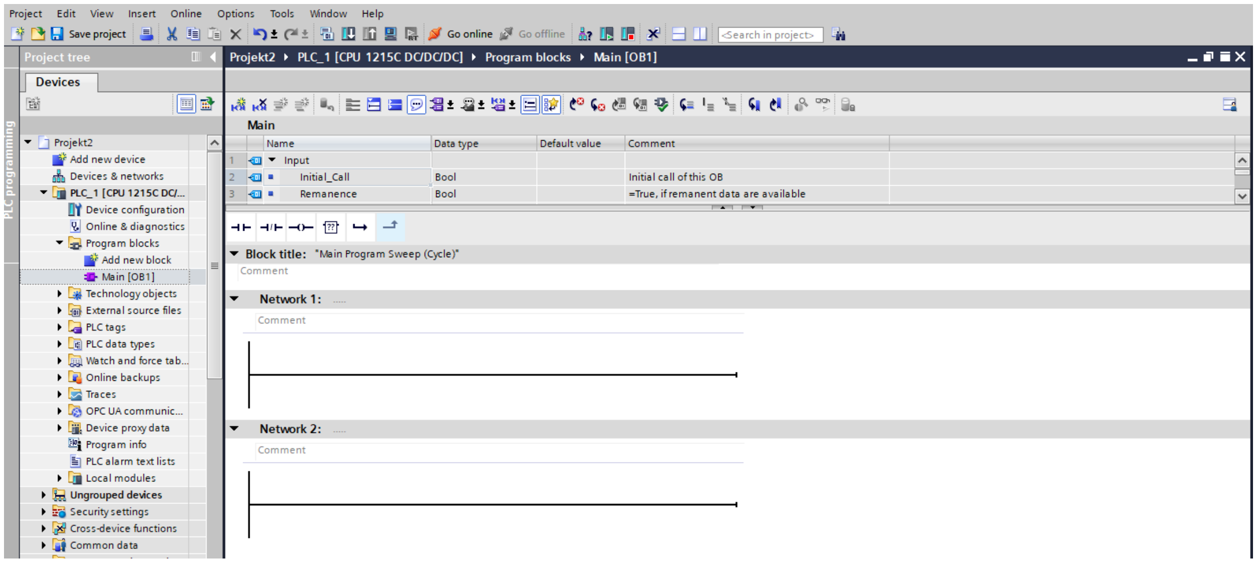
Software

- TIA Portal V15 or newer
- PLCSIM Advanced V2.0 or newer

11.2 TIA Portal

TIA Portal is a Siemens software for programming PLCs. Also, PLCs can also be simulated using *PLCSIM Advanced*.

The TIA Portal environment includes a networks window. Networks act like code lines; code (each network) is run by the CPU from up to down every PLC cycle (i.e., network 1, then network 2, and so on). Programming in networks is useful for good code organization, especially if the programming language used is LAD or FBD.



	Name	Data type	Default value	Comment
1	Input			
2	Initial_Call	Bool		Initial call of this OB
3	Remanence	Bool		=True, if remanent data are available

Block title: "Main Program Sweep (Cycle)"
Comment

Fig. 11.1: Part of the TIA Portal environment - a new project

11.2.1 Programming Blocks

The design of PLC codes in TIA Portal is done with modular blocks. There are divided into 4 types according to their use.

These are:

- **Organizational block**: These provide structure to program. They serve as a link between the user program and the OS of PLC. Main [OB1] is a type of this block.
- **Function block**: This type of block is used to create code snippets which has its own data storage i.e., it has its own variable memory where the values are stored after the code snippet finishes its intended task.
- **Function code**: This type of block is generally used to create a reusable code structure. This block does not have a database linked to it i.e., it does not save values it calculates. Thus, a local stack of temporary values is used and gets deleted once the code snippet is exited (after executing its intended task).
- **Data block**: This is used to store data received from the code snippets. The data block can be either global DB or can be an instance DB.

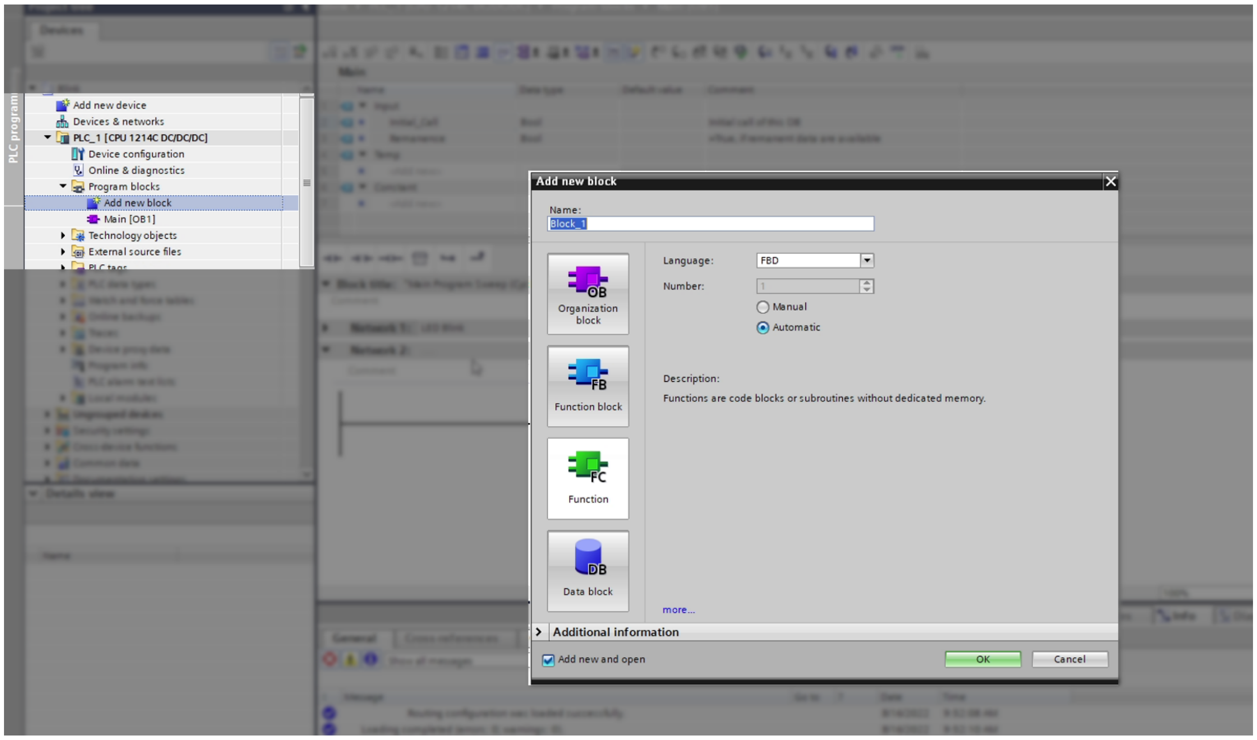
11.2.2 Creating a new function block

A new project will only have a block called `Main [OB1]`. This is the project's main function. All user created functions and function blocks will be called inside the main function.

The block `Main [OB1]` in TIA Portal can be written using either LAD or FBD standard programming languages. In the context of this module, LAD will be used.

Note: The programming language of `Main [OB1]` can be changed through right click on `Main [OB1] -> Switch programming language`.

To create a block, click on `Add new block` under `Program blocks`.



Name: Block_1

Language: FBD

Number: 1

Manual [✓ Automatic]

Description:

Functions are code blocks or subroutines without dedicated memory.

Fig. 11.2: Types of Programming blocks

Note: Blocks are useful for organizing and structuring a program.

The following figure shows a new `function block` that is named `State_machine`.

Variables can be declared and used inside the `function block`.

To run this FC, you will have to include it inside the `Main [OB1] block`.

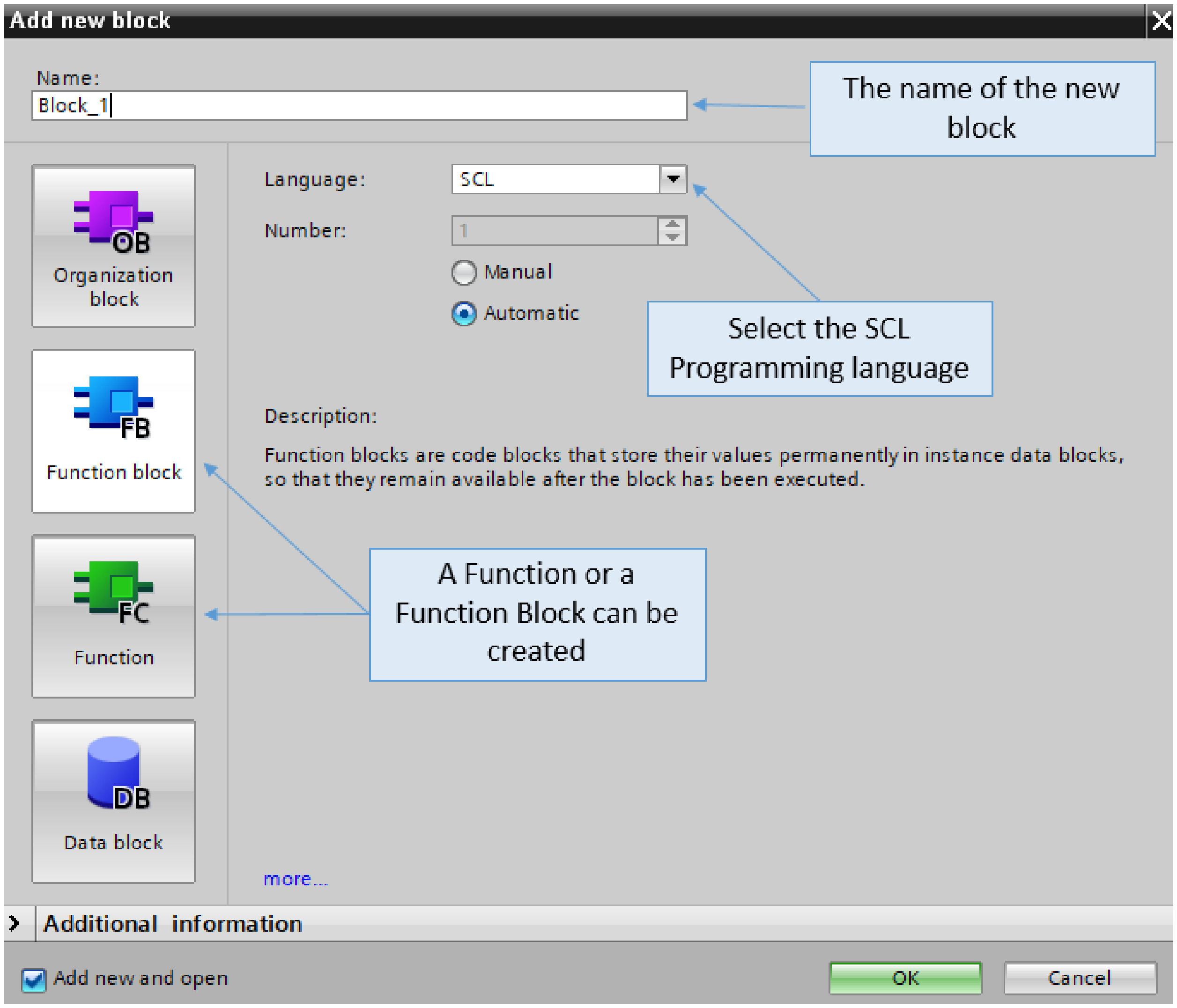
To do so, drag and drop the `function block` into a `Main [OB1]` network to create instance of the `function block`. Once you drag it on the rail of Main, a pop-up to create a DB would appear. Click `OK` to create the DB. This `data block` stores all the instance variables related to that instance of the FC.

Note: All types of `Functions` have to be called in `Main [OB1]` to execute them. If they are not in `Main [OB1]`, they won't run or do their task.

A `data block` is not created to save variables of `function code / functions`. No instances of `function code / functions` exist.

`Function blocks`, however, are initiated as instances. Variables that belong to a specific instance of a `function block` are saved in a `data block` that specifically stores variables of that instance only. Creating another instance of the same `function block` type will create another `data block` for the new instance. Hence DB instance for LED_1 will be different from LED_2 and will operate separately.

Note: `Functions` with inputs have to have inputs connected to them in `Main [OB1]`. If no inputs are connected to the function, a compilation error will occur.


Add new block
Organization block
Function block
Function
Data block
more...
Additional information
OK
Cancel
The name of the new block
Select the SCL Programming language
A Function or a Function Block can be created

Name: Block_1

Language: SCL

Number: 1

Manual

[✓ Automatic]

Description:

Function blocks are code blocks that store their values permanently in instance data blocks, so that they remain available after the block has been executed.

[✓ Add new and open]

Fig. 11.3: Creating a new block

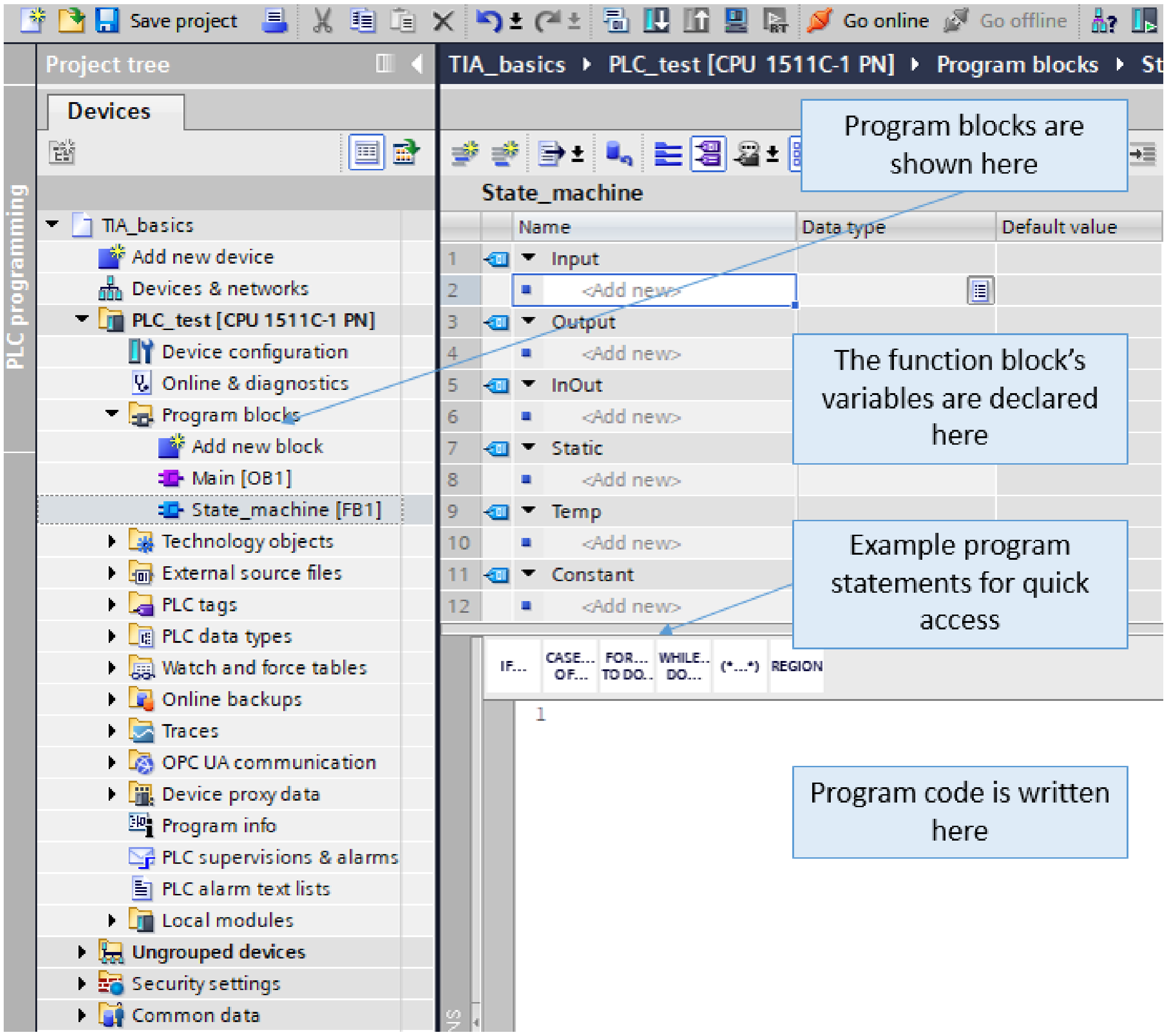
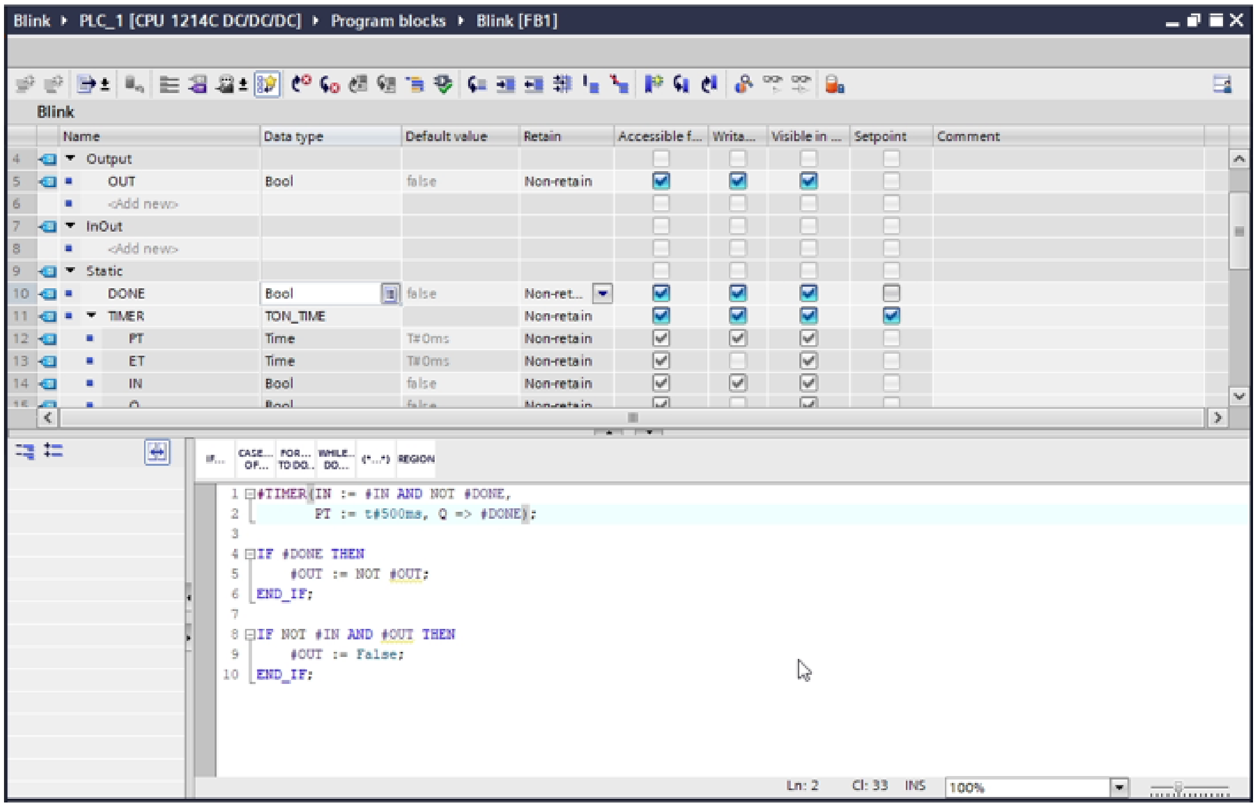


Fig. 11.4: TIA Portal function block environment



	Name	Data type	Default value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
4	▾ Output				☐	☐	☐	☐	
5	OUT	Bool	false	Non-retain	✓	✓	✓	☐	
6	<Add new>				☐	☐	☐	☐	
7	▾ InOut				☐	☐	☐	☐	
8	<Add new>				☐	☐	☐	☐	
9	▾ Static				☐	☐	☐	☐	
10	DONE	Bool	false	Non-ret...	✓	✓	✓	☐	
11	▾ TIMER	TON_TIME		Non-retain	✓	✓	✓	✓	
12	PT	Time	T#0ms	Non-retain	✓	✓	✓	☐	
13	ET	Time	T#0ms	Non-retain	✓	☐	✓	☐	
14	IN	Bool	false	Non-retain	✓	✓	✓	☐	
15	Q	Bool	false	Non-retain	✓	☐	✓	☐	

```
#TIMER(IN := #IN AND NOT #DONE,
       PT := t#500ms, Q => #DONE);

IF #DONE THEN
    #OUT := NOT #OUT;
END_IF;

IF NOT #IN AND #OUT THEN
    #OUT := False;
END_IF;
```

Fig. 11.5: Example Function Block

`Function blocks`, on the other hand, do not require inputs to be connected to them in `Main [OB1]`. Each instance of a function block has its own `Data block`, in which the inputs of that specific instance is saved.

11.2.3 Using Predefined Functions

TIA Portal provides various predefined functions ready for use. These can be found on the right side under `Basic instructions`.

For example, if you need a Mathematics based function, expand `Math functions` and select the required function.

Note: User-defined `functions` and `function blocks` are used in the same way in LAD.

