1501431 Intelligent Control Systems

Course Description:
Fundamental theories and mathematics for analyzing and designing a control system, PID control, sensors and devices in control, Novel principle of artificial intelligence and its applications in control systems, Industrial control system*, Precision control in automation*, Practical AI in industrial control*.
(*modified in the framework of an Erasmus + project: Asean Factori 4.0 Across South East Asian Nations: From Automation and Control Training to the Overall Roll-out of Industry 4.0 609854-EPP-1-2019-1-FR-EPPKA2-CBHE-JP)

Learning outcome:
1. Students can discuss the content of intelligent control system.
2. Students can analyze the behavior of intelligent control system.
3. Students understand the function of industrial control system.

Lecturer:
Assoc. Prof. Punnarumol Temdee, Ph.D.
Asst. Prof. Roungsan Chaisricharoen, Ph.D.
Asst. Prof. Santichai Wicha, Ph.D.
Lect. Chayapol Kamyod, Ph.D.

Credit: 3(2-2)
Lecture: 30 Hours (20 hours of modified content)
Lab: 30 Hours (20 hours of modified content)

Assessments:

Attendance	10%
HW/CW	20%
Midterm	25%
Final	25%
Project	20%

Lecture (seminar):

Content	Hours
Introduction to control engineering	2
Analog control system	4
Digital control system	4
Control system for industrial*	4
Speed and precision of a system*	4
Intelligent algorithms for control system*	4
Application of industrial control*	4
Plant simulation and emulation*	4

(*modified in the framework of an Erasmus + project: Asean Factori 4.0 Across South East Asian Nations: From Automation and Control Training to the Overall Roll-out of Industry 4.0 609854-EPP-1-2019-1-FR-EPPKA2-CBHE-JP)

Lab (internship):

Content	Hours
Software for control engineering	2
Model of analog control system	4
Model of digital control system	4
Analog I/O of PLC*	2
Analog I/O of emulation card*	2
Emulation of PLC analog I/O*	2
Simulation of analog output*	2
Simulation of feedback control*	2
Emulation of analog output*	2
Emulation of feedback control*	2
PLC controllers*	2
Multiple PLC connection*	2
Synchronization of multiple PLC*	2

(*modified in the framework of an Erasmus + project: Asean Factori 4.0 Across South East Asian Nations: From Automation and Control Training to the Overall Roll-out of Industry 4.0 609854-EPP-1-2019-1-FR-EPPKA2-CBHE-JP)

1501431 Intelligent Control Systems

Program: Bachelor program in Computer Engineering

Credit: 3(2-2) Lecture: 30 Hours Lab: 30 Hours

FACTORI 4.0 Erasmus +

Co-funded by the Erasmus+ Programme of the European Union

This course has been modified in the framework of an Erasmus + project: Asean Factori 4.0 Across South East Asian Nations: From Automation and Control Training to the Overall Roll-out of Industry 4.0

609854-EPP-1-2019-1-FR-EPPKA2-CBHE-JP

1st Semester, Academic Year: 2024

Assoc. Prof. Punnarumol Temdee, Ph.D.

Asst. Prof. Roungsan Chaisricharoen, Ph.D.

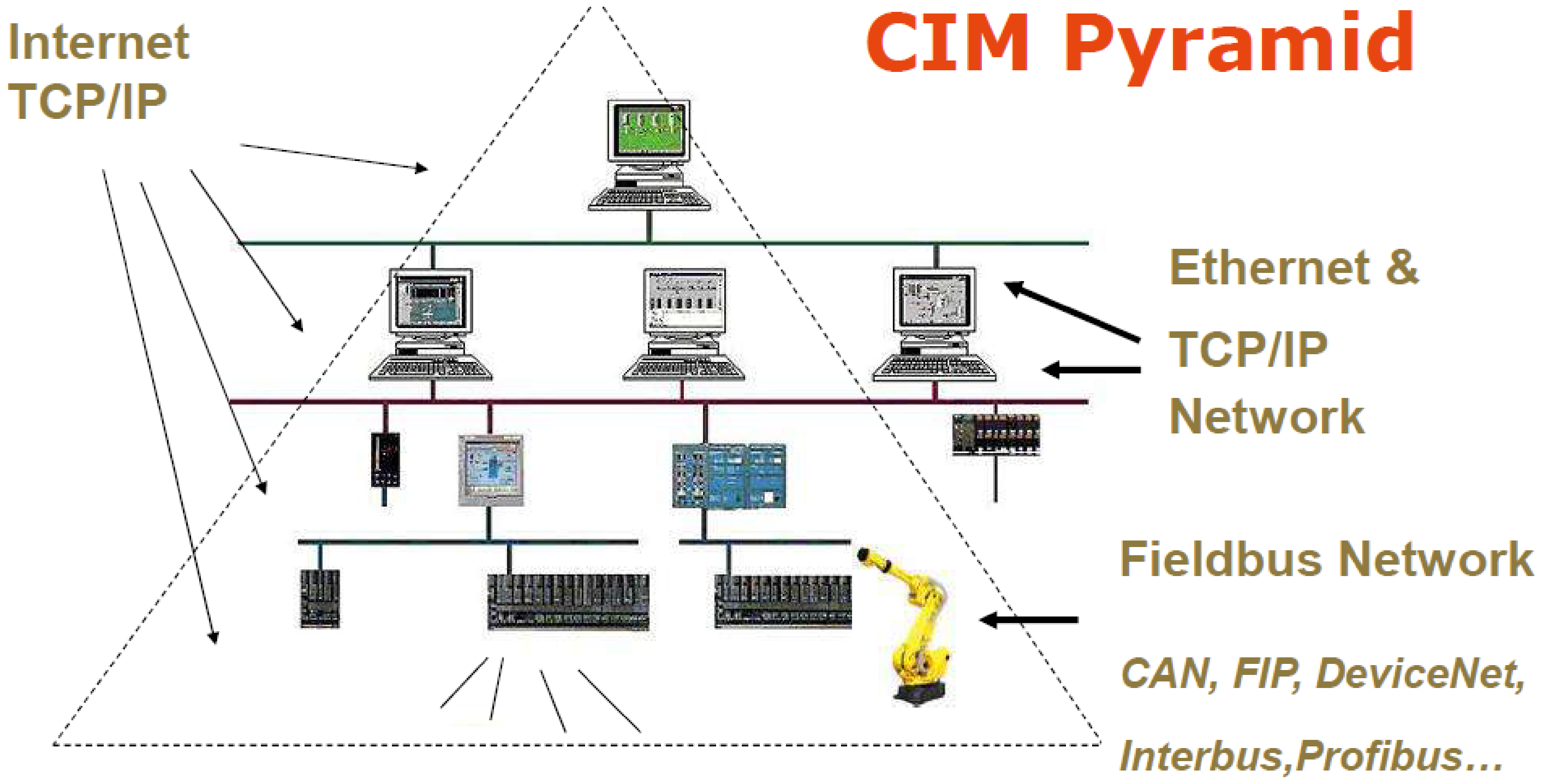
Asst. Prof. Santichai Wicha, Ph.D.

Lect. Chayapol Kamyod, Ph.D.

Lecture 01: Control system for industrial

2 sessions, 4 hrs

CIM Pyramid

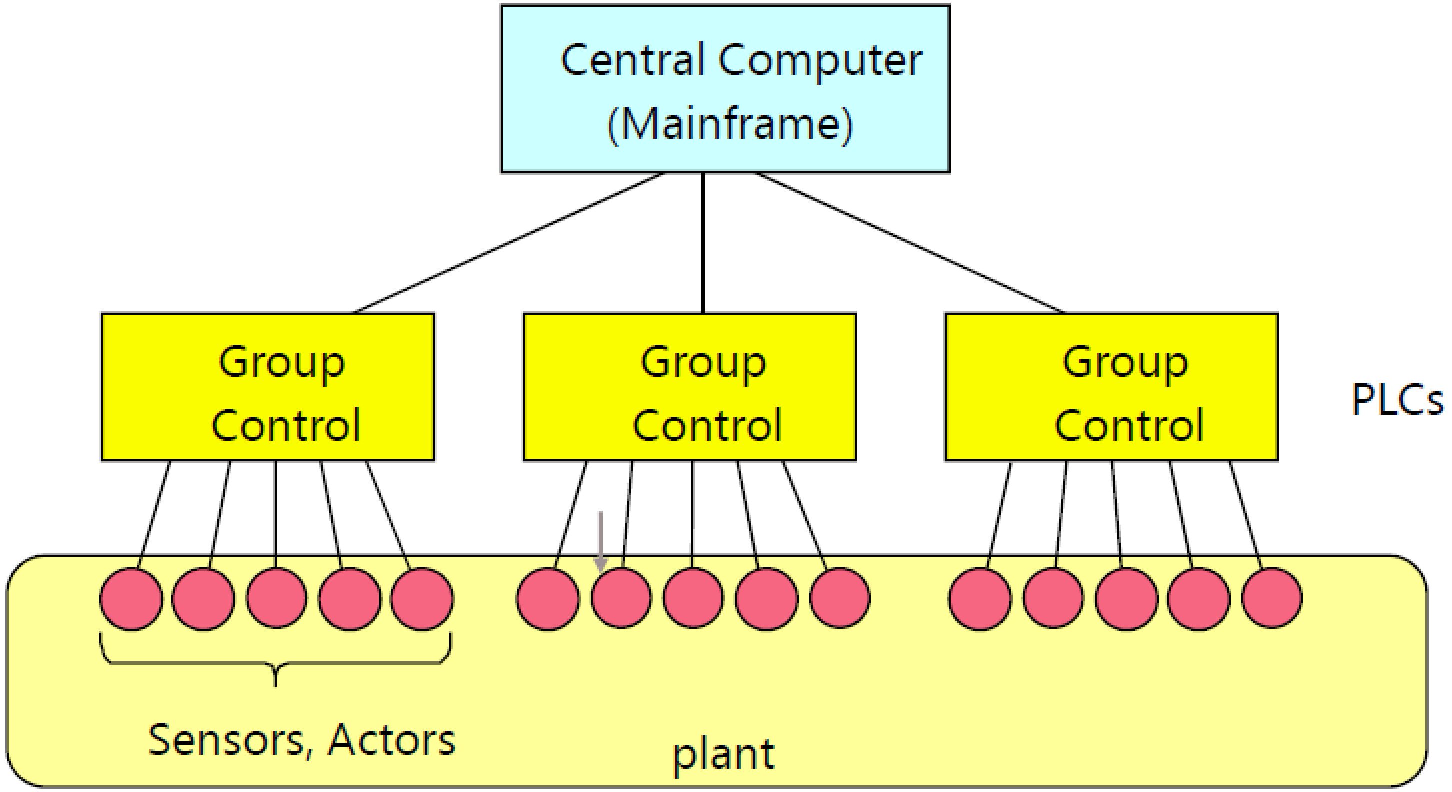


Computer-integrated manufacturing (**CIM**)

Describe the complete automation of manufacturing processes

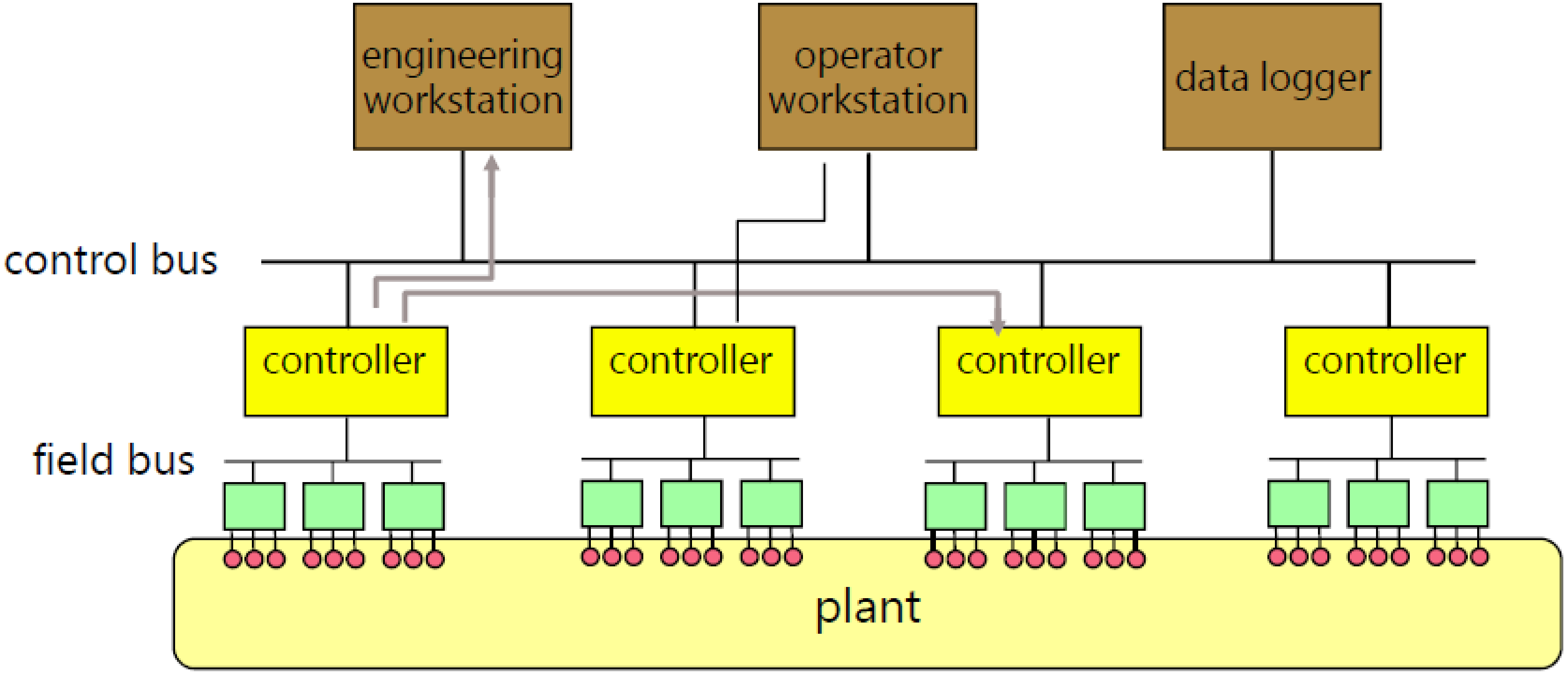
Several network layers

Centralized Control Architecture



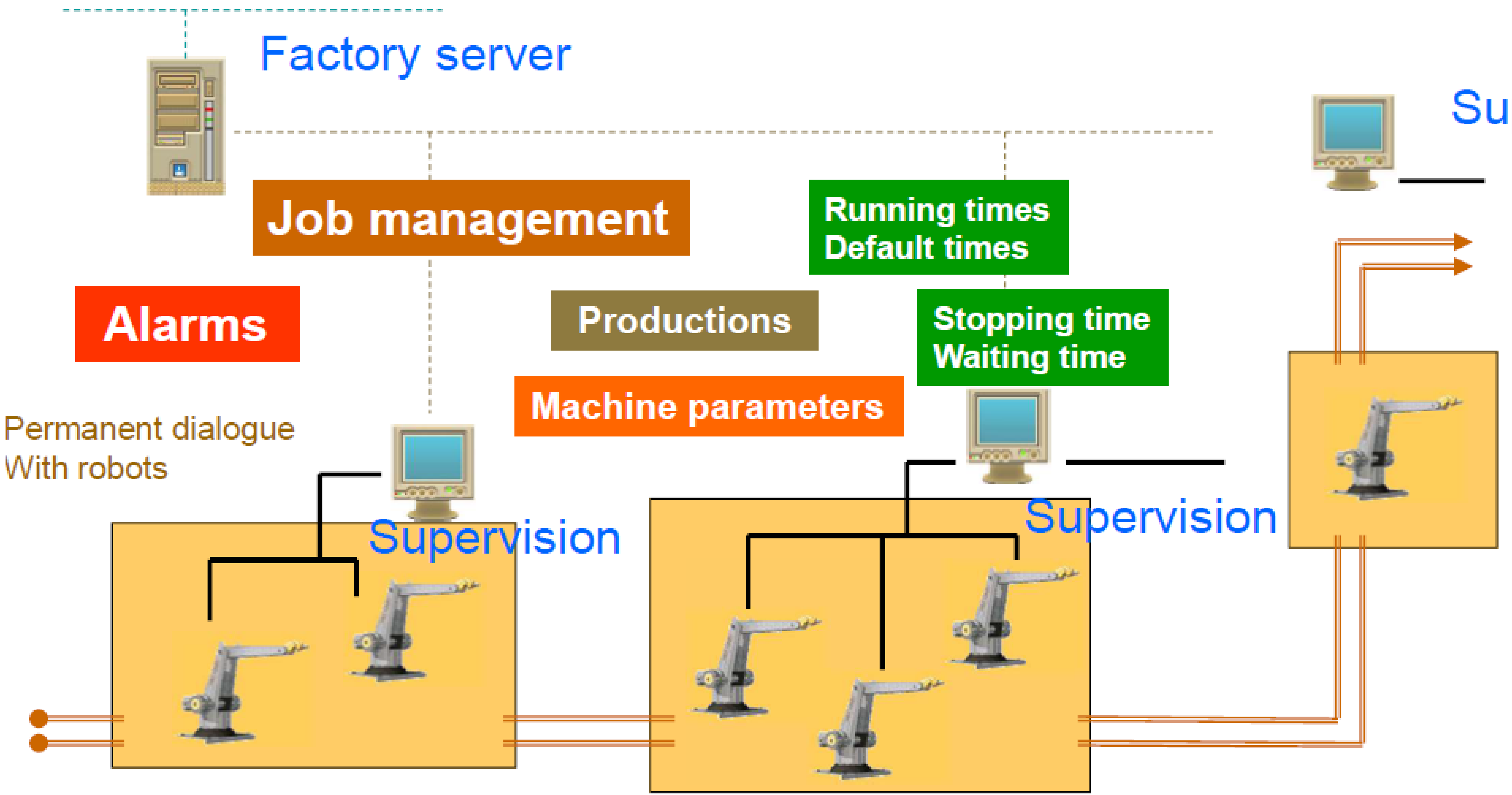
Classical, hierarchical, centralised architecture.
The central computer only monitors and forwards commands to the PLCs

Decentralized Control System (DCS)



All controllers can communicate as peers (without going through a central master), restricted only by throughput and modularity considerations.

Supervision

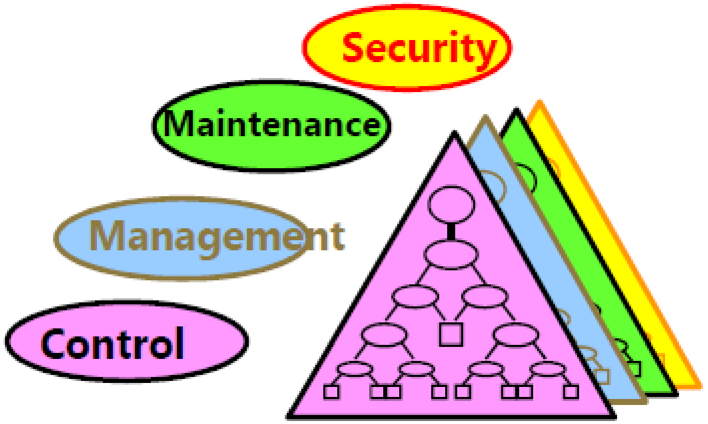


Supervisors
Collect and process information from the robots
Basis of the factory's information system

Context: Automation system evolution

UGA Université Grenoble Alpes

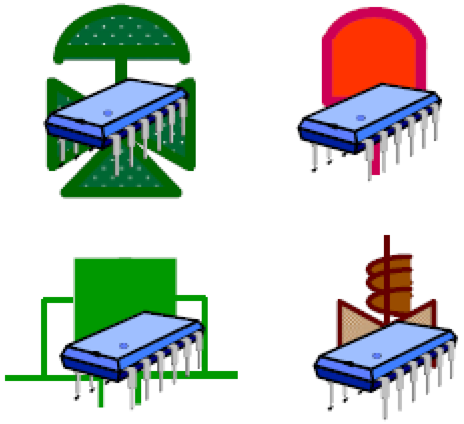
Needs:



Increased number of services

More complex architectures →

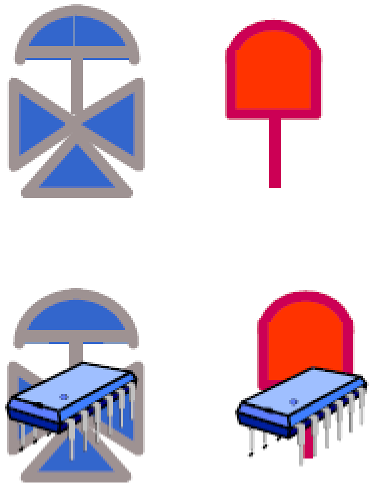
Components :



Various capacities and functionalities availability

Dependability hard to evaluate and to qualify

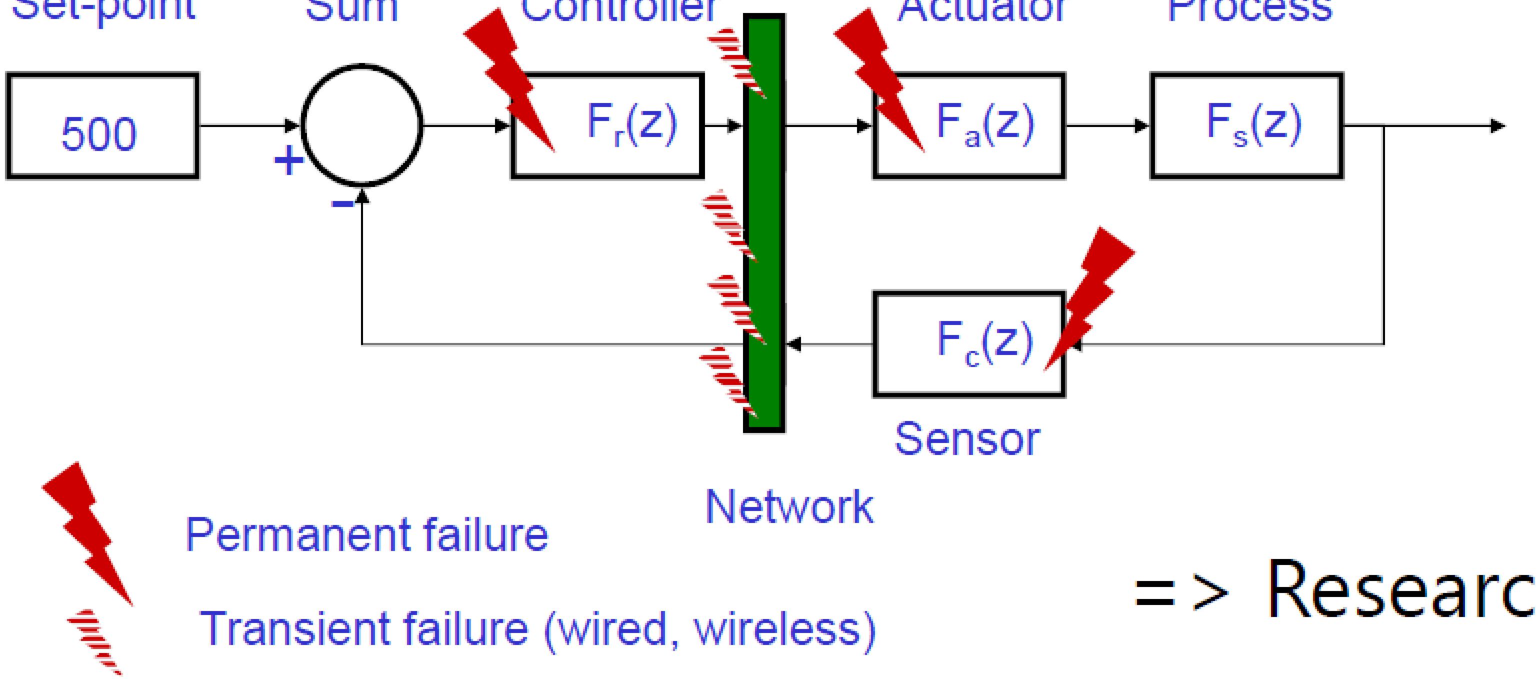
From analog to digital and from smart to intelligent…



- Analog sensors and actuators
  - Hardware and analytical Redundancies
  - « Classiques" studies of dependability
- Digital sensors and actuators
  - A/D Interfaces, processing units, delays…
  - Software, implementation
- « Smart » sensors and actuators
  - Embedded intelligence, local decision
- « Intelligents » sensors and actuators
  - Communicating Interface
  - Diagnosic, monitoring, checking, embedded decision
  - Instrument contributing of the global « intelligence » of the system
- Intelligence vs. Complexity => consequences on Dependability

?

Failures integration



=> Research aspects

Failure Modes
- Continuous/sampled
- Discrete events

Time scales

-Speed (modulation rate, throughput) of the networks
-System time constant
-Time between failures

Exemple de SCADA

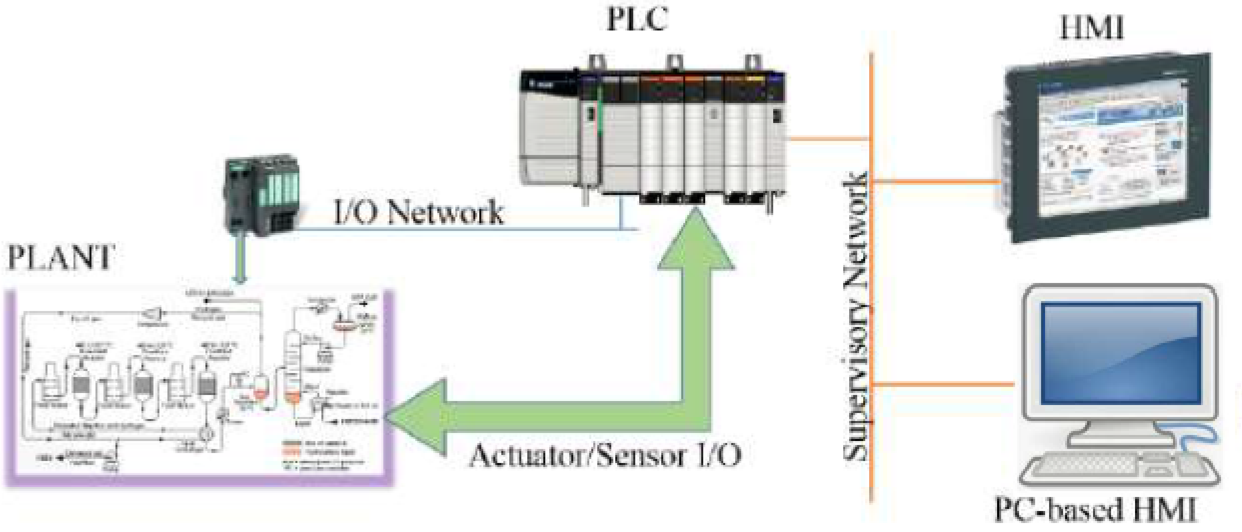
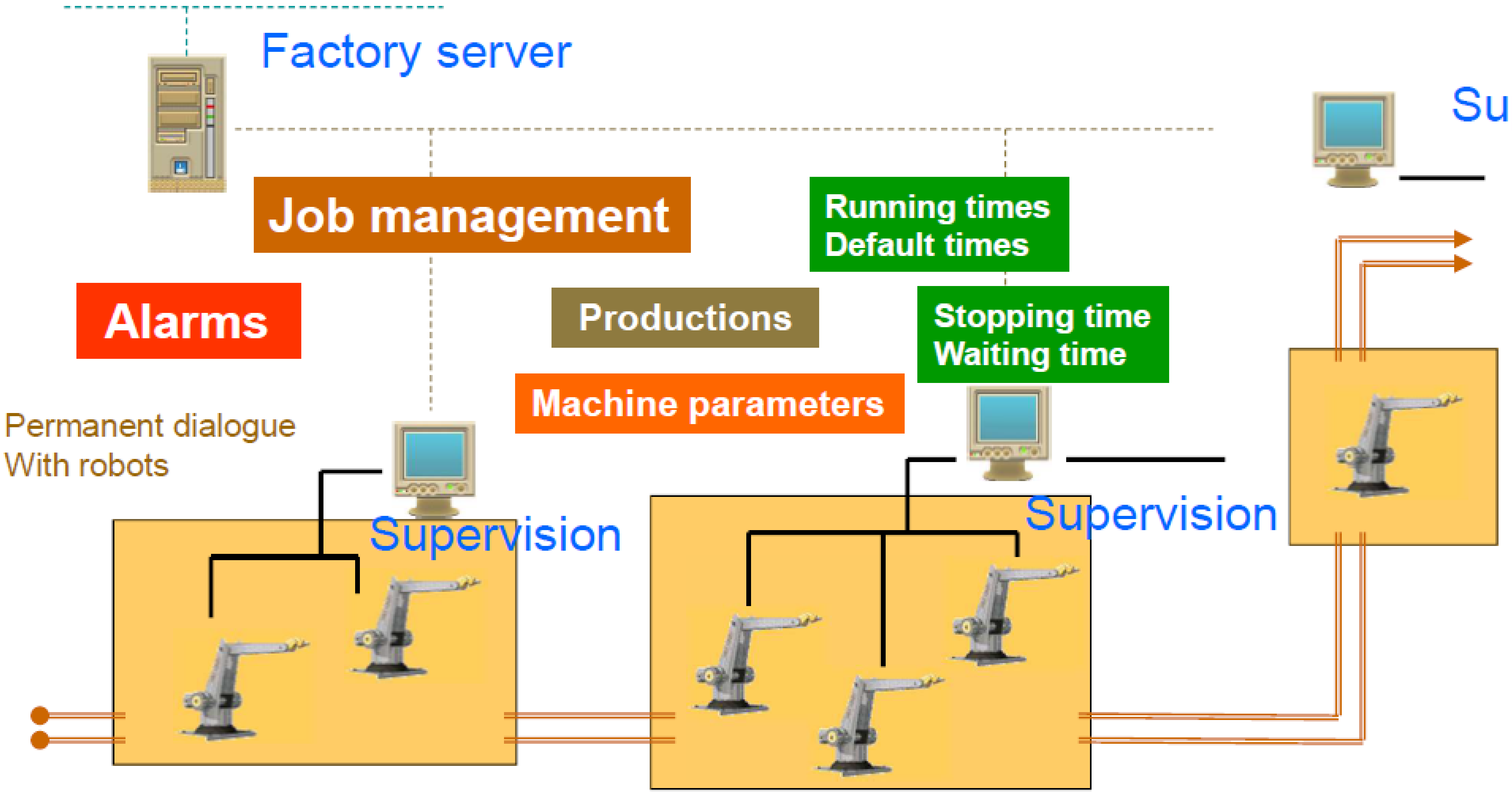


Figure 1. The simple SCADA system

Supervisory Control And Data Acquisition

Supervision : computerized monitoring and control of automated manufacturing processes
- Data acquisition
- Manual or automatic modification of process control parameters
- Use of PLCs, special machines, robots...

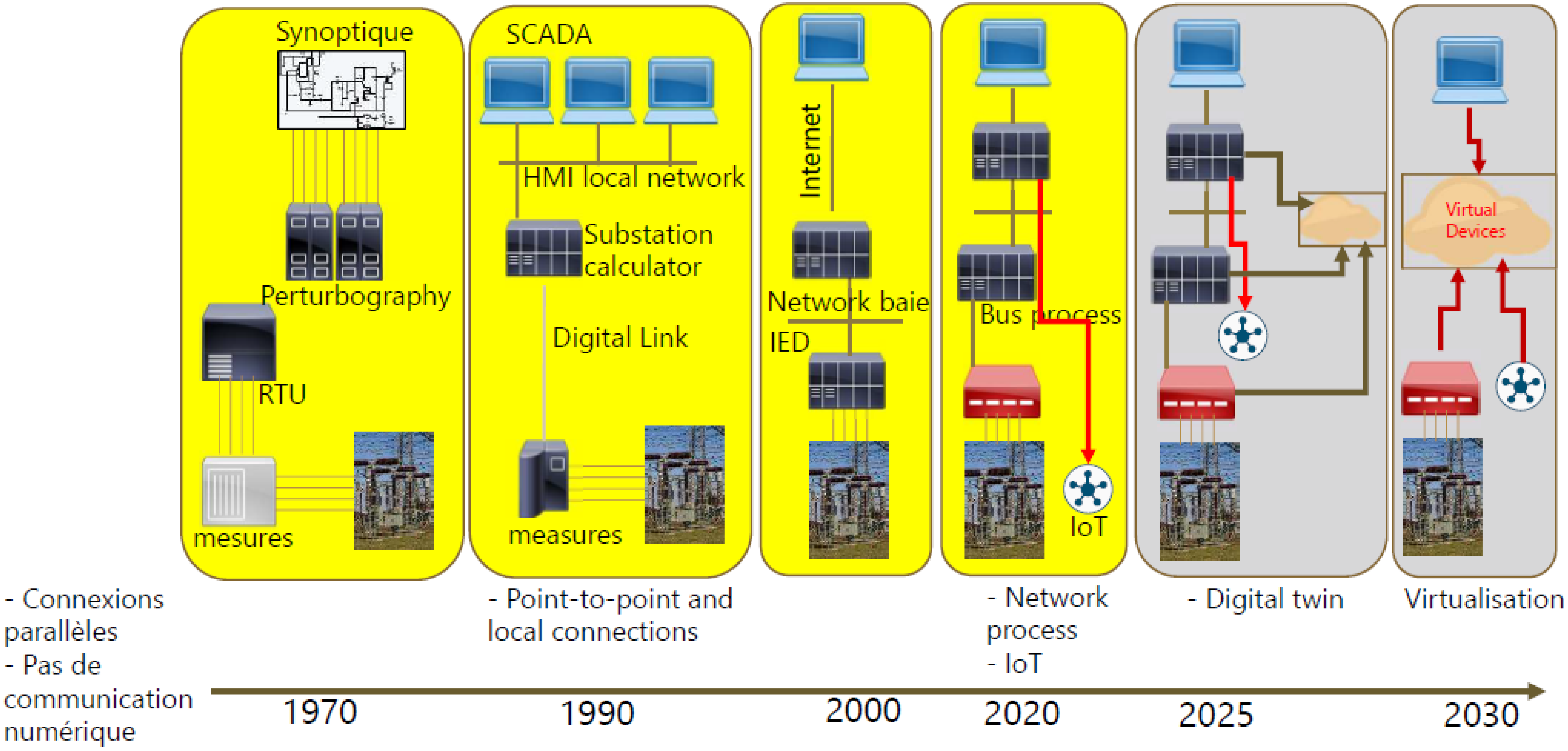
Supervision



Supervisors
Collect and process information from the robots
Basis of the factory's information system

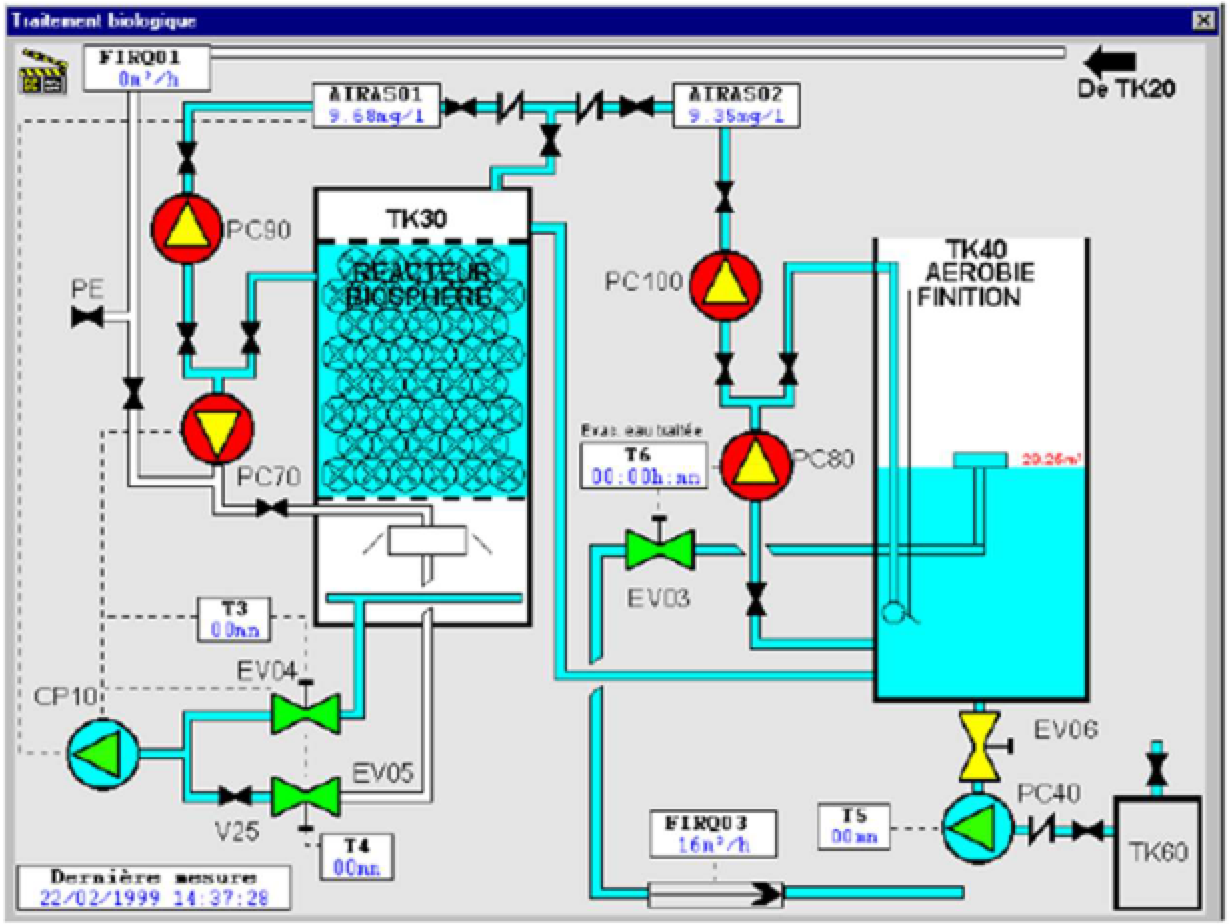
Supervision and Cloud technologies

- Example : evolution of the electrical substation



Supervision functions

Synoptic: essential function of the supervision, provides a synthetic, dynamic and instantaneous representation of all the means of production of the unit

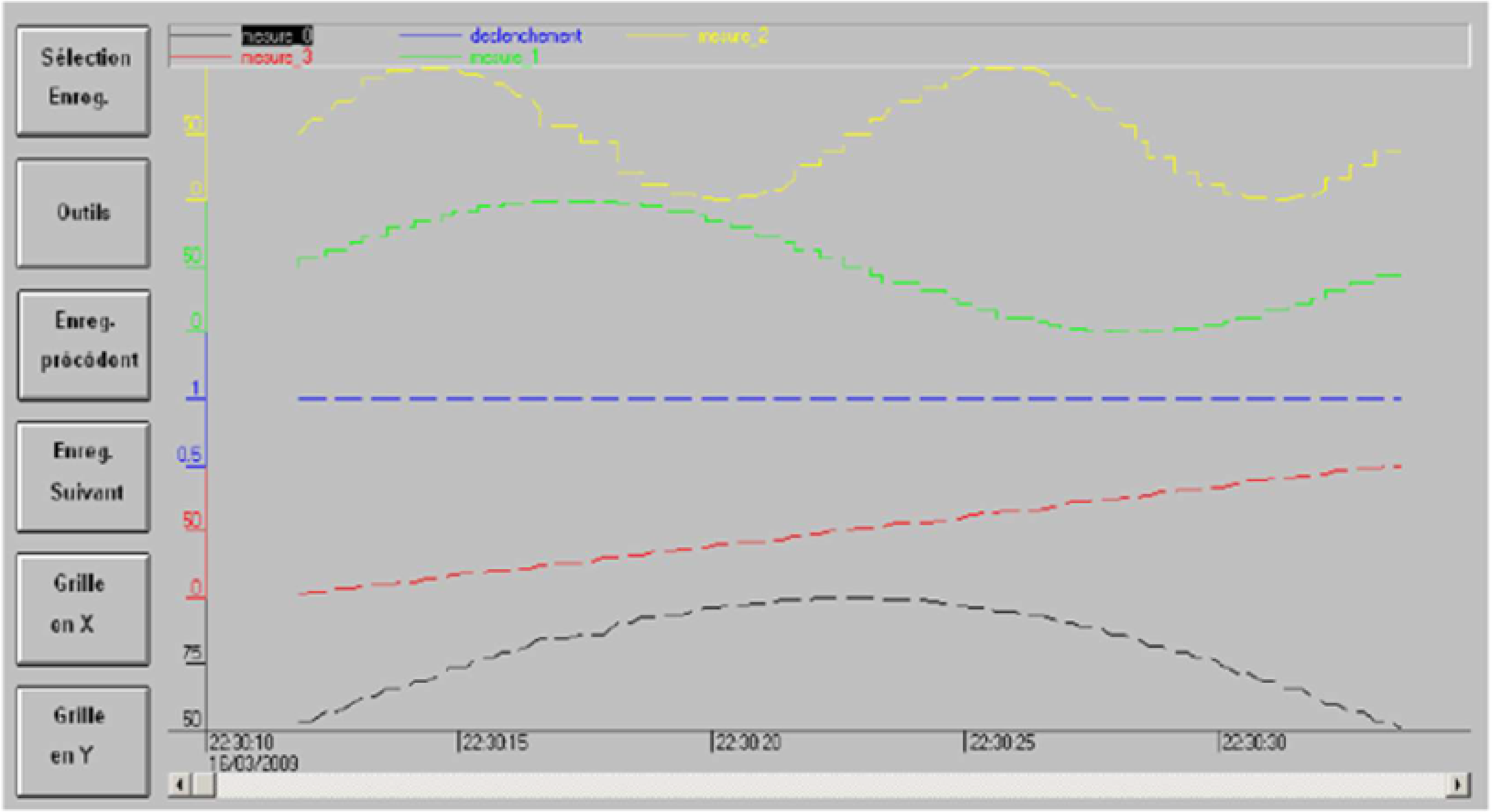


Supervision functions

Curves:

- gives a graphical representation of different process data

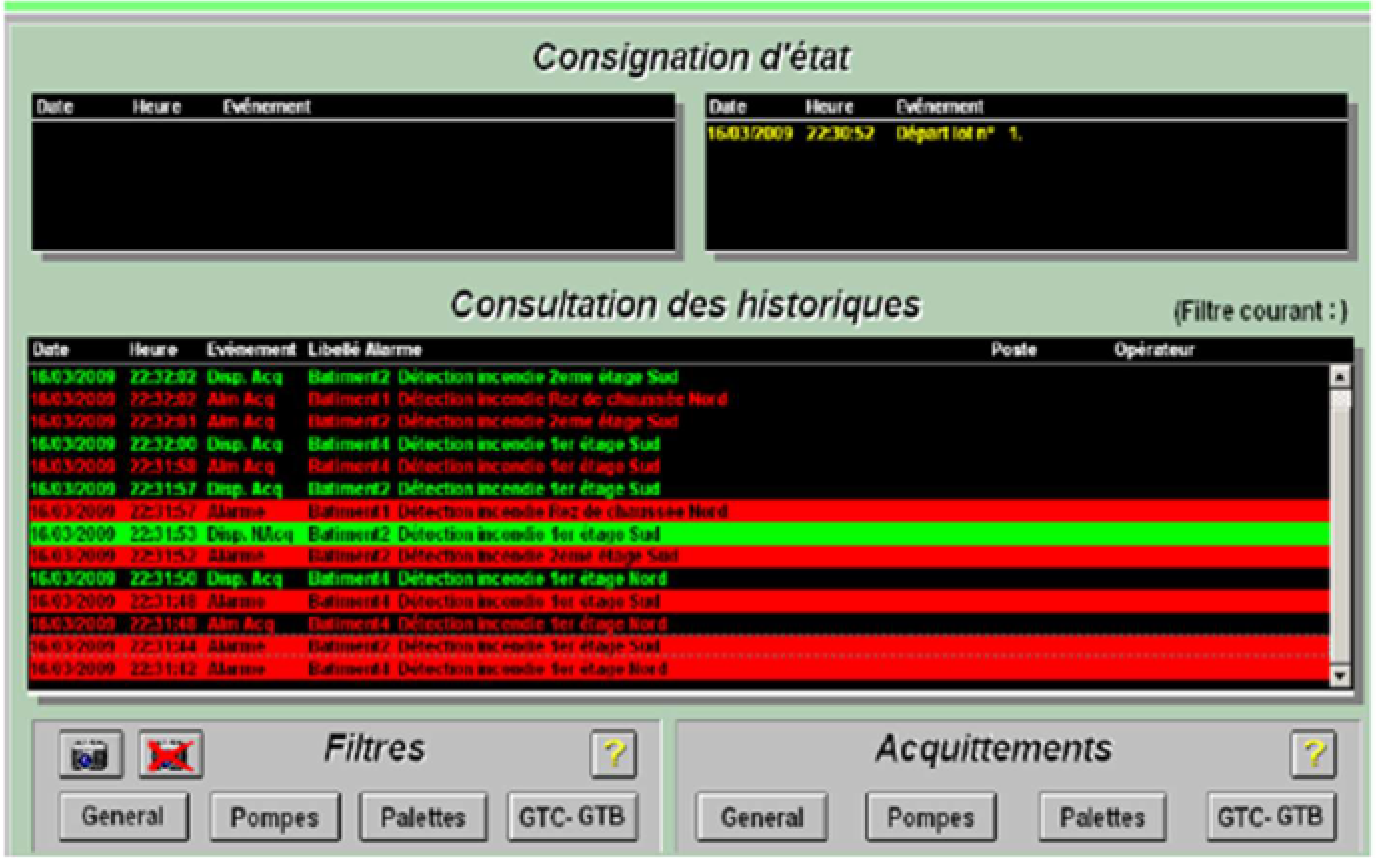
- gives the tools to analyze the historical variables



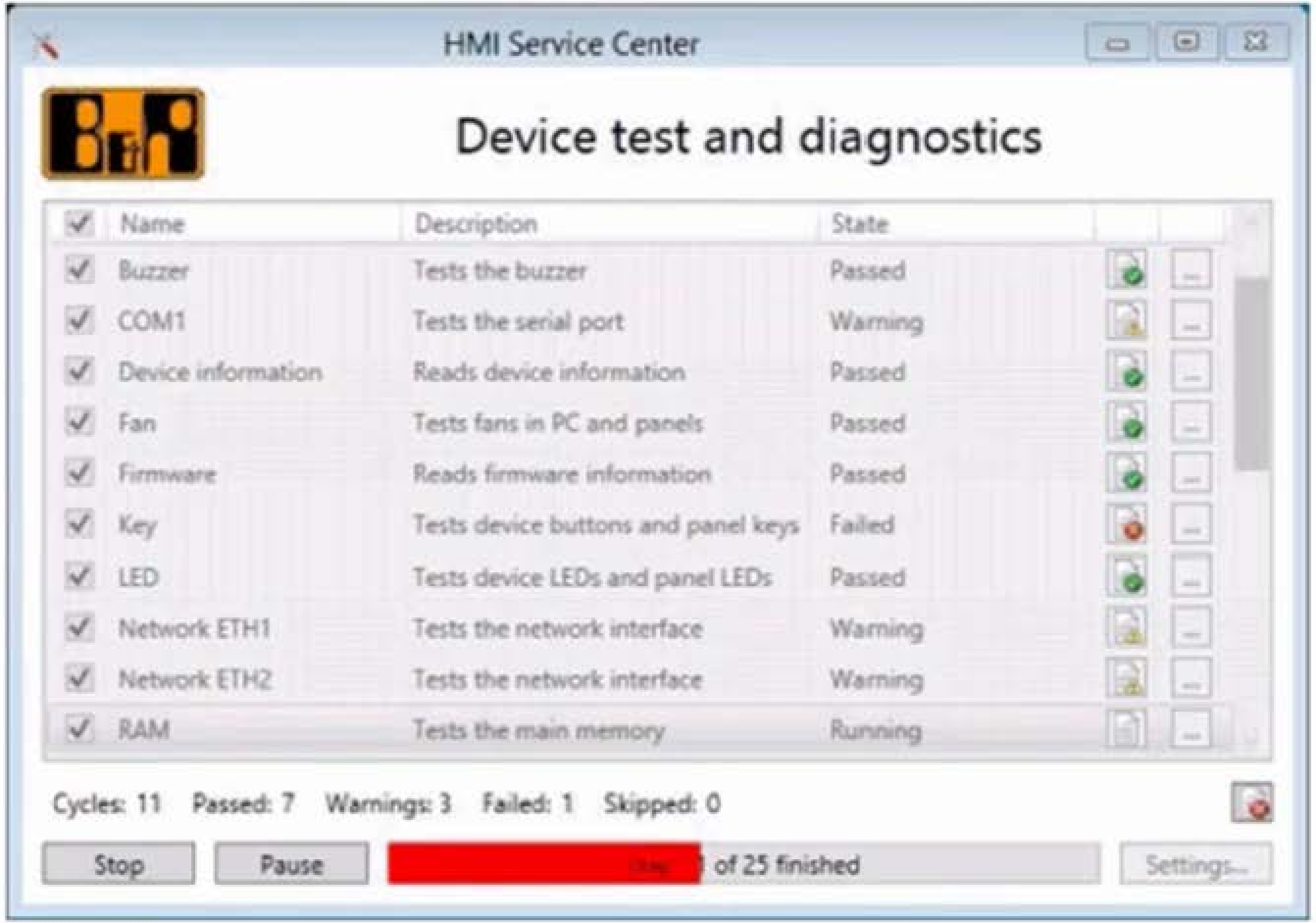
Supervision functions

Alarms

- Calculates in real time the conditions for triggering alarms
- Displays all alarms according to priority rules
- gives management tools
- ensures the recording of all the steps of the alarm processing



Supervision functions



Alarms

Circumscribe the cause of the feared event (cause of the incident)
Limit the impact of the event, protect (consequences)
Be able to assess the system after the incident: repair, reconfigure (total and partial redundancies)
Reconstruct, recover the system: time required for it to be operational again, what happens and what are the recovery steps? (Activity Return Plan)

Other related aspects: robustness, resilience (ability to maintain the system as well as possible in a situation of "attacks")

Alarms detection

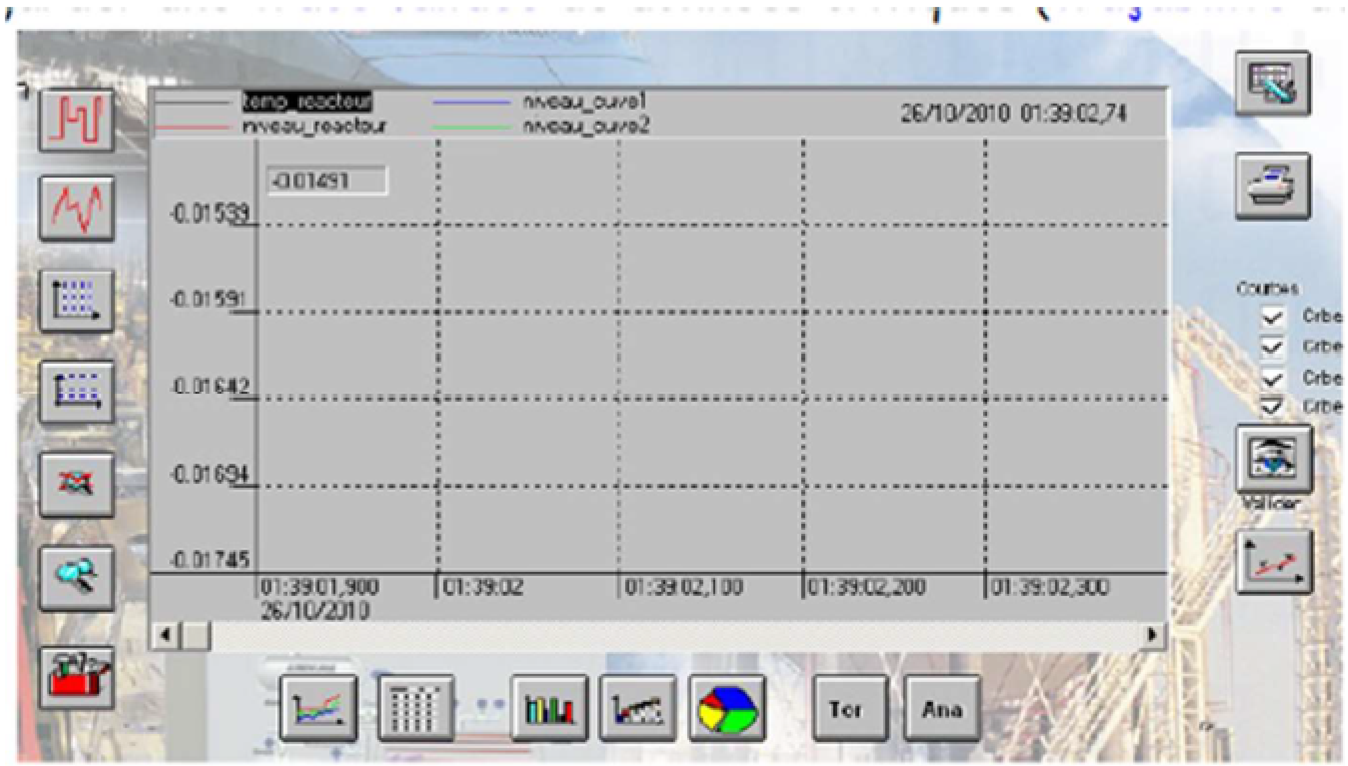
- TP (true positive) corresponds to correctly identified alarms
- FP (false positive) corresponds to authentic behavior identified as faulty
- TN (True Negative) corresponds to the correct rejection of authentic behavior
- FN (False Negative) corresponds to undetected failures
- Two metrics are used to evaluate the performance of alarm detection
  - True Positive Rate TPR=TP/(TP+FN)

    => 1 if no False Negative
  - False Positive Rate FPR=FP/(FP+TN)

    => 0 if no False Positive

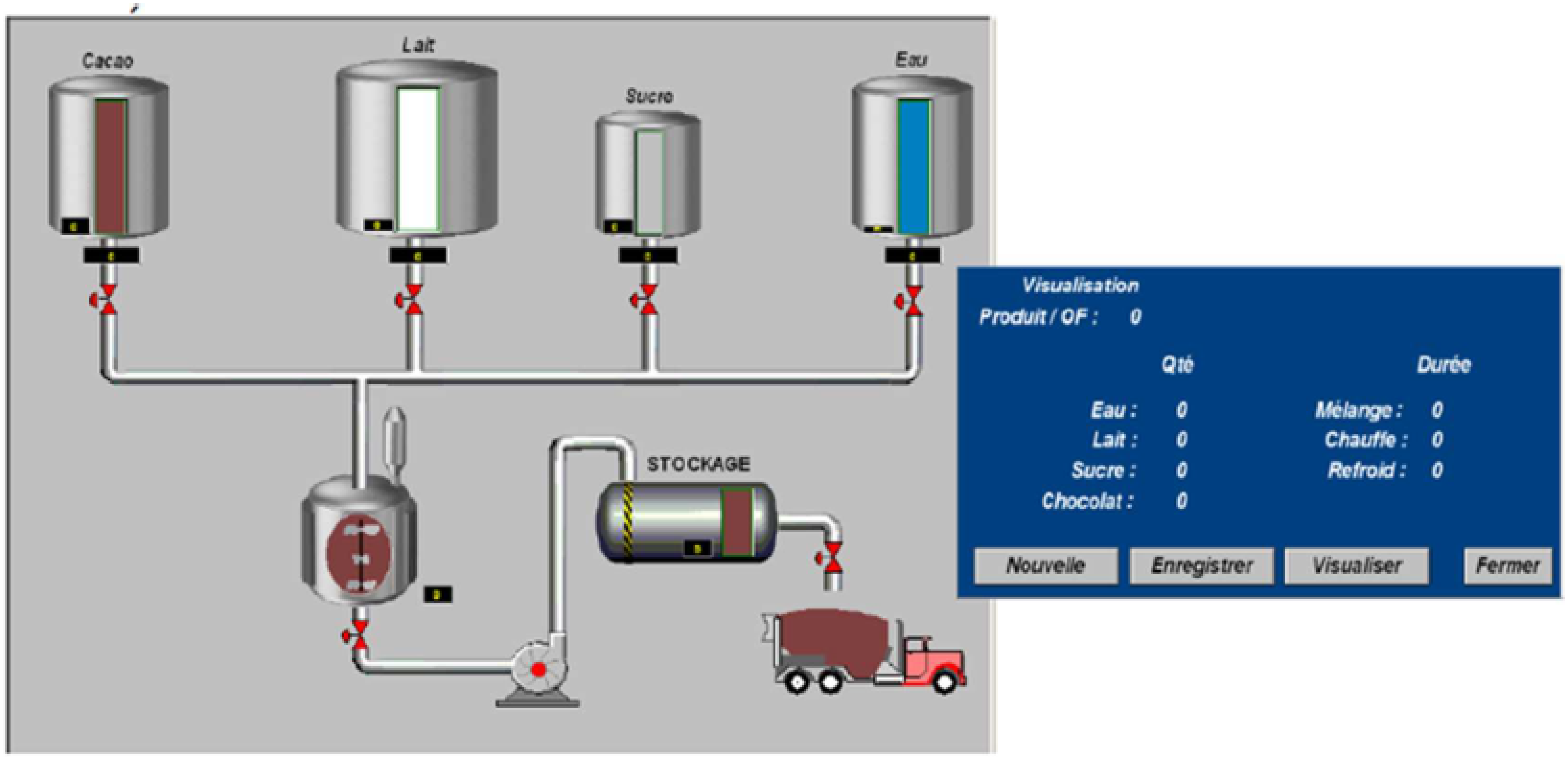
Supervision functions

Historicization of the process:
- Allows the saving of time-stamped events (selective archiving)
- provides search tools in the archived years
- provides the possibility to run the synoptic again with archived data (replay function)
- allows to keep a validated trace of critical data (traceability of production data)



Supervision functions

Management of production lines and recipes:
- Provides a tool for managing production batches
- Manages the parameters of the machines for each batch (recipes)



Exercise

- Study the process of freeze dry
  - Specify a set of parameters to be monitored
  - Design supervision and alams

Lecture 02: Speed and precision of a system

2 sessions, 4 hrs

Tuning of a control system

The root locus method alone can help tune the system by showing the effect of a parameter's variation. However, this may not be enough for the situation where the existed root locus is not passing the required area. In this case, an additional pole and zero compensation can help bend the interested root locus to the shape that can deliver the desired response. The compensator acts as a controller and is placed before the system's process.

DC Motor

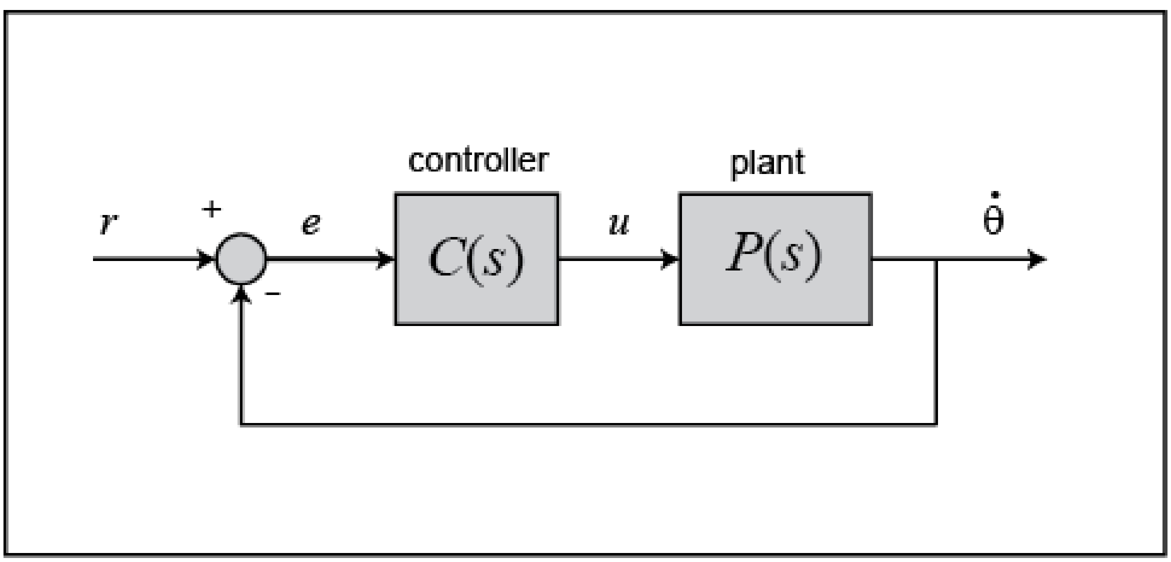
From the main problem, the dynamic equations in the Laplace domain and the open-loop transfer function of the DC Motor are the following.

$$s(Js+b)\Theta(s) = KI(s) \qquad (1)$$

$$(Ls+R)I(s) = V(s) - Ks\Theta(s) \qquad (2)$$

$$P(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2} \qquad [\frac{rad/sec}{V}] \qquad (3)$$

The structure of the control system has the form shown in the figure below.



DC Motor model

For a 1-rad/sec step reference, the design criteria are the following.

- Settling time less than 2 seconds
- Overshoot less than 5%
- Steady-state error less than 1%

Now let's design a controller. Create a new m-file and type in the following commands.

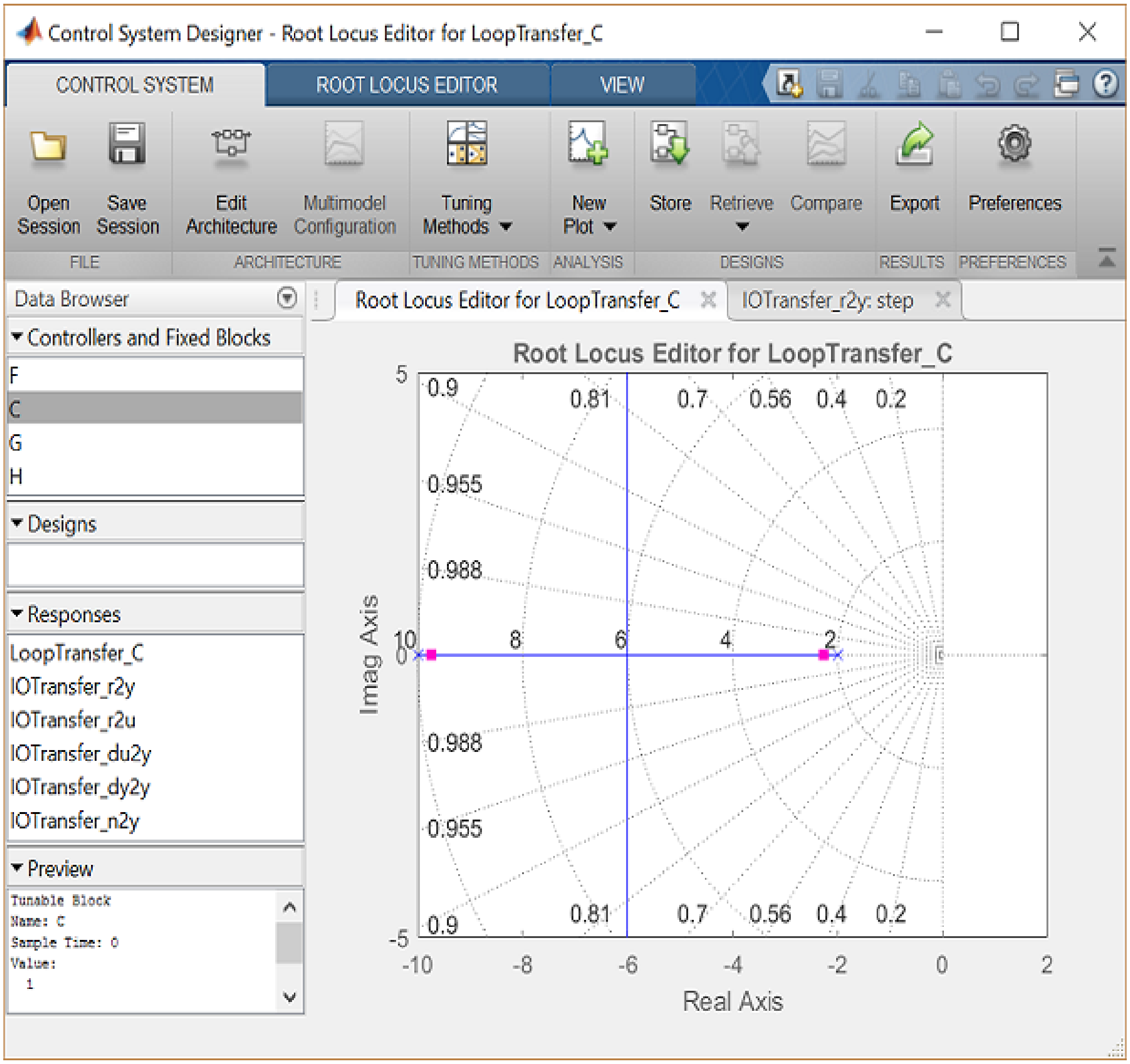
```
J = 0.01;

b = 0.1;

K = 0.01;

R = 1;

L = 0.5;

s = tf('s');
P_motor = K/((J*s+b)*(L*s+R)+K^2);
```

Drawing the open-loop root locus

The main idea of root locus design is to predict the closed-loop response from the root locus plot which depicts possible closed-loop pole locations and is drawn from the open-loop transfer function. Then by adding zeros and/or poles via the controller, the root locus can be modified in order to achieve a desired closed-loop response.

We will use for our design the **Control System Designer** graphical user interface. This tool allows you to graphically tune the controller via the root locus plot. Let's first view the root locus for the uncompenstated plant. This is accomplished by adding the command controlSystemDesigner('rlocus', P_motor) to the end of your m-file and running the file at the command line or by going to the **APPS** tab of the MATLAB toolstrip and clicking on the app icon under **Control System Analysis and Design**.

One window titled **Control System Designer** will open initially having the form shown in the figure below. In the window, you will be able to see both the root locus plot and the closed-loop step response of the transfer function passed via the controlSystemDesigner function. If the string 'rlocus' is omitted from the function call, the default initial window includes the Bode plot, in addition to the root locus plot and closed-loop step response plot. You can arrange the position of plots from the **VIEW** tab of the **Control System Designer** window. Right-clicking on the root locus plot and selecting **Grid** will make your window appear as follows.