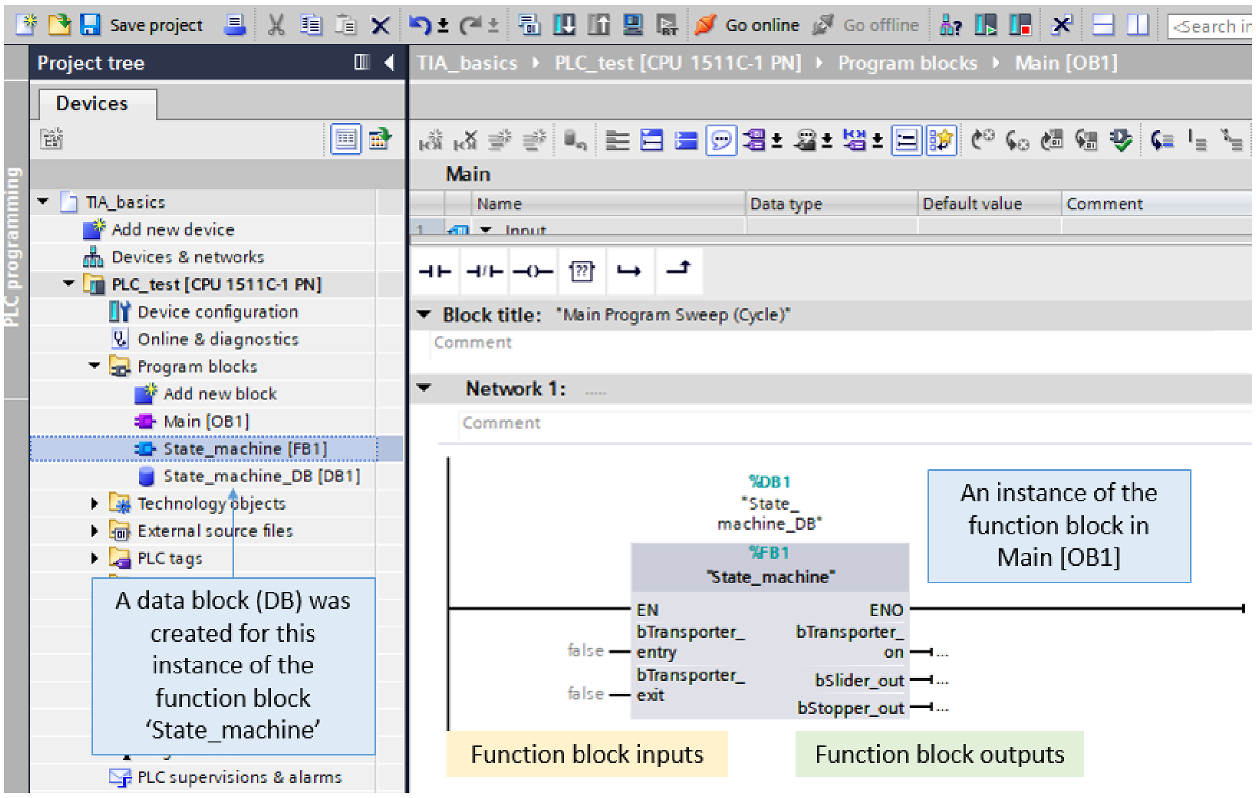


Fig. 11.6: A function block instance in Main

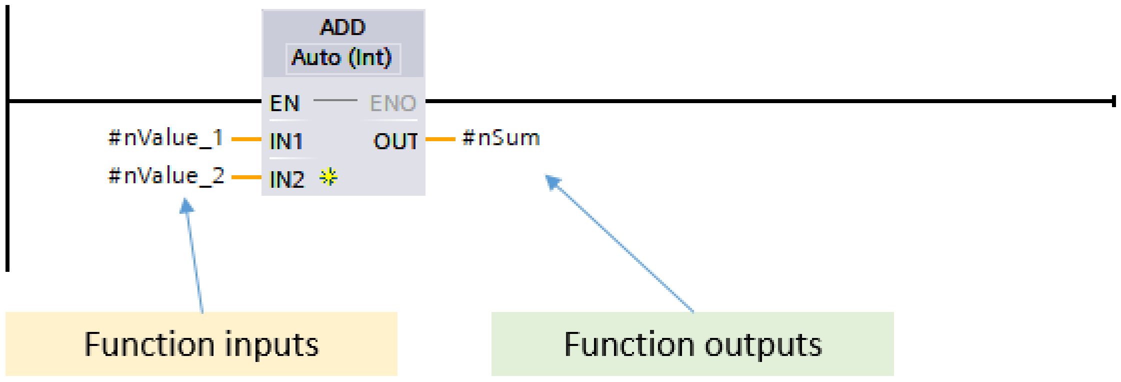


Fig. 11.7: An example of a standard function.

11.2.4 Watch and Force Tables

Watch and force tables are tables that are used to monitor variables in real-time while the PLC is running. Watch and force tables can also be used to overwrite variables in real-time. Because they can change variables' values, they are useful for testing a program. In this module, watch tables will be used to monitor and set variables while the program is running.

Deep Dive: Watch vs Force Tables

Force tables manipulate the peripherals. In this module, watch tables will be used to monitor and test the program. Watch tables are recommended for testing the program. For big programs, every scenario of inputs and outputs could be summed up in a watch table.

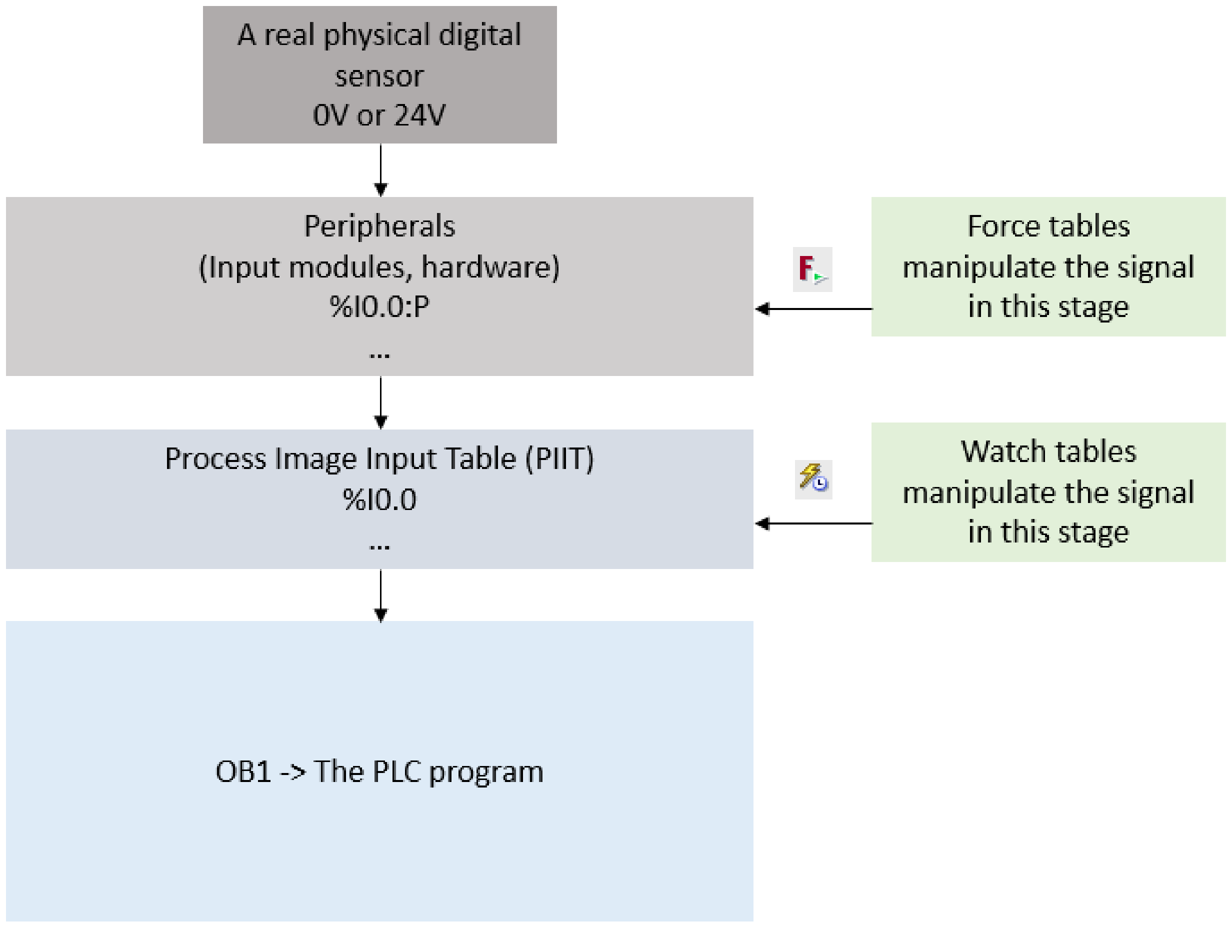

A real physical digital sensor
0V or 24V
Peripherals
(Input modules, hardware)
%I0.0:P
...
Force tables manipulate the signal in this stage
Process Image Input Table (PIIT)
%I0.0
...
Watch tables manipulate the signal in this stage
OB1 -> The PLC program

Fig. 11.8: The stage at which watch and force tables manipulate the signals

Creating Watch Tables

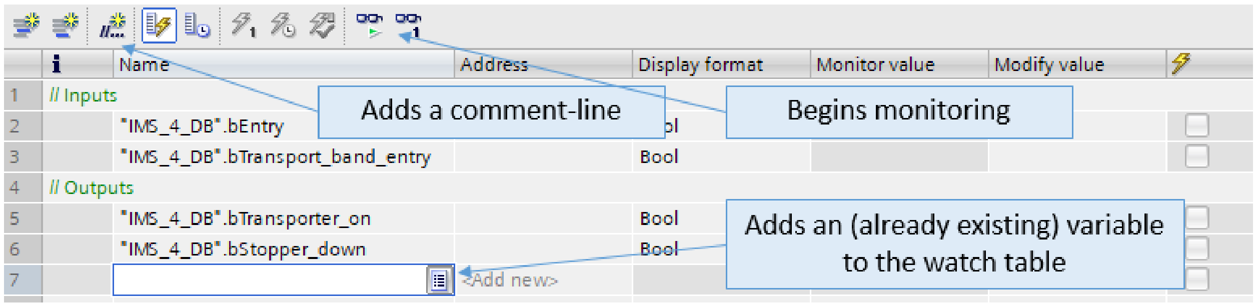
To create a new watch table, select `Add new watch table` under `Watch and force tables` in the `Project tree`.

Add the variables you need in the rows.

Set the PLC in *Online* mode (using `Online -> Go online`) and in *RUN* mode (`Online -> Start CPU`)

Once done, select the *Monitoring tool from the menu above*. Now you can watch the changes in the variables and also modify their values if required.

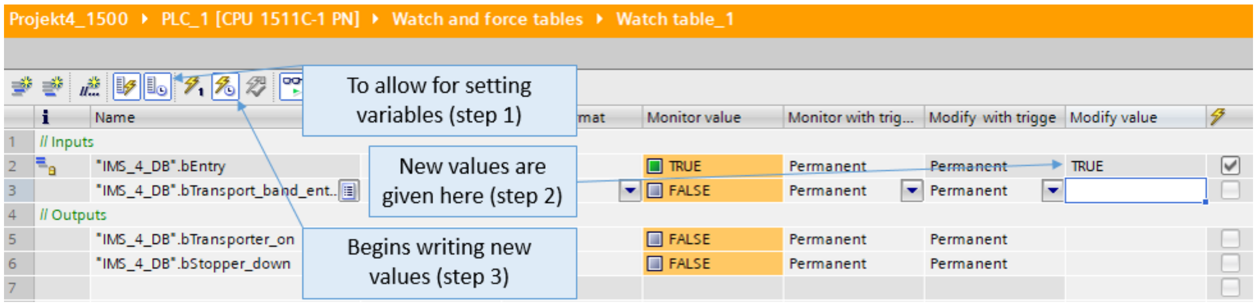
Hint: Watch tables are on PC hence you do not have to download the code again to the PLC.



	i	Name	Address	Display format	Monitor value	Modify value	⚡
1		// Inputs					
2		"IMS_4_DB".bEntry		ol			☐
3		"IMS_4_DB".bTransport_band_entry		Bool			☐
4		// Outputs					
5		"IMS_4_DB".bTransporter_on		Bool			☐
6		"IMS_4_DB".bStopper_down		Bool			☐
7			<Add new>				☐

Fig. 11.9: An example set-up of a watch table

Note: All the variables shown in this watch table are part of a created `data block` called `IMS_4_DB`, hence the `"IMS_4_DB".` at the beginning of the variables' names. The address column is empty because these variables in this example were not mapped to a PLC I/O module, but are rather variables that belong to an instance of a user-defined `function block` called `IMS_4`.


To allow for setting variables (step 1)
New values are given here (step 2)
Begins writing new values (step 3)

Projekt4_1500 ▸ PLC_1 [CPU 1511C-1 PN] ▸ Watch and force tables ▸ Watch table_1

	i	Name	mat	Monitor value	Monitor with trig...	Modify with trigge	Modify value	⚡
1		// Inputs						
2		"IMS_4_DB".bEntry		🟩 TRUE	Permanent	Permanent	TRUE	✓
3		"IMS_4_DB".bTransport_band_ent...		☐ FALSE	Permanent	Permanent		☐
4		// Outputs						
5		"IMS_4_DB".bTransporter_on		☐ FALSE	Permanent	Permanent		☐
6		"IMS_4_DB".bStopper_down		☐ FALSE	Permanent	Permanent		☐
7								☐

Fig. 11.10: An example watch table in action

Danger: Using a watch or a force table to **manipulate** variables influence the real system and overrides the program on the PLC. Be sure that it is safe to do so when working with a real PLC as it may be connected to moving parts or could unintentionally release liquids through valves.

11.2.5 Timers

There are various types of timers available according to IEC standard. Here are the commonly used timers with their timing diagram.

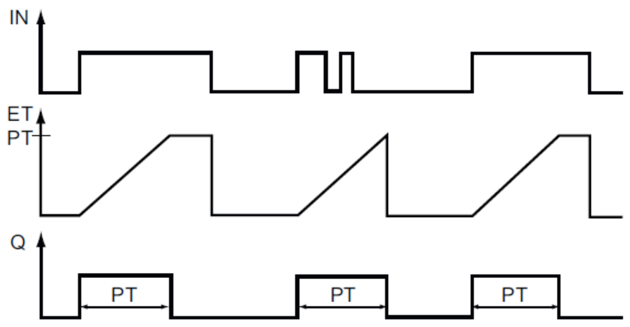
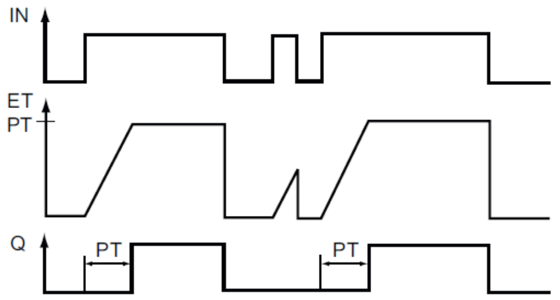
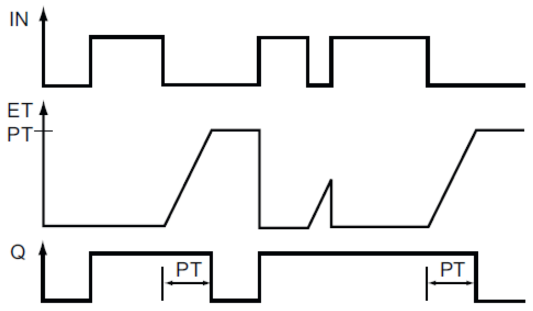
Timer	Timing diagram
TP: Generate pulse The TP timer generates a pulse with a preset width time.	
TON: Generate ON-delay The TON timer sets output Q to ON after a preset time delay.	
TOF: Generate OFF-delay The TOF timer resets output Q to OFF after a preset time delay.	

Fig. 11.11: Timers (ref: Siemens S7-1200 Programmable controller System Manual)

11.2.6 Monitoring runtime values

You can monitor runtime values using *Monitor Tool*. It looks like the image below. You will find it at various locations.

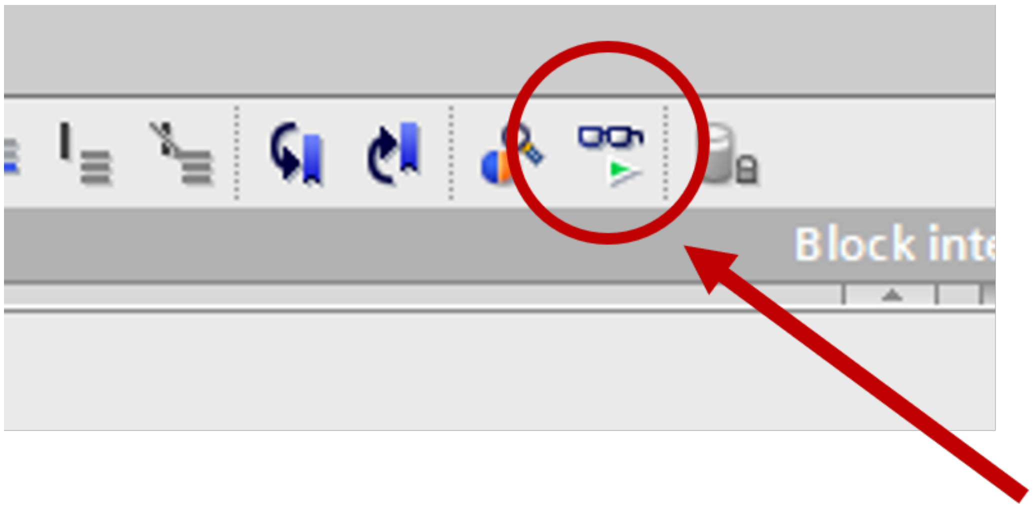


Fig. 11.12: Monitor Tool

CHAPTER

TWELVE

TIA OPC-UA

12.1 Configuring OPC-UA server in TIA Portal

To activate an OPC UA server:

- Go to `Device configuration` from `Project tree` from left side.
- Set the view to `Device view`
- Click on the PLC image and open `Properties`
- Go to `OPC UA` -> `Server` -> `General`
- Under Accessibility of the server, check the option `Activate OPC UA server` to activate it.
- In case a security pop-up appears, accept it (click OK).
- Find `Runtime licenses` in the same window (Properties)
- Under `Type of purchased license`, select SIMATIC OPC UA S7…
- Go back to `OPC UA` -> `Server` -> `General`
- Copy the server address.

Note: Write down (and copy it) the complete server address of the PLC. Please DO NOT forget opc.tcp and port number!

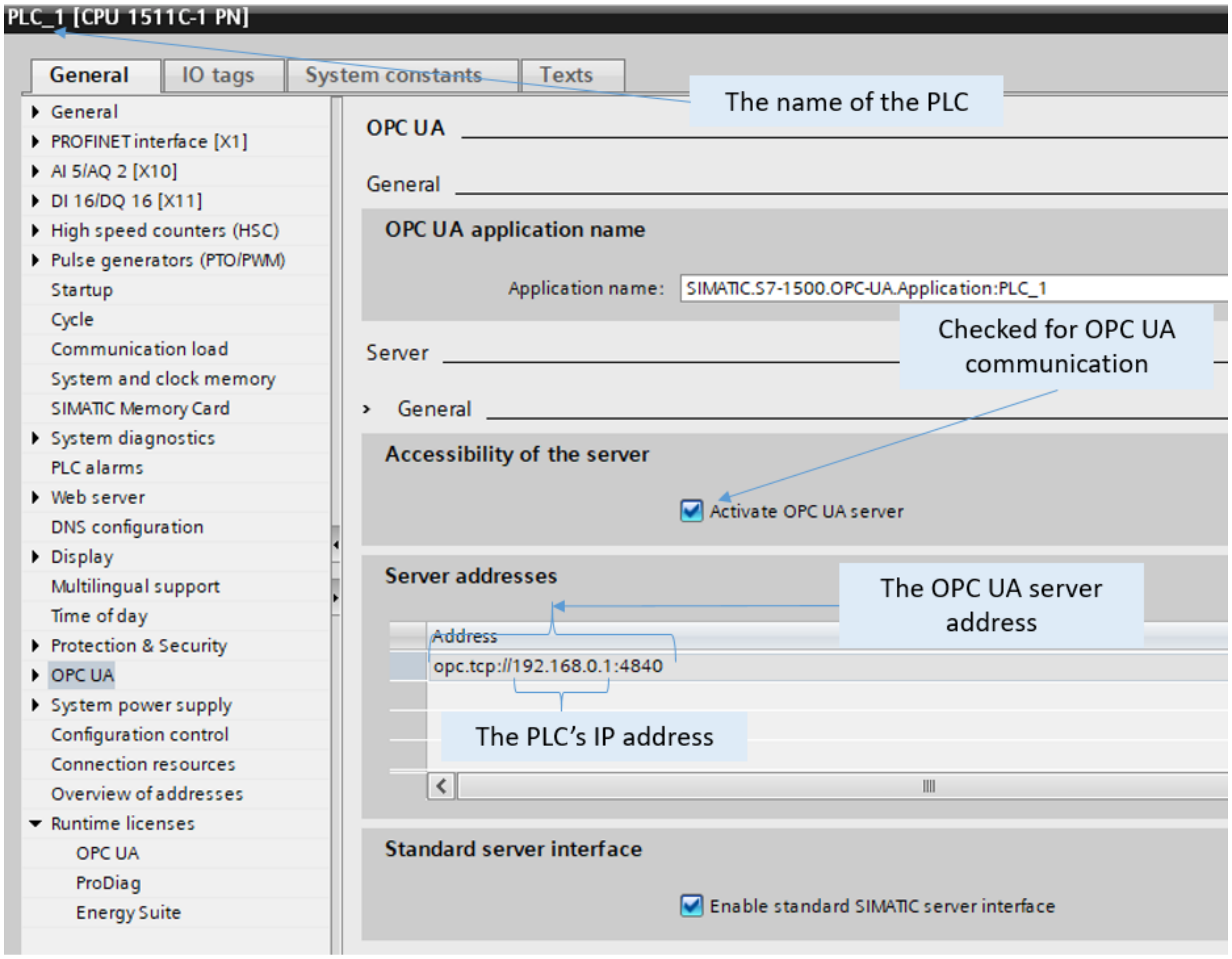
This information will be needed later when a client (for example UA Expert, Prosys OPC UA Browser or Siemens NX MCD) tries to communicate to the PLC over OPC UA.

Note: The OPC-UA server will not work if license is not set properly.

Warning: FOR THIS MODULE, PLEASE DO NOT CHANGE ANY MORE SETTINGS.

Danger: Be careful in setting security for the OPC-UA server when working on REAL PROJECTS.

Here, you are not working on hazardous equipment, hence *No Security* option under `Properties` -> `OPC UA` -> `Server` -> `Security` is enabled. Learn more in user manual before changing these settings.