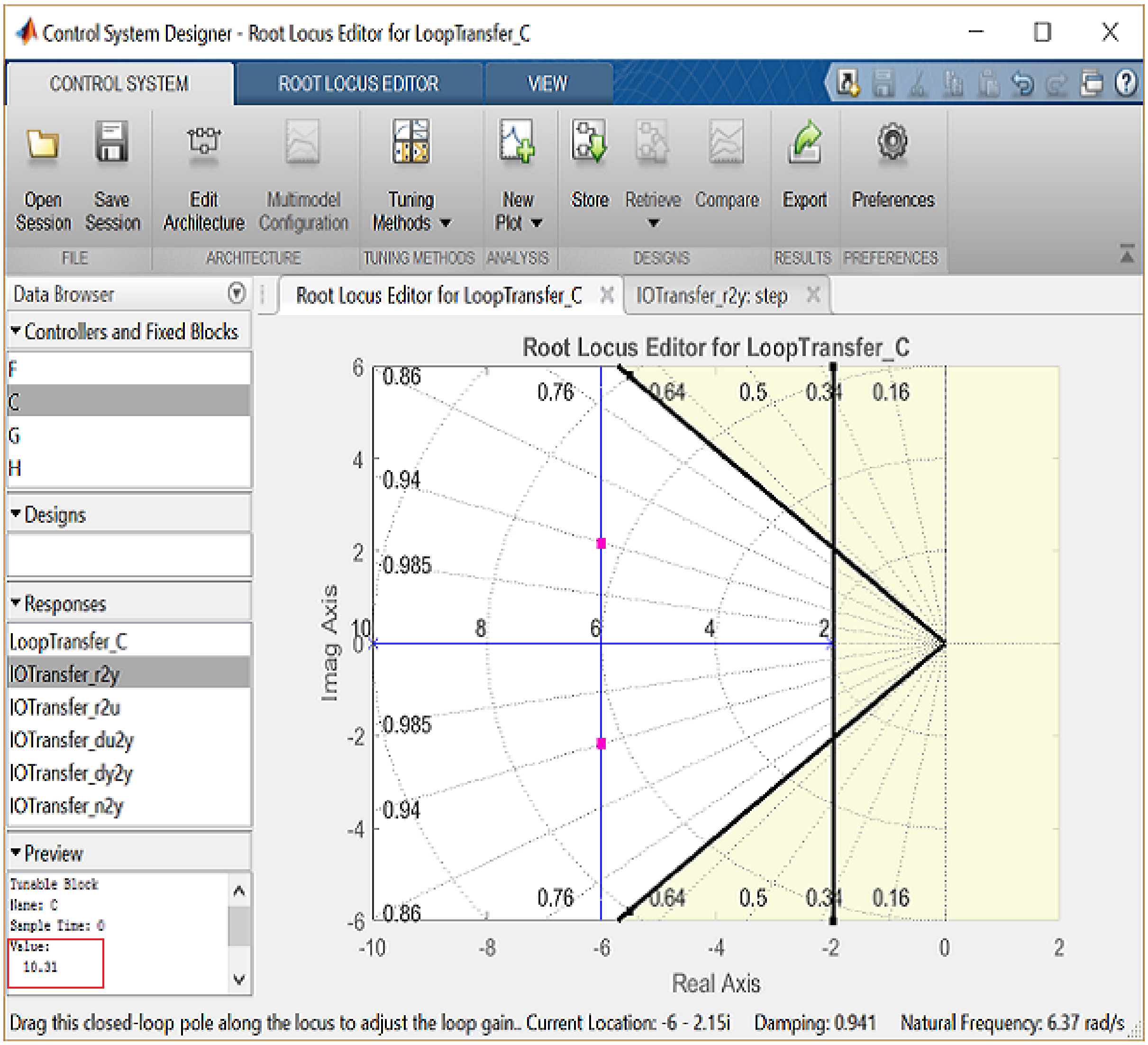
Root locus of a DC motor

Finding the loop gain

Recall that our design requirements specify that the settling time be less than 2 seconds and that the overshoot be less than 5%. The location of the system's closed-loop poles provide information regarding the system's transient response. The **Control System Designer** allows you to specify the region in the complex *s*-plane corresponding to specific design requirements. The provided regions correspond to a canonical second-order system, but in general are a good place to start from even for higher-order systems or systems with zeros.

These desired regions can be added to the root locus plot by right-clicking on the plot and choosing **Design Requirements > New** from the resulting menu. You can add many design requirements including Settling time, Percent overshoot, Damping ratio, Natural frequency, and generic Region constraint.

Adding our settling time and percent overshoot requirements to the root locus plot produces the following figure.

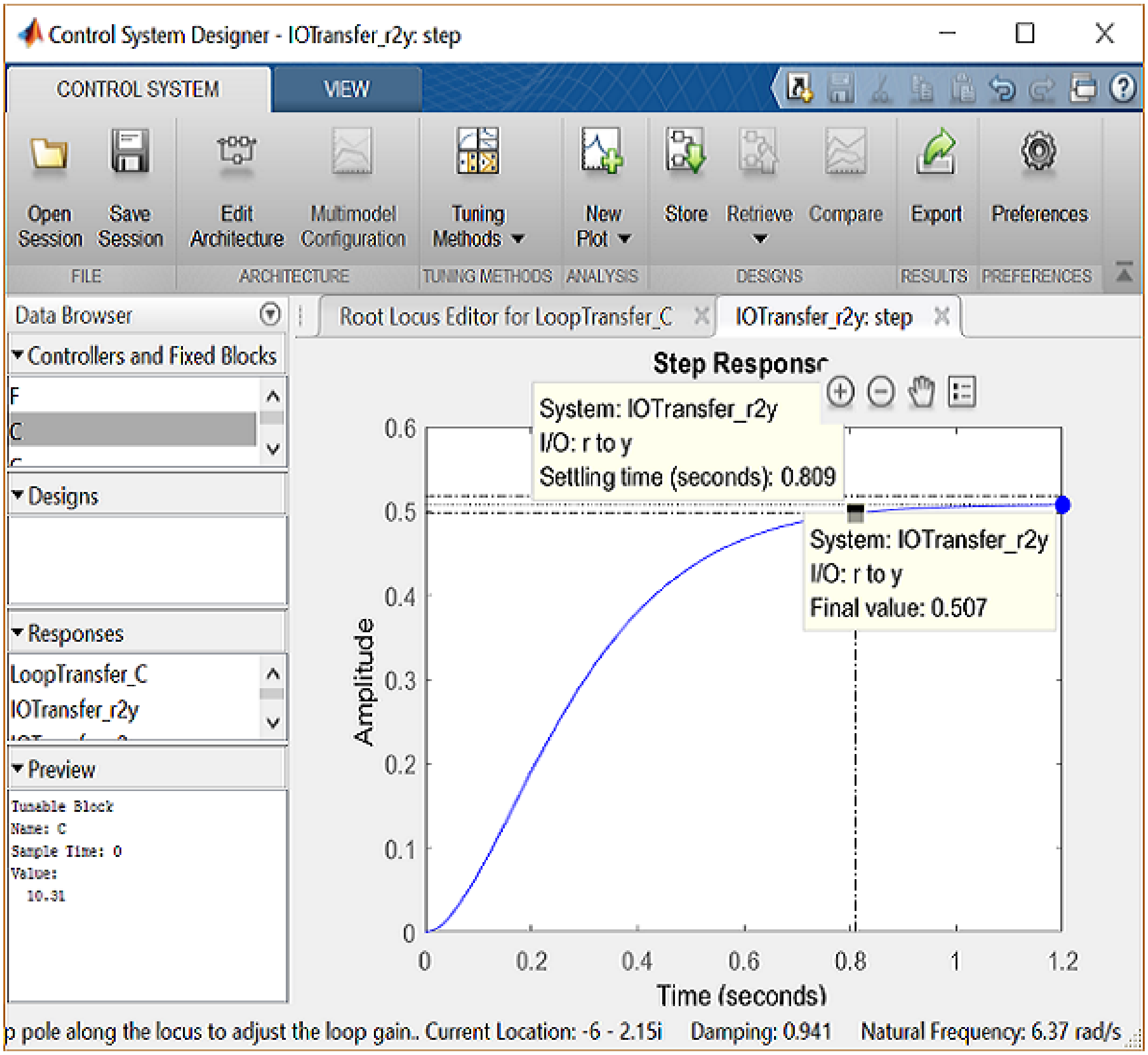


settling time and percent overshoot requirements

The resulting desired region for the closed-loop poles is shown by the unshaded region of the above figure. More specifically, the two rays centered at the origin represent the overshoot requirement; the smaller the angle these rays make with the negative real-axis, the less overshoot is allowed. The vertical line at $s = -2$ represents the settling time requirement, where the farther to left the closed-loop poles are located the smaller the settling time is. From examination of the above figure, there are values of the loop gain that will place both closed-loop poles in the desired region. This can be seen from the fact that the two branches of the root locus are symmetric and pass through the unshaded region. Furthermore, since the closed-loop system has two poles with no zeros, placing the closed-loop poles in the shown region will guarantee satisfaction of our transient response requirements.

You can select a specific pair of closed-loop poles from the resulting figure in order to determine the corresponding loop gain that places the poles at that location. For our system, let's choose to place the closed-loop poles so that they are located on the vertical branches of the root-locus between the real axis and the boundary of the overshoot requirement. The pink boxes on the root locus indicate the location of the closed-loop poles for the current loop gain. Clicking on the pink boxes and dragging them along the root locus to the desired location automatically modifies the controller to place the closed-loop poles at the indicated position. Let us drag a closed-loop pole to a location near -6 + 2i. The pole location will be indicated at the bottom of the window along with the corresponding damping ratio and natural frequency. We can also check the corresponding loop gain in the lower left corner by clicking on **C** in the **Controllers and Fixed Blocks** tab. The loop gain, as we can see in the figure, is approximately 10.

We can check the closed-loop step response for the system with this new gain by moving to the **IOTransfer_r2y: step** tab. If you have accidentally closed this tab, you can re-open it from the **Control System Designer** window by clicking on the **New Plot** menu and selecting **New Step**. In response, a new window titled **New Step to plot** will appear. From the **Select Responses to Plot** menu, then choose **IOTransfer_r2y** and click the button **Plot**. The response of the output y of the closed-loop system for a step reference r will then appear in the **Control System Designer** window. You can also identify some characteristics of the step response. Specifically, right-click on the figure and under **Characteristics** choose **Settling Time**. Then repeat for **Steady State**. Your figure will appear as shown below.



characteristics of the step response

From inspection of the above, one can see that there is no overshoot and the settling time is less than one second, therefore, the overshoot and settling time requirements are satisfied. However, we can also observe that the steady-state error is approximately 50%. If we increase the loop gain to reduce the steady-state error, the overshoot will become too large. You can see this for yourself by graphically moving the closed-loop poles vertically upward along the root locus, this corresponds to increasing the loop gain. The step response plot will change automatically to reflect the modified loop gain. We will attempt to add a lag controller to reduce the steady-state error requirement while still satisfying the transient requirements.

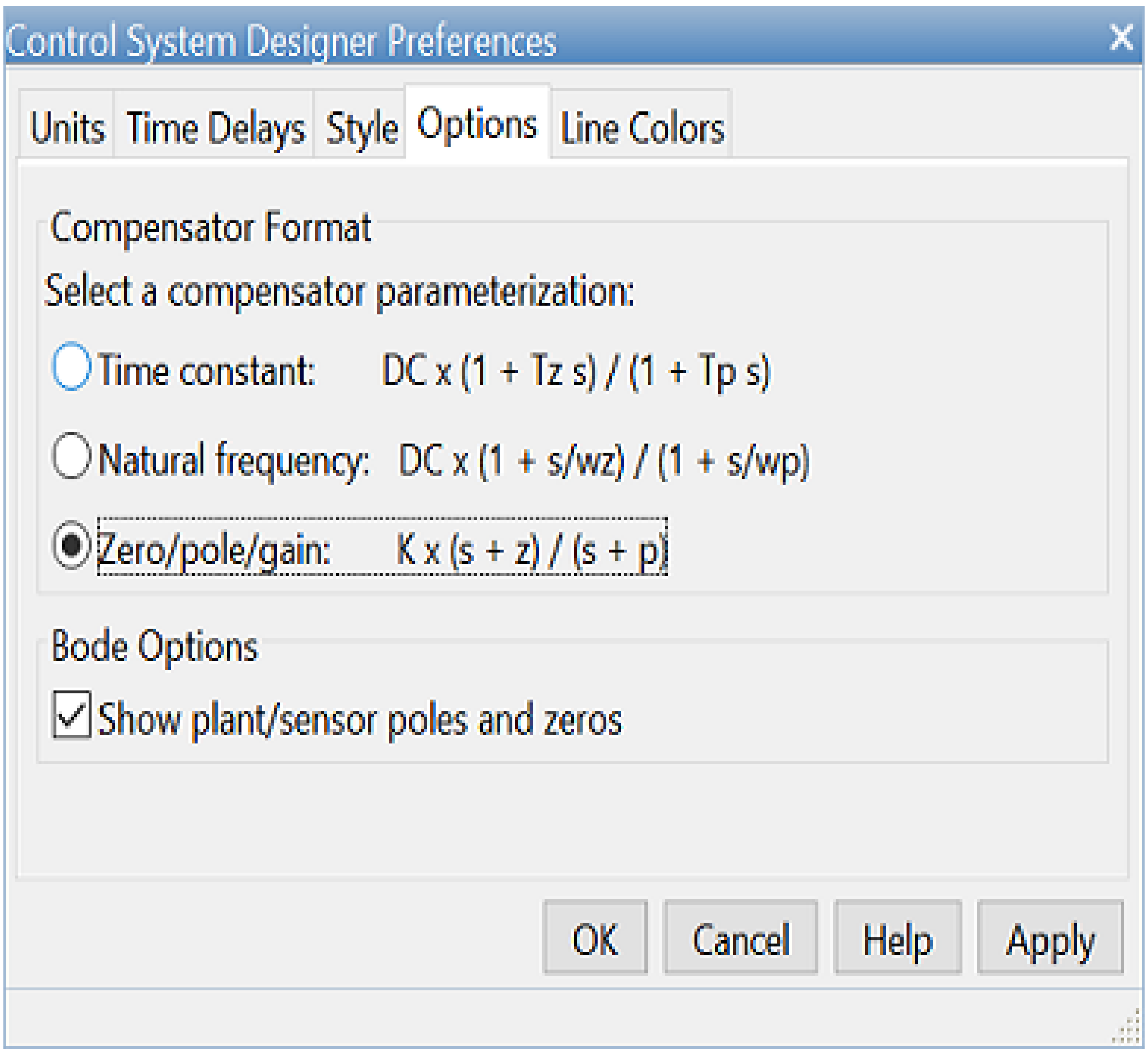
Adding a lag controller

In the above we saw that the overshoot and settling time criteria were met with the proportional controller, but the steady-state error requirement was not. A **lag**

compensator is one type of controller known to be able to reduce steady-state error. However, we must be careful in our design to not increase the settling time too much. Let's first try adding a lag compensator of the form given below.

$$C(s) = \frac{(s+1)}{(s+0.01)} \tag{4}$$

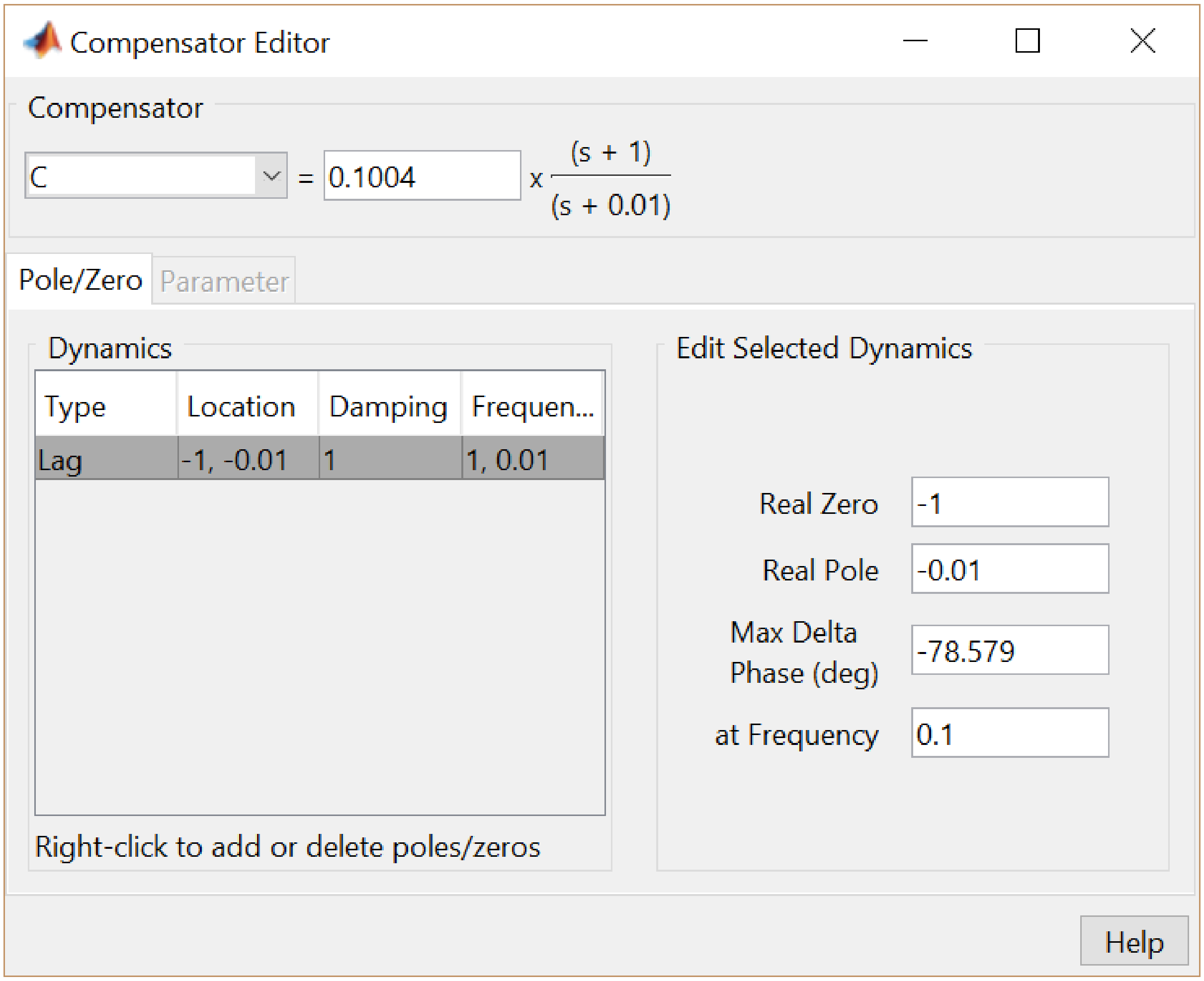
We can use the **Control System Designer** to design our lag compensator. To make the **Control System Designer** have a compensator parameterization corresponding to the one shown above, click on the **Preferences** menu at the top of the **Control System Designer** window. Then From the **Options** tab, select a **Zero/pole/gain** parameterization as shown below.



Options of Control System Designer

To add the lag compensator, right click on the root locus plot and select **Edit Compensator**. To add a pole zero pair to your compensator, in the **Compensator**

Editor window, right-click the **Dynamics** table and select **Add Pole/Zero > lag**. After that, enter the **Real Zero** and **Real Pole** locations as shown in the following figure.

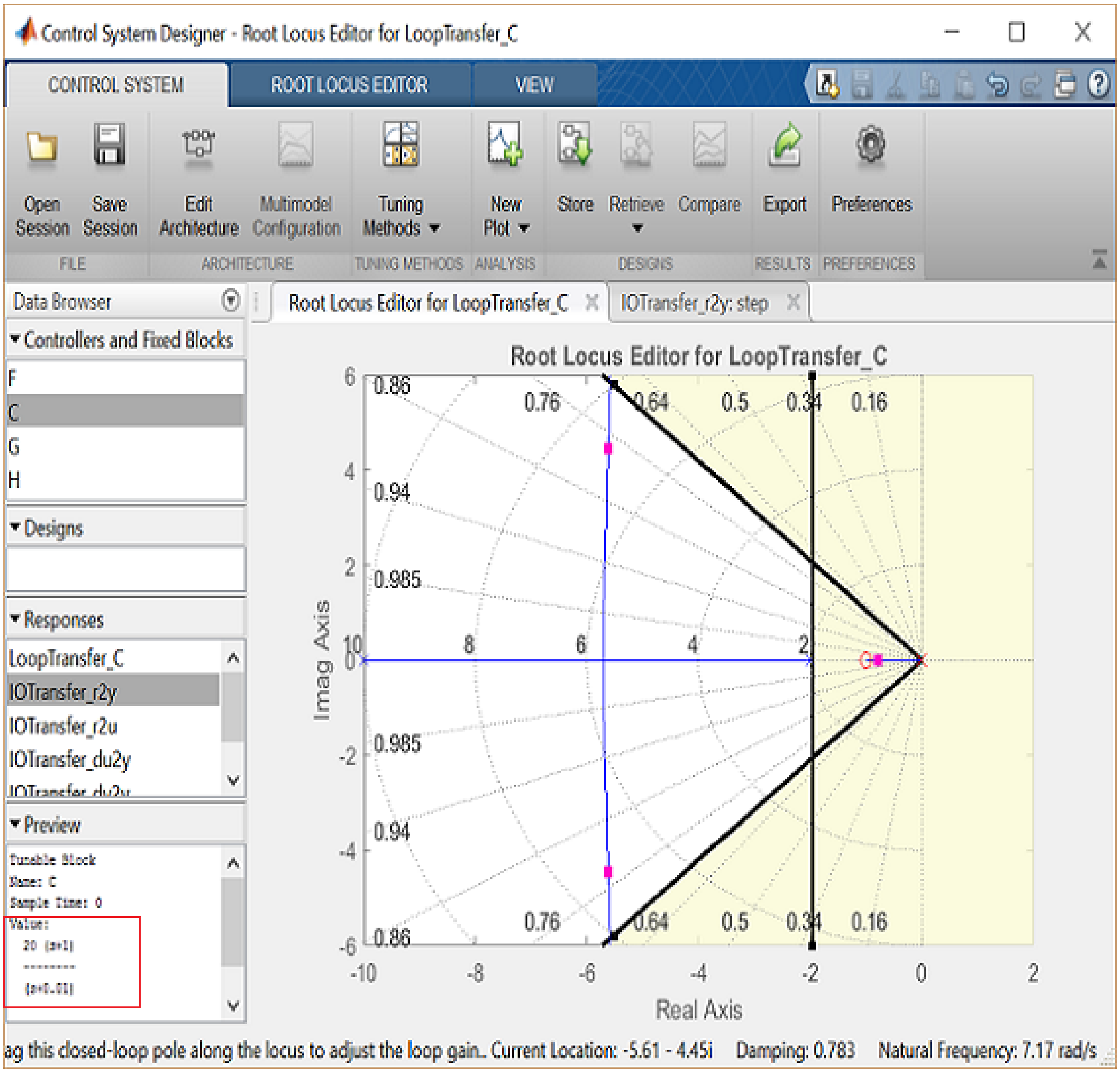


Editing a lag controller

Note that the maximum phase lag contributed by the compensator and the frequency where it is located are updated to match the pole and zero locations chosen.

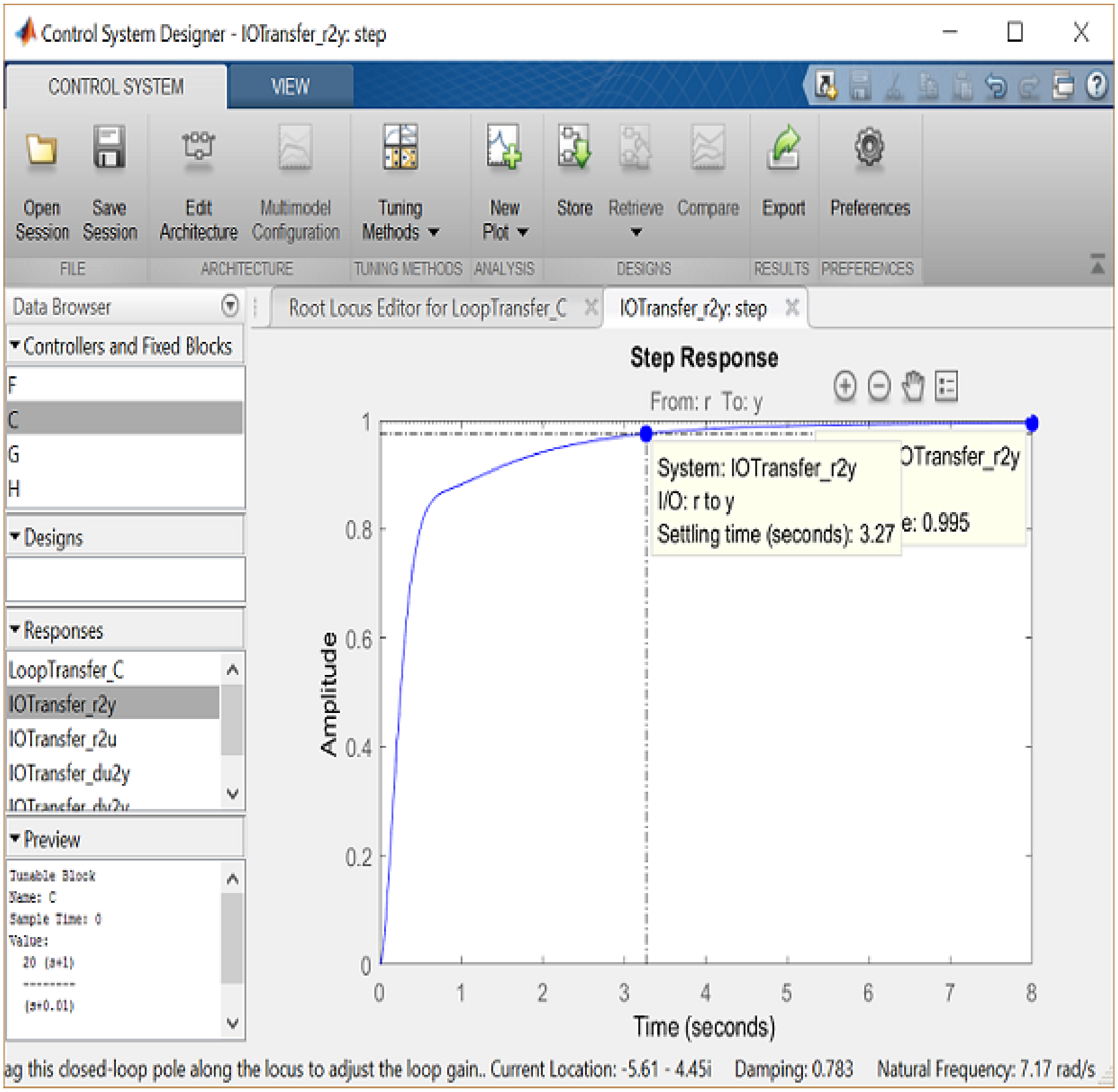
Finding the loop gain with a lag controller

Notice how the root locus has changed to reflect the addition of the pole and zero from the lag compensator as shown in the figure below. We can again choose closed-loop pole locations to attempt to achieve our desired transient requirements. Let's attempt to place two of the closed-loop poles in our desired region near the boundary of the overshoot requirement. For example, a loop gain of approximately 20 (set in the **Compensator Editor**) will place the poles at the positions shown in the figure below.



Choosing poles locations

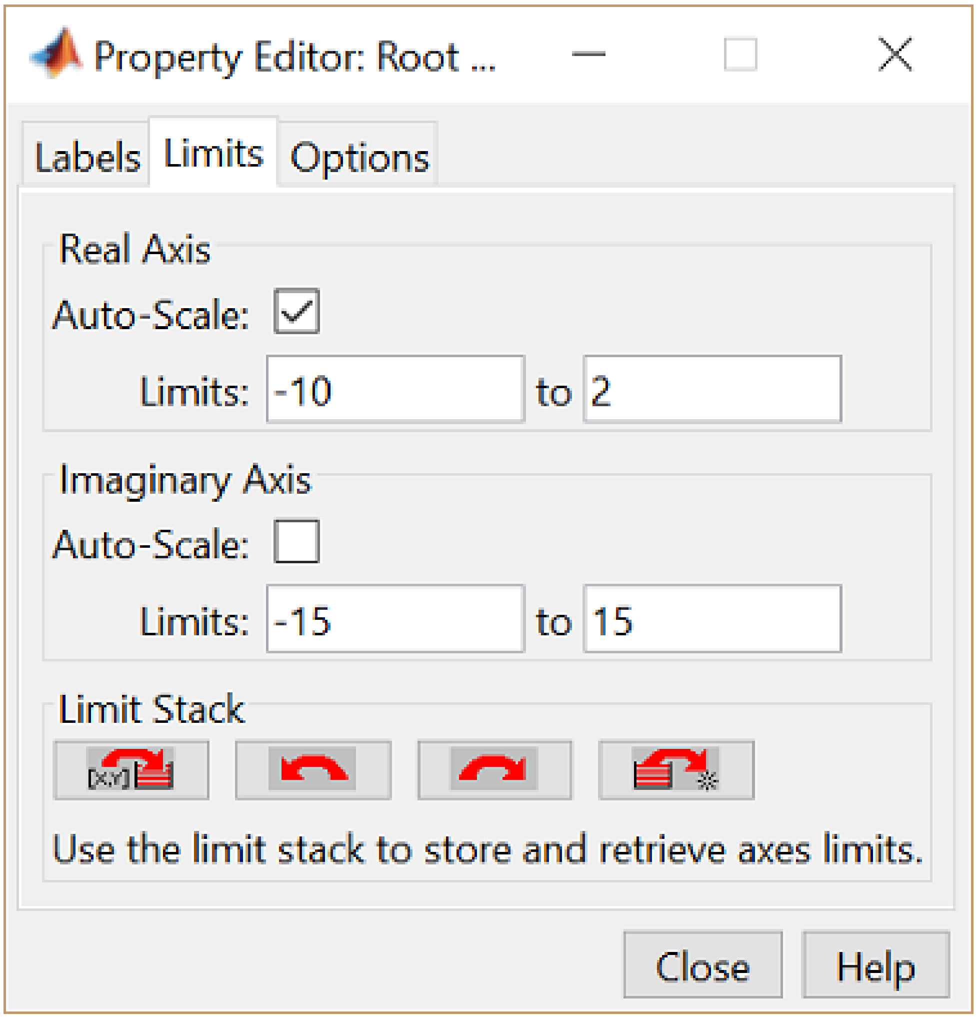
The corresponding closed-loop step response will then update automatically to match the figure shown below.



Corresponding step response

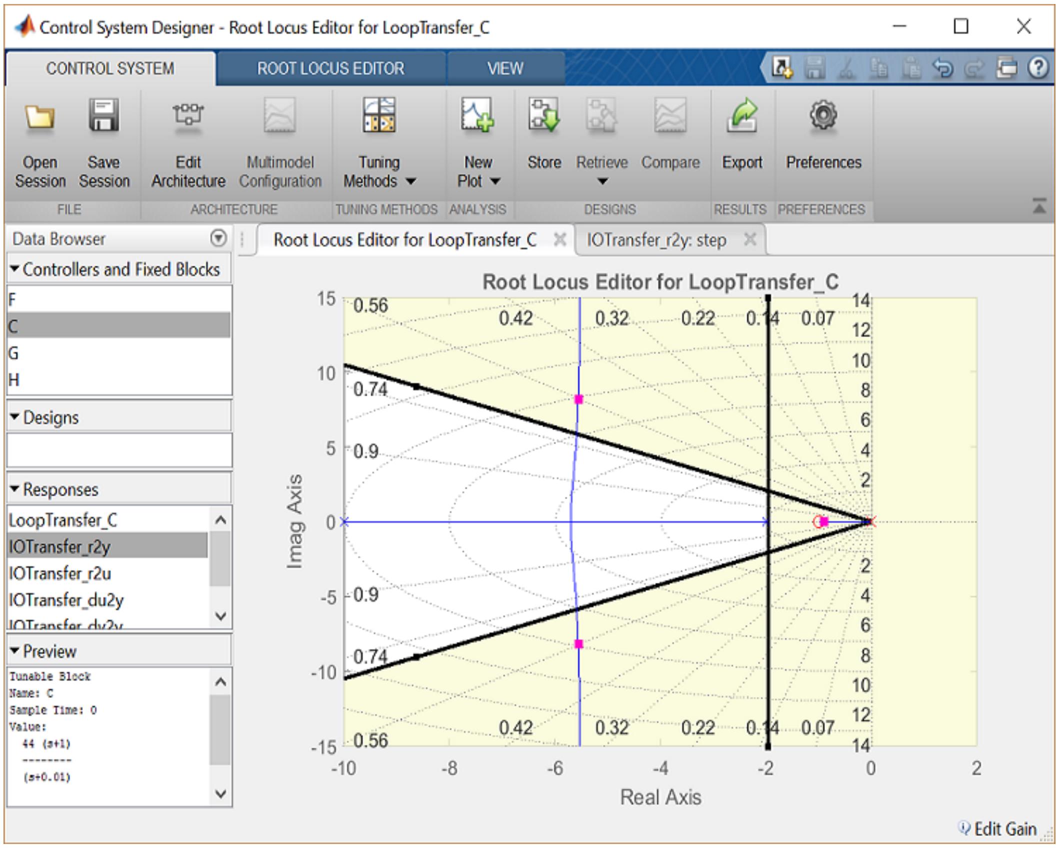
As you can see, the response is not quite satisfactory even though two of the closed-loop poles were placed in the desired region. The reason for this is because the closed-loop system no longer has the form of a canonical second-order system. Specifically, there is a third pole on the real axis indicated in the root locus plot above that is outside of the desired region. The fact that this third pole is to the right of the two conjugate poles placed above means that it will slow the system response down, that is why the settling time requirement is no longer met. Additionally, the overshoot requirement is met easily even though the two conjugate poles are near the edge of the allowed region. This is due again to the third pole which is well damped and tends to dominate the response because it is "slower" than the other poles. What this means is that we can further increase the loop gain such that the conjugate poles move beyond the diagonal lines while still meeting the overshoot requirement.

You can now return to the root locus plot and graphically move the conjugate poles farther away from the real axis; this corresponds to increasing the loop gain. As you move the closed-loop poles a sufficient distance, the limits of the plot should update automatically. Alternatively, you can change the limits manually by right-clicking on the root locus and selecting **Properties** to open the **Property Editor**. Then you can click on the **Limits** tab and change the imaginary axis limits to [-15,15], for example, as shown below.

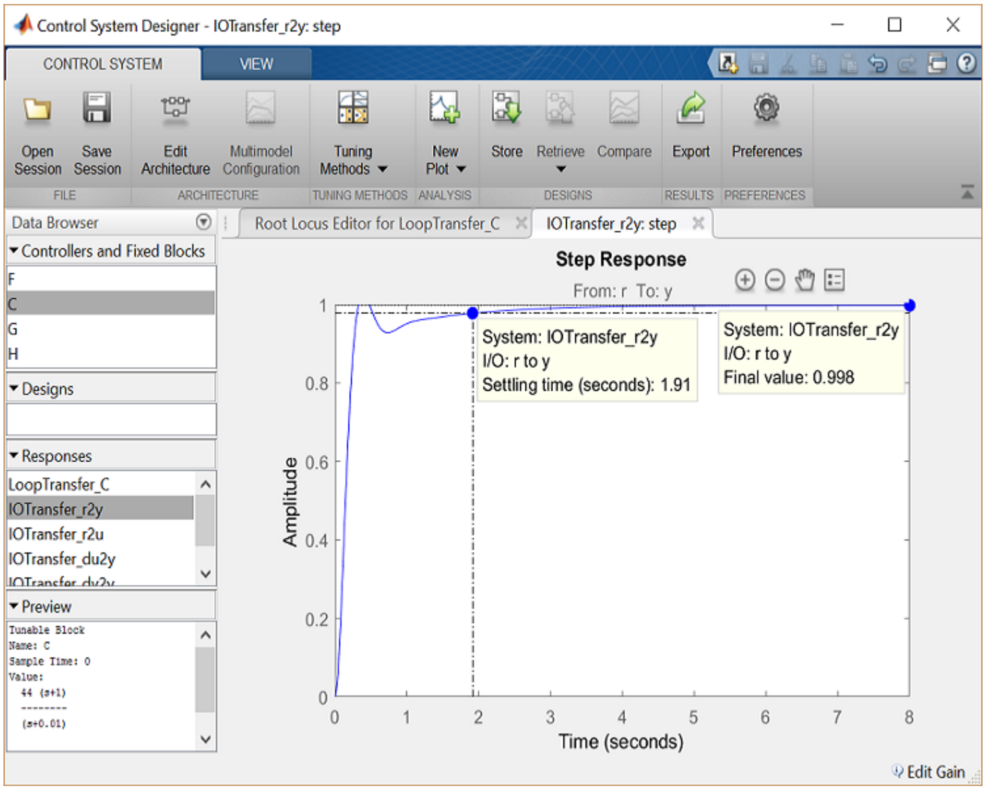


Editing root locus limit

Experiment with different gains (closed-loop pole locations) until you achieve the desired response. Below is the root locus with a loop gain of 44 and the corresponding closed-loop step response.



Tuned root locus

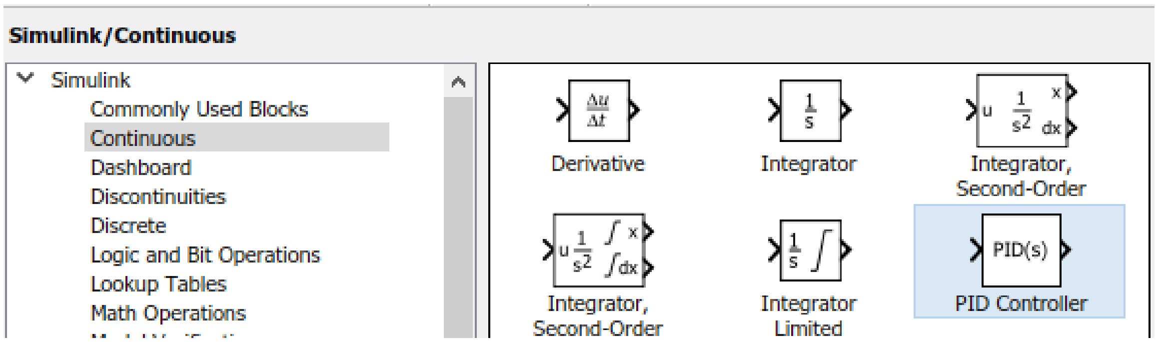


Tuned step response

Now the settling time is less than 2 seconds and the steady-state error and overshoot requirements are still met. As you can see, the root locus design process requires some trial and error. The **Control System Designer** is very helpful in this process. Using the **Control System Designer**, it is very easy to tune your controller and immediately see the effect on the root locus and various analysis plots, like the closed-loop step response. If we had not been able to get a satisfactory response by tuning the loop gain, we could have tried moving the pole and zero of the lag compensator or we could have tried a different type of dynamic compensator (additional poles and/or zeros).

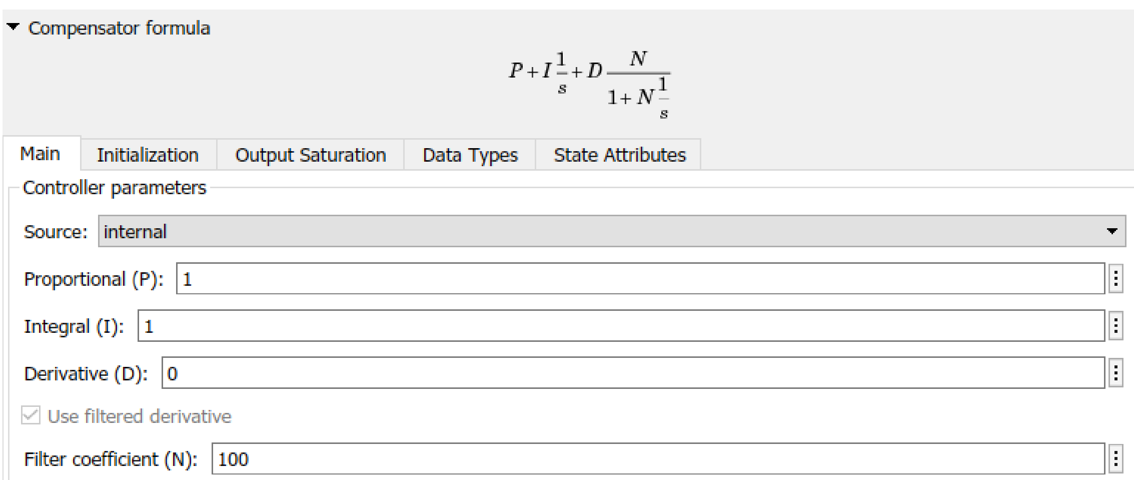
PID Controller Design

The proportional integral derivative controller (PID is the parallel configuration of a proportional (P), an integral (I), and a derivative controller (D). The Simulink provides the PID controller as a block shown in Figure below, where the configuration is shown in Figure below.



The PID controller in Simulink

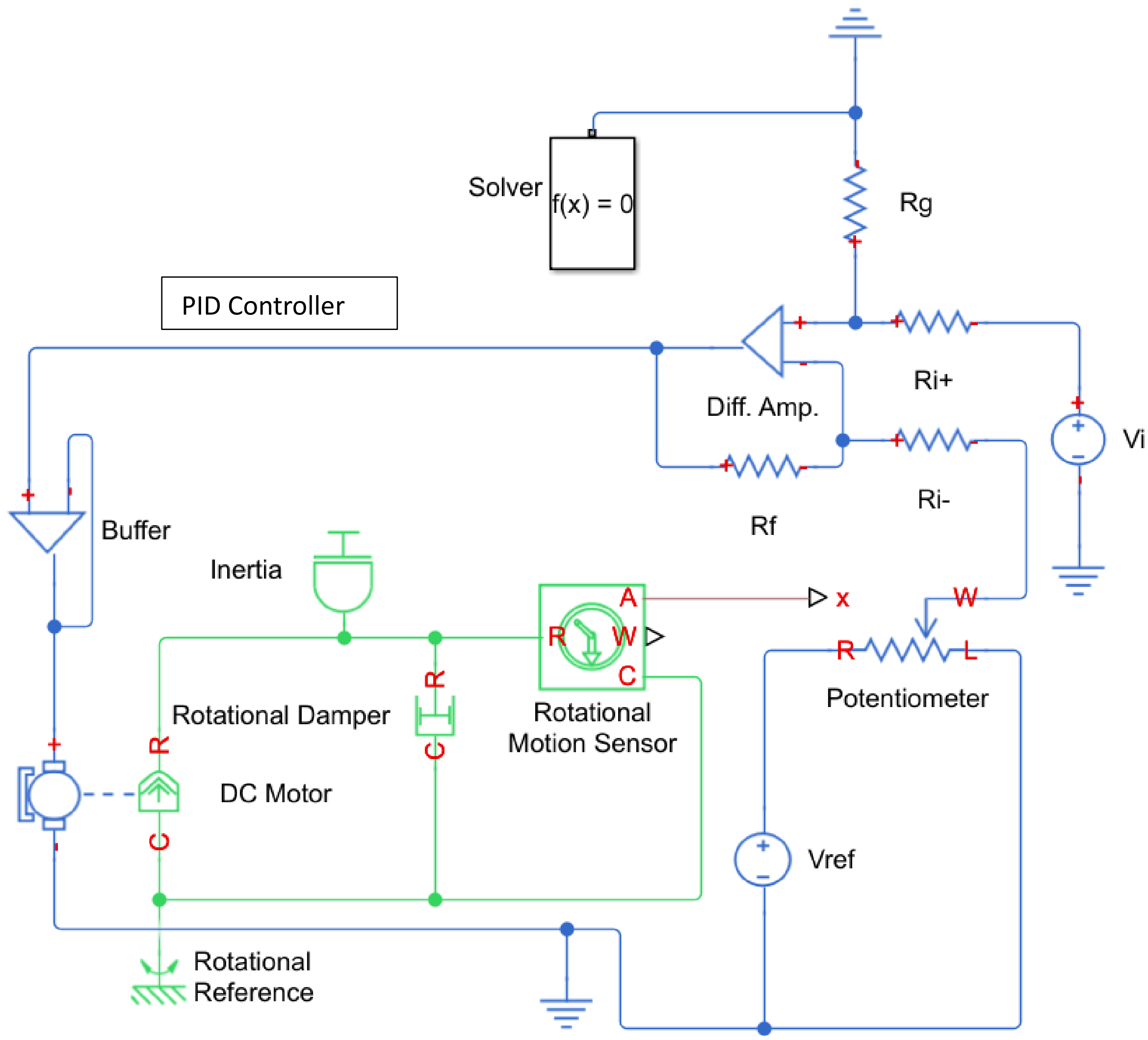
The gain of each parallel controller can be adjusted separately. These gains can be set to zero for the controller-type that is not to use as Figure shows the undeployed derivative controller as its gain is zero. The filter coefficient is specific to the derivative controller and is active if and only if the derivative controller operates.



PID options

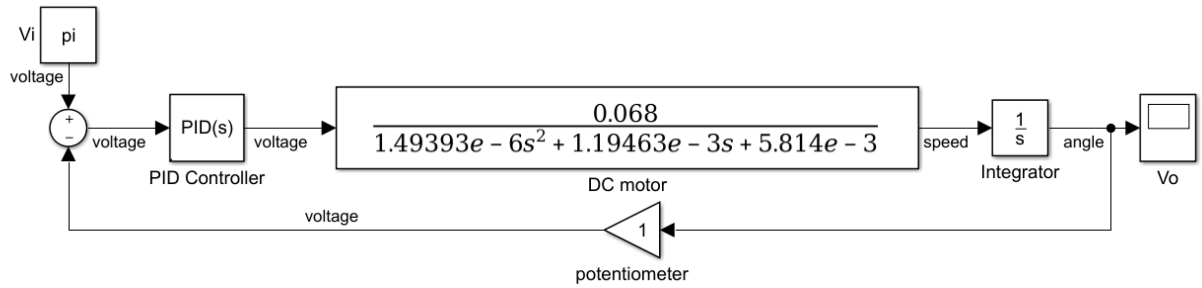
Position Control of a DC Motor

The based problem for the introduction of the PID tuning is the Simscape's model of closed-loop position control of a DC motor loaded by the inertia of $0.01\ kgm^2$ and a rotational damper of $0.01\ Nm/\frac{rad}{s}$ as shown in Figure , where a tuned PID controller is to place between the differential amplifier and the buffer. The voltage source (V_i) input represents the desired angle of movement in radian. The potentiometer provides the unity feedback by converting the angular rotation into the voltage in a 1:1 ratio.

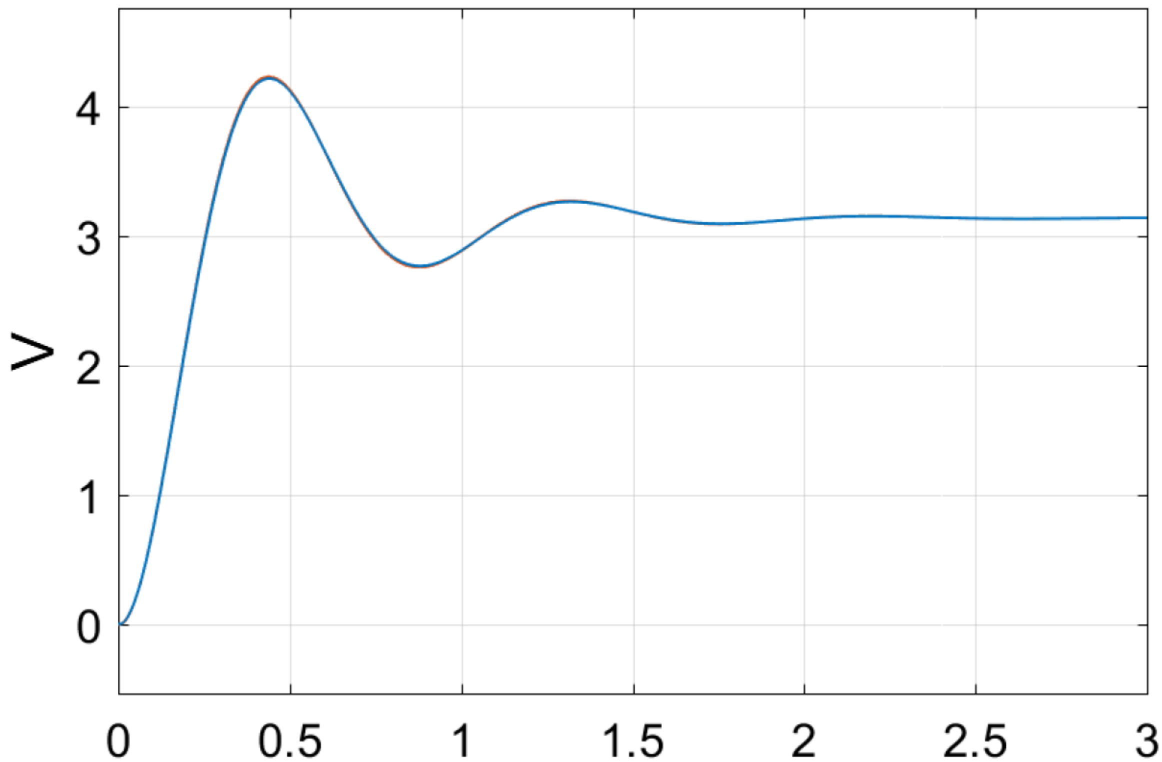


Position control system of a DC motor

For the tuning process, the system of Figure is modelled into the Simulink schematic of Figure , in which the PID controller is not providing any advantage to the response as it is configured as $K_P = 1, K_I = 0$, and $K_D = 0$. The test signal is a step input representing the rotation of π radian, which is emulated by the π volt input of Figure . The simulation results are shown in Figure , where Simulink's and Simscape's models provide the equivalent responses. This setting is the scenario to be tuned through the upcoming techniques.



Equivalent Simulink's schematic



Step response

Ziegler-Nichols Tuning Method

In 1942, the Ziegler-Nichols method was introduced, which became popular for its systematic focus on fast response with reasonable oscillation. In addition, the disturbance rejection of the tuned response is also admirable. The method is a kind of heuristic search that is based on the multiple steps listed as follow:

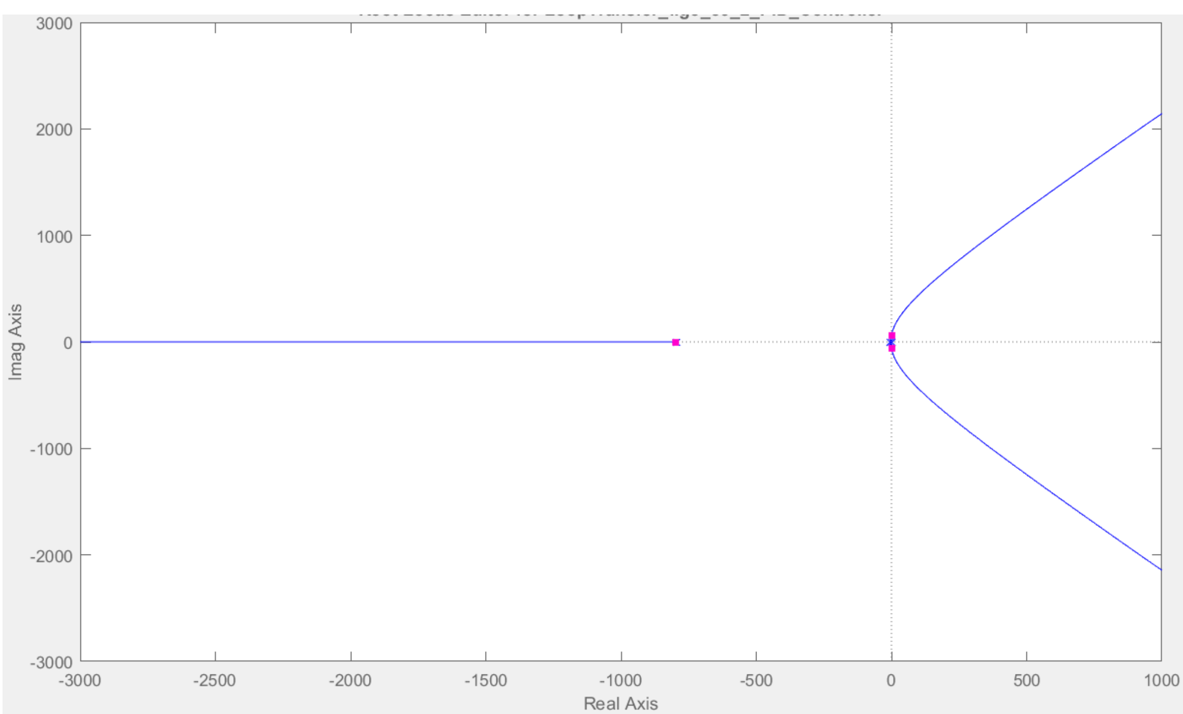
- Set K_I and K_D to 0
- Start with a low level of K_P

- Gradually increase K_P until the system is at the edge of stability
- Records the obtained K_P as K_U, which is meant "the ultimate gain"
- Records the oscillated period based on K_U as T_U
- Select the type of controllers (P, PI, or PID)
- Follow the guideline of K_P, K_I and K_D in Table based on the selected controller

Gains recommendation of Ziegler-Nichols method

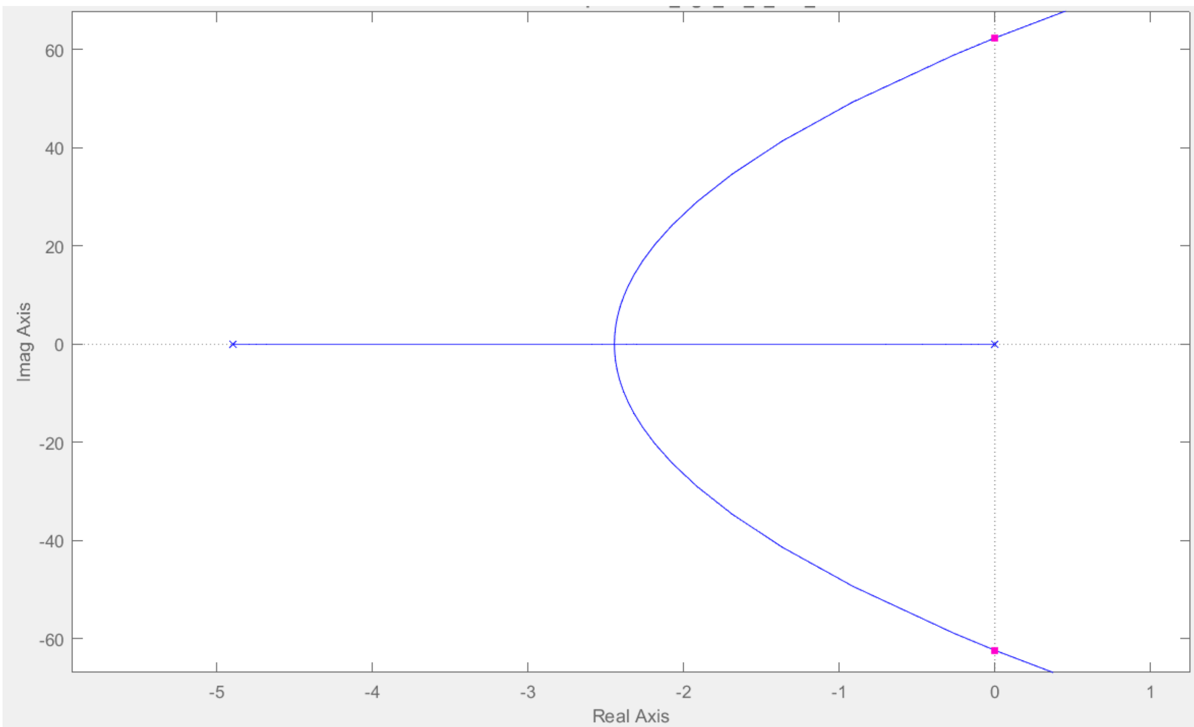
Controller Type	Controller Function	K_P	K_I	K_D
Proportional (P)	$G_c(s) = K_P$	$0.5K_U$		
Proportional Integral (PI)	$G_c(s) = K_P + \frac{K_I}{s}$	$0.45K_U$	$\frac{0.54K_U}{T_U}$	
Proportional Integral Derivative (PID)	$G_c(s) = K_P + \frac{K_I}{s} + K_D s$	$0.6K_U$	$\frac{1.2K_U}{T_U}$	$\frac{0.6K_U T_U}{8}$

The process can be initiated by observing the root locus of the system to indicate the ultimate gain K_U. The overall root locus is displayed in Figure , which is too wide to observe the situation around the origin.



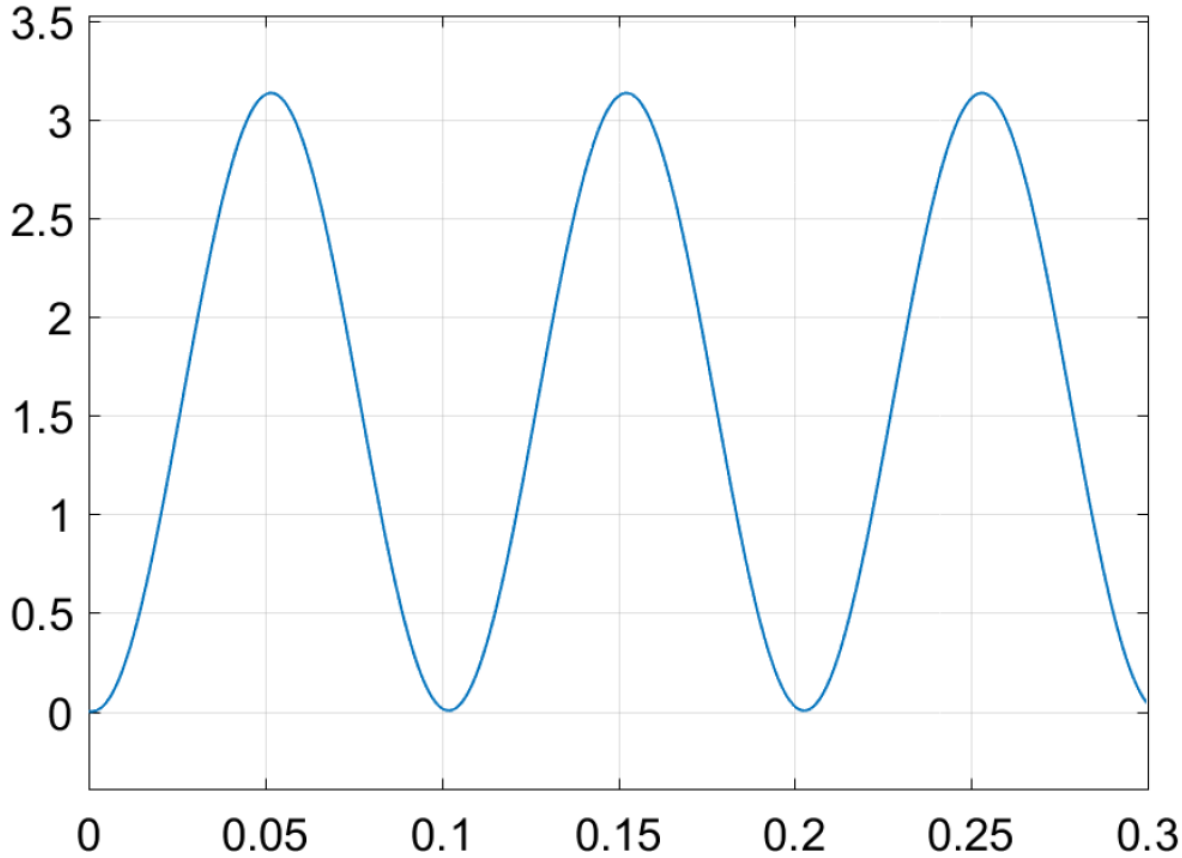
Root locus of the position control of a DC motor

The zoomed version is shown in Figure which clearly indicates the instability issue as the proportional gain K_P increases. The ultimate gain K_U is recorded at 68.37, where the locus of the dominant poles is crossing the imaginary axis.



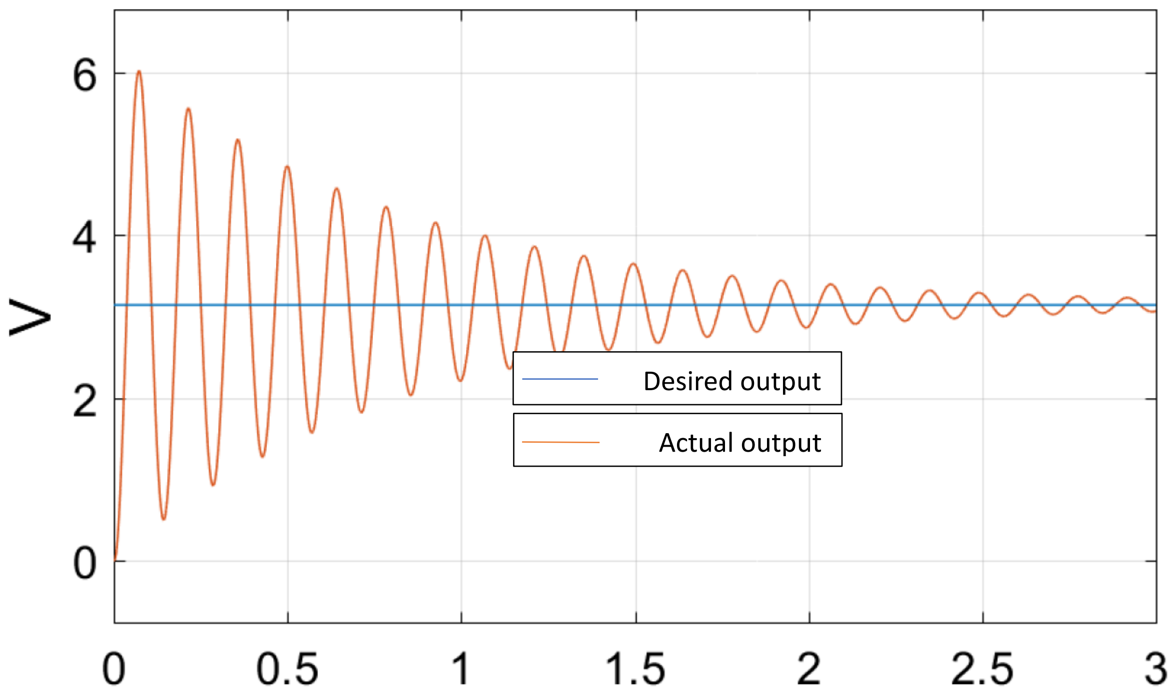
Zoomed root locus

The next is to record the period of oscillation. The step response based on $K_P = 68.37$, $K_I = 0$, and $K_D = 0$ is displayed in Figure , in which the oscillation period T_U is obviously at 0.1 s.



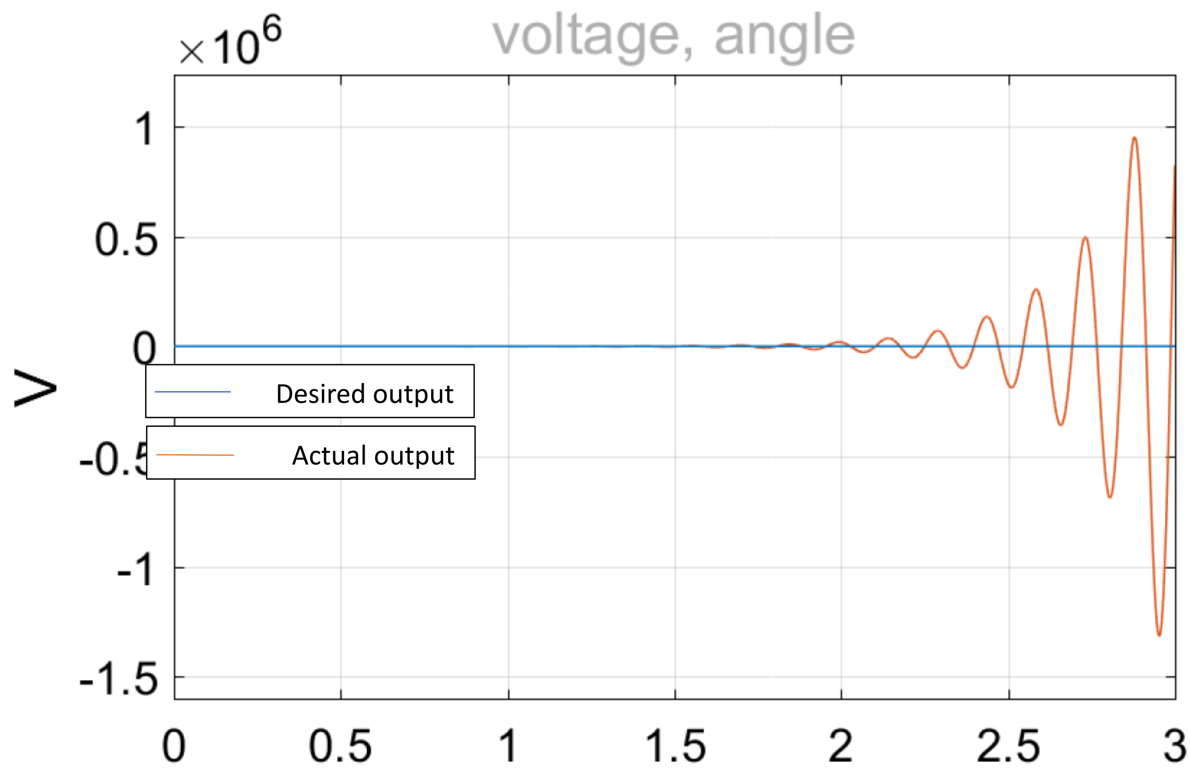
Step response at $K_P = 68.37$

In this case, the input voltage of πV represents the angle position of π radian. Based on the untuned response of Figure , the target of tuning is to obtain a step response of zero steady-state error within 1 second settling time (T_s) and the overshoot less than 3.5V or approximately 11%. A good strategy in PID design is to try the least complicated one first and then move on to the higher complicated version if the current one is not working. Therefore, a proportional controller based on $K_P = 0.5 \times 68.37$ is tried first, and the corresponding step response is shown in Figure , which is clearly not within the requirements of $T_s \leq 1s$ and overshoot less than 3.5V.

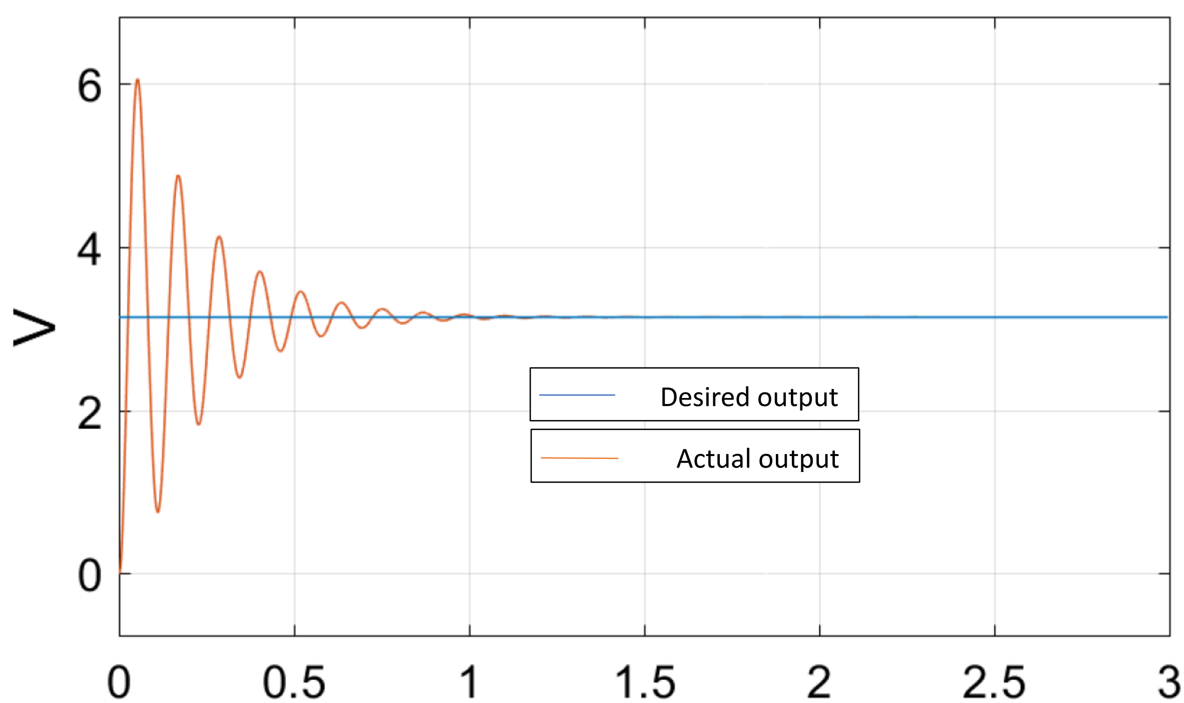


P controlled system of Ziegler-Nichols method

The more complicated PI controller is tried by setting $K_P = 0.45 \times 68.37$ and $K_I = \frac{68.37}{0.1} \times 0.54$. The simulated response is shown in Figure 11.21, which is turned out to be worse than the response of the P-controlled system as the response of the PI-controlled system is unstable. The PID controller is the last one to be tried as the gains are set as $K_P = 0.6 \times 68.37$, $K_I = \frac{68.37}{0.1} \times 1.2$, and $K_D = 68.37 \times 0.1 \times \frac{0.6}{8}$. The simulated response is shown in Figure 11.22, in which the settling time is within the requirement of 1s, but the overshoot is clearly far from expected. This sample shows a scenario that the Ziegler-Nichols method is not able to deliver a required response. However, this does not mean that this tuning method is not good as it has famously adapted for many decades since its introduction. The solution from this approach is a good starting set of gains that can be refined iteratively to obtain the desired response. In addition, the Ziegler-Nichols method is well known for its aggressive characteristics. Therefore, a fast but high overshoot is quite common to the tuned response. By the way, please be reminded that there is no silver bullet to track down every problem as every system has its own characteristics to be treated in customization.



PI controlled system of Ziegler-Nichols method



PID controlled system of Ziegler-Nichols method

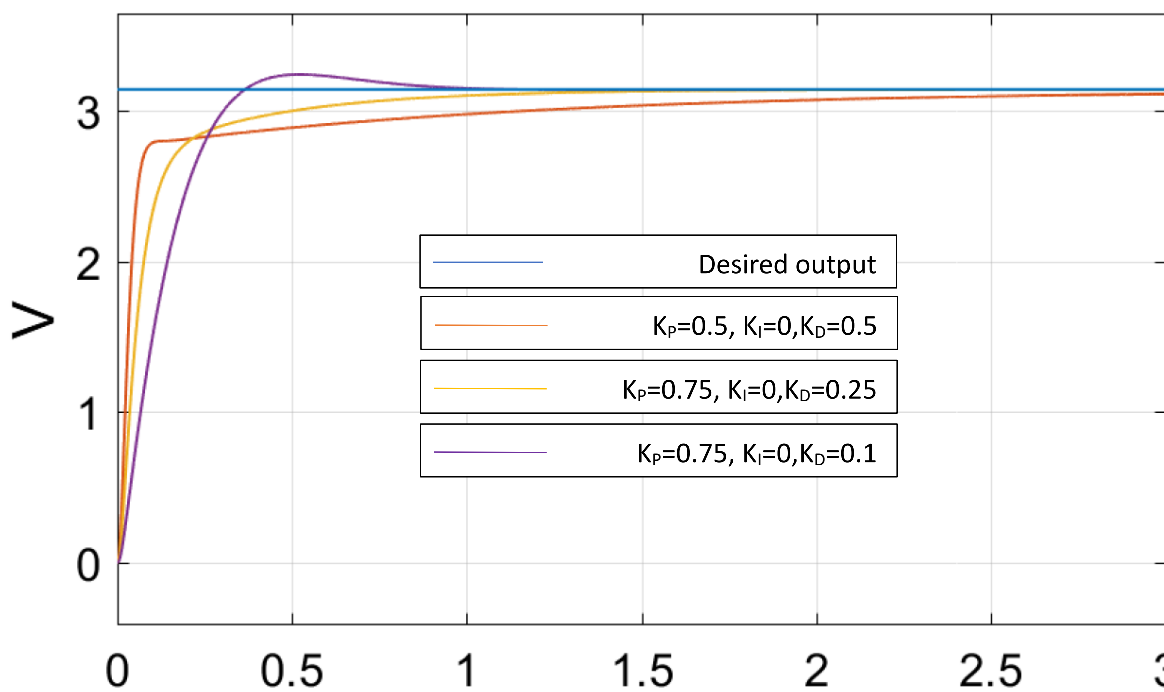
Manual PID Tuning Method

A PID controller can be manually tuned by adjusting K_P, K_I, and K_D based on the guideline shown in Table .

Effects of PID controller based on each parameter

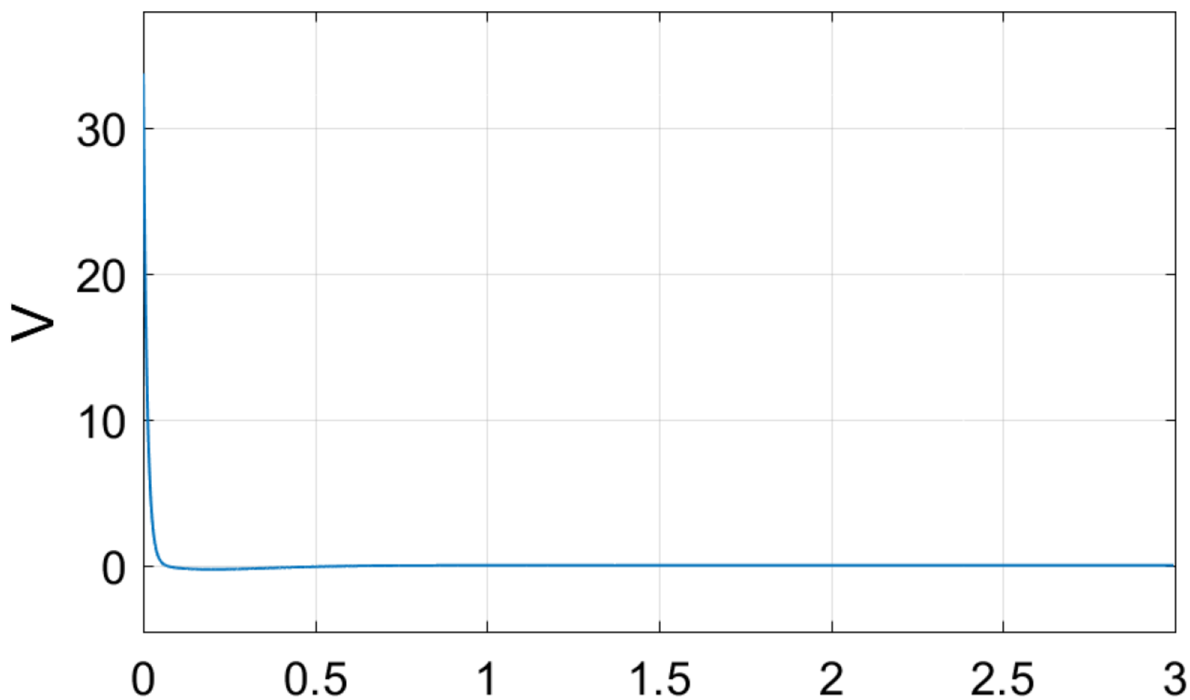
Action	Rise Time	Overshoot	Settling Time	Steady-State error
Increasing K_P	Decrease	Increase	Minimal effect	Decrease
Increasing K_I	Decrease	Increase	Increase	Significantly Decrease
Increasing K_D	Minimal effect	Decrease	Decrease	No effect

Please be noted that the listed guideline is applicable to many systems but not all. Therefore, the designer is recommended to observe the effects and adapt rather than be strict with the guideline. By the way, the guideline of Table will be tested on the position control system of a DC motor in Figure that was tested on the Ziegler-Nichols method. The initial set trial is not to apply the integral controller as the system is already of type-I, in which the steady-state error is zero by default. The parameters to be set are then K_P and K_D, which are tried and simulated as shown in Figure .



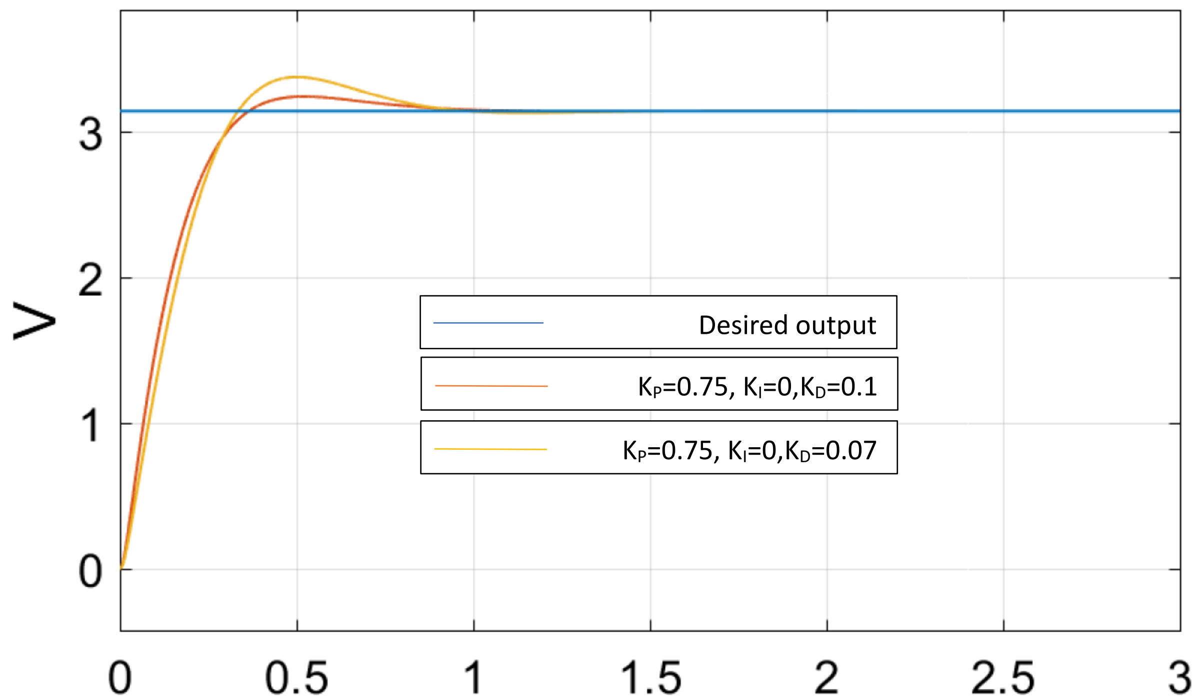
Step responses based on manual tunings

The first trial is based on the small value of K_P and K_D at 0.5. The obtained response is too slow on the settling time and too low on the overshoot. This result clearly implies that K_P is too low and K_D is too high. The second trial is then based on $K_P = 0.75$ and $K_D = 0.25$. The response this time is faster but is still a little bit too slow as the overshoot is too much oppressed. The last trial keeps the K_P at 0.75 while the K_D is reduced again to only 0.1. The obtained response looks good as it passes the settling time requirement of 1 s with a slight overshoot of less than 3.5V. However, if the design of $K_P = 0.75$ and $K_D = 0.1$ is to be used practically, the control signal sent to the motor is observed, which must not be over the motor's specification of 24V. The output of the PID controller is then observed and displayed in Figure, which is maximumly at around 35V. Therefore, this design is good at the model level but may not be suitable for practical implementation.



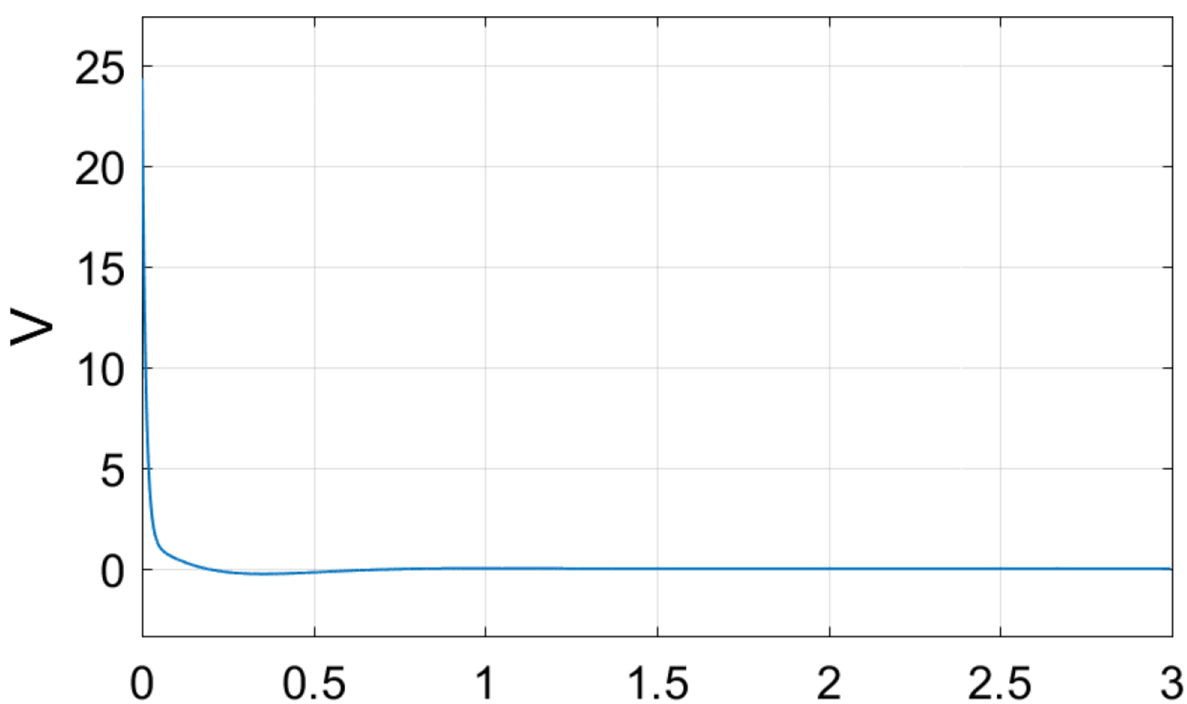
Controlling signal of $K_P = 0.75$ *and* $K_D = 0.1$

It is the act of the differentiator that initially raises the controlling voltage beyond the limitation because it detects the slope of the error and converts it to the corresponding control signal. As the response is rising fast, the error slope is increasing, which will make the differentiator term increase. Therefore, the modification is to slightly decrease the K_D from 0.1 to 0.07, lessening the peak of the controlling signal. The simulated comparison responses are shown in Figure, where the decreased K_D does increase the overshoot and settling time but is still within the requirements.



Tuned responses with different K_D

The reason for this minor tuning is simulated and displayed in Figure, which presents the controlling signal that is to be fed to the DC motor. Based on scarifying the little amount of overshoot and settling time of the response, the peak controlling signal is now within the limit.



Controlling signal based on $K_P = 0.75$ *and* $K_D = 0.07$

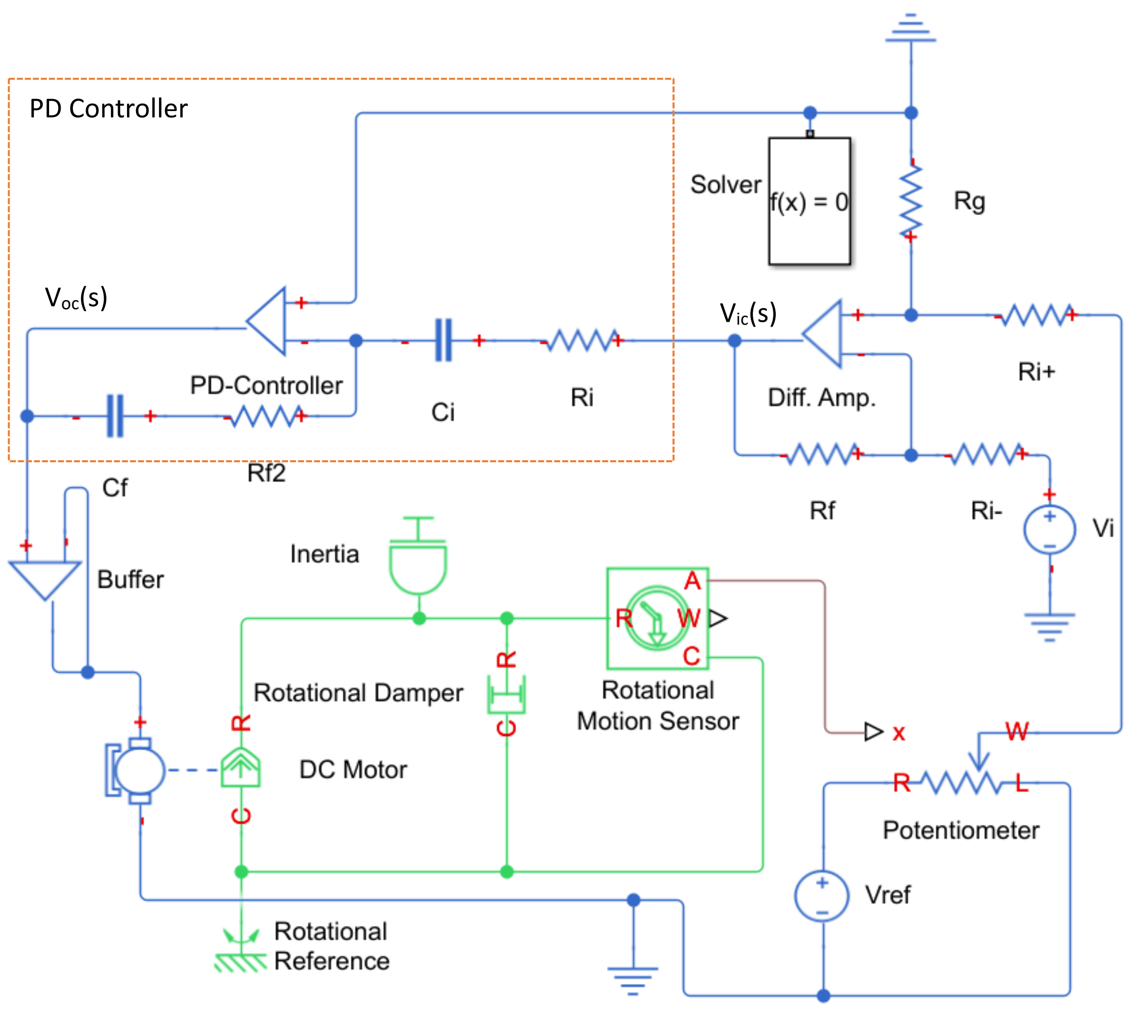
A practical implementation of the finalized design of the PID controller is to be delivered to complete the example. It is actually a PD controller based on $K_P = 0.75$, $K_D = 0.07$, and $N = 100$. The first task is to set the controller's function, which is resulted in:

$$G_c(s) = (0.75 + 0.07 \times 100)\frac{s + \frac{0.75 \times 100}{0.75 + 0.07 \times 100}}{s + 100}$$

Based on the calculations, the PD controller function is reduced to:

$$G_c(s) = 7.75\frac{s + 9.6744}{s + 100}$$

There are choices of practical PID circuits. In this case, a simplified structure is deployed as shown in Figure, which is enough to supply the control function of eq.



Simplified PD controller in application

Based on the circuit's parameters of the deployed controller, its circuit-based function is:

$$\frac{V_{oc}(s)}{V_{ic}(s)} = -\frac{R_f}{R_i}\frac{s+\frac{1}{R_f C_f}}{s+\frac{1}{R_i C_i}}$$

The left tasks are the parameter matching of eq. 11.6 and 11.7. The first set of parameter to be matched is R_i and C_i as:

$$\frac{1}{R_i C_i} = 100$$

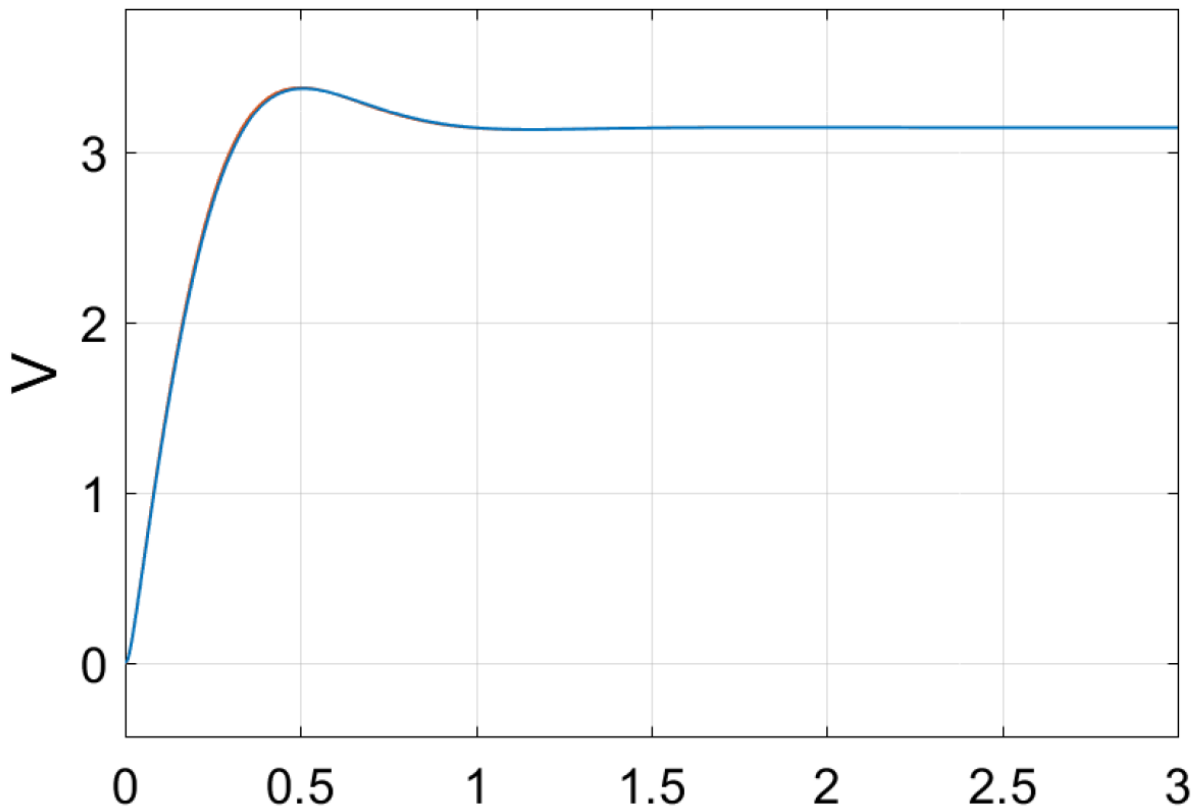
Then, R_i is set to $100k\Omega$, which will make $C_i = 100nF$ for matching the condition of eq. 5.27. Based on $R_i = 100k$, R_f is easily set to $775k\Omega$ according to the condition:

$$\frac{R_f}{R_i} = 7.75$$

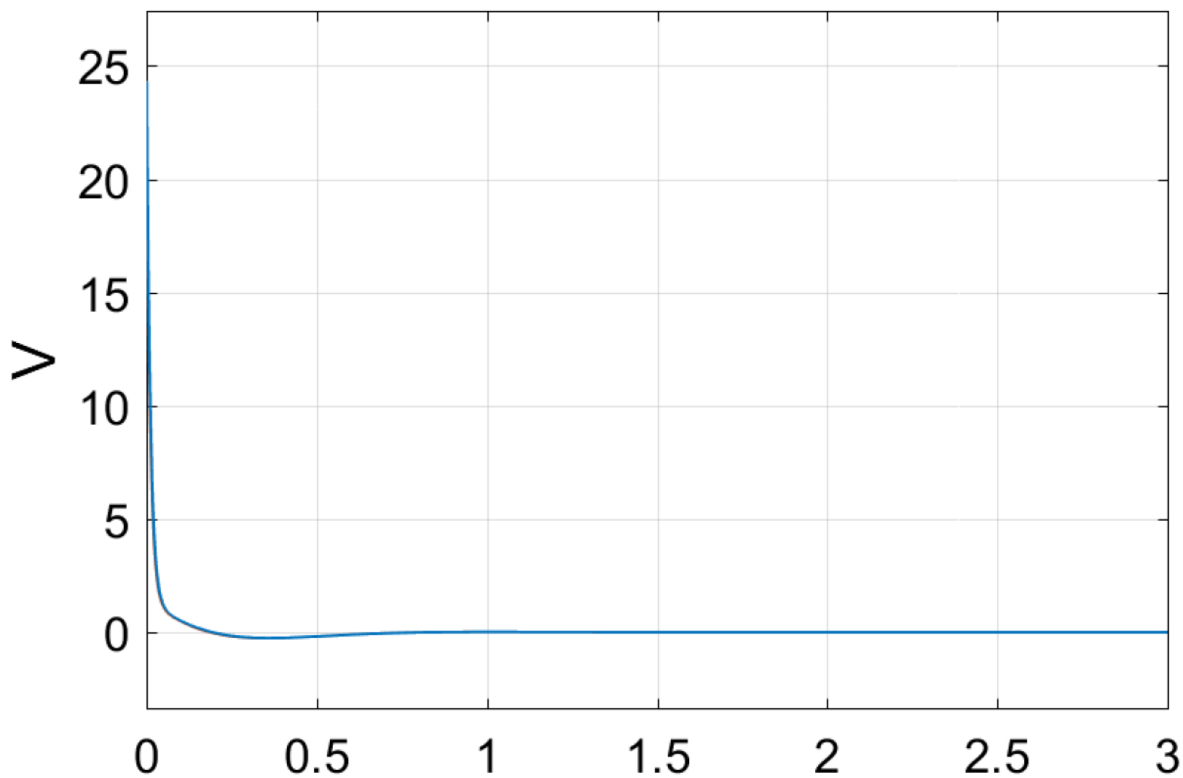
The final circuit parameter to be obtained is the C_f which is based on the condition:

$$\frac{1}{R_f C_f} = 9.6744$$

As R_f is already set to $775k\Omega$, C_f is forced to be $133.37nF$. The circuit parameters of the PD controller are now completely determined and set to Simscape's model of Figure . The simulations are conducted to compare the performance of Simulink's mathematical model and Simscape's practical model. The compared responses are displayed in Figure , indicating the high level of compatibility as the two simulated responses are almost identical. The control signal of Simscape's model is also simulated and compared to the control signal of Simulink's model as displayed in Figure , where they are shown to be compatible and within the limit. By the way, the difference between the simplified PD controller of Figure and the full-scale PD controller previously introduced in Figure is the support of further tuning. The simplified model employs only one op-amp but cannot be tuned easily as circuit parameters are tightly dependent on each other. The full-scale version requires at least three op-amps but can be tuned semi-freely through a potentiometer which is actually a variable resistor. Therefore, the choices of implementation depend on what the designer emphasizes, as there always are pros and cons to be traded off in engineering designs.



Responses based on compatible models



Control signals based on compatible models

It is clear that the tuning of a control system is to be conducted based on multiple concerns as there is no single approach to fit all the problems. The design is often needed to be in the loops of calibration, but the knowledge in guidelines and tools can significantly help decrease the required time compared to the blind search.

Exercise

- Model a centrifugal pump system
  - To deliver water of 200m length and 30m height
  - Desired response is the flowrate
  - Input to the process G(s) is the input voltage of a DC motor that drives the pump
- Design a controller to have zero steady-state error
  - Minimizing the settling time

Lecture 03: Intelligent algorithms for control system

2 sessions, 4 hrs

Lesson 3: Artificial Neural Network Controller

Artificial Neural Network (ANN) is a collection of elements called 'Neuron' in a systematic way. A neuron is a simulation of a biological neuron in a human nervous system. An artificial neural network is inspired by the human nervous system. A biological neuron is a building block of a human nervous system. It can take multiple inputs at a time through sensory inputs (like 5 senses: touch, see, smell, hear and taste) and other neurons, process the inputs through its nucleus, and outputs the processed information, if it is significant. See Figure 3.1a for the basic structure of a biological neuron. A single neuron is not able to contribute much in overall decision-making processes, however, the ability to decision making and thoughtfulness comes from the ability to work in parallel. There are billions and trillions of neurons in a human nervous system, which are working in a parallel manner; each in an independent way. Since they are too many, if some of them are notworking, the network performance is not affected much. Hence, such a structure works in a parallel but asynchronous way and offers a high level of fault tolerance with distributed control. As per the famous saying, 'I think therefore I am' by Rene Descartes (1637), intelligence results from the ability to think. If such a concept is incorporated within a machine, would the machine be able to think or not is the basic inspiration behind the artificial neural network. Figure 3.1b presents a simulation of a neuron, known as an artificial neuron. As stated, the ability to think and hence intelligent decision making comes from the collection of a large number of neurons working together towards a global solution. For this, the neurons should be arranged in a proper structure or topology. Hopfield network, perception, and multilayer perceptron with their variations, and self-organizing maps are a few prominent neural network topologies to experiment with artificial neural networks. The following section describes these artificial neural network models in brief.

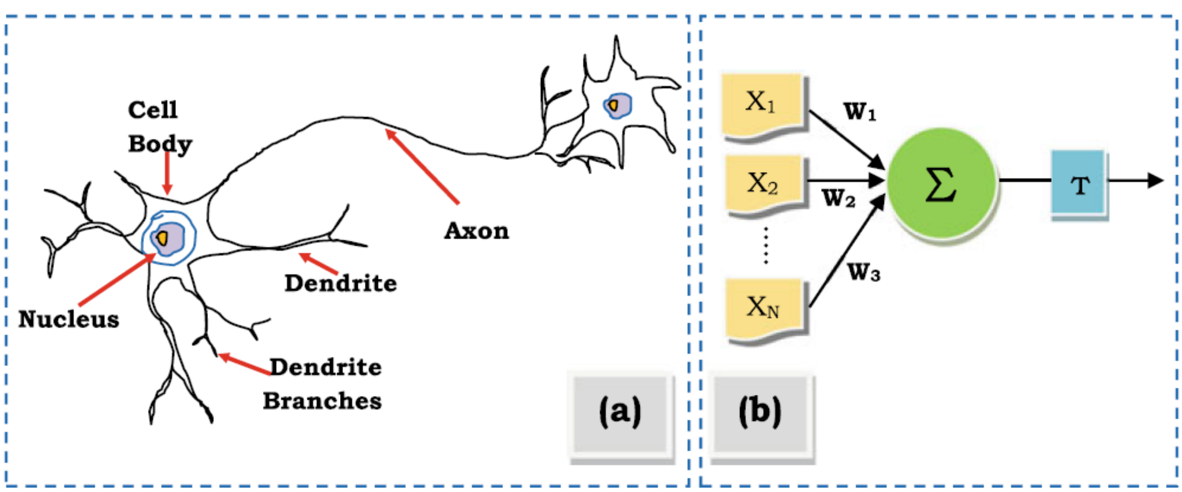


Figure 3.1 Biological and artificial neuron

Single Perceptron

Frank Rosenblatt (1958) proposed an artificial neuron inspired by the human nervous system between the eyes and the brain. The main objective of the group of neurons between the eye and brain is to perceive the image acquired. Hence, the name of the suggested model of a neuron is 'Perceptron'. As per the proposed model, the perceptron welcomes multiple inputs, which are weighted in nature. Beside the weighted inputs, the perceptron also has a processing function, called activation or transfer function. The activation function processes the acquired inputs. There is a threshold function along with a neuron which determines the further action of the neuron. If the processed value generated from the weighted inputs to the neuron is significant enough, the neuron communicates the output by sending it further–to other neurons, if any. This phenomenon is called firing the output i.e. a perceptron's basic function is to collect several weighted inputs, process it, and fires the processed value further if the processed value is significant. As a perceptron can choose to fire the output further or not; it is used to divide the problem space into two classes. Hence, all linear- kind of '*to be or not to be*' type of problems can be effectively solved with a single perceptron. Figure 3.2 illustrates a single perceptron solving a problem of selection of a course for a student to study further based on inputs of parents. Figure 3.2 illustrates a single perceptron that solves a classical '*to be or not to be*' problem for the selection of a course.

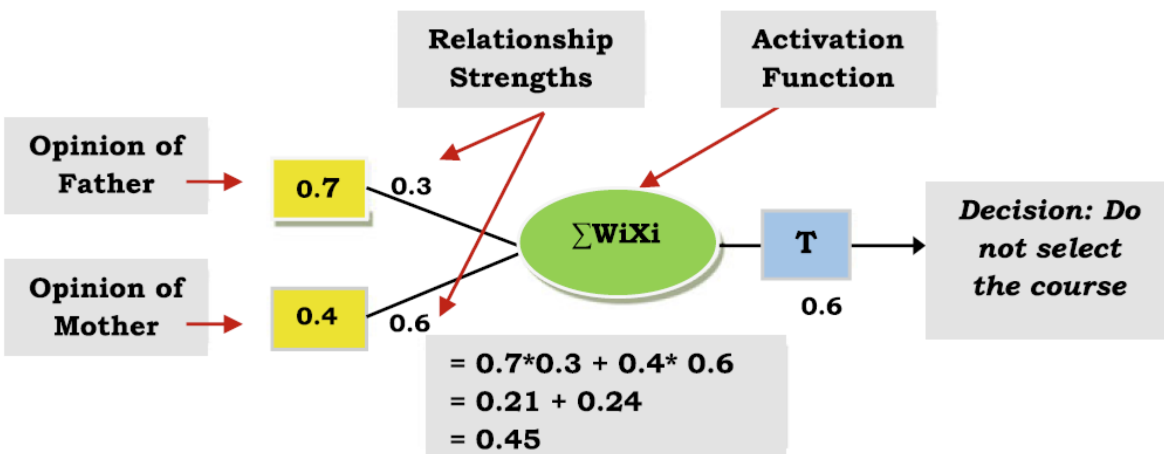


Figure 3.2 Selection of a course

As shown in the figure the neuron takes two inputs from both the parents about a possible course and processes it through an activation function available in the nucleus of the neuron. Here, the activation function is ∑WiXi. Parents opinions are encoded as X1 (0.7) and X2 (0.4). The relationship strengths of the candidate who wants to select the course with the parents are given asW1 (0.3) andW2 (0.6) for both the parents respectively. As per the sample values shown in the figure, the activation function calculates the value 0.45, which is less than the threshold value 0.6. Hence, the neuron decides not to fire, and the course is not selected. Different weights and values of parents' opinions can be tried for a better understanding of the problem. This perceptron classifies the problem into two categories to select or not to select the course; hence, called the linearly separable

problem. In the case of linearly separable problems, data are usually separated by a line (hyper-plane in an advanced dimension).
A generic learning mechanism for a linearly separable problem, which is also known as 'fixed increment perceptron learning' can be given as shown in the following code.

1. Let x(n) = input vector given as {+1, x1(n), x2(n), …,xm(n)}
   w(n) = weight vector as {b, w1(n), w2(n), …, wm(n)}
   b = bias
   a(n) = actual response
   r(n) = desired response
   l = learning rate parameter
2. Initialize w(0) = 0
3. Activate perceptron by applying input vector x(n) and desired response r(n)
4. Compute actual response of perceptron a(n) = f[w(n)x(n)]
5. If r(n) and a(n) are different then w(n + 1) = w(n) + l[d(n)-a(n)]x(n), where r(n)= ±1
6. Go to step 3 till all patterns properly classified

In the case of a popular tool called Support Vector Machine, the same strategy is implemented. The support vector machine is a simple neural network model that considers the given data and tries to classify them into two classes. Through the SVM, classification is done in the presence of data.

Multilayer Perceptron

As stated, a single perceptron cannot solve non-linearly separable problems, which are much complicated but fall into real-life problems category; hence, a multilayer perceptron structure is proposed. The structure is illustrated in Figure 3.3.

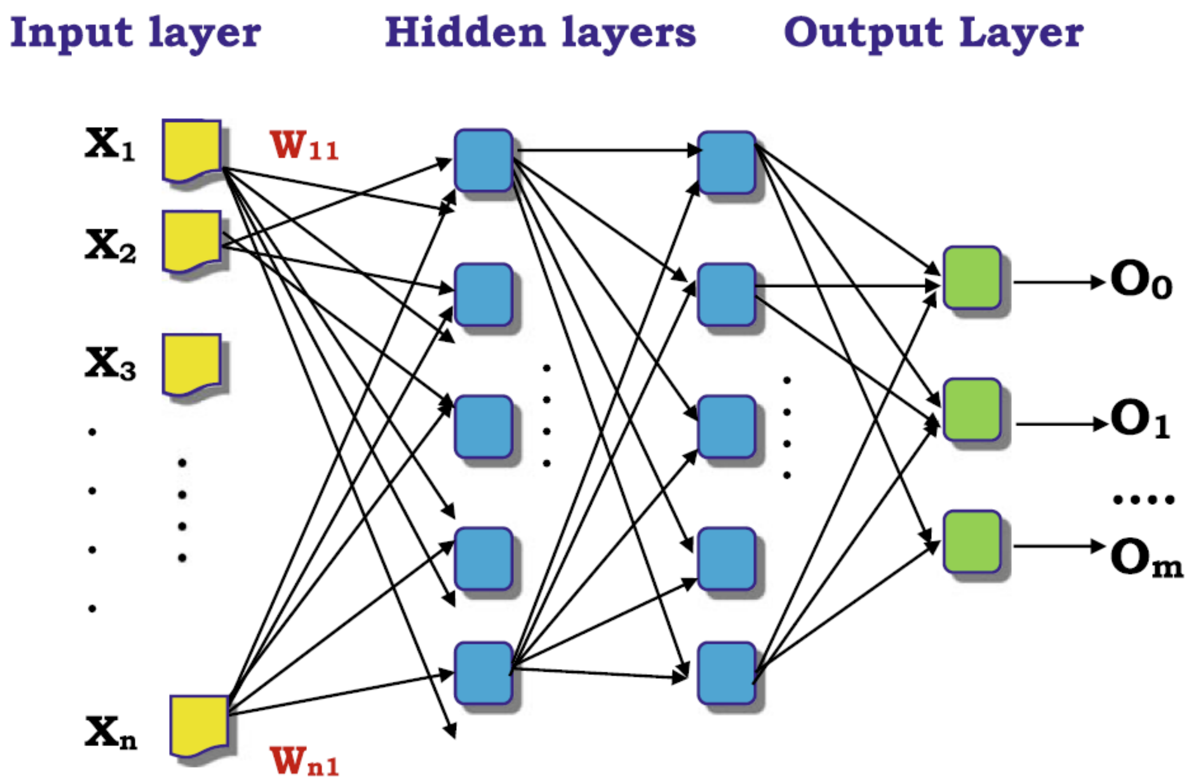


Figure 3.3 Multilayer perceptron

In the multilayer perceptron, neurons are arranged in different layers. These neurons are often called nodes. These layers are called the input layer, hidden layer, and output layer. The input layer is responsible for collecting input from the environment. The output layer produces the output. Hidden layers are invisible, and help in learning. At least one layer of each category is required to form a multilayer neural network. Hidden layers are generally many; in case of simple multilayer perceptron there may be one, two, or three hidden layers. However, in the case of deep learning, there are multiple hidden layers. Typically, the input and output layers are one each. Depending on applications more than one input and more than one output layers can be planned. The following algorithm illustrates popular heuristics to design a multilayer perceptron.

1. Verify the nature of the problem. Typically where many data are available but there is a lack of generalized logic, one may go for multilayer perceptron ANN
2. Select critical parameters that play an important role in decision making. For this, one needs to study the data available on hand. Alternatively, a few successful cases where such decisions are made can be considered. Total number of such important and critical parameters is, say 'n'

3. Create an input layer (I) containing ‘n’ number of neurons. Also, assign its activation function as the value of the input
4. Identify possible choices/output options for the problem. Say this number is ‘m’
5. Create one or two hidden layers (H1 and H2) containing an average of input and output number of nodes; that is (‘n’ + ‘m’)/2. Assign activation function in each neuron of every layer. Typical activation functions are softmax, sigmoid, hyperbolic tangent, rectified linear activation unit, etc. The activation function at the first hidden layer involves input values from the input layer nodes with their weights. The activation function at the second hidden layer involves the previous layer (hidden) nodes’ values with their weights
6. Create an output layer (O) containing ‘m’ number of nodes. Assign an output activation function to each neuron in the output layer. The activation function at the output layer involves values from the last hidden layer nodes with their weights
7. Connect all neurons in such a way that ‘each neuron is connected in a forward direction to every neuron of the adjacent layer’. This makes the network fully connected, feed-forward (as all the connections are in a forward direction only) multilayer neural network
8. Assign random weights to each connection
9. Train the network with collected valid data sets

Design NARMA-L2 Neural Controller in Simulink

The neurocontroller described in this section is referred to by two different names: feedback linearization control and NARMA-L2 control. It is referred to as feedback linearization when the plant model has a particular form (companion form). It is referred to as NARMA-L2 control when the plant model can be approximated by the same form. The central idea of this type of control is to transform nonlinear system dynamics into linear dynamics by canceling the nonlinearities. This section begins by presenting the companion form system model and showing how you can use a neural network to identify this model. Then it describes how the identified neural network model can be used to develop a controller. This is followed by an example of how to use the NARMA-L2 Control block, which is contained in the Deep Learning Toolbox™ blockset.

Identification of the NARMA-L2 Model

As with model predictive control, the first step in using feedback linearization (or NARMA-L2) control is to identify the system to be controlled. You train a neural network to represent the forward dynamics of the system. The first step is to choose a model structure to use. One standard model that is used to represent general discrete-time nonlinear systems is the nonlinear autoregressive-moving average (NARMA) model:

$$y(k+d) = N[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-n+1)] \quad (3.1)$$

where $u(k)$ is the system input, and $y(k)$ is the system output. For the identification phase, you could train a neural network to approximate the nonlinear function N. This is the identification procedure used for the NN Predictive Controller.

If you want the system output to follow some reference trajectory $y(k+d) = y_r(k+d)$, the next step is to develop a nonlinear controller of the form:

$$u(k) = G[y(k), y(k-1), \ldots, y(k-n+1), y_r(k+d), u(k-1), \ldots, u(k-m+1)] \quad (3.2)$$

The problem with using this controller is that if you want to train a neural network to create the function G to minimize mean square error, you need to use dynamic backpropagation. This can be quite slow. One solution, proposed by Narendra and Mukhopadhyay, is to use approximate models to represent the system. The controller used in this section is based on the NARMA-L2 approximate model:

$$\begin{aligned}\hat{y}(k+d) &= f[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-m+1)] \\ &+ g[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-m+1)] \cdot u(k)\end{aligned} \quad (3.3)$$

This model is in companion form, where the next controller input $u(k)$ is not contained inside the nonlinearity. The advantage of this form is that you can solve for the control input that causes the system output to follow the reference $y(k+d) = y_r(k+d)$. The resulting controller would have the form

$$u(k) = \frac{y_r(k+d) - f[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-n+1)]}{g[y(k), y(k-1), \ldots, y(k-n+1), u(k-1), \ldots, u(k-n+1)]} \quad (3.4)$$

Using this equation directly can cause realization problems, because you must determine the control input $u(k)$ based on the output at the same time, $y(k)$. So, instead, use the model

$$\begin{aligned}y(k+d) &= f[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-n+1)] \\ &+ g[y(k), \ldots, y(k-n+1), u(k), \ldots, u(k-n+1)] \cdot u(k+1)\end{aligned} \quad (3.5)$$

where $d \geq 2$. The following figure shows the structure of a neural network representation.

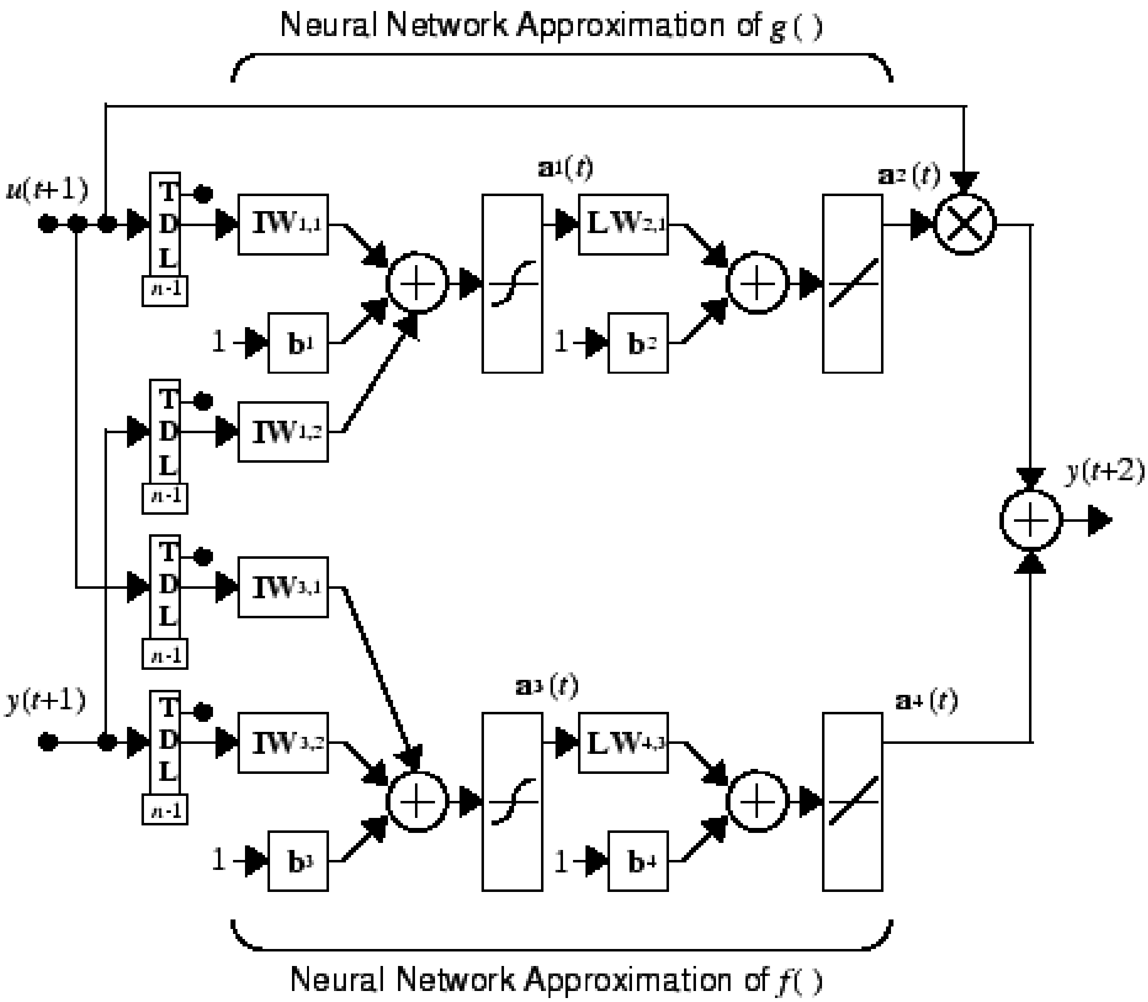


Figure 3.4 Structure of a neural network representation

NARMA-L2 Controller

Using the NARMA-L2 model, you can obtain the controller

$$u(k+1) = \frac{y_r(k+d) - f[y(k), \ldots, y(k-n+1), u(k), \ldots, u(k-n+1)]}{g[y(k), \ldots, y(k-n+1), u(k), \ldots, u(k-n+1)]} \quad (3.6)$$

which is realizable for $d \geq 2$. The following figure is a block diagram of the NARMA-L2 controller.

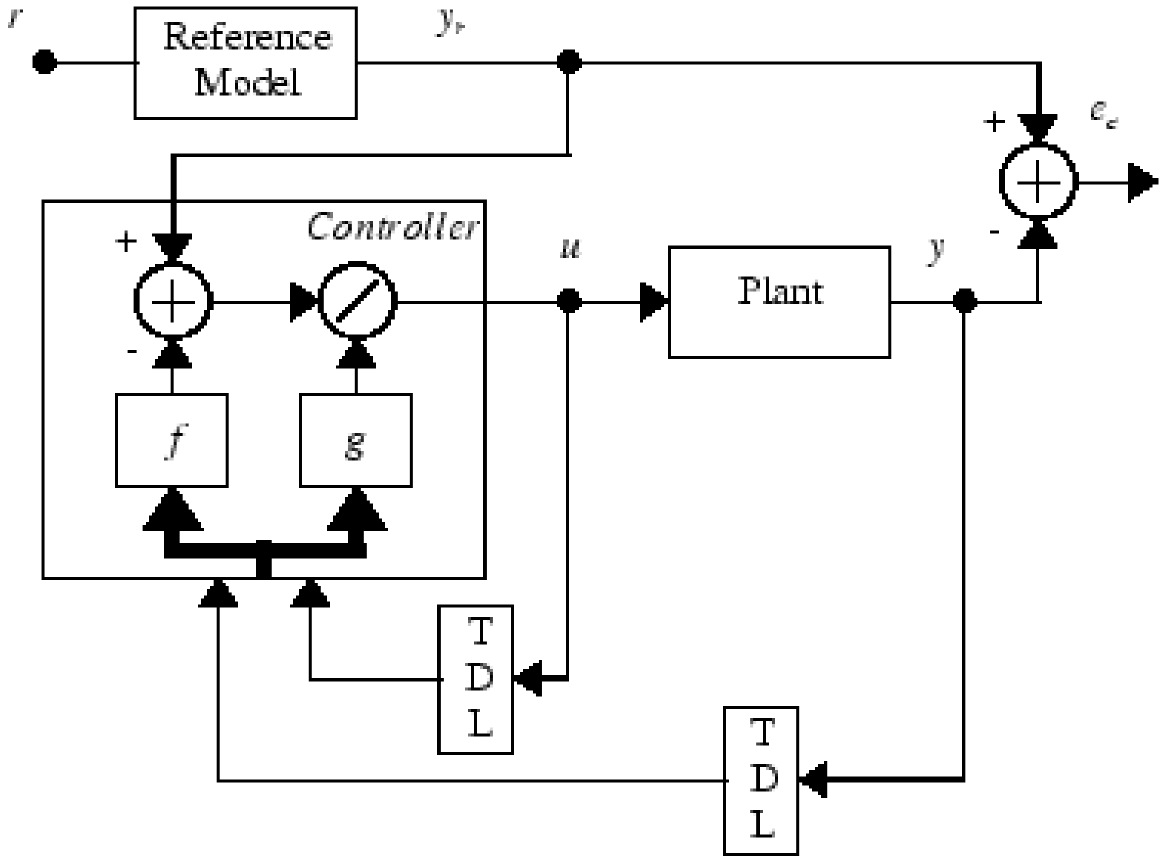


Figure 3.5 Bock diagram of the NARMA-L2 controller

This controller can be implemented with the previously identified NARMA-L2 plant model, as shown in the following figure.

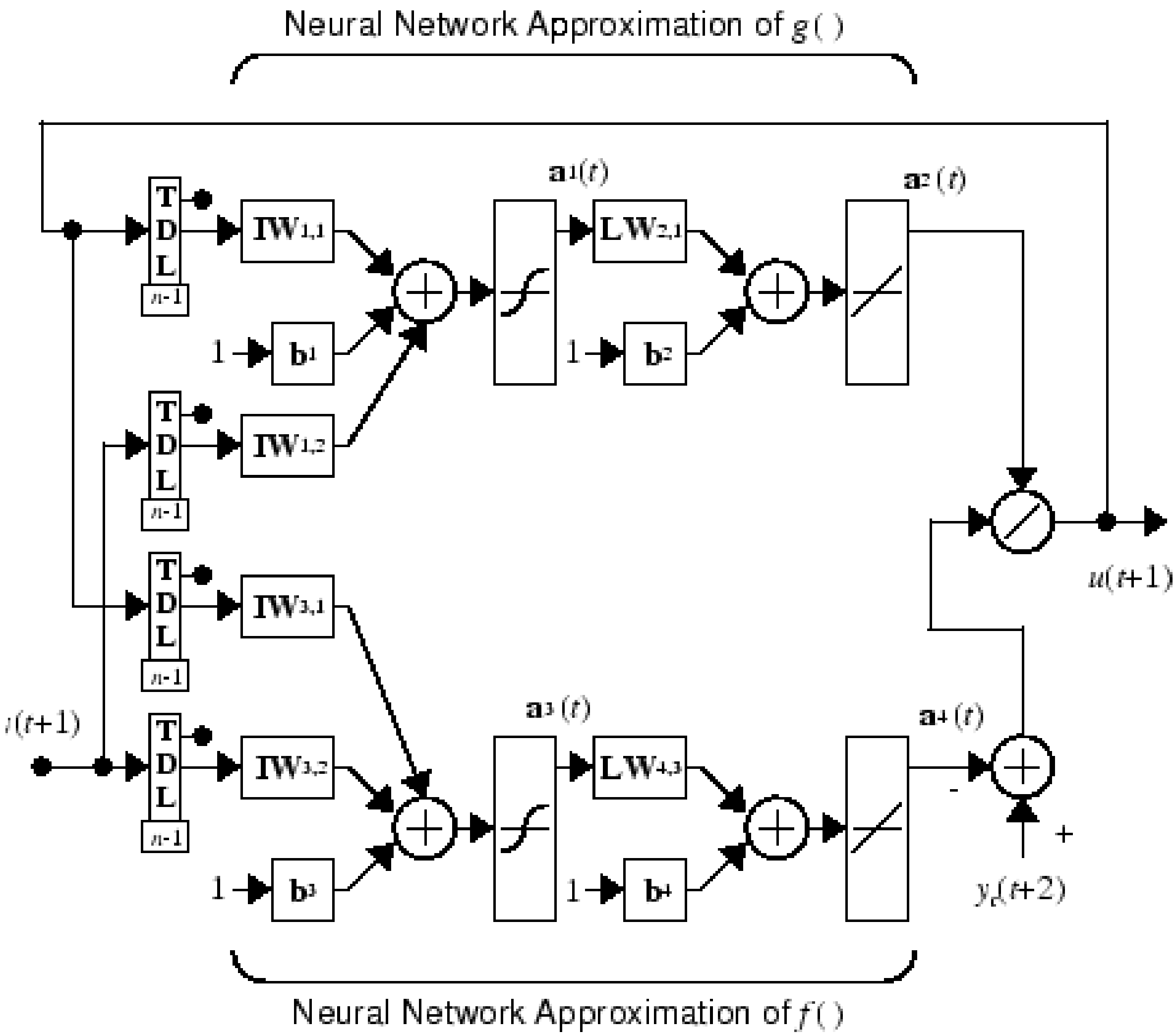


Figure 3.6 NARMA-L2 plant model

Use the NARMA-L2 Controller Block

This section shows how the NARMA-L2 controller is trained. The first step is to copy the NARMA-L2 Controller block from the Deep Learning Toolbox block library to the Simulink® Editor. An example model is provided with the Deep Learning Toolbox software to show the use of the NARMA-L2 controller. In this example, the objective is to control the position of a magnet suspended above an electromagnet, where the magnet is constrained so that it can only move in the vertical direction, as in the following figure.

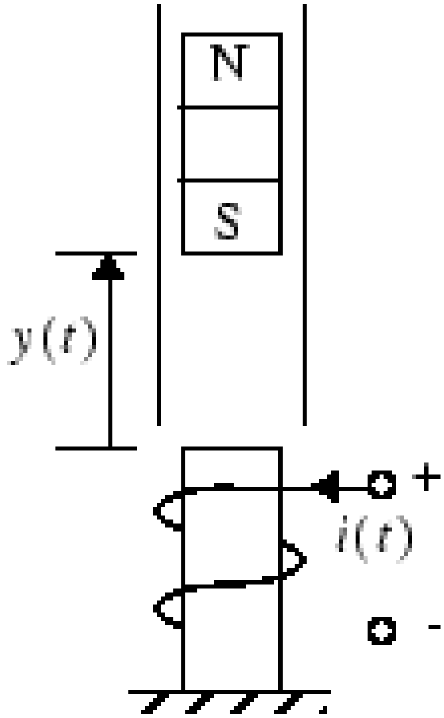


Figure 3.7 A magnet suspended above an electromagnet

The equation of motion for this system is

$$\frac{d^2y(t)}{dt^2} = -g + \frac{\alpha}{M}\frac{i^2(t)}{y(t)} - \frac{\beta}{M}\frac{dy(t)}{dt} \quad (3.7)$$

where y(t) is the distance of the magnet above the electromagnet, i(t) is the current flowing in the electromagnet, M is the mass of the magnet, and g is the gravitational constant. The parameter β is a viscous friction coefficient that is determined by the material in which the magnet moves, and α is a field strength constant that is determined by the number of turns of wire on the electromagnet and the strength of the magnet. To run this example:

1. Start MATLAB®.
2. Type narmamaglev in the MATLAB Command Window. This command opens the Simulink Editor with the following model. The NARMA-L2 Control block is already in the model.

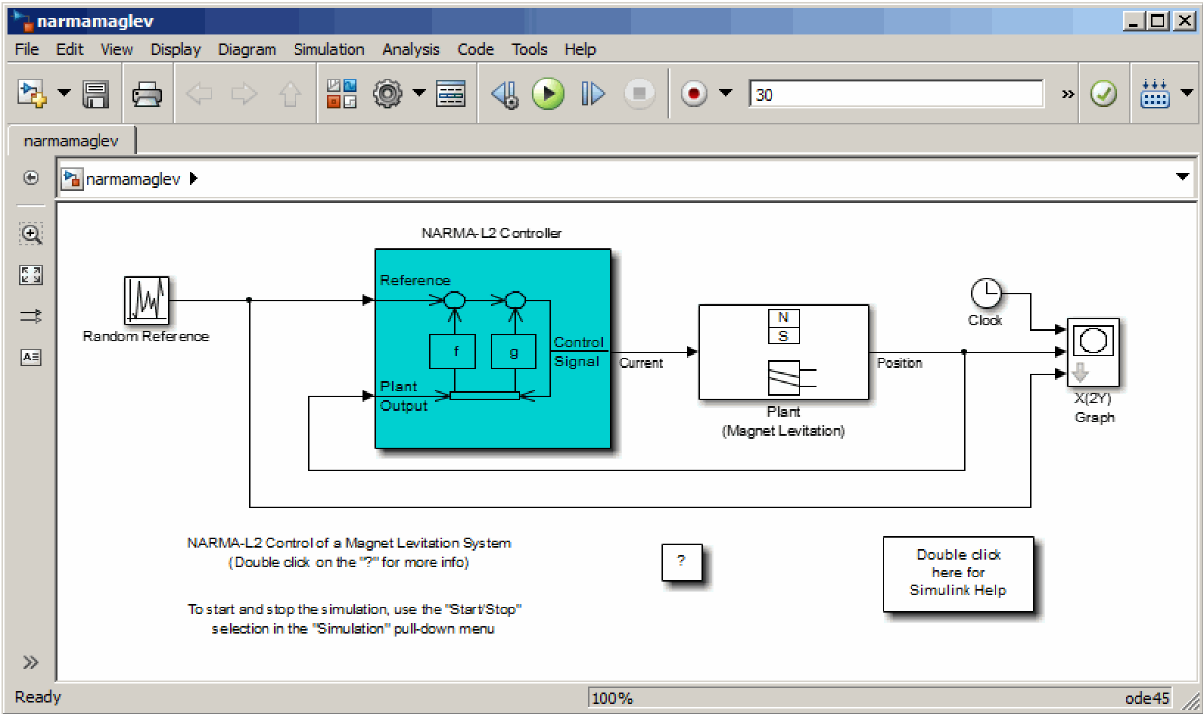


Figure 3.8 narmamaglev sample

3. Double-click the NARMA-L2 Controller block. This opens the following window. This window enables you to train the NARMA-L2 model.

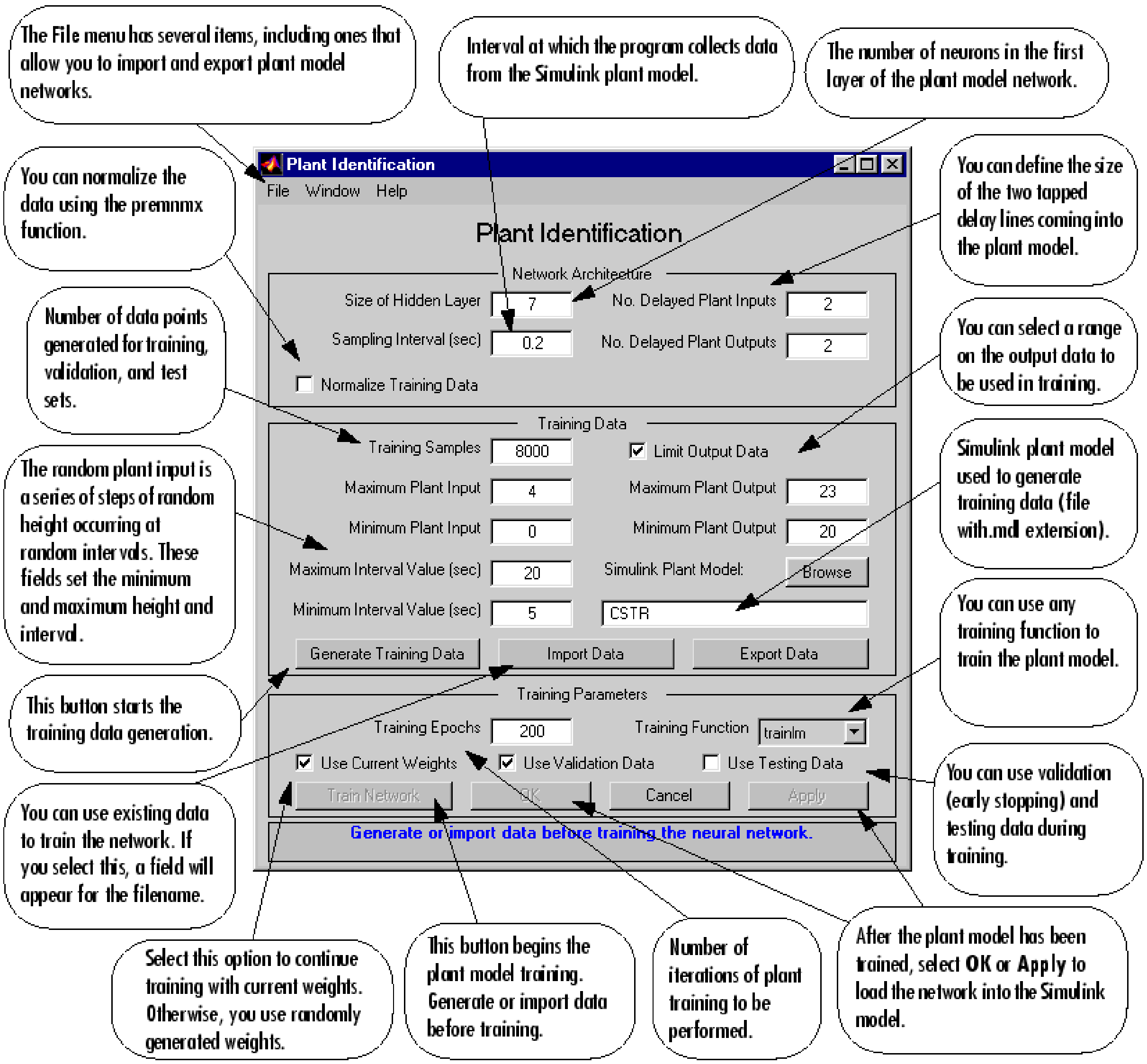


Figure 3.9 Plant identification

4. Click Generate Training Data. The program generates training data by applying a series of random step inputs to the Simulink plant model. The potential training data is then displayed in a figure similar to the following.

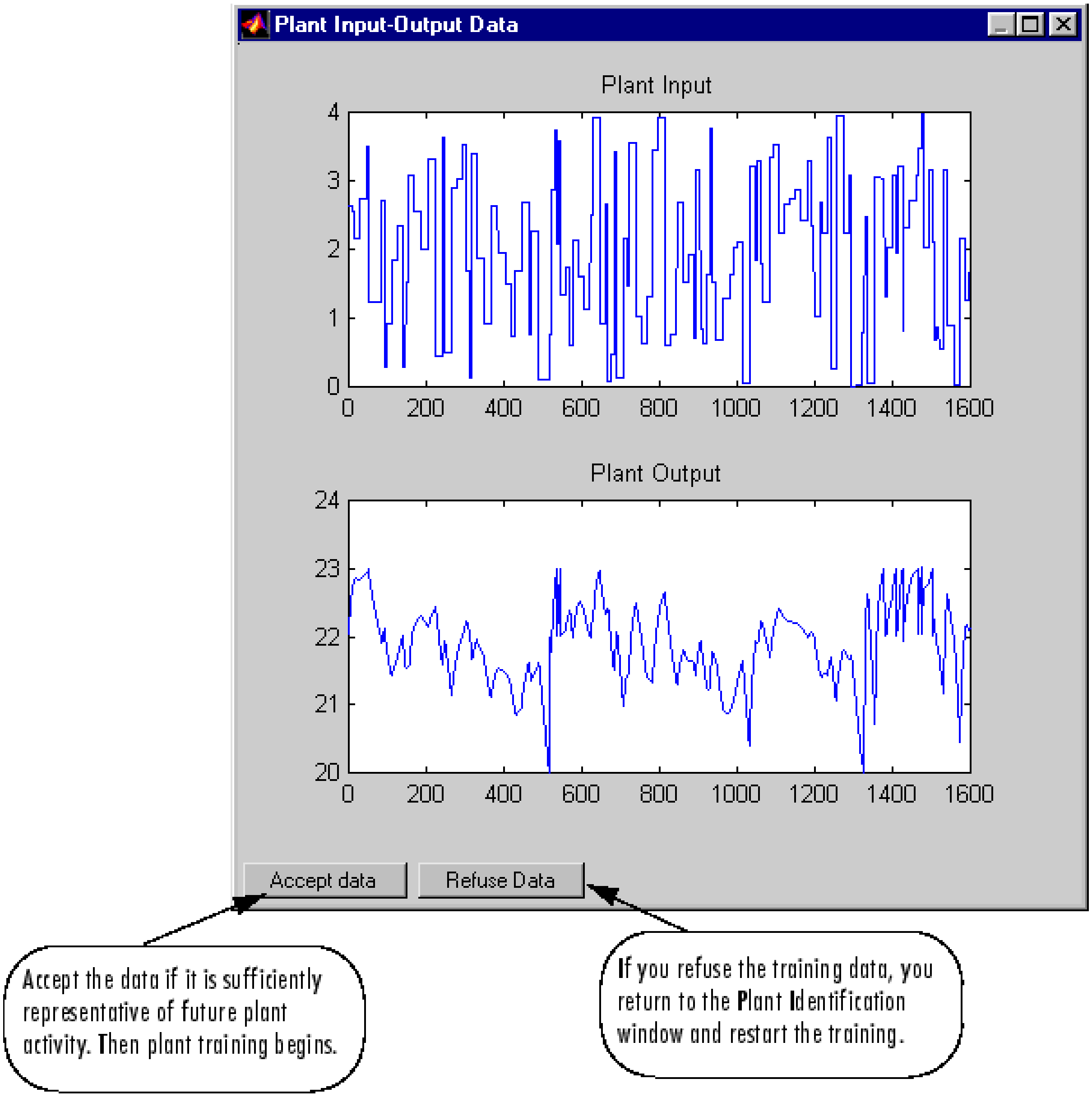


Figure 3.10 Potential training data

5. Click Accept Data, and then click Train Network in the Plant Identification window. Plant model training begins. The training proceeds according to the training algorithm (trainlm in this case) you selected. After the training is complete, the response of the resulting plant model is displayed, as in the following figure. (There are also separate plots for validation and testing data, if they exist.) You can then continue training with the same data set by selecting Train Network again, you can Erase Generated Data and generate a new data set, or you can accept the current plant model and begin simulating the closed loop system. For this example, begin the simulation, as shown in the following steps.