Application name: SIMATIC.S7-1500.OPC-UA.Application:PLC_1

✓ Activate OPC UA server

Address
opc.tcp://192.168.0.1:4840

✓ Enable standard SIMATIC server interface

Fig. 12.1: Activating OPC UA server

12.2 Creating OPC-UA Server Interface

To add a server interface, perform the following steps:

1. Select the PLC name from the Project tree.
2. Expand the and select *OPC UA communication*
3. Select *Server interfaces* and expand it
4. Click on *Add new server interface* and rename the interface created
5. From the *Add new server interface* pop-up, select *Server interface* and click OK
6. Add the tags required to be monitored or to be modified using OPC Client. You will have to drag the tags from *OPC UA elements* to *server interface*
7. Upload the changes to the PLC

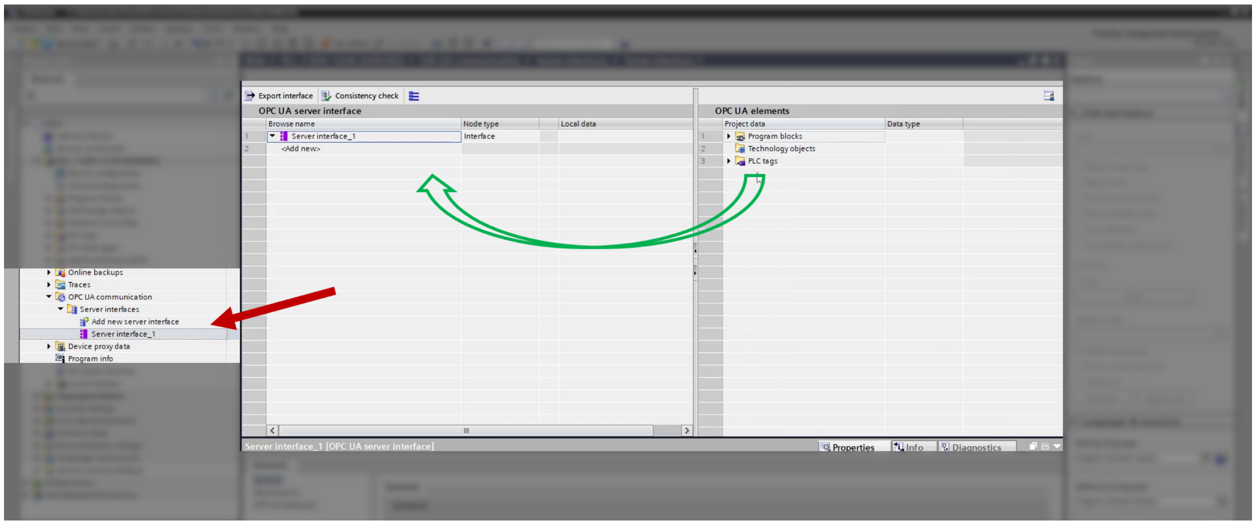


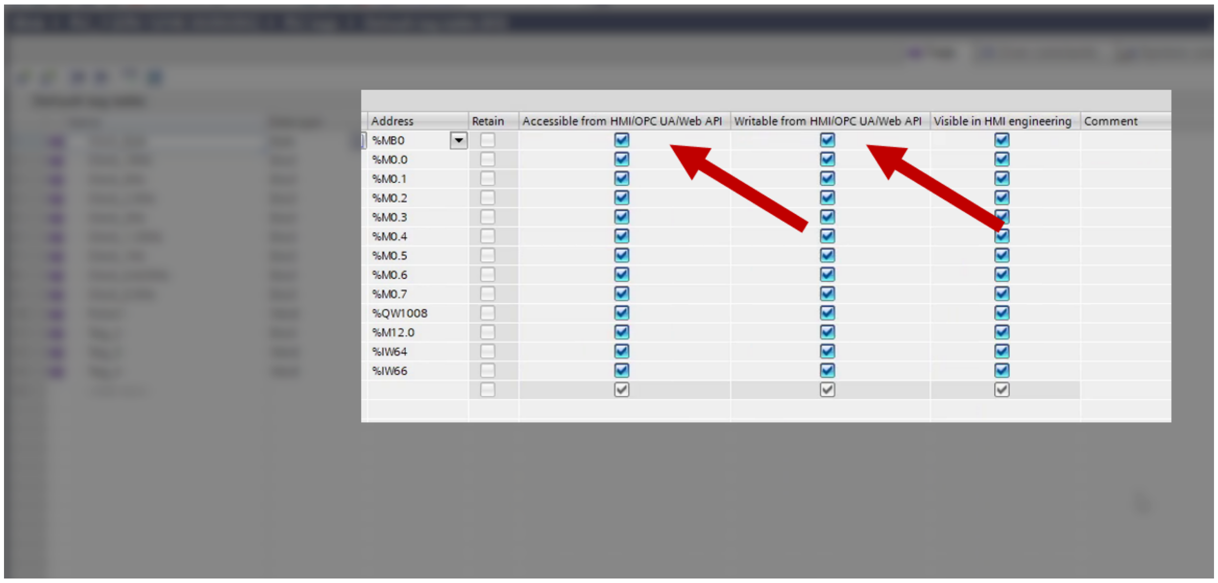
Fig. 12.2: OPC-UA Server Interface

Note: Keep the PLC connected with LAN cable.

- When uploading any hardware changes, keep the PLC in `STOP` mode and in offline mode. i.e., `Online` -> `Stop CPU` & `Online` -> `Go offline`
- When uploading only software changes, you can keep PLC in Online mode.

Danger: Be cautious in setting access of variable. You should not provide open access to all variables (tags).

Set the access in *Default Tag Table*



Address	Retain	Accessible from HMI/OPC UA/Web API	Writable from HMI/OPC UA/Web API	Visible in HMI engineering	Comment
%MB0	☐	✓	✓	✓	
%M0.0	☐	✓	✓	✓	
%M0.1	☐	✓	✓	✓	
%M0.2	☐	✓	✓	✓	
%M0.3	☐	✓	✓	✓	
%M0.4	☐	✓	✓	✓	
%M0.5	☐	✓	✓	✓	
%M0.6	☐	✓	✓	✓	
%M0.7	☐	✓	✓	✓	
%QW1008	☐	✓	✓	✓	
%M12.0	☐	✓	✓	✓	
%IW64	☐	✓	✓	✓	
%IW66	☐	✓	✓	✓	
	☐	✓	✓	✓	

Fig. 12.3: Default Tag Table

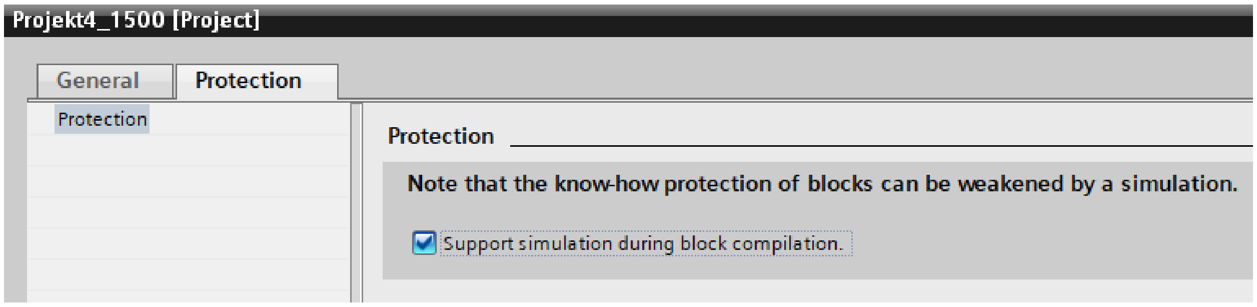
CHAPTER

THIRTEEN

PLCSIM ADVANCED

13.1 Setting up simulation support in TIA Portal

Before starting a PLC simulation, one should first make sure the TIA Portal program allows simulations. To do that, right click on `Project's name` in the `Project tree``and select ``Properties`. In the `Protection` tab, a tick has to be placed next to `Support simulation during block compilation`.


Projekt4_1500 [Project]

Protection

Note that the know-how protection of blocks can be weakened by a simulation.

☑ Support simulation during block compilation.

Fig. 13.1: Adding support for simulation in TIA Portal

Warning: PLCSIM does not offer simulated PLCs with communication features. If communication features, such as OPC-UA are required, simulations must be carried out using PLCSIM Advanced.

13.2 Launching PLCSIM Advanced and creating a PLC instance

Now PLCSIM can be launched. For that, double click on the `PLCSIM Advanced` icon on the desktop or search for `PLCSIM Advanced` in the search bar and click `Enter`.

Note: PLCSIM Advanced does not launch a window (in older PLCSIM Advanced versions). When it runs, it shows up on the tray icon. A right click on the tray icon will show the control panel.

Note: The PLC name and IP address have to be identical to those in the TIA Portal program. Use a subnet mask of `255.255.255.0`. Please note that the subnet mask depends on the network settings.

After `Start` is clicked, a virtual PLC instance should run.

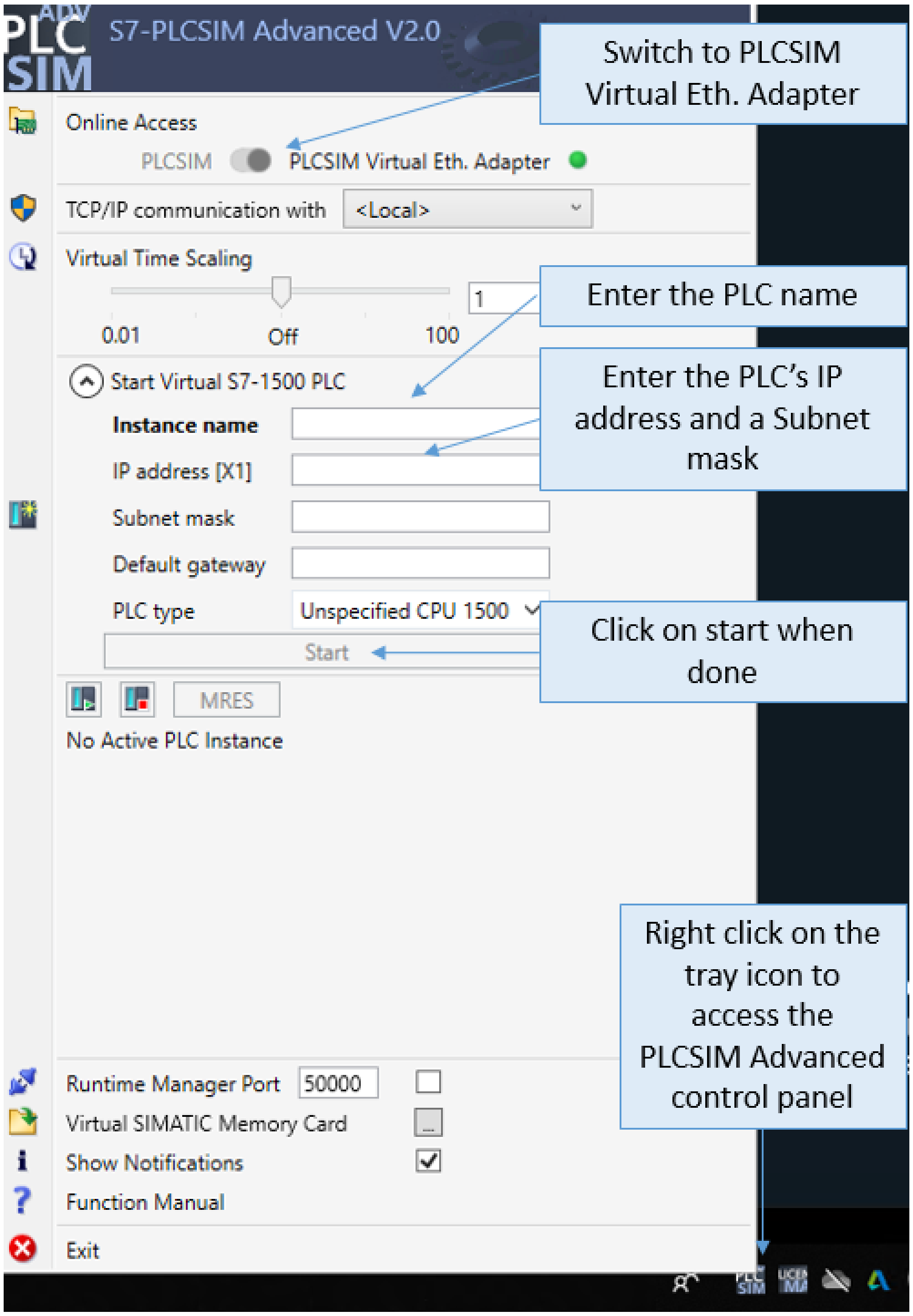


Fig. 13.2: Setting up a PLC instance

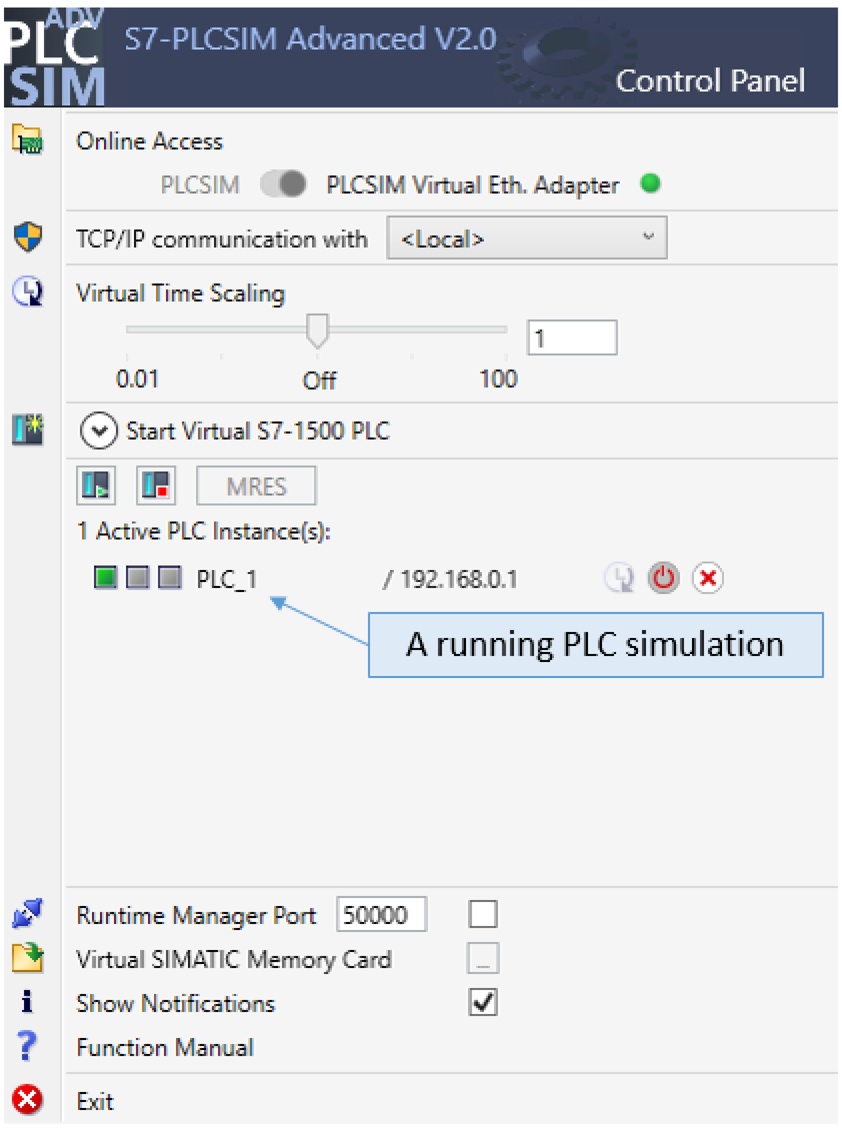


Fig. 13.3: A running virtual PLC instance

13.2.1 Deep Dive: PLCSIM vs PLCSIM Advanced

With TIA Portal, it is possible to simulate a Siemens PLC and test a program on the simulated PLC. With the software PLCSIM, it is possible to create such a simulation (12xx and 15xx PLCs), however, the simulated PLC has no communication capabilities. The software PLCSIM Advanced (a separate software than PLCSIM) can simulate Siemens 15xx PLCs with communication capabilities.

13.2.2 Summary

Simulating a PLC using PLCSIM Advanced allows for communication. The simulated PLC can communicate with other devices (through OPC-UA for example), allowing for a more realistic validation of control code.

CHAPTER

FOURTEEN

KONZEPTION EINER FERTIGUNGSANLAGE FÜR SANDWICHES

Anhand des Beispiels der Sandwichproduktion werden in kurzer Form alle notwendigen Arbeitsschritte umgesetzt, um ein Fertigungskonzept zu entwickeln. Bitte beachten Sie, dass dies nur eine kurze Übersicht darstellt und die einzelnen Schritte in der Realität deutlich ausführlicher beleuchtet werden müssen.

14.1 Teil 1: Anforderungen

14.1.1 Grundinformationen und Anforderungen

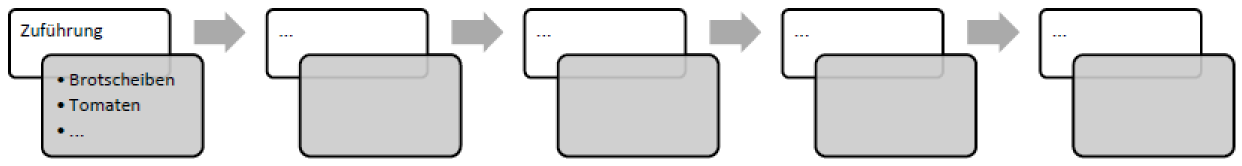
Als Produkt wird ein handelsübliches Sandwich gewählt, welches abgepackt in Supermärkten gekauft werden kann. Anhand der Variation der Beläge wird eine Variantenvielfalt erzeugt.

Das Sandwich soll im vorliegenden Fall mit 4 verschiedenen Zutaten in allen Kombinationen belegt werden können. Dem Fertigungsprozess werden die Brotscheiben, die geschnittenen Tomaten, die Sauce, der Käse und der Schinken zugeführt. Die Fertigungsanlage soll die Brotscheiben belegen, zuschneiden und am Ende verpacken. Die fertigen Sandwiches sollen mittels einer visuellen Kontrolle überprüft werden.

Einen Beispielprozess zur Sandwichproduktion finden Sie hier:

https://youtu.be/YE0pjv-3Yzs

Todo: Nennen Sie die Arbeitsschritte, die für die Sandwichproduktion erforderlich sind



Hint: Ein grober Arbeitsablauf mit 5-10 Schritten ist ausreichend. Sie können eine andere Art der Visualisierung nutzen und müssen nicht das vorgegebene Layout weiterführen.

14.1.2 Erstellen Sie eine grundlegende Anforderungsliste für die Produktion

Während der Planungsphase lassen sich Anpassungen bzw. Änderungen noch kostengünstig durchführen, Änderungen nach Fertigstellung der Anlage werden teuer.

Todo: Nehmen Sie geeignete Daten für die Anforderungsliste an.

Flächen-/Raumbedarf	
Einsatzdauer und Taktzeiten	
Kostenrahmen	
Flexibilität	
Wunsch nach bestimmten Komponenten	
Zeitpunkt der gewünschten Einsatzbereitschaft	
Einbindung in vorhandene Anlagen/Systeme	
Ziele der Anlage: Produktivität, Qualität	
Funktionen/Aufgaben	
Automatisierungsgrad	

14.1.3 Voraussetzungen für eine erfolgreiche Automatisierung

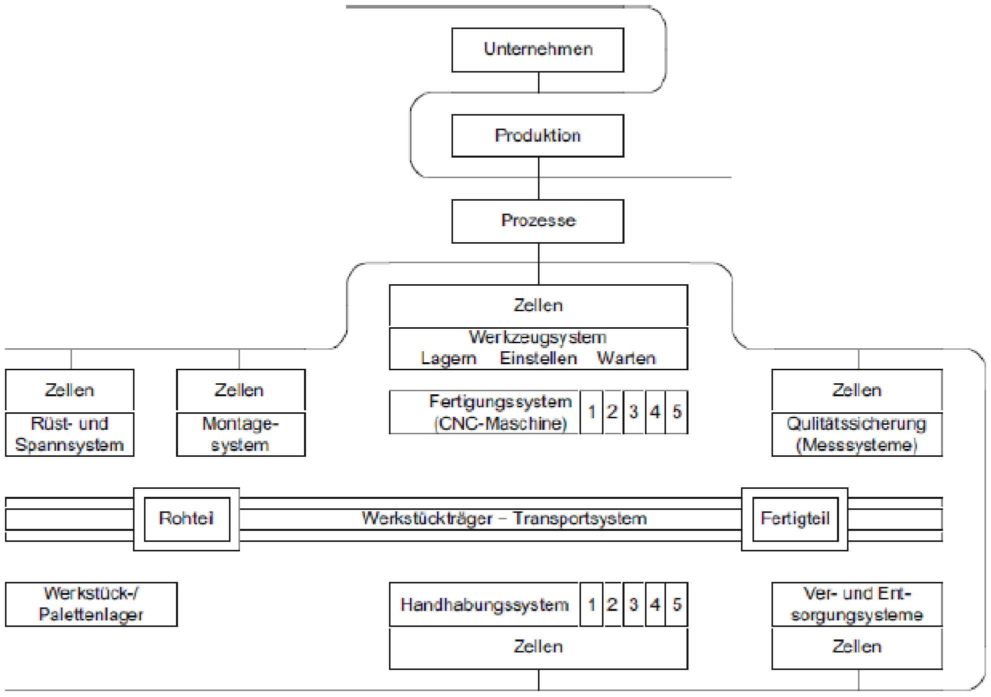
Todo: Beantworten Sie die folgenden Fragen: a) Ist das Produkt automatisierungsgerecht? b) Ist der Herstellungsprozess automatisierungsgerecht?

Beachten Sie nachfolgend auch den wirtschaftlichen Automatisierungsgrad. Anzustreben bzw. zu beachten ist also Folgendes:

- nur das automatisieren, was sinnvoll ist, und nicht das, was technisch möglich ist (Nutzen/ Kosten),
- ein vernünftiges Verhältnis von menschlicher Arbeit und Maschinenarbeit,
- kleine, überschaubare Einheiten bilden,
- mit zunehmender Komplexität der technischen Strukturen wächst der Aufwand stärker als die Systemwirksamkeit.

14.1.4 Gliederung des Gesamtsystems in Subsysteme

Die Gliederung eines Gesamtsystems in Teil-/Subsysteme führt zu einer besseren Übersicht über die Funktionen und Zusammenhänge in der Anlage und damit zu besseren und schnelleren Lösungen.