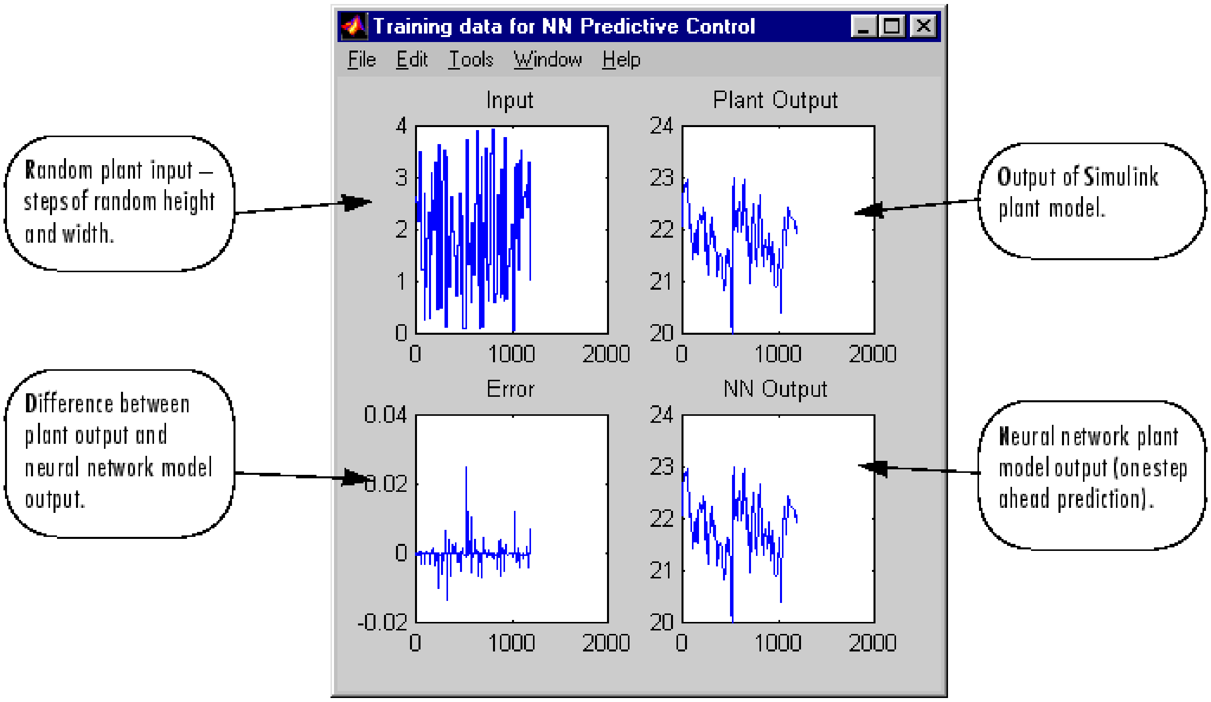


Figure 3.11 Resulting plant model

6. Return to the Simulink Editor and start the simulation by choosing the menu option Simulation > Run. As the simulation runs, the plant output and the reference signal are displayed, as in the following figure.

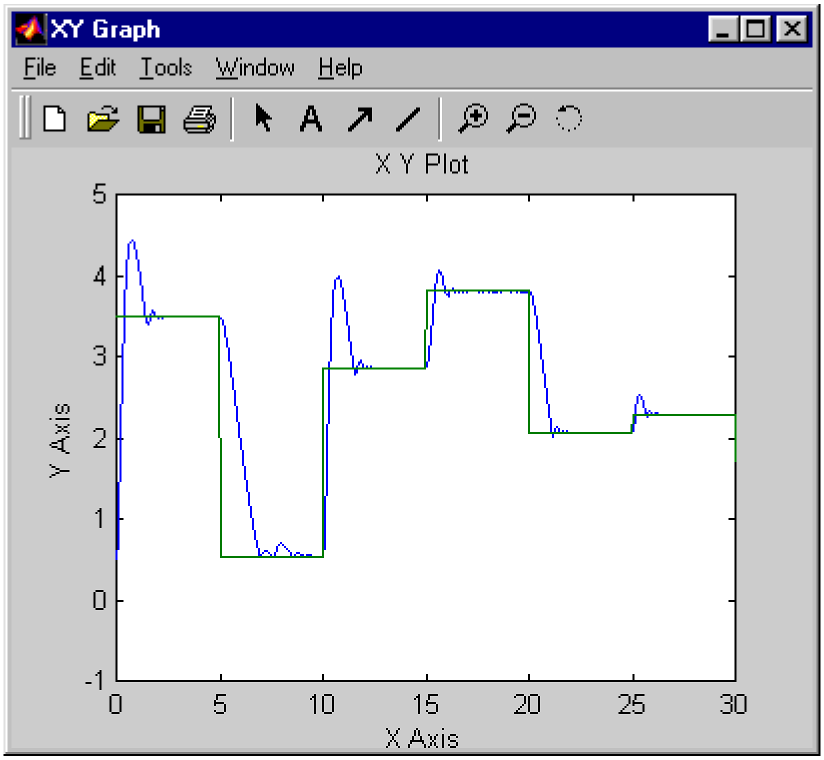


Figure 3.12 Simulation result

References

[1] Priti Srinivas Sajja - Illustrated Computational Intelligence: Examples and Applications (2021, Springer)

[2] Neural Network Control Systems, [Online]. Available: https://www.mathworks.com/help/deeplearning/neural-network-control-systems.html?s_tid=CRUX_lftnav. [Accessed 2021].

Exercise

- Model a centrifugal pump system
  - To deliver water of 200m length and 30m height
  - Desired response is the flowrate
  - Input to the process G(s) is the input voltage of a DC motor that drives the pump
- Applied an ANN-based controller to the system

Lecture 04: Application of industrial control

2 sessions, 4 hrs

Robotics

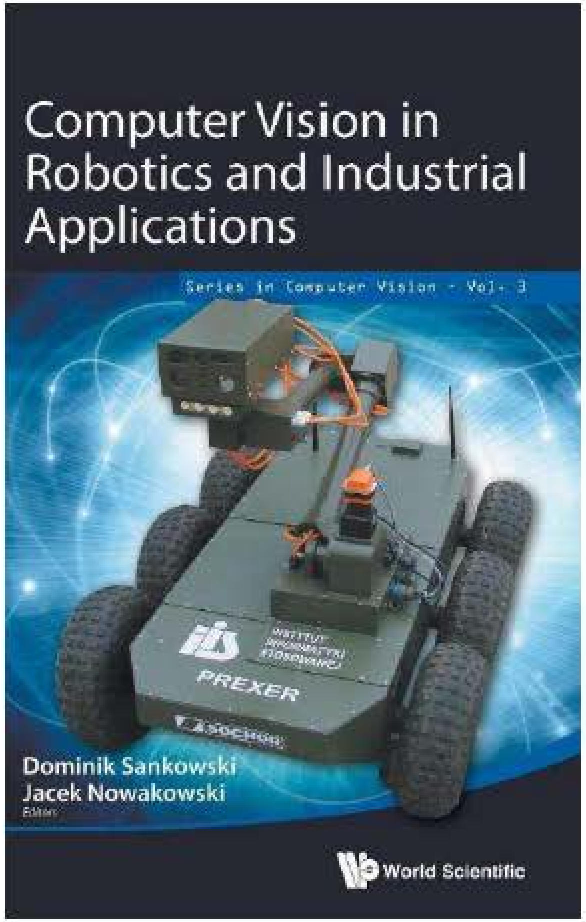
End 1970s

2020s: Many dimensions

Robotics

Training an Industrial Robot Using AI

Robotics & Visions



Vision-guided robotics

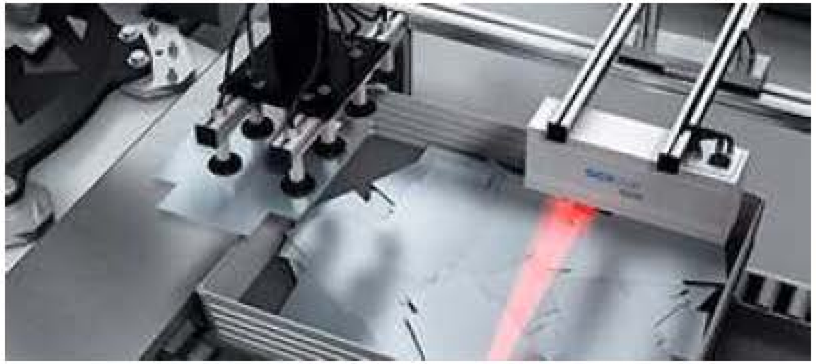
2D-Vision
3D-Vision
Radars
Lasers

Industrial Robots in Extreme Conditions

Robotics

Autonomous trans-pallet

Robotic preparation of parts orders on the assembly line



Automated part picking (bulk items)

Simplified guidance of a Universal Robots robot

Locating parts in boxes

Picking up raw components for assembly

On-line quality control

3D sampling on tape

Robotics

COBOT

Industrial
Collaborative
Robot

Robotics

Robots have long been used in industry, but they are evolving to be more autonomous, interact with each other, and work more safely with humans

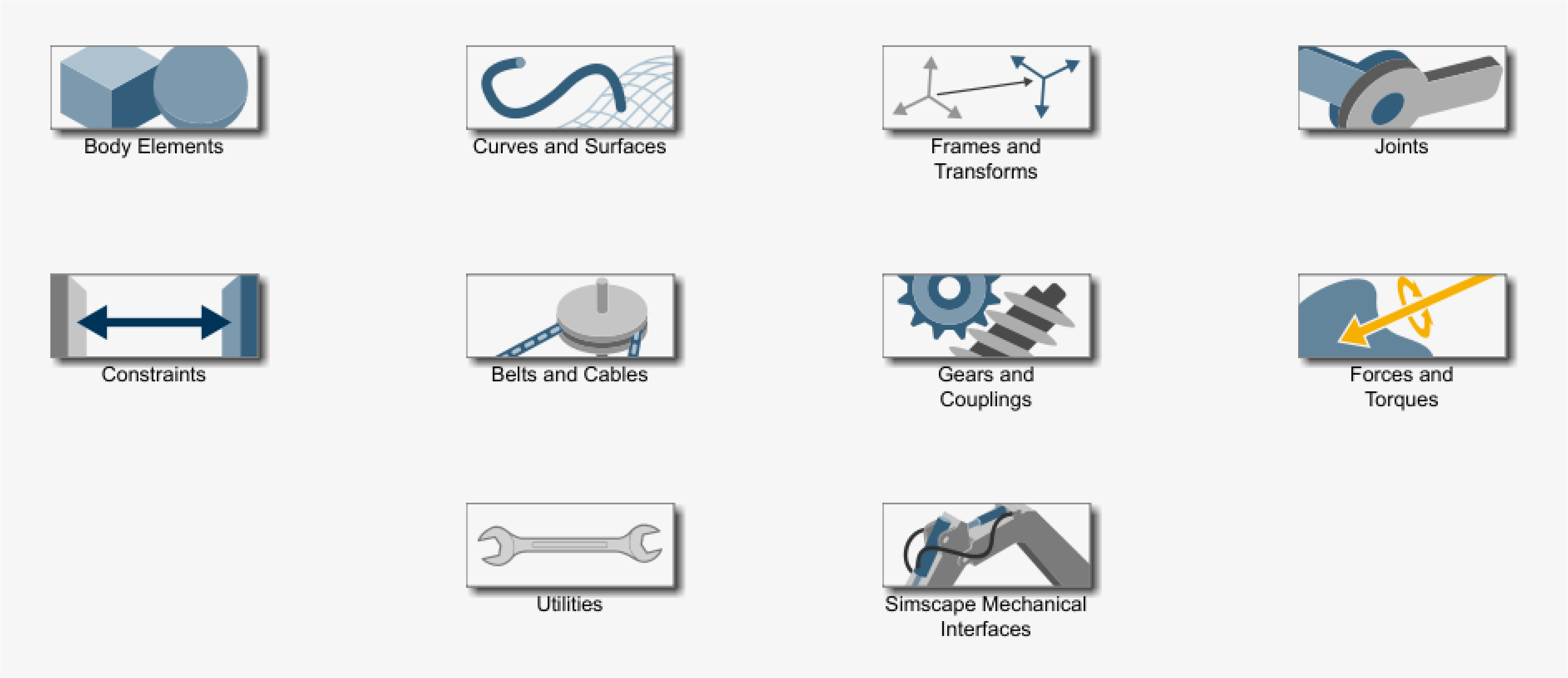
Cobots, or collaborative robots, are much less scary and take care not to hurt people. Cobots are equipped with sensors and software so that they do not need to be separated from human workers. In the factory of the future, cobots will assist the human operator

When it comes to safety, an ISO(10218) 34 standard specifies and describes requirements and recommendations for safety around robots in the industrial setting only. But since 2016, a new ISO 15066 35 standard addresses human-robot interaction in industry, and gives specifications on the safety of cobots to be implemented.

Global Industrial Robot Market 2020 by Manufacturers, Type and Application, Forecast to 2025

https://www.marketsandresearch.biz/report/98579/global-industrial-robot-market-2020-by-manufacturers-type-and-application-forecast-to-2025

Simscape Multibody



Modeling Constant Velocity Joints - Power Take-Off Shaft

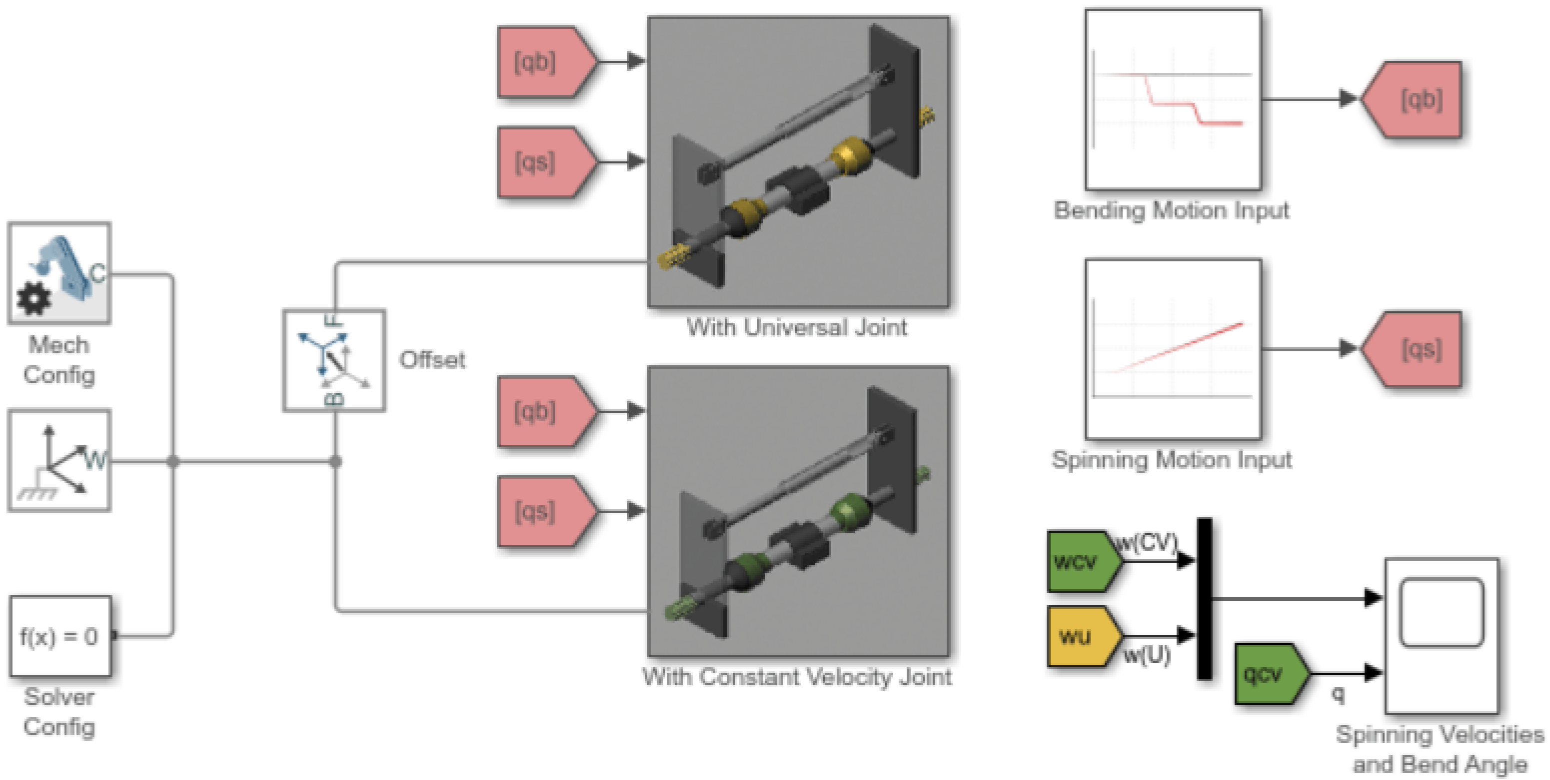
R2021b

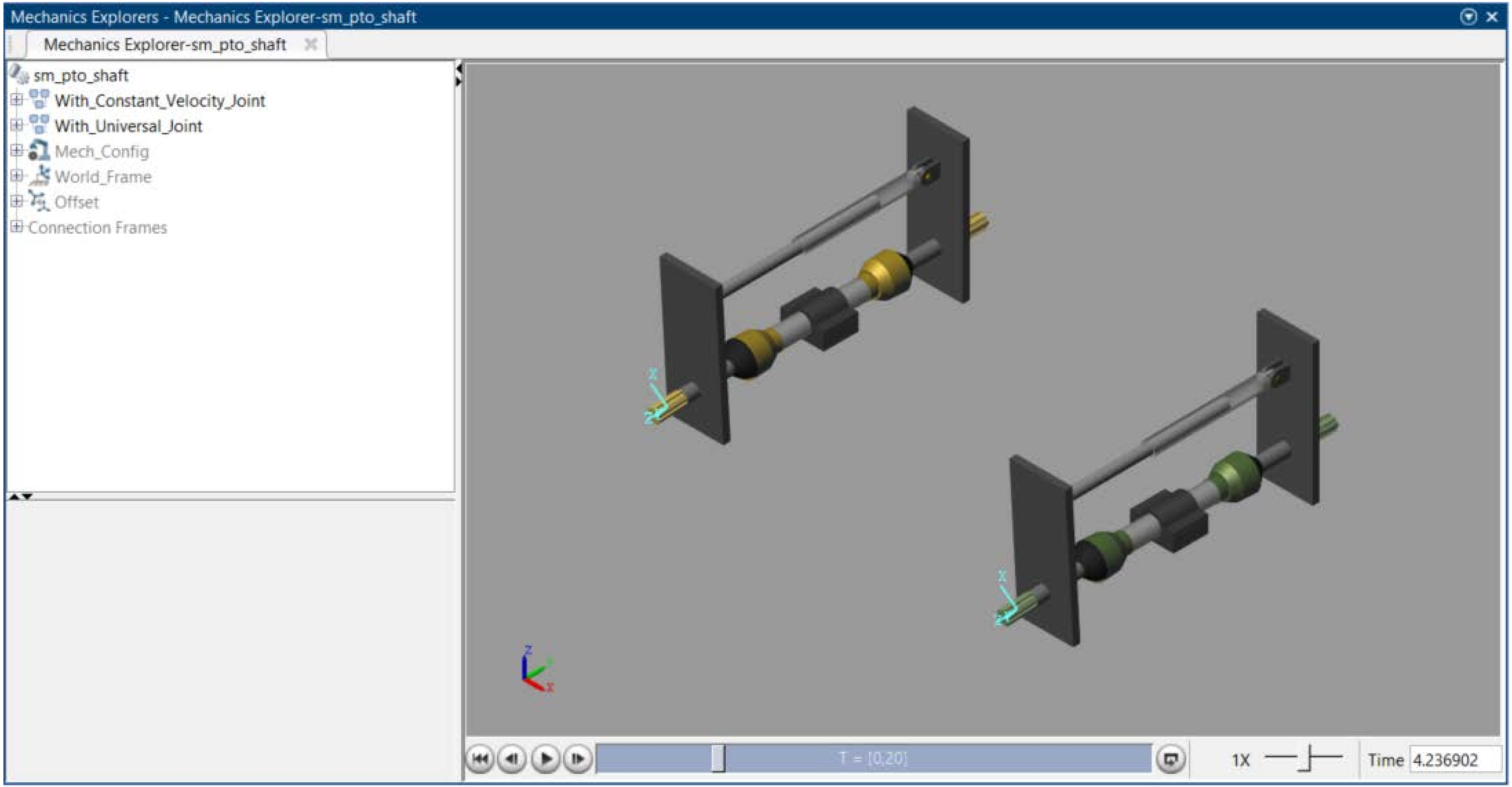
Open Model

This example shows a Power Take-Off (PTO) shaft, a device for transferring power from tractor engines to auxiliary equipment such as soil tillers and wood chippers. The model includes two PTO subsystems identical in every sense but their joints. One contains universal (U) joints, the other constant-velocity (CV) joints.

At large bend angles, U joints lead to uneven rotations, subjecting the adjoining shafts to relatively large vibrations and elevated internal stresses. CV joints eliminate these by allowing the shafts to spin at constant velocity, resulting in a smooth motion profile no matter the bend angle.

A single set of motion inputs governs the behavior of the PTO subsystems. The Scope block plots the resulting angular velocities and bend angles of the shafts, enabling you to compare the kinematics of the two joint types.




Mechanics Explorers - Mechanics Explorer-sm_pto_shaft
Mechanics Explorer-sm_pto_shaft
sm_pto_shaft
With_Constant_Velocity_Joint
With_Universal_Joint
Mech_Config
World_Frame
Offset
Connection Frames
T = [0,20]
1X
Time 4.236902

Exercise

- Model a 2 joints mechanical arms based on Simscape Multibody
  - Specify its functions
  - Model its mechanics
  - Define the control signal
  - Define the feedback signal

Lecture 05: Plant simulation and emulation

2 sessions, 4 hrs

Software and communications

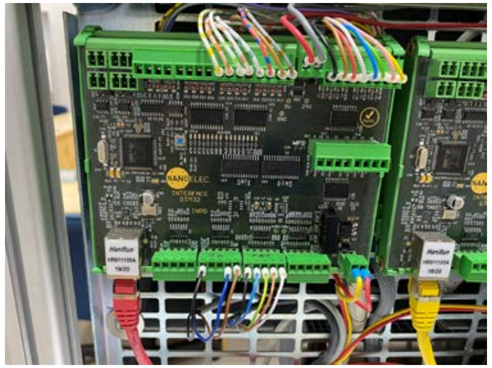
- MathWorks Simulink
  - Request, read, and write to GICS
- Simulink Desktop Real-Time
  - Synchronize Simulink to real-time processing and communication
- Simscape
  - Simulate physical systems (optional)
- Communication
  - UDP
- PLC ←→ GICS ← → Simulink
  - GICS is the middleman to link PLC and Simulink
  - Simulink emulates practical process
  - PLC controls the process

Emulation concept

PLC

Direct connected
I/O
Analog voltage

GICS card



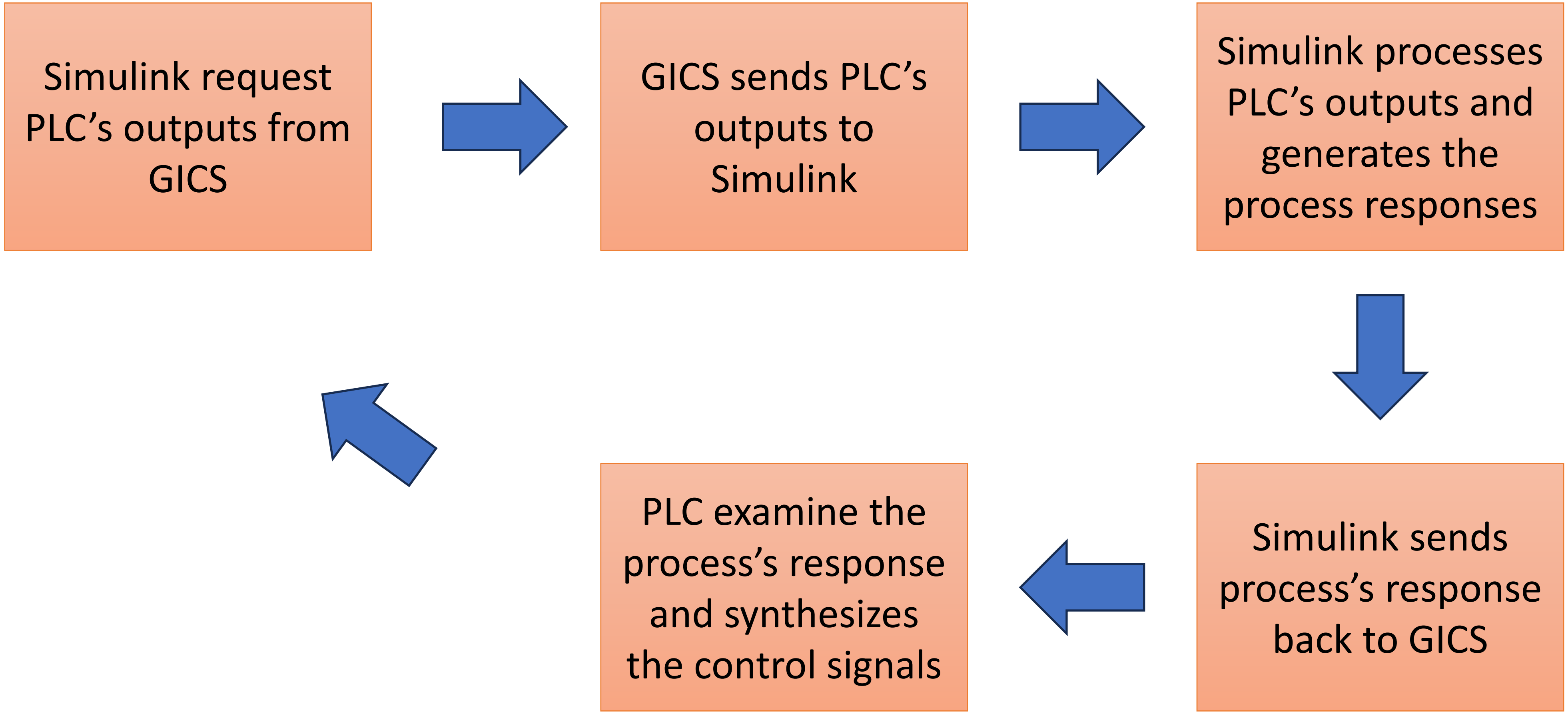
UDP packet
binary

MathWorks Simulink

Controller

Virtual
Process or
Plant

Communication loop



Scenario

PLC's 2-channel output to GICS

- Read by Simulink
- From PLC: -27000 to 27000
  - Sawtooth wave
  - 16-bit signed integer
- GICS's record: 0 – 35xx
  - 12-bit unsigned integer
- Simulink's read: 0 – 35xx
  - 16-bit unsigned integer

Simulink's 4-channel output to GICS

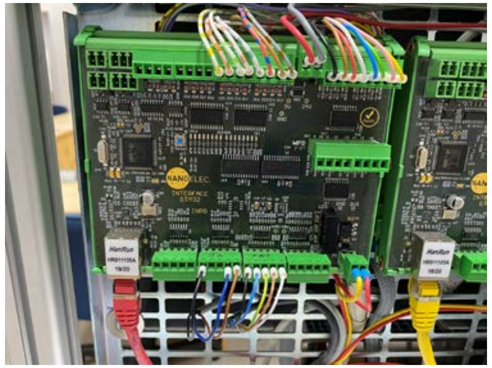
- Read by PLC
- From Simulink: 50 – 950
  - Sinewave
  - 16-bit unsigned integer
- GICS's record: 50 - 950
  - 10-bit unsigned integer
- PLC's read: -25000 $\pm \Delta$ to 25000 $\pm \Delta$
  - 16-bit signed integer

Configuration

PLC

Direct connected
I/O
Analog voltage

GICS card



UDP packet
binary

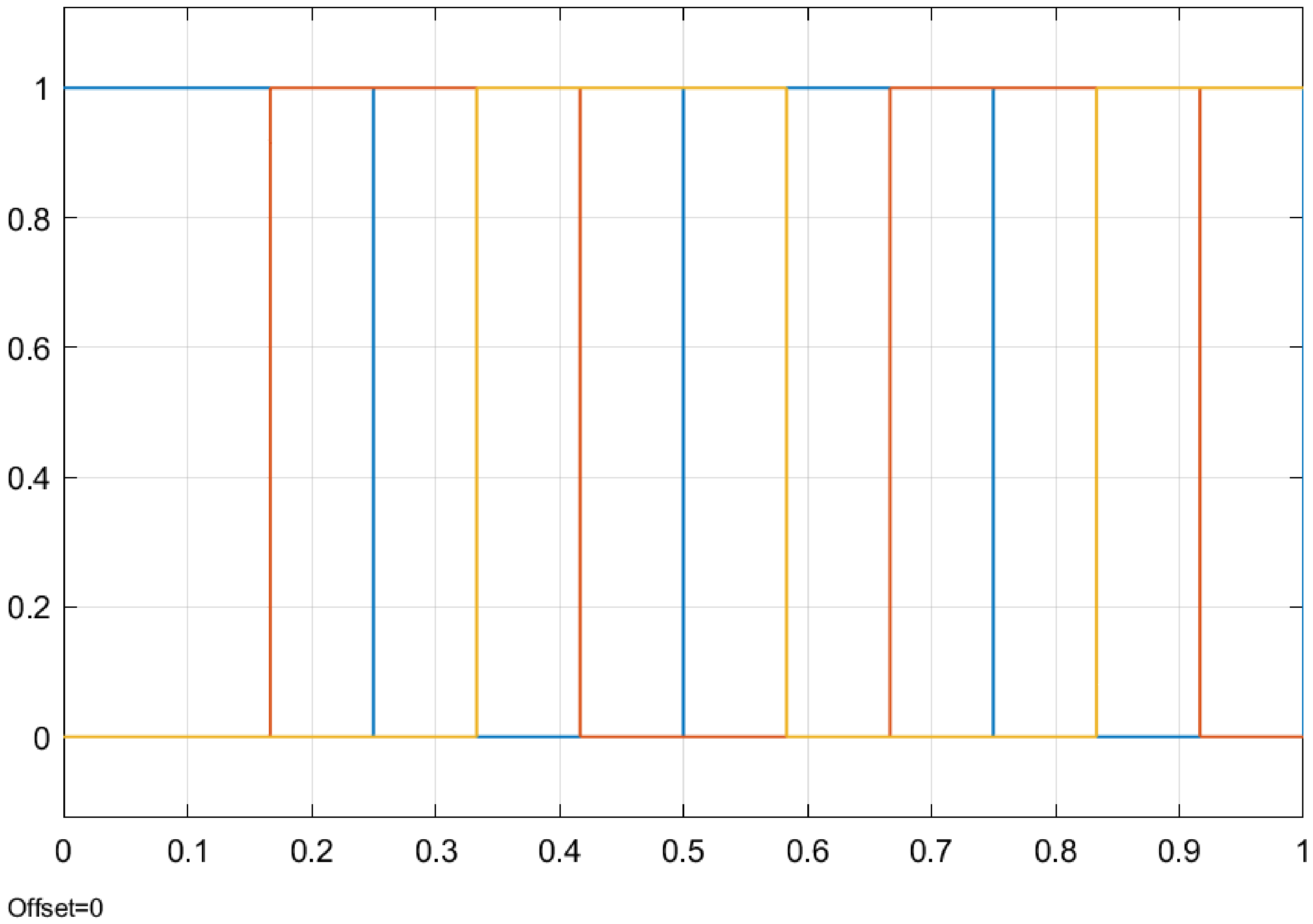
MathWorks Simulink

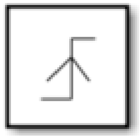
-Read process output from %IW0
-Send control signal via %QW2

-Read control signal from PLC directly to AI3
-Read process output from Simulink via UDP packet and keep in AO1

-Read control signal from GICS's AI3
-Send UDP packet to update GICS's AO1

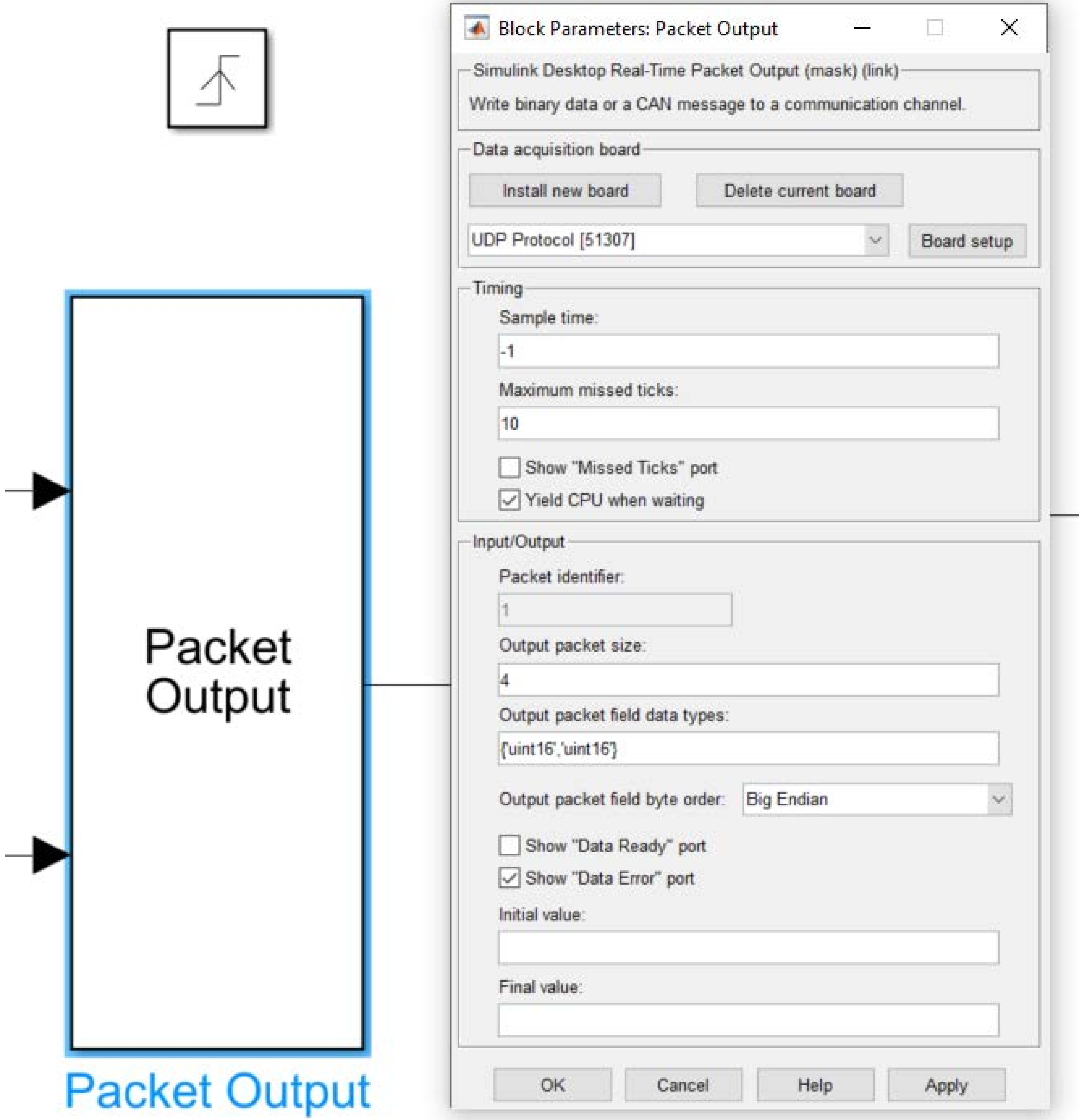
GICS communication cycles



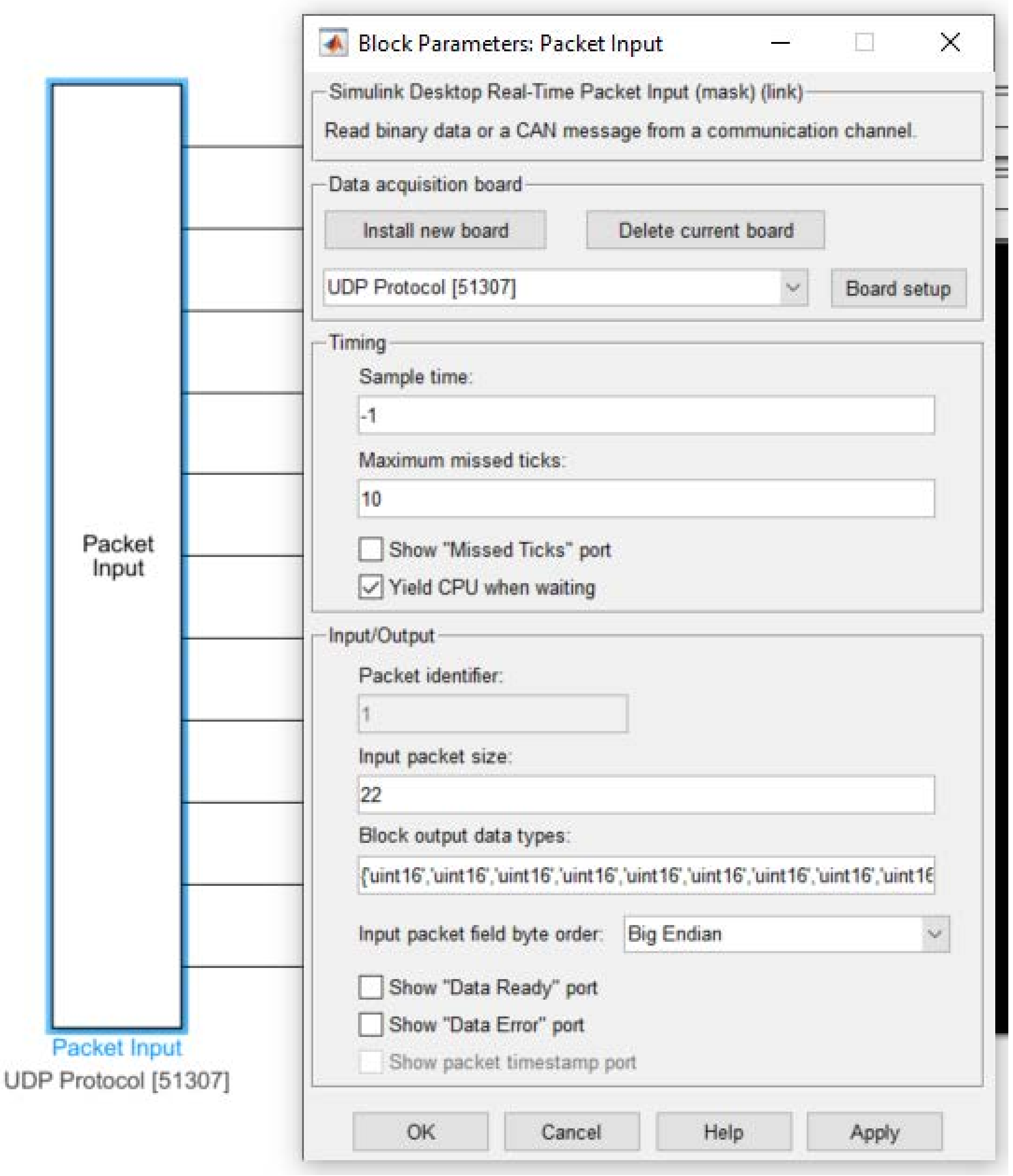


- -Triggered at the rising edge
- -Loop of sequences are:
  - - Send request
  - - Read from GICS
  - - Write response to GICS

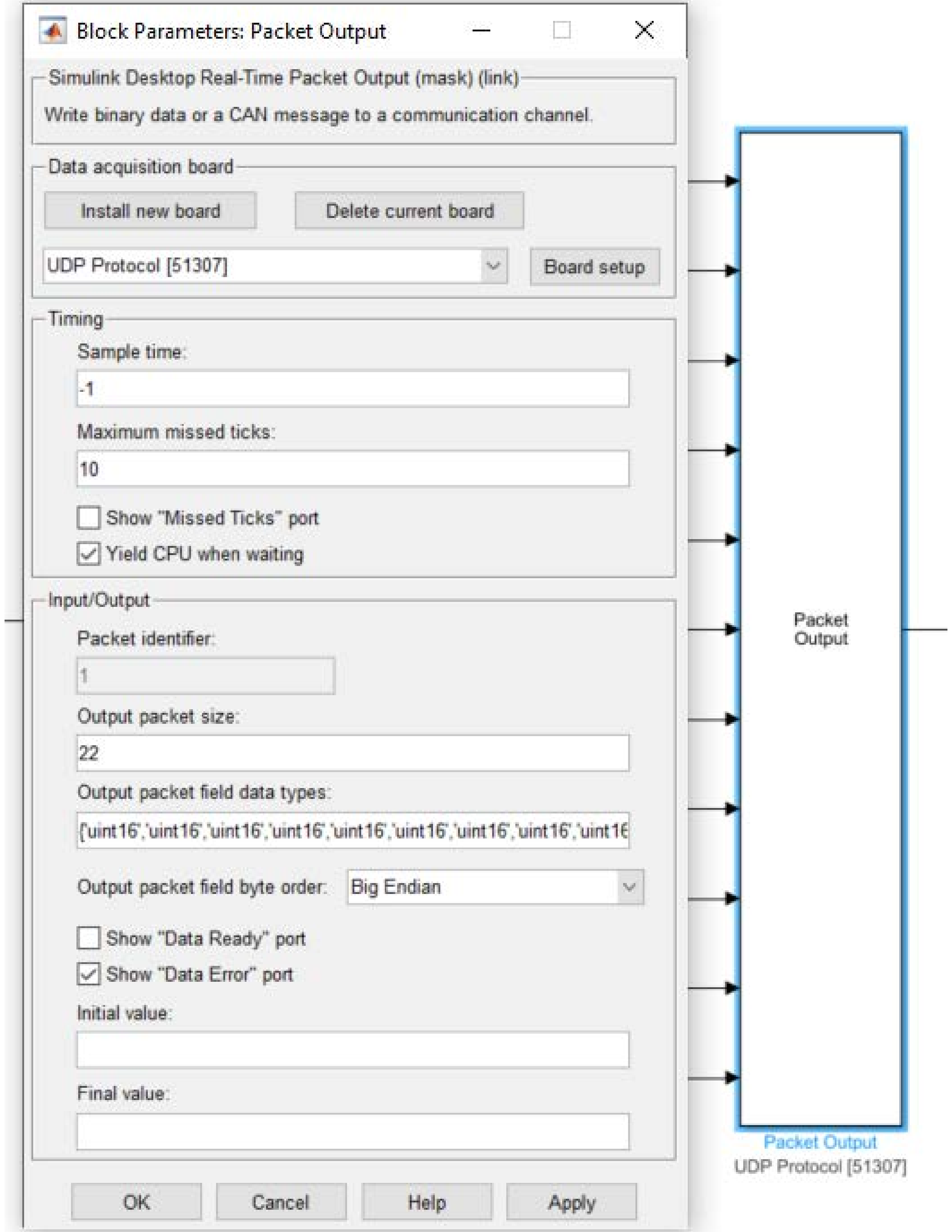
Request setting



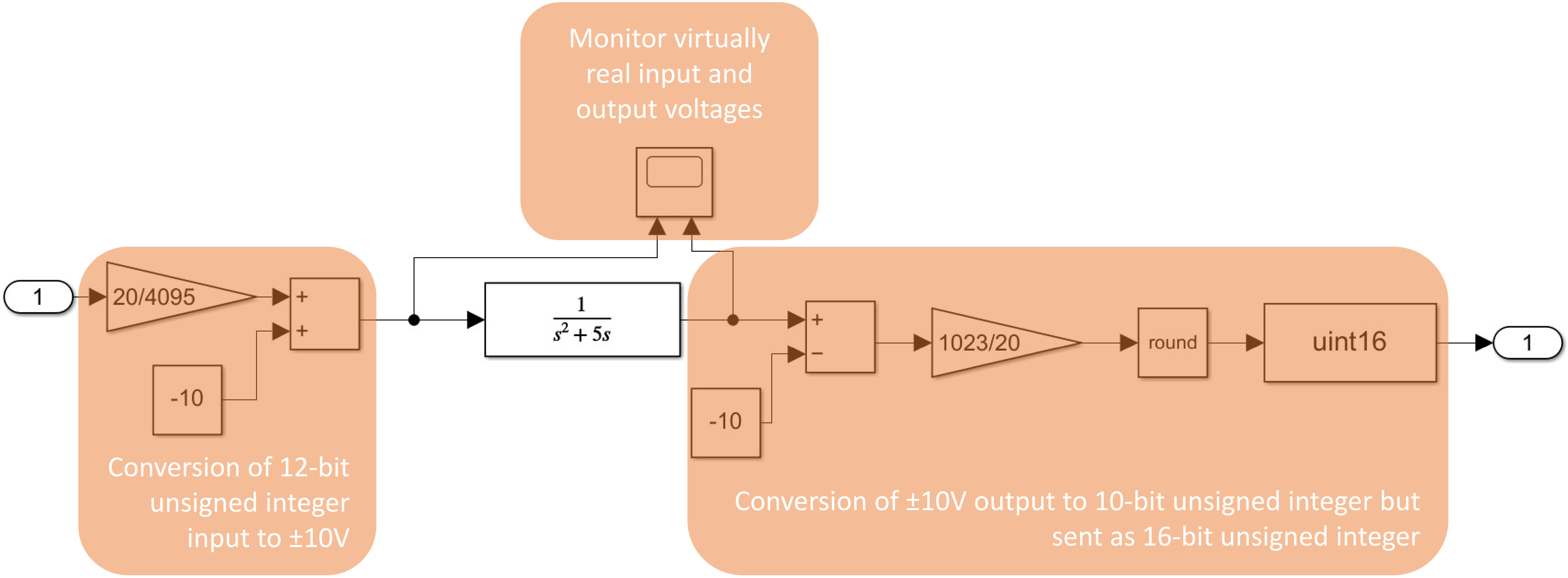
Read setting



Write setting



Virtual Process



Exercise

- Model an industrial bottle filling process
  - Model the filler and other mechanics in Simulink
  - All control signals (digital and analog) are from PLC
  - Use GICS as the interface between PLC and the virtual process in Simulink
  - Use HMI as a SCADA

1501431 Intelligent Control Systems

Program: Bachelor program in Computer Engineering

Credit: 3(2-2) Lecture: 30 Hours Lab: 30 Hours

1st Semester, Academic Year: 2024

Assoc. Prof. Punnarumol Temdee, Ph.D.

Asst. Prof. Roungsan Chaisricharoen, Ph.D.

Asst. Prof. Santichai Wicha, Ph.D.

Lect. Chayapol Kamyod, Ph.D.

FACTORI 4.0 Erasmus +

Co-funded by the Erasmus+ Programme of the European Union

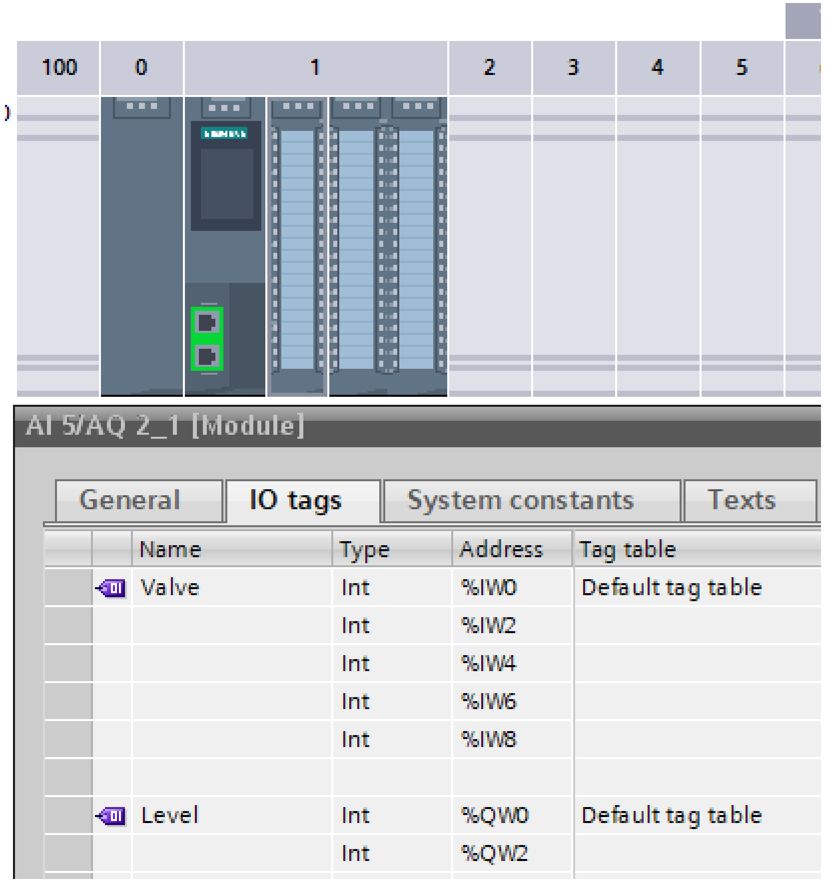
This course has been modified in the framework of an Erasmus + project: Asean Factori 4.0 Across South East Asian Nations: From Automation and Control Training to the Overall Roll-out of Industry 4.0

609854-EPP-1-2019-1-FR-EPPKA2-CBHE-JP

Lab 01: Analog I/O of PLC

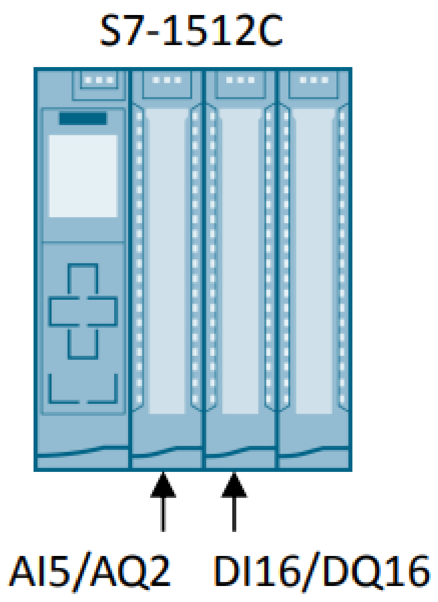
Setting up the project

- Setting
  - Create new project named “Lab01”
  - Add PLC
  - Config the network
- Config analog ports
  - Analog I/O module is in the first slot as shown in the figure on the right
  - Inputs are the addresses labeled as “%IW_ ”
  - Outputs are the address labeled as “%QW _”
  - Use %IW0 and %QW0



Analog Module

Figure 1-1



Inputs

- 5 channel
- Datatype: integer
- Cycle time: 1ms
- -10 V to +10 V
- Resolution: 16-bit including sign

Outputs

- 2 channel
- Datatype: integer
- Cycle time: 1ms
- -10 V to +10 V
- Resolution: 16-bit including sign

C.3 Representation of input ranges

The tables below set out the digitized representation of the input ranges separately for bipolar and unipolar input ranges. The resolution is 16 bits.

Table C- 4 Bipolar input ranges

Dec. value	Measured value in %	Data word																Range
		2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
32767	>117.589	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Overflow
32511	117.589	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	Overrange
27649	100.004	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	
27648	100.000	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	Nominal range
1	0.003617	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
0	0.000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
-1	-0.003617	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
-27648	-100.000	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	
-27649	-100.004	1	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	Underrange
-32512	-117.593	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
-32768	<-117.593	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Underflow

C.3.1 Representation of analog values in voltage measuring ranges

The following tables list the decimal and hexadecimal values (codes) of the possible voltage measuring ranges.

Table C- 6 Voltage measuring ranges ±10 V, ±5 V

Values		Voltage measuring range		Range
dec.	hex.	±10 V	±5 V	
32767	7FFF	>11.759 V	>5.879 V	Overflow
32511	7EFF	11.759 V	5.879 V	Overrange
27649	6C01			
27648	6C00	10 V	5 V	Nominal range
20736	5100	7.5 V	3.75 V	
1	1	361.7 µV	180.8 µV	
0	0	0 V	0 V	
-1	FFFF			
-20736	AF00	-7.5 V	-3.75 V	
-27648	9400	-10 V	-5 V	
-27649	93FF			Underrange
-32512	8100	-11.759 V	-5.879 V	
-32768	8000	<-11.759 V	<-5.879 V	Underflow

Mapping input

- A tachometer can sense rotational speed from 0 to 4000 rpm
  - Output: -10V (0 rpm) – 10V (4000 rpm)
- Compile PLC’s input into rpm
  - -10 V input → -27648 = 0 rpm
  - 10 V input → 27648 = 4000 rpm
  - rpm = (input + 27648) x 4000/55296
- Lab
  - Write a ladder diagram to map this input to rpm

C.4 Representation of output ranges

The tables below set out the digitalized representation of the output ranges separately for bipolar and unipolar ranges. The resolution is 16 bits.

Table C- 16 Bipolar output ranges

Dec. value	Output value in %	Data word																Range
		2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
32511	117.589	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	Maximum output value*
32511	117.589	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	Overrange
27649	100.004	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	
27648	100.000	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	Nominal range
1	0.003617	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
0	0.000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
-1	-0.003617	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
-27648	-100.000	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	
-27649	-100.004	1	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	Underrange
-32512	-117.593	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
-32512	-117.593	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	Minimum output value**

* When values > 32511 are specified, the output value is limited to 117.589%.

** When values < -32512 are specified, the output value is limited to -117.593%.

C.4.1 Representation of analog values in the voltage output ranges

The tables below list the decimal and hexadecimal values (codes) of the possible voltage output ranges.

Table C- 18 Voltage output range ±10 V

Values			Voltage output range	Range
	dec.	hex.	±10 V	
>117.589%	>32511	>7EFF	11.76 V	Maximum output value
117.589%	32511	7EFF	11.76 V	Overrange
	27649	6C01		
100%	27648	6C00	10 V	Nominal range
75%	20736	5100	7.5 V	
0.003617%	1	1	361.7 µV	
0%	0	0	0 V	
	-1	FFFF	-361.7 µV	
-75%	-20736	AF00	-7.5 V	
-100%	-27648	9400	-10 V	
	-27649	93FF		Underrange
-117.593%	-32512	8100	-11.76 V	
<-117.593%	<-32512	< 8100	-11.76 V	Minimum output value

Mapping output

- A linear mechanical actuator of distance 0 to 10cm
  - Input: -10V (0 cm) – 10V (10 cm)
- Compile cm into PLC’s output
  - -10 V output → -27648 = 0 cm
  - 10 V output → 27648 = 10 cm
  - output = -27648 + cm x (55296/10)
- Lab
  - write a ladder diagram to generate this output

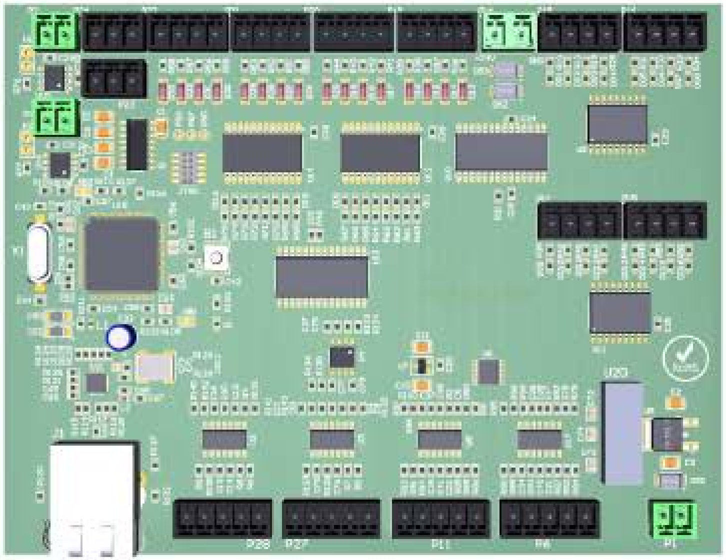
Exercise

- Design a mapping of
  - Input ranged from -7V to 7V that linearly represent the temperature of -50 to 50 degree Celsius
  - Output to a DC drive of a DC motor
    - 0V → 0 rpm
    - 5V → 5000 rpm
- Write the ladder diagrams of both cases

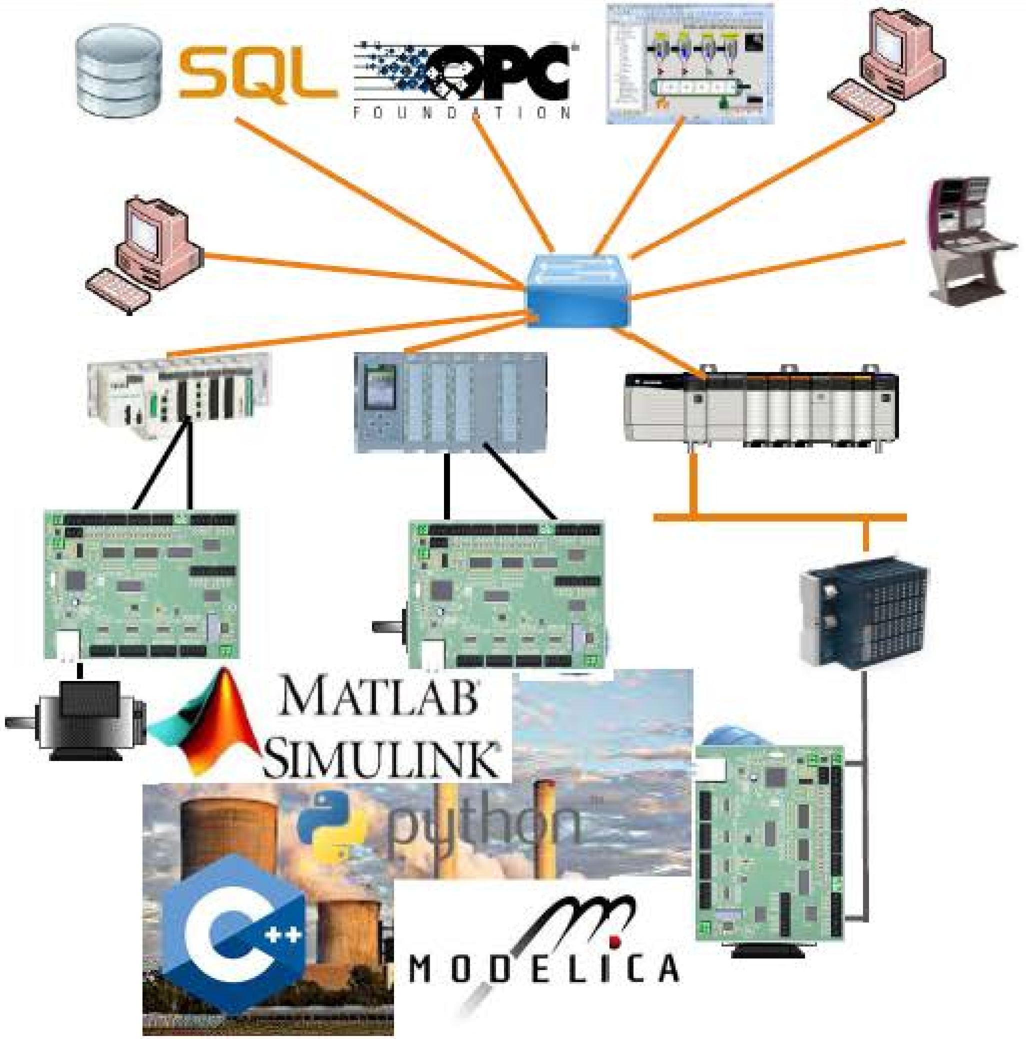
Lab 02: Analog I/O of emulation card

HARDWARE IN THE LOOP SYSTEM

- **Home made electronic interface card**



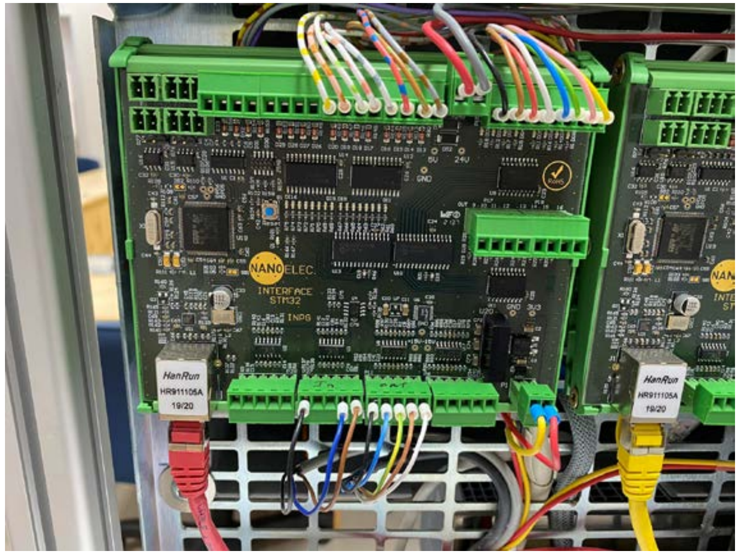
- ▶ **24 sensors and 24 actuators**
  - 16 digital inputs / 16 digital outputs
  - 8 analog inputs / 8 analog outputs
- ▶ **Less than 500€**
- ▶ **Reasonable timing performance (< 10 ms response time)**
- ▶ **Easily chain (Ethernet addressing)**



Configuration

Output → Input

Input ← Output

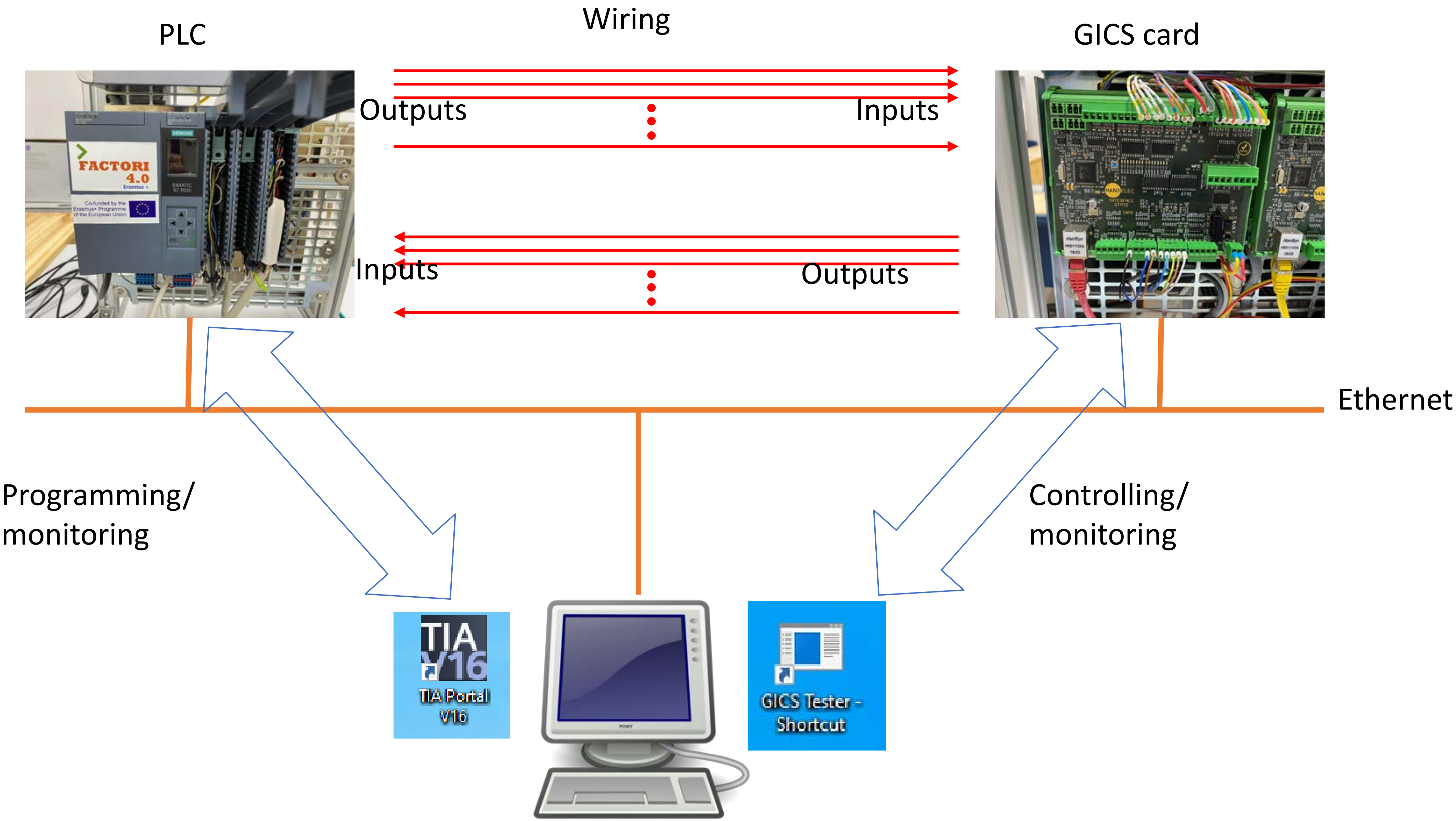


PLC

GICS card

PLC I/O are programmed in TIA Portal

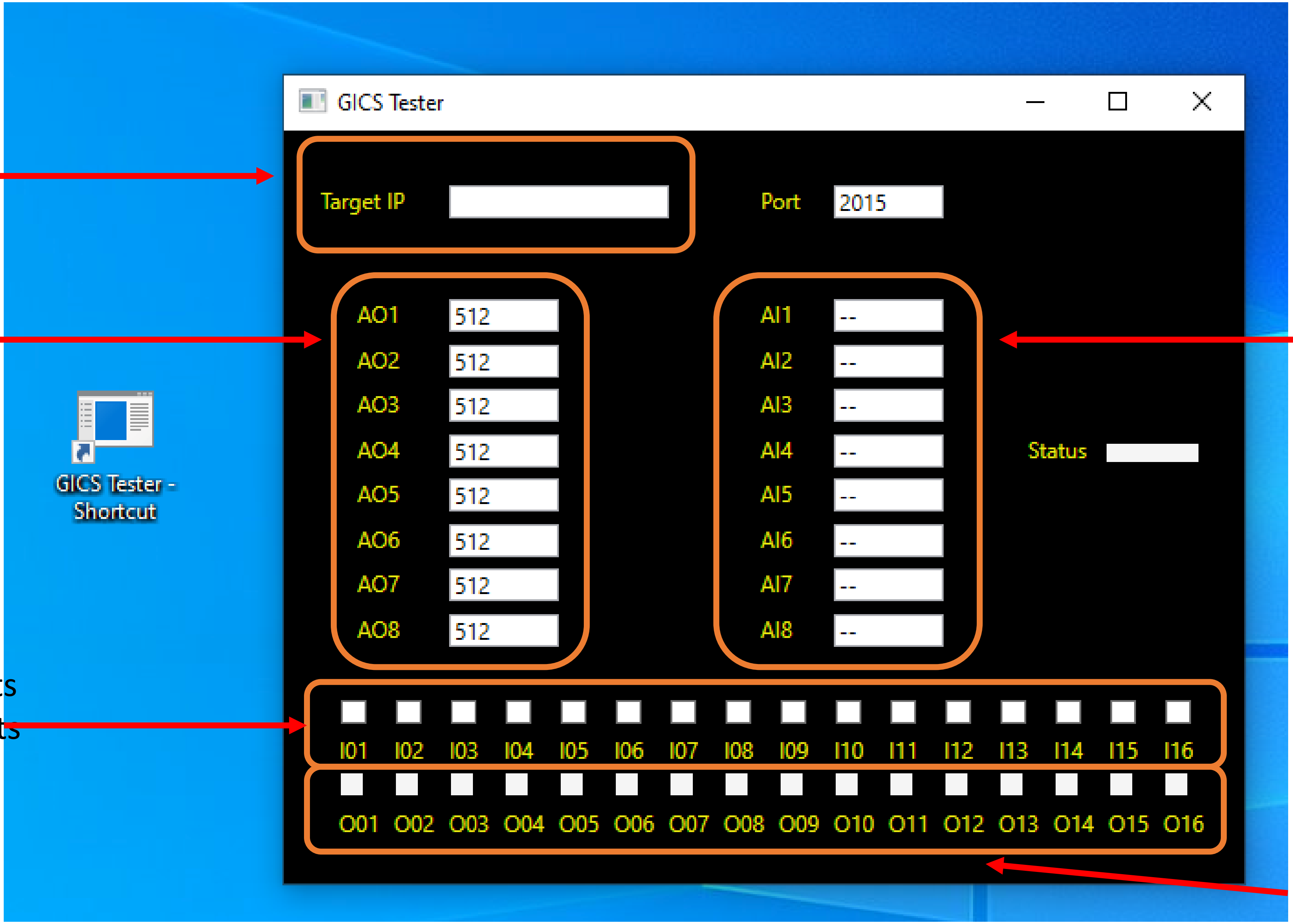
GICS's I/O are manipulated in a special program called "GICS Tester".