Komponenten	Beispiele
Materialflusssysteme	schienengebundene Transportwagen, Palettenumlaufsysteme, Transportbänder
Handhabungssysteme	Einlegegeräte, Roboter
Rüst-und Spannsysteme	automatische Spannvorrichtungen, Spannpaletten, Palettenwechsler
Fertigungssysteme	CNC-Bearbeitungszentren, automatische Montagestationen, Reinigungsstationen
Werkzeugsysteme	Werkzeugspeicher, Werkzeugwechsler, Einstellung und Codierung, Werkzeugüberwachung, Werkzeugdatenbereitstellung
Steuer- und Regelungssysteme	numerische/speicherprogammierbare Steuerungen, Regeleinrichtungen, PCs
Informations- und Kommunikationssysteme	Rechner, Netzwerke, Bussysteme
Leitsysteme	Leitrechner, Netzwerke, PPS-Programme
Systeme zur Maschinendaten- und Betriebsdatenerfassung (MDE, BDE)	Sensoren, Codiersysteme
Qualitätssicherungssysteme	BDE, MDE plus entsprechende Programme
Versorgungs- und Entsorgungssysteme	Spänebeseitigung, Abfallentsorgung, Recycling

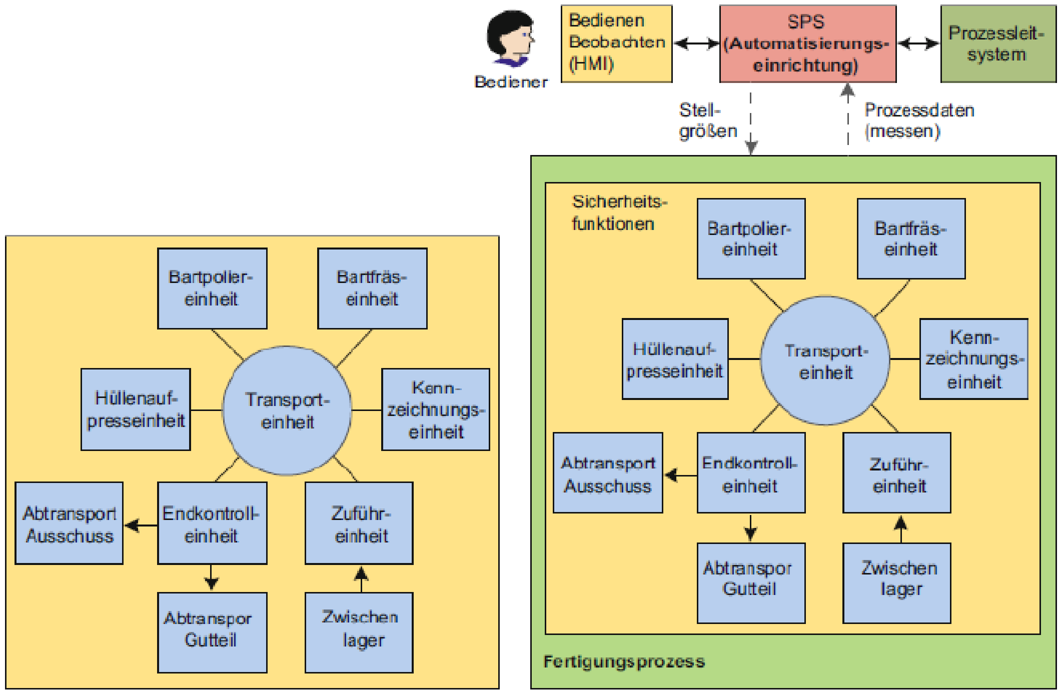
Todo: Nutzen Sie die folgende Tabelle, um Ihr Gesamtsystem zu organisieren. Beispiele sind bereits in der Tabelle gelistet, diese können Sie an Ihren Prozess anpassen.

Man brauch jemanden, der (Funktion)…	Teilsystem	Analoge Komponente eines flexiblen Fertigungssystems
das Brot heranschafft.	Zuführeinheit	Versorgungs- und Entsorgungssysteme
das Brot transportiert.	Transporteinheit	
das „Sagen“ hat, also einen Chef.	Prozessleitsystem oder Werker über Panel	Leitsystem
das Weitergeben von Information ermöglicht.	Automatisierungseinrichtung mit Bus-System sowie Sensoren und Aktoren	Steuer- und Regelungssystem, Informations- und Kommunikationssystem, Systeme zur Datenerfassung
Etc.		

14.1.5 Definition von Arbeitsstationen

Todo: Skizzieren Sie die Grobstruktur der Fertigungsanlage für Sandwiches. Beachten Sie hierbei, dass nicht für jede Funktion eine Arbeitsstation bereitgestellt werden muss. Zudem sollen die Stationen gleichmäßig ausgelastet sein, um Engpässe zu verhindern. Besonders teure Komponenten (z. B. Roboter) sollen gut ausgelastet sein.

Beispiel zum Vergleich: Grobstruktur einer Fertigungsanlage für Schlüssel



Hint: Eine Skizze inkl. Automatisierungseinrichtungen ist ausreichend.

14.2 Teil 2: Steuerung für die Fertigungsanlage für Sandwichbrote

Die Sandwichscheiben sollen in einer Vorfertigung hergestellt werden. Hierbei werden folgende Schritte umgesetzt:

- Zuführung Zutaten (Mehl, Wasser, etc.)
- Mischen/ Kneten der Zutaten
- Portionieren
- Gären
- Backen
- Separieren
- Schneiden

14.2.1 Anforderungen an Steuerungssysteme

(siehe Vorlesung „Organisation von Steuerungskomponenten“)

Todo: Nennen Sie die Anforderungen an die Steuerungstechnik für den Fertigungsbereich „Mischen“.

Todo: Nennen Sie die Anforderungen an die Steuerungstechnik für den Fertigungsbereich „Backen“.

Todo: Nennen Sie die Anforderungen an die Steuerungstechnik für den Fertigungsbereich „Schneiden“.

14.2.2 Steuerungssysteme und -arten

Todo: Welches Steuerungsmittel (mechanisch, hydraulisch, pneumatisch, elektrisch) würden Sie für eine Pick-&-Place Anwendung von Brotscheiben wählen?Begründen Sie Ihre Antwort.

Todo: Welches Steuerungsmittel (mechanisch, hydraulisch, pneumatisch, elektrisch) würden Sie für die Backofensteuerung wählen, wenn es eine hohe Variantenvielfalt an Brotsorten gibt? Begründen Sie Ihre Antwort.

14.3 Teil 3: Sensorik und Aktoren für die Fertigungsanlage für Sandwichbrote

14.3.1 Sensorik

(siehe Kapitel „Organisation von Steuerungskomponenten“)

Todo: Wählen Sie einen geeigneten Sensor zum Wiegen der Grundzutaten Mehl und Wasser aus.

Todo: Füllen Sie die folgende Tabelle aus. Für die Begründungen sind 1-2 Sätze ausreichend.

Digital oder Analog (inkl. Begründung)	
Funktionsprinzip (inkl. Begründung)	
Sensorparameter: Nennen Sie drei Auswahlkriterien für den Anwendungsfall (siehe Folie „Sensoren – Zusammenfassung“)	
Auswahl eines realen Sensors: Fügen Sie den Namen und die Internetseite ein	
Eigenschaften des realen Sensors: Listen Sie die wichtigsten (5-10) Eigenschaften aus dem Datenblatt aus und markieren Sie die zuvor genannten Auswahlkriterien	
Einbau: Beschreiben Sie in 2-3 Sätzen einen möglichen Einbauort in die Produktionslinie	

Todo: Wählen Sie einen geeigneten Sensor zum Zählen der fertiggebackenen Brote für eine automatisierte Fertigungslinie aus.

Todo: Füllen Sie die folgende Tabelle aus. Für die Begründungen sind 1-2 Sätze ausreichend.

Digital oder Analog (inkl. Begründung)	
Funktionsprinzip (inkl. Begründung)	
Sensorparameter: Nennen Sie drei Auswahlkriterien für den Anwendungsfall (siehe Folie „Sensoren – Zusammenfassung“)	
Auswahl eines realen Sensors: Fügen Sie den Namen und die Internetseite ein	
Eigenschaften des realen Sensors: Listen Sie die wichtigsten (5-10) Eigenschaften aus dem Datenblatt aus und markieren Sie die zuvor genannten Auswahlkriterien	
Einbau: Beschreiben Sie in 2-3 Sätzen einen möglichen Einbauort in die Produktionslinie	

Todo: Wählen Sie einen geeigneten Sensor zur Überwachung der Temperatur im Backautomaten aus.

Todo: Füllen Sie die folgende Tabelle aus. Für die Begründungen sind 1-2 Sätze ausreichend.

Digital oder Analog (inkl. Begründung)	
Funktionsprinzip (inkl. Begründung)	
Sensorparameter: Nennen Sie drei Auswahlkriterien für den Anwendungsfall (siehe Folie „Sensoren – Zusammenfassung“)	
Auswahl eines realen Sensors: Fügen Sie den Namen und die Internetseite ein	
Eigenschaften des realen Sensors: Listen Sie die wichtigsten (5-10) Eigenschaften aus dem Datenblatt aus und markieren Sie die zuvor genannten Auswahlkriterien	
Einbau: Beschreiben Sie in 2-3 Sätzen einen möglichen Einbauort in die Produktionslinie	

14.3.2 Aktorik zum Transport einzelner Brote vom Förderband auf eine Rutsche

Die fertigen Brote liegen nach dem Backvorgang auf einem Förderband und sollen vom Förderband aus auf verschiedene Rutschen verteilt werden.

Todo: Nennen Sie drei Möglichkeiten zur Realisierung des Sortiervorgangs und beschreiben Sie die Möglichkeiten in 2-3 Sätze.

Todo: Bewerten Sie die Möglichkeiten in Bezug auf Komplexität, erwartete Kosten und Flexibilität in Hinblick auf unterschiedliche Brotdimensionen.

Todo: Wählen Sie eine Möglichkeit aus und nennen Sie die dafür benötigten Aktoren.

CHAPTER

FIFTEEN

BASICS OF AUTOMATION TECHNOLOGY

1. Ordnen Sie die folgenden Begriffe dem Schaubild zu

a) Sondermaschinen

b) Bearbeitungszentren

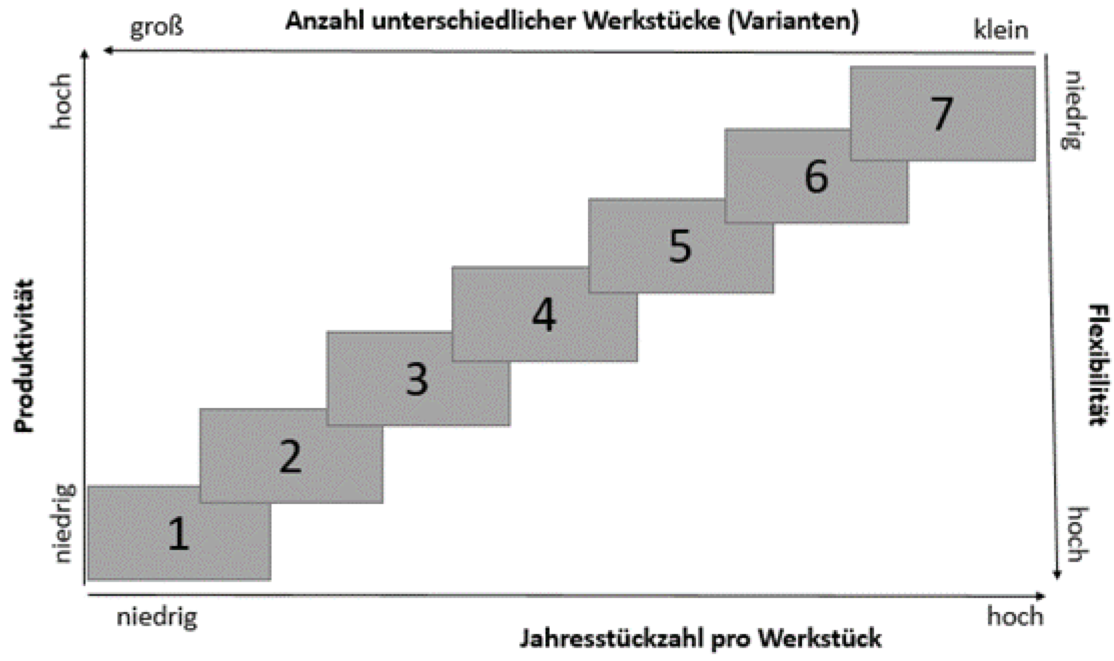
c) CNC-Maschinen

d) Konventionelle Universalmaschinen

e) Transferstraßen

f) Flexible Fertigungssysteme

g) Flexible Fertigungszellen


groß
Anzahl unterschiedlicher Werkstücke (Varianten)
klein
hoch
Produktivität
niedrig
niedrig
Flexibilität
hoch
1
2
3
4
5
6
7
niedrig
Jahresstückzahl pro Werkstück
hoch

CHAPTER

SIXTEEN

BASICS OF PNEUMATIC ELEMENTS

16.1 Circuit 1

Create a pneumtatic circuit in FluidSIM with the following properties:

- Cylinder 1 extends when button 1 is pressed
- When cylinder 1 is in the end position, cylinder 2 extends
- When cylinder 2 is in the end position, both cylinders retract again

Do only use valves to create the program logic.

16.2 Circuit 2

Create a circuit in FluidSIM with the following properties:

- Cylinder 1 extends when pushbutton 1 is pressed
- Cylinder 1 retracts when pushbutton 2 is pressed
- When cylinder 1 is in the end position, cylinder 2 extends
- As soon as cylinder 1 is no longer in the end position, cylinder 2 retracts again
- Cylinder 3 moves out when cylinder 1 and 2 are in the end position
- Cylinder 3 retracts when cylinders 1 and 2 are in the home position

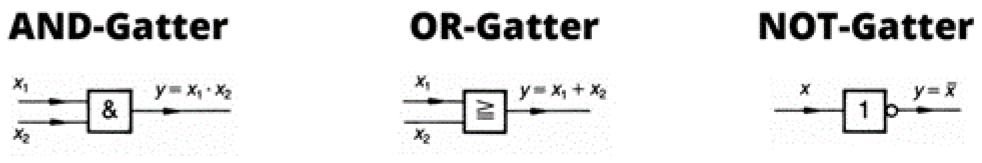
Do only use valves to create the program logic.

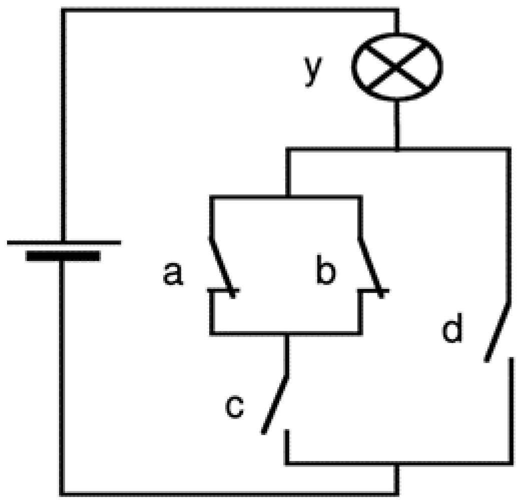
CHAPTER

SEVENTEEN

PNEUMATICS AND THE DIGITAL MODULE

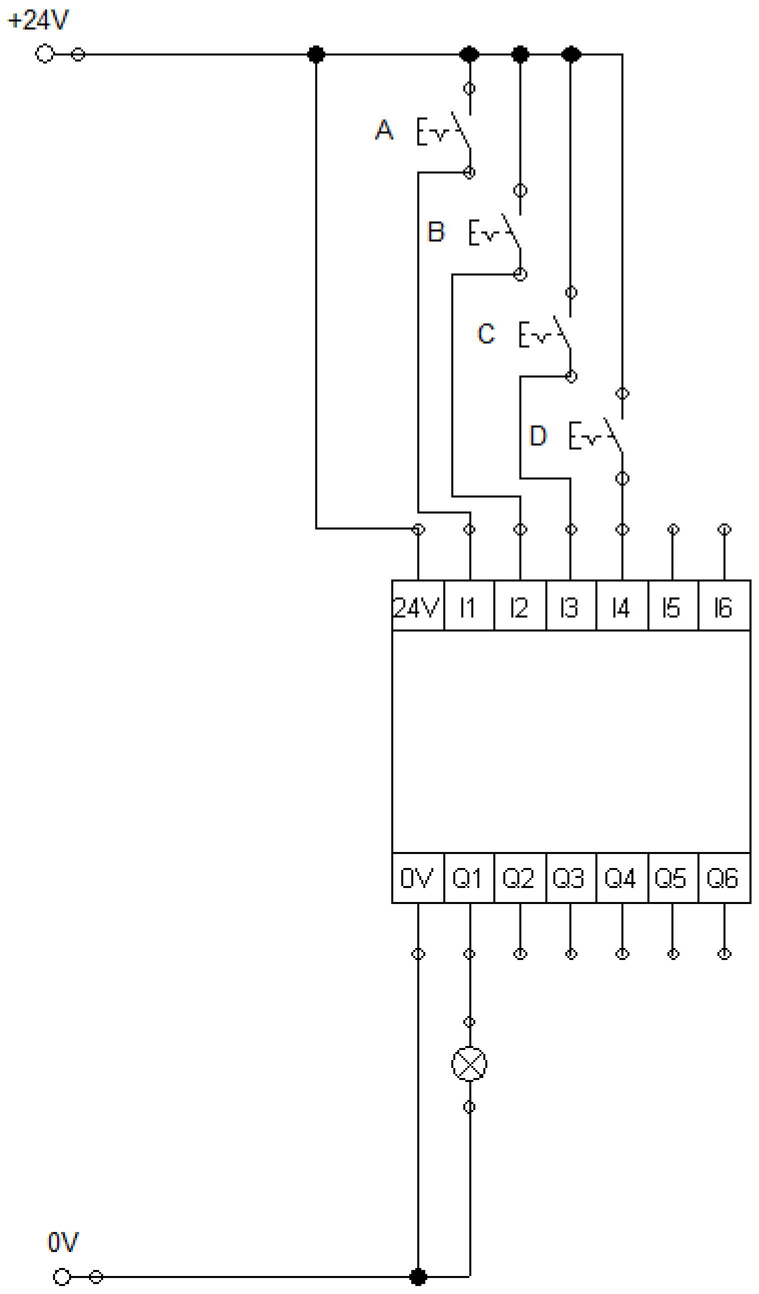
17.1 Logic modules





Todo: Implement the contact logic shown with logic modules (inputs d, c, b, a / output y).

Hint: If you want to test your contact logic in FluidSim, you can use the setup shown in the following figure.


+24V
A
B
C
D
24V
I1
I2
I3
I4
I5
I6
0V
Q1
Q2
Q3
Q4
Q5
Q6
0V

17.2 Pneumatic feed drive with digital module

A feed drive with pneumatic components is to be implemented that operates under certain conditions.

- The feed drive is implemented by a cylinder that returns to its home position when depressurized
- The piston speed is influenced by means of a control element
- **The piston may only extend if the following conditions are met:**
  - A two-hand control must be ensured (Two pushbuttons must be permanently actuated during the feed movement. This ensures that the operator's hands are out of the danger zone).
  - The piston is not in its end position.
- As soon as the end position is reached, the piston automatically retracts completely (even if both buttons are still pressed).

Todo: Create a circuit in FluidSIM with the specified properties and realize the program logic by means of a digital module.

Todo: Create the whole logic in the digital module; do **not** add logic outside (e.g. by using normally closed contacts or parallel paths)

17.3 Industrial roller shutter door

Description:

- The rolling gate is opened and closed by the gatekeeper using an "up" and "down" button.
- The movement can be interrupted at any time using a STOP button.
- Two motors are used: One motor for the movement upwards, one motor for the movement downwards.
- When the gate is fully open or closed, the motor is switched off.
- An indicator light shows the operation of the door 5 seconds before the movement and during the movement.
- A safety pressure bar ensures that the gate stops on contact, immediately moves up and stops at the top.

Todo: Create a principle sketch of the rolling gate

Todo: Create the circuit diagram with logic gates in the digital module and with the hardware components (motors, pushbuttons, etc.)

RELAY TECHNOLOGY

18.1 Traffic light using step chains

Description:

- The process starts when a button is pushed
- Red phase 5 seconds
- Yellow phase 3 seconds
- Green phase 10 seconds

Todo: Create a state diagram for a German traffic light system.

Todo: Since this is a step chain, relay technology can be used for the implementation. To do this, create the required circuit diagram in FluidSIM.

Hint: Take the sequential circuit from the lecture as a starting point.

CHAPTER

NINETEEN

PLC - GENERAL KNOWLEDGE

19.1 IEC 61131 - Comprehension questions

- What is the cycle time of a task?
- What is the difference between an FB and FC?
- What is meant by multitasking?
- What standard functions (FCs) and standard function blocks (FBs) are there according to IEC? Name four examples.
- Can a function have several outputs?
- Is it possible to address directly represented variables in a function block, e.g. the input %IX6.0? What problem does this cause?
- What types of variables are there in IEC? What is the difference between global, local and direct variables?
- What data types are there in the IEC?
- How is the communication within a program?
- How is the communication between different programs?
- Which programming languages are there according to IEC 61131?
- Create an example in ST for each of the following statements: `IF`, `CASE`, `FOR`, and `EXIT`.

CHAPTER

TWENTY

PLC - PRACTICAL EXERCISES

20.1 Industrial roller door

Description:

- Assumption: At system start the rolling gate is closed.
- The rolling gate is operated by the gatekeeper by means of a pushbutton. If the button is pressed in the closed state, the gate opens until a limit switch is reached. If the button is pressed in the open state, the gate closes after a waiting time of 5s until a limit switch is reached. If the button is pressed while moving or during the waiting time, the gate stops. If the button is pressed again, the gate moves in the opposite direction than before the stop.
- Two motors are used (open & close). It must be ensured that only one motor is controlled at a time. When the rolling gate is fully open or closed, the motors are switched off.
- In addition, there is an emergency stop switch and an acknowledgement button. If the emergency stop button is pressed, the door stops. To restart the rolling door after an emergency stop, the acknowledgement button must be pressed.
- A safety pressure bar ensures that the gate stops on contact, immediately moves up and remains up.
- Maintenance is required after N upward movements of the rolling gate (upward movements are counted when the upper limit switch is reached). N is a static variable and is set to the value N=5 for test purposes. The execution of the maintenance is confirmed by means of the acknowledgement button.
- A signal column (red, green, yellow, blue, orange) indicates the status of the gate. Red: gate closed; Green: gate open; Orange: gate waiting; Yellow: gate in motion; Blue: maintenance required

Todo: Create a principle sketch according to the description.