PLC
Wiring
GICS card
Outputs
Inputs
Inputs
Outputs
Ethernet
Programming/
monitoring
Controlling/
monitoring
TIA V16
TIA Portal V16
GICS Tester - Shortcut

Card’s IP address

Card’s analog outputs or PLC’s analog inputs

Card’s digital outputs or PLC’s digital inputs



Card’s analog inputs or PLC’s analog outputs

Card’s digital inputs or PLC’s digital outputs

Detail of Analog I/O

Outputs

- 8 channels
- Voltage range: -10V to 10V
- Resolution: 10-bit
- 0V at 512
- -10V at 0
- 10V at 1023

Inputs

- 8 channels
- Voltage range: -10V to 10V
- Resolution: 12-bit
- 0V at 2048
- -10V at 0
- 10V at 4095

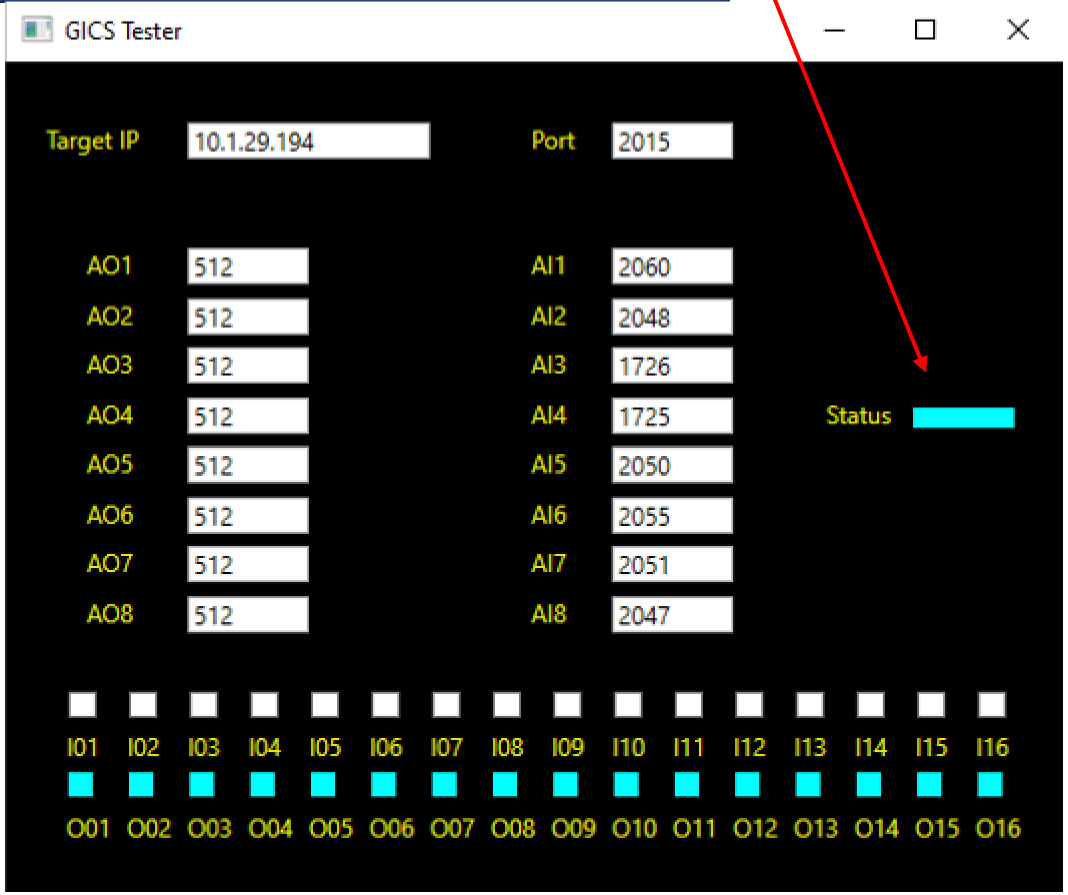
I/O equations

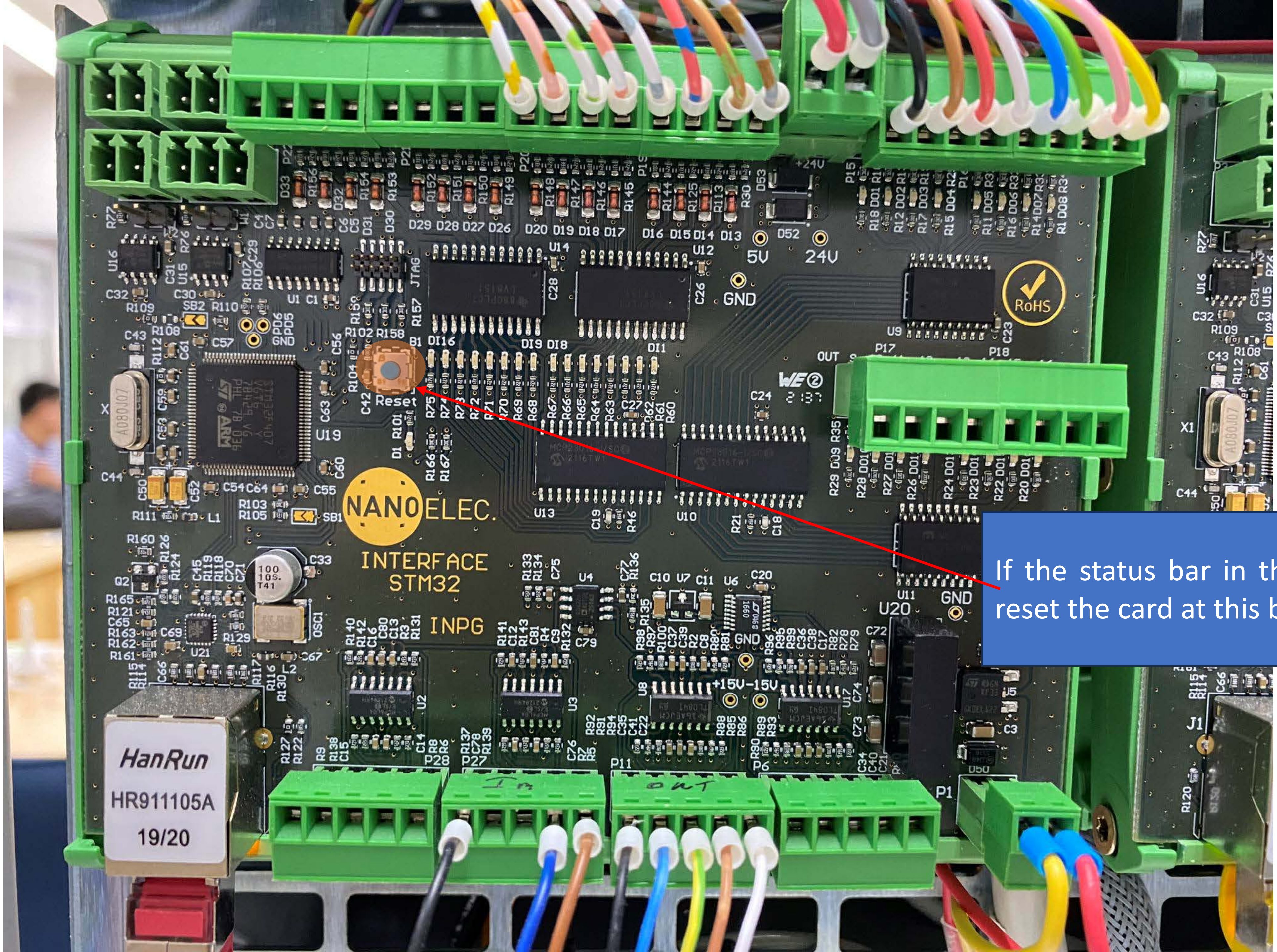
- Vo → required output voltage
- Od → 10-bit decimal of output voltage
- Od = $\lfloor 0 + $ (Vo + 10)(1024/20)$\rceil$
  - Round to the nearest integer

- Vi → sensed input voltage
- Id → read 12-bit input decimal
  - Vi = $-10 +$ Id(20/4095)

Operating the card

- Open the GICS Tester
- Put the IP address “10.1.29.194”
- Check if the status is blue or not





If the status bar in the GICS Tester is not blue, reset the card at this button.

Exercise

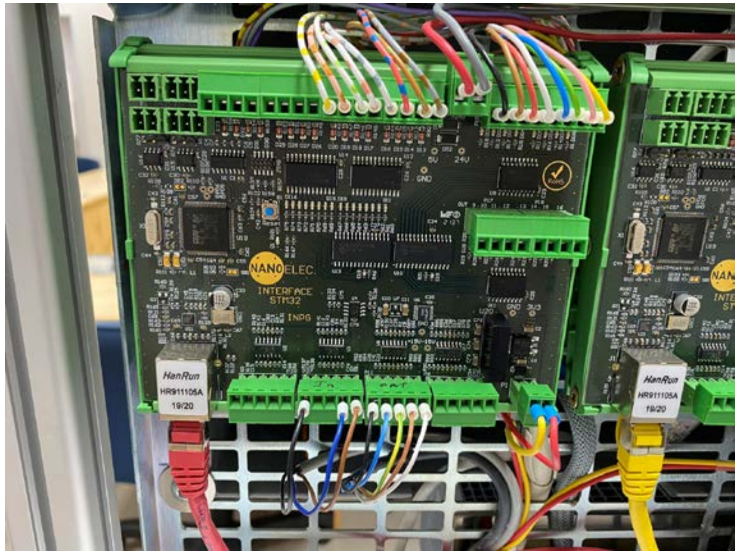
- Convert the following output voltage to the card's output
  - -10V
  - -8V
  - -5V
  - 0V
  - 4V
  - 7V
  - 9V
- Interpret the following card's input to input voltage
  - 10
  - 1000
  - 1700
  - 2400
  - 3900
  - 4050
  - 4095

Lab 03: Emulation of PLC analog I/O

Configuration

Output → Input

Input ← Output

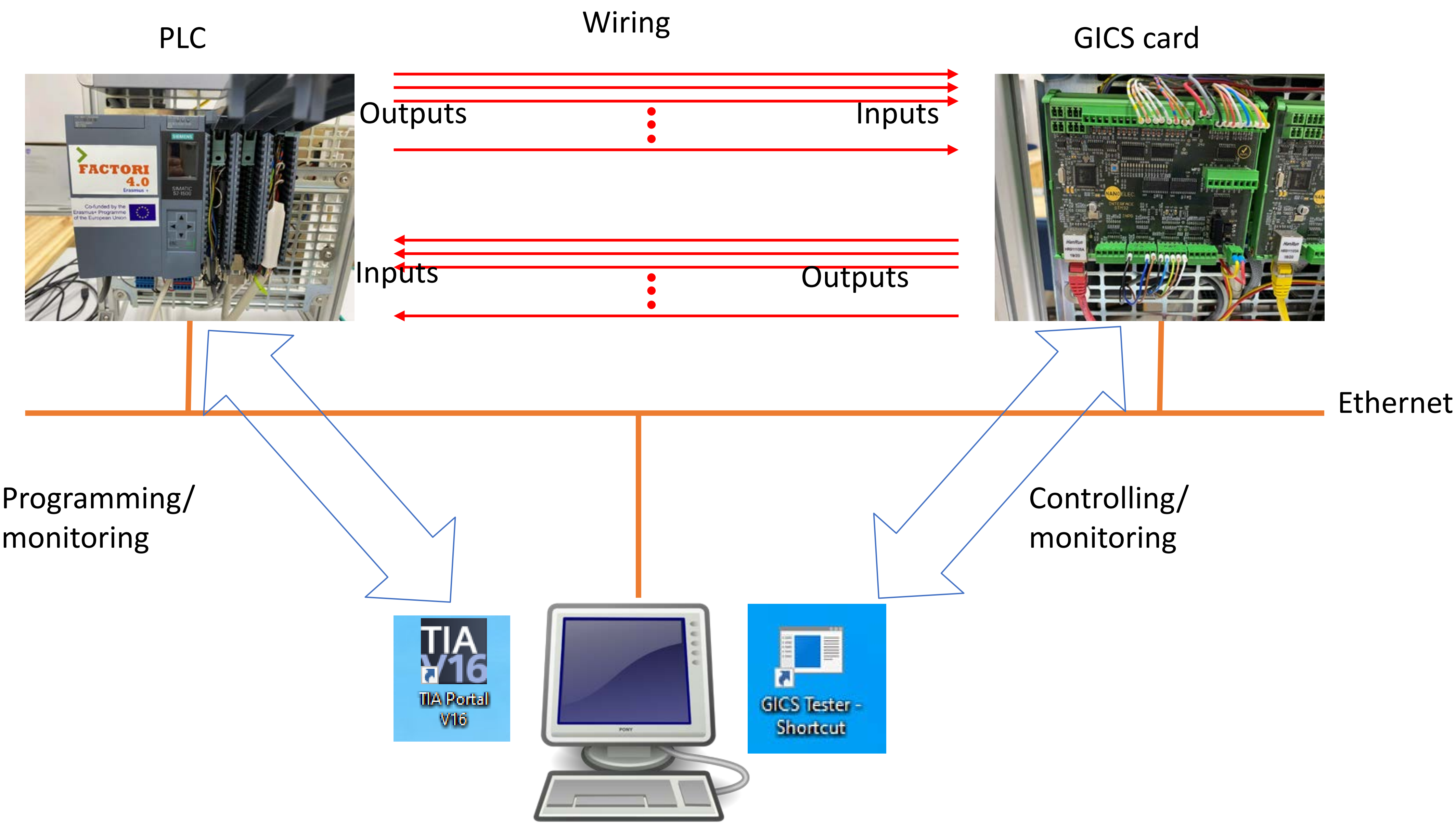


PLC

GICS card

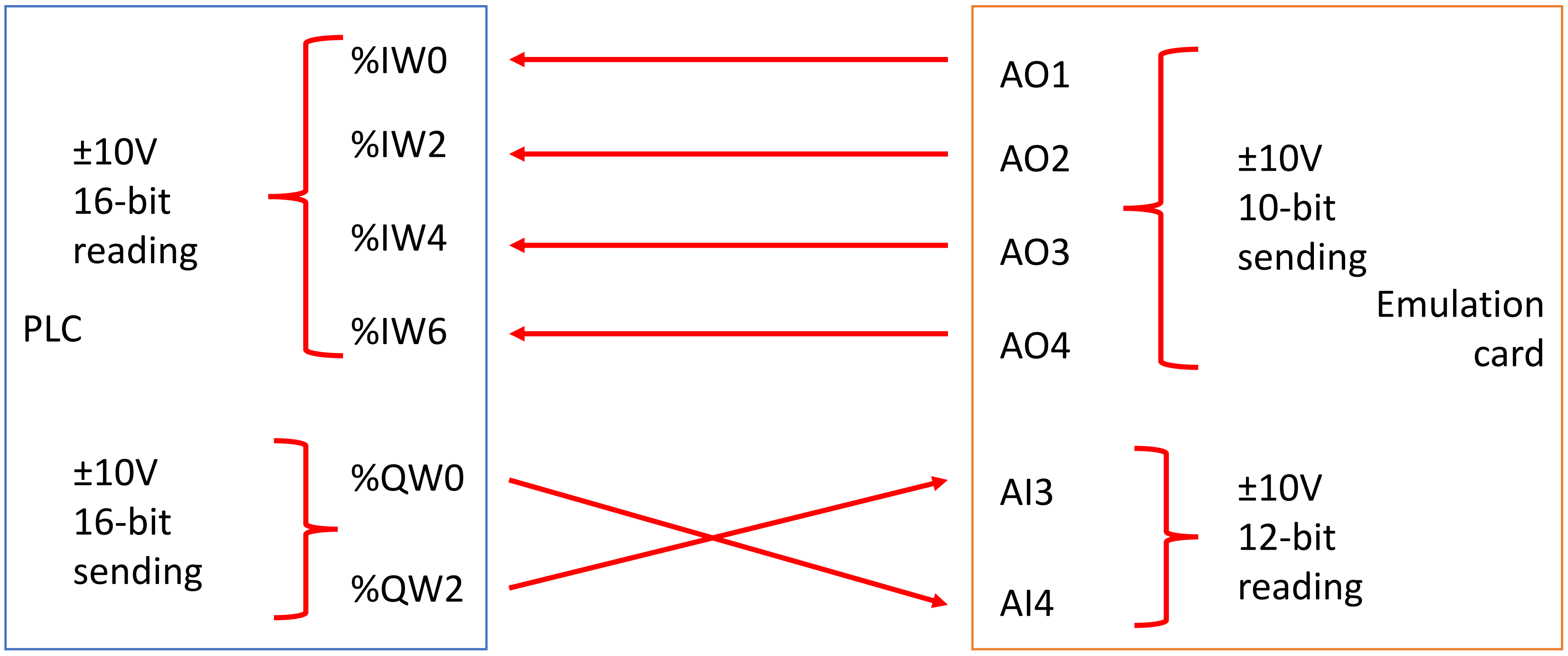
PLC I/O are programmed in TIA Portal

GICS's I/O are manipulated in a special program called "GICS Tester".



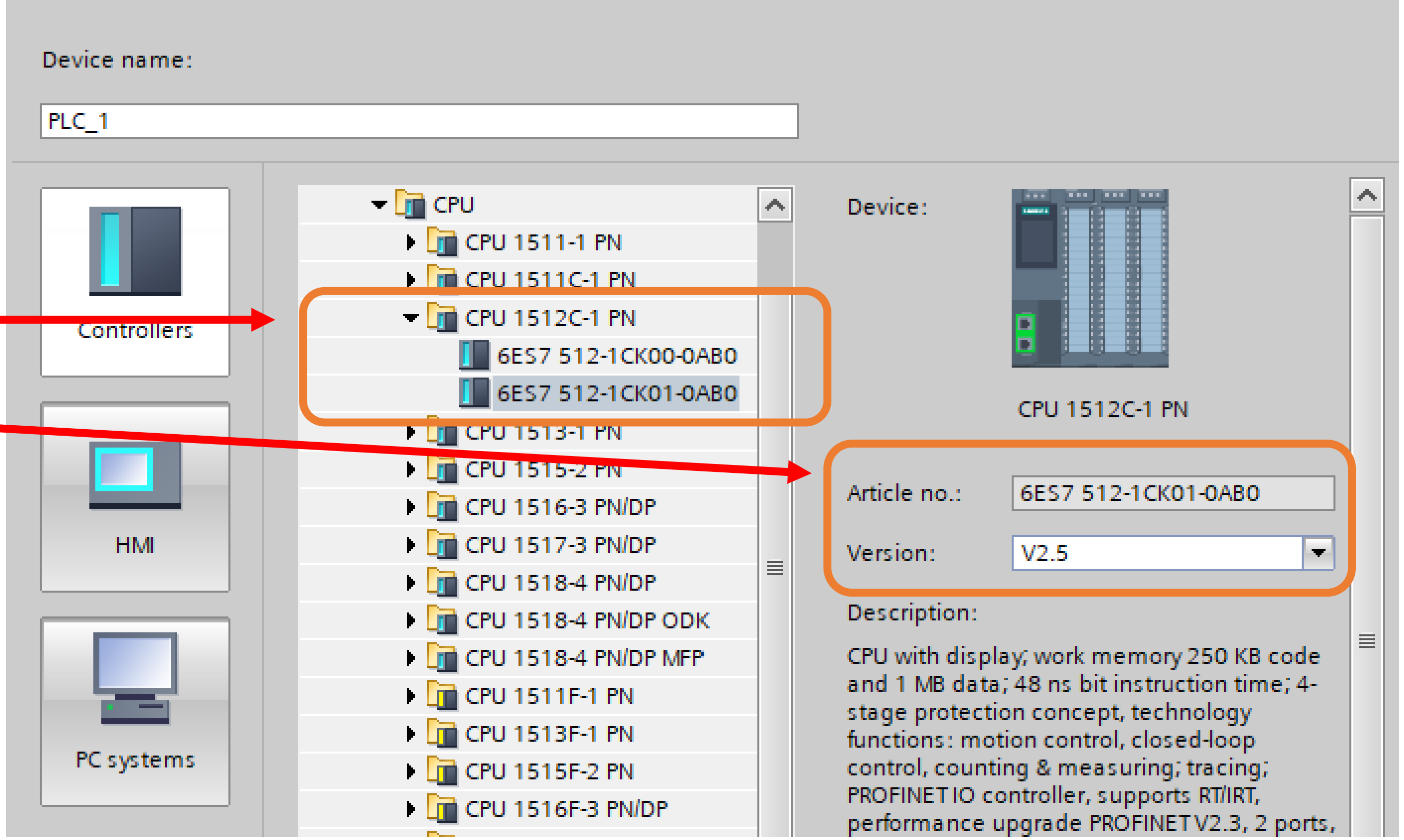
PLC
Wiring
GICS card
Outputs
Inputs
Inputs
Outputs
Ethernet
Programming/
monitoring
Controlling/
monitoring
TIA V16
TIA Portal V16
GICS Tester - Shortcut

Physical connection



Configuration

- Create a project with PLC of firmware version 2.5

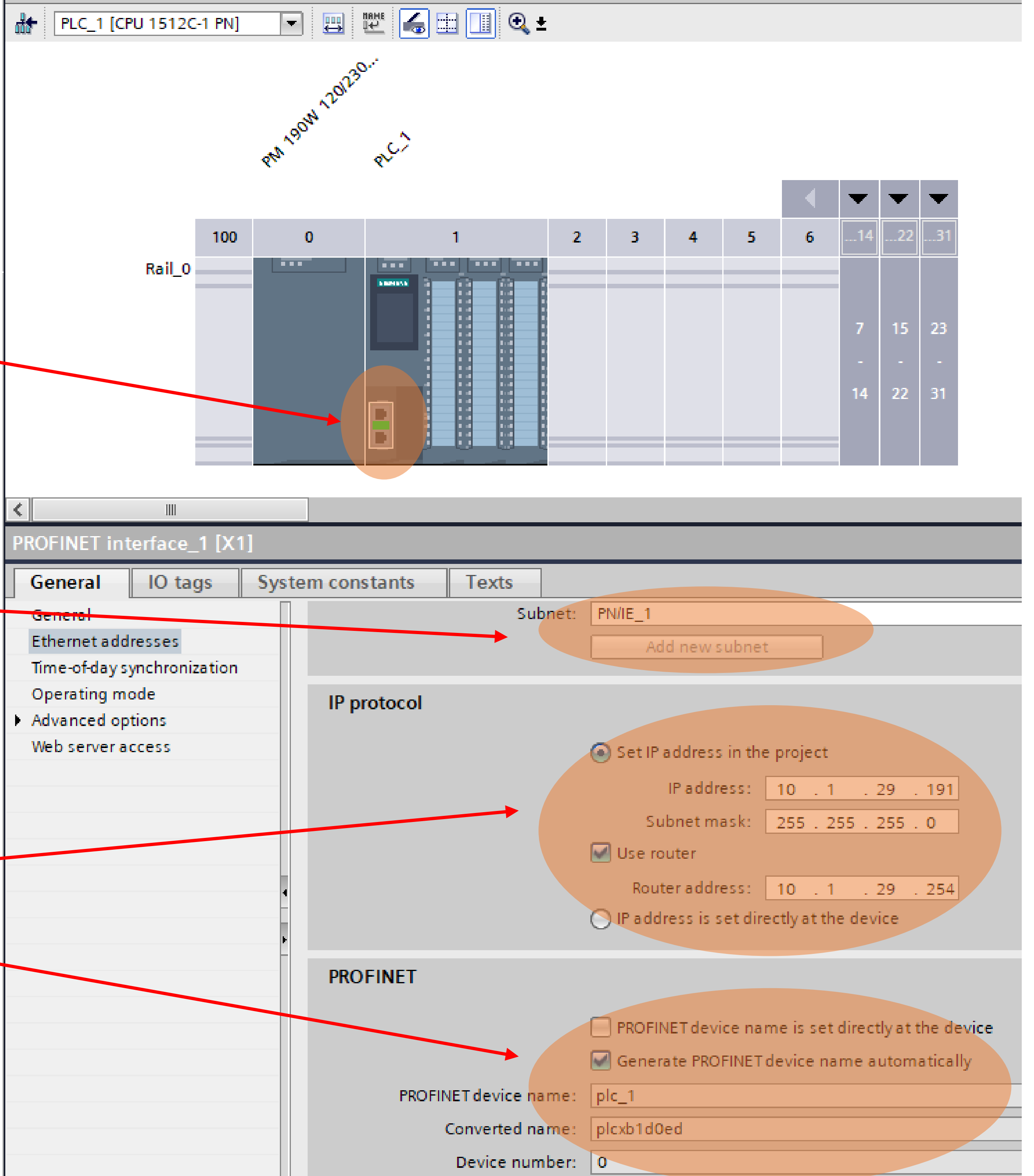


Network configuration

Select Ethernet symbol

Add new subnet

Set IP and Profinet



Tag table

Default tag table

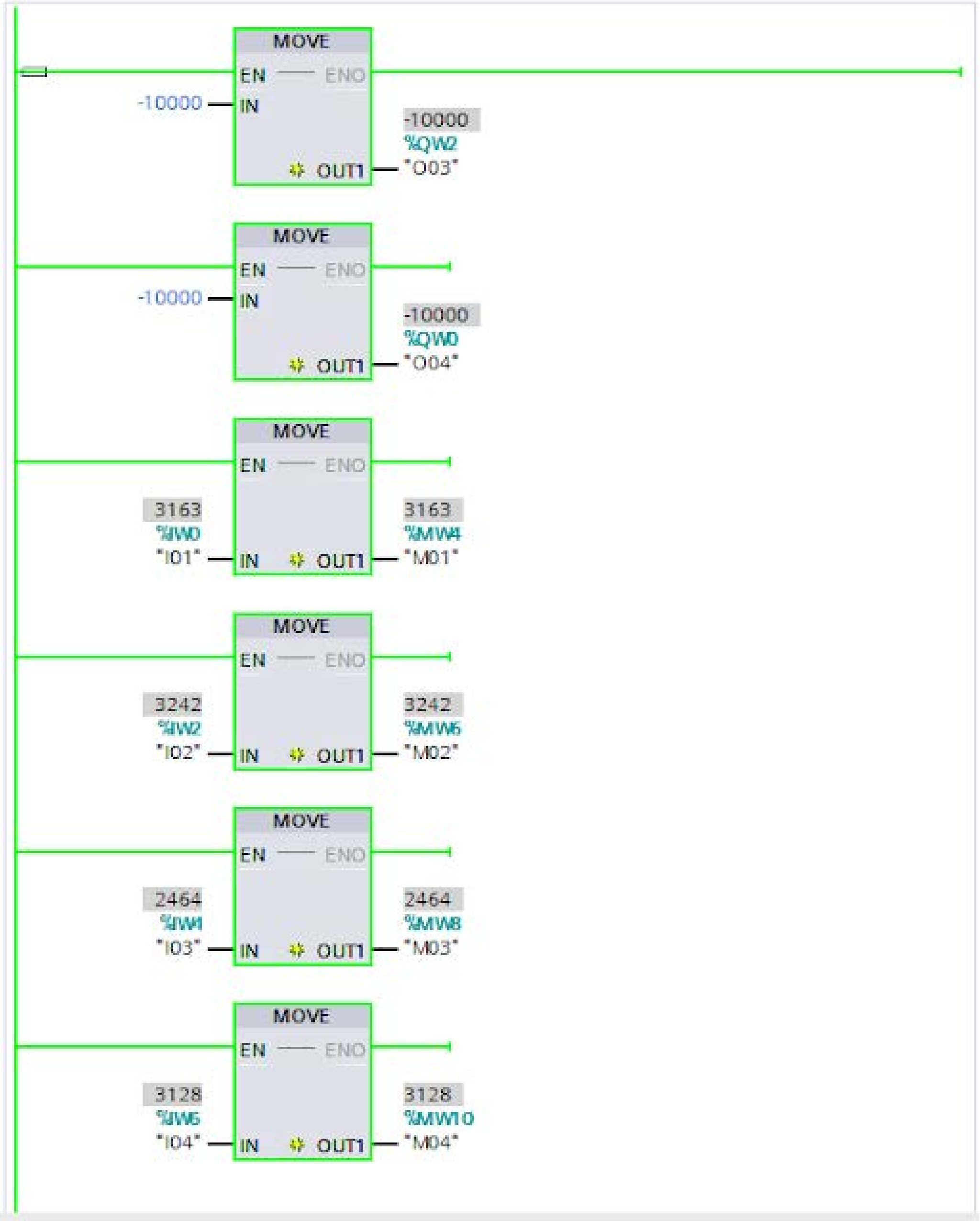
	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Supervis...	Comment
1	I01	Int	%IW0	☐	☑	☑	☑		
2	I02	Int	%IW2	☐	☑	☑	☑		
3	I03	Int	%IW4	☐	☑	☑	☑		
4	I04	Int	%IW6	☐	☑	☑	☑		
5	O04	Int	%QW0	☐	☑	☑	☑		
6	O03	Int	%QW2	☐	☑	☑	☑		
7	M01	Int	%MW4	☐	☑	☑	☑		
8	M02	Int	%MW6	☐	☑	☑	☑		
9	M03	Int	%MW8	☐	☑	☑	☑		
10	M04	Int	%MW10	☐	☑	☑	☑		
11	<Add new>			☐	☑	☑	☑		

Input mapping (rows 1–4)

Output mapping (rows 5–6)

Memory declaration for store each input (rows 7–10)

Ladder diagram for reading and sending



2-channel sending of value -10000

4-channel reading and store in memory

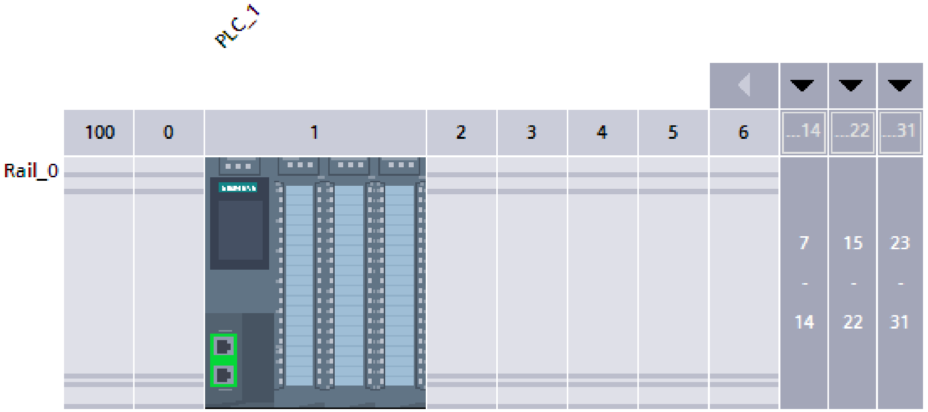
Compile and load config/code to PLC

In practical, at this point, the code and config are to be loaded to the destination PLC.

TIA Portal run in a PC

Compiled codes and configs

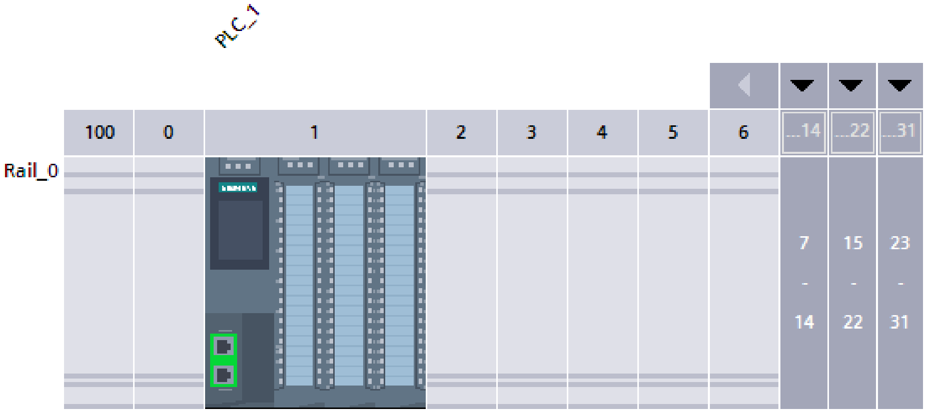
Download



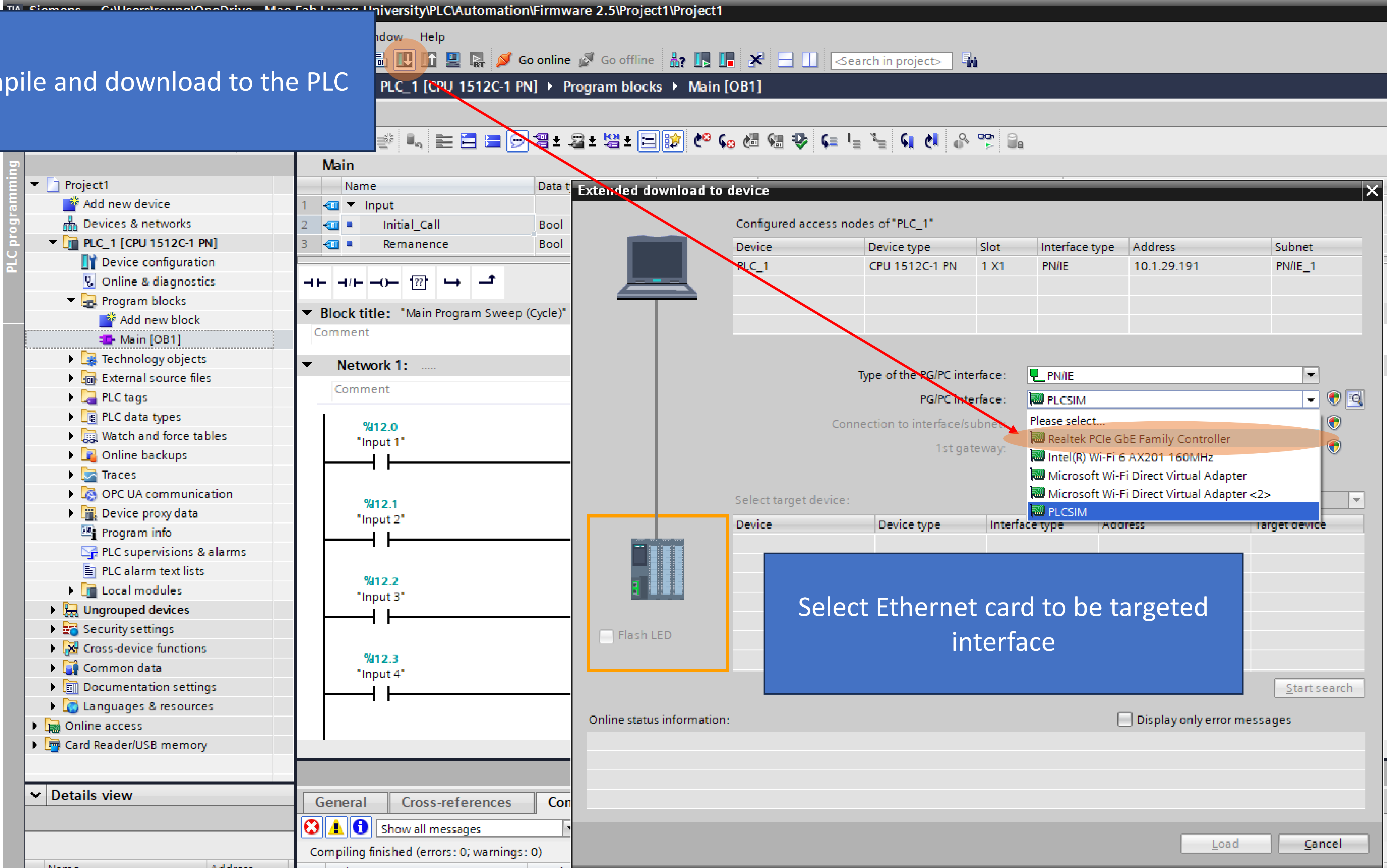
If there is no error, the TIA Portal and the targeted PLC are svnchronize. The PLC will run automatically but the Portal can monitor and debug.

TIA Portal run in a PC

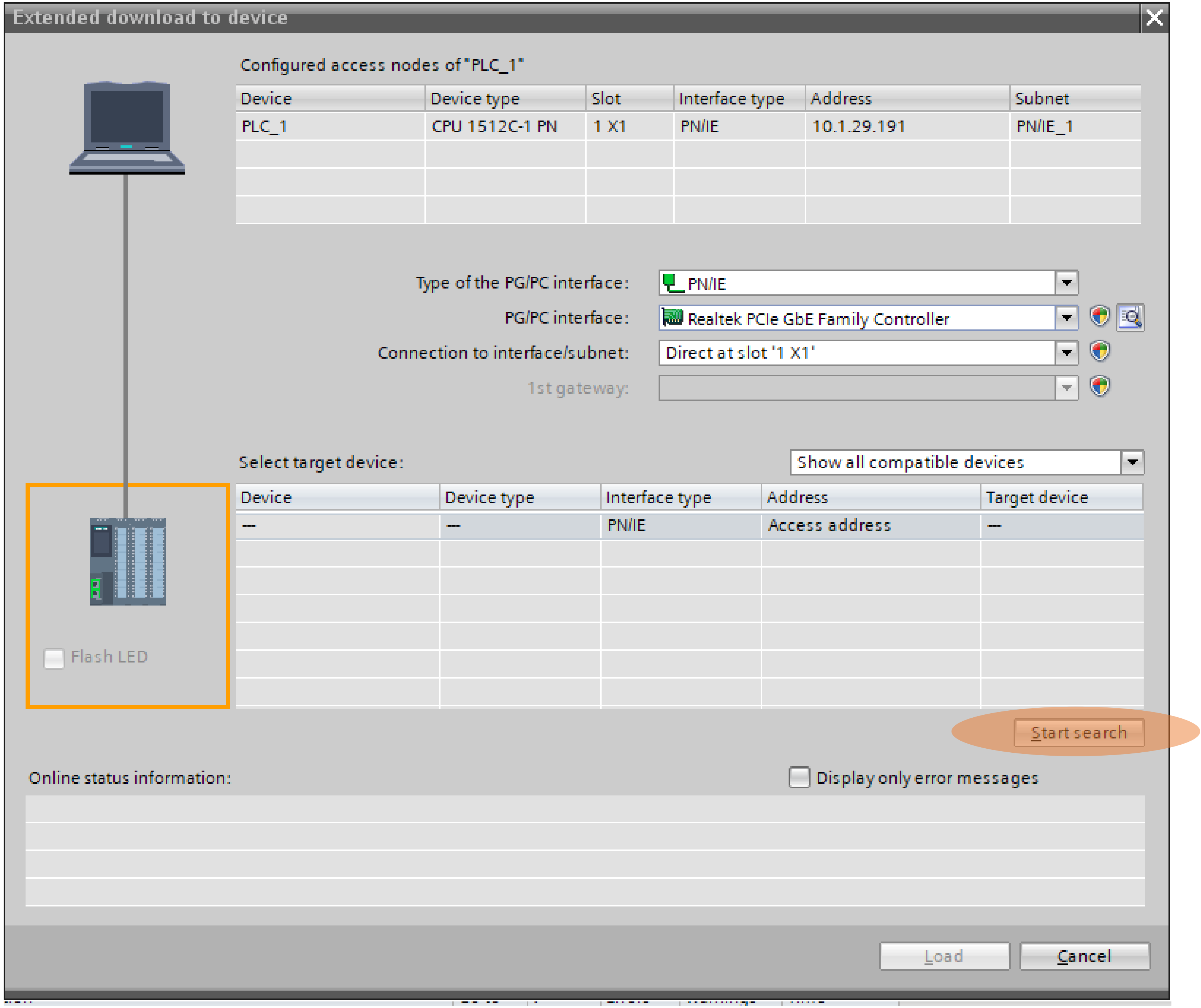
Synchronize



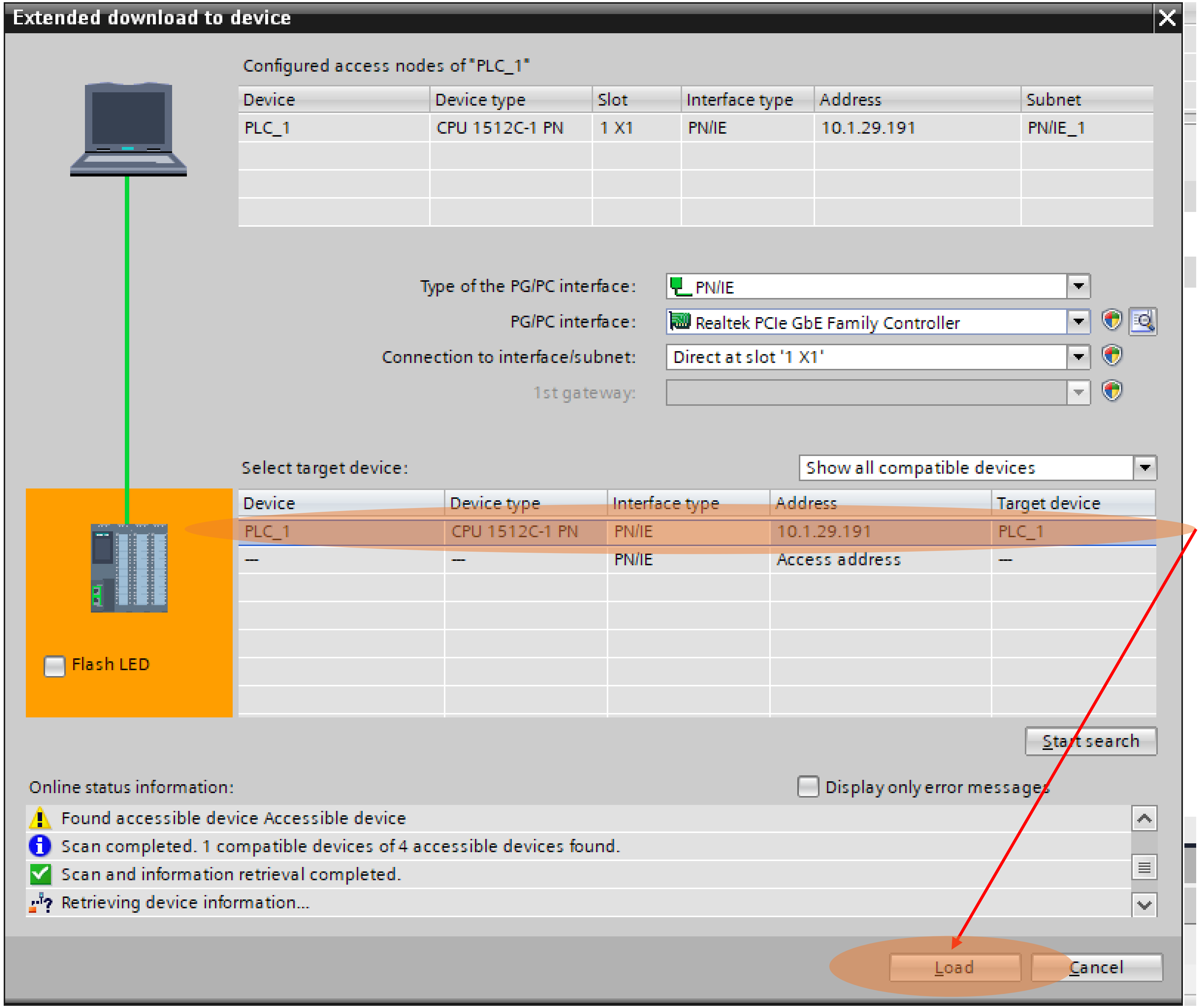
Compile and download to the PLC



Select Ethernet card to be targeted interface

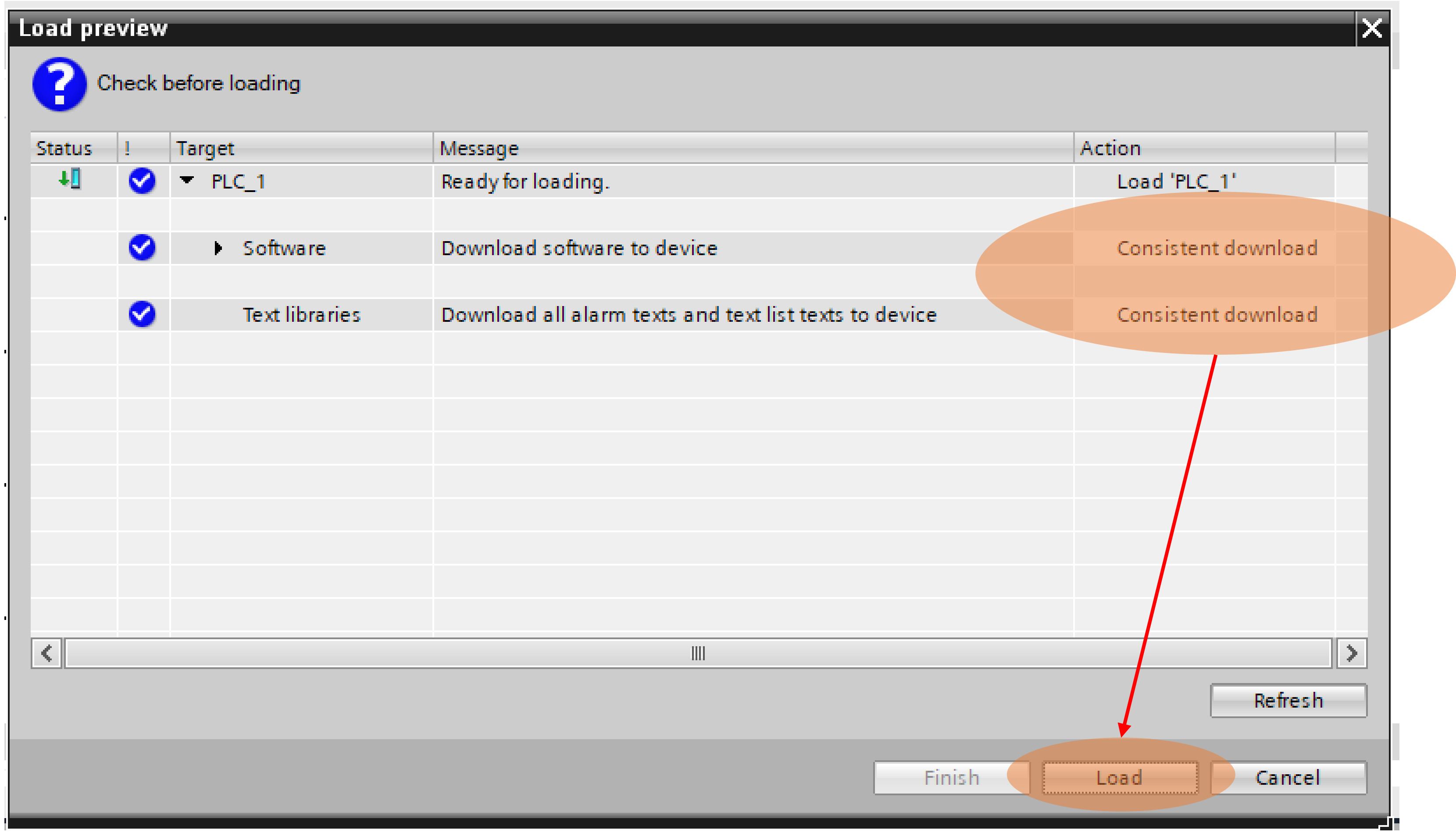


Search for the PLC in the subnet



Found the targeted PLC and select it

Load the config and code to the PLC



Check the loading option

Load the config and code to the PLC

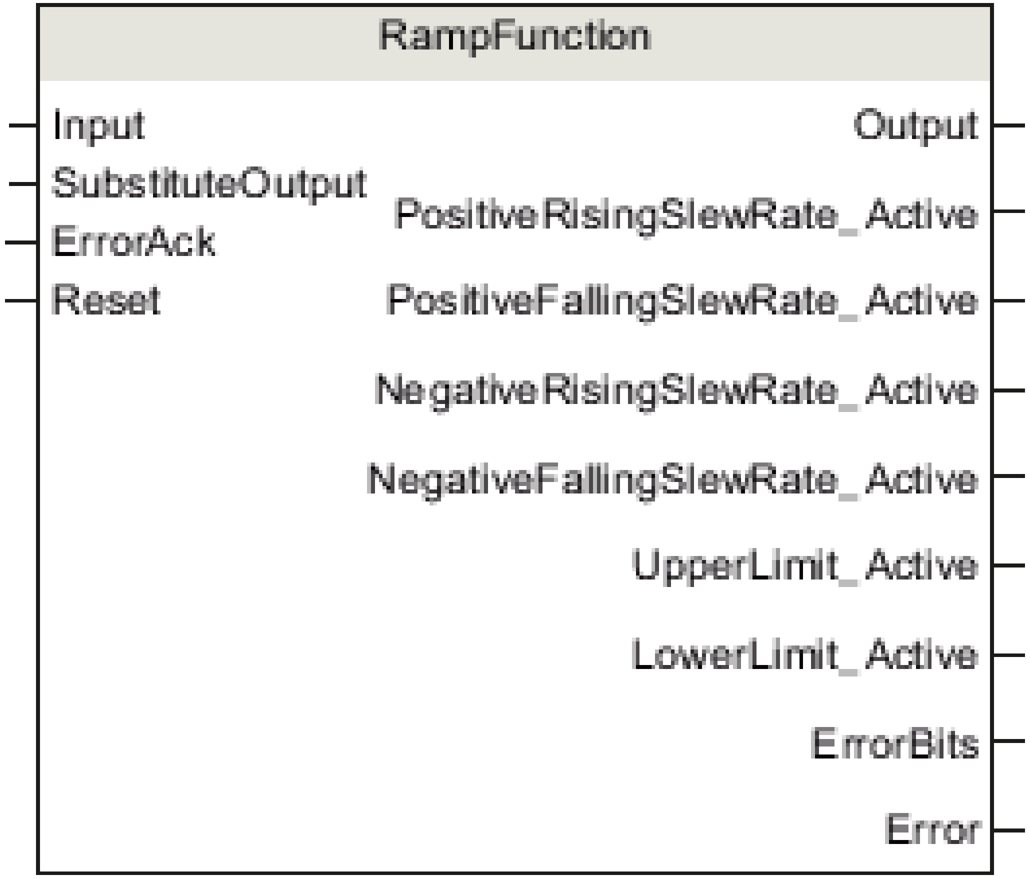
Exercise

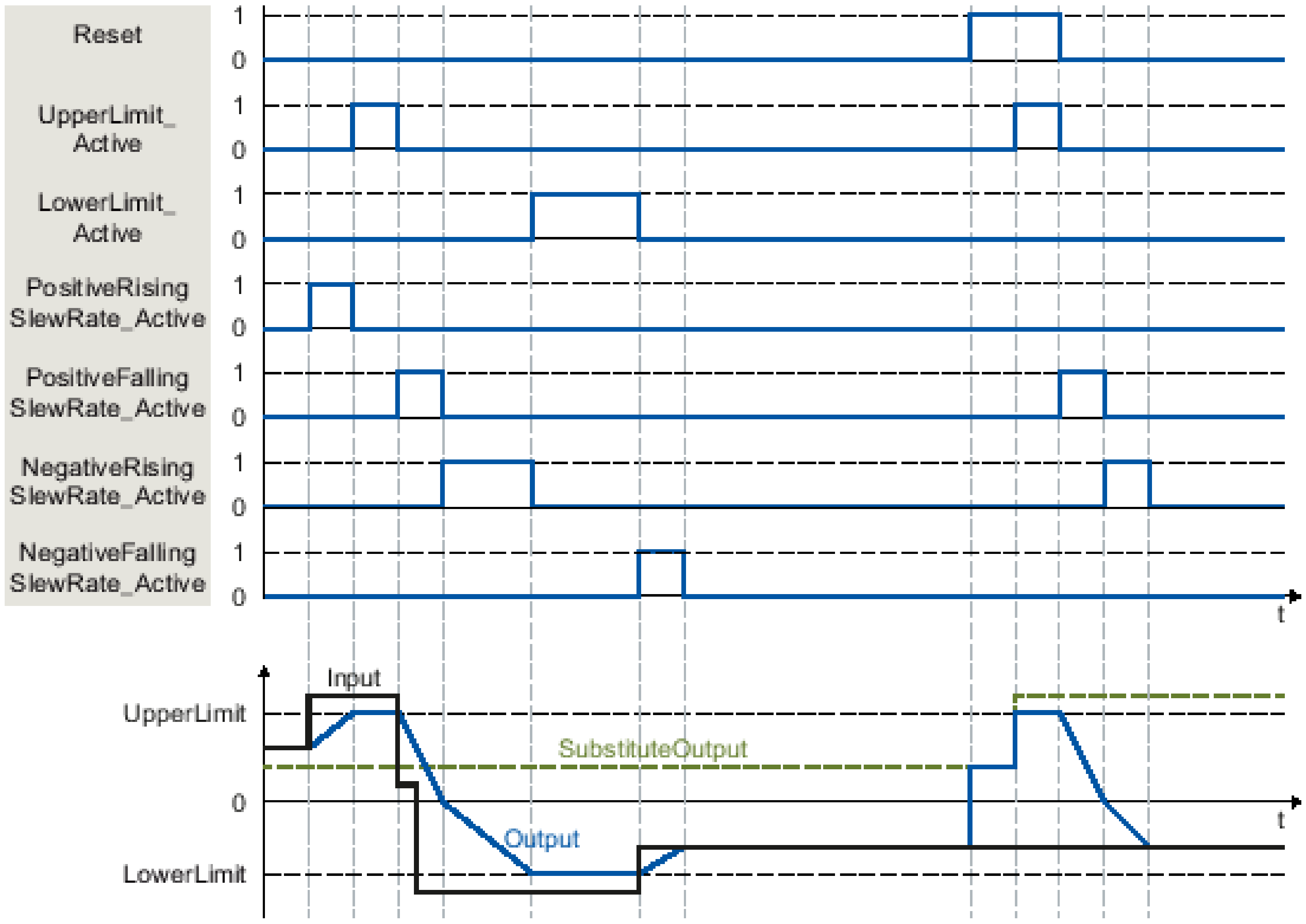
- Send the following value from the card and observe the value read by the PLC
  - 0
  - 100
  - 300
  - 500
  - 800
  - 1000
- Send the following value from PLC and observe the value read by the card
  - -27648
  - -15000
  - -5000
  - 0
  - 10000
  - 20000

Lab 04: Simulation of analog output

RampFunction

- Act as an integrator
  - Increase or decrease by a certain amount (SlewRate) over a second)
  - Naturally linear but can adjust SlewRate to act as nonlinear function
- Can simulate output as a function of driving force
- Output both positive and negative with adjusted limitation (UpperLimit or LowerLimit)
- Can reset to the specified value (SubstituteOutput) which is default at “0”





Reset
UpperLimit_
Active
LowerLimit_
Active
PositiveRising
SlewRate_Active
PositiveFalling
SlewRate_Active
NegativeRising
SlewRate_Active
NegativeFalling
SlewRate_Active
1
0
t
Input
UpperLimit
SubstituteOutput
0
Output
LowerLimit
t

Parameter	Data type	Default
Input	REAL	0.0
SubstituteOutput	REAL	0.0

Tag	Data type	Default
PositiveRisingSlewRate	REAL	10.0
PositiveFallingSlewRate	REAL	10.0
NegativeRisingSlewRate	REAL	10.0
NegativeFallingSlewRate	REAL	10.0
UpperLimit	REAL	100.0
LowerLimit	REAL	0.0

Output will gradually increase or decrease by the defined “SlewRate” to the “Input”

There are four “SlewRate” to be set:

- PositiveRising
- PositiveFalling
-NegativeRising
-NegativeFalling

Output is bounded by UpperLimit and LowerLimit

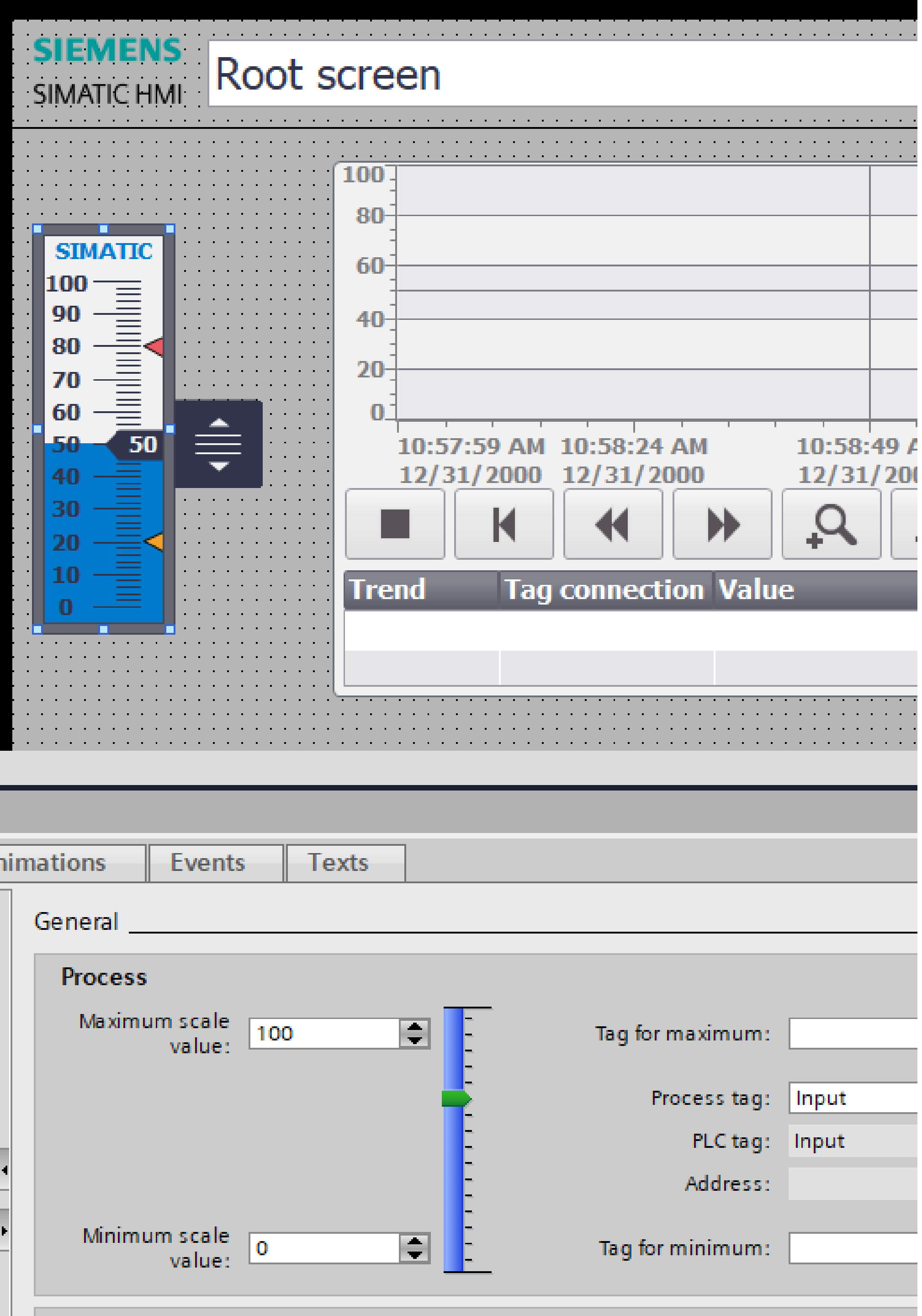
PLC and HMI tags

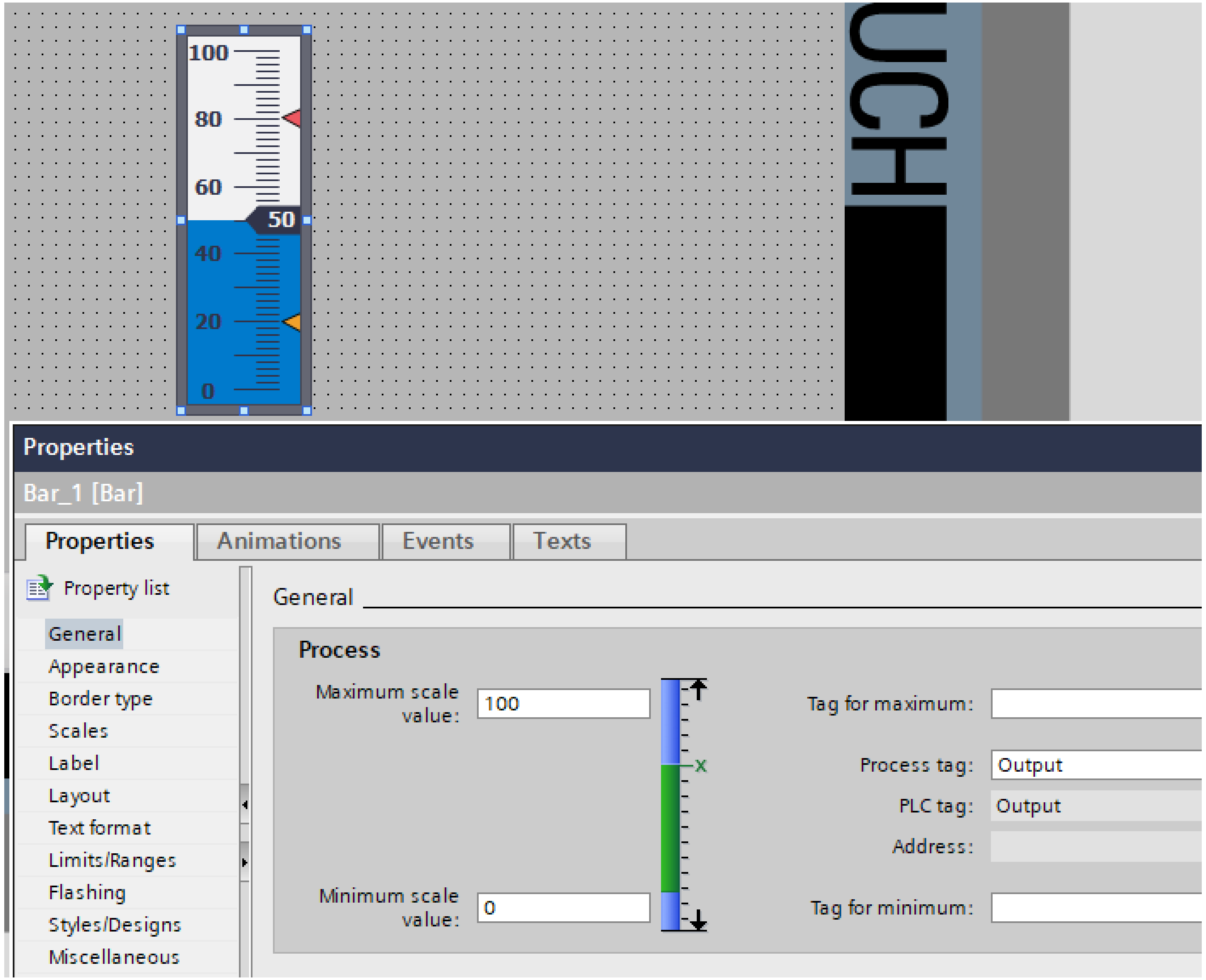
Default tag table PLC

		Name	Data type	Address	Retain	Acces...	Writa...	Visibl...
1		Output	Int	%QW0	☐	☑	☑	☑
2		Input	Int	%MW0	☐	☑	☑	☑
3		<Add new>			☐	☑	☑	☑