Hint: Integrate the conditions of each state (entry, do, exit). Define the transitions between the states Follow the hints in the Ilias Wiki “Basics” “Statechart”. Use “speaking names” to designate the variables and the states to make the program more understandable.

Todo: Create a state machine according to the description.

Todo: Implement the state machine in ST. Use the same variable names as in the state diagram. Use an enumeration for the logic of the program; the enumeration contains the names of the previously defined states.

Hint: Use standard visualization components to simulate the following components: Pushbutton; Signal tower (four LEDs); Motors (one LED each); Counter value of maintenance (text field).

Hint: Use a counter function block for the generation of maintenance intervals. In the visualization, show how many passes are missing until maintenance is done.

CHAPTER

TWENTYONE

TIA SETUP

The setup of Siemens PLC is done with the help of TIA Portal application

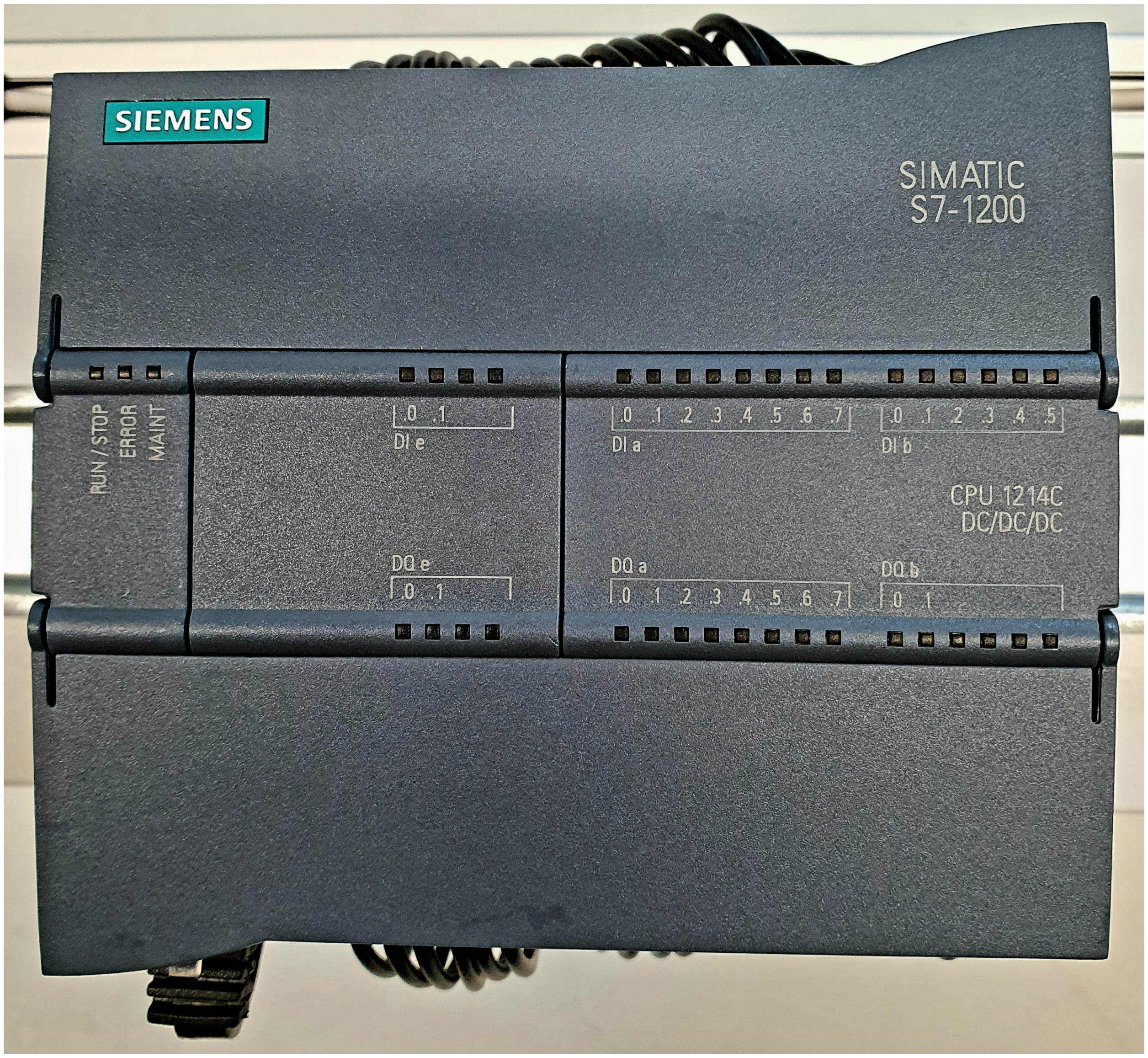


Fig. 21.1: Siemens PLC S1200

TIA Portal is a Siemens software for programming PLCs.

21.1 Creating a new project

To create a new project inside TIA Portal V16, click on `Create new Project` from the start menu and provide name, path in which the project will be saved, and an optional comment.

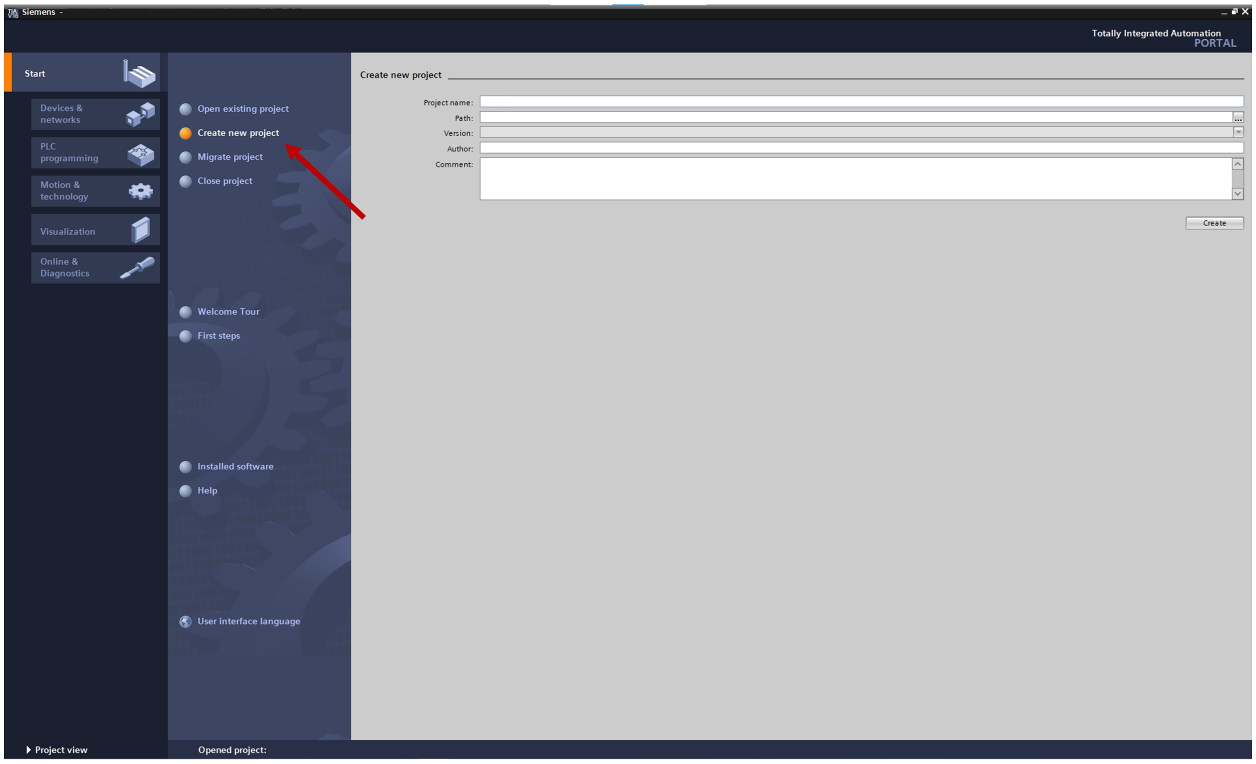


Fig. 21.2: TIA Portal Startup interface.

Following this, guided project creation steps will show up on the screen.

21.2 Adding a PLC to a project

To add a PLC to the current project, first find the model of the PLC. Click on `Configure networks` under `Devices & networks` from the menu.

Once done, follow the steps below to add it (images added for reference):

1. Click on `Configure networks`
2. Go to `Online` -> `Accessible devices...`. A pop-up window will open.
3. Select `PN/IE` inside *Type of the PG/PC interface:*. Check for the PG/PC interface and set it to LAN connection from drop down list. For example, Intel(R) PRO/1000
4. Click on `Start search` to find connected devices to computer's LAN. Once scan is complete, you will see a device in the list with properties like Address, MAC Address etc. Read the IP Address on your PLC and find it in the list. It should be like `192.168.##.##`.

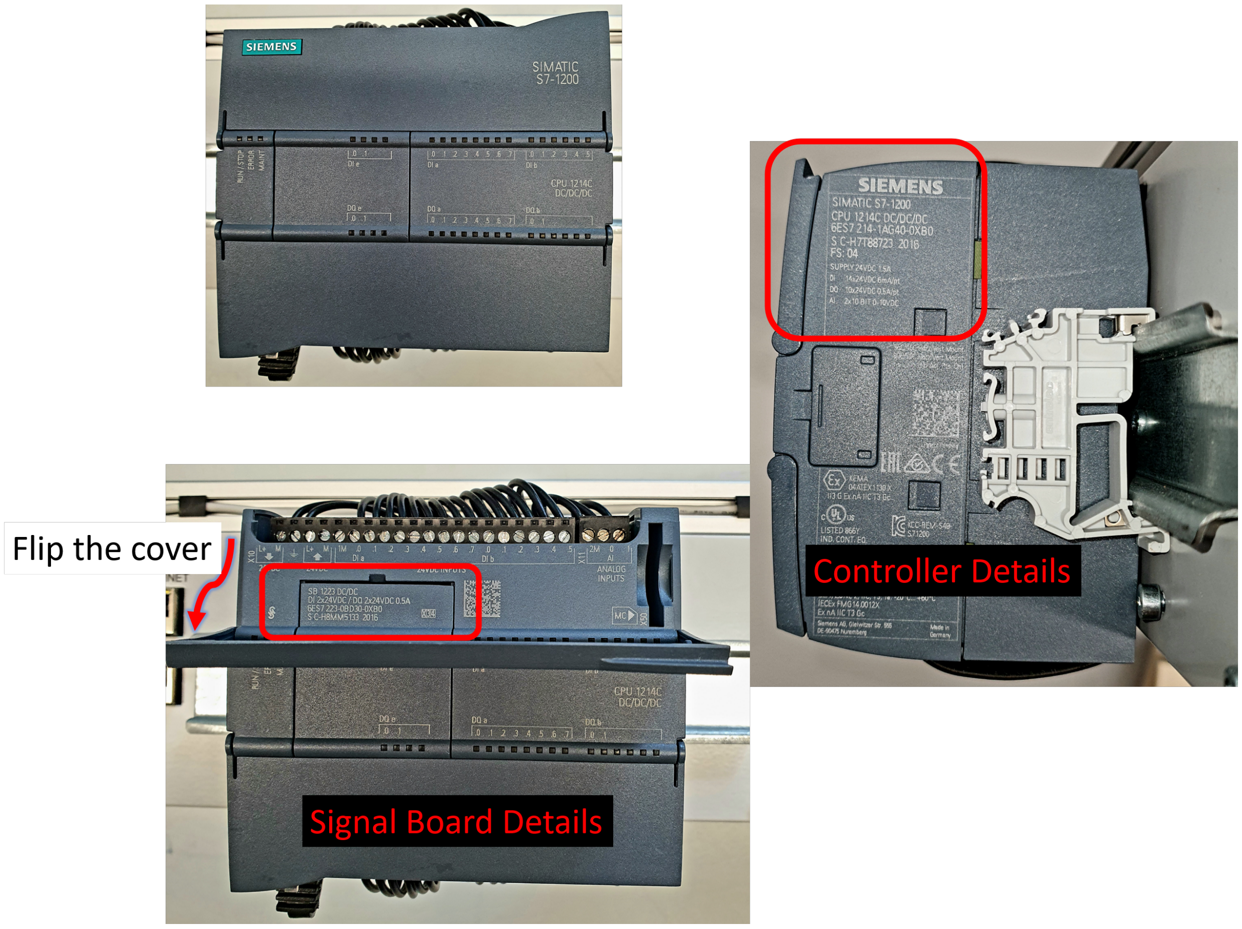
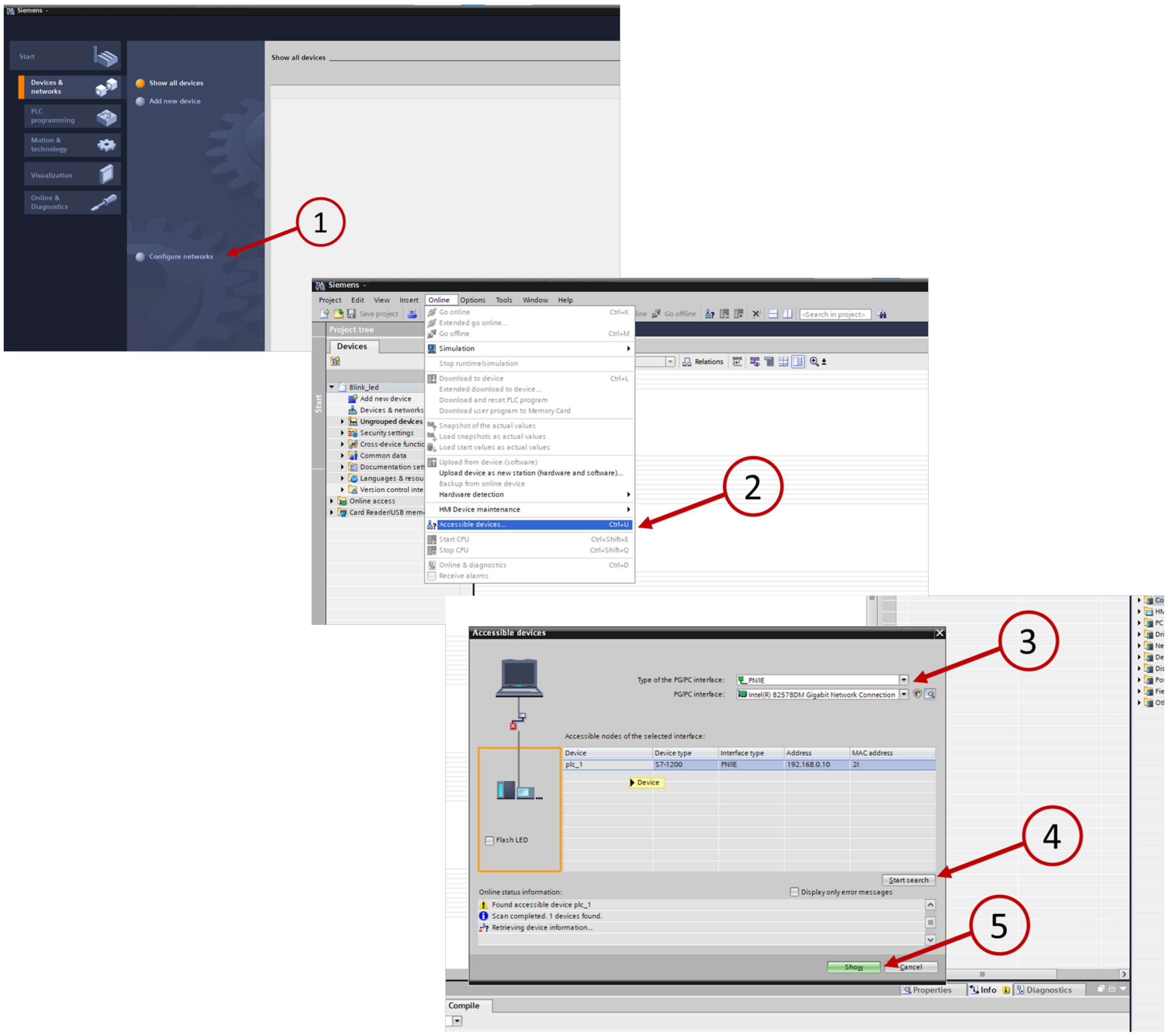


Fig. 21.3: PLC Setup in Project part 1

5. To confirm it's your device, click on the device and press Flash LED. You will then see the Power LED on PLC starts to blink. This confirms that you have selected your own PLC. Next click on `Show`.



Type of the PG/PC interface: PN/IE
PG/PC interface: Intel(R) 82578DM Gigabit Network Connection

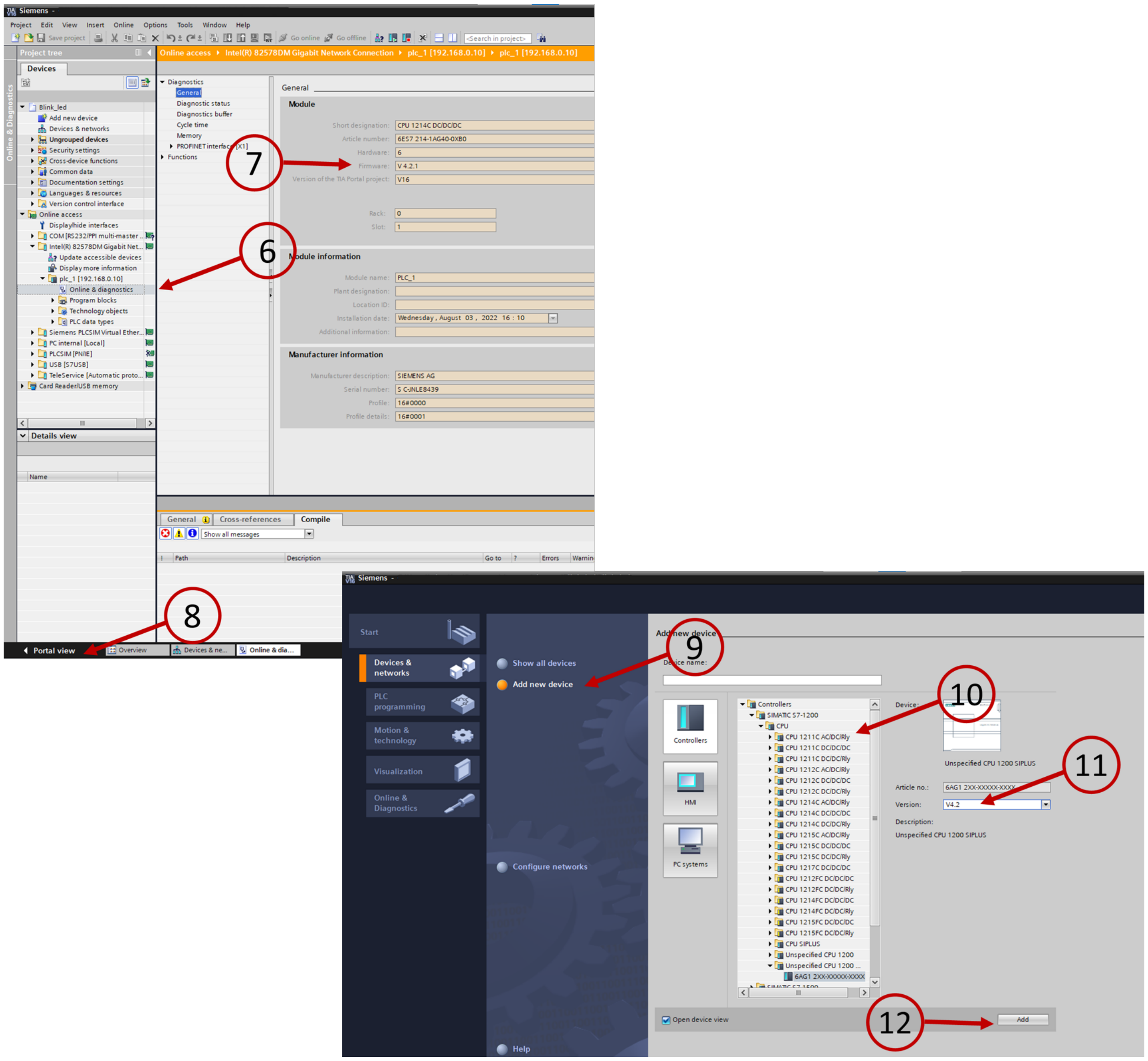
Accessible nodes of the selected interface:

Device	Device type	Interface type	Address	MAC address
plc_1	S7-1200	PN/IE	192.168.0.10	28

Online status information:
- Found accessible device plc_1
- Scan completed. 1 devices found.
- Retrieving device information...

Fig. 21.4: PLC Setup in Project part 1

6. The pop-up window closes and on the left side, a tree list (project tree) structure expands. Here you will find the PLC you selected.
7. Expand the tree of the PLC and double click on `Online & diagnostics`. A new window opens with details about the PLC.
8. Check for the Firmware version and note it down.
9. Click on `Portal view` and return to the `Add new device` panel.
10. Click on `Add new device`
11. Select the correct PLC from the list and use the correct firmware version.
12. Once done, click `Add` and the selected PLC will be added to your project.



General

Module

Short designation: CPU 1214C DC/DC/DC
Article number: 6ES7 214-1AG40-0XB0
Hardware: 6
Firmware: V 4.2.1
Version of the TIA Portal project: V16
Rack: 0
Slot: 1

Module information

Module name: PLC_1
Plant designation:
Location ID:
Installation date: Wednesday , August 03 , 2022 16 : 10
Additional information:

Manufacturer information

Manufacturer description: SIEMENS AG
Serial number: S C-JNLE8439
Profile: 16#0000
Profile details: 16#0001

Add new device

Device: Unspecified CPU 1200 SIPLUS
Article no.: 6AG1 2XX-XXXXX-XXXX
Version: V4.2
Description:
Unspecified CPU 1200 SIPLUS

Fig. 21.5: PLC Setup in Project part 2

21.3 Adding a signal board to PLC

1. A new window will open with image of the selected PLC. Go to `Device view` if not already there.
2. Click on the center of the PLC to add signal board.
3. From the catalogue on the right side, select the correct signal board and drag it on the top of the center of the PLC. You will mostly find the right board in `Signal boards` -> `DI/DQ`.
4. Now your PLC is ready to be programmed.