Default tag table HMI

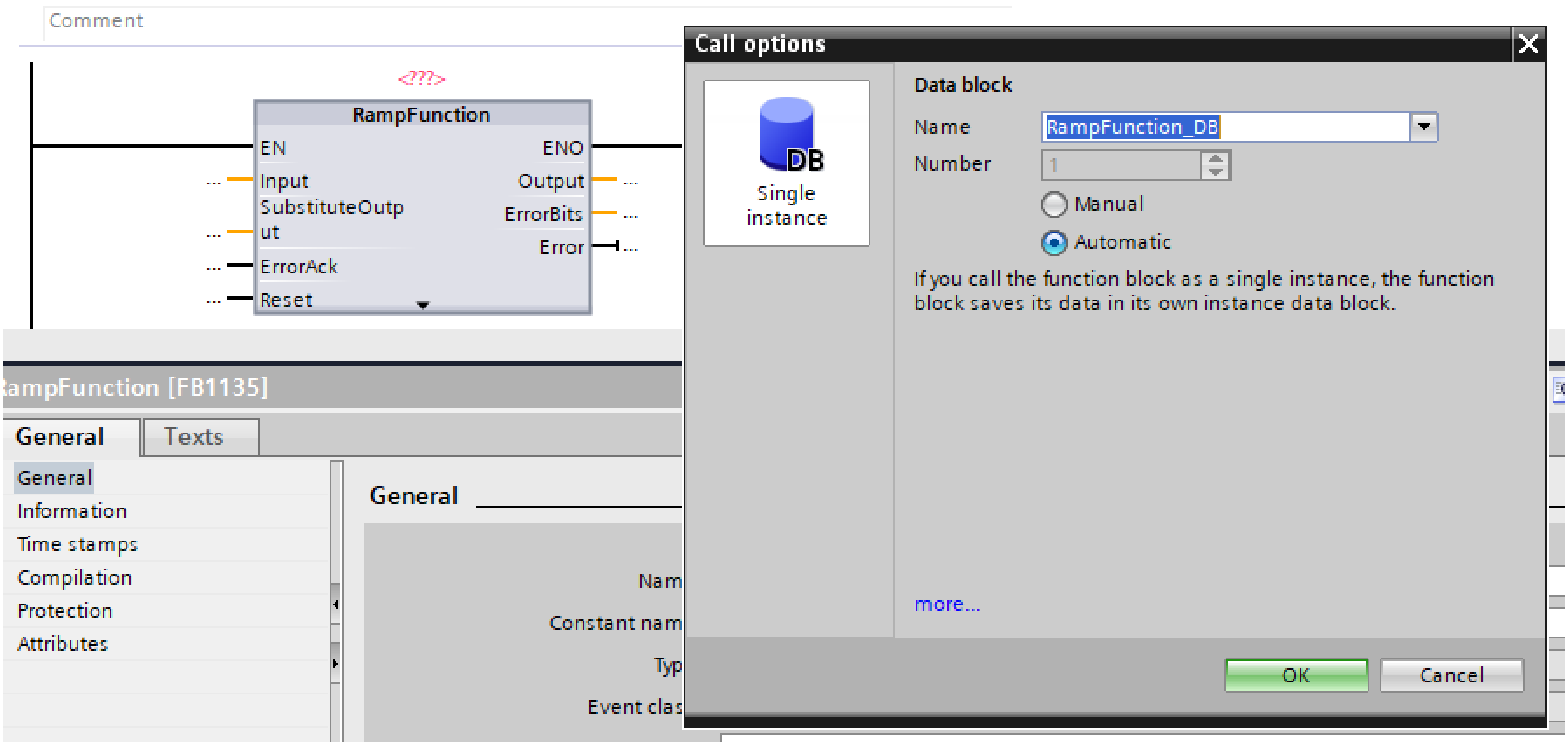
	Name ▲	Data type	Connection	PLC name	PLC tag	Address	Access mode	Acquisition cycle
	Tag_ScreenNumber	UInt	<Internal tag>		<Undefi...			1 s
	Input	Int	HMI_Connectio...	PLC_1	Input		<symbolic ac...	100 ms
	Output	Int	HMI_Connectio...	PLC_1	Output		<symbolic ac...	100 ms



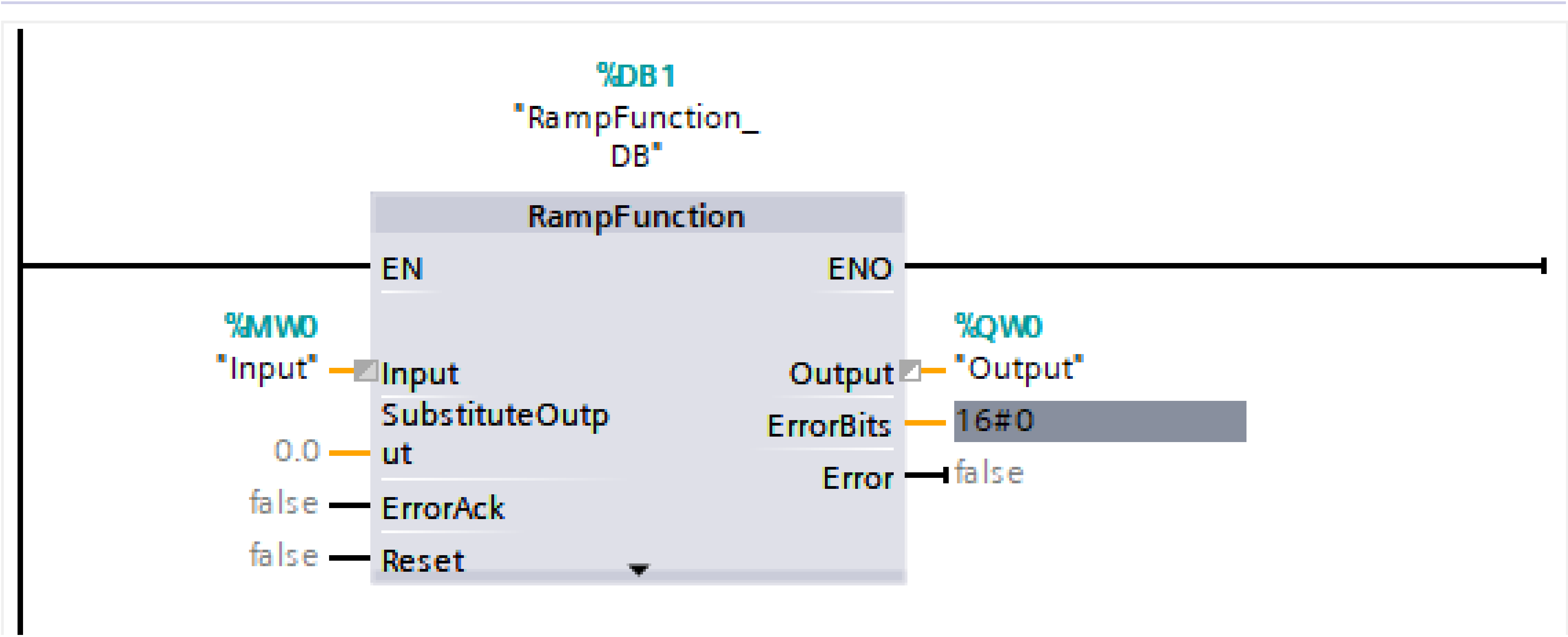


Config and link Slide input and bar output to HMI tags

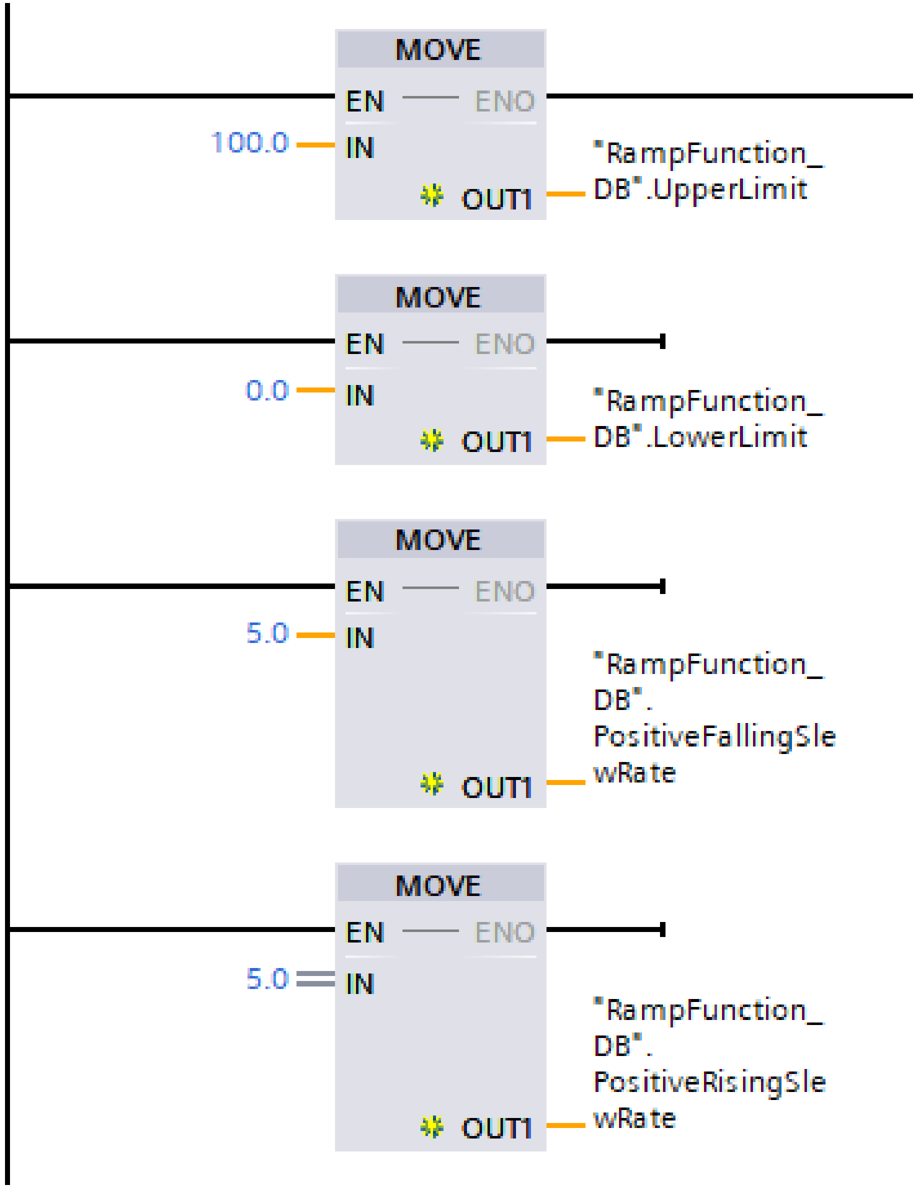
Add a RampFunction block



Link input and output to the RampFunction



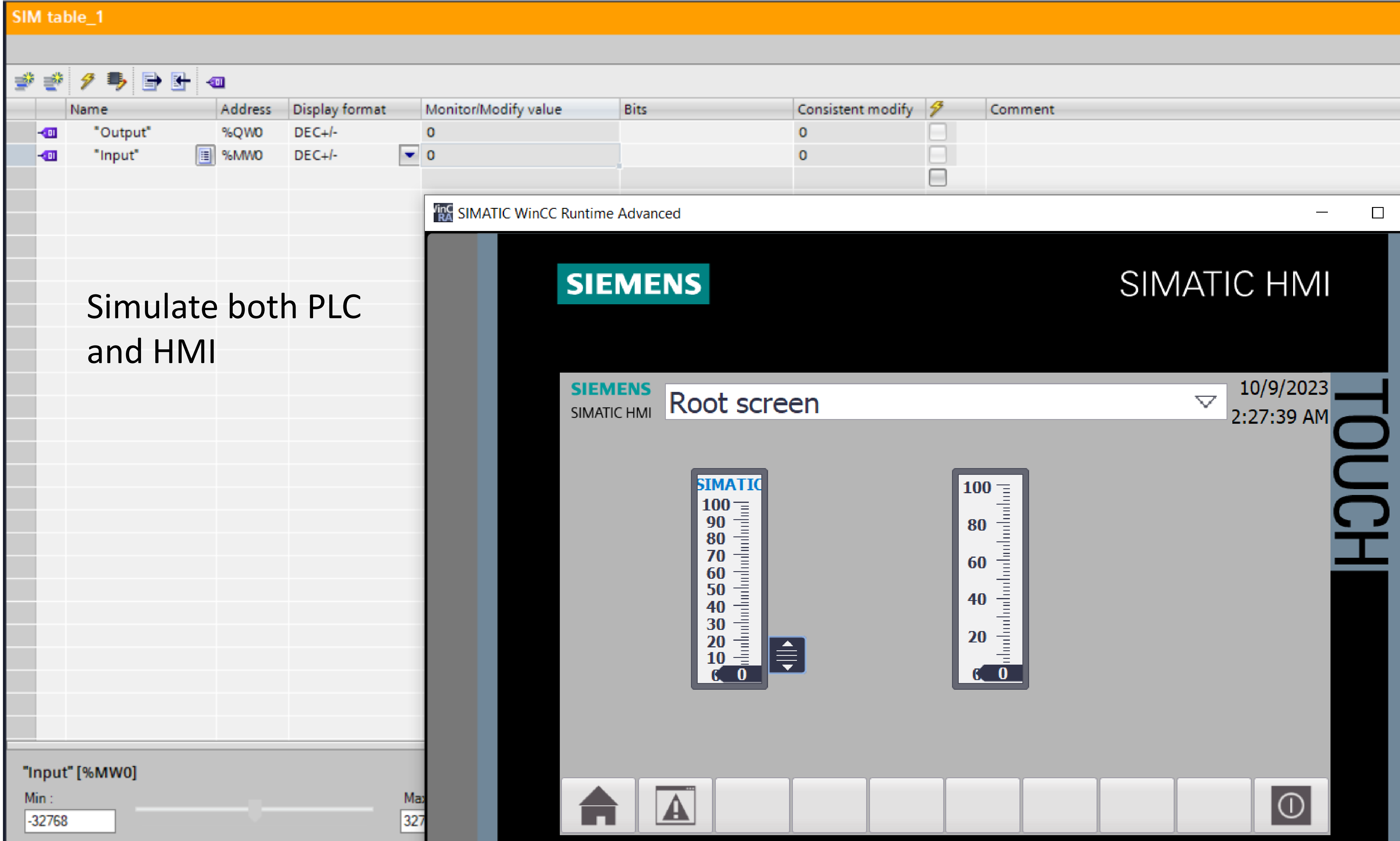
Ex: Simulate the function of y(t) = 5t via the RampFunction
The slope of this function is 5, so SlewRate = 5



Set the upper and lower limits

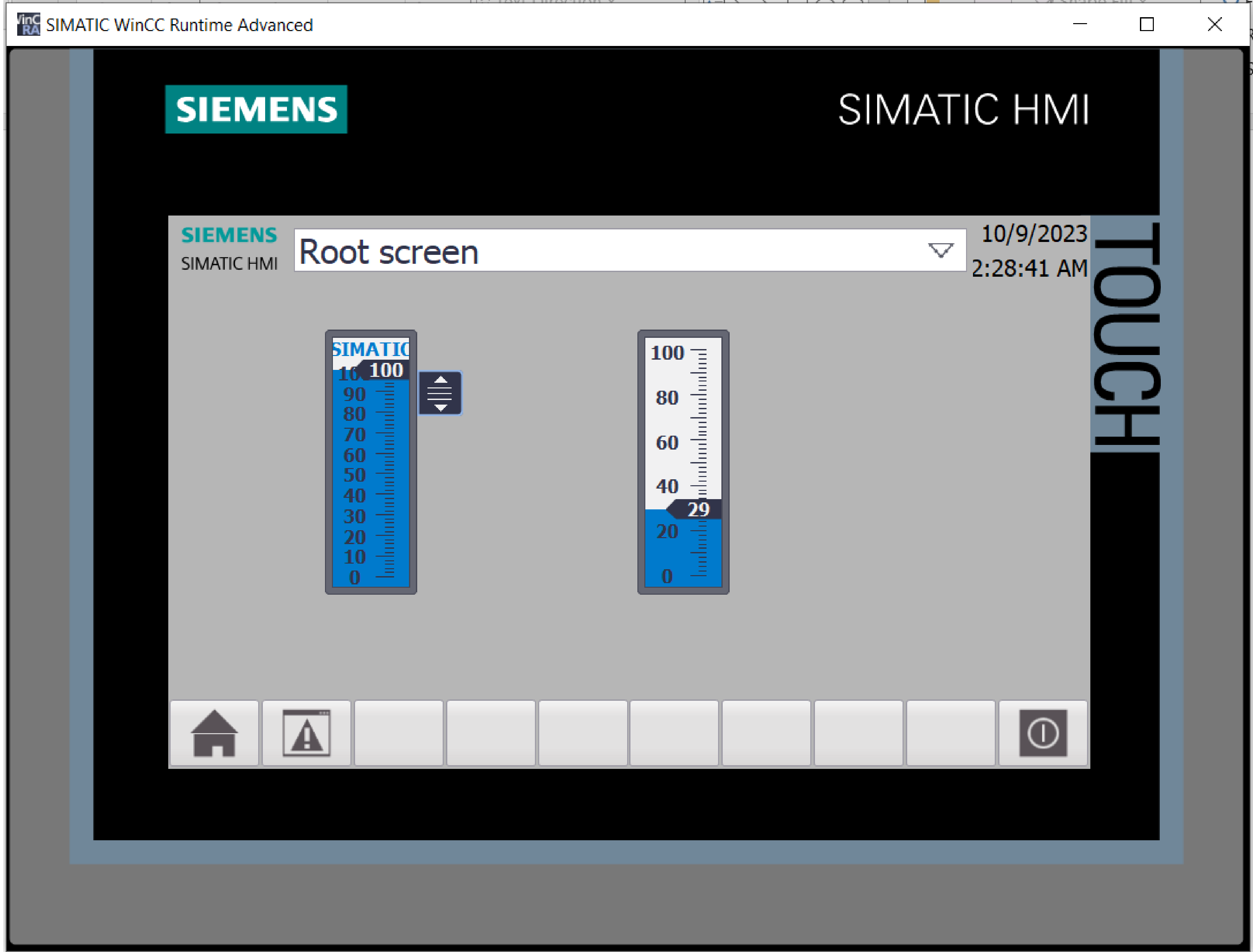
-Set the Rising and Falling slew rate
-Only positive rising and falling are set because the boundary is from 0 - 100

Simulate both PLC and HMI



Set the slide bar on the left to set the desired output to 100

The output bar on the right will move up to 100 in 20 seconds

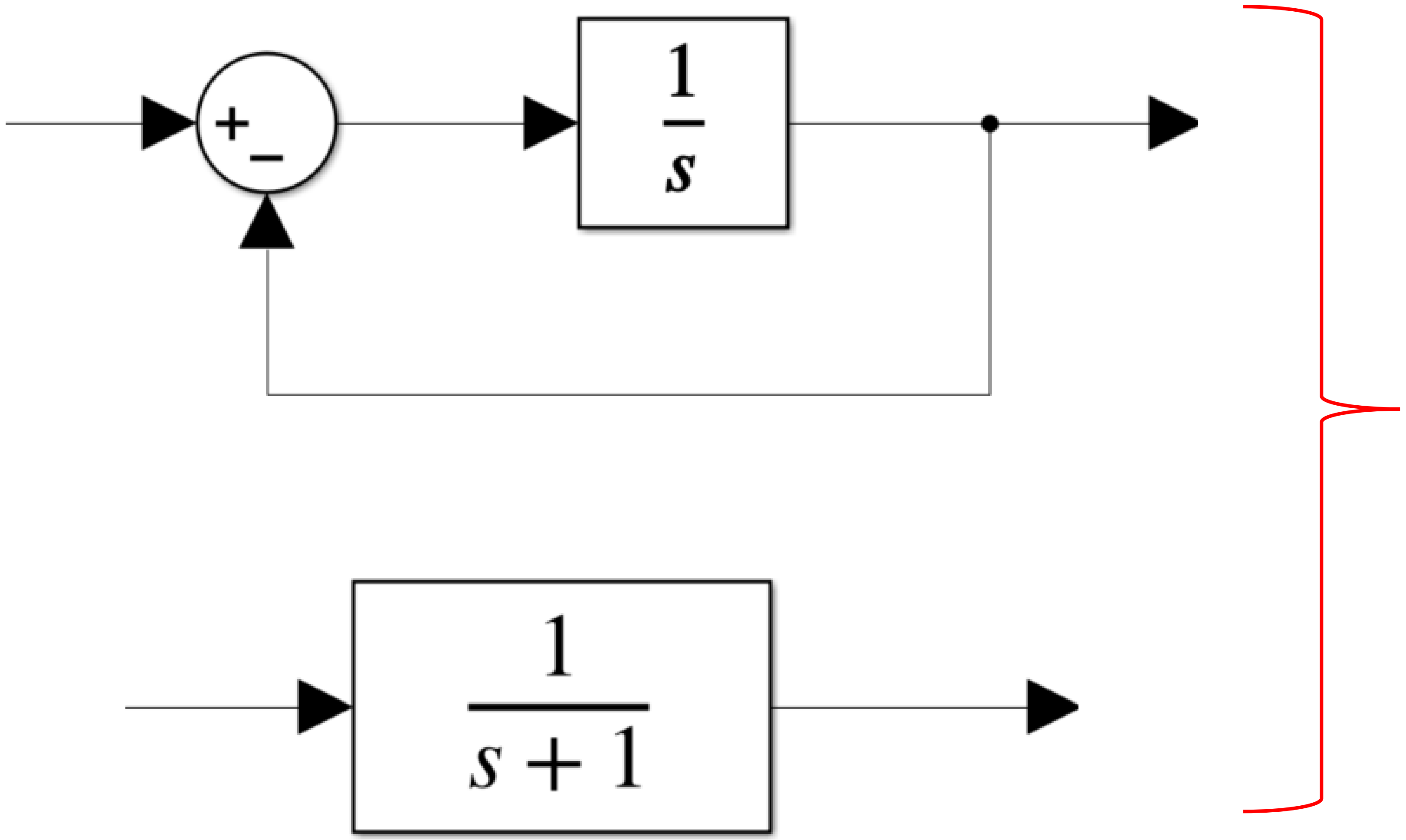


Exercise

- SlewRate can be non-constant via a specific function to generate nonlinear output
  - Ex. Output from a RampFunction can be input to another RampFunction
- Output can be manipulate by external function to further adjusting
- Try to simulate the following functions:
  - $y(t) = 5t + 100$
  - $y(t) = t^2$

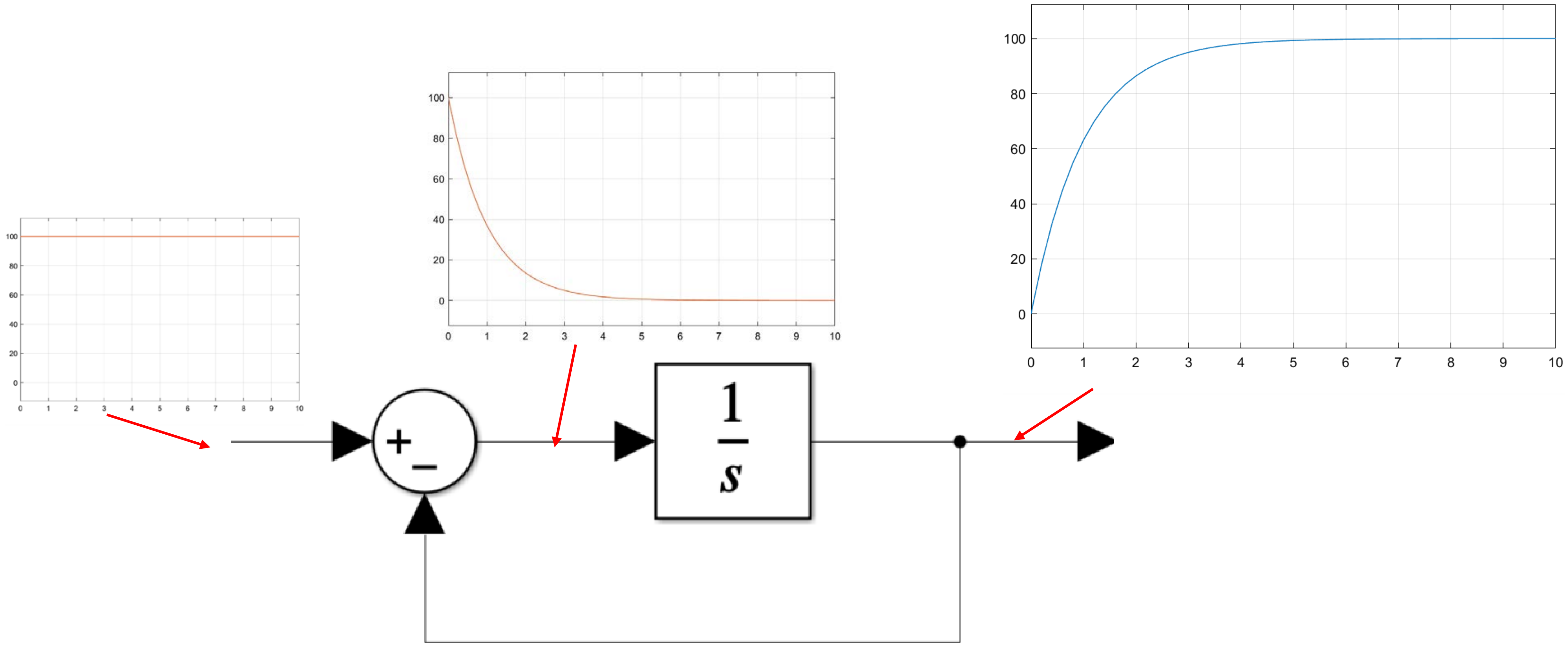
Lab 05: Simulation of feedback control

A simple feedback control system

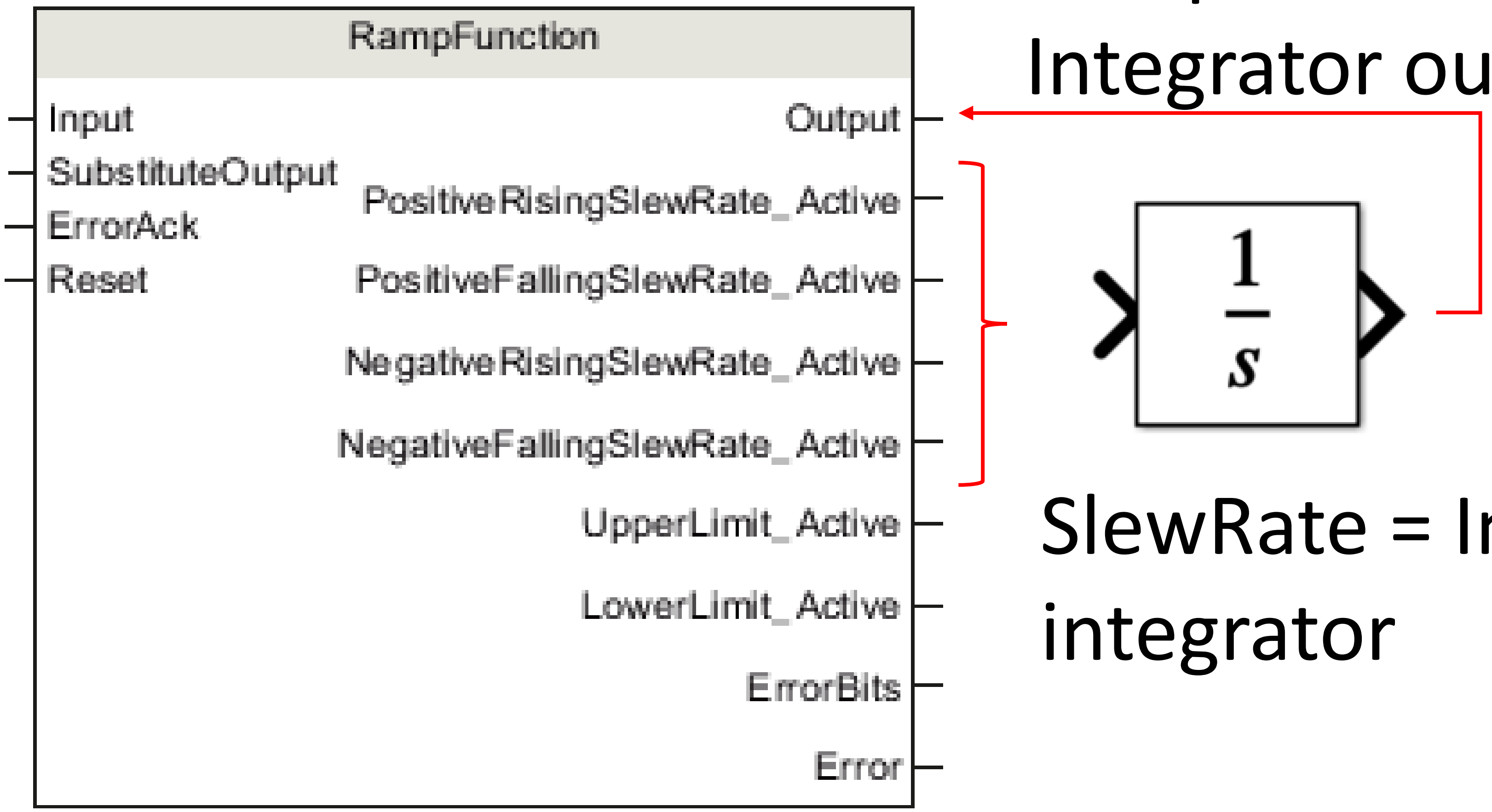


Equivalent to each other: one in feedback form, another in reduced transfer function form

Signals



Implementing an Integrator by RampFunction



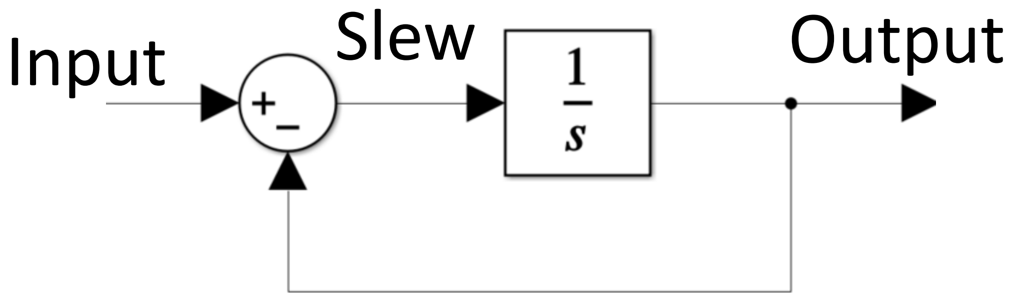
RampFunction Output = Integrator output

SlewRate = Input of an integrator

PLC Tag table

Default tag table

		Name	Data type	Address
1		slew	Int	%MW0
2		aSlew	Int	%MW2
3		force	Int	%MW4
4		rSlew	Real	%MD6
5		Input	Int	%IW0
6		Output	Int	%QW0
7		Reset	Bool	%I10.0

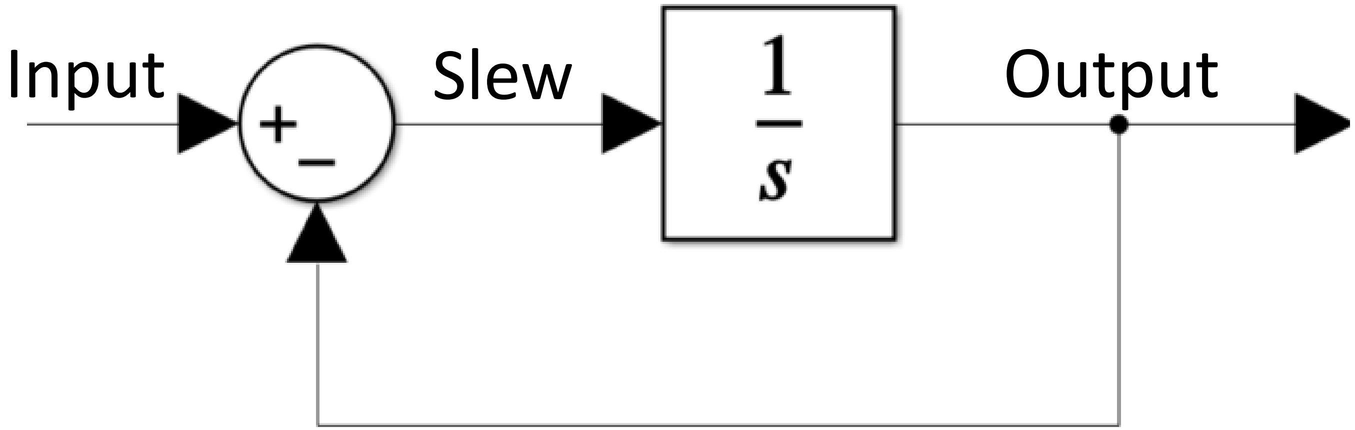


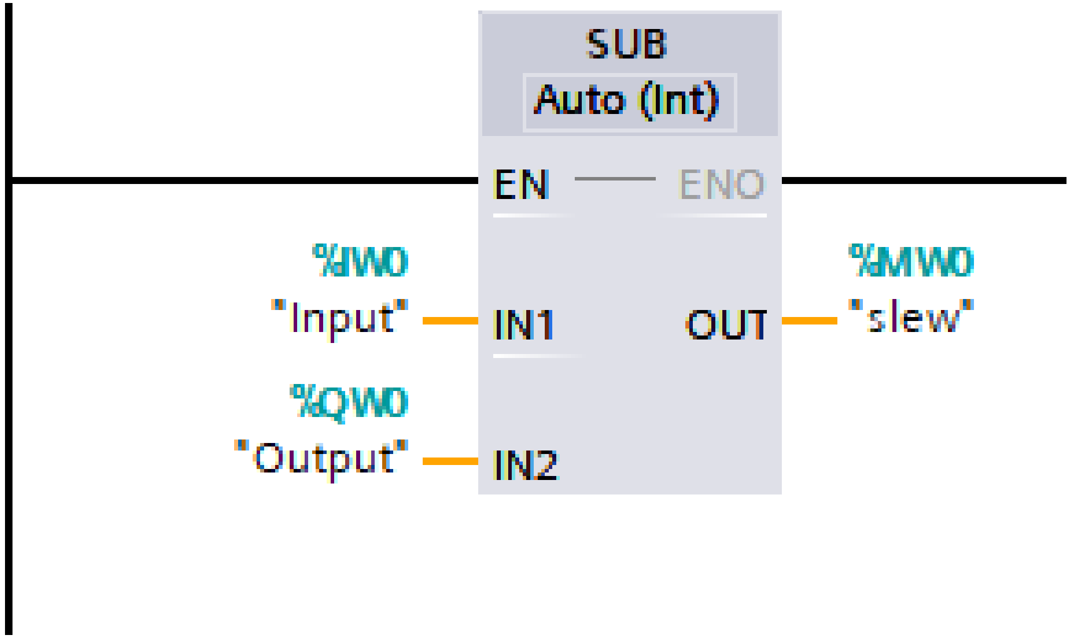
For setting SlewRate in RampFunction

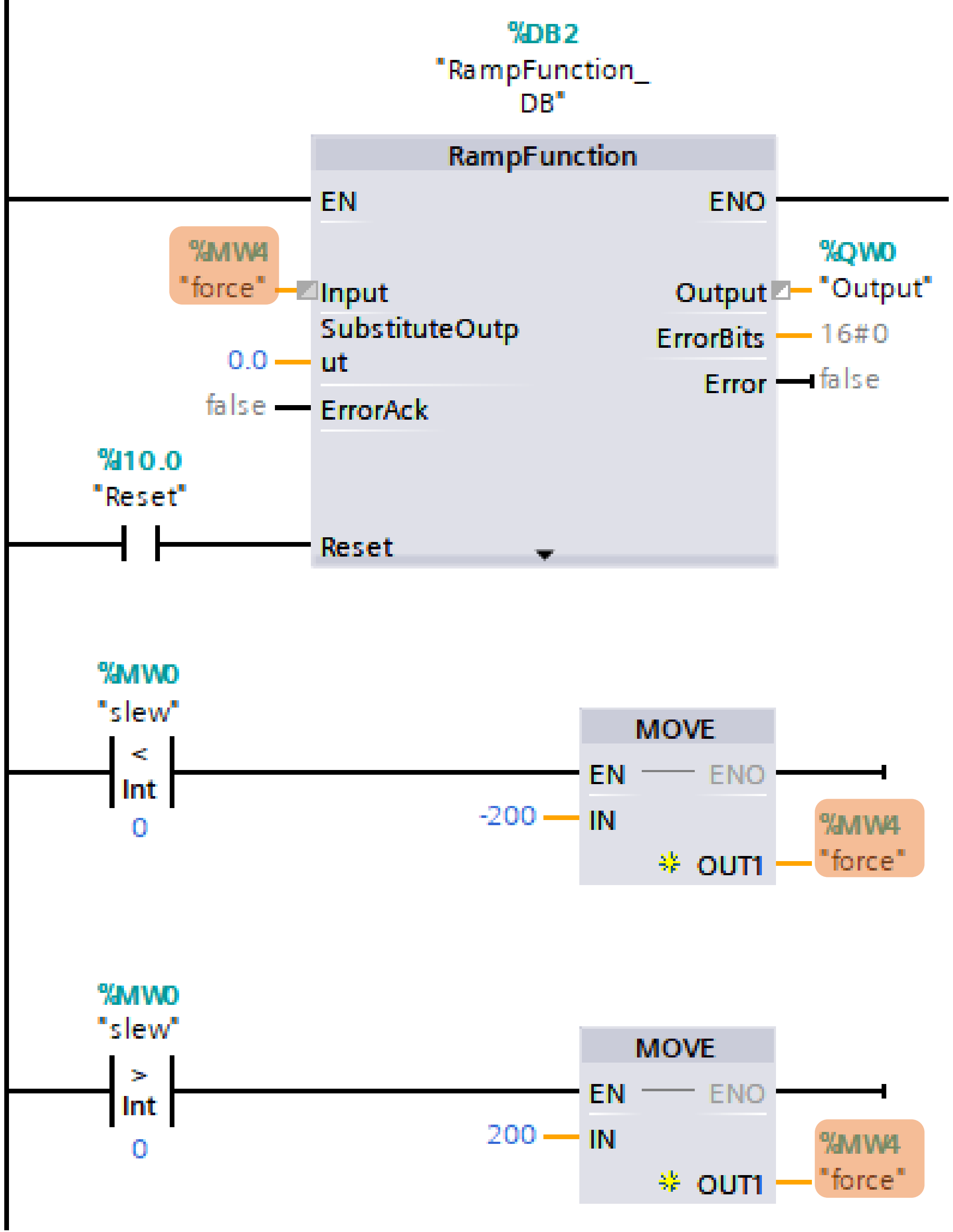
For forcing RampFunction’s output up or down

For resetting RampFunction

Ladder Diagram for feedback loop



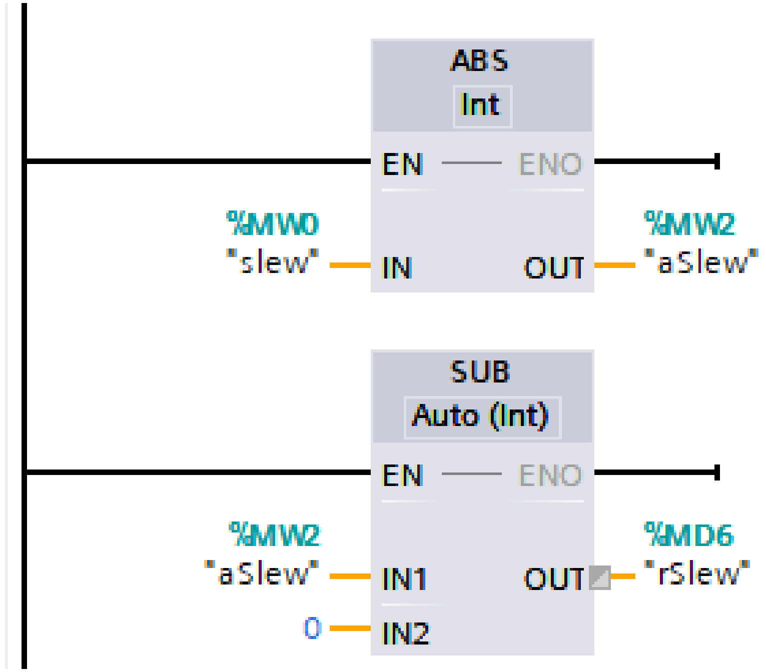




Output of RampFunction will move up or down according to the variable “force” applied to its Input

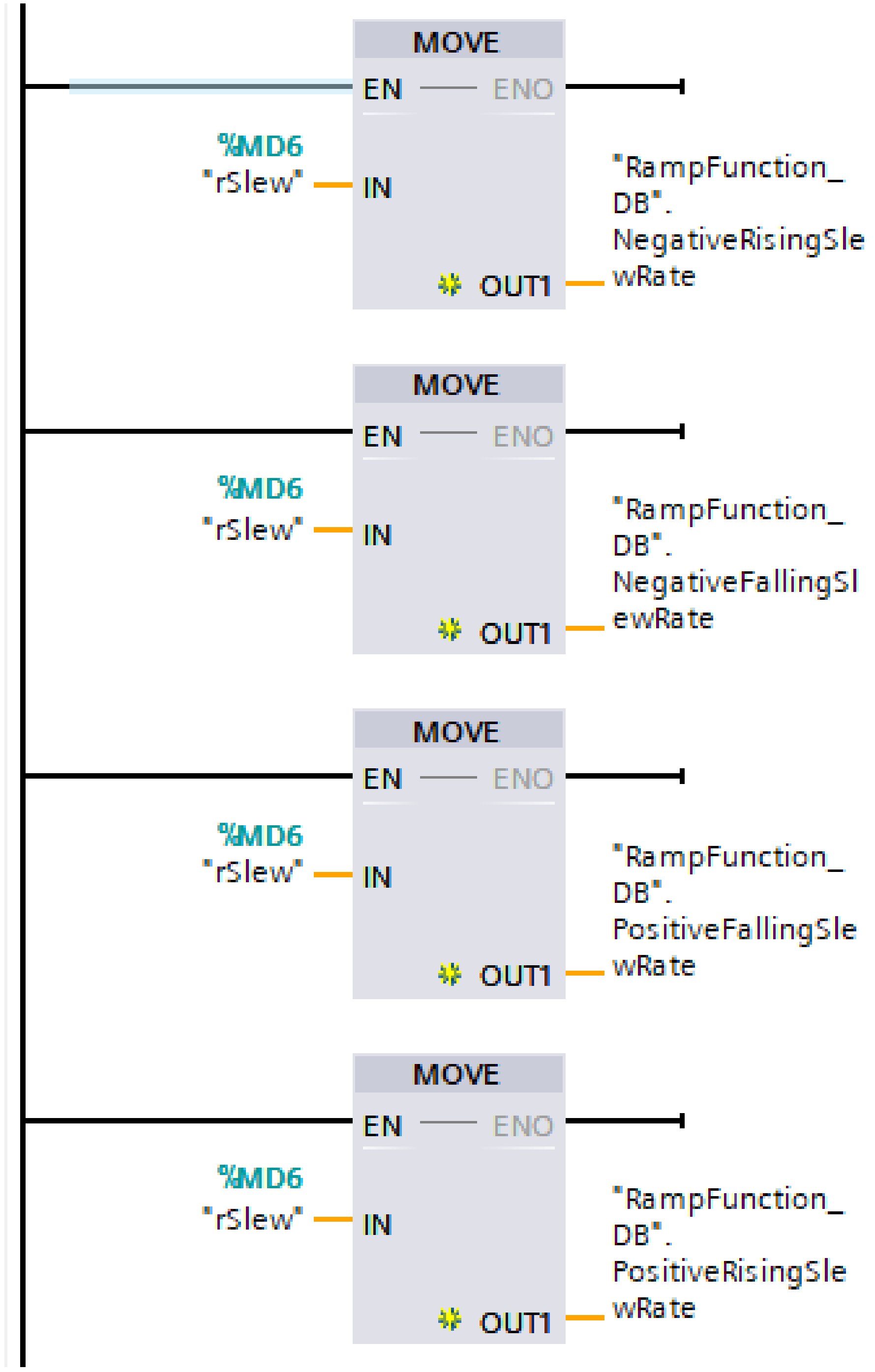
Slew < 0: Input is less than output, forcing output down to match the input

Slew > 0: Input is greater than output, forcing output up to match the input



Make slew positive value

Make slew to be real value



To set the SlewRate of RampFunction, a positive real value is required.

Simulation

SIM table_1

Name	Address	Display format	Monitor/Modify value	Bits
"Input":P	%IW0:P	DEC+/-	0	
"Output"	%QW0	DEC+/-	0	
"slew"	%MW0	DEC+/-	0	
"Reset":P	%I10.0:P	Bool	FALSE	

Set “Input”:P to 100

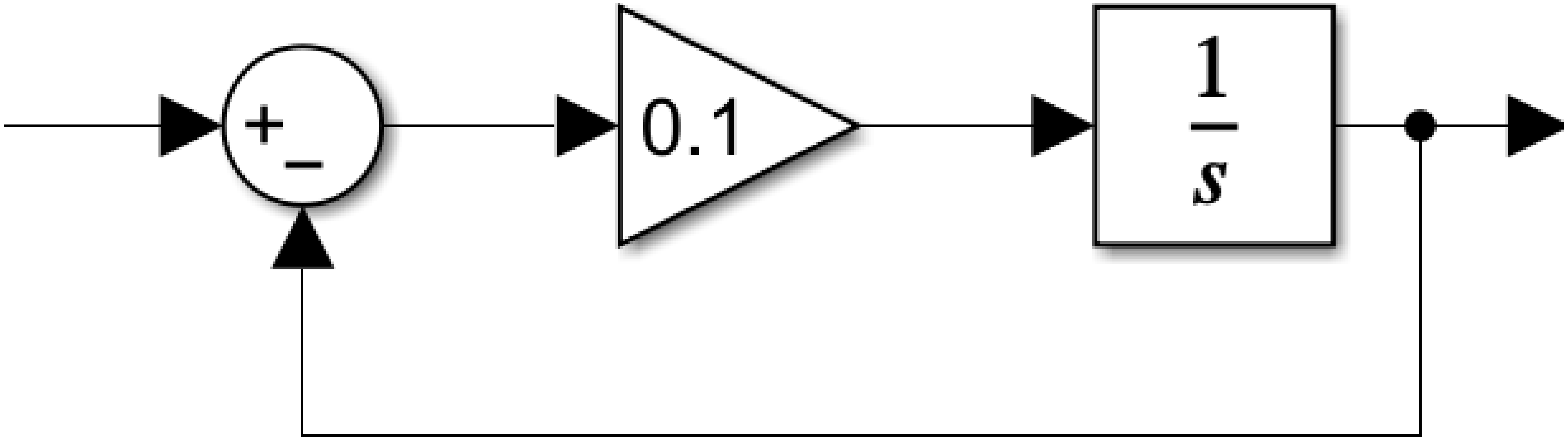
SIM table_1

Name	Address	Display format	Monitor/Modify value	Bits
"Input":P	%IW0:P	DEC+/-	100	
"Output"	%QW0	DEC+/-	100	
"slew"	%MW0	DEC+/-	0	
"Reset":P	%I10.0:P	Bool	FALSE	

Output should reach 100 in 4 seconds

Exercise

- Config and simulate a feedback control system of the following figure in PLC.



Lab 06: Emulation of analog output

Software and communications

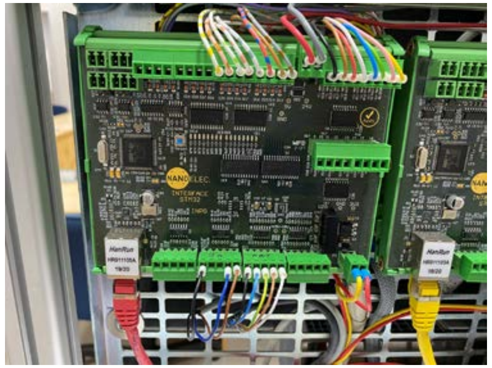
- MathWorks Simulink
  - Request, read, and write to GICS
- Simulink Desktop Real-Time
  - Synchronize Simulink to real-time processing and communication
- Simscape
  - Simulate physical systems (optional)
- Communication
  - UDP
- PLC ←→ GICS ← → Simulink
  - GICS is the middleman to link PLC and Simulink
  - Simulink emulates practical process
  - PLC controls the process

Emulation concept

PLC

Direct connected
I/O
Analog voltage

GICS card



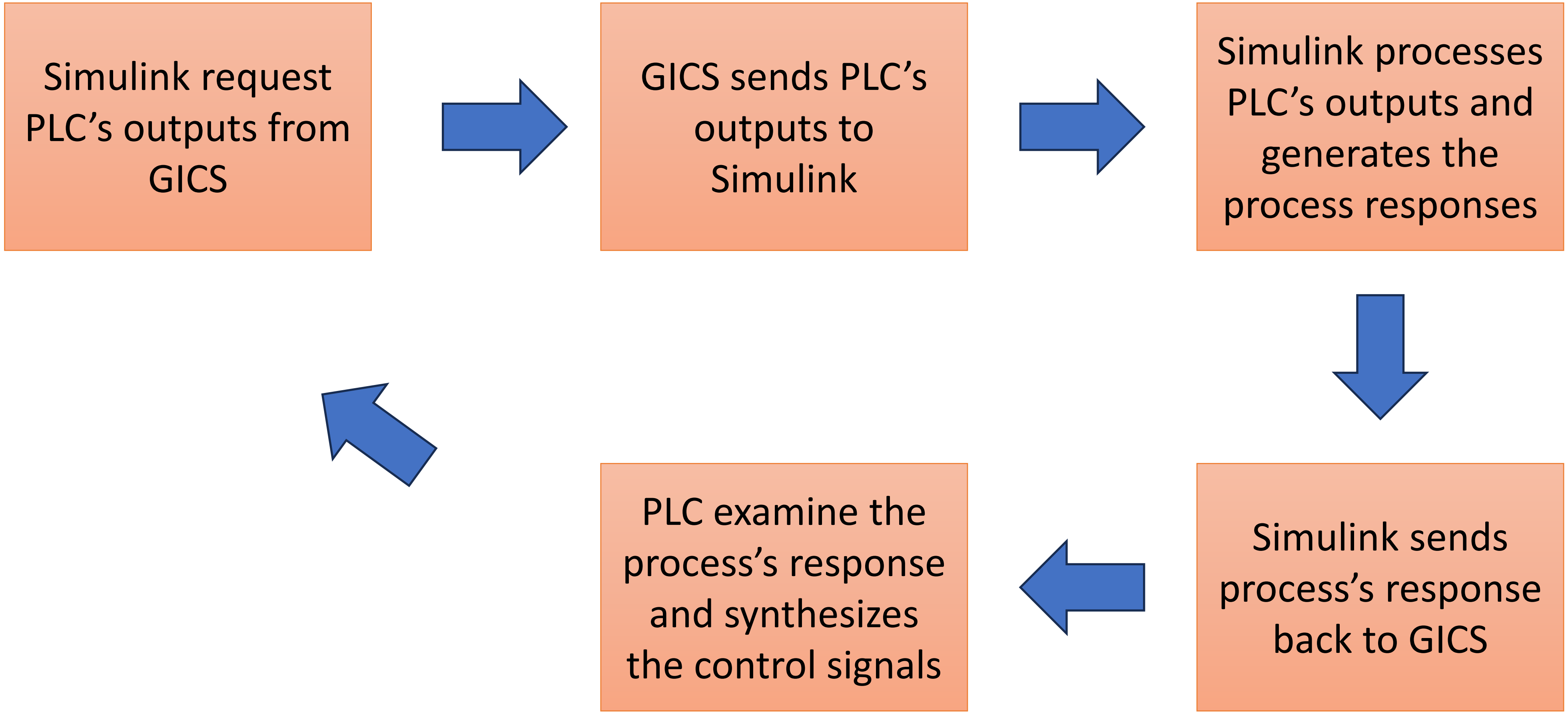
UDP packet
binary

MathWorks Simulink

Controller

Virtual
Process or
Plant

Communication loop



Scenario

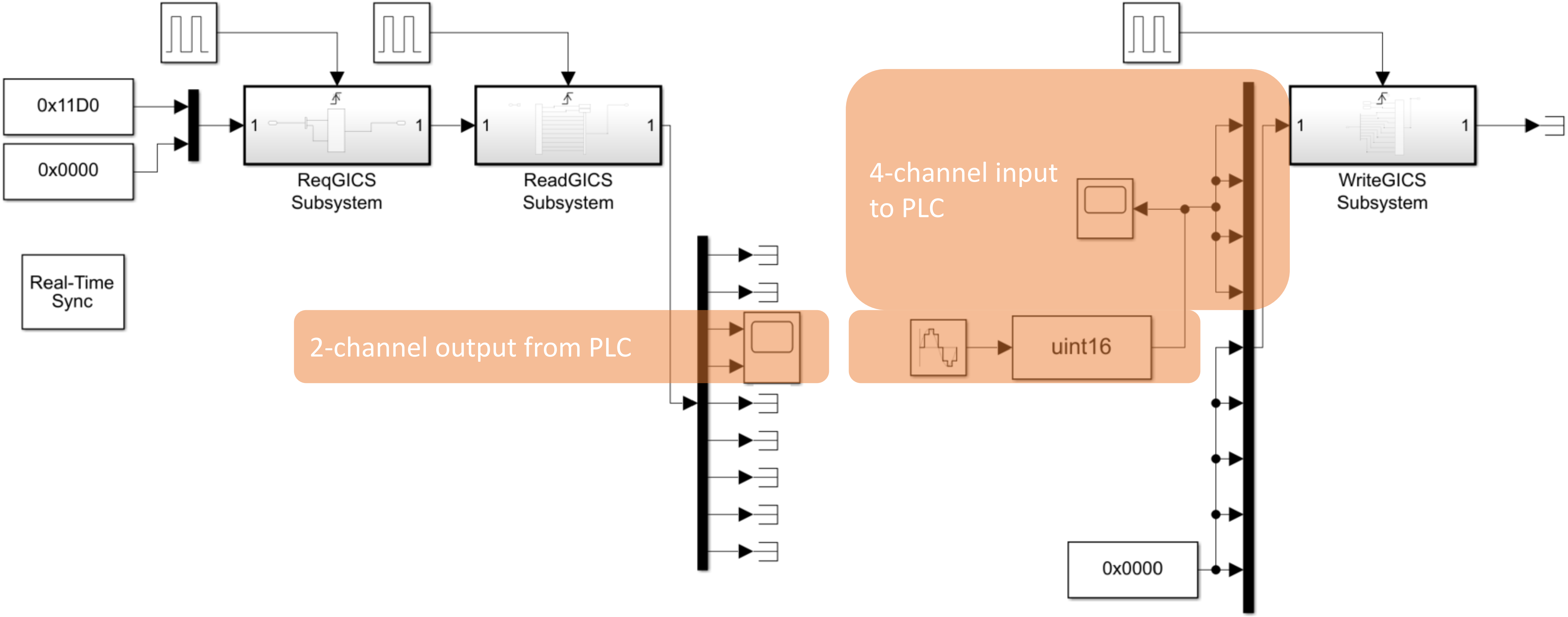
PLC's 2-channel output to GICS

- Read by Simulink
- From PLC: -27000 to 27000
  - Sawtooth wave
  - 16-bit signed integer
- GICS's record: 0 – 35xx
  - 12-bit unsigned integer
- Simulink's read: 0 – 35xx
  - 16-bit unsigned integer

Simulink's 4-channel output to GICS

- Read by PLC
- From Simulink: 50 – 950
  - Sinewave
  - 16-bit unsigned integer
- GICS's record: 50 - 950
  - 10-bit unsigned integer
- PLC's read: -25000 ± Δ to 25000 ± Δ
  - 16-bit signed integer

Simulink’s diagram of “GICS_rw.slx”

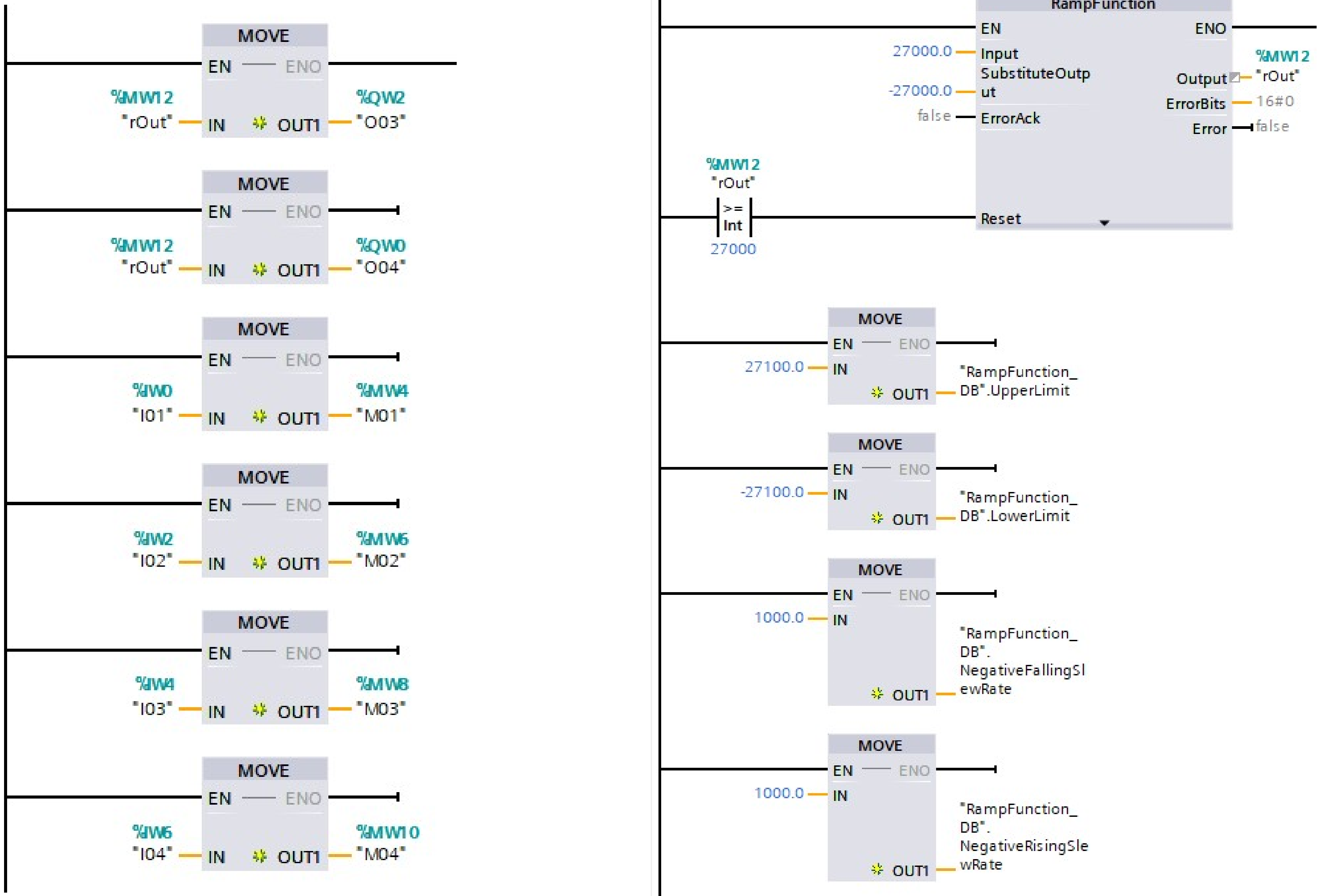


PLC’s tag

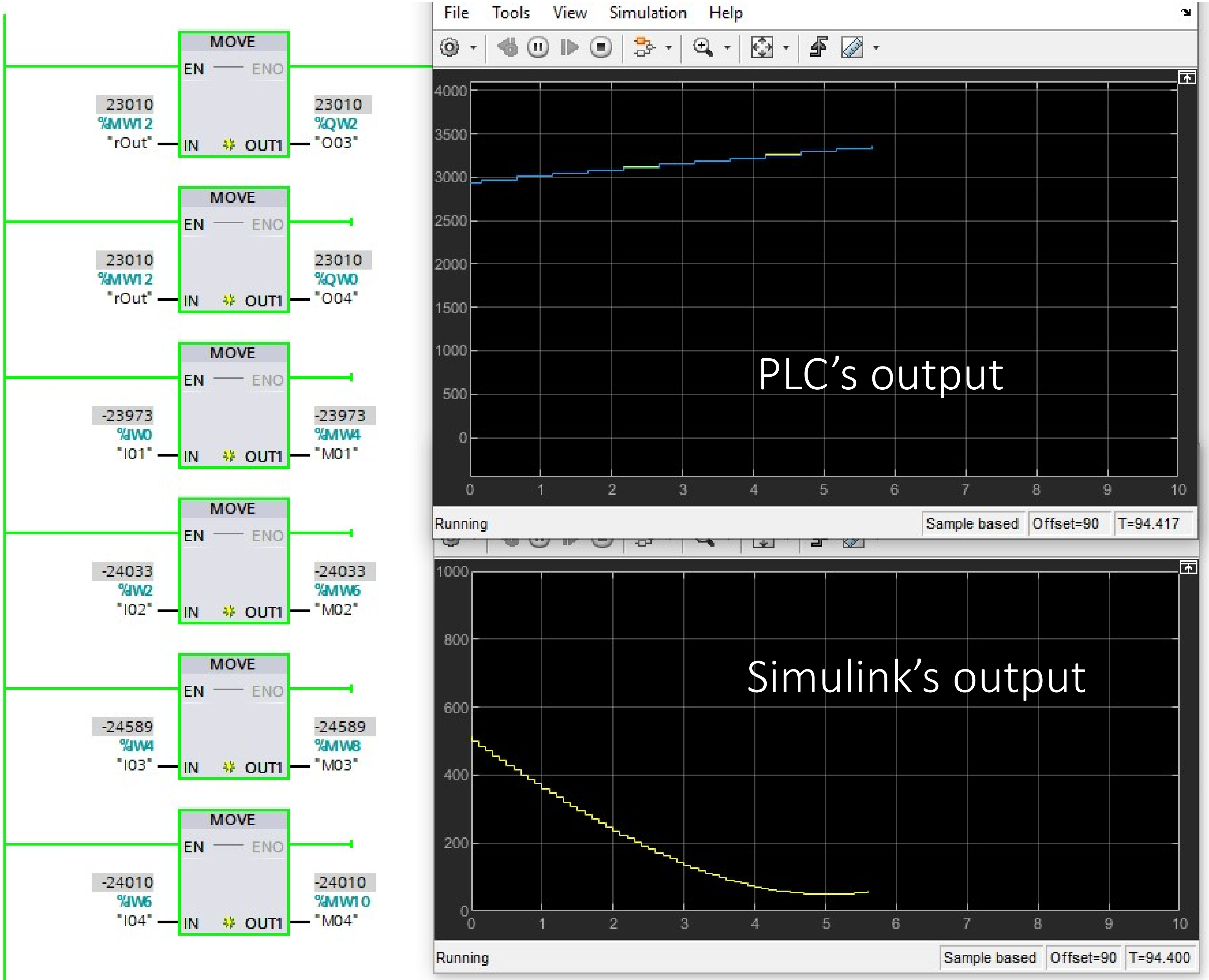
Default tag table

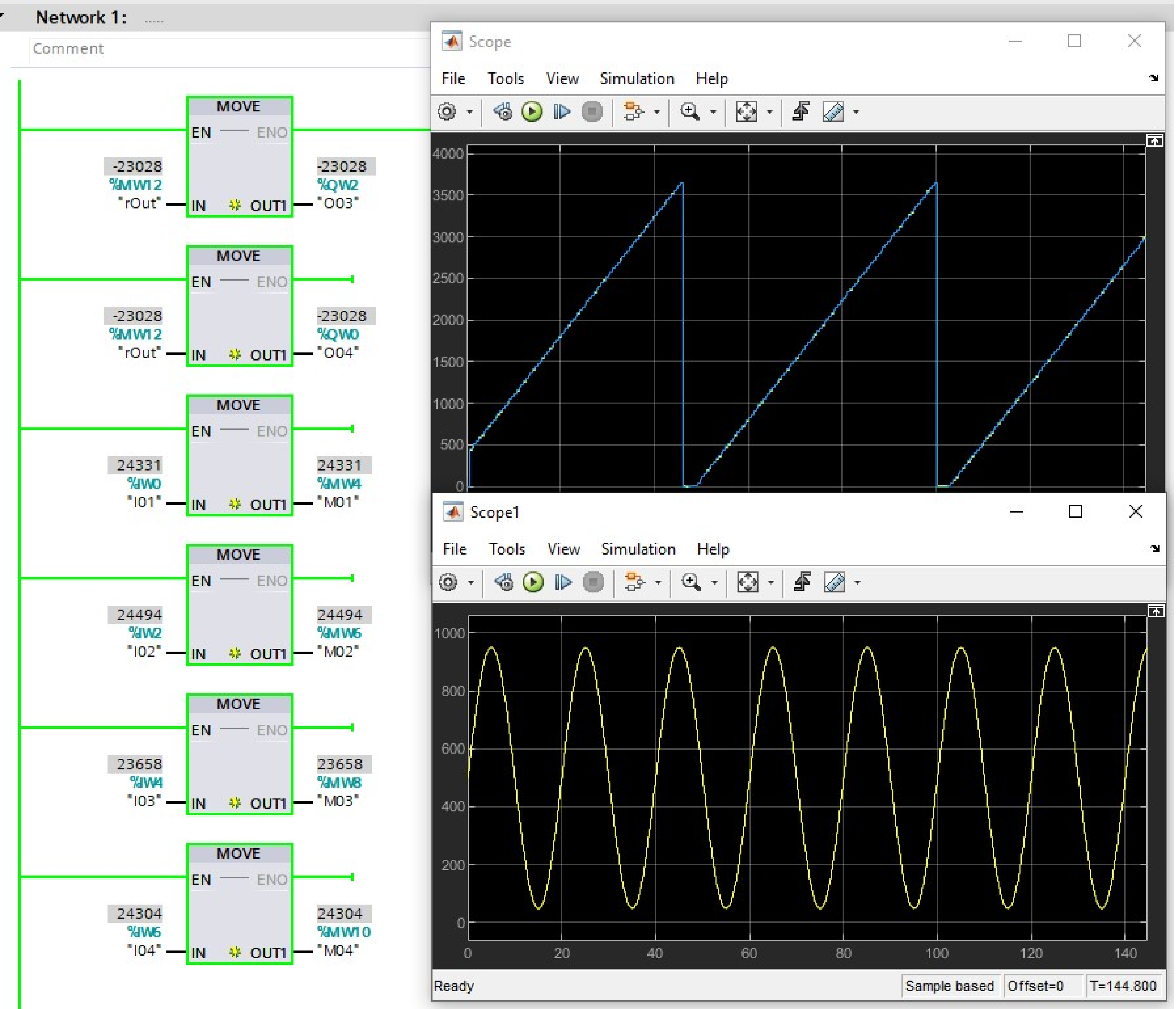
		Name	Data type	Address
1		I01	Int	%IW0
2		I02	Int	%IW2
3		I03	Int	%IW4
4		I04	Int	%IW6
5		O04	Int	%QW0
6		O03	Int	%QW2
7		M01	Int	%MW4
8		M02	Int	%MW6
9		M03	Int	%MW8
10		M04	Int	%MW10
11		rOut	Int	%MW12