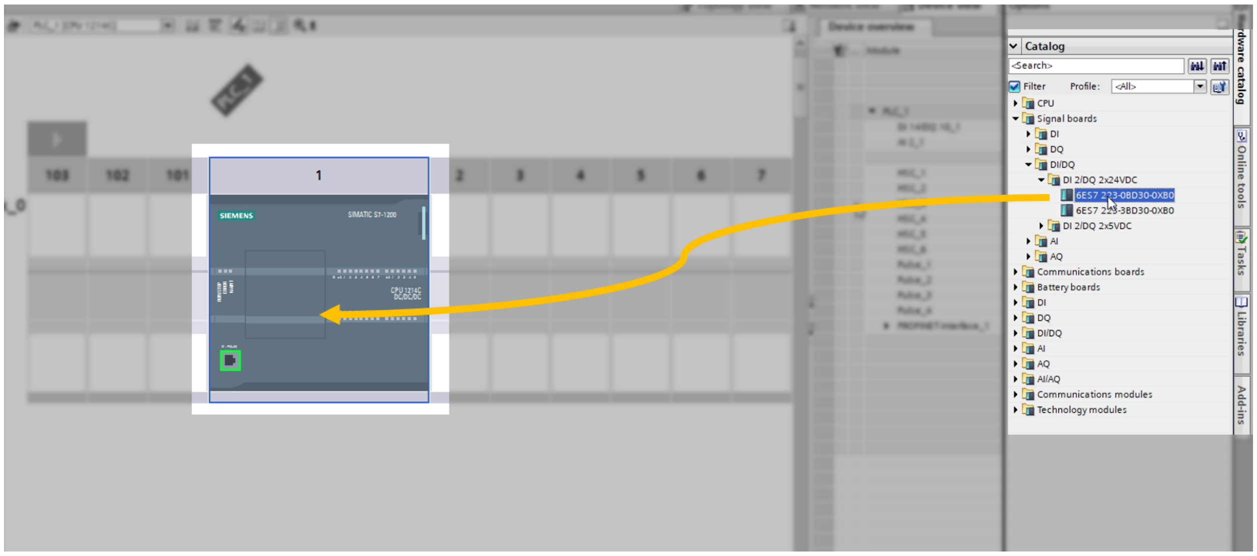


Fig. 21.6: Adding signal board to PLC

If the version is not visible, use the following configuration provided with the PLC.

Once everything is set, the PLC should look like this.

21.4 Changing IP address for Project

To change the IP-Address of the project, follow the steps below:

1. Right click on the PLC from the left side project tree.
2. Click on `Properties`. A pop-up window will open.
3. In that window, find `PROFINET interface` -> `Ethernet addressess` -> `IP protocol`
4. Here, enter the `IP address:` provided with the PLC. Click `OK` once done.

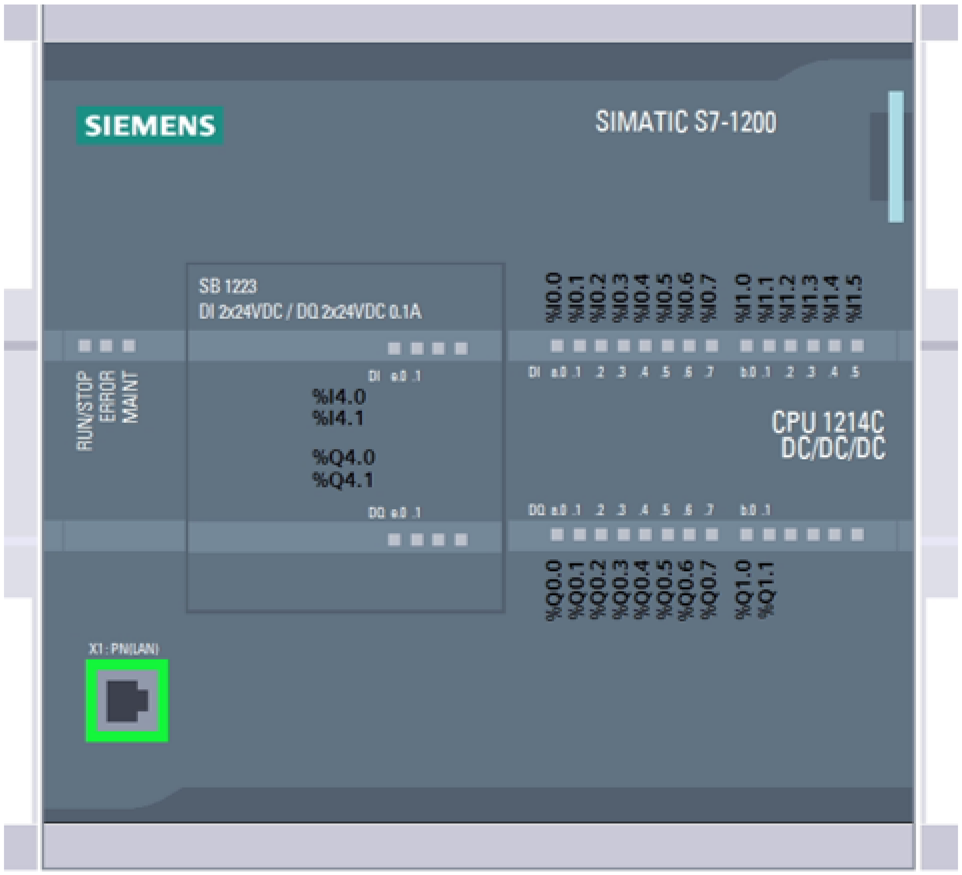


Fig. 21.7: PLC after setup in TIA Portal

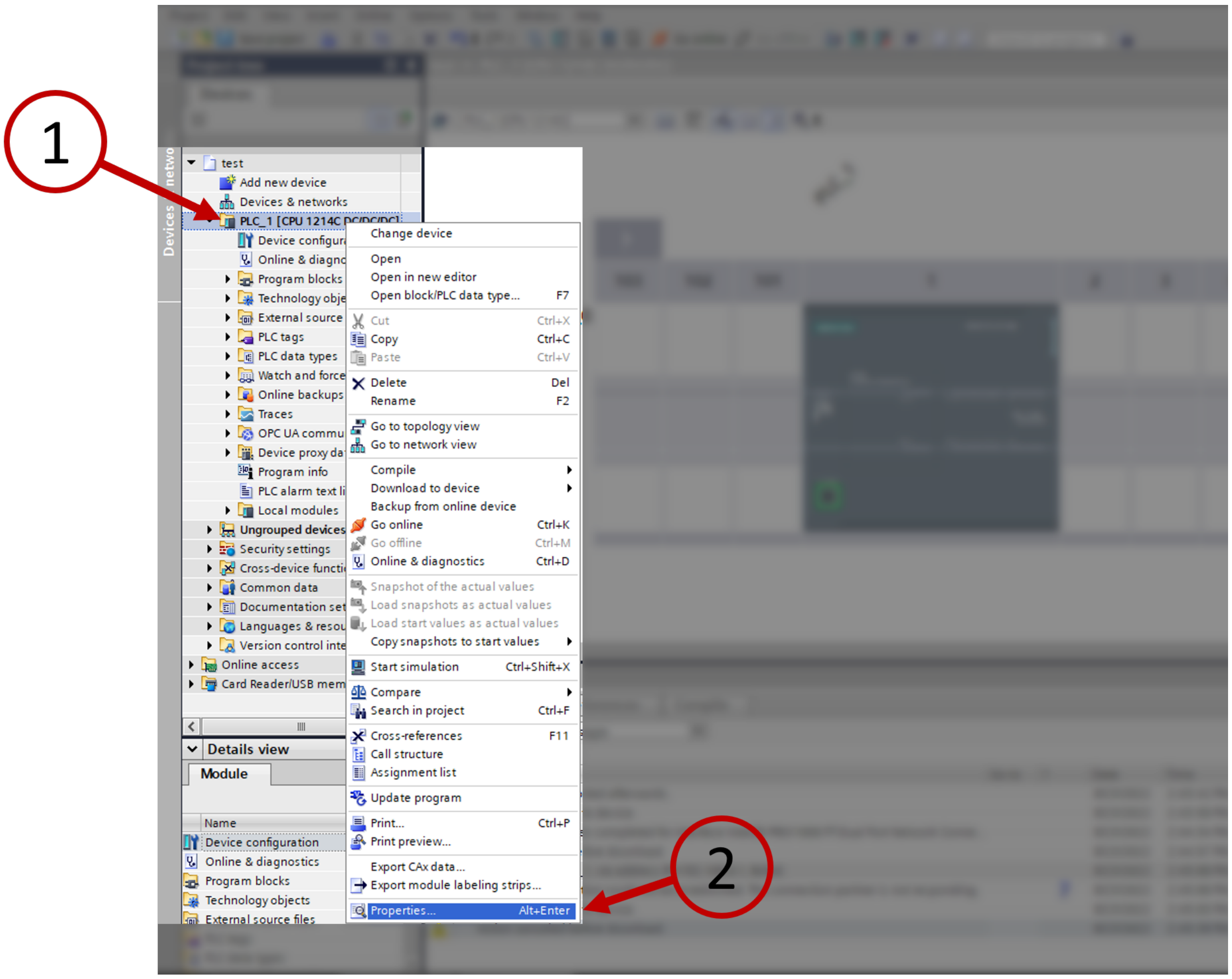
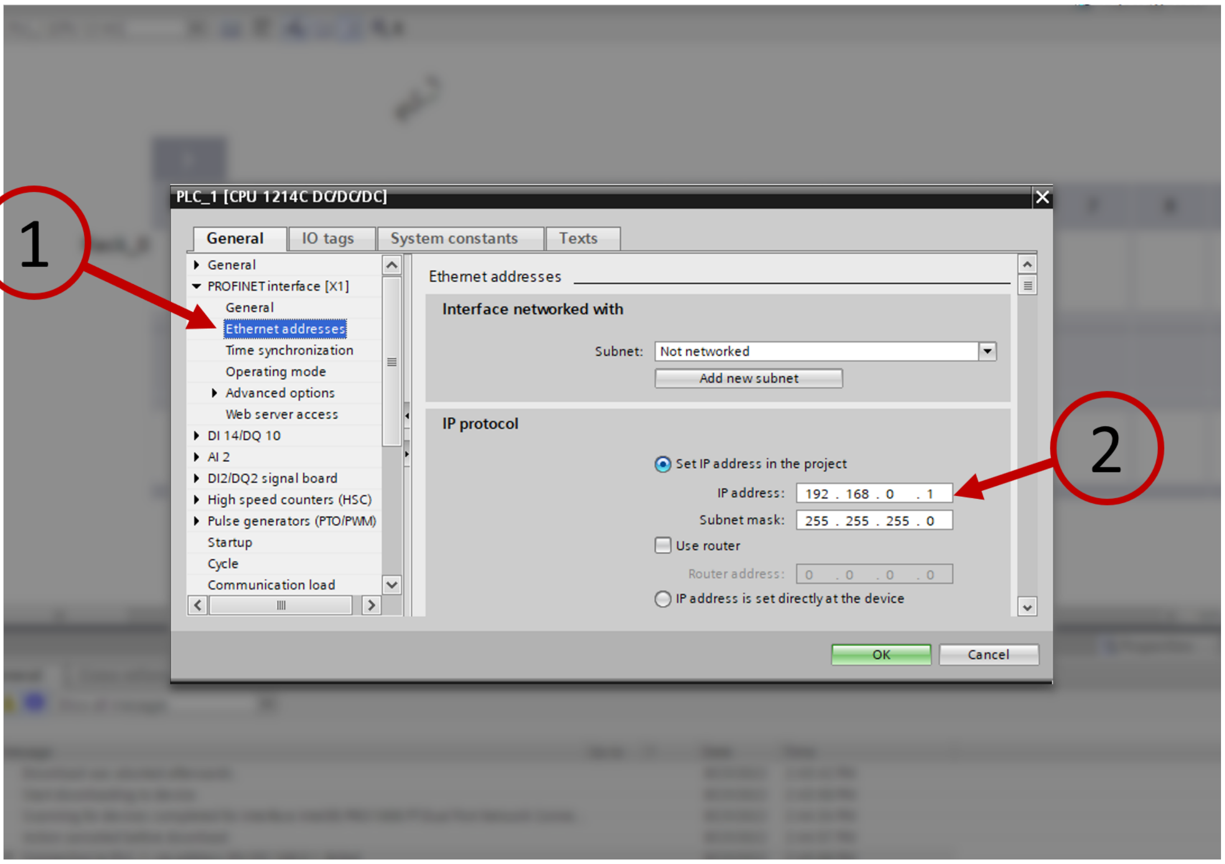


Fig. 21.8: Change IP of project part 1



Ethernet addresses

Interface networked with

Subnet: Not networked

Add new subnet

IP protocol

[✓ Set IP address in the project]

IP address: 192 . 168 . 0 . 1

Subnet mask: 255 . 255 . 255 . 0

Use router

Router address: 0 . 0 . 0 . 0

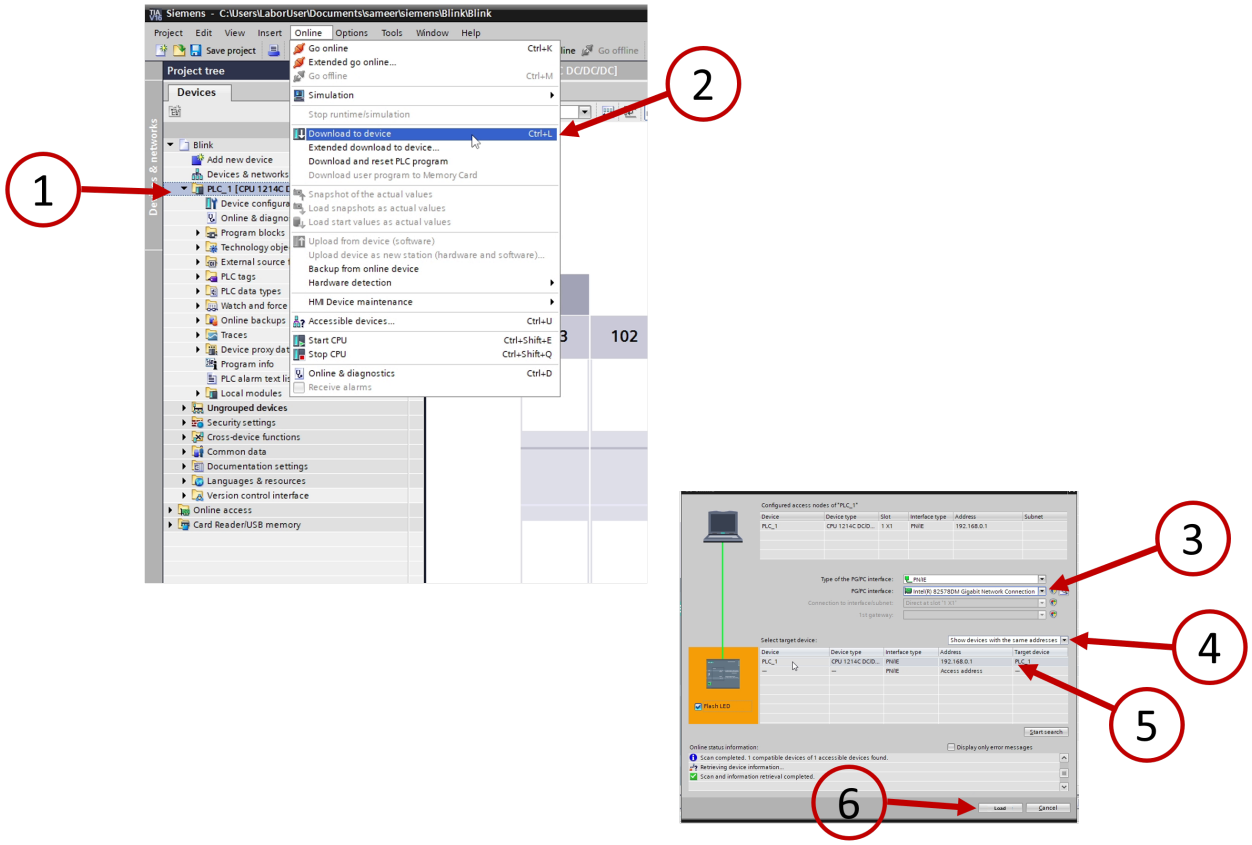
IP address is set directly at the device

Fig. 21.9: Change IP of project part 2

21.5 Uploading code to PLC

Initially you will have to add the plc on your network, to your code. To do that and to upload the code for the 1st time, follow the steps below (images added for reference):

1. To upload code to PLC, select the PLC from the left project tree structure.
2. Select `Online` -> `Download to device` option from the menu. A pop-up window will open.
3. Check for the PG/PC interface and set it to LAN connection from drop down list. For example, Intel(R) 82578DM
4. Click on `Start search` to find connected devices to computer's LAN.
5. Once scan is complete, you will see a device in the list with properties like device name, IP Address etc. To confirm it's your device, click on the device and press Flash LED. You will then see the Power LED on PLC starts to blink. This confirms that you have selected your own PLC.
6. Click on `Load` button. This will compile your code and show the possible errors and warnings. Once the code is checked, this will upload the code to the PLC. After that, click on `Finish` to close the pop-up window.



Configured access nodes of "PLC_1"

Device	Device type	Slot	Interface type	Address	Subnet
PLC_1	CPU 1214C DC/D...	1 X1	PN/IE	192.168.0.1	

Type of the PG/PC interface: PN/IE
PG/PC interface: Intel(R) 82578DM Gigabit Network Connection
Connection to interface/subnet: Direct at slot '1 X1'
1st gateway:

Select target device: Show devices with the same addresses

Device	Device type	Interface type	Address	Target device
PLC_1	CPU 1214C DC/D...	PN/IE	192.168.0.1	PLC_1
—	—	PN/IE	Access address	—

Online status information:
Scan completed. 1 compatible devices of 1 accessible devices found.
Retrieving device information...
Scan and information retrieval completed.

Fig. 21.10: Uploading code to PLC for 1st time

Note: Keep the PLC connected with LAN cable.

- When uploading any hardware changes, keep the PLC in `STOP` mode and in offline mode. i.e., `Online` -> `Stop CPU` & `Online` -> `Go offline`

- When uploading only software changes, you can keep PLC in Online mode.

Note: When uploading to a Virtual PLC, switch to `Siemens PLCSIM Virtual Ethernet Adapter` in #Step 3. Rest of the steps remain same.

After uploading for the 1st time, you can now upload only software changes until you don't have any new hardware configuration or hardware property changes.

To do so, right click on the main code block and click on `Download to device -> Software (only changes)`

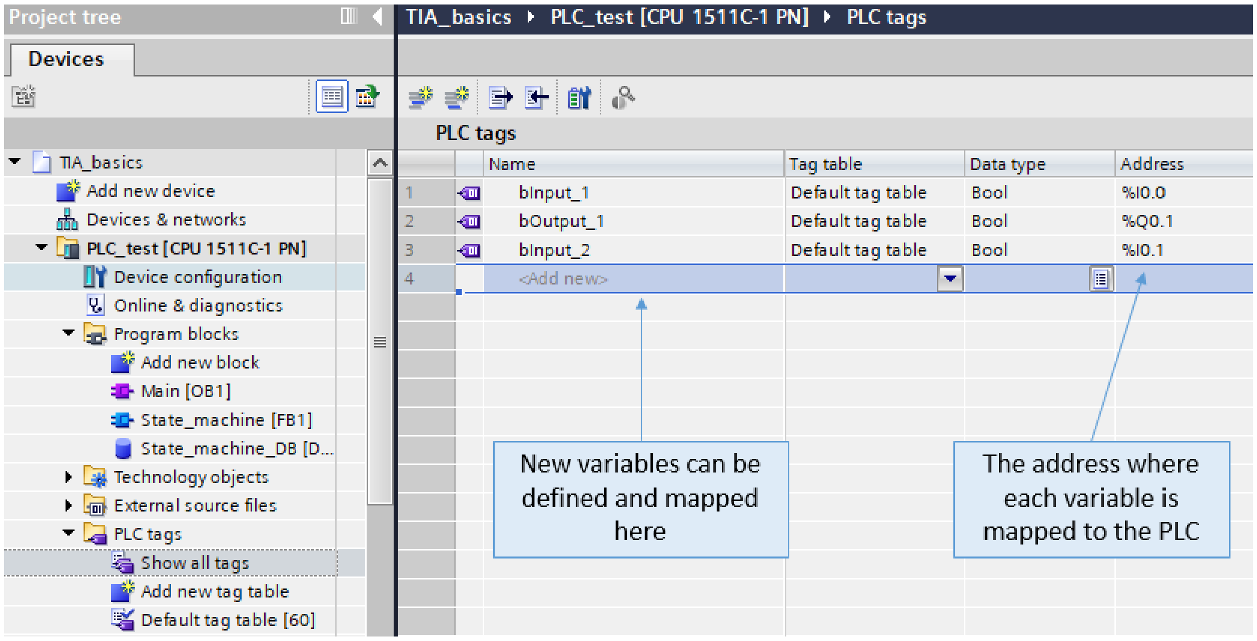
If you have hardware changes or property changes, right click on PLC name from the left project tree. Click on `Download to device -> Hardware and software (only changes)`. You might have to Go offline and then Online to see the changes.

21.6 Creating PLC tags for Pins

`PLC tags` are the PLC variables. These particular variables are mapped to the PLC I/O modules. Tag tables are used to view the PLC tags. One can add new tag tables to sum up variables that are related together. When a variable needs to be connected to a physical signal that is outside the PLC (e.g., a sensor reading coming in), it is defined in a PLC tag table.

To define project tags,

1. Go to the left side project tree. Expand the PLC and expand PLC tags.
2. Under this, click on `Add new tag table`. Rename the new table with a proper name for example, out_pins.
3. Double click on the table name, a new pop-up window should open. Here you define variable names for the pins you wish to use in your programs. A sample list is shown below.



PLC tags

	Name	Tag table	Data type	Address
1	bInput_1	Default tag table	Bool	%I0.0
2	bOutput_1	Default tag table	Bool	%Q0.1
3	bInput_2	Default tag table	Bool	%I0.1
4	<Add new>			

Fig. 21.11: Defining PLC Tag

Note: This section supports features like drag to add sequential list like MS Excel.

You can also import & export this tag table from & to MS Excel or other spreadsheet application.

Note:

You will see the following in Address column

- I -> Input pin
- Q -> Output pin
- M -> Memory location

When you add variables for input/output pins of PLC, they also replace on PLC in `Device View`.

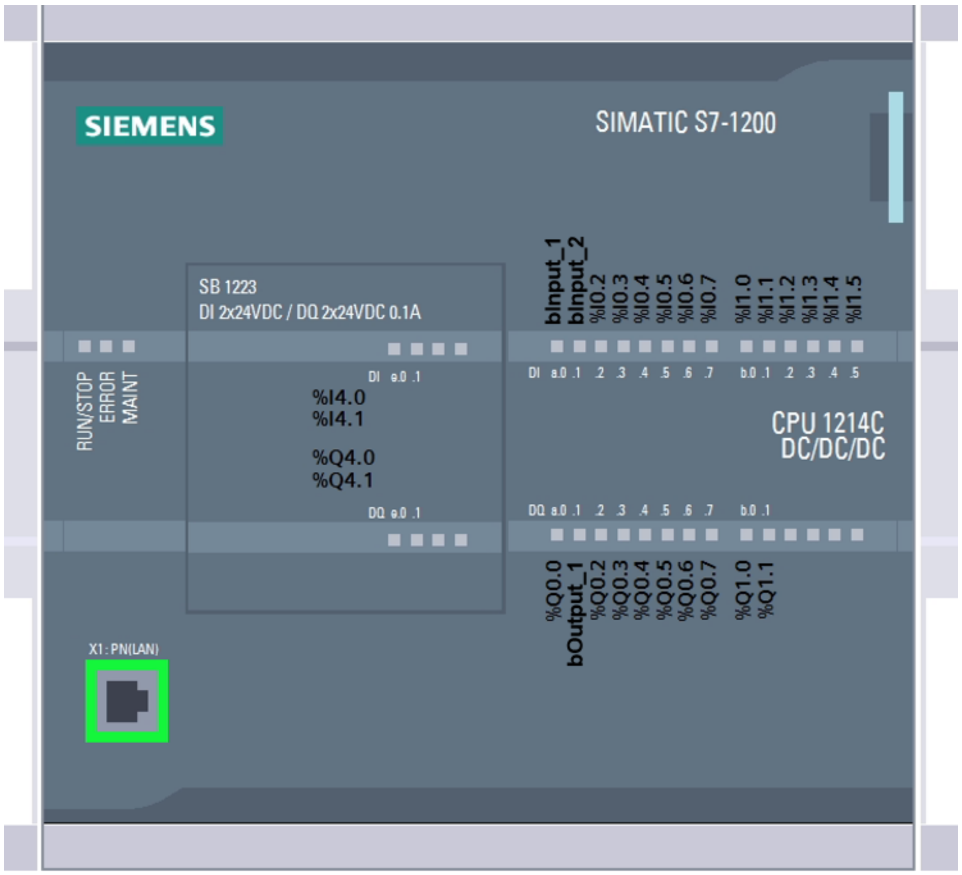


Fig. 21.12: Device view after variables to Tag Table

CHAPTER

TWENTYTWO

UML BASICS

22.1 UML Notation Elements

The following chapter contains excerpts from the book “UML 2 kompakt” by Heide Balzert. For a complete view of the UML notation you must refer to the book.

22.1.1 Object

In the object-oriented software development an object possesses a certain condition and reacts with a defiierten behavior to its environment. In addition, each object has an identity that distinguishes it from all other objects.

The state of an object includes the attributes or their current values and the respective object relationships to other objects. Attributes are inherent, unchanging characteristics of the object, while attribute values may be subject to change.

The behavior of an object is described by a set of operations. A change or a query of the state is only possible by means of the operations.

The object is represented in UML as a rectangle, which can be divided into two fields. In the upper field the object is designated as follows:

:Class
    if the object is anonymous, only the class name is specified.

object:Class
    if the object is to be addressed by a name.

object
    if the object name is sufficient to identify the object and the name of the class is evident from the context.

The name of the object is always underlined. Object names start with a lowercase letter in UML, class names with an uppercase letter. Anonymous objects are used if it is any object of the class. Object names are used to name a specific object of the class for the system analyst.

In the lower field - optionally - the relevant attributes of the object in the respective context are entered. The UML allows the following alternatives:

attribute
    [type = value] .

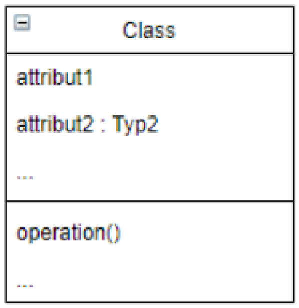
attribute = value
    recommended, since the value can often be used to identify what type it is.

attribute
    useful if the value of the attribute is not of interest.

22.1.2 Class

A class defines for a collection of objects their structure (attributes), behavior (operations) and relationships (associations and generalization structures).

The class name is a noun in the singular. Thus, it describes a single object of the class. Examples: Employee, Car, Customer.



22.2 Use Case Diagram

A **use case** describes the functionality of the software system that an actor must perform to obtain a desired result or to achieve a goal. Use cases should allow to talk to the future user about the functionality of the software system without getting lost in details right away.

An **actor** is a role played by a user of the software system. Actors can be humans or other automated systems. They are always external to the system.

A **use case diagram** gives a good overview of the software system and its interfaces to the environment at a high level of abstraction. The actors are often entered as stick figures, but can also be represented by a rectangle or pictogram (e.g., as a computer symbol).

A **line** between actor and use case means that communication is taking place. The system under consideration is modeled as a large rectangle that includes all use cases.

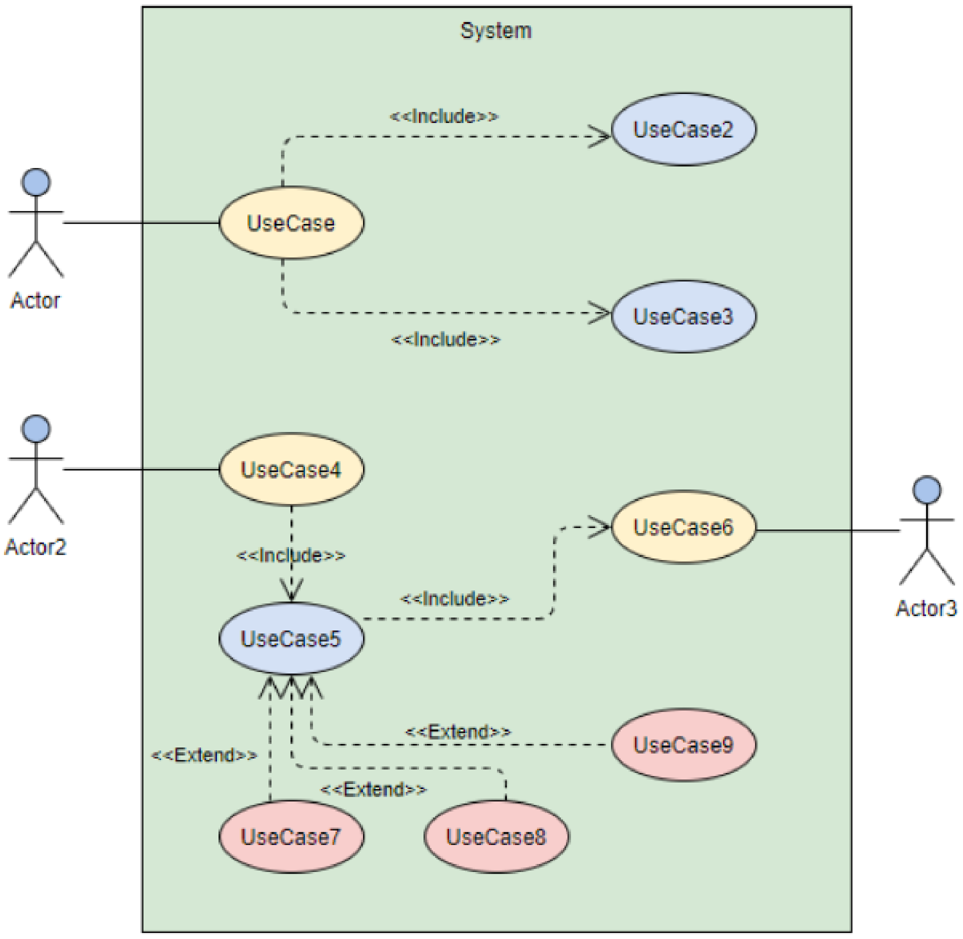
With the help of the **extend relationship**, a use case A is extended by a use case B. The use case A describes the basic functionality, the use case B specifies extensions. Use case A can be executed alone or together with the extensions. For an extension to be inserted, a condition must be met. This condition can be specified as a note or comment if required and appended to the extend relationship.

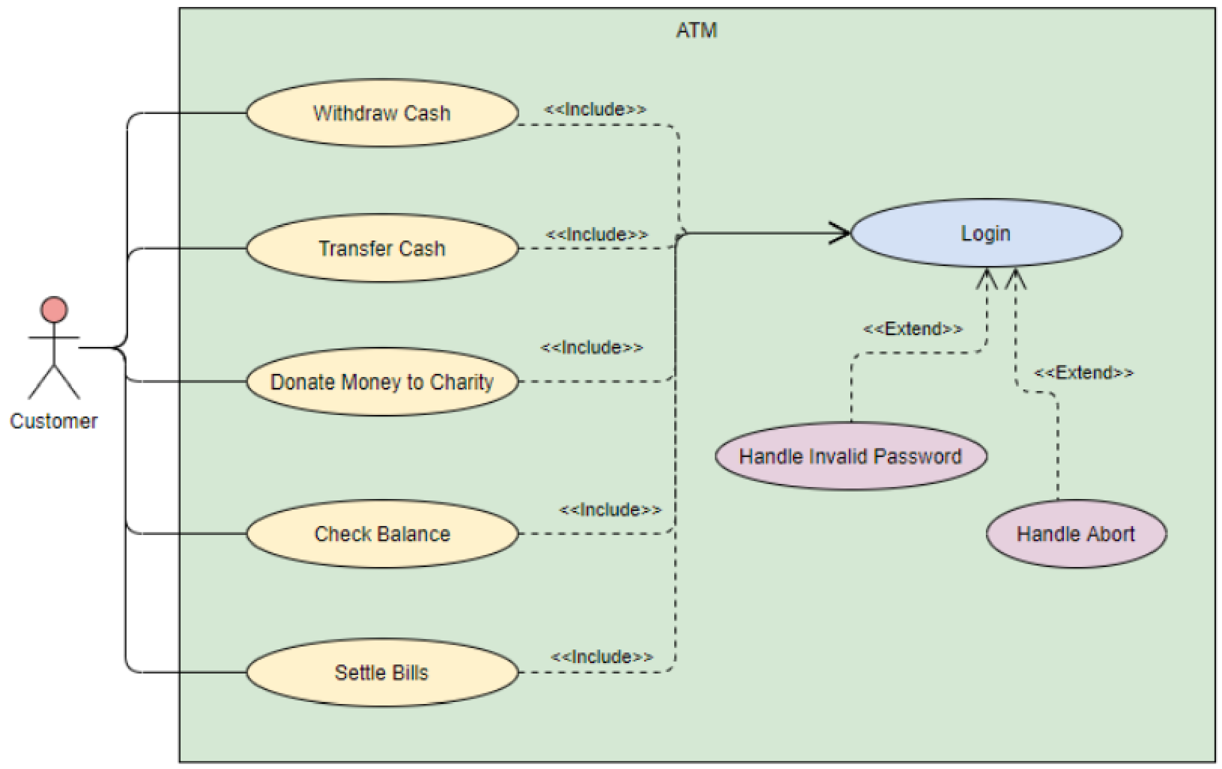
The **include relationship** allows the common functionality of use-cases A and B to be described by a use-case C. The use case C is not optional, but is always required for the correct execution of A and B. The include relationship eliminates the need to describe the same behavior multiple times. In contrast to the extend relationship, the execution of use case C is not dependent on any condition.

Use case diagrams are created per use case. A visual connection of multiple use cases is not typical, here individual diagrams are used.

To represent boundaries, e.g. of devices, packages can be used.

Hint: Use case diagrams can be created online: https://app.diagrams.net/


System
<<Include>>
UseCase2
Actor
UseCase
UseCase3
<<Include>>
Actor2
UseCase4
UseCase6
Actor3
<<Include>>
<<Include>>
UseCase5
<<Extend>>
UseCase9
<<Extend>>
<<Extend>>
UseCase7
UseCase8



22.3 Sequence Diagram

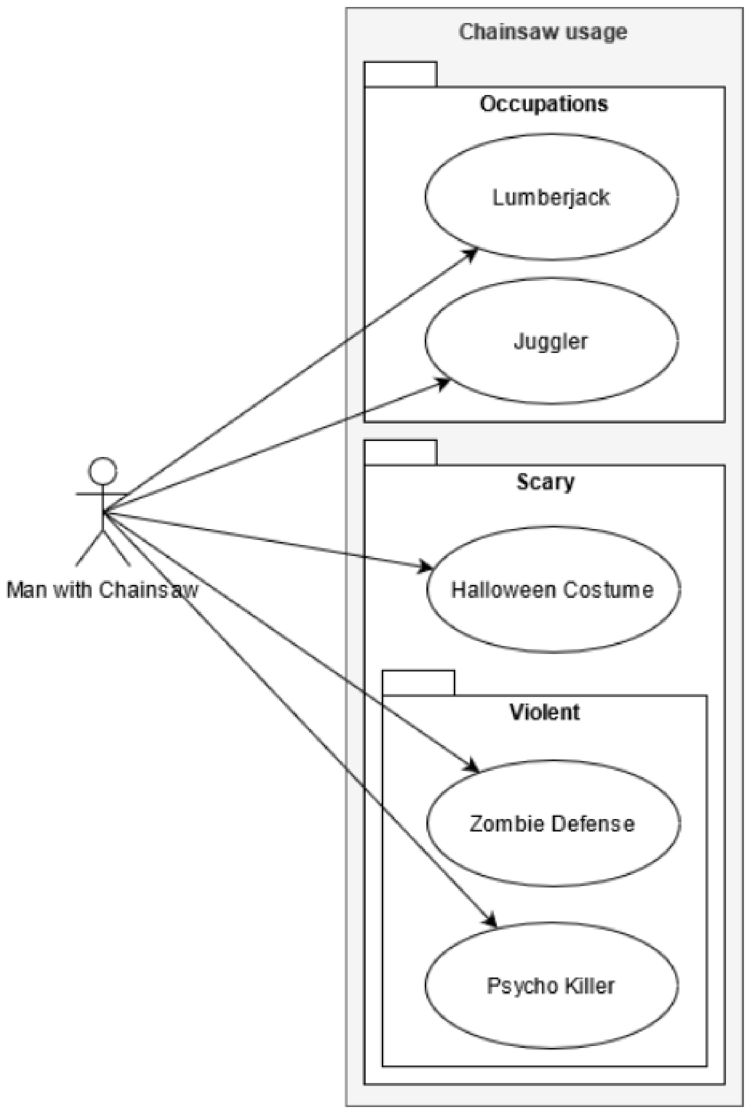
A sequence diagram shows the interaction between several communication partners. Each communication partner is represented by a rectangle (header) with a line (lifeline), which can be dashed and represents the lifetime of the communication partner.

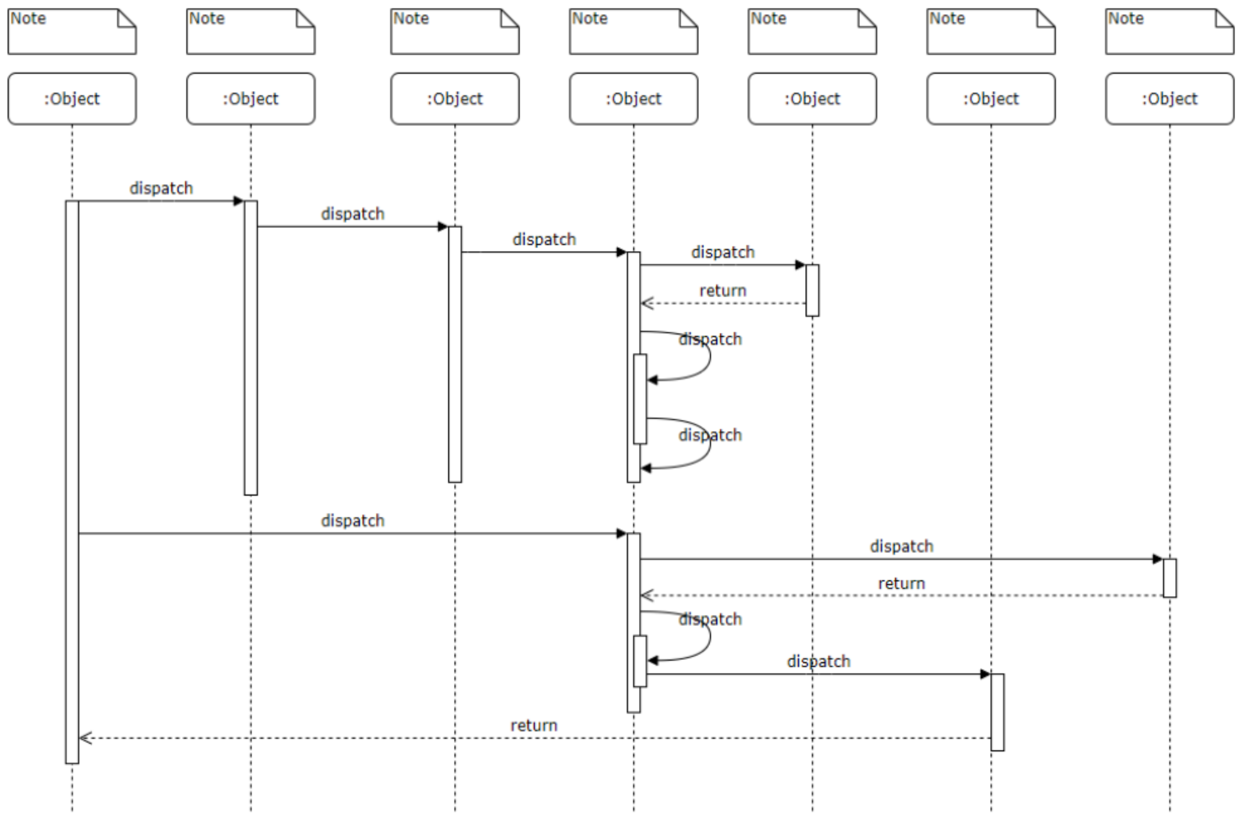
The interaction between communication partners can be done by a synchronous or an asynchronous message. In the **synchronous message**, the sender waits until the receiver has completed the requested processing. The receiver sends a response message to the sender, which implicitly communicates the end of the requested processing and may also contain response data.

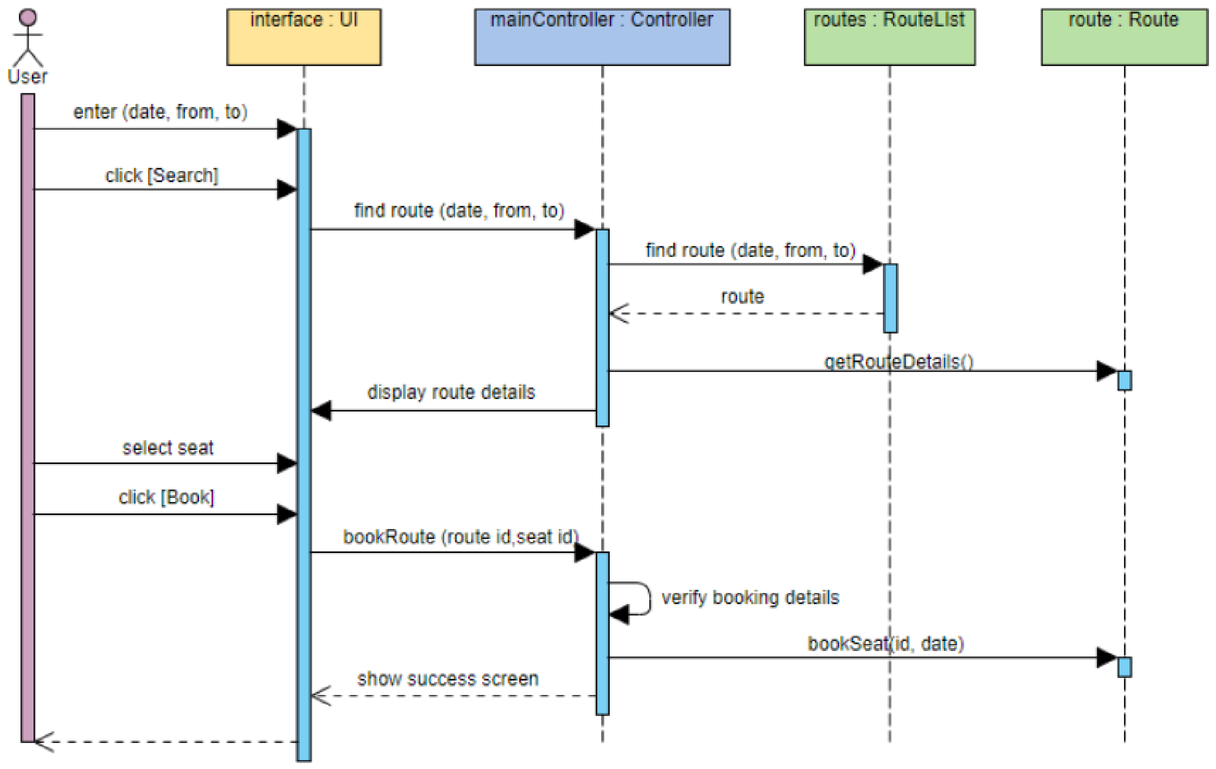
In the **asynchronous message**, the sender does not wait for the receiver to complete the processing, but continues its own processing in parallel.