Ladder diagram



Real-Time synchronizing between PLC and Simulink



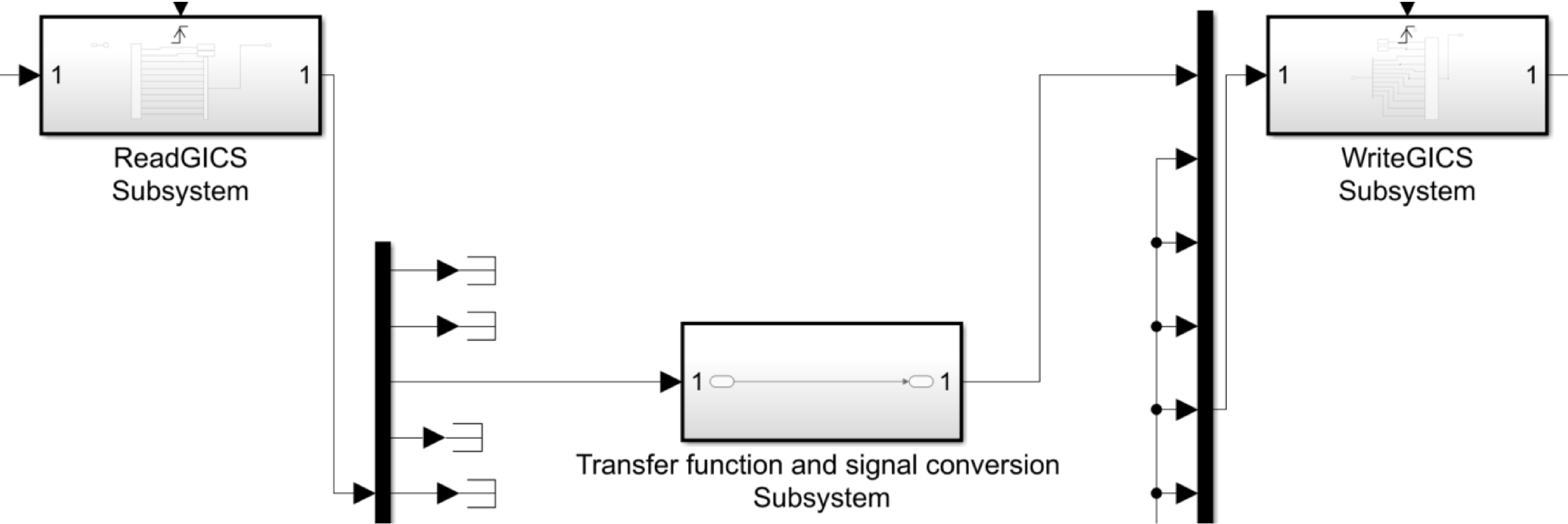


-Signal's condition at the end of Simulink's real-time session

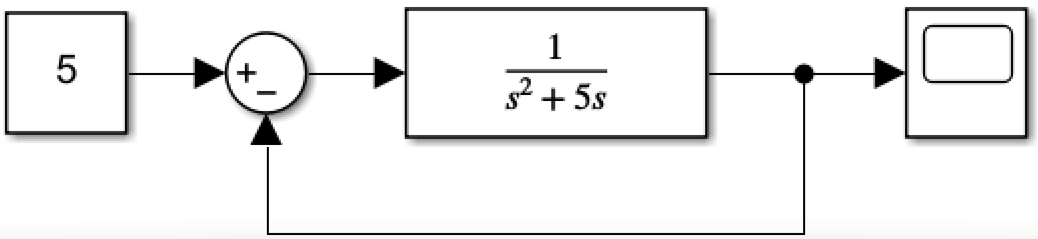
-PLC is still working

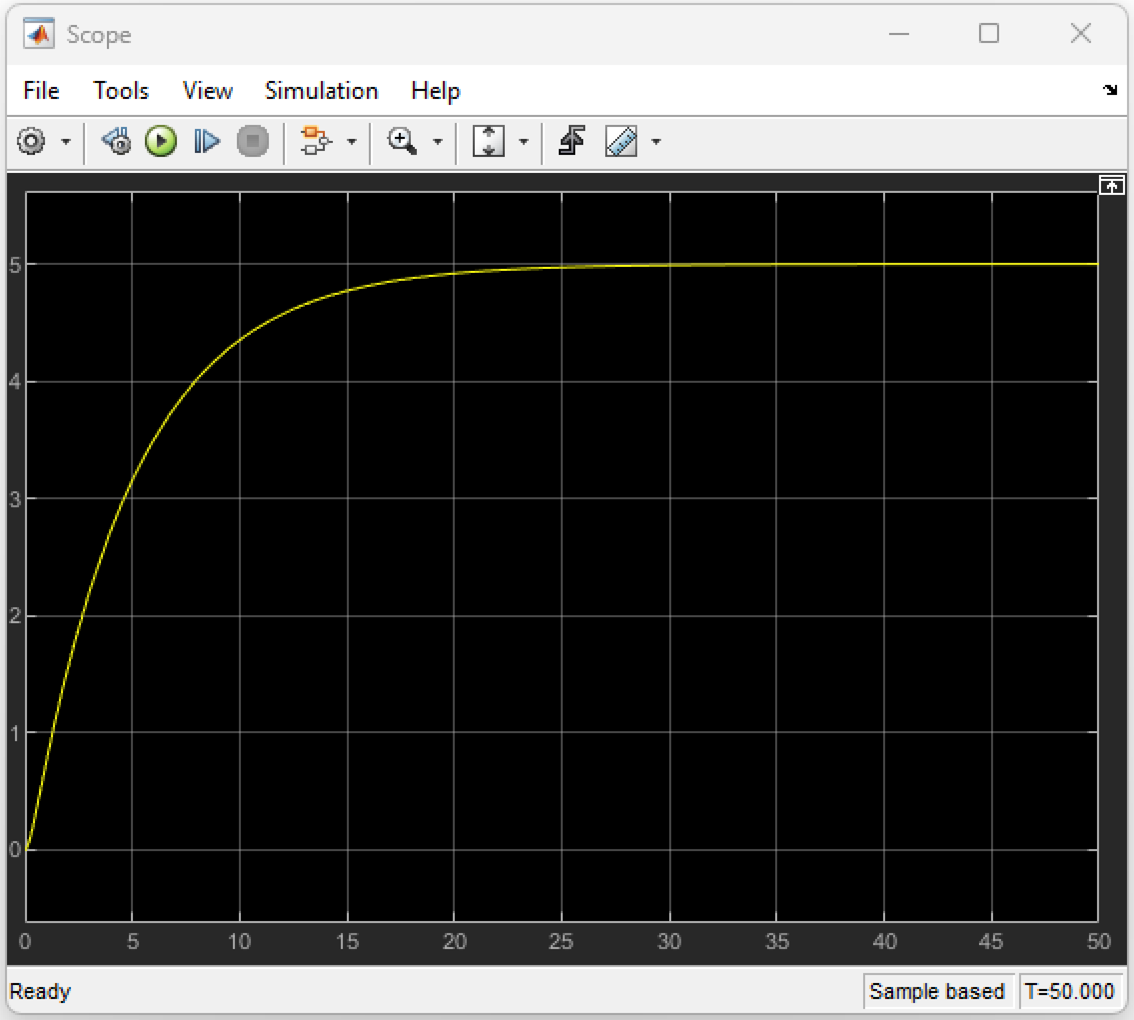
Exercise

- Put a plant process between PLC's output and Simulink's output
  - Noted that the sampling time is 0.5 second
- Any 1st-order analog transfer function: one input, one output
  - -9V ≤ Input/Output ≤ 9V
- Put the appropriate number conversion to I/O
- Implement as a subsystem as shown in the below figure
- The process should be performed well by 0.5s sampling time



Lab 07: Emulation of feedback control





Emulation of a simple feedback loop

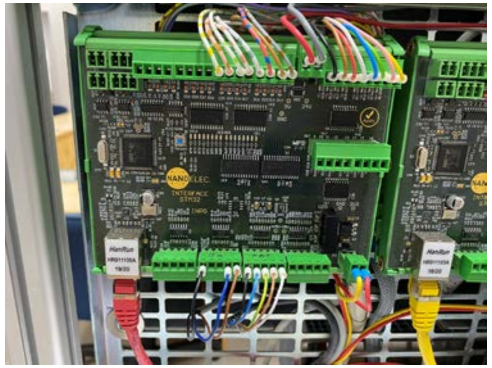
- Controller is a PLC
- the process $G(s) = \frac{1}{s^2+5s}$ is implemented in Simulink as a virtual process
- Step input of 5V is set in PLC
- Control signal (input - output) is provided by PLC
- Output is provided by Simulink

Configuration

PLC

Direct connected
I/O
Analog voltage

GICS card



UDP packet
binary

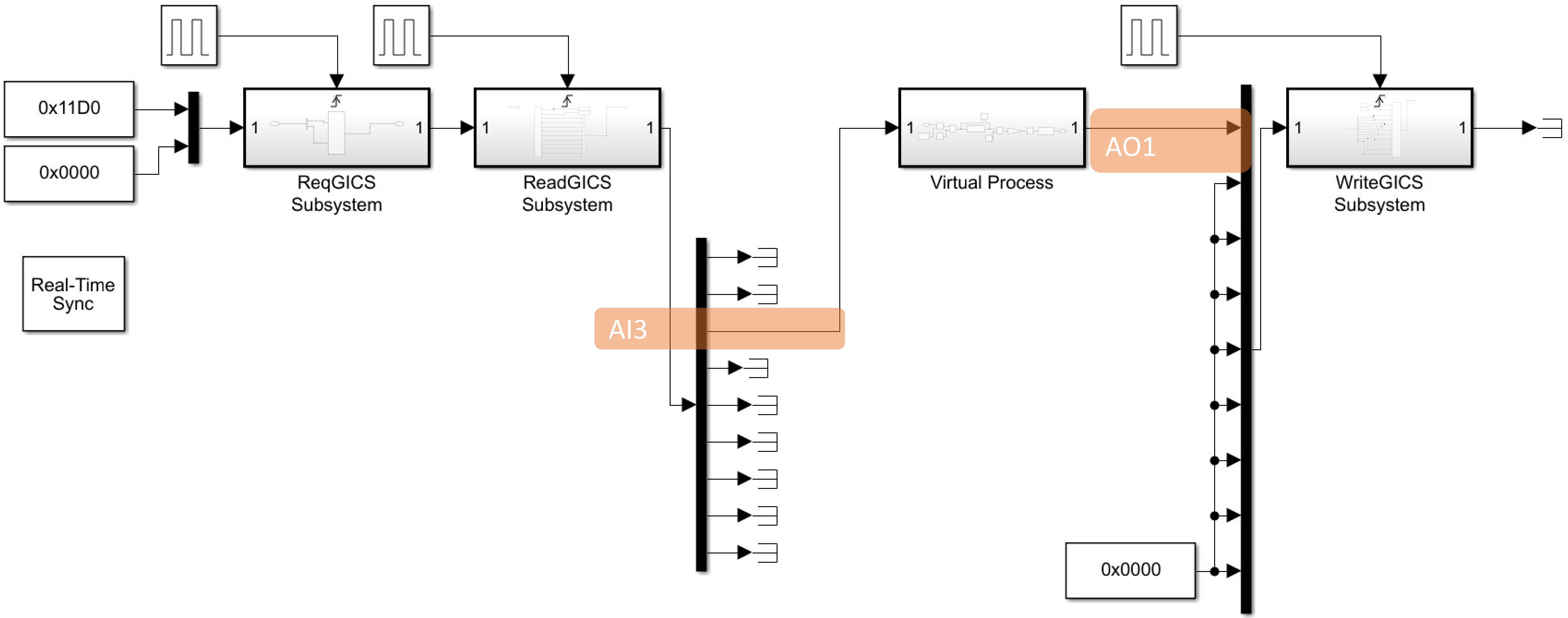
MathWorks Simulink

-Read process output
from %IW0
-Send control signal via
%QW2

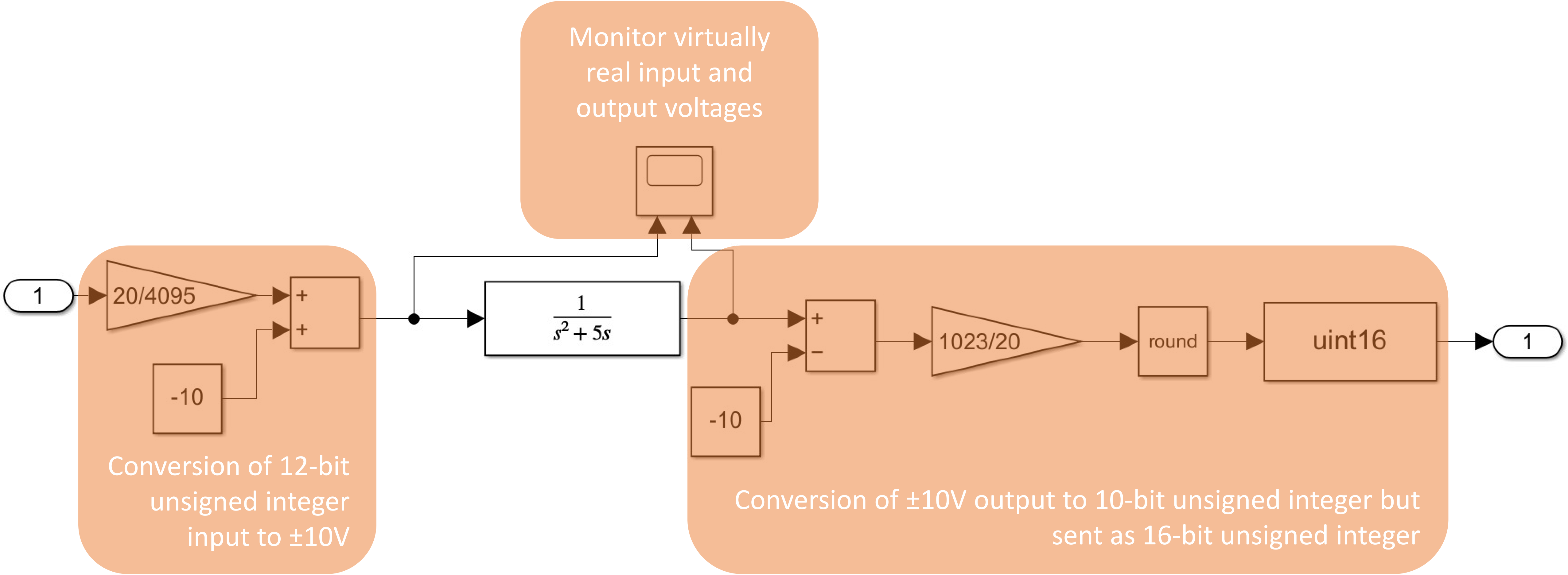
-Read control signal
from PLC directly to AI3
-Read process output
from Simulink via UDP
packet and keep in AO1

-Read control signal
from GICS's AI3
-Send UDP packet to
update GICS's AO1

Simulink Framework



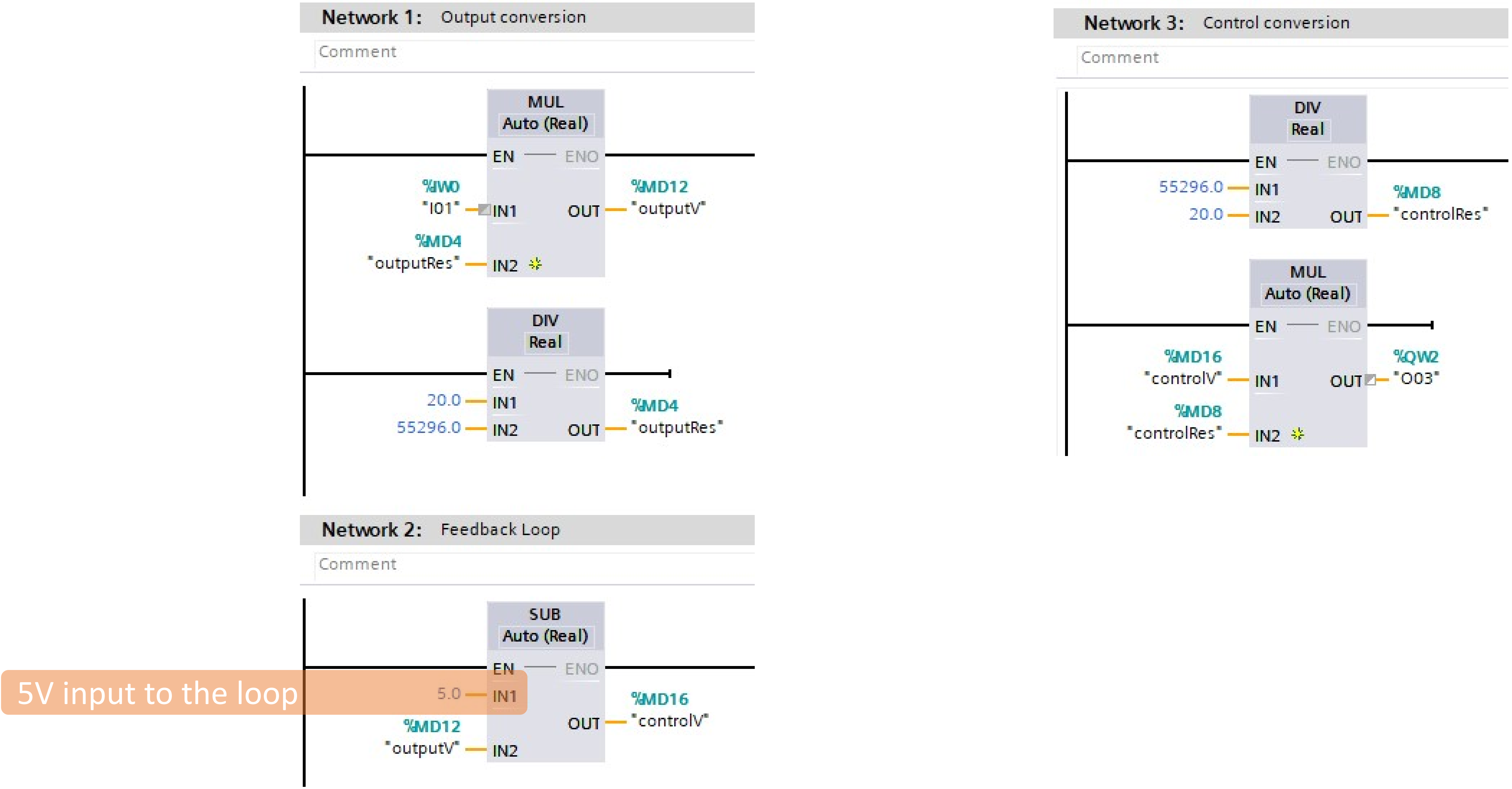
Virtual Process



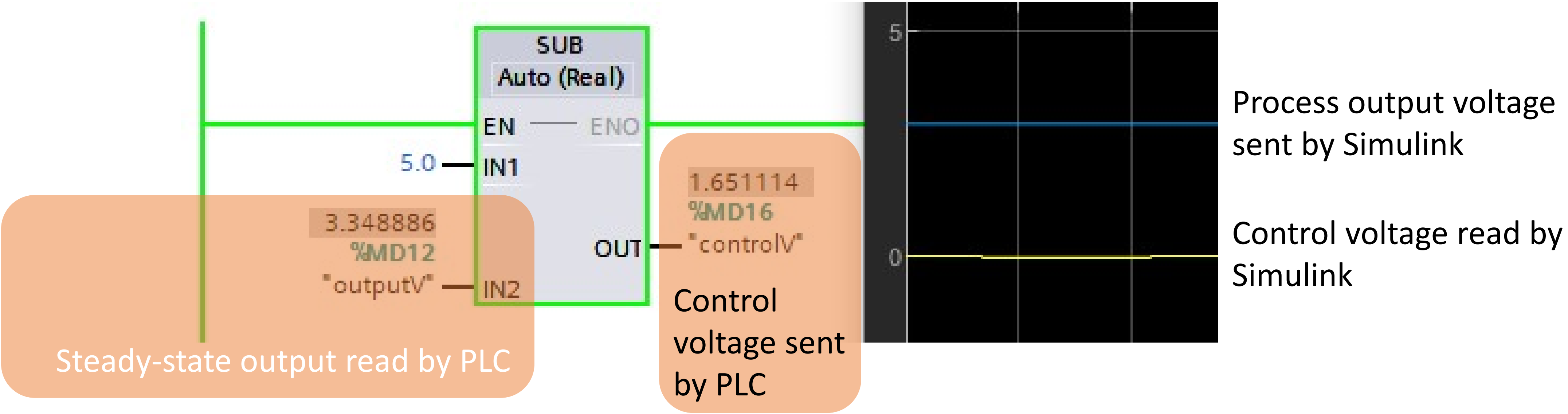
PLC tags

		Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Supervis...	Comment
1		I01	Int	%IW0	☐	☑	☑	☑		Output from the virtual process in Simulink
2		O03	Int	%QW2	☐	☑	☑	☑		Control signal as the output of PLC
3		outputRes	Real	%MD4	☐	☑	☑	☑		Resolution of Simulink's output (V per integer)
4		controlRes	Real	%MD8	☐	☑	☑	☑		Resoluton of PLC control signal (interger per V)
5		outputV	Real	%MD12	☐	☑	☑	☑		Output voltage recognized by PLC
6		controlV	Real	%MD16	☐	☑	☑	☑		Intended control voltage from PLC

Ladder diagram



Emulation through GICS with 0.5s sampling period



The steady-state output is capped at around 3.35V because the control voltage sensed by Simulink is around 0. However, the PLC is meant to send 1.65V control signal.

Therefore, the offset voltage of the control voltage is around -1.65V.

Vcs → Control voltage read by Simulink, Vcp → Control voltage sent by PLC

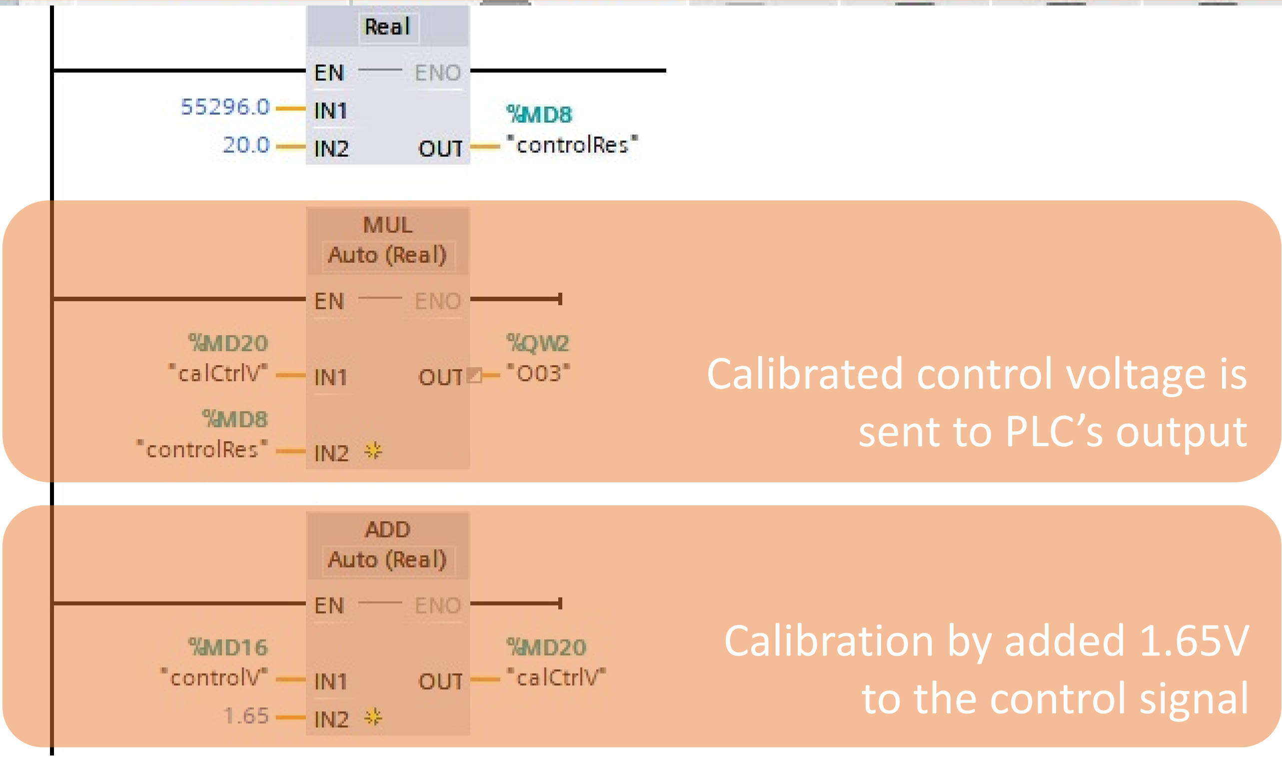
$$V_{cs} = V_{cp} - 1.65$$

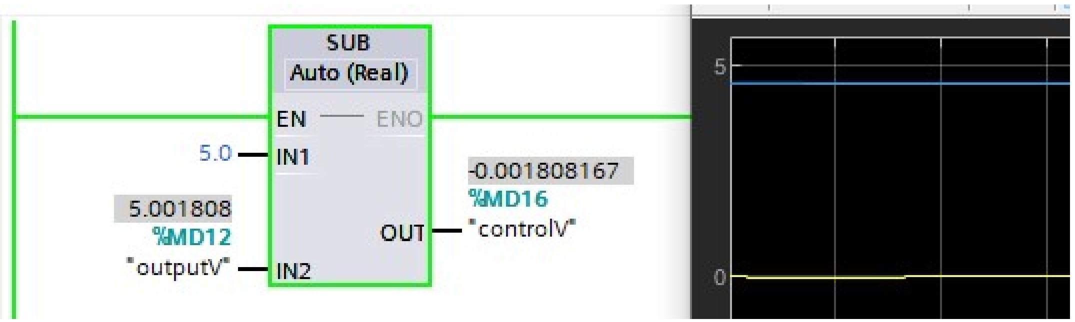
Calibration of the control signal from PLC

Default tag table

		Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Supervis...	Comment
1		I01	Int	%IW0	☐	☑	☑	☑		Output from the virtual process in Simulink
2		O03	Int	%QW2	☐	☑	☑	☑		Control signal as the output of PLC
3		outputRes	Real	%MD4	☐	☑	☑	☑		Resolution of Simulink's output (V per integer)
4		controlRes	Real	%MD8	☐	☑	☑	☑		Resoluton of PLC control signal (interger per V)
5		outputV	Real	%MD12	☐	☑	☑	☑		Output voltage recognized by PLC
6		controlV	Real	%MD16	☐	☑	☑	☑		Intended control voltage from PLC
7		calCtrlV	Real	%MD20	☐	☑	☑	☑		Calibrated control voltage from PLC

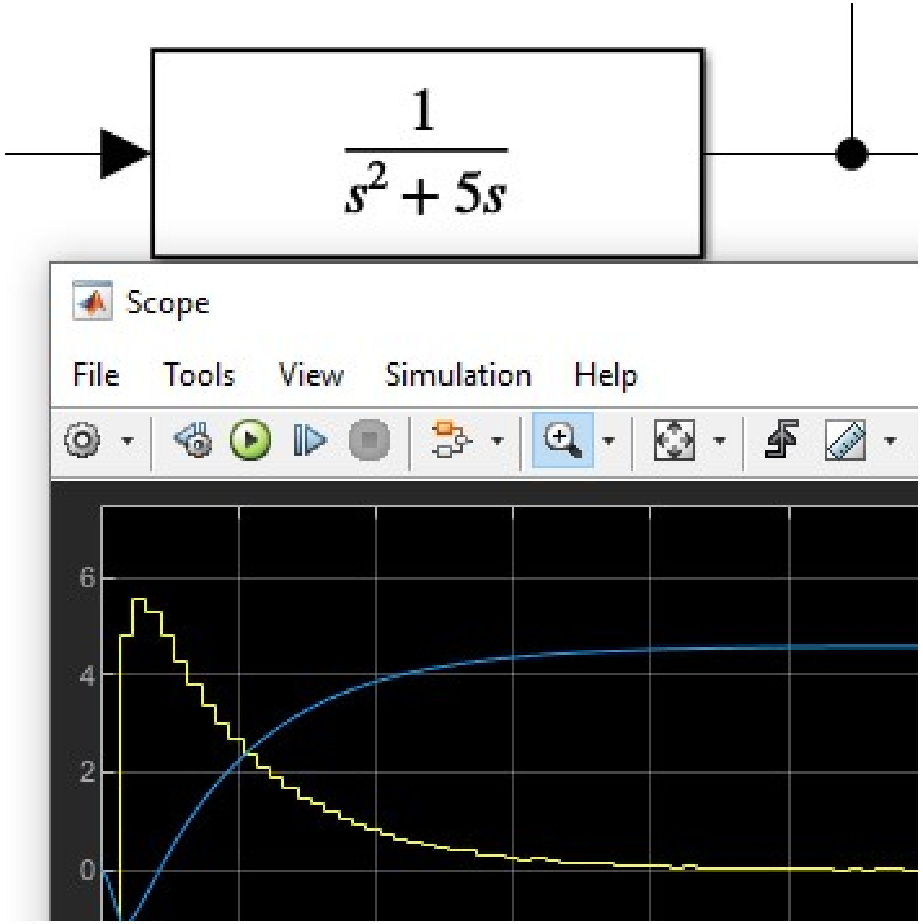
Added variable





In steady-state, the control voltage in PLC and Simulink are almost identical at zero.
However, the process output read by PLC is a little bit higher than the process output sent by Simulink
Therefore, there is also an offset in GICS's output to PLC.

Control and output voltages of the virtual process



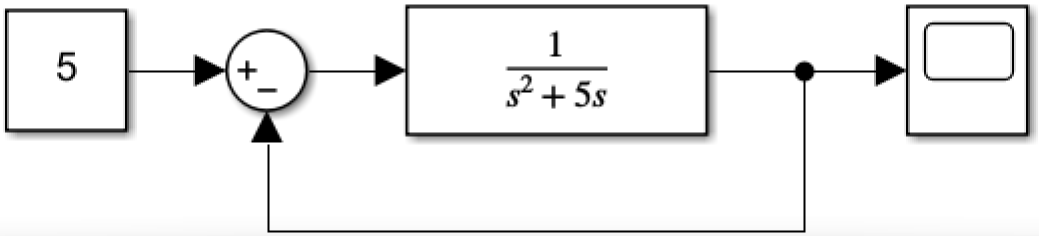
-The reference must be measured from the virtual process not the PLC I/O.
-It is where the real work is virtually done.
-It is emulating a physical hardware and sensor.
-In Simulink's simulation, we can assume that sensor is not wrong. Therefore, calibration must be done in PLC coding.
-However, in real practical equipment, sensor should be test for its validity.

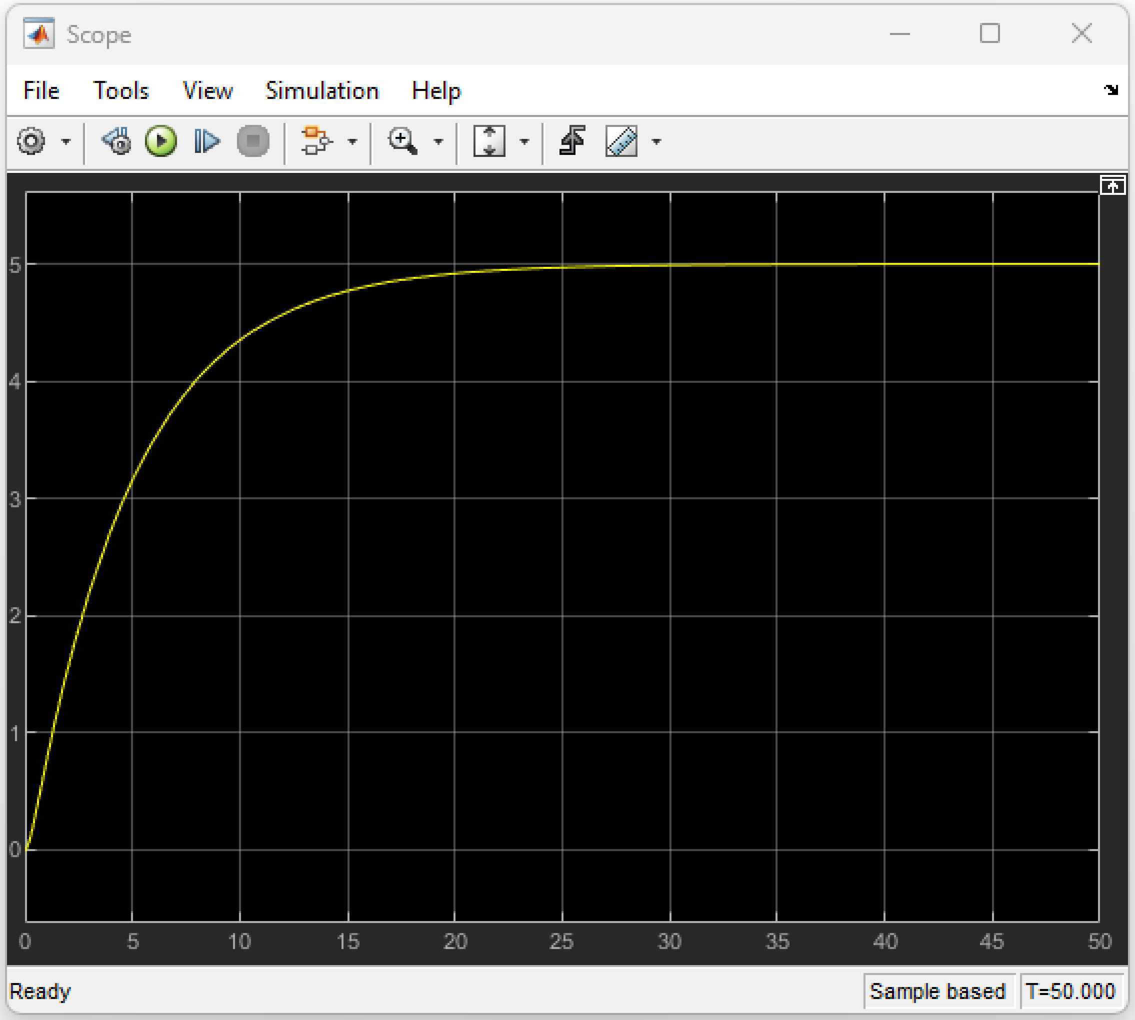
Process output voltage sent by Simulink

Control voltage read by Simulink

Exercise

- Calibrate the PLC reading from Simulink virtual process output
- Make the emulation response closed to the theory shown in the right figure.
- Config the HMI to show the target response, the process output, and the control signal.

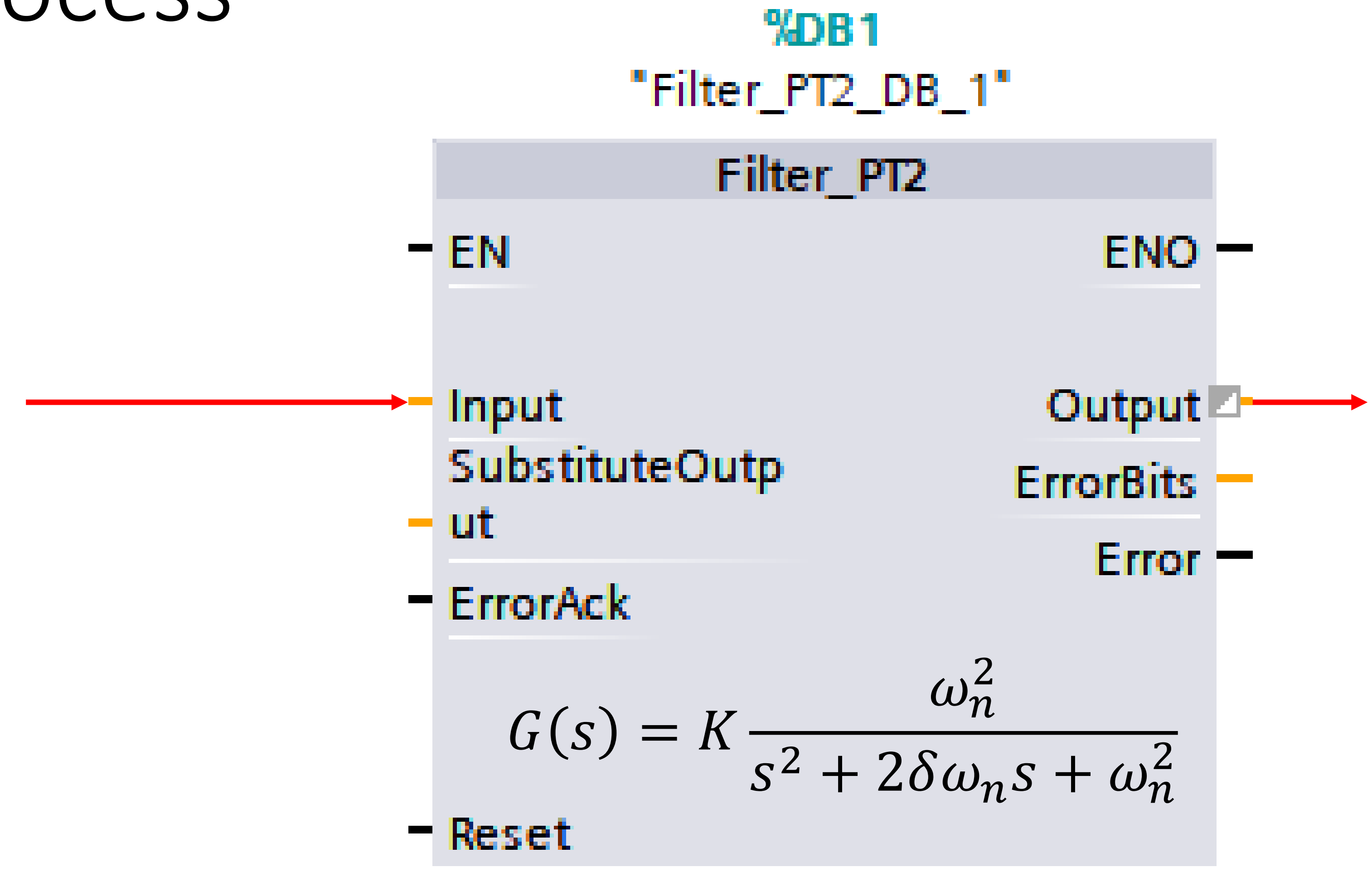




Lab 08: PLC controllers

Second-order process

- Simulation of all-pole second-order response via Filter_PT2 block
- You can specify the following filter parameters:
  - Proportional gain (K)
  - Time constant (τ)
  - Damping (ζ)
- ω = 1/τ
- τ = 1/ω

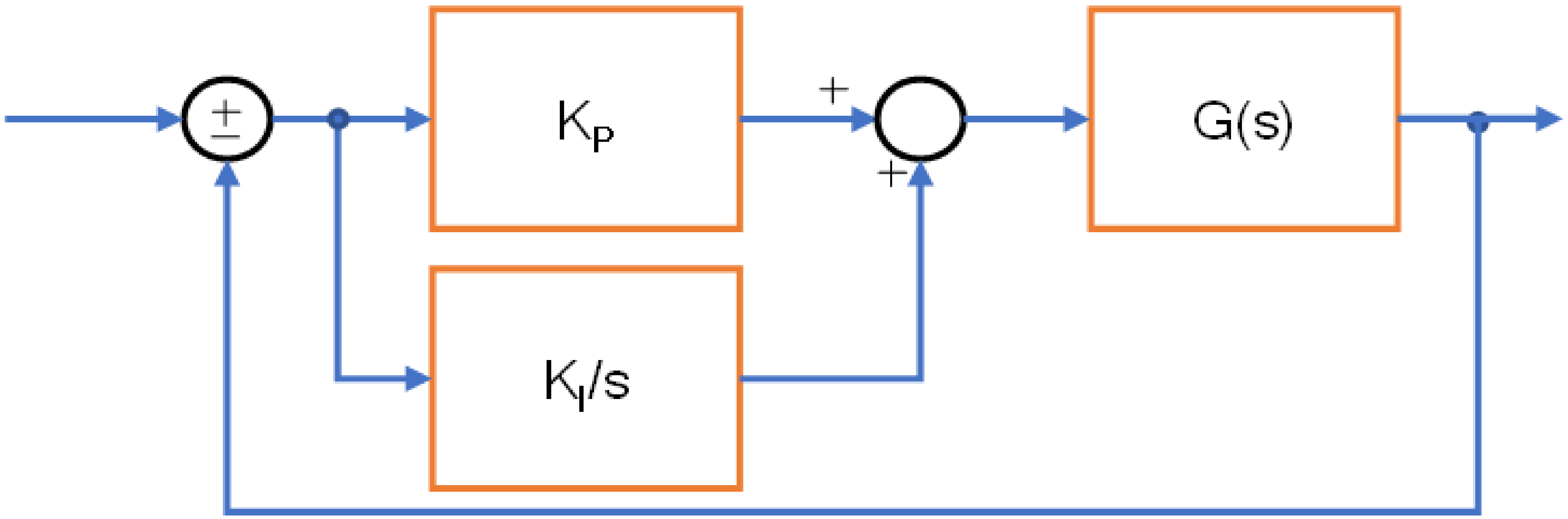


$$G(s) = \frac{\text{Output}(s)}{\text{Input}(s)} = \frac{\text{Gain}}{1 + 2 \cdot \text{Damping} \cdot \text{TimeConstant} \cdot s + \text{TimeConstant}^2 \cdot s^2}$$

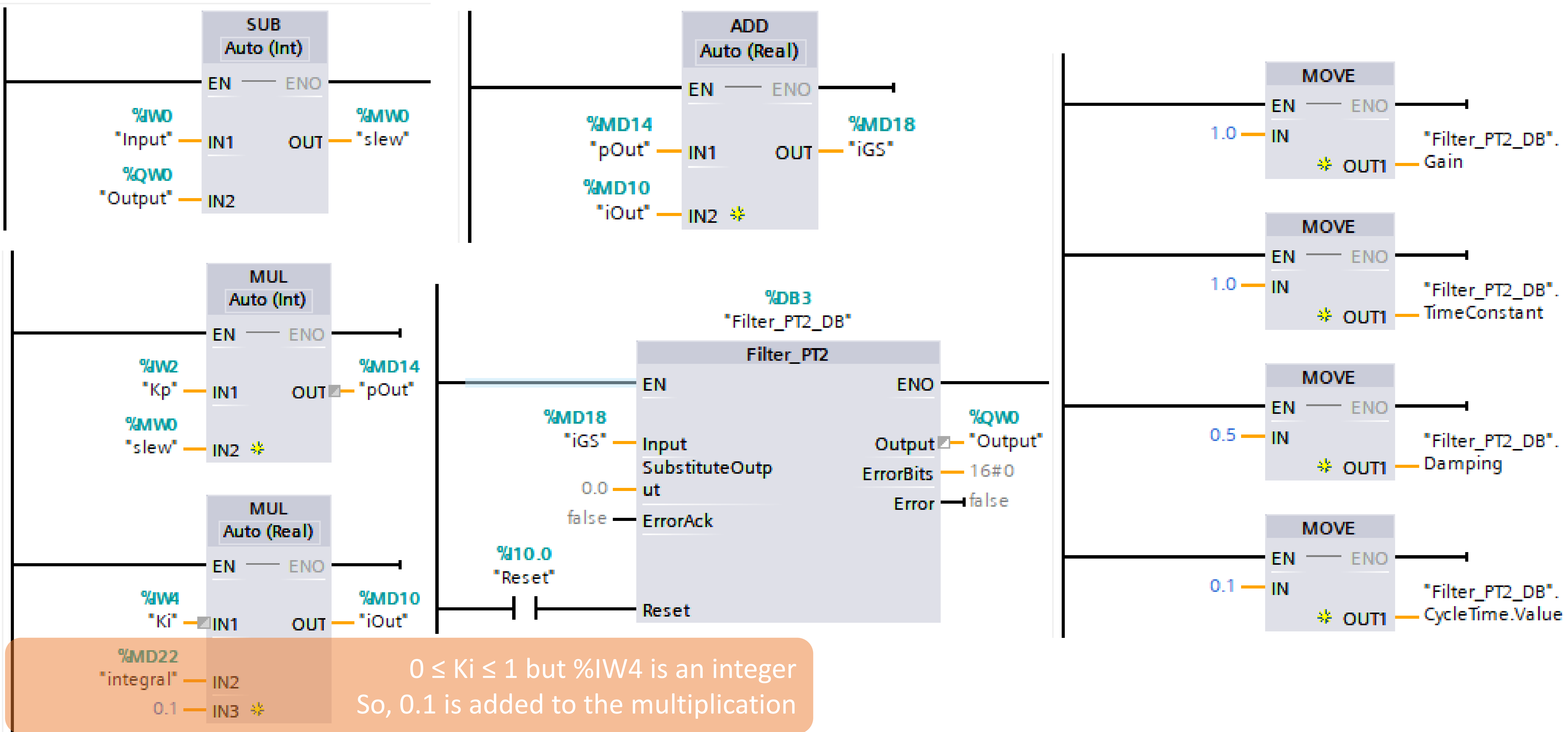
Tag table

slew	Int	%MW0	Input - Output
aSlew	Int	%MW2	Absolute value of slew
force	Int	%MW4	Force the RampFunction to go up or down
Input	Int	%IW0	Desired Response
Output	Int	%QW0	Actual Response
Reset	Bool	%I10.0	Reduce Output to 0
rSlew	Real	%MD6	Real absolute value of slew for setting RampFunction
Kp	Int	%IW2	Propotional Gain
Ki	Int	%IW4	Integral Gain
iOut	Real	%MD10	Integral Output
pOut	Real	%MD14	Propotional Output
iGS	Real	%MD18	Input to the G(s)
integral	Real	%MD22	Output of RampFunction as an Integrator

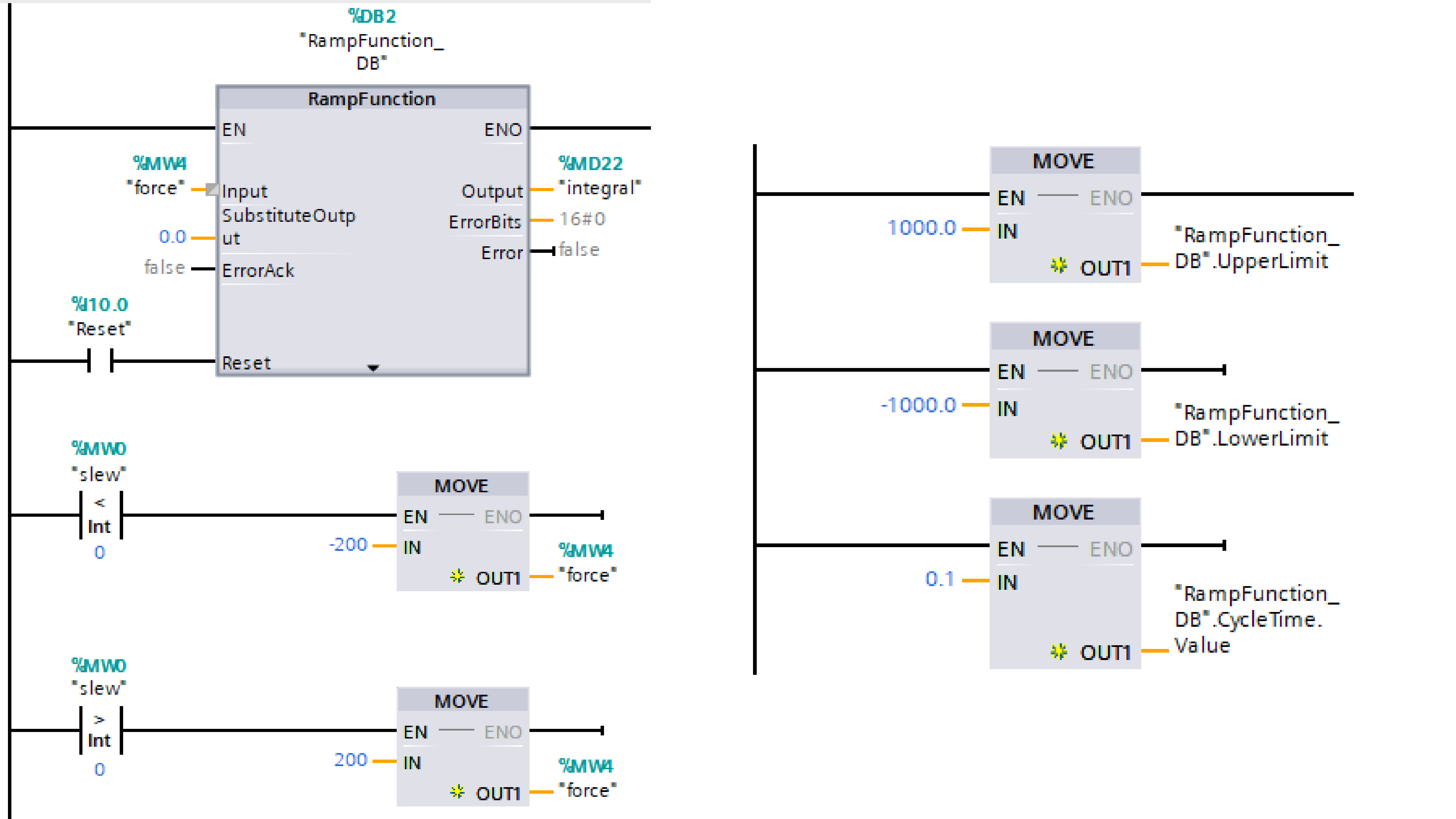
Proportional-Integral (PI) Controller



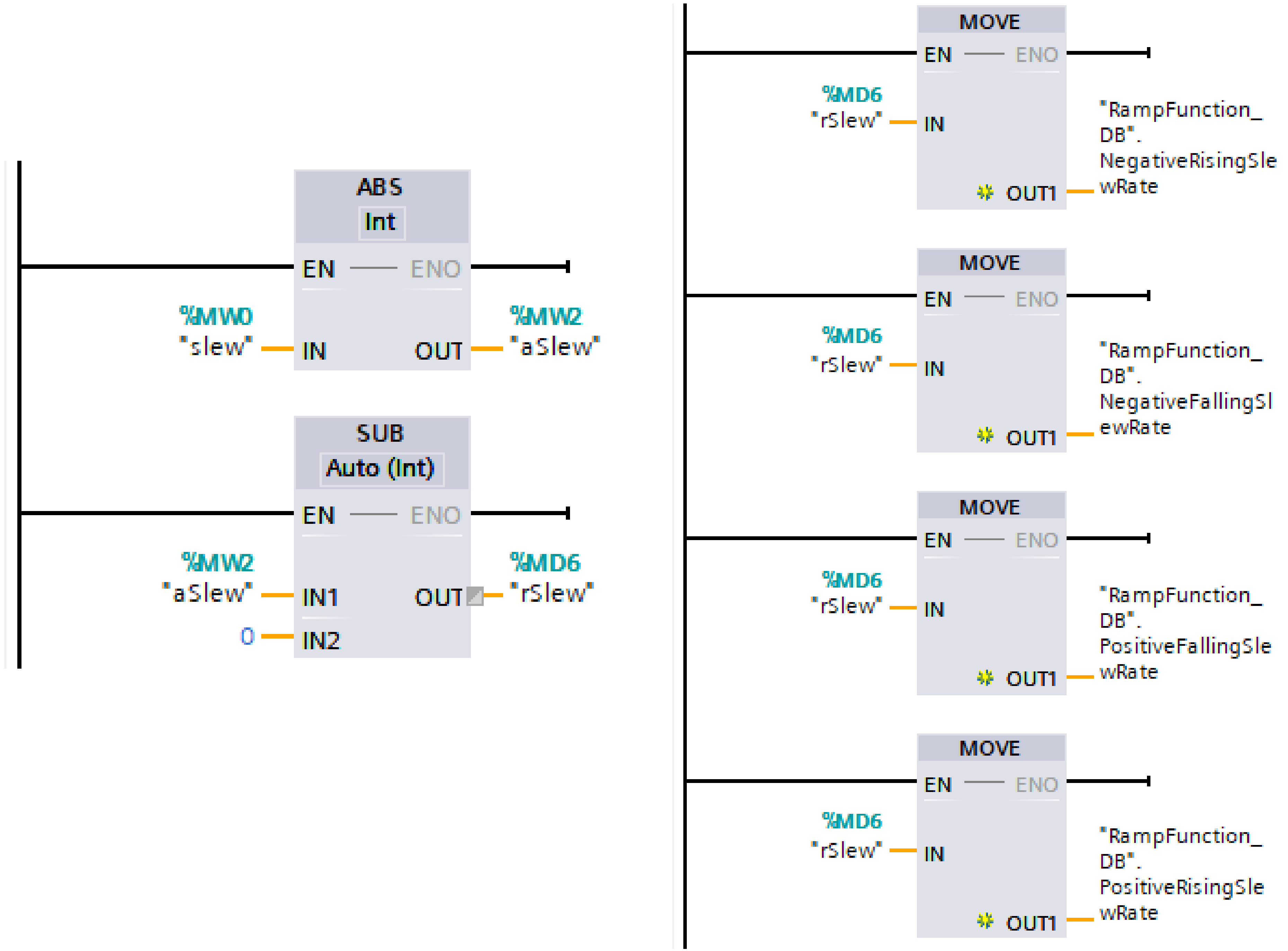
PI controller implementation



0 ≤ Ki ≤ 1 but %IW4 is an integer
So, 0.1 is added to the multiplication



%DB2
"RampFunction_DB"
RampFunction
EN
ENO
%MW4
"force"
Input
SubstituteOutput
0.0
false
ErrorAck
Output
ErrorBits
Error
%MD22
"integral"
16#0
false
%I10.0
"Reset"
Reset
%MW0
"slew"
<
Int
0
MOVE
EN
ENO
-200
IN
OUT1
%MW4
"force"
%MW0
"slew"
>
Int
0
MOVE
EN
ENO
200
IN
OUT1
%MW4
"force"
MOVE
EN
ENO
1000.0
IN
OUT1
"RampFunction_DB".UpperLimit
MOVE
EN
ENO
-1000.0
IN
OUT1
"RampFunction_DB".LowerLimit
MOVE
EN
ENO
0.1
IN
OUT1
"RampFunction_DB".CycleTime.Value


ABS
Int
EN
ENO
%MW0
"slew"
IN
OUT
%MW2
"aSlew"
SUB
Auto (Int)
EN
ENO
%MW2
"aSlew"
IN1
OUT
%MD6
"rSlew"
0
IN2
MOVE
EN
ENO
%MD6
"rSlew"
IN
OUT1
"RampFunction_DB".NegativeRisingSlewRate
MOVE
EN
ENO
%MD6
"rSlew"
IN
OUT1
"RampFunction_DB".NegativeFallingSlewRate
MOVE
EN
ENO
%MD6
"rSlew"
IN
OUT1
"RampFunction_DB".PositiveFallingSlewRate
MOVE
EN
ENO
%MD6
"rSlew"
IN
OUT1
"RampFunction_DB".PositiveRisingSlewRate

Simulation

SIM table_1

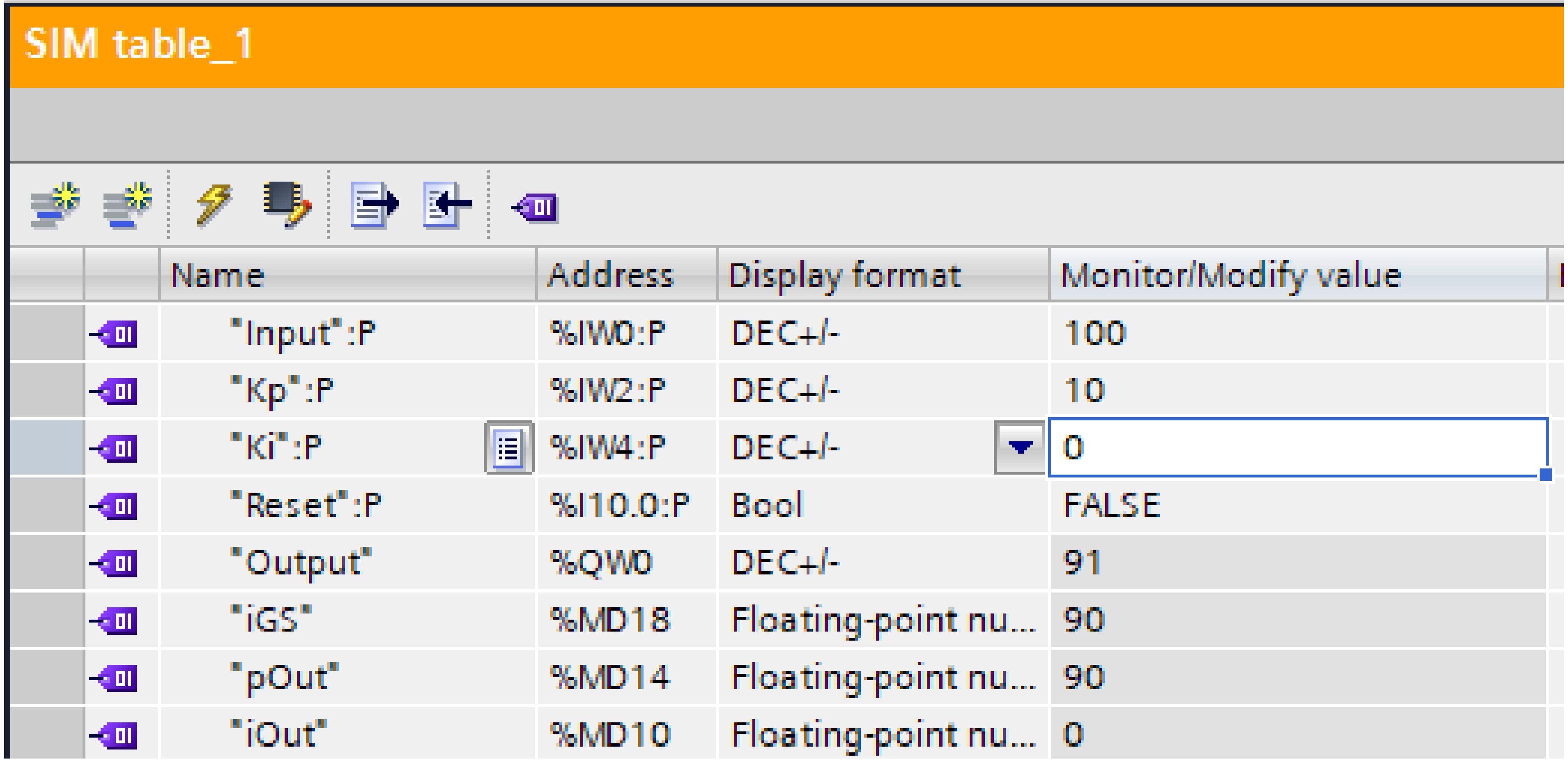
Name	Address	Display format	Monitor/Modify value
"Input":P	%IW0:P	DEC+/-	100
"Kp":P	%IW2:P	DEC+/-	0
"Ki":P	%IW4:P	DEC+/-	0
"Reset":P	%I10.0:P	Bool	FALSE
"Output"	%QW0	DEC+/-	0
"iGS"	%MD18	Floating-point nu...	0
"pOut"	%MD14	Floating-point nu...	0
"iOut"	%MD10	Floating-poin...	0

Set “Input”:P to 100

SIM table_1

Name	Address	Display format	Monitor/Modify value
"Input":P	%IW0:P	DEC+/-	100
"Kp":P	%IW2:P	DEC+/-	1
"Ki":P	%IW4:P	DEC+/-	0
"Reset":P	%I10.0:P	Bool	FALSE
"Output"	%QW0	DEC+/-	50
"iGS"	%MD18	Floating-point nu...	50
"pOut"	%MD14	Floating-point nu...	50
"iOut"	%MD10	Floating-point nu...	0

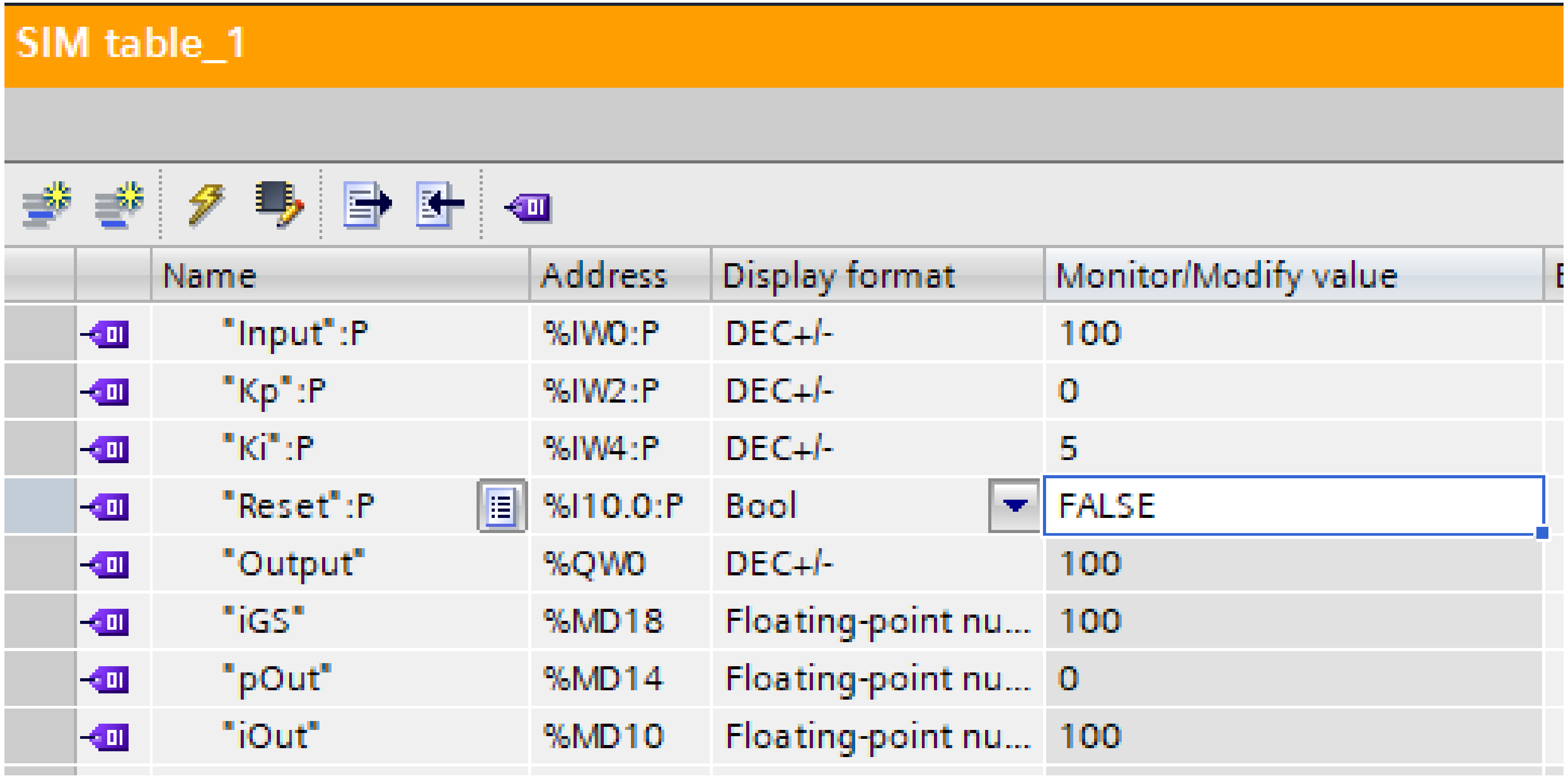
With Kp = 1, the final output is just half of the input as in theory.
In addition, the output is underdamped as the damping ratio is 0.5.



SIM table_1

	Name	Address	Display format	Monitor/Modify value
	"Input":P	%IW0:P	DEC+/-	100
	"Kp":P	%IW2:P	DEC+/-	10
	"Ki":P	%IW4:P	DEC+/-	0
	"Reset":P	%I10.0:P	Bool	FALSE
	"Output"	%QW0	DEC+/-	91
	"iGS"	%MD18	Floating-point nu...	90
	"pOut"	%MD14	Floating-point nu...	90
	"iOut"	%MD10	Floating-point nu...	0

Increasing Kp will increase the output but it will never reach the goal no matter how much of the Kp is applied.
In addition, increasing Kp will increase overshoot.



SIM table_1

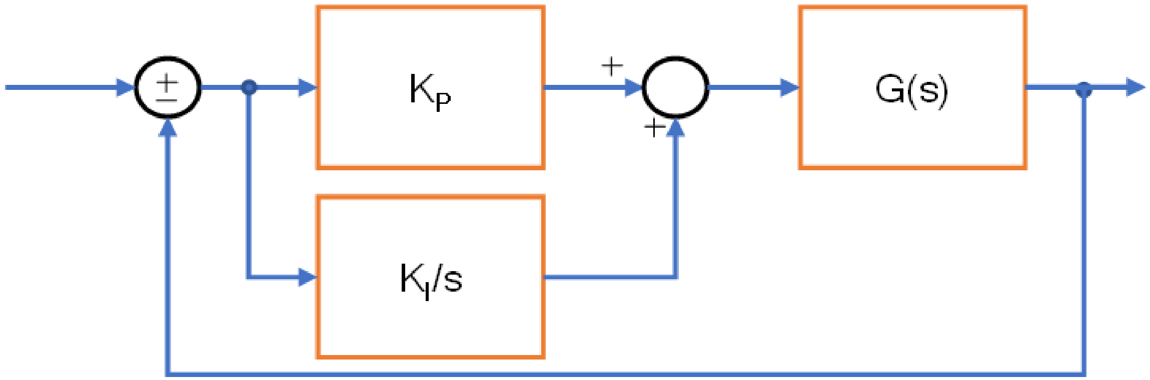
Name	Address	Display format	Monitor/Modify value
"Input":P	%IW0:P	DEC+/-	100
"Kp":P	%IW2:P	DEC+/-	0
"Ki":P	%IW4:P	DEC+/-	5
"Reset":P	%I10.0:P	Bool	FALSE
"Output"	%QW0	DEC+/-	100
"iGS"	%MD18	Floating-point nu...	100
"pOut"	%MD14	Floating-point nu...	0
"iOut"	%MD10	Floating-point nu...	100

As the process is second-order, an integral controller will help reduce the steady-state error.

Try increasing Ki, and observe the final value of the output. With “Ki”:P” is set to 5 (real Ki = 0.5 due to 0.1 factor in the ladder diagram), the final output will reach the goal. The output is still underdamped due to 0.5 damping ratio.