Synchronous messages are indicated by an arrow with a filled arrow spit, and asynchronous messages are indicated by an arrow with an open arrow spit. The reply of a synchronous message is a dashed arrow. The amount of time it takes for a communication partner to perform the requested processing can be modeled by a bar on the dashed line.

Hint: Sequence diagrams can be created online: https://sequencediagram.org/


Chainsaw usage
Occupations
Lumberjack
Juggler
Scary
Halloween Costume
Violent
Zombie Defense
Psycho Killer
Man with Chainsaw


Note
:Object
dispatch
return
dispatch
return
dispatch
return


User
interface : UI
mainController : Controller
routes : RouteList
route : Route
enter (date, from, to)
click [Search]
find route (date, from, to)
find route (date, from, to)
route
getRouteDetails()
display route details
select seat
click [Book]
bookRoute (route id,seat id)
verify booking details
bookSeat(id, date)
show success screen

22.4 State Machine Diagram

22.4.1 Procedure & Checklist

Brainstorming

Create a table with the following columns:

- 1st column: All states,
- 2nd column: All events that can occur internally or externally,
- 3rd column: All processing steps that must be listed.

Which states does the automat receive?

- The starting point is the initial state.
- By which events is a state left?
- Which subsequent states occur?
- By what is the state defined (attribute values, object relations)?

Does the state machine need a final state?

- If the final state is reached, the processing of the state machine ends.
- If the state machine describes a life cycle, the termination of the state machine can be equated with the end of life of the object.
- No processing may be performed in a final state; it may not have any output arrows.

Which activities are to be modeled?

- Is there any processing associated with a state diagram?
- Do all incoming transitions of a state have the same activity? –> entry activity
- Do all outgoing transitions of a state possess the same activity? –> exit activity
- Is the processing linked to the duration of the state? –> do-activity

Which events are to be modeled?

- External events from the user or from other objects;
- Temporal events (duration, time);
- Internally generated events of the class or use case.

Analytical steps - Appropriate state name.

- Describes a specific time period.
- Does not contain a verb.

Analytical steps - Are all transitions correctly entered?

- Is every state reachable?
- Can each state - except the final state - be exited?
- Are the events of the transition unique?
- Can events occur that are not covered by the specified events?

Source: “UML 2 compact with checklists”, Heide Balzert

22.4.2 Further

Hint: State diagrams can be created online on draw.io (https://app.diagrams.net/).

Note: In the literature there are different uses of the "entry", "do" and "exit" conditions. We use the following notatin:

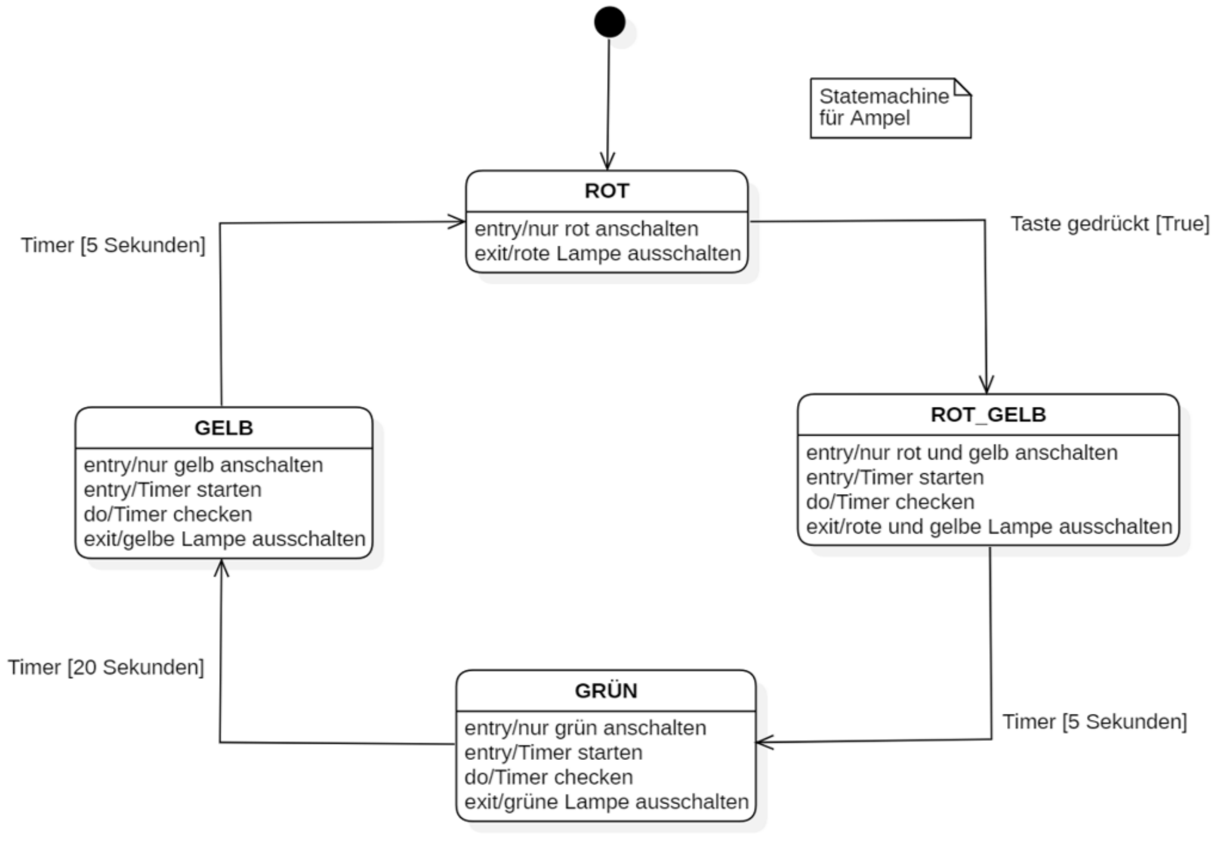
- Entry activity: action which is triggered when entering the state, e.g. `light = False`
- Do activity: action which is executed as long as you are in this state, e.g. count sheep
- Exit activity: action which is executed when one leaves the state, e.g. `sleep = True`

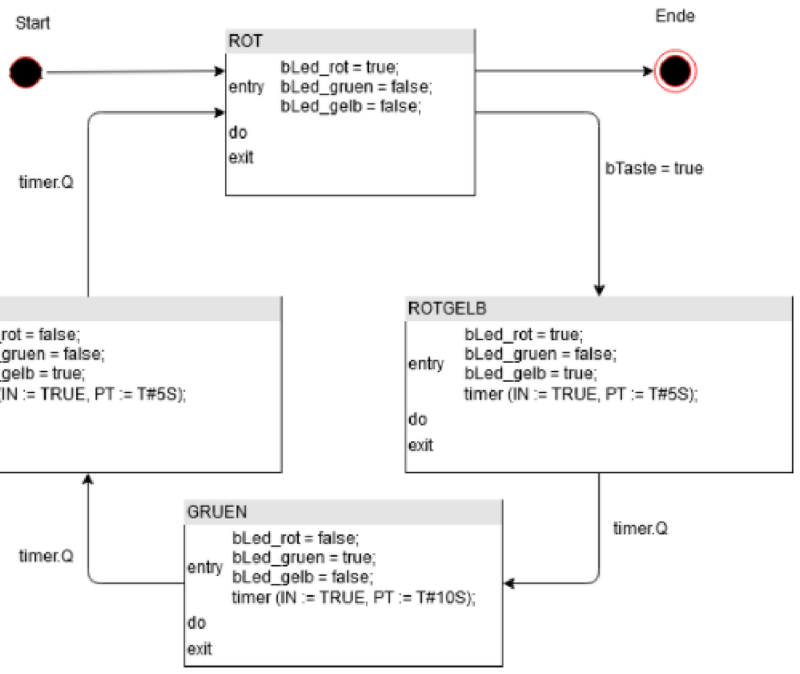
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/

https://youtu.be/hbJ48fbJbrQ

22.4.3 Examples for state diagrams

Example for the circuit of a person traffic light

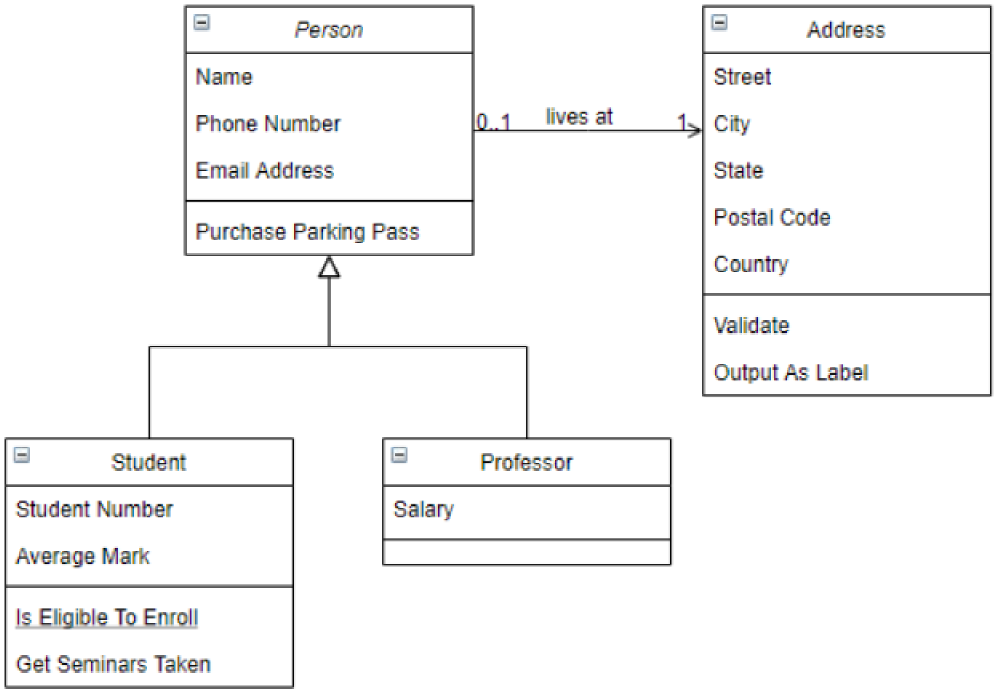


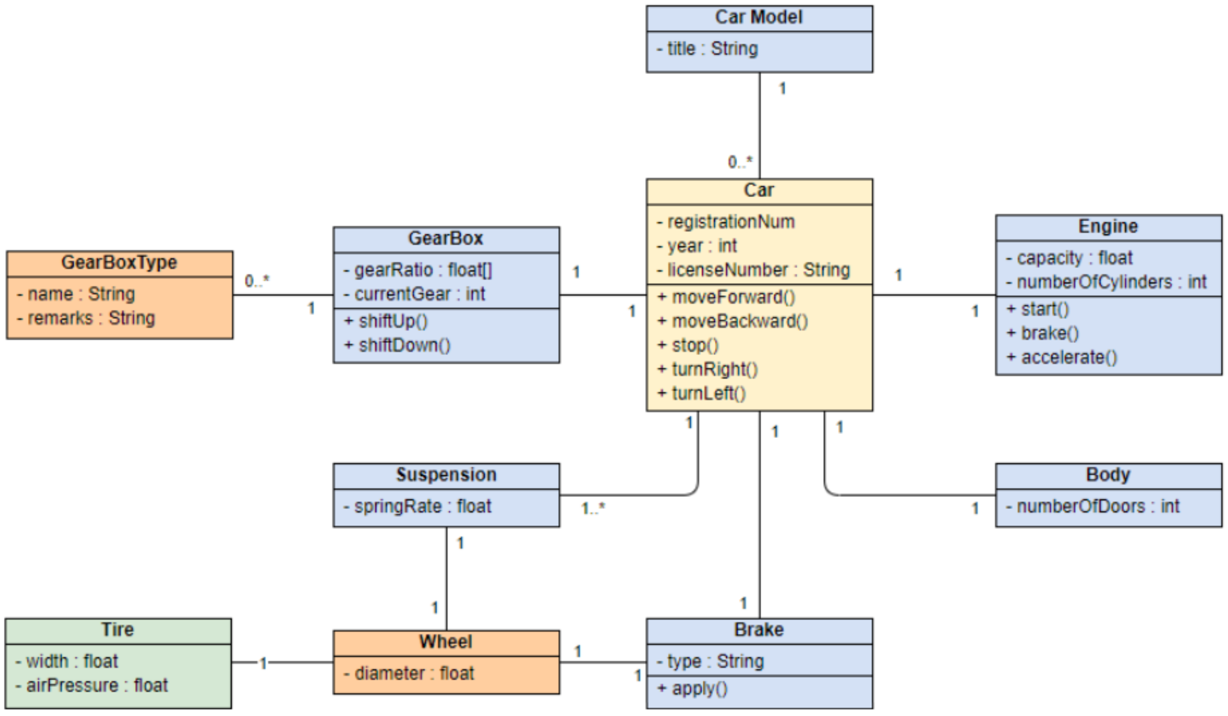


22.5 Class Diagram

The class diagram represents the classes with attributes and operations, generalization and associations between classes.

Hint: Class diagrams can be created online: https://app.diagrams.net/


Person
Name
Phone Number
Email Address
Purchase Parking Pass
0..1
lives at
1
Address
Street
City
State
Postal Code
Country
Validate
Output As Label
Student
Student Number
Average Mark
Is Eligible To Enroll
Get Seminars Taken
Professor
Salary


Car Model
- title : String
1
0..*
Car
- registrationNum
- year : int
- licenseNumber : String
+ moveForward()
+ moveBackward()
+ stop()
+ turnRight()
+ turnLeft()
GearBoxType
- name : String
- remarks : String
0..*
1
GearBox
- gearRatio : float[]
- currentGear : int
+ shiftUp()
+ shiftDown()
1
1
1
1
Engine
- capacity : float
- numberOfCylinders : int
+ start()
+ brake()
+ accelerate()
1
1
1
Suspension
- springRate : float
1..*
Body
- numberOfDoors : int
1
1
1
1
Tire
- width : float
- airPressure : float
1
Wheel
- diameter : float
1
1
Brake
- type : String
+ apply()

CHAPTER

TWENTYTHREE

OPC-UA CLIENT

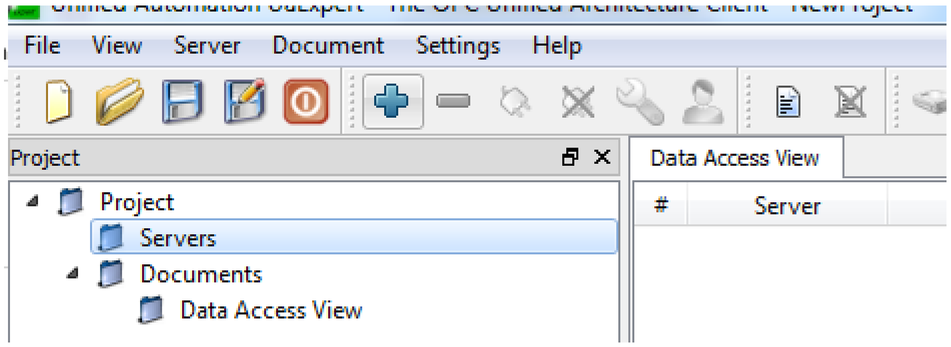
23.1 UA Expert

There are a lot of different OPC tools available. In this project the free to use tool `UA Expert` from Unified Automation GmbH is selected.

Hint: UA-Expert creates SSL certificates on the first startup. The information UA-Expert needs to know are relevant to fulfill the SSL-specifications and they are used to identify the owner of the certificate. These informations are - in our case - not important. You can add anything to the required fields.

Warning: UA-Expert requires the IP-address and port of the opc-ua server.

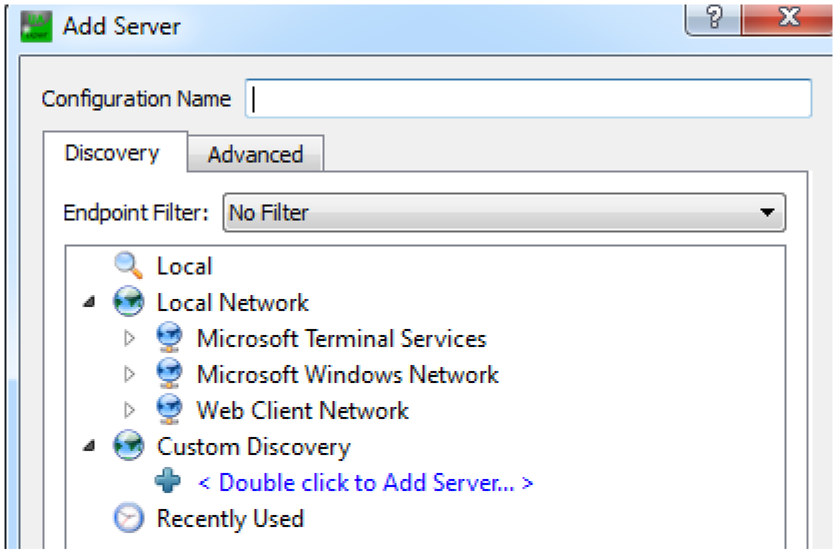
Start the tool and press the + Button to add a new connection. Use the "Double click to Add a Server…".



Fill in the field with the link to the OPC-UA Server (i.e. the IP Address of the PLC). Next, browse to the tree behind the IP-address down and accept the dialog. Open the connection with the connect button.

Hint: A PLC might need to be in running mode for the OPC-UA server to be enabled.

Now you can browse through your OPC-UA `Objects` and search for the variables your OPC-UA server exposes. It is possible to subscribe to data changes of variables by dragging them into the center window.

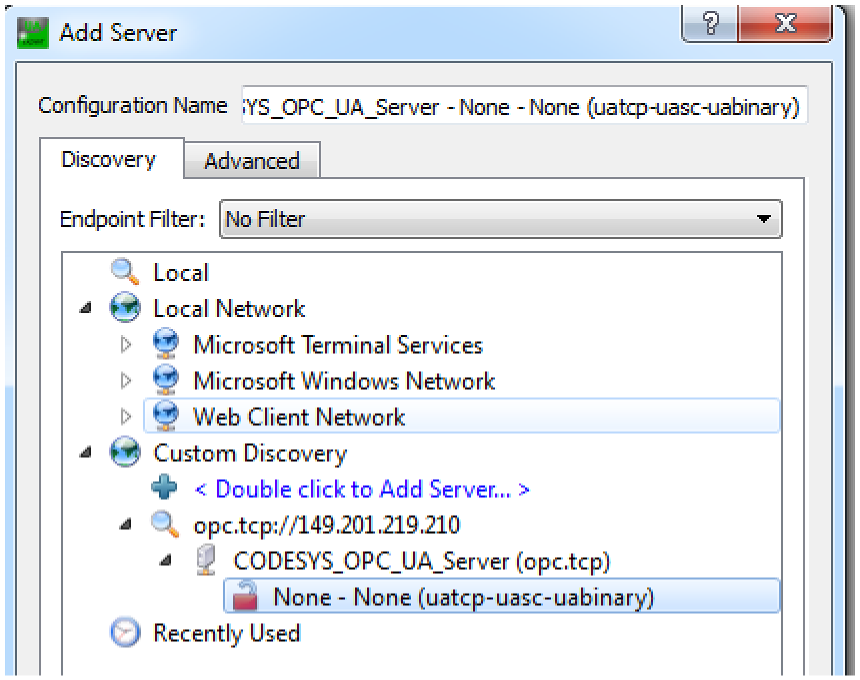

Add Server

Configuration Name: |

Endpoint Filter: No Filter

Discovery
Advanced
Local
Local Network
Microsoft Terminal Services
Microsoft Windows Network
Web Client Network
Custom Discovery
< Double click to Add Server... >
Recently Used

Fig. 23.1: Add a new connection and a new server


Add Server

Configuration Name: ;YS_OPC_UA_Server - None - None (uatcp-uasc-uabinary)

Endpoint Filter: No Filter

Discovery
Advanced
Local
Local Network
Microsoft Terminal Services
Microsoft Windows Network
Web Client Network
Custom Discovery
< Double click to Add Server... >
opc.tcp://149.201.219.210
CODESYS_OPC_UA_Server (opc.tcp)
None - None (uatcp-uasc-uabinary)
Recently Used

Fig. 23.2: OPC-UA connection setup

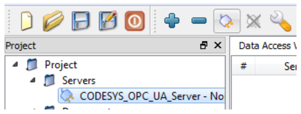
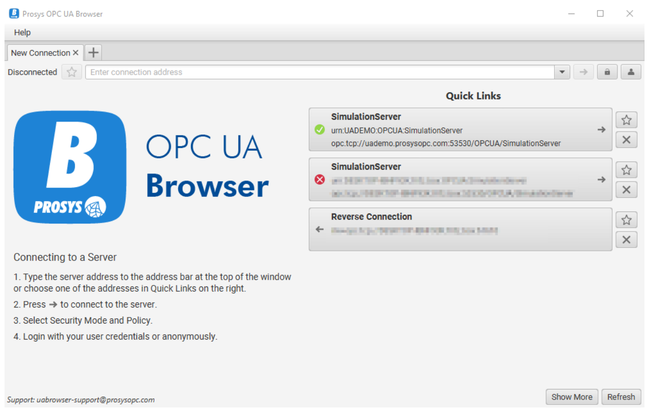


Fig. 23.3: Connection to server

23.2 Prosys OPC-UA Browser

There are a lot of different OPC tools available. In this project the free to use tool Prosys OPC UA Browser from Prosys OPC Ltd is selected.

Warning: Prosys OPC-UA Browser requires the IP-address and port of the opc-ua server.



SimulationServer
urn:UADEMO:OPCUA:SimulationServer
opc.tcp://uademo.prosysopc.com:53530/OPCUA/SimulationServer

SimulationServer
[redacted]

Reverse Connection
[redacted]

Connecting to a Server

1. Type the server address to the address bar at the top of the window or choose one of the addresses in Quick Links on the right.
2. Press → to connect to the server.
3. Select Security Mode and Policy.
4. Login with your user credentials or anonymously.

Support: uabrowser-support@prosysopc.com

Start the tool and select the field `Enter connection address`. Fill in the address and port to the OPC-UA Server (i.e. the IP Address of the PLC and the default Port 4840). Next, click on the arrow and open the connection to the OPC-UA server.

Hint: A PLC might need to be in running mode for the OPC-UA server to be enabled.

Now you can browse through your OPC-UA `Objects` and search for the variables your OPC-UA server exposes. It is also possible to subscribe to data changes of variables.