Exercise

- Compare the simulation with theory
- Is there chances of instability while increasing Ki?
  - Explain and prove with root-locus method
- Implement and simulate the following system:



$$G(s) = \frac{4}{s^2 + 3s + 4}$$

Lab 09: Multiple PLC connection

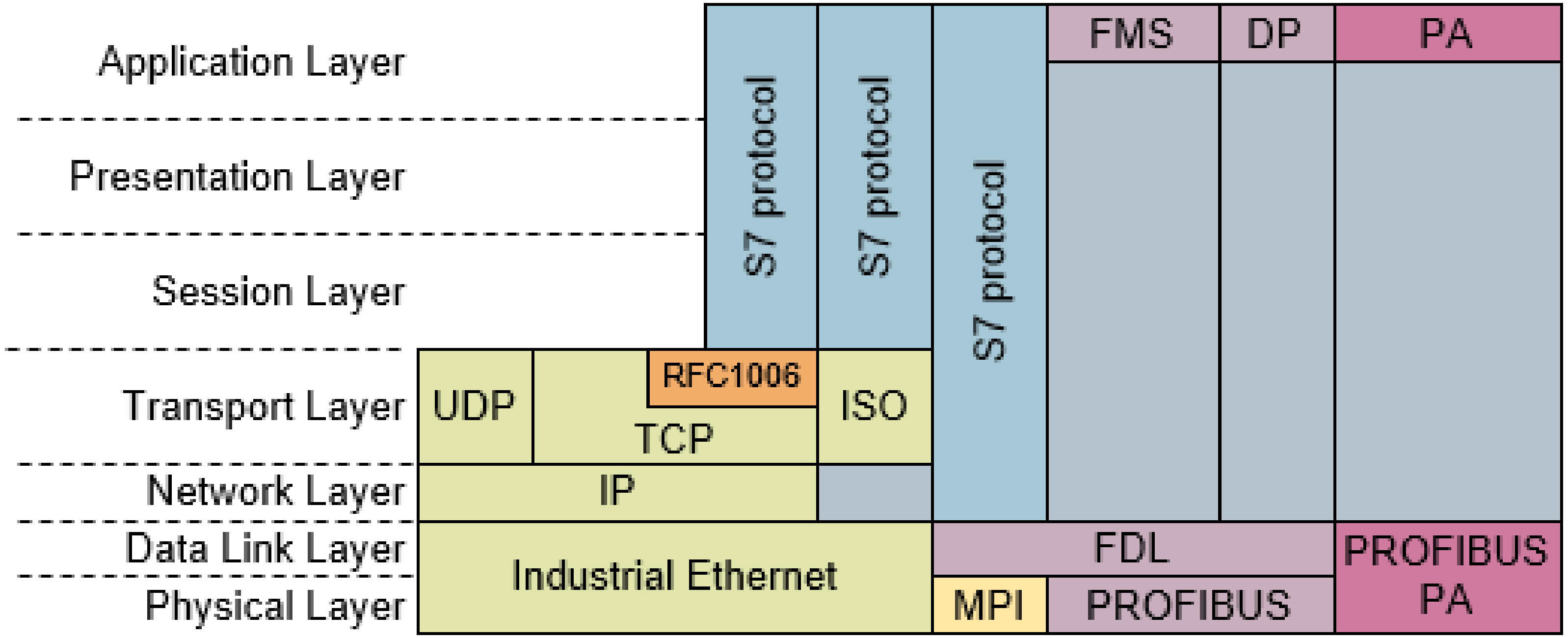
S7 communication services

All SIMATIC S7 CPUs and C7 CPUs have integrated S7 communication services with which the user program can read and write data.

The following functions are available to you for the S7 CPUs and C7 CPUs regardless of the bus system used, so that you can use S7 communication via Industrial Ethernet, PROFIBUS or MPI:

- System function blocks (SFBs): in STEP 7 V5.x for S7-400 CPUs
- Function blocks (FBs): in STEP 7 V5.x for S7-300 CPUs and C7-CPUs
- Instructions: in TIA Portal for S7-300 CPUs, S7-400 CPUs, S7-1200 CPUs and S7-1500 CPUs

Position of the S7 protocol in the ISO-OSI reference model.



Service	Description
PUT / GET	This service is a unidirectional read/write service for transferring small volumes of data to and from a station.
BSEND / BRCV	This service is a bidirectional and block-oriented service for transferring large volumes of data between two stations.
USEND / URCV	This service is a bidirectional and uncoordinated service for transferring small volumes of data between two stations.

Table 1

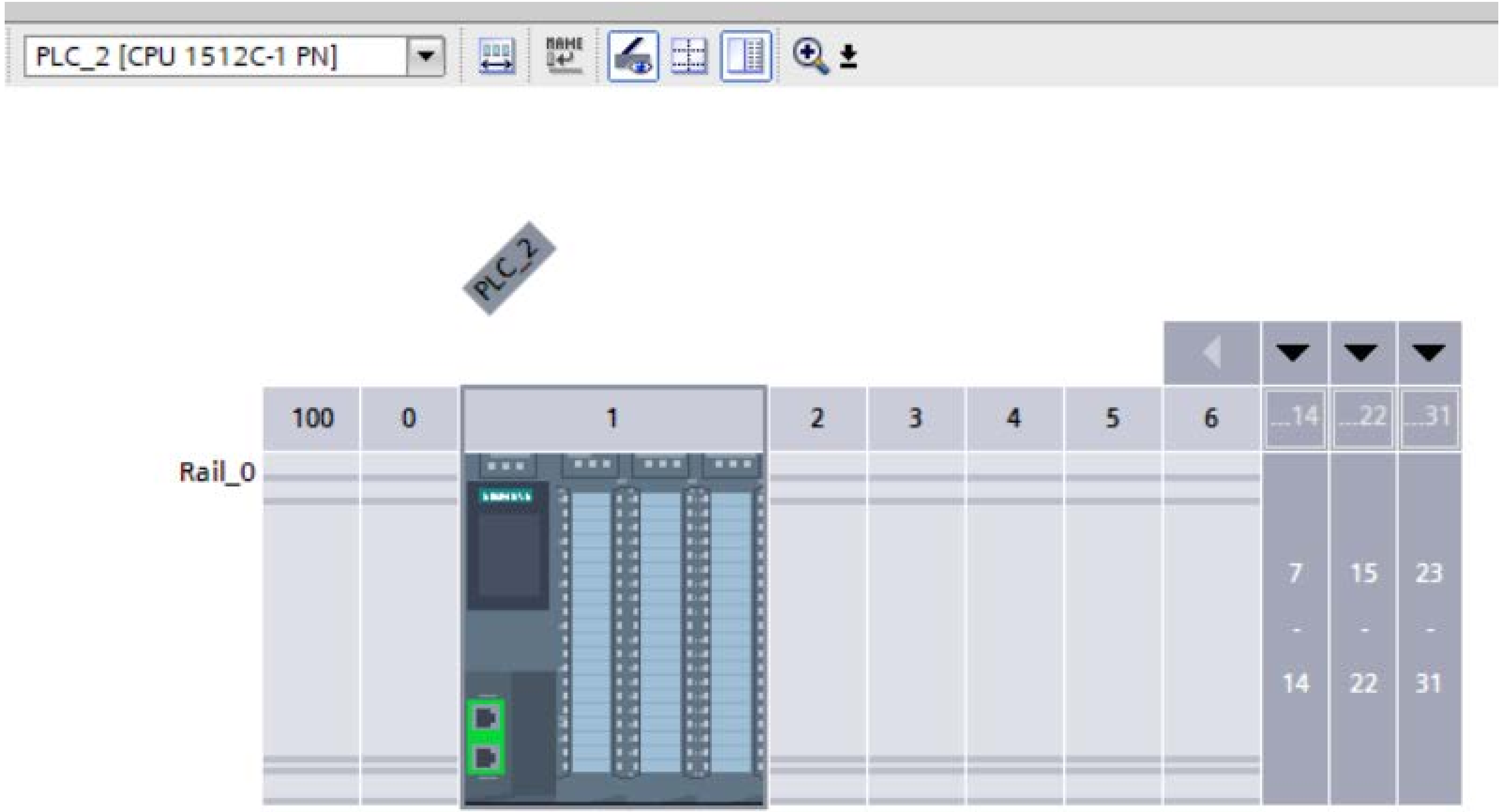
User data size
The S7 protocol permits transfer of data from 1 byte to 64 Kbytes. The maximum data size depends on the service used and the S7 CPU used.

Service	S7-300 CPU	S7-400 CPU	S7-1200 CPU	S7-1500 CPU
PUT / GET	160 bytes	400 bytes	160 bytes	880 bytes
BSEND / BRCV	32768 bytes / 65534 bytes	65534 bytes	-	▪ 65534 bytes with standard access ▪ 65535 bytes with optimized access
USEND / URCV	160 bytes	440 bytes	-	920 bytes

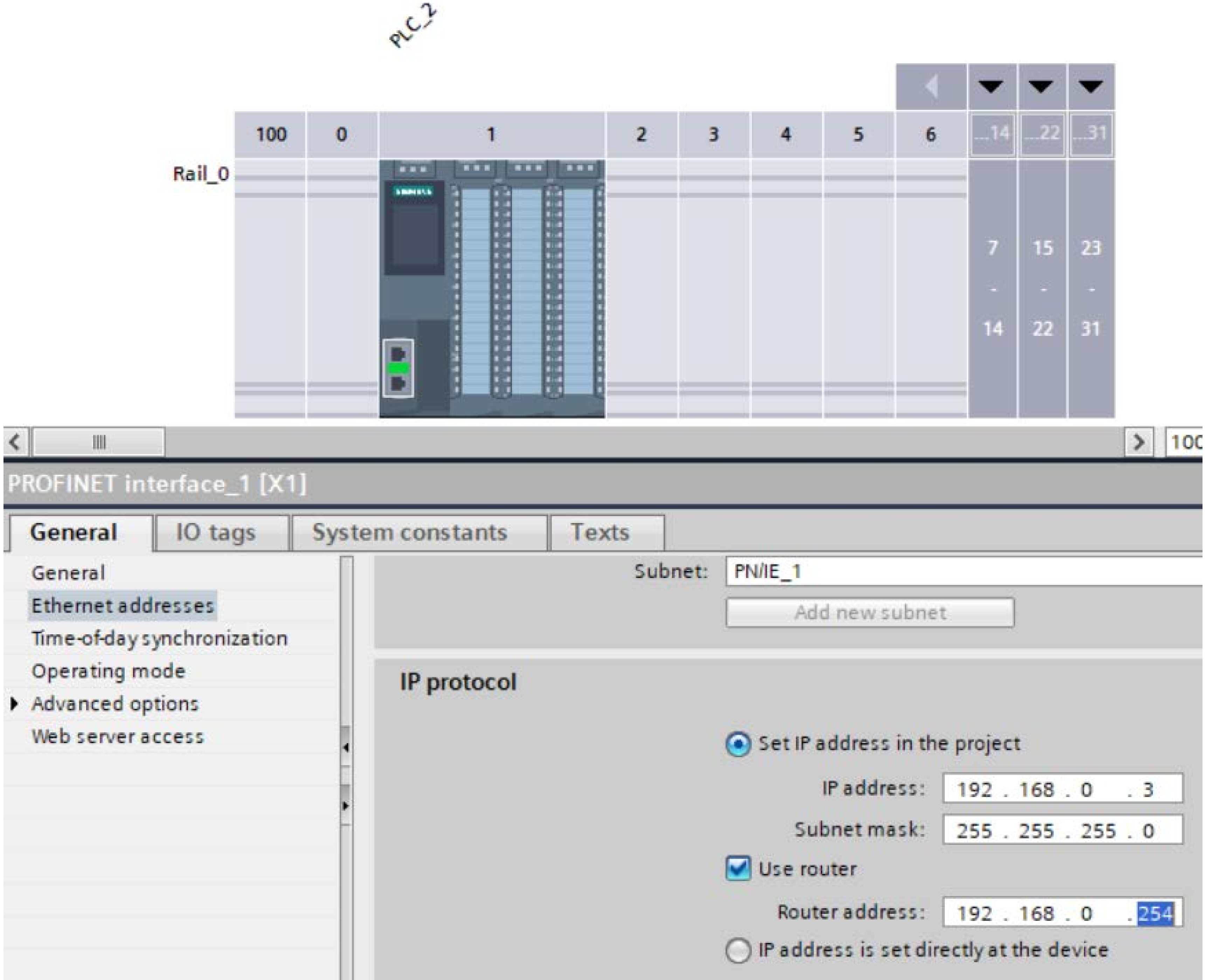
Properties	PUT / GET	BSEND / BRCV	USEND / URCV
Memory areas	M, D, E, A, T, Z	M, D, E, A, T, Z	M, D, E, A, T, Z
Data consistency	▪ 8 to 32 bytes ▪ 32 bytes to total length[1)2)]	Total length per job[2)]	Total length per job[2)]
Communication principle	Client / Server	Client / Client	Client / Client
Maximum number of connections	See CPU specification	See CPU specification	See CPU specification
Functions	▪ FB15 / SFB15 "PUT" ▪ FB14 / SFB14 "GET"	▪ FB12 / SFB12 "BSEND" ▪ FB13 / SFB13 "BRCV"	▪ FB8 / SFB8 "USEND" ▪ FB9 / SFB9 "URCV"

Revisit the project file of the previous lab (Lab 8)

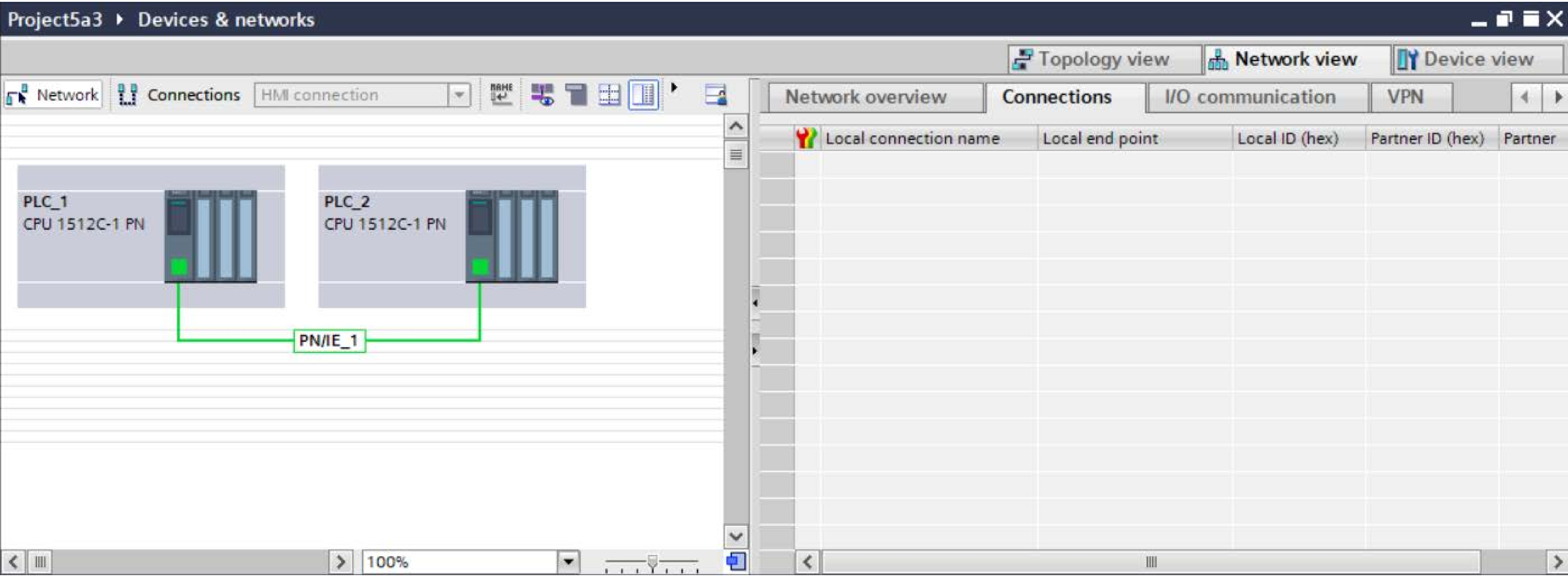
- Add a new PLC to the project



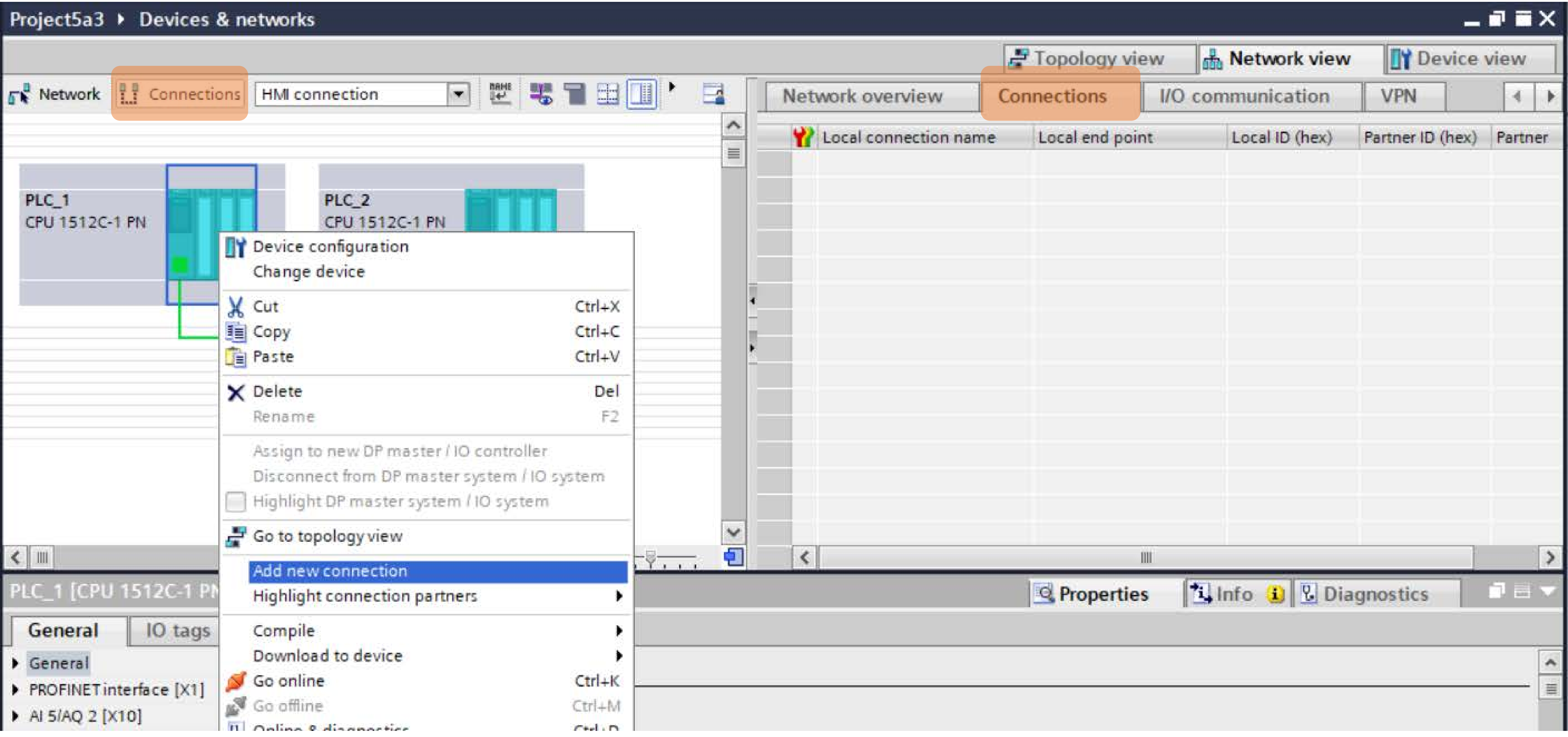
Config the IP and subnet

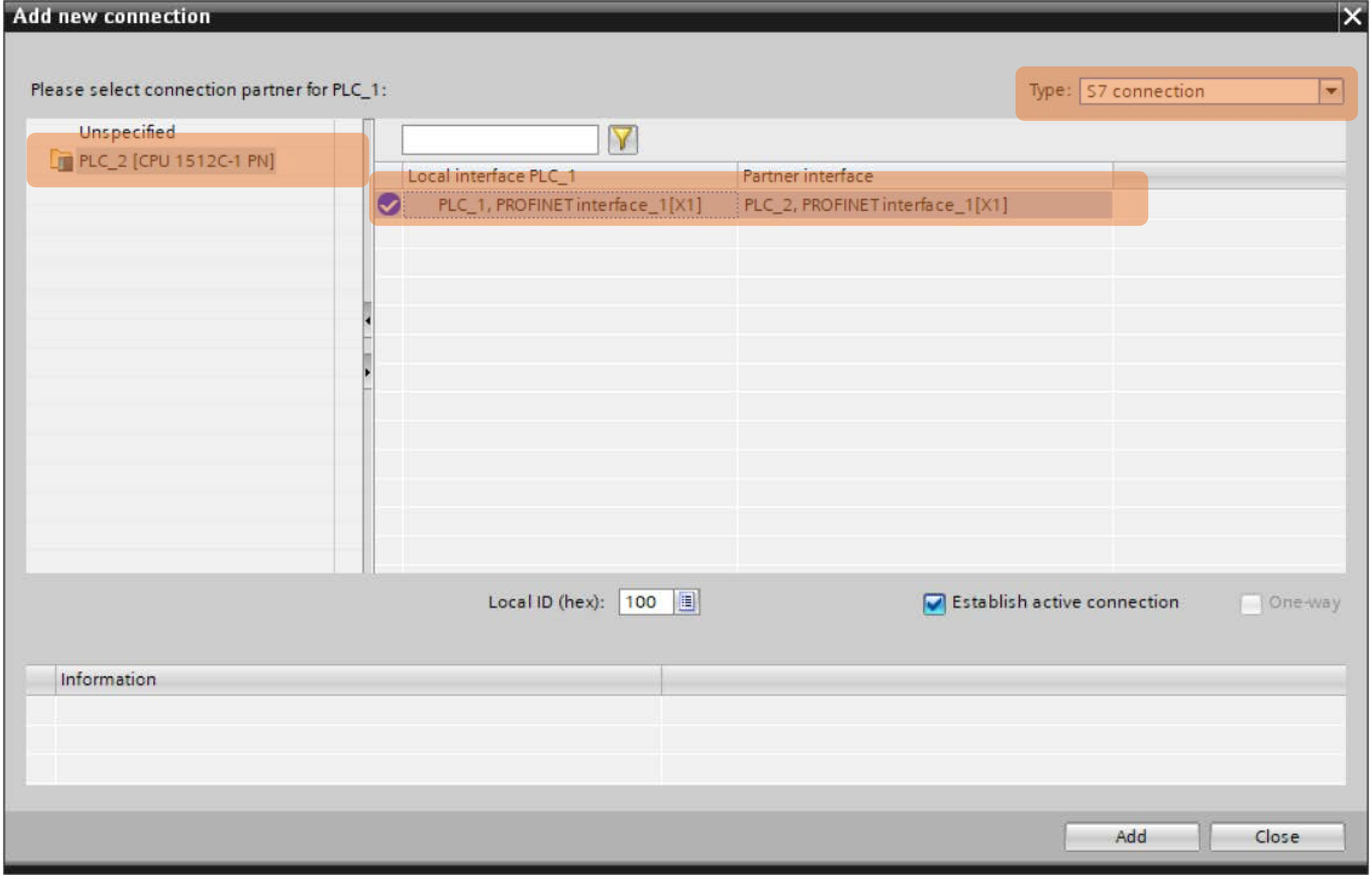


Both PLCs are belong to the subnet but there is no connection.

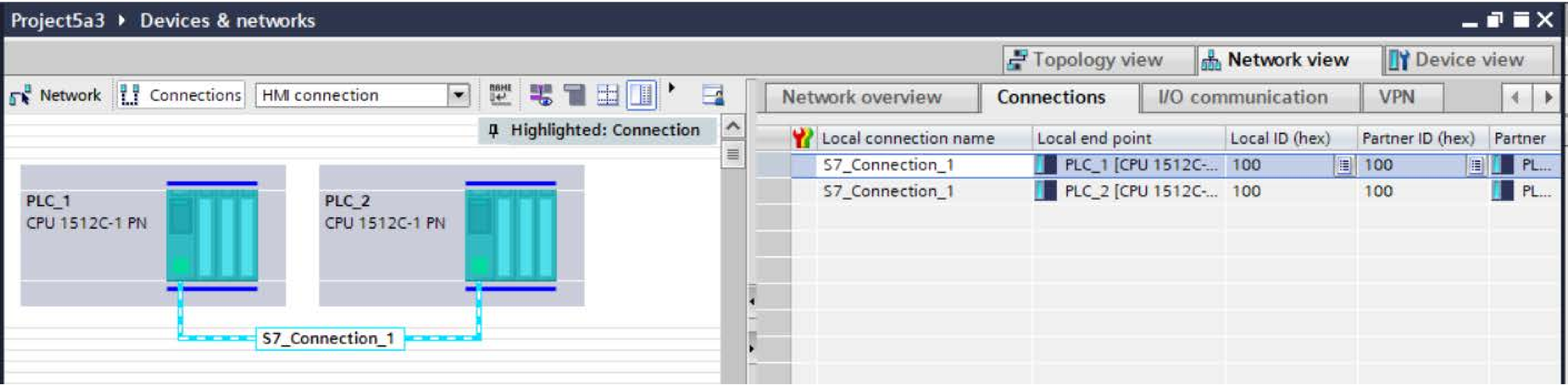


A new connection must be added.

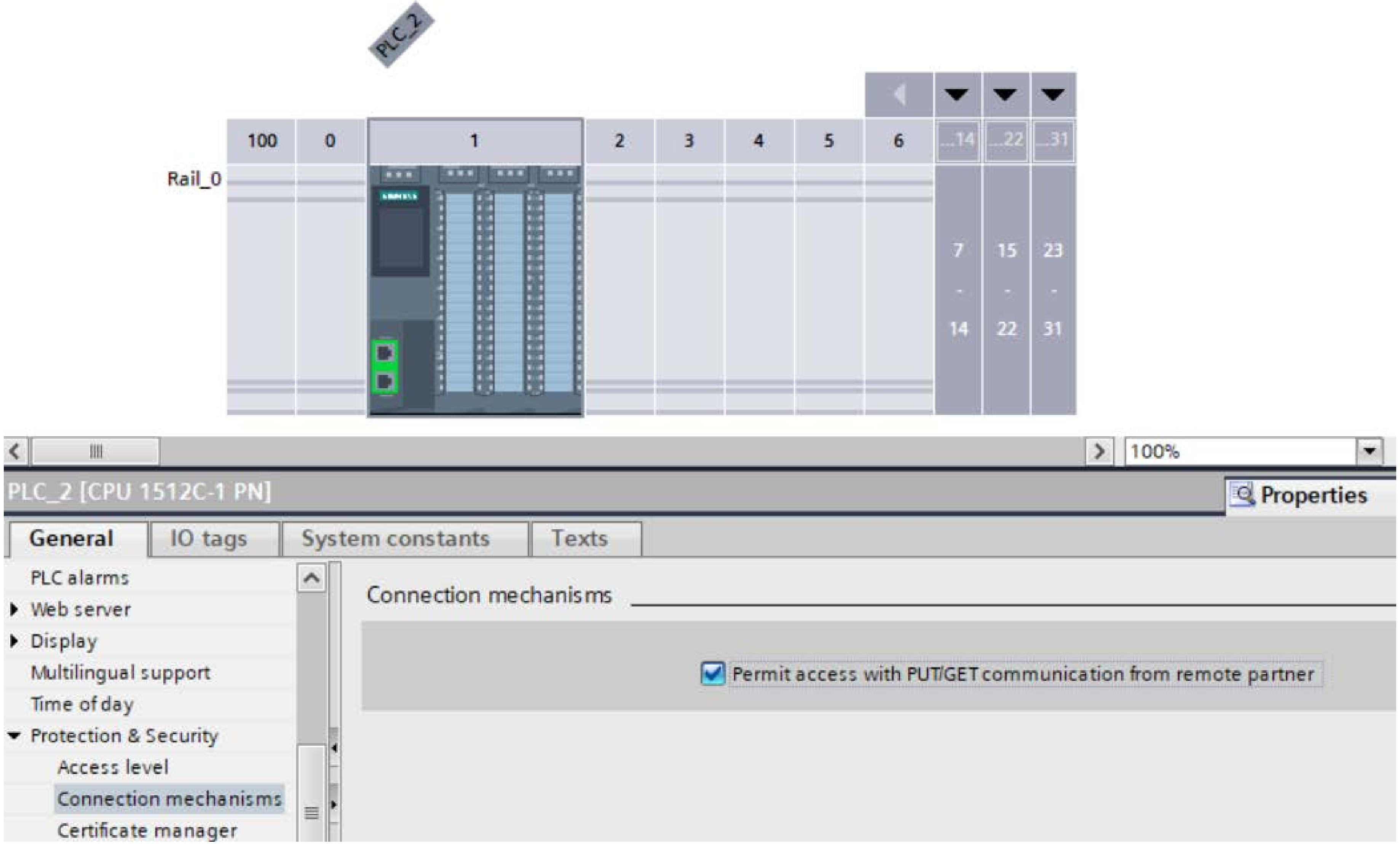



Add new connection
Please select connection partner for PLC_1:
Type: S7 connection
Unspecified
PLC_2 [CPU 1512C-1 PN]
Local interface PLC_1
Partner interface
PLC_1, PROFINET interface_1[X1]
PLC_2, PROFINET interface_1[X1]
Local ID (hex): 100
Establish active connection
One-way
Information
Add
Close

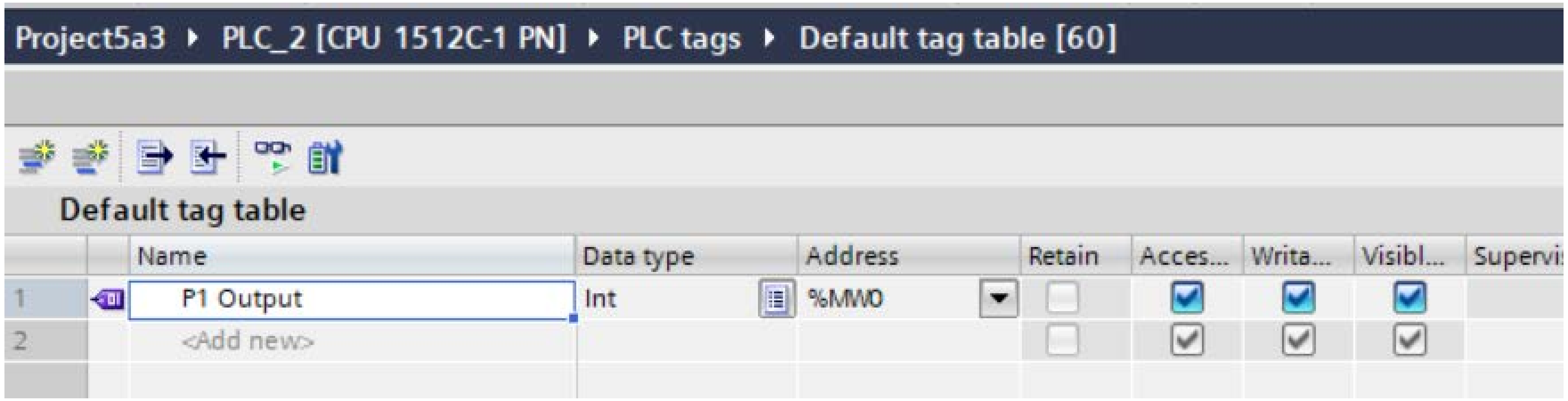
S7 connection between two PLCs are established.



Communication via PUT

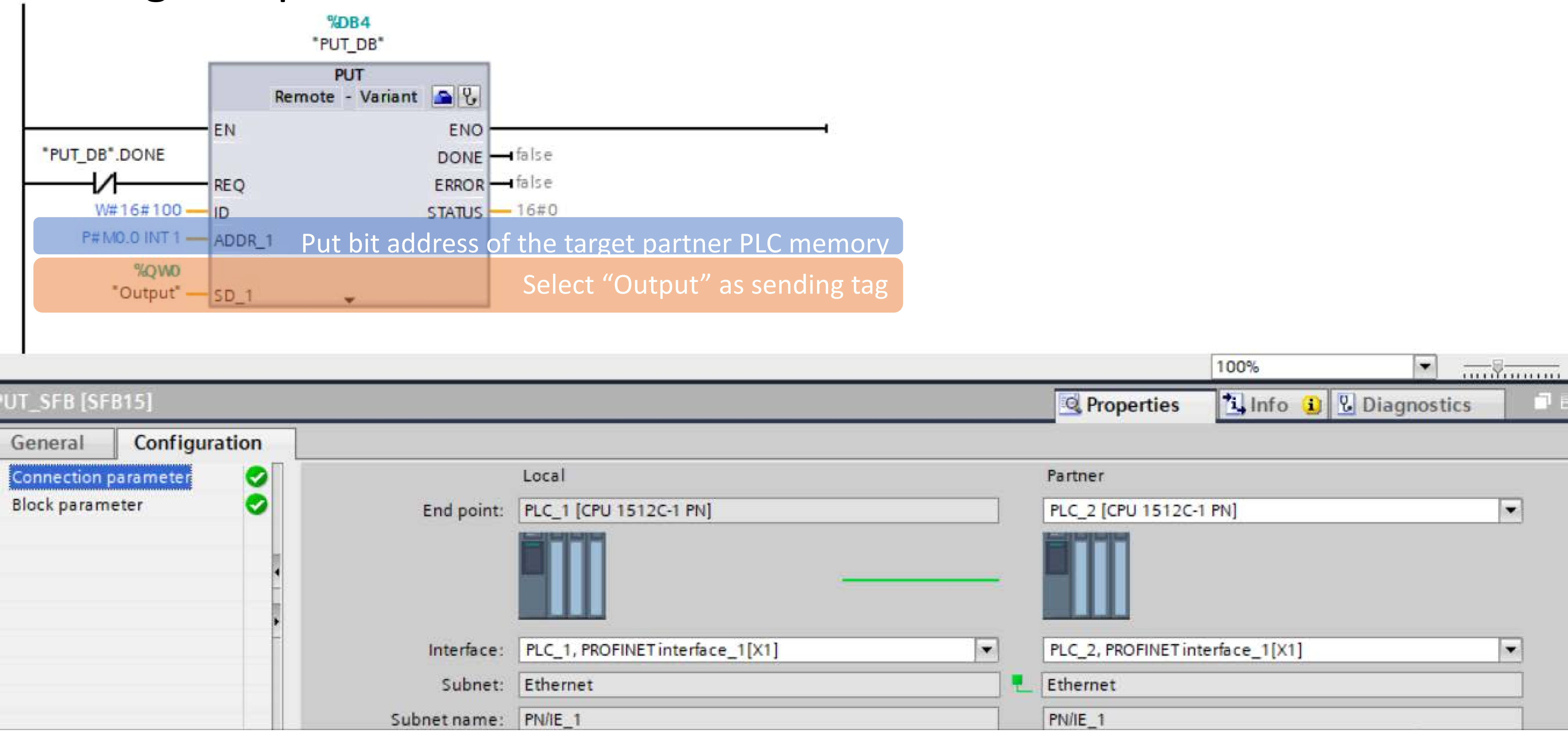


The access must be permitted.

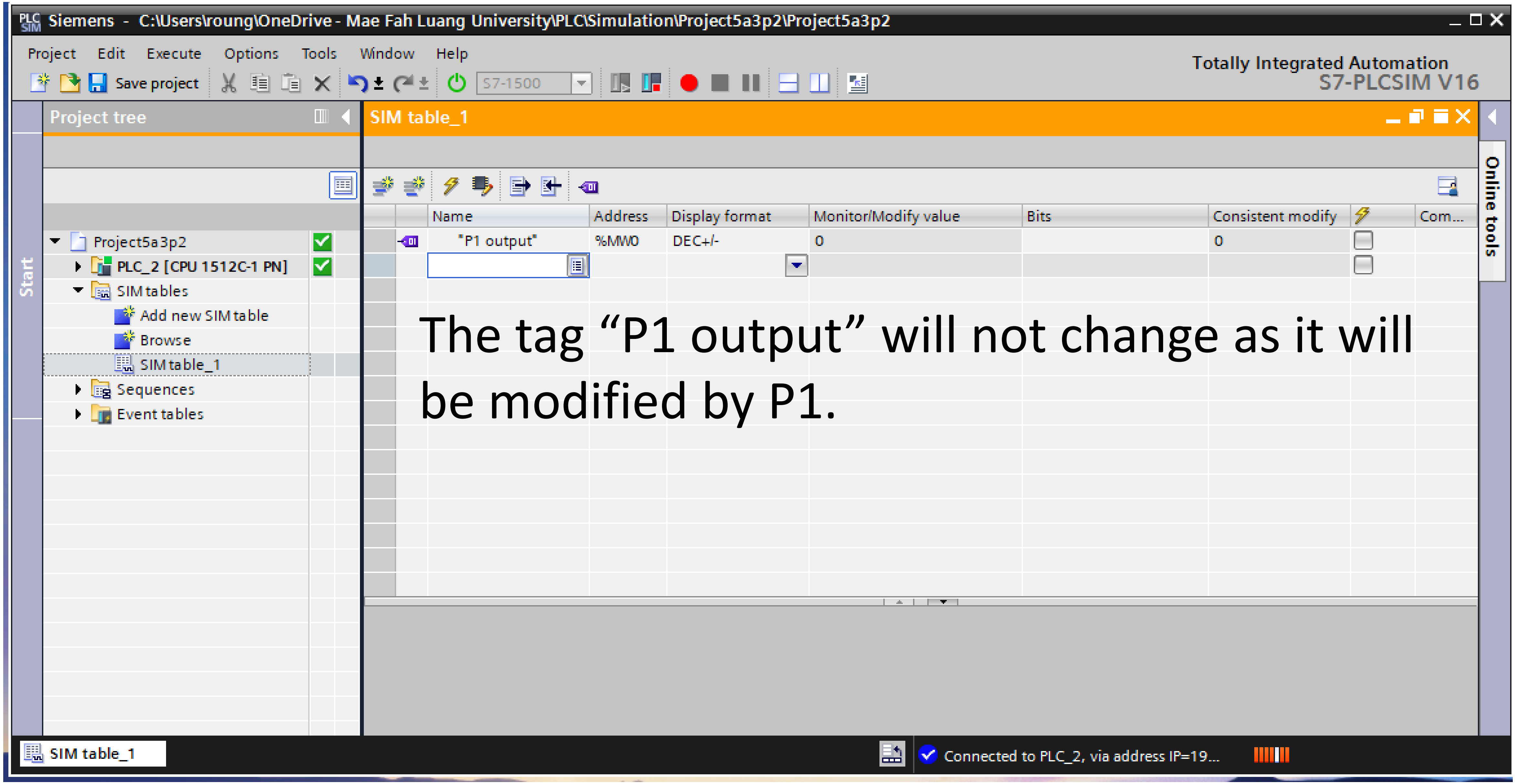


Create tag in PLC2 to get info from PLC1.

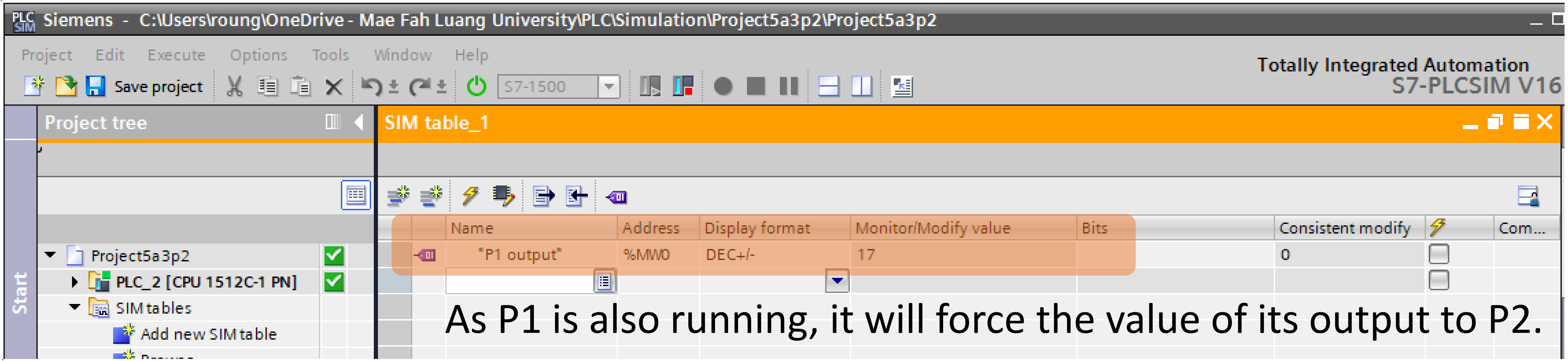
Config the connection parameter of the PUT block by matching the targeted partner PLC



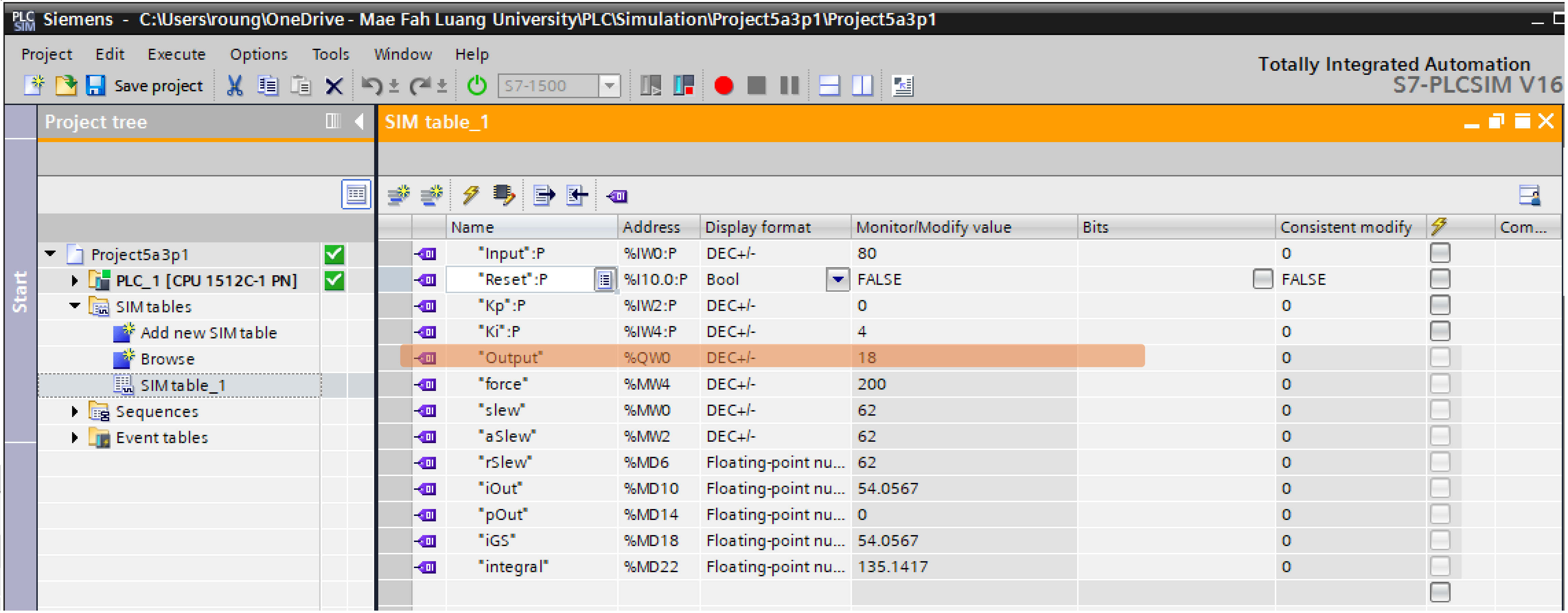
Simulating the PLC2



The tag “P1 output” will not change as it will be modified by P1.



As P1 is also running, it will force the value of its output to P2.



Exercise

- Remove the PUT block from PLC1
- Use GET block at PLC2 to obtain PLC1’s output instead.

Lab 10: Synchronization of multiple PLC

Revisit the PLCs in Lab-9

- PLC2 will reset the output of the plant reported by PLC1 if it is exceeding a certain level set in PLC2
- Synchronization
  - PLC1 put the value of its output to PLC2
  - PLC2 put the value of its resetting command to PLC1

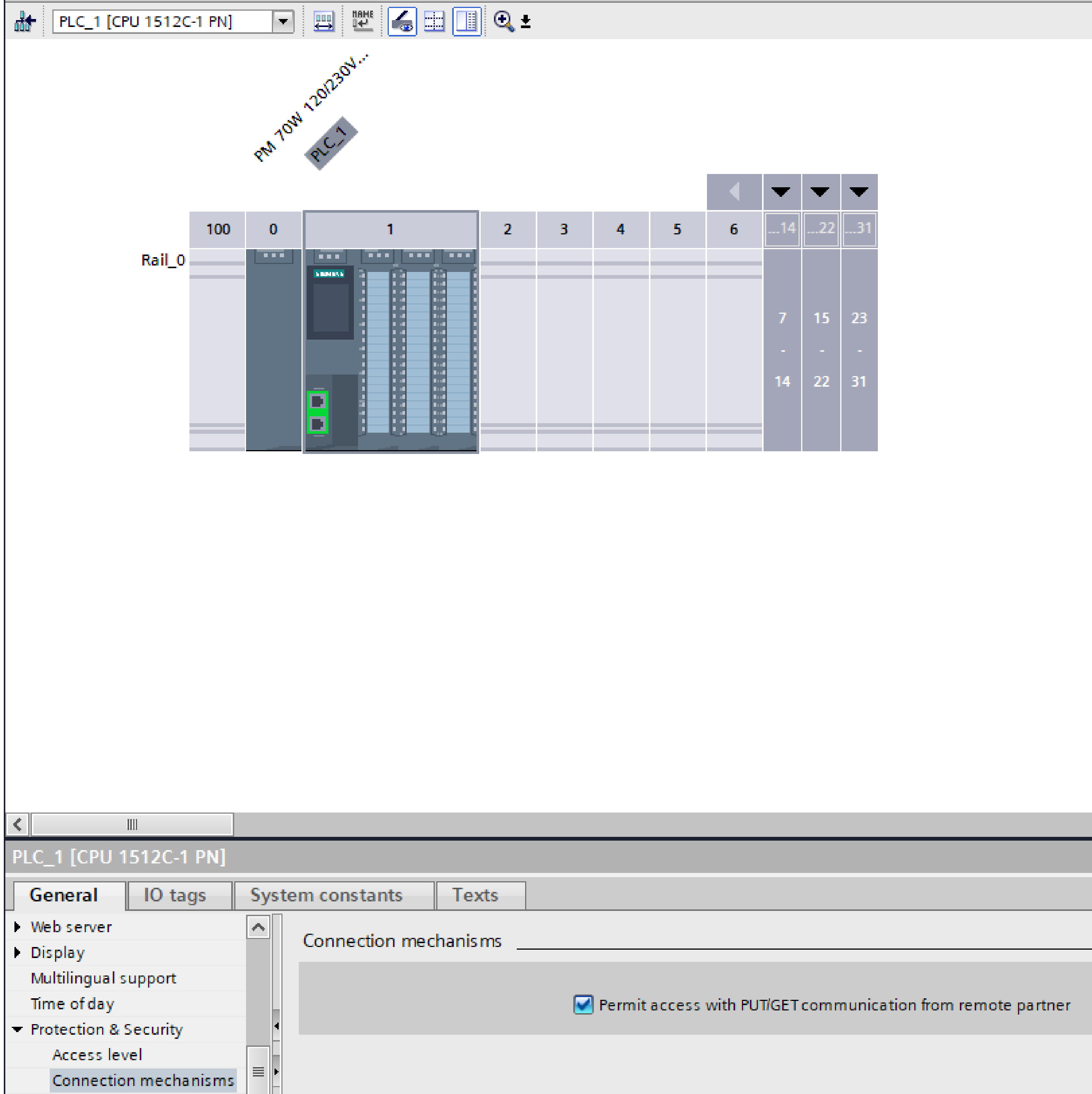
PLC1 tags

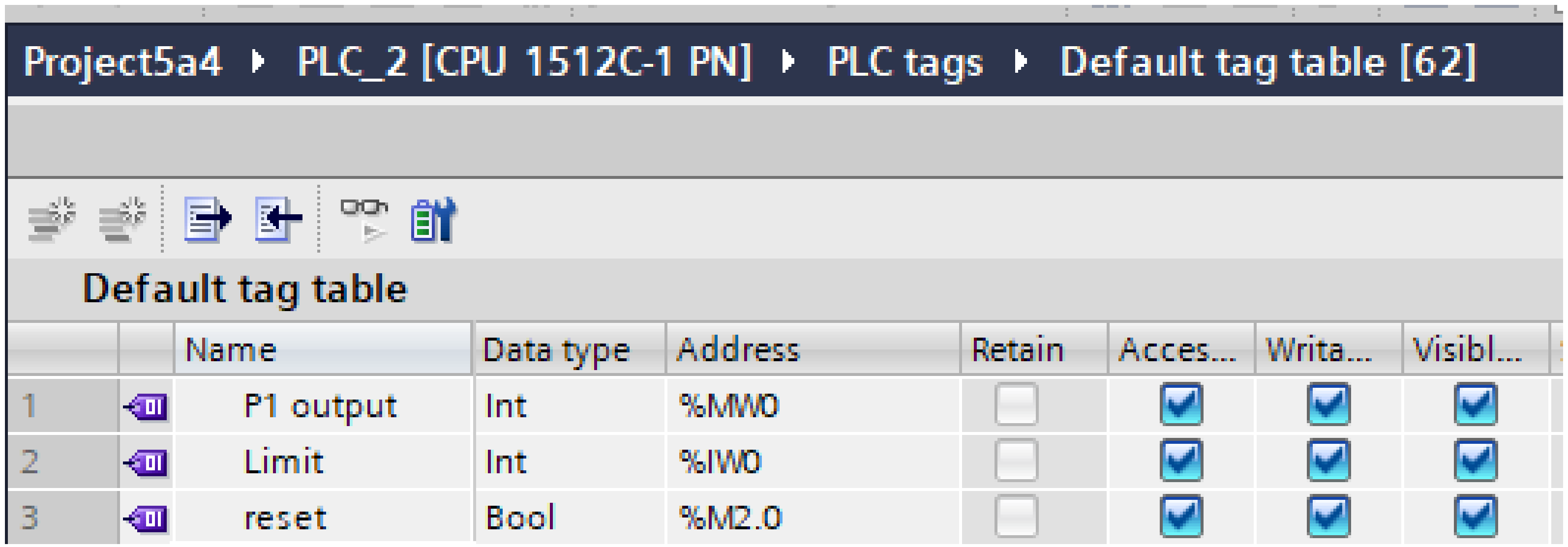
Project5a4 ▸ PLC_1 [CPU 1512C-1 PN] ▸ PLC tags ▸ Default tag table [73]

Default tag table

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Supervis...	Comment
1	slew	Int	%MW0	☐	☑	☑	☑		Input - Output
2	aSlew	Int	%MW2	☐	☑	☑	☑		Absolute value of slew
3	force	Int	%MW4	☐	☑	☑	☑		Force the RampFunction to go up or down
4	Input	Int	%IW0	☐	☑	☑	☑		Desired Response
5	Output	Int	%QW0	☐	☑	☑	☑		Actual Response
6	rSlew	Real	%MD6	☐	☑	☑	☑		Real absolute value of slew for setting Ram
7	Kp	Int	%IW2	☐	☑	☑	☑		Propotional Gain
8	Ki	Int	%IW4	☐	☑	☑	☑		Integral Gain
9	iOut	Real	%MD10	☐	☑	☑	☑		Integral Output
10	pOut	Real	%MD14	☐	☑	☑	☑		Propotional Output
11	iGS	Real	%MD18	☐	☑	☑	☑		Input to the G(s)
12	integral	Real	%MD22	☐	☑	☑	☑		Output of RampFunction as an Integrator
13	reset	Bool	%M26.0	☐	☑	☑	☑		Reset command issued by PLC2

Permit access with PUT/GET on PLC1



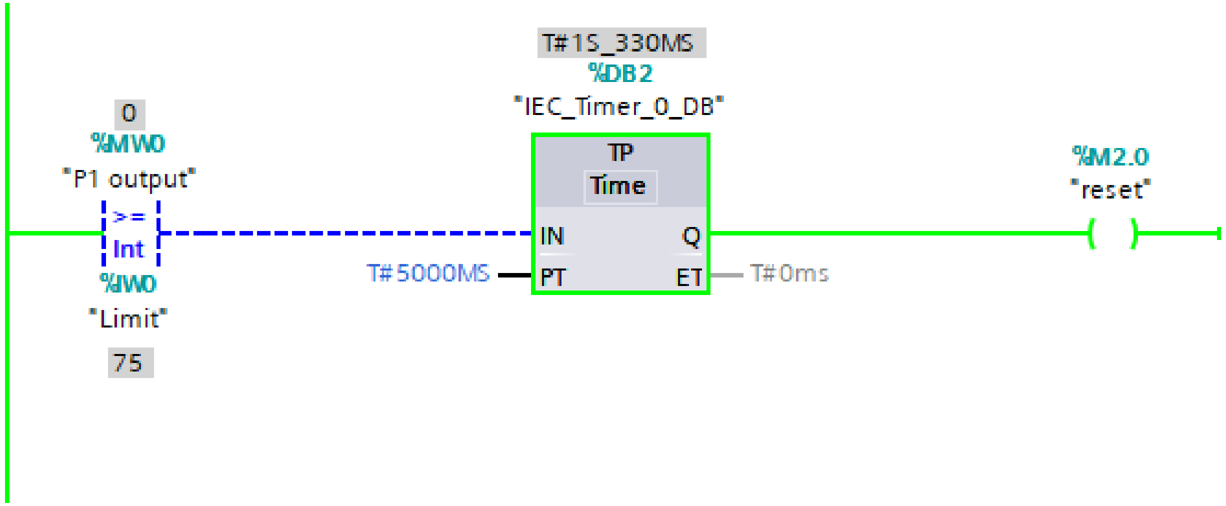


Project5a4 ▸ PLC_2 [CPU 1512C-1 PN] ▸ PLC tags ▸ Default tag table [62]

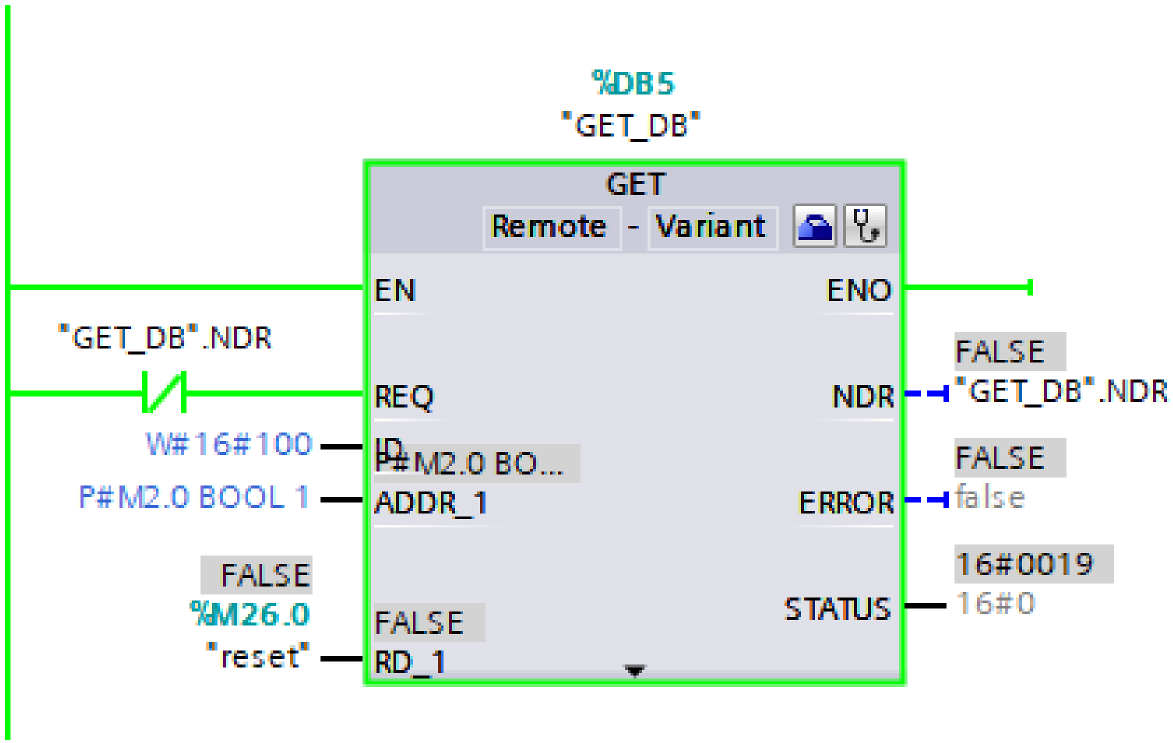
Default tag table

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...
1	P1 output	Int	%MW0	☐	☑	☑	☑
2	Limit	Int	%IW0	☐	☑	☑	☑
3	reset	Bool	%M2.0	☐	☑	☑	☑

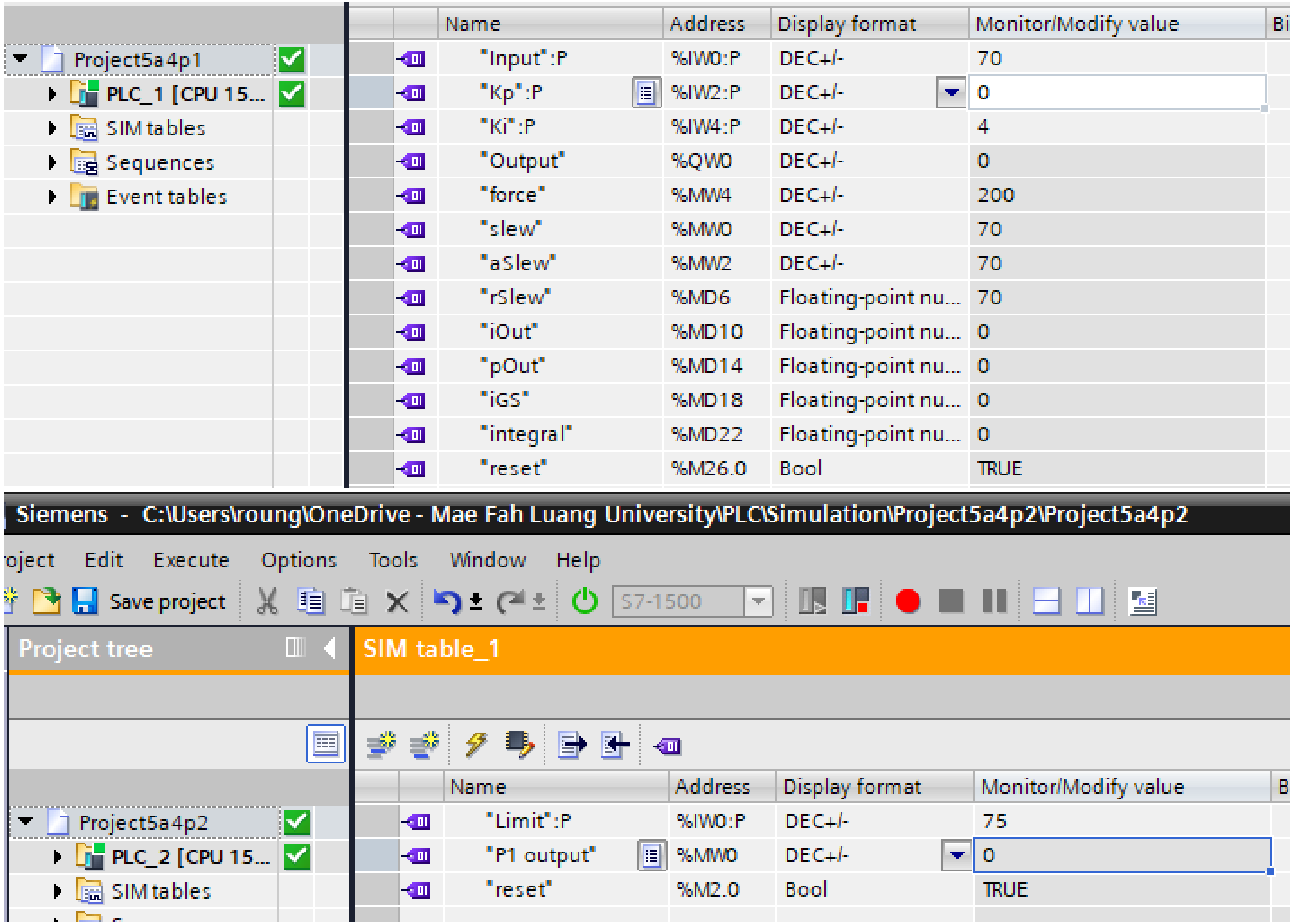
PLC2 tag



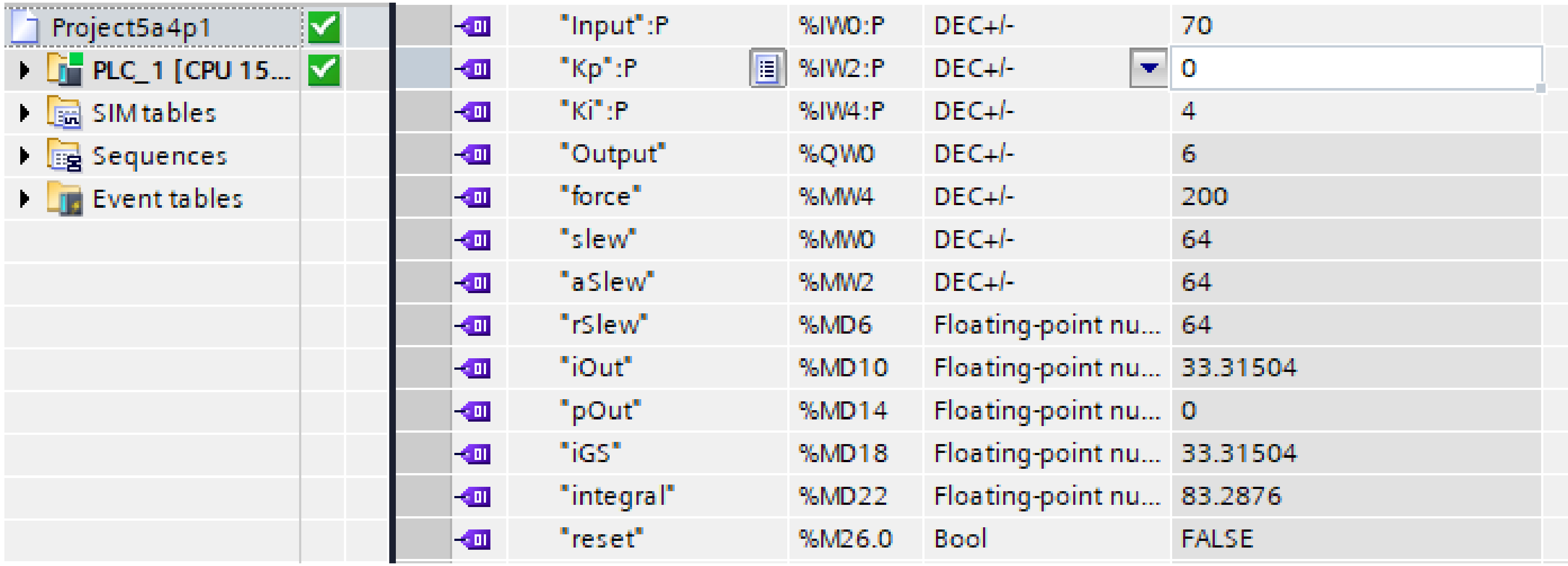
Ladder diagram of PLC2 issuing “reset” logic

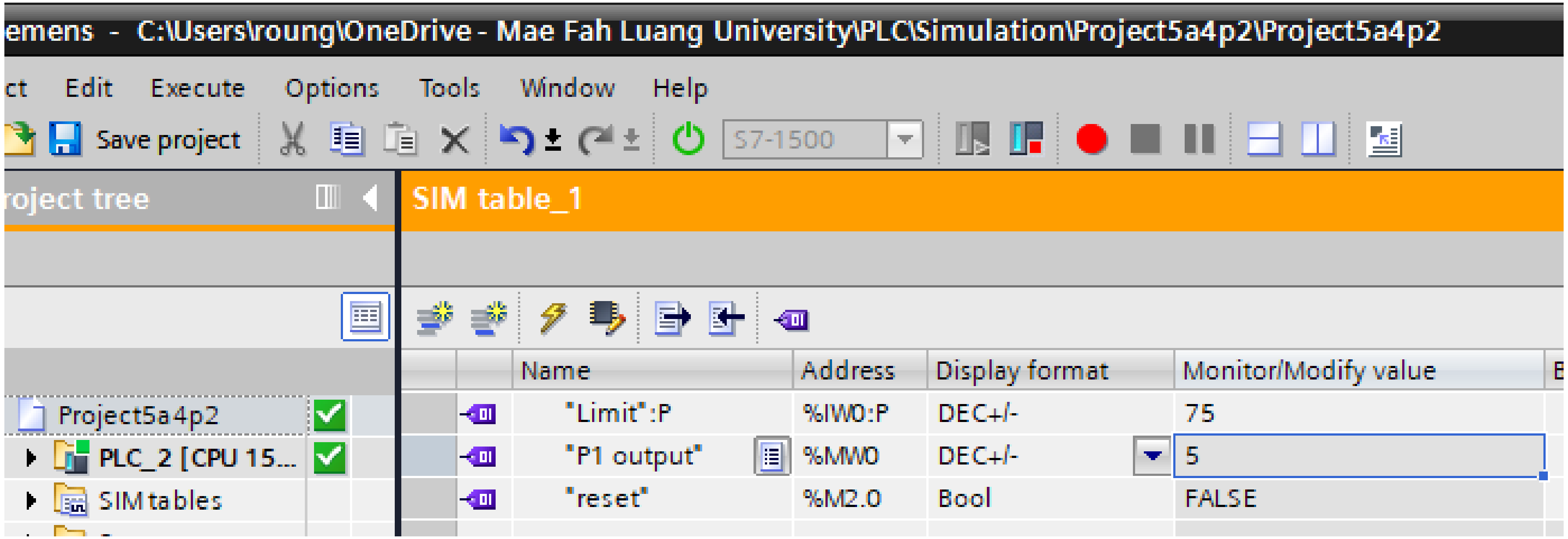


Ladder diagram of PLC1 reading the logic “reset” from M2.0 of PLC2



PLC1 and PLC2 are synchronizing as PLC1 sends its output to PLC2, and read the “reset” logic from PLC2.


Project5a4p1
PLC_1 [CPU 15...
SIM tables
Sequences
Event tables
"Input":P | %IW0:P | DEC+/- | 70
"Kp":P | %IW2:P | DEC+/- | 0
"Ki":P | %IW4:P | DEC+/- | 4
"Output" | %QW0 | DEC+/- | 6
"force" | %MW4 | DEC+/- | 200
"slew" | %MW0 | DEC+/- | 64
"aSlew" | %MW2 | DEC+/- | 64
"rSlew" | %MD6 | Floating-point nu... | 64
"iOut" | %MD10 | Floating-point nu... | 33.31504
"pOut" | %MD14 | Floating-point nu... | 0
"iGS" | %MD18 | Floating-point nu... | 33.31504
"integral" | %MD22 | Floating-point nu... | 83.2876
"reset" | %M26.0 | Bool | FALSE


emens - C:\Users\roung\OneDrive - Mae Fah Luang University\PLC\Simulation\Project5a4p2\Project5a4p2
ct Edit Execute Options Tools Window Help
Save project
S7-1500
roject tree
SIM table_1
Name | Address | Display format | Monitor/Modify value
Project5a4p2
PLC_2 [CPU 15...
SIM tables
"Limit":P | %IW0:P | DEC+/- | 75
"P1 output" | %MW0 | DEC+/- | 5
"reset" | %M2.0 | Bool | FALSE

Exercise

- Add a PLC to do an additional filling as same as PLC1 do
- Make PLC2 issuing a reset signal to the PLC as same as with PLC